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

Cours 3:
Langage d'assemblage du MIPS R3000
(Partie 1)

Enseignante: Chafika Benkherourou

Introduction:

- Ecrire un programme directement en langage machine (0 et 1) est une tâche très difficile.
- Au lieu d'utiliser le langage machine, on utile le langage d'assemblage.
- Le langage d'assemblage est la représentation symbolique du code binaire des instructions.
- Il est proche du langage machine et lisible par les humains.

Langage machine Vs Langage d'assemblage:

- L'instruction qui additionne le contenu des registres $t3 et $t4 et de stocker le résultat dans $t6 est codée par :

00000001011011000111000000100000

- L'humain trouve une difficulté pour comprendre la signification de cette instruction
- La signification de l'instruction est beaucoup plus claire en langage d'assemblage qu'en langage machine,
- Elle est représentée en langage assembleur comme suit :

add $t6, $t3, $t4

Le langage MIPS:

Un programme en Mips est composé de deux parties:

- Data section : contient la déclaration de données.
- Text section: contient le code du programme.

Format d'un programme MIPS:

```
.data
    # -----

.text
main:
    # -----
    # -----
  li    $v0, 10
  syscall
.end  main
```


Les commentaires:

Les commentaires permettent de donner plus d'explication sur le code

Ceci est un commentaire

; Ceci est un commentaire

Déclaration de données:

- Les données (constantes et variables) doivent être déclarées dans « .Data » section.
- Les données doivent commencer par une lettre suivie des lettres, chiffres ou caractères spéciaux.
- Le format général de la déclaration d'une donnée est:

```
<variableName>:        .<dataType>      <initialValue>
```


Déclaration de données:

- Déclaration des entiers :
  - Une valeur entière décimale est notée 253.
  - Une valeur entière hexadécimale est notée 0xF5A (préfixée par zéro suivi de x).
  - Les entiers sont déclarés par :

    .word, .half, .byte

Exemple :

```
wVar1:      .word       500000
wVar2:      .word       -100000

bVar1:      .byte       5
bVar2:      .byte       -3

hVar1:      .half       5000
hVar2:      .half       -3000
```

Déclaration de données:

- Les chaînes de caractères :

  Les chaînes de caractères sont déclarées par:

  .ascii, .asciiz

Remarque:

.asciiz termine la chaîne de caractères par (0). Le but est de faire savoir au compilateur la fin de la chaîne.

Exemple : la déclaration suivante défini une chaîne de caractères « message » de type asciiz et qui a comme contenu « hello world ».

« \n »: est un retour à la ligne

```
message:    .asciiz     "Hello World\n"
```


Déclaration de données:

- Déclaration des nombres réels :
- Les nombres réels sont déclarés par :

  .float .double

Exemple :

- Les déclarations suivantes sont utilisées pour définir la variable « pi » sur 32 bits par le type .float et l'initialiser à 3,14159.

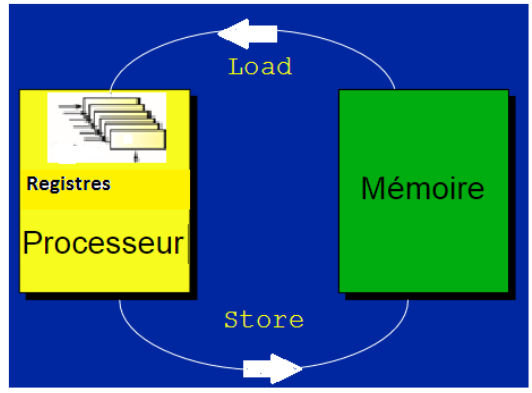
- Et la variable « tao » qui prend le type .double et sera enregistrée sur 64 bits. Elle est initialisée à 6,28318.

```
pi:         .float      3.14159
tao:        .double     6.28318
```

Les instructions du langage MIPS R3000

Les instructions de lecture/écriture

- Ces instructions transfèrent les données entre la mémoire et les registres.
- L'opération de chargement (Load) permet de lire une donnée de la mémoire et la charger dans un registre.
- L'opération de sauvegarde (Store) permet d'écrire le contenu d'un registre dans la mémoire.



Les instructions de lecture/écriture

- La mémoire est accessible uniquement par les instructions « load » et « store ».

-

load word	lw
load byte	lb
load byte unsigned	lbu
load half	lh
load half unsigned	lhu

load immediate	li
load address	la

Les instructions de lecture/écriture

1. Les instruction de chargement: (Load)

- lw - Load Word -

  lw $Rd, mem

- Charge le contenu de l'adresse mémoire dans le registre $Rd.

Exemple:

lw $t0, var1

Les instructions de lecture/écriture

1. Les instruction de chargement: (Load)

- li - Load immediate -

  li $Rd, imm

- Charge la valeur immédiate « imm » (constante) dans le registre $Rd.

Exemple:

li $t0, 27

Après l'exécution $t0=27

Les instructions de lecture/écriture

1. Les instruction de chargement: (Load)

- la - Load address-

la $Rd, adr

- Charge l'adresse de la donnée dans le registre $Rd.

Les instructions de lecture/écriture

2. Les instruction de sauvegarde: (Store)

store word	sw
store half	sh
store byte	sb
move	move

- sw - Store Word -

sw $Rs, mem

- Sauvegarde le contenu du registre source $Rs dans la mémoire.

Exemple:

```
li    $t0, 27
sw    $t0, num                # num = 27
```

Les instructions de lecture/écriture

2. Les instruction de sauvegarde: (Store)

- move $Rd, $Rs
- Copie le contenu du registre $Rs dans le registre $Rd.

Exemple:

```
li    $t0, 42
move  $t1, $t0
```

- Cette instruction donne au registre $t0 la valeur 42 ensuite copie le contenu de $t0 dans $t1.
- $t1 =42

Remarque:

Il n'y a pas d'instruction move permettant de faire copier une donnée d'une cellule mémoire vers une autre.

Les instructions du langage MIPS R3000

Les appels système (syscall)

Les appels système

syscall

- Pour exécuter certaines fonctions système, principalement les entrées/sorties (lire ou écrire un nombre, ou un caractère), il faut utiliser des appels système.
- Pour exécuter un appel système, il faut chercher:
  - L'argument dans le registre $a0 ou $a1.
  - Le numéro de l'appel système est contenu dans le registre $v0
  - syscall

$v0	commande	argument	résultat
1	print_int	$a0 = entier	
4	print_string	$a0 = adresse de chaîne	
5	read_int		$v0 = entier lu
8	read_string	$a0 = adresse de chaîne, $a1 = longueur max	
10	exit		

Les appels système

syscall

- Ecrire un entier: (Print_int)

Il faut mettre l'entier à écrire dans le registre $a0 et exécuter l'appel système numéro 1.

Exemple:

```
li  $a0, 10     # load argument $a0=10
li  $v0, 1      # call code to print integer
syscall         # print $a0
```

Les appels système

syscall

- Lire un entier : (Read_int)

Consiste à exécuter l'appel système numéro 5 et récupérer le résultat dans le registre $v0.

Exemple:

```
li          $v0, 5          # system call code for Read Integer
syscall                     # reads the value into $v0
```

Les appels système
syscall

- Lire une chaîne de caractères : (Read_string)

Il faut:

- Adresse du début de la chaîne et la passer dans $a0
- La longueur maximum et la mettre dans $a1
- Exécuter l'appel système numéro 8.

Le résultat sera mis dans l'espace pointé.

Exemple:

```
li      $v0, 8       # system call code for Read a String
la      $a0, buffer  # load address of  input buffer into $a0
li      $a1, 60      # Length of buffer
syscall
```

Les appels système
syscall

- Ecrire une chaîne de caractères: (Print_string)

Une chaîne de caractères étant identifiée par une adresse de début de la chaîne.

Il faut passer cette adresse dans $a0 et exécuter l'appel système numéro 4 pour l'afficher.

- Exemple:

```
li      $v0, 4       # system call code for Print a String
la      $a0, buffer  # Load address of output buffer into $a0
syscall
```

Les appels système

syscall

- Quitter le programme:

L'appel système numéro 10 effectue la sortie du programme

```
li          $v0, 10          # terminate program run and
syscall                      # return control to  system
```

Les appels système

```
#  Example program to display a string and an integer.
# ------------------------------------------------------------
#  Data Declarations

.data
hdr:        .ascii      "Example\n"
            .asciiz     "The answer is :  "
number:     .word       42

# ------------------------------------------------------------
#  text/code section

.text
.globl      main
main:
      la    $a0, hdr          # addr of NULL terminated string
      li    $v0, 4            # call code for print string
      syscall                 # system call

      li    $v0, 1            # call code for print integer
      lw    $a0, number       # value for integer to print
      syscall                 # system call

# -----
#  Done, terminate program.
      li    $v0, 10           # call code for terminate
      syscall                 # system call
.end main
```