Subroutine call and return: `bsr` (`jsr`) and `rts`

- **Marked differences between methods in high level language and assembler**
  - Methods (or subroutines) in high level languages have very nice syntax structures to highlight where the method begins and ends...

    Methods (or subroutines) in assembler are nothing more than a series of instructions with a **label**
  - Methods in high level languages have
    - Input parameters
    - Return values

    In assembler programming, input parameters and return values are **symbolic** - they are **agreements** on **where** the input value are stored.

    As a result, in some examples, you will **NOT** see the **names** of the input parameters in the assembler code !!!
- **Assembler Instructions to implement subroutine (method) calling**
  - Modern computer provides 2 instructions that user can use to implement:
    - function (method) invocation (calling a method): **BSR**
    - returning from a function call: **RTS**
  - Syntax of the Branch to Subroutine (**BSR**) instruction:

```
BSR  label

Effect:
       (1) Push the Program Counter (PC) onto the system stack
       (2) Branch to memory location marked by the label "label"
```

    - The operation "Push the Program Counter" onto the system stack has the effect of **saving** the **address** of the instruction that **follows** the **BSR** instruction on the system stack !!!
    - This address is where the program must resume when the subroutine ends.
    - This retrurn address is a "bread crump" - using the analogy of Hansel and Gretel....
- **Example using the BSR instruction**

  Suppose the follwoing program segment is located in the following memory locations: (the address of the locations is given in column 1 and the instructions are given in ASSEMBLER code rather than BINARY code)

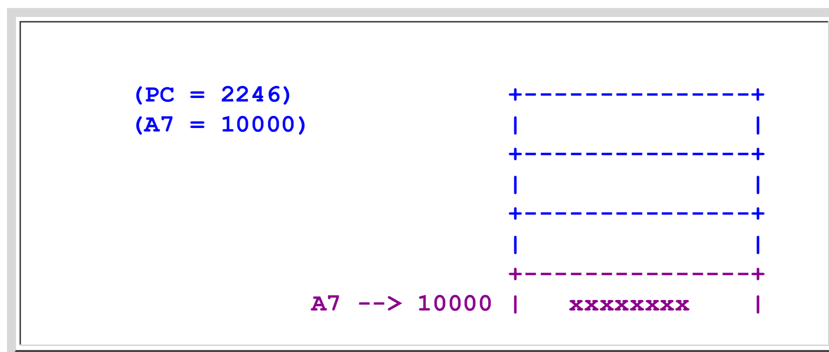
```
Memory          Instruction
Address:        in the memory address:
---------       ----------------------


 2244:              BSR     label
 2246:              MOVE.L  #0, Dummy1


 4012:     label:   MOVE.L  #0, Dummy2
 4014:              RTS
```

- **Important Fact**:

  at the moment that the CPU is executing the BSR instruction, the program counter would have been incremented and points to the next instruction (which is at address 2246.

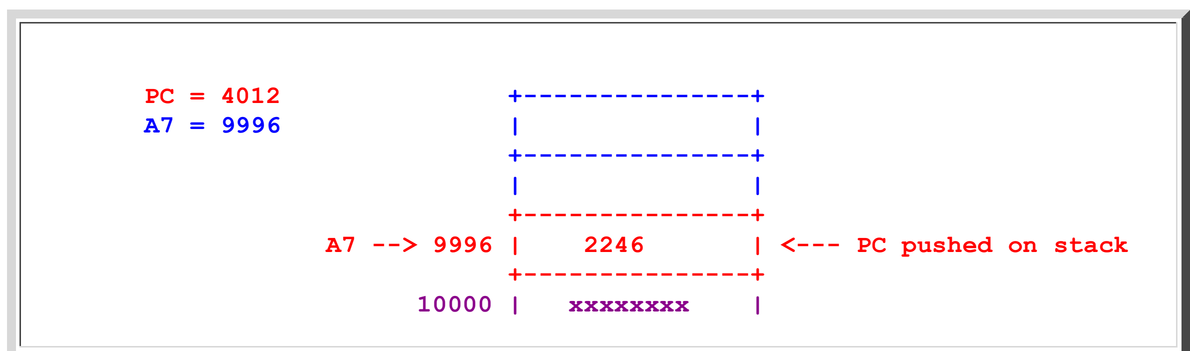
  Therefore, PC is equal to 2246 (address of the NEXT instruction)

  Supose at the moment that the CPU is executing the BSR instruction (i.e., **before** the BSR instruction is executed), the stack stack point A& = 10000, so the stack looks like this:



  Then:

  AFTER executing the "BSR label" instruction, the stack will be changed into:



  In addition:

  the PC will contain the value of "label" - so the program made a JUMP to address "label" - ("label" marks the memory address 4012 !):

- **Returning from a subroutine call**
  - The Return from Subroutine (RTS) instruction

```
RTS

Effect:

  (1) Pop the top of the stack into the Program Counter (PC)
```

- **Example using the RTS instruction** We continue with the example from above:

```
Memory         Instruction
Address:       in the memory address:
---------      -----------------------


 2244:              BSR     label
 2246:          MOVE.L #0, Dummy1


 4012:    label: MOVE.L #0, Dummy2
 4014:              RTS
```

```
PC = 4012      +---------------+
A7 = 9996      |               |
               +---------------+
               |               |
               +---------------+
  A7 --> 9996  |     2246      |
               +---------------+
        10000  |   xxxxxxxx    |
```

Suppose the CPU fetched "RTS" and executes it...

AFTER the CPU finishes executing "RTS", the stack will be changed to:

```
PC = 2246                 +---------------+
A7 = 10000                |               |
                          +---------------+
                          |               |
                          +---------------+
                     9996 |     2246      | <- NOT part of the stack !
                          +---------------+
              A7 -->10000 |   xxxxxxxx    |
```

  - **NOTE:** although the value **2246** is still in memory, it is **NO LONGER** part of the system stack - because the stack top (indicated by A7) has moved **below** that memory location !!!!

Note that the value **2246** which was at the top of the stack is now in the PC !!!

> In other words, the PC has been updated to **2246** (In computer science jargon: **2246** was POPPED from the program stack into the PC.)

Note also that the value **2246** is the location **AFTER** the **BSR** instruction !!!

Becasue PC = 2246, the **next** instruction that the CPU will fetch and execute is the one **after** the **BSR** instruction !!

That is **exactly** the location where you want to be when you **return** from the called function !!!!

- **"Format" of a method/function in assembler code**
  - While functions/methods look very "formidable" in high level languages (such as Java), functions/methods written in assembler does not look like much:
  - A function in assembler code looks something like this:

```
FuncName:   ......
            ......  (assembler instructions that comprise
            ......   the body of the function)
            ......
            RTS
```

  - Needless to say that this is a far-cry from the "nice-looking" (human readable) block structures in a high level language.
  - Furthermore, functions/methods written in assembler are **very hard** to discern - especially if you remember that there are **many labels** all over the place from **IF** and **WHILE** statements !!!

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