Liste des instructions MIPS

Instructions de transferts

Syntaxe Assembleur		Opérations	Commentaires	Effet	Format
Opérations de transferts ALU (*move from/to*)					
mfhi	Rd	Copie du champ Hi	*cf.* mult/div	Rd ← Hi	R
mflo	Rd	Copie du champ Lo	*cf.* mult/div	Rd ← Lo	R
mthi	Rs	Chargement du champ Hi	*cf.* mult/div	Hi ← Rs	R
mtlo	Rs	Chargement du champ Lo	*cf.* mult/div	Lo ← Rs	R
lui	Rt, I	Chargement immédiat	Chargement d'une constante	Rt ← I << 16	I
Opérations de transferts registre-RAM (*load/store*)					
lw	Rt, I(Rs)	Chargement mot		Rt ← RAM[Rs+I]	I
sw	Rt, I(Rs)	Déchargement mot		RAM[Rs+I] ← Rt	I
lh	Rt, I(Rs)	Chargement demi-mot	Extension de signe	Rt ← RAM[Rs+I]	I
lhu	Rt, I(Rs)	Chargement demi-mot non-signé	Sans extension de signe	Rt ← RAM[Rs+I]	I
sh	Rt, I(Rs)	Déchargement demi-mot		RAM[Rs+I] ← Rt	I
lb	Rt, I(Rs)	Chargement octet	Extension de signe	Rt ← RAM[Rs+I]	I
lbu	Rt, I(Rs)	Chargement octet non-signé	Sans extension de signe	Rt ← RAM[Rs+I]	I
sb	Rt, I(Rs)	Déchargement octet		RAM[Rs+I] ← Rt	I

Instructions de branchement de code

Syntaxe Assembleur		Opérations	Commentaires	Effet	Format
Saut conditionnel (*branch if equal/not equal/greater/less... [than zero] [and link]*)					
beq	Rs, Rt, Etq	Branchement si égalité		PC ← PC + 4 + 4 × I si Rs=Rt PC ← PC + 4 sinon	I
bne	Rs, Rt, Etq	Branchement si différent		PC ← PC + 4 + 4 × I si Rs≠Rt PC ← PC + 4 sinon	I
bgez	Rs, Etq	Branchement si sup. ou égal à 0		PC ← PC + 4 + 4 × I si Rs⩾ 0 PC ← PC + 4 sinon	I
bgtz	Rs, Etq	Branchement si sup. à 0		PC ← PC + 4 + 4 × I si Rs > 0 PC ← PC + 4 sinon	I
blez	Rs, Etq	Branchement si inf. ou égal à 0		PC ← PC + 4 + 4 × I si Rs ⩽ 0 PC ← PC + 4 sinon	I
bltz	Rs, Etq	Branchement si inf. à 0		PC ← PC + 4 + 4 × I si Rs < 0 PC ← PC + 4 sinon	I
bgezal	Rs, Etq	Branchement si sup. ou égal et liaison		PC ← PC + 4 + 4 × I si Rs ⩾ 0 PC ← PC + 4 sinon R31 ← PC + 4 dans tous les cas	I
bltzal	Rs, Etq	Branchement si inf. ou égal et liaison		PC ← PC + 4 + 4 × I si Rs < 0 PC ← PC + 4 sinon R31 ← PC + 4 dans tous les cas	I
Saut inconditionnel (*jump [and link]*)					
j	Etq	Saut étiquette		PC ← PC + 4[31...28].4×Etq	J
jal	Etq	Saut et liaison		R31 ← PC + 4 PC ← PC + 4[31...28].4×Etq	J
jr	Rs	Saut sur registre		PC ← Rs	R
jalr	Rs	Saut et lien sur registre		R31 ← PC + 4 PC ← Rs	R
jalr	Rd, Rs	Saut et lien sur registre		Rd ← PC + 4 PC ← Rs	R

Architecture des ordinateurs – Mémento MIPS – Olivier Marchetti

Instructions arithmétiques

Syntaxe Assembleur		Opérations	Commentaires	Effet	Format
Opérations d'addition et de soustraction					
add	Rd, Rs, Rt	addition	Détection de dépassement de capacité	Rd ← Rs + Rt	R
sub	Rd, Rs, Rt	soustraction	Détection de dépassement de capacité	Rd ← Rs − Rt	R
addu	Rd, Rs, Rt	addition (non-signée)	Pas de détection de dépassement de capacité	Rd ← Rs + Rt	R
subu	Rd, Rs, Rt	soustraction (non-signée)	Pas de détection de dépassement de capacité	Rd ← Rs − Rt	R
addi	Rt, Rs, I	addition (op. imm.)	Détection de dépassement de capacité	Rt ← Rs + I	I
addiu	Rt, Rs, I	addition (op. imm., non signée)	Pas de détection de dépassement de capacité	Rt ← Rs + I	I
Opérations de multiplication et de division					
mult	Rs, Rt	multiplication		Hi ← $pref_{32}(Rs \times Rt)$ Lo ← $suff_{32}(Rs \times Rt)$	R
multu	Rs, Rt	multiplication opérandes non signés		Hi ← $pref_{32}(Rs \times Rt)$ Lo ← $suff_{32}(Rs \times Rt)$	R
div	Rs, Rt	division		Hi ← Rs mod Rt Lo ← Rs/Rt	R
divu	Rs, Rt	division opérandes non signés		Hi ← Rs mod Rt Lo ← Rs/Rt	R

Instructions logiques

Syntaxe Assembleur		Opérations	Commentaires	Effet	Format
Opérations logiques de base (bits-à-bits)					
or	Rd, Rs, Rt	OU logique		Rd ← Rs | Rt	R
and	Rd, Rs, Rt	ET logique		Rd ← Rs & Rt	R
xor	Rd, Rs, Rt	OU exclusif logique		Rd ← Rs ^ Rt	R
nor	Rd, Rs, Rt	NON OU logique		Rd ← ∼(Rs | Rt)	R
ori	Rt, Rs, I	OU logique avec op. imm.	immédiat non-signé	Rt ← Rs | I	I
andi	Rt, Rs, I	ET logique avec op. imm.	immédiat non-signé	Rt ← Rs & I	I
xori	Rt, Rs, I	OU exclusif avec op. imm.	immédiat non-signé	Rt ← Rs ^ I	I
Opérations de décalage de bits (*shift left/right logical)*					
sllv	Rd, Rt, Rs	Décalage à gauche avec registre	Rs : 5 bits de poids faibles	Rd ← Rt << Rs	R
srlv	Rd, Rt, Rs	Décalage à droite avec registre	Rs : 5 bits de poids faibles	Rd ← Rt >> Rs	R
srav	Rd, Rt, Rs	Décalage arithmétique* à droite avec registre	Rs : 5 bits de poids faibles	Rd ← Rt >>* Rs	R
sll	Rd, Rt, Dec	Décalage à gauche		Rd ← Rt << Dec	R
srl	Rd, Rt, Dec	Décalage à droite		Rd ← Rt >> Dec	R
sra	Rd, Rt, Dec	Décalage à droite arithmétique	Extension de signe	Rd ← Rt >>* Dec	R
Opérations de test de conditions (*set if less than*)					
slt	Rd, Rs, Rt	positionné si inférieur à		Rd ← 1 si (Rs < Rt) Rd ← 0 sinon	R
sltu	Rd, Rs, Rt	positionné si inférieur à (op. non-signé)		Rd ← 1 si (Rs < Rt) Rd ← 0 sinon	R
slti	Rt, Rs, I	positionné si inférieur à (op. immédiat)	extension de signe pour l'immédiat	Rt ← 1 si (Rs < I) Rt ← 0 sinon	I
sltiu	Rt, Rs, I	positionné si inférieur à (op. immédiat, non-signé)		Rt ← 1 si (Rs < I) Rt ← 0 sinon	I

Directives assembleur

Indiquer le segment mémoire

- `.text [adr]` : la donnée sera dans le segment texte (*i.e.* avec les instructions du programme – à l'adresse `adr` si fournie).
- `.data [adr]` : la donnée sera stockée dans le segment des données (à l'adresse `adr` si fournie).
- `.stack` : la donnée sera sur la pile.
- `.k{text,data}` : la donnée sera dans le segment {texte, données} du noyau du système d'exploitation.

Décrire la visibilité

- `.extern Etq taille` : déclare la donnée enregistrée à l'adresse `Etq` sur `taille` octets consécutifs comme étant globale au fichier. Place la donnée dans le segment des données.
- `.globl Etq` : déclare l'étiquette `Etq` est un symbole globale visible dans d'autres fichiers assembleur.

Décrire et enregistrer les données

Les enregistrements mémoire suivants sont réalisés à des adresses consécutives.

- `.align n` : réalise l'alignement mémoire de la donnée suivante sur une frontière de 2^n octects.
- `.space n` : définit un espace de n octets consécutifs dans le segment des données.
- `.ascii ch` : enregistre en mémoire la chaîne de caractère `ch` sans le caractère de fin de chaîne.
- `.asciiz ch` : idem, mais avec le caractère de fin de chaîne.
- `.byte` $b_1,\ldots,b_n$: enregistre n octets.
- `.double` $d_1,\ldots,d_n$: enregistre n flottants double précision.
- `.float` $f_1,\ldots,f_n$: enregistre n flottants simple précision.
- `.half` $h_1,\ldots,h_n$: enregistre n quantités 16 bits (alignement demi-mot).
- `.word` $w_1,\ldots,w_n$: enregistre n quantités 32 bits (alignement mot).

Appels système

Méthodologie

1. Ecrire l'instruction pour charger le service voulu dans le registre `$v0`.
2. [Ecrire les instructions pour charger des arguments dans les registres `$a0`, `$a1`...(*cf.* documentation MIPS).]
3. Ecrire l'instruction `syscall`.
4. [Ecrire les instructions pour récupérer une valeur de retour de l'appel `syscall`.]

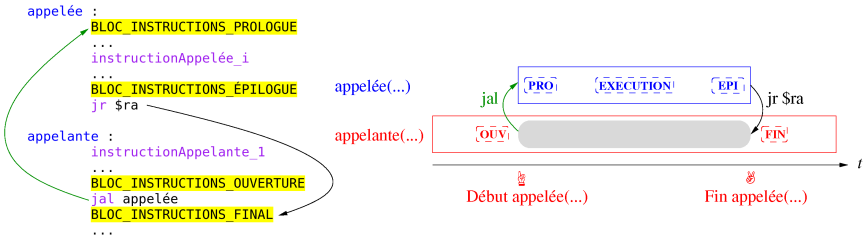
Service	Code Service	Arguments	Valeur de retour
afficher entier	1	`$a0` : stocke l'entier voulu	
afficher ch. car.	4	`$a0` : adresse de la chaîne	
lire entier	5		`$v0` : entier lu
lire ch. car.	8	`$a0` : adresse du tampon	
		`$a1` : nb max car. lu	
exit	10		

Fonction

Méthodologie

Procéder en 5 étapes :

ouverture, prologue, exécution, épilogue, final.



1. **OUV**erture : l'appelante prépare la réalisation effective de l'appelée (sauvegarde éventuelle de registres).
2. **PRO**logue : l'appelée prépare son bloc de données sur la pile.
3. **EXECUTION** : l'appelée exécute ses instructions propres.
4. **EPI**logue : l'appelée rend la main en restituant le contexte de l'appelant et en dépilant son bloc mémoire de la pile.
5. **FIN**al : l'appelante restaure éventuellement des registres et reprend le code de la fonction implémentée.

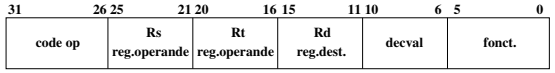
Registres à sauvegarder lors d'un appel

Nom	Numéro	Usage	Appelante	Appelée
$v0–$v1	$2–$3	Val. retour et Rés. fonction	[√]	–
$a0–$a3	$4–$7	Arg. fonct.	[√]	–
$t0–$t7	$8–$15	Temp.	[√]	–
$s0–$s7	$16–$23	Temp. sauv.	–	[√]
$t8–$t9	$24–$25	Temp.	[√]	–
$sp	$29	Pointeur pile	–	[√]
$fp	$30	Pointeur bloc	–	[√]
$ra	$31	Adr. retour	–	[√]

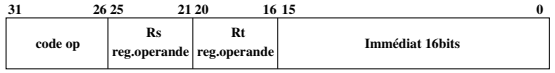
Toutes les sauvegardes de registres sont codifiées par la convention décrite précédemment. Cependant, la sauvegarde d'un registre donné n'est pas nécessairement systématique. Ce caractère non-systématique est traduit dans le tableau ci-dessus par la notation entre crochets (usuellement utilisée pour la syntaxe des options en programmation). Une entrée du tableau entre crochets se trouve conditionnée d'une part par le fait qu'un appel de fonction a bien eu lieu, et d'autre part selon l'usage ou non qui est fait du registre par l'appelante ou l'appelée. Par exemple, si la fonction appelante a besoin d'accéder aux valeurs qu'elle a stocké dans les registres $t0–$t9, elle doit les sauvegarder dans son bloc de pile avant l'appel de fonction car ces registres sont susceptibles d'être modifiés par la fonction appelée sans précaution particulière. Si ce n'est pas le cas, la fonction appelante ne les empile pas.

Codage des instructions

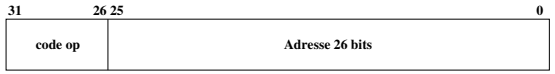
Format d'instruction R registre–registre



Format d'instruction I immédiat



Format d'instruction J saut d'instructions



CODE OP

Bits 31:29 \ Bits 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								

BCOND

Bit 20 \ Bit 16	0	1
0	BLTZ	BGEZ
1	BGTZAL	BGEZAL

COPRO

Bit 25 \ Bit 23	0	1
0	MFC0	MTC0
1	RFE	

SPECIAL

Bits 5:3 \ Bits 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								

Codage des données

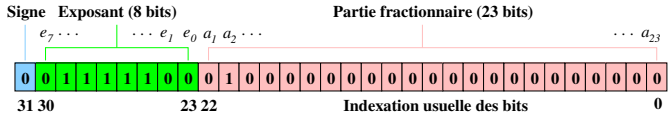
Table d'encodage ASCII

	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		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
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

Exemple : le caractère `'A'` est encodé par 65 $(4 \times 16 + 1)$, ou encore $(41)_{16}$.

Remarque : cette table est aussi facilement accessible sur les systèmes d'exploitation de type GNU/Linux en tapant `man ascii` dans un terminal.

Format d'encodage des flottants simple précision IEEE-754



Quelques remarques : l'exposant encodé est un exposant biaisé. Il faut retrancher le biais $B = 127$ de cette valeur pour obtenir l'exposant réel du nombre encodé. Seule la partie fractionnaire de la notation scientifique normalisée est encodée dans la mantisse. Ainsi, l'exemple ci-dessus encode le nombre $0,15625$.