Details of function call

Call *mysub
mov.w R10, &result

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

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

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 executing the call instruction

Call *mysub

mov.w R10, &result

SP 0x03ff

mysub:

mov.w R11, R10

ret

Stack contains one word

picture of the stack when program starts exactly
the sub-routine
picture of the stack after executing the ret instruction

call *mysub
mov.w R10, &result

SP
0x0400

mysub:
mov.w R11, R10
ret

PC
address of ret popped

RAM
0x200
0x202
0x3f8
0x3fa
0x3fc
0x3ff

0xdddd
0xcccc
0xbbbb

stack empty

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 $0x03FA

Use and modify R10, R11

At the end of the subroutine:
- pop.W R11
- pop.W R10
- SP $0x03FF

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

At the beginning of the subroutine

Sub.W *4, SP

SP 0x03FA

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

At the end of the subroutine

add.W *4, SP

SP 0x03FF

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 #0000000000000000b, 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
    ; 0(SP) is our local variable

    mov.w        #0000000000000001b, 0(SP); 0(SP) holds bit for testing, start with the
    ; 0SB
    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
    ; Go to label L1 if there are more bits to test
    jnz         MoreBits             ; loop is exited when R11 becomes zero and there are
    ; no more bits to test

    add.w       #2, R10             ; reclaim the stack

    ret