POP @Rn | 8 n | (Pop indirect)

The low order ACC byte is loaded from the memory location whose address resides in Rn *after* Rn is decremented by 1 and the high order ACC byte is cleared. Branch conditions reflect the final 2 byte ACC contents which will always be positive and never minus 1. The carry is cleared. Because Rn is decremented prior to loading the ACC, single byte stacks may be implemented with the ST @Rn and POP @Rn operations (Rn is the stack pointer).

Example:

```
15 34 A0      SET   R5, A034    Init stack pointer.
10 04 00      SET   R0, 4         Load 4 into ACC.
35            ST    @R5         Push 4 onto stack.
10 05 00      SET   R0, 5         Load 5 into ACC.
35            ST    @R5         Push 5 onto stack.
10 06 00      SET   R0, 6         Load 6 into ACC.
35            ST    @R5         Push 6 onto stack.
85            POP   @R5         Pop 6 off stack into ACC.
85            POP   @R5         Pop 5 off stack.
85            POP   @R5         Pop 4 off stack.
```

STP @Rn | 9 n | (Store pop indirect)

The low order ACC byte is stored into the memory location whose address resides in Rn *after* Rn is decremented by 1. Then the high order ACC byte is stored into the memory location whose address resides in Rn after Rn is again decremented by 1. Branch conditions will reflect the 2 byte ACC contents which are not modified. STP @Rn and PLA @Rn are used together to move data blocks beginning at the greatest address and working down. Additionally, single byte stacks may be implemented with the STP @Rn and LDA @Rn ops.

Example:

```
14 34 A0      SET   R4, A034    Init pointers.
15 22 90      SET   R5, 9022
84            POP   @R4         Move byte from A033
95            STP   @R5           to 9021.
84            POP   @R4         Move byte from A032
95            STP   @R5           to 9020.
```

ADD Rn | A n | (Add)

The contents of Rn are added to the contents of the ACC (R0) and the low order 16 bits of the sum restored in ACC. The 17th sum bit becomes the carry and other branch conditions reflect the final ACC contents.

Example:

```
10 34 76      SET   R0, 7634    Init R0 (ACC)
11 27 42      SET   R1, 4227      and R1.
A1            ADD   R1          Add R1 (sum = B858,
                                  carry clear)
A0            ADD   R0          Double ACC (R0) to 7086
                                  with carry set.
```

SUB Rn | B n | (Subtract)

The contents of Rn are subtracted from the ACC contents by performing a two's complement addition:

ACC ← ACC + $\overline{Rn}$ + 1

The low order 16 bits of the subtraction are restored in the ACC. The 17th sum bit becomes the carry and other branch conditions reflect the final ACC contents. If the 16 bit unsigned ACC contents are greater than or equal to the 16 bit unsigned Rn contents then the carry is set, otherwise it is cleared. Rn is not disturbed.

Example:

```
10 34 76      SET   R0, 7634    Init R0 (ACC)
11 27 42      SET   R1, 4227      and R1.
A1            SUB   R1          Subtract R1 (diff = 340D
                                  with carry set)
A0            SUB   R0          Clears ACC (R0)
```


POPD @Rn | C n | (POP Double byte indirect)

Rn is decremented by 1 and the high order ACC byte is loaded from the memory location whose address now resides in Rn. Then Rn is again decremented by 1 and the low order ACC byte is loaded from the corresponding memory location. Branch conditions reflect the final ACC contents. The carry is cleared. Because Rn is decremented *prior* to loading each of the ACC halves, double byte stacks may be implemented with the STD @ Rn and POPD @ Rn operations. (Rn is the stack pointer).

Example:

```
15 34 A0      SET   R5, A034    Init stack pointer.
10 12 AA      SET   R0, AA12      Load AA12 into ACC.
75            STD   @R5         Push AA12 onto stack.
10 34 BB      SET   R0, BB34      Load BB34 into ACC.
75            STD   @R5         Push BB34 onto stack.
10 56 CC      SET   R0, CC56      Load CC56 into ACC.
75            STD   @R5
C5            POPD  @R5         Pop CC56 off stack.
C5            POPD  @R5         Pop BB34 off stack.
C5            POPD  @R5         Pop AA12 off stack.
```

CPR Rn | D n | (Compare)

The ACC (R0) contents are compared to Rn by performing the 16 bit binary subtraction ACC-Rn and storing the low order 16 difference bits in R13 for subsequent branch tests. If the 16 bit unsigned ACC contents are greater than or equal to the 16 bit unsigned Rn contents then the carry is set, otherwise it is cleared. No other registers, including ACC and Rn, are disturbed.

Example:

```
15 34 A0      SET   R5, A034    Pointer to memory.
16 BF A0      SET   R6, A0BF    Limit address.
10 00 00 LOOP SET   R0, 0       Zero data.
75            STD   @R5         Clear 2 locs, incr R5 by 2.
25            LD    R5          Compare pointer R5
D6            CPR   R6            to limit R6.
02 F8         BNC   LOOP        Loop if carry clear.
```

INR Rn | E n | (Increment)

The contents of Rn are incremented by 1. The carry is cleared and other branch conditions reflect the incremented value.

Example:

```
15 34 A0      SET   R5, A034    Init R5 (pointer)
10 00 00      SET   R0, 0       Zero to R0.
55            ST    @R5         Clears loc A034 and incrs
                                  R5 to A035.
E5            INR   R5          Incr R5 to A036
55            ST    @R5         Clears loc A036 (not A035)
```

DCR Rn | F n | (Decrement)

The contents of Rn are decremented by 1. The carry is cleared and other branch conditions reflect the decremented value.

Example: (Clear nine bytes beginning at loc A034)

```
15 34 A0      SET   R5, A034    Init pointer.
14 09 00      SET   R4, 9       Init count.
10 00 00      SET   R0, 0       Zero ACC.
55        LOOP ST   @R5         Clear a mem byte.
F4            DCR   R4          Decr count.
07 FC         BNZ   LOOP        Loop until zero.
```

SWEET16 Nonregister Instructions

RTN | 0 0 | (Return to 6502 mode)

Control is returned to the 6502 and program execution continues at the location immediately following the RTN instruction. The 6502 registers and status conditions are restored to their original contents (prior entering SWEET16 mode).