

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

ADD	Rd, Rs, Rt	$Rd = Rs + Rt$ (OVERFLOW TRAP)
ADDI	Rd, Rs, CONST16	$Rd = Rs + CONST16^{\pm}$ (OVERFLOW TRAP)
ADDIU	Rd, Rs, CONST16	$Rd = Rs + CONST16^{\pm}$
ADDU	Rd, Rs, Rt	$Rd = Rs + Rt$
CLO	Rd, Rs	$Rd = \text{COUNTLEADINGONES}(Rs)$
CLZ	Rd, Rs	$Rd = \text{COUNTLEADINGZEROS}(Rs)$
LA	Rd, LABEL	$Rd = \text{ADDRESS}(\text{LABEL})$
LI	Rd, IMM32	$Rd = \text{IMM32}$
LUI	Rd, CONST16	$Rd = \text{CONST16} \ll 16$
MOVE	Rd, Rs	$Rd = Rs$
NEGU	Rd, Rs	$Rd = -Rs$
SEB ^{R2}	Rd, Rs	$Rd = Rs_{7:0}^{\pm}$
SEH ^{R2}	Rd, Rs	$Rd = Rs_{15:0}^{\pm}$
SUB	Rd, Rs, Rt	$Rd = Rs - Rt$ (OVERFLOW TRAP)
SUBU	Rd, Rs, Rt	$Rd = Rs - Rt$

SHIFT AND ROTATE OPERATIONS

ROTR ^{R2}	Rd, Rs, BITS5	$Rd = Rs_{\text{BITS5-1:0}} :: Rs_{31:\text{BITS5}}$
ROTRV ^{R2}	Rd, Rs, Rt	$Rd = Rs_{\text{RT40-1:0}} :: Rs_{31:\text{RT40}}$
SLL	Rd, Rs, SHIFT5	$Rd = Rs \ll \text{SHIFT5}$
SLLV	Rd, Rs, Rt	$Rd = Rs \ll Rt_{40}$
SRA	Rd, Rs, SHIFT5	$Rd = Rs^{\pm} \gg \text{SHIFT5}$
SRAV	Rd, Rs, Rt	$Rd = Rs^{\pm} \gg Rt_{40}$
SRL	Rd, Rs, SHIFT5	$Rd = Rs^{\emptyset} \gg \text{SHIFT5}$
SRLV	Rd, Rs, Rt	$Rd = Rs^{\emptyset} \gg Rt_{40}$

LOGICAL AND BIT-FIELD OPERATIONS

AND	Rd, Rs, Rt	$Rd = Rs \& Rt$
ANDI	Rd, Rs, CONST16	$Rd = Rs \& \text{CONST16}^{\emptyset}$
EXT ^{R2}	Rd, Rs, P, S	$Rs = Rs_{\text{SP+S-1:P}}^{\emptyset}$
INS ^{R2}	Rd, Rs, P, S	$Rd_{\text{P+S-1:P}} = Rs_{S-1:0}$
NOP		No-OP
NOR	Rd, Rs, Rt	$Rd = \sim(Rs Rt)$
NOT	Rd, Rs	$Rd = \sim Rs$
OR	Rd, Rs, Rt	$Rd = Rs Rt$
ORI	Rd, Rs, CONST16	$Rd = Rs \text{CONST16}^{\emptyset}$
WSBH ^{R2}	Rd, Rs	$Rd = Rs_{23:16} :: Rs_{31:24} :: Rs_{7:0} :: Rs_{15:8}$
XOR	Rd, Rs, Rt	$Rd = Rs \oplus Rt$
XORI	Rd, Rs, CONST16	$Rd = Rs \oplus \text{CONST16}^{\emptyset}$

CONDITION TESTING AND CONDITIONAL MOVE OPERATIONS

MOVN	Rd, Rs, Rt	IF $Rt \neq 0$, $Rd = Rs$
MOVZ	Rd, Rs, Rt	IF $Rt = 0$, $Rd = Rs$
SLT	Rd, Rs, Rt	$Rd = (Rs^{\pm} < Rt^{\pm}) ? 1 : 0$
SLTI	Rd, Rs, CONST16	$Rd = (Rs^{\pm} < \text{CONST16}^{\pm}) ? 1 : 0$
SLTIU	Rd, Rs, CONST16	$Rd = (Rs^{\emptyset} < \text{CONST16}^{\emptyset}) ? 1 : 0$
SLTU	Rd, Rs, Rt	$Rd = (Rs^{\emptyset} < Rt^{\emptyset}) ? 1 : 0$

MULTIPLY AND DIVIDE OPERATIONS

DIV	Rs, Rt	$Lo = Rs^{\pm} / Rt^{\pm}$; $Hi = Rs^{\pm} \text{MOD } Rt^{\pm}$
DIVU	Rs, Rt	$Lo = Rs^{\emptyset} / Rt^{\emptyset}$; $Hi = Rs^{\emptyset} \text{MOD } Rt^{\emptyset}$
MADD	Rs, Rt	$ACC += Rs^{\pm} \times Rt^{\pm}$
MADDU	Rs, Rt	$ACC += Rs^{\emptyset} \times Rt^{\emptyset}$
MSUB	Rs, Rt	$ACC -= Rs^{\pm} \times Rt^{\pm}$
MSUBU	Rs, Rt	$ACC -= Rs^{\emptyset} \times Rt^{\emptyset}$
MUL	Rd, Rs, Rt	$Rd = Rs^{\pm} \times Rt^{\pm}$
MULT	Rs, Rt	$ACC = Rs^{\pm} \times Rt^{\pm}$
MULTU	Rs, Rt	$ACC = Rs^{\emptyset} \times Rt^{\emptyset}$

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	$PC += \text{OFF18}^{\pm}$
BAL	OFF18	$RA = PC + 8$, $PC += \text{OFF18}^{\pm}$
BEQ	Rs, Rt, OFF18	IF $Rs = Rt$, $PC += \text{OFF18}^{\pm}$
BEQZ	Rs, OFF18	IF $Rs = 0$, $PC += \text{OFF18}^{\pm}$
BGEZ	Rs, OFF18	IF $Rs \geq 0$, $PC += \text{OFF18}^{\pm}$
BGEZAL	Rs, OFF18	$RA = PC + 8$; IF $Rs \geq 0$, $PC += \text{OFF18}^{\pm}$
BGTZ	Rs, OFF18	IF $Rs > 0$, $PC += \text{OFF18}^{\pm}$
BLEZ	Rs, OFF18	IF $Rs \leq 0$, $PC += \text{OFF18}^{\pm}$
BLTZ	Rs, OFF18	IF $Rs < 0$, $PC += \text{OFF18}^{\pm}$
BLTZAL	Rs, OFF18	$RA = PC + 8$; IF $Rs < 0$, $PC += \text{OFF18}^{\pm}$
BNE	Rs, Rt, OFF18	IF $Rs \neq Rt$, $PC += \text{OFF18}^{\pm}$
BNEZ	Rs, OFF18	IF $Rs \neq 0$, $PC += \text{OFF18}^{\pm}$
J	ADDR28	$PC = PC_{31:28} :: \text{ADDR28}^{\emptyset}$
JAL	ADDR28	$RA = PC + 8$; $PC = PC_{31:28} :: \text{ADDR28}^{\emptyset}$
JALR	Rd, Rs	$Rd = PC + 8$; $PC = Rs$
JR	Rs	$PC = Rs$

LOAD AND STORE OPERATIONS

LB	Rd, OFF16(Rs)	$Rd = \text{MEM8}(Rs + \text{OFF16}^{\pm})^{\pm}$
LBU	Rd, OFF16(Rs)	$Rd = \text{MEM8}(Rs + \text{OFF16}^{\pm})^{\emptyset}$
LH	Rd, OFF16(Rs)	$Rd = \text{MEM16}(Rs + \text{OFF16}^{\pm})^{\pm}$
LHU	Rd, OFF16(Rs)	$Rd = \text{MEM16}(Rs + \text{OFF16}^{\pm})^{\emptyset}$
LW	Rd, OFF16(Rs)	$Rd = \text{MEM32}(Rs + \text{OFF16}^{\pm})$
LWL	Rd, OFF16(Rs)	$Rd = \text{LOADWORDLEFT}(Rs + \text{OFF16}^{\pm})$
LWR	Rd, OFF16(Rs)	$Rd = \text{LOADWORDRIGHT}(Rs + \text{OFF16}^{\pm})$
SB	Rs, OFF16(Rt)	$\text{MEM8}(Rt + \text{OFF16}^{\pm}) = Rs_{7:0}$
SH	Rs, OFF16(Rt)	$\text{MEM16}(Rt + \text{OFF16}^{\pm}) = Rs_{15:0}$
SW	Rs, OFF16(Rt)	$\text{MEM32}(Rt + \text{OFF16}^{\pm}) = Rs$
SWL	Rs, OFF16(Rt)	$\text{STOREWORDLEFT}(Rt + \text{OFF16}^{\pm}, Rs)$
SWR	Rs, OFF16(Rt)	$\text{STOREWORDRIGHT}(Rt + \text{OFF16}^{\pm}, Rs)$
ULW	Rd, OFF16(Rs)	$Rd = \text{UNALIGNED_MEM32}(Rs + \text{OFF16}^{\pm})$
USW	Rs, OFF16(Rt)	$\text{UNALIGNED_MEM32}(Rt + \text{OFF16}^{\pm}) = Rs$

ATOMIC READ-MODIFY-WRITE OPERATIONS

LL	Rd, OFF16(Rs)	$Rd = \text{MEM32}(Rs + \text{OFF16}^{\pm})$; LINK
SC	Rd, OFF16(Rs)	IF ATOMIC, $\text{MEM32}(Rs + \text{OFF16}^{\pm}) = Rd$; $Rd = \text{ATOMIC} ? 1 : 0$

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("mfcc0 %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'1
    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.