Corrigé de la série de TD N° 4

Corrigé de l'exercice 01 :

Apres exécution des instructions ci-dessous nous obtenons :

1) MOV AL, 32H
MOV BL, 79H
AND AL, BL **[AL = 30H]** (AND : 'ET' logique)

2) MOV AL, 55H
MOV BL, 79H
OR AL, BL **[AL = 7DH]** (OR : 'OU' logique)

3) MOV AL, 39H
MOVBL, 67H
XOR AL, BL **[AL = 5EH]** (XOR : OU exclusif)

4) MOV AL, 23H
NOT AL **[AL = DCH]** (Complément à 1)

5) MOV AL, 41H
NEG AL **[AL = BFH]** (Complément à 2)

6) MOV AL, 11001011B ; (CBH)
SHR AL, 1 (**SHR** : **S**hift **R**ight) décalage logique à droite

1	1	0	0	1	0	1	1	**avant**

0	1	1	0	0	1	0	1	**1**	**après**

65H CF = 1

7) MOV BL, 76H
DEC BL
SHR BL, 1

0	1	1	1	0	1	0	1	**avant**

0	0	1	1	1	0	1	0	**1**	**après**

3AH CF = 1

8) MOV AL, 59H
MOV CL, 3
SHL AL, CL (**SHL**: **S**hift **L**eft) décalage logique à gauche

0	1	0	1	1	0	0	1	**avant**

0	1	1	0	0	1	0	0	0	**après**

C8H CF = 0

NB: Dans le cas où le décalage (ou rotation) demandé est > 1 on utilise le registre CL

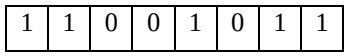
9) MOV AL, 11001011B
SAR AL, 1 (**SAR**: **S**hift **A**rithmetically **R**ight) décalage arithmétique à droite

1	1	0	0	1	0	1	1	**avant**

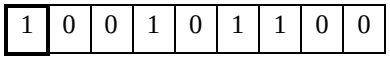
1	1	1	0	0	1	0	1	**1**	**après**

E5H CF = 1

10) MOV AL, 11001011B
MOV CL, 2
SAL AL, CL (**SAL**: **S**hift **A**rithmetically **L**eft) décalagearithmétique à gauche



avant

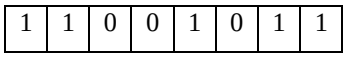


après

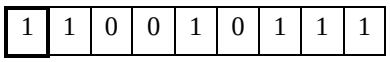
2CH CF = 1

NB: SHL ≡ SAL les deux instructions sont identiques
SHR ≠ SAR les deux instructions sont différentes

11) MOV AL, 11001011B
ROL AL, 1 (**ROL**: **Ro**tate **L**eft) rotation à gauche



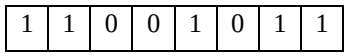
avant



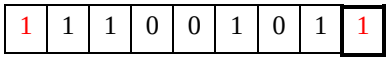
après

97H CF = 1

12) MOV AL, 11001011B
ROR AL, 1 (**ROR**: **Ro**tate **Right**) rotation à droite



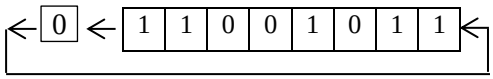
avant



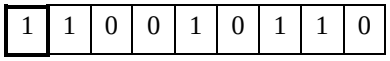
après

E5H CF = 1

13) CLC ; forçage du carry flag à '0' CF ←0
MOV AL, 11001011B
RCL AL, 1 (**RCL**: **R**otate through **C**arry **L**eft) rotation via le carry flag à gauche



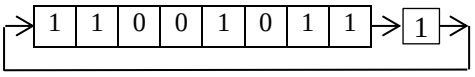
avant



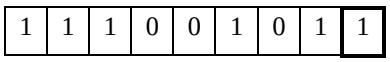
après

96H CF = 1

14) STC ; forçage du carry flag à '1' CF ←1
MOV AL, 11001011B
RCR AL, 1 (**RCR**: **R**otate through **C**arry **R**ight) rotation à via le carry flag à droite



avant



après

E5H CF = 1

Corrigé de l'exercice 02 :

Le contenu des registres généraux après l'exécution des instructions est le suivant :

1)
```
MOV   AL, 49H
MOV   BL, 17H
ADD   AL, BL ;        AL = 60H
```

2)
```
MOV   AL, 10H
MOV   BL, 20H
SUB   BL, AL   ;  BL ← BL-AL   BL = 10H
```

3)
```
MOV   AL, 10H
MOV   BL, 20H
SUB   AL, BL ;   AL = F0H
```

4)
```
MOV   AL, 6H
MOV   BL, 10H
XCHG  AL, BL ;  AL = 10H, BL = 6H
```
XCHG : E**xch**an**g**e, Permute le contenu des deux opérandes)

Corrigé de l'exercice 03 :

1)
```
MOV   AL,6H
MOV   BL,10H
MUL   BL   ; Le résultat de la multiplication est sauvegardé dans AX puisque l'opérande (OP = BL) est sur 8 bits , AX = AL × BL = 6H × 10H = 60H
```

2)
```
MOV   AX,6H
MOV   BX,10H
MUL   BX   ; Le résultat de la multiplication est sauvegardé dans DX:AX puisque l'opérande (OP = BX) est sur 16 bits, DX:AX = AX × BX = 60H DX=0000H et AX=0060H
```

a) Le résultat obtenu dans **1)** et le même que celui trouvé dans **2)**, mais la différence réside dans le format de représentation, le premier est représenté sur un mot (AX), par contre le deuxième est représenté sur un double mot (DX :AX).

NB : un mot contient 2 octets (16bits) et un double mot contient 2 mots ou 4 octets (32 bits)

3)
```
MOV   AL,10
NEG   AL          ; AL = F6H soit (-10 en décimal)
MOV   BL,100      ; BL = 100D = 64H
IMUL  BL          ; AX = AL × BL = F6H × 64H = FC18H      AX = FC18H
```

4)
```
MOV   AL,10
NEG   AL          ; AL = F6H soit (-10 en décimal)
MOV   BL,100      ; BL = 100D = 64H
MUL   BL          ; AX = AL × BL = F6H × 64H = 6018H      AX = 6018H
```

b) Le résultat de la multiplication dans 3) est diffèrent de celui obtenu dans 4), puisque, l'instruction '**IMUL**' traite les opérandes (AL et BL) comme des nombres entiers signés sur 8 bits, et fournie un résultat signé sur 16 bits. Cependant, l'instruction '**MUL**' traite les deux opérandes comme deux nombres entiers non-signés sur 8 bits, et fournie un résultat non-signé sur 16 bits.

5)
```
MOV   AX,804H     ; AX = 2052D
MOV   BL,82H      ; BL = 130D
DIV   BL          ; Q dans AL=0FH (15D) et R dans AH = 66H (102D)
```

6)
```
MOV   AX,804H     ; AX = 2052D
MOV   BL,82H      ; BL = -126D
IDIV  BL          ; Q dans AL=F0H (-16D) et R dans AH = 24H (36D) non-signé
```

c) Le quotient de la division dans 5) est diffèrent de celui obtenu dans 6), puisque, l'instruction '**IDIV**' traite les opérandes (AX et BL) comme des nombres entiers signés, et fournie un quotient signé sur 8 bits. Cependant, l'instruction '**DIV**' traite les deux opérandes comme deux nombres entiers non-signés, et fournie un quotient non-signé sur 8 bits.

7)
```
MOV   AX,5H
MOV   DX,10H
MOV   BX,100H
DIV   BX          ; le double mot DX:AX = 00100005H
                  ; Q dans AX = 1000H et R dans DX = 0005H
```

d) Le programme effectue une division de (DX:AX)/BX, le quotient est sauvegardé dans AX et le reste dans DX.

Corrigé de l'exercice 04 :

Le tableau ci-dessous contient les instructions équivalentes et contraires rangées les unes devant les autres :

Type	Instructions équivalentes		Instructions contraires	
NON SIGNEES	JA	JNBE	JP/JPE	JNP/JPO
	JAE	JNB	JS	JNS
	JB	JNAE	JO	JNO
	JBE	JNA	JC	JNC
SIGNEES	JG	JNLE	JNE	JNZ
	JGE	JNL	JE	JZ
	JL	JNGE		
	JLE	JNG		

Even : en français **pair**
Odd : en français **impair**

Corrigé de l'exercice 05 :

a)
```
        MOV   AL, 53H
etiq1:  CMP   AL, 41H
        JB    etiq1 ;
etiq2:  ADD   AL, F0H
        JNC   etiq2 ;
```
R1 : le programme ne fera pas un saut vers etiq1, puisque AL **> 41H**

R2 : le programme ne fera pas un saut vers etiq2, puisque **AL dépasse FFH donc CF = 1**

b)
```
        MOV   AL, 53H
label:  CMP   AL, 41H
        JA    label ;
```
R3 : le programme fera un saut vers label, puisque **AL > 41H**

c)
```
        MOV  AX, 10
etiq:   MOV  BX, 5
        SUB  AX, BX
        JP   etiq ;
```
R4 : le programme fera un saut vers etiq, puisque **AX = 0000 0000 0000 0101B contient un nombre de 1 pair)**

d)
```
        MOV  AL, 10
ABV:    MOV  BL, 5
        NEG  BL ; après négation du BL,  BL = FBH
        CMP  AL, BL
        JA   ABV ;
```
R5 : le programme ne fera pas un saut puisque (AL =10) < (BL = FBH). L'instruction JA traite le contenu de BL comme un nombre non-signé BL = FBH = 251D

e)
```
        MOV  AL, 10
GRT:    MOV  BL, 5
        NEG  BL
        CMP  AL, BL
        JGE  GRT ;
```
R6 : le programme fera un saut puisque (AL =10) > (BL = FBH). L'instruction JGE traite le contenu de BL comme un nombre signé BL= FBH = -5D

Corrigé de l'exercice 06 :

1) L'instruction 'CALL' empile la valeur de IP de l'instruction qui précède CALL, d'adresse logique **0171H**, dans la pile, c'est à dire elle effectue un PUSH (**SP ←SP−2**), donc SP = FFFCH, le sommet de la pile prendra la valeur **0171H**, puis le registre IP pointe sur le début du sous-programme 'ALI', donc IP = 230H.

2) L'instruction '**RET**' dépile la pile (**SP ←SP+2**), donc SP = FFFEH. Le registre IP prendra la valeur **0171H**. c'est-à-dire l'adresse IP de l'instruction qui précède l'instruction CALL.

Corrigé de l'exercice 07 :

Après exécution du programme ci-dessous, nous obtenons le tableau ci-contre

```
      MOV  SI,  0010H
      MOV  CX,  3
      MOV  AL,  1
ENC:  ADD  AL,  [SI] ; AL ← AL + le contenu de la case mémoire pointée par SI
      INC  SI
      MOV  [SI],  AL
      DEC  CX
      JNZ  ENC
      END
```

3	← 0010H
4	
8	
16	

SI=10H, CX=3, AL=1,

(1): AL=4, SI=11H, CX=2,

(2): AL=4+4=8, SI=12H, CX=1,

(3): AL=8+8=16, SI=13H, CX=0,

Fin de programme

Corrigé de l'exercice 08 :

Après l'exécution du programme ci-dessous nous obtenons le tableau ci-contre

```
      MOV   SI, 100H
      MOV   DI, 101H
ETQ1: MOV   AL, [SI]
      XOR   AL, 0C5H
      ADD   AL, 2
      MOV   [DI], AL
      INC   SI
      INC   DI
      CMP   SI, 103H
      JNZ   ETQ1
END
```

11 H	100H
D6 H	101H
15 H	102H
D2 H	103H

SI=100H, DI=101H,

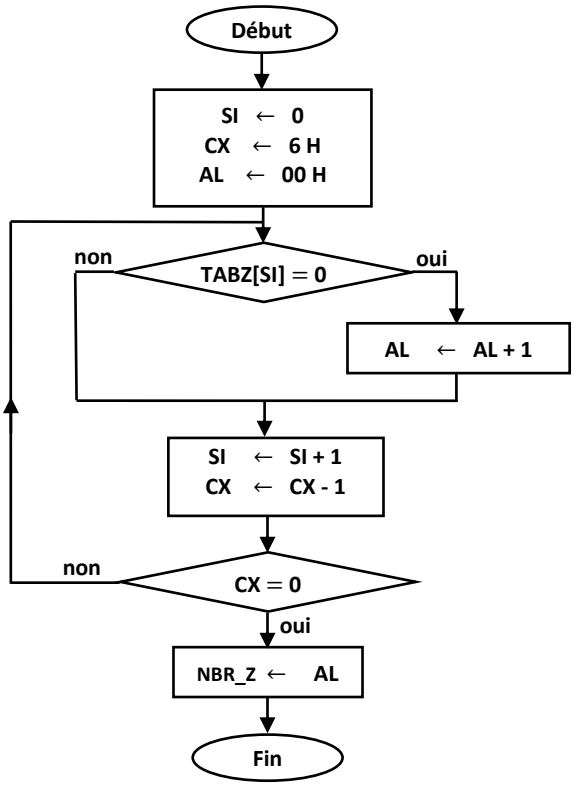
(1): AL=11H, AL=D4H, AL=D6H, SI=101H, DI=102H

(2): AL=D6H, AL=13H, AL=15H, SI=102H, DI=103H

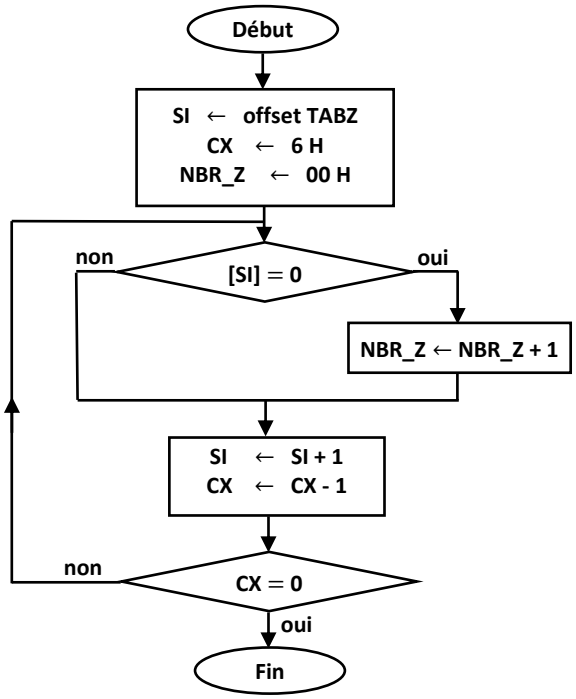
(3): AL=15H, AL=D0H, AL=D2H, SI=103H, DI=104H

Fin de programme

Corrigé de l'exercice 09 :



Organigramme 1



Organigramme 2

```
Name "nul_tab"
ORG  100H
.DATA  ; Permet d'indiquer le début du segment de données.
TABZ    db   12, 0, 84, 100, 220, 0 ; Déclaration du tableau TABZ
NBR_Z  db   0 ; Déclaration d'une variable sur un octet initialisée à 0
.CODE ; Permet d'indiquer le début du segment de code
MOV      AX, @DATA ; Chargement de l'adresse du data segment
MOV      DS, AX       ; dans le registre data Segment (DS)
;-------------------------------------------------------------------------------

MOV      SI, 0 ; Initialisation du pointeur du tableau
MOV      CX, 6 ; Chargement de la taille du tableau dans CX
MOV      AL, 0

encore:
CMP      TABZ [SI], 0
JNZ      mis_pnt ; Sauter vers mis_pnt si TABZ [SI] = 0
INC      AL ; Incrémentation du AL
mis_pnt:
INC      SI ; Mise à jour du pointeur
DEC      CX ; Mise à jour du compteur
JNZ      encore
MOV      NBR_Z, AL ; Sauvegarder le nombre de '0' dans NBR_Z

MOV      AH, 4CH ; Fin de programme
INT      21H      ; Retour au DOS
ENDP
```

Programme 1

```
Name "nul_tab"
ORG  100H
.DATA
TABZ    db   12, 0, 84, 100, 220, 0
NBR_Z  db   0
.CODE
MOV      AX, @DATA
MOV      DS, AX
;-------------------------------------------------------------------------------

LEA      SI, TABZ ; Chargement de l'adresse du début de TABZ
MOV      CX, 6 ; Chargement de la taille du tableau dans CX

encore:
CMP      [SI], 0
JNZ      mis_pnt
INC      NBR_Z ; Incrémentation de NBR_Z
mis_pnt:
INC      SI ; Mise à jour du pointeur
DEC      CX ; Mise à jour du compteur
JNZ      encore

MOV      AH, 4CH ; Fin de programme
INT      21H      ; Retour au DOS
ENDP
```

Programme 2

- **Le premier programme** utilise AL pour compter le nombre de '0'. Le contenu de AL sera sauvegardé dans la variable **NBR_Z**. Le registre SI est utilisé pour pointer sur les cases mémoires du tableau 'TABZ', qui contient N octets, le registre SI est initialisé à '0'.
- Dans **le deuxième programme** la variable **NBR_Z** est utilisée **directement** comme compteur de '0'. Au début du programme, le registre SI est chargé par l'adresse effective du début du tableau 'TABZ'. SI doit prendre

des valeurs comprises entre : **Adr_TABZ+0 à Adr_TABZ+N-1** (où N est la taille du tableau).

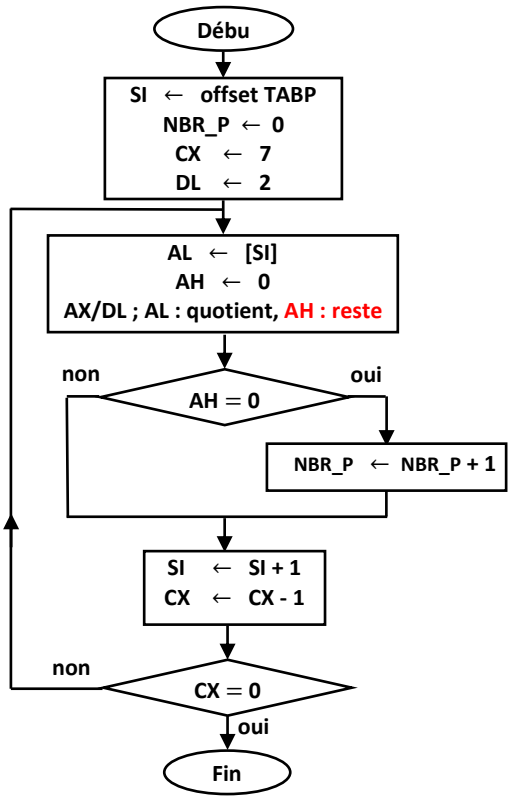
- Si **Adr_TABZ = 102H** et **N = 6** alors **SI** varie de **102H … 107H**

L'instruction **LEA SI, TABZ** permet de charger l'adresse effective du début du tableau 'TABZ' dans le registre source index **SI**. L'instruction **LEA SI, TABZ** et **MOV SI, offset TABZ** sont équivalentes.

LEA SI, TABZ ≡ MOV SI, offset TABZ

Corrigé de l'exercice 10 :

Vérification de la parité par division de l'opérande par '2'



Organigramme 1

```
Name "nbr_pair1"
ORG  100H
.DATA
TABP   db   11, 0, 80, 99, 220, 1, 5
NBR_P db   0
.CODE
            MOV      AX, @DATA
            MOV      DS, AX
;-------------------------------------
            MOV      SI, offset TABP
            MOV      CX, 7
            MOV      DL, 2

enc:        MOV      AL, [SI]
            MOV      AH, 0
            DIV      DL
            CMP      AH, 0
            JNZ      mis_ptr
            INC      NBR_P

mis_ptr:    INC      SI
            DEC      CX
            JNZ      enc

MOV   AH, 4CH
INT    21H
ENDP
```

Programme 1

```
Name "nbr_pair1"
ORG  100H
.DATA
TABP   db   11, 0, 80, 99, 220, 1, 5
NBR_P db   0
.CODE
            MOV      AX, @DATA
            MOV      DS, AX
;-------------------------------------
            LEA      SI, TABP
            MOV      CX, 7
            MOV      DL, 2

enc:        MOV      AL, [SI]
            MOV      AH, 0
            DIV      DL
            CMP      AH, 0
            JNZ      mis_ptr
            INC      NBR_P

mis_ptr:    INC      SI
            LOOP     enc

MOV   AH, 4CH
INT    21H
ENDP
```

Programme 2

Test de parité

Si le reste de la division d'un nombre par 2 est égale à zéro, alors on dit que le nombre est pair, si le reste de la division est égale à 1 on dit que le nombre est impair.

Donc pour compter les nombres pairs dans le tableau TABP on doit tester la parité de tous ses éléments. Pour effectuer le test (c.-à-d. division par 2) on doit charger la valeur de chaque case mémoire du tableau dans le registre AX pour la diviser en 2, puisque l'instruction de division **'DIV'** utilise le registre accumulateur AX.

Le tableau TABP est organisé en octet (8 bits), et le registre AX est de taille 16 bits, donc pour charger une case mémoire dans le registre AX (16 bits), on doit charger le contenu de la case mémoire dans AL et mettre à zéro l'octet poids fort du registre AX (AH ← 0). Une fois la division effectuée, le quotient est sauvegardé dans AL et le reste dans AH. Si AH=0 le nombre est pair Sion il est impair.

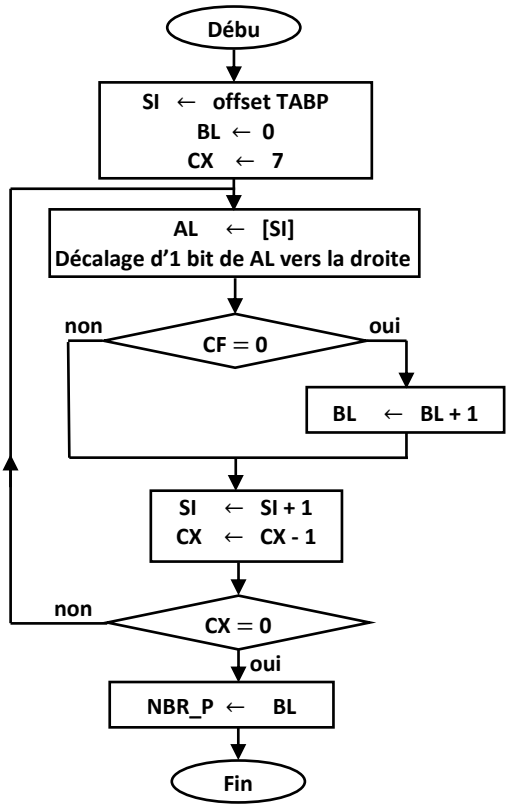
Rappel :

LOOP etq

L'instruction **loop** décrémente la valeur du registre **CX**, et fait un saut vers l'étiquette (**etq**) si CX est diffèrent de zéro. Cette instruction est équivalente aux instructions suivantes :

```
DEC  CX
JNZ  etq
```

Vérification de la parité par l'utilisation des instructions logiques de décalage et de rotation



Organigramme

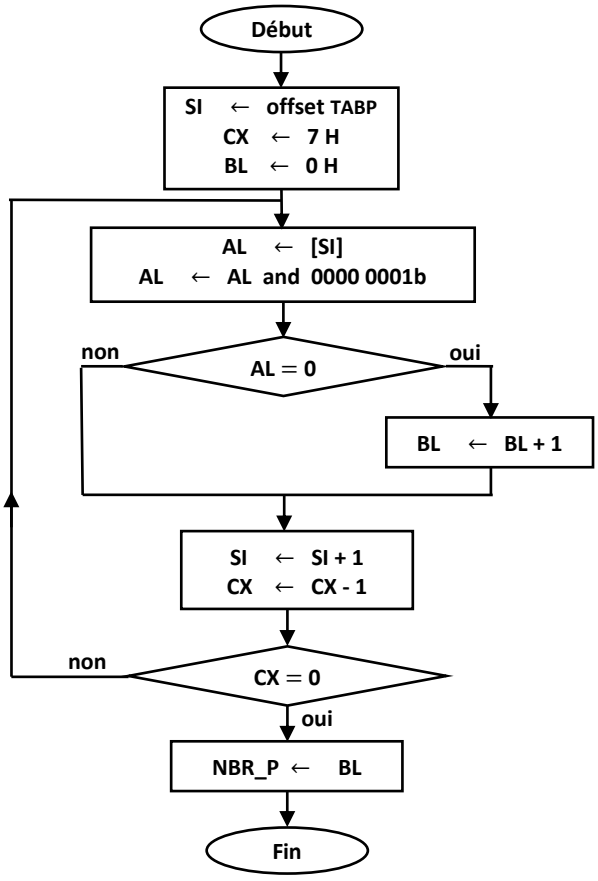
```
Name "nbr_pair2"
ORG  100H
.DATA
TABP   db   11, 0, 80, 99, 220, 1, 5
NBR_P db   0
.CODE
            MOV      AX, @DATA
            MOV      DS, AX
;-------------------------------------
            LEA      SI, TABP
            MOV      CX, 7
            MOV      BL, 0

encore:     MOV      AL, [SI]
            SHR      AL, 1
            JC       mis_ptr
            INC      BL

mis_ptr:    INC      SI
            LOOP     encore
            MOV      NBR_P, BL
;-------------------------------------
            MOV   AH, 4CH
            INT   21H
            ENDP
```

Programme 1

```
Name "nbr_pair2"
ORG  100H
.DATA
TABP   db   11, 0, 80, 99, 220, 1, 5
NBR_P db   0
.CODE
            MOV      AX, @DATA
            MOV      DS, AX
;-------------------------------------
            LEA      SI, TABP
            MOV      CX, 7
            MOV      BL, 0

encore:     MOV      AL, [SI]
            ROR      AL, 1
            JC       mis_ptr
            INC      BL

mis_ptr:    INC      SI
            LOOP     encore
            MOV      NBR_P, BL
;-------------------------------------
            MOV   AH, 4CH
            INT   21H
            ENDP
```

Programme 2

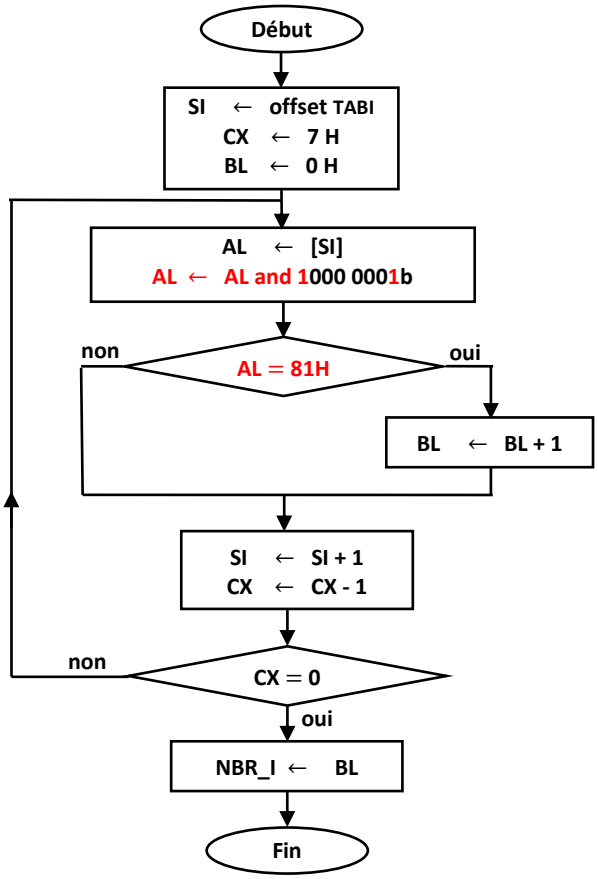
Vérification de la parité en utilisant la technique de masquage des bits




```
Name "nbr_pair3"
ORG  100H
.DATA
TABP   db   11, 0, 80, 99, 220, 1, 5
NBR_P db   0
.CODE
            MOV      AX, @DATA
            MOV      DS, AX
;-------------------------------------
            LEA      SI, TABP
            MOV      CX, 7
            MOV      BL, 0

enc:        MOV      AL, [SI]
            AND      AL, 00000001b
            CMP      AL, 0
            JNZ      mis_ptr
            INC      BL
mis_ptr:    INC      SI
            DEC      CX
            JNZ      enc
            MOV      NBR_P, BL
;-------------------------------------
            MOV   AH, 4CH
            INT   21H
            ENDP
```

Programme

Corrigé de l'exercice 11 :




```
Name "nbr_impair_sup128"
ORG  100H
.DATA
TABI      db   11, 129, 80, 99, 221, 1, 128
NBR_I     db   0
.CODE
          MOV      AX, @DATA
          MOV      DS, AX
;-------------------------------------------------------
          LEA      SI, TABI
          MOV      CX, 7
          MOV      BL, 0

enc:      MOV      AL, [SI]
          AND      AL, 10000001b
          CMP      AL, 81H
          JNZ      mis_ptr
          INC      BL
mis_ptr:  INC      SI
          DEC      CX
          JNZ      enc
          MOV      NBR_I, BL
;-------------------------------------------------------
          MOV   AH, 4CH
          INT   21H
          ENDP
```

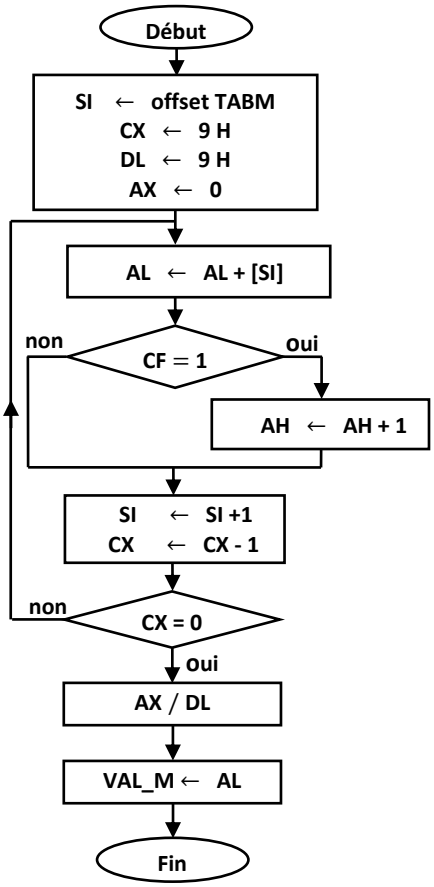
Dans cet exercice, on s'intéresse au bit de poids fort et au bit de poids faible. Le masque est de la forme suivante :

1	0	0	0	0	0	0	1

Si le **bit de poids faible** est à '1' on dit que le nombre est pair, dans le cas contraire il est impair. Si le **bit de poids fort** est à 1, le nombre est ≥ 128, dans le cas contraire, il est < 128.

Corrigé de l'exercice 12 :

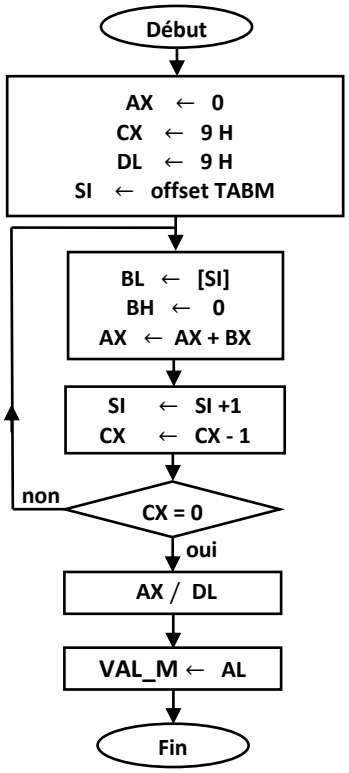
Organigramme 1 :



Programme 1:

```
name "moy_tableau"
ORG 100h
.DATA
TABM    db 2, 5, 100, 1, 4, 60, 120, 12, 52
VAL_M   db 0
.CODE  ; déclaration du code segment
        MOV     AX,@DATA
        MOV     DS,AX
;-------------------------------------------------------------
        MOV     AX, 0
        MOV     CX, 9
        MOV     DL, 9
        LEA     SI, TABM

etiq2:  ADD     AL, [SI]
        JNC     etiq1
        INC      AH
etiq1:  INC     SI
        LOOP    etiq2
        DIV     DL
        MOV     VAL_M, AL
;-------------------------------------------------------------
        MOV     AH, 4CH
        INT     21H
        ENDP
```

Organigramme 2:



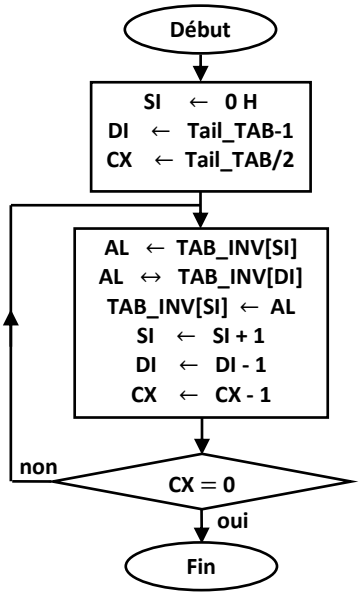
Programme 2:

```
name "moy_tableau"
ORG 100h
.DATA
TABM  db 2, 5, 100, 1, 4, 60, 120, 12, 52
VAL_M db 0
.CODE  ; déclaration du code segment
MOV     AX,@DATA
MOV     DS,AX
;-------------------------------------------------------------
        MOV    AX, 0
        MOV    CX, 9
        MOV    DL, 9
        LEA    SI, TABM ; charger l'offset du tab dans SI
etiq2:
        MOV    BL, [SI]
        MOV    BH, 0
        ADD    AX , BX
        INC    SI
        LOOP   etiq2
        MOV    DL, 9
        DIV    DL
        MOV    VAL_M, AL
;-------------------------------------------------------------
        MOV    AH, 4CH     ; fin de programme
        INT    21H         ; retour au DOS
        END
```

Corrigé de l'exercice 13 :

Solution 1 : cette solution utilise deux pointeurs SI et DI.

Organigramme 1 :



Programme 1 :

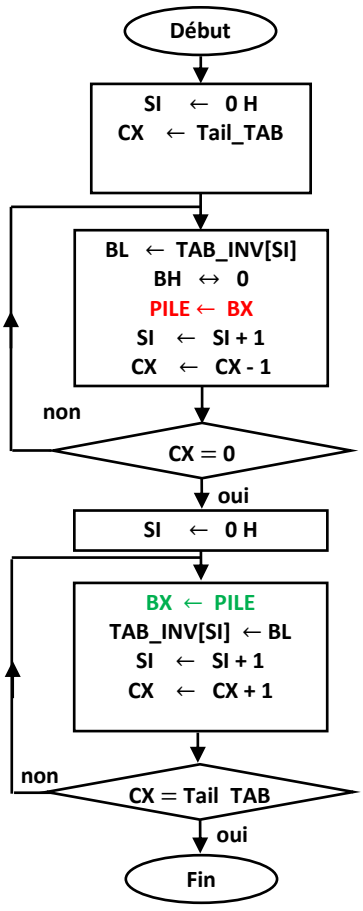
```
NAME "inv_tableau"
ORG 100h
.DATA
TAB_INV    db  1, 2, 7, 12H, 74H, 22H, 34H
Tail_TAB   equ  7
;-------------------------------------------------------------
.CODE  ; déclaration du code segment
MOV      AX,@DATA
MOV      DS,AX
         MOV     CX, (Tail_TAB/2) ;  CX ← 3
         MOV     SI, 0 ; SI pointe sur le premier élément du tab
         MOV     DI, Tail_TAB-1 ; DI pointe sur le dernier élément du tab
enc:     MOV     AL, TAB_INV [SI]
         XCHG    AL, TAB_INV [DI]
         MOV     TAB_INV [SI], AL
         INC     SI
         DEC     DI
         LOOP    enc
;-------------------------------------------------------------
         MOV     AH, 4CH
         INT     21H
         ENDP
```

La directive **'equ'** est utilisé pour déclarer une constante (**equ** pour designer 'Equal' en anglais)

Dans ce programme nous avons introduit l'instruction **'XCHG'**, qui permet la permutation de deux opérandes sans avoir recours à une variable intermédiaire.

Solution 2 : cette solution est basée sur l'utilisation de la pile.

Organigramme 2 :

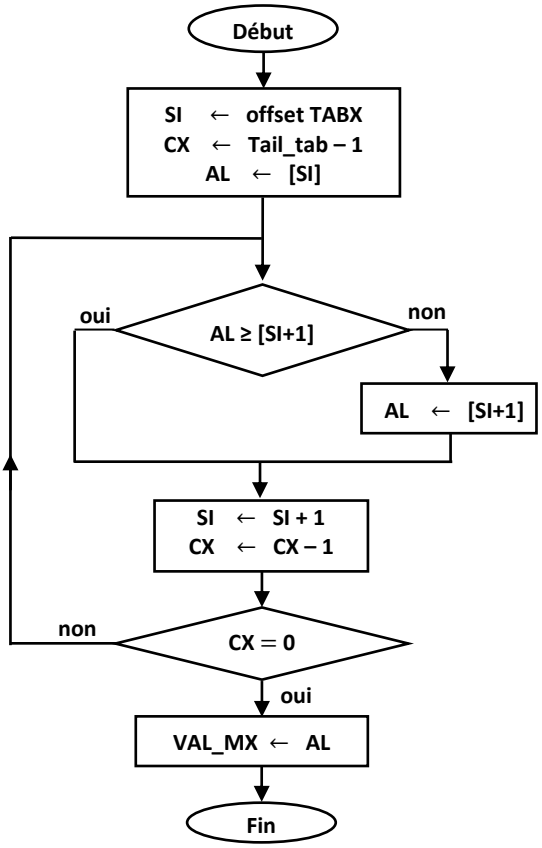


Programme 2 :

```
NAME "inv_tableau"
ORG 100h
.DATA
TAB_INV    db  1, 2, 7, 12H, 74H, 22H, 34H
Tail_TAB   equ  7
;-------------------------------------------------------------
.CODE  ; déclaration du code segment
         MOV     AX,@DATA
         MOV     DS,AX

         MOV     SI, 0       ; SI pointe sur le premier élément du tab
         MOV     CX, Tail_TAB        ;  CX ⍰ 7
enc:     MOV     BL, TAB_INV [SI]
         MOV     BH, 0
         PUSH    BX
         INC     SI
         DEC     CX
         JNZ     enc

         MOV     SI, 0
enc1:    POP     BX
         MOV     TAB_INV[SI], BL
         INC     SI
         INC     CX
         CMP     CX, Tail_TAB
         JNE     enc1
;-------------------------------------------------------------
         MOV     AH, 4CH
         INT    21H
         ENDP
```

Corrigé de l'exercice 14 :

Organigramme :

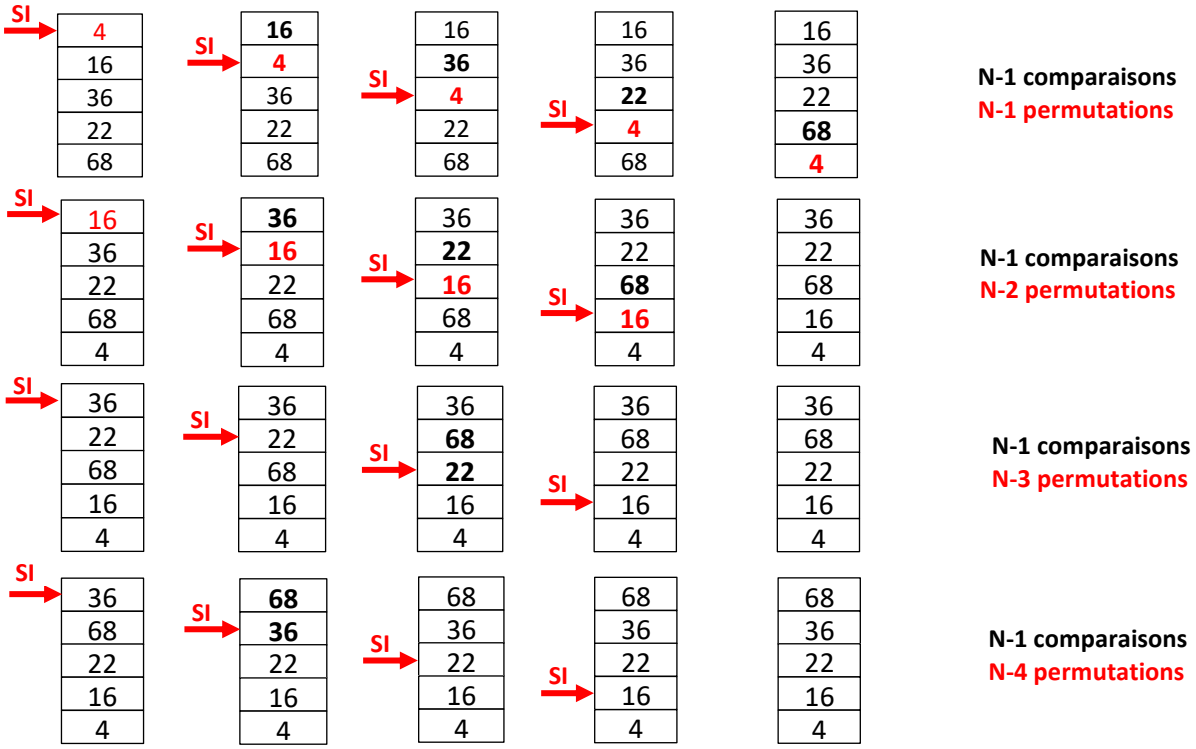


Programme :

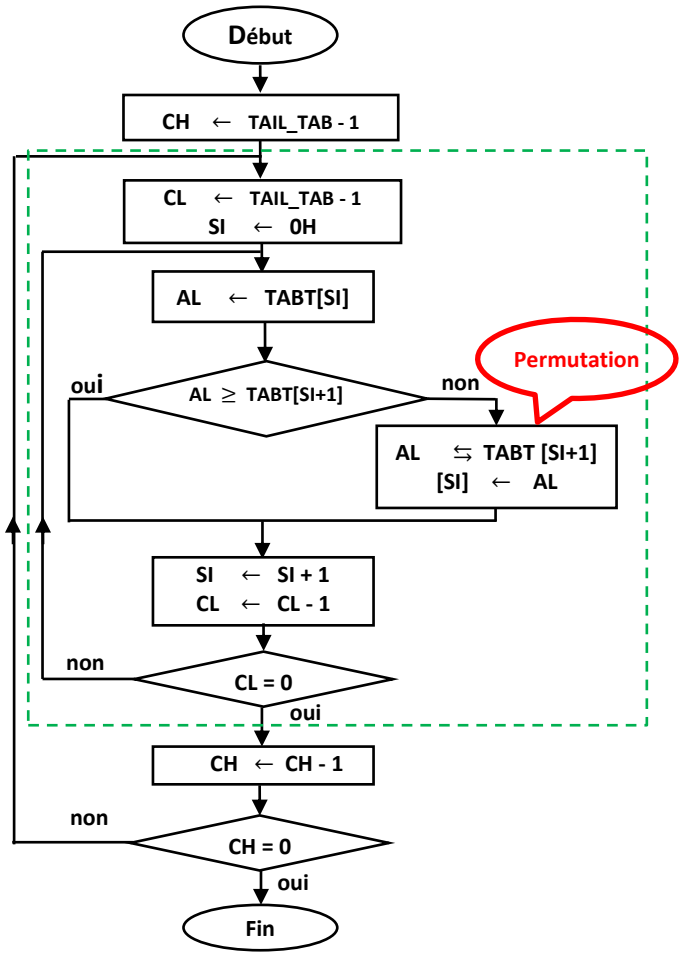
```
Name "max_tab"
ORG  100H
.DATA
TABX     db   12, 2, 84, 100, 66, 72
VAL_MX   db  0
Tail_tab  equ  6
.CODE
          MOV    AX, @DATA
          MOV    DS, AX
;-------------------------------------------------------------------------
          LEA      SI, TABX ; chargement de l'offset du tab
          MOV      CX, Tail_tab - 1
          MOV      AL, [SI] ; AL contient le premier élément
encore:
          CMP      AL, [SI+1]
          JAE      mis_ptr ; saut si [AL] ≥ [SI+1]
          MOV      AL, [SI+1] ; sauvegarder le nouveau MAX
mis_ptr:
          INC      SI
          DEC      CX
          JNZ      encore
          MOV      VAL_MX, AL ; sauvegarde du MAX
;-------------------------------------------------------------------------
          MOV      AH, 4CH
          INT      21H
          ENDP
```

Corrigé de l'exercice 15 :

Pour trier un tableau en ordre décroissant, on effectue une permutation du contenu de la case pointée par [SI] avec celui de la case [SI+1], et ce, lorsque [SI+1] > [SI+1], puis on passe à la case suivante jusqu'à atteindre la dernière case du tableau. On répète cette opération (N-1) fois. (N : la taille du tableau, exemple N=5)



Organigramme :



Programme :

```
NAME "tri_tableau"
ORG 100h
.DATA
TABT      db  4, 16, 36, 22, 68
TAIL_TAB  equ 5
.CODE  ; déclaration du code segment
          MOV     AX, @DATA
          MOV     DS, AX
;-------------------------------------------------------
          MOV     CH, TAIL_TAB - 1
etiq3:
          MOV     CL, TAIL_TAB - 1
          MOV     SI, 0

etiq2:
          MOV     AL, TABT[SI]
          CMP     AL, TABT[SI+1],
          JAE     etiq1
          XCHG    AL, TABT[SI+1]
          MOV     TABT[SI], AL
etiq1:
          INC     SI
          DEC     CL
          JNZ     etiq2
          DEC     CH
          JNZ     etiq3
;-------------------------------------------------------
          MOV     AH, 4CH
          INT     21H
          ENDP
```

La solution de l'exercice n'est pas optimale, nous constatons que le nombre total de comparaisons effectuées est de $(N-1)(N-1) = (N-1)^2$ Comparaisons. Exemple : pour un tableau de taille N=101 éléments, le nombre de comparaisons à effectuer est 10 000 comparaisons.

Optimisation de la première solution :

Si on optimise la solution, cela permettra de réduire considérablement la charge de calcul.
En fait, le programme effectue N-1 comparaison mais les permutations entre les cases diminuent de 1 chaque nouvelle exécution. Pour notre exemple, la taille du tableau est égale à N = 5 alors :

- **1ére exécution** du programme effectue **4 permutations**
- **2éme exécution** du programme effectue **3 permutations**
- **3éme exécution** du programme effectue **2 permutations**
- **4éme exécution** du programme effectue **1 permutation**

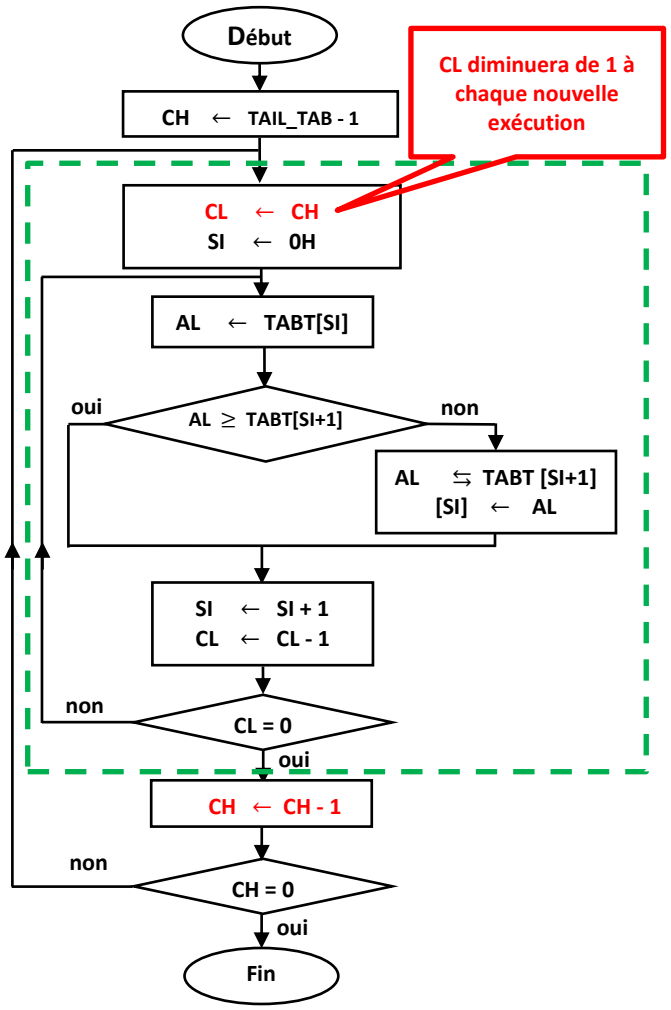
Le nombre d'opérations effectuées jusqu'au trie du tableau est 1 + 2 + 3 + 4. On constate que le nombre permutations est une suite arithmétique de raison $\boldsymbol{r = 1}$ et de premier terme $\boldsymbol{u_0 = 1}$. La somme $\boldsymbol{S}$ d'une suite arithmétique est donnée par l'expression $\boldsymbol{S = u_0 + u_1 + \cdots + u_n = (n+1)\frac{u_0+u_n}{2} = (n+1)\frac{2u_0+nr}{2}}$
Donc si un tableau contient N élément, la somme de nombre d'opération est :

$$\boldsymbol{S = u_0 + u_1 + \cdots + u_{N-2} = ((N-2)+1)\frac{u_0+u_{N-2}}{2} = (N-1)\frac{2u_0+(N-2)\cdot r}{2}}$$

$$u_0 = 1 \; et \; r = 1 \; \text{donc} \; \boldsymbol{S = \frac{(N-1)N}{2}}$$

Exemple : si un tableau contient 101 éléments, le nombre total de permutations est $\boldsymbol{S = \frac{100 \cdot 101}{2} = 5050}$ permutations, contre $\boldsymbol{(N-1)^2 = 10000}$ comparaisons obtenues avec le premier algorithme.

La solution ci-dessous permet d'optimiser l'algorithme proposé précédemment



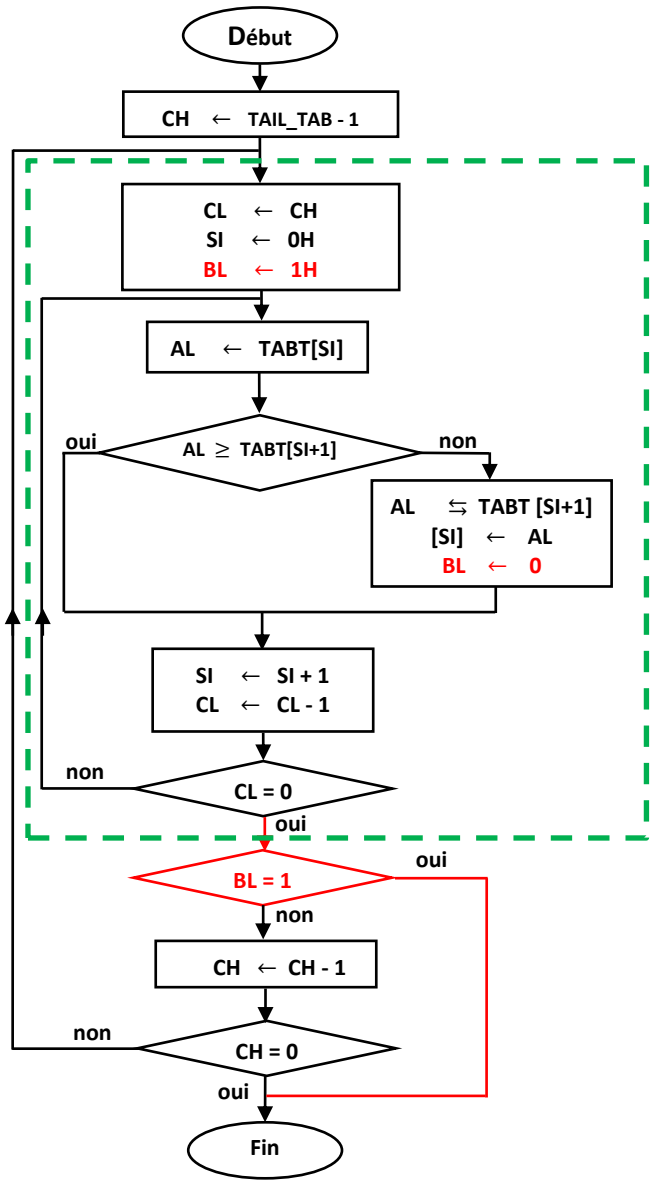

```
NAME "tri_tableau"
ORG 100h
.DATA
TABT      db  4, 16, 36, 22, 68
TAIL_TAB  equ 5
.CODE  ; déclaration du code segment
          MOV     AX, @DATA
          MOV     DS, AX
;-------------------------------------------------------------
          MOV     CH, TAIL_TAB - 1
etiq3:
          MOV     CL, CH
          MOV     SI, 0

etiq2:
          MOV     AL, TABT[SI]
          CMP     AL, TABT[SI+1],
          JAE     etiq1
          XCHG    AL, TABT[SI+1]
          MOV     TABT[SI], AL
etiq1:
          INC     SI
          DEC     CL
          JNZ     etiq2
          DEC     CH
          JNZ     etiq3
;-------------------------------------------------------------
          MOV     AH, 4CH
          INT     21H
          ENDP
```

Amélioration du temps d'exécution de la deuxième solution :

Généralement le programme n'effectue toutes les itérations afin qu'il soit trié, parfois le tableau est presque ordonné (c'est-à-dire proche à l'ordre désiré).

Si aucune permutation n'a été effectuée lors de l'exécution de la boucle interne (la boucle à l'intérieur du cadre vert), alors cela veut dire que le tableau est trié. Alors, il nécessaire **d'arrêter l'exécution du programme**. Cela permettra un gain considérable de temps d'exécution et diminuera la charge de calcul.

Pour ce faire, nous avons introduit une variable qui permet de détecter si une permutation a été effectuée. Dans notre exemple, on initialise le registre BH par la valeur '1', dans le cas où une permutation est effectuée le programme met à '0' le registre BL. Dans le cas ou BH garde sa valeur initial '1', cela veut dire que le tableau n'a subi aucune permutation, donc il bien trié alors il faut arrêter le programme.




```
NAME "tri_tableau"
ORG 100h
.DATA
TABT      db  4, 16, 36, 22, 68
TAIL_TAB  equ 5
.CODE  ; déclaration du code segment
          MOV     AX, @DATA
          MOV     DS, AX
;-------------------------------------------------------
          MOV     CH, TAIL_TAB - 1
etiq3:
          MOV     CL, CH
          MOV     SI, 0
          MOV     BL, 1
etiq2:
          MOV     AL, TABT[SI]
          CMP     AL, TABT[SI+1],
          JAE     etiq1
          XCHG    AL, TABT[SI+1]
          MOV     TABT[SI], AL
          MOV     BL, 0
etiq1:
          INC     SI
          DEC     CL
          JNZ     etiq2
          CMP     BL, 1
          JE      fin
          DEC     CH
          JNZ     etiq3
;-------------------------------------------------------
Fin:      MOV     AH, 4CH
          INT     21H
          ENDP
```

Exemple : on essaye d'exécuter les trois solutions sur les tableaux de 10 octets ci-dessous

TABT1 db 25, 66, 45, 33, 22, 14, 1, 77, 100, 200 (complètement désordonné)

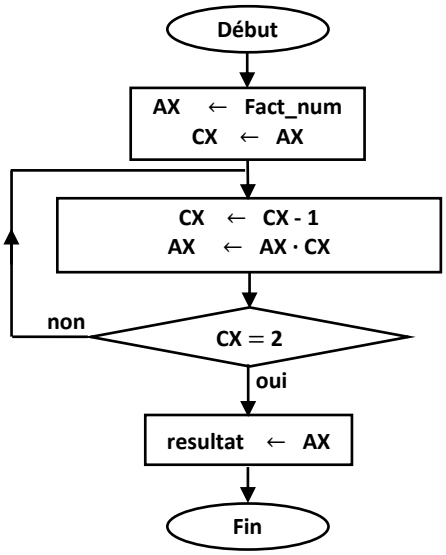
TABT2 db 25, 66, 100, 200, 77, 45, 33, 22, 14, 1 (moyennement ordonné)

TABT3 db 100, 200, 77, 45, 66, 33, 25, 22, 14, 1 (presque ordonné)

	Formule de calcul de nombre de permutation	Nombre de permutations		
		TABT1	TABT2	TABT3
Solution 1	$S_1 = (N-1)^2$	81	81	81
Solution 2	$S_2 = (N-1)N/2$	45	45	45
Solution 3	S_3 : dépend essentiellement de l'organisation du tableau	45	30	17

Solution de l'exercice 16 :

Le programme ci-dessous calcule le factoriel d'un nombre compris entre 2 et 8




```
Name  "factoriel"
ORG 100h
.DATA
Resultat   dw  0
Fact_num  equ   6 ; le nombre choisi (exemple égale à 6)

.code
            mov       ax, @data
            mov       ds, ax
; -------------------------------------------------------------
            mov       ax, Fact_num
            mov       cx, ax
enc:        dec       cx
            mul       cx ;    AX ← AX · CX
            cmp       cx, 2
            jnz       enc
            mov       resultat, ax
; -------------------------------------------------------------
            mov       ah, 4ch ; fin de programme
            int 21h   ; retour au DOS
            endp
```

Corrigé de l'exercice 17 :

Pour déterminer le nombre des voyelles dans une chaine de caractères, le programme compare chaque élément de la chaine caractères avec toutes les voyelles. Dix comparaisons sont nécessaires, cinq comparaisons avec les voyelles de forme minuscule ('a', 'e', 'i', 'o', 'u') et cinq d'autres avec les voyelles de forme majuscule ('A', 'E', 'I', 'O', 'U'). Pour minimiser le nombre de comparaisons, nous convertissons les caractères minuscules ('a' < caractère < 'z') en majuscule, puis nous effectuons seulement la comparaison avec les voyelles de forme majuscule.

```
NAME "compteur_voyelle"
.stack 100h
.data
chaine_caract      db       'programme assembleur'
long_chaine        dw       20
nbr_voy db ?
.code
            MOV     AX, @data
            MOV     DS, AX
; ---------------------------------------------------------------------
            MOV     SI, offset chaine_caract ;
            MOV     CX, long_chaine
            MOV      BL, 00 ; compteur de nombre de voyelle
encore:
            MOV     AL, [SI]
            CMP     AL, 'a'
            JB      test_voyelle
            CMP     AL, 'z'
            JA      test_voyelle
            SUB     AL, 20H ; convertir en caractères majuscule
test_voyelle:
            CMP     AL, 'A'
            JNZ     test_E
            INC     BL
            JMP     mis_pt
test_E:
            CMP     AL, 'E'
            JNZ     test_I
            INC     BL
            JMP     mis_pt
test_I:
            CMP     AL, 'I'
            JNZ     test_O
            INC     BL
            JMP     mis_pt
test_O:
            CMP     AL, 'O'
            JNZ     test_U
            INC     BL
            JMP     mis_pt
test_U:
            CMP     AL, 'U'
            JNZ     mis_pt
            INC     BL
mis_pt:
            INC     SI
            LOOP    encore
            MOV     nbr_voy, BL
; ---------------------------------------------------------------------
            MOV     AX, 4C00H
            INT     21H
            ENDP
```

Solution de l'exercice 18 :

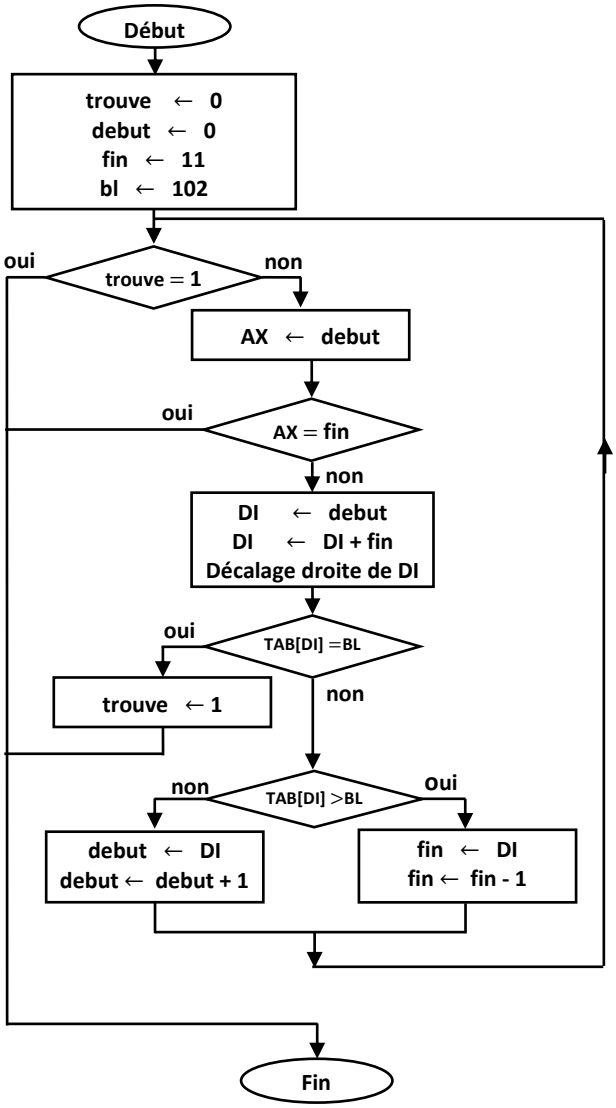
Le programme ci-dessous permet de chercher un mot dans une phrase.

```
NAME "cherche_mot"
ORG 100h
.data
phrase     db   ' thagaste est une belle ville' ;  la phrase
mot        db   'belle' ; le mot cherché
taille_mot  equ   5
taille_phrase equ 28
trouve     db   0
.code
           mov    ax,  @data
           mov    ds,  ax
; ------------------------------------------------------------------------
           mov       si, 0
           mov       di, 0
           mov       cl, taille_mot
           mov       ch, taille_phrase
           mov       bx, 0
encore:
           mov       al, phrase[si]
           cmp       al, mot[di]
           jz        inc_bl
           mov       di, 0
           cmp       bx, 0
           jz        inc_si
           sub       si, bx
           mov       bx, 0
inc_si:
           inc       si
           dec       ch
           jnz       encore
           jmp       fin_prog
inc_bl:
           inc       si
           inc       di
           inc       bx
           cmp       bx, taille_mot
           jnz       encore
           mov       trouve, 1
fin_prog :
;------------------------------------------------------------------------
           MOV       ah, 4ch ; fin de programme
           INT       21h   ; retour au DOS
           ENDP
```

Solution de l'exercice 19 :

```
Name  "somme_MATRIX"
org 100h
.data
MTX  db  23, 34, 24, 11, 42, 15, 12, 15, 37, 6, 5, 12
Somme_ln db 4 dup(?) ;
Somme_cl db 3 dup(?) ;
Somme   db 0
cpt_ln    dw ?
cpt_cl    dw ?
nbr_ln    equ 4
nbr_cl    equ 3
.code
          MOV     AX, @data
          MOV     DS, AX
;-----------------------------------------------------------------
; Calcul de la somme de chaque ligne
          MOV     cpt_ln, nbr_ln
          MOV     DI,0
          LEA     SI, MTX

new_ln:   MOV     CX, nbr_cl
          MOV     Somme, 0
m_Ln:     MOV     AH, [SI]
          ADD     Somme , AH
          INC     SI
          LOOP    m_Ln

          MOV     AH, Somme
          MOV     Somme_ln[DI], AH
          INC     DI
          DEC     cpt_ln
          JNZ     new_ln
; Calcul de la somme de chaque colonne
          MOV     cpt_ln, nbr_ln
          MOV     cpt_cl, nbr_cl
          MOV     DI,0
          MOV     BX, 00

new_cl:   MOV     CX, nbr_ln
          MOV     Somme,0
          LEA     SI, MTX
          ADD     SI, BX

m_cl:     MOV     ah, [SI]
          ADD     Somme , ah
          ADD     SI, nbr_cl
          LOOP    m_cl

          MOV     ah, Somme
          MOV     Somme_cl[DI], ah
          INC     BX
          INC     DI
          DEC     cpt_cl
          JNZ     new_cl
;-----------------------------------------------------------------
          MOV     ah, 4ch ; fin de programme
          INT     21h   ; retour au DOS
          ENDP
```

Solution de l'exercice 20 :




```
name "Recherche dichotomique"
org 100h
.stack 200h
.data
TAB db  1, 2, 3, 4, 55, 66, 77, 88, 99, 100, 102, 104
val_cher   db   102 ; la valeur recherchée
debut      dw  0
fin        dw  11
trouve     db  0
.code
        MOV    AX, @data
        MOV    DS, AX
;-------------------------------------------------------------
        MOV    BL,  val_cher
ENC:
        CMP    trouve,  1
        JZ     FINISH
        MOV    AX,  debut
        CMP    AX,  fin
        JA     FINISH

        MOV    DI,  debut
        ADD    DI,  FIN
        SHR    DI,  1       ; DI/2

        CMP    TAB[DI],  BL
        JNZ    DIFF
        MOV    trouve,  1
        JMP    FINISH
DIFF:
        CMP    TAB[DI], BL
        JA     ABV
        MOV    debut, DI
        INC    debut
        JMP    ENC
ABV:
        MOV    fin, DI
        DEC    fin
        JMP    ENC
FINISH:
        ;-------------
        MOV AH,4CH
        INT 21H
        ENDP
```