Nombres de 0 à 15

Décimal	Hexa	Binaire	Octal	Binaire
0	0	0000	0	000
1	1	0001	1	001
2	2	0010	2	010
3	3	0011	3	011
4	4	0100	4	100
5	5	0101	5	101
6	6	0110	6	110
7	7	0111	7	111
8	8	1000	-	-
9	9	1001	-	-
10	A	1010	-	-
11	B	1011	-	-
12	C	1100	-	-
13	D	1101	-	-
14	E	1110	-	-
15	F	1111	-	-

Puissances de 2, 8 et 16

n	2^n	8^n	16^n
-4	0,0625	0,000244140625	-
-3	0,125	0,001953125	0,000244140625
-2	0,25	0,015625	0,00390625
-1	0,5	0,125	0,0625
0	1	1	1
1	2	8	16
2	4	64	256
3	8	512	4.096
4	16	4.096	65.536
5	32	32.768	1.048.576
6	64	262.144	16.777.216
7	128	2.097.152	268.435.456
8	256	16.777.216	4.294.967.296
9	512	134.217.728	68.719.476.736
10	1.024	1.073.741.824	-
11	2.048	8.589.934.592	-

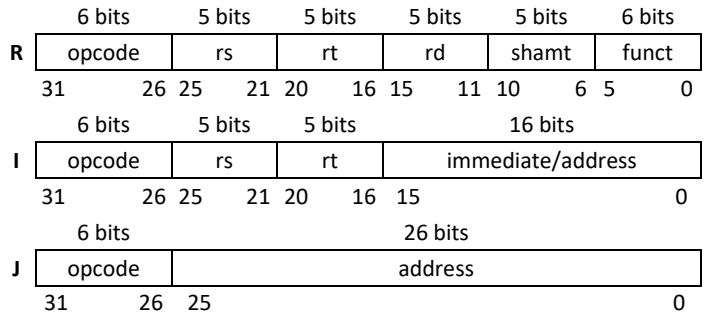
Code ASCII (7 bits) - Extrait

Code	Binaire	Car	Code	Binaire	Car
64	1000000	@	96	1100000	`
65	1000001	A	97	1100001	a
66	1000010	B	98	1100010	b
67	1000011	C	99	1100011	c
68	1000100	D	100	1100100	d
69	1000101	E	101	1100101	e
70	1000110	F	102	1100110	f
71	1000111	G	103	1100111	g
72	1001000	H	104	1101000	h
73	1001001	I	105	1101001	i
74	1001010	J	106	1101010	j
75	1001011	K	107	1101011	k
76	1001100	L	108	1101100	l
77	1001101	M	109	1101101	m
78	1001110	N	110	1101110	n
79	1001111	O	111	1101111	o
80	1010000	P	112	1110000	p
81	1010001	Q	113	1110001	q
82	1010010	R	114	1110010	r
83	1010011	S	115	1110011	s
84	1010100	T	116	1110100	t
85	1010101	U	117	1110101	u
86	1010110	V	118	1110110	v
87	1010111	W	119	1110111	w
88	1011000	X	120	1111000	x
89	1011001	Y	121	1111001	y
90	1011010	Z	122	1111010	z
91	1011011	[	123	1111011	{
92	1011100	\	124	1111100	|
93	1011101	]	125	1111101	}
94	1011110	^	126	1111110	~
95	1011111	_	127	1111111	DEL

Registres généraux du processeur MIPS

Name	Number	Use	Preserv
$zero	0	Constant value 0	NA
$v0-$v1	2-3	Function results, expr. evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved temporaries	Yes
$t8-t9	24-25	Temporaries	No
$sp	29	Stack pointer	Yes
$ra	31	Return address	Yes

Formats des instructions MIPS



Instructions et pseudo-instructions utilitaires

Instr.	Syntax	Meaning
la	`la $t1, label`	Load Address : Set $t1 to label's address
li	`li $t1, imm`	Load Immediate : Set $t1 to immediate
move	`move $t1, $t2`	Move : Set $t1 to contents of $t2
mfhi	`mfhi $t1`	Move from Hi : Set $t1 to contents of Hi
mflo	`mflo $t1`	Move from Lo : Set $t1 to contents of Lo
mthi	`mthi $t1`	Move to Hi : Set Hi to contents of $t1
mtlo	`mtlo $t1`	Move to Lo : Set Lo to contents of $t1
break	`break imm`	Terminate execution with exception code
nop	nop	Null operation
syscall	syscall	System call specified by value in $v0

Instructions de base de l'assembleur MIPS

Instr.	Syntax	Operation	Fmt	Op/Fct	Meaning
add	add rd, rs, rt	R[rd] = R[rs] + R[rt]	R	$0/20_{hex}$	Add
addi	addi rt, rs, imm	R[rt] = R[rs] + SignExtImm	I	8_{hex}	Add Immediate
addiu	addiu rt, rs, imm	R[rt] = R[rs] + SignExtImm	I	9_{hex}	Add Immediate Unsigned
addu	addu rd, rs, rt	R[rd] = R[rs] + R[rt]	R	$0/21_{hex}$	Add Unsigned
and	and rd, rs, rt	R[rd] = R[rs] & R[rt]	R	$0/24_{hex}$	And
andi	andi rt, rs, imm	R[rt] = R[rs] & ZeroExtImm	I	C_{hex}	And Immediate
beq	beq rs, rt, addr	If (R[rs] == R[rt]) PC = PC + 4 + BranchAddr	I	4_{hex}	Branch On Equal
bgtz	bgtz rs, addr	If (R[rs] > 0) PC = PC + 4 + BranchAddr	I	7_{hex}	Branch On Greater Than Zero
blez	blez rs, addr	If (R[rs] <= 0) PC = PC + 4 + BranchAddr	I	6_{hex}	Branch On Less Than Or Equal Zero
bne	bne rs, rt, addr	If (R[rs] != R[rt]) PC = PC + 4 + BranchAddr	I	5_{hex}	Branch On Not Equal
j	j addr	PC = JumpAddr	J	2_{hex}	Jump
jal	jal addr	R[31] = PC + 8 ; PC = JumpAddr	J	3_{hex}	Jump And Link
jalr	jalr rs	R[31] = PC + 8 ; PC = R[rs] ;	R	$0/09_{hex}$	Jump And Link Register
jalr	jalr rd, rs	R[rd] = PC + 8 ; PC = R[rs] ;	R	$0/09_{hex}$	Jump And Link Register
jr	jr rs	PC = R[rs]	R	$0/08_{hex}$	Jump Register
lb	lb rt, imm(rs)	R[rt] = SignExt(M[R[rs] + SignExtImm](7:0))	I	20_{hex}	Load Byte
lbu	lbu rt, imm(rs)	R[rt] = ZeroExt(M[R[rs] + SignExtImm](7:0))	I	24_{hex}	Load Byte Unsigned
lh	lh rt, imm(rs)	R[rt] = SignExt(M[R[rs] + SignExtImm](15:0))	I	21_{hex}	Load Halfword
lhu	lhu rt, imm(rs)	R[rt] = ZeroExt(M[R[rs] + SignExtImm](15:0))	I	25_{hex}	Load Halfword Unsigned
lui	lui rt, imm	R[rt] = {imm, 16'b0}	I	F_{hex}	Load Upper Immediate
lw	lw rt, imm(rs)	R[rt] = M[R[rs] + SignExtImm]	I	23_{hex}	Load Word
or	or rd, rs, rt	R[rd] = R[rs] | R[rt]	R	$0/25_{hex}$	Or
ori	ori rt, rs, imm	R[rt] = R[rs] | ZeroExtImm	I	D_{hex}	Or Immediate
nor	nor rd, rs, rt	R[rd] = ~ (R[rs] | R[rt])	R	$0/27_{hex}$	Nor
xor	xor rd, rs, rt	R[rd] = R[rs] ^ R[rt]	R	$0/26_{hex}$	Xor
xori	xori rt, rs, imm	R[rt] = R[rs] ^ ZeroExtImm	I	E_{hex}	Xor Immediate
sll	sll rd, rt, shamt	R[rd] = R[rt] << shamt	R	$0/00_{hex}$	Shift Left Logical
sllv	sllv rd, rt, rs	R[rd] = R[rt] << R[rs]	R	$0/04_{hex}$	Shift Left Logical Variable
sra	sra rd, rt, shamt	R[rd] = R[rt] >> shamt	R	$0/03_{hex}$	Shift Right Arithmetic
srav	srav rd, rt, rs	R[rd] = R[rt] >> R[rs]	R	$0/07_{hex}$	Shift Right Arithmetic Variable
srl	srl rd, rt, shamt	R[rd] = R[rt] >>> shamt	R	$0/02_{hex}$	Shift Right Logical
srlv	srlv rd, rt, rs	R[rd] = R[rt] >>> R[rs]	R	$0/06_{hex}$	Shift Right Logical Variable
slt	slt rd, rs, rt	R[rd] = (R[rs] < R[rt]) ? 1 : 0	R	$0/2A_{hex}$	Set Less Than
slti	slti rt, rs, imm	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	I	A_{hex}	Set Less Than Immediate
sltiu	sltiu rt, rs, imm	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	I	B_{hex}	Set Less Than Immediate Unsigned
sltu	sltu rd, rs, rt	R[rd] = (R[rs] < R[rt]) ? 1 : 0	R	$0/2B_{hex}$	Set Less Than Unsigned
sb	sb rt, imm(rs)	M[R[rs] + SignExtImm](7:0) = R[rt](7:0)	I	28_{hex}	Store Byte
sh	sh rt, imm(rs)	M[R[rs] + SignExtImm](15:0) = R[rt](15:0)	I	29_{hex}	Store Halfword
sw	sw rt, imm(rs)	M[R[rs] + SignExtImm] = R[rt]	I	$2B_{hex}$	Store Word
sub	sub rd, rs, rt	R[rd] = R[rs] - R[rt]	R	$0/22_{hex}$	Subtract
subu	subu rd, rs, rt	R[rd] = R[rs] - R[rt]	R	$0/23_{hex}$	Subtract Unsigned

SignExtImm = extension du signe / ZeroExtImm = ajout de zéros / (7:0) = bit 7 à 0 / 16'b0 = 16 bits à 0