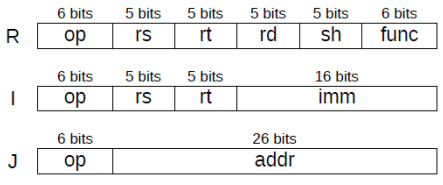
CS2640 MIPS Reference Card v0.4

Mmemonics	Operands	Instruction	Register transfer	T Op/Fn
add	rd, rs, rt	Add	rd = rs + rt	R 0/20
sub	rd, rs, rt	Subtract	rd = rs - rt	R 0/22
addi	rt, rs, imm	Add Imm.	rt = rs + imm±	I 8
addu	rd, rs, rt	Add Unsigned	rd = rs + rt	R 0/21
subu	rd, rs, rt	Subtract Unsigned	rd = rs - rt	R 0/23
addiu	rt, rs, imm	Add Imm. Unsigned	rt = rs + imm±	I 9
mult	rs, rt	Multiply	{hi, lo} = rs * rt	R 0/18
mul	rd, rs, rt	Multiply without overflow	rd = rs * rt	R c/2
div	rs, rt	Divide	lo = rs / rt; hi = rs % rt	R 0/1a
multu	rs, rt	Multiply Unsigned	{hi, lo} = rs * rt	R 0/19
mulu	rd, rs, rt	Multiply without overflow, uns	rd = rs * rt	R 0/19
divu	rs, rt	Divide Unsigned	lo = rs / rt; hi = rs % rt	R 0/1b
mfhi	rd	Move From HJ	rd = hi	R 0/10
mflo	rd	Move From LO	rd = lo	R 0/12
mthi	rs	Move to HI	hi = rs	R 0/11
mtlo	rs	Move to LO	lo = rs	R 0/13
and	rd, rs, rt	And	rd = rs & rt	R 0/24
or	rd, rs, rt	Or	rd = rs | rt	R 0/25
nor	rd, rs, rt	Nor	rd = ~(rs | rt)	R 0/27
xor	rd, rs, rt	eXclusive Or	rd = rs ^ rt	R 0/26
andi	rt, rs, imm	And Imm.	rt = rs & imm0	I c
ori	rt, rs, imm	Or Imm.	rt = rs | imm0	I d
xori	rt, rs, imm	eXclusive Or Imm.	rt = rs ^ imm0	I e
sll	rd, rt, sh	Shift Left Logical	rd = rt << sh	R 0/0
srl	rd, rt, sh	Shift Right Logical	rd = rt >>> sh	R 0/2
sra	rd, rt, sh	Shift Right Arithmetic	rd = rt >> sh	R 0/3
sllv	rd, rt, rs	Shift Left Logical Variable	rd = rt << rs	R 0/4
srlv	rd, rt, rs	Shift Right Logical Variable	rd = rt >>> rs	R 0/6
srav	rd, rt, rs	Shift Right Arithmetic Variable	rd = rt >> rs	R 0/7
slt	rd, rs, rt	Set if Less Than	rd = rs < rt ? 1 : 0	R 0/2a
sltu	rd, rs, rt	Set if Less Than Unsigned	rd = rs < rt ? 1 : 0	R 0/2b
slti	rt, rs, imm	Set if Less Than Imm.	rt = rs < imm± ? 1 : 0	I a
sltiu	rt, rs, imm	Set if Less Than Imm. Unsigned	rt = rs < imm± ? 1 : 0	I b
j	addr	Jump	PC = PC &0xF0000000 | (addr0<< 2)	J 2
jal	addr	Jump And Link	$ra = PC + 8; PC = PC&0xF0000000 | (addr0<< 2)	J 3
jr	rs	Jump Register	PC = rs	R 0/8
jalr	rs	Jump And Link Register	$ra = PC + 8; PC = rs	R 0/9
beq	rt, rs, offset	Branch if Equal	if (rs == rt) PC += 4 + (imm±<< 2)	I 4
bne	rt, rs, offset	Branch if Not Equal	if (rs != rt) PC += 4 + (imm±<< 2)	I 5
syscall		*System Call*	c0_cause = 8 << 2; c0_epc = PC; PC = 0x80000080	R 0/c
lui	rt, imm	Load Upper Imm.	rt = imm << 16	I f
lb	rt, imm(rs)	Load Byte	rt = SignExt(M1[rs + imm±])	I 20
lbu	rt, imm(rs)	Load Byte Unsigned	rt = M1[rs + imm±] & 0xFF	I 24
lh	rt, imm(rs)	Load Half	rt = SignExt(M2[rs + imm±])	I 21
lhu	rt, imm(rs)	Load Half Unsigned	rt = M2[rs + imm±] & 0xFFFF	I 25
lw	rt, imm(rs)	Load Word	rt = M4[rs + imm±]	I 23
sb	rt, imm(rs)	Store Byte	M1[rs + imm±] = rt	I 28
sh	rt, imm(rs)	Store Half	M2[rs + imm±] = rt	I 29
sw	rt, imm(rs)	Store Word	M4[rs + imm±] = rt	I 2b



registers

$0	**$zero**
$1	**$at**
$2–$3	**$v0–$v1**
$4–$7	**$a0–$a3**
$8–$15	**$t0–$t7**
$16–$23	**$s0–$s7**
$24–$25	**$t8–$t9**
$26–$27	**$k0–$k1**
$28	**$gp**
$29	**$sp**
$30	**$fp**
$31	**$ra**
hi	**–**
lo	**–**
PC	**–**
co $13	**c0_cause**
co $14	**c0_epc**

syscall codes for SPIM

1	printInt($a0)
2	printFloat($f12)
3	printDble($f12)
4	printStr($a0)
5	$v0←readInt()
6	$f0←readFloat()
7	$f0←readDble()
8	readStr($a0,$a1)
9	$v0←sbrk($a0)
10	exit
11	printChar($a0)
12	$v0←readChar()
13	$v0←open file(fn,md)
14	read file(fd,bf,len)
15	Write file(fd,bf,len)
16	close file(fd)

exception causes

0	interrupt
1	TLB protection
2	TLB miss L/F
3	TLB miss S
4	bad address L/F
5	bad address S
6	bus error F
7	bus error L/S
8	syscall
9	break
a	reserved instr.
b	coproc. unusable
c	arith. overflow

F: fetch instr.

L: load data

S: store data

Pseudo Instructions src : register or immediate value

Pseudo	Operands	Instruction	Register transfer
move	rd, rs	Move	rd = rs
li	rd,imm	Move immediate	rd = imm
la	rd, label	Load address	rd = &label
mul	rd, rs, src	Multiply *(no overflow)*	rd = rs * src
div	rd, rs, src	Divide	rd = rs / src
rem	rd, rs, src	Remainder	rd = rs % src
add	rs, rd, imm	Add immediate, *use addi*	rd = rd + imm
add	rd, imm	Add immediate	rd += imm
sub	rd, rs, src	Subtract immediate	rd = rs – imm
sub	rd imm	Subtract immediate	rd -= imm
b	offset	Branch	goto offset
beqz	rs, label	Branch on equal zero	if (rs == 0) goto label
bnez	rs, label	Branch on not equal zero	if (rs != 0) goto label
bgez	rs, label	Branch on Greater Than or Equal to Zero	if (rs >= 0) goto label
bgtz	rs, label	Branch on Greater Than Zero	if (rs > 0) goto label
blez	rs, label	Branch on Less Than or Equal to Zero	if (rs <= 0) goto label
bltz	rs, label	Branch on Less Than Zero	if (rs < 0) goto label
beq	rs, src, label	Branch on equal	if (rs == src) goto label
bne	rs, src, label	Branch on not equal	if (rs != src) goto label
bge	rs, src, label	Branch on greater than equal	if (rs >= src) goto label
bgt	rs, src, label	Branch on greater than	if (rs > src) goto label
ble	rs, src, label	Branch on less than equal	if (rs <= src) goto label
blt	rs, src, label	Branch on less than	if (rs < src) goto label
seq	rd, rs, src	Set equal	rd = rs == src ? 1 : 0
sne	rd, rs, src	Set not equal	rs = rt != src ? 1 : 0
sge	rd, rs, src	Set greater than equal	rs = rt >= src ? 1 : 0
sgt	rd, rs, src	Set greater than	rs = rt > src ? 1 : 0
sle	rd, rs, src	Set less than equal	rs = rt <= src ? 1 : 0
slt	rd, rs, src	Set less than	rs = rt < src ? 1 : 0

```
            .data
[label:]    .word   val[:count], …      # count: number of words
            .byte   val[:count], …      # count: number of bytes
            .space  count               # count: number of bytes
            .ascii  "array of char"     # without terminated \0
            .asciiz "C-string"          # \0 terminated
```