Stack Operations

Muhammad Afzaal
m.afzaal@nu.edu.pk

Book Chapter

- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 5
  - Section 5.4

Stack

- A LIFO (Last In First Out) data structure
- New value is added to the top of stack
- Existing values are removed from the top of stack
- An essential part of calling from and returning to the procedures
- Real life example
  - A stack of plates

Runtime Stack (1/3)

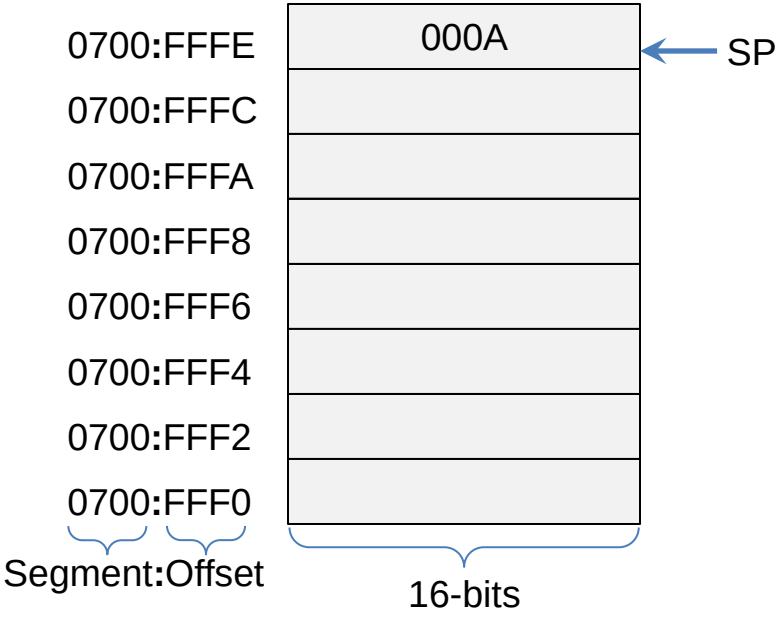
- A memory array managed by CPU using `ESP/SP` (Extended Stack Pointer) register and `SS` (Stack Segment)
- `ESP/SP` always points to the last value pushed on the top of stack and holds offset of that value in Stack Segment
- `ESP/SP` cannot be manipulated directly instead it can be modified indirectly by instructions such as `PUSH`, `POP`, `CALL`, `RET`

Runtime Stack (2/3)

- In protected mode i.e. 32-bit mode, size of each stack location is 32-bits
- In real-address mode i.e. 16-bit mode, size of each stack location is 16-bits
- emu8086 uses real-address mode
- Runtime Stack is different from Stack Abstract Data Type which is typically written in a HLL

Runtime Stack (3/3)

- SS contains the base address of Stack Segment
- SP contains the offset of value at the top of stack

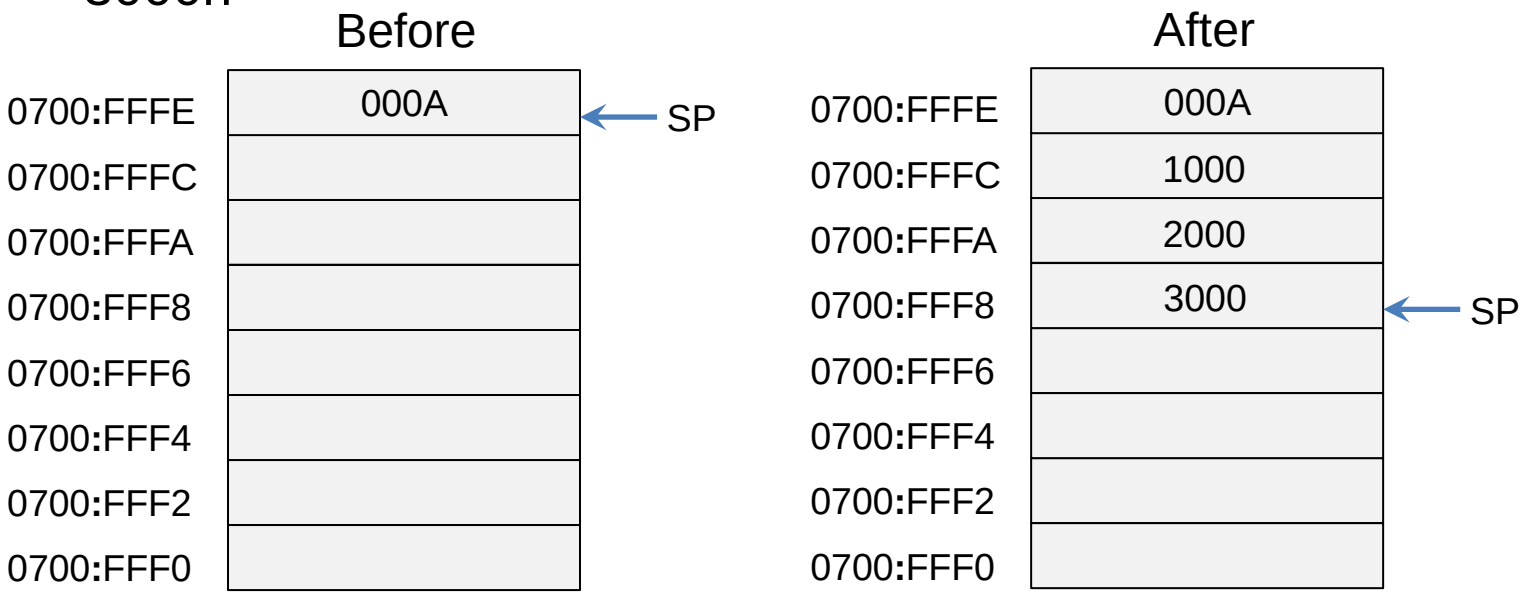


Push Operation (1/3)

- A push operation in stack puts the value on the top location available in stack and decrements the stack pointer by size of stack element
- Size of each stack element is 32 bits in protected address mode
- Size of each stack element is 16 bits in real-address mode

Push Operation (2/3)

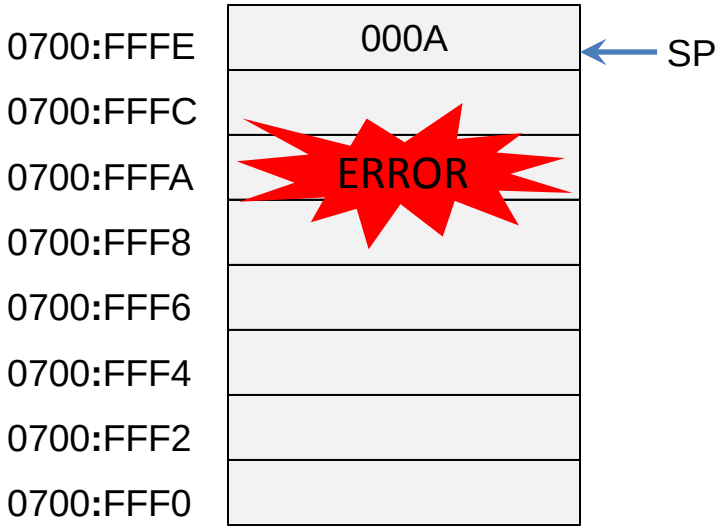
- SP is decremented by 2 with each push operation
- These values are pushed on stack
  - 1000h
  - 2000h
  - 3000h



Push Operation (3/3)

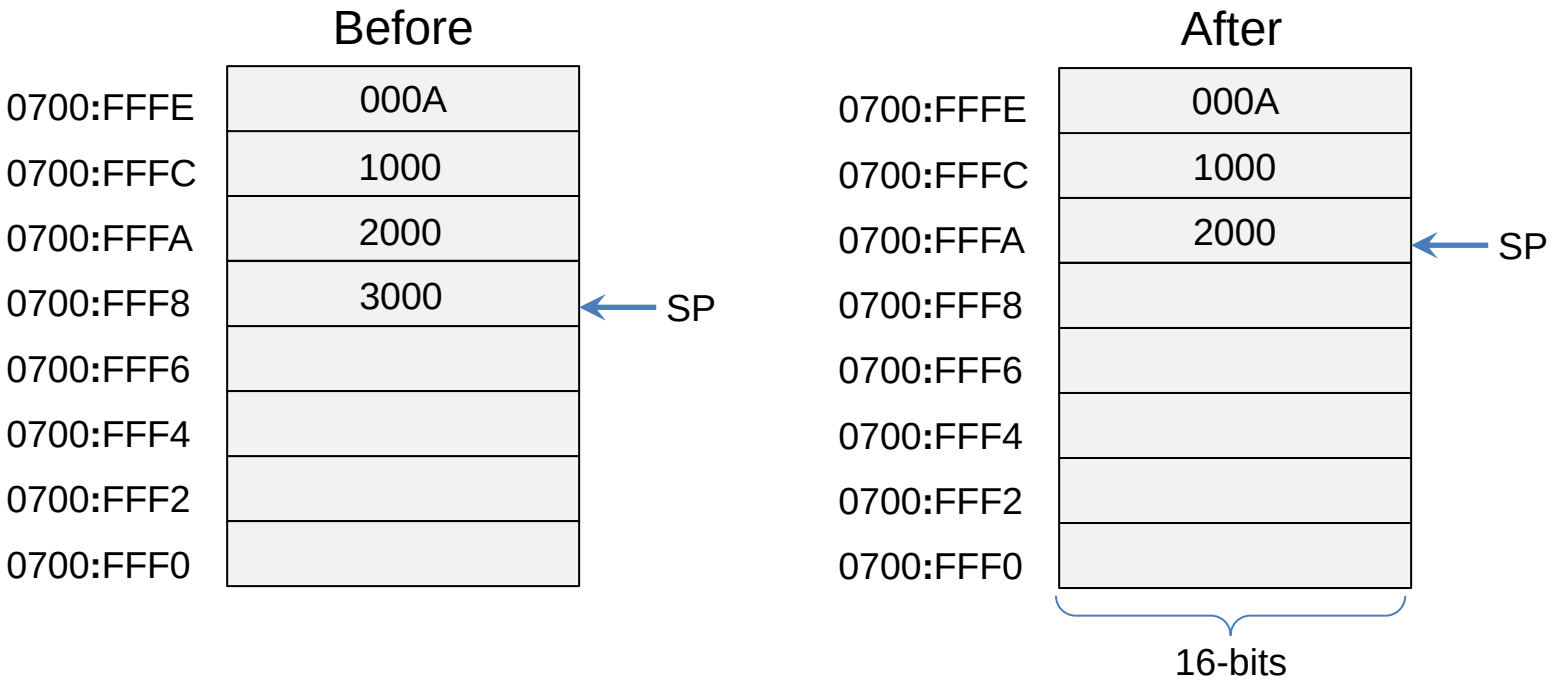
- This value is pushed on stack

  1000 0000h



Pop Operation

- Removes value from top of stack
- SP is incremented by stack element size with each pop operation



PUSH Instruction

- `PUSH` instruction is executed in two steps
  - First decrements SP by the size of stack element
  - Then copies the source operand on top of stack
- `PUSH` instruction formats
  - `PUSH reg/mem16` → contents of 16-bit register or 16-bit memory location is pushed on stack
  - `PUSH imm16` → 16-bit immediate value is pushed on stack
- Examples are
  - `PUSH AX`
  - `PUSH 10h`

POP Instruction

- `POP` instruction is executed in two steps
  - First the contents of stack element pointed to by SP are copied into destination operand
  - Then SP is incremented by the size of stack element
- Only one `POP` instruction formats
  - `POP reg/mem16` → copies the value pointed to by SP into 16-bit register or 16-bit memory location
- Examples are
  - `POP AX`
  - `POP var` ;where var is a 16-bit memory location

PUSHF and POPF Instructions

- `PUSHF` is used to push EFLAGS register on the stack
- `POPF` pops the stack into EFLAGS register
- When using these instructions, make sure program's execution path does not skip over `POPF` instruction
- Syntax is
  - `PUSHF`
  - `POPF`

PUSHA and POPA Instructions

- `PUSHA` instruction pushes all 16-bit general purpose register on the stack in given order
  - `AX, CX, DX, BX, SP, BP, SI, DI`
- `POPA` instruction pops the same registers in the revers order
- Useful when modifying many general purpose registers inside a procedure

Stack Applications

- Registers can be saved temporarily when used for more than one purpose
- When `CALL` instruction executed, return address is saved on the stack
- Arguments are passed to a subroutine by pushing them on the stack
- Stack can be used as temporary storage for local variables inside a subroutine