S O L U T I O N

Exercice 1 (3 * 2 Pts) (Temps_référence : 15 mn)

Q1/ Donner les valeurs du registre **AL** après exécution des instructions suivantes :

```
MOV   AL , 20 ;   AL = 20
SHL   AL , 8 ;    AL = 00
SHR   AL , 8 ;    AL = 00
```

Q2/ Pour le programme suivant, trouver la valeur de **(V)** telle que la valeur **FINALE** du registre **AX** soit = **4800H** :

```
MOV   AX , 72 ;   AX = 72 = 0048H
ROL   AX , V ;    AX = 4800H => V= 8
```

Q3/ On donne la valeur initiale: CX = **0101 1010 0101 1010 b** ;
Donner les valeurs (finales) des registres **BX** et **CX pendant et après** l'exécution du programme suivant :

```
PUSH CX          ; CX = 5A5Ah ; [SP,SP-1]=5A5Ah   BX = ??
INC SP           ; CX = 5A5Ah ; [SP,SP-1]=??5Ah   BX = ??
AND  CX , FFFFH ; ; CX = 5A5Ah                    BX = ??
SHR  CX , 4H ;    ; CX = 05A5h                    BX = ??
MOV  BL , 3H ;    ; CX = 05A5h                    BX = 0003H
XOR  BX , BX ;    ; CX = 05A5h                    BX = 0000h
POP  CL ;         ; CX = 055Ah                    BX = 0000h
```

Exercice 2 (8 Pts) *SYNTHESE D'UN PRG ASM 'x86'* (Temps_référence : 20 mn)

On demande, un programme assembleur 'x86', répondant aux étapes et exigences suivantes :

Hypothèses : (1) Valeurs initiales de (SI), (DI) et (SP) quelconques ;

Etapes du programme :

1)- chargement d'une donnée de 2 octets, pointée par (SI + 1), vers un registre 16 bits (= donnée **chargée**) ; **1.0pt**

2)- sauvegarde de cette donnée dans un autre registre 16 bits ; (= donnée **sauvegardée**) ; **1.0pt**

3)- application d'un masque pour identifier le 1er et le 15ème bit de cette donnée chargée ; **1.5pt**

4)- si ces bits (à identifier) sont tous deux nuls, on transfert la donnée sauvegardée dans la pile ;

5)- si ces bits (à identifier) sont tous deux égaux à « 1 », on transfert la donnée sauvegardée vers une adresse pointée par (DI) ; (4) & 5) : **3.5pt**

6)- On réitère ces étapes jusqu'à la fin de traitement de 50 données successives. **1.0pt**

```
6)-                MOV CX, 50
1)-                MOV AX, [SI + 1] ; ( = donnée chargée ) ;
2)-                MOV DX, AX ; ( = donnée sauvegardée ) ;
3)-                AND AX, 4001H ;
4) & 5)-           JZ Premier&15emeTs2Nuls ;
                   JMP Premier&15emeBitNonNuls
Premier&15emeTs2Nuls:
                   PUSH DX
                   JMP RIEN_a_FAIRE
Premier&15emeBitNonNuls :
                   MOV BX, AX
                   AND AX, 1
                   JZ RIEN_a_FAIRE
                   MOV AX, BX
                   AND AX, 4000H
                   JZ RIEN_a_FAIRE ; (*)
                   MOV [DI], DX;   1er&15embitTs2=1
                   ADD DI, 2
RIEN_a_FAIRE :     ADD SI,2
6)-                LOOP RELANCE
                   END
```

NB : Autres solutions :

1/ Les lignes 10 à 14 [« *AND AX, 1* » à « *JZ Rien_a_faire ;(*)* »] peuvent être remplacées par [*CMP AX, 4001H ; JZ Rien_a_faire ; (*)*]

2/ Le masque '4001H' sur AX peut être remplacé par '40H' sur AH suivi de '01H' sur AL séparément.

Exercice 3 (*6 Pts*) *ANALYSE D'UN PRG ASM 'x86'* *(Temps_référence : 40 mn)*

On donne le programme suivant, (vu en T. D.) :

```
Jmp start

Data_Sce        db      50, 20, 45, 1, 35, 46, 47, 99, 7, 10

1.      start:          lea     si, Data_Sce    ; charge dans SI l'@ du 1er élément
2.                      mov     di, si
3.                      add     di, 50          ; DI est supposée = SI+50
4.                      mov     cx, 10          ; Boucle externe : 10 itérations
5.      bcl_ext:        mov     dx, cx
6.                      dec     dx              ; boucle interne
7.                      push    si
8.                      mov     al, [si]
9.      bcl_int:        inc     si
10.                     mov     bl, [si]
11.                     cmp     al, bl
12.                     jle     ALppetiKeBL
13.                     mov     [si], al
14.                     xchg    al, bl
15.                     jmp     AL_TJR_Min
16.     ALppetiKeBL:    mov     bp, cx
17.                     sub     bp, dx
18.                     mov     [si-bp], bl
19.     AL_TJR_Min:     dec     dx
20.                     jnz     bcl_int
21.                     mov     [di], al
22.                     inc     di
23.                     pop     si
24.                     loop    bcl_ext
25.                     ret
```

1) Modifier le programme pour obtenir un « TRI par valeurs DECROISSANTES ». **1.5pt**

(13) mov [SI], al → mov [SI], bl

(18) mov [SI-bp], bl → mov [SI-bp], al

(21) mov [DI], al → mov [DI], bl

NB : Autres solutions :

1/ Les lignes 12 « **jle ALppetiKeBL** » → « **jle ALppetiKeBL** »

2/ Les lignes 13,14 & 15 ←→ lignes 16, 17 & 18

2) Modifier le programme initial pour obtenir un « TRI CROISSANT de 10 données de 2 octets chacune » **1.0pt**

AL → AX
BL → BX
Data_Sce db (. . .) → Data_Sce dw (. . .)

3) Quel est l'intérêt de la ligne (13) ? Quel est l'intérêt de la ligne (18) ? Expliquer **1.0pt**

Ecrasement du "min" dans la zone (SI)

4) On remplace (en ligne 7) l'instruction « PUSH SI » par les 2 instructions successives :
« PUSH SI » et
« ADD SP, 2 » **1.0pt**

Modifier le programme initial de façon à garder la même fonction du programme.

en ligne 23 : « POP SI » → « SUB SP,2
POP SI »

5) On remplace les deux instructions (19 et 20) par l'instruction « loop BCL_INT » ; quelles seront les conséquences pour ce nouveau programme modifié ? **1.5pt**

Boucle infinie