一、算术逻辑类指令

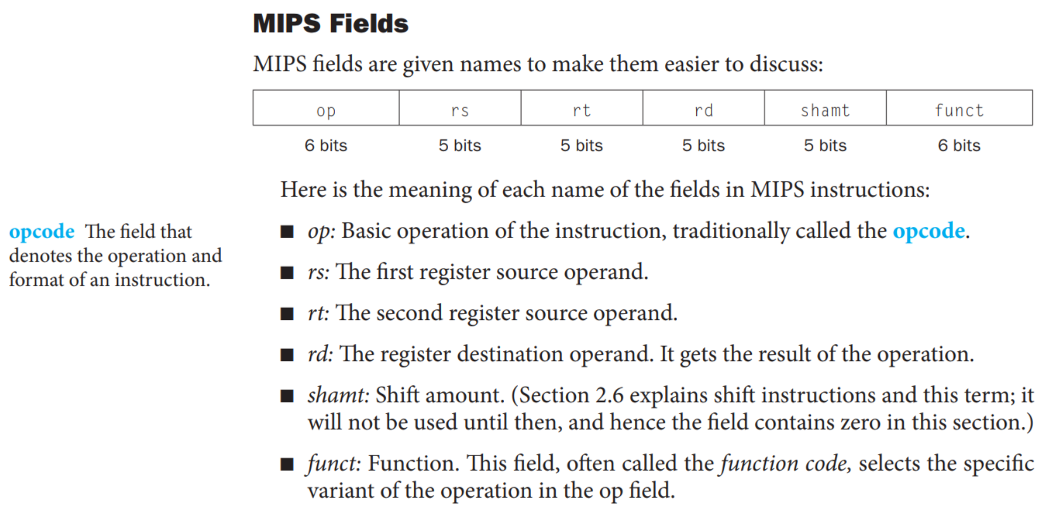
编译过程

$$High\ level\ language(e.g.\ C, Java): a = b + c; \tag{1}$$
$$Intermediate\ Representation\ for\ MIPS:\ add\ a,\ b,\ c \tag{2}$$
$$Assembly\ Language\ for\ MIPS: add\ \ \$t0,\ \$s1,\ \$s2 \tag{3}$$
$$Machine\ Language\ for\ MIPS: 000000\ 10001\ 10010\ 01000\ 00000\ 100000 \tag{4}$$

> 结合第一章的复习。

MIPS汇编转变成机器语言

R-format instruction



MIPS Fields

MIPS fields are given names to make them easier to discuss:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

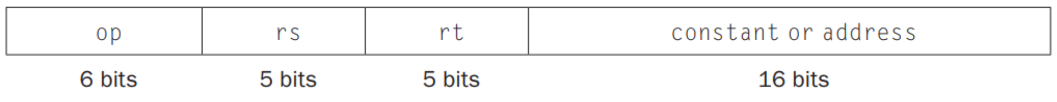
Here is the meaning of each name of the fields in MIPS instructions:

opcode The field that denotes the operation and format of an instruction.

- *op:* Basic operation of the instruction, traditionally called the **opcode**.
- *rs:* The first register source operand.
- *rt:* The second register source operand.
- *rd:* The register destination operand. It gets the result of the operation.
- *shamt:* Shift amount. (Section 2.6 explains shift instructions and this term; it will not be used until then, and hence the field contains zero in this section.)
- *funct:* Function. This field, often called the *function code,* selects the specific variant of the operation in the op field.

> 操作码与功能码来共同确定一条指令的功能。

I-format instruction(e.g. addi, load/store)



op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

> ```
> lw $t0, 8($s3)
> ```
>
> s3：基址寄存器
>
> 8：偏移量

> load与store是I型指令，所以说地址总共有16位地址，意味着`lw`指令可以加载相对于寄存器rs中地址偏移$[-2^{15}, 2^{15}-1]$个字节的值(32768Byte⇒64KB)。

补充：逻辑操作

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

```
sll  $t2,$s0,4  # reg $t2 = reg $s0 << 4 bits
```

We delayed explaining the *shamt* field in the R-format. Used in shift instructions, it stands for *shift amount*. Hence, the machine language version of the instruction above is

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

> RS寄存器被置为0，rt表示源操作数($S0)，rd表示目的操作数($T2)。shmat表示位移量

练习：手动给指令做翻译

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

FIGURE 2.6 MIPS architecture revealed through Section 2.5. The two MIPS instruction formats so far are R and I. The first 16 bits are the same: both contain an *op* field, giving the base operation; an *rs* field, giving one of the sources; and the *rt* field, which specifies the other source operand, except for load word, where it specifies the destination register. R-format divides the last 16 bits into an *rd* field, specifying the destination register; the *shamt* field, which Section 2.6 explains; and the *funct* field, which specifies the specific operation of R-format instructions. I-format combines the last 16 bits into a single *address* field.

练习：猜猜看是那个指令?

> (关于t0, t1, t2的编号，看附录A)

Check Yourself

What MIPS instruction does this represent? Choose from one of the four options below.

op	rs	rt	rd	shamt	funct
0	8	9	10	0	34

1. `sub $t0, $t1, $t2`
2. `add $t2, $t0, $t1`
3. `sub $t2, $t1, $t0`
4. `sub $t2, $t0, $t1`

存储程序的概念

今天的计算机建立在两个关键原则之上：

1. 指令用数字表示。
2. 指令存储在内存中以供读取或写入，就像数据一样。这些原则导致了存储程序的概念；

> 具体来说，内存可以包含编辑器程序的源代码、相应的编译机器代码、编译程序正在使用的文本，甚至是生成机器代码的编译器。
>
> 指令作为数字的一个结果是程序通常作为二进制数字文件交付。商业含义是计算机可以继承现成的软件，只要它们与现有的指令集兼容。这种“二进制兼容性”通常会导致行业围绕少数指令集架构进行调整。

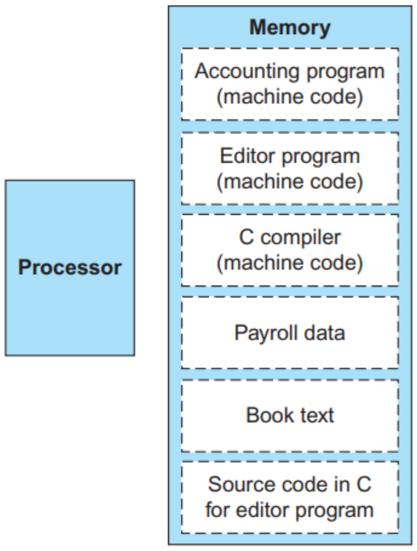

Memory
Accounting program (machine code)
Editor program (machine code)
C compiler (machine code)
Payroll data
Book text
Source code in C for editor program
Processor

FIGURE 2.7 The stored-program concept. Stored programs allow a computer that performs accounting to become, in the blink of an eye, a computer that helps an author write a book. The switch happens simply by loading memory with programs and data and then telling the computer to begin executing at a given location in memory. Treating instructions in the same way as data greatly simplifies both the memory hardware and the software of computer systems. Specifically, the memory technology needed for data can also be used for programs, and programs like compilers, for instance, can translate code written in a notation far more convenient for humans into code that the computer can understand.

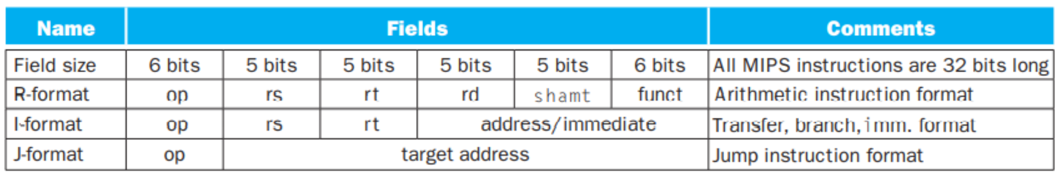
二、决策类(BRANCH)指令

> 编译器使用slt、slti、beq、bne和固定值0(总是可以通过读取寄存器＄zero来获得)来创建所有的比较条件：相等、不等、小于、小于或等于、大于、大于或等于。

> SLT: Set Less than

> 在关于条件码寄存器的部分补充全部知识。此处skip

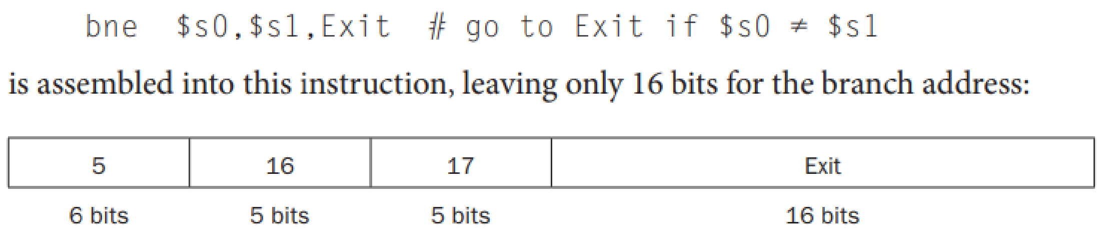
J-format instruction



Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

> j-format的格式如上所述，由6位的op与一个target address来表示。与立即数寻址不同的是，jump的寻址范围更大。
>
> Branch类指令属于J-format

BRANCH指令(J-format instruction)



```
bne  $s0,$s1,Exit  # go to Exit if $s0 ≠ $s1
```

is assembled into this instruction, leaving only 16 bits for the branch address:

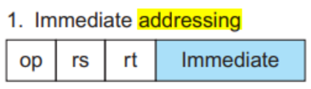
5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

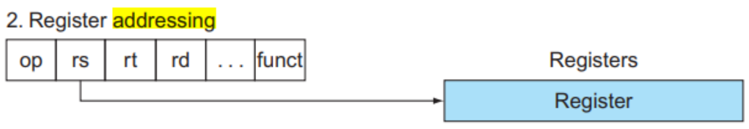
如果这样的话，我们仅仅可以访问16bits的地址空间，因此我们在此采用PC相对寻址：

$$Program\ counter\ = Register\ + Branch\ address \tag{5}$$

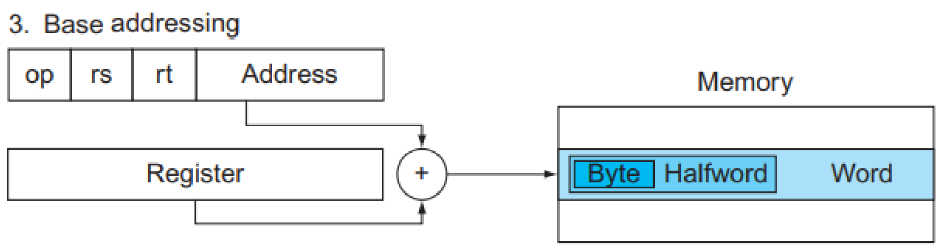
SUMMARY: MIPS寻址相关问题的总结。

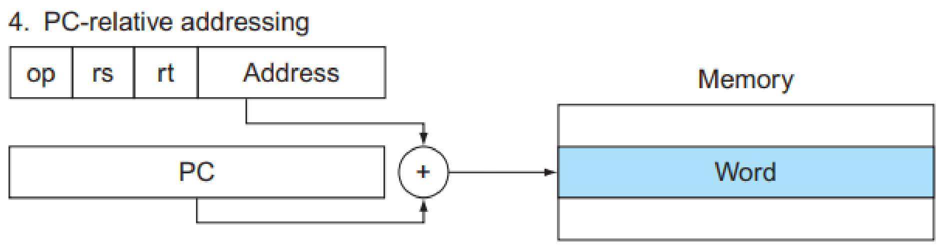
1. 立即数与寄存器寻址



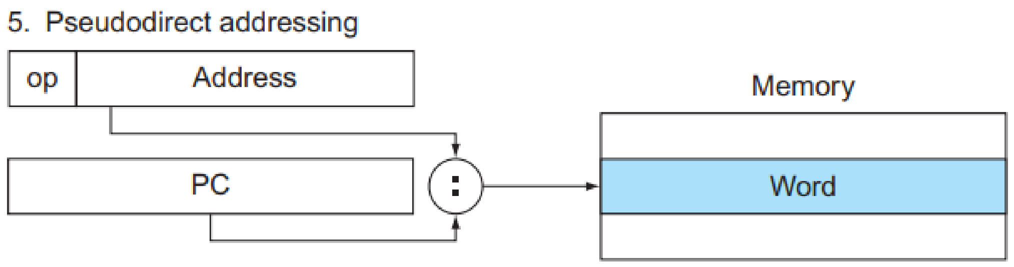


2. 基址寻址与PC相对寻址





3. 伪直接寻址



> 由于MIPS按字寻址，所以说PC相对寻址会将16位地址左移2位与PC相加，而伪直接寻址把26位地址左移2位与PC计数器的高4位相连。

练习题

在MIPS中条件分支的地址范围(K=1024)是多大?

1. 地址在$0 \sim 64K-1$之间;
2. 地址在$0 \sim 256K-1$之间;

3. 分支前后地址范围各自大约32K;
4. 分支前后地址范围各大约128K;

在MIPS中跳转和跳转链接指令的地址范围(M=1024K)是多大?

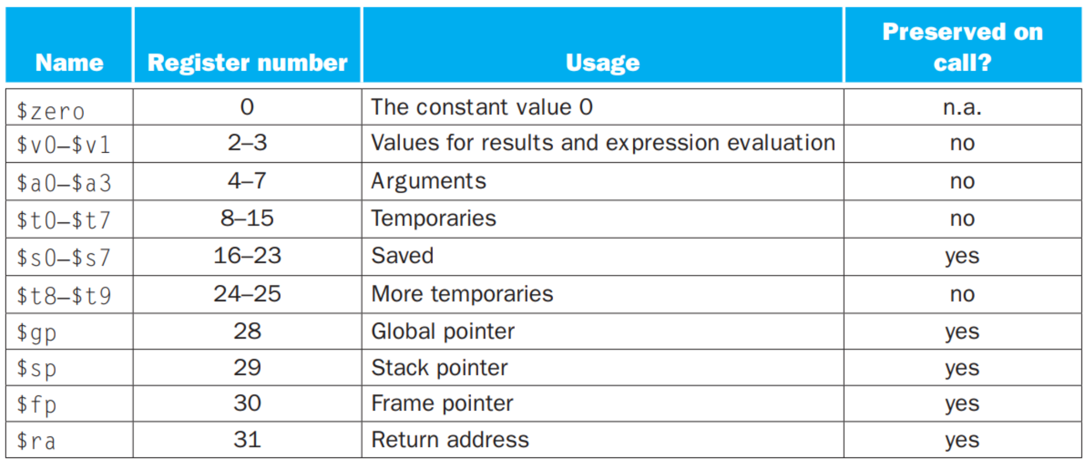
1. 地址在0～64M-1之间
2. 地址在0～256M-1之间
3. 分支前后地址范围各自大约32M
4. 分支前后地址范围各自大约128M
5. 由PC提供高6位地址的64M大小的块中任意地址
6. 由PC提供高4位地址的256M大小的块中任意地址

三、对过程的支持

?

附录

A. 调用约定



Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

FIGURE 2.14 MIPS register conventions. Register 1, called `$at`, is reserved for the assembler (see Section 2.12), and registers 26–27, called `$k0–$k1`, are reserved for the operating system. This information is also found in Column 2 of the MIPS Reference Data Card at the front of this book.

B. MIPS指令SUMMARY

见PDF