TABLE DES MATIERES

1. Rappel d'arithmétique binaire : 3
1.1. système décimal : 3
1.2. système binaire : entier non signé sur 1 octet 3
1.3. système binaire : entier signé sur 1 octet 3
1.4. Système binaire : réel à virgule fixe sur 1 octet : 3
1.5. Système binaire : réel à virgule flottante sur 1 octet : 3
2. Structure interne du 8088 : 4
2.1. Les registres : 4
2.2. Le registre d'état (flags) : 4
2.3. L'E.U et la B.I.U : 5
2.4. Exemple d'exécution d'une instruction : 5
3. Les modes d'adressage : 7
3.1. Adressage immédiat : 7
3.2. Adressage direct : 7
3.3. Adressage indirect (ou adressage indexé) : 7
4. Les drapeaux (flags) du registre d'état :exemples 7
4.1. Addition binaire :exemple du ADD dest,srce 7
4.2. Explication sur les drapeaux changés par l'instruction ADD : 8
4.3. Addition binaire signée et non signée : 8
4.4. Algorithme d'addition non signée avec prise en compte du dépassement : 9
4.5. Algorithme d'addition signée avec prise en compte du dépassement : 9
5. Notion d'algorithmes : 10
6. Carte à 8088 : 12
6.1. Le décodage d'adresse : 12
6.1.1. Présentation : 12
6.1.2. Décodage par démutiplexeur 74LS138 : 12
6.1.3. Décodage par opérateur logique : 13
6.1.4. Décodage par réseaux logiques variables : 13
6.1.5. Décodage variable : 13
6.2. Le principe du bus multiplexé : 14
6.3. Exemple : 14
7. Les procédures : 15
7.1. Passages de paramètres par registres : 15
7.2. Passage de paramètre par la pile : 17

8. Gestion d'un processus : 18
8.1. Par attente active : 18
8.2. Par interruption : 20
9. Gestion de l'affichage d'un écran LCD :le LM020 21
9.1. Connexion de l'afficheur à la carte : 21
9.2. programmation de la carte 21

1. Rappel d'arithmétique binaire :

1.1. système décimal :

2	0	0
10^2	10^1	10^0

$= 2 \times 10^2 + 0 \times 10^1 + 0 \times 10^0$

1.2. système binaire : entier non signé sur 1 octet

1	0	0	1	0	0	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$= 2^7 + 2^4 + 2^0 = 128 + 16 + 1 = 145_{10} = x$

MSB LSB

$0 < x < 255$

1.3. système binaire : entier signé sur 1 octet

1	0	0	1	0	0	0	1
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$= -1 \times 2^7 + 2^4 + 2^0 = -128 + 16 + 1 = -111_{10} = x$

$-128 < x < 127$

Notion de complément à 2 notée CA2 :
Le CA2 permet le passage d'un nombre positif en son équivalent négatif . Il suffit de faire le complément de l'entier et de lui ajouter 1.

Exemple :passage de 65 à -65

```
          0 1 0 0 0 0 0 1
CA1 →     1 0 1 1 1 1 1 0
        +               1
          1 0 1 1 1 1 1 1 = -65
```

1.4. Système binaire : réel à virgule fixe sur 1 octet :

1	0	0	1	0	0	0	1
-2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
				0,5	0,25	0,125	0,062

= 1 0 0 1 , 0 0 0 1 = -8 + 1 + 0,062 = - 6,937 = x

$-8 < x < 7,937$

1.5. Système binaire : réel à virgule flottante sur 1 octet :

On définit un nombre flottant de la façon suivante : $x = \pm A \cdot 2^{\pm x}$

Avec : $\pm A$: mantisse
$\pm x$: exposant

Exemple pour 1 octet avec la mantisse codée sur 5 bits et l'exposant codé sur 3 bits :

1	0	0	1	0	0	0	1
-2^4	2^3	2^2	2^1	2^0	-2^2	2^1	2^0

Mantisse Exposant

$= 1\,0\,0\,1\,0\,0\,0\,1 = (-16 + 2) * 2^1 = -14 = x$

$-128 < x < 120$

2. ***Structure interne du 8088 :***

Le 8088 est un microprocesseur 8 bits externes et 16 bits internes (largeur du bus de données), alors que le 8086 est un microprocesseur 16 bits. La structure interne du 8088 est donné en annexe.

2.1. Les registres :

8 registres généraux 16 bits divisés en 2 groupes : (Cf Annexe)

Registres de travail	AX : accumulateur (résultat de l'opération) BX : pointeur pour la mémoire de données CX : compteur DX : brouillon

Chacun des registres de travail peut être scindé en deux parties. Ces registres sont destinés aux manipulations de données.

Registres d'adressage	SP : pointeur de pile BP : pointeur pour les manipulations sur la pile SI : pointeur source pour la manipulation de chaîne de car. DI : pointeur destination pour la manipulation de chaîne de car

Les registres d'adressage peuvent effectuer des opérations de manipulation de données et, en plus, participent aux opérations d'adressage indirect.

4 registres de segment 16 bits:

Registre de segment	CS : registre de segment de code DS : registre de segment de données ES : registre de segment de données supplémentaire SS : registre de segment de pile

Les registres de segment permettent de faire le lien entre le bus d'adresse externe de 20 bits et le bus d'adresse interne de 16 bits.

2.2. Le registre d'état (flags) :

Le registre d'état noté FLAGS permet de contrôler la véracité des opérations faîtes sur l'A.L.U. et est utilisé lors des sauts conditionnels. Il est constitué des drapeaux suivants :
CF : Carry Flag : retenue pour les opérations sur les entiers naturels
OF : Overflow : cas de débordement sur les entiers relatifs
ZF : Zero Flag : si le résultat de l'opération est egal à 0 alors ZF = 1
SF : Sign Flag : SF = 1 si le résultat est négatif (bit de poids fort du résultat de l'opération égal à 1)
PF : Parity Flag : PF = 1 si le nombre de 1 (du résultat) est pair.
AF : Auxiliary Flag : ajustement pour le calcul en BCD
IF : Interrupt Flag : permet de masquer les interruption . IF = 1
TF : Trace Flag : utilisé par le débugger.
DF : Direction Flag : utilisé par les opération de manipulation de caractères.

2.3. L'E.U et la B.I.U :

L'unité centrale est divisé en 2 unités :

- l'unité d'exécution notée E.U. comporte les registres généraux, l'A.L.U., le registre d'état, et reçoit les instructions au travers d'une file d'attente (instruction queue).
- l'unité d'interface de bus ou B.I.U. comporte les registres de segment et une mini A.L.U. qui calcule l'adresse sur 20 bits.

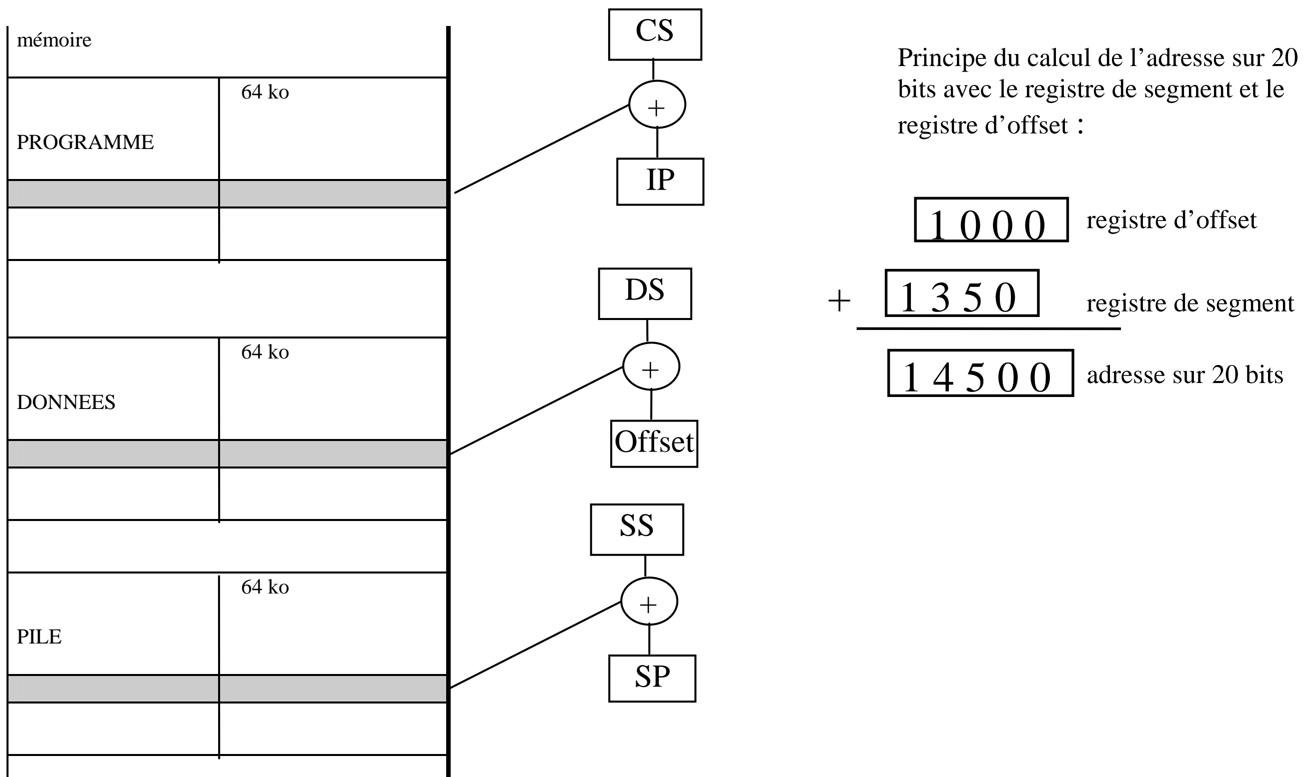


fig.1 :principe de l'adressage en mémoire par segment

2.4. Exemple d'exécution d'une instruction :

Nous allons expliquer le fonctionnement du CPU par un exemple simple :l'addition de 2 variables

mov ax, var1
add ax, var2
mov var3, ax

Après compilation du langage en assembleur, on suppose que le code exécutable a été chargé en mémoire et que le CS :IP a été initialisé sur le début du code exécutable. On se retrouve alors dans la configuration suivante :

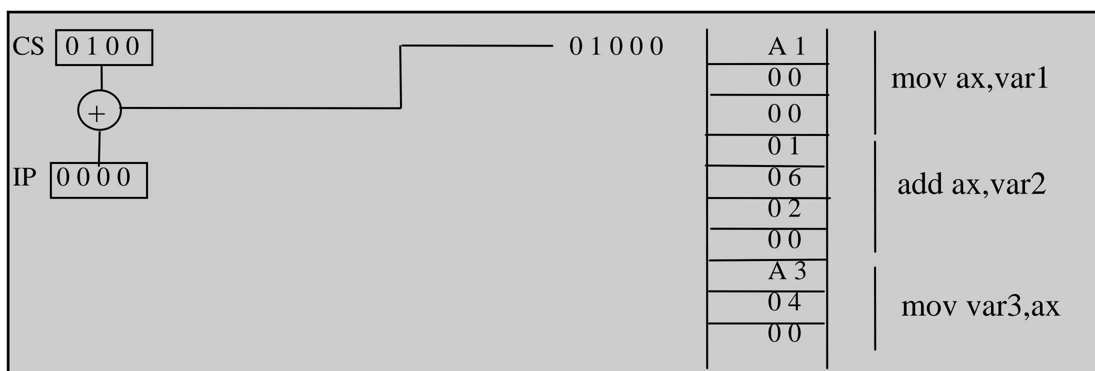


fig.2 :code exécutable à l'adresse 01000h

De plus le compilateur a alloué au variables var1, var2, var3 les espaces en mémoire données figure 3 :

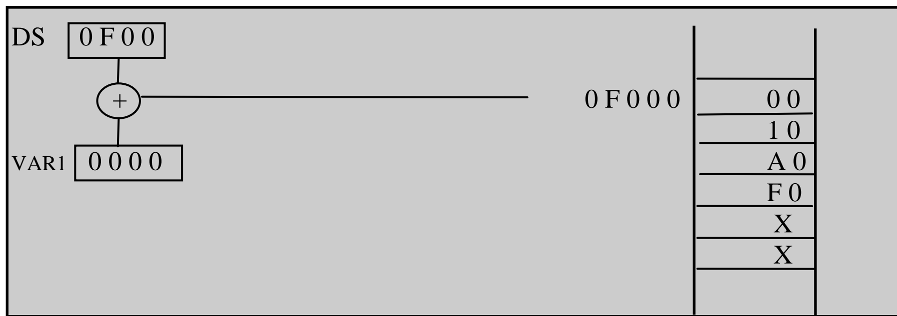


fig.3 :Données placées à l'adresse 0F000h

- <u>exécution de mov ax,var1</u> : mettre dans ax la valeur contenue à l'adresse donnée par var1.

FETCH	IR	←	M [CS : IP] ; IP = IP + 1
	MAR_L	←	M [CS : IP] ; IP = IP + 1
	MAR_H	←	M [CS : IP] ; IP = IP + 1
EXECUTE	AL	←	M [DS :MAR] ; MAR = MAR + 1
	AH	←	M [DS : MAR]

Après l'exécution de l'instrucion :
AX = 1000h
IP = 0003h

MAR : registre mémoire d'adresse. Ce registre est un des registres internes utilisés seulement par le CPU.

- <u>exécution de add ax,var2</u> :additionner le contenu de ax à la valeur contenue à l'adresse donnée par var2.

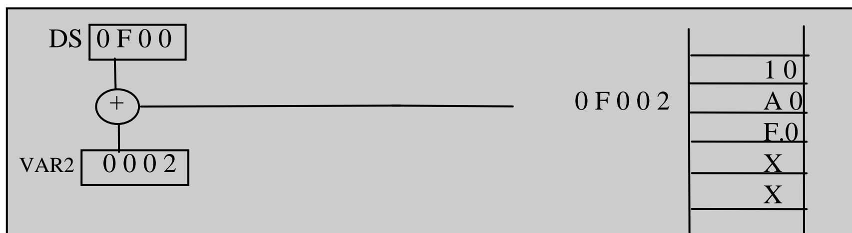


fig.4 :Données placées à l'adresse 0F002h

FETCH	IR_H	←	M [CS : IP] ; IP = IP + 1
	IR_L	←	M [CS : IP] ; IP = IP + 1
	MAR_L	←	M [CS : IP] ; IP = IP + 1
	MAR_H	←	M [CS : IP] ; IP = IP + 1
EXECUTE	TL	←	M [DS :MAR] ; MAR = MAR + 1
	TH	←	M [DS : MAR]
	AX	←	AX + T

Après l'exécution de l'instrucion :
AX = 00A0h
IP = 0007h
CF = 1

<u>exécution de mov var3,ax</u> :placer le contenu de ax dans la case mémoire pointée par var3.

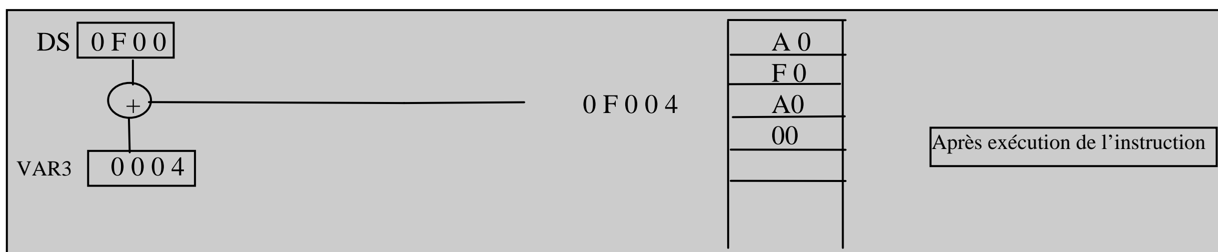


Fig.5 : résultat de l'addition

FETCH	IR	← M [CS : IP] ; IP = IP + 1
	MAR_L	← M [CS : IP] ; IP = IP + 1
	MAR_H	← M [CS : IP] ; IP = IP + 1
EXECUTE	M [DS :MAR] ← AL ; MAR = MAR + 1	
	M [DS : MAR] ← AH	

3. Les modes d'adressage :

3.1. Adressage immédiat :

mov ax, CA01h
mov al, 25
and al,10000000b

L'adressage immédiat n'est utilisable qu'avec les instructions suivantes : ADC,ADD,AND,CMP,OR,SBB,SUB,TEST,XOR

3.2. Adressage direct :

mov ax, var1 ; le compilateur remplace var1 par son offset. ax←M([DS] :[offset])
and bx,var2 ;bx←bx ET M([DS] :[offset])
lea DX,requete ;DX← offset de requete

On appelle offset la valeur de l'adresse contenue dans les 2 octets qui suivent le code opération.

3.3. Adressage indirect (ou adressage indexé) :

Différence entre l'adressage direct et l'adressage indirect :

Adressage direct : l'adresse (ou l'offset) de la valeur à lire (ou à écrire) se trouve dans le code exécutable (Cf page 5 :exemple d'exécution d'une instruction). L'adresse est donc figée et ne peut pas changer pendant l'exécution du programme.

Adressage indirect : l'adresse est contenue dans un registre et peut donc évoluer au cours du programme. Par exemple une lecture d'une zone mémoire sera faite avec un adressage indirect avec une incrémentation du registre pointant sur l'espace mémoire à lire.

mov ax, [bx] ; transfert dans (ax) du contenu de la mémoire pointée par [bx]
sub ax,[bx + 6] ;soustraction de (ax) avec le contenu de la mémoire pointée par [bx + 6]
add cl,mat[bx+si] ;addition de (cl) avec le contenu de la ligne (bx) et de la colonne (si) de la matrice d'offset mat

4. Les drapeaux (flags) du registre d'état :exemples

4.1. Addition binaire :exemple du ADD dest,srce

ADD dest,srce ; dest = dest + srce

Exemples :

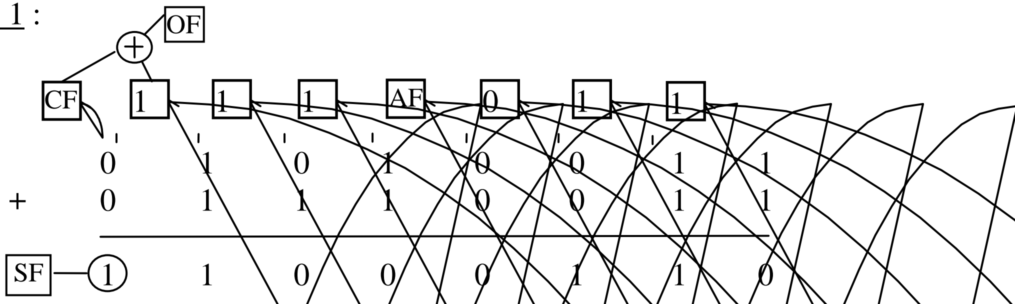
ADD	registre, registre	ADD CL, AL	ADD DX, SI	
ADD	mémoire, registre	ADD var1, AL	ADD var2, BX	ADD [BX], AL ADD [BX+SI], CX
ADD	registre, mémoire	ADD AL, var1	ADD AX, var2	ADD SI, mat[BX+DI]
ADD	registre, valeur	ADD DH, 5	ADD BP, 1039h	
ADD	mémoire, valeur	ADD var1, 01001001b	ADD [DI], 5	

modifié : OF CF SF ZF PF AF

4.2. ***Explication sur les drapeaux changés par l'instruction ADD :***

- CF retenue du bit 7 (MSB → MSB + 1)
- OF ou exclusif entre la retenue du bit 6 et la retenue du bit 7
- SF valeur du bit 7 (MSB)
- AF : retenue de bit 3 → bit 4
- ZF résultat = 0
- PF nombre pair de 1

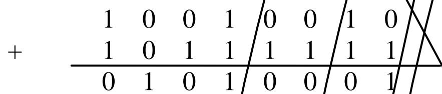
exemple 1 :



Résultat :

OF	CF	SF	ZF	PF	AF
1	0	1	0	1	0

exemple 2 :



Résultat :

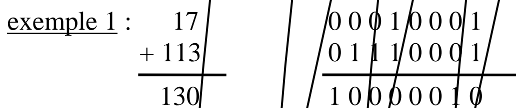
OF	CF	SF	ZF	PF	AF
1	1	0	0	0	1

Après une addition de nombres signés ***OF = 1*** si la somme de 2 nombres positifs donne un résultat négatif ou si la somme de 2 nombres négatifs donne un résultat positif.

4.3. ***Addition binaire signée et non signée :***

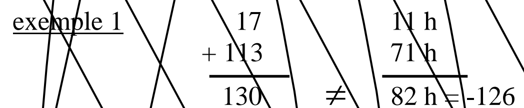
Addition de 2 nombres binaires non signés	***Addition de 2 nombres binaires signés***

exemple 1 :



CF = 0 ***résultat correct***

exemple 1

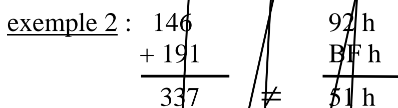


OF = 1

SF = 1

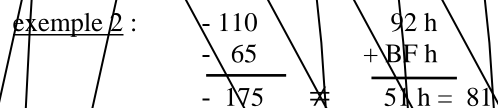
dépassement :

exemple 2 :



CF = 1 ***dépassement***

exemple 2 :



OF = 1

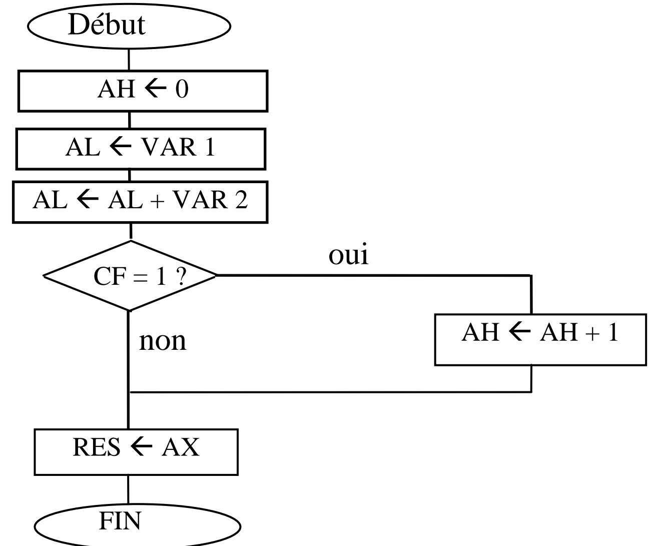
SF = 0

dépassement

Les deux exemples vus précédemment nous montrent que les effets de dépassement ne sont pas les mêmes pour une addition de nombres non signés et de nombres signés. Lors d'un calcul il faudra prendre en compte les dépassements en fonction du type des valeurs binaires utilisées.

4.4. Algorithme d'addition non signée avec prise en compte du dépassement :

```
donnees segment
VAR1,VAR2     db        15,115
RES           dw        ?
donnees ends

piles         segment
              dw        50    dup(?)
piles         ends

code    segment
        assume cs :code, ss :pile, ds :donnees
add_non_signe     proc
                        MOV AH, O ; ou sub ah ah
                        MOV AL, VAR1
                        ADD AL , VAR 2
                        JNC non_dépassement
                        INC AH
non_dépassement :       MOV RES, AX    ; 0 < AX < 510
                        RET
add_non_signe     endp

main :    ..........
code            ends
```

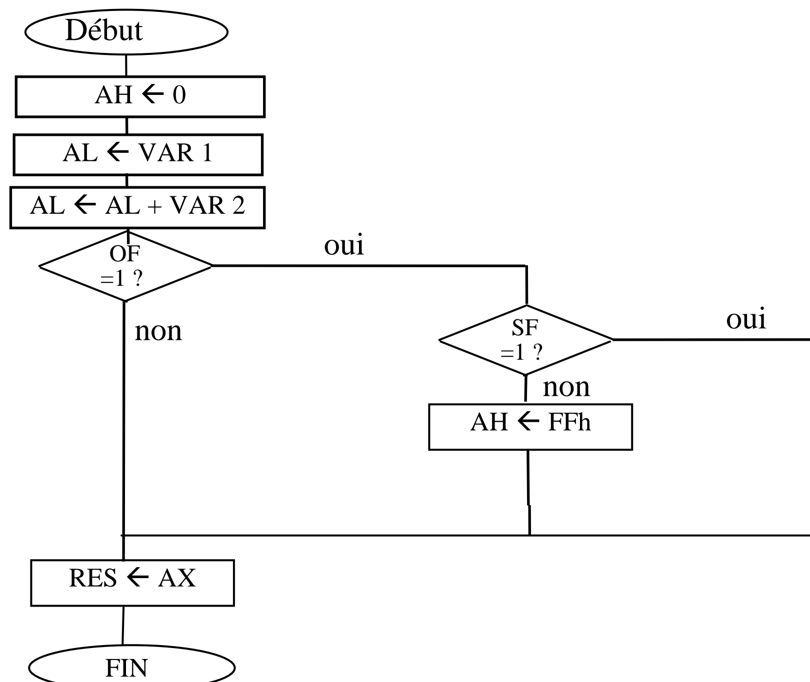


4.5. Algorithme d'addition signée avec prise en compte du dépassement :

Si OF =1 alors il y a dépassement. Pour corriger le résultat, il faut utiliser le drapeau de signe SF.

Si SF = 1 alors c'est la somme de 2 nombres positifs qui doit donner un résultat positif : AH =00h

Si SF = 0 alors c'est la somme de 2 nombres négatifs qui doit donner un résultat négatif :AH =FFh



Remarque : dans cet exemple, comme dans l'exemple précédent l'addition de 2 octets est placé dans 1 mot de 16 bits.

```
donnees       segment
VAR1,VAR2     db       15,115
RES           dw       ?
donnees       ends

piles         segment
              dw       50    dup(?)
piles         ends

code     segment
         assume cs :code, ss :pile, ds :donnees
add_signe          proc
                   MOV AH, O ; ou sub ah ah
                   MOV AL, VAR1
                   ADD AL , VAR 2
                   JNO    fin              ; saut si OF = 0 soit pas de dépassement
                   JGE    fin              ; saut si SF = OF =1 soit AX positif
                   MOV    AH, FFh          ; sinon AX est négatif
fin            :   MOV RES, AX             ; - 256 < AX < 255
                   RET
add_signe          endp

main :    ..........
code          ends
```

5. Notion d'algorithmes :

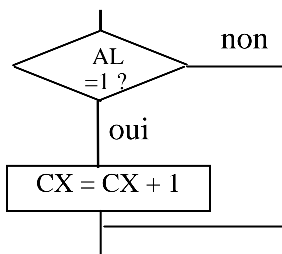
L'analyse d'un problème peut se faire de 2 façons : par l'utilisation d'un langage évolué ou par l'utilisation d'un algorigramme. La première méthode a l'avantage d'être simple et facile à écrire. Par contre la transcription en langage assembleur n'est pas aisée. L'algorigramme est lui lourd à écrire mais colle mieux au langage assembleur. Ceux sont donc les 2 méthodes qui sont données ci-dessous afin que chacun puisse utiliser l'une ou l'autre méthode lors de son analyse
.

Langage évolué

1/ si .... alors........ :

si AL = 1 alors
CX = CX + 1

Algorigramme



Assembleur

```
         cmp    AL,1
         jne    fin_si
         inc    CX
fin_si :
```

Langage évolué

2/ si .... alors........sinon...... :

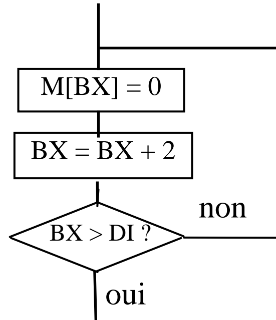
si M[BX] < 0
alors AX = AX + 1
sinon CX = CX - 1

```
        jnb    sinon
        inc    AX
        jmp    fin_si
sinon : dec    CX
fin_si :
```

<u>3/ Structures itératives :</u>

3.1/répéter ...... jusqu'à

Répéter
M[BX] = 0
BX = BX + 2
jusqu'à BX > DI




```
repete : mov    word ptr [BX],0
         add    BX,2
         cmp    BX,DI
         jbe    repete
```

3.2/Tant que ..... faire ..

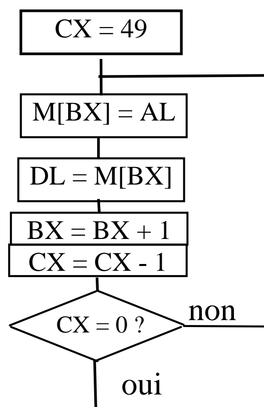
Tant que BX ≤ DI
M[BX] = 0
BX = BX + 2
fin_tant_que

algorigramme peu représentatif
La structure ne s'y prête pas

```
tant_que :  cmp BX,DI
            jnbe fin_tt_que
            mov word ptr[BX],0
            add   BX,2
            jmp   tant_que
fin_tt_que :
```

3.3/Pour i = ... faire ..

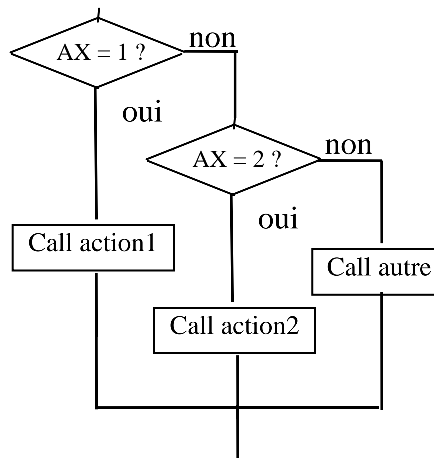
Pour i = 1 à 50 faire
M[BX] = AL
DL = M[BX]
BX = BX + 1
Fin_pour




```
           mov   CX,49
repeter :  mov byte ptr[BX],AL
           mov DL,byte ptr[BX]
           inc  BX
           loop repeter
```

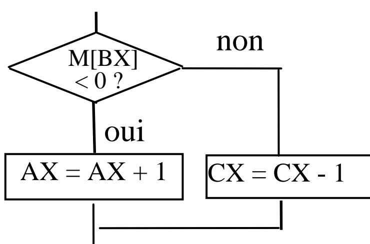
<u>4/ Sélections multiples ::</u>

Cas où :
AX = 1 : action1 ;break
AX = 2 :action2 ;break
AX > 2 :autre
fin_cas_où




```
         cmp   AX,1
         jne   suiv1
         call  action1
         jmp   fin_cas_ou
 suiv1 :cmp   AX,2
         jne   suiv2
         call  action2
         jmp   fin_cas_ou
 suiv2 : call autre
fin_cas_ou :
```

Algorigramme ***Assembleur***




```
cmp word ptr [BX] , 0
```

6. Carte à 8088 :

Le 8088 est un microprocesseur 16 bits internes, avec 8 bits de données externes multiplexées avec les 8 bits d'adresses de poids faibles (LSB). Le bus d'adresse est sur 20 bits, soit un espace mémoire adressable de 1Mo.

6.1. Le décodage d'adresse :

6.1.1. Présentation :

Lors du fonctionnement d'un système micro-programmé, le microprocesseur doit connaître les adresses des différents éléments . Pour cela les constructeurs donnent un "mapping mémoire" ou "cartographie mémoire" de leur système afin que l'utilisateur puisse programmer .

Exemple :

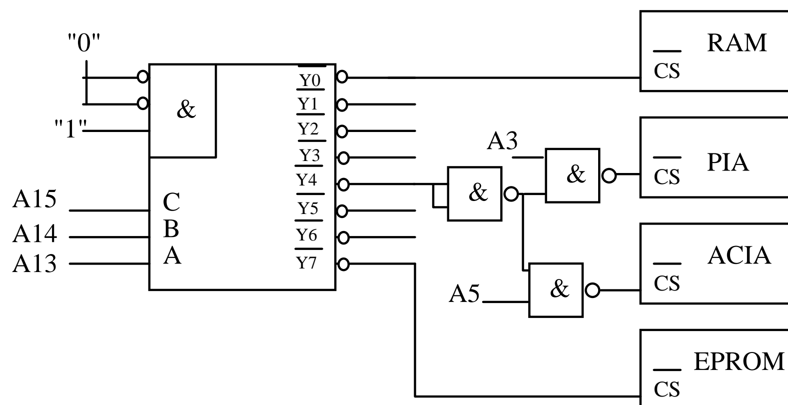
Capacité mémoire : EPROM 4 ko, ACIA 2 octets PIA 4 octets RAM 8 ko

Zone	Adresses
EPROM	$FFFF – $F000
ACIA 6850	$8221 – $8220
PIA 6821	$820B – $8208
RAM	$1FFF – $0000

	A15 A14 A13 A12	A11 A10 A9 A8	A7 A6 A5 A4	A3 A2 A1 A0	
EPROM	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	$FFFF
	1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0	$F000
ACIA	1 0 0 0	0 0 1 0	0 0 1 0	0 0 0 1	$8221
	1 0 0 0	0 0 1 0	0 0 1 0	0 0 0 0	$8220
PIA	1 0 0 0	0 0 1 0	0 0 0 0	1 0 1 1	$820B
	1 0 0 0	0 0 1 0	0 0 0 0	1 0 0 0	$8208
RAM	0 0 0 1	1 1 1 1	1 1 1 1	1 1 1 1	$1FFF
	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	$0000

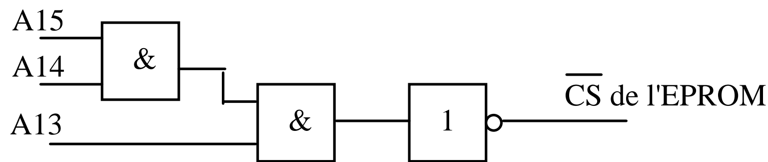
6.1.2. Décodage par démutiplexeur 74LS138 :

Le 74LS138 sélectionne 1 ligne parmi huit .



6.1.3. Décodage par opérateur logique :

Utilisation d'un assemblage d'inverseurs et d'opérateurs "ET", le champ d'adressage est fixé lors de la conception de la carte .



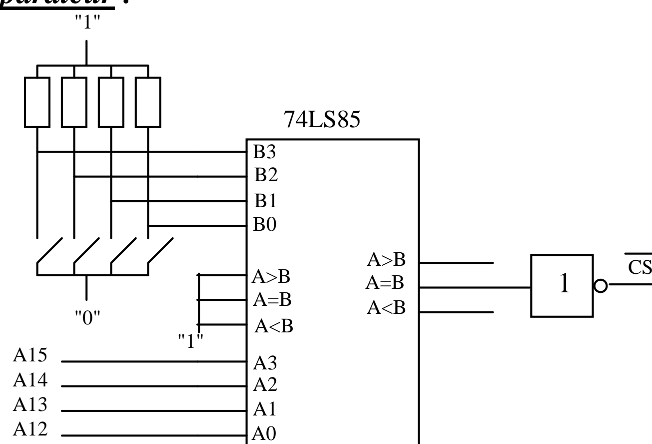
6.1.4. Décodage par réseaux logiques variables :

Ce type de décodage est utilisé lorsque l'organisation interne de l'espace mémoire est complexe, ou lorsque le fabriquant veut protéger son produit contre les reproductions éventuelles .Utilisation de PAL.

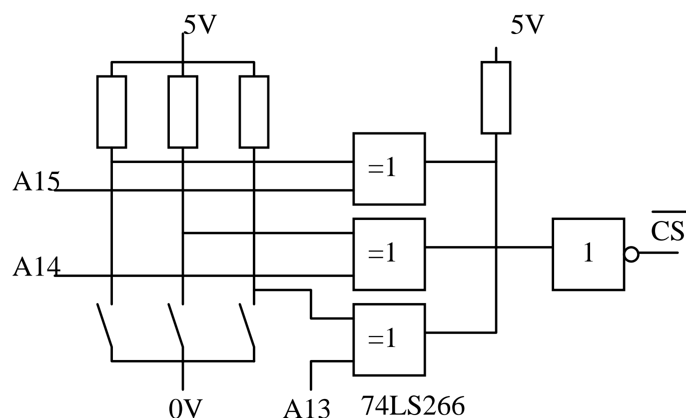
6.1.5. Décodage variable :

Il est parfois nécessaire que l'utilisateur puisse fixer lui même l'adresse d'un circuit ou d'une carte par des straps ou des interrupteurs . Ce principe est utilisé dans les systèmes susceptibles de recevoir plusieurs cartes d'extension .

Utilisation d'un comparateur **:**



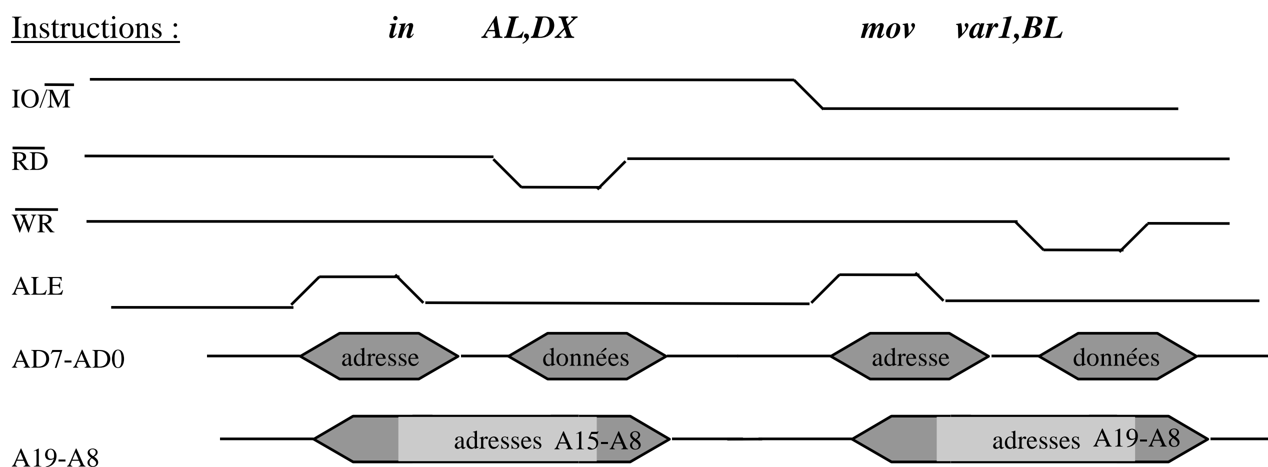
Utilisation d'opérateurs logiques **:**



6.2. Le principe du bus multiplexé :

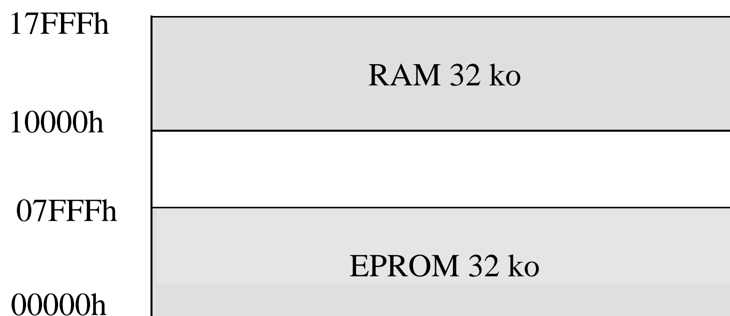
Le chronogramme d'un accès mémoire pour récupérer la données à lire ou à écrire est donné ci-dessous. Nous remarquerons le principe du bus multiplexé commandé par la broche ALE (broche 25). De plus il existe 2 instructions principales pour les accés mémoires :

- le ***mov dest,srce*** qui place la broche ***IO/M*** (broche 28) à ***0*** et qui donne accès à l'espace mémoire d'1 Mo.
- le ***in srce*** ou le ***out srce*** qui place la broche ***IO/M à 1*** et qui donne accès à l'espace mémoire de 64 ko (adressage des port E/S).



6.3. Exemple :

Nous désirons construire une carte à base de 8088 avec un espace adressable donné ci-dessous :



Cet espace mémoire sera lu ou ecrit par l'instruction ***mov***, espace adressable sur 1Mo.
Compléter la carte donnée page suivante :

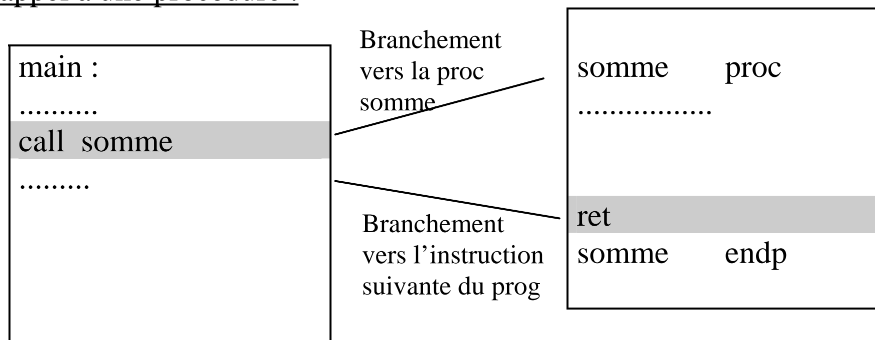
- faire le décodage d'adresses en utilisant soit des portes NAND, soit le décodeur 3 vers 8 (74HC138 donné en ANNEXES),
- compléter le câblage des broches 1 et 11 du 74HC573 (8 bascules D, Cf ANNEXES)

7. Les procédures :

Dans un programme en langages évolué (Pascal, C ...) comme dans un programme en assembleur, il y a toujours des bouts de code qui sont utilisés plusieurs fois. Il est alors intéressant de les placer dans une procédure (assembleur ou Pascal) ou une fonction (langage C). De plus, une programmation « intelligente » se doit d'être ***modulaire*** afin que la lecture du programme se fasse sur 1 ou 2 écrans maximums.

Il est donc nécessaire d'utiliser les procédures et ceci de façon quasi-systématique.

Principe d'appel à une procédure :



Lors de l'appel à une procédure par l'instruction ***call offset_proc***, le CPU sauvegarde le pointeur de programme IP dans la pile et fait IP = offset_proc (offset_proc est l'adresse en mémoire de la procédure somme)

A la fin de la procédure, l'instruction ***ret*** récupère IP dans la pile et se branche vers l'instruction suivante du programme principal.

Rem : il est possible de faire un appel à une procédure en dehors du segment de code CS. Dans ce cas, on utilise l'instruction ***call far*** .

Dans la majorité des cas, une procédure est associée à des paramètres d'entrée et de sortie. En Pascal, somme (res,3,5) calculera par exemple 3+5 et placera le résultat dans la variable ***res***. En langage C, c'est l'instruction res = somme (3,5) qui donnera le même résultat.

En assembleur, la syntaxe sera moins limpide, mais le principe reste le même. Il existe 2 solutions pour passer des paramètres à une procédure ;

- le passage de paramètres par registres,
- le passage de paramètre par la pile.

7.1. Passages de paramètres par registres :

Un exemple est donné à la page suivante.

Cette méthode est :

- la plus simple à utiliser.
- utilisée si le nombre de paramètres n'est pas trop important

```
title       passage de parametres par registres
;           nombres d'elements négatifs d'un vecteur
;
;           Nom du fichier :     neg_reg.asm
;-----------------------------------------------------------------
;
donnees         segment                                 ;definition des donnees
vecteur dw      5,-2,9,-5,3,17,45,-21,40,78
taille          dw      10
negatif  dw     ?
donnees         ends
;-----------------------------------------------------------------
pile            segment         stack                   ; definition de la taille de la pile
                dw      100     dup( ?)
pile            ends
;-----------------------------------------------------------------
;
code            segment                                 ; debut du programme
                assume    cs :code, ss :pile,ds :donnees          ;affectation de cs et ss
;
;-----------------------------------------------------------------
;           Procedure nb_elem_neg
;
;           But : recherche le nombre d'element negatifs dans un tableau
;                   d'entiers relatifs codees sur 16 bits
;
;           entree : bx = offset du tableau
;                           cx = nombre d'elements du tableau
;
;           sortie :  ax = nombre d'elements negatif

;-----------------------------------------------------------------
;
nb_elem_neg     proc
                sub     ax,ax                           ; initialisation de ax
pour :          cmp     word ptr  [bx] , 0              ; 'element est-il
                jge     positif                         ; positif ou nul ?
                inc     ax ax                           ; non, incrementer le compteur
positif :
              add       bx,2                            ; oui, incrementer le pointeur
                loop    pour                            ; et recommencer
                ret                                     ; retour procedure

nb_elem_neg     endp

;-----------------------------------------------------------------
;           Programme principal utilisant la procedure nb_elem_neg
;           envoie l'offset du tableau et le nombre d'element
;           recoit le nombre d'element negatif dans AX
;-----------------------------------------------------------------
main :          mov     ax,donnees                      ; initialisation du registre
                mov     ds,ax                           ; de segment de donnees

                lea     bx,vecteur                      ; bx = offset de tableau
                mov     cx,taille                       ; nombre d'element du tableau
                call    nb_elem_neg                     ; appel à la procedure
                mov     negatif,ax                      ; sauvegarde du résultat
                exit
code            ends
```

7.2. Passage de paramètre par la pile :

Cette méthode est :

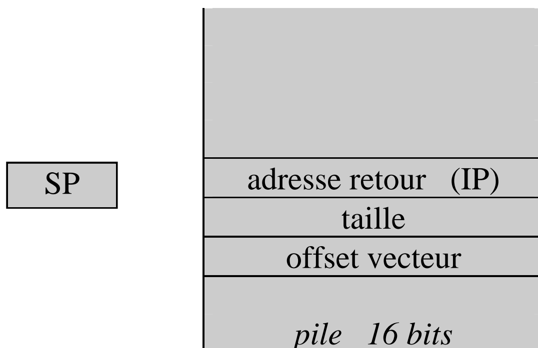
- utilisée par les compilateurs
- plus lente que la méthode précédente
- utilisable dans tous les cas.

```
title       passage de parametres par la pile
;           nombres d'elements négatifs d'un vecteur
;
;           Nom du fichier :    neg_pile.asm
;-----------------------------------------------------------------
;donnees          segment                                ;definition des donnees
vecteur dw        5,-2,9,-5,3,17,45,-21,40,78
taille            dw      10
negatif  dw       ?
donnees           ends
;-----------------------------------------------------------------
pile              segment         stack                  ; definition de la taille de la pile
                  dw      100     dup( ?)
pile              ends
;-----------------------------------------------------------------
;
code              segment                                ; debut du programme
                  assume    cs :code, ss :pile ,ds :donnees          ;affectation de cs et ss
;;-----------------------------------------------------------------
;           Procedure nb_elem_neg_pile
;
;           But :  même but que nb_elem_neg
;
;           entree : parametres passes par la pile,  offset du tableau, nombre d'elements
;
;           sortie :            ax = nombre d'elements negatif
;-----------------------------------------------------------------
;
nb_elem_neg_pile        proc
                  push    bp                             , sauvegarde de bp qui devient
                  mov     bp,sp                          ; le pointeur pour l'acces à la pile
                  push    bx                             ; sauvegarde des registres
                  push    cx                             ; utilises dans la procedure

                  mov     bx, [ bp + 6 ]                 ; acquisition de l'offset du tableau
                  mov     cx, [ bp + 4 ]                 ; acquisition du nombre d'element

                  sub     ax,ax                          ; initialisation de ax
pour :            cmp     word ptr  [bx] , 0             ; 'element est-il
                  jge     positif                        ; psitif ou nul ?
                  inc     ax                             ; non, incrementer le compteur
positif :
                  add     bx,2                           ; oui, incrementer le pointeur
                  loop    pour                           ; et recommencer

                  pop     cx                             ; restauration du contexte
                  pop     bx
                  pop     bp
                  ret                                    ; retour procedure
nb_elem_neg       endp
```

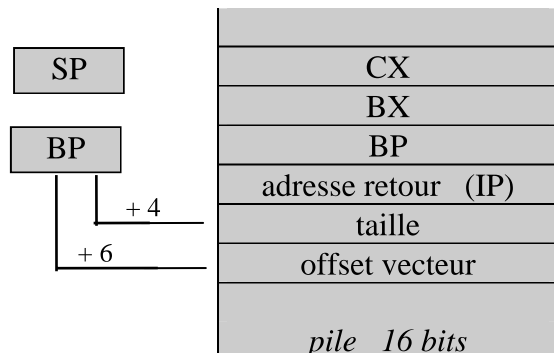
```
;----------------------------------------------------------------
;         Programme principal utilisant la procedure nb_elem_neg
;         envoie l'offset du tableau et le nombre d'element
;         recoit le nombre d'element negatif dans AX
;----------------------------------------------------------------

main :          mov     ax,donnees              ; initialisation du registre
                mov     ds,ax                   ; de segment de donnees

                lea     ax,vecteur              ; bx = offset de tableau
                push    ax                      ; passage par la pile
                push    taille                  ; nombre d'elements passes par la pile
                call    nb_elem_neg_pile        ; appel à la procedure
                mov     negatif,ax              ; sauvegarde du résultat
                exit
code            ends
```

Explications du passage par la pile :

Etat de la pile à l'entrée de la procédure : après le call nb_elem_neg_pile.



Etat de la pile après l'instruction push cx dans la procédure nb_elem_neg_pile



8. Gestion d'un processus :

Une carte à microprocesseur ou à microcôntroleur prend (dans la majorité des cas) en charge la conduite d'un processus, que ce soit la gestion d'un clavier, la communication avec un PC ou un terminal, la commande d'un moteur, la commande d'un processus industriel (fonction d'automate), etc...

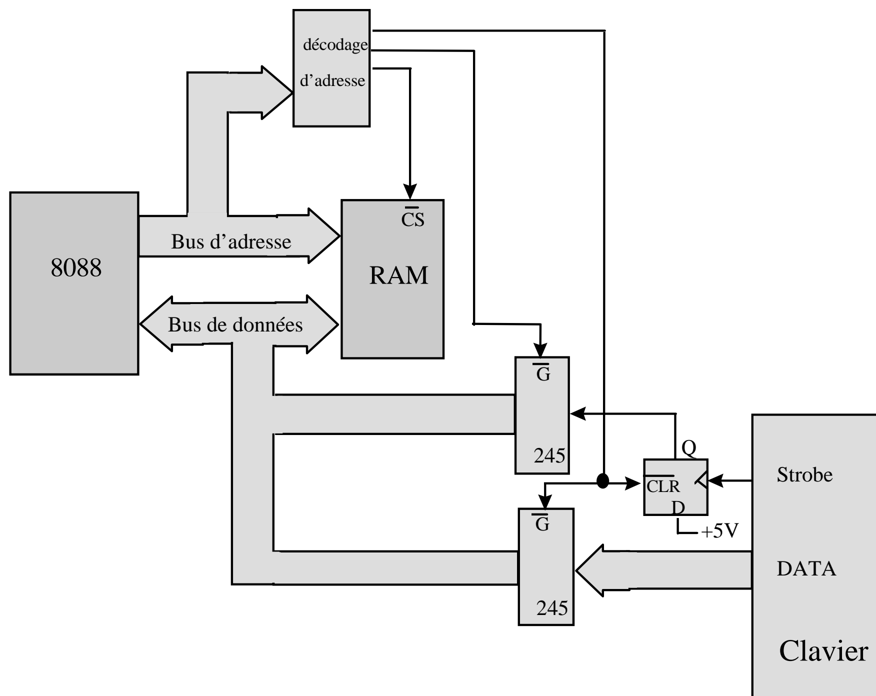
Cette commande peut s'effectuer de 2 façons :

- par attente active
- par interruption

8.1. Par attente active :

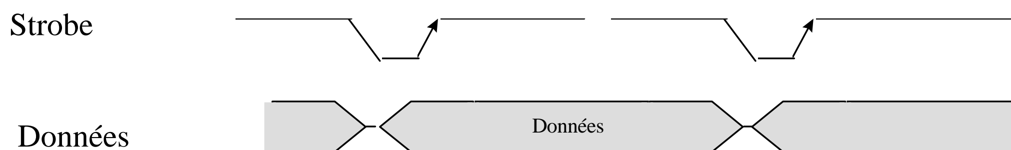
Prenons un exemple : pour cela nous utiliserons un clavier à liaison parallèle qui envoie des caractères de façon aléatoire avec un strobe pour la gestion de la communication.

Le principe du montage est donné à la page suivante.



lecture d'un clavier parallèle par attente active

Le chronogramme des signaux envoyés par le clavier est donné ci-dessous :



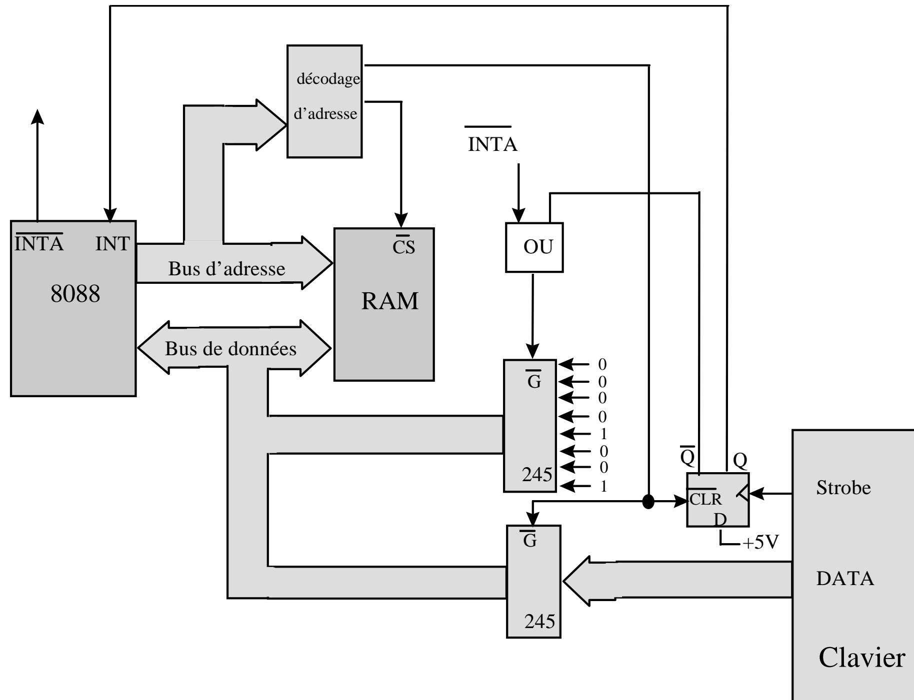
<u>Exercice :</u> en fonction du chronogramme, du schéma de principe et du programme de lecture du clavier, compléter le schéma donné à la page suivante (folio noté ATTENTE ACTIVE).Les documentations du 74HC245 et du 74HC74 sont données en ANNEXES .

La procédure de gestion du clavier est donné ci-dessous :

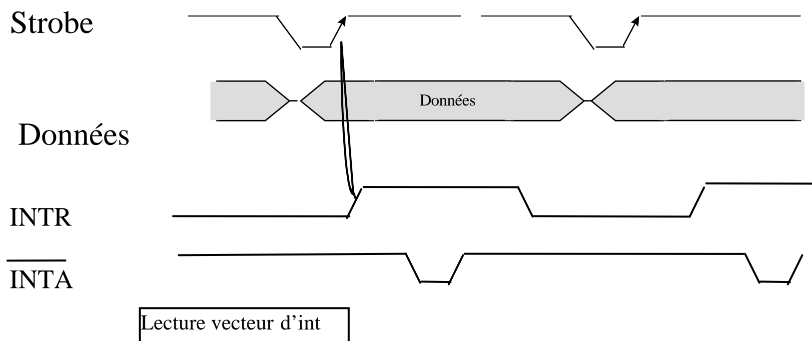
```
;---------------------------------------------------------------------
;          procedure      entree_clavier
;          But :    attente du changement d'etat du strobe
;                   puis lecture du caractere
;          sortie : AL= code ASCII du caractere
;-----------------------------------------------------------------------
entree_clavier  proc
                push   dx               , sauvegarde de dx
                mov    dx,0000h         ; chargement de l'adresse du strobe
attente :       in     al,dx            ; lecture de l'octet de commande
                test   al,128           ; si strobe = 1
                jnz    attente          ; alors attente
lecture :
                mov    dx,8000h         ; chargement de l'adresse des donnees
pour :          in     al,dx            ; lecture du caractere
                pop    dx               ; restaurztion de dx
                ret                     ; retour procedure
lecture_clavier endp
```

8.2. Par interruption :

En reprenant le même montage fait précédemment et après quelques petites modifications données ci-dessous, il est maintenant possible de gérer le processus par interruption :



Chronogramme :



Principe de fonctionnement : Au front montant du Strobe, le signal INTR passe à 1. Le CPU sauve alors le CS IP et les flags dans la pile puis place la broche $\overline{\text{INTA}}$ à 0. Le CPU lit alors le numéro du vecteur d'interruption sur le bus de données et se branche à CS= 4 * (numéro d'it) + 2 et IP = 4 * (numéro d'it)

La procédure d'interruption est la suivante :

```
;----------------------------------------------------------------------
;           procedure        interrup_clavier
;           But :     place en memoire le caractere lu
;----------------------------------------------------------------------


interrup_clavier            proc
```

```
                push    dx                              , sauvegarde de dx
                push    ax
                mov     dx,8000h                        ; chargement de l'adresse de la donnee
                in      al,dx                           ; lecture de l'octet de donnees
                mov     [si],al                         ; sauvegarde en mémoire
                inc     si                              ; pour donnee suivante
                pop     dx                              ;restauration de dx
                pop     ax                              ;restauration de ax
                iret                                    ; restaure CS,IP,flags
interrup_clavier        endp
;------------------------------------------------------------
;       initialisation du vecteur d'interruption
;       a placer en debut du programme principal
;------------------------------------------------------------
init_it proc
sub     ax, ax
mov     es, ax
mov     dx, offset interrup_clavier             ; chargement de l'offset de la
mov     es : [ 4 * 9 ], dx                      ; procedure d'it à l'adresse de l'it 9
mov     dx, seg interrup_clavier                ; chargement du segment de la
mov     es : [ 4 * 9 ], dx                      ; procedure d'it à l'adresse de l'it 9
ret
init_it endp
```

9. Gestion de l'affichage d'un écran LCD :le LM020

9.1. Connexion de l'afficheur à la carte :

Pour finir avec l'étude du 8088, nous allons compléter la carte déjà développée en lui rajoutant l'afficheur LM020. Un résumé de la documentation technique est donné en ANNEXE.

Pour cela, reprenez le schéma (folio noté attente active) et connecter les broches 4, 5 et 6 du LM020. Nous pourrons par exemple, prendre l'espace mémoire suivant :

- écriture ou lecture d'un mot de commande à l'adresse 08000h
- écriture ou lecture d'un caractère à l'adresse 88000h

Votre mal de tête ne passe pas ?

Passez donc au chapitre 9.2 :

9.2. programmation de la carte

Nous désirons nous servir de la carte pour effectuer des opérations (somme, soustraction, multiplication et division) sur des entiers.

Les nombres rentrés ne devront pas dépasser 9999, par contre on pourra effectuer des opérations successives et ceci quel que soit l'opérateur. Exemple d'opération réalisables par la carte : 32 + 58 -33 = .Dans un premier temps on ne prendra en compte que des nombres entiers positifs avec un résultat positif.

Donner un algorithme, donnant une ébauche du programme, en prenant soin de faire une découpe par procédure. Et bonne chance !