2. Index into block

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

LDA 780,X

The X-REG is presumed to contain $EO-$FF. Because $780 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 (YC) its entry point must remain fixed at $3F8. The rest of RELOC may be relocated anywhere in memory (which is trivial since RELOC contains no absolute 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.
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-II.

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 SEGMENT
BEG parameter ($800) and need not be specified. The initial
segment relocation command may be abbreviated as follows:

* 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-II 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.

<table>
<thead>
<tr>
<th>SOURCE SEGMENTS</th>
<th>DEST SEGMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$800$</td>
<td>$800$</td>
</tr>
<tr>
<td>$800$-$8A6$</td>
<td>$800$-$8A6$</td>
</tr>
<tr>
<td>$8A6$</td>
<td>$8A6$</td>
</tr>
<tr>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>$900$</td>
<td>$8A7$</td>
</tr>
<tr>
<td>Program B</td>
<td>Program B</td>
</tr>
<tr>
<td>$9F1$</td>
<td>$900$-$9F1$</td>
</tr>
</tbody>
</table>

SOURCE BLOCK $900$-$9F1$  DEST BLOCK $8A7$-$998$

SOURCE SEGMENTS $800$-$8A6$ (A)  DEST SEGMENTS $800$-$8A6$ (A)

$900$-$9F1$ (B)  $8A7$-$998$ (B)

(a) Load RELOC
(b) Define blocks (program B only)

* $8A7 < 900 \cdot 9F1 \; Y^C \; *$
(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.


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

SOURCE SEGMENTS

\[
\begin{array}{|c|c|}
\hline
\text{SOURCE BLOCK $8C4$-$97F$} & \text{DEST BLOCK $86C$-$97B$} \\
\hline
\text{SOURCE SEGMENTS} & \text{DEST SEGMENTS} \\
\hline
$800$-$88F$ (code) & $800$-$88F$ (code) \\
$890$-$8AF$ (data) & $890$-$8AF$ (data) \\
$8B0$-$8BF$ (code) & $8B0$-$8BF$ (code) \\
$8C4$-$90F$ (code) & $8C0$-$90B$ (code) \\
$910$-$93F$ (data) & $90C$-$93B$ (data) \\
$940$-$97F$ (code) & $93C$-$97B$ (code) \\
\hline
\end{array}
\]

Remove 4 bytes here ($8C0$-$8C3$)
(a) Load RELOC
(b) Define blocks

* 8C0 < 8C4 . 97F YC *

(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-II monitor ($F800-$FFFF) to run in RAM ($800-$FFF)

SOURCE BLOCK $F700-$FFFF  DEST BLOCK $700-$FFF
(see example (2))

SOURCE SEGMENTS $F800-$F961 (code) DEST SEGMENTS $800-$961 (code)
$F962-$FA42 (data)    $962-$A42 (data)
$FA43-$FB18 (code)    $A43-$B18 (code)
$FB19-$FB1D (data)    $B19-$B1D (data)
$FB1E-$FFCB (code)    $B1E-$FCB (code)
$FFCC-$FFFF (data)    $FCC-$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 $Y^C$ *

(c) Segments

   * 800 < F800 . F961 $Y^C$ (first segment, code)
   * . FA42 M            (data)
   * . FB18 $Y^C$        (code)
   * . FB1D M            (data)
   * . FFCB $Y^C$        (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-II which are used by RELOC. If adapted to other systems, the convenient and flexible parameter passing capability of the APPLE-II monitor may be sacrificed.

1. The APPLE-II monitor command

   \[ * A_d < A_1 . A_2 Y^C \quad (A_1, A_2, \text{and} \ A_4 \text{are addresses}) \]

   vectors to location $3F8$ with the value $A_1$ in locations $3C$ (low and $3D$ (high)), $A_2$ in locations $3E$ (low) and $3F$ (high), and $A_4$ in locations $42$ (low) and $43$ (high). Location $34$ (YSAV) holds an index to the next character of the command buffer (after the $Y^C$). The command buffer (IN) begins at $200$.

2. If $Y^C$ is followed by an \(*\) then the block parameters are simply preserved as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Preserved at</th>
<th>SWEET16 Reg Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEST BLOCK BEG</td>
<td>$8, 9$</td>
<td>TOBEG</td>
</tr>
<tr>
<td>SOURCE BLOCK BEG</td>
<td>$2, 3$</td>
<td>FRMBEG</td>
</tr>
<tr>
<td>SOURCE BLOCK END</td>
<td>$4, 5$</td>
<td>FRMEND</td>
</tr>
</tbody>
</table>

3. If $Y^C$ is not followed by and \(*\) 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-II monitor subroutine which determines the length of a 6502 instruction in the variable LENGTH (location $2F) given the opcode in the A-REG.

<table>
<thead>
<tr>
<th>Instruction type</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td>0</td>
</tr>
<tr>
<td>1 byte</td>
<td>0</td>
</tr>
<tr>
<td>2 byte</td>
<td>1</td>
</tr>
<tr>
<td>3 byte</td>
<td>2</td>
</tr>
</tbody>
</table>

5. The code from XLATE to SW16RT ($3D9-$3E6) uses the APPLE-II 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-II 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.
6502 RELOCATION SUBROUTINE

4:36 P.M., 11/10/1977

1  TITLE '6502 RELOCATION SUBROUTINE'
2  *******************************************
3   *
4   * 6502 RELOCATION *
5   * SUBROUTINE *
6   *
7   * 1. DEFINE BLOCKS *
8   *   *A4<A1.A2 ~Y *
9   *   (^Y IS CTRL-Y) *
10  *
11  * 2. FIRST SEG *
12  *   *A4<A1.A2 ~Y *
13  *   (IF CODE) *
14  *
15  *   *A4>A1.A2 M *
16  *   (IF MOVE) *
17  *
18  * 3. SUBSEQUENT SEGS *
19  *   *.A2 ~Y OR *.A2 M *
20  *
21  * WOZ 11-10-77 *
22  * APPLE COMPUTER INC. *
23  *
24  *******************************************
25  PAGE
<table>
<thead>
<tr>
<th>Line</th>
<th>Symbol</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>SUBTTL</td>
<td>EPZ</td>
<td>$2</td>
<td>SWEET16 REG 1.</td>
</tr>
<tr>
<td>27</td>
<td>R1L</td>
<td>EPZ</td>
<td>$8</td>
<td>3-BYTE INST FIELD.</td>
</tr>
<tr>
<td>28</td>
<td>INST</td>
<td>EPZ</td>
<td>$2F</td>
<td>LENGTH CODE.</td>
</tr>
<tr>
<td>30</td>
<td>YSAV</td>
<td>EPZ</td>
<td>$34</td>
<td>CMND BUF POINTER.</td>
</tr>
<tr>
<td>31</td>
<td>A1L</td>
<td>EPZ</td>
<td>$3C</td>
<td>APPLE-II MON PARAM AREA.</td>
</tr>
<tr>
<td>32</td>
<td>A4L</td>
<td>EPZ</td>
<td>$42</td>
<td>APPLE-II MON PARAM REG 4</td>
</tr>
<tr>
<td>33</td>
<td>IN</td>
<td>EQU</td>
<td>$200</td>
<td>MON CMND BUF.</td>
</tr>
<tr>
<td>34</td>
<td>SW16</td>
<td>EQU</td>
<td>$F689</td>
<td>SWEET16 ENTRY.</td>
</tr>
<tr>
<td>35</td>
<td>INSDS2</td>
<td>EQU</td>
<td>$F88E</td>
<td>DISASSEMBLER ENTRY.</td>
</tr>
<tr>
<td>36</td>
<td>NXTA4</td>
<td>EQU</td>
<td>$FCB4</td>
<td>POINTER INCR SUBR.</td>
</tr>
<tr>
<td>37</td>
<td>FRMBEG</td>
<td>EPZ</td>
<td>$1</td>
<td>SOURCE BLOCK BEGIN.</td>
</tr>
<tr>
<td>38</td>
<td>FRMEND</td>
<td>EPZ</td>
<td>$2</td>
<td>SOURCE BLOCK END.</td>
</tr>
<tr>
<td>39</td>
<td>TOBEG</td>
<td>EPZ</td>
<td>$4</td>
<td>DEST BLOCK BEGIN.</td>
</tr>
<tr>
<td>40</td>
<td>ADR</td>
<td>EPZ</td>
<td>$6</td>
<td>ADR PART OF INST.</td>
</tr>
<tr>
<td>41</td>
<td>PAGE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6502 RELOCATION SUBROUTINE

4:36 P.M., 11/10/1977

SUBTIL 6502 RELOCATION SUBROUTINE

ORG $3A6

CMND BUF POINTER.

CMND $3A6

NEXT CMND CHAR.

"*"?

NO, RELOC CODE SEG.

ADVANCE POINTER.

MOVE BLOCK PARAMS

FROM APPLE-II MON

AREA TO SW16 AREA.

R1=SOURCE BEG, R2=

SOURCE END, R4=DEST BEG

COPY 3 BYTES TO

SW16 AREA.

CALCULATE LENGTH OF

INST FROM OPCODE.

0=1 BYTE, 1=2 BYTE,

2=3 BYTE.

WEED OUT NON-ZERO-PAGE

2 BYTE INSTRS (IMM).

IF ZERO PAGE ADR

THEN CLEAR HIGH BYTE.

IF ADR OF ZERO PAGE

OR ABS IS IN SOURCE

(FRM) BLOCK THEN

SUBSTITUTE ADR-

SOURCE BEG+DEST BEG.

COPY LENGTH BYTES

OF INST FROM

SW16 AREA TO

DEST SEGMENT. UPDATE

SOURCE, DEST SEGMENT

POINTERS. LOOP IF NOT

BEYOND SOURCE SEG END.

ENTRY FROM MONITOR.

********SUCCESSFUL ASSEMBLY: NO ERRORS
CROSS-REFERENCE: 6502 RELOCATION SUBROUTINE
A1L 003C 0050 0056
A4L 0042 0082
ADR 0006 0072 0074 0078
FRMBEG 0001 0075
FRMEND 0002 0071
GETINS 03BD 0059
IN 0200 0045
INIT 03B3 0053
INSDS2 F88E 0060
INST 000B 0057 0064 0069 0081
LENGTH 002F 0061 0085
NXTA4 FCB4 0084
RIL 0002 0051
RELOC 03A6 0090
RELOCC 03BB 0047 0087
STINS2 03E9 0086
STINST 03E7 0066 0068
SW16 F689 0070
SW16RT 03E6 0073 0076
TOBEG 0004 0077
USRLOC 03F8
XIMATE 03D9 0063
YSAV 0034 0044 0048
FILE:
This page is not part of the original Wonderbook
The Woz Wonderbook

DOCUMENT

Apple-II

Renumbering and Appending
BASIC Programs

15 November 1977

This page is not part of the original Wonderbook
This page is not part of the original Wonderbook
RENUMBERING AND APPENDING

BASIC PROGRAMS

on the

APPLE-II COMPUTER

S. Wozniak (WOZ)

November 15, 1977
RENUMERATING AND APPENDING APPLE-II BASIC PROGRAMS

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" 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.

---


* In fact, while several texts detail how the boundary conditions of a KENO game lead to predictable outcomes, finished programs seldom exhibit this property.
The APPLE-II 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 one of three machine
level programs. The renumber operation (RENUM) requires user speci-
fication 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.

```
RANGE BEGIN  200
RANGE END    340
NEW BEGIN    100
NEW INCREMENT 10
```

A second RENUM entry renumbers the entire program, relieving
the user of the need to specify the range 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.
USING RENUM/APPEND

1. Load RENUM/APPEND (* 300.3D4 R)

Note that the high-order bytes of page 3 are not loaded, preventing inadvertant alteration of the interrupt and user function (YC) vectors. The '"" is generated by the MONITOR, not the user.

2. Load a BASIC program.

3. To renumber entire program:

    POKE 2, START L           User must supply low and high bytes
    POKE 3, START H           of new STARTing line number.
    POKE 4, INCR L           User must supply low and high bytes
    POKE 5, INCR H           of new line number INCrement.
    CALL 768                         (does not alter locations 2-5)

Note: START L is equivalent to START MOD 256

        START H is equivalent to START / 256

4. To renumber a range of the program

    POKE 2, START L
    POKE 3, START H
    POKE 4, INCR L
    POKE 5, INCR H
    POKE 6, RANGE START L    User must supply low and high bytes
    POKE 7, RANGE START H    of renumber range starting line number.
    POKE 8, RANGE END L     User must supply low and high bytes
    POKE 9, RANGE END H     of renumber range ending line number.
    CALL 776                         (does not alter locations 2-9)
5. To append program #2 (larger line numbers) to program #1 (smaller line numbers):

(a) Load program #2
(b) CALL 956

   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 973 to recover the original program.
ERRORS

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 973 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

1. When appending a program, always load the one with 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 GO TO 100 + ALPHA will be modified only to reflect the new line number assigned to the old line 100.
APPLE-II 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-II 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 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-II 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 preceeded by $B0-$B9. All constants are in this form including line number references such as 500 in the statement GO TO 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

LOMEM (start of variables) ($4A,4B)

BASIC VARIABLES

PV (Variable Pointer, end of variables) ($CC,CD)

PP (Program Pointer, start of program) ($CA,CB)

BASIC PROGRAM

first line

last line

HIMEM (end of program) ($4C,4D)

Figure 2 - LINE REPRESENTATION

<table>
<thead>
<tr>
<th>length byte</th>
<th>low</th>
<th>high</th>
<th>items</th>
<th>'end-of-line' token</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$01</td>
</tr>
</tbody>
</table>
Figure 3 - ITEMS

Constant:  
$B0$-$B9$  
\text{low}  \quad \text{high}  
\text{value}  \quad \text{positive}  
\text{token}

Name (ABC):  
$C1$  
$C2$  
$C3$  
\text{negative}  
\text{ASCII}  \quad \text{positive}  
\text{token}

String Constant ("123"):  
$B28$  
$B21$  
$B22$  
$B23$  
$B29$  
\text{quote}  
\text{token}  \quad \text{negative}  
\text{ASCII}  \quad \text{quote}  
\text{token}

REM:  
$5D$  
\text{REM}  
\text{token}  \quad \text{negative}  
\text{ASCII}  \quad \text{'end-of-line'}  
\text{token}

Tokens: $00$-$7F$  
GO TO  \quad -$5F$  
GOSUB  \quad -$5C$  
THEN \text{ ln}  \quad -$24$  
LIST  \quad -$74$  
LIST , \quad -$75$  
STR CON \quad -$28$  
REM  \quad -$5D$  
EOL  \quad -$01$  
\text{(tokens used by RENUMBER)}
RENUMBER - THEORY OF OPERATION

Because of the rigid internal representation of APPLE-II 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 BASIC's. Fortunately all constants in APPLE-II BASIC (including line number references) are preconverted to binary.

The normal renumber subroutine entry point is RENUM ($308). The RENX entry ($300) conveniently sets the renumber range for the user such that the entire program will be renumbered. RENUM extensively uses SWEET16, the code-saving 16-bit interpretive machine built into the APPLE-II. Occasional 6502 code is interspersed throughout RENUM for even greater code efficiency.

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
GOSUB
THEN 1no
LIST
LIST
```

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

Exceptions such as empty line number table and null program 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 smaller line numbers will be read off tape. The current program resides between two 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 its original value and both programs will be joined. No line number overlap check is performed.

Original Program   After Load   HIMEM Restored

```
<table>
<thead>
<tr>
<th>PP</th>
<th>Prog #1</th>
<th>PP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP</td>
<td>Prog #2</td>
<td>PP</td>
</tr>
<tr>
<td></td>
<td>HIMEM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>PP</th>
<th>Prog #1</th>
<th>PP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prog #2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HIMEM</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>PP</th>
<th>Prog #1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prog #2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HIMEM</td>
<td></td>
</tr>
</tbody>
</table>
```
RENUMBER EXAMPLE

Original

Renumber lines 100-110
to start at 150
spaced by 10

>LIST
  1 GOTO 100
  2 GOSUB 103
  3 IF TRUE THEN 107
  4 LIST 109,110
100 REM
103 REM
107 REM
109 REM
110 REM
200 FOR I=1 TO 10
210 PRINT I
220 NEXT I
230 GOTO 1

>POKE 2, 150 MOD 256
>POKE 2, 150 / 256
>POKE 3, 150 MOD 256
>POKE 3, 150 / 256
>POKE 4, 10 MOD 256
>POKE 4, 10 / 256
>POKE 5, 100 MOD 256
>POKE 5, 100 / 256
>POKE 6, 100 MOD 256
>POKE 6, 100 / 256
>POKE 7, 100 MOD 256
>POKE 7, 100 / 256
>POKE 8, 110 MOD 256
>POKE 8, 110 / 256
>POKE 9, 110 MOD 256
>POKE 9, 110 / 256

>CALL 776
100->150
103->160
107->170
109->180
110->190

>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 EXAMPLE (cont)

Renumber lines 100-110 to start at
10 spaced by 5

>POKE 2, 10 MOD 256
>POKE 3, 10 / 256
>POKE 4, 5 MOD 256
>POKE 5, 5 / 256
>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
APPEND EXAMPLE

>LIST
  100 REM
  200 REM THE ORIGINAL PROGRAM
  300 REM

>CALL 956

>LIST
  10 REM
  20 REM THIS PROGRAM CAME FROM TAPE
  30 REM
  100 REM
  200 REM THE ORIGINAL PROGRAM
  300 REM
APPLE-II BASIC RENUMBER/APPEND SUBROUTINES


TITLE 'APPLE-II BASIC RENUMBER/APPEND SUBROUTINES'

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

*********************************************************************

*********************************************************************
<table>
<thead>
<tr>
<th>Line</th>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>SUBTL</td>
<td>6502 EQUATES</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ROL</td>
<td>EPZ $0</td>
<td>LOV-ORDER SW16 90 BYTE</td>
</tr>
<tr>
<td>24</td>
<td>ROL1</td>
<td>EPZ $1</td>
<td>HI-ORDER.</td>
</tr>
<tr>
<td>25</td>
<td>R1IL</td>
<td>EPZ $16</td>
<td>LOV-ORDER SW16 R11 BYZ</td>
</tr>
<tr>
<td>26</td>
<td>R1IH</td>
<td>EPZ $17</td>
<td>HI-ORDER.</td>
</tr>
<tr>
<td>27</td>
<td>HIMEM</td>
<td>EPZ $4C</td>
<td>BASIC HIMEM POINTER.</td>
</tr>
<tr>
<td>28</td>
<td>PPL</td>
<td>EPZ $CA</td>
<td>BASIC PROG POINTER.</td>
</tr>
<tr>
<td>29</td>
<td>PUL</td>
<td>EPZ $CC</td>
<td>BASIC VAR POINTER.</td>
</tr>
<tr>
<td>30</td>
<td>MEMFULL</td>
<td>EQU $E36B</td>
<td>BASIC MEM FULL ERROR.</td>
</tr>
<tr>
<td>31</td>
<td>PDEC</td>
<td>EQU $E51B</td>
<td>BASIC DECIMAL PRINT $M</td>
</tr>
<tr>
<td>32</td>
<td>RANGE1</td>
<td>EQU $E668</td>
<td>BASIC RANGE ERROR.</td>
</tr>
<tr>
<td>33</td>
<td>LOAD</td>
<td>EQU $F0DF</td>
<td>BASIC LOAD SUBR.</td>
</tr>
<tr>
<td>34</td>
<td>SV16</td>
<td>EQU $F639</td>
<td>SVEET16 ENTRY.</td>
</tr>
<tr>
<td>35</td>
<td>CROUT</td>
<td>EQU $FD8E</td>
<td>CAR RET SUBR.</td>
</tr>
<tr>
<td>36</td>
<td>COUT</td>
<td>EQU $FDED</td>
<td>CHAR OUT SUBR.</td>
</tr>
<tr>
<td>37</td>
<td></td>
<td>PAGE</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>SUBTTL</td>
<td>SWEET16 EQUATES</td>
<td>PAGE: 3</td>
</tr>
<tr>
<td>----</td>
<td>--------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>39</td>
<td>ACC</td>
<td>EPZ $0</td>
<td>SWEET16 ACCUMULATOR.</td>
</tr>
<tr>
<td>40</td>
<td>NEVLOW</td>
<td>EPZ $1</td>
<td>NEW INITIAL LNO.</td>
</tr>
<tr>
<td>41</td>
<td>NEVINCR</td>
<td>EPZ $2</td>
<td>NEW LNO INCR.</td>
</tr>
<tr>
<td>42</td>
<td>LNLOW</td>
<td>EPZ $3</td>
<td>LOW LNO OF RENUM RANGE</td>
</tr>
<tr>
<td>43</td>
<td>LH1</td>
<td>EPZ $4</td>
<td>HI LNO OF RENUM RANGE.</td>
</tr>
<tr>
<td>44</td>
<td>TBLSTRT</td>
<td>EPZ $5</td>
<td>LNO TABLE START.</td>
</tr>
<tr>
<td>45</td>
<td>TBLNDX1</td>
<td>EPZ $6</td>
<td>LNO TABLE LIMIT.</td>
</tr>
<tr>
<td>46</td>
<td>TBLIM</td>
<td>EPZ $7</td>
<td>SCRATCH REG.</td>
</tr>
<tr>
<td>47</td>
<td>SCR8</td>
<td>EPZ $8</td>
<td>HIMEM (END OF PRGM).</td>
</tr>
<tr>
<td>48</td>
<td>SCR9</td>
<td>EPZ $9</td>
<td>SCRATCH REG.</td>
</tr>
<tr>
<td>50</td>
<td>PRGNDX</td>
<td>EPZ $A</td>
<td>PASS 1 PROG INDEX.</td>
</tr>
<tr>
<td>51</td>
<td>PRGNDX1</td>
<td>EPZ $B</td>
<td>ALSO PROG INDEX.</td>
</tr>
<tr>
<td>52</td>
<td>NEVLN</td>
<td>EPZ $C</td>
<td>NEXT 'NEW LNO'.</td>
</tr>
<tr>
<td>53</td>
<td>NEWLN1</td>
<td>EPZ $D</td>
<td>PRIOR 'NEW LNO' ASSIGN</td>
</tr>
<tr>
<td>54</td>
<td>TBLND</td>
<td>EPZ $E</td>
<td>PASS 2 LNO TABLE END.</td>
</tr>
<tr>
<td>55</td>
<td>PRGNDX2</td>
<td>EPZ $F</td>
<td>PASS 2 PROG INDEX.</td>
</tr>
<tr>
<td>56</td>
<td>CHRO</td>
<td>EPZ $0</td>
<td>ASCII '0'.</td>
</tr>
<tr>
<td>57</td>
<td>CHRA</td>
<td>EPZ $1</td>
<td>ASCII 'A'.</td>
</tr>
<tr>
<td>58</td>
<td>MODE</td>
<td>EPZ $2</td>
<td>CONST/LNO MODE.</td>
</tr>
<tr>
<td>59</td>
<td>TBLNDX2</td>
<td>EPZ $3</td>
<td>LNO TBL IDX FOR UPDATE</td>
</tr>
<tr>
<td>60</td>
<td>OLDLN</td>
<td>EPZ $4</td>
<td>OLD LNO FOR UPDATE.</td>
</tr>
<tr>
<td>61</td>
<td>STRCON</td>
<td>EPZ $5</td>
<td>BASIC STR CON TOKEN.</td>
</tr>
<tr>
<td>62</td>
<td>REM</td>
<td>EPZ $6</td>
<td>BASIC REM TOKEN.</td>
</tr>
<tr>
<td>63</td>
<td>R13</td>
<td>EPZ $7</td>
<td>SWEET16 REG 13 (CPR REG)</td>
</tr>
<tr>
<td>64</td>
<td>THEN</td>
<td>EPZ $8</td>
<td>BASIC THEN TOKEN.</td>
</tr>
<tr>
<td>65</td>
<td>LIST</td>
<td>EPZ $9</td>
<td>BASIC LIST TOKEN.</td>
</tr>
<tr>
<td>66</td>
<td>SRCR</td>
<td>EPZ $A</td>
<td>SCRATCH REG FOR APPEND</td>
</tr>
</tbody>
</table>
APPLE-II BASIC RENUMBER SUBROUTINE - PASS 1


ORG $300

0300: 20 89 F6 70  RENX
       JSR  SW16
       SUB  ACC
       ST   LNLOW
       ST   LNH1
       DCR  LNH1
       RTN

030B: 20 89 F6 76  RENUM
       JSR  SW16
       SET  SCR8,HIMEM
       LDD  @SCR9
       ST   HMEM
       TO HIEMEM.
       SET  SCR9,PUL+2

0311: 19 CE 00 80  POPD @SCR9
       BASIC VAR PTR
       ST   TBLSTRT
       TBLSTRT AND TBLNDX1.
       ST   TBLNDX1
       LD   NEYLOW
       COPY NEWLOW (INITIAL)
       ST   NEWLN
       ST   NEYNI
       ST   NEYII

0317: 3B 85 85  ST   NEYII
       ST   NEYII

0318: 3C 86 86  ST   NEYII
       ST   NEYII

0319: 3C 87 87  POPD @SCR9
       BASIC PROG PTR
       ST   TBLIM
       TO TBLIM
       ST   PRGNDX
       AND PRGNDX.
       LD   PRGNDX

031C: 29 90 90  PASS1
       CPR  HMEM
       IF PRGNDX >= HMEM
       THEN DONE PASS 1.

031D: D8 91  CPR  HMEM
       INR  ACC
       IF < 2 BYTES AVAIL IN
       LNO TABLE THEN RETURN
       WITH 'MEM FULL' MSG.
       LD   @PRGNDX1

0322: 20 95 20  INR  ACC
       ST   PRGNDX1
       LD   TBLNDX1

0323: D7 96  D7  CPR  TBLIM
       ST   PRGNDX1
       LD   TBLNDX1

0324: 03 38 97  CPR  TBLIM
       ST   PRGNDX1
       LD   TBLNDX1

0326: 4A 98 4A  CPR  TBLIM
       ST   PRGNDX1
       LD   TBLNDX1

0327: A9 99  A9  ADD   PRGNDX
       ST   PRGNDX
       ADD   PRGNDX
       ST   TBLNDX1
       LDD  @PRGNDX1
       CPR  LNH1
       CPR  LNH1
       CPR  LNH1
       BNC  PIA
       LD   PIA
       LD   PIA
       LD   PIA
       LD   PIA

0330: 07 30 106  BNC  PIA
       STD  @TBLNDX1
       ADD  TO LNO TABLE.
       RTN

0332: 76 107 76  RTN
       PCI  TBLNDX1
       ADD  TO LNO TABLE.
       RTN

0333: 00 108 00  RTN
       LDA  ROH
       *** 6502 CODE ***
       LDA  ROL
       LDX  ROL
       JSR  PRDEG
       PRINT OLD LNO '->' NEW
       (RO,R11) IN DECIMAL.
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       JSR  COUT
       *** END 6502 CODE ***
APPLE II BASIC RENUMBER SUBROUTINE - PASS 1


0353: 3C 122 ST NEWLN1 COPY NEWLN TO NEWLN1
0354: A2 123 ADD NEWINCR AND INCR NEWLN BY
0355: 3B 124 ST NEWLN NEWINCR.
0356: 0D 125 NUL (WILL SKIP NEXT INST).
0357: D1 126 PIB CPR NEWLOW IF LOW LNO < NEWLOW
0358: 02 C2 127 BNC PASS1 THEN RANGE ERR.
0359: 00 128 RERR RTN PRINT 'RANGE ERR' MSG
035B: 4C 68 EE 129 JMP RANGERR AND RETURN.
035E: 00 130 MERR RTN PRINT 'MEM FULL' MSG
035F: 4C 68 EE 131 JMP MEMFULL AND RETURN.
0362: EC 132 PIC INR NEWLN1 IF HI LNO <= MOST RECH
0363: DC 133 CPR NEWLN1 NEWLN THEN RANGE ERR
0364: 02 F4 134 BNC RERR
0365: PAGE
APPLE-II BASIC RENUMBER SUBROUTINE - PASS 2


DISTRIBUTED UNDER THE CREATIVE COMMONS LICENSE ON PAGE 5

Page 0191 of 0213
APPLE-II BASIC RENUMBER ROUTINE -- PASS 2


03AF:  F0    190
03B0:  06 BA  191
03B2:  ID 74 00 192
03B5:  BD    193
03B6:  09 01  194
03B8:  BQ    195
03BA:  3C    196
03BB:  01 DI  197

DCR ACC
BZ P2A
SET LIST,$74
SUB LIST
BNMI CONTS2
CONTST
SUB ACC
ST MODE
BR ITEM
PAGE

EOL (TOKEN $01)?
SET MODE = 0 IF LIST
OR LIST COMMA ($73,4)
CLEAR MODE FOR LNO
UPDATE CHECK.
CHECK NEXT 'BASIC ITEM.
APPLE-II BASIC APPEND SUBROUTINE


199 SUBTL APPLE-II BASIC APPEND SUBROUTINE

033C: 20 59 F6 200 APPEND JSR SW16
033F: 1C 4E 00 201 SET SCRC, HIMEM+2
03C2: 00 202 POPD @SCRC SAVE HIMEM
03C3: 38 203 ST HMEM
03C4: 19 CA 00 204 SET SCR9, PPR
03C7: 69 205 LDD @SCR9 SET HIMEM TO PRESERVE
03CA: 7C 206 STD @SCRC PROGRAM
03C9: 00 207 RTN
03C2: 20 DF F0 208 JSR LOAD LOAD FROM TAPE
03CD: 20 59 F6 209 JSR SW16
03D0: 69 210 POPD @SCRC RESTORE HIMEM TO SHOW
03D1: 28 211 LD HMEM BOTH PROGRAMS
03D2: 7C 212 STD @SCRC (OLD AND NEW)
03D3: 00 213 DONE RTN RETURN
03D4: 60 214 RTS

*****SUCCESSFUL ASSEMBLY: NO ERRORS
CROSS-REFERENCE ATAYA-11 BASIC RENUMBER/APPEND SUBROUTINES

ACC 0000 0071 0095 0190 0195
APPEND 033C
CARTOK 039A 0170
CHRO 0309 0137 0159
CHRA 030A 0138 0171
CONT2 0399 0194
CONTST 0358 0183 0186 0159
COUT 013D 0113 0115
COUT 018E 0119
CSE 03D3 0141
GCON 0329 0172
HEL 0040 0077 0201
HEL1 0008 0079 0091 0140 0203 0211
ITEM 038D 0197
LIST 000D 0192 0193
LUSH 0004 0073 0074 0134
LOW 0003 0072 0102
LOAD 0F5D 0208
KENVFULL 036B 0131
KERR 035E 0097
MODE 000C 0163 0166 0196
NEWINCR 0002 0123 0151
NEWLN 0003 0089 0121 0124
NEWLN1 000C 0036 0122 0132 0133 0148 0150 0152 0160
NEWLOW 0001 0084 0126 0147
NEWLN 000D 0144 0157
PIA 0332 0105
PIB 0357 0103
PIC 0362 0106
PTA 036C 0191
PASS1 031C 0127
PASS2 0366 0092
PPL 00CA 0204
PEDEC 051B 0111 0118
PENDX 0009 0089 0090 0099 0100
PENDX1 009A 0093 0098 0101
PENDX2 0007 0139 0142 0143 0159 0161 0165 0168 0173 0174 0176 0177
FIL 00CC 0080
ROH 0001 0109
ROL 0000 0110
R1H 0017 0116
RUL 0016 0117
R13 0000 0184 0185
RANGERR 0E68 0129
REH 000C 0180 0181
RENUM 0308
RENX 0300
ERR 035A 0134
SCRB 0008 0077 0078
SCR9 0009 0080 0081 0087 0204 0205
SCR 000C 0201 0202 0206 0210 0212
SKIPASC 0394 0175 0179 0182
STRESH 0003 0162 0178
SW16 0589 0070 0076 0120 0200 0209
STRIM 0007 0068 0096
TLD 0006 0154
TDL 0006 0083 0094 0107
TDL 0083 0146 0153 0156
TDL 0005 0022 0145
TEN 000D 0187 0188

Distributed under the Creative Commons License on page 5
Page 0194 of 0213
<table>
<thead>
<tr>
<th>CROSS REFERENCE:</th>
<th>APPLE-II BASIC RENUMBER/APPEND SUBROUTINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>UD2</td>
<td>0377</td>
</tr>
<tr>
<td>UD3</td>
<td>0385</td>
</tr>
<tr>
<td>UPDATE</td>
<td>0372</td>
</tr>
<tr>
<td></td>
<td>0167</td>
</tr>
</tbody>
</table>
This page is not part of the original Wonderbook
The Woz Wonderbook

References

03 November 2004

This page is not part of the original Wonderbook
This page is not part of the original Wonderbook
The Woz Wonderbook

References
David T Craig • 03 November 2004

Here is a list of Apple Computer and Apple-II computer technical and historical reference materials that may prove beneficial to readers of the Woz Wonderbook who want to know more about the details behind this document and the Apple-II computer in the late 1970's.

These more polished references originated in publicly published Apple Computer documents, magazine articles, and Apple-II enthusiast private materials.

David T Craig (shirlgato@cybermesa.com) has digital copies of all of these materials. These materials may possibly be provided with DigiBarn's Woz Wonderbook digital materials via its web site or a CD.

SYSTEM DESCRIPTION: THE APPLE-II
Steve Wozniak • BYTE Magazine • May 1977

This description of the Apple-II computer by its main designer provides a concise description of this computer's technical features.

MICROCOMPUTER FOR USE WITH VIDEO DISPLAY

This is Apple Computer's patent for the Apple-II computer assigned to Steve Wozniak. Dry reading, but has some good Apple-II technical information. Available on the US Patent Office web site http://www.uspto.gov/patft/.

APPLE-II HISTORY
Steven Weyhrich • http://apple2history.org/history/ • 1991-2003

This great web site contains a cornucopia of accurate Apple-II historical information. If you want to learn about the origins of Apple Computer and the Apple-I and Apple-II computers, this is the place to go. Also available on the internet at http://www.blinkenlights.com/classiccmp/apple2history.html.

SWEET-16: THE 6502 DREAM MACHINE
Steve Wozniak • BYTE Magazine • November 1977

This is Steve Wozniak's comprehensive description of his SWEET-16 16-bit byte-code "meta microprocessor" interpreter built into the Apple-II Integer BASIC ROM. Wozniak's Apple-II system description in BYTE May 1977 also has a short description of SWEET-16.

This page is not part of the original Wonderbook
APPLE-II REFERENCE MANUAL ("RED BOOK")
Apple Computer • January 1978

This is Apple Computer's first published technical reference manual for the Apple-II computer. It is commonly referred to as the "Red Book" because it has a red cover. The Red Book's contents (155 pages) were based on the Woz Wonderbook but in a more polished format, but is not as comprehensive or readable as the later Apple-II reference manuals. A good PDF scan of the Red Book can be found on the internet at http://bitsavers.org/pdf/apple/ along with several other older Apple-II manuals.

APPLE-II REFERENCE MANUAL
Apple Computer • 1979 • Document # 030-0004-01

This is Apple Computer's first revision of the Apple-II Red Book. This 275 page manual is much improved over the Red Book and tremendously improved over the Woz Wonderbook materials. Note the Apple document number (030-0004-01) which indicates this is a technical manual (030), is document number 4 (0004), and is revision 1 (01) which means this is Apple's 4th published manual.

APPLE-II MINI MANUAL
Apple Computer • 1977-1978

This 68 page manual from Apple Computer appears to be the predecessor to the Red Book from 1978. As such, I would date this manual in the 1977-1978 range. More complete and more detailed than the Woz Wonderbook, but not as good as the Red Book. A good PDF scan of this manual can be found on the internet at http://bitsavers.org/pdf/apple/.

THE WOZ PAK ]
Call-A.P.P.L.E. Magazine • 15 November 1979

This 138 page document contains a large number of technical documents about the Apple-II computer courtesy of Apple Computer and Call-A.P.P.L.E. magazine. This is better organized and more comprehensive than the Woz Wonderbook or the Red Book, but not as good as the Apple-II Reference Manual from 1979. Contains a detailed article on the Apple-II floating point package.

PEEKING AT CALL-A.P.P.L.E.
Call-A.P.P.L.E. Magazine • 1978 and 1979

This 2 volume set (volume 1 dated 1978 has 92 pages, volume 2 dated 1979 has 206 pages) contains lots of Apple Computer re-produced technical information and original Call-A.P.P.L.E. magazine information. Well worth reading.
PROGRAMMER'S AID #1:
INSTALLATION AND OPERATING MANUAL
Apple Computer • 1978 • Document # 030-0026-01

This 113 page Apple manual describes the special programming built into the Programmer's Aid #1 ROM chip (there was never an Aid #2 chip AFAIK). Includes several 6502 assembly language programs by Steve Wozniak which used his SWEET-16 16-bit byte-code interpreter. Includes more polished information for the Integer BASIC renumber and append programs described in the Woz Wonderbook.

FLOATING POINT ROUTINES FOR THE 6502
Steve Wozniak & Roy Rankin
Dr. Dobb's Journal of Computer Calisthenics & Orthodontia • August 1976

This is an article on the Apple-II floating point package pre-dating the Woz Wonderbook. Has more details about this package than the Wonderbook. Available on the internet at www.strotmann.de/twiki/bin/view/APG/AsmAppleFloatingPoint. Concerning authorship of this floating point package, web site http://linux.monroeccc.edu/~paulsm/dg/dg32.htm says Wozniak wrote the core package routines (e.g. ADD) and Rankin wrote the transcendental routines (e.g. LOG).

DISASSEMBLER PROGRAM FOR THE 6502
Steve Wozniak & Allen Baum
Dr. Dobb's Journal of Computer Calisthenics & Orthodontia • September 1976

This is an article on the Apple-II 6502 disassembler pre-dating the Woz Wonderbook. Available on the internet at http://users.telenet.be/kiml-6502/kun/i14/p06.html.

THE APPLE II PLUS PERSONAL COMPUTER SYSTEM
Apple Computer • November 1981

This is Apple Computer's data sheet for the Apple-II Plus computer, the successor to the Apple-II computer. Shows how some of the enhancement ideas documented in the Woz Wonderbook and the Red Book were implemented by Apple.

PRELIMINARY APPLE BASIC USERS MANUAL
Apple Computer • October 1976

This 16 page manual seems to be Apple Computer's first user manual for its Apple-II Integer BASIC programming language. The Woz Wonderbook is very lacking in Integer BASIC information for the user. A good PDF scan of this manual can be found on the internet at http://bitsavers.org/pdf/apple/.

This page is not part of the original Wonderbook
APPLE TECH NOTES

Apple Computer and the International Apple Core (IAC) • July 1982

This 500 page document contains an extensive collection of Apple Computer technical notes from 1982 covering the Apple-II and Apple-III computer families. Many Apple-II hardware, software and documentation errata details are here. Includes articles about the Apple-II mini-assembler and cassette interfacing. A treasure trove of early Apple system technical information.

APPLE-II SYSTEM MONITOR ROM LISTING

Apple Computer • 1977

For detailed information about the internal software workings of the Apple-II computer the source listing for the Apple-II System Monitor ROM is the key. Available in the Apple-II reference manual dated 1979 or on the internet at http://members.buckeye-express.com/marksm/6502/.

STEVE WOZNIAK INTERVIEW: HOMEBREW TO CHAMPAGNE

Apple Orchard Magazine • Spring 1981

An early interview with Steve Wozniak in which he provides contemporary details about Apple Computer’s origins and early days.

STEVE WOZNIAK INTERVIEW: THE APPLE STORY

BYTE Magazine • December 1984

A great interview with Steve Wozniak by BYTE magazine with lots of Apple Computer and Apple-II information. Also includes a retrospective on SWEET-16, Wozniak’s 16-bit byte-code interpreter. This is available on the internet at http://apple2history.org/museum/articles/byte8412/byte8412.html.

STEVE WOZNIAK INTERVIEW: STEVE WOZNIAK UNBOUND

SlashDot Interview • January 2000

http://slashdot.org/interviews/00/01/07/1124211.shtml

This 2000 interview of Steve Wozniak contains some good 24 year recollections about Apple Computer’s origins and early years.

This page is not part of the original Wonderbook
The Woz Wonderbook

DOCUMENT

Bill Goldberg Interview

19 April 2004

This page is not part of the original Wonderbook
The Woz Wonderbook

Bill Goldberg Interview

Bruce Damer • 19 April 2004

Source:
http://www.digibarn.com/collections/books/woz-wonderbook/goldberg-on-woz-wonderbook.mp3 (3.3 MB file)

Transcript created by
David T Craig <shirlgato@cybermesa.com> -- 02 November 2004

Interviewer: Bruce Damer <bdamer@digitalspace.com>
Interviewee: Bill Goldberg <billau@coastside.net>

Interview duration: 3:39 minutes

BACKGROUND

The "Woz Wonderbook" was a compilation of notes from Steve Wozniak's filing cabinet that served as the first documentation and technical support manual for the Apple II computer (before the more famous "red book" of January 1978). Bill Goldberg, longtime Apple employee, donated his copy of the Wonderbook to the Digibarn (thanks Bill!). At the time he was at Apple there was only a single copy of this thick binder of photocopied notes, diagrams and such to be found in the Apple library. Bill, being in the technical support role and a natural pack rat, made a copy of the Wonderbook.

INTERVIEW TRANSCRIPT

BILL GOLDBERG: Here it's faded. This is the Woz Wonderbook. And its disorganized but I found the copy of this in the Apple library and immediately made some copies of it.

BRUCE DAMER: So this was before the Red Book?

BILL GOLDBERG: This is what the Red Book was made from.

BRUCE DAMER: Oh gosh.

BILL GOLDBERG: Actually, I've got one or two Red Books for you.

BRUCE DAMER: Wonderful, because the Red Book we have is on loan.

BILL GOLDBERG: Actually, in Service Engineering we would get the leftovers of things. People would say "we don't need any more of this". So we had two cases of Red Books and a few of us in the department said "Hmm, these are worth something" and we divided them up.

This page is not part of the original Wonderbook
BRUCE DAMER: Wow.

BILL GOLDBERG: So, anyway, in here you will find some of the stuff typed, a number of different articles, but you will also find, unfortunately the xerox did the best job it could and it has faded over the years, but there's handwritten notes.

BRUCE DAMER: So Woz wrote these notes?

BILL GOLDBERG: Uh-Hmm [yes]. Here's a listing with some hand disassembly and his comments. Article on the disassembler.

BRUCE DAMER: So this is Woz's hand notes?

BILL GOLDBERG: Well, it's hand notes, it's various articles.

BRUCE DAMER: Here's a disassembled disassembler.

BILL GOLDBERG: Uh ha. But all written by hand.

BRUCE DAMER: Written by hand. Yup.

BILL GOLDBERG: And let's see. Here for instance, here's an article on the cassette system.

BRUCE DAMER: Ok.

BILL GOLDBERG: We (he?) gave up on using the cassette, but this actually is his handwritten notes on the cassette system. So ...

BRUCE DAMER: This is a big book. He must have sat for hours writing this down.

BILL GOLDBERG: You know, somebody just went through a file drawer of his notes and put it in a binder.

BRUCE DAMER: Oh.

BILL GOLDBERG: And there was only one of these in the Apple library. So ...

BRUCE DAMER: Wow.

BILL GOLDBERG: Either I or one of my colleagues checked it out and made some copies because this was going to disappear into obscurity.

BRUCE DAMER: This is the Woz Wonderbook?

BILL GOLDBERG: This is what it was called on the spine.

BRUCE DAMER: This would have been [19]77?

This page is not part of the original Wonderbook.
BILL GOLDBERG: Uhm, actually the first article on the first of this has a date of 9/20/77 [20 September 1977]. So, but this is just a collection of a lot of different ... this actually goes into explaining ...

BRUCE DAMER: Yeah ...

BILL GOLDBERG: Yup. The detail that, you know ... I'm sure some of this is hideously proprietary but who will ever know.

BRUCE DAMER: Well, not at this point.

BILL GOLDBERG: Ok, so that's the Woz ...

BRUCE DAMER: Woz ...

BILL GOLDBERG: Actually, it won't hurt to write on the spine ... here take a pen, so it's in your handwriting. That was my handwriting, so there's nothing special about that.

###
This page is not part of the original Wonderbook
The Woz Wonderbook

DOCUMENT

Credits

This page is not part of the original Wonderbook
The Woz Wonderbook

Credits

Thanks to Bill Goldberg for donating this copy of the Woz Wonderbook.

The DigiBarn Computer Museum and Curator Bruce Damer for providing it to the education and research community.

David T Craig is to be thanked for resurrecting the Wonderbook into a modern digital format.

And of course, thanks to Steve Wozniak for creating the Woz Wonderbook!

Steve Wozniak circa 1977 and 1981

This page is not part of the original Wonderbook
This page is not part of the original Wonderbook
The Woz Wonderbook

The End

This page is not part of the original Wonderbook