Exemple 2 :
L'instruction: bgt $t0, 0, debut peut être simplifiée par l'instruction:
bgtz $t0, debut

10. Solution de la série d'exercices n°2

Exercice 1
Ecrire un programme en assembleur qui permet d'afficher les caractères de l'alphabet en majuscule. La solution doit se baser principalement sur la manipulation du code ASCII.

Solution :

L'idée de base peut être exprimée d'une manière algorithmique par :

FOR code_ascii =65 → 90 DO afficher_carcatere (code_ascii)

Ce qui peut être réécrit par :

code_ascii←65

debut_boucle :

afficher_carcatere (code_ascii)
code_ascii← code_ascii+1

IF code_ascii> 90 THEN se brancher vers "sortir_boucle"
Se brancher vers "debut_boucle"

sortir_boucle:

Cette partie est traduite en assembleur par (dans cette solution, le code ascii est mémorisé dans le registre $t0):

```
li $t0, 65   # code ascii de A dans $t0

debut_boucle :

# afficher_carcatere

move $a0, $t0
li $v0, 11
syscall

add $t0, $t0, 1   # passer au code ascii suivant
```

```
bgt $t0,  90, sortir_boucle  # test de sortie de la boucle

b debut_boucle

sortir_boucle :
```

Le programme complet est alors :

```
.text
.globl main
.ent main
main:

li $t0, 65   # code ascii de A dans $t0

debut_boucle :

# afficher_carcatere

move $a0, $t0
li $v0, 11
syscall

add $t0, $t0, 1   # passer au code ascii suivant

bgt $t0,  90, sortir_boucle  # test de sortie de la boucle

b debut_boucle


sortir_boucle :

li $v0, 10
syscall

.end main
```

Exercice 2 :

Dans la solution qui a été proposé pour l'exercice 3 de la série 1, aucun contrôle de la saisie n'a été fait. En effet n'importe quel symbole peut être saisi, pas uniquement les caractères en minuscule. Ajouter le contrôle de la saisie à cette solution.

Solution :

Le contrôle de la saisie peut être exprimé d'une manière algorithmique par :

debut:
lire (caractère)

IF 97 ≤code ASCII (caractère) ≤122 THEN se brancher vers "suite"

Se brancher vers debut

suite:
... la suite du programme.....

Ce qui peut être réécrit par :

debut:
lire (caractère)

IF 97 ≤code ASCII (caractère) THEN se brancher vers "suiteIF"
Se brancher vers debut
suiteIF :
IF code ASCII (caractère) ≤122 THEN se brancher vers "suite"
Se brancher vers debut

suite:
... la suite du programme.....

Cette partie est traduite en assembleur par :

debut:

li $v0, 12
syscall

bge $v0, 97, suiteIF
j debut
suiteIF :
ble $v0, 122, suite
j debut

suite:

... la suite du programme.....

Le programme devient alors :

```
.data
save: .space 1  #pour sauvegarder le caractere saisi
.text
.globl main
.ent main
main:
```

```
# **********lire le caractere
debut:
li $a0, 0x0a
li $v0, 11
syscall

li $v0, 12
syscall

bge $v0,  97, suiteIF
j debut
suiteIF:
ble $v0,  122, suite
j debut

suite:

#**********sauvegarder le caractere
sb $v0, save

#**********saut de ligne
li $a0, 0x0a   # code ascii(\n) =10
li $v0, 11
syscall

#********** transformer le caractere en majuscule..le resultat de la transformation
# dans $t0 **********

lb $t0, save
sub $t0, $t0, 32

#********** Affichage du caractere
move $a0, $t0
li $v0, 11
syscall

li $v0, 10
syscall
.end main
```

Exercice 3 :

```
.data
puiss: .word 0
.text
.globl main
.ent main
main:
```

```
# ********************lire le nombre
debut:

li $v0, 5
syscall

blt $v0,9, suite
j debut

suite:

li $t0, 0  # $t0 est le compteur de boucle
li $t1, 1  # $t1 pour recevoir le resultat de la puissance

debut2:

add $t0, $t0, 1

mul $t1, $v0, $t1
beq $t0, 10, suite2

j debut2

suite2:

sw $t1, puiss  # la variable puiss pour le resultat du calcul
#**********************************saut de ligne
li $a0, 10
li $v0, 11
syscall
#********************** Affichage de puiss
lw $a0, puiss
li $v0, 1
syscall

li $v0, 10
syscall
.end main
```

<u>Exercice 4</u> :

Partie I)

```
.data
Vect: .word 0, 0, 0, 0, 0
.text
.globl main
.ent main
main:
# ********************saisir dans le vecteur Vect ******************
```

```
la $t0, Vect  # mettre l'adresse de Vect dans $t0,
              # c-a-d $t0 pointe vers le 1er element de Vect

li $t1, 1     # $t1 est le compteur de la boucle

debut:

li $v0, 5    # lire
syscall      # l'entier

sw $v0, ($t0)  # mettre l'entier dans le vecteur

add $t0, $t0, 4 # passer à l'élément ou le word suivant de Vect

add $t1, $t1, 1   # incrementer le compteur de boucle

bgt $t1, 5, suite  # test de sortie de la boucle

b debut

suite:

#**********************************saut de ligne
li $a0, 10
li $v0, 11
syscall
# ********************afficher les elements du vecteur Vect
la $t0, Vect  #  $t0 pointe vers le 1er element de Vect
li $t1, 1   # $t1 est le compteur de la boucle

debut2:
lw $t2, ($t0)  # mettre l'element pointé par $t0 dans $t2
move $a0, $t2   #afficher l'element de Vect
li $v0, 1
syscall

li $a0, 32 #afficher un espace
li $v0, 11 #pour separer les
syscall    # elements affichés. code ascii=32

add $t0, $t0, 4 # passer à l'élément suivant.....

add $t1, $t1, 1   # incrementer le compteur de boucle

bgt $t1, 5, suite2   # test de sortie de la boucle

b debut2

suite2:
li $v0, 10
```

```
syscall
.end main
```

Partie II)
1)

```
.data
Vect: .byte 0, 0, 0, 0, 0
som: .word 0

.text
.globl main
.ent main
main:

# ********************saisir dans le vecteur Vect ******************
la $t0, Vect  # mettre l'adresse de Vect dans $t0,
            # c-a-d $t0 pointe vers le 1er element de Vect

li $t1, 1     # $t1 est le compteur de la boucle

debut:
li $v0, 5    # lire
syscall      # l'entier

sb $v0, ($t0)  # mettre l'entier dans le vecteur

add $t0, $t0, 1 # passer à l'élément ou le byte suivant de Vect

add $t1, $t1, 1   # incrementer le compteur de boucle

bgt $t1, 5, suite  # test de sortie de la boucle

b debut
suite:

#*********************************saut de ligne
li $a0, 10
li $v0, 11
syscall

# ********************calculer la somme des elements du vecteur Vect
la $t0, Vect  #  $t0 pointe vers le 1er element de Vect
li $t1, 1   # $t1 est le compteur de la boucle
li $t3, 0  # $t3  pour contenir la somme des elements de Vect

debut2:

lb $t2, ($t0)  # mettre l'element pointé par $t0  dans $t2
add $t3, $t3, $t2   #  c-a-d  $t3 <-- $t3 + element de Vect
```

```
add $t0, $t0, 1 # passer à l'élément suivant.....
add $t1, $t1, 1   # incrementer le compteur de boucle
bgt $t1, 5, suite2   # test de sortie de la boucle
b debut2

suite2:

sw $t3, som   # transferer la somme dans la variable som
lw $a0, som   #afficher la somme
li $v0, 1
syscall

li $v0, 10
syscall
.end main
```

2)

```
.data
Vect: .byte 0, 0, 0, 0, 0
prod: .word 0

.text
.globl main
.ent main
main:

# ********************saisir dans le vecteur Vect ******************
la $t0, Vect  # mettre l'adresse de Vect dans $t0,
            # c-a-d $t0 pointe vers le 1er element de Vect
li $t1, 1     # $t1 est le compteur de la boucle

debut:

li $v0, 5    # lire
syscall      # l'entier

sb $v0, ($t0)  # mettre l'entier dans le vecteur
add $t0, $t0, 1 # passer à l'élément ou le byte suivant de Vect
add $t1, $t1, 1   # incrementer le compteur de boucle
bgt $t1, 5, suite  # test de sortie de la boucle

b debut

suite:
#**********************************saut de ligne
li $a0, 10
li $v0, 11
syscall
# ********************calculer le produit des elements du vecteur Vect
la $t0, Vect  #  $t0 pointe vers le 1er element de Vect
```

```
li $t1, 1   # $t1 est le compteur de la boucle
li $t3, 1  # $t3  pour contenir le produit des elements de Vect

debut2:
lb $t2, ($t0)  # mettre l'element pointé par $t0  dans $t2
mulo $t3, $t3, $t2   #  c-a-d  $t3 <-- $t3 * element de Vect

add $t0, $t0, 1 # passer à l'élément suivant.....
add $t1, $t1, 1   # incrementer le compteur de boucle
bgt $t1, 5, suite2   # test de sortie de la boucle

b debut2

suite2:
sw $t3, prod   # transferer produit dans la variable prod
lw $a0, prod   #afficher le produit
li $v0, 1
syscall
li $v0, 10
syscall
.end main
```

Exercice5:

1)

```
.data
Vect: .byte 0, 0, 0, 0, 0
MIN: .byte 0
MAX: .byte 0
message1: .asciiz "MIN = "
message2: .asciiz "  MAX = "

.text
.globl main
.ent main
main:

# ********************saisir dans le vecteur Vect ******************
la $t0, Vect  # mettre l'adresse de Vect dans $t0,
            # c-a-d $t0 pointe vers le 1er element de Vect

li $t1, 1     # $t1 est le compteur de la boucle

debut:
li $v0, 5    # lire
syscall      # l'entier

sb $v0, ($t0)  # mettre l'entier dans le vecteur
```

```
add $t0, $t0, 1 # passer à l'élément ou le word suivant de Vect
add $t1, $t1, 1   # incrementer le compteur de boucle
bgt $t1, 5, suite  # test de sortie de la boucle

b debut

suite:
#**********************************saut de ligne
li $a0, 10
li $v0, 11
syscall

# ********** trouver l'element MIN et l'element MAX du vecteur Vect
la $t0, Vect  #  $t0 pointe vers le 1er element de Vect
li $t1, 1   # $t1 est le compteur de la boucle

lb $t3, ($t0)  # mettre le 1er element de Vect dans $t3   $t3 va contenir le MIN
lb $t4, ($t0)  # mettre le 1er element de Vect dans $t4   $t4 va contenir le MAX
debut2:

add $t0, $t0, 1 # passer à l'élément suivant.....
add $t1, $t1, 1   # incrementer le compteur de boucle
lb $t2, ($t0)  # mettre l'element pointé par $t0 dans  $t2
blt $t2, $t3, maj_MIN
b test_MAX

maj_MIN:
move $t3, $t2

test_MAX:
bgt $t2, $t4, maj_MAX
b testing

maj_MAX:
move $t4, $t2

testing:
beq $t1, 5, suite2   # test de sortie de la boucle
b debut2

suite2:
sb $t3, MIN
sb $t4, MAX

la $a0, message1  #afficher message 1
li $v0, 4
syscall

lb $a0, MIN   #afficher MIN
li $v0, 1
```

```
syscall
la $a0, message2  #afficher message 2
li $v0, 4
syscall

lb $a0, MAX   #afficher MAX
li $v0, 1
syscall

li $v0, 10
syscall

.end main
```

2)

```
.data
Vect: .byte 0, 0, 0, 0, 0

.text
.globl main
.ent main
main:

# ********************saisir dans le vecteur Vect ******************
la $t0, Vect  # mettre l'adresse de Vect dans $t0,
                # c-a-d $t0 pointe vers le 1er element de Vect

li $t1, 1     # $t1 est le compteur de la boucle

debut:
li $v0, 5    # lire
syscall      # l'entier

sb $v0, ($t0)  # mettre l'entier dans le vecteur

add $t0, $t0, 1 # passer à l'élément ou le word suivant de Vect

add $t1, $t1, 1   # incrementer le compteur de boucle

bgt $t1, 5, suite  # test de sortie de la boucle

b debut

suite:
#**********************************saut de ligne
li $a0, 10
li $v0, 11
syscall
# ********************trier  les elements du vecteur Vect
```

```
boucleTri:
li $t4, 0   # $t4 est utilisé pour signaler les permutations
la $t0, Vect  #  $t0 pointe vers le 1er element de Vect
li $t1, 1   # $t1 est le compteur de la boucle

debut2:
lb $t2, ($t0)  # mettre l'element pointé par $t0 dans $t2
lb $t3, 1($t0)  # mettre l'element ayant l'adresse $t0+1 dans $t3.
                #Dans l'ecriture 1($t0), la valeur du pointeur t0 n'a
               # pas été changé malgré que l'expression 1($t0)  nous permet d'accéder à
             #l'element Vect[$t0+1]. Dans l'ecriture 1($t0), il n' ya pas eu
             #d'incrementation de $t0. Le mode d'adressage utilisé ici est l'adressage de
             # base, qui est une forme plus generale que l'adressage indirect.

bgt $t2, $t3, permuter  # si l'element Vect[i] > Vect[i+1] alors permuter(Vect[i],
                        #Vect[i+1])
b continue

permuter:
sb $t2, 1($t0)
sb $t3, ($t0)
li $t4, 1         # $t4=1 s'il y a eu une permutation

continue:
add $t0, $t0, 1 #  pointer l'élément suivant.....

add $t1, $t1, 1   # incrementer le compteur de boucle
blt $t1, 5, debut2   # test de sortie de la boucle
bnez $t4 boucleTri

# ********************afficher les elements du vecteur Vect après le tri
la $t0, Vect  #  $t0 pointe vers le 1er element de Vect
li $t1, 1   # $t1 est le compteur de la boucle

debut3:
lb $t2, ($t0)  # mettre l'element pointé par $t0 dans $t2
move $a0, $t2   #afficher l'element de Vect
li $v0, 1
syscall

li $a0, 32 #afficher un espace
li $v0, 11 #pour separer les
syscall    # elements affichés. code ascii=32


add $t0, $t0, 1 # passer à l'élément suivant.....
add $t1, $t1, 1   # incrementer le compteur de boucle
bgt $t1, 5, suite3   # test de sortie de la boucle
b debut3
```

```
suite3:
li $v0, 10
syscall
.end main
```