计算机组成原理

指令系统（2）

王浩宇,教授

haoyuwang@hust.edu.cn

https://howiepku.github.io/

本节目录

- 指令系统的发展
- 指令格式
- 指令和数据的寻址方式
- 指令类型
- 指令格式设计
- CISC和RISC
- 指令系统举例：MIPS

指令类型

- **指令分类与基本指令类型**
  - 数据传送类指令
    - 一般传送指令：　MOV　AX，BX
    - 数据交换指令：　XCHG
    - 堆栈操作指令：　PUSH，POP
  - 运算类指令
    - 算术运算指令：加、减、乘、除以及加1、减1、比较
    - 逻辑运算指令：
    - 移位指令
  - 程序控制类指令
    - 程序控制类指令用于控制程序的执行方向，并使程序具有测试、分析与判断的能力。
  - 输入和输出指令、字符串处理指令、特权指令、其他指令

指令格式设计的主要内容

- 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
- 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

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种寻址操作，完成60种操作，各寻址方式均可在2K主存范围内取得操作数，并可在1K范围内保存运算结果。问应采用什么样的指令格式？指令字长最少应为多少位？执行一条指令最多要访问多少次主存？

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，并给出不同寻址方式下的寻址范围(设PC寄存器16位，变址寄存器16位)

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？

$$((2^6-15)\times 2^6-34)\times 2^6$$

指令格式设计课堂练习

采用将操作码字段扩展到没有使用的地址码字段的指令格式设计方案的主要目的是（ ）

- A. 减少指令长度
- B. 充分利用地址字段，提高指令效率
- C. 保持指令长度不变，增加指令数量
- D. 减少地址码数量

指令格式设计课堂练习

某计算机采用32位单字长二地址指令，每个地址码为12位（含寻址方式字段），若已经250条二地址指令，则还可以定义多少条单地址指令（　）24K

$(2^8-250)2^{12}=24K$

A. 4K

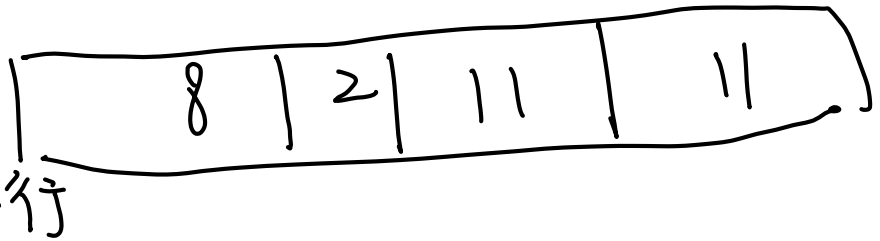
B. 8K

C. 16K

D. 24K

指令格式设计课堂练习

某计算机字长32位，内存空间为4M，采用单字长二地址指令，操作码长度固定。要求支持200条指令，要求支持的寻址方式为4种，下列寻址方式中，可支持访问整个存储空间的寻址方式是



- A. 直接寻址
- B. 寄存器寻址 ?? 为什么不行
- C. 寄存器间接寻址 √
- D. 变址寻址 √

指令格式设计课堂练习

某计算机机器字长和存储字长32位，采用双字长，下列描述中正确的是（ ）

- A. 采用双字长指令有利于支持更多的指令和设计位数更多的地址字段
- B. 顺序寻址方式下，PC <---- (PC) + 8
- C. 在该计算机中完成RS型指令，至少需要访问内存3次
- D. 采用双字长指令有利于支持更多的寻址方式

本节目录

- 指令系统的发展
- 指令格式
- 指令和数据的寻址方式
- 指令类型
- 指令格式设计
- CISC和RISC
- 指令系统举例：MIPS

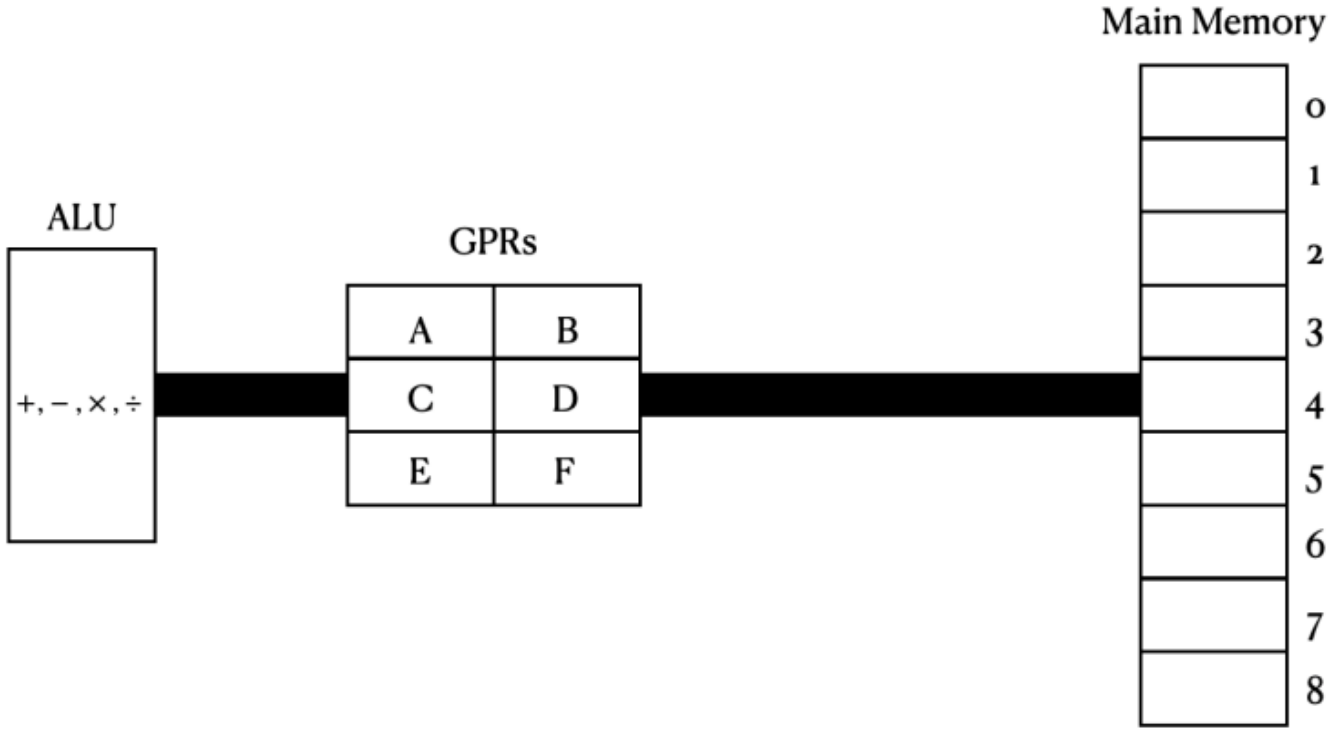
Archi-tecture	Bits	Version	Intro-duced	Max # operands	Type	Design	Registers (excluding FP/vector)	Instruction encoding	Branch evaluation	Endian-ness	Extensions
6502	8		1975	1	Register–Memory	CISC	3	Variable (8- to 24-bit)	Condition register	Little	
6809	8		1978	1	Register–Memory	CISC	3	Variable (8- to 32-bit)	Condition register	Big	
680x0	32		1979	2	Register–Memory	CISC	8 data and 8 address	Variable	Condition register	Big	
8080	8		1974	2	Register–Memory	CISC	7	Variable (8 to 24 bits)	Condition register	Little	
8051	32 (8→32)		1977?	1	Register–Register	CISC	32 in 4-bit 16 in 8-bit 8 in 16-bit 4 in 32-bit	Variable (8-bit to 128 bytes)	Compare and branch	Little	
x86	16, 32, 64 (16→32→64)		1978	2 (integer) 3 (AVX)[a] 4 (FMA4)[4]	Register–Memory	CISC	8 (+ 4 or 6 segment reg.) (16/32-bit) 16 (+ 2 segment reg. gs/cs) (64-bit) 32 with AVX-512	Variable (8086 ~ 80386: variable between 1 and 6 bytes /w MMU + intel SDK, 80486: 2 to 5 bytes with prefix, pentium and onward: 2 to 4 bytes with prefix, x64: 4 bytes prefix, third party x86 emulation: 1 to 15 bytes w/o prefix & MMU . SSE/MMX: 4 bytes /w prefix AVX: 8 Bytes /w prefix)	Condition code	Little	x87, IA-32, MMX, 3DNow!, SSE, SSE2, PAE, x86-64, SSE3, SSSE3, SSE4, BMI, AVX, AES, FMA, XOP, F16C
Alpha	64		1992	3	Register–Register	RISC	32 (including "zero")	Fixed (32-bit)	Condition register	Bi	MVI, BWX, FIX, CIX
ARC	16/32/64 (32→64)	ARCv3[5]	1996	3	Register–Register	RISC	16 or 32 including SP user can increase to 60	Variable (16- or 32-bit)	Compare and branch	Bi	APEX User-defined instructions
ARM/A32	32	ARMv1–v9	1983	3	Register–Register	RISC	15	Fixed (32-bit)	Condition code	Bi	NEON, Jazelle, VFP, TrustZone, LPAE
Thumb/T32	32	ARMv4T-ARMv8	1994	3	Register–Register	RISC	7 with 16-bit Thumb instructions 15 with 32-bit Thumb-2 instructions	Thumb: Fixed (16-bit), Thumb-2: Variable (16- or 32-bit)	Condition code	Bi	NEON, Jazelle, VFP, TrustZone, LPAE

CISC VS RISC

- **复杂指令系统计算机**
（Complex Instruction Set Computer，CISC）
X86，X86-64，AMD64等

- **精简指令系统计算机**
（Reduced Instruction Set Computer，RISC）
ARM，MIPS，SPARC等

- 举个栗子



CISC VS RISC

- 指令系统的两个不同方向

增强原有指令的功能，设置更加复杂的新指令，实现软件功能的硬件化

减少指令种类和简化指令功能，提高指令的运行速度

CISC复杂指令系统计算机

- 随着VLSI技术的发展，硬件成本下降，软件成本上升，促使人们在指令系统中增加更多、更复杂的指令。
- (1) 更好的支持高级语言
  - 增加语义接近高级语言的指令
- (2) 简化编译
  - 机器指令的语义与高级语言的语义接近时，编译器设计相对简单，编译效率也会提高
- (3) 满足系列计算机软件向后兼容的需求
  - 为了程序兼容，只能扩充而不能减少原来的指令
- (4) 对操作系统的支持
  - 操作系统功能越来越复杂，需要指令系统提供相应功能指令的支持
- (5) 为满足寻址方式的需要，必须设计多种寻址方式

CISC复杂指令系统计算机

- 指令系统庞大复杂，指令数目一般多达两三百条
- 寻址方式多，指令格式多
- CISC指令格式长短不一，需要不同的时钟周期来完成
  - 执行较慢的指令影响系统的性能
  - 不利于采用先进的指令级并行技术
- 指令使用频度不均衡
  - 80:20 rule (Pareto)
- 大量复杂指令的控制逻辑不规整，不适于VLSI工艺
  - 微程序的使用反而制约了速度提高（微码的存控速度比CPU慢5-10倍）
- 软硬功能分配
  - 可划分的粒度变大，灵活性小

RISC精简指令系统计算机

- 基本思想：针对CISC指令系统指令种类太多、指令格式不规范、寻址方式太多的缺点，通过减少指令种类、规范指令格式和简化寻址方式来方便处理器内部的并行处理，从而大幅度地提高处理器的性能
  - 通过简化计算机指令功能，使指令的平均执行周期减少，从而提高计算机的工作主频，同时大量使用通用寄存器来提高指令执行的速度

RISC精简指令系统计算机

- 选取使用频率最高的一些简单指令，复杂指令的功能由简单指令的组合来实现
- RISC一定采用指令流水线技术，大部分指令的操作都在一个时钟周期内完成
- 只有Load/Store（取数/存数）指令访存，其余指令都在寄存器之间进行
- 采用简单的格式和寻址方式，指令长度固定
- 面向寄存器的结构
- 以硬布线控制为主，不用或少用微程序控制
- 注重编译优化工作，力求有效地支持高级语言程序

CISC VS RISC

	CISC	RISC
指令系统	复杂，庞大	简单，精简
指令数目	一般大于200条	一般小于100条
指令字长	不固定	固定
可访存指令	不加限制	只有Load/Store指令
各种指令执行时间	相差较大	绝大多数在一个周期内完成
各种指令使用频度	相差很大	都比较常用
通用寄存器数量	较少	多
目标代码	难以用优化编译生成高效的目标代码程序	采用优化的编译程序，生成代码较为高效
控制方式	绝大多数为微程序控制	绝大多数为组合逻辑控制
指令流水线	可以通过一定方式实现	必须实现

However… the current debate of CISC and RISC

- Cross market appearance of both ISAs
  - e.g., ARM-based Servers, and X86-based mobile devices
- Performance differences are generated by ISA-independent microarchitecture differences
- The energy consumption is also ISA-independent
  - Energy use is also primarily impacted by the design choice and not by the ISA
- One ISA is not fundamentally more efficient

Table 10. Summary of Findings.

	#	Finding	Support	Representative Data: A8/Atom
Performance	1	Large performance gaps exist	Fig-2	2× to 997×
	2	Cycle-count gaps are less than 2.5× (A8 to Atom, A9 to i7)	Fig-3	≤ 2.5×
	3	x86 CPI < ARM CPI: x86 ISA overheads hidden by μarch	Fig-3 & 4	A8: 3.4 Atom: 2.2
	4	ISA performance effects indistinguishable between x86 and ARM	Table-6 Fig-5 & 6	inst. mix same short x86 insts
	5	μarchitecture, not the ISA, responsible for performance differences	Table-8	324× Br MPKI 4× L2-misses
	6	Beyond micro-op translation, x86 ISA introduces no overheads over ARM ISA	Table-7	
Power	1	x86 *implementations* draw more power than ARM *implementations*	Fig-11	Atom/A8 raw power: 3×
	2	Choice of power or perf. optimization impacts power use more than ISA	Fig-12	Atom/A8 power @1 GHz: 0.6×
	3	Energy use primarily a design choice; ISA's impact insignificant	Fig-13	Atom/A8 raw energy: 0.8×
Trade-offs	1	High-perf processors require more power than lower-performance processors	Fig-14	A8/A9: 1.8× i7/Atom: 10.9×
	2	It is the μ-architecture and design methodology that really matters	Fig-15	ED: i7@2GHz<A9 A15 best for ED i7 best for $ED^{1.4}$

6. Conclusions

In this work, we revisit the RISC vs. CISC debate considering contemporary ARM and x86 processors running modern workloads to understand the role of ISA on performance, power, and energy. During this study, we encountered infrastructure and system challenges, missteps, and software/hardware bugs. Table 9 outlines these issues as a potentially useful guide for similar studies. Our study suggests that whether the ISA is RISC or CISC is irrelevant, as summarized in Table 10, which includes a key representative quantitative measure for each analysis step. We reflect on whether there are certain metrics for which RISC or CISC matters, and place our findings in the context of past ISA evolution and future ISA and microarchitecture evolution.

Considering area normalized to the 45nm technology node, we observe that A8's area is 4.3mm^2, AMD's Bobcat's area is 5.8mm^2, A9's area is 8.5 mm^2, and Intel's Atom is 9.7 mm^2 [4, 25, 27]. The smallest, the A8, is smaller than Bob-

"Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures", HPCA 2013

本节目录

- 指令系统的发展
- 指令格式
- 指令和数据的寻址方式
- 指令类型
- 指令格式设计
- CISC和RISC
- **指令系统举例：MIPS**

MIPS指令概述

- MIPS (Microprocessor without Intellocked Pipleline 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； **Million Instructions Per Second**
- 属于精简指令集计算机RISC(Reduced Instruction Set Computer)；

复杂指令集计算机CISC(Complex Instruction Set Computer)；

- 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

MIPS指令集的特点

MIPS指令集有以下特点：

- 简单的Load/Store结构
- 易于流水线CPU设计
- 易于编译器开发
- MIPS指令的寻址方式非常简单，每条指令的操作也非常简单

MIPS指令格式概述

- 只有三种指令格式

R 型指令

- ♦ Rs,Rt分别为第一、二源操作数；Rd为目标操作数；

I 型指令

- ♦ 双目、Load/Store: Rs和立即数是源操作数，Rt为目标操作数；
- ♦ 条件转移：Rs,Rt均为源操作数；

J 型指令

- ♦ 26位立即数作为跳转目标地址的部分地址

MIPS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)

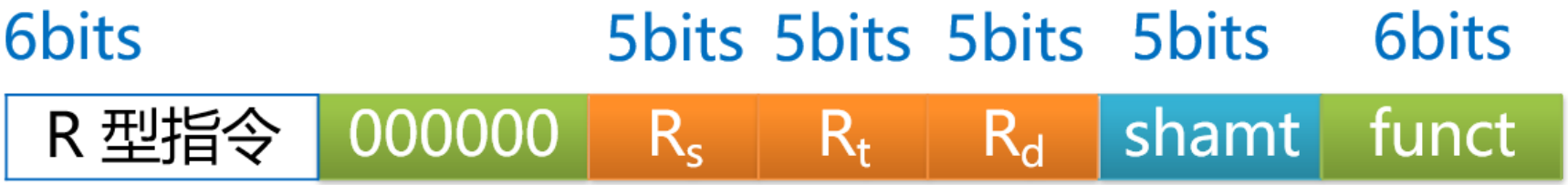
- 还有32个32位单精度浮点寄存器f_0-f_{31}

- 还有2个32位乘、商寄存器 H_i 和L_0；乘法法分别存放64位乘积的高、低 32位；除法时分别存放余数和商。

MIPS32指令集寻址方式

- 在MIPS32指令集中，不单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。

MIPS32指令集寻址方式

- **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。

OP	R_s	R_t	立即数
6bits	$s2	$s1	16bits

MIPS32指令集寻址方式

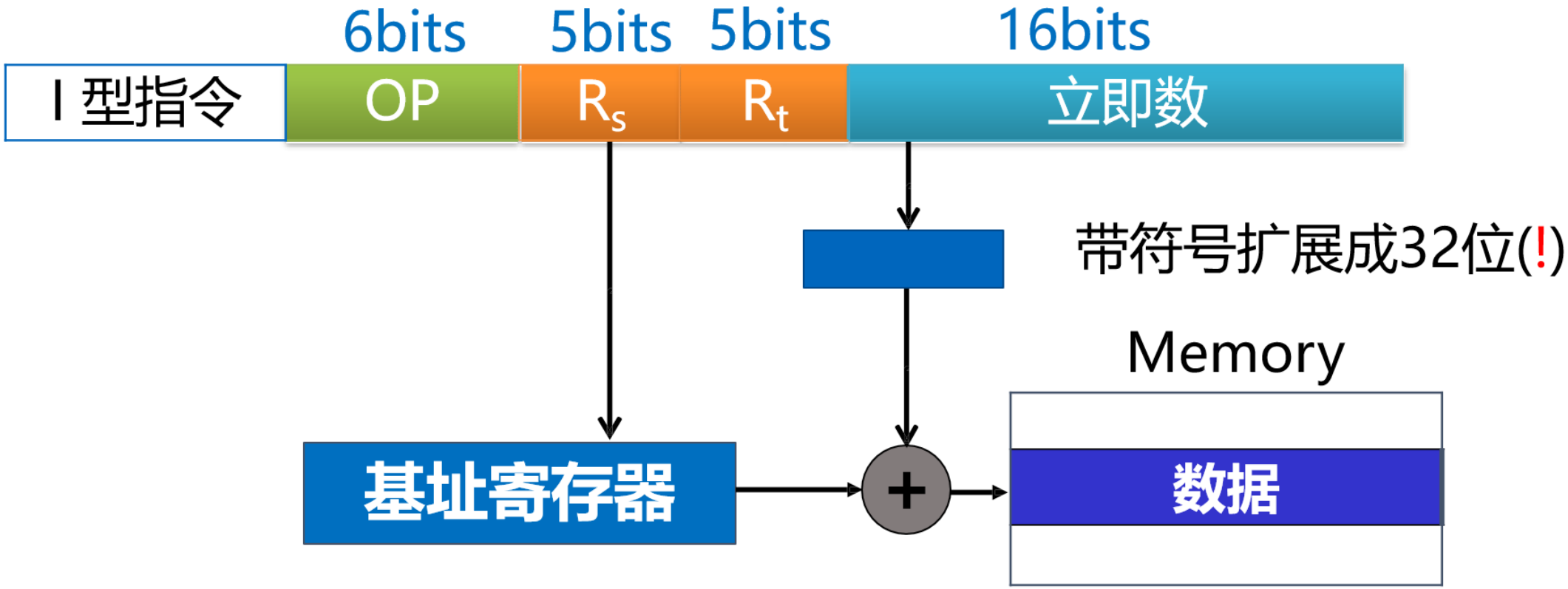
- **寄存器直接寻址(Register Addressing)**

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

000000	R_s	R_t	R_d	shamt	funct
6bits	$s1	$s2	$t0	00000	6bits

MIPS32指令集寻址方式

- **基址寻址(Basic Addressing)**

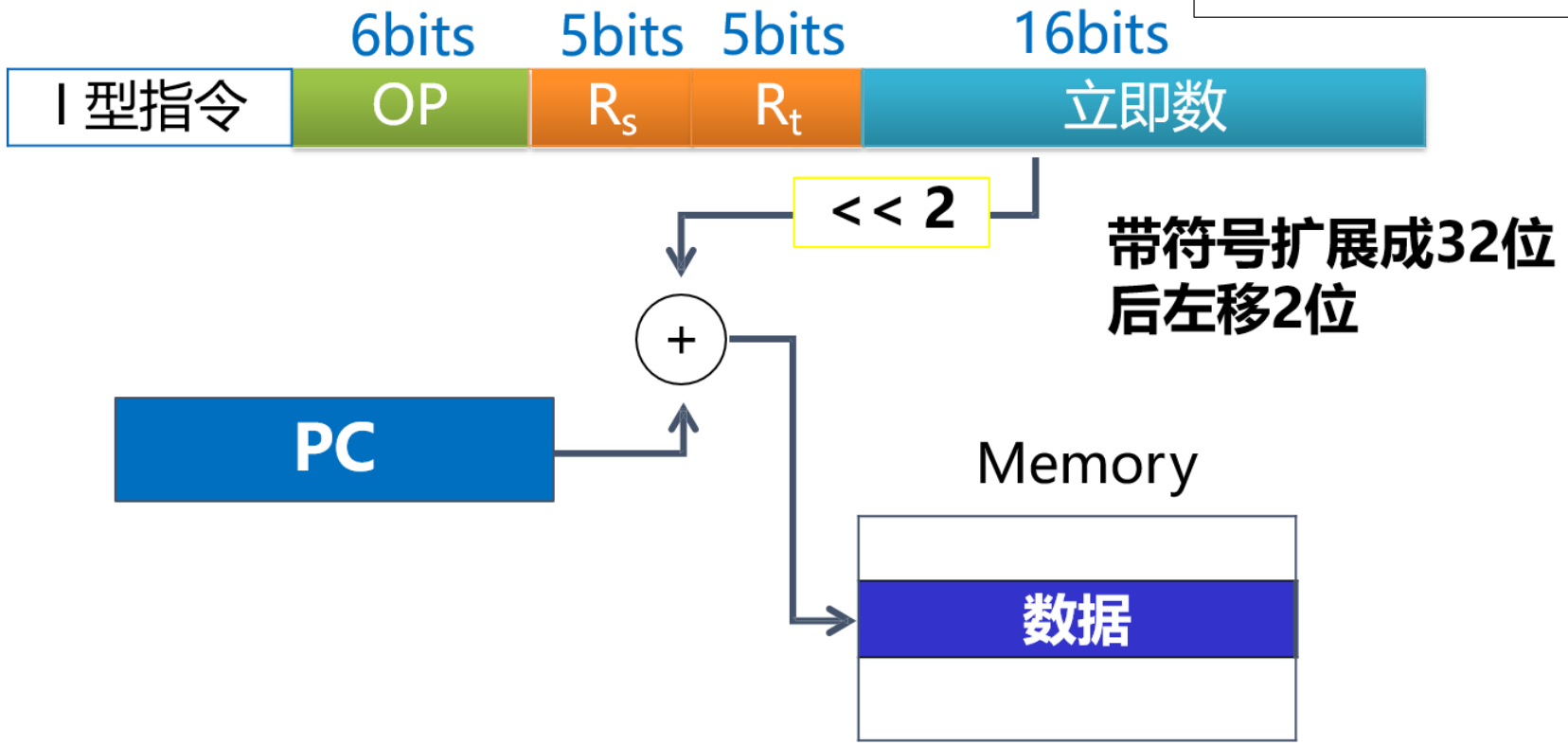


  - 使用基址寻址的指令：lw ,sw, lh, sh, lb, lbu等

MIPS32指令集寻址方式

- **相对寻址**

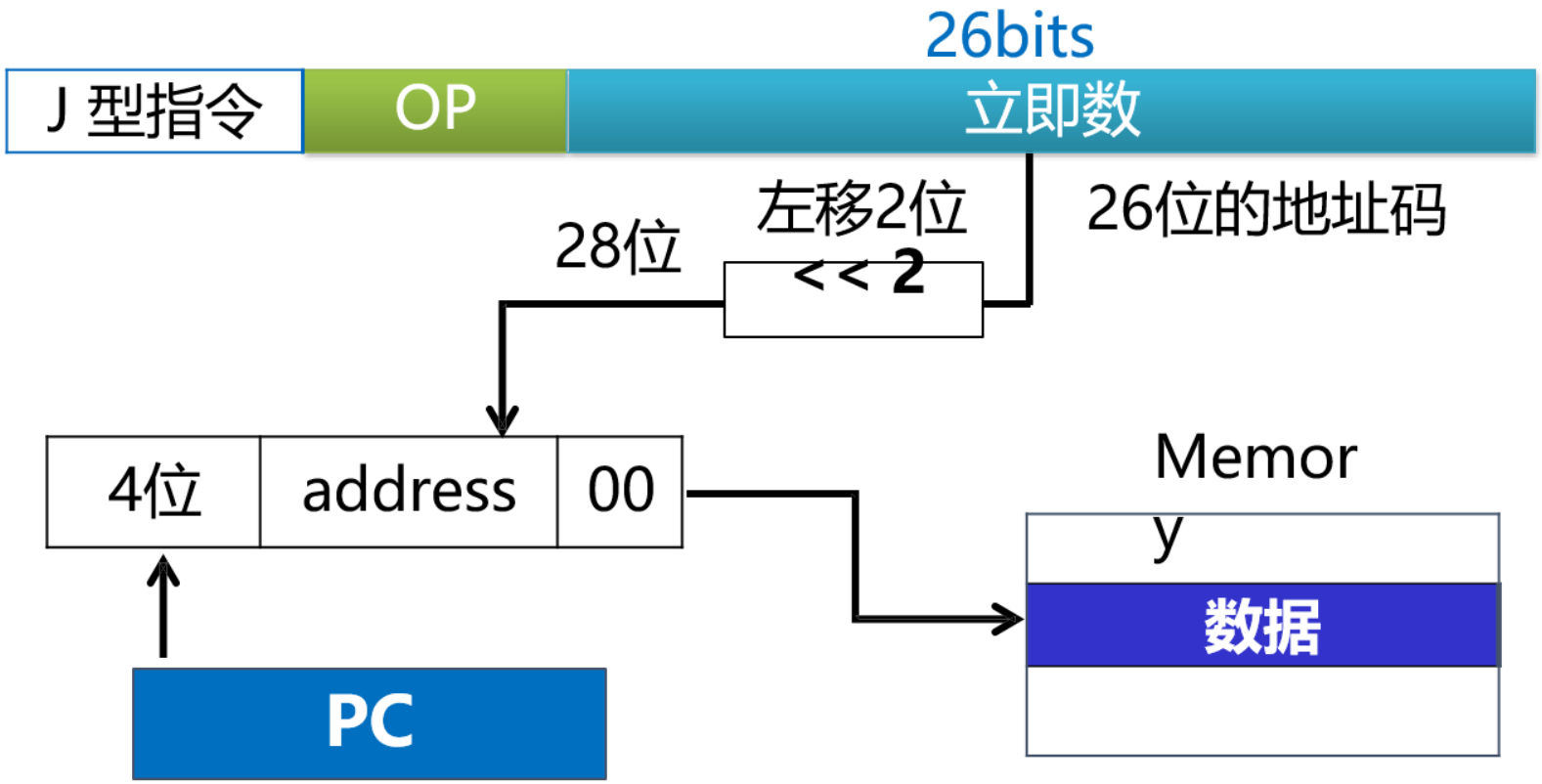
```
if (GRP[rs] == GPR[rt])
PC = PC + 4 + BranchAddr
```



- 使用相对寻址的指令：beq, bne

MIPS32指令集寻址方式

- 伪直接寻址(页面寻址)



- 使用伪直接寻址的指令：j, jal

MIPS32指令集寻址方式总结

- MIPS指令寻址方式比较少，只有5种寻址方式
- R型指令的寻址方式只有寄存器寻址；I型指令的寻址方式有寄存器寻址、立即数寻址、相对寻址；J型指令只有一种伪直接寻址

序号	寻址方式	有效地址EA/操作数S	指令示例
1	立即数寻址	S = imm	addi, rt, rs, imm
2	寄存器寻址	S = R[rt]	add rd rs rt
3	寄存器相对寻址/基址寻址	EA = R[rs]+imm	lw rt, imm(rs)
4	相对寻址	EA = PC+4+Disp	beq rs rt imm
5	伪直接寻址	EA = $\{(PC+4)_{31:28}$, address,00$\}$	j address

MIPS32指令集寻址方式练习

1 某采用相对寻址的MIPS Ⅰ型指令，其立即数字段的值内容为 1110000011100011，则计算操作数有效地址时，与PC内容相加的偏移量是（ ）

- A. 1111111111111111 1000001110001100 为什么左移2位??
- B. 1111111111111111 1110000011100011
- C. 0000000000000000 1110000011100011
- D. 0000000000000000 1000001110001100

MIPS32指令集寻址方式练习

2 下列关于MIPS特点的描述中，正确的是（ ）

- [x] A. 寻址方式简单
- [x] B. 属于精减指令集计算机RISC
- [x] C. 只有Load/Store指令才访问存储器
- [x] D. 寄存器数量较多

MIPS32指令集寻址方式练习

3 MIPS指令分为R、 I、 J三种类型的指令，下列关于MIPS指令格式的描述中，正确的是（ ）

- [x] A. 指令长度固定
- [x] B. 操作码字段长度固定
- [x] C. 指令中寄存器字段长度固定
- [ ] D. 立即数字段长度固定

MIPS32指令集寻址方式练习

4 下列关于 MIPS 寻址的下列描述中， 正确的是（ ）

- [x] A. 相对寻址时，将32位地址左移两位的目的是为了实现按32位整数边界对齐存放
- [x] B. 伪直接寻址时，26位直接地址左移两位的目的是为了使32位地址的低两位为0，实现按32位的整数边界对齐存放
- [ ] C. 立即数寻址时，指令中的立即数直接送给指令中指定的寄存器
- [x] D. MIPS指令中不单独设置寻址方式字段

MIPS指令详解

- R型指令

操作数和保存结果均通过寄存器进行；

- op：操作码，所有R型指令中都全为0；
- rs：寄存器编号，对应第1个源操作数；
- rt：寄存器编号，对应第2个源操作数；
- rd：寄存器编号，据此保存结果；
- shamt：常数，在移位指令中使用；
- funct：功能码，指定指令的具体功能；

MIPS指令详解

- R型指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
add	000000	rs	rt	rd	00000	100000	寄存器加
sub	000000	rs	rt	rd	00000	100010	寄存器减
and	000000	rs	rt	rd	00000	100100	寄存器与
or	000000	rs	rt	rd	00000	100101	寄存器或
xor	000000	rs	rt	rd	00000	100110	寄存器异或
sll	000000	00000	rt	rd	sa	000000	逻辑左移
srl	000000	00000	rt	rd	sa	000010	逻辑右移
sra	000000	00000	rt	rd	sa	000011	算术右移
jr	000000	rs	00000	00000	00000	001000	寄存器跳转

MIPS指令详解

- R型指令存在3种不同类型
- 3寄存器R型指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
add	000000	rs	rt	rd	00000	100000	寄存器加
sub	000000	rs	rt	rd	00000	100010	寄存器减
and	000000	rs	rt	rd	00000	100100	寄存器与
or	000000	rs	rt	rd	00000	100101	寄存器或
xor	000000	rs	rt	rd	00000	100110	寄存器异或

指令功能：$rd ← $rs op $rt

MIPS指令详解

- 2寄存器R型指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
sll	000000	00000	rt	rd	sa	000000	逻辑左移
srl	000000	00000	rt	rd	sa	000010	逻辑右移
sra	000000	00000	rt	rd	sa	000011	算术右移

指令功能：$rd ← $rt shift sa

MIPS指令详解

- 1寄存器R型指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
jr	000000	rs	00000	00000	00000	001000	寄存器跳转

jr rs;

PC ← rs

MIPS指令详解

- I型指令

操作数中涉及立即数，结果保存到寄存器；

- op：标识指令的操作功能；
- rs：第1个源操作数，是寄存器操作数；
- rt：目的寄存器编号，用来保存运算结果；
- imm：第2个源操作数，立即数；

MIPS指令详解

- I型指令

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
addi	001000	rs	rt	imm	寄存器和立即数“加”
andi	001100	rs	rt	imm	寄存器和立即数“与”
ori	001101	rs	rt	imm	寄存器和立即数“或”
xori	001110	rs	rt	imm	寄存器和立即数“异或”
lw	100011	rs	rt	imm	从存储器中读取数据
sw	101011	rs	rt	imm	把数据保存到存储器
beq	000100	rs	rt	imm	寄存器相等则转移
bne	000101	rs	rt	imm	寄存器不等则转移
lui	001111	00000	rt	imm	设置寄存器的高16位

MIPS指令详解

I型指令存在4种不同类型

- **面向运算的I型指令**

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
addi	001000	rs	rt	imm	寄存器和立即数“加”
andi	001100	rs	rt	imm	寄存器和立即数“与”
ori	001101	rs	rt	imm	寄存器和立即数“或”
xori	001110	rs	rt	imm	寄存器和立即数“异或”

addi/andi/ori/xori rt, rs, imm; # $rt ← $rs op E(imm)

第一条指令是进行符号扩展，其余是0扩展

MIPS指令详解

- 面向访存的I型指令

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
lw	100011	rs	rt	imm	从存储器中读取数据
sw	101011	rs	rt	imm	把数据保存到存储器

MIPS32中唯一两条访问存储器的指令(RISC)

```
lw  rt, imm(rs)   #  $rt ← mem[$rs+E(imm)]
sw  rt, imm(rs)   #  mem[$rs+E(imm)] ←$rt
```

MIPS指令详解

- **面向数位设置的I型指令**

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
lui	001111	00000	rt	imm	设置寄存器的高16位

lui rt, imm # $rt ← imm<<16 (空位补0)

MIPS指令详解

- 面向条件转移(分支)的I型指令

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
beq	000100	rs	rt	imm	寄存器相等则转移
bne	000101	rs	rt	imm	寄存器不等则转移

beq rs, rt, imm　　#if($rs==$rt) PC←PC+E(imm)<<2

bne rs, rt, imm　　#if($rs!=$rt) PC←PC+E(imm)<<2

是标准的PC相对寻址方式

其中imm要先“带符号扩展”成32位，再左移2位。

MIPS指令详解

- J型指令

指令	[31:26]	[25:0]	指令功能
j	000010	address	无条件跳转
jal	001100	address	调用与联接

j address; $PC← ($PC+4)$_{H4}$ ∪ (address<<2)

jal address; $ra←$PC+4 (保存返回地址)

$PC ← ($PC+4)$_{H4}$ ∪ (address<<2)

MIPS指令练习

1 下列关于MIPS R型指令的描述中，正确的是（ ）

- A. 不同功能的R型指令使用的寄存器数量不一定相同
- B. 所有R型指令的操作码OP字段的值均为 000000
- C. R型指令既有算术运算指令，也有逻辑运算指令
- D. R型指令不支持访问主存的指令

MIPS指令练习

2 下列关于MIPS Ⅰ型指令的描述中，正确的是（ ）

- A. Ⅰ型指令包括访问内存的指令
- B. Ⅰ型指令包括条件转移指令
- C. Ⅰ型指令包括立即数运算指令
- D. Ⅰ型指令支持给寄存器赋立即数的操作

MIPS指令练习

3 下列关于 MIPS J型指令的描述中，正确的是（ ）

- [x] A. J型指令支持无条件跳转指令
- [x] B. J型指令只使用伪直接寻址方式
- [x] C. J 型指令执行后，PC寄存器的值最后两位一定为00
- [ ] D. 所有J型指令均不使用MIPS的任何通用寄存器