Computer Organization & Architecture

Chapter 2 – Stacks & Subroutines

Zhang Yang 张杨

cszyang@scut.edu.cn

Autumn 2025

Content of this lecture

- 2.6 Stacks
- 2.7 Subroutines

Stacks (1)

- A *stack* is a list of data elements, usually words, with the accessing restriction that elements can be added or removed at one end of the list only.
- The structure is sometimes referred to as a *pushdown* stack or *last-in–first-out* (LIFO) stack.
- Push
  - Place a new item on the stack.
- Pop
  - Remove the top item from the stack.
- In modern computers, a stack is implemented by using a portion of the main memory.
- Programmer can create a stack in the memory.
- There is often a special processor stack as well.

Stacks (2)

- Processor Stack
  - Processor has stack pointer (SP) register that points to top of the processor stack.
  - Assume a byte-addressable memory with a 32-bit word length.
  - Push operation involves two instructions:

    ```
    Subtract    SP, SP, #4
    Store       Rj, (SP)
    ```

  - Pop operation also involves two instructions:

    ```
    Load        Rj, (SP)
    Add         SP, SP, #4
    ```

Stacks (3)

- Processor

Stack (ctd.)

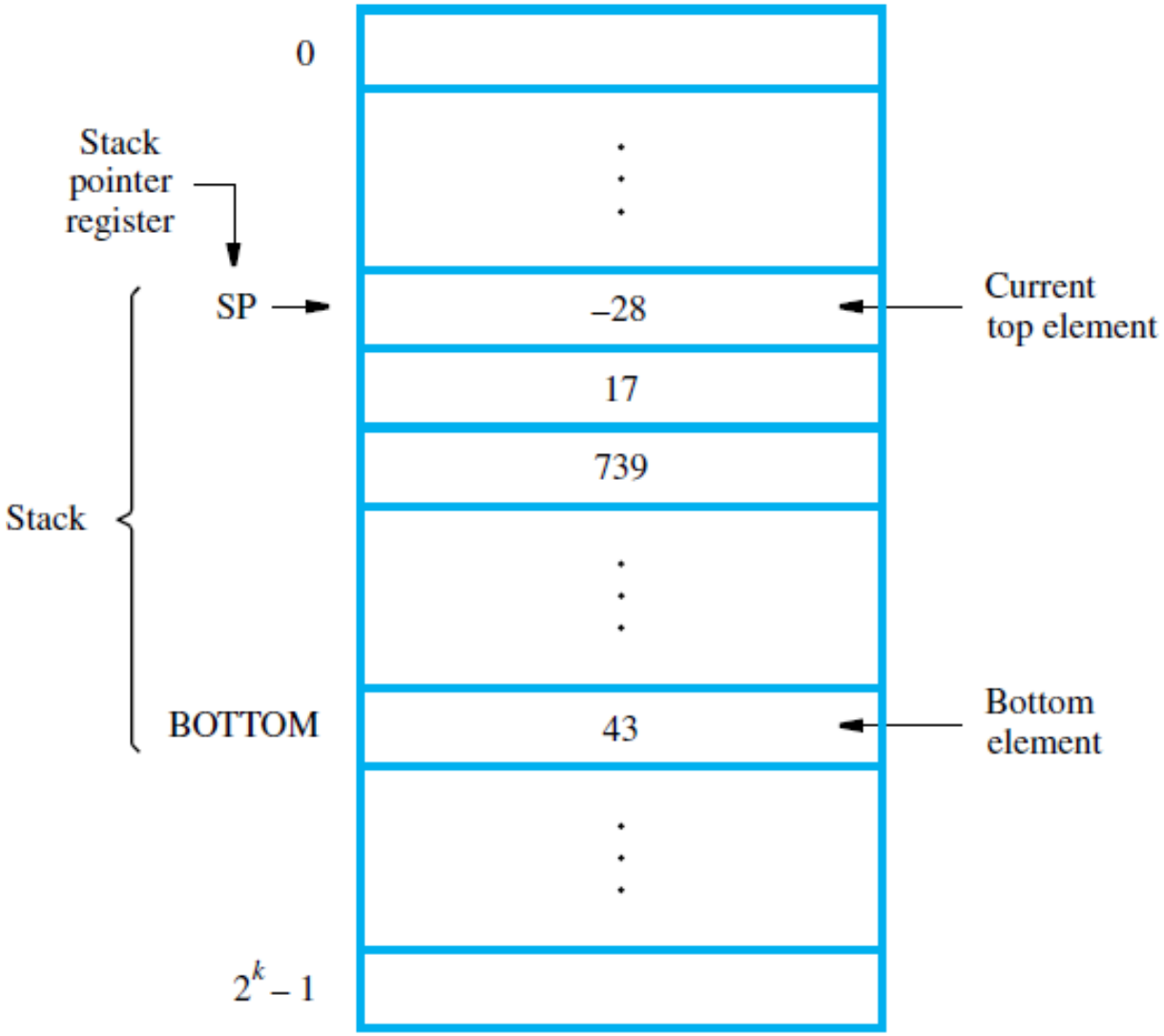

Figure 2.14 A stack of words in the memory.

Stacks (4)

- Processor Stack (ctd.)

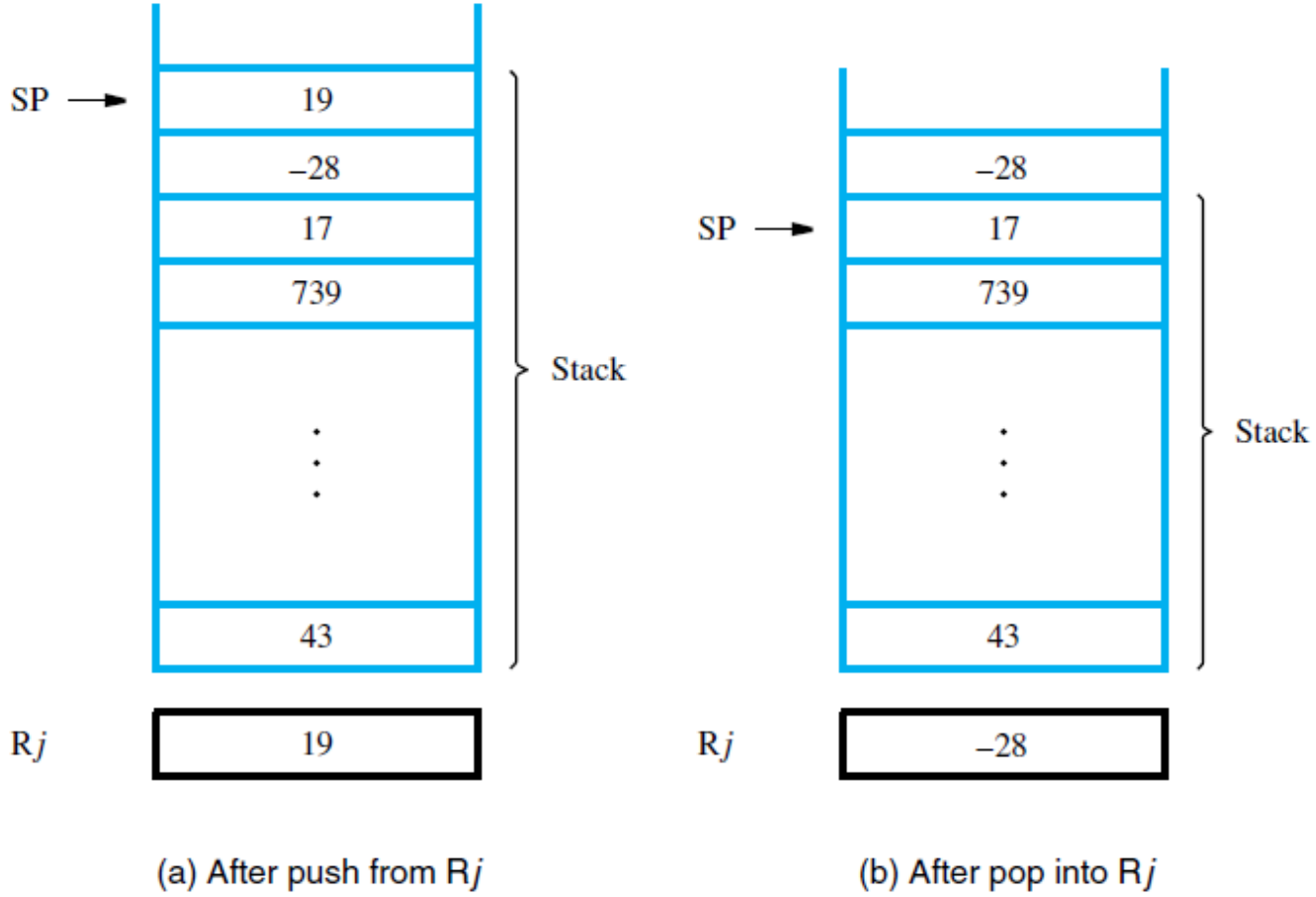

Figure 2.15 Effect of stack operations on the stack in Figure 2.14.

Subroutines (1)

- In a given program, a particular task may be executed many times using different data.
  - Examples: mathematical function, list sorting
- Implement task in one block of instructions.
  - This is called a subroutine.
- Rather than reproduce entire subroutine block in each part of program, use a subroutine call.
  - Special type of branch with Call instruction.

Subroutines (2)

- Branching to same block of instructions saves space in memory, but must branch back.
  - The subroutine must return to calling program after executing last instruction in subroutine.
  - This branch is done with a Return instruction.
- Subroutine Linkage
  - Subroutine can be called from different places.
  - How can return be done to correct place?
    - This is the issue of subroutine linkage.

Subroutines (3)

- Subroutine Linkage (ctd.)
  - During execution of Call instruction, PC is updated to point to instruction after Call.
  - Save this address for Return instruction to use.
  - Simplest method: place address in link register.
  - Call instruction performs two operations:
    - Store updated PC contents in link register,
    - Then branch to target (subroutine) address.
  - Return just branches to address in link register.

Subroutines (4)

- Subroutine Linkage (ctd.)

Memory location	Calling program	Memory location	Subroutine SUB
	⋮		
200	Call SUB	1000	first instruction
204	next instruction		⋮
	⋮		Return

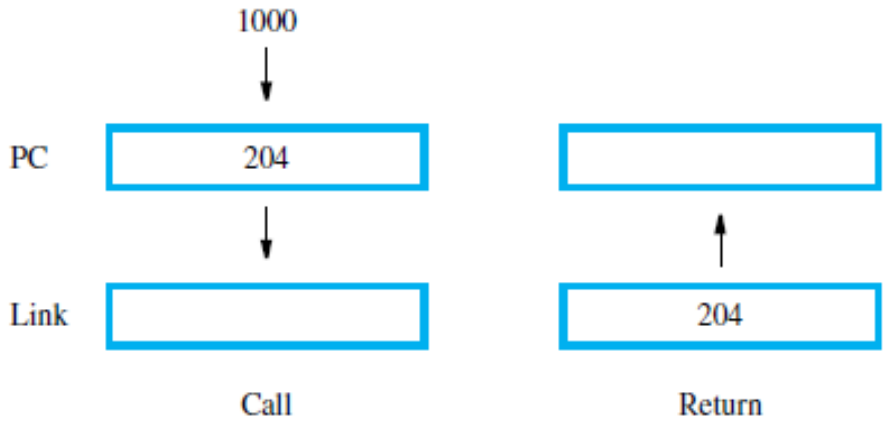

Figure 2.16 Subroutine linkage using a link register.

Subroutines (5)

- Subroutine Nesting & The Processor Stack
  - We can permit one subroutine to call another, which results in subroutine nesting.
  - Link register contents after first subroutine call are overwritten after second subroutine call.
  - First subroutine should save link register on the processor stack before second call.
  - After return from second subroutine, first subroutine restores link register.
  - Subroutine nesting can be carried out to any depth.
  - Return addresses are generated and used in a last-in–first-out order. This suggests that the return addresses associated with subroutine calls should be pushed onto the processor stack.