Indirect Addressing Example

Pseudocode to add a column of numbers:

```
sum = 0;
for (i = count-1 ; i>=0 ; i--)
sum = sum + column[i];
```

Beginning of Assembly Language Code using Direct Addressing:

```
; Code fragment to add a column of numbers:
; count contains the number of numbers to be added
; A will hold the sum
; X will hold the iteration count
; initialize the sum (A) to 0, the iterator (X) to count-1

        ldx  count      ; load the count into X
        dex             ; X = count - 1
        clra            ; clear A (initialize sum to 0)

; begin a loop to add the numbers.
; fetch a number from memory and add it to the sum

loop:   ldab column???
```

Indirect Addressing Example (continued)

Listing file of HC11 code to add a column of numbers using indrect addressing

```
INDIR.ASM               Assembled with CASMW  6/22/00  1:50:21 PM  PAGE 1

                         1  ; Code fragment to add a column of numbers:
                         2
                         3  ; count contains the number of numbers to be added
                         4  ; A will hold the sum.
                         5  ; X will hold the iteration count.
                         6  ; Y will hold the address of the next column entry.
                         7  ; initialize the sum (A) to 0, the iterator (X) to
                         8  ; count, and the Y register to the address of the
                         9  ; first item in the column.
                        10
 0000 [05] FE001B       11          ldx     count   ; load the count into X
 0003 [02] 4F           12          clra            ; initialize sum to 0
 0004 [04] 18CE001D     13          ldy     #column ; y <- address of column[1]
                        14
                        15  ; begin a loop to add the numbers.
                        16  ; fetch a number from memory and add it to the sum
                        17
 0008 [05] 18E600       18  loop:   ldab    0,y     ; B <- column[i]
 000B [02] 1B           19          aba             ; Add this to sum
                        20
                        21  ; now check if we're done
                        22
 000C [03] 09           23          dex             ; X = X - 1
 000D [03] 2704         24          beq     done    ; if X=0, we're done
                        25
                        26  ; if we're not done, increment Y and continue
                        27
 000F [04] 1808         28          iny             ; Y<-address of column[i+1]
 0011 [03] 20F5         29          bra     loop    ; jump back to loop start
                        30
                        31  ; we're done, so store the sum in memory, then halt.
                        32
 0013 [04] B70024       33  done:   staa    sum
 0016 [02] 07           34          tpa
 0017 [02] 847F         35          anda    #$7f
 0019 [02] 06           36          tap
 001A [02] CF           37          stop
```

Listing file of HC11 code to add a column of numbers using indrect addressing (continued, showing the data area including the column to add)

```
                        38
                        39  ; and here's the data area
                        40
 001B       0007        41  count:  dw      7
 001D       01          42  column: db      $1
 001E       02          43          db      $2
 001F       03          44          db      $3
 0020       05          45          db      $5
 0021       07          46          db      $7
 0022       0B          47          db      $b
 0023       0D          48          db      $d
 0024       00          49  sum:    db      0
                        50
                        51  ; and set up the reset vector
                        52
 FFFE                   53          org     $fffe
 FFFE       0000        54  reset:  dw      $0000
                        55

 Symbol Table

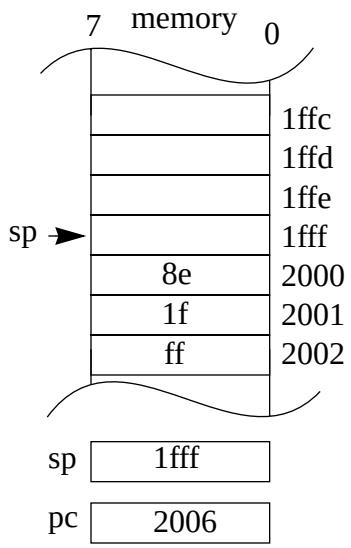
COLUMN              001D
COUNT               001B
DONE                0013
LOOP                0008
RESET               FFFE
SUM                 0024
```

Stack example, showing jumps to and returns from nested subroutines

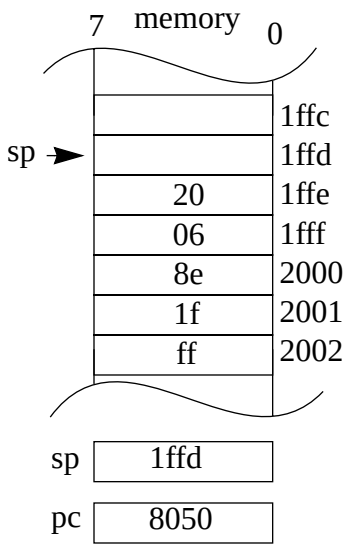
```
STACK.ASM                Assembled with CASMW  6/22/00  1:18:21 PM  PAGE 1
                          1  ; this chunk of code has a main program which calls
                          2  ; a subroutine, and that subroutine calls another
                          3  ; subroutine.  This code demonstrates the use of the
                          4  ; stack for storing return addresses of subroutines,
                          5  ; and the use of the JSR and RTS instructions.
                          6
                          7  ; initialize SP so the stack will start at address
                          8  ; 1fffh and grow downward toward 0000h
                          9  ; Start the program at 2000h, so there's no overlap
                         10
 2000                    11         org    $2000     ; start the program at 2000h
 2000 [03] 8E1FFF        12  main:  lds    #$1fff    ; initialize stack to 1fffh
                         13
                         14  ; call the first of the dummy subroutines, which
                         15  ; will call the other one
                         16
 2003 [06] BD8050        17         jsr    subr1
                         18
                         19  ; then we're done: set up the stop bit and stop
                         20
 2006 [02] 07            21         tpa
 2007 [02] 847F          22         anda   #$7f
 2009 [02] 06            23         tap
 200A [02] CF            24         stop
                         25
                         26  ; here's the first of our dummy subroutines. we'll
                         27  ; put our subroutines somewhere else in memory.
                         28  ; subr1 will do nothing useful except call subr2
                         29
 8050                    30         org    $8050     ; subroutines begin here
 8050 [02] 8600          31  subr1: ldaa   #$00      ; clear A (something to do)
 8052 [06] BD8058        32         jsr    subr2     ; call the next dummy subrtn
 8055 [02] C600          33         ldab   #$00      ; clear B (something to do)
 8057 [05] 39            34         rts              ; return from subr1 to main
                         35
                         36  ; here's the second of our dummy subroutines. it
                         37  ; will do nothing useful except return
                         38
 8058 [02] 8600          39  subr2: ldaa   #$00      ; clear A (something to do)
 805A [05] 39            40         rts              ; return from subr1 to main
                         41
                         42  ; set up the reset vector
 FFFE                    43         org    $fffe
 FFFE      2000          44  reset: dw     $2000
```

Stack, SP, PC frames for the STACK.ASM example

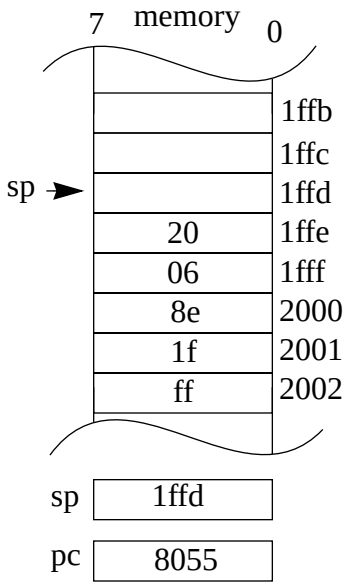
after fetch of jsr subr1



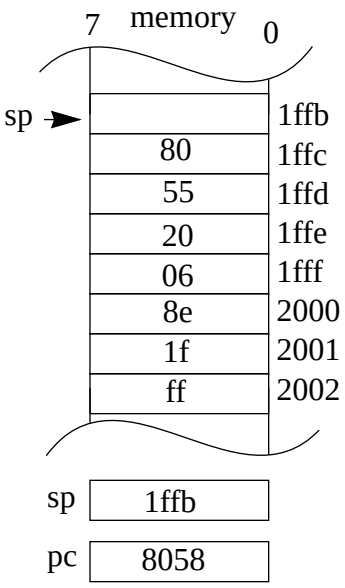
after execution of jsr subr1



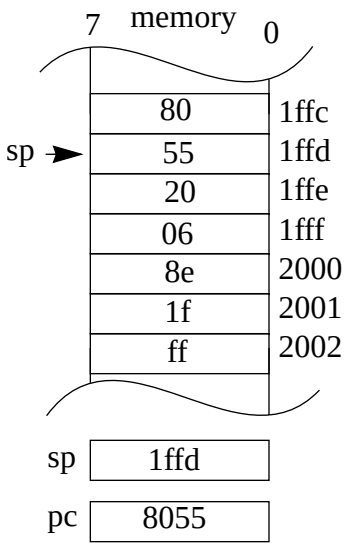
after fetch of jsr subr2



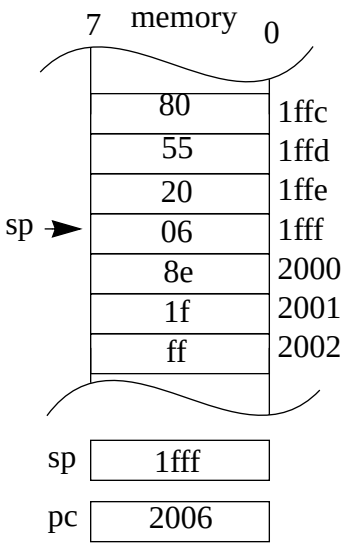
after execution of jsr subr2



after execution of rts
(from subr2)



after execution of rts
(from subr1)



More stack stuff: template for subroutine register saving

```
; standard template code for subroutine prologue and epilogue.
; save all the CPU registers so we can use them freely, then
; restore them at the end so the caller doesn't notice that
; they've been used.

; note that since the stack is organized last-in-first-out,
; we pop (pull, on the HC11) the registers from the stack in
; the opposite order from which we pushed them.

my_sub: pshb
        psha
        pshx
        pshy

; useful code goes here
...
; end of useful code

        puly
        pulx
        pula
        pulb
        rts
```