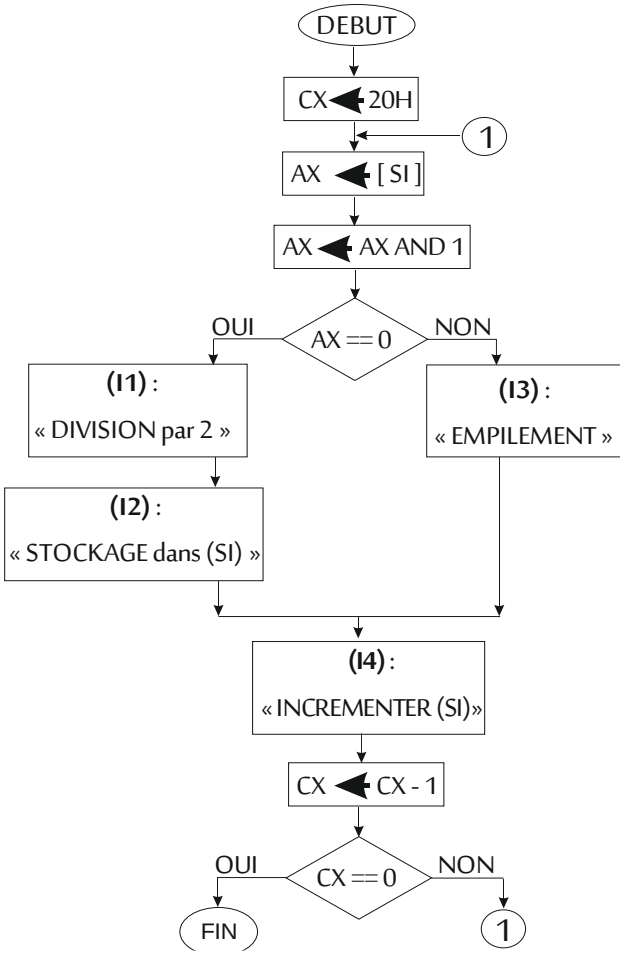
- TOUS DOCUMENTS INTERDITS
- TOUT échange (documents ou autres biens) INTERDIT (effaceur, etc..)
- TOUS équipements électroniques ETTEINTS (phones, calculatrices . .)

EXERCICE 1 (10 Pts) ORGANIGRAMME vs PRG ASM 'x86' (Temps_réf. : 25 mn)

On donne l'organigramme suivant :



1) Identifier (I1), (I2), (I3) et (I4) ? (*Indiquer les instructions assembleur 'x86' correspondantes*)

I1 : « SHR AX (ou DX), 1 » ou « DIV AX (ou DX), 2 »
I2 : « MOV [SI], AX (ou DX) »
I3 : « PUSH AX (ou DX) »
I4 : « ADD SI , 2 »

2) Cet organigramme comporte **UNE** erreur (instruction qui manque) : corriger cette erreur.

Entre « AX ← [SI] » et « AX ← AX AND 1 » : insérer «DX ← AX» .. en conséquence, la réponse de Q1 devient :
I1 : « SHR DX, 1 » ou « DIV DX, 2 »
I2 : « MOV [SI], DX »
I3 : « PUSH DX »
I4 : « ADD SI , 2 »

3) Proposer un programme en assembleur 'x86' correspondant à cet organigramme (l'utilisation de l'instruction « LOOP » est souhaitable).

(VU en TD !!) :

```
            MOV CX, 20H
Debut :     MOV AX, [ SI ]
            MOV DX, AX
            AND AX, 1
            JZ Cas_PAIR
            PUSH DX
```

```
                JMP SUITE
Cas_PAIR:       SHR DX, 1
                MOV [ SI ], DX
SUITE :         ADD SI, 2
                LOOP Debut
                < INT 21H>
```

4) Indiquer, parmi les propositions suivantes, celle/celles qui correspond/ correspondent à la fonction de ce programme :
   a. Traite 20 datas de 2 octets chacune ;
   b. Tri de parité de 20 datas de 1 octet chacune ;
   c. Effectue la division 20 fois ;
   d. Autre : TRI de PARITé de 20H=32 données de 2 octets : les datas impaires sont stockées en pile, les datas paires sont remplacées/écrasées par leur moitié dans leur même emplacement de la zone SI.

   NB : si vous sélectionnez le choix « d », vous devrez formuler une proposition personnelle et précise.

5) Déduire de l'organigramme **le nombre** et **la taille** des données traitées : 20H=32 données de 2 octets **justifier** CX=20H=32 données (nombre d'itérations), 1 data par itération ; taille=2 octets car chaque donnée est chargée depuis [SI] vers AX dont la taille est 2 octets
6) Si la valeur de (SI)_initiale = 1060H, quelle sera sa valeur finale (SI_Finale) ?

SI_Finale = SI_Initiale + Nbre_déplacements = 1060 + 2*20H = 10A0H

EXERCICE 2 (10 Pts) ANALYSE & SYNTHESE / PRG ASM 'x86' (Temps_réf. : 45 mn)

On considère le programme en assembleur 'x86', listé ci-dessous :

Indications générales :

(1) La zone (SI) contient 4 données de 2 octets chacune, désignées (D0, D1, D2 et D3);

(2) Pour chaque data (Di) à l'ordre (i), on désignera par (DiH) l'octet de poids Fort et (DiL) l'octet de poids faible ;

(3) L'utilisation du registre (BP) est **identique** à celle de (BX)

```
MOV SI , 1000H;          SI = 1000 H // @ des datas à traiter
MOV BX, SI ;             BX = 1000 H // positionne BX …
ADD BX , 6H;             BX = 1006 H // … par rapport à SI (décalage de +6
                                        Octets => BX pointe sur D3 (4è data)
MOV AX , [SI] ;          chargement DATA_0 ds AX : AX = D0
PUSH AX ;                save en pile : [ SP ] = D0
MOV DL , [BX] [1] ;  Deplacement (D3H) → Registre DL // octet FORT. ..
MOV [SI], DL ;           Ecrasement . . . @ de (D0L) ← octet FORT D3=D3H
MOV DL , [BX] ;      Deplacement (D3L) → Registre DL // octet Faible. ..
MOV [SI+1], DL ;         Ecrasement . . . @ de (D0H) ← octet Faible D3=D3L
SUB BX,2 ;               BX pointe sur D2 (3è data)
MOV AX , [SI] [2] ;      chargement DATA_1 ds AX : AX = D1
PUSH AX ;                save en pile : [ SP ] = D1
MOV DL , [BX] [1] ;  Deplacement (D2H) → Registre DL // octet FORT. ..
MOV [SI+2], DL ;         Ecrasement . . . @ de (D1L) ← octet FORT D2=D2H
MOV DL , [BX] ;      Deplacement (D2L) → Registre DL // octet Faible. ..
MOV [SI+3], DL ;         Ecrasement . . . @ de (D1H) ← octet Faibl D2=D2L
POP DX ;             chargement DATA_1 depuis PILE ds DX : DX = D1
MOV [SI+4], DH ;         Ecrasement . . . @ de (D2L) ← octet FORT D1=D1H
MOV [SI+5], DL ;         Ecrasement . . . @ de (D2H) ← octet Faible D1=D1L
POP DX ;             chargement DATA_0 depuis PILE ds DX : DX = D0
MOV [SI+6], DH ;         Ecrasement . . . @ de (D3L) ← octet FORT D0=D0H
MOV [SI+7], DL ;         Ecrasement . . . @ de (D3H) ← octet Faible D0=D0L
```

ANALYSE du PROGRAMME :

1) Commenter les instructions de ce programme. (Voir ci-haut)

2) En déduire la fonction générale de ce programme. : Ecrasement des 4 données D0 à D3 par D3 à D0, en inversant l'ordre des données et des octets (faible → FORT .. & ..FORT → faible)

3) Quelle sera la valeur finale de BX ? BX = 1000H + 6 – 2 = 1004 H

4) Quelle sera la valeur finale de SI ? SI = 1000 H (inchangée)

SYNTHESE du PROGRAMME :

5) **Réduire** la taille de ce programme en introduisant des structures répétitives (sauts conditionnels, boucles, ..) (**proposer une version optimisée de ce programme**)

```
                MOV SI , 1000H;
                MOV BX, SI ;
                ADD BX , 6H;

                MOV BP, 0 ;
                MOV CX, 2 ;
REPETER_1 :
                MOV AX , [SI+ BP] ;
                PUSH AX ;
                MOV DL , [BX] [1] ;
                MOV [SI+BP], DL ;
                MOV DL , [BX]  ;
                MOV [SI+BP +1], DL ;
                SUB BX,2 ;
                ADD BP, 2 ;
                LOOP REPETER_1 ;

                MOV CX, 2 ;
REPETER_2 :
                ADD BP, 2 ;
                POP DX ;
                MOV [SI+BP], AH ;
                MOV [SI+BP+1], AL ;
                LOOP REPETER_2 ;
```