Architecture des Ordinateurs (4)

Evelyne Moreau – Alexandre Chanson

MIPS

- Sortie en 1985
- Mot de 32 bits
- 32 registres à usage général
- 32 registres pour arithmétique IEEE754

Avantage de MIPS

- **Simple**
  - Pas d'instructions compliquées -> RISC
  - Avec MARS exécution pas à pas
  - Gestion de la mémoire « facile »
- **Puissant**
  - Appels système
  - « fonctions »
  - Jump plus intuitif avec les labels
  - « variables »

Registres en MIPS

Position	Nom	Description
0	zero	Contient la valeur 0
1	$at	Réservé
2-3	$v0 - $v1	Valeurs de retour des fonctions
4-7	$a0 - $a3	Arguments pour les fonctions
8-15	$t0 - $t7	Temporaires (registres de travail)
16-23	$s0 - $s7	Préservés par les appels de fonctions (registres de travail)
24-25	$t8 - $t9	Temporaires (registres de travail)
26-27	$k0 - $k1	Réservés
28	$gp	Pointeur vers le segment mémoire de travaile (.data)
29	$sp	Pointeur de la pile
30	$fp	Pointeur de bloc
31	$ra	Adresse de retour

(Quelques) Instructions en MIPS

Instruction	Paramètres	Description
add	$t0, $t1, $t2	$t0 = $t1 **+** $t2
sub	$t0, $t1, $t2	$t0 = $t1 **-** $t2
addi	$t2, $t3, valeur	$t2 = $t3 **+** valeur
and	$t0, $t1, $t2	$t0 = $t1 **AND** $t2
or	$t0, $t1, $t2	$t0 = $t1 **OR** $t2
xor	$t0, $t1, $t2	$t0 = $t1 **XOR** $t2
syscall		Voir
move	$t2,$t3	$t2 = $t3

(Quelques) Instructions en MIPS

Instruction	Paramètres	Description
lw	$t0, label/adresse	Chargement de la valeur dans $t0
sw	$t0, label/adresse	Déchargement de la valeur dans $t0
li	$t0, valeur	Charger valeur dans $t0
j	label	saut au label
jal	label_fonction	Copie la valeur du compteur de programme dans $ra et saute au label_fonction
jr	$ra	saute à la valeur de $ra
beq, bne	$t0, $t1, label	Saute au label si $t0 est égal/différent de $t1
blt, bgt	$t0, $t1, label	Saute au label si $t0 est plus petit/grand que $t1

Appel système en MIPS

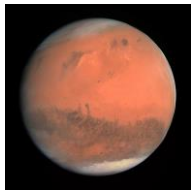
Service rendu par le système	Code du service	Arguments	Retour
Afficher un entier	1	$a0 : stocke l'entier affiché	
Afficher une chaîne de caractères	4	$a0 : adresse de la chaîne affichée	
Lire un entier	5		$v0 : entier lu
Lire une chaîne de caractères	8	$a0 : adresse du tampon $a1 : nb max car. lu (tailel du tampon)	
Fin de programme	10		

1. Ecrire le code de service dans $v0
2. Ecrire les arguments dans les bons registres
3. Appel système : **syscall**

Installation de l'IDE MARS

- **Prérequis: Java version 5 ou plus, Pentium 4, 256 Mo de RAM**
- **Télécharger ici : http://courses.missouristate.edu/KenVollmar/mars/download.htm**
- **Alternative -> Simulateur/IDE en ligne : https://wepsim.github.io/wepsim/ws_dist/wepsim-classic.html?mode=ep&example=11&notify=false**

Lancer L'IDE

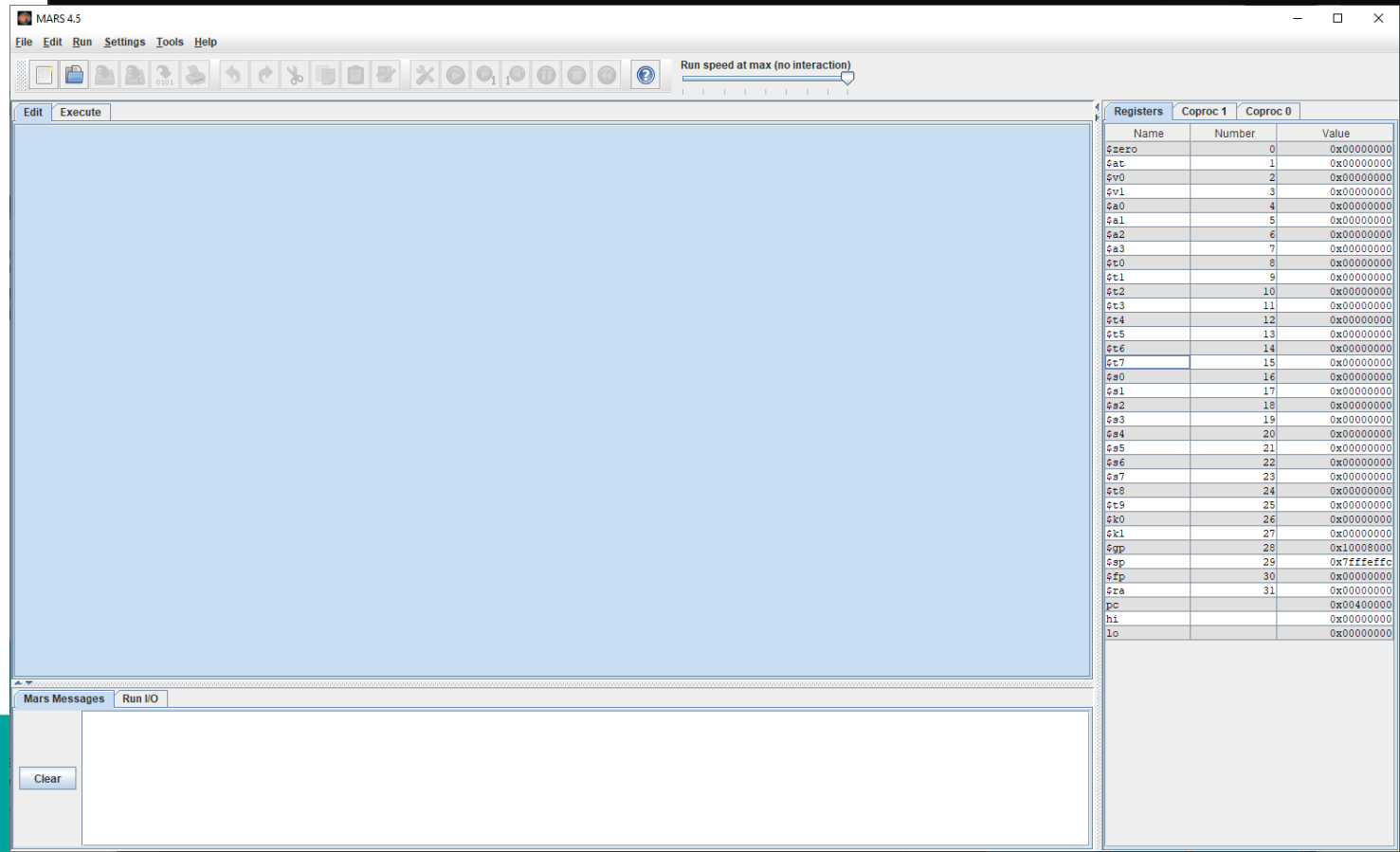


- Lancez par double clic
- Ou en ligne de commande (ex. windows 10)
- Un GUI devrait s'ouvrir

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\achan> java -version
openjdk version "14.0.1" 2020-04-14
OpenJDK Runtime Environment (build 14.0.1+7)
OpenJDK 64-Bit Server VM (build 14.0.1+7, mixed mode, sharing)
PS C:\Users\achan> cd .\Downloads\
PS C:\Users\achan\Downloads> java -jar .\Mars4_5.jar
```

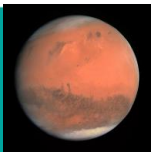


Exemple de programme en MIPS

Addition de deux entiers (exemple du CM précèdent)

addition.asm

```
1    .data
2    a:  .word 9
3    b:  .word 12
4
5    .text
6    lw          $t1, a
7    lw          $t2, b
8    add         $t3, $t1, $t2
```



Structures de Contrôle en assembleur

- **Ce qui affecte l'ordre d'éxécution des instructions**
  - Si … alors … [Sinon …]
  - Tant que … alors …
  - Pour … alors …
  - Appel de fonctions

If *condition* Then …

```
if (n > 8):
    print("OK");
```

- **Si n est plus grand que 8**
  - alors on envoie « OK » sur la sortie standard

If *condition* Then …

```
.data
n:  .word 9
ok:  .asciiz   "OK\n"

.text
lw          $t1, n
bge         $t1, 8, si
j           endif
si:
la          $a0, ok
li          $v0, 4
syscall

endif:
# Suite du code
```

If *condition* Then … Else …

```
if n > 8:
    print("OK");
else:
    n = n + 1;
```

- **Si n est plus grand que 8**
  - alors on envoie « OK » sur la sortie standard
- **Si non**
  - alors n prend la valeur n + 1

If *condition* Then … Else …

```
.data
n: .word 9
ok: .asciiz  "OK\n"

.text
lw      $t1, n
bge     $t1, 8, si
j       sinon
si:
la      $a0, ok
li      $v0, 4
syscall

sinon:
addi    $t1, $t1, 1

# Suite du code
```

While *condition* : …

```
while n < 8:
    n = n + 1;
```

- Tant que n est plus petit que 8
- n prend la valeur n + 1

While *condition* : …

```
.data
n:  .word 4

.text
lw      $t1, n

boucle:
blt     $t1, 8, si
j fin_boucle
si:
addi $t1, $t1, 1
j boucle

fin_boucle:
# Suite du code
```

For i in [0,n] : …

```
for (i = 0; i < 5, i++):
    n = n + 1;
```

- **Pour i allant de 0 à 4**
- **n prend la valeur n + 1**

For i in [0,n] : …

```
.data
n: .word 4

.text
lw      $t1, n
li      $t0, 0

boucle:
blt     $t0, 5, si
j       fin_boucle
si:
addi    $t1, $t1, 1
addi    $t0, $t0, 1
j       boucle

fin_boucle:
# Suite du code
```

Fonctions

```
#main
int a = 7
int b = mul_par_2(a)

# fonction
int mul_par_2(int y):
    return y + y
```

- Objectif primaire --> réutilisation
- On ne peux pas utiliser les mêmes branchements que précédemment

```
.data
a: .word 7

.text
lw      $t1, a

move $a0, $t1 # On prépare les arguments
jal mul_par_2 # jal saute à la bonne adresse + set $ra

li      $v0, 10 # Fin du programme -> syscall mode 10
syscall


# Fonction
mul_par_2:

move $t1, $a0 # Cahrger argument dans un registre de travail
add $v0, $t1, $t1 # placer resultat dans un registre retour

jr $ra # retour à l'appelante
```

Exemple de programme complet

- **TD 1 – Exercice 3**