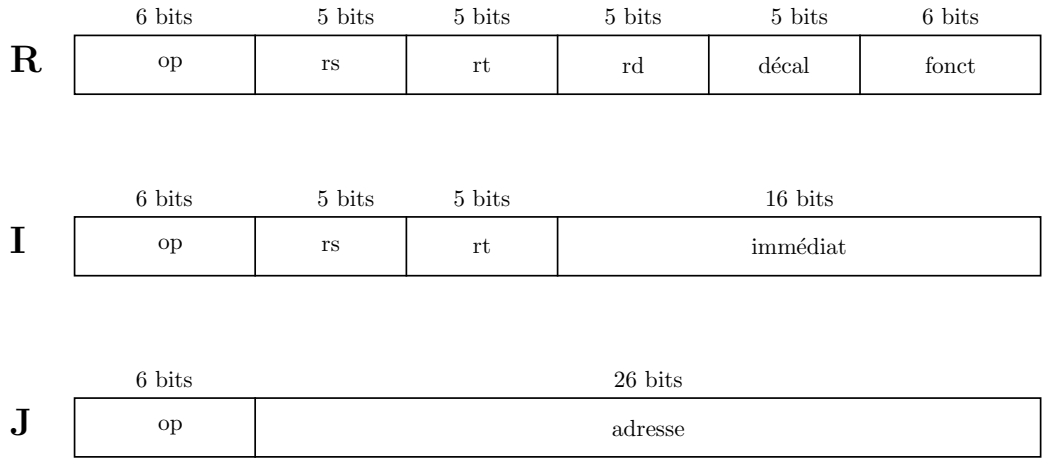
Mémo MIPS

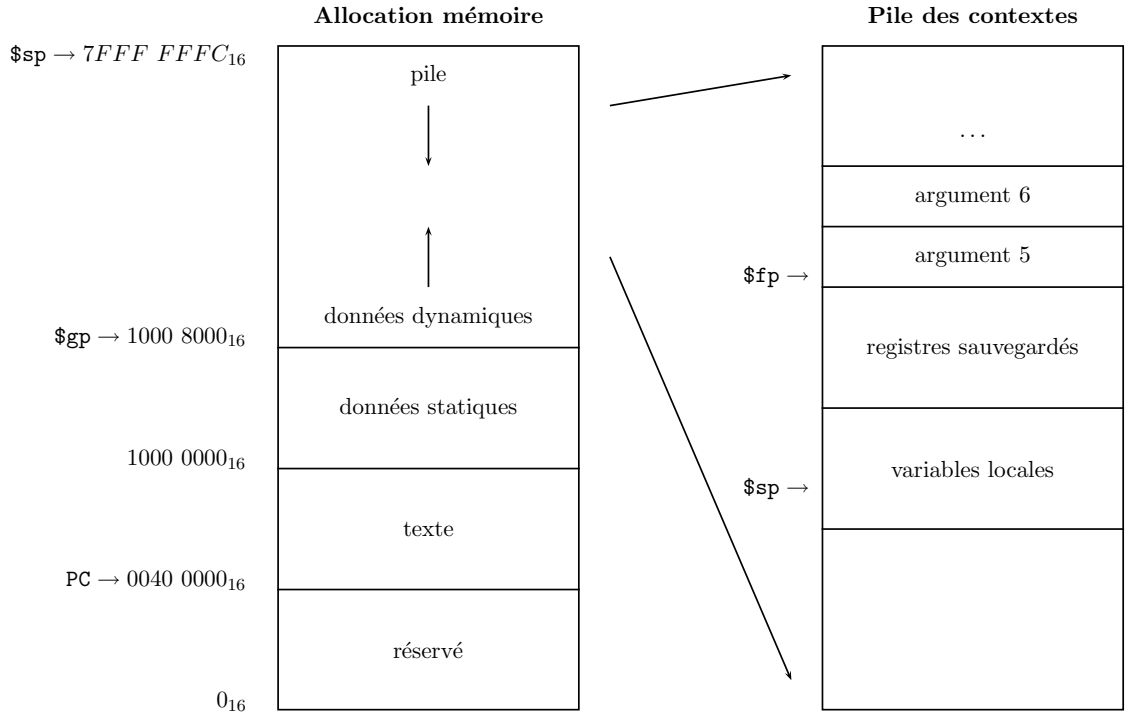
Registres

Nom	Numéro	Utilisation	A sauvegarder
`$zero`	0	constante 0	
`$at`	1	*réservé* à l'assembleur	
`$v0`–`$v1`	2–3	retours de fonctions	non
`$a0`–`$a3`	4–7	passage de paramètres	oui
`$t0`–`$t7`	8–15	registres généraux	non
`$s0`–`$s7`	16–23	registres généraux	oui
`$t8`–`$t9`	24–25	registres généraux	non
`$k0`–`$k1`	26–27	*réservé* au noyau système	
`$gp`	28	pointeur global	oui
`$sp`	29	pointeur de pile	oui
`$fp`	30	pointeur de frame	oui
`$ra`	31	adresse de retour de fonction	oui

Formats d'instruction



Gestion de la mémoire



Instructions

Nom	Exemple	Format	Description
add	`add $s3, $s2, $s1`	R	`s3=s2+s1 ;`
addi	`add $s3, $s2, 100`	I	`s3=s2+100 ;`
and	`and $s3, $s2, $s1`	R	`s3=s2&&s1 ;`
beq	`beq $t0, $t1, Etiquette`	I	`if (t0=t1) goto Etiquette ;`
bne	`bne $t0, $t1, Etiquette`	I	`if (t0 !=t1) goto Etiquette ;`
j	`j Etiquette`	J	`goto Etiquette ;`
jal	`jal Etiquette`	J	`ra=PC+4 ; goto Etiquette ;`
jr	`jr $ra`	J	Saut à l'adresse contenue dans le registre `ra`, cette adresse étant alignée sur une frontière de mot
la	`la $t0, Etiquette`	pseudo-instr.	`lui $t0, adresse Etiquette>>16` `ori $t0, $t0, adresse Etiquette & 0xFFFF`
lui	`lui $t0, 0xFF14`	I	`t0=0xFF14<<16 ;`
lw	`lw $s0, 32($sp)`	I	`s0=Mem[sp+32] ;`
mul	`mul $s3, $s2, $s1`	R	`s3=s2*s1 ;`
nor	`nor $s3, $s2, $s1`	R	`s3=~(s2|s1) ;`
or	`or $s3, $s2, $s1`	R	`s3=s2|s1 ;`
ori	`ori $s3, $s2, 0xFFFF`	I	`s3=s2|0xFFFF ;`
sll	`sll $s2, $s1, 4`	I	`s2=s1<<4 ;`
srl	`srl $s2, $s1, 4`	I	`s2=s1>>4 ;`
slt	`slt $t0, $s1, $s0`	R	`if (s1<s0) t0=1 ; else t0=0 ;`
slti	`slt $t0, $s1, 2`	I	`if (s1<2) t0=1 ; else t0=0 ;`
sub	`sub $s3, $s2, $s1`	R	`s3=s2-s1 ;`
sw	`sw $s0, 32($sp)`	I	`Mem[sp+32]=s0 ;`
xor	`xor $s3, $s2, $s1`	R	`s3=s2||s1 ;`

Conventions d'appel de fonctions sur MIPS32

- La pile croît des adresses hautes vers les adresses basses : on soustrait à `$sp` pour allouer de l'espace dans la pile, on ajoute à `$sp` pour rendre de l'espace dans la pile ;
- les déplacement dans la pile se font sur des mots mémoire entiers (multiples de quatre octets) ;
- passage de paramètres : tout paramètre plus petit que 32 bits est automatiquement promu sur 32 bits ;
- les quatre premiers paramètres sont passés par les registres `$a0` à `$a3` ; tout paramètre supplémentaire est passé par la pile ;
- valeur de retour : toute valeur de format inférieur ou égal à 32 bits est retournée par le registre `$v0` (sur 64 bits `$v1` est utilisé avec `$v0`).

Directives d'assemblage

Dit à l'assembleur comment interpéter ce qui suit en mémoire.

`.ascii <chaîne>` : ce qui suit la directive est une chaîne de caractères ;

`.asciiz <chaîne>` : ce qui suit la directive est une chaîne terminée par le caractère `\0`

`.byte <b1, ..., bn>` : range `b1`, ..., `bn` dans n octets successifs ;

`.word <w1, ..., wn>` : range `w1`, ..., `wn` dans n mots successifs ;

`.data <@>` : ce qui suit la directive est placé dans le segment de données ; si l'argument optionnel `@` est présent, les éléments qui suivent sont rangés consécutivement à partir de l'adresse `@` ;

`.text <@>` : ce qui suit la directive est placé dans le segment Texte ; si l'argument optionnel `@` est présent, les éléments qui suivent sont rangés consécutivement à partir de l'adresse `@` ;

`.glob <étiquette>` : ce qui suit la directive est une `étiquette` globale et peut être référencée à partir d'autres fichiers.

Exemple

```
            .data
tableau :   .word           0x00000001,
                            0x00000002,
                            0x00000003,
                            0x00000004,
                            0x00000005
            .text
main :      .glob main
            la $t0, tableau
            ...
```

Appels système : `syscall`

Toutes les entrées–sorties sont prises en charge par la routine système `syscall`

Opération	`$v0 =`	Arguments	Valeur retour
`print_int`	1	`$a0` = entier à afficher	
`print_string`	4	`$a0` = chaîne à afficher	
`read_int`	5		`$v0` = entier lu
`read_string`	8	`$a0` = buffer, `$a0` = longueur	
`exit`	10		
`print_char`	11	`$a0` = car. à afficher	`$a0` = car. lu
`read_char`	12		`$a0` = car. lu
`exit2`	17	`$a0` = résultat	

Exemple

```
        .data
str :   .asciiz             ‘‘Bonjour’’
        .text
main :  ori $v0, $zero, 4   # $v0 <- 4
        la $a0, str         # $a0 <- str
        syscall             # affiche ‘‘Bonjour’’
        ...
        ori $v0, $zero, 10  # $v0 <- 10
        syscall             # appel système de
                            # terminaison de programme
```