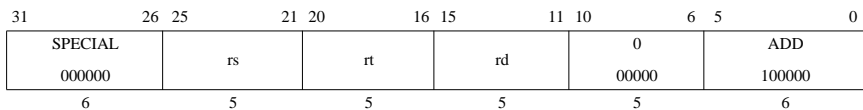


Add Word
ADD



Format: ADD rd, rs, rt MIPS32

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

Description: $GPR[rd] \leftarrow GPR[rs] + GPR[rt]$
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

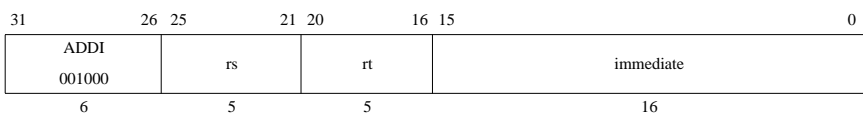
Operation:

```
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: $GPR[rt] \leftarrow GPR[rs] + immediate$
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:

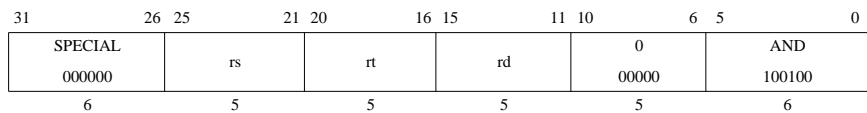
```
temp ← (GPR[rs]31 || GPR[rs]31..0) + sign_extend(immediate)
if temp32 ≠ temp31 then
    SignalException(IntegerOverflow)
else
    GPR[rt] ← temp
endif
```

Exceptions:
Integer Overflow

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

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:

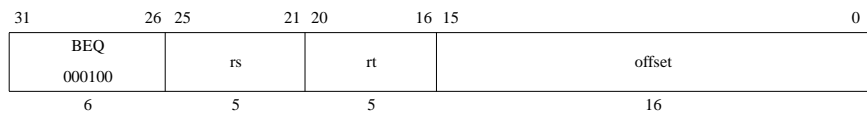
GPR[rd] ← GPR[rs] and GPR[rt]

Exceptions:

None

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 || 02)
      condition ← (GPR[rs] = GPR[rt])
I+1: if condition then
      PC ← PC + target_offset
endif
```

Exceptions:

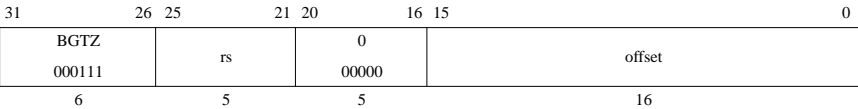
None

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 || 02)
      condition ← GPR[rs] > 0GPRLEN
I+1: if condition then
      PC ← PC + target_offset
endif
```

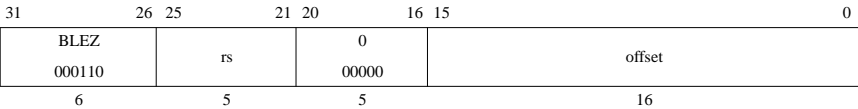
Exceptions:

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 || 02)
      condition ← GPR[rs] ≤ 0GPRLEN
I+1: if condition then
      PC ← PC + target_offset
endif
```

Exceptions:

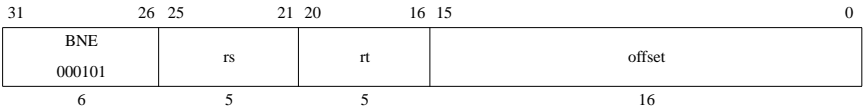
None

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 || 02)
   condition ← (GPR[rs] ≠ GPR[rt])
I+1: if condition then
      PC ← PC + target_offset
   endif
```

Exceptions:

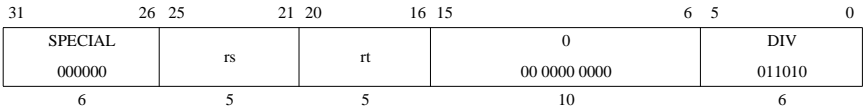
None

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 is placed 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

Programming Notes:

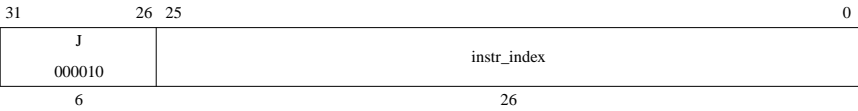
No arithmetic exception occurs under any circumstances. If divide-by-zero or overflow conditions are detected and some action taken, then the divide instruction is typically followed by additional instructions to check for a zero divisor and/or for overflow. If the divide is asynchronous then the zero-divisor check can execute in parallel with the divide. The action taken on either divide-by-zero or overflow is either a convention within the program itself, or more typically within the system software; one possibility is to take a BREAK exception with a *code* field value to signal the problem to the system software.

As an example, the C programming language in a UNIX® environment expects division by zero to either terminate the program or execute a program-specified signal handler. C does not expect overflow to cause any exceptional condition. If the C compiler uses a divide instruction, it also emits code to test for a zero divisor and execute a BREAK instruction to inform the operating system if a zero is detected.

In some processors the integer divide 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 divide so that other instructions can execute in parallel.

Historical Perspective:

In MIPS I through MIPS III, if either of the two instructions preceding the divide is an MFHI or MFLO, the result of the MFHI or MFLO is UNPREDICTABLE. Reads of the HI or LO special register must be separated from subsequent instructions that write to them by two or more instructions. This restriction was removed in MIPS IV and MIPS32 and all subsequent levels of the architecture.



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:
$$I+1:PC \leftarrow PC_{GPRLEN-1..28} \parallel instr_index \parallel 0^2$$

Exceptions:

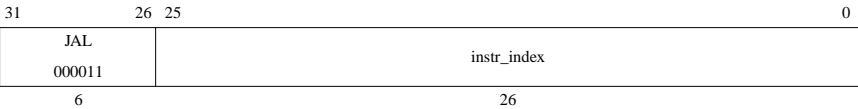
None

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 ← PCGPRLEN-1..28 || instr_index || 02
```

Exceptions:

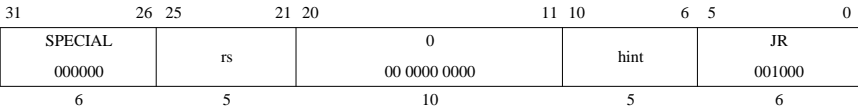
None

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:

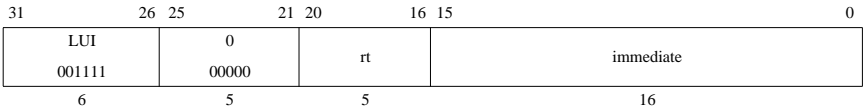
```
I: temp ← GPR[rs]
I+1: if Config1CA = 0 then
    PC ← temp
else
    PC ← tempGPRLEN-1..1 || 0
    ISAMode ← temp0
endif
```

Exceptions:

None

Programming Notes:

Software should use the value 31 for the *rs* field of the instruction word on return from a JAL, JALR, or BGEZAL, and should use a value other than 31 for remaining uses of JR.



Format: LUI *rt*, *immediate* MIPS32

Purpose:

To load a constant into the upper half of a word

Description: $GPR[rt] \leftarrow immediate \parallel 0^{16}$

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

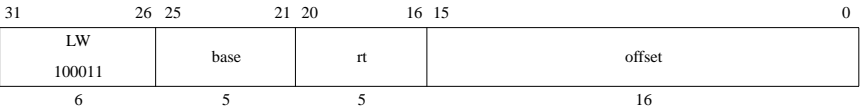
Operation:

$GPR[rt] \leftarrow immediate \parallel 0^{16}$

Exceptions:

None

Load Word **LW**



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

Purpose:

To load a word from memory as a signed value

Description: $GPR[rt] \leftarrow memory[GPR[base] + offset]$

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.

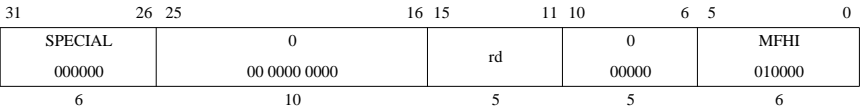
Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 02 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: $GPR[rd] \leftarrow HI$

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

Restrictions:

None

Operation:

```
GPR[rd] ← HI
```

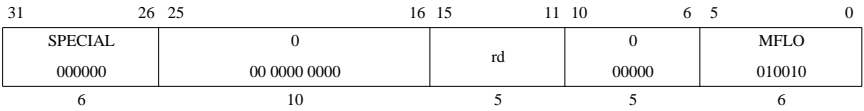
Exceptions:

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 RegisterMFLO



Format: MFLO rd MIPS32

Purpose:

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

Description: $GPR[rd] \leftarrow LO$

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

Restrictions: None

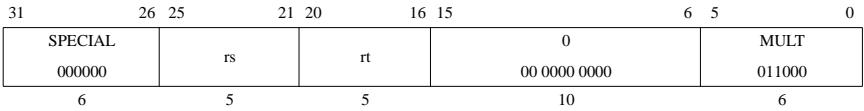
Operation:
 $GPR[rd] \leftarrow LO$

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 WordMULT



Format: MULT rs, rt MIPS32

Purpose:

To multiply 32-bit signed integers

Description: $(HI, LO) \leftarrow GPR[rs] \times GPR[rt]$

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:

$$\begin{aligned} \text{prod} &\leftarrow GPR[rs]_{31..0} \times GPR[rt]_{31..0} \\ LO &\leftarrow \text{prod}_{31..0} \\ HI &\leftarrow \text{prod}_{63..32} \end{aligned}$$

Exceptions:

None

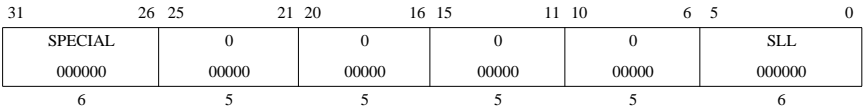
Programming Notes:

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.

No OperationNOP



Format: NOPAssembly Idiom

Purpose:

To perform no operation.

Description:

NOP is the assembly idiom used to denote no operation. The actual instruction is interpreted by the hardware as SLL r0, r0, 0.

Restrictions:

None

Operation:

None

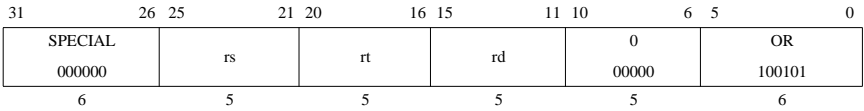
Exceptions:

None

Programming Notes:

The zero instruction word, which represents SLL, r0, r0, 0, is the preferred NOP for software to use to fill branch and jump delay slots and to pad out alignment sequences.

OrOR



Format: OR rd, rs, rtMIPS32

Purpose:

To do a bitwise logical OR

Description:

$GPR[rd] \leftarrow GPR[rs] \text{ or } GPR[rt]$
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:

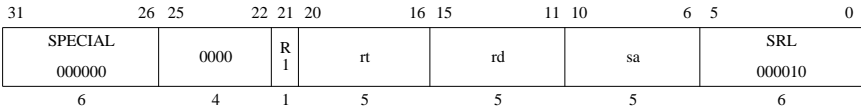
$GPR[rd] \leftarrow GPR[rs] \text{ or } GPR[rt]$

Exceptions:

None

Rotate Word Right

ROTR



Format: ROTR rd, rt, sa SmartMIPS Crypto, MIPS32 Release 2

Purpose:
To execute a logical right-rotate of a word by a fixed number of bits

Description: $GPR[rd] \leftarrow GPR[rt] \leftrightarrow (right) \ sa$
The contents of the low-order 32-bit word of GPR *rt* are rotated right; the word result is placed in GPR *rd*. The bit-rotate amount is specified by *sa*.

Restrictions:

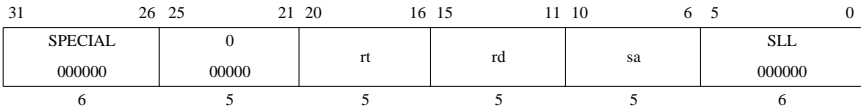
Operation:

```
if ((ArchitectureRevision() < 2) and (Config3SM = 0)) then
    UNPREDICTABLE
endif
s      ← sa
temp   ← GPR[rt]s-1..0 || GPR[rt]31..s
GPR[rd]← temp
```

Exceptions:
Reserved Instruction

Shift Word Left Logical

SLL



Format: SLL rd, rt, sa MIPS32

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

Description: $GPR[rd] \leftarrow GPR[rt] \ll sa$
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:

Operation:

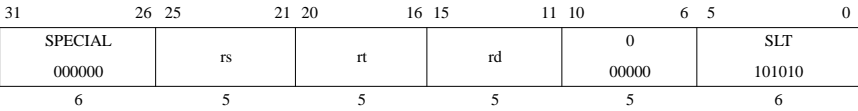
```
s      ← sa
temp   ← GPR[rt](31-s)..0 || 0s
GPR[rd]← temp
```

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: $GPR[rd] \leftarrow (GPR[rs] < GPR[rt])$
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.

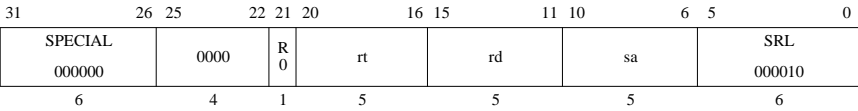
Restrictions:
None

Operation:

```
if GPR[rs] < GPR[rt] then
    GPR[rd] ← 0GPRLEN-1 || 1
else
    GPR[rd] ← 0GPRLEN
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: $GPR[rd] \leftarrow GPR[rt] \gg sa$ (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*.

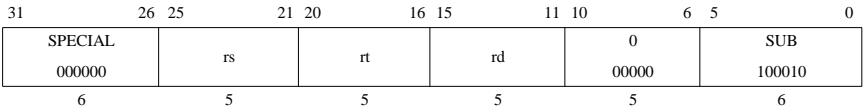
Restrictions:
None

Operation:

```
s      ← sa
temp   ← 0s || 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: $GPR[rd] \leftarrow GPR[rs] - GPR[rt]$

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
```

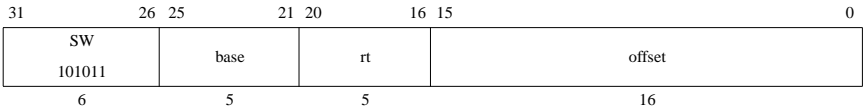
Exceptions:

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: $memory[GPR[base] + offset] \leftarrow GPR[rt]$

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:

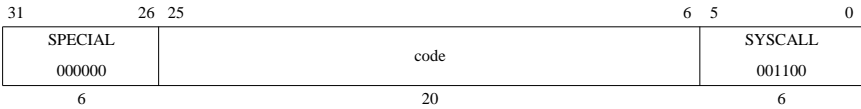
```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 02 then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, STORE)
dataword ← GPR[rt]
StoreMemory (CCA, WORD, dataword, pAddr, vAddr, DATA)
```

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.

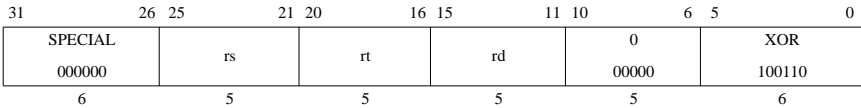
Restrictions:
None

Operation:
`SignalException(SystemCall)`

Exceptions:
System Call

Exclusive OR

XOR



Format: XOR rd, rs, rt

MIPS32

Purpose:
To do a bitwise logical Exclusive OR

Description: $GPR[rd] \leftarrow GPR[rs] \text{ XOR } GPR[rt]$
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:
 $GPR[rd] \leftarrow GPR[rs] \text{ xor } GPR[rt]$

Exceptions:
None