*Listing 2, continued:*

```
0950: 49 FF     00218          EOR  #$FF
0952: F0 C4     00219          BEQ  BR1      BRANCH IF SO
0954: 60        00220          RTS
0955: 0A        00221 BNM1     ASL  A        DOUBLE RESULT-REG INDEX
0956: AA        00222          TAX
0957: B5 00     00223          LDA  R0L,X
0959: 35 01     00224          AND  R0H,X    CHK BOTH BYTES FOR NO $FF
095B: 49 FF     00225          EOR  #$FF
095D: D0 B9     00226          BNE  BR1      BRANCH IF NOT MINUS 1
095F: 60        00227 NUL      RTS
0960: A2 10     00228 RS       LDX  #$10     12+2 FOR R12 AS STK PNTR
0962: 20 DD 08  00229          JSR  DCR      DECR STACK POINTER
0965: A1 00     00230          LDA  (R0L,X)  POP HIGH RETURN ADR TO PC
0967: 85 1F     00231          STA  R1SH
0969: 20 DD 08  00232          JSR  DCR      SAME FOR LOW-ORDER BYTE
096C: A1 00     00233          LDA  (R0L,X)
096E: 85 1E     00234          STA  R1SL
0970: 60        00235          RTS
0971: 4C 3E 08  00236 RTN      JMP  RTNI
                00237 *
                00238 * REG SAVE/RESTORE ROUTINES
                00239 * FOR NON-APPLE-II SYSTEMS
                00240 *
                00241 ASAV     EPZ  $45
                00242 XSAV     EPZ  $46
                00243 YSAV     EPZ  $47
                00244 PSAV     EPZ  $48
0974: 85 45     00245 SAVE     STA  ASAV
0976: 86 46     00246          STX  XSAV     SAVE 6502 REG CONTENTS.
0978: 84 47     00247          STY  YSAV
097A: 08        00248          PHP
097B: 68        00249          PLA
097C: 85 48     00250          STA  PSAV
097E: 60        00251          RTS
097F: A5 48     00252 RESTORE  LDA  PSAV
0981: 48        00253          PHA
0982: A5 45     00254          LDA  ASAV
0984: A6 46     00255          LDX  XSAV     RESTORE 6502 REG CONTENTS.
0986: A4 47     00256          LDY  YSAV
0988: 28        00257          PLP
0989: 60        00258          RTS
```

*Table 1:*

**SWEET16 OP CODE SUMMARY**

| | Register Ops | | | | Nonregister Ops | |
|---|---|---|---|---|---|---|
| | | | | 00 | RTN | (Return to 6502 mode) |
| 1n | SET | Rn | Constant (Set) | 01 | BR ea | (Branch always) |
| 2n | LD | Rn | (Load) | 02 | BNC ea | (Branch if No Carry) |
| 3n | ST | Rn | (Store) | 03 | BC ea | (Branch if Carry) |
| 4n | LD | @Rn | (Load indirect) | 04 | BP ea | (Branch if Plus) |
| 5n | ST | @Rn | (Store indirect) | 05 | BM ea | (Branch if Minus) |
| 6n | LDD | @Rn | (Load double indirect) | 06 | BZ ea | (Branch if Zero) |
| 7n | STD | @Rn | (Store double indirect) | 07 | BNZ ea | (Branch if NonZero) |
| 8n | POP | @Rn | (Pop indirect) | 08 | BM1 ea | (Branch if Minus 1) |
| 9n | STP | @Rn | (Store pop indirect) | 09 | BNM1 ea | (Branch if Not Minus 1) |
| An | ADD | Rn | (Add) | 0A | BK ea | (Break) |
| Bn | SUB | Rn | (Sub) | 0B | RS | (Return from Subroutine) |
| Cn | POPD | @Rn | (Pop double indirect) | 0C | BS ea | (Branch to Subroutine) |
| Dn | CPR | Rn | (Compare) | 0D | | (Unassigned) |
| En | INR | Rn | (Increment) | 0E | | (Unassigned) |
| Fn | DCR | Rn | (Decrement) | 0F | | (Unassigned) |

**SWEET16 Operation Code Summary:** Table 1 summarizes the list of SWEET16 operation codes, which are explained in further detail one by one in the descriptions which follow the table. The program of listing 2 implements the execution of these interpretive codes after a call to the entry point **SW16.** Return to the calling program and normal noninterpretive operation is accomplished with the RTN mnemonic of SWEET16.

**SWEET16 – REGISTER OPERATIONS**

SET Rn, Constant [1 n] [low] [high] (Set)

constant

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

Example:

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

LD Rn [2 n] (Load)

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

Example:

```
15 34 A0        SET   R5, A034
25              LD    R5            ACC now contains A034
```

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 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 least significant bit is zero. If ADD, SUB or CPR instructions generate carries, then this index is incremented, setting the least significant bit, which becomes a carry flag.

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

Most SWEET16 nonregister operations are relative branches. The corresponding subroutines determine whether or not the "prior result" meets the specified branch condition and if so update the SWEET16 program counter by adding the displacement value (−128 to +127 bytes).

The RTN operation restores the 6502 register contents, pops the subroutine return stack and jumps indirect through the SWEET16 program counter register. This transfers control to the 6502 at the instruction immediately following the RTN instruction.

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

Any number of subroutine levels may be implemented within SWEET16 code via the BS (Branch to Subroutine) and RS (Return from Subroutine) instructions. The user must initialize and otherwise not disturb R12 if the SWEET16 subroutine capability is used since it is utilized as the automatic subroutine return stack pointer.

### Memory Allocation and User Modifications

The only storage that must be allocated for SWEET16 variables are 32 consecutive locations in page zero for the SWEET16 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.

You may wish to add some of your own

**Text continued on page 159**



ST Rn [3 n] (Store)

The ACC (R0) 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
36              ST    R6            of R5 to R6.
```

LD @Rn [4 n] (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.

Example:

```
15 34 A0        SET   R5, A034
45              LD    @R5           ACC is loaded from
                                    memory location A034
                                    and R5 is incremented
                                    to A035.
```

ST @Rn [5 n] (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 pointers R5 and R6
16 22 90        SET   R6, 9022      with A034 and 9022.
45              LD    @R5           Move a byte from location
56              ST    @R6           A034 to location 9022. Both
                                    pointers are incremented.
```

LDD @Rn [6 n] (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 ACC byte is
                                    loaded from location A034,
                                    the high order byte from
                                    location A035. R5 is incre-
                                    mented to A036.
```

STD @Rn [7 n] (Store double byte indirect)

The low order ACC byte is stored into the 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 pointers R5 and R6
16 22 90        SET   R6, 9022      with A034 and 9022. Move
65              LDD   @R5           double byte from locations
76              STD   @R6           A034 and A035 to locations
                                    9022 and 9023. Both pointers
                                    are incremented by 2.
```