7.3 MIPS 31条指令介绍

MIPS 指令集（共 31 条）									
助记符	指令格式						示例	示例含义	操作及其解释
Bit #	31..26	25..21	20..16	15..11	10..6	5..0			
R-type	op	rs	rt	rd	shamt	func			
add	000000	rs	rt	rd	00000	100000	add $1,$2,$3	$1=$2+$3	rd <- rs + rt ； 其中 rs=$2，rt=$3, rd=$1
addu	000000	rs	rt	rd	00000	100001	addu $1,$2,$3	$1=$2+$3	rd <- rs + rt ； 其中 rs=$2，rt=$3, rd=$1, 无符号数 无溢出
sub	000000	rs	rt	rd	00000	100010	sub $1,$2,$3	$1=$2-$3	rd <- rs - rt ； 其中 rs=$2，rt=$3, rd=$1
subu	000000	rs	rt	rd	00000	100011	subu $1,$2,$3	$1=$2-$3	rd <- rs - rt ； 其中 rs=$2，rt=$3, rd=$1, 无符号数 无溢出
and	000000	rs	rt	rd	00000	100100	and $1,$2,$3	$1=$2 & $3	rd <- rs & rt ； 其中 rs=$2，rt=$3, rd=$1
or	000000	rs	rt	rd	00000	100101	or $1,$2,$3	$1=$2 | $3	rd <- rs | rt ； 其中 rs=$2，rt=$3, rd=$1
xor	000000	rs	rt	rd	00000	100110	xor $1,$2,$3	$1=$2 ^ $3	rd <- rs xor rt ； 其中 rs=$2，rt=$3, rd=$1(异或)
nor	000000	rs	rt	rd	00000	100111	nor $1,$2,$3	$1=~($2 | $3)	rd <- not(rs | rt) ； 其中 rs=$2，rt=$3, rd=$1(或非)

slt	000000	rs	rt	rd	00000	101010	slt $1,$2,$3	if($2<$3) $1=1 else $1=0	if (rs < rt) rd=1 else rd=0 ； 其中 rs=$2, rt=$3, rd=$1
sltu	000000	rs	rt	rd	00000	101011	sltu $1,$2,$3	if($2<$3) $1=1 else $1=0	if (rs < rt) rd=1 else rd=0 ； 其中 rs=$2, rt=$3, rd=$1 (无符号数)
sll	000000	00000	rt	rd	shamt	000000	sll $1,$2,10	$1=$2<<10	rd <- rt << shamt ； shamt 存放移位的位数, 也就是指令中的立即数,其中 rt=$2, rd=$1
srl	000000	00000	rt	rd	shamt	000010	srl $1,$2,10	$1=$2>>10	rd <- rt >> shamt ； (logical) ， 其中 rt=$2, rd=$1
sra	000000	00000	rt	rd	shamt	000011	sra $1,$2,10	$1=$2>>10	rd <- rt >> shamt ； (arithmetic) 注意符号位保留 其中 rt=$2, rd=$1
sllv	000000	rs	rt	rd	00000	000100	sllv $1,$2,$3	$1=$2<<$3	rd <- rt << rs ； 其中 rs=$3，rt=$2, rd=$1
srlv	000000	rs	rt	rd	00000	000110	srlv $1,$2,$3	$1=$2>>$3	rd <- rt >> rs ；(logical)其中 rs=$3，rt=$2, rd=$1
srav	000000	rs	rt	rd	00000	000111	srav $1,$2,$3	$1=$2>>$3	rd <- rt >> rs ； (arithmetic) 注意符号位保留 其中 rs=$3，rt=$2, rd=$1

jr	000000	rs	00000	00000	00000	001000	jr $31	goto $31	PC <- rs
I-type	op	rs	rt	immediate					
addi	001000	rs	rt	immediate			addi $1,$2,100	$1=$2+100	rt <- rs + (sign-extend)immediate ； 其中 rt=$1,rs=$2 有溢出
addiu	001001	rs	rt	immediate			addiu $1,$2,100	$1=$2+100	rt <- rs + (sign-extend)immediate ； 其中 rt=$1,rs=$2 无溢出
andi	001100	rs	rt	immediate			andi $1,$2,10	$1=$2 & 10	rt <- rs & (zero-extend)immediate ； 其中 rt=$1,rs=$2
ori	001101	rs	rt	immediate			andi $1,$2,10	$1=$2 | 10	rt <- rs | (zero-extend)immediate ； 其中 rt=$1,rs=$2
xori	001110	rs	rt	immediate			andi $1,$2,10	$1=$2 ^ 10	rt <- rs xor (zero-extend)immediate ； 其中 rt=$1,rs=$2
lui	001111	00000	rt	immediate			lui $1,100	$1=100*65536	rt <- immediate*65536 ； 将 16 位立即数放到目标寄存器高 16 位，目标寄存器的低 16 位填 0

lw	100011	rs	rt	immediate	lw $1,10($2)	$1=memory[$2+10]	rt <- memory[rs + (sign-extend)immediate] ； rt=$1,rs=$2
sw	101011	rs	rt	immediate	sw $1,10($2)	memory[$2+10]=$1	memory[rs + (sign-extend)immediate] <- rt ； rt=$1,rs=$2
beq	000100	rs	rt	immediate	beq $1,$2,10	if($1==$2) goto PC+4+40	if (rs == rt) PC <- PC+4 + (sign-extend)immediate<<2
bne	000101	rs	rt	immediate	bne $1,$2,10	if($1!=$2) goto PC+4+40	if (rs != rt) PC <- PC+4 + (sign-extend)immediate<<2
slti	001010	rs	rt	immediate	slti $1,$2,10	if($2<10) $1=1 else $1=0	if (rs <(sign-extend)immediate) rt=1 else rt=0 ； 有符号比较 其中 rs=$2，rt=$1
sltiu	001011	rs	rt	immediate	sltiu $1,$2,10	if($2<10) $1=1 else $1=0	if (rs <(sign-extend)immediate) rt=1 else rt=0 ； 无符号比较 其中 rs=$2，rt=$1
J-type	op	address					
j	000010	address			j 10000	goto 10000	PC <- (PC+4)[31..28],address,0,0 ； address = 10000*4
jal	000011	address			jal 10000	$31<-PC+4; goto 10000	$31<-PC+4； PC <- (PC+4)[31..28],address,0,0 ； address = 10000*4

ADD

格式： ADD rd, rs, rt

目的：32位数相加

描述：rd ← rs + rt

将通用寄存器中存的32位数据rs与rt相加产生一个32位数据存入目标寄存器rd。

- 如果发生了溢出，则rd不改变并且产生一个溢出的异常。
- 如果相加不溢出，则产生的32位数据直接存入目标寄存器rd。

操作:

temp ←(GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) + (GPR[rt]$_{31}$||GPR[rt]$_{31..0}$)

if temp$_{32}$ ≠temp$_{31}$ then

SignalException(IntegerOverflow)

else

GPR[rd] ←temp

Endif

ADDU

格式：ADDU rd, rs, rt

目的：32位数据相加

描述：rd ← rs + rt

将通用寄存器中存的32位数据rs与rt相加产生一个32位数据存入目标寄存器rd。

在任何情况下都不会有溢出的异常。

操作：

temp ← GPR[rs] + GPR[rt]

GPR[rd] ← temp

ADDI

格式：ADDI rt, rs, immediate

目的：使32位数据与一个立即数相加

描述：rt ← rs + immediate

16位有符号立即数与通用寄存器rs中的32位数相加产生一个32位的数存入目标寄存器rt。

- 如果发生了溢出，则rt不改变并且产生一个溢出的异常。
- 如果相加不溢出，则结果存入目标寄存器rt。

操作：

temp ←(GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) + sign_extend(immediate)

if temp$_{32}$ ≠ temp$_{31}$ then

SignalException(IntegerOverflow)

else

GPR[rt] ←temp

endif

ADDIU

格式: ADDIU rt, rs, immediate

目的: 使32位数据与一个立即数相加

描述: rt ← rs + immediate

一个16位有符号的立即数与通用寄存器rs中的32位数相加产生一个32位的数存入目标寄存器rt。

在任何情况下都不会有溢出的异常。

操作:

temp ←GPR[rs] + sign_extend(immediate)

GPR[rt] ← temp

AND

格式：AND rd, rs, rt

目的：按位逻辑与

描述：rd ← rs AND rt

将通用寄存器rs和rd中的数据每一位做按位与操作，将结果存入目标寄存器rd中。

操作：

GPR[rd] ← GPR[rs] and GPR[rt]

ANDI

格式：ANDI rt, rs, immediate

目的：与一个常数做按位逻辑与

描述：rt ← rs AND immediate

将16位立即数做0扩展后与通用寄存器rs中的32位数据做按位与，将结果存入目标寄存器rt。

操作：

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

BEQ

格式：BEQ rs, rt, offset

目的：比较通用寄存器的值，然后做pc相关的分支跳转

描述：比较通用寄存器的值，然后做pc相关的分支跳转。

如果 rs = rt，那么将offset左移两位，再进行符号扩展到32位与当前pc相加，形成有效转移地址，转到该地址。

如果rs != rt，则继续执行下条指令。

操作：

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

condition ←(GPR[rs] = GPR[rt])

I+1: if condition then

PC ←PC + target_offset

endif

BNE

格式：BNE rs, rt, offset

目的：比较通用寄存器的值，然后做pc相关的分支跳转

描述：如果 rs != rt ，那么将会跳转到现在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。

如果rs = rt，则继续执行。

操作：

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

condition ← (GPR[rs] ≠ GPR[rt])

I+1: if condition then

PC ← PC + target_offset

endif

J

格式：J target

目的：在256MB的范围内跳转

描述：该指令无条件跳转到一个绝对地址，instr_index有26位，在左移过后访问空间能达到256M。

操作：

I:

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

JAL

格式：JAL target

目的：在256MB范围内执行一个过程调用

描述：在跳转到指定地址执行子程序调用的同时，**在31号寄存器中存放返回地址（当前地址后的第二条指令地址）**。

操作：

I：GPR[31] ←PC + 8

I+1：PC ←$PC_{GPRLEN-1..28}$ || instr_index || 0^2

JR

格式：JR rs

目的：使用寄存器的跳转指令

描述：PC ← rs

跳转地址存放在通用寄存器rs中，直接跳转到寄存器所存地址。

操作：

I: temp ← GPR[rs]

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

PC ← temp

else

PC ← $temp_{GPRLEN-1..1}$ || 0

ISAMode ← $temp_0$

endif

LUI

格式：LUI rt, immediate

目的：把一个立即数载入到寄存器的高位，低位补0

描述：rt ← immediate || 0^{16}

将一个16位的立即数载入到通用寄存器rt的高位，低16位补0。

操作：

GPR[rt] ← immediate || 0^{16}

LW

格式：LW rt, offset(base)

目的：从内存读取一个字的有符号数据

描述：rt ← memory[base+offset]

从内存中基地址加偏移量所得到的准确地址中的内容加载到通用寄存器rt中。

操作：

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

NOR

格式：NOR rd, rs, rt

目的：按位逻辑或非

描述：rd ← rs NOR rt

将通用寄存器rs和rt中的数据每一位做按位或非操作，将结果存入目标寄存器rd中。

操作：

GPR[rd] ← GPR[rs] nor GPR[rt]

OR

格式：OR rd, rs, rt

目的：按位逻辑或

描述：rd ← rs or rt

将通用寄存器rs和rt中的数据每一位做按位或操作，将结果存入目标寄存器rd中。

操作：

GPR[rd] ← GPR[rs] or GPR[rt]

ORI

格式：ORI rt, rs, immediate

目的：和一个常数做按位逻辑或

描述：rt ← rs or immediate

将通用寄存器rs和经过0扩展的立即数每一位做按位或操作，将结果存入目标寄存器rd中。

操作：

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

SLL

格式：SLL rd, rt, sa

目的：通过数字填充逻辑左移

描述：rd ← rt << sa

将通用寄存器rt的内容左移sa位，空余出来的位置用0来填充，把结果存入rd寄存器。

操作：

s ← sa

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

GPR[rd] ← temp

SLLV

格式：SLLV rd, rt, rs

目的：通过数字填充逻辑左移

描述：rd ← rt << rs

将通用寄存器rt的内容逻辑左移，左移的位数保存在rs寄存器中，空余出来的位置用0来填充，把结果存入rd寄存器。

操作：

$s \leftarrow GPR[rs]_{4..0}$

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

$GPR[rd] \leftarrow temp$

SLT

格式：SLT rd, rs, rt

目的：通过小于的比较来记录结果

描述：rd ←(rs < rt)

比较在rs和rt寄存器中保存的有符号数，用boolean值保存结果到rd寄存器中。

如果rs小于rt，则结果为1，反之结果为0。算数比较不会引起溢出异常。

操作：

if GPR[rs] < GPR[rt] then

GPR[rd] ← $0^{GPRLEN-1}$ || 1

else

GPR[rd] ← 0^{GPRLEN}

endif

SLTI

格式：SLTI rt, rs, immediate

目的：通过跟立即数小于的比较来记录结果

描述：rt ←(rs < immediate)

比较rs中的数和经过符号扩展的16位立即数，用boolean值保存结果到rd寄存器中。如果rs中的数小于立即数，则结果为1，反之结果为0。算数比较不会引起溢出异常。

操作：

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

SLTIU

格式：SLTIU rt, rs, immediate

目的：通过跟立即数进行无符号小于的比较来记录结果

描述：rt ←(rs < immediate)

把rs中的数和经过符号扩展的16位立即数进行无符号小于比较，用boolean值保存结果到rd寄存器中。如果rs小于符号扩展的立即数，则结果为1，反之结果为0。算数比较不会引起溢出异常。

操作：

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

SLTU

格式：SLTU rd, rs, rt

目的：通过无符号小于的比较来记录结果

描述：rd ←(rs < rt)

比较在rs和rt寄存器中保存的无符号数，用boolean值保存结果到rd寄存器中。

如果rs小于rt，则结果为1，反之结果为0。算数比较不会引起溢出异常。

操作：

if (0 || GPR[rs]) < (0 || GPR[rt]) then

GPR[rd] ← $0^{GPRLEN-1}$ || 1

else

GPR[rd] ← 0^{GPRLEN}

endif

SRA

格式：SRA rd, rt, sa

目的：通过数字填充算术右移

描述：rd ←rt >> sa (arithmetic)

将通用寄存器rt中的32位内容右移sa位，高位用rt[31]来填充，结果存入通用寄存器rd。

操作：

s ← sa

temp ← $(GPR[rt]31)^s$ || $GPR[rt]_{31..s}$

GPR[rd] ← temp

SRAV

格式：SRAV rd, rt, rs

目的：通过数字填充算术右移

描述：rd ←rt >> rs (arithmetic)

将通用寄存器rt中的32位内容右移，高位用rt[31]来填充，结果存入通用寄存器rd。右移的位数由通用寄存器rs中的0-4bit确定。

操作：

$s \leftarrow GPR[rs]_{4..0}$

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

$GPR[rd] \leftarrow temp$

SRL

格式：SRL rd, rt, sa

目的：通过数字填充逻辑右移

描述：rd ←rt >> sa (logical)

将通用寄存器rt中的32位内容右移sa位，高位用0来填充，结果存入通用寄存器rd。

操作：

s ← sa

temp ← 0^s || $GPR[rt]_{31..s}$

GPR[rd] ← temp

SRLV

格式：SRLV rd, rt, rs

目的：通过数字填充逻辑右移

描述：rd ←rt >> rs (logical)

将通用寄存器rt中的32位内容右移，高位用rt[31]来填充，结果存入通用寄存器rd。右移的位数由通用寄存器rs中的0-4bit确定。

操作：

$s \leftarrow GPR[rs]_{4..0}$

$temp \leftarrow 0^{s} \,||\, GPR[rt]_{31..s}$

$GPR[rd] \leftarrow temp$

SUB

格式：SUB rd, rs, rt

目的：与32位数相减

描述：rd ←rs - rt

将通用寄存器中存的32位数据rs与rt相减产生一个32位数据存入目标寄存器rd。

- 如果发生了溢出，则rd不改变并且产生一个溢出的异常。
- 如果不溢出，则产生的32位数据直接存入目标寄存器rd。

操作：

temp ← (GPR[rs]$_{31}$||GPR[rs]$_{31..0}$) - (GPR[rt]$_{31}$||GPR[rt]$_{31..0}$)

if temp$_{32}$ ≠ temp$_{31}$ then

SignalException(IntegerOverflow)

else

GPR[rd] ← temp$_{31..0}$

endif

SUBU

格式：SUBU rd, rs, rt

目的：32位数据相减

描述：rd ←rs - rt

将通用寄存器中存的32位数据rs与rt相减产生一个32位数据存入目标寄存器rd。

在任何情况下都不会有溢出的异常。

操作：

temp ← GPR[rs] - GPR[rt]

GPR[rd] ← temp

SW

格式：SW rt, offset(base)

目的：存一个字到内存

描述：memory[base+offset] ←rt

将通用寄存器rt中的32位数据存入内存中的有效地址，有效地址由基地址和16位偏移量相加所得。

操作：

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)

XOR

格式：XOR rd, rs, rt

目的：按位逻辑异或

描述：rd ←rs XOR rt

将通用寄存器rs和rt中的内容按位进行异或操作，将结果存入rd中。

操作：

GPR[rd] ← GPR[rs] xor GPR[rt]

XORI

格式：XORI rt, rs, immediate

目的：和一个常数做按位逻辑异或

描述：rt ←rs XOR immediate

将通用寄存器rs和经过0扩展的立即数每一位做按位异或操作，将结果存入目标寄存器rd中。

操作：

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

谢谢聆听

Thank You