Stephen Wozniak
Apple Computer
20863 Stevens Creek Blvd, B3C
Cupertino CA 95014

SWEET16: The 6502 Dream Machine

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 hybrid of the MOS Technology 6502 and RCA 1800 architectures, 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 nonexistent 16 bit "metaprocessor" in software, interpreter style, which I call SWEET16. This metaprocessor was sketched at the end of my article in May 1977 BYTE, and the purpose of this article is to fill in the details of SWEET16.

SWEET16 is based around sixteen 16 bit registers called R0 to R15, actually implemented as 32 memory locations. R0 doubles as the SWEET16 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 SWEET16 subroutines are used. All other SWEET16 registers are at the user's unrestricted disposal.

SWEET16 instructions fall into register and nonregister categories. The register operations specify one of the 16 registers to be used as either a data element or a pointer to data in memory depending on the specific instruction. For example, the instruction INR R5 uses R5 as data and ST @R7 uses R7 as a pointer to data in memory. Except for the SET instruction, register operations only require one byte. The nonregister operations are primarily 6502 style branches with the second byte specifying a ±127 byte displacement relative to the address of the following instruction. If a prior register operation result meets a specified branch condition, the displacement is added to SWEET16's program counter, effecting a branch.

SWEET16 is intended as a 6502 enhancement package, not a stand alone processor. A 6502 program switches to SWEET16 mode with a subroutine call, and subsequent code is interpreted as SWEET16 instructions. The nonregister operation RTN returns the user program to the 6502's direct execution mode after restoring the internal register contents (A, X, Y, P and S). The example of listing 1 illustrates how to use SWEET16 in some program segment.

```
SWEET16
   300 B9 00 02          LDA  IN,Y      Get a char.
   303 C9 CD             CMP  "M"       "M" for move?
   305 D009              BNE  NOMOVE      No, skip move.
   307 20 00 08          JSR  SW16        Yes, call SWEET16.
   30A 41       MLOOP    LD   @R1       R1 holds source address.
   30B 52                ST   @R2       R2 holds dest. address.
   30C F3                DCR  R3        Decrement length.
   30D 07 FB             BNZ  MLOOP       Loop until done.
   30F 00                RTN            Return to 6502 mode.
   310 C9 C5    NOMOVE   CMP  "E"       "E" char?
   312 D013              BEQ  EXIT        Yes, exit.
   314 C8                INY              No, continue.
```

Note: Registers A, X, Y, P and S are not disturbed by SWEET16.

Listing 1: Use of SWEET16 within an assembly language program is accomplished by executing a subroutine call to the SWEET16 entry point (address 307 here). This call preserves the processor registers at the time of entry and begins interpretive execution. End of interpretive execution is signaled by a RTN operation code of SWEET16, at which point all the processor registers will be restored.

Instruction Descriptions

The SWEET16 op code list is short and uncomplicated. Excepting relative branch displacements, hand assembly is trivial. All register op codes are formed by combining two hexadecimal digits, one for the op code and one to specify a register. For example, op codes 15 and 45 both specify register R5 while codes 23, 27 and 29 are all ST (store) operations. Most register operations of SWEET16 are assigned to numerically adjacent pairs to facilitate remembering them. Thus LD and ST are op codes 2n and 3n respectively, while LD @ and ST @ are codes 4n and 5n.

Operation codes 00 to 0C (hexadecimal) are assigned to the 13 nonregister operations. Except for RTN (op code 0), BK (0A), and RS (B), the nonregister operations are 6502 style relative 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 operation result, the displacement is added to the program counter effecting a branch. Except for BR (Branch always) and BS (Branch to Subroutine), the branch operation codes are assigned in complementary pairs, rendering them easily remembered for hand coding. For example, Branch if Plus and Branch if Minus are op codes 04 and 05, while Branch if Zero and Branch if NonZero are op codes 06 and 07.

Theory of Operation

SWEET16 execution mode begins with a subroutine call to SW16 (see listing 2, an assembly of SWEET16). The user must insure that the 6502 is in hexadecimal mode upon entry. [*For those unfamiliar with the 6502, arithmetic is either decimal or hexadecimal (binary) depending on a programmable flag. . .*CH] All 6502 registers are saved at this time, to be restored when a SWEET16 RTN instruction returns control to the 6502. If you can tolerate indefinite 6502 register contents upon exit, approximately 30 µs may be saved by entering SWEET16 at location SW16 + 3. Because this might cause an inadvertent switch from hexadecimal to decimal mode, it is advisable to enter at SW16 the first time through.

After saving the 6502 registers, SWEET16 initializes its program counter (R15) with the subroutine return address off the 6502 stack. SWEET16's program counter points to the location preceding the next instruction to be executed. Following the subroutine call are 1 byte, 2 byte, or 3 byte long SWEET16 instructions, stored in ascending

Listing 2: SWEET16 assembly. The SWEET16 program, assembled to reside at location 800 hexadecimal, is presented by this listing. The primary entry point is at the beginning, location SW16. An alternate entry point if there is no need to save processor registers is at location 803 in this assembly, SW16+3.

```
                        SWEET16 INTERPRETER
11:18 A.M.,  THU, MAY 12, 1977

                00001 ************************
                00002 *                      *
                00003 *   APPLE-II PSEUDO    *
                00004 * MACHINE INTERPRETER  *
                00005 *                      *
                00006 *     S. WOZNIAK       *
                00007 * APPLE COMPUTER INC   *
                00008 *                      *
                00009 ************************
                00010  TITLE "SWEET16 INTERPRETER"
                00011 R0L    EPZ  $0
                00012 R0H    EPZ  $1
                00013 R14H   EPZ  $1C
                00014 R15L   EPZ  $1E
                00015 R15H   EPZ  $1F
                00016 S16PAG EQU  $FF
                00017        ORG  $800
0800: 20 T4 09  00018 SW16   JSR  SAVE      PRESERVE 6502 REG CONTENTS
0803: 68        00019        PLA
0804: 85 1E     00020        STA  R15L      INIT SWEET16 PC
0806: 68        00021        PLA            FROM RETURN
0807: 85 1F     00022        STA  R15H        ADDRESS
0809: 20 0F 08  00023 SW16B  JSR  SW16C     INTERPRET AND EXECUTE
080C: 4C 09 08  00024        JMP  SW16B     ONE SWEET16 INSTR.
080F: E6 1E     00025 SW16C  INC  R15L
0811: D0 02     00026        BNE  SW16D     INCR SWEET16 PC FOR FETCH
0813: E6 1F     00027        INC  R15H
0815: A9 F7     00028 SW16D  LDA  #S16PAG
0817: 48        00029        PHA            PUSH ON STACK FOR RTS
0818: A0 00     00030        LDY  #$0
081A: B1 1E     00031        LDA  (R15L),Y  FETCH INSTR
081C: 29 0F     00032        AND  #$F       MASK REG SPECIFICATION
081E: 0A        00033        ASL  A         DOUBLE FOR 2-BYTE REG'S
081F: AA        00034        TAX            TO X-REG FOR INDEXING
0820: 4A        00035        LSR  A
0821: 51 1E     00036        EOR  (R15L),Y  NOW HAVE OPCODE
0823: F0 0B     00037        BEQ  TOBR      IF ZERO THEN NON-REG OP
0825: 86 1D     00038        STX  R14H      INDICATE'PRIOR RESULT REG'
0827: 4A        00039        LSR  A
0828: 4A        00040        LSR  A         OPCODE*2 TO LSB'S
0829: 4A        00041        LSR  A
082A: A8        00042        TAY            TO Y-REG FOR INDEXING
082B: B9 5B 08  00043        LDA  OPTBL-2,Y LOW-ORDER ADR BYTE
082E: 48        00044        PHA            ONTO STACK
082F: 60        00045        RTS            GOTO REG-OP ROUTINE
0830: E6 1E     00046 TOBR   INC  R15L
0832: D0 02     00047        BNE  TOBR2     INCR PC
0834: E6 1F     00048        INC  R15H
0836: BD 5B 08  00049 TOBR2  LDA  OPTBL+X   LOW-ORDER ADR BYTE
0839: 48        00050        PHA            ONTO STACK FOR NON-REG OP
083A: A5 1D     00051        LDA  R14H      'PRIOR RESULT REG' INDEX
083C: 4A        00052        LSR  A         PREPARE CARRY FOR BC, BNC.
083D: 60        00053        RTS            GOTO NON-REG OP ROUTINE
083E: 68        00054 RTNE   PLA            POP RETURN ADDRESS
083F: 68        00055        PLA
0840: 20 7F 09  00056        JSR  RESTORE   RESTORE 6502 REG CONTENTS
0843: 6C 1E 00  00057        JMP  (R15L)    RETURN TO 6502 CODE VIA PC
0846: B1 1E     00058 SETE   LDA  (R15L),Y  HIGH-ORDER BYTE OF CONST
0848: 95 01     00059        STA  R0H,X
084A: 88        00060        DEY
084B: B1 1E     00061        LDA  (R15L),Y  LOW-ORDER BYTE OF CONSTANT
084D: 95 00     00062        STA  R0L,X
084F: 98        00063        TYA            Y-REG CONTAINS 1
0850: 38        00064        SEC
0851: 65 1E     00065        ADC  R15L      ADD 2 TO PC
0853: 85 1E     00066        STA  R15L
0855: 90 02     00067        BCC  SET2
0857: E6 1F     00068        INC  R15H
0859: 60        00069 SET2   RTS
085A: T9        00070 OPTBL  DFB  SET-1     (1X)
0858: 70        00071 RTNBL  DFB  RTN-1     (0)
085C: TB        00072        DFB  LD-1      (2X)
085D: 14        00073        DFB  BR-1      (1)
085E: 84        00074        DFB  ST-1      (3X)
085F: 15        00075        DFB  BNC-1     (2)
0860: 9C        00076        DFB  LDAT-1    (4X)
0861: 26        00077        DFB  BC-1      (3)
0862: 8D        00078        DFB  STAT-1    (5X)
0863: 29        00079        DFB  BP-1      (4)
0864: BE        00080        DFB  LDDAT-1   (6X)
0865: 38        00081        DFB  BM-1      (5)
0866: C8        00082        DFB  STDAT-1   (7X)
0867: 3T        00083        DFB  BZ-1      (6)
0868: A6        00084        DFB  POP-1     (8X)
0869: 48        00085        DFB  BNZ-1     (T)
086A: DD        00086        DFB  STPAT-1   (9X)
086B: 49        00087        DFB  BM1-1     (8)
086C: FC        00088        DFB  ADD-1     (AX)
086D: 54        00089        DFB  BNM1-1    (9)
086E: E5        00090        DFB  SUB-1     (BX)
086F: TC        00091        DFB  BK-1      (A)
0870: AA        00092        DFB  POPD-1    (CX)
0871: 5F        00093        DFB  RS-1      (B)
0872: E7        00094        DFB  CPR-1     (DX)
0873: 0A        00095        DFB  BS-1      (C)
0874: 95        00096        DFB  INR-1     (EX)
0875: 5E        00097        DFB  NUL-1     (D)
```