

DOCUPAK

VOLUME 1

**APPLE PUGET SOUND PROGRAM LIBRARY EXCHANGE
A Washington State Non-Profit Corporation**

**THE
WOZPAK][**

and Other Assorted Goodies

**A Collection of APPLE II Documentation
Furnished Through Courtesy of
APPLE COMPUTER, INC.
Assembled and Distributed by
APPLE PUGET SOUND PROGRAM LIBRARY EXCHANGE**

\$22.50

6708 39th Avenue SW

Seattle, Washington 98136

(206) 932-6588

WOZPAK II

INTRODUCTION

At last the mythical WOZPAK][is here.

For those who have waited a long, long time – it finally does exist. A note of appreciation to the people who contributed to this project:

First to **KEN SMITH** who worked long hours editing both the text and programs, and with his Diablo Printer produced the final copy of this book;

To **BOB CLARDY** and **RALPH THIERS** for editorial assistance;

To **STU BRUMMETT**, for entering into the word processor many pages of text;

To **DICK SEDOWICK**, and **DAVE McINTOSH** for the "Sweet 16" compendium.

Many thanks to **STEVE WOZNIAK** ("WOZ") and **APPLE COMPUTER** for all the other untitled articles in this volume.

Finally to the publishers of *MICRO* and *DR. DOBBS JOURNAL* for permission to reprint their articles.

The optional **WOZPAK** Cassette contains the programs in this volume. The first program on the cassette is an index giving the address instructions for all the machine language programs. All programs are recorded twice on the tape.

The second set are recorded with a different recorder and a short voice input separates the two recordings.

Dick Hubert – November 15, 1979

APPLE PUGETSOUND PROGRAM LIBRARY EXCHANGE

6708 39TH Avenue Southwest
Seattle, Washington 98136
(206) 932-6588

presents

**WOZPAK II
(DOCUPAK Volume 1)**

Table of Contents

TED II+	3
Line Number Cross Reference	9
Symbol Cross Reference	13
Renumber/Append	23
Pack & Load.	37
Machine Code Relocation Notes.	45
Tape Verify	57
High Resolution Graphics Subroutines.	61
Shape Generator	79
SWEET 16 Introduction	85
SWEET 16 The 6502 Dream Machine	89
Lazarus	99
Floating Point Routines	107
Auto Repeat for MONITOR Commands	117
Integer Basic Subroutine Calls	119
STAR TREK	121
HI-RES Color Modification	129
Color Killer Modification	135

BLANK PAGE

TED II+ EDITOR/ASSEMBLER

a Beginning Review
of a Beginner's Assembler
by Ken Smith (a Beginner)

BLANK PAGE

WOZPAK II

From the stories I've heard, this is probably the 697th version of the TED II ASSEMBLER. Hopefully it is as good or better than the previous versions. This review of TED II+ is the result of a great deal of trial and error, and a little bit of luck!

The version of TED II ASSEMBLER that I have used was designed to be used on a 48K APPLE][with DOS 3.2. However, it now works on 32K or 36K APPLE][s, and it can be used on systems without disk. Source files can be saved to tape from within TED II+ itself (see LOAD & SAVE commands).

There are two modes of operation used in TED II+, EXEC and TED (for EXECutive and Text Editor). The prompt character for EXEC is '%'. ':' is the prompt character for TED mode. The EXEC mode provides the interface for all disk functions. The following is a summary of EXEC commands.

- C :DISK CATALOG
Displays disk catalog. Source files are saved by TED II+ as <NAME>.S" The ".S" subscript allows TED II+ to identify source files. Hit any key to return to EXEC menu.
- D :DIRECT COMMAND TO DISK
Allows direct access to DOS without leaving program. This can be used to rename or delete files, load printer routines, or specify a drive# for subsequent L & S commands. For example, "CATALOG,D2" will set drive #2 as the specified disk drive. Add the ".S" subscript for commands, if applicable.
- L :LOAD FILE FROM DISK
Loads existing source file from disk. <NAME> should not include ".S", as TED II+ will provide it for a Load.
- S :SAVE FILE TO DISK
Source file will be saved to disk with <NAME> provided by user. TED II+ will provide ".S" subscript.
- A :APPEND FILES (CREATES '-A' FILE)
Loads two existing disk files, puts them together, and saves the new file to disk in one operation. User is asked for 'ORIGINAL FILE' and 'FILE

TO APPEND'. The new file created will be '<ORIGINAL FILE NAME>-A', and the other files will be unchanged. The new file will also be in memory.

T :ENTER TED II+ EDITOR/ASSEMBLER
This shifts operation from EXEC mode to TED mode. TED mode commands are covered in detail in the next section.

Q :QUIT TED II+
This exits the program when 'Q' is entered in EXEC mode. A 'Q' entered while in TED mode will return control to EXEC mode. Should you accidentally exit the program, 'RUN' [RETURN] will restart without losing the source file in memory.

Should you change your mind, the D,L,S, and A commands can be aborted with a carriage return.

After entering TED mode, the following commands apply. These are direct commands. Note the upper and lower case letters of the commands. Those letters in upper case are required; the lower case letters optional. An example of command syntax is included, where needed.

Add

Puts user into 'Add Lines' mode. If a source file is in memory, 'Add' will start with the next available line number. If not, it will start with 'Line 1'. Add is terminated with a CONTROL-D as the first character on the line.

Insert I<line#>

Operates like 'Add', except lines are inserted into source starting at <line#>. Also terminated with a CONTROL-D. Lines after the insertion line are automatically renumbered. Lines may be inserted before any existing line.

The following commands allow the user to specify the range or ranges of lines upon which the command will act. If no range is specified, the entire file will be acted upon. A range may be a single <line#> or '<line#>,<line#>'. In addition, multiple ranges may be specified as '<range>/<range>' using "/" as a delimiter.

WOZPAK II

Delete D<line#> or D<range>

Deletes line or lines specified. The remaining lines are renumbered by TED II+. Because of this, be sure to specify the ranges from highest to lowest. Otherwise, the wrong lines will be deleted! 'Delete' requires a <line#> or a <range> as part of the command.

Find F<range>"<string>"

Locates and prints all lines in which <string> occurs. If a <range> is specified, only that range will be searched.

Change C<range>"<s1>"<s2>"

Provides selective substitution of <s2> for <s1>. When Change command is entered, user is asked if "ALL OR SOME (A/S)" occurrences are to be changed. If 'All' is chosen, <s2> will be substituted for <s1> at every occurrence of <s1> within the range, if specified. I have found it better to use the 'Some' choice since it allows the user to monitor the changes. Each line is displayed as it will appear with the change made. If the user hits the 'ESC' key, the change will NOT be made in that line. Any other key will allow the change to be made. Be sure to enter the " as shown in the example.

COPY COPY<L#1>,<L#2> TO <L#3>

Copies the range of lines specified by <L#1>,<L#2> to BEFORE <L#3>. After the move is made, the original lines (while possibly renumbered) are not deleted. The user must do this seperatly.

Edit E<range> or E<line#>

This is where we fix the mistakes! Each line to be edited is printed on the screen with the cursor at the beginning. Editing of lines is accomplished with the following controls:

- [RETURN] Accept the line as it appears on the screen.
- CTRL-D Deletes the character under the cursor.
- CTRL-I Inserts character(s) into the line at the cursor position. Terminated with a space, a forward arrow or a [RETURN].
- CTRL-F Moves the cursor to character entered after 'CTRL-F'. (find)
- CTRL-Q accept line from beginning to cursor position.
- CTRL-R Restart editing of current line with previous changes deleted.
- CTRL-X Exit editing mode.

␣

The forward and back arrows can be used to position the cursor on the line being edited. CTR-U and CTRL-H can be used for cursor moves, also.

List L<line#> or L<range>

Lists the line(s) specified. If no <range> is specified, the entire file will be listed. The listing can be stopped at any time by hitting the space bar. The list will advance one line each time that the space bar is hit. Any other key will continue the listing. CTRL-C will abort.

General TED II+ Housekeeping commands:

HImem: HI:<decimal#>

Sets highest memory limit for source and symbol tables.

LOmem: LO:<decimal#>

Sets lowest memory limit for source table and performs a NEW. Normally set to 7424. If LOMEM: is set below this point, TED II+ may overwrite itself.

NEW

Deletes existing source file. NEW does not reset memory limits.

IN#

Same as Basic IN#, but pretty much useless. A CTRL-C exit from listing or an 'END ASSEMBLY' resets to IN#0.

PR#

Same as Basic PR#. Quitting to EXEC resets to PR#0. This command is used for hard copy listing of the source file. PR# is also a 'PSEUDO-OP', as discussed in that section, and is used as such for hard copy assembly listings.

LEN

Provides display of size of source table and remaining memory available.

TABS TABS<col>/<col>/<col>" "

This seems to be the one command most likely to bomb the program if incorrectly entered. TABS is used to set tab stops for listing the source file. <col> is the decimal column of the stop. The quotes are used to define the tab character, (a space).

LOAD

This load command is available for non-disk systems. It loads a source file from Tape.

WOZPAK II

SAVE

Saves source file to Tape.

Note for non-disk users: TED II+ can be saved to tape as a machine-language program. After loading and running this Integer Basic version, exit the program and go into Monitor. Save it to tape with "800.1CFFW". Tape files can be saved and transferred to disk later. The Cold Start entry is \$800; Warm Start is \$803. (A warm start does not reset source pointers, thus source is not erased on reentry)

An instruction line entered into the source file consists of four parts:

LABEL....OPERATOR....OPERAND....COMMENT

If the label field is not used on a line, a space must be entered in this field. The remaining fields are separated from each other by a space each. An asterisk '*' entered into the label position on a line is used for Title or Comment lines. On single byte op-codes (i.e. ROL,ROR,etc), a ";" can be used in the operand field so that comment indexing is maintained. No space is needed between ';' and <comment>.

PSEUDO-OPS

TED II+ supports all 6502 Mnemonics, Sweet-16 operators, and it's own Pseudo-Operator set:

....ORG....\$<addr>

Sets address where final program will run. Object code may be actually written elsewhere, but is written to run at the specified address.

....OBJ....\$<addr>

Specifies starting address where object code is actually written. The default address of OBJ and ORG is \$7000 on a 48K APPLE][. (\$5000 on a < 48K APPLE).

<label>....EQU....<expr>

Assigns value of <expr> to <label>. <expr> is used for a decimal value <256. '\$<expr>' is used for Hex addresses or constants. Zero-page addresses may be expressed in decimal, if desired.

....DS....<expr>

Reserves <expr> number of bytes at the location of DS in the program. <expr> may be decimal or Hex (\$<expr>).

....DA....<label>

Stuffs address of <label> into memory,

low byte first, then high byte.

....DW....<expr>

Stuffs hex equivalent of <expr> into memory. <expr> may be any legal type of expression.

....HEX....<hex bytes>

Writes listed hex bytes into memory. Only first 3 bytes will be listed during assembly. Use 2-digit numbers (3A01FF)etc.

....ASC....'<string>' or "<string>"

Writes ASCII characters of <string> into memory. If <string> is delimited by ', then high bit of character is clear. If " is used, high bit is set. Only the first three bytes stored will be printed during Assembly, but all characters are written into memory.

....DCI....'<string>' or "<string>"

Similar to ASC, except the last character will have the high bit clear or set OPPOSITE of that set by choice of ' or ".

....PAG....

Clears screen or Form-Feeds printer.

....LST.<ON> or <OFF>

Turns assembly listing on or off as specified. Any portion or all of the listing may be turned on or off with this command. Default is 'ON'.

....PR#.<expr> or \$<addr>

Sends assembly listing to Slot #<expr> =1-7. If '\$<addr>', listing will be sent to routine at <addr>.

....END....

Optional operator signaling End of source file. Assembler will stop at END. If target labels or EQUs are located after END, errors will occur.

TED II+ also accepts:

....BLT....<label>, branch less than.

Same as BCC operator.

....BGE....<label>, branch greater than/equal. Same as BCS operator.

Legal Immediate Operands are:

('#' is used to signal Immediate)

#>LABEL Low byte of LABEL address.

#>LABEL High byte of LABEL address.

#"<chr> ASCII of <chr> High bit set.

#'<chr> ASCII of <chr> High bit clear.

##\$<num> Hex number.

#<num> decimal number.

Operands may be modified with the arithmetic operators: '+', '-', and '*'. These are one-byte modifiers only.

ASM is the direct command given to TED II+ to start the assembly process. If the list option is on, the assembler will halt at any errors detected. It stops at the line BEFORE the line with the error in it. Hitting the space bar will advance one line each time it is hit. Any other key will allow the assembler to continue running. A CTRL-C will exit the assembler at any time. Also, during assembly, the space bar may be used to temporarily halt the listing for examination. Any other key will continue assembly.

Any errors detected during LST OFF will not halt the assembler and will be displayed. A CTRL-C will exit the assembler at any time.

At the completion of assembly, the number of errors (in Dec.) will be shown, as well as the number of Bytes generated. This number does NOT include memory saved with the DS op-code, so take that into consideration when saving the object code. I generally put a 'NOP' at the very end of the source, and use the address to figure the length of the program.

After assembly, the user is in TED II+ Mode for editing or whatever.

While not perfect, TED II+ is probably the easiest assembler to use, especially for a beginner. There are occasions when it bombs for no apparent reason. The only thing to do then is to try to recover the source file and save it. RESET will get you out. Re-entering BASIC often does not work, since some pointers may have been destroyed. I have found that 'EFCG' (the BASIC run entry point) in monitor will restart the EXEC program. Save the file to disk and then COMPLETELY restart the whole thing over. This generally will take care of any problems when everything else fails. For some reason, TED II+ syntax errors occasionally write over the assembler itself.

Non-disk users can examine locations \$0A-0F to find the source table pointers. \$0A-0B contain the LOMEM (start of source) address, low byte first. \$0C-0D is HIMEM. \$0E-0F is the 'end of source' pointer. Thus, even if the assembler is totally destroyed, the source can be recovered using these pointers. With a little figuring, the disk user can BSAVE the source to disk using these pointers. In this case, the user MUST add the ".S" subscript 8

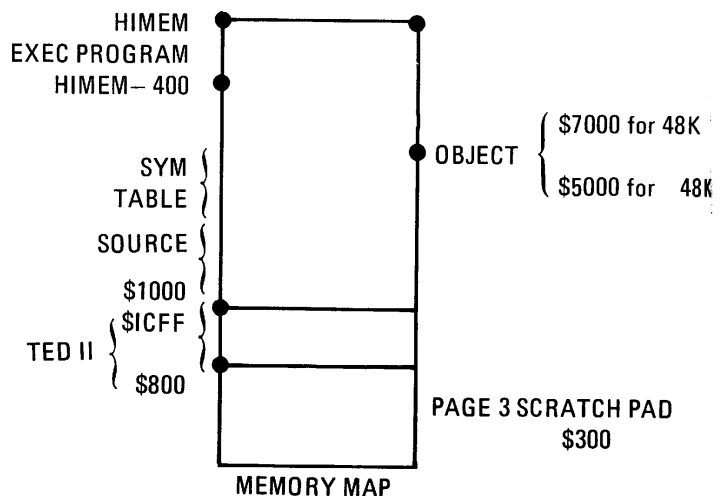
to the File name, so that TED II+ can find the file when the program is re-run.

The symbol table, which is created just above the end of the source, can be overwritten if the OBJ location is set too low. This usually is indicated by an 'UNDEFINED LABEL' error, when you KNOW that you used that label somewhere!

The SET operator (SWEET 16) does not recognize an immediate mode label. For this operator to work, the value to be assigned to a SWEET 16 register must be assigned a label, and that label entered as the register assignment (see the LAZARUS source listing, for example).

When creating a 'Title Box' using the asterisk, it's a good idea to keep the box as narrow as possible. I have found that a wide title box has an annoying habit of bombing the EDITOR.

That's all, folks! I think that I included all of the features of TED II+. The best way to learn how to use this assembler is to USE IT! A little practice will make its use almost automatic. I think you'll find it an easy and effective program development tool.



CREDIT WHERE CREDIT IS DUE DEPT:

This version of the Editor/Assembler represents a major overhaul of earlier versions of TED II, due to the efforts of Gary Shannon, Andy Hertzfeld, and Neil Konzen. I take the blame for the Integer Basic EXECUTIVE part of the program. It was based on some routines from an EXEC program written by Andy Hertzfeld, added to, subtracted from, and ended up as the EXEC front end that is presented here.

LINE NUMBER CROSS REFERENCE

for APPLE][
Integer BASIC Programs

BLANK PAGE

WOZPAK II

LINE NUMBER CROSS REFERENCE (LINE X-REF) generates a cross reference listing of all line numbers in an APPLE][Integer BASIC program. This routine scans a program, displays each line number that is referenced by statements in other lines, and displays those line numbers. Complex line number references such as "GOTO 100 +X" are flagged, notifying the user of possible renumbering conflicts.

Because LINE X-REF will flag complex line references, it should be used to check a program before renumbering (using RENUMBER/APPEND). A flagged line can be noted, and then hand-patched after the program is renumbered.

To use LINE XREF, simply load the routine (\$800.\$8F6), make sure the program to be checked is in memory, and call the routine. An output slot may be specified, if desired, before the routine is called.

LINE XREF contains no absolute memory references, so it may be moved without modification. CAUTION: you should not be in 'AUTO LINE NUMBER' mode when LINE XREF is called.

References to non-existent lines are not flagged. 'RUN <line number>' references are not flagged as it is very unlikely that such a statement exists in a program. Hand-patched delete statements (DEL 0,40) are flagged.

A source listing for LINE XREF was unavailable, but an APPLE disassembled listing of LINE XREF appears on the following page.

An example of the output from LINE XREF is shown below. An asterisk is used to flag line numbers that may provide renumbering conflicts.

```
>LIST
  5 GOSUB 10: GOSUB 10
 10 IF I=1 THEN GOTO 5
 15 PRINT
 20 GOTO I
 25 PRINT
 30 GOTO I*10
 35 PRINT
 40 GOTO 5: GOTO 5: GOTO I+5
 45 PRINT
 50 GOTO 10*I
100 LIST 5,100
```

>CALL2048

```

      LINE 10 references LINE 5
5- 10 40 40 100
10- 5 5 50*
      LINE 50 references LINE 10*I
      (note the '*' flag)
100- 100
***** 20 30* 40*
>
```

References such as 'GOTO X' or 'GOTO X+10' are displayed on last line. (note 5 '*')

0800-	A6 CA	LDX	\$CA
0802-	A5 CB	LDA	\$CB
0804-	A0 00	LDY	#\$00
0806-	84 0C	STY	\$0C
0808-	F0 0A	BEQ	\$0814
080A-	88	DEY	
080B-	A5 06	LDA	\$06
080D-	18	CLC	
080E-	71 06	ADC	(\$06),Y
0810-	AA	TAX	
0811-	98	TYA	
0812-	65 07	ADC	\$07
0814-	86 06	STX	\$06
0816-	85 07	STA	\$07
0818-	E4 4C	CPX	\$4C
081A-	E5 4D	SBC	\$4D
081C-	90 07	BCC	\$0825
081E-	A5 0C	LDA	\$0C
0820-	F0 01	BEQ	\$0823
0822-	60	RTS	

WOZPAK II

0823-	E6 0C	INC	\$0C	0889-	10 F8	BPL	\$0883
0825-	C8	INY		088B-	8A	TXA	
0826-	84 0E	STY	\$0E	088C-	05 0C	ORA	\$0C
0828-	B1 06	LDA	(\$06),Y	088E-	85 0D	STA	\$0D
082A-	85 08	STA	\$08	0890-	B0 D1	BCS	\$0863
082C-	C8	INY		0892-	A0 00	LDY	#\$00
082D-	B1 06	LDA	(\$06),Y	0894-	A5 04	LDA	\$04
082F-	85 09	STA	\$09	0896-	71 04	ADC	(\$04),Y
0831-	A6 CA	LDX	\$CA	0898-	AA	TAX	
0833-	A5 CB	LDA	\$CB	0899-	98	TYA	
0835-	88	DEY		089A-	65 05	ADC	\$05
0836-	86 04	STX	\$04	089C-	C8	INY	
0838-	85 05	STA	\$05	089D-	D0 97	BNE	\$0836
083A-	E4 4C	CPX	\$4C	089F-	A6 08	LDX	\$08
083C-	E5 4D	SBC	\$4D	08A1-	A5 09	LDA	\$09
083E-	B0 CA	BCS	\$080A	08A3-	20 1B E5	JSR	\$E51B
0840-	B1 04	LDA	(\$04),Y	08A6-	A9 AD	LDA	#\$AD
0842-	85 0A	STA	\$0A	08A8-	20 ED FD	JSR	\$FDED
0844-	C8	INY		08AB-	A2 01	LDX	#\$01
0845-	B1 04	LDA	(\$04),Y	08AD-	A5 24	LDA	\$24
0847-	85 0B	STA	\$0B	08AF-	C9 21	CMP	#\$21
0849-	84 0D	STY	\$0D	08B1-	90 05	BCC	\$08B8
084B-	90 14	BCC	\$0861	08B3-	20 8E FD	JSR	\$FD8E
084D-	C8	INY		08B6-	A2 07	LDX	#\$07
084E-	B1 04	LDA	(\$04),Y	08B8-	20 4A F9	JSR	\$F94A
0850-	C8	INY		08BB-	A6 0A	LDX	\$0A
0851-	C5 08	CMP	\$08	08BD-	A5 0B	LDA	\$0B
0853-	D0 06	BNE	\$085B	08BF-	20 1B E5	JSR	\$E51B
0855-	B1 04	LDA	(\$04),Y	08C2-	68	PLA	
0857-	C5 09	CMP	\$09	08C3-	48	PHA	
0859-	FD 06	BEQ	\$0861	08C4-	C9 0B	CMP	#\$0B
085B-	A5 0C	LDA	\$0C	08C6-	90 09	BCC	\$08D1
085D-	FD 04	BEQ	\$0863	08C8-	C9 75	CMP	#\$75
085F-	EA	NOP		08CA-	FD 05	BEQ	\$08D1
0860-	EA	NOP		08CC-	A9 AA	LDA	#\$AA
0861-	E6 0D	INC	\$0D	08CE-	20 ED FD	JSR	\$FDED
0863-	C8	INY		08D1-	68	PLA	
0864-	B1 04	LDA	(\$04),Y	08D2-	10 A3	BPL	\$0877
0866-	C9 C1	CMP	#\$C1	08D4-	48	PHA	
0868-	90 05	BCC	\$086F	08D5-	C6 0E	DEC	\$0E
086A-	C8	INY		08D7-	D0 D2	BNE	\$08AB
086B-	B1 04	LDA	(\$04),Y	08D9-	20 8E FD	JSR	\$FD8E
086D-	30 FB	BMI	\$086A	08DC-	20 8E FD	JSR	\$FD8E
086F-	C9 B0	CMP	#\$B0	08DF-	A5 0C	LDA	\$0C
0871-	B0 DA	BCS	\$084D	08E1-	FD BC	BEQ	\$089F
0873-	C6 0D	DEC	\$0D	08E3-	A2 05	LDX	#\$05
0875-	FD 5D	BEQ	\$08D4	08E5-	A9 AA	LDA	#\$AA
0877-	C9 28	CMP	#\$28	08E7-	20 ED FD	JSR	\$FDED
0879-	FD EF	BEQ	\$086A	08EA-	CA	DEX	
087B-	C9 5D	CMP	#\$5D	08EB-	D0 F8	BNE	\$08E5
087D-	FD EB	BEQ	\$086A	08ED-	FD BC	BEQ	\$08AB
087F-	C9 02	CMP	#\$02	08EF-	0D 03 7C	ORA	\$7C03
0881-	A2 07	LDX	#\$07	08F2-	01 50	ORA	(\$50,X)
0883-	5D EF 08	EOR	\$08EF,X	08F4-	78	SEI	
0886-	FD 04	BEQ	\$088C	08F5-	03	???	
0888-	CA	DEX		08F6-	5F	???	

12

SYMBOL CROSS REFERENCE
for APPLE][
Integer BASIC Programs

BLANK PAGE

WOZPAK II

SYMREF is a useful aid for APPLE][Integer BASIC Program development. It provides a complete listing of all variables that appear in the program, as well as all of the line numbers in which that variable appears. SYMREF can be used by the programmer to find misspellings, extra variables, and unassigned variable names. This routine will flag a FOR statement that has no corresponding NEXT statement. SYMREF also provides extra documentation for the programmer's records to assist later modifications to the program.

TO USE SYMREF:

Load BASIC program into memory.
Set LOMEM to '2560'
Load SYMREF (800.9E5)
Select printer, if desired.
'CALL 2048'(RETURN)
(or \$800G in monitor)

Be sure to set LOMEM before calling SYMREF, as the routine creates a table of variable names above LOMEM. If LOMEM is not set above SYMREF, the routine will overwrite itself.

All variable names used in the BASIC program will be printed in alphabetic order on the left edge of the output device. Each is followed by a list of all of the line numbers in which it appears, in ascending sequence. If the name is never assigned a value, an asterisk (*) is printed with the name as an "Undefined Variable" warning. Input variables are considered as being "undefined", unless the name is defined elsewhere in the BASIC program.

If there is insufficient memory for SYMREF to create its name table, the message 'MEM FULL' is displayed and SYMREF halts and returns control to the user.

SYMREF can be relocated by changing the 'JMP & JSR' destinations in lines 140,220, and 320 of the assembled listing.

The following is a sample Integer BASIC Program to illustrate the operation of SYMREF:

```
>LIST
 10 FOR UNDEFINED=1 TO 100
 15 FOR DEFINED=1 TO 100
 20 A=1
 25 PRINT I: PRINT I
 30 PRINT A: PRINT A: PRINT I
 35 NEXT DEFINED
 40 A$="12345"
 45 INPUT "INPUT VAR",INPUTVAR$

 50 PRINT A$: PRINT I$
```

```
>CALL2048
```

```
A- 20 30 30
```

```
A$- 40 50
```

```
DEFINED- 15 35
```

```
I*- 25 25 30
```

```
I$*- 50
```

```
INPUTVAR$*- 45
```

```
UNDEFINED*- 10
```

```
>
```

WOZPAK II

```

2 *****
3 *
4 *      APPLE ][ INTEGER BASIC      *
5 *      SYMBOL CROSS-REFERENCE     *
6 *
7 * APPLE PUGET SOUND PROGRAM LIBRARY EXCHANGE *
8 *   6708 39TH AVE. SW   SEATTLE, WA 98136 *
9 *
10 *      SOURCE CODE FROM 'THE WOZPAK' *
11 * COURTESY OF S. WOZNAK  APPLE COMPUTER CO. *
12 *
13 *      DECIPHERED AND ASSEMBLED     *
14 *      (AND SLIGHTLY MODIFIED BY)  *
15 *   KEN SMITH  TACOMA, WA  JUNE 10, 1979 *
16 *
17 *****
18 IBEGL  EQU  $4
19 IBEGH  EQU  $5
20 NENDL  EQU  $6
21 NENDH  EQU  $7
22 IENDL  EQU  $8
23 IENDH  EQU  $9
24 NBEGL  EQU  $A
25 NBEGH  EQU  $B
26 PL     EQU  $C
27 PH     EQU  $D
28 PO     EQU  $E
29 PN     EQU  $F
30 IL     EQU  $10
31 IH     EQU  $11
32 JL     EQU  $12
33 JH     EQU  $13
34 KL     EQU  $14
35 KH     EQU  $15
36 LL     EQU  $16
37 LH     EQU  $17
38 ML     EQU  $18
39 MH     EQU  $19
40 MODE   EQU  $1A
41 REF    EQU  $1B
42 HIMEML EQU  $4C
43 HIMEMH EQU  $4D
44 PPL    EQU  $CA
45 PPH    EQU  $CB
46 *
47 CH     EQU  $24
48 MEMFULL EQU  $E36B
49 PRDEC  EQU  $E51B
50 PRBL   EQU  $F94A
51 CROUT  EQU  $FD8E
52 COUT   EQU  $FDED
53 *
54      ORG  $800
55 *
56 *      INIT SYMBOL TABLE POINTERS
57 *      PPL,H (BASIC 'START
58 *      OF PROGRAM)
59 *
0800: A2 03 59 SYMREF LDX  #$3    -> IBEGL,H AND IENDL,H
0802: B5 CA 60 INIT   LDA  PPL,X  PVL,H ('END OF
0804: 95 04 61 STA  IBEGL,X VARIABLES)

```

WOZPAK II

```

0806: 95 08 63 STA IENDL,X -> NBEGL,H AND NENDL,H
0808: CA 64 DEX . THEN SCAN PROGRAM WITH
0809: 10 F7 65 BPL INIT MODE $FP TO FILL
080B: 30 39 66 BMI SCANPROG SYMBOL TABLE
080D: 20 8E FD 67 PRSYM JSR CROUT
0810: 20 8E FD 68 JSR CROUT OUTPUT 2 CAR RET'S
0813: A0 00 69 LDY #$0
0815: B1 18 70 LDA (ML),Y ML,H POINTS TO NAME TABLE
0817: 18 71 CLC . INDEX -RELATIVE TO
0818: 65 0A 72 ADC NBEGL NBEGL,H- FOR CURRENT
081A: 85 16 73 STA LL XREF SYMBOL
081C: C8 74 INY
081D: B1 18 75 LDA (ML),Y CALC POINTER TO FIRST
081F: 65 0B 76 ADC NBEGH CHAR OF NAME TABLE
0821: 85 17 77 STA LH ENTRY IN LL,H
0823: 88 78 DEY
0824: 38 79 SEC
0825: B1 16 80 PRNAME LDA (LL),Y NAME TABLE BYTE IN FORM
81 * ASCII * 2 +(NOT LAST)
0827: 6A 82 ROR
0828: 08 83 PHP
0829: C9 C0 84 CMP #$C0 SUBSTITUTE ASCII '$'
082B: D0 02 85 BNE PRNAM1 FOR STRING TOKEN
082D: A9 A4 86 LDA #$A4
082F: 20 ED FD 87 PRNAM1 JSR COUT OUTPUT CHAR OF NAME
0832: 28 88 PLP
0833: C8 89 INY . INCREMENT INDEX
0834: B0 EF 90 BCS PRNAME LOOP IF NOT LAST CHAR
0836: B1 16 91 LDA (LL),Y
0838: F0 05 92 BEQ PRNAM2 IF ATTRIBUTE BYTE (FOLLOWING
083A: A9 AA 93 LDA #$AA NAME) IS NON ZERO THEN
083C: 20 ED FD 94 PRNAM2 JSR COUT OUTPUT '*' AS UNDEFINED
083F: A9 AD 95 PRNAM2 LDA #$AD SYMBOL WARNING
0841: 20 ED FD 96 JSR COUT OUTPUT '-'
0844: A2 00 97 LDX #$0
0846: 86 1A 98 SCANPROG STX MODE MODE IS NONZERO FIRST
0848: A6 CA 99 LDX PPL TIME ONLY (AFTER INIT)
084A: A5 CB 100 LDA PPH BASIC 'START OF PROGRAM'
084C: 86 0C 101 NXTLINE STX PL
084E: 85 0D 102 STA PH INIT PROGRAM SCAN POINTER
0850: E4 4C 103 CPX HIMEML (POINTS TO START OF LINES)
0852: E5 4D 104 SBC HIMEMH IF '<'END OF PROGRAM' THEN
0854: 90 1C 105 BCC SCANLIN CONTINUE--(SCAN THIS LINE)
0856: A4 04 106 LDY IBEGL
0858: A5 05 107 LDA IBEGH FIRST TIME--(MODE NON ZERO)
085A: A6 1A 108 LDX MODE SET 'CURRENT SYMBOL POINTER'
085C: D0 09 109 BNE NEXTSYM ML, MH TO 'START OF SYMBOL
085E: A5 18 110 LDA ML INDICES' IBEGL,H
0860: 69 01 111 ADC #$1
0862: A8 112 TAY . AFTER--ADD 2 TO ML,H
0863: A5 19 113 LDA MH TO ADVANCE ONE SYMBOL
0865: 69 00 114 ADC #$0 INDEX (2 BYTES EACH)
0867: 84 18 115 NEXTSYM STY ML
0869: 85 19 116 STA MH IF 'CURRENT SYMBOL POINTER'
086B: C4 08 117 CPY IENDL ML,H < 'END OF INDICES'
086D: E5 09 118 SBC IENDH (IENDL,H) THEN BEGIN SCAN
086F: 90 9C 119 BCC PRSYM FOR THIS SYMBOL
0871: 60 120 RTS . ELSE, DONE
121 *
0872: A0 00 122 SCANLIN LDY #$0 LINE SCAN INDEX REL TO PL,H

```

			WOZPAK II			
0874:	98	124		TYA	-	START OF LINE. CLR REF.
0875:	C8	125	CONST	INY	-	SKIP 2 BYTE CONSTANT
0876:	C8	126		INY		
0877:	85 1B	127	ITEM	STA	REF	SET REF (0='ASSIGN MODE')
0879:	C8	128	ITEM1	INY		
087A:	B1 0C	129		LDA	(PL),Y	NEXT ITEM OF BASIC PROGRAM
087C:	30 27	130		BMI	NOTOKN	IF NEG, THEN CONST OR NAME.
087E:	C9 28	131		CMP	#\$28	'STRCON'
0880:	F0 04	132		BEQ	SKPASC	
0882:	C9 5D	133		CMP	#\$5D	'REM'
0884:	D0 05	134		BNE	TOKN1	IF \$STRCON OR REM TOKENS
0886:	C8	135	SKPASC	INY	-	THEN SKIP ALL SUBSEQUENT
0887:	B1 0C	136		LDA	(PL),Y	NEG (ASCII) BYTES
0889:	30 FB	137		BMI	SKPASC	
088B:	C9 02	138	TOKN1	CMP	#\$2	CLEAR CARRY ONLY IF EOL
088D:	A2 09	139		LDX	#\$9	
088F:	5D DC 09	140	REFTST	EOR	REFTB,X	IF COLON, THEN, FOR, LET,
0892:	F0 E3	141		BEQ	ITEM	INPUT, STRING INP, INP COMMA,
0894:	CA	142		DEX	-	OR STRINCOM, THEN SET
0895:	10 F8	143		BPL	REFTST	REF=0 TO INDICATE THAT
0897:	B0 DE	144		BCS	ITEM	NEXT (IMMEDIATE) NAME
0899:	A0 00	145		LDY	#\$0	IS ASSIGNED A VALUE
089B:	A5 0C	146		LDA	PL	
089D:	71 0C	147		ADC	(PL),Y	AT EOL, ADD LENGTH BYTE
089F:	AA	148		TAX	-	TO LINE POINTER PL,H
08A0:	98	149		TYA		
08A1:	65 0D	150		ADC	PH	
08A3:	90 A7	151		BCC	NXTLINE	(ALWAYS TAKEN)
08A5:	C9 C0	152	NOTOKN	CMP	#\$C0	
08A7:	90 CC	153		BCC	CONST	IF <\$C0 THEN FOLLOWED BY
08A9:	84 0E	154		STY	PO	2-BYTE CONSTANT.
08AB:	84 0F	155	NAMESCAN	STY	PN	SCAN NAME (NEG ASCII)
08AD:	C8	156		INY	-	WITH PO AND PN INDICES
08AE:	B1 0C	157		LDA	(PL),Y	TO FIRST, LAST CHARS.
08B0:	30 F9	158		BMI	NAMESCAN	
08B2:	C9 40	159		CMP	#\$40	INCLUDE 'S' TOKEN IN NAME
08B4:	F0 F5	160		BEQ	NAMESCAN	
08B6:	A5 04	161	SEARCH	LDA	IBEGL	IBEGL,H -> JL,H
08B8:	A6 05	162		LDX	IBEGH	LOW POINTERS
08BA:	85 12	163		STA	JL	
08BC:	86 13	164		STX	JH	
08BE:	A5 08	165		LDA	IENDL	IENDL,H -> KL,H
08C0:	A6 09	166		LDX	IENDH	HIGH POINTERS
08C2:	85 14	167	NEWK	STA	KL	
08C4:	86 15	168		STX	KH	
08C6:	E4 13	169	XTEST	CPX	JH	IF JL,H = KL,H THEN
08C8:	D0 04	170		BNE	KPLUSJ	EXIT WITH CARRY SET
08CA:	C5 12	171		CMP	JL	(NOT FOUND)
08CC:	F0 52	172		BEQ	ADDSYM	
08CE:	29 FE	173	KPLUSJ	AND	#\$FE	
08D0:	18	174		CLC	-	(JL,H+KL,H)/2 -> IL,H
08D1:	65 12	175		ADC	JL	
08D3:	29 FD	176		AND	#\$FD	
08D5:	85 10	177		STA	IL	IL,H FORCED ODD OR EVEN
08D7:	E6 10	178		INC	IL	TO AGREE WITH JL,H
08D9:	8A	179		TXA	-	AND KL,H
08DA:	65 13	180		ADC	JH	
08DC:	6A	181		ROR		
08DD:	85 11	182		STA	IH	
08DF:	66 10	183		ROR	IL	

			WOZPAK II	
08E1:	AA	185	TAX	
08E2:	A0 00	186	LDY	#\$0
08E4:	B1 10	187	LDA	(IL),Y
08E6:	38	188	SEC	
08E7:	E5 0E	189	SBC	PO NBEGL,H + INDEX (IL,H)L,H
08E9:	08	190	PHP	. -PO -> LL,H FOR BASE
08EA:	18	191	CLC	. ADDR TO NAME TABLE ENTRY
08EB:	65 0A	192	ADC	NBEGH CORRESPONDING TO IL,H
08ED:	85 16	193	STA	LL
08EF:	C8	194	INY	
08F0:	B1 10	195	LDA	(IL),Y
08F2:	65 0B	196	ADC	NBEGH
08F4:	28	197	PLP	
08F5:	69 FF	198	ADC	#\$FF
08F7:	85 17	199	STA	LH
08F9:	A4 0E	200	LDY	PO INDEX FOR NAME SCAN
08FB:	C4 0F	201	CPY	PN SET CARRY IF LAST CHAR
08FD:	B1 0C	202	LDA	(PL),Y OF USER SYMBOL
08FF:	2A	203	ROL	
0900:	49 01	204	EOR	#\$1 FORM ASCII * 2 + (NOT LAST)
0902:	D1 16	205	CMP	(LL),Y
0904:	FD 13	206	BEQ	MATCH
0906:	A5 10	207	LDA	IL IL,H -> KL,H IF USER SYMBOL
0908:	90 B8	208	BCC	NEWK LOWER (ALPHABETICALLY)
090A:	69 01	209	ADC	#\$1 THAN NAME TABLE ENTRY
090C:	85 12	210	STA	JL
090E:	8A	211	TXA	. IL,H -> JL,H IF USER SYMBOL
090F:	69 00	212	ADC	#\$0 GREATER (ALPHABETICALLY)
0911:	85 13	213	STA	JH THAN NAME TABLE ENTRY
0913:	A5 14	214	LDA	KL
0915:	A6 15	215	LDX	KH
0917:	90 AD	216	BCC	XTEST (ALWAYS TAKEN)
0919:	C8	217	INY	. INCR INDEX IF CHAR MATCH
091A:	4A	218	LSR	
091B:	B0 DE	219	BCS	SYMSCN LOOP IF NOT LAST CHAR, ELSE
091D:	4C AB 09	220	JMP	FOUND EXIT WITH CARRY CLEAR (FOUND)
0920:	A5 0F	221	LDA	PN ADDSYM
0922:	38	222	SEC	
0923:	E5 0E	223	SBC	PO NENDL,H + PO + PN+1 -> LL,H
0925:	65 06	224	ADC	NENDL (NEW NEND)
0927:	85 16	225	STA	LL
0929:	A5 07	226	LDA	NENDH
092B:	69 00	227	ADC	#\$0
092D:	85 17	228	STA	LH
092F:	A5 04	229	LDA	IBEGL
0931:	85 10	230	STA	IL IBEGL,H -> IL,H
0933:	E9 01	231	SBC	#\$1
0935:	85 14	232	STA	KL
0937:	A8	233	TAY	. IBEGL,H -2 -> KL,H AND Y,X
0938:	A5 05	234	LDA	IBEGH (NEW IBEG)
093A:	85 11	235	STA	IH
093C:	E9 00	236	SBC	#\$0
093E:	85 15	237	STA	KH
0940:	AA	238	TAX	
0941:	C4 16	239	CPY	LL IF NEW IBEG < NEW NEND
0943:	E5 17	240	SBC	LH THEN EXIT WITH CARRY
0945:	90 61	241	BCC	MFULLX CLEAR (MEM FULL)
0947:	84 04	242	STY	IBEGL
0949:	86 05	243	STX	IBEGH NEW IBEG -> IBEG
094B:	A0 00	244	LDY	#\$0

```

WOZPAK II
094D: A5 10 246 SPREAD LDA IL
094F: C5 12 247 CMP JL IF IL,H = JL,H THEN
0951: A5 11 248 LDA IH DONE SPREADING INDEX
0953: E5 13 249 SBC JH TABLE FOR INSERT
0955: B0 12 250 BCS ADDNAME
0957: B1 10 251 LDA (IL),Y MOVE BYTE OF INDEX TABLE
0959: 91 14 252 STA (KL),Y 2 LOCATIONS LOWER
095B: E6 10 253 INC IL
095D: D0 02 254 BNE SPRD2
095F: E6 11 255 INC IH INCR IL,H AND KL,H
0961: E6 14 256 SPRD2 INC KL
0963: D0 E8 257 BNE SPREAD
0965: E6 15 258 INC KH
0967: 90 E4 259 BCC SPREAD (ALWAYS TAKEN)
0969: A5 06 260 ADDNAME LDA NENDL
096B: E5 0A 261 SBC NBEGL
096D: 91 14 262 STA (KL),Y
096F: A5 07 263 LDA NENDH NENDL,H - NBEGL,H ->(KL,H)
0971: E5 0B 264 SBC NBEGH (NEW INDEX)
0973: C8 265 INY
0974: 91 14 266 STA (KL),Y
0976: A5 16 267 LDA LL
0978: 85 06 268 STA NENDL NEW NENDL,H -> NENDL,H
097A: 18 269 CLC
097B: E5 0F 270 SBC PN OLD NEND - PO TO LL,H
097D: 85 16 271 STA LL
097F: A5 17 272 LDA LH
0981: 85 07 273 STA NENDH
0983: E9 00 274 SBC #$0
0985: 85 17 275 STA LH
0987: A4 0E 276 LDY PO
0989: C4 0F 277 ADDCHR CPY PN
098B: B1 0C 278 LDA (PL),Y
098D: 2A 279 ROL
280 *
098E: 49 01 281 EOR #$1 ADD CHAR TO NAME TABLE
0990: 91 16 282 STA (LL),Y ENTRY. USE ASCII * 2
0992: C8 283 INY +(NOT LAST). SCAN FROM
0993: 4A 284 LSR (PL),PO TO (PL),PN
0994: B0 F3 285 BCS ADDCHR
0996: A5 06 286 LDA NENDL SET CARRY IF NENDL,H
0998: C5 04 287 CMP IBEGH = IBEGH,H (NO ROOM
099A: A5 07 288 LDA NENDH FOR ATTRIBUTE BYTE)
099C: E5 05 289 SBC IBEGH
099E: E6 06 290 INC NENDL INCR 'NAME TABLE END'
09A0: D0 02 291 BNE ADDRIF POINTER NENDL,H
09A2: E6 07 292 INC NENDH
09A4: A5 1B 293 ADDRIF LDA REF REF AS ATTRIBUTE
09A6: 90 07 294 BCC FOUND1 (0=UNASSIGNED)
09A8: 4C 6B E3 295 MFULLX JMP MEMFULL -ERROR-
09AB: A5 1B 296 FOUND LDA REF IF REF=0 (UNASSIGNED)
09AD: D0 02 297 BNE FOUND2 THEN SET ATTRIBUTE
09AF: 91 16 298 FOUND1 STA (LL),Y BYTE TO 0
09B1: A2 FF 299 FOUND2 LDX #$FF
09B3: B5 19 300 SYMTST LDA MH,X IF NOT CURRENT SYMBOL
09B5: 55 11 301 EOR IH,X (IL,H NOT = TO ML,H)
09B7: 05 1A 302 ORA MODE OR MODE = NONZERO
09B9: D0 1C 303 BNE TOITEM THEN DON'T PRINT REF
09BB: E8 304 INX LINE NUMBER
09BC: F0 F5 305 BEQ SYMTST (LEAVES X=1)

```

WOZPAK II

09BE:	A5 24	307		LDA	CH	IF CURSOR BEYOND COL#32
09C0:	C9 21	308		CMP	#\$21	THEN OUTPUT CARR RET
09C2:	90 05	309		BCC	PREF	AND TAB 6 SPACES
09C4:	20 8E FD	310		JSR	CROUT	
09C7:	A2 06	311		LDX	#\$6	
09C9:	20 4A F9	312	PREF	JSR	PRBL	ELSE, OUTPUT SINGLE SPACE
09CC:	A0 01	313		LDY	#\$1	
09CE:	B1 0C	314		LDA	(PL),Y	
09D0:	AA	315		TAX	.	PRINT CURRENT LINE
09D1:	C8	316		INY	.	NUMBER (POINTED TO
09D2:	B1 0C	317		LDA	(PL),Y	BY ML,H)
09D4:	20 1B E5	318		JSR	PRDEC	
09D7:	A4 0F	319	TOITEM	LDY	PN	RESTORE LINE SCAN
09D9:	4C 79 08	320		JMP	ITEM1	GET NEXT BASIC ITEM
		321	*			
09DC:	01 75 06	322	REFTB	HEX	017506	INT BASIC TOKENS
09DF:	0A 0B 70	323		HEX	0A0B70	TO IDENTIFY VARIABLES
09E2:	26 03	324		HEX	2603	
09E4:	55 58	325		HEX	5558	

--- END ASSEMBLY ---

TOTAL ERRORS: 00

486 BYTES OF OBJECT CODE
WERE GENERATED THIS ASSEMBLY.

:PR#0

:ASM

BLANK PAGE

RENUMBERING AND APPENDING BASIC PROGRAMS

BLANK PAGE

WOZPAK II

The answer to the question "What do 5, 11, 36, 150, 201, and 588 have in common?" is given as "Adjacent rooms in the Warsaw Hilton"(1) but might just as well be "Adjacent line numbers in my last BASIC program." The laws of entropy insure that the line numbers of a debugged and operational BASIC program give the appearance of having been selected by a KENO machine.* Many a time I have spent an extra hour to retype a finished program while spacing the line numbers evenly just to make it "look good".

Another difficulty which I have experienced is joining two BASIC programs into a single, larger one. This 'append' operation is easier to accomplish by hand than renumbering. The sophisticated user can examine the BASIC memory map and perform some manual manipulations to join the programs providing that the line numbers do not overlap. Still, the manual append operation is highly prone to error.

The APPLE][Basic user now has a solution to these needs in the form of a hand- or tape-loadable program, RENUM/APPEND, described herein. The CALL command is used to activate 1 of 3 machine level programs. The renumber operation (RENUM) requires user specification of the original line number range over which renumbering is to occur, the new initial line number to be applied to the range, and the new line number increment to use. The example below specifies that lines 200 to 340 be renumbered starting with 100 and spaced by 10's.

```
START 100
STEP 10
FROM 200
TO 340
```

A second RENUM entry renumbers the entire program, relieving the user of the need to specify the range of begin and end parameters. The append operation (APPEND) reads the second user (BASIC) program off tape with the first in memory.

Renumber and Append error conditions (memory full and line number overlap) are detected just as in BASIC. In case of error, the user is notified and no program alteration occurs.

[ed. note] RENUM/APPEND has been relocated to eliminate DOS pointer conflicts. It is necessary to set Basic LOMEM: to 2304 (or higher) to prevent the routine from overwriting itself. The user is reminded to convert the addresses given in the text for entry points.

(1) The Official Polish/Italian Joke Book, L. Wilde, Pinnacle Books, New York, N.Y., 1973, p. 17

* In fact, while several texts detail how the boundary conditions of a KENO game lead to predictable outcomes, finished programs seldom exhibit this property.

OLD ENTRY	NEW ENTRY
CALL 768	CALL 2048
CALL 776	CALL 2056
CALL 999	CALL 1255
CALL 1016	CALL 1272

WOZPAK II

USING RENUM/APPEND

ERRORS

1. Load RENUM/APPEND (* 300.3FF R)
 Note that the '*' is generated by the MONITOR, not the user.
2. Load a BASIC program.
3. To renumber an entire program.
 > CLR (clears variable table)
 > START = expr
 > STEP = expr
 > CALL 768
 Note: START and STEP must be specified in the order shown.

1. If not enough free memory exists to contain the line number table during pass 1 of RENUM, then the message '(beep) *** MEM FULL ERR' is displayed and no renumbering occurs. The same message is displayed if not enough free memory exists to hold the product of an APPEND. In the case of APPEND, the user will have to type the BASIC command CALL 1016 to recover his original program. The user can free additional memory by eliminating all active BASIC variables with the CLR command.
2. If renumbering results in a line number overlap (detected during pass 1 of RENUM) then the message '(beep) *** RANGE ERR' is displayed, and no renumbering occurs. This error may mean that one or more parameters were not specified or were incorrectly specified.

CAUTIONS

4. To renumber a range of the program.
 > CLR (clears variable table)
 > START = expr
 > STEP = expr
 > FROM = expr
 > TO = expr
 > CALL 776

1. When appending a program, always load the one with the greater line numbers first.
2. The user must be aware that branch target expressions may not be renumbered. For example, the statement GO TO ALPHA will not be modified by RENUM. The statement, GOTO 100 + ALPHA, will be modified only to reflect the new line number assigned to the old line 100.

5. To append program #2 (larger line numbers) to program #1 (smaller line numbers).
 (a) Load Program #2
 (b) CALL 999
 Be sure you are running the tape of program #1 as this command will load it.
 (c) If you get a memory full error, then use the command CALL 1016 to recover the original program.

APPLE][BASIC STRUCTURE

An understanding of the internal representation of a BASIC program is necessary in order to develop RENUMBER and APPEND algorithms. Figure 1 illustrates the significant pointers for a program in memory. Variable and symbol table assignment begins at the location whose address is contained in the pointer LOMEM (\$4A and \$4B where '\$' stands for hex). This is \$800 (2048) on the APPLE][unless changed by the user with the LOMEM: command. a second pointer, PV (Variable Pointer, at \$CC and \$CD) contains the address of the location immediately following the last location allocated to variables. PV is equal

WOZPAK II

to LOMEM if no variables are actively assigned as is the case after a NEW, CLR, or LOMEM: command. As variables are assigned PV increases.

The BASIC program is stored beginning with the lowest numbered line at the location whose address is contained in the pointer PP (Program Pointer, at \$CA and \$CB). The pointer HIMEM (\$4C and \$4D) contains the address of the location immediately following the last byte of the last line of the program. This is normally the top of memory unless changed by the user with the HIMEM: command. As the program grows, PP decreases. PP is equal to HIMEM if there is no program in memory. Adequate checks in the BASIC insure that PV never exceeds PP. This, in essence, says that variables and program are not permitted to overlap.

Lines of a BASIC program are not stored as they were originally entered (in ASCII) on the APPLE][due to a pre-translation stage. Internally each line begins with a length byte which may serve as a link to the next line. The length byte is immediately followed by a two-byte line number stored in binary, low-order byte first. Line numbers range from 0 to 32767. The line number is followed by 'items' of various types, the final of which is an 'end-of-line' token (\$01). Refer to Figure 2.

Single bytes of value less than \$80 (128) are 'tokens' generated by the translator. Each token stands for a fixed unit of text as required by the syntax of the language BASIC. Some stand for keywords such as PRINT or THEN while others stand for punctuation or operators such as ',' or '+.

Integer constants are stored as three consecutive bytes. The first contains \$B0-\$B9 (ASCII '0'-'9') signifying that the next two contain a binary constant stored low-order byte first. The line number itself is not preceded by \$B0-\$B9. All constants are in this form including line number references such as 500 in the statement GOTO 500. Constants are always followed by a token. Although one or both bytes of a constant may be positive (less than \$80) they are not tokens.

Variable names are stored as consecutive ASCII characters with the high-order bit set. The first character is between \$C1 and \$DA (ASCII 'A'-'Z'), distinguishing names from constants. All names are terminated

by a token which is recognizable by a clear high-order bit. The '\$' in string names such as A\$ is treated as a token.

String constants are stored as a token of value \$28 followed by ASCII text (with high-order bits set) followed by a token of value \$29. REM statements begin with the REM token (\$5D) followed by ASCII text (with high-order bits set) followed by the 'end-of-line' token.

FIGURE 1 - MEMORY MAP

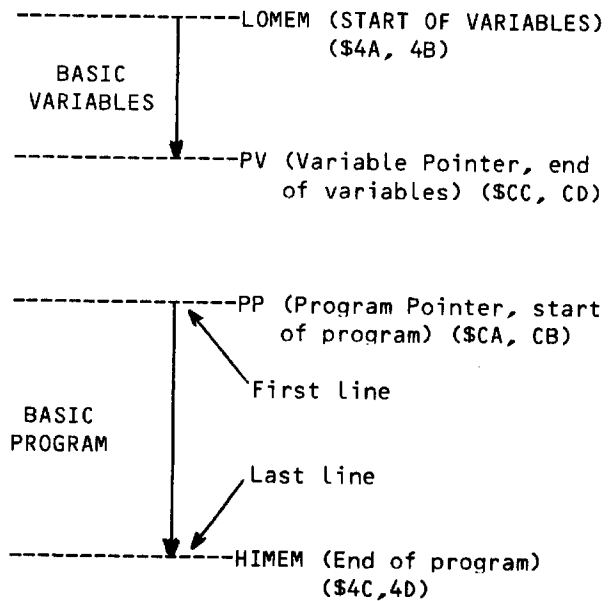
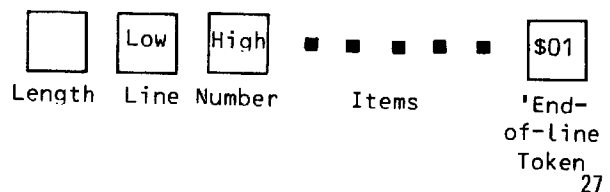
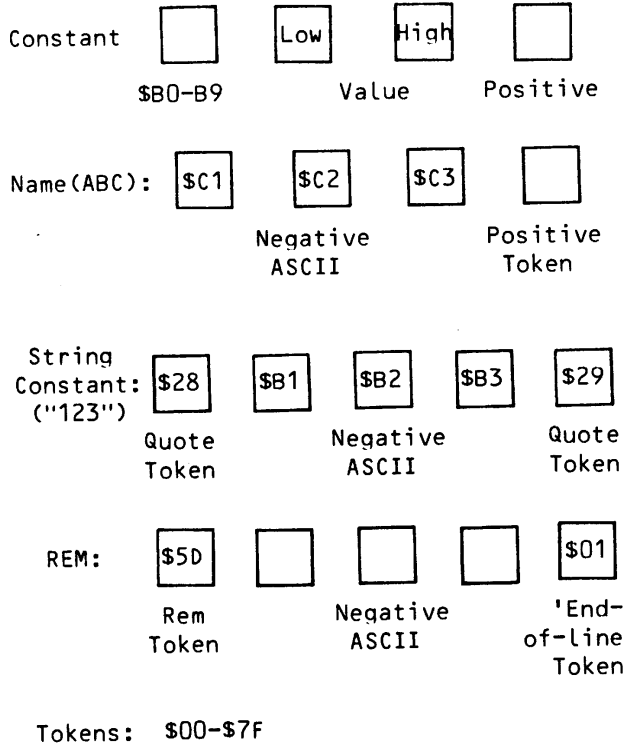


FIGURE 2 - LINE REPRESENTATION



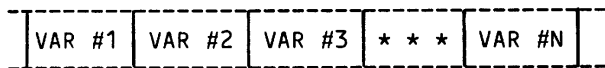
WOZPAK II

FIGURE 3 - ITEMS



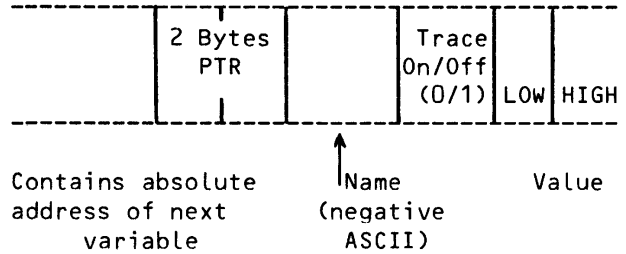
- GOTO - \$5F
- GOSUB - \$5C
- THEN ln - \$24
- LIST - \$74 (tokens used)
- LIST - \$75 (by RENUMBER)
- STR CON - \$28
- REM - \$5D
- EOL - \$01
- DEL - \$09
- DEL - \$0A
- RUN ln - \$07

INTEGER BASIC
VARIABLE STORAGE



LOMEM (\$4A,4B)
'Start of Vars'

PV (\$CC,CD)
'End of Vars'



For each simple variable, 3 bytes plus the number of characters in the variable name precede it's value.

Location of values for first 4 variables START, STEP, FROM, AND TO as used by RENUM.

VARIABLE	VALUE AT
START	LOMEM + \$08 (\$ For Hex)
STEP	LOMEM + \$11
FROM	LOMEM + \$1A
TO	LOMEM + \$21

RENUMBER - THEORY OF OPERATION

Because of the rigid internal representation of APPLE][BASIC programs (insured by the translator syntax check) writing a renumber program was a somewhat easier task than it would have been on many small BASICS. Fortunately all constants in APPLE][BASIC (including line number references) are preconverted to binary.

The normal renumber subroutine entry point is RENUM (\$308). The RENX entry conveniently sets the renumber range for the user such that the entire program will be renumbered. RENUM extensively uses SWEET 16 the code-saving 16-bit interpretive machine built into the APPLE][.(1) Occasional 6502 code is interspersed throughout RENUM for even greater code efficiency.

WOZPAK II

RENUM scans the entire program from beginning to end twice. During pass 1 a line number table is built containing all line numbers of the program found to be within the renumber range. This table begins at the address specified by the BASIC variable pointer, PV, and is limited in length by the program pointer, PP. Each entry is two bytes long. A memory full error occurs if not enough free memory is available for the table.

As line numbers are entered in the table, corresponding new line numbers are generated, and both new and old are displayed. Should the new line numbers result in an 'out of ascending sequence' condition, then a range error occurs and renumbering is terminated. It is assumed that the line numbers of the original program are in ascending sequence.

The purpose of pass 2 is to scan the entire BASIC program while updating all references of line numbers found in the table to new assignments. Aside from the line numbers themselves, the line number references sought are identified as constants immediately preceded by one of the following tokens:

```
GOTO ln
GOSUB ln
THEN ln
LIST ln
LIST ln, ln
DEL ln
DEL ln, ln
RUN ln
```

No other statement normally permitted within an APPLE][BASIC program may contain a line number reference. No errors will occur during pass 2.

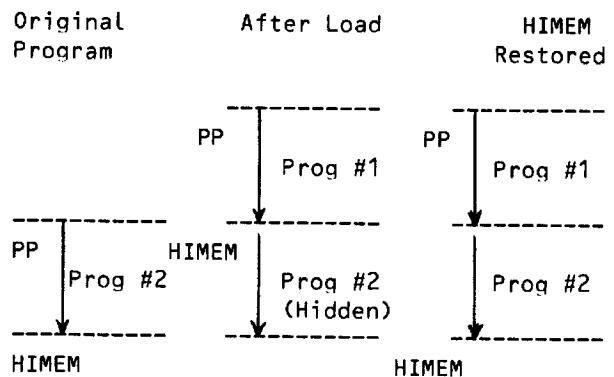
Exceptions such as empty line number table and null programs are properly considered by both passes of RENUM.

APPEND - THEORY OF OPERATION

When APPEND is called, the user program with larger line numbers will be in memory and the one with the smaller line numbers will be read off tape. The current program resides between pointers, PP and HIMEM. HIMEM is preserved and set to the value contained in PP.

This 'hides' the original program and prepares to load a new one immediately above it in memory.

The BASIC load subroutine is called and a normal memory full error condition will result if not enough free memory is available to contain both programs. If this error occurs then the original program will still be hidden. Fortunately, it can be recovered by calling the tail end of APPEND AT \$3CD, which simply restores HIMEM. If the load is successful then HIMEM is restored to it's final value and both programs will be joined. No line number overlap check is performed.



(1) Byte magazine, Nov. 1977

WOZPAK II

RENUMBER EXAMPLE

Original	Renumber lines 100-110 to start at 150 spaced by 10
>LIST	
1 GOTO 100	
2 GOSUB 103	>CLR
3 IF TRUE THEN 170	>START=150
4 LIST 109,110	>STEP=10
100 REM	>FROM=100
103 REM	>TO=110
107 REM	>CALL 776
109 REM	100->150
110 REM	103->160
200 FOR I=1 TO 10	107->170
210 PRINT I	109->180
220 NEXT I	110->190
230 GOTO 1	

RENUMBER EXAMPLE (Cont.)

```

>LIST
  1 GOTO 150
  2 GOSUB 160
  3 IF TRUE THEN 170
  4 LIST 180,190
150 REM
160 REM
170 REM
180 REM
190 REM
200 FOR I=1 TO 10
210 PRINT I
220 NEXT I
230 GOTO 1
    
```

Renumber all lines to
start at 10 spaced by 5

```

>CLR
>START=10
>STEP=5
>CALL 768
1->10
2->15
3->20
4->25
150->30
160->35
170->40
180->45
190->50
200->55
210->60
220->65
230->70
    
```

```

>LIST
 10 GOTO 30
 15 GOSUB 35
 20 IF TRUE THEN 40
 25 LIST 45,50
 30 REM
 35 REM
 40 REM
 45 REM
 50 REM
 55 FOR I=1 TO 10
 60 PRINT I
 65 NEXT I
 70 GOTO 10
    
```

WOZPAK

```

3 *****
4 *
5 * APPLE ][ BASIC RENUMBER/APPEND SUBROUTINES *
6 *
7 *          VERSION 2          *
8 *
9 *          RENUMBER:         *
10 *          >CLR              *
11 *          >START=          *
12 *          >STEP=          *
13 *          >CALL 2048       *
14 *
15 *          OPTIONAL:        *
16 *          >FROM=          *
17 *          >TO=            *
18 *          >CALL 2056       *
19 *
20 *          USE RENX ENTRY    *
21 *          FOR RENUMBER ALL  *
22 *
23 *          WOZ          APRIL 12, 1978 *
24 *          APPLE COMPUTER, INC. *
25 *
26 *****
27 *
28 ROL      EQU $0      LOW-ORDER SW16 R0 BYTE.
29 ROH      EQU $1      HI-ORDER.
30 R11L     EQU $16     LOW-ORDER SW16 R11 BYTE.
31 R11H     EQU $17     HIGH-ORDER.
32 HIMEM    EQU $4C     BASIC HIMEM POINTER.
33 PPL      EQU $CA     BASIC PROGRAM POINTER.
34 PVL      EQU $CC     BASIC VARIABLES POINTER.
35 MEMFULL  EQU $E36B   BASIC MEM FULL ERROR.
36 PRDEC    EQU $E51B   BASIC DECIMAL PRINT SUBR.
37 RANGERR  EQU $EE68   BASIC RANGE ERROR.
38 LOAD     EQU $F0DF   BASIC TAPE LOAD SUBR.
39 SW16     EQU $F689   SWEET-16 ENTRY.
40 CROUT    EQU $FD8E   CARRIAGE RETURN SUBR.
41 COUT     EQU $FDED   CHARACTER OUT SUBR.
42 *
43 * SWEET 16 EQUATES
44 *
45 ACC      EQU $0      SWEET 16 ACCUMULATOR.
46 NEWLOW   EQU $1      NEW INITIAL LNO.
47 NEWINCR  EQU $2      NEW LNO INCR.
48 LNLOW    EQU $3      LOW LNO OF RENUM RANGE.
49 LNHI     EQU $4      HI LNO OF RENUM RANGE.
50 TBLSTRT  EQU $5      LNO TABLE START.
51 TBLNDX1  EQU $6      PASS 1 LNO TBL INDEX.
52 TBLIM    EQU $7      LNO TABLE LIMIT.
53 SCR8     EQU $8      SCRATCH REG.
54 HMEM     EQU $8      HIMEM (END OF PROGRAM).
55 SCR9     EQU $9      SCRATCH REG.
56 PRGNDX   EQU $9      PASS 1 PROGRAM INDEX.
57 PRGNDX1  EQU $A      ALSO PROG INDEX..
58 NEWLN    EQU $B      NEXT "NEW LNO"
59 NEWLN1   EQU $C      PRIOR "NEW LNO" ASSIGN.
60 TBLND    EQU $6      PASS 2 LNO TABLE END.
61 PRGNDX2  EQU $7      PASS 2 PROG INDEX.
62 CHRO     EQU $9      ASCII "0"
    
```

WOZPAK II

```

64 CHRA EQU $A ASCII "A"
65 MODE EQU $C CONST/LNO MODE.
66 TBLNDX2 EQU $B LNO TBL INDX FOR UPDATE.
67 OLDLN EQU $D OLD LNO FOR UPDATE.
68 STRCON EQU $B BASIC STR CON TOKEN.
69 REM EQU $C BASIC REM TOKEN.
70 R13 EQU $D SWEET 16 REG. 13 (CPR REG).
71 THEN EQU $D BASIC THEN TOKEN.
72 LIST EQU $D BASIC LIST TOKEN.
73 DEL EQU $D BASIC DEL TOKEN.
74 SCRC EQU $C SCRATCH REG. FOR APPEND.
75 *
76 ORG $800
77 *
*****
79 * APPLE ][ INTEGER BASIC RENUMBER SUBROUTINE-PASS 1
*****
81 *
0800: 20 89 F6 82 RENX JSR SW16 RENUMBER ALL ENTRY
0803: 80 83 SUB ACC
0804: 33 84 ST LNLOW
0805: 34 85 ST LNHI
0806: 01 06 86 BR RNUM2
0808: 20 89 F6 87 RENUM JSR SW16 OPTIONAL RANGE ENTRY.
080B: 14 01 00 88 SET LNHI,1
080E: 19 4E 00 89 RNUM2 SET SCR9,HIMEM+2
0811: C9 90 POPD @SCR9
0812: 38 91 ST HMEM BASIC HIMEM POINTER TO HMEM.
0813: C9 92 POPD @SCR9 BASIC START OF VARS.
0814: 39 93 ST SCR9
0815: 69 94 LDD @SCR9 SKIP NAME 'START' (8 BYTES)
0816: 69 95 LDD @SCR9 OF FIRST BASIC VAR. IN SYM. TBL.
0817: 69 96 LDD @SCR9
0818: 69 97 LDD @SCR9
0819: 69 98 LDD @SCR9 (VALUE OF 'START')
081A: 31 99 ST NEWLOW
081B: 69 100 LDD @SCR9 SKIP NAME 'STEP' (7 BYTES)
081C: 69 101 LDD @SCR9
081D: 69 102 LDD @SCR9
081E: 49 103 LD @SCR9
081F: 69 104 LDD @SCR9 (VALUE OF 'STEP')
0820: 32 105 ST NEWINCR
0821: F4 106 DCR LNHI DECREMENT 'TO' TO -1 IF 'ACC'.
0822: 08 0B 107 BM1 RNUM3
0824: 69 108 LDD @SCR9 SKIP NAME 'FROM' (7 BYTES)
0825: 69 109 LDD @SCR9
0826: 69 110 LDD @SCR9
0827: 49 111 LD @SCR9
0828: 69 112 LDD @SCR9 (VALUE OF 'FROM')
0829: 33 113 ST LNLOW
082A: 69 114 LDD @SCR9 SKIP NAME 'TO' (5 BYTES).
082B: 69 115 LDD @SCR9
082C: 49 116 LD @SCR9
082D: 69 117 LDD @SCR9 (VALUE OF 'TO')
082E: 34 118 ST LNHI
082F: 19 CE 00 119 RNUM3 SET SCR9,PVL+2
0832: C9 120 POPD @SCR9 BASIC VAR PNTR TO
0833: 35 121 ST TBLSTRT TBLSTRT AND TBLNDX1.
0834: 36 122 ST TBLNDX1
0835: 21 123 LD NEWLOW COPY NEWLOW (INITIAL)

```

```

                                WOZPAK II
0836: 3B          125          ST   NEWLN          TO NEWLN.
0837: 3C          126          ST   NEWLN1
0838: C9          127          POPD @SCR9          BASIC PROG. PNTR
0839: 37          128          ST   TBLIM          TO TBLIM AND PRGNDX.
083A: 39          129          ST   PRGNDX
083B: 29          130          LD   PRGNDX
                                PASS1
083C: D8          131          CPR  HMEM          IF PRGNDX >= HMEM
083D: 03 46      132          BC   PASS2          THEN DONE PASS 1.
083F: 3A          133          ST   PRGNDX1
0840: 26          134          LD   TBLNDX1
0841: E0          135          INR  ACC          IF < TWO BYTES AVAIL IN
0842: D7          136          CPR  TBLIM          LNO TABLE THEN RETURN
0843: 03 38      137          BC   MERR          WITH 'MEM FULL' MESSAGE.
0845: 4A          138          LD   @PRGNDX1
0846: A9          139          ADD  PRGNDX          ADD LENGTH BYTE TO PROG INDEX.
0847: 39          140          ST   PRGNDX
0848: 6A          141          LDD  @PRGNDX1          LINE NUMBER.
0849: D3          142          CPR  LNLOW          IF <LNLOW THEN GOTO P1B.
084A: 02 2A      143          BNC  P1B
084C: D4          144          CPR  LNHI          IF >LNHI THE GOTO P1C.
084D: 02 02      145          BNC  P1A
084F: 07 30      146          BNZ  P1C
0851: 76          147          STD  @TBLNDX1          ADD TO LNO TABLE.
                                P1A
0852: 00          148          RTN
0853: A5 01      149          LDA  ROH          **** 6502 CODE ****
0855: A6 00      150          LDX  ROL
0857: 20 1B E5   151          JSR  PRDEC          PRINT OLD LNO "->" NEW LNO
085A: A9 AD      152          LDA  #$AD          (RO,R11) IN DECIMAL.
085C: 20 ED FD   153          JSR  COUT
085F: A9 BE      154          LDA  #$BE
0861: 20 ED FD   155          JSR  COUT
0864: A5 17      156          LDA  R11H
0866: A6 16      157          LDX  R11L
0868: 20 1B E5   158          JSR  PRDEC
086B: 20 8E FD   159          JSR  CROUT
086E: 20 8C F6   160          JSR  SW16+3          **** END 6502 CODE ****
0871: 2B          161          LD   NEWLN
0872: 3C          162          ST   NEWLN1
0873: A2          163          ADD  NEWINCR          COPY NEWLN TO NEWLN1 AND INCR
0874: 3B          164          ST   NEWLN          NEWLN BY NEWINCR.
0875: 0D          165          HEX  OD          'NULL' (WILL SKIP NEXT INSTR.
0876: D1          166          CPR  NEWLOW          IF LOW LNO < NEW LNO THEN RANGE ERR.
0877: 02 C2      167          BNC  PASS1
0879: 00          168          RTN          ;PRINT "RANGE ERR" AND RETURN.
087A: 4C 68 EE   169          JMP  RANGERR
087D: 00          170          MERR          ;PRINT "MEM FULL" AND RETURN.
087E: 4C 6B E3   171          JMP  MEMFULL
0881: EC          172          P1C          INR  NEWLN1          IF HI LNO <= MOST RECENT NEWLN
0882: DC          173          CPR  NEWLN1          THEN RANGE ERROR.
0883: 02 F4      174          BNC  RERR
                                *
                                *****
177 * APPLE ][ BASIC RENUMBER/APPEND SUBROUTINE - PASS 2
                                *****
179 *
0885: 19 B0 00   180          PASS2          SET  CHRO,B0          ASCII "0".
0888: 1A C1 00   181          SET  CHRA,C1          ASCII "A".
088B: 27          182          P2A          LD   PRGNDX2
088C: D8          183          CPR  HMEM          IF PROG INDEX=HMEM THEN DONE PASS
088D: 03 6F      184          BC   DONE

```

```

                                WOZPAK II
088F: E7          186          INR PRGNDX2  SKIP LENGTH BYTE.
0890: 67          187          LDD @PRGNDX2 LINE NUMBER.
0891: 3D          188  UPDATE ST  OLDLN  SAVE OLD LNO.
0892: 25          189          LD  TBLSTRT
0893: 3B          190          ST  TBLNDX2  INIT LNO TABLE INDEX.
0894: 21          191          LD  NEWLOW  INIT NEWLN TO NEWLOW.
0895: 1C 00 00   192          SET NEWLN1,0 (WILL SKIP NEXT 2 INSTRUCTIONS)
                                193          ORG  *-2
0896: 2C          194  UD2   LD  NEWLN1
0897: A2          195          ADD NEWINCR  ADD INCR TO NEWLN1.
0898: 3C          196          ST  NEWLN1
0899: 2B          197          LD  TBLNDX2  IF LNO TBL IDX=TBLND THEN DONE
089A: B6          198          SUB TBLND  SCANNING LNO TABLE.
089B: 03 07      199          BC  UD3
089D: 6B          200          LDD @TBLNDX2 NEXT LNO FROM TABLE.
089E: BD          201          SUB OLDLN  LOOP TO UD2 IF NOT SAME AS OLDLN.
089F: 07 F5      202          BNZ UD2
08A1: C7          203          POPD @PRGNDX2 REPLACE OLD LNO WITH CORRESPONDING
08A2: 2C          204          LD  NEWLN1  NEW LINE.
08A3: 77          205          STD @PRGNDX2
08A4: 1B 28 00   206  UD3   SET STRCON,28 STR CON TOKEN.
08A7: 1C 00 00   207          SET MODE,0  (SKIPS NEXT TWO INSTRUCTIONS)
                                208          ORG  *-2
08A8: 67          209  GOTCON LDD @PRGNDX2
08A9: FC          210          DCR MODE  IF MODE=0 THEN UPDATE LNO REF.
08AA: 08 E5      211          BM1 UPDATE
08AC: 47          212  ITEM  LD  @PRGNDX2  BASIC TOKEN.
08AD: D9          213          CPR CHRO
08AE: 02 09      214          BNC CHKTOK  CHECK TOKEN FOR SPECIAL.
08B0: DA          215          CPR CHRA  IF >= "0" AND < "A" THEN SLIP CONST
08B1: 02 F5      216          BNC GOTCON  OR UPDATE.
08B3: F9          217  SKPASC DCR PRGNDX
08B4: 67          218          LDD @PRGNDX2  SKIP ALL NEG. BYTES OF STR CON,
08B5: 05 FC      219          BM  SKPASC  REM, OR NAME.
08B7: F7          220          DCR PRGNDX2
08B8: 47          221          LD  @PRGNDX2
08B9: DB          222  CHKTOK CPR STRCON  STR CON TOKEN?
08BA: 06 F7      223          BZ  SKPASC  YES, SKIP SUBSEQUENT BYTES.
08BC: 1C 5D 00   224          SET REM,5D
08BF: DC          225          CPR REM  REM TOKEN?
08C0: 06 F1      226          BZ  SKPASC  YES, SKIP SUBSEQUENT LINE.
08C2: 08 1F      227          BM1 CONSTST GOSUB, LOOK FOR LINE NUMBER.
08C4: FD          228          DCR R13
08C5: FD          229          DCR R13  (TOKEN $5F IS GOTO)
08C6: 06 1B      230          BZ  CONSTST
08C8: 1D 24 00   231          SET THEN,24
08CB: DD          232          CPR THEN
08CC: 06 15      233          BZ  CONSTST  'THEN' LNO, LOOK FOR LNO.
08CE: FD          234          DCR ACC
08CF: 06 BA      235          BZ  P2A  EOL (TOKEN 01)
08D1: 1D 09 00   236          SET DEL,9  'DEL' OR 'DEL X,X', LOOK FOR LNO.
08D4: DD          237          CPR DEL  (TOKENS $9, $A)
08D5: 08 0C      238          BM1 CONSTST
08D7: 06 0A      239          BZ  CONSTST
08D9: ED          240          INR R13
08DA: ED          241          INR R13
08DB: 08 06      242          BM1 CONSTST  (RUN LN= TOKEN 7)
08DD: 1D 74 00   243          SET LIST,74
08E0: BD          244          SUB LIST  SET MODE=0 IF LIST OR LIST COMM
08E1: 09 01      245          BNM1 CONTS2  (TOKENS $74, $75)

```


WOZPAK II

```

08E3: 80      247  CONTST  SUB  ACC      CLEAR MODE FOR LNO.
08E4: 3C      248  CONTS2   ST   MODE     UPDATE CHECK.
08E5: 01 C5   249          BR   ITEM
250  *
*****
252  *  APPLE ][ BASIC APPEND SUBROUTINE
*****
254  *
08E7: 20 89 F6 255  APPEND   JSR  SW16
08EA: 1C 4E 00 256          SET  SCRC,HIMEM+2
08ED: CC      257          POPD @SCRC      SAVE HIMEM
08EE: 38      258          ST   HMEM
08EF: 19 CA 00 259          SET  SCR9,PPL
08F2: 69      260          LDD  @SCR9
08F3: 7C      261          STD  @SCRC      SET HIMEM TO PRESERVE PROGRAM.
08F4: 00      262          RTN
08F5: 20 DF F0 263          JSR  LOAD      LOAD FROM TAPE.
08F8: 20 89 F6 264          JSR  SW16
08FB: CC      265          POPD @SCRC      RESTORE HIMEM TO SHOW BOTH
08FC: 28      266          LD   HMEM      PROGRAMS (OLD AND NEW)
08FD: 7C      267          STD  @SCRC
08FE: 00      268  DONE    RTN
08FF: 60      269          RTS
    
```

--- END ASSEMBLY ---

TOTAL ERRORS: 0

260 BYTES GENERATED THIS ASSEMBLY

BLANK PAGE

APPLE][INTEGER BASIC
SUBROUTINE PACK and LOAD
by
Richard F. Suitor

BLANK PAGE

WOZPAK II

[ed. note] This routine was written by Richard F. Suitor and published in MICRO #6. Apple Pugetsound Program Library Exchange wishes to thank Dick Suitor and MICRO magazine for their permission to reprint this article.

The first issue of CONTACT, the APPLE newsletter, gave a suggestion for loading assembly language routines with a BASIC program. Simply summarized, one drops the pointer of the BASIC beginning below the assembly portion, adds a BASIC statement that will restore the pointer and SAVes. The procedure is simple and effective, but has two limitations. First, it is inconvenient if BASIC and the routines are widely separated (and is very tricky if the routines start at \$800, just above the display portion of memory). Second, a program so saved cannot be used with another HIMEM and is thus inconvenient to share or to submit to a software exchange.

The subroutine presented here avoids these difficulties at the expense of the effort to implement it. It is completely position independent; it may be moved from place to place in core with the monitor move command and used at the new location without modification. It makes extensive use of SWEET 16, the 16 bit interpreter supplied as part of the Apple Monitor ROM.

To use the routine from Apple Integer BASIC, CALL MKUP, where MKUP is 128 (decimal) plus the first address of the routine. The prompt shown is "@". Respond with the hex limits of the routine to be stored, as BBBB.EEEE (BBBB is the beginning address, EEEE is the ending; the same format that the monitor uses). Several groups may be specified on one line separated by spaces, or several lines. Type 'S' after the last group to complete the pack and return to BASIC. The program can now be saved.

To load, enter BASIC and LOAD. When complete, RUN. The first RUN will move all routines back to their original location and return control to BASIC. It will not RUN the program; subsequent RUNs will. (*1)

A LIST of the program after calling MKUP and before the first RUN will show one BASIC statement (which initiates the restoration process) and gibberish. If this is done, RESET followed by CTRL-C will return control to BASIC.

WARNING #1: The routine must be placed in core where it will not overwrite itself during the Pack. The start of the routine must be above HIMEM (e.g. in the high resolution display region), or \$17A+4*N+W below the start of the BASIC program where N is the number of routines stored and W is the total number of words in all of these routines. Also, those routines that are highest in memory should be packed first to avoid overwriting during pack or restore. Otherwise it is not necessary to worry about overwriting during the restore process; only \$1A words just below the BASIC program are used.

WARNING #2: Do not attempt to edit the program after calling MKUP. If editing is necessary, RUN once to unpack, then edit and call MKUP again.

The routine works as follows. It first packs the restore routine just below the BASIC program. It then packs other routines as requested, with the first address and the number of bytes (words). When S is given, it packs itself with the information to restore LOMEM and the beginning of the BASIC program. The first \$46 words of the routine form a BASIC statement which will initiate the restoration process when RUN is typed.

If a particular HIMEM is needed by the program (e.g. for high resolution programs) it must be entered before LOADING. The LOMEM will be reset by the restoration process to the value it had when MKUP was called.

I do not have a SWEET 16 assembler, hence all of those op codes are listed as tables of data.(*2) In the listing, comments indicate where constants and relative displacements are differences between labels in the routine.

Some convenient load and entry points are:

BASO (load)		MKUP (entry)	
hex		hex	decimal
800		880	2176
A90		B10	2832
104C		10CC	4300
2050		20D0	8400
3054		30D4	12500

[ed. notes]

(*1) Since this article was written, the RUNNING entry to basic has been added, allowing only one RUN to be required. See assembly listing.

(*2) Listing has been rewritten on TED II+ showing SWEET 16 op-codes.

WOZPAK II

```

2      *
3      *INT BASIC SUBR PACK & LOAD
4      *CALL BASO+128(DEC)
5      *
6      * CONVERTED 5/17/79
7      * CONTAINS SW16 MACRO DEFS
8      *
9      * SEE MICRO#6 OR BEST OF MICRO
10     * FOR DOC.
11     *
12     * CHANGE LAST INST. TO
13     * JMP BRUN TO UNPACK&RUN
14     * IN ONE STEP
15     *
16     * POSITION INDEPENDENT PROG.
17             OBJ   $3000
18             ORG   $800
19     *
20     ACCL      EQU   $00
21     BSOL      EQU   $02
22     TABL      EQU   $04
23     TBCL      EQU   $06
24     HIMS      EQU   $08
25     LMRT      EQU   $0A
26     BPRG      EQU   $0C
27     FRML      EQU   $0E
28     NBYT      EQU   $10
29     BPR2      EQU   $12
30     PTLN      EQU   $14
31     XTAB      EQU   $16
32     SKPL      EQU   $18
33     MODE      EQU   $31
34     YSAV      EQU   $34
35     PRMP      EQU   $33
36     LMML      EQU   $4A
37     HIML      EQU   $4C
38     LMWL      EQU   $CC
39     BBSL      EQU   $CA
40     JSRL      EQU   $CE
41     BSC2      EQU   $E003 BASIC
42     BRUN      EQU   $EFEC RUN BASIC
43     BUFF      EQU   $0200
44     SW16      EQU   $F689
45     GTNM      EQU   $FFA7
46     PBL2      EQU   $F94A
47     COUT      EQU   $FDED
48     BELL      EQU   $FF3A
49     GTLN      EQU   $FD67
50     *
51     *BASIC INST. TO RESTORE
0800: 46 00 00 52     BASO      HEX   460000
0803: 64 B1 01 53             HEX   64B101
0806: 00 65 B7 54             HEX   0065B7
0809: 4C 00 03 55             HEX   4C0003
080C: 64 B2   56             HEX   64B2
080E: 02 00 65 57             HEX   020065
0811: 38 2E 3F 58             HEX   382E3F
0814: B2 CA   59             HEX   B2CA
0816: 00 72 12 60             HEX   007212
0819: B7 46 00 61             HEX   B74600
    
```

```

WOZPAK II
081C: 72 1F 63      HEX 721F
081E: B2 00 01 64   HEX B20001
0821: U3 64 B3 65   HEX 0364B3
0824: 03 00 66      HEX 0300
0826: 65 38 2E 67   HEX 65382E
0829: 3F B2 CB 68   HEX 3FB2CB
082C: 00 72 69      HEX 0072
082E: 12 38 2E 70   HEX 12382E
0831: 3F B2 CA 71   HEX 3FB2CA
0834: 00 72 72      HEX 0072
0836: 12 B7 46 73   HEX 12B746
0839: 00 72 15 74   HEX 007215
083C: B2 00 75      HEX B200
083E: 01 72 03 76   HEX 017203
0841: 4D B1 01 77   HEX 4DB101
0844: 00 01 78      HEX 0001
0846: D8 80         *INIT. RESTORE OP
0847: A2 01 81      PTBK CLD
0849: B5 CA 82      PT02 LDA BBSL,X
084B: 95 02 83      STA BSOL,X
084D: B5 4C 84      LDA HIML,X
084F: 95 08 85      STA HIMS,X
0851: CA 86         DEX
0852: 10 F5 87      BPL PT02
0854: 20 89 F6 88   JSR SW16
0857: 10 52 01 89   SET 0,(PTLP-BAS0)
085A: 18 57 01 90   SET 8,(PTLP+5-BAS0)
085D: A1 91         ADD 1
085E: 37 92         ST 7
085F: 67 93         LDD @7
0860: 35 94         ST 5
0861: 67 95         LDD @7
0862: 36 96         ST 6
0863: 24 97         LD 4
0864: B6 98         SUB 6
0865: 36 99         ST 6
0866: 1A 11 00 100  SET 10,(ST16+1-PLP1)
0869: BA 101        SUB 10
086A: 3A 102        ST 10
086B: 67 103        LDD @7
086C: 33 104        ST 3
086D: 00 105        RTN
086E: A2 01 106     LDX #1
0870: B5 0A 108     *SET LOMEM & BASIC PROG START
0872: 95 4A 109     PT04 LDA LMRT,X
0874: 95 CC 110     STA LMML,X
0876: B5 0C 111     LDA BPRG,X
0878: 95 CA 112     STA BBSL,X
087A: CA 113        DEX
087B: 10 F3 114     BPL PT04
087D: 6C 14 00 115  JMP (PTLL) TO RESTORE LP
0880: A2 01 116     *SUBR TO SET UP PACK
0882: B5 4A 118     MKUP LDX #1
0884: 95 0A 119     MK21 STA LMML,X
0886: B5 CA 120     LDA BBSL,X
0888: 95 12 121     STA BPR2,X
088A: 95 0C 122     STA BPRG,X

```

WOZPAK II

```

088C: B5 CE 124 LDA JSRL,X
088E: 95 04 125 STA TABL,X
0890: B5 4C 126 LDA HIML,X
0892: 95 08 127 STA HIMS,X
0894: CA 128 DEX
0895: 10 EB 129 BPL MK21
130 *INIT & PACK RESTORE LP
0897: 20 89 F6 131 JSR SW16
089A: 24 132 LD 4
089B: B9 133 SUB 9
089C: 39 134 ST 9
089D: 11 80 00 135 SET 1,(MKUP-BAS0)
08A0: 22 136 LD 2
08A1: B1 137 SUB 1
08A2: 31 138 ST 1
08A3: 10 52 01 139 SET 0,(PTLP-BAS0)
08A6: A1 140 ADD 1
08A7: 32 141 ST 2
08A8: 18 18 00 142 SET 8,(ST16-PTLP)
08AB: A8 143 ADD 8
08AC: 33 144 ST 3
08AD: E3 145 INR 3
08AE: 1C 50 00 146 SET 12,($50) *SW16 STACK
08B1: 0C 42 147 BS MV52
08B3: 00 148 MK22 RTN
08B4: A9 C0 149 MK01 LDA #$C0
150 *GET LIMITS & PACK PROGS
08B6: 85 33 151 STA PRMP
08B8: A9 00 152 LDA #0
08BA: 85 31 153 STA MODE
08BC: 20 67 FD 154 JSR GTLN
08BF: 86 16 155 STX XTAB
08C1: A0 00 156 LDY #0
08C3: B9 00 02 157 LDA BUFF,Y
08C6: C9 D3 158 CMP #$D3 S
08C8: F0 68 159 BEQ MK10
08CA: 20 A7 FF 160 MK06 JSR GTNM
08CD: C9 A7 161 CMP #$A7 F(.) (SEE MON.)
08CF: F0 10 162 BEQ MK02
08D1: 98 163 MERR TYA
08D2: AA 164 TAX
08D3: 20 4A F9 165 JSR PBL2 ERROR INDICATOR
08D6: A9 DE 166 LDA #$DE '!'
08D8: 20 ED FD 167 JSR COUT
08DB: 20 3A FF 168 JSR BELL
08DE: 18 169 MK05 CLC
08DF: 90 D3 170 BCC MK01
08E1: E6 31 171 MK02 INC MODE
08E3: 20 A7 FF 172 JSR GTNM
173 *A1 & A3 NOW HAVE 1ST #,A2 2D
174 *SET UP MOVE TO JUST BELOW (BBSL)
175 *AND LOWER BBSL
08E6: 20 89 F6 176 JSR SW16
08E9: 01 1E 177 BR SM02
08EB: 18 3C 00 178 MV51 SET 8,($3C)
08EE: 68 179 LDD @8
08EF: 32 180 ST 2
08F0: 68 181 LDD @8
08F1: 33 182 ST 3
08F2: B2 183 SUB 2

```



```

                                WOZPAK II
08F3: 38          185          ST      8
08F4: E3          186          INR     3
08F5: 83          187          MV52   POP    @3
08F6: 96          188          STP    @6
08F7: 23          189          LD     3
08F8: D2          190          CPR    2
08F9: 07 FA      191          BNZ    MV52
08FB: 28          192          LD     8
08FC: 33          193          ST     3
08FD: 18 08 00  194          SET    8,8
0900: 88          195          POP    @8
0901: 96          196          STP    @6
0902: 88          197          POP    @8
0903: 96          198          STP    @6
0904: 88          199          POP    @8
0905: 96          200          STP    @6
0906: 88          201          POP    @8
0907: 96          202          STP    @6
0908: 0B          203          RS
0909: 0C E0      204          SM02   BS     MV51
090B: 00          205          SM03   RTN
090C: C9 EC      206          MK09   CMP    #SEC F(S)
090E: F0 22      207          BEQ    MK10
0910: C9 C6      208          CMP    #C6 F(CR)
0912: F0 A0      209          BEQ    MK01
0914: C9 99      210          CMP    #99 F( )
0916: F0 03      211          BEQ    MK12
0918: D0 B7      212          BNE    MERR
091A: C8          213          MK11   INY
091B: B9 00 02  214          MK12   LDA    BUFF,Y
091E: C4 16      215          CPY    XTAB
0920: B0 92      216          BCS    MK01
0922: C9 A0      217          CMP    #A0 BLANK
0924: F0 F4      218          BEQ    MK11
0926: C9 8D      219          CMP    #8D
0928: F0 8A      220          BEQ    MK01
092A: C9 D3      221          CMP    #D3 S
092C: F0 04      222          BEQ    MK10
092E: C6 31      223          DEC    MODE
0930: F0 98      224          BEQ    MK06 ALWAYS
                225          *PACK 1ST PART & CLEAN UP
0932: 20 89 F6  226          MK10   JSR    SW16
0935: 21          227          LD     1
0936: 32          228          ST     2
0937: 18 52 01  229          SET    8,(PTLP-BASD)
093A: A8          230          ADD    8
093B: 37          231          ST     7
093C: 25          232          LD     5
093D: 77          233          STD    @7
093E: 29          234          LD     9
093F: 77          235          STD    @7
0940: 21          236          LD     1
0941: 77          237          STD    @7
0942: 27          238          LD     7
0943: 33          239          ST     3
0944: 0C AF      240          BS     MV52
0946: 66          241          SM04   LDD    @6
0947: 66          242          LDD    @6
0948: 00          243          RTN
0949: A5 0C      244          LDA    BPRG
    
```

WOZPAK II

```

094B: 85 CA 246 STA BBSL
094D: A5 0D 247 LDA BPRG+01
094F: 85 CB 248 STA BBSL+01
0951: 60 249 RTS
      250 *RESTORE LOOP
0952: 20 89 F6 251 PTLP JSR SW16
0955: 61 252 PLP0 LDD @1
0956: 33 253 ST 3
0957: 61 254 LDD @1
0958: 38 255 ST 8
0959: 00 256 RTN
095A: 20 89 F6 257 PLP1 JSR SW16
095D: 41 258 MV60 LD @1
095E: 53 259 ST @3
095F: F8 260 DCR 8
0960: 04 FB 261 BP MV60
0962: 21 262 LD 1
0963: D6 263 CPR 6
0964: 05 EF 264 BM PLP0
0966: 00 265 PLP2 RTN
0967: 4C 03 E0 266 JMP BSC2
      267 * OR JMP BRUN
096A: 00 268 ST16 HEX 00
    
```

--- END ASSEMBLY ---

TOTAL ERRORS: 00

363 BYTES OF OBJECT CODE
WERE GENERATED THIS ASSEMBLY.

APPLE][MACHINE CODE RELOCATION

BLANK PAGE

WOZPAK II

Quite frequently I have encountered situations calling for relocation of machine language (not BASIC) programs on my 6502-based APPLE][computer. Relocation means that the new version must run properly from different memory locations than the original. Because of the relative branch instruction, certain small 6502 programs need simply be moved and not altered. Others require only minor hand modification, which is simplified on the APPLE][by the built-in disassembler which pinpoints absolute memory reference instructions such as JMPs and JSRs. However, most of the situations which I have encountered involved rather lengthy programs containing multiple data segments interspersed with code. For example, I once spent over an hour to hand-relocate the 8K byte APPLE][monitor and BASIC to run from RAM addresses and at least one error probably went by undetected. That relocation can now be accomplished in a couple of minutes using the relocation program described herein.

The following situations call for program relocation:

- (1) Two programs which were written to run in identical locations must now reside and run in memory concurrently.
- (2) A program currently runs from ROM. In order to modify its operation experimentally, a version must be generated which runs from RAM (different addresses).
- (3) A program currently running in RAM must be converted to run from EPROM or ROM addresses.
- (4) A program currently running on a 16K machine must be relocated in order to run on a 4K machine. Furthermore, the relocation may have to be performed on the smaller machine.
- (5) Due to memory mapping differences, a program running on an APPLE-I (or other 6502-based computer falls into the unusable address space on an

APPLE][(or other) computer.

- (6) Due to operating system variable assignment differences either the page-zero or non-page-zero variable allocation for a specific program may have to be modified when moving the program from one make of computer to another.
- (7) A program exists as several chunks strewn around memory which must be combined in a single, contiguous block.
- (8) A program has outgrown the available memory space and must be relocated to a larger 'free' space.
- (9) A program insertion or deletion requires a chunk of the program to move a few bytes up or down.

PROGRAM MODEL

It is easy to visualize relocation as taking a program which resides and runs in a 'source block' of memory and creating a modified version in a 'destination block' which runs properly. This model dictates that the relocation must be performed in an environment in which the program may, in fact, reside in both blocks. In many cases the relocation is being performed because this is impossible. For example, a program written to begin at location \$400 on the APPLE-I (\$ stands for hex) falls in the APPLE][screen memory range. It must be loaded elsewhere on the APPLE][prior to relocation.

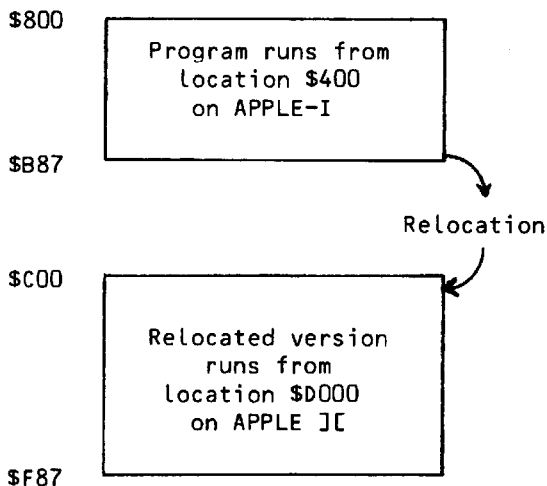
A more versatile program model is as follows. A program or section of a program runs in a memory range termed the 'source block' and resides in a range termed the 'source segments'. Thus, a program written to run at location \$400 may reside at location \$800. The program is to be relocated so that it will run in a range termed the 'destination

WOZPAK II

block' although it will reside in a range termed 'destination segments' (not necessarily the same). Thus a program may be relocated such that it will run from location \$D000 (a ROM address) yet reside beginning at location \$C00 prior to being saved on tape or used to burn EPROMs (obviously, the relocated program cannot immediately reside at locations reserved for ROM). In some cases the source and destination segments may overlap.

BLOCKS AND SEGMENTS EXAMPLE

Location
During
Relocation



SOURCE BLOCK: \$400-\$787

DEST BLOCK: \$D000-\$D387

SOURCE SEGMENTS: \$800-\$B87

DEST SEGMENTS: \$C00-\$F87

THE RELOCATION ALGORITHM

- (1) Set SOURCE PTR to beginning of source segment and DEST PTR to beginning of destination segment.
- (2) Copy 3 bytes from source segment (using SOURCE PTR) to temp INST area.
- (3) Determine instruction length from opcode (1,2, or 3 byte).
- (4) If two byte instruction with non-zero-page addressing mode (immediate or relative) then go to (7).
- (5) If two byte instruction then clear 3rd byte so address field is 0-255 (zero page).
- (6) If address field (2nd and 3rd bytes of INST area) falls within source block, then substitute.

ADR-SOURCE BLOCK BEGIN + DEST BLOCK BEGIN

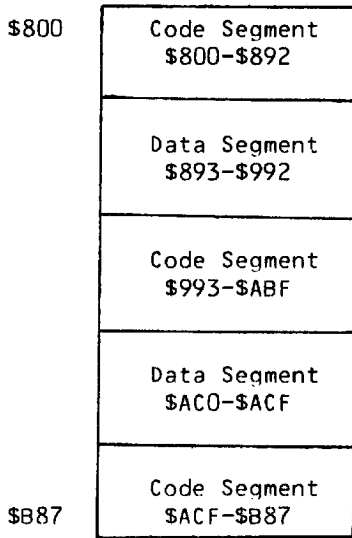
- (7) Move 'length' bytes from INST area to dest segment (using DEST PTR). Update SOURCE and DEST PTRs by length.
- (8) If SOURCE PTR is less than or equal to SOURCE SEGMENT END then goto (2), else done.

DATA SEGMENTS

The problem with relocating a large program all at once is that data (tables, text, etc.) may be interspersed throughout the code. Thus data may be 'relocated' as though it were code or might cause some code not to be relocated due to boundary uncertainty introduced when the data takes on the multi-byte attribute of code. This problem is circumvented by considering the 'source segments' and 'destination segments' sections to contain both code and data segments.

WOZPAK II

CODE AND DATA SEGMENTS EXAMPLE



The source code segments are relocated to the 'destination segments' area and the source data segments are moved. Note that several commands will be necessary to accomplish the complete relocation.

USAGE

1. Load RELOC by hand or off tape into memory locations \$3A6-\$3FA. note that locations \$3FB-\$3FF are not disturbed by tape load versions to insure that the APPLE][interrupt vectors are not clobbered. The Monitor user function, Control-Y, will now call RELOC as a subroutine at location \$3F8.
2. Load the source program into the 'source segments' area of memory if it is not already there. Note that this need not be where the program normally runs.
3. Specify the source and destination block parameters, remembering that the blocks are the locations that the program normally runs from, not the locations occupied by the source and destination segments during the relocation. If only a portion of a program is to be relocated then that portion alone is specified as the block.

* DEST BLOCK BEG < SOURCE

BLOCK BEG . END Yc *

Note that the syntax of this command closely resembles that of the MONITOR 'MOVE' command. The initial '*' is generated by the MONITOR, not typed by the user.

4. Move all data segments and relocate all code segments in sequential (increasing address) order.

First Segment (if CODE)

* DEST SEGMENT BEG < SOURCE
SEGMENT BEG . END Yc.

First Segment (if DATA)

* DEST SEGMENT BEG < SOURCE
SEGMENT BEG . END M

Subsequent segments (if CODE)

* SOURCE SEGMENT END Yc
(Relocation)

Subsequent segments (if DATA)

* SOURCE SEGMENT END M (Move)

Note that it is wise to prepare a list of segments (code and data) prior to relocation.

If the relocation is performed 'in place' (SOURCE and DEST SEGMENTS reside in identical locations) then the SOURCE SEGMENT BEG parameter may be omitted from the first segment relocate (or move).

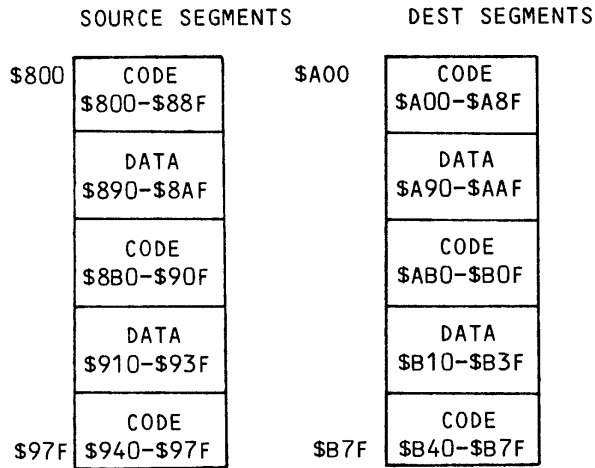
EXAMPLES

1. Straightforward Relocation

Program A resides and runs in locations \$800-\$97F. The relocated version will reside and run in locations \$A00-\$97F.

WOZPAK II

2. Index into block



SOURCE BLOCK \$800-\$97F

DEST BLOCK \$A00-\$B7F

SOURCE SEGMENTS \$800-\$97F

DEST SEGMENTS \$A00-\$B7F

(a) Load RELOC

(b) Define blocks

* A00 < 800 . 97F Yc *

(c) Relocate first segment (code).

* A00 < 800 . 88F Yc

(d) Move and relocate subsequent segments in order.

* . 8AF M (data)

* . 90F Yc (code)

* . 93F M (data)

* . 97F Yc (code)

Note that step (d) illustrates abbreviated versions of the following commands:

* A90 < 890 . 8AF M (data)

* AB0 < 8B0 . 90F Yc (code)

* B10 < 910 . 93F M (data)

* B40 < 940 . 97F Yc (code)

Assume that the program of example 1 uses an indexed reference into the data segment at \$890 as follows:

LDA 7B0,X

The X-REG is presumed to contain \$E0-\$FF. Because \$7B0 is outside the source block, it will not be relocated. This may be handled in one of two ways.

- (a) The exception is fixed by hand, or
- (b) The block specifications begin one page lower than the addresses at which the original and relocated programs begin to account for all such 'early references'. In step (b) of example (1) change to:

* 900 < 700 . 97F Yc*

Note that program references to the 'prior page' (in this case the \$7XX page) which are not intended to be relocated, will be.

3. Immediate Address References

Assume that the program of example (1) has an immediate reference which is an address. For example,

```
LDA #$3F
STA LOCO
LDA #$08
STA LOC1
JMP (LOCO)
```

In this example, the LDA #\$08 will not be changed during relocation and the user will have to hand-modify it to \$0A.

4. User function (Yc) programs

Relocating programs such as RELOC introduces another irregularity. Because RELOC uses the MONITOR user function command Control-Y (Yc), its entry point must remain fixed at \$3F8. The rest of RELOC may be relocated anywhere in memory (references other than the JMP at \$3F8). The user must leave the JMP at \$3F8 undisturbed or find some way other than Yc to pass parameters.

WOZPAK II

5. Unusable block ranges

A program was written to run from locations \$400-\$78F on an APPLE-I. A version which will run in ROM locations \$D000-\$D38F must be generated. The source (and destination) segments may reside in locations \$800-\$B8F on the APPLE II where relocation is performed.

SEGMENTS, SOURCE AND DEST

Locations during relocation

\$800	CODE \$800-\$97F
	DATA \$980-\$9FF
\$B8F	CODE \$A00-\$B8F

Runs from locations \$400-\$78F on APPLE-I but must be relocated to run from locations \$D000-\$D38F on the APPLE][.

SOURCE BLOCK \$400-\$78F

DEST BLOCK \$D000-\$D38F

SOURCE SEGMENTS \$800-\$B8F

DEST SEGMENTS \$800-\$B8F

(A) Load RELOC

(b) Load original program into locations \$800-\$B8F (despite the fact that it doesn't run there).

(c) Specify block parameters (i.e. where the original and relocated versions will run).

*D000 < 400 . 78F Yc *

(d) Move and relocate all segments in order.

* 800 < 800 . 97F Yc (first segment, code)

* . 9FF M (data)

* . B8F Yc (code)

Note that because the relocation is

done 'in place' the SOURCE SEGMENT BEG parameter is the same as the DEST SEG BEG parameter (\$800) and need not be specified. The initial segment relocation command may be abbreviated:

* 800 <. 97F Yc

6. The program of example (1) need not be relocated but the page zero variable allocation is from \$30 to \$3F. Because these locations are reserved for the APPLE][system monitor, the allocation must be changed to locations \$80-\$8F. The source and destination blocks are thus not the program but rather the variable area.

SOURCE BLOCK \$20-\$2F

DEST BLOCK \$80-\$8F

SOURCE SEGMENTS \$800-\$97F

DEST SEGMENTS \$800-\$97F

(a) Load RELOC

(b) Define blocks

*80 < 20.2F Yc *

(c) Relocate code segments and move data segments in place.

* 800 <.88F Yc (code)

* .8AF M (data)

* .90F Yc (code)

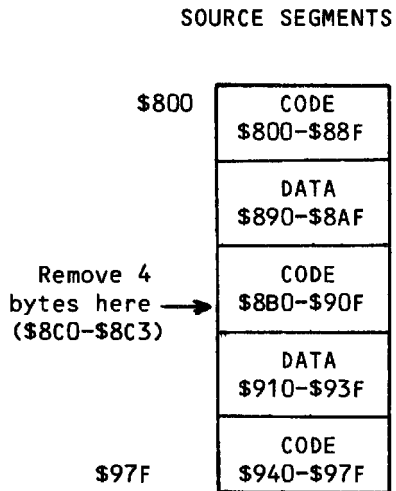
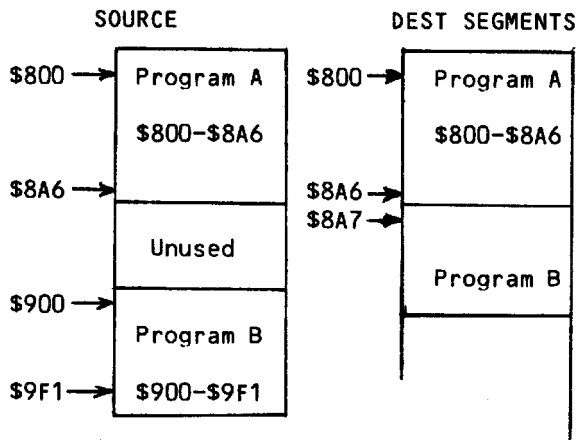
* .93F M (data)

* .97F Yc (code)

7. Split blocks with cross-referencing

Program A resides and runs in locations \$800-\$8A6. Program B resides and runs in locations \$900-\$9F1. A single, contiguous program is to be generated by moving program B so that it immediately follows program A. Each of the programs contains memory references within the other. It is assumed that the programs contain no data segments.

WOZPAK II



SOURCE BLOCK \$900-\$9F1

DEST BLOCK \$8A7-\$998

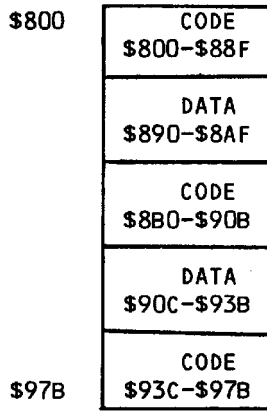
SOURCE SEGMENTS \$800-\$8A6 (A)
\$900-\$9F1 (B)

DEST SEGMENTS \$800-\$8A6 (A)
\$8A7-\$998 (B)

SOURCE BLOCK \$8C4-\$97F

SOURCE SEGMENTS \$800-\$88F (code)
\$890-\$8AF (data)
\$8B0-\$8BF (code)
\$8C4-\$90F (code)
\$910-\$93F (data)
\$940-\$97F (code)

DEST SEGMENTS



DEST BLOCK \$8C0-\$97B

DEST SEGMENTS \$800-\$88F (code)
\$890-\$8AF (data)
\$8B0-\$8BF (code)
\$8C0-\$90B (code)
\$90C-\$93B (data)
\$93C-\$97B (code)

(a) Load RELOC

(b) Define blocks (program B only)

* 8A7 < 900 . 9F1 Yc *

(c) Relocate each of the two programs individually. Program A must be relocated even though it does not move.

* 800 < . 8A6 Yc (program A, 'in place')

* 8A6 < 900 . 9F1 Yc (program B, not 'in place')

Note that any data segments within the two programs would necessitate additional relocation and move commands.

8. Code deletion.

4 bytes of code are to be removed from within a program and the program is to contract accordingly.

(a) Load RELOC

(b) Define blocks

* 8C0 < 8C4 . 97F Yc *

WOZPAK II

(c) Relocate code segments and move data segments in ascending address sequence.

- * 800 < . 88F Yc (code, 'in place')
- * . 8AF M (data)
- * . 8BF Yc (code)
- * 8C0 < 8C4 . 90F Yc (code, not 'in place')
- * . 93F M (data)
- * . 97F Yc (code)

(d) Relative branches crossing the deletion boundary will be incorrect since the relocation process does not modify them (only zero-page and absolute memory references). The user must patch these by hand.

9. Relocating the APPLE][MONITOR (\$F800-\$FFFF) to run in RAM (\$800-\$FFF).

SOURCE BLOCK \$F700-\$FFFF
(see example (2))

SOURCE SEGMENTS \$F800-\$F961 (code)
\$F962-\$FA42 (data)
\$FA43-\$FB18 (code)
\$FB19-\$FB1D (data)
\$FB1E-\$FFCB (code)
\$FFCC-\$FFFF (data)

DEST BLOCK \$700-\$FFF

DEST SEGMENTS \$800-\$961 (code)
\$962-\$A42 (data)
\$A43-\$B18 (code)
\$B19-\$B1D (data)
\$B1E-\$FCB (code)
\$FCC-\$FFF (data)

IMMEDIATE ADDRESS REFS (see example (3))

\$FFBF

\$FEA8

(more if not relocating to page boundary)

(a) Load RELOC

(b) Block parameters

* 700 < F700 . FFFF Yc *

(c) Segments

- * 800 < F800 . F961 Y (first segment, code)
- * . FA42 M (data)
- * . FB18 Y (code)
- * . FB1D M (data)
- * . FFCB Y (code)
- * . FFFF M (data)

(c) Immediate address references

- * FBF : E (was \$FE)
- * EA8 : E (was \$FE)

OTHER 6502 SYSTEMS

The following details illustrate features specific to the APPLE][which are used by RELOC. If adapted to other systems, the convenient and flexible parameter passing capability of the APPLE][monitor may be sacrificed.

1. The APPLE][monitor command
* A1 < A2 . A3 Yc
(A1, A2, and A3 are addresses)
vectors to location \$3F8 with the value A1 in locations \$3C (low) and \$3D (high), A2 in locations \$3E (low) and \$3F (high), and A3 in locations \$42 (low) and \$43 (high). Locaton \$34 (YSAV) holds an index to the next character of the command buffer (after the Yc). The command buffer (IN) begins at \$200.

2. If Yc is not followed by an '*' then the block parameters are simply preserved as follows:

Parameter	Preserved at	SWEET16 Reg Name
DEST BLOCK BEG	\$8, \$9	TOBEG
SOURCE BLOCK BEG	\$2, \$3	FRMBEG
SOURCE BLOCK END	\$4, \$5	FRMEND

WOZPAK II

3. If Yc is not followed by an 'A', then a segment relocation is initiated at RELOC2 (\$3BB). Throughout, A1 (\$3C, \$3D) is the source segment pointer and A4 (\$42, \$43) is the destination segment pointer.
4. INSDS2 is an APPLE][monitor subroutine which determines the length of a 6502 instruction in the variable LENGTH (location \$2F) given the opcode in the A-REG.
5. The code from XLATE TO SW16RT (\$3D9-\$3E6) uses the APPLE][16-bit interpretive machine, SWEET16. The target address of the 6502 instruction being relocated (locations \$C low and \$D high) occupies the SWEET16 register named ADR. If ADR is between FRMBEG and FRMEND (inclusive) then it is replaced by ADR-FRMBEG + TOBEG.
6. NXTA4 is an APPLE][monitor subroutine which increments A1 (source segment index) and A4 (destination segment index). If A1 exceeds A2 (source segment end) then the carry is set, otherwise it is cleared.

Instruction type	LENGTH
Invalid	0
1 byte	0
2 byte	1
3 byte	2

NOTE: MACHINE CODE RELOCATION was originally written to run at \$3A6 to \$3FA. Because this wipes out DOS entry points, the routine is reassembled at \$1000. The first part of the routine (lines 54-60 of the assembly listing) writes the CONTROL-Y vector at \$3F8. '1000G' entered at the start of using RELOC will write the vector to allow the routine to operate properly.

WOZPAK II

```

3 *****
4 *
5 * 6502 RELOCATION *
6 * SUBROUTINE *
7 *
8 * 1. DEFINE BLOCKS *
9 * *A4<A1.A2 ^Y *
10 * (^Y IS CTRL-Y) *
11 *
12 * 2. FIRST SEGMENT *
13 * *A4<A1.A2 ^Y *
14 * (IF CODE) *
15 *
16 * *A4<A1.A2 M *
17 * (IF MOVE) *
18 *
19 * 3. SUBSEQUENT SEGS *
20 * *.A2 ^Y OR *.A2 M *
21 *
22 * WOZ 11-10-77 *
23 * APPLE COMPUTER CO.INC. *
24 *
25 * (SLIGHTLY MODIFIED BY) *
26 * KEN SMITH TACOMA,WA *
27 *
28 *****
29 *
30 *RELOCATION SUBROUTINE EQUATES
31 *
32 R1L EQU $2 ;SWEET16 REG. 1.
33 INST EQU $B ;3-BYTE INST. FIELD
34 LENGTH EQU $2F ;LENGTH CODE.
35 YSAV EQU $34 ;CMND BUF POINTER.
36 A1L EQU $3C ;APPLE ][ MON PARAM AREA.
37 A4L EQU $42 ;APPLE ][ MON PARAM REG 4.
38 IN EQU $200 ;MON CMND BUF.
39 CTRLY EQU $3F8 ;CONTROL-Y VECTOR ADR.
40 SW16 EQU $F689 ;SWEET16 ENTRY.
41 INSDS2 EQU $F88E ;DISASSEMBLER ENTRY.
42 NXTA4 EQU $FCB4 ;POINTER INCR SUBR.
43 FRMBEG EQU $1 ;SOURCE BLOCK BEGIN.
44 FRMEND EQU $2 ;SOURCE BLOCK END.
45 TOBEG EQU $4 ;DEST BLOCK BEGIN.
46 ADR EQU $6 ;ADR PART OF INST.
47 OBJ $7000
48 ORG $1000
49 *
*****
51 * CONTROL-Y 'JMP' TO RELOC SETUP
*****
53 *
1000: A9 4C 54 INITY LDA #$4C ;'JMP' FOR CONTROL-Y ENTRY.
1002: 8D F8 03 55 STA CTRLY
1005: A9 10 56 LDA #<RELOC ;GET ROUTINE ADDRESS AND
1007: 8D F9 03 57 STA CTRLY+1 ;WRITE INTO $3F9&A FOR
100A: A9 10 58 LDA #>RELOC ;CONTROL-Y JMP DESTINATION.
100C: 8D FA 03 59 STA CTRLY+2
100F: 60 60 RTS

```

WOZPAK II

```

*****
* 6502 RELOCATION SUBROUTINE
*****
*
63
*
65
1010: A4 34 66 RELOC LDY YSAV ;CMND BUF POINTER.
1012: B9 00 02 67 LDA IN,Y ;NEXT CMND CHAR.
1015: C9 AA 68 CMP #$AA ;'*'?
1017: D0 0C 69 BNE RELOC2 ;NO, RELOC CODE SEG.
1019: E6 34 70 INC YSAV ;ADVANCE POINTER.
101B: A2 07 71 LDX #$7
101D: B5 3C 72 INIT LDA A1L,X ;MOVE BLOCK PARAMS
101F: 95 02 73 STA R1L,X ;FROM APPLE ][ MON
1021: CA 74 DEX . ;AREA TO SW16 AREA.
1022: 10 F9 75 BPL INIT ;R1=SOURCE BEG, R2=
1024: 60 76 RTS . ;SOURCE END, R4=DEST BEG.
1025: A0 02 77 RELOC2 LDY #$2
1027: B1 3C 78 GETINS LDA (A1L),Y ;COPY 3 BYTES TO
1029: 99 0B 00 79 STA INST,Y ;SW16 AREA.
102C: 88 80 DEY
102D: 10 F8 81 BPL GETINS
102F: 20 8E F8 82 JSR INSDS2 ;CALCULATE LENGTH OF
1032: A6 2F 83 LDX LENGTH ;INST FROM OPCODE.
1034: CA 84 DEX . ;0=1 BYTE, 1=2 BYTE,
1035: D0 0C 85 BNE XLATE ;2=3 BYTE.
1037: A5 0B 86 LDA INST
1039: 29 0D 87 AND #$D ;WEED OUT NON ZERO-PAGE
103B: F0 14 88 BEQ STINST ;2 BYTE INSTS. (IMMED)
103D: 29 08 89 AND #$8 ;IF ZERO-PAGE ADR
103F: D0 10 90 BNE STINST ;THEN CLEAR HIGH BYTE.
1041: 85 0D 91 STA INST+2
1043: 20 89 F6 92 XLATE JSR SW16 ;IF ADR OF ZERO-PAGE
1046: 22 93 LD FRMEND ;OR ABS IS IN SOURCE
1047: D6 94 CPR ADR ;(FRM) BLOCK THEN
1048: 02 06 95 BNC SW16RT ;SUBSTITUTE ADR-
104A: 26 96 LD ADR ;SOURCE BEG+DEST BEG.
104B: B1 97 SUB FRMBEG
104C: 02 02 98 BNC SW16RT
104E: A4 99 ADD TOBEG
104F: 36 100 ST ADR
1050: 00 101 SW16RT RTN
1051: A2 00 102 STINST LDX #$0
1053: B5 0B 103 STINS2 LDA INST,X
1055: 91 42 104 STA (A4L),Y ;COPY LENGTH BYTES
1057: E8 105 INX . ;OF INST FROM
1058: 20 B4 FC 106 JSR NXTA4 ;SW16 AREA TO
105B: C6 2F 107 DEC LENGTH ;DEST SEGMENT. UPDATE
105D: 10 F4 108 BPL STINS2 ;SOURCE, DEST SEGMENT
105F: 90 C4 109 BCC RELOC2 ;POINTERS. LOOP IF NOT
1061: 60 110 RTS . ;BEYOND SOURCE SEG END.

```

--- END ASSEMBLY ---

TOTAL ERRORS: 0

98 BYTES GENERATED THIS ASSEMBLY

APPLE][TAPE VERIFY ROUTINE
for Integer Basic
and Machine Language Programs

WOZPAK II

This routine provides a method of checking programs saved on tape against a program in memory. Errors can occur due to a number of reasons, including defects in manufacture of the tape or improper operation of the tape recorder. With this routine, a program can be checked immediately after it is saved to verify that a good SAVE was accomplished. The TAPE VERIFY routine will read the program saved on tape and check it byte for byte against the program in memory.

The routine is assembled to run at memory location \$1000, so that there is a minimum of interference with other routines that may need to be verified. Thus, it does not destroy DOS, either HIRES screen, HIRES routines, or most other routines.

TO VERIFY AN INTEGER BASIC PROGRAM:

Make sure that the program to be verified is in memory.
'CALL 4096' (do not hit [RETURN] yet)
Start tape in 'Play' mode.
Hit [RETURN].

TO VERIFY MACHINE LANGUAGE PROGRAM:

Enter Monitor (CALL -151)
Type '1076G [RETURN]. This activates 'CONTROL-Y'.
Enter the range to be verified, I.E. 'C00.C85 (CONTROL-Y)'.
Start tape in 'Play' mode.
Hit [RETURN]

If no errors are detected, the APPLE][will 'BEEP' and the appropriate prompt character will reappear at the end of the verify.

Should an error be found, the memory location which does not agree with the tape will be displayed. Next the byte in memory will be displayed, and then the byte which was read from the tape will be shown in parenthesis.

NOTE: The routine will return control to the user after the first error is found. After the corrections is made, the routine should be rerun to check for any other errors.

To observe the operation of the routine, first call the routine and verify a tape against an unchanged program. Then, change ONE character in the program. Call the routine, and it will find and display the non-matching byte.

TO RELOCATE THE ROUTINE:

Change the JSR addresses in lines 52 and 55 of the assembled listing. Also change the high and low address bytes that are written into the CONTROL-Y vector (\$3F9-\$3FA) as written in lines 104 and 106 of the assembly listing.

WOZPAK II

```

2 *****
3 *
4 *      APPLE ][ TAPE VERIFY ROUTINE      *
5 *
6 * APPLE PUGET SOUND PROGRAM LIBRARY EXCHANGE *
7 * 6708 39TH AVE. SW      SEATTLE, WA 98138 *
8 *
9 *      SOURCE CODE FROM 'THE WOZPAK'      *
10 * COURTESY OF S. WOZNAK  APPLE COMPUTER CO. *
11 *
12 *      DECIPHERED AND ASSEMBLED          *
13 *      (AND SLIGHTLY MODIFIED BY)       *
14 *      KEN SMITH  TACOMA, WA  JUNE 14,1979 *
15 *
16 *****
17 *
18 CHKSUM  EQU  $2E
19 A1L     EQU  $3C
20 HIMEMH EQU  $4D
21 XSAVE  EQU  $D8
22 PPH    EQU  $CB
23 LSTOR  EQU  $CF
24 *
25 BASHDR EQU  $F11E
26 BASREAD EQU $F12C
27 HEADR  EQU  $FCC9
28 RDBYTE EQU  $FCEC
29 RDBIT  EQU  $FCFD
30 RD2BIT EQU  $FCFA
31 *
32 NXTA1  EQU  $FCBA
33 CHKSUMOK EQU $FF26
34 PRERR  EQU  $FF2D
35 PRA1   EQU  $FD92
36 COUT   EQU  $FDED
37 PRBYTE EQU  $FDDA
38 *
39      ORG  $1000
40 *
*****
1000: 86 D8 42 VFYBSC STX  XSAVE      PRESERVE X-REG
1002: 38    43          SEC
1003: A2 FF 44          LDX  #$FF
1005: B5 4D 45 GETLEN LDA  HIMEMH,X  CALC PROGRAM LENGTH
1007: F5 CB 46          SBC  PPH,X    AND STORE ($CE,$CF)
1009: 95 CF 47          STA  LSTOR,X
100B: E8    48          INX
100C: F0 F7 49          BEQ  GETLEN
100E: 20 1E F1 51 *
1011: 20 1F 10 52          JSR  BASHDR  SET ADDRESSES FOR BASIC
1014: A2 01 53          JSR  TAPEVFY  HEADER READ
1016: 20 2C F1 54          LDX  #$01
1019: 20 1F 10 55          JSR  BASREAD  SET ADDRESSES FOR BASIC
101C: A6 D8 56          JSR  TAPEVFY  PROGRAM READ
101E: 60    57          LDX  XSAVE
101F: 20 FA FC 59          RTS
1022: A9 16 60 TAPEVFY JSR  RD2BIT
1024: 20 C9 FC 61          LDA  #$16    SYNCHRONIZE ON
                          JSR  HEADR    TAPE HEADER

```

WOZPAK II

```

1027: 85 2E 63          STA  CHKSUM
1029: 20 FA FC 64      JSR  RD2BIT
102C: A0 24 65      VRFY2 LDY  #$24
102E: 20 FD FC 66      JSR  RDBIT
1031: B0 F9 67          BCS  VRFY2
1033: 20 FD FC 68      JSR  RDBIT
1036: A0 3B 69      LDY  #$3B
1038: 20 EC FC 70      VRFY3 JSR  RDBYTE  READ A BYTE FROM TAPE
103B: F0 0E 71      BEQ  EXTDEL  (ALWAYS TAKEN)
103D: 45 2E 72      CKSUM EOR  CHKSUM  UPDATE RUNNING CHECKSUM
103F: 85 2E 73          STA  CHKSUM  INC A1, COMPARE TO
1041: 20 BA FC 74      JSR  NXTA1  A2 (CARRY SET IF >=)
1044: A0 34 75      LDY  #$34
1046: 90 F0 76          BCC  VRFY3  LOOP UNTIL A1>A2
1048: 4C 26 FF 77      JMP  CHKSUMOK SOUND BELL AFTER CKSUM VFY
                                78
                                *
104B: EA 79      EXTDEL NOP  .      EXTRA DELAY TO
104C: EA 80      NOP  .      EQUALIZE TIMING
104D: EA 81      NOP  .      (12 USEC)
                                *
104E: C1 3C 83      CMP  (A1L,X)
1050: F0 EB 84      BEQ  CKSUM  BYTE MATCHES!
*****
1052: 48 86          PHA
1053: 20 2D FF 87      JSR  PRERR  OUTPUT 'ERR'
1056: 20 92 FD 88      JSR  PRA1  OUTPUT '(A1)-'
1059: B1 3C 89          LDA  (A1L),Y
105B: 20 DA FD 90      JSR  PRBYTE OUTPUT CONTENTS OF A1
105E: A9 A0 91      LDA  #$A0
1060: 20 ED FD 92      JSR  COUT  OUTPUT A SPACE
1063: A9 A8 93      LDA  #$A8
1065: 20 ED FD 94      JSR  COUT  OUTPUT '('
1068: 68 95          PLA
1069: 20 DA FD 96      JSR  PRBYTE OUTPUT BYTE FROM TAPE
106C: A9 A9 97      LDA  #$A9
106E: 20 ED FD 98      JSR  COUT  OUTPUT ')'
1071: A9 8D 99      LDA  #$8D
1073: 4C ED FD 100     JMP  COUT  OUTPUT CAR RTN AND RETURN
*****
1076: A9 4C 102     CTLYINIT LDA  #$4C  INITIALIZE CONTROL
1078: 8D F8 03 103     STA  $3F8  'Y' ENTRY FOR USE WITH
107B: A9 1F 104     LDA  #<TAPEVFY MACHINE LANGUAGE
107D: 8D F9 03 105     STA  $3F9  ROUTINES
1080: A9 10 106     LDA  #>TAPEVFY
1082: 8D FA 03 107     STA  $3FA
1085: 60 108     RTS

```

APPLE][HI-RES GRAPHICS SUBROUTINES

WOZPAK II

The APPLE][computer comes with a high-resolution color graphics display mode of 280 horizontal by 192 vertical resolution. Because 8K Bytes of RAM are dedicated solely to maintaining the HI-RES display, a minimum 12K system (configured for HI-RES) is required to use this mode. For practical reasons, 16K Bytes is the strongly recommended minimum. A 6502 machine language subroutine package has been developed to simplify efficient use of the APPLE][HI-RES display for assembly language and BASIC programmers. The routines for initializing the HI-RES display, plotting points, drawing lines, and drawing shapes are described herein.

USING THE HI-RES SUBROUTINES

Despite the fact that HI-RES graphics commands are not built into APPLE][BASIC, a convenient scheme for referencing the subroutines and their parameters by name has been devised.

The first statement of a program using the HI-RES subroutines should be as follows:

```
0 X0=Y0=COLR=SHAPE=ROT=SCALE
```

The purpose of this statement is to enter the first 6 BASIC variable names into the variable table in a fixed sequence. When executed, each of the 6 parameters will be assigned storage at fixed locations relative to the address contained in the BASIC 'start of variables' pointer (LOMEM), making them readily accessible by the HI-RES subroutines.

Different parameter names may be used provided that they retain the same number of characters. This is necessary to insure that the storage locations for each, relative to LOMEM, do not change. For example, the name XX could be used in place of X0, but XCOORD could not.

The parameters SHAPE, ROT, and SCALE are used only by the HI-RES shape draw subroutines and may be omitted from programs using only the PLOT and LINE features. Omitting unnecessary variable definitions is one method of enhancing the overall performance of some BASIC programs on the APPLE][and is thus desirable.

FIRST LINE OF PROGRAMS NOT USING THE SHPAE DRAW SUBRROUTINES

```
0 X0=Y0=COLR
62
```

After the parameter names have been defined, the HI-RES subroutine names themselves may be defined and assigned corresponding subroutine entry addresses as values. Calling subroutines by name is preferable to calling them by entry address because the entry addresses may vary in future versions of the HI-RES subroutines, and names are better self documenting.

Absolute CALL	CALL by name
.	5 INIT=2048
.	.
.	.
100 CALL 2048	100 CALL INIT
.	.
.	.
200 CALL 2048	200 CALL INIT

In the above CALL by name example, should the INIT subroutine entry address change to -12288, only line 5 need be changed. In the absolute call example, lines 100 and 200 (and any others referencing the INIT subroutine) will have to be changed. The self documenting advantage of the CALL by name example should be apparent.

The following statement lists all HI-RES subroutine entry initializations available to BASIC programs. Other names may be used.

```
5 INIT=2048: CLEAR=2062: BKGND=2865:
  POSN=2809: PLOT=2830: LINE=2836:
  DRAW=2871: DRAW1=2874: XDRAW=2884:
  XDRAW1=2887: FIND=2556
```

The allowable color specification values may also be referenced by name, if the initialization statement below is included in your program. Note that GREEN is preceded by LET to avoid a syntax error due to confusion with the 'GR' command.

```
7 BLACK=0: LET GREEN=42: VIOLET=85:
  WHITE=127
```

If your APPLE][has been modified for additional HI-RES colors, the following assignments are also valid.

```
8 ORANGE=170: BLUE=213: BLACK2=128:
  WHITE2=255
```

Unnecessary variable definitions in the program should be avoided as they will slow some programs. Therefore, a program

WOZPAK II

should not define VIOLET=85 unless it uses the color VIOLET. The example given below illustrates condensed initialization statements for a program using only the INIT, PLOT, and DRAW subroutines, and the colors GREEN and WHITE.

```
0 X0=Y0=COLR=SHAPE=ROT=SCALE
5 INIT=2048:PLOT=2830:DRAW=2871
7 LET GREEN=42:WHITE=127
```

In extreme cases, any of the following techniques will further enhance program performance.

- (1) Omit the color and subroutine name initializations. Refer to colors and subroutines by value, not name. This does not apply to the parameter references.
 - (2) Define the most frequently used program variable names PRIOR to the subroutine and color name initializations (Lines 5 & 7 in the prior examples). The example below will speed up programs extensively referencing variables I, J, and K.
- ```
0 X0=Y0=COLR=SHAPE=ROT=SCALE
2 I=J=K
5 INIT=2048:CLEAR=2062:BKGND=2865:
 POSN=2809.....etc.
7 BLACK=0:LET GREEN=42:.....etc.
```
- (3) Use the parameter names as program variables when possible, the parameters are the most quickly accessed BASIC variables.

INITIALIZATION SUBROUTINES

The normal HI-RES display consists of a 280 horizontal by 160 vertical grid above 4 lines of text and is initiated with the BASIC command:

```
> CALL INIT
```

The INIT subroutine also clears the HI-RES display and initializes other HI-RES subroutines. After calling INIT, the programmer may eliminate the 4 line text display, extending the HI-RES display to a 192 line vertical resolution, with the following command:

```
> POKE -16302,0
```

The 4-line text display may be re-

stored at any time as follows:

```
> POKE -16301,0
```

Valid X-coordinates vary from 0 (leftmost) to 279 (rightmost). Valid Y-coordinates vary from 0 (topmost) to 159 or 191 (bottommost), depending on whether or not the 4 line text display is enabled.

At any time after INIT is called, the entire HI-RES display may be cleared with the CLEAR subroutine as shown below.

```
> CALL CLEAR
```

The HI-RES display may be quickly set to any background color with the BKGND subroutine. BKGND expects a color specification in the BASIC variable COLR. The example below turns the entire HI-RES display green.

```
0 X0=Y0=COLR
5 INIT=2048:BKGND=2865:LET GREEN=42
10 CALL INIT
20 COLR=GREEN
30 CALL BKGND
40 END
```

Only the colors previously mentioned (BLACK, GREEN, VIOLET, and WHITE)\* may be specified in COLR. Do not make up your own. For example, COLR=YELLOW is not allowed.

If COLR is greater than 255 when BKGND is called, a range error will occur. The message "\*\*\*RANGE ERR" will be displayed and the program will halt.

POINTS AND LINES

The PLOT subroutine is used to plot a single point of the HI-RES display in a specified color. COLR must be less than 255, X0 must be 0 to 279, Y0 must be 0 to 191 when PLOT is called or a range error will result and the program will halt. The program below plots one white dot at X-coordinate 35, and Y-coordinate 55 (35, 55) and one at (85,90).

```
0 X0=Y0=COLR
5 INIT=2048:PLOT=2380:WHITE=127
10 CALL INIT
20 COLR=WHITE
30 X0=35:Y0=55:CALL PLOT
40 X0=85:Y0=90:CALL PLOT
50 END
```

## WOZPAK II

Connecting any two coordinates with a straight line is almost as easy as plotting points. After plotting one end point as shown in the example above, the other end point is specified in X0 and Y0 and the LINE subroutine is called. As with the PLOT subroutine, COLR must be less than 256, X0 must be 0 to 279, and Y0 must be 0 to 191 or a range error will result and the program will halt. The following example draws a white line from (35,40) to (170,100), a green line from (270,10) to (5,145), and a violet line from (20,70) to (190,110).

```

0 X0=Y0=COLR
5 INIT=2048:PLOT=2830:LINE=2836:
 LET GREEN=42:VIOLET=85:WHITE=127
10 CALL INIT
20 COLR=WHITE:X0=35:Y0=40:CALL PLOT
25 X0=170:Y0=100:CALL LINE
30 COLR=GREEN:X0=270:Y0=10:CALL PLOT
35 X0=5:Y0=145:CALL LINE
40 COLR=VIOLET:X0=20:Y0=70:CALL PLOT
45 X0=190:Y0=110:CALL LINE
50 END

```

The following example illustrates that the parameter variables may be used as FOR loop indices. Horizontal violet lines are drawn on a green background at every tenth vertical coordinate.

```

0 X0=Y0=COLR
5 INIT=2048:BKGND=2865:PLOT=2830:
 LINE=2836:LET GREEN=42:VIOLET=85
10 CALL INIT
20 COLR=GREEN:CALL BKGND
30 COLR=VIOLET
40 FOR Y0=5 TO 155 STEP 10
50 X0=10:CALL PLOT:X0=270:CALL LINE
60 NEXT Y0:END

```

Multiple lines which are connected endpoint to endpoint may be drawn without intervening PLOT calls. In the example below, a white line connects (10,20) to (250,70), and a green line connects (250,70) to (20,150), and a violet line connects (20,150) to (260,30).

```

0 X0=Y0=COLR
5 INIT=2048:PLOT=2830:LINE=2836:
 LET GREEN=42:VIOLET=85:WHITE=127
10 CALL INIT
20 COLR=WHITE:X0=10:Y0=20:CALL PLOT
30 X0=250:Y0=70:CALL LINE
40 X0=20:Y0=150:COLR=GREEN:CALL LINE
50 X0=260:Y0=30:COLR=VIOLET:CALL LINE
60 END
64

```

## CAUTION

Do not attempt to draw a line prior to the first PLOT. Because the first end-point has not been defined, the line may be drawn in random memory locations, not necessarily restricted to the screen memory.

## DRAWING SHAPES

Up to 255 different shapes may be defined, edited, and saved with the SHAPE-GEN program, available from APPLE. (See HIRES Shape Generator program-next sect.) After loading the HI-RES subroutines such a 'shape tape' (containing a shape table) may be read as follows:

1. Position shape tape in recorder.
2. Load shape tape with the BASIC COMMAND: >CALL 3001
3. Start the recorder (PLAY). The above command immediately begins reading tape.
4. Wait for two beeps.

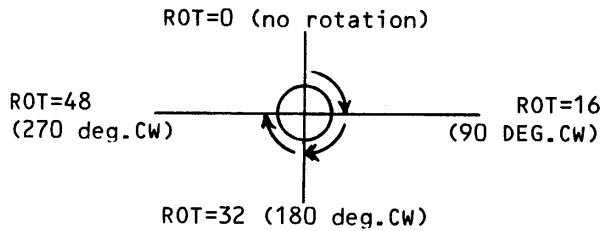
Shape tables always load at address \$C00 with the HI-RES subroutines located at \$800-\$BFF. Upon loading a shape table, the BASIC 'start of variables' pointer (LOMEM) is set to contain the address of the location immediately following the last shape table byte.

If not enough free memory is available to contain the shape table then the message '\*\*\*MEM FULL ERR' will be given. If no beep is heard when loading a shape table then something is probably wrong with the tape connection and you will have to hit RESET and re-enter BASIC. If you hear a single beep and then the system hangs, it means your shape tape is probably bad, and after hitting RESET and re-entering BASIC, you may have to restore the LOMEM setting to \$C00 (3072) with the command: >LOMEM:3072.

The DRAW subroutine is used to display any of the predefined shapes included in the current shape table. The origin, or 'beginning point', of the shape is specified in X0 and Y0, and the color is specified in COLR, as with PLOT. The shape number desired is specified in SHAPE. For example, SHAPE=3 specifies that the third shape of the current shape table is to be drawn. A scale factor is specified in the variable SCALE and a rotation in ROT. A scale factor of 4 implies a shape 4 times the defined size.

WOZPAK II

A scale factor of 0 is always interpreted as 256.



COLR must be 0 to 255, X0 must be 0 to 279, Y0 must be 0 to 191, ROT must be 0 to 255 (due to MOD 64 arithmetic, ROT=64 is equivalent to ROT=0), SCALE must be 0 to 255, and SHAPE must be greater than zero and less than or equal to the current number of shape table shapes, or else a range error will result when DRAW is called and the program will halt. In other words, the programmer will always be notified if HI-RES subroutines are called with any invalid parameters.

The following program example draws shape number 3 in white at a 90 degree clockwise rotation and a scale factor of 2. The origin is at (140,80). It is assumed that a shape table with at least 3 shape definitions has been loaded.

```

0 X0=Y0=COLR=SHAPE=ROT=SCALE
5 INIT=2048:DRAW=2871
7 WHITE=127
10 CALL INIT
20 X0=140:Y0=80:COLR=WHITE
30 SHAPE=3:ROT=16:SCALE=2
40 CALL DRAW
50 END

```

The XDRAW subroutine is identical in operation to the DRAW subroutine except that the defined shape is exclusive-ORed onto the screen. The EX-OR operation complements all screen memory bits of the shape, 0's become 1's and vice-versa. No color need be specified. A unique property of XDRAW is that 2 successive calls with identical parameters will first cause a shape to be drawn (in white) and then erased. The following program example causes the rotation of shape number 3 to track paddle 0. XDRAW is used for both the draw and erase operations. Although the color is optional, the variable COLR may not be omitted from the parameter declarations (line 0), or the SHAPE, ROT, and SCALE parameters will not be assigned storage in their standard locations relative to LOMEM.

```

0 X0=Y0=COLR=SHAPE=ROT=SCALE
5 INIT=2048:XDRAW=2884
10 CALL INIT
20 X0=140:Y0=80:SHAPE=3:SCALE=2
30 R=0:GOTO 60:REM DRAW FIRST SHAPE
40 R=PDL(0):IF R=ROT THEN GOTO 30
50 CALL XDRAW:REM ERASE AT OLD ROT
60 ROT=R:CALL XDRAW:REM DRAW AT NEW ROT
70 GOTO 40:REM CHECK FOR ROT CHANGE
80 END

```

DRAW1 and XDRAW1 are identical to DRAW and XDRAW respectively, except that the most recently plotted (or drawn) point serves as the shape origin and the current color is not updated. Thus, X0, Y0, and COLR are not specified.

If you draw a shape and then wish to draw a line from the final plot position of that shape to a fixed coordinate, you may do so. After drawing the shape, you must call FIND prior to calling LINE. The FIND subroutine determines the X-Y coordinates of the final shape plot position (or current plot position if used after other subroutines) and uses it as the beginning endpoint of the following call to LINE. The following program example draws a shape and then a violet line from the final plot position of the shape to (10,25).

```

0 X0=Y0=COLR=SHAPE=ROT=SCALE
5 INIT=2048:LINE=2836:DRAW=2871:
 FIND=2556
7 VIOLET=85:WHITE=127
10 X0=140:Y0=80:COLR=WHITE:SHAPE=3
 ROT=0:SCALE=1:CALL DRAW
20 CALL FIND
30 X0=10:Y0=25:COLR=VIOLET:CALL LINE
40 END

```

COLLISIONS

Overlapping shapes define points of 'collision'. The DRAW and XDRAW subroutines return a collision count in the absolute location \$32A (810 decimal). The collision count will be constant for a fixed shape, rotation, scale, and background, provided that no collisions with other shapes are detected. The difference between the 'standard' collision value and the encountered value (while drawing a shape) is a true collision indicator.

```

100 CALL DRAW
110 COUNT=PEEK (810)

```

WOZPAK II

The HI-RES subroutines may be appended to an INTEGER BASIC program, using the PACK&LOAD routine that appears elsewhere in this book, making a two-step loading unnecessary.

SUMMARY

| SUBROUTINE | CALL-ADDRESS | PARAMETERS                        |
|------------|--------------|-----------------------------------|
| INIT       | 2048         |                                   |
| CLEAR      | 2062         |                                   |
| BKGND      | 2865         | COLR                              |
| POSN       | 2809         | XO, YO, COLR                      |
| PLOT       | 2830         | XO, YO, COLR                      |
| LINE       | 2836         | XO, YO, COLR                      |
| DRAW       | 2871         | XO, YO, COLR<br>SHAPE, ROT, SCALE |
| DRAW1      | 2874         | SHAPE, ROT, SCALE                 |
| XDRAW      | 2884         | XO, YO, COLR<br>SHAPE, ROT, SCALE |
| XDRAW1     | 2887         | SHAPE, ROT, SCALE                 |
| FIND       | 2556         |                                   |
| SHAPE LOAD | 3001         |                                   |

For NO TEXT display-----POKE -16302,0  
 For mixed GRAPHICS/TEXT-----POKE -16301,0  
 Select secondary screen display-----  
 -----POKE -16299,0  
 Select primary screen display-----  
 -----POKE -16300,0  
 Select secondary screen plotting-----  
 -----POKE 806,64  
 Select primary screen plotting-----  
 -----POKE 806,32  
 (Defaults are: GRAPHICS/TEXT, primary  
 screen display, and primary screen  
 plotting)  
 Collision Detect (shape draw only)-----  
 -----PEEK (810)

HIGH-RES SUBROUTINES SEGMENT MAP

|      |             |
|------|-------------|
| CODE | \$800-\$9E8 |
| DATA | \$9E9-\$9FB |
| CODE | \$9FC-\$BFF |

HI-RES PARAMETER LOCATIONS  
(beyond LOMEM)

| PARAMETER | Locations beyond LOMEM |
|-----------|------------------------|
| XO        | \$05, \$06             |
| YO        | \$0C, \$0D             |
| COLR      | \$15, \$16             |
| SHAPE     | \$1F, \$20             |
| ROT       | \$27, \$28             |
| SCALE     | \$31, \$32             |

Note: Each parameter is two bytes in length. The low-order byte is stored in the lesser of the two locations assigned.

HCOLOR

| COLOR  | EVEN HNDX       | ODD HNDX        |
|--------|-----------------|-----------------|
| BLACK  | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| GREEN  | 0 0 1 0 1 0 1 0 | 0 1 0 1 0 1 0 1 |
| VIOLET | 0 1 0 1 0 1 0 1 | 0 0 1 0 1 0 1 0 |
| WHITE  | 0 1 1 1 1 1 1 1 | 0 1 1 1 1 1 1 1 |
| BLACK2 | 1 0 0 0 0 0 0 0 | 1 0 0 0 0 0 0 0 |
| ORANGE | 1 0 1 0 1 0 1 0 | 1 1 0 1 0 1 0 1 |
| BLUE   | 1 1 0 1 0 1 0 1 | 1 0 1 0 1 0 1 0 |
| WHITE2 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 |

HI-RES INTERNAL VARIABLES

SHAPEL, SHAPEH (\$1A, \$1B) On-the-fly shape pointer.  
 HCOLOR1 (\$1C) on-the-fly color byte.  
 COUNTH (\$1D) High-order byte of step count for LINE.  
 HBASL, HBASH (\$26, \$27) On-the-fly base address.  
 HMASK (\$30) On-the-fly BIT MASK.  
 QDRNT (\$53) 2 LSB's are rotation quadrant for DRAW.  
 XOL, XOH (\$320, \$321) Most recent X-coordinate. Used for initial endpoint of LINE. Updated by PLOT, LINE, and FIND; not DRAW.  
 YO (\$322) Most recent Y-coordinate (see XOL, XOH).  
 BXSAV (\$323) Saves 6502 X-Register during HI-RES calls from BASIC.  
 HCOLOR (\$324) color specification to PLOT, POSN.  
 HNDX (\$325) On-the-fly byte index from BASE ADDRESS.  
 HPAG (\$326) Starting page of plot memory. Normally \$20 for plotting in primary HI-RES display memory (\$2000-\$3FFF).  
 SCALE (\$327) On-the-fly scale factor for DRAW.  
 SHAPXL, SHAPXH (\$328, \$329) 'Start of shape table' pointer.  
 COLLSN (\$32A) Collision count for DRAW and XDRAW.

NOTES:

(\*) Since this article was written, APPLE Computer Company made a production change in the APPLE ][. Only the early APPLE ][ computers displayed 2 HI-RES colors (besides Black and White). See the HI-RES Color Modification article for information on the additional colors available on the later models.

The APPLESOFT programming manual has further information on the creation of SHAPE TABLES and SHAPE TAPES. Refer to Chapter 9 for further information.



WOZPAK II

```

2 *****
3 *
4 * APPLE][HI-RESOLUTION *
5 * GRAPHICS SUBROUTINES *
6 *
7 * BY WOZ 9/13/77 *
8 *
9 * ALL RIGHTS RESERVED *
10 *
11 *****
12 *
13 * HI-RES EQUATES
14 *
15 SHAPEL EQU $1A POINTER TO
16 SHAPEH EQU $1B SHAPE LIST.
17 HCOLOR1 EQU $1C RUNNING COLOR MASK.
18 COUNTH EQU $1D
19 HBASL EQU $26 BAS ADR FOR CURRENT
20 HBASH EQU $27 HI-RES PLOT LINE.
21 HMASK EQU $30
22 A1L EQU $3C MONITOR A1.
23 A1H EQU $3D
24 A2L EQU $3E MONITOR A2.
25 A2H EQU $3F
26 LOMEML EQU $4A BASIC 'START OF VARS'.
27 LOMEMH EQU $4B
28 DXL EQU $50 DELTA-X FOR HLIN, SHAPE.
29 DXH EQU $51
30 SHAPEX EQU $51 SHAPE TEMP.
31 DY EQU $52 DELTA-Y FOR HLIN, SHAPE.
32 QDRNT EQU $53 ROT QUADRANT (SHAPE).
33 EL EQU $54 ERROR FOR HLIN.
34 EH EQU $55
35 PPL EQU $CA BASIC 'START OF PROG'.
36 PPH EQU $CB
37 PVL EQU $CC BASIC 'END OF VARS'.
38 PVH EQU $CD
39 ACL EQU $CE BASIC ACC.
40 ACH EQU $CF
41 XOL EQU $320 PRIOR X-COORD SAVE
42 XOH EQU $321 AFTER HLIN OR HPLOT.
43 YO EQU $322 HLIN, HPLLOT Y-COORD SAVE.
44 BXSAV EQU $323 X-REG SAVE FOR BASIC.
45 HCOLOR EQU $324 COLOR FOR HPLLOT, HPOSN.
46 HNDX EQU $325 HORIZ OFFSET SAVE.
47 HPAG EQU $326 HI-RES PAGE ($20-NORM).
48 SCALE EQU $327 SCALE FOR SHAPE, MOVE.
49 SHAPXL EQU $328 START OF
50 SHAPXH EQU $329 SHAPE TABLE.
51 COLLSN EQU $32A COLLISION COUNT.
52 SHSTRT EQU $C00 START OF SHAPE TABLE.
53 HIRES EQU $C057 SWITCH TO HIRES SCREEN.
54 MIXSET EQU $C053 SELECT TEXT/GRAPHICS.
55 TXTCLR EQU $C050 SELECT GRAPHICS MODE.
56 MEMFULL EQU $E36B BASIC 'MEM FULL ERR'.
57 RANGERR EQU $EE68 BASIC 'RANGE ERR'.
58 ACADR EQU $F11E 2-BYTE TAPE READ SETUP.
59 RD2BIT EQU $FCFA TWO-EDGE TAPE SENSE.
60 READ EQU $FEFD TAPE READ (A1.A2).
61 READX1 EQU $FF02 READ WITHOUT HEADER.

```

WOZPAK II

```

* *****
* HIGH RESOLUTION GRAPHICS INITS
* RAM VERSION $800 TO $BFF
* *****
*
* ORG $800
*
0800: A9 20 70 SETHRL LDA #$20 INIT FOR $2000-3FFF
0802: 8D 26 03 71 STA HPAG HI-RES SCREEN MEMORY.
0805: AD 57 C0 72 LDA HIRES SET HIRES DISPLAY MODE
0808: AD 53 C0 73 LDA MIXSET WITH TEXT AT BOTTOM.
080B: AD 50 C0 74 LDA TXTCLR SET GRAPHICS MODE.
080E: A9 00 75 HCLR LDA #$0
0810: 85 1C 76 BKGND0 STA HCOLOR1 SET FOR BLACK BKGND.
0812: AD 26 03 77 BKGND LDA HPAG
0815: 85 1B 78 STA SHAPEH INIT HI-RES SCREEN MEM
0817: A0 00 79 LDY #$0 FOR CURRENT PAGE, NORMALLY
0819: 84 1A 80 STY SHAPEL $2000-3FFF OR $4000-5FFF.
081B: A5 1C 81 BKGND1 LDA HCOLOR1
081D: 91 1A 82 STA (SHAPEL),Y
081F: 20 A2 08 83 JSR CSHFT2 (SHAPEL,H) WILL SPECIFY
0822: C8 84 84 INY ; 32 SEPERATE PAGES
0823: D0 F6 85 BNE BKGND1 THROUGHOUT THE INIT.
0825: E6 1B 86 INC SHAPEH
0827: A5 1B 87 LDA SHAPEH
0829: 29 1F 88 AND #$1F TEST FOR DONE.
082B: D0 EE 89 BNE BKGND1
082D: 60 90 91 RTS
*

* HI-RES GRAPHICS POSITION AND PLOT SUBRS
* *****
*
082E: 8D 22 03 96 HPOSN STA Y0 ENTER WITH Y IN A-REG,
0831: 8E 20 03 97 STX XOL XL IN X-REG,
0834: 8C 21 03 98 STY XOH AND XH IN Y-REG.
0837: 48 99 99 PHA
0838: 29 C0 100 AND #$C0
083A: 85 26 101 STA HBASL FOR Y-COORD=00ABCDEF
083C: 4A 102 102 LSR ; CALCULATES BASE ADDR
083D: 4A 103 103 LSR ; IN HBASL,HBASH FOR
083E: 05 26 104 ORA HBASL ACCESSING SCREEN MEMORY
0840: 85 26 105 STA HBASL VIA (HBASL),Y
0842: 68 106 106 PLA ; ADDRESSING MODE.
0843: 85 27 107 STA HBASH
0845: 0A 108 108 ASL ; CALCULATES
0846: 0A 109 109 ASL ; HBASH=PPPFHGCD,
0847: 0A 110 110 ASL ; HBASL=EABAB000
0848: 26 27 111 ROL HBASH
084A: 0A 112 112 ASL ; WHERE PPP=001 FOR $2000
084B: 26 27 113 ROL HBASH SCREEN MEM RANGE AND
084D: 0A 114 114 ASL ; PPP=010 FOR $4000-7FFF
084E: 66 26 115 ROR HBASL GIVEN Y-COORD=ABCDEFGH
0850: A5 27 116 LDA HBASH
0852: 29 1F 117 AND #$1F
0854: 0D 26 03 118 ORA HPAG
0857: 85 27 119 STA HBASH
0859: 8A 120 120 TXA ; DIVIDE X0 BY 7 FOR
085A: C0 00 121 CPY #$0 INDEX FROM BASE ADR
085C: F0 05 122 BEQ HPOSN2 (QUOTIENT) AND BIT

```

```

 WOZPAK II
085E: A0 23 124 LDY #$23 WITHIN SCREEN MEM BYTE
0860: 69 04 125 ADC #$4 (MASK SPEC'D BY REM'DR.
0862: C8 126 HPOSN1 INY
0863: E9 07 127 HPOSN2 SBC #$7 SUBTRACT OUT SEVENS.
0865: B0 FB 128 BCS HPOSN1
0867: 8C 25 03 129 STY HNDX WORKS FOR XO FROM
086A: AA 130 TAX ; 0 TO 279, LOW-ORDER
086B: BD EA 08 131 LDA MSKTBL-$F9,X BYTE IN X-REG, HIGH
086E: 85 30 132 STA HMASK IN Y-REG ON ENTRY.
0870: 98 133 TYA
0871: 4A 134 LSR
0872: AD 24 03 135 LDA HCOLOR IF ON ODD BYTE (CARRY SET)
0875: 85 1C 136 HPOSN3 STA HCOLOR1 THEN ROTATE (*)
0877: B0 29 137 BCS CSHFT2 BIT FOR 180 DEGREE SHIFT
0879: 60 138 RTS ; PRIOR TO COPY TO (*)
087A: 20 2E 08 139 HPL0T JSR HPOSN
087D: A5 1C 140 HPL0T1 LDA HCOLOR1 CALC BIT POSN IN HBASL,H
087F: 51 26 141 EOR (HBASL),Y HNDX, AND HMASK FROM
0881: 25 30 142 AND HMASK Y-COORD IN A-REG,
0883: 51 26 143 EOR (HBASL),Y X-COORD IN X,Y-REGS.
0885: 91 26 144 STA (HBASL),Y FOR ANY '1' BITS OF H(*)
0887: 60 145 RTS ; SUBSTITUTE CORRESPONDING
 146 * BIT OF HCOLOR1.
 147 *

 * HI-RES GRAPHICS L,R,U,D SUBRS

 *
0888: 10 24 152 LFTRT BPL RIGHT USE SIGN FOR LFT/RT SUBR.
088A: A5 30 153 LEFT LDA HMASK
088C: 4A 154 LSR ; SHIFT LOW-ORDER
088D: B0 05 155 BCS LEFT1 7 BITS OF HMASK
088F: 49 C0 156 EOR #$C0 ONE BIT TO LSB.
0891: 85 30 157 LR1 STA HMASK
0893: 60 158 RTS
0894: 88 159 LEFT1 DEY ; DECR HORIZ INDEX.
0895: 10 02 160 BPL LEFT2
0897: A0 27 161 LDY #$27 WRAP AROUND SCREEN.
0899: A9 C0 162 LEFT2 LDA #$C0 NEW HMASK, RIGHTMOST
089B: 85 30 163 NEWNDX STA HMASK DOT OF BYTE.
089D: 8C 25 03 164 STY HNDX UPDATE HORIZ INDEX.
08A0: A5 1C 165 CSHIFT LDA HCOLOR1
08A2: 0A 166 CSHFT2 ASL ; ROTATE LOW-ORDER
08A3: C9 C0 167 CMP #$C0 7 BITS OF HCOLOR1
08A5: 10 06 168 BPL RTS1 ONE BIT POSN.
08A7: A5 1C 169 LDA HCOLOR1
08A9: 49 7F 170 EOR #$7F ZXYXYXYX -> ZYXYXYX
08AB: 85 1C 171 STA HCOLOR1
08AD: 60 172 RTS1 RTS
08AE: A5 30 173 RIGHT LDA HMASK SHIFT LOW ORDER
08B0: 0A 174 ASL
08B1: 49 80 175 EOR #$80 7 BITS OF HMASK
08B3: 30 DC 176 BMI LR1 ONE BIT TO MSB.
08B5: A9 81 177 LDA #$81
08B7: C8 178 INY ; NEXT BYTE.
08B8: C0 28 179 CPY #$28
08BA: 90 DF 180 BCC NEWNDX
08BC: A0 00 181 LDY #$0 WRAP AROUND SCREEN IF (*)
08BE: B0 DB 182 BCS NEWNDX ALWAYS TAKEN.
08C0: 18 183 LRUDX1 CLC ; NO 90 DEG ROT (X-OR).

```

```

WOZPAK II
08C1: A5 51 185 LRUDX2 LDA SHAPEX
08C3: 29 04 186 AND #$4 IF B2=0 THEN NO PLOT.
08C5: F0 27 187 BEQ LRUD4
08C7: A9 7F 188 LDA #$7F FOR EX-OR INTO SCREEN (*)
08C9: 25 30 189 AND HMASK
08CB: 31 26 190 AND (HBASL),Y SCREEN BIT SET?
08CD: D0 1B 191 BNE LRUD3
08CF: EE 2A 03 192 INC COLLSN
08D2: A9 7F 193 LDA #$7F
08D4: 25 30 194 AND HMASK
08D6: 10 12 195 BPL LRUD3 ALWAYS TAKEN.
08D8: 18 196 LRUD1 CLC ; NO 90 DEG ROT.
08D9: A5 51 197 LRUD2 LDA SHAPEX
08DB: 29 04 198 AND #$4 IF B2=0 THEN NO PLOT.
08DD: F0 0F 199 BEQ LRUD4
08DF: B1 26 200 LDA (HBASL),Y
08E1: 45 1C 201 EOR HCOLOR1 SET HI-RES SCREEN BIT
08E3: 25 30 202 AND HMASK TO CORRESPONDING HCOLOR.
08E5: D0 03 203 BNE LRUD3 IF BIT OF SCREEN CHANGES,
08E7: EE 2A 03 204 INC COLLSN THEN INCR COLLSN DET.
08EA: 51 26 205 LRUD3 EOR (HBASL),Y
08EC: 91 26 206 STA (HBASL),Y
08EE: A5 51 207 LRUD4 LDA SHAPEX ADD QUADRANT TO
08F0: 65 53 208 ADC QDRNT SPECIFIED VECTOR
08F2: 29 03 209 AND #$3 AND MOVE LFT, RT,
210 EQ3 EQU *-1 UP, OR DOWN BASED
08F4: C9 02 211 CMP #$2 ON SIGN AND CARRY.
08F6: 6A 212 ROR
08F7: B0 8F 213 LRUD BCS LFTRT
08F9: 30 30 214 UPDOWN BMI DOWN4 SIGN FOR UP/DOWN SELECT.
08FB: 18 215 UP CLC
08FC: A5 27 216 LDA HBASH CALC BASE ADDRESS
08FE: 2C EA 09 217 BIT EQ1C (ADR OF LEFTMOST BYTE)
0901: D0 22 218 BNE UP4 FOR NEXT LINE UP
0903: 06 26 219 ASL HBASL IN (HBASL,HBASH)
0905: B0 1A 220 BCS UP2 WITH 192-LINE WRAP AROUND.
0907: 2C F3 08 221 BIT EQ3
090A: F0 05 222 BEQ UP1
090C: 69 1F 223 ADC #$1F **** BIT MAP ****
090E: 38 224 SEC
090F: B0 12 225 BCS UP3 FOR ROW = ABCDEFGH
0911: 69 23 226 UP1 ADC #$23
0913: 48 227 PHA
0914: A5 26 228 LDA HBASL HBASL=EABAB000
0916: 69 B0 229 ADC #$B0 HBASH=PPPFHGCD
0918: B0 02 230 BCS UP5
091A: 69 F0 231 ADC #$F0 WHERE PPP=001 FOR PRI(*)
091C: 85 26 232 UP5 STA HBASL HI-RES PAGE ($2000-3FFF).
091E: 68 233 PLA
091F: B0 02 234 BCS UP3
0921: 69 1F 235 UP2 ADC #$1F
0923: 66 26 236 UP3 ROR HBASL
0925: 69 FC 237 UP4 ADC #$FC
0927: 85 27 238 UPDOWN1 STA HBASH
0929: 60 239 RTS
092A: 18 240 DOWN CLC
092B: A5 27 241 DOWN4 LDA HBASH
092D: 69 04 242 ADC #$4 CALC BASE ADR FOR NEXT(*)
243 EQ4 EQU *-1 DOWN TO(HBASL,HBASH).
092F: 2C EA 09 244 BIT EQ1C

```

WOZPAK II

```

0932: D0 F3 246 BNE UPDWN1
0934: 06 26 247 ASL HBASL WITH 192-LINE WRAPAROUND.
0936: 90 19 248 BCC DOWN1
0938: 69 E0 249 ADC #$E0
093A: 18 250 CLC
093B: 2C 2E 09 251 BIT EQ4
093E: F0 13 252 BEQ DOWN2
0940: A5 26 253 LDA HBASL
0942: 69 50 254 ADC #$50
0944: 49 F0 255 EOR #$F0
0946: F0 02 256 BEQ DOWN3
0948: 49 F0 257 EOR #$F0
094A: 85 26 258 DOWN3 STA HBASL
094C: AD 26 03 259 LDA HPAG
094F: 90 02 260 BCC DOWN2
0951: 69 E0 261 DOWN1 ADC #$E0
0953: 66 26 262 DOWN2 ROR HBASL
0955: 90 D0 263 BCC UPDWN1
 264 *
 266 *****
 * HI-RES GRAPHICS LINE DRAW SUBRS

 *
0957: 48 268 *
0957: 48 269 HLINRL PHA
0958: A9 00 270 LDA #$0 SET (XOL,XOH) AND
095A: 8D 20 03 271 STA XOL YO TO ZERO FOR
095D: 8D 21 03 272 STA XOH REL LINE DRAW
0960: 8D 22 03 273 STA YO (DX,DY).
0963: 68 274 PLA
0964: 48 275 HLIN PHA ; ON ENTRY:
0965: 38 276 SEC ; XL: A-REG
0966: ED 20 03 277 SBC XOL XH: X-REG
0969: 48 278 PHA ; Y: Y-REG
096A: 8A 279 TXA
096B: ED 21 03 280 SBC XOH
096E: 85 53 281 STA QDRNT CALC ABS(X-XO)
0970: B0 0A 282 BCS HLIN2 IN (DXL,DXH).
0972: 68 283 PLA
0973: 49 FF 284 EOR #$FF X DIR TO SIGN BIT
0975: 69 01 285 ADC #$1 OF QDRNT.
0977: 48 286 PHA ; 0=RIGHT (DX POS)
0978: A9 00 287 LDA #$0 1=LEFT (DX NEG)
097A: E5 53 288 SBC QDRNT
097C: 85 51 289 HLIN2 STA DXH
097E: 85 55 290 STA EH INIT (EL,EH) TO
0980: 68 291 PLA ; ABS (X-XO)
0981: 85 50 292 STA DXL
0983: 85 54 293 STA EL
0985: 68 294 PLA
0986: 8D 20 03 295 STA XOL
0989: 8E 21 03 296 STX XOH
098C: 98 297 TYA
098D: 18 298 CLC
098E: ED 22 03 299 SBC YO CALC -ABS(Y-YO)-1
0991: 90 04 300 BCC HLIN3 IN DY.
0993: 49 FF 301 EOR #$FF
0995: 69 FE 302 ADC #$FE
0997: 85 52 303 HLIN3 STA DY ROTATE Y DIR INTO
0999: 8C 22 03 304 STY YO QDRNT SIGN BIT
099C: 66 53 305 ROR QDRNT (0=UP, 1=DOWN)

```

WOZPAK II

```

099E: 38 307 SEC
099F: E5 50 308 SBC DXL INIT (COUNTL, COUNTH)
09A1: AA 309 TAX ; TO -(DELTX, DELTY+1)
09A2: A9 FF 310 LDA #$FF
09A4: E5 51 311 SBC DXH
09A6: 85 1D 312 STA COUNTH
09A8: AC 25 03 313 LDY HNDX HORIZ INDEX.
09AB: B0 05 314 BCS MOVEX2 ALWAYS TAKEN.
09AD: 0A 315 MOVEX ASL ; MOVE IN X-DIR. USE
09AE: 20 88 08 316 JSR LFRTRT QDRNT B6 FOR LFT/RT SET.
09B1: 38 317 SEC
09B2: A5 54 318 MOVEX2 LDA EL ASSUME CARRY SET.
09B4: 65 52 319 ADC DY (EL,EH)-DELTY TO (EL,EH)
09B6: 85 54 320 STA EL NOTE: DY IS (-DELTY)-1
09B8: A5 55 321 LDA EH CARRY CLR IF (EL,EH)
09BA: E9 00 322 SBC #$0 GOES NEG.
09BC: 85 55 323 HCOUNT STA EH
09BE: B1 26 324 LDA (HBASL),Y SCREEN BYTE.
09C0: 45 1C 325 EOR HCOLOR1 PLOT DOT OF HCOLOR1.
09C2: 25 30 326 AND HMASK CURRENT BIT MASK.
09C4: 51 26 327 EOR (HBASL),Y
09C6: 91 26 328 STA (HBASL),Y
09C8: E8 329 INX ; DONE (DELTX+DELTY)
09C9: D0 04 330 BNE HLIN4 DOTS?
09CB: E6 1D 331 INC COUNTH
09CD: F0 6B 332 BEQ RTS2 YES, RETURN.
09CF: A5 53 333 HLIN4 LDA QDRNT FOR DIRECTION TEST.
09D1: B0 DA 334 BCS MOVEX IF CAR SET, (EL,EH) PLOT
09D3: 20 F9 08 335 JSR UPDWN IF CLR, NEG, MOVE (*)
09D6: 18 336 CLC
09D7: A5 54 337 LDA EL (EL,EH)+DELTX
09D9: 65 50 338 ADC DXL TO (EL,EH).
09DB: 85 54 339 STA EL
09DD: A5 55 340 LDA EH CAR SET IF (EL,EH) GOES
09DF: 65 51 341 ADC DXH (*)
09E1: 50 D9 342 BVC HCOUNT ALWAYS TAKEN.
09E3: 81 82 84 343 MSKTBL HEX 818284
09E6: 88 90 A0 344 HEX 8890A0
09E9: C0 345 HEX C0
09EA: 1C 346 EQ1C HEX 1C
09EB: FF FE FA 347 COS HEX FFFEFA
09EE: F4 EC E1 348 HEX F4ECE1
09F1: D4 C5 B4 349 HEX D4C5B4
09F4: A1 8D 78 350 HEX A18D78
09F7: 61 49 31 351 HEX 614931
09FA: 18 FF 352 HEX 18FF
 353 *
 355 * *****
 357 * HI-RES GRAPHICS COORDINATE RESTORE SUBR
 359 * *****
09FC: A5 26 358 HFIND LDA HBASL CONVERTS BASE ADDR
09FE: 0A 359 ASL
09FF: A5 27 360 LDA HBASH TO Y-COORD.
0A01: 29 03 361 AND #$3
0A03: 2A 362 ROL ; FOR HBASL=EABAB00
0A04: 05 26 363 ORA HBASL HBASH=PPFPGHCD
0A06: 0A 364 ASL
0A07: 0A 365 ASL ; GENERATE
0A08: 0A 366 ASL ; Y-COORD=ABCDEFGH

```

```

WOZPAK II
OA09: 8D 22 03 368 STA Y0
OA0C: A5 27 369 LDA HBASH (PPP=SCREEN PAGE,
OA0E: 4A 370 LSR ; NORMALLY 001 FOR
OA0F: 4A 371 LSR ; $2000-$3FFF
OA10: 29 07 372 AND #$7 HI-RES SCREEN)
OA12: 0D 22 03 373 ORA Y0
OA15: 8D 22 03 374 STA Y0 CONVERTS HNDX (INDEX
OA18: AD 25 03 375 LDA HNDX FROM BASE ADR)
OA1B: 0A 376 ASL ; AND HMASK (BIT
OA1C: 6D 25 03 377 ADC HNDX MASK) TO X-COORD
OA1F: 0A 378 ASL ; IN (XOL,XOH)
OA20: AA 379 TAX ; (RANGE $0-133)
OA21: CA 380 DEX
OA22: A5 30 381 LDA HMASK
OA24: 29 7F 382 AND #$7F
OA26: E8 383 HFIND1 INX
OA27: 4A 384 LSR
OA28: D0 FC 385 BNE HFIND1
OA2A: 8D 21 03 386 STA XOH
OA2D: 8A 387 TXA
OA2E: 18 388 CLC ; CALC HNDX*7 +
OA2F: 6D 25 03 389 ADC HNDX LOG(BASE 2) HMASK.
OA32: 90 03 390 BCC HFIND2
OA34: EE 21 03 391 INC XOH
OA37: 8D 20 03 392 HFIND2 STA XOL
OA3A: 60 393 RTS2 RTS
394 *
396 * *****
* HI-RES GRAPHICS SHAPE DRAW SUBR

*
398
OA3B: 86 1A 399 DRAW STX SHAPEL DRAW DEFINITION
OA3D: 84 1B 400 STY SHAPEH POINTER.
OA3F: AA 401 DRAW1 TAX
OA40: 4A 402 LSR ; ROT ($0-$3F)
OA41: 4A 403 LSR
OA42: 4A 404 LSR ; QDRNT 0=UP,1=RT,
OA43: 4A 405 LSR ; 2=DWN, 3=LFT.
OA44: 85 53 406 STA QDRNT
OA46: 8A 407 TXA
OA47: 29 0F 408 AND #$F
OA49: AA 409 TAX
OA4A: BC EB 09 410 LDY COS,X SAVE COS AND SIN
OA4D: 84 50 411 STY DXL VALS IN DXL AND DY.
OA4F: 49 0F 412 EOR #$F
OA51: AA 413 TAX
OA52: BC EC 09 414 LDY COS+1,X
OA55: C8 415 INY
OA56: 84 52 416 STY DY
OA58: AC 25 03 417 DRAW2 LDY HNDX BYTE INDEX FROM
OA5B: A2 00 418 LDX #$0 HI-RES BASE ADDR.
OA5D: 8E 2A 03 419 STX COLLSN CLEAR COLLISION COUNT.
OA60: A1 1A 420 LDA (SHAPEL,X) 1ST. SHAPE DEF BYTE.
OA62: 85 51 421 DRAW3 STA SHAPEX
OA64: A2 80 422 LDX #$80
OA66: 86 54 423 STX EL EL,EH FOR FRACTIONAL
OA68: 86 55 424 STX EH L,R,U,D VECTORS.
OA6A: AE 27 03 425 LDX SCALE SCALE FACTOR.
OA6D: A5 54 426 DRAW4 LDA EL
OA6F: 38 427 SEC ; IF FRAC COS OVRFL

```

WOZPAK II

```

OA70: 65 50 429 ADC DXL THEN MOVE IN
OA72: 85 54 430 STA EL SPECIFIED VECTOR
OA74: 90 04 431 BCC DRAW5 DIRECTION.
OA76: 20 D8 08 432 JSR LRUD1
OA79: 18 433 CLC
OA7A: A5 55 434 DRAW5 LDA EH IF FRAC SIN OVRFL
OA7C: 65 52 435 ADC DY THEN MOVE IN
OA7E: 85 55 436 STA EH SPECIFIED VECTOR
OA80: 90 03 437 BCC DRAW6 DIRECTION+90 DEG.
OA82: 20 D9 08 438 JSR LRUD2
OA85: CA 439 DRAW6 ; LOOP ON SCALE
OA86: D0 E5 440 BNE DRAW4 FACTOR.
OA88: A5 51 441 LDA SHAPEX
OA8A: 4A 442 LSR ; NEXT 3-BIT VECTOR
OA8B: 4A 443 LSR ; OF SHAPE DEF.
OA8C: 4A 444 LSR
OA8D: D0 D3 445 BNE DRAW3 NOT DONE THIS BYTE.
OA8F: E6 1A 446 INC SHAPEL.
OA91: D0 02 447 BNE DRAW7 NEXT BYTE OF
OA93: E6 1B 448 INC SHAPEH SHAPE DEFINITION.
OA95: A1 1A 449 DRAW7 LDA (SHAPEL,X)
OA97: D0 C9 450 BNE DRAW3 DONE IF ZERO.
OA99: 60 451 RTS
 452 *

 * HI-RES GRAPHICS SHAPE EX-OR SUBR

 *
OA9A: 86 1A 457 XDRAW STX SHAPEL SHAPE DEFINITION
OA9C: 84 1B 458 STY SHAPEH POINTER.
OA9E: AA 459 XDRAW1 TAX
OA9F: 4A 460 LSR
OAA0: 4A 461 LSR
OAA1: 4A 462 LSR
OAA2: 4A 463 LSR
OAA3: 85 53 464 STA QDRNT
OAA5: 8A 465 TXA
OAA6: 29 0F 466 AND #$F
OAA8: AA 467 TAX
OAA9: BC EB 09 468 LDY COS,X SAVE COS AND SIN
OAAC: 84 50 469 STY DXL VALS IN DXL AND DY.
OAAE: 49 0F 470 EOR #$F
OAB0: AA 471 TAX
OAB1: BC EC 09 472 LDY COS+1,X
OAB4: C8 473 INY
OAB5: 84 52 474 STY DY
OAB7: AC 25 03 475 XDRAW2 LDY HNDX INDEX FROM HI-RES
OABA: A2 00 476 LDX #$0 BASE ADR.
OABC: 8E 2A 03 477 STX COLLSN CLEAR COLLISION DETECT.
OABF: A1 1A 478 LDA (SHAPEL,X) 1ST SHAPE DEF BYTE.
OAC1: 85 51 479 XDRAW3 STA SHAPEX
OAC3: A2 80 480 LDX #$80
OAC5: 86 54 481 STX EL EL,EH FOR FRACTIONAL
OAC7: 86 55 482 STX EH L,R,U,D VECTORS.
OAC9: AE 27 03 483 XDRAW4 LDX SCALE SCALE FACTOR.
OACC: A5 54 484 LDA EL
OACE: 38 485 SEC ; IF FRAC COS OVRFL
OACF: 65 50 486 ADC DXL THEN MOVE IN
OAD1: 85 54 487 STA EL SPECIFIED VECTOR
OAD3: 90 04 488 BCC XDRAW5 DIRECTION.

```



```

 WOZPAK II
OAD5: 20 C0 08 490 JSR LRUDX1
OAD8: 18 CLC
OAD9: A5 55 492 XDRAW5 LDA EH IF FRAC SIN OVRFL
OADB: 65 52 493 ADC DY THEN MOVE IN
OADD: 85 55 494 STA EH SPECIFIED VECTOR
OADF: 90 03 495 BCC XDRAW6 DIRECTION+90 DEG.
OAE1: 20 D9 08 496 JSR LRUD2
OAE4: CA XDRAW6 DEX ; LOOP ON SCALE
OAE5: D0 E5 498 BNE XDRAW4 FACTOR.
OAE7: A5 51 499 LDA SHAPEX
OAE9: 4A LSR
OAEA: 4A LSR
OAEB: 4A LSR
Oaec: D0 D3 503 BNE XDRAW3
OAEe: E6 1A 504 INC SHAPEL
OAF0: D0 02 505 BNE XDRAW7 NEXT BYTE OF
OAF2: E6 1B 506 INC SHAPEH SHAPE DEF.
OAF4: A1 1A 507 XDRAW7 LDA (SHAPEL,X)
OAF6: D0 C9 508 BNE XDRAW3 DONE IF ZERO.
OAF8: 60 RTS
 *

 * HI-RES GRAPHICS ENTRY POINTS
 * FROM APPLE][INTEGER BASIC

 *
 512
 *
 513
 *

 *
OAF9: 20 90 0B 516 BPOSN JSR PCOLR POSN CALL, COLOR FROM
OAFc: 8D 24 03 517 STA HCOLOR BASIC.
OAFf: 20 AF 0B 518 JSR GETYO YO FROM BASIC.
OB02: 48 PHA
OB03: 20 9A 0B 520 JSR GETXO XO FROM BASIC.
OB06: 68 PLA
OB07: 20 2E 08 522 JSR HPOSN
OB0A: AE 23 03 523 LDX BXSAV
OB0D: 60 RTS
OB0E: 20 F9 0A 525 BPLOt JSR BPOSN PLOT CALL (BASIC).
OB11: 4C 7D 08 526 JMP HPLOt1
OB14: AD 25 03 527 BLIN1 LDA HNDX SET HCOLOR1 FROM
OB17: 4A LSR
OB18: 20 90 0B 529 JSR PCOLR BASIC VARI COLR.
OB1B: 20 75 08 530 JSR HPOSN3
OB1E: 20 9A 0B 531 BLINE JSR GETXO LINE CALL, GET XO
OB21: 8A TXA ; FROM BASIC.
OB22: 48 PHA
OB23: 98 TYA
OB24: AA TAX
OB25: 20 AF 0B 536 JSR GETYO YO FROM BASIC.
OB28: A8 TAY
OB29: 68 PLA
OB2A: 20 64 09 539 JSR HLIN
OB2D: AE 23 03 540 LDX BXSAV
OB30: 60 RTS
OB31: 20 90 0B 542 BGND JSR PCOLR BACKGROUND CALL.
OB34: 4C 10 08 543 JMP BKGND0
 *

 * HI-RES GRAPHICS DRAW SUBRS

 *
 544
 *
 546
 *

 *
OB37: 20 F9 0A 549 BDRAW1 JSR BPOSN

```

```

WOZPAK II
OB3A: 20 51 0B 551 BDRAW JSR BDRAWX DRAW CALL FROM BASIC.
OB3D: 20 3B 0A 552 JSR DRAW
OB40: AE 23 03 553 LDX BXSAV
OB43: 60 554 RTS
OB44: 20 F9 0A 555 BXDRW1 JSR BPOSN
OB47: 20 51 0B 556 BXDRAW JSR BDRAWX EX-OR DRAW
OB4A: 20 9A 0A 557 JSR XDRAW FROM BASIC.
OB4D: AE 23 03 558 LDX BXSAV
OB50: 60 559 RTS
OB51: 8E 23 03 560 BDRAWX STX BXSAV SAVE FOR RTS TO BASIC.
OB54: A0 32 561 LDY #$32
OB56: 20 92 0B 562 JSR PBYTE SCALE FROM BASIC.
OB59: 8D 27 03 563 STA SCALE
OB5C: A0 28 564 LDY #$28
OB5E: 20 92 0B 565 JSR PBYTE ROT FROM BASIC &
OB61: 48 566 PHA ; SAVE ON STACK.
OB62: AD 28 03 567 LDA SHAPXL
OB65: 85 1A 568 STA SHAPEL START OF
OB67: AD 29 03 569 LDA SHAPXH SHAPE TABLE.
OB6A: 85 1B 570 STA SHAPEH
OB6C: A0 20 571 LDY #$20
OB6E: 20 92 0B 572 JSR PBYTE SHAPE FROM BASIC.
OB71: F0 39 573 BEQ RERR1
OB73: A2 00 574 LDX #$0
OB75: C1 1A 575 CMP (SHAPEL,X) > NUMBR OF SHAPES?
OB77: F0 02 576 BEQ BDRWX1
OB79: B0 31 577 BCS RERR1 YES, RANGE ERROR.
OB7B: 0A 578 BDRWX1 ASL
OB7C: 90 03 579 BCC BDRWX2
OB7E: E6 1B 580 INC SHAPEH
OB80: 18 581 CLC
OB81: A8 582 BDRWX2 TAY ; SHAPE# * 2.
OB82: B1 1A 583 LDA (SHAPEL),Y
OB84: 65 1A 584 ADC SHAPEL
OB86: AA 585 TAX ; ADD 2-BYTE INDEX
OB87: C8 586 INY ; TO SHAPE TABLE
OB88: B1 1A 587 LDA (SHAPEL),Y START ADDRESS
OB8A: 6D 29 03 588 ADC SHAPXH (X LOW, Y HI).
OB8D: A8 589 TAY
OB8E: 68 590 PLA ; ROT FROM STACK.
OB8F: 60 591 RTS
592 *

594 * HI-RES GRAPHICS BASIC
595 * PARAMETER FETCH SUBRS

597 *
OB90: A0 16 598 PCOLR LDY #$16
OB92: B1 4A 599 PBYTE LDA (LOMEML),Y
OB94: D0 16 600 BNE RERR1 GET BASIC PARAM.
OB96: 88 601 DEY ; (ERR IF >255)
OB97: B1 4A 602 LDA (LOMEML),Y
OB99: 60 603 RTSB RTS
OB9A: 8E 23 03 604 GETXO STX BXSAV SAVE FOR RTS TO BASIC.
OB9D: A0 05 605 LDY #$5
OB9F: B1 4A 606 LDA (LOMEML),Y XO LOW-ORDER BYTE
OBA1: AA 607 TAX
OBA2: C8 608 INY
OBA3: B1 4A 609 LDA (LOMEML),Y XO HI-ORDER BYTE
OBA5: A8 610 TAY

```

WOZPAK II

```

OBA6: ED 18 612
OBA8: E9 01 613
OBAA: 90 ED 614
OBAC: 4C 68 EE 615
OBAF: A0 0D 616
OBB1: 20 92 0B 617
OBB4: C9 C0 618
OBB6: B0 F4 619
OBB8: 60 620
 621
 *
 623 *****
 * HI-RES GRAPHICS SHAPE
 * TAPE LOAD SUBR.
 624 *****
 *
OBB9: 8E 23 03 627 SHLOAD STX BXSAV SAVE FOR RTS TO BASIC.
OBBc: 20 1E F1 628 JSR ACADR READ 2-BYTE LENGTH INTO
OBBF: 20 FD FE 629 JSR READ BASIC ACC ($C,$F).
OBC2: A9 00 630 LDA #<SHSTRT
OBC4: 85 3C 631 STA A1L
OBC6: 8D 28 03 632 STA SHAPXL
OBC9: 18 633 CLC
OBcA: 65 CE 634 ADC ACL
OBcC: A8 635 TAY
OBcD: A9 0C 636 LDA #>SHSTRT
OBcF: 85 3D 637 STA A1H
OBD1: 8D 29 03 638 STA SHAPXH
OBD4: 65 CF 639 ADC ACH
OBD6: B0 25 640 BCS MFULL1 NOT ENOUGH MEMORY!
OBD8: C4 CA 641 CPY PPL
OBDa: 48 642 PHA
OBD8: E5 CB 643 SBC PPH
OBDd: 68 644 PLA
OBDc: B0 1D 645 BCS MFULL1
OBE0: 84 3E 646 STY A2L
OBE2: 85 3F 647 STA A2H
OBE4: C8 648 INY
OBE5: D0 02 649 BNE SHLOD1
OBE7: 69 01 650 ADC #$1
OBE9: 84 4A 651 SHLOD1 STY LOMEML
OBEb: 85 4B 652 STA LOMEMH
OBEd: 84 CC 653 STY PVL
OBEF: 85 CD 654 STA PVH
OBF1: 20 FA FC 655 JSR RD2BIT
OBF4: A9 03 656 LDA #$3 .5 SEC HEADER
OBF6: 20 02 FF 657 JSR READX1
OBF9: AE 23 03 658 LDX BXSAV
OBFc: 60 659 RTS
OBFd: 4C 6B E3 660 MFULL1 JMP MEMFULL

```

---

**BLANK PAGE**

---

PRODUCING AND USING SHAPE TABLES

by Robert C. Clardy

---

**BLANK PAGE**

---

WOZPAK II

One of the most versatile graphics capabilities of the APPLE ][ is provided by the use of the DRAW and XDRAW routines in the HIRES graphics package provided by the APPLE company. Each of these two functions can be used to draw (virtually instantaneously) a predefined shape onto the HIRES screen. The XDRAW function has the additional characteristic that each point is exclusive-ORed to points already on the screen at that location. Each dot has it's previous color reversed. A dot which was previously white becomes black, orange becomes green, etc.

The exclusive-OR feature has several useful effects. A shape can be XDRAWN once, then erased by XDRAWing it again at the same location. Further, a shape can be XDRAWN on a background picture and then erased, leaving the original picture intact. This feature can be used to move cars/tanks/spaceships across a background display without disturbing it. A good example that many have seen is the STAR WARS program; the crosshairs are not disturbed even when a fighter moves over them.

To use the draw functions, the following steps must be taken:

- (1) Load the HIRES graphics routines
- (2) Obtain a shape file (255 shapes max) by:
  - (a) Loading existent shape table
  - (b) Creating a shape table
- (3) Initialize the HIRES variables properly (see previous section).
- (4) Specify SHAPE, COLR, XO, YO, and SCALE as required (see previous section).
- (5) Draw the shape (Use CALL 2871 or CALL 2884. See previous section).

Each of these steps will be discussed in greater detail below.

There are several versions of the HIRES graphics routines available. The easiest to use are found in the HIRES SUBROUTINES program appearing in the previous section. These routines can be appended to an INTEGER BASIC program using the PACK & SAVE routine also appearing in this issue. Once run, it stores the HIRES routines in memory locations 2048-3071 (\$800-\$8FF) and sets LOMEM to 3072. Remember that the HIRES screen is stored in locations 8192-16383 (\$2000-\$3FFF). Set

HIMEM:8192 or LOMEM:16384 so your program does not interfere with the HIRES display.

With the HIRES routines loaded, you can load a shape table from cassette by typing CALL 3001, starting your recorder, and pressing return. To load from disk, simply BLOAD <filename>,A<address>. You must then store the table's starting address in locations 808 and 809 by doing: POKE 808,<address> MOD 256 and POKE 809,<address> / 256.

If you do not have an existent shape table, you can make one with the program listed below. The program is completely self prompting, allowing the user to draw up to 255 shapes per shape table. Each shape is produced point by point. The total bytes used by each shape and the total number of memory locations used by the entire table is constantly displayed. Make certain that the table does not go beyond location 4150 (LOMEM for the program as written).

The program, as is, will run on a 16K APPLE. The room allocated for shapes is, however, rather small (see Table 1). If additional shape storage is required, it can be obtained by deleting lines 1630 through 1700 and line 0. As can be seen in the table, this gives 710 additional bytes for shapes. If your system has more than 16K, delete lines 0 and 1.

WARNING: Because of the nature of the HIRE routines, you cannot move the HIRES cursor up more than 2 units with the plot turned off. If this is done, the shape will be terminated at that point.

TABLE 1

| OPTION | SYSTEM | HIMEM | LOMEM | SHAPE | BYTE |
|--------|--------|-------|-------|-------|------|
| 1      | 16K    | 8192  | 4150  | 1078  |      |
| 2      | 16K    | 8192  | 4860  | 1788  |      |
| 3      | >16K   | **    | 16384 | **    |      |

\*\* Limited by system size.

(ed.note) This program contains 'illegal' commands. If the user is unfamiliar with the methods to enter a 'LOMEM:' into a program, the required LOMEM: must be entered before the program is run. (The version on the tape is as listed)

## WOZPAK II

```

0 LOMEM:4150: HIMEM:8192: GOTO
8
1 LOMEM:4860: HIMEM:8192: GOTO
8
2 LOMEM:16384
8 XO=Y0=COLR=SHAPE=ROT=SCALE:
SHAPE=1: POKE 808,0: POKE 809
,12: GOTO 1615
10 V= PEEK (-16384): IF V<128 THEN
10: POKE -16368,0: RETURN
11 CALL -936: PRINT : TAB 15: RETURN

31 BYTE=0: CALL INIT:PFLAG=0:Y0=
79:X0=139:COLR=255: GOSUB 800

36 PT=3072+2*SHAPE:LOC=3072+ PEEK
(PT)+256* PEEK (PT+1):LOC1=
LOC
38 VTAB 22: TAB 34: PRINT "ON "
;: POKE 50,63: PRINT "OFF":
POKE 50,255
50 GOSUB 10
100 IF V=213 THEN 200: IF V=196
THEN 250: IF V=210 THEN 300
: IF V=204 THEN 350: IF V=208
THEN 400: IF V=134 THEN 1090
: IF V=151 THEN 31
150 GOTO 50
200 Y0=OLDY-1:A=4: IF PFLAG=0 THEN
A=0: GOTO 360
250 Y0=OLDY+1:A=6: IF PFLAG=0 THEN
A=2: GOTO 360
300 X0=OLDX+1:A=5: IF PFLAG=0 THEN
A=1: GOTO 360
350 X0=OLDX-1:A=7: IF PFLAG=0 THEN
A=3
360 GOSUB 1000: GOSUB 800: GOTO
50
400 PFLAG= NOT PFLAG: IF PFLAG=
0 THEN 410
402 VTAB 22: TAB 34: POKE 50,63
: PRINT "ON";: POKE 50,255:
PRINT " OFF": GOTO 50
410 VTAB 22: TAB 34: POKE 50,255
: PRINT "ON ";: POKE 50,63:
PRINT "OFF": POKE 50,255: GOTO
50
800 IF Y0<0 THEN Y0=0: IF Y0>159
THEN Y0=159: IF X0<0 THEN
X0=0: IF X0>279 THEN X0=279

805 IF PFLAG=0 THEN 820:COLR=255

810 CALL PLOT: GOTO 840
820 COLR=0:T1=X0:T2=Y0:X0=OLDX:
Y0=OLDY: CALL PLOT:X0=T1:Y0=
T2
830 COLR=255: CALL PLOT
840 OLDX=X0:OLDY=Y0: RETURN

1000 F=0:D=D+1:D(D)=A: IF D<3 THEN
RETURN
1005 IF D(3)#0 THEN 1010
1006 F=1: GOTO 1030
1010 IF D(3)<4 THEN 1030
1020 Q=D(3):D(3)=0
1030 Z=D(1)+D(2)*8+D(3)*64
1035 BYTE=BYTE+1: VTAB 22: PRINT
"BYTES=";BYTE: PRINT "LOC ="
;LOC
1040 POKE LOC,Z:LOC=LOC+1: IF LOC>
LOW-2 THEN 1490: IF F=1 THEN
1050
1045 IF Q=0 THEN 1070
1050 IF D(2)#0 THEN 1060
1055 D(1)=0:D(2)=Q:D(3)=0:Q=0:D=
2: RETURN
1060 D(1)=Q:D(2)=0:D(3)=0:Q=0:D=
1: RETURN
1070 FOR I=1 TO 3:D(I)=0: NEXT I:
D=0: RETURN
1090 Z=D(1)+D(2)*8+D(3)*64: POKE
LOC,Z
1100 IF Z=0 THEN 1120
1110 LOC=LOC+1: POKE LOC,0
1120 PRINT "VECTOR TABLE: FROM "
;LOC1;" TO ";LOC
1300 COLR=255:X0=139:Y0=79:SCALE=
1: CALL INIT: CALL DRAW
1310 PRINT "IS THIS THE SHAPE THAT YO
U WANT (Y/N)?": GOSUB 10: IF
V#206 THEN 1320: GOSUB 11: PRINT
"TRY AGAIN.": GOTO 31
1320 GOSUB 11: PRINT "ANOTHER SHAPE (
Y/N)?": GOSUB 10: CALL -936
: IF V=206 THEN 1500
1330 GOSUB 11: PRINT "SHAPE # ";
SHAPE+1:SHAPE=SHAPE+1
1340 IF SHAPE>SH THEN 1490:LOC=LOC+
1: POKE 3072+2*SHAPE,(LOC MOD
256): POKE 3073+2*SHAPE,(LOC/
256)-12: GOTO 31
1490 GOSUB 11: PRINT "SORRY, SHAPE TA
BLE IS FULL."
1500 PRINT "SAVE SHAPE TABLE TO TAPE
(Y/N)?": GOSUB 10: IF V#217
THEN 1550: PRINT "START RECORDE
R AND PRESS RETURN": GOSUB
10
1510 POKE 4140,(LOC-3071) MOD 256
: POKE 4141,(LOC-3071)/256:
POKE 60,44: POKE 61,16: POKE
62,45: POKE 63,16: CALL -307

1520 POKE 60,0: POKE 61,12: POKE
62,LOC MOD 256: POKE 63,LOC/
256: CALL -307

```



## WOZPAK II

```

1550 PRINT "SAVE SHAPE TABLE TO DISC
(Y,N)?": GOSUB 10: IF V#217
THEN END : INPUT "FILE NAME "
,F$
1560 PRINT "BSAVE";F$;"A3072,L"
;LOC-3072+5: END
1601 POKE 808,0: POKE 809,12: POKE
3072,50: POKE 3074,102: POKE
3075,0
1615 DIM F$(20),D(3):INIT=2048:CLEAR=
2062:PLOT=2830:DRAW=2871
1620 TEXT : CALL -936: VTAB 10: TAB
6: PRINT "HIRES SHAPE TABLE GENE
RATOR": PRINT : TAB 10: PRINT
"BY ROBERT C. CLARDY"
1625 FOR I=1 TO 1000: NEXT I
1630 VTAB 14: PRINT "THIS PROGRAM WAS
ADAPTED FROM A SIMILAR ROUTINE
WRITTEN BY GARY D. HAWKINS AS IT
";
1640 PRINT "APPEARED IN THE JULY, 197
8 ISSUE OF CREATIVE COMPUTIN
G. THE PROGRAM WAS"
1650 PRINT "MODIFIED TO RUN WITH THE
UPDATED HIRES ROUTINES AND TO A
LLOW DIFINITION OF"
1660 PRINT "MULTIPLE SHAPES. TO USE T
HE PROGRAM, USE THE FOLLOWING
COMMANDS:"
1670 PRINT : TAB 6: PRINT "U = MOVE U
P ONE SPACE": TAB 6: PRINT
"D = MOVE DOWN ONE SPACE": TAB
6: PRINT "R = MOVE RIGHT ONE SPA
CE"
1680 TAB 6: PRINT "L = MOVE LEFT ONE
SPACE": TAB 6: PRINT "P = INVERT
POINT STATUS": TAB 6: PRINT
"CONTROL W = START OVER (WIPE)"
: TAB 6
1690 PRINT "CONTROL F = SHAPE IS FINI
SHED": PRINT : PRINT "IF POINT S
TATUS IS 'ON', A POINT WILL"
1700 PRINT "BE PLOTTED. IF IT IS 'OFF
', IT WON'T. PRESS ANY KEY TO
BEGIN.": GOSUB 10: POKE 34
,21: CALL -936
1705 INPUT "HOW MANY SHAPES DO YOU WA
NT (LIMIT 255)",SH: IF SH<0
OR SH>255 THEN 1705: POKE
808,0: POKE 809,12: POKE 3072
,SH
1710 POKE 3074,(2*SH+2) MOD 256:
POKE 3075,(2*SH+2)/256:LOW=
256* PEEK (75)+ PEEK (74): CALL
-936: GOTO 31

```

---

**BLANK PAGE**

---

WOZPAK II

SWEET 16 Introduction  
by Dick Sedgewick

Sweet 16 is probably the least used and least understood seed in the Apple ][.

In exactly the same sense that Integer and Applesoft Basics are languages, Sweet 16 is a language. Compared to the Basics, however, it would be classed as a low level language, with a strong likeness to conventional 6502 Assembly language.

To use Sweet 16, you must learn the language - and to quote "WOZ", "The op code list is short and uncomplicated". "WOZ", of course is Mr. Apple, and the creator Sweet 16.

Sweet 16 is ROM based in every Apple ][ from \$F689 to \$F7FC, and is listed in the Red Book on pages 96-99. It has its own op codes and instruction sets, and uses the SAVE and RESTORE routines from the Apple Monitor to preserve the 6502 registers when in use, allowing Sweet 16 to be used as a subroutine.

It uses the first 32 locations on zero page to set up its 16 double byte registers, and is therefore not compatible with Applesoft Basic without some additional efforts.

The original article, "Sweet 16: The 6502 Dream Machine", first appeared in Byte Magazine, November 1977 and later in the original "WOZ PAK". This article is included again as text material to help understand the use and implementation of Sweet 16.

Additionally, a trivial introductory program is included to encourage the timid to dive in, and then Andy Hertzfeld's program 'Lazarus' is reprinted from Dr. Dobbs, Volume 3, Issue 6. Lazarus is an exceptional program written to resurrect lost Integer Basic programs, and Andy's outstanding commentary allows Lazarus to do double duty as text material.

More recent examples of the use of Sweet 16 are found in the Programmer's Aid #1, in the Renumber, Append, and Relocate programs. The Programming Aid Operating Manual contains complete source assembly listings, indexed on page 65.

Finally, the friendly help of the CALL-

A.P.P.L.E. Group is available to hand-hold and counsel your adventures with Sweet 16.

The demonstration program is written to be introductory and simple, consisting of three parts:

1. Integer Basic Program
2. Machine Language Subroutine
3. Sweet 16 Subroutine

The task of the program will be to move data. Parameters of the move will be entered in the Integer Basic Program.

The "CALL 768" (\$300) at line 120, enters a 6502 machine language subroutine having the single purpose of entering a Sweet 16 subroutine and subsequently returning to Basic (addresses \$300, \$301, \$302, and \$312 respectively). The Sweet 16 subroutine of course performs the move, and is entered at hex locations \$303 to \$311 (see listing Number 3).

After the move, the screen will display three lines of data, each 8 bytes long, and await entry of a new set of parameters. The three lines of data displayed on the screen are as follows:

Line 1: The first 8 bytes of data starting at \$800, which is the fixed source of data to be moved (in this case, the string a\$).

Line 2: The first 8 bytes of data starting at the hex address entered as the destination of the move (high order byte only).

Line 3: The first 8 bytes of data starting at \$0000 (the first four Sweet 16 registers).

The display of 8 bytes of data was chosen to simply illustrate what goes on.

Integer Basic has its own way of recording the string A\$ (see page 35 in the "Red Book"). Because the name chosen for the string "A\$" is stored in 2 bytes, a total of 5 housekeeping bytes precedes the data entered as A\$, leaving only 3 additional bytes available for display. Integer Basic also adds a housekeeping byte at the end of a string, known as the "string terminator". Consequently, for the purposes of the convenience of the dis-

WOZPAK II

play, and to see the string terminator as the 8th byte, the string data entered via the keyboard should be limited to two characters, and will appear as the 6th and 7th bytes. Additionally, parameters to be entered include the number of bytes to be moved. A useful range for this demonstration would be 1-8 inclusive, but of course 1-255 will work.

Finally, the starting address of the destination of the move must be entered. Again, for simplicity, the high order byte only is entered, and the program allows a choice between Decimal 9 and the H.O.B. of program pointer 1, to avoid unnecessary problems (in this demonstration enter a decimal number between dec 9 and 144 for a 48K APPLE).

The 8 bytes of data displayed from \$00 will enable one to observe the condition of the Sweet 16 registers after a move has been accomplished, and thereby understand how the Sweet 16 program works.

From the text article "Sweet 16: The 6502 Dream Machine", it will be remembered that Sweet 16 can establish 16 double byte registers, starting at \$00. That means that Sweet 16 can use the first 32 addresses on zero page.

The "events" occurring in this demonstration program can be studied in the first 4 Sweet 16 registers, therefore the 8 byte display starting at \$0000 is large enough, for this purpose. These 4 registers are established as R0, R1, R2, and R3:

- R0 \$0000 & 0001--Sweet 16 accumulator
- R1 \$0002 & 0003--Source address
- R2 \$0004 & 0005--Destination address
- R3 \$0006 & 0007--No. of bytes to move
- .
- .
- R14 \$001C & 001D--Prior result register
- R15 \$001E & 001F--Sweet 16 prgm counter

Additionally, an examination of registers R14 and R15 will extend an understanding of Sweet 16, being fully explained in the "WOZ" text. Notice that the HOB of R14, (located at \$1D) contains \$06, which is the doubled register specification (3X2 \$06). R15, the Sweet 16 program counter contains the address of the next operation, (as it did for each step during execution of the program), which was \$0312

86

when execution ended, and 6502 machine code resumes.

To try a sample run, enter the Integer basic program as shown in Listing #1. Of course, REM statements can be omitted, and line 10 is only helpful if the machine code is to be stored on disk.

The Listing #2 must also be entered starting at \$300.

Note that a 6502 disassembly does not look like Listing #3, but Andy Hertzfeld's Sweet 16 disassembler would create a correct disassembly.

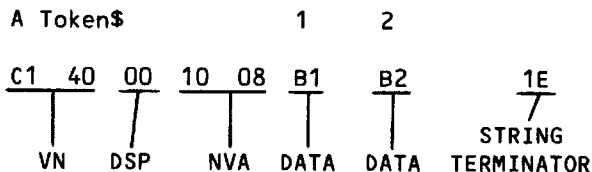
Enter RUN and RETURN  
 Enter 12 and hit RETURN (A\$ - A\$ string data)  
 Enter 8 and hit RETURN (number of bytes to be moved)  
 Enter 10 and hit RETURN (high order byte of destination)

The display should appear as follows:

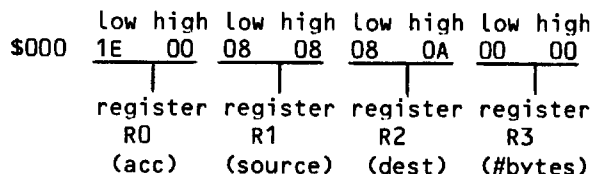
```
$0800-C1 40 00 10 08 B1 B2 1E (SOURCE)
$0A00-C1 40 00 10 08 B1 B2 1E (DEST.)
$0000-1E 00 08 08 08 0A 00 00 (Sweet 16)
```

Note that the 8 bytes stored at \$0A00 are identical to the 8 bytes starting at \$0800, indicating an accurate move of 8 bytes length has been made. It will be seen that they were moved one byte at a time starting with token C1 and ending with token 1E (if moving less than 8 bytes, the data following the moved data would be whatever exists at those locations before the move).

The bytes have the following significance, as defined on page 35 of the "Red Book":



The Sweet 16 registers are shown:



WOZPAK II

LISTING #1

```

10 PRINT "BLOAD SWEET"
20 CALL -936: DIM A$(10)
30 INPUT "ENTER STRING A$";A$
40 INPUT "ENTER # BYTES ";B
50 IF B=0 THEN 40: REM AT LEAST 1
60 POKE 778,B: REM POKE LENGTH
70 INPUT "ENTER DESTINATION";A
80 IF A>PEEK (203)-1 THEN 70
90 IF A<PEEK (205)+1 THEN 70
100 POKE 776,A: REM POKE DESTINATION
110 M=8: GOSUB 160: REM DISPLAY
120 CALL 768: REM GOTO $0300
130 M=A: GOSUB 160: REM DISPLAY
140 M=0: GOSUB 160: REM DISPLAY
150 PRINT: PRINT: GOTO 30
160 POKE 60,0: POKE 61,M
170 CALL-605: RETURN

```

The low order byte of R0, the Sweet 16 accumulator, has \$1E in it, which was the last byte moved, (the 8th).

The low order byte of the source register R1 started as \$00 and was incremented eight times, one for each byte of moved data.

The high order byte of the destination register R2 contains \$0A, which was entered as 10 (the variable A) and poked into the Sweet 16 code. The LOB of R2 was incremented exactly like R1.

Finally, register R3, the register that stores the number of bytes to be moved had been poked to 8 (the variable B) and decremented 8 times as each byte got moved, there by ending up \$0000.

By entering character strings and varying the number of bytes to moved, the Sweet 16 registers can be observed and in fact, the contents predicted.

Working with this demonstration program, and study of the text material will soon enable the writing of Sweet 16 programs to perform additional 16 bit manipulations. The unassigned op codes mentioned in the WOZ "Dream Machine" article should present a most interesting opportunity to "play".

Sweet 16 as a language---or tool---opens a new direction to Apple ][ owners without spending a dime, and it's been there all the time.

For "Appelites" who desire to learn machine language programming, Sweet 16 can be used as a starting point. Having less op codes to learn and excellent text material to use, one could be effective very soon. Enjoy....

LISTING #2

Enter code as follows:

```

300:20 89 F6 11 00 08 12 00 00 13 00 41 52
 F3 07 FB 00 60

```

A
B

LISTING #3

```

Sweet 16
$300 20 89 F6 JSR $F689
$303 11 00 08 SET R1 source address
$306 12 00 00 SET R2 destination address
$309 13 00 00 SET R3 length
 A
 B
$30C 41 LD @ R1
$30D 52 ST @ R2
$30E F3 DCR R3
$30F 07 FB BNZ $30C
$311 00 RTN
$312 60 RTN

```

Data will be poked from the Integer Basic program - "A" from Line 100  
 "B" from Line 60

---

**BLANK PAGE**

---

SWEET 16 - THE 6502 DREAM MACHINE

---

**BLANK PAGE**

---



WOZPAK II

While writing APPLE BASIC for a 6502 microprocessor I repeatedly encountered a variant of MURPHY'S LAW. Briefly stated, any routine operating on 16-bit data will require at least twice the code that it should. Programs making extensive use of 16-bit pointers (such as compilers, editors, and assemblers) are included in this category. In my case, even the addition of a few double-byte instructions to the 6502 would have only slightly alleviated the problem. What I really needed was a 6502/RCA 1800 hybrid - a powerful 8-bit data handler complemented by an easy to use processor with an abundance of 16-bit registers and excellent pointer capability. My solution was to implement a non-existent (meta) 16-bit processor in software, interpreter style, which I call SWEET 16.

SWEET 16 is based on sixteen 16-bit registers (R0-R15), actually 32 memory locations. R0 doubles as the SWEET 16 accumulator (ACC), R15 as the program counter (PC), and R14 as the status register. R13 holds compare instruction results and R12 is the subroutine return stack pointer if SWEET 16 subroutines are used. All other SWEET 16 registers are at the user's unrestricted disposal.

SWEET 16 instructions fall into register and non-register categories. The register ops specify one of the sixteen registers to be used as either a data element or a pointer to data in memory depending on the specific instruction. For example INR R5 uses R5 as data and ST @R7 uses R7 as a pointer to data in memory. Except for the SET instruction, register ops take 1 byte of code each. The non-register ops are primarily 6502 style branches with the second byte specifying a +/-127 byte displacement relative to the address of the following instruction. Providing that the prior register op result meets a specified branch condition, the displacement is added to SWEET 16 PC, effecting a branch.

SWEET 16 is intended as a 6502 enhancement package, not a stand-alone processor. A 6502 program switches to SWEET 16 mode with a subroutine call and subsequent code is interpreted as SWEET 16 instructions. The non-register op RTN returns the user program to 6502 mode after restoring the internal register contents (A, X, Y, P, and S). The following example illustrates how to use SWEET 16.

|              |               |                          |
|--------------|---------------|--------------------------|
| 300 B9 00 02 | LDA IN,Y      | get a char.              |
| 303 C9 CD    | CMP "M"       | "M" for move             |
| 305 D0 09    | BNE NOMOVE    | No, skip<br>move         |
| 307 20 89 F6 | JSR SW16      | Yes, call<br>SWEET 16    |
| 30a 41       | MLOOP LD @R1  | R1 holds<br>source addr. |
| 30B 52       | ST @R2        | R2 holds<br>dest. addr.  |
| 30C F3       | DCR R3        | Decr. length             |
| 30D 07 FB    | BNZ MLOOP     | Loop until<br>done       |
| 30F 00       | RTN           | Return to<br>6502 mode.  |
| 310 C9 C5    | MLOOP CMP "E" | "E" char?                |
| 312 D0 13    | BEQ EXIT      | Yes, exit                |
| 314 C8       | INY           | No, cont.                |

Note: Registers A, X, Y, P, and S are not disturbed by SWEET 16.

INSTRUCTION DESCRIPTIONS

The SWEET 16 opcode list is short and uncomplicated. Excepting relative branch displacements, hand assembly is trivial. All register opcodes are formed by combining two hex digits, one for the opcode and one to specify a register. For example, opcodes 15 and 45 both specify register R5 while codes 23, 27 and 29 are all ST ops. Most register ops are assigned in complementary pairs to facilitate remembering them. Thus LD and ST are opcodes 2N and 3N respectively, while LD @ and ST @ are codes 4N and 5N.

Opcodes 0 to C (hex) are assigned to the thirteen non-register ops. Except for RTN (opcode 0), BK (0A), and RS (B), the non-register ops are 6502 style branches. The second byte of a branch instruction contains a +/-127 byte displacement value (in two's complement form) relative to the address of the instruction immediately following the branch. if a specified branch condition is met by the prior register op result, the displacement is added to the PC effecting a branch. Except for BR (Branch always) and BS (Branch to Subroutine), the branch opcodes are assigned in complementary pairs, rendering them easily remembered for hand coding. For example, Branch if Plus and Branch if Minus are opcodes 4 and 5 while

WOZPAK II

Branch if Zero and Branch if NonZero are opcodes 6 and 7.

NON-REGISTER OPS  
(continued)

SWEET 16 OPCODE SUMMARY

REGISTER OPS

- 1n SET Rn, Constant (Set)
- 2n LD Rn (Load)
- 3n ST Rn (Store)
- 4n LD @Rn (Load Indirect)
- 5n ST @Rn (Store Indirect)
- 6n LDD @Rn (Load Double Indirect)
- 7n STD @Rn (Store Double Indirect)
- 8n POP @Rn (Pop Indirect)
- 9n STP @Rn (Store Pop Indirect)
- An ADD Rn (Add)
- Bn SUB Rn (Sub)
- Cn POPD @Rn (Pop Double Indirect)
- Dn CPR Rn (Compare)
- En INR Rn (Increment)
- Fn DCR Rn (Decrement)

NON-REGISTER OPS

- 00 RTN (Return to 6502 mode)
- 01 BR ea (Branch always)
- 02 BNC ea (Branch if No Carry)
- 03 BC ea (Branch if Carry)
- 04 BP ea (Branch if Plus)
- 05 BM ea (Branch if Minus)
- 06 BZ ea (Branch if zero)
- 07 BNZ ea (Branch if NonZero)
- 92

- 08 BM1 ea (Branch if Minus 1)
- 09 BNM1 ea (Branch if Not Minus 1)
- 0A BK (Break)
- 0B RS (Return from Subroutine)
- 0C BS ea (Branch to Subroutine)
- 0D (Unassigned)
- 0E (Unassigned)
- 0F (Unassigned)

REGISTERS OPS

SET Rn, Constant 1n Low High (Set)

The 2-byte constant is loaded into Rn (n=0 to F, hex) and branch conditions set accordingly. The carry is cleared.

EXAMPLE  
15 34 A0 SET R5,A034 R5 now contains A034

LD Rn 2n (Load)

The ACC (R0) is loaded from Rn and branch conditions set according to the data transferred. The carry is cleared and contents of Rn are not disturbed.

EXAMPLE  
15 34 A0 SET R5,A034  
24 LD R5 ACC now contains A034

ST Rn 3n (Store)

The ACC is stored into Rn and branch conditions set according to the data transferred. The carry is cleared and the ACC contents are not disturbed.

EXAMPLE  
25 LD R5 Copy the contents of  
36 ST R6 of R5 to R6.

WOZPAK II

LD @Rn 4n (Load indirect)

The low-order ACC byte is loaded from the memory location whose address resides in Rn and the high-order ACC byte is cleared. Branch conditions reflect the final ACC contents which will always be positive and never minus 1. The carry is cleared. After the transfer, Rn is incremented by 1.

ACC byte is loaded from location A034, the high-order byte from A035. R5 is incr. to A036.

EXAMPLE

```
15 34 A0 SET R5, A034
45 LD @R5 ACC is loaded
 from memory
 location A034
 R5 is incre-
 mented to A034
```

ST @Rn 5n (Store indirect)

The low-order ACC byte is stored into the memory location whose address resides in Rn. Branch conditions reflect the 2-byte ACC contents. The carry is cleared. After the transfer Rn is incremented by 1.

EXAMPLE

```
15 34 A0 SET R5, A034 Load point-
16 22 90 SET R6, 9022 ers R5 & R6
45 LD @R5 Move a byte with A034
 from A034 and 9022.
56 ST @R6 Both point-
 ers are in-
 cremented.
```

LDD @Rn 6n (Load double-byte indirect)

The low order ACC byte is loaded from the memory location whose address resides in Rn and Rn is then incremented by 1. The high order ACC byte is loaded from the memory location whose address resides in the incremented Rn and Rn is again incremented by 1. Branch conditions reflect the final ACC contents. The carry is cleared.

EXAMPLE

```
15 34 A0 SET R5, A034
65 LDD @R5 The low-order
```

STD @Rn 7n (Store double-byte indirect)

The low-order ACC byte is stored into memory location whose address resides in Rn and Rn is then incremented by 1. The high-order ACC byte is stored into the memory location whose address resides in the incremented Rn and Rn is again incremented by 1. Branch conditions reflect the ACC contents which are not disturbed. The carry is cleared.

EXAMPLE

```
15 34 A0 SET R5, A034 Load point-
16 22 90 Set R6, 9022 ers R5 & R6
65 STD @R6 Move double with A034
 byte from and 9022.
76 STD @R6 A034 & A035 Move double
 to 9022 and 9023. both
 pointers
 are incr.
 by 2.
```

POP @Rn 8n (Pop indirect)

The low order ACC byte is loaded from the memory location whose address resides in Rn after Rn is decremented by 1 and the high order ACC byte is cleared. Branch conditions reflect the final 2-byte ACC contents which will always be positive and never minus 1. The carry is cleared. Because Rn is decremented prior to loading the ACC, single byte stacks may be implemented with the ST @Rn and POP @Rn ops (Rn is the stack pointer).

EXAMPLE

```
15 34 A0 SET R5, A034 Init stack
10 04 00 SET R0, 4 pointer
35 ST @R5 Load 4 into
 ACC
 Push 4 onto
 stack.
```

WOZPAK II

|          |           |                          |                              |
|----------|-----------|--------------------------|------------------------------|
| 10 05 00 | SET R0, 5 | Load 5 into ACC          | (R0) to 70B6 with carry set. |
| 35       | ST @R5    | Push 5 onto stack        |                              |
| 10 06 00 | SET R0, 6 | Load 6 into ACC          |                              |
| 35       | ST @R5    | Push 6 onto stack        |                              |
| 85       | POP @R5   | Pop 6 off stack into ACC |                              |
| 85       | POP @R5   | Pop 5 off stack          |                              |
| 85       | POP @R5   | Pop 4 off stack          |                              |

SUB Rn Bn (Subtract)

The contents of Rn are subtracted from the ACC contents by performing a two's complement addition:

$$ACC = ACC + Rn + 1$$

The low order 16 bits of the subtraction are restored in the ACC. The 17th sum bit becomes the carry and other branch conditions reflect the final ACC contents. If the 16-bit unsigned ACC contents are greater than or equal to the 16-bit unsigned Rn contents then the carry is set, otherwise it is cleared. Rn is not disturbed.

EXAMPLE

|          |              |                                          |
|----------|--------------|------------------------------------------|
| 10 34 76 | SET R0, 7634 | Init R0(ACC)                             |
| 11 27 42 | SET R1, 4227 | and R1                                   |
| A1       | SUB R1       | Subtract R1 (diff = 340D with carry set) |
| A0       | SUB R0       | Clears ACC (R0)                          |

STP @Rn 9n (STORE POP indirect)

The low order ACC byte is stored into the memory location whose address resides in Rn after Rn is decremented by 1. Then the high-order ACC is stored into the memory location whose address resides in Rn after Rn is again decremented by 1. Branch conditions will reflect the 2-byte ACC contents which are not modified. STP @Rn and POP @Rn are used together to move data blocks beginning at the greatest address and working down. Additionally, single-byte stacks may be implemented with the STP @Rn ops.

EXAMPLE

|          |              |                             |
|----------|--------------|-----------------------------|
| 14 34 A0 | SET R4, A034 | Init pntrs.                 |
| 15 22 90 | SET R5, 9022 |                             |
| 84       | POP @R4      | Move byte from A033 to 9021 |
| 95       | STP @R5      |                             |
| 84       | POP @R4      | Move byte from A032 to 9020 |
| 95       | STP @R5      |                             |

POPD @Rn Cn (Pop Double-byte indirect)

Rn is decremented by 1 and the high-order ACC byte is loaded from the memory location whose address now resides in Rn. Rn is again decremented by 1 and the low-order ACC byte is loaded from the corresponding memory location. Branch conditions reflect the final ACC contents. The carry is cleared. Because Rn is decremented prior to loading each of the ACC halves, double-byte stacks may be implemented with the STD @Rn and POPD @Rn ops (Rn is the stack pointer).

EXAMPLE

|          |              |                      |
|----------|--------------|----------------------|
| 15 34 A0 | SET R5, A034 | Init stack pointer   |
| 10 12 AA | SET R0, AA12 | Load AA12 into ACC   |
| 75       | STD @R5      | Push AA12 onto stack |

ADD Rn An (Add)

The contents of Rn are added to the contents of the ACC (R0) and the low-order 16 bits of the sum restored in ACC. The 17th sum bit becomes the carry and other branch conditions reflect the final ACC contents.

EXAMPLE

|          |              |                                 |
|----------|--------------|---------------------------------|
| 10 34 76 | SET R0, 7634 | Init R0(ACC)                    |
| 11 27 42 | SET R1, 4227 | and R1                          |
| A1       | ADD R1       | Add R1 (sum= B85B, carry clear) |
| A0       | ADD R0       | Double ACC                      |

WOZPAK II

10 34 BB SET R0, BB34 Load BB34 into ACC  
 75 STD @R5 Push BB34 onto stack  
 10 56 CC SET R0, CC56 Load CC56 into ACC  
 C5 POPD @R5 Pop CC56 off stack

CPR Rn Dn (Compare)

The ACC (R0) contents are compared to Rn by performing the 16-bit binary subtraction ACC-Rn and storing the low order 16 difference bits in R13 for subsequent branch tests. If the 16-bit unsigned ACC contents are greater than or equal to the 16-bit unsigned Rn contents then the carry is set, otherwise it is cleared. No other registers, including ACC and Rn are disturbed.

EXAMPLE

15 34 A0 SET R5, A034 Pointer to memory.  
 16 BF A0 SET R6, A0BF Limit address.  
 10 00 00 LOOP SET R0, 0 Zero data.  
 75 STD @R5 Clear 2 locs, incr R5 by 2.  
 25 LD R5 Compare pointer R5 to limit R6.  
 D6 CPR R6  
 02 F8 BNC LOOP Loop if carry clear.

INR Rn En (Increment)

The contents of Rn are incremented by 1. The carry is cleared and other branch conditions reflect the incremented value.

EXAMPLE

15 34 A0 SET R5, A034 (Pointer)  
 10 00 00 SET R0, 0 Zero to R0  
 55 ST @R5 Clears loc A034 and incrs R5 to A034  
 E5 INR R5 Incr R5 to A036.  
 55 ST @R5 Clears loc A036 (not A035)

DCR Rn Fn (Decrement)

The contents of Rn are decremented by 1. The carry is cleared and other branch conditions reflect the decremented value.

EXAMPLE (Clear 9 bytes beginning at location A034)

15 34 A0 SET R5, A034 Init pointer.  
 14 09 00 SET R4, 9 Init count.  
 10 00 00 SET R0, 0 Zero ACC  
 55 LOOP ST @R5 Clear a mem byte.  
 F4 DCR R4 Decr.count.  
 07 FC BNZ LOOP Loop until zero.

NON-REGISTER INSTRUCTIONS

RTN 00 (Return to 6502 mode)

Control is returned to the 6502 and program execution continues at the location immediately following the RTN instruction. The 6502 registers and status conditions are restored to their original contents (prior to entering SWEET 16 mode).

BR ea 01 d (Branch always)

An effective address (ea) is calculated by adding the signed displacement byte (d) to the PC. The PC contains the address of the instruction immediately following the BR, or the address of the BR op plus 2. The displacement is a signed twos complement value from -128 to +127. Branch conditions are not changed.

Note that the effective address calculation is identical to that for 6502 relative branches. The hex add & subtract features of the APPLE ][ monitor may be used to calculate displacements.

d = \$80 ea = PC + 2 - 128  
 d = \$81 ea = PC + 2 - 127  
 d = \$FF ea = PC + 2 - 1  
 d = \$00 ea = PC + 2 + 0  
 d = \$01 ea = PC + 2 + 1  
 d = \$7E ea = PC + 2 +126  
 d = \$7F ea = PC + 2 +127

WOZPAK II

EXAMPLE

\$300: 01 50 BR \$352

BNC ea 02 d (Branch if No Carry)

A branch to the effective address is taken only if the carry is clear, otherwise execution resumes as normal with the next instruction. Branch conditions are not changed.

BC ea 03 d (Branch if Carry set)

A branch is effected only if the carry is set. Branch conditions are not changed.

BP ea 04 d (Branch if Plus)

A branch is effected only if the prior 'result' (or most recently transferred data) was positive. Branch conditions are not changed.

EXAMPLE (Clear mem from A034 to A03F)

|               |              |                         |
|---------------|--------------|-------------------------|
| 15 34 A0      | SET R5, A034 | Init pointer            |
| 14 3F A0      | SET R4, A03F | Init limit.             |
| 10 00 00 LOOP | SET R0, 0    |                         |
| 55            | ST @R5       | Clear mem byte, incr R5 |
| 24            | LD R4        | Compare limit to ptr.   |
| D5            | CPR R5       | Loop until done.        |
| 04 F8         | BP LOOP      |                         |

BM ea 05 d (Branch if Minus)

A branch is effected only if prior 'result' was minus (negative, MSB=1). Branch conditions are not changed.

BZ ea 06 d (Branch if Zero)

A branch is effected only if the prior 'result' was zero. Branch conditions are not changed.

96

BNZ ea 07 d (Branch if NonZero)

A branch is effected only if the prior 'result' was non-zero. Branch conditions are not changed.

BM1 ea 08 d (Branch if Minus 1)

A branch is effected only if the prior 'result' was minus 1 (\$FFFF hex). Branch conditions are not changed.

BNM1 ea 09 d (Branch if Not Minus 1)

A branch is effected only if the prior 'result' was not minus 1 (\$FFFF hex). Branch conditions are not changed.

BRK 0A (break)

A 6502 BRK (break) instruction is execution. SWEET 16 may be reentered nondestructively at SW16D after correcting the stack pointer to its value prior to executing the BRK.

RS 0B (Return from SWEET 16 subroutine)

RS terminates execution of a SWEET 16 subroutine and returns to the Sweet 16 calling program which resumes execution (in SWEET 16 mode). R12, which is the SWEET 16 subroutine return stack pointer, is decremented twice. Branch conditions are not changed.

BS ea 0C d (Branch to SWEET 16 Subroutine)

A branch to the effective address (PC +2 +d) is taken and execution is resumed in SWEET 16 mode. The current PC is pushed onto a 'SWEET 16 subroutine return address' stack whose pointer is R12, and R12 is incremented by 2. The carry is cleared and branch conditions set to indicate the current ACC contents.

EXAMPLE (Calling a 'memory move' subroutine to move A034-A03B

to 3000-3007)

WOZPAK II

```

300: 14 34 A0 SET R5, A034 Init
 ptr 1.
303: 14 3B A0 SET R4, A03B Init
 limit 1
306: 16 00 30 SET R6, 3000 Init
 ptr 2.
309: 0C 15 BS MOVE Call
 move
 subtrtn.
.
.
.
320: 45 MOVE LD @R5 Move one
321: 56 ST @R6 byte
322: 24 LD R4
323: D4 CPR R5 Test if
 done.
324: 04 FA BP MOVE Return.
326: 0B RS

```

specification is doubled to account for the 3-byte SWEET 16 registers and placed in the X-reg for indexing. Then the instruction type is determined. Register ops place the doubled register specification in the high order byte of R14 indicating the 'prior result register' to subsequent branch instructions. Non-register ops treat the register specification (right-hand half-byte) as their opcode, increment the SWEET 16 PC to point at the displacement byte of branch instructions, load the A-reg with the 'prior result register' index for branch condition testing, and clear the Y-reg.

THEORY OF OPERATION

SWEET 16 execution mode begins with a subroutine call to SW16. The user must insure that the 6502 is in hex mode upon entry. All 6502 registers are saved at this time, to be restored when a SWEET 16 RTN instruction returns control to the 6502. If you can tolerate indefinite 6502 register contents upon exit, approximately 30 usec may be saved by entering at SW16 + 3. Because this might cause an inadvertent switch from hex to decimal mode, it is advisable to enter at SW16 the first time through.

After saving the 6502 registers, SWEET 16 initializes its PC (R15) with the subroutine return address off the 6502 stack. SWEET 16's PC points to the location preceding the next instruction to be executed. Following the subroutine call are 1-, 2-, and 3-byte SWEET 16 instructions, stored in ascending memory locations like 6502 instructions. The main loop at SW16B repeatedly calls the 'execute instruction' routine at SW16C which examines on opcode for type and branches to the appropriate subroutine to execute it.

Subroutine SW16C increments the PC (R15) and fetches the next opcode, which is either a register op of the form OP REG with OP between 1 and 15 or a non-register op of the form 0 OP with OP between 0 and 13. Assuming a register op, the register

WHEN IS AN RTS REALLY A JSR?

Each instruction type has a corresponding subroutine. The subroutine entry points are stored in a table which is directly indexed into by the opcode. By assigning all the entries to a common page, only a single byte of address need be stored per routine. The 6502 indirect jump might have been used as follows to transfers control to the appropriate subroutine.

```

LDA #ADRH High-order byte.
STA IND+1
LDA OPTBL,X Low-order byte.
STA IND
JMP (IND)

```

To save code the subroutine entry address (minus 1) is pushed onto the stack, high-order byte first. A 6502 RTS (ReTurn from Subroutine) is used to pop the address off the stack and into the 6502 PC (after incrementing by 1). The net result is that the desired subroutine is reached by executing a subroutine return instruction!

OPCODE SUBROUTINES

The register op routines make use of the 6502 'zero page indexed by X' and 'indexed by X indirect' addressing modes to access the specified registers and indirect data. The 'result' of most registers ops is left in the specified register and can be sensed by subsequent branch instructions since the register specification is saved in the high-order byte of R14. This specification is changed to indicate R0 (ACC) for ADD and SUB instructions and R13

## WOZPAK II

for the CPR (compare) instruction.

Normally the high-order R14 byte holds the 'prior result register' index times 2 to account for the 2-byte SWEET16 registers and thus the LSB is zero. If ADD, SUB, or CPR instructions generate carries, then this index is incremented, setting the LSB.

The SET instruction increments the PC twice, picking up data bytes in the specified register. In accordance with 6502 convention, the low-order data byte precedes the high-order byte.

Most SWEET 16 nonregister ops are relative branches. The corresponding subroutines determine whether or not the 'prior result' meets the specified branch condition and if so update the SWEET 16 PC by adding the displacement value (-128 to +127 bytes).

The RTN op restores the 6502 register contents, pops the subroutine return stack and jumps indirect through the SWEET 16 PC. This transfers control to the 6502 at the instruction immediately following the RTN instruction.

The BK op actually executes a 6502 break instruction (BRK), transferring control to the interrupt handler.

Any number of subroutine levels may be implemented within SWEET 16 code via the BS (Branch to Subroutine) and RS (Return from Subroutine) instructions. The user must

initialize and otherwise not disturb R12 if the SWEET 16 subroutine capability is used since it is utilized as the automatic subroutine return stack pointer.

## MEMORY ALLOCATION

The only storage that must be allocated for SWEET 16 variables are 32 consecutive locations in page zero for the SWEET 16 registers, four locations to save the 6502 register contents, and a few levels of the 6502 subroutine return address stack. If you don't need to preserve the 6502 register contents, delete the SAVE and RESTORE subroutines and the corresponding subroutine calls. This will free the four page zero locations ASAV, XSAV, YSAV, and PSAV.

## USER MODIFICATIONS

You may wish to add some of your own instructions to this implementation of SWEET 16. If you use the unassigned opcodes \$0E and \$0F, remember that SWEET 16 treats these as 2-byte instructions. You may wish to handle the break instruction as a SWEET 16 call, saving two bytes of code each time you transfer into SWEET 16 mode. Or you may wish to use the SWEET 16 BK (Break) op as a 'CHAROUT' call in the interrupt handler. You can perform absolute jumps within SWEET 16 by loading the ACC (R0) with the address you wish to jump to (minus 1) and executing a ST R15 instruction.



LAZARUS

A Program to Resurrect BASIC Programs  
on the APPLE ][ Computer

By Andy Hertzfeld  
Reprinted from Dr. Dobb's Journal of  
Computer Calisthenics & Orthodontia,  
Box E, Menlo Park, CA 94025

---

**BLANK PAGE**

---

## WOZPAK II

Many unfortunate souls have experienced the frustration which occurs when you realize that you have just accidentally erased the program that you were working on. Kicking the computer may provide some emotional satisfaction, but it won't bring your program back. This article describes Lazarus, a short machine language program which can resurrect inadvertently erased BASIC programs on the APPLE ][ computer.

When an APPLE ][ BASIC program is erased, it is usually not destroyed. It is probably still sitting around somewhere in memory; however, the BASIC interpreter no longer knows where to find it, so it is effectively lost. Lazarus scans through memory trying to find the largest valid BASIC program that it can. If it finds one it resets the interpreter's pointers to point to the found program, thereby resurrecting it. If no valid program can be found an error message is printed. In either case, control is returned to the BASIC interpreter.

Of course, things are not quite this simple. If there are many program fragments floating around the workspace, Lazarus might not resurrect the one you are interested in. If you had performed some deletions of low-numbered statements prior to the erasure, Lazarus may restore the deleted lines along with the rest of the program. If your program was located in an unusual part of memory, Lazarus may not know where to look for it. Finally, it is possible but unlikely that Lazarus could introduce spurious statements at the beginning and end of the resuscitated program. In practice these problems are rare; Lazarus should be successful most of the time.

## THE DETAILS

To understand how Lazarus works, you must first know something about the internal representation of an APPLE ][ integer BASIC program. We will examine this briefly here; for a more detailed exposition, see Steve Wozniak's article in Dr. Dobb's #23 (vol. 3, issue 3) or the APPLE reference material, when they finally get around to putting it out (the current reference manual isn't very helpful).

The BASIC "workspace" is the area of memory used to store the currently active BASIC program and its associated variables. On the APPLE ][, it is defined by two zero-page pointers: LOMEM, which points to the lowest address's byte in the workspace and HIMEM, which points one past the highest. Program statements are placed into memory starting at HIMEM, and are pushed down as each successive statement is entered. Another pointer called PP (for program pointer) is used to point to the first, lowest numbered statement. PP decreases as the program grows; a program is erased by setting PP to HIMEM. Thus, all Lazarus has to do is scan memory from LOMEM to HIMEM and isolate the beginning and end of the longest BASIC program residing there. To accomplish the resurrection, it simply stores the address of the start of the program into HIMEM.

As each line of the BASIC program is entered, it is translated into a compact internal form consisting of four parts. First there is a length byte indicating the total length of the statement. It is followed by a 2-byte line number, low order byte first. Next comes a string of code syllables or tokens which make up the body of the statement. Finally, there is a special end-of-statement token (\$01) at the end of every statement.

Knowing this information, it is not too hard to find a BASIC program segment; simply scan through memory searching for an end-of-statement token. When one is found, we can check if it ends a statement by examining length bytes. The fact that the statement's line numbers must be in ascending order is used to perform an additional validity check. The program segment ends as soon as this structure is violated.

There may be valid fragments floating around the workspace (like pieces of the Star Trek program you played an hour ago); Lazarus will resurrect the largest one, since that is usually the one we are interested in. Since that is not always the one we want, another entry point is provided that causes the last segment (which tends to be the most recent program worked on) to be chosen.

## WOZPAK II

Lazarus was implemented using SWEET 16, a very useful little 16-bit virtual machine that is interpreted by the APPLE-][ firmware. SWEET 16 saves a lot of code at the expense of a significantly longer execution time. Since Lazarus does a lot of 16-bit operations, SWEET 16 was able to spare me a large amount of tedious coding. (See Wozniak's article elsewhere in this volume for a detailed description of SWEET 16).

## USING LAZARUS

The code for Lazarus is relocatable since all of its branches are relative and the only local storage it uses is in page zero. \$300 seems to be a convenient place to load it. You can load it for the first time by going into the monitor mode and typing:

```
300: A9 FF D0 02 A9 00 85 F0
308: 85 F1 20 89 F6 B0 3A E0
310: 35 13 04 00 16 4A 00 17
318: 4C 00 66 36 32 67 37 22
320: D7 03 59 42 D5 07 F8 22
328: F0 38 39 E8 48 D3 02 18
330: A9 34 44 D5 07 12 27 D4
338: 05 0E 24 F0 39 68 31 E4
340: 64 D1 05 04 29 38 01 E3
348: 22 F0 34 D9 06 D1 F4 F4
350: F4 11 05 00 84 D1 06 0D
358: E1 10 FF 00 D1 02 C0 24
360: D6 05 BC 01 EF 29 32 E2
368: B4 31 18 F0 00 68 06 04
370: 21 DA 02 AB 21 3A 24 3B
378: 29 3C 01 A3 2A 06 11 11
380: CA 00 2B 71 11 4C 00 2C
388: E0 71 00 20 3A FF 4C 03
390: E0 00 20 2D FF 4C 03 E0
```

Once it has been manually loaded, it can be saved by typing "300.3A0w" and then subsequently loaded back in by typing "300.3A0R". (Editor's note: For disk save and load, use "BSAVE LAZARUS, A\$300,L\$A0" and "BLOAD LAZARUS", respectively).

Lazarus is very easy to use. After cursing yourself out a few times for accidentally erasing your program, return to monitor and load Lazarus. Then return to BASIC and type "CALL 768" or "CALL 772". The former will cause the longest program segment in the workspace to be restored; the latter will cause the last to be selected. When it returns, do a "LIST" to see what is found. If nothing could be dredged up, an error message is printed. If both entry points recover the wrong program, you can adjust LOMEM and HIMEM accordingly and try again. If all else fails, you can still always kick the computer.

A useful extension to Lazarus would be to recover the variable table along with the program, which could be very important in some cases. I would like to thank Bruce Tognazzini for providing the initial idea for this program.

A complete source listing follows...

WOZPAK II

```

4 * LAZARUS SOURCE LISTING

6 LOMEM EQU $4A
7 HIMEM EQU $4C
8 PP EQU $CA
9 BASIC EQU $E000
10 BELL EQU $FF3A
11 ERR EQU $FF2D
12 SW16 EQU $F689 SWEET 16 ENTRY
13 **** SWEET 16 EQUATES ****
14 4 EQU 4
15 5 EQU 5
16 FO EQU $FO
17 FF EQU $FF
18 R0 EQU $0
19 R1 EQU $1 AUXILIARY
20 R2 EQU $2 MAIN SCAN POINTER
21 R3 EQU $3 CONTAINS $4 FOR COMPARISON
22 R4 EQU $4 AUXILIARY; BACKWARDS SCAN POINTER
23 R5 EQU $5 CONTAINS $1 FOR COMPARISONS
24 R6 EQU $6 LOMEM
25 R7 EQU $7 HIMEM
26 R8 EQU $8 USED IN FORWARD SCAN
27 R9 EQU $9 USED IN FORWARD SCAN
28 R10 EQU 10 MAXIMUM SIZE FOUND SO FAR
29 R11 EQU 11 PP OF MAX SEGMENT
30 R12 EQU 12 HIMEM OF MAX SEGMENT
31 *
32 ORG $300
33 *
0300: A9 FF 34 LDA #$FF SET FLAG TO ALL ONES
0302: D0 02 35 BNE STORE ALWAYS TAKEN.
0304: A9 00 36 LDA #$00 SET FLAG TO ALL ZEROS.
0306: 85 F0 37 STORE STA $FO
0308: 85 F1 38 STA $F1 SET THE SWITCH AT $FO
39 *
40 * INITIALIZE REGISTERS, ETC.
41 *
030A: 20 89 F6 42 JSR SW16
030D: B0 43 SUB R0 ZERO THE ACCUMULATOR
030E: 3A 44 ST R10 ZERO MAX SIZE.
030F: E0 45 INR R0
0310: 35 46 ST R5 SET R5 TO 1.
0311: 13 04 00 47 SET R3,4 SET R3 TO 4.
0314: 16 4A 00 48 SET R6,LOMEM GET ADDRESS OF LOMEM.
0317: 17 4C 00 49 SET R7,HIMEM HIMEM, TOO.
031A: 66 50 LDD @R6
031B: 36 51 ST R6 SET R6 TO LOMEM.
031C: 32 52 ST R2 INITIALIZE SCAN PTR. THERE, TOO.
031D: 67 53 LDD @R7
031E: 37 54 ST R7 SET R7 TO HIMEM.
55 *
56 * THE MAIN LOOP STARTS HERE. WE
57 * SCAN FOR AN END-OF-STATEMENT TOKEN ($01).
58 *
031F: 22 59 MAINLOOP LD R2 TEST TO SEE IF WE'RE PAST
0320: D7 60 CPR R7 HIMEM; IF WE ARE, WE ARE
0321: 03 59 61 BC ALLDONE FINALLY DONE.
0323: 42 62 LD @R2 PICK UP THE NEXT BYTE.

```

```

 WOZPAK II
0324: D5 64 CPR R5 IS IT THE EOS TOKEN?
0325: 07 F8 65 BNZ MAINLOOP IF ITS NOT, CONTINUE THE SCAN.
0327: 22 66 LD R2 INITIALIZE R8 & R9
0328: F0 67 DCR R0 TO POINT TO THE
0329: 38 68 ST R8 END-OF-STATEMENT TOKEN.
032A: 39 69 ST R9
70 *
71 * AT THIS POINT A CANDIDATE FOR A STATEMENT
72 * HAS BEEN FOUND. THE FOLLOWING LOOP PERFORMS
73 * VALIDITY CHECKS ON SUCCESSIVE STATEMENTS
74 * UNTIL AN INVALID ONE IS FOUND OR HIMEM IS
75 * REACHED. R9 POINTS TO THE LAST BYTE OF
76 * THE MOST RECENT VALID STATEMENT.
77 *
032B: E8 78 CHKLOOP INR R8 BUMP R8 TO THE NEW LENGTH BYTE
032C: 48 79 LD @R8 PICK UP THE LENGTH BYTE.
032D: D3 80 CPR R3 MAKE SURE ITS GREATER THAN 4.
032E: 02 18 81 BNC INVALID IF ITS NOT, ITS INVALID.
0330: A9 82 ADD R9
0331: 34 83 ST R4 R4 NOW HOLDS THE END ADDRESS.
0332: 44 84 LD @R4 PICK UP THE LAST BYTE OF STATEMENT.
0333: D5 85 CPR R5 IT BETTER BE AN EOS TOKEN.
0334: 07 12 86 BNZ INVALID IF ITS NOT, ITS INVALID.
87 *
88 * AT THIS POINT, R8 POINTS ONE PAST THE BEGINNING OF A
89 * VALID STATEMENT AND R4 POINTS TO THE START OF THE NEXT,
90 * POSSIBLY VALID ONE. WE PERFORM LINE NO. VALIDITY CHECKS
91 * AND TEST AGAINST HIMEM.
92 *
0336: 27 93 LD R7 PICK UP HIMEM AND SEE IF
0337: D4 94 CPR R4 WE'VE PASSED IT.
0338: 05 0E 95 BM INVALID IF WE HAVE, ITS INVALID.
033A: 24 96 LD R4
033B: F0 97 DCR R0 ITS VALID, SO MARK IT
033C: 39 98 ST R9 IN R9.
033D: 68 99 LDD @R8 PICK UP THE LINE NUMBER OF THE
033E: 31 100 ST R1 OLD LINE AND SAVE IN R1.
033F: E4 101 INR R4
0340: 64 102 LDD @R4 PICK UP THE NEWER LINE NUMBER
0341: D1 103 CPR R1 AND COMPARE IT WITH THE OLDER ONE.
0342: 05 04 104 BM INVALID IF NEW IS LESS THAN OLD, ITS NO GOOD.
0344: 29 105 LD R9 THE STATEMENT'S OK SO
0345: 38 106 ST R8 UPDATE R8 WITH R9.
0346: 01 E3 107 BR CHKLOOP GO EXAMINE THE NEXT STATEMENT.
108 *
109 * AT THIS POINT, WE HAVE REACHED THE END OF A VALID
110 * PROGRAM SEGMENT;
111 * R9 POINTS TO THE LAST BYTE OF THE LAST VALID STATEMENT
112 * R2 POINTS ONE PAST THE LAST BYTE OF THE FIRST VALID STATEMENT
113 * IF R2=R9+1, WE DIDN'T FIND ANYTHING
114 *
0348: 22 115 INVALID LD R2 POINT R4 TO THE LAST BYTE OF
0349: F0 116 DCR R0 OF THE LAST VALID STATEMENT.
034A: 34 117 ST R4
034B: D9 118 CPR R9 IF R9=R4 THEN NOTHING WAS FOUND
034C: 06 D1 119 BZ MAINLOOP SO FORGET IT AND GO CONTINUE SCAN.
034E: F4 120 DCR R4
034F: F4 121 DCR R4 SUBTRACT 3 FROM R4 TO PREPARE
0350: F4 122 DCR R4 FOR THE BACKWARD SCAN.
0351: 11 05 00 123 SET R1,5 R1 WILL HOLD THE TENTATIVE LENGTH.

```

104

WOZPAK II

```

125 *
126 * IN THE FOLLOWING LOOP WE SCAN BACK, SEARCHING FOR THE
127 * BEGINNING OF THE SEGMENT'S INITIAL STATEMENT.
128 *
0354: 84 129 SCANLOOP POP @R4 PICK UP THE NEXT BYTE.
0355: D1 130 CPR R1 COMPARE WITH THE LENGTH COUNT.
0356: 06 OD 131 BZ DONE IF THEY'RE EQUAL, WE'RE DONE.
0358: E1 132 INR R1 INCREMENT R1 FOR THE NEXT ITERATION.
0359: 10 FF 00 133 SET RO,FF TEST IF WE'VE GONE 256 BYTES.
035C: D1 134 CPR R1 IF WE HAVE, ITS TOO LONG
035D: 02 CO 135 BNC MAINLOOP SO GIVE UP THE SEGMENT.
035F: 24 136 LD R4 MAKE SURE LOMEM ISN'T PASSED.
0360: D6 137 CPR R6
0361: 05 BC 138 BM MAINLOOP IF IT HAS, GIVE UP.
0363: 01 EF 139 BR SCANLOOP GO CHECK THE NEXT BYTE.
140 *
141 * AT THIS POINT, WE HAVE FOUND A VALID PROGRAM SEGMENT. R4
142 * POINTS TO THE FIRST BYTE OF THE SEGMENT; R9 POINTS TO
143 * THE LAST BYTE.
144 *
0365: 29 145 DONE LD R9 MAKE R2 POINT PAST THE RECENTLY
0366: 32 146 ST R2 FOUND SEGMENT TO CONTINUE THE
0367: E2 147 INR R2 SCAN IN THE MAIN LOOP.
0368: 84 148 SUB R4 COMPUTE THE LENGTH OF THE SEGMENT
0369: 31 149 ST R1 AND SAVE IT IN R1.
036A: 18 FO 00 150 SET R8,FO GET THE ADDRESS OF THE FLAG.
036D: 68 151 LDD @R8 LOAD THE FLAG.
036E: 06 04 152 BZ UPDATE IF FLAG=0, ALWAYS DO IT.
0370: 21 153 LD R1 RESTORE THE SIZE
0371: DA 154 CPR R10 AND COMPARE WITH THE MAX SO FAR.
0372: 02 AB 155 BNC MAINLOOP IF ITS SMALLER, WE DON'T WANT IT.
0374: 21 156 UPDATE LD R1 RESTORE THE SIZE
0375: 3A 157 ST R10 AND UPDATE THE MAX SIZE WITH IT.
0376: 24 158 LD R4
0377: 3B 159 ST R11 UPDATE THE MAX PP.
0378: 29 160 LD R9
0379: 3C 161 ST R12 UPDATE THE MAX HIMEM.
037A: 01 A3 162 BR MAINLOOP GO FIND THE NEXT SEGMENT.
163 *
164 * WE REACH THIS POINT WHEN THE MAIN LOOP HAS RUN ITS COURSE
165 * (I.E., THE MAIN SCAN POINTER HAS REACHED HIMEM). SET PP AND
166 * HIMEM APPROPRIATELY AND THEN RETURN TO THE BASIC
167 * INTERPRETER. PRINT "ERR" IF NOTHING FOUND.
168 *
037C: 2A 169 ALLDONE LD R10 GET THE SIZE OF WHAT WE FOUND.
037D: 06 12 170 BZ ERROR IF ITS ZERO, SIGNAL AN ERROR.
037F: 11 CA 00 171 SET R1,PP GET THE ADDRESS OF PP.
0382: 2B 172 LD R11
0383: 71 173 STD @R1 SET PP TO START OF PROGRAM.
0384: 11 4C 00 174 SET R1,HIMEM GET THE ADDRESS OF HIMEM.
0387: 2C 175 LD R12
0388: E0 176 INR R0 INCREMENT IT BY 1.
0389: 71 177 STD @R1 SET HIMEM TO POINT ONE PAST PROGRAM.
038A: 00 178 RTN RETURN TO 6502 MODE.
038B: 20 3A FF 179 JSR BELL RING THE BELL TO WAKE UP USER.
038E: 4C 03 E0 180 JMP BASIC+3 JUMP BACK TO INTERPRETER.
0391: 00 181 ERROR RTN RETURN TO 6502 MODE.
0392: 20 2D FF 182 JSR ERR PRINT THE ERROR MESSAGE (& RING BELL)
0395: 4C 03 E0 183 JMP BASIC+3 RETURN TO THE BASIC INTERPRETER.

```

---

**BLANK PAGE**

---



FLOATING POINT PACKAGE

---

**BLANK PAGE**

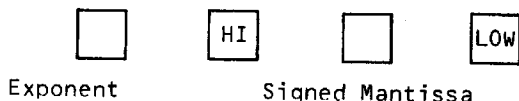
---

WOZPAK II

The mantissa-exponent, or 'floating point' numerical representation is widely used by computers to express values with a wide dynamic range. With floating point representation, the number  $7.5 \times 10^{22}$  requires no more memory to store than the number 75 does. We have allowed for binary floating point arithmetic on the APPLE ][ computer by providing a useful subroutine package in ROM, which performs the common arithmetic functions. Maximum precision is retained by these routines and overflow conditions such as 'divide by zero' are trapped for the user. The 4-byte floating point number representation is compatible with future APPLE products such as floating point BASIC.

A small amount of memory in Page Zero is dedicated to the floating point workspace, including the two floating-point accumulators, FP1 and FP2. After placing operands in these accumulators, the user calls subroutines in the ROM which perform the desired arithmetic operations, leaving results in FP1. Should an overflow condition occur, a jump to location \$3F5 is executed, allowing a user routine to take appropriate action.

FLOATING POINT REPRESENTATION



1. Mantissa

The floating point mantissa is stored in two's complement representation with the sign at the most significant bit (MSB) position of the high-order mantissa byte. The mantissa provides 24 bits of precision, including sign, and can represent 24-bit integers precisely. Extending precision is simply a matter of adding bytes at the low order end of the mantissa.

Except for magnitudes less than  $2^{-128}$  (which lose precision) mantissa are normalized by the floating point routines to retain maximum precision. That is, the numbers are adjusted so that the upper two high-order mantissa bits are unequal.

HIGH-ORDER MANTISSA BYTE  
 01.XXXXXX Positive mantissa.  
 10.XXXXXX Negative mantissa.

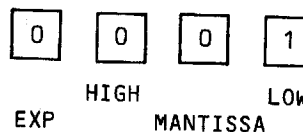
00.XXXXXX Unnormalized mantissa,  
 11.XXXXXX Exponent = -128.

2. Exponent.

The exponent is a binary scaling factor (power of two) which is applied to the mantissa. Ranging from -128 to +127, the exponent is stored in standard two's complement representation except for the sign bit which is complemented. This representation allows direct comparison of exponents, since they are stored in increasing numerical sequence. The most negative exponent, corresponding to the smallest magnitude, -128, is stored as \$00 (\$ means hexadecimal) and the most positive, +127, is stored as \$FF (all ones).

| EXPONENT | STORED AS       |
|----------|-----------------|
| +1       | 10000001 (\$81) |
| +2       | 10000010 (\$82) |
| +3       | 10000011 (\$83) |
| -1       | 01111111 (\$7F) |
| -2       | 01111110 (\$7E) |
| -3       | 01111101 (\$7D) |

The smallest magnitude which can be represented is  $2^{-150}$ .



The largest positive magnitude which can be represented is  $+2^{128-1}$ .



FLOATING POINT REPRESENTATION EXAMPLES

| DECIMAL NUMBER | HEX EXPONENT | HEX MANTISSA |
|----------------|--------------|--------------|
| + 3            | 81           | 60 00 00     |
| + 4            | 82           | 40 00 00     |
| + 5            | 82           | 50 00 00     |
| + 7            | 82           | 70 00 00     |
| +12            | 83           | 60 00 00     |
| +15            | 83           | 78 00 00     |
| +17            | 84           | 44 00 00     |
| +20            | 84           | 50 00 00     |
| +60            | 85           | 78 00 00     |

WOZPAK II

|     |    |          |
|-----|----|----------|
| - 3 | 81 | A0 00 00 |
| - 4 | 81 | 80 00 00 |
| - 5 | 82 | B0 00 00 |
| - 7 | 82 | 90 00 00 |
| -12 | 83 | A0 00 00 |
| -15 | 83 | 88 00 00 |
| -17 | 84 | BC 00 00 |
| -20 | 84 | B0 00 00 |
| -60 | 85 | 88 00 00 |

point form.

Entry: The two addends are in FP1 and FP2 respectively. For maximum precision, both should be normalized.

Uses: SWPALGN, ADD, NORM, RTLOG.

Exit: The normalized sum is left in FP1. FP2 contains the addend of greatest magnitude. E is altered but sign is not. The A-REG is altered and the X-REG is cleared. The sum mantissa is truncated to 24 bits.

Caution: Overflow may result if the sum is less than  $-2^{128}$  or greater than  $+2^{128} - 1$ . If so, a jump to location \$3F5 is executed leaving 0 in X1, and twice the proper sum in the mantissa M1. The sign bit is left in the carry, 0 for positive, 1 for negative.

FLOATING POINT SUBROUTINE DESCRIPTIONS

FCOMPL subroutine (address \$F4A4)

Purpose: FCOMPL is used to negate floating point numbers.

Entry: A normalized or unnormalized value is in FP1 (floating point accumulator 1).

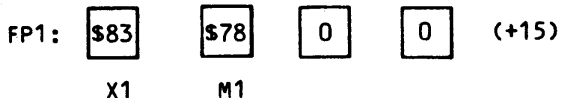
Uses: NORM, RTLOG.

Exit: The value in FP1 is negated and then normalized to retain precision. The 3-byte FP1 extension, E, may also be altered but FP2 and SIGN are not disturbed. The 6502 A-REG is altered and the X-REG is cleared. The Y-REG is not disturbed.

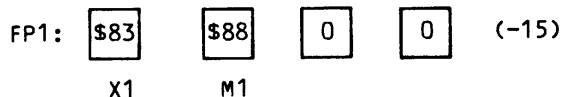
Caution: Attempting to negate  $-2^{128}$  will result in an overflow since  $+2^{128}$  is not representable, and a jump to location \$3F5 will be executed, with the following contents in FP1.



Example: Prior to calling FCOMPL, FP1 contains +15.

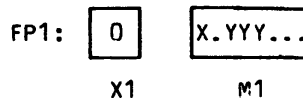


After calling FCOMPL as a subroutine, FP1 contains -15.

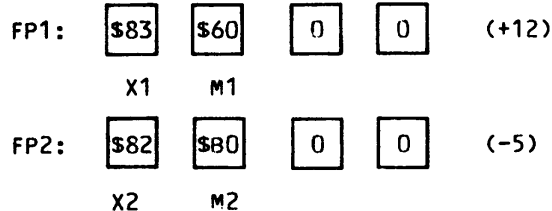


FADD subroutine (address \$F46E)

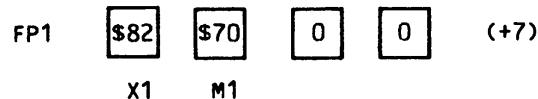
Purpose: To add two numbers in floating point form.



(For carry=0, true sum= $X.YYY... \times 2^{128}$ )  
Example: Prior to calling FADD, FP1 contains +12 and FP2 contains -5.



After calling FADD, FP1 contains +7 (FP2 contains +12).



FSUB subroutine (address \$F468)

Purpose: To subtract two floating point numbers.

Entry: The minuend is in FP1 and the subtrahend is in FP2. Both should be normalized to retain maximum precision prior to calling FSUB.

Uses: FCOMPL, ALGNSWP, FADD, ADD, NORM, RTLOG.

Exit: The normalized difference is in FP1

WOZPAK II

with the mantissa truncated to 24 bits. FP2 holds either the minuend or the negated subtrahend, whichever is of greater magnitude. E is altered but SIGN and SCR are not. The A-REG is altered and the X-REG is cleared. The Y-REG is not disturbed.

Cautions: An exit to location \$3F5 is taken if the result is less than  $-2^{128}$  or greater than  $+2^{128}-1$ , or if the subtrahend is  $-2^{128}$ .

Example: Prior to calling FSUB, FP1 contains +7 (minuend) and FP2 contains -5 (subtrahend).

FP1: 

|      |      |   |   |
|------|------|---|---|
| \$82 | \$70 | 0 | 0 |
|------|------|---|---|

 (+12)  
X1 M1

FP2: 

|      |      |   |   |
|------|------|---|---|
| \$82 | \$80 | 0 | 0 |
|------|------|---|---|

 (- 5)  
X2 M2

After calling FSUB, FP1 contains +12 and FP2 contains +7.

FP1: 

|      |      |   |   |
|------|------|---|---|
| \$83 | \$60 | 0 | 0 |
|------|------|---|---|

 (+12)  
X1 M1

FMUL subroutine (address \$F48C)

Purpose: To multiply floating point numbers.

Entry: The multiplicand and multiplier must reside in FP1 and FP2 respectively. Both should be normalized prior to calling FMUL to retain maximum precision.

Uses: MD1, MD2, RTLOG1, ADD, MDEND.

Exit: The signed normalized floating point product is left in FP1. M1 is truncated to contain the 24 most significant mantissa bits (including sign). The absolute value of the multiplier mantissa (M2) is left in FP2. E, SIGN, and SCR are altered. The A- and X-REGs are altered and the Y-REG contains \$FF upon exit.

Cautions: An exit to location \$3F5 is taken if the product is less than  $-2^{128}$  or greater than  $+2^{128}-1$ .

Notes: FMUL will run faster if the absolute value of the multiplier mantissa contains fewer '1's than the absolute value of the multiplicand mantissa.

Example: Prior to calling FMUL, FP1 contains +12 and FP2 contains -5.

FP1: 

|      |      |   |   |
|------|------|---|---|
| \$83 | \$60 | 0 | 0 |
|------|------|---|---|

 (+12)  
X1 M1

FP2: 

|      |      |   |   |
|------|------|---|---|
| \$82 | \$80 | 0 | 0 |
|------|------|---|---|

 (- 5)  
X2 M2

After calling FMUL, FP1 contains -60 and FP2 contains +5.

FP1: 

|      |      |   |   |
|------|------|---|---|
| \$85 | \$88 | 0 | 0 |
|------|------|---|---|

 (-60)  
X1 M1

FP2: 

|      |      |   |   |
|------|------|---|---|
| \$82 | \$50 | 0 | 0 |
|------|------|---|---|

 (+ 5)  
X2 M2

FDIV subroutine (address \$F4B2)

Purpose: To perform division of floating point numbers.

Entry: The normalized dividend is in FP2 and the normalized divisor is in FP1.

Exit: The signed normalized floating point quotient is left in FP1. The mantissa (M1) is truncated to 24 bits. The 3-bit M1 extension (E) contains the absolute value of the divisor mantissa. MD2, SIGN, and SCR are altered. The A- and X-REGs are altered and the Y-REG is cleared.

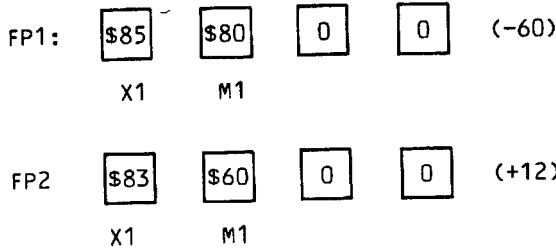
Uses: MD1, MD2, MDEND.

Cautions: An exit to location \$3F5 is taken if the quotient is less than  $-2^{128}$  or greater than  $+2^{128}-1$ .

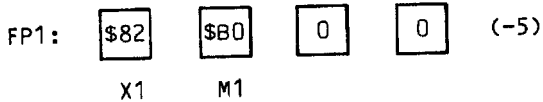
WOZPAK II

Notes: MD2 contains the remainder mantissa (equivalent to the MOD function). The remainder exponent is the same as the quotient exponent, or 1 less if the dividend mantissa magnitude is less than the divisor mantissa magnitude.

Example: Prior to calling FDIV, FP1 contains -60 (dividend), and FP2 contains +12 (divisor).



After calling FMUL, FP1 contains -5 and M2 contains 0.



FLOAT Subroutine (address \$F451)

Purpose: To convert integers to floating point representation.

Entry: A signed (two's complement) 2-byte integer is stored in M1 (high-order byte) and M1+1 (low-order byte). M1+2 must be cleared by user prior to entry.

Uses: NORM1.

Exit: The normalized floating point equivalent is left in FP1. E, FP2, SIGN, and SCR are not disturbed. The A-REG contains a copy of the high-order mantissa byte upon exit but the X- and Y-REGs are not disturbed. The carry is cleared.

Notes: To float a 1-byte integer, place it in M1+1 and clear M1 as well as M1+2 prior to calling FLOAT.

FLOAT takes approximately 3 msec. longer to convert zero to floating point form than other arguments. The user may check for zero prior

to calling FLOAT and increase throughput.

```

*
* LOW-ORDER INT. BYTE IN A-REG
* HIGH-ORDER BYTE IN Y-REG
*
85 FA XFLOAT STA M1+1
84 F9 STY M1 INIT MANT1
A0 00 LDY #0
84 FB STY M1+2
05 D9 ORA M1 CHK BOTH
BYES FOR
D0 03 BNE TOFLOAT ZERO
85 F8 STA X1 IF SO CLR X1
60 RTS AND RETURN
4C 51 F4 TOFLOAT JMP FLOAT ELSE FLOAT
INTEGER

```

Example: Float +274 (\$0112 hex)

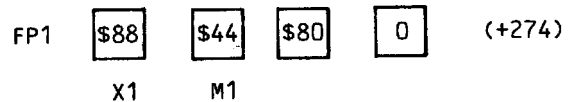
CALLING SEQUENCE

```

A0 01 LDY #01 HIGH-ORDER
INTEGER BYTE
A9 12 LDA #12 LOW-ORDER
INTEGER BYTE
84 F9 STY M1
85 FA STA M1+1
A9 00 LDA #00
85 F8 STA M1+2
20 51 F4 JSR FLOAT

```

Upon returning from FLOAT, FP1 contains the floating point representation of +274.



FIX subroutine (address \$F640)

Purpose: To extract the integer portion of a floating point number with truncation (ENTIER function).

Entry: A floating point value is in FP1. It need not be normalized.

Uses: RTAR.

Exit: The two-byte signed two's complement representation of the integer portion is left in M1 (high-order byte) and M1+1 (low-order byte). The floating point values +24.63 and -61.2 are converted to the integers

WOZPAK II

+24 and -61 respectively. FP1 and E are altered but FP2, E, SIGN, and SCR are not. The A- and X-REGs are altered but the Y-REG is not.

Example: The floating point value +274 is in FP1 prior to calling FIX.

FP1: 

|      |      |      |   |
|------|------|------|---|
| \$88 | \$44 | \$80 | 0 |
|------|------|------|---|

 (+274)  
           X1      M1

After calling FIX, M1 (high-order byte) and M1+1 (low-order byte) contain the integer representation of +274 (\$0112).

FP1: 

|      |      |      |   |
|------|------|------|---|
| \$8E | \$01 | \$12 | 0 |
|------|------|------|---|

  
           X1      M1

Note: FP1 contains an unnormalized representation of +274 upon exit.

NORM Subroutine (address \$F463)

Purpose: To normalize the value in FP1, thus insuring maximum precision.

Entry: A normalized or unnormalized value is in FP1.

Exit: The value in FP1 is normalized. A zero mantissa will exit with X1=0 (2 exponent). If the exponent on exit is -128 (X1=0) then the mantissa (M1) is not necessarily normalized (with the two high-order mantissa bits unequal). E, FP2, SIGN, AND SCR are not disturbed. The A-REG is disturbed but the X- and Y-REGs are not. The carry is set.

Example: FP1 contains +12 in unnormalized form (as .0011 x 2 ).

FP1: 

|      |      |   |   |
|------|------|---|---|
| \$86 | \$0C | 0 | 0 |
|------|------|---|---|

 (+12)  
           x1      M1

Upon exit from NORM, FP1 contains +12 in normalized form (as 1.1 x 2 ).

FP1: 

|      |      |   |   |
|------|------|---|---|
| \$83 | \$60 | 0 | 0 |
|------|------|---|---|

 (+12)  
           X1      M1

NORM1 subroutine (address \$F455)

Purpose: To normalize a floating point value in FP1 when it is known the exponent is not -128 (X1 =0) upon entry.

Entry: An unnormalized number is in FP1. The exponent byte should not be 0 for normal use.

Exit: The normalized value is in FP1. E, FP2, SIGN, and SCR are not disturbed. The A-REG is altered but the X- and Y-REGs are not.

ADD Subroutine (address \$F425)

Purpose: To add the two mantissas (M1 and M2) as 3-byte integers.

Entry: Two mantissas are in M1 (through M1+2) and M2 (through M2+2). They should be aligned, that is with identical exponents, for use in the FADD and FSUB subroutines.

Exit: the 24-bit integer sum is in M1 (high-order byte in M1, low-order byte in M1+2). FP2, X1, E, SIGN and SCR are not disturbed. The A-REG contains the high-order byte of the sum, the X-REG contains \$FF and the Y-REG is not altered. The carry is the '25th' sum bit.

Example: FP1 contains +5 and FP2 contains +7 prior to calling ADD.

FP1: 

|      |      |   |   |
|------|------|---|---|
| \$82 | \$50 | 0 | 0 |
|------|------|---|---|

 (+5)  
           X1      M1

FP2: 

|      |      |   |   |
|------|------|---|---|
| \$82 | \$70 | 0 | 0 |
|------|------|---|---|

 (+7)

Upon exit, M1 contains the overflow value for +12. Note that the sign bit is incorrect. This is taken care of with a call to the right shift routine.

FP: 

|      |      |   |   |
|------|------|---|---|
| \$82 | \$C0 | 0 | 0 |
|------|------|---|---|

 (+12)

ABSWAP Subroutine (address \$F437)

Purpose: To take the absolute value of FP1 and then swap FP1 with FP2.

WOZPAK II

Note that two sequential calls to ABSWAP will take the absolute values of both FP1 and FP2 in preparation for a multiply or divide.

Entry: FP1 and FP2 contain floating point values.

Exit: The absolute value of the original FP1 contents are in FP2 and the original FP2 contents are in FP1. The least significant bit of SIGN is complemented if a negation takes place (if the original FP1 contents are negative) by means of an increment. SCR and E are used. The A-REG contains a copy of X2, the X-REG is cleared, and the Y-REG is not altered.

RTAR Subroutine (address \$F47D)

Purpose: To shift M1 right one bit position while incrementing X1 to compensate for scale. This is roughly the opposite of the NORM subroutine.

Entry: A normalized or unnormalized floating point value is in FP1.

Exit: The 6-byte field MANT1 and E is shifted right one bit arithmetically and X1 is incremented by 1 to retain proper scale. The sign bit of MANT1 (MSB of M1) is unchanged. FP2, SIGN, and SCR are not disturbed. The A-REG contains the least significant byte of E (E+2), the X-REG is cleared, and the Y-REG is not disturbed.

Caution: If X1 increments of 0 (overflow) then an exit to location \$3F5 is taken, the A-REG contains the high-order MANT1 byte, M1 and X1 is cleared. FP2, SIGN, SCR, and the X- and Y-REGs are not disturbed.

Uses: RTLOG

Example: Prior to calling RTAR, FP1 contains the normalized value -7.

FP1    

|      |      |   |   |
|------|------|---|---|
| \$83 | \$A0 | 0 | 0 |
|------|------|---|---|

    (-7)  
          X1        M1

114

After calling RTAR, FP1 contains the unnormalized value -7 (note that precision is lost off the low-order end of M1).

FP1    

|      |      |   |   |
|------|------|---|---|
| \$84 | \$D0 | 0 | 0 |
|------|------|---|---|

    (-7)  
          X1        M1

Note: M1 sign bit is unchanged.

RTLOG subroutine (address \$F480)

Purpose: To shift the 6-byte field MANT1 and E one bit to the right (toward the least significant bit). The 6502 carry bit is shifted into the high-order M1 bit. This is useful in correcting binary sum overflows.

Entry: A normalized or unnormalized floating point value is in FP1. The carry must be cleared or set by the user since it is shifted into the sign bit of M1.

Exit: Same as RTAR except that the sign of M1 is not preserved (it is set to the value of the carry bit on entry)

Caution: Same as RTAR.

Example: Prior to calling RTLOG, FP1 contains the normalized value -12 and the carry is clear.

FP1:    

|      |      |   |   |
|------|------|---|---|
| \$83 | \$A0 | 0 | 0 |
|------|------|---|---|

    (-12)  
          X1        M1

After calling RTLOG, M1 is shifted one bit to the right and the sign bit is clear. X1 is incremented by 1.

FP1:    

|      |      |   |   |
|------|------|---|---|
| \$84 | \$50 | 0 | 0 |
|------|------|---|---|

    (+20)  
          X1        M1

Note: The bit shifted off the end of MANT1 is rotated into the high-order bit of the 3-byte extension E. The 3-byte E field is also shifted one bit to the right.



WOZPAK II

Exit: The 3 bytes of M1 are cleared (or all set to the contents of the X-REG on Entry) and the Y-REG is loaded with \$17. The sign bit of the A-REG is complemented and a copy of the A-REG is stored in X1. FP2, SIGN, SCR, and the X-REG are not disturbed.

Uses: NORM.

Caution: Exponent overflow results in an exit to location \$3F5. Exponent underflow results in an early return from the calling subroutine (FDIV or FMUL) with a floating point zero in FP1. Because MD2 pops a return address off the stack, it may only be called by another subroutine.

RTLOG1 subroutine (address \$F484)

Purpose: To shift MANT1 and E right one bit without adjusting X1. This is used by the multiply loop. The carry is shifted into the sign bit of MANT1.

Entry: M1 and E contain a 6-byte unsigned field. E is the 3-byte low-order extension of MANT1.

Exit: Same as RTLOG except that X1 is not altered and an overflow exit cannot occur.

MD2 subroutine (address \$F4E2)

Purpose: To clear the 3-byte MANT1 field for FMUL and FDIV, check for initial result exponent overflow (and underflow), and initialize the X-REG to \$17 for loop counting.

Entry: the X-REG is cleared by the user since it is placed in the 3 bytes of MANT1. The A-REG contains the result of an exponent addition (FMUL) or subtraction (FDIV). The carry and sign status bits should be set according to this addition or subtraction for overflow and underflow determination.

---

**BLANK PAGE**

---

AUTO-REPEAT FOR  
APPLE ][ MONITOR COMMANDS

## WOZPAK II

It is occasionally desirable to automatically repeat a MONITOR command or command sequence on the APPLE ][ computer. For example, intermittently bad RAM bits in the \$800-\$FFF address range (\$ stands for hex) may be detected by verifying those locations with themselves using the MONITOR verify command:

```
*800<800.FFFV[Cr] ([Cr] is RETURN)
```

Because this type of problem may be intermittent, multiple verifications may be necessary before the problem is detected. Typing the verify command over and over is a tedious chore which may not even catch the bug, since the RAMs are not fully exercised while the user is typing.

The APPLE ][ MONITOR command input buffer begins at location \$200 and is scanned from beginning to end after the user finishes typing a line by typing a carriage return. An index to the next executable character of the buffer resides in location \$34 while any function is being executed. By adding the command '34:0' to the end of a MONITOR command sequence, the user causes scanning to

resume at the beginning. Because the '34:0' command leaves the MONITOR in a 'store' mode, an 'N' command should begin the line. The following is an example of a command sequence which verifies the locations \$800-\$FFF with themselves, automatically repeating:

```
*N800<800.FFFV 34:0 [Cr]
```

(Note that the space between the final command and [return] is necessary for this feature to work properly)

Multiple command sequences accepted by the APPLE ][ MONITOR may also be automatically repeated. For example, the following command sequence clears all bits in the address range \$400-\$5FF, verifies all of these locations with themselves, sets them all to ones, verifies them again, and repeats:

```
*N400:0 N401<400.5FEM 400<400.5FFV 400:FF
N401<400.5FEM 400<400.5FFV 34:0 [Cr]
```

Because this example uses screen memory locations, it is observable on the display. The repeating command may be halted by hitting RESET. Since the cursor is only generated for keyboard entry, it will disappear while the example repeats.

DIRECT CALLS TO APPLE ][  
INTEGER BASIC FUNCTIONS

## WOZPAK II

The following APPLE ][ Integer BASIC entry points are directly callable:

```
LIST -- JSR $E04B CALL -8117
RUN -- JSR $EFEC CALL -4116
RUN* -- JSR $E836 CALL -6090
SAVE -- JSR $F140 CALL -3776
LOAD -- JSR $F0DF CALL -3873
```

The random number function may be accessed directly as follows:

1. Place the modulus (RND argument) in locations \$CE (low) and \$CF (high).
2. LDX #\$20 (may use other values--X-REG used as BASIC stack pointer.)
3. JSR \$EF51
4. X-REG will now be one less and the random value returned is in locations \$6F (low) and \$BF (high).

NOTE: If X were less (or greater) than \$20, then the correspondingly lower (or higher) locations.

\* This RUN entry does NOT delete the BASIC Variables.

## PRINT DECIMAL

(2-byte integer 0 to 65535)

1. Specify leading character option in location \$FA.
  - 0 = no leading characters (normal BASIC mode)
  - "0" = leading zeros, 5-char. field
  - "blank" = leading blanks
  - "(char)" = leading (char)
2. LDX low byte
3. LDA high byte
4. JSR \$E51B

## CAUTIONS:

- a. Location \$F8 should be positive when PRINT DEC is called, or the printed characters will also be added to the BASIC input buffer (\$200-\$2FF) using the Y-REG as an index (as in AUTO LINE NUMBER mode).
- b. Locations \$C9, \$F2, \$F3, \$F9 are used as temporaries.
- c. Exit conditions:
  - A-REG contains last char printed.
  - X-REG contains \$FF.
  - Y-REG undisturbed (unless caution (a) ignored)
  - Negative flag is set.

APPLE ][ TREK

By Wendell Sander

---

**BLANK PAGE**

---



## WOZPAK II

APPLE ][ TREK is a sophisticated space war game in which the player, as Captain of the Starship ENTERPRISE is sent on a search and destroy mission against the KLINGON Empire Fleet. The APPLE ][ computer creates the game environment, operates the KLINGON Cruisers in combat and transforms the APPLE ][ keyboard and display into a spaceship command console.

## THE GALAXY

APPLE ][ TREK is played in a galaxy which is represented by a grid of 64 quadrants charted as an 8 by 8 array. Each quadrant contains 64 sectors, again in an 8 by 8 array. The sector is the elemental location in the universe and may be occupied by only one of a star, a KLINGON, or the ENTERPRISE. The galaxy is a closed space in which the opposite edges are actually adjacent to each other. Moving to Galactic East of the Eastmost quadrant, the ENTERPRISE will enter the Westmost quadrant. At the start of a mission, the APPLE places all stars, Klingons, bases, and your ship at random.

## THE STARSHIP ENTERPRISE

The ENTERPRISE is a powerful craft with somewhat more firepower and energy capacity than the battle cruisers of the Klingon fleet. However, the ENTERPRISE is usually heavily outnumbered and is easily destroyed unless good maneuvering and firing strategies are used.

The ENTERPRISE has two forms of weaponry; Photon Torpedoes (PH TORPS) and Phasers. The ENTERPRISE normally carries 10 PH TORPS which may be restocked through energy conversion or by visiting a base.

The ENTERPRISE carries energy in three forms; available energy, shield energy, and PH TORPS. The available energy is used for Phasers, propulsion, and to operate the ship's sensory and display subsystems.

Shield energy is carried in ship's shields to absorb hits from attacking Klingons. If the ENTERPRISE receives an enemy hit that reduces its shield energy too low, then the ENTERPRISE may sustain damage to its subsystems (see damage report).

PH TORPS are considered as energy in determining the maximum allowable energy aboard the ENTERPRISE. Also, PH TORPS can be converted directly into energy or formed from energy. The transformation consumes energy.

The ENTERPRISE receives new energy from contact with restocking bases and from on-board Dilithium Crystals. In addition, energy reallocation between available energy, shield energy, and PH TORPS may be directed by the captain. Rendezvous with a restocking base brings the ENTERPRISE back to maximum and repairs all sub-systems. The restocking base is depleted by this action and can not be used again. The on-board Dilithium Crystals generate energy at a rate of 50 units per 0.1 stardate.

## KLINGON BATTLE CRUISERS

The Klingon Battle Cruiser is somewhat less powerful than the ENTERPRISE but usually runs in squadrons of several ships at a time and therefore may have greater joint firepower than the ENTERPRISE in a battle. Each Klingon has 800 units of energy and 3 Photon Torpedoes (PH TORPS) at the beginning of a battle. The Klingon energy may be used for Phasor fire, movement, or to absorb hits from the ENTERPRISE. When the Klingon energy is reduced to zero, the Klingon is destroyed.

The Klingons have both Phasors and PH TORPS which operate the same as the ENTERPRISE (see below) except that the Klingons cannot lock PH TORPS, or convert between PH TORPS and ENERGY.

Klingon's can move one sector distance per turn during a battle. They may retreat into adjacent quadrants and become restocked at that time.

## CONSOLE DISPLAY

The console display for the APPLE TREK mission is presented in four display segments. The top third of the screen displays the Galactic Record, the lower left of the screen contains a detailed display of the current quadrant, the lower right of the screen is a status display, and the center of the screen is reserved for command I/O.

WOZPAK II

THE GALACTIC RECORD

The Galactic Record is displayed at the beginning of the game or whenever any navigation command is entered. The Galactic Record is an 8 by 8 array of numbers representing a summary of the number of Klingons, Bases, and stars in each quadrant of the galaxy that have been observed during the game. Figure 1 shows a typical Galactic Record. The number shown in a square of the Galactic Record should be interpreted digit by digit. The ones digit is the number of stars in that quadrant, the tens digit is the number of bases, and the hundreds digit is the number of Klingons. Some examples would be:

```
305 3 Klingons, no bases, 5 stars
13 1 base, 3 stars
4 4 stars
```

The Galactic Record contains data for each quadrant that the ENTERPRISE has occupied. In addition, the data for the 8 surrounding quadrants are presented if the Long Range Sensor is operational.

The Galactic Record display area is also used for several other utility displays such as the Probe and Damage Report.

THE QUADRANT DISPLAY

The Quadrant Display presents a detailed picture of the quadrant currently occupied by the ENTERPRISE. Figure 2 illustrates a typical Quadrant Display. The display is in inverse video. The location of the ENTERPRISE is indicated an "E", Klingons are shown as "K", stars as "\*", and bases as "B".

THE STATUS DISPLAY

The Status Display presents a brief summary of the current status of the ENTERPRISE. The display includes current sector location, years remaining in the mission, current stardate, condition code (green = no problems, yellow = low on energy, and red = Klingons present), shields (percentage of total energy that will go to shields), shield energy, available energy, number of PH TORPS,

number of Klingons, and number of bases. The last line of the Status Display indicates the course coordinates set by the on-board computer to permit fire and move sequences.

The Status Display area is also used to display the list of possible commands whenever a non-legal command is entered.

FIGURE 1  
SAMPLE GALACTIC RECORD

```
: : : : : : : :
: : :3 :104:4 : : : :
: : :12 :8 :21 : : : :
: : :3 :312:104:4 : : :
: : :11 :2 :3 :16 : : :
: : :1 :202:4 :5 : : :
: : : : : : : :
: : : : : : : :
```

Note: The quadrant that the ENTERPRISE is in will appear in inverse video.

FIGURE 2  
SAMPLE QUADRANT DISPLAY

```

 1
 2
 3
 * *
 K *
 K E
 *
 *
 B
 1 2 3 4 5 6 7 8
```

Note: Display is in inverse video.

## WOZPAK II

## COMMANDS

The ENTERPRISE is controlled by entering the following commands from the keyboard:

- 1 - Navigation
- 2 - Set Shield Energy
- 3 - Damage Report
- 4 - Phasers
- 5 - Ph Torps
- 6 - Load Ph Torps
- 7 - Computer
- 8 - Probe
- 9 - Self Destruct

The request for a command appears in the left center display area. The commands operate as described below.

## 1-NAVIGATION

The Navigation command is used to move the ENTERPRISE to different sectors within a quadrant, or to different quadrants within the galaxy. After pressing "1", the Galactic Record is displayed. If Warp Drive is not damaged, the question WARP OR ION (W OR I)? will appear. Warp Drive is used for movement between quadrants. A warp factor will be requested. Movement will occur to a quadrant that is that many units distant. Similarly, with Ion Drive, duration will be requested and will specify the number of sectors to move within the quadrant. With both warp and ion drive, a course is also requested. This is simply the angle or direction that you wish the ENTERPRISE to travel in. Galactic North (up on your screen) is 0 degrees, East (to the right) is 90 degrees, etc. Any number between 0 and 359 may be entered.

## 2 - SET SHIELD ENERGY

The ENTERPRISE maintains a certain portion of its available energy in the shields to absorb blows from Klingon fire. If the shield energy goes below 10 units, damage may be sustained to the operating systems of the ENTERPRISE. The Set Shield parameter sets the percentage of total energy that goes to shields. This value is initially 50%.

## 3 - DAMAGE REPORT

Command 3 will cause the top portion of the screen to display the current status of all of the ENTERPRISE systems such as Phasers, PH TORPS, Computer, etc. If the subsystem is operational, the display will indicate OK. If damage has occurred, the display will indicate the estimated time required to repair the subsystem.

## 4 - PHASERS

Firing the Phasers will cause a blast of energy to be shot at all targets within the quadrant. The energy is equally divided among the targets and is diminished by the distance between the ENTERPRISE and the target. The available energy of the ENTERPRISE is decreased by the amount of Phaser fire. Hits on the Klingons will decrease their energy by a like amount. Phasers can be locked on to one or more targets using the computer (see below). In that case, the energy is spread evenly between the selected targets. Phaser fire is not

## WOZPAK II

blocked by stars or Klingons.

### 5 - PH TORPS

Photon Torpedoes may be fired under manual or automatic control. Under manual control, only a single TORP may be fired. A trajectory must be input (angle is similar to course discussed above). Under automatic control, you can use the computer to lock-on to any number of targets (see computer section). The computer then directs the PH TORPS to their targets. The only disadvantage to automatic fire is that the Klingons then get to shoot first.

Photon Torpedoes will cause 500 energy units of damage if a hit is made. The torpedo will hit the first object that it encounters on the given trajectory, be that a Klingon, a star, or a base.

### 6 - LOAD PH TORPS

Photon Torpedoes can be converted to or from energy using this command. You will be asked how many TORPS to load. A positive number response will convert energy to PH TORPS at a rate of 500 units of energy each. A negative input will convert PH TORPS to energy at the same rate.

### 7 - COMPUTER

The captain of the ENTERPRISE has an APPLE-81 computer at his disposal, giving him a major advantage over his Klingon opponents. The computer has 7 command options which will be displayed whenever a non-legal input is made (such as 0 or 8). These options allow you to compute a course angle to any given set of sector or quadrant coordinates, compute a trajectory to a Klingon, lock Phasers or PH TORPS to any number of targets, lock in a course, or display ship's status. Control can be returned to command mode by entering the return command (option 7 while in Computer mode).

### 8 - PROBE

The Probe command is used to assess the strength of Klingons that are in the same quadrant as the ENTERPRISE. The command will display the coordinates, energy levels, number of PH TORPS, and "lock" status of all Klingons in the current quadrant. The "lock" status specifies whether or not the Klingon's Phasers or PH TORPS are locked-on to the ENTERPRISE or not. The Probe infor-

WOZPAK II

mation is displayed in the Galactic Record display area.

9 - SELF DESTRUCT

The Self Destruct command should be reserved until there is no hope of the survival of the ENTERPRISE. It will cause the ENTERPRISE to explode, hopefully taking any nearby Klingons with it.

END OF A MISSION

A mission will end when one of three events occurs. If all Klingons are destroyed, the ENTERPRISE has successfully completed its mission. If the time allotted for the mission runs out, the mission ends with the Klingons still threatening the empire. Finally, if the ENTERPRISE is destroyed, the mission is considered a failure. Your performance will be rated under each of these circumstances and an appropriate dispatch from Star Fleet Command will be sent. Your rating will depend on the overall success of your mission, on the number of bases used up, the amount of time required, and the amount of energy and PH TORPS used.

---

**BLANK PAGE**

---

APPLE ][  
HI-RES COLOR MODIFICATION

---

**BLANK PAGE**

---



WOZPAK II

The High Resolution Color Graphics capability of the APPLE ][ computer is one of its features that set it apart from most other personal computers. Early versions of the APPLE ][ had the ability to display four colors: Black, White, Green, and Violet. A production change was made, and the ability to display Blue and Orange was added. The modification presented here allows those early APPLES to be updated to 6 color capability.

PLEASE NOTE: This modification will VOID the warranty (if still in effect) on the APPLE ][.

- (1) Remove the ten screws securing the plastic top piece to the metal bottom plate. Six of these are flat-head screws around the perimeter of the bottom plate, and four are round head screws located at the front lip of the computer. All are Phillips screws. Do not remove the screws securing the power supply or the nylon PC board standoffs.
- (2) Lift the plastic top piece from the bottom plate, taking care not to damage the ribbon cable connecting the keyboard to the main PC board. This cable must be disconnected from one end or the other.
- (3) Disconnect the power supply from the PC board.
- (4) Remove the #8 nut and lockwasher that secures the center of the PC board. This may not be found on the earlier APPLE ][ computers.
- (5) Carefully disengage each of the 6 nylon insulating standoffs from the PC board. (7 on earlier versions)
- (6) Carefully lift the PC board from the bottom plate.

Now that your APPLE ][ is in pieces, there are two wiring methods for this modification. It is HIGHLY recommended that you choose the second (below the board method), since the final result will have a much more professional appearance. It will not be visible when the APPLE ][ is reassembled. (Now that's professional!) The wiring itself is much easier to keep track of if you use small pieces of tape (with the Integrated Circuit numbers written on them) on the bottom of the board.

In the following instructions, the connections required are designated by: ( <chip number> - <pin number> ) See the APPLE Red Manual for IC number designations.

ABOVE THE BOARD WIRING METHOD

- (1) Lift the following IC pins from their sockets. This requires bending the pins (careful!).

|        |        |         |
|--------|--------|---------|
| (A8-1) | (A8-6) | (A8-13) |
| (A9-1) | (A9-2) | (A9-9)  |

- (2) Mount a 74LS74 (dual C-D flip-flop) and a 74LS02 (quad NOR gate) in the APPLE ][ breadboard area (A11 to A14 region).
- (3) Wire the circuit as shown in Fig. 1. (\* indicates a connection to a pin which is out of its socket)



APPLE ][  
COLOR KILLER MODIFICATION

---

**BLANK PAGE**

---

WOZPAK II

The APPLE ]['s color graphics are great, but there are times when color is not needed. Printing a page of text on the screen is a situation when it is not desirable to have color on the screen. The early APPLES did not disable the burst signal (the signal that tells the color television to turn on the color circuits), so the color was on all the time.

This simple modification will provide this 'color on/color off' switching in the APPLE to make the screen display color or black & white as necessary.

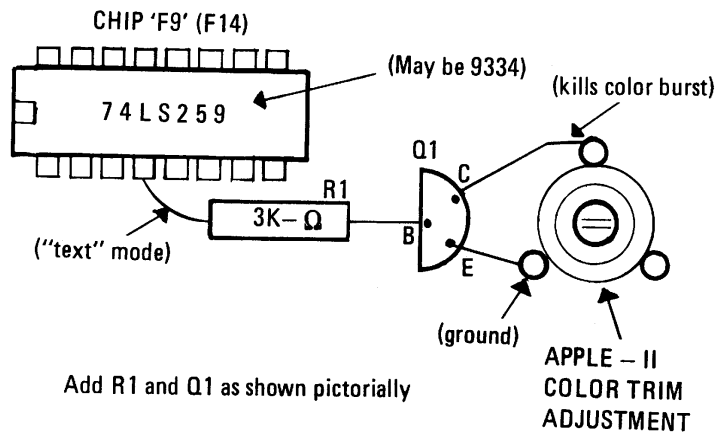
PARTS NEEDED:

- 3K to 5K 1/4 watt resistor
- NPN transistor (2N222 or equivalent)

Wire the circuit as shown in figure 1. Be sure to unplug your APPLE ][ whenever making ANY connections or modifications.

PLEASE NOTE: This, or any other, modification will VOID the warranty on your APPLE ][, if it is still in effect.

APPLE - II COLOR KILLER MOD



The  
End