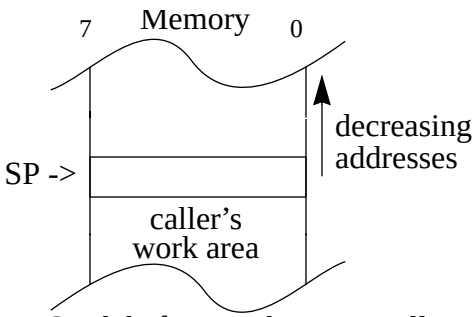
Building a stack frame:

Working Example: Calling a subroutine with two parameters and a return value, and some local variables. The call looks like

```
retval = mysub(parm1, parm2);
```

Assume parm1 is two bytes, parm2 and retval are one byte each.



Stack before a subroutine call
(from within another block,
the "caller")

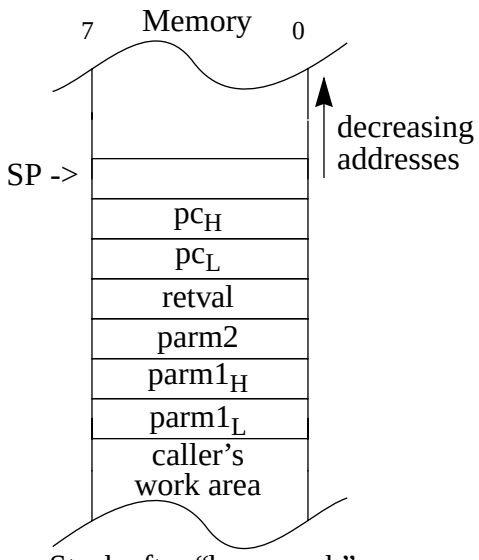
Assembly code to prepare for the call:

```
; calculate value for parm1,
; then push it onto the stack
      ...         ; put parm1 in "D"
      pshb
      psha

; calculate parm2, and push it
      ...         ; put parm2 in "A"
      psha

; leave a byte for the return
; value
      des

; and make the subroutine call
      bsr   mysub
```

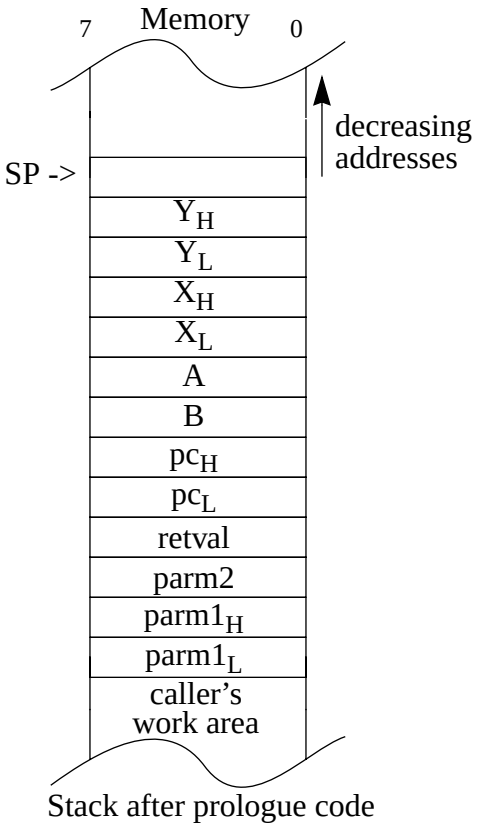


Stack after "bsr mysub"

Building a stack frame - 2

Start of assembly code for the subroutine "mysub"

```
; standard subroutine prologue
; code to save registers
mysub:      pshb
            psha
            pshx
            pshy
```

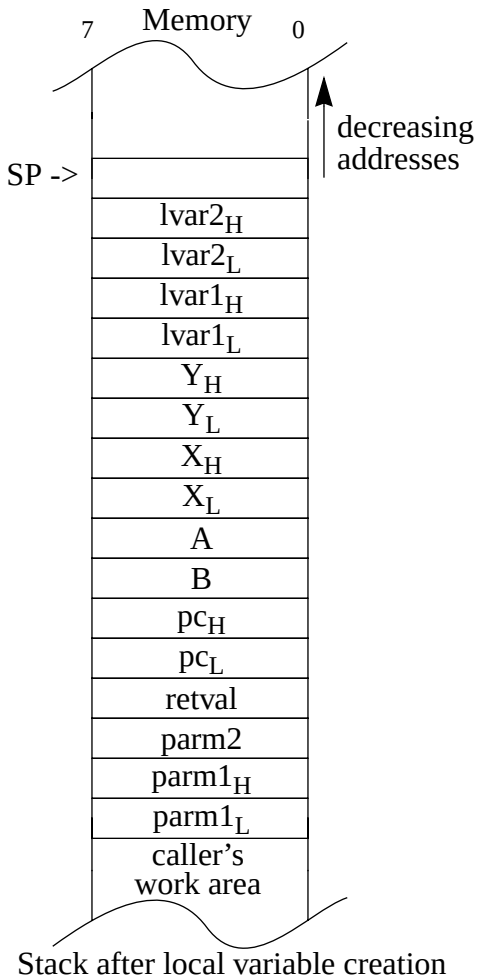


Stack after prologue code

Assume that in this subroutine, there are two local variables, lvar1 and lvar2, both to occupy 2 bytes. lvar1 will be initialized to the value "42" and lvar2 will be uninitialized when defined.

Continuation of assembly code for subroutine "mysub", including variable initialization:

```
; initialize lvar1 to 42
            ldx   #42
            pshx
; set aside 2 bytes for lvar2. Easiest
; way to do this is to push x again
            pshx
```



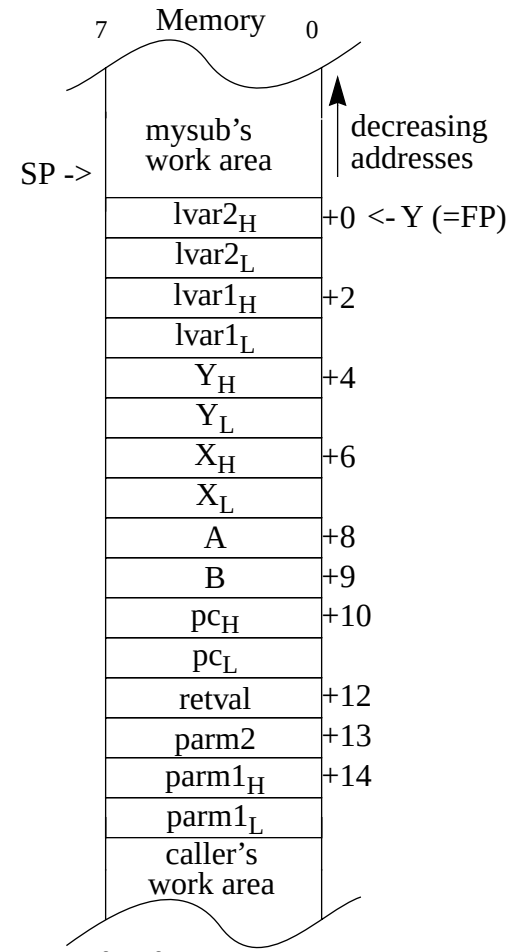
Stack after local variable creation

Building a stack frame - 3

A frame pointer is a memory address that allows us to access all of these parameters and variables without messing with the stack pointer. The HC11 does not have a dedicated register to store a frame pointer. For this example, let's store the frame pointer in "Y".

Continuation of assembly code for subroutine "mysub", including frame pointer initialization:

```
; initialize Y to be the frame pointer
      tsy
```



Stack after frame pointer initialization showing frame pointer and offsets

When we're done with the subroutine, we'll want to discard the local variables and restore the register values.

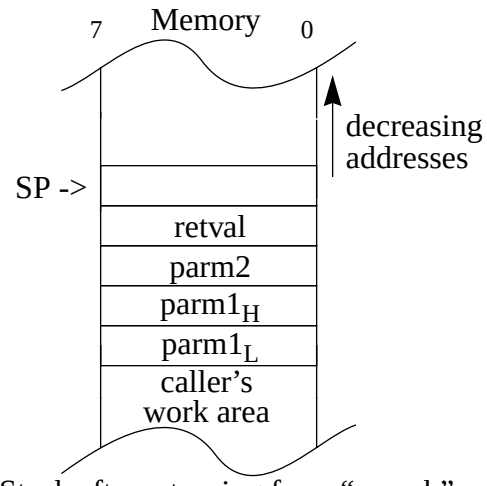
Continuation of assembly code for subroutine "mysub", after the subroutine execution

```
; strip the local variables and
; restore the registers. we don't care
; about the values of the parameters,
; so just do two 2-byte pops.

cleanup:    pulx
            pulx

; restore the registers and return

            puly
            pulx
            pula
            pulb
            rts
```



Stack after returning from "mysub"