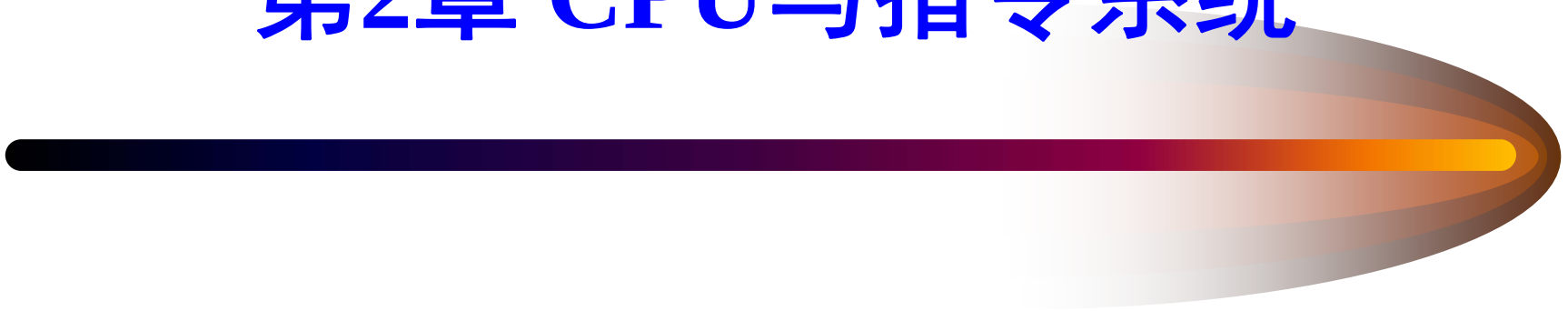
第2章 CPU与指令系统



2.1 基本概念

2.2 指令周期

2.3 指令的执行过程

2.4 指令的流水执行

2.5 指令和数据的寻址方式

2.6 指令系统分类

2.1 CPU基本概念

CPU的基本功能

- 指令控制
- 操作控制
- 数据运算
- 异常和中断处理

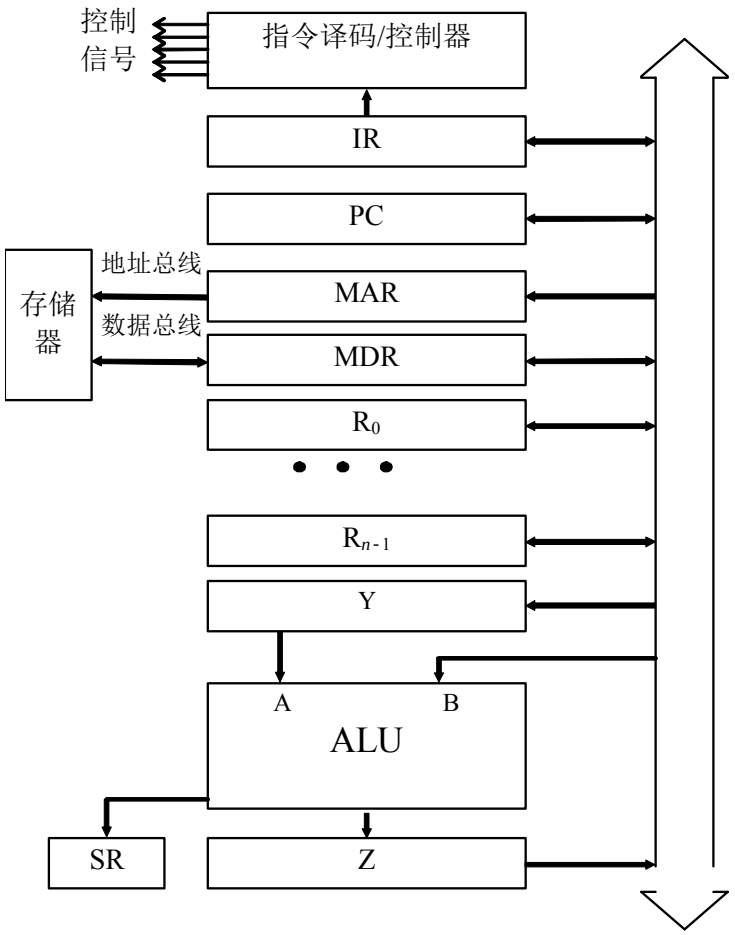
CPU的扩展功能

- 存储管理
- 数据缓存
- 总线管理
- 功耗管理

2.1 CPU基本概念

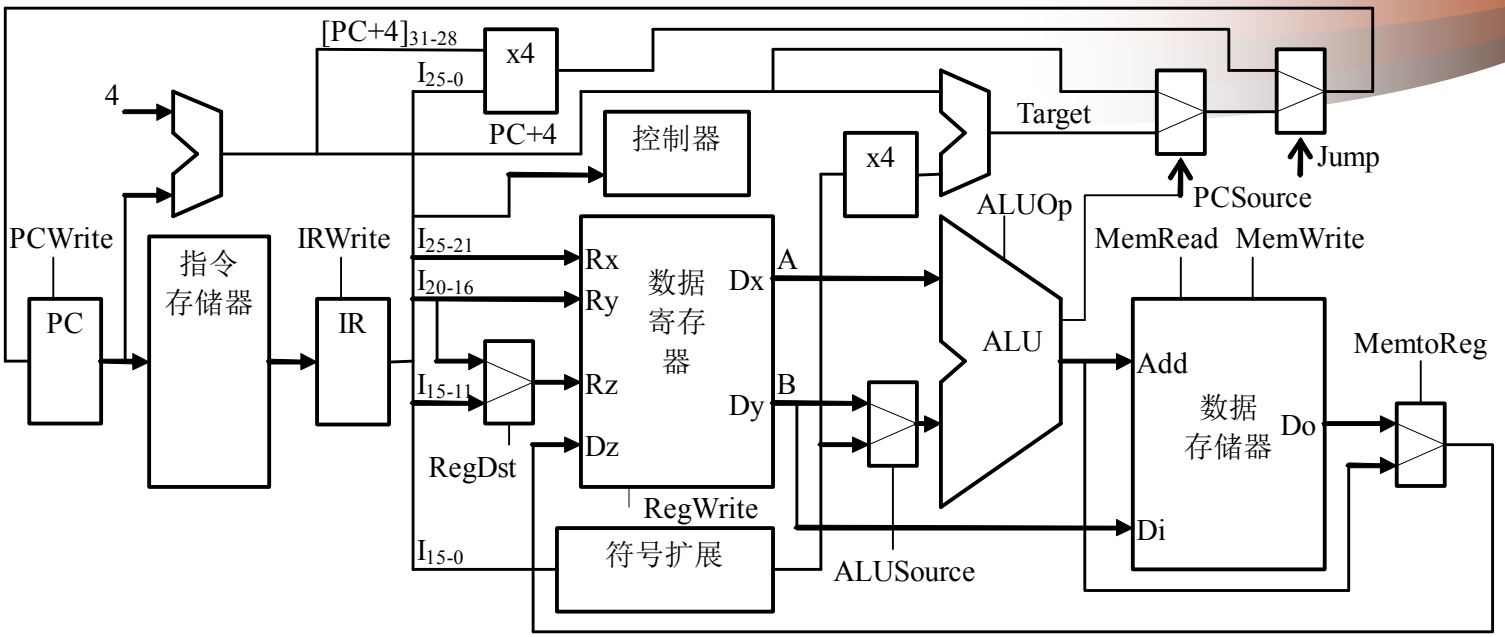
- **CPU的基本构成：**
  - 控制器，运算器，寄存器，[cache、MMU]
- 寄存器：用于（临时）存放各种信息。
  - 指令寄存器（IR）：存放当前执行的指令，为指令译码器提供指令信息。
  - 程序计数器（PC）：存放指令的地址，从存储器取指令时根据PC值进行。
  - 数据寄存器（DR）：存放操作数和运算结果，以减少访问存储器的次数。
  - 地址寄存器（AR）：存放操作数的地址。
  - 状态寄存器（SR）：存储运算中的状态，作为控制程序的条件。
- 数据通路：寄存器与ALU之间传递信息的线路。通常有2种建立方法：
  - 用数据总线（单总线，双总线，多总线）
  - 用专用通路（如MIPS）

数据总线结构



- 在各寄存器以及**ALU**之间建立一条或几条公共的数据总线，寄存器间的数据传输通过这些总线完成。
- 一条总线可以连接多个部件，总线连接方式可以减少线路的数量。
- 总线上可以有多个部件同时接收数据，但任一时刻只能有一个部件向同一条总线发送数据。
- 常用的是单总线结构，即数据通路只用一条总线构成，一次传输一个数据。

专用通路结构



- 在各寄存器与**ALU**之间根据指令执行过程中的操作和数据流向来安排功能部件并建立相应的数据传输通路。
- 每条数据传输通路都是专用的，不共享使用。
- 在数据传输和操作中可以做到互不相关，控制比较简单，各寄存器之间的数据传输可以并行进行，从而达到较高的性能。

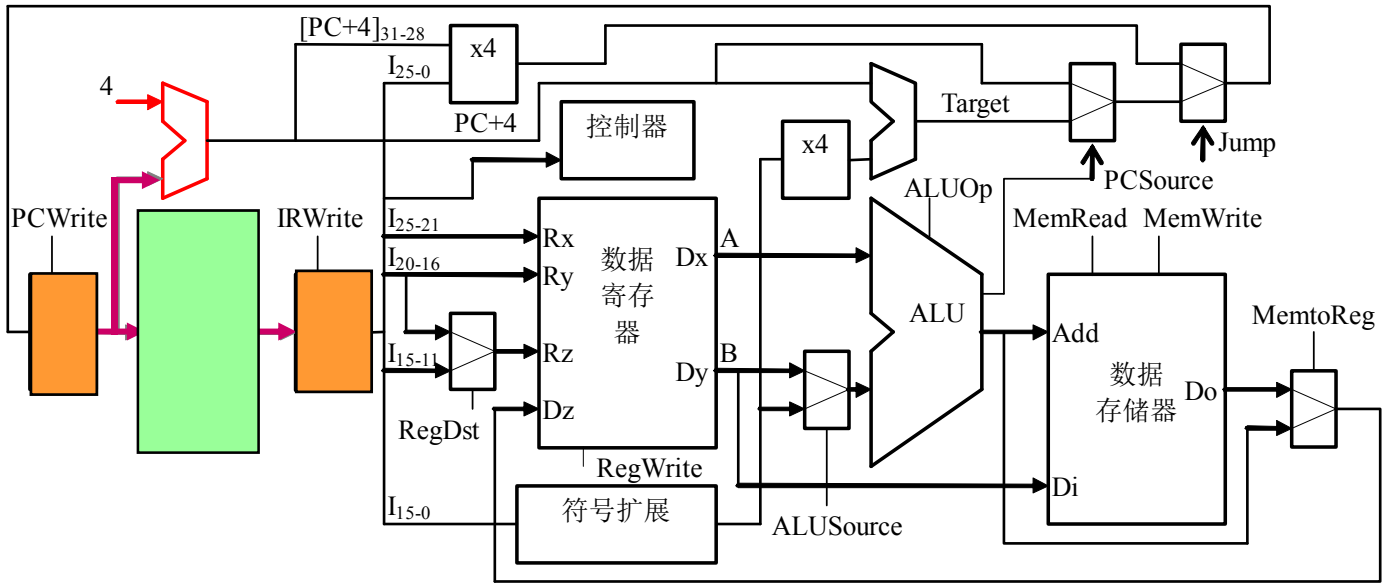
时间概念

- 指令周期
  - 是从一条指令的启动到下一条指令的启动的时间间隔。
- 机器周期
  - 指令周期中包含若干个基本操作步骤，如访问存储器和运算等。每个基本操作的时间称为机器周期。
- 时钟周期
  - 是计算机时钟主频的周期。
  - 一个机器周期可以包含若干个时钟周期

2.2 指令的执行过程

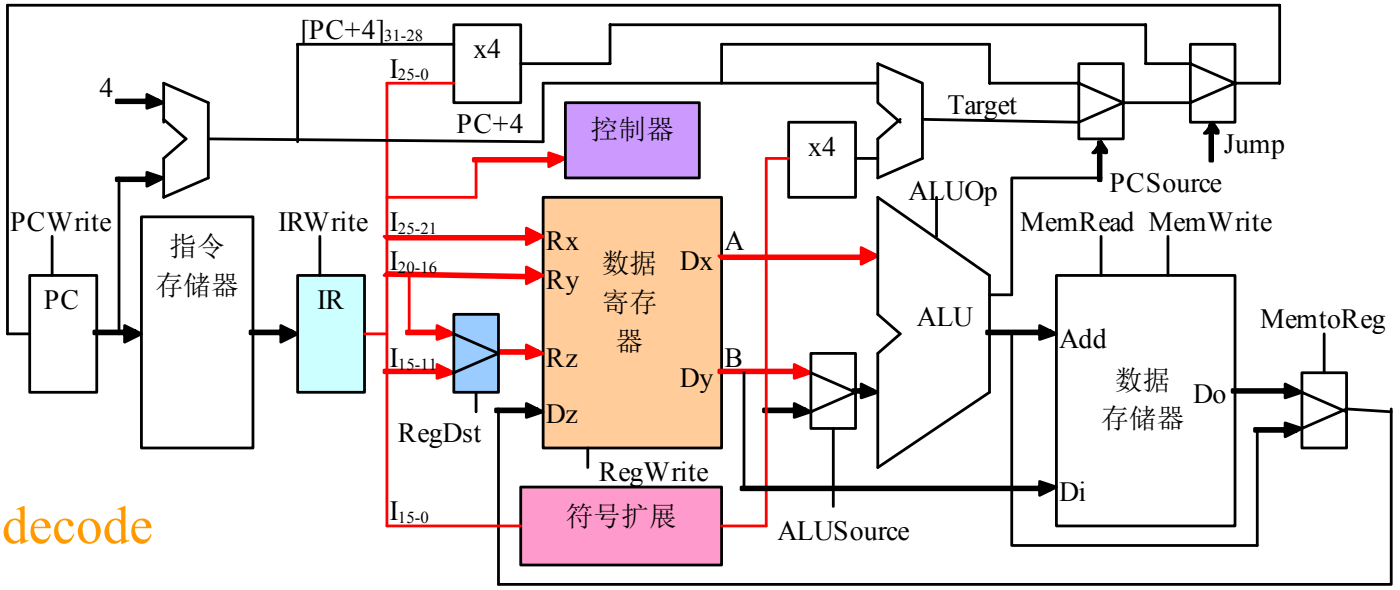
例：运算指令的执行过程

- 采用专用通路结构：如**add $1, $2, $3**
- (1) 取指令。`IR = Memory[PC]`　　instruction fetch
  - `PC = PC + 4`



一、运算指令的执行过程

- 专用通路结构：如add $1, $2, $3
  - (2) 指令译码及读取操作数。A = R[IR[25:21]]
    - B = R[IR[20:16]]

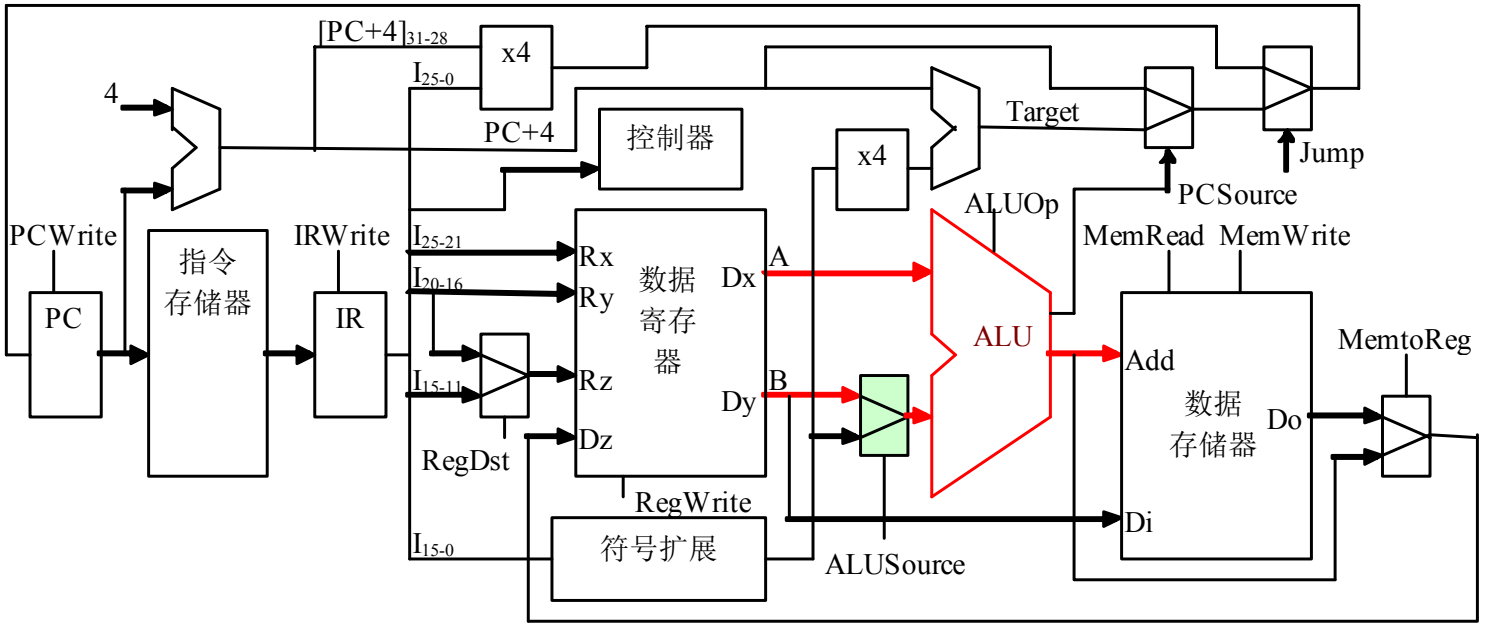


Instruction decode

Operand fetch

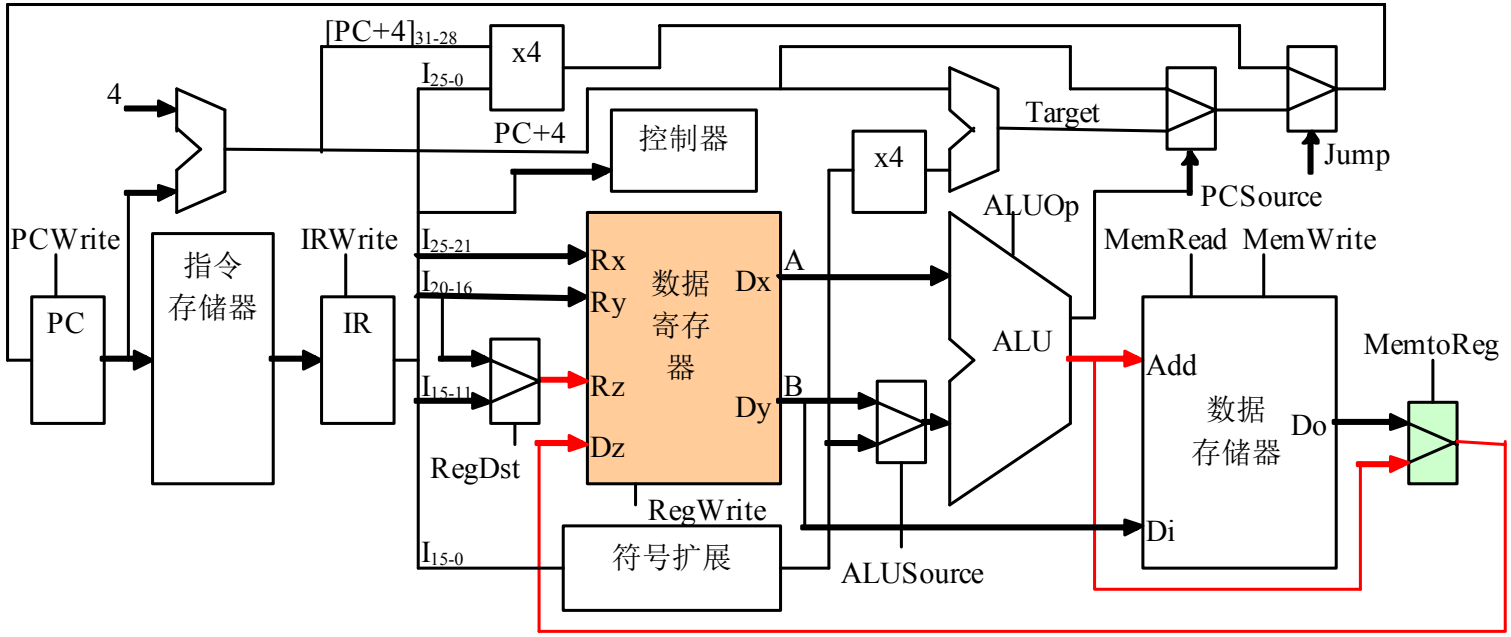
一、运算指令的执行过程

- 专用通路结构：如add $1, $2, $3
- (3) 执行。ALUoutput = A + B　execution



一、运算指令的执行过程

- 专用通路结构：如add $1, $2, $3
- (4) 写回。`R[IR[15:11]] = ALUoutput` write back



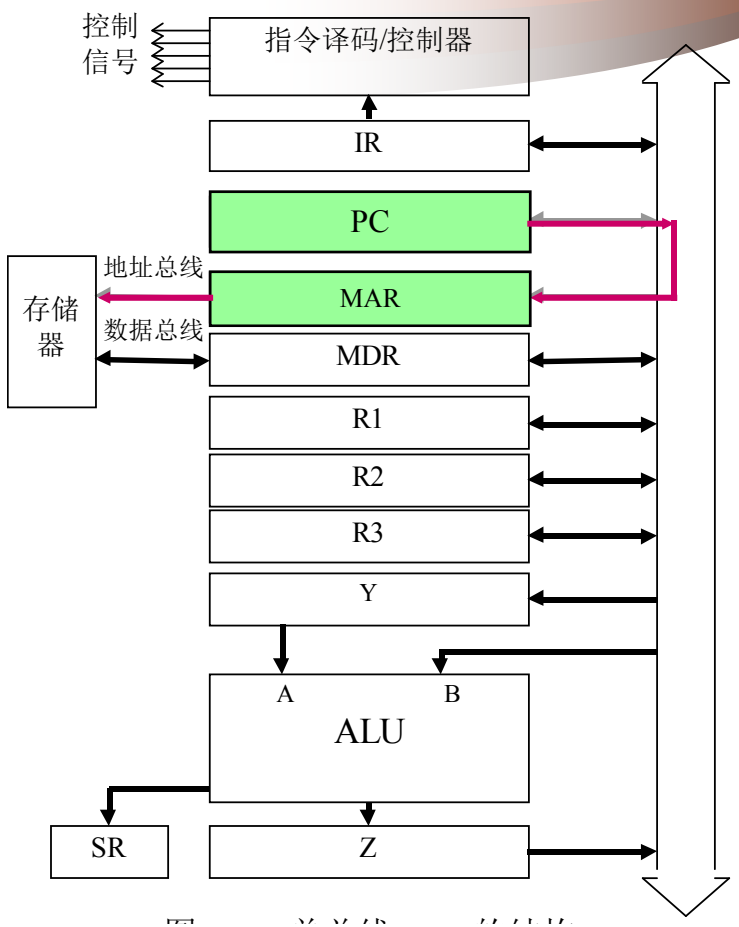
一、运算指令的执行过程

采用单总线结构：

– 如**ADD R3, R1, R2**

(1) PC→MAR

(2) PC+1→PC



一、运算指令的执行过程

单总线结构：

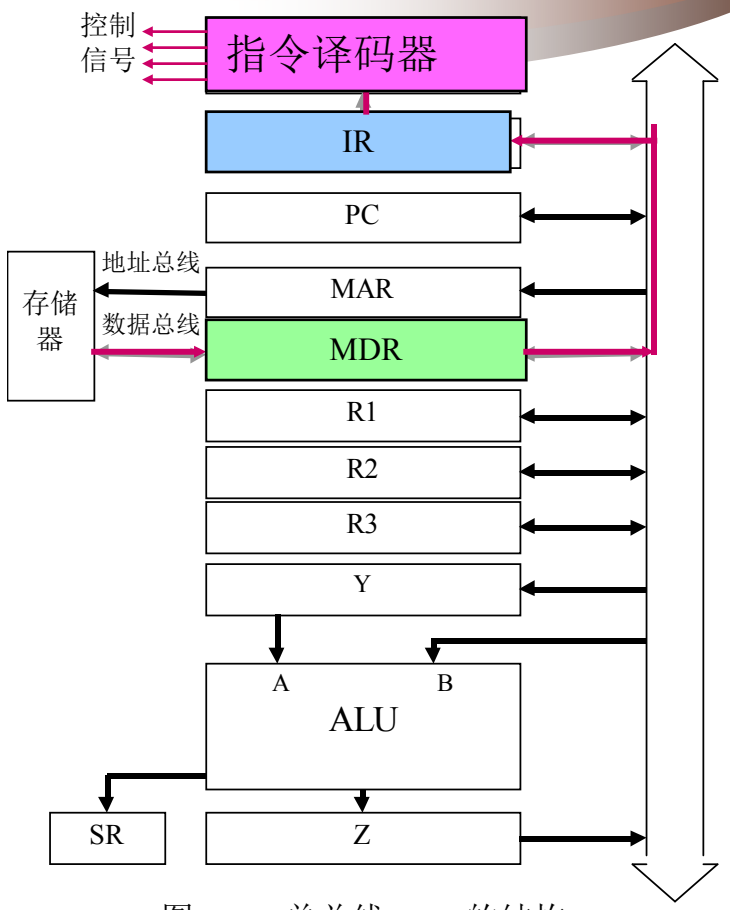
– 如ADD R3, R1, R2

(1) PC→MAR

(2) PC+1→PC

(3) DBUS→MDR

(4) MDR→IR



一、运算指令的执行过程

单总线结构：

– 如ADD R3, R1, R2

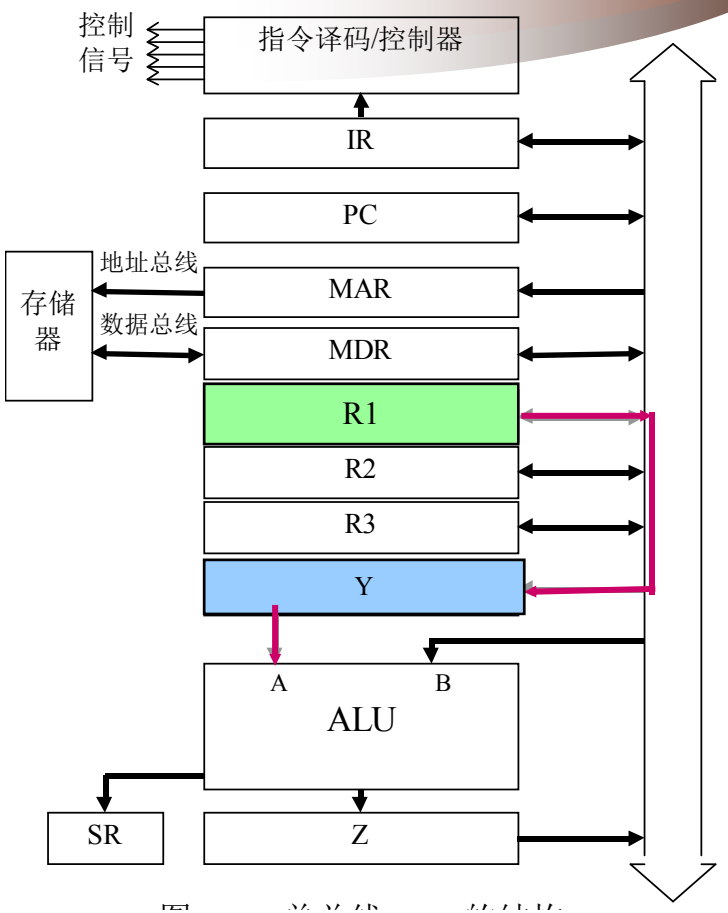
(1) PC→MAR

(2) PC+1→PC

(3) DBUS→MDR

(4) MDR→IR

(5) R1→Y



一、运算指令的执行过程

单总线结构：

– 如ADD R3, R1, R2

(1) PC→MAR

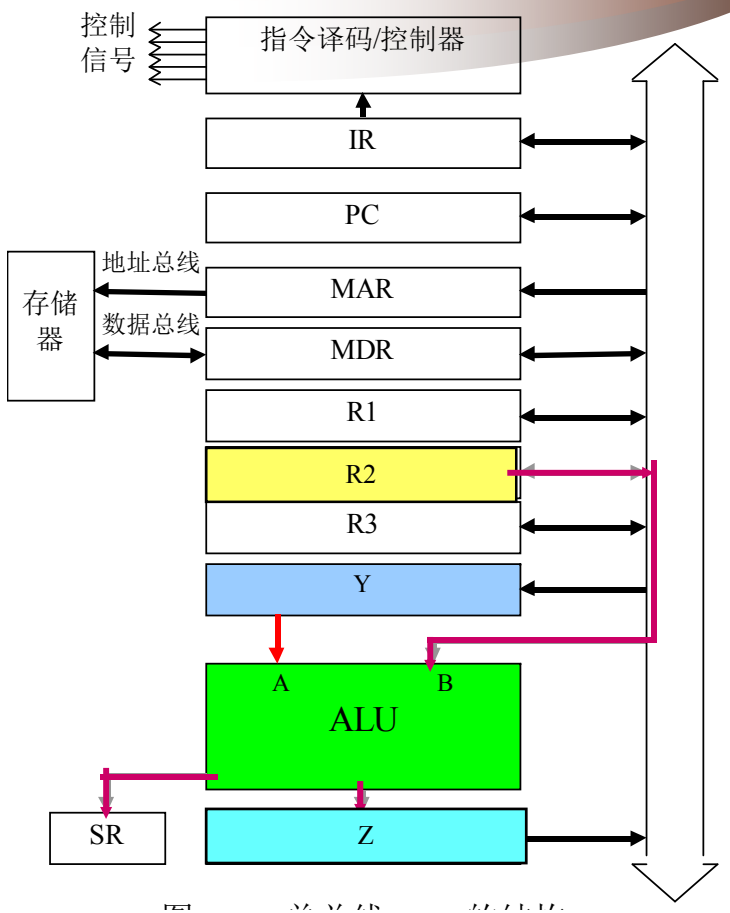
(2) PC+1→PC

(3) DBUS→MDR

(4) MDR→IR

(5) R1→Y

(6) R2 + Y→Z



一、运算指令的执行过程

单总线结构：

– 如ADD R3, R1, R2

(1) PC→MAR

(2) PC+1→PC

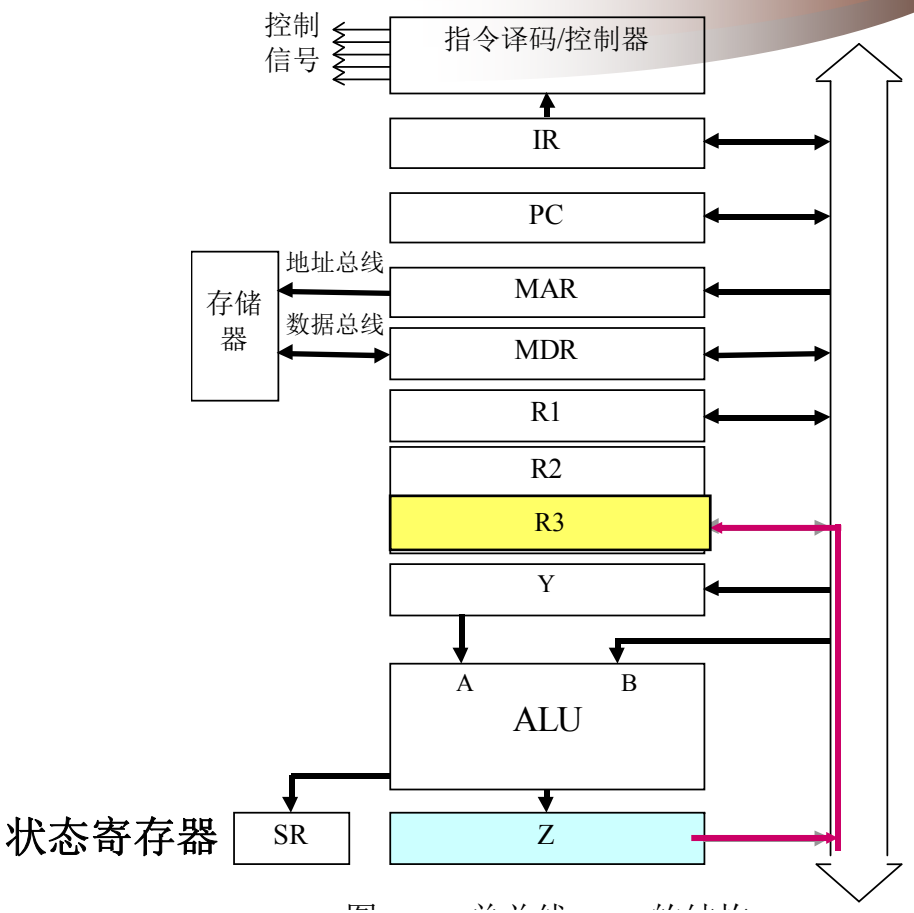
(3) DBUS→MDR

(4) MDR→IR

(5) R1→Y

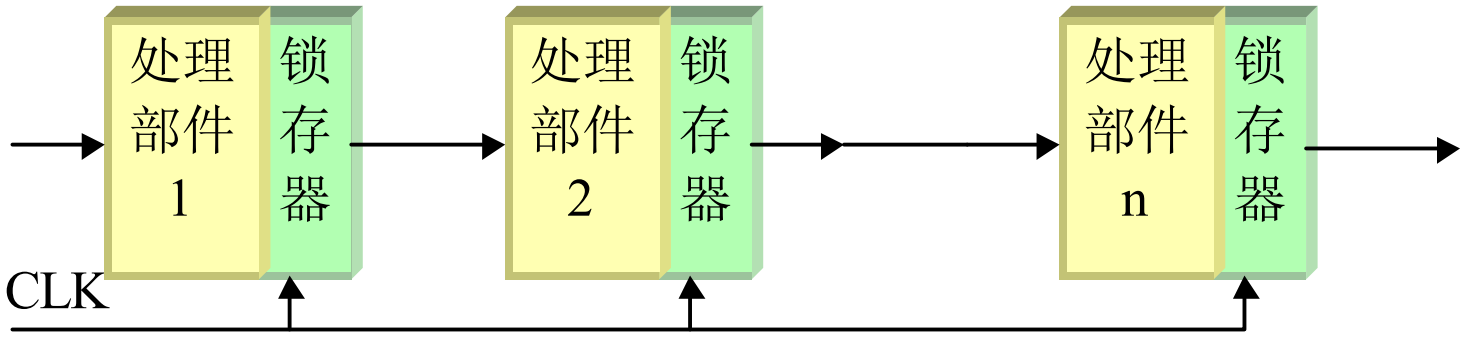
(6) R2 + Y→Z

(7) Z→R3



2.4 指令的流水执行

- **流水技术**用于**提高指令的执行速度和数据运算速度。**
- **流水工作方式**
  - 将一个计算任务细分成若干个子任务
  - 每个子任务由专门的部件处理
  - 多个计算任务依次进行并行处理



流水周期

2.4 指令的流水执行

- 计算机流水线中的计算任务可以是可以是一个算术逻辑运算操作，也可以是一条指令的执行。
- 指令级流水线是把一条指令的执行过程分成多个子过程，由各个部件进行轮流处理后完成执行过程。
- 不必等到上一条指令的完成就可以开始下一条指令的执行。
- 指令的流水线在高性能的微处理器中被普遍采用。

指令的流水线

- 指令流水线由一系列串联的流水段组成。
  - 每个流水段完成指令执行的一个操作步骤。
  - 各个流水段之间设有缓冲寄存器（流水寄存器），以暂时保存上一个流水段对指令处理的结果。
  - 流水线中的每个流水段构成流水线的一级。
  - 在专用通路结构的CPU中，通常采用5个流水段（5级流水）
    - ① 取指（IF）--Instruction Fetch
    - ② 译码（ID）--Instruction Decode
    - ③ 运算执行（EX）--Execute
    - ④ 访存（MEM）--Memory
    - ⑤ 写回（WB）--Write Back

流水线的性能

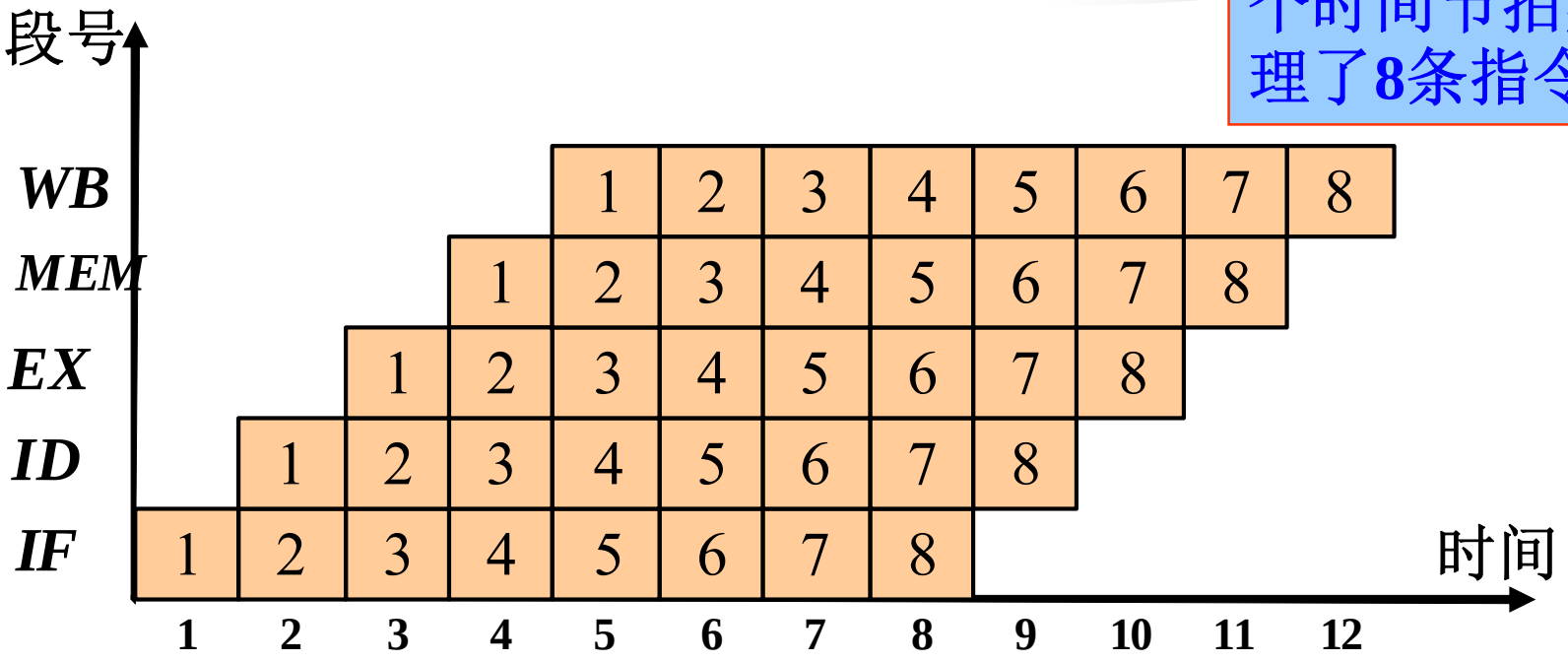
1. 吞吐率
   - 衡量指令流水线的一个重要指标。
   - 单位时间内流水线能处理的任务数量
   - 与流水的节拍时间（流水周期）有关。
   - 可将流水周期定为各流水段处理时间的最大值，加上流水寄存器的延迟时间。
2. 加速比
   - 流水方式的工作速度与等效的顺序工作方式时间的比值

时空图-根据流水段来画

无流水时：5个时间节拍处理1条指令。

有流水时：12个时间节拍处理了8条指令

- 5级流水线在执行8条指令时：



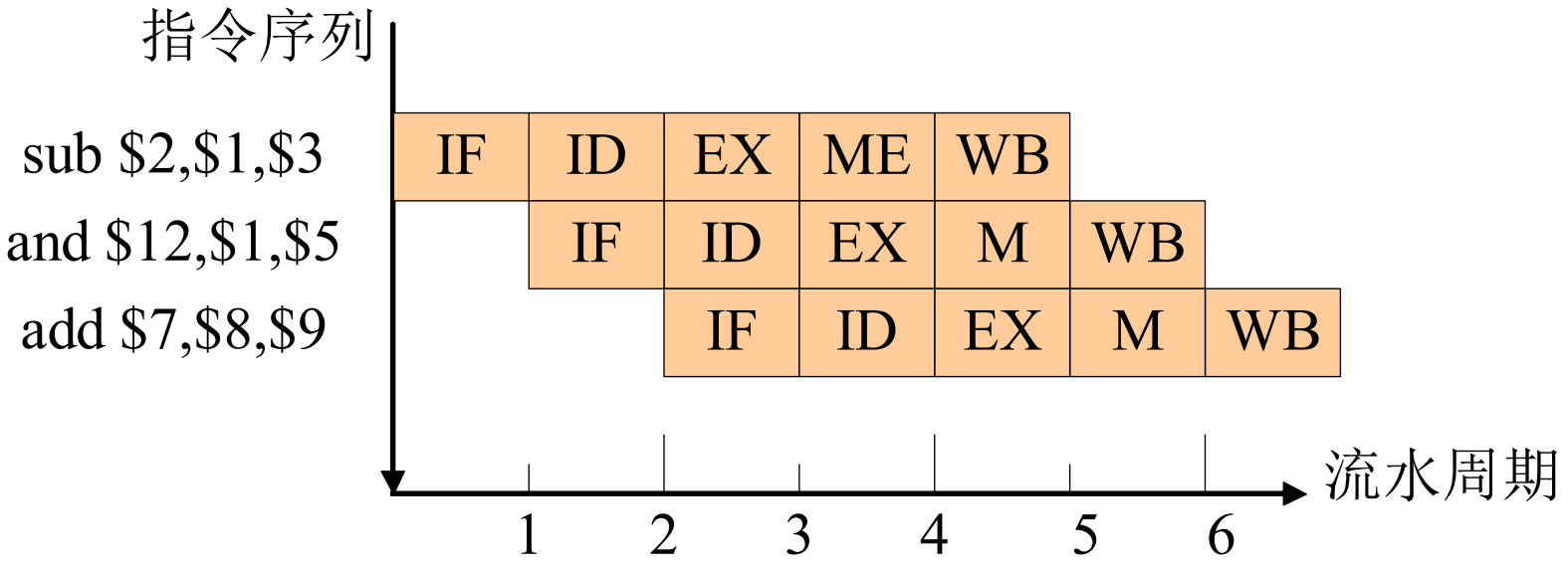
- 建立时间：从第一条指令进入流水线到离开流水线的时间。
- 排空时间：从最后一条指令进入流水线到离开流水线的时间。

流水线的作用

- 由时空图可见：
  - 对进入流水线的每条指令，其执行时间都没有缩短。
  - 指令流水通过指令之间的重叠来提高吞吐率。
  - 在流水线中，当任务饱满时，任务源源不断地输入流水线，不论有多少个流水段，每隔一个流水周期都能输出一个任务。
  - 从而在宏观上提高了处理速率。

时空图-根据指令序列来画（常用）

- 便于分析指令之间的关系



指令流水线的相关性

1. 资源相关（结构相关）
   - 流水执行的多条指令同时使用同一个部件。
   - 由于硬件资源不够造成的，与硬件结构有关。
2. 数据相关
   - 流水执行的多条指令访问相同的数据，使得相关的指令不能并行地执行。
   - **RAW**（读后写）**WAR**（读后写）**WAW**（写后写）
3. 控制相关
   - 转移指令引起的相关
   - 无法确定那一条指令是后继指令，使得后继指令不能进入流水线。

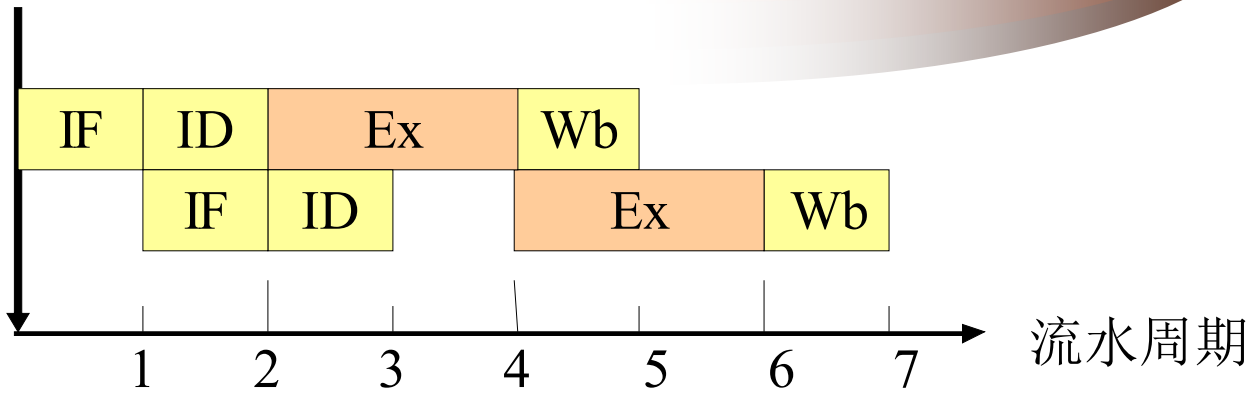
指令流水线的资源相关

Addd: 双精度加法指令

F2: 浮点数寄存器

addd \$1,\$2,\$3

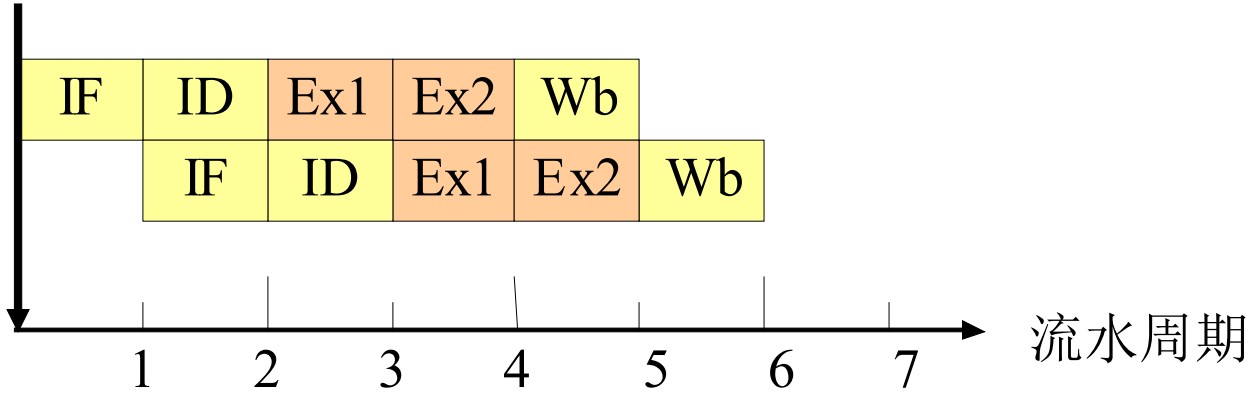
addd \$4,\$5,\$6



(a) 当执行部件不可流水，资源相关

addd \$1,\$2,\$3

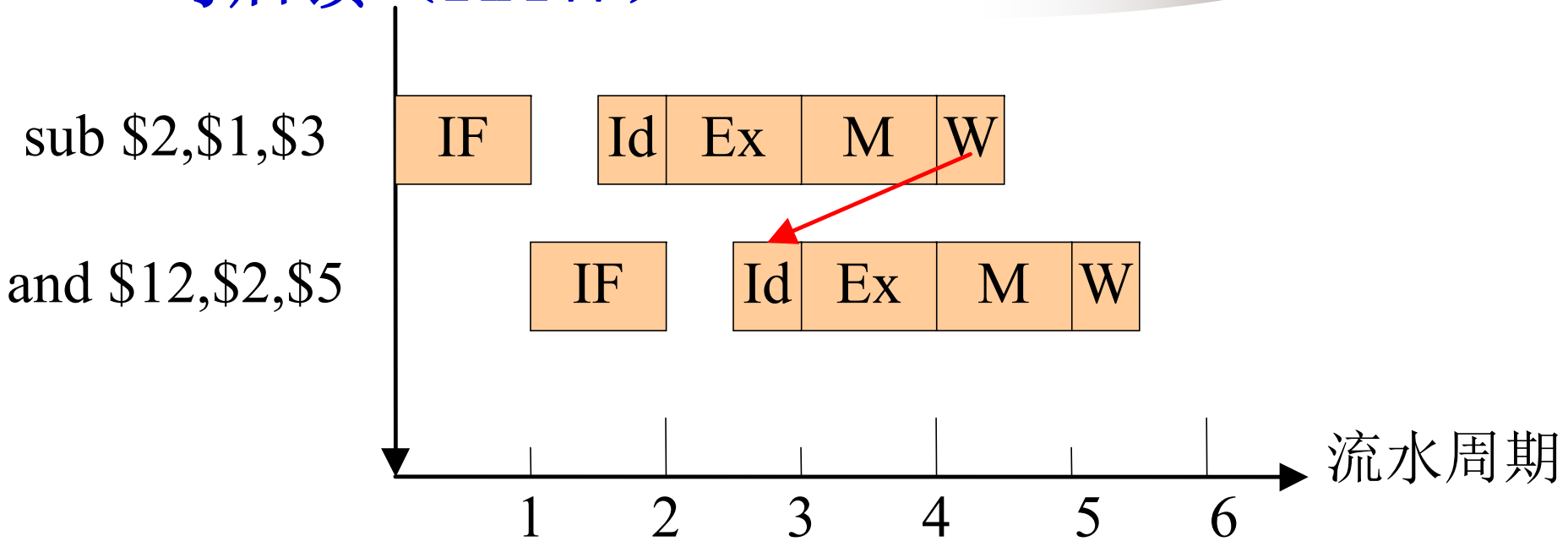
addd \$4,\$5,\$6



(b) 当执行部件可流水，无资源相关

指令流水线的数据相关

- 写后读（**RAW**）

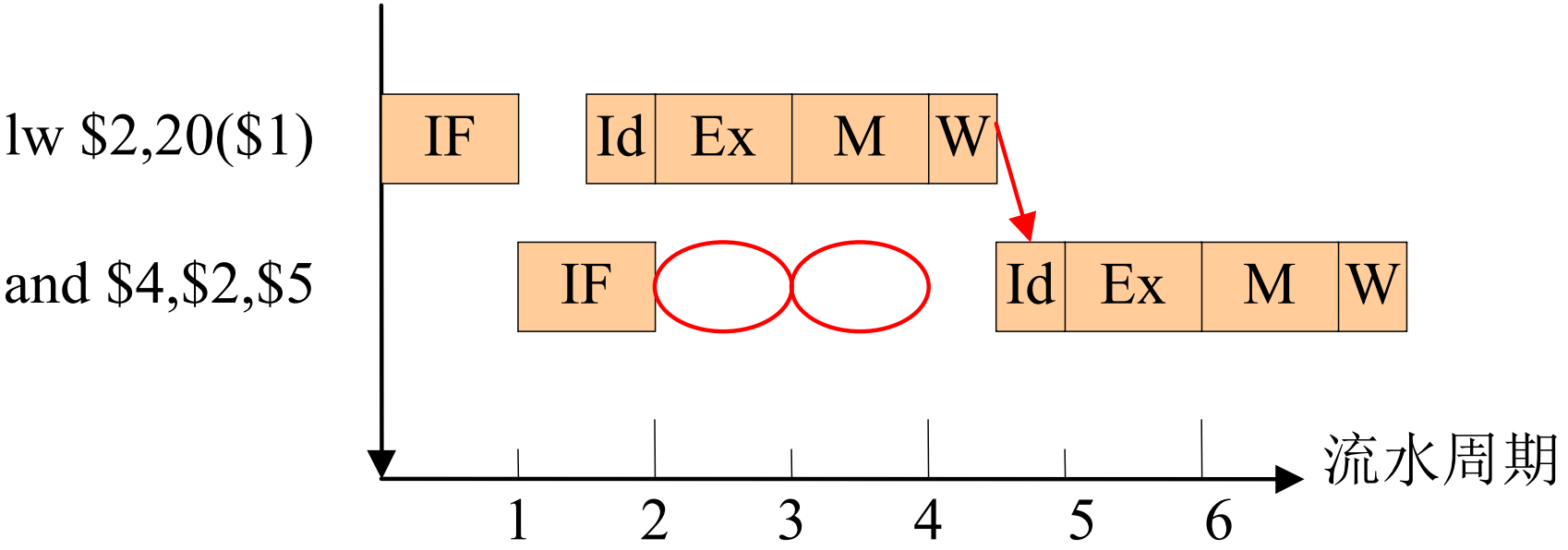


消除数据相关影响的方法

- 停顿（硬件措施）
- 编译检测（软件措施）
- 相关专用通路（硬件措施）

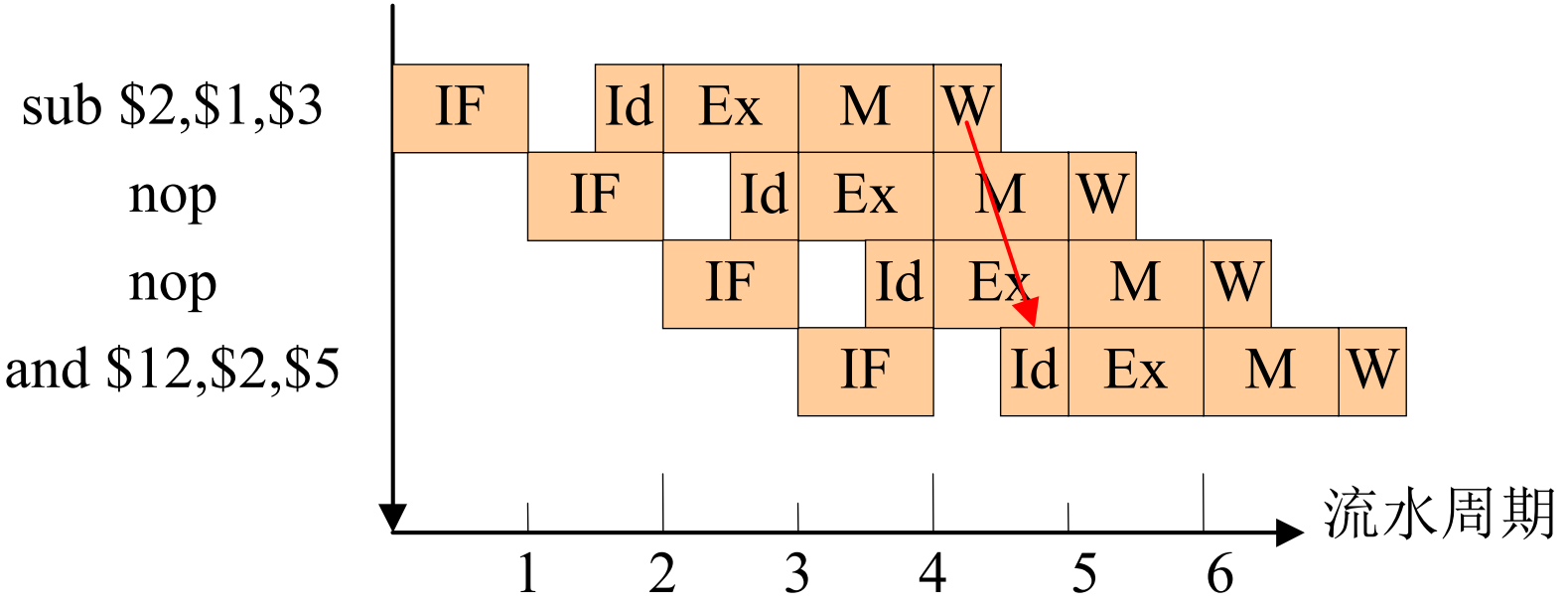
解决数据相关性的方法

- 流水线的停顿：停顿N个节拍



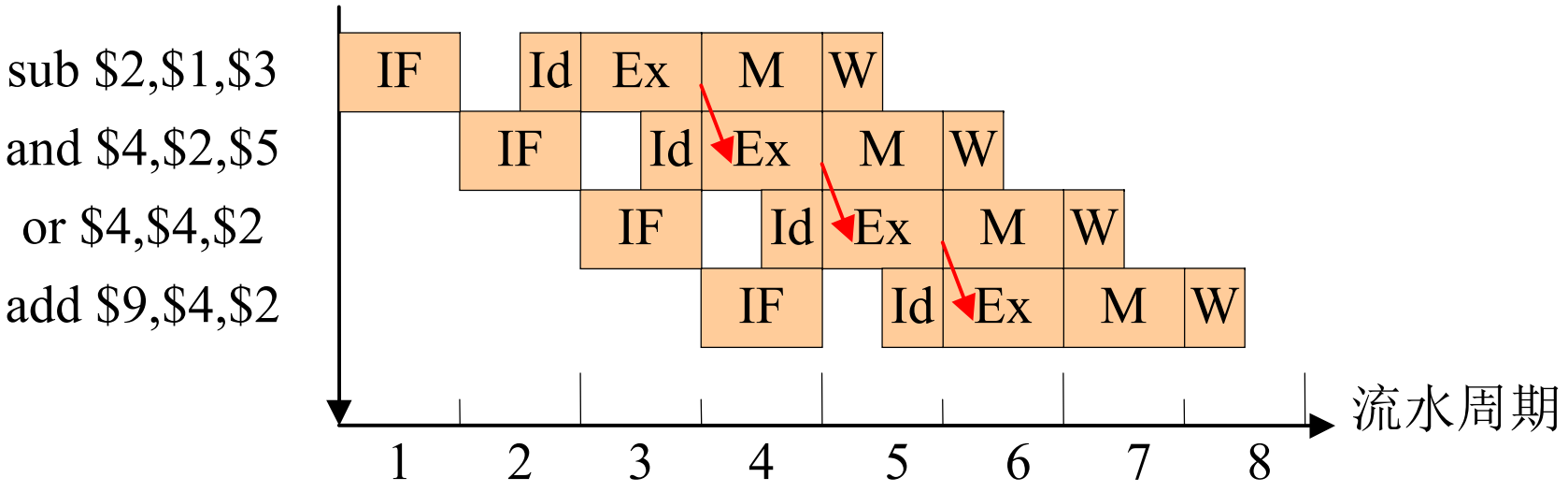
解决数据相关性的方法

- 编译检测：加入空操作（NOP）



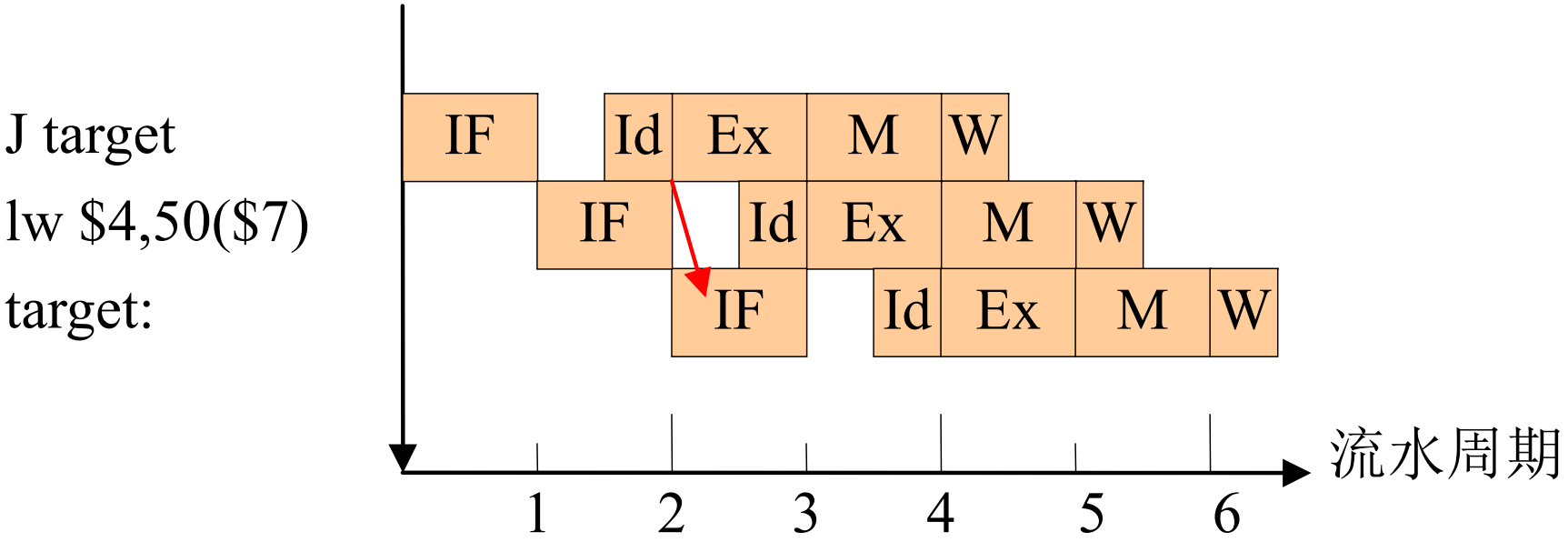
解决数据相关性的方法

- 相关专用通道
  - 在**ALU**的输出端到它的输入端之间设置一条数据线路，使得**RAW**相关时，上一条指令的结果能直接送到**ALU**的输入端。
  - 使得下一条指令能够及时得到上一条指令的结果，而不用到寄存器中去读取，从而避免了流水线的停顿。



指令流水线的控制相关

- 改进转移指令执行性能的方式
  - 将转移指令的执行在流水线中提前进行



典型指令流水线的特点

- 流水线分为IF、ID、EX、MEM、WB五个等长的时间阶段
- 转移指令在第二个阶段被识别
  - 从而可确定如何取下一条指令
- 没有相关专用通路
- 一个时钟周期内只能启动执行一条指令
- 一个时钟周期内只能写回一条指令
- 读操作数从指令译码时开始
  - 读到为止

2.5 指令系统

- 指令格式和指令编码
- 指令和数据的寻址方式
- 指令系统

2.5.1 指令格式和指令编码

指令格式 instruction format

操作码[，地址码][,条件码][,下一条指令的地址]

一、操作码 opcode

固定长度操作码：便于译码，扩展性差

可变长度操作码：能缩短指令平均长度

二、地址码 addressing code

零地址指令，如**NOP, CLR**

一地址指令，如**INC R1**

二地址指令，如**ADD R1, R2**

三地址指令，如**ADD R1, R2, R3**

指令格式和指令编码

三、指令长度

固定长度：取指快、译码简单。

单字长、双字长、多字长

可变长度：可提高编码效率

四、指令助记符

- 助记符 **mnemonics**
  - 如**add**、**r1**
- 伪指令 **directive**
- 累加器 **accumulator**
  - 存放操作数和操作结果的一个寄存器
- 通用寄存器 **general purpose register**
  - 存放操作数和操作结果的一组寄存器

指令和数据的寻址方式

- 存储器中既存储指令，又存储数据。
- 在存储器中寻找指令或数据的方法有多种：
  - 按地址寻找
  - 按内容寻找
  - 按顺序寻找
- 在绝大多数计算机中都采用按地址寻找的方式，指令或数据的寻找问题变成了构成其地址的问题
- 在按地址寻找存储内容的计算机中，对指令的地址码进行编码，以形成操作数，寻找操作数在寄存器或存储器中地址的方式称为寻址方式。

指令和数据的寻址方式

1. 数据类型

- 数值型
  - 整型
    - 字节、字、双字
  - 浮点数
    - 单精度、双精度
- 字符型
  - **ASCII**

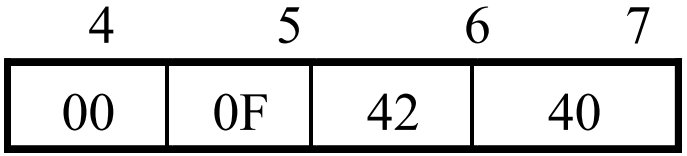
数据类型在指令中的表达

- 操作码表达
- 地址码表达
- 数据编码表达

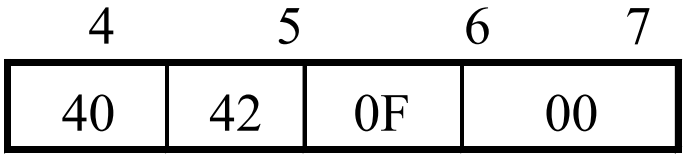
指令和数据的寻址方式

2. 数据存储字节顺序(Endianness)

- 大数端(Big Endian)
  - 最低字节存储在高地址
- 小数端(Little Endian)
  - 最低字节存储在低地址
- 例:数据000F4240:



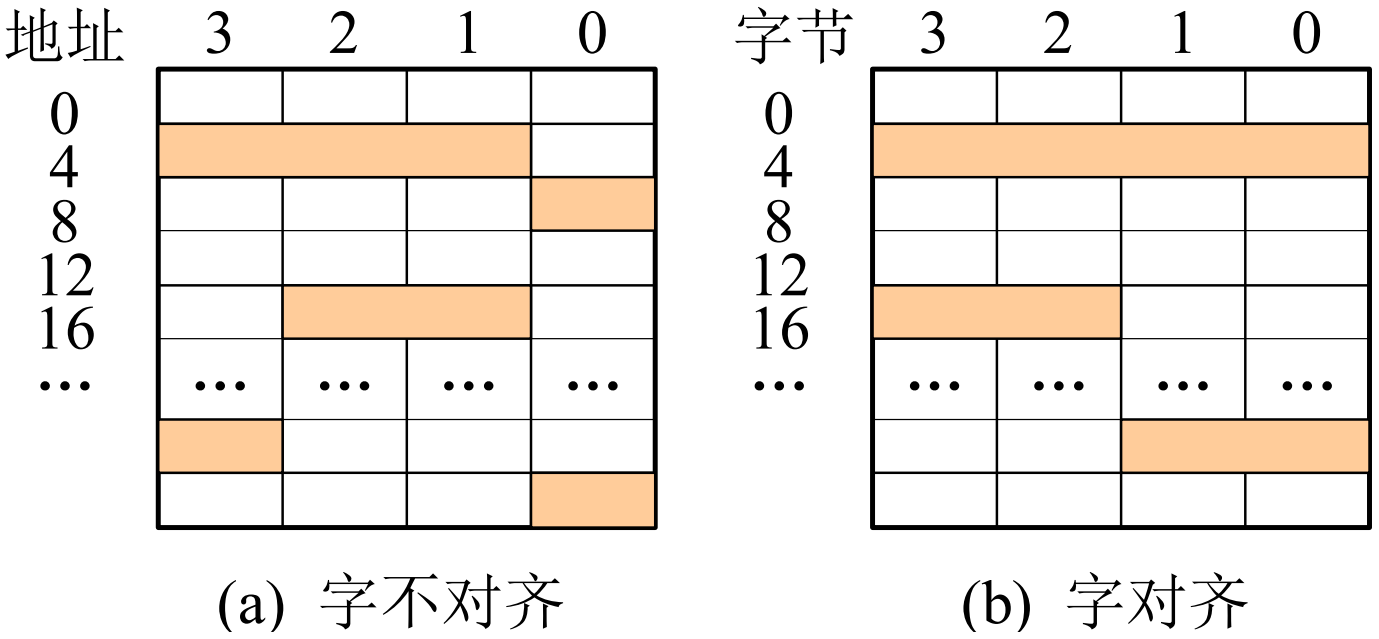
(a) 大数端存储方式



(b) 小数端存储方式

数据类型及其存储方式

- **3. 数据对齐方式(Alignment)**



(a) 字不对齐　　(b) 字对齐

指令和数据的寻址方式

指令的寻址方式

顺序执行

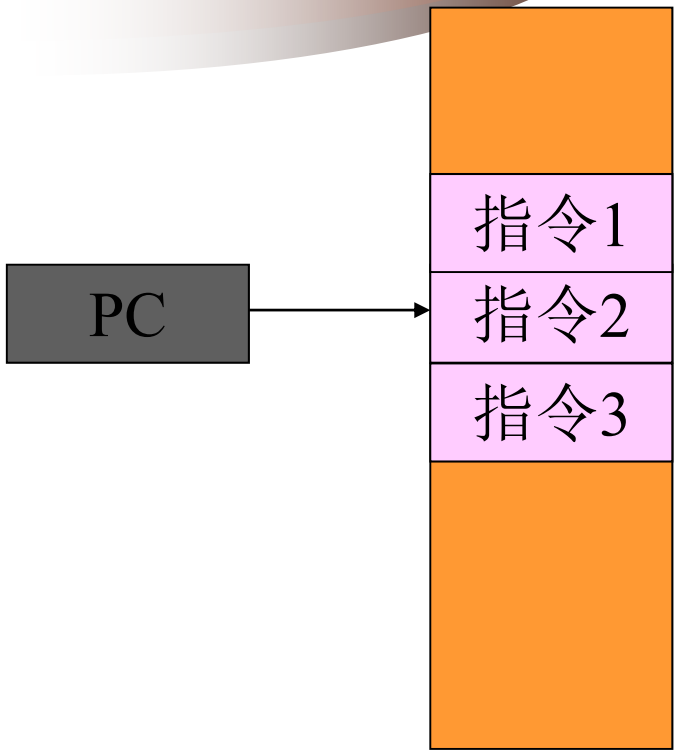
程序计数器(PC)寻址

非顺序执行

转移指令

如: **jump $1000**

bgt R1



操作数寻址方式

addressing modes

1. 隐含方式

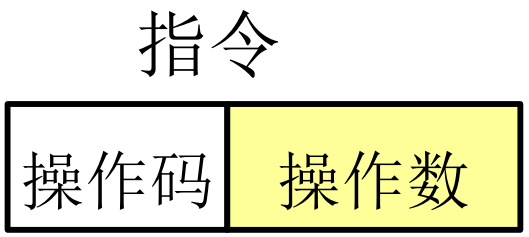
如“**ADD ADR**”中的累加器

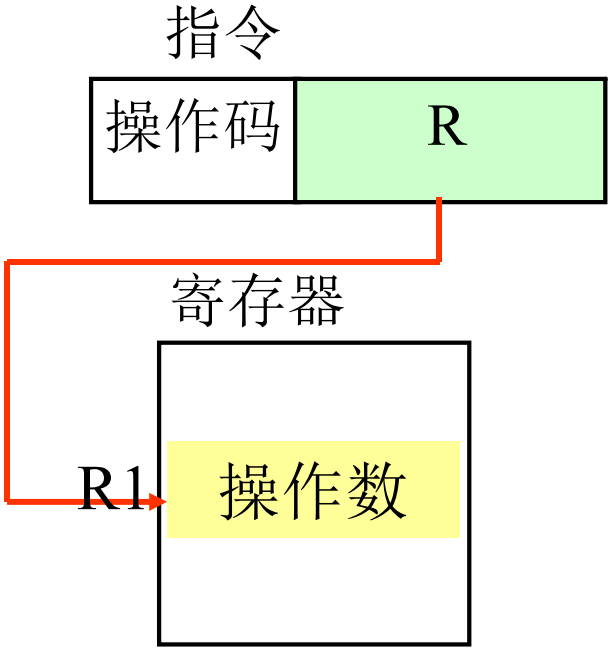
2. 立即数方式

如**INT #3**

3. 寄存器方式

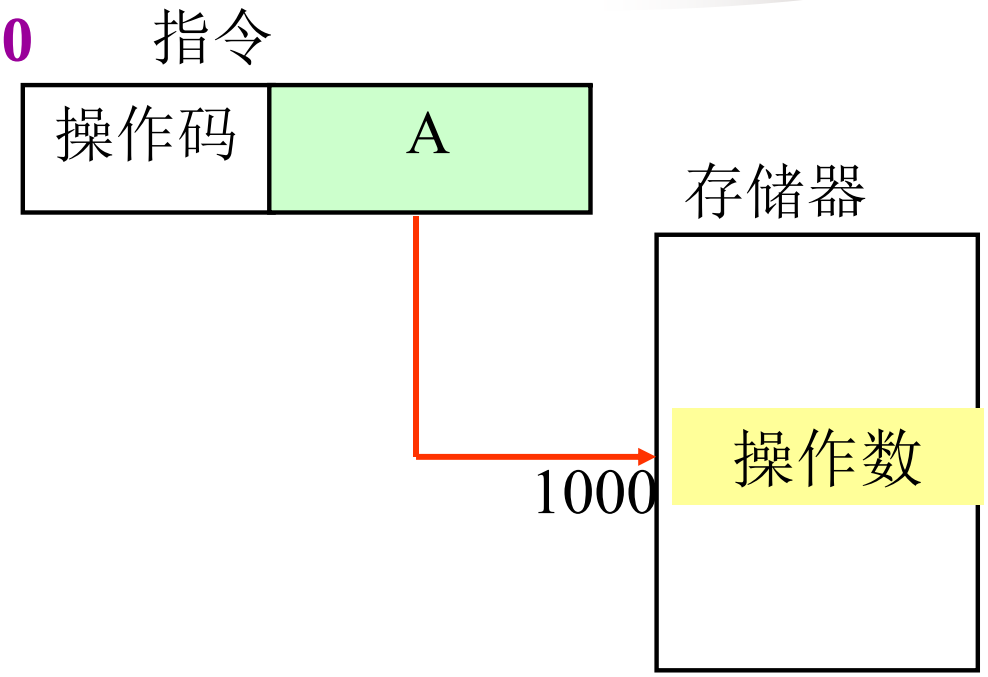
如**INC R1**





操作数寻址方式（续）

- **4.** 直接寻址
  - 如**INC 1000**

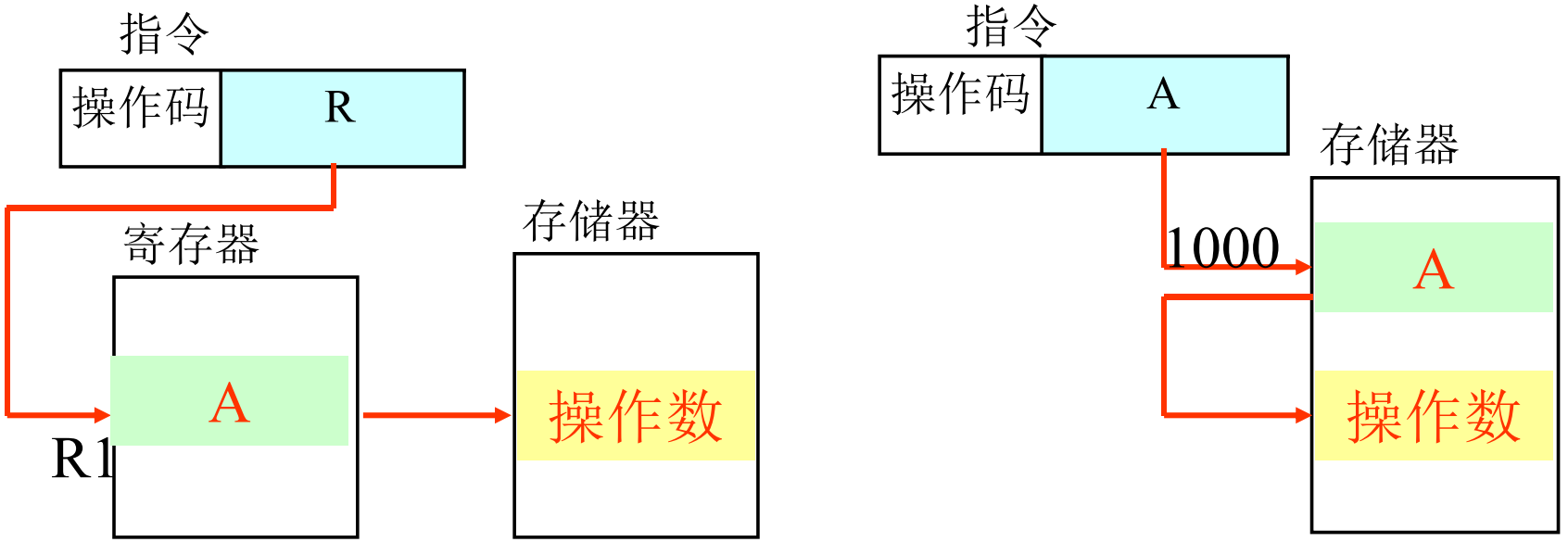


操作数寻址方式（续）

5. 间接寻址

寄存器间接，如INC (R1)

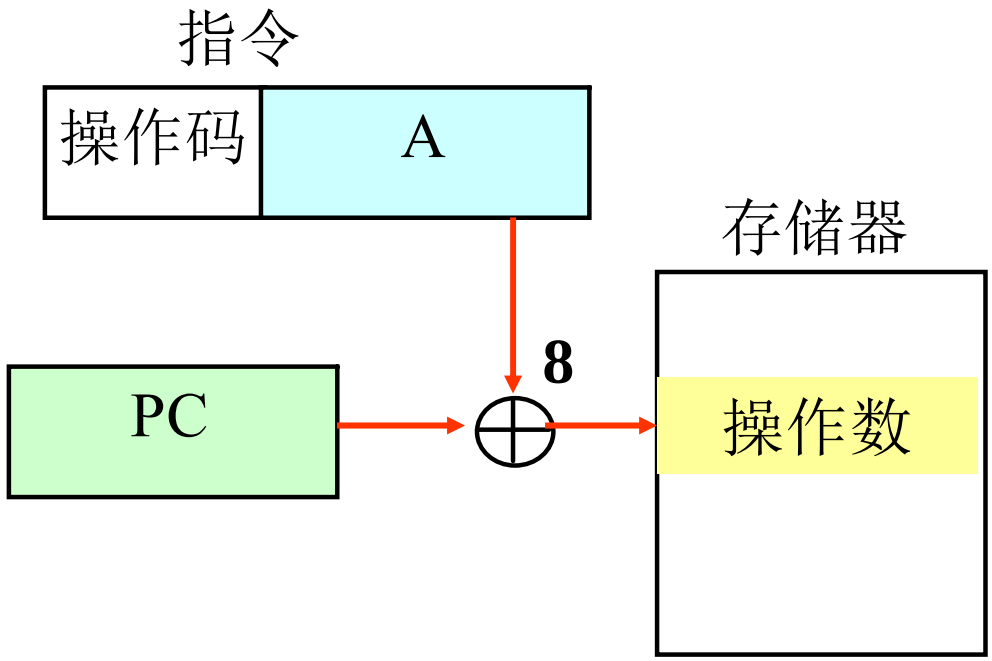
存储器间接，如INC (1000)



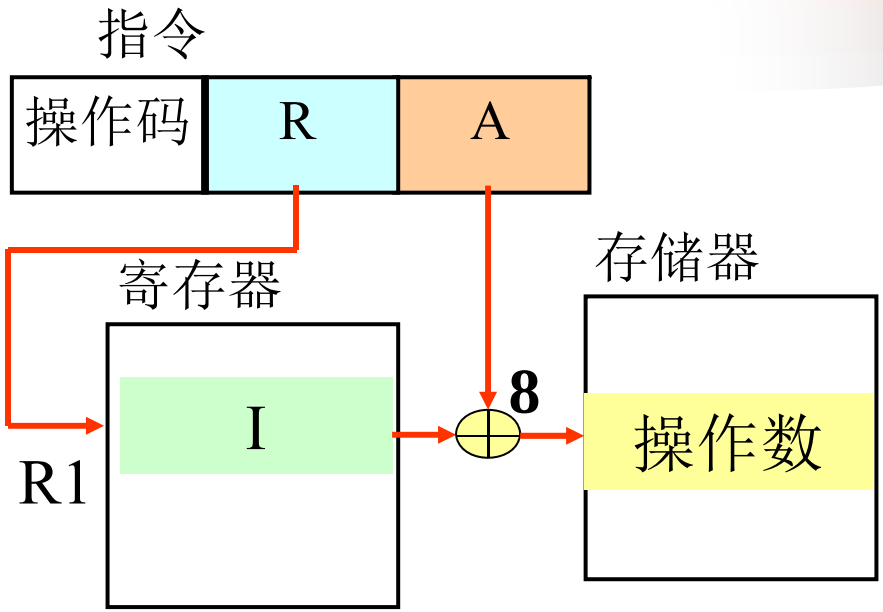
操作数寻址方式（续）

6. 相对寻址

如INC 8(PC)



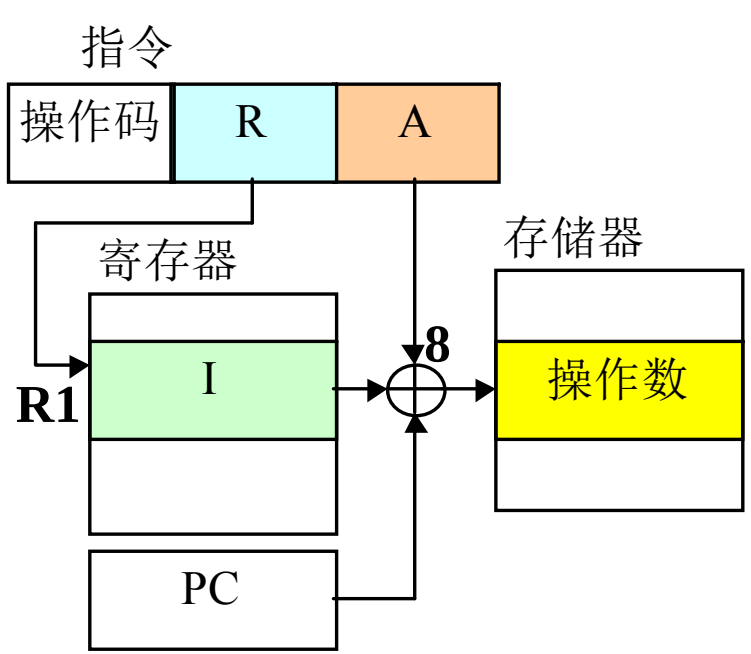
7. 变址和基址寻址
如INC 8(R1)

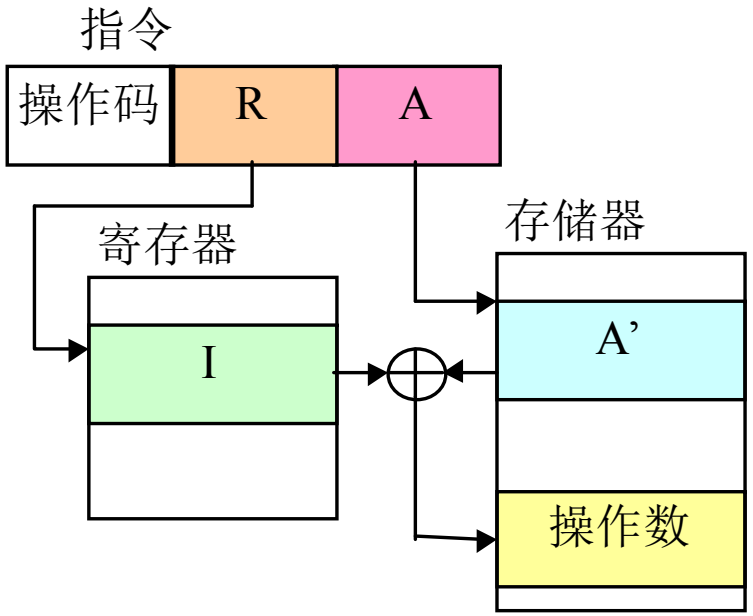


- 变址寻址 **indexed mode**
  - –便于数组访问
- 基址寻址 **based mode**
  - –可扩大寻址范围，可实现程序浮动

8. 复合寻址

如: INC 8(PC+R1)
INC (R1)(1000)





各种常见寻址方式的汇编指令表示

寻址方式	汇编表示	操作内容
直接寻址	load adr	ac←mem[adr]
间接寻址	load (adr)	ac←mem[mem[adr]]
相对寻址	load adr(pc)	ac←mem[pc+adr]
立即寻址	load #n	ac←n
变址寻址	load adr(rn)	ac←mem[adr+rn]
寄存器寻址	load rn	ac←rn
寄存器间接寻址	load (rn)	ac←mem[rn]

指令系统

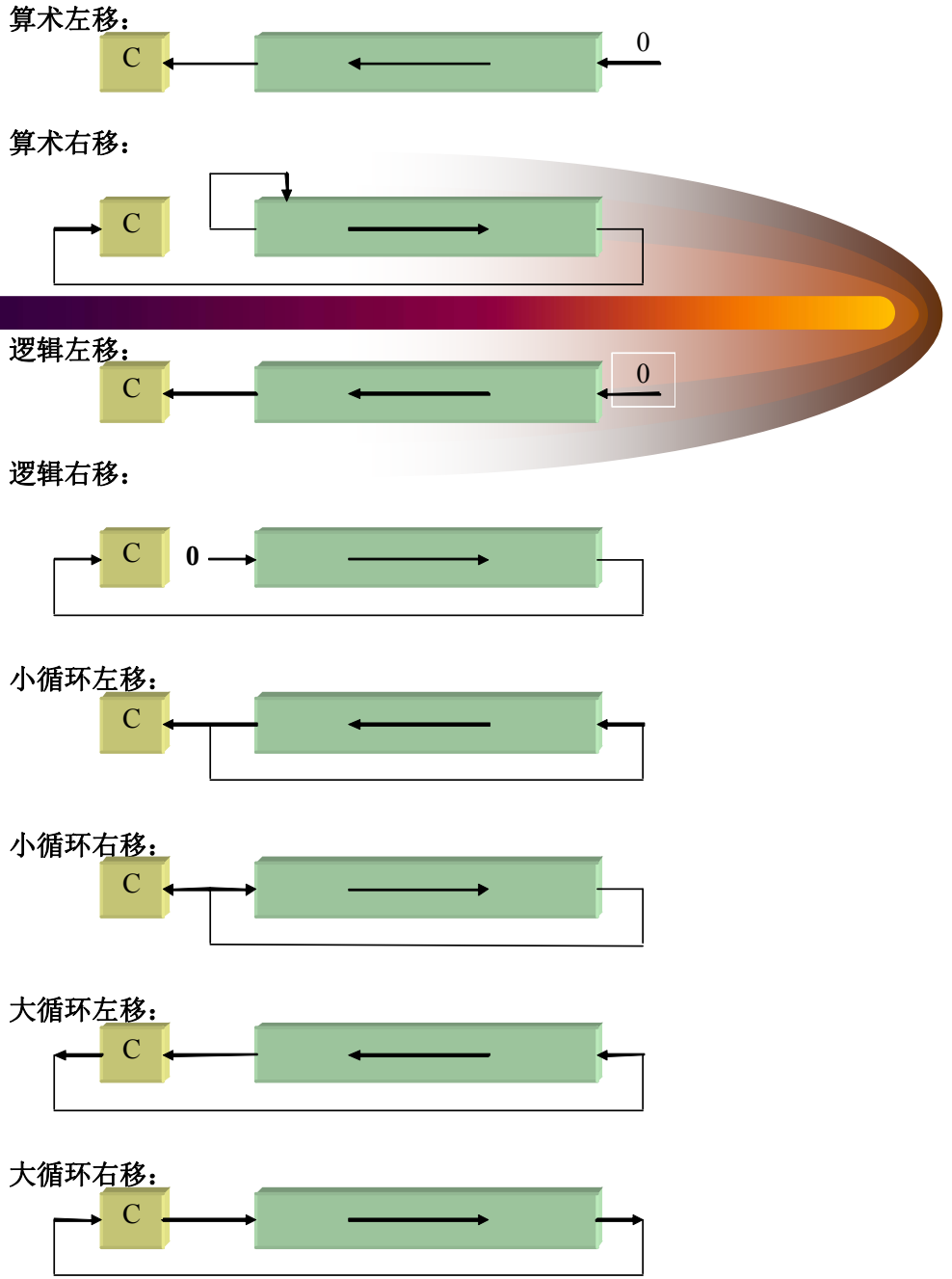
- **一、指令集设计原则**
  - 完备性：能够覆盖所需的各种功能
  - 正交性：无功能完全相同的指令
  - 可扩充性：保留一定余量的操作码空间以供以后功能扩展。
  - 有效性：利用该指令系统编写的程序能高效地运行。
  - 兼容性：机器指令的通用性。

常见指令类型

- 数据传送： move, load, store
- 算术运算： add, sub, mult, div, comp
- 逻辑运算： and, or, neg, shift

移位运算

shift



Data shift operations

指令类型

- 数据传送：**move, load, store**
- 算术运算：**add, sub, mult, div, comp**
- 逻辑运算：**and, or, neg, shift**
- 程序控制：**jump, branch, jsr, ret, int**
- 输入输出：**in, out**
- 堆栈操作：**push, pop**
- 字符串：如**alpha**中**CMPBGE, INSWH**等
- 多媒体指令：**DSP**
- 系统指令：如奔腾机中

程序流控制-转移指令

- 条件转移 vs 无条件转移
- 相对转移 vs 绝对转移
- 状态寄存器SR

- **N**：符号标志位 **Negative**
- **Z**：全零标志位 **Zreo**
- **V**：溢出标志位 **Overflow**
- **C**：进位/借位标志位 **Carry**
- **P**：奇偶标志位 **Parity**

程序流控制-转子指令

- 功能
  - 实现子程序调用
- 步骤
  - 将下一条指令的地址(PC的值)存放在一个临时存储位置
  - 将子程序的起始地址装入PC中
- 子程序返回指令
  - 将存放在临时存储位置的指令地址取出
  - 放回PC

CISC和RISC

- 复杂指令系统计算机（**CISC**）
  - **Complex Instruction Set Computer**
  - 指令系统复杂：指令数目大于**200**条，寻址方式多，指令格式多。
  - 指令串行执行：执行步骤多，需要多个时钟周期。
  - 各种指令都可访问存储器。
  - 有较多专用寄存器。
  - 编译程序难以进行高效的代码优化。

CISC和RISC

- 精简指令系统计算机（**RISC**）
  - **Reduced Instruction Set Computer**
  - 简化的指令系统
  - 以寄存器-寄存器方式工作
  - 采用流水技术
  - 使用较多的通用寄存器以减少访存
  - 采用优化编译技术

	CISC Complex Instruction Set Computer	RISC Reduced Instruction Set Computer
指令种类和长度	种类多，长度可变	种类少，长度固定
指令操作数	**Mem[addr]，Reg，Imm**	**Reg**
寻址方式	多	少
访存指令	无特别限制	只有**Load/store**指令允许访问存储器
流水设计	难	容易
硬件设计	复杂	简单
编译器	简单	复杂
典型代表	**MCS-51，Intel X86**	**MIPS，PowerPC，ARM，MPS430，PIC的MCU**

采用RISC设计思想的计算机产品

- HP公司的Alpha系列、PA-RISC系列
- IBM和Motorola公司的PowerPC系列
- SGI公司的MIPS系列
- SUN公司的SPARC系列
- 嵌入式处理器：ARM系列

寻址方式

寻址方式	地址生成算法
立即	操作数 = A
寄存器	EA = R
偏移量	EA = (SR)+ A
基址	EA = (SR)+(B)
基址加偏移量	EA = (SR) + (B) + A
比例索引加偏移量	EA = (SR) + (I)×S + A
基址加索引和偏移量	EA = (SR) + (B) + (I) + A
基址加比例索引和偏移量	EA = (SR) + (I)×S + (B) + A
相对	EA = (PC) + A

数据类型

数据类型	描述
通用	二进制表示的字节、字、双字、四倍字
整型	补码表示的字节、字、双字
序数型	无符号的字节、字、双字整型数据
BCD	用一个字节表示的一位 0 到 9 之间的十进制数
紧缩 BCD	用一个字节表示的两位 0 到 99 之间的十进制数
近指针	一个 32 位有效地址，代表段内或不分段内存的偏移量
位段	一个连续位串，每一位作为独立的单元，长度可达 2^{32} - 1 位
字节串	一个连续的字节、字或双字串，长度为 0 到 2^{32} - 1 字节
浮点数	IEEE 754 标准的单精度、双精度、扩展精度数据

指令类型——数据传输指令

指令	描述
MOV	寄存器和存储器之间移动数据
PUSH	将操作数压栈
PUSHA	将所有寄存器压栈
MOVSX	符号扩展移动字节、字、双字
LEA	装入有效地址
XLAT	查表转换
IN,OUT	对 I/O 空间的输入输出操作

指令类型——算术指令

指令	描述
ADD	操作数加法
SUB	操作数减法
MUL	无符号单/双精度乘法
IDIV	整型数除法
AND	“与”操作
BTS	位测试并置 1
BSF	前向位扫描
SHL/SHR	逻辑左移/右移
SAL/SAR	算术左移/右移
ROL/ROR	循环左移/右移
SETcc	根据条件设置字节为 0 或 1

指令类型——程序转移指令

指令	描述
CALL	子程序调用
JMP	无条件转移
JE	相等/为零时转移
LOOPE	相等/为零时循环
INT	中断
INTO	溢出时中断

指令类型——

- 串操作指令
  - MOVS 移动字节、字、双字串
  - LODS　　装入字节、字、双字串
- 支持高级语言指令
  - ENTER　　建立一个堆栈框架
  - LEAVE　　恢复ENTER指令的动作
  - BOUND　　检查数组的值在上下界之间
-

指令数量统计

CPU	指令数
8086/8087	89/77
I286	190
I386	204
i486	216
Pentium	222
Pentium Pro	230
Pentium MMX	287
Pentium II	291
Pentium III	361
Pentium 4	505

Intel Architecture Processors

Intel Processor	Date of Product Intro-duction	Perform-ance_ in MIPs[1]	Max. CPU Frequency at Intro-duction	No. of Transis-tors on the Die	Main CPU Register Size[2]	Extern. Data Bus Size[2]	Max. Extern. Addr. Space	Caches in CPU Pack-age[3]
8086	1978	0.8	8 MHz	29 K	16	16	1 MB	None
Intel 286	1982	2.7	12.5 MHz	134 K	16	16	16 MB	Note 3
Intel386™ DX	1985	6.0	20 MHz	275 K	32	32	4 GB	Note 3
Intel486™ DX	1989	20	25 MHz	1.2 M	32	32	4 GB	8KB L1
Pentium®	1993	100	60 MHz	3.1 M	32	64	4 GB	16KB L1
Pentium® Pro	1995	440	200 MHz	5.5 M	32	64	64 GB	16KB L1; 256KB or 512KB L2
Pentium II®	1997	466	266	7 M	32	64	64 GB	32KB L1; 256KB or 512KB L2
Pentium® III	1999	1000	500	8.2 M	32 GP 128 SIMD-FP	64	64 GB	32KB L1; 512KB L2