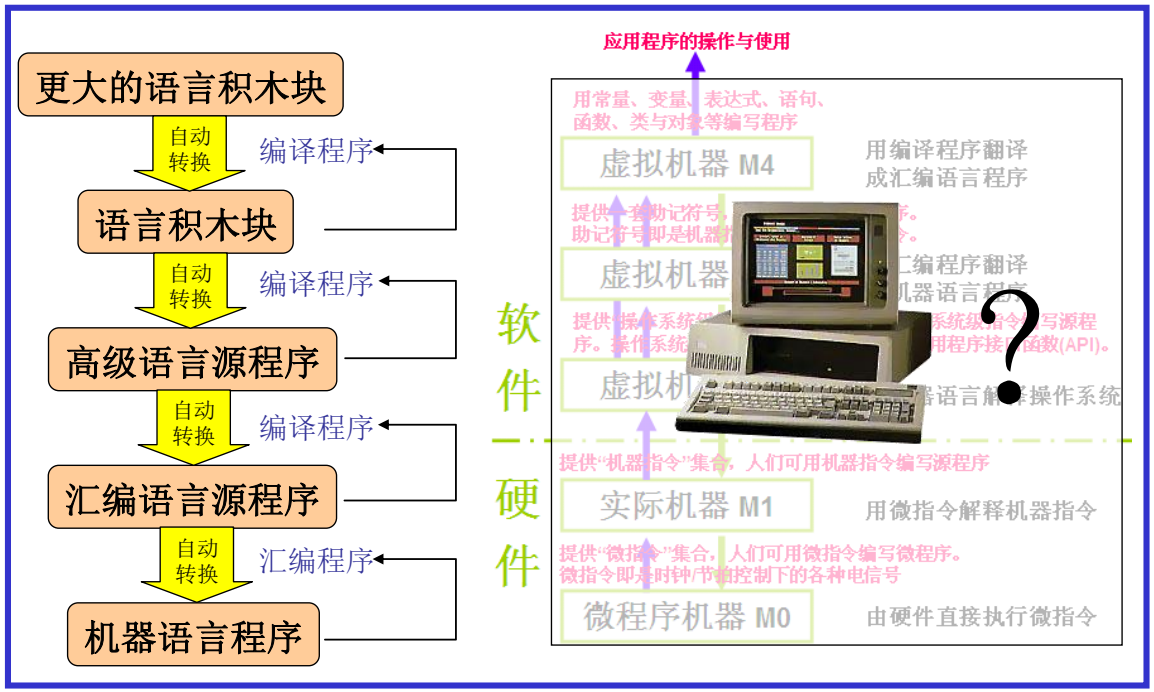
第5讲 由机器语言到高级语言：程序编写与编译

理解：“如何编写计算机可以执行的程序?” “为什么编写程序越来越方便?”以及“用各种语言编写的程序，机器为什么可以执行?”

内容提要

基本目标: 理解如何编写计算机可以执行的程序



基本思维：高级语言与汇编语言➔语言与编译器➔高级语言程序的构成要素➔不同层面的计算机

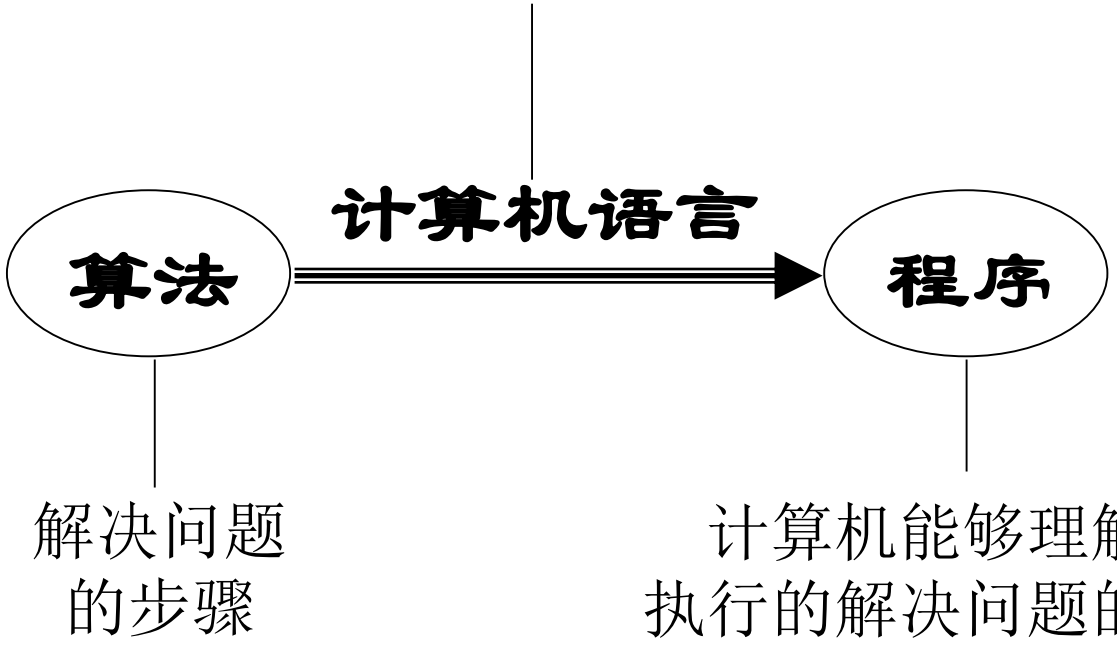
由机器语言到汇编语言

----机器语言、指令系统、机器语言程序

----汇编语言、源程序与汇编程序

算法、计算机语言与计算机程序

步骤书写的规范、语法规则、标准的集合
是人和计算机都能理解的语言



计算机语言---机器语言

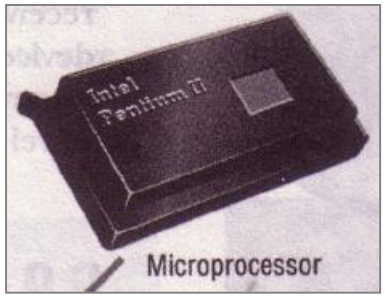
◆**指令系统：**CPU用二进制和编码提供的可以解释并执行的命令的集合。

操作码	地址码
100001	10 00000111
100010	11 00001010

◆**机器语言：**用二进制和编码方式提供的指令系统所编写程序的语言被称为机器语言

◆所有程序都需转换成机器语言程序，计算机才能执行

问：用机器语言编写程序存在什么问题呢？



计算7+10并存储的程序

100001 10
00000111

100010 10
00001010

100101 11
00000110

111101 00

计算机语言---汇编语言

◆用符号编写程序 ==➔ 翻译 ==➔ 机器语言程序

◆人们提供了用助记符编写程序的规范/标准。同时开发了一个翻译程序，实现了将符号程序自动转换成机器语言程序的功能。

操作码　　地址码

100001　1000000111

MOV A, 7

计算7+10并存储的程序

```
MOV  A,  7
ADD  A,  10
MOV  (6),  A
HLT
```

◆**汇编语言：** 是用助记符号编写程序的语言。

◆**汇编语言源程序：** 是用汇编语言编出的程序。

◆**汇编程序：** 是将汇编语言源程序翻译成机器语言程序的程序。

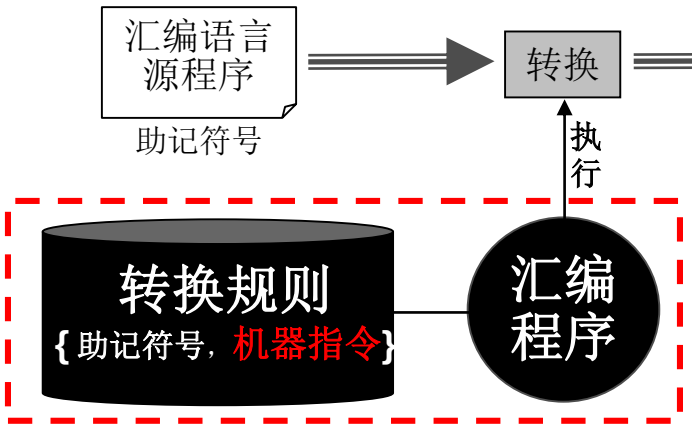
计算机语言---汇编语言---汇编程序(编译器)

◆汇编语言程序处理过程

汇编语言

用助记符号书写程序的规范、语法规则、标准的集合

是人和计算机都能理解的语言



机器语言

机器指令的集合

是计算机能够理解并执行，但人理解困难的语言

编制

```
MOV A, 7
ADD A, 10
MOV (6), A
HLT
```

完成7+10并存储的汇编语言程序

由汇编程序自动转换

```
10000110
00000111
10001011
00001010
10010111
00000110
11110100
```

执行

完成7+10并存储的机器语言程序

由汇编语言到高级语言

----高级语言、源程序与编译器

计算机语言---高级语言

◆人们提供了类似于**自然语言方式、以语句为单位书写程序的规范/标准**。并开发了一个**翻译程序**，实现了将语句程序**自动**翻译成机器语言程序的功能。

◆**高级语言：** 是用类似自然语言的语句编写程序的语言。

◆**高级语言源程序：** 是用高级语言编出的程序。

◆**编译程序：** 是将高级语言源程序翻译成机器语言程序的程序。

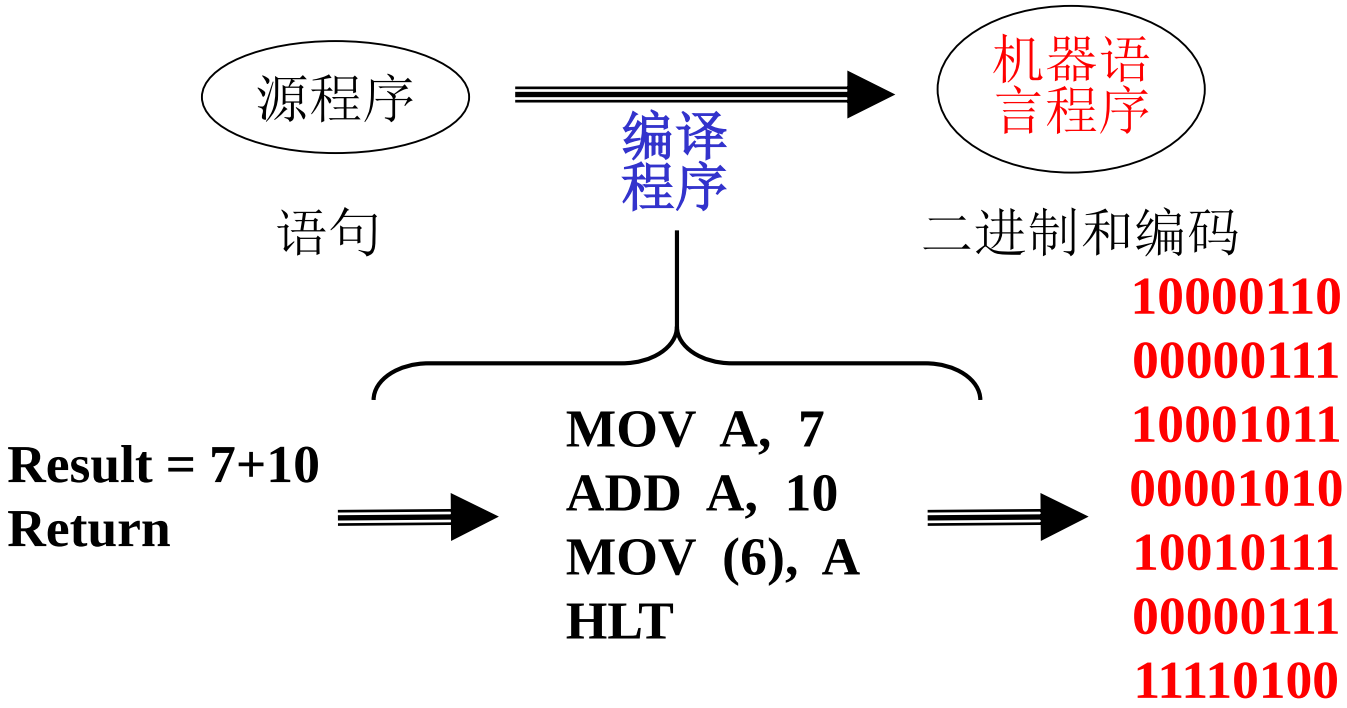
计算7+10并存储的程序

```
Result = 7+10;
Return
```

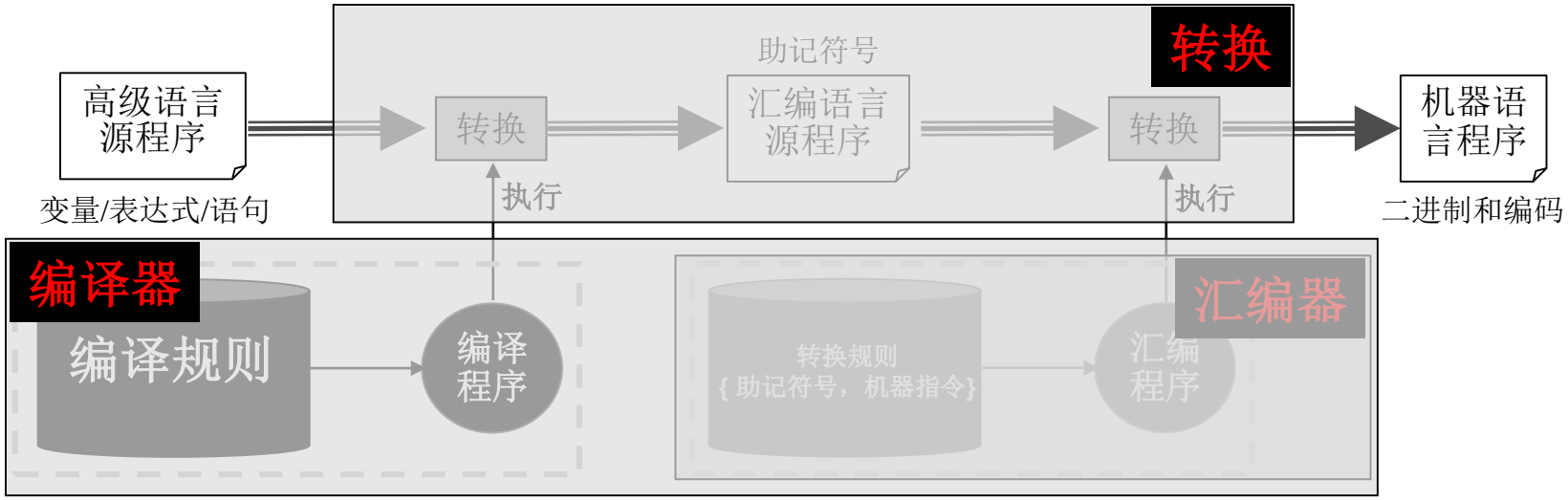
◆高级语言：机器无关性；一条高级语言语句往往可由若干条机器语言语句实现且不具有对应性

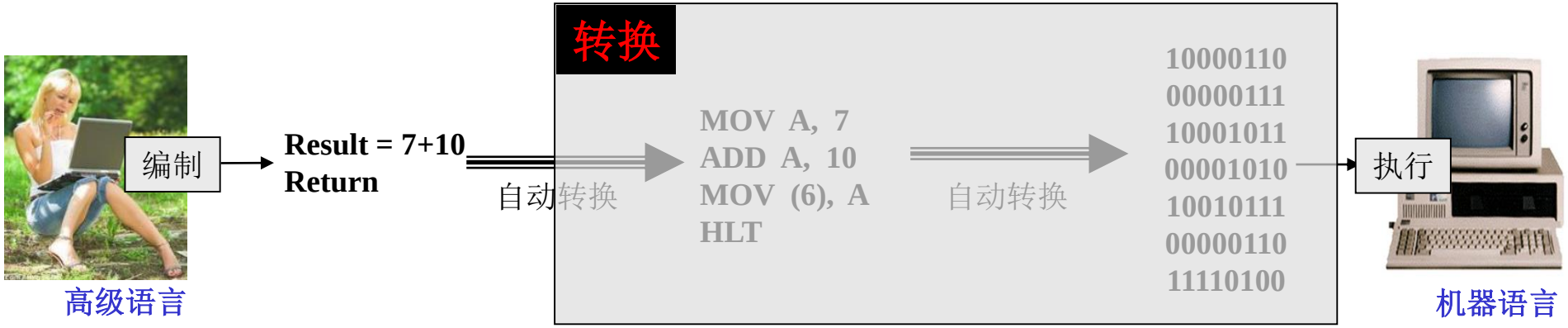
◆汇编语言：机器相关性；汇编语言语句和机器语言语句有对应性

高级语言程序处理过程示意



高级语言编译器


转换
助记符号
高级语言
源程序
转换
汇编语言
源程序
转换
机器语
言程序
变量/表达式/语句
执行
执行
二进制和编码
编译器
编译规则
编译
程序
转换规则
{助记符号，机器指令}
汇编器
汇编
程序


编制
Result = 7+10
Return
转换
自动转换
MOV A, 7
ADD A, 10
MOV (6), A
HLT
自动转换
10000110
00000111
10001011
00001010
10010111
00000110
11110100
执行
高级语言
机器语言

高级语言编译器-基本思想

由“具体的”运算式到“模式”运算式

Result = 7 + 10;

Sum = 8 + 15;

K = 100 + 105;

… …

→ **V = C + C;**

注： Result：具体的变量 7, 10：具体的常量	变化的部分	注： V：变量 C：常量
= 赋值符号 + 加法运算符号 ; 语句结束符	不变的部分 (保留字)	= 赋值符号 + 加法运算符号 ; 语句结束符

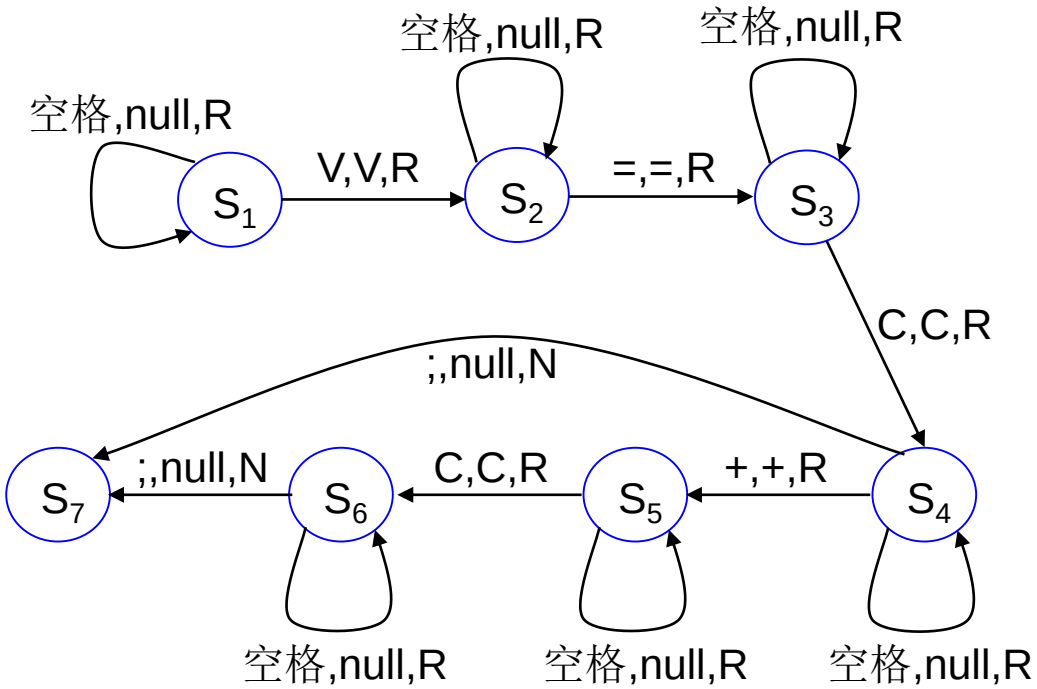
“模式”运算式的识别及常量、变量的标识

V = C + C;

注：字母表{V，C，=，+，空格，；}；S_1起始状态；S_7终止状态；null表示什么也不写回。

Result = 7 + 10;

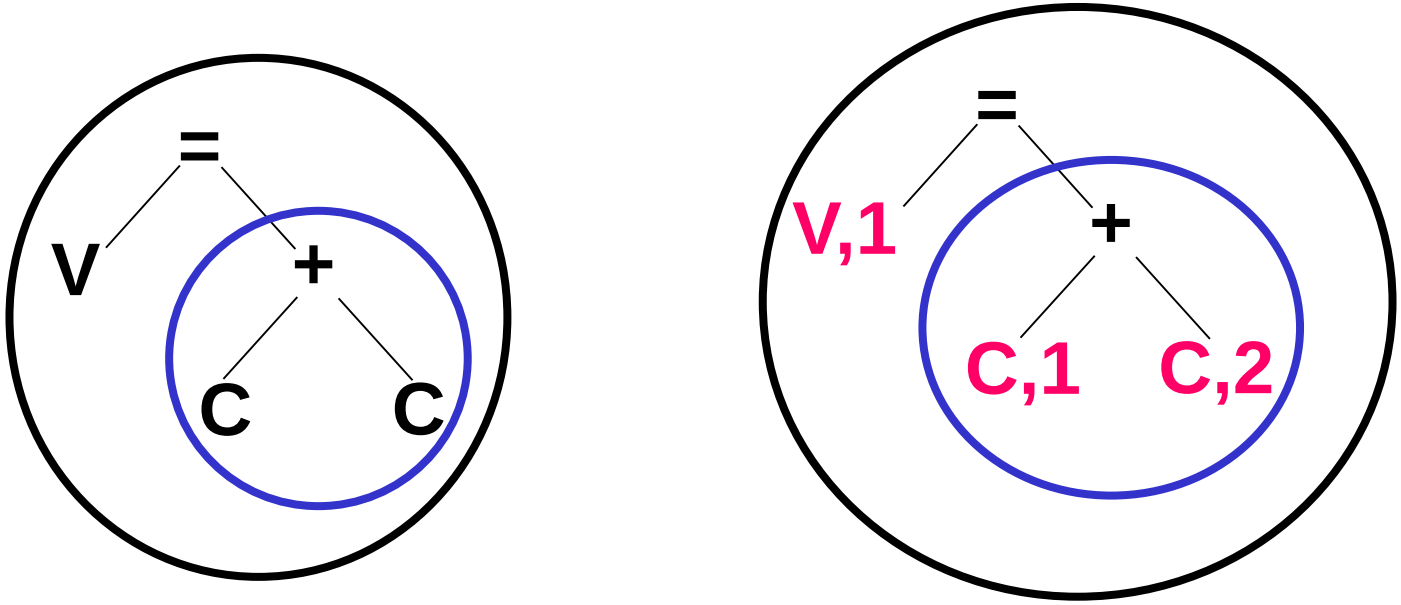
(V, 1) = (C, 1) + (C, 2);



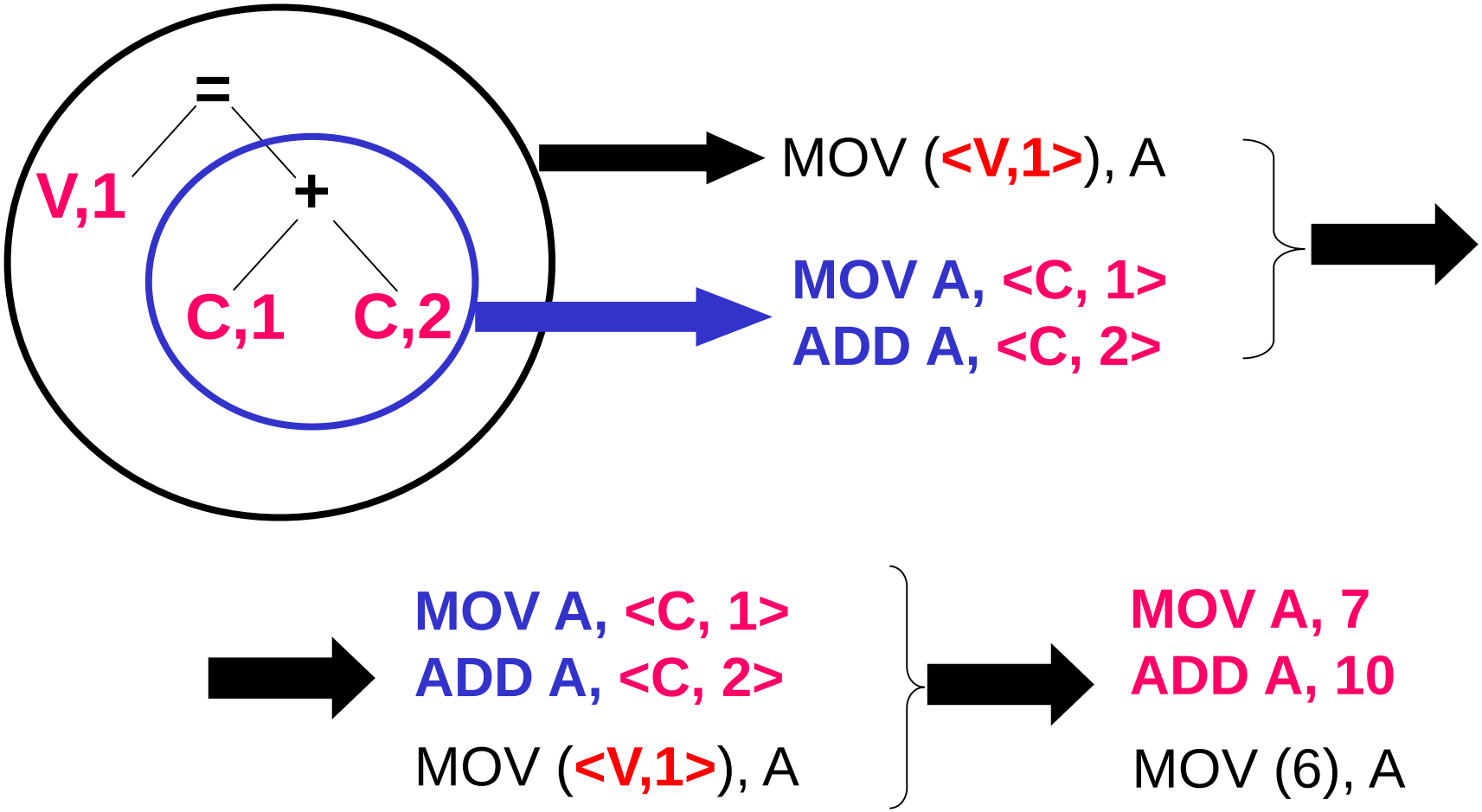
(c)能识别两种模式 “V=C;”和“V=C+C;”并能去除空格的图灵机示意图

复杂模式转换为简单模式及其组合

V = C + C;

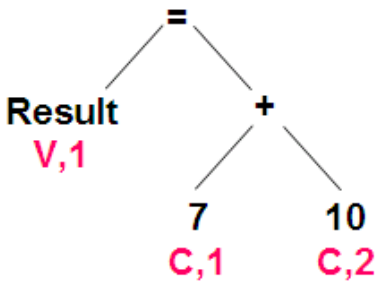


将简单模式转换成汇编语言语句序列，用常量值和变量地址进行替换，组合次序调整，得到最后的汇编语言程序



Result = 7 + 10;

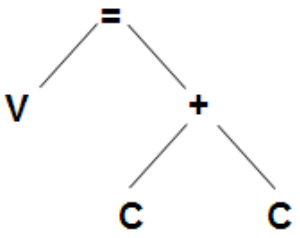
注：
Result：具体的变量
7, 10：具体的常量
=：赋值符号
+：加法运算符号



(a)一种具体的语句
及其解析结构

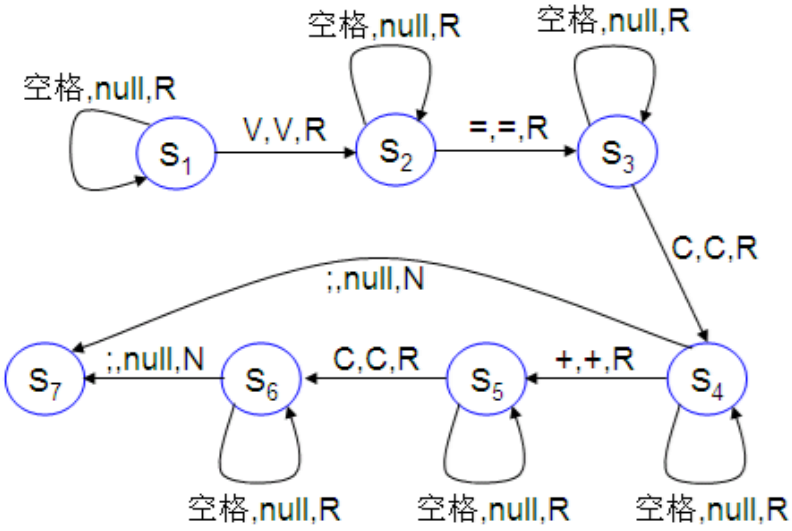
V = C + C;

注：
V：变量
C：常量
=：赋值符号
+：加法运算符号

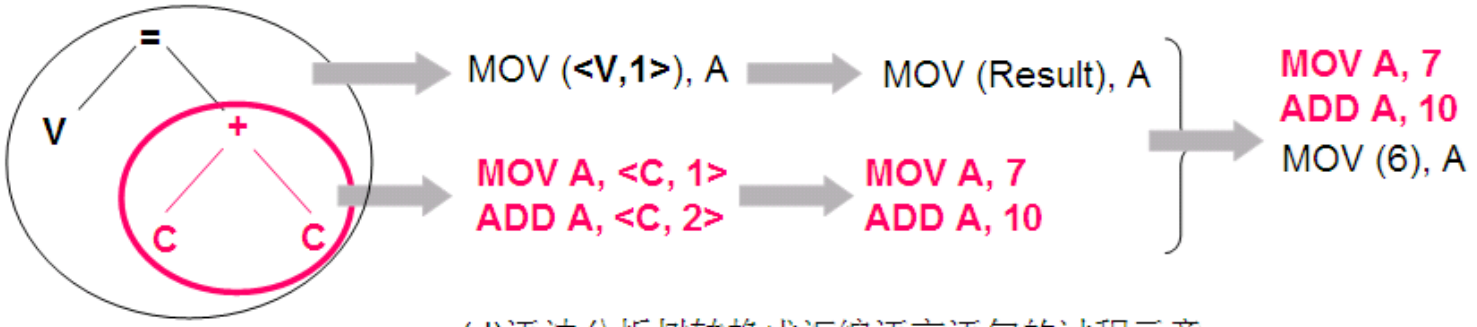


(b)图(a)所示语句的一种
模式及其解析结构

注：字母表{V，C，=，+，空格，;}；S_1起始状态；S_7终止状态；null表示什么也不写回。



(c)能识别两种模式“V=C;”和 “V=C+C;”
并能去除空格的图灵机示意图

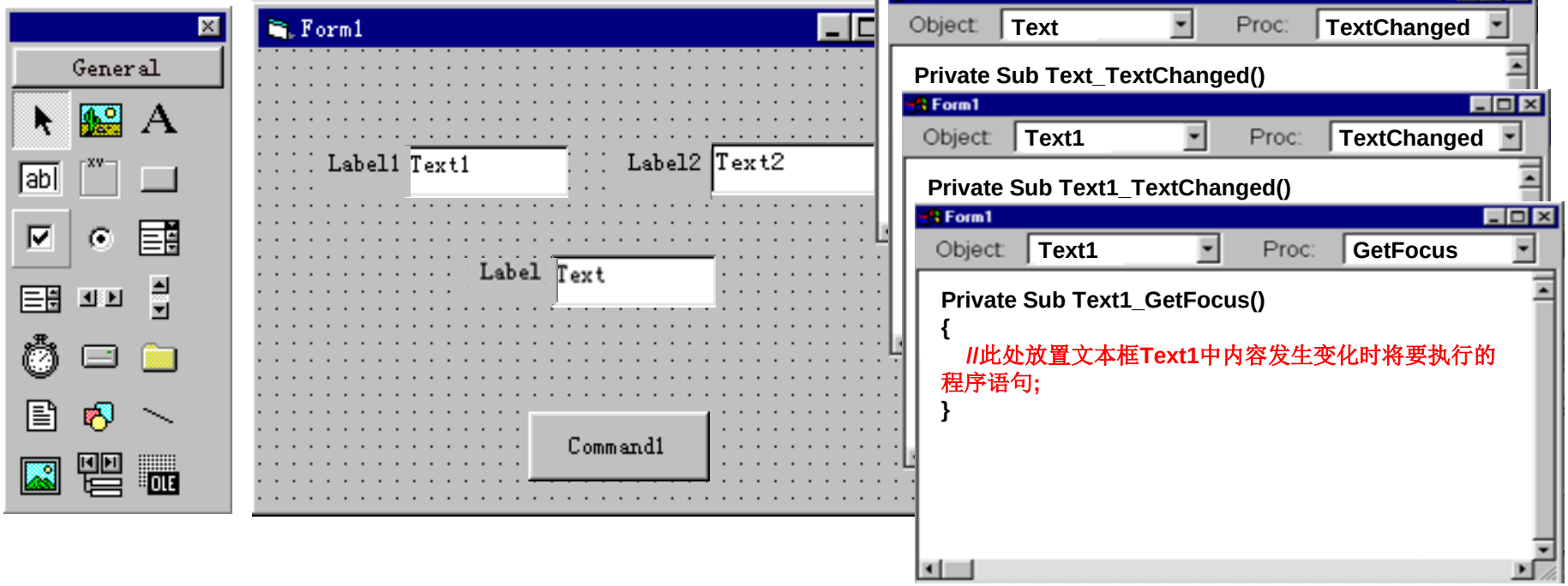


(d)语法分析树转换成汇编语言语句的过程示意

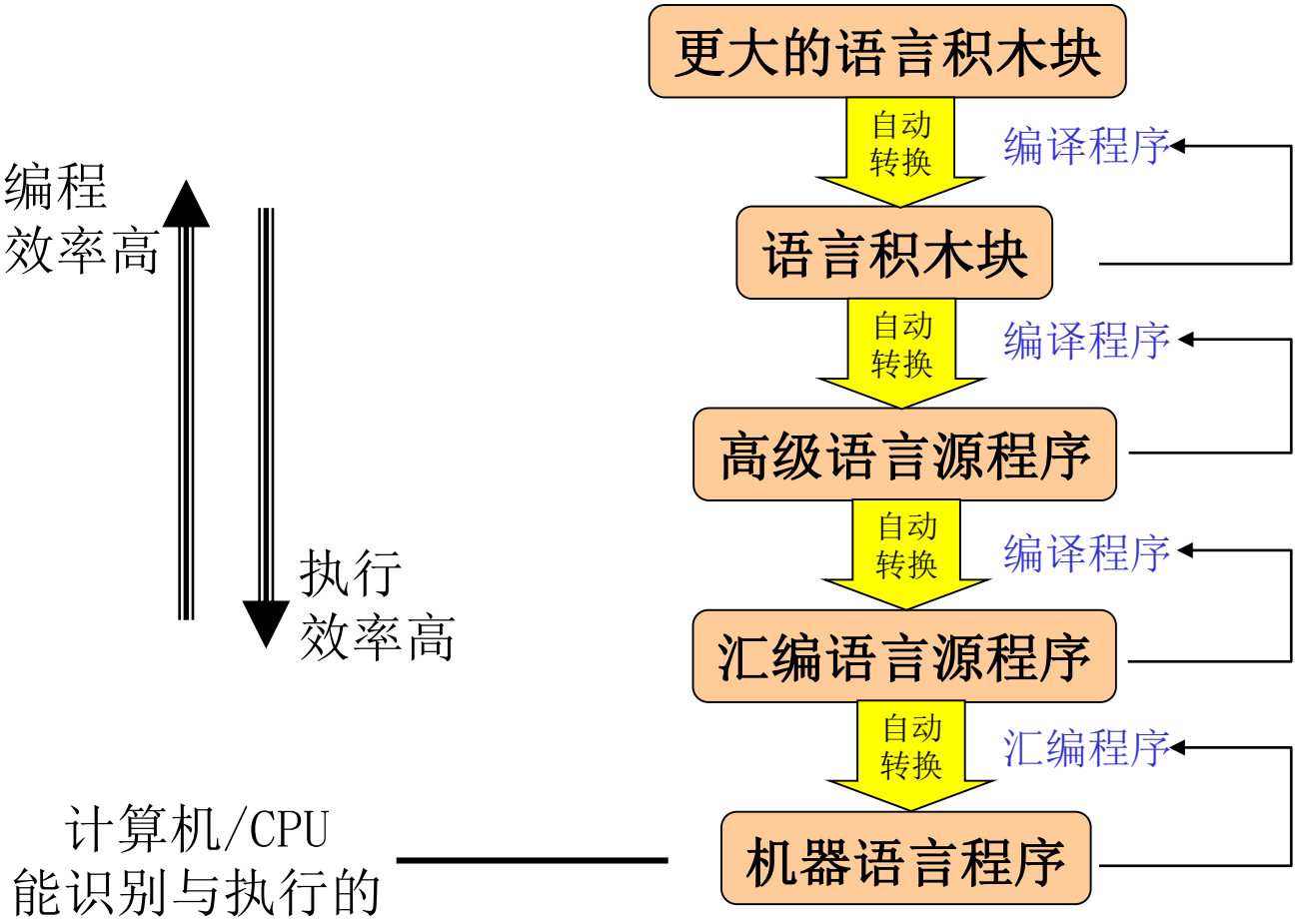
计算机语言的发展

面向对象的程序设计语言与可视化构造语言

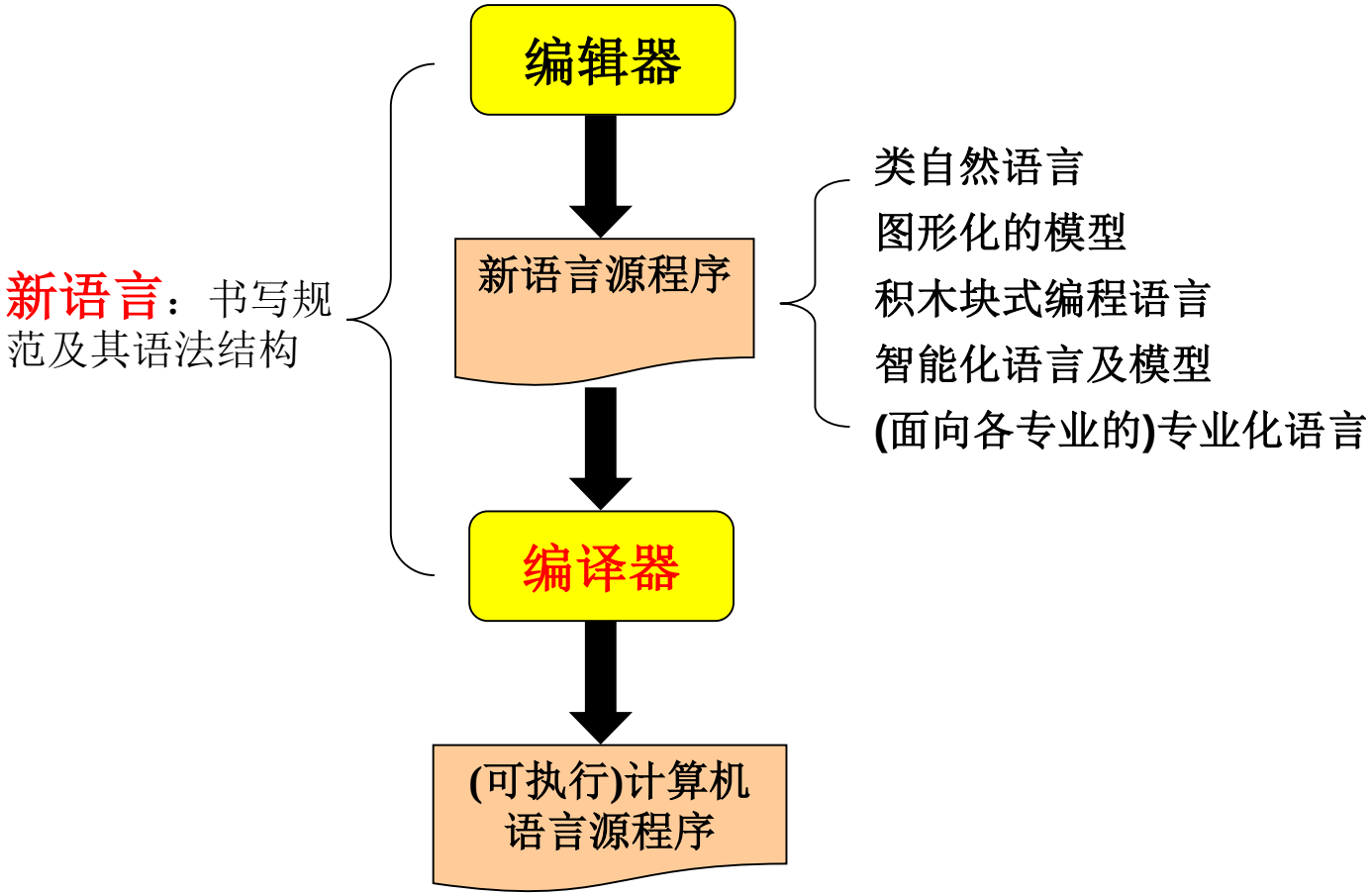
----像堆积木一样构造程序



计算机语言发展的基本思维



不仅要用语言，还要发明新语言



计算机技术是伴随着计算机语言的不断发展而发展起来的

◆因计算机语言获得图灵奖的

●1966 A.J. Perlis: 编程技术和编译架构

●1972 E.W. Dijkstra: ALGOL语言

●1974 Donald E. Knuth: 程序语言1977 John Backus : 高级语言,Fortran

●1979 Kenneth E. Iverson: 编程语言,APL

●1980 C. Antony R. Hoare: 编程语言

●1981 Edgar F. Codd: 关系数据库语言

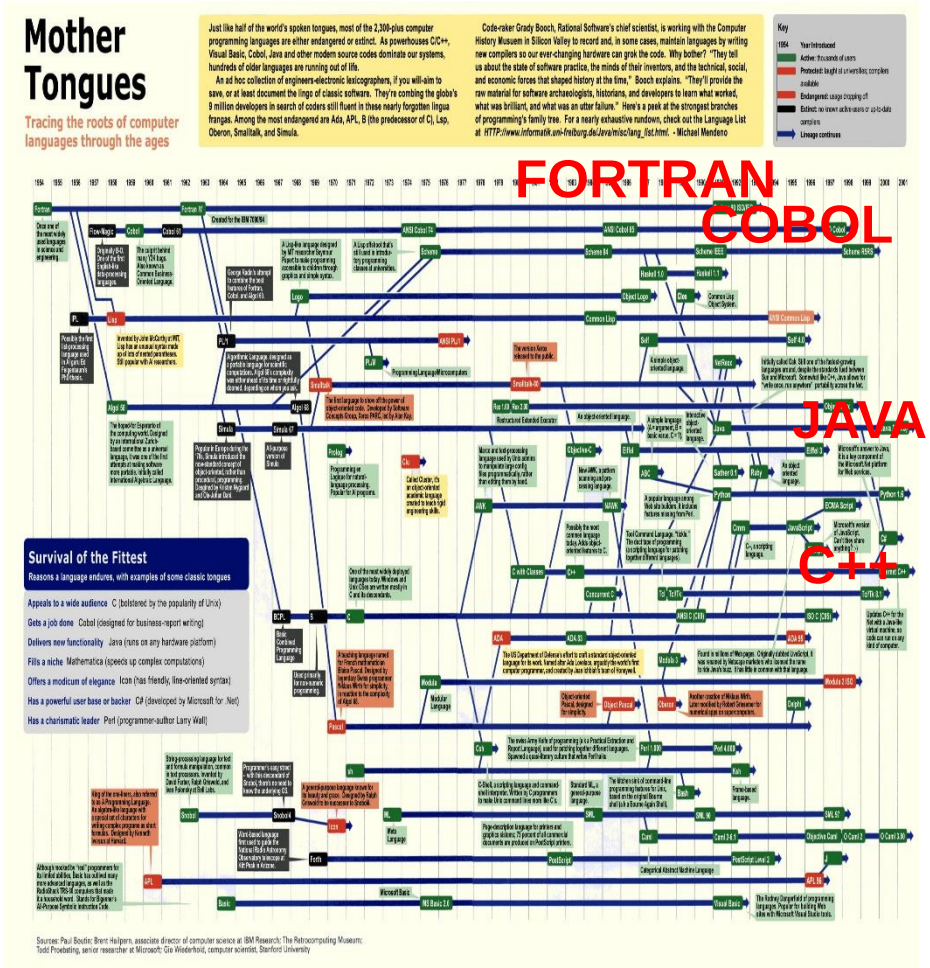
●1984 Niklaus Wirth：开发了EULER、ALGOL-W、 MODULA和PASCAL一系列崭新的计算语言。

●1987 John Cocke：编译器

●2001 Ole-Johan Dahl、Kristen Nygaard: 面向对象编程,SIMULA I 和SIMULA 67中。

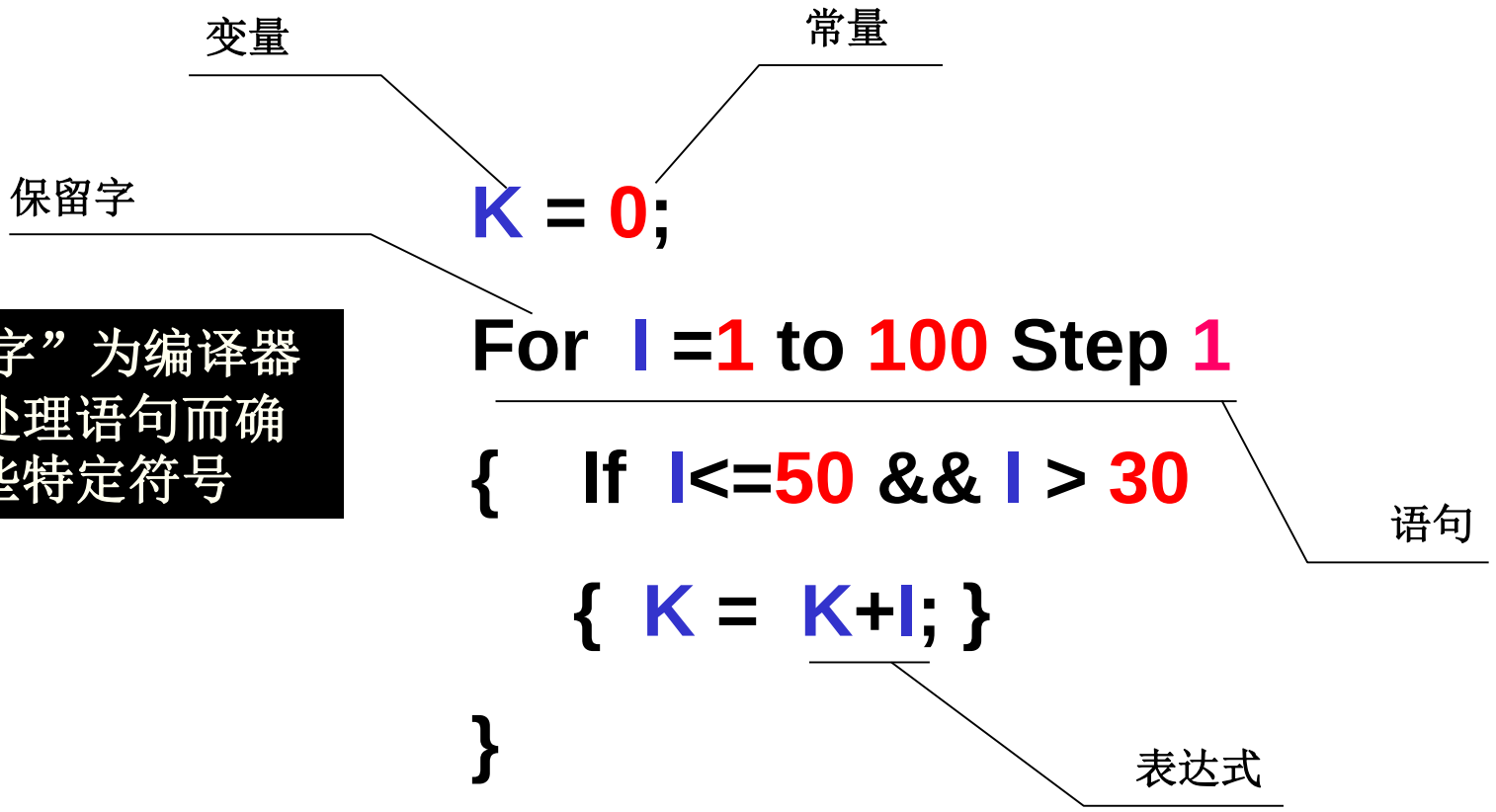
●2003 Alan Kay :面向对象语言，Smalltalk

●2005 Peter Naur:Algol60程序语言。2006 Fran Allen: 编译器



计算机语言(程序)的基本构成要素(I)

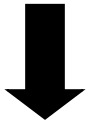
认识计算机语言程序



“保留字”为编译器识别和处理语句而确定的一些特定符号

常量、变量与表达式

A1 + (B2 – x1 + 76) * 3



(+ A1 (* (+ (- B2 x1) 76) 3)

◆算术表达式示例。算术表达式的结果是一数值；

A1 + (B2 – x1 + 76) * 3

(B2 + yy4) / L3 – xx3

◆比较表达式示例。比较表达式的计算结果是逻辑“真”或“假”；

Grade < 90

Grade >= 70

N4 < A1 + B2 + 20 //注：A1+B2+20为算术表达式，计算完后再与N4的值进行比较

◆逻辑表达式示例。逻辑表达式的计算结果是逻辑“真”或“假”；

(x1 >= A1) && (B2 <> y2)

◆将表达式的计算结果赋值给一变量：赋值语句

M = X>Y+50;

M = (X>Y) AND (X<Y);

K = K + (5 * K);

语句与程序控制

◆顺序结构

```
G5 = 1;
G6 = 2;
G7 = 3;
G8 = 4;
G9 = 5;
G9 = G9 + G8;
G9 = G9 + G7;
G9 = G9 + G6;
G9 = G9 + G5;
```

程序执行示例

```
G5 = 1;
G6 = 2;
G7 = 3;
G8 = 4;
G9 = 5;
G9 = G9 + G8;
G9 = G9 + G7;
G9 = G9 + G6;
G9 = G9 + G5;
```

语句与程序控制

◆分支结构

IF 条件表达式 {
(条件为真时运行的)程序语句序列1 }
ELSE {
(条件为假时运行的)程序语句序列2 }

```
If  D1>D2
{    D1=D1-5;   }
Else
{   D1=D1+10;  }
```

```
Y = 50;
Z = 80;
X = 30;
X = Z + Y;
If  Y > Z   {
      X = X - Y;  }
Else {
      X= X - Z;   }
X = X + Y;
If  X > Z  {  X = Y; }
X = X - Z;
If  X>Y
{ X = X - Y;   }
```

语句与程序控制

X	130
Y	50
Z	80

```
Y = 50;
Z = 80;
X = 30;
X = Z + Y;
If  Y > Z   {
      X = X - Y;  }
Else {
      X= X - Z;   }
X = X + Y;
If  X > Z  {  X = Y; }
X = X - Z;
If  X>Y
{ X = X - Y;   }
```

语句与程序控制

◆循环结构(有界循环结构)

For (计数器变量 = 起始值 To 结束值 [增量表达式])

{ 循环体的程序语句序列 }

Next [计数器变量]

```
Sum=0;
For  I = 1 to 5 Step 1
{  Sum = Sum + I;  }
Next I
//继续其他语句
```

```
Sum=0;
For  I =1 to 10000 Step 2
{ Sum = Sum + I; }
Next I
```

Sum [redacted]

I [redacted]

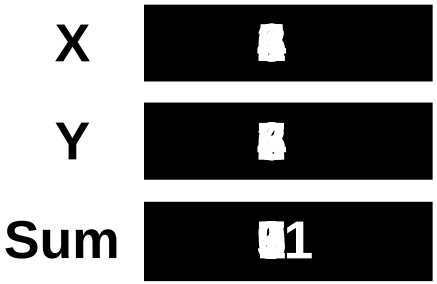
语句与程序控制

◆循环结构(条件循环结构)

Do

{ 循环体的程序语句序列 }

While (条件表达式);




```
X=1;
Y=2;
Sum=0;
Do {
    Sum = X+Y;
    X=X+1;
    Y=Y+1;
} While (Sum<=10)
//其他语句
```

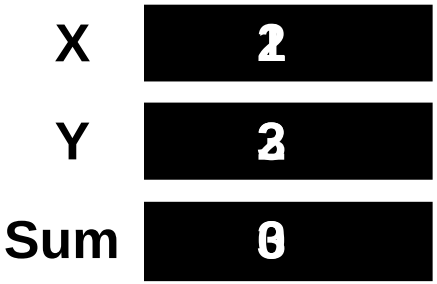
语句与程序控制

◆循环结构(条件循环结构)

Do

{ 循环体的程序语句序列 }

While (条件表达式);




```
X=1;
Y=2;
Sum=0;
Do {
    Sum = X+Y;
    X=X+1;
    Y=Y+1;
} While (Sum<0)
//其他语句
```

语句与程序控制

◆循环结构(条件循环结构)

While (条件表达式)

Do { 循环体的程序语句序列 }

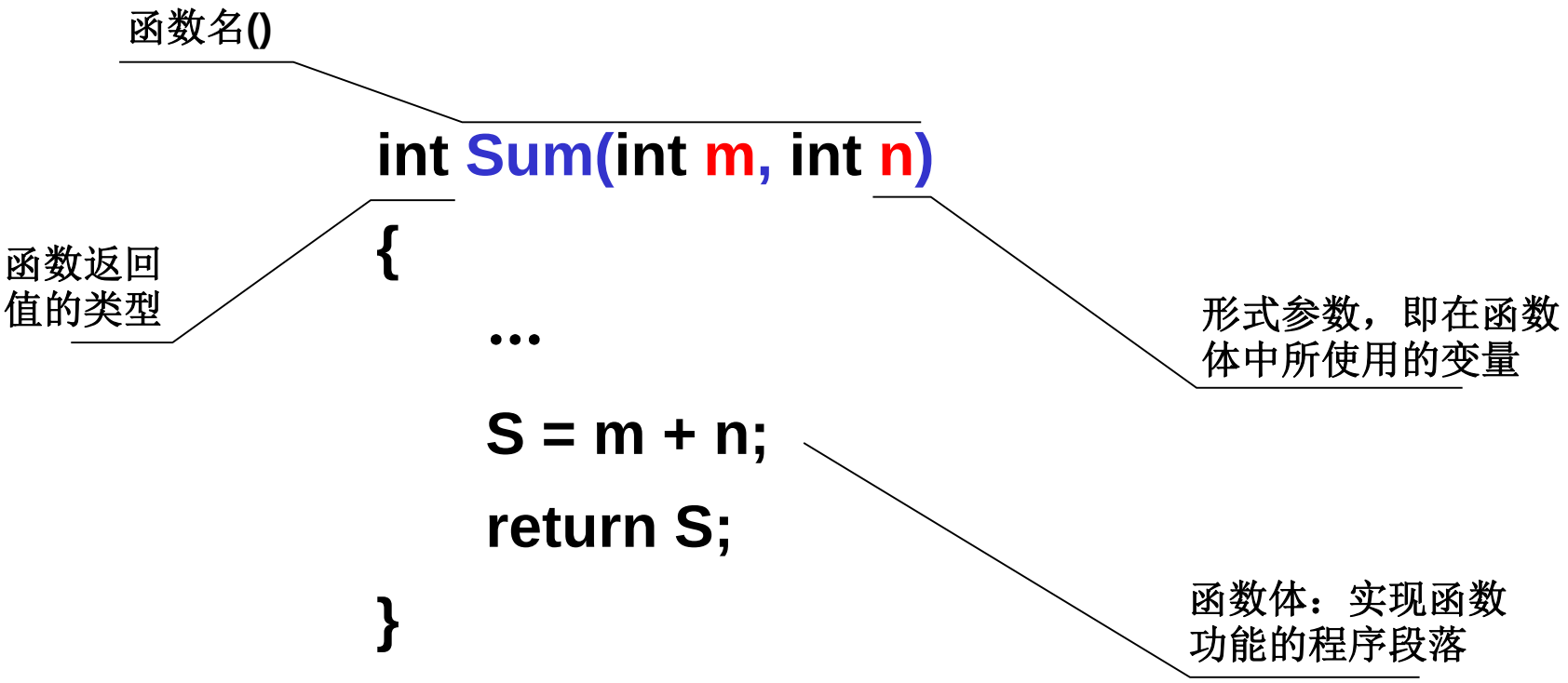
X	1
Y	2
Sum	0

```
X=1;
Y=2;
Sum=0;
While (Sum<0)
    Do {
        Sum = X+Y;
        X=X+1;
        Y=Y+1;
        }
<其他语句>
```

计算机语言(程序)的基本构成要素(II)

函数是很重要的程序构造手段，你知道吗？

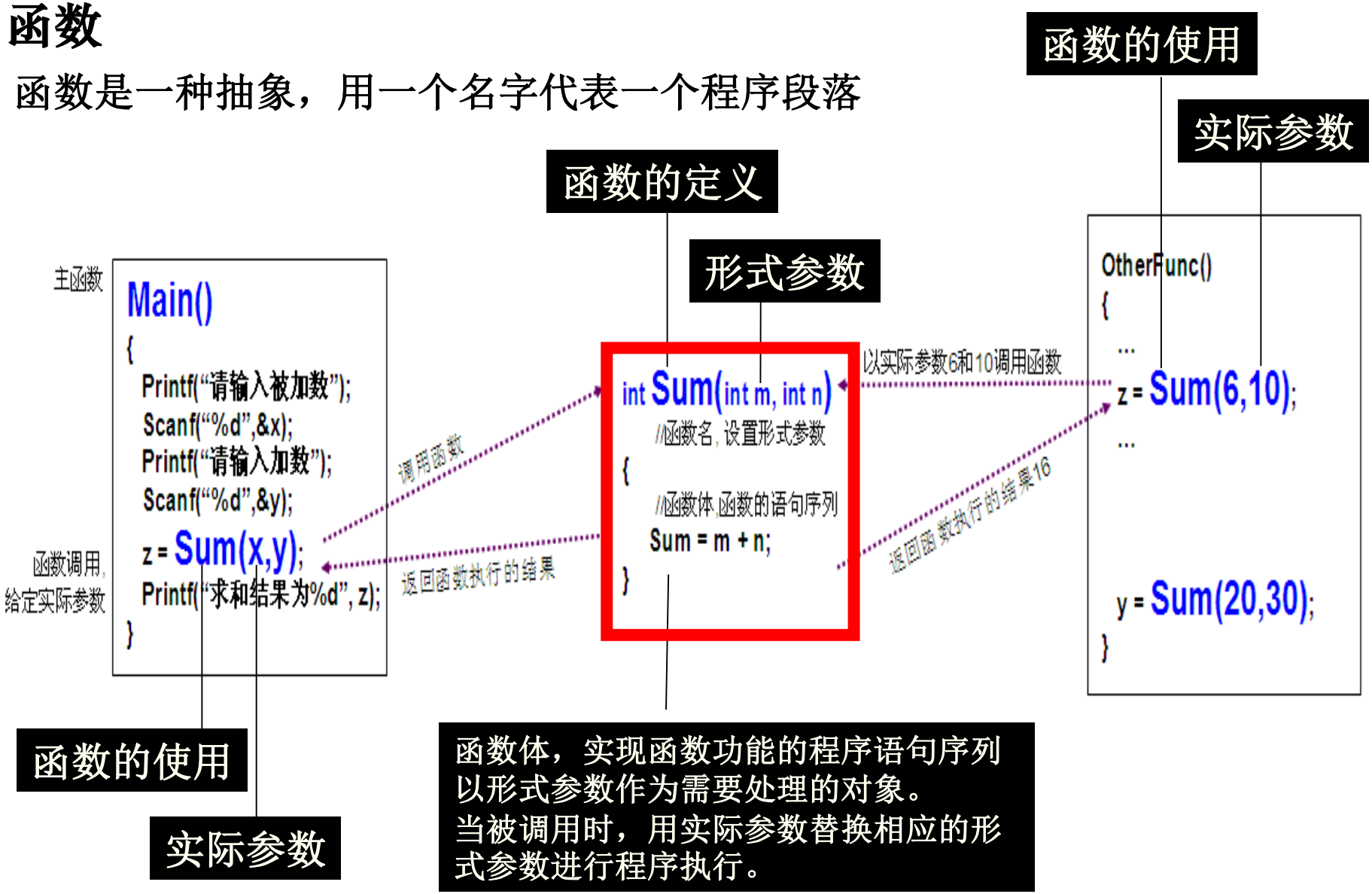
函数



数学上的函数只是一个符号表达，而计算机程序中的函数则是一段可以执行的程序

函数

函数是一种抽象，用一个名字代表一个程序段落

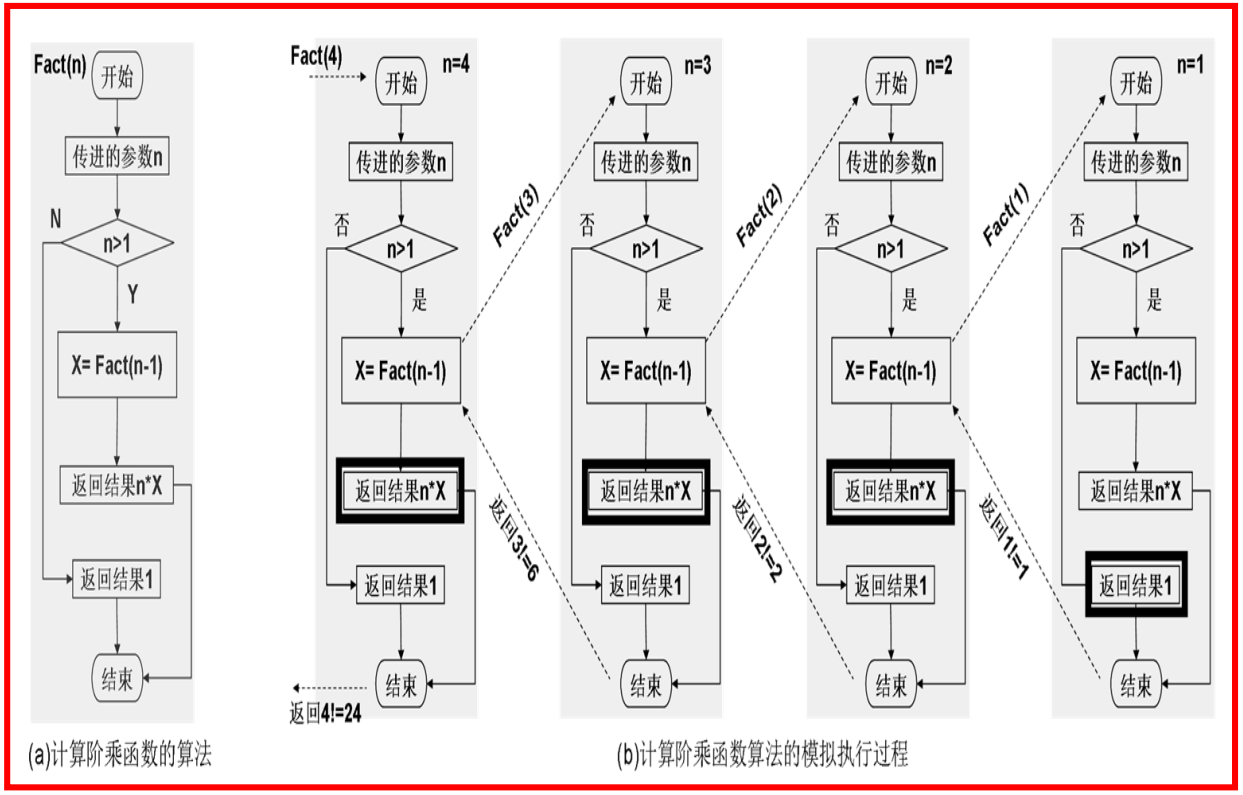


系统提供的可以使用的函数类别

- **数学运算函数**，如三角函数、指数与对数函数、开方函数等；例如sin(α)，Log(x)等；
- **数据转换函数**，如字母大小写变换、数值型数字和字符型数字相互转换等；
- **字符串操作函数**，如取子串、计算字符串长度等；例如，Len("abcd")；
- **输入输出函数**，如输入输出数值、字符、字符串等；例如，Printf(…),Scanf(…)等；
- **文件操作函数**，如文件的打开、读取、写入、关闭等；
- **其它函数**，如取系统日期、绘制图形等。

程序示例：阶乘的递归程序如下示意

```
long int Fact(int n)
{    long int x;
     If (n > 1)
     { x = Fact(n-1);
     /*递归调用*/
        return n*x;   }
     else return 1;
   /*递归基础*/
}
```



程序示例：阶乘的迭代程序如下示意

```
long int Fact(int n)
{    int  counter;
     long product = 1;
     for  counter = 1 to n step 1
     { product = product * counter;   }
      /*迭代*/
     return product;
}
```

	Product	Counter
初始值	1	
循环第1次	1	1
循环第2次	1	2
循环第3次	2	3
循环第4次	6	4
循环第5次	24	5
循环第6次	120	6

$$n!=\begin{cases}1 & \text{当} n\le 1\text{时}\\ n\times(n-1)\times\ldots\times 1 & \text{当} n>1\text{时}\end{cases}$$

变量及其存储

◆变量与存储单元

	变量名	变量值
Int X=23;	*x*	00000000 00010111
Char Y= 'AB;'	*y*	01000001 01000010
Char Z= 'ABCD';	*Z*	01000001 01000010
		01000011 01000100
		00000000

存储地址	存储内容
00000000 00000001	00000000 00010111
00000000 00000010	01000001 01000010
00000000 00000011	01000001 01000010
00000000 00000100	01000011 01000100
00000000 00000101	00000000
00000000 00000110	

变量及其存储

◆ “变量”与“指针变量”

String *P=”ABCD”;	P
String v = “ABCD”;	v *p

存储地址	存储内容	
00000000 00000000	00000100	00001000
00000000 00000001	00000000	00000100
00000000 00000010	00001100	00001010
00000000 00000011	00000000	00000000
00000000 00000100	01000001	01000010
00000000 00000101	01000011	01000100
00000000 00000110	00000000	00000000
00000000 00000111	01000001	01000010
00000000 00001000	01000011	01000100
00000000 00001001	00000000	00000000

变量及其存储

◆“变量”与“变量类型”及其存储

用名字表示的存储地址，即变量名	存储地址	存储内容(即变量值)
Mark	00000000 00000000 00000000 00000001 00000000 00000010 00000000 00000011	(注：可通过赋值发生改变)
Sum	00000000 00000100 00000000 00000101	(注：可通过赋值发生改变)
Distance	00000000 00000110 00000000 00000111	(注：可通过赋值发生改变)

多元素变量及其存储

◆向量或列表是有序数据的集合型变量，向量中的每一个元素都属于同一个数据类型，用一个统一的向量名和下标来唯一的确定向量中的元素。在程序设计语言中，又称为数组。

◆向量名通常表示该向量的起始存储地址，而向量下标表示所指向元素相对于起始存储地址的偏移位置。

向量实例

82	Mark[0]
95	Mark[1]
100	Mark[2]
60	Mark[3]
80	Mark[4]

编写求上述数组中值的平均值的程序

```
n = 4;
Sum=0;
For  J =0 to n Step 1
{    Sum = Sum + mark[ J ];
        }
Next  J
Avg = Sum/(n+1);
```

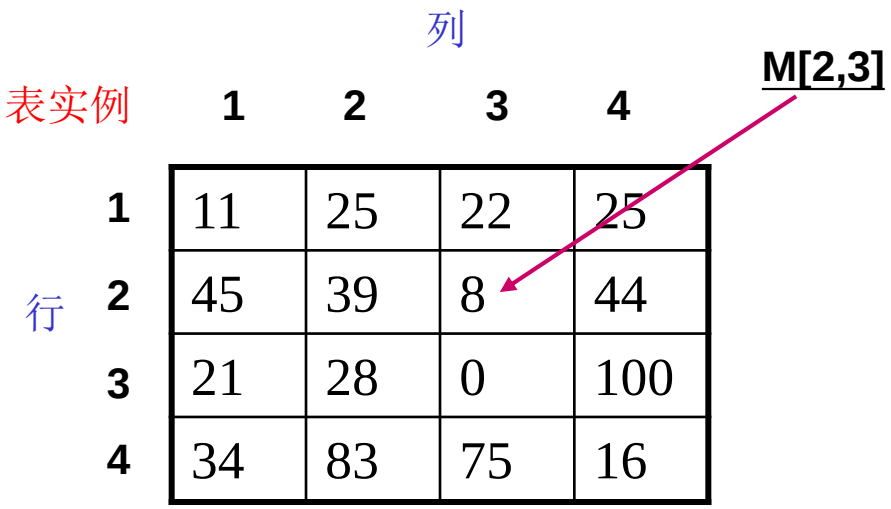
向量存储实例

用变量名和元素位置共同表示存储地址，即向量		存储地址	存储内容(即变量值)
Mark	[0]	00000000 00000000 00000000 00000001	(注：82 的 4 字节二进制数可通过赋值发生改变)
	[1]	00000000 00000010 00000000 00000011	(注：95 的 4 字节二进制数可通过赋值发生改变)
	[2]	00000000 00000100 00000000 00000101	(注：100 的 4 字节二进制数可通过赋值发生改变)
	[3]	00000000 00000110 00000000 00000111	(注：60 的 4 字节二进制数可通过赋值发生改变)
	[4]	00000000 00001000 00000000 00001001	(注：80 的 4 字节二进制数可通过赋值发生改变)

多元素变量使得程序可通过下标来操作多元素变量中的每一个元素

多元素变量及其存储

◆矩阵或表是按行按列组织数据的集合型变量，通常是一个二维向量，可表示为如M[2,3]或M[2][3]形式，即用符号名加两个下标来唯一确定表中的一个元素，前一下标为行序号，后一下标为列序号。系统会自动将其转换为对应的存储地址，找到相应的存储单元。在程序设计语言中，矩阵或表是一个多维数组变量。



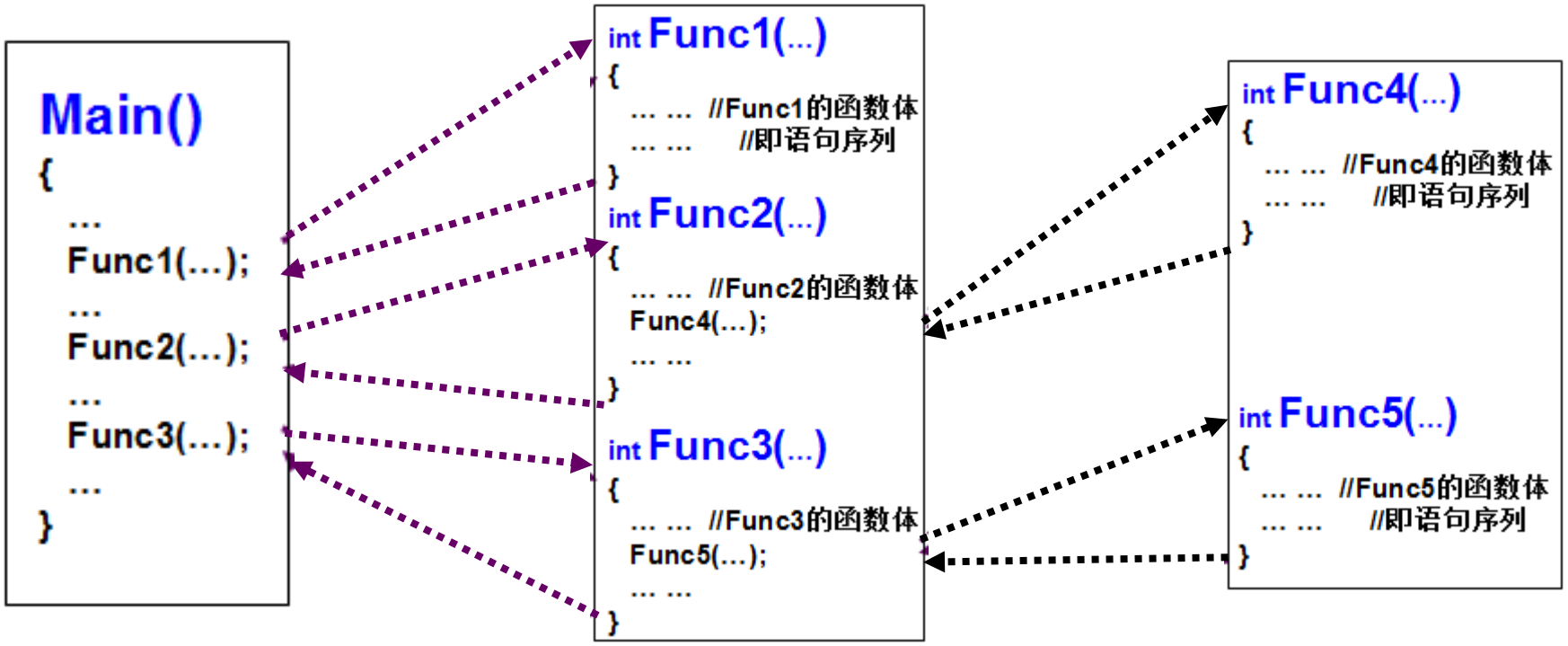

```
Sum=0;
For  I =1 to 4 Step 1
{   For  J =1 to 4 Step 1
    { Sum = Sum + M[I][J];  }
    Next  J
}
Next  I
Avg = Sum/16;
```

逻辑上是二维的按行、列下标来操作一个元素，如M[2,3]或M[2][3]；物理上仍旧是一维存储的，由“表起始地址+ (行下标-1)*列数/行+列下标”。这种转换可由系统自动完成，程序中只需按下标操作即可，即如M[2][3]

传统程序构造及其表达方法----由粗到细

为控制复杂性，先以函数来代替琐碎的细节，着重考虑函数之间的关系，以及如何解决问题

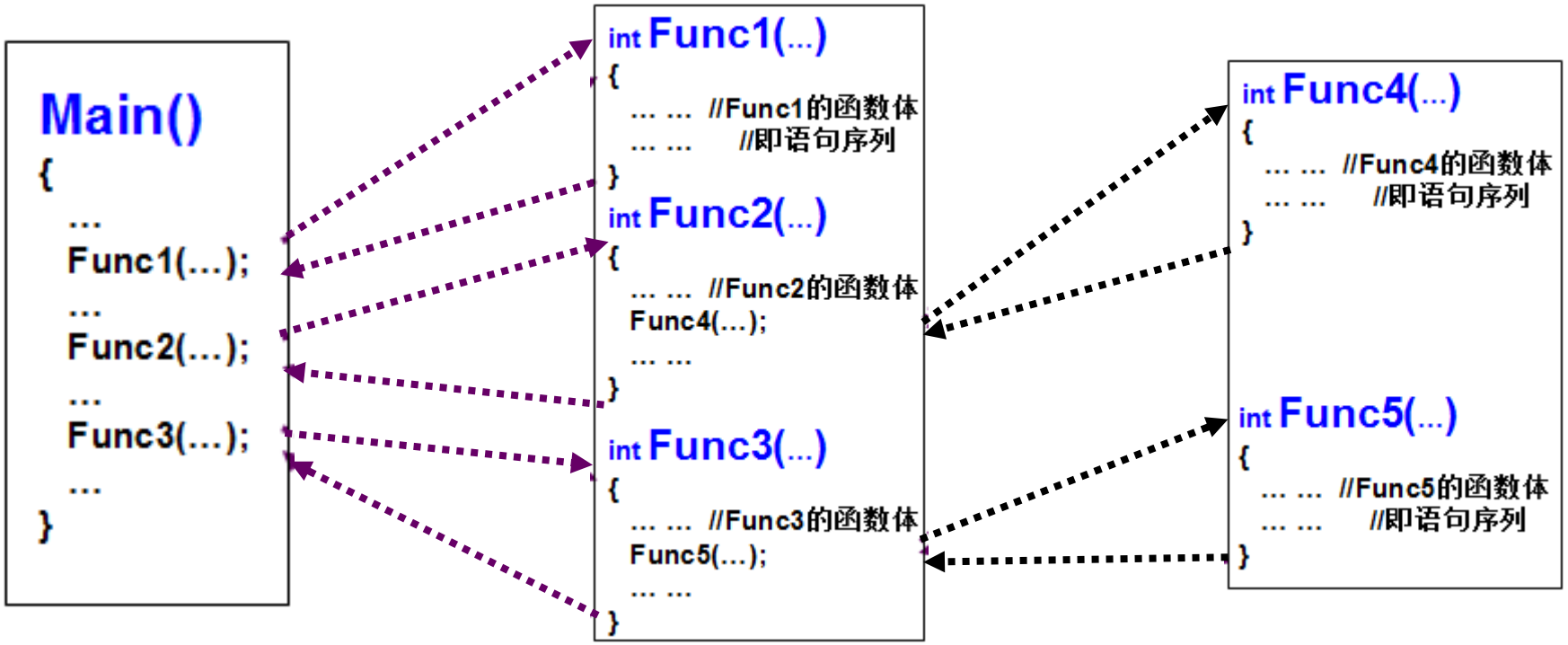
在前一阶段考虑清楚后或编制完成后，再编写其中的每一个函数。而函数的处理同样采取这种思路



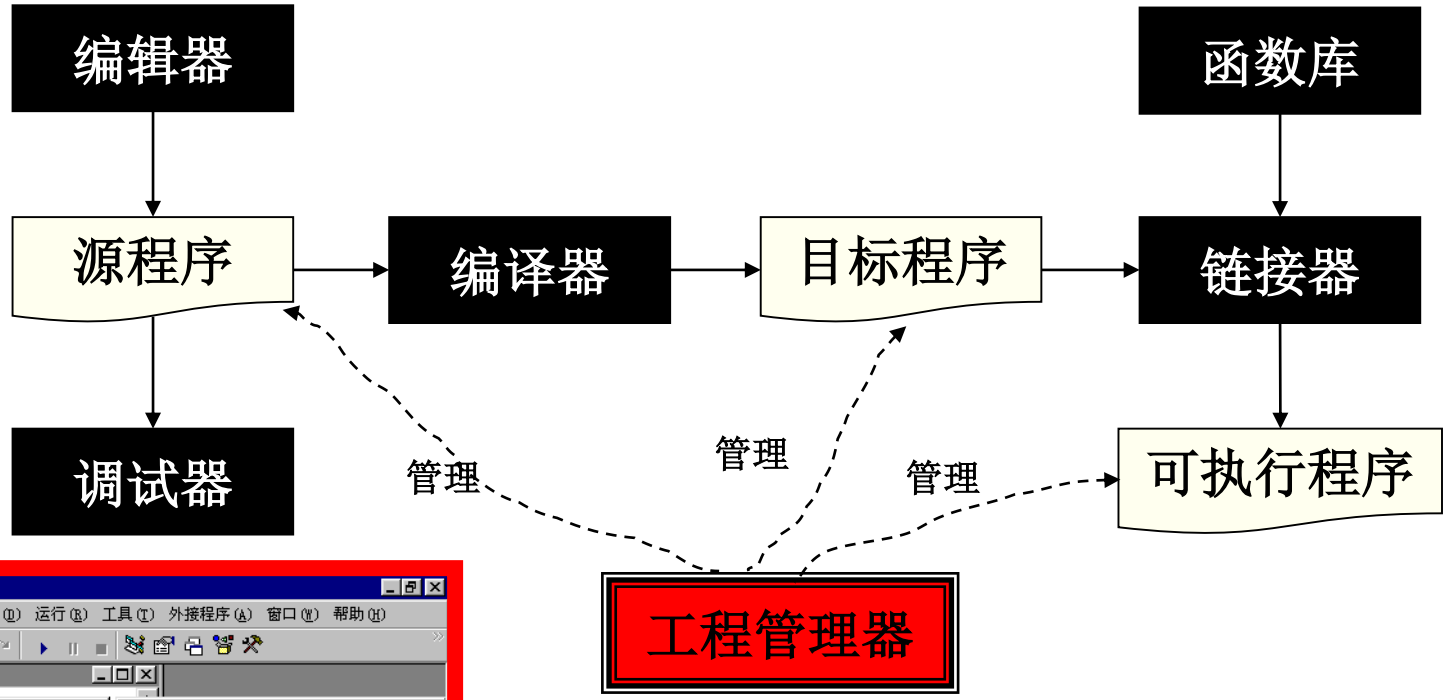
传统程序构造及其表达方法----也可以由细到粗

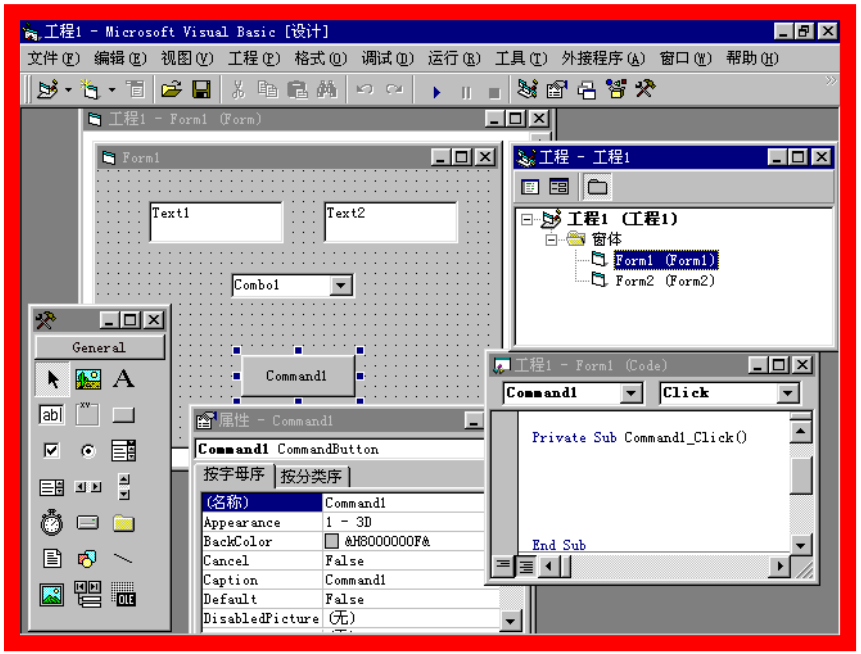
上一层次的函数依据下层函数来编写，确认正确后再转至更上层问题处理

首先编写一些基础性的函数，并确定其正确后，再处理上一层次的问题。



程序开发环境

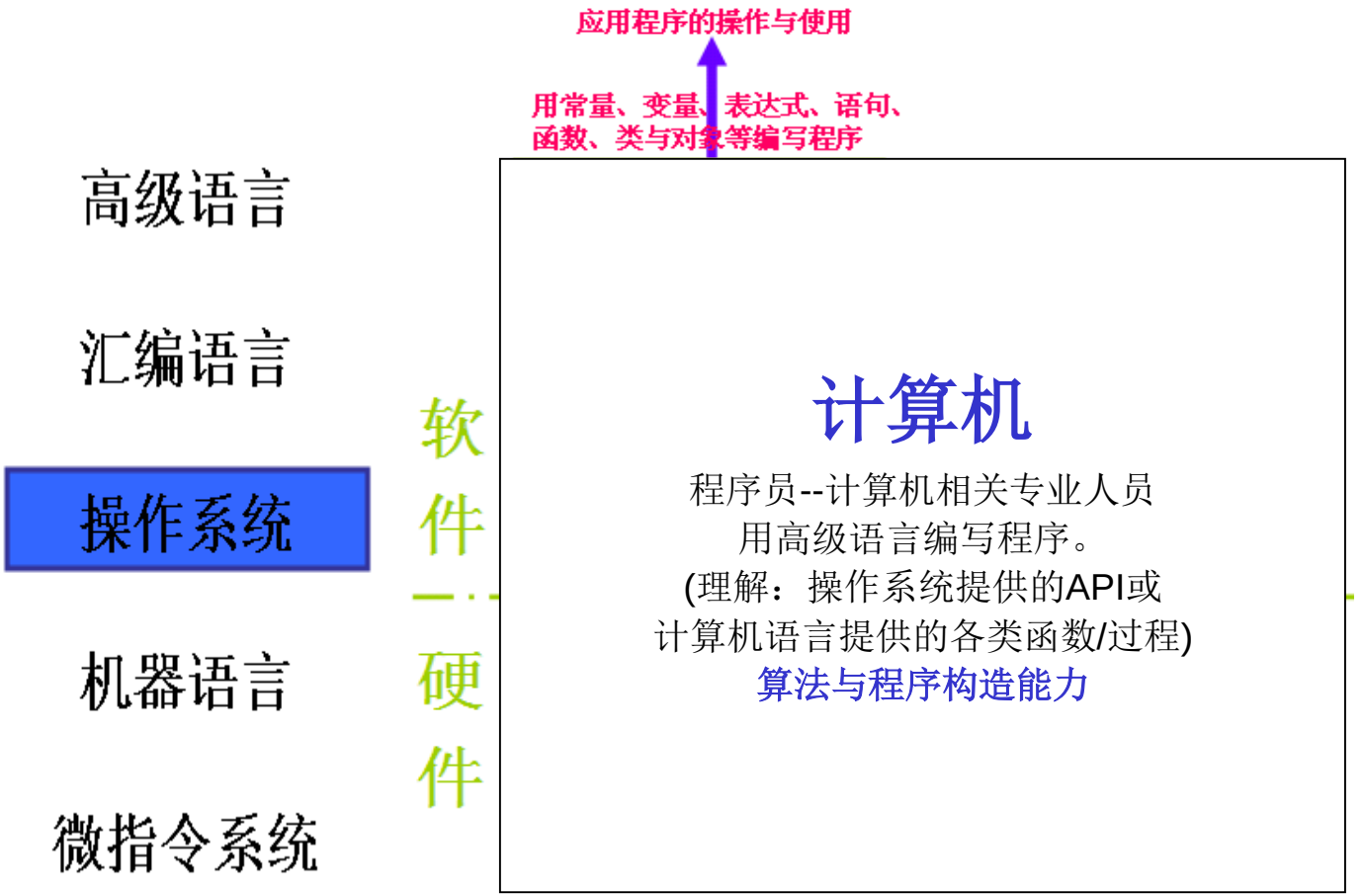

编辑器
函数库
源程序
编译器
目标程序
链接器
调试器
管理
管理
管理
可执行程序
工程管理器


工程1 - Microsoft Visual Basic [设计]
文件(F) 编辑(E) 视图(V) 工程(P) 格式(O) 调试(D) 运行(R) 工具(T) 外接程序(A) 窗口(W) 帮助(H)
工程1 - Form1 (Form)
Form1
Text1
Text2
Combo1
Command1
工程 - 工程1
工程1 (工程1)
窗体
Form1 (Form1)
Form2 (Form2)
General
工程1 - Form1 (Code)
Command1
Click
Private Sub Command1_Click()
End Sub
属性 - Command1
Command1 CommandButton
按字母序
按分类序
(名称) Command1
Appearance 1 - 3D
BackColor &H8000000F&
Cancel False
Caption Command1
Default False
DisabledPicture (无)

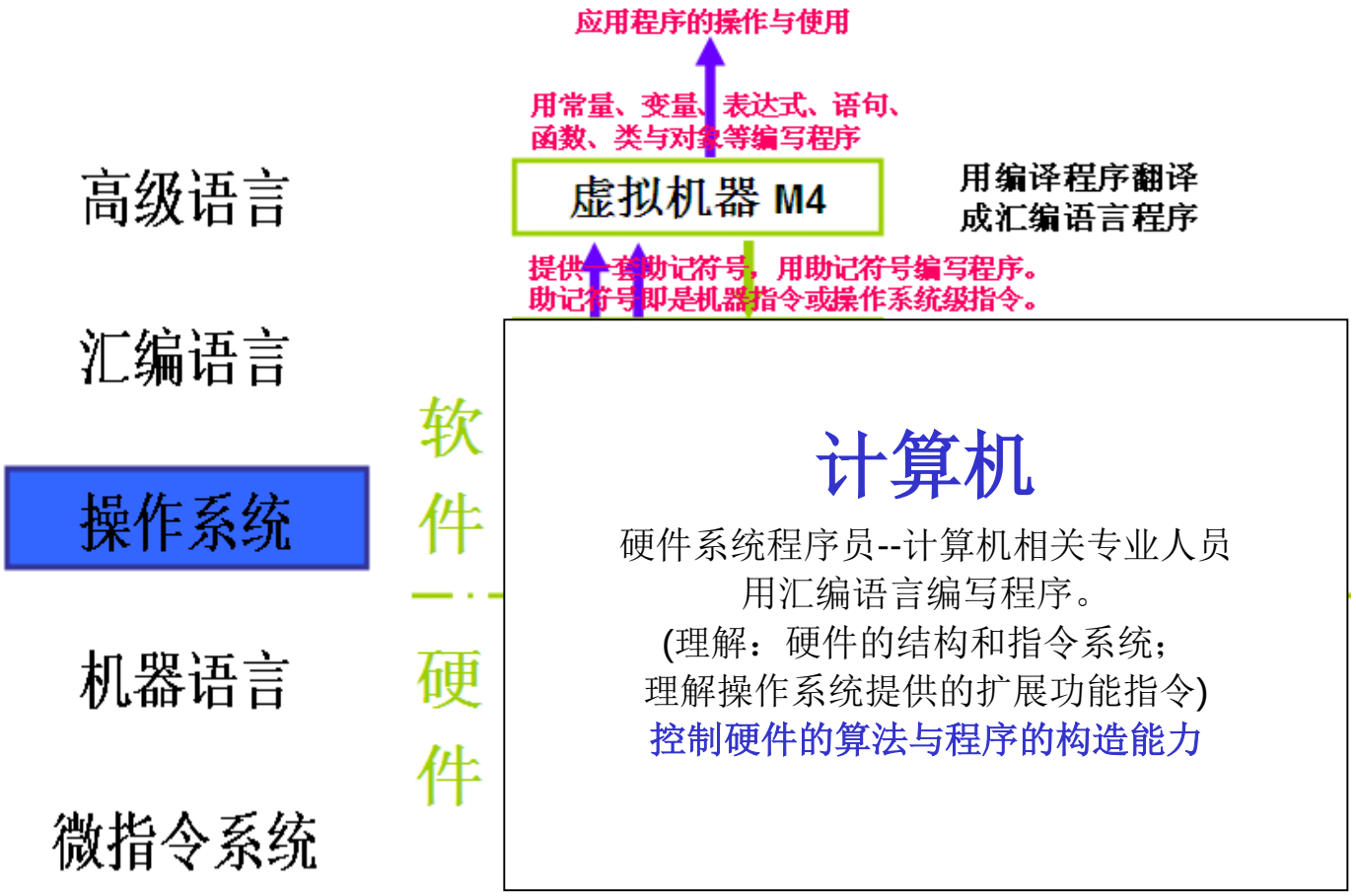
不同抽象层级计算机(虚拟机器)

计算机语言促进了计算机处理能力的不断增强

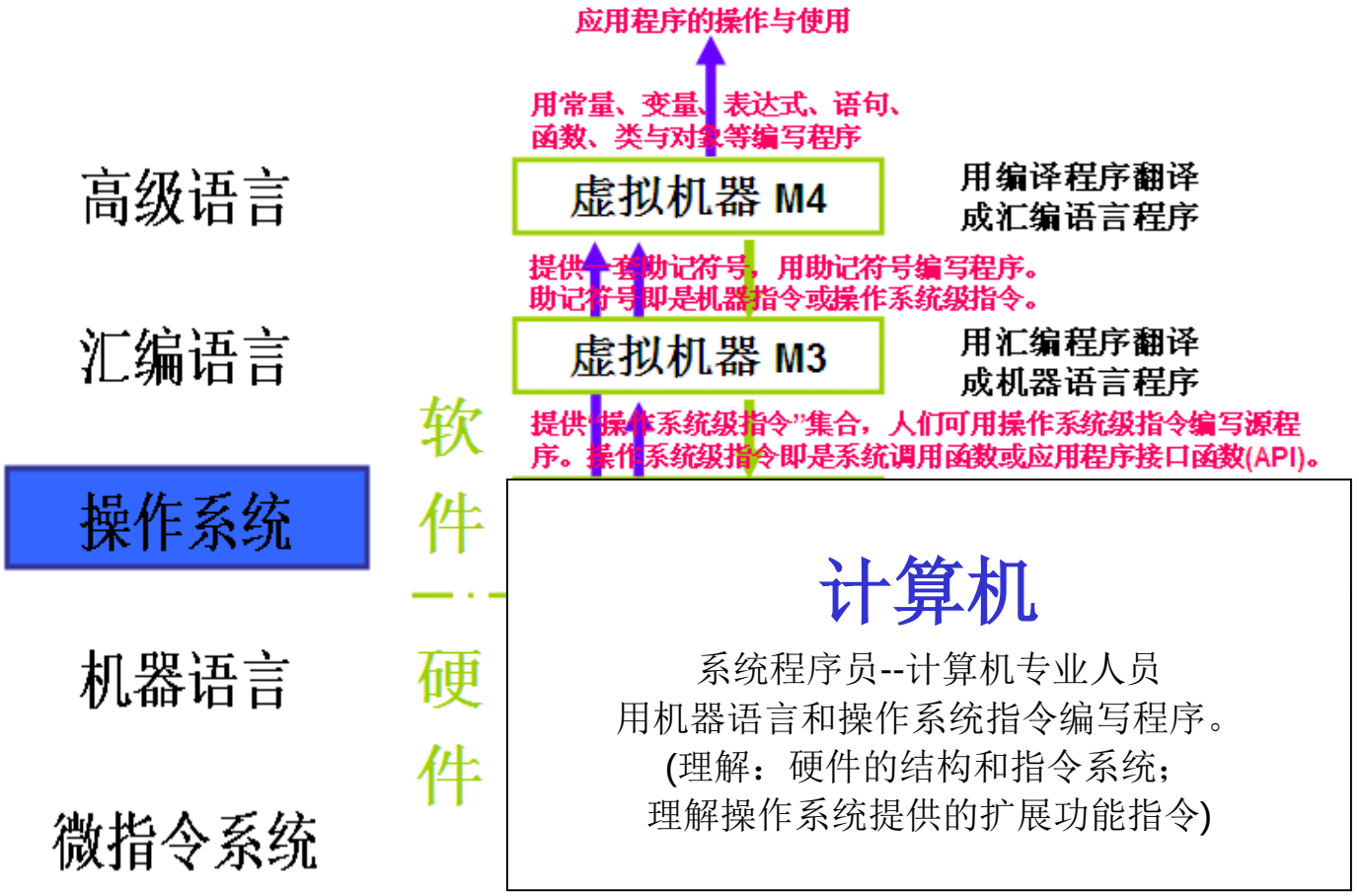
计算机语言促进了计算机处理能力的不断增强



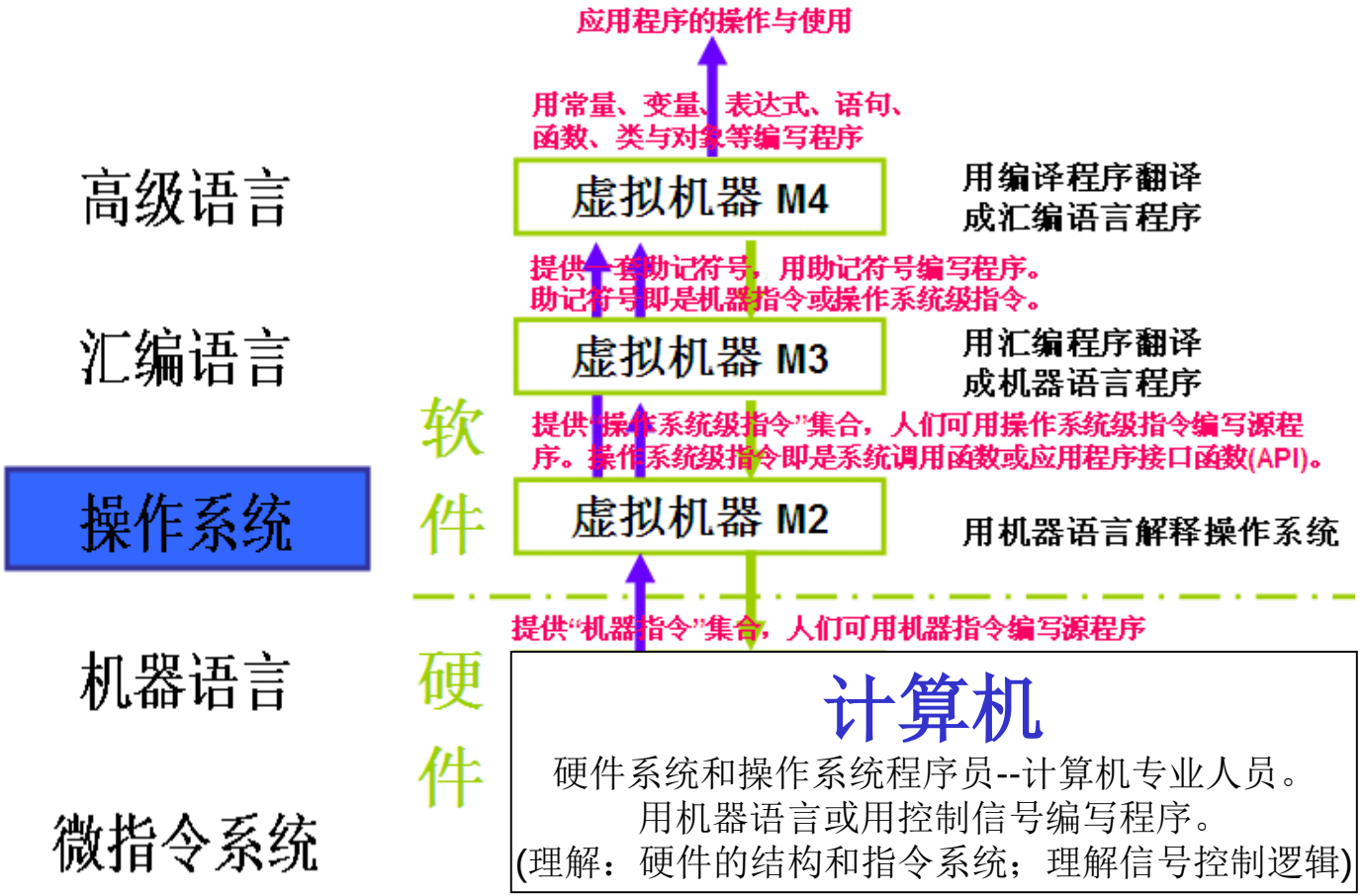
计算机语言促进了计算机处理能力的不断增强



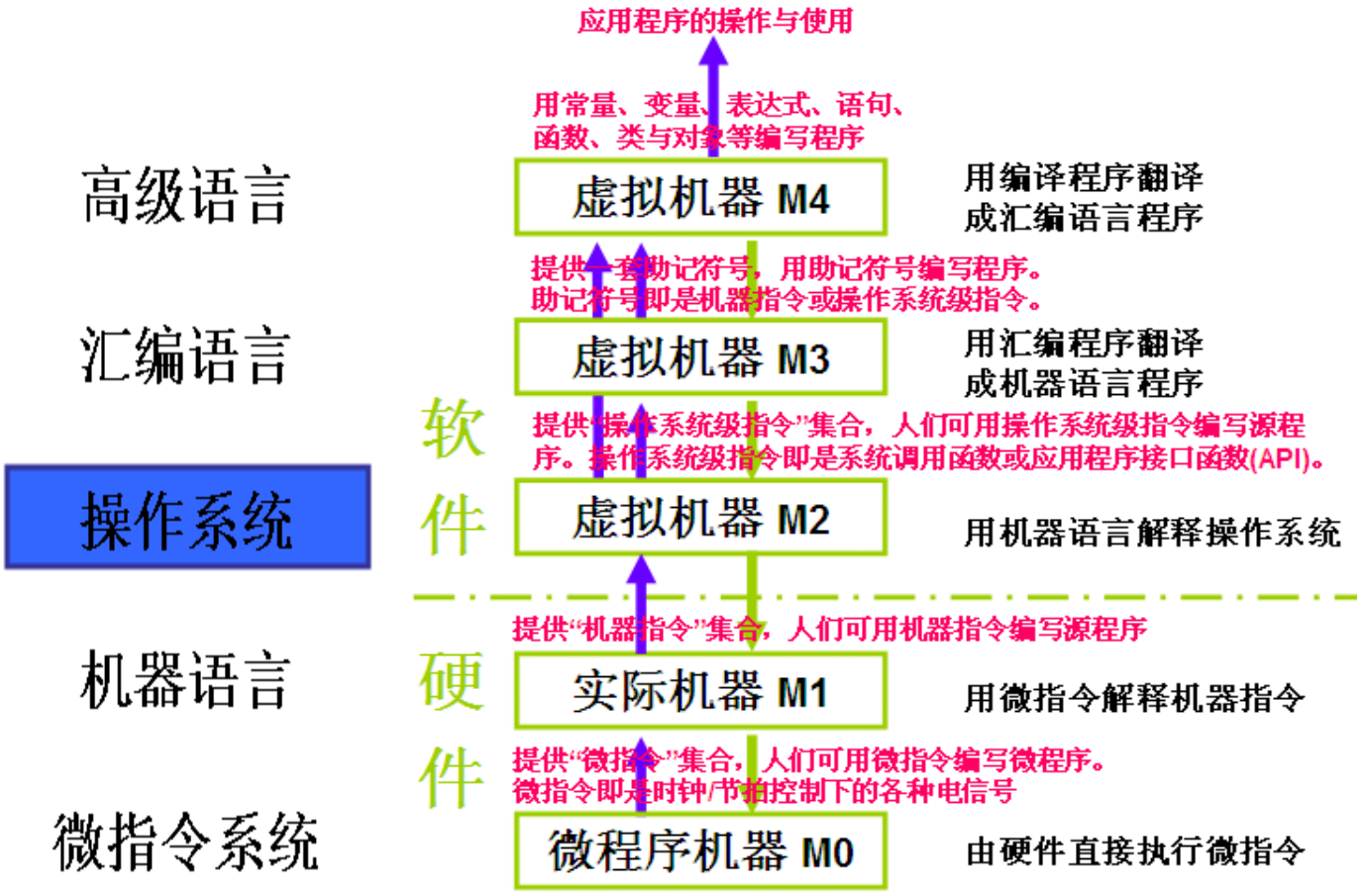
计算机语言促进了计算机处理能力的不断增强



计算机语言促进了计算机处理能力的不断增强

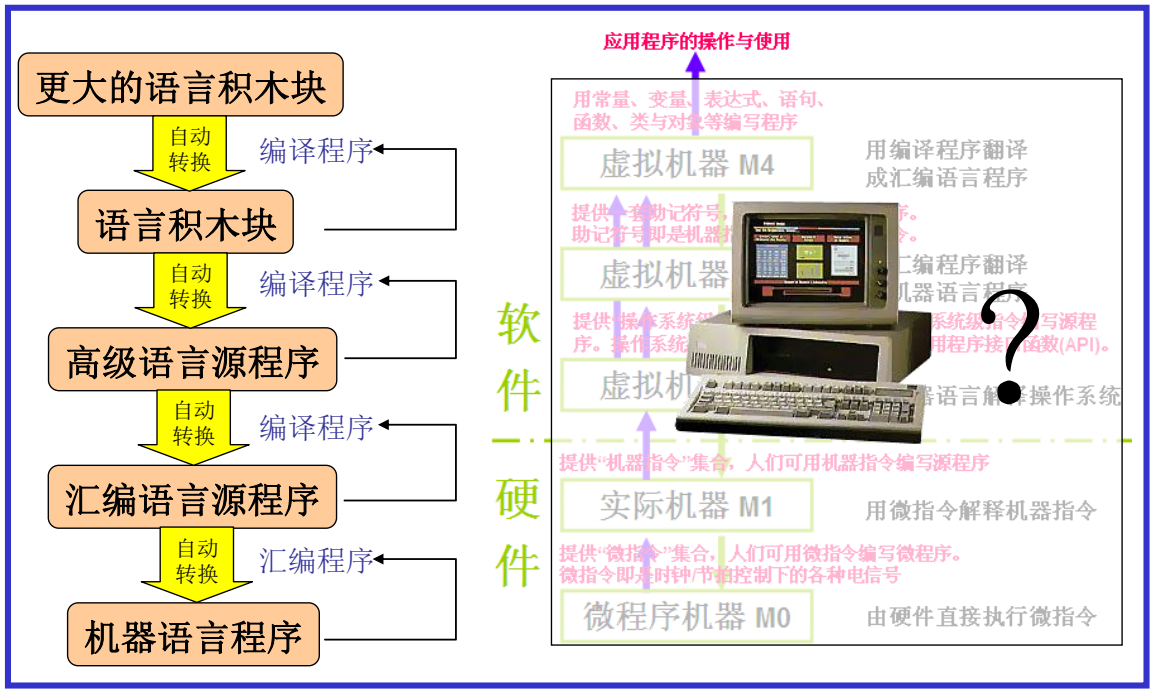


计算机语言促进了计算机处理能力的不断增强



本讲小结

基本目标: 理解如何编写计算机可以执行的程序



基本思维：高级语言与汇编语言➔语言与编译器➔高级语言程序的构成要素➔不同层面的计算机

第5讲 由机器语言到高级语言：程序编写与编译

Questions & Discussion?

2015