ECET 310-001
Chapter 4

W. Barnes, 9/2006, rev'd. 11/07

Ref. Huang, Han-Way, *The HCS12/9S12: An Introduction to Software and Hardware Interfacing,* Thomson/Delmar.

In This Set of Slides:

- Data Structures (Stack, Arrays, Strings)
- Search of Sorted and Unsorted arrays
- Strings
- Subroutines
  - Usage Rules
  - Stack
  - Leas
  - Stack Frame
- Bubble Sort Example
- D-Bug12 I/O Functions
  - Printf function

Program = data structures + algorithm

Three Data structures to be discussed

1. Stack: a last-in-first-out data structure
2. Array: a set of elements of the same type
3. String: a sequence of characters terminated by a special character

- **Stack:**

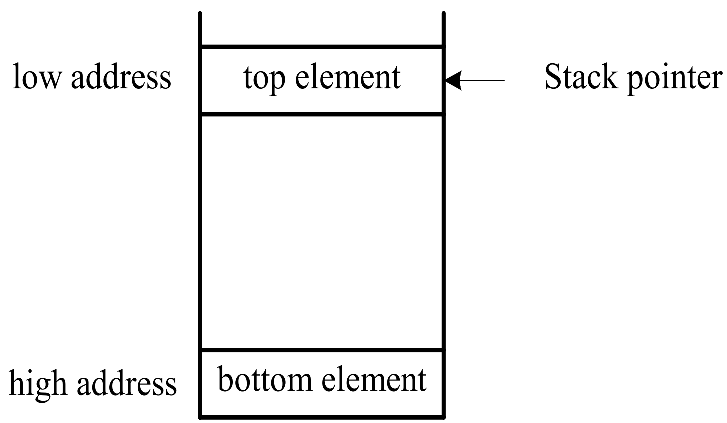

Figure 4.1 Diagram of the HCS12 stack

Stack cont'd, Push and Pull Instructions

- **The stack grows down in memory**
- ***pushes* pre-decrement while *pulls* post-increment.**
- **Note the equivalent instructions, which help explain what's happening**
- **CCR push and pull have no equivalent instructions, so CCR can only be accessed via the stack**

Table 4.1 HCS12 push and pull instructions and their equivalent load and store instructions

Mnemonic	Function	Equivalent instruction
psha	push A into the stack	staa 1, -SP
pshb	push B into the stack	stab 1, -SP
pshc	push CCR into the stack	none
pshd	push D into stack	std 2, -SP
pshx	push X into the stack	stx 2, -SP
pshy	push Y into the stack	sty 2, -SP
pula	pull A from the stack	ldaa 1, SP+
pulb	pull B from the stack	ldab 1, SP+
pulc	pull CCR from the stack	none
puld	pull D from the stack	ldd 2, SP+
pulx	pull X from the stack	ldx 2, SP+
puly	pull Y from the stack	ldy 2, SP+

Indexable Data Structures

- Vectors (one dimension) and matrices (multi-dimensioned) are indexable data structures.
- First element of a vector is associated with the index 0 to facilitate the address calculation.
- Directives **db**, **dc.b**, **fcb** define arrays of 8-bit elements.
- Directives **dw**, **dc.w**, and **fdb** define arrays of 16-bit elements.

Example 4.2

Write a program to find out if the array vec_x contains a value, ***key***.
The array has 16-bit elements and is <u>not sorted</u>.

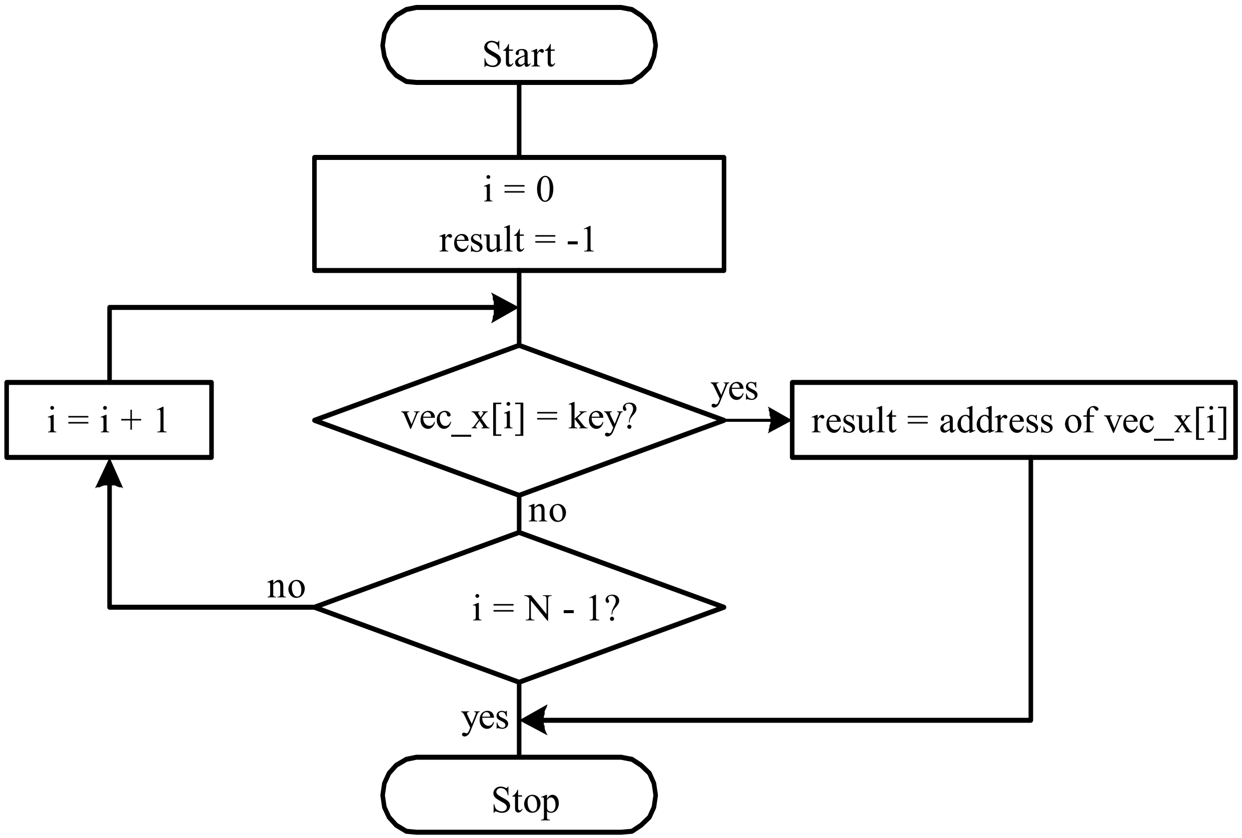

Figure 4.3 Flowchart for sequential search

Code for Search of an Unsorted Array

```
; looks for 16-bit key and if found stores the address at result, otherwise –1 is stored at result
; this program contains a true do until C loop
N           equ     15          ; array count
notfound    equ     -1
key         equ     190         ; define the searching key

            org     $1500
result      rmw     1           ; reserve a word for result

            org     $2000
            ldy     #N          ; set up loop count
            ldd     #notfound
            std     result      ; initialize the search result with default of notfound, -1 or $FFFF
            ldd     #key
            ldx     #vec_x      ; place the starting address of vec_x in X
loop        cpd     2,X+        ; compare the key with array element & update pointer
            beq     found
            dbne    Y,loop      ; have we gone through the whole array?
            bra     done        ; only get to here if key is not found
found       dex                 ; need to restore the value of X to point to the
            dex                 ; matched element
            stx     result
done        swi
vec_x       dw      13,15,320,980,42,86,130,319,430,4, 90,20,18,55,30
            end
```

Q. What will be the value in result after above has executed?

Binary Search of a Sorted Array

(Takes advantage of the fact array is sorted to increase efficiency/decrease execution time)

<u>Algorithm:</u> Compare key with middle element, if equal then done, if key>middle element then continue search in upper half of array, if key<middle element then continue search n lower half of array

<u>For following: max,min,mean are pointers, not actual data</u>

- **Step 1: Initialize variables max and min to n -1 and 0, respectively.**
- **Step 2: If max < min, then stop since no element matches the key.**
- **Step 3: Let mean = (max + min)/2**
- **Step 4: If key = arr[mean], then key is found in the array, exit.**
- **Step 5: If key < arr[mean], then set max to mean - 1 and go to step 2.**
- **Step 6: If key > arr[mean], then set min to mean + 1 and go to step 2.**

Example 4.3 Write a program to implement the binary search algorithm for a sorted array and also a sequence of instructions to test it.
(longer program than last but more efficient if you have a sorted array)

```
n       equ     15          ; array count
key     equ     83          ; key to be searched

        org     $1500
max     rmb     1           ; maximum index value for comparison
min     rmb     1           ; minimum index value for comparison
mean    rmb     1           ; the average of max and min
result  rmb     1           ; search result

        org     $2000
        clra
        staa    min         ; initialize min to 0 (i.e., point to first number in array)
        staa    result      ; initialize result to 0
        ldaa    #n-1
        staa    max         ; initialize max to n-1 (i.e., point to last number in array)
        ldx     #arr        ; use X as the pointer to the array
loop    ldab    min
        cmpb    max
        lbhi    notfound    ;Long Branch to notfound if min > max
        addb    max         ; compute mean
        lsrb                ;           "       (max + min)/2

;Continued on next slide
```

Binary Search continued

```
            stab    mean        ; save mean
            ldaa    b,x         ; A ← element arr[mean] uses B, mean, as offset
            cmpa    #key
            beq     found
            bhi     search_lo
            ldaa    mean
            inca
            staa    min         ; place mean+1 in min to continue
            bra     loop
search_lo   ldaa    mean
            deca
            staa    max
            bra     loop
found       ldaa    #1
            staa    result
notfound    swi
arr         db      1,3,6,9,11
            db      61,63,64,65,67
            db      80,83,85,88,90
            end
```

Strings

- **String def**.: A sequence of characters terminated by a NULL (ASCII code 0) or other special character such as EOT (ASCII code 4).

- To be understood, a binary number must be converted to ASCII

- Conversion method: divide the binary number by 10 repeatedly until the quotient is zero. $30 is added to each remainder.

- **Example 4.4** Write a program to convert the unsigned 8-bit binary number in accumulator A into BCD digits terminated by a NULL character. Each digit is represented in ASCII code.

Solution:

  - For 8 bits, the largest number would be 255, thus 4 bytes, *including* the null, are needed to hold the converted BCD digits.
  - Repeated division by 10 method is used.
  - See program on next page.

```
test_dat    equ     34
            org     $1000
buf         db      4           ; to hold the decimal string
temp        db      2           ;       "
            org     $2000
            lds     #$2000      ; initialize SP (recall stack goes down in memory)
            ldab    #test_dat
            ldy     #buf        ;use Y to point to decimal string
            tstb
            bne     normal
            movb    #$30,buf    ;store ascii 0 (30) but get here only if test_dat = 0
            clr     buf+1       ; terminate the string with an actual, not ascii, zero
            bra     done
normal      movb    #0,1,-sp    ; store the NULL delimiter in the stack
            clra
loop        ldx     #10
            idiv
            addb    #$30        ; convert to ASCII code (rem in D but no bigger than B)
            pshb                ; push into stack
            cpx     #0          ; get out of loop when quotient is finally 0
            beq     reverse     ;  "
            xgdx                ; otherwise, put quotient back in B for next division
            bra     loop
reverse     tst     0,sp                ;move numbers in reverse order into buf
            beq     done                ;done when NULL byte reached
            movb    1,sp+,1,y+
            bra     reverse
done        swi
            end
```

Example 4.6: Convert an ASCII String Representing a BCD Number Into a Signed Binary Number

- **Algorithm**

Step 1

sign ← 0
error ← 0
number ← 0

Step 2

If the character pointed to by **in_ptr** is the minus sign, then
sign ← 1
in_ptr ← in_ptr + 1

Step 3

If the character pointed to by in_ptr is the NULL character,
then go to step 4.
else if the character is not a BCD digit, then
error ← 1; go to step 4;
else
number ← number * 10 + m[in_ptr] - $30;
in_ptr ← in_ptr + 1;
go to step 3;

Step 4

If sign = 1 and error = 0 ,then
number ← two's complement of number;
else
stop;

See program on next slide

ASCII String to Signed Binary

```
minus       equ     $2D         ; ASCII code of minus sign
            org     $1000
in_buf      fcc     "9889"      ; input ASCII to be converted
            dB      0           ; null character to terminate ASCII
out_buf     db      2           ; holds the converted binary value
buf2        db      1           ; holds a zero
buf1        db      1           ; holds the current digit value
sign        db      1           ; holds the sign of the number
error       db      1           ; indicates the occurrence of illegal character
            org     $1500
            clr     sign
            clr     error
            clr     out_buf
            clr     out_buf+1
            clr     buf2
            ldx     #in_buf
            ldaa    0,x
            cmpa    #minus      ; is the first character a minus sign?
            bne     continue    ; branch if not minus
            inc     sign        ; set the sign to 1
            inx                 ; move the pointer
continue    ldaa    1,x+        ; is the current character a NULL character?
            lbeq    done        ; yes, we reach the end of the string
            cmpa    #$30        ; is the character not between 0 to 9?
```

=========CONTINUED ON NEXT PAGE==========

ASCII String to Signed Binary Cont'd.

```
                lblo    in_error    ; get out if number not valid, below 0
                cmpa    #$39        ; "
                lbhi    in_error    ; get out if number not valid, above 9
                suba    #$30        ; convert to the BCD digit value
                staa    buf1        ; save the digit temporarily
                ldd     out_buf
                ldy     #10
                emul                ; Y:D ← D * Y
                addd    buf2        ; add the current digit value
                std     out_buf     ; Y holds 0 and should be ignored
                bra     continue
in_error        ldaa    #1
                staa    error
done            ldaa    sign        ; check to see if the original number is negative
                beq     positive
                ldaa    out_buf     ; if negative, compute its two's complement
                ldab    out_buf+1   ;           "
                coma                ;           "
                comb                ;           "
addd            #1      ;           "
                std     out_buf
positive        swi
                end
```

Subroutines

- A sequence of instructions called from various places in the program
- Allows the same operation to be performed with different parameters
- Simplifies the design of complex program by using ‘divide and conquer’

- Instructions related to subroutine calls:
  - **bsr** <rel> ; branch to subroutine
  - **jsr** <opr> ; jump to subroutine
  - **rts** ; return from subroutine

  - **call** <opr> ; used for expanded memory
  - **rtc** ; return from subroutine

Program Structure w/Subroutines

Notes:

1. Main will call various subroutines but also a subroutine can call another, for example subroutine 2.1 could call 3.1

2. A subroutine calling itself is 'recursion' but you've got to know what you are doing!

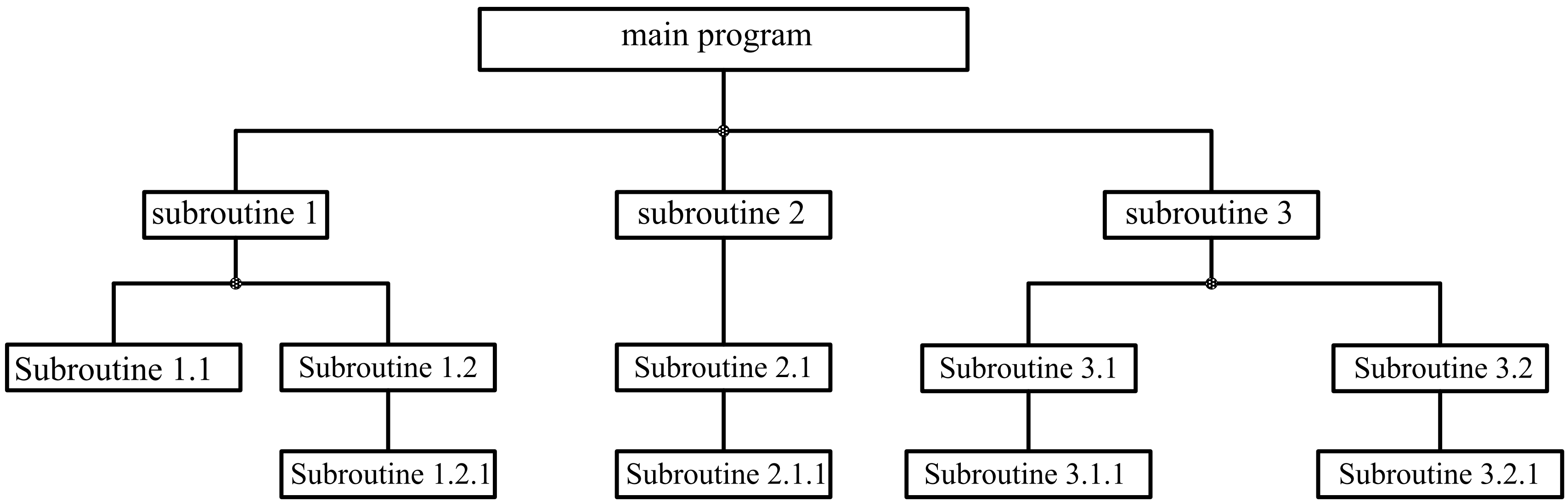

Figure 4.7 A structured program

General Subroutine Processing

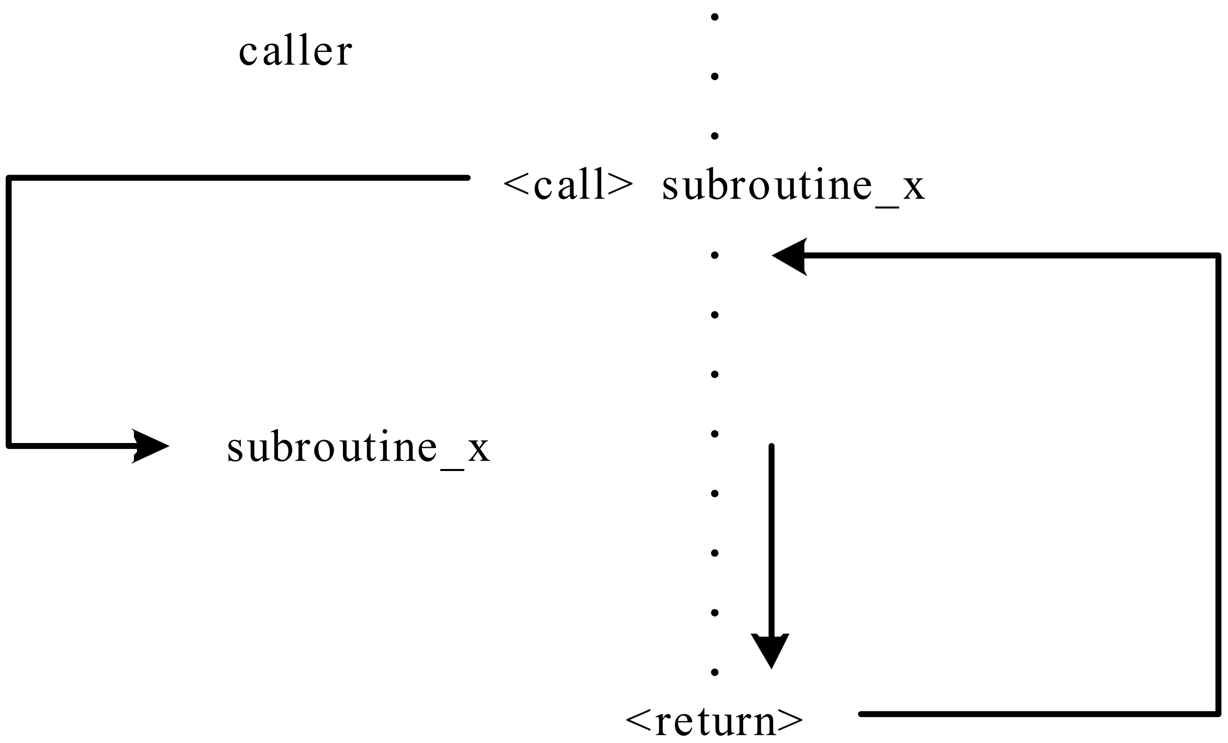

Figure4.8 Program flow during a subroutine call

Important Subroutine Issues

- **Keep subroutines independent/portable**
  - Do not use direct or extended addressing
  - Keep in mind the subroutine may be called from numerous locations including other subroutines
- **Know how a subroutine affects registers or make sure that it doesn't**
  - Comments should be used at beginning of routine to aid in writing the caller
  - If needed, push registers on stack at beginning of a subroutine and pull them just before *rts* or *rtc*
- **Parameter passing, to or from subroutine**
  - By registers: send/receive actual data and/or address pointers
  - By stack: send/receive actual data and/or address pointers via the stack but make sure SP points to the return address when rts is executed

Two Ways of Preserving Registers

Discuss: advantages and disadvantages of each

1. Incorporate saving in subx:

```
        bsr  subx
.
.
        bsr subx
.
        swi

subx    psha    ;saving a and x
        pshx
.
.
        pulx    restoring a and x
        pula
        rts

        end
```

2. Incorporate saving in main:

```
        psha
        pshx
        bsr     subx
        pulx
        pula
.
.
        psha
        pshx
        bsr     subx
        pulx
        pula
        swi
subx
.
.
        rts
        end
```

In Class Exercise Regarding the Stack

List file is on next slide

Show stack values as program is executed

What would happen if a *psha* were placed between lines 12 and 13?

```
1.   ;web_stackex.asm
2.             org     $1500
3.   sum       rmb   1
4.             org     $2000
5.             lds     #$2000
6.             ldaa    #12
7.             ldab    #15
8.             jsr     subra
9.             staa    sum
10.            swi
11.  subra     aba
12.            jsr     subrb
13.            rts
14.  subrb     clc
15.            sbca    #11       ;there is no immediate subtraction w/o carry
16.            rts
17.            end
```

In Class Exercise Regarding the Stack Cont'd.

```
as12, an absolute assembler for Motorola MCU's, version 1.2e

                         ;web_stackex.asm
1500                           org     $1500
1500                     sum   rmb     1
2000                           org     $2000
2000 cf 20 00                  lds     #$2000
2003 86 0c                     ldaa    #12
2005 c6 0f                     ldab    #15
2007 16 20 0e                  jsr     subra
200a 7a 15 00                  staa    sum
200d 3f                        swi
200e 18 06               subra aba
2010 16 20 14                  jsr     subrb
2013 3d                        rts
2014 10 fe               subrb clc
2016 82 0b                     sbca     #11 ;there is no immediate subtraction w/o carry
2018 3d                        rts
                               end
```

```
;webex4.asm, using eg04rev as a subroutine
;given three arrays of 8-bit numbers, look for a key value in each
;if found place address in result, if key not found, store –1 in result
;based on modifying ex. 4.2 in text and making it a subroutine
N1        equ     4
N2        equ     5
N3        equ     4
key       equ     100
          org     $1500
result1   rmw     1          ;storing addresses, thus need to reserve words
result2   rmw     1
result3   rmw     1
          org     $2000
          lds     #$2000   ;code grows up in memory and stack grows down/not interfering
          ldx     #array1  ;preparing to call search for the first time
          ldaa    #N1        ;  “
          ldab    #key       ; “ (not incorporating into search to keep the subr. Independent)
          bsr     search     ; using relative addressing because subroutine is close
          stx     result1
          ldx     #array2    ;array2 search starts here
          ldaa    #N2
          ldab    #key
          bsr     search
          stx     result2
          ldx     #array3    ;array3 search starts here
          ldaa    #N3
          ldab    #key
          bsr     search
          stx     result3
          swi                ; Actual exit from program
     ;Test Data and Subroutine SEARCH  IS ON NEXT SLIDE
```

```
;-------------------subroutine search----------------------------------------
;on entry:    x contains pointer to array
;             a contains N
;             b contains key value
;
;on return:   x contains result (Address or -1 if not found)

search   nop
loop     cmpb    1,x+     ;x(i) = key?
         beq     found
         dbne    a,loop   ;if not, decrement counter and continue
         ldx     #$ffff   ; only executed if key not in array
         rts                      ; “
found    dex                      ;restore X so it points to matched value
         rts              ;this rts is executed if key is found in data

array1   db      3,66, 100,44
array2   db      2,150,30,55,88
array3   db      200,100,56,109
         end
```

NOTE the *print screen* on the next page which shows:

1. Disassembly to see where data is stored: starting at $2036
2. Program execution and memory display of results: 20 38, FF FF (-1), and 20 40
3. The stack showing the last address which was stored in the stack: 20 28 (address following last bsr)

```
>

>asm 2000

xx:2000  CF2000        LDS   #$2000             >

xx:2003  CE2036        LDX   #$2036             >.

>g 2000

User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2028  2000  2040  0000     03:64        1001 0000
xx:2028  A7            NOP

>md 1500


1500  20 38 FF FF - 20 40 03 8F - 21 57 74 6B - 23 94 3A 73     8.. @..!Wtk#.:s
>

>md 1ff0


1FF0  4D C9 8E 15 - AC 99 00 90 - 64 03 20 40 - 00 00 20 28    M.......d. @.. (
>

>

>

>

>

>

>

>

>
```

Using **leas** (Load Effective Address into SP)

– Local variables **allocation** (by caller)

- **leas -n,sp ;** efficiently allocates n bytes in the stack for local variables by decrementing SP

– Local variables **de-allocation** (by subroutine)

- **Leas n,sp** ;efficiently de-allocates n bytes from the stack

Stack Frame
(also called activation record)

- Def: The region in the stack that holds incoming parameters, the subroutine return address, local variables, and saved registers

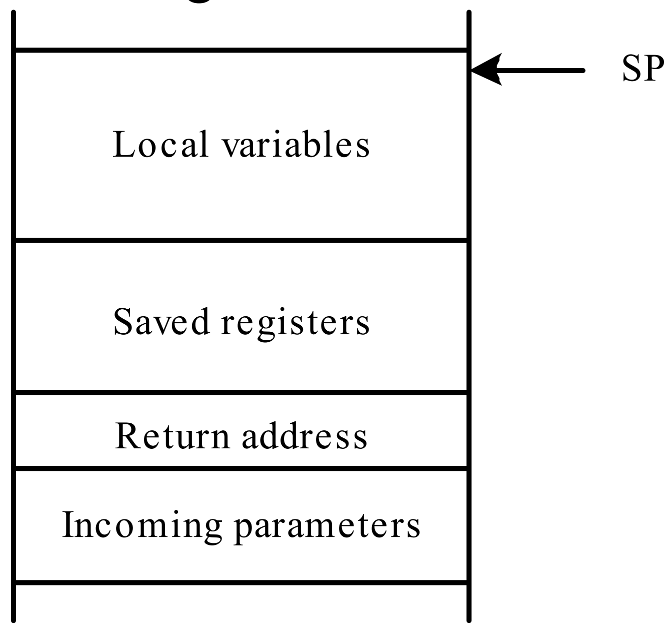

Figure 4.9 Structure of the 68HC12 stack frame

Example 4.10 rev'd. Draw the stack frame for the following program segment after the leas –10,sp instruction is executed.

```
            ldd     #$1234 ;1st onto stack
            pshd
            ldx     #$4000 ;2nd onto stack
            pshx
            jsr     sub_xyz
            …
sub_xyz     pshd
            pshx
            pshy
            leas    -10,sp   ; allocate space
            …
; now sp can be used as a pointer
; such as   stab  2,sp; stores B at 1 of the 10 locations
            . . .
            leas    +10,sp   ;de-allocate space
            puly,  etc.
            rts
```

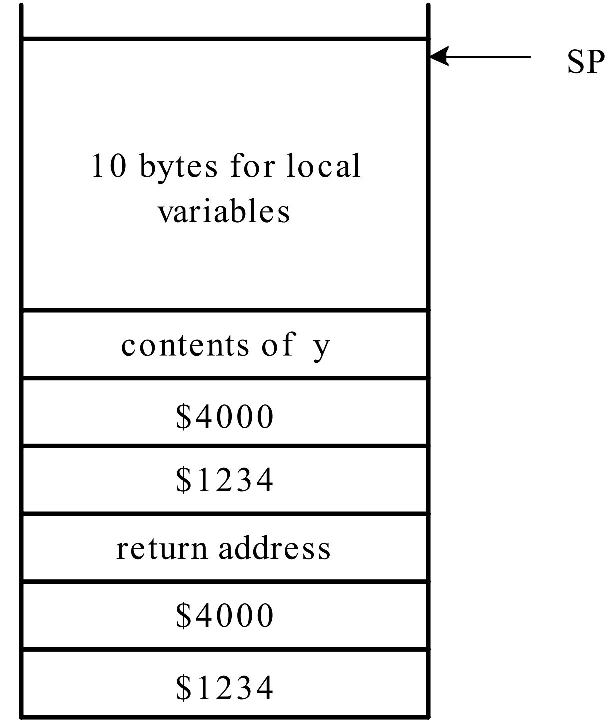

Figure 4.10 Stack frame of example 4.10

Bubble Sort Algorithm
for sorting N elements in ascending order
(inefficient but straightforward)

1. If x[i] > x[i+1], then switch
2. Inc i
3. If n-1 comparisons, go to 4, otherwise return to step 1
4. n <= N – 1 (last element guaranteed to be max and no need to examine again)
5. If n = 0, exit, otherwise return to

Enhancement: Use a flag, for each pass, which is tested to see if any exchange made and, if not, discontinue because array is sorted

Previous slide: Algorithm

This slide: Main, used for testing

Next slide: Subroutine

Following slide: Print Screen showing execution

```
;webex_bubl.asm
n                 equu    10
        org   $1500
array             db    $ED,$33,$44,$22,$00,$75,$15,$5A,$12,$AA
        org   $2000
        ldx   #array
        ldy   #n
        jsr   bublsort
        swi
```

```
;***subroutine bublsort***
;on entry>
;               x points to array (assumes unsigned numbers)
;               y contains N
;on return>
;               array has been sorted into ascending order
;               registers A,B,X, and Y are changed
bublsort        pshx
                dey                                 ; n-1 comparisons
                pshy
                beq         done        ;depends on pshy not affecting flags
loop            ldaa        0,x
                cmpa        1,x
                bls         contin
                ldab        1,x
                stab        0,x
                staa        1,x
contin          inx
                dbne        y,loop
                swi                     ;stops here for testing purposes
                puly
                pulx
                bra         bublsort
done            leas        4,sp        ;reset SP because of two pushes
                rts
```

NOTE Execution on next slide: maximum values bubbling to top and the final X value less each time because Y is decremented after each loop

```
User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2020  3BFA  1509  0000     ED:AA        1001 1000
xx:2020  31             PULY

>md 1500


1500  33 44 22 00 - 75 15 5A 12 - AA ED 74 6B - 23 94 7A 63    3D".u.Z...tk#.zc
>g

User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2020  3BFA  1508  0000     75:12        1001 1011
xx:2020  31             PULY

>md 1500


1500  33 22 00 44 - 15 5A 12 75 - AA ED 74 6B - 23 94 7A 63    3".D.Z.u..tk#.zc
>g

User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2020  3BFA  1507  0000     5A:12        1001 1001
xx:2020  31             PULY

>g

User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2020  3BFA  1506  0000     44:12        1001 1001
xx:2020  31             PULY

>md 1500


1500  00 22 15 33 - 12 44 5A 75 - AA ED 74 6B - 23 94 7A 63    .".3.DZu..tk#.zc
>_
```

Using the D-Bug12 Functions for I/O

Table 4.2 D-Bug12 monitor (version 4.x.x) routines

Subroutine	Function	pointer address
far main ( )	Start of D-Bug12	$EE80
getchar ( )	Get a character from SCI0 or SCI1	$EE84
putchar ( )	Send a character out to SCI0 or SCI1	$EE86
printf ( )	Formatted string output-translates binary values to string	$EE88
far GetCmdLine ( )	Get a line of input from the user	$EE8A
far sscanhex ( )	Convert ASCII hex string to a binary integer	$EE8E
isxdigit ( )	Check if a character (in B) is a hex digit	$EE92
toupper ( )	Convert lower-case characters to upper-case	$EE94
isalpha ( )	Check if a character is alphabetic	$EE96
strlen ( )	Returns the length of a NULL-terminated string	$EE98
strcpy ( )	Copy a NULL-terminated string	$EE9A
far out2hex ( )	Output 8-bit number as 2 ASCII hex characters	$EE9C
far out4hex ( )	Output a 16-bit number as 4 ASCII hex characters	$EEA0
SetUserVector ( )	Setup a vector to a user's interrupt service routine	$EEA4
far WriteEEByte()	Write a byte to the on-chip EEPROM memory	$EEA6
far EraseEE ( )	Bulk erase the on-chip EEPROM memory	$EEAA
far ReadMem ( )	Read data from the HCS12 memory map	$EEAE
far WriteMem ( )	Write data to the HCS12 memory map	$EEB2

Rules for using D-Bug12 I/O Functions

(All functions listed in Table 4.2 are written in C language.)

- The first parameter to the function is passed in accumulator D. The rest are pushed onto the stack in the reverse order they are listed in the function declaration.
- Parameters of type **char** will occupy the lower order byte of a word pushed onto the stack and must be converted to type **int**.
- Parameters pushed onto the stack before the function is called remain on the stack when the function returns. The caller “removes” passed parameters from the stack using the LEAS instruction.
- All 8- and 16-bit values are returned in accumulator D. A returned value of type **char** is returned in accumulator B. Boolean function results are 0 for false and non-zero for true.
- Registers are not preserved and, if needed, must be saved on the stack before calling the function.

Using the *printf* function

<u>Notes on Next Slide:</u>

- Uses the *printf* to send a message and data to the terminal
- By putting printf in a loop, one can print an array of numbers
- As required the last number (num2) printed is the first pushed on the stack
- An error occurred in assembling: “delimiter missing” due to improper quotes at beginning of string, retyped and was ok.
- Extra line feeds and carriage returns were added to provide space after output
- The values are converted to decimal before printing.

```
printf      equ     $EE88           ;function call to output a character
CR          equ     $0D
LF          equ     $0A
num1        equ     $23
num2        equ     $AA
            org     $1500
msg         fcc     "The value of num1 is  %d and num2 is  %d."
            fcb     CR,LF,CR,LF,CR,LF
            fcb     0                ;end of string character
            org     $2000
            lds     #$2000
            ldd     #num2    ;note last must go in stack first
            pshd
            ldab    #num1    ;now the first can go on the stack
            pshd
            ldd     #msg
            ldx     printf
            jsr     0,x
            leas    4,sp   ;remove data pushed on stack (reset stack pointer)
            swi
            end
```

Messages Terminal

```
>g 2000

The value of num1 is  35 and num2 is  170.


User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2015  2000  1527  1530    00:30         1001 0000
xx:2015  BF3323        CPS   $3323

>
```