计算机组织与体系结构

Computer Architectures

陆俊林

2024年春季学期

第四讲 RISC和MIPS指令（2）

本讲要点

首先继续讲解主要的MIPS指令，其次分析一个MIPS汇编程序示例，然后对比分析常见的若干种指令系统，最后简要介绍ARM指令系统的特点。

阅读教材“COD”：第2章，附录E

主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

I MIPS指令分类说明：分支

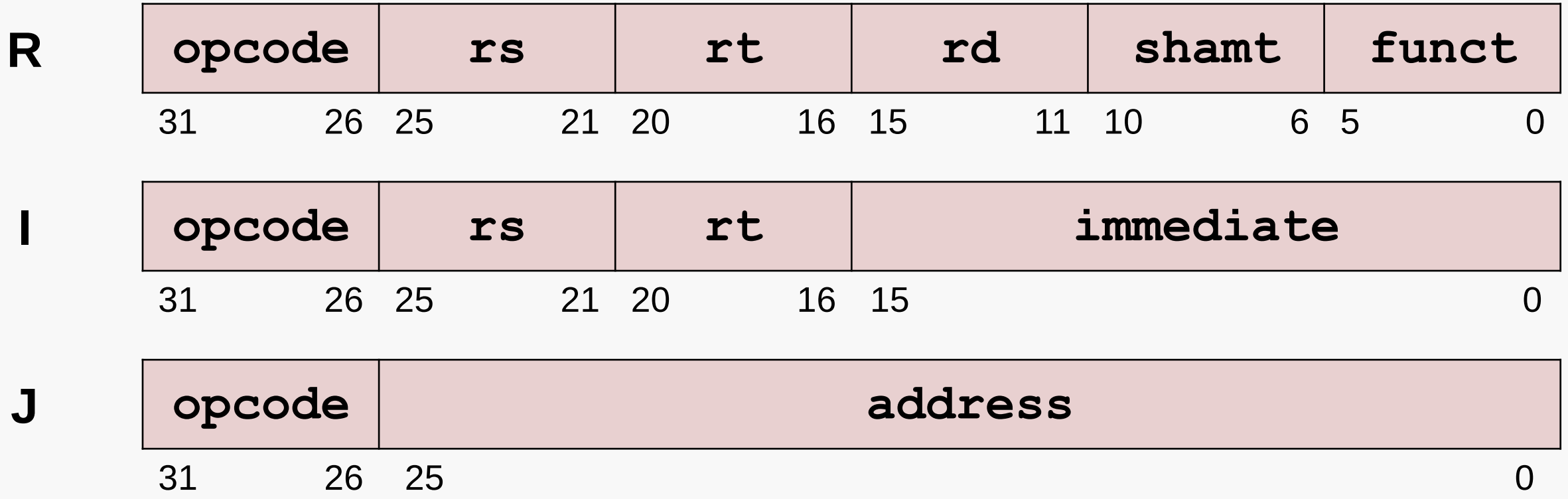
II MIPS汇编程序示例

III 指令系统概览

IV 指令系统举例

MIPS指令的基本格式

- R：Register，寄存器
- I：Immediate，立即数
- J：Jump，无条件转移



不同维度的指令分类（示例）

运算指令	add rd,rs,rt sll rd,rt,*shamt*	addi rt,rs,*imm* slti rt,rs,*imm*	/
访存指令	/	lw rt,*imm*(rs) sw rt,*imm*(rs)	/
分支指令	jr rs	beq rs,rt,*imm*	j *addr*
	R型指令	I型指令	J型指令

分支指令的分类

- **Branch**
  - 分支：改变控制流
- **Conditional Branch**
  - 条件分支：根据比较的结果改变控制流
  - 两条指令：branch *if* equal (`beq`)；branch *if not* equal (`bne`)
- **Unconditional Branch**
  - 非条件分支：无条件地改变控制流
  - 一条指令： jump (`j`)

对比：x86的转移指令

分组	格式	功能	测试条件
无条件转移指令	JMP LABEL	无条件转移	
	CALL LABEL	过程调用	
	RET	过程返回	

分组		格式	功能	测试条件
条件转移指令	根据某一状态标志转移	JC LABEL	有进位时转移	CF=1
		JNC LABEL	无进位时转移	CF=0
		JP/JPE LABEL	奇偶位为1时转移	PF=1
		JNP/JPO LABEL	奇偶位为0时转移	PF=0
		JZ/JE LABEL	为零/相等时转移	ZF=1
		JNZ/JNE LABEL	不为零/不相等时转移	ZF=0
		JS LABEL	负数时转移	SF=1
		JNS LABEL	正数时转移	SF=0
		JO LABEL	溢出时转移	OF=1
		JNO LABEL	无溢出时转移	OF=0

分组		格式	功能	测试条件
条件转移指令	对无符号数	JB/JNAE LABEL	低于/不高于等于时转移	CF=1
		JNB/JAE LABEL	不低于/高于等于时转移	CF=0
		JA/JNBE LABEL	高于/不低于等于时转移	CF=0且ZF=0
		JNA/JBE LABEL	不高于/低于等于时转移	CF=1或ZF=1
	对有符号数	JL/JNGE LABEL	小于/不大于等于时转移	SF≠OF
		JNL/JGE LABEL	不小于/大于等于时转移	SF=OF
		JG/JNLE LABEL	大于/不小于等于时转移	ZF=0且SF=OF
		JNG/JLE LABEL	不大于/小于等于时转移	ZF=1或SF≠OF

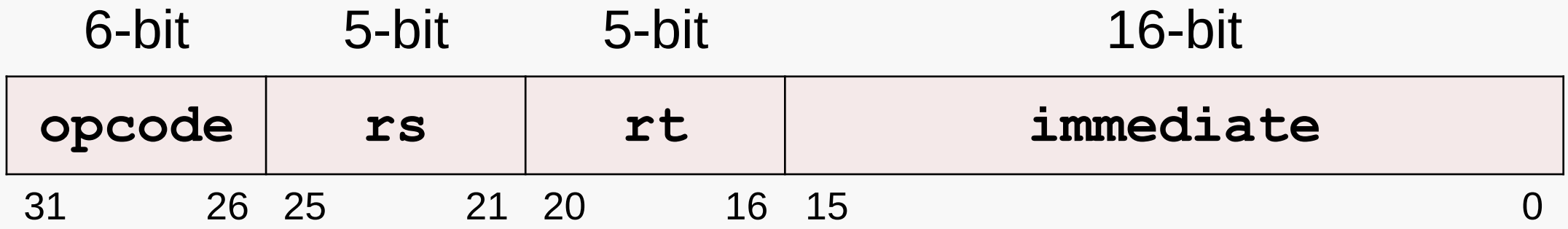
条件分支指令（I型）

- 条件分支
  - `beq rs,rt,imm  # opcode=4`
  - `bne rs,rt,imm  # opcode=5`

- 格式：**beq reg1,reg2,L1**

```
if (value in reg1)==(value in reg2)
   goto L1
```

I



条件分支指令的示例

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

C语言代码

```
 beq $s3,$s4,True      # branch i==j
 sub $s0,$s1,$s2       # f=g-h(false)
 j Fin                 # goto Fin

True: add $s0,$s1,$s2 # f=g+h (true)
Fin: …
```

MIPS汇编语言代码

条件分支指令的示例

```
if (g < h)
 goto Less;
#g:$s0,h:$s1
```

C语言代码

```
slt $t0,$s0,$s1
     # $t0 = 1, if g<h
bne $t0,$0,Less
     # goto Less, if $t0!=0
     # Register $0 always
     # contains the value 0
```

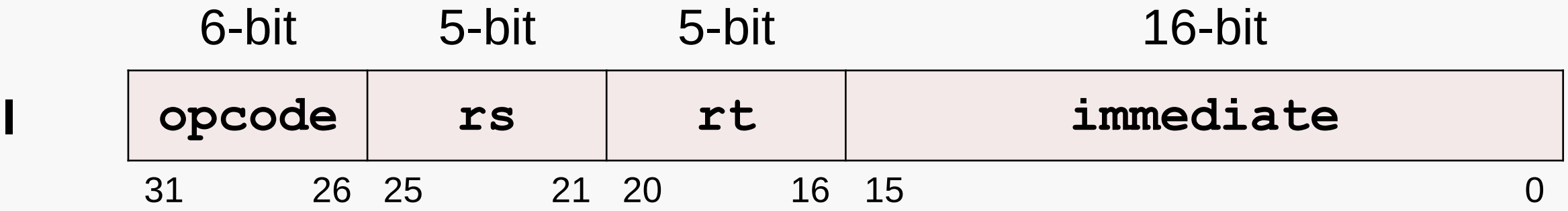
0号寄存器固定是0

MIPS汇编语言代码

问题：如何判断>、≥、≤的情况？
简单的方法是增加新指令，但是“Simpler is Better”

条件分支指令的目标地址范围

- 如何充分发挥16-bit的作用？
  - 以当前PC为基准，16-bit位移量可以表示$\pm 2^{15}$ bytes
  - MIPS的指令长度固定为32-bit（word）
  - 16-bit位移量可以表示 $\pm 2^{15}$ words = $\pm 2^{17}$ bytes（±128KB）
- 目标地址计算方法：
  - 分支条件不成立，`PC = PC + 4` = next instruction
  - 分支条件成立，`PC = (PC+4) + (immediate*4)`

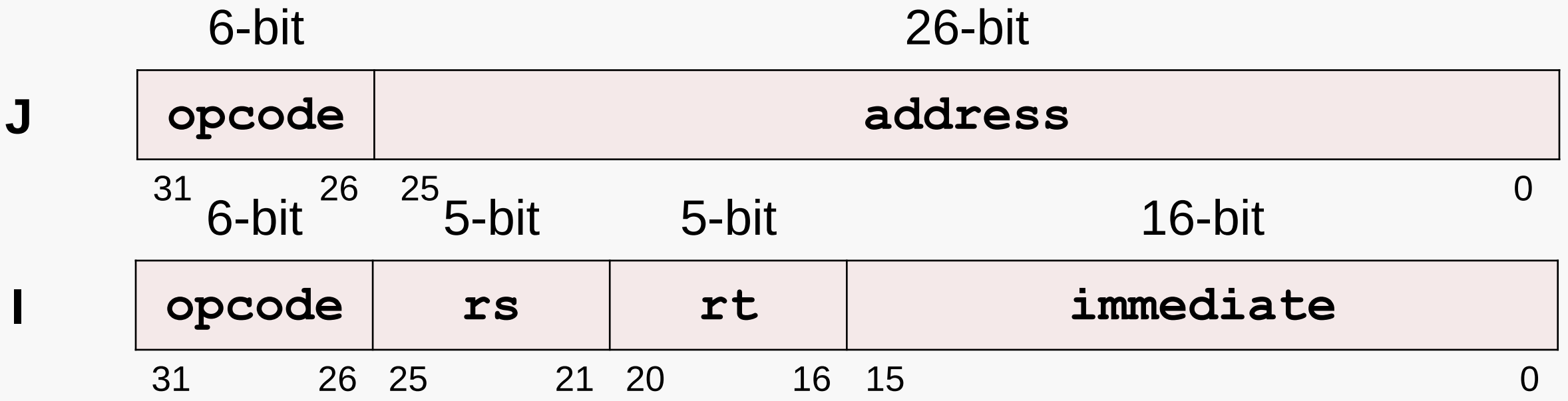


不同维度的指令分类（示例）

运算指令	add rd,rs,rt sll rd,rt,*shamt*	addi rt,rs,*imm* slti rt,rs,*imm*	/
访存指令	/	lw rt,*imm*(rs) sw rt,*imm*(rs)	/
分支指令	jr rs	beq rs,rt,*imm*	j *addr*
	R型指令	I型指令	J型指令

非条件分支指令（J型）

- 在不需要条件判断的情况下，如何扩大目标地址范围
  - 理想情况，直接使用32-bit地址
  - 冲突：MIPS的指令长度固定为32-bit，opcode占用了6-bit
- 目标地址计算方法：
  - New PC ={(PC+4)[31..28], address, 00}



两种分支指令示例

- 假设变量和寄存器的对应关系如下

  f → $s0　　g → $s1　　h → $s2

  i → $s3　　j → $s4

```
if (i == j)                    bne $s3,$s4,Else
  f = g + h;                   add $s0,$s1,$s2
else                           j Exit
  f = g - h;               Else:sub $s0,$s1,$s2
                               Exit:
```

对比：x86转移指令的目标地址范围

- 短转移：JMP　SHORT　LABEL
  - 操作：IP←IP+8位的位移量（-128~127Byte）
- 近转移：JMP　NEAR PTR　LABEL
  - 操作：IP←IP+16位的位移量（±32KByte）
- 远转移：JMP　FAR PTR　LABEL
  - 操作：IP←LABEL的偏移地址；CS ←LABEL的段基值

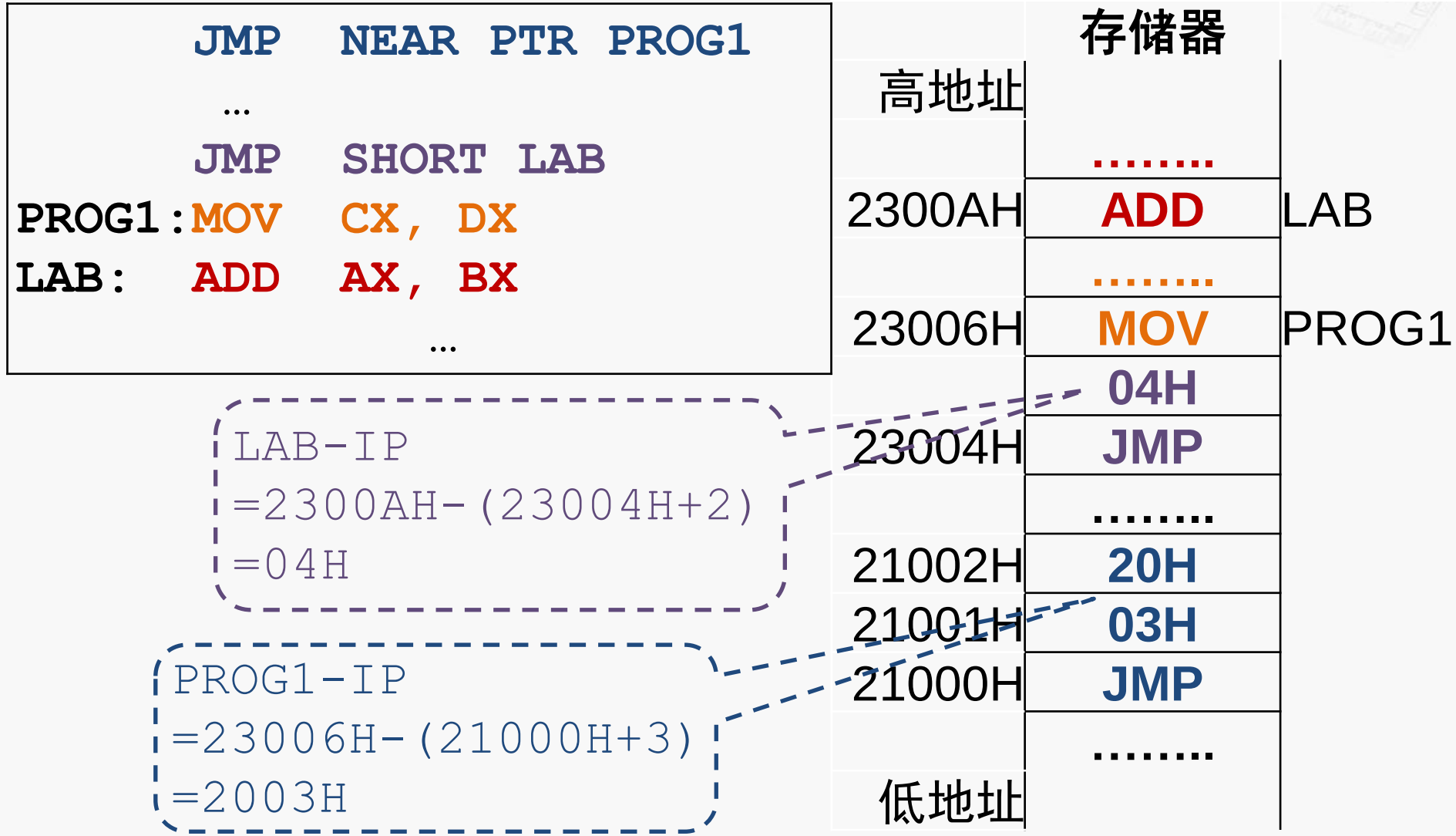
1. 位移量是一个带符号数，为LABEL的偏移地址与当前EIP/IP值之差
2. 从80386开始，近转移可以使用32位的位移量

说明

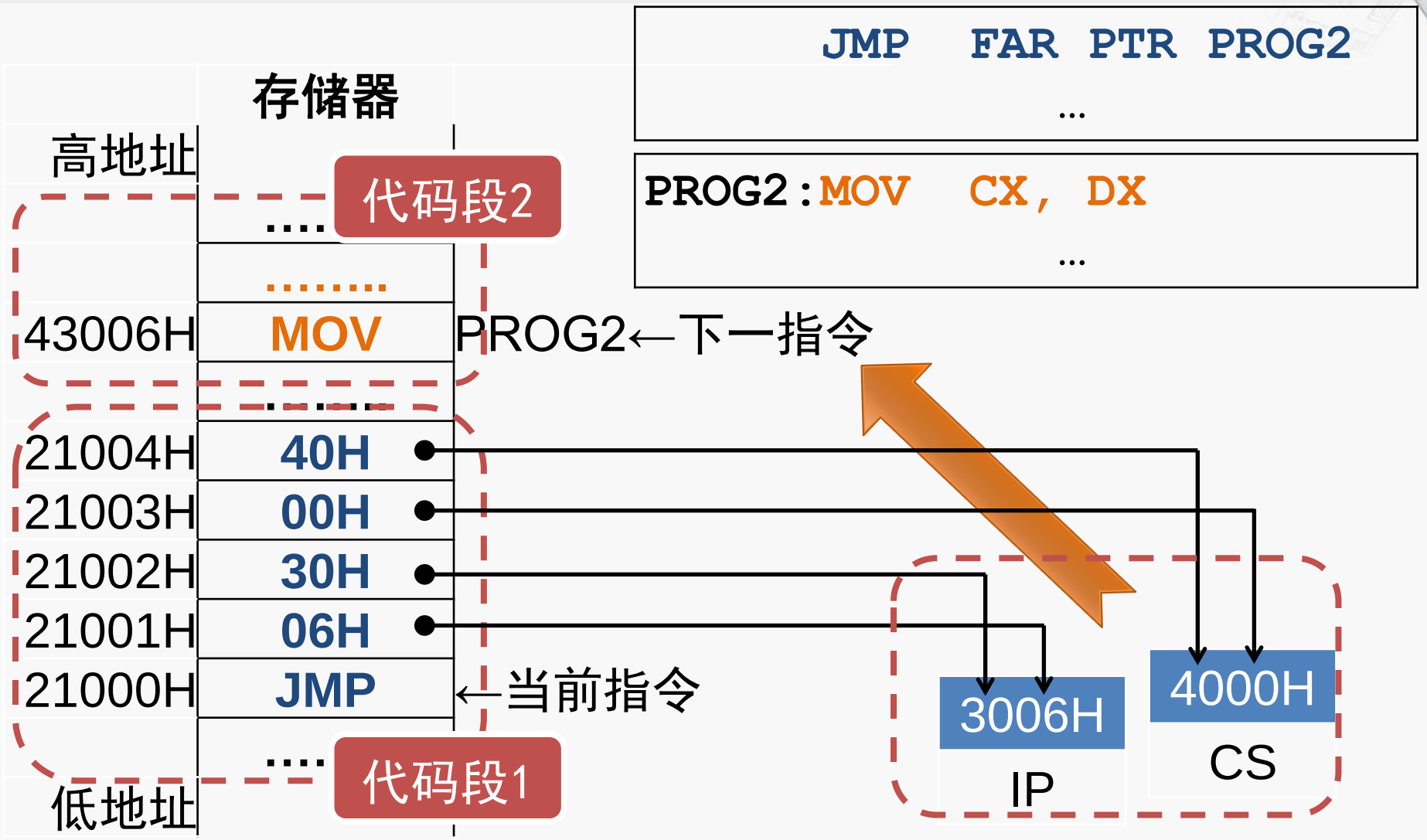
	操作码				
短转移	EB	8-bit位移量			
近转移	E9	16-bit位移量			
远转移	EA	IP	IP	CS	CS

（对比x86）段内直接转移的执行过程

```
        JMP   NEAR PTR PROG1
          …
        JMP   SHORT LAB
PROG1:  MOV   CX, DX
LAB:    ADD   AX, BX
                …
```



（对比x86）段间直接转移的执行过程

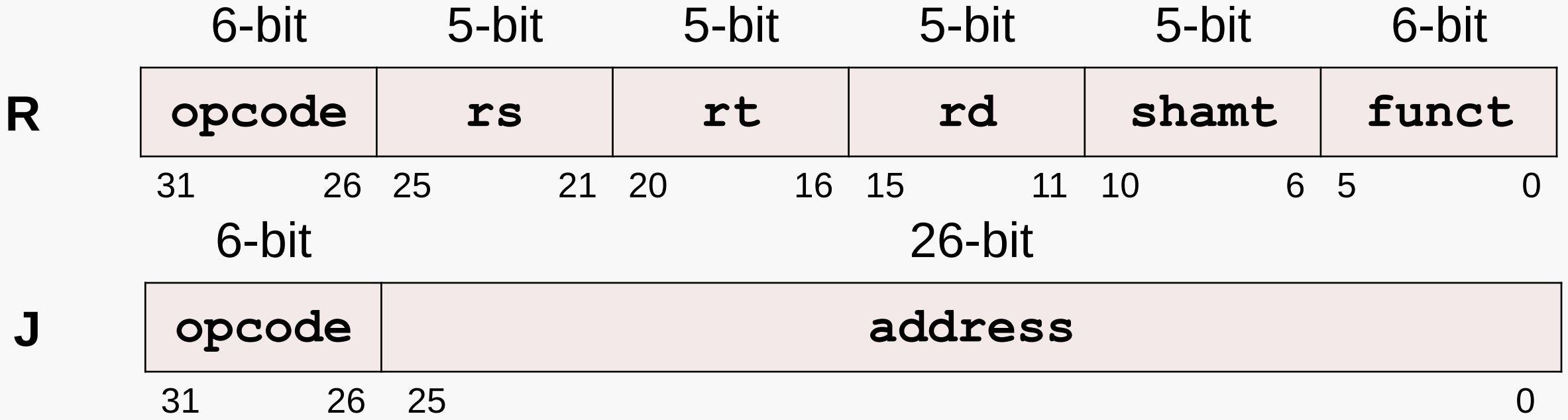


不同维度的指令分类（示例）

	R型指令	I型指令	J型指令
运算指令	add rd,rs,rt sll rd,rt,*shamt*	addi rt,rs,*imm* slti rt,rs,*imm*	/
访存指令	/	lw rt,*imm*(rs) sw rt,*imm*(rs)	/
分支指令	jr rs	beq rs,rt,*imm*	j *addr*

非条件分支指令（R型）

- J型指令的目标地址范围：2^{28}bytes（256MB）
  - 相对于当前PC的最远距离为±2^{28}bytes（±256MB）——为什么？
- 如何到达更远的目标地址？
  - （1）2次调用`j`指令；（2）使用`jr`指令：`jr rs`



（对比x86）段间间接转移

- 格式：JMP　DWORD PTR　OPR

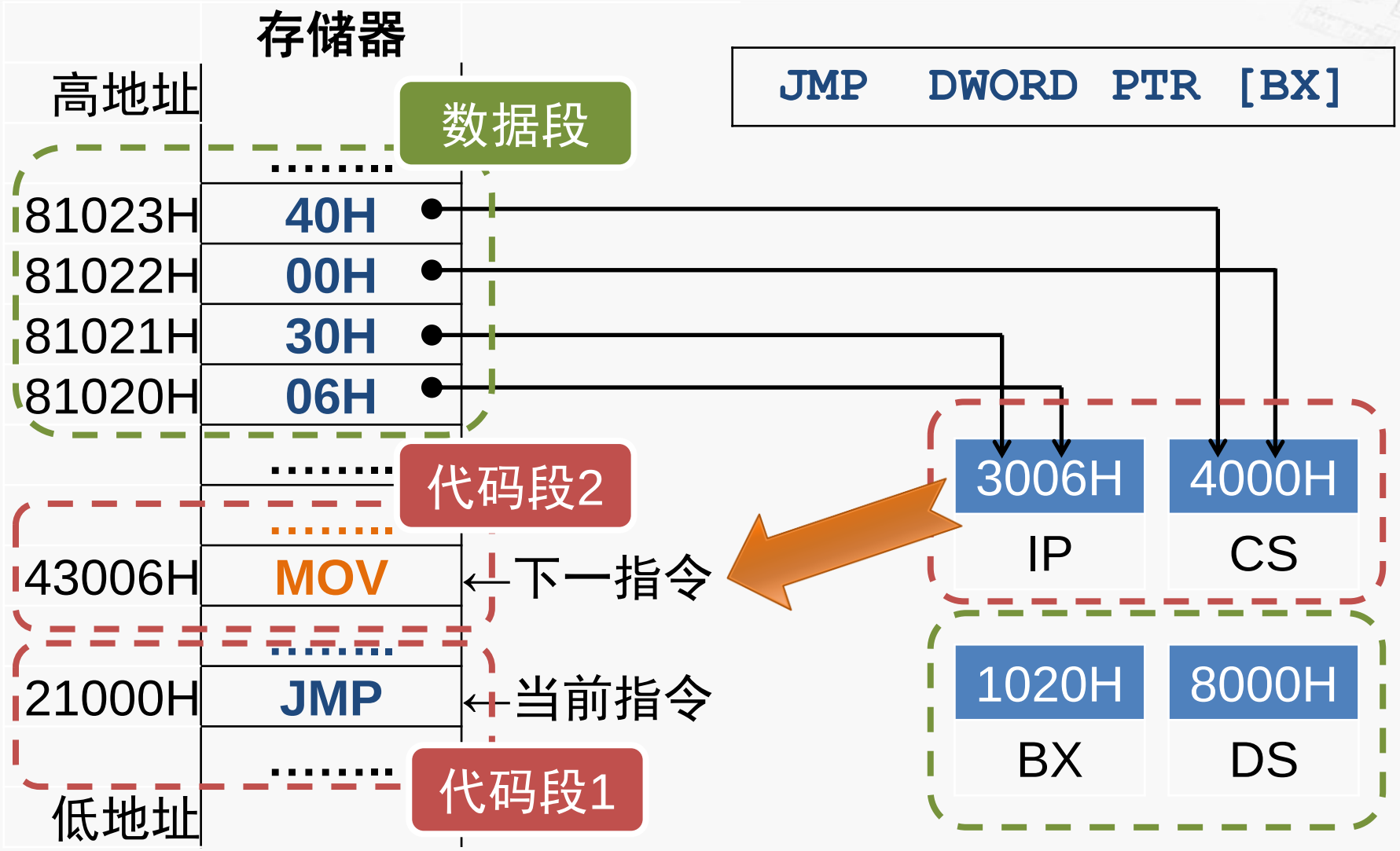
- 操作

  ① 寻址到OPR指定的存储器单元双字

  ② 将该双字中的低字送到IP寄存器中

  ③ 将该双字中的高字送到CS寄存器中

（对比x86）段间间接转移的执行过程



主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

I MIPS指令分类说明：分支

II MIPS汇编程序示例

III 指令系统概览

IV 指令系统举例

汇编器提供的伪指令

- 寄存器传送
  - 格式：`move dst,src`
  - 实际：`addi dst,src,0`
- 装载地址：Load Address (`la`)
  - 格式：`la dst,label`
  - 如何实现？
- 装载32位立即数：Load Immediate (`li`)
  - 格式：`li dst,imm`
  - 如何实现？

示例程序

```
// 复制字符串:  x[] to y[]

char *p, *q;

p = &x[0];  // p = 字符串x的第一个字符的地址
q = &y[0];  // q = 字符串y的第一个字符的地址

while((*q++ = *p++) != '\0') ;
```

MIPS汇编语言实现

```
        lw $t1,Base Address
        lw $s1,0($t1)           # $s1 = p
        lw $s2,4($t1)           # $s2 = q
Loop:   lb $t2,0($s1)           # $t2 = *p
        sb $t2,0($s2)           # *q = $t2
        addi $s1,$s1,1          # p = p + 1
        addi $s2,$s2,1          # q = q + 1
        beq $t2,$zero,Exit      # if *p==0, Exit
        j Loop                  # goto Loop
Exit:   # 复制N个字符需要N*6+3条指令
```

对比：x86的串操作指令

作用

- 对存储器中的数据串进行每次一个元素的操作
- 串的基本单位是字节或字（即“一个元素”）
- 串长度可达64KB

分类

- 共5条串操作指令
- 另有3种重复前缀，与串操作指令配合使用

（对比x86）串操作指令的列表

分组	助记符	功能
串操作指令	MOVS （MOVSB，MOVSW）	串传送 （字节串传送，字串传送）
	CMPS （CMPSB，CMPSW）	串比较 （字节串比较，字串比较）
	SCAS （SCASB，SCASW）	串扫描 （字节串扫描，字串扫描）
	LODS （LODSB，LODSW）	取串 （取字节串，取字串）
	STOS （STOSB，STOSW）	存串 （存字节串，存字串）
重复前缀	REP	无条件重复前缀
	REPE / REPZ	相等/为零重复前缀
	REPNE / REPNZ	不相等/不为零重复前缀

（对比x86）MOVS指令说明

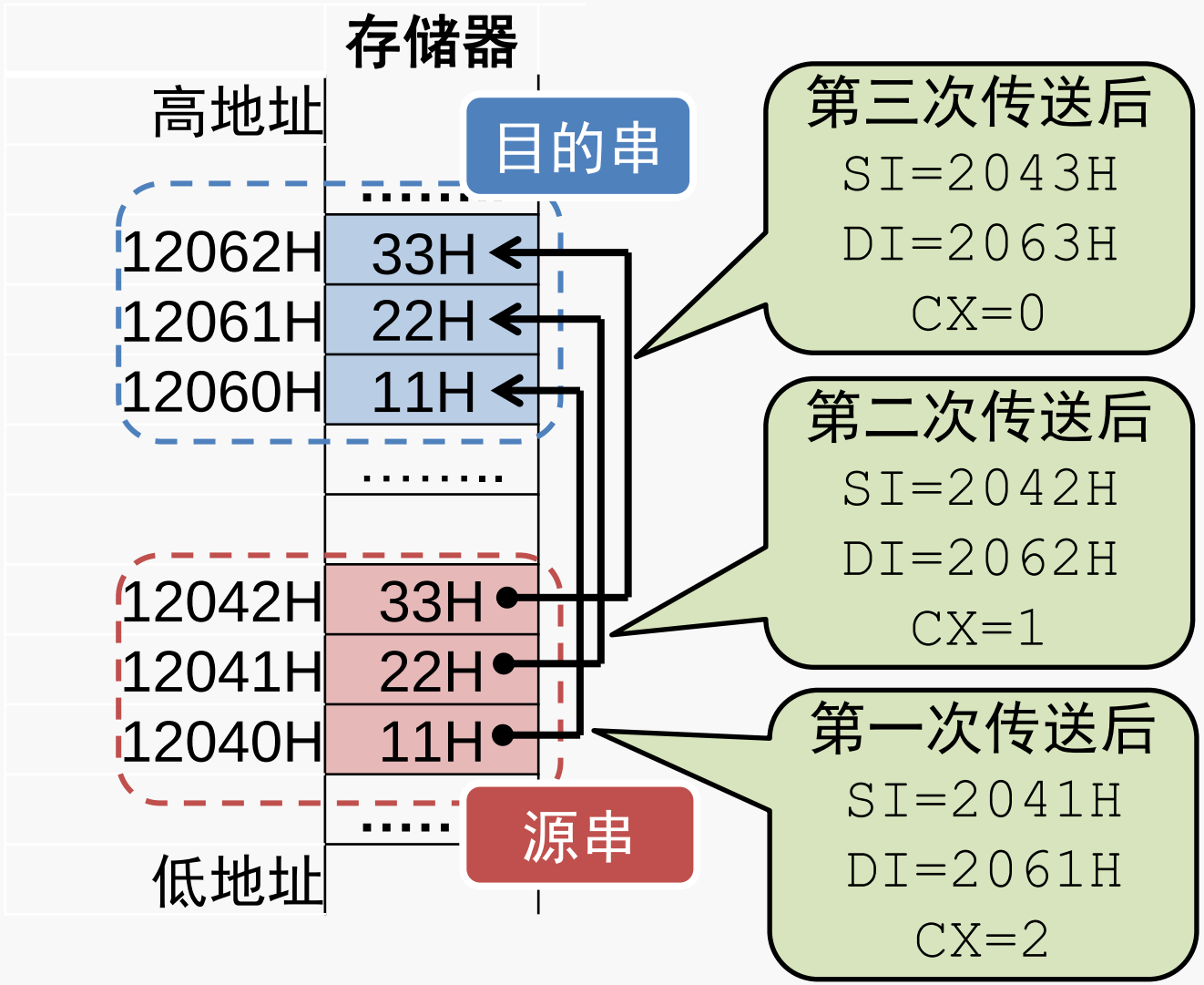
MOVSB指令（字节串传送）

- 格式：MOVSB
- 操作：将地址DS:SI所指向的存储器字节单元传送到地址ES:DI；修改SI和DI的内容，指向串的下一元素

MOVSW指令（字串传送）

- 格式：MOVSW
- 操作：将地址DS:SI所指向的存储器字单元传送到地址ES:DI；修改SI和DI的内容，指向串的下一元素

（对比x86）串传送指令示例



设DS=1000H

```
MOV  AX,  DS
MOV  ES,  AX
MOV  SI,  2040H
MOV  DI,  2060H
CLD
MOV  CX,  3
REP  MOVSB
```

```
MOVSB;第一次传送
MOVSB;第二次传送
MOVSB;第三次传送
```

（对比x86）重复前缀说明（1）

REP（无条件重复）

- 格式：REP 串操作指令
- 操作：当CX≠0时，重复执行串操作指令
- 适用：MOVS（串传送），STO（存串）

带重复前缀的串操作指令执行时间可能很长，执行过程中允许中断

说明

```
        lw $t1,Base Address
        lw $s1,0($t1)              # $s1 = p
        lw $s2,4($t1)              # $s2 = q
Loop:   lb $t2,0($s1)              # $t2 = *p
        sb $t2,0($s2)              # *q = $t2
        addi $s1,$s1,1             # p = p + 1
        addi $s2,$s2,1             # q = q + 1
        beq $t2,$zero,Exit         # if *p==0, Exit
        j Loop                     # goto Loop
Exit:   # 复制N个字符需要N*6+3条指令
```

MIPS没有此类长时间重复操作的复杂指令，每条指令的操作相对单一

对比

（对比x86）串操作指令的共同特性

- 隐含操作数
  - 源串地址为DS:SI，目的串地址为ES:DI
  - 串的长度在CX寄存器中
- 处理完一个串元素后的操作（硬件自动完成）

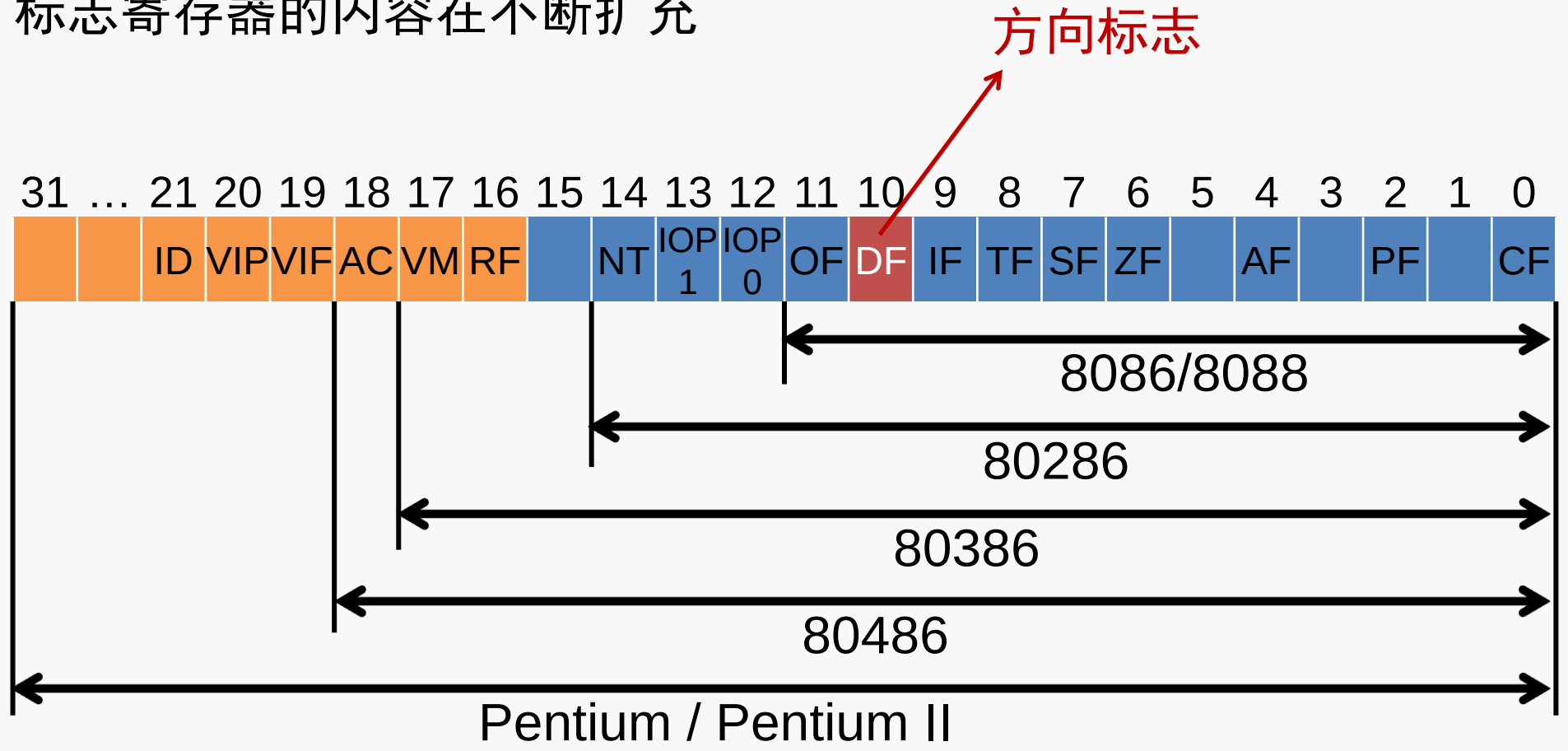
① 若使用重复前缀，则CX ←CX-1

② 修改SI和DI：

标志位 \ 串元素位宽	字节	字
方向标志DF=0	SI←SI+1 DI←DI+1	SI←SI+2 DI←DI+2
方向标志DF=1	SI←SI-1 DI←DI-1	SI←SI-2 DI←DI-2

（对比x86）标志寄存器的说明

- 标志寄存器EFLAGS/FLAGS
  - 用于指示微处理器的状态并控制它的操作
  - 标志寄存器的内容在不断扩充



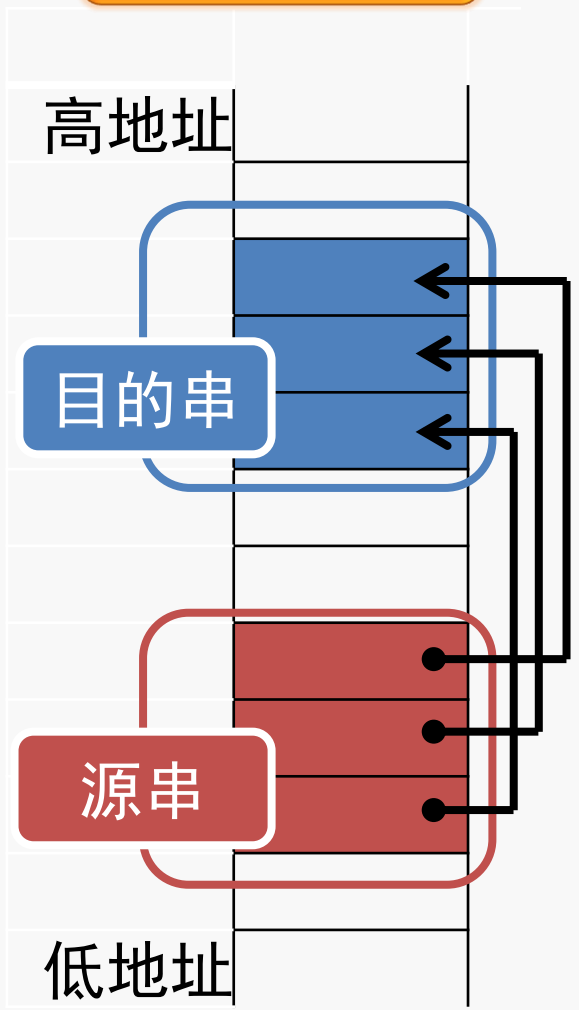
（对比x86）串传送方向

- 设置DF=0
  - 从“源串”的低地址开始传送
  - 传送过程中，SI和DI自动增量修改
- 设置DF=1
  - 从“源串”的高地址开始传送
  - 传送过程中，SI和DI自动减量修改

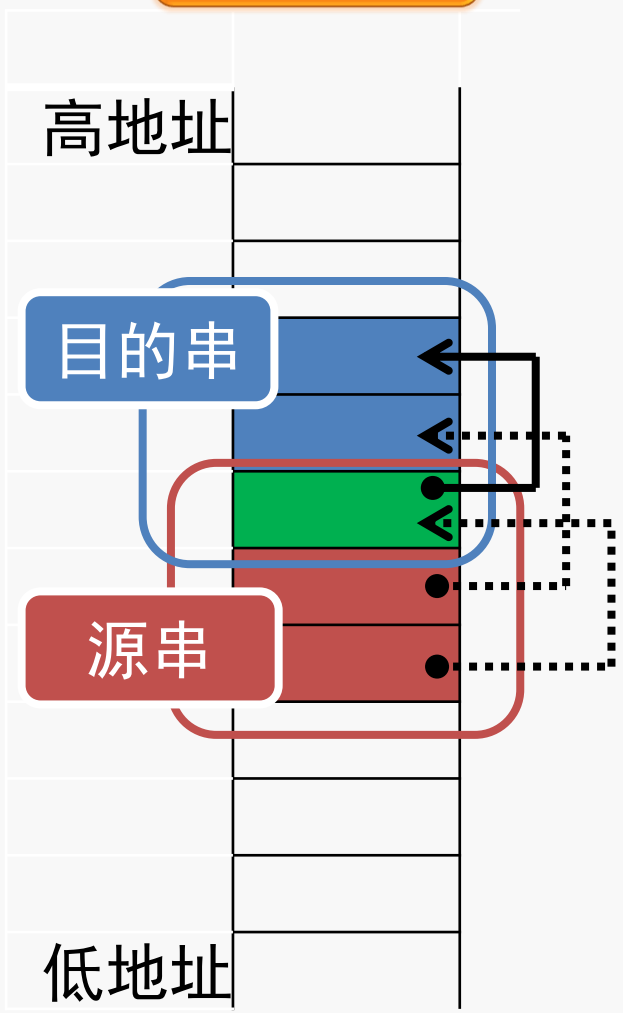
- 用途
  - 应对“源串”和“目的串”的存储区域部分重叠的问题

（对比x86）方向标志的作用

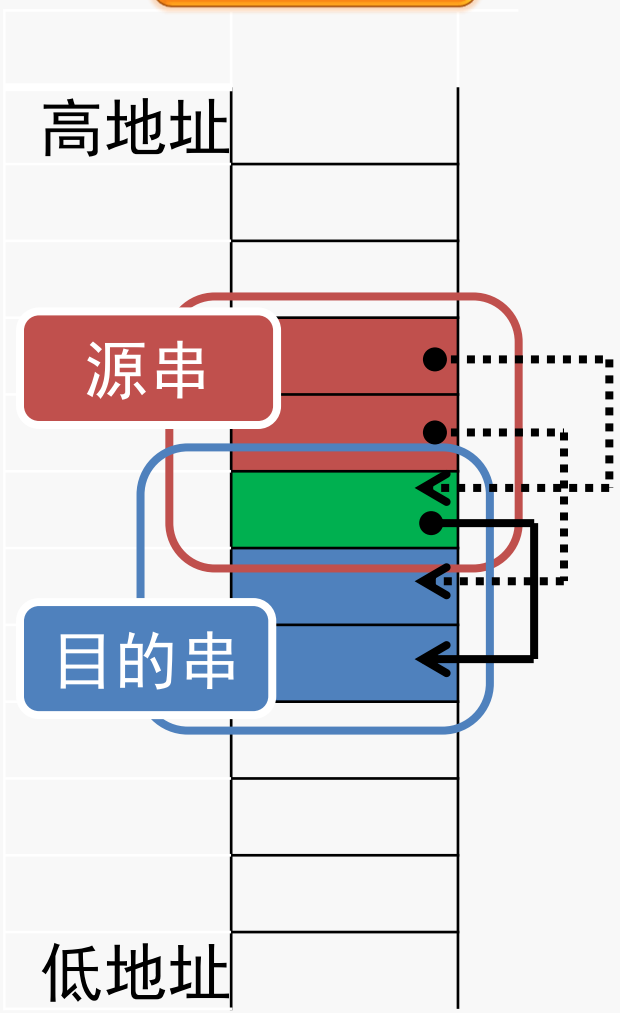
1. DF=0或1



2. DF=1



3. DF=0



（对比x86）重复前缀说明

REPNE和REPNZ（不相等/不为零重复）

- 格式：REPNE 串操作指令
  或 REPNZ 串操作指令

- 操作：当ZF=0且CX≠0时，重复执行

- 适用：CMPS（串比较），SCAS（串扫描）

（对比x86） SCAS指令说明

SCASB指令 / SCASW指令（字节/字串扫描）

- 格式：SCASB　　格式： SCASW
- 操作：
  - 比较“AL/AX的内容”和“目的串当前字节/字”
  - 修改DI而不修改SI
  - 不回写结果，只影响标志位
- 说明：
  - 使用重复前缀REPE / REPZ或REPNE / REPNZ
  - 寻找“目的串”中第一个与AL/AX的值相同或不相同的元素

（对比x86）串操作指令示例

```
CLD
MOV   DI, 2060H
MOV   CX, 100
MOV   AL, '$'

REPNE   SCASB
JZ   label_zer
…
label_zer:
…
```

从地址[2060H]开始的100个字节中寻找$字符

因为如果找到 $，那么zero flag会被SCASB设置为1，然后SCASB因为REPNE会退出

若在串中找到$字符，则跳转

主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

I MIPS指令分类说明：分支

II MIPS汇编程序示例

III 指令系统概览

IV 指令系统举例

1970年代的体系结构

	IBM 360/370	Intel 8086	Motorola 68000	DEC VAX
Date announced	1964/1970	1978	1980	1977
Instruction size(s) (bits)	16, 32, 48	8, 16, 24, 32, 40, 48	16, 32, 48, 64, 80	8, 16, 24, 32,..., 432
Addressing (size, model)	24 bits, flat/ 31 bits, flat	4 + 16 bits, segmented	24 bits, flat	32 bits, flat
Data aligned?	Yes 360/No 370	No	16-bit aligned	No
Data addressing modes	2/3	5	9	= 14
Protection	Page	None	Optional	Page
Page size	2 KB & 4 KB	—	0.25 to 32 KB	0.5 KB
I/O	Opcode	Opcode	Memory mapped	Memory mapped
Integer registers (size, model, number)	16 GPR . 32 bits	8 dedicated data . 16 bits	8 data and 8 address . 32 bits	15 GPR . 32 bits
Separate floating-point registers	4 . 64 bits	Optional: 8 . 80 bits	Optional: 8 . 80 bits	0
Floating-point format	IBM (floating hexadecimal)	IEEE 754 single, double, extended	IEEE 754 single, double, extended	DEC

RISC体系结构（服务器和桌面应用）

	Alpha	MIPS I	PA-RISC 1.1	PowerPC	SPARC v.8
Date announced	1992	1986	1986	1993	1987
Instruction size (bits)	32	32	32	32	32
Address space (size, model)	64 bits, flat	32 bits, flat	48 bits, segmented	32 bits, flat	32 bits, flat
Data alignment	Aligned	Aligned	Aligned	Unaligned	Aligned
Data addressing modes	1	1	5	4	2
Protection	Page	Page	Page	Page	Page
Minimum page size	8 KB	4 KB	4 KB	4 KB	8 KB
I/O	Memory mapped	Memory mapped	Memory mapped	Memory mapped	Memory mapped
Integer registers (number, model, size)	31 GPR . 64 bits	31 GPR . 32 bits	31 GPR . 32 bits	32 GPR . 32 bits	31 GPR . 32 bits
Separate floating-point registers	31 . 32 or 31 . 64 bits	16 . 32 or 16 . 64 bits	56 . 32 or 28 . 64 bits	32 . 32 or 32 . 64 bits	32 . 32 or 32 . 64 bits
Floating-point format	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double	IEEE 754 single, double

RISC体系结构（嵌入式应用）

	ARM	Thumb	SuperH	M32R	MIPS16
Date announced	1985	1995	1992	1997	1996
Instruction size (bits)	32	16	16	16/32	16/32
Address space (size, model)	32 bits, flat	32 bits, flat	32 bits, flat	32 bits, flat	32/64 bits, flat
Data alignment	Aligned	Aligned	Aligned	Aligned	Aligned
Data addressing modes	6	6	4	3	2
Integer registers (number, model, size)	15 GPR x 32 bits	8 GPR + SP, LR x 32 bits	16 GPR x 32 bits	16 GPR x 32 bits	8 GPR + SP, RA x 32/64 bits
I/O	Memory mapped	Memory mapped	Memory mapped	Memory mapped	Memory mapped

RISC的寻址模式

Addressing mode	Alpha	MIPS64	PA-RISC 2.0	PowerPC	SPARC v.9
Register + offset (displacement or based)	X	X	X	X	X
Register + register (indexed)	—	X (FP)	X (Loads)	X	X
Register + scaled register (scaled)	—	—	X	—	—
Register + offset and update register	—	—	X	X	—
Register + register and update register	—	—	X	X	—

Addressing mode	ARM v.4	Thumb	SuperH	M32R	MIPS16
Register + offset (displacement or based)	X	X	X	X	X
Register + register (indexed)	X	X	X	—	—
Register + scaled register (scaled)	X	—	—	—	—
Register + offset and update register	X	—	—	—	—
Register + register and update register	X	—	—	—	—
Register indirect	—	—	X	X	—
Autoincrement, autodecrement	X	X	X	X	—
PC-relative data	X	X (loads)	X	—	X (loads)

x86寻址模式的使用比例

Addressing mode	Integer average	FP average
Register indirect	13%	3%
Base + 8-bit disp.	31%	15%
Base + 32-bit disp.	9%	25%
Indexed	0%	0%
Based indexed + 8-bit disp.	0%	0%
Based indexed + 32-bit disp.	0%	1%
Base + scaled indexed	22%	7%
Base + scaled indexed + 8-bit disp.	0%	8%
Base + scaled indexed + 32-bit disp.	4%	4%
32-bit direct	20%	37%

主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

I MIPS指令分类说明：分支

II MIPS汇编程序示例

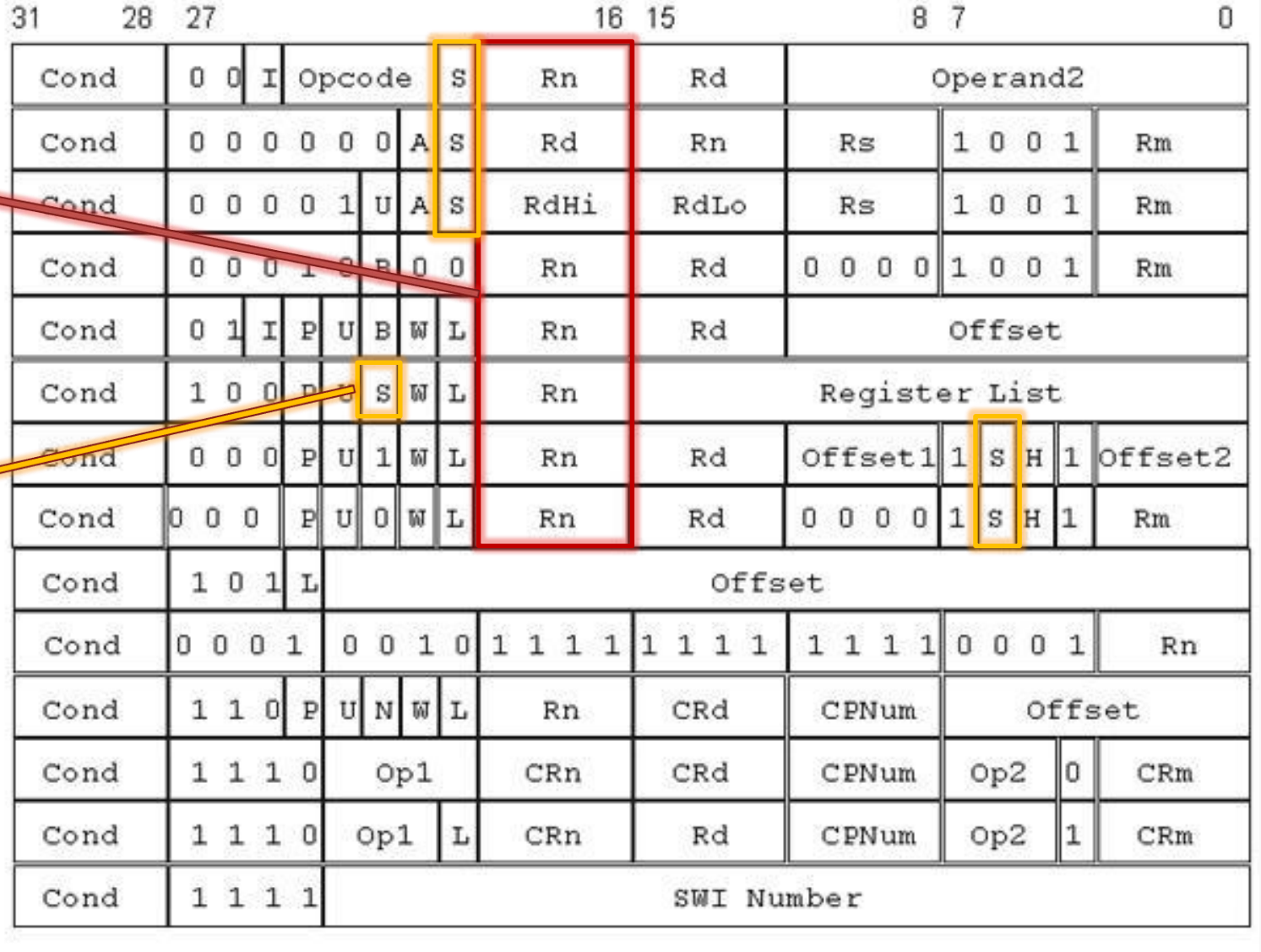
III 指令系统概览

IV 指令系统举例

ARM指令格式

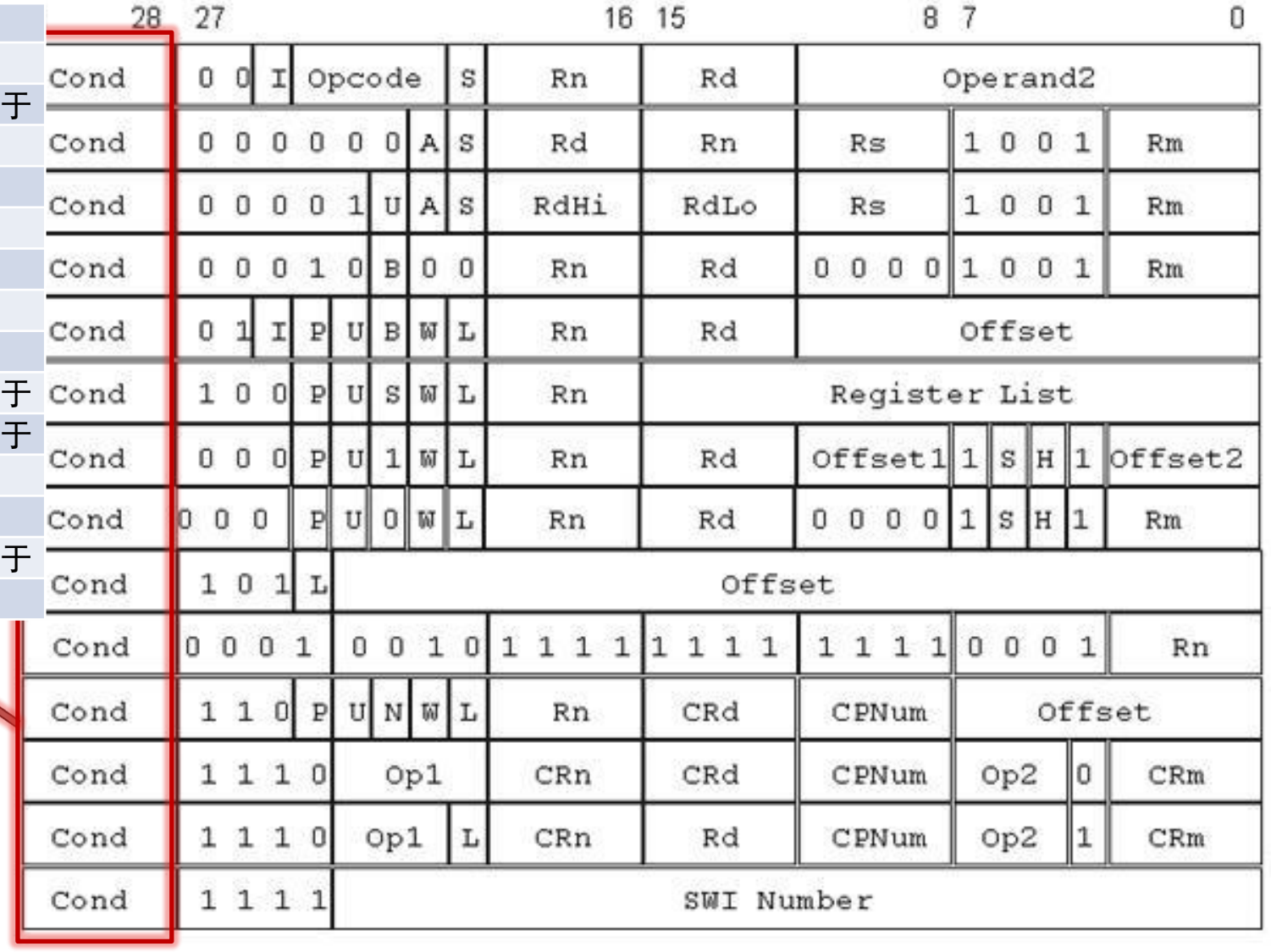
寄存器编号位域只有4-bit，只能寻址16个通用寄存器

功能位域的位置不统一，给指令译码带来不便



ARM指令格式

条件码	助记符	条件	含义
0000	EQ	Z置位	相等
0001	NE	Z清零	不相等
0010	CS/HS	C置位	无符号数大于或等于
0011	CC/LO	C清零	无符号数小于
0100	MI	N置位	负数
0101	PL	N清零	正数或零
0110	VS	V置位	溢出
0111	VC	V清零	未溢出
1000	HI	C置位Z清零	无符号数大于
1001	LS	C清零Z置位	无符号数小于或等于
1010	GE	N等于V	带符号数大于或等于
1011	LT	N不等于V	带符号数小于
1100	GT	Z清零且(N等于V)	带符号数大于
1101	LE	Z置位且(N不等于V)	带符号数小于或等于
1110	AL	忽略	无条件执行



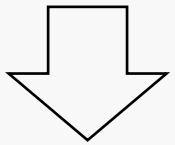
所有的ARM指令都带有4-bit的条件码，都可以条件执行

ARM的条件执行

条件码	助记符	条件	含义
0000	EQ	Z置位	相等
0001	NE	Z清零	不相等
0010	CS/HS	C置位	无符号数大于或等于
0011	CC/LO	C清零	无符号数小于
0100	MI	N置位	负数
0101	PL	N清零	正数或零
0110	VS	V置位	溢出
0111	VC	V清零	未溢出
1000	HI	C置位Z清零	无符号数大于
1001	LS	C清零Z置位	无符号数小于或等于
1010	GE	N等于V	带符号数大于或等于
1011	LT	N不等于V	带符号数小于
1100	GT	Z清零且(N等于V)	带符号数大于
1101	LE	Z置位且(N不等于V)	带符号数小于或等于
1110	AL	忽略	无条件执行

C代码

```
if(a>b)  a++ ;
else   b++ ;
```

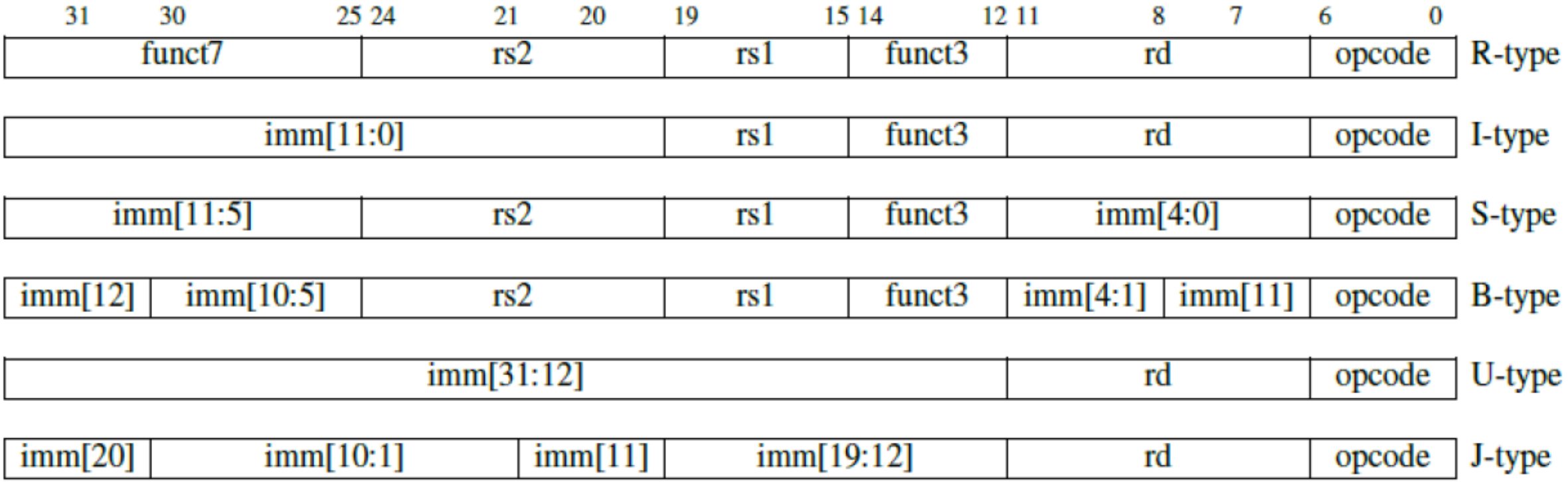


ARM汇编代码

```
CMP    R0,R1
ADDHI  R0,R0,#1 ;if R0>R1,R0=R0+1
ADDLS  R1,R1,#1 ;if R0≤R1,R1=R1+1
```

RISC-V指令格式

- R 型：寄存器-寄存器操作
- I 型：短立即数和访存load 操作
- S 型：访存 store 操作
- B 型：条件跳转操作
- U 型：长立即数操作
- J 型：无条件跳转操作



本讲到此结束，谢谢

欢迎继续学习本课程

计算机组织与体系结构 Computer Architectures

主讲：陆俊林

北京大学 Peking University