Université KASDI MERBAH – Ouargla
Faculté des Nouvelles Technologies de l'Information et de la Communication
Département d'Informatique et Technologie de l'information

Cours architectures des ordinateurs

Cours 5:
Les tableaux et les piles

Enseignante: Chafika Benkherourou

Introduction:

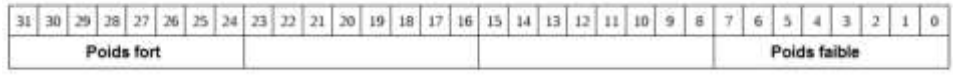
- Les principaux types de données manipulés par le MIPS R3000 sont: les entiers, les réels et les caractères.

- Les structures de données comme les tableaux et les piles sont utilisées en MIPS pour stocker des données les unes à la suite des autres.

- Pour les manipuler, il faut d'abord connaître comment ils sont stockés en mémoire.

Stockage en mémoire:

- La mémoire est vue comme un tableau d'octets, qui contient aussi bien les données que les instructions.
- L'unité d'adressage est l'octet.
- Les instructions sont codées sur 32 bits.
- Pour enregistrer une instruction quatre (4) octets sont nécessaires.
- La représentation d'un mot (32-bits) en mémoire est comme suit:

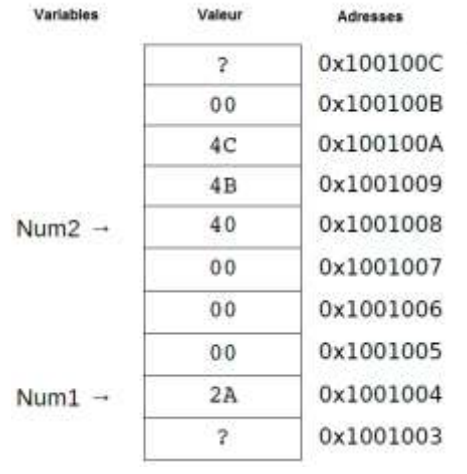


Stockage en mémoire:

Exemple:

- num1: .word 42
- num2: .word 5.000.000

- 42_{10} est représentée en hexadécimal par : 0x0000002A
- $5{,}000{,}000_{10}$ est représenté en hexadécimal par 0x004C4B40.
- La représentation de ces deux mots en mémoire est :



Les tableaux

- Un tableau est une zone mémoire dans laquelle les données sont rangées les unes à la suite des autres;
- Il est repéré par l'adresse du premier élément;
- On accède aux différentes cases par déplacement par rapport à la première;

Les tableaux

- Un tableau est déclaré comme suit:

```
.data
   T : .word    3, 6, 8, 9, 2
```

Cette déclaration permet l'allocation en mémoire d'un tableau de 5 éléments (mots de 32 bits) qui sont: {3,6,8,9,2}

Les tableaux

- Le programme suivant permet de faire la somme des éléments d'un tableau:

```
.data
    tab : .word  -2, 3, 7, -6,0
.text

main :
    li $a0, 0    #somme
    li $a1, 5    #taille
    la $a2, tab  #$a2 reçoit l'adresse du premier élément

bcl:

    lw  $a3, ($a2)       #lire une valeur du tableau
    add $a0, $a0, $a3    #ajouter la valeur lu dans $a0
    addi $a2, 4          #incrémenter le pointeur du tableau par 4
    addi $a1, -1         #décrémenter le compteur de la boucle bcl

    bne $a1, $zero, bcl  # si la valeur n'est pas nulle brancher sur bcl

li $v0, 1         #appel système pour print-int
syscall           # imprimer la somme
li $v0, 10
syscall
```

Les tableaux

- Pour récupérer l'adresse du premier élément dans le tableau, on utilise l'instruction suivante:

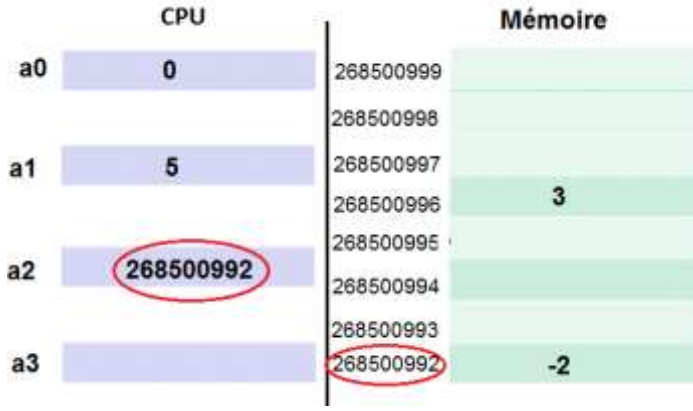
```
la  $a2, tab
```

- Après l'exécution de cette instruction le registre $a2 contient l'adresse du premier élément du tableau.

Les tableaux

- Le contenu des registres et de la mémoire après l'exécution de la partie suivante est:

```
.data
        tab : .word -2, 3, 7, -6, 0
.text
main :
        li $a0, 0       #somme
        li $a1, 5       #taille
        la $a2, tab     #$a2 reçoit l'adresse du premier élément
```



Les tableaux

- L'exécution de la boucle permet de faire la somme tant qu'on est pas arrivé à la fin du tableau. L'instruction suivante permet de brancher à la boucle si le contenu du registre $a1 n'est pas nul.

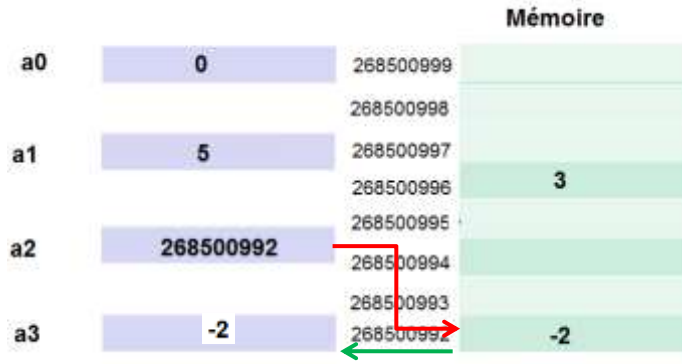
```
bne  $a1, $zero, bcl
```

- L'instruction qui permet de lire une valeur du tableau est la suivante:

```
lw  $a3, ($a2)
```

Les tableaux

Le contenu de l'adresse mentionnée par $a2 est sauvegardée dans le registre $a3.



Les tableaux

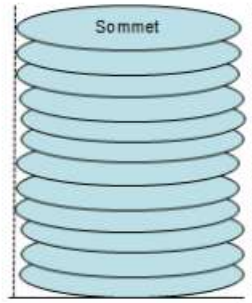
Exemple:

- Ecrire un programme qui permet de donner la somme des nombres négatifs et la somme des nombres positifs dans le tableau suivant:

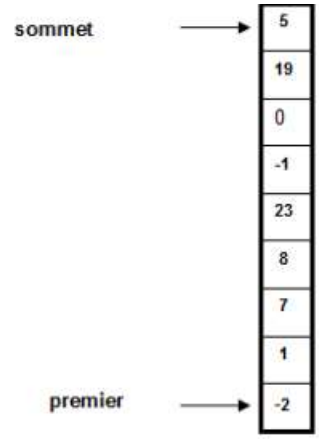
  Tab: { - 4, 5 , 8 , -1 , 6 , 7 , 13 , - 7 , - 8 }

Les piles

- La pile est une structure de données, qui permet de stocker les données dans l'ordre LIFO (Last In First Out) - Dernier Entré Premier Sorti).
- La récupération des données sera faite dans l'ordre inverse de leur insertion.



Pile d'assiettes

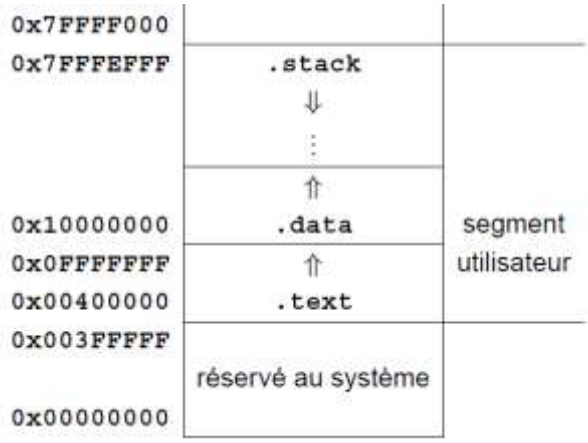


Les piles

- On peut comparer l'organisation de la pile à une pile d'assiettes: on peut parfaitement rajouter une assiette au sommet de la pile, ou enlever celle qui est au sommet, mais on ne peut pas toucher aux autres.
- La pile est utilisée dans la programmation pour stocker les informations durant les appels des procédures et des fonctions.
- Pour ajouter un élément à la pile on utilise l'opération **Push** (Empiler).
- Pour retirer un élément de la pile on utilise l'opération **Pop** (Dépiler).
- Remarque: les opérations Push et Pop sont implémentées manuellement.

Les piles

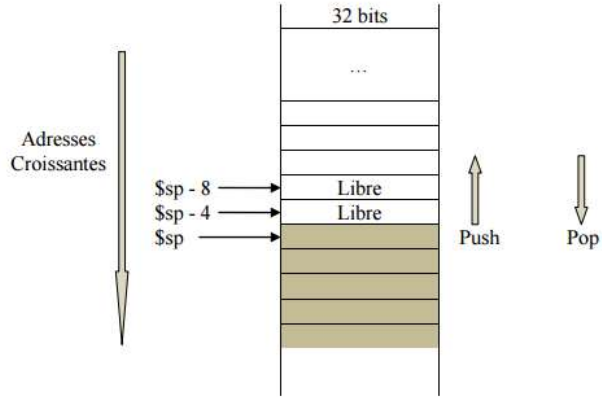
Organisation de la pile:



- La pile est stockée dans le segment **.stack**.

Les piles

Organisation de la pile:



- Contrairement aux sections **.data** et **.text**, la pile s'étend vers les adresses décroissantes.

Les piles

Organisation de la pile:

- Le sommet de la pile est pointé par le registre **$sp**.
- Tout les éléments empilés et dépilés de la pile ont 32 bits.

Exemple:

Considérons le tableau suivant, on veut le stocker dans une pile:

a={7,19,37}

Les opération suivantes permettent d'empiler les éléments du tableau dans la pile:

- push a[0]
- push a[1]
- push a[2]

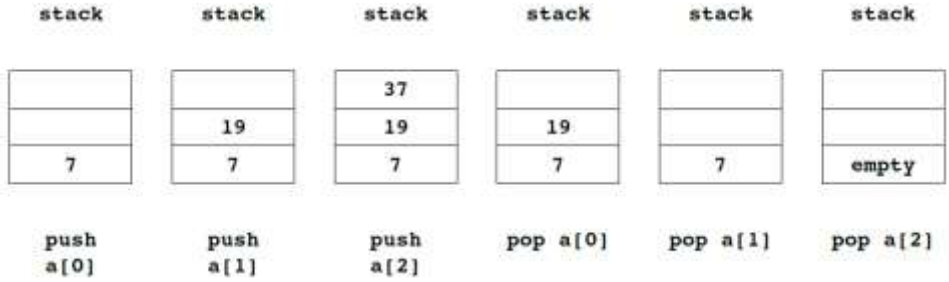
Les opérations suivantes permettent de retirer les éléments de la pile:

- pop a[0]
- pop a[1]
- pop a[2]

Les piles

Exemple:

- Le schéma suivant montre le déroulement des opérations **Push** et **Pop** précédentes.



- Quel est le contenu du tableau après l'exécution?

Les piles

- Push (empiler):
- Empiler ou ajouter un élément à la pile signifie soustraire 4 du registre **$sp** et stocker le résultat dans cette adresse.

- Exemple :
- l'instruction '**Push $t0**' effectue les opérations suivantes:

```
subu $sp, $sp, 4      # Mettre à jour $sp ,
sw $t0, ($sp)         # ranger $t0 au sommet de la pile .
```

Les piles

- Pop (dépiler):
  - Retirer un élément de la pile signifie ajouter 4 au registre **$sp.**
  - Dépiler un élément signifie qu'il est copié à un autre endroit (un registre).

- Exemple :

- L'instruction '**pop $t0**' effectue les opérations suivantes:

```
lw $t0, ($sp)         # copie le sommet dans $t0
addu $sp, $sp, 4      # Mettre à jour $sp ,
```

Les piles

Exemple: Ce programme permet de renverser l'ordre des éléments d'un tableau en utilisant une pile.

```
.data
        array:  .word 1, 3, 5, 7, 9, 11, 13, 15, 17, 19
                .word 21, 23, 25, 27, 29, 31, 33, 35, 37, 39
                .word 41, 43, 45, 47, 49, 51, 53, 55, 57, 59
        length: .word 30
espace: .asciiz " "
rett: .asciiz"\n"
.text


main:

# # Loop to read items from array and push onto stack andplace.

la $t0, array   # array starting address
li $t1, 0       # loop index, i=0
lw $t2, length  # length

pushLoop:
lw $t4, ($t0)   # get array[i]

move $a0,$t4
li $v0,1
syscall
li $v0,4
la $a0,espace
syscall

subu $sp, $sp,4 # push array[i]
sw $t4, ($sp)
add $t1, $t1, 1 # i = i+1
add $t0, $t0, 4 # update array address
blt $t1, $t2, pushLoop # if i<length, continue
```

Les piles

```
# # Loop to pop items from stack and write into array.

la $t0, array # array starting address
li $t1, 0 # loop index, i=0
lw $t2, length # length (redundant line)

li $v0,4
la $a0,rett
syscall

popLoop:
lw $t4, ($sp)


move $a0,$t4
li $v0,1
syscall
li $v0,4
la $a0,espace
syscall

addu $sp, $sp, 4 # pop array[i]
sw $t4, ($t0) # set array[i]
add $t1, $t1, 1 # i = i+1
add $t0, $t0, 4 # update array address
blt $t1, $t2, popLoop # if i<length, continue

# # Done, terminate program.
li $v0, 10 # call code for terminate
syscall # system call

.end main
```