
BR ea | 0 1 | d d | (Branch Always)

An effective address (ea) is calculated by adding the signed displacement byte (dd) to the program counter. The program counter contains the address of the instruction immediately *following* the BR, or the address of the BR operation plus 2. The displacement is a signed two's complement value from −128 to +127. Branch conditions are not changed. Note that effective address calculation is identical to that for 6502 relative branches.

Some examples:

```
dd = $80    ea = PC + 2 − 128
dd = $81    ea = PC + 2 − 127
dd = $FF    ea = PC + 2 − 1
dd = $00    ea = PC + 2 + 0
dd = $01    ea = PC + 2 + 1
dd = $7E    ea = PC + 2 + 126
dd = $7F    ea = PC + 2 + 127
```

Example:

```
$300: 01 50      BR  $352
```

BNC ea | 0 2 | d 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 | 0 3 | d d | (Branch if Carry set)

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

BP ea | 0 4 | d 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 loc 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
D5              CPR   R5            pointer.
04 F8           BP    LOOP        Loop until done.
```

BM ea | 0 5 | d d | (Branch if Minus)

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

BZ ea | 0 6 | d d | (Branch if Zero)

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

BNZ ea | 0 7 | d d | (Branch if NonZero)

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

BM1 ea | 0 8 | d d | (Branch if Minus 1)

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

BNM1 ea | 0 9 | d d | (Branch if Not Minus 1)

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

Text continued from page 154

instructions to this implementation of SWEET16. If you use the unassigned op codes $0E and $0F, remember that SWEET16 treats these as 2 byte instructions. You may wish to handle the break instruction as a SWEET16 call, saving two bytes of code each time you transfer into SWEET16 mode. Or you may wish to use the SWEET16 BK (Break) operation as a "CHAROUT" call in the interrupt handler. You can perform absolute jumps within SWEET16 by loading the ACC (R0) with the address you wish to jump to (minus 1) and executing a ST R15 instruction.

And as a final thought, the ultimate modification for those who do not use the 6502 processor would be to implement a version of SWEET16 for some other microprocessor design. The idea of a low level interpretive processor can be fruitfully implemented for a number of purposes, and achieves a limited sort of machine independence for the interpretive execution strings. I found this technique most useful for the implementation of much of the software of the Apple II computer; I leave it to readers to explore further possibilities for SWEET16.■

BRK | 0 A | (Break)

A 6502 BRK (break) instruction is executed. SWEET16 may be reentered nondestructively at SW16D after correcting the stack pointer to its value prior to executing the BRK.

RS | 0 B | (Return from SWEET16 Subroutine)

RS terminates execution of a SWEET16 subroutine and returns to the SWEET16 calling program which resumes execution (in SWEET16 mode). R12, which is the SWEET16 subroutine return stack pointer, is decremented twice. Branch conditions are not changed.

BS ea | 0 C | d d | Branch to SWEET16 Subroutine

A branch to the effective address (PC + 2 + d) is taken and execution is resumed in SWEET16 mode. The current PC is pushed onto a "SWEET16 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)

```
300: 15 34 A0         SET   R5, A034    Init pointer 1.
303: 14 3B A0         SET   R4, A03B    Init limit 1.
306: 16 00 30         SET   R6, 3000    Init pointer 2.
309: 0C 15            BS    MOVE        Call move subroutine.
       .
       .
320: 45         MOVE  LD   @R5          Move one
321: 56               ST   @R6            byte.
322: 24               LD    R4
323: D5               CPR   R5          Test if done.
324: 04 FA            BP    MOVE
326: 0B               RS                Return.
```