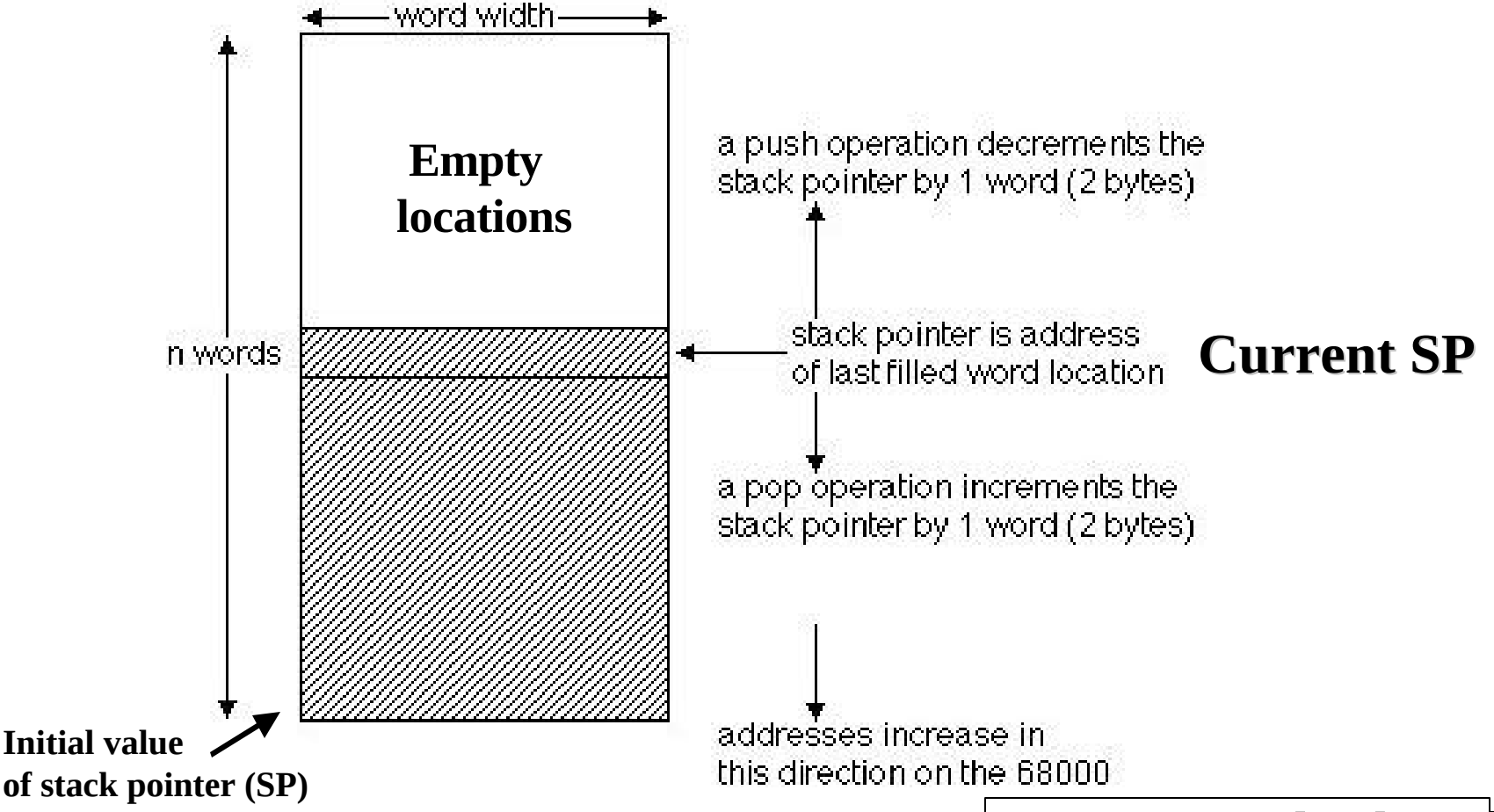
Stacks

- **A stack is a First In Last Out (FILO) buffer containing a number of data items usually implemented as a block of *n* consecutive bytes, words or long words in memory .**
- **The address of the last data item placed into the stack is pointed to by the Stack Pointer (SP).**
- **In the example below, the stack is composed of words.**



Common Uses of Stacks

- **Data storage:**
  - **This is similar to an array, but is more useful for handling input/output information.**
- **Subroutines calls:**
  - **Before a subroutine is called, the return address (the address of the next instruction of the calling program) is pushed (saved) onto the stack and the stack pointer is adjusted.**
- **Subroutines Returns:**
  - **Upon completion of a subroutine the return address is popped (retrieved) from the stack and loaded in the program counter (PC) and the stack pointer is adjusted.**
- **Subroutine Parameter-Passing:**
  - **Parameters (operands, addresses, etc.) needed by a subroutine may be passed to the subroutine by pushing them onto the stack forming local variables.**
  - **Upon completion such a subroutine all evidence of parameter-passing must be removed from the stack (stack clean-up).**

68000 Stacks

In the 68000:

- **Stack addresses begin in high memory ($07FFE for example) and are pushed toward low memory ($07F00 for example). i.e. 68000 stacks grow into low memory.**
- **Other CPUs might do this in the reverse order (grow in high memory).**
- **Normally, address register A7 is used as a main stack pointer (SP) in the 68000. Using this register for other addressing purposes may lead to incorrect execution.**
- **68000 main stack item size: One word for data. One long word for addresses.**
- **User-defined stacks that use other item sizes (byte, long word), may be created by using address registers other than A7.**

Initializing The Stack Pointer

- **It's the programmer's responsibility to initialize the stack. This involves two steps:**
  - **Initialize the stack pointer: The initial starting address or bottom of the stack.**
  - **Allocate sufficient memory for items to be pushed onto the stack. This could be done by locating the initial stack pointer at a very high memory address.**

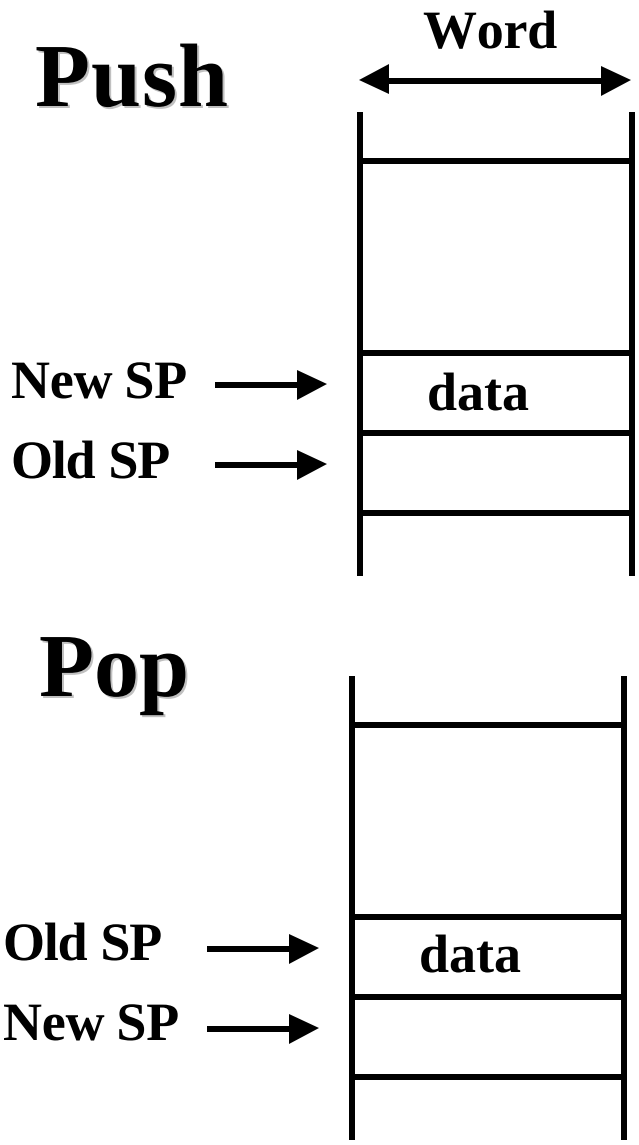
Example:

```
INITSP    EQU         $07FFE          Value of INITSP
          MOVEA.L     #INITSP,A7      Initialize  SP, A7
```

Or …

```
          MOVEA.L     #INITSP,SP      Initialize  SP
```

Stack Push, Pop Operations



To push an item onto the stack:

- The stack pointer must be decremented by one word (i.e decremented by 2)
- The word-sized information or data contained in the register or memory location addressed by <source> is put on the stack.

MOVE <source>,-(SP)

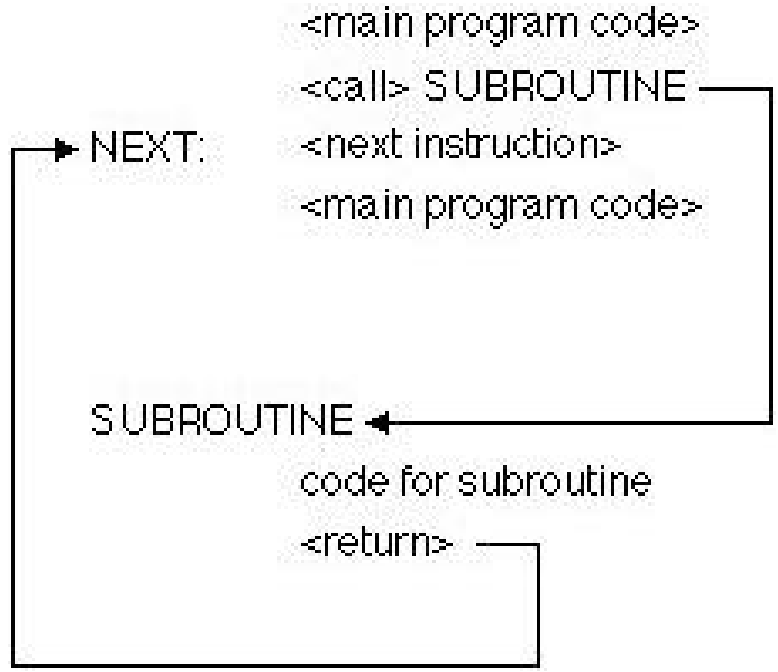
To pop an item off the stack:

- The information or data is read from the stack.
- The stack pointer incremented by one word
- The information is put into the register or memory location addressed by <destination>.

MOVE (SP)+,<destination>

Subroutines Basics

- **A subroutine is a sequence of, usually, consecutive instructions that carries out a single specific function or a number of related functions needed by calling programs.**
- **A subroutine can be called from one or more locations in a program.**
- **Subroutines may be used where the same set of instructions sequence would otherwise be repeated in several places in the program.**



- **Advantages of subroutines:**
  - **Make programs more human readable.**
  - **Simplify the code debugging process.**

68000 Subroutine Calling Instructions

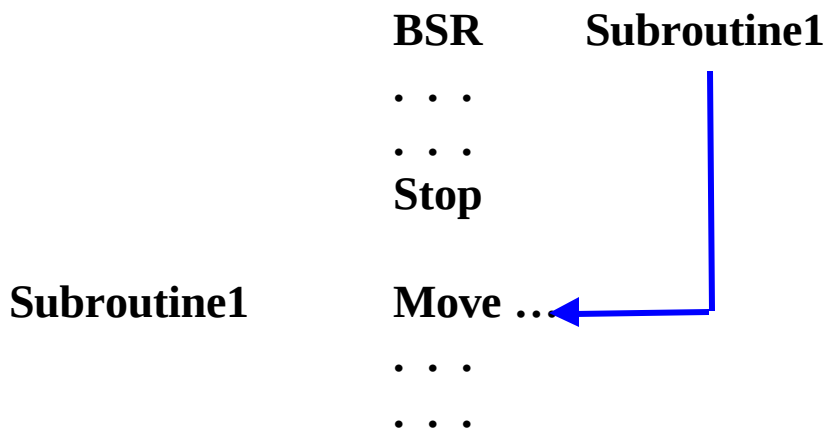
BSR <subroutine_label> Branch to SubRoutine

- **Where subroutine_label is the address label of the first instruction of the subroutine.**
- **subroutine_label must be within no more than a 16-bit signed offset, i.e. within plus or minus 32K of the BSR instruction.**
- **Does not affect CCR**

BSR does the following:

- **Pre-decrement the stack pointer by 4 (long word)**
- **Push the program counter, (PC) onto the stack.**
- **Load the program counter with the subroutine address**

Example:



68000 Subroutine Calling Instructions

JSR <EA> Jump to SubRoutine

- Similar in functionality to BSR, addressing mode <EA> must be a memory addressing mode.

  i.e. <EA> cannot be a data or address register.

- The advantages of this instruction:
  - A number of different addressing modes are supported.
  - The address of the subroutine can be determined dynamically at execution time

  Þ Allows the selection of the subroutine to call at runtime

- JSR does not affect CCR
- JSR is the most common form used for calling a subroutine.

68000 Subroutine Return Instruction

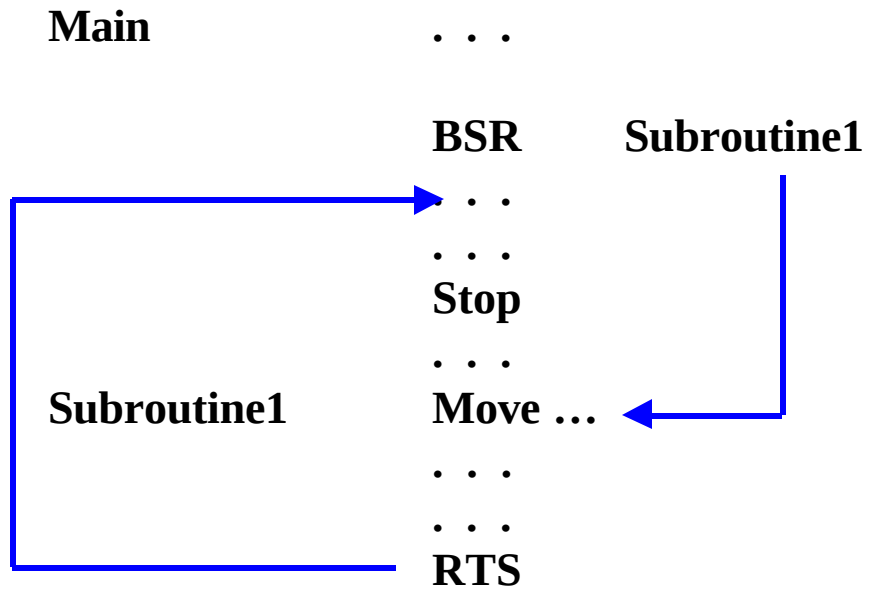
RTS ReTurn from Subroutine

- Pops the long word (return address) off of the top of the stack and puts it in the program counter in order to start executing after the point of the subroutine call.
- Post increments the stack pointer A7 by 4
- Equivalent to the instruction:

  JMP (SP)+

- Does not affect CCR

Example:



Passing Parameters to Subroutines

Parameters may be passed to a subroutine by using:

- **Memory locations:**
  - **This is similar to using static or global data in high level languages.**
  - **Does not produce position independent code and may produce unexpected side effects.**
- **Data and Address Registers:**
  - **Efficient, position-independent.**
  - **It reduces the number of registers available for use by the programmer.**
- **Stacks:**
  - **This is the standard, general-purpose approach for parameter passing. The LINK and UNLK instructions may be used to create and destroy temporary storage on the stack.**
  - **Similar to the approach used by several high-level languages including "C".**

Two Mechanisms for Passing Parameters

By Value:

- Actual value of the parameter is transferred to the subroutine .
- This is the safest approach unless the parameter needs to be updated.
- Not suitable for large amounts of data.
- In order to pass a parameter by value through the stack, one may use the instruction:

MOVE <EA>,-(SP)

By Reference:

- The address of the parameter is transferred.
- This is necessary if the parameter is to be changed.
- Recommended in the case of large data volume.
- In order to pass a parameter by reference through the stack, one may use the instruction:

PEA <EA>