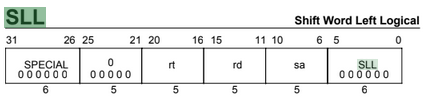
Instrucciones:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R-type | OpCode(5-0) | I-type | OpCode (31-26) | J-type | funct (5-0) |
| SLL | [000000] | LB | [100000] | JR | [001000] |
| SRL | [000010] | LH | [100001] | JALR | [001001] |
| SRA | [000011] | LW | [100011] |  |  |
| SLLV | [000100] | LWU | [100111] |  |  |
| SRLV | [000110] | LBU | [100100] |  |  |
| SRAV | [000111] | LHU | [100101] |  |  |
| ADDU | [100001] | SB | [101000] |  |  |
| SUBU | [100011] | SH | [101011] |  |  |
| AND | [100100] | SW | [101011] |  |  |
| OR | [100101] | ADDI | [001000] |  |  |
| XOR | [100110] | ANDI | [001100] |  |  |
| NOR | [100111] | ORI | [001101] |  |  |
| SLT | [101010] | XORI | [001110] |  |  |
|  |  | LUI | [001111] |  |  |
|  |  | SLTI | [001010] |  |  |
|  |  | BEQ | [000100] |  |  |
|  |  | BNE | [000101] |  |  |
|  |  | J | [000010] |  |  |
|  |  | JAL | [000011] |  |  |

Descripción:

R-type

SLL (Shift Word Left Logical):

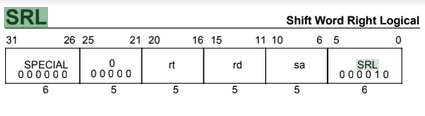


Formato: SLL rd, rt, sa

Propósito: Mover hacia la izquierda una palabra (rt) una cantidad de bits (sa) y guardarlo en otro registro(rd)

Descripción: rd ← rt << sa

SRL (Shift Word Right Logical):

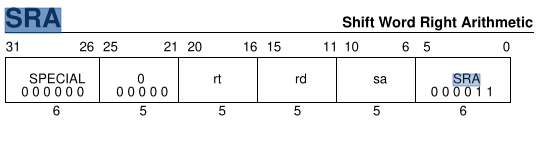


Formato: SRL rd, rt, sa

Propósito: Desplazar hacia la derecha (logical) una palabra una cantidad de bits (sa) y guardarlo en un registro (rd).

Descripción: rd ← rt >> sa

SRA (Shift Word Right Arithmetic):

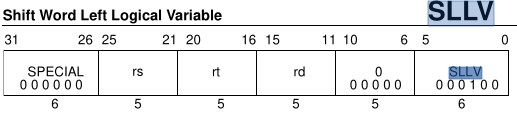


Format: SRA rd, rt, sa

Purpose: To arithmetic right shift a word by a fixed number of bits.

Description: rd ← rt >> sa

SLLV (Shift Word Left Logical Variable):



Format: SLLV rd, rt, rs

Purpose: To left shift a word by a variable number of bits.

Description: rd ← rt << rs

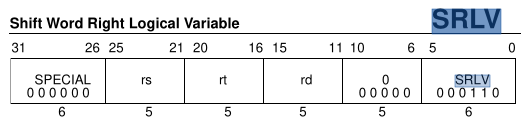
The contents of the low-order 32-bit word of GPR rt are shifted left, inserting zeroes

into the emptied bits; the result word is placed in GPR rd. The bit shift count is

specified by the low-order five bits of GPR rs. If rd is a 64-bit register, the result word

is sign-extended.

SRLV (Shift Word Right Logical Variable):



Format: SRLV rd, rt, rs

Purpose: To logical right shift a word by a variable number of bits.

Description: rd ← rt >> rs

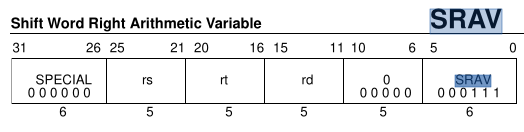
The contents of the low-order 32-bit word of GPR rt are shifted right, inserting zeros

into the emptied bits; the word result is placed in GPR rd. The bit shift count is

specified by the low-order five bits of GPR rs. If rd is a 64-bit register, the result word

is sign-extended.

SRAV (Shift Word Right Arithmetic Variable):



Format: SRAV rd, rt, rs

Purpose: To arithmetic right shift a word by a variable number of bits.

Description: rd ← rt >> rs

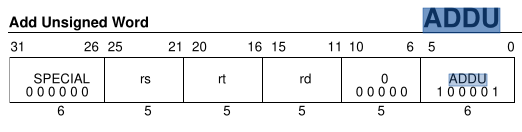
The contents of the low-order 32-bit word of GPR rt are shifted right, duplicating the

sign-bit (bit 31) in the emptied bits; the word result is placed in GPR rd. The bit shift

count is specified by the low-order five bits of GPR rs. If rd is a 64-bit register, the result

word is sign-extended.

ADDU (Add Unsigned Word):

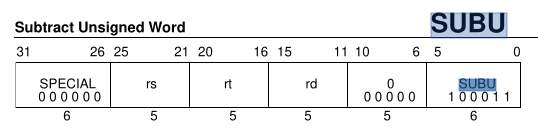


Format: ADDU rd, rs, rt

Purpose: To add 32-bit integers.

Description: rd ← rs + rt

SUBU (Subtract Unsigned Word):

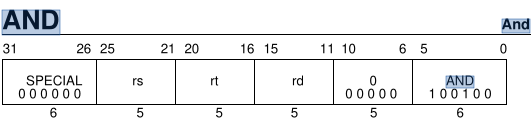


Format: SUBU rd, rs, rt

Purpose: To subtract 32-bit integers.

Description: rd ← rs - rt

AND:

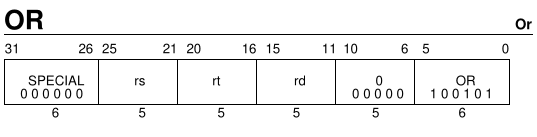


Format: AND rd, rs, rt

Purpose: To do a bitwise logical AND.

Description: rd ← rs AND rt

OR:

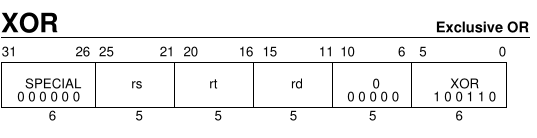


Format: OR rd, rs, rt

Purpose: To do a bitwise logical OR.

Description: rd ← rs OR rt

XOR (Exclusive Or):

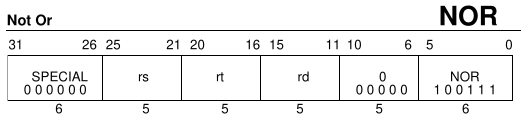


Format: XOR rd, rs, rt

Purpose: To do a bitwise logical EXCLUSIVE OR.

Description: rd ← rs XOR rt

NOR:

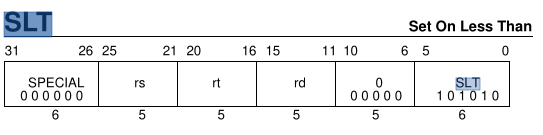


Format: NOR rd, rs, rt

Purpose: To do a bitwise logical NOT OR.

Description: rd ← rs NOR rt

SLT (Set on Less Than):



Format: SLT rd, rs, rt

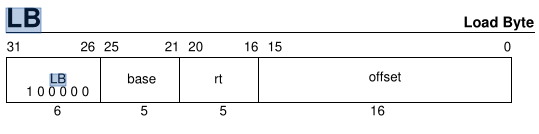
Purpose: To record the result of a less-than (<) comparison.

Description: rd ← (rs < 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 0 (false).

I-Type

LB (Load Byte):



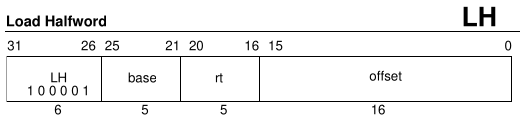
Format: LB rt, offset(base)

Purpose: To load a byte from memory as a signed value.

Description: rt ← memory[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.

LH (Load Halfword):



Format: LH rt, offset(base)

Purpose: To load a halfword from memory as a signed value.

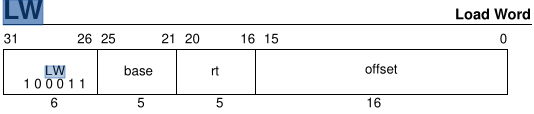
MIPS I

Description: rt ← memory[base+offset]

The contents of the 16-bit halfword at the memory location specified by the aligned

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.

LW (Load Word):



Format: LW rt, offset(base)

Purpose: To load a word from memory as a signed value.

Description: rt ← memory[base+offset]

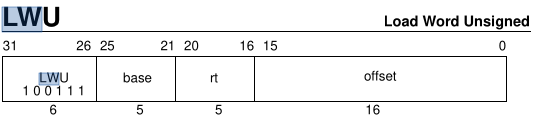
The contents of the 32-bit word at the memory location specified by the aligned

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.

LWU (Load Word Unsigned):



Format: LWU rt, offset(base)

Purpose: To load a word from memory as an unsigned value.

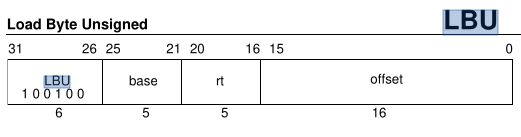
Description: rt ← memory[base+offset]

The contents of the 32-bit word at the memory location specified by the aligned

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.

LBU (Load Byte Unsigned):



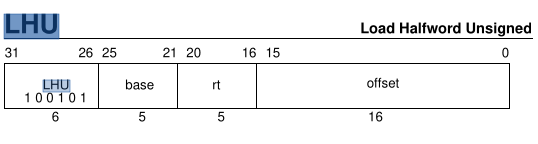
Format: LBU rt, offset(base)

Purpose: To load a byte from memory as an unsigned value.

Description: rt ← memory[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.

LHU (Load Halfword Unsigned):



Format: LHU rt, offset(base)

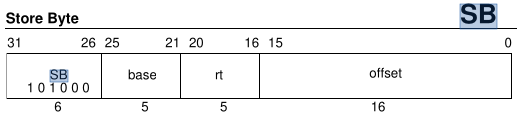
Purpose: To load a halfword from memory as an unsigned value.

Description: rt ← memory[base+offset]

The contents of the 16-bit halfword at the memory location specified by the aligned

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.

SB (Store Byte):



Format: SB rt, offset(base)

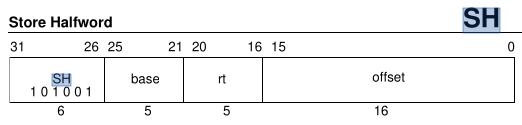
Purpose: To store a byte to memory.

Description: memory[base+offset] ← 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.

SH (Store Halfword):



Format: SH rt, offset(base)

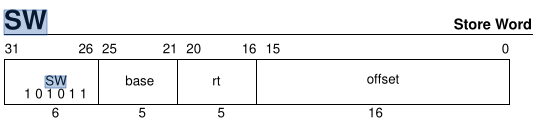
Purpose: To store a halfword to memory.

Description: memory[base+offset] ← rt

The least-significant 16-bit halfword of register 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.

SW (Store Word):



Format: SW rt, offset(base)

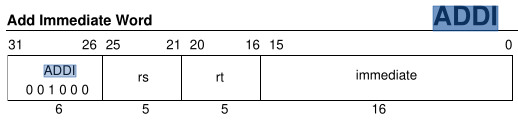
Purpose: To store a word to memory.

Description: memory[base+offset] ← rt

The least-significant 32-bit word of register 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.

ADDI (Add Immediate Word):



Format: ADDI rt, rs, immediate

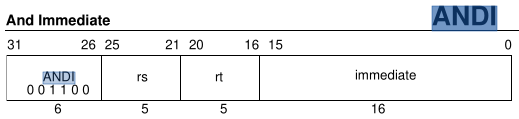
Purpose: To add a constant to a 32-bit integer. If overflow occurs, then trap.

Description: rt ← 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 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 rt.

ANDI (And Immediate):



Format: ANDI rt, rs, immediate

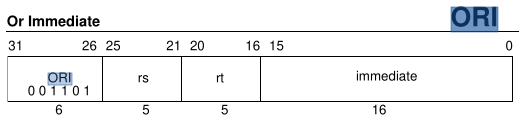
Purpose: To do a bitwise logical AND with a constant.

Description: rt ← rs AND immediate

The 16-bit immediate is zero-extended to the left and combined with the contents of

GPR rs in a bitwise logical AND operation. The result is placed into GPR rt.

ORI (Or Immediate):



Format: ORI rt, rs, immediate

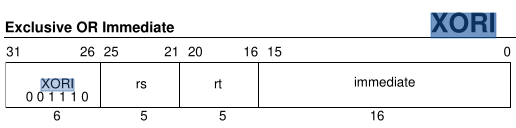
Purpose: To do a bitwise logical OR with a constant.

Description: rd ← 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.

XORI (Exclusive Or Immediate):



Format: XORI rt, rs, immediate

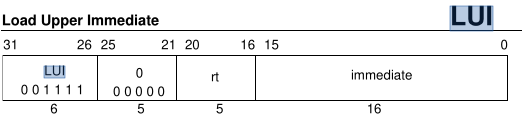
Purpose: To do a bitwise logical EXCLUSIVE OR with a constant.

Description: rt ← rs XOR immediate

Combine the contents of GPR rs and the 16-bit zero-extended immediate in a bitwise

logical exclusive OR operation and place the result into GPR rt.

LUI (Load Upper Immediate):



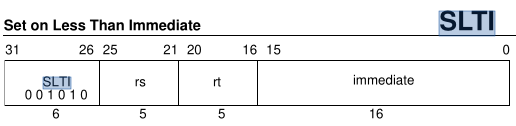
Format: LUI rt, immediate

Purpose: To load a constant into the upper half of a word.

Description: 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 sign-extended and placed into GPR rt.

SLTI (Set on Less Than Immediate):



Format: SLTI rt, rs, immediate

Purpose: To record the result of a less-than comparison with a constant.

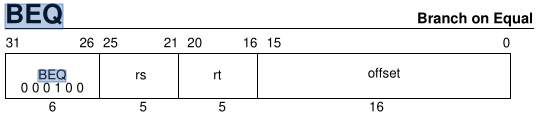
Description: rt ← (rs < immediate)

Compare the contents of GPR rs and the 16-bit signed immediate as signed integers and record the Boolean result of the comparison in GPR rt. If GPR rs is less than immediate

the result is 1 (true), otherwise 0 (false).

The arithmetic comparison does not cause an Integer Overflow exception.

BEQ (Branch on Equal):



Format: BEQ rs, rt, offset

Purpose: To compare GPRs then do a PC-relative conditional branch.

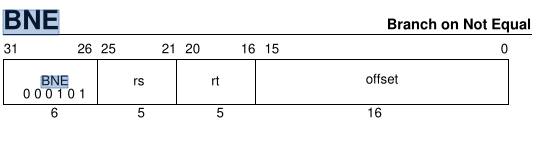
Description: if (rs = rt) then branch

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.

BNE (Branch on Not Equal):



Format: BNE rs, rt, offset

Purpose: To compare GPRs then do a PC-relative conditional branch.

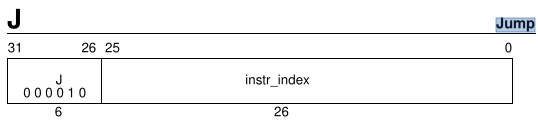
Description: if (rs ≠ rt) then branch

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.

J (Jump):



Format: J target

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

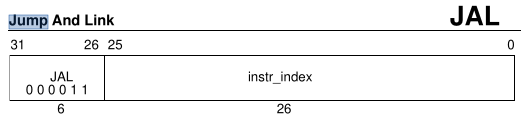
field shifted left 2 bits. The remaining upper bits are the corresponding bits of the

address of the instruction in the delay slot (not the branch itself).

Jump to the effective target address. Execute the instruction following the jump, in the

branch delay slot, before jumping.

JAL (Jump and Link):



Format: JAL target

Purpose: To 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, where execution would continue 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

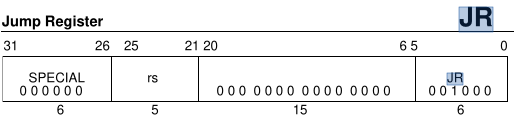
address of the instruction in the delay slot (not the branch itself).

Jump to the effective target address. Execute the instruction following the jump, in the

branch delay slot, before jumping.

J-Type

JR (Jump Register):



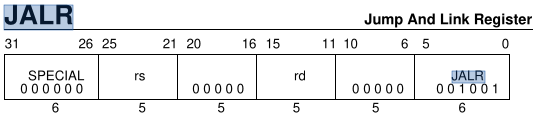
Format: JR rs

Purpose: To branch to an instruction address in a register.

Description: PC ← rs

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

JALR (Jump and Link Register):



Format: JALR rs (rd = 31 implied)

Format: JALR rd, rs

Purpose: To procedure call to an instruction address in a register.

Description: rd ← return\_addr, PC ← rs

Place the return address link in GPR rd. The return link is the address of the second

instruction following the branch, where execution would continue after a procedure call.

Jump to the effective target address in GPR rs. Execute the instruction following the

jump, in the branch delay slot, before jumping.