

Instructions MIPS

	Assembleur		Opération		Effet	Format
arithmétique et logique	Add	Rd, Rs, Rt	Add	<i>Overflow detection</i>	$Rd \leftarrow Rs + Rt$	R
	Sub	Rd, Rs, Rt	Subtract	<i>Overflow detection</i>	$Rd \leftarrow Rs - Rt$	R
	Addu	Rd, Rs, Rt	Add	<i>No Overflow</i>	$Rd \leftarrow Rs + Rt$	R
	Subu	Rd, Rs, Rt	Subtract	<i>No Overflow</i>	$Rd \leftarrow Rs - Rt$	R
	Addi	Rt, Rs, I	Add Immediate	<i>Overflow detection</i>	$Rt \leftarrow Rs + I$	I
	Addiu	Rt, Rs, I	Add Immediate	<i>No Overflow</i>	$Rt \leftarrow Rs + I$	I
	Or	Rd, Rs, Rt	Logical Or		$Rd \leftarrow Rs \text{ or } Rt$	R
	And	Rd, Rs, Rt	Logical And		$Rd \leftarrow Rs \text{ and } Rt$	R
	Xor	Rd, Rs, Rt	Logical Exclusive-Or		$Rd \leftarrow Rs \text{ xor } Rt$	R
	Nor	Rd, Rs, Rt	Logical Not Or		$Rd \leftarrow Rs \text{ nor } Rt$	R
	Ori	Rt, Rs, I	Or Immediate	<i>Unsigned immediate</i>	$Rt \leftarrow Rs \text{ or } I$	I
	Andi	Rt, Rs, I	And Immediate	<i>Unsigned immediate</i>	$Rt \leftarrow Rs \text{ and } I$	I
	Xori	Rt, Rs, I	Exclusive-Or Immediate	<i>Unsigned immediate</i>	$Rt \leftarrow Rs \text{ xor } I$	I
	Sllv	Rd, Rt, Rs	Shitf Left Logical Variable	<i>5 lsb of Rs is significant</i>	$Rd \leftarrow Rt \ll Rs$	R
	Srlv	Rd, Rt, Rs	Shitf Right Logical Variable	<i>5 lsb of Rs is significant</i>	$Rd \leftarrow Rt \gg Rs$	R
	Srav	Rd, Rt, Rs	Shitf Right Arithmetical Variable	<i>5 lsb of Rs is significant *with sign extension</i>	$Rd \leftarrow Rt \gg *Rs$	R
	Sll	Rd, Rt, sh	Shitf Left Logical		$Rd \leftarrow Rt \ll sh$	R
	Srl	Rd, Rt, sh	Shitf Right Logical		$Rd \leftarrow Rt \gg sh$	R
	Sra	Rd, Rt, sh	Shitf Right Arithmetical	<i>*with sign extension</i>	$Rd \leftarrow Rt \gg *sh$	R
	Lui	Rt, I	Load Upper Immediate	<i>16 lowers bits of Rt are set to zero</i>	$Rt \leftarrow I \parallel "0000"$	I
	Slt	Rd, Rs, Rt	Set if Less Than		$Rd \leftarrow 1 \text{ if } Rs < Rt \text{ else } 0$	R
	Sltu	Rd, Rs, Rt	Set if Less Than <i>Unsigned</i>		$Rd \leftarrow 1 \text{ if } Rs < Rt \text{ else } 0$	R
	Slti	Rt, Rs, I	Set if Less Than <i>Immediate</i>	<i>Sign extended Immediate</i>	$Rt \leftarrow 1 \text{ if } Rs < I \text{ else } 0$	I
	Sltiu	Rt, Rs, I	Set if Less Than <i>Immediate</i>	<i>Sign extended Immediate</i>	$Rt \leftarrow 1 \text{ if } Rs < I \text{ else } 0$	I
	Mult	Rs, Rt	Multiply	<i>LO&lt;-32 low significant bits HI&lt;-32 high significant bits</i>	$Rs * Rt$	R
	Multu	Rs, Rt	Multiply Unsigned	<i>LO&lt;-32 low significant bits HI&lt;-32 high significant bits</i>	$Rs * Rt$	R
	Div	Rs, Rt	Divide	<i>LO&lt;-Quotient HI&lt;-Remainder</i>	$Rs / Rt$	R
	Divu	Rs, Rt	Divide Unsigned	<i>LO&lt;-Quotient HI&lt;-Remainder</i>	$Rs / Rt$	R
HI/LO	Mfhi	Rd	Move From HI		$Rd \leftarrow HI$	R
	Mflo	Rd	Move From LO		$Rd \leftarrow LO$	R
	Mthi	Rs	Move To HI		$HI \leftarrow Rs$	R
	Mtlo	Rs	Move To LO		$LO \leftarrow Rs$	R
SYS	Syscall		System call			R

Lecture et écriture	Lw	Rt, I(Rs)	Load Word	<i>Sign extended immediate</i>	$Rt \leftarrow M(Rs + I)$	I
	Sw	Rt, I(Rs)	Store Word	<i>Sign extended immediate</i>	$M(Rs + I) \leftarrow Rt$	I
	Lh	Rt, I(Rs)	Load Half Word	<i>Sign extended immediate. Two bytes from storage are located into the 2 less significant bytes of Rt. The sign of these 2 bytes is extended on the 2 most significant bytes.</i>	$Rt \leftarrow M(Rs + I)$	I
	Lhu	Rt, I(Rs)	Load Half Word Unsigned	<i>Sign extended immediate. Two bytes from storage are located into the 2 less significant bytes of Rt, others bytes are set to zero.</i>	$Rt \leftarrow M(Rs + I)$	I
	Sh	Rt, I(Rs)	Store Half Word	<i>Sign extended immediate/. The two less significant bytes of Rt are stored into the storage.</i>	$M(Rs + I) \leftarrow Rt$	I
	Lb	Rt, I(Rs)	Load Byte	<i>Sign extended immediate. One byte from storage is located into the less significant bytes of Rt. The sign of this byte is extended on the 3 most significant bytes.</i>	$Rt \leftarrow M(Rs + I)$	I
	Lbu	Rt, I(Rs)	Load Byte Unsigned	<i>Sign extended immediate. One byte from storage is located into the less significant bytes of Rt, others bytes are set to zero.</i>	$Rt \leftarrow M(Rs + I)$	I
	Sb	Rt, I(Rs)	Store Byte	<i>Sign extended immediate. The less significant byte of Rt is stored into the storage.</i>	$M(Rs + I) \leftarrow Rt$	I
	Beq	Rs, Rt, label	Branch if Equal		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs = Rt$ $PC \leftarrow PC + 4 \text{ if } Rs \neq Rt$	I
	Bne	Rs, Rt, label	Branch if Not Equal		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs \neq Rt$ $PC \leftarrow PC + 4 \text{ if } Rs = Rt$	I
Saut et branchement	Bgez	Rs, label	Branch if Greater or Equal Zero		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs \geq 0$ $PC \leftarrow PC + 4 \text{ if } Rs < 0$	I
	Bgtz	Rs, label	Branch if Greater Than Zero		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs > 0$ $PC \leftarrow PC + 4 \text{ if } Rs \leq 0$	I
	Blez	Rs, label	Branch if Less or Equal Zero		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs \leq 0$ $PC \leftarrow PC + 4 \text{ if } Rs > 0$	I
	Bltz	Rs, label	Branch if Less Than Zero		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs < 0$ $PC \leftarrow PC + 4 \text{ if } Rs \geq 0$	I
	Bgezal	Rs, label	Branch if Greater or Equal Zero And Link		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs \geq 0$ $PC \leftarrow PC + 4 \text{ if } Rs < 0$ $R31 \leftarrow PC + 4 \text{ in both cases}$	I
	Bltzal	Rs, label	Branch if Greater Than Zero And Link		$PC \leftarrow PC + 4 + (I * 4) \text{ if } Rs < 0$ $PC \leftarrow PC + 4 \text{ if } Rs \geq 0$ $R31 \leftarrow PC + 4 \text{ in both cases}$	I
	J	Label	Jump		$PC \leftarrow PC[31:28] \parallel I * 4$	J
	Jal	Label	Jump and Link		$R31 \leftarrow PC + 4$ $PC \leftarrow PC[31:28] \parallel I * 4$	J
	Jr	Rs	Jump Register		$PC \leftarrow Rs$	R
	Jalr	Rs	Jump and Link Register		$R31 \leftarrow PC + 4$ $PC \leftarrow Rs$	R
	Jalr	Rd, Rs	Jump and Link Register		$Rd \leftarrow PC + 4$ $PC \leftarrow Rs$	R

## Format de codage des instructions

31	25	20	15	10	5	0	
OPCODE	RS	RT	RD	SH	FUNC		Format R
OPCODE	RS	RT	IMD16				Format I
OPCODE	IMD26						Format J

## Codage des codes opération des instructions

		DECODAGE OPCODE							
		INS 28:26							
		000	001	010	011	100	101	110	111
INS 31 : 29	000	SPECIAL	BCOND	J	JAL	BEQ	BNE	BLEZ	BGTZ
	001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
	010	COPRO							
	011								
	100	LB	LH		LW	LBU	LHU		
	101	SB	SH		SW				
	110								
	111								

		OPCODE = SPECIAL							
		INS 2:0							
		000	001	010	011	100	101	110	111
INS 5 : 3	000	SLL		SRL	SRA	SLLV		SRLV	SRAV
	001	JR	JALR			SYSCALL	BREAK		
	010	MFHI	MTHI	MFLO	MTLO				
	011	MULT	MULTU	DIV	DIVU				
	100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
	101			SLT	SLTU				
	110								
	111								

## Appels système dans Mars

Avant de réaliser un appel système (avec syscall), il faut placer dans le registre \$2 le numéro de l'appel système demandé. Il faut aussi donner les paramètres de l'appel quand il y en a. Le passage de ces paramètres se fait par les registres, les registres \$4 et/ou \$5 sont utilisés. La valeur de retour (s'il y en a une) se trouve après l'appel dans le registre \$2.

**Écrire (afficher) un entier en décimal sur la console :**

- Appel système numéro 1.
- 1 paramètre : l'entier à écrire sur la console qui doit être placé dans le registre \$4.

**Lire un entier sur la console :**

- Appel système numéro 5.
- Valeur de retour (dans \$2 après l'appel) : l'entier lu.

**Écrire (afficher) un caractère sur la console :**

- Appel système numéro 11.
- 1 paramètre : le caractère (code ASCII) à écrire sur la console qui doit être placé dans le registre \$4.

**Lire un caractère sur la console :**

- Appel système numéro 12.
- Valeur de retour (dans \$2 après l'appel) : le caractère lu.

**Écrire une chaîne de caractères sur la console :**

- Appel système numéro 4.
- 1 paramètre : l'adresse de la chaîne de caractères à écrire doit être placée dans le registre \$4. La chaîne de caractère doit se terminer par le caractère de fin de chaîne (de valeur 0).

**Lire une chaîne de caractères sur la console :**

- Appel système numéro 8.
- 2 paramètres :
  - l'adresse mémoire à partir de laquelle la chaîne de caractères lue sera sauvegardée doit être placée dans le registre \$4.
  - la taille maximale de la chaîne de caractères attendue (caractère de fin de chaîne inclus) doit être placée dans le registre \$5 (en octet).

**Attention**, avec le simulateur MARS, la chaîne de caractères se termine par '\n' puis par '\0' sauf si la chaîne est exactement de la taille maximale caractère de fin de chaîne inclus.

**Terminer un programme :**

- Appel système numéro 10.

## Directives assembleur

**.align n** : aligne le compteur d'adresse de la section concernée sur une adresse telle que les n bits de poids faible soient à zéro (c'est-à-dire une adresse multiple de 2^n).

**.ascii chaîne [, autrechaîne, ...]** : place à partir de l'adresse du compteur d'adresse de la section concernée la suite de caractères entre guillemets. S'il y a plusieurs chaînes, elles sont placées à la suite. Cette chaîne peut contenir des séquences d'échappement du langage.

**.asciiiz chaîne [, autrechaîne, ...]** : identique à la précédente, la seule différence étant qu'elle ajoute un zéro binaire à la fin de chaque chaîne.

**.byte n [, m, ...]** : les valeurs de n [et m,...] représentées sur 1 octet (tronquées sur 8 bits) sont placées à des adresses successives de la section, à partir de l'adresse du compteur d'adresse de cette section.

**.half n [, m, ...]** : les valeurs de n [et m,...] représentées sur 2 octets (tronquées sur 16 bits) sont placées à des adresses successives de la section, à partir de l'adresse du compteur d'adresse de la section.

**.word n [, m, ...]** : les valeurs de n [et m, ...] représentées sur 4 octets sont placées dans des adresses successives de la section, à partir de l'adresse du compteur d'adresse de la section.

**.space n** : un espace de n octets est réservé à partir du compteur d'adresse de la section concernée. Les octets sont initialisés à la valeur 0 qui est la valeur par défaut

## Conventions d'usage des registres GPR

\$0	Vaut 0 en lecture. Non modifié par une écriture
\$1 (at)	Réservé à l'assembleur pour les macro-instructions. L'usage est interdit pour les programmes.
\$2, \$3 (v0, v1)	Utilisés pour les calculs temporaires et la valeur de retour des fonctions, \$2 est utilisé pour indiquer le n° de service syscall
\$4 .. \$7 (a0 .. a3)	Utilisés pour le passage des arguments de fonctions, leurs valeurs ne sont pas préservées lors des appels de fonctions. Les autres arguments sont placés dans la pile.
\$8 .. \$15, \$24, \$25 (t0 .. t9)	Registres de travail non-persistants, leurs valeurs ne sont pas préservées lors des appels de fonctions
\$16 .. \$23 (s0 .. s8)	Registres persistants, leurs valeurs sont préservées par les appels de fonctions
\$26, \$27 (k0, k1)	Réservés à l'usage noyau du système d'exploitation
\$28 (gp)	Pointeur sur la partie des variables globales (segment data) Utilisé par le compilateur, non utilisé dans cette UE
\$29 (sp)	Pointeur de pile
\$30 (fp)	Pointeur de cadre (pour les contextes de fonctions)
\$31 (ra)	Contient l'adresse de retour d'une fonction

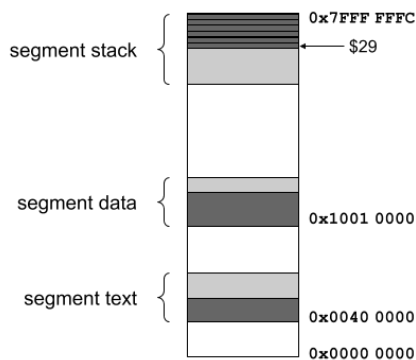
## Définition des acronymes

at : Assembleur Temporary  
a0 ... a3 : Argument 0 à Argument 3  
s0 ... s8 : Saved value 0 à Saved value 8  
gp : Global Pointer  
fp : Frame Pointer

v0, v1 : Value 0 et Value 1  
t0 ... t9 : Temporary 0 à Temporary 9  
k0, k1 : Kernel temporary 0 et 1  
sp : Stack Pointer  
ra : Return Address

## Organisation de la mémoire

### Mémoire de l'utilisateur



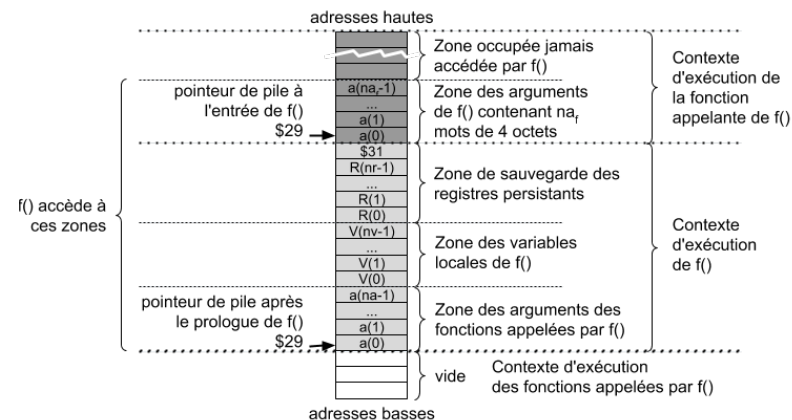
– Les zones blanches correspondent à des adresses légales mais en dehors de segments autorisés.

– Les zones en gris clair représentent des segments autorisés mais vides (non remplis).

– Les zones en gris foncé représentent les données ou les instructions présentes dans les segments autorisés.

## Pile et conventions d'appel de la fonction g par f

Avec na nombre de mots pour les arguments des fonctions appelées par f  
nr nombre de registres persistants utilisés par f  
nv nombre de mots pour le stockage de variables locales de f



## Code ASCII

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8																
9																
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ