7.4 MIPS 23条扩展指令介绍

本节介绍乘除法运算、对LO/HI寄存器的读写、内存半字和字节的存取操作、CP0的异常处理指令和CP0寄存器的读写，以及一些跳转指令

Mips 指令集（共 23 条）								
指令	指令说明	指令格式	OP31-26	RS25-21	RT20-16	RD15-11	SA10-6	FUNCT5-0
div	除	DIV rs, rt	000000			00000	00000	011010
divu	除（无符号）	DIVU rs, rt	000000			00000		011011
mult	乘	MULT rs, rt	000000			00000		011000
multu	乘（无符号）	MULTU rs, rt	000000			00000		011001
bgez	大于等于 0 时分支	BGEZ rs, offset	000001		00001			
jalr	跳转至寄存器所指地址，返回地址保存	JALR rs	000000		00000			001001

lbu	取字节(无符号)	LBU rt, offset(base)	100100					
lhu	取半字(无符号)	LHU rt, offset(base)	100101					
lb	取字节	LBU rt, offset(base)	100000					
lh	取半字	LHU rt, offset(base)	100001					
sb	存字节	SB rt, offset(base)	101000					
sh	存半字	SH rt, offset(base)	101001					
break	断点	BREAK	000000					001101
syscall	系统调用	SYSCALL	000000					001100
eret	异常返	ERET	010000	10000	00000	00000	00000	011000

	回							
mfhi	读 Hi 寄存器	MFHI rd	000000	**00000**	00000		00000	010000
mflo	读 Lo 寄存器	MFLO rd	000000	**00000**	00000		00000	010010
mthi	写 Hi 寄存器	MTHI rd	000000		00000	00000	00000	010001
mtlo	写 Lo 寄存器	MTLO rd	000000		00000	00000	00000	010011
mfc0	读 CP0 寄存器	MFC0 rt, rd	010000	**00000**			00000	00000
mtc0	写 CP0 寄存器	MTC0 rt, rd	010000	**00100**			00000	00000
clz	前导零计数	CLZ rd, rs	011100				00000	100000
teq	相等异常	TEQ rs, rt	000000					110100

MULT

格式：MULT rs, rt

目的：32位有符号数乘法

描述：(HI, LO) ←rs* rt

在通用寄存器rs和rt中的数据当做有符号数来进行乘法运算产生一个64位的结果， 结果的低32位写入LO寄存器，高32位写入HI寄存器。 在任何情况下不会产生算术异常。

操作：

$prod \leftarrow GPR[rs]_{31..0} * GPR[rt]_{31..0}$

$LO \leftarrow prod_{31..0}$

$HI \leftarrow prod_{63..32}$

MULTU

格式：MULTU rs, rt

目的：32位无符号数乘法

描述：(HI, LO) ←rs * rt

在通用寄存器rs和rt中的数据当做无符号数来进行乘法运算产生一个64位的结果， 结果的低32位写入LO寄存器，高32位写入HI寄存器。 在任何情况下不会产生算术异常。

操作：

prod← (0 || GPR[rs]31..0) * (0 || GPR[rt]31..0)

LO ← $prod_{31..0}$

HI ← $prod_{63..32}$

DIV

格式：DIV rs, rt

目的：32位有符号数除法

描述：(HI, LO) ←rs / rt

将通用寄存器rs和rt中的数据当做有符号数来进行除法运算。32位的商数被存入寄存器LO，32位的余数被存入HI。在任何情况下不会有算术异常的发生。

操作：

$q \leftarrow GPR[rs]_{31..0}$ div $GPR[rt]_{31..0}$

$LO \leftarrow q$

$r \leftarrow GPR[rs]_{31..0}$ mod $GPR[rt]_{31..0}$

$HI \leftarrow r$

DIVU

格式：DIVU rs, rt

目的：32位无符号数除法

描述：(HI, LO) ←rs / rt

将通用寄存器rs和rt中的数据当做无符号数来进行除法运算。32位的商数被存入寄存器LO，32位的余数被存入HI。在任何情况下不会有算术异常的发生。

操作：

$q \leftarrow (0 \,||\, GPR[rs]_{31..0})\ div\ (0 \,||\, GPR[rt]_{31..0})$

$r \leftarrow (0 \,||\, GPR[rs]_{31..0})\ mod\ (0 \,||\, GPR[rt]_{31..0})$

$LO \leftarrow sign\_extend(q_{31..0})$

$HI \leftarrow sign\_extend(r_{31..0})$

MFLO

格式：MFLO rd

目的：复制特殊寄存器LO的内容到通用寄存器

描述：rd ←LO

特殊寄存器LO中的数据复制到通用寄存器rd中。

操作：

GPR[rd] ← LO

MTHI

格式：MTHI rs

目的：复制通用寄存器的内容到特殊寄存器HI中

描述：HI ←rs

通用寄存器rs中的内容复制到特殊寄存器HI中。

操作：

HI ← GPR[rs]

MTLO

格式：MTLO rs

目的：复制通用寄存器的内容到特殊寄存器LO中

描述：LO ←rs

通用寄存器rs中的内容复制到特殊寄存器LO中。

操作：

LO ← GPR[rs]

MFHI

格式：MFHI rd

目的：复制特殊寄存器HI的内容到通用寄存器

描述：rd ←HI

特殊寄存器HI中的数据复制到通用寄存器rd中。

操作：

GPR[rd] ← HI

LB

格式：LB rt, offset(base)

目的：从内存加载一个有符号字节

描述：rt ←memory[base+offset]

在内存中通过基地址和偏移量相加所得的有效地址中取一个8bit的字节通过符号扩展后存入rt寄存器中。

操作：

vAddr ← sign_extend(offset) + GPR[base]

(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)

$pAddr \leftarrow pAddr_{PSIZE-1..2} \,||\, (pAddr_{1..0} \text{ xor } ReverseEndian^{2})$

memword← LoadMemory (CCA, BYTE, pAddr, vAddr, DATA)

$byte \leftarrow vAddr_{1..0} \text{ xor } BigEndianCPU^{2}$

$GPR[rt] \leftarrow sign\_extend(memword_{7+8*byte..8*byte})$

LBU

格式：LBU rt, offset(base)

目的：从内存加载一个无符号字节

描述：rt ←memory[base+offset]

在内存中通过基地址和偏移量相加所得的有效地址中取一个8bit的字节通过0扩展后存入rt寄存器中。

操作：

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 $BigEndianCPU^{2}$

GPR[rt] ← zero_extend($memword_{7+8*byte..8*byte}$)

LH

格式：LH rt, offset(base)

目的：从内存加载一个有符号半字

描述：rt ←memory[base+offset]

在内存中通过基地址和偏移量相加所得的有效地址中取一个半字通过符号扩展后存入rt寄存器中。

操作：

vAddr ← sign_extend(offset) + GPR[base]

if vAddr0 ≠ 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 (BigEndianCPU || 0)

GPR[rt] ← sign_extend($memword_{15+8*byte..8*byte}$)

LHU

格式：LHU rt, offset(base)

目的：从内存加载一个无符号半字

描述：rt ←memory[base+offset]

在内存中通过基地址和偏移量相加所得的有效地址中取一个半字通过0扩展后存入rt寄存器中。

操作：

vAddr ← sign_extend(offset) + GPR[base]

if vAddr0 ≠ 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 (BigEndianCPU || 0)

GPR[rt] ← zero_extend($memword_{15+8*byte..8*byte}$)

SB

格式：SB rt, offset(base)

目的：存一字节到内存

描述：memory[base+offset] ←rt

在rt寄存器中最低8位的数据被存入到由基地址和偏移量相加所得有效地址的内存中。

操作：

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 $BigEndianCPU^2$

dataword ← $GPR[rt]_{31-8*bytesel..0}$ || $0^{8*bytesel}$

StoreMemory (CCA, BYTE, dataword, pAddr, vAddr, DATA)

SH

格式：SH rt, offset(base)

目的：存一个半字到内存

描述：memory[base+offset] ←rt

在rt寄存器中最低16位的数据被存入到由基地址和偏移量相加所得有效地址的内存中。

操作：

```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr0 ≠ 0 then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, STORE)
pAddr ← pAddr_PSIZE-1..2 || (pAddr_11..0 xor (ReverseEndian || 0))
bytesel← vAddr_11..0 xor (BigEndianCPU || 0)
dataword← GPR[rt]_31–8*bytesel..0 || 0^8*bytesel
StoreMemory (CCA, HALFWORD, dataword, pAddr, vAddr, DATA)
```

BGEZ

格式：BGEZ rs, offset

目的：检测通用寄存器的值然后进行pc相关的跳转

描述：如果 rs >= 0那么将会在指令延时槽执行后跳转到现在pc与偏移量offset（如果是16位需扩展到18位）相加后所得的指令。

操作：

I: target_offset ← sign_extend(offset || 0^2)

condition ← GPR[rs] ≥ 0^{GPRLEN}

I+1: if condition then

PC ← PC + target_offset

endif

JALR

格式: JALR rs (rd = 31 implied)

JALR rd, rs

目的: 执行一个在寄存器中存放指令地址的过程调用

描述: rd ←return_addr, PC ←rs

使用寄存器的跳转指令，并且带有链接功能，指令的跳转地址在寄存器rs中，跳转发生时指令的返回地址存在31号寄存器中。

操作:

I: temp ← GPR[rs]

GPR[rd] ← PC + 8

（Mars上执行的是PC+4）

I+1: if $Config1_{CA}$ = 0 then

PC ← temp

else

...

PC← $temp_{GPRLEN-1..1}$ || 0

ISAMode ← $temp_0$

endif

...

BREAK

格式：BREAK

目的：引起一个断点异常

描述：当一个断点异常发生时，将立刻并且不可控制的转入异常处理。

操作：

SignalException(Breakpoint)

注：CP0中的cause寄存器要有硬件通路支持写入正确的异常类型码，同时要有硬件通路支持CP0中的EPC寄存器写入PC+4（异常返回地址，即break指令的下一条指令地址）

SYSCALL

格式：SYSCALL

目的：引发一个系统调用异常

描述：当一个系统调用发生时，将立刻并且不可控制的转入异常处理。

操作：

SignalException(SystemCall)

注：CP0中的cause寄存器要有硬件通路支持写入正确的异常类型码，同时要有硬件通路支持CP0中的EPC寄存器写入PC+4（异常返回地址，即break指令的下一条指令地址）

TEQ

格式：TEQ rs, rt

目的：比较寄存器值根据条件引发异常

描述：比较寄存器rs和rt的值，若相等则引发一个自陷异常

操作：

```
if GPR[rs] = GPR[rt] then
SignalException(Trap)
endif
```

注：CP0中的cause寄存器要有硬件通路支持写入正确的异常类型码，同时要有硬件通路支持CP0中的EPC寄存器写入PC+4（异常返回地址，即break指令的下一条指令地址）

ERET

格式：ERET

目的：从异常中断返回

描述：在所有中断处理过程结束后返回到中断指令。ERET不执行下一条指令。

操作：

```
if StatusERL = 1 then
temp ← ErrorEPC
StatusERL ← 0
else
temp← EPC
StatusEXL ← 0
endif
...
...
if IsMIPS16Implemented() then
PC ← temp31..1 || 0
ISAMode ← temp0
else
PC ← temp
endif
LLbit ← 0
```

将CP0的EPC寄存器中的地址送PC寄存器，从异常返回主程序

MFC0

格式： MFC0 rt, rd （实现该格式，将CP0的rd寄存器内容送CPU的rt寄存器）

MFC0 rt, rd, sel

目的：把一个数据从特殊寄存器移到通用寄存器

描述：rt ←CPR[0,rd,sel]

由rd和sel选择协处理器0中的特殊寄存器，把它的内容转移到通 用寄存器rt中。

操作：

data ← CPR[0,rd,sel]

GPR[rt] ←data

MTC0

格式：MTC0 rt, rd （实现该格式，将CPU的rt寄存器内容送CP0的rd寄存器）

MTC0 rt, rd, sel

目的：把一个数据从通用寄存器移到特殊寄存器

描述：CPR[r0, rd, sel] ← rt

由rd和sel选择协处理器0中的特殊寄存器，把通用寄存器rt中的内容转移到特殊寄存器中。

操作：

data ← GPR[rt]

CPR[0,rd,sel] ← data

CLZ

格式：CLZ rd, rs

目的：计算32位字中的前导零个数

描述：计算rs寄存器中32位数前导零的个数，存入rd寄存器中。

操作：

```
temp ← 32
for i in 31 .. 0
    if GPR[rs]i = 1 then
        temp ← 31 – i
        break
    endif
endfor
GPR[rd] ← temp
```

谢谢聆听

Thank You