Second Recursive Function: Fibonacci

- **The *classic* Fibonacci function**
  - The **classic Fibonacci function**

```
int fib(int n)
{
  if (n = 0)
     return 1;
  else if (n == 1)
     return 1;
  else
   {
     return fib(n-1) + fib(n-2);
   }
}
```

- **Calling the Fibonacci function**
  - The **Fibonacci** function is **called** with a statement that look like this:

```
  int n, result;

  result = fib(n);
```

  - **Passing** parameter **n** from **main** program to **fib()**:

    - **Because** the **Fibonacci** function is ***recursive***, **each invocation** must has its **own** copy of **parameter variables**
    - This can ***only*** be **realized** by using a **stack** !!!

Therefore, the **main** program **must** pass the parameter **n** to **Fibonacci** by pushing **n** onto the system stack:

```
    move.l n, -(a7)
```

This **instruction** will create the following **stack structure:**

```
+---------------------+ <------------ Stack pointer (A7)
|     parameter n     |
+---------------------+
|     .......         |
|   rest of the stack |
|     .......         |
```

- **Next**, the **main** program will **call** the `fib( )` **function** with a **bsr** instruction:

```
   bsr   fib
```

This will **push** the **return address** on the **stack** and create the **following stack structure**:

```
+---------------------+ <------------ Stack pointer (A7)
|     return address  |
+---------------------+
|     parameter n     |
+---------------------+
|     .......         |
|   rest of the stack |
|     .......         |
```

- **The *Prelude* of the Fibonacci function**

  - If you **look** at the **Fibonacci function** ***carefully***:

```
int fib(int n)
{
  if (n = 0)
     return 1;
  else if (n == 1)
     return 1;
  else
   {
     return fib(n-1) + fib(n-2);
   }
}
```

You will **notice** that there are ***two*** (**recursive**) calls to **Fibonacci**.

- **Fact:**

  - **In order** to ***compute***:

```
fib(n-1) + fib(n-2)
```

we **must**:

- **Call** the function `fib(n-1)`

  (and **obtain** the **return value (x)**)

- ***Then***, **call** the function `fib(n-2)`

  and **obtain** the **return value (y)**

And add **x** and **y**

- ***Very* important fact:**

  - ***After*** we **obtained** the **return value (x)** from `fib(n-1)`, we ***cannot*** compute the **result *yet*** !!!
  - We must ***still*** find the ***second* Fibonacci value** (`fib(n-2)`) **before** we can compute the **result** !!!

  Conclussion:

  - We must ***save*** the **return value (x)** in a ***safe* place** !!!

  Where is a ***safe* place** in ***recursive* programming**:

  - The ***only* safe place** in to ***recursive* programming** to **store** values is:
    - ***Local* variables** on the ***stack*** !!!

  Therefore:

  - We **must** create ***one* local variable** to **save** the **return value (x)** of the **call** `fib(n-1)`

- The **prelude** of the **Fibonacci function** is **therefore**:

```
********************************** PRELUDE
```

```
        move.l a6, -(a7)      ; Save caller's frame pointer
        move.l a7, a6         ; Setup my own frame pointer
        suba.l #4, a7         ; Allocate space for local variable for fib(n-1)
*********************************
```

I will explain what each one does below. Make sure that you realise that the structure of the stack frame is like this when the prelude is **always** executed:

```
+---------------------+ <------------ Stack pointer (A7)
|     return address  |
+---------------------+
|     parameter n     |
+---------------------+
|     .......         |
|  rest of the stack  |
|     .......         |
```

- move.l a6, -(a7)

  This will save the frame pointer on the stack, creating this partial stack frame structure:

```
+---------------------+ <------------ Stack pointer (A7)
|     saved a6        |
+---------------------+
|     return address  |
+---------------------+
|     parameter n     |
+---------------------+
|     .......         |
|  rest of the stack  |
|     .......         |
```

- move.l a7, a6

  This will make the frame pointer A6 points to the stack frame that is now being built:

```
+---------------------+ <---- Frame pointer A6 & Stack pointer (A7)
|     saved a6        |       point to the same location....
+---------------------+
|     return address  |
+---------------------+
|     parameter n     |
+---------------------+
|     .......         |
|  rest of the stack  |
|     .......         |
```

- suba.l #4, a7

  This will push the stack pointer A7 4 bytes up, allocating 1 integer variables -- used to save the return value of fib(n-1).

```
+---------------------+ <---- Stack pointer (A7)
|     help (local var)|
+---------------------+ <---- Frame pointer (A6)
|     saved a6        |
+---------------------+
|     return address  |
+---------------------+
|     parameter n     |
```

```
+--------------------+
|    .......          |
|  rest of the stack  |
|    .......          |
```

- When the prelude is finish, the stack frame is complete and the actual function can begin.

- **How to access the parameter and local variables inside the `fib` function**
  - **How** to access the **parameter** and the **local variables** inside the **`fib( )` function**:
    - Parameter **n** is located 8 bytes **below** starting from the address contained in the frame pointer A6.

      So the address mode that will let you get to this variable is **8(A6)**

    - Local variable **help** is located 4 bytes **above** starting from the address contained in the frame pointer A6.

      So the address mode that will let you get to this variable is **-4(A6)**

- **Calling `fib( )` from within `fib( )`:**
  - **How Fibonacci** calls **itself:**
  - **Fact:**
    - It is the ***same* way** as **how** the **main program** calls the **Fibonacci function** !!!

  - **Method:**
    - **Pass** the **parameter** on the **stack**
    - **Call** the **Fibonacci** function
    - **Clean up** the **parameter**
    - **Use** the **return value** (in **D0**)

    Make sure you **pop the parameter** from the stack after Fibonacci returns - because the parameter has not been cleaned up.

The following is the program fragment where Fibonacci calls fib(n-1):

```
      move.l 8(a6), d0     ; retrieve parameter n into register d0
      sub.l #1, d0         ; d0 = n - 1
*
* ---------------------------- ; fib is calling fib now !!!!
*
       move.l d0, -(a7)     ; Push (n-1) as parameter
       bsr    fib           ; Call fib(n-1)
       adda.l #4,a7         ; Clean up parameter from stack

       move.l d0, -4(a6)    ; help = return value of fib(n-1) in register D0
```

Fibonacci will call itself a second time with value n-2. The following is the program fragment where Fibonacci calls fib(n-2):

```
      move.l 8(a6), d0     ; retrieve parameter n into register d0
      sub.l #2, d0         ; d0 = n - 2
*
* ---------------------------- ; fib is calling fib again....
*
       move.l d0, -(a7)     ; Push (n-2) as parameter
       bsr    fib           ; Call fib(n-2)
       adda.l #4,a7         ; Clean up parameter from stack

       add.l -4(a6),d0        ; Compute the value: fib(n-1)+fib(n-2)
```

I have highlighted the difference between the first call and the second. The second call uses a different parameter value and stores the return value in a different local variable !

- The **full** assembler program:

```
* ==============================================
* main: result = fib(n)
* ==============================================

Start:
        movea.l #12345,a6       ; Store something in a6 to make it dramatic

        move.l n,-(a7)          ; Call fib(n)
        bsr    fib
        adda.l #4,a7            ; pop parameter off the stack
        move.l d0,result        ; result = return value

Stop:   nop

n:      dc.l 5                  ; variable n (input)
result: ds.l 1                  ; variable result (output)



* ========================================================= Fib
* int fib(int n)
```

```
* {
*   if (n = 0)
*      return 1;
*   else if (n == 1)
*      return 1;
*   else
*    {
*      return fib(n-1) + fib(n-2);
*    }
* }
*
* ----------------------------------------------------
* Input: n on stack
* Output: fib(n) in register d0

fib:
********************************* PRELUDE
        move.l a6,-(a7)         ; Save caller's frame pointer
        move.l a7,a6            ; Setup my own frame pointer
        suba.l #4,a7            ; Allocate space for local var. "help"
*********************************
* Start of function....

        move.l 8(a6),d0         ; n
        cmp.l  #0,d0            ; n == 0 ?
        bne    else1

        move.l #1,d0            ; return(1)

********************************* POSTLUDE
        move.l a6,a7            ; Deallocate local variable(s)
        move.l (a7)+,a6         ; restore caller's frame pointer
*********************************
        rts

else1:  move.l 8(a6),d0         ; n
        cmp.l  #1,d0            ; n == 1 ?
        bne    else2

        move.l #1,d0            ; return(1)

********************************* POSTLUDE
        move.l a6,a7            ; Deallocate local variable(s)
        move.l (a7)+,a6         ; restore caller's frame pointer
*********************************
        rts

else2:
********************************* fib(n-1)
        move.l 8(a6),d0         ; n
        sub.l #1,d0             ; n - 1
        move.l d0,-(a7)         ; Push (n-1)
        bsr    fib              ; call fib(n-1) - will return to next instruction

        adda.l #4,a7            ; Clean up: Pop parameter (n-1) from stack

        move.l d0,-4(a6)        ; Save return value (fib(n-1)) in local var !!

********************************* compute fib(n-2)
        move.l 8(a6),d0         ; n
        sub.l #2,d0             ; n - 2
        move.l d0,-(a7)         ; Push (n-2)
        bsr    fib              ; call fib(n-2) - will return to next instruction

        adda.l #4,a7            ; Clean up: Pop parameter (n-2) from stack
```

```
**********
        add.l -4(a6),d0       ; Compute the return value: fib(n-1)+fib(n-2)

********************************* POSTLUDE
        move.l a6,a7           ; Deallocate local variable(s)
        move.l (a7)+,a6        ; restore caller's frame pointer
*********************************
        rts

End:
        end
```

- **Example Program:** (Demo above code)

  Example

  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `as255 fib`
- To run: use **Egtapi** `m68000`