计算机组成原理

第七章 指令系统

阙夏
计算机与信息学院
2025/5/13

大　纲

指令系统

(一)指令系统的基本概念

(二)指令格式

(三)寻址方式

(四)数据的对齐和大/小端存放方式

(五)CISC和RISC的基本概念

(六)高级语言程序与机器级代码之间的对应

1.编译器，汇编器和链路器的基本概念

2.选择结构语句的机器级表示

3.循环结构语句的机器级表示

4.过程(函数)调用对应的机器级表示

Contents

7.1 机器指令

一、指令格式

指令：就是让计算机识别并执行某种操作的命令。

指令系统：一台计算机中所有机器指令的集合。
称为这台计算机的指令系统。

系列计算机：指基本指令系统相同且基本体系，结构相同的一系列计算机。

系列机能解决软件兼容问题的必要条件是该系列的各种机种有共同的指令集，而且新推出的机种的指令系统一定包含旧机种的所有指令，因此在旧机种上运行的各种软件可以不加任何修改地在新机种上运行。

教材P319“向上兼容”

7.1 机器指令

指令的一般格式：

n位操作码字段的指令系统最多能够表示2^n条指令。

操作码字段	地址码字段

1. 操作码　指令应该执行什么性质的操作和具有何种功能。反映机器做什么操作。

(1) **长度固定**（译码简单）

用于指令字长较长的情况，RISC

如 IBM 370　　操作码 8 位

(2) **长度可变**

操作码分散在指令字的不同字段中

(3) 扩展操作码技术（教材P301）

操作码的位数随地址数的减少而增加

OP	A_1	A_2	A_3	
4 位操作码	0000	A_1	A_2	A_3	最多15条三地址指令
	0001	A_1	A_2	A_3	
	⋮	⋮	⋮	⋮	
	1110	A_1	A_2	A_3	
8 位操作码	1111	0000	A_2	A_3	最多15条二地址指令
	1111	0001	A_2	A_3	
	⋮	⋮	⋮	⋮	
	1111	1110	A_2	A_3	
12 位操作码	1111	1111	0000	A_3	最多15条一地址指令
	1111	1111	0001	A_3	
	⋮	⋮	⋮	⋮	
	1111	1111	1110	A_3	
16 位操作码	1111	1111	1111	0000	16条零地址指令
	1111	1111	1111	0001	
	⋮	⋮	⋮	⋮	
	1111	1111	1111	1111	

(3)扩展操作码技术

操作码的位数随地址数的减少而增加

OP	A_1	A_2	A_3
4 位操作码			
0000	A_1	A_2	A_3
0001	A_1	A_2	A_3
⋮	⋮	⋮	⋮
1110	A_1	A_2	A_3
8 位操作码			
1111	0000	A_2	A_3
1111	0001	A_2	A_3
⋮	⋮	⋮	⋮
1111	1110	A_2	A_3
12 位操作码			
1111	1111	0000	A_3
1111	1111	0001	A_3
⋮	⋮	⋮	⋮
1111	1111	1110	A_3
16 位操作码			
1111	1111	1111	0000
1111	1111	1111	0001
⋮	⋮	⋮	⋮
1111	1111	1111	1111

两个原则：
1.编码不能重复
2.短码不能为长码前缀

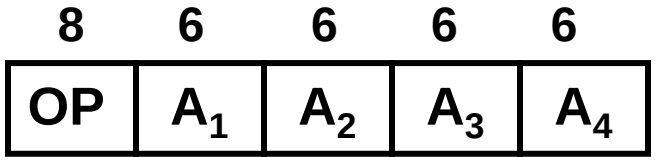
三地址指令操作码
每减少一种最多可多构成
2^4 种二地址指令

二地址指令操作码
每减少一种最多可多构成
2^4 种一地址指令

使用频率高用短码
使用频率低用长码

2. 地址码

(1) 四地址



A_1 第一操作数地址

A_2 第二操作数地址

A_3 结果的地址

A_4 下一条指令地址

(A_1) OP $(A_2) \longrightarrow A_3$

设指令字长为 32 位

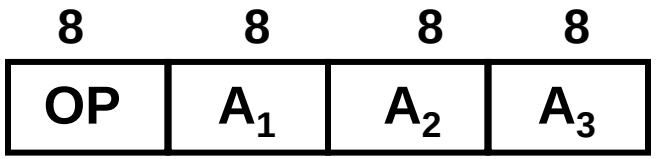
操作码固定为 8 位

4 次访存

寻址范围 $2^6 = 64$

若 PC 代替 A_4

(2) 三地址



(A_1) OP $(A_2) \longrightarrow A_3$

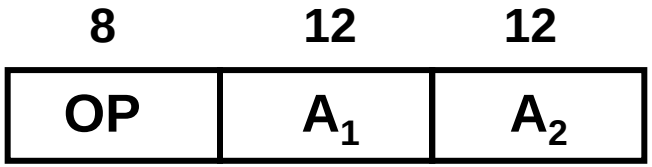
4 次访存

寻址范围 $2^8 = 256$

若 A_3 用 A_1 或 A_2 代替

7.1 机器指令

(3) 二地址



(A_1) OP $(A_2) \longrightarrow A_1$　　4 次访存

或

(A_1) OP $(A_2) \longrightarrow A_2$　　寻址范围 $2^{12} = 4$ K

若结果存于 ACC　　3次访存　　若ACC 代替 A_1（或A_2）

(4) 一地址

2 次访存

(ACC) OP $(A_1) \longrightarrow$ ACC　　寻址范围 $2^{24} = 16$ M

(5) 零地址　　无地址码

1.无需任何操作数，如空操作指令、停机指令等；2.所需操作数地址是默认的。

二、指令字长

指令字长决定于
- 操作码的长度
- 操作数地址的长度
- 操作数地址的个数

1. 指令字长 固定

指令字长 = 存储字长 = 机器字长（早期计算机）

2. 指令字长 可变

按字节的倍数变化

小结

- **当用一些硬件资源代替指令字中的地址码字段后**
  - 可扩大指令的寻址范围
  - 可缩短指令字长
  - 可减少访存次数
- **当指令的地址字段为寄存器时**

  三地址　OP　R_1，R_2，R_3

  二地址　OP　R_1，R_2

  一地址　OP　R_1

  - 可缩短指令字长
  - 指令执行阶段不访存

单选题 1分

设置

【2017统考真题】某计算机按字节编址，指令字长固定且只有两种指令格式，其中三地址指令29条、二地址指令107 条，每个地址字段为6位，则指令字长至少应该是( )。

A 24位

B 26位

C 28位

D 32位

提交

7.2 操作数类型和操作种类

一、操作数类型

地址	绝对地址：无符号整数；相对地址：有符号数
数字	定点数、浮点数、十进制数
字符	ASCII
逻辑数	逻辑运算

二、数据在存储器中的存放方式

例7.1 12345678H，分别用大端小端方式存储

存储器中的数据存放：边界对齐

字地址	字节地址			
0	0	1	2	3
4	4	5	6	7
8	8	9	10	11

字地址	字节地址			
0	12H	34H	56H	78H
4	4	5	6	7
8	8	9	10	11

高位字节 地址为字地址（大端）

字地址	字节地址			
0	78H	56H	34H	12H
4	4	5	6	7
8	8	9	10	11

低位字节 地址为字地址（小端）

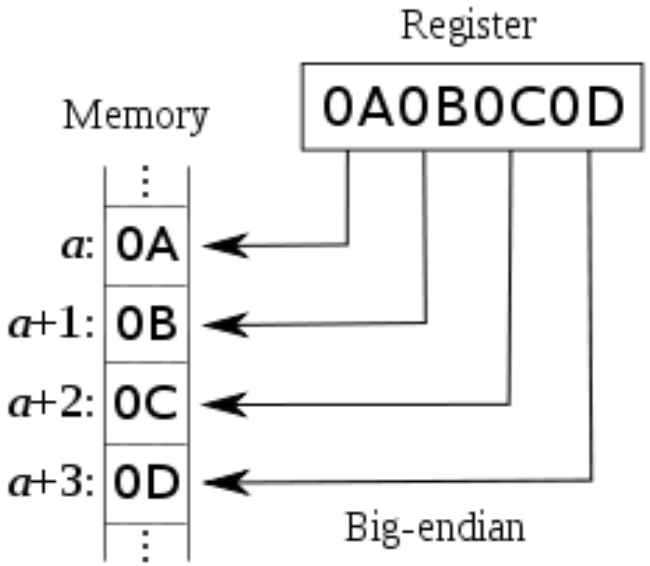
BIG Endian versus Little Endian

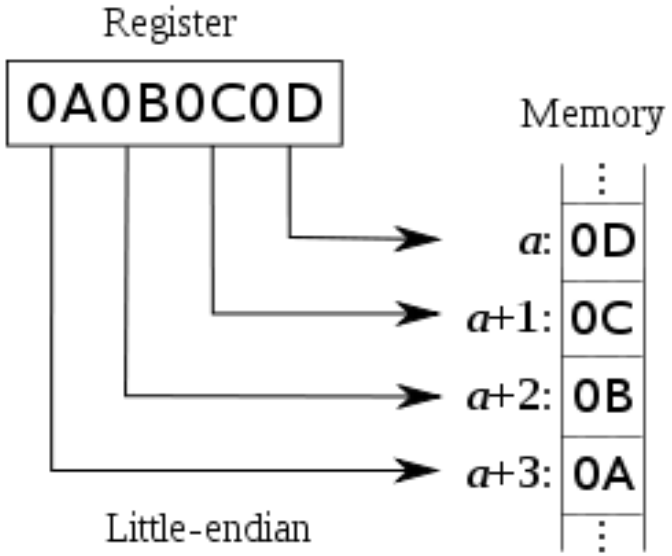
大端Big-Endian

低地址存放最高有效位（**MSB**），既高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

小端Little-Endian

低地址存放最低有效位（LSB），既低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。





7.2 操作数类型和操作种类

三、操作类型

1. 数据传送

源	寄存器	寄存器	存储器	存储器
目的	寄存器	存储器	寄存器	存储器
例如	MOVE	STORE MOVE PUSH	LOAD MOVE POP	MOVE

置“1”，清“0”

2. 算术逻辑操作

加、减、乘、除、增 1、减 1、求补、浮点运算、十进制运算

与、或、非、异或、位操作、位测试、位清除、位求反

如 8086

ADD SUB MUL DIV INC DEC CMP NEG

AAA AAS AAM AAD

AND OR NOT XOR TEST

7.2 操作数类型和操作种类

3. 移位操作

算术移位　　逻辑移位

循环移位（带进位和不带进位）

4. 转移

(1) 无条件转移　JMP

(2) 条件转移

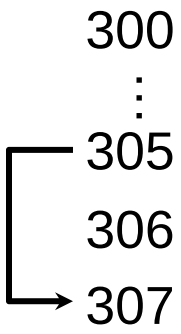
结果为零转　(Z = 1)　JZ

结果溢出转　(O = 1)　JO

结果有进位转　(C = 1)　JC

跳过一条指令　SKP

如

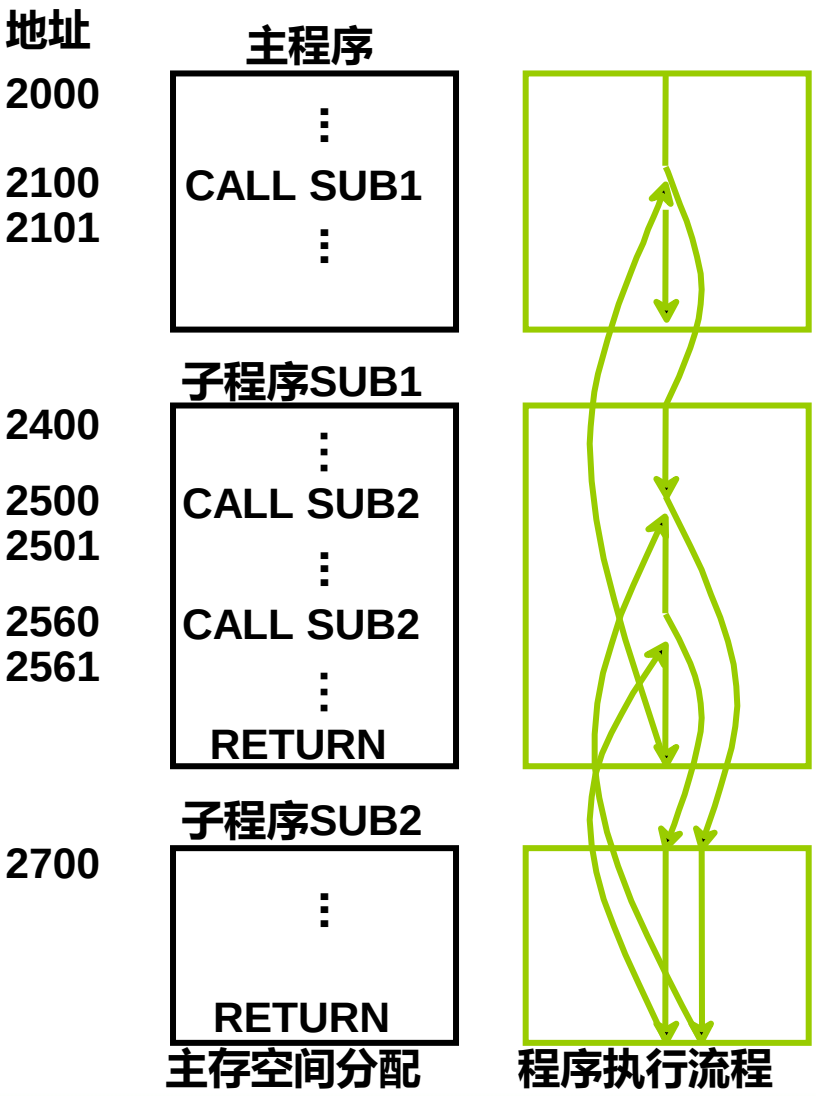


完成触发器

SKP DZ D = 0 则跳

7.2 操作数类型和操作种类

(3) 调用和返回



7.2 操作数类型和操作种类

(4) 陷阱（Trap）与陷阱指令

陷阱：计算机系统的意外事故（如电压不稳。故障等）。
陷阱指令：是处理陷阱的指令。

- 一般不提供给用户直接使用

  在出现事故时，由 CPU 自动产生并执行（隐指令）

- 设置供用户使用的陷阱指令

  如 8086　　INT　TYPE　　软中断

  提供给用户使用的陷阱指令，完成系统调用

5. 输入输出

入　　端口地址 ⟶ CPU 的寄存器

如　　IN　AL, *m*　　IN　AL, DX

出　　CPU 的寄存器 ⟶ 端口地址

如　　OUT　*n*, AL　　OUT　DX, AL

7.3 寻址方式

寻址方式

确定 本条指令 的 **操作数地址**

下一条 欲执行 指令 的 **指令地址**

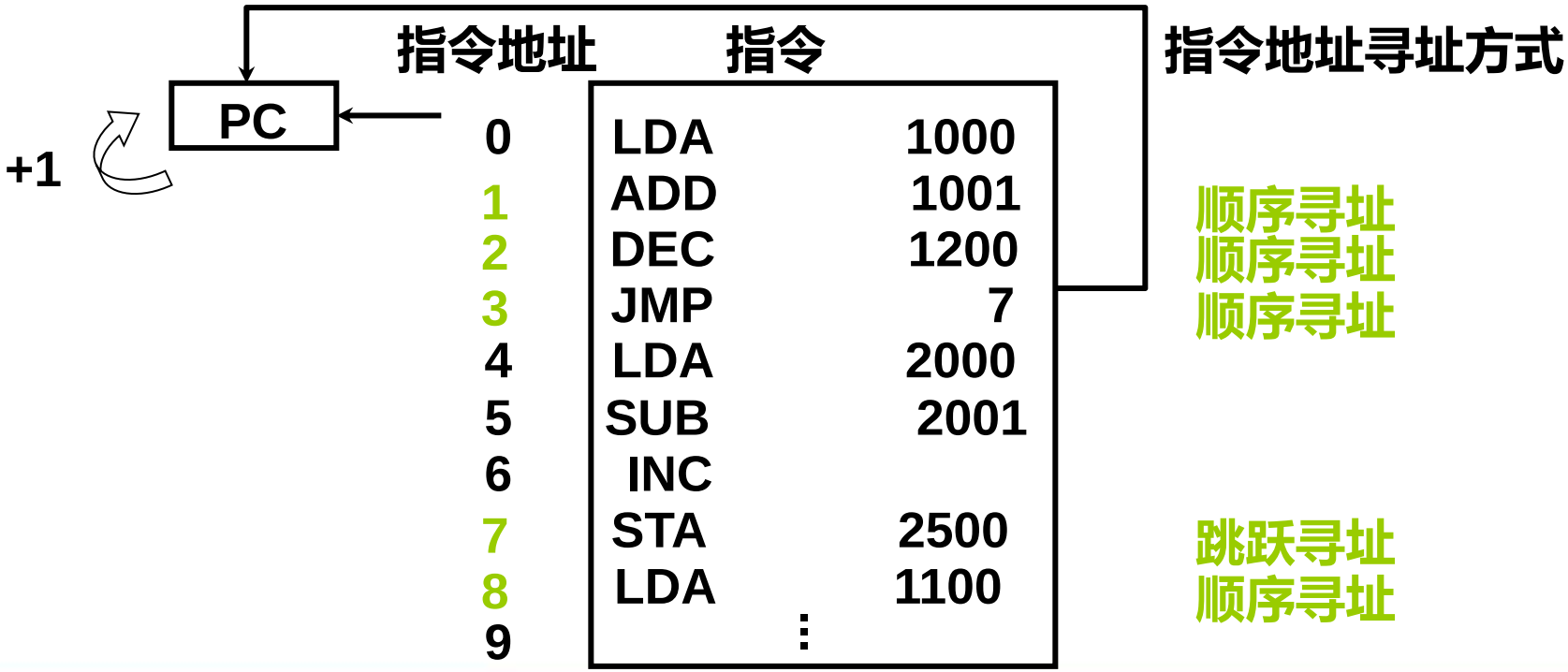
寻址方式
- 指令寻址
- 数据寻址

7.3 寻址方式

一、指令寻址

这里的+1是指下一条指令

- 顺序：(PC) + 1 ⟶ PC
- 跳跃：由转移指令指出，如 JMP



7.3 寻址方式

二、数据寻址

操作码	寻址特征	形式地址 A

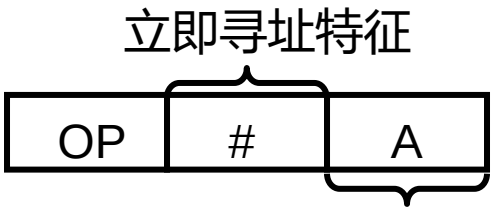
形式地址 指令字中的地址 **A**

有效地址 操作数的真实地址 **EA**

约定 **指令字长 = 存储字长 = 机器字长**

1. 立即寻址

形式地址 A 就是操作数



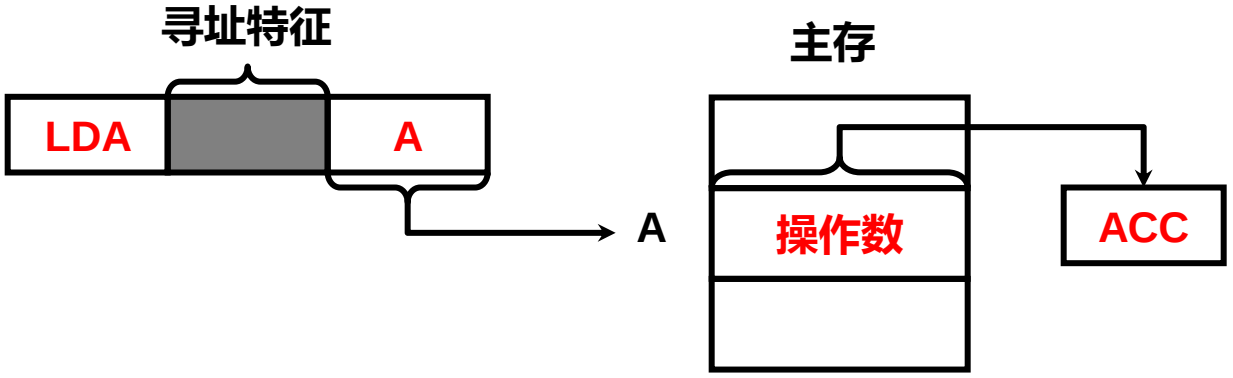
可正可负 **补码**

- **指令执行阶段不访存**
- **A 的位数限制了立即数的范围**

7.3 寻址方式

2. 直接寻址

EA = A　　有效地址由形式地址直接给出

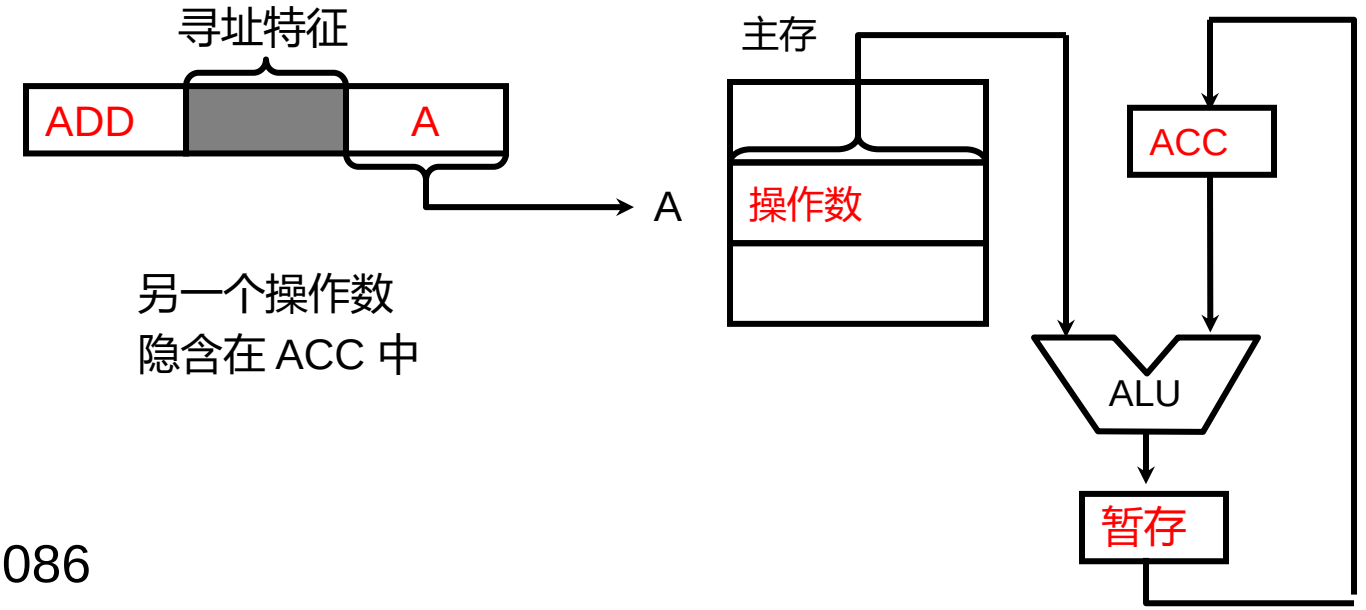


- 执行阶段访问一次存储器
- A 的位数决定了该指令操作数的寻址范围
- 操作数的地址不易修改（必须修改A）

7.3 寻址方式

3. 隐含寻址

操作数地址隐含在操作码中



如 8086

MUL 指令　　被乘数隐含在 AX（16位）或 AL（8位）中

MOVS 指令　　源操作数的地址隐含在 SI 中

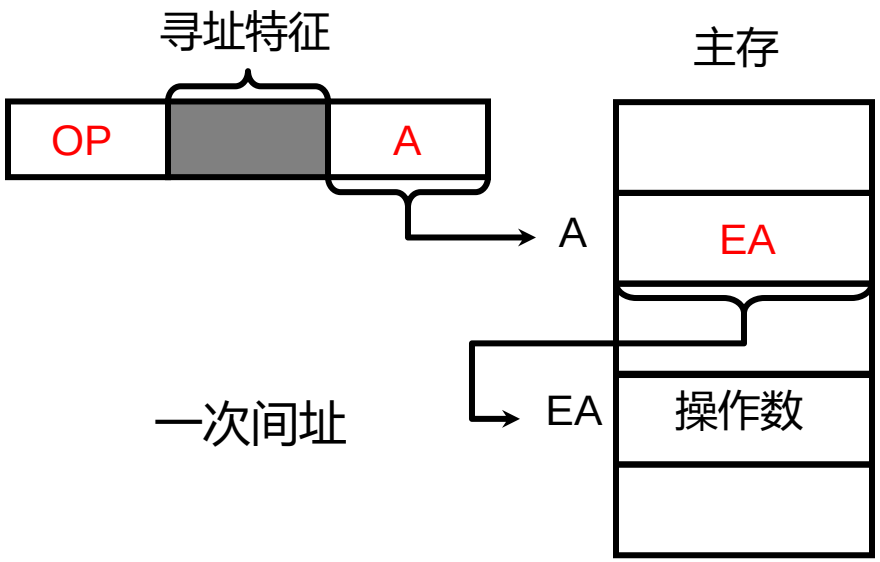
目的操作数的地址隐含在 DI 中

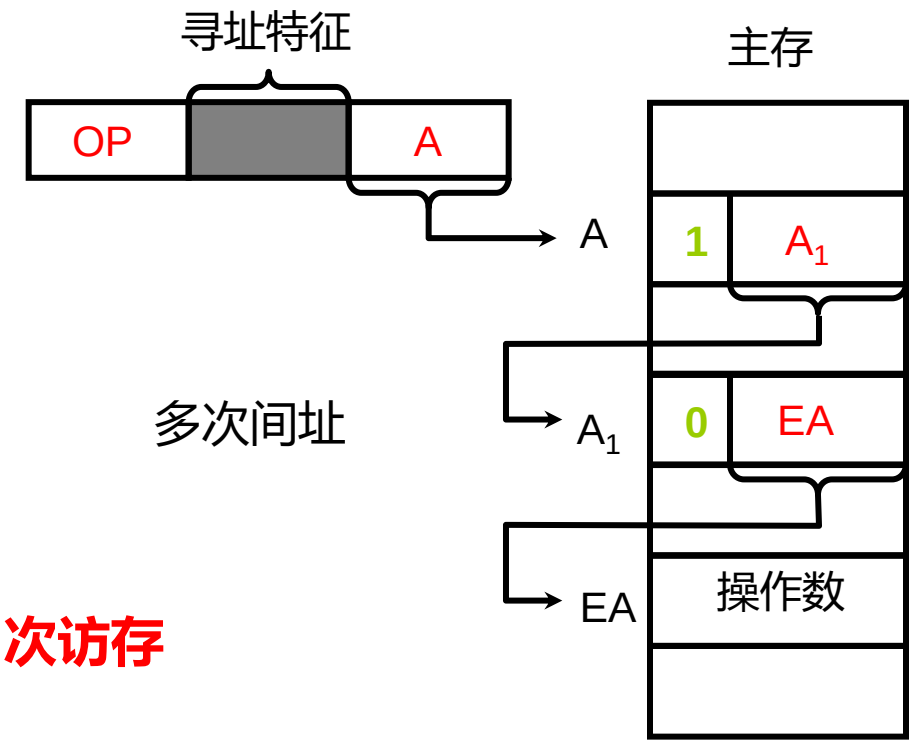
- **指令字中少了一个地址字段，可缩短指令字长**

7.3 寻址方式

4. 间接寻址

EA = （A）　有效地址由形式地址间接提供



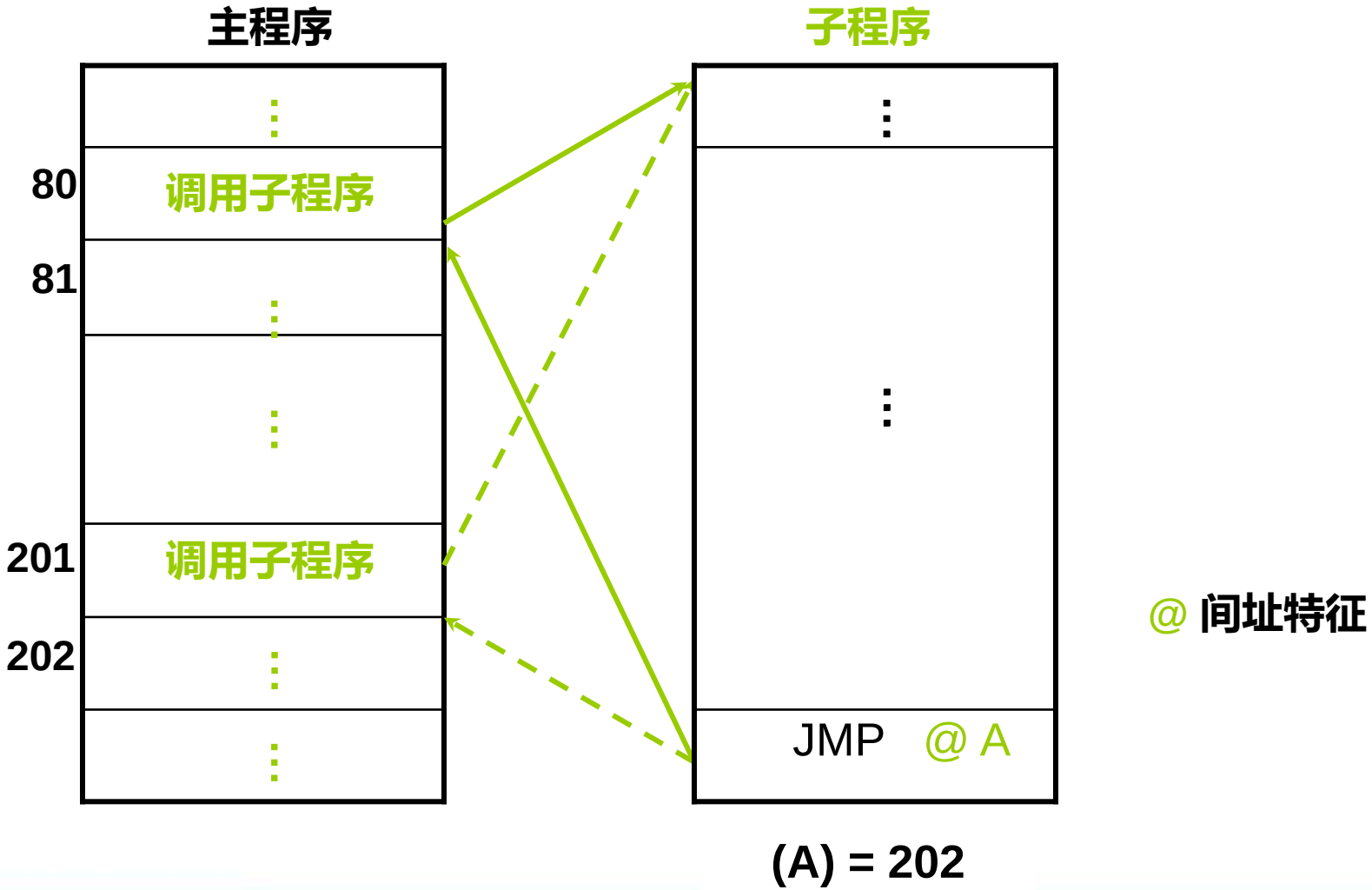


- 执行指令阶段 2 次访存
- 可扩大寻址范围
- 便于编制程序

多次访存

7.3 寻址方式

间接寻址编程举例

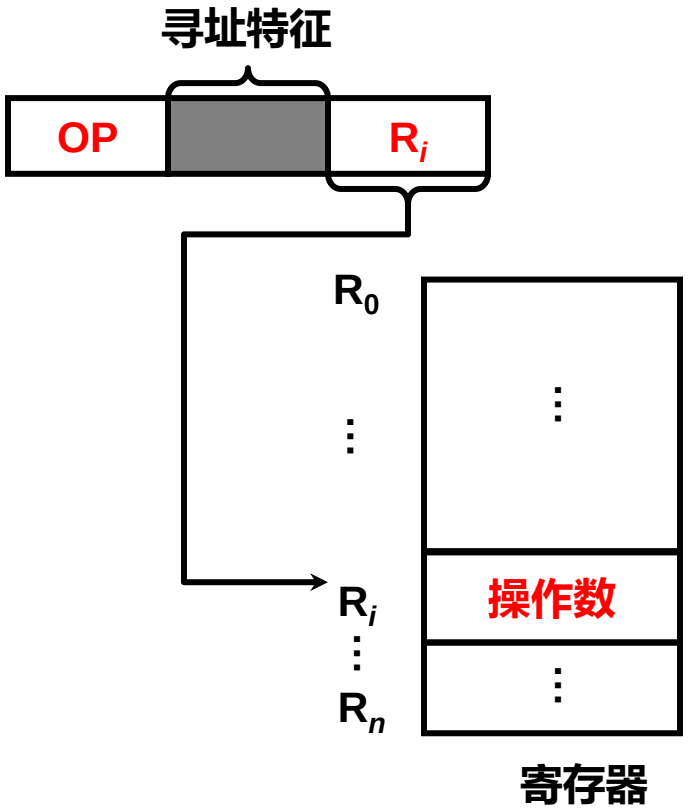


7.3 寻址方式

5. 寄存器（直接）寻址

$EA = R_i$

有效地址即为寄存器编号

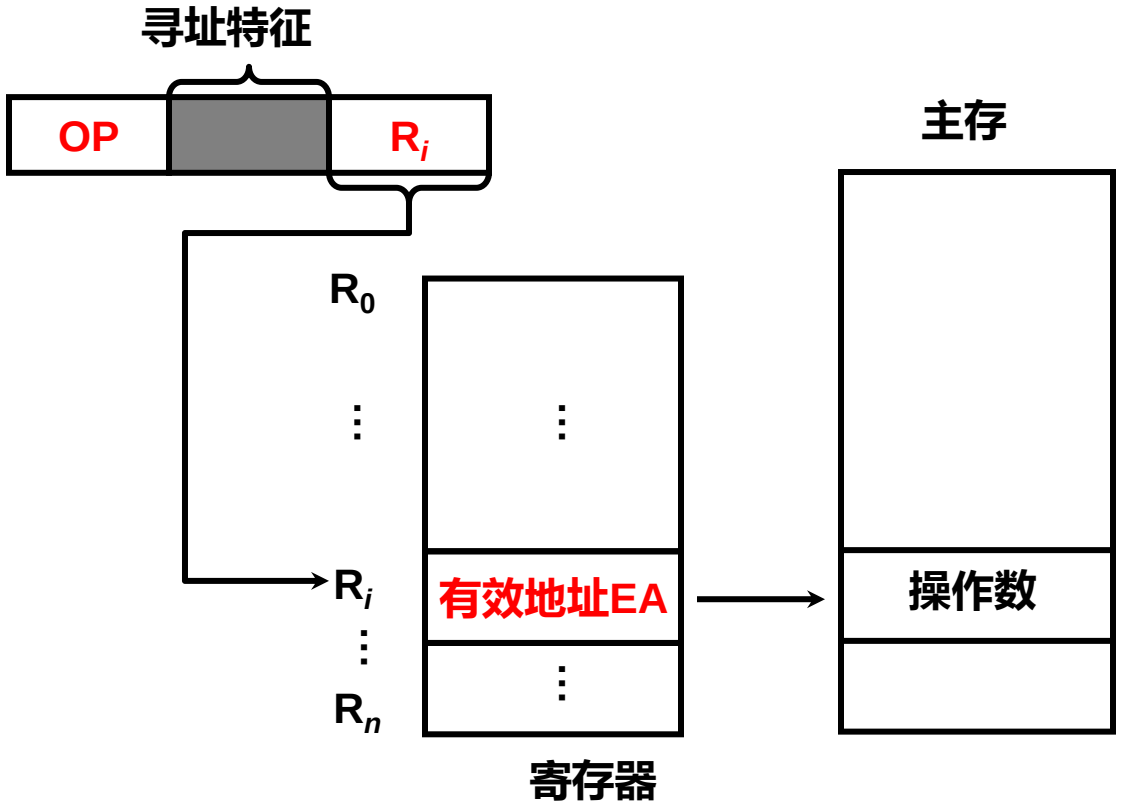


- 执行阶段不访存，只访问寄存器，执行速度快
- 寄存器个数有限，可缩短指令字长

7.3 寻址方式

6. 寄存器间接寻址

EA = (R_i)　　有效地址在寄存器中



- 有效地址在寄存器中， 操作数在存储器中，执行阶段访存
- 比一般间接寻址速度快
- 便于编制循环程序

单选题 1分

设置

_____有利于编制循环程序。

A 基址寻址

B 相对寻址

C 寄存器间接寻址

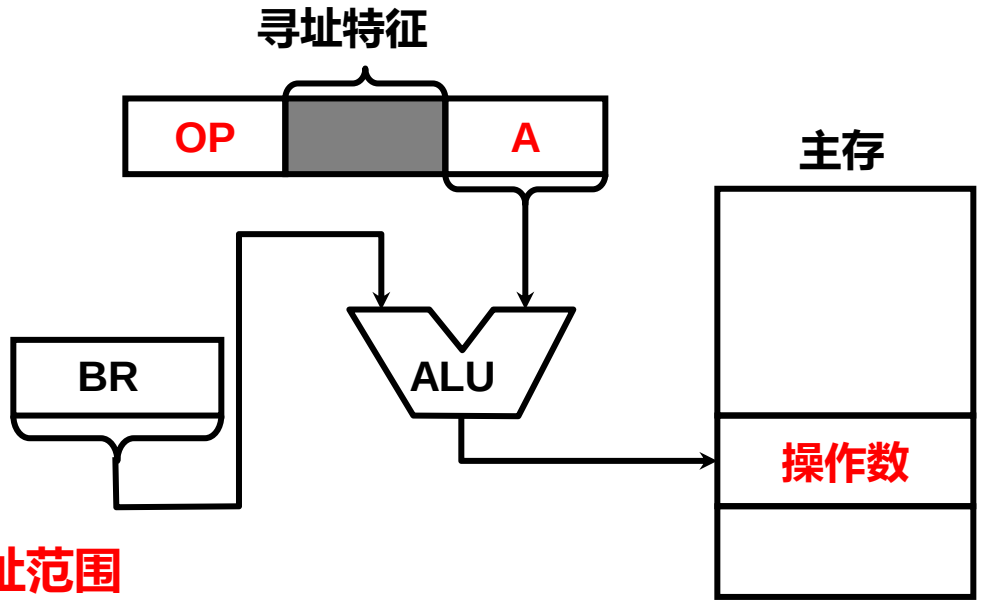
D 直接寻址

提交

7.3 寻址方式

7. 基址寻址 (1) 采用专用寄存器作基址寄存器

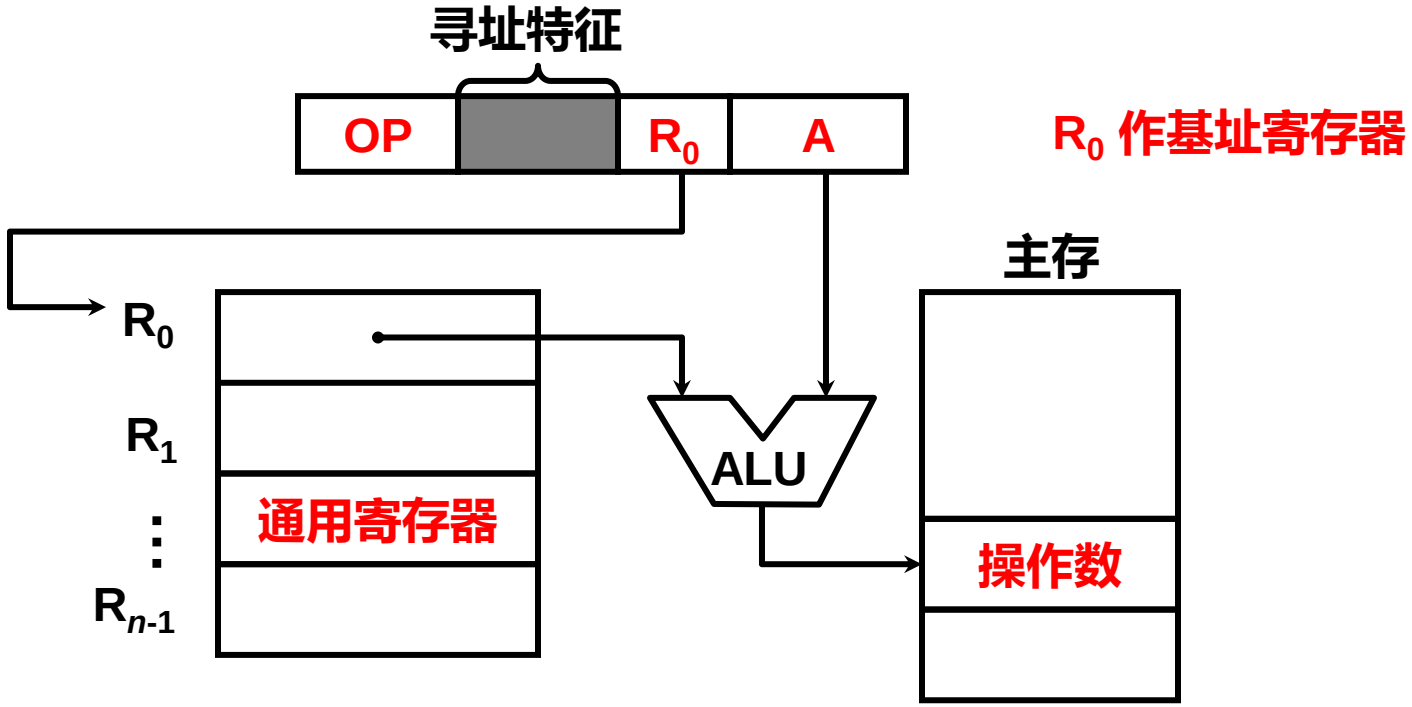
EA = (BR) + A　　BR 为基址寄存器



- 可扩大寻址范围
- 有利于多道/浮动程序，但偏移量偏小
- BR 内容由操作系统或管理程序确定
- 在程序的执行过程中 BR 内容不变，形式地址 A 可变

7.3 寻址方式

(2) 采用通用寄存器作基址寄存器



- 由用户指定哪个通用寄存器作为基址寄存器
- 基址寄存器的内容由操作系统确定
- 在程序的执行过程中 R_0 内容不变，形式地址 A 可变

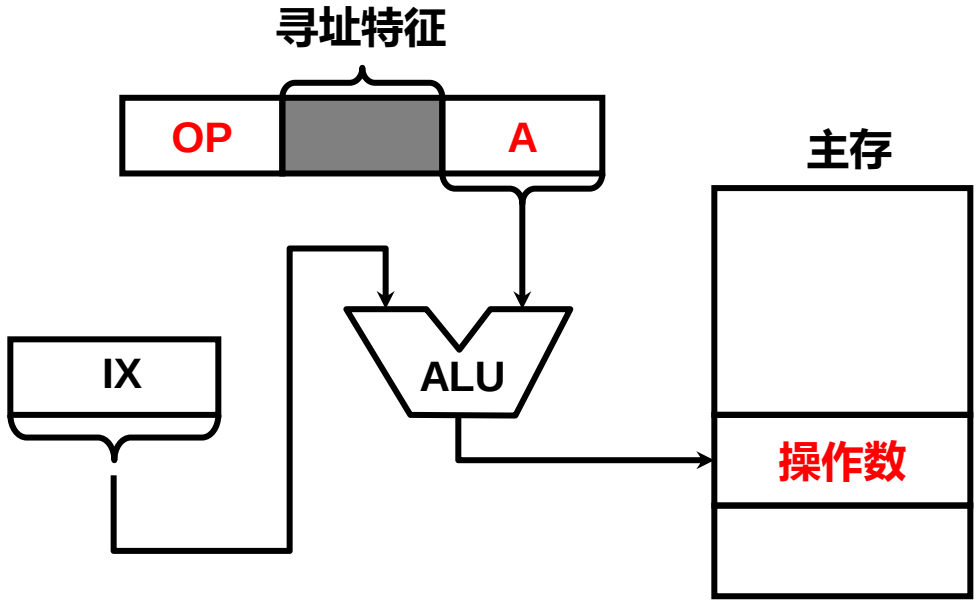
7.3 寻址方式

8. 变址寻址

EA = (IX) +A

IX 为变址寄存器（专用）

通用寄存器也可以作为变址寄存器



- 可扩大寻址范围
- IX 的内容由用户给定
- 在程序的执行过程中 IX 内容可变，形式地址 A 不变
- 便于处理循环/数组问题

例7.2　设数据块首地址为 D，求 *N* 个数的平均值

直接寻址

```
LDA    [D]
ADD    [D + 1]
ADD    [D + 2]
  ⋮
ADD    [D + ( N -1 )]
DIV    # N
STA    ANS
```

共 *N* + 2 条指令

变址寻址

```
      LDA    # 0          0 ⟶ ACC
      LDX    # 0          X 为变址寄存器
M     ADD    X, [D]       D 为形式地址
      INX                 (X) +1 ⟶ X
      CPX    # N          (X) 和 #N 比较
      BNE    M            结果不为零则转
      DIV    # N
      STA    ANS
```

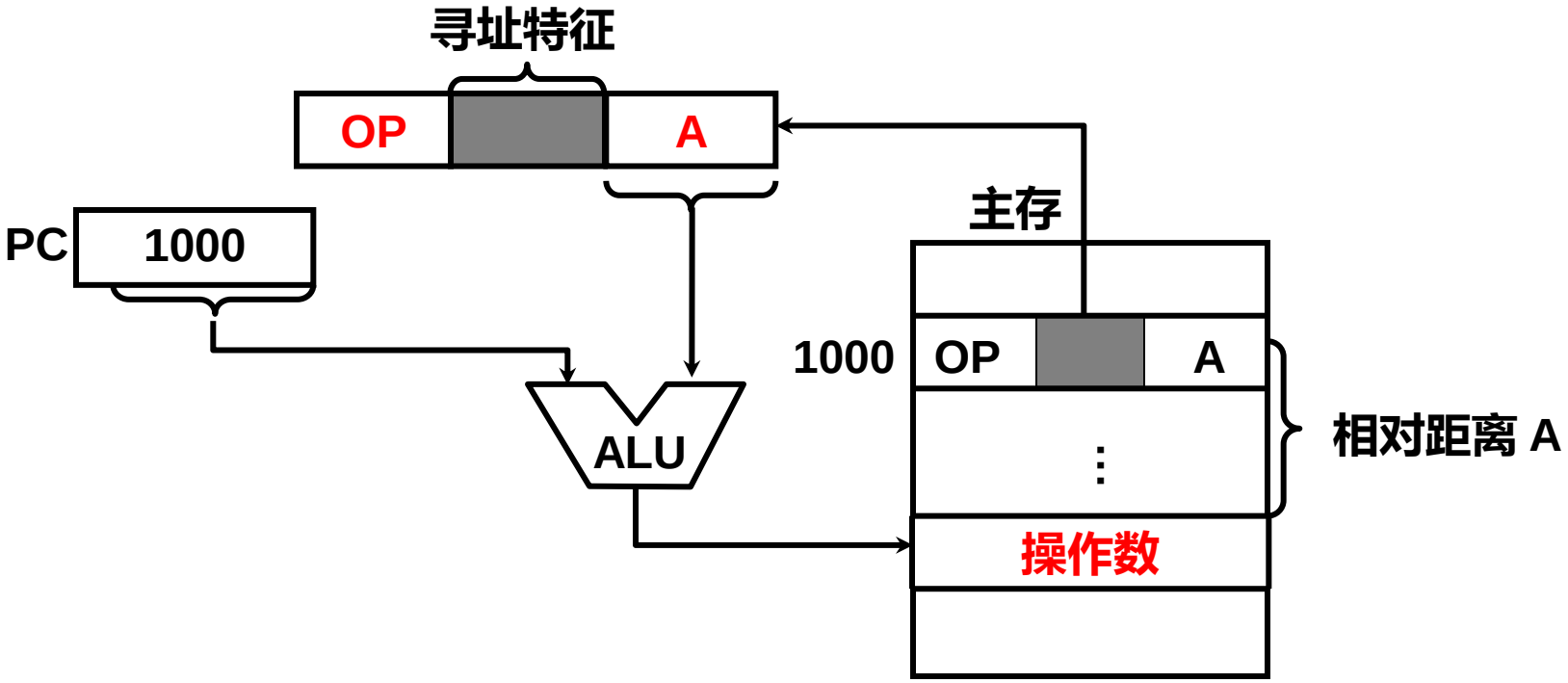
共 8 条指令

7.3 寻址方式

9. 相对寻址

EA = (PC) + A

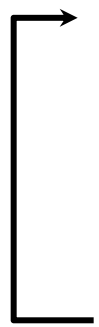
A 是相对于当前指令的位移量（可正可负，补码）



- A 的位数决定操作数的寻址范围
- 可用于编制浮动程序
- 广泛用于转移指令

7.3 寻址方式

(1) 相对寻址举例



	LDA	# 0	
	LDX	# 0	
M	ADD	X, D	
M+1	INX		
M+2	CPX	# N	* 相对寻址特征
M+3	BNE	M	* – 3
	DIV	# N	
	STA	ANS	

M 随程序所在存储空间的位置不同而不同

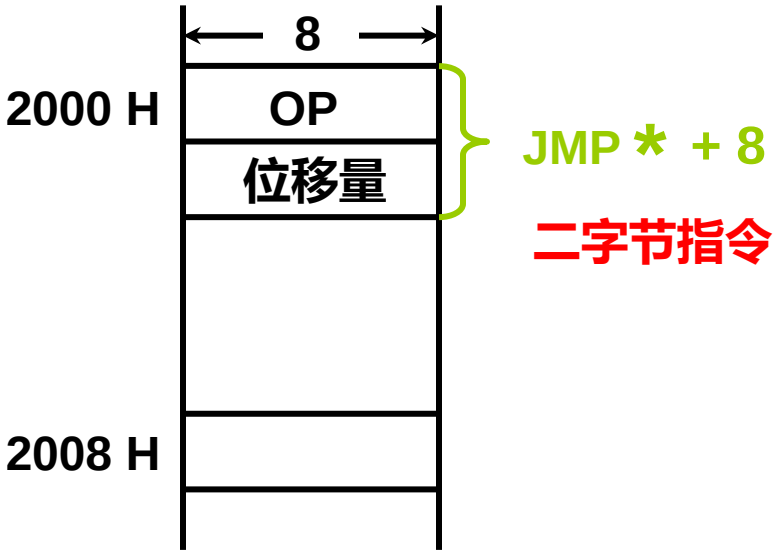
而指令 BNE * – 3 与 指令 ADD X, D 相对位移量不变

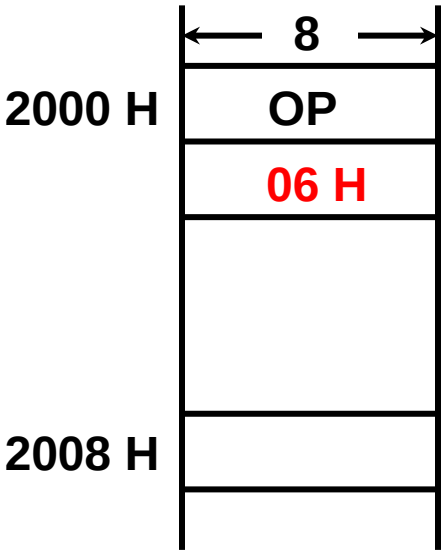
指令 BNE * – 3 操作数的有效地址为

EA = (M+3) – 3 = M

7.3 寻址方式

(2) 按字节寻址的相对寻址举例





设 当前指令地址 PC = 2000H

转移后的目的地址为 2008H

因为 取出 JMP * + 8 后 PC = 2002H

故 JMP * + 8 指令 的第二字节为 2008H - 2002H = 06H

单选题 1分

设置

【2017统考真题】下列寻址方式中，最适合按下标顺序访问一维数组元素的是( ) 。

A 相对寻址

B 寄存器寻址

C 直接寻址

D 变址寻址

提交

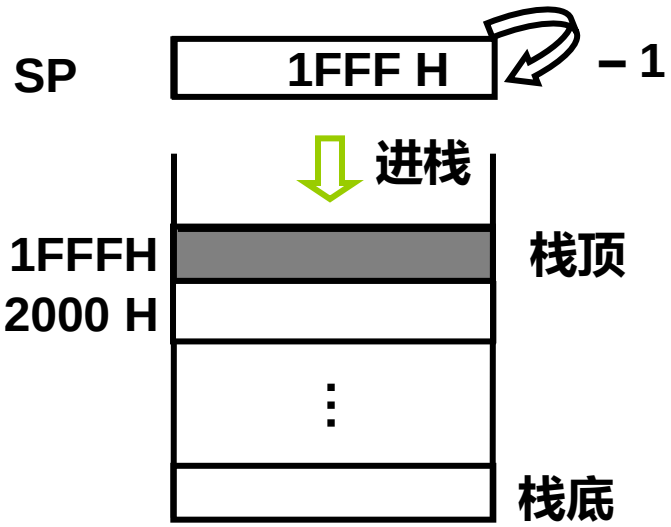
10. 堆栈寻址

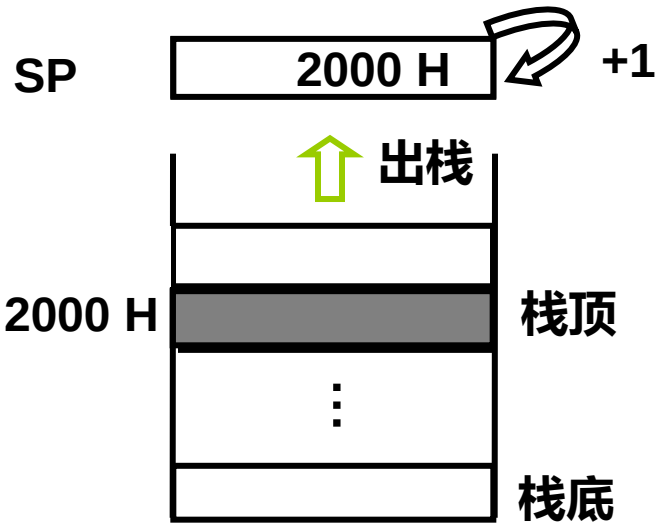
(1) 堆栈的特点

堆栈 { 硬堆栈 多个寄存器 堆栈计算机
软堆栈 指定的存储空间 8086计算机

先进后出（一个入出口）　　**栈顶地址** 由 SP 指出

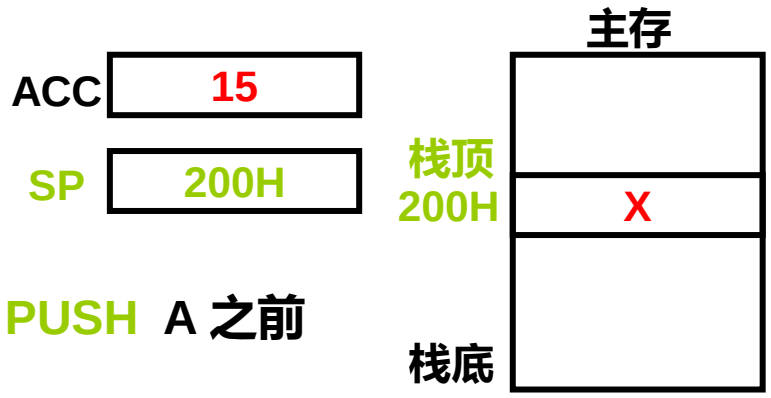
进栈　(SP) – 1 ⟶SP　　**出栈**　(SP) + 1 ⟶ SP

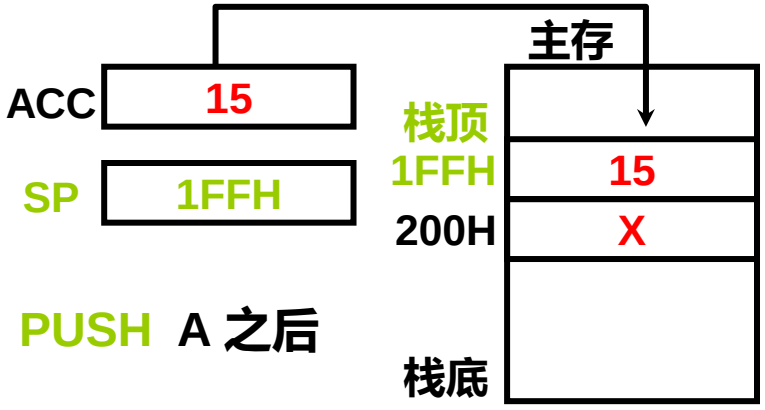


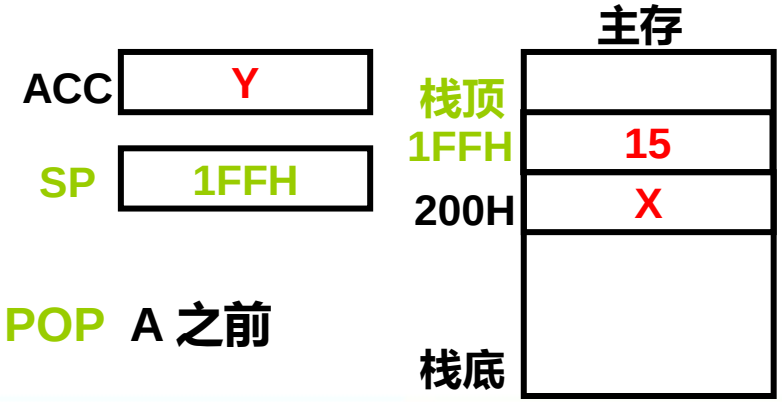


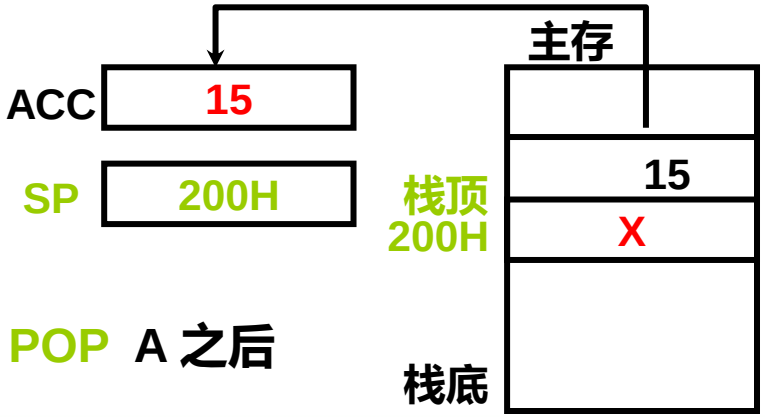
10. 堆栈寻址

(2) 堆栈寻址举例









7.3 寻址方式

*补充：堆栈的分类

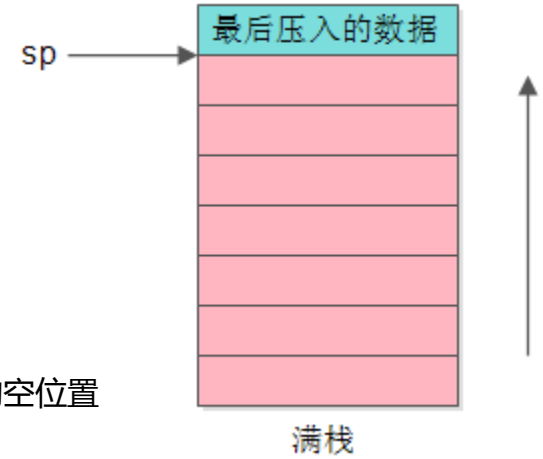
满堆栈：入栈先改SP，再放数；出栈先取数，再改SP；
空堆栈：入栈先放数，再改SP；出栈先改SP，再取数；

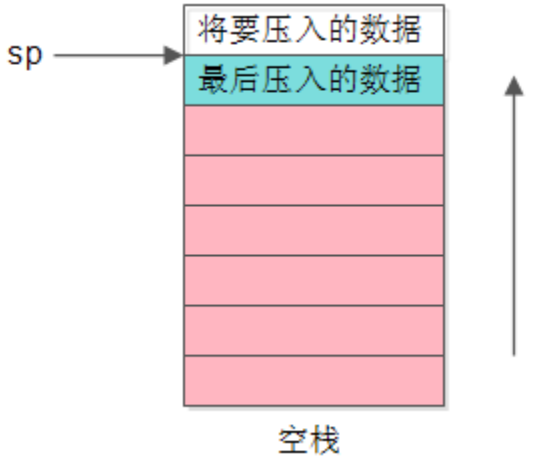
- 1、满/空栈

根据SP指针指向的位置，栈可以分为满栈和空栈。

满栈：当堆栈指针总是指向最后压入堆栈的数据

空栈：当堆栈指针总是指向下一个将要放入数据的空位置





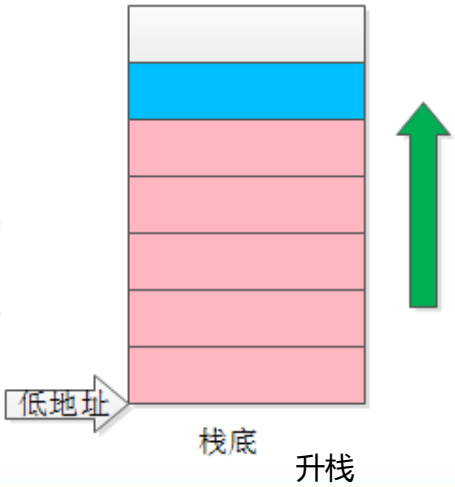
- 2、升（增）/降（减）栈

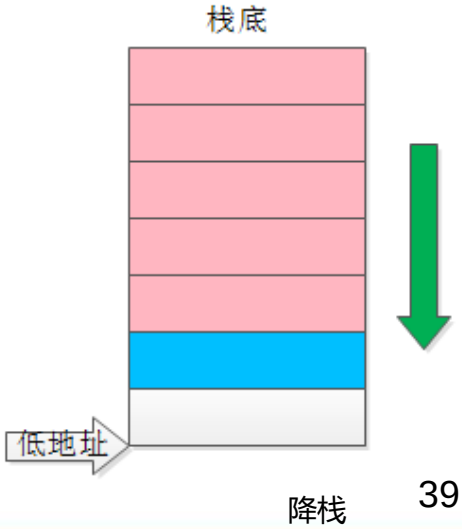
根据SP指针移动的方向，栈可以分为升栈和降栈

升栈：随着数据的入栈，SP指针从低地址->高地址移动

降栈：随着数据的入栈，SP指针从高地址->低地址移动

ARM是满降栈





7.3 寻址方式

(3) SP 的修改与主存编址方法有关

① 按 字 编址

进栈　(SP) – 1 ⟶ SP

出栈　(SP) + 1 ⟶ SP

② 按 字节 编址

存储字长 16 位

进栈　(SP) – 2 ⟶ SP

出栈　(SP) + 2 ⟶ SP

存储字长 32 位

进栈　(SP) – 4 ⟶ SP

出栈　(SP) + 4 ⟶ SP

填空题 2分

设置

假设（R）=1000，（1000）=2000，（2000）=3000，（3000）=5000，（PC）=4000，则以下寻址方式下访问到的指令操作数的值是多少?

(1)直接寻址，操作数值 [填空1]

(2)间接寻址，操作数值 [填空2]

(3) 寄存器间接寻址，操作数值 [填空3]

(4) 相对寻址，操作数值 [填空4]

作答

7.4 指令格式举例

一、设计指令格式时应考虑的各种因素

1. 指令系统的 **兼容性**　　（向上兼容）　　教材P319

2. 其他因素

操作类型	包括指令个数及操作的难易程度
数据类型	确定哪些数据类型可参与操作
指令格式	指令字长是否固定
	操作码位数、是否采用扩展操作码技术，
	地址码位数、地址个数、寻址方式类型
寻址方式	指令寻址、操作数寻址
寄存器个数	寄存器的多少直接影响指令的执行时间

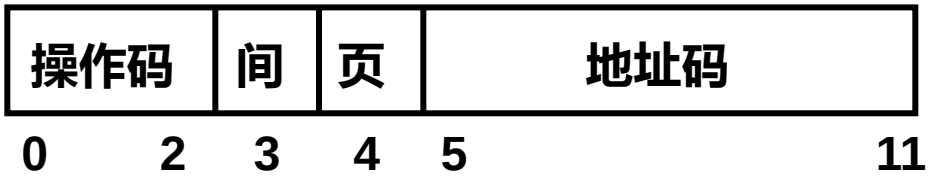
二、指令格式举例

1. PDP – 8

指令字长固定 12 位

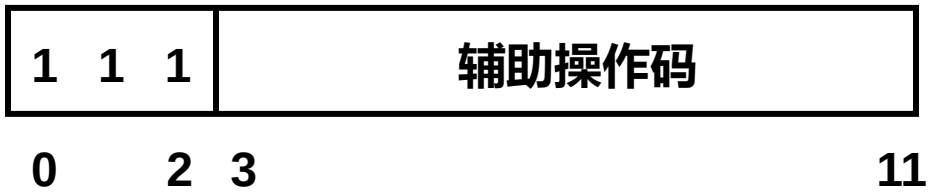
教材P320-321

访存类指令



I/O 类指令

寄存器类指令



采用扩展操作码技术

2. PDP – 11

指令字长有 16 位、32 位、48 位三种

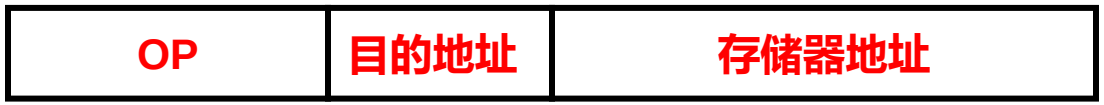
教材P321

零地址 (16 位)

扩展操作码技术

一地址 (16 位)

二地址 R – R (16 位)



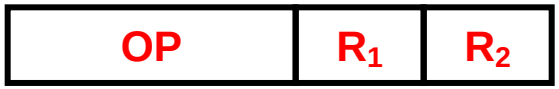
二地址 R – M (32 位)

二地址 M – M (48 位)

3. IBM 360

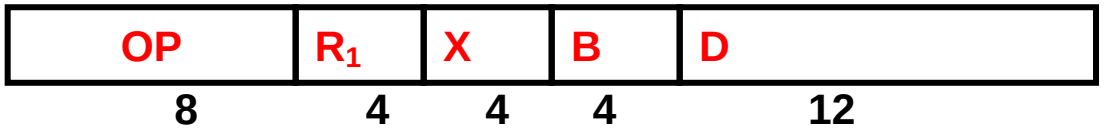
教材P321

RR 格式



二地址 R – R

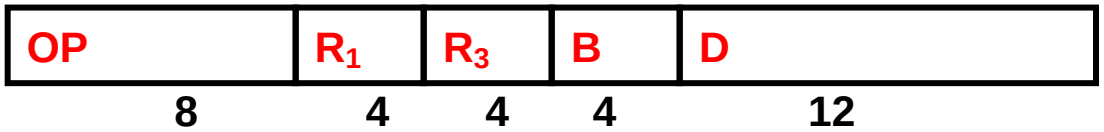
RX 格式



二地址 R – M

基址加变址寻址

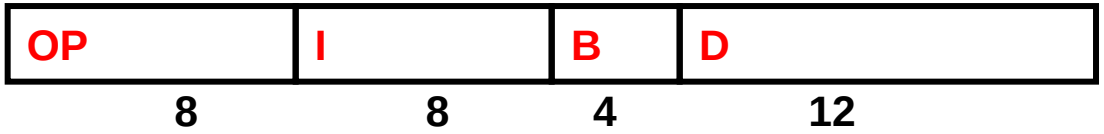
RS 格式



三地址 R – M

基址寻址

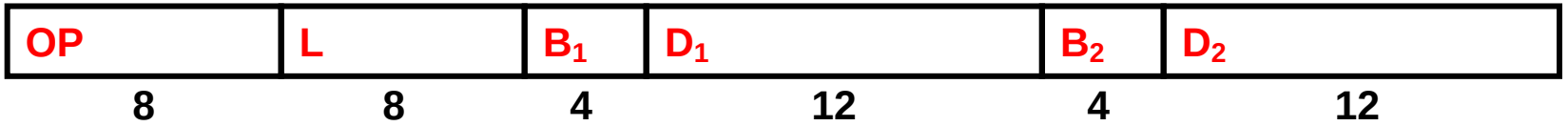
SI 格式



立即数 – M

基址寻址

SS 格式



二地址 M – M

基址寻址

(1) 指令字长　　1 ~ 6 个字节

INC　AX　　1 字节

MOV　WORD PTR[0204], 0138H　　6 字节

(2) 地址格式

零地址　　NOP　　1 字节

一地址　　CALL　　段间调用　　5 字节

CALL　　段内调用　　3 字节

二地址　　ADD AX，BX　　2 字节　　寄存器 – 寄存器

ADD AX，3048H　　3 字节　　寄存器 – 立即数

ADD AX，[3048H]　　4 字节　　寄存器 – 存储器

三、指令格式设计举例-自学* 7.4

教材P322-325

课后练习 7.4

7.19 CPU内有32个32位的通用寄存器，设计一种能容纳64种操作的指令系统。假设指令字长等于机器字长，试回答以下问题。

(1) 如果主存可直接或间接寻址，采用寄存器－存储器型指令，能直接寻址的最大存储空间是多少？画出指令格式并说明各字段的含义。

(2) 在满足(1)的前提下，如果采用通用寄存器作基址寄存器，则上述寄存器－存储器型指令的指令格式有何特点？画出指令格式并指出这类指令可访问多大的存储空间？

中国慕课大学
刘宏伟

课后练习 7.4

7.19 CPU 内有 32 个 32 位的通用寄存器，设计一种能容纳 64 种操作的指令系统。假设指令字长等于机器字长，试回答以下问题。

（1）如果主存可直接或间接寻址，采用寄存器－存储器型指令，能直接寻址的最大存储空间是多少？画出指令格式并说明各字段的含义。

（2）在满足（1）的前提下，如果采用通用寄存器作基址寄存器，则上述寄存器－存储器型指令的指令格式有何特点？画出指令格式并指出这类指令可访问多大的存储空间？

答：（1）指令字长32位，容纳64种操作，则操作码字段6位，寻址特征1位，寄存器-存储器型指令，寄存器地址段5位，存储器地址段=32-6-1-5=20位。

直接寻址的最大寻址空间范围是2^{20}=1M字

（寻址一般以字为单位）

（2）如采用基址寻址，则指令格式中应给出址寄存器号，以指定哪一一个通用寄存器用作基址寄存器。指令格式变为:

其中: B可省(B为基址寻址标志)，BRi为基址寄存器号。基址寻址时:

寻址的最大空间=2^{32}=4G字

其寻址范围仅与基址位数有关，与形式地址位数无关。

7.5 RISC 技术

一、RISC 的产生和发展

RISC（Reduced Instruction Set Computer）

CISC（Complex Instruction Set Computer）

20—80规律 —— RISC技术

- 典型程序中 80% 的语句仅仅使用处理机中 20% 的指令
- 执行频度高的简单指令，因复杂指令的存在，执行速度无法提高

? 能否用 20% 的简单指令组合不常用的80% 的指令功能

二、RISC 的主要特征

- 选用使用频度较高的一些 简单指令，
  复杂指令的功能由简单指令来组合
- 指令 长度固定、指令格式种类少、寻址方式少
- 只有 LOAD / STORE 指令访存，
  其余指令的操作都在寄存器之间进行。
- CPU 中有多个 通用 寄存器
- 采用 流水技术，大部分指令在一个时钟周期内完成
- 采用 组合逻辑 实现控制器
- 采用 优化 的 编译 程序

三、CISC 的主要特征

- 系统指令 **复杂庞大**，各种指令使用频度相差大
- 指令 **长度不固定、指令格式种类多、寻址方式多**
- **访存** 指令 **不受限制**
- CPU 中设有 **专用寄存器**
- 大多数指令需要 **多个时钟周期** 执行完毕
- 采用 **微程序** 控制器
- **难以** 用 **优化编译** 生成高效的目的代码

四、RISC和CISC 的比较 7.5

1. RISC更能 充分利用 VLSI 芯片的面积

2. RISC 更能 提高计算机运算速度

   指令数、指令格式、寻址方式少，
   通用 寄存器多，采用 组合逻辑 ，
   便于实现 指令流水

3. RISC 便于设计，可 降低成本，提高 可靠性

4. RISC 有利于编译程序代码优化

5. RISC 不易 实现 指令系统兼容

四、RISC和CISC 的比较 7.5

类别 / 对比项目	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	定长
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

单选题 1分

设置

[2009统考真题]下列关于RISC的说法中，错误的是(　　)。

A RISC 普遍采用微程序控制器

B RISC大多数指令在一个时钟周期内完成

C RISC的内部通用寄存器数量相对CISC多

D RISC的指令数、寻址方式和指令格式种类相对CISC少

提交

锲而舍之，朽木不折；
锲而不舍，金石可镂。

计 算 机 组 成 原 理

处理器分类

架构类型	架构名称	推出公司	推出时间	主要授权商
CISC	X86	Intel, AMD	1978	海光，兆芯
RISC	ARM	ARM	1985	苹果，三星，英伟达，高通，海思，TI等
	MIPS	MIPS	1981	龙芯，炬力等
	POWER	IBM	1990	IBM

X86	ARM	POWER	MIPS	DSP
Intel AMD	ARM 德州仪器 意法半导体 海思	IBM	MIPS 博通 CAVIUM	德州仪器

CISC　　RISC

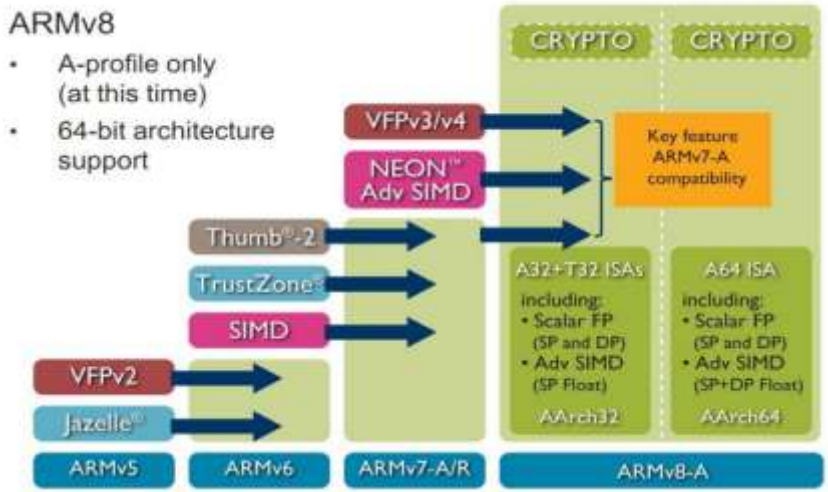
- CISC(Complex Instruction Set Computer)，复杂指令集。早期的CPU全部是CISC架构，它的设计目的是要用最少的机器语言指令来完成所需的计算任务。这种架构会增加CPU结构的复杂性和对CPU工艺的要求，但对于编译器的开发十分有利。
- RISC(Reduced Instruction Set Computer)，精简指令集。RISC架构要求软件来指定各个操作步骤。这种架构可以降低CPU的复杂性以及允许在同样的工艺水平下生产出功能更强大的CPU，但对于编译器的设计有更高的要求。

主流计算架构比较

	X86	ARM	POWER
指令集	• CISC, 复杂指令集	• RISC，精简指令集	• RISC，精简指令集
架构	• 重核架构，高性能高功耗	• 多核架构，均衡的性能功耗比	• 重核架构，高性能内核
生态	• 生态非常成熟，通用性强	• 生态正在快速发展与完备	• 聚焦大小型机和HPC
开放性	• 封闭架构，英特尔及AMD主导	• 开放平台，IP授权的商业模式	• 开放平台（2019.8），IBM主导

指令集：RISC vs CISC

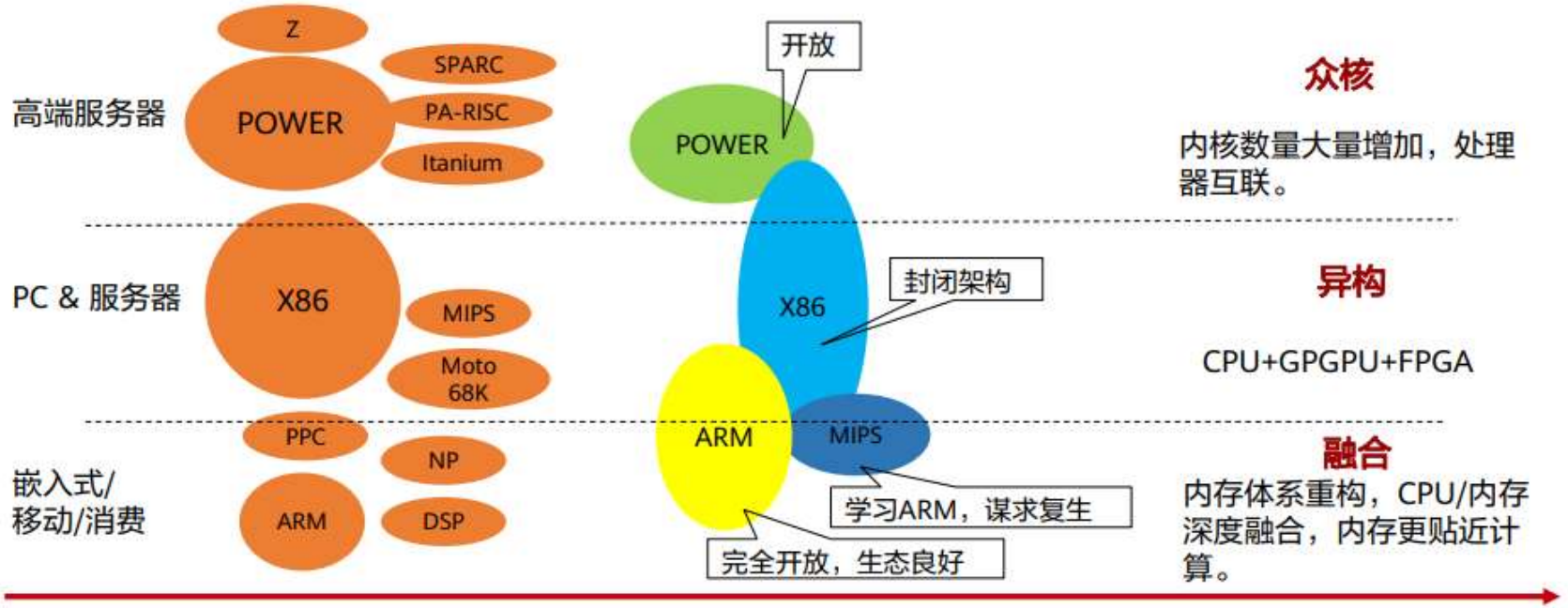
- ARM：使用精简指令集（RISC），大幅简化架构，仅保留所需要的指令，可以让整 个处理器更为简化，拥有小体积、高效能的特性；ARMv8架构支持64位操作，指令 32位，寄存器64位，寻址能力64位；指令集使用NEON扩展结构；
- X86：使用复杂指令集（CISC)，以增加处理器本身复杂度为代价，换取更高的性能； X86指令集从MMX，发展到了SSE，AVX；



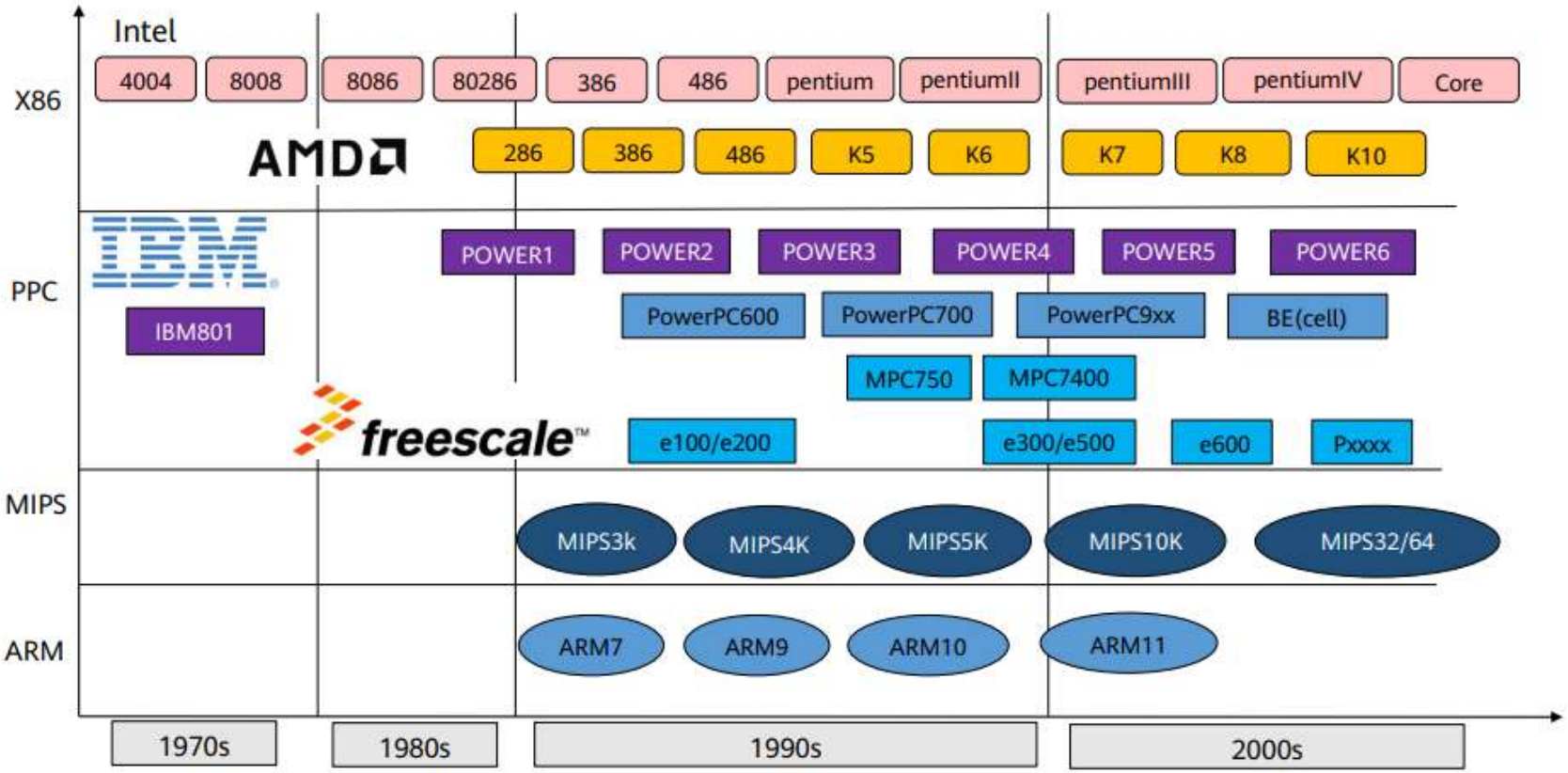
ARMv8架构的特点：

- 31个64位通用寄存器，原来架构只有15个通用寄存器；
- 新指令集支持64位运算，指令中的寄存器编码由4位扩充到5位；
- 新指令集仍然是32位，减少了条件执行指令，条件执行指令的4位编 码释放出来用于寄存器编码；
- 堆栈指针SP和程序指针PC都不再是通用寄存器了，同时推出了零值 寄存器（类似PowerPC的r0）；
- A64与A32的高级SIMD和FP相同；
- 高级SIMD与VFP共享浮点寄存器，支持128位宽的vector；
- 新增加解密指令。

处理器发展趋势

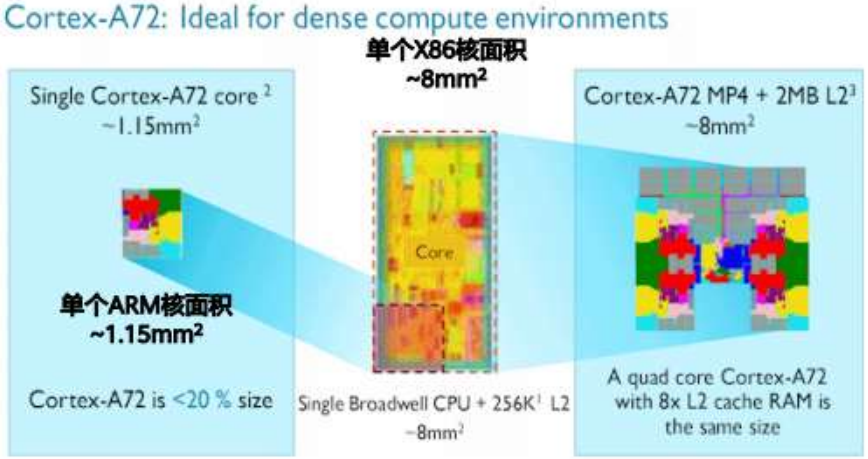

高端服务器
Z
POWER
SPARC
PA-RISC
Itanium
PC & 服务器
X86
MIPS
Moto 68K
嵌入式/
移动/消费
PPC
NP
ARM
DSP
开放
POWER
X86
封闭架构
ARM
MIPS
学习ARM，谋求复生
完全开放，生态良好
众核
内核数量大量增加，处理器互联。
异构
CPU+GPGPU+FPGA
融合
内存体系重构，CPU/内存深度融合，内存更贴近计算。
过去：架构众多，百花齐放
现在：生态成熟，架构垄断
未来：摩尔定律失效，寻求多方向突破

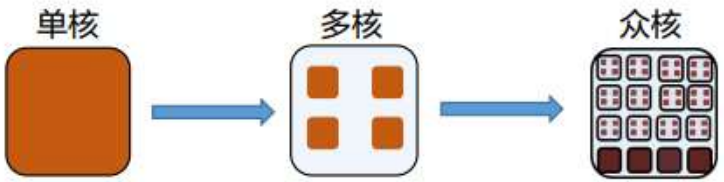
主流CPU发展路径


Intel
X86
4004
8008
8086
80286
386
486
pentium
pentiumII
pentiumIII
pentiumIV
Core
AMD
286
386
486
K5
K6
K7
K8
K10
IBM
PPC
POWER1
POWER2
POWER3
POWER4
POWER5
POWER6
IBM801
PowerPC600
PowerPC700
PowerPC9xx
BE(cell)
MPC750
MPC7400
freescale
e100/e200
e300/e500
e600
Pxxxx
MIPS
MIPS3k
MIPS4K
MIPS5K
MIPS10K
MIPS32/64
ARM
ARM7
ARM9
ARM10
ARM11
1970s
1980s
1990s
2000s

ARM提供更多计算核心

- 工艺、主频遇到瓶颈后，开始通过增加核数的方式来提升性能；
- 芯片的物理尺寸有限制，不能无限制的增加；
- ARM的众核横向扩展空间优势明显。





ARM处理器体系结构

- ARM内核工作模式:
  - 用户模式（user）：正常程序执行模式；
  - 快速中断模式（FIQ）：高优先级的中断产生会进入该种模式，用于高速通道传输；
  - 外部中断模式（IRQ）：低优先级中断产生会进入该模式，用于普通的中断处理；
  - 特权模式（Supervisor）：复位和软中断指令会进入该模式；
  - 数据访问中止模式（Abort）：当存储异常时会进入该模式；
  - 未定义指令中止模式（Undefined）：执行未定义指令会进入该模式；
  - 系统模式（System）：用于运行特权级操作系统任务；
  - 监控模式（Monitor）：可以在安全模式和非安全模式之间切换；

ARM处理器体系结构

- ARM体系结构的指令集（Instruction Set）
  - ARM指令集
  - Thumb指令集
  - Thumb-2指令集

ARM处理器体系结构

- ARM的微体系结构（Micro-architecture）
  - ARM处理器内核（Processor Core）
  - ARM处理器（Processor）
  - 基于ARM架构处理器的片上系统（SoC）

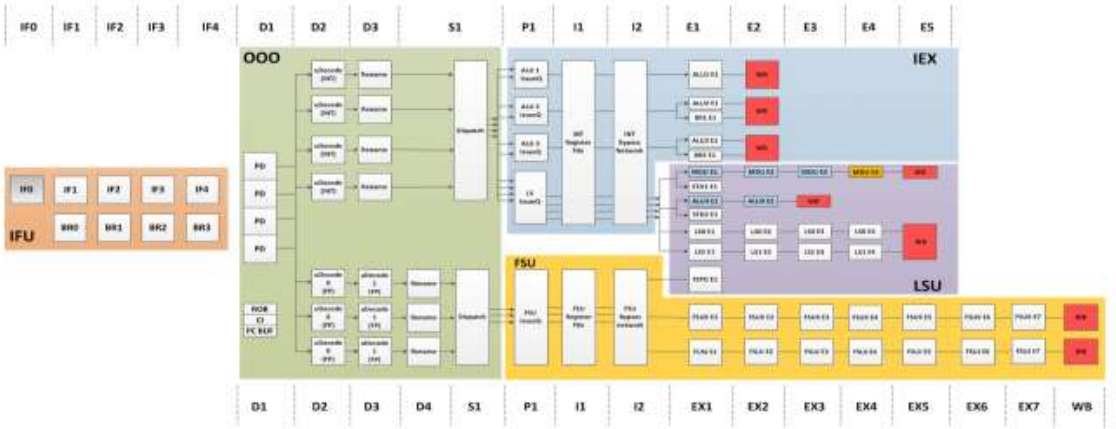
ARM公司授权体系

- ARM目前在全球拥有大约1000个授权合作商、320家伙伴，但是购买架构授权的厂家不超过20家，中国有华为、飞腾获得了架构授权。

ARM处理器体系结构

- ARM流水线的执行顺序：
  - 取指令（Fetch）：从存储器读取指令；
  - 译码（Decode）：译码以鉴别它是属于哪一条指令；
  - 执行（Execute）：将操作数进行组合以得到结果或存储器地址；
  - 缓冲/数据（Buffer/data）：如果需要，则访问存储器以存储数据；
  - 回写：（Write-back）：将结果写回到寄存器组中；

基于ARMv8的鲲鹏流水线技术



Taishan coreV110 Pipeline Architecture

- Branch预测和取指流水线解耦设计，取指流水线每拍最多可提供32Bytes指令供译码，分支预测流水线可以不受取指流水停顿影响，超前进行预测处理；
- 定浮点流水线分开设计，解除定浮点相互反压，每拍可为后端执行部件提供4条整型微指令及3条浮点微指令；
- 整型运算单元支持每拍4条ALU运算（含2条跳转）及1条乘除运算；
- 浮点及SIMD运算单元支持每拍2条ARM Neon 128bits 浮点及SIMD运算；
- 访存单元支持每拍2条读或写访存操作，读操作最快4拍完成，每拍访存带宽为2x128bits读及1x128bits写；

ARM处理器体系结构

- ARM处理器的分类：
  - ARM经典处理器（Classic Processors）；
  - ARM Cortex应用处理器；
    - 面向复杂操作系统和用户应用的Cortex-A（Applications，应用）系列
    - 针对实时处理和控制应用的Cortex-R（Real-time，实时）系列
    - 针对微控制器与低功耗应用优化的Cortex-M（Microcontroller）系列
  - ARM Cortex嵌入式处理器；
  - ARM专业处理器

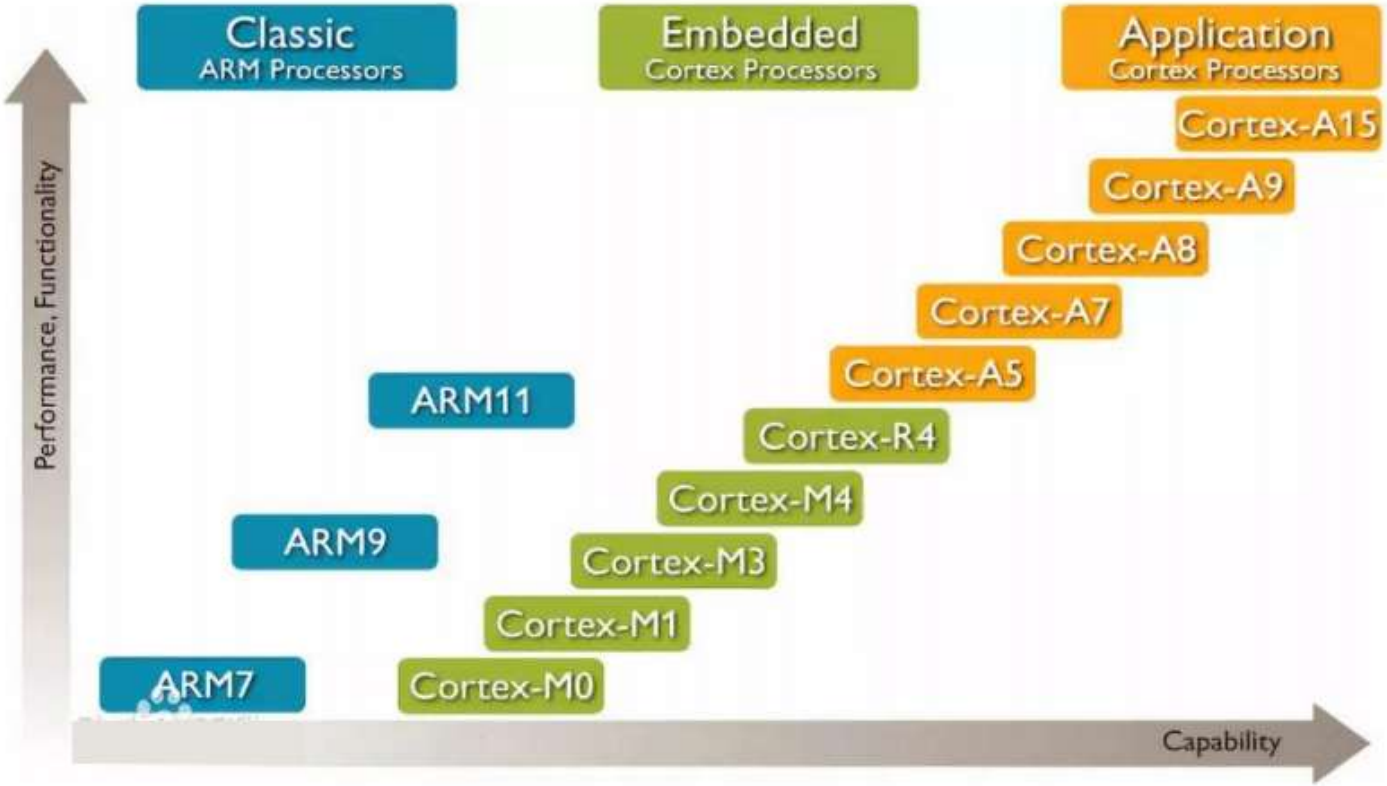
ARM 处理器系列命名规则

- 命名格式：ARM {x} {y} {z} {T} {D} {M} {I}
  - x：处理器系列，是共享相同硬件特性的一组处理器，如：ARM7TDMI、ARM740T 都属于 ARM7 系列
  - y：存储管理 / 保护单元
  - z：Cache
  - T：Thumb，Thumb16 位译码器
  - D：Debug，JTAG 调试器
  - M：Multipler，快速乘法器
  - I：Embedded ICE Logic，嵌入式跟踪宏单元

ARM架构发展史

架构	处理器家族
ARMv1	ARM1
ARMv2	ARM2、ARM3
ARMv3	ARM6、ARM7
ARMv4	StrongARM、ARM7TDMI、ARM9TDMI
ARMv5	ARM7EJ、ARM9E、ARM10E、XScale
ARMv6	ARM11、ARM Cortex-M
ARMv7	ARM Cortex-A、ARM Cortex-M、ARM Cortex-R
ARMv8	Cortex-A50[9]

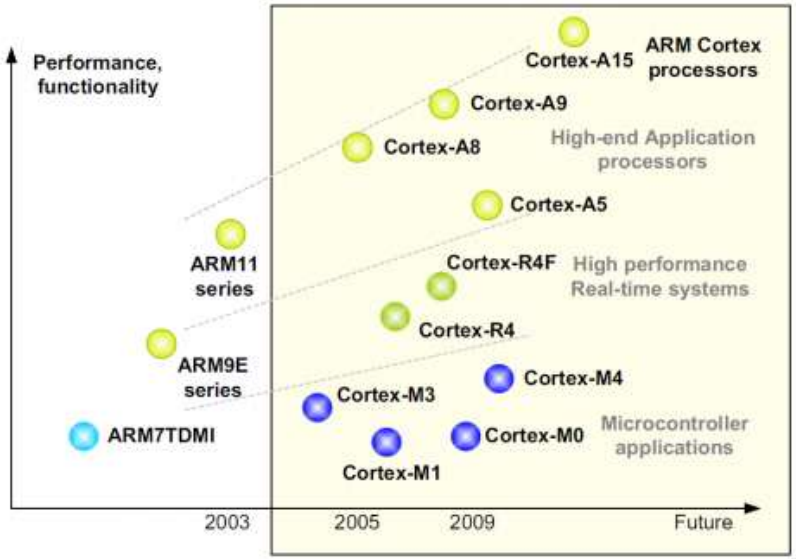
ARM架构发展史

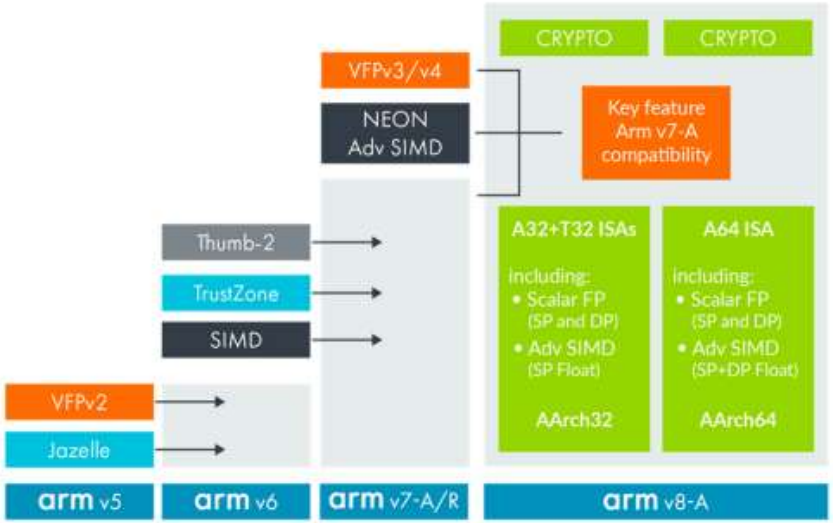


ARM架构发展史

- 从ARM v7开始，CPU命名为Cortex，并划分为A、R、M三大系列，分别为不同的市场提供服务；
  - A (Application)系列：应用型处理器，面向具有复杂软件操作系统的面向用户的应用，为手机、平板、AP等终端设备提供全方位的解决方案；
  - R (Real-Time)系列：实时高性能处理器，为要求可靠性、高可用性、容错功能、可维护性和实时响应的嵌入式系统提供高性能计算解决方案；
  - M (Microcontroller)系列：高能效、易于使用的处理器，主要用于通用低端，工业，消费电子领域微控制器。

ARM架构发展史


Performance, functionality
Cortex-A15
ARM Cortex processors
Cortex-A9
Cortex-A8
High-end Application processors
Cortex-A5
ARM11 series
Cortex-R4F
High performance Real-time systems
Cortex-R4
ARM9E series
Cortex-M4
Cortex-M3
ARM7TDMI
Cortex-M0
Microcontroller applications
Cortex-M1
2003
2005
2009
Future


CRYPTO
CRYPTO
VFPv3/v4
NEON Adv SIMD
Key feature Arm v7-A compatibility
Thumb-2
TrustZone
SIMD
A32+T32 ISAs
including:
• Scalar FP (SP and DP)
• Adv SIMD (SP Float)
AArch32
A64 ISA
including:
• Scalar FP (SP and DP)
• Adv SIMD (SP+DP Float)
AArch64
VFPv2
Jazelle
arm v5
arm v6
arm v7-A/R
arm v8-A

ARM服务器处理器的优势

- 低功耗一直以来都是ARM架构芯片最大的优势；
- ARM架构的芯片在成本、集成度方面也有较大的优势；
- 端、边、云全场景同构互联与协同；
- 更高的并发处理效率；
- 多元化的市场供应

华为鲲鹏处理器

1991 第一颗传输网络的ASIC芯片

2005 第一颗基于ARM的无线基站芯片

2009 第一颗基于ARM的移动端处理器

2014 第一颗基于ARM的64位处理器

2016 业界第一颗支持多路ARM 处理器

2019 业界第一颗7nm数据中心处理器

2021

2023

服务器内部视图