程序的机器级表示

MIPS 指令系统简介

主讲人: 邓倩妮

上海交通大学

本节内容

- MIPS 指令系统

MIPS的设计者、RISC的先驱

John Hennessy

- 1977年，进入斯坦福大学任职
- 1981年，领导RISC微处理器的研究小组
- 1984年，共同创立MIPS计算机系统公司
- 2000年，任斯坦福大学校长
- 2018年，与Patterson 共同获得图灵奖

约翰·亨尼西

大卫· 帕特森

David A. Patterson

- 1976~2016年， UC Berkeley计算机教授；
- 1990年, 与Hennessy合著教材：Computer Architecture: A Quantitative Approach
- 2016年, 加入谷歌TPU团队
- 2018年,与Hennessy 共同获得图灵奖

MIPS指令集发展过程

- 1981年， MIPS I
- MIPS II
- MIPS III
- MIPS IV
- MIPS V
- 嵌入式指令体系MIPS16
- 1999年， MIPS32、MIPS64
- 龙芯2号： MIPS64

MIPS 的设计思想

- MIPS的全称
  - Microprocessor without Interlocked Piped Stages
- 主要关注点
  - 减少指令的类型
  - 降低指令复杂度
- 基本原则
  - A simpler CPU is a faster CPU.

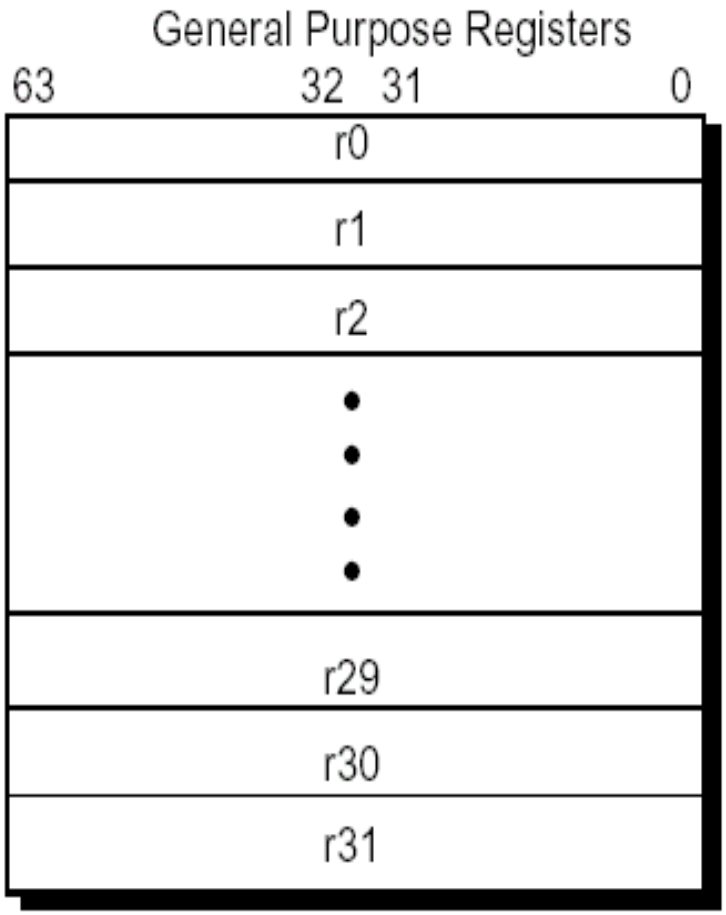
MIPS指令系统的特征

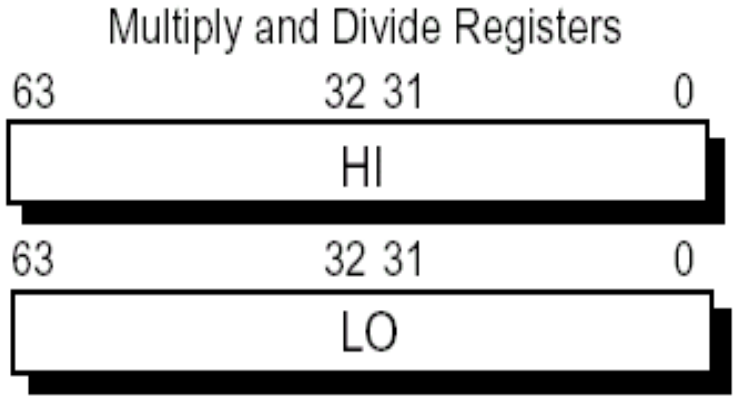
- 简单
  - 指令定长，都是32位。
  - 指令数量少、功能简单
  - 指令格式简单，只有三种：
    - 立即数型（I）、转移型（J）、寄存器型（R）
  - 操作数寻址方式少：
    - 基址加16位偏移量的访存寻址
    - 立即数寻址
    - 寄存器寻址

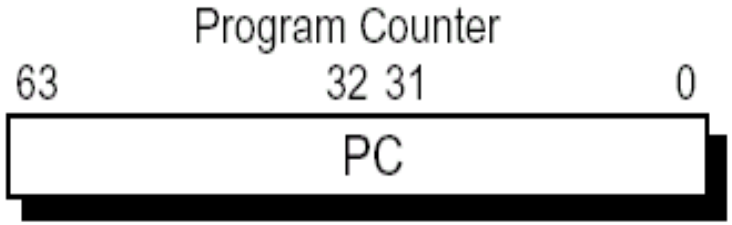
MIPS指令系统的特征（续）

- 使用较多寄存器
  - 寄存器访问速度比存储器快
  - 32个通用寄存器
- 只有Load和Store指令可以访问存储器
  - 例如，不支持x86指令的这种操作：ADD AX,[3000H]
- 需要优秀编译器支持
  - 无互锁流水段，依靠编译器进行指令序列的重新安排，减少流水线中出现的冲突

MIPS寄存器


General Purpose Registers
63
32
31
0
r0
r1
r2
r29
r30
r31


Multiply and Divide Registers
63
32
31
0
HI
63
32
31
0
LO

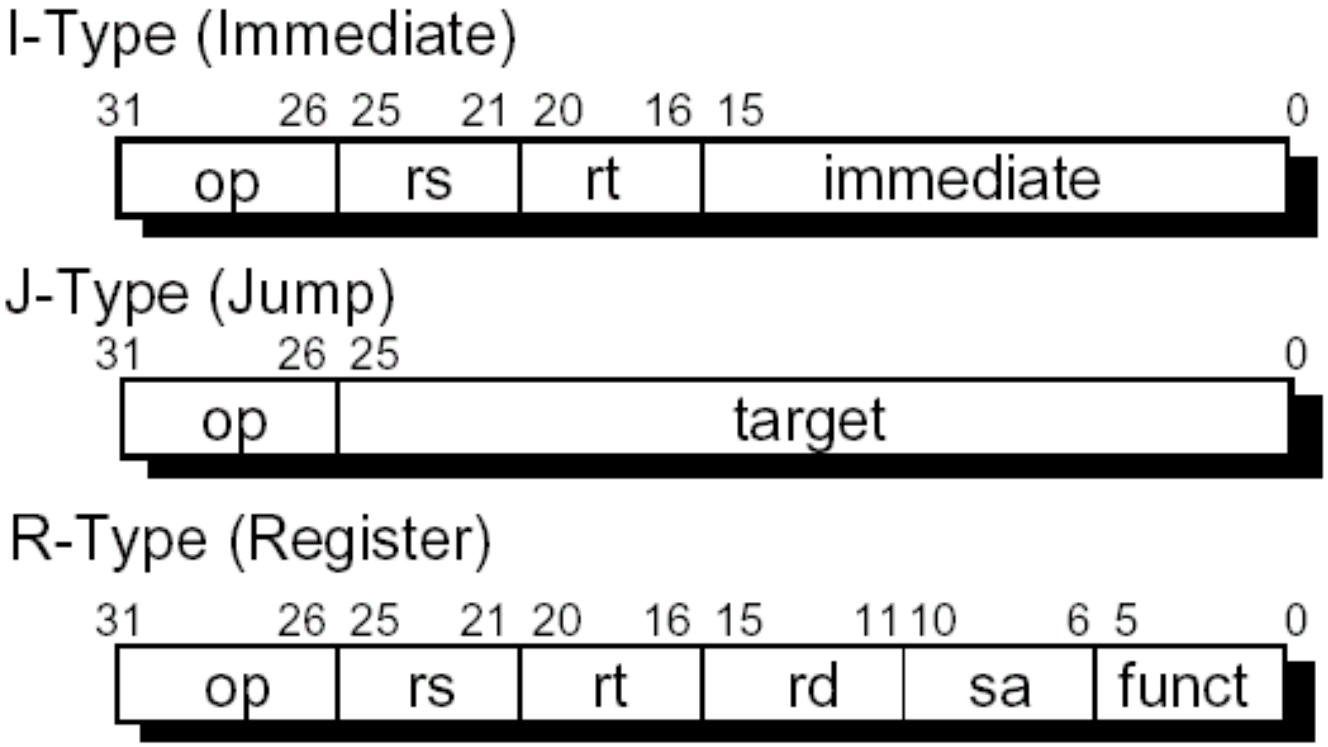

Program Counter
63
32
31
0
PC

MIPS通用寄存器（32位宽）用途

编号	名称	用途	编号	名称	用途
0	$zero	The Constant Value 0	24-25	$t8-$t9	Temporaries
1	$at	Assembler Temporary	26-27	$k0-$k1	Reserved for OS Kernel
2-3	$v0-$v1	Values for Function Results and Expression Evaluation	28*	$gp	Global Pointer
4-7	$a0-a3	Arguments	29*	$sp	Stack Pointer
8-15	$t0-$t7	Temporaries	30*	$fp	Frame Pointer
16-23*	$s0-$s7	Saved Temporaries	31*	$ra	Return Address

* 在过程调用时需要保存， Preserved across a call

MIPS指令的三种格式



MIPS指令操作码定义

I28~I26 I31~I29	000	001	010	011	100	101	110	111
000	R 格式	Bltz/gez	j	jal	beq	bne	blez	bgtz
001	addi	addiu	slti	sltiu	andi	ori	xori	lui
010	TLB 指令	浮点指令						
011								
100	lb	lh	lwl	lw	lbu	lhu	lwr	
101	sb	sh	swl	sw			swr	
110	lwc0	lwc1						
111	swc0	swc1						

MIPS数据传递指令

指令举例	操作	说明
lw $1,100($2)	$1=M[$2+100]	装入字
sw $1,100($2)	M[$2+100]=$1	存储字
lui $1,100	$1=100×$2^{16}$	装入立即数到高位

MIPS 数据传递指令 Memory Access

两种数据传送指令：

- `lw $t0, 4($s3)  #load word from memory`

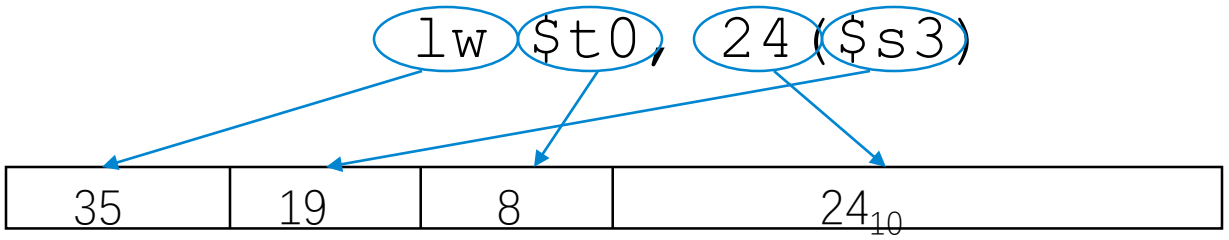
  `sw $t0, 8($s3)  #store word to memory`

- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5 bit address

- 要访问的存储地址– a 32 bit address –基址加16位偏移量 formed by adding the contents of the base address register to the offset value

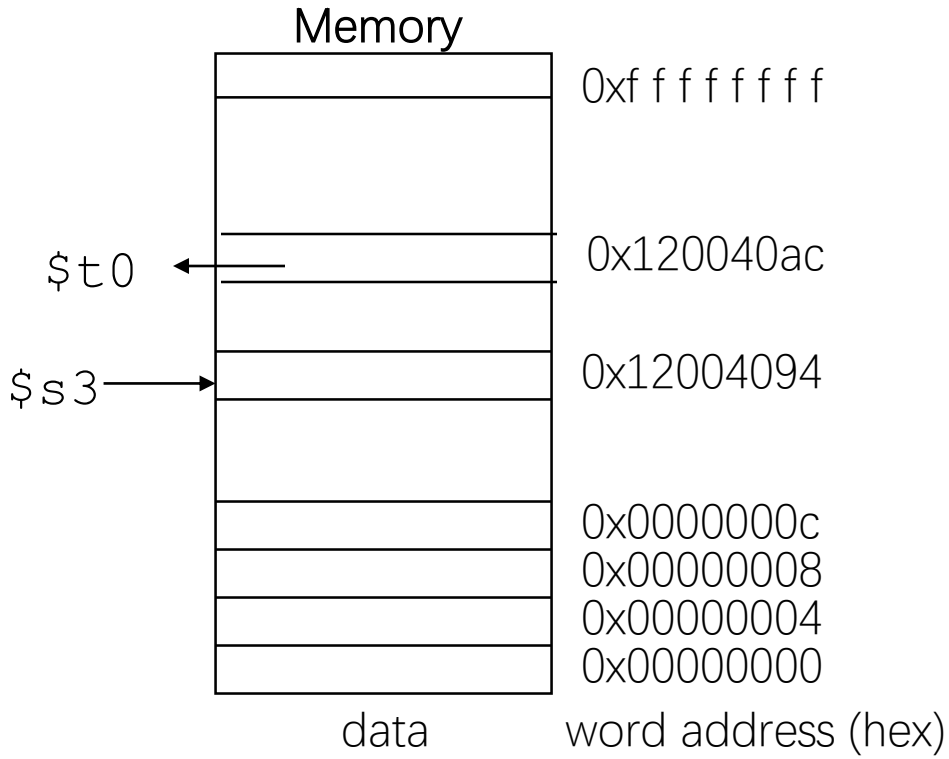
  16位偏移量 将访问范围限制在基地址偏移量为 $\pm 2^{15}$ 或 32,768 bytes 的范围

Load/Store 指令格式 (I 型):



24_{10} + $s3 =

. . . 0001 1000
+ . . . 1001 0100
. . . 1010 1100 =
0x120040ac



指令：load 和 store 一个字节

```
lb   $t0, 1($s3)   #load byte from memory
sb   $t0, 6($s3)   #store byte to  memory
```

- 如何 load 和 store 一个字节?

  load byte places the byte from memory in the rightmost 8 bits of the destination register

  - what happens to the other bits in the register?

  store byte takes the byte from the rightmost 8 bits of a register and writes it to a byte in memory

  - what happens to the other bits in the memory word?

Example of Loading and Storing Bytes

- 给定指令序列、当前的存储器状态，大端方式存储：

```
add    $s3, $zero, $zero
lb     $t0, 1($s3)
sb     $t0, 6($s3)
```

Memory	
0x00000000	24
0x00000000	20
0x00000000	16
0x10000010	12
0x01000402	8
0xFFFFFFFF	4
0x009012A0	0
Data	Word Address (Decimal)

- What value is left in $t0?

  $t0 = 0x00000090

- What word is changed in Memory and to what?

  mem(4) = 0xFFFF90FF

- What if the machine was little Endian?

  $t0 = 0x00000012

  mem(4) = 0xFF12FFFF

MIPS算术运算指令

指令举例	操作	说明
add $1,$2,$3	$1=$2+$3	寄存器加法
sub $1,$2,$3	$1=$2-$3	寄存器减法
addi $1,$2,100	$1=$2+100	立即数加法
addu $1,$2,$3	$1=$2+$3	无符号数加法
subu $1,$2,$3	$1=$2-$3	无符号数减法
addiu $1,$2,100	$1=$2+100	无符号立即数加法
mfco $1,$epc	$1=$epc	读取异常 PC
mult $2,$3	Hi,Lo=$2×$3	乘法
multu $2,$3	Hi,Lo=$2×$3	无符号数乘法
div $2,$3	Lo=$2/$3, Hi=$2 mod $3	除法
divu $2,$3	Lo=$2/$3, Hi=$2 mod $3	无符号数除法
mfhi $1	$1=Hi	从 Hi 中取数据
mflo $1	$1=lo	从 lo 中取数据

MIPS 立即数(Immediate)指令 (I 型):

- 典型代码中常常会用到一些值不算大的常数
- 直接在指令中包含这些常数，速度比从存储器或寄存器中获得它们要快?

```
addi  $sp, $sp, 4      #$sp = $sp + 4

slti $t0, $s2, 15      #$t0 = 1 if $s2<15
```

- The constant is kept inside the instruction itself!

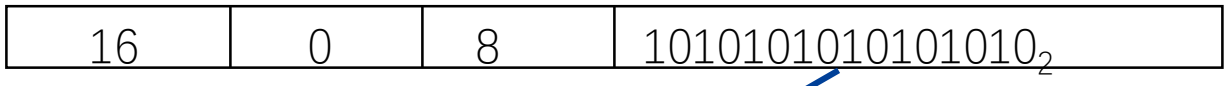
  立即数的范围： $+2^{15}-1$ to -2^{15}

what about upper 16 bits?

How About Larger Constants?

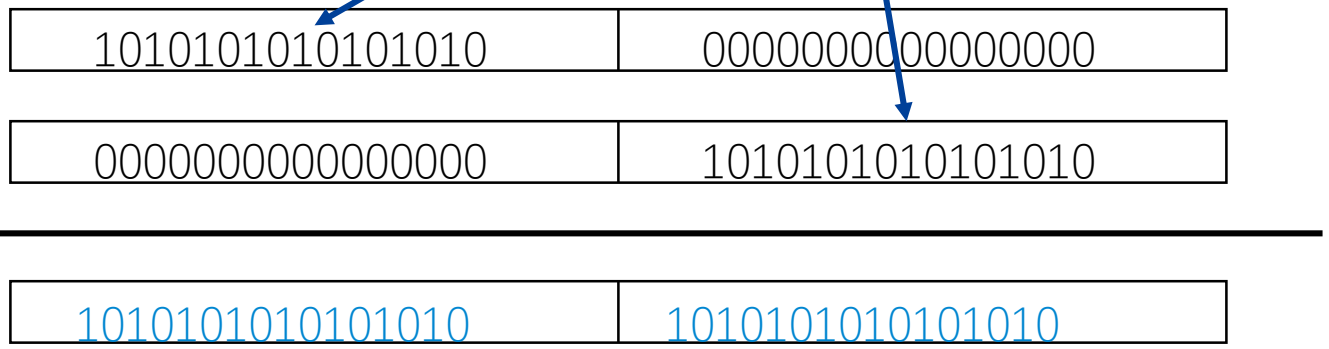
- 使用两条指令

```
lui $t0, 1010101010101010
```



-

```
ori $t0, $t0, 1010101010101010
```



why can't addi be used as the second instruction for this 32 bit constant?

MIPS Shift Operations

- Need operations to pack and unpack 8-bit characters into 32-bit words
- Shifts move all the bits in a word left or right

```
sll $t2, $s0, 8      #$t2 = $s0 << 8 bits
srl $t2, $s0, 8      #$t2 = $s0 >> 8 bits
```

- Instruction Format (R format)

- Such shifts are called logical because they fill with zeros

  Notice that a 5-bit shamt field is enough to shift a 32-bit value 2^5 – 1 or 31 bit positions

MIPS Logical Operations

- There are a number of bit-wise logical operations in the MIPS ISA

```
and $t0, $t1, $t2    #$t0 = $t1 & $t2

or  $t0, $t1, $t2    #$t0 = $t1 | $t2

nor $t0, $t1, $t2    #$t0 = not($t1 | $t2)
```

- Instruction Format (R format)


```
andi $t0, $t1, 0xFF00      #$t0 = $t1 & ff00

ori  $t0, $t1, 0xFF00      #$t0 = $t1 | ff00
```

- Instruction Format (I format)

MIPS逻辑运算指令

指令举例	操作	说明
and $1,$2,$3	$1=$2&$3	与操作
or $1,$2,$3	$1=$2 | $3	或操作
andi $1,$2,100	$1=$2&100	立即数与操作
ori $1,$2,100	$1=$2 | 100	立即数或操作
sll $1,$2,10	$1=$2<<10	逻辑左移
srl $1,$2,10	$1=$2>>10	逻辑右移

MIPS R格式指令扩展操作码定义

I5~I3 \ I2~I0	000	001	010	011	100	101	110	111
000	sll		srl	sra	sllv		srlv	srav
001	j $r	jalr			syscall	break		
010	mfhi	mthi	mflo	mtlo				
011	mult	multu	div	divu				
100	add	addu	sub	subu	and	or	xor	nor
101			slt	sltu				
110								
111								

MIPS Control Flow Instructions

- MIPS conditional branch instructions:

```
bne $s0, $s1, Lbl #go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl #go to Lbl if $s0=$s1
```

  - Ex:

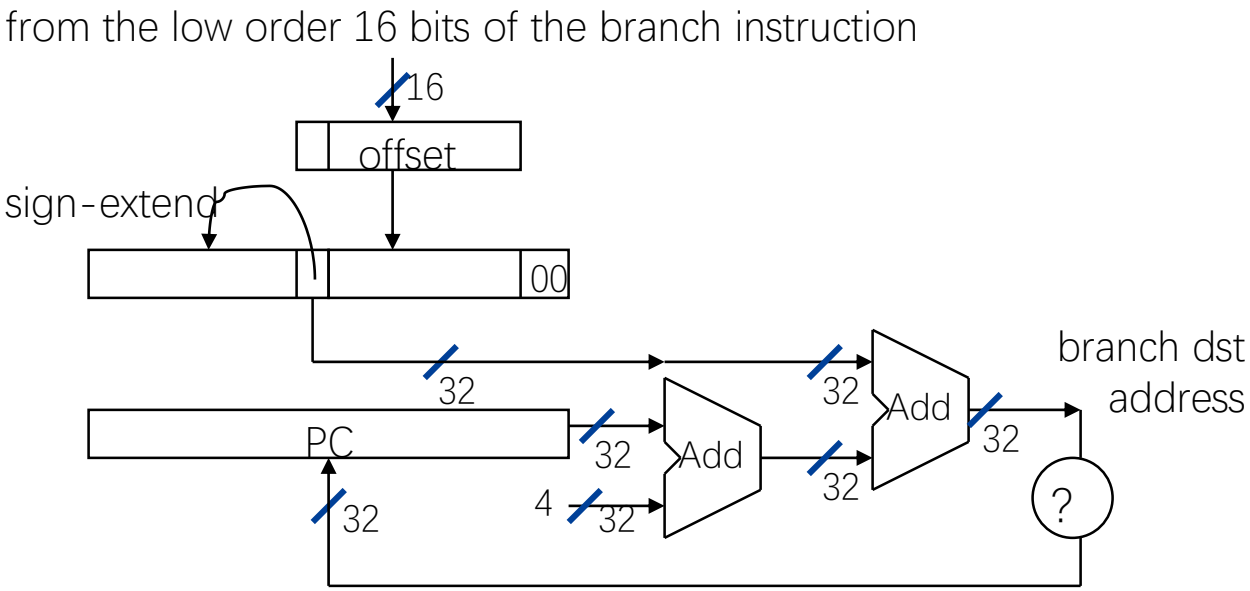
```
        if (i==j) h = i + j;

            bne $s0, $s1, Lbl1
            add $s3, $s0, $s1
Lbl1:       ...
```

- Instruction Format (I format):

0x05	16	17	16 bit offset

- How is the branch destination address specified?

Specifying Branch Destinations



- Use a register (like in lw and sw) added to the 16-bit offset
  - which register? Instruction Address Register (the PC)
    - its use is automatically implied by instruction
    - PC gets updated (PC+4) during the fetch cycle so that it holds the address of the next instruction
  - limits the branch distance to -2^{15} to $+2^{15}-1$ (word) instructions from the (instruction after the) branch instruction, but most branches are local anyway

In Support of Branch Instructions

- We have `beq`, `bne`, but what about other kinds of branches (e.g., branch-if-less-than)? For this, we need yet another instruction, `slt`
- Set on less than instruction:

```
slt $t0, $s0, $s1  # if $s0 < $s1          then
                           # $t0 = 1                  else
                           # $t0 = 0
```

- Instruction format (R format):

- Alternate versions of `slt`

```
slti $t0, $s0, 25   # if $s0 < 25 then $t0=1 ...

sltu $t0, $s0, $s1  # if $s0 < $s1 then $t0=1 ...

sltiu $t0, $s0, 25  # if $s0 < 25 then $t0=1 ...
```

Aside: More Branch Instructions

- Can use `slt`, `beq`, `bne`, and the fixed value of 0 in register `$zero` to create other conditions
  - less than `blt $s1, $s2, Label`

```
slt  $at, $s1, $s2      #$at set to 1 if
bne  $at, $zero, Label  #$s1 < $s2
```

  - less than or equal to `ble $s1, $s2, Label`
  - greater than `bgt $s1, $s2, Label`
  - great than or equal to `bge $s1, $s2, Label`

- Such branches are included in the instruction set as pseudo instructions - recognized (and expanded) by the assembler

  Its why the assembler needs a reserved register (`$at`)

MIPS Instructions

Consider a comparison instruction:

```
    slt   $t0, $t1, $zero
```

and $t1 contains the 32-bit number 1111 01…01

What gets stored in $t0?

MIPS Instructions

Consider a comparison instruction:

```
    slt   $t0, $t1, $zero
```

and $t1 contains the 32-bit number 1111 01…01

What gets stored in $t0?

The result depends on whether $t1 is a signed or unsigned number – the compiler/programmer must track this and accordingly use either slt or sltu

```
slt    $t0, $t1, $zero     stores  1 in $t0
sltu   $t0, $t1, $zero     stores  0 in $t0
```

Bounds Check Shortcut

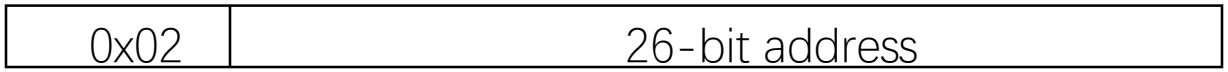
- Treating signed numbers as if they were unsigned gives a low cost way of checking if 0 ⩽ x < y (index out of bounds for arrays)

```
 sltu $t0, $s1, $t2  # $t0 = 0 if
                     # $s1 > $t2 (max)
                     # or $s1 < 0 (min)
beq $t0, $zero, IOOB # go to IOOB if
                     # $t0 = 0
```

- The key is that negative integers in two's complement look like large numbers in unsigned notation. Thus, an unsigned comparison of x < y also checks if x is negative as well as if x is less than y.

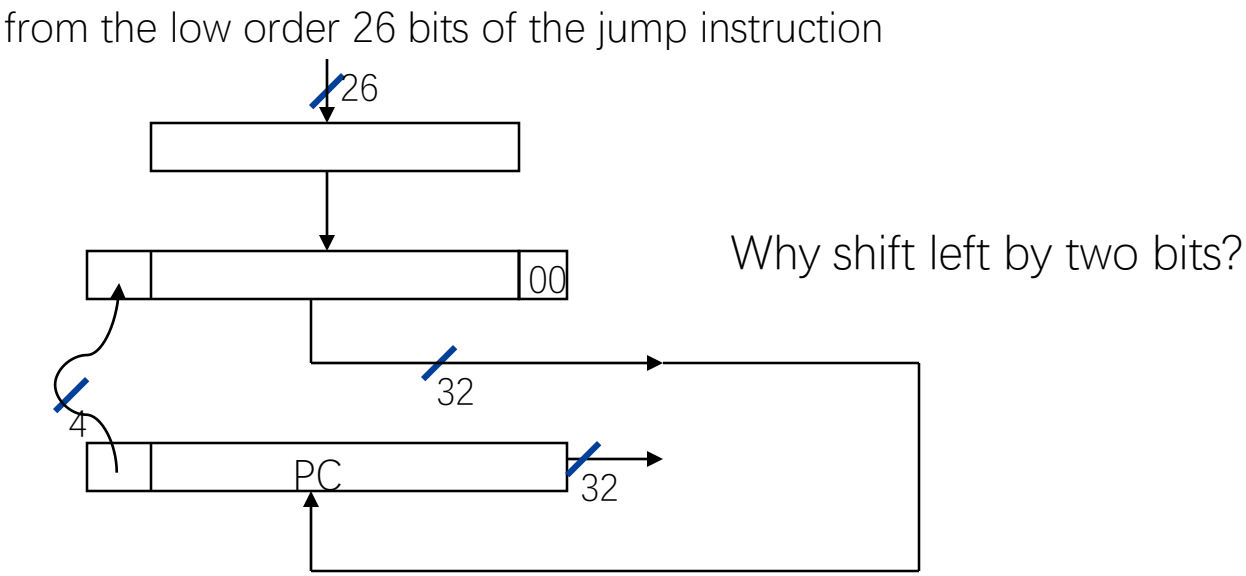
Other Control Flow Instructions

- Instruction Format (J Format):



- MIPS also has an unconditional branch instruction or jump instruction:

```
j   label          #go to label
```



Aside: Branching Far Away

- What if the branch destination is further away than can be captured in 16 bits?
  - The assembler comes to the rescue – it inserts an unconditional jump to the branch target and inverts the condition

```
        beq   $s0, $s1, L1
```

becomes

```
        bne   $s0, $s1, L2
        j     L1
    L2:
```

MIPS条件转移指令

指令举例	操作	说明
beq $1,$2,100	if($1==$2) go to PC+4+100	相等时转移
bne $1,$2,100	if($1!=$2) go to PC+4+100	不相等时转移
slt $1,$2,$3	if($2<$3) $1=1;else $1=0	小于时置位
slti $1,$2,100	if($2<100) $1=1;else $1=0	小于立即数时置位
sltu $1,$2,$3	if($2<$3) $1=1;else $1=0	小于无符号数时置位
sltiu $1,$2,100	if($2<100) $1=1;else $1=0	无符号数小于立即数时置位

MIPS无条件转移指令

指令举例	操作	说明
j 10000	go to 10000	转移到 10000
jr $31	go to $31	转移到$31
jal 10000	$31=PC+4; go to 10000	转移并链接

Compiling Another While Loop

- Compile the assembly code for the C `while` loop where i is in `$s3`, k is in `$s5`, and the base address of the array `save` is in `$s6`

```
while (save[i] == k)
     i += 1;
```

Compiling Another While Loop

- Compile the assembly code for the C `while` loop where i is in `$s3`, k is in `$s5`, and the base address of the array `save` is in `$s6`

```
while (save[i] == k)
     i += 1;
```

```
Loop:   sll  $t1, $s3, 2
        add  $t1, $t1, $s6
        lw   $t0, 0($t1)
        bne  $t0, $s5, Exit
        addi $s3, $s3, 1
        j    Loop
Exit:   . . .
```

思考题

- MIPS指令系统有哪些优点? 哪些缺点?

谢谢！