Passing and returning values to subroutines using the stack

Calling program:

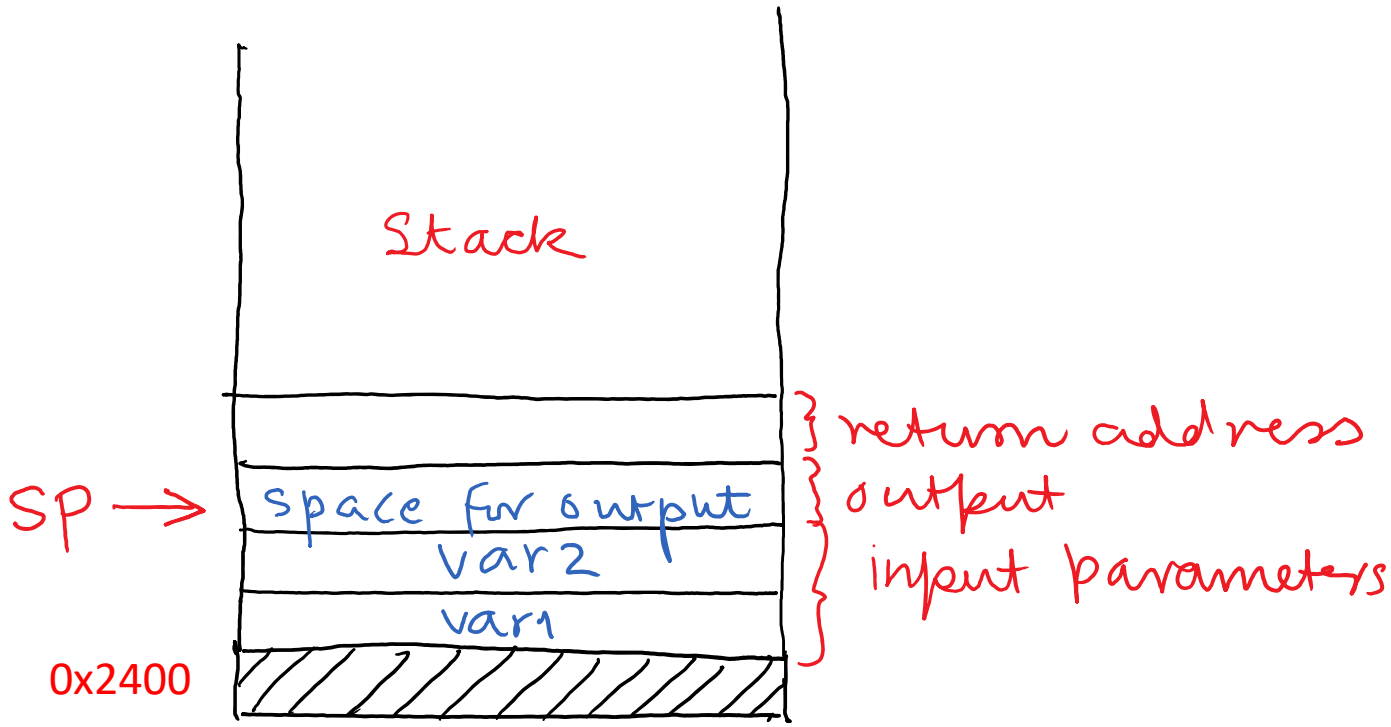
Calling program sets aside space for parameters on the stack, and places values of parameters in stub space

Calling program sets aside space for return value(s) on the stack

```
push.W   &var1
push.W   &var2
sub.W    #2, SP
```

} input parameters

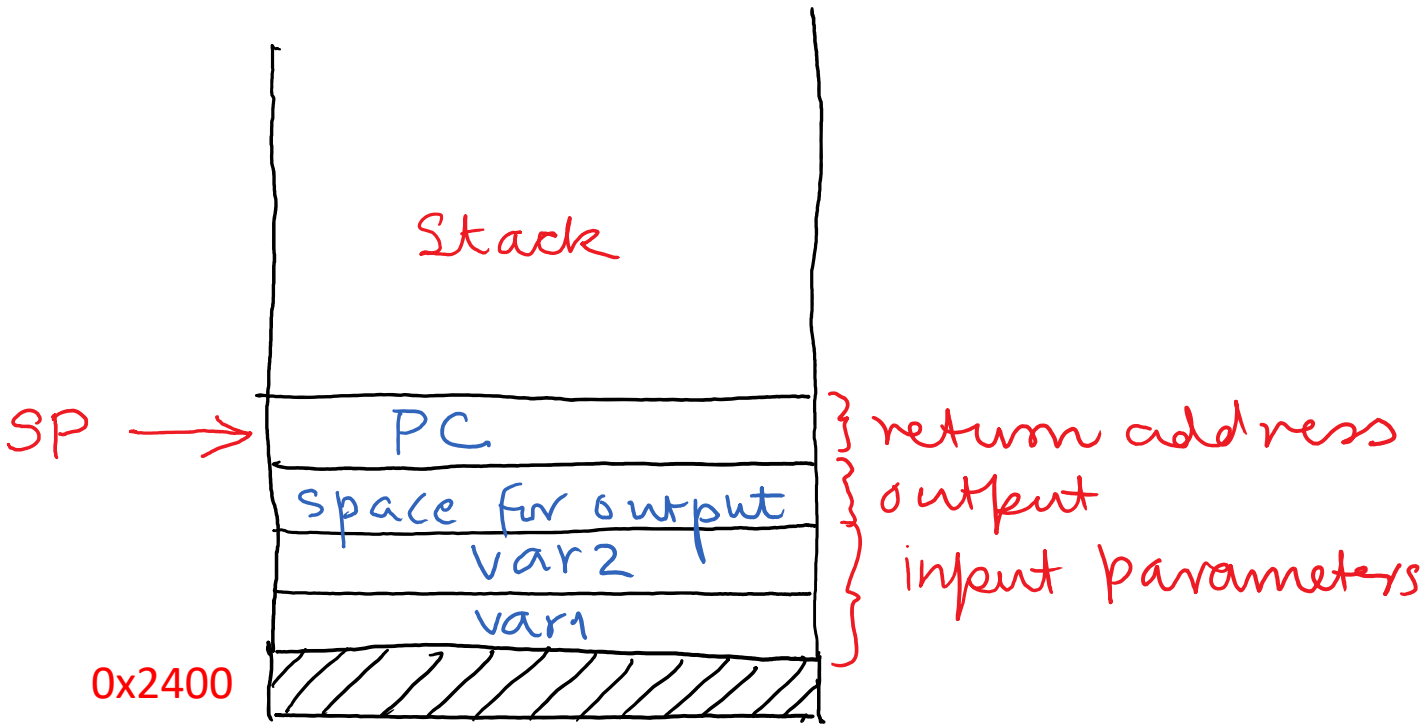
→ space for output



Calling program calls the subroutine

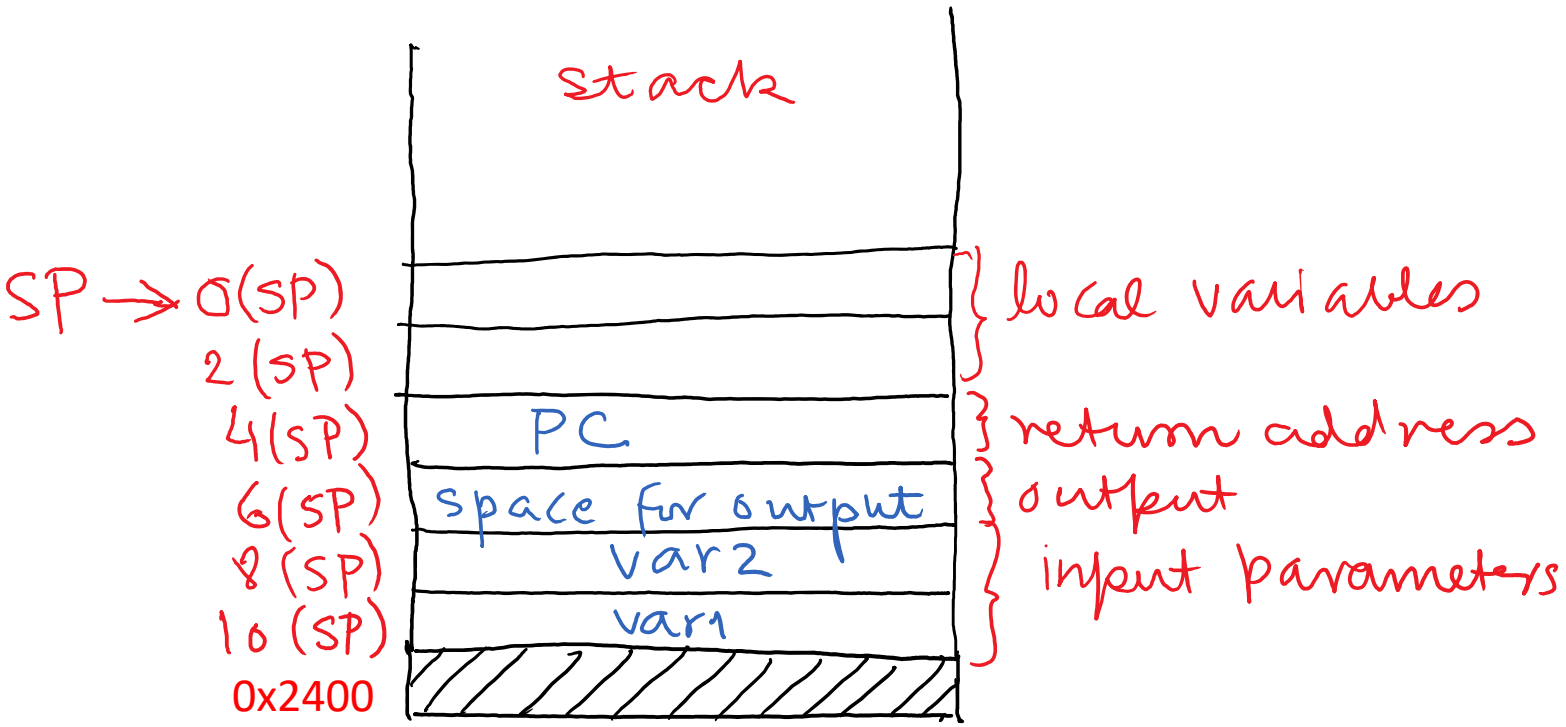
call &mysub

Runtime places PC value on the stack



Subroutine:

Subroutine sets aside space for local variables on the stack.



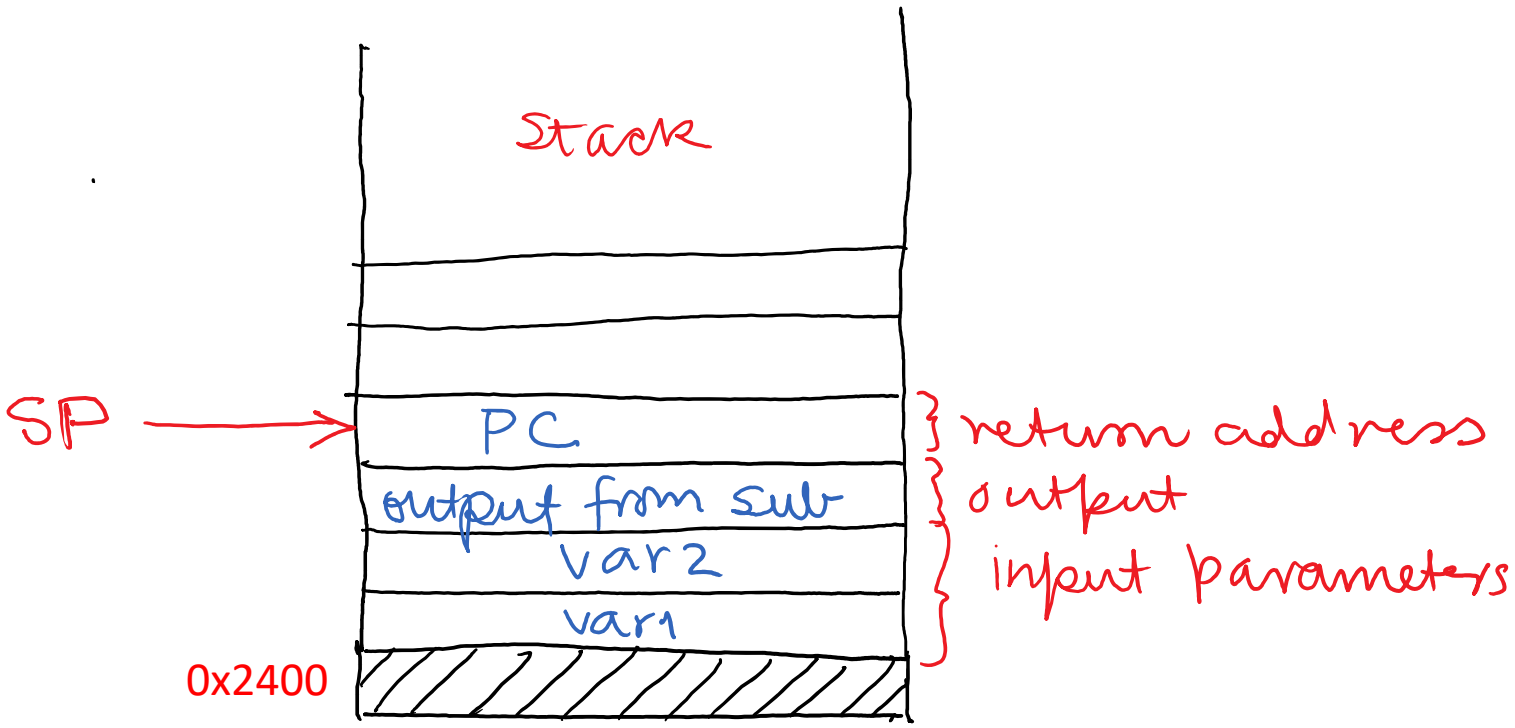
sub.w #4, SP

local variables	0(SP), 2(SP)
input parameters	8(SP), 10(SP)
output	6(SP)

Subroutine uses input parameters local variables and places result in output(s).

before exiting subroutine reclaims space for local variables from the stack

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



ret of subroutine is executed

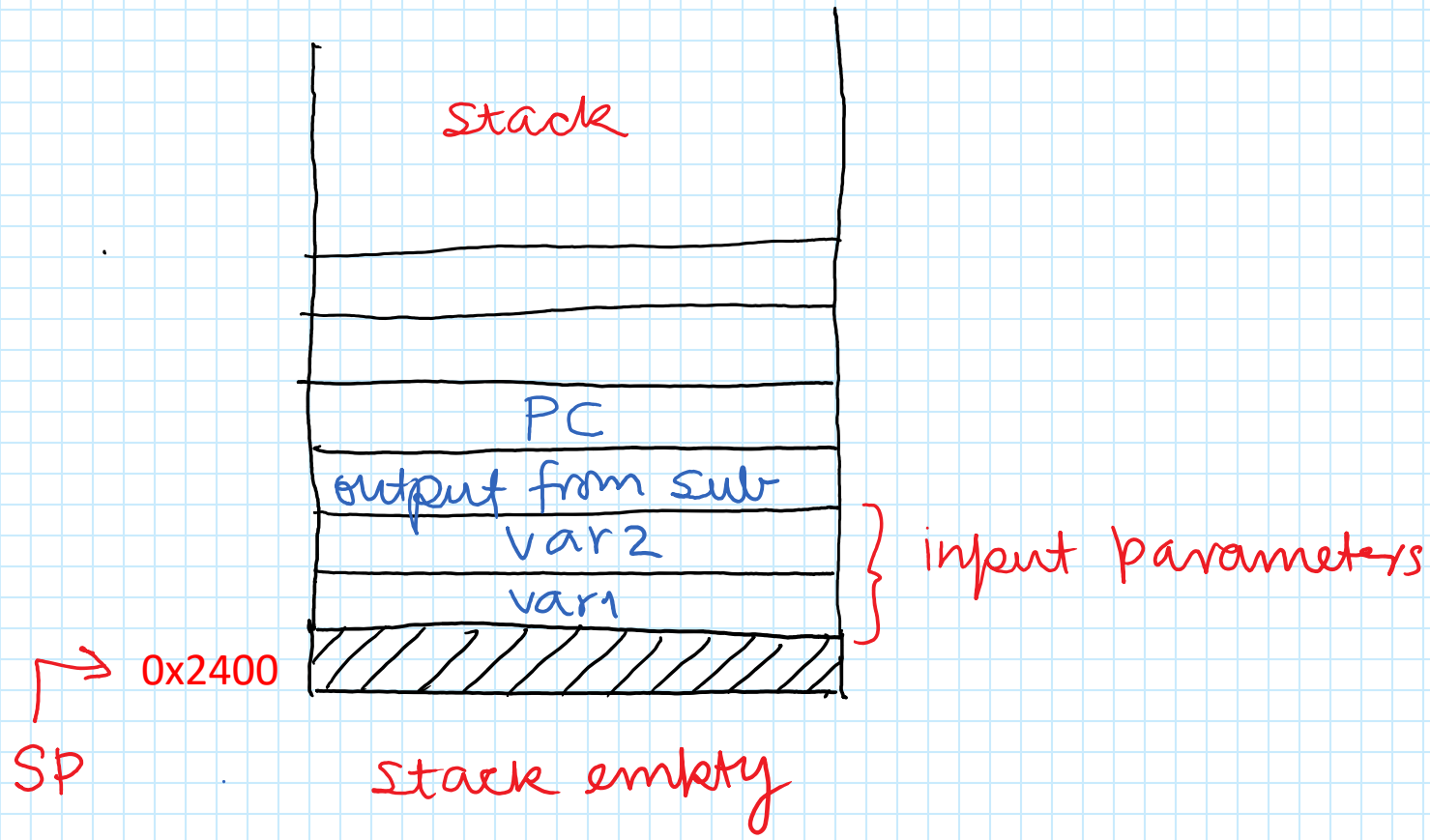
```
ret
```

Calling program pops the output from the stack

```
pop.w  R5
```

Calling program reclaims space for input parameters

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



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 clearly 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  #0000000000000001b, 0(SP)   ; 0(SP) holds bit for testing, start with the lsb
            mov.w       #0,     4(SP)       ; 4(SP) holds sum of set bits in the number

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    #1111000001000011b    ; 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    #1111111111111111b    ; 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
```