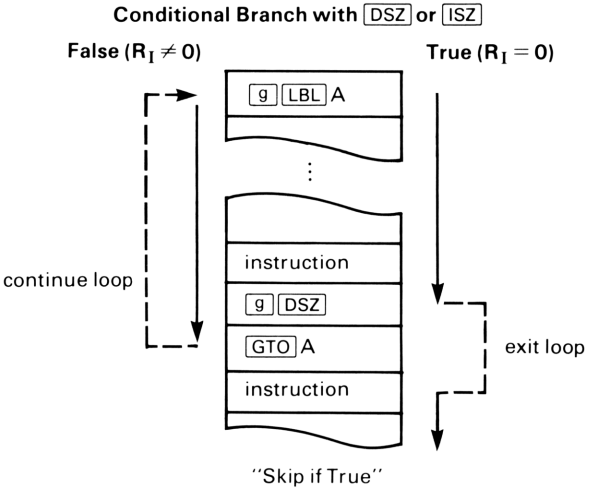
Loop Control with Counters: DSZ and ISZ

The DSZ (*decrement and skip next line if counter equals zero*) and ISZ (*increment and skip if zero*) functions can control loop execution by referencing and adjusting (incrementing/decrementing) a counter value in the Index register. Then, when that counter value reaches zero, program execution skips one line.

Each time one of these functions is encountered in a running program, the given counter value in the Index register is either *decremented* (DSZ) or *incremented* (ISZ) by *one*. If the resulting value equals zero, the next instruction is skipped. This allows exit from a loop if the skipped line was a branch into a loop.

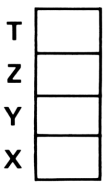


The value in R_I is interpreted according to the current complement mode. It can be positive or negative, in integer or floating-point format. The instructions DSZ and ISZ do not affect the status of the carry and out-of-range flags.

Example

A "checksum" routine can be used to test the integrity of stored data values. Using #B you can determine the sum of a bit pattern and then compare that sum to the sum of the same bit pattern at a later time.

The following program sums all the bits in the bit pattern in a given storage register, yielding a checksum. The contents of storage registers R_A through R_1 are sequentially checksummed. As the bits are summed, they are added to the updated, double-sized checksum being held in registers Y and Z. This is what the stack contains just before line 012:



Z — Current checksum: most significant word.

Y — Current checksum: least significant word.

X — Number whose bits will be summed and added to the current double-word contents in Y and Z.

The resulting checksum will be placed in registers X and Y.

This program uses [DSZ] to decrement a register pointer in the Index register and to control conditional loop branching.

Keystrokes	Display		
[g] [P/R]	000–		
[f] CLEAR [PRGM]	000–		
[g] [LBL] D	001–	43,22, d	
[f] SET COMPL [UNSGN]	002–	42 3	Unsigned mode for adding bits.
4	003–	4	
[f] [WSIZE]	004–	42 44	Word size four bits.
[HEX]	005–	23	
A	006–	A	
[STO] [I]	007–	44 32	Stores number of top register (R_A) in R_I.
0	008–	0	
[ENTER]	009–	36	Initializes checksum to 0.
[g] [LBL] 0	010–	43,22, 0	Start of summing loop. (Enables stack lift.)
[RCL] [(i)]	011–	45 31	Recalls contents of current register whose number is stored in R_I.

Keystrokes	Display		Comments
g #B	012–	43 7	Sums the bits in the X-register.
+	013–	40	Adds this sum to least significant part of current checksum. Might set carry flag.
x≷y	014–	34	Brings most significant part of current checksum into X.
0	015–	0	Places 0 in X.
g RLC	016–	43 C	Places a 1 into X if a carry was generated in the preceding addition.
+	017–	40	Adds carry bit to most significant part of checksum.
x≷y	018–	34	Returns least significant part of checksum to X.
g DSZ	019–	43 23	Decrements the current register number stored in R_I.
GTO 0	020–	22 0	If register number in R_I is not yet zero, then continues with loop.
g RTN	021–	43 21	

Now, calculate an updated checksum (bit summation) given the following 4-bit hexadecimal values in R_1 through R_A:

R_1: A	R_3: B	R_5: 3	R_7: A	R_9: D
R_2: 7	R_4: 1	R_6: D	R_8: 2	R_A: 6

Keystrokes	Display	(STATUS: 0-04-0000)
g P/R		Returns to Run mode.
HEX		
A STO 1	A h	Store the above values in R_1 through R_A.
⋮	⋮	
6 STO A	6 h	

Keystrokes	Display	
GSB D	6 h	Least significant bits of double-word checksum.
x≷y	1 h	Most significant bits: sum of bits in above pattern is 16_{16} or 22_{10}.

When writing or analyzing a program, it is often helpful to use a diagram showing the contents of the stack before and after each instruction. The stack diagrams below show the movement of the stack contents in the loop portion (LBL 0: lines 010 through 019) of the above program.

On the eighth iteration of this loop, the carry is set in step 013 when the checksum for the contents of R_3 is added to the prior checksum (equalling E_{16}), thereby exceeding a single word size. This iteration is shown here. (The A in the T- and Z-registers is a remnant from lines 006 and 007.)

Line ➡	010	011	012	013	014
R_I	3	3	3	3	3
T	A	A	A	A	A
Z	A	0	0	A	A
Y	0	E	E	0	1
X	E	b	3	1	0
Keys ➡	g LBL 0	RCL (i)	g #B	+	x≷y

Line 012 does a checksum of the contents of the register currently addressed by R_I, and line 013 adds this checksum to the least significant part of the checksum. Lines 014 to 017 add in the carry bit from the previous add to the most significant word of the checksum.