T. P 01

Matière : Architecture des ordinateurs

Rappel :

Add :
Add Rd, Rs, Rt Addition : "detection d'overflow" Rd ← Rs + Rt

Addu :
Addu Rd, Rs, Rt Addition pas de détection overflow Rd ← Rs + Rt

Addiu :
Addiu Rd, Rs, I Add immediate + pas de détection d'overflow Rd ← Rs + I

Addi :
Addi Rd, Rs, I Addition immediate (I :constante) Rd ← Rs + I

Sll
Sll Rd, Rt, I Shift left logique Rd ← Rt << I(décalé de I positions)

syscall

syscall allows you to call upon the basic system functions. To use syscall, first set $v0 with the code of the function you want to call, then use syscall. The exact codes available may depend on the specific system used, but the following are examples of common sytem calls.

code	call	arguments	results
1	print integer	$a0 = integer to print	
2	print float	$f12 = float to print	
3	print double	$f12 = float to print	
4	print string	$a0 = address of beginning of string	
5	read integer		integer stored in $v0
6	read float		float stored in $f0
7	read double		double stored in $f0
8	read string	$a0 = pointer to buffer, $a1 = length of buffer	string stored in buffer
9	sbrk (allocate memory buffer)	$a0 = size needed	$v0 = address of buffer
10	exit		

Directives Assembleur

These are a subset of the MIPS assembler directives that are used to tell the assembler something about names that are used in a MIPS assembly file:

```
.text    # the succeeding lines contain instructions
.data    # the succeeding lines contain data
.globl name  # name is global symbol (visible to code in other files)
.asciiz  "a string\n"   # stores a null terminated string in memory
```

T. P 01

Matière : Architecture des ordinateurs

Enoncé 1:

1. Écrire le programme suivant par un éditeur de texte et le sauvegarder avec le nom «console.s»,
2. Lancez PCSPIM,
3. Chargez votre programme (File → open ou ctrl + O),
4. Exécutez le programme (Simulator → go ou F5),
5. Commentez le contenu des différents registres.

```
# console.s
.text
.globl __start
__start:
       # Afficher "7" sur la console
        addiu $a0, $zero, 7       # Charger a0 avec l'entier à afficher.
        addiu $v0, $zero, 1       # Charger v0 avec syscall 1 = imprimer entier
        syscall                   # Exécuter le syscall

       # Lire un entier de la console
        addiu $v0, $zero, 5       # Charger v0 avec syscall 5 = Lire un entier
        syscall                   # Lire un entier
        addu $s0, $zero, $v0      # Mettre l'entier dans s0 pour sauvgarde
                                  # avant la sortie.
       # Sortie
        addiu $v0, $zero, 10      # Préparation pour sortir (appel système 10)
        syscall                   # sortie
```

Enoncé 2:

1. même question que énoncé 1.
2. Que fait ce programme ?

```
# simple.s
    .text
    .globl __start
__start:

           addiu $t0, $0, 3
           addiu $t1, $0, 4
           addu  $t2, $t0, $t1
           sll   $t3, $t2, 2
           addiu $v0, $0, 10
           syscall # Exit
```