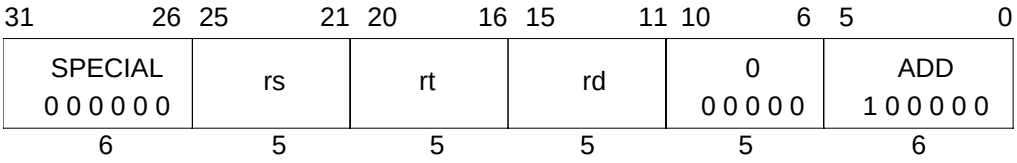
ADD — Add Word



Format: ADD rd, rs, rt **MIPS I**

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

Description: rd ← rs + 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 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:

On 64-bit processors, if either GPR *rt* or GPR *rs* do not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is undefined.

Operation:

```
if (NotWordValue(GPR[rs]) or NotWordValue(GPR[rt])) then UndefinedResult() endif
temp ←GPR[rs] + GPR[rt]
if (32_bit_arithmetic_overflow) then
    SignalException(IntegerOverflow)
else
    GPR[rd] ←sign_extend(temp31..0)
endif
```

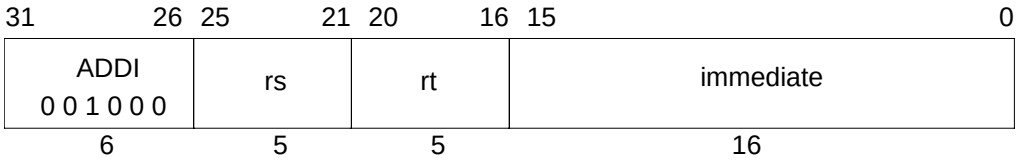
Exceptions:

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 **MIPS I**

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*.

Restrictions:

On 64-bit processors, if GPR *rs* does not contain a sign-extended 32-bit value (bits 63..31 equal), then the result of the operation is undefined.

Operation:

```
if (NotWordValue(GPR[rs])) then UndefinedResult() endif
temp ←GPR[rs] + sign_extend(immediate)
if (32_bit_arithmetic_overflow) then
    SignalException(IntegerOverflow)
else
    GPR[rt] ←sign_extend(temp31..0)
endif
```

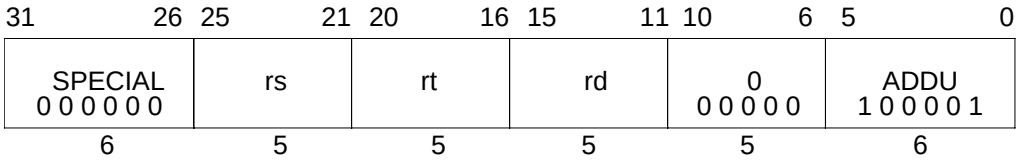
Exceptions:

Integer Overflow

Programming Notes:

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

Add Unsigned Word — ADDU



Format: ADDU rd, rs, rt **MIPS I**

Purpose: To add 32-bit integers.

Description: rd ← rs + 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:

On 64-bit processors, if either GPR *rt* or GPR *rs* do not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is undefined.

Operation:

```
if (NotWordValue(GPR[rs]) or NotWordValue(GPR[rt])) then UndefinedResult() endif
temp    ←GPR[rs] + GPR[rt]
GPR[rd]← sign_extend(temp31..0)
```

Exceptions:

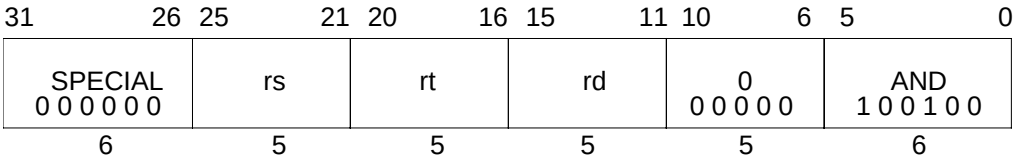
None

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. It is appropriate for arithmetic which is not signed, such as address arithmetic, or integer arithmetic environments that ignore overflow, such as "C" language arithmetic.

AND

And



Format: AND rd, rs, rt **MIPS I**

Purpose: To do a bitwise logical AND.

Description: rd ← rs AND 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

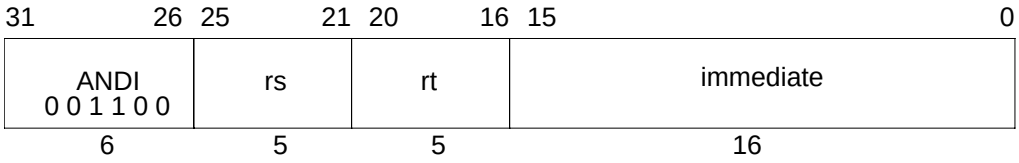
Operation:

GPR[rd] ← GPR[rs] and GPR[rt]

Exceptions:

None

And Immediate — ANDI



Format: ANDI rt, rs, immediate **MIPS I**

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*.

Restrictions:

None

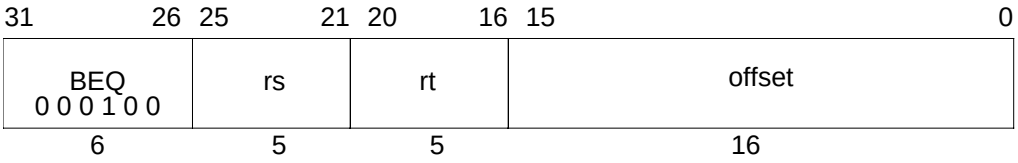
Operation:

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

Exceptions:

None

BEQ — Branch on Equal



Format: BEQ rs, rt, offset **MIPS I**

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.

Restrictions:

None

Operation:

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

Exceptions:

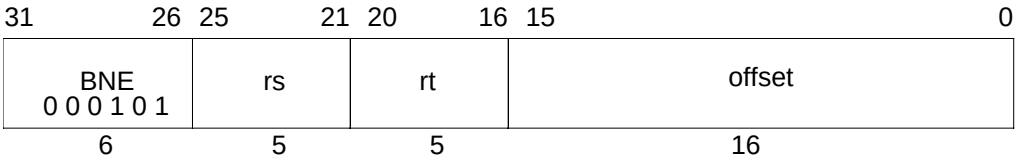
None

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 more distant addresses.

BNE

Branch on Not Equal



Format: BNE rs, rt, offset **MIPS I**

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.

Restrictions:

None

Operation:

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

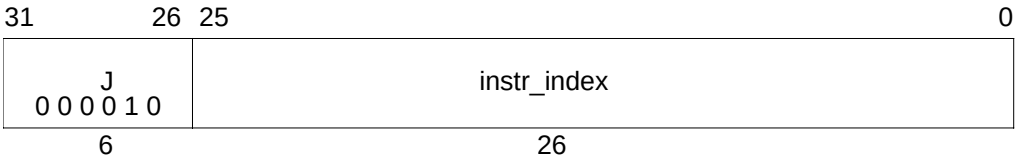
Exceptions:

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 more distant addresses.

J — Jump



Format: J target **MIPS I**

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.

Restrictions:

None

Operation:

I:
I+1: $PC \leftarrow PC_{GPRLEN..28} \,||\, instr\_index \,||\, 0^2$

Exceptions:

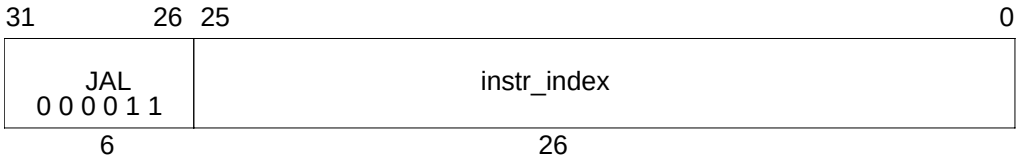
None

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 to anywhere in the region from anywhere in the region which a signed relative offset would not allow.

This definition creates the boundary case where the branch instruction is in the last word of a 256 MB region and can therefore only branch to the following 256 MB region containing the branch delay slot.

Jump And Link — JAL



Format: JAL target **MIPS I**

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.

Restrictions:

None

Operation:

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

Exceptions:

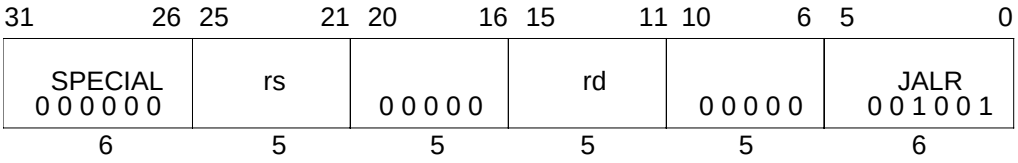
None

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 to anywhere in the region from anywhere in the region which a signed relative offset would not allow.

This definition creates the boundary case where the branch instruction is in the last word of a 256 MB region and can therefore only branch to the following 256 MB region containing the branch delay slot.

JALR — Jump And Link Register



Format: JALR rs (rd = 31 implied)

Format: JALR rd, rs **MIPS I**

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.

Restrictions:

Register specifiers *rs* and *rd* must not be equal, because such an instruction does not have the same effect when re-executed. The result of executing such an instruction is undefined. This restriction permits an exception handler to resume execution by re-executing the branch when an exception occurs in the branch delay slot.

The effective target address in GPR *rs* must be naturally aligned. If either of the two least-significant bits are not -zero, then an Address Error exception occurs, not for the jump instruction, but when the branch target is subsequently fetched as an instruction.

Operation:

```
I:   temp ← GPR[rs]
     GPR[rd] ← PC + 8
I+1: PC ← temp
```

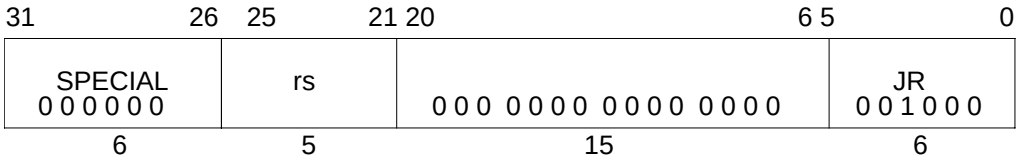
Exceptions:

None

Programming Notes:

This is the only branch-and-link instruction that can select a register for the return link; all other link instructions use GPR 31 The default register for *GPR rd*, if omitted in the assembly language instruction, is GPR 31.

Jump Register — JR



Format: JR rs **MIPS I**

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.

Restrictions:

The effective target address in GPR *rs* must be naturally aligned. If either of the two least-significant bits are not -zero, then an Address Error exception occurs, not for the jump instruction, but when the branch target is subsequently fetched as an instruction.

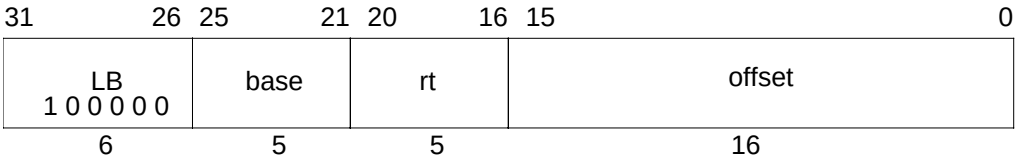
Operation:

```
I:   temp ← GPR[rs]
I+1: PC ← temp
```

Exceptions:

None

LB — Load Byte



Format: LB rt, offset(base) **MIPS I**

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.

Restrictions:

None

Operation: **32-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddr(PSIZE-1).. 2 || (pAddr1..0 xor ReverseEndian2)
memword ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
byte ← vAddr1..0 xor BigEndianCPU2
GPR[rt] ← sign_extend(memword7+8*byte..8*byte)
```

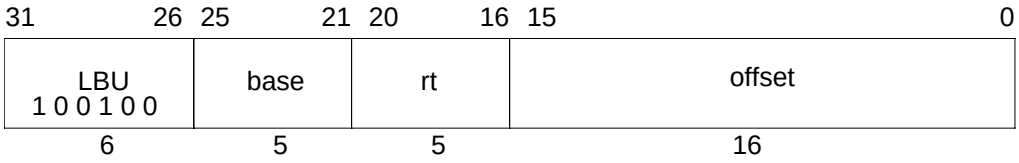
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE–1..3 || (pAddr2..0 xor ReverseEndian3)
memdouble ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
byte ← vAddr2..0 xor BigEndianCPU3
GPR[rt] ← sign_extend(memdouble7+8*byte..8*byte)
```

Exceptions:

TLB Refill, TLB Invalid
Address Error

Load Byte Unsigned — LBU



Format: LBU rt, offset(base) **MIPS I**

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.

Restrictions:

None

Operation: **32-bit processors**

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

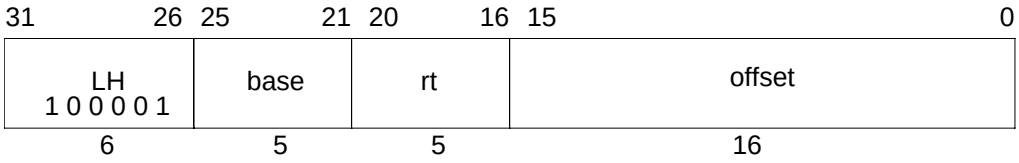
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddr_PSIZE–1..3 || (pAddr_2..0 xor ReverseEndian^3)
memdouble ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
byte ← vAddr_2..0 xor BigEndianCPU^3
GPR[rt] ← zero_extend(memdouble_7+8* byte..8* byte)
```

Exceptions:

TLB Refill, TLB Invalid
Address Error

Load Halfword — LH



Format: LH rt, offset(base) **MIPS I**

Purpose: To load a halfword from memory as a signed 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, 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:

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

MIPS IV: The low-order bit of the *offset* field must be zero. If it is not, the result of the instruction is undefined.

Operation: **32-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr0) ≠ 0 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE – 1..2 || (pAddr1..0 xor (ReverseEndian || 0))
memword ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
byte ← vAddr1..0 xor (BigEndianCPU || 0)
GPR[rt] ← sign_extend(memword15+8*byte..8* byte)
```

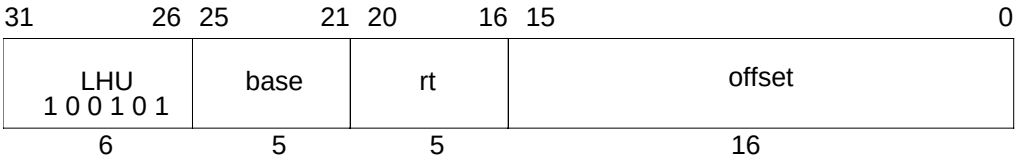
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr0) ≠ 0 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE – 1..3 || (pAddr2..0 xor (ReverseEndian || 0))
memdouble ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
byte ← vAddr2..0 xor (BigEndianCPU2 || 0)
GPR[rt] ← sign_extend(memdouble15+8*byte..8* byte)
```

Exceptions:

TLB Refill , TLB Invalid
Bus Error
Address Error

LHU — Load Halfword Unsigned



Format: LHU rt, offset(base) **MIPS I**

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.

Restrictions:

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

MIPS IV: The low-order bit of the *offset* field must be zero. If it is not, the result of the instruction is undefined.

Operation: **32-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr0) ≠ 0 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE – 1..2 || (pAddr1..0 xor (ReverseEndian || 0))
memword ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
byte ← vAddr1..0 xor (BigEndianCPU || 0)
GPR[rt] ← zero_extend(memword15+8*byte..8*byte)
```

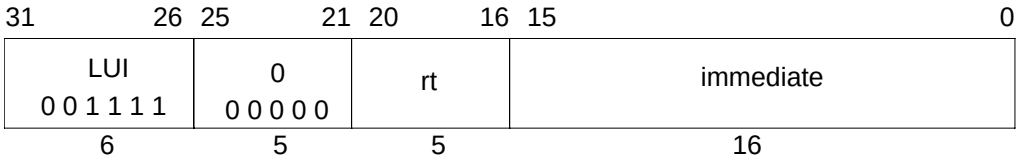
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr0) ≠ 0 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE – 1..3 || (pAddr2..0 xor (ReverseEndian2 || 0))
memdouble ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
byte ← vAddr2..0 xor (BigEndianCPU2 || 0)
GPR[rt] ← zero_extend(memdouble15+8*byte..8*byte)
```

Exceptions:

TLB Refill, TLB Invalid
Address Error

Load Upper Immediate — LUI



Format: LUI rt, immediate **MIPS I**

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*.

Restrictions:

None

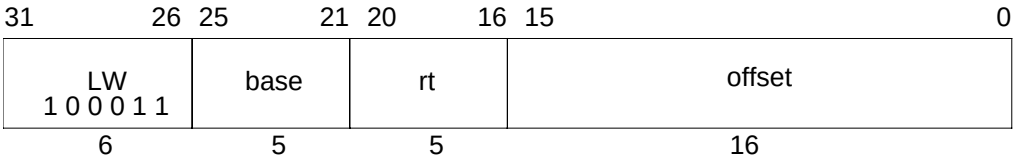
Operation:

GPR[rt] ← sign_extend(immediate || 0^{16})

Exceptions:

None

LW — Load Word



Format: LW rt, offset(base) **MIPS I**

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.

Restrictions:

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

MIPS IV: The low-order 2 bits of the *offset* field must be zero. If they are not, the result of the instruction is undefined.

Operation: **32-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr1..0) ≠ 0^2 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
memword ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
GPR[rt] ← memword
```

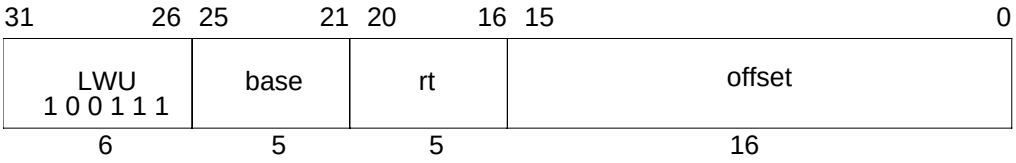
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr1..0) ≠ 0^2 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE-1..3 || (pAddr2..0 xor (ReverseEndian || 0^2))
memdouble ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
byte ← vAddr2..0 xor (BigEndianCPU || 0^2)
GPR[rt] ← sign_extend(memdouble31+8*byte..8*byte)
```

Exceptions:

TLB Refill, TLB Invalid
Bus Error
Address Error

LWU — Load Word Unsigned



Format: LWU rt, offset(base) **MIPS III**

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.

Restrictions:

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

MIPS IV: The low-order 2 bits of the *offset* field must be zero. If they are not, the result of the instruction is undefined.

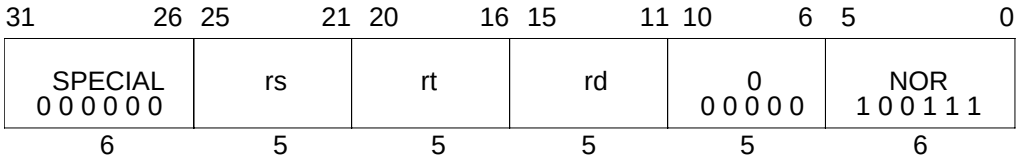
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr1..0) ≠ 0^2 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE-1..3 || (pAddr2..0 xor (ReverseEndian || 0^2))
memdouble ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
byte ← vAddr2..0 xor (BigEndianCPU || 0^2)
GPR[rt] ← 0^32 || memdouble31+8*byte..8*byte
```

Exceptions:

TLB Refill, TLB Invalid
Bus Error
Address Error
Reserved Instruction

Not Or **NOR**



Format: NOR rd, rs, rt **MIPS I**

Purpose: To do a bitwise logical NOT OR.

Description: rd ← rs NOR rt

The contents of GPR *rs* are combined with the contents of GPR *rt* in a bitwise logical NOR operation. The result is placed into GPR *rd*.

Restrictions:

None

Operation:

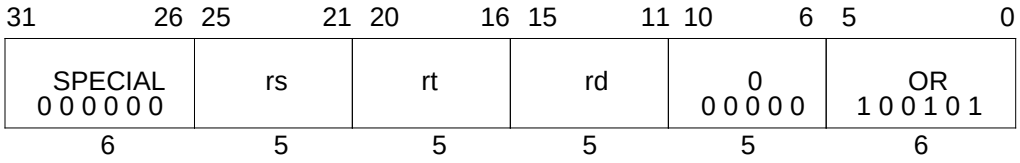
GPR[rd] ← GPR[rs] nor GPR[rt]

Exceptions:

None

OR

Or



Format: OR rd, rs, rt **MIPS I**

Purpose: To do a bitwise logical OR.

Description: rd ← rs OR 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

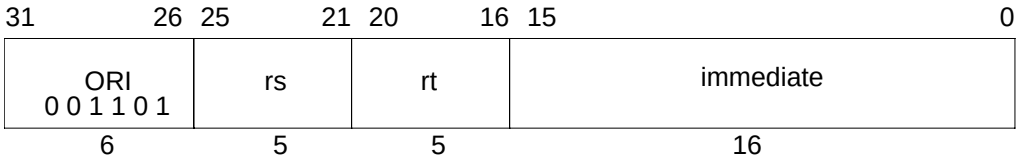
Operation:

```
GPR[rd] ← GPR[rs] or GPR[rt]
```

Exceptions:

None

Or Immediate — ORI



Format: ORI rt, rs, immediate **MIPS I**

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*.

Restrictions:

None

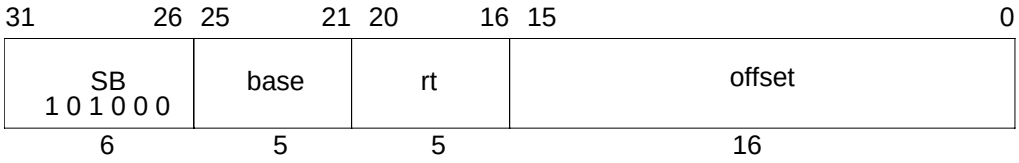
Operation:

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

Exceptions:

None

Store Byte — SB



Format: SB rt, offset(base) **MIPS I**

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.

Restrictions:

None

Operation: **32-bit processors**

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

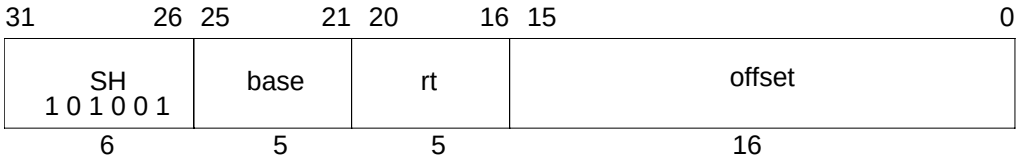
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, STORE)
pAddr ← pAddr_PSIZE-1..3 || (pAddr_2..0 xor ReverseEndian^3)
byte ← vAddr_2..0 xor BigEndianCPU^3
datadouble ← GPR[rt]_63–8*byte..0 || 0^8*byte
StoreMemory (uncached, BYTE, datadouble, pAddr, vAddr, DATA)
```

Exceptions:

TLB Refill, TLB Invalid
TLB Modified
Bus Error
Address Error

Store Halfword — SH



Format: SH rt, offset(base) **MIPS I**

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.

Restrictions:

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

MIPS IV: The low-order bit of the *offset* field must be zero. If it is not, the result of the instruction is undefined.

Operation: **32-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr0) ≠ 0 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, STORE)
pAddr ← pAddrPSIZE-1..2 || (pAddr1..0 xor (ReverseEndian || 0))
byte ← vAddr1..0 xor (BigEndianCPU || 0)
dataword ← GPR[rt]31–8*byte..0 || 0^8*byte
StoreMemory (uncached, HALFWORD, dataword, pAddr, vAddr, DATA)
```

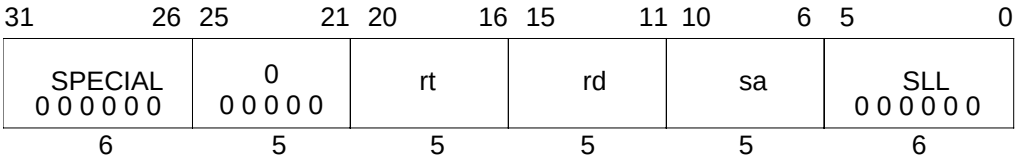
Operation: **64-bit processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr0) ≠ 0 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, STORE)
pAddr ← pAddrPSIZE-1..3 || (pAddr2..0 xor (ReverseEndian^2 || 0))
byte ← vAddr2..0 xor (BigEndianCPU^2 || 0)
datadouble ← GPR[rt]63–8*byte..0 || 0^8*byte
StoreMemory (uncached, HALFWORD, datadouble, pAddr, vAddr, DATA)
```

Exceptions:

TLB Refill, TLB Invalid
TLB Modified
Address Error

SLL — Shift Word Left Logical



Format: SLL rd, rt, sa **MIPS I**

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

Description: rd ← rt << sa

The contents of the low-order 32-bit word of GPR *rt* are shifted left, inserting zeroes into the emptied bits; the word result is placed in GPR *rd*. The bit shift count is specified by *sa*. If *rd* is a 64-bit register, the result word is sign-extended.

Restrictions:

None

Operation:

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

Exceptions:

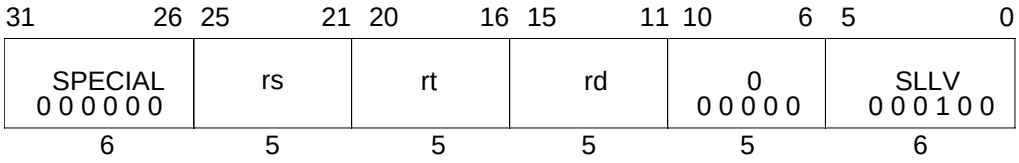
None

Programming Notes:

Unlike nearly all other word operations the input operand does not have to be a properly sign-extended word value to produce a valid sign-extended 32-bit result. The result word is always sign extended into a 64-bit destination register; this instruction with a zero shift amount truncates a 64-bit value to 32 bits and sign extends it.

Some assemblers, particularly 32-bit assemblers, treat this instruction with a shift amount of zero as a NOP and either delete it or replace it with an actual NOP.

Shift Word Left Logical Variable — SLLV



Format: SLLV rd, rt, rs **MIPS I**

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.

Restrictions:

None

Operation:

$s \leftarrow GP[rs]_{4..0}$
$temp \leftarrow GPR[rt]_{(31-s)..0} \,||\, 0^s$
$GPR[rd] \leftarrow sign\_extend(temp)$

Exceptions:

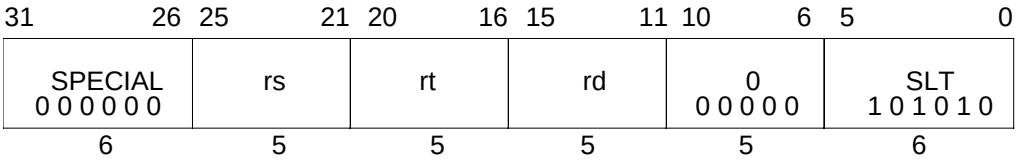
None

Programming Notes:

Unlike nearly all other word operations the input operand does not have to be a properly sign-extended word value to produce a valid sign-extended 32-bit result. The result word is always sign extended into a 64-bit destination register; this instruction with a zero shift amount truncates a 64-bit value to 32 bits and sign extends it.

Some assemblers, particularly 32-bit assemblers, treat this instruction with a shift amount of zero as a NOP and either delete it or replace it with an actual NOP.

SLT — Set On Less Than



Format: SLT rd, rs, rt **MIPS I**

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).

The arithmetic comparison does not cause an Integer Overflow exception.

Restrictions:

None

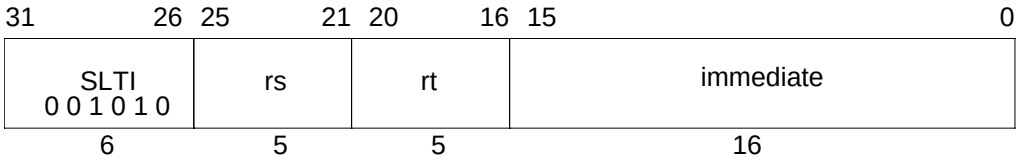
Operation:

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

Exceptions:

None

Set on Less Than Immediate — SLTI



Format: SLTI rt, rs, immediate **MIPS I**

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.

Restrictions:

None

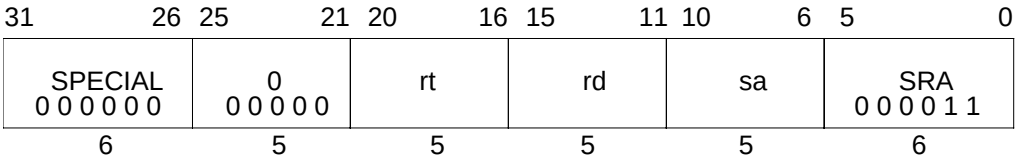
Operation:

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

Exceptions:

None

SRA — Shift Word Right Arithmetic



Format: SRA rd, rt, sa **MIPS I**

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

Description: rd ← rt >> sa (arithmetic)

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 *sa*. If *rd* is a 64-bit register, the result word is sign-extended.

Restrictions:

On 64-bit processors, if GPR *rt* does not contain a sign-extended 32-bit value (bits 63..31 equal) then the result of the operation is undefined.

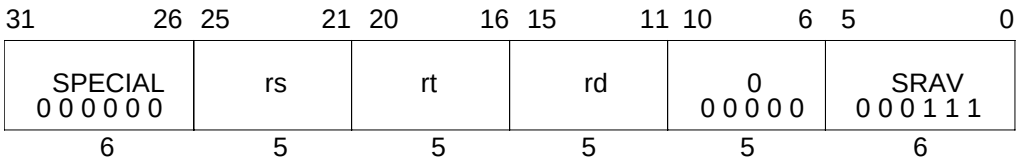
Operation:

```
if (NotWordValue(GPR[rt])) then UndefinedResult() endif
s       ← sa
temp    ← (GPR[rt]31)^s || GPR[rt]31..s
GPR[rd]← sign_extend(temp)
```

Exceptions:

None

Shift Word Right Arithmetic Variable — SRAV



Format: SRAV rd, rt, rs **MIPS I**

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

Description: rd ← rt >> rs (arithmetic)

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.

Restrictions:

On 64-bit processors, if GPR *rt* does not contain a sign-extended 32-bit value (bits 63..31 equal) then the result of the operation is undefined.

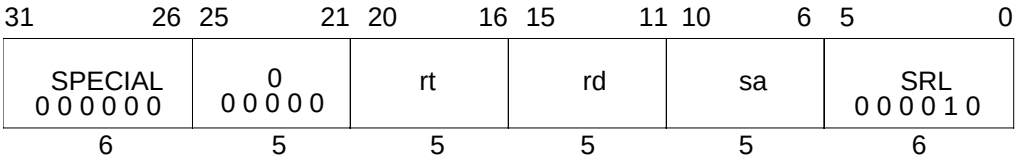
Operation:

```
if (NotWordValue(GPR[rt])) then UndefinedResult() endif
s       ← GPR[rs]4..0
temp    ← (GPR[rt]31)s || GPR[rt]31..s
GPR[rd]← sign_extend(temp)
```

Exceptions:

None

SRL — Shift Word Right Logical



Format: SRL rd, rt, sa **MIPS I**

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

Description: rd ← 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 count is specified by *sa*. If *rd* is a 64-bit register, the result word is sign-extended.

Restrictions:

On 64-bit processors, if GPR *rt* does not contain a sign-extended 32-bit value (bits 63..31 equal) then the result of the operation is undefined.

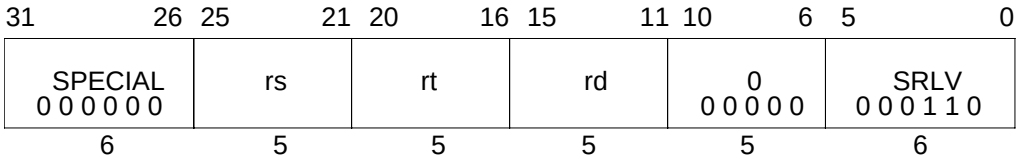
Operation:

```
if (NotWordValue(GPR[rt])) then UndefinedResult() endif
s       ← sa
temp    ← 0^s || GPR[rt]_31..s
GPR[rd]← sign_extend(temp)
```

Exceptions:

None

Shift Word Right Logical Variable — SRLV



Format: SRLV rd, rt, rs **MIPS I**

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

Description: rd ← rt >> rs (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 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.

Restrictions:

On 64-bit processors, if GPR *rt* does not contain a sign-extended 32-bit value (bits 63..31 equal) then the result of the operation is undefined.

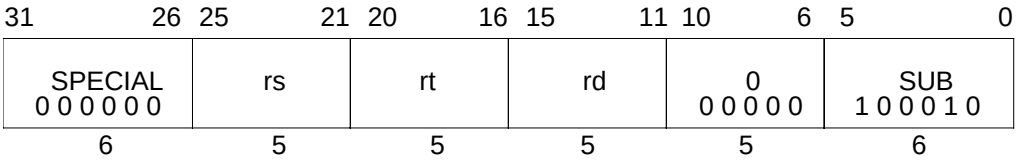
Operation:

```
if (NotWordValue(GPR[rt])) then UndefinedResult() endif
s       ← GPR[rs]4..0
temp    ← 0^s || GPR[rt]31..s
GPR[rd]← sign_extend(temp)
```

Exceptions:

None

SUB — Subtract Word



Format: SUB rd, rs, rt **MIPS I**

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

Description: rd ← rs - 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:

On 64-bit processors, if either GPR *rt* or GPR *rs* do not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is undefined.

Operation:

```
if (NotWordValue(GPR[rs]) or NotWordValue(GPR[rt])) then UndefinedResult() endif
temp ← GPR[rs] - GPR[rt]
if (32_bit_arithmetic_overflow) then
    SignalException(IntegerOverflow)
else
    GPR[rd] ←temp
endif
```

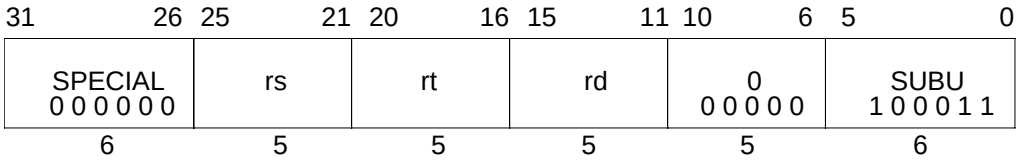
Exceptions:

Integer Overflow

Programming Notes:

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

Subtract Unsigned Word — SUBU



Format: SUBU rd, rs, rt **MIPS I**

Purpose: To subtract 32-bit integers.

Description: rd ← rs - rt

The 32-bit word value in GPR *rt* is subtracted from 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:

On 64-bit processors, if either GPR *rt* or GPR *rs* do not contain sign-extended 32-bit values (bits 63..31 equal), then the result of the operation is undefined.

Operation:

```
if (NotWordValue(GPR[rs]) or NotWordValue(GPR[rt])) then UndefinedResult() endif
temp ←GPR[rs] - GPR[rt]
GPR[rd] ←temp
```

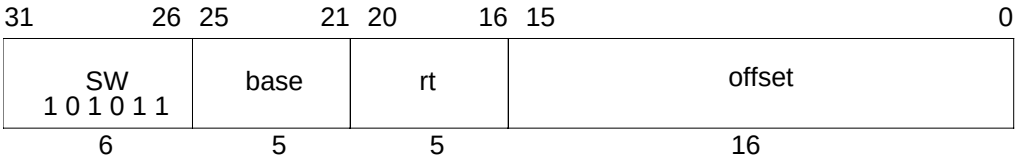
Exceptions:

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. It is appropriate for arithmetic which is not signed, such as address arithmetic, or integer arithmetic environments that ignore overflow, such as “C” language arithmetic.

SW — Store Word



Format: SW rt, offset(base) **MIPS I**

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.

Restrictions:

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

MIPS IV: The low-order 2 bits of the *offset* field must be zero. If they are not, the result of the instruction is undefined.

Operation: **32-bit Processors**

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

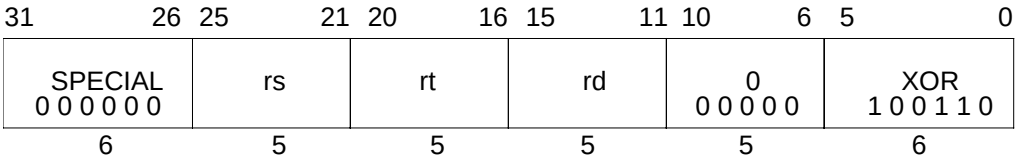
Operation: **64-bit Processors**

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr1..0) ≠ 0^2 then SignalException(AddressError) endif
(pAddr, uncached) ← AddressTranslation (vAddr, DATA, STORE)
pAddr ← pAddrPSIZE-1..3 || (pAddr2..0 xor (ReverseEndian || 0^2)
byte ← vAddr2..0 xor (BigEndianCPU || 0^2)
datadouble ← GPR[rt]63-8*byte || 0^8*byte
StoreMemory (uncached, WORD, datadouble, pAddr, vAddr, DATA)
```

Exceptions:

TLB Refill, TLB Invalid
TLB Modified
Address Error

XOR — Exclusive OR



Format: XOR rd, rs, rt **MIPS I**

Purpose: To do a bitwise logical EXCLUSIVE OR.

Description: rd ← rs XOR 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

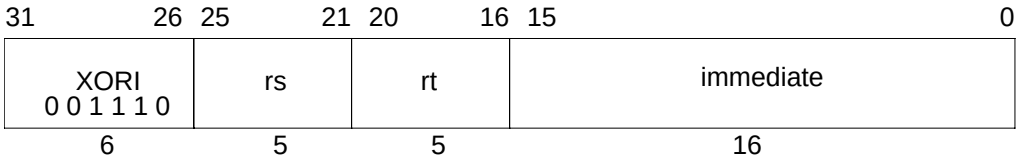
Operation:

```
GPR[rd] ← GPR[rs] xor GPR[rt]
```

Exceptions:

None

Exclusive OR Immediate — XORI



Format: XORI rt, rs, immediate **MIPS I**

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*.

Restrictions:

None

Operation:

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

Exceptions:

None