Support TP2 AO (Suite)

Les registres de MIPS :

La liste des **32** registres programmables de **32 bits** sur **MIPS**, est comme suit :

Nom	Numéro	Description
$zero	$0	constante zéro
$at	$1	réserve pour l'assembleur
$v0	$2	retour de fonction
$v1	$3	retour de fonction
$a0	$4	argument de fonction
$a1	$5	argument de fonction
$a2	$6	argument de fonction
$a3	$7	argument de fonction
$t0	$8	temporaire
$t1	$9	temporaire
$t2	$10	temporaire
$t3	$11	temporaire
$t4	$12	temporaire
$t5	$13	temporaire
$t6	$14	temporaire
$t7	$15	temporaire

Nom	Numéro	Description
$s0	$16	sauvegardé
$s1	$17	sauvegardé
$s2	$18	sauvegardé
$s3	$19	sauvegardé
$s4	$20	sauvegardé
$s5	$21	sauvegardé
$s6	$22	sauvegardé
$s7	$23	sauvegardé
$t8	$24	temporaire
$t9	$25	temporaire
$k0	$26	pour noyau système
$k1	$27	pour noyau système
$gp	$28	pointeur global
$sp	$29	pointeur de pile
$fp	$30	pointeur de frame
$ra	$31	registre d'adresse

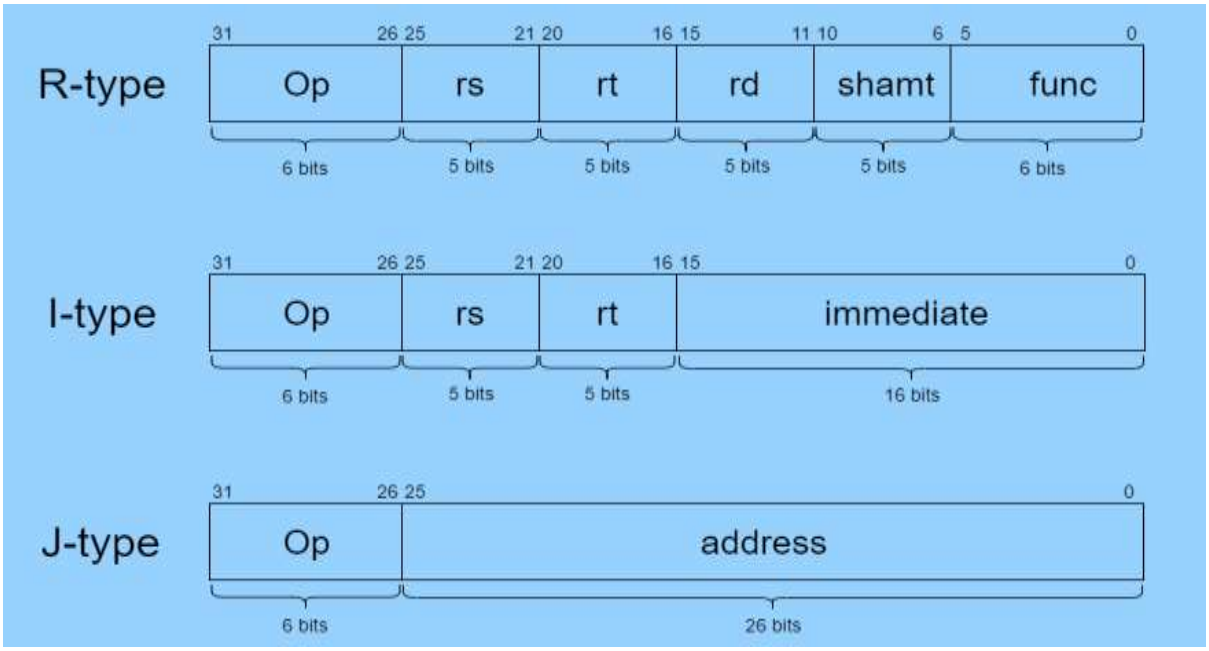
Remarque 1 : Un registre peut être désigné par son nom ou son numéro.

Remarque 2 : Un registre doit toujours être précédé par un **$** lors de son utilisation dans une instruction assembleur.

Remarque 3 : Il existe aussi **2** autres registres qui ne sont pas listés ici, c'est ***lo*** (pour ***low***) et ***hi*** (pour ***high***). Ils sont, exclusivement, utilisés par les instructions de multiplication et division.
Dans le langage assembleur MIPS, les registres généraux utilisés pour contenir les données sont les registres temporaires et les registres sauvegardés, la différence entre les deux est que les registres temporaires ne doivent pas être sauvegardés par le programmeur lors d'un appel de fonction. Les registres ***a***, ***v*** et ***ra*** sont utilisés pour le mécanisme des appels de fonction, les registres ***a*** doivent contenir les arguments, les registres ***v*** la valeur de retour, et ***ra*** l'adresse de retour.

Les formats d'encodage des instructions :

3 formats d'encodage sont possibles dans MIPS :



Op : ***Op****code*, le code de l'instruction.
rs : registre **s**ource.
rt : deuxième **r**egistre **s**ource.
rd : registre destination.
shamt : ***sh****ift* ***amount***, nombre de bits à décaler dans l'instruction shift.
func : ***func****tion*, le code de la fonction/opération.
address : adresse mémoire sur **26 bits**.

Chaque instruction MIPS est contenue sur **32 bits**, l'encodage des **3** types d'instruction possibles est décrit plus haut.

1. **Les instructions de type R (R** pour **R**egistre**)** utilisent **3** registres pour les opérations arithmétiques et logiques, **2** registres sources comme opérandes et un registre destination pour le résultat, **5 bits** sont utilisés pour coder le numéro du registre utilisé.
2. **Les instructions de type** I (**I** pour **i**mmédiate) utilisent aussi **2** opérandes et un registre pour le résultat, la différence avec le type **R** est que l'un de ces opérandes est une immédiate sur **16 bits**, ils sont utilisés pour différents types d'instructions incluant aussi des opérations arithmétiques et logiques.
3. **Les instructions de type J (J** pour **J**ump, ou saut**)** sont réservées pour le saut.

Remarque : On peut observer sur les **3** formats, que seulement **3** manières d'utiliser l'information dans une instruction MIPS sont possibles, soit l'information est dans un registre de **32 bits**, soit une valeur immédiate de **16** bits, soit une adresse de **26 bits** (Les **5 bits** de ***shampt*** sont particuliers aux instructions de **shift**).

Les instructions arithmétiques et logiques :

Instruction	Type	Syntaxe	Description
Addition (Addition)	R	**add** rd,rs,rt	rd ← rs + rt
	I	addi rt,rs,imm	rt ← rs + imm
Substraction (Soustraction)	R	**sub** rd,rs,rt	rd ← rs - rs
	I	subi rt,rs,imm	rt ← rs - imm
And (ET)	R	**and** rd,rs,rt	rd ← rs & rs
	I	andi rt,rs,imm	rt ← rs & imm
Or (OU)	R	**or** rd,rs,rt	rd ← rs | rs
	I	ori rt,rs,imm	rt ← rs | imm

Exemples :

```
add   $s0, $s2, $s3
subi $t0, $t5, 3
and   $s3, $t5, $t1
```

Remarque 1 : Les opérations logiques sont des opérations binaires (**bitwise** : bit-par-bit).

Remarque 2 : Les autres opérations logiques comme le ***not***, ***nand*** ou ***xnor*** sont réalisées en composition à partir des instructions présentes dans le jeu d'instruction.

Les instructions de multiplication et division

Les instructions concernant la division et la soustraction sont des instructions particulières dans le sens où leur résultat nécessite **64 bits** de mémoire, la **multiplication** de **2** nombres de **32 bits** exige **64 bits** d'espace pour le résultat, et **la division entière** exige aussi **32 bits** pour le **quotient** (résultat de division entière), et **32 bits** pour le **reste de division**. Pour cela ces **2** opérations utilisent les registres **32 bits lo** (pour **lo**w) et **hi** (pour **hi**gh) qui sont exclusivement utilisés par les instructions de **multiplication** et de **division**.

Instruction	Type	Syntaxe	Description
Divide (Division)	R	**DIV** rs,rt	HI=rs%rt (Modulo); LO=rs/rt (Quotient)
Divide **U**nsigned	R	**DIVU** rs,rt	HI=rs%rt (Modulo); LO=rs/rt (Quotient)
Move **F**rom **HI**	R	**MFHI** rd	rd=HI
Move **F**rom **LO**	R	**MFLO** rd	rd=LO
Move **T**o **HI**	R	**MTHI** rs	HI=rs
Move **T**o **LO**	R	**MTLO** rs	LO=rs
Multiply (Multiplication)	R	**MUL** rd,rs,rt	HI,LO=rs*rt ; rd=LO
Multiply	R	**MULT** rs,rt	HI,LO=rs*rt
Multiply **U**nsigned	R	**MULTU** rs,rt	HI,LO=rs*rt

Remarque 1 : Les **2 points** dans la description (**hi:lo**) indiquent une concaténation, ça veut dire que les **2** registres de **32 bits hi** et **lo** sont mis côte à côte pour construire un registre de **64 bits**.

Remarque 2 : Les **2** instructions **mfhi** (**M**ove **f**rom **hi**) et **mflo** (**M**ove **f**rom **lo**) sont des instructions qui permettent de copier les valeurs de **hi** et **lo** dans l'un des **32** registres programmables. Il faut rappeler que **hi** et **lo** ne sont pas des registres programmables donc ils ne peuvent pas être utilisés directement dans les instructions MIPS.

Remarque 3 : L'instruction **mul** est identique à l'instruction **mult** mais elle fait en plus copier la valeur de **lo** dans **rd**, ça favorise son utilisation lorsque la multiplication est sur des petites grandeurs.

Les instructions de transfert de données :

Les instructions de transfert de données sont très importantes. Elles permettent de faire le transfert de l'information d'un emplacement vers un autre. Par exemple :

- le transfert de la mémoire vers un registre est fait par l'instruction **lw** (**l**oad **w**ord)
- et l'inverse, d'un registre vers la mémoire est **sw** (**s**tore **w**ord),
- **move** est pour un transfert d'un registre vers un autre,
- **li** le transfert d'une immédiate vers un registre,
- et **la** d'une adresse vers un registre.

Instruction	Type	Syntaxe	Description
move	P	move rd,rs	rd ← rs
li	P	li rd,imm	rd ← imm
la	P	la rd,label	rd ← label
lw	I	lw rt,imm(rs)	rt ← MEM[rs+imm]
sw	I	sw rt,imm(rs)	MEM[rs+imm] ← rt

Remarque 1 : Le type **P** (pour **P**seudo-instruction) n'est pas un type proprement dit. Réellement, les instructions de ce type sont créées par l'assembleur à partir d'autres instructions équivalentes.
Exemple : move rd,rs est réellement add rd,rs,$zero.

Remarque 2 : Les **2** instructions **lw** et **sw** forment leurs adresses mémoires d'une manière analogue à la notion **segment/offset**, dans le sens où l'adresse est formée par l'ajout d'un décalage **imm** à une adresse de départ **rs**.

Remarque 3 : L'instruction ***load*** possède d'autres variantes en plus du **lw**, il y'a **lb** pour *load* ***b****yte* qui fait charger une valeur de la mémoire sur **1 octet** (**8bits**), et **lh** pour *load* ***h****alf* qui fait charger une valeur sur **16 bits** (**demi-mot**).

Les instructions de comparaison :

Instruction	Type	Syntaxe	Description
slt	R	slt rd,rs,rt	**If**(rs < rt) rd ← 1 **else** rd ← 0
	I	slti rt,rs,imm	**If**(rs < imm) rt ← 1 **else** rt ← 0
sle	P	sle rd,rs,rt	**If**(rs <= rt) rd ← 1 **else** rd ← 0
	P	slei rt,rs,imm	**If**(rs <= imm) rt ← 1 **else** rt ← 0
seq	P	seq rd,rs,rt	**If**(rs == rt) rd ← 1 **else** rd ← 0
	P	seqi rt,rs,imm	**If**(rs == imm) rt ← 1 **else** rt ← 0
sne	P	sne rd,rs,rt	**If**(rs != rt) rd ← 1 **else** rd ← 0
	P	snei rt,rs,imm	**If**(rs != imm) rt ← 1 **else** rt ← 0
sgt	P	sgt rd,rs,rt	**If**(rs > rt) rd ← 1 **else** rd ← 0
	P	sgti rt,rs,imm	**If**(rs > imm) rt ← 1 **else** rt ← 0
sge	P	sge rd,rs,rt	**If**(rs >= rt) rd ← 1 **else** rd ← 0
	P	sgei rt,rs,imm	**If**(rs >= imm) rt ← 1 **else** rt ← 0

Remarque : À l'exception de **slt** et **slti**, les autres instructions de comparaison ne sont pas réelles. Ce sont des pseudo-instructions fournies par l'assembleur pour faciliter la programmation.

Les instructions de saut :

Les instructions de saut (**jump**) et de branchement (**branch**) permettent de faire des sauts dans l'exécution d'un programme. Ça permet réellement d'implémenter au niveau assembleur les instructions de tests comme le **if**, **if-else**, **switch** et les instructions de boucle comme **for**, **while**, **do-while**.

Exemple : Le code MIPS correspondant au fragment du programme suivant ;

```
t2 := 0;
while t1 > 0 do begin t2 := t2 + t1; t1 := t1 - 1 end
```

est le suivant :

```
  li $t2, 0           # t2 := 0
  while :             # début de la boucle
  blez $t1, done
  add $t2, $t2, $t1   # t2 := t2 + t1
  sub $t1, $t1, 1     # t1 := t1 - 1
  j while             # retour vers l 'étiquette while
  done:               # suite du programme
```

Instruction	Type	Syntaxe	Description
Jump	J	**j** address	PC ← address
	R	jr rs	PC ← rs
Jump **a**nd **l**ink	J	**jal** address	$ra ← PC+4 ; PC ← address
	R	**jal**r rs	$ra ← PC+4 ; PC ← rs

Remarque 1 : Les adresses dans l'assembleur sont implémentées par le mécanisme des labels.

Remarque 2 : Les instructions ***jump*** *and* ***link*** sont utilisées lors d'appel de fonction. En plus d'un saut, ces instructions sauvegardent aussi le **PC+4** dans le registre **$ra**, faisant office d'adresse de retour lors de la fin de la fonction. **PC+4** est l'adresse suivante de l'adresse actuelle sachant qu'une adresse est représentée sur **32 bits** (**4 octet**).

Les instructions de branchement :

Ce sont des instructions semblables aux instructions de saut, la principale différence est qu'ils doivent satisfaire une condition pour que le saut soit effectué.

Instruction	Type	Syntaxe	Description
beq (**b**ranch on **eq**ual)	I	beq rs,rt,imm	**If**(rs == rt) PC ← imm **else** PC ← PC+4
bne (**b**ranch on **n**ot **e**qual)	I	bne rs,rt,imm	**If**(rs != rt) PC ← imm **else** PC ← PC+4
blt (**b**ranch on **l**ess **t**han)	P	blt rs,rt,imm	**If**(rs < rt) PC ← imm **else** PC ← PC+4
ble (**b**ranch on **l**ess or **eq**ual)	P	ble rs,rt,imm	**If**(rs <= rt) PC ← imm **else** PC ← PC+4
bgt (**b**ranch on **g**reat **t**han)	P	bgt rs,rt,imm	**If**(rs > rt) PC ← imm **else** PC ← PC+4
bge (**b**ranch on **g**reat or **eq**ual)	P	bge rs,rt,imm	**If**(rs >= rt) PC ← imm **else** PC ← PC+4

Remarque 1 : L'adresse de branchement ici est implémentée dans l'immédiate, c'est le même mécanisme de labels que va utiliser l'assembleur pour réaliser cette implémentation.

Remarque 2 : Il existe des variantes pour ces instructions avec un **Z** comme suffixe indiquant que la comparaison se fait par rapport à **z**éro.
Exemple : beq**z** rs,imm est une instruction qui applique la comparaison entre ***rs*** et **zéro** comme condition de saut.

Remarque 3 : Sur toutes les instructions déjà faites, il existe des instructions avec un suffixe **U** (pour **u**nsigned). C'est pour dire que l'instruction fait l'opération sur des valeurs avec la représentation à valeur **absolue** (non signé).
Exemple : add**u** rd,rs,rt.

Remarque 4 : Il faut savoir que dans MIPS on a globalement **3** types de données à manipuler, les **immédiates**, les **registres** et les **adresses** (les labels).

Les Fonction dans MIPS :

L'implémentation des fonctions dans MIPS implique l'utilisation de **7 registres spécifiques**, **$a0**, **$a1**, **$a2**, **$a3** pour le passage d'arguments, **$v0**, **$v1** pour le retour de fonction, et **$ra** pour l'adresse de retour de la fonction. Les étapes d'exécution de l'appel d'une fonction sont :

- remplir les registres d'argument **$a0**, **$a1**, **$a2**, **$a3** avec les paramètres à faire passer à la fonction.
- appel de la fonction avec l'instruction **jal**. Ça permet de sauter au début de la fonction, et de sauvegarder l'adresse de retour.
- la fonction va toujours commencer par récupérer les paramètres des registres **$a0**, **$a1**, **$a2**, **$a3**.
- à la fin d'exécution de la fonction, elle doit remplir la valeur de retour dans **$v0**, **$v1**.
- L'instruction de retour est **jr $ra**, (**$ra** contient l'adresse l'instruction juste après **jal**).
- la valeur retournée par la fonction est dans **$v0**, **$v1**.

Remarque : L'instruction **jal** (**jump and link**) est l'instruction d'appel d'une fonction. Elle fait un saut vers le label de l'adresse de la première instruction de la fonction, et le **link** est l'opération de sauvegarder l'adresse de l'instruction juste après l'instruction **jal (PC+4)** dans **$ra**.