Datasheet

CC100-C Processor Instruction Set Architecture

1.0

Scope

This document contains the CC100-C Processor Instruction Set Architecture. Instruction set is composed of several functional groups. Each instruction is described in detail along with their purpose, exception generation and limitations.

Contents

1. Instruction Set

This chapter describes the instruction set architecture for the CC100-C processing unit.

The instruction set is derived from the MIPS®-II [1] architecture. All differences are described in detail herein.

The CC100-C processing unit pipeline is fully interlocked and there are no architectural delayed loads. Programs will execute correctly when the loaded data is used by the instructions following the loads, but this may require extra clock cycles.

All branches have architectural delay of one instruction. When a branch is taken, the instruction immediately following the branch instruction, in the branch delay slot, is executed before the branch to the target instruction takes place.

[1] MIPS is registered trademark of Imagination Technologies LLC. ChipCraft Sp. z .o.o. is not associated with Imagination Technologies LLC in any way.

1.1 Functional Instruction Groups

1.1.1 3-Operand ALU Instructions

Table 1.1. 3-Operand ALU Instructions

Mnemonic	*Description*
ADD	Add Word
ADDU	Add Unsigned Word
SUB	Subtract Word
SUBU	Subtract Unsigned Word
SLT	Set On Less Than
SLTU	Set On Less Than Unsigned
AND	And
OR	Or
XOR	Exclusive Or
NOR	Nor

1.1.2 ALU Instructions With an Immediate Operand

Table 1.2. ALU Instructions With an Immediate Operand

Mnemonic	*Description*
ADDI	Add Immediate Word
ADDIU	Add Immediate Unsigned Word
SLTI	Set on Less Than Immediate
SLTIU	Set on Less Than Immediate Unsigned
ANDI	And Immediate
ORI	Or Immediate
XORI	Exclusive Or Immediate
LUI	Load Upper Immediate

1.1.3 Shift Instructions

Table 1.3. Shift Instructions

Mnemonic	*Description*
SLL	Shift Word Left Logical
SRL	Shift Word Right Logical
SRA	Shift Word Right Arithmetical
SLLV	Shift Word Left Logical Variable
SRLV	Shift Word Right Logical Variable
SRAV	Shift Word Right Arithmetical Variable

1.1.4 Multiply and Divide Instructions

Table 1.4. Multiply and Divide Instructions

Mnemonic	*Description*
MULT	Multiply Word
MULTU	Multiply Word Unsigned
DIV	Divide Word
DIVU	Divide Unsigned Word
MFHI	Move From HI
MTHI	Move To HI
MFLO	Move From LO
MTLO	Move To LO

1.1.5 Aligned CPU Load/Store Instructions

Table 1.5. Aligned CPU Load/Store Instructions

Mnemonic	*Description*
LB	Load Byte
LBU	Load Byte Unsigned
SB	Store Byte
LH	Load Halfword
LHU	Load Halfword Unsigned
SH	Store Halfword
LW	Load Word
SH	Store Word

1.1.6 Jump and Branch Instructions

Table 1.6. Jump Instructions Jumping Within a 256 Megabyte Region

Mnemonic	*Description*
J	Jump
JAL	Jump And Link

Table 1.7. Jump Instructions to Absolute Address

Mnemonic	*Description*
JR	Jump Register
JALR	Jump And Link Register

Table 1.8. PC-Relative Conditional Branch Instructions Comparing 2 Registers

Mnemonic	*Description*
BEQ	Branch on Equal
BNE	Branch on Not Equal

Table 1.9. PC-Relative Conditional Branch Instructions Comparing Against Zero

Mnemonic	*Description*
BLEZ	Branch on Less Than or Equal Zero
BGTZ	Branch on Grater than Zero
BLTZ	Branch on Less Than Zero
BGEZ	Branch on Greater Than or Equal Zero
BLTZAL	Branch on Less Than Zero and Link
BGEZAL	Branch on Grater than Zero and Link

1.1.7 Atomic Update CPU Load/Store Instructions

Table 1.10. Atomic Update CPU Load/Store Instructions

Mnemonic	*Description*
LL	Load Linked Word
SC	Store Conditional Word

1.1.8 Exception Instructions

Table 1.11. Exception Instructions

Mnemonic	*Description*
SYSCALL	System Call
BREAK	Breakpoint

1.1.9 Trap-on-Condition Instructions Comparing Two Registers

Table 1.12. Trap-on-Condition Instructions Comparing Two Registers

Mnemonic	*Description*
TGE	Trap if Greater Than or Equal
TGEU	Trap if Greater Than or Equal Unsigned
TLT	Trap if Less Than
TLTU	Trap if Less Than Unsigned
TEQ	Trap if Equal
TNE	Trap if Not Equal

Table 1.13. Trap-on-Condition Instructions Comparing an Immediate

Mnemonic	*Description*
TGEI	Trap if Greater Than or Equal Immediate
TGEIU	Trap if Greater Than or Equal Unsigned Immediate
TLTI	Trap if Less Than Immediate
TLTIU	Trap if Less Than Unsigned Immediate
TEQI	Trap if Equal Immediate
TNEI	Trap if Not Equal Immediate

1.1.10 Privileged Instructions

Table 1.14. Privileged Instructions

Mnemonic	*Description*
RFE	Return From Exception
MFC0	Move Word To Coprocessor-0
MTC0	Move Word From Coprocessor-0

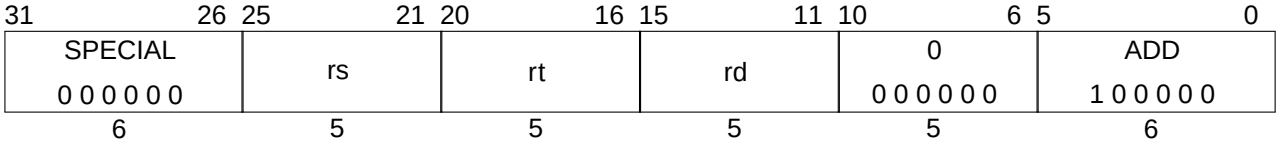
1.1.11 Serialization Instructions

Table 1.15. Serialization Instructions

Mnemonic	*Description*
SYNC	Synchronize Shared Memory

1.2 Instruction Descriptions

1.2.1 ADD



Format: ADD rd, rs, rt

Purpose: To add two 32-bit integers. If the overflow occurs, than trap.

Operation:

if $(32\_bit\_arithmetic\_overflow)$ **then**

$SignalException(IntegerOverflow)$

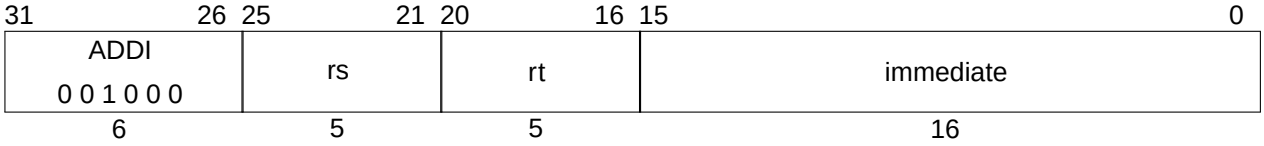
else

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

end if

Exceptions: Integer Overflow.

1.2.2 ADDI



Format: ADDI rt, rs, immediate

Purpose: To add 16-bit constant to a 32-bit integer. If the overflow occurs, than trap.

Operation:

if $(32\_bit\_arithmetic\_overflow)$ **then**

$SignalException(IntegerOverflow)$

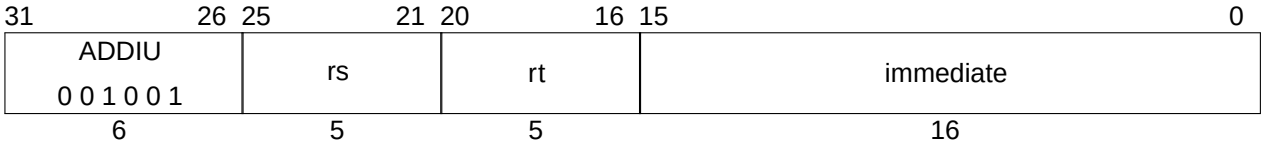
else

$GPR[rt] \leftarrow GPR[rs] + sign\_extend(immediate)$

end if

Exceptions: Integer Overflow.

1.2.3 ADDIU



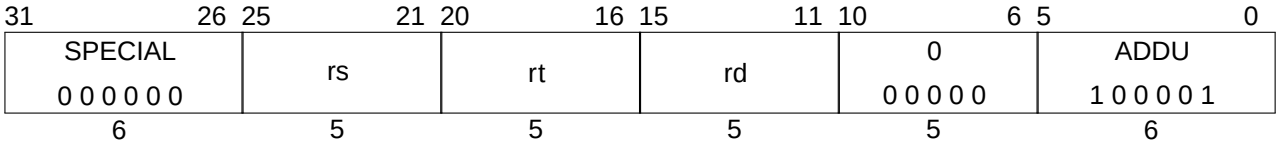
Format: ADDIU rt, rs, immediate

Purpose: To add 16-bit constant to a 32-bit integer.

Operation: $GPR[rt] \leftarrow GPR[rs] + sign\_extend(immediate)$

Exceptions: None.

1.2.4 ADDU



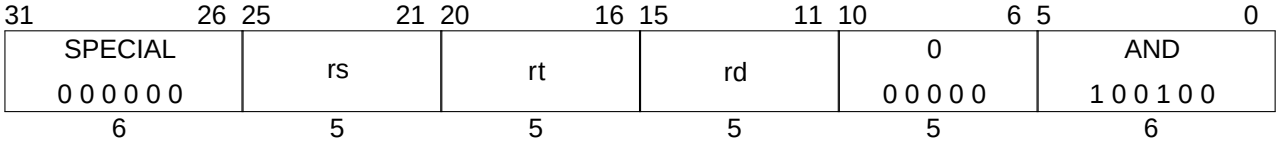
Format: ADDU rd, rs, rt

Purpose: To add two 32-bit integers.

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

Exceptions: None.

1.2.5 AND



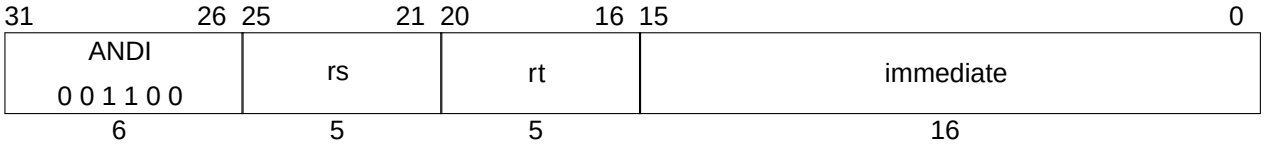
Format: AND rd, rs, rt

Purpose: To do a bitwise logical AND.

Operation: $GPR[rd] \leftarrow GPR[rs]\ AND\ GPR[rt]$

Exceptions: None.

1.2.6 ANDI



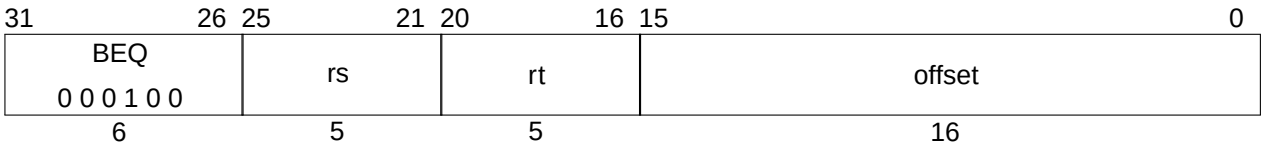
Format: ANDI rt, rs, immediate

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

Operation: $GPR[rt] \leftarrow GPR[rs]\ AND\ zero\_extend(immediate)$

Exceptions: None.

1.2.7 BEQ



Format: BEQ rs, rt, offset

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

Description: 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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

Operation:

I : $condition \leftarrow (GPR[rs] == GPR[rt])$

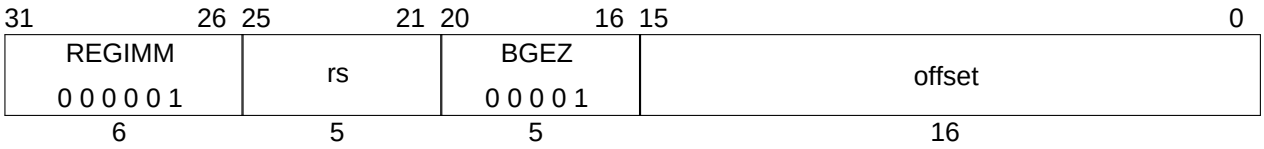
if $(condition)$ **then**

$I+1$: $PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$

end if

Exceptions: None.

1.2.8 BGEZ



Format: BGEZ rs, offset

Purpose: To test a GPR then do a PC-relative conditional branch.

Description: 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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

Operation:

I : $condition \leftarrow (GPR[rs] \geq 0)$

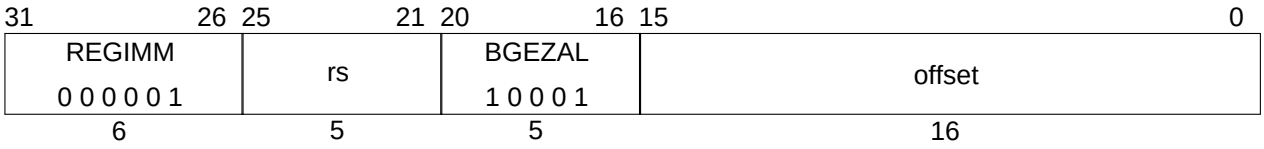
if $(condition)$ **then**

$I+1$: $PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$

end if

Exceptions: None.

1.2.9 BGEZAL



Format: BGEZAL rs, offset

Purpose: To test a GPR then do a PC-relative conditional branch.

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.

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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

Restrictions: GPR 31 must not be used for the source register *rs*, because such an instruction does not have the same effect when re-executed.

Operation:

I :

$$condition \leftarrow (GPR[rs] \geq 0)$$
$$GPR[31] \leftarrow PC + 8$$

$I+1$:

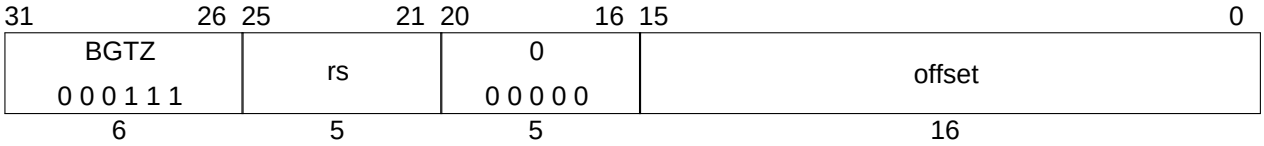
if $(condition)$ **then**

$$PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$$

end if

Exceptions: None.

1.2.10 BGTZ



Format: BGTZ rs, offset

Purpose: To test a GPR then do a PC-relative conditional branch.

Description: 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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

Operation:

I : $condition \leftarrow (GPR[rs] > 0)$

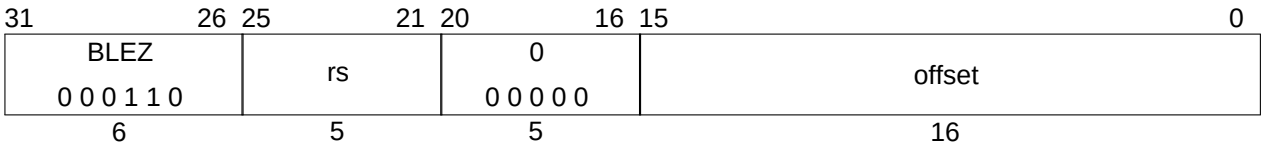
if $(condition)$ **then**

$I+1$: $PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$

end if

Exceptions: None.

1.2.11 BLEZ



Format: BLEZ rs, offset

Purpose: To test a GPR then do a PC-relative conditional branch.

Description: 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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

Operation:

I : $condition \leftarrow (GPR[rs] \leq 0)$

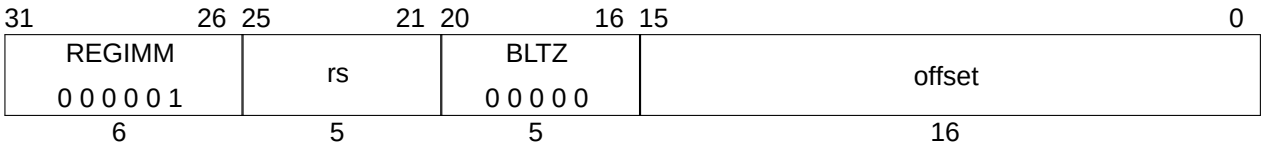
if $(condition)$ **then**

$I+1$: $PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$

end if

Exceptions: None.

1.2.12 BLTZ



Format: BLTZ rs, offset

Purpose: To test a GPR then do a PC-relative conditional branch.

Description: 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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

Operation:

I : $condition \leftarrow (GPR[rs] < 0)$

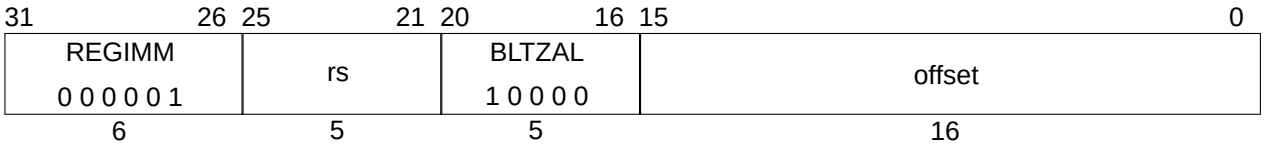
if $(condition)$ **then**

$I+1$: $PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$

end if

Exceptions: None.

1.2.13 BLTZAL



Format: BLTZAL rs, offset

Purpose: To test a GPR then do a PC-relative conditional procedure call.

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.

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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

Restrictions: GPR 31 must not be used for the source register *rs*, because such an instruction does not have the same effect when re-executed.

Operation:

I :

$$condition \leftarrow (GPR[rs] < 0)$$
$$GPR[31] \leftarrow PC + 8$$

$I+1$:

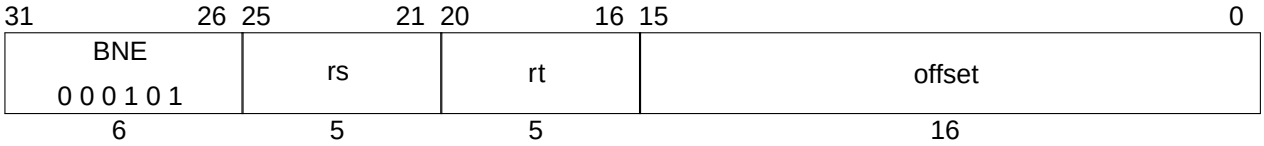
if $(condition)$ **then**

$$PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$$

end if

Exceptions: None.

1.2.14 BNE



Format: BNE rs, rt, offset

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

Description: 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 condition is true, branch to the effective target address after the instruction in the delay slot is executed.

I : $condition \leftarrow (GPR[rs] \neq GPR[rt])$

Operation:

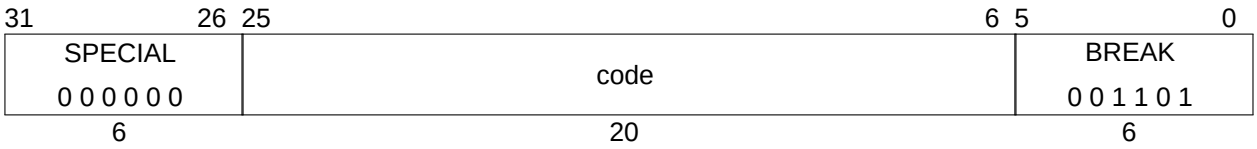
if $(condition)$ **then**

$I+1$: $PC \leftarrow PC + sign\_extend(offset \parallel 0^2)$

end if

Exceptions: None.

1.2.15 BREAK



Format: BREAK

Purpose: To cause a Breakpoint exception.

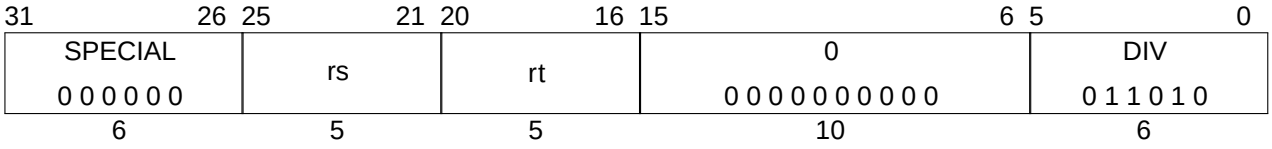
Description: A breakpoint exception occurs, immediately and unconditionally transferring control to the exception handler.

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: The *code* field value of 0x0F is reserved. Using this code will not generate exception but will cause processing unit to enter debug mode.

Exceptions: Breakpoint.

1.2.16 DIV



Format: DIV rs, rt

Purpose: To divide 32-bit signed integers.

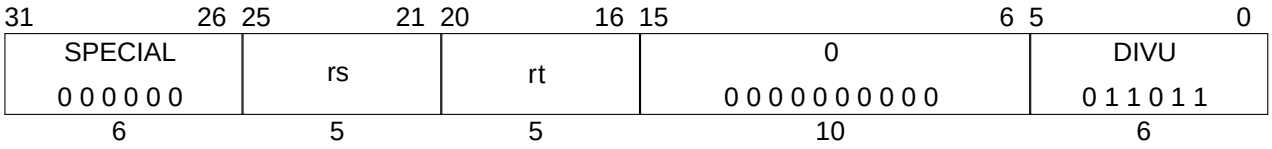
Operation: $LO \leftarrow sign\_extend(GPR[rs]\ div\ GPR[rt])$

$HI \leftarrow sign\_extend(GPR[rs]\ mod\ GPR[rt])$

Restrictions: If the divisor in GPR *rt* is zero, the arithmetic result is undefined.

Exceptions: None.

1.2.17 DIVU



Format: DIVU rs, rt

Purpose: To divide 32-bit unsigned integers.

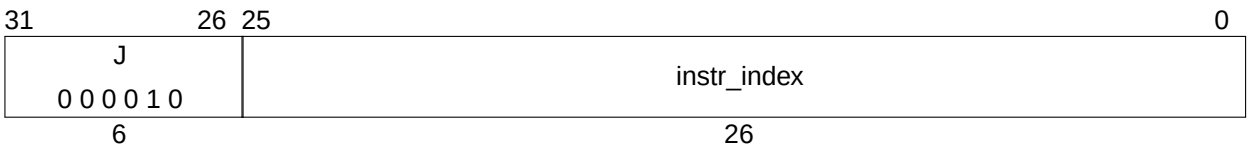
Operation: $LO \leftarrow sign\_extend(0 \parallel GPR[rs]\ div\ 0 \parallel GPR[rt])$

$HI \leftarrow sign\_extend(0 \parallel GPR[rs]\ mod\ 0 \parallel GPR[rt])$

Restrictions: If the divisor in GPR *rt* is zero, the arithmetic result is undefined.

Exceptions: None.

1.2.18 J



Format: J target

Purpose: To branch within the current 256 MB aligned region.

Description: 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).

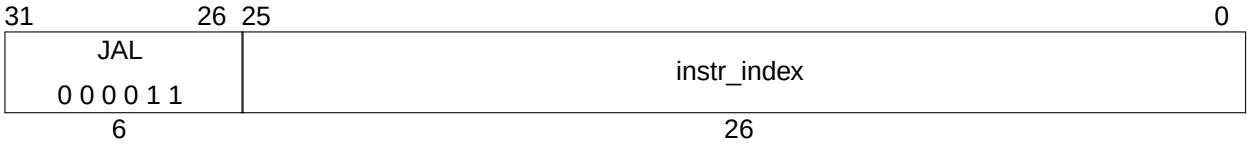
Operation:

I :

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

Exceptions: None.

1.2.19 JAL



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.

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).

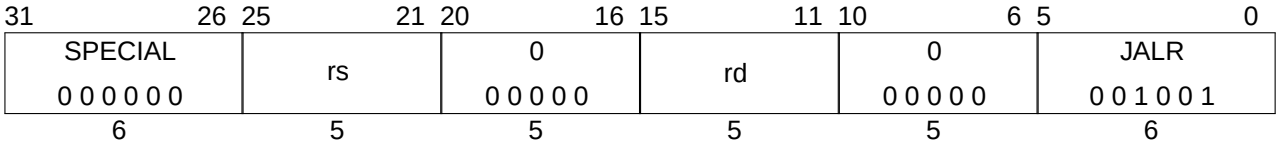
Operation:

$I:$ $GPR[31] \leftarrow PC + 8$

$I+1:$ $PC \leftarrow PC_{GPRLLEN..28} \parallel instr\_index \parallel 0^2$

Exceptions: None.

1.2.20 JALR



Format: JALR rs (rd = 31 implied)

JALR rd, rs

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

Description: 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*.

Trap if value stored in GPR *rs* is not word aligned.

Restrictions: Register specifiers *rs* and *rd* must not be equal, because such an instruction does not have the same effect when re-executed.

Operation:

I :

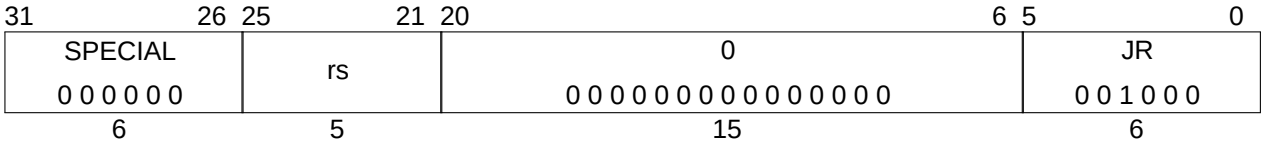
$condition \leftarrow false$
if $(rs\_word\_aligned)$ **then**
  $temp \leftarrow GPR[rs]$
  $GPR[rd] \leftarrow PC + 8$
  $condition \leftarrow true$
else
  $SignalException(AddressException)$
end if

$I+1$:

if $(condition)$ **then**
  $PC \leftarrow temp$
end if

Exceptions: Address Exception.

1.2.21 JR



Format: JR rs

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

Description: Jump to the effective target address in GPR *rs*.
Trap if value stored in GPR *rs* is not word aligned.

Operation:

I :

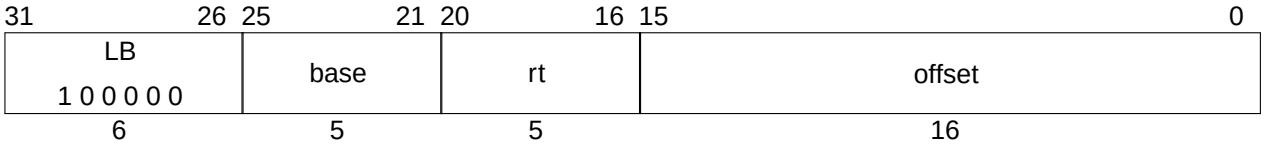
$condition \leftarrow false$
if $(rs\_word\_aligned)$ **then**
  $temp \leftarrow GPR[rs]$
  $condition \leftarrow true$
else
  $SignalException(AddressException)$
end if

$I+1$:

if $(condition)$ **then**
  $PC \leftarrow temp$
end if

Exceptions: Address Exception.

1.2.22 LB



Format: LB rt, offset(base)

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

Description: 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.

Trap if the effective address points to the reserved memory map region.

Operation:

if $(valid\_region)$ **then**

$GPR[rt] \leftarrow sign\_extend(memory[GPR[base] + offset])$

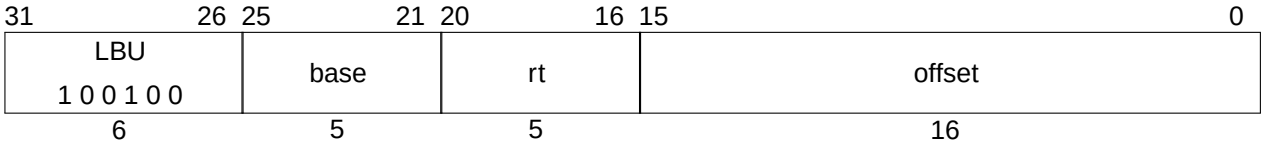
else

$SignalException(LoadException)$

end if

Exceptions: Load Exception.

1.2.23 LBU



Format: LBU rt, offset(base)

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

Description: 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.

Trap if the effective address points to the reserved memory map region.

Operation:

if $(valid\_region)$ **then**

$GPR[rt] \leftarrow zero\_extend(memory[GPR[base] + offset])$

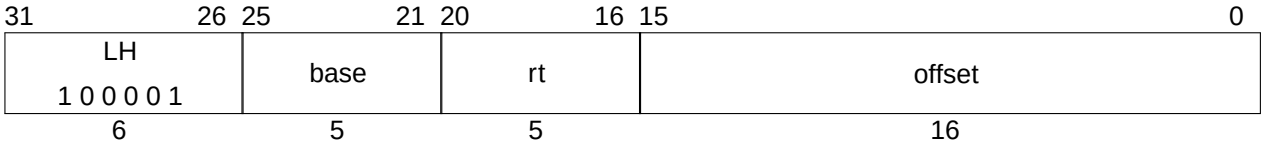
else

$SignalException(LoadException)$

end if

Exceptions: Load Exception.

1.2.24 LH



Format: LH rt, offset(base)

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

Description: The contents of the 16-bit halfword 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.

Trap if the effective address is not halfword aligned.

Trap if effective address points to the reserved memory map region.

Operation:

if $(addr\_halfword\_aligned\ \&\ valid\_region)$ **then**

$GPR[rt] \leftarrow sign\_extend(memory[GPR[base] + offset])$

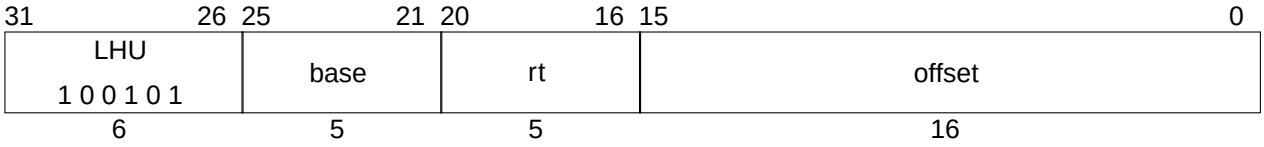
else

$SignalException(LoadException)$

end if

Exceptions: Load Exception.

1.2.25 LHU



Format: LHU rt, offset(base)

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

Description: The contents of the 16-bit halfword 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.

Trap if the effective address is not halfword aligned.

Trap if effective address points to the reserved memory map region.

Operation:

if $(addr\_halfword\_aligned\ \&\ valid\_region)$ **then**

$GPR[rt] \leftarrow zero\_extend(memory[GPR[base] + offset])$

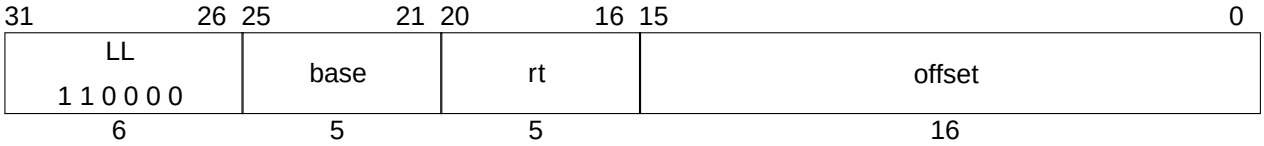
else

$SignalException(LoadException)$

end if

Exceptions: Load Exception.

1.2.26 LL



Format: LL rt, offset(base)

Purpose: To load a word from memory for an atomic read-modify-write.

Description: The 16-bit signed offset is added to the contents of GPR *base* to form the effective address.

The contents of the 32-bit word at the memory location specified by the effective address are fetched, sign-extended, and placed in GPR *rt*. This begins a RMW sequence on the current processor.

There is one active RMW sequence per processor. When an LL is executed it starts the active RMW sequence replacing any other sequence that was active.

Trap if the effective address is not word aligned.

Trap if effective address points to the reserved memory map region.

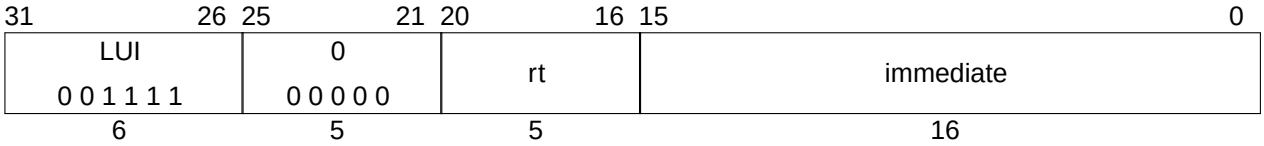
Restrictions: The addressed location must be cachable data region.

Operation:

if $(addr\_word\_aligned \ \& \ valid\_region)$ **then**
  $GPR[rt] \leftarrow sign\_extend(memory[GPR[base] + offset])$
  if $(cachable\_data\_region)$ **then**
    $LLbit \leftarrow 1$
  end if
else
  $SignalException(LoadException)$
end if

Exceptions: Load Exception.

1.2.27 LUI



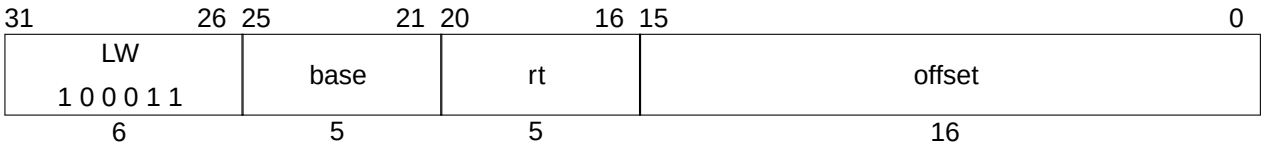
Format: LUI rt, immediate

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

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

Exceptions: None.

1.2.28 LW



Format: LW rt, offset(base)

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

Description: The contents of the 32-bit word 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.

Trap if the effective address is not word aligned.

Trap if effective address points to the reserved memory map region.

Operation:

if $(addr\_word\_aligned \ \& \ valid\_region)$ **then**

$GPR[rt] \leftarrow sign\_extend(memory[GPR[base] + offset])$

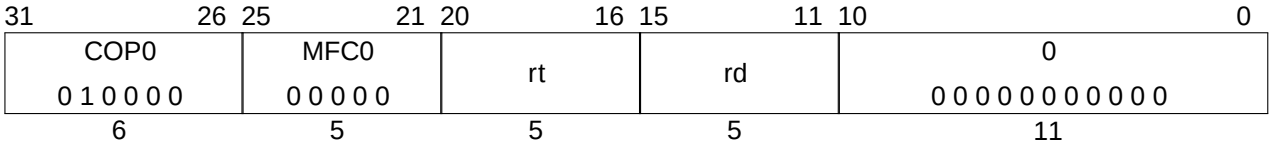
else

$SignalException(LoadException)$

end if

Exceptions: Load Exception.

1.2.29 MFC0



Format: MFC0 rt, rd

Purpose: To copy a word from an Interrupt Controller (CP0) register to a GPR.

Description: Instruction is internally decoded as load from the tightly-coupled Interrupt Controller address space.
Translation between *rs* field to Interrupt Controller register name is given below.
Trap if in user mode.

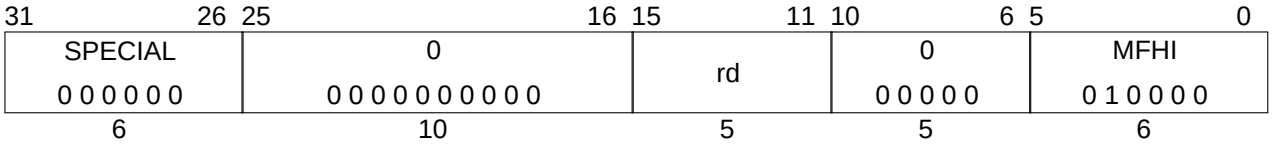
rd	ICR
8	Bad Address Register
12	Status Register
13	Exception Cause Register
14	Return Address Register

Operation:

if $(kernel\_mode)$ **then**
$GPR[rt] \leftarrow ICR[rd])$
else
$SignalException(PrivilegedData)$
end if

Exceptions: Privileged Data.

1.2.30 MFHI



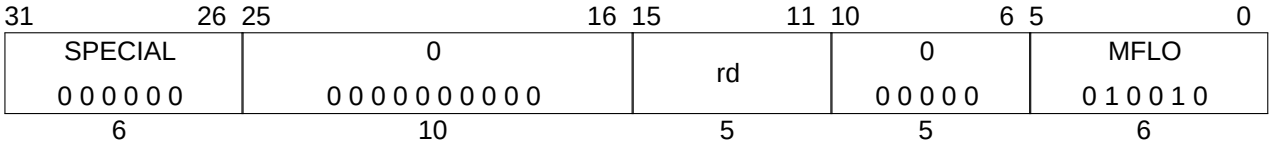
Format: MFHI rd

Purpose: To copy the special purpose HI register to a GPR.

Operation: $GPR[rd] \leftarrow HI$

Exceptions: None.

1.2.31 MFLO



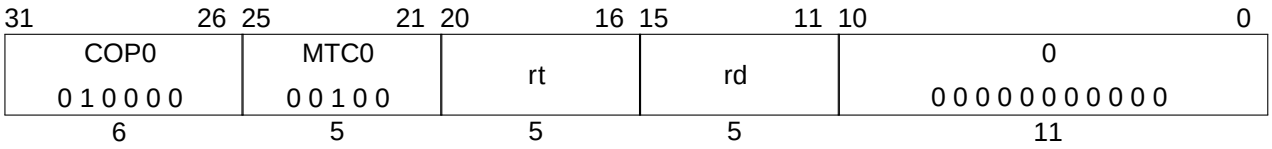
Format: MFLO rd

Purpose: To copy the special purpose LO register to a GPR.

Operation: $GPR[rd] \leftarrow LO$

Exceptions: None.

1.2.32 MTC0



Format: MTC0 rt, rd

Purpose: To copy a word from a GPR to an Interrupt Controller (CP0) register.

Description: Instruction is internally decoded as store to the tightly-coupled Interrupt Controller address space.
Translation between *rs* field to Interrupt Controller register name is given below.
Trap if in user mode.

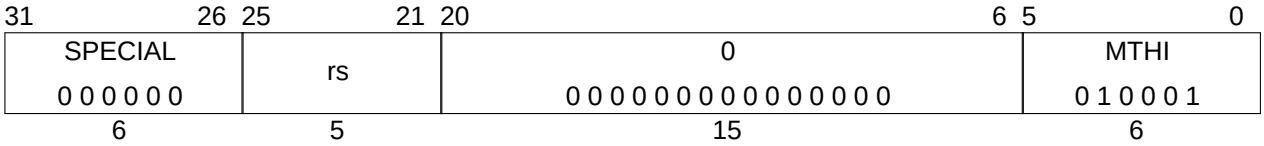
rd	ICR
8	Bad Address Register
12	Status Register
13	Exception Cause Register
14	Return Address Register

Operation:

if $(kernel\_mode)$ **then**
$ICR[rd] \leftarrow GPR[rt])$
else
$SignalException(PrivilegedData)$
end if

Exceptions: Privileged Data.

1.2.33 MTHI



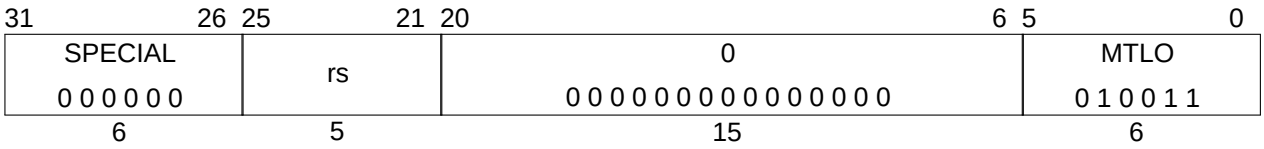
Format: MTHI rs

Purpose: To copy a GPR to the special purpose HI register.

Operation: $HI \leftarrow GPR[rs]$

Exceptions: None.

1.2.34 MTLO



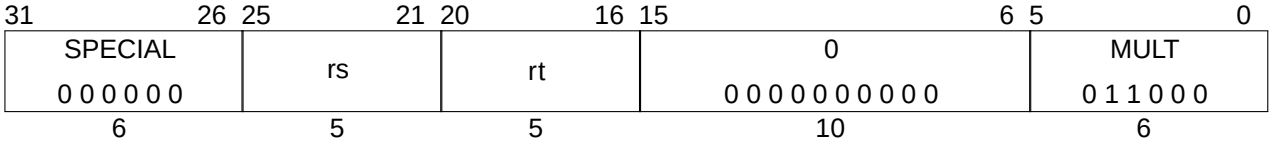
Format: MTLO rs

Purpose: To copy a GPR to the special purpose LO register.

Operation: $LO \leftarrow GPR[rs]$

Exceptions: None.

1.2.35 MULT



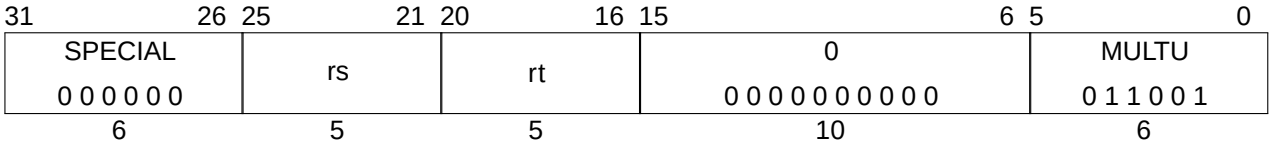
Format: MULT rs, rt

Purpose: To multiply 32-bit signed integers.

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

Exceptions: None.

1.2.36 MULTU



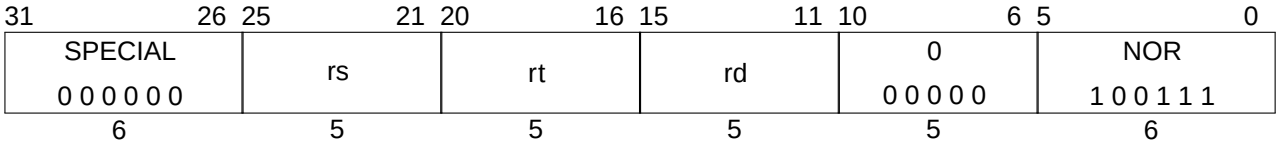
Format: MULTU rs, rt

Purpose: To multiply 32-bit unsigned integers.

Operation: $(HI, LO) \leftarrow (0 \parallel GPR[rs]) \times (0 \parallel GPR[rt])$

Exceptions: None.

1.2.37 NOR



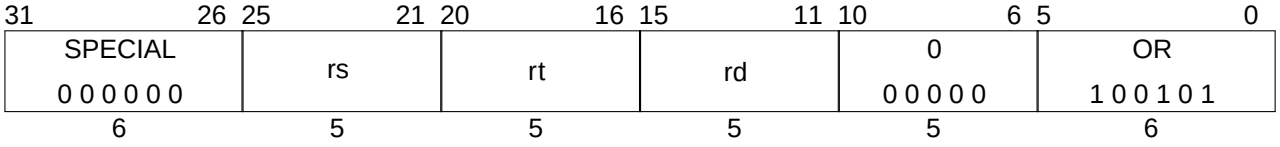
Format: NOR rd, rs, rt

Purpose: To do a bitwise logical NOT OR.

Operation: $GPR[rd] \leftarrow GPR[rs]\ NOR\ GPR[rt]$

Exceptions: None.

1.2.38 OR



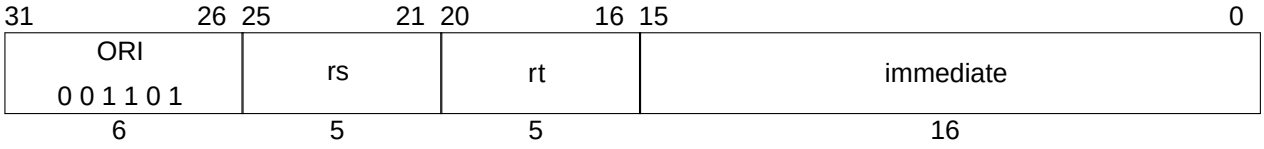
Format: OR rd, rs, rt

Purpose: To do a bitwise logical OR.

Operation: $GPR[rd] \leftarrow GPR[rs]\ OR\ GPR[rt]$

Exceptions: None.

1.2.39 ORI



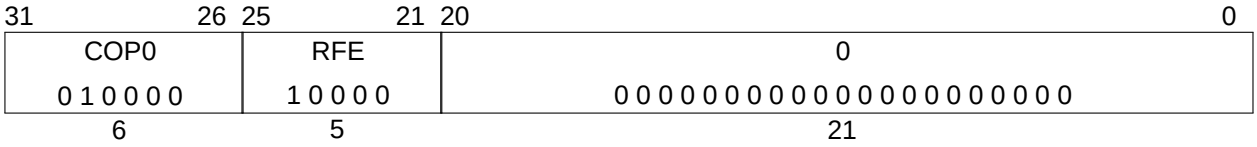
Format: ORI rt, rs, immediate

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

Operation: $rt \leftarrow GPR[rs]\ OR\ zero\_extend(immediate)$

Exceptions: None.

1.2.40 RFE



Format: RFE

Purpose: To perform return from exception.

Operation:

if $(kernel\_mode)$ **then**

$(OIEN, PIEN, CIEN) \leftarrow (0, OIEN, PIEN)$

$(OMODE, PMODE, CMODE) \leftarrow (0, OMODE, PMODE)$

$(IRQ\_HISTORY[6:0]) \leftarrow (0, IRQ\_HISTORY[6:1])$

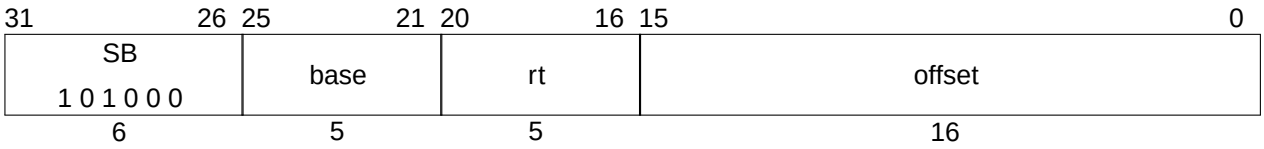
else

$SignalException(PriviledgedInstruction)$

end if

Exceptions: Privileged Instruction.

1.2.41 SB



Format: SB rt, offset(base)

Purpose: To store a byte to memory.

Description: 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.

Trap if the effective address points to the reserved memory map region.

Operation:

if $(valid\_region)$ **then**

$sign\_extend(memory[GPR[base] + offset]) \leftarrow GPR[rt][7:0]$

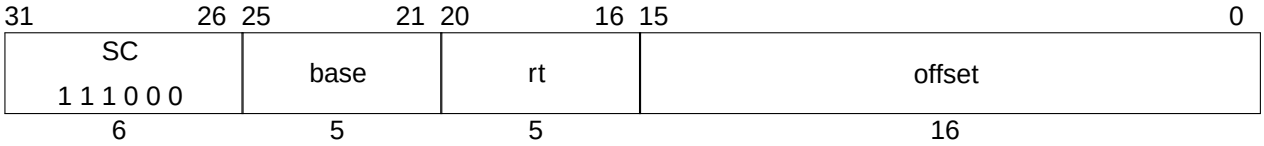
else

$SignalException(StoreException)$

end if

Exceptions: Store Exception.

1.2.42 SC



Format: SC rt, offset(base)

Purpose: To store a word to memory to complete an atomic read-modify-write.

Description: The SC completes the RMW sequence begun by the preceding LL instruction executed on the processor. If it would complete the RMW sequence atomically, then the 32-bit word of GPR *rt* is stored into memory at the location specified by the aligned effective address and a one, indicating success, is written into GPR *rt*. Otherwise, memory is not modified and a zero, indicating failure, is written into GPR *rt*.

The 16-bit signed offset is added to the contents of GPR *base* to form the effective address.

Trap if the effective address is not word aligned.

Trap if effective address points to the reserved memory map region.

The SC will fail (*LLbit* will be cleared) if a coherent store is completed between the execution of LL and SC by another processor or coherent I/O module into the block of physical memory containing the word. The size of a block is a single word for processor without data cache and a single cache line for processor with data cache.

The SC will fail if exception occurs on the processor executing the LL/SC. *LLbit* is cleared during return from the exception.

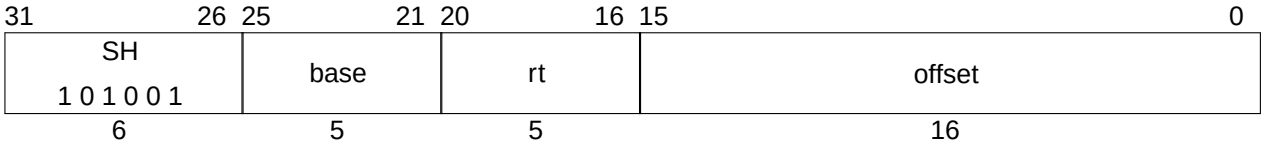
Restrictions: The addressed location must be cachable data region. Otherwise the store will not fail and the content of GPR *rt* will become undefined.

Operation:

if $(addr\_word\_aligned\ \&\ valid\_region)$ **then**
  if $LLbit$ **then**
    $sign\_extend(memory[GPR[base] + offset]) \leftarrow GPR[rt][31:0]$
    $GPR[rt] \leftarrow 0^{31} \parallel 1$
    $LLbit \leftarrow 0$
  else
    $GPR[rt] \leftarrow 0^{32}$
  end if
else
  $SignalException(StoreException)$
end if

Exceptions: Store Exception.

1.2.43 SH



Format: SH rt, offset(base)

Purpose: To store a halfword to memory.

Description: The least-significant 16-bit halfword 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.

Trap if the effective address is not halfword aligned.

Trap if the effective address points to the reserved memory map region.

Operation:

if $(addr\_halfword\_aligned\ \&\ valid\_region)$ **then**

$sign\_extend(memory[GPR[base] + offset]) \leftarrow GPR[rt][15:0]$

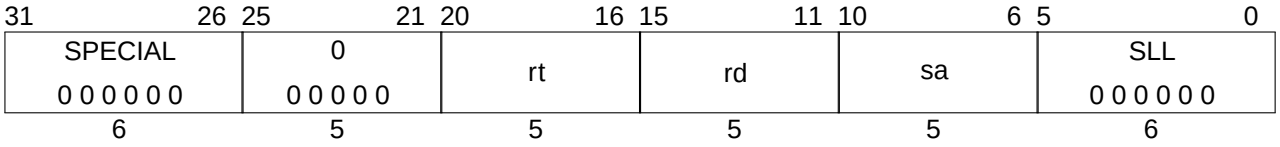
else

$SignalException(StoreException)$

end if

Exceptions: Store Exception.

1.2.44 SLL



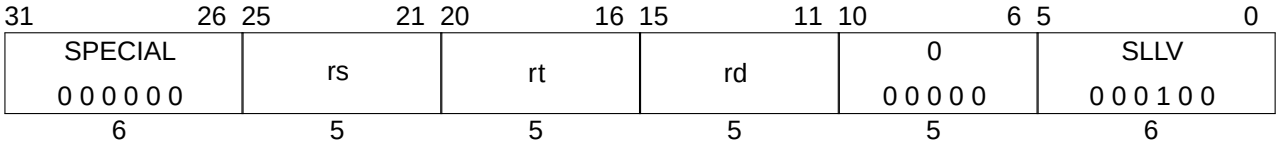
Format: SLL rd, rt, sa

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

Operation: $GPR[rd] \leftarrow sign\_extend(GPR[rt]_{31..sa} \parallel 0^{sa})$

Exceptions: None.

1.2.45 SLLV



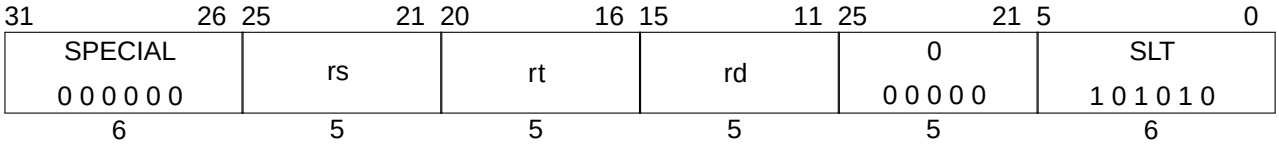
Format: SLLV rd, rt, rs

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

Operation: $GPR[rd] \leftarrow sign\_extend(GPR[rt]_{31..GPR[rs]_{4..0}} \parallel 0^{GPR[rs]^{4..0}})$

Exceptions: None.

1.2.46 SLT



Format: SLT rd, rs, rt

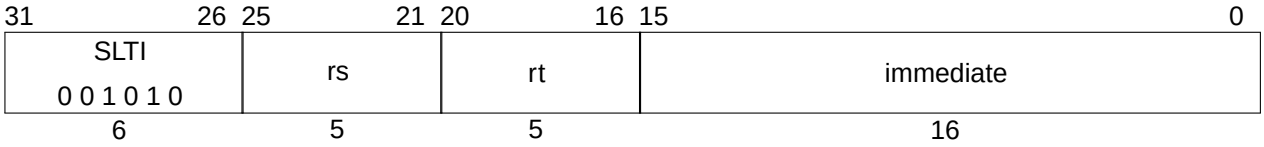
Purpose: To record the result of a less-than comparison.

Operation:

if $(GPR[rs] < GPR[rt])$ **then**
$GPR[rd] \leftarrow 0^{31} \parallel 1$
else
$GPR[rd] \leftarrow 0^{32}$
end if

Exceptions: None.

1.2.47 SLTI



Format: SLTI rt, rs, immediate

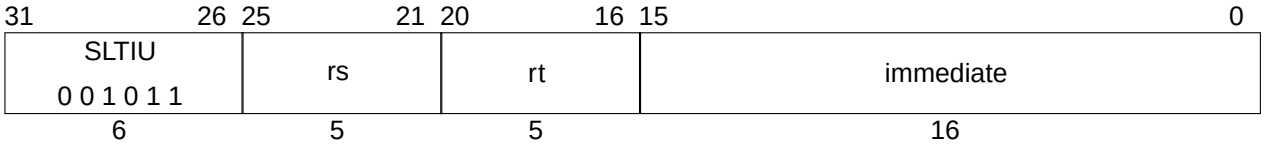
Purpose: To record the result of a less-than comparison with a constant.

Operation:

if $(GPR[rs] < sign\_extend(immediate))$ **then**
$GPR[rd] \leftarrow 0^{31} \parallel 1$
else
$GPR[rd] \leftarrow 0^{32}$
end if

Exceptions: None.

1.2.48 SLTIU



Format: SLTIU rt, rs, immediate

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

Operation:

if $((0 \parallel GPR[rs]) < (0 \parallel sign\_extend(immediate)))$ **then**

$GPR[rd] \leftarrow 0^{31} \parallel 1$

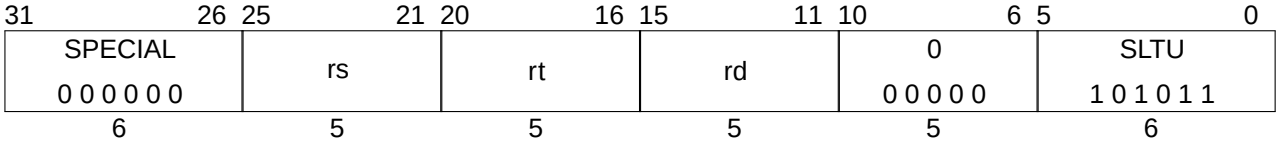
else

$GPR[rd] \leftarrow 0^{32}$

end if

Exceptions: None.

1.2.49 SLTU



Format: SLTU rd, rs, rt

Purpose: To record the result of an unsigned less-than comparison.

Operation:

if $((0 \parallel GPR[rs]) < (0 \parallel GPR[rt]))$ **then**

$rd \leftarrow 0^{31} \parallel 1$

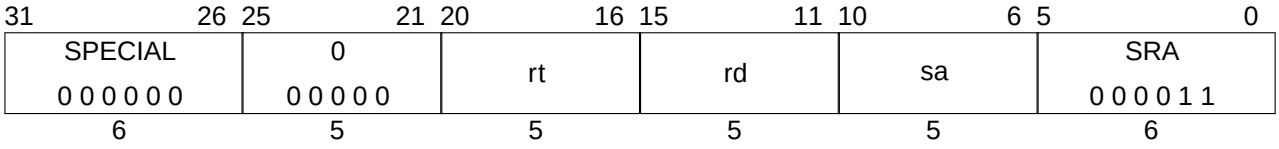
else

$rd \leftarrow 0^{32}$

end if

Exceptions: None.

1.2.50 SRA



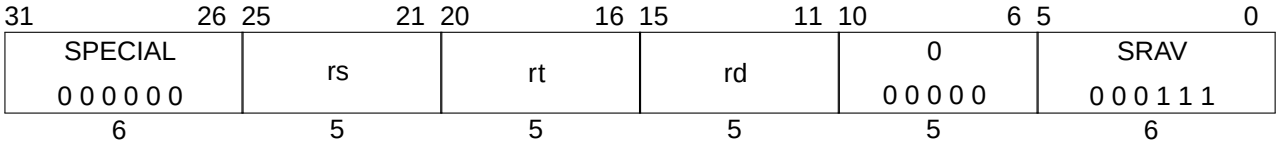
Format: SRA rd, rt, sa

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

Operation: $GPR[rd] \leftarrow sign\_extend(GPR[rt]_{31}^{sa}) \parallel GPR[rt]_{31..sa}$

Exceptions: None.

1.2.51 SRAV



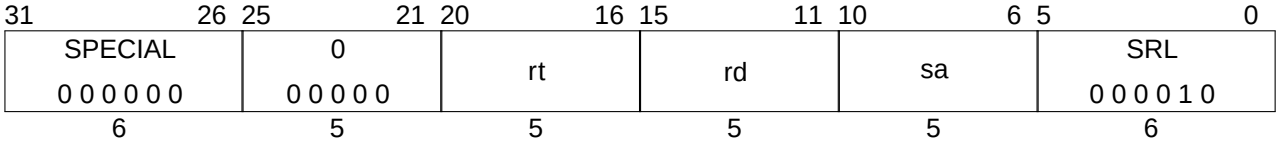
Format: SRAV rd, rt, rs

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

Operation: $GPR[rd] \leftarrow sign\_extend(GPR[rt]_{31}^{GPR[rs]_{4..0}}) \parallel GPR[rt]_{31..GPR[rs]_{4..0}}$

Exceptions: None.

1.2.52 SRL



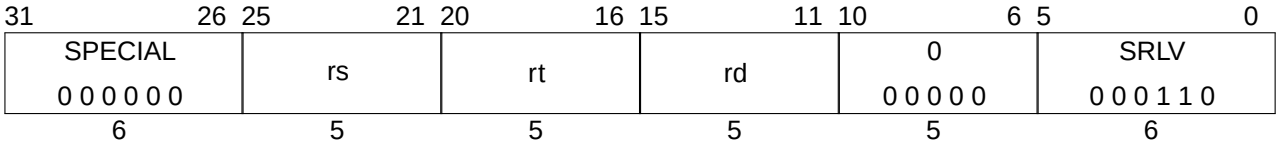
Format: SRL rd, rt, sa

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

Operation: $GPR[rd] \leftarrow sign\_extend(0^{sa}) \parallel GPR[rt]_{31..sa}$

Exceptions: None.

1.2.53 SRLV



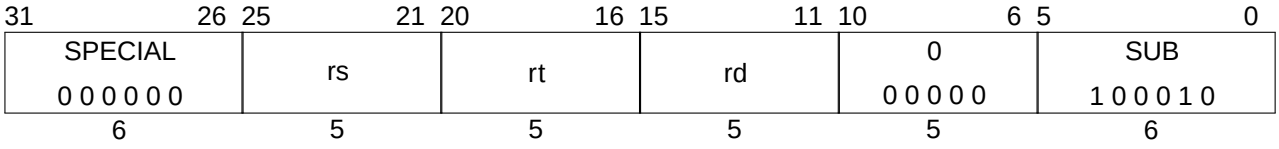
Format: SRLV rd, rt, rs

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

Operation: $GPR[rd] \leftarrow sign\_extend(0^{GPR[rs]_{4..0}}) \parallel GPR[rt]_{31..GPR[rs]_{4..0}}$

Exceptions: None.

1.2.54 SUB



Format: SUB rd, rs, rt

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

Operation:

if $(32\_bit\_arithmetic\_overflow)$ **then**

$SignalException(IntegerOverflow)$

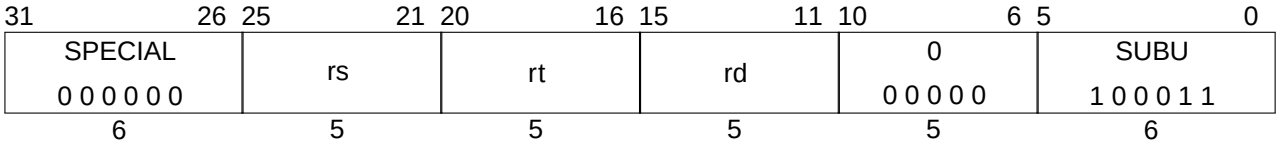
else

$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$

end if

Exceptions: Integer Overflow.

1.2.55 SUBU



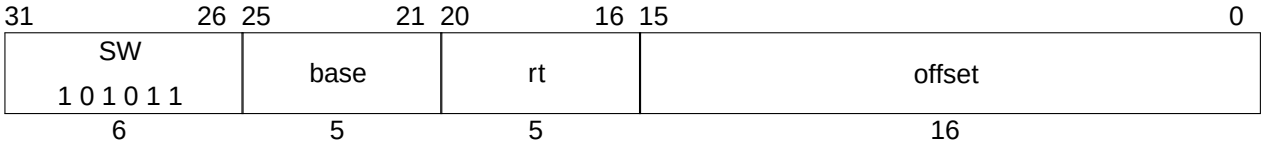
Format: SUBU rd, rs, rt

Purpose: To subtract 32-bit integers.

Operation: $GPR[rd] \leftarrow GPR[rs] - GPR[rt]$

Exceptions: None.

1.2.56 SW



Format: SW rt, offset(base)

Purpose: To store a word to memory.

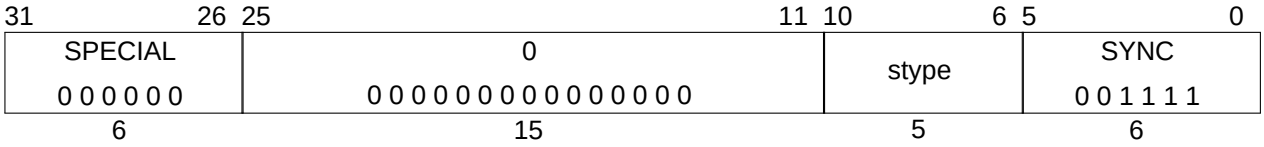
Description: The 32-bit word of GPR *rt* is stored in memory at the location specified by the effective address.
The 16-bit signed offset is added to the contents of GPR *base* to form the effective address.
Trap if the effective address is not word aligned.
Trap if the effective address points to the reserved memory map region.

Operation:

if $(addr\_word\_aligned\ \&\ valid\_region)$ **then**
$sign\_extend(memory[GPR[base] + offset]) \leftarrow GPR[rt][31:0]$
else
$SignalException(StoreException)$
end if

Exceptions: Store Exception.

1.2.57 SYNC



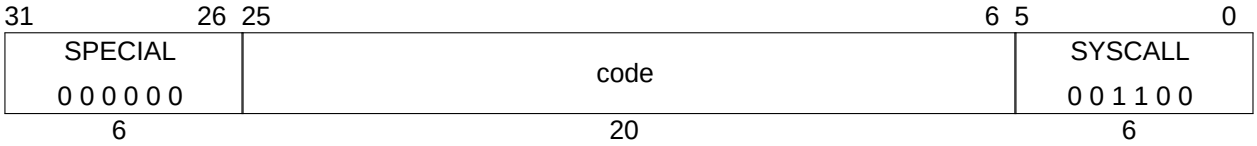
Format: SYNC (stype = 0 implied)

Purpose: To order loads and stores to shared memory in a multiprocessor system.

Description: Instruction stalls the processor pipeline till all its stores are completed. Stores are completed when the stored value is visible to every other processor in the system.
The *stype* values 1-31 are reserved; they produce the same result as the value zero.

Exceptions: None.

1.2.58 SYSCALL



Format: SYSCALL

Purpose: To cause a System Call exception.

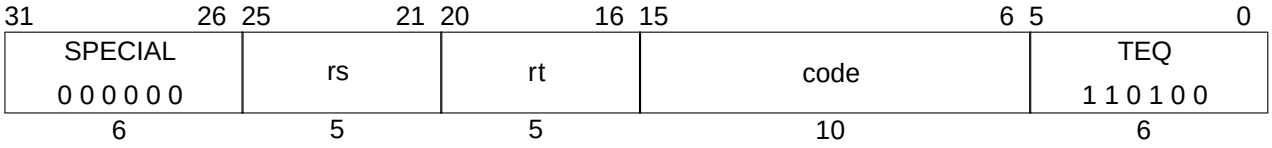
Description: A system call exception occurs, immediately and unconditionally transferring control to the exception handler.

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: In kernel mode the *code* field value of 0x0F is reserved and will not generate exception. In kernel mode the 0x0F code is recognized by the Multicore Controller and starts procedure of stopping cores other than Core 0.

Exceptions: Breakpoint.

1.2.59 TEQ



Format: TEQ rs, rt

Purpose: To compare GPRs and do a conditional Trap.

Description: Compare the contents of GPR *rs* and GPR *rt* as signed integers; if GPR *rs* is equal to GPR *rt* then take a Trap exception.

The contents of the *code* field are ignored by hardware and may be used to encode information for system software. To retrieve the information, system software must load the instruction word from memory.

Operation:

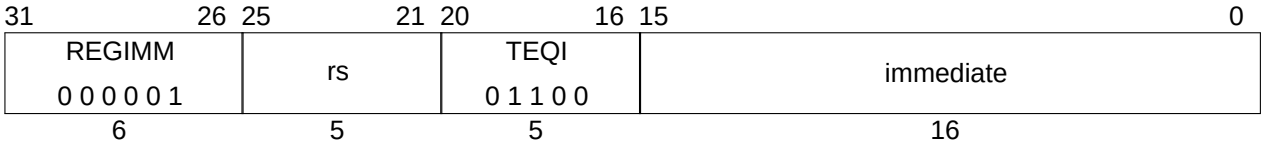
if $(GPR[rs] = GPR[rt])$ **then**

$SignalException(Trap)$

end if

Exceptions: Trap.

1.2.60 TEQI



Format: TEQI rs, immediate

Purpose: To compare a GPR to a constant and do a conditional Trap.

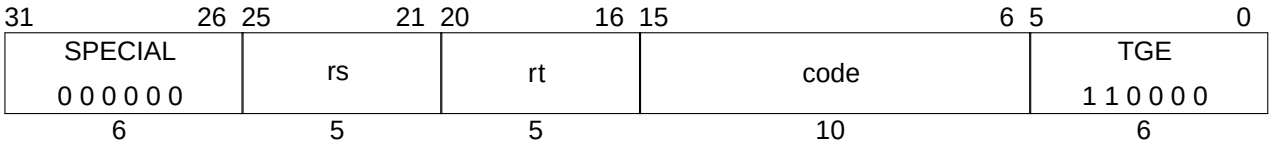
Description: Compare the contents of GPR *rs* and the 16-bit signed *immediate* as signed integers; if GPR *rs* is equal to *immediate* then take a Trap exception.

Operation:

if $(GPR[rs] = sign\_extend(immediate))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.61 TGE



Format: TGE rs, rt

Purpose: To compare GPRs and do a conditional Trap.

Description: Compare the contents of GPR *rs* and GPR *rt* as signed integers; if GPR *rs* is greater than or equal to GPR *rs* then take a Trap exception.

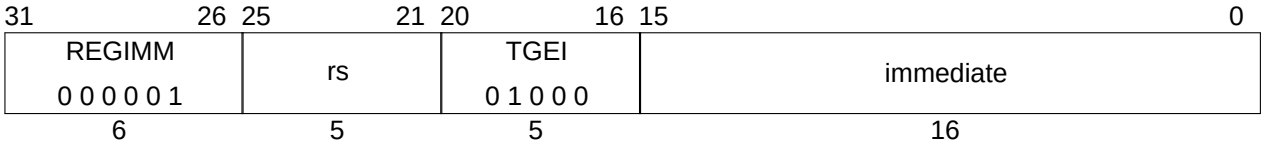
The contents of the *code* field are ignored by hardware and may be used to encode information for system software. To retrieve the information, system software must load the instruction word from memory.

Operation:

if $(GPR[rs] \geq GPR[rt])$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.62 TGEI



Format: TGEI rs, immediate

Purpose: To compare a GPR to a constant and do a conditional Trap.

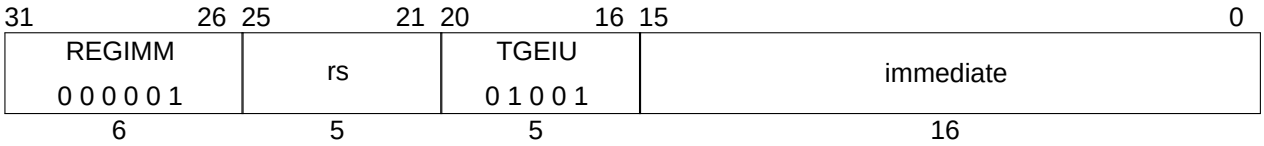
Description: Compare the contents of GPR *rs* and the 16-bit signed *immediate* as signed integers; if GPR *rs* is greater than or equal to *immediate* then take a Trap exception.

Operation:

if $(GPR[rs] \geq sign\_extend(immediate))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.63 TGEIU



Format: TGEIU rs, immediate

Purpose: To compare a GPR to a constant and do a conditional Trap.

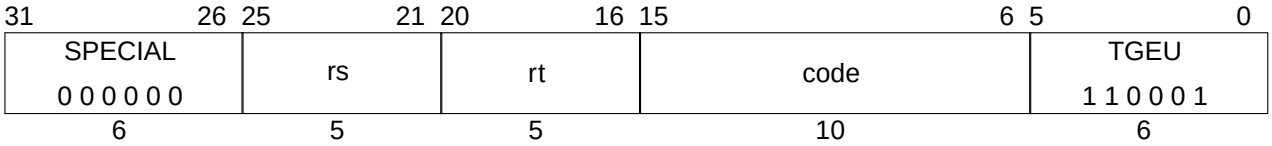
Description: Compare the contents of GPR *rs* and the 16-bit sign-extended *immediate* as unsigned integers; if GPR *rs* is greater than or equal to immediate then take a Trap exception.

Operation:

if $((0 \parallel GPR[rs]) \geq (0 \parallel sign\_extend(immediate)))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.64 TGEU



Format: TGEU rs, rt

Purpose: To compare GPRs and do a conditional Trap.

Description: Compare the contents of GPR *rs* and GPR *rt* as unsigned integers; if GPR *rs* is greater than or equal to GPR *rt* then take a Trap exception.

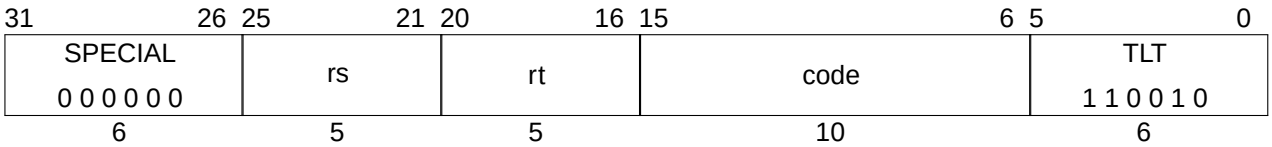
The contents of the *code* field are ignored by hardware and may be used to encode information for system software. To retrieve the information, system software must load the instruction word from memory.

Operation:

if $((0 \parallel GPR[rs]) \geq (0 \parallel GPR[rt]))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.65 TLT



Format: TLT rs, rt

Purpose: To compare GPRs and do a conditional Trap.

Description: Compare the contents of GPR *rs* and GPR *rt* as signed integers; if GPR *rs* is less than GPR *rt* then take a Trap exception.

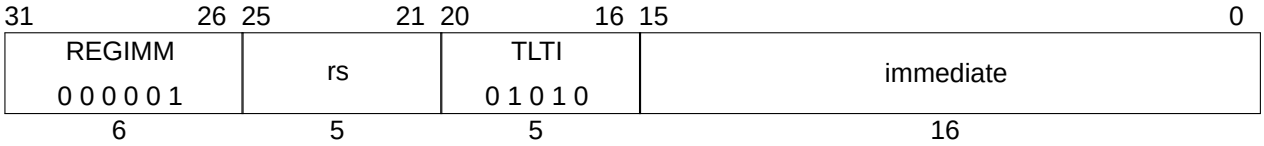
The contents of the *code* field are ignored by hardware and may be used to encode information for system software. To retrieve the information, system software must load the instruction word from memory.

Operation:

if $(GPR[rs] < GPR[rt])$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.66 TLTI



Format: TLTI rs, immediate

Purpose: To compare a GPR to a constant and do a conditional Trap.

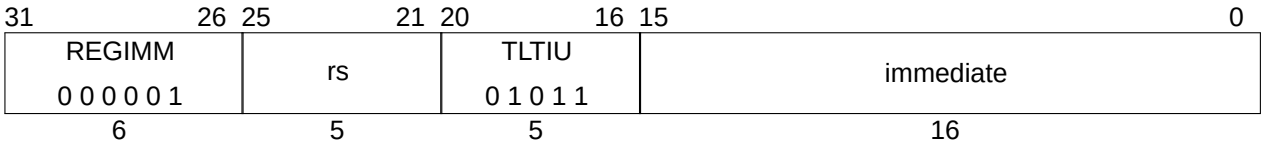
Description: Compare the contents of GPR *rs* and the 16-bit signed *immediate* as signed integers; if GPR *rs* is less than *immediate* then take a Trap exception.

Operation:

if $(GPR[rs] < sign\_extend(immediate))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.67 TLTIU



Format: TLTIU rs, immediate

Purpose: To compare a GPR to a constant and do a conditional Trap.

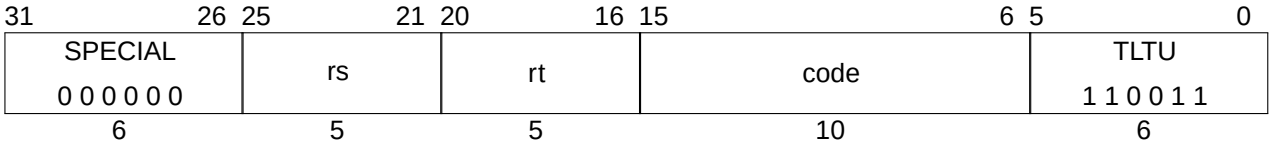
Description: Compare the contents of GPR *rs* and the 16-bit sign-extended *immediate* as unsigned integers; if GPR *rs* is less than *immediate* then take a Trap exception.

Operation:

if $((0 \parallel GPR[rs]) < (0 \parallel sign\_extend(immediate)))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.68 TLTU



Format: TLTU rs, rt

Purpose: To compare GPRs and do a conditional Trap.

Description: Compare the contents of GPR *rs* and GPR *rt* as unsigned integers; if GPR *rs* is less than GPR *rs* then take a Trap exception.

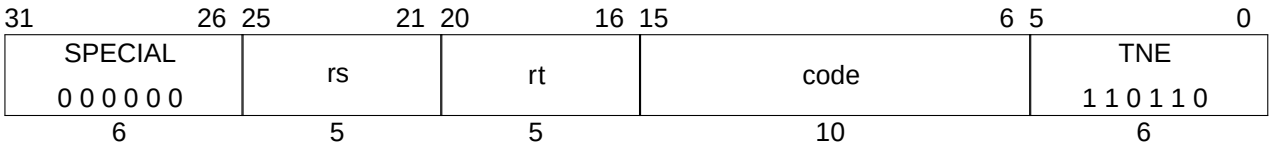
The contents of the *code* field are ignored by hardware and may be used to encode information for system software. To retrieve the information, system software must load the instruction word from memory.

Operation:

if $((0 \parallel GPR[rs]) < (0 \parallel GPR[rt]))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.69 TNE



Format: TNE rs, rt

Purpose: To compare GPRs and do a conditional Trap.

Description: Compare the contents of GPR *rs* and GPR *rt* as signed integers; if GPR *rs* is not equal to GPR *rs* then take a Trap exception.

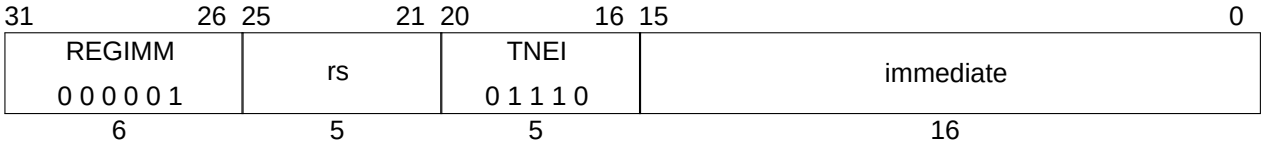
The contents of the *code* field are ignored by hardware and may be used to encode information for system software. To retrieve the information, system software must load the instruction word from memory.

Operation:

if $(GPR[rs] \neq GPR[rt])$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.70 TNEI



Format: TNEI rs, immediate

Purpose: To compare a GPR to a constant and do a conditional Trap.

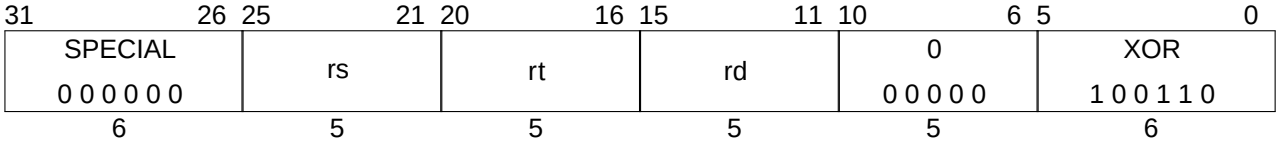
Description: Compare the contents of GPR *rs* and the 16-bit signed *immediate* as signed integers; if GPR *rs* is not equal to *immediate* then take a Trap exception.

Operation:

if $(GPR[rs] \neq sign\_extend(immediate))$ **then**
$SignalException(Trap)$
end if

Exceptions: Trap.

1.2.71 XOR



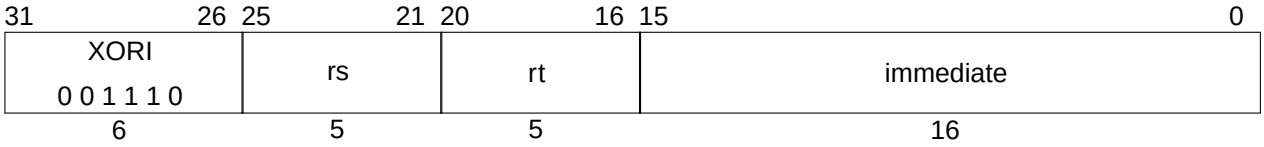
Format: XOR rd, rs, rt

Purpose: To do a bitwise logical EXCLUSIVE OR.

Operation: $GPR[rd] \leftarrow GPR[rs]\ XOR\ GPR[rt]$

Exceptions: Trap.

1.2.72 XORI



Format: XORI rt, rs, immediate

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

Operation: $GPR[rt] \leftarrow GPR[rs]\ XOR\ zero\_extend(immediate)$

Exceptions: None.

ChipCraft Sp. z o.o. Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND www.chipcraft-ic.com

CC100-ISA-Doc_102017.