杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

□2□ MIPS体系结构与指令系统

第 2 章 MIPS 体系结构与指令系统

2.1 MIPS体系结构

2.2 MIPS指令系统

2.1 MIPS体系结构

- 1、概述
- 2、MIPS的内存映射
- 3、MIPS的异常

1、概述

- MIPS 是 Microprocessor without interlocked piped stages architecture （无内部互锁流水级的微处理器）的缩写。
- MIPS 架构是一种 **RISC 处理器架构**，当今最高的每平方毫米性能和 SoC 设计中最低的能耗。
- 指令系统经过通用处理器指令架构 MIPS I~MIPS V 和嵌入式指令体系结构 MIPS16、 MIPS32 到 MIPS64 的发展已经十分成熟。
- 向下兼容，比如， MIPS64 位扩展对 MIPS32 位的工作模式向下兼容。
- **MIPS32 体系结构以 MIPS II 指令集架构为基础**，选择性加入了 MIPS III、 MIPS IV、 MIPS V，提高了代码生成和数据移动的效率。

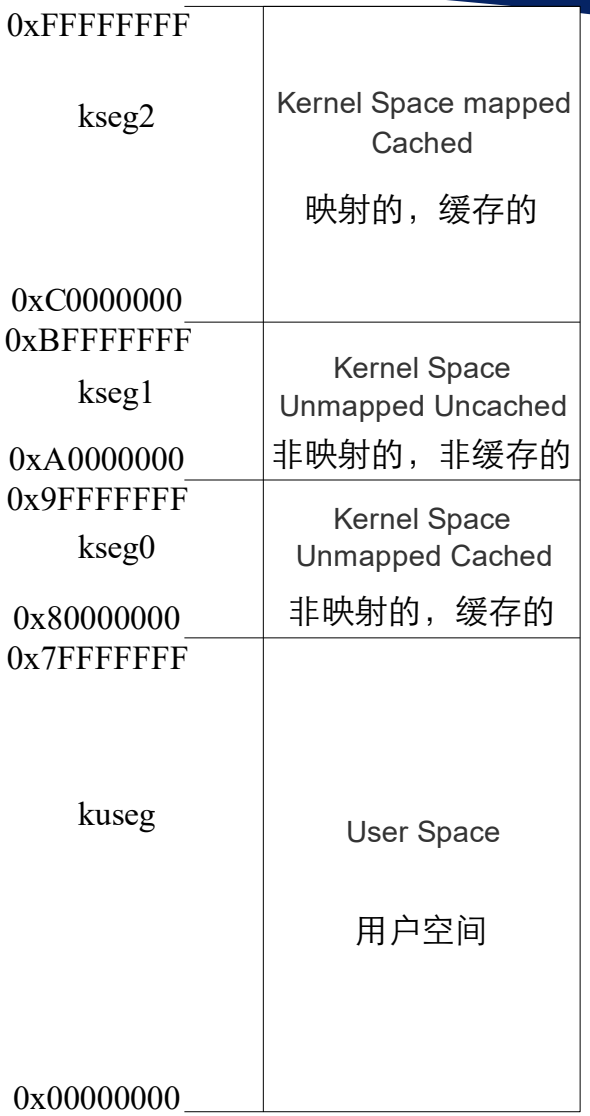
2、 MIPS 的内存映射

❖MIPS 的内存映射

- 程序中访问的地址，范围在系统提供给用户使用的地址空间中，称之为**虚拟地址**。
- **物理地址**是硬件结构中存储器的实际地址。
- 虚拟地址空间大于物理地址空间
- 内存管理单元 **MMU** 完成虚拟地址到物理地址的转换工作。

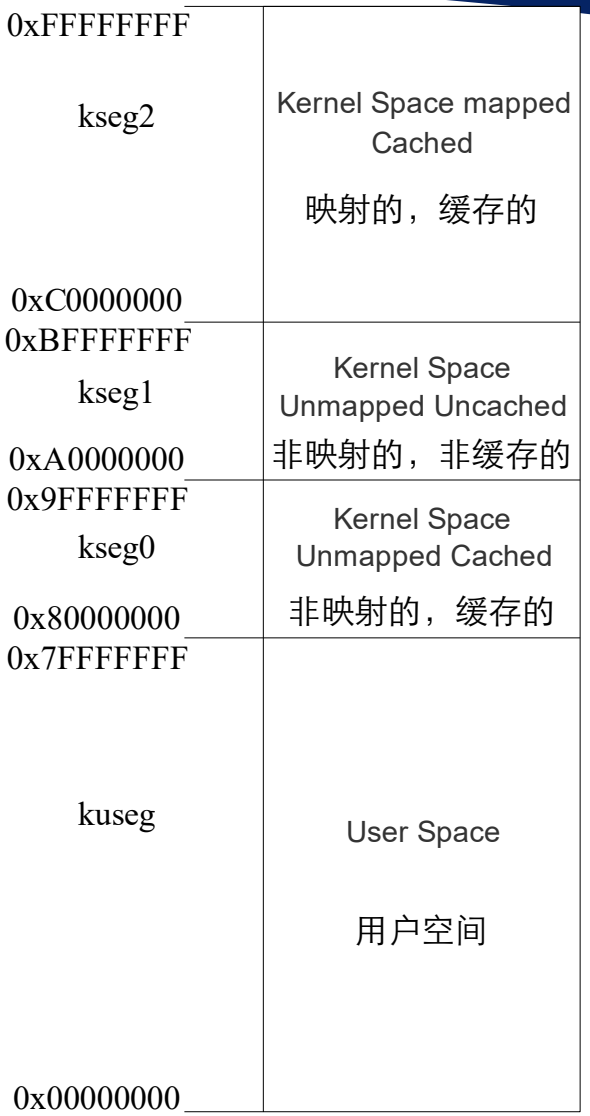
2、 MIPS 的内存映射

- MIPS 存储器按字节编址，大端模式（高地址放低字节），数据要求字边界对齐，只能通过 load/store 指令来访问存储器数据。
- kuseg:
  - （0x00000000-0x7FFFFFFF）虚拟地址的低 2GB 空间，是用户空间。系统上电时，kuseg 是不允许存取的，必须等到 MMU TLB 初始化之后才可以访问。



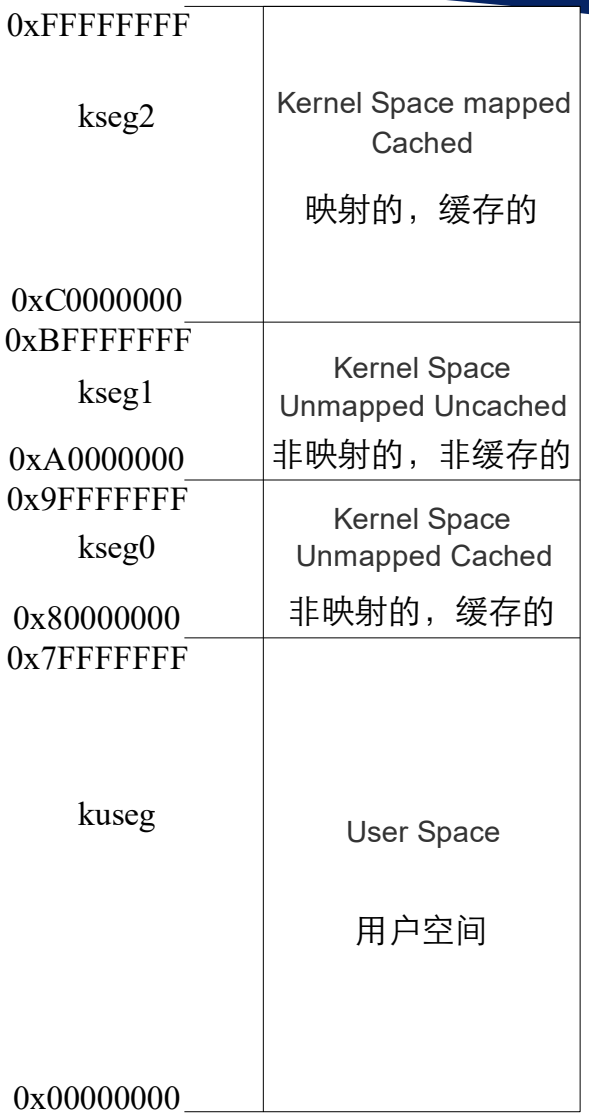
2、 MIPS 的内存映射

- **kseg0:**
  - **（ 0x80000000-0x9FFFFFFF ） 512MB ，仅限于内核模式（ Kernel Mode ） 可访问。**
  - **在没有 MMU 的系统中，该段空间用于存放大多数程序和数据。**
  - **在有 MMU 的系统中，该段空间存放操作系统内核，如内核代码段，或者内核中的堆栈。**



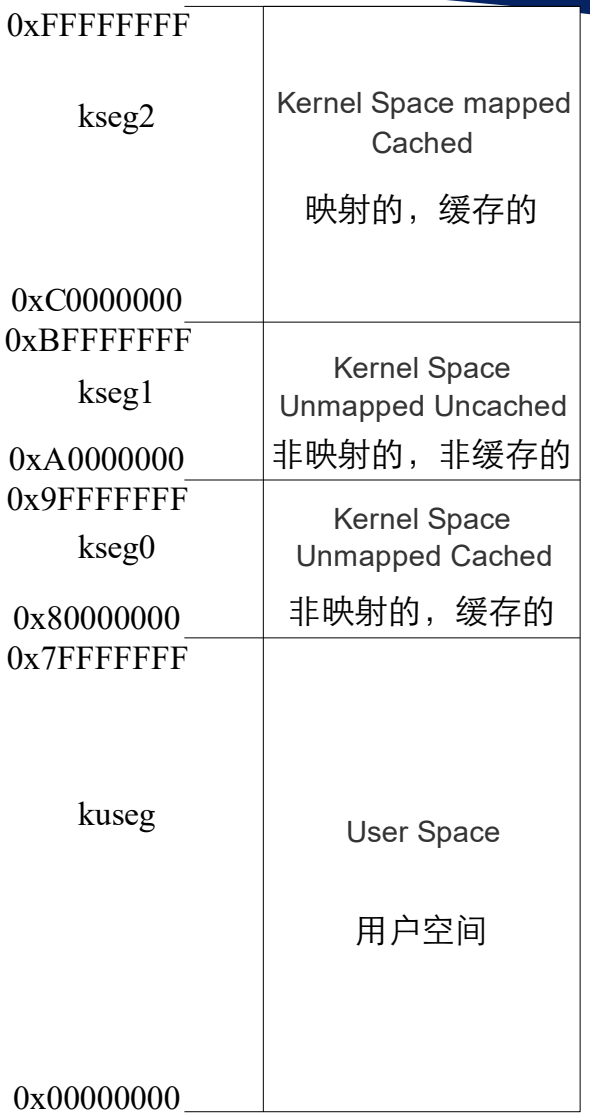
2、 MIPS 的内存映射

- kseg1:
  - （0xA0000000-0xBFFFFFFF） 512MB，仅限于内核模式可访问。
  - kseg0 和 kseg1 这两段空间逻辑地址到物理地址的映射关系都不通过 MMU，而是由硬件直接确定，且两段空间对应的物理空间重叠。区别是 kseg0 使用高速缓存， kseg1 不使用高速缓存，因此软件访问 kseg1 时速度比较慢，但是，对于硬件 I/O 寄存器来说，不存在 Cache 一致性问题。
  - 刚上电时， MMU 和 Cache 均未初始化，因为 kseg1 不使用高速缓存，所以 kseg1 是唯一在系统启动时能正常工作的内存映射空间。 MIPS 的程序上电启动地址即入口向量 0xBFC00000 位于 kseg1 内，入口向



2、 MIPS 的内存映射

- **kseg2:**
  - **（ 0xC0000000-0xFFFFFFFF ） 1GB ，仅限于内核模式可访问，为操作系统的内核所用。**
  - **逻辑地址通过 MMU 映射到物理地址。**
  - **有时候会看到在 MIPS 系统中 kseg2 被分成两等分，分别称为 kseg2 和 kseg3 ，两等分中的低半部分 kseg2 对于监管者模式可用。**



3、MIPS 的异常

- MIPS 系统里，中断、陷阱、系统调用和任何可以中断程序正常执行流的情况统称为“**异常**”。
- 当一个异常发生时总会有一条被异常打断的指令称为 "**异常受害者**" （exception victim).
- “**精确异常**”是指异常受害者前面的所有指令要执行完流水线的最后一个阶段；该指令以及其后的指令都要被取消。

3、MIPS 的异常

- **精确异常的处理：**
  1. 异常受害者和其后的指令被终止；
  2. 当异常受害者前的最后一条指令执行到流水线最后一个阶段，异常更新 CP0 的各个寄存器为异常状态；
  3. 程序计数器 PC 改变到相应的异常向量地址；
  4. 清除前面流水线的异常位。
- **MIPS 的异常和中断能够：**
  1. 提供一个合法地从用户模式到内核模式的切换通道，使得程序能够访问如 CP0、 KSeg 内存等内核模式才允许访问的资源；
  2. 处理一些非法的操作；
  3. 处理外部和内部的中断。与 IA32 架构区别的是，MIPS 所有的中断均来自 0 号 Exception 。

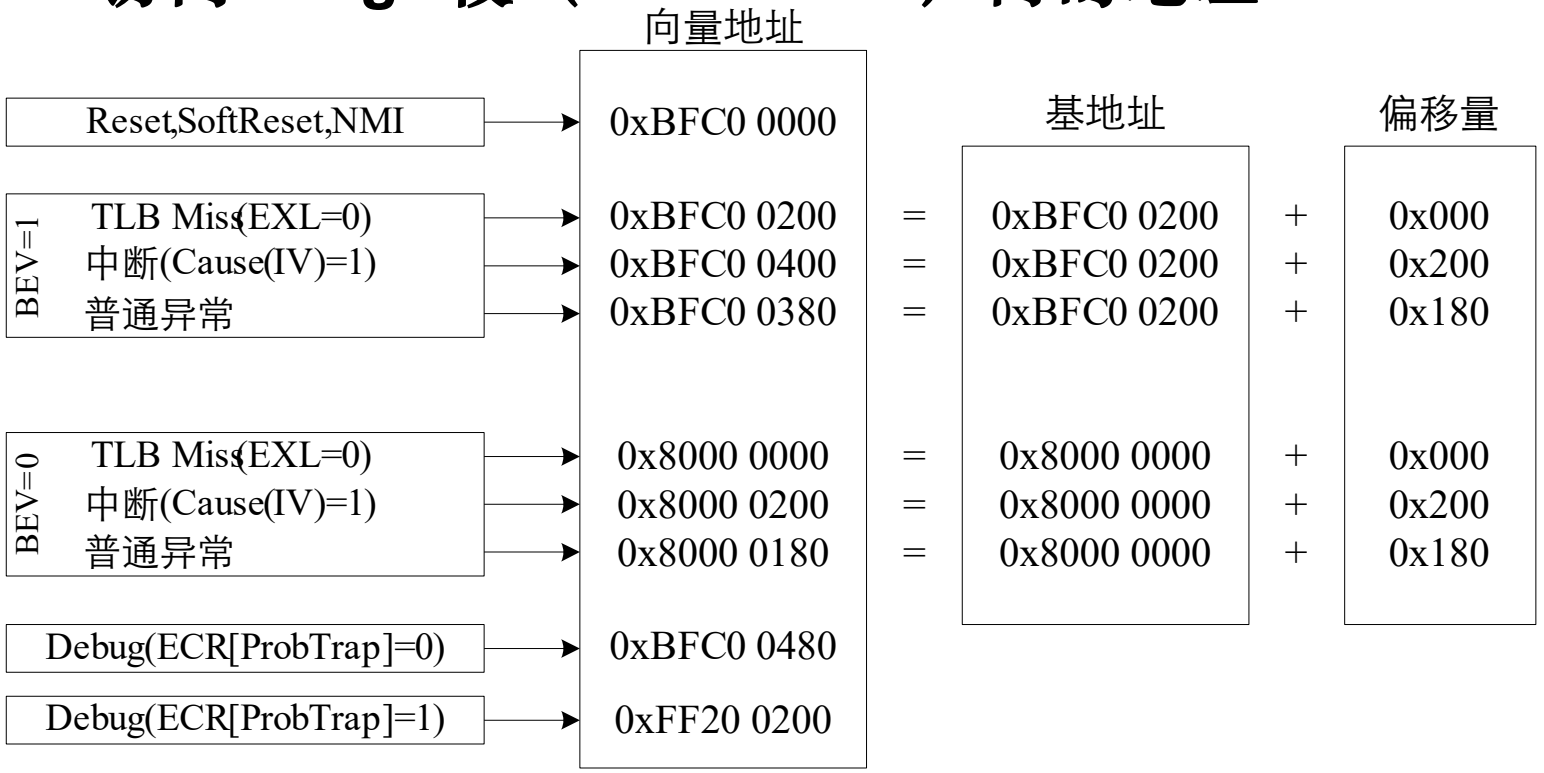
3、MIPS 的异常

- **异常发生时， MIPS 的处理**
  1. 保存现场寄存器组（ Register File ）
  2. 把当前 PC 保存在 EPC 或 ErrorEPC 中，以便异常处理结束后返回。
  3. 计算异常处理的入口向量地址。
  4. 进入内核模式，关中断。
  5. 按入口向量地址调用异常服务程序
  6. 返回
  7. 把 EPC 或 ErrorEPC 的值写回 PC ，从被打断的地方继续执行

3、MIPS 的异常

❖异常向量地址：

- BEV 表示异常发生时，系统是否正处于引导启动过程中。引导启动时， cache 未完成初始化，只能访问 kseg1 段（ uncached ）内的地址



3、MIPS 的异常

- MIPS 异常类型：
  - Exception 0-5, 8-11, 13, 23 较为常见
  - 0: Interrupt，外部中断
  - ……

- 1、MIPS32数据类型与寄存器
- 2、MIPS32

  指令格式、寻址方式和指令分类

1、 MIPS32 数据类型与寄存器

一． 数据类型

- 位（ bit ）、字节（ 8bit， Byte ）、半字（ 16bit ， Half word ）、字（ 32bit， word ）和双字（ 64bit， doulb word ）。
- 浮点处理单元 FPU 处理的数据类型有： 32 位单精度浮点数、 64 位双精度浮点数等。

□□MIPS32 的寄存器

- MIPS 寄存器分为三类：通用寄存器、专用寄存器和浮点寄存器。

1、MIPS32 数据类型与寄存器

① 通用寄存器

□□	□□□□□	□□
$0	$zero	□□
$1	$at	□□□□□□
$2$3	$v0-$v1	□□□□□□□□□□□□□□□□
$4-$7	$a0-$a3	□□□□□□
$8-$15	$t0-$t7	□□□□□□□□□□□□□□□□
$16-$23	$s0-$s7	□□□□□□□□□□□□□□□□□□□□□□□□

1、MIPS32 数据类型与寄存器

① 通用寄存器

□□	□□□□□	□□
$24-$25	$t8-$t9	□□□□□□□□□□□□□□□□□□□□
$26-$27	$k0-$k1	□□□□□□□□□□□□□□□□□□□□□□□□□□□□
$28	$gp	□□□□
$29	$sp	□□□□
$30	$fp	□□□
$31	$ra	□□□□□□

1、 MIPS32 数据类型与寄存器

② 专用寄存器

- 两个 32 位的乘商寄存器 hi 和 lo
- MIPS 中用程序计数器 PC 指出下一条指令的地址 . 因为 MIPS 存储器按字节编址，而每条指令固定长度是 32 位，因此， MIPS 取指后 PC 值自动加 4 ，指向存储器中的下一条指令。
- 注意区别 CR0 中的 Register 14 ： EPC 。 EPC 是异常程序计数器，当异常发生时，该寄存器保存系统正在执行的指令的地址

1、 MIPS32 数据类型与寄存器

③ 浮点寄存器

- MIPS 提供 32 个浮点寄存器： $f0,$f1,...,$f31 ，对于双精度数， MIPS 的方法是使用成对的浮点数寄存器。向浮点数寄存器读取或写入的指令是： lwcl 和 swcl 。例如：

```
lwcl        $f4,4($sp)
      # 取出一个 32 位浮点数到 f4 寄存器
add.s $f2,$f4,$f6    # 单精度浮点数加法运算 f2=f4+f6
swcl $f2,32($sp)     # 将 f2 中的单精度浮点数存入存储器
```

1、MIPS32 数据类型与寄存器

④ MIPS 协处理器中的 CP0 寄存器

通常在以下情况下会使用到 CP0 寄存器：

- **系统加电后，设置状态寄存器**来使 CPU 进入正确的引导状态。
- **处理异常。**任何 MIPS 异常（除特别的内存管理异常而外），都将调用一个入口地址固定的“通用异常处理程序”。通过 CP0 的异常原因寄存器找出异常的类型，再根据类型分别处理。
- **异常返回**。异常返回时，需要把状态寄存器设置回原来的值，恢复用户模式，开放中断使能，返回到异常程序计数器所指的地方。
- **中断**。状态寄存器用中断掩码来实现中断的优先级设置。硬件没有提供中断优先逻辑，完全由软件决定。

1、 MIPS32 数据类型与寄存器

④ MIPS 协处理器中的 CP0 寄存器

MIPS 提供了一些特殊的指令来对 CP0 操作，主要有：

mfc0 rt, rd　　# 该指令要求 rd 是 CP0 中的寄存器， rt 是用户模式下可用的 32 位通用寄存器。执行 mfc0 指令， (rd)→rt 。

mtc0 rt, rd　　# 该指令要求 rd 是 CP0 中的寄存器， rt 是用户模式下可用的 32 位通用寄存器。执行 mtc0 指令， (rt)→rd 。

1、 MIPS32 数据类型与寄存器

④ MIPS 协处理器 CP0 中的寄存器

MIPS 提供了一些特殊的指令来对 CP0 操作，主要有：

mfhi/mflo rt　　# 该指令要求 rt 是用户模式下可用的 32 位通用寄存器。 执行 mfhi 或 mflo 指令，(hi/lo)→rt 。

mthi/mtlo rt　　# 该指令要求 rt 是用户模式下可用的 32 位通用寄存器。 执行 mthi 或 mtlo 指令， (rt)→hi/lo 。

1、 MIPS32 数据类型与寄存器

④ MIPS 协处理器 CP0 中的寄存器

Register 12： Status，也写作 SR 寄存器，用于保存处理器的状态和处理器控制

内容包括 CU0~CU3(第 28~31 位)，复位向量 BEV，中断屏蔽位 8~15， KUc、 IEc0~1， KUp、 IEp2~3、 KUo、 IEo 。

- CU0~CU3(28~31) 用于设置协处理器 0~3 的可用性。
- KUc 为 1 时表示运行在内核模式，可以访问所有的地址空间和协处理器 0；
- KUc 为 0 时运行在用户模式。用户模式下只能访问 kuseg 的地址空间。

1、 MIPS32 数据类型与寄存器

④ MIPS 协处理器 CP0 中的寄存器

Register 12： Status，也写作 SR 寄存器，用于保存处理器的状态和处理器控制

- KUp、 IEp2~3、 KUo、 IEo 构成了深度为 2 的栈，异常发生时，硬件自动压栈， rfe 指令从异常返回时，从栈中恢复数值。具体操作是，当异常发生时，硬件把 KUp、 IEp 的值保存到 KUo、 IEo 中，把 KUc、 IEc 的值保存到 KUp、 IEp 中，并且将 KUc、 IEc 分别设置为 1 和 0，即内核模式和关中断。异常返回时 rfe 指令把 KUp、 IEp 的内容复制到 KUc、 IEc 中，把 KUo、 IEo 的内容复制到 KUp、 IEp 中。

1、 MIPS32 数据类型与寄存器

④ **MIPS 协处理器 CP0 中的寄存器**

Register 14： EPC，异常程序计数器。异常发生时，该寄存器保存系统正在执行的指令的地址。

2、 MIPS32 指令格式、寻址方式和指令分类

一. 指令格式：

- MIPS32 的每条**指令长度固定是 32 位； 3 种指令格式**：

□□□□	□□□□□□ 32 □□						□□
	31-26	25-21	20-16	15-11	10-6	5-0	
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)	□□□□
I	opcode (6)	rs (5)	rt (5)	offset/immediate (16)			□□□□□□□□□□ □□□□
J	opcode (6)	address (26)					□□□□

2、MIPS32 指令格式、寻址方式和指令分类

□□□	□□□□□□□					
	31..26	25..21	20..16	15..11	10..6	5..0
R □□□	op	rs	rt	rd	shamt	func

- R(register) 型指令，也叫 RR 型指令。该类型指令的源操作数和目的寄存器都是寄存器操作数。最高 6 位是操作码（opcode），第 25-21 位、第 20-16 位和第 15-11 位，连续三个 5 位二进制码来表示三个寄存器的地址，第 10-6 位表示移位的位数，如果未使用移位操作，则第 10-6 位置全 0，第 5-0 位为 6 位的功能码(function), 它与第 31-26 位的 opcode 码共同决定 R 型指令的具体操作方式。
- 算术指令都是 R 型指令。

2、MIPS32 指令格式、寻址方式和指令分类

□□□	□□□□□□□□			
	31..26	25..21	20..16	15..0
I □□□	op	rs	rt	immediate

- I(immediate) 型指令，是立即数型指令。该类型指令的一个源操作数是 16 位立即数，另两个操作数是寄存器操作数，因此也称为 R-I 型指令。最高 6 位是操作码（ opcode ），第 25-21 位和第 20-16 位，连续两个 5 位二进制码分别表示两个寄存器的地址，第 15-0 位是一个 16 位二进制码表示的立即数。指令执行时， 16 位立即数需要进行符号扩展或 0 扩展，变成 32 位操作数才能参与运算。数据传输、分支、立即数指令采用此类指令结构

2、MIPS32 指令格式、寻址方式和指令分类

□□□	□□□□□□□□	
	31..26	25..0
J	op	address

- J(jump) 型指令，是无条件转移指令。该类型指令使用一个 26 位的**直接地址**（第 25-0 位）。因为 MIPS 采用 32 位定长指令，所以每条指令都占 4 个存储单元，指令地址总是 4 的倍数。于是，只要**将当前 PC 值的高 4 位拼上 26 位直接地址，末尾两位置 0，即可得到 32 位的目标转移地址**。

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

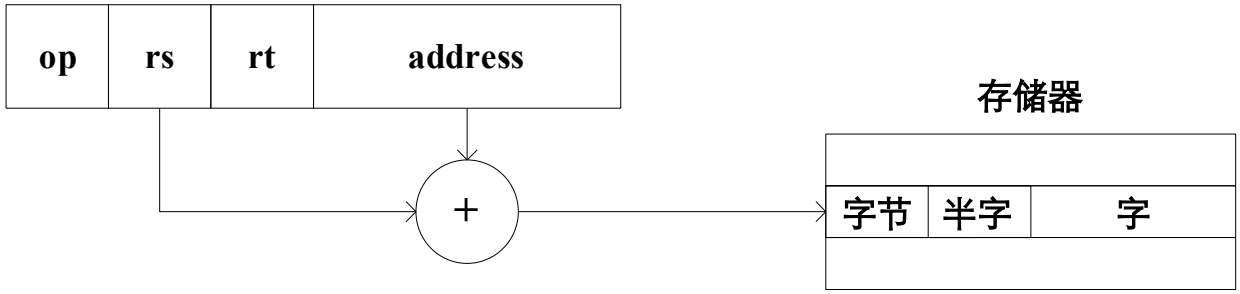
① 寄存器寻址，操作数是寄存器


```
or  $s1,$s2,$s3          # $s1=$s2 | $s3
```

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

② 基址寻址或偏移量寻址，操作数在存储器中，存储器地址是指令中基址寄存器和常数之和



例如：　　lw $t0,addr($t3)

如果 addr=0, 也可以写成　　lw $t0,($t3)

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

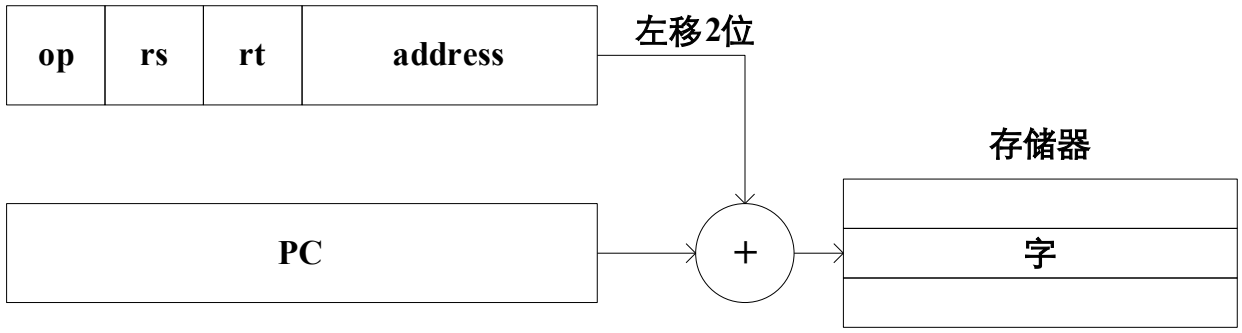
③ 立即数寻址，操作数是指令给出的常数

例：　addi $s1,$s2,100　　# $s1=$s2+100

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

④ 相对寻址，目标地址是当前 PC 值和指令中的常数之和， 16 位的常数在与 PC 相加之前要左移 2 位（相当于乘以 4 ）



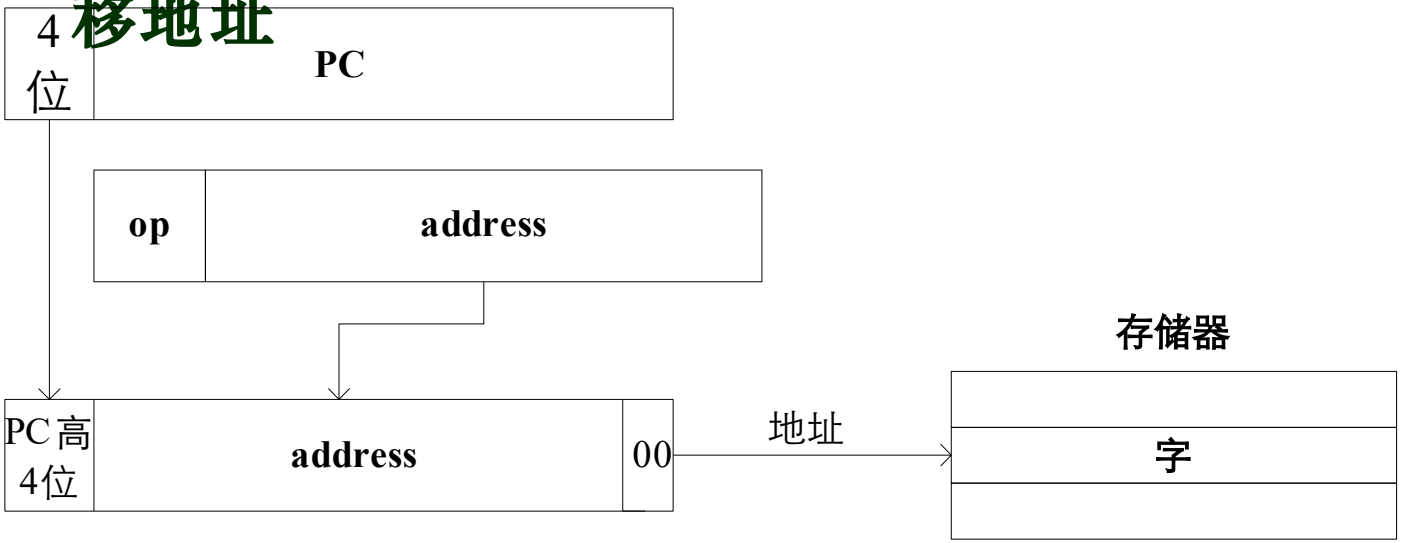
例： bne $s1,$s2,25　　# if $s1≠$s2 then goto 2

5*4+ 当前 pc 值（即 bne 指令的 PC+4)

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

⑤ 伪直接寻址，将当前 PC 值的高 4 位拼上 26 位直接地址，末尾两位置 0，即可得到 32 位的目标转移地址



2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

- 从 R、I、J 三种指令类型的角度讨论寻址方式
- R 型指令：三个操作数均是寄存器寻址方式。

例如：

```
or  $s1,$s2,$s3          # $s1=$s2 | $s3
```

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

- I 型指令：有寄存器寻址、立即数寻址、相对寻址、基址或偏移量寻址四种寻址方式。
  - ① 若 I 型指令是双目运算指令，则 rs 寄存器和立即数分别作为源操作数， rt 寄存器是目的操作数；

例如：

```
addi $sp,$sp,4    # $sp=$sp+4 ，立即数寻址
```

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

- ② 若 I 型指令是访存指令 load/store，则存储器地址由 rs 寄存器加上符号扩展的立即数得到，load 指令将存储内容读出到 rt 寄存器中，store 指令将 rt 寄存器写入存储器中。

例如：

```
lw $t0,32($s1)          # 假如 s1 里是数组 A 的起始地址，
```

存储器按字节编址，因此 $s1+32 指向 A[8]，连续读 4 个字节，读出 A[8] 的值置入 t0 寄存器。这是寄存器相对寻址（基址寻址）方式，s1 是基地址寄存器，用来装载数组的起始地址，常数 32 是偏移量，表示数组元素的下标值。

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

- ③ 若 I 型指令是**条件转移指令**，则对 rs 和 rt 寄存器的内容进行指定运算，根据结果决定是否跳转。**转移目标地址**等于当前 PC 值加上符号扩展后的立即数，这是**相对寻址**。偏移量可正可负。在对数组和堆栈寻址时，相对寻址比其他寻址方式更具优势。
- 例如：
- bne $s0,$s1,L0 # 如果 #$s0≠s1，则跳转到 L0 处。这是相对寻址。 PC=(PC)+16 位偏移量，请注意：与偏移量相加的 PC 值是 bne 指令的后一条指令的的 PC 值，即 bne 指令的 PC 值 +4 。

2、MIPS32 指令格式、寻址方式和指令分类

二．寻址方式

- **J 型指令：只有伪直接寻址这一种寻址方式。因为 J 型指令都是无条件转移指令，而 MIPS 系统采用 32 位定长指令，每条指令占 4 个存储单元，指令地址总是 4 的倍数。所以只要将当前 PC 值的高 4 位拼上 26 位直接地址，末尾两位置 0，即可得到 32 位的目标转移地址。**

2、 MIPS32 指令格式、寻址方式和指令分类

❖三、指令分类：

- 汇编指令助记符规则：以 d 开头表示 64 位版本指令，以“ u”结尾表示无符号数指令，以“ i”结尾表示立即数指令，以“ b”、“ w”和“ d”结尾，分别表示字节、字和双字操作
- 空操作指令：空操作指令有 nop 和 ssnop
- 数据传送指令
  - 寄存器间数据传送指令：
    - move： 伪指令，寄存器之间的传送
    - movf、 movt： 根据浮点条件标志，有条件地进行整数寄存器之间的传递；
    - movn、 movz： 根据另一个寄存器的状态，有条件地进行整数寄存器之间的传递

2、 MIPS32 指令格式、寻址方式和指令分类

- **数据传送指令**
  - **常数加载指令：** la 是获取某些标号地址或程序中变量地址的宏指令， li 是装入立即数常数的指令， lui 指令把 16 位立即数加载到寄存器的高位，寄存器低 16 位清 0 。
  - la $t1, var　　# 把 var 在主存中的地址装载到 t1 寄存器中， var 也可以是程序中定义的一个子程序标签的地址。
  - li $t1, 5　　# $t1 = 5， 5 是立即数
  - lui $t0,255

lui $t0,255对应的机器指令

001111	00000	01000	0000 0000 1111 1111

指令执行后，
$t0的内容

0000 0000 1111 1111	0000 0000 0000 0000

2、 MIPS32 指令格式、寻址方式和指令分类

- **算术 / 逻辑运算指令**
  - 加减运算指令 add、 addu、 addiu、 subu、 sub
  - 混合算术运算指令：取绝对值指令 abs，取反指令 negu、 neg
  - 按位逻辑指令： and、 andi、 or、 ori、 xor、 xori、 nor、 not
  - 循环和移位指令： rol、 ror、 sll、 sllv、 srl、 srlv、 sra、 srav
  - 条件设置指令： slt、 slti、 sltiu、 sltu：硬件指令，如果条件满足则写入 1，否则写入 0， seq、 sge、 sgeu、 sgt、 sgtu、 sle、 slue、 sne 根据更复杂的条件设置目的寄存器
  - 整数乘除法及求余指令： div、 divu、 divo、 divou；mul、 mulo、 mulou、 mult、 multu、 rem、 remu、

2、 MIPS32 指令格式、寻址方式和指令分类

- **访存指令：**
  - lw、 lb （寄存器高位填入符号位）、 lbu （寄存器高位填入 0 ）、 lh （寄存器高位填入符号位）、 lhu （寄存器高位填入 0 ）、把数据预取到缓冲的指令 pref、 sb、 sh、 sw
  - 浮点存取指令：存取单 / 双精度浮点数的指令 l.s 、 l.d 、 s.s 、 s.d

2、MIPS32 指令格式、寻址方式和指令分类

- **程序控制类指令**
  - **分支指令：与 PC 相关的跳转指令称为“分支指令”，以 b 开头**

```
b target   #  基于当前指令地址的无条件相对跳转，跳转范围比较短
beq $t0,$t1,target          #  branch to target if  $t0 = $t1
blt $t0,$t1,target  #  branch to target if  $t0 < $t1
ble $t0,$t1,target #  branch to target if  $t0 ≤ $t1
bgt $t0,$t1,target #  branch to target if  $t0 > $t1
bge $t0,$t1,target          #  branch to target if  $t0 ≥ $t1
```

2、 MIPS32 指令格式、寻址方式和指令分类

- **程序控制类指令**
  - **跳转指令 (Jumps) ： 绝对地址的跳转称为“跳转指令”，跳转指令以 j 开头。**

    j 指令是无条件跳转到一个绝对地址，访问 256M 代码空间。

    jr 指令是跳转到某寄存器指示的地址处。

    例如：

```
j target          #  无条件跳转到标号 target 处
jr $t3            #  跳转目标地址在 $t3 寄存器中
```

2、 MIPS32 指令格式、寻址方式和指令分类

- **子程序调用指令**
  - 子程序调用称为“跳转并链接”或“分支并链接”，指令以 ...al 结尾。

    Jal/jalr 是直接 / 间接的子程序调用，跳转到指定地址，并把返回地址放到 $ra 寄存器中，返回地址是当前指令地址 +8 。这是因为紧跟在调用指令 jal 后面立即执行的指令称做调用的延迟槽，返回地址应该是延迟槽指令后面的那条指令。

2、 MIPS32 指令格式、寻址方式和指令分类

- **子程序调用指令**

调用子程序：

jal sub_label　　#　把返回地址（返回地址是当前指令地址 +8，即当前指令的下下条指令的地址）放到 $ra 寄存器中，然后跳转子程序， sub_label 是子程序的标号

例：返回地址是第三行的指令，而不是 jal 指令后面的 move 指令

```
    jal    printf                # 返回地址是 xxx 指令的地址
    move   $4,$6
    xxx                          # 子程序返回到该指令
```

2、MIPS32 指令格式、寻址方式和指令分类

- **子程序调用指令**

  子程序调用返回：

  jr $ra　　#　跳转到 $ra 寄存器指示的地址处

  ，$ra 中是调用子程序之前保存的主程序返回地址。

  注意，返回地址存放在 $ra 寄存器中。如果子程序调用了下一级子程序，或者是递归调用，那么需要将返回地址保存在堆栈中，因为每执行一次 jal 指令就会覆盖 $ra 中的返回地址。

2、 MIPS32 指令格式、寻址方式和指令分类

- **断点及陷阱指令**
  - break 指令产生“断点”类型的异常，可以在宏扩展指令中产生陷阱或用于调试器。
  - sdbbp 指令产生 EJTAG 异常的断点指令。
  - syscall 指令产生一个约定用于系统调用的异常类型。
  - teq、 teqi、 tge、 tgei、 tgeiu、 tgeu、 tlt、 tlti、 tltiu、 tltu、 tne、 tnei 是条件异常指令，对一个或两个操作数进行条件测试，条件满足则触发异常。

2、 MIPS32 指令格式、寻址方式和指令分类

- **协处理器 0 的功能指令**
  - 协处理器与通用寄存器之间的双向数据传递
    - cfc0 、 ctc0 是把数据拷进 / 拷出协处理器 0 的控制寄存器的指令。
    - mfc0 、 mtc0 是在通用寄存器和协处理器 0 寄存器之间交换数据的指令。
  - 用于 CPU 控制的特殊指令 :eret 是异常返回指令。

2、MIPS32 指令格式、寻址方式和指令分类

- 浮点操作指令支持 IEEE754 的单精度和双精度格式。
  - 加法： add.s(单精度） , add.d （双精度）
  - 减法： sub.s, sub.d
  - 乘法： mul.s, mul.d, 除法： div.s, div.d
  - 比较： bclt （条件为真跳转）， bclf （条件为假跳转）， c.x.s, c.x.d ，其中 x 可能为相等 (eq) ，不等 (neq) ，小于 (lt) ，小于等于 (le) ，大于 (gt) ，大于等于 (ge)
  - 条件转移： bclt(条件为真跳转） , bclf （条件为假跳转）
  - 数据传送： lwcl(读存储器字到浮点寄存器)， swcl (把浮点寄存器内容写入存储器) 。

■用户模式下对“底层”硬件的有限访问指令

2、 MIPS32 指令格式、寻址方式和指令分类

四. 常用指令列表及指令编码

表 2-5 MIPS 指令译码表-1

OP(31:26)								
28-26 / 31-29	000	001	010	011	100	101	110	111
000	R 型指令	bltz/gez	j	jal	beq	bne	blez	bgtz
001	addi	addiu	slti	sltiu	andi	ori	xori	lui
010	TLB	FLPt						
011								
100	lb	lh	lwl	lw	lbu	lhu	lwr	
101	sb	sh	swl	sw				
110	lwc0	lwc1						
111	swc0	swc1						

2、 MIPS32 指令格式、寻址方式和指令分类

四. 常用指令列表及指令编码

表 2-6 MIPS 指令译码表-2

OP(31:26）=010000（TLB), rs(25:21)								
23-21 / 25-24	000	001	010	011	100	101	110	111
000	mfc0		cfc0		mtc0		ctc0	
001								
010								
011								
100								
101								
110								
111								

2、MIPS32 指令格式、寻址方式和指令分类

四．常用指令列表及指令编码

表 2-7 MIPS 指令译码表-3

OP(31:26）=000000（R 型指令），func(5:0)								
2-0 / 5-3	000	001	010	011	100	101	110	111
000	sll		srl	sra	sllv		srlv	srav
001	jr	jalr			sysycall	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								

MIPS32 核心指令集

第第第	第第第第 （位）						第第	第第第第	第第第第第第
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
add	0	rs	rt	rd	0	100000	add $s1,$s2,$s3	$s1=$s2+$s3	rd←rs+rt；其中 rs = $s2， rt=$s3,rd=$s1，有符号数加
addu	0	rs	rt	rd	0	100001	addu $s1,$s2,$s3	$s1=$s2+$s3	rd←rs+rt；其中 rs = $s2， rt=$s3,rd=$s1，无符号数加
sub	0	rs	rt	rd	0	100010	sub $s1,$s2,$s3	$s1=$s2-$s3	rd←rs-rt；其中 rs = $s2， rt=$s3,rd=$s1，有符号数减
subu	0	rs	rt	rd	0	100011	subu $s1,$s2,$s3	$s1=$s2-$s3	rd←rs-rt；其中 rs = $s2， rt=$s3,rd=$s1，无符号数减
and	0	rs	rt	rd	0	100100	and $s1,$s2,$s3	$s1=$s2&$s3	rd←rs&rt；其中 rs = $s2， rt=$s3,rd=$s1，按位与运算

MIPS32 核心指令集

□□□	□□□□（位）						□□	□□□□	□□□□□□
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
or	0	rs	rt	rd	0	100101	or $s1,$s2,$s3	$s1=$s2|$s3	rd←rs|rt；其中 rs = $s2， rt=$s3,rd=$s1，按位或运算
xor	0	rs	rt	rd	0	100110	xor $s1,$s2,$s3	$s1=$s2^$s3	rd←rs xor rt；其中 rs = $s2， rt=$s3,rd=$s1，按位异或运算
nor	0	rs	rt	rd	0	100111	nor $s1,$s2,$s3	$s1=~($s2|$s3)	rd←not(rs|rt)；其中 rs = $s2， rt=$s3,rd=$s1，按位或非运算
slt	0	rs	rt	rd	0	101010	slt $s1,$s2,$s3	if($s2<$s3) $s1=1 else$s1=0	if(rs<rt)rd=1 else rd=0；其中 rs = $s2， rt=$s3,rd=$s1（有符号数）
sltu	0	rs	rt	rd	0	101011	sltu $s1,$s2,$s3	if($s2<$s3) $s1=1 else$s1=0	if(rs<rt)rd=1 else rd=0；其中 rs = $s2， rt=$s3,rd=$s1（无符号数）

MIPS32 核心指令集

□□□	□□□□（位）						□□	□□□□	□□□□□□□
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
sll	0	0	rt	rd	shamt	000000	sll $s1,$s2,10	$s1=$s2<<10	rd←rt<<shamt；逻辑左移。 shamt 中存放移位的位数，即□□□□□□ $s2 rd=$s1
srl	0	0	rt	rd	shamt	000010	srl $s1,$s2,10	$s1=$s2>>10	rd←rt>>shamt；逻辑右移，其中 rt=$s2,rd=$s1
sra	0	0	rt	rd	shamt	000011	sra $s1,$s2,10	$s1=$s2>>10	rd←rt>>shamt；立即数是移位次数，算术右移，保留符号位。□□ $s2 rd=$s1
sllv	0	rs	rt	rd	0	000100	sllv $s1,$s2,$s3	$s1=$s2<<$s3	rd←rt<<rs；逻辑左移，其中 rs = $s3， rt=$s2,rd=$s1
srlv	0	rs	rt	rd	0	000110	srlv $s1,$s2,$s3	$s1=$s2>>$s3	rd←rt>>rs；逻辑右移，其中 rs = $s3， rt=$s2,rd=$s1

MIPS32 核心指令集

□□□	□□□□（位）						□□	□□□□	□□□□□□
	31..26	25..21	20..16	15..11	10..6	5..0			
R 型指令	op	rs	rt	rd	shamt	func			
srav	0	rs	rt	rd	0	111	srav $s1,$s2,$s3	$s1=$s2>>$s3	rd←rt>>rs；移位次数存于寄存器中，算术右移，保留符号位。□□ = $s3 rt=$s2 rd=$s1
jr	0	rs	0	0	0	001000	jr $s31	go to $ra	PC←rs, □□ = $s31

□□□	□□□□□□				□□	□□	□□□□□□
	31..26	25..21	20..16	15..0			
	op	rs	rt	immediate			
addi	001000	rs	rt	immediate	addi $s1,$s2,100	$s1=$s2+100	rt←rs+ □□□ rt=$s1,rs=$s2
addiu	001001	rs	rt	immediate	addiu $s1,$s2,100	$s1=$s2+100	rt←rs+ □□□ rt=$s1,rs=$s2
andi	001100	rs	rt	immediate	andi $s1,$s2,10	$s1=$s2&10	rt←rs& □□□ rt=$s1,rs=$s2
ori	001101	rs	rt	immediate	andi $s1,$s2,10	$s1=$s2|10	rt←rs| □□□ rt=$s1,rs=$s2
xori	001110	rs	rt	immediate	andi $s1,$s2,10	$s1=$s2^10	rt←rs xor □□□ rt=$s1,rs=$s2

□□□	□□□□□□				□□	□□	□□□□□□
	31..26	25..21	20..16	15..0			
	op	rs	rt	immediate			
lui	001111	0	rt	immediate	lui $s1,100	$s1=100*65536	rt←immediate*65536 □□ 16 □□ □□□□□□□□□□ 16 □□ 0
lw	100011	rs	rt	immediate	lw $s1,10($s2)	$s1=memory[$s2+10]	rt←memory[rs+(sign-extend)immediate] □ rt=$s1,rs=$s2
sw	101011	rs	rt	immediate	sw $s1,10($s2)	memory[$s2+10]=$s1	memory[rs+ □ rt=$s1,rs=$s2
bne	000101	rs	rt	immediate	bne $s1,$s2,10	if($s1!=$s2) goto PC+4+40	if(rs!=rt)PC←PC+4+
slti	001010	rs	rt	immediate	slti $s1,$s2,10	if($s2<10) $s1=1 else $s1=0	if(rs< □□□□ $s2 □ rt=$s1
sltiu	001011	rs	rt	immediate	sltiu $s1,$s2,10	if($s2<10) $s1=1 else $s1=0	if(rs< □□□□ $s2 □ rt=$s1

MIPS32 核心指令集

□□□	□□□□□□□		□□	□□□□	□□□□□□
	31..26	25..0			
J □ □□	op	address			
j	000010	address	j 10000	goto 10000	PC←(PC+4) [31..28],address,00 □ address=10000/4
jal	000011	address	jal 10000	$s31←PC+4 ; goto 10000	$s31←PC+4 □ PC←(PC+4) [31..28],address,00

MIPS32 核心指令集

□□□□	□□	□□	□□□□□	
			op31..26	Address25..0
j 10000	jump to 2500*4	□□□	2	2500
		□□□□	000010	00 0000 0000 0000 1001 1100 0100
jal 10000	$s31=PC+4	□□□	3	2500
		□□□□	000011	00 0000 0000 0000 1001 1100 0100

使用 MIPS 指令解决问题（实现编译）

- 例 1： a=b+c+d+e
- 使用 MIPS 汇编指令实现：
  - add a,b,c　#a=b+c
  - add a,a,d　#a=b+c+d
  - add a,a,e　#a=b+c+d+e
- 编译器还需要为变量分配寄存器：
  - 譬如选择寄存器 s0、 s1、 s2、 s3 分配给变量 a、 b、 c、 d，则编译成的 MIPS 代码即为：
    - add $s0,$s1,$s2　# $s0 = $s1+$s2= b+c
    - add $s0,$s0,$s3　# $s0 = $s0+$s3 =b+c+d
    - add $s0,$s0,$s4　# $s0 = $s0+$s4 =b+c+d+e

使用 MIPS 指令解决问题（实现编译）

- 例 2： f=(g+h)-(i+j)
- 使用 MIPS 汇编指令实现，还需要临时变量：
  - add t0,g,h　　#t0=g+h
  - add t1,i,j　　#t1=i+j
  - sub f,t0,t1　　#f=t0-t1=(g+h)-(i+j)
- 编译器为变量分配寄存器，且选择 2 个寄存器存放临时变量 t0 、 t1 ：
  - f~j:$S0~$S4;　　t0:$t0;　　t1:$t1
  - add $t0,$s1,$s2　　# $t0 = $s1+$s2= g+h
  - add $t1,$s3,$s4　　# $t1 = $s3+$s4 =i+j
  - sub $s0, $t0,$t1　　# $s0 = $t0-$t1 =(g+h)-(i+j)

使用 MIPS 指令解决问题（实现编译）

- MIPS 指令系统的指令特征 1：
- ***算术运算类指令，均是三操作数指令，且只能是寄存器类型的操作数或者立即数类型操作数。***
- 例 3： k=k+4;
  - addi $s3, $s3, 4
- ***寄存器类型和立即数类型（常数）操作数，比从存储器中取操作数要快得多。***
- 例 4： w=100;
  - 为变量 w 分配寄存器 $s0:
  - addi $s0, $zero,100; #$s0= 0+100=100, 完成了***常数到寄存器的传送操作***，作为伪指令 move $s0,

使用 MIPS 指令解决问题（实现编译）

每个字 32 位，占 4 个地址

- 例 5： x=y+A[8];
  - A 是含有 100 个字的数组，存放在存储器中，其起始地址（基地址 / 数组元素 0 的地址）存放在 $s2 中；则首先需要取数，再运算。设 x、 y: $s0、 $s1
    - lw $t0, 32($s2) #$t0=A[8]
    - add $s0,$s1, $t0 #$s0=$s1+$t0; 即 x=y+A[8]

- MIPS 指令系统的指令特征 2：
- 对于***存储器类型的操作数，只能***通过***取数（lw）、存数指令（sw）访问***，不能直接参加运算。

使用 MIPS 指令解决问题（实现编译）

- 例 6： A[12]=y+A[8];
  - 数组 A 的基地址存放在 $s2 中，为 y 分配寄存器 $s1;
  - lw $t0,32($s2) #$t0=A[8]
  - add $t0,$s1,$t0 #$t0=$s1+$t0; 即 $t0=y+A[8]
  - sw $t0,48($s2) # A[12]=y+A[8];

MIPS 的机器指令编码

- 3 种格式： R、 I、 J

汇编指令	格式	机器指令码
	R	op rs rt rd shamt func
add $s0,$s1,$s2	R	000000 10001 10010 10000 00000 100000
sub $s0, $t0,$t1	R	000000 01000 01001 10000 00000 100010
sll $s0,$s1,10	R	000000 00000 10001 10000 01010 000000
	I	op rs rt imme
addi $s0, $zero, 100;	I	001000 00000 10000 0000000001100100
lw $t0, 32($s2)	I	100011 10010 01000 0000000000100000
sw $t0, 48($s2)	I	101011 10010 01000 0000000000110000

The End!