Using the stack to *pass parameters* and *store local variables* --- and accessing the variables using the *stack pointer*.

- **Using the stack to store local variables**

  - The system stack can also be used to store local variables.

  - Just like **parameters**, the **local variables** of a subroutine is **only** active (needed) when the subroutine is running.

    So it is very efficient to store local variables on the stack because the **order** in which the subroutines become active/inactive is **FILO** - exactly what a stack does.

  - **Example in high-level language:**

```
main:

  int A[10];
  int sum;

  sum = ArraySum(A, 10);
```

```
int ArraySum(int A[], int n)
{
   int i, s;    // Local variables

   s = 0;
   for (i = 0; i < n; i++)
      s = s + A[i];

   return (s);
}
```

  - **"main" in assembler language:**

```
main:

    move.l #A, -(a7)        * Pass address of array A
    move.l #10, -(a7)       * Pass # elements in array

    bsr     ArraySum

    adda.l #8, a7           * remove (pop) #A and #10 from stack

    move.l d0, sum          * put return value in variable "sum"
```

  - **Stack content when subroutine "sum" begins execution:**

```
          +----------------+
          | free space     |
          +----------------+ <----- a7 (stack pointer)
   0(a7)  | return address|
          +----------------+
   4(a7)  |      n   (#10)  |
         +---------------+
  8(a7)  |      A   (#A) |
         +---------------+
         |    .....      |
```

- **Subroutine "ArraySum" in assembler language:**

  Pay special attention to how **a7** is used to access the parameters on the stack !!!

The subroutine returns the value in **register D0**.

```
ArraySum:
        suba.l #8, a7           * Create 2 local variables on stack !!!

* Note: NOW the stack is:
*
* Offsets
*         +---------------+ <----- a7 (stack pointer)
*   0(a7) |      s        |     (you decide which location is s and i)
*         +---------------+
*   4(a7) |      i        |     (This program uses s and i in the given manner)
*         +---------------+
*   8(a7) | return address|
*         +---------------+
*  12(a7) |      n  (10)  |
*         +---------------+
*  16(a7) |      A  (#A)  |
*         +---------------+

        move.l #0, 0(a7)        * s = 0
        move.l #0, 4(a7)        * i = 0
While:
        move.l 4(a7), d0        * puts local variable i in d0
        move.l 12(a7), d1       * puts parameter n in d1
        cmp.l  d1, d0

        BGE     WhileEnd        * Exit while loop if i >= n

* ---- body of while loop

        movea.l 16(a7), a0      * put base address of array in A0
*                                 (prepare to access A[i])

        move.l  4(a7), d0       * now d0 = i
        muls    #4, d0          * offset is now in d0
        move.l  (a0, d0.w), d0  * put A[i] in d0

        add.l   d0, 0(a7)       * add A[i] to local variable s

        move.l  4(a7), d0
        add.l   #1, d0
        move.l  d0, 4(a7)       * i = i + 1

        BRA While

WhileEnd:

        move.l  0(a7), d0       * Return s in the agreed location (d0)

* Note: the stack is STILL:
```

```
*
* Offsets
*           +----------------+ <----- a7 (stack pointer)
*    0(a7)  |       s        |      (you decide which location is s and i)
*           +----------------+
*    4(a7)  |       i        |      (This program uses s and i in the given manner)
*           +----------------+
*    8(a7)  | return address|
*           +----------------+
*   12(a7)  |       n   (10)   |
*           +----------------+
*   16(a7)  |       A   (#A)   |
*           +----------------+
*
* If you return NOW, your program will NOT pop the return address
* into the Program counter and it will CRASH !!!

        adda.l #8, a7       * Remove local variables !!!

* NOW the stack is:
*
* Offsets
*           +----------------+ <----- a7 (stack pointer)
*    8(a7)  | return address|
*           +----------------+
*   12(a7)  |       n   (10)   |
*           +----------------+
*   16(a7)  |       A   (#A)   |
*           +----------------+
*
* NOW you can rexecute the return instruction !!!

        rts
```

- **Example Program:** (Demo above code) Example
    - Prog file: click here

- **So what is the big deal about *passing parameters* and *storing local variables* on the system stack ?**
    - Consider the following sequence of **function calls** and the corresponding **creation** of **parameter variables** and **local variables** on the **system stack**:

```
  main --------->  func1 -----------> func2 -------------> func3

Stack:
                                                             +------------+
                                                             | func3 local|
                                                             +------------+
                                                             | ret. func2 |
                                                             +------------+
                                                             | func3 param|
                                          +------------+     +------------+
                      | func2  local|      | func2  local|
                      +-------------+      +-------------+
                      | ret.  func1 |      | ret.  func1 |
                      +-------------+      +-------------+
                      | func2  param|      | func2  param|
+-------------+       +-------------+      +-------------+
| func1  local|       | func1  local|      | func1  local|
+-------------+       +-------------+      +-------------+
| ret.  main  |       | ret.  main  |      | ret.  main  |
+-------------+       +-------------+      +-------------+
| func1  param|       | func1  param|      | func1  param|
+-------------+       +-------------+      +-------------+
```

Notice that:

- **Each time** a **function** is **invoked**, a **new set** of **parameter variables** and **local variables** are **created** for **that invocation**
- In other words:
  - The **parameter variables** and **local variables** are **private** for **each method invocation**
- That is **exactly** the **problem** that we **must solve** in order to **implement *recursion*** (See: click here)

- **A short-coming of our implementation**
  - The **previous example** uses the **stack pointer A7** to **access parameter variables** and **local variables** of the function.
  - **Often**, we may need to **change** a function **after we have written the code**
  - The **technique** of **using the *stack pointer*** to access the **parameter variables** and **local variables** has a **severe short-coming** when we need to:
    - ***Add* or *remove*** one or more **parameter variable(s)** or **local variable(s)** from the function

Because:

- When a **parameter variable** or a **local variable** is **added** or **deleted**, the **offset** of **some (other) variables** are **changed** !!!

  The **relative position** of **some variable** from the **top of the stack** may be **changed**
- We **used the *offset* from the stack top** to access the **appropriate variable**

- When the **offset** of a **variable** is changed, we need to **adjust the *offset*** of the variable.

- This is a **very messy affair** !!!

○ **Example:**

Offsets used to access variables BEFORE adding extra local variable

```
       +---------------+ <----- a7
 0(a7) |       s       |
       +---------------+
 4(a7) |       i       |
       +---------------+
 8(a7) | return address|
       +---------------+
12(a7) |       n   (10)|
       +---------------+
16(a7) |       A   (#A)|
       +---------------+
```

Offsets used to access variables AFTER adding extra local variable

```
       +---------------+ <----- a7
 0(a7) |       x       |
       +---------------+
 4(a7) |       s       |
       +---------------+
 8(a7) |       i       |
       +---------------+
12(a7) | return address|
       +---------------+
16(a7) |       n   (10)|
       +---------------+
20(a7) |       A   (#A)|
       +---------------+

  All offsets has changed !!!
```

- The offset of the **existing variables** from **A7** are **CHANGED**

- So the **existing program code** is accessing the existing variables **INCORRECTLY** !!!

- This will obvious cause major headaches when the programmer needs to alter the program at a later time...