Set de Instruções da Máquina Nativa (1)

Menemónica	*Descrição*	*OAMV*	*E/O*	*Algoritmo*
Instruções de Transferência Memória-Registo (*Load*)				
`lb Rdst,addr`	Load Byte			Rdst = *(byte)addr
`lbu Rdst,addr`	Load Byte Unsigned			Rdst = *(unsigned byte)addr
`lw Rdst,addr`	Load Word			Rdst = *(word)addr
`lwcz CReg,addr`	Load Word Coprocessor z			C_zReg = *(word)addr
Instruções de Transferência Registo-Memória (*Store*)				
`sb Rsrc,addr`	Store Byte			*addr = (byte)Rsrc
`sw Rsrc,addr`	Store Word			*addr = (word)Rsrc
`swcz Creg,addr`	Store Word Coprocessor z			*(word)addr = C_zReg
Instruções de Transferência Registo-Registo (*Move*)				
`mfhi Rdst`	Move From HI			Rdst = HI
`mflo Rdst`	Move From LO			Rdst = LO
`mthi Rsrc`	Move To HI			HI = Rsrc
`mtlo Rsrc`	Move To LO			LO = Rsrc
`mfcz Rdst,CReg`	Move From Coprocessor C_z			Rdst = C_zReg
`mtcz Rsrc,CReg`	Move To Coprocessor C_z			C_zReg = Rsrc
`mov.d FPdst,FPsrc`	Move Double			FPdst = FPsrc
`mov.s FPdst,FPsrc`	Move Single			FPdst = FPsrc
Instruções de Manipulação de Constantes (*Load Immediate*)				
`lui Rdst,Imm`	Load Upper Immediate			Rdst = Imm << 16
Instruções de Cálculo sobre Inteiros: Operações Aritméticas				
`add Rdst,Rsrc1,Rsrc2`	Add	`Rdst,Rsrc,IMM`	✓	Rdst = Rsrc + Src
`addi Rdst,Rsrc,Imm`	Add Immediate	`Rdst,Rsrc,IMM`	✓	Rdst = Rsrc + Imm
`addiu Rdst,Rsrc,Imm`	Add Immediate Unsigned	`Rdst,Rsrc,IMM`		Rdst = Rsrc + Imm
`addu Rdst,Rsrc1,Rsrc2`	Add Unsigned	`Rdst,Rsrc,IMM`		Rdst = Rsrc + Src
`div Rsrc1,Rsrc2`	Divide		✓	HI = Rsrc % Rsrc2; LO = Rsrc / Rsrc2
`divu Rsrc1,Rsrc2`	Divide Unsigned			HI = Rsrc % Rsrc2; LO = Rsrc / Rsrc2
`mult Rsrc1,Rsrc2`	Multiply			HI,LO = Rsrc1 * Rsrc2
`multu Rsrc1,Rsrc2`	Multiply Unsigned			HI,LO = Rsrc1 * Rsrc2
`sub Rdst,Rsrc1,Rsrc2`	Subtract	`Rdst,Rsrc,IMM`	✓	Rdst = Rsrc - Src
`subu Rdst,Rsrc1,Rsrc2`	Subtract Unsigned	`Rdst,Rsrc,IMM`		Rdst = Rsrc - Src

Set de Instruções da Máquina Nativa (2)

Menemónica		*Descrição*	*OAMV*	*E/O*	*Algoritmo*
Instruções de Cálculo sobre Inteiros: Operações Lógicas Bit-a-Bit (*Bitwise*)					
and	Rdst,Rsrc1,Rsrc2	And (Bitwise)	Rdst,Rsrc,IMM		Rdst = Rsrc & Src
andi	Rdst,Rsrc,Imm	And Immediate (Bitwise)	Rdst,Rsrc,IMM		Rdst = Rsrc & Imm
nor	Rdst,Rsrc1,Rsrc2	Nor (Bitwise)	Rdst,Rsrc,IMM		Rdst = ~(Rsrc | Src)
or	Rdst,Rsrc1,Rsrc2	Or (Bitwise)	Rdst,Rsrc,IMM		Rdst = Rsrc | Src
ori	Rdst,Rsrc,Imm	Or Immediate (Bitwise)	Rdst,Rsrc,IMM		Rdst = Rsrc | Imm
xor	Rdst,Rsrc1,Rsrc2	XOR	Rdst,Rsrc,IMM		Rdst = Rsrc ^ Src
xori	Rdst,Rsrc,Imm	XOR Immediate	Rdst,Rsrc,IMM		Rdst = Rsrc ^ Imm
Instruções de Cálculo sobre Inteiros: Operações de Deslocamento (*Shift*)					
sll	Rdst,Rsrc1,Imm5	Shift Left Logical	Rdst,Rsrc,IMM		Rdst = Rsrc << Src
sllv	Rdst,Rsrc1,Rsrc2	Shift Left Logical Variable			Rdst = Rsrc << Rsrc2
sra	Rdst,Rsrc1,Imm5	Shift Right Arithmetic	Rdst,Rsrc,IMM		Rdst = Rsrc >> Src
srav	Rdst,Rsrc1,Rsrc2	Shift Right Arithmetic Variable			Rdst = Rsrc >> Rsrc2
srl	Rdst,Rsrc1,Imm5	Shift Right Logical	Rdst,Rsrc,IMM		Rdst = Rsrc >> Src
srlv	Rdst,Rsrc1,Rsrc2	Shift Right Logical Variable			Rdst = Rsrc >> Rsrc2
Instruções de Comparação					
slt	Rdst,Rsrc1,Rsrc2	Set on Less Than	Rdst,Rsrc,IMM		Rdst = (Rsrc < Src) ? 1 : 0
sltu	Rdst,Rsrc1,Rsrc2	Set on Less Than Unsigned	Rdst,Rsrc,IMM		Rdst = (Rsrc < Src) ? 1 : 0
slti	Rdst,Rsrc,Imm	Set on Less Than Immediate	Rdst,Rsrc,IMM		Rdst = (Rsrc < Imm) ? 1 : 0
sltiu	Rdst,Rsrc,Imm	Set on Less Than Imm. Unsigned	Rdst,Rsrc,IMM		Rdst = (Rsrc < Imm) ? 1 : 0
Instruções de Salto Relativo (*Branch*) e Salto Absoluto (*Jump*)					
bczf	Label	Branch Coprocessor z FALSE			if (c_z_flag == FALSE) goto label
bczt	Label	Branch Coprocessor z TRUE			if (c_z_flag == TRUE) goto label
beq	Rsrc1,Rsrc2,Label	Branch on Equal	Rsrc1,IMM,Label		if (Rsrc1 == Src2) goto label
bgez	Rsrc,Label	Branch on Greater Than or Equal Zero			if (Rsrc1 >= 0) goto label
bgezal	Rsrc,Label	Branch on Greater Than or Equal Zero and Link			if (Rsrc1 >= 0) {$31=PC; goto label }
bgtz	Rsrc,Label	Branch on Greater Than Zero			if (Rsrc1 > 0) goto label
blez	Rsrc,Label	Branch on Less Than or Equal Zero			if (Rsrc1 <= 0) goto label
bltz	Rsrc,Label	Branch on Less Than Zero			if (Rsrc1 < 0) goto label
bltzal	Rsrc,Label	Branch on Less Than Zero and Link			if (Rsrc1 < 0) {$31=PC; goto label }
bne	Rsrc1,Rsrc2,Label	Branch on Not Equal	Rsrc1,IMM,Label		if (Rsrc1 != Src2) goto label
j	Label	Jump			goto label
jal	Label	Jump and Link			$31=PC; goto label
jalr	Rsrc	Jump and Link Register			$31=PC; goto Rsrc
jr	Rsrc	Jump Register			goto Rsrc

Set de Instruções da Máquina Nativa (3)

Menemónica	*Descrição*	*E/O*	*Algoritmo*
Instruções de Cálculo em Vírgula Flutuante			
`abs.p FPdst,FPsrc`	Absolute Value		FPdst = | FPsrc |
`add.p FPdst,FPsrc1,FPsrc2`	Addition		FPdst = FPsrc1 + FRsrc2
`c.eq.p FPsrc1,FPsrc2`	Compare Equal		fp_flag = (FPsrc1 == FPsrc2) ? true : false
`c.le.p FPsrc1,FPsrc2`	Compare Less Than or Equal		fp_flag = (FPsrc 1<= FPsrc2) ? true : false
`c.lt.p FPsrc1,FPsrc2`	Compare Less Than		fp_flag = (FPsrc 1< FPsrc2) ? true : false
`cvt.d.s FPdst,FPsrc`	Convert Single to Double		FPdst = (double)FPsrc
`cvt.d.w FPdst,FPsrc`	Convert Integer to Double		FPdst = (double)FPsrc
`cvt.s.d FPdst,FPsrc`	Convert Double to Single		FPdst = (float)FPsrc
`cvt.s.w FPdst,FPsrc`	Convert Integer to Single		FPdst = (float)FPsrc
`cvt.w.d FPdst,FPsrc`	Convert Double to Integer		FPdst = (int)FPsrc
`cvt.w.s FPdst,FPsrc`	Convert Single to Integer		FPdst = (int)FPsrc
`div.p FPdst,FPsrc1, FPsrc2`	Divide		FPdst = FPsrc1 / FPsrc2
`mul.p FPdst,FPsrc1, FPsrc2`	Multiply		FPdst = FPsrc1 * FPsrc2
`neg.p FPdst,FPsrc`	Negate		FPdst = 0 - FPsrc
`sub.p FPdst,FPsrc1, FPsrc2`	Subtract		FPdst = FPsrc1 - FPsrc2
Instruções para Manipulação de Excepções e *Traps*			
`break n`	Break		
`nop`	No Operation		
`eret`	Return From Exception		
`syscall`	System Call (ver tabela VII)		

Tabela I: Registos do MIPS e convenção de uso

Nome Lógico	Nome Real	Uso Convencionado
$zero	$0	Constante 0
$at	$1	Reservado pelo assemblador
$v0..$v1	$2..$3	Cálculo de expressões e valor de retorno das funções.
$a0..$a3	$4..$7	Primeiros 4 parâmetros das funções
$t0..$t7	$8..$15	Geral (não são preservados pelas funções)
$s0..$s7	$16..$23	Geral (não podem ser alterados pelas funções)
$t8..$t9	$24..$25	Geral (não são preservados pelas funções)
$k0..$k1	$26..$27	Reservado pelo *kernel* do S.O.
$gp	$28	Ponteiro para área global (*Global Pointer*)
$sp	$29	*Stack Pointer*
$fp	$30	*Frame Pointer*
$ra	$31	Endereço de retornos das funções (*Return Address*)

Set de Instruções da Máquina Virtual (1)

Menemónica	*Descrição*	*E/O*	*Algoritmo*
Instruções de Transferência Memória-Registo (*Load)*			
`l.d FPdst,addr`	Load Double		FPdst = *(double)addr
`l.s FPdst,addr`	Load Single		FPdst = *(single)addr
Instruções de Transferência Registo-Memória (*Store)*			
`s.d FPsrc,addr`	Store Double		*(double)addr = FPsrc
`s.s FPsrc,addr`	Store Single		*(single)addr = FPsrc
Instruções de Transferência Registo-Registo (*Move*)			
`move Rdst,Rsrc`	Move		Rdst = Rsrc

Set de Instruções da Máquina Virtual (2)

Menemónica	*Descrição*	*E/O*	*Algoritmo*
Instruções de Manipulação de Constantes (*Load Immediate*)			
`la Rdst,sym`	Load Address		Rdst = sym
`li Rdst,IMM`	Load Immediate		Rdst = IMM
`l.d FPdst,addr`	Load Immediate Double		FPdst = double
`l.s FPdst,addr`	Load Immediate Single		FPdst = float
Instruções de Cálculo sobre Inteiros: Operações Aritméticas			
`abs Rdst,Rsrc`	Absolute Value		Rdst = | Rsrc |
`div Rdst,Rsrc,Src`	Division	✓	Rdst = Rsrc / Src
`divu Rdst,Rsrc,Src`	Division Unsigned	✓	Rdst = Rsrc / Src
`mul Rdst,Rsrc,Src`	Multiply		Rdst = Rsrc * Src
`mulu Rdst,Rsrc,Src`	Multiply Unsigned		Rdst = Rsrc * Src
`mulo Rdst,Rsrc,Src`	Multiply	✓	Rdst = Rsrc * Src
`mulou Rdst,Rsrc,Src`	Multiply Unsigned	✓	Rdst = Rsrc * Src
`neg Rdst,Rsrc`	Negate		Rdst = - Rsrc
`negu Rdst,Rsrc`	Negate		Rdst = 0 - Rsrc
`rem Rdst,Rsrc,Src`	Remainder	✓	Rdst = Rsrc % Src
`remu Rdst,Rsrc,Src`	Remainder Unsigned	✓	Rdst = Rsrc % Src
Instruções de Cálculo sobre Inteiros: Operações Lógicas Bit-a-Bit (*Bitwise*)			
`not Rdst,Rsrc`	Not (Bitwise)		
Instruções de Cálculo sobre Inteiros: Operações de Rotação (*Rotate*)			
`rol Rdst,Rsrc,Src`	Rotate Left		Rdst = Rsrc <<^{o} Src
`ror Rdst,Rsrc,Src`	Rotate Right		Rdst = Rsrc >>^{o} Src
Instruções de Comparação			
`seq Rdst,Rsrc,Src`	Set on Equal		Rdst = (Rsrc == Src) ? 1 : 0
`sge Rdst,Rsrc,Src`	Set on Greater Than or Equal		Rdst = (Rsrc >= Src) ? 1 : 0
`sgeu Rdst,Rsrc,Src`	Set on Greater Than or Equal Unsigned		Rdst = (Rsrc >= Src) ? 1 : 0
`sgt Rdst,Rsrc,Src`	Set on Greater Than		Rdst = (Rsrc > Src) ? 1 : 0
`sgtu Rdst,Rsrc,Src`	Set on Greater Than Unsigned		Rdst = (Rsrc > Src) ? 1 : 0
`sle Rdst,Rsrc,Src`	Set on Less Than or Equal		Rdst = (Rsrc <= Src) ? 1 : 0
`sleu Rdst,Rsrc,Src`	Set on Less Than or Equal Unsigned		Rdst = (Rsrc <= Src) ? 1 : 0
`sne Rdst,Rsrc,Src`	Set on Not Equal		Rdst = (Rsrc != Src) ? 1 : 0
Instruções de Salto Relativo (*Branch*) e Salto Absoluto (*Jump*)			
`b Label`	Branch		goto label
`beqz Rsrc, Label`	Branch on Equal Zero		if (Rsrc == 0) goto label
`bge Rsrc,Src,Label`	Branch on Greater Than or Equal		if (Rsrc >= Src) goto label
`bgeu Rsrc,Src,Label`	Branch on Greater Than or Equal Unsigned		if (Rsrc >= Src) goto label
`bgt Rsrc,Src,Label`	Branch on Greater Than		if (Rsrc > Src) goto label
`bgtu Rsrc,Src,Label`	Branch on Greater Than Unsigned		if (Rsrc > Src) goto label
`ble Rsrc,Src,Label`	Branch on Less Than or Equal		if (Rsrc <= Src) goto label
`bleu Rsrc,Src,Label`	Branch on Less Than or Equal Unsigned		if (Rsrc <= Src) goto label
`blt Rsrc,Src,Label`	Branch on Less Than		if (Rsrc < Src) goto label
`bltu Rsrc,Src,Label`	Branch on Less Than Unsigned		if (Rsrc < Src) goto label
`bnez Rsrc,Label`	Branch on Not Equal Zero		if (Rsrc != 0) goto label

Tabela II: Registos de I/O mapeado em memória

Nome	Endereço	Bit 7-3	Bit 1	Bit 0
Controlo de Recepção	0xffff0000	Não usados	Int Enable	Ready
Dados do Receptor	0xffff0004	Byte recebido		
Controlo de Emissão	0xffff0008	Não usados	Int Enable	Ready
Dados do Emissor	0xffff000c	Byte a enviar		

Tabela III: Registos da FPU do MIPS e convenção de uso

Nome Lógico	**Uso Convencionado**
$f0($f1) ... $f2($f3)	Cálculo de expressões e valor de retorno das funções.
$f4($f5) ... $f10($f11)	Geral (não são preservados pelas funções)
$f12($f13) ... $f14($f15)	Passagem de parâmetros para funções.
$f16($f17) ... $f18($f19)	Geral (não são preservados pelas funções)
$f20($f21) ... $f30($f31)	Geral (não podem ser alterados pelas funções)

Tabela IV: Registos do CP0 do MIPS

Nome Lógico	**Nome Real**	**Conteúdo**
$BadVAddr	$8	Endereço de memória inválido que causou a excepção
$Status	$12	Interrupt mask & Enable bits
$Cause	$13	Tipo de excepção e interrupt bits
$EPC	$14	Endereço da instrução que causou a excepção

Tabela V: Valores dos bits [5..2] do registo Cause

Valor	**Nomel**	**Significado**
0	INT	External Interrupt
4	ADDRL	Add error exception (load or store)
5	ADDRS	Add error exception (fetch)
6	IBUS	Bus error on instruction fetch
7	DBUS	Bus error on data load or store
8	SYSCALL	Syscall exception
9	BKPT	Break point exception
10	RI	Reserved instruction exception
12	OVF	Overflow exception

Tabela VI: Notação			
`Imm`	Valor imediato (constante) de 16 bits	`addr`	Endereço na forma **Imm(Rsrc) = (Rsrc) + Imm**
`IMM`	Valor imediato de 32 bits	B_k`(Rsrc)`	**Byte** índice **k** de **Rsrc**
`Rsrc(1,2)`	Registo fonte (1 ou 2)	`FPdst`	Registo destino do coprocessador aritmético
`(Rsrc)`	Conteudo de **Rsrc**	`FPsrc(1,2)`	Registo fonte do coprocessador aritmético (1 ou 2)
`Rdst`	Registo destino	C_z	Coprocessador nº **z**
`OAMV`	Operandos Alternativos em Modo Virtual	`Src`	**Rsrc** ou **IMM**
`E/O`	Excepção gerada em caso de overflow	`sym`	Endereço do símbolo (label) sym
`CReg`	Registo do Coprocessador C_z	`Imm5`	Valor imediato (constante) de 5 bits

Tabela VII: *System Calls* do MARS			
Protótipo equivalent em C	**$v0**	**Parâmetros de entrada**	**Retorno**
void print_int10(**int** value)	1	$a0 = value	
void print_float(**float** value)	2	$f12 = value	
void print_double(**double** value)	3	$f12 = value	
void print_string(**char** *str)	4	$a0 = str	
int read_int(**void**)	5		$v0
float read_float(**void**)	6		$f0
double read_double(**void**)	7		$f0
void read_string(**char** *buf, **int** length)	8	$a0 = buf, $a1 = length	
void * sbrk(**int** amount)	9	$a0 = amount	$v0
void exit(**void**)	10		
void print_char(**char** value)	11	$a0 = character	
char read_char(**void**)	12		$v0
void print_int16(**unsigned int** value)	34	$a0	
void print_int2(**unsigned int** value)	35	$a0	
void print_intu10(**unsigned int** value)	36	$a0	

Tabela VIII - Directivas do Assembler	
Directivas	**Descrição**
Para controlo dos Segmentos	
`.data <address>`	Coloca os próximos items no segmento de dados do utilizador (opcionalmente a partir de *address*).
`.text <address>`	Coloca os próximos items no segmento de código do utilizador(opcionalmente a partir de *address*). Todos os items devem medir 32 bits, ou seja, serem instruções ou palavras (*words*).
`.kdata <address>`	Coloca os próximos items no segmento de dados do *kernel*(opcionalmente a partir de *address*).
`.ktext <address>`	Coloca os próximos items no segmento de código do *kernel*(opcionalmente a partir de *address*). Todos os items devem medir 32 bits, ou seja, serem instruções ou palavras (*words*).
Para criação de constantes e variáveis em memória:	
`.eqv label, valor`	Substitui todas as ocorrências de *label* no programa por *valor*.
`.ascii str`	Armazena uma *string* em memória sem lhe acrescentar o terminador NULL.
`.asciiz str`	Armazena uma *string* em memória acrescentando-lhe o terminador NULL.
`.byte` $b_1, \ldots, b_n$	Armazena as grandezas de 8 bits $b_1, \ldots, b_n$ em sucessivos bytes de memória.
`.half` $h_1, \ldots, h_n$	Armazena as grandezas de 16 bits $h_1, \ldots, h_n$ em sucessivas meias palavras de memória.
`.word` $w_1, \ldots, w_n$	Armazena as grandezas de 32 bits $w_1, \ldots, w_n$ em sucessivas palavras de memória.
`.float` $f_1, \ldots, f_n$	Armazena os números em vírgula flutuante com precisão simples (32 bits) $f_1, \ldots, f_n$ em posições de memória sucessivas.
`.double` $d_1, \ldots, d_n$	Armazena os números em vírgula flutuante com precisão dupla (64 bits) $d_1, \ldots, d_n$ em posições de memória sucessivas.
`.space n`	Reserva *n* bytes no segmento de dados, sem inicialização.
Para controlo do alinhamento:	
`.align n`	Alinha o próximo item num endereço múltiplo de 2^n. Por exemplo `.align 2` seguido de `.word xpto` garante que a palavra `xpto` é armazenada num endereço múltiplo de 4.
`.align 0`	Desliga o alinhamento automático das directivas `.half`, `.word`, `.float`, e `.double` até à próxima directiva `.data` ou `.kdata`.
Para referências externas:	
`.globl sym`	Declara que o símbolo `sym` é global e pode ser referenciado em outros ficheiros.
`.extern sym size`	Declara que o item associado a `sym` ocupa size bytes e é um símbolo global. Esta directiva permite ao assemblador armazenar o item numa porção do segmento de dados que seja eficientemente acedido através do registo `$gp`.