```
;
; params.asm
;
; Fall, 2019
; An example to illustrate passing parameters via
; the stack.
; Adapted from Mr. Jason Corless's code
; Modified by Victoria Li

; The stack pointer is in I/O space, so if we
; want to use LDS and STS instructions we have to
; use the alternate addresses.
;
; SPH is 0x3E if using IN/OUT and 0x5E if using LDS/STS
; SPL is 0x3D if using IN/OUT and 0x5D if using LDS/STS
;
;.equ SPH=0x5E
;.equ SPL=0x5D

;
; The Z register is the combination of R31:R30
; since ZH, ZL are defined in "m2560def.inc", we don't need redefine them here.
;.def ZH=r31
;.def ZL=r30
.def temp=r16
.def n1=r0
.def n2=r1
.def sumH=r19
.def sumL=r18

;to read "m2560def.inc", in "Solution Explorer" -> under project name, in this case, "project6"
;-> "Dependencies" ->"m2560def.inc"
;in file "m2560def.inc", SPL/H are defined as following
; ****    .equ  SPL     = 0x3d
; ****    .equ  SPH     = 0x3e
; therefore, must use IN/OUT for read and store

                ; initialize the stack pointer (SP), so that SP points to 0x21FF
                ldi temp, low(RAMEND) ;.equ RAMEND  = 0x21ff <- defined in line 1747 of file "m2560def.inc"
                out SPL, temp
                ldi temp, high(RAMEND)
                out SPH, temp

                ; call the subroutine
                ; Note that it is the caller's responsibility
                ; to push the parameters on the stack before
                ; the call and pop the parameters from the
                ; stack after the call
                ;
                ; push the first parameter
                ldi temp, 0xEE
                push temp
                ; push the second parameter
                ldi temp, 0xCC
                push temp
                call add_num
                ; now that the subroutine has returned
                ; pop the parameters we previously pushed. Why? To restore the stack to the state before the call
                pop temp
                pop temp
                ; At this point, the stack is empty, which
                ; is what we want.

done:           jmp done

; note: add two 8-bit numbers the result may be 9 bits if there is a carry,
; therefore, the sum is store in register pair sumH:sumL - r19:r18
; add_num sumH:sumL=n1+n2  0x1BA=0xEE + 0xCC
;
; This subroutine demonstrates using the stack to pass
; parameters and using the stack to 'protect' registers
; that are used in the subroutine.
;
; By protecting registers this subroutine uses, the callers
; are free to use any registers.
;
;
; After the Z register is set to be the stack pointer, the
; stack frame looks like:
;
; |      | <- Z and SP
; | n2   |  saved register <- register n2 (r1) is going to be used in the subroutine, preserve its value on stack
; | n1   |  saved register <- register n1 (r0) is going to be used in the subroutine, preserve its value on stack
; | ZH   |  saved register
; | ZL   |  saved register
; | ret  |  return address
; | ret  |  return address
; | ret  |  return address
; | 0xCC |  parameter n2 (Z + 8)
; | 0xEE |  parameter n1 (Z + 9)
;
add_num:
                ; first protect the Z register, since we will use it
                push ZL
                push ZH
                ; now protect r0 and r1 since they will be used
                ; to store the parameters
                push n1 ;r0
                push n2 ;r1

                ; load the value in stack pointer into the Z register
                in ZH, SPH
                in ZL, SPL

                ; get the 1st parameter pushed on the stack:
                ldd n1, Z+9
                ; get the 2nd parameter pushed on the stack:
                ldd n2, Z+8

                ; sumH:sumL=n1+n2
                clr sumH
                mov sumL, n1
                add sumL, n2
                rol sumH ;carry bit is brought into position 0 of sumH

add_num_end: ; This is where we return from the subroutine
                ; restore the registers protected on entry
                ; into the subroutine
                pop n2
                pop n1
                pop ZH
                pop ZL
                ret
```