**TP 4 - Solution**

**Module: Architectures des ordinateurs**

**Université de Jijel**

**2ième année Licence Informatique-2020/2021**

**Exercice 1**

Expliquer ce que fait chacun des trois programmes suivants :

;prgm3

**include 'emu8086.inc'**

data segment

**tab dw 0,2,4,6,8,10,12,14,16,18**

data ends

code segment

assume ds:data,cs:code

start:

MOV AX,data

MOV DS,AX

**print 'Hello'**

**printn ' World!'**

printn 'Enter a number

between 1 and 10:'

**CALL scan\_num**

CMP CX,1

JB lab2

CMP CX,10

JA lab2

lab0:MOV SI,0

lab1:MOV AX,tab[SI]

**CALL print\_num**

PRINTN ''

ADD SI,2

LOOP lab1

JMP lab3

lab2:print 'Error'

lab3:hlt

code ends

DEFINE\_PRINT\_NUM

DEFINE\_PRINT\_NUM\_UNS

DEFINE\_SCAN\_NUM

end start

;prgm2

data segment

x dw 5h

y dw 6h

msg db **'Hello'**

**tab1 dw 2, 4, 6, 8, 10**

tab2 dw 5 **dup**(0)

data ends

code segment

assume ds:data,cs:code

start:

MOV AX,data

MOV DS,AX

MOV AX,0

MOV CX,5

MOV **SI**,0

lab: ADD AX**,tab1[SI]**

**ADD SI,2**

**LOOP lab**

code ends

end start

;prgm1

**data segment**

x **dw** 5h

y **dw** 6h

z **dw** 0

**data ends**

**code segment**

**assume ds:data,cs:code**

**start:**

**MOV AX,data**

**MOV DS,AX**

MOV AX,x

MOV BX,y

ADD AX,BX

MOV z,AX

MOV BX, z

hlt

**code ends**

**end start**

Declaring (initialized) variables in .data section) :

<label> <type> <value>

Types:

**db** **b**yte

**dw** **w**ord

**dd** **d**ouble-word

**dq** quad-word

***n dup(value):*** duplicate value for *n* times.

* DI: the Destination Register
* IP: the Instruction Pointer
* **CS: the Code Segment Register**
* **DS: the Data Segment Register**
* SS: the Stack Segment Register
* ES: the Extra Segment Register
* LOOP: Decrease CX, jump to label if CX not zero.
* SCAN\_NUM: procedure that gets a SIGNED number from the keyboard, and stores the result in CX register. To use it declare: DEFINE\_SCAN\_NUM before END directive.
* RINT\_NUM:procedure that prints a signed number in AX register. To use it declare: DEFINE\_PRINT\_NUM and DEFINE\_PRINT\_NUM\_UNS before END directive.
* PRINT *string*: macro with one parameter, allows to print out a string.
* PRINTN *string*: macro with 1 parameter prints out a string. It is the same as PRINT but automatically adds *carriage return* at the end of the string.
* AX: the Accumulator
* BX: the Base Register
* **CX: the Count Register**
* DX: the Data Register
* SP: the Stack Pointer
* BP: the Base Pointer
* **SI: the Source Index Register**

**Exercice 2**

**1)** Ecrire un programme, en langage assembleur 8086, qui permet de compter les nombres nuls dans un tableau d’octets de longueur 30 et débutant à l’adresse [00h], le résultat sera placé à l’adresse [200h].

**2)** Ajouter ou programme précédent les déclarations ci-dessous, exécuter le programme puis expliquer les résultats.

msg db 'Hello world'

tab2 dw 5 dup(1)

**;Exo2- Solution 2**

data segment

;données

msg db 'Hello world'

tab2 dw 5 dup(1)

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

;code

MOV SI,00H

MOV BX,00h

MOV CX,30

;boucle et test

etq2: CMP [SI],0

JNZ etq1

INC BX

etq1: INC SI

**LOOP etq2**

MOV [200h], BX

hlt

code ends

end start

**;Exo2-Solution 1**

data segment

;données

msg db 'Hello world'

tab2 dw 5 dup(1)

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

;code

MOV SI,00H

MOV BX,00h

MOV CX,30

;boucle et test

etq2: CMP [SI],0

JNZ etq1

INC BX

etq1: INC SI

DEC CX

JNZ etq2

MOV [200h], BX

hlt

code ends

end start

**Exercice 3**

**1)** Ecrire un programme qui permet de trier par ordre croissant un tableau de longueur N = 100h débutant à l’adresse [200h].

**; Un programme qui permet de trier les éléments d’un tableau**

data segment

Tab dw 12, 46, 2, 11, 9, 22, 3, 7, 1, 31

nbr dw 10 ; nombre des éléements du tableau Tab

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

LEA AX, Tab ; Lire l'adresse du tableau

PUSH AX ; Empile l'adresse du tableau

PUSH nbr ; Empile le nombre des éléments du tableau

CALL triTab ; appel de la procédure de tri

hlt

triTab proc near ;début de la procédure de tri

MOV BP, SP ; adresse sommet pile

MOV SI, [BP+4] ; Lire l'adresse du tableau

MOV CX, [BP+2] ; Lire le nombre des éléments du tableau lab1: MOV AX, [SI]

MOV BX, SI

MOV DX, CX

lab2: DEC DX

CMP DX,0

JE lab3

ADD BX, 2

CMP AX, [BX]

JLE lab2

XCHG AX, [BX]

MOV [SI], AX

JMP lab2

lab3: ADD SI,2

LOOP lab1

RET

triTab endP ;fin de la procédure

code ends

end start

**; Un programme qui permet de trier les éléments d’un tableau**

**; débutant à l’adresse 200h**

include 'emu8086.inc'

data segment

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

; Appel de la procédure de Tri

PUSH 200h

PUSH 100

**CALL triTab**

; Appel de la procédure d'affichage

PUSH 200h

PUSH 100

**CALL afficherTab**

hlt

; Définition de la procédure de tri

; Définition de la procédure d’affichage

code ends

DEFINE\_PRINT\_NUM

DEFINE\_PRINT\_NUM\_UNS

DEFINE\_SCAN\_NUM

end start

**; Un programme qui permet d’afficher un tableau**

include 'emu8086.inc'

data segment

Tab dw 11, 4, 6, 8, 10, 12, 14

nbr dw 7 ; nombre des éléments du tableau Tab

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

LEA AX, Tab ; Lire l’adresse du tableau

PUSH AX ; Empile l'adresse du tableau

PUSH nbr ; Empile le nombre des éléments du tableau

**CALL afficherTab**; Appel de la procédure d’affichage

hlt

afficherTab proc near ; début de la procédure

MOV BP, SP ; adresse sommet pile

MOV SI, [BP+4] ; Lire l'adresse du tableau

MOV CX, [BP+2] ; Lire le nombre des éléments du tableau

lab: MOV AX, [SI]

CALL print\_num

PRINTN ''

ADD SI,2

LOOP lab

RET

afficherTab endP ; fin de la procédure

code ends

DEFINE\_PRINT\_NUM

DEFINE\_PRINT\_NUM\_UNS

DEFINE\_SCAN\_NUM

end start

**Exercice 4**

**1)** Ecrire une procédure (appelée *somme*) permettant de calculer la somme de deux nombres. Cette procédure utilise le passage de paramètres par registres: Les deux nombres sont passés par les registres AX et AX et le résultat sera placé dans AX.

**2)** Appeler la procédure *somme* à partir d’un autre programme (le programme appelant).

**;Exo4-Q1 et Q2-Solution 1**

;EXO4

data segment

; Déclaration des variables

x dw 4h

y dw 6h

z dw 0

data ends

code segment

assume ds:data,cs:code

start:

MOV AX,data

MOV DS,AX

; Début du code du programme appelant

MOV AX, x

MOV BX, y

**call** **somme** ; appel de la procédure

MOV y, AX

hlt

;définition de la procédure

**somme** proc near ; Début de la procédure

ADD AX, BX

**RET**

**somme** endP ; Fin de la procédure

code ends

end start

**;Exo4-Q1 et Q2-Solution 2**

;EXO4

data segment

; Déclaration des variables

x dw 4h

y dw 6h

z dw 0

data ends

code segment

assume ds:data,cs:code

start:

MOV AX,data

MOV DS,AX

; Début du code du programme appelant

MOV AX, x

MOV BX, y

**call** **somme** ; appel de la procédure

MOV y, AX

hlt

;définition de la procédure

**somme:**

ADD AX, BX

**RET**

code ends

end start

**3**) Redéfinir la procédure précédente en utilisant le mode de passage de paramètres par pile.

**;Exo4-Q3-Solution 2**

data segment

; Déclaration des variables

x dw 4h

y dw 6h

z dw 0

data ends

code segment

assume ds:data,cs:code

start:

MOV AX,data

MOV DS,AX

; Début du code du programme appelant

PUSH x  ; Empile argument 1

PUSH y ; Empile argument 2

**call** **somme** ; Appel de la procédure

MOV y, AX

hlt

; Définition de la procédure

**somme** proc near

**PUSH BP** ; Sauvegarde BP

MOV BP, SP ; Adresse sommet pile

MOV AX, [BP+**4**] ; Charge argument 2

ADD AX, [BP+**6**] ; Charge argument 1

**POP BP** ; restaure ancien BP

**RET**

**somme** endP

code ends

end start

**;Exo4-Q3-Solution 1**

data segment

; Déclaration des variables

x dw 4h

y dw 6h

z dw 0

data ends

code segment

assume ds:data,cs:code

start:

MOV AX,data

MOV DS,AX

; Début du code du programme appelant

PUSH x  ; Empile argument 1

PUSH y ; Empile argument 2

**call** **somme** ; Appel de la procédure

MOV y, AX

hlt

; Définition de la procédure

**somme** proc near

MOV BP, SP ; Adresse sommet pile

MOV AX, [BP+2] ; Charge argument 2

ADD AX, [BP+4] ; Charge argument 1

**RET**

**somme** endP

code ends

end start

**1)** L'appel de la procédure *somme* avec passage par la pile est :

**PUSH x** ; x = 4

**PUSH y** ; y = 6

**CALL somme**

**2)** La procédure *somme* va lire la pile pour obtenir la valeur des paramètres. Pour cela, il faut bien comprendre quel est le contenu de la pile après le CALL :

SP → IP (adresse de retour)

SP+2 → x = 6 (deuxième paramètre)

SP+4 → x = 4 (premier paramètre)

**3)** La valeur de retour est laissée dans AX.

**4)** La deuxième version de la procédure ne modifie pas la valeur contenue par BP avant l'appel.

**Exercice 5**

Définir puis appeler chacune des procédures suivantes :

1. *fact*: qui calcule le factoriel n! d’un entier naturel.
2. *fibo* : qui calcule n termes de la suite de Fibonacci : U0=1, U1=1, Un+1=Un + Un-1.
3. *Pgcd* : qui calcule le plus grand commun diviseur de deux nombres.
4. Une procédure qui permet de déterminer le maximum dans un tableau d’entiers.

**;Exo5-Q4 - Max de Tab**

data segment

Tab dw 12, 46, 2, 11, 9, 22, 83, 7, 1

nbr dw 9

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

LEA AX, Tab

PUSH AX

PUSH nbr

CALL maxTab

hlt

**maxTab** proc near

MOV BP, SP

MOV SI, [BP+4]

MOV CX, [BP+2]

MOV AX, [SI]

lab1: DEC CX

CMP CX,0

JE lab2

ADD SI,2

CMP AX, [SI]

JGE lab1

MOV AX, [SI]

JMP lab1

lab2: **RET**

**maxTab** endP

code ends

end start

**;Exo5-Q2 - Fibo**

data segment

x dw 4

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

CMP x,0

JL fin

PUSH x

**call** **fact**

fin: hlt

**fibo** proc near

MOV BP, SP

MOV DX, [BP+2]

MOV AX,1

MOV BX,1

AND DX,DX

JZ finFibo

boucle: SUB DX,1

JZ finFibo

MOV CX,AX

ADD AX,BX

MOV BX,CX

JMP boucle

finFibo: **RET**

**fibo** endP

code ends

end start

**;Exo5-Q1 - Fact**

data segment

x dw 4

data ends

code segment

assume ds:data,cs:code

start:

**MOV AX,data**

**MOV DS,AX**

CMP x,0

JL fin

PUSH x

**call** **fact**

fin: hlt

**fact** proc near

MOV BP, SP

MOV AX, [BP+2]

AND AX,AX

JNZ lab1

MOV AX,1

JMP finFact

Lab1: MOV BX,AX

lab2: DEC BX

JZ fin

MUL BX

JMP lab2

finFact: RET

**RET**

**fact** endP

code ends

end start

**Rappel de cours**

**Passage de paramètre par pile :** Le registre BP est utilisé. Il permet de lire des valeurs sur la pile sans les dépiler ni modifier SP. BP permet un mode d'adressage indirect spécial.

**Exemple :**

Le code suivant permet de lire (charger dans le registre AX) le sommet de la pile sans dépiler.

**MOV BP, SP** ; BP pointe sur le sommet

**MOV AX, [BP]** ; Lit sans dépiler.

Le mot suivant est lu avec :

**MOV AX, [BP+2]** ; + 2 car chaque élément de la pile contient 2 octets.

* ***SP*** (Stack Pointer) : Ce registre pointe sur le sommet de la pile et il est mis à jour automatiquement par les instructions d'empilement et de dépilement
* ***BP***(Base Pointer) : Ce registre pointe la base de la région de la pile contenant les données accessibles (variables locales, paramètres, etc.) à l'intérieur d'une procédure.
* En général, l'accumulateur est utilisé pour transmettre le résultat des procédures.