Assembly Language

Lecture FIVE

Memory segmentation and addressing

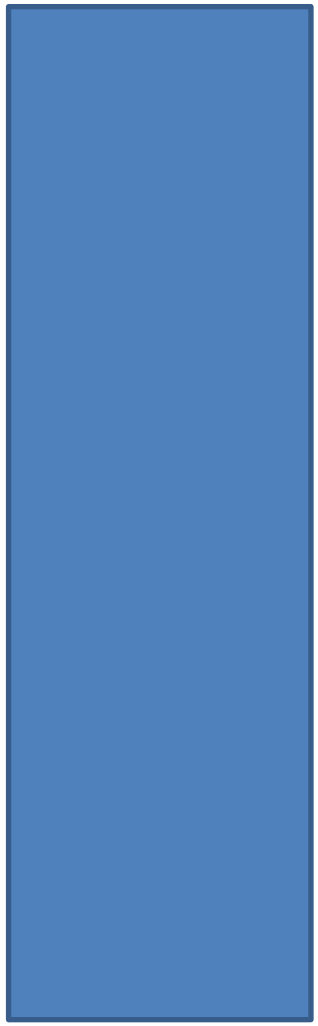
Addressing lines in 8086 are (20 line)

Size of memory = $2^{20} = 1\ Mbyte$

Every address in memory have 20 bit

The registers have (16 bits), then the memory will be addressing by any register is

$$2^{16} = 64\ Kbyte$$



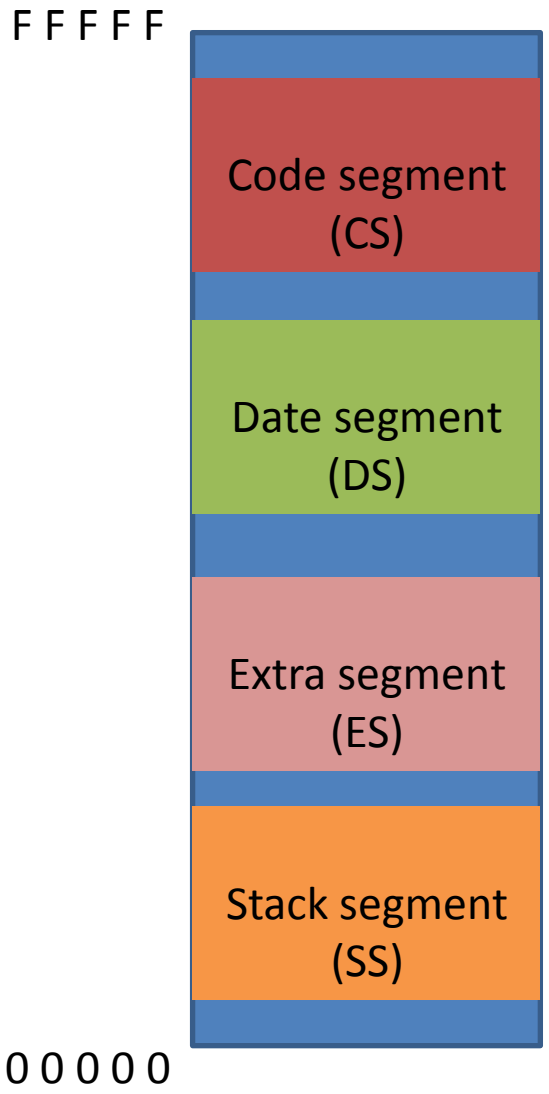
Memory segmentation and addressing

Code segment register (CS): is used for addressing memory location in the code segment of the memory

Data segment register (DS): points to the data segment of the memory where the data is stored

Extra Segment Register (ES): also refers to a segment in the memory which is another data segment in the memory.

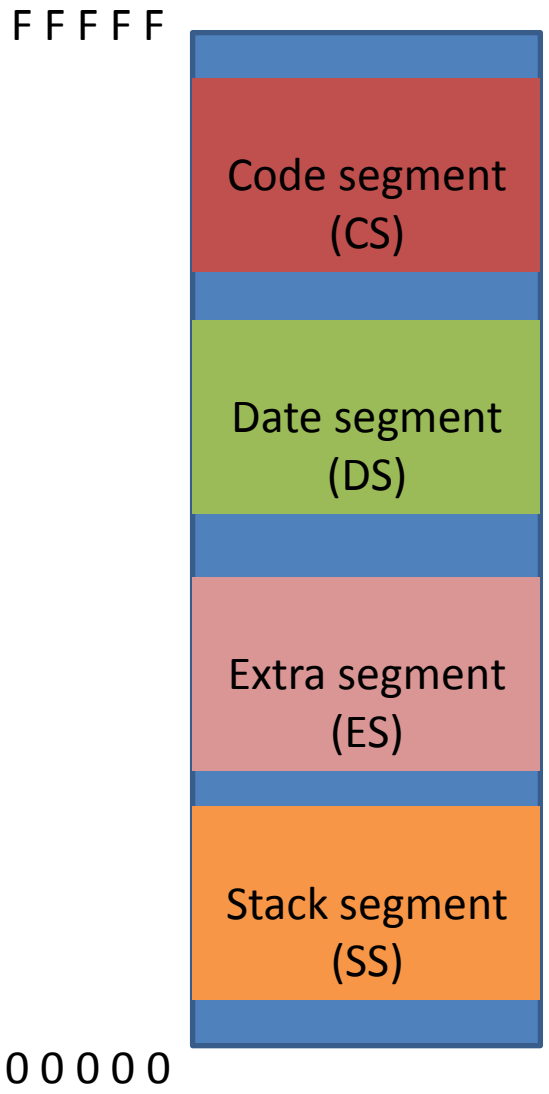
Stack Segment Register (SS): is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.



Memory segmentation and addressing

Segment : offset address

SEGMENT	SEGMENT REGISTER	OFFSET REGISTER
Code Segment	CSR	**Instruction Pointer (IP)**
Data Segment	DSR	**Source Index (SI)**
Extra Segment	ESR	**Destination Index (DI)**
Stack Segment	SSR	**Stack Pointer (SP) / Base Pointer (BP)**



Segment : Offset Address

- Logical Address is specified as **segment:offset**
- Physical address is obtained by shifting the segment address 4 bits to the left and adding the offset address.
- Thus the physical address of the logical address **A4FB:4872** is:

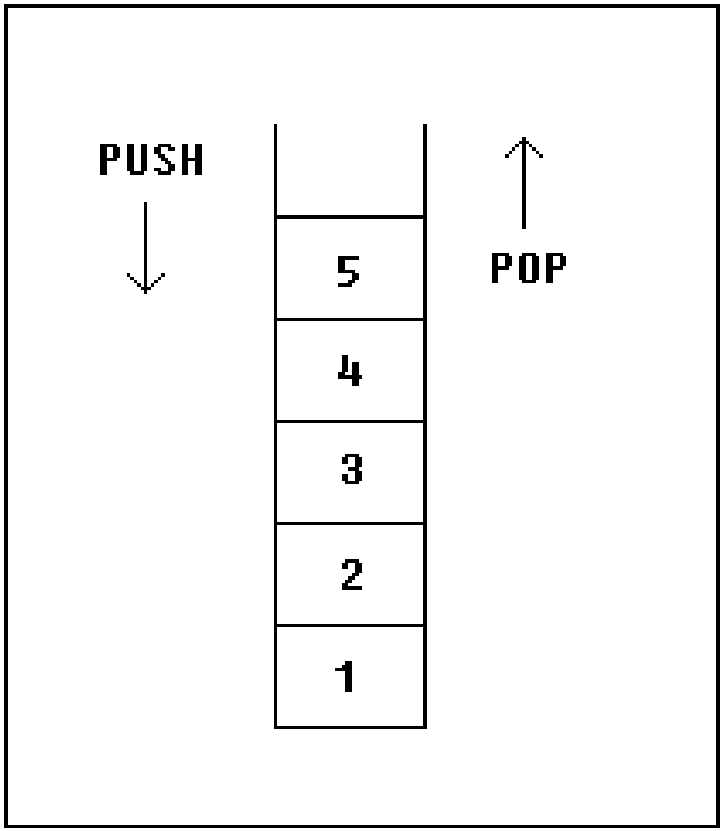
$$\begin{array}{r} \text{A4FB0} \\ +\ \ 4872 \\ \hline \text{A9822} \end{array}$$

Segments, Segment Registers & Offset Registers

- Segment Size = 64KB
- Maximum number of segments possible = 16
- Logical Address – 16 bits
- Physical Address – 20 bits
- 2 Logical Addresses for each Segments.
  - Base Address (16 bits)
  - Offset Address (16 bits)
- Segment registers are used to store the Base address of the segment.

Stack segment

- The stack uses **LIFO** (Last In First Out) algorithm, this means that if we push these values one by one into the stack:
  1, 2, 3, 4, 5
  the first value that we will get on pop will be **5**, then **4**, **3**, **2**, and only then **1**.



Stack segment

There are two instructions that work with the stack:

PUSH - stores 16 bit value in the stack.

POP - gets 16 bit value from the stack.

PUSH

- Syntax for **PUSH** instruction:

  PUSH REG

  PUSH SREG

  PUSH memory

  PUSH immediate (works only on 80186 and later)
- **REG**: AX, BX, CX, DX, DI, SI, BP, SP
- **SREG**: DS, ES, SS, CS
- **memory**: [BX], [BX+SI+7], 16 bit variable, etc.
- **immediate**: 5, -24, 3Fh, 10001101b, etc.

POP

- Syntax for **POP** instruction:

  POP REG

  POP SREG

  POP memory

- **REG**: AX, BX, CX, DX, DI, SI, BP, SP
- **SREG**: DS, ES, SS, (except CS)
- **memory**: [BX], [BX+SI+7], 16 bit variable, etc.

A Simple Stack Example

MOV AX, 1234h

PUSH AX ; store value of AX in stack.

MOV AX, 5678h ; modify the AX value.

POP AX ; restore the original value of AX.

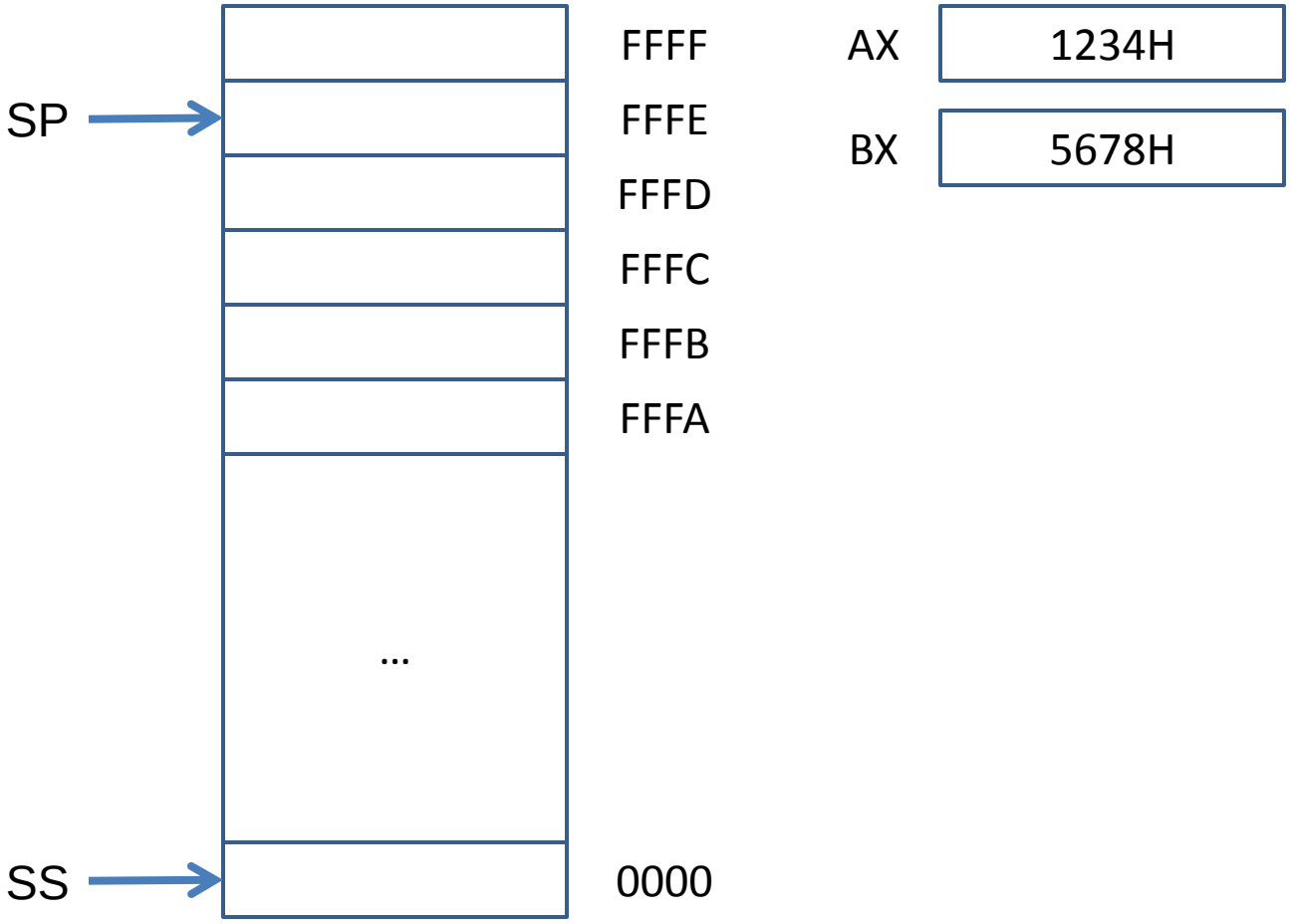
Exchanging Values Using Stack

```
MOV AX, 1234h ; store 1234h in AX.
MOV BX, 5678h ; store 5678h in BX

PUSH AX ; store value of AX in stack.
PUSH BX ; store value of BX in stack.

POP AX ; set AX to original value of BX.
POP BX ; set BX to original value of AX.
```

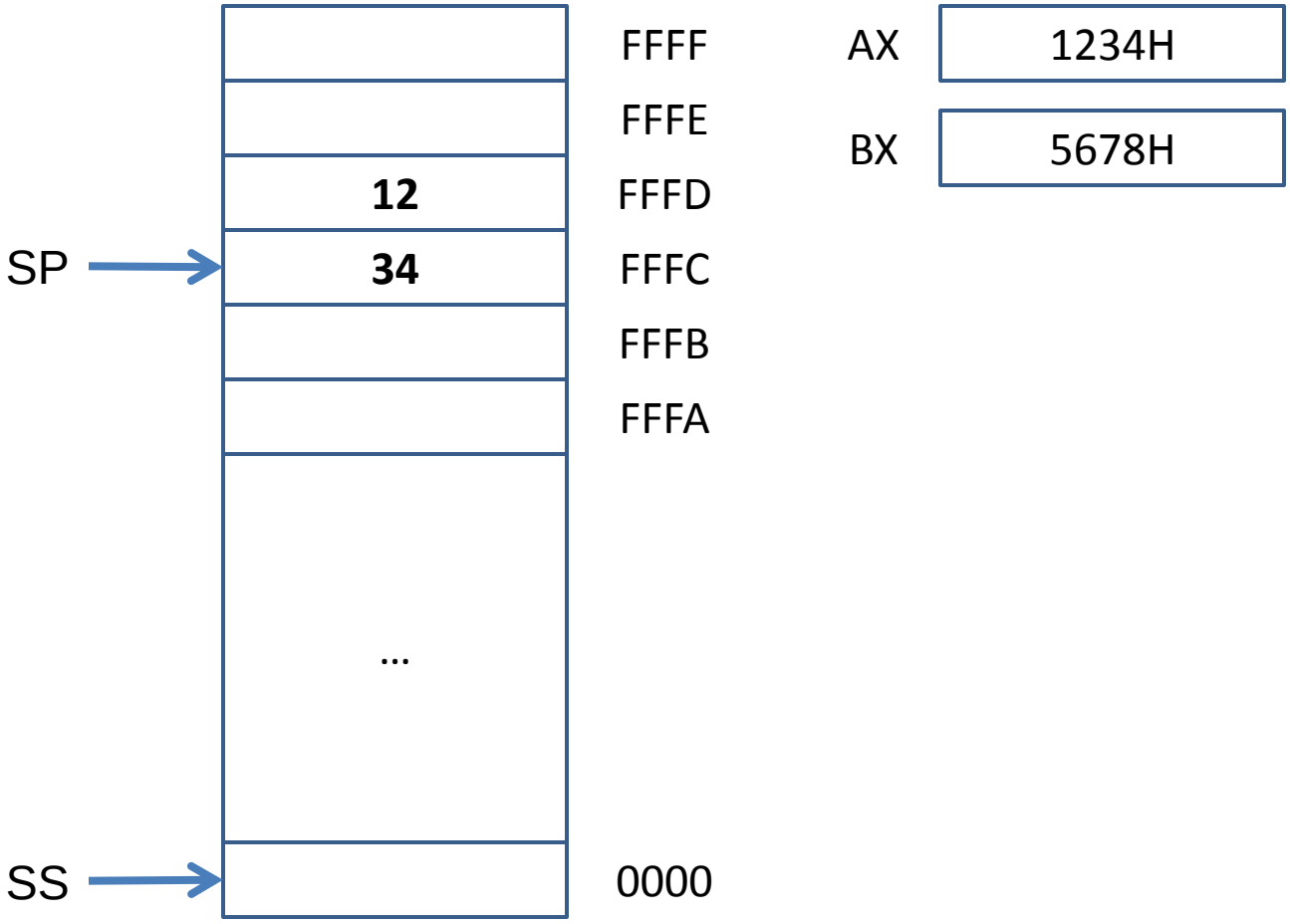
PUSH and POP

- "**PUSH *source***" instruction does the following:
  - Subtract **2** from **SP** register.
  - Write the value of ***source*** to the address **SS:SP**.
- "**POP *destination***" instruction does the following:
  - Write the value at the address **SS:SP** to ***destination***.
  - Add **2** to **SP** register.

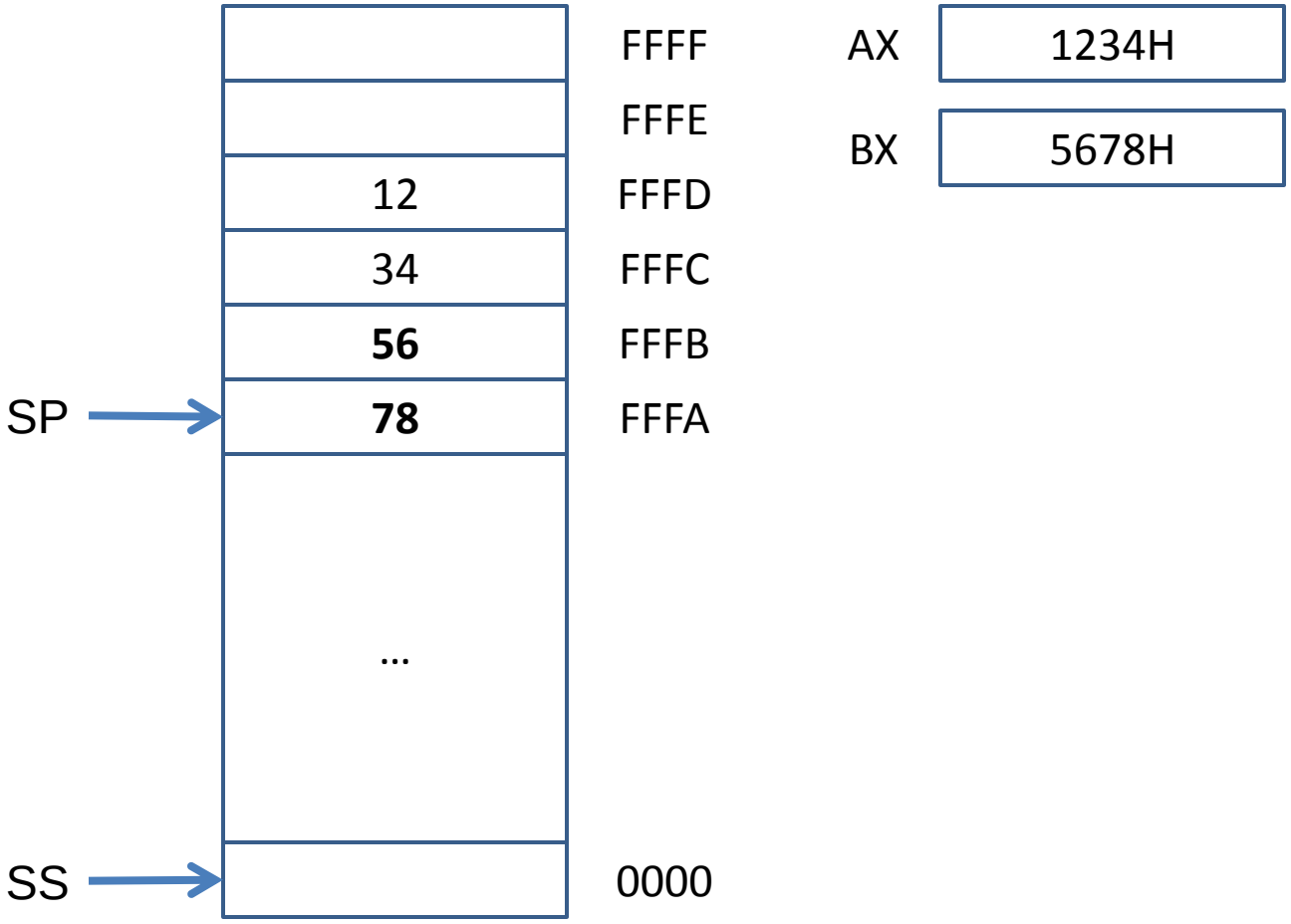
Initial State of the Stack



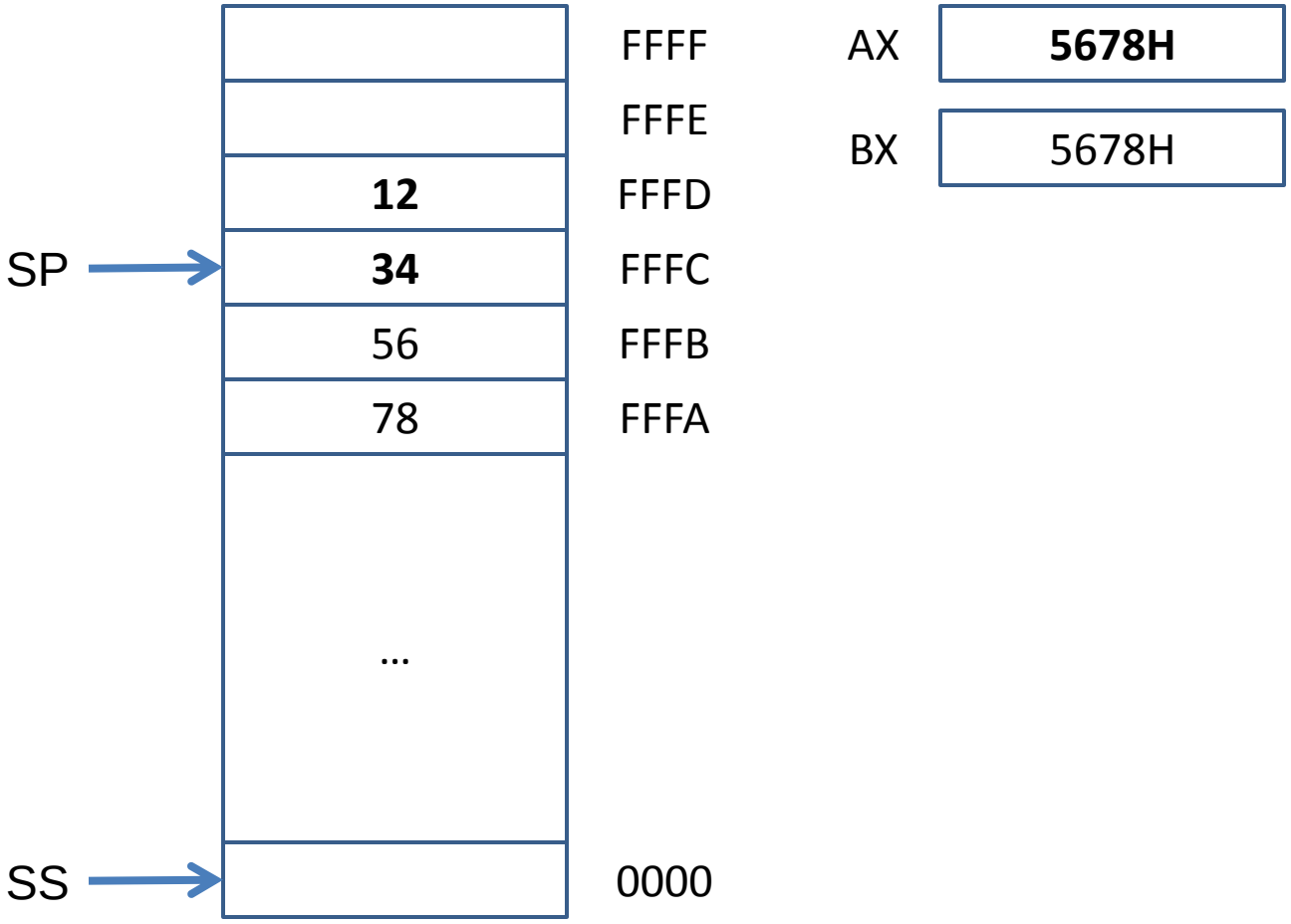
After PUSH AX



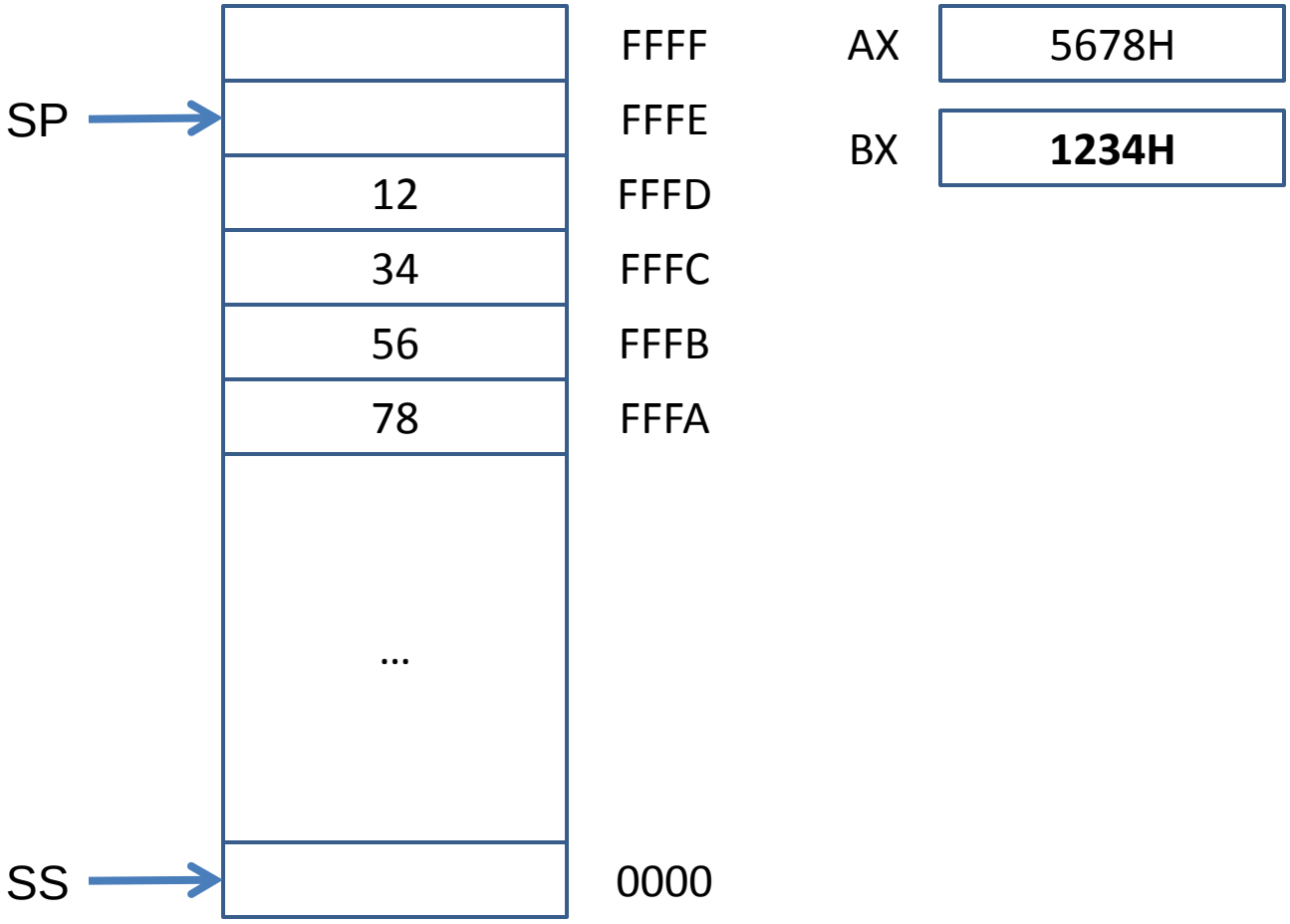
After PUSH BX



After POP AX


FFFF
FFFE
12
FFFD
SP
34
FFFC
56
FFFB
78
FFFA
...
SS
0000
AX
5678H
BX
5678H

After POP BX


SP
SS
FFFF
FFFE
FFFD
FFFC
FFFB
FFFA
0000
12
34
56
78
...
AX
5678H
BX
1234H