Computer Science Dept.

Department	Computer Science	:القسم
Subject Name:	Microprocessors - (6)	أسم المادة :
Year of Study:	2024-2023	:السنة الدراسية
Term:	Second Term	:الفصل الدراسي
Email	ali.sadoon@uoa.edu.iq	Email
Instructor Name:	Ali Saadoon AHMED	:أسم التدريسي

OUTLINE

STACK

FLAGS

STACK

What is a stack, and why is it needed?

- The stack is a section of read/write memory (RAM) used by the CPU to store information *temporarily*. CPU needs this storage area since there are *only limited number* of registers.

STACK

How stacks are accessed

- **SS** (stack segment) and **SP** (stack pointer) must be loaded to access stack in the memory.
- Every register in the CPU (except segment registers and SP) can be stored in the stack and loaded from the stack.

STACK

Pushing onto the stack

- Storing the CPU register in the stack is called a *push*.

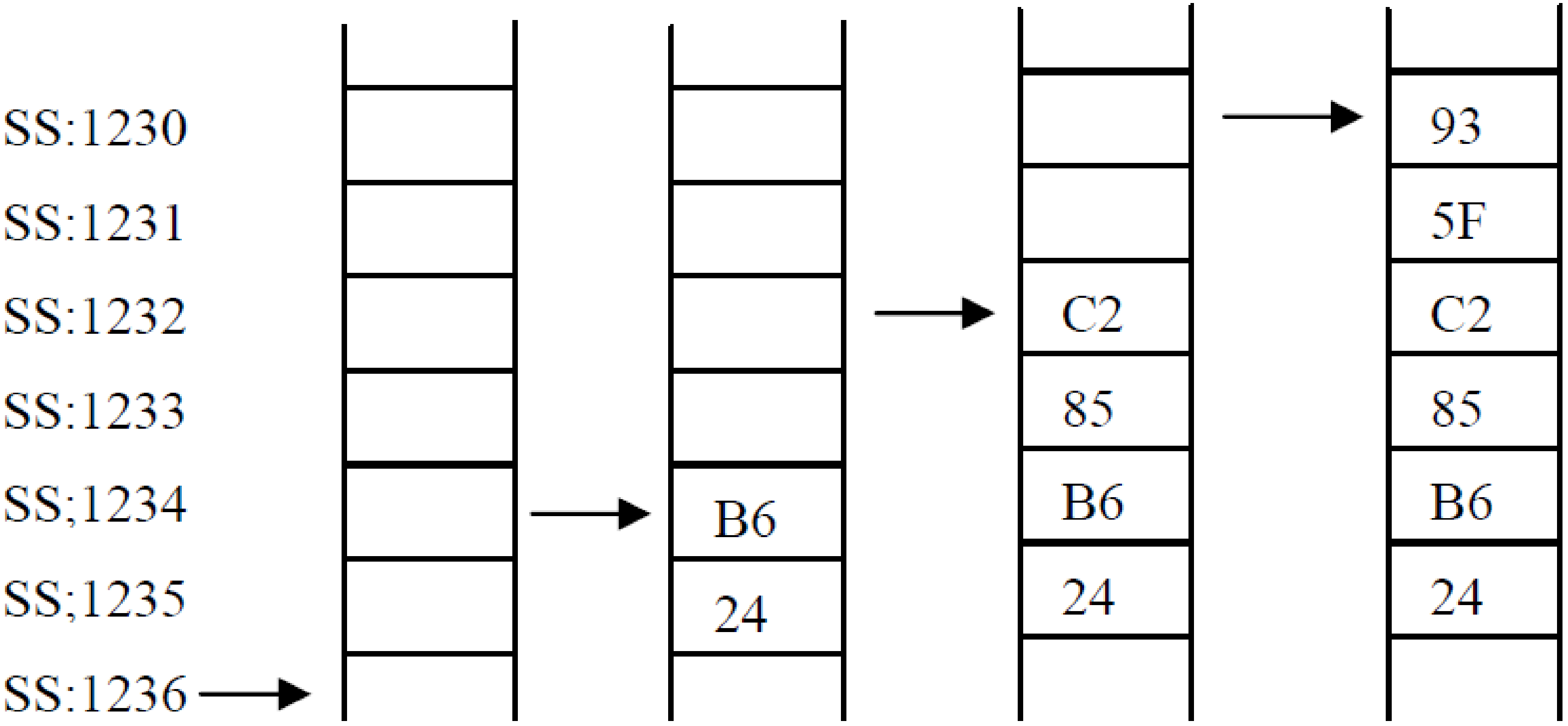
Ex: SP=1236, AX=24B6, DI=85C2, and DX=5F93, show the contents of the stack as each instruction is executed.

PUSH AX

PUSH DI

PUSH DX

STACK



	START SP=1236	After PUSH AX SP=1234	After PUSH DI SP=1232	After PUSH DX SP=1230
SS:1230				93
SS:1231				5F
SS:1232			C2	C2
SS:1233			85	85
SS;1234		B6	B6	B6
SS;1235		24	24	24
SS:1236				

STACK

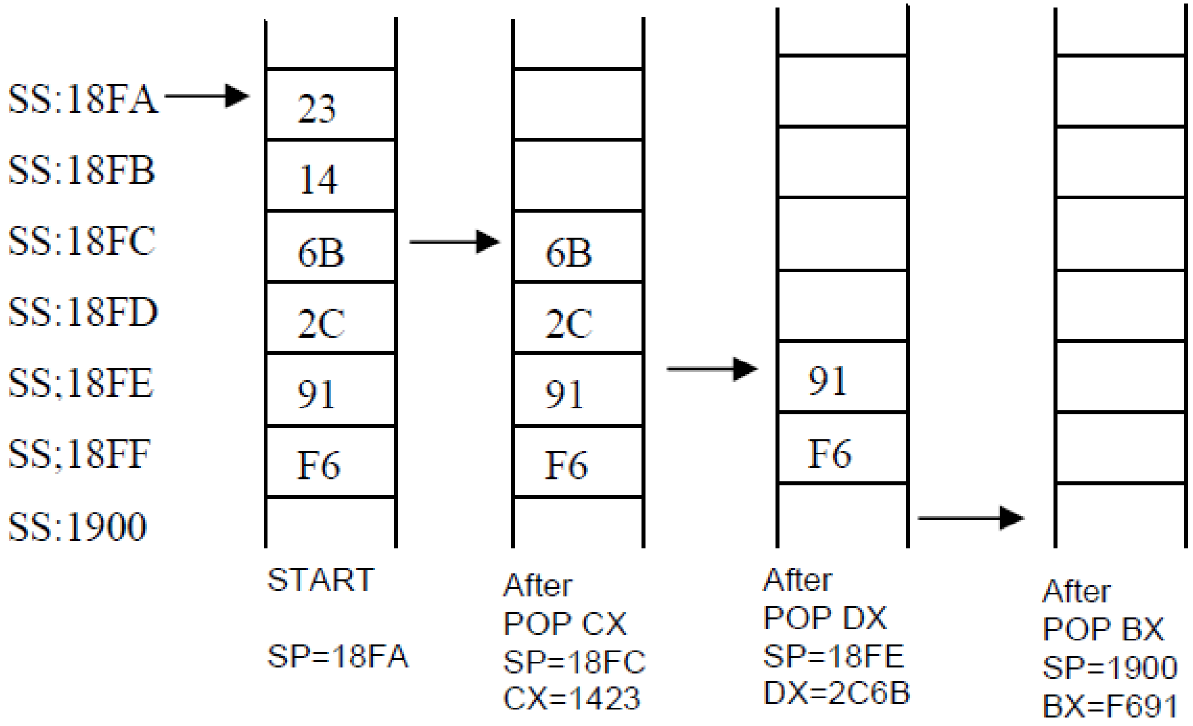
Popping the stack

- Loading the contents of the stack into the CPU register is called a *pop*.

Ex: assume that the stack is shown below, and SP=18FA, show the contents of the stack and registers as each of the following instructions is executed.

```
POP CX
POP DX
POP BX
```

STACK



SS:18FA
SS:18FB
SS:18FC
SS:18FD
SS:18FE
SS;18FF
SS:1900

START	After POP CX	After POP DX	After POP BX
23			
14			
6B	6B		
2C	2C		
91	91	91	
F6	F6	F6	

START

SP=18FA

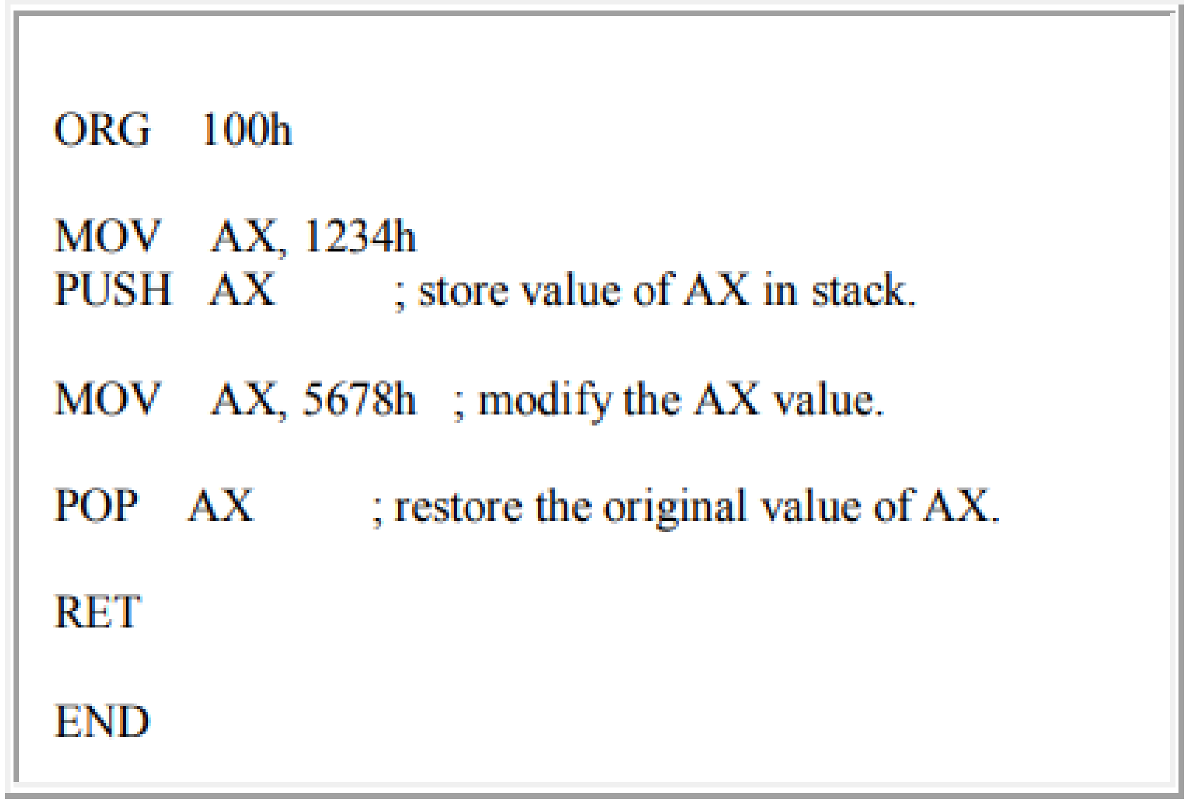
After
POP CX
SP=18FC
CX=1423

After
POP DX
SP=18FE
DX=2C6B

After
POP BX
SP=1900
BX=F691

STACK

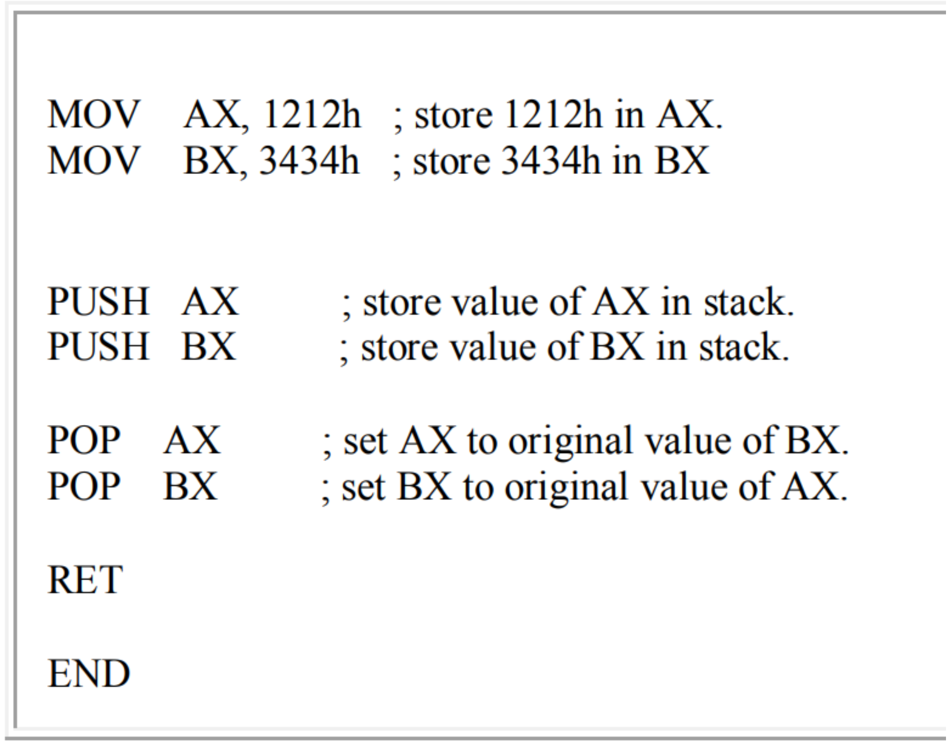
Here is an example:



```
ORG    100h

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.

RET

END
```

STACK

- Another use of the stack is for exchanging the values,
- here is an example:

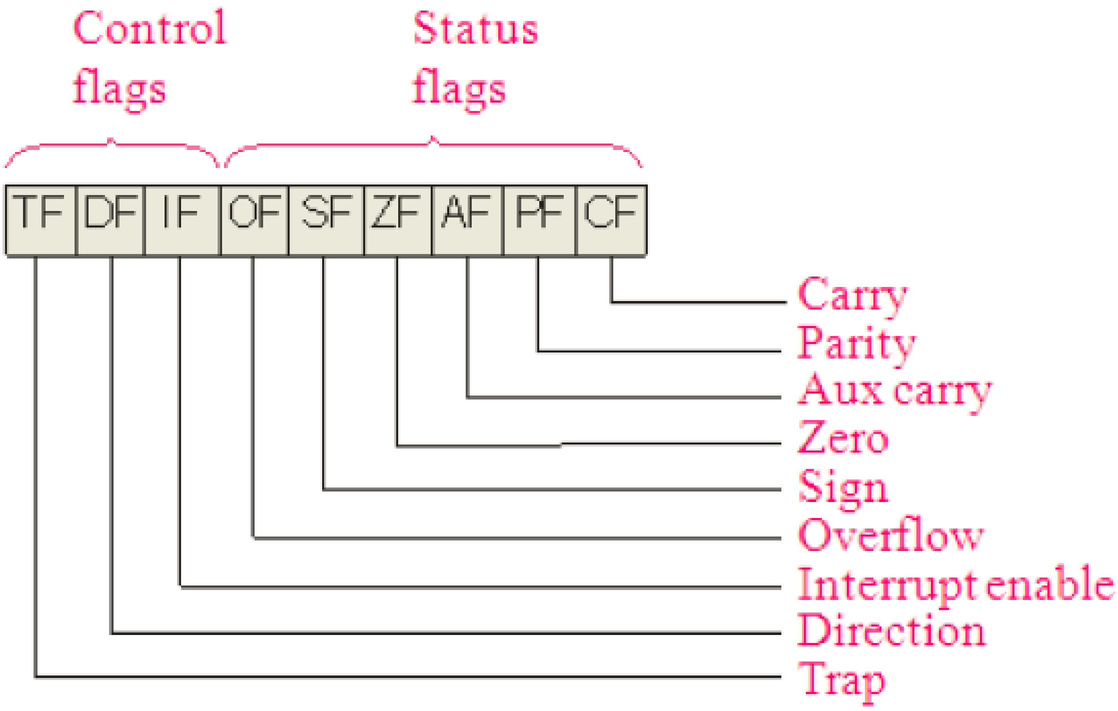


```
MOV   AX, 1212h  ; store 1212h in AX.
MOV   BX, 3434h  ; store 3434h in BX


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.

RET

END
```

The exchange happens because stack uses LIFO (Last In First Out) algorithm, so when we push **1212h** and then **3434h**, on pop we will first get **3434h** and only after it **1212h**.

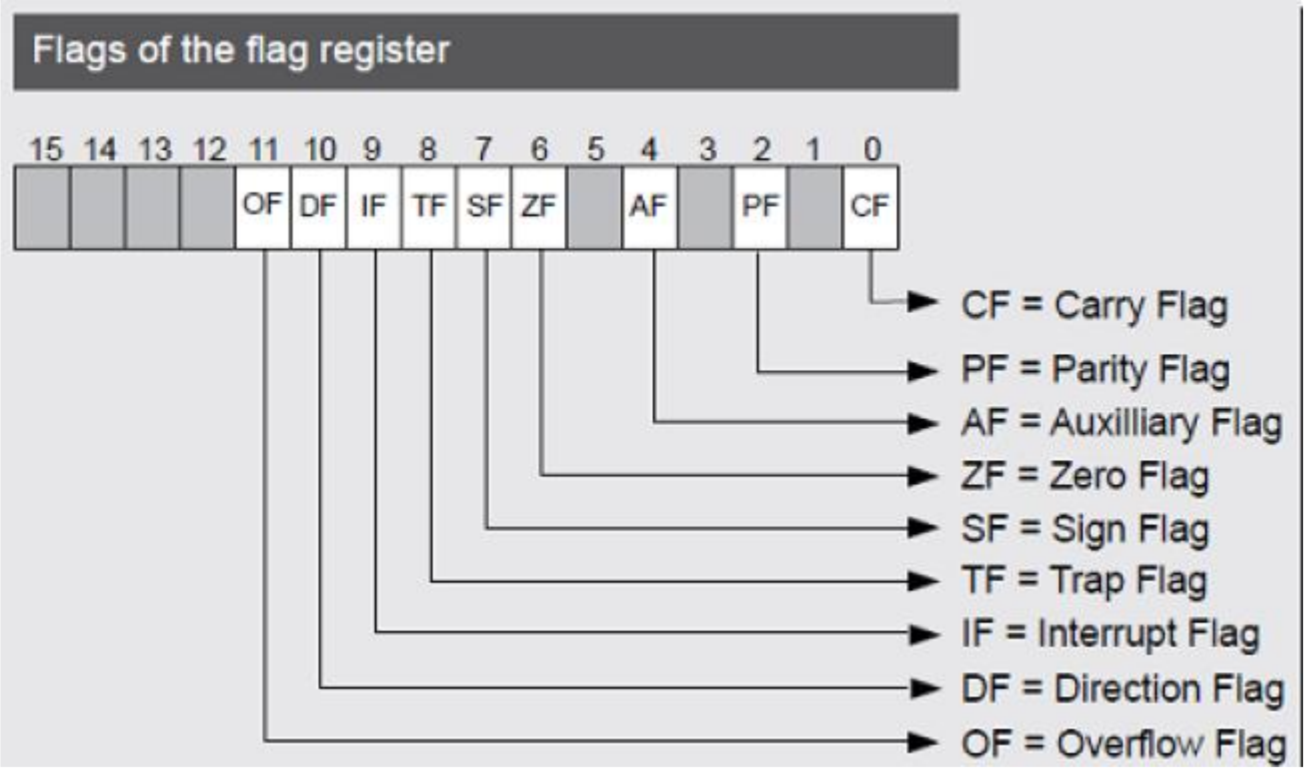
Flag

- ***The Flags:***
- The flag register is a 16-bit register 9 bits from 16 bits are only used as control bits (flags). A status flag is a one – bit indicator used to reflect a certain condition after an arithmetic or logic operation by the ALU.



Flag

The Flags:



Flags of the flag register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
▨	▨	▨	▨	OF	DF	IF	TF	SF	ZF	▨	AF	▨	PF	▨	CF

CF = Carry Flag

PF = Parity Flag

AF = Auxilliary Flag

ZF = Zero Flag

SF = Sign Flag

TF = Trap Flag

IF = Interrupt Flag

DF = Direction Flag

OF = Overflow Flag

Flag

Overflow Flag (OF): set if the result is a too large positive number or is a too small negative number to fit into the destination operand.

Direction Flag (DF): if set then strings manipulation instructions will auto-decrement index register. If cleared, then the index registers will be auto-incremented.

Interrupt-enable Flag (IF): setting this bit enables maskable interrupts.

Single-step Flag (TF): if set then a single-step interrupt will occur after the next instruction.

Flag

Zero Flag (ZF): set if the result is zero.

Auxiliary carry Flag (AF): set if there was a carry from or borrow to bits 0-3 in the AL register.

Parity Flag (PF): set if parity (the number of “1” bits) in the low-order byte of the result is even.

Carry Flag (CF): set if there was a carry from or borrow to the most significant bit during last result calculation.

Sign Flag (SF): set if the most significant bit of the result is set.

Flag

Flag Register and ADD instruction

- The flag bits affected by the ADD instructions are CF, PF, AF, ZF, SF and OF.

Ex: Show how the flag register is affected by the addition of 38H and 2FH?

Flag

Solution:

- MOV BH,38H ; BH=38H
- ADD BH,2FH ; BH = BH + 2F = 38 + 2F= 67H

38	0011 1000
+ 2F	0010 1111
67	0110 0111

EXECUTION UNITS (EU) IN 8086/8088 MP:

CF = 0 since there is no carry beyond d7

PF = 0 since there is odd number of 1`s in the result

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 0 since d7 of the result is zero

Flag

Ex: Show how the flag register is affected by the following addition?

Solution:

- MOV AX,34F5H ;   AX =34F5H
- ADD AX,95EBH ;   AX = CAE0H

```
      34F5          0011 0100 1111 0101
+     95EB          1001 0101 1110 1011
---------------------------------------
      CAE0          1100 1010 1110 0000
```

Flag

CF = 0 since there is no carry beyond d15

PF = 0 since there is odd number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 1 since d15 of the result is 1

Flag

Use of zero flag for looping

- Zero flag is used to implement the program loops. Loop refers to a set of instructions that is repeated a number of times.
- The following example shows the implementation of the loop concept in the program which adds 5 bytes of data.

Ex:

```
MOV CX,05 ; CX holds the loop count
MOV BX,0200H ; BX holds the offset data address
MOV AL,00 ; initialize AL
ADD_LP: ADD AL,[BX] ; add the next byte to AL
INC BX ; increment the data pointer
DEC CX ; decrement the loop counter
JNZ ADD_LP ; jump to the next iteration if the counter not zero
```

Computer Science Dept.

THANK YOU

By:

Ali Saadoon AHMED