MIPS32® Instruction Set Quick Reference

R_D	— DESTINATION REGISTER
R_S, R_T	— SOURCE OPERAND REGISTERS
R_A	— RETURN ADDRESS REGISTER (R31)
PC	— PROGRAM COUNTER
ACC	— 64-BIT ACCUMULATOR
LO, HI	— ACCUMULATOR LOW ($ACC_{31:0}$) AND HIGH ($ACC_{63:32}$) PARTS
±	— SIGNED OPERAND OR SIGN EXTENSION
∅	— UNSIGNED OPERAND OR ZERO EXTENSION
::	— CONCATENATION OF BIT FIELDS
R2	— MIPS32 RELEASE 2 INSTRUCTION
DOTTED	— ASSEMBLER PSEUDO-INSTRUCTION

PLEASE REFER TO "MIPS32 ARCHITECTURE FOR PROGRAMMERS VOLUME II:
THE MIPS32 INSTRUCTION SET" FOR COMPLETE INSTRUCTION SET INFORMATION.

ARITHMETIC OPERATIONS

X	ADD	R_D, R_S, R_T	$R_D = R_S + R_T$ (OVERFLOW TRAP)
X	ADDI	$R_D, R_S, $ CONST16	$R_D = R_S + $ CONST16$^{±}$ (OVERFLOW TRAP)
X	ADDIU	$R_D, R_S, $ CONST16	$R_D = R_S + $ CONST16$^{±}$
X	ADDU	R_D, R_S, R_T	$R_D = R_S + R_T$
	CLO	R_D, R_S	$R_D = $ COUNTLEADINGONES(R_S)
	CLZ	R_D, R_S	$R_D = $ COUNTLEADINGZEROS(R_S)
	LA	$R_D, $ LABEL	$R_D = $ ADDRESS(LABEL)
	LI	$R_D, $ IMM32	$R_D = $ IMM32
X	LUI	$R_D, $ CONST16	$R_D = $ CONST16 $<< 16$ Rt=
	MOVE	R_D, R_S	$R_D = R_S$
	NEGU	R_D, R_S	$R_D = -R_S$
	SEB^{R2}	R_D, R_S	$R_D = R_{S7:0}{}^{±}$
	SEH^{R2}	R_D, R_S	$R_D = R_{S15:0}{}^{±}$
X	SUB	R_D, R_S, R_T	$R_D = R_S - R_T$ (OVERFLOW TRAP)
X	SUBU	R_D, R_S, R_T	$R_D = R_S - R_T$

SHIFT AND ROTATE OPERATIONS

	ROTR^{R2}	$R_D, R_S, $ BITS5	$R_D = R_{S BITS5-1:0} :: R_{S31:BITS5}$
	ROTRV^{R2}	R_D, R_S, R_T	$R_D = R_{S RT4:0-1:0} :: R_{S31:RT4:0}$
X	SLL	$R_D, R_S, $ SHIFT5	$R_D = R_S << $ SHIFT5 Rt << shift5
X	SLLV	R_D, R_S, R_T	$R_D = R_S << R_{T4:0}$ Rt << Rs[4:0]
	SRA	$R_D, R_S, $ SHIFT5	$R_D = R_S{}^{±} >> $ SHIFT5
	SRAV	R_D, R_S, R_T	$R_D = R_S{}^{±} >> R_{T4:0}$
	SRL	$R_D, R_S, $ SHIFT5	$R_D = R_S{}^{∅} >> $ SHIFT5
	SRLV	R_D, R_S, R_T	$R_D = R_S{}^{∅} >> R_{T4:0}$

LOGICAL AND BIT-FIELD OPERATIONS

X	AND	R_D, R_S, R_T	$R_D = R_S$ & R_T
X	ANDI	$R_D, R_S, $ CONST16	$R_D = R_S$ & CONST16$^{∅}$
	EXT^{R2}	R_D, R_S, P, S	$R_S = R_{S P+S-1:P}{}^{∅}$
	INS^{R2}	R_D, R_S, P, S	$R_{D P+S-1:P} = R_{S S-1:0}$
	NOP		NO-OP
X	NOR	R_D, R_S, R_T	$R_D = \sim(R_S | R_T)$
	NOT	R_D, R_S	$R_D = \sim R_S$
X	OR	R_D, R_S, R_T	$R_D = R_S	R_T$
X	ORI	$R_D, R_S, $ CONST16	$R_D = R_S	$ CONST16$^{∅}$
	WSBH^{R2}	R_D, R_S	$R_D = R_{S23:16} :: R_{S31:24} :: R_{S7:0} :: R_{S15:8}$
X	XOR	R_D, R_S, R_T	$R_D = R_S \oplus R_T$
X	XORI	$R_D, R_S, $ CONST16	$R_D = R_S \oplus $ CONST16$^{∅}$

CONDITION TESTING AND CONDITIONAL MOVE OPERATIONS

	MOVN	R_D, R_S, R_T	IF $R_T \neq 0$, $R_D = R_S$
	MOVZ	R_D, R_S, R_T	IF $R_T = 0$, $R_D = R_S$
X	SLT	R_D, R_S, R_T	$R_D = (R_S{}^{±} < R_T{}^{±})$? 1 : 0
X	SLTI	$R_D, R_S, $ CONST16	$R_D = (R_S{}^{±} < $ CONST16$^{±})$? 1 : 0
X	SLTIU	$R_D, R_S, $ CONST16	$R_D = (R_S{}^{∅} < $ CONST16$^{∅})$? 1 : 0
X	SLTU	R_D, R_S, R_T	$R_D = (R_S{}^{∅} < R_T{}^{∅})$? 1 : 0

MULTIPLY AND DIVIDE OPERATIONS

DIV	R_S, R_T	$L_O = R_S{}^{±} / R_T{}^{±}$; $H_I = R_S{}^{±}$ MOD $R_T{}^{±}$
DIVU	R_S, R_T	$L_O = R_S{}^{∅} / R_T{}^{∅}$; $H_I = R_S{}^{∅}$ MOD $R_T{}^{∅}$
MADD	R_S, R_T	ACC += $R_S{}^{±} \times R_T{}^{±}$
MADDU	R_S, R_T	ACC += $R_S{}^{∅} \times R_T{}^{∅}$
MSUB	R_S, R_T	ACC −= $R_S{}^{±} \times R_T{}^{±}$
MSUBU	R_S, R_T	ACC −= $R_S{}^{∅} \times R_T{}^{∅}$
MUL	R_D, R_S, R_T	$R_D = R_S{}^{±} \times R_T{}^{±}$
MULT	R_S, R_T	ACC = $R_S{}^{±} \times R_T{}^{±}$
MULTU	R_S, R_T	ACC = $R_S{}^{∅} \times R_T{}^{∅}$

ACCUMULATOR ACCESS OPERATIONS

MFHI	R_D	$R_D = H_I$
MFLO	R_D	$R_D = L_O$
MTHI	R_S	$H_I = R_S$
MTLO	R_S	$L_O = R_S$

JUMPS AND BRANCHES (NOTE: ONE DELAY SLOT)

	B	OFF18	PC += OFF18$^{±}$
	BAL	OFF18	R_A = PC + 8, PC += OFF18$^{±}$
X	BEQ	$R_S, R_T, $ OFF18	IF $R_S = R_T$, PC += OFF18$^{±}$
	BEQZ	$R_S, $ OFF18	IF $R_S = 0$, PC += OFF18$^{±}$
	BGEZ	$R_S, $ OFF18	IF $R_S \geq 0$, PC += OFF18$^{±}$
	BGEZAL	$R_S, $ OFF18	R_A = PC + 8; IF $R_S \geq 0$, PC += OFF18$^{±}$
	BGTZ	$R_S, $ OFF18	IF $R_S > 0$, PC += OFF18$^{±}$
	BLEZ	$R_S, $ OFF18	IF $R_S \leq 0$, PC += OFF18$^{±}$
	BLTZ	$R_S, $ OFF18	IF $R_S < 0$, PC += OFF18$^{±}$
	BLTZAL	$R_S, $ OFF18	R_A = PC + 8; IF $R_S < 0$, PC += OFF18$^{±}$
X	BNE	$R_S, R_T, $ OFF18	IF $R_S \neq R_T$, PC += OFF18$^{±}$
	BNEZ	$R_S, $ OFF18	IF $R_S \neq 0$, PC += OFF18$^{±}$
X	J	ADDR28	PC = PC$_{31:28} :: $ ADDR28$^{∅}$
	JAL	ADDR28	R_A = PC + 8; PC = PC$_{31:28} :: $ ADDR28$^{∅}$
	JALR	R_D, R_S	R_D = PC + 8; PC = R_S
	JR	R_S	PC = R_S

LOAD AND STORE OPERATIONS

	LB	$R_D, $ OFF16(R_S)	$R_D = $ MEM8($R_S + $ OFF16$^{±})^{±}$
	LBU	$R_D, $ OFF16(R_S)	$R_D = $ MEM8($R_S + $ OFF16$^{±})^{∅}$
	LH	$R_D, $ OFF16(R_S)	$R_D = $ MEM16($R_S + $ OFF16$^{±})^{±}$
	LHU	$R_D, $ OFF16(R_S)	$R_D = $ MEM16($R_S + $ OFF16$^{±})^{∅}$
X	LW	$R_D, $ OFF16(R_S)	$R_D = $ MEM32($R_S + $ OFF16$^{±})$
	LWL	$R_D, $ OFF16(R_S)	$R_D = $ LOADWORDLEFT($R_S + $ OFF16$^{±})$
	LWR	$R_D, $ OFF16(R_S)	$R_D = $ LOADWORDRIGHT($R_S + $ OFF16$^{±})$
	SB	$R_S, $ OFF16(R_T)	MEM8($R_T + $ OFF16$^{±}) = R_{S7:0}$
	SH	$R_S, $ OFF16(R_T)	MEM16($R_T + $ OFF16$^{±}) = R_{S15:0}$
X	SW	$R_S, $ OFF16(R_T)	MEM32($R_T + $ OFF16$^{±}) = R_S$
	SWL	$R_S, $ OFF16(R_T)	STOREWORDLEFT($R_T + $ OFF16$^{±}$, R_S)
	SWR	$R_S, $ OFF16(R_T)	STOREWORDRIGHT($R_T + $ OFF16$^{±}$, R_S)
	ULW	$R_D, $ OFF16(R_S)	$R_D = $ UNALIGNED_MEM32($R_S + $ OFF16$^{±})$
	USW	$R_S, $ OFF16(R_T)	UNALIGNED_MEM32($R_T + $ OFF16$^{±}) = R_S$

ATOMIC READ-MODIFY-WRITE OPERATIONS

LL	$R_D, $ OFF16(R_S)	$R_D = $ MEM32($R_S + $ OFF16$^{±})$; LINK
SC	$R_D, $ OFF16(R_S)	IF ATOMIC, MEM32($R_S + $ OFF16$^{±}) = R_D$; R_D = ATOMIC ? 1 : 0

MIPS

MD00565 Revision 01.01

Registers

0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call

Default C Calling Convention (O32)

Stack Management
- The stack grows down.
  - Subtract from $sp to allocate local storage space.
  - Restore $sp by adding the same amount at function exit.
- The stack must be 8-byte aligned.
  - Modify $sp only in multiples of eight.

Function Parameters
- Every parameter smaller than 32 bits is promoted to 32 bits.
- First four parameters are passed in registers $a0−$a3.
  - 64-bit parameters are passed in register pairs:
    - Little-endian mode: $a1:$a0 or $a3:$a2.
    - Big-endian mode: $a0:$a1 or $a2:$a3.
- Every subsequent parameter is passed through the stack.
  - First 16 bytes on the stack are not used.
  - Assuming $sp was not modified at function entry:
    - The 1st stack parameter is located at 16($sp).
    - The 2nd stack parameter is located at 20($sp), etc.
  - 64-bit parameters are 8-byte aligned.

Return Values
- 32-bit and smaller values are returned in register $v0.
- 64-bit values are returned in registers $v0 and $v1:
  - Little-endian mode: $v1:$v0.
  - Big-endian mode: $v0:$v1.

MIPS32 Virtual Address Space

kseg3	0xE000.0000	0xFFFF.FFFF	Mapped	Cached
ksseg	0xC000.0000	0xDFFF.FFFF	Mapped	Cached
kseg1	0xA000.0000	0xBFFF.FFFF	Unmapped	Uncached
kseg0	0x8000.0000	0x9FFF.FFFF	Unmapped	Cached
useg	0x0000.0000	0x7FFF.FFFF	Mapped	Cached

Reading The Cycle Count Register From C

```
unsigned mips_cycle_counter_read()
{
    unsigned cc;
    asm volatile("mfc0 %0, $9" : "=r" (cc));
    return (cc << 1);
}
```

Assembly-Language Function Example

```
# int asm_max(int a, int b)
# {
#   int r = (a < b) ? b : a;
#   return r;
# }

    .text
    .set    nomacro
    .set    noreorder

    .global asm_max
    .ent    asm_max

asm_max:
    move    $v0, $a0        # r = a
    slt     $t0, $a0, $a1   # a < b ?
    jr      $ra             # return
    movn    $v0, $a1, $t0   # if yes, r = b

    .end    asm_max
```

C / Assembly-Language Function Interface

```
#include <stdio.h>

int asm_max(int a, int b);

int main()
{
    int x = asm_max(10, 100);
    int y = asm_max(200, 20);
    printf("%d %d\n", x, y);
}
```

Invoking MULT And MADD Instructions From C

```
int dp(int a[], int b[], int n)
{
    int i;
    long long acc = (long long) a[0] * b[0];
    for (i = 1; i < n; i++)
        acc += (long long) a[i] * b[i];
    return (acc >> 31);
}
```

Atomic Read-Modify-Write Example

```
atomic_inc:
    ll      $t0, 0($a0)     # load linked
    addiu   $t1, $t0, 1     # increment
    sc      $t1, 0($a0)     # store cond'l
    beqz    $t1, atomic_inc # loop if failed
    nop
```

Accessing Unaligned Data
note: ULW and USW automatically generate appropriate code

Little-Endian Mode		Big-Endian Mode	
LWR	R_D, off16(R_S)	LWL	R_D, off16(R_S)
LWL	R_D, off16+3(R_S)	LWR	R_D, off16+3(R_S)
SWR	R_D, off16(R_S)	SWL	R_D, off16(R_S)
SWL	R_D, off16+3(R_S)	SWR	R_D, off16+3(R_S)

Accessing Unaligned Data From C

```
typedef struct
{
    int u;
} __attribute__((packed)) unaligned;


int unaligned_load(void *ptr)
{
    unaligned *uptr = (unaligned *)ptr;
    return uptr->u;
}
```

MIPS SDE-GCC Compiler Defines

__mips	MIPS ISA (= 32 for MIPS32)
__mips_isa_rev	MIPS ISA Revision (= 2 for MIPS32 R2)
__mips_dsp	DSP ASE extensions enabled
_MIPSEB	Big-endian target CPU
_MIPSEL	Little-endian target CPU
_MIPS_ARCH_CPU	Target CPU specified by −march=CPU
_MIPS_TUNE_CPU	Pipeline tuning selected by −mtune=CPU

Notes
- Many assembler pseudo-instructions and some rarely used machine instructions are omitted.
- The C calling convention is simplified. Additional rules apply when passing complex data structures as function parameters.
- The examples illustrate syntax used by GCC compilers.
- Most MIPS processors increment the cycle counter every other cycle. Please check your processor documentation.