ECE3120: Computer Systems Chapter 4: Subroutines

Manjeera Jeedigunta
http://blogs.cae.tntech.edu/msjeedigun21
Email: msjeedigun21@tntech.edu
Tel: 931-372-6181, Prescott Hall 120

Issues in Subroutine Calls

1. ***Parameter passing***
   - Use registers
   - Use the stack
   - Use global memory
2. ***Returning results***
   - Use registers
   - Use the stack
   - Use global memory
3. ***Local variable allocation***
   - Allocated by the callee
   - The following instruction is the most efficient method of local variable allocation.

   ```
   leas  -n,sp       ; allocate n bytes in the stack for local variables
   ```

4. ***Local variable deallocation***
   - space allocated to local variables must be deallocated
   - The following instruction is the most efficient method of local variable deallocation.

   ```
   leas  n,sp        ; deallocate n bytes from the stack
   ```

Stack Frame

- The region in the stack that holds incoming parameters, the subroutine return address, local variables, and saved registers is referred to as stack frame.
- The stack frame is also called **activation record**.

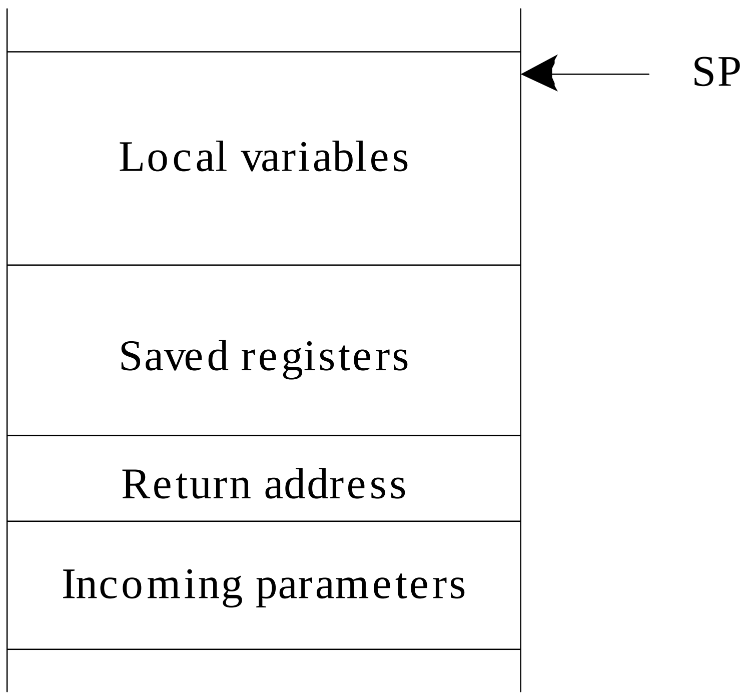

Figure 4.9 Structure of the 68HC12 stack frame

Binary Search ..Recap

Step 1

Initialize variables max and min to n -1 and 0, respectively.

Step 2

If max < min, then stop. 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 and also a sequence of instructions to test it.

Solution:

```
n       equ     30                  ; array count
key     equ     69                  ; key to be searched
arr     db      1,3,6,9,11,20,30,45,48,60
        db      61,63,64,65,67,69,72,74,76,79
        db      80,83,85,88,90,110,113,114,120,123
        org     $1000
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     $1500
        clra
        staa    min                 ; initialize min to 0
        staa    result              ; initialize result to 0
        ldaa    #n-1
        staa    max                 ; initialize max to n-1
        ldx     #arr                ; use X as the pointer to the array
```

Step 1

Max=N-1;Min=0

Step 2

Max<Min à stop

Step 3

Mean=max+min/2

Step 4

Key=arr[mean] à found

Step 5

Key<arr[mean]

Max=mean-1;goto step2

Step 6

Key>arr[mean]

Min=mean+1;goto setp2

```
loop        ldab    min
            cmpb    max
            lbhi    notfound
            addb    max         ; compute mean
            lsrb                ;          “
            stab    mean        ; save mean
            ldaa    b,x         ; get a copy of the element
                                ;arr[mean]
            cmpa    #key
            beq     found
            bhi     search_lo   ;key<arr[mean]
            ldaa    mean        ;key>arr[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
            end
```

Step 1

Max=N-1;Min=0

Step 2

Max<Min à stop

Step 3

Mean=max+min/2

Step 4

Key=arr[mean] à found

Step 5

Key<arr[mean]

Max=mean-1;goto step2

Step 6

Key>arr[mean]

Min=mean+1;goto setp2

Example: Convert the binary search program into a subroutine so that it can be called by other program units. Let the starting address of the array, array count and the key to be matched be passed via the stack and the result to be returned in an accumulator D

The main program is as follows:

```
mean     equ  0              ; distance from the top of the stack
max      equ  1              ;distance ;;
min      equ  2              ;distance
key_lo   equ  8              ;key local
arcnt_lo equ  9              ;N in local variables
arr_lo   equ  10             ;array address in local
n        equ  30             ; array count
key      equ  2              ; key to be searched
arr      db   1,3,6,9,11,20,30,45,48,60
         db   61,63,64,65,67,69,72,74,76,79
         db   80,83,85,88,90,110,113,114,120,123
         org  $1000
result   rmb  1
         org  $1500
         lds  #$1500         ; initialize the stack pointer
         ldx  #arr               ; pass array base address
         pshx                ;    “
         ldaa #n             ; pass array count
         psha                ;    “
         ldaa #key           ; pass the search key
         psha                ;    “
         jsr  bin_search
         staa result
         swi
```

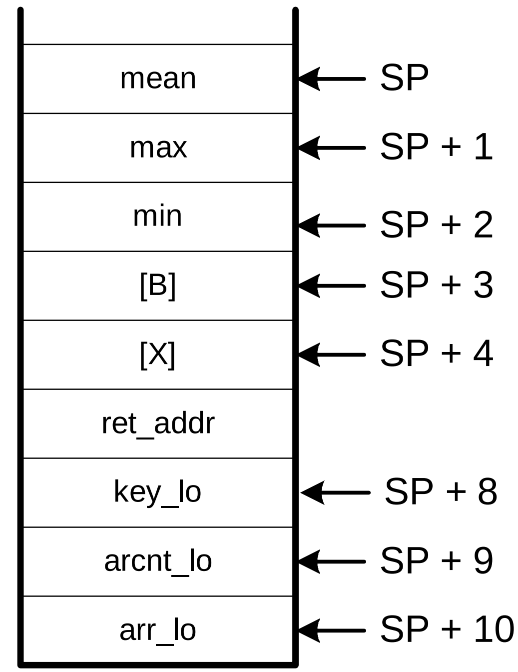

Figure E4.2 Stack for Ex4.1

The following subroutine implements the binary search subroutine and returns the search result in A. A contains a 1 if the key is found in the array. Otherwise, it contains a 0.

```
bin_search pshx
           pshb
           leas   -3,sp          ; allocate 3 bytes for local
           clr    min,sp         ; initialize min to 0
           ldaa   arcnt_lo,sp    ;#n
           deca                  ;n-1
           staa   max            ; initialize max to arcnt - 1
           ldx    arr_lo,sp      ; use X as the pointer to the array
loop       ldab   min,sp
           cmpb   max,sp
           lbgt   notfound       ; if min > max, then not found
           addb   max,sp         ; compute mean
           lsrb                  ;        "
           stab   mean,sp        ; save mean
           ldaa   b,x            ; get a copy of the element arr[mean]
           cmpa   key_lo,sp
           beq    found
           bgt    search_lo
           ldaa   mean,sp
           inca
           staa   min,sp         ; place mean+1 in min to continue
           bra    loop
search_lo  ldaa   mean,sp
           deca
           staa   max,sp
           bra    loop
found      ldaa   #1
           bra    done
notfound   ldaa   #0
done       leas   3,sp
           pulb
           pulx
           rts
           end
```

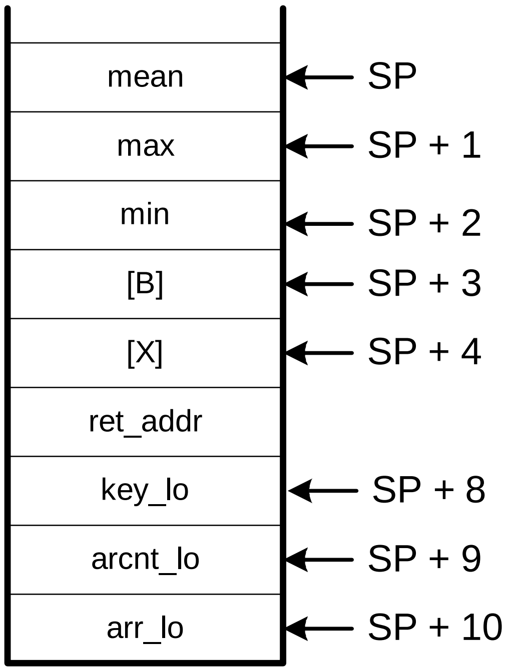

Figure E4.2 Stack for Ex4.1

Finding the Square Root

- One of the methods for finding the square root of an integer is based on the following equation:

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} \qquad (4.1)$$

- Equation 4.1 can be transformed into

$$n^2 = \sum_{i=0}^{n-1} (2i+1) \qquad (4.2)$$

- The algorithm for finding the square root of an integer (q) based on equation 4.2 is illustrated in the flowchart shown in Figure 4.16.

n^2<q
n^2=q
n^2>q
n à sqrt(q)

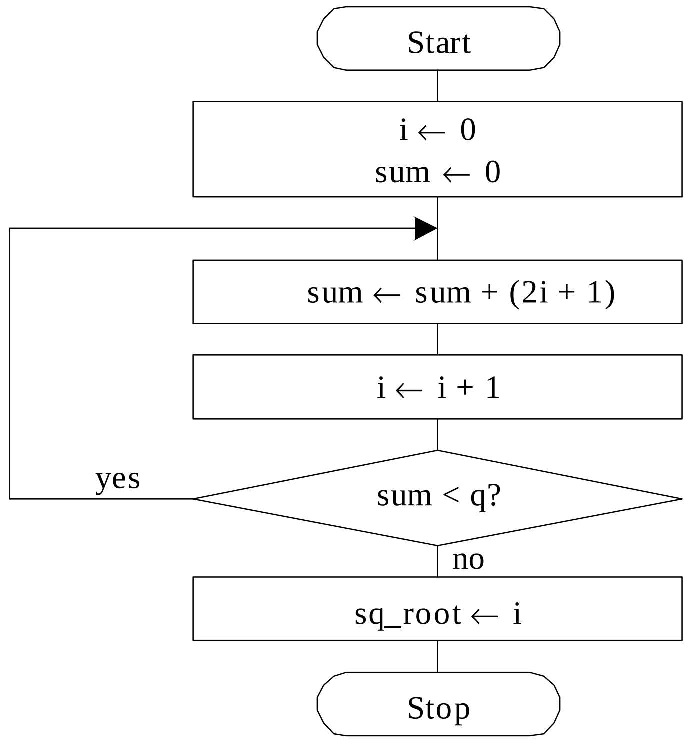

Figure 4.16 Algorithm for finding the square root of integer q.

Example 4.14 Write a subroutine to implement the square root algorithm. This subroutine should be able to find the square root of a 32-bit unsigned integer. The parameter is pushed onto the stack and the square root is returned in accumulator D.

Solution:

The stack frame is shown in Figure 4.17. The subroutine and the instruction sequence for testing the subroutine is shown in the following pages.

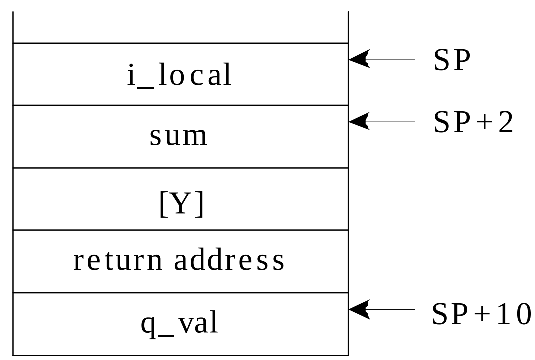

Figure 4.17 Stack frame of example 4.14

Example 4.14 :-Finding Square Root

```
q_hi     equ   $000F        ; upper word of q
q_lo     equ   $4240        ; lower word of q
i_local  equ   0            ; distance of local variable i from the top of the stack
sum      equ   2            ; distance of local variable sum from the top of the stack
q_val    equ   10           ; distance of incoming parameter q from the top of stack
local    equ   6            ; number of bytes allocated to local variables

         org   $800
sq_root  rmb   2            ; to hold the square root of q
         org   $1000
         ldd   #q_lo
         pshd
         ldd   #q_hi
         pshd
         jsr   find_sq_root
         std   sq_root
         leas  4,sp
         swi
```

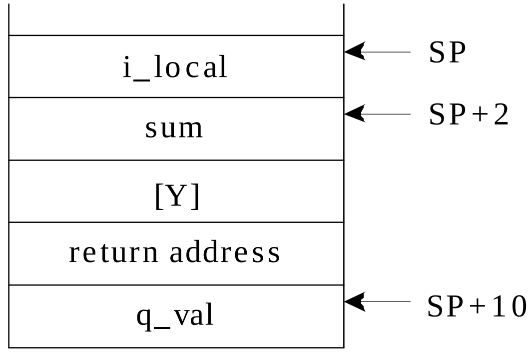

Figure 4.17 Stack frame of example 4.14

Subroutine: finding sq root

```
find_sq_root    pshy                    ; save y in the stack
                leas        -local,sp   ; allocate local variables
                ldd         #0          ; initialize local variable i to 0
                std         i_local,sp  ;           "
                std         sum,sp      ; initialize local variable sum to 0
                std         sum+2,sp    ;           "
loop            ldd         i_local,sp
                ldy         #2
                emul                    ; compute 2i
; add 2i to sum
                addd        sum+2,sp
                std         sum+2,sp
                tfr         y,d
                adcb        sum+1,sp
                stab        sum+1,sp
                adca        sum,sp
                staa        sum,sp
```

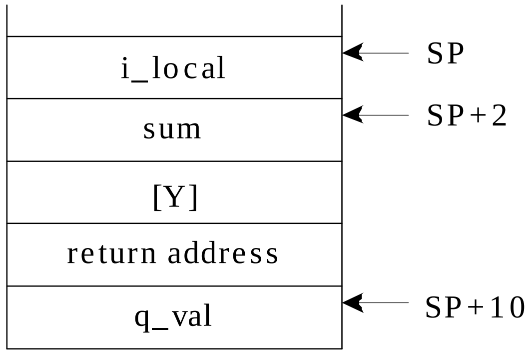

Figure 4.17 Stack frame of example 4.14

```
        ; add 1 to sum (need to propagate carry to the most significant byte of
sum)
        ldaa    #1
        adda    sum+3,sp
        staa    sum+3,sp
        ldaa    sum+2,sp
        adca    #0
        staa    sum+2,sp
        ldaa    sum+1,sp
        adca    #0

        staa    sum+1,sp
        ldaa    sum,sp
        adca    #0
        staa    sum,sp
; increment i by 1
        ldd     i_local,sp
        addd    #1
        std     i_local,sp
```

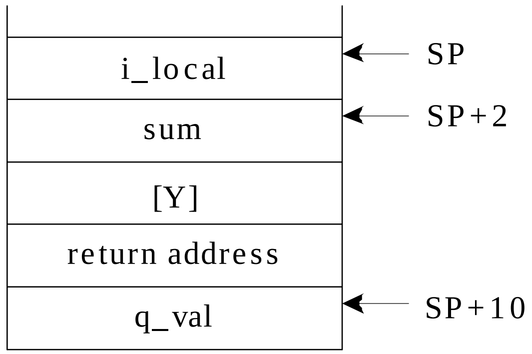

Figure 4.17 Stack frame of example 4.14

Contd..

```
; compare sum to q_val by performing subtraction (need consider borrow)
        ldd     sum+2,sp
        subd    q_val+2,sp
        ldaa    sum+1,sp
        sbca    q_val+1,sp
        ldaa    sum,sp
        sbca    q_val,sp
        lblo    loop
        ldd     i_local,sp          ; place sq_root in D before return
; deallocate space used by local variables
exit    leas    local,sp
        puly
        rts
        end
```

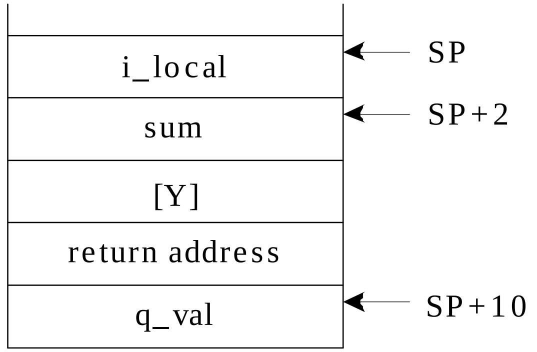

Figure 4.17 Stack frame of example 4.14