ALLEGREX™
Instruction Manual

Table of Contents

MIPS Instructions

add

Add

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 1 0 0 0 0 0
6	5	5	5	11

MIPS I

Syntax:

```
add  rd,rs,rt
```

Description:

The contents of registers rs and rt are added together and the result is stored in register rd. If a two's complement overflow occurs, an exception is generated.

Operation:

```
temp $\leftarrow$ (GPR[rs]$_{31}$ || GPR[rs]$_{31..0}$) + (GPR[rt]$_{31}$ || GPR[rt]$_{31..0}$)
if (temp$_{32}$ $\neq$ temp$_{31}$) then
    SignalException(IntegerOverflow)
else
    GPR[rd] $\leftarrow$ temp
endif
```

Exceptions:

```
Integer Overflow exception
```

addi

Add Immediate

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 0 0	rs	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
addi  rt,rs,immediate
```

Description:

The 16-bit immediate field is sign-extended to 32 bits and added to register rs. The 32-bit result is stored in register rt.

If a two’s complement overflow occurs, an exception is generated.

Operation:

```
temp ← (GPR[rs]$_{31}$ || GPR[rs]$_{31..0}$) + sign_extend(immediate)
if (temp$_{32}$ $\neq$ temp$_{31}$) then
    SignalException(IntegerOverflow)
else
    GPR[rt] ← temp
endif
```

Exceptions:

```
Integer Overflow exception
```

addiu

Add Immediate Unsigned

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 0 1	rs	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
addiu  rt,rs,immediate
```

Description:

The 16-bit immediate field is sign-extended to 32 bits and added to register rs. The 32-bit result is stored in register rt.

No exception is generated even if an overflow occurs.

Operation:

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

Exceptions:

None

addu

Add Unsigned

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 1 0 0 0 0 1
6	5	5	5	11

MIPS I

Syntax:

```
addu  rd,rs,rt
```

Description:

The contents of registers rs and rt are added together and the result is stored in register rd.

No exception is generated even if a two's complement overflow occurs.

Operation:

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

Exceptions:

None

and

AND

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 1 0 0 1 0 0
6	5	5	5	11

MIPS I

Syntax:

```
and  rd,rs,rt
```

Description:

A bitwise AND (logical product) is performed for the contents of registers rs and rt and the result is stored in register rd.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Operation:

GPR[rd] ← GPR[rs] and GPR[rt]

Exceptions:

None

andi

AND Immediate

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 0 0	rs	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
andi  rt,rs,immediate
```

Description:

The 16-bit immediate field is zero-extended, and a bitwise AND (logical product) is performed between the zero-extended value and the contents of register rs. The result is stored in register rt.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Operation:

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

Exceptions:

None

bc1f

Branch on FPU False

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 1 0 0 0	0 0 0 0 0	offset
6	10		16

MIPS I

FPU

Syntax:

```
bc1f  offset
```

Description:

When the FPU condition is false, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the bc1f instruction.

A nop is automatically inserted by the compiler.

The bc1f instruction cannot be placed in the delay slot of a branch/jump instruction.

The ctc1 instruction cannot be placed in the delay slot of the bc1f instruction.

Operation:

```
I – 1:
    condition ← notCOC[1]
I :
    target_offset ← sign_extend(offset || $0^2$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

```
Coprocessor Unusable exception
```

bc1fl

Branch on FPU False Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 1 0 0 0	0 0 0 1 0	offset
6	10		16

MIPS II

FPU

Syntax:

```
bc1fl  offset
```

Description:

When the FPU condition is false, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the bc1fl instruction.

A nop is automatically inserted by the compiler.

The bc1fl instruction cannot be placed in the delay slot of a branch/jump instruction.

The ctc1 instruction cannot be placed in the delay slot of the bc1fl instruction.

Operation:

```
I – 1:
    condition ← notCOC[1]
I :
    target_offset ← sign_extend(offset || $0^2$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

```
Coprocessor Unusable exception
```

bc1t

Branch on FPU True

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 1 0 0 0	0 0 0 0 1	offset
6	10		16

MIPS I

FPU

Syntax:

```
bc1t  offset
```

Description:

When the FPU condition is true, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the bc1t instruction.

A nop is automatically inserted by the compiler.

The bc1t instruction cannot be placed in the delay slot of a branch/jump instruction.

The ctc1 instruction cannot be placed in the delay slot of the bc1t instruction.

Operation:

```
I − 1:
    condition ← COC[1]
I :
    target_offset ← sign_extend(offset || $0^2$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

```
Coprocessor Unusable exception
```

bc1tl

Branch on FPU True Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 1 0 0 0	0 0 0 1 1	offset
6	10		16

MIPS II

FPU

Syntax:

```
bc1tl  offset
```

Description:

When the FPU condition is true, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

To change the FPU condition, at least one nop is required between the FPU instruction that changes the condition and the bc1tl instruction.

A nop is automatically inserted by the compiler.

The bc1tl instruction cannot be placed in the delay slot of a branch/jump instruction.

The ctc1 instruction cannot be placed in the delay slot of the bc1tl instruction.

Operation:

```
I – 1:
    condition ← COC[1]
I :
    target_offset ← sign_extend(offset || $0^2$)
I +1:
    if (condition) then
        PC ← PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

```
Coprocessor Unusable exception
```

beq

Branch on Equal

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 0	rs	rt	offset
6	5	5	16

MIPS I

Syntax:

```
beq  rs,rt,offset
```

Description:

When register rs is equal to register rt, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

The beq instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

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

Exceptions:

None

beql

Branch on Equal Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 1 0 0	rs	rt	offset
6	5	5	16

MIPS II

Syntax:

```
beql  rs,rt,offset
```

Description:

When register rs is equal to register rt, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The beql instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

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

Exceptions:

None

bgez

Branch on Greater than or Equal to Zero

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	0 0 0 0 1	offset
6	5	5	16

MIPS I

Syntax:

```
bgez  rs,offset
```

Description:

When register rs is greater than or equal to zero, the program branches with a one instruction delay to the branch target address.
The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

The bgez instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] ≥ $0^{32}$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

None

bgezal

Branch on Greater than or Equal to Zero And Link

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	1 0 0 0 1	offset
6	5	5	16

MIPS I

Syntax:

```
bgezal  rs,offset
```

Description:

When register rs is greater than or equal to zero, the program branches with a one instruction delay to the branch target address.
The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.
The address of the instruction following the delay slot is stored in register r31 (the link register).

Restrictions:

The bgezal instruction cannot be placed in the delay slot of a branch/jump instruction.
The register r31 (link register) cannot be specified for the register rs.
The instruction for changing the register r31 (link register) cannot be placed in the delay slot of the bgezal instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] ≥ $0^{32}$)
    GPR[31] ← PC + 8
I + 1 :
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

None

bgezall

Branch on Greater than or Equal to Zero And Link Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	1 0 0 1 1	offset
6	5	5	16

MIPS II

Syntax:

```
bgezall  rs,offset
```

Description:

When register rs is greater than or equal to zero, the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. The address of the instruction following the delay slot is stored in register r31 (the link register). If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The bgezall instruction cannot be placed in the delay slot of a branch/jump instruction.
The register r31 (link register) cannot be specified for the register rs.
The instruction for changing the register r31 (link register) cannot be placed in the delay slot of the bgezall instruction.

Operation:

```
I :
    target offset ← sign_extend(offset || $0^{2}$)
    condition ← (GPR[rs] $\geq 0^{32}$)
    GPR[31] ← PC + 8
I + 1 :
    if (condition) then
        PC ← PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

```
None
```

bgezl

Branch on Greater than or Equal to Zero Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	0 0 0 1 1	offset
6	5	5	16

MIPS II

Syntax:

```
bgezl  rs,offset
```

Description:

When register rs is greater than or equal to zero, the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The bgezl instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] $\geq 0^{32}$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

None

bgtz

Branch on Greater Than Zero

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 1 1	rs	0 0 0 0 0	offset
6	5	5	16

MIPS I

Syntax:

```
bgtz  rs,offset
```

Description:

When register rs is greater than zero, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

The bgtz instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] > $0^{32}$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

None

bgtzl

Branch on Greater Than Zero Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 1 1 1	rs	0 0 0 0 0	offset
6	5	5	16

MIPS II

Syntax:

```
bgtzl  rs,offset
```

Description:

When register rs is greater than zero, the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The bgtzl instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] > $0^{32}$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

None

blez

Branch on Less than or Equal to Zero

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 1 0	rs	0 0 0 0 0	offset
6	5	5	16

MIPS I

Syntax:

```
blez  rs,offset
```

Description:

When register rs is less than or equal to zero, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

The blez instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] $\leq 0^{32}$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

None

blezl

Branch on Less than or Equal to Zero Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 1 1 0	rs	0 0 0 0 0	offset
6	5	5	16

MIPS II

Syntax:

```
blezl  rs,offset
```

Description:

When register rs is less than or equal to zero, the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The blezl instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset $\leftarrow$ sign_extend(offset || $0^2$)
    condition $\leftarrow$ (GPR[rs] $\leq 0^{32}$)
I + 1:
    if (condition) then
        PC $\leftarrow$ PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

None

bltz

Branch on Less Than Zero

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	0 0 0 0 0	offset
6	5	5	16

MIPS I

Syntax:

```
bltz  rs,offset
```

Description:

When register rs is less than zero, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

The bltz instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] < $0^{32}$)
I + 1 :
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

None

bltzal

Branch on Less Than Zero And Link

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	1 0 0 0 0	offset
6	5	5	16

MIPS I

Syntax:

```
bltzal  rs,offset
```

Description:

When register rs is less than zero, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

The address of the instruction following the delay slot is stored in register r31 (the link register).

Restrictions:

The bltzal instruction cannot be placed in the delay slot of a branch/jump instruction.

The register r31 (link register) cannot be specified for the register rs.

The instruction for changing the register r31 (link register) cannot be placed in the delay slot of the bltzal instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^{2}$)
    condition ← (GPR[rs] < $0^{32}$)
    GPR[31] ← PC + 8
I + 1 :
    if (condition) then
        PC ← PC + target_offset
    endif
```

Exceptions:

None

bltzall

Branch on Less Than Zero And Link Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	1 0 0 1 0	offset
6	5	5	16

MIPS II

Syntax:

```
bltzall  rs,offset
```

Description:

When register rs is less than zero, the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. The address of the instruction following the delay slot is stored in register r31 (the link register). If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The bltzall instruction cannot be placed in the delay slot of a branch/jump instruction.
The register r31 (link register) cannot be specified for the register rs.
The instruction for changing the register r31 (link register) cannot be placed in the delay slot of the bltzall instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^2$)
    condition ← (GPR[rs] < $0^{32}$)
    GPR[31] ← PC + 8
I + 1 :
    if (condition) then
        PC ← PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

None

bltzl

Branch on Less Than Zero Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1	rs	0 0 0 1 0	offset
6	5	5	16

MIPS II

Syntax:

```
bltzl  rs,offset
```

Description:

When register rs is less than zero, the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The bltzl instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

```
I :
    target_offset ← sign_extend(offset || $0^{2}$)
    condition ← (GPR[rs] < $0^{32}$)
I + 1:
    if (condition) then
        PC ← PC + target_offset
    else
        NullifyCurrentInstruction()
    endif
```

Exceptions:

None

bne

Branch on Not Equal

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 1	rs	rt	offset
6	5	5	16

MIPS I

Syntax:

```
bne  rs,rt,offset
```

Description:

When register rs is not equal to register rt, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

Restrictions:

The BNE instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

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

Exceptions:

None

bnel

Branch on Not Equal Likely

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 1 0 1	rs	rt	offset
6	5	5	16

MIPS II

Syntax:

```
bnel  rs,rt,offset
```

Description:

When register rs is not equal to register rt, the program branches with a one instruction delay to the branch target address.

The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

If the branch is not taken, the instruction in the branch delay slot is discarded.

Restrictions:

The BNEL instruction cannot be placed in the delay slot of a branch/jump instruction.

Operation:

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

Exceptions:

None

break

Break Point

31 30 29 28 27 26	25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6	5 4 3 2 1 0
0 0 0 0 0 0	code	0 0 1 1 0 1
6	20	6

MIPS I

Syntax:

```
break  code
```

Description:

A breakpoint exception is generated and control is immediately passed to the exception handler.

The code field can contain any arbitrary value.

Operation:

SignalException(Breakpoint)

Exceptions:

```
Break Point exception
```

cfc1

Move Control from FPU

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 0 0 1 0	rt	fs	0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

MIPS I

FPU

Syntax:

```
cfc1  rt,fs
```

Description:

The contents of FPU control register fs are transferred to CPU register rt. This instruction is only defined when fs is 0 or 31.

Restrictions:

Only 0 or 31 is valid for fs.

Operation:

```
I :
    temp ← FCR[fs]
I + 1:
    GPR[rt] ← temp
```

Exceptions:

```
Coprocessor Unusable exception
```

ctc1

Move Control to FPU

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 0 1 1 0	rt	fs	0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

MIPS I

FPU

Syntax:

```
ctc1  rt,fs
```

Description:

The contents of CPU register rt are transferred to FPU control register fs.

This instruction is only defined when fs is 31.

A Floating-Point exception will occur if any Cause bit and its corresponding Enable bit of FCR31 get set to 1 as a result of executing this instruction.

After this instruction completes, the FPU pipeline will stall until the updated control register settings are reflected.

Restrictions:

Only 31 is valid for fs.

The ctc1 instruction cannot be executed continuously after the ctc1 instruction which causes an exception.

After executing the ctc1 instruction, at least one nop is required until executing an instruction for referring to FCR31.

Operation:

```
I :
    temp ← GPR[rt]
I + 1 :
    FCR[fs] ← temp
    COC[1] ← $FCR[fs]_{23}$
```

Exceptions:

```
Coprocessor Unusable exception
Floating-Point exception
```

FPU Exceptions:

```
Unimplemented Operation exception
Invalid Operation exception
Division By Zero exception
Inexact exception
Overflow exception
Underflow exception
```

div

Divide

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0
6	5	5	16

MIPS I

Syntax:

```
div  rs,rt
```

Description:

The contents of register rs are divided by the contents of register rt. The 32-bit quotient is stored in special register LO, and the 32-bit remainder is stored in special register HI. The operands are treated as signed integers.

Operation:

```
q ← GPR[rs] / GPR[rt]
LO ← q
r ← GPR[rs] % GPR[rt]
HI ← r
```

Exceptions:

None

divu

Divide Unsigned

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	rs					rt					0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1
6						5					5					16															

MIPS I

Syntax:

```
divu  rs,rt
```

Description:

The contents of register rs are divided by the contents of register rt.

The 32-bit quotient is stored in special register LO, and the 32-bit remainder is stored in special register HI. The operands are treated as unsigned integers.

Operation:

```
q ← (0 || GPR[rs]) / (0 || GPR[rt])
LO ← q
r ← (0 || GPR[rs]) % (0 || GPR[rt])
HI ← r
```

Exceptions:

None

eret

Exception Return

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
6						26																									

MIPS III

Syntax:

```
eret
```

Description:

Return from interrupt, exception, or error. The LLbit is cleared to 0.

There is no delay slot of a branch/jump instruction, and the instruction immediately following the eret instruction is discarded.

Restrictions:

Do not place this instruction in the delay slot of a branch/jump instruction.

Two nops are required when an eret instruction follows an mtc0 instruction.

Operation:

```
if (StatusERL = 1) then
    PC ← ErrorEPC
    $\text{Status}_{\text{ERL}}$ ← 0
else
    PC ← EPC
    $\text{Status}_{\text{EXL}}$ ← 0
endif
LLbit ← 0
```

Exceptions:

```
Coprocessor Unusable exception
```

j

Jump

31 30 29 28 27 26	25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0	target
6	26

MIPS I

Syntax:

```
j  target
```

Description:

The program jumps with a one instruction delay to the destination address. The destination address is formed by shifting the 26-bit target left two bits and concatenating it with the high-order 4 bits of the PC.

Operation:

```
I :
    temp ← target
I + 1:
    PC ← $PC_{31..28}$ || temp || $0^2$
```

Exceptions:

None

jal

Jump And Link

31 30 29 28 27 26	25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 1	target
6	26

MIPS I

Syntax:

```
jal  target
```

Description:

The program jumps with a one instruction delay to the destination address. The destination address is formed by shifting the 26-bit target left two bits and concatenating it with the high-order 4 bits of the PC. The address of the instruction following the delay slot is stored in r31 (the link register).

Restrictions:

The instruction for changing the register r31 (link register) cannot be placed in the delay slot of the jal instruction.

Operation:

```
I :
    $GPR[31] \leftarrow PC + 8$
    $temp \leftarrow target$
I + 1 :
    $PC \leftarrow PC_{31..28} \,||\, temp \,||\, 0^2$
```

Exceptions:

None

jalr

Jump And Link Register

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	0 0 0 0 0	rd	0 0 0 0 0 0 0 1 0 0 1
6	5	5	5	11

MIPS I

Syntax:

```
jalr  rd,rs
```

Description:

The program jumps with a one instruction delay to the address in the rs register. The address of the instruction following the delay slot is stored in register rd. If rd is not specified, it is assumed to be r31 by default.

Restrictions:

The same register cannot be specified for rd and rs.
The instruction for changing the register rd or the register rs cannot be placed in the delay slot of the jalr instruction.

Operation:

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

Exceptions:

None

jr

Jump Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	rs					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
6						5					21																				

MIPS I

Syntax:

```
jr  rs
```

Description:

The program jumps with a one instruction delay to the address in the rs register.

Operation:

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

Exceptions:

None

lb

Load Byte

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0	base	rt	offset
6	5	5	16

MIPS I

Syntax:

```
lb  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The byte in memory at that address is sign-extended and loaded into register rt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
pAddr ← $pAddr_{PSIZE-1..2}$ || ($pAddr_{1..0}$ xor $ReverseEndian^2$)
memword ← LoadMemory(CCA, BYTE, pAddr, vAddr, DATA)
byte ← $vAddr_{1..0}$ xor $ReverseEndian^2$
GPR[rt] ← sign_extend($memword_{7+8*byte..8*byte}$)
```

Exceptions:

Address Error exception

Bus Error exception

lbu

Load Byte Unsigned

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 1 0 0	base	rt	offset
6	5	5	16

MIPS I

Syntax:

```
lbu  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The byte in memory at that address is zero-extended and loaded into register rt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
pAddr ← $pAddr_{PSIZE-1..2}$ || ($pAddr_{1..0}$ xor ReverseEndian$^2$)
memword ← LoadMemory(CCA, BYTE, pAddr, vAddr, DATA)
byte ← $vAddr_{1..0}$ xor ReverseEndian$^2$
GPR[rt] ← zero_extend($memword_{7+8*byte..8*byte}$)
```

Exceptions:

```
Address Error exception
Bus Error exception
```

lh

Load Halfword

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 1	base	rt	offset
6	5	5	16

MIPS I

Syntax:

```
lh  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The halfword in memory at that address is sign-extended and loaded into register rt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if ($vAddr_0 \neq 0$) then
    SignalException(AddressError)
endif
(pAddr, CCA) . ← AddressTranslation(vAddr, DATA, LOAD)
pAddr ← $pAddr_{PSIZE-1..2}$ || ($pAddr_{1..0}$ xor (ReverseEndian || 0))
memword ← LoadMemory(CCA, HALFWORD, pAddr, vAddr, DATA)
byte ← $vAddr_{1..0}$ xor (ReverseEndian || 0)
GPR[rt] ← sign_extend($memword_{15+8*byte..8*byte}$)
```

Exceptions:

```
Address Error exception
Bus Error exception
```

lhu

Load Halfword Unsigned

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td colspan="5">base</td><td colspan="5">rt</td><td colspan="16">offset</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="16">16</td></tr>
</table>

MIPS I

Syntax:

```
lhu  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The halfword in memory at that address is zero-extended and loaded into register rt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if ($vAddr_0 \neq 0$) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
pAddr ← $pAddr_{PSIZE-1..2}$ || ($pAddr_{1..0}$ xor (ReverseEndian || 0))
memword ← LoadMemory(CCA, HALFWORD, pAddr, vAddr, DATA)
byte ← $vAddr_{1..0}$ xor (ReverseEndian || 0)
GPR[rt] ← zero_extend($memword_{15+8*byte..8*byte}$)
```

Exceptions:

```
Address Error exception
Bus Error exception
```

lui

Load Upper Immediate

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 1 1	0 0 0 0 0	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
lui rt,immediate
```

Description:

The 16-bit immediate value is shifted left by 16 bits to produce a 32-bit word. The low-order 16 bits are set to 0 and the result is stored in register rt.

Operation:

GPR[rt] ← immediate || 0^{16}

Exceptions:

None

lw

Load Word

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 1 1	base	rt	offset
6	5	5	16

MIPS I

Syntax:

```
lw  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The word in memory at that address is loaded into register rt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if ($vAddr_{1..0} \neq 0^2$) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
GPR[rt] ← memword
```

Exceptions:

```
Address Error exception

Bus Error exception
```

lwc1

Load Word to FPU

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 0 0 0 1	base	ft	offset
6	5	5	16

MIPS I

FPU

Syntax:

```
lwc1  ft,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The word in memory at that address is loaded into FPU register ft.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base];
if ($vAddr_{1..0} \neq 0^2$) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
StoreFPR(ft, UNINTERPRETED_WORD, memword)
```

Exceptions:

```
Coprocessor Unusable exception
Address Error exception
Bus Error exception
```

lwl

Load Word Left

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 1 0	base	rt	offset
6	5	5	16

MIPS I

Syntax:

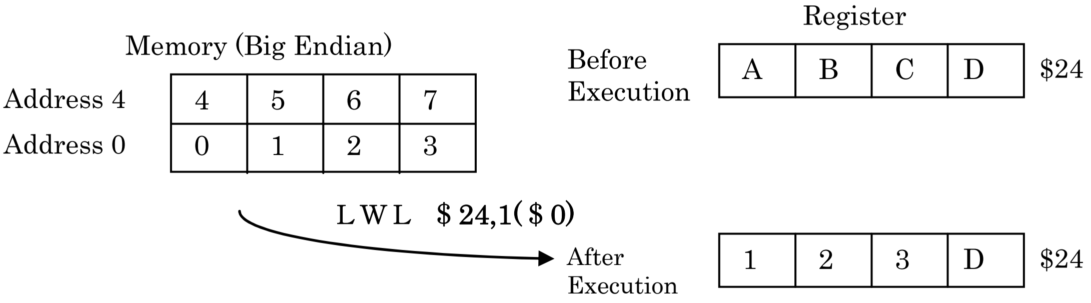
```
lwl  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The word in memory at that address is shifted left so that the byte at that address is at the leftmost end of the word.

The result of the shift operation is merged into register rt.



Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
if (BigEndianMem=0) then
    pAddr ← pAddr$_{PSIZE-1..2}$ || $0^2$
endif
byte ← vAddr$_{1..0}$ xor ReverseEndian$^2$
memword ← LoadMemory(CCA, byte, pAddr, vAddr, DATA)
temp ← memword$_{7+8*byte..0}$ || GPR[rt]$_{23-8*byte..0}$
GPR[rt] ← temp
```

Exceptions:

```
Address Error exception
Bus Error exception
```

lwr

Load Word Right

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 1 1 0	base	rt	offset
6	5	5	16

MIPS I

Syntax:

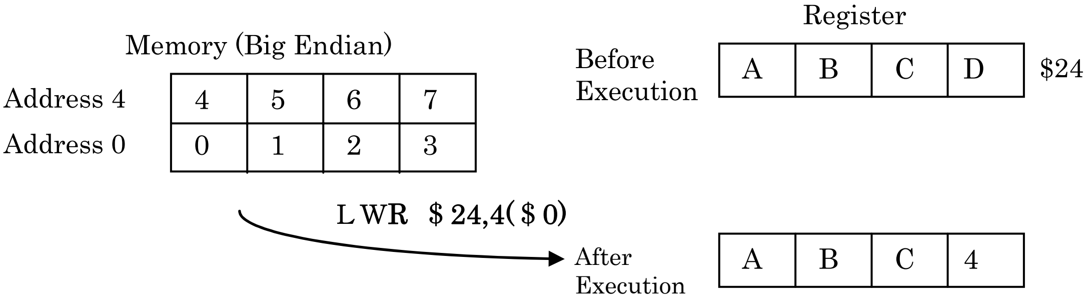
```
lwr  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The word in memory at that address is shifted right so that the byte at that address is at the rightmost end of the word.

The result of the shift operation is merged into register rt.



Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
if (BigEndianMem=0) then
    pAddr ← $pAddr_{PSIZE-1..2}$ || $0^2$
endif
byte ← $vAddr_{1..0}$ xor $ReverseEndian^2$
memword ← LoadMemory(CCA, byte, pAddr, vAddr, DATA)
temp ← $memword_{31..32-8*byte..0}$ || $GPR[rt]_{31-8*byte..0}$
GPR[rt] ← temp
```

Exceptions:

```
Address Error exception
```

```
Bus Error exception
```

mfc0

Move From Coprocessor0

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="5">rt</td><td colspan="5">rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="11">11</td></tr>
</table>

MIPS I

Syntax:

```
mfc0  rt,rd
```

Description:

The contents of CP0 register rd are transferred to CPU register rt.

Operation:

```
data ← CP0R[rd]
GPR[rt] ← data
```

Exceptions:

```
Coprocessor Unusable exception
```

mfc1

Move From FPU

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 0 0 0 0	rt	fs	0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

MIPS I

FPU

Syntax:

```
mfc1  rt,fs
```

Description:

The contents of FPU general purpose register fs are transferred to CPU register rt.

Operation:

```
data ← ValueFPR(fs,UNINTERPRETED_WORD)
GPR[rt] ← data
```

Exceptions:

Coprocessor Unusable exception

mfhi

Move From HI Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	rd					0	0	0	0	0	0	1	0	0	0	0
6						10										5					11										

MIPS I

Syntax:

```
mfhi  rd
```

Description:

The contents of special register HI are transferred to register rd.

Operation:

GPR[rd] ← HI

Exceptions:

None

mflo

Move From LO Register

31 30 29 28 27 26	25 24 23 22 21 20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	rd	0 0 0 0 0 0 1 0 0 1 0
6	10	5	11

MIPS I

Syntax:

```
mflo  rd
```

Description:

The contents of special register LO are transferred to register rd.

Operation:

GPR[rd] ← LO

Exceptions:

```
None
```

mtc0

Move To Coprocessor0

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td colspan="5">rt</td><td colspan="5">rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="11">11</td></tr>
</table>

MIPS I

Syntax:

```
mtc0  rt,rd
```

Description:

The contents of CPU register rt are transferred to CP0 register rd.

Operation:

DATA ← GPR[rt]
CP0R[rd] ← DATA

Exceptions:

Coprocessor Unusable exception

mtc1

Move To FPU

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1	0 0 1 0 0	rt	fs	0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

MIPS I

FPU

Syntax:

```
mtc1  rt,fs
```

Description:

The contents of CPU register rt are transferred to FPU general-purpose register fs.

Operation:

```
data ← $GPR[rt]_{31..0}$
StoreFPR(fs,UNINTERPRETED_WORD,data)
```

Exceptions:

```
Coprocessor Unusable exception
```

mthi

Move To HI Register

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
6	5	21

MIPS I

Syntax:

```
mthi  rs
```

Description:

The contents of register rs are transferred to special register HI.

Operation:

HI ← GPR[rs]

Exceptions:

```
None
```

mtlo

Move To LO Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	rs					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
6						5					21																				

MIPS I

Syntax:

```
mtlo  rs
```

Description:

The contents of register rs are transferred to special register LO.

Operation:

LO ← GPR[rs]

Exceptions:

None

mult

Multiply

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
6	5	5	16

MIPS I

Syntax:

```
mult  rs,rt
```

Description:

The contents of register rs are multiplied by the contents of register rt and the 64-bit result is stored in special registers HI and LO. The operands are treated as signed integers.

Operation:

```
$temp \leftarrow (GPR[rs] \times GPR[rt])$
$HI \leftarrow temp_{63..32}$
$LO \leftarrow temp_{31..0}$
```

Exceptions:

```
None
```

multu

Multiply Unsigned

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
6	5	5	16

MIPS I

Syntax:

```
multu  rs,rt
```

Description:

The contents of register rs are multiplied by the contents of register rt and the 64-bit result is stored in special registers HI and LO. The operands are treated as unsigned integers.

Operation:

```
temp ← (GPR[rs] × GPR[rt])
HI ← $temp_{63..32}$
LO ← $temp_{31..0}$
```

Exceptions:

None

nop

No Operation

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6						26																									

MIPS I

Syntax:

```
nop
```

Description:

This instruction does nothing. It is interpreted by the actual hardware as "SLL r0, r0, 0".

Operation:

None

Exceptions:

None

nor

NOR

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 1 0 0 1 1 1
6	5	5	5	11

MIPS I

Syntax:

```
nor  rd,rs,rt
```

Description:

A bitwise NOR (negative OR) is performed for the contents of registers rs and rt and the result is stored in register rd.

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

Operation:

GPR[rd] ← GPR[rs] nor GPR[rt]

Exceptions:

None

or

OR

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 1 0 0 1 0 1
6	5	5	5	11

MIPS I

Syntax:

```
or  rd,rs,rt
```

Description:

A bitwise OR (logical sum) is performed for the contents of registers rs and rt and the result is stored in register rd.

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Operation:

GPR[rd] ← GPR[rs] or GPR[rt]

Exceptions:

None

ori

OR Immediate

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 0 1	rs	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
ori  rt,rs,immediate
```

Description:

The 16-bit immediate field is zero-extended and a bitwise OR (logical sum) is performed with the contents of register rs. The result is stored in register rt.

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Operation:

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

Exceptions:

None

sb

Store Byte

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 0 0 0	base	rt	offset
6	5	5	16

MIPS I

Syntax:

```
sb  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The low-order byte of register rt is stored in memory at that address.

Bus error exceptions will not be reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
pAddr ← $pAddr_{PSIZE-1..2}$ || ($pAddr_{1..0}$ xor $ReverseEndian^2$ )
bytesel ← $vAddr_{1..0}$ xor $ReverseEndian^2$
dataword ← $GPR[rt]_{31-8*bytesel..0}$ || $0^{8*bytesel}$
StoreMemory(CCA, BYTE, dataword, pAddr, vAddr, DATA)
```

Exceptions:

```
Address Error exception
```

sh

Store HalfWord

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="6">1 0 1 0 0 1</td><td colspan="5">base</td><td colspan="5">rt</td><td colspan="16">offset</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="16">16</td></tr>
</table>

MIPS I

Syntax:

```
sh  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The low-order halfword of register rt is stored in memory at that address.

Bus error exceptions will not be reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if ($vAddr_0 \neq 0$) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
pAddr ← $pAddr_{PSIZE-1..2}$ || ($pAddr_{1..0}$ xor (ReverseEndian || 0))
bytesel ← $vAddr1_{1..0}$ xor (ReverseEndian || 0)
dataword ← $GPR[rt]_{31-8*bytesel..0}$ || $0^{8*bytesel}$
StoreMemory(CCA, HALFWORD, dataword, pAddr, vAddr, DATA)
```

Exceptions:

```
Address Error exception
```

sll

Shift Left Logical

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
0 0 0 0 0 0	0 0 0 0 0	rt	rd	sa	0 0 0 0 0 0
6	5	5	5	5	6

MIPS I

Syntax:

```
sll  rd,rt,sa
```

Description:

The contents of register rt are shifted left sa bits. Zeroes are inserted into the low-order bit positions from the right.

The 32-bit result is stored in register rd.

Operation:

```
$s \leftarrow sa$
$temp \leftarrow GPR[rt]_{(31-s)..0} \,||\, 0^{s}$
$GPR[rd] \leftarrow temp$
```

Exceptions:

None

sllv

Shift Left Logical Variable

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0 0 0 1 0 0
6	5	5	5	11

MIPS I

Syntax:

```
sllv  rd,rt,rs
```

Description:

The contents of register rt are shifted left. Zeroes are inserted into the low-order bit positions from the right. The shift amount is specified by the low-order 5 bits of register rs.

The 32-bit result is stored in register rd.

Operation:

```
$s \leftarrow GPR[rs]_{4..0}$
$temp \leftarrow GPR[rt]_{(31-s)..0} \parallel 0^s$
$GPR[rd] \leftarrow temp$
```

Exceptions:

None

slt

Set on Less Than

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	1	0	1	0
6						5					5					5					11										

MIPS I

Syntax:

```
slt  rd,rs,rt
```

Description:

The contents of registers rs and rt are compared as 32-bit signed integers. If the comparison result is rs < rt, 1 is stored in register rd, otherwise 0 is stored.

Operation:

```
if (GPR[rs] < GPR[rt]) then
    GPR[rd] ← $0^{31}$ || 1
else
    GPR[rd] ← $0^{32}$
endif
```

Exceptions:

None

slti

Set on Less Than Immediate

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 1 0	rs	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
slti  rt,rs,immediate
```

Description:

The contents of register rs and the 16-bit immediate value sign-extended to 32 bits are compared as 32-bit signed integers. If the comparison result is rs < immediate, 1 is stored in register rt, otherwise 0 is stored.

Operation:

```
if (GPR[rs] < sign_extend(immediate)) then
    GPR[rt] ← $0^{31}$ || 1
else
    GPR[rt] ← $0^{32}$
endif
```

Exceptions:

None

sltiu

Set on Less Than Immediate Unsigned

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 1 1	rs	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
sltiu  rt,rs,immediate
```

Description:

The contents of register rs and the sign-extended 16-bit immediate value are compared as 32-bit unsigned integers. If the comparison result is rs < immediate, 1 is stored in register rt, otherwise 0 is stored.

Because the 16-bit immediate value is sign-extended before comparison, the range of numeric values that the immediate represents is not sequential, but split into two areas; around the smallest and largest 32-bit unsigned integers. That is [0,32767] and [max_unsigned-32767, max_unsigned], respectively.

Operation:

```
if ((0 || GPR[rs]) < (0 || sign_extend(immediate))).then
    GPR[rt] ← $0^{31}$ || 1
else
    GPR[rt] ← $0^{32}$
endif
```

Exceptions:

None

sltu

Set on Less Than Unsigned

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="5">rs</td><td colspan="5">rt</td><td colspan="5">rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="11">11</td></tr>
</table>

MIPS I

Syntax:

```
sltu  rd,rs,rt
```

Description:

The contents of registers rs and rt are compared as 32-bit unsigned integers. If the comparison result is rs < rt, 1 is stored in register rd, otherwise 0 is stored.

Operation:

```
if ((0 || GPR[rs]) < (0 || GPR[rt])) then
    GPR[rd] ← $0^{31}$ || 1
else
    GPR[rd] ← $0^{32}$
endif
```

Exceptions:

None

sra

Shift Right Arithmetic

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
0 0 0 0 0 0	0 0 0 0 0	rt	rd	sa	0 0 0 0 1 1
6	5	5	5	5	6

MIPS I

Syntax:

```
sra  rd,rt,sa
```

Description:

The contents of register rt are shifted right sa bits. The sign is extended into the high-order bit positions.

The 32-bit result is stored in register rd.

Operation:

```
$s \leftarrow sa$
$temp \leftarrow ((GPR[rt]_{31})^{s} \,||\, GPR[rt]_{31..s})$
$GPR[rd] \leftarrow temp$
```

Exceptions:

None

srav

Shift Right Arithmetic Variable

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0 0 0 1 1 1
6	5	5	5	11

MIPS I

Syntax:

```
srav  rd,rt,rs
```

Description:

The contents of register rt are shifted right. The sign is extended into the high-order bit positions. The shift amount is specified by the low-order 5 bits of register rs.

The 32-bit result is stored in register rd.

Operation:

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

Exceptions:

None

srl

Shift Right Logical

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
0 0 0 0 0 0	0 0 0 0 0	rt	rd	sa	0 0 0 0 1 0
6	5	5	5	5	6

MIPS I

Syntax:

```
srl  rd,rt,sa
```

Description:

The contents of register rt are shifted right sa bits. Zeroes are inserted into the high-order bit positions from the left.

The 32-bit result is stored in register rd.

Operation:

```
$s \leftarrow sa$
$temp \leftarrow (0^{s} \| GPR[rt]_{31..s})$
$GPR[rd] \leftarrow temp$
```

Exceptions:

None

srlv

Shift Right Logical Variable

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="5">rs</td><td colspan="5">rt</td><td colspan="5">rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="11">11</td></tr>
</table>

MIPS I

Syntax:

```
srlv  rd,rt,rs
```

Description:

The contents of register rt are shifted right. Zeroes are inserted into the high-order bit positions from the left. The shift amount is specified by the low-order 5 bits of register rs.

The 32-bit result is stored in register rd.

Operation:

```
s ← $\mathrm{GPR}[rs]_{4..0}$
temp ← $(0^{s} \,||\, \mathrm{GPR}[rt]_{31..s})$
GPR[rd] ← temp
```

Exceptions:

None

sub

Subtract

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="5">rs</td><td colspan="5">rt</td><td colspan="5">rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="11">11</td></tr>
</table>

MIPS I

Syntax:

```
sub rd,rs,rt
```

Description:

The contents of register rt are subtracted from the contents of register rs and the result is stored in register rd.

If a two’s complement overflow occurs, an exception is generated.

Operation:

```
temp ← $(\mathrm{GPR[rs]}_{31} \,\|\, \mathrm{GPR[rs]}_{31..0}) - (\mathrm{GPR[rt]}_{31} \,\|\, \mathrm{GPR[rt]}_{31..0})$
if ($\mathrm{temp}_{32} \neq \mathrm{temp}_{31}$) then
    SignalException(IntegerOverflow)
else
    GPR[rd] ← temp
endif
```

Exceptions:

```
Integer Overflow exception
```

subu

Subtract Unsigned

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 1 0 0 0 1 1
6	5	5	5	11

MIPS I

Syntax:

```
subu  rd,rs,rt
```

Description:

The contents of register rt are subtracted from the contents of register rs and the result is stored in register rd.

No exception is generated even if a two’s complement overflow occurs.

Operation:

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

Exceptions:

None

sw

Store Word

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 0 1 1	base	rt	offset
6	5	5	16

MIPS I

Syntax:

```
sw  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The contents of register rt are stored in memory at that address. Bus error exceptions will not be reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if ($vAddr_{1..0} \neq 0^2$) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
dataword ← GPR[rt]
StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)
```

Exceptions:

```
Address Error exception
```

swc1

Store Word from FPU

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 0 0 1	base	ft	offset
6	5	5	16

MIPS I

FPU

Syntax:

```
swc1  ft,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The contents of FPU register ft are stored in memory at that address.

Bus error exceptions will not be reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr$_{1..0}$ ≠ 0$^2$) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
dataword ← ValueFPR(ft, UNINTERPRETED_WORD)
StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)
```

Exceptions:

```
Coprocessor Unusable exception

Address Error exception
```

swl

Store Word Left

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 0 1 0	base	rt	offset
6	5	5	16

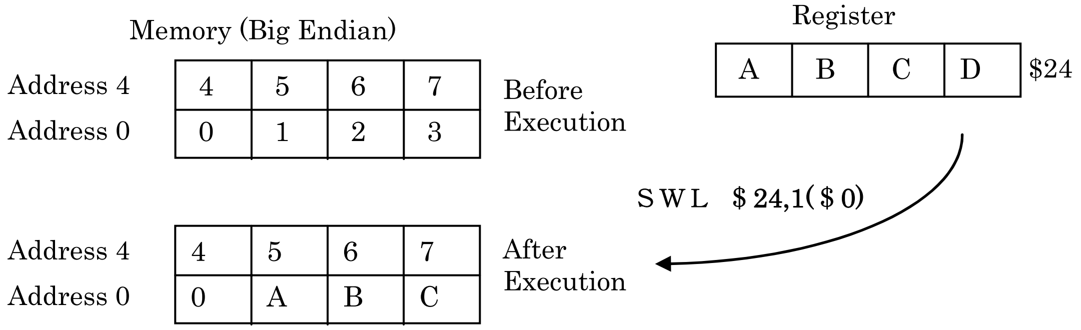
MIPS I

Syntax:

```
swl  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The contents of register rt are shifted right so that the leftmost byte is in the same position in the word as the byte at the generated address. The byte (bytes) that contains the original data of register rt is stored in the byte (bytes) from the byte position of the specified address to the word boundary on the right side. Bus error exceptions are not reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.



Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
if (BigEndianMem=0) then
    pAddr ← $\text{pAddr}_{\text{PSIZE}-1..2}$ || $0^2$
endif
byte ← $\text{vAddr}_{1..0}$ xor $\text{ReverseEndian}^2$
dataword ← $0^{24-8*\text{byte}}$ || $\text{GPR[rt]}_{31..24-8*\text{byte}}$
StoreMemory(CCA, byte, dataword, pAddr, vAddr, DATA)
```

Exceptions:

```
Address Error exception
```

swr

Store Word Right

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 0	base	rt	offset
6	5	5	16

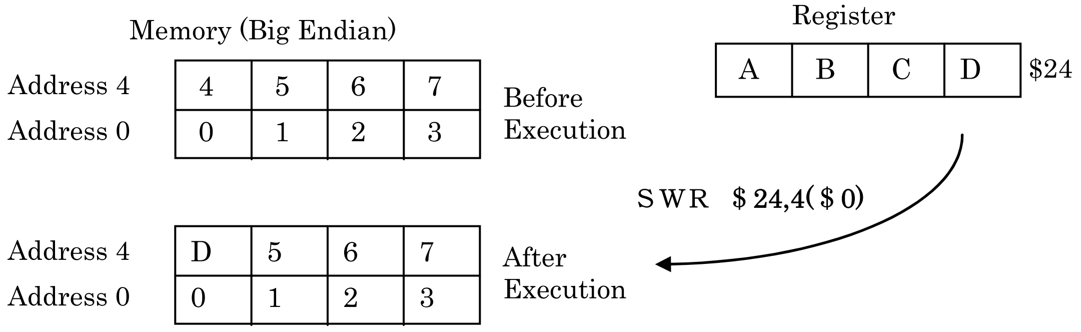
MIPS I

Syntax:

```
swr  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address. The contents of register rt are shifted left so that the rightmost byte is in the same position in the word as the byte at the generated address. The byte (bytes) that contains the original data of register rt is stored in the byte (bytes) from the byte position of the specified address to the word boundary on the left side. Bus error exceptions are not reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.



Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr,CCA) ← AddressTranslation(vAddr, DATA, STORE)
if (BigEndianMem=0) then
    pAddr ← $pAddr_{PSIZE-1..2}$ || $0^2$
endif
byte ← $vAddr_{1..0}$ xor $ReverseEndian^2$
dataword ← $GPR[rt]_{31-8*byte}$ || $0^{8*byte}$
StoreMemory(CCA,WORD − byte, dataword, pAddr, vAddr, DATA)
```

Exceptions:

```
Address Error exception
```

syscall

System Call

31 30 29 28 27 26	25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6	5 4 3 2 1 0
0 0 0 0 0 0	code	0 0 1 1 0 0
6	20	6

MIPS I

Syntax:

```
syscall  code
```

Description:

A system call exception occurs, immediately transferring control to the exception handling program. The code field is arbitrary.

Operation:

SignalException(SystemCall)

Exceptions:

```
System Call exception
```

xor

XOR

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 1 0 0 1 1 0
6	5	5	5	11

MIPS I

Syntax:

```
xor  rd,rs,rt
```

Description:

A bitwise Exclusive OR (exclusive logical sum) is performed for the contents of registers rs and rt and the result is stored in register rd.

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Operation:

GPR[rd] ← GPR[rs] xor GPR[rt]

Exceptions:

None

xori

Exclusive OR Immediate

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 1 0	rs	rt	immediate
6	5	5	16

MIPS I

Syntax:

```
xori  rt,rs,immediate
```

Description:

The 16-bit immediate field is zero-extended and a bitwise Exclusive OR (exclusive logical sum) is performed with the contents of register rs. The result is stored in register rt.

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Operation:

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

Exceptions:

None

ALLEGREX™ Instructions

clo

Count Leading One

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	0 0 0 0 0	rd	0 0 0 0 0 0 1 0 1 1 1
6	5	5	5	11

ALLEGREX™

Syntax:

```
clo  rd,rs
```

Description:

The number of consecutive ones in register rs are counted starting from the leftmost bit (MSB). The result (0-32) is stored in register rd.

Operation:

```
temp ← 32
for (i) in(31..0)
     if ($GPR[rs]_i = 0$) then
            temp ← 31 − i
            break
     endif
endfor
GPR[rd] ← temp
```

Exceptions:

None

clz

Count Leading Zero

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	0 0 0 0 0	rd	0 0 0 0 0 0 1 0 1 1 0
6	5	5	5	11

ALLEGREX™

Syntax:

```
clz  rd,rs
```

Description:

The number of consecutive zeroes in register rs are counted starting from the leftmost bit (MSB). The result (0-32) is stored in register rd.

Operation:

```
temp ← 32
for (i) in(31..0)
    if (GPR[rs]$_i$ = 1) then
        temp ← 31 − i
        break
    endif
endfor
GPR[rd] ← temp
```

Exceptions:

None

madd

Multiply Add

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
6	5	5	16

ALLEGREX™

Syntax:

```
madd  rs,rt
```

Description:

This is a product-sum operation for multiplying the contents of registers rs and rt as signed integers, adding the 64-bit product to the 64-bit value obtained by concatenating special registers HI and LO, and updating the special registers HI and LO.

Operation:

```
temp ← (HI || LO) + (GPR[rs] × GPR[rt])
HI ← $temp_{63..32}$
LO ← $temp_{31..0}$
```

Exceptions:

None

maddu

Multiply Add Unsigned

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1
6	5	5	16

ALLEGREX™

Syntax:

```
maddu  rs,rt
```

Description:

This is a product-sum operation for multiplying the contents of registers rs and rt as unsigned integers, adding the 64-bit product to the 64-bit value obtained by concatenating special registers HI and LO, and updating the special registers HI and LO.

Operation:

```
temp ← (HI || LO) + (GPR[rs] × GPR[rt])
HI ← $temp_{63..32}$
LO ← $temp_{31..0}$
```

Exceptions:

None

msub

Multiply Subtract

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0
6	5	5	16

ALLEGREX™

Syntax:

```
msub  rs,rt
```

Description:

This is a product-difference operation for multiplying the contents of registers rs and rt as signed integers, subtracting the 64-bit product from the 64-bit value obtained by concatenating special registers HI and LO, and updating the special registers HI and LO.

Operation:

```
temp ← (HI || LO) − (GPR[rs] × GPR[rt])
HI ← $\text{temp}_{63..32}$
LO ← $\text{temp}_{31..0}$
```

Exceptions:

None

msubu

Multiply Subtract Unsigned

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	rs					rt					0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1
6						5					5					16															

ALLEGREX™

Syntax:

```
msubu  rs,rt
```

Description:

This is a product-difference operation for multiplying the contents of registers rs and rt as unsigned integers, subtracting the 64-bit product from the 64-bit value obtained by concatenating special registers HI and LO, and updating the special registers HI and LO.

Operation:

```
temp ← (HI || LO) − (GPR[rs] × GPR[rt])
HI ← $temp_{63..32}$
LO ← $temp_{31..0}$
```

Exceptions:

None

max

Select Max

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="5">rs</td><td colspan="5">rt</td><td colspan="5">rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="11">11</td></tr>
</table>

ALLEGREX™

Syntax:

```
max  rd,rs,rt
```

Description:

The contents of registers rs and rt are compared as 32-bit signed integers. The contents of the register containing the larger value are stored in register rd.

Operation:

```
if (GPR[rs] < GPR[rt]) then
    GPR[rd] ← GPR[rt]
else
    GPR[rd] ← GPR[rs]
endif
```

Exceptions:

None

min

Select Min

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="6">0 0 0 0 0 0</td><td colspan="5">rs</td><td colspan="5">rt</td><td colspan="5">rd</td><td colspan="11">0 0 0 0 0 1 0 1 1 0 1</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="11">11</td></tr>
</table>

ALLEGREX™

Syntax:

```
min  rd,rs,rt
```

Description:

The contents of registers rs and rt are compared as 32-bit signed integers. The contents of the register containing the smaller value are stored in register rd.

Operation:

```
if (GPR[rs] < GP R[rt]) then
    GPR[rd] ← GPR[rs]
else
    GPR[rd] ← GPR[rt]
endif
```

Exceptions:

None

movn

Move Conditional on Not Zero

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0 0 1 0 1 1
6	5	5	5	11

ALLEGREX™

Syntax:

```
movn  rd,rs,rt
```

Description:

If the contents of register rt are not equal to zero, the contents of register rs are copied to register rd.

Operation:

```
if (GPR[rt] ≠ 0) then
    GPR[rd] ← GPR[rs]
endif
```

Exceptions:

None

movz

Move Conditional on Zero

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0 0 1 0 1 0
6	5	5	5	11

ALLEGREX™

Syntax:

```
movz  rd,rs,rt
```

Description:

If the contents of register rt are equal to zero, the contents of register rs are copied to register rd.

Operation:

```
if (GPR[rt] = 0) then
    GPR[rd] ← GPR[rs]
endif
```

Exceptions:

None

ext

Extract Bit Field

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
0 1 1 1 1 1	rs	rt	size−1	pos	0 0 0 0 0 0
6	5	5	5	5	6

ALLEGREX™

Syntax:

```
ext  rt,rs,pos,size
```

Description:

size bits are extracted from register rs starting at offset bit position pos within the word. The result is stored right-justified in register rt. The high-order bits of register rt are filled with zeros.

Restrictions:

The values of pos and size must be specified so that the sum of pos and size is equal to or less than 32.

Operation:

```
if (pos + size > 32) then
    UNPREDICTABLE
endif
temp ← $0^{32 - \mathrm{size}} \,||\, \mathrm{GPR[rs]}_{(\mathrm{size} + \mathrm{pos}-1)..\mathrm{pos}}$
GPR[rt] ← temp
```

Exceptions:

None

ins

Insert Bit Field

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="5">rs</td><td colspan="5">Rt</td><td colspan="5">pos+size−1</td><td colspan="5">pos</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td colspan="6">6</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="5">5</td><td colspan="6">6</td></tr>
</table>

ALLEGREX™

Syntax:

```
ins  rt,rs,pos,size
```

Description:

size bits are extracted from register rs starting from the low-order bit position and inserted into register rt at offset bit position pos within the word.

Restrictions:

The value of size must be specified to be equal to or more than 1.

Operation:

```
if (size < 1) then
    UNPREDICTABLE
endif
GPR[rt] ← $\mathrm{GPR[rt]}_{31..(\mathrm{size}+\mathrm{pos})}$ || $\mathrm{GPR[rs]}_{(\mathrm{size}-1)..0}$ || $\mathrm{GPR[rt]}_{(\mathrm{pos}-1)..0}$
```

Exceptions:

None

seb

Sign-Extend Byte

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 1	0 0 0 0 0	rt	rd	1 0 0 0 0 1 0 0 0 0 1
6	5	5	5	11

ALLEGREX™

Syntax:

```
seb  rd,rt
```

Description:

The lowest byte of register rt is sign-extended to 32 bits. The result is stored in register rd.

Operation:

GPR[rd] ← sign_extend(GPR[rt]$_{7..0}$)

Exceptions:

None

seh

Sign-Extend Halfword

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	0	0	0	0	rt					rd					1	1	0	0	0	1	0	0	0	0	0
6						5					5					5					11										

ALLEGREX™

Syntax:

```
seh  rd,rt
```

Description:

The low-order halfword of register rt is sign-extended to 32 bits. The result is stored in register rd.

Operation:

GPR[rd] ← sign_extend(GPR[rt]$_{15..0}$)

Exceptions:

None

bitrev

Bit Reverse

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 1	0 0 0 0 0	rt	rd	1 0 1 0 0 1 0 0 0 0 0
6	5	5	5	11

ALLEGREX™

Syntax:

```
bitrev  rd,rt
```

Description:

The contents of register rt are swapped bit-for-bit within the word. The 32-bit result is stored in register rd.

Operation:

$\mathrm{GPR[rd]} \leftarrow (\mathrm{GPR[rt]}_0 \mathbin{||} \mathrm{GPR[rt]}_1 \mathbin{||} \ldots \mathbin{||} \mathrm{GPR[rt]}_{31})$

Exceptions:

None

rotr

Rotate Word Right

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
0 0 0 0 0 0	0 0 0 0 1	rt	rd	sa	0 0 0 0 1 0
6	5	5	5	5	6

ALLEGREX™

Syntax:

```
rotr  rd,rt,sa
```

Description:

The contents of register rt are rotated right by right-shifting them sa bits. The 32-bit result is stored in register rd.

Operation:

```
s ← sa
temp ← (GPR[rt]$_{s-1..0}$ || GPR[rt]$_{31..s}$)
GPR[rd] ← temp
```

Exceptions:

None

rotrv

Rotate Word Right Variable

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0	rs	rt	rd	0 0 0 0 1 0 0 0 1 1 0
6	5	5	5	11

ALLEGREX™

Syntax:

```
rotrv  rd,rt,rs
```

Description:

The contents of register rt are rotated right by performing a right shift. The shift amount is specified by the low-order 5 bits of register rs.

The 32-bit result is stored in register rd.

Operation:

```
$s \leftarrow \mathrm{GPR}[rs]_{4..0}$
$temp \leftarrow (\mathrm{GPR}[rt]_{s-1..0} \,||\, \mathrm{GPR}[rt]_{31..s})$
$\mathrm{GPR}[rd] \leftarrow temp$
```

Exceptions:

None

wsbh

Word Swap Bytes within Halfword

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 1	0 0 0 0 0	rt	rd	0 0 0 1 0 1 0 0 0 0 0
6	5	5	5	11

ALLEGREX™

Syntax:

```
wsbh  rd,rt
```

Description:

The contents of register rt are swapped byte-for-byte within each halfword of the register. The 32-bit result is stored in register rd.

Operation:

GPR[rd] ← (GP R[rt]$_{23..16}$ || GPR[rt]$_{31..24}$ || GPR[rt]$_{7..0}$ || GPR[rt]$_{15..8}$)

Exceptions:

None

wsbw

Word Swap Bytes within Word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	0	0	0	0	rt					rd					0	0	0	1	1	1	0	0	0	0	0
6						5					5					5					11										

ALLEGREX™

Syntax:

```
wsbw  rd,rt
```

Description:

The contents of register rt are swapped byte-for-byte within the word. The 32-bit result is stored in register rd.

Operation:

$GPR[rd] \leftarrow (GPR[rt]_{7..0} \,||\, GPR[rt]_{15..8} \,||\, GPR[rt]_{23..16} \,||\, GPR[rt]_{31..24})$

Exceptions:

None

cache : Index Invalidate (I)

Index Invalidate (I)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 0 0 1 0 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x04),offset(base)
```

Description:

This instruction is used for clearing a line in the instruction cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The generated address is used to obtain an index that points to a pair of entries in the cache. The appropriate entry, either WayA or WayB, is selected and invalidated (the Valid bit is cleared to 0) according to the following rules.

1) When both WayA and WayB are valid: Invalidate the oldest entry as determined by the LRU value

2) When either WayA or WayB is valid, but not both: Invalidate the valid entry and clear the Lock bit

3) When both WayA and WayB are invalid: Do nothing

To clear both entries WayA and WayB, issue this instruction twice in a row to the same address (index).

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Index Unlock (I)

Index Unlock (I)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 0 0 1 1 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x06),offset(base)
```

Description:

This instruction is used for canceling the lock on an instruction cache line.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The generated address is used to obtain an index that points to a cache line. The Lock bit for that cache line is cleared (the lock on the cache line is canceled).

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Hit Invalidate (I)

Hit Invalidate (I)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 0 1 0 0 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x08),offset(base)
```

Description:

This instruction is used to invalidate a specific line in the instruction cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The tag for the cache index is obtained from the generated address. If the tag indicates that the addressed line is present in the cache (a cache hit), its entry is invalidated (its Valid bit is cleared). If the Lock bit in the tag is 1 and the LRU bit points to the other entry (in other words, the target entry is locked), the Lock bit is cleared (the lock is canceled).

If the addressed line is not present in the cache, no operation is performed.

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Fill (I)

Fill (I)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
	base	func	offset
1 0 1 1 1 1		0 1 0 1 0	
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x0a),offset(base)
```

Description:

This instruction is used for explicitly filling a specific line in the instruction cache, if the line is not in the cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

If the cache line at the generated address is not in the cache (a cache miss), the cache line is filled. If the line is already in the cache (a cache hit), no operation is performed. If the Lock bit = 0 (not locked), the LRU bit is updated to point to the way that is not being filled.

During the instruction cache fill, the pipeline is interlocked so no other processing can be performed in parallel.

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
Bus Error exception
```

cache : Fill with Lock (I)

Fill with Lock (I)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 0 1 0 1 1	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x0b),offset(base)
```

Description:

This instruction is used for explicitly filling and locking a specific line in the instruction cache, if the line is not in the cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

If the cache line at the generated address is not in the cache (a cache miss), the cache line is filled. If the line is already in the cache (a cache hit), no operation is performed. If the Lock bit = 0 (not locked), the LRU bit is updated to point to the way that is not being filled.

This instruction also sets the Lock bit. As a result, the LRU bit is held, and the target line will be locked and not be subject to replacement.

During the instruction cache fill, the pipeline is interlocked so no other processing can be performed in parallel.

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
Bus Error exception
```

cache : Index Writeback Invalidate (D)

Index Writeback Invalidate (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 0 1 0 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x14),offset(base)
```

Description:

This instruction is used to flush and clear a line in the data cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The generated address is used to obtain an index that points to a pair of entries in the cache. The appropriate entry, either WayA or WayB, is selected according to the rules listed below, and if the Dirty bit in the entry is set, the line is written back to main memory and the entry is invalidated (the Valid bit is cleared to 0). If the Dirty bit is not set, the entry is only invalidated and no writeback is performed.

1) When both WayA and WayB are valid:　Select the oldest as determined by the value of the LRU bit

2) When either WayA or WayB is valid, but not both:　Select the valid entry and clear the Lock bit

3) When both WayA and WayB are invalid:　Do nothing.

To flush and clear both entries WayA and WayB, issue this instruction twice in a row to the same address (index).

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Index Unlock (D)

Index Unlock (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 0 1 1 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x16),offset(base)
```

Description:

This instruction is used to unlock a line in the data cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The generated address is used to obtain an index that points to a cache line. The Lock bit for that cache line is cleared (the lock on the cache line is canceled).

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Create Dirty Exclusive (D)

Create Dirty Exclusive (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 1 0 0 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x18),offset(base)
```

Description:

This instruction is used to create a dirty line in the data cache. It prevents unnecessary cache fills from being performed on lines that are only going to be fully written.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

An entry in the data cache is created for the line at the generated address, in the data cache tag. At the same time, the Dirty bit is set in the tag. This instruction does not perform a data fill.

If all the words in the cache line are written right after this instruction is executed, the cache line can be completed without having to perform a wasteful cache fill. If the Lock bit = 0 (not locked), the LRU bit will be updated to point to the way that was not created.

This instruction can be executed in user mode.

Specifying a non-cache address with the Create Dirty Exclusive (D) instruction will cause indeterminate behavior and is thus prohibited.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Hit Invalidate (D)

Hit Invalidate (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 1 0 0 1	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x19),offset(base)
```

Description:

This instruction is used to invalidate a specific line in the data cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The tag for the cache index is obtained from the generated address. If the tag indicates that the addressed line is present in the cache (a cache hit), its entry is invalidated (its Valid bit is cleared). If the Lock bit in the tag is 1 and the LRU bit points to the other entry (in other words, the target entry is locked), the Lock bit is cleared (the lock is canceled).

If the addressed line is not present in the cache, no operation is performed.

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Hit WriteBack (D)

Hit WriteBack (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 1 0 1 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x1a),offset(base)
```

Description:

This instruction is used to perform an explicit writeback of a dirty line in the data cache.

It is used to explicitly synchronize data in the cache with data in main memory.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The tag for the cache index is obtained from the generated address. If the tag indicates that the addressed line is present in the cache (a cache hit), and if its Dirty bit is set, a writeback is performed on that cache line and the Dirty bit is cleared.

If the addressed line is not present in the cache, no operation is performed.

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Hit WriteBack Invalidate(D)

Hit WriteBack Invalidate(D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 1 0 1 1	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x1b),offset(base)
```

Description:

This instruction is used to perform an explicit writeback of a dirty line in the data cache. It is used to explicitly synchronize data in the cache with data in main memory. After the writeback is performed, the line is invalidated.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

The tag for the cache index is obtained from the generated address. If the tag indicates that the addressed line is present in the cache (a cache hit), and if its Dirty bit is set, a writeback is performed on that cache line and the line is invalidated. If the Dirty bit is not set, the line will only be invalidated. If the Lock bit in the tag is 1 and the LRU bit points to the other entry (in other words, the target entry is locked), the Lock bit is cleared (the lock is canceled).

If the addressed line is not present in the cache, no operation is performed.

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Create Dirty Exclusive with Lock (D)

Create Dirty Exclusive with Lock (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 1 1 0 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x1c),offset(base)
```

Description:

This instruction is used to create a dirty line in the data cache and lock it. It prevents unnecessary cache fills from being performed on lines that are only going to be fully written.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

An entry in the data cache is created for the line at the generated address, in the data cache tag. At the same time, the Dirty bit is set in the tag. This instruction does not perform a data fill.

If all the words in the cache line are written right after this instruction is executed, the cache line can be completed without having to perform a wasteful cache fill. If the Lock bit = 0 (not locked), the LRU bit will be updated to point to the way that was not created. This instruction also sets the Lock bit. As a result, the LRU bit is held, and the target line will be locked and not be subject to replacement.

This instruction can be executed in user mode.

Specifying a non-cache address with the Create Dirty Exclusive with Lock(D) instruction will cause indeterminate behavior and is thus prohibited.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
```

cache : Fill (D)

Fill (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	base	func 1 1 1 1 0	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x1e),offset(base)
```

Description:

This instruction is used for explicitly filling a specific line in the data cache, if the line is not in the cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

If the cache line at the generated address is not in the cache (a cache miss), the cache line is filled. If the line is already in the cache (a cache hit), no operation is performed. If the Lock bit = 0 (not locked), the LRU bit is updated to point to the way that is not being filled.

When the instruction cache is being filled, the pipeline is interlocked so no other processing can be performed in parallel. However, when a data cache is being filled, as long as the following instruction is not a load/store or cache instruction, that instruction can be executed in parallel with the data cache fill (the data cache fill is a non-blocking operation).

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

```
Address Error exception
Bus Error exception
```

cache : Fill with Lock (D)

Fill with Lock (D)

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 1 1	Base	func 1 1 1 1 1	offset
6	5	5	16

ALLEGREX™

Syntax:

```
cache  func(=0x1f),offset(base)
```

Description:

This instruction is used for explicitly filling a specific line in the data cache and locking it, if the line is not in the cache.

The 16-bit offset is sign-extended and added to the contents of the base register to generate the address of a cache line. The cache operation indicated by the 5-bit func code is performed on that cache line.

If the cache line at the generated address is not in the cache (a cache miss), the cache line is filled. If the line is already in the cache (a cache hit), no operation is performed. If the Lock bit = 0 (not locked), the LRU bit is updated to point to the way that is not being filled.

This instruction also sets the Lock bit. As a result, the LRU bit is held, and the target line will be locked and not be subject to replacement.

When the instruction cache is being filled, the pipeline is interlocked so no other processing can be performed in parallel. However, when a data cache is being filled, as long as the following instruction is not a load/store or cache instruction, that instruction can be executed in parallel with the data cache fill (the data cache fill is a non-blocking operation).

This instruction can be executed in user mode.

Operation:

```
vAddr ← GPR[base] + sign_extend(offset)
CacheOp(func, vAddr)
```

Exceptions:

Address Error exception

```
        Bus Error exception
```

sync

Synchronize Shared Memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
6						26																									

ALLEGREX™

Syntax:

```
sync
```

Description:

The pipeline is stalled until the CPU's cache writeback buffer and non-cache write buffer have emptied. This is done to block the execution of subsequent instructions.

Operation:

SyncOperation()

Exceptions:

None

ll

Load Linked

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 0 0 0 0	base	rt	offset
6	5	5	16

ALLEGREX™

Syntax:

```
ll  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

The word in memory at that address is loaded into register rt and the LLbit is set to 1.

Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if (vAddr$_{1..0}$ ≠ 0$^{2)}$ then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressT ranslation(vAddr, DATA, LOAD)
memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
GPR[rt] ← memword
LLbit ← 1
```

Exceptions:

```
Address Error exception
Bus Error exception
```

sc

Store Conditional

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 0 0 0	base	rt	offset
6	5	5	16

ALLEGREX™

Syntax:

```
sc  rt,offset(base)
```

Description:

The 16-bit offset is sign-extended and added to the contents of the base register to generate an address.

If the LLbit is 1, the contents of register rt are stored in memory at that address and 1 is returned in register rt.

If the LLbit is 0, no store is performed and 0 is returned in register rt.

Bus error exceptions are not reported because writing is done via a buffer. If a bus error occurs, it is handled by the system as an interrupt.

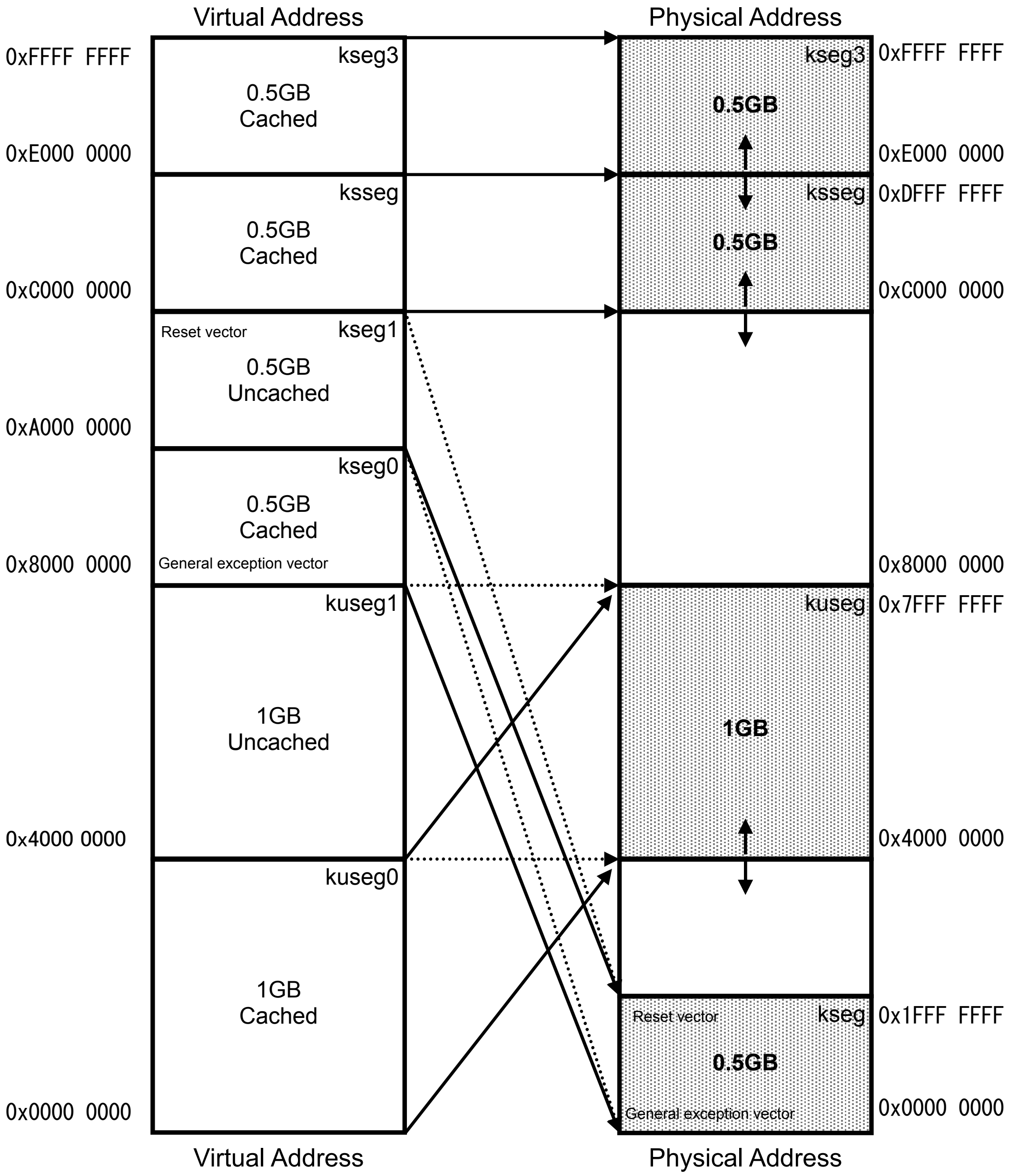
Operation:

```
vAddr ← sign_extend(offset) + GPR[base]
if ($vAddr_{1..0} \neq 0^2$) then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
dataword ← GPR[rt]
if (LLbit) then
    StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)
endif
GPR[rt] ← ($0^{31}$ || LLbit)
```

Exceptions:

```
Address Error exception
```

Virtual－Physical Address Map



CACHE Instruction func Code Table

func Code	Cache	Instruction	Available Modes
0	I	—	—
1	I	—	—
2	I	—	—
3	I	—	—
4	I	Index Invalidate	u / s/ k
5	I	—	—
6	I	Index Unlock	u / s/ k
7	I	—	—
8	I	Hit Invalidate	u / s/ k
9	I	—	—
10	I	Fill	u / s/ k
11	I	Fill with Lock	u / s/ k
12	I	—	—
13	I	—	—
14	I	—	—
15	I	—	—
16	D	—	—
17	D	—	—
18	D	—	—
19	D	—	—
20	D	Index Writeback Invalidate	u / s/ k
21	D	—	—
22	D	Index Unlock	u / s/ k
23	D	—	—
24	D	Create Dirty Exclusive	u / s/ k
25	D	Hit Invalidate	u / s/ k
26	D	Hit Writeback	u / s/ k
27	D	Hit Writeback Invalidate	u / s/ k
28	D	Create Dirty Exclusive with Lock	u / s/ k
29	D	—	—
30	D	Fill	u / s/ k
31	D	Fill with Lock	u / s / k

u: User mode, s: Supervisor mode, k: Kernel mode

CPU Instruction Set Code Table

Opcode

	Bit 28-26							
Bit 31-29	0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	COP1	COP2	COP3	BEQL	BNEL	BLEZL	BGTZL
3	VFPU0	VFPU1	*	VFPU3	SPECIAL2	*	*	SPECIAL3
4	LB	LH	LWL	LW	LBU	LHU	LWR	*
5	SB	SH	SWL	SW	*	*	SWR	CACHE
6	LL	LWC1	LWC2	*	VFPU4	LQUC2	LQC2	VFPU5
7	SC	SWC1	SWC2	*	VFPU6	SQUC2	SQC2	VFPU7

SPECIAL func

	Bit 2-0							
Bit 5-3	0	1	2	3	4	5	6	7
0	SLL		SRL/ROTR	SRA	SLLV		SRLV/ROTRV	SRAV
1	JR	JALR	MOVZ	MOVN	SYSCALL	BREAK	*	SYNC
2	MFHI	MTHI	MFLO	MTLO	*	*	CLZ	CLO
3	MULT	MULTU	DIV	DIVU	MADD	MADDU	*	*
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	*	*	SLT	SLTU	MAX	MIN	MSUB	MSUBU
6	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*

SPECIAL rs

	Bit23-21							
Bit25-24	0	1	2	3	4	5	6	7
0	SRL	ROTR						
1								
2								
3								

SPECIAL sa

	Bit 8-6							
Bit 10-9	0	1	2	3	4	5	6	7
0	SRLV	ROTRV						
1								
2								
3								

REGIMM sa

	Bit 18-16							
Bit 20-19	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1	*	*	*	*	*	*	*	*
2	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3	*	*	*	*	*	*	*	*

COPz rs

	Bit 23-21							
Bit 25-24	0	1	2	3	4	5	6	7
0	MFCz		CFCz	MFHCz	MTCz		CTCz	MTHCz
1	BC							
2	COPz							
3								

COPz rt (BC)

	Bit 18-16							
Bit 20-19	0	1	2	3	4	5	6	7
0	BCzF	BCzT	BCzFL	BCzTL				
1								
2								
3								

COP0 rs

	Bit 23-21							
Bit 25-24	0	1	2	3	4	5	6	7
0	MFC0				MTC0			
1	BC							
2	COPz							
3								

COP0 func

	Bit 2-0							
bit 4-3	0	1	2	3	4	5	6	7
0		*	*				*	
1	*							
2	*							
3	ERET							

SPECIAL2 func

	Bit 2-0							
Bit 5-3	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

SPECIAL2 rs

	Bit 23-21							
Bit 25-24	0	1	2	3	4	5	6	7
0								
1								
2								
3								

SPECIAL3 func

	Bit 2-0							
Bit 5-3	0	1	2	3	4	5	6	7
0	EXT				INS			
1								
2								
3								
4	BSHFL							
5								
6								
7								

BSHFL sa

	Bit 8-6							
Bit 10-9	0	1	2	3	4	5	6	7
0			WSBH	WSBW				
1								
2	SEB				BITREV			
3	SEH							

🟨	**MIPS-I**	🟪	**FPU**
🟧	**MIPS-II**	🟦	***ALLEGREX™***