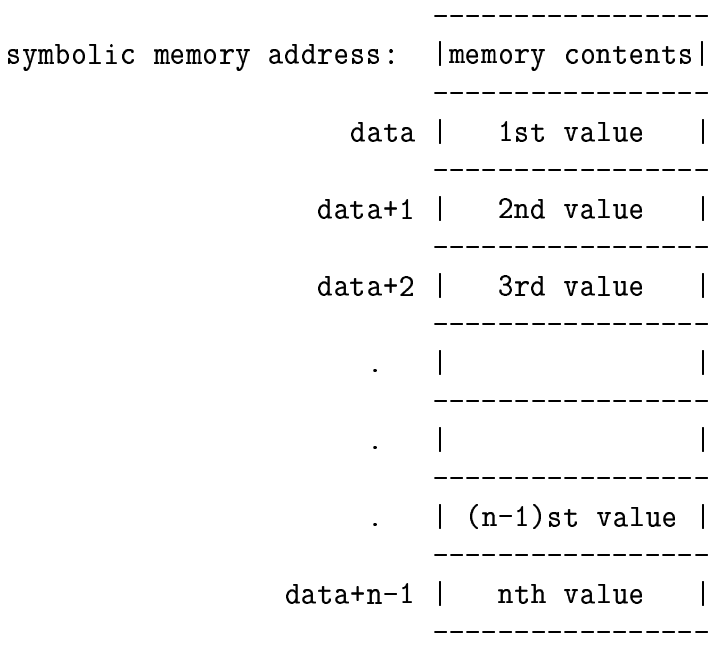
Looping Example and Call By Reference for the MAC-1

This example is that of a function subroutine that adds up an array of data and returns the resulting sum in the accumulator (ac) register. In this case the name of the "n" element array is "data", which is also the symbolic address of the first entry in the n-element array. In this case "data" is a one dimensional array; but even if it were a two dimensional array, or any multidimensional array, with a total of "n" elements (after multiplying the counts of the numbers of elements in the rows and columns, etc.) it would be stored in memory as a single column comprising "n" elements (i.e., words) with symbolic starting address "data" identifying the location of the 1^{st} element of the array, address "data + 1" identifying the 2^{nd} element, "data + 2" identifying the 3^{rd} element, and so on until finally address "data + n - 1" identifies the n^{th} element of the array, as seen in the following figure:



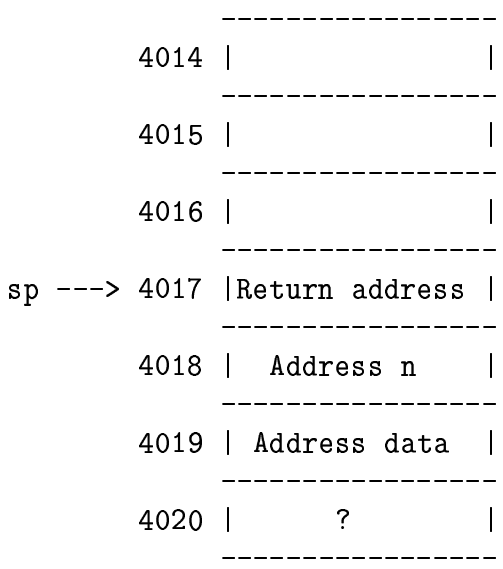
By knowing the base address ("data") one can refer to the elements in the array using indexed addressing. Location "data" has the same address as location "data + 0", and the next location is "data + 1", etc. On the MAC-1 the only register that can be used as an index register is the stack pointer (sp) register because it has instructions "insp" and "desp" that can increment or decrement its contents. On most computers that possess other registers that can be used as index registers it is much safer to leave the stack pointer inviolable; i.e., don't mess with it because a subroutine call caused by a hardware interrupt can occur at any time thus overwriting a memory location in the vicinity where the sp currently points. On the MAC-1, however, interrupts are not possible; so in this case only it is possible to use the stack pointer register as an index register and point it sometimes at the stack (when doing calls and returns and retrieving parameters passed to subroutines there) and point it sometimes elsewhere in memory (such as at the array "data") when it is desirable to do so.

Looping Example and Call by Reference for the MAC-1 Continued

Call by reference will be used to pass addresses as parameters to the called procedure. (This is certainly the case for the array address "data" because it is not efficient in either memory space usage or time to push for example a thousand values from the array data onto the stack instead of just a single base address; but call by value could also be used to communicate the number of elements "n" in the array to the called function because this a a single value.) The form of the call to the function "asum" is as follows:

$$z = \text{asum(n,data)}$$

The sum of the array elements in the array data will be returned in the accumulator (ac) register, and the calling program will (upon return from the function) store the result in location "z". Following the C compiler parameter order convention, we assume that the address "data" is pushed onto the stack first, followed by the address "n". Upon entry to the function "asum" the stack looks like this:



So that the entire function code appears together on one page, the MAC-1 code for function asum is shown on the next page.

Looping Example and Call by Reference for the MAC-1 Continued

```
  /integer function asum
        ENTRY   asum    /Declare label asum as entry point
savsp   RES     1       /Reserve a place to save sp register contents
cnt     RES     1       /Reserve a location for the loop count value
tsum    RES     1       /Reserve a location to hold the accumulated sum
asum    lodl    1       /Pick up first parameter value; address n
        pshi            /Push content of address n onto stack
        pop             /Get contents of n into the ac register
        stod    cnt     /Put value (n) into location cnt
        loco    0       /Clear the ac to zero
        stod    tsum    /Initialize accumulated sum to zero
        lodl    2       /Pick up 2nd parameter value; array base address ‘‘data’’
        swap            /Put address ‘‘data’’ in sp and get sp content into ac
        stod    savsp   /Save sp content where it points into the stack
loop    lodd    cnt     /Pick up loop count value
        subd    (1)     /Decrement the count value
        stod    cnt     /Save the current value of the loop counter
        jneg    done    /If loop count goes negative, we are done
        lodd    tsum    /Pick up current sub total in ac
        addl    0       /Add to ac content of current array element; at m(sp)
        stod    tsum    /Put subtotal away before using ac for another purpose
        insp    1       /Bump up array pointer:  data+1, data+2, etc.
        jump    loop    /Jump back to top of loop until done
done    lodd    savsp   /Pick up pointer back to the stack area of memory
        swap            /Put it into the stack pointer register
        lodd    tsum    /Leave the final array sum value in the ac register
        retn            /Return
        END
```