

Lecture 19: MIPS Assembly Language

主要内容

- ◆ MIPS指令格式
 - R-类型 / I-类型 / J-类型
- ◆ MIPS寄存器
 - 长度 / 个数 / 功能分配
- ◆ MIPS操作数
 - 寄存器操作数 / 存储器操作数 / 立即数 / 文本 / 位
- ◆ MIPS指令寻址方式
 - 立即数寻址 / 寄存器寻址 / 相对寻址 / 伪直接寻址 / 偏移寻址
- ◆ MIPS指令类型
 - 算术 / 逻辑 / 数据传送 / 条件分支 / 无条件转移
- ◆ MIPS汇编语言形式
 - 操作码的表示 / 寄存器的表示 / 存储器数据表示
- ◆ 机器语言的解码（反汇编）
- ◆ 高级语言、汇编语言、机器语言之间的转换
- ◆ 过程调用与堆栈

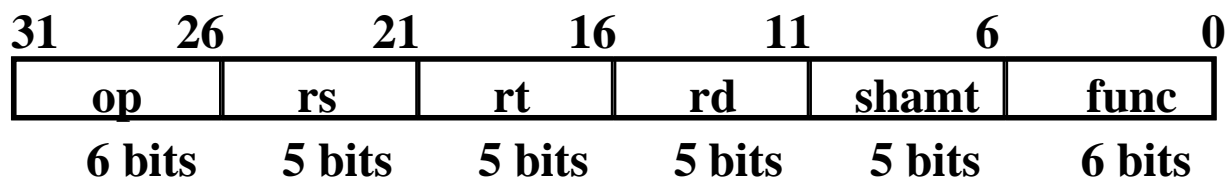
MIPS指令格式

- 所有指令都是32位宽，须按字地址对齐

R-Type指令

- 有三种指令格式

– R-Type

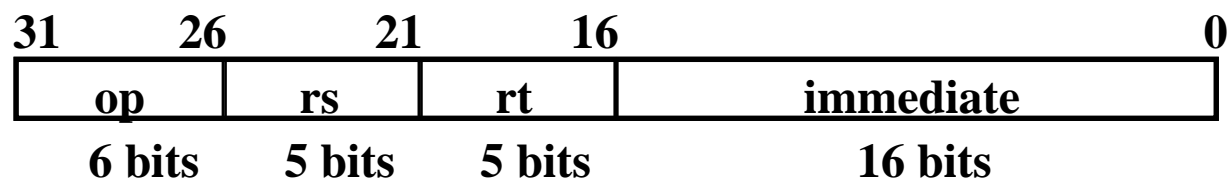


两个操作数和结果都在寄存器的运算指令。如： **sub rd, rs, rt**

– I-Type

- 运算指令：一个寄存器、一个立即数。如： **ori rt, rs, imm16**
- LOAD和STORE指令。如： **lw rt, rs, imm16**
- 条件分支指令。如： **beq rs, rt, imm16**

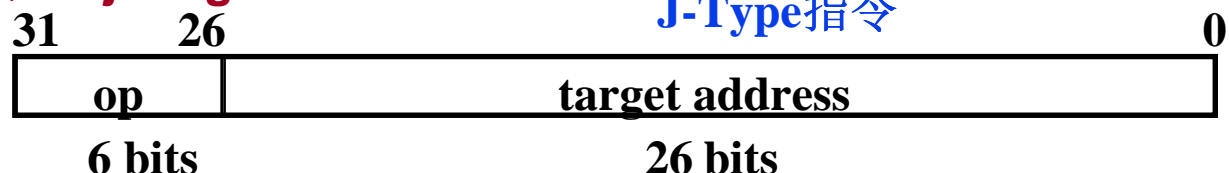
I-Type指令



– J-Type

无条件跳转指令。如： **j target**

J-Type指令



MIPS指令字段含义

R-Type指令

OP: 操作码

rs: 第一个源操作数寄存器

rt: 第二个源操作数寄存器

rd: 结果寄存器

shamt: 移位指令的位移量

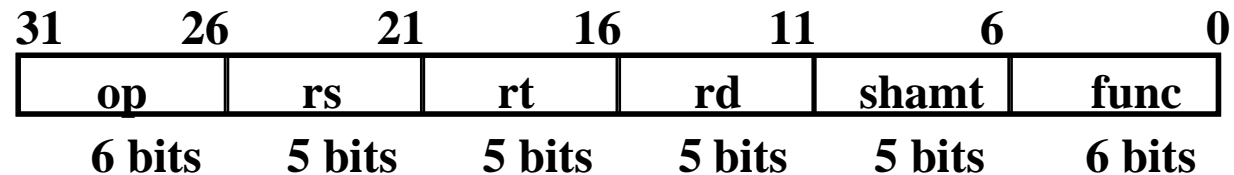
func: R-Type指令的OP字段是特定的“000000”，具体操作由func字段给定。例如: func=“100000”时, 表示“加法”运算。

操作码的不同编码定义不同的含义, 操作码相同时, 再由功能码定义不同的含义!

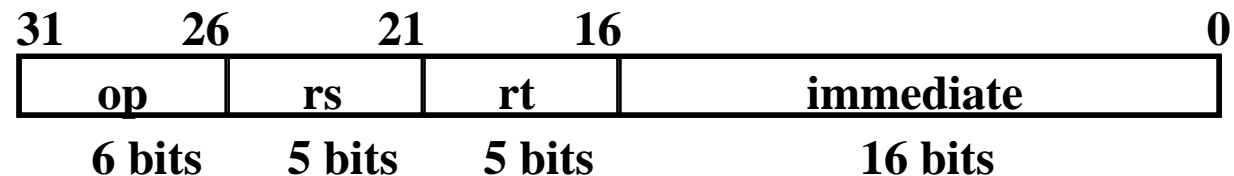
immediate: 立即数或load/store指令和分支指令的偏移地址

target address: 无条件转移地址的低26位。将PC高4位拼上26位直接地址, 最后添2个“0”就是32位目标地址。为何最后两位要添“0”?

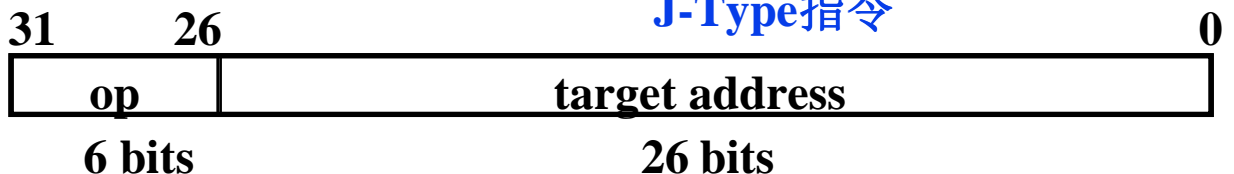
指令按字地址对齐, 所以每条指令的地址都是4的倍数 (最后两位为0)。



I-Type指令



J-Type指令



OP字段的含义（MIPS指令的操作码编码/解码表）

op(31:26)								
op=0:R型; op=2/3:J型; 其余:I型								
28-26	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
31-29								
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper imm
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	lbu	lhu	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	lwc0	lwc1						
7(111)	swc0	swc1						

[Back to Load/Store](#)

[BACK to Assemble](#)

R-型指令的解码（**op=0**时，**func**字段的编码/解码表）

op(31:26)=000000 (R-format), func(5:0)								
2-0 5-3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
	shift left logical		shift right logical	sra	sllv		srlv	srav
	jump reg.	jalr			syscall	break		
	mfhi	mthi	mflo	mtlo				
	mult	multu	div	divu				
	add	addu	subtract	subu	and	or	xor	not or (nor)
			set l t.	sltu				

add指令的func字段为100000B（32）

div指令的func字段为多少？

011010B（26）！

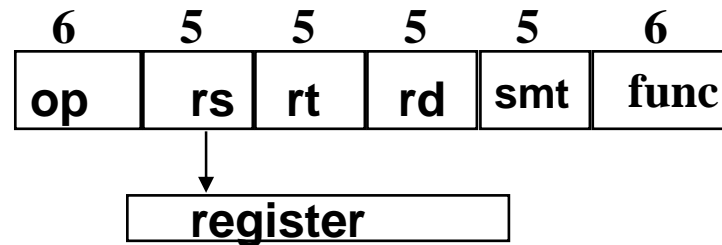
[**BACK to Assemble**](#)

MIPS Addressing Modes (寻址方式)

有专门的寻址方式
字段 (Mod) 吗?

R-format:

Register



没有! 由指令格式来
确定, 而指令格式由
op来确定!

I-format:

Immediate

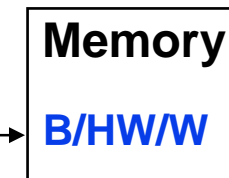


还记得如何确定的吗?

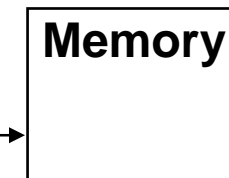
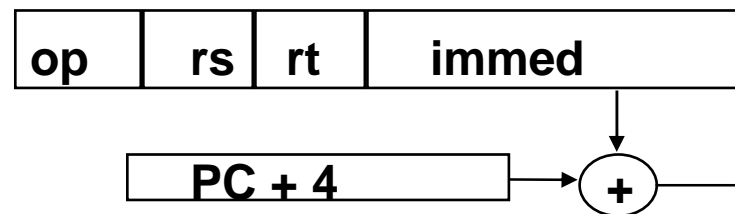
Base或index
基址或变址



Byte / Half Word / Word

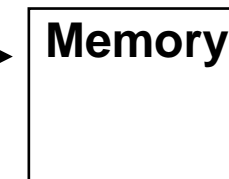
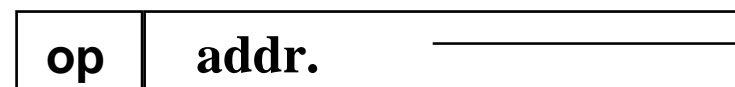


PC-relative
相对寻址



J-format:

Pseudodirect
伪直接寻址



为什么称伪直接? 还记得如何得到最终地址的吗?
最终地址 = $PC_{31 \sim 28} || \text{addr.} || 00$ 位数: $4 + 26 + 2 = 32$

Example: 汇编形式与指令的对应

- ◆ 若从存储器取来一条指令为00AF8020H，则对应的汇编形式是什么？

32位指令代码：0000 0000 1010 1111 1000 0000 0010 0000

指令的前6位为000000，根据[指令解码表](#)知，是一条R-Type指令，按照R-Type指令的格式

31	6 bits	26	5 bits	21	5 bits	16	5 bits	11	5 bits	6	6 bits	0
op		rs		rt		rd		shamt		func		
000000		00101		01111		10000		00000		100000		

得到：rs=00101, rt=01111, rd=10000, shamt=00000, funct=100000

1. 根据[R-Type指令解码表](#)，知是“add”操作（非移位操作）
2. rs、rt、rd的十进制值分别为5、15、16，从[MIPS寄存器功能表](#)知：

rs、rt、rd分别为：\$a1、\$t7、\$s0

故对应的汇编形式为：

add \$s0 , \$a1, \$t7

功能：\$a1 + \$t7 → \$s0

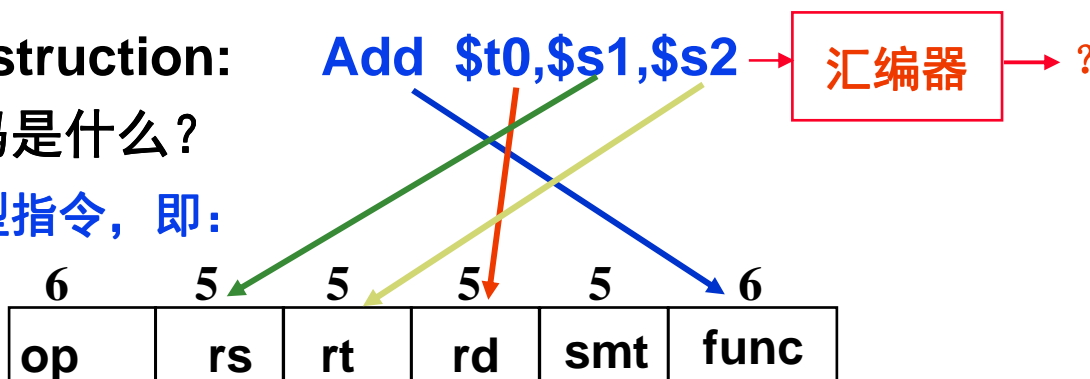
这个过程称为“反汇编”，可用来破解他人的二进制代码（可执行程序）。

Example: 汇编形式与指令的对应

◆ 若MIPS Assembly Instruction:

则对应的指令机器代码是什么?

从助记符表中查到Add是R型指令, 即:



Decimal representaton:

6	5	5	5	5	6
0	17	18	8	0	32
R-Type	\$s1	\$s2	\$t0	No shift	Add

Binary representaton:

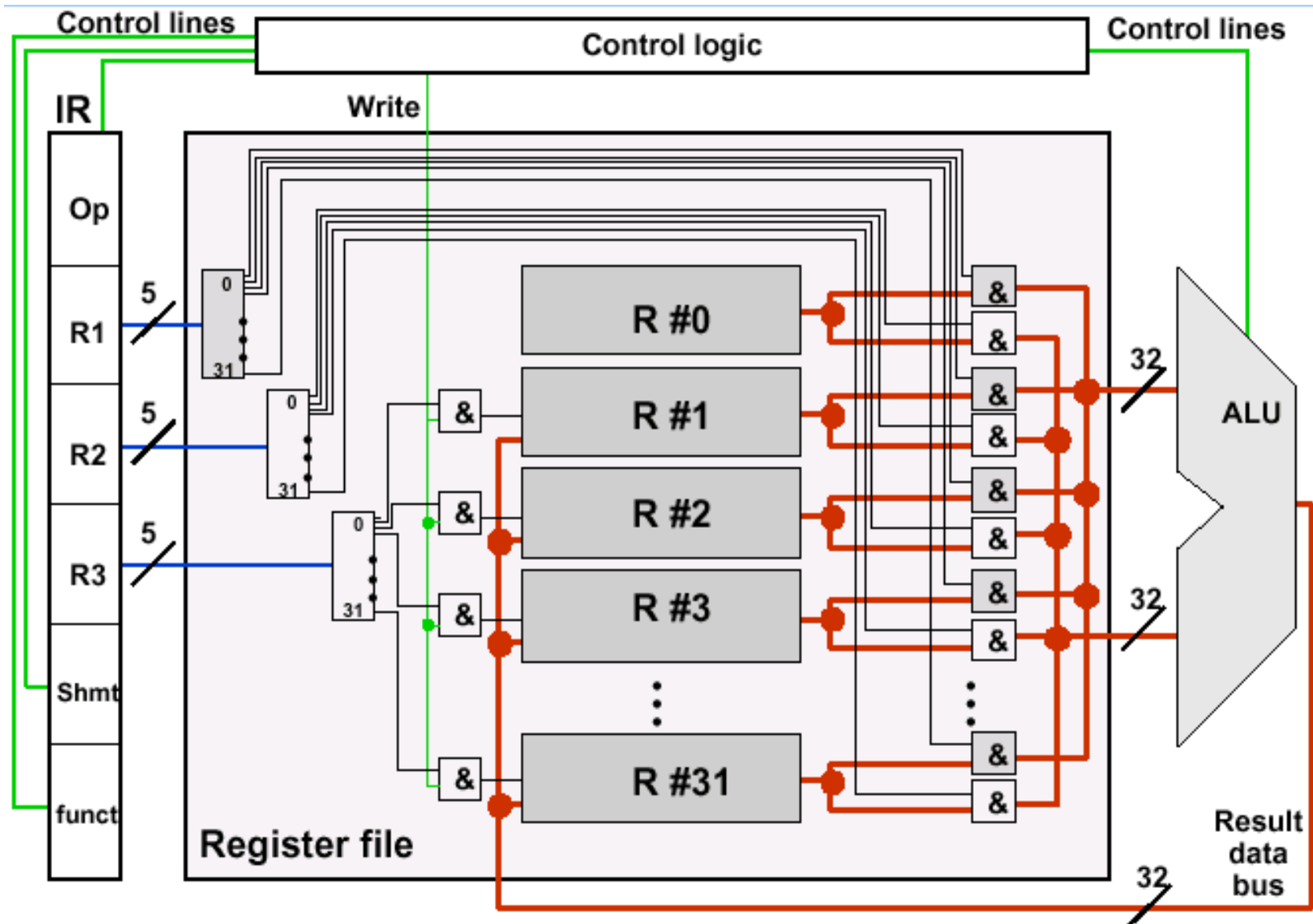
6	5	5	5	5	6
000000	10001	10010	01000	00000	100000

问题: 如何知道是R型指令?

根据汇编指令中的操作码助记符查表能知道是什么格式!

这个过程称为“汇编”, 所有汇编源程序都必须汇编成二进制机器代码才能让机器直接执行!

MIPS Circuits for R-Type Instructions



问题：你能给出R-型指令在上述通路中的大致执行过程吗？

MIPS R-type指令实现电路的执行过程

Phase1: Preparation (1: 准备阶段)

- ⌚ 装入指令寄存器IR
- ⌚ 以下相应字段送控制逻辑
 - op field (OP字段)
 - func field (func字段)
 - shmt field (shmt字段)
- ⌚ 以下相应字段送寄存器堆
 - 第一操作数寄存器编号
 - 第二操作数寄存器编号
 - 存放结果的目标寄存器编号

这个过程描述仅是示意性的，实际上整个过程需要时钟信号的控制，并有其他部件参与。
将在下一章详细介绍。

Phase2: Execution(2: 执行阶段)

- ⌚ 寄存器号被送选择器
- ⌚ 对应选择器输出被激活
- ⌚ 被选寄存器的输出送到数据线
- ⌚ 控制逻辑提供：
 - ALU操作码
 - 写信号 等
- ⌚ 结果被写回目标寄存器

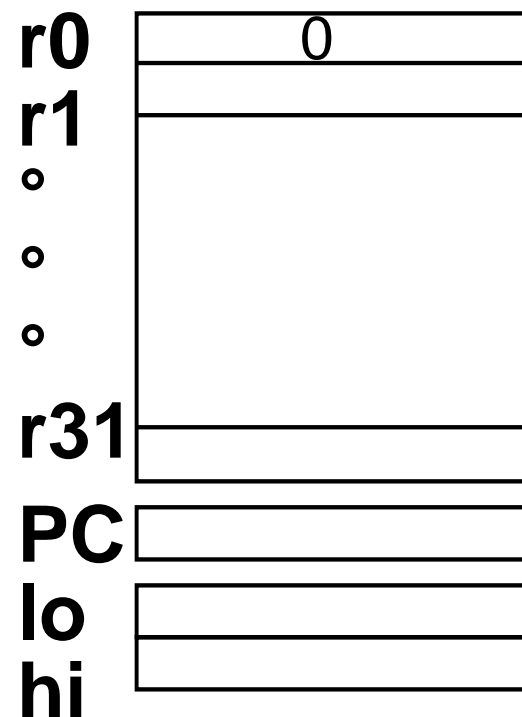
MIPS指令中寄存器数据和存储器数据的指定

◆寄存器数据指定：

- 31 x 32-bit GPRs ($r0 = 0$)
- 寄存器编号占5 bit
- 32 x 32-bit FP regs ($f0 \sim f31$, paired DP)
- HI, LO, PC: 特殊寄存器 (无需编号)
- 寄存器功能和2种汇编表示方式

◆存储器数据指定

- 32-bit machine --> 可访问空间: 2^{32} bytes
- Big Endian(大端方式)
- 只能通过Load/Store指令访问存储器数据
- 数据地址通过一个32位寄存器内容加16位偏移量得到
- 16位偏移量是带符号整数，故应符号扩展
- 数据要求按边界对齐 (地址是4的倍数)



SKIP

MIPS寄存器的功能定义和两种汇编表示

Name	number	Usage	Reserved on call?
zero	0	constant value =0(恒为0)	n.a.
at	1	reserved for assembler(为汇编程序保留)	n.a.
v0 ~ v1	2 ~ 3	values for results(过程调用返回值)	no
a0 ~ a3	4 ~ 7	Arguments(过程调用参数)	yes
t0 ~ t7	8 ~ 15	Temporaries(临时变量)	no
s0 ~ s7	16 ~ 23	Saved(保存)	yes
t8 ~ t9	24 ~ 25	more temporaries(其他临时变量)	no
k0 ~ k1	26 ~ 27	reserved for kernel(为OS保留)	n.a.
gp	28	global pointer(全局指针)	yes
sp	29	stack pointer (栈指针)	yes
fp	30	frame pointer (帧指针)	yes
ra	31	return address (过程调用返回地址)	yes

Registers are referenced either by number—\$0, ... \$31, or by name —\$t0, \$s1... \$ra.

zero	at	v0-v1	a0 - a3	t0 - t7	s0 - s7	t8 - t9	k0 - k1	gp	sp	fp	ra
------	----	-------	---------	---------	---------	---------	---------	----	----	----	----

0 1 2 - 3 4 - 7 8 --- 15 16 --- 23 24 - 25 26 - 27 28 29 30 31

[BACK to Assemble](#)

[BACK to Procedure](#)

[BACK to last](#)

MIPS arithmetic and logic instructions

需要判溢出，溢出时发生“异常”

<i>Instruction</i>	<i>Example</i>	<i>Meaning</i>	<i>Comments</i>
add	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; exception possible
subtract	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; exception possible
add immediate	addi \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; exception possible
multiply	mult \$2,\$3	Hi, Lo = $\$2 \times \3	64-bit signed product
divide	div \$2,\$3	Lo = $\$2 \div \3 , Hi = $\$2 \bmod \3	Lo = quotient, Hi = remainder
Move from Hi	mfhi \$1	$\$1 = \text{Hi}$	get a copy of Hi
Move from Lo	mflo \$1	$\$1 = \text{lo}$	
<i>Instruction</i>	<i>Example</i>	<i>Meaning</i>	<i>Comment</i>
and	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	Logical AND
or	or \$1,\$2,\$3	$\$1 = \$2 \$3$	Logical OR
xor	xor \$1,\$2,\$3	$\$1 = \$2 \oplus \$3$	Logical XOR
nor	nor \$1,\$2,\$3	$\$1 = \sim(\$2 \$3)$	Logical NOR

这里没有全部列出，还有其他指令，如addu（不带溢出处理）， addui 等

Example: 算术运算

E.g. $f = (g+h) - (i+j)$,
assuming f, g, h, i, j be assigned to $\$1, \$2, \$3, \$4, \$5$

```
add $7, $2, $3  
add $8, $4, $5  
sub $1, $7, $8
```

寄存器资源由编译器分配！

简单变量尽量被分配在寄存器中，为什么？

程序中的常数如何处理呢？

E.g. $f = (g+100) - (i+50)$ →

```
addi $7, $2, 100  
addi $8, $4, 50  
sub $1, $7, $8
```

MIPS data transfer instructions

<i>Instruction</i>	<i>Comment</i>	<i>Meaning</i>
SW \$3, 500(\$4)	Store word	\$3 \rightarrow (\$4+ 500)
SH \$3, 502(\$2)	Store half	Low Half of \$3 \rightarrow (\$2+ 502)
SB \$2, 41(\$3)	Store byte	LQ of \$2 \rightarrow (\$3+ 41)
LW \$1, -30(\$2)	Load word	(\$2 -30) \rightarrow \$1
LH \$1, 40(\$3)	Load half	(\$3+40) \rightarrow LH of \$1
LB \$1, 40(\$3)	Load byte	(\$3+40) \rightarrow LQ of \$1

操作数长度的不同由不同的操作码指定。

问题：为什么指令必须支持不同长度的操作数？

因为高级语言中的数据类型有char, short, int, long,.....等，故需要存取不同长度的操作数；操作数长度和指令长度没有关系

Example (Base register)

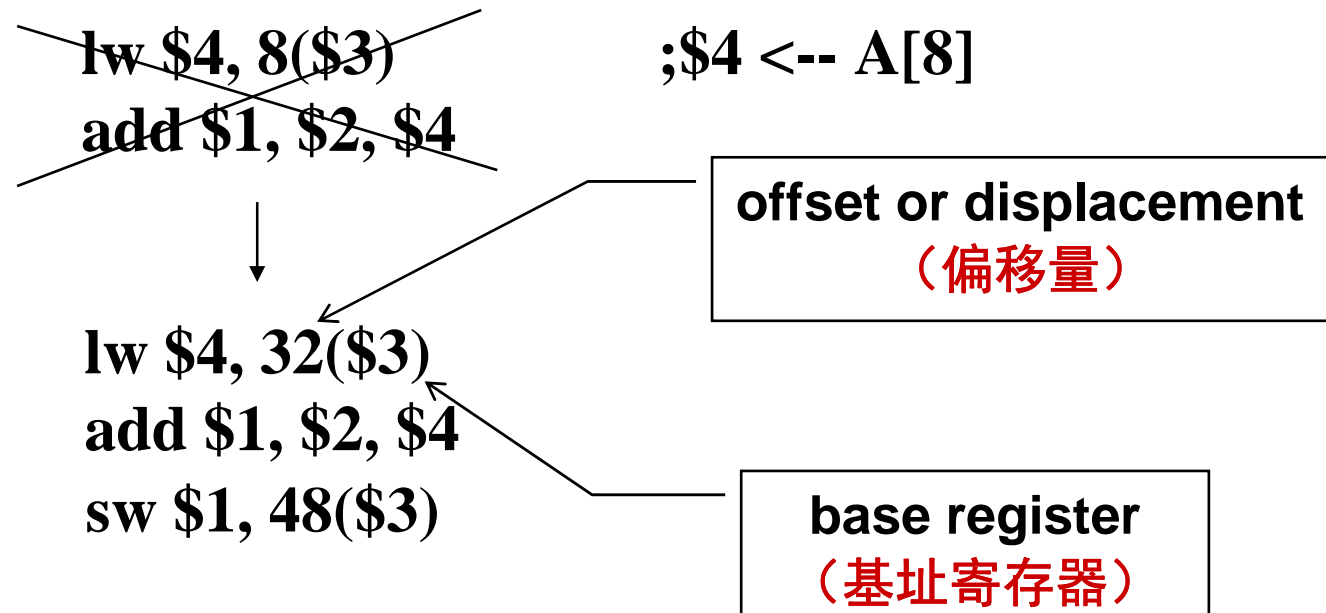
Assume A is an array of 100 **words**, and compiler has associated the variables **g** and **h** with the register **\$1** and **\$2**. Assume the base address of the array is in **\$3**. Translate

$g = h + A[8]$

有没有问题？

有！因为各数组元素是字，并按字节编址，故各占4个单元。

$A[12] = h + A[8]$



MIPS的call/return/ jump/branch和compare指令

<i>Instruction</i>	<i>Example</i>	<i>Meaning</i>
jump register	jr \$31	go to \$31 <i>For switch, procedure return</i> (对应过程返回)
jump and link	jal 10000	\$31 = PC + 4; go to 10000 <i>For procedure call</i> (对应过程或函数调用)
jump	j 10000	go to 10000 <i>Jump to target address</i>

call / return

Pseudoinstruction *blt, ble, bgt, bge* 伪指令：由若干指令（即指令序列）实现。
not implemented by hardware, but synthesized by assembler

set on less than	slt \$1,\$2,\$3	if (\$2 < \$3) \$1=1; else \$1=0
set less than imm.	slti \$1,\$2,100	if (\$2 < 100) \$1=1; else \$1=0

} 按补码比较大小

问题：指令中立即数是多少？ 100=0064H

branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100
-----------------	-----------------	--------------------------------

问题：指令中立即数是多少？ 25=0019H

branch on not eq.	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100
-------------------	-----------------	--------------------------------

} 汇编中给出的是立即数符号扩展后乘4得到的值

而分支指令中给出的是相对于当前指令的指令条数！

[BACK to Procedure](#)

Example: if-then-else语句和“=”判断

```
if (i == j)
    f = g+h ;
else
    f = g-h ;
```

Assuming variables i, j, f, g, h, ~ \$1, \$2, \$3, \$4, \$5

```
                bne $1, $2, else          ; i!=j, jump to else
                add $3, $4, $5
                j  exit                    ; jump to exit
else:           sub $3, $4, $5
exit:
```

Example: “less than”判断

if (a < b) f = g+h ; else f = g-h ;

Assuming variables a, b, f, g, h, ~ \$1, \$2, \$3, \$4, \$5

✗	slt \$6, \$1, \$2	; if a<b, \$6=1, else \$6=0
	bne \$6, \$zero, else	; \$6!=0, jump to else
	add \$3, \$4, \$5	
	j exit	; jump to exit

else: sub \$3, \$4, \$5

exit:

✓	slt \$6, \$1, \$2	; if a<b, \$6=1, else \$6=0
	beq \$6, \$zero, else	; \$6=0, jump to else
	add \$3, \$4, \$5	
	j exit	; jump to exit

else: sub \$3, \$4, \$5


exit:

Example: Loop循环

Loop: $g = g + A[i];$
 $i = i + j;$
 if ($i \neq h$) go to **Loop**:

Assuming variables $g, h, i, j \sim \$1, \$2, \$3, \4 and base address of array is in $\$5$

Loop: add \$7, \$3, \$3	; $i*2$	加法比乘法快!
add \$7, \$7, \$7	; $i*4$	也可用移位来实现乘法!
add \$7, \$7, \$5		\$3中是i, \$7中是 $i*4$
lw \$6, 0(\$7)	; $\$6 = A[i]$	
add \$1, \$1, \$6	; $g = g + A[i]$	
add \$3, \$3, \$4		
bne \$3, \$2, Loop		



编译器和汇编语言程序员不必计算分支指令的地址，而只要用标号即可！汇编器完成地址计算