

# ARCHITECTURE DES ORDINATEURS

## 7)- Introduction à l' **ASSEMBLEUR 'x86' (II)**

Biblio ::    **1/« *Assembly Langage for INTEL-based computers*»**

[Kip R. IRVINE] – Ed. Prentice Hall, 1999 – ISBN: 0-13-660390-4.

**2/« *An assembly langage introduction to computer architecture (using the intel pentium)*»**

[K. MILLER] – Ed. Oxford University Press, 1999 – ISBN: 0-19-512376-X

**3/«*The Intel microprocessors : Architecture, Programming & Interfacing*»**

[Barry B. BREY] – Ed. Prentice Hall 2006 – ISBN: 0-13-119506-9.

**4/ «*The Hardware Bible*»**

[Winn L. Rosch] – Ed. QUE /McMillan computer Publishing – ISBN: 0-7897-1743-3.

@ web::

1-<http://css.csail.mit.edu/6.858/2013/readings/>

2-<http://www.ustudy.in/node/>

## 7)- Introduction à l'ASM 'x86' (II)



### Objectifs :

(I) « PILE » : Utilité, Intérêt & définition

(II) « PILE » : Mécanismes de fonctionnement

(III) « PILE » : Illustration (prg ASM)

## 7)- Introduction à l'ASM 'x86' (II)



### « PILE » : quel intérêt ?

; exemple / illustration = '**PILE & PERTE D'INFORMATION**'

```
mov ax, 11h  
mov cx, 1Fh  
add ax, cx
```

; ... Suite instructions (*ax\_initial*) ???



« **add** » ::= > Résultat, mais ..

**Pb:** PERTE DE LA VALEUR (AX)

## 7)- Introduction à l'ASM 'x86' (II)

### « PILE » : Intérêt ?

; exemple / illustration = '**PILE & PERTE D'INFORMATION**'

```
mov ax, 11h  
mov cx, 1Fh  
add ax, cx
```

; ... Suite instructions (*ax\_initial*) ???

« **add** » ::= > Résultat, mais ..

**Pb:** PERTE DE LA VALEUR (AX)

En L.E. ::

**Solution::** 'Déclaration de ' $\infty$ ' nouvelles variables'

## 7)- Introduction à l'ASM 'x86' (II)

### « PILE » : Intérêt # 'SAVE'

; exemple / illustration = '**PILE & PERTE D'INFORMATION**'

```
mov ax, 11h  
mov cx, 1Fh
```

; ... Sauvegarde de (ax) en pile

'SAUVE' ax

; ... Instructions ...

'RECUP' ← ax

```
add ax, cx
```

; ... Suite instructions (ax\_initial) :: OK

« **SAUVE** » ::=> Sauvegarde .. **DE LA VALEUR Initiale (AX)**

« **RECUP** » ::=> Restitution.. **DE LA VALEUR Initiale (AX)**

## 7)- Introduction à l'ASM 'x86' (II)

### « *PILE* » : Intérêt # 'SAVE'

; exemple / illustration = '**PILE & PERTE D'INFORMATION**'

```
mov ax, 11h  
mov cx, 1Fh
```

; ... Sauvegarde de (ax) en pile

```
push ax
```

; ... Instructions ...

```
pop ax
```

```
add ax, cx
```

; ... Suite instructions (ax\_initial) :: OK

« **push** » ::= > Sauvegarde .. DE LA  
VALEUR Initiale (AX)

« **pop** » ::= > Restitution.. DE LA  
VALEUR Initiale (AX)

## 7)- Introduction à l'ASM 'x86' (II)

### « PILE » : Intérêt # 'SAVE EXPLICITE'

; exemple / illustration = 'PILE & PERTE D'INFORMATION'

```
mov ax, 11h  
mov cx, 1Fh
```

; ... Sauvegarde de (ax) en pile

**push** ax

; ... Instructions ...

**pop** ax

```
add ax, cx
```

; ... Suite instructions (ax\_initial) :: OK



« **push** » / « **pop** » ::= >

- manipulation EXPLICITE de la PILE
- Instruction MONO -opérande

## 7)- Introduction à l'ASM 'x86' (II)

« *PILE* » : Intérêt # *'SAVE IMPLICITE'*

```
Int main()
```

```
{  
fct_A();
```

```
...  
}
```

CS (main)

IP (main)

@

```
Void fct_A()
```

```
{  
int a1, a2
```

```
fct_B();
```

```
...  
}
```

a1

a2

CS (fct\_A)

IP (fct\_A)

données

```
Void fct_B()
```

```
{  
int b1, b2
```

```
...  
}
```

b1

b2



## 7)- Introduction à l'ASM 'x86' (II)



### *« PILE » : Définition*

*SAVE 'IMPLICITE' ou  
'EXPLICITE'*



*Recours INDISPENSABLE à la PILE pour  
TOUTE SAUVEGARDE 'TEMPORAIRE'*

*PILE: Espace mémoire (de la RAM) destiné à  
contenir des infos 'temporaires'*

## 7)- Introduction à l'ASM 'x86' (II)



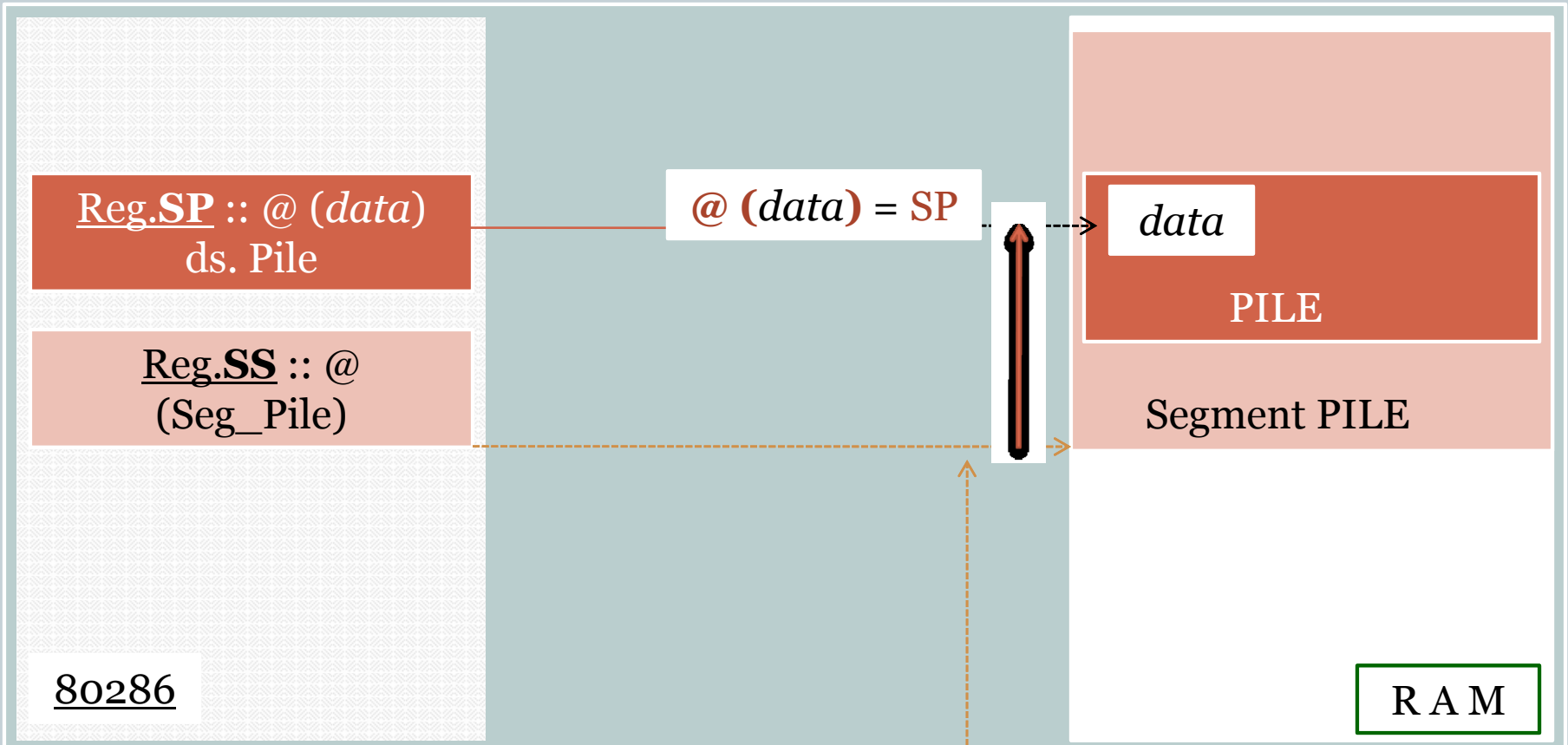
### Thèmes & Objectifs :

(I) « PILE » : Mécanismes de fonctionnement

(II) « PILE » : Illustration (prg ASM)

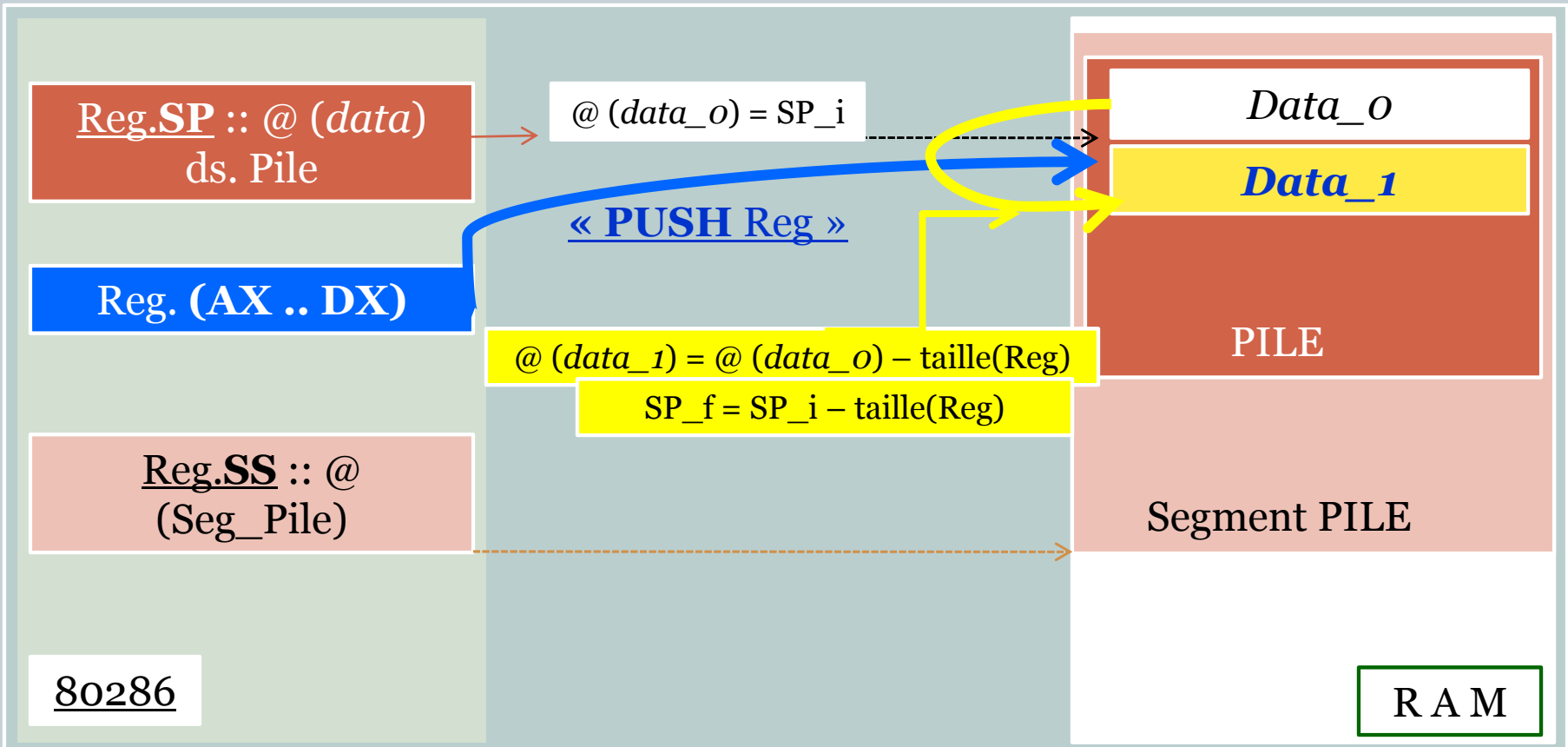
## 7)- Introduction à l'ASM 'x86' (II)

### Mécanismes de fonctionnement : « PILE »



## 7)- Introduction à l'ASM 'x86' (II)

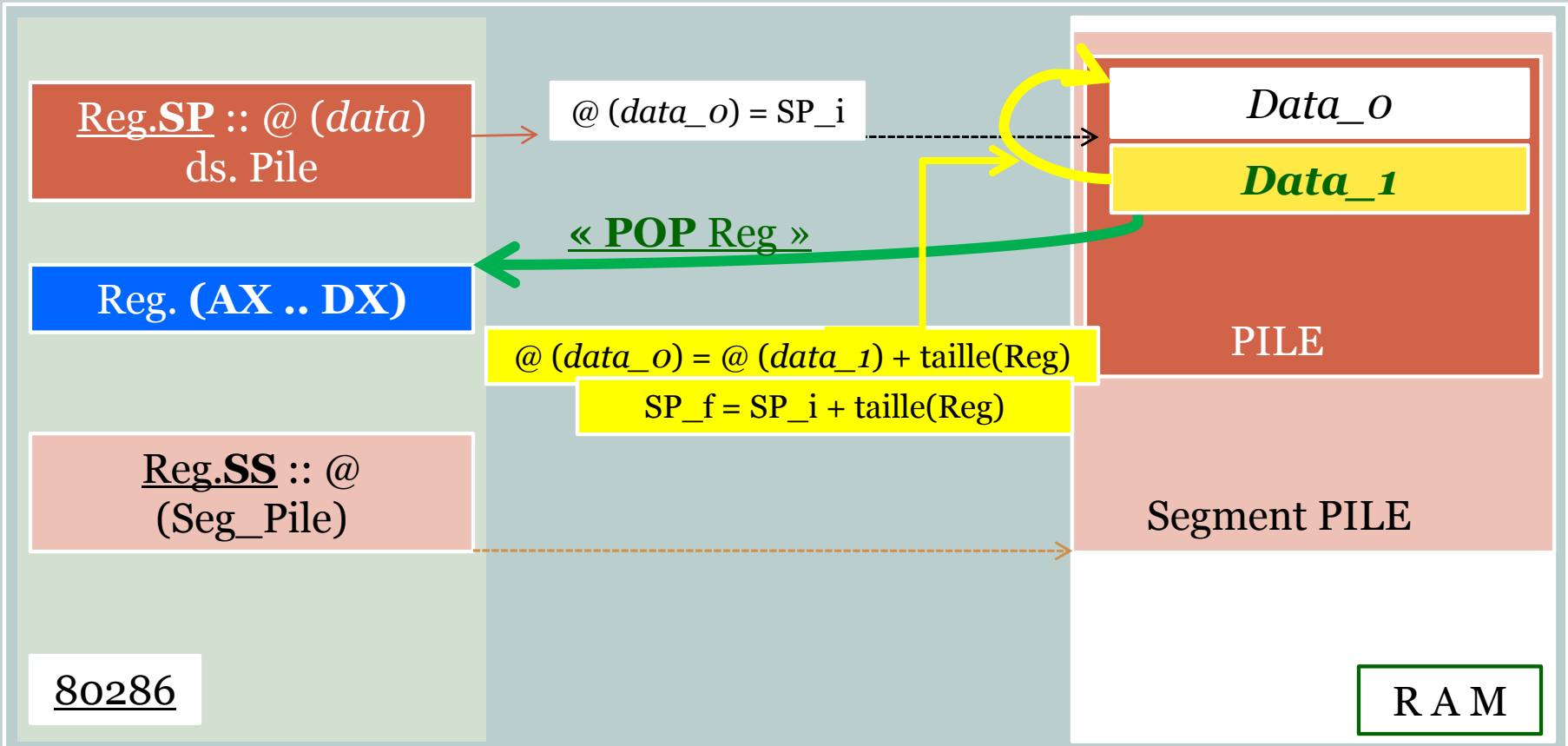
### Mécanismes de fonctionnement : « Stockage / PILE »



## 7)- Introduction à l'ASM 'x86' (II)



### Mécanismes de fonctionnement : « Déstockage / PILE »



## 7)- Introduction à l'ASM 'x86' (II)



### Mécanismes de fonctionnement : « Equivalence / PUSH & POP »

#### « PUSH Reg »

( 1 )

$[SP\_i] \leftarrow Reg$

( 2 )

$SP\_f = SP\_i - \text{taille}(Reg)$

#### « POP Reg »

( 1 )

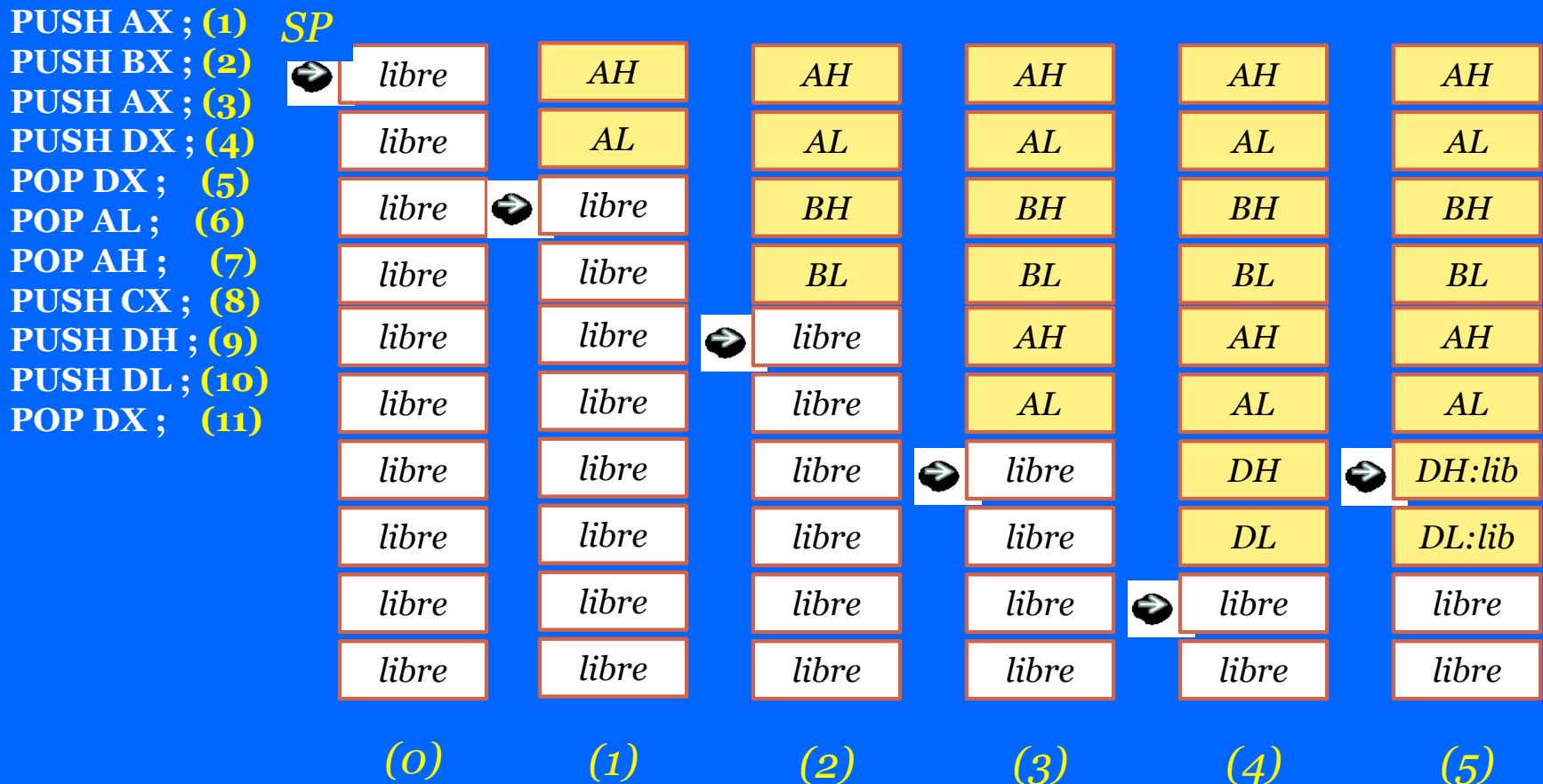
$SP\_f = SP\_i - \text{taille}(Reg)$

( 2 )

$Reg \leftarrow [SP\_i]$

# 7)- Introduction à l'ASM 'x86' (II)

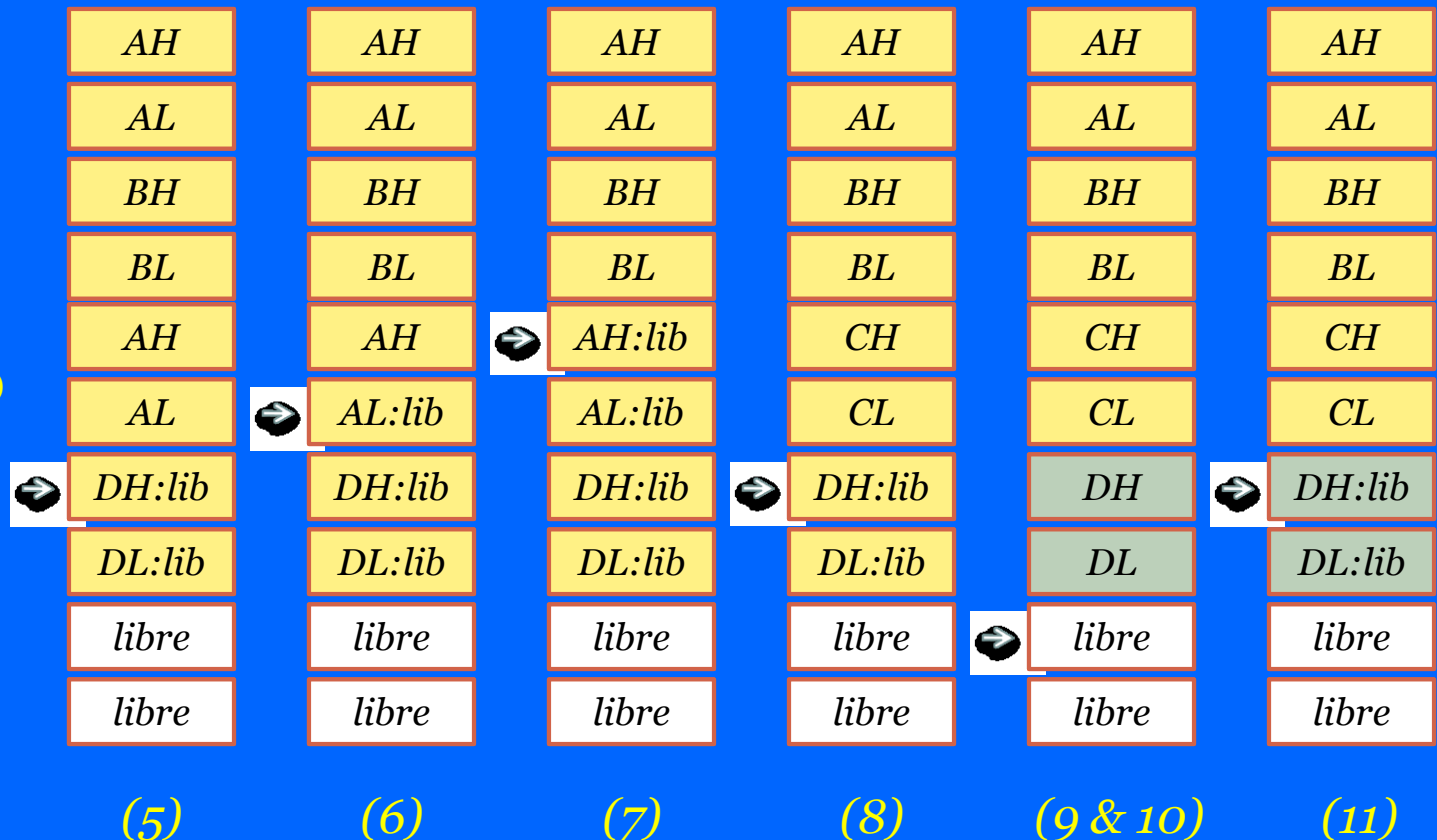
## Mécanismes de fonctionnement : « Illustration »



# 7)- Introduction à l'ASM 'x86' (II)

## Mécanismes de fonctionnement : « Illustration »

PUSH AX ; (1)  
 PUSH BX ; (2)  
 PUSH AX ; (3)  
 PUSH DX ; (4)  
 POP DX ; (5)  
 POP AL ; (6)  
 POP AH ; (7)  
 PUSH CX ; (8)  
 PUSH DH ; (9)  
 PUSH DL ; (10)  
 POP DX ; (11)





## 7)- Introduction à l'ASM 'x86' (II)



Conséquence : « je retiens ... / PILE »

**PILE** :: ...

1. Espace mémoire (de la RAM) destiné à contenir des infos 'temporaires' (à courte durée de vie)
2. Stockage par 'PUSH' et déstockage par 'POP'
3. Stockage =>  $SP\_nouv = SP\_last - \text{taille}(\text{reg})$
4. Déstockage =>  $SP\_nouv = SP\_last + \text{taille}(\text{reg})$
5. Gestion IMPLICITE & AUTOMATIQUE de l'@ (*data*) :: valeur du reg.« SP »
6. NB: **(a)** certaines instructions (appel de procédure) provoquent implicitement des « PUSH / POP » (mécanismes de stockage/déstockage en pile).  
**(b)** le pointeur **SP** pointe **TOUJOURS** sur un octet considéré **LIBRE**
7. Modèles de PILES : LIFO vs FIFO

## 7)- Introduction à l'ASM 'x86' (II)



### Thèmes & Objectifs :

(I) « PILE » : Mécanismes de fonctionnement

(II) « PILE » : Illustration (prg ASM)

## 7)- Introduction à l'ASM 'x86' (II)



; exemple / illustration = '**MANIP de la PILE :: Assymétrie PUSH / POP**'

**; les stockages & destockages ne sont pas “symétriques”**

mov ax, 10h

mov cx, 1Fh

**push ax** ←

**pop bx** ←

and bx,cx

Stockage de la valeur de **AX** dans la pile

Déstockage de la pile dans le registre **BX**