The Stack and Procedures

Computer Organization and Assembly Language

Contents

- The Stack
- Stack Operations
- Stack Application
- Procedures

Overview

- The stack segment of a program is used for temporary storage of data and addresses.
- A *stack* is one-dimensional data structure.
- Items are added and removed from one end of the structure; that is, it is processed in a "last-in, first-out" manner.
- The most recent addition to the stack is called the top of the stack.

Stack

- A program must set aside a block of memory to hold the stack.
- We declared a stack segment in the code as,
- .STACK 100H
- When the program is loaded in memory, SS will contain the segment number of the stack segment.
- For the preceding stack declaration, SP (stack pointer) is initialized to 100h, which represents empty stack.
- When the stack is not empty, SP contains the offset address of the top of the stack.

PUSH and PUSHF

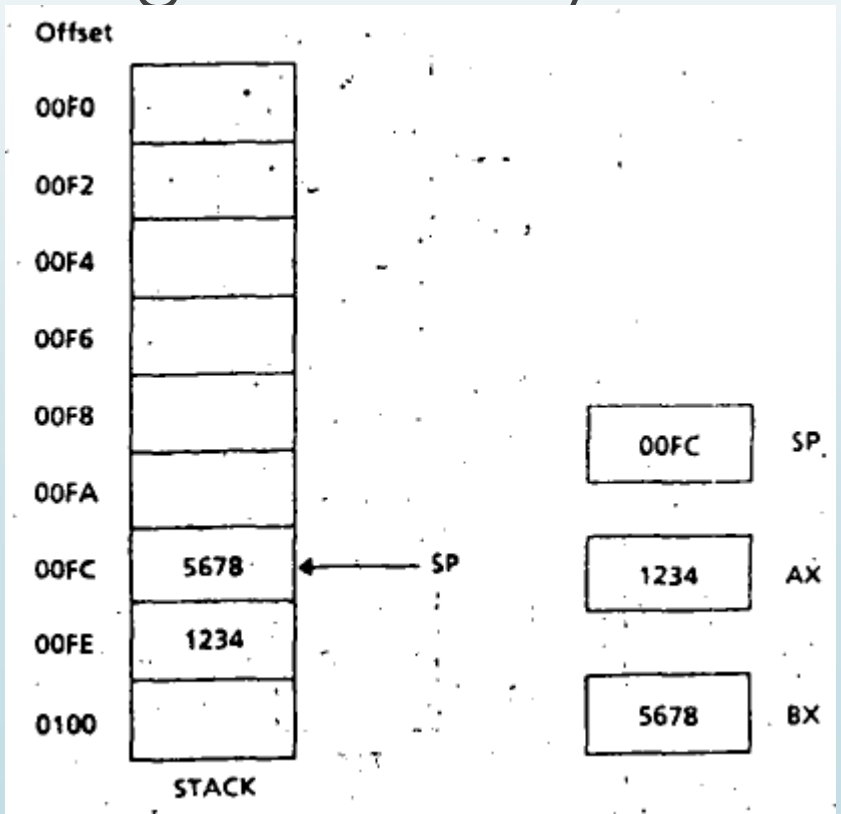
- To add a new data to the stack we PUSH it on stack as
- PUSH source
- where source is a 16-bit register or memory word. For example,
- PUSH AX
- PUSH execution does the following:
- 1. SP is decreased by 2
- 2. A copy of the source content is moved to the address specified by SS:SP.
- The instruction PUSHF, which has no operands, pushes the contents of the FLAGS register onto the stack.

SP Register

- Initially, SP contains the offset address of the memory location immediately following the stack segment; the first PUSH decreases SP by 2, making it point to the last word in the stack segment.

SP After PUSH Operations

- Since a PUSH decreases SP, the stack grows toward the beginning of memory.



POP and POPF

- POP removes the top item from the stack as:
- POP destination
- where destination is a 16-bit register (except IP) or memory word. For example,
- POP BX
- Executing POP causes this to happen:
- 1. The content of SS:SP (the top of the stack) is moved to the destination.
- 2. SP is Increased by 2.

POP and POPF

- The Instruction POPF pops the top of the stack into the FLAGS register.
- There is no effect of PUSH, PUSHF, POP, POPF on the flags.
- PUSH and POP are word operations, so a byte Instruction such as

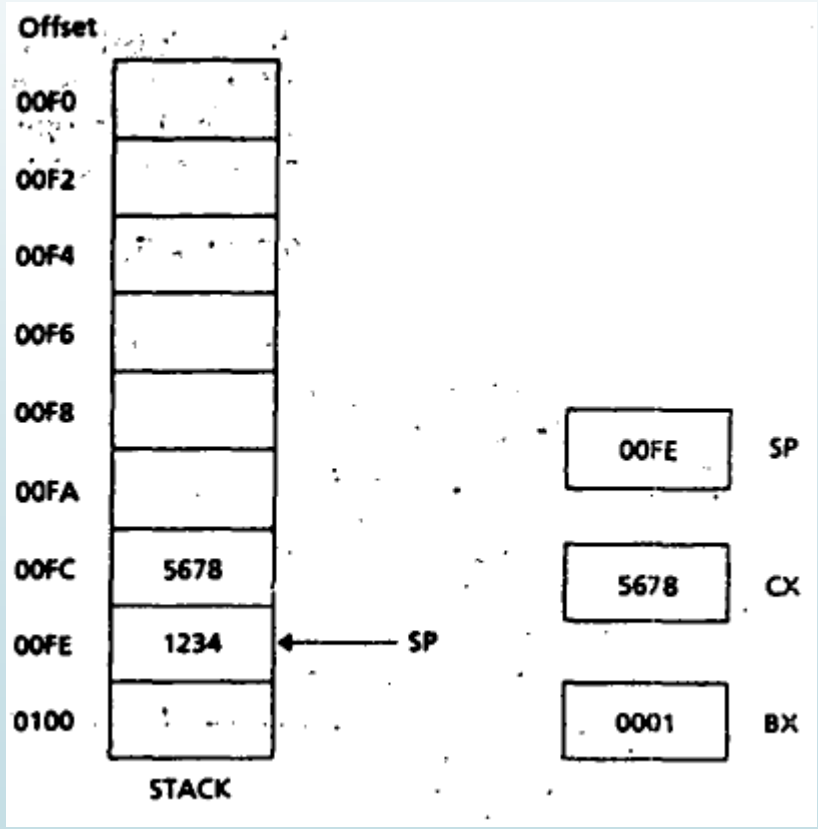
  PUSH DL

- is illegal.
- So is a push of immediate data illegal in 8086, such as

  PUSH 2

Stack After POP CX

- The figure below shows stack after *POP CX*



OS and Stack

- The operating system also uses the stack.
- For example, to implement the INT 21h functions, DOS saves any registers it uses on the stack and restores them when the interrupt routine is completed.
- This does not cause a problem for the user since any values DOS pushes onto the stack are popped off by DOS before it returns control to the program.

Procedures

- To solve a problem, it is decompose into subproblems that are easier to solve than the original problem.
- Procedures are used to solve these subproblems.
- One of the procedures is the main procedure, which is entry point to the program.
- The main procedure can call other procedures.
- The procedures can also call each other, or call itself (recursion).

Procedures

- When one procedure calls another, control transfers to the called procedure and its instructions are executed;
- The called procedure returns control to the caller at the next instruction after the call statement.

Procedure Syntax

- The syntax of procedure declaration is:

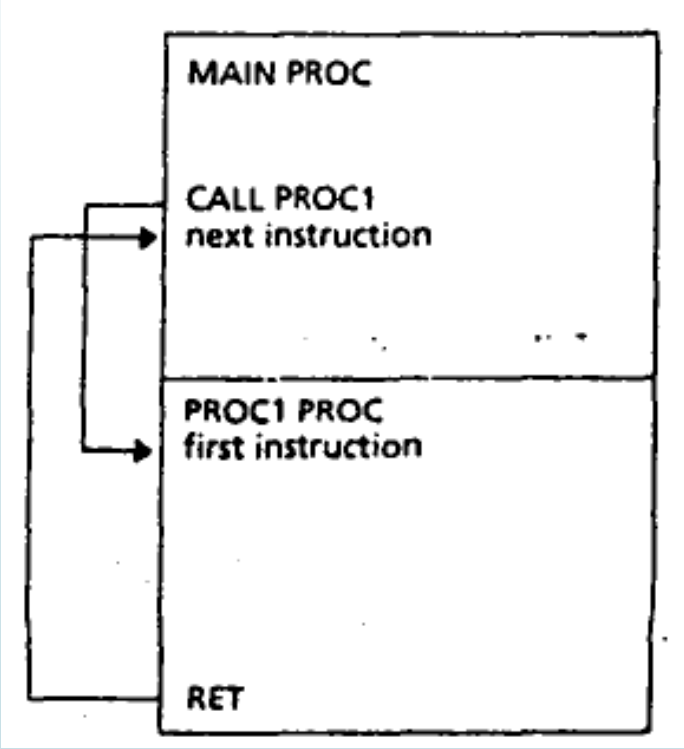
```
name PROC type
    ;body of the procedure
    RET
name ENDP
```

Procedure Syntax

- Name is the user-defined name of the procedure.
- The optional ***type*** is NEAR or FAR (if omitted, NEAR is assumed).
- NEAR means that the statement that calls the procedure is in the same segment as the procedure itself.
- FAR means that the calling statement is in a different segment.

Procedure Call

- The operating system also uses the stack.



RET Statement

- The **RET** (return) instruction causes control to transfer back to the calling procedure.
- Every procedure (except the main procedure) should have a RET, usually in the end.

Passing Data Between Procedures

- A procedure must have a way to receive values from the procedure that calls it, and a way to return results.
- Unlike high-level language procedures, assembly language procedures do not have parameter lists, so Registers and the Stack can be used for parameters and return values.

CALL and RET

- To invoke a procedure, **CALL** instruction is used.
- There are two kinds of procedure calls, direct and indirect.
- The syntax of a direct procedure call is

  CALL name
- where name is the name of a procedure.
- The syntax of an indirect call is

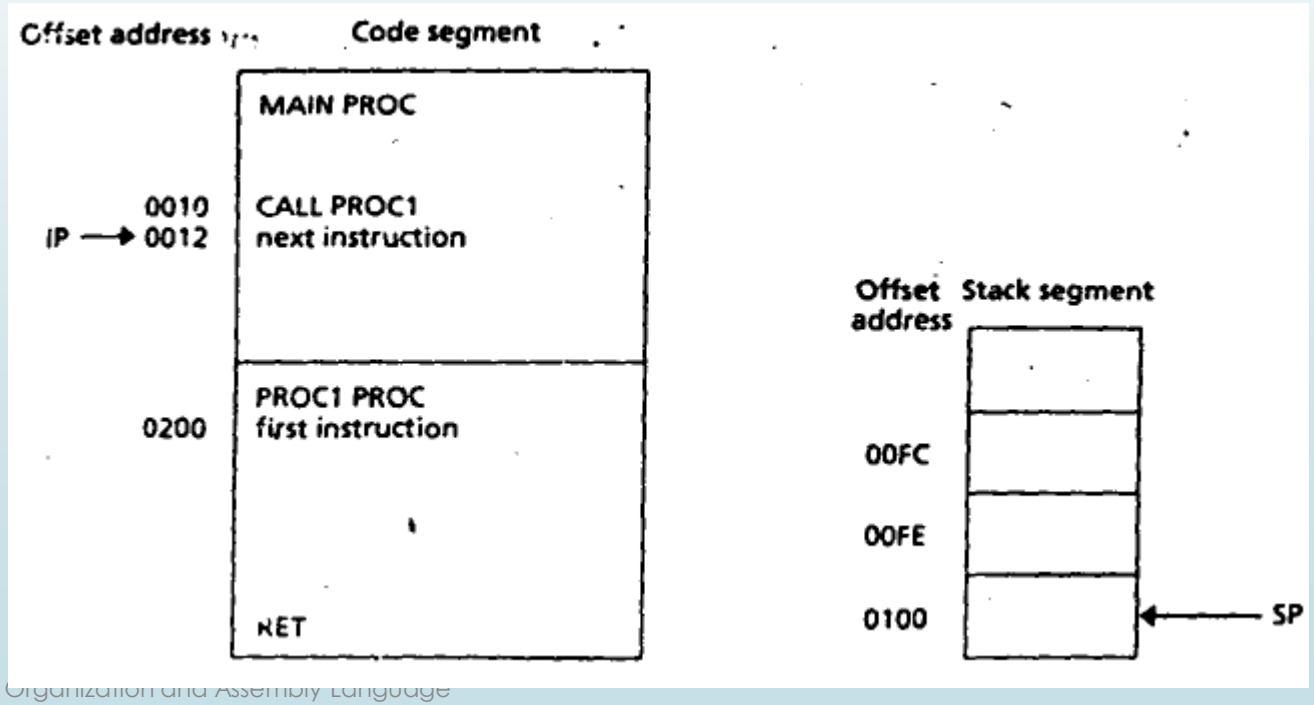
  CALL address_expression
- where address_expression specifics a register or memory location containing address of a procedure.

Executing a CALL instruction

- The return address to the calling program Is saved on the stack.
- This is the offset of the next instruction after the CALL statement.
- The segment:offset of this instruction is in CS:IP at the time the call is executed.
- IP register gets the offset address of the first instruction of the procedure.
  - This transfers control to the procedure.

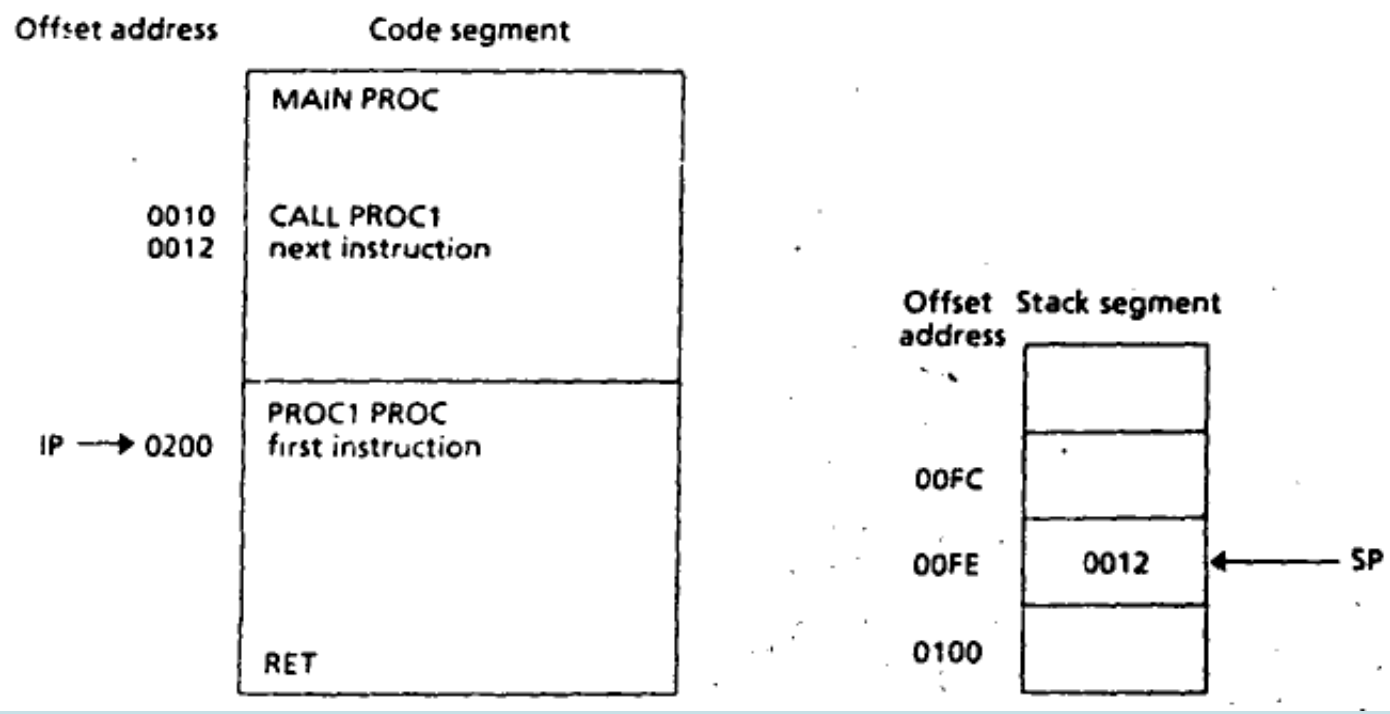
Procedure Call and the Stack

- IP Register and the Stack before procedure call.



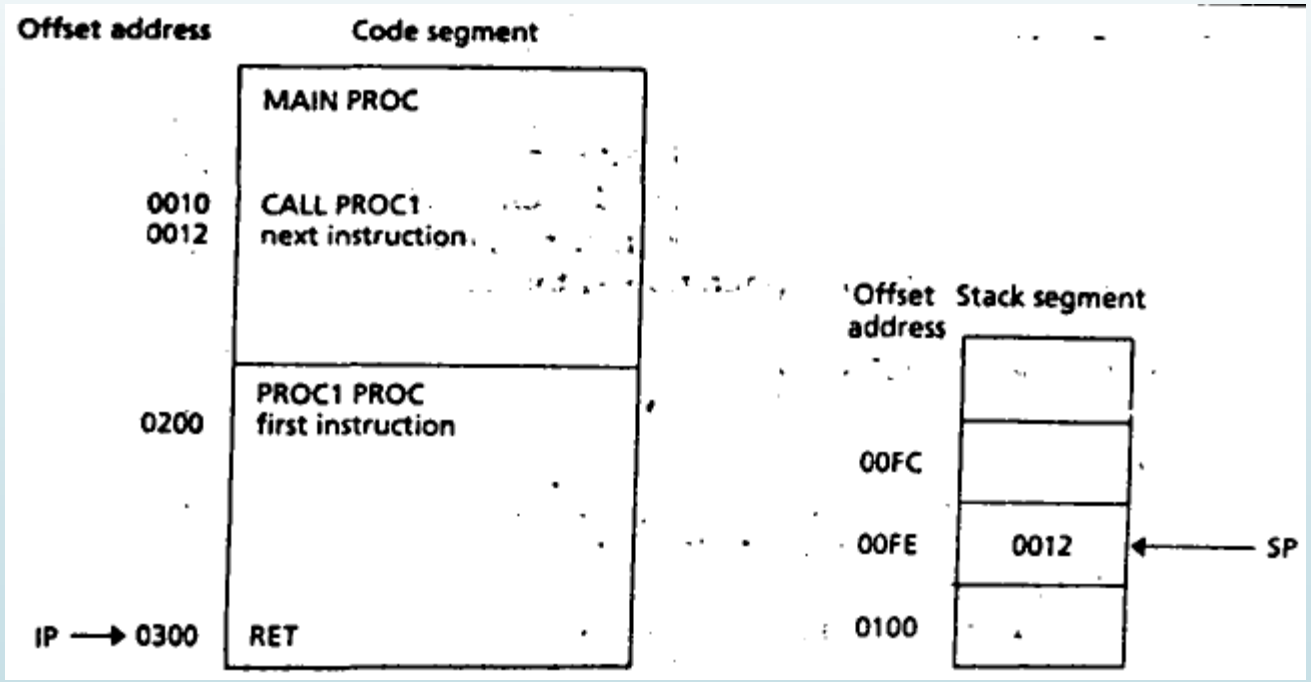
Procedure Call and the Stack

- IP Register and the Stack after procedure call.



Procedure Call and the Stack

- IP Register and the Stack before RET statement.



Procedure Call and the Stack

- IP Register and the Stack after RET statement.