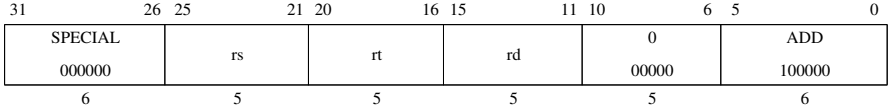
## Add Word

**ADD**




**Format:** `ADD rd, rs, rt` **MIPS32**

**Purpose:**

To add 32-bit integers. If an overflow occurs, then trap.

**Description:** `GPR[rd] ← GPR[rs] + GPR[rt]`

The 32-bit word value in GPR *rt* is added to the 32-bit value in GPR *rs* to produce a 32-bit result.

- If the addition results in 32-bit 2's complement arithmetic overflow, the destination register is not modified and an Integer Overflow exception occurs.
- If the addition does not overflow, the 32-bit result is placed into GPR *rd*.

**Restrictions:**

None

**Operation:**

```
temp ← (GPR[rs]31||GPR[rs]31..0) + (GPR[rt]31||GPR[rt]31..0)
if temp32 ≠ temp31 then
   SignalException(IntegerOverflow)
else
   GPR[rd] ← temp
endif
```

**Exceptions:**

Integer Overflow

**Programming Notes:**

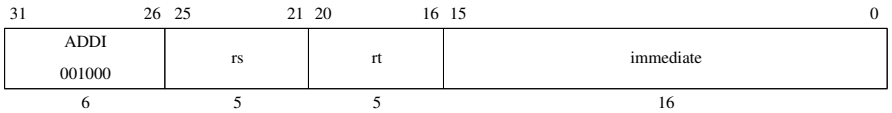
ADDU performs the same arithmetic operation but does not trap on overflow.





## Add Immediate Word

**ADDI**




**Format:** `ADDI rt, rs, immediate` **MIPS32**

**Purpose:**

To add a constant to a 32-bit integer. If overflow occurs, then trap**.**

**Description:** `GPR[rt] ¨ GPR[rs] + immediate`

The 16-bit signed *immediate* is added to the 32-bit value in GPR *rs* to produce a 32-bit result.

- If the addition results in 32-bit 2's complement arithmetic overflow, the destination register is not modified and an Integer Overflow exception occurs.
- If the addition does not overflow, the 32-bit result is placed into GPR *rt*.

**Restrictions:**

None

**Operation:**

```
temp ← (GPR[rs]31||GPR[rs]31..0) + sign_extend(immediate)
if temp32 ≠ temp31 then
   SignalException(IntegerOverflow)
else
   GPR[rt] ← temp
endif
```

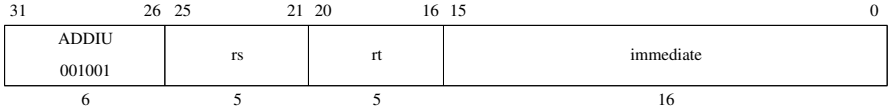
**Exceptions:**

Integer Overflow

**Programming Notes:**

ADDIU performs the same arithmetic operation but does not trap on overflow.

## Add Immediate Untrapped Word — ADDIU




**Format:** `ADDIU rt, rs, immediate` **MIPS32**

**Purpose:**

To add a constant to a 32-bit integer

**Description:** `GPR[rt] ← GPR[rs] + immediate`

The 16-bit signed *immediate* is added to the 32-bit value in GPR *rs* and the 32-bit arithmetic result is placed into GPR *rt*.

No Integer Overflow exception occurs under any circumstances.

**Restrictions:**

None

**Operation:**

```
temp ← GPR[rs] + sign_extend(immediate)
GPR[rt]← temp
```

**Exceptions:**

None

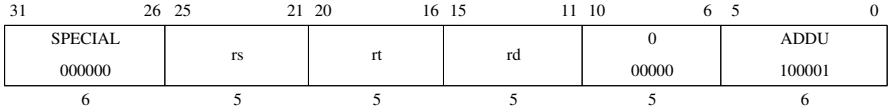
**Programming Notes:**

The term “unsigned” in the instruction name is a misnomer; this operation is 32-bit modulo arithmetic that does not trap on overflow. This instruction is appropriate for unsigned arithmetic, such as address arithmetic, or integer arithmetic environments that ignore overflow, such as C language arithmetic.





## Add Untrapped Word — ADDU




**Format:** `ADDU rd, rs, rt` **MIPS32**

**Purpose:**

To add 32-bit integers

**Description:** `GPR[rd] ← GPR[rs] + GPR[rt]`

The 32-bit word value in GPR *rt* is added to the 32-bit value in GPR *rs* and the 32-bit arithmetic result is placed into GPR *rd*.

No Integer Overflow exception occurs under any circumstances.

**Restrictions:**

None

**Operation:**

```
temp ← GPR[rs] + GPR[rt]
GPR[rd] ← temp
```

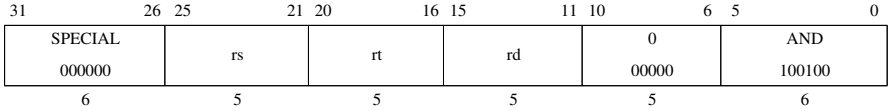
**Exceptions:**

None

**Programming Notes:**

The term “unsigned” in the instruction name is a misnomer; this operation is 32-bit modulo arithmetic that does not trap on overflow. This instruction is appropriate for unsigned arithmetic, such as address arithmetic, or integer arithmetic environments that ignore overflow, such as C language arithmetic.

## And AND




**Format:** `AND rd, rs, rt` **MIPS32**

**Purpose:**

To do a bitwise logical AND

**Description:** `GPR[rd] ← GPR[rs] AND GPR[rt]`

The contents of GPR *rs* are combined with the contents of GPR *rt* in a bitwise logical AND operation. The result is placed into GPR *rd*.

**Restrictions:**

None

**Operation:**

```
GPR[rd] ← GPR[rs] and GPR[rt]
```

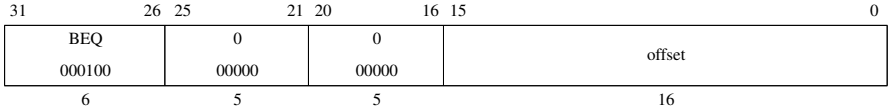
**Exceptions:**

None





## Unconditional Branch B




**Format:** `B offset` **Assembly Idiom**

**Purpose:**

To do an unconditional branch

**Description:** `branch`

B offset is the assembly idiom used to denote an unconditional branch. The actual instruction is interpreted by the hardware as BEQ r0, r0, offset.

An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

**Restrictions:**

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

**Operation:**

```
I:     target_offset ← sign_extend(offset || 0^2)
I+1:   PC ← PC + target_offset
```

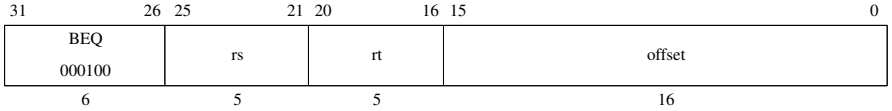
**Exceptions:**

None

**Programming Notes:**

With the 18-bit signed instruction offset, the conditional branch range is ± 128 Kbytes. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

## Branch on Equal — BEQ




**Format:** `BEQ rs, rt, offset` **MIPS32**

**Purpose:**

To compare GPRs then do a PC-relative conditional branch

**Description:** `if GPR[rs] = GPR[rt] then branch`

An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If the contents of GPR *rs* and GPR *rt* are equal, branch to the effective target address after the instruction in the delay slot is executed.

**Restrictions:**

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

**Operation:**

```
I:     target_offset ← sign_extend(offset || 0^2)
          condition ← (GPR[rs] = GPR[rt])
I+1:   if condition then
          PC ← PC + target_offset
       endif
```

**Exceptions:**

None

**Programming Notes:**

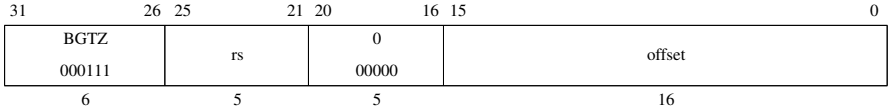
With the 18-bit signed instruction offset, the conditional branch range is ± 128 Kbytes. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

BEQ r0, r0 offset, expressed as B offset, is the assembly idiom used to denote an unconditional branch.





## Branch on Greater Than Zero — BGTZ




**Format:** `BGTZ rs, offset` **MIPS32**

**Purpose:**

To test a GPR then do a PC-relative conditional branch

**Description:** `if GPR[rs] > 0 then branch`

An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If the contents of GPR *rs* are greater than zero (sign bit is 0 but value not zero), branch to the effective target address after the instruction in the delay slot is executed.

**Restrictions:**

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

**Operation:**

```
I:     target_offset ← sign_extend(offset || 0^2)
       condition ← GPR[rs] > 0^GPRLEN
I+1:   if condition then
          PC ← PC + target_offset
       endif
```

**Exceptions:**

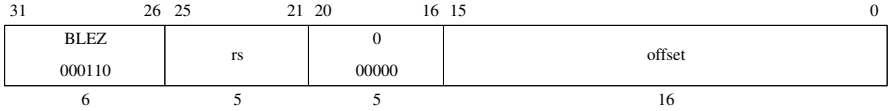
None

**Programming Notes:**

With the 18-bit signed instruction offset, the conditional branch range is ± 128 KBytes. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

## Branch on Less Than or Equal to Zero

**BLEZ**




**Format:** `BLEZ rs, offset` **MIPS32**

**Purpose:**

To test a GPR then do a PC-relative conditional branch

**Description:** `if GPR[rs] ≤ 0 then branch`

An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If the contents of GPR *rs* are less than or equal to zero (sign bit is 1 or value is zero), branch to the effective target address after the instruction in the delay slot is executed.

**Restrictions:**

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

**Operation:**

```
I:      target_offset ← sign_extend(offset || 0^2)
        condition ← GPR[rs] ≤ 0^GPRLEN
I+1:    if condition then
            PC ← PC + target_offset
        endif
```

**Exceptions:**

None

**Programming Notes:**

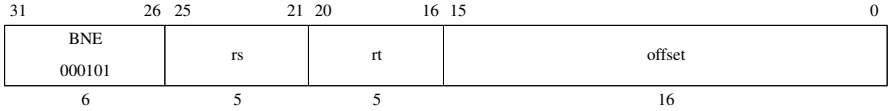
With the 18-bit signed instruction offset, the conditional branch range is ± 128 KBytes. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.





## Branch on Not Equal

**BNE**




**Format:** `BNE rs, rt, offset` **MIPS32**

**Purpose:**

To compare GPRs then do a PC-relative conditional branch

**Description:** `if GPR[rs] ≠ GPR[rt] then branch`

An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If the contents of GPR *rs* and GPR *rt* are not equal, branch to the effective target address after the instruction in the delay slot is executed.

**Restrictions:**

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

**Operation:**

```
I:      target_offset ← sign_extend(offset || 0^2)
        condition ← (GPR[rs] ≠ GPR[rt])
I+1:    if condition then
            PC ← PC + target_offset
        endif
```

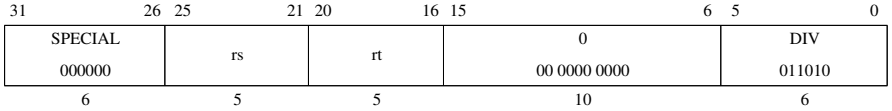
**Exceptions:**

None

**Programming Notes:**

With the 18-bit signed instruction offset, the conditional branch range is ± 128 KBytes. Use jump (J) or jump register (JR) instructions to branch to addresses outside this range.

## Divide Word — DIV




**Format:** `DIV rs, rt` **MIPS32**

**Purpose:**

To divide a 32-bit signed integers

**Description:** `(HI, LO) ← GPR[rs] / GPR[rt]`

The 32-bit word value in GPR *rs* is divided by the 32-bit value in GPR *rt,* treating both operands as signed values. The 32-bit quotient is placed into special register *LO* and the 32-bit remainder isplaced into special register *HI*.

No arithmetic exception occurs under any circumstances.

**Restrictions:**

If the divisor in GPR *rt* is zero, the arithmetic result value is **UNPREDICTABLE**.

**Operation:**

```
q  ← GPR[rs]31..0 div GPR[rt]31..0
LO ← q
r  ← GPR[rs]31..0 mod GPR[rt]31..0
HI ← r
```

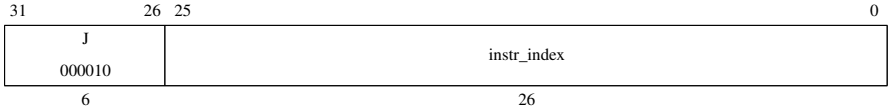
**Exceptions:**

None





## Jump — J




**Format:** `J target` **MIPS32**

**Purpose:**

To branch within the current 256 MB-aligned region

**Description:**

This is a PC-region branch (not PC-relative); the effective target address is in the “current” 256 MB-aligned region. The low 28 bits of the target address is the *instr_index* field shifted left 2 bits. The remaining upper bits are the corresponding bits of the address of the instruction in the delay slot (not the branch itself).

Jump to the effective target address. Execute the instruction that follows the jump, in the branch delay slot, before executing the jump itself.

**Restrictions:**

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

**Operation:**

```
I:
I+1:PC ← PCGPRLEN-1..28 || instr_index || 0^2
```

**Exceptions:**

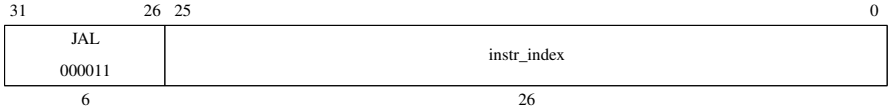
None

**Programming Notes:**

Forming the branch target address by catenating PC and index bits rather than adding a signed offset to the PC is an advantage if all program code addresses fit into a 256 MB region aligned on a 256 MB boundary. It allows a branch from anywhere in the region to anywhere in the region, an action not allowed by a signed relative offset.

This definition creates the following boundary case: When the jump instruction is in the last word of a 256 MB region, it can branch only to the following 256 MB region containing the branch delay slot.

## Jump and Link JAL




**Format:** `JAL target` **MIPS32**

**Purpose:**

To execute a procedure call within the current 256 MB-aligned region

**Description:**

Place the return address link in GPR 31. The return link is the address of the second instruction following the branch, at which location execution continues after a procedure call.

This is a PC-region branch (not PC-relative); the effective target address is in the "current" 256 MB-aligned region. The low 28 bits of the target address is the *instr_index* field shifted left 2 bits. The remaining upper bits are the corresponding bits of the address of the instruction in the delay slot (not the branch itself).

Jump to the effective target address. Execute the instruction that follows the jump, in the branch delay slot, before executing the jump itself.

**Restrictions:**

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

**Operation:**

```
I:  GPR[31]← PC + 8
I+1:PC     ← PC_GPRLEN-1..28 || instr_index || 0^2
```

**Exceptions:**

None

**Programming Notes:**

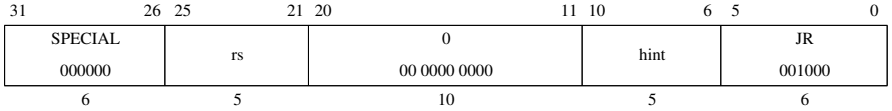
Forming the branch target address by catenating PC and index bits rather than adding a signed offset to the PC is an advantage if all program code addresses fit into a 256 MB region aligned on a 256 MB boundary. It allows a branch from anywhere in the region to anywhere in the region, an action not allowed by a signed relative offset.

This definition creates the following boundary case: When the branch instruction is in the last word of a 256 MB region, it can branch only to the following 256 MB region containing the branch delay slot.





## Jump Register JR




**Format:** `JR rs` **MIPS32**

**Purpose:**

To execute a branch to an instruction address in a register

**Description:** `PC ← GPR[rs]`

Jump to the effective target address in GPR *rs*. Execute the instruction following the jump, in the branch delay slot, before jumping.

For processors that implement the MIPS16e ASE, set the *ISA Mode* bit to the value in GPR *rs* bit 0. Bit 0 of the target address is always zero so that no Address Exceptions occur when bit 0 of the source register is one

**Restrictions:**

The effective target address in GPR *rs* must be naturally-aligned. For processors that do not implement the MIPS16e ASE, if either of the two least-significant bits are not zero, an Address Error exception occurs when the branch target is subsequently fetched as an instruction. For processors that do implement the MIPS16e ASE, if bit 0 is zero and bit 1 is one, an Address Error exception occurs when the jump target is subsequently fetched as an instruction.

In release 1 of the architecture, the only defined hint field value is 0, which sets default handling of JR. In Release 2 of the architecture, bit 10 of the hint field is used to encode an instruction hazard barrier. See the JR.HB instruction description for additional information.

Processor operation is **UNPREDICTABLE** if a branch, jump, ERET, DERET, or WAIT instruction is placed in the delay slot of a branch or jump.

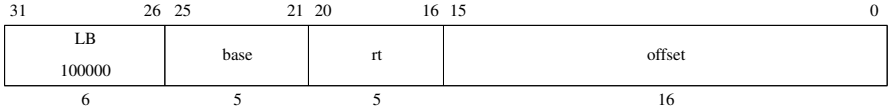
**Operation:**

```
I:  temp ← GPR[rs]
I+1:if Config1_CA = 0 then
       PC ← temp
    else
       PC ← temp_GPRLEN-1..1 || 0
       ISAMode ← temp_0
    endif
```

**Exceptions:**

None

## Load Byte LB




**Format:** `LB rt, offset(base)` **MIPS32**

**Purpose:**

To load a byte from memory as a signed value

**Description:** `GPR[rt] ← memory[GPR[base] + offset]`

The contents of the 8-bit byte at the memory location specified by the effective address are fetched, sign-extended, and placed in GPR *rt*. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

**Restrictions:**

None

**Operation:**

```
vAddr  ← sign_extend(offset) + GPR[base]
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
pAddr  ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor ReverseEndian^2)
memword← LoadMemory (CCA, BYTE, pAddr, vAddr, DATA)
byte   ← vAddr_1..0 xor BigEndianCPU^2
GPR[rt]← sign_extend(memword_7+8*byte..8*byte)
```

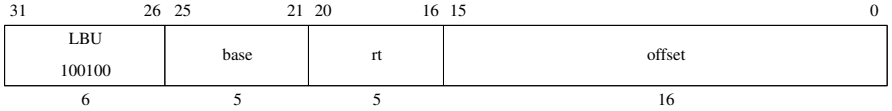
**Exceptions:**

TLB Refill, TLB Invalid, Address Error, Watch





## Load Byte Unsigned LBU




**Format:** `LBU rt, offset(base)` **MIPS32**

**Purpose:**

To load a byte from memory as an unsigned value

**Description:** `GPR[rt] ← memory[GPR[base] + offset]`

The contents of the 8-bit byte at the memory location specified by the effective address are fetched, zero-extended, and placed in GPR *rt*. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

**Restrictions:**

None

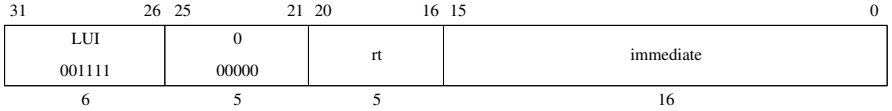
**Operation:**

```
vAddr  ← sign_extend(offset) + GPR[base]
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
pAddr  ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor ReverseEndian^2)
memword← LoadMemory (CCA, BYTE, pAddr, vAddr, DATA)
byte   ← vAddr_1..0 xor BigEndianCPU^2
GPR[rt]← zero_extend(memword_7+8*byte..8*byte)
```

**Exceptions:**

TLB Refill, TLB Invalid, Address Error, Watch

## Load Upper Immediate — LUI




**Format:** `LUI rt, immediate` **MIPS32**

**Purpose:**

To load a constant into the upper half of a word

**Description:** `GPR[rt] ← immediate || 0^16`

The 16-bit *immediate* is shifted left 16 bits and concatenated with 16 bits of low-order zeros. The 32-bit result is placed into GPR *rt*.

**Restrictions:**

None

**Operation:**

```
GPR[rt] ← immediate || 0^16
```

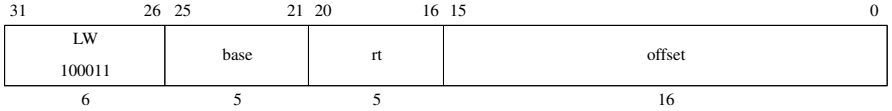
**Exceptions:**

None





## Load Word — LW




**Format:** `LW rt, offset(base)` **MIPS32**

**Purpose:**

To load a word from memory as a signed value

**Description:** `GPR[rt] ← memory[GPR[base] + offset]`

The contents of the 32-bit word at the memory location specified by the aligned effective address are fetched, sign-extended to the GPR register length if necessary, and placed in GPR *rt*. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

**Restrictions:**

The effective address must be naturally-aligned. If either of the 2 least-significant bits of the address is non-zero, an Address Error exception occurs.

**Operation:**

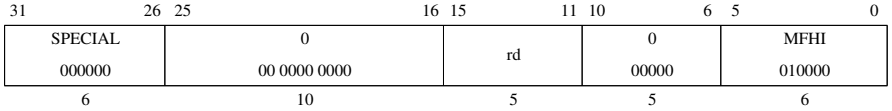
```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr_1..0 ≠ 0^2 then
   SignalException(AddressError)
endif
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
memword← LoadMemory (CCA, WORD, pAddr, vAddr, DATA)
GPR[rt]← memword
```

**Exceptions:**

TLB Refill, TLB Invalid, Bus Error, Address Error, Watch

## Move From HI Register

**MFHI**




**Format:** `MFHI rd` **MIPS32**

**Purpose:**

To copy the special purpose *HI* register to a GPR

**Description:** `GPR[rd] ← HI`

The contents of special register *HI* are loaded into GPR *rd*.

**Restrictions:**

None

**Operation:**

```
GPR[rd] ← HI
```

**Exceptions:**

None

**Historical Information:**

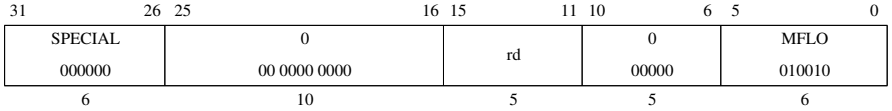
In the MIPS I, II, and III architectures, the two instructions which follow the MFHI must not moodify the HI register. If this restriction is violated, the result of the MFHI is **UNPREDICTABLE**. This restriction was removed in MIPS IV and MIPS32, and all subsequent levels of the architecture.





## Move From LO Register

**MFLO**




**Format:** `MFLO rd` **MIPS32**

**Purpose:**

To copy the special purpose *LO* register to a GPR

**Description:** `GPR[rd] ← LO`

The contents of special register *LO* are loaded into GPR *rd*.

**Restrictions: None**

**Operation:**

```
GPR[rd] ← LO
```

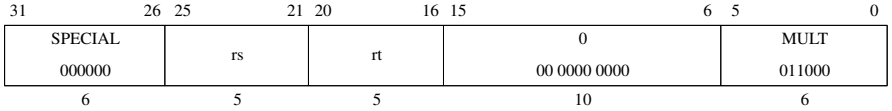
**Exceptions:**

None

**Historical Information:**

In the MIPS I, II, and III architectures, the two instructions which follow the MFHI must not moodify the HI register. If this restriction is violated, the result of the MFHI is **UNPREDICTABLE**. This restriction was removed in MIPS IV and MIPS32, and all subsequent levels of the architecture.

## Multiply Word — MULT




**Format:** `MULT rs, rt` **MIPS32**

**Purpose:**

To multiply 32-bit signed integers

**Description:** `(HI, LO) ← GPR[rs] ×GPR[rt]`

The 32-bit word value in GPR *rt* is multiplied by the 32-bit value in GPR *rs*, treating both operands as signed values, to produce a 64-bit result. The low-order 32-bit word of the result is placed into special register *LO*, and the high-order 32-bit word is splaced into special register *HI*.

No arithmetic exception occurs under any circumstances.

**Restrictions:**

**None**

**Operation:**

```
prod   ← GPR[rs]31..0 × GPR[rt]31..0
LO     ← prod31..0
HI     ← prod63..32
```

**Exceptions:**

None

**Programming Notes:**

In some processors the integer multiply operation may proceed asynchronously and allow other CPU instructions to execute before it is complete. An attempt to read *LO* or *HI* before the results are written interlocks until the results are ready. Asynchronous execution does not affect the program result, but offers an opportunity for performance improvement by scheduling the multiply so that other instructions can execute in parallel.

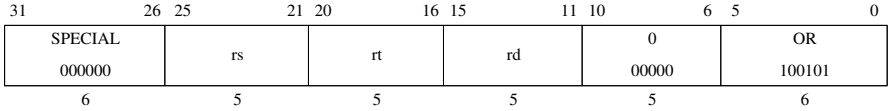
Programs that require overflow detection must check for it explicitly.

Where the size of the operands are known, software should place the shorter operand in GPR *rt*. This may reduce the latency of the instruction on those processors which implement data-dependent instruction latencies.





## Or — OR




**Format:** `OR rd, rs, rt` **MIPS32**

**Purpose:**

To do a bitwise logical OR

**Description:** `GPR[rd] ← GPR[rs] or GPR[rt]`

The contents of GPR *rs* are combined with the contents of GPR *rt* in a bitwise logical OR operation. The result is placed into GPR *rd*.

**Restrictions:**

None

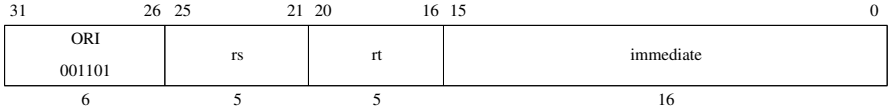
**Operation:**

```
GPR[rd] ← GPR[rs] or GPR[rt]
```

**Exceptions:**

None

## Or Immediate — ORI




**Format:** `ORI rt, rs, immediate` **MIPS32**

**Purpose:**

To do a bitwise logical OR with a constant

**Description:** `GPR[rt] ← GPR[rs] or immediate`

The 16-bit *immediate* is zero-extended to the left and combined with the contents of GPR *rs* in a bitwise logical OR operation. The result is placed into GPR *rt*.

**Restrictions:**

None

**Operation:**

```
GPR[rt] ← GPR[rs] or zero_extend(immediate)
```

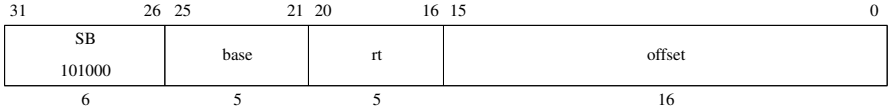
**Exceptions:**

None





## Store Byte — SB




**Format:** `SB rt, offset(base)` **MIPS32**

**Purpose:**

To store a byte to memory

**Description:** `memory[GPR[base] + offset] ← GPR[rt]`

The least-significant 8-bit byte of GPR *rt* is stored in memory at the location specified by the effective address. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

**Restrictions:**

None

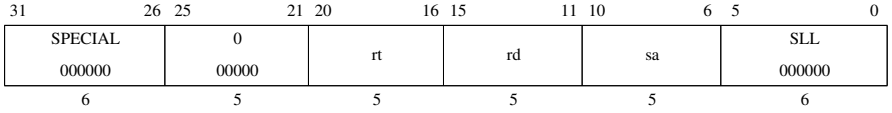
**Operation:**

```
vAddr     ← sign_extend(offset) + GPR[base]
(pAddr, CCA)← AddressTranslation (vAddr, DATA, STORE)
pAddr     ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor ReverseEndian^2)
bytesel   ← vAddr_1..0 xor BigEndianCPU^2
dataword  ← GPR[rt]_31-8*bytesel..0 || 0^8*bytesel
StoreMemory (CCA, BYTE, dataword, pAddr, vAddr, DATA)
```

**Exceptions:**

TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error, Watch

## Shift Word Left Logical — SLL




**Format:** `SLL rd, rt, sa` **MIPS32**

**Purpose:**

To left-shift a word by a fixed number of bits

**Description:** `GPR[rd] ← GPR[rt] << sa`

The contents of the low-order 32-bit word of GPR *rt* are shifted left, inserting zeros into the emptied bits; the word result is placed in GPR *rd*. The bit-shift amount is specified by *sa.*

**Restrictions:**

None

**Operation:**

```
s       ← sa
temp    ← GPR[rt](31-s)..0 || 0^s
GPR[rd]← temp
```

**Exceptions:**

None

**Programming Notes:**

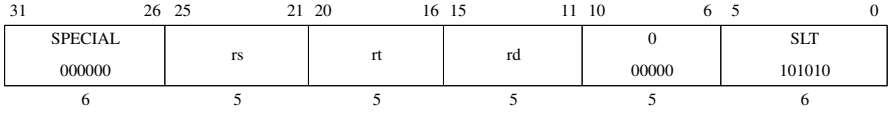
SLL r0, r0, 0, expressed as NOP, is the assembly idiom used to denote no operation.

SLL r0, r0, 1, expressed as SSNOP, is the assembly idiom used to denote no operation that causes an issue break on superscalar processors.





## Set on Less Than — SLT




**Format:** `SLT rd, rs, rt` **MIPS32**

**Purpose:**

To record the result of a less-than comparison

**Description:** `GPR[rd] ← (GPR[rs] < GPR[rt])`

Compare the contents of GPR *rs* and GPR *rt* as signed integers and record the Boolean result of the comparison in GPR *rd*. If GPR *rs* is less than GPR *rt,* the result is 1 (true); otherwise, it is 0 (false).

The arithmetic comparison does not cause an Integer Overflow exception.

**Restrictions:**

None

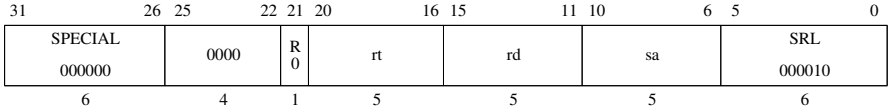
**Operation:**

```
if GPR[rs] < GPR[rt] then
   GPR[rd] ← 0^(GPRLEN-1) || 1
else
   GPR[rd] ← 0^GPRLEN
endif
```

**Exceptions:**

None

## Shift Word Right Logical — SRL




**Format:** `SRL rd, rt, sa` **MIPS32**

**Purpose:**

To execute a logical right-shift of a word by a fixed number of bits

**Description:** `GPR[rd] ← GPR[rt] >> sa (logical)`

The contents of the low-order 32-bit word of GPR *rt* are shifted right, inserting zeros into the emptied bits; the word result is placed in GPR *rd*. The bit-shift amount is specified by *sa.*

**Restrictions:**

**None**

**Operation:**

```
s      ← sa
temp   ← 0^s || GPR[rt]31..s
GPR[rd]← temp
```

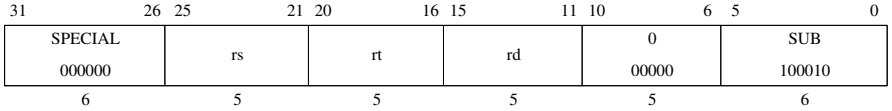
**Exceptions:**

None





## Subtract Word — SUB




**Format:** `SUB rd, rs, rt` **MIPS32**

**Purpose:**

To subtract 32-bit integers. If overflow occurs, then trap

**Description:** `GPR[rd] ← GPR[rs] - GPR[rt]`

The 32-bit word value in GPR *rt* is subtracted from the 32-bit value in GPR *rs* to produce a 32-bit result. If the subtraction results in 32-bit 2's complement arithmetic overflow, then the destination register is not modified and an Integer Overflow exception occurs. If it does not overflow, the 32-bit result is placed into GPR *rd*.

**Restrictions:**

**None**

**Operation:**

```
temp ← (GPR[rs]31||GPR[rs]31..0) – (GPR[rt]31||GPR[rt]31..0)
if temp32 ≠ temp31 then
   SignalException(IntegerOverflow)
else
   GPR[rd] ← temp31..0
endif
```

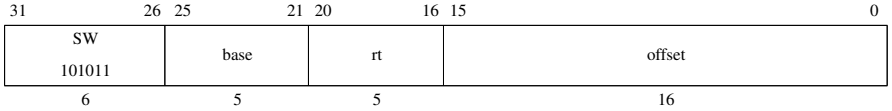
**Exceptions:**

Integer Overflow

**Programming Notes:**

SUBU performs the same arithmetic operation but does not trap on overflow.

## Store Word SW




**Format:** `SW rt, offset(base)` **MIPS32**

**Purpose:**

To store a word to memory

**Description:** `memory[GPR[base] + offset] ← GPR[rt]`

The least-significant 32-bit word of GPR *rt* is stored in memory at the location specified by the aligned effective address. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

**Restrictions:**

The effective address must be naturally-aligned. If either of the 2 least-significant bits of the address is non-zero, an Address Error exception occurs.

**Operation:**

```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 0^2 then
   SignalException(AddressError)
endif
(pAddr, CCA)← AddressTranslation (vAddr, DATA, STORE)
dataword← GPR[rt]
StoreMemory (CCA, WORD, dataword, pAddr, vAddr, DATA)
```

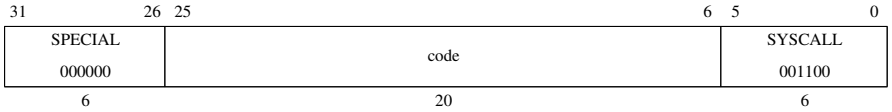
**Exceptions:**

TLB Refill, TLB Invalid, TLB Modified, Address Error, Watch





## System Call SYSCALL




**Format:** `SYSCALL` **MIPS32**

**Purpose:**

To cause a System Call exception

**Description:**

A system call exception occurs, immediately and unconditionally transferring control to the exception handler.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

**Restrictions:**

None

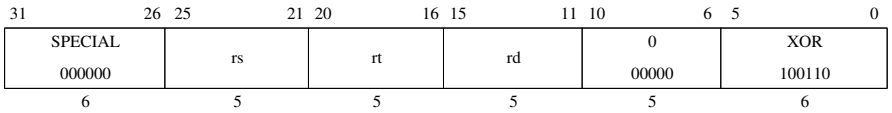
**Operation:**

```
SignalException(SystemCall)
```

**Exceptions:**

System Call

## Exclusive OR XOR




**Format:** `XOR rd, rs, rt` **MIPS32**

**Purpose:**

To do a bitwise logical Exclusive OR

**Description:** `GPR[rd] ← GPR[rs] XOR GPR[rt]`

Combine the contents of GPR *rs* and GPR *rt* in a bitwise logical Exclusive OR operation and place the result into GPR *rd*.

**Restrictions:**

None

**Operation:**

```
GPR[rd] ← GPR[rs] xor GPR[rt]
```

**Exceptions:**

None