on exit from the subroutine. In this way the subroutine can preserve the value of the register and not corrupt the value as would otherwise be the case.

See Chapter 15 for more information on using the stack.

3.2.2 The Link Register, LR or R*14*

Register R*14* is also known as the *Link Register* or LR.

It is used to hold the return address for a subroutine. When a subroutine call is performed via a BL instruction, R*14* is set to the address of the next instruction. To return from a subroutine you need to copy the Link Register into the Program Counter. This is typically done in one of the two ways:

- Execute either of these instructions:

  ```
  MOV   PC, LR        or        BAL   LR
  ```

- On entry to the subroutine store R*14* to the stack with an instruction of the form:

  ```
  STMFD   SP!,{⟨registers⟩, LR}
  ```

  and use a matching instruction to return from the subroutine:

  ```
  LDMFD   SP!,{⟨registers⟩, PC}
  ```

  This saves the Link Register on the stack at the start of the subroutine. On exit from the subroutine it collects all the values it placed on the stack, including the return address that was in the Link Register, except it returns this address directly into the Program Counter instead.

See Chapter ?? on page ?? for further details of using the stack, and Chapter 15 on page 113 for further details on using subroutines.

When an exception occurs, the exception mode's version of R*14* is set to the address after the instruction which has just been completed. The SPSR is a copy of the CPSR just before the exception occurred. The return from an exception is performed in a similar way to a subroutine return, but using slightly different instructions to ensure full restoration of the state of the program that was being executed when the exception occurred. See 3.4 on page 29 for more details.

3.2.3 The program counter, PC or R*15*

Register R*15* holds the *Program Counter* known as the PC. It is used to identify which instruction is to be preformed *next*. As the PC holds the address of the next instruction it is often referred to as an *instruction pointer*. The name "program counter" dates back to the times when program instructions where read in off of punched cards, it refers to the card position within a stack of cards. In spite of its name it does not actually count anything!

Reading the program counter

When an instruction reads the PC the value returned is the address of the current instruction plus 8 bytes. This is the address of the instruction *after* the *next* instruction to be executed[2].

[2] This is caused by the processor having already fetched the next instruction from memory while it was deciding what the current instruction was. Thus the PC is still the next instruction to be executed, but that is not the instruction immediately after the current one.