指令系统

大连理工大学 赖晓晨

指令系统概述

大连理工大学 赖晓晨

David A. Patterson

加州大学伯克利分校的计算机科学教授

计算机科学界的先驱人物

主要贡献：

- 精简指令集
- RAID
- 计算机簇

一、指令系统

- 物理的计算机只能够执行机器语言程序，组成程序的每一条语句称作一条**机器指令**，一种计算机能够执行的机器指令的集合就是这种计算机的**指令系统**。
- 计算机**设计者**的重要工作之一为如何设计指令系统，计算机的**使用者**根据每一条指令的功能，来操纵计算机。

二、机器指令格式

指令的一般格式包括两部分：

操作码字段	地址码字段

1. **操作码**：位数反映机器指令数目，内容反映机器做什么操作。

   操作码类别：

   - 长度固定：如 RISC 机器，指令规整，译码简单
   - 长度可变：操作码分散在指令字的不同字段中，控制器设计复杂。

二、机器指令格式

2. **地址码**：用来指定该指令操作数的地址、结果的地址，以及（可能有的）下一条指令的地址。

例如指令：　MOV　AX, [40]

地址“40”指明了要操作的**源操作数**的地址，AX指明了**目的操作数**的地址。

三、不同地址数目的指令

1. 四地址指令

设指令字长为32位，操作码固定为8位。

8	6	6	6	6
OP	A_1	A_2	A_3	A_4

执行阶段4 次访存

寻址范围 $2^6 = 64$

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

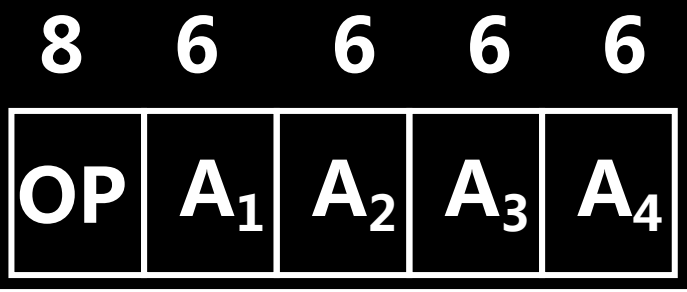
A_4 下一条指令地址

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

三、不同地址数目的指令

1. 四地址指令

设指令字长为32位，操作码固定为8位。



A_1 第一操作数地址

A_2 第二操作数地址

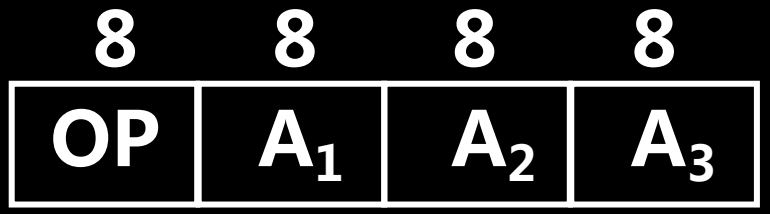
程序中大部分指令是顺序执行的，可以用程序计数器PC来指明下一条指令地址，因此可以去掉A4字段。

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

三、不同地址数目的指令

2. 三地址指令

设指令字长仍为32位，操作码和地址码均为8位



4 次访存

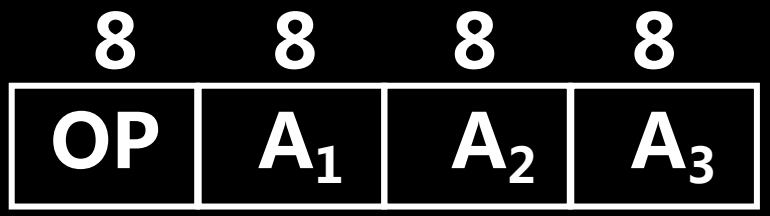
(A_1) OP (A_2) ⟶ A_3

寻址范围 $2^8 = 256$

三、不同地址数目的指令

2. 三地址指令

设指令字长仍为32位，操作码和地址码均为8位



4 次访存

(A_1) OP (A_2) ⟶ A_3

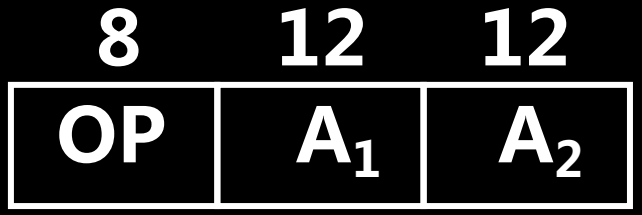
寻址范围 $2^8 = 256$

如果A_3用A_1或者A_2代替，那么A_3可以省略。

三、不同地址数目的指令

3. 二地址指令

设指令字长仍为32位，操作码8位，地址码12位



(A_1) OP (A_2) ⟶ A_1

或

(A_1) OP (A_2) ⟶ A_2

寻址范围 $2^{12} = 4$ K

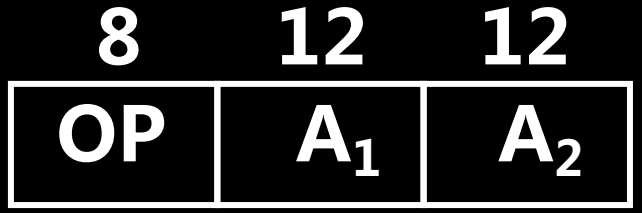
4 次访存

若结果存于 ACC，则3次访存。

三、不同地址数目的指令

3. 二地址指令

设指令字长仍为32位，操作码8位，地址码12位



(A_1) OP (A_2) ⟶ A_1

4 次访存

若结果存于 ACC，

或

如果一个操作数可以隐含为ACC，则指令中可以只给出一个地址码。

三、不同地址数目的指令

4. 一地址指令

设指令字长仍为32位，操作码8位，地址码24位

(ACC) OP (A_1) ⟶ ACC

2 次访存

寻址范围 2^{24} = 16 M

三、不同地址数目的指令

5. 零地址指令

指令系统中，有一种指令可以不设置地址字段，即零地址指令。如：NOP、HLT指令，不需要地址码，RET、IRET等指令，操作数的地址隐含在堆栈中。

指令字小结

1. 当用一些硬件资源代替指令字中的地址码字段时，可以：
   - 扩大指令寻址范围
   - 缩短指令字长
   - 减少访存次数
2. 如果指令的地址字段为寄存器：
   - 可以缩短指令字长
   - 指令执行阶段不访存

四、操作码扩展技术

- 通过操作码扩展（对应的地址码长度收缩）技术，能够有效的缩短指令的平均长度。
- 例如，某计算机有5条指令，两种设计方法。

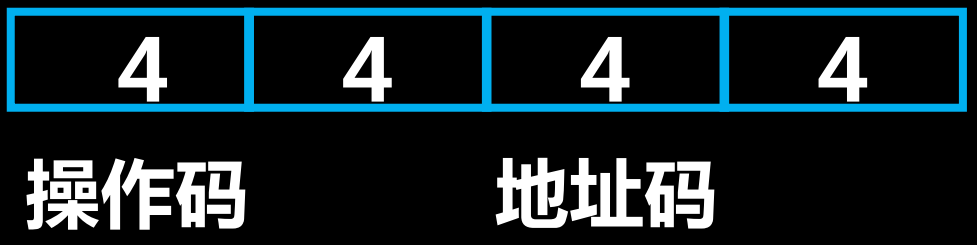
  等长操作码，3位；变长操作码，平均2.4位。

000
001
010
011
100

00
01
10
110
111

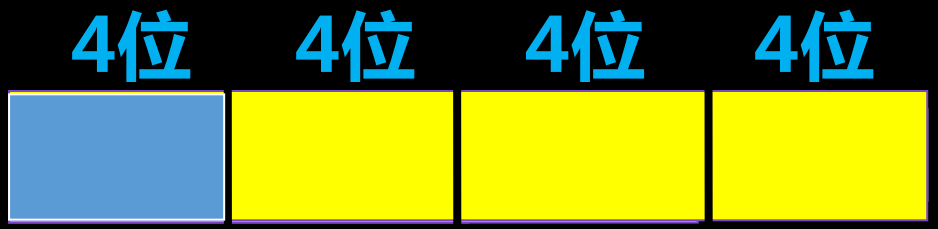
例题1

假设一台计算机指令字长16位，操作码与地址码都为4位，请列出几种可能的操作码设计方法。



固定操作码

固定4位操作码：



0000 XXXX YYYY ZZZZ

⋮

1111 XXXX YYYY ZZZZ

} 16条三地址指令

扩展操作码1

设计15条三地址指令

0000 XXXX YYYY ZZZZ
⋮
1110 XXXX YYYY ZZZZ

} 15条三地址指令

留下一个码点，用来扩展二地址指令。

扩展操作码1

二地址指令留下一个码点，用来扩展一地址指令。

扩展操作码1

一地址指令留下一个码点，用来扩展零地址指令。

扩展操作码1

16条零地址指令。

扩展操作码2

另一种扩展方法

0000 XXXX YYYY ZZZZ
1110 XXXX YYYY ZZZZ
} 15条三地址指令

1111 0000 XXXX YYYY
1111 1101 XXXX YYYY
} 14条二地址指令

1111 1110 0000 XXXX
1111 1110 1111 XXXX
} 16条一地址指令

1111 1111 0000 XXXX
1111 1111 1110 XXXX
} 15条一地址指令

} 31

1111 1111 1111 0000
1111 1111 1111 1111
} 16条零地址指令

指令字长

- 指令字长取决于操作码长度、地址码长度和地址码个数。指令按照字长是否可变分为两种：
  - 指令字长**固定**：指令字长=存储字长
  - 指令字长**可变**：按字节的整数倍变化
- 指令字长可变，导致控制电路复杂，多字长指令需要多次访存，应尽量把常用指令设计为单字长或短字长指令。

小问题

五、RISC技术

1. RISC的产生和发展

- RISC（ Reduced Instruction Set Computer）
- CISC（ Complex Instruction Set Computer）
- VLSI技术的发展
- 典型程序中 80% 的语句仅仅使用处理机中 20% 的指令
- 复杂指令集计算机设计复杂、成本高、维护困难
- 能否用 20% 的简单指令组合不常用的80% 的指令功能

2. RISC的主要特征

- 选用使用频度高的一些简单指令，复杂指令用简单指令组合。
- 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。
- CPU中有多个通用寄存器。
- 采用流水技术,一个时钟周期完成一条指令。
- 采用组合逻辑实现控制器。
- 采用优化的编译程序。

3. CISC的主要特征

- 指令系统庞大复杂，各种指令使用频度差别大。
- 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。

4. RISC和CISC的比较

- RISC更能充分利用VLSI芯片的面积
- RISC更能提高计算机运算速度
  - 指令数目、指令格式、寻址方式少
  - 通用寄存器多，采用组合逻辑
  - 便于流水线操作
- RISC便于设计、成本低、可靠性高
- RISC有利于编译程序代码优化
- RISC不易实现指令系统兼容

5. RISC计算机举例

- DLX：教学用，多元未饱和型指令集结构。
- PowerPC：IBM/Apple/Motorola，可伸缩性好、方便灵活。
- MIPS：卖的最好的RISC CPU。
- SPARC：SUN Microsystems，可扩充处理器架构。
- 龙芯：中科院计算技术研究所。
- ARM：ARM（Advanced RISC Machine）公司。

扩展阅读： 指令格式设计准则

- 短指令优于长指令，但指令长度太小则会增加译码和重叠执行的难度；
- 指令中必须有足够的空间来表示所有的操作类型，并为将来指令集扩展留有余量；
- 合理选择地址字段中位的数量，太短则降低寻址粒度，太长则增加指令长度。

操作类型

大连理工大学 赖晓晨

John L. Hennessy

美国计算机科学家
MIPS科技公司创办人
斯坦福大学校长

一、操作数类型

- 地址：有时地址也需要计算，此时地址也是数据
- 数字：定点数、浮点数、十进制数。
- 字符：ASCII码。
- 逻辑数据：每一位都代表真（1）或假（0）的布尔类型数据，这种数字串即为逻辑数。

二、数据存储方式

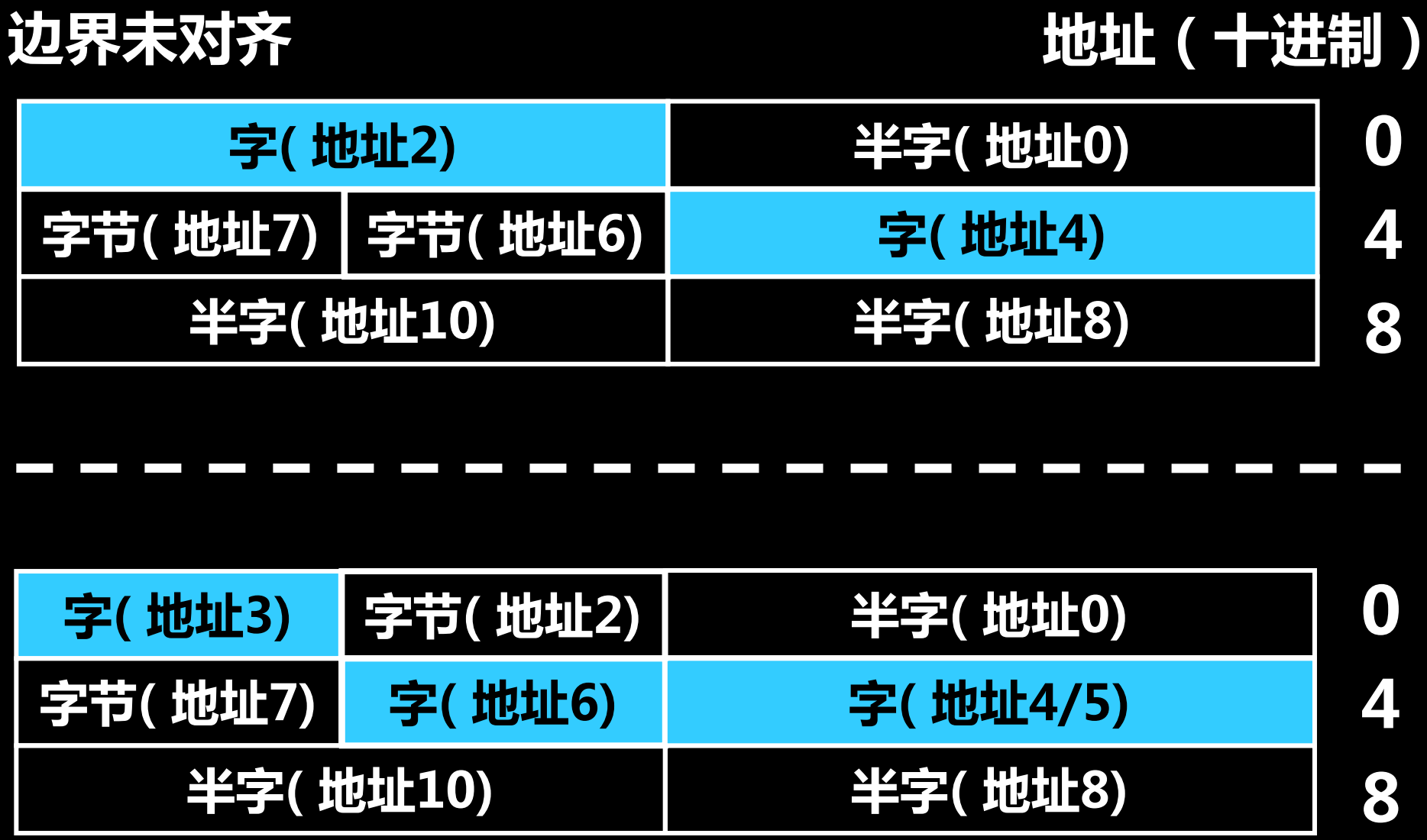
- 两种字节序：小端 vs 大端
- 对齐方式：

  为了便于硬件实现，同时提高机器运行速度，通常要求多字节数据在存储器中满足“边界对齐”的要求。即，字节数据可以任意存放；半字存放在偶数地址；字存放在末两位地址为0处；双字存放在末三位地址为0处。

边界对齐

边界对齐（32）位机				地址（十进制）
字（地址 0）				0
字（地址 4）				4
字节（地址11）	字节（地址10）	字节（地址 9）	字节（地址 8）	8
字节（地址15）	字节（地址14）	字节（地址13）	字节（地址12）	12
半字（地址18） ✓		半字（地址16） ✓		16
半字（地址22） ✓		半字（地址20） ✓		20
双字（地址24） ▲				24
双字				28
双字（地址32） ▲				32
双字				36

边界未对齐举例


边界未对齐
地址（十进制）
字(地址2)
半字(地址0)
0
字节(地址7)
字节(地址6)
字(地址4)
4
半字(地址10)
半字(地址8)
8
字(地址3)
字节(地址2)
半字(地址0)
0
字节(地址7)
字(地址6)
字(地址4/5)
4
半字(地址10)
半字(地址8)
8

边界未对齐的恶果（ARM）

```
int readint(__packed int *data)
{    return *data;   }
```

```
readint
   BIC     r3,r0,#3          ; r3 = data & 0xFFFFFFFC
   AND     r0,r0,#3          ; r0 = data & 0x00000003
   MOV     r0,r0,LSL #3  ; r0 = bit offset of data word
   LDMIA   r3,{r3,r12}    ; r3, r12 = 8 bytes read from r3
   MOV     r3,r3,LSR r0  ; These three instructions
   RSB     r0,r0,#0x20    ; shift the 64 bit value r12.r3
   ORR     r0,r3,r12,LSL r0        ; right by r0 bits
   MOV     pc,r14            ; return r0
```

三、操作类型

1. 数据传送类指令

源	目的		
寄存器	寄存器	MOV AX, BX	
寄存器	存储器	MOV [20], AX	STORE指令
存储器	寄存器	MOV AX, [20]	LOAD指令
存储器	存储器	MOV [20], [30]	

堆栈操作： PUSH AX　　POP AX

清零、置1：MOV AX, 0　　MOV AX, 1

2. 运算类指令

- 算术运算：加、减、乘、除、求补、浮点、十进制运算

  ADD AX, 20　　DIV AX, 3

- 逻辑运算：与、或、非、异或

  AND AX, 30　　XOR AX, 30

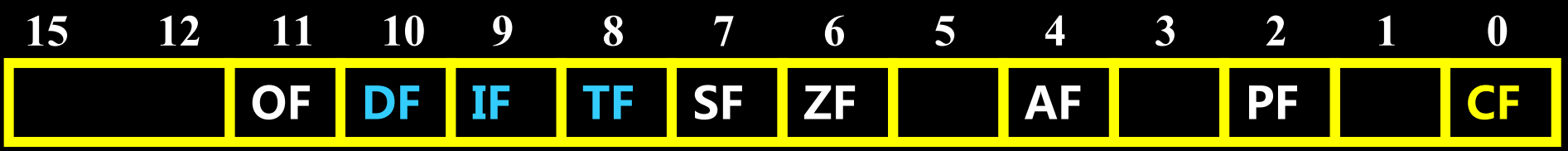
- 其他：位测试、位清除、位求反

3. 移位指令

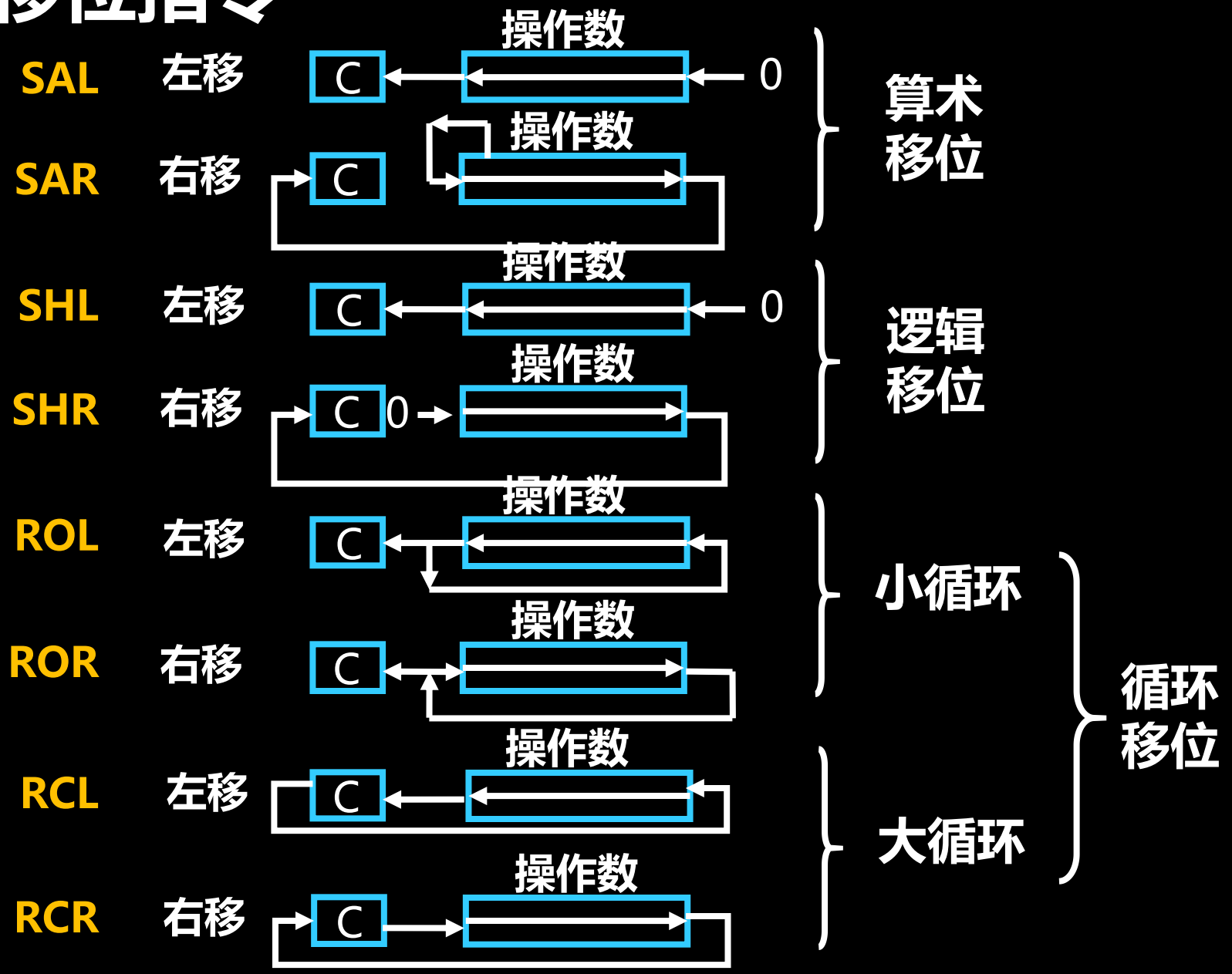
一般来说，有8种移位指令：

算术左移、算术右移、逻辑左移、逻辑右移、小循环左移、小循环右移、大循环左移、大循环右移

无论哪一种移位，移出位都保存在进位标志C中。



移位指令


SAL
左移
操作数
C
0
SAR
右移
操作数
C
算术
移位
SHL
左移
操作数
C
0
SHR
右移
操作数
C
0
逻辑
移位
ROL
左移
操作数
C
ROR
右移
操作数
C
小循环
RCL
左移
操作数
C
RCR
右移
操作数
C
大循环
循环
移位

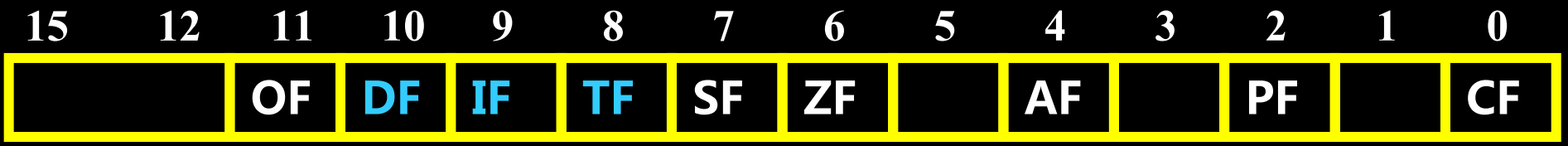
4. 转移指令

(1) 无条件转移指令

直接跳转到某处，不取决于任何条件。类似C中 的goto语句。例如：JMP LOOP

(2) 条件转移指令

根据机器当前的程序状态字中的某位来决定是否执行转移。例如： JZ LOOP



(3) 调用与返回指令

类比C程序中的函数调用，以及函数返回。
例如：CALL PRO1、RET

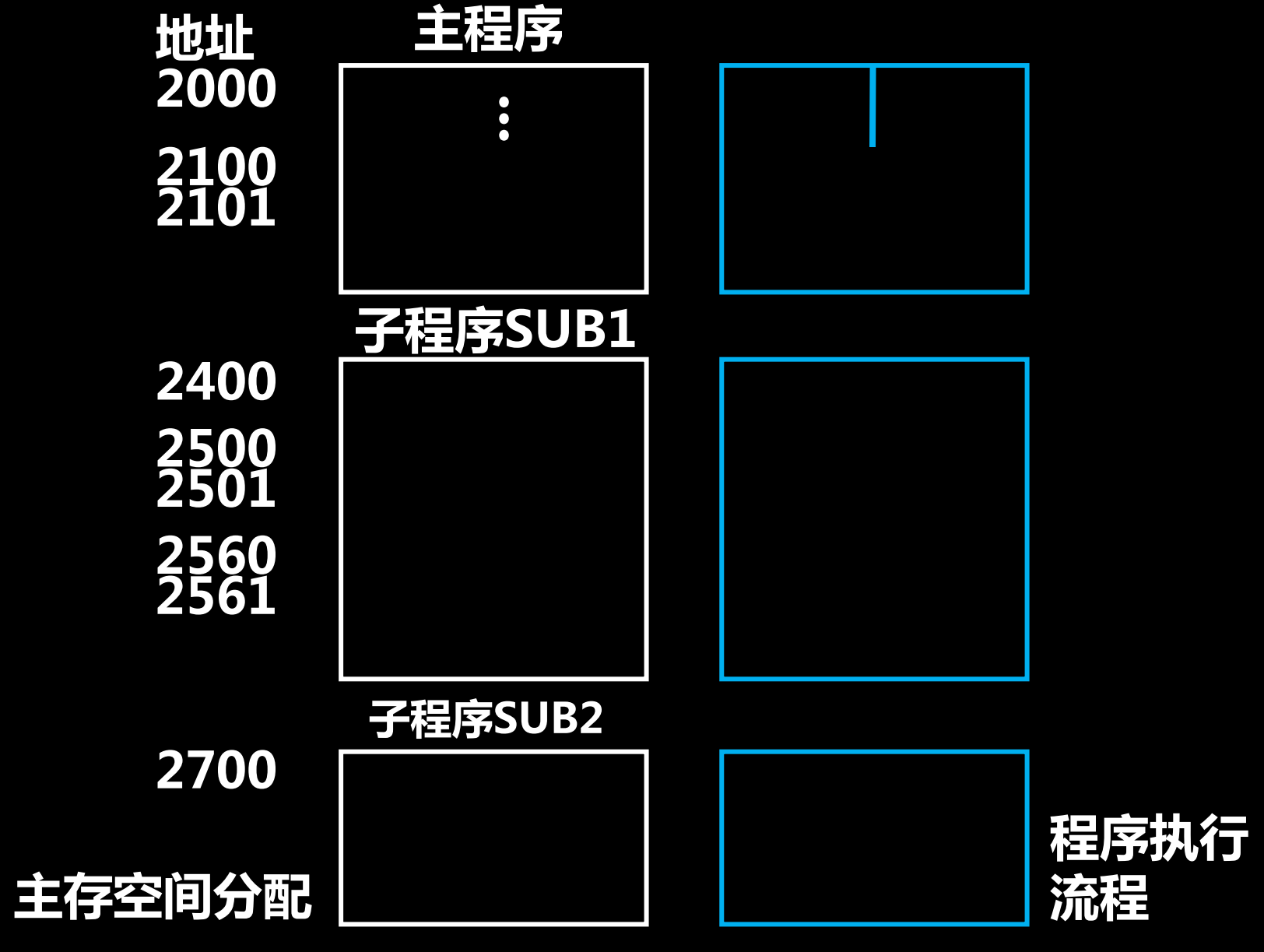
子程序调用需要注意以下几点：

- 子程序可以在多处被调用
- 子程序调用可以嵌套
- CALL与RET指令配对使用
- 要妥善保存子程序的返回地址
  专用寄存器、堆栈

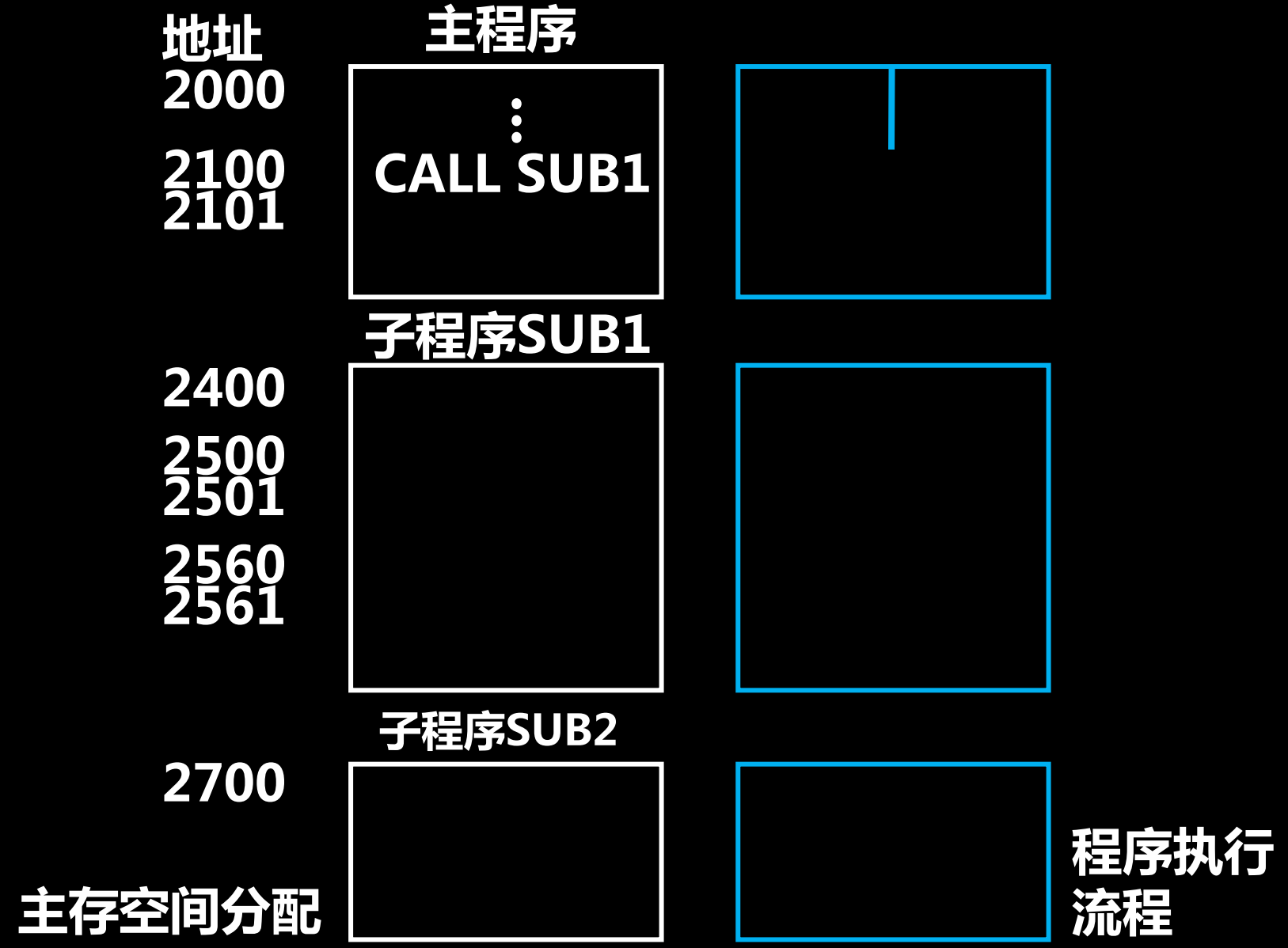
子程序调用程序


地址
主程序
2000
2100
2101
子程序SUB1
2400
2500
2501
2560
2561
子程序SUB2
2700
主存空间分配
程序执行
流程

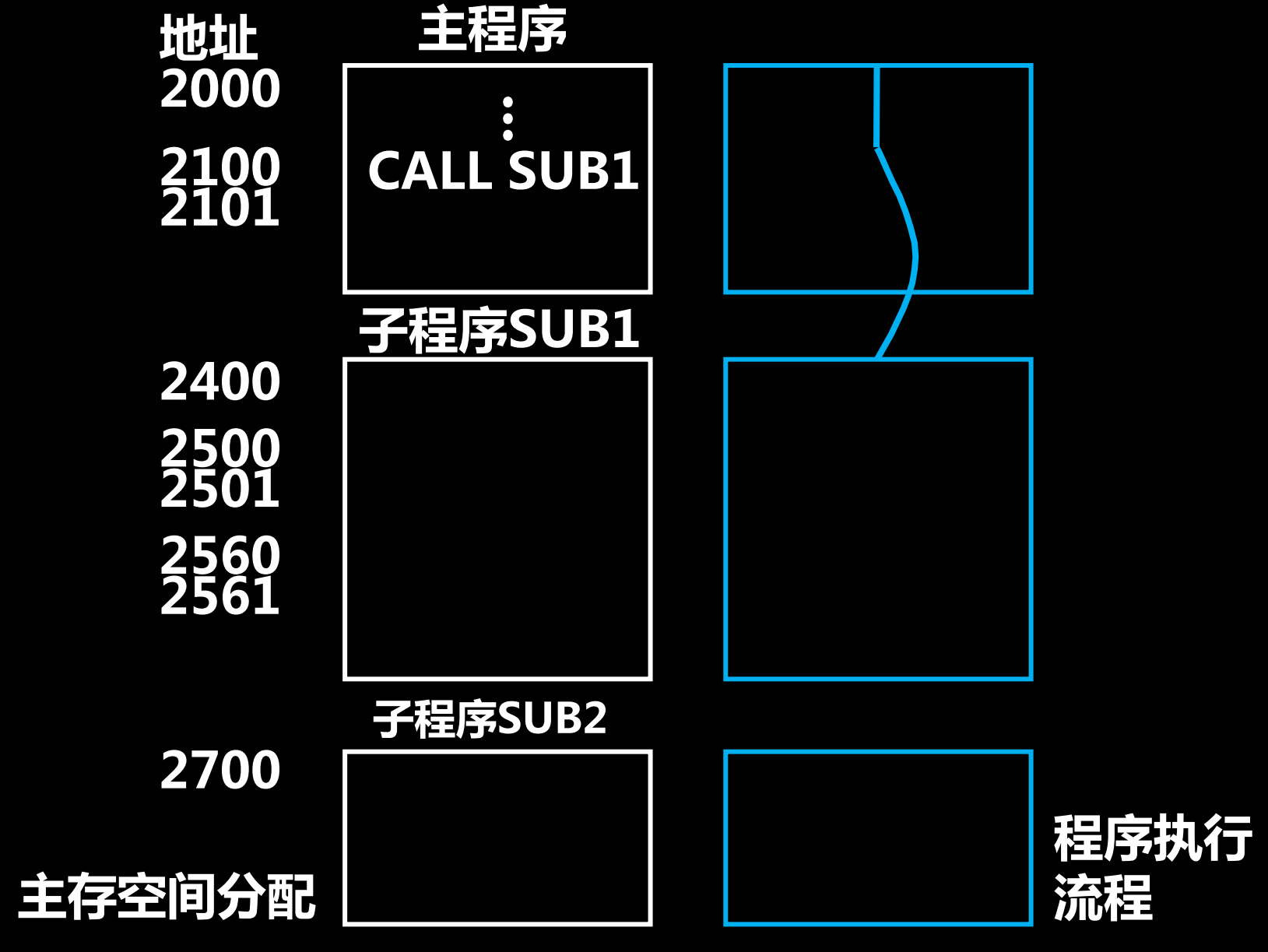
子程序调用程序


地址
主程序
2000
2100
2101
子程序SUB1
2400
2500
2501
2560
2561
子程序SUB2
2700
主存空间分配
程序执行
流程

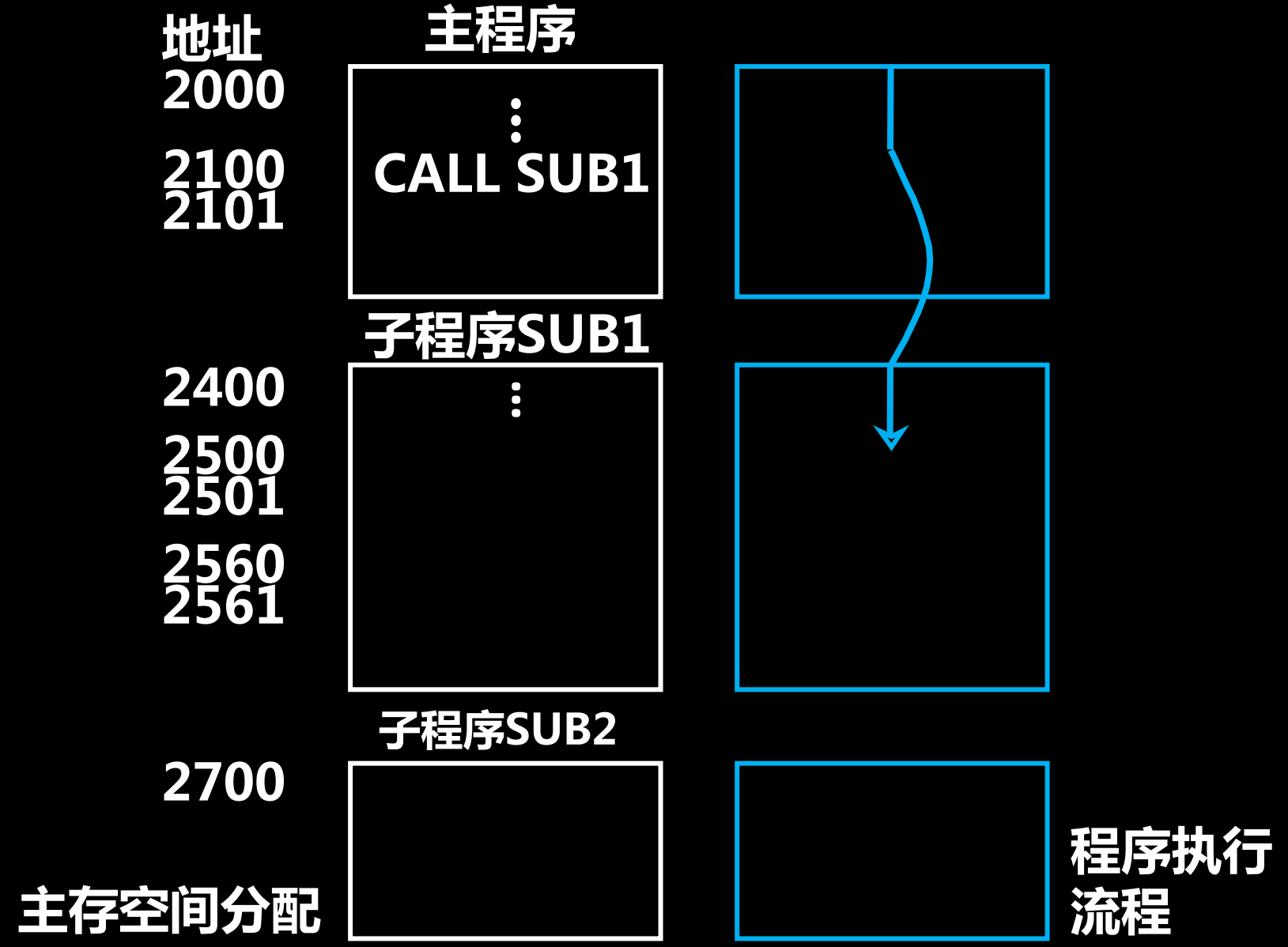
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
子程序SUB2
2700
主存空间分配
程序执行
流程

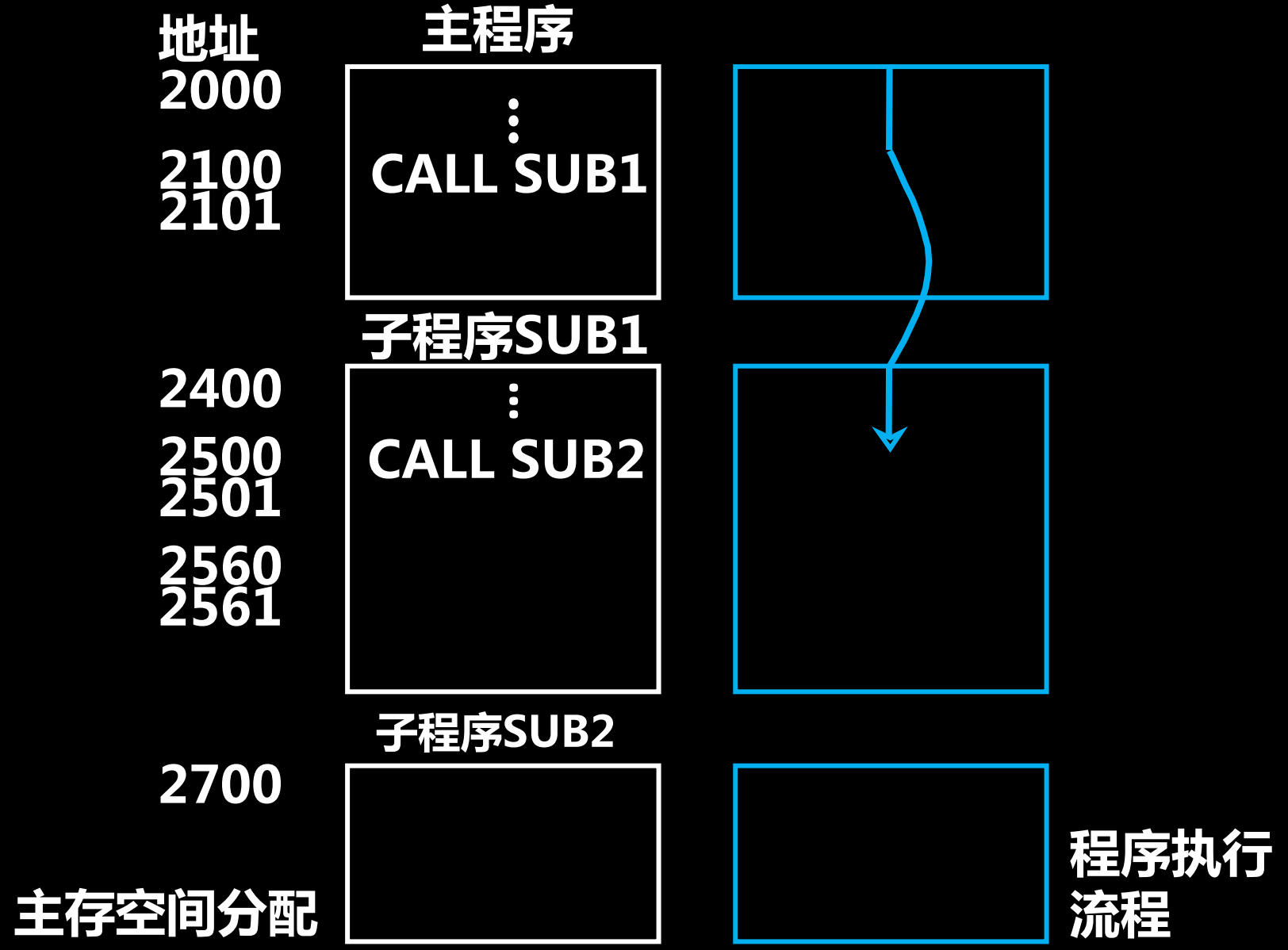
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
子程序SUB2
2700
主存空间分配
程序执行
流程

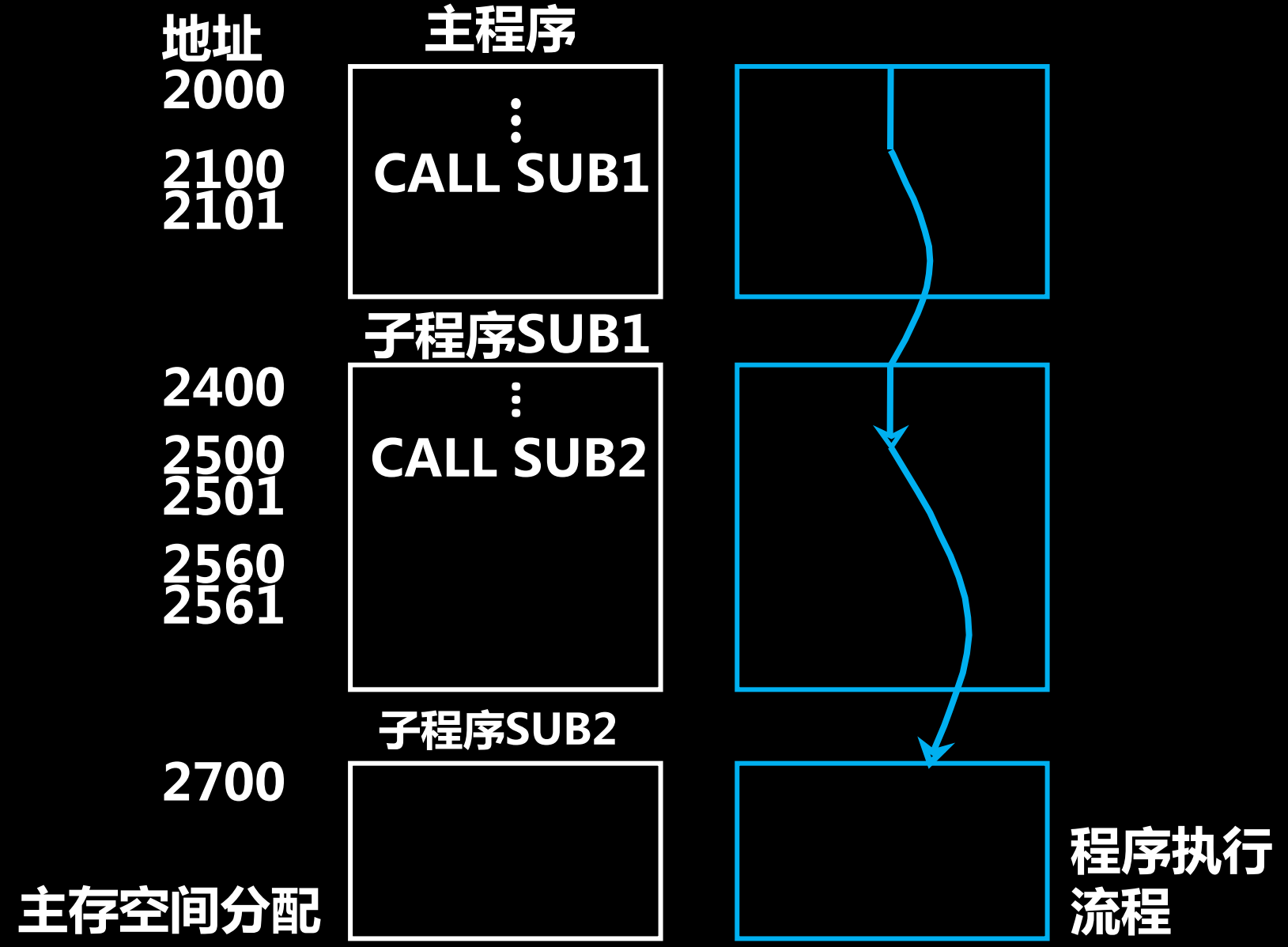
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
子程序SUB2
2700
主存空间分配
程序执行
流程

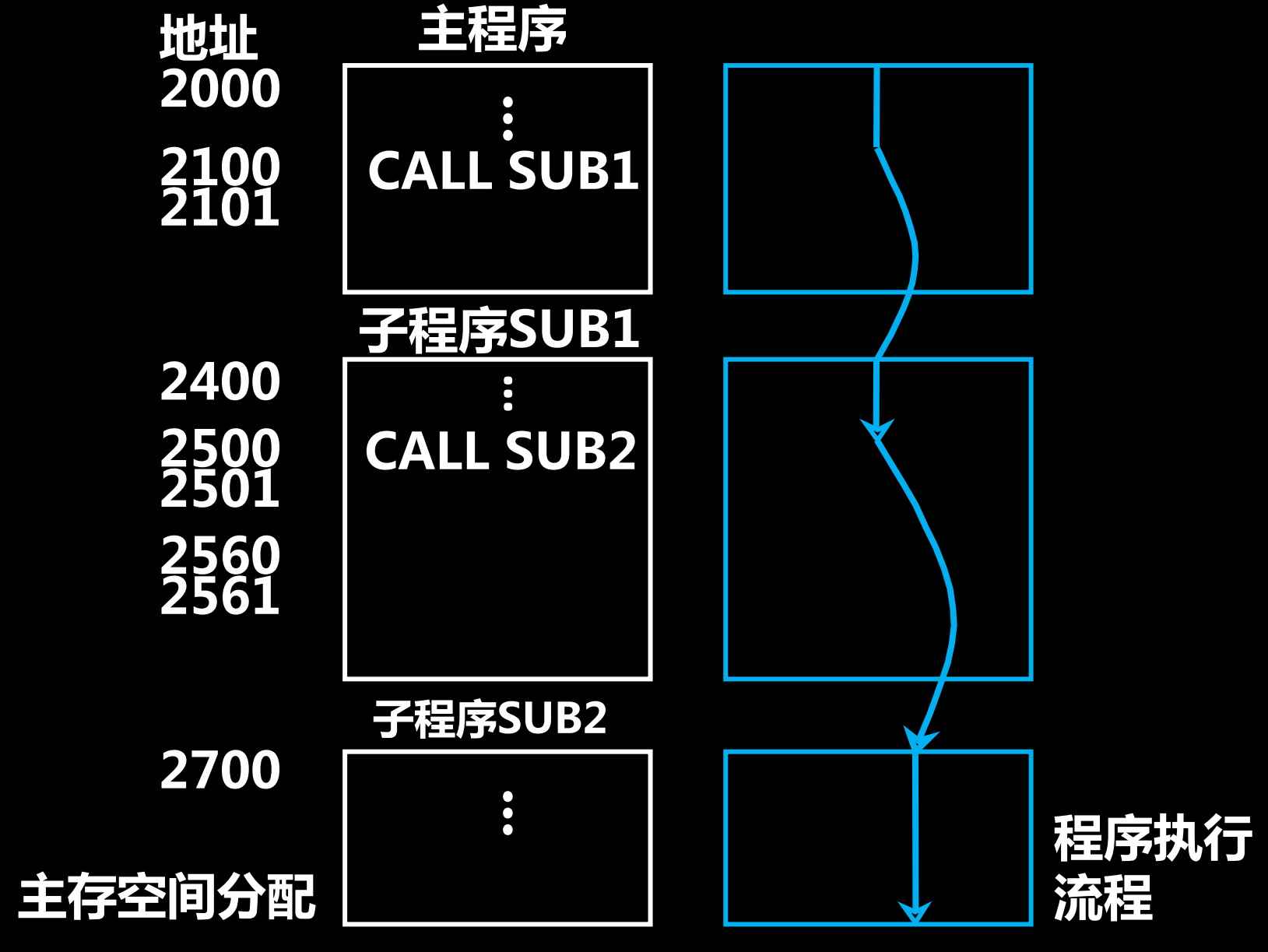
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
子程序SUB2
2700
主存空间分配
程序执行
流程

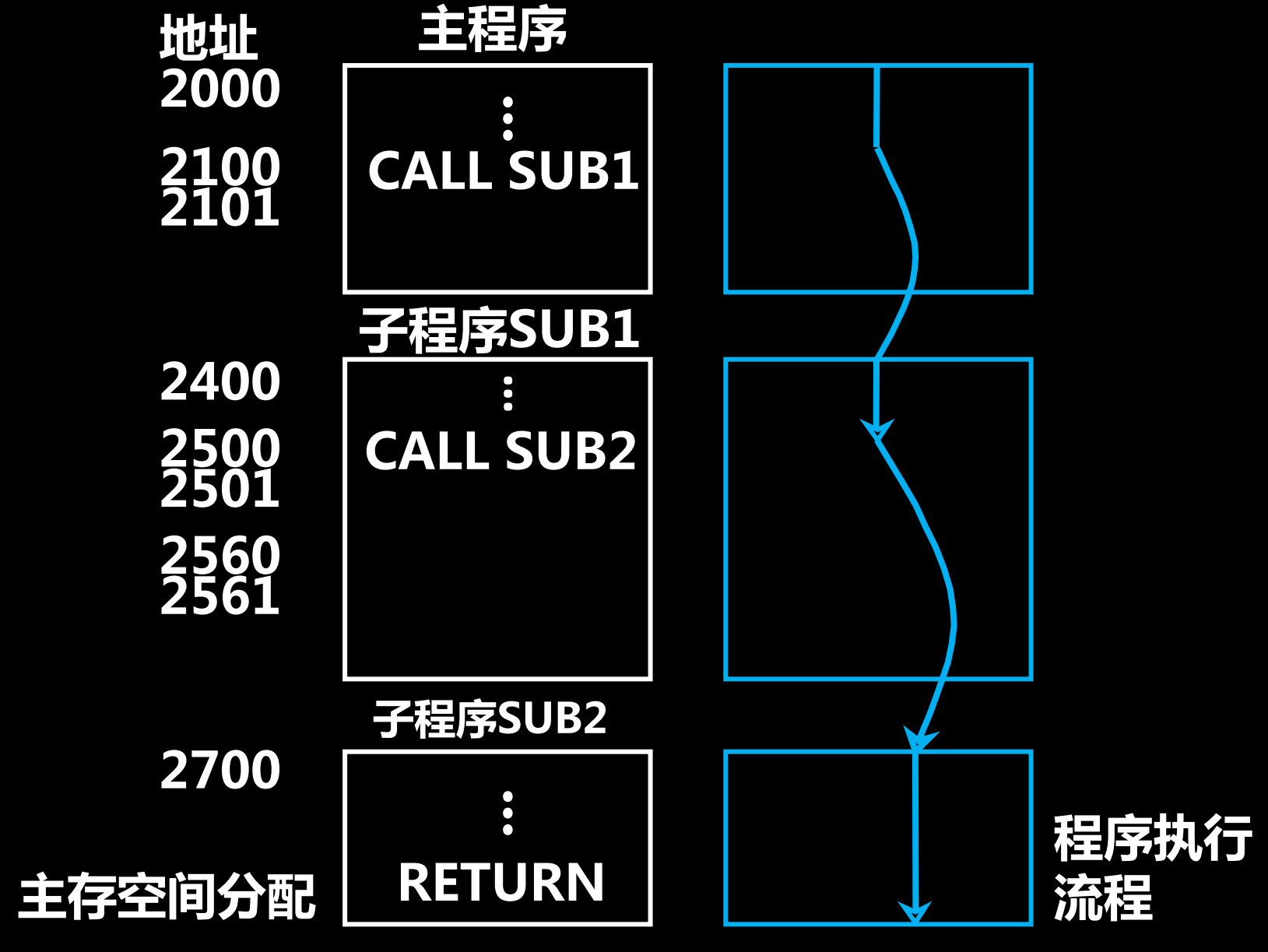
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
子程序SUB2
2700
主存空间分配
程序执行
流程

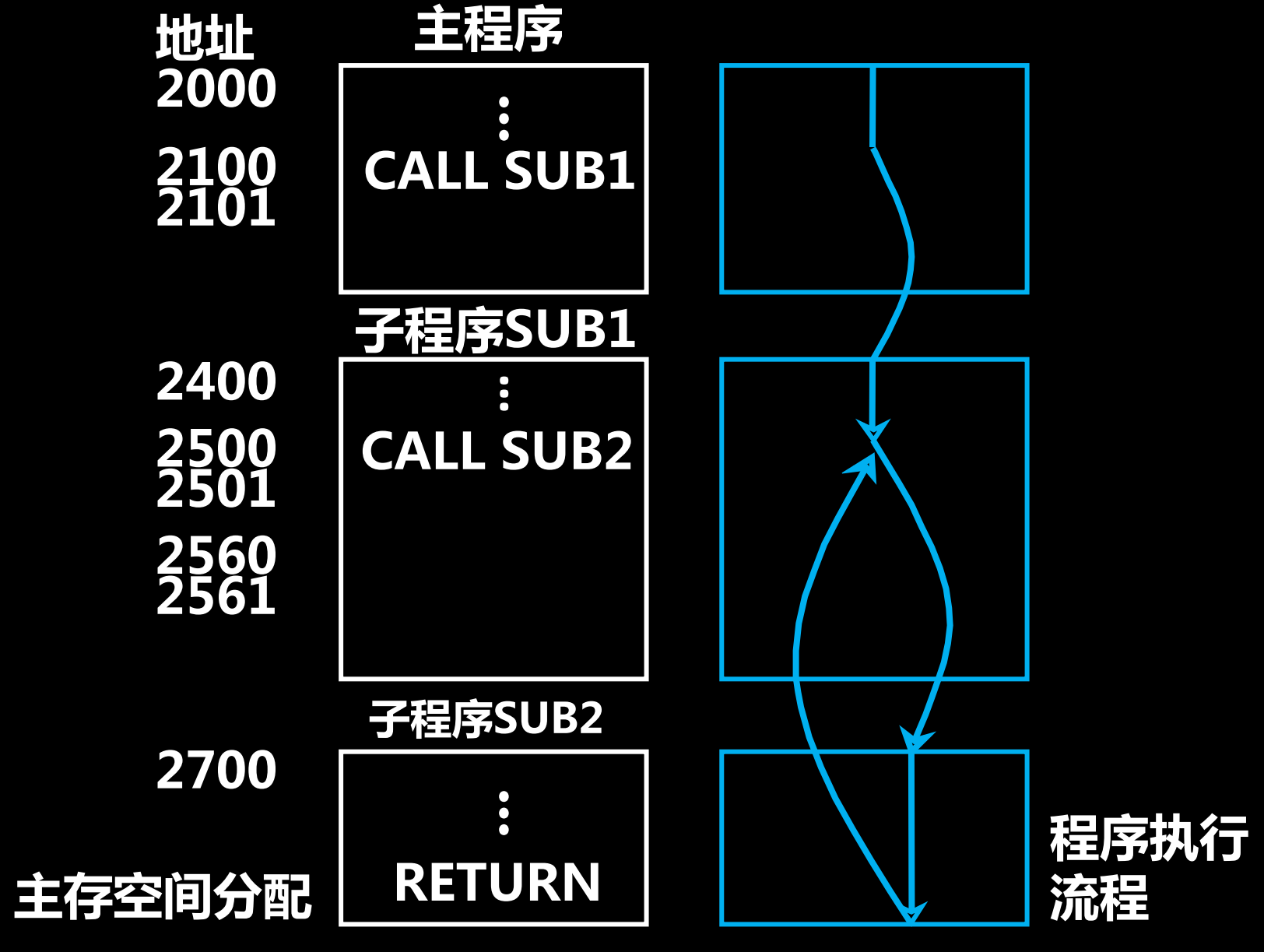
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
子程序SUB2
2700
主存空间分配
程序执行
流程

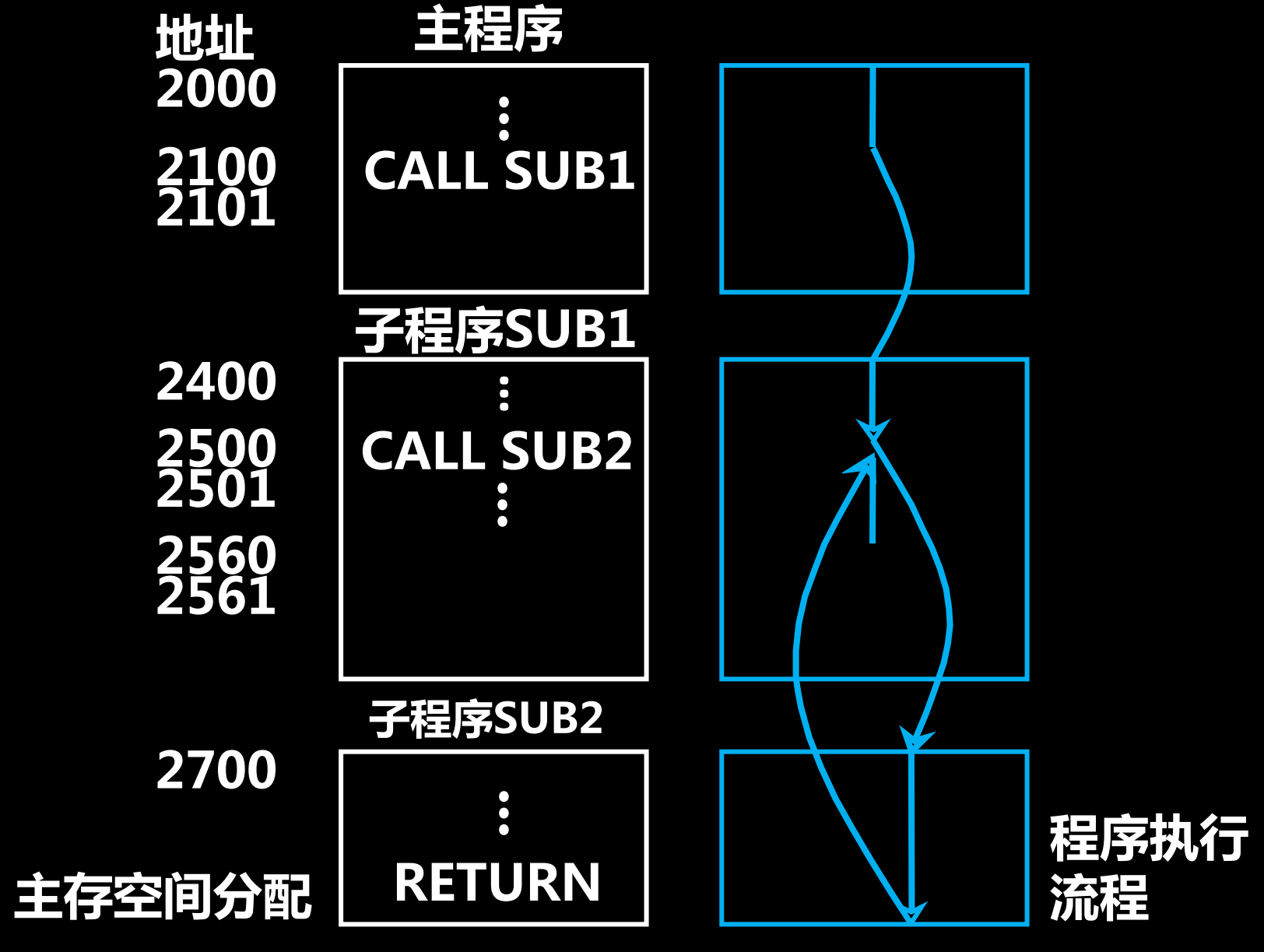
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行
流程

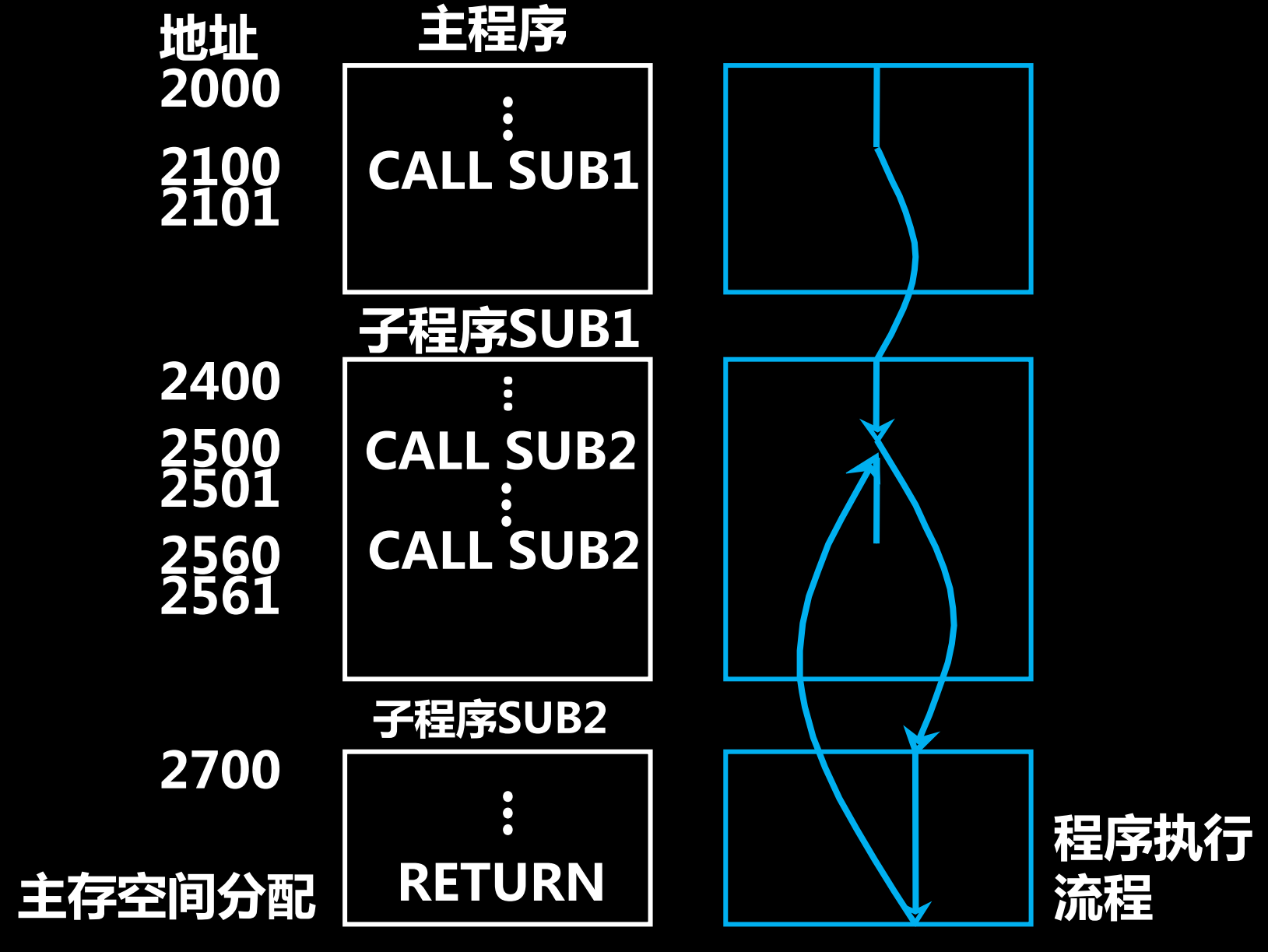
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行
流程

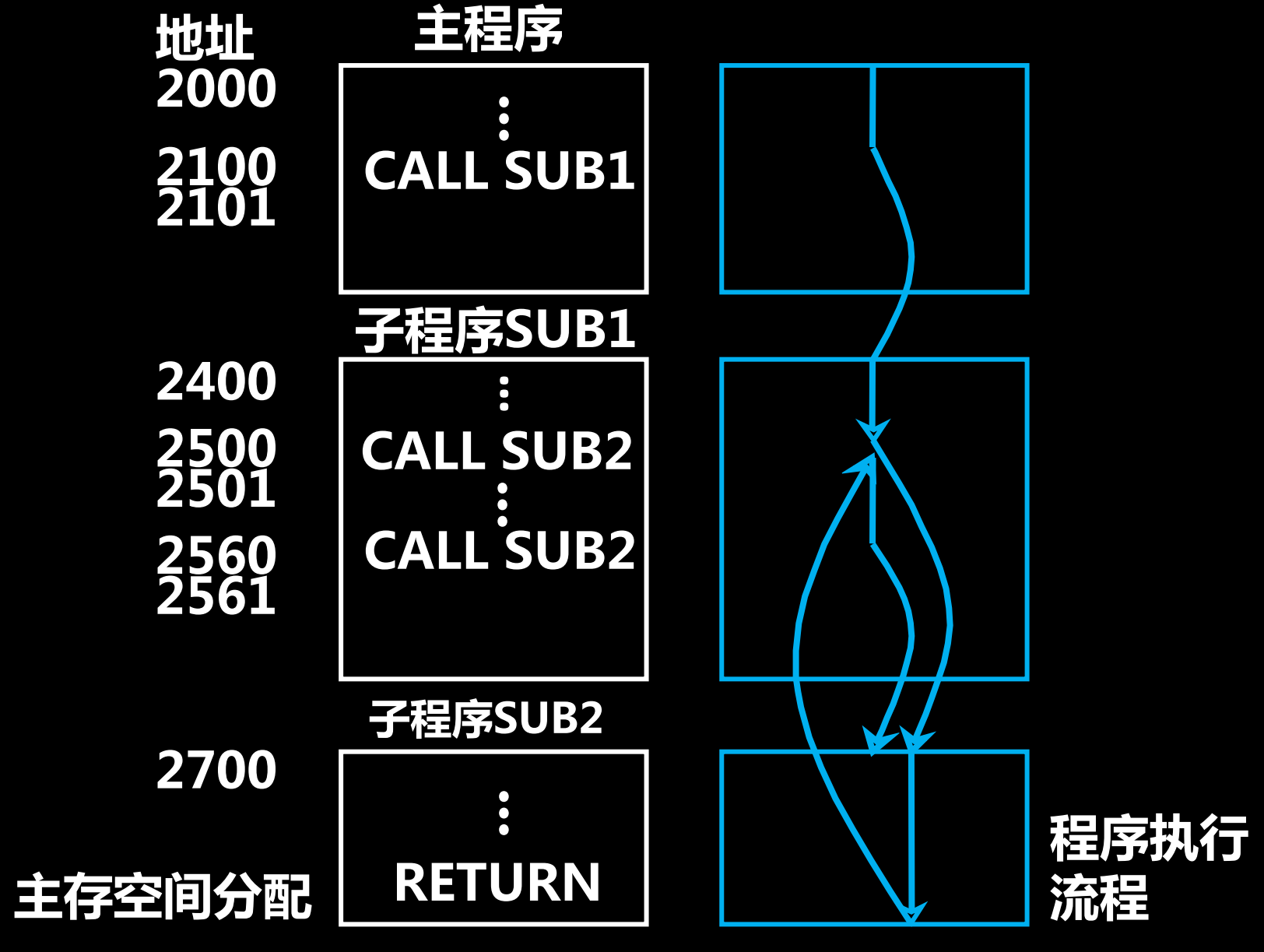
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行流程

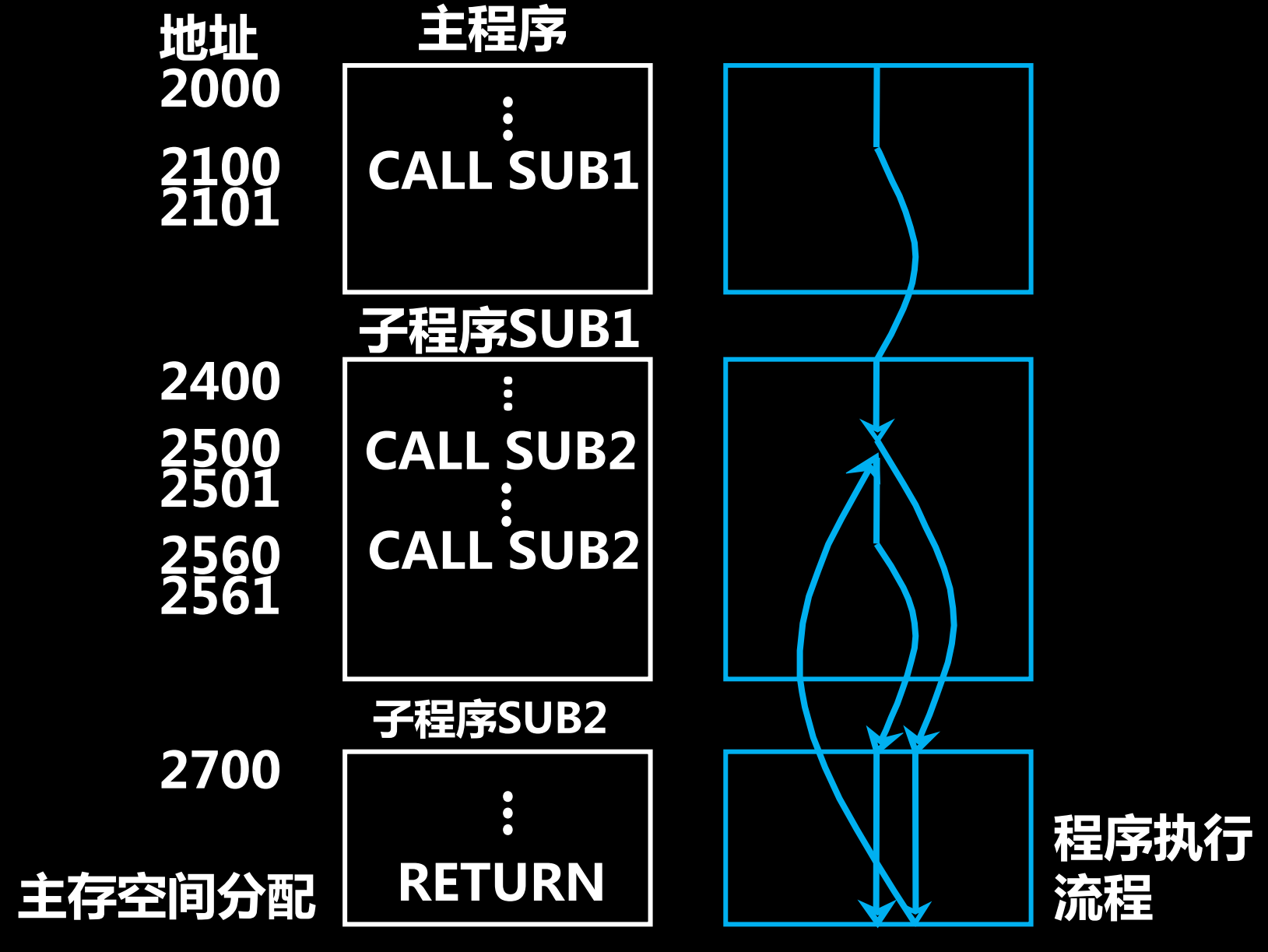
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行流程

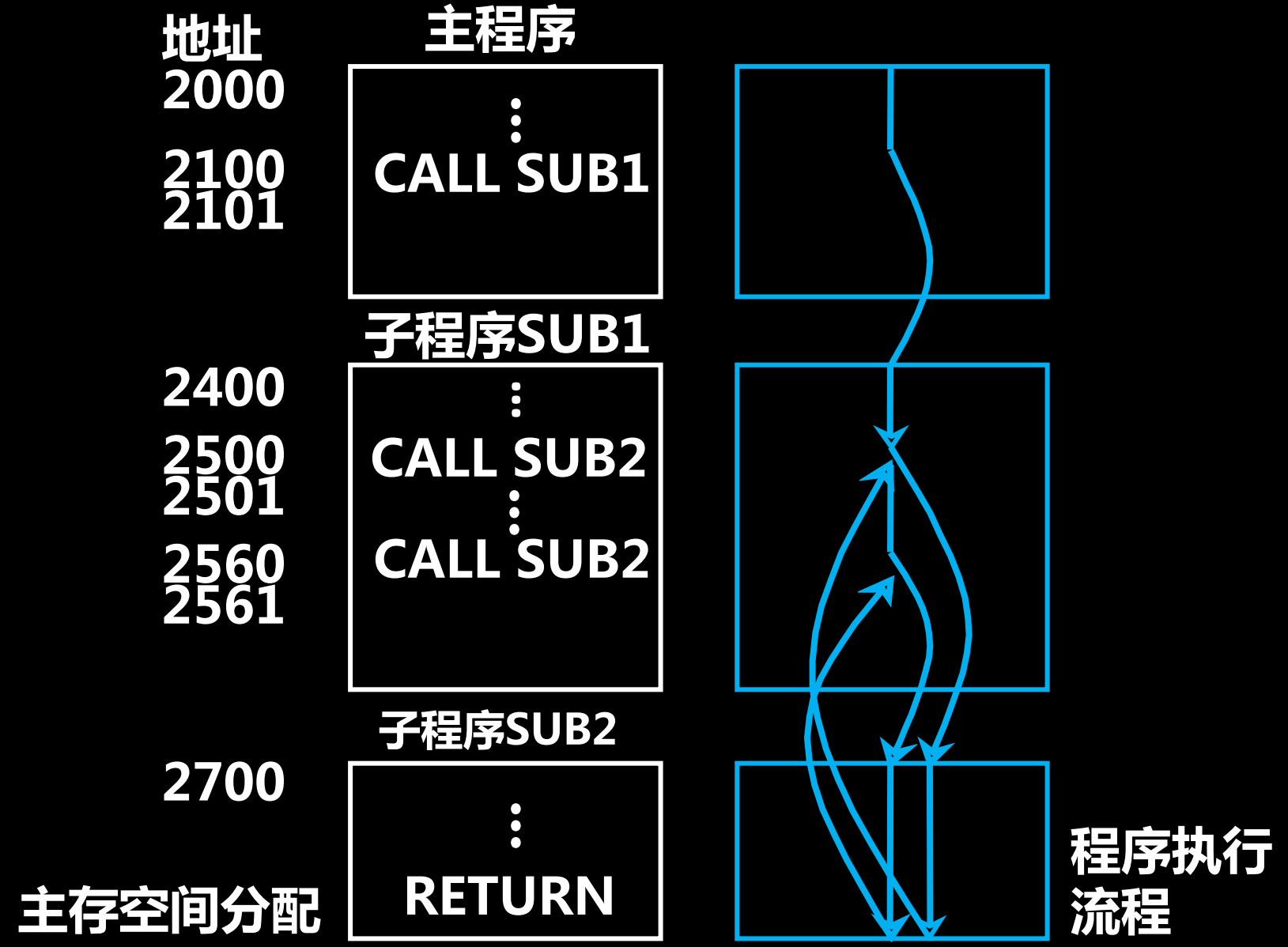
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行
流程

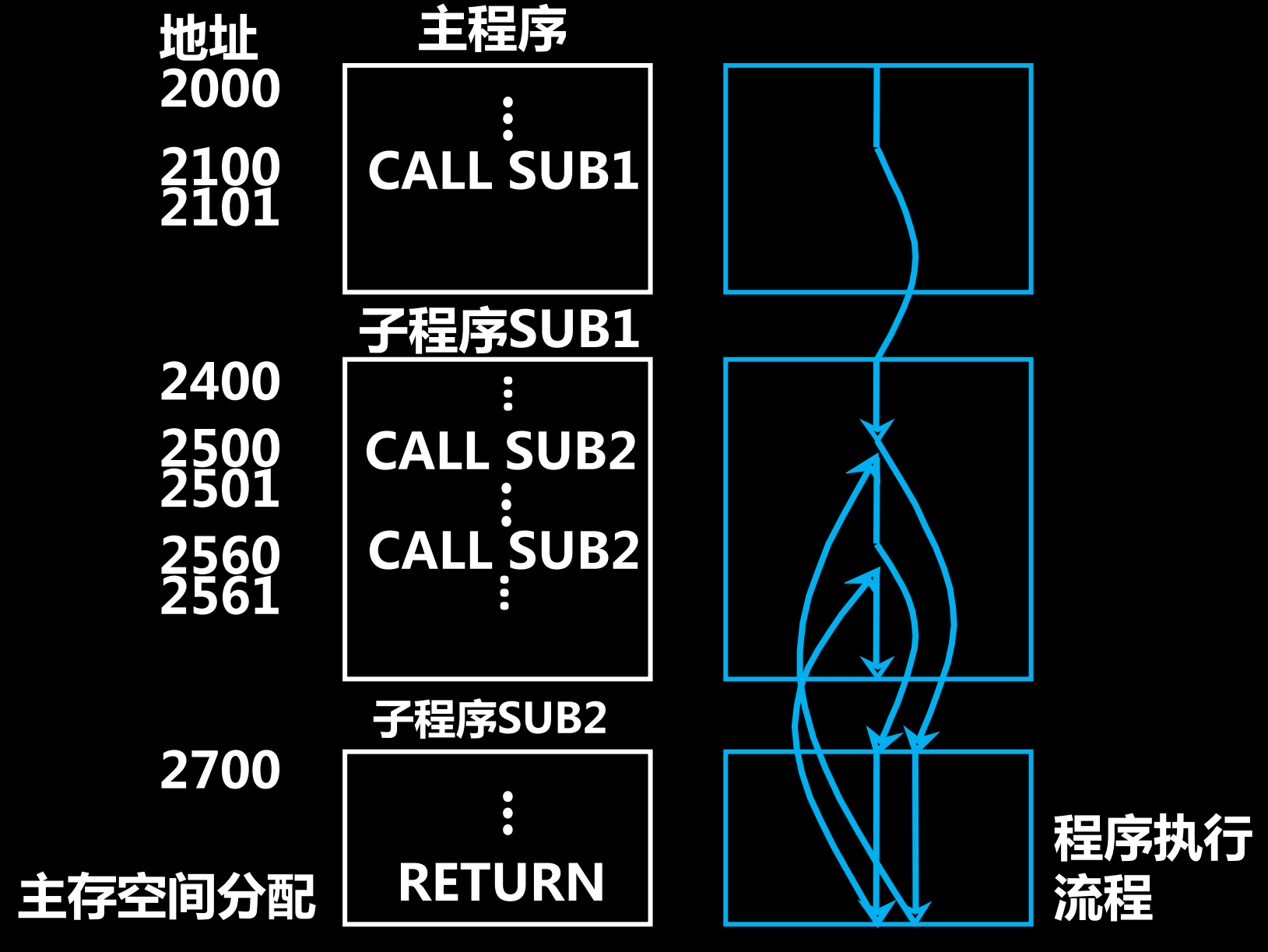
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行流程

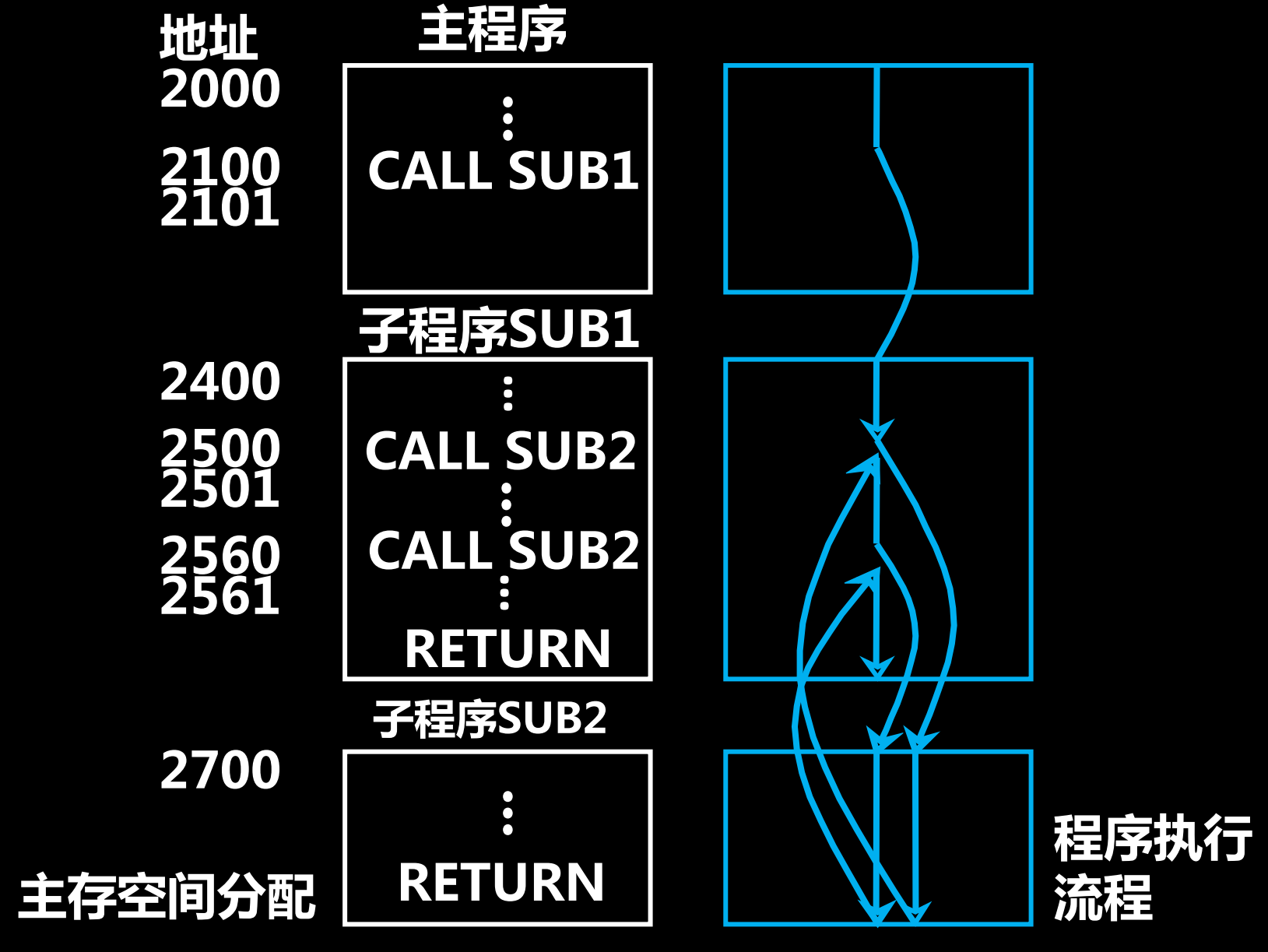
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行
流程

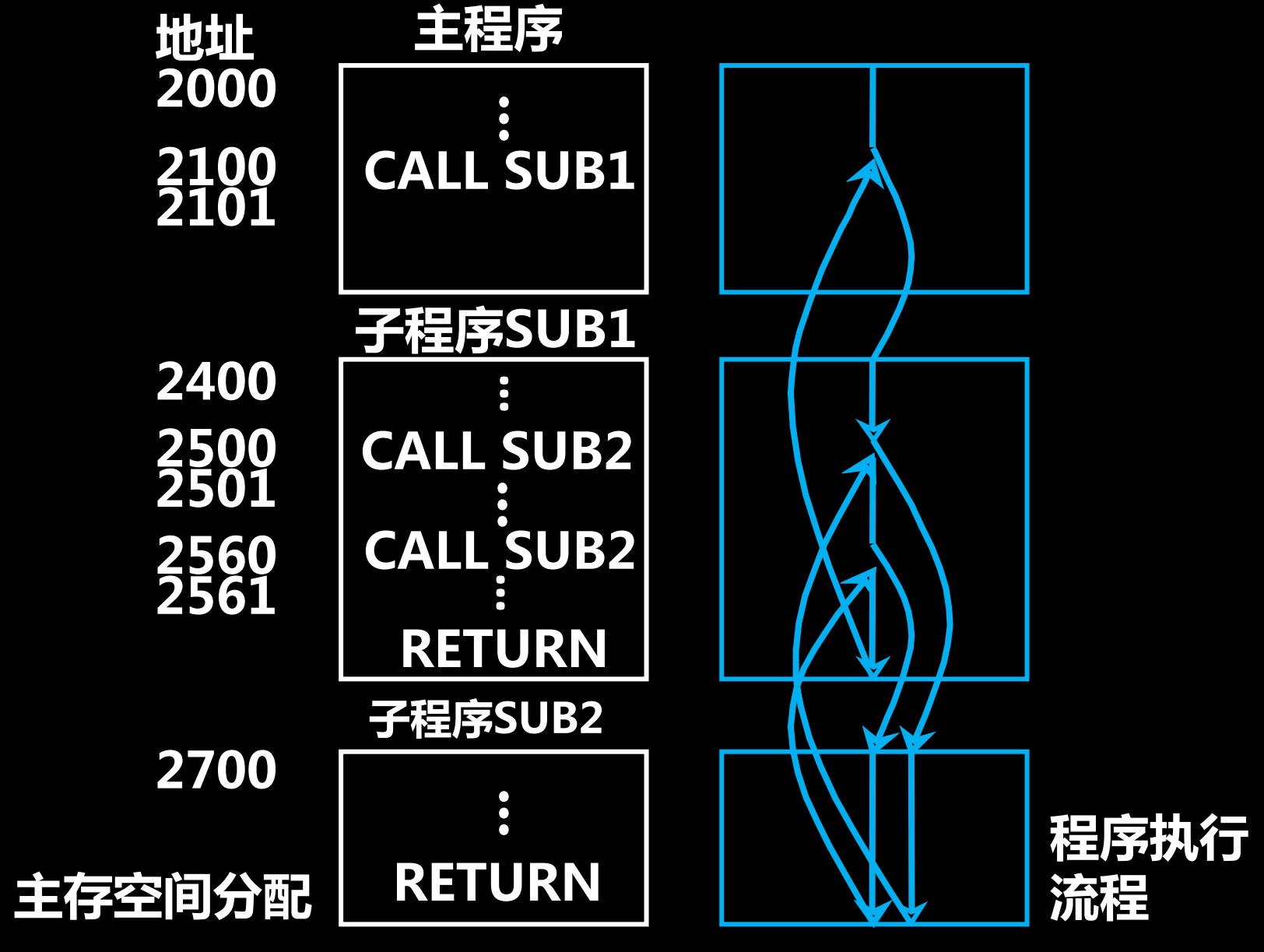
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
子程序SUB2
2700
RETURN
主存空间分配
程序执行
流程

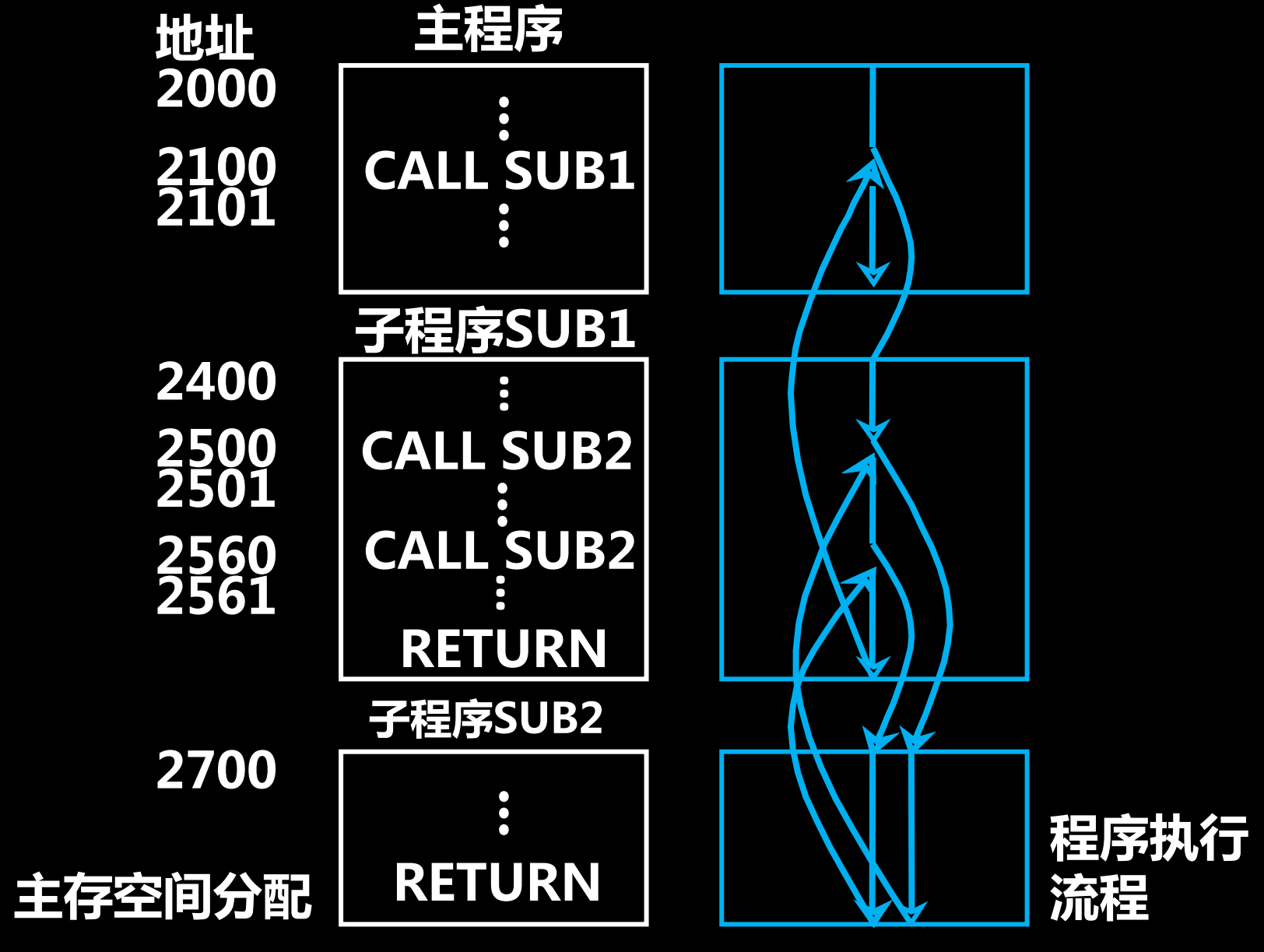
子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
RETURN
子程序SUB2
2700
RETURN
主存空间分配
程序执行流程

子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
RETURN
子程序SUB2
2700
RETURN
主存空间分配
程序执行流程

子程序调用程序


地址
主程序
2000
2100
2101
CALL SUB1
子程序SUB1
2400
2500
2501
2560
2561
CALL SUB2
CALL SUB2
RETURN
子程序SUB2
2700
RETURN
主存空间分配
程序执行流程

(4) 陷阱（TRAP）指令

- 一旦机器运行出现意外故障（未定义指令、除0、设备故障、电压不稳），计算机发出陷阱信号（陷阱隐指令），暂停当前指令的执行，转入故障处理程序。陷阱指令不提供给用户使用，由机器自动执行。
- 也有某些机器提供陷阱指令，例如IBM - PC提供的INT ××软中断指令，用来完成系统调用

5. 输入输出指令

对I/O单独编址的计算机，设置有专门的输入输出指令，用来操纵外设。例如：

IN　AX, [20] 外设端口→ CPU 的寄存器

OUT DX, AX　CPU 的寄存器→ 外设端口

独立编址 vs 统一编址

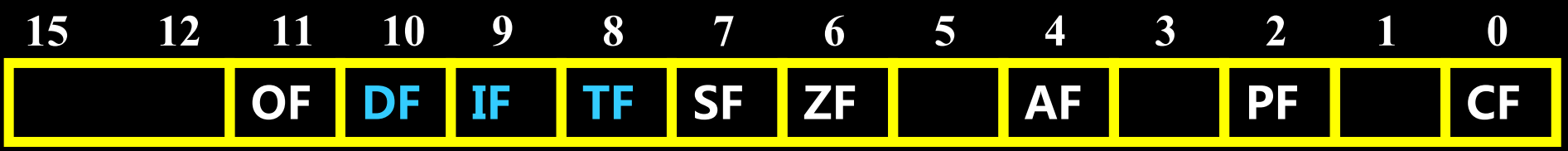
6. 其他指令

- 停机指令、空操作指令、开中断指令、关中断指令、置条件码指令。
- 字符串传送、字符串比较、字符串查询
- 特权指令（操作系统用）
- 向量指令
- 多处理机指令

推荐阅读：8086的程序状态字

程序状态字PSW（FLAG）是CPU内部的一个寄存器，用来存放两类信息：

- **体现当前指令执行结果的各种状态信息**，如有无进位（C位），有无溢出（O位），结果正负（S位），结果是否为零（Z位），奇偶标志位（P位）等；状态位全部自动设置。
- **控制信息**，如允许中断(I位)，跟踪标志（T位）等。



寻址方式（1）

大连理工大学 赖晓晨

我国计算机科学的先驱，夏培肃院士

- 1952年，开始研究通用电子数字计算机，国内最早；
- 1953年，华罗庚领导成立我国第一个计算机科研小组。
- 1956年，参加筹建中科院计算所；
- 1960年，她设计成功我国第一台电子计算机，107机；
- 1968年，夏培肃提出最大时间差流水线原理（比美国学者早一年提出），大大缩短流水线计算机时钟周期。
- 20世纪70年代末期，她负责研制成功高速阵列处理机，性能优于同期美国禁运机型；
- 夏培肃于1978年和1986年先后创办《计算机学报》和英文学报《Journal of Computer Science and Technology》，并担任第一任主编。

寻址方式类别

- 指令寻址
- 数据寻址

一、指令寻址

顺序　　（PC）+ 1 ⟶ PC

跳跃　　由转移指令指出

二、数据寻址

数据寻址有多种，需要在指令中明确指出采用哪一种寻址方式，可以专门设置一个寻址方式特征字段，或纳入操作码中。

操作码	寻址特征	形式地址

形式地址 A　　指令字中的地址

有效地址 EA　　操作数的真实地址

有效地址由形式地址根据寻址方式来确定。

约定　　**指令字长 = 存储字长 = 机器字长**

1. 立即寻址

又称作立即数寻址，即指令中的形式地址部分不是一个操作数的地址，而是操作数本身。

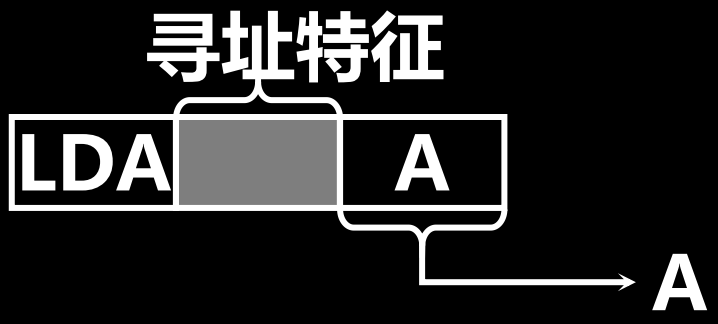
MOV AX, 20H

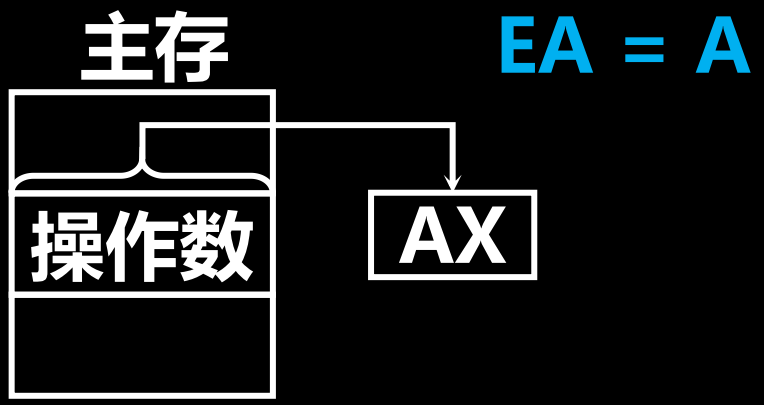
可正可负 补码

- 指令执行阶段不访存
- A 的位数限制了立即数的范围

2. 直接寻址

指令中的形式地址部分即为有效地址。



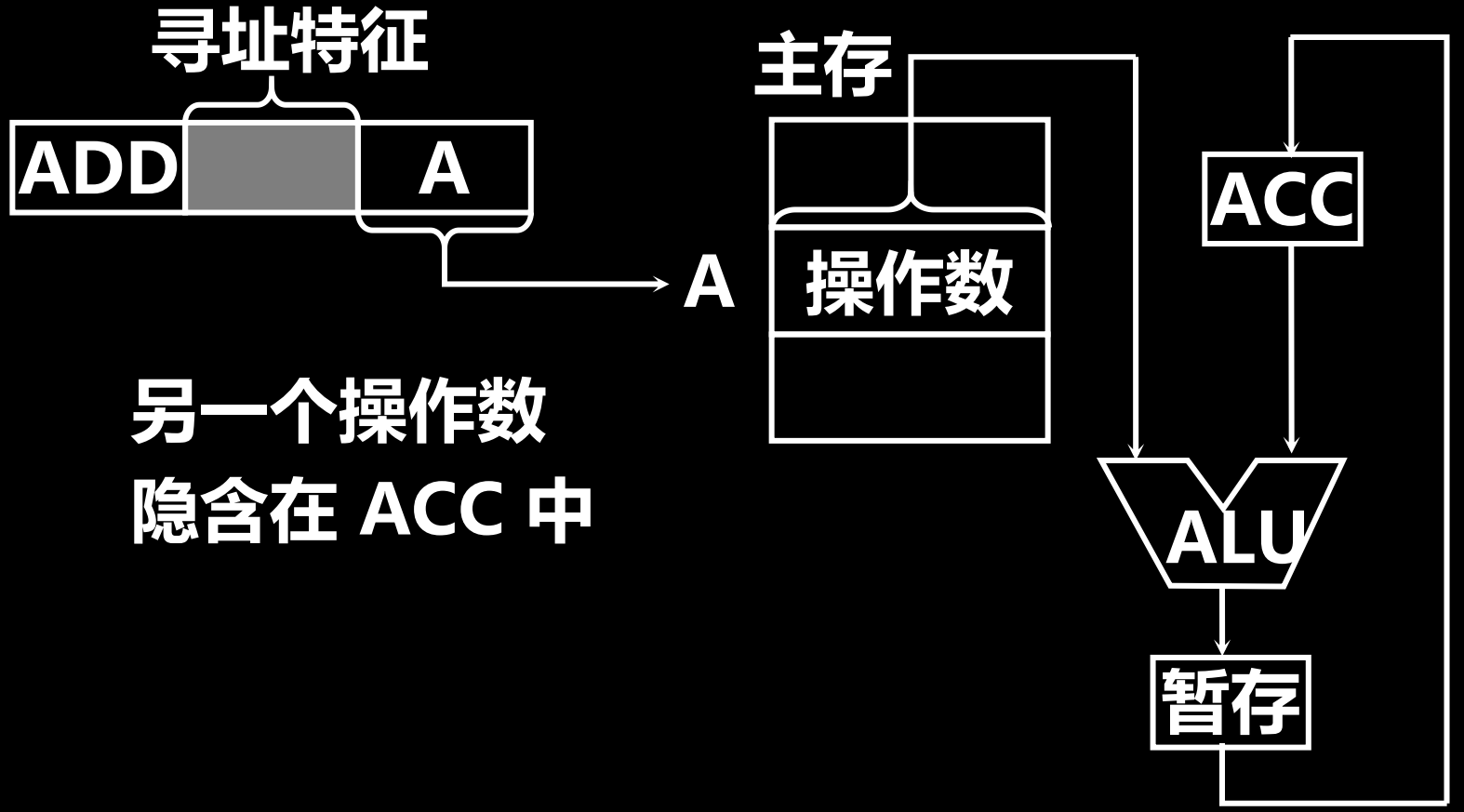


MOV AX, [20H]

- 执行阶段访问一次存储器
- A 的位数限制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）

3. 隐含寻址

指令中不直接给出操作数地址，操作数地址通常隐含在操作码或某个（约定）寄存器中。



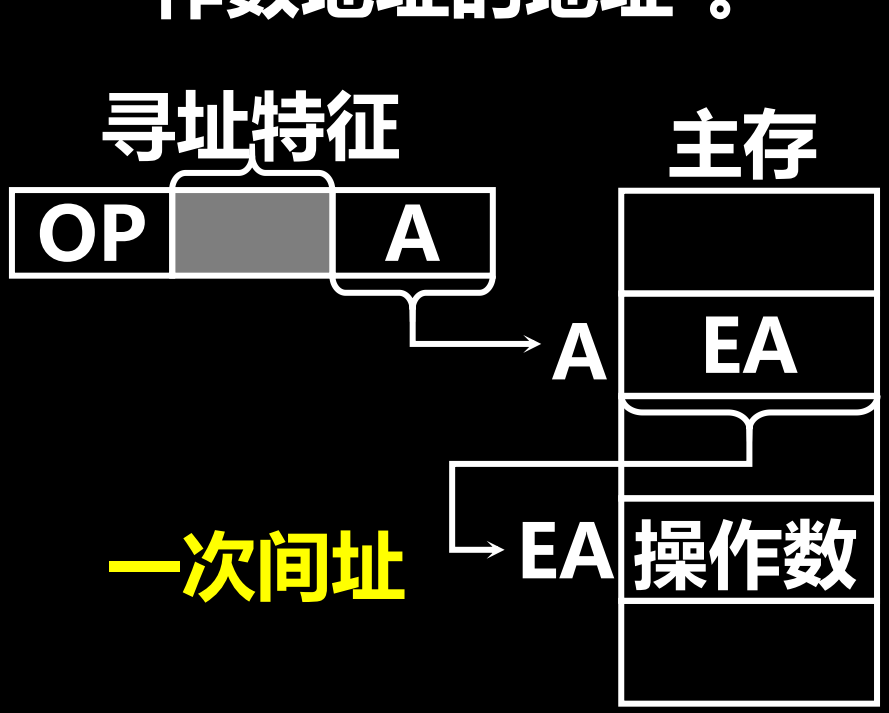
8086指令集中的隐含寻址

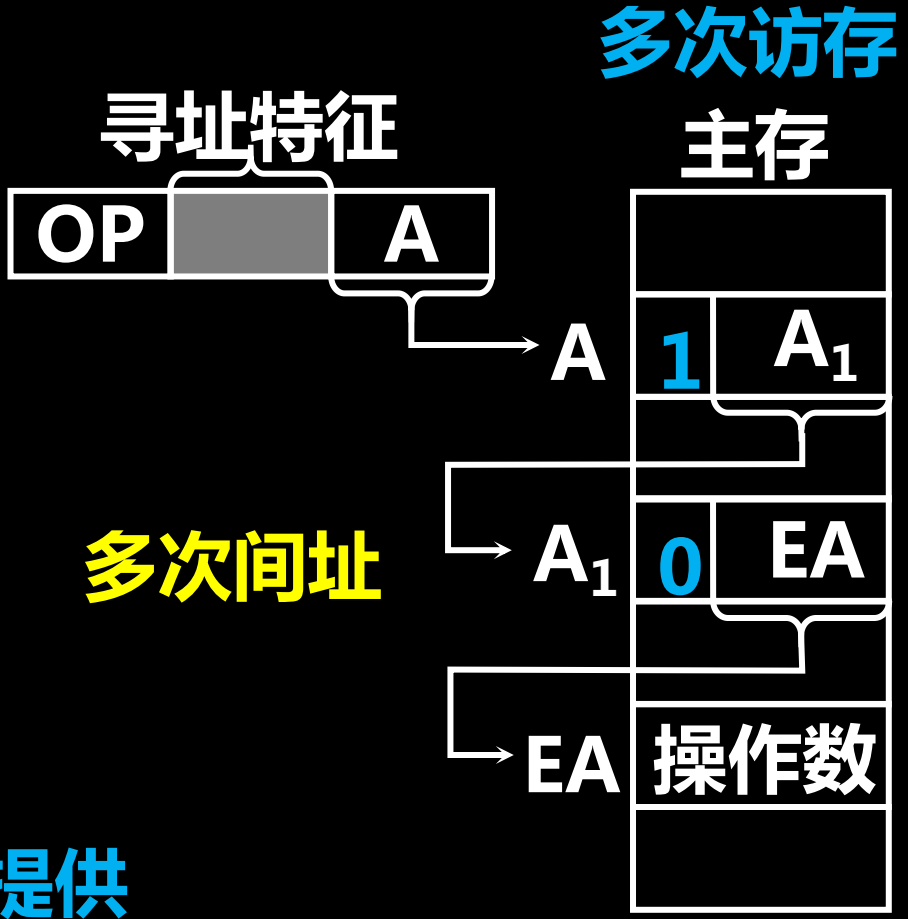
- 如8086的MUL指令，被乘数隐含在 AX（16位）或 AL（8位）中；
- MOVS指令源操作数的地址隐含在 SI 中，目的操作数的地址隐含在 DI 中。

指令字中少了一个地址字段，可缩短指令字长

4. 间接寻址

指令中的形式地址不是操作数的地址，而是“操作数地址的地址”。



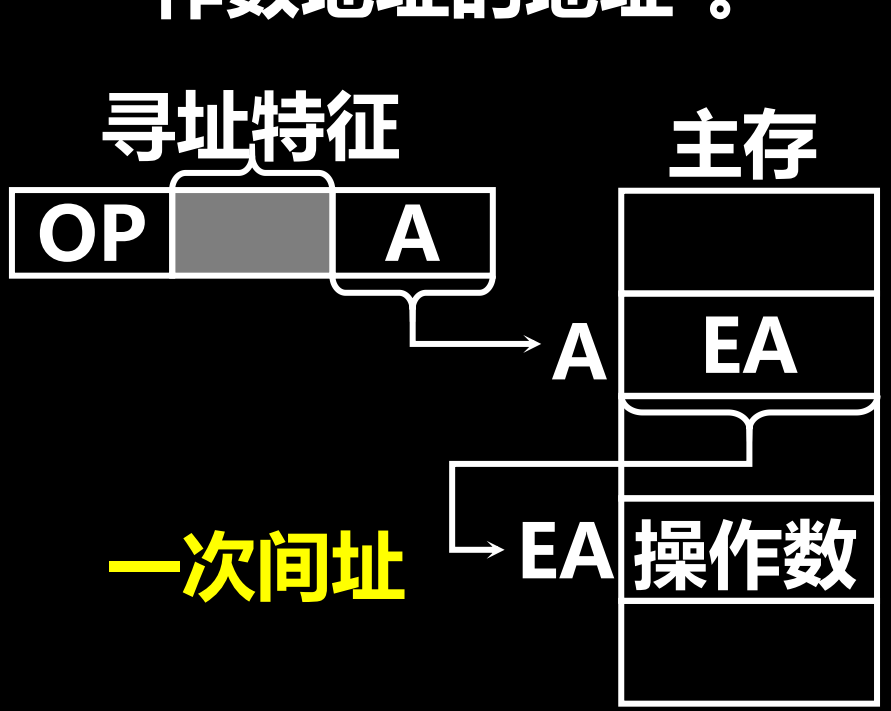


EA =（A）

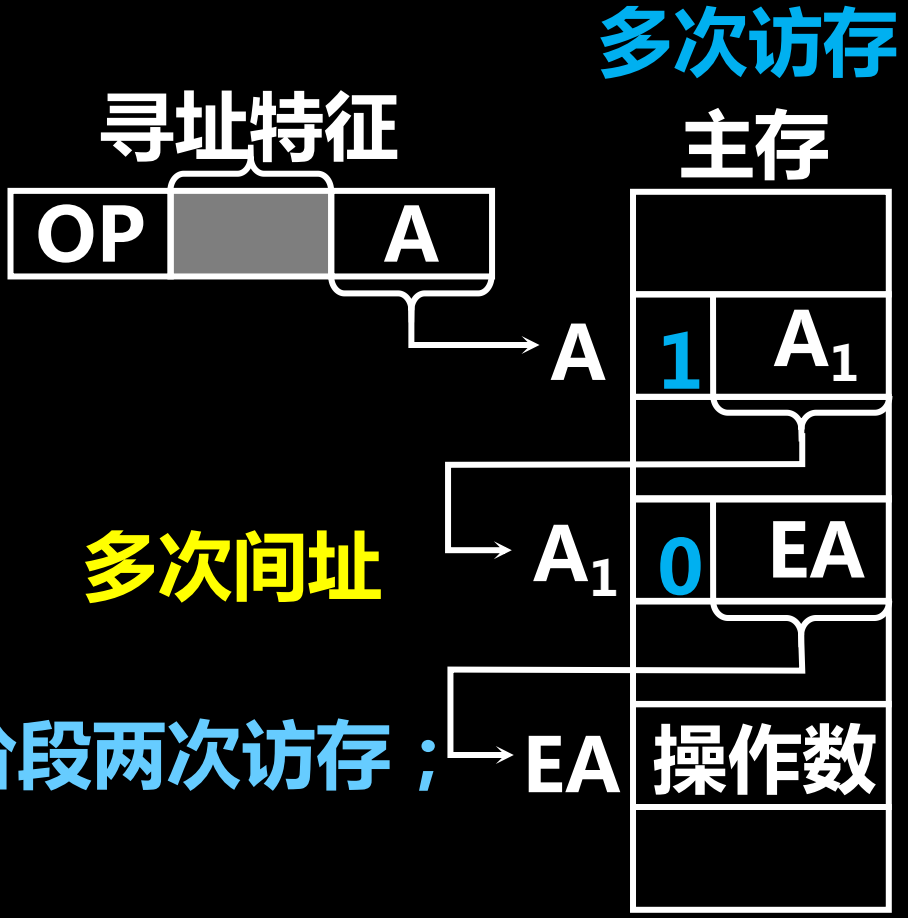
有效地址由形式地址间接提供

4. 间接寻址

指令中的形式地址不是操作数的地址，而是“操作数地址的地址”。



多次访存

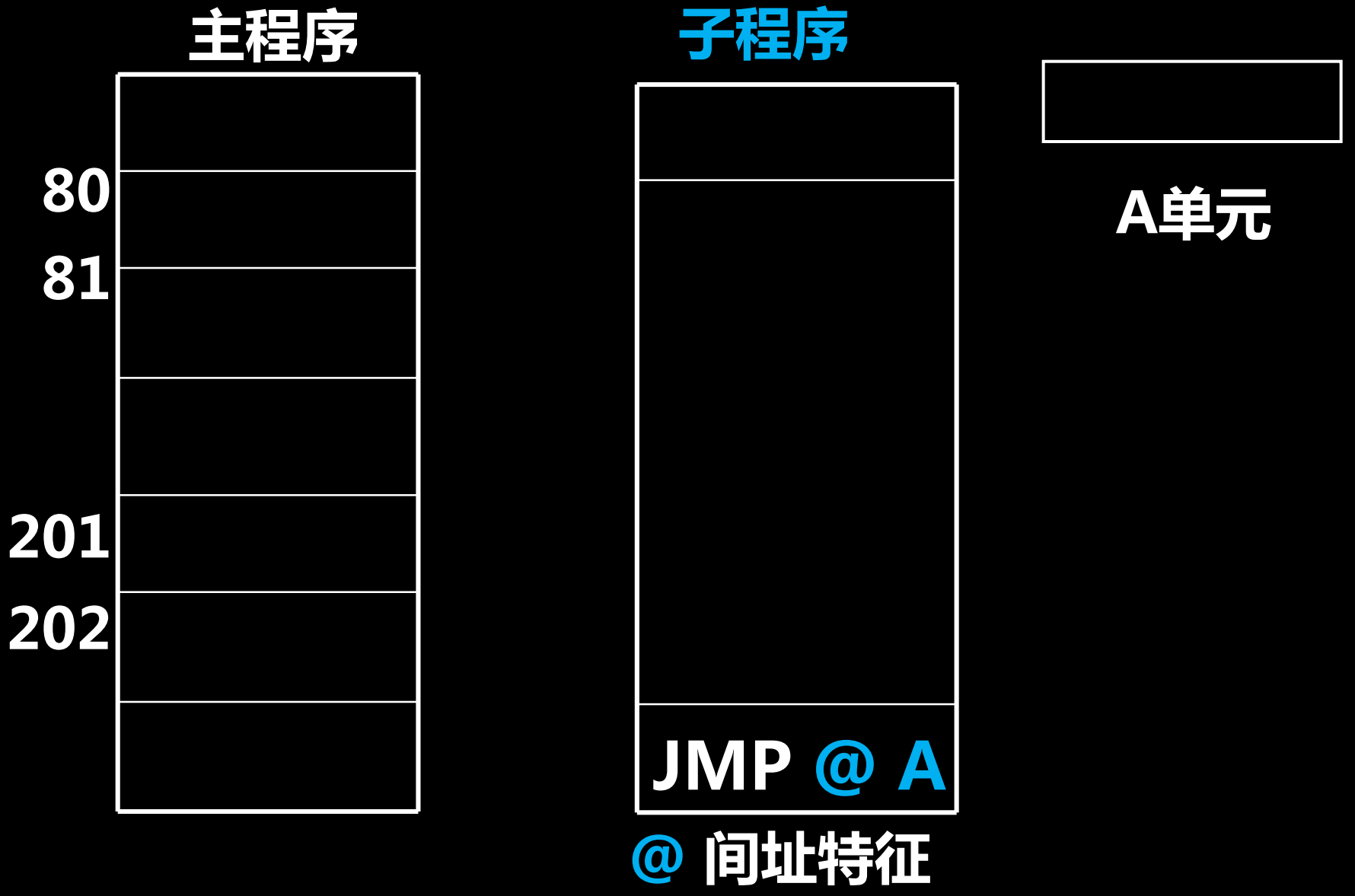


一次间接寻址，指令执行阶段两次访存；

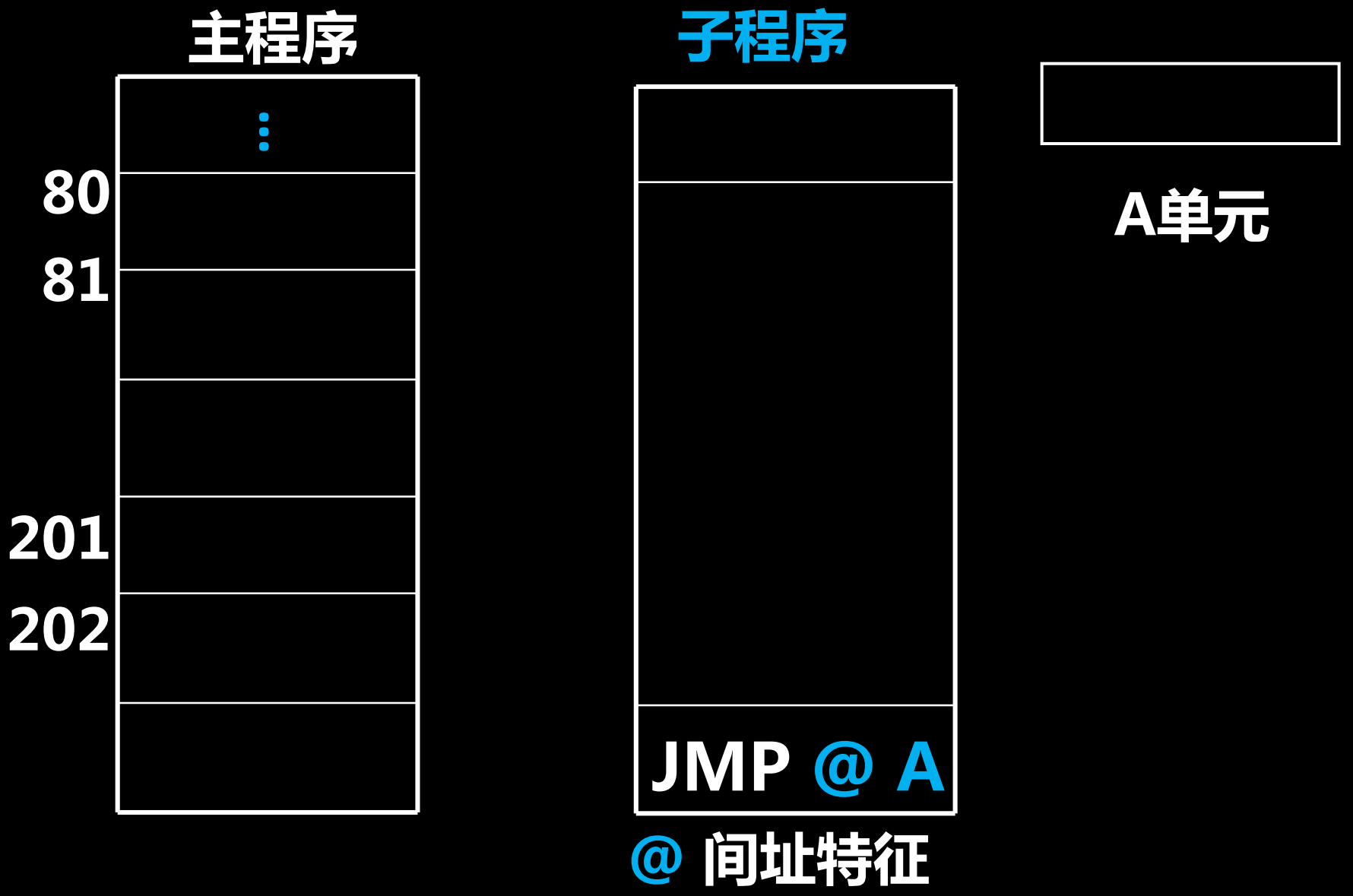
可以扩大寻址范围

便于编制转移程序

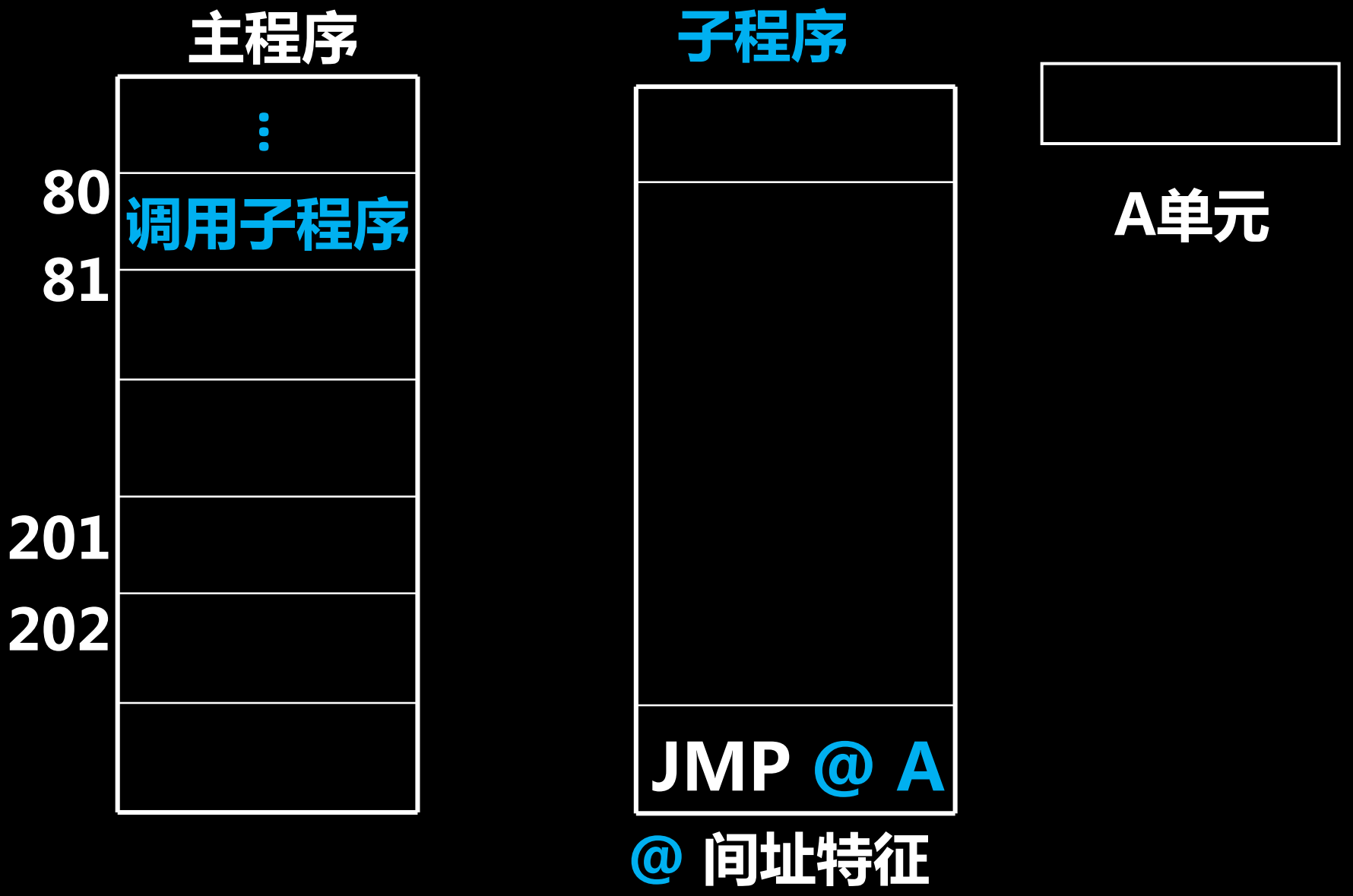
间接寻址编程举例



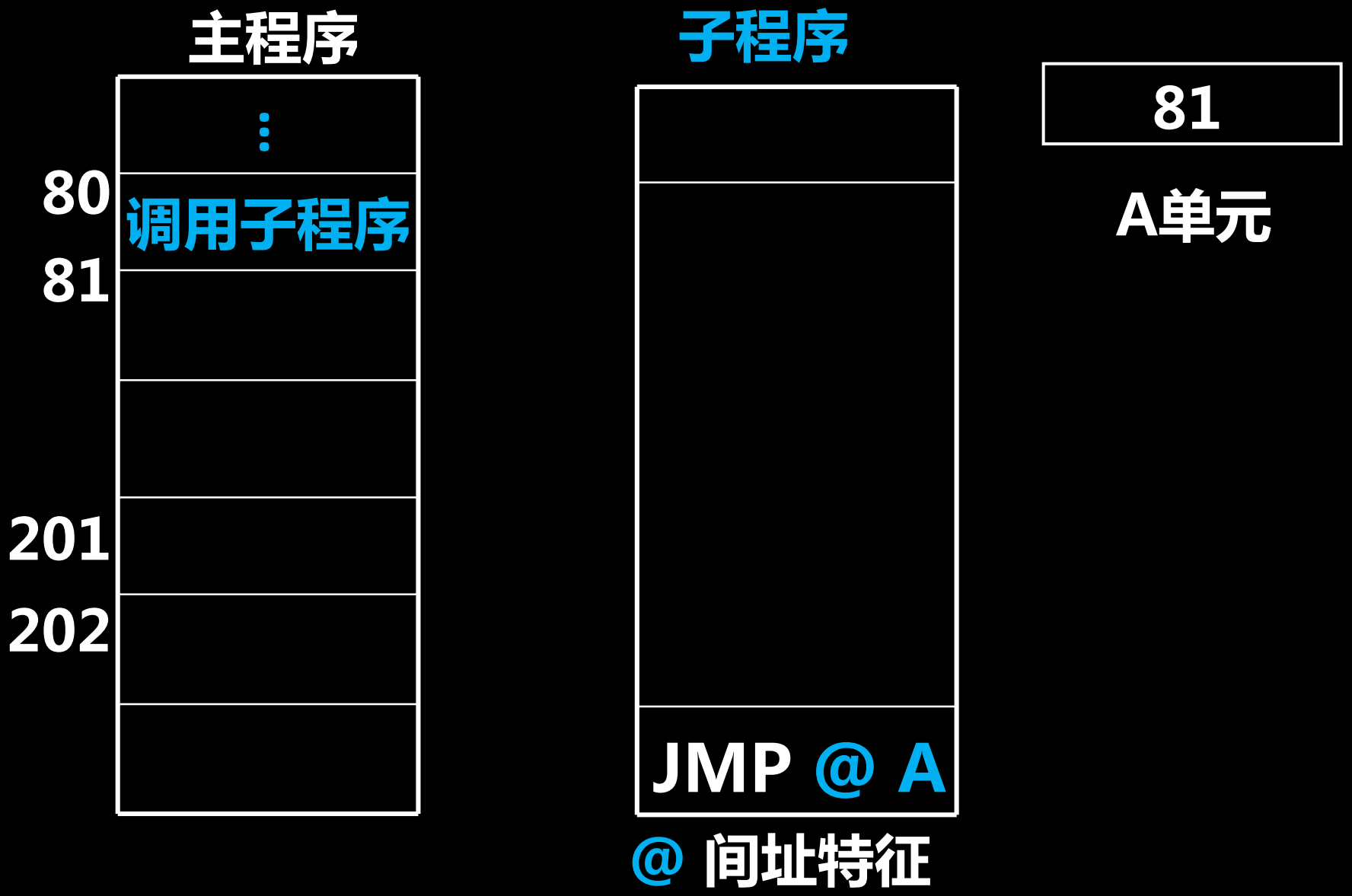
间接寻址编程举例



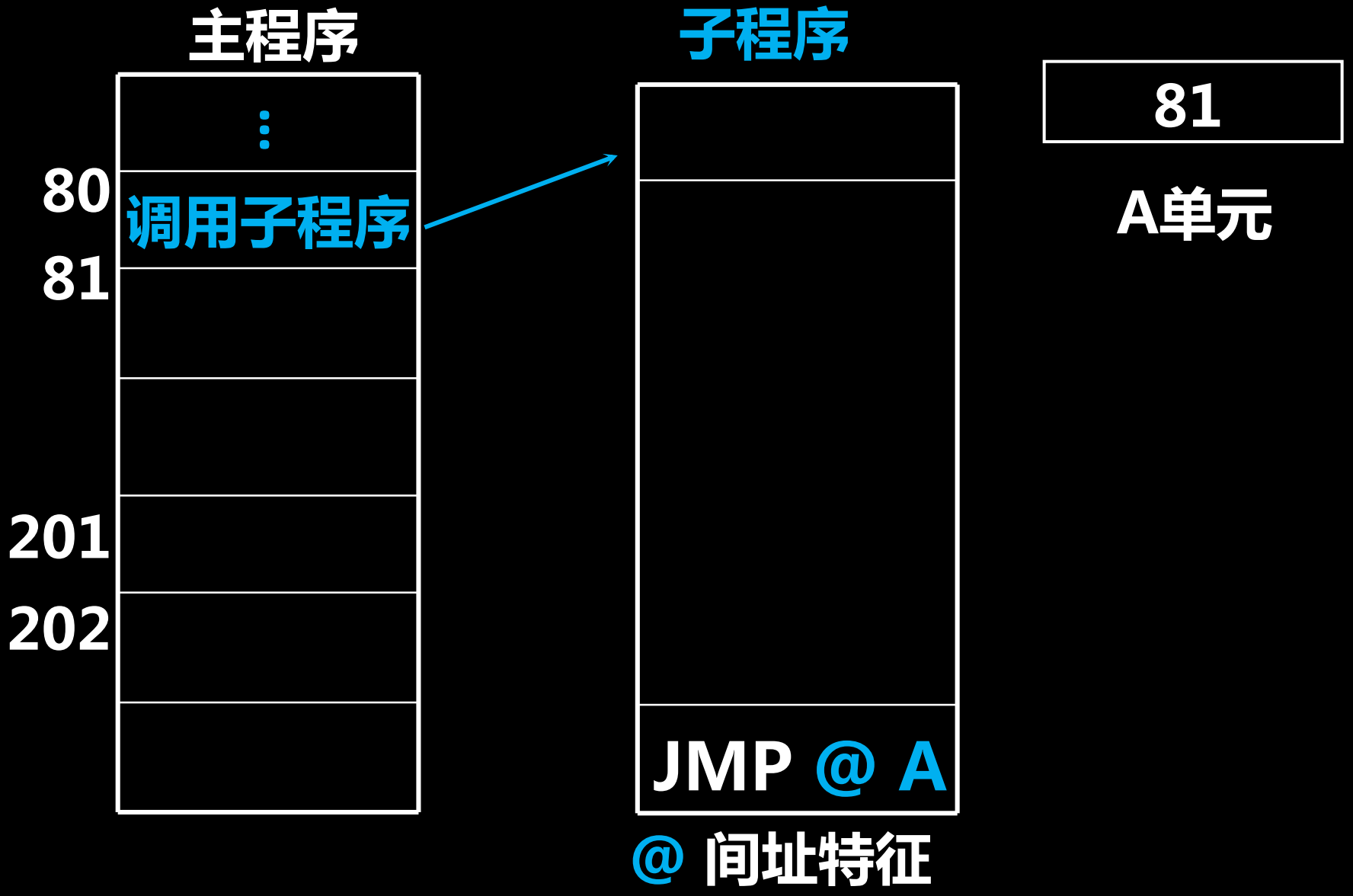
间接寻址编程举例



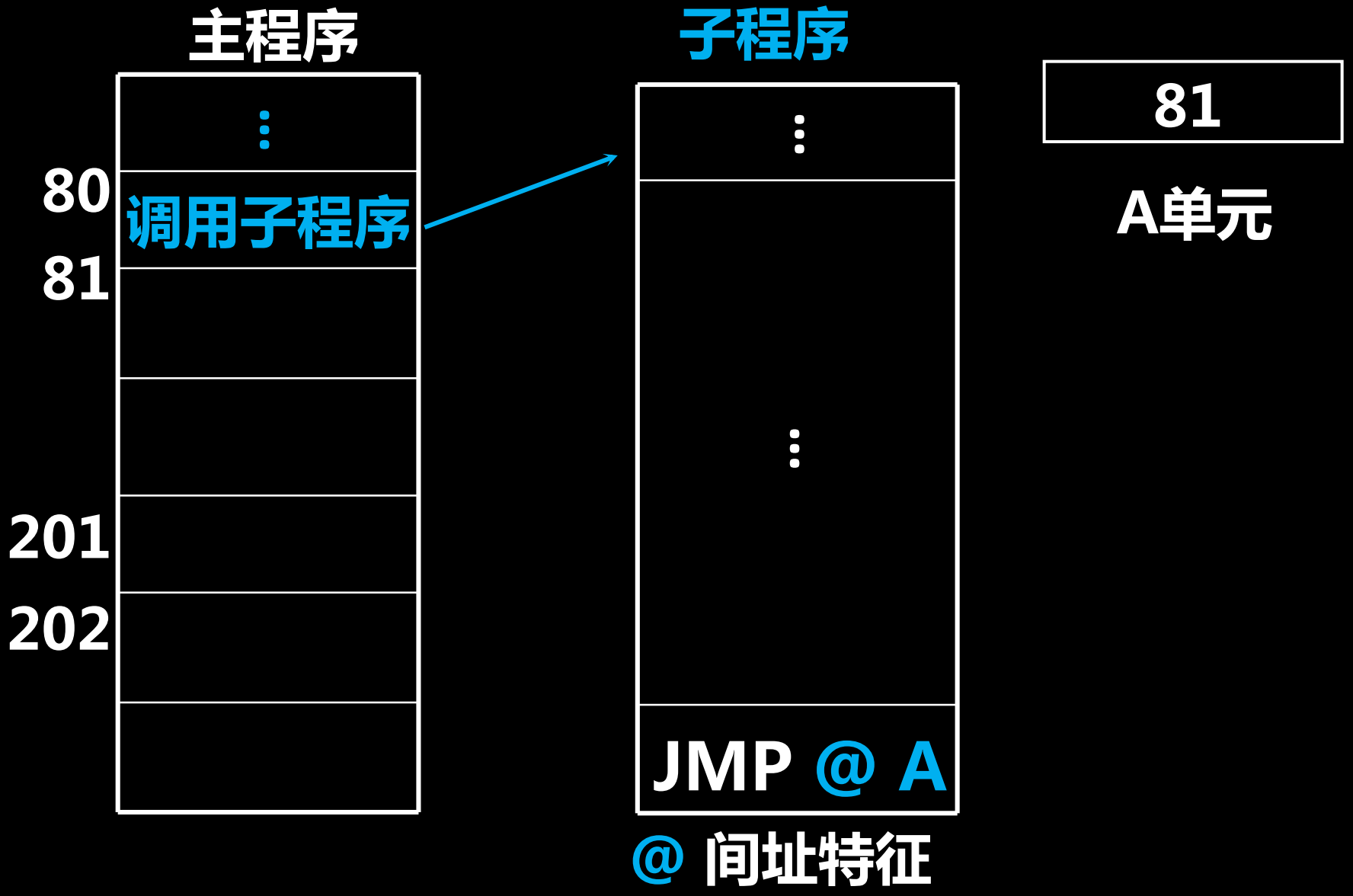
间接寻址编程举例



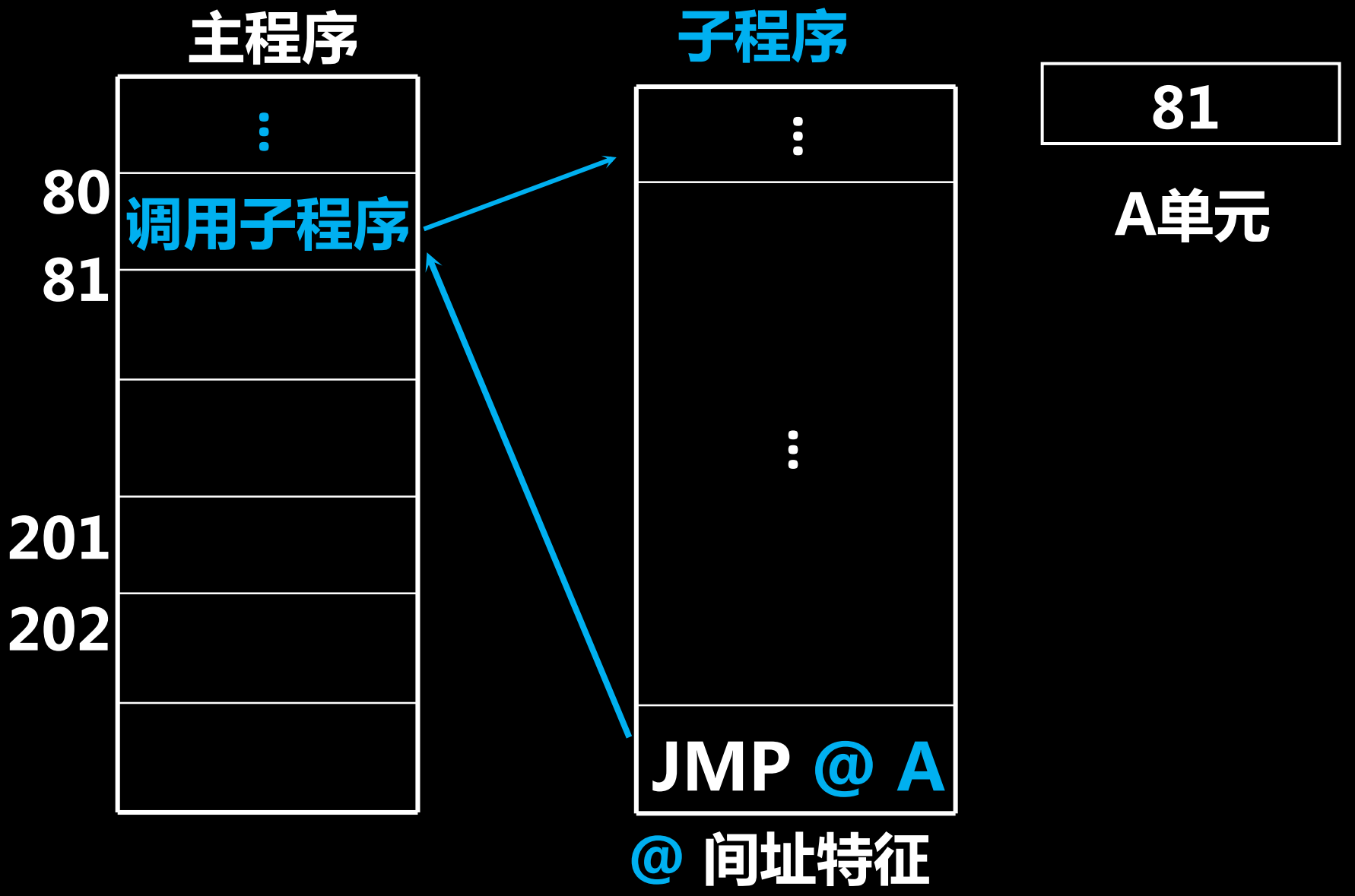
间接寻址编程举例



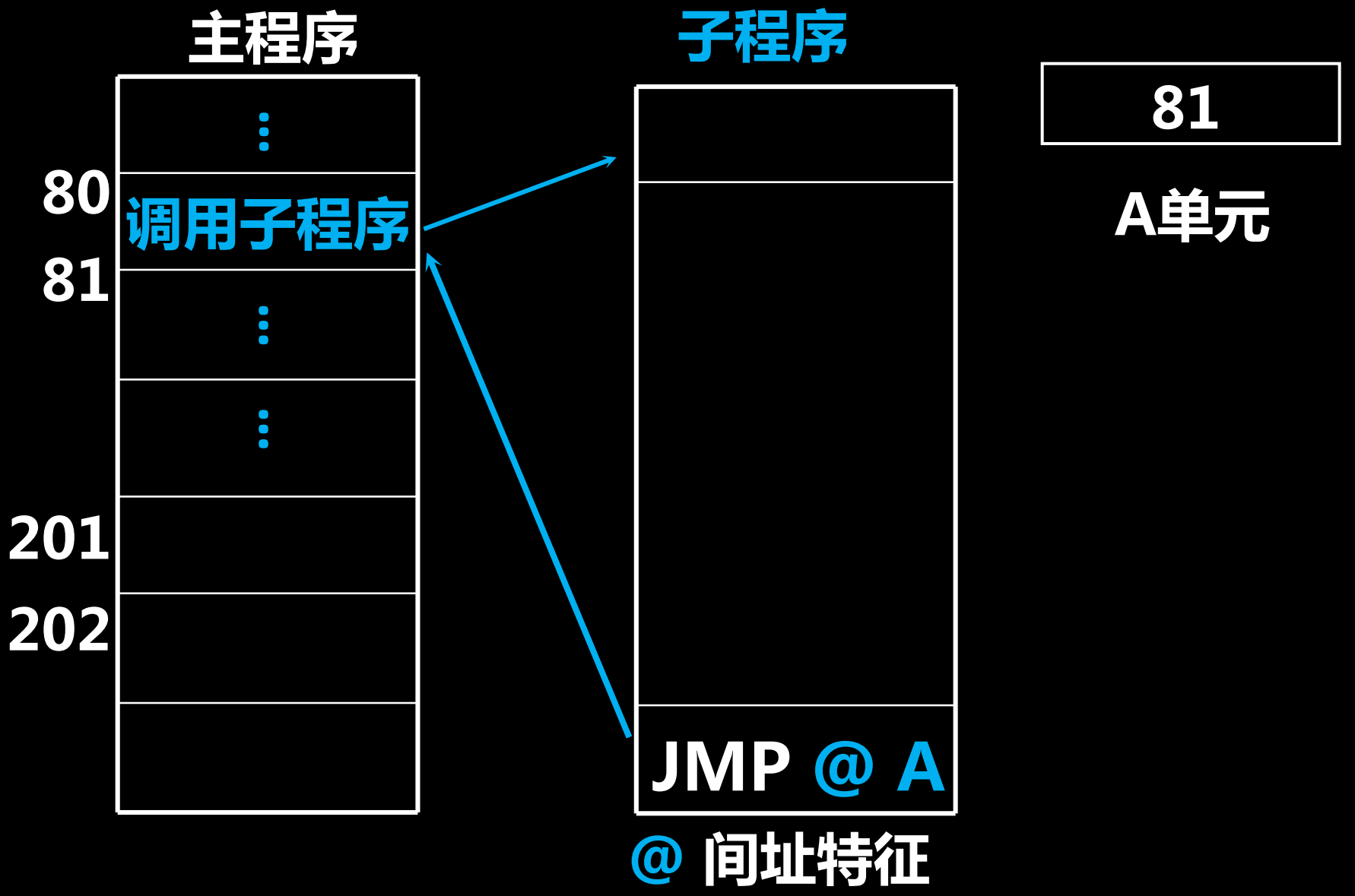
间接寻址编程举例



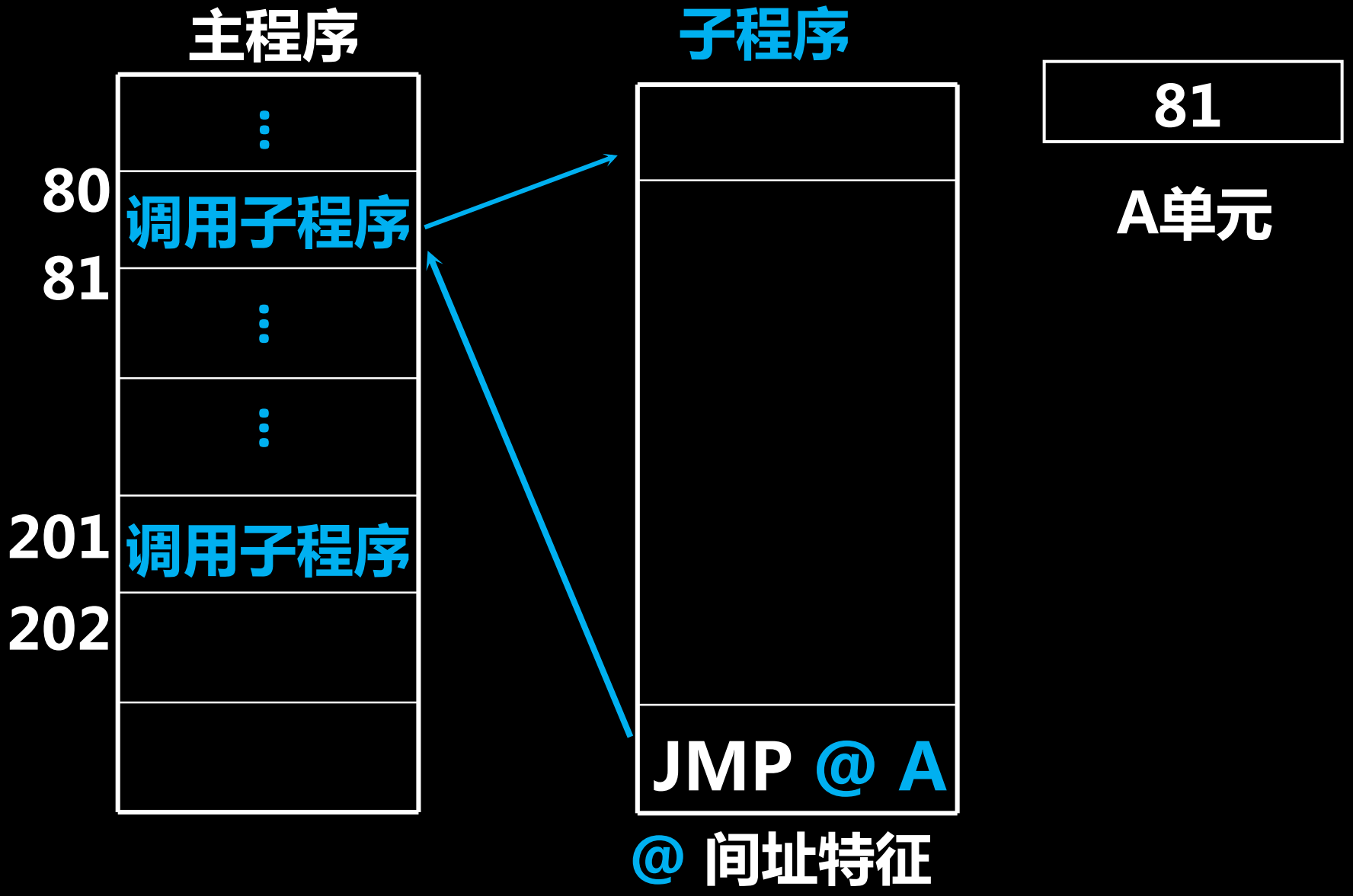
间接寻址编程举例



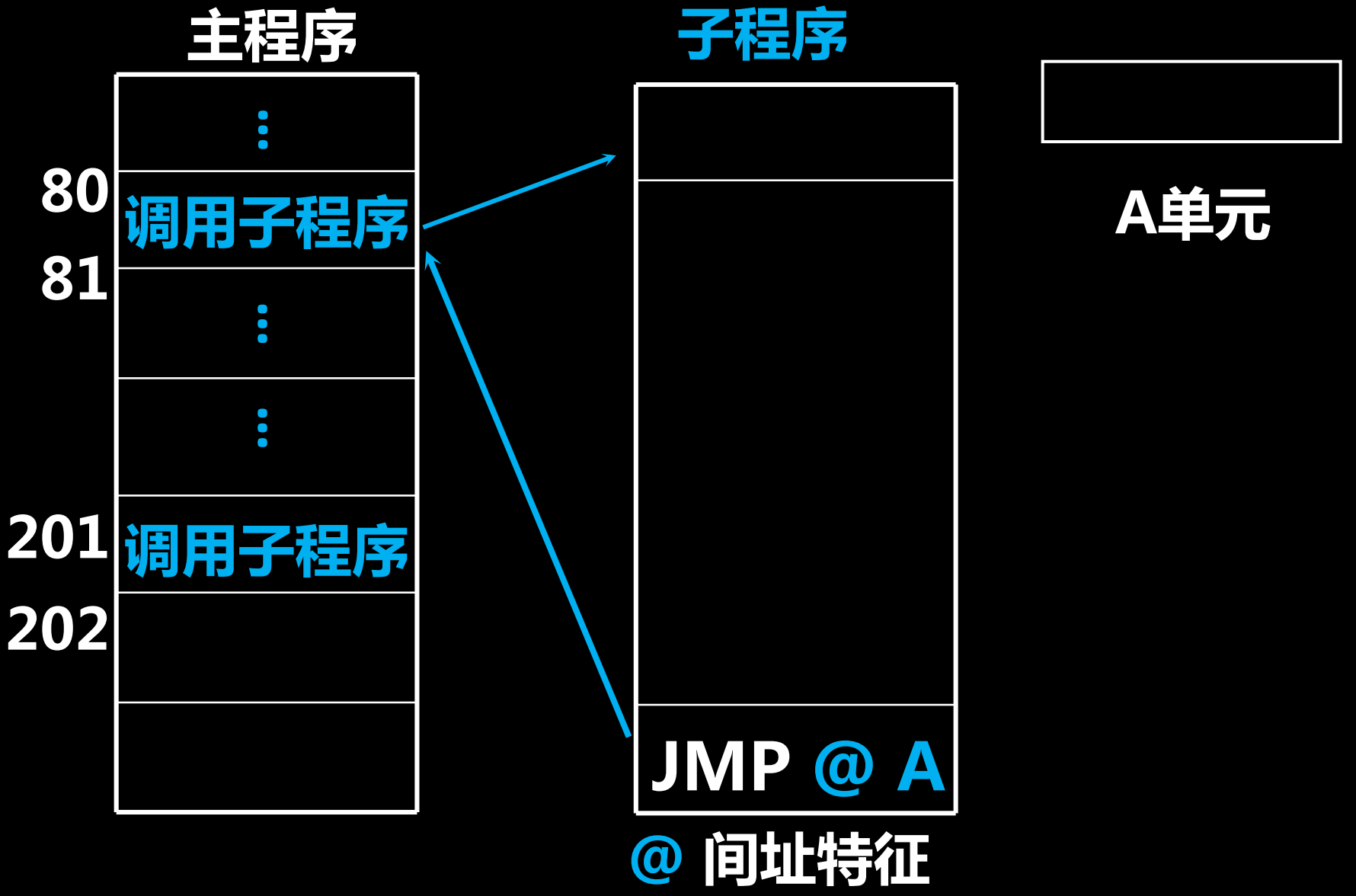
间接寻址编程举例



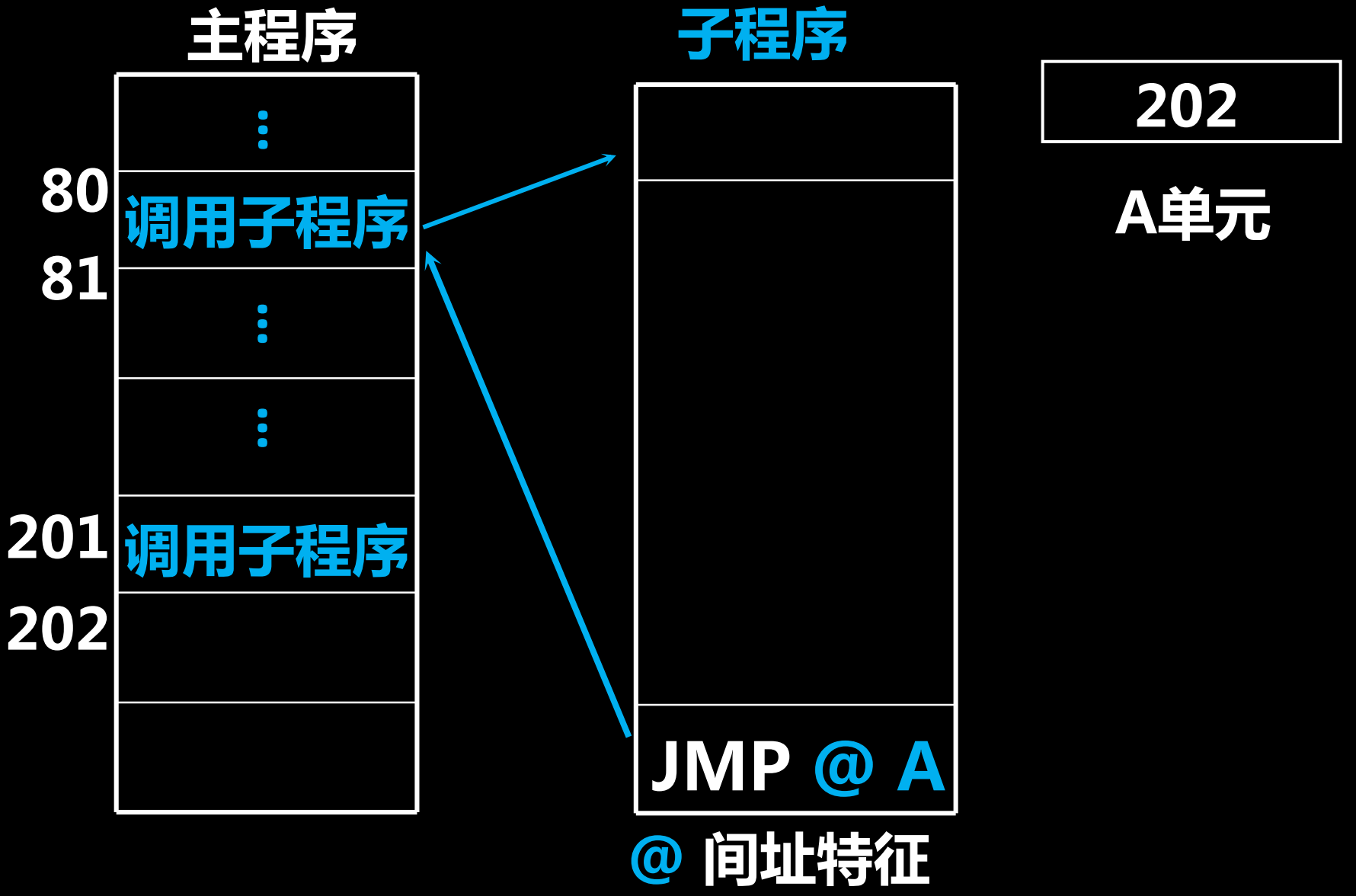
间接寻址编程举例



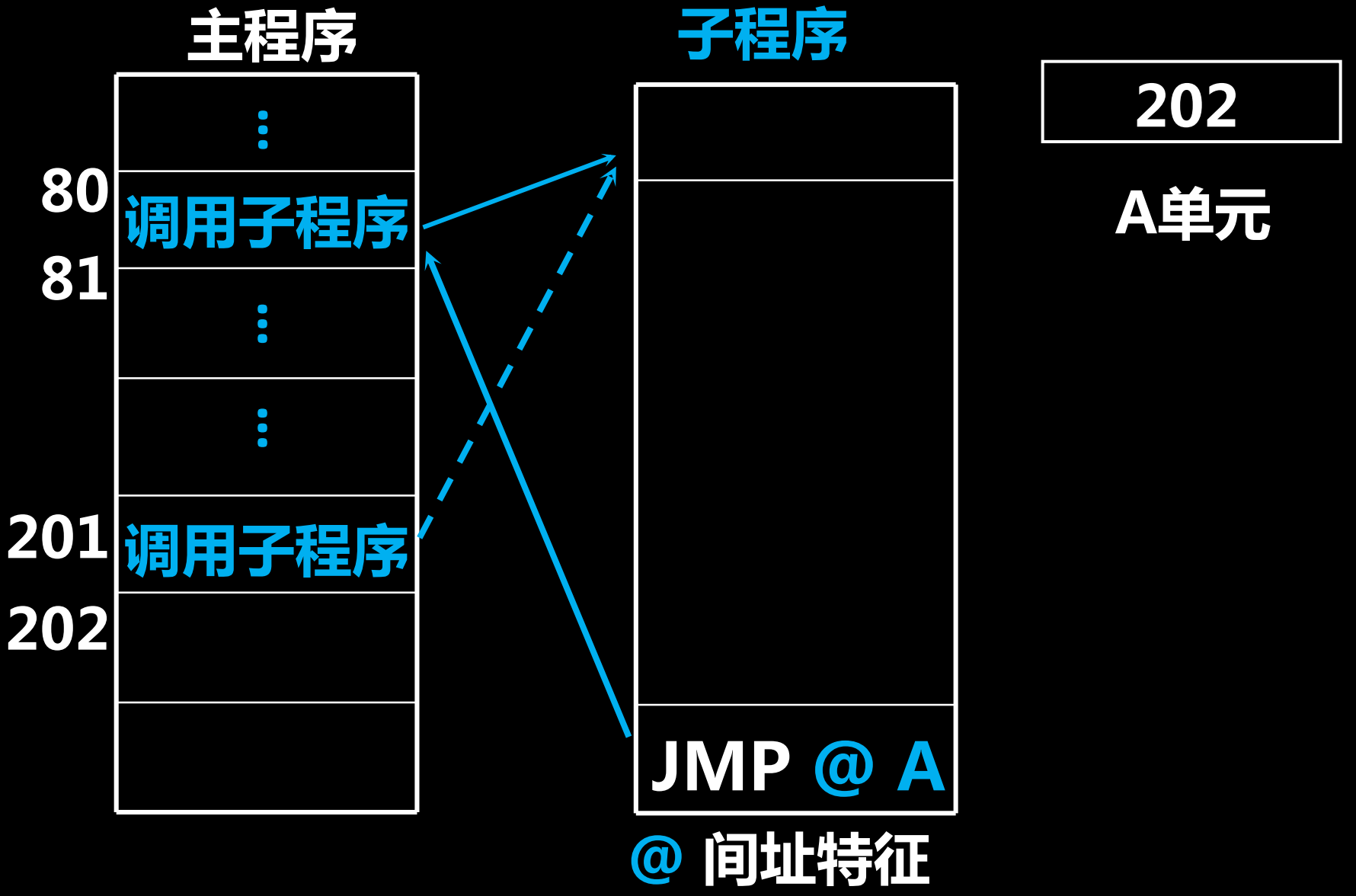
间接寻址编程举例



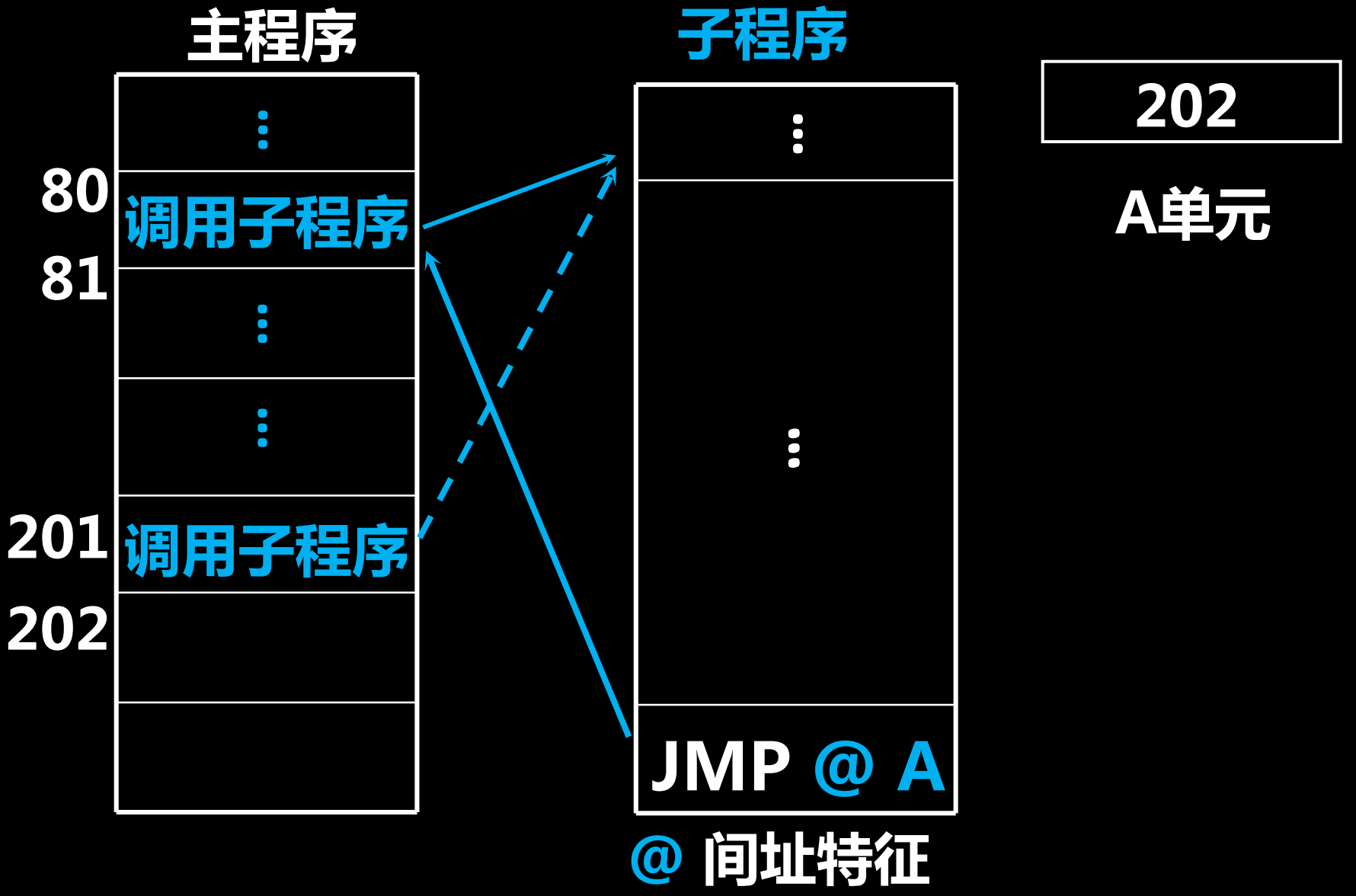
间接寻址编程举例



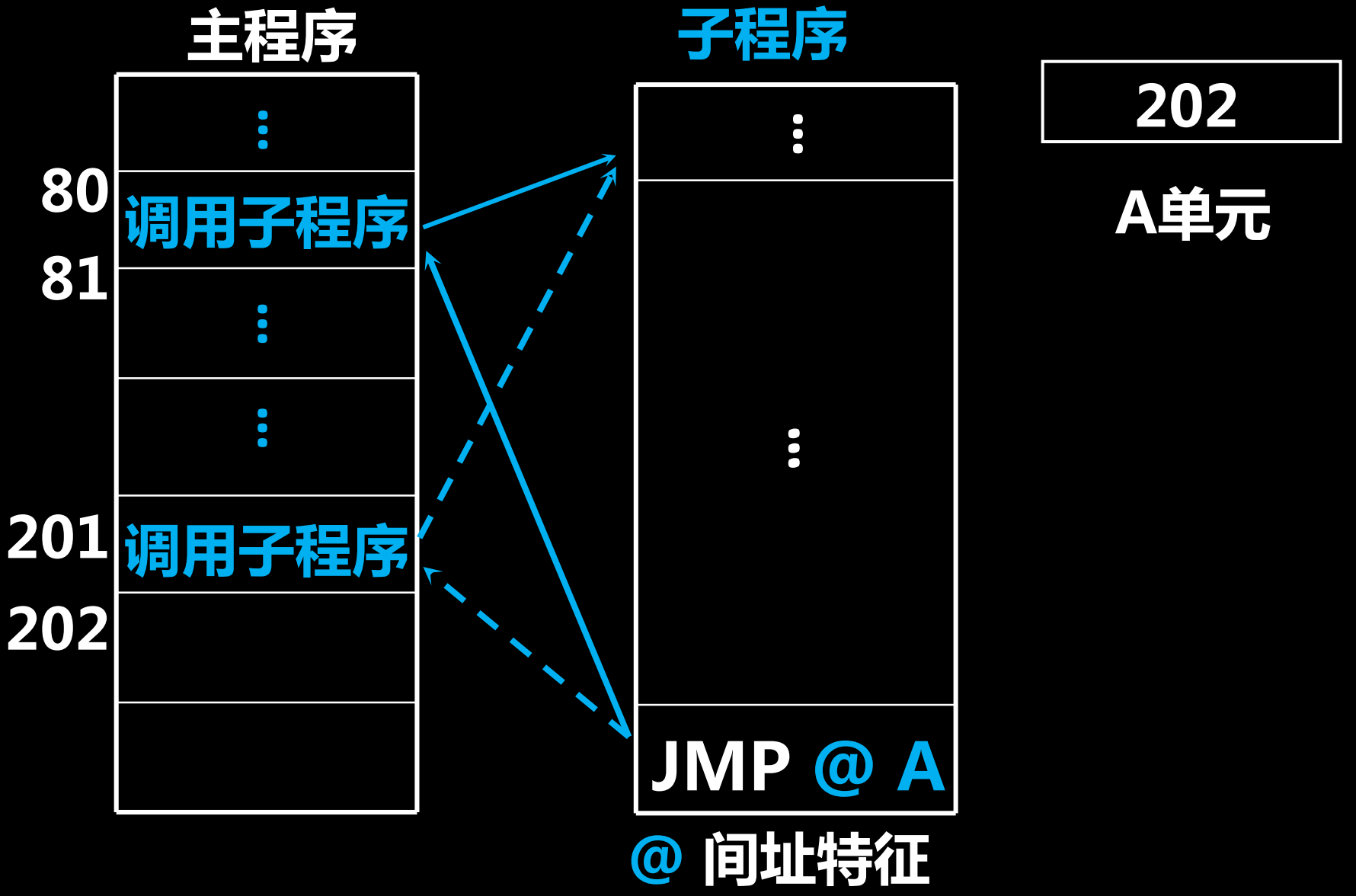
间接寻址编程举例



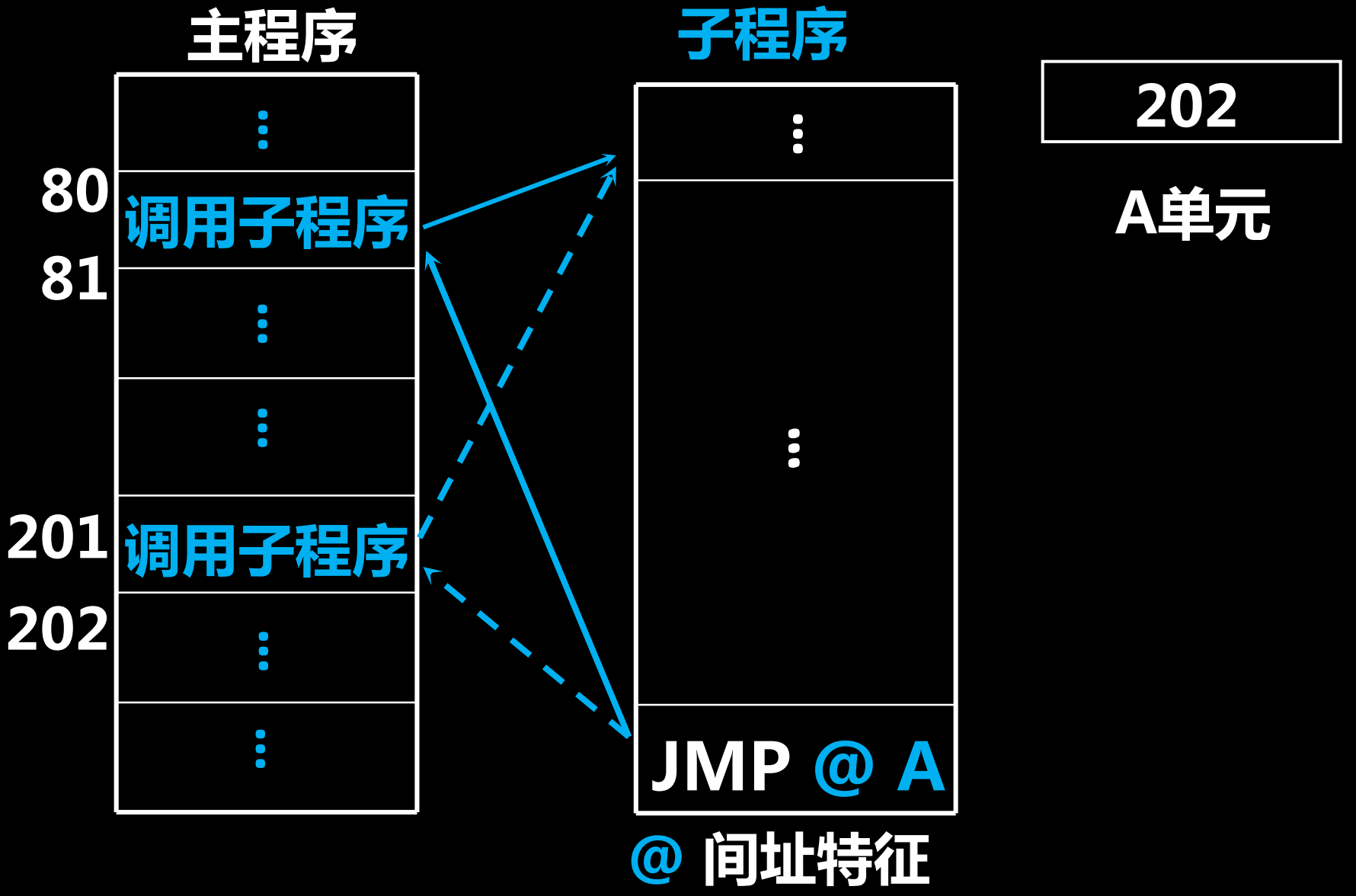
间接寻址编程举例



间接寻址编程举例

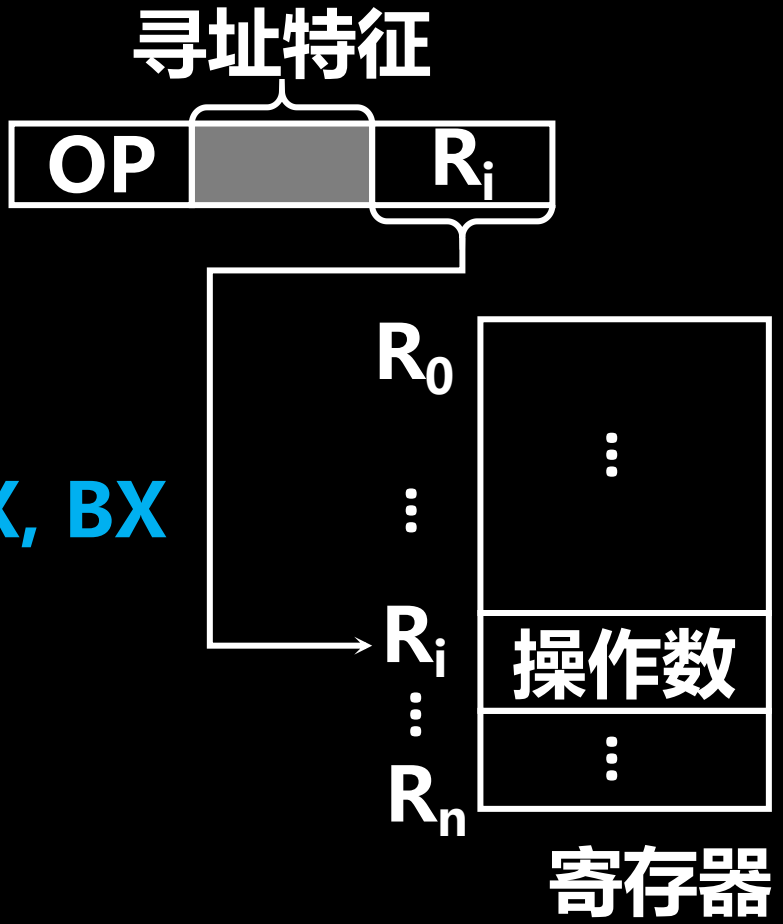


间接寻址编程举例



5. 寄存器寻址

指令中的形式地址直接指出寄存器的编号，操作数存储于寄存器中。



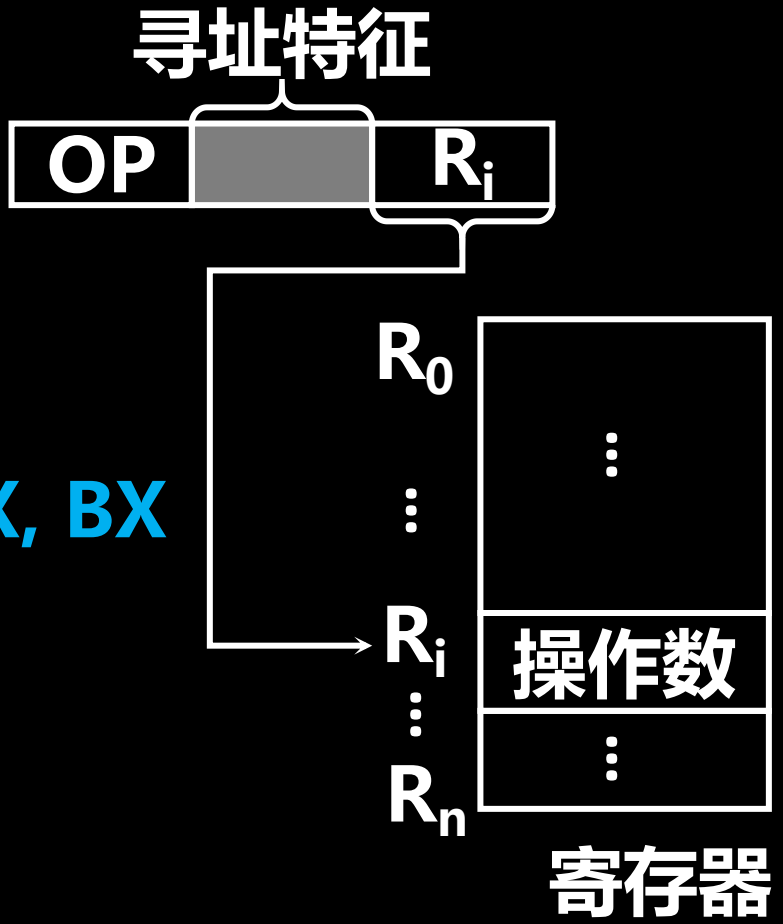
$EA = R_i$

有效地址即为寄存器编号

MOV AX, BX

5. 寄存器寻址

指令中的形式地址直接指出寄存器的编号，操作数存储于寄存器中。



MOV AX, BX

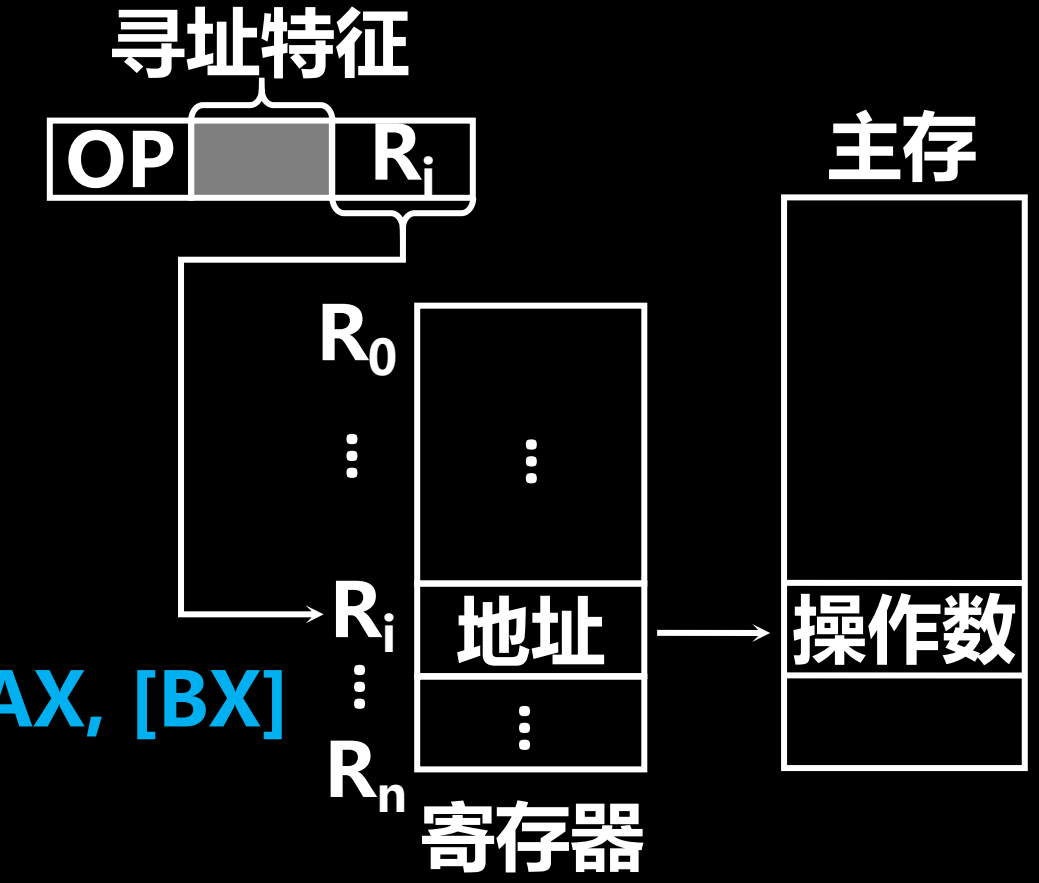
$EA = R_i$

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

6. 寄存器间接寻址

指令中的形式地址为寄存器的编号，寄存器的内容是操作数的有效地址。

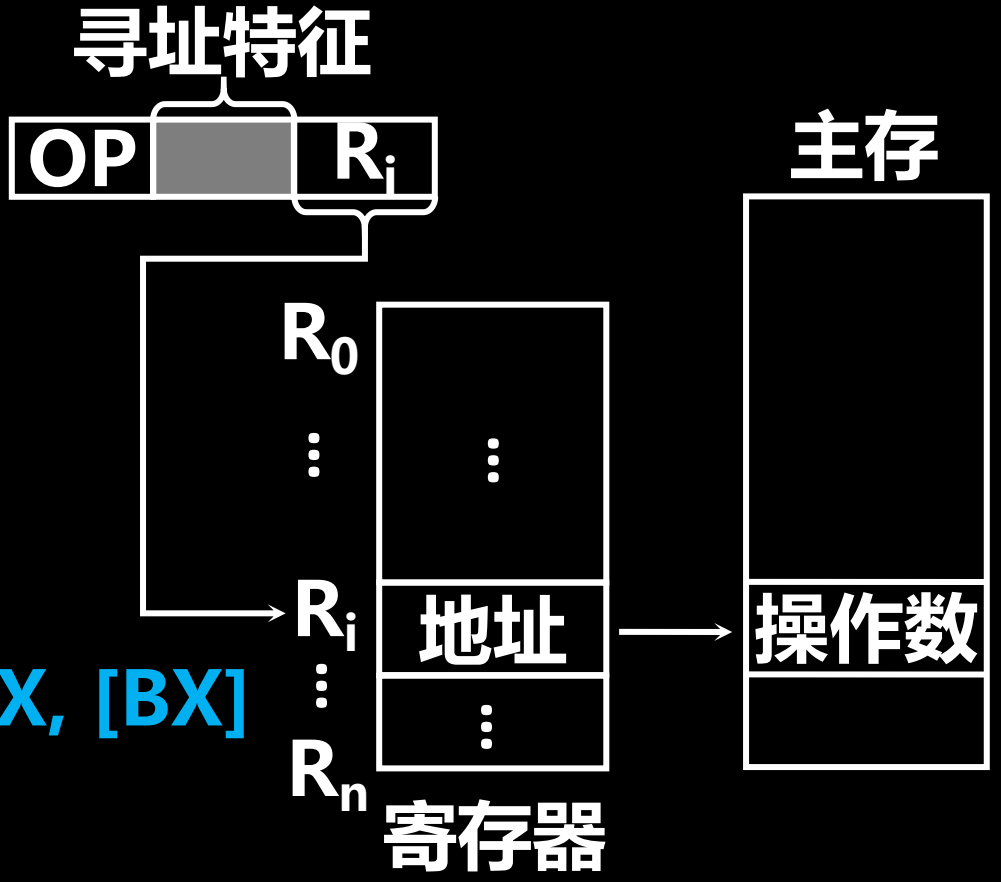


EA = (R_i)

有效地址在寄存器中

6. 寄存器间接寻址

指令中的形式地址为寄存器的编号，寄存器的内容是操作数的有效地址。



$EA = (R_i)$

MOV AX, [BX]

有效地址在寄存器中，操作数在存储器中，执行阶段访存便于编制循环程序

推荐阅读：8086寄存器的功能

8086共有14个16位寄存器

AH	AL	累加器
BH	BL	基址寄存器
CH	CL	计数器
DH	DL	数据寄存器

FLAGS	程序状态字
IP	程序计数器

SI	源变址寄存器
DI	目的变址寄存器
BP	基址指针
SP	堆栈指针

CS	代码段寄存器
SS	堆栈段寄存器
DS	数据段寄存器
ES	附加段寄存器

寻址方式（2）

大连理工大学 赖晓晨

龙芯处理器

- 2001年5月：龙芯课题组正式成立
- 2001年8月：龙芯1号设计与验证系统成功启动Linux操作系统
- 2004年9月28日：龙芯2C流片成功
- 2012年10月：八核32纳米龙芯3B1500流片成功（1.5GHz主频，支持向量运算加速，最高峰值计算能力达到192GFLOPS，高性能功耗比）

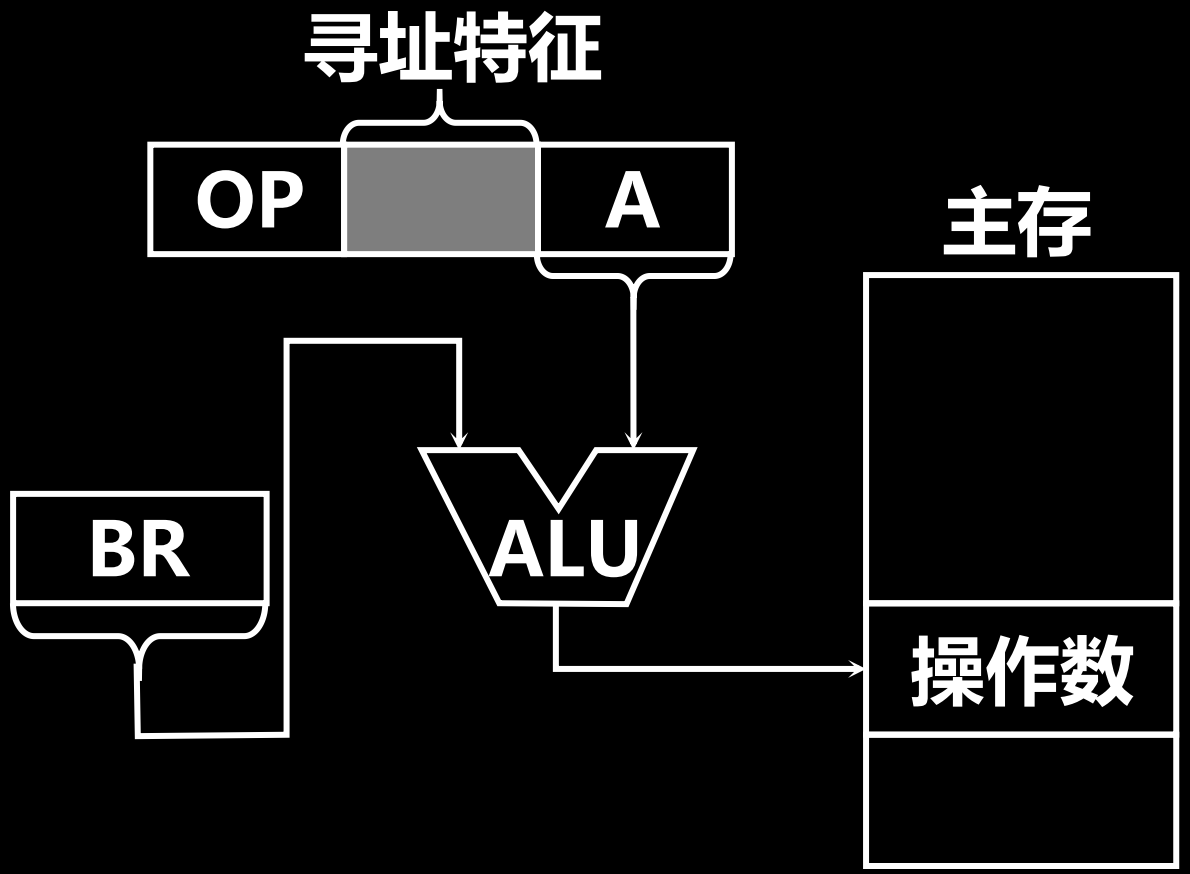
7. 基址寻址

指令中的形式地址与基址寄存器内容之和为有效地址

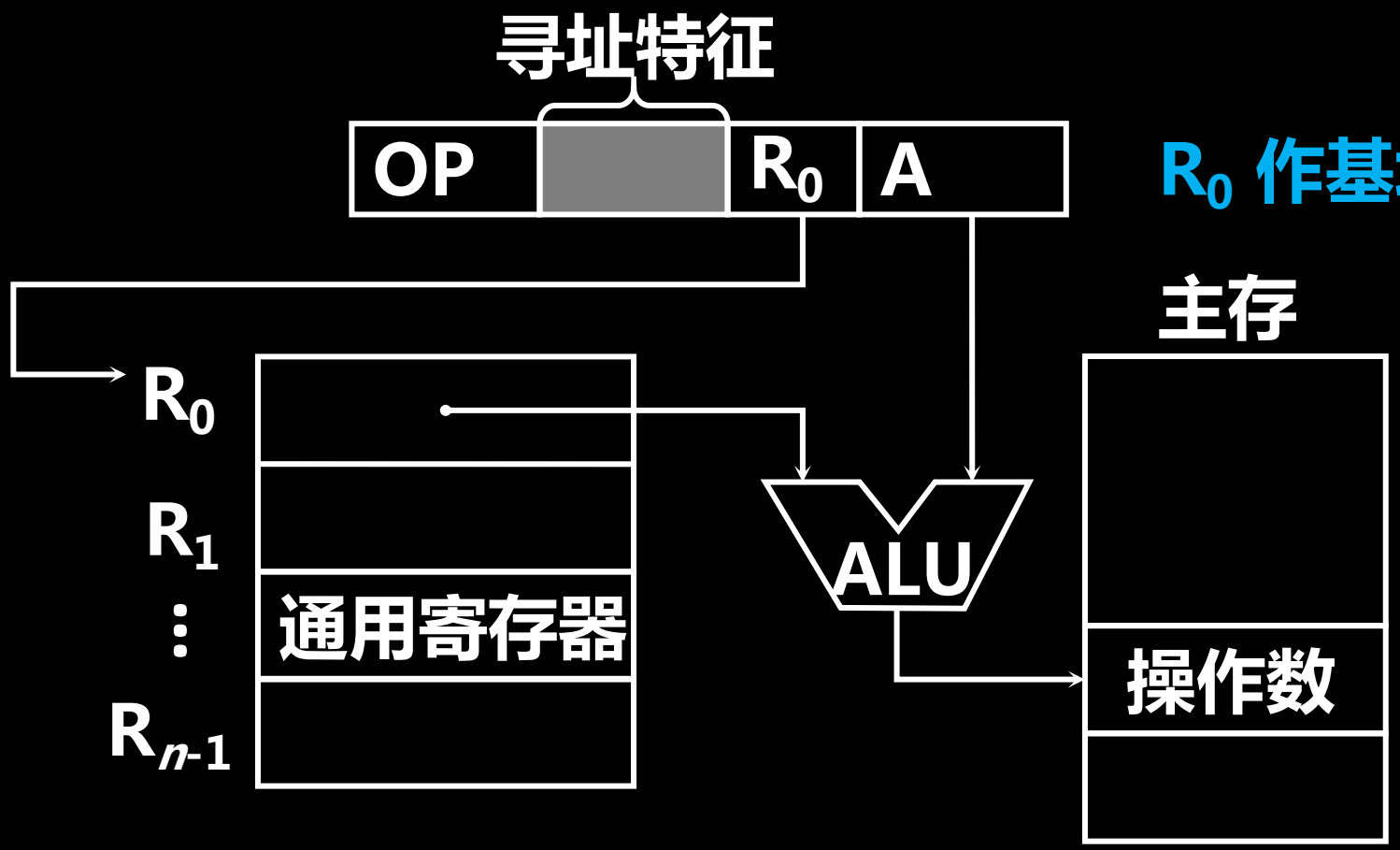
- 采用专用寄存器作为基址寄存器（隐式）
- 采用通用寄存器作为基址寄存器（显式）

(1) 专用寄存器基址寻址

EA = (BR) + A　　BR 为基址寄存器



(2) 通用寄存器基址寻址



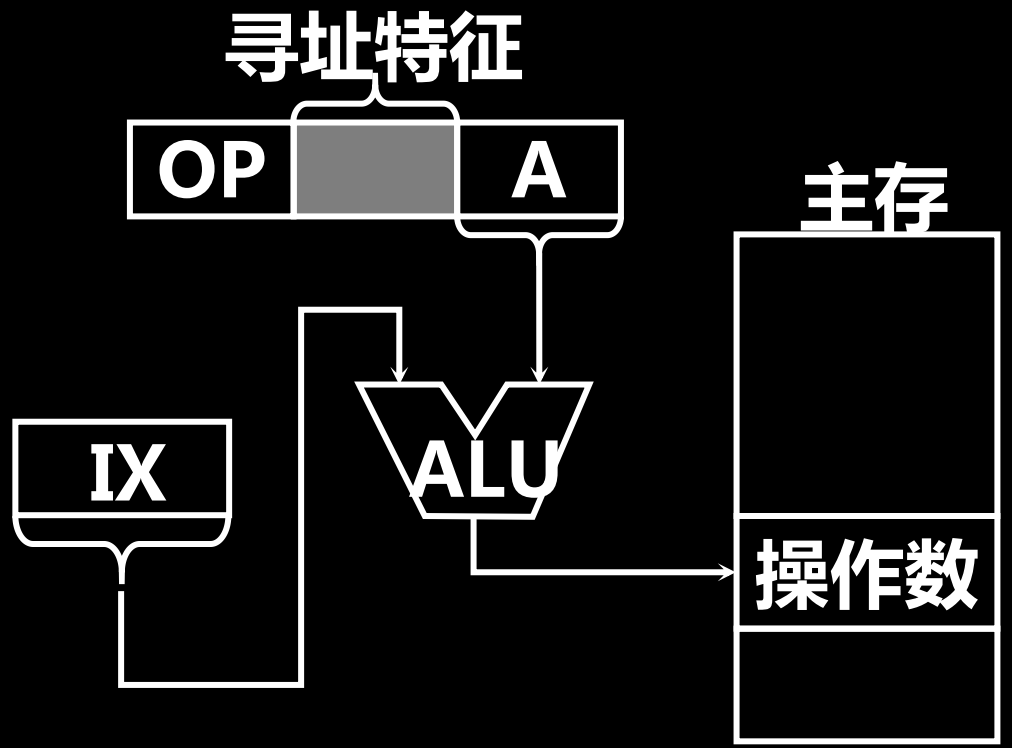
- 可指定由哪个通用寄存器作为基址寄存器
- 在程序的执行过程中R_0 内容不变，形式地址A可变。

基址寻址的作用

- 可扩大寻址范围
- 有利于多道程序
- 基址寄存器内容由操作系统或管理程序确定

8. 变址寻址

指令中的形式地址与变址寄存器内容之和为有效地址。



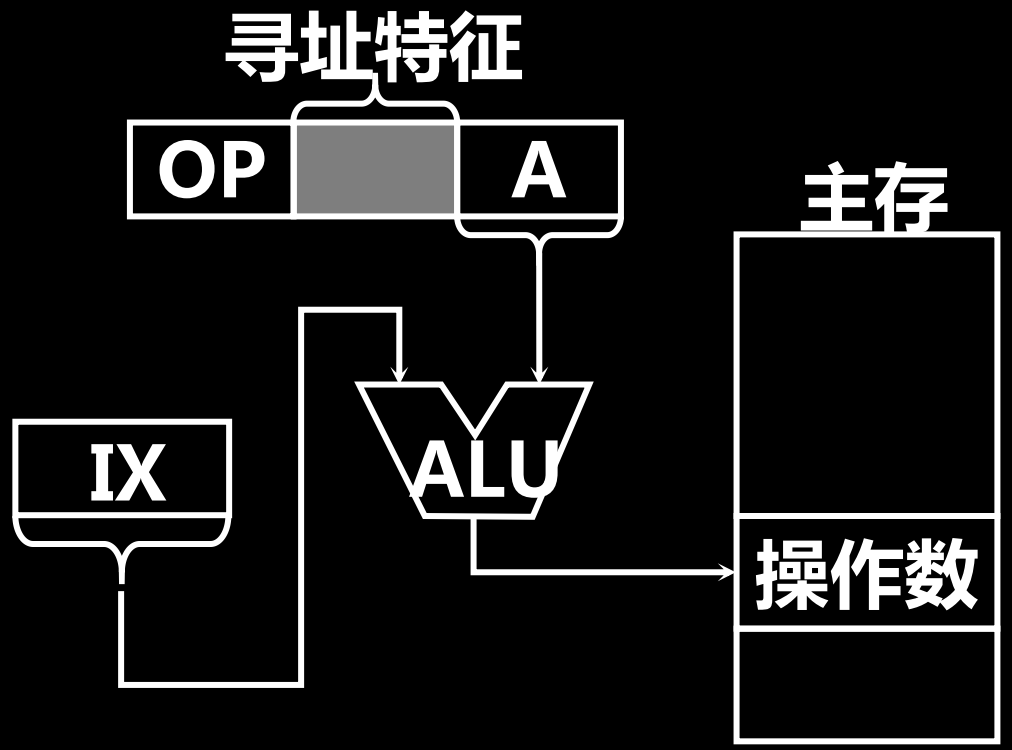
EA = (IX) + A

IX 为变址寄存器

通用寄存器也可以作为变址寄存器

8. 变址寻址

指令中的形式地址与变址寄存器内容之和为有效地址。



EA = (IX) + A

可以扩大寻址范围
IX的内容由用户指定
在程序执行过程中，IX内容可变，形式地址A不变
便于处理数组问题

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

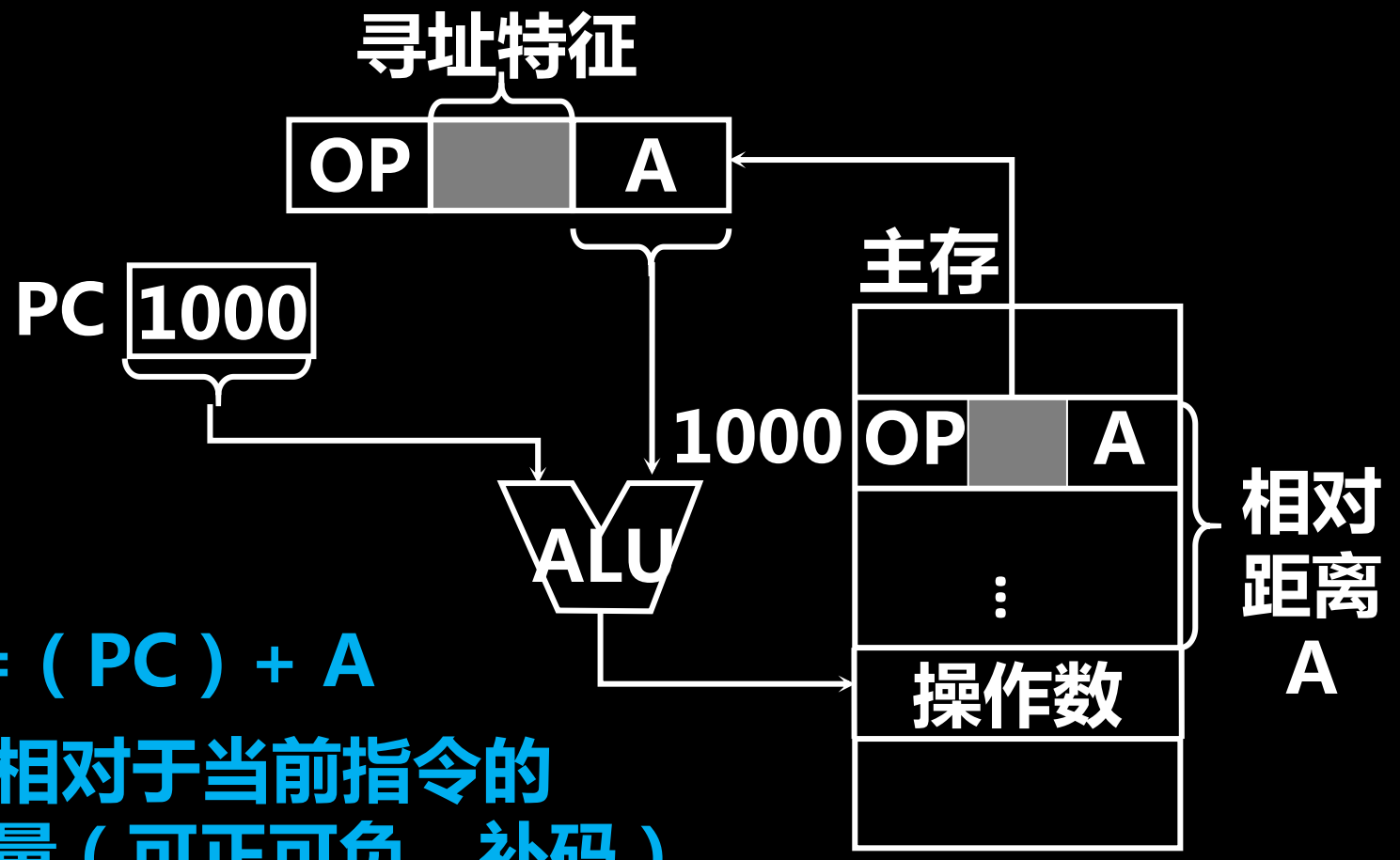
变址寻址

```
      LDA   # 0
      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

9. 相对寻址

有效地址为程序计数器PC的值与形式地址之和

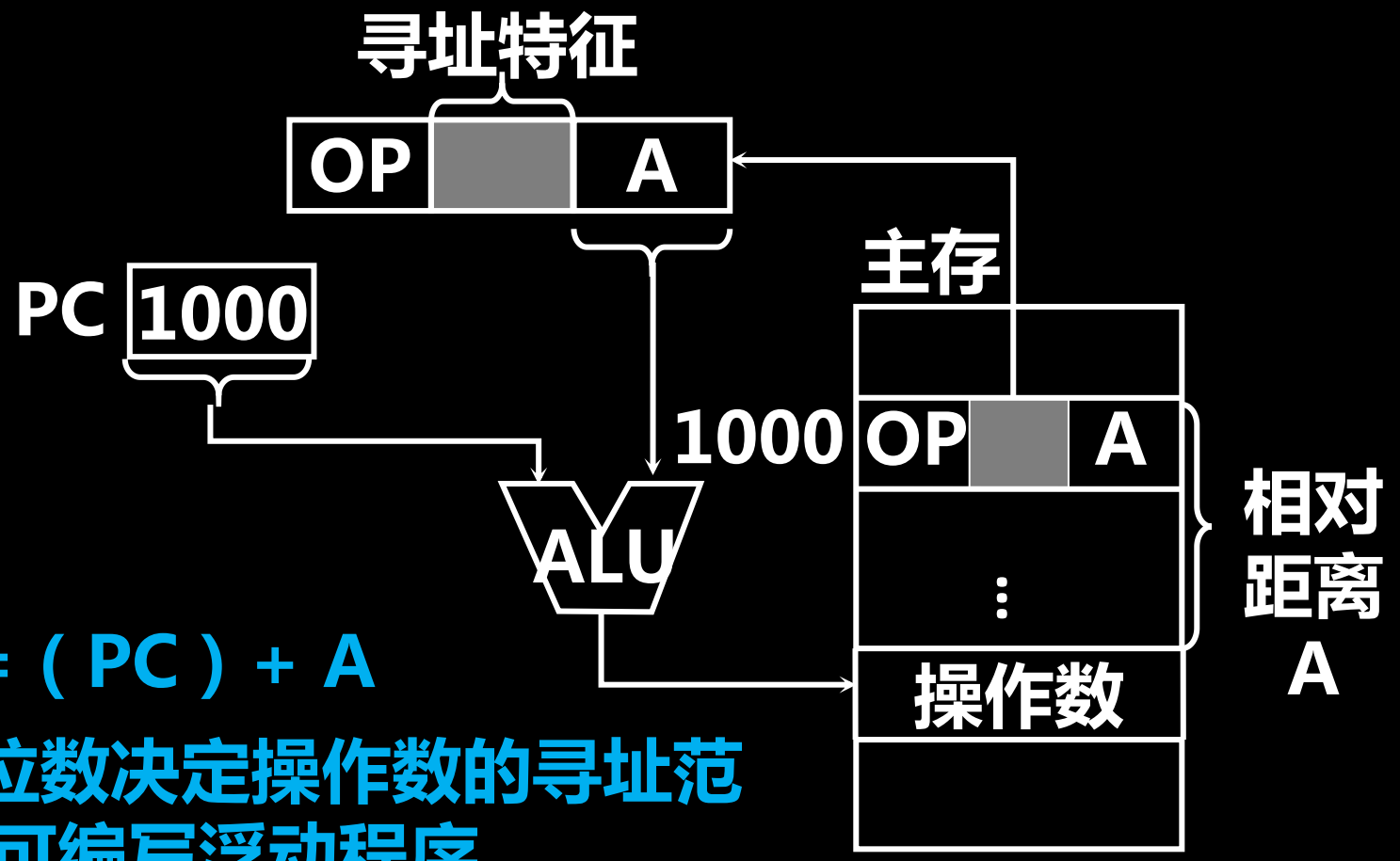


EA = (PC) + A

A 是相对于当前指令的
位移量（可正可负，补码）

9. 相对寻址

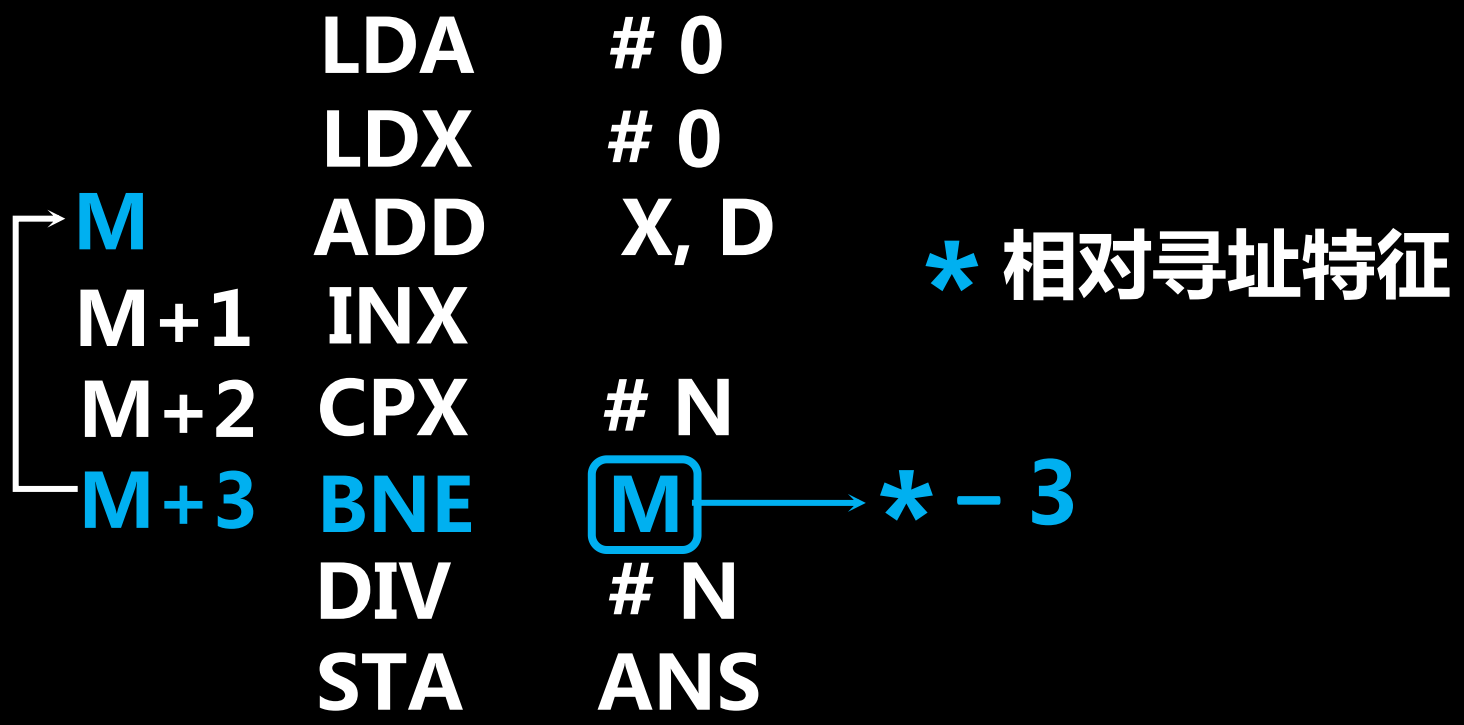
有效地址为程序计数器PC的值与形式地址之和



EA = (PC) + A

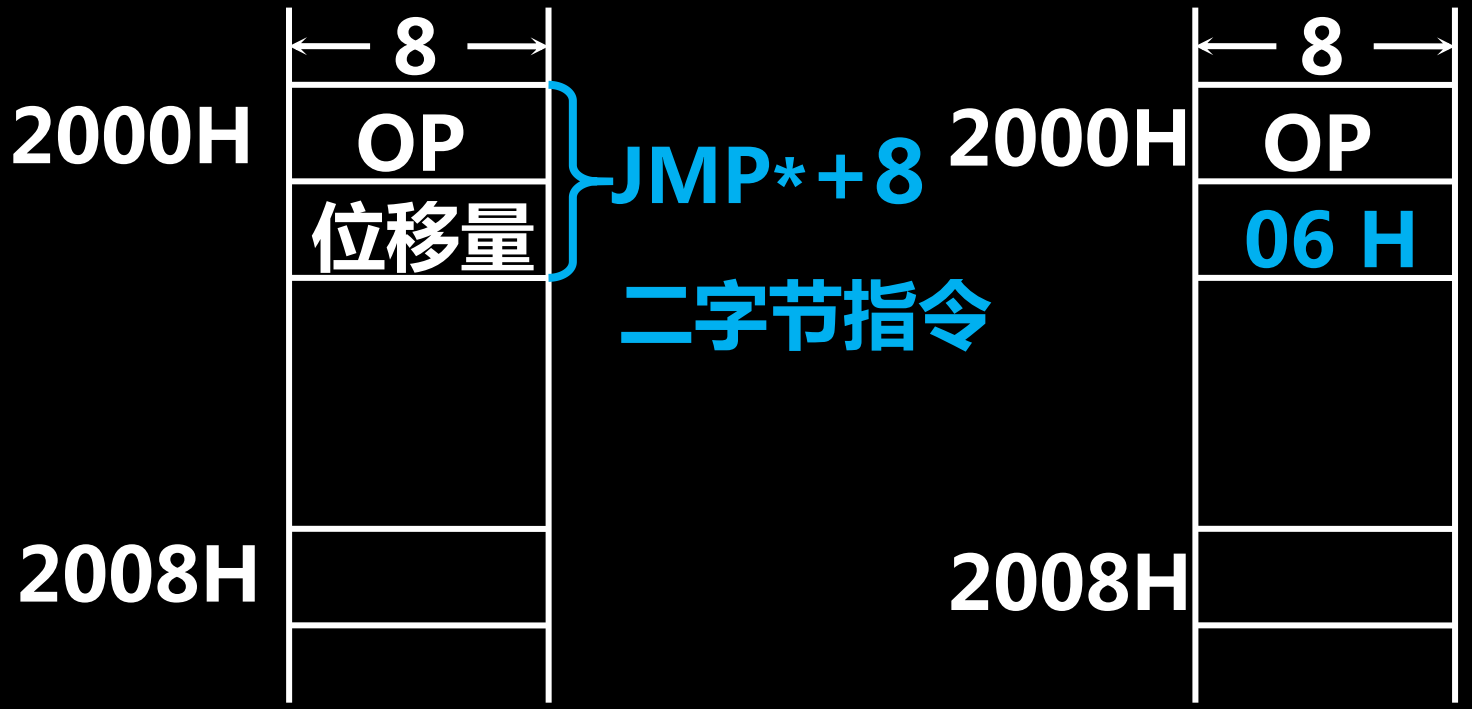
A的位数决定操作数的寻址范围，可编写浮动程序

相对寻址举例



采用相对寻址之后，程序可存放于内存任意位置，有利于程序浮动。

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，求位移量应为多少。

因为：取出 JMP*+8后 PC=2002H

故：JMP*+8 指令 的第二字节为

2008H-2002H=06H

10. 堆栈寻址

(1) 堆栈的运行方式：

后进先出、先进后出

(2) 堆栈的种类：

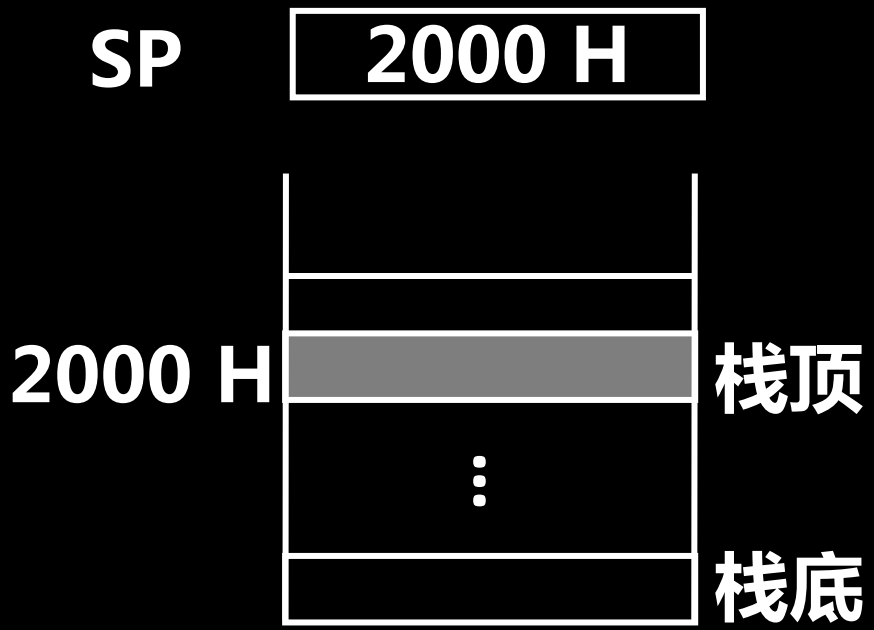
① 硬堆栈：寄存器型（栈底浮动，栈顶不变）

② 软堆栈：存储器型（栈底不变，栈顶浮动）

堆栈工作过程

先进后出（一个入出口）

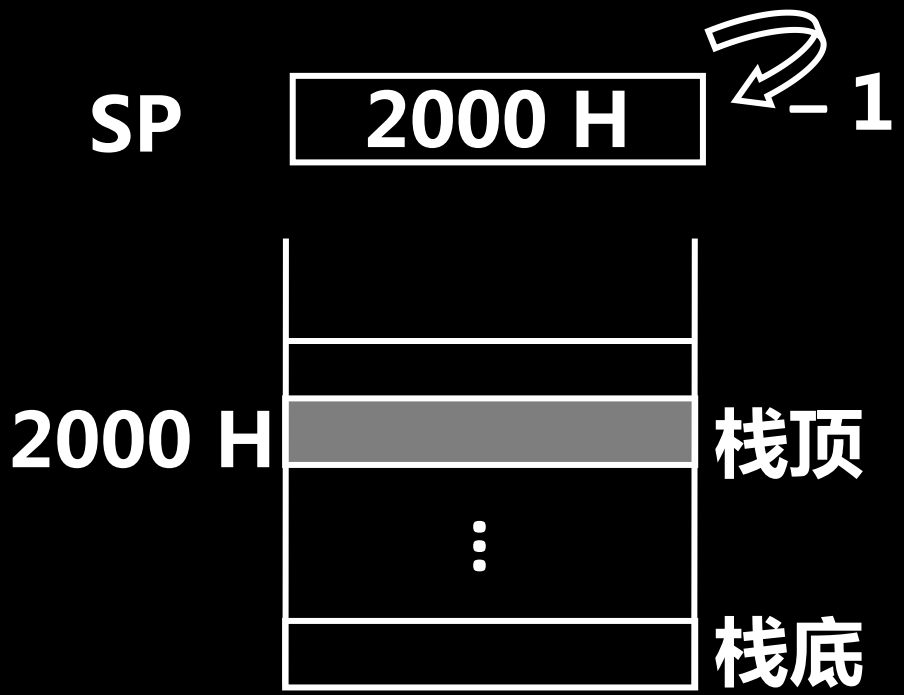
进栈　（SP）– 1　　SP



堆栈工作过程

先进后出（一个入出口）

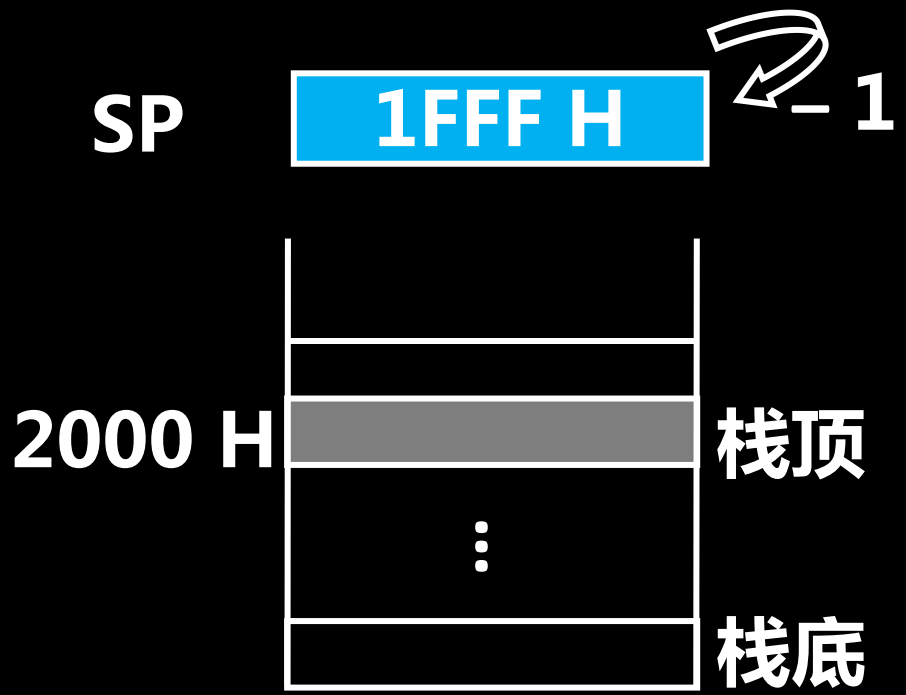
进栈　（SP）– 1　　SP



堆栈工作过程

先进后出（一个入出口）

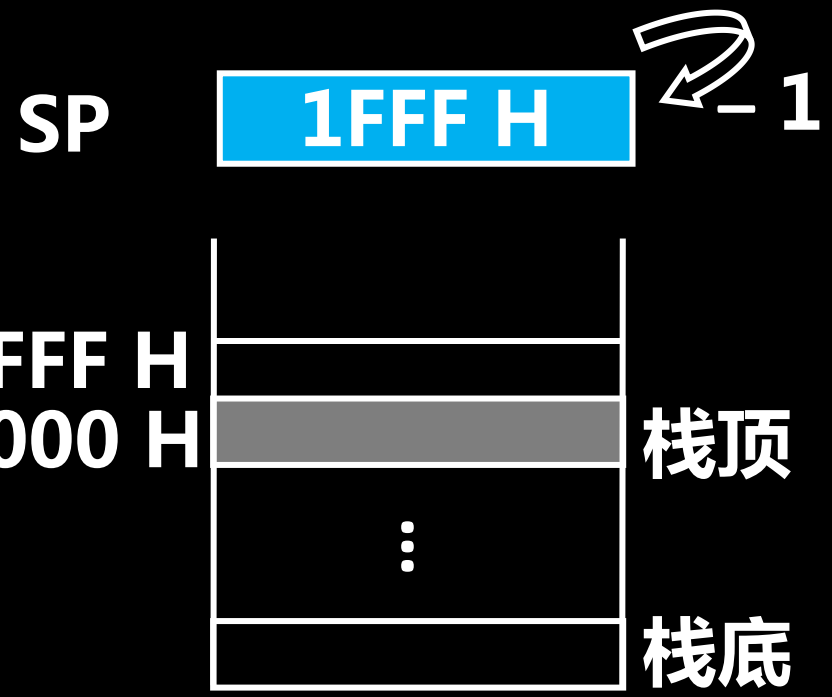
进栈　（SP）– 1　　SP



堆栈工作过程

先进后出（一个入出口）

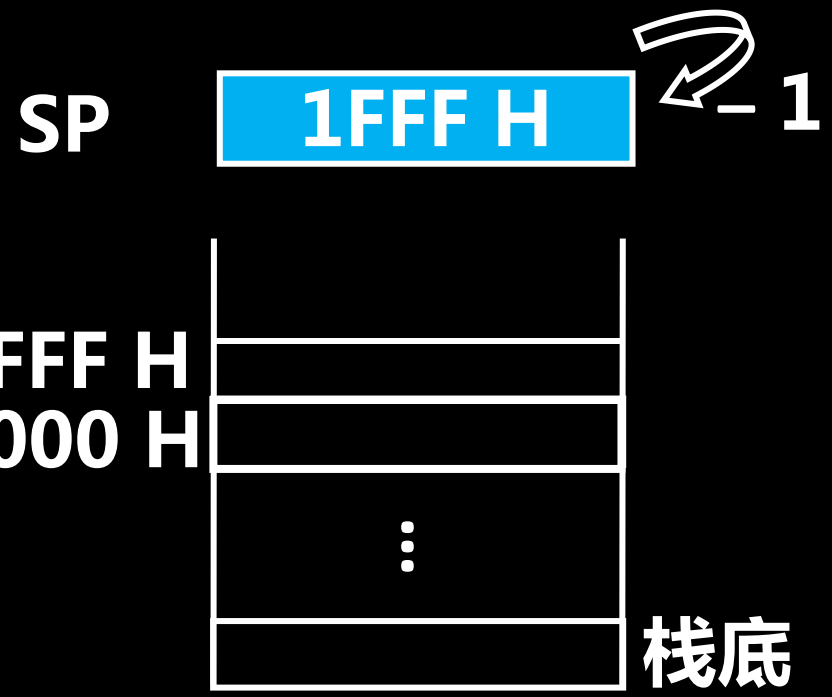
进栈　（SP）– 1　　SP



堆栈工作过程

先进后出（一个入出口）

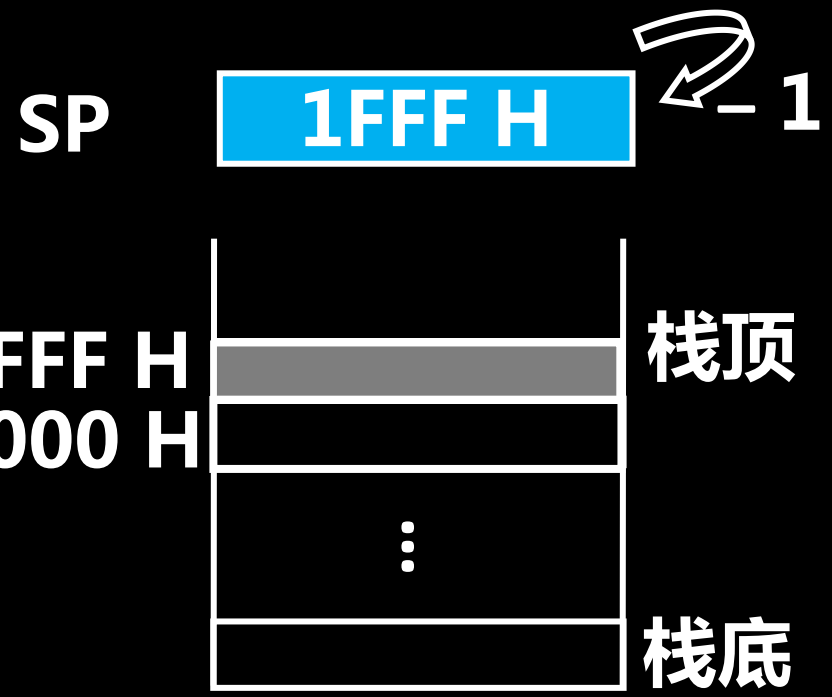
进栈　（SP）– 1　　SP



堆栈工作过程

先进后出（一个入出口）

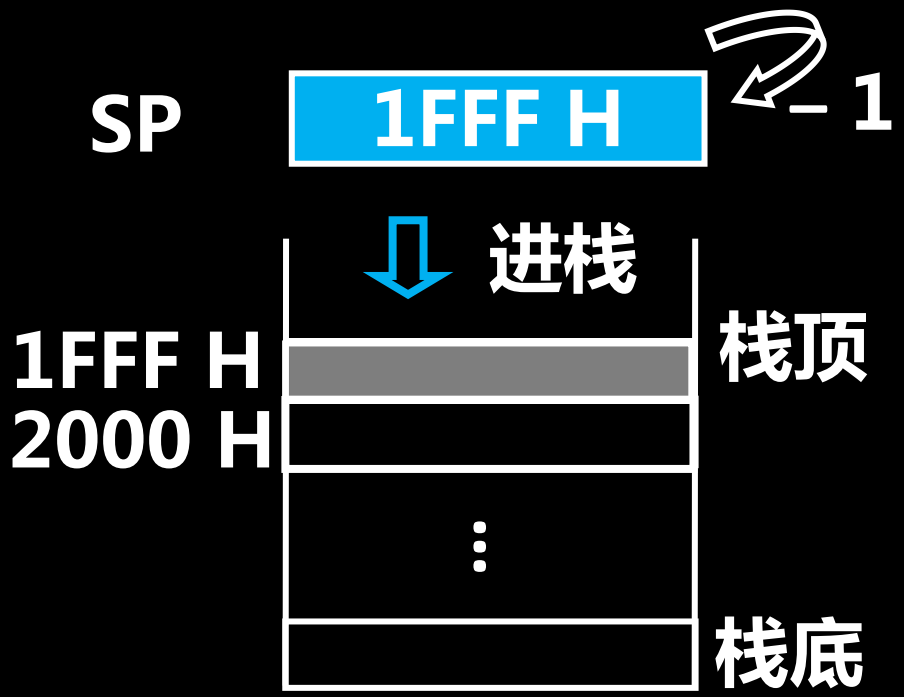
进栈　（SP）– 1　　SP



堆栈工作过程

先进后出（一个入出口）

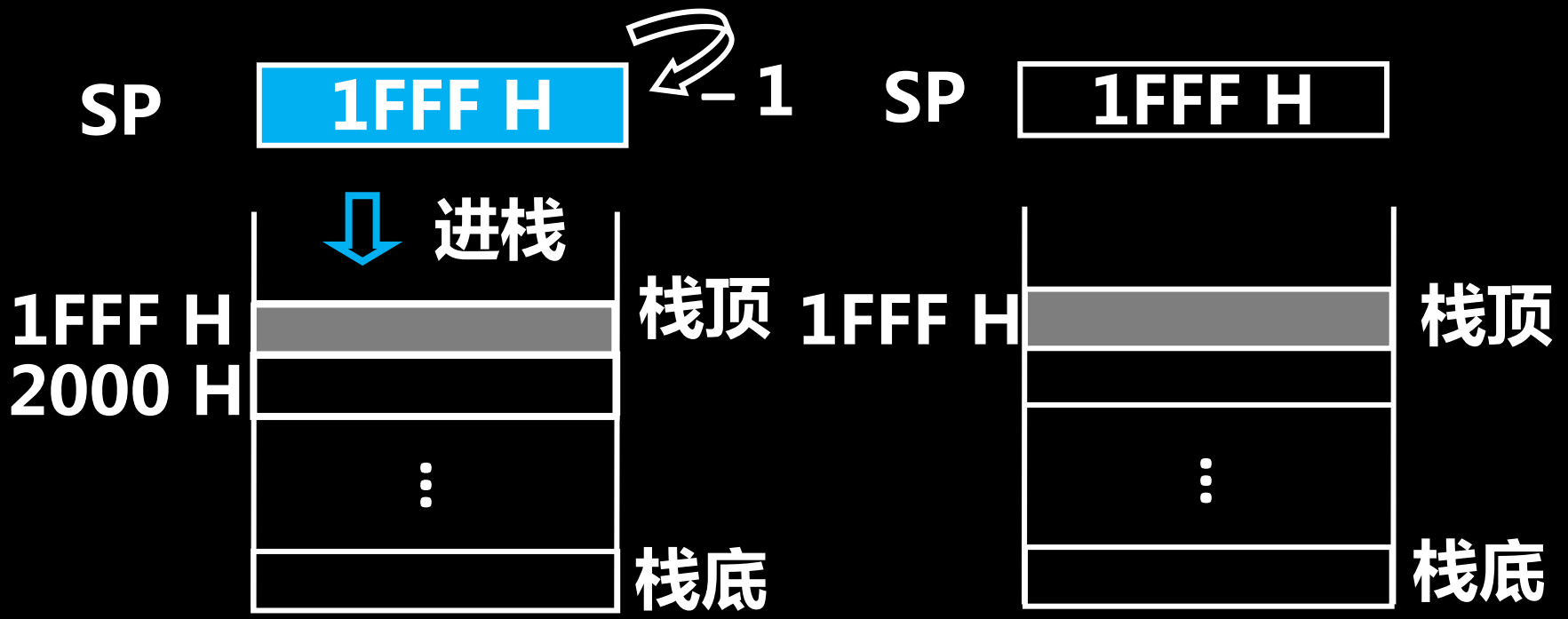
进栈 （SP）– 1 SP



堆栈工作过程

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**

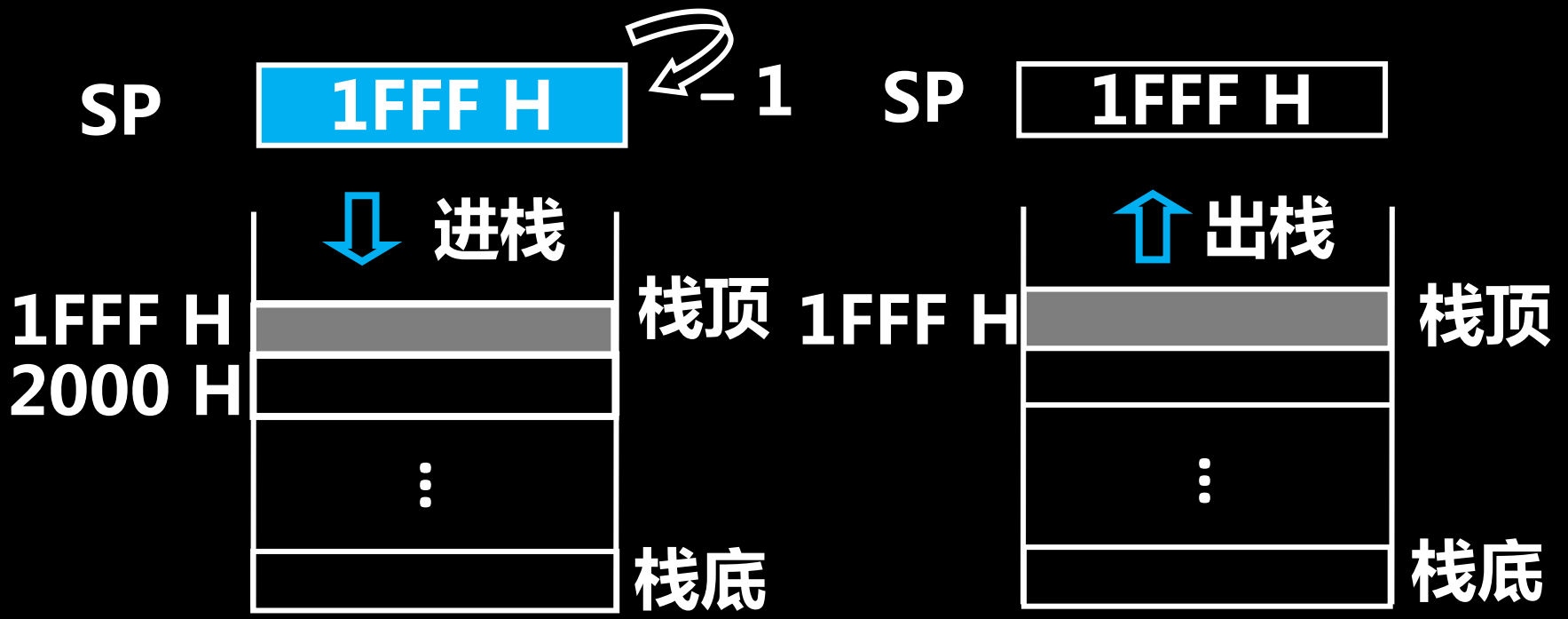


堆栈工作过程

先进后出（一个入出口）

进栈 （SP）– 1 SP

出栈 （SP）+ 1 SP



栈顶地址 由 SP 指出

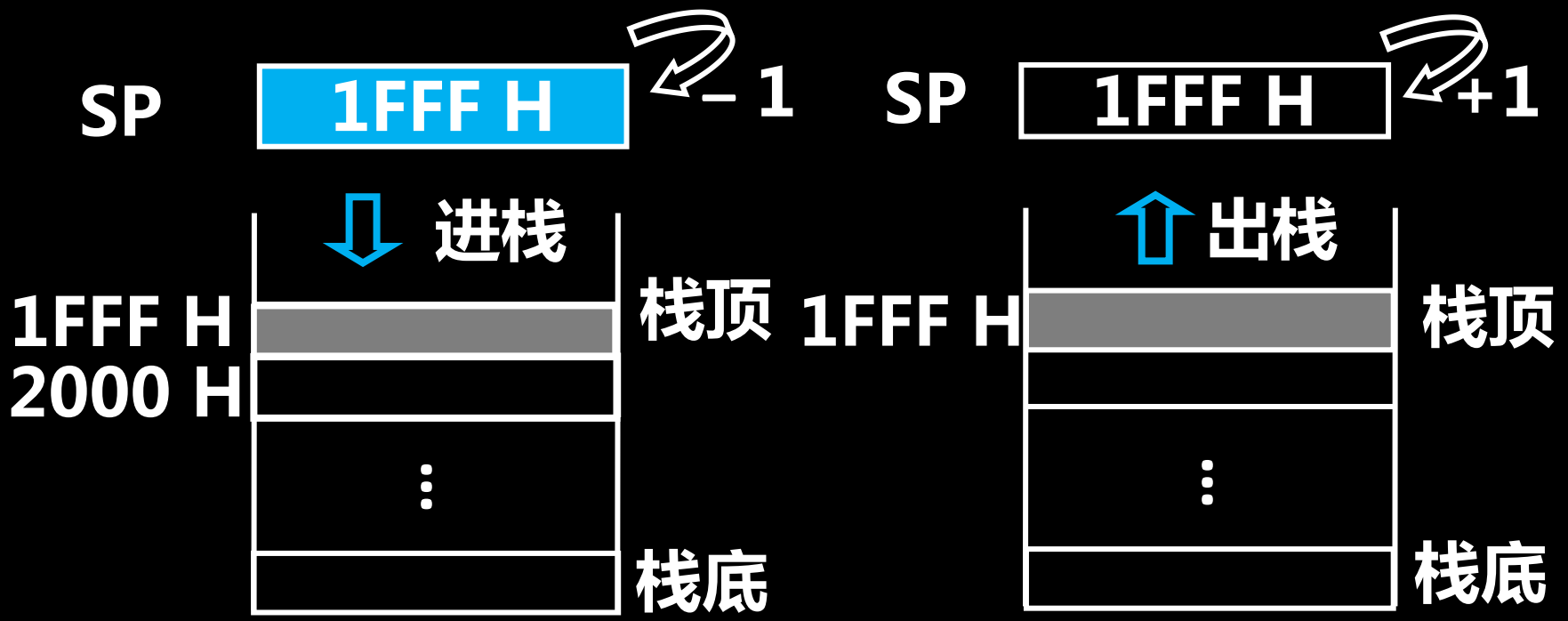
堆栈工作过程

先进后出（一个入出口）

进栈 （SP）– 1 SP

出栈 （SP）+ 1 SP

栈顶地址 由 SP 指出



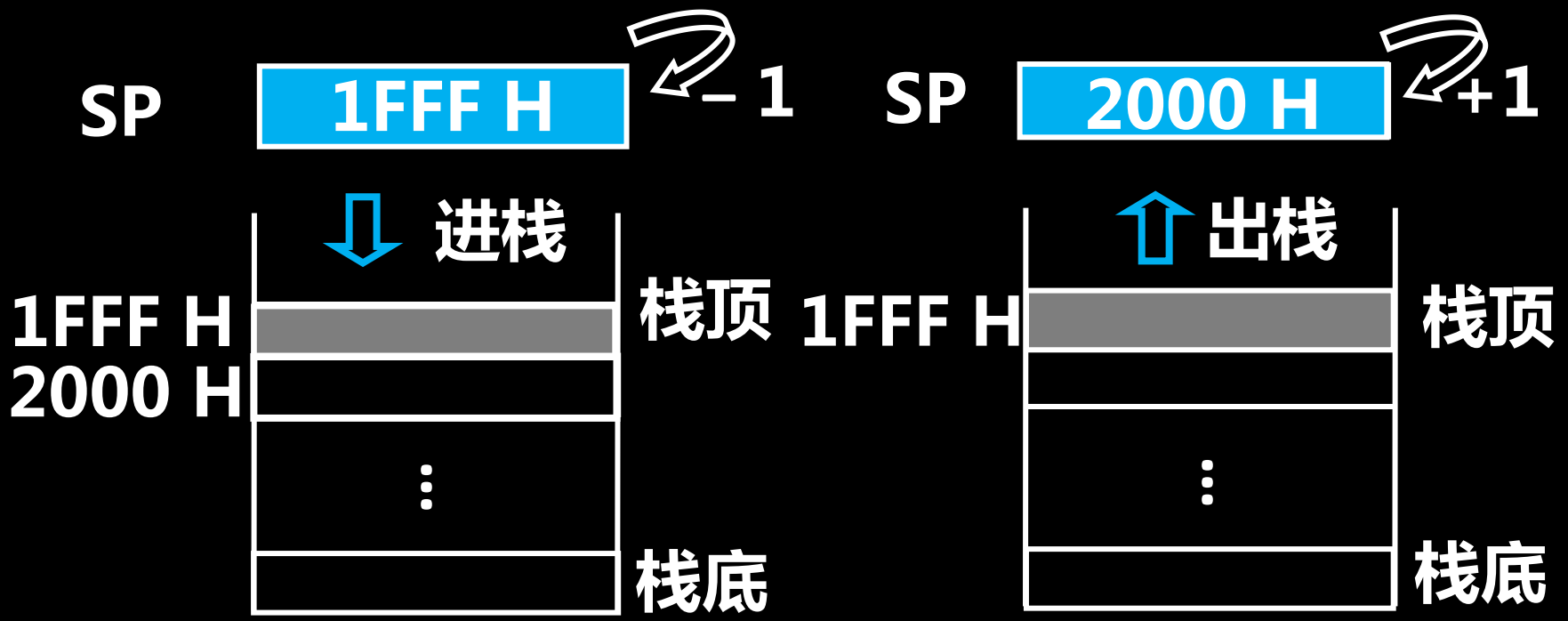
堆栈工作过程

先进后出（一个入出口）

进栈 （SP）– 1 SP

出栈 （SP）+ 1 SP

栈顶地址 由 SP 指出



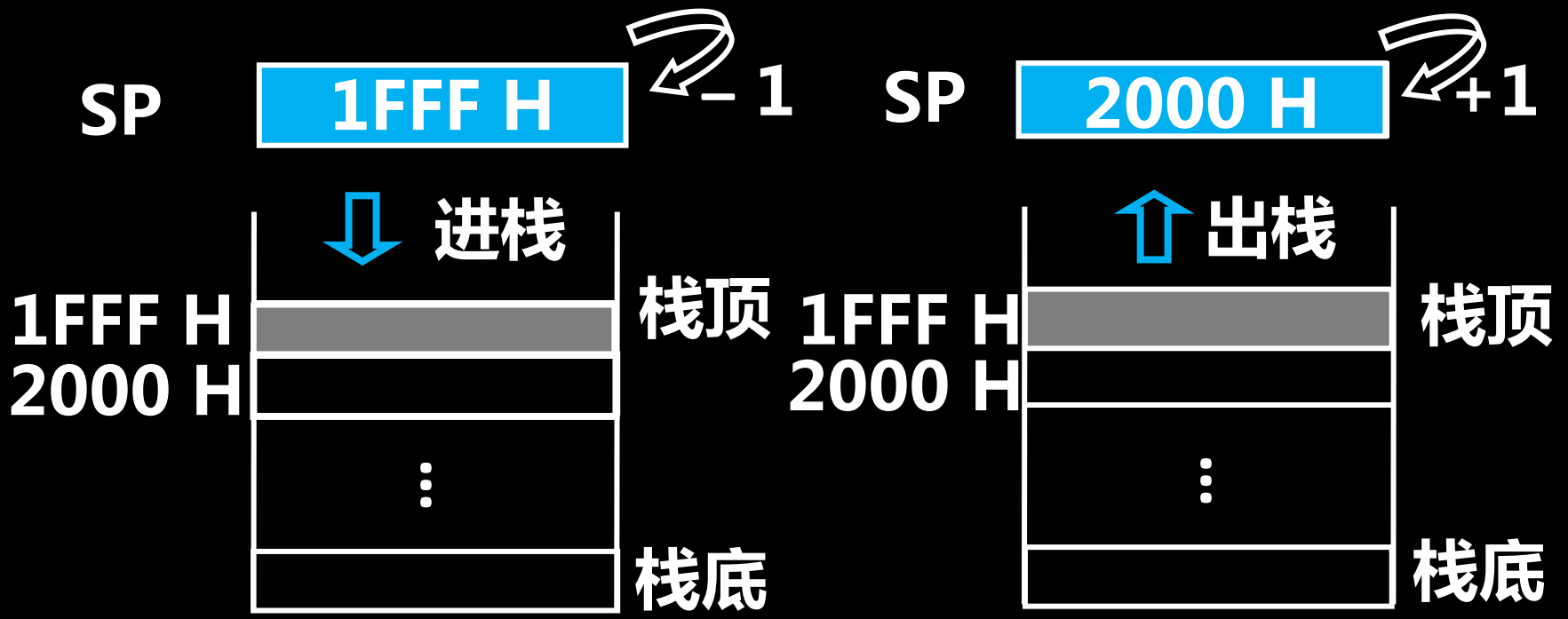
堆栈工作过程

先进后出（一个入出口）

进栈 （SP）– 1 SP

出栈 （SP）+ 1 SP

栈顶地址 由 SP 指出



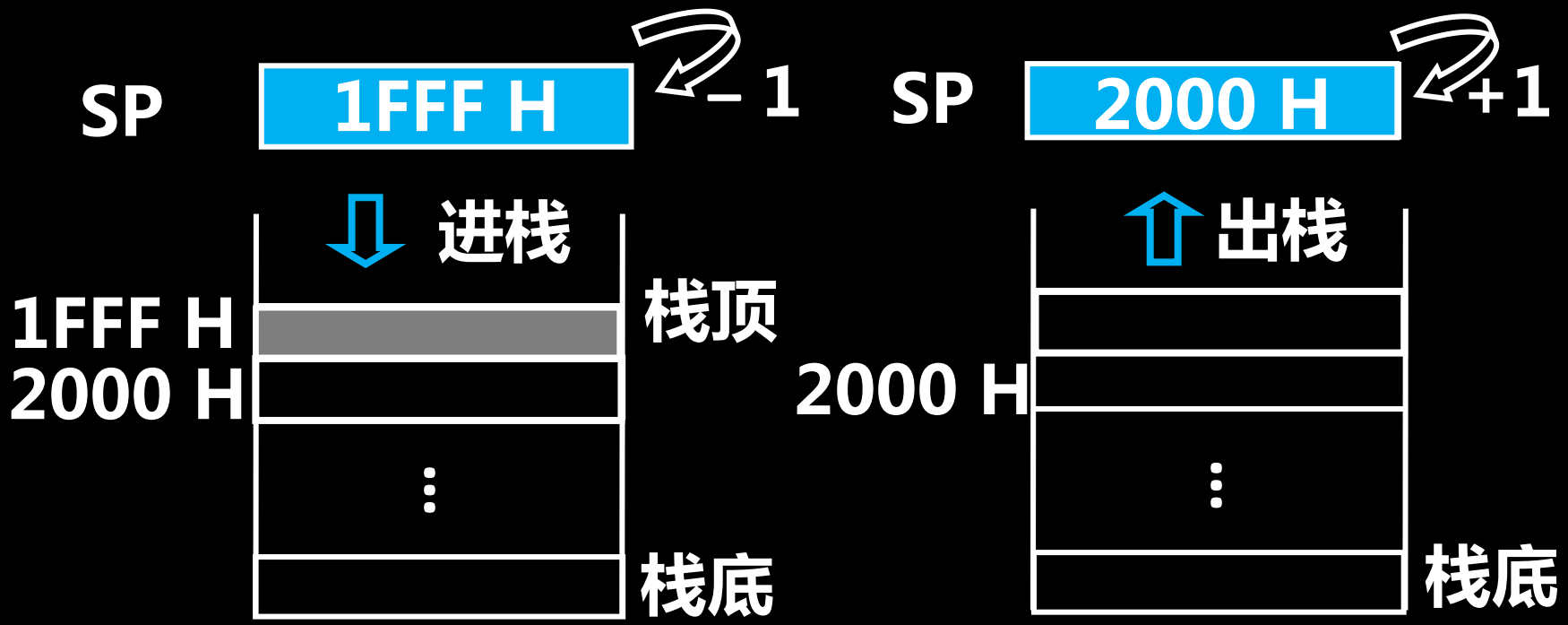
堆栈工作过程

先进后出（一个入出口）

进栈 （SP）– 1 SP

出栈 （SP）+ 1 SP

栈顶地址 由 SP 指出



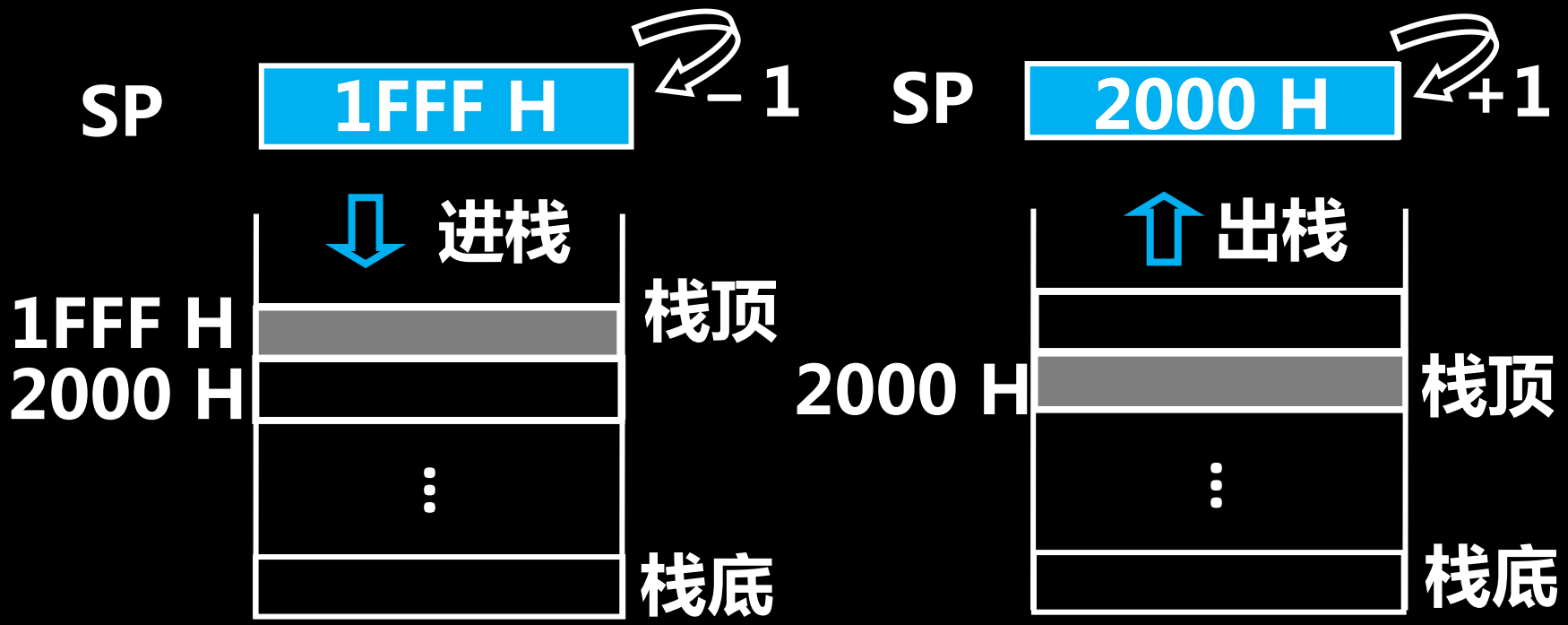
堆栈工作过程

先进后出（一个入出口）

进栈 （SP）– 1 SP

出栈 （SP）+ 1 SP

栈顶地址 由 SP 指出



堆栈指针与主存编址

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

堆栈的工作形式

1. 满栈递减
2. 满栈递增
3. 空栈递减
4. 空栈递增

推荐阅读：RISC处理器

- 选用使用频度高的一些简单指令，复杂指令用简单指令组合。
- 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。
- CPU中有多个通用寄存器。
- 采用流水技术,一个时钟周期完成一条指令。
- 采用组合逻辑实现控制器。
- 采用优化的编译程序。

计算机组织与结构

大连理工大学 赖晓晨