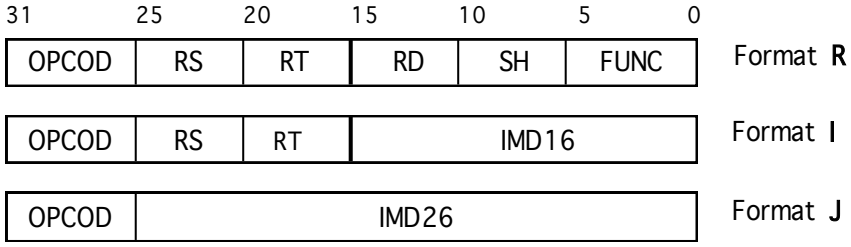
Instructions MIPS

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

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

Format de codage des instructions



Codage des codes opération des instructions

DECODAGE OPCOD

INS 31:29 \ INS 28:26	000	001	010	011	100	101	110	111
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								

OPCOD = SPECIAL

INS 5:3 \ INS 2:0	000	001	010	011	100	101	110	111
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 chaine (de valeur 0).

Lire une chaîne de caractères sur la console :
- Appel système numéro 8.
- 2 paramètres :
    1. l'adresse mémoire à partir de laquelle la chaîne de caractères lue sera sauvegardée doit être placée dans le registre $4.
    2. 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 chaine 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.

.asciiz 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 1octet (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

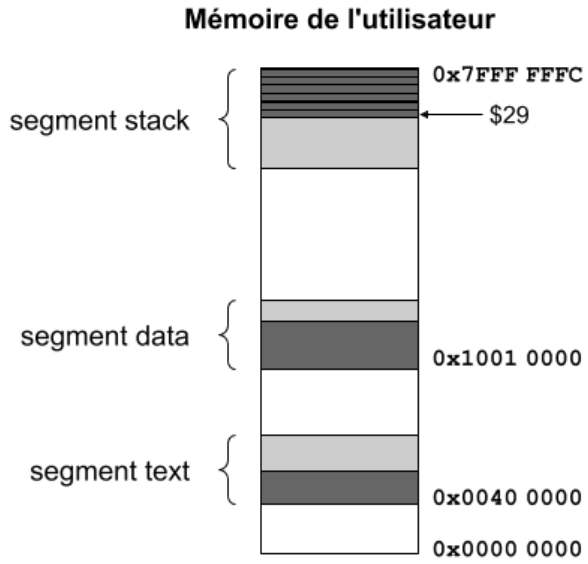
Conventions d'usage des registres GPR

$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



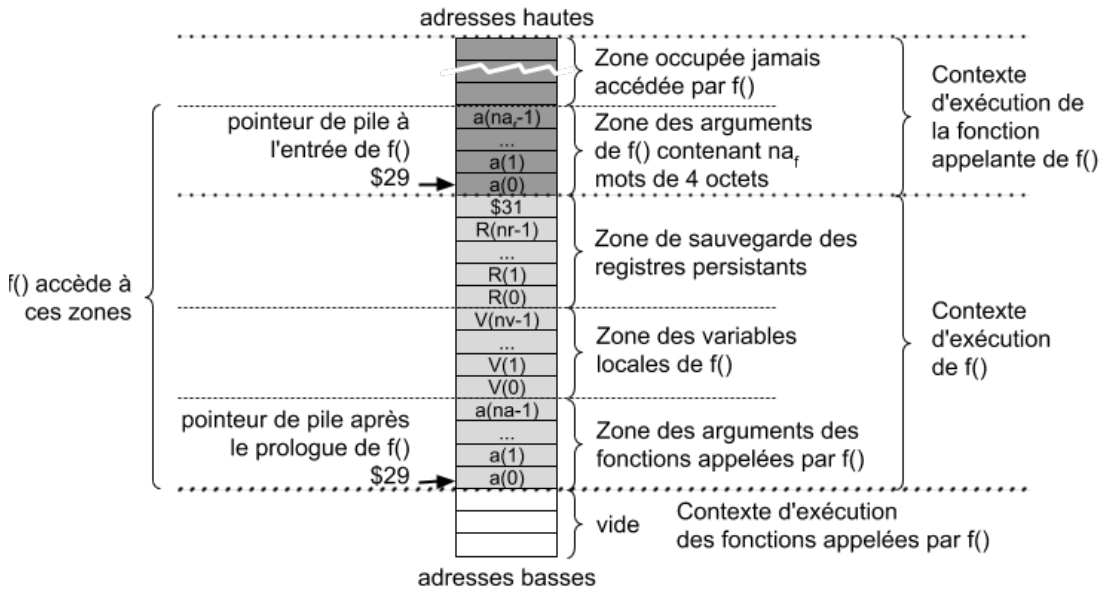
– Les zones blanches correspondent à des adresses légales mais en dehors de segments autorisés.

– 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