Details of function call

Call *mysub
mov.w R10, &result

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

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

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

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 executing the call instruction:

- PC: PC
- Call *mysub
- mov.w R10, &result

**SP**: 0x03FE

**mysub**: mov.w R11, R10
ret

**RAM**:
- 0x0200
- 0x0202
- 0x03FA
- 0x03FC
- 0x03FE
- 0x0400

Stack contains one word

picture of the stack when program starts executing the subroutine
picture of the stack after executing the `ret` instruction

```
call *mymysub
mov.w R10, &result
```

```
mymsub:
    mov.w R11, R10
    ret

PC
address of < popped
```

```
RAM

0x0200
0x0202
0x03F8
0x03FA
0x03FC
0x03FE

stack empty

0xDDDD
0xCCCC
0xBBBB

address of
```

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

SP \(0x03FA\)

Use and modify R10, R11

At the end of the subroutine:
- pop W R11
- pop W R10
- SP

SP \(0x03FE\)

ret pops top of stack into PC

Stack contains three words

\(0x03F8\)
\(0x03FA\)
\(0x03FC\)
\(0x03FE\)
\(0x0400\)

Stack contains one word

\(0x03F8\)
\(0x03FA\)
\(0x03FC\)
\(0x03FE\)
\(0x0400\)

0xDDDD
R11
R10
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.

At the beginning of the subroutine:

Subr. W × 4, SP

SP 0x03FA

At the end of the subroutine:

add. W × 4, SP

SP 0x03FE

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

Stack contains three words

6×DDDD
0x...
0x...
PC

0x03F8
0x03FA
0x03FC
0x03FE
0x0400

Stack contains one word

0x03F8
0x03FA
0x03FC
0x03FE
0x0400

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 #000000000000000000000001b, 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:
    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.w R10                             ; 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
    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:
    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