

Listing 2, continued:

```

00201 49 FF 00210 EOR #5FF BRANCH IF SO
00202 70 CA 00211 BIC Rn1
00203 50 00212 R15
00204 2A 00213 ASL A DOUBLE RESULT-REG INDEX
00205 4A 00214 TAX
00206 80 00215 LDA R0L,X
00207 35 00216 AND R0L,Y GEX BOTH BYTES FOR NO SFF
00208 49 FF 00217 LDR #5FF
00209 50 00218 SWX R0L BRANCH IF NOT MINUS 1
00210 60 00219 R15
00211 42 10 00220 LDR #318 12*2 FOR #12 AS STK PTR
00212 20 00 00 00221 JSR DCR DECR STACK POINTER
00213 A1 00 00222 LDA (R0L+X) POP HIGH RETURN ADDR TO PC
00214 35 1F 00223 STA R15H
00215 20 00 00 00224 JSR DCR SANE FOR LOW-ORDER BYTE
00216 A1 00 00225 LDA (R0L+X)
00217 35 1E 00226 STA R15L
00218 60 00227 RTS
00219 4C 3E 00 00228 RTN JRP RTNE
00229 * REG SAVE/RESTORE ROUTINES
00230 * FOR NON-APPLE-II SYSTEMS
00231 *
00232 ASAV EPR $45
00233 XSAV EPR $46
00234 YSAV EPR $47
00235 PSAV EPR $48
00236 SAVE STA ASAV
00237 STX XSAV SAVE 6502 REG CONTENTS.
00238 STY YSAV
00239 POP R15
00240 PLA
00241 STA PSAV
00242 RTS
00243 AS 48 00244 RESTORE LDA PSAV
00245 48 00245 STA ASAV
00246 45 45 00246 LDA ASAV
00247 46 46 00247 STA XSAV
00248 47 47 00248 LDA XSAV
00249 48 48 00249 STA YSAV
00250 49 49 00250 PLP
00251 50 00251 RTS
00252 50 00252 RTN
00253 50 00253 RTN
00254 50 00254 RTN
00255 50 00255 RTN
00256 50 00256 RTN
00257 50 00257 RTN
00258 50 00258 RTN

```

Table 1:

SWEET16 OP CODE SUMMARY

Register Ops	Nonregister Ops
1n SET Rn Constant (Set)	00 RTN (Return to 6502 mode)
2n LD Rn (Load)	01 BR ea (Branch always)
3n ST Rn (Store)	02 BNC ea (Branch if No Carry)
4n LD @Rn (Load indirect)	03 BC ea (Branch if Carry)
5n ST @Rn (Store indirect)	04 BP ea (Branch if Plus)
6n LDD @Rn (Load double indirect)	05 BM ea (Branch if Minus)
7n STD @Rn (Store double indirect)	06 BZ ea (Branch if Zero)
8n POP @Rn (Pop indirect)	07 BNZ ea (Branch if NonZero)
9n STP @Rn (Store pop indirect)	08 BM1 ea (Branch if Minus 1)
An ADD Rn (Add)	09 BNM1 ea (Branch if Not Minus 1)
Bn SUB Rn (Sub)	0A BK ea (Break)
Cn POPD @Rn (Pop double indirect)	0B RS (Return from Subroutine)
Dn CPR Rn (Compare)	0C BS ea (Branch to Subroutine)
En INR Rn (Increment)	0D (Unassigned)
Fn DCR Rn (Decrement)	0E (Unassigned)
	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)
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
	36	ST R6
		Copy the contents 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
	16 22 90	SET R6, 9022
	45	LD @R5
	56	ST @R6
		Load pointers R5 and R6 with A034 and 9022. Move a byte from location 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 incremented 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
	16 22 90	SET R6, 9022
	65	LDD @R5
	76	STD @R6
		Load pointers R5 and R6 with A034 and 9022. Move double byte from locations A034 and A035 to locations 9022 and 9023. Both pointers are incremented by 2.

## FINALLY. A State-of-the-Art Tool For Learning Software Design.

And at an affordable price. The Modu-Learn™ home study course from Logical Services.

Now you can learn microcomputer programming in ten comprehensible lessons. At home. In your own time. At your own pace.

You learn to solve complex problems by breaking them down into easily programmed modules. Prepared by professional design engineers, the Modu-Learn™ course presents systematic software design techniques, structured program design, and practical examples from real 8080A micro-computer applications. All in a modular sequence of 10 lessons... more than 500 pages, bound into one practical notebook for easy reference.

You get diverse examples, problems, and solutions. With thorough background material on micro-computer architecture, hardware/software trade-offs, and useful reference tables. All for only \$49.95.

For \$49.95 you learn design techniques that make software work for you. Modu-Learn™ starts with the basics. Our problem-solution approach enables you to "graduate" as a programmer.

See Modu-Learn™ at your local computer store or order now using the coupon below.

Please send the Modu-Learn™ course for me to examine. Enclosed is \$49.95 (plus \$2.00 postage and handling) or my Mastercharge/BankAmericard authorization.

Name: \_\_\_\_\_  
Address: \_\_\_\_\_  
City: \_\_\_\_\_ State: \_\_\_\_\_  
Card #: \_\_\_\_\_  
Expiration date: \_\_\_\_\_  
Signature: \_\_\_\_\_



711 Stierlin Road  
Mountain View, CA 94043  
(415) 965-8365

**LOGICAL**  
SERVICES INCORPORATED