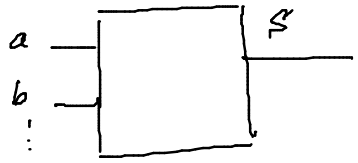
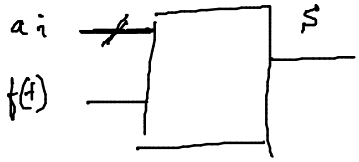
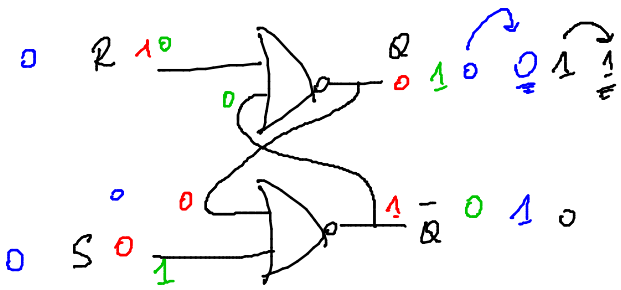
Sequence 2 — A12

Logique combinatoire → logique seq^{ie}.





L'élt de base — bascule

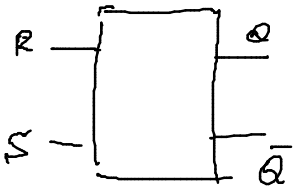


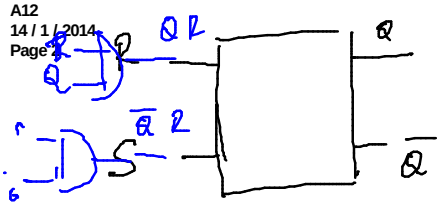
R	S	Q_{n-1}	Q_n	
0	0	1	1	Set
0	1	1	0	reset
1	0	0	0	
1	1			

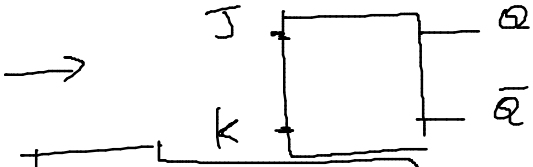
même niveau

$Q = 0 \rightarrow Q_{n+1} = 0$

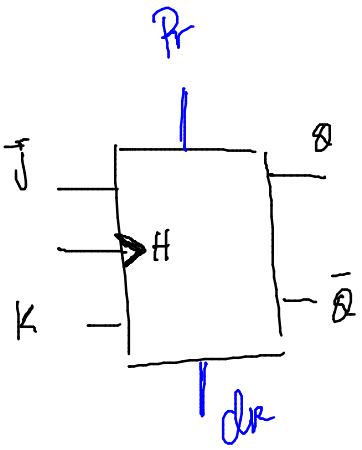
$Q = 1 \rightarrow Q_{n+1} = 1$



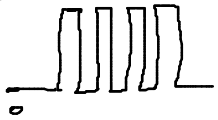


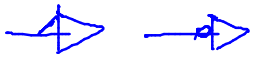


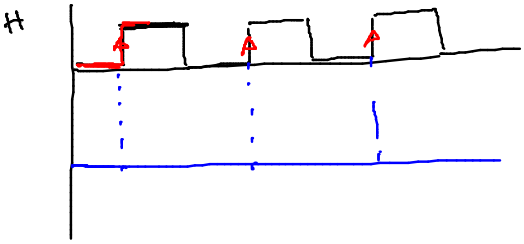
J	k	Qn
0	0	Qn-1
1	0	1
0	1	0
1	1	$\overline{Qn-1}$



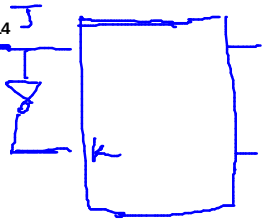
J = 1
k = 1





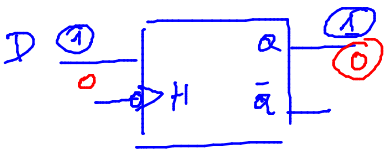


(Pr et clr) → entrées asynchrones. prioritaires.



1 0
0 1

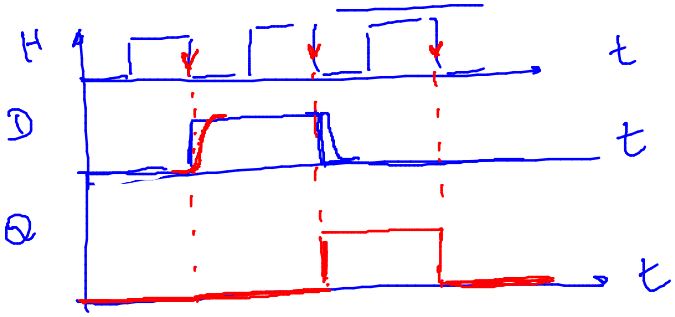
Bascule D.



Copie la valeur de D

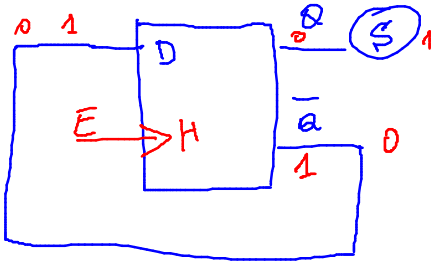
t_{PLH}

Rq

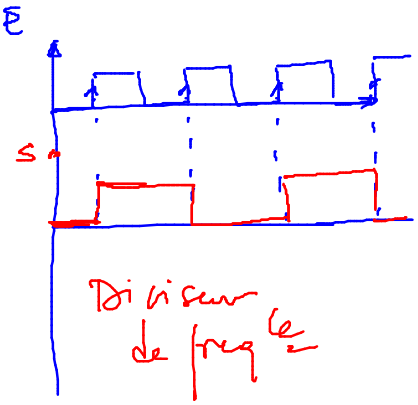


à $t=0$
$Q=0$

E/1

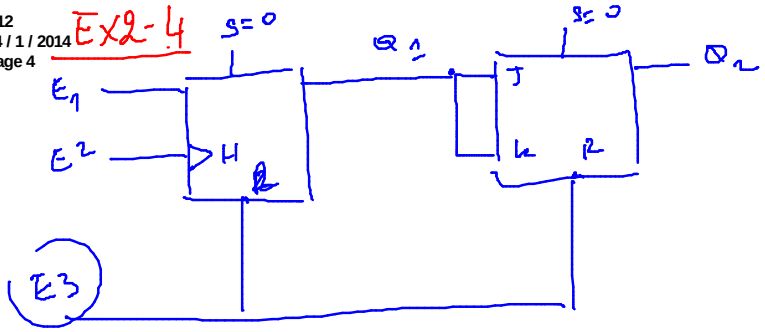


A $t=0$; $Q=0$

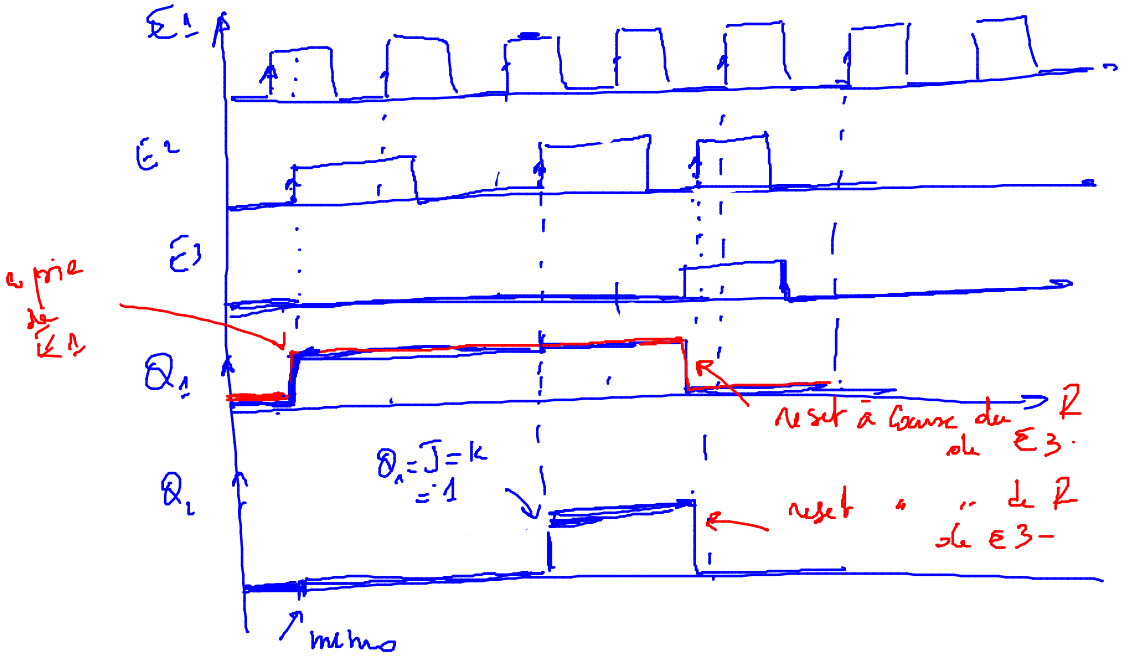


Diviseur de fréq

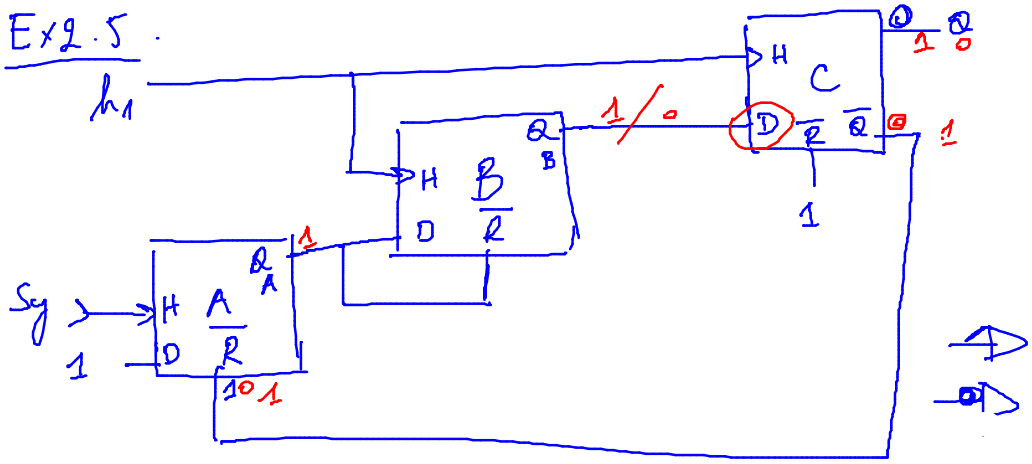
Ex2-4

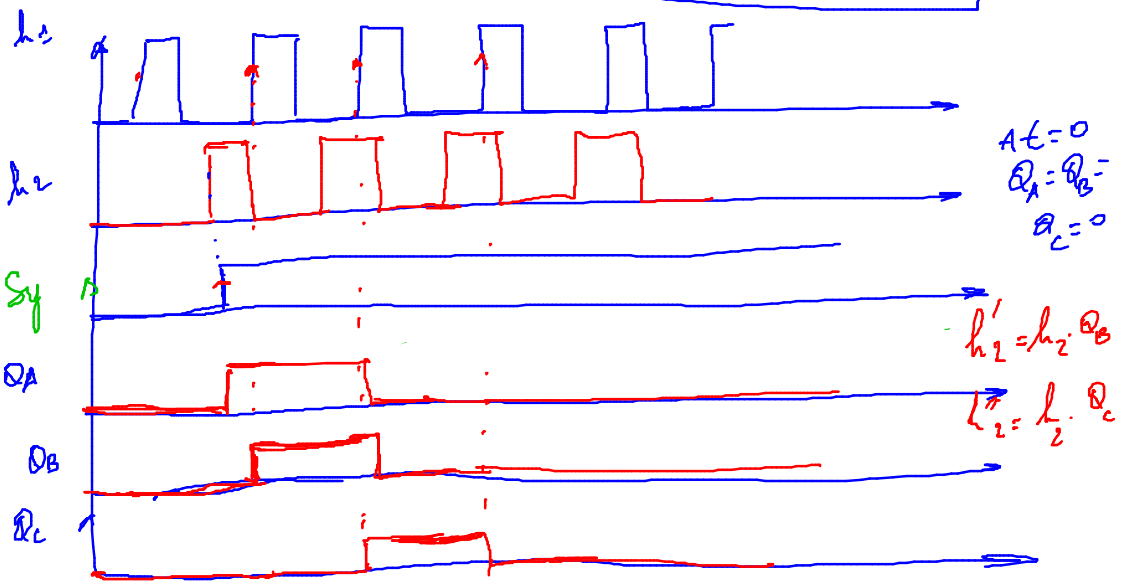


(S=0, R=1) → reset



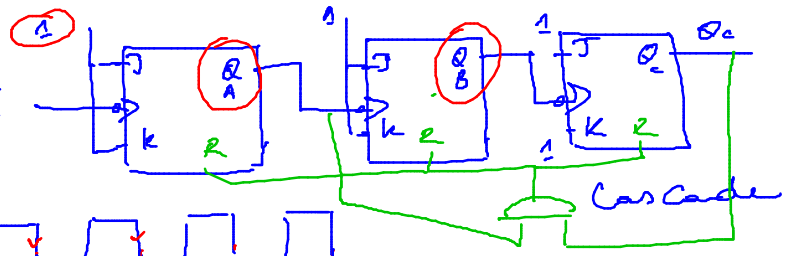
Ex2.5

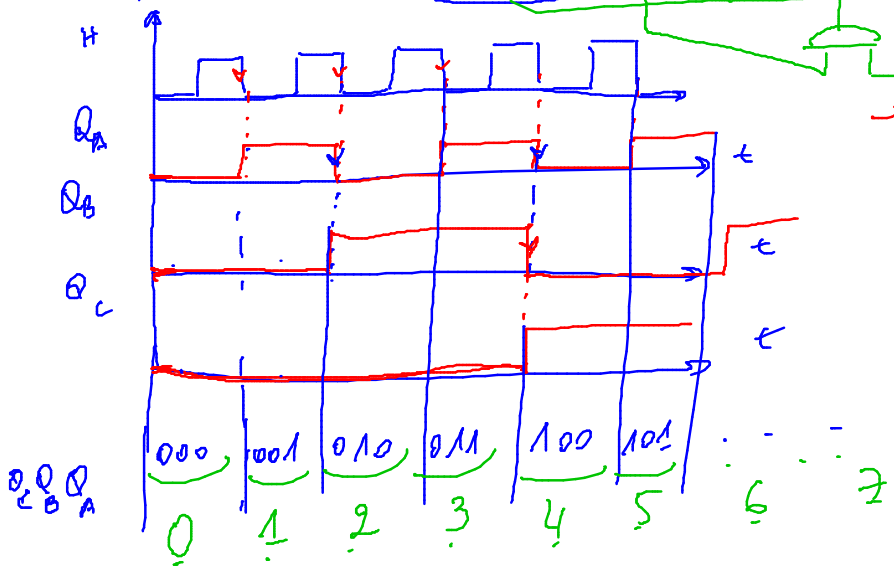

h1
Sy
1
H
D
A
R
QA
1
10 1
H
D
B
R
QB
1/0
H
C
D
R
Q
1
0
1
1


h1
h2
Sy
QA
QB
QC
At = 0
QA = QB =
QC = 0
h'2 = h2 · QB
h''2 = h2 · QC

Seq 3

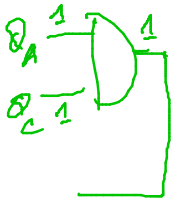
Compteur





à $t=0$

$Q_A = Q_B = Q_C = 0$



Reset

100
101
110
111

Compteur (Cascade) asynchrone

Décompteur / Modulo

8 états

3

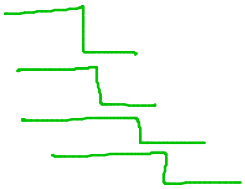
2

$Q_C Q_B Q_A$

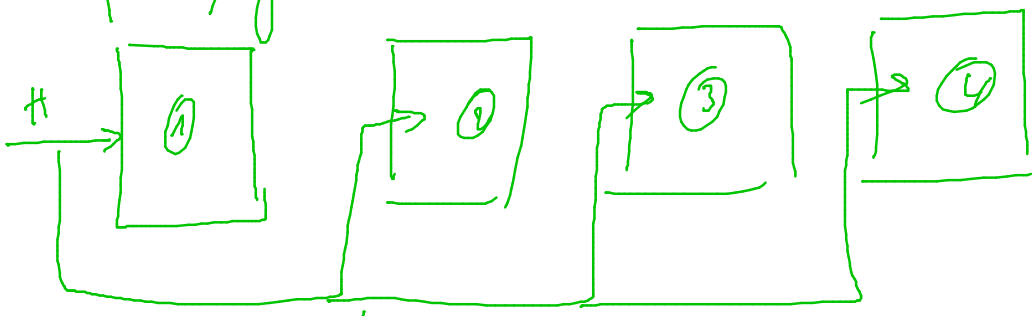
Compteur modulo 5 : 101

Compteur asynchrone.

limites.



Compteur synchrone voir Ex. 3-1



Q_C	Q_B	Q_A	$J_A = K_A$	$J_B = K_B$	$J_C = K_C$
0	0	0	1	0	0
0	0	1	1	1	0
0	1	0	⋮	⋮	⋮
0	1	1	⋮	⋮	⋮
1	0	0	⋮	⋮	⋮
⋮					

$J_A = 1 \quad K_A = 1$

$J_B = Q_0$

$J_C = Q_0 \, Q_1$

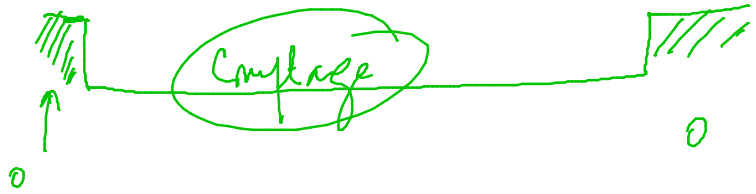
$J_D = Q_0 \, Q_1 \, Q_2$

Ex3. Système synchro.

Q_3	Q_2	Q_1	Q_0	$J_0 = K_0$	$J_1 = K_1$	$J_2 = K_2$	$J_3 = K_3$
				1	Q_0	$Q_0 Q_1$	$Q_0 Q_1 Q_2$
0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1				
⋮							

Ex3.1. Compteur synch. modulo 16

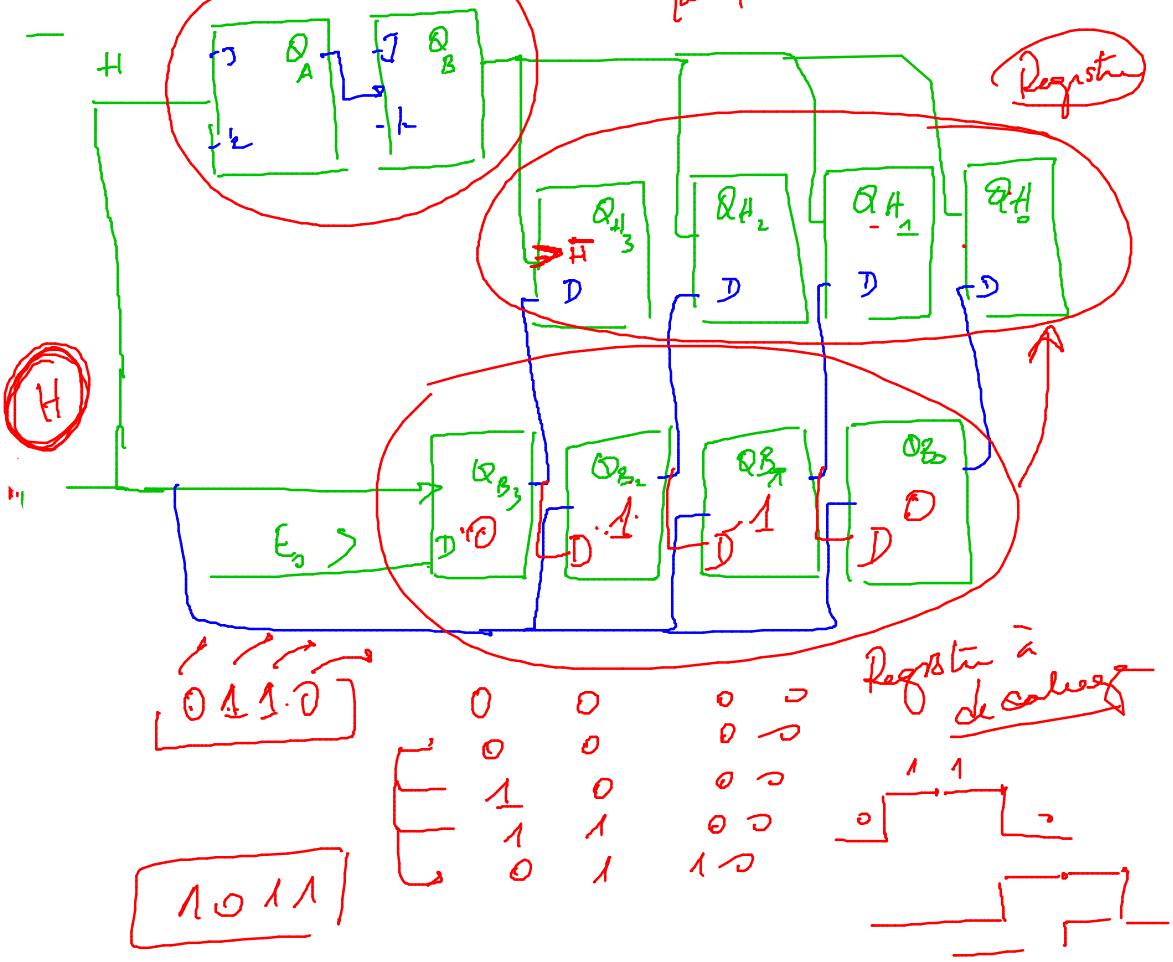
RAZ.



Ex : 3.2

Diviseur de fréquence par 4.

2^2



0 1 1 0

0	0	0	0
0	0	0	0
1	0	0	0
1	1	0	0
0	1	1	0

Registre à décalage

1 0 1 1

Séq4

q1) direct

q2) immédiat ~~5 = i ; MOV 5,[i]~~

q3) indirect ex [BX + SI]

q4) immédiat ex MOV [i], 5 — @ immédiat — @ direct

q5) immédiat

q6) [BX + SI + dépl] NB : déplacement est une constante

MOV AX, [BX + SI + 5] → accès au segment de données

q7) indirect et + précisément "basé, indexé avec déplacement"

q8) BX = registre de base

q9) SI = registre d'index

q10) [BX + ~~SI + 5~~]

⟹ [BX] = @age basé

q11) [~~BX + SI~~ + 5]

⟹ [5] = adressage direct

q12) [~~BX +~~ SI + 5] = adressage indexé avec déplacement

q13) [BP + SI + 5]

⟹ accès au segment de pile si le registre de base est BP

Exo 4.2

q1) LDS et LES

LDS charge un couple de registres (DS et un registre offset)

ex: LDS BX, [pointeur far]

NB: un pointeur far contient une adresse segmentée

La partie offset est chargée dans BX
segment DS

q2) LEA (Load Effective Address)

LEA BX, [BX+SI+5]

la somme des adresses BX, SI et 5 est chargée dans BX

ex : LEA BX, [2000]
2000 est chargé dans BX

NB : LEA n'accède pas à la mémoire

q3) L'adresse effective est une adresse offset

q4) ADD opg, opd, le resultat est stoché dans l'opérande gauche

opg ← opg + opd

q5)

AL = | 0101 | 1010 | → 5A

& | 0000 | 1111 | → 0F

= 00001010

AL = | 0 | A |

95 suite)

AL = | 0101 | 1010 |

4 décalages à droite
SHR = Shift Right

0 → | 0000 | 0101 | perdus

Exo 4.2 : Nombres signés

b_7							b_0
1	1	1	1	1	1	1	1

$$n = \underbrace{-b_7 \times 2^7}_{-128} + \underbrace{b_6 \times 2^6 + \dots + b_1 \times 2^1 + b_0 \times 2^0}_{\substack{111.1111 \\ 64+32+\dots+2+1 \\ 127}}$$

$$-128 + 127$$

$$-1$$

sur 2 octets

b_{15}															b_0
1	1	1													1

$$n = \underbrace{-b_{15} \cdot 2^{15}}_{-32768} + \underbrace{b_{14} \times 2^{14} + \dots + b_1 \times 2^1 + b_0 \times 2^0}_{+32767}$$

$$\underline{-32768 + 32767}$$

$$-1$$

93)

$$\begin{array}{rl} & 0111.1111.1111.1111 = 32767 \\ + & 0000.0000.0000.0001 = 1 \\ \hline = & \boxed{1000\ 0000\ 0000\ 0000} = -32768 \end{array}$$

aberration = erreur de calcul liée aux limites du pouvoir représentatif des variables

⚠ un nombre signé sur 16 bits

$-32768 \simeq +32767$

NB : $32768 = 2^{15}$

$\boxed{1 \mid 000.0000.0000.0000}$

32767

0111. 1111 1111 1111

TP2 : adr.c

#define constante 2

// AX = 1 (@ immédiat)

```
MOV AX, 1
```

// AX = constante (@ immédiat)

```
MOV AX, constante
```

// AX = variable1 (@ direct)

```
MOV AX, [variable1]
```

// AX = variable2 (@ basé sur BX)

```
MOV BX, offset variable2
MOV AX, [BX]
```

ou bien

```
LEA BX, [variable2]
MOV AX, [BX]
```

NB

int variable1;
int variable2

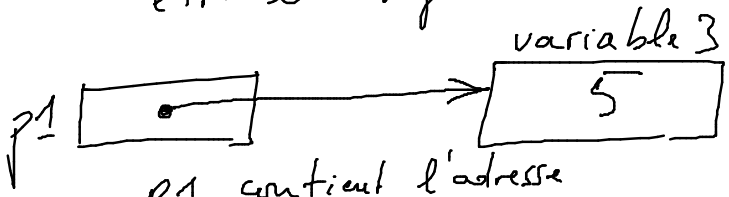
// p1 = & variable3; (via SI)

```
MOV  SI, offset variable3
MOV  [p1], SI
```

```
int  *p1;
int  variable3;
```

<u>NB</u>: LEA SI, [variable3]
convient aussi

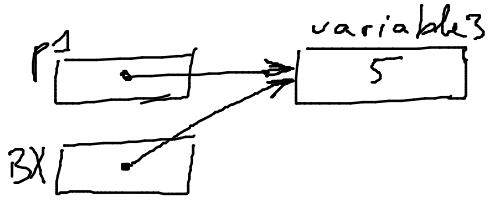
<u>NB</u>: ~~LEA AX, ...~~
~~LEA [p1], ...~~
l'opérande gauche de LEA doit être un registre d'adresse



p1 contient l'adresse de variable 3

// AX = *p1 ;

```
MOV  BX,[p1]
MOV  AX,[BX]
```



mais

MOV AX,[p1] n'accède pas à l'objet pointé par p1 mais à la valeur de p1 (même adresse)

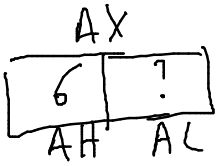
~~MOV AX,[[p1]]~~

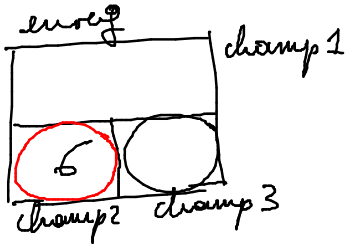
// AH = enreg. champ2 ; (@ Basé BX avec dépl)

MOV BX, offset enreg

MOV AH, [BX+ 2]

NB : typedef struct {
int champ1;
char champ2, champ3;
} rec_t;
rec_t enreg;





NB : on pouvait faire plus simple

MOV AH, [enreg + 2]

enreg : son adresse est connue lors de la compilation

la somme enreg + 2 est réalisée au moment de la compilation

Ex : si l'@ de enreg est 2000
l'@ de enreg.champ 2 est 2002

tout se passe comme si on avait écrit MOV AH, [2002]

NB : 2002 est une adresse offset qui se retrouve dans le code machine

```
// AX = i ; (@ indexé par SI)
LEA  SI,[i]
MOV  AX,[SI]

// AL = enreg. champ3 (@ indexé SI
                        + dépl)
MOV  SI, offset enreg

MOV  AL,[SI+3]
```

<u>NB:</u> 3 = taille du champ1
+ taille du champ2