寄存器：

在mipsy实现的MIPS架构中，MIPS拥有32个32位的通用寄存器以及两个特殊寄存器Hi和Lo，用于操作64位整数。(在MIPS架构中，Hi和Lo是两个特殊的寄存器，用于存放乘法和除法操作的结果。这两个寄存器不同于32个通用目的寄存器，它们专门设计用来处理64位整数运算的结果，因为32位寄存器无法容纳超过32位的结果。Hi寄存器：用来存储乘法结果的前32位数字，或者是除法操作的余数。Lo寄存器：用来存储乘法结果的后32位数字，或者是除法操作的商。)

这32个通用寄存器可以通过$0到$31或者符号名称来引用，并按照以下方式使用：

| **寄存器** | **名称** | **描述** |
| --- | --- | --- |
| $0 | $zero | 值0；写入操作会被忽略 |
| $1 | $at | 汇编器暂存器；保留供汇编器使用 |
| $2 $3 | $v0 $v1 | 表达式求值的结果或函数返回值 |
| $4~$7 | $a0~$a3 | 函数/子程序的前四个参数 |
| $8~$15 | $t0~$t7 | 临时的；调用者依赖它们的值必须在调用子程序之前保存，因为它们可能会被覆盖 |
| $16~$23 | $s0~$s7 | 已保存的；子程序必须保证它们的值不变，比如通过恢复它们的值 |
| $24 $25 | $t8 $t9 | 临时的；调用者依赖它们的值必须在调用子程序之前保存，因为它们可能会被覆盖 |
| $26 $27 | $k0 $k1 | 内核使用；可能会意外改变——避免在用户程序中使用 |
| $28 | $gp | 全局指针（全局区域的地址） |
| $29 | $sp | 堆栈指针（堆栈顶部） |
| $30 | $fp | 帧指针（当前堆栈帧的底部）；如果不使用帧指针，变成保存寄存器 |
| $31 | $ra | 最近一次调用者的返回地址 |

内存：

图示

低可信度描述已自动生成

| **段名** | **基址** | **描述** |
| --- | --- | --- |
| text | 0x00400000 | 用户程序代码所在位置。在Mipsy中，这是内存中唯一可以执行指令的区域，其初始大小为256 KiB。这是唯一一个可以执行指令的内存区域。在Mipsy中，这部分内存也可写。在真实系统中，这部分内存通常是只读的。 |
| data | 0x10000000 | 用户数据所在位置；其初始大小为256 KiB，但大小不固定，可通过 sbrk 系统调用最多增加到1 MiB。这部分内存不可执行。 |
| stack | 0x7ffffffc | 函数调用堆栈；向负地址方向增长。其初始大小为64 KiB，但会根据需要增长到最大256 KiB。这部分内存不可执行。 |
| k\_text | 0x80000000 | 受保护的可执行代码，用户模式下不可访问；在真实系统中，操作系统的内核代码会映射到这里。在Mipsy中，入口点代码被加载在这里；其初始大小为64 KiB。 |
| k\_data | 0x90000000 | 受保护的数据，用户模式下不可访问；在真实系统中，操作系统的数据会映射到这里。在Mipsy中，入口点数据被加载在这里；其初始大小为64 KiB，但会根据需要增长到最多1 MiB。 |

语法：

每条指令写在一行上，有如下的通用格式：



每条指令的操作数数量可能不同，但可以是零到三个。在下面的描述中，使用以下符号来描述指令操作数。

| **操作** | **描述** |
| --- | --- |
| Rn | 一个寄存器 — 通常，Rs 和 Rt 作为源操作数，Rd 作为目的操作数；寄存器可以通过数字名（$0 至 $31）或符号名（$sN，$tN 等）指定。 |
| Imm | 一个字面常量值，或“立即数”；可以是八进制、十进制、十六进制或字符字面量；如果后面跟着一个数字（比如 **Imm16**），那么该数字指定了位宽，并暗示了值的范围。 |
| Label | 一个与内存地址相关联的符号名。 |
| Addr | 一个内存地址，格式如下所述。 |

许多指令有一个地址操作数；这些可以用多种格式编写：

| **格式** | **地址** |
| --- | --- |
| Label | 一个与内存地址相关联的符号名。 |
| (Rn) | 存储在寄存器 Rn 中的值（间接地址）。 |
| Imm(Rn) | 立即数 Imm 与存储在寄存器 Rn 中的值之和。 |
| Label(Rn) | Label 的地址和存储在寄存器 Rn 中的值之和。适用于访问堆栈。 |
| Label + Imm | Label 的地址和立即数 Imm 之和。适用于访问结构体。 |
| Label + Imm(Rn) | Label 的地址、立即数 Imm 以及存储在寄存器 Rn 中的值之和。适用于访问结构阵列 |

指令：

Mipsy模拟器实现了MIPS32指令集中的指令，以及伪指令（看起来像MIPS指令，但实际上并没有在真实硬件上提供）。真实的MIPS指令会用一个勾选符号（✓）标记。所有其他指令都是伪指令。在表达式中的操作符具有与它们在C语言中相同的含义。

CPU运算指令：

|  |  |  |  |
| --- | --- | --- | --- |
| ✓add | add | add R1, R2, R3 | 将R2+R3的值赋予R1 |
| ✓addi | add immediately | addi R1, R2, Imm16 | 将R2与16位宽的数字相加，结果赋予R1 |
| ✓addu | add unsigned | addu R1, R2, R3 | add的无符号数版本，其他一样 |
| ✓addiu | add unsigned immediately | addiu R1, R2, Imm16 | addi的无符号数版本，其他一样 |
| ✓sub | subtract | sub R1, R2, R3 | 将R2-R3的值赋予R1 |
| ✓subu | subtract unsigned | subu R1, R2, R3 | sub的无符号数版本，其他一样 |
| ✓mul | multiply | mul R1, R2, R3 | 将R2×R3的值赋予R1 |
| ✓mult | multiply | mult R1, R2 | 将R1×R2的结果中低32位存储在Lo  寄存器中，高32位存储在Hi寄存器中 |
| ✓multu | multiply unsigned | multu R1, R2 | mult的无符号数版本，其他一样 |
| ✓madd | multiply and add | madd R1, R2 | 将R2×R3的值加给hi, lo组成的数字，  并记录到hi, lo中 |
| ✓maddu | multiply and add unsigned | maddu R1, R2 | madd的无符号数版本，其他一样 |
| ✓msub | multiply and subtract | msub R1, R2 | hi, lo组成的数字减R2×R3的值，  并记录到hi, lo中 |
| ✓msubu | multiply and subtract unsigned | msubu R1, R2 | msub的无符号数版本，其他一样 |
| div | device | div R1, R2, R3 | 将R2÷R3的值赋予R1 |
| ✓div | device | div R1, R2 | 将R1÷R2的值赋予lo，将R1÷R2的余数  赋予hi |
| divu | device unsigned | divu R1, R2, R3 | div的无符号数版本，其他一样 |
| ✓divu | device unsigned | divu R1, R2 | div的无符号数版本，其他一样 |
| rem | remainder | rem R1, R2, R3 | 将R2÷R3的余数赋予R1 |
| remu | remainder unsigned | remu R1, R2, R3 | rem的无符号数版本，其他一样 |
| ✓clo | count leading ones | clo R1, R2 | 数R2二进制最高位开始的连续1的数量，  赋予R1 |
| ✓clz | count leading zeros | clz R1, R2 | 数R2二进制最高位开始的连续0的数量，  赋予R1 |
| ✓seb | sign extend byte | seb R1, R2 | 将8位宽的R2符号扩展到32位宽，  赋予R1 |
| ✓seh | sign extend halfword | seh R1, R2 | 将16位宽的R2符号扩展到32位宽，  赋予R1 |
| seq | set if equal | seq R1, R2, R3 | 如果R2==R3，将R2值赋予R1，  反之则无动作 |
| sne | set if not equal | sne R1, R2, R3 | 如果R2!=R3，将R2值赋予R1，  反之则无动作 |
| sle | set if less or equal | sle R1, R2, R3 | 如果R2<=R3，将R2值赋予R1，  反之则无动作 |
| sleu | set if less or equal unsigned | sleu R1, R2, R3 | sle的无符号数版本，其他一样 |
| ✓slt | set if less than | slt R1, R2, R3 | 如果R2<R3，将R2值赋予R1，  反之则无动作 |
| ✓sltu | set if less than unsigned | sltu R1, R2, R3 | slt的无符号数版本，其他一样 |
| ✓slti | set if less than immediately | slti R1, R2, Imm16 | 如果R2<Imm16，将R2值赋予R1，  反之则无动作 |
| ✓sltiu | set if less than unsigned | sltiu R1, R2, Imm16 | slti的无符号数版本，其他一样 |
| sgt | set if greater than | sgt R1, R2, R3 | 如果R2>R3，将R2值赋予R1，  反之则无动作 |
| sgtu | set if greater than unsigned | sgtu R1, R2, R3 | sgt的无符号数版本，其他一样 |
| sge | set if greater or equal | sge R1, R2, R3 | 如果R2>=R3，将R2值赋予R1，  反之则无动作 |
| sgeu | set if greater or equal unsigned | sgeu R1, R2, R3 | sge的无符号数版本，其他一样 |
| abs | absolute value | abs R1, R2 | 取R2的绝对值，赋予R1 |
| neg | negate | neg R1, R2 | 取R2的相反数，赋予R1 |
| negu | negate unsigned | negu R1, R2 | neg的无符号数版本，其他一样 |

CPU逻辑指令：

|  |  |  |  |
| --- | --- | --- | --- |
| ✓and | and | and R1, R2, R3 | 将R2&(按位与)R3的值赋予R1 |
| ✓andi | and immediately | andi R1, R2, Imm16 | 将R2&(按位与)Imm16的值赋予R1 |
| ✓or | or | or R1, R2, R3 | 将R2|(按位或)R3的值赋予R1 |
| ✓ori | or immediately | ori R1, R2, Imm16 | 将R2|(按位或)Imm16的值赋予R1 |
| ✓nor | not or | nor R1, R2, R3 | 将~(R2|R3)的值赋予R1 |
| ✓xor | exclusive or | xor R1, R2, R3 | 将R2^(按位异或)R3的值赋予R1 |
| ✓xori | exclusive or immediately | xori R1, R2, Imm16 | 将R2^(按位异或)Imm16的值赋予R1 |
| not | not | not R1, R2 | 将~R2的值赋予R1 |

CPU位移命令：

|  |  |  |  |
| --- | --- | --- | --- |
| rol | rotate left | rol R1, R2, R3 | 将R2的二进制数向左旋转R3位，将值赋予R1 |
| ror | rotate righy | ror R1, R2, R3 | 将R2的二进制数向右旋转R3位，将值赋予R1 |
| ✓rotr | rotate word right | rotr R1, R2, Imm16 | 将R2的二进制数向右旋转Imm16位，将值赋予R1 |
| ✓rotrv | rotate word right variable | rotrv R1, R2, R3 | 将R2的二进制数向右旋转R3位，将值赋予R1 |
| ✓sll | shift word left logical | sll R1, R2, Imm16 | 将R2的二进制数逻辑左移Imm16位，将值赋予R1 |
| ✓sllv | shift word left logical variable | sllv R1, R2, R3 | 将R2的二进制数逻辑左移R3位，将值赋予R1 |
| ✓sra | shift word right arithmetic | sra R1, R2, Imm16 | 将R2的二进制数算术右移Imm16位，将值赋予R1 |
| ✓srav | shift word right arithmetic variable | srav R1, R2, R3 | 将R2的二进制数算数右移R3位，将值赋予R1 |
| ✓srl | shift word right logical | srl R1, R2, Imm16 | 将R2的二进制数逻辑右移Imm16位，将值赋予R1 |
| ✓srlv | shift word right logical variable | srlv R1, R2, R3 | 将R2的二进制数逻辑右移R3位，将值赋予R1 |

CPU读取存储及内存操作指令：

|  |  |  |  |
| --- | --- | --- | --- |
| li | load immidiately | li R1 Imm | 将Imm的数值赋予R1 |
| la | load address | la R1 label | 将标签label赋予R1 |
| ✓lui | load upper immidiately | lui R1, Imm16 | 将Imm16<<16的值赋予R1,即设置R1  的高16位 |
| ✓lb | load byte | lb R1, offset16(R2(内存地址)) | 将内存R2(内存地址)+offset16处的8bit  的值符号扩展到32bit，赋予R1 |
| ✓lbu | load byte unsigned | lbu R1, offset16(R2(内存地址)) | lb的无符号数版本，其他一样 |
| ✓lh | load halfword | lh R1, offset16(R2(内存地址)) | 将内存R2(内存地址)+offset16处的16bit  的值符号扩展到32bit，赋予R1 |
| ✓lhu | load halfword unsigned | lhu R1, offset16(R2(内存地址)) | lh的无符号数版本，其他一样 |
| ✓lw | load word | lw R1, offset16(R2(内存地址)) | 将内存R2(内存地址)+offset16处的32bit  的值赋予R1 |
| ✓sb | save byte | sb R1, offset16(R2(内存地址)) | 将R1的最低的8bit存储到内存R2(内存  地址)+offset16处 |
| ✓sh | save halfword | sb R1, offset16(R2(内存地址)) | 将R1的最低的16bit存储到内存R2(内存  地址)+offset16处 |
| ✓sw | save word | sw R1, offset16(R2(内存地址)) | 将R1存储到内存R2(内存地址)+offset16处 |
| push | push register to stack | push R1 | 将栈地址下移32bits,将R1存储到当前位置 |
| pop | pop register from stack | pop R1 | 将栈地址上移32bits，将当前位置的值  赋予R1 |
| begin | create stack frame | begin |  |
| end | destroy stack frame | end |  |

CPU移动指令：

|  |  |  |  |
| --- | --- | --- | --- |
| ✓mfhi | move form hi register | mfhi R1 | 将hi寄存器的值赋予R1 |
| ✓mflo | move form lo register | mflo R1 | 将lo寄存器的值赋予R1 |
| ✓mthi | move to hi register | mthi R1 | 将R1的值赋予hi寄存器 |
| ✓mtlo | move to lo register | mtlo R1 | 将R1的值赋予lo寄存器 |
| ✓movz | move conditional on zero | movz R1, R2, R3 | 如果R3==0，将R2的值赋予R1 |
| ✓movn | move conditional on not zero | movn R1, R2, R3 | 如果R3！=0，将R2的值赋予R1 |
| move | move | move R1, R2 | 将R2的值赋予R1 |

CPU分支及跳转指令：

|  |  |  |  |
| --- | --- | --- | --- |
| b | unconditional branch | b label/offset16 | 使用相对地址进行跳转，  跳转至pc+offset16\*4的位置 |
| ✓beq | branch on equal | beq R1, R2, label/offset16 | 如果R1==R2，跳转到label |
| beq | branch on equal | beq R1, Imm, label/offset16 | 如果R1==Imm，跳转到label |
| beqz | branch on equal to zero | beqz R1, label/offset16 | 如果R1==0，跳转到label |
| ✓bne | branch on not equal | bne R1, R2, label/offset16 | 如果R1!=R2，跳转到label |
| bne | branch on not equal | bne R1, Imm, label/offset16 | 如果R1!=Imm，跳转到label |
| bnez | branch on not equal to zero | bnez R1, label/offset16 | 如果R1!=0，跳转到label |
| bge | branch on greater or equal | bge R1, R2, label/offset16 | 如果R1>=R2，跳转到label |
| bge | branch on greater or equal | bge R1, Imm, label/offset16 | 如果R1>=Imm，跳转到label |
| bgeu | branch on greater or equal unsigned | bgeu R1, R2, label/offset16 | bge的无符号数版本，其他一样 |
| bgeu | branch on greater or equal unsigned | bgeu R1, Imm, label/offset16 | bge的无符号数版本，其他一样 |
| ✓bgez | branch on greater or equal to zero | bgez R1, label/offset16 | 如果R1>=0，跳转到label |
| bgt | branch on greater than | bgt R1, R2, label/offset16 | 如果R1>R2，跳转到label |
| bgt | branch on greater than | bgt R1, Imm, label/offset16 | 如果R1>Imm，跳转到label |
| bgtu | branch on greater than unsigned | bgtu R1, R2, label/offset16 | bgt的无符号数版本，其他一样 |
| bgtu | branch on greater than unsigned | bgtu R1, Imm, label/offset16 | bgt的无符号数版本，其他一样 |
| ✓bgtz | branch on greater than to zero | bgtz R1, label/offset16 | 如果R1>0，跳转到label |
| blt | branch on less than | blt R1, R2, label/offset16 | 如果R1<R2，跳转到label |
| blt | branch on less than | blt R1, Imm, label/offset16 | 如果R1<Imm，跳转到label |
| bltu | branch on less than unsigned | bltu R1, R2, label/offset16 | blt的无符号数版本，其他一样 |
| bltu | branch on less than unsigned | bltu R1, Imm, label/offset16 | blt的无符号数版本，其他一样 |
| ✓bltz | branch on less than to zero | bltz R1, label/offset16 | 如果R1<0，跳转到label |
| ble | branch on less or equal | ble R1, R2, label/offset16 | 如果R1<=R2，跳转到label |
| ble | branch on less or equal | ble R1, Imm, label/offset16 | 如果R1<=Imm，跳转到label |
| bleu | branch on less or equal unsigned | bleu R1, R2, label/offset16 | ble的无符号数版本，其他一样 |
| bleu | branch on less or equal unsigned | bleu R1, Imm, label/offset16 | ble的无符号数版本，其他一样 |
| ✓blez | branch on less or equal to zero | blez R1, label/offset16 | 如果R1<=0，跳转到label |
| ✓j | jump | j label/address26 | 跳转至label |
| ✓jal | jump and link | jal label/address26 | 跳转至label，并将当前  address+4bits保存到$ra |
| ✓jr | jump register | jr R1 | j的使用寄存器的版本,其他一样 |
| ✓jalr | jump and link register | jalr R1 | jal的使用寄存器的版本,其他一样 |
| ✓jalr | jump and link register | jalr R1, R2 | 将当前address+4bits保存  到R1中，跳转到R2 |

CPU陷阱指令：

|  |  |  |  |
| --- | --- | --- | --- |
| ✓syscall | system call | li $v0 Imm \n syscall | 呼叫系统来执行某些操作 |
| ✓break | break | break | 添加断点(用于Debug，调试等) |
| ✓teq | trap if equal | teq R1, R2 | 如果R1==R2，trap |
| ✓teqi | trap if equal immediately | teqi R1, Imm16 | 如果R1==Imm16，trap |
| ✓tne | trap if not equal | tne R1, R2 | 如果R1!=R2，trap |
| ✓tnei | trap if not equal immediately | tnei R1, Imm16 | 如果R1!=Imm16，trap |
| ✓tge | trap if greater or equal | tge R1, R2 | 如果R1>=R2，trap |
| ✓tgeu | trap if greater or equal unsigned | tgeu R1, R2 | tge的无符号数版本，其他一样 |
| ✓tgei | trap if greater or equal immediately | tgei R1, Imm16 | 如果R1>=Imm16，trap |
| ✓tgeiu | trap if greater or equal immediately unsigned | tgeiu R1, Imm16 | tgei的无符号数版本，其他一样 |
| tgt | trap if greater than | tgt R1, R2 | 如果R1>R2，trap |
| tgtu | trap if greater than unsigned | tgtu R1, R2 | tgt的无符号数版本，其他一样 |
| tgti | trap if greater than immediately | tgti R1, Imm16 | 如果R1>Imm16，trap |
| tgtiu | trap if greater than immediately unsigned | tgtiu R1, Imm16 | tgti的无符号数版本，其他一样 |
| ✓tlt | trap if less than | tlt R1, R2 | 如果R1<R2，trap |
| ✓tltu | trap if less than unsigned | tltu R1, R2 | tlt的无符号数版本，其他一样 |
| ✓tlti | trap if less than immediately | tlti R1, Imm16 | 如果R1<Imm16，trap |
| ✓tltiu | trap if less than immediately unsigned | tltiu R1, Imm16 | tlti的无符号数版本，其他一样 |
| tle | trap if less or equal | tle R1, R2 | 如果R1<=R2，trap |
| tleu | trap if less or equal unsigned | tleu R1, R2 | tle的无符号数版本，其他一样 |
| tlei | trap if less or equal immediately | tlei R1, Imm16 | 如果R1<=Imm16，trap |
| tleiu | trap if less or equal immediately unsigned | tleiu R1, Imm16 | tlei的无符号数版本，其他一样 |

CPU控制指令：

|  |  |  |  |
| --- | --- | --- | --- |
| nop | no operation | nop | 什么也不做 |

系统服务：

mipsy 模拟器提供了许多与宿主系统交互的机制，用于提供输入和输出、文件操作以及其他杂项服务，我们将其称为“系统调用”或“syscall”。这些通过在寄存器 $v0 中存储服务代码后，调用 syscall 指令来触发。

|  |  |
| --- | --- |
| 打印命令 | |
| $v0 | 对应操作 |
| 1 | 打印整数（print\_int）: $a0 中的整数被打印到控制台作为有符号十进制数。 |
| 2 | 打印浮点数（print\_float）: $f12 中的浮点数被打印到控制台，格式为 %.8f，这意味着打印时保留八位小数。 |
| 3 | 打印双精度浮点数（print\_double）: $f12 和 $f13 中的双精度浮点数被打印到控制台，格式为 %.18g，g 格式符表示根据数值的大小和精度，自动选择 %f 或 %e（科学计数法）格式。 |
| 4 | 打印字符串（print\_string）: $a0 引用的以 null 结尾的字节数组被打印到控制台作为 ASCII 字符串。 |
| 11 | 打印字符（print\_character）: $a0 中的字符被打印到控制台，类似于 C 语言中的 putchar 函数。 |
| 读取命令 | |
| 5 | 读取整数（read\_int）: 从控制台读取一个整数值，语义上类似于 C 语言的 atoi 函数，结果存储在 $v0 寄存器中。 |
| 6 | 读取浮点数（read\_float）: 从控制台读取一个浮点值，语义上类似于 C 语言的 atof 函数，结果存储在 $f0 寄存器中。 |
| 7 | 读取双精度浮点数（read\_double）: 从控制台读取一个双精度浮点值，语义上类似于 C 语言中的 atof 函数，结果存储在 $f0 和 $f1 寄存器中。 |
| 8 | 读取字符串（read\_string）: 将字符串读入提供的缓冲区（由 $a0 引用），直到读取的字节数达到 $a1 指定的大小为止，结果字符串以 null 字符结尾。 |
| 12 | 读取字符（read\_character）: 从控制台读取下一个字符，结果存储在 $v0 寄存器中，这个调用与 C 语言中的 getchar 函数相似。 |
| 文件操作命令 | |
| 13 | open 系统调用 $a0: 文件名（字符指针） $a1: 访问模式标志（整数） $a2: 文件模式（mode\_t 类型，通常用于设置新文件的权限） $v0: 返回值，文件描述符（一个小的非负整数） 功能：打开由 $a0 指定的文件，$a1 指定访问模式，如果文件需要创建，则 $a2 指定文件模式。成功时返回文件描述符，失败时返回负数。 |
| 14 | read 系统调用 $a0: 文件描述符（整数） $a1: 缓冲区指针 $a2: 要读取的字节数（整数） $v0: 返回值，读取的字节数 功能：从 $a0 指定的文件描述符对应的文件中读取 $a2 字节到 $a1 指向的缓冲区中。成功时返回读取的字节数，出错时返回 -1。 |
| 15 | write 系统调用 $a0: 文件描述符（整数） $a1: 缓冲区指针 $a2: 要写入的字节数（整数） $v0: 返回值，写入的字节数 功能：将 $a1 指向的缓冲区中的 $a2 字节写入到 $a0 指定的文件描述符对应的文件中。成功时返回写入的字节数，出错时返回 -1。 |
| 16 | close 系统调用 $a0: 文件描述符（整数） $v0: 返回值，成功标志 功能：关闭 $a0 指定的文件描述符对应的文件。成功时返回 0，出错时返回 -1。 |
| 进程服务命令 | |
| 9 | sbrk 系统调用 $a0: 整数，表示要增加的字节数。 功能：扩展数据段（.data 段）的大小，增加 $a0 指定的字节数。这个系统调用是一个底层的原语，经常用于实现像 malloc 这样的内存分配函数。 |
| 10 | 程序退出，返回码为 0。这是一个没有参数的系统调用，它会使程序终止执行并返回到操作系统。 |
| 17 | $a0: 整数，表示退出码。 功能：程序退出，返回 $a0 指定的退出码。这个系统调用允许程序指定退出时的状态码，这个状态码可以被程序的父进程读取，用来判断程序的执行是否成功。 |