



第八章 指令系统(二)

秦磊华 计算机学院

- 8.4 指令格式设计
- 8.5 指令系统发展方向
- 8.6 MIPS指令
- 8.7 RISC-V 指令
- 8.8 ARM 指令

CONTENT



1. 指令格式设计的主要内容

操作码字段	寻址方式	地址码字段
-------	------	-------

- ◆ 根据指令数量的要求及是否支持操作码扩展，确定操作码字段的位数
- ◆ 根据对操作数的要求确定地址码字段的个数和位数
- ◆ 根据寻址方式的要求，为每个地址码字段确定寻址方式字段位数
- ◆ 确定采用定长指令还是变长指令
- ◆ 上述各方面的协调
- ◆ 设计方案分析及优化再设计

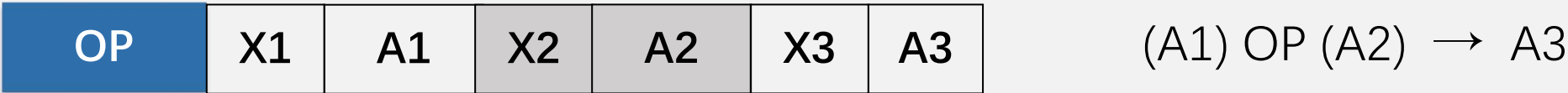


8.4 指令格式设计

2. 指令格式设计举例

例1 某机字长32位，采用三地址指令，支持8种寻址方式和60条指令，可在2K主存范围内访问操作数、1K主存范围内保存运算结果。设计该指令格式？指令字长最少应为多少位？执行一条指令最多访问多少次主存(不考虑访存缺页中断)？

解：根据题目条件，指令格式如下：



OP = 6位

X1 = X2 = X3 = 3, 共 9位

A1=A2=11, A3=10位, 共 32位

47位

指令占2个存储字，最多访存5次！

如何优化？



8.4 指令格式设计

2. 指令格式设计举例

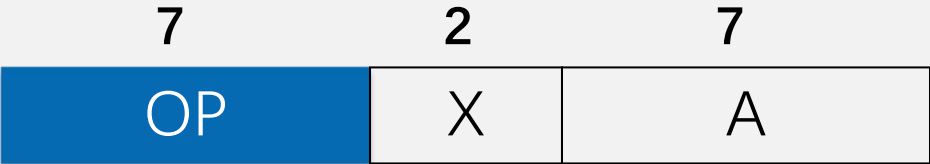
例2 字长16位，主存64K，指令单字长单地址，80条指令。寻址方式有直接、间接、相对、变址。请设计指令格式

解：80条指令 \Rightarrow OP字段需要7位($2^7=128$)

4种寻址方式 \Rightarrow 寻址方式特征位2位

单地址位长度: $16-7-2=7$ 位

指令格式：



2. 指令格式设计举例

7	2	7
OP	X	A

设PC寄存器16位 变址寄存器16位

相对寻址 $E = (PC) + A$ ，寻址范围为 $64K$

变址寻址 $E = (R) + A$ ，寻址范围为 $64K$

直接寻址 $E = A$ ，寻址范围为 $0 \sim 128$

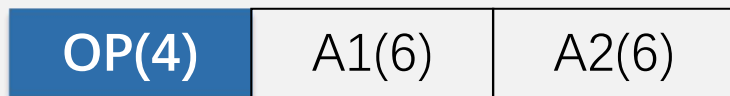
间接寻址 $E = (A)$ ，寻址范围为 $64K$

设计评价:

◆访问主存的方式太多 ◆缺立即数寻址

2. 指令格式设计举例

例3 设某指令系统指令字长16位，每个地址码为6位。若要求设计二地址指令15条、一地址指令34条，问最多还可设计多少条零地址指令？



解: 操作码按从短码到长码进行扩展编码

假定二地址指令编码: (0000 – 1110) 共15条, 1111作为扩展标识;

则一地址指令编码的全集可表示为: 1111 000000 – 111111(假定操作码扩向A1);

一地址指令只需要34条, 多余的30种一地址指令编码(假定为: 100010 – 111111)
作为向零地址指令扩展的标识;

故最多可支持的故零地址指令数为: 30×2^6 条



8.4 指令格式设计

2. 指令格式设计举例

例4. 某计算机字长为16位，主存地址空间大小为128KB，按字编址。采用单字长指令格式，指令各字段定义如图，转移地址采用相对寻址方式，相对偏移量用补码表示。寻址方式如图。



M _s /M _d	寻址方式	助记符	含义
000B	寄存器直接	R _n	操作数=(R _n)
001B	寄存器间接	(R _n)	操作数=((R _n))
010B	寄存器间接，自增	(R _n)+	操作数=((R _n)), (R _n)+1→(R _n)
011B	相对	D(R _n)	转移目标地址=(PC)+(R _n)

注 (x) 表示存储器地址x或寄存器x的内容

16



(1)该指令系统最多可有多少条指令？该计算机最多有多少个通用寄存器？存储器地址寄存器MAR和存储器数据寄存器MDR至少需要多少位？

16/16

8

2. 指令格式设计举例

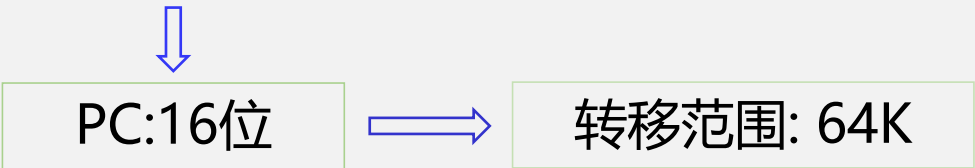
例4. 某计算机字长为16位，主存地址空间大小为128KB，按字编址。采用单字长指令格式，指令各字段定义如图，转移地址采用相对寻址方式，相对偏移量用补码表示。寻址方式如图。



M _s /M _d	寻址方式	助记符	含义
000B	寄存器直接	R _n	操作数=(R _n)
001B	寄存器间接	(R _n)	操作数=((R _n))
010B	寄存器间接，自增	(R _n)+	操作数=((R _n)), (R _n)+1→(R _n)
011B	相对	D(R _n)	转移目标地址=(PC)+(R _n)

注 (x) 表示存储器地址x或寄存器x的内容

(2) 转移指令的目标地址范围是多少?



8.4 指令格式设计

2. 指令格式设计举例

例4. 某计算机字长为16位，主存地址空间大小为128KB，按字编址。采用单字长指令格式，指令各字段定义如图，转移地址采用相对寻址方式，相对偏移量用补码表示。寻址方式如图。



M _s /M _d	寻址方式	助记符	含义
000B	寄存器直接	R _n	操作数=(R _n)
001B	寄存器间接	(R _n)	操作数=((R _n))
010B	寄存器间接，自增	(R _n)+	操作数=((R _n)), (R _n)+1→(R _n)
011B	相对	D(R _n)	转移目标地址=(PC)+(R _n)



(3)操作码0010B表示加法指令(add)，寄存器R4，R5编号分别为100B和101B，R4内容为1234H，R5内容为5678H，地址1234H中内容为5678H，地址5678H中内容为1234H，add(R4),(R5)+,逗号前为源操作数，逗号后为目的操作数,对应的机器码是多少？该指令执行后，哪些寄存器和存储单元的内容会发生改变？改变后的内容是什么？

1. RISC与CISC 概述

1) CISC(Complex Instruction System Computer:复杂指令系统计算机)

- ◆指令数量多, 指令功能复杂, 寻址方式多

- ◆ Intel X86

2) RISC(Reduced Instruction System Computer :精简指令系统计算机)

- ◆指令数量少, 指令功能单一, 寻址方式少

- ◆MIPS、RISC-V

3) CSIC、RISC互相融合

2. RISC

- ◆ 指令条数少，只保留使用频率最高的简单指令，指令定长

- ◆ Load/Store架构

只有存/取数指令才能访问存储器，其余指令的操作都在寄存器之间进行

- ◆ 指令长度固定，指令格式简单、寻址方式简单

- ◆ CPU设置大量寄存器（32~192）

- ◆ 一个机器周期完成一条机器指令


- ◆ CPU采用硬布线控制

3. 指令集体系结构 Instruction Set Architecture (ISA)

- ◆ 可执行的指令集合、格式、操作数的类型及相关规定
 - ◆ 可用的寄存器组的结构及应用方法
 - ◆ 可用的存储空间的大小及编址方式
 - ◆ 主存中数据的大、小端存放
 - ◆ 指令执行过程的控制方式
-
- ◆ 支持不同指令的CPU，其体系结构存在不同
 - ◆ 使用不同指令的程序设计者，需要了解的硬件特性不同
 - ◆ ISA 是硬件与软件的接口

8.5 指令系统发展方向

3. 指令集体系结构 Instruction Set Architecture (ISA)

	1970 DEC PDP-11	1992 ALPHA (64位)
	1978 x86 , 2001 IA64	
	1980 PowerPC	
	1981 MIPS	
	1985 SPARC	
	1991 arm	
	2016 RISC-V	



8.5 指令系统发展方向

3. 指令集体系结构 Instruction Set Architecture (ISA)

CPU名称	CPU架构	指令集	应用	
华为鲲鹏/海思	arm	RISC	服务器、华为手机	授权
中电飞腾	arm	RISC	服务器、手机	授权
澎湃s1	arm	RISC	服务器、小米手机	
海光/中科曙光	x86	CISC	服务器, 笔记本	IP授权
中科龙芯	mips	MIPS	服务器, 笔记本	授权+自研
上海申威	x86	alpha	服务器, 笔记本	授权+自研
上海兆芯	x86	CISC	服务器, 笔记本	威盛合资

采用授权指令系统可以研制产品，但不可能形成**自主产业生态**，就像中国人可以用英文写文章，但不可能基于英文形成民族文化 胡伟武(龙芯中科技术股份有限公司董事长)

3. 指令集体系结构 Instruction Set Architecture (ISA)



敢于构建新的指令系统的生态，是因为积累让我们掌握了9个能力

第一类是3个**基础编译器**，包括GCC、LLVM、GOLANG;

第二类是三个**虚拟机**，包含Java虚拟机、JavaScript虚拟机、.NET虚拟机;

第三类是**二进制翻译系统**，包括X86、ARM和MIPS指令系统的翻译。

4. 自主可控IT生态

实现IT底层架构的自主可控，不仅涉及基础硬件和基础软件的自主设计与实现，更需要对基础硬件、软件进行系统级适配、调优、完善。因此计算机产业自主可控应包含完整的IT生态系统相关软硬件的设计、集成、调优等多方面的核心知识和系级工程实践能力。



8.5 指令系统发展方向

4. 自主可控IT生态



积极投身我国信息产业自主可控国家战略，积极参加华为开发者社区、华为开源社区



8.6 MIPS指令

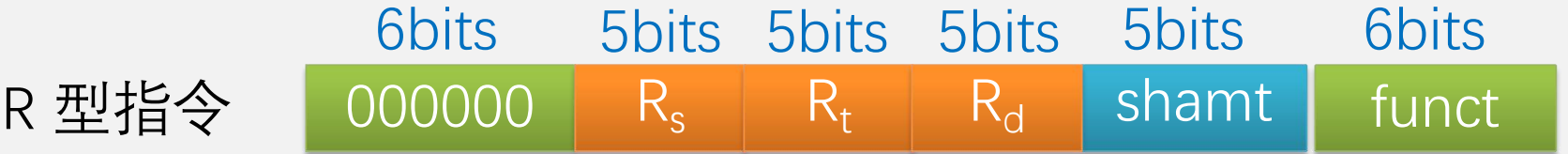
1. MIPS指令概述

MIPS (Million Instructions Per Second)

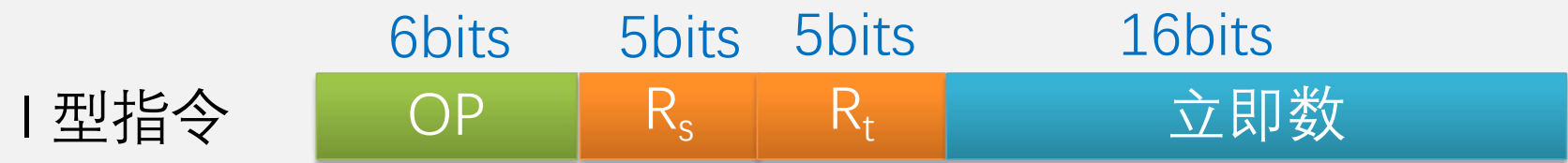


- ◆ MIPS (Microprocessor without Intellocked Pipeline Stages)是80年代初期由斯坦福大学Hennessy教授领导的研究小组研制成功;
- ◆ 属于精简指令集计算机
- ◆ MIPS指令集有MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, 和MIPS64多个版本;
- ◆ 早期主要用于嵌入式系统, 如Windows CE的设备, 路由器, 家用网关和视频游戏机, 现在已经在PC机、服务器中得到广泛应用

2. MIPS指令格式



◆ R_s , R_t 分别为第一、二源操作数; R_d 为目标操作数; shamt:移位位数



◆ 双目、Load/Store: R_s 和立即数是源操作数, R_t 为目标操作数;

◆ 条件转移: R_s , R_t 均为源操作数;



◆ 26位立即数作为跳转目标地址的部分地址



3. MIPS 使用的寄存器

寄存器名	寄存器编号	用途说明
\$s0	0	保存固定的常数0
\$at	1	汇编器的临时变量
\$v0 ~ \$v1	2 ~ 3	子函数调用返回结果
\$a0 ~ \$a3	4 ~ 7	函数调用参数1 ~ 3
\$t0 ~ \$t7	8 ~ 15	临时变量，函数调用时不需要保存和恢复
\$s0 ~ \$s7	16 ~ 23	函数调用时需要保存和恢复的寄存器变量
\$t8 ~ \$t9	24 ~ 25	临时变量，函数调用时不需要保存和恢复
\$k0 ~ \$k1	26 ~ 27	中断、异常处理程序使用
\$gp	28	全局指针变量(Global Pointer)
\$sp	29	堆栈指针变量(Stack Pointer)
\$fp	30	帧指针变量(Frame Pointer)
\$ra	31	返回地址(Return Address)

- ◆ 还有32个32位单精度浮点寄存器 $f0-f31$
- ◆ 还有2个32位乘、商寄存器 Hi 和Lo；乘法分别存放64位乘积的高、低 32位；除法时分别存放余数和商。



8.6 MIPS指令

4. MIPS 使用的寄存器

	6bits	5bits	5bits	5bits	5bits	6bits
指令	OP	rs	rt	rd	shamt	funct
add	0	Reg	Reg	Reg	0	32 ₁₀
sub	0	Reg	Reg	Reg	0	34 ₁₀
and	0	Reg	Reg	Reg	0	36 ₁₀
or	0	Reg	Reg	Reg	0	37 ₁₀
nor	0	Reg	Reg	Reg	0	39 ₁₀
sll	0	0	Reg	Reg	X	0 ₁₀
srl	0	0	Reg	Reg	X	2 ₁₀
jr	0	Reg			0	8 ₁₀

R型指令

add \$s1,\$s2,\$s3

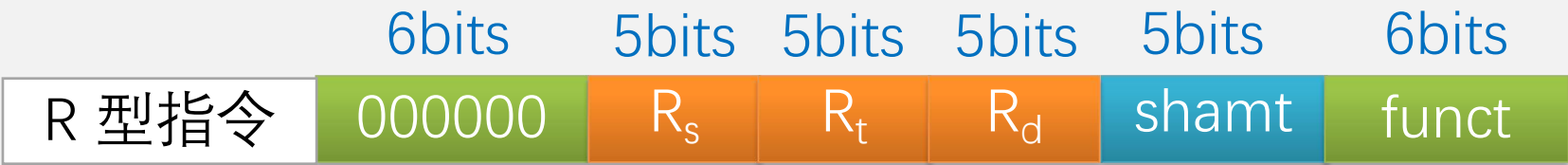


机器码 0x2538820

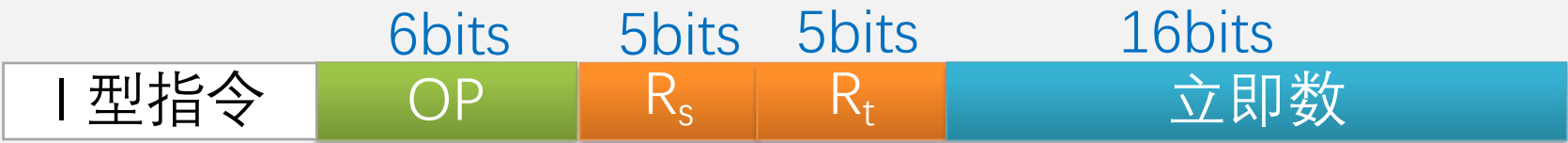
5. MIPS 的寻址方式

在MIPS32指令集中，**不设寻址方式说明字段**

◆ R型指令：由op和funct字段共同隐含说明当前的寻址方式；



◆ I型和J型指令：由op字段隐含说明当前指令使用的寻址方式。





5. MIPS 的寻址方式

◆ 立即数寻址 (Immediate addressing)



```
addi s1, s2, 10      ($s1 ← $s2 + E(10) )
```

注意：汇编格式和编码格式段的对应关系。



5. MIPS 的寻址方式

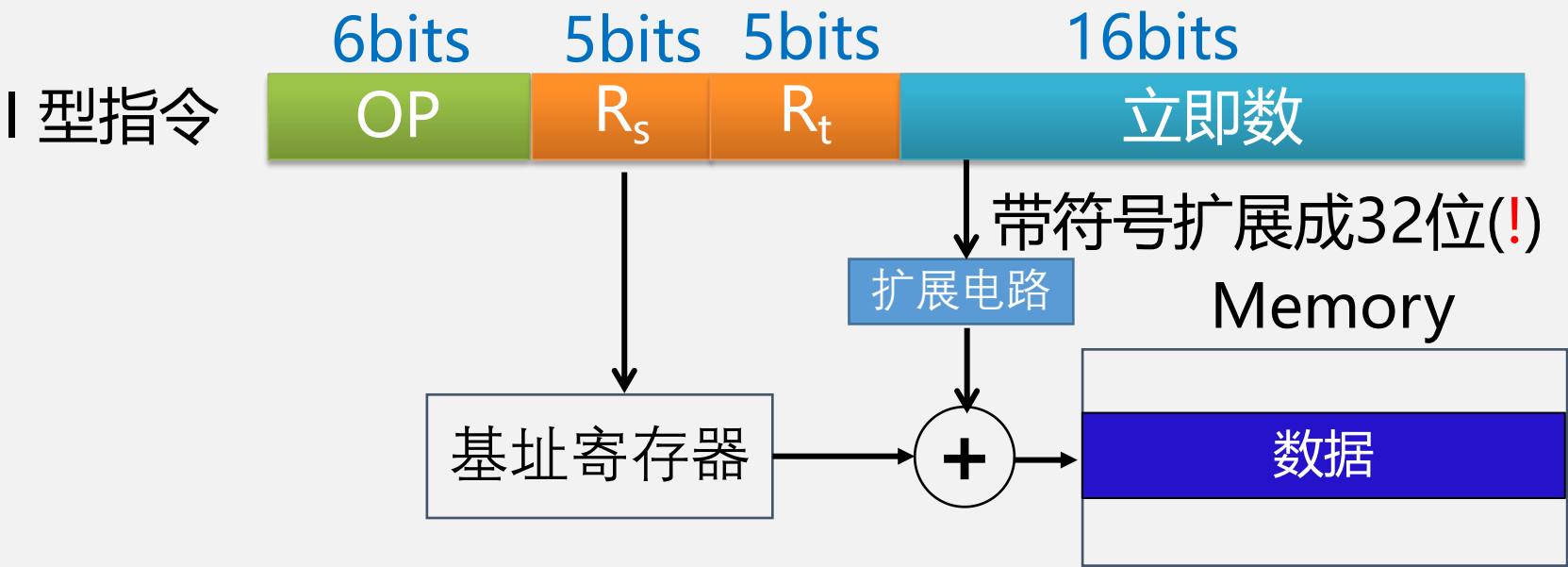
◆ 寄存器(直接)寻址(Register Addressing)



```
add $t0,$s1,$s2    ($t0=$s1+$s2)
```

5. MIPS 的寻址方式

◆ 基址寻址(Basic Addressing)



◆ 使用基址寻址的指令: lw ,sw, lh, sh, lb, lbu等

LB rt , offset (base)

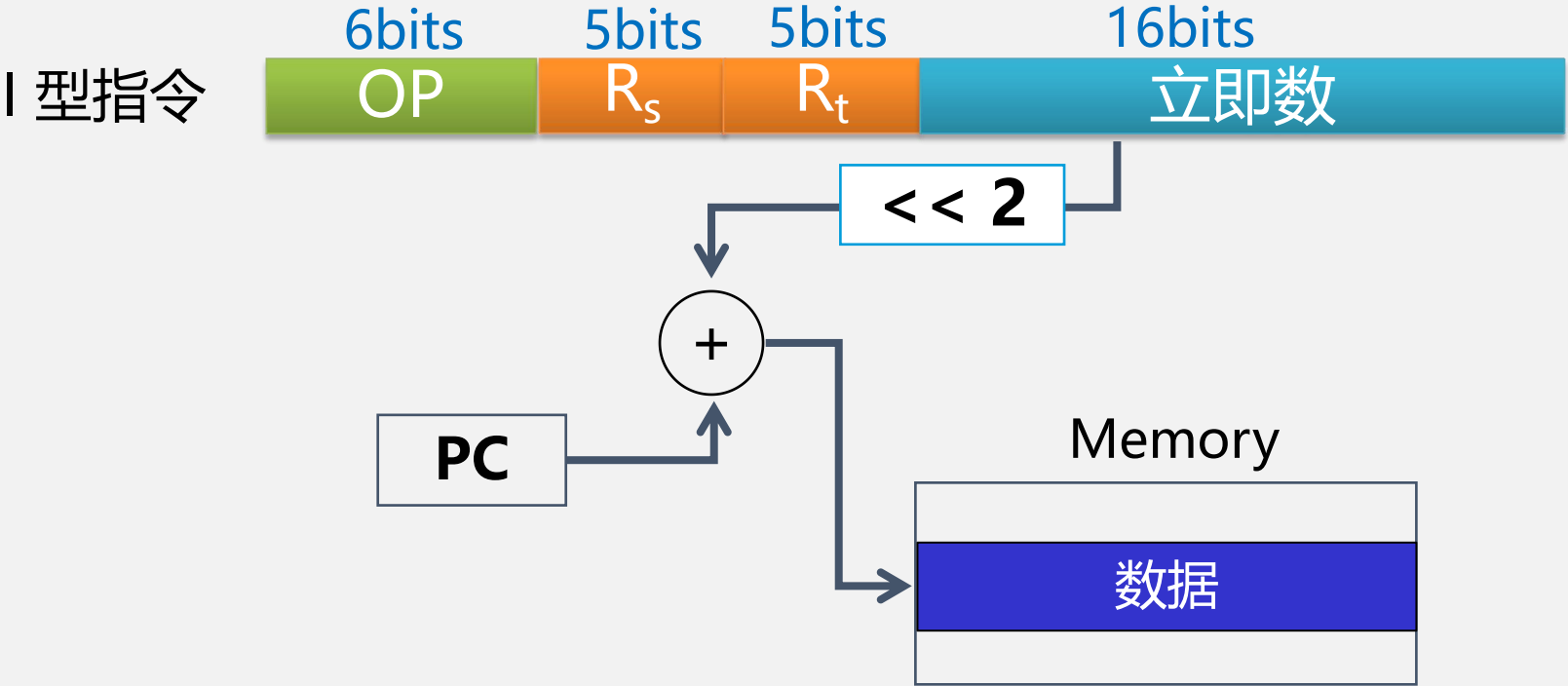


8.6 MIPS指令

5. MIPS 的寻址方式

◆相对寻址

```
if (GRP[rs] == GPR[rt])  
PC = PC + 4 + BranchAddr
```

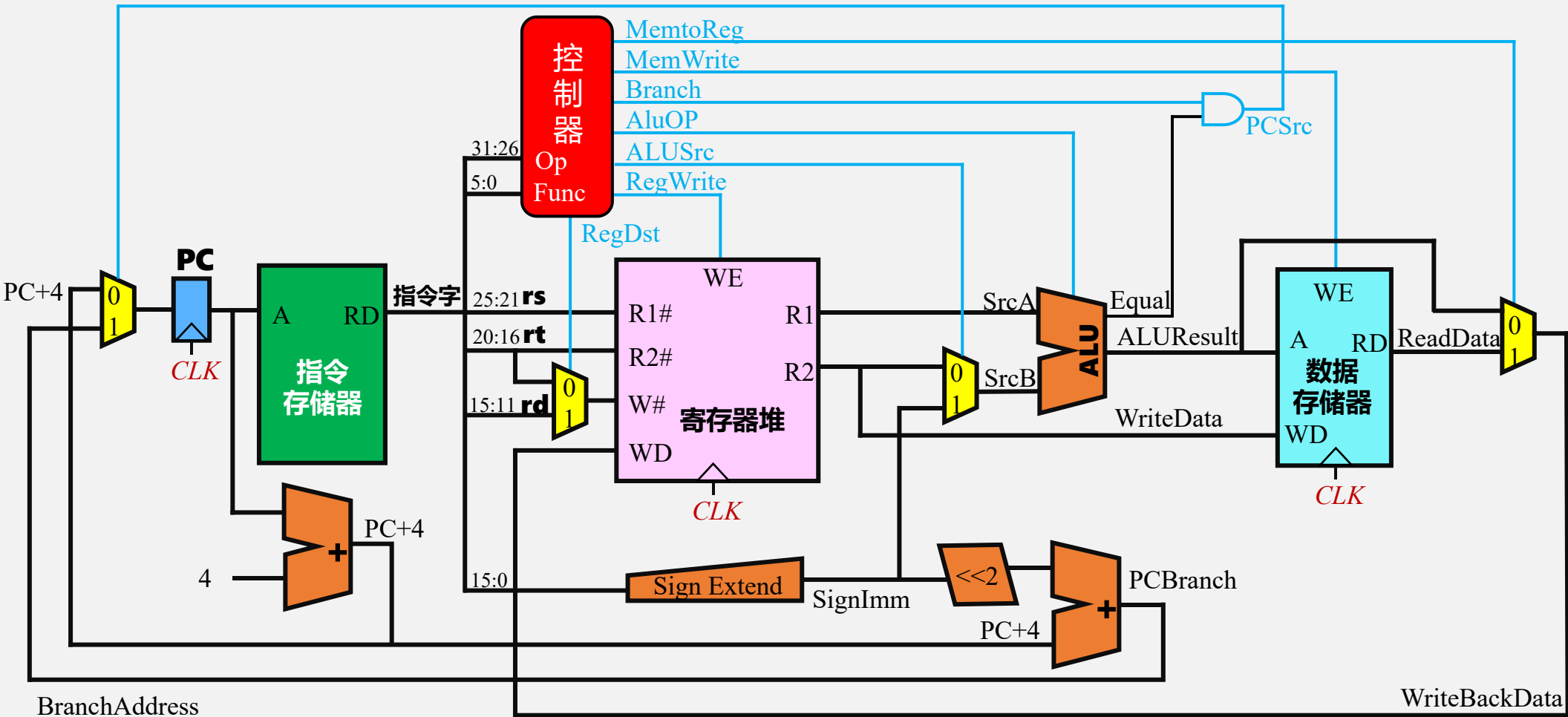


带符号扩展成32位后左移2位

◆使用相对寻址的指令：beq, bne

◆详细情况需查MIPS指令手册

6. MIPS CPU ISA的硬件体现



7 MIPS ISA的编程体现

如何用MIPS汇编程序段实现 C语言表达式 $a = b + c + d - e$ 的功能?

```
add $t0, $s1, $s2  # temp = b + c
```

```
add $t0, $t0, $s3  # temp = temp + d
```

```
sub $s0, $t0, $s4  # a = temp - e
```

你能给出对应的X86汇编语言实现程序段吗?



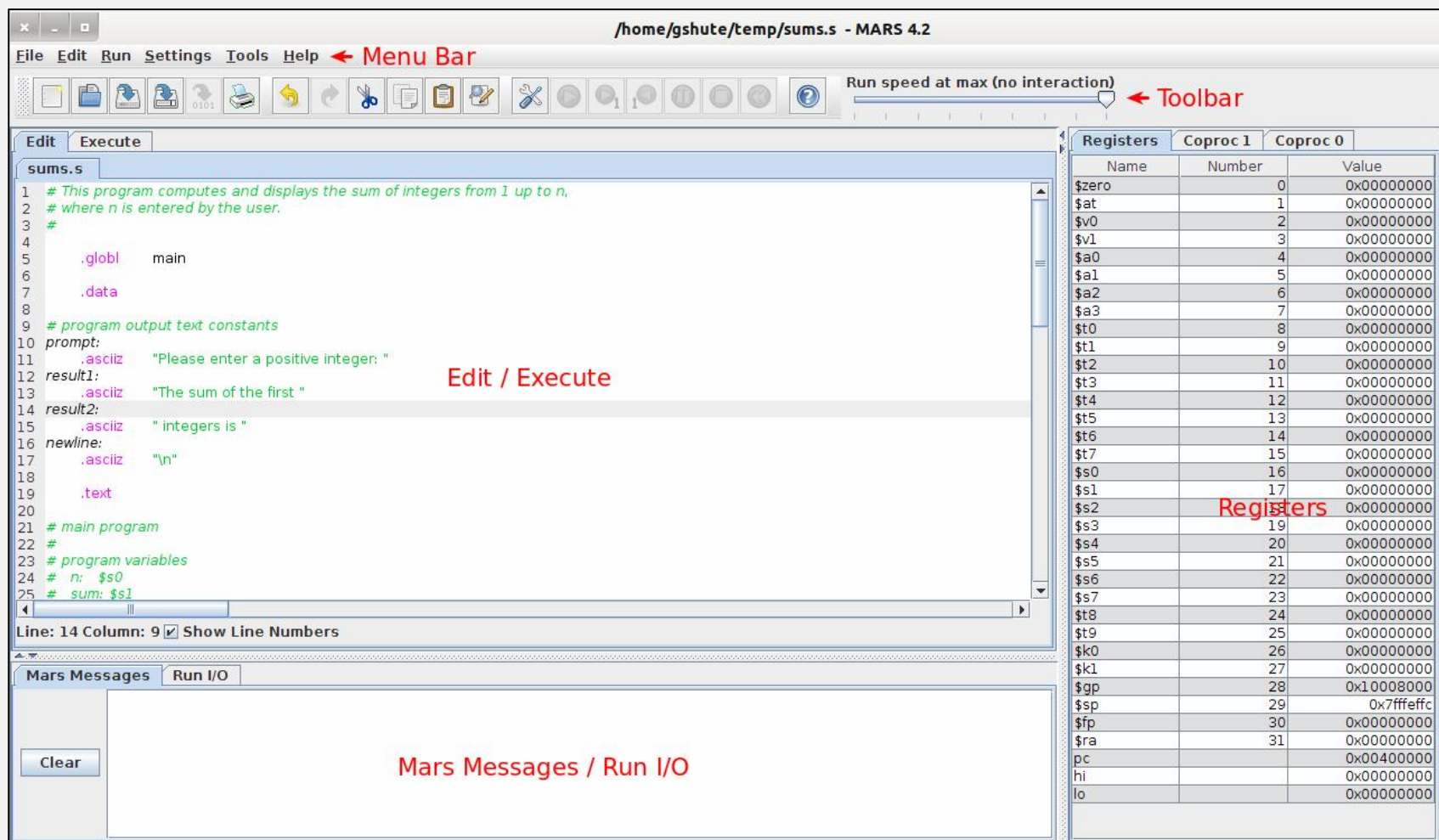
8.6 MIPS指令

7. MIPS ISA的编程体现

◆读写内存的操作实现

```
g = h + A[8];                                (in C)
lw  $t0,32($s3)    # $s3为A[0]地址    (in MIPS)
add $s1,$s2,$t0    # g=h+A[8] , $s2 保存变量h的值
sw  $t0,48($s3)    # store A[12]
```

8. MIPS 开源仿真-汇编器:MARS



1. RISC-v 的生态价值

- ◆ Linux是开源软件生态的基石。基于Linux，人们开发Python、LLVM（low level virtual machine）、GCC等完整的工具链，创造MySQL、Apache、Hadoop等大量开源软件，逐渐形成一个价值超过150亿美元的开源软件生态
- ◆ 在芯片设计领域，RISC-V有望像Linux那样成为计算机芯片与系统创新的基石。更重要的是要形成一个基于RISC-V的开源芯片设计生态，包括开源工具链、开源IP、开源SoC等等。
- ◆ 开源对中国的互联网产业的意义尤为重大，不仅提升互联网企业的技术研发能力，也大大降低了互联网产业创新的门槛，如今3-5位开发人员在几个月时间里就能快速开发出一个互联网应用。

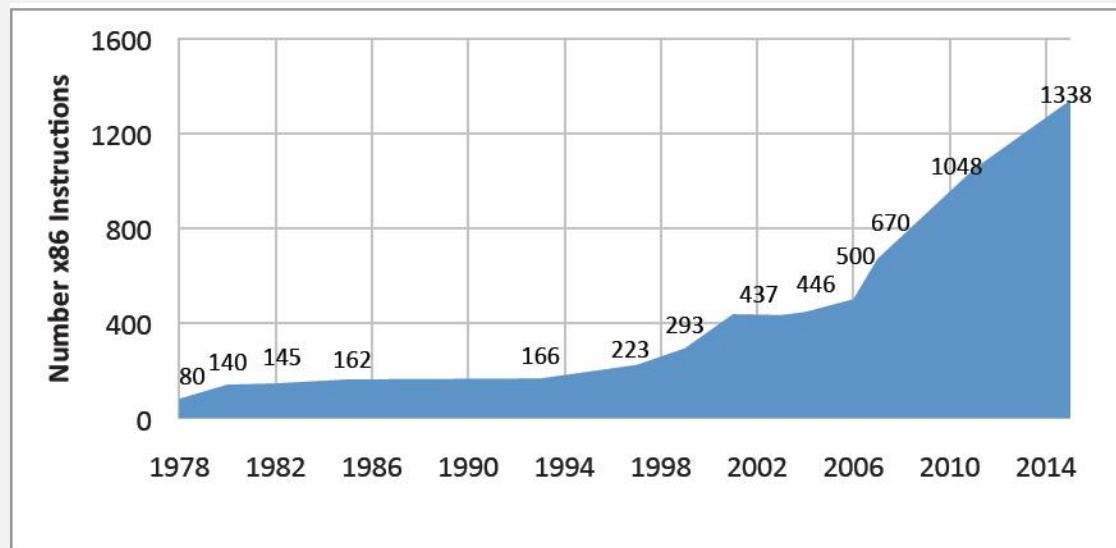
2. RISC-v 的特征(为什么需要它)

RISC-V的目标是成为一个通用的指令集架构 (ISA)

- ◆适应包括从嵌入式控制器，到最快的高性能计算机等各种规模的处理器;
- ◆能兼容各种流行的软件栈和编程语言;
- ◆适应所有实现技术，包括FPGA、ASIC、全定制芯片;
- ◆对所有微体系结构样式都有效:微编码或硬连线控制;顺序或乱序执行流水线;单发射或超标量等等;
- ◆支持广泛的专业化，成为定制加速器的基础;
- ◆稳定的、基础的指令集架构。

3. 模块化与增量型ISA

◆ 计算机体系结构的传统方法是增量ISA，为了保持向后兼容，新处理器既要实现新的ISA扩展，还必须实现过去的所有扩展。



◆ **X86**在**1978**年诞生时有**80**余条指令，到**2015**年增长到**1338**条(16倍)，并且仍在增长。**2015**年在英特尔有过**3600**条 的统计结果[Rodgers and Uhlig 2017]，平均每4天增长一条。

3. 模块化与增量型ISA

- ◆ RISC-V不同于传统的ISA，它基于模块化理念，基于固定的核心模块RV32I，运行一个完整软件栈，为编译器、操作系统和汇编语言程序员提供稳定支撑。
- ◆ 模块化来源于可选的标准扩展，根据应用程序需要，硬件可以包含或不包含这些扩展。

4. RV32I 指令格式(六类)

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0				
funct7				rs2			rs1		funct3		rd			opcode		R-type		
imm[11:0]						rs1		funct3		rd			opcode		I-type			
imm[11:5]				rs2			rs1		funct3		imm[4:0]			opcode		S-type		
imm[12]		imm[10:5]			rs2			rs1		funct3		imm[4:1]		imm[11]		opcode		B-type
imm[31:12]									rd			opcode			U-type			
imm[20]		imm[10:1]			imm[11]		imm[19:12]			rd			opcode			J-type		

- ◆ RISC-V 将源寄存器rs1， rs2和目标寄存器rd固定位置， 简化指令译码；
- ◆ 立即数分散在不同位置， 且排列做了一些移位轮换， 符号位固定在第31位。将指令信号的扇出和立即数多路复用的成本降低近2倍， 简化了低端实现中的数据通路逻辑， 可加速符号扩展电路， 与译码并行

1. ARM 指令格式

<opcode> {<cond>} {S} <Rd> ,<Rn>{,<operand2>}

其中 **<>** 号内的项是必须的，**{ }** 号内的项是可选的。

opcode: 指令助记符; **cond**: 执行条件;

S: 是否影响 CPSR 寄存器的值;

Rd: 目标寄存器; **Rn**: 第1个操作数的寄存器;

operand2: 第2个操作数;



8.8 ARM 指令

1. ARM 指令格式

$$\langle \text{opcode} \rangle \{ \langle \text{cond} \rangle \} \{ S \} \quad \langle \text{Rd} \rangle, \langle \text{Rn} \rangle \{, \langle \text{operand2} \rangle \}$$

其中 $\langle \rangle$ 号内的项是必须的， $\{ \}$ 号内的项是可选的。

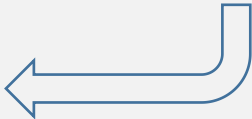
灵活的使用第2个操作数“**operand2**”能够提高代码效率。

- ◆ **immed_8r**——常数表达式;
- ◆ **Rm**——寄存器方式;
- ◆ **Rm,shift**——寄存器移位方式;



1. ARM 指令格式

<opcode> {<cond>} {S} <Rd> ,<Rn>{,<operand2>}



操作码	条件助记符	标志	含义
0000	EQ	Z=1	相等
0001	NE	Z=0	不相等
0010	CS/HS	C=1	无符号数大于或等于
0011	CC/LO	C=0	无符号数小于
0100	MI	N=1	负数
0101	PL	N=0	正数或零
0110	VS	V=1	溢出
0111	VC	V=0	没有溢出
1000	HI	C=1,Z=0	无符号数大于
1001	LS	C=0,Z=1	无符号数小于或等于
1010	GE	N=V	有符号数大于或等于
1011	LT	N!=V	有符号数小于
1100	GT	Z=0,N=V	有符号数大于
1101	LE	Z=1,N!=V	有符号数小于或等于
1110	AL	任何	无条件执行 (指令默认条件)
1111	NV	任何	从不执行(不要使用)

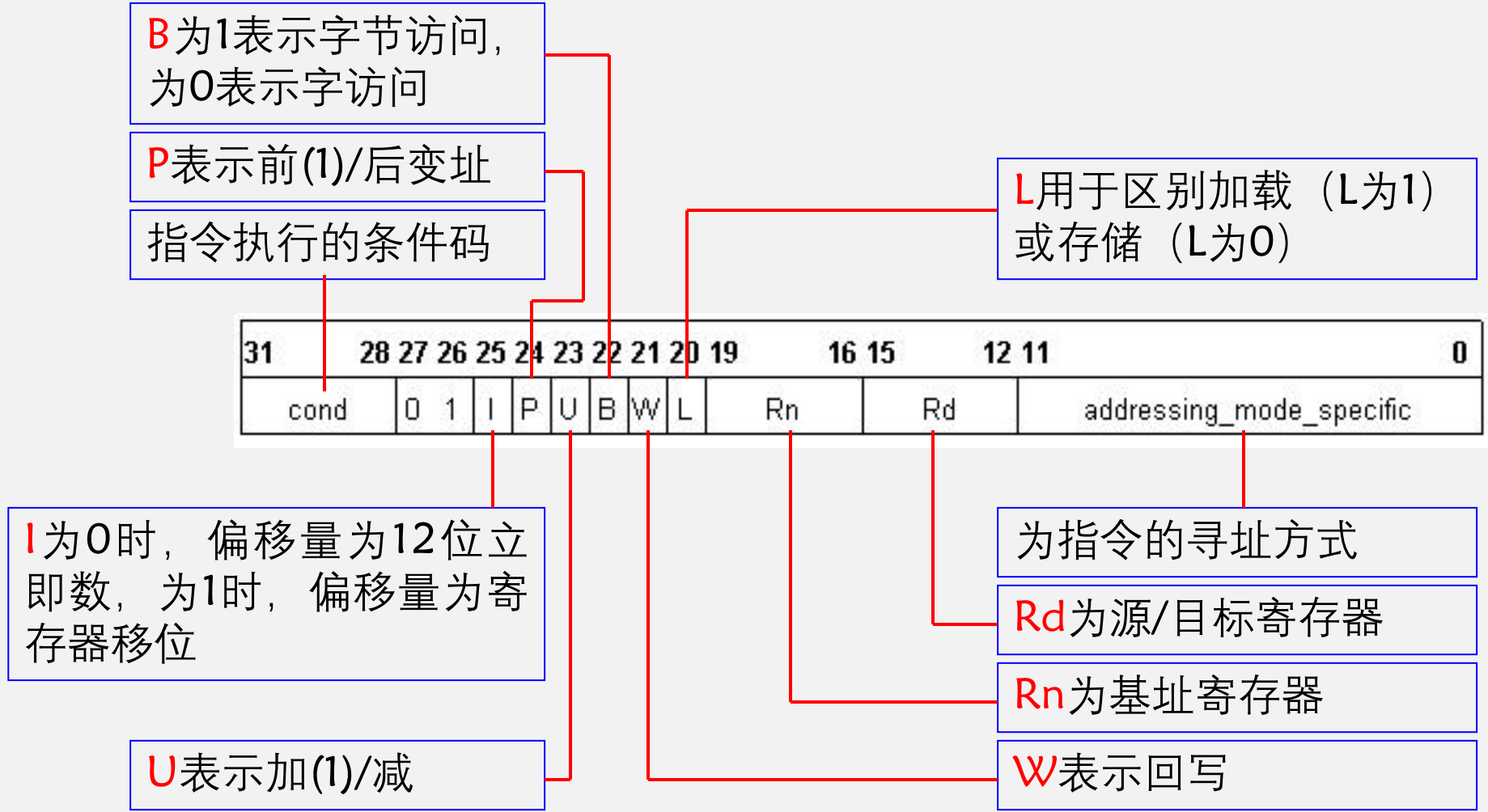
- ◆使用条件码“cond” 可实现高效的逻辑操作， 提高代码效率。
- ◆所有ARM指令都可 以条件执行。若指令不标明条件代码， 将默认为无条件（AL） 执行。

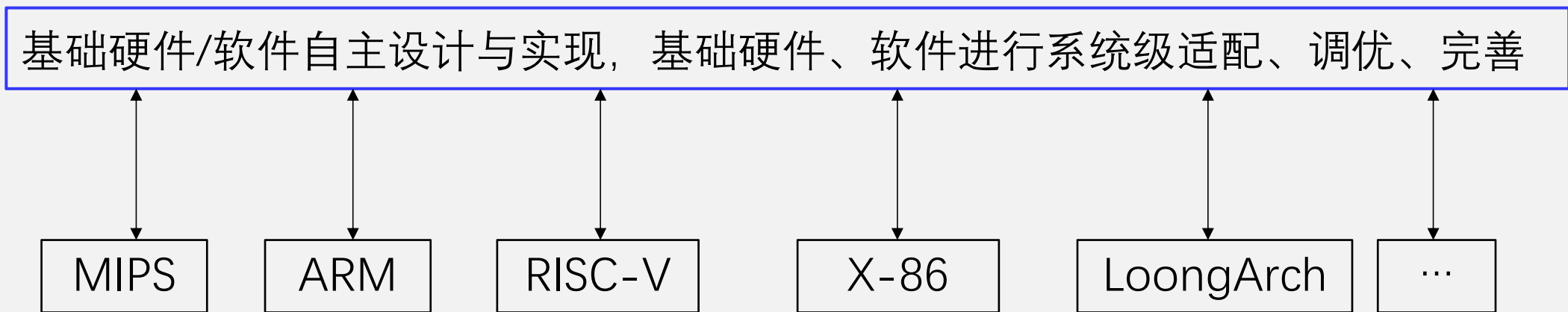


8.8 ARM 指令

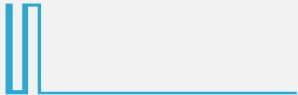
2. ARM 指令应用举例

LDR和STR——字和无符号字节加载/存储指令编码





积极投身我国信息产业自主可控国家战略，积极参加华为开发者社区、华为开源社区



第二部分完