Passing and returning values to subroutines using the stack

Calling program:

Calling program sets aside space for parameters on the stack, and places values of parameters in this space.

Calling program sets aside space for return value(s) on the stack.
push.w &var1
push.w &var2
sub.w x2, SP

{ input parameters

space for output

return address

output

input parameters

Stack

6x400

SP ->
Calling program calls the subroutine

```plaintext
call x'my subroutine'
```

Runtime places PC value on the stack.

```
Stack
```

- return address
- output
- input parameters

```
PC
space for output
var2
var1

6×400
```

SP →
Subroutine:

Subroutine sets aside space for local variables on the stack.

\[
\begin{array}{c}
\text{SP} ightarrow 0 \text{(SP)} \\
2 \text{(SP)} \\
4 \text{(SP)} \\
6 \text{(SP)} \\
8 \text{(SP)} \\
10 \text{(SP)} \\
6 \times 400
\end{array}
\]

\{ local variables \}
\{ return address \}
\{ output \}
\{ input parameters \}

\text{SWI.W x4, SP}

local variables: \( 0 \text{(SP)}, 2 \text{(SP)} \)
input parameters: \( 8 \text{(SP)}, 10 \text{(SP)} \)
output: \( 6 \text{(SP)} \)

Subroutine uses input parameters local variables and places result in output (S).
before exiting subroutine reclaim space for local variables from the stack

```
add.w $4, SP
```

illustration of stack diagram:

- SP
- PC
- Output from sub
- var2
- var1

{ return address
{ output
{ input parameters

ret of subroutine is executed

```
ret
```
calling program pops the output from the stack

```
pop.w R5
```

calling program reclaims space for input parameters

```
add.w #4, SP
```

![Stack Diagram]

- Stack
- PC
- Output from sub
- var2
- var1

Stack empty

\[ \text{SP} \rightarrow 0x400 \]
Important rule to remember!

If a program (subroutine) puts something on the stack then it is the responsibility of that program (subroutine) to reclaim space for whatever it stored on the stack.
Note:

It is the responsibility of the subroutine to cleanly state in its contract:

* where on the stack it is expecting the input parameters
* where on the stack it should place the output value(s)
Program: Subroutine3-4

;----------------------------------------------------------
;    Subroutine: NoOfOnes
;    Counts the number of ones in the binary representation of a number
;
;    SP -> Return Address
;    Output (word)
;    Input (word)
;----------------------------------------------------------

NoOfOnes:
    sub.w #2, SP                     ; set aside space on the stack for one word
                                    ; 0(SP) is our local variable
                                    ; 2(SP) is PC (return address)
                                    ; 4(SP) is where output goes
                                    ; 6(SP) is input

    mov.w    #0, 4(SP)               ; 4(SP) holds sum of set bits in the number

    mov.w    #0000000000000001b, 0(SP) ; 0(SP) holds bit for testing, start with the lsb

MoreBits:
    bit.w 0(SP), 6(SP)                ; test the bit
    jnc  BitNotSet                    ; if bit is not set then go to label NotSet
    inc.w 4(SP)                       ; if you are here then bit is set, increase sum of bits

BitNotSet:                       ; if you are here then bit is not set
    rla.w 0(SP)                      ; move the bit to test to left by one position
    jnz  MoreBits                   ; Go to label L1 if there are more bits to test
                                    ; loop is exited when 0(SP) becomes zero and there are
                                    ; no more bits to test
    add.w#2, SP                     ; reclaim one word from the stack

    ret
Program calling NoOfOnes

Program: Subroutine-4

.data
result1: .space 2
result2: .space 2

.text

push.w #111100001000011b ; input for subroutine
sub.w #2, SP ; space for output from subroutine
call #NoOfOnes
pop.w&result1 ; pop result from stack
add.w#2, SP ; reclaim stack mem for input

push.w #111111111111111b ; input for subroutine
sub.w #2, SP ; space for output from subroutine
call #NoOfOnes
pop.w&result2 ; pop result from stack
add.w#2, SP ; reclaim stack mem for input

loop: jmp loop