

看懂PowerPC汇编之指令集架构

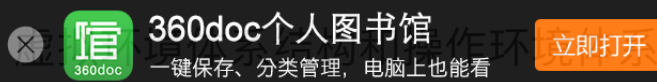
2011-08-12  360doc个人图书馆

[举报](#)

看懂PowerPC汇编，需要如下3方面的知识：

- 1.PowerPC指令集架构即Power ISA，可以从Power.org获得，包括寄存器定义，数据模型，寻址方式和指令定义以及指令助记符；
- 2.PowerPC ABI即应用程序二进制接口，即寄存器的使用规范和栈调用结构；
- 3.PowerPC Pseudo-ops，即.text, .align n等汇编语言中常用的伪操作符。

PowerPC ISA分为3个级别即“Book”，分别对应于用户指令集体系结构，



结构。其中Book III分化出了服务器版本Book III-S（经典PowerPC架构）和嵌入式版本Book III-E（专门为嵌入式优化的版本）。

1.寄存器定义：

PowerPC处理器寄存器分为2大类-专用寄存器和非专用寄存器。其中，非专用寄存器包括32个通用目的寄存器（GPR），32个浮点寄存器（FPR），条件寄存器（CR），浮点状态和控制寄存器（FPSCR）；专用寄存器主要包括连接寄存器（LR），计数寄存器（CTR），机器状态寄存器（MSR）以及时间基准寄存器（TBL/TBU）等等。PPC4xx系列处理器还有DCR寄存器，需要用专门的指令访问。这里有两点需要注意：

1. PowerPC处理器可以运行于两个级别，即用户模式和特权模式。用户

模式下，仅有GPR，FPR，CR，FPSCR，LR，CTR，XER以及TBL/TBU可以访问。从Power ISA 2.05开始，DCR寄存器也可以在通过用户模式DCR访问指令进行访问。

2.PowerPC处理器没有专用的栈指针寄存器和PC指针寄存器，也就是说硬件不负责维护调用栈。

2.数据模型：

PowerPC 支持如下数据格式：byte, halfword, word, doubleword,quadword, 同时默认支持big-endian字节序，即MSB（最高有效字节，例如0x12345678中0x12即MSB）保存在低地址。little-endian字节序可以通过修改设置支持。

注意：PowerPC习惯，msb（最高有效位）为bit0，lsb为bit31.

3.寻址方式

PowerPC没有专门的IO操作指令，所有地址访问一视同仁，并且只支持地址和寄存器之间的访问。因此寻址方式非常简单，可以概括为2类6种：

3.1 Load/store/算术/逻辑/cache指令：

- a) 寄存器间接寻址模式，通常写作RA或者RB；
- b) 寄存器间接立即数索引寻址模式，即（基址寄存器+立即数偏移）寻址模式，通常写作d(RA)；
- c) 寄存器间接索引寻址模式（基址寄存器 + 偏移寄存器）寻址模式，通常写作RA，RB。

注意：对于三种模式，若寄存器为GPR0，则其内容被忽略，并以0代替其内容。

3.2 跳转指令：

- a) 立即数寻址模式；
- b) 链接寄存器（LR）模式，即目的地址被保存在LR中；
- c) 计数寄存器（CTR）模式，即目的地址保存在CTR中。

注意：实际上还有一些特殊的跳转指令rfi/rfci/rfmc, 其目的地址保存在SRR0/CSRR0/MCSRR0中。

4.指令定义和指令助记符：

PowerPC指令的长度都是4字节，但是种类繁多，而且有些指令极其复杂。因此，通常情况下，PowerPC汇编编程中采用指令和助记符混用的方式。助记符主要用来简化内存访问、算术运算、逻辑运算等常用指令，例如用bne target代替bc4,2, target表示不为零则跳转。下面仅以一些常见的汇编代码片段来做一些简单的归纳，具体信息请参考相应的处理器core用户手册或者PowerISA。注意，cache和MMU指令会跟其实现一起介绍，此处不再赘述。

4.1 读取一个word(0x12345678)到目的寄存器

```
lis RA, 0x12345678@h /* 高16位(0x1234)偏移16位后变成0x12340000  
放进RA */
```

```
ori RB, RA, 0x12345678@l /* RA与低16位(0x5678)相或后构成完整数据  
放进RA */
```

注意：PowerPC指令中，i后缀表示立即数，s后缀表示左移16位。例如addi、addis、ori、oris等。这段代码也可以用来读取某个变量的值，只需要把立即数替换成变量名。

2.从某个地址(0x56789abc)读取数据

```
lis RA, 0x56789abc@ha /* 调整后的高16位 (0x5679) 偏移16位后变成 (0x56790000) 放进RA */
```

```
lwz RB, 0x56789abc@l (RA) /* RA加上低16位(0xffff9abc) 构成完整数据地址，然后将其内容放进RB */
```

注意：

@l,@h和@ha：用于算术运算的操作数（包括addi的操作数）时，@l获取的是符号扩展的低16位数据(0xffff9abc)，因此高16位必须进行根据bit15进行调整，而不能简单的使用@h来获取。

load/store指令的通常格式为(l/st)(w/h/b)(z/br)(u/x) RA, ... 其中l代表读取数据到RA中；st代表将RA的内容写入内存；w/h/b分别代表针对word、halfword/byte进行操作；z仅用于读取数据，代表字或者半字读到寄存器中后将其高位清零（注意：对word的读取无意义，但必须加）；br代表读取后或者存入前对数据进行字节序反转；u代表从某个内存地址读取数据后，同时更新保存内存地址的寄存器；x代表使用寄存器间接索引寻址模式（基址寄存器 + 偏移寄存器）寻址模式(默认使用寄存器间接立即数索引寻址模式)

这段代码也可以用来访问数组或者结构中的某个成员，或者指针指向的地址块中的数据，只需把立即数换成相应的数组名、结构名或者指针即

可。

3.入栈/出栈操作

入栈：

```
stwu 1, -16(1) /* 原始栈指针GPR1保存在新栈顶（原始栈指针减去16）  
*/
```

```
mflr 0 /* LR暂存在GPR1 */
```

```
stw 0, 20(1) /* 保存LR到调用者的栈中 */
```

出栈：

```
lwz 0, 20(1) /* LR出栈 */
```

```
mtlr 0 /* 恢复LR */
```

```
addi r1, r1, 16 /* 销毁栈 */
```

注意：栈定义会在ABI部分详细解释，此处不再赘述。

4.循环

```
li 0, 10 /* 循环次数暂存入GPR0 */
```

```
mtctr 0 /* 更新计数寄存器 CTR*/
```

label:

```
... /* 需要重复执行的指令 */
```

```
bdnz label /* 递减计数器，不等于0则跳转到标号处重复执行 */
```

注意：

label类似于C语言中的标号，因此在同一段代码中，不能重复。因此，习惯中使用数字标号来代替。数字标号作为跳转指令的目的地址时，通常在其后一个b表示向最近的低地址数字标号跳转，加f表示向最近的高地址数字标号跳转。例如beq 1f或者blt 2b。注意，标号，不管是数字形式还是字母形式，都可以用作一个代表标号所对应行的地址的变量使用。例如，可以用lis RA, 1f@ha; lwz RB, 1f@l(RA)可以用来获取标号1f处的指令内容。

跳转指令分为条件跳转和无条件跳转两种：

无条件跳转主要有b（跳转到相对地址）\ba（跳转到绝对地址）\blr（跳转到连接寄存器）\bctr（跳转到计数寄存器）4种；（注意,rfi、rfci、rfmci等异常返回指令也可以看做是特殊的无条件跳转。）

条件跳转指令分为两种：需要配合比较指令/算术指令/逻辑运算指令使用的beq（相等或者为0则跳转）/bne（不等或者非0则跳转）/blt（小于则跳转）/bgt（大于则跳转）/ble（小于等于则跳转）/bge（大于等于则跳转）/bni（不小于则跳转）/bng（不小于则跳转）等和需要配合计数寄存器CTR使用的bdz（CTR递减到0则跳转）/bdnz（CTR没有递减到0则跳转）等。

此外条件跳转指令还可以增加后缀a表示跳转到绝对地址；所有跳转指令增加后缀l表示同时更新连接寄存器（LR），用于子程序调用。

配合条件跳转指令使用的比较指令主要是cmp(l)w(i)，其中l表示逻辑比较既无符号数比较，w表示比较2个word，i表示寄存器内容跟立即数进行比较；

配合条件跳转指令使用的算术指令必须加上后缀“.”用以表示更新条件寄存器CR，主要有add（寄存器内容相加）/addi（寄存器内容跟立即数相加）/addis（立即数左移16位后跟寄存器内容相加）和subi（寄存器内容减去立即数）/subis（寄存器内容减去左移16位后的立即数）/subf（从RB（第三个参数）中减去RA（第二个参数）的内容放入RT（第一个参数））；

配合条件跳转指令使用的逻辑指令也必须加上后缀“.”用以表示更新条件寄存器CR，主要包括and(i)(s)/or(i)(s)。

CTR等专用寄存器必须通过专用命令mfspr rD,SPR和mtspr SPR,rS或者mfctr/mtctr/mflr/mtlr/mfspr/mtspr/mftbl/mftbu/mttbl/mttbu助记符进行操作。此外，DCR寄存器必须通过mfdcr/mtdc/mfdcrx/mtdcx/mfdcrux/mtdcux等进行操作。注意，后面4条DCR指令仅在Power ISA 2.05及更高版本支持。

5.强大的位操作指令

PowerPC有3条强大的位操作指令，几乎能实现你能想象的所有位操作。

a)rlwinm(.) rA,rS,SH,MB,ME 寄存器RS的内容循环左移立即数SH位，然后跟立即数MB和ME形成的MASK相与后放进RA

b)rlwnm(.) rA,rS,RB,MB,ME 类似于上一条指令，只是把左移的位数放到了寄存器RB中

c)rlwimi(.) rA,rS,SH,MB,ME 寄存器RS的内容循环左移立即数SH位，然后跟立即数MB和ME形成的MASK相与，再把RA的内容跟立即数MB和

ME形成的MASK的补码相与，即清掉RA中MASK对应的位，最后把处理后的RS和RA的内容相或，放入RA中

注意：MASK形成的规则是，如果MB小于等于ME，则MB到ME之间的位全部置1，包括这两位，形成MASK;否则，MB到ME之间的位清0，其他位包括这两位置1，形成MASK。

下面给出几个例子：

a)从立即数0x12345678（RS）中抽取bit 20-23，并左移16位，从而得到0x06000000.

```
rlwinm rA, rS, 16, 4, 7
```

具体过程如下：0x12345678循环左移16位得到0x56781234，然后与MASK0x0f00 0000 (MASK[4,7])相与。

* 该指令可以用来抽取C语言代码或者寄存器中的位域。

b)清除立即数0x12345678（RS）的bit 28 - 31，并右移24位，从而得到0x0012 3456

```
rlwinm rA, rS, 24, 8, 31
```

具体过程如下：0x12345678循环左移24位得到0x78123456，然后与MASK0x00ff ffff (MASK[8,31])相与。

* 该指令可以进行除数或者乘数为2的倍数的乘法/除法操作。

c)清除立即数0x12345678（RS）的bit 6，从而得到0x10345678

```
rlwinm rA, rS, 0, 7, 5
```


具体过程如下：0x12345678循环左移0位，仍是0x56781234，然后与MASK0xfdf fff (MASK[7,5])相与。

* 该指令可以用来清除C语言代码或者寄存器中的位域。

d)抽取0x87654321(RS)的bit 24-31，用以对立即数0x12345678 (RA)的bit 8-15进行先清除后置位的操作，从而得到0x12215678.

rlwimi rA, rS, 16, 8, 15

具体过程如下：0x87654321(RS)循环左移16位得到0x43218765，然后与 MASK0x00ff 0000(MASK[8,15]) 相 与 得 到 0x0021 0000； 再 把 0x12345678 (RA) 与 MASK0x00ff 0000(MASK[8,15])的补码0xff00 ffff相与，得到0x1200 5678; 最后0x0021 0000跟0x12005678相与，得到0x12215678。


* 该指令可以用来清除C语言代码或者寄存器中的某个位域，然后对该位域进行赋值的操作。

广告 X

十个吓人的口臭原因

口臭必看

诱发口臭的因素打死都想不到!



gukc.lvchengky.com

更多精彩，请关注【360doc个人图书馆】

1. 点击  **【360doc个人图书馆】**，快速关注
2. 或：搜索微信号 **【360doc】**，进行关注