Details of function call

```
call #mysub
mov.w R10, &result
```

when the call instruction is executed the address of the next instruction is placed on the stack, the program execution jumps to the label mysub and starts executing the instructions in the subroutine

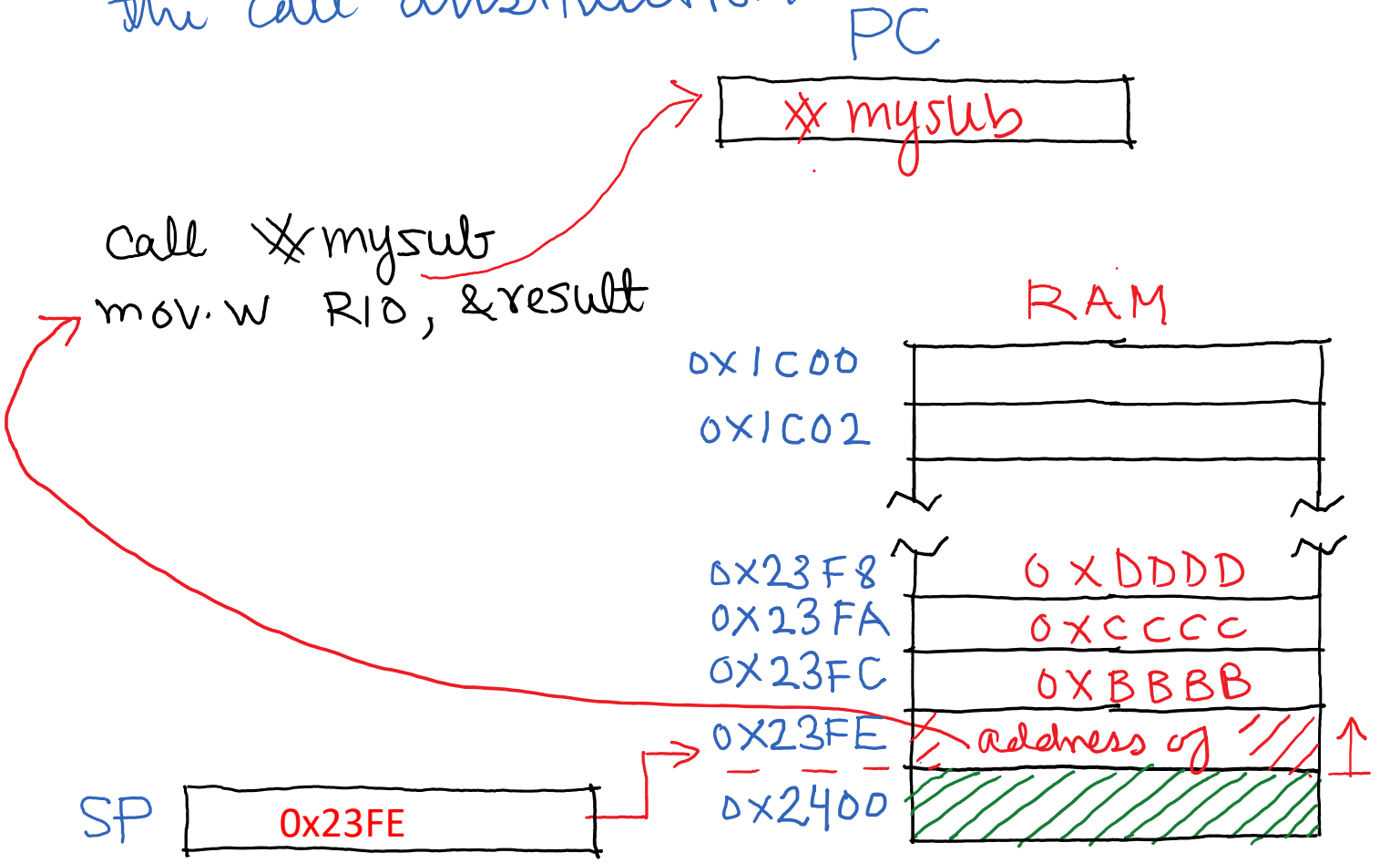
```
mysub:
- - -
- - -
mov.w R11, R10
ret
```

when the program executes the return statement, the stack is popped into the program counter and execution continues from there

the stored "old" PC value should be at the top of the stack

Note: If you push values to the stack inside the subroutine then make sure that you remove all those values from the stack before the ret statement!!!

Stack picture after exicuting the call instruction



Stack contains one word

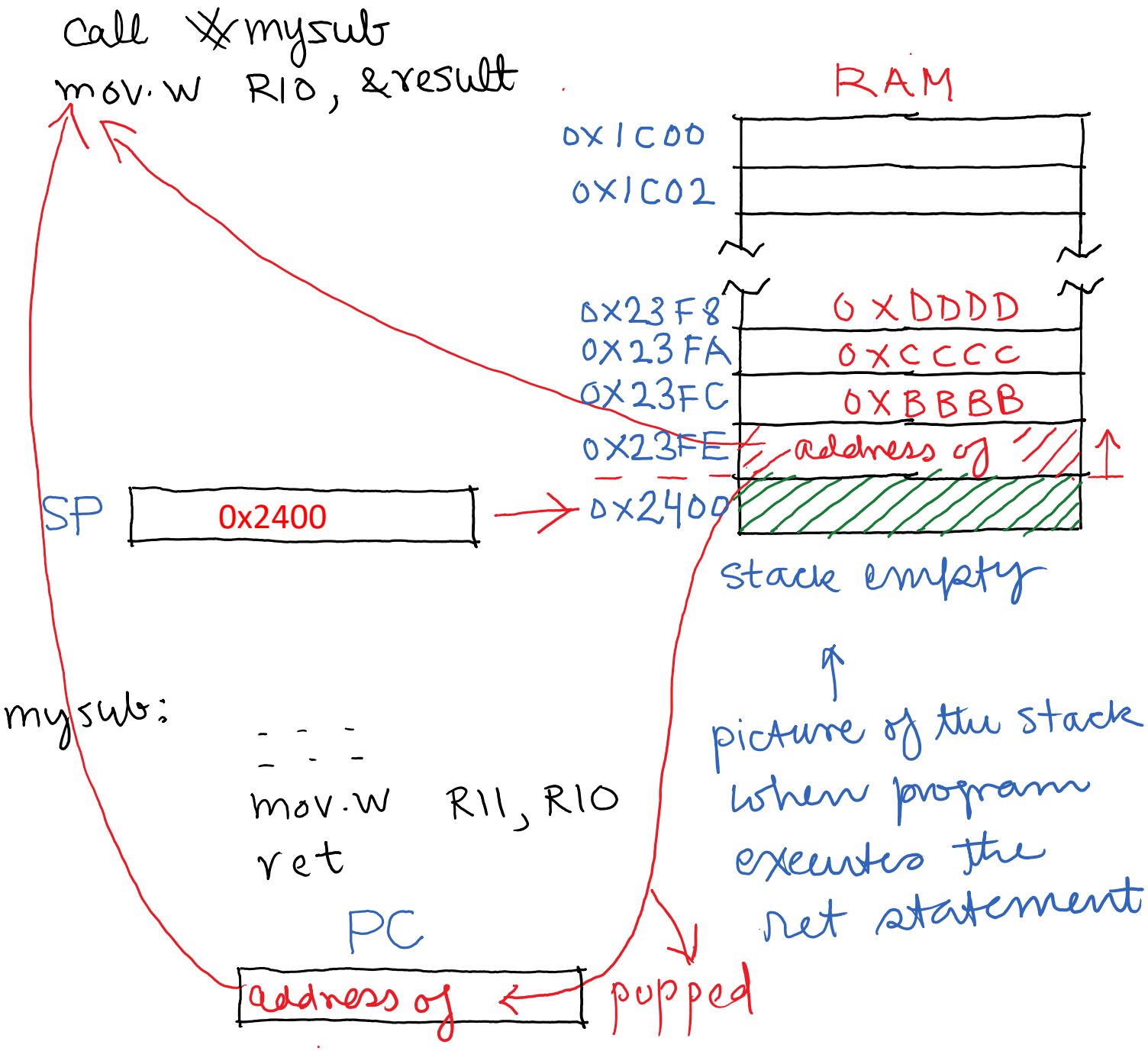
mysub:

- - -
- - -

mov.w R11, R10
ret

picture of the stack when program starts executing the subroutine

picture of the stack after executing the ret instruction



picture of the stack when program executes the ret statement

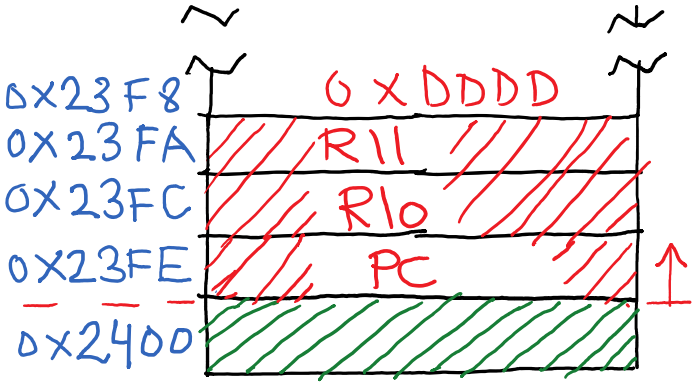
How to preserve local register "variables" inside a subroutine

assume subroutine is going to use R10 and R11 as local register "variables"

At the beginning of the subroutine

```
push.w R10
push.w R11
```

SP → 0x23FA



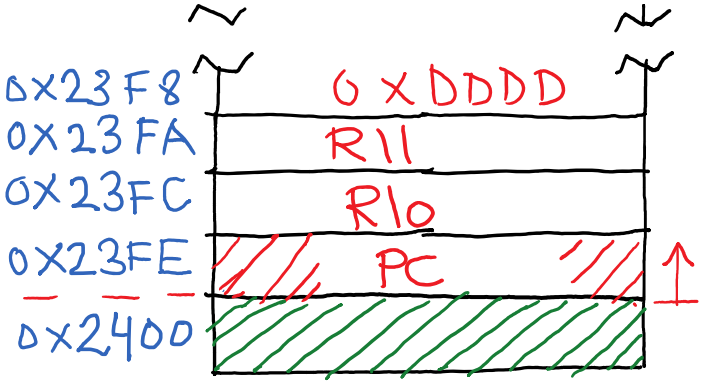
Stack contains three words

use and modify R10, R11

At the end of the subroutine

```
pop.w R11
pop.w R10
```

SP → 0x23FE

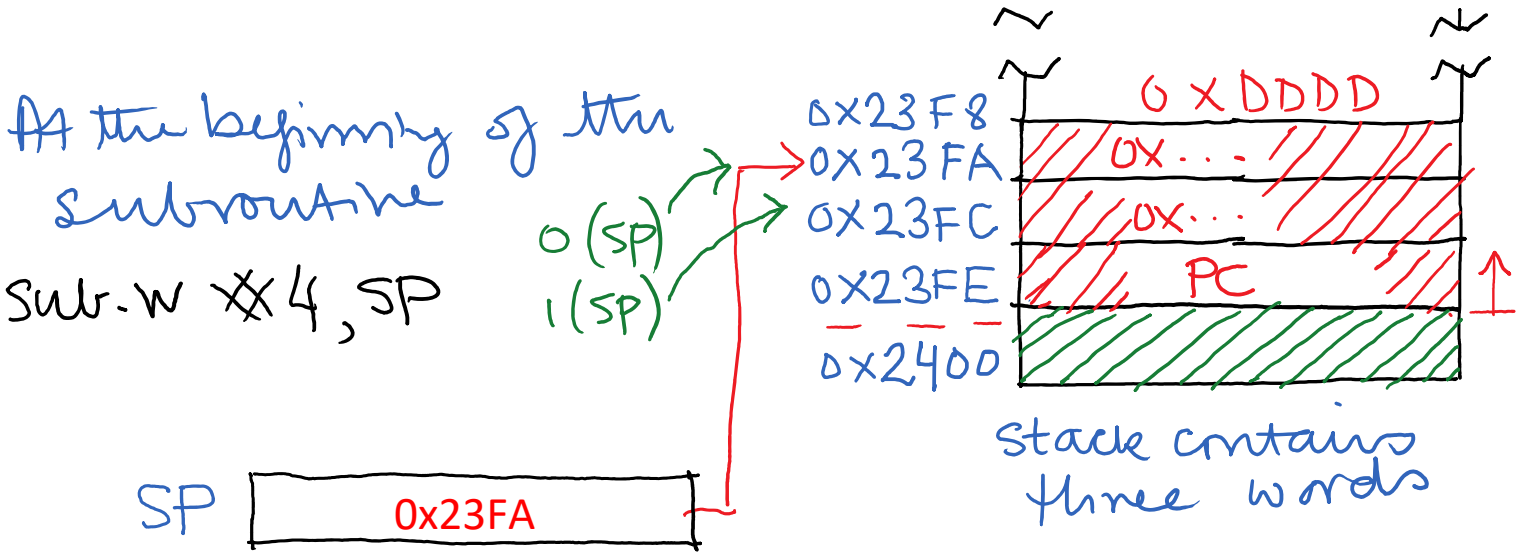


Stack contains one word

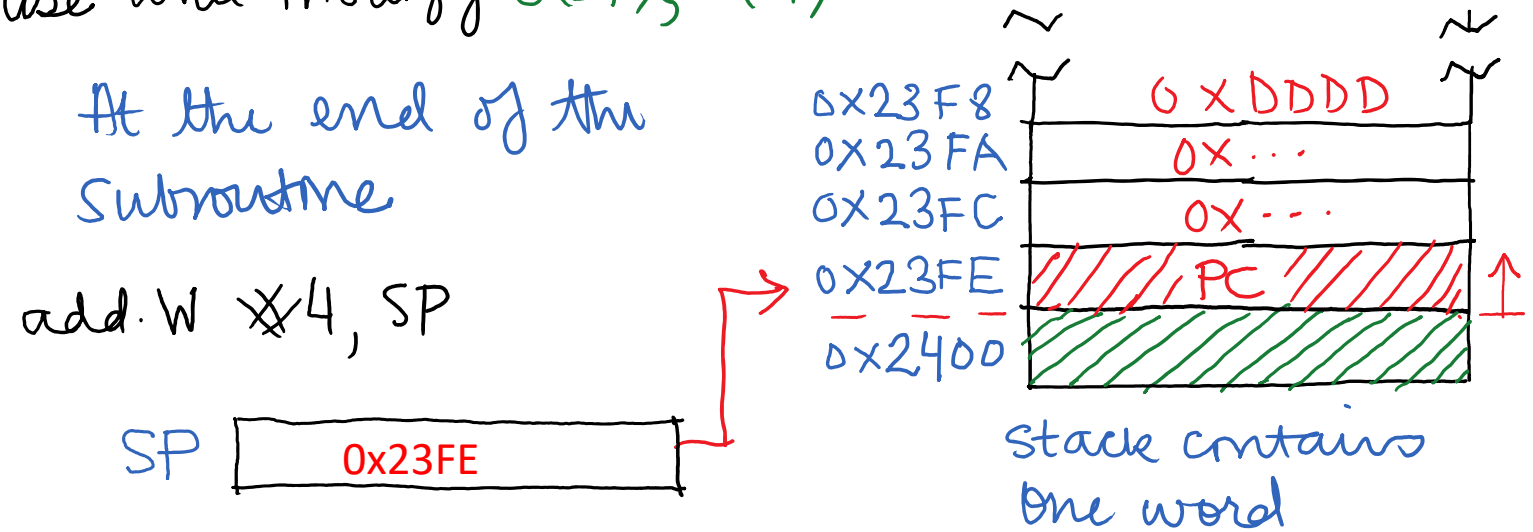
ret pops top of stack into PC

you can use stack based local variables if the execution speed of register variables is not required

assume subroutine is going to use two local variables



use and modify 0(SP), 1(SP)



ret pops top of stack into PC

Example: Restoring local Register

Program: Subroutine 3-2

```
;-------------------------------------------------------------------------
;                          Subroutine: NoOfOnes
;       Counts the number of ones in the binary representation of a number
;       Input R12:    (not modified)
;       Output R11:
;-------------------------------------------------------------------------

NoOfOnes:
            push.w  R10                         ; store R10's value

            mov.w       #0000000000000001b, R10; R10 holds bit for testing, start with the lsb
            mov.w       #0,     R11             ; R11 holds sum of set bits in the
            number

MoreBits:
            bit.w  R10, R12                     ; test the bit
            jnc         BitNotSet               ; if bit is not set then go to label NotSet
            inc.w  R11                  ; if you are here then bit is set, increase sum of bits

BitNotSet:                                      ; if you are here then bit is not set
            rla.w  R10                      ; 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 R11 becomes zero and there are
                                            ; no more bits to test

            pop.wR10                        ; restore R10's value

            ret
```

Example: stack based local var

Program: Subroutine 3-3

```
;-------------------------------------------------------------------------
;                           Subroutine: NoOfOnes
;       Counts the number of ones in the binary representation of a number
;       Input R12:    (not modified)
;       Output R11:
;-------------------------------------------------------------------------

NoOfOnes:
            sub.w #2, SP                    ; set aside space on the stack for one word
                                            ; 0(SP) is our local variable


            mov.w       #0000000000000001b, 0(SP); 0(SP) holds bit for testing, start with the
            lsb
            mov.w       #0,     R11         ; R11 holds sum of set bits in the
            number

MoreBits:
            bit.w  0(SP), R12               ; test the bit
            jnc         BitNotSet           ; if bit is not set then go to label NotSet
            inc.w  R11                      ; 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 R11 becomes zero and there are
                                            ; no more bits to test


            add.w #2, R10                   ; reclaim the stack

            ret
```