Microprocessor and Micro-controller

Stack and it's Operations

Professor Dr. Md. Rabiul Islam
Dept. of Computer Science & Engineering
Rajshahi University of Engineering & Technology
Rajshahi-6204, Bangladesh.
rabiul_cse@yahoo.com

Outline

- ❑ Overview of the operation of stack
- ❑ Declaration of stack in Assembly Language program
- ❑ Memory mapping with segment and offset for stack declaration
- ❑ Operation of PUSH instruction
- ❑ Operation of POP instruction
- ❑ Examples of Executing PUSH and POP Instructions
- ❑ Operation of FLAG register with stack
  - ➢ Reasons of using FLAG register in stack
  - ➢ Operation of PUSHF and POPF instructions

References

Chapter 8 Yutha Yu and Charles Marut, “Assembly Language Programming and Organization of the IBM PC", McGraw-Hill International Edition, 1992.

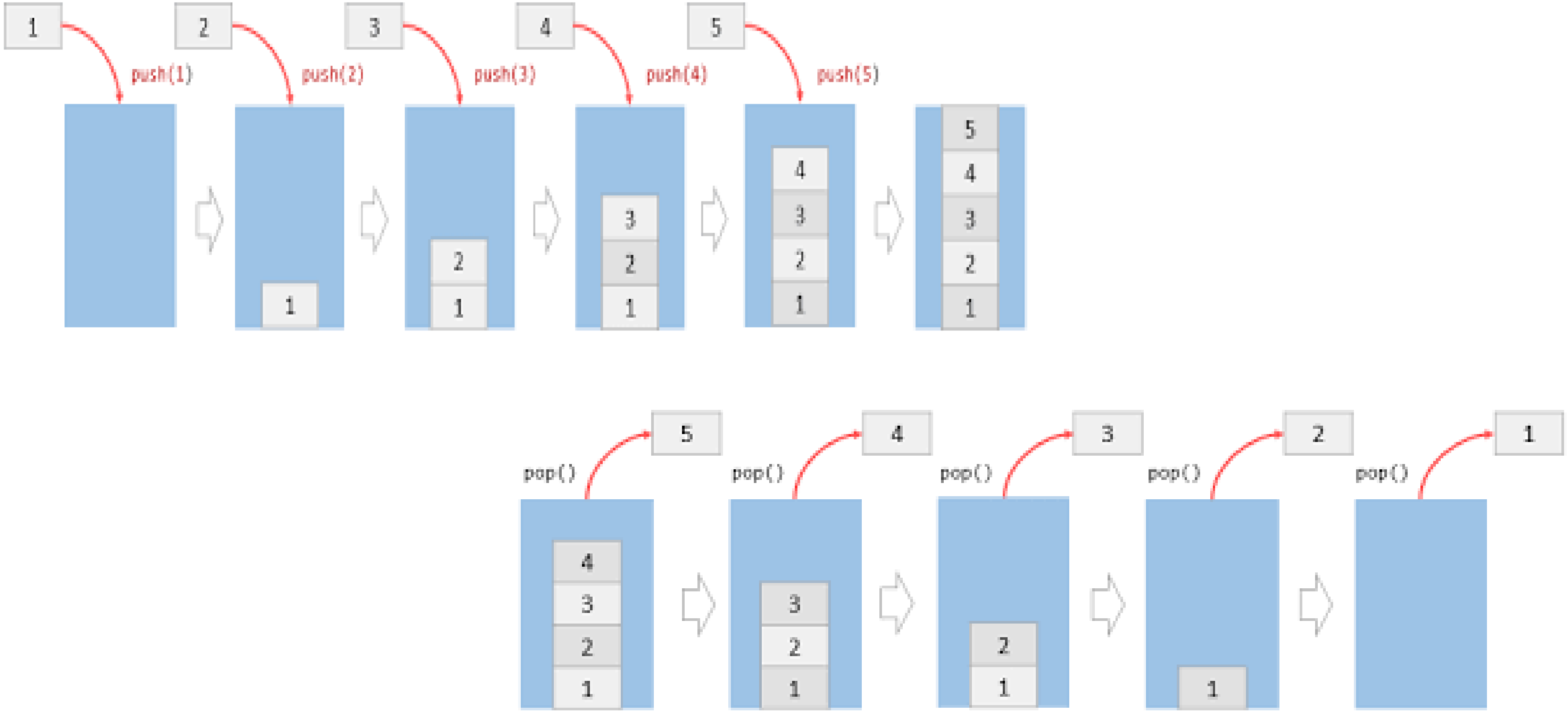
Overview of the Operation of Stack

✓ The stack segment of a program is used for temporary storage of data and addresses

✓ A stack is a one-dimensional data structure

✓ Items are added to and removed from one end of the structure using a "Last In First Out" technique (LIFO)

✓ The top of the stack is the last addition to the stack



Declaration of Stack in Assembly Language Program

✓ The purpose of the stack segment declaration is to set aside a block of memory (the stack area) to store the stack.

✓A program must set aside a block of memory to hold the stack. The stack area should be big enough to contain the stack at its maximum size. We have been doing this by declaring a stack segment; for example,

STACK l00H

✓ The statement .STACK 100H in the program sets aside a block of 100 bytes of memory to hold the stack.

✓ If size is omitted, by default 1kB is set aside for the stack.

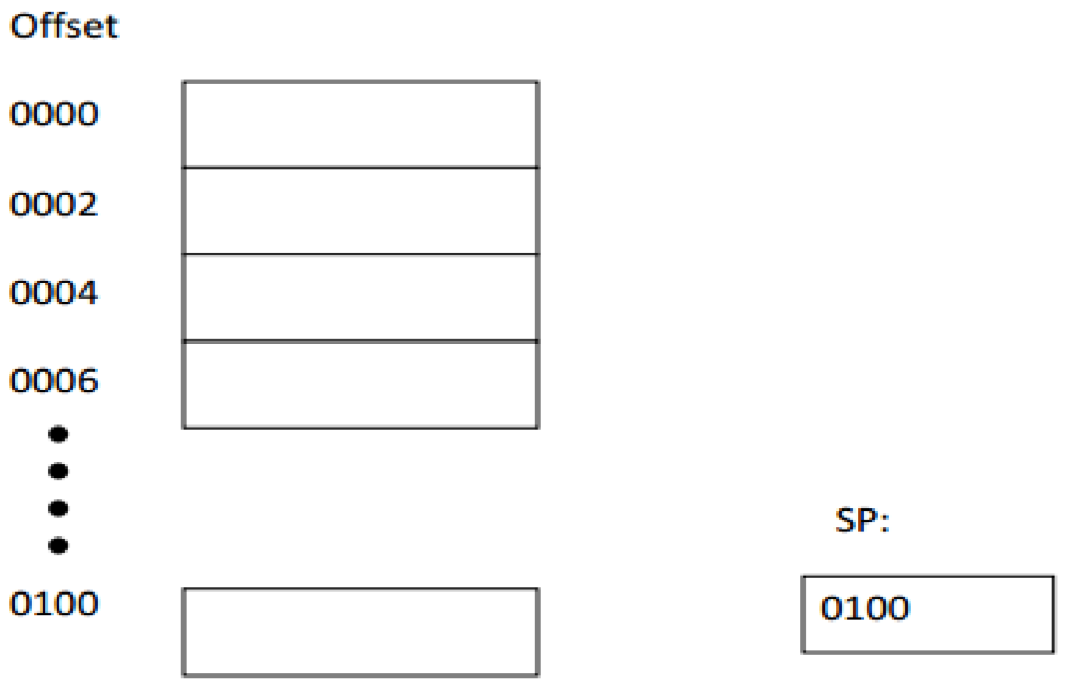
Memory Mapping with Segment and Offset for Stack Declaration

✓ The SS (Stack Segment Register) contains the segment number of the stack segment

✓ The complete segment:offset address to access the stack is SS:SP

✓Initially before any data or addresses have been placed on the stack, the SP contains the offset address of the memory location immediately following the stack segment.

Empty Stack



(Beyond the end of the stack)

Operation of PUSH Instruction

✓ PUSH instruction adds a new word to the stack

✓ SYNTAX: PUSH source

where source is a 16-bit register or memory word

✓ PUSH instruction causes

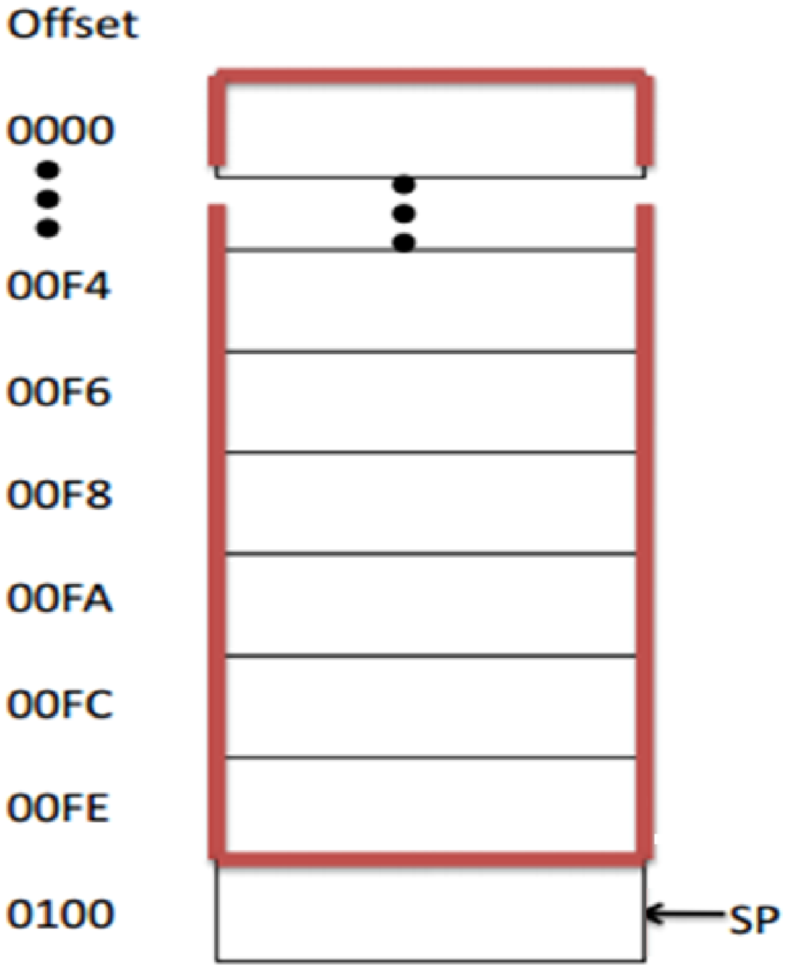
- ➢ The stack pointer (SP) to be decreased by 2.
- ➢ Then a copy of the value in the source field is placed in the address specified by SS:SP.

✓ Initially SP points to a location immediately following the stack. The first push decreases SP by 2, making it point to the last word in the stack

✓ Because each PUSH decreases the SP, the stack is filled a word at a time backwards from the last available word in the stack toward the beginning of the stack.

Operation of PUSH Instruction

How words are added to stack?



(Beyond the end of the stack)

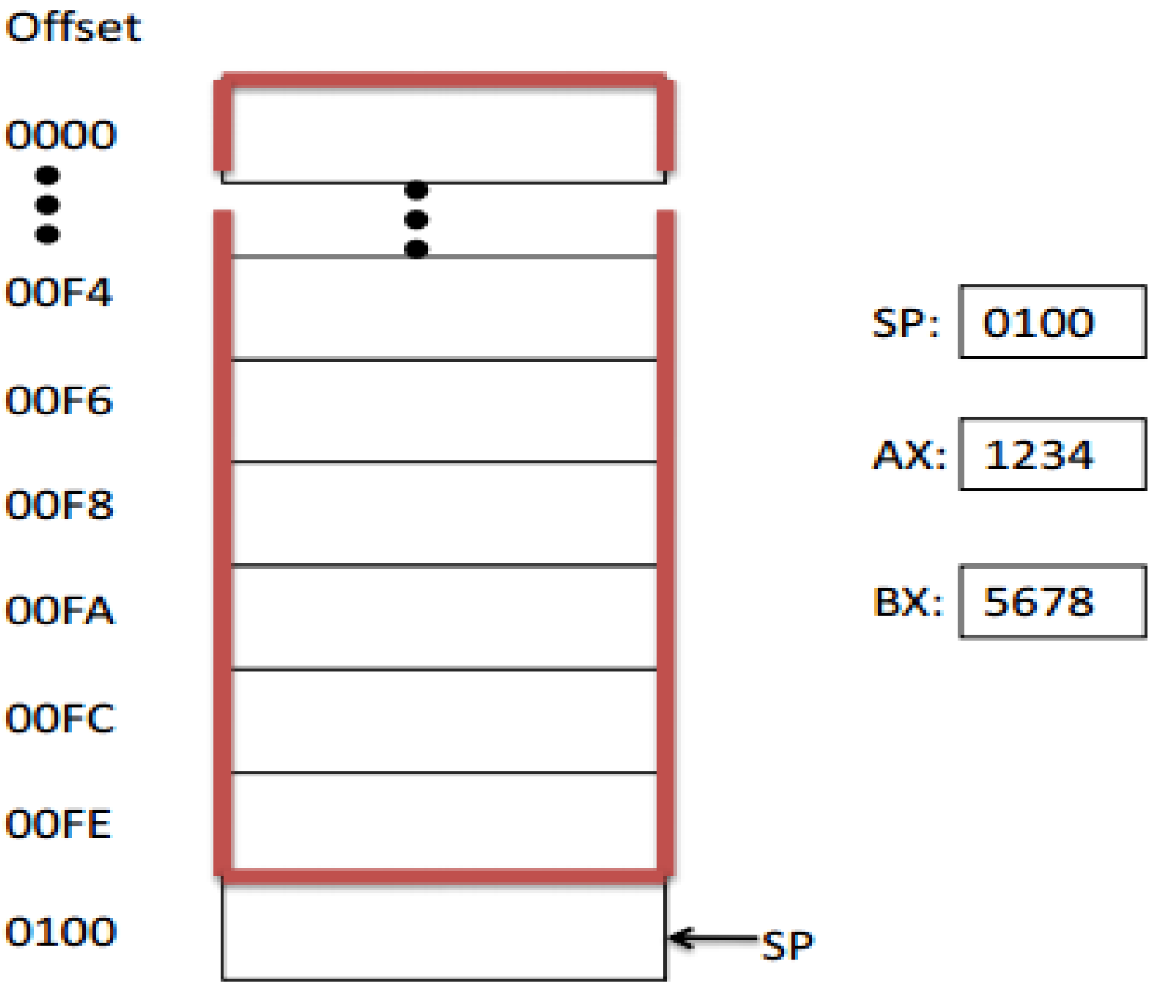
SP:

It stack is empty, SP has a value of 100h; otherwise it has a value between 0000-00FEh

Operation of PUSH Instruction

How words are added to stack?

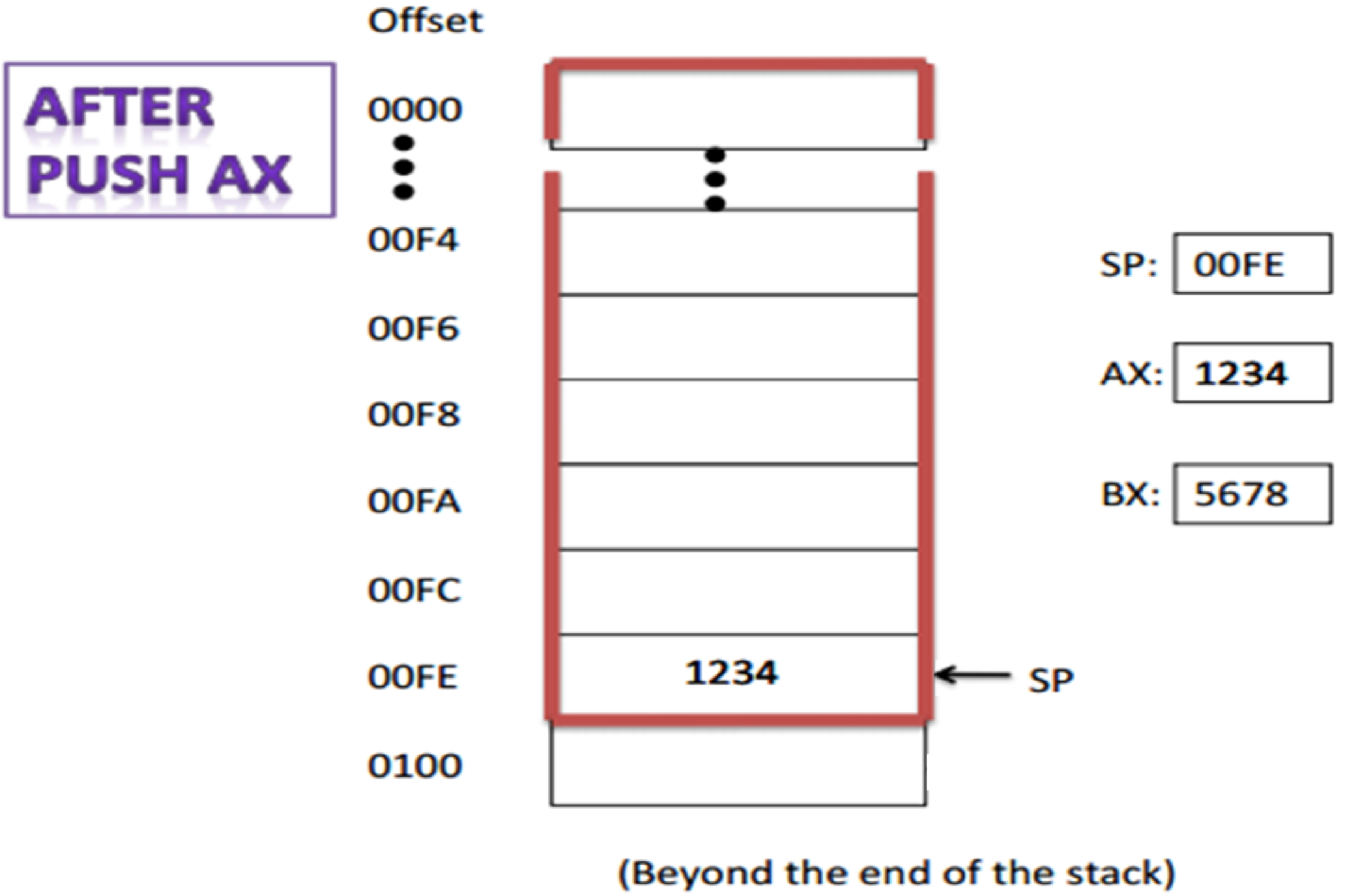
EMPTY STACK



(Beyond the end of the stack)

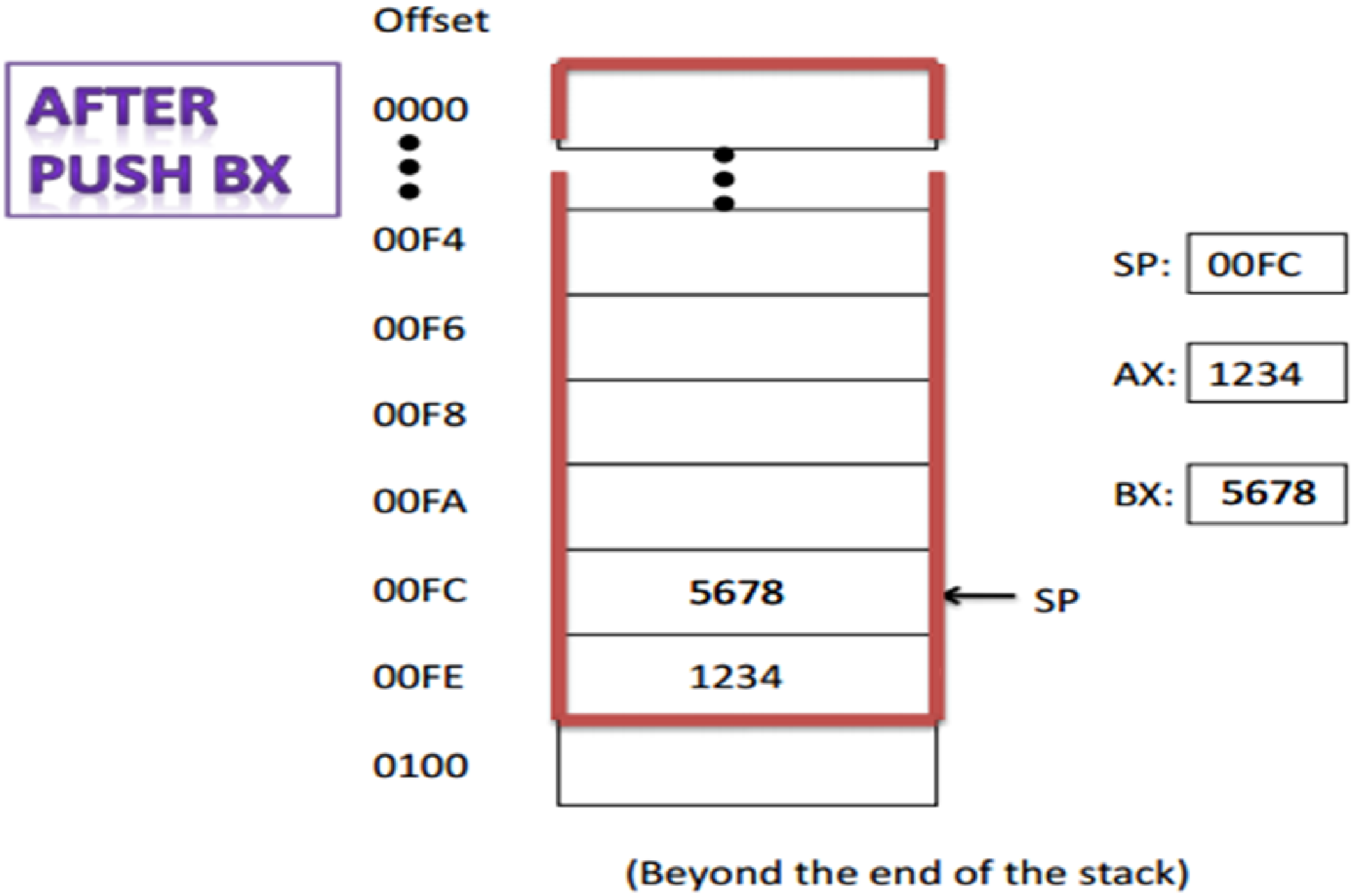
Operation of PUSH Instruction

How words are added to stack?



Operation of PUSH Instruction

How words are added to stack?



Operation of POP Instruction

- ✓ POP instruction removes the last word placed on the stack
- ✓ SYNTAX: POP destination

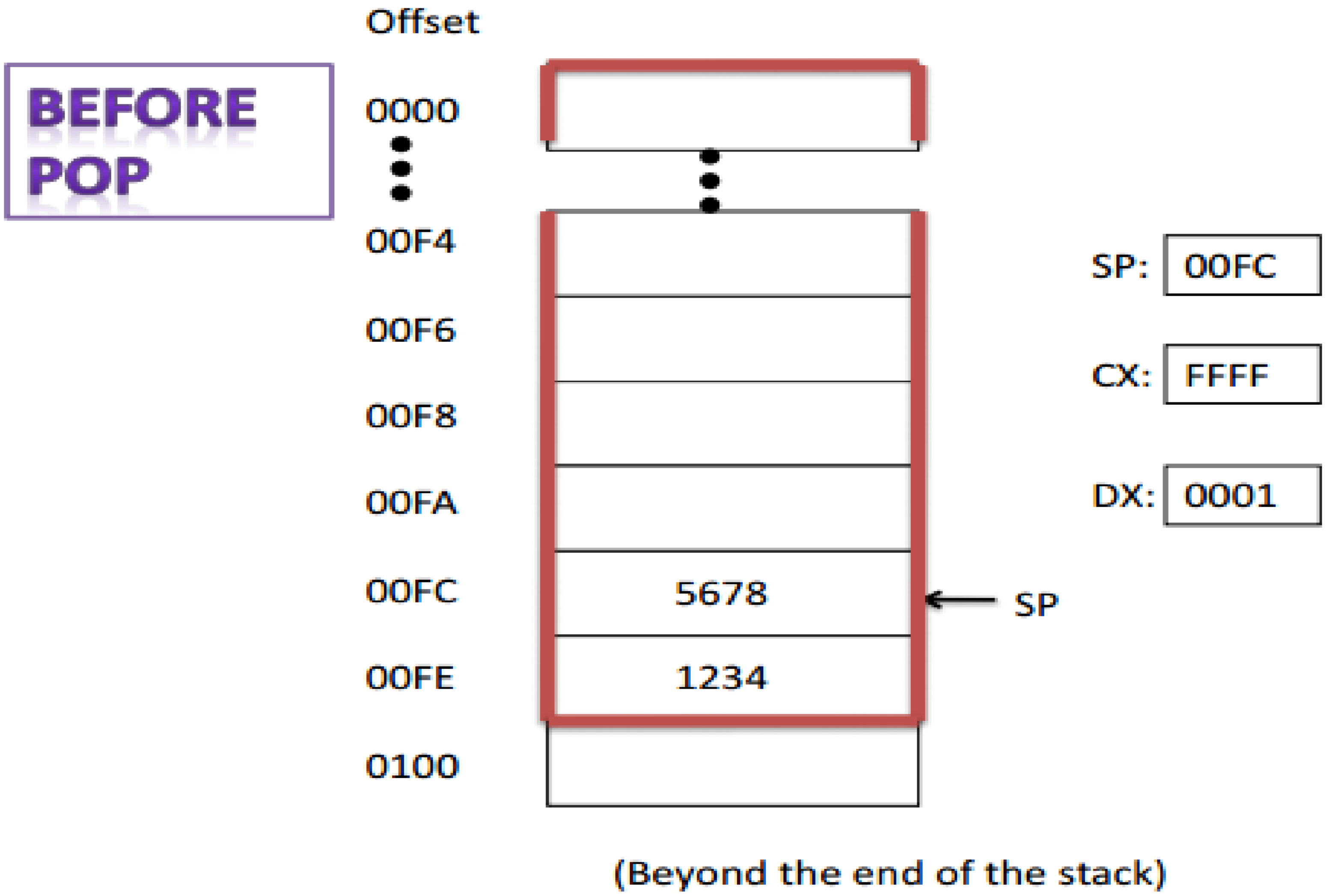
  where destination is a 16-bit register or memory word
- ✓ POP instruction causes
  - ➢ The contents of SS:SP to be moved to the destination field
  - ➢ It increases the stack pointer (SP) by 2

Restrictions:

1. PUSH and POP work only with words
2. Byte and immediate data operands are illegal

Operation of POP Instruction

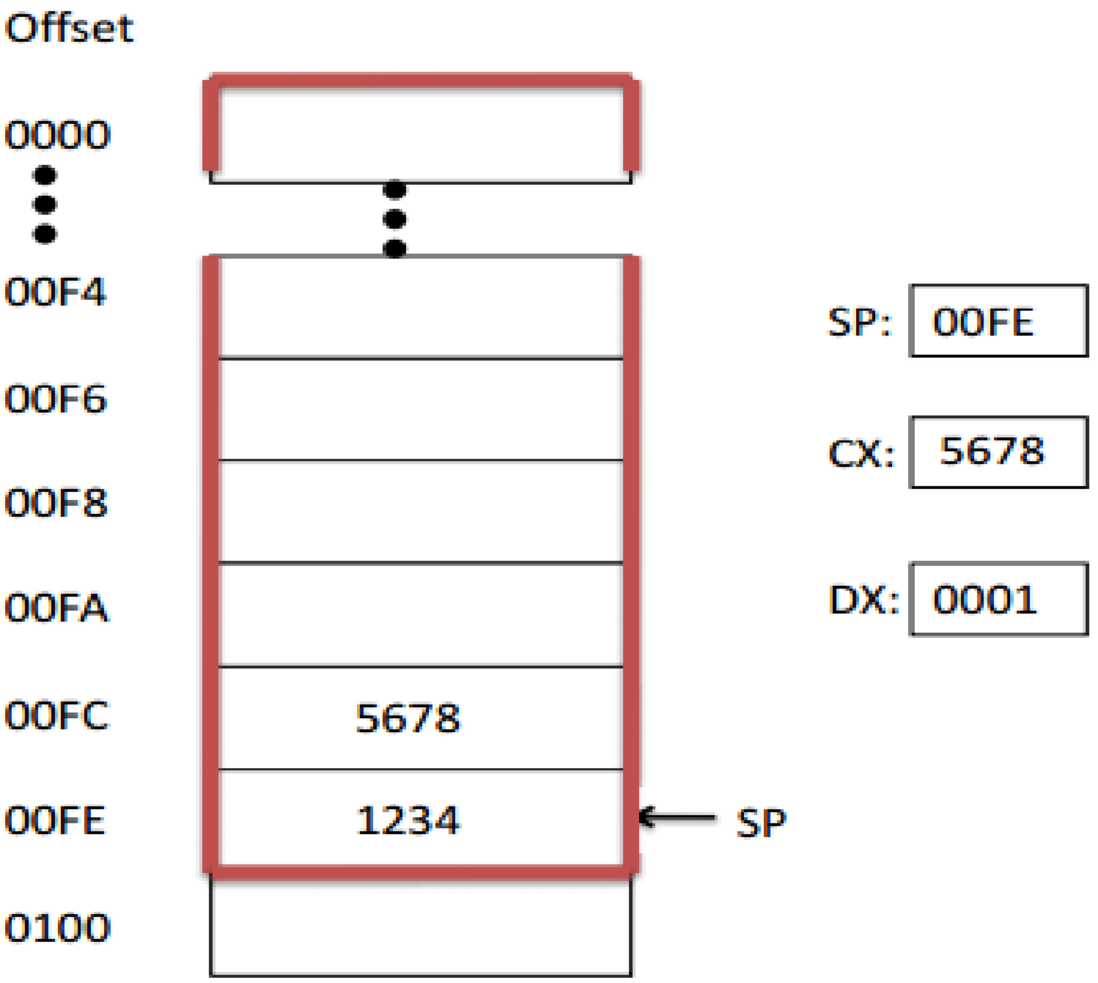
How words are removed from stack?



Operation of POP Instruction

How words are removed from stack?

AFTER POP CX



Offset	
0000	
⋮	⋮
00F4	
00F6	
00F8	
00FA	
00FC	5678
00FE	1234 ← SP
0100	

SP: 00FE

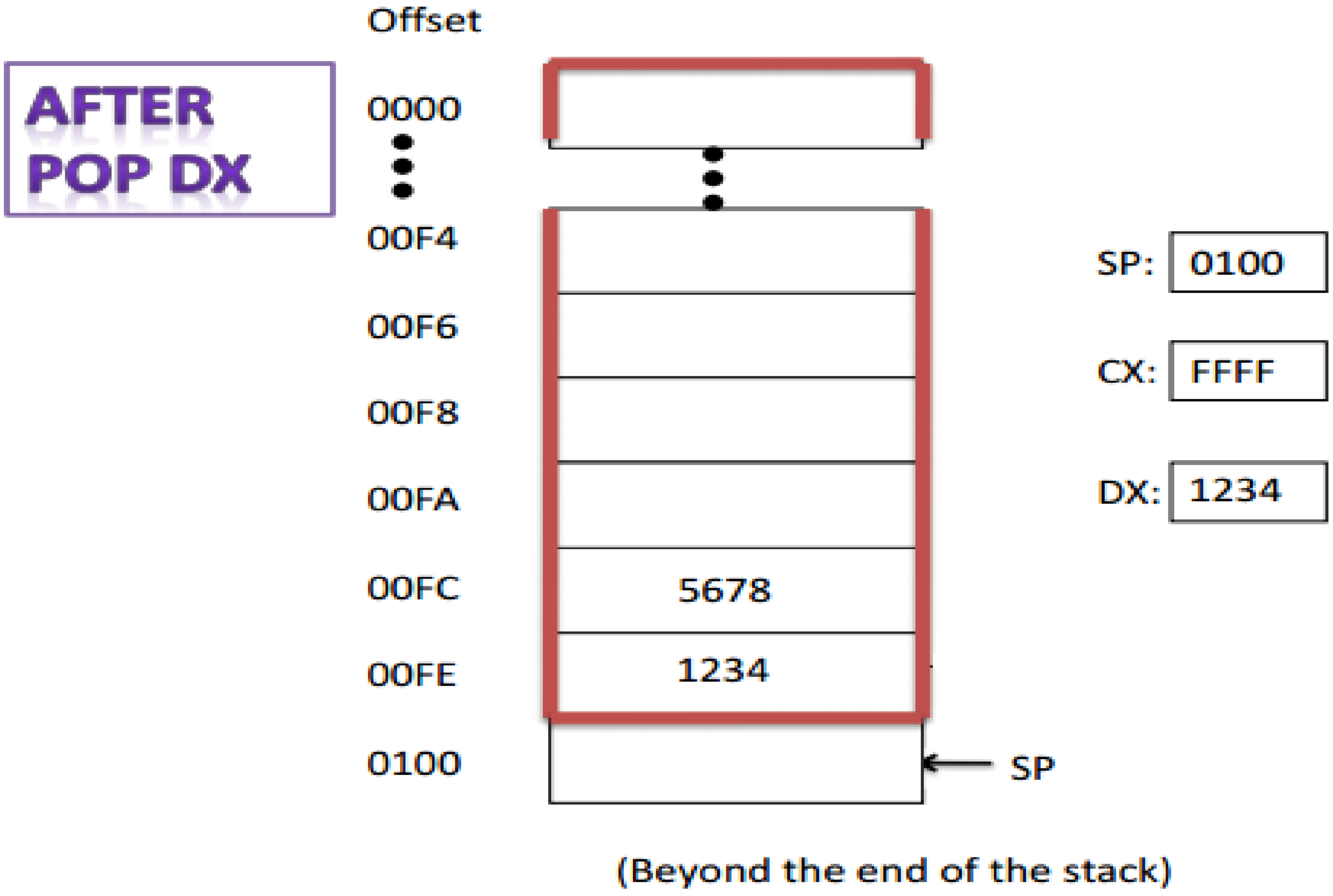
CX: 5678

DX: 0001

(Beyond the end of the stack)

Operation of POP Instruction

How words are removed from stack?



Example of Executing PUSH and POP Instructions

Example 01

- AX = 3245H
- BX = 1234H
- CX = ABCDH
- SP = FEH

PUSH AX
PUSH CX
POP BX
AX =?
BX =?
CX =?
SP =?

Example 02

- AX = 3245H
- BX = 1234H
- CX = ABCDH
- SP = FEH

PUSH BX
PUSH CX
POP BX
POP AX
PUSH CX
PUSH BX
POP CX
PUSH AX
POP BX
AX =?
BX =?
CX =?
SP =?

Example 03

- AX = 3245H
- BX = 1234H
- CX = ABCDH
- SP = FEH

PUSH BX
PUSHF
POPF
PUSH CX
POP BX
POP AX
PUSH CX
PUSH BX
POP CX
PUSH AX
POP BX
AX =?
BX =?
CX =?
SP =?

Operation of FLAG Register with Stack

Reasons of using FLAG Register with Stack

- ✓ **PUSHF and POPF** are most **used** in writing interrupt service routines, where anyone must be able to save and restore the environment, that is, all machine registers, to avoid disrupting machine operations while servicing the interrupt.

- ✓ When Multiple process wants to execute the CPU at a time, mode switch or context switch operation is performed. In context switching, FLAG registers is required to temporarily store into stack.

- ✓ When one process is executing the microprocessor and one that time another process wants to execute the CPU. The FLAG register of the 1st process is required to store into stack for start up the execution of 2nd process. Now FLAG register is working for 2nd process. After executing the 2nd process, values of FLAG register are extracted from stack and start processing for the 1st process.

Operation of FLAG Register with Stack

Operation of PUSHF and POPF Instructions

PUSHF

- ✓ SYNTAX: PUSHF
- ✓ Pushes (copies) the contents of the FLAGS register onto the stack.
- ✓ It has no operands.

POPF

- ✓ SYNTAX: POPF
- ✓ Pops (copies) the contents of the top word in the stack to the FLAGS register.
- ✓It has no operands.

NOTES:

- ➢PUSH, POP, and PUSHF do not affect the flags !!
- ➢POPF could theoretically change all the flags because it resets the FLAGS REGISTER to some original value that you have previously saved with the PUSHF instruction

Thank you