MIPS Registers and Usage Convention

Register Name	Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Expression evaluation and results of a function
v1	3	Expression evaluation and results of a function
a0	4	Argument 1
a1	5	Argument 2
a2	6	Argument 3
a3	7	Argument 4
t0	8	Temporary (not preserved across call)
t1	9	Temporary (not preserved across call)
t2	10	Temporary (not preserved across call)
t3	11	Temporary (not preserved across call)
t4	12	Temporary (not preserved across call)
t5	13	Temporary (not preserved across call)
t6	14	Temporary (not preserved across call)
t7	15	Temporary (not preserved across call)
s0	16	Saved temporary (preserved across call)
s1	17	Saved temporary (preserved across call)
s2	18	Saved temporary (preserved across call)
s3	19	Saved temporary (preserved across call)
s4	20	Saved temporary (preserved across call)
s5	21	Saved temporary (preserved across call)
s6	22	Saved temporary (preserved across call)
s7	23	Saved temporary (preserved across call)
t8	24	Temporary (not preserved across call)
t9	25	Temporary (not preserved across call)
k0	26	Reserved for OS kernel
k1	27	Reserved for OS kernel
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address (used by function call)

System Services

Service	System Call Code	Arguments	Result
print_int	1	`$a0` = integer	
print_float	2	`$f12` = float	
print_double	3	`$f12` = double	
print_string	4	`$a0` = string	
read_int	5		integer (in `$v0`)
read_float	6		float (in `$f0`)
read_double	7		double (in `$f0`)
read_string	8	`$a0` = buffer, `$a1` = length	
sbrk	9	`$a0` = amount	address (in `$v0`)
exit	10		

Assembler Directives

`.align n`	Align the next datum on a 2^n byte boundary. For example, .align 2 aligns the next value on a word boundary. .align 0 turns off automatic alignment of .half, .word, .float, and .double directives until the next .data or .kdata directive.
`.ascii str`	Store the string in memory, but do not null-terminate it.
`.asciiz str`	Store the string in memory and null-terminate it.
`.byte b1, ..., bn`	Store the *n* values in successive bytes of memory.
`.data`	The following data items should be stored in the data segment. If the optional argument *addr* is present, the items are stored beginning at address *addr*.
`.double d1, ..., dn`	Store the *n* floating point double precision numbers in successive memory locations.
`.extern sym size`	Declare that the datum stored at sym is size bytes large and is a global symbol. This directive enables the assembler to store the datum in a portion of the data segment that is efficiently accessed via register $gp.
`.float f1, ..., fn`	Store the *n* floating point single precision numbers in successive memory locations.
`.globl sym`	Declare that symbol sym is global and can be referenced from other files.
`.half h1, ..., hn`	Store the *n* 16-bit quantities in successive memory halfwords.
`.kdata`	The following data items should be stored in the kernel data segment. If the optional argument *addr* is present, the items are stored beginning at address *addr*.
`.ktext`	The next items are put in the kernel text segment. In SPIM, these items may only be instructions or words (see the .word directive below). If the optional argument *addr* is present, the items are stored beginning at address *addr*.
`.space n`	Allocate *n* bytes of space in the current segment (which must be the data segment in SPIM).
`.text`	The next items are put in the user text segment. In SPIM, these items may only be instructions or words (see the .word directive below). If the optional argument *addr* is present, the items are stored beginning at address *addr*.
`.word w1, ..., wn`	Store the *n* 32-bit quantities in successive memory words.

SPIM Instruction Set

Arithmetic and Logical Instructions

`abs Rdest, Rsrc`	Absolute Value
`add Rdest, Rsrc1, Src2`	Addition (with overflow)
`addi Rdest, Rsrc1, Imm`	Addition Immediate (with overflow)
`addu Rdest, Rsrc1, Src2`	Addition (without overflow)
`addiu Rdest, Rsrc1, Imm`	Addition Immediate (without overflow)
`and Rdest, Rsrc1, Src2`	AND
`andi Rdest, Rsrc1, Imm`	AND Immediate
`div Rsrc1, Rsrc2`	Divide (with overflow)
`divu Rsrc1, Rsrc2`	Divide (without overflow)
`div Rdest, Rsrc1, Src2`	Divide (with overflow)
`divu Rdest, Rsrc1, Src2`	Divide (without overflow)
`mul Rdest, Rsrc1, Src2`	Multiply (without overflow)
`mulo Rdest, Rsrc1, Src2`	Multiply (with overflow)
`mulou Rdest, Rsrc1, Src2`	Unsigned Multiply (with overflow)
`mult Rsrc1, Rsrc2`	Multiply
`multu Rsrc1, Rsrc2`	Unsigned Multiply
`neg Rdest, Rsrc`	Negate Value (with overflow)
`negu Rdest, Rsrc`	Negate Value (without overflow)
`nor Rdest, Rsrc1, Src2`	NOR
`not Rdest, Rsrc`	NOT
`or Rdest, Rsrc1, Src2`	OR
`ori Rdest, Rsrc1, Imm`	OR Immediate
`rem Rdest, Rsrc1, Src2`	Remainder
`remu Rdest, Rsrc1, Src2`	Unsigned Remainder
`rol Rdest, Rsrc1, Src2`	Rotate Left
`ror Rdest, Rsrc1, Src2`	Rotate Right
`sll Rdest, Rsrc1, Src2`	Shift Left Logical
`sllv Rdest, Rsrc1, Rsrc2`	Shift Left Logical Variable
`sra Rdest, Rsrc1, Src2`	Shift Right Arithmetic
`srav Rdest, Rsrc1, Rsrc2`	Shift Right Arithmetic Variable
`srl Rdest, Rsrc1, Src2`	Shift Right Logical
`srlv Rdest, Rsrc1, Rsrc2`	Shift Right Logical Variable
`sub Rdest, Rsrc1, Src2`	Subtract (with overflow)
`subu Rdest, Rsrc1, Src2`	Subtract (without overflow)
`xor Rdest, Rsrc1, Src2`	XOR
`xori Rdest, Rsrc1, Imm`	XOR Immediate

Constant-Manipulating Instructions

`li Rdest, imm`	Load Immediate
`lui Rdest, imm`	Load Upper Immediate

Comparison Instructions

`seq Rdest, Rsrc1, Src2`	Set Equal
`sge Rdest, Rsrc1, Src2`	Set Greater Than Equal
`sgeu Rdest, Rsrc1, Src2`	Set Greater Than Equal Unsigned
`sgt Rdest, Rsrc1, Src2`	Set Greater Than
`sgtu Rdest, Rsrc1, Src2`	Set Greater Than Unsigned
`sle Rdest, Rsrc1, Src2`	Set Less Than Equal
`sleu Rdest, Rsrc1, Src2`	Set Less Than Equal Unsigned
`slt Rdest, Rsrc1, Src2`	Set Less Than
`slti Rdest, Rsrc1, Imm`	Set Less Than Immediate
`sltu Rdest, Rsrc1, Src2`	Set Less Than Unsigned
`sltiu Rdest, Rsrc1, Imm`	Set Less Than Unsigned Immediate
`sne Rdest, Rsrc1, Src2`	Set Not Equal

Branch and Jump Instructions

`b label`	Branch instruction
`bczt label`	Branch Coprocessor *z* True
`bczf label`	Branch Coprocessor *z* False
`beq Rsrc1, Src2, label`	Branch on Equal
`beqz Rsrc, label`	Branch on Equal Zero
`bge Rsrc1, Src2, label`	Branch on Greater Than Equal
`bgeu Rsrc1, Src2, label`	Branch on GTE Unsigned
`bgez Rsrc, label`	Branch on Greater Than Equal Zero
`bgezal Rsrc, label`	Branch on Greater Than Equal Zero And Link
`bgt Rsrc1, Src2, label`	Branch on Greater Than
`bgtu Rsrc1, Src2, label`	Branch on Greater Than Unsigned
`bgtz Rsrc, label`	Branch on Greater Than Zero
`ble Rsrc1, Src2, label`	Branch on Less Than Equal
`bleu Rsrc1, Src2, label`	Branch on LTE Unsigned
`blez Rsrc, label`	Branch on Less Than Equal Zero
`bgezal Rsrc, label`	Branch on Greater Than Equal Zero And Link
`bltzal Rsrc, label`	Branch on Less Than And Link
`blt Rsrc1, Src2, label`	Branch on Less Than
`bltu Rsrc1, Src2, label`	Branch on Less Than Unsigned
`bltz Rsrc, label`	Branch on Less Than Zero
`bne Rsrc1, Src2, label`	Branch on Not Equal
`bnez Rsrc, label`	Branch on Not Equal Zero
`j label`	Jump
`jal label`	Jump and Link
`jalr Rsrc`	Jump and Link Register
`jr Rsrc`	Jump Register

Load Instructions

`la Rdest, address`	Load Address
`lb Rdest, address`	Load Byte
`lbu Rdest, address`	Load Unsigned Byte
`ld Rdest, address`	Load Double-Word
`lh Rdest, address`	Load Halfword
`lhu Rdest, address`	Load Unsigned Halfword
`lw Rdest, address`	Load Word
`lwcz Rdest, address`	Load Word Coprocessor *z*
`lwl Rdest, address`	Load Word Left
`lwr Rdest, address`	Load Word Right
`ulh Rdest, address`	Unaligned Load Halfword
`ulhu Rdest, address`	Unaligned Load Halfword Unsigned
`ulw Rdest, address`	Unaligned Load Word

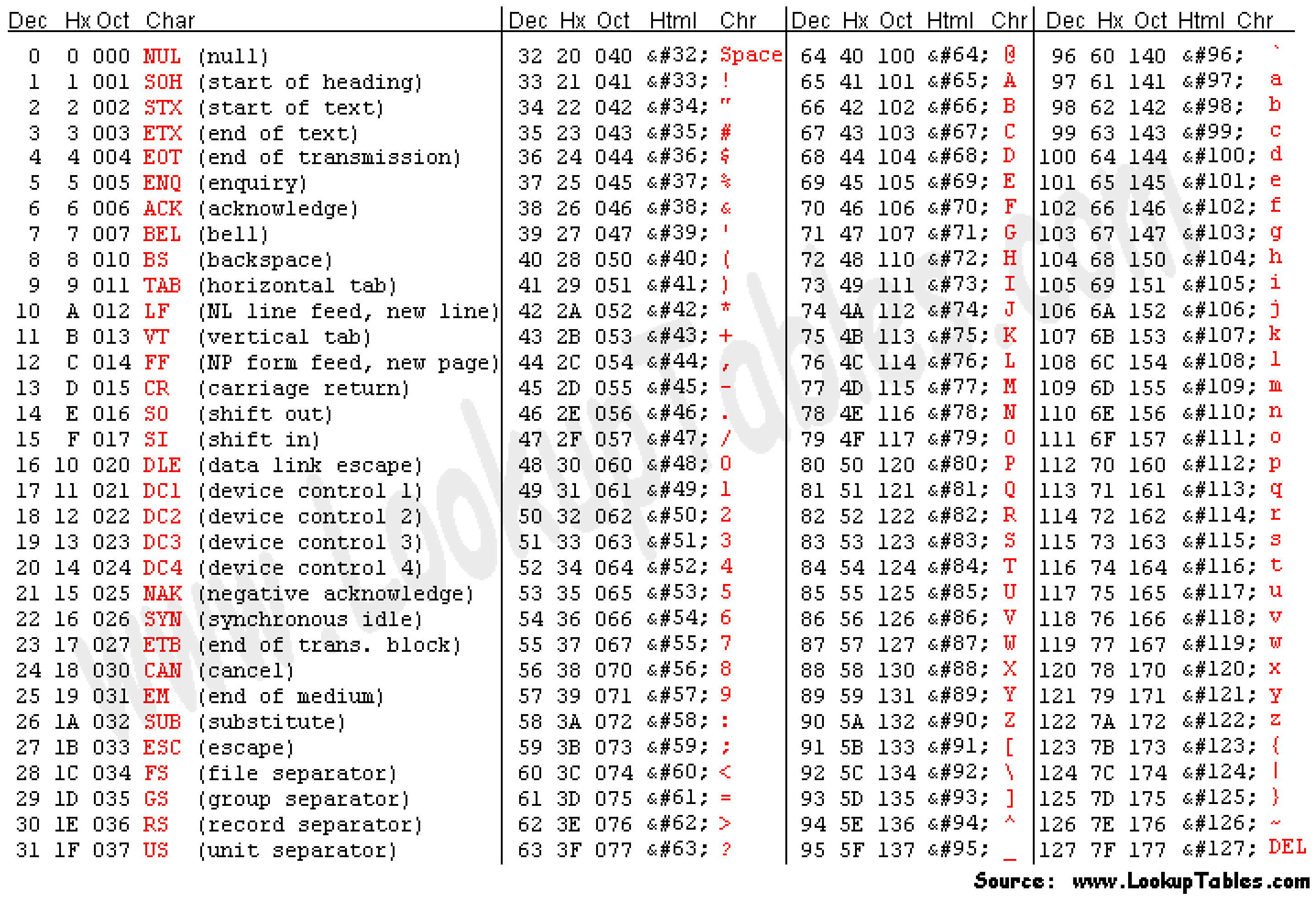
Store Instructions

`sb Rsrc, address`	Store Byte
`sd Rsrc, address`	Store Double-Word
`sh Rsrc, address`	Store Halfword
`sw Rsrc, address`	Store Word
`swcz Rsrc, address`	Store Word Coprocessor *z*
`swl Rsrc, address`	Store Word Left
`swr Rsrc, address`	Store Word Right
`ush Rsrc, address`	Unaligned Store Halfword
`usw Rsrc, address`	Unaligned Store Word

Data Movement Instructions

`move Rdest, Rsrc`	Move
`mfhi Rdest`	Move From hi
`mflo Rdest`	Move From lo
`mthi Rdest`	Move To hi
`mtlo Rdest`	Move To lo
`mfcz Rdest, CPsrc`	Move From Coprocessor z
`mfc1.d Rdest, FRsrc1`	Move Double From Coprocessor 1
`mtcz Rsrc, CPdest`	Move To Coprocessor z

ASCII Table



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com