MIPS32® Instruction Set Quick Reference

Rd	—	Destination register
Rs, Rt	—	Source operand registers
Ra	—	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	Rd, Rs, Rt	Rd = Rs + Rt (overflow trap)
X	ADDI	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$ (overflow trap)
X	ADDIU	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$
X	ADDU	Rd, Rs, Rt	Rd = Rs + Rt
	CLO	Rd, Rs	Rd = CountLeadingOnes(Rs)
	CLZ	Rd, Rs	Rd = CountLeadingZeros(Rs)
	LA	Rd, label	Rd = Address(label)
	LI	Rd, imm32	Rd = imm32
X	LUI	Rd, const16	Rd = const16 << 16 Rt=
	MOVE	Rd, Rs	Rd = Rs
	NEGU	Rd, Rs	Rd = –Rs
	SEB^{R2}	Rd, Rs	Rd = $Rs_{7:0}^{\pm}$
	SEH^{R2}	Rd, Rs	Rd = $Rs_{15:0}^{\pm}$
X	SUB	Rd, Rs, Rt	Rd = Rs – Rt (overflow trap)
X	SUBU	Rd, Rs, Rt	Rd = Rs – Rt

Shift and Rotate Operations

	ROTR^{R2}	Rd, Rs, bits5	Rd = $Rs_{BITS5-1:0}$ :: $Rs_{31:BITS5}$
	ROTRV^{R2}	Rd, Rs, Rt	Rd = $Rs_{RT4:0-1:0}$ :: $Rs_{31:RT4:0}$
X	SLL	Rd, Rs, shift5	Rd = Rs << shift5 Rt << shift5
X	SLLV	Rd, Rs, Rt	Rd = Rs << $Rt_{4:0}$ Rt << Rs[4:0]
X	SRA	Rd, Rs, shift5	Rd = Rs$^{\pm}$ >> shift5
X	SRAV	Rd, Rs, Rt	Rd = Rs$^{\pm}$ >> $Rt_{4:0}$
X	SRL	Rd, Rs, shift5	Rd = Rs$^{\varnothing}$ >> shift5
X	SRLV	Rd, Rs, Rt	Rd = Rs$^{\varnothing}$ >> $Rt_{4:0}$

Logical and Bit-Field Operations

X	AND	Rd, Rs, Rt	Rd = Rs & Rt
X	ANDI	Rd, Rs, const16	Rd = Rs & const16$^{\varnothing}$
	EXT^{R2}	Rd, Rs, P, S	Rs = $Rs_{P+S-1:P}^{\varnothing}$
	INS^{R2}	Rd, Rs, P, S	$Rd_{P+S-1:P}$ = $Rs_{S-1:0}$
	NOP		No-op
X	NOR	Rd, Rs, Rt	Rd = ~(Rs | Rt)
	NOT	Rd, Rs	Rd = ~Rs
X	OR	Rd, Rs, Rt	Rd = Rs | Rt
X	ORI	Rd, Rs, const16	Rd = Rs | const16$^{\varnothing}$
	WSBH^{R2}	Rd, Rs	Rd = $Rs_{23:16}$ :: $Rs_{31:24}$ :: $Rs_{7:0}$ :: $Rs_{15:8}$
X	XOR	Rd, Rs, Rt	Rd = Rs ⊕ Rt
X	XORI	Rd, Rs, const16	Rd = Rs ⊕ const16$^{\varnothing}$

Condition Testing and Conditional Move Operations

X	MOVN	Rd, Rs, Rt	if Rt ≠ 0, Rd = Rs
X	MOVZ	Rd, Rs, Rt	if Rt = 0, Rd = Rs
X	SLT	Rd, Rs, Rt	Rd = (Rs$^{\pm}$ < Rt$^{\pm}$) ? 1 : 0
X	SLTI	Rd, Rs, const16	Rd = (Rs$^{\pm}$ < const16$^{\pm}$) ? 1 : 0
X	SLTIU	Rd, Rs, const16	Rd = (Rs$^{\varnothing}$ < const16$^{\varnothing}$) ? 1 : 0
X	SLTU	Rd, Rs, Rt	Rd = (Rs$^{\varnothing}$ < Rt$^{\varnothing}$) ? 1 : 0

Multiply and Divide Operations

DIV	Rs, Rt	Lo = Rs$^{\pm}$ / Rt$^{\pm}$; Hi = Rs$^{\pm}$ mod Rt$^{\pm}$
DIVU	Rs, Rt	Lo = Rs$^{\varnothing}$ / Rt$^{\varnothing}$; Hi = Rs$^{\varnothing}$ mod Rt$^{\varnothing}$
MADD	Rs, Rt	Acc += Rs$^{\pm}$ × Rt$^{\pm}$
MADDU	Rs, Rt	Acc += Rs$^{\varnothing}$ × Rt$^{\varnothing}$
MSUB	Rs, Rt	Acc −= Rs$^{\pm}$ × Rt$^{\pm}$
MSUBU	Rs, Rt	Acc −= Rs$^{\varnothing}$ × Rt$^{\varnothing}$
MUL	Rd, Rs, Rt	Rd = Rs$^{\pm}$ × Rt$^{\pm}$
MULT	Rs, Rt	Acc = Rs$^{\pm}$ × Rt$^{\pm}$
MULTU	Rs, Rt	Acc = Rs$^{\varnothing}$ × Rt$^{\varnothing}$

Accumulator Access Operations

MFHI	Rd	Rd = Hi
MFLO	Rd	Rd = Lo
MTHI	Rs	Hi = Rs
MTLO	Rs	Lo = Rs

Jumps And Branches (Note: One Delay Slot)

	B	off18	PC += off18$^{\pm}$
	BAL	off18	Ra = PC + 8, PC += off18$^{\pm}$
X	BEQ	Rs, Rt, off18	if Rs = Rt, PC += off18$^{\pm}$
	BEQZ	Rs, off18	if Rs = 0, PC += off18$^{\pm}$
X	BGEZ	Rs, off18	if Rs ≥ 0, PC += off18$^{\pm}$
X	BGEZAL	Rs, off18	Ra = PC + 8; if Rs ≥ 0, PC += off18$^{\pm}$
X	BGTZ	Rs, off18	if Rs > 0, PC += off18$^{\pm}$
X	BLEZ	Rs, off18	if Rs ≤ 0, PC += off18$^{\pm}$
X	BLTZ	Rs, off18	if Rs < 0, PC += off18$^{\pm}$
X	BLTZAL	Rs, off18	Ra = PC + 8; if Rs < 0, PC += off18$^{\pm}$
X	BNE	Rs, Rt, off18	if Rs ≠ Rt, PC += off18$^{\pm}$
	BNEZ	Rs, off18	if Rs ≠ 0, PC += off18$^{\pm}$
X	J	addr28	PC = $PC_{31:28}$ :: addr28$^{\varnothing}$
X	JAL	addr28	Ra = PC + 8; PC = $PC_{31:28}$ :: addr28$^{\varnothing}$
X	JALR	Rd, Rs	Rd = PC + 8; PC = Rs
X	JR	Rs	PC = Rs

Load and Store Operations

	LB	Rd, off16(Rs)	Rd = mem8(Rs + off16$^{\pm}$)$^{\pm}$
	LBU	Rd, off16(Rs)	Rd = mem8(Rs + off16$^{\pm}$)$^{\varnothing}$
	LH	Rd, off16(Rs)	Rd = mem16(Rs + off16$^{\pm}$)$^{\pm}$
	LHU	Rd, off16(Rs)	Rd = mem16(Rs + off16$^{\pm}$)$^{\varnothing}$
X	LW	Rd, off16(Rs)	Rd = mem32(Rs + off16$^{\pm}$)
	LWL	Rd, off16(Rs)	Rd = LoadWordLeft(Rs + off16$^{\pm}$)
	LWR	Rd, off16(Rs)	Rd = LoadWordRight(Rs + off16$^{\pm}$)
	SB	Rs, off16(Rt)	mem8(Rt + off16$^{\pm}$) = $Rs_{7:0}$
	SH	Rs, off16(Rt)	mem16(Rt + off16$^{\pm}$) = $Rs_{15:0}$
X	SW	Rs, off16(Rt)	mem32(Rt + off16$^{\pm}$) = Rs
	SWL	Rs, off16(Rt)	StoreWordLeft(Rt + off16$^{\pm}$, Rs)
	SWR	Rs, off16(Rt)	StoreWordRight(Rt + off16$^{\pm}$, Rs)
	ULW	Rd, off16(Rs)	Rd = unaligned_mem32(Rs + off16$^{\pm}$)
	USW	Rs, off16(Rt)	unaligned_mem32(Rt + off16$^{\pm}$) = Rs

Atomic Read-Modify-Write Operations

LL	Rd, off16(Rs)	Rd = mem32(Rs + off16$^{\pm}$); link
SC	Rd, off16(Rs)	if Atomic, mem32(Rs + off16$^{\pm}$) = Rd; Rd = 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	Rd, off16(Rs)	LWL	Rd, off16(Rs)
LWL	Rd, off16+3(Rs)	LWR	Rd, off16+3(Rs)
SWR	Rd, off16(Rs)	SWL	Rd, off16(Rs)
SWL	Rd, off16+3(Rs)	SWR	Rd, off16+3(Rs)

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.