前置

1. mips指令集中，有很多指令实际上是并不是真正懂的原始指令，而是类似于宏的存在，汇编器编译时，会把这类指令转换成(多条)原始指令

寄存器

1. $v0 返回值

2. $sp 栈顶指针寄存器

3. $fp ($s8) 栈帧基址寄存器

4. $ra 函数返回地址

> $0 永远为0
> $a0 -- $a3 参数寄存器
> $t0 -- $t9 临时寄存器
> $s0 -- $s7 (进入子函数)可能需要保存的寄存器

> 还有其他寄存器不介绍

汇编语言

变量声明

> 和arm类似
> varName: varType varValue
> varType有: .ascii, .asciiz, .byte, .half, .word, .quad, .space ....

```
     .text
.global  __start   #  是__start不是_start
__start:
     #  body

     .data
hello:
      .asciiz "hello world"
```

系统调用

- 调用号: $v0
- 参数: $a0 -- ??
- 系统调用指令: syscall
- 返回值放 $v0

函数

函数调用约定

1. 保存环境(如果需要的话)

> $t0 - $t9 由调用者保存

2. 设置参数

> $a0 -- $a3; 多余的放栈上

3. 跳转

> jal jalr
> bal

4. 进入子函数，创建栈帧，保存上下文
5. 返回值放 $v0

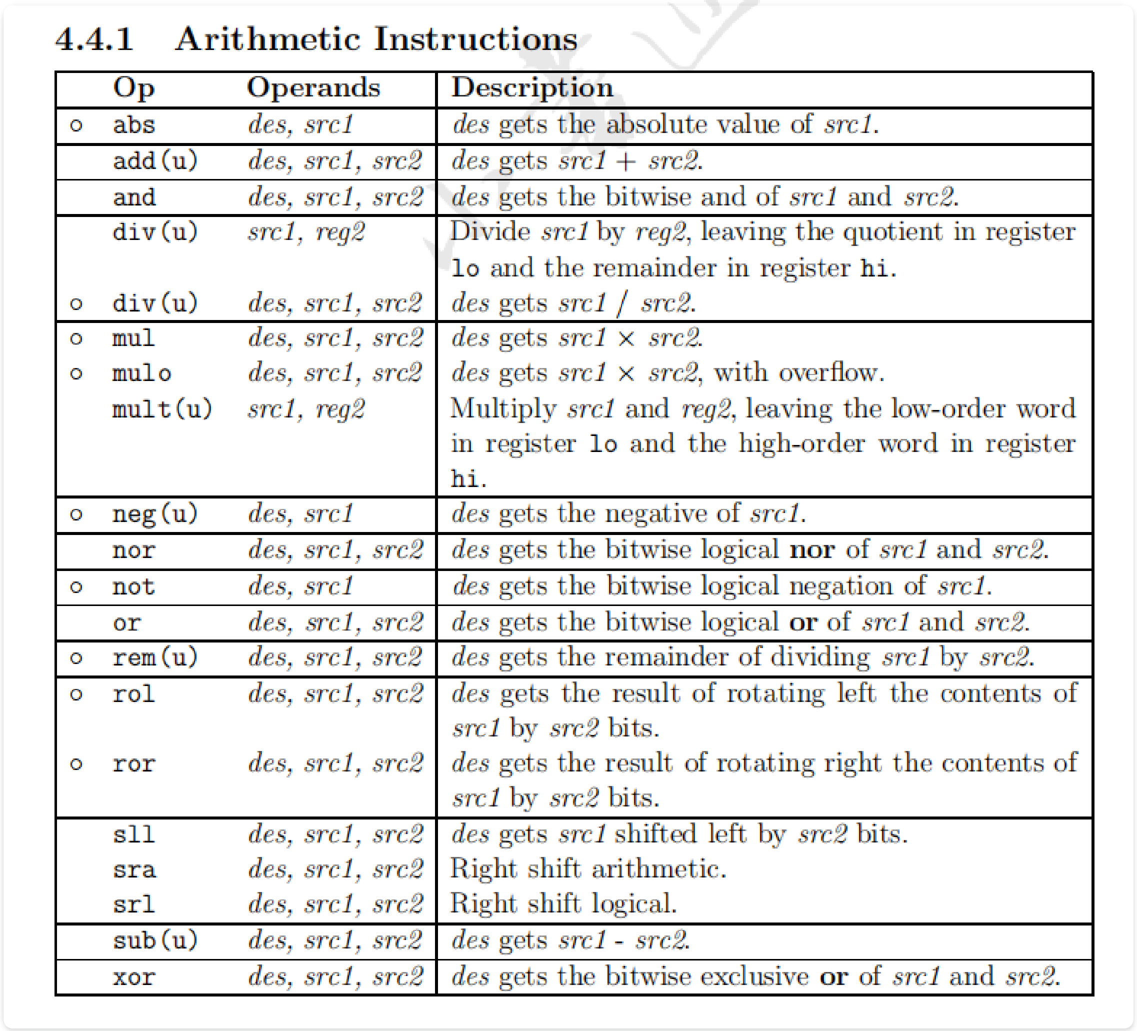
栈帧

> $fp, $sp 以及局部变量的位置和 x86 类似，但是有多一个 gp (暂时不知道是什么)
> 不需要保存 $ra 到栈的时候不会保存(类似于arm)

指令

- des must always be a register.
- src1 must always be a register.
- reg2 must always be a register.
- src2 may be either a register or a 32-bit integer
- addr must be an valid address.

算术指令

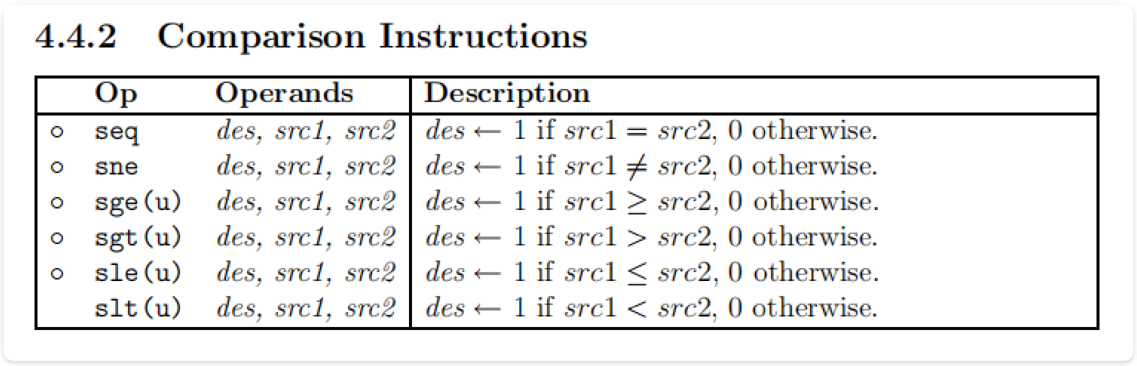


4.4.1 Arithmetic Instructions

	Op	Operands	Description
∘	`abs`	*des, src1*	*des* gets the absolute value of *src1*.
	`add(u)`	*des, src1, src2*	*des* gets *src1* + *src2*.
	`and`	*des, src1, src2*	*des* gets the bitwise and of *src1* and *src2*.
	`div(u)`	*src1, reg2*	Divide *src1* by *reg2*, leaving the quotient in register `lo` and the remainder in register `hi`.
∘	`div(u)`	*des, src1, src2*	*des* gets *src1* / *src2*.
∘	`mul`	*des, src1, src2*	*des* gets *src1* × *src2*.
∘	`mulo`	*des, src1, src2*	*des* gets *src1* × *src2*, with overflow.
	`mult(u)`	*src1, reg2*	Multiply *src1* and *reg2*, leaving the low-order word in register `lo` and the high-order word in register `hi`.
∘	`neg(u)`	*des, src1*	*des* gets the negative of *src1*.
	`nor`	*des, src1, src2*	*des* gets the bitwise logical **nor** of *src1* and *src2*.
∘	`not`	*des, src1*	*des* gets the bitwise logical negation of *src1*.
	`or`	*des, src1, src2*	*des* gets the bitwise logical **or** of *src1* and *src2*.
∘	`rem(u)`	*des, src1, src2*	*des* gets the remainder of dividing *src1* by *src2*.
∘	`rol`	*des, src1, src2*	*des* gets the result of rotating left the contents of *src1* by *src2* bits.
∘	`ror`	*des, src1, src2*	*des* gets the result of rotating right the contents of *src1* by *src2* bits.
	`sll`	*des, src1, src2*	*des* gets *src1* shifted left by *src2* bits.
	`sra`	*des, src1, src2*	Right shift arithmetic.
	`srl`	*des, src1, src2*	Right shift logical.
	`sub(u)`	*des, src1, src2*	*des* gets *src1* - *src2*.
	`xor`	*des, src1, src2*	*des* gets the bitwise exclusive **or** of *src1* and *src2*.

ins1

比较指令

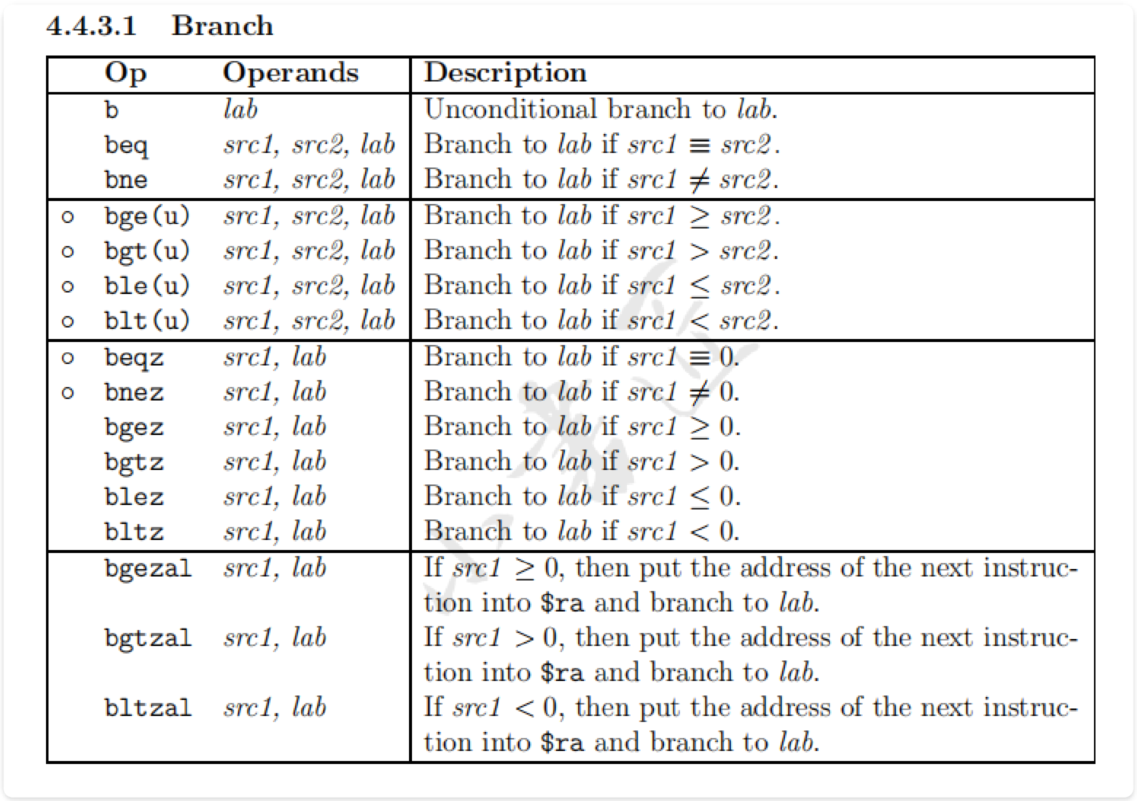


4.4.2 Comparison Instructions

	Op	Operands	Description
○	`seq`	*des, src1, src2*	$des \leftarrow 1$ if $src1 = src2$, 0 otherwise.
○	`sne`	*des, src1, src2*	$des \leftarrow 1$ if $src1 \neq src2$, 0 otherwise.
○	`sge(u)`	*des, src1, src2*	$des \leftarrow 1$ if $src1 \geq src2$, 0 otherwise.
○	`sgt(u)`	*des, src1, src2*	$des \leftarrow 1$ if $src1 > src2$, 0 otherwise.
○	`sle(u)`	*des, src1, src2*	$des \leftarrow 1$ if $src1 \leq src2$, 0 otherwise.
	`slt(u)`	*des, src1, src2*	$des \leftarrow 1$ if $src1 < src2$, 0 otherwise.

ins2

分支指令

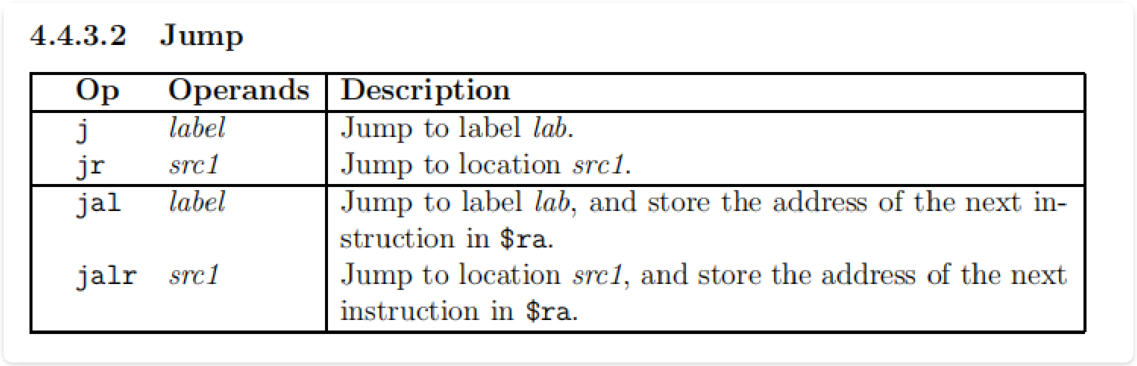


4.4.3.1 Branch

	Op	Operands	Description
	`b`	*lab*	Unconditional branch to *lab*.
	`beq`	*src1, src2, lab*	Branch to *lab* if $src1 \equiv src2$.
	`bne`	*src1, src2, lab*	Branch to *lab* if $src1 \neq src2$.
○	`bge(u)`	*src1, src2, lab*	Branch to *lab* if $src1 \geq src2$.
○	`bgt(u)`	*src1, src2, lab*	Branch to *lab* if $src1 > src2$.
○	`ble(u)`	*src1, src2, lab*	Branch to *lab* if $src1 \leq src2$.
○	`blt(u)`	*src1, src2, lab*	Branch to *lab* if $src1 < src2$.
○	`beqz`	*src1, lab*	Branch to *lab* if $src1 \equiv 0$.
○	`bnez`	*src1, lab*	Branch to *lab* if $src1 \neq 0$.
	`bgez`	*src1, lab*	Branch to *lab* if $src1 \geq 0$.
	`bgtz`	*src1, lab*	Branch to *lab* if $src1 > 0$.
	`blez`	*src1, lab*	Branch to *lab* if $src1 \leq 0$.
	`bltz`	*src1, lab*	Branch to *lab* if $src1 < 0$.
	`bgezal`	*src1, lab*	If $src1 \geq 0$, then put the address of the next instruction into `$ra` and branch to *lab*.
	`bgtzal`	*src1, lab*	If $src1 > 0$, then put the address of the next instruction into `$ra` and branch to *lab*.
	`bltzal`	*src1, lab*	If $src1 < 0$, then put the address of the next instruction into `$ra` and branch to *lab*.

ins3

调转指令

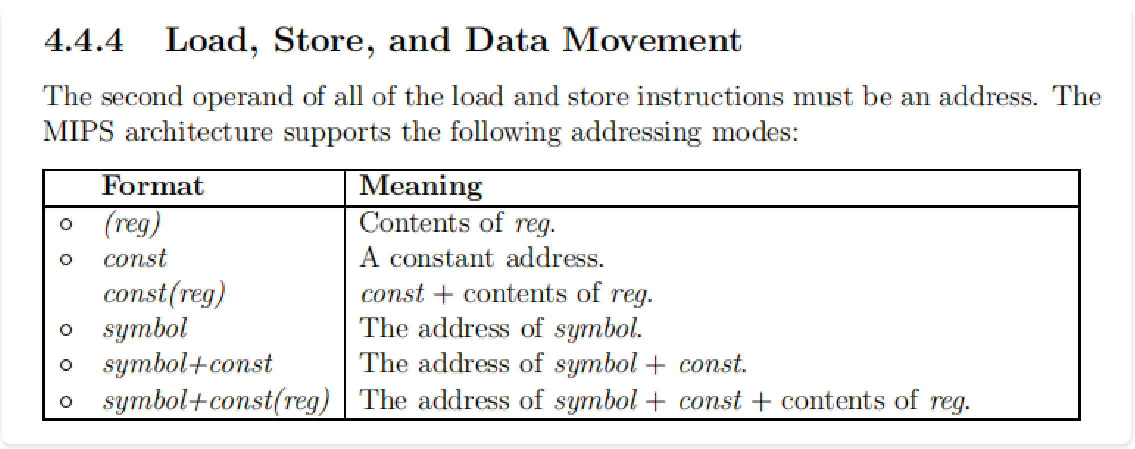


4.4.3.2 Jump

Op	Operands	Description
`j`	*label*	Jump to label *lab*.
`jr`	*src1*	Jump to location *src1*.
`jal`	*label*	Jump to label *lab*, and store the address of the next instruction in `$ra`.
`jalr`	*src1*	Jump to location *src1*, and store the address of the next instruction in `$ra`.

ins4

数据操作指令



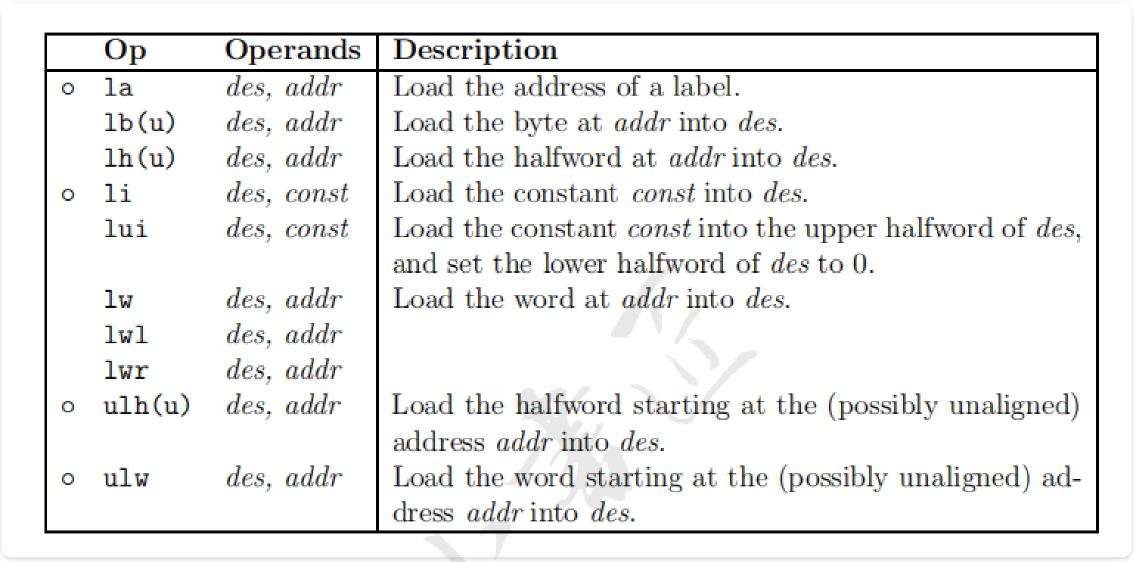
4.4.4 Load, Store, and Data Movement

The second operand of all of the load and store instructions must be an address. The MIPS architecture supports the following addressing modes:

	Format	Meaning
○	*(reg)*	Contents of *reg*.
○	*const*	A constant address.
	const(reg)	*const* + contents of *reg*.
○	*symbol*	The address of *symbol*.
○	*symbol+const*	The address of *symbol* + *const*.
○	*symbol+const(reg)*	The address of *symbol* + *const* + contents of *reg*.

ins5

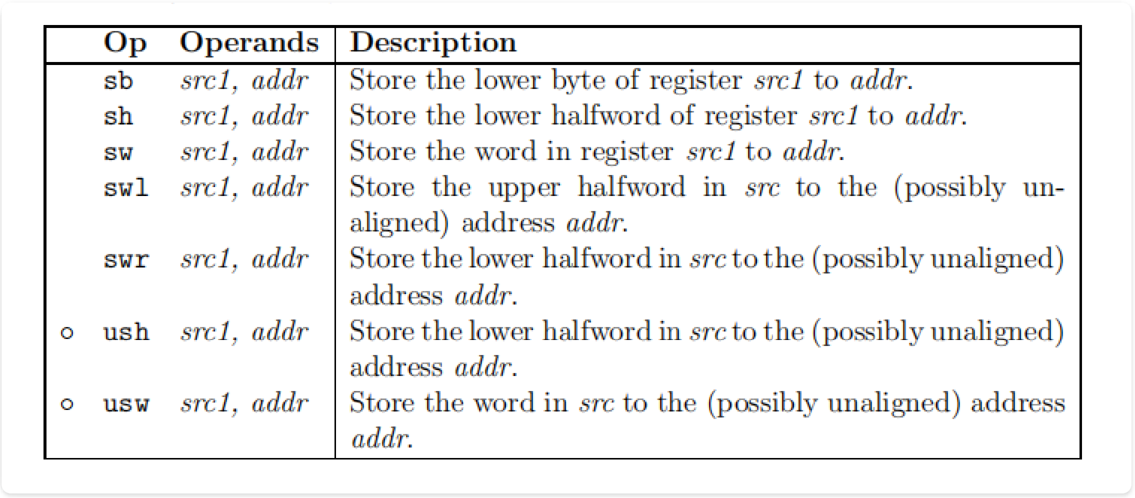
1. load



	Op	Operands	Description
○	`la`	*des, addr*	Load the address of a label.
	`lb(u)`	*des, addr*	Load the byte at *addr* into *des*.
	`lh(u)`	*des, addr*	Load the halfword at *addr* into *des*.
○	`li`	*des, const*	Load the constant *const* into *des*.
	`lui`	*des, const*	Load the constant *const* into the upper halfword of *des*, and set the lower halfword of *des* to 0.
	`lw`	*des, addr*	Load the word at *addr* into *des*.
	`lwl`	*des, addr*	
	`lwr`	*des, addr*	
○	`ulh(u)`	*des, addr*	Load the halfword starting at the (possibly unaligned) address *addr* into *des*.
○	`ulw`	*des, addr*	Load the word starting at the (possibly unaligned) address *addr* into *des*.

load_ins

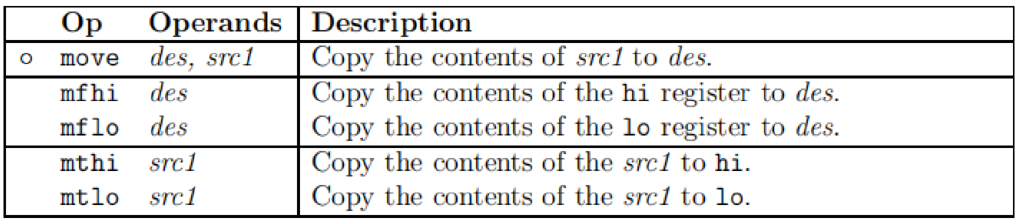
2. store



	Op	Operands	Description
	`sb`	*src1, addr*	Store the lower byte of register *src1* to *addr*.
	`sh`	*src1, addr*	Store the lower halfword of register *src1* to *addr*.
	`sw`	*src1, addr*	Store the word in register *src1* to *addr*.
	`swl`	*src1, addr*	Store the upper halfword in *src* to the (possibly unaligned) address *addr*.
	`swr`	*src1, addr*	Store the lower halfword in *src* to the (possibly unaligned) address *addr*.
○	`ush`	*src1, addr*	Store the lower halfword in *src* to the (possibly unaligned) address *addr*.
○	`usw`	*src1, addr*	Store the word in *src* to the (possibly unaligned) address *addr*.

store_ins

3. mov



	Op	Operands	Description
○	`move`	*des*, *src1*	Copy the contents of *src1* to *des*.
	`mfhi`	*des*	Copy the contents of the `hi` register to *des*.
	`mflo`	*des*	Copy the contents of the `lo` register to *des*.
	`mthi`	*src1*	Copy the contents of the *src1* to `hi`.
	`mtlo`	*src1*	Copy the contents of the *src1* to `lo`.

mov_ins