



yha)

[博客园](#) [首页](#) [新随笔](#) [联系](#) [订阅](#) [管理](#)

mips的32个寄存器

MIPS comes with 32 general purpose registers named \$0. . . \$31

Registers also have symbolic names reflecting their conventional use:

\$0 \$zero constant 0

\$1 \$at used by assembler

\$2 \$v0 function result

\$3 \$v1 function result

\$4 \$a0 argument 1

\$5 \$a1 argument 2

\$6 \$a2 argument 3

\$7 \$a3 argument 4

\$8 \$t0 unsaved temporary

\$9 \$t1 unsaved temporary

\$10 \$t2 unsaved temporary

\$11 \$t3 unsaved temporary

\$12 \$t4 unsaved temporary

\$13 \$t5 unsaved temporary

\$14 \$t6 unsaved temporary

\$15 \$t7 unsaved temporary

\$16 \$s0 saved temporary

\$17 \$s1 saved temporary

\$18 \$s2 saved temporary

\$19 \$s3 saved temporary

\$20 \$s4 saved temporary

\$21 \$s5 saved temporary

\$22 \$s6 saved temporary

\$23 \$s7 saved temporary

\$24 \$t8 unsaved temporary
\$25 \$t9 unsaved temporary
\$26 \$k0 reserved for EXCEPTION
\$27 \$k1 reserved for EXCEPTION
\$28 \$gp pointer to global data
\$29 \$sp stack pointer
\$30 \$fp frame pointer
\$31 \$ra return address

寄存器号	符号名	用途
0	始终为0	看起来象浪费,其实很有用
1	at	保留给汇编器使用
2-3	v0,v1	函数返回值
4-7	a0-a3	前头几个函数参数
8-15	t0-t7	临时寄存器,子过程可以不保存就使用
24-25	t8,t9	同上
16-23	s0-s7	寄存器变量,子过程要使用它必须先保存 然后在退出前恢复以保留调用者需要的值
26,27	k0,k1	保留给异常处理函数使用
28	gp	global pointer;用于方便存取全局或者静态变量
29	sp	stack pointer
30	s8/fp	第9个寄存器变量;子过程可以用它做frame pointer
31	ra	返回地址

硬件上这些寄存器并没有区别(除了0号),区分的目的是为了不同的编译器产生的代码可以通用

=====

lui 中i表示加载常数

li r, c:加载16bit或32bit常数到r

lui r, c:加载16bit常数到r的高16位load constant halfword c into upper halfword of register r
(translation of pseudo instructions)

伪指令 翻译的实际指令

not r, s ==> nor r, s, \$0

move r, s ==> or r, s, \$0

li r, c ==> ori r, \$0, c load immediate (c: 16 bit constant)

li r, 0xABCDEF00==> lui \$at, 0xABCD和ori r, \$at, 0xEF00 (c: 32 bit constant)

and \$t0, \$t0, 0xFFFFF00==> lui \$at, 0xFFFF

ori \$at, 0xFF00

and \$t0, \$t0, \$at

.ascii s ASCII encoded characters of string s

.asciiz s like .ascii, null-terminated

.word w1, w2, . . . 32-bit words w1, w2, . . .

.half h1, h2, . . . 16-bit halfwords h1, h2, . . .

.byte b1, b2, . . . 8-bit bytes b1, b2, . . .

.float f1, f2, . . . 32-bit single precision floating point numbers f1, f2, . . .

.double d1, d2, . . . 64-bit double precision floating point numbers d1, d2, . . .

.space n n zero bytes

使用la伪指令访问数据区

```
la $t0, str
lb $t1, ($t0) # access byte at address $t0 ('f')
add $t0, $t0, 3
lb $t2, ($t0) # access byte at address $t0 + 3 ('b')
```

```
.data
str: .asciiz "foobar"
```

load word/halfword/byte at address a into target register r

```
lw r, a
lh r, a sign extension
lb r, a sign extension
lhu r, a no sign extension
lbu r, a no sign extension
store word/halfword/byte in register r at address a
sw r, a
sh r, a stores low halfword
sb r, a stores low byte
```

Example (copy a sequence of n bytes from address src to address dst):

```
.text
.globl __start
__start:
    # length n of byte sequence - 1
    li    $t0, 5
copy:
    lb     $t1, src($t0) # pseudo! (src: 32 bits wide)
    sb     $t1, dst($t0)
    sub    $t0, $t0, 1
    bgez   $t0, copy
.data
```

src: .byte 0x11, 0x22, 0x33, 0x44, 0x55, 0x66

dst: .space 6

=====

<http://www.mips-in-china.com/Study/ShowArticle.asp?ArticleID=147>

mfc0 - move from c0

cfc0 - copy from c0

```
mfc0    t0,c0_status
lui     at,0x1000
ori     at,at,0x1f
or      t0,t0,at
xori    t0,t0,0x1f
mtc0    t0,c0_statu
```

于协处理器CP0的访问，需要使用特别的指令。这些指令属于“特权级指令”，只有在内核态(Kernel Mode)下才能执行。如果在用户态

下，会引起一个异常(Exception)。

对CP0的主要操作有以下的指令：

mfcc0 rt, rd 将CP0中的rd寄存器内容传输到rt通用寄存器；

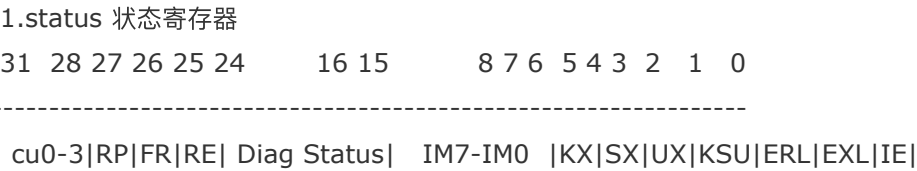
mtcc0 rt, rd 将rt通用寄存器中内容传输到CP0中寄存器rd；

mfhi/mflo rt 将CP0的hi/lo寄存器内容传输到rt通用寄存器中；

mthi/mtlo rt 将rt通用寄存器内容传输到CP0的hi/lo寄存器中；

当MIPS体系结构演进到MIPS IV的64位架构后，新增了两条指令dmfc0和dmtc0，向CP0的寄存器中读/写一个64bit的数据。

r4k MIPS CPU中和异常相关的控制寄存器(这些寄存器由协处理器cp0控制,有独立的存取方法)有：



其中KSU,ERL,EXL,IE位在这里很重要：

KSU: 模式位 00 -kernel 01--Supervisor 10--User

ERL: error level,0->normal,1->error

EXL: exception level,0->normal,1->exception,异常发生是EXL自动置1

IE: interrupt Enable, 0 -> disable interrupt,1->enable interrupt

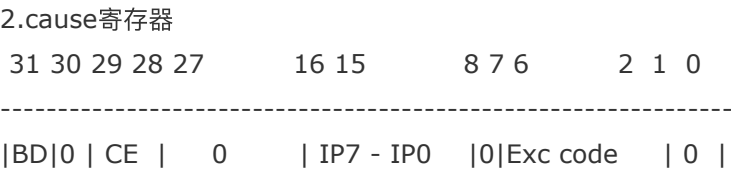
(IM位则可以用于enable/disable具体某个中断,ERL||EXL=1 也使得中断不能响应)

系统所处的模式由KSU,ERL,EXL决定：

User mode: KSU = 10 && EXL=0 && ERL=0

Supervisor mode(never used): KSU=01 && EXL=0 && ERL=0

Kernel mode: KSU=00 || EXL=1 || ERL=1



异常发生时cause被自动设置

其中：

BD指示最近发生的异常指令是否在delay slot中

CE 发生coprocessor unusable异常时的coprocessor编号(mips有4个cp)

IP: interrupt pending, 1->pending,0->no interrupt,CPU有6个中断引脚,加上两个软件中断(最高两个)

Exc code:异常类型,所有的外设中断为0,系统调用为8,...

3.EPC

对一般的异常,EPC包含：

. 导致异常的指令地址(virtual)

or. if 异常在delay slot指令发生,该指令前面那个跳转指令的地址

当EXL=1时,处理器不写EPC

4.和存储相关的：

context,BadVaddr,Xcontext,ECC,CacheErr,ErrorEPC

以后再说

一般异常处理程序都是先保存一些寄存器,然后清除EXL以便嵌套异常，

清除KSU保持核心态,IE位看情况而定;处理完后恢复一些保存内容以及CPU状态

=====

=====

=====

MIPS 指令集(共 31条)									
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	00000 0	rs	rt	rd	00000	10000 0	add \$1,\$2, \$3	\$1=\$ 2+\$3	rd <- rs + rt ; 其中rs =\$2, rt=\$3, rd=\$1
addu	00000 0	rs	rt	rd	00000	10000 1	addu \$1,\$2, \$3	\$1=\$ 2+\$3	rd <- rs + rt ; 其中rs =\$2, rt=\$3, rd=\$1, 无符号 数
sub	00000 0	rs	rt	rd	00000	10001 0	sub \$1,\$2, \$3	\$1=\$ 2-\$3	rd <- rs - rt ; 其中rs =\$2,

									rt=\$3, rd=\$1
subu	00000 0	rs	rt	rd	00000	10001 1	subu \$1,\$2, \$3	\$1=\$ 2-\$3	rd <= rs - rt ; 其中rs =\$2, rt=\$3, rd=\$1, 无符号 数
and	00000 0	rs	rt	rd	00000	10010 0	and \$1,\$2, \$3	\$1=\$ 2 & \$3	rd <= rs & rt ; 其中rs =\$2, rt=\$3, rd=\$1
or	00000 0	rs	rt	rd	00000	10010 1	or \$1,\$2, \$3	\$1=\$ 2 \$3	rd <= rs rt ; 其中rs =\$2, rt=\$3, rd=\$1
xor	00000 0	rs	rt	rd	00000	10011 0	xor \$1,\$2, \$3	\$1=\$ 2 ^ \$3	rd <= rs xor rt ; 其中rs =\$2, rt=\$3, rd=\$1 (异或)
nor	00000 0	rs	rt	rd	00000	10011 1	nor \$1,\$2, \$3	\$1=~ (\$2 \$ 3)	rd <= not(rs rt) ; 其中rs =\$2, rt=\$3,

									rd=\$1 (或非)
slt	00000 0	rs	rt	rd	00000	10101 0	slt \$1,\$2, \$3	if(\$2< \$3) \$1=1 else \$1= 0	if (rs < rt) rd=1 e lse rd =0 ; 其中rs =\$2, rt=\$3, rd=\$1
sltu	00000 0	rs	rt	rd	00000	10101 1	sltu \$1,\$2, \$3	if(\$2< \$3) \$1=1 else \$1= 0	if (rs < rt) rd=1 e lse rd =0 ; 其中rs =\$2, rt=\$3, rd=\$1 (无符 号数)
sll	00000 0	00000	rt	rd	shamt	00000 0	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	00000 0	00000	rt	rd	shamt	00001 0	srl \$1,\$2, 10	\$1=\$ 2>>10	rd <- rt >> shamt ; (logica

									l) , 其中 rt=\$2, rd=\$1
sra	00000 0	00000	rt	rd	shamt	00001 1	sra \$1,\$2, 10	\$1=\$ 2>>10	rd <- rt >> shamt ; (arith metic) 注意符 号位保 留 其中 rt=\$2, rd=\$1
sllv	00000 0	rs	rt	rd	00000	00010 0	sllv \$1,\$2, \$3	\$1=\$ 2<<\$3	rd <- rt << rs ; 其中rs =\$3, rt=\$2, rd=\$1
srlv	00000 0	rs	rt	rd	00000	00011 0	srlv \$1,\$2, \$3	\$1=\$ 2>>\$3	rd <- rt >> rs ; (logica l)其中 rs= \$3, rt=\$2, rd=\$1
srav	00000 0	rs	rt	rd	00000	00011 1	srav \$1,\$2, \$3	\$1=\$ 2>>\$3	rd <- rt >> rs ; (arith metic) 注意符 号位保 留 其中rs =\$3,

									rt=\$2, rd=\$1
jr	00000 0	rs	00000	00000	00000	00100 0	jr \$31	goto \$ 31	PC <- rs
I-type	op	rs	rt	immediate					
addi	00100 0	rs	rt	immediate			addi \$ 1,\$2,1 00	\$1=\$ 2+100	rt <- rs + (sign- extend)imme diate ; 其中 rt=\$1, rs=\$2
addiu	00100 1	rs	rt	immediate			addiu \$1,\$2, 100	\$1=\$ 2+100	rt <- rs + (zero- extend)imme diate ; 其中 rt=\$1, rs=\$2
andi	00110 0	rs	rt	immediate			andi \$ 1,\$2,1 0	\$1=\$ 2 & 10	rt <- rs & (zero- extend)imme diate ; 其中 rt=\$1, rs=\$2
ori	00110 1	rs	rt	immediate			andi \$ 1,\$2,1 0	\$1=\$ 2 10	rt <- rs (zero- extend)imme diate ; 其中

							rt=\$1, rs=\$2
xori	00111 0	rs	rt	immediate	andi \$ 1,\$2,1 0	\$1=\$ 2 ^ 10	rt <- rs xor (zero- extend)imme diate ; 其中 rt=\$1, rs=\$2
lui	00111 1	00000	rt	immediate	lui \$1, 100	\$1=1 00*65 536	rt <- immed iate*6 5536 ; 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	10001 1	rs	rt	immediate	lw \$1, 10(\$2)	\$1=m emory [\$2 +10]	rt <- memo ry[rs + (sign- extend)imme diate] ; rt=\$1, rs=\$2
sw	10101 1	rs	rt	immediate	sw \$1 ,10(\$2)	memo ry[\$2+ 10] =\$1	memo ry[rs + (sign-

							extend)immediate] ← rt ; rt=\$1, rs=\$2
beq	00010 0	rs	rt	immediate	beq \$ 1,\$2,1 0	if(\$1= =\$2) goto PC+4+ (sign-extend))immediate< <2	if (rs == rt) PC ← PC+4 + (sign-extend))immediate< <2
bne	00010 1	rs	rt	immediate	bne \$ 1,\$2,1 0	if(\$1! =\$2) goto PC+4+ (sign-extend))immediate< <2	if (rs != rt) PC ← PC+4 + (sign-extend))immediate< <2
slti	00101 0	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 ; 其中 rs= \$2, rt=\$1
sltiu	00101 1	rs	rt	immediate	sltiu \$1,\$2,	if(\$2< 10)	if (rs <(zero

					10	\$1=1 else \$1=0	- extend)immediate) rt=1 else rt=0 ; 其中 rs=\$2, rt=\$1
J-type	op	address					
j	00001 0	address			j 100 00	goto 1 0000	PC <= (PC+4) [31..28],address,0,0 ; addresses=10000/4
jal	00001 1	address			jal 10 000	\$31<=PC+4; goto 1 0000	\$31<=PC+4 ; PC <= (PC+4) [31..28],address,0,0 ; addresses=10000/4

注意：因为MIPS16只有16个16位的寄存器，所以JAL指令中\$31改成\$15，所有立即数均无需扩展，LUI指令直接就是将立即数付给RT寄存器。

yha)
关注 - 0
粉丝 - 5
+加关注

0

推荐

0

反对

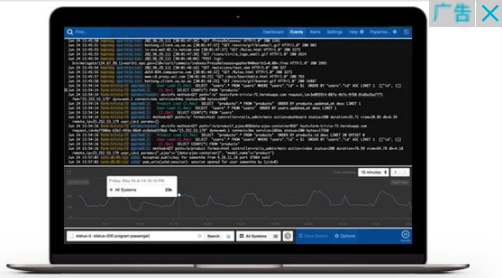
« 上一篇： C语言中内存分配
» 下一篇： openWRT自学---初始化过程和主要脚本的分析--转

posted @ 2017-10-19 16:41 yha) 阅读(3750) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部


注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)， [访问](#) 网站首页。


- 【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】开发者上云福利，腾讯云1核4G云服务器11元/月起
- 【推荐】开年盛典，百度智能云1核1G云服务器84元/年
- 【推荐】12知识点+20干货案例+110面试题，助你拿offer！| Python面试宝典
- 【推荐】《Flutter in action》开放下载！闲鱼Flutter企业级实践精选



Cloud-Based Log Management

Thousands of users trust Papertrail™ to manage their log messages.





- 相关博文：
- MIPS寄存器介绍
 - MIPS汇编指令集
 - MIPS指令学习二
 - 自己动手写CPU之第五阶段（3）——MIPS指令集中的逻辑、移位与空指令
 - MIPS32的内部寄存器。
- » 更多推荐...

精品问答：微服务架构 Spring 核心知识 50 问

最新 **IT** 新闻:

- 切断传染源有多重要? 拆下一个水泵把手便可阻止一场大瘟疫
 - 国内首次: 农行企业网银兼容国产麒麟操作系统
 - 小米 10 Pro 评测: 1 亿像素拍照纤毫毕现, 完全没有短板的旗舰机
 - 蚂蚁金服AAAI收录论文曝光
 - 研究机构: 苹果去年半导体采购支出361亿美元 取代三星成第一大买家
- » 更多新闻...

历史上的今天:

2017-10-19 C语言中内存分配

Copyright © 2020 yha)

Powered by .NET Core 3.1.1 on Linux