第七章 运行时的存储组织及管理

- 7.1 概述
- 7.2 静态存储分配
- 7.3 动态存储分配

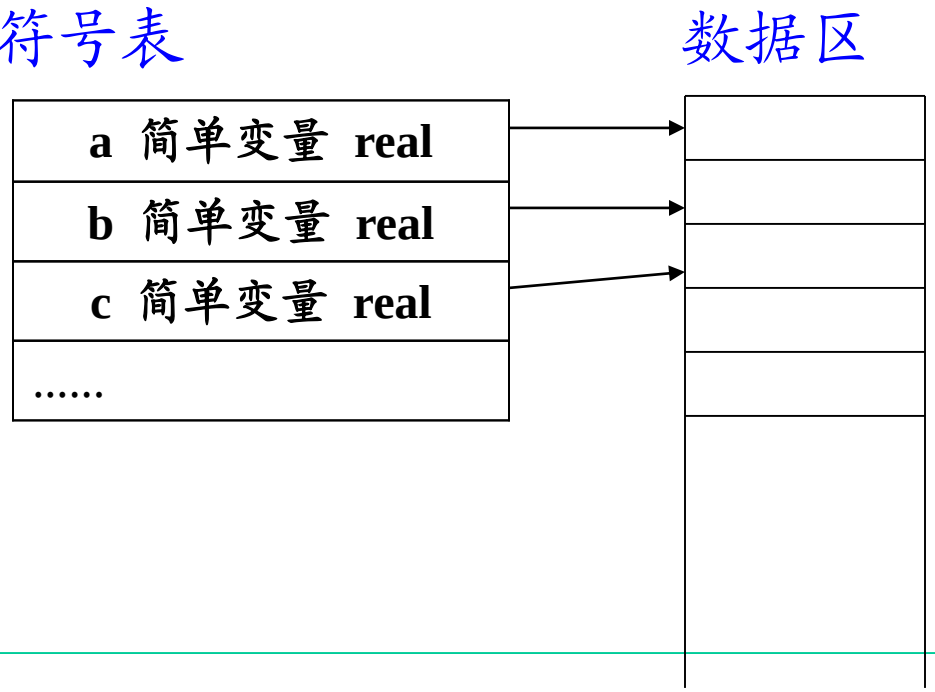
7.1 概述

(1) 运行时的存储组织及管理

目标程序运行时所需要存储空间的组织与管理以及源程序中变量存储空间的分配。

例: real a, b, c ;
...
a:= b*c ;

取b
* c
送 a



(2) 静态存储分配和动态存储分配

静态存储分配

在编译阶段由编译程序实现对存储空间的管理，和为源程序中的变量分配存储的方法。

条件

如果在编译时能够确定源程序中变量在运行时的数据空间大小，且运行时不改变，那么就可以采用静态存储分配方法。

但是并不是所有数据空间大小都能在编译过程中确定！

动态存储分配

在目标程序运行阶段由目标程序实现对存储空间的组织与管理，和为源程序中的变量分配存储的方法。

特点

- 在目标程序运行时进行分配。
- 编译时要生成进行动态分配的目标指令。

7.2 静态存储分配

(1) 分配策略

由于每个变量所需空间的大小在编译时已知，因此可以用简单的方法给变量分配目标地址。

- 开辟一数据区。（首地址在加载时确定）
- 按编译顺序给每个模块分配存储。
- 在模块内部按顺序给模块的变量分配存储，一般用相对地址，所占数据区的大小由变量类型确定。
- 目标地址填入变量的 符号表中。

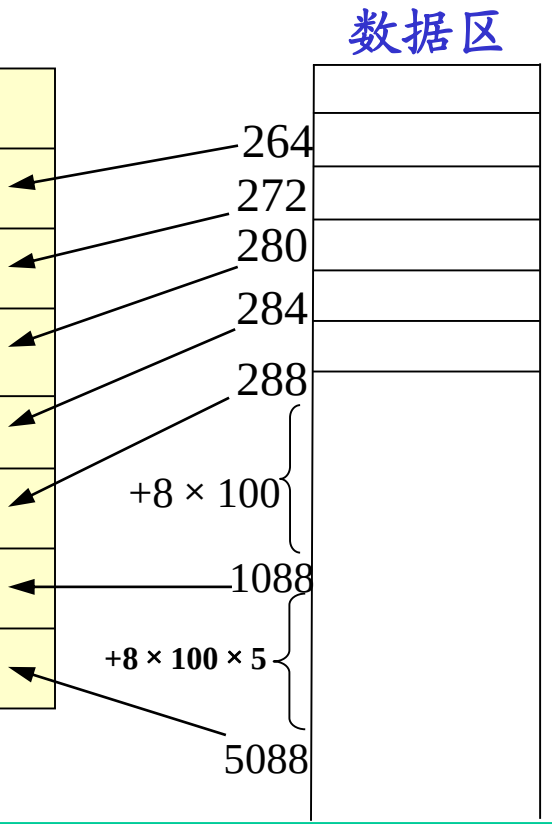
这种分配策略要求语言不允许指针或动态分配，不允许递归调用过程。典型的例子是Fortran77。

例：有下列Fortran 程序段

```
real        MAXPRN, RATE
integer     IND1, IND2
real        PRINT(100),YPRINT(5,100),TOTINT
```

假设整数占4个字节大小，实数占8个字节大小，则符号表中各变量在数据区中所分配的地址为：

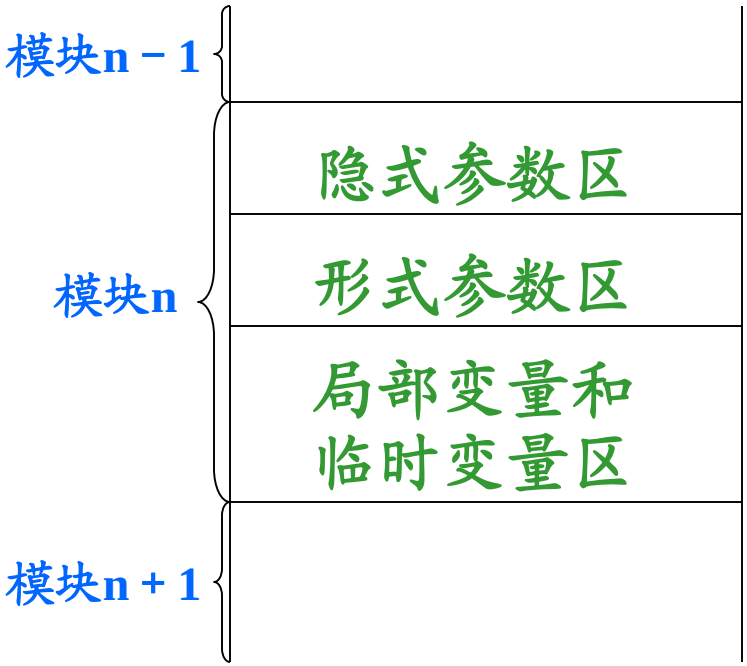
名字	类型	维数	地址
MAXPRN	r	0	264
RATE	r	0	272
IND1	i	0	280
IND2	i	0	284
PRINT	r	1	288
YPRINT	r	2	1088
TOTINT	r	0	5088



(2) 模块（FORTRAN子程序）的完整数据区

编译程序除了要给源程序（模块）中的各种变量分配数据区以外，对子程序来说还要在数据区中保留返回地址，对形式参数也要分配存储以存放相应的实参的信息。另外在编译时还要分配临时变量空间（如存放表达式计算的中间结果等）。

下面给出FORTRAN子程序的典型数据区



隐式参数区：返回地址
函数返回值

形式参数区：存放相应实参信息（值或地址）

7.3 动态存储分配

由于编译时还不能具体确定某些数据空间的大小，故对它们分配存储空间必须在程序运行时进行。这时，编译程序生成有关存储分配的目标代码，实际上的分配要在目标程序运行时进行。这种分配方式称为动态存储分配。

对于分程序结构，而且允许递归调用的语言，常使用栈式动态存储分配，即使用一个类似于堆栈的“运行栈”来实现数据区的分配。

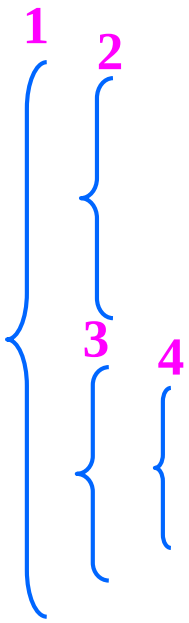
分配策略是：整个数据区为一个堆栈

(1) 当进入一个过程时，在栈顶为其分配一个数据区。

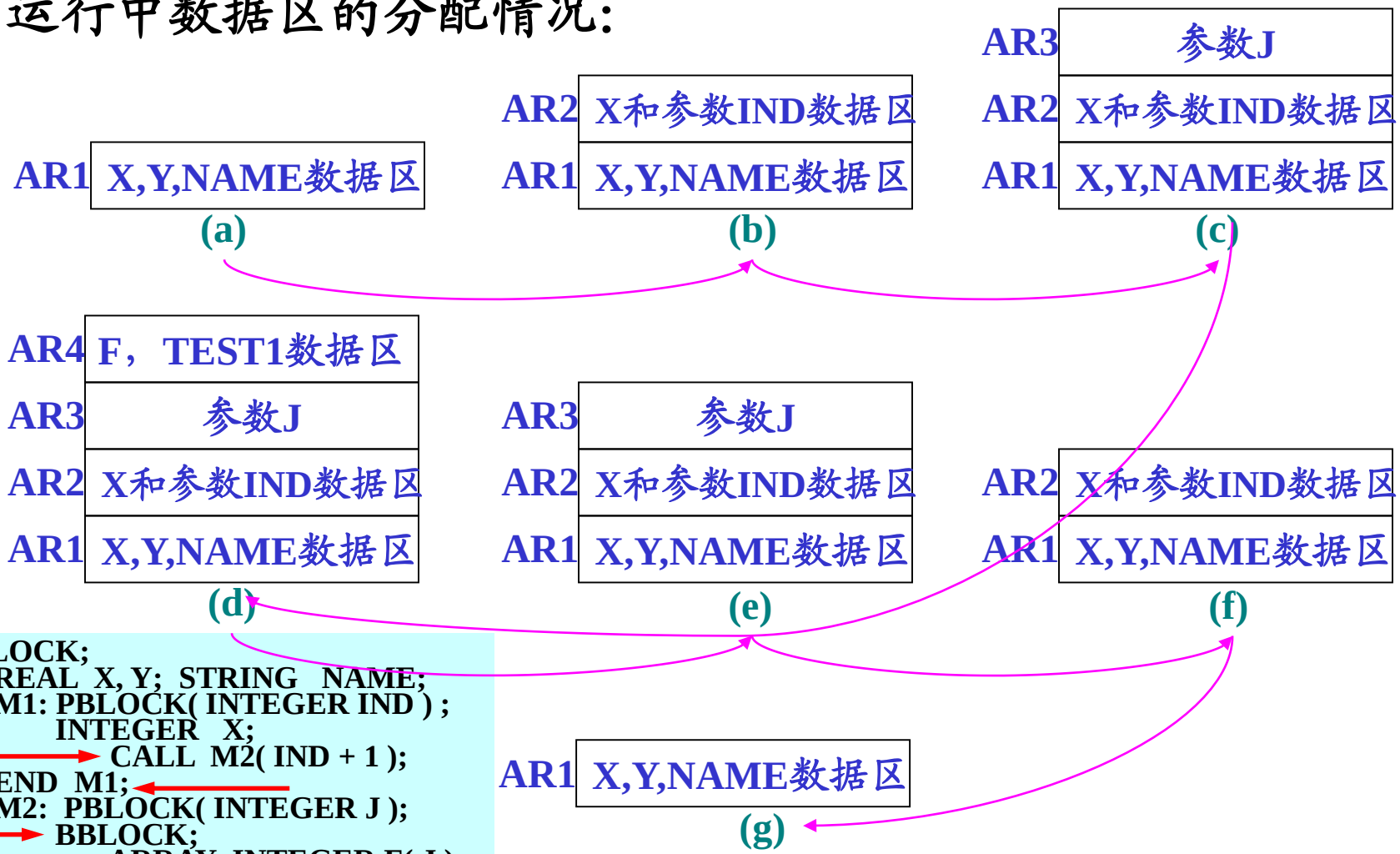
(2) 当退出一个过程时，撤消该过程的数据区。

例1:

```
1   BBLOCK;
        REAL  X, Y;  STRING   NAME;
2       M1: PBLOCK( INTEGER IND ) ;
            INTEGER   X;

            CALL  M2( IND + 1 );

        END  M1;
3       M2:  PBLOCK( INTEGER J );
4           BBLOCK;
                ARRAY  INTEGER F( J );
                LOGICAL TEST1;
                …
            END
        END  M2;

    CALL  M1( X / Y )

END
```



运行中数据区的分配情况:




```
BBLOCK;
    REAL  X, Y;  STRING   NAME;
    M1: PBLOCK( INTEGER IND ) ;
          INTEGER   X;
          CALL  M2( IND + 1 );
    END  M1;
    M2:  PBLOCK( INTEGER J );
         BBLOCK;
              ARRAY  INTEGER F( J );
              LOGICAL TEST1;
         END
    END  M2;
    CALL  M1( X / Y )
END
```

7.3.1 活动记录

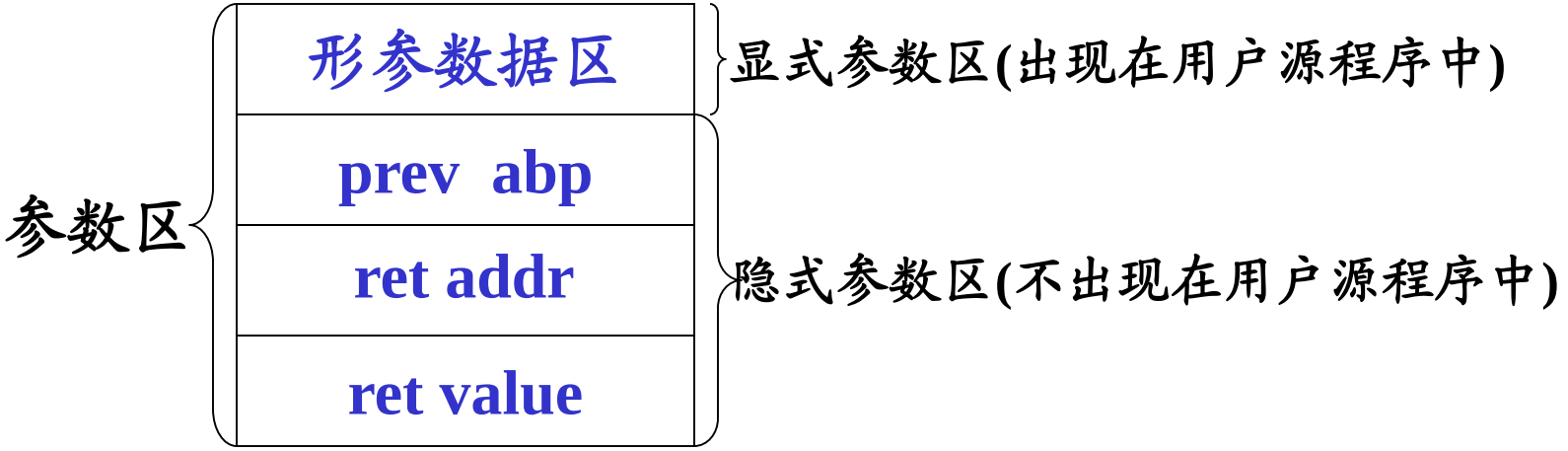
一个典型的活动记录可以分为三部分:

局部数据区
参数区
display区

(1) 局部数据区:

存放模块中定义的各个局部变量。

(2) 参数区： 存放隐式参数和显式参数。



prev abp: 存放调用模块记录基地址。函数执行完时，释放其数据区，数据区指针指向调用前的位置。

ret addr: 返回地址，即调用语句的下一条执行指令地址。

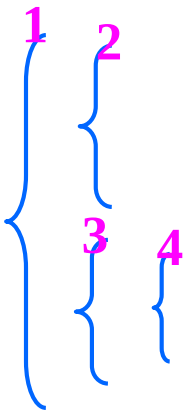
ret value : 函数返回值（无值则空）。

形参数据区：每一形参都要分配数据空间，形参单元中存放实参值或者实参地址。

局部数据区
参数区
display区

(3) display区：存放各外层模块活动记录的基地址。

对于例1中所举的程序段，模块4可以引用模块1和模块3中所定义的变量，故在模块4的display区，应包括AR1和AR3的基地址。



变量二元地址（BL、ON）

BL: 变量声明所在的层次。可以用它找到该层数据区开始地址。

（注意为嵌套层次，并列过程具有相同层次）

ON: 相当于显示参数区开始位置的位移（相对地址）。

例如：程序模块1　　过程块M1

x（1，0）　　IND（2，0）

y（1，1）　　x（2，1）

NAME（1，2）

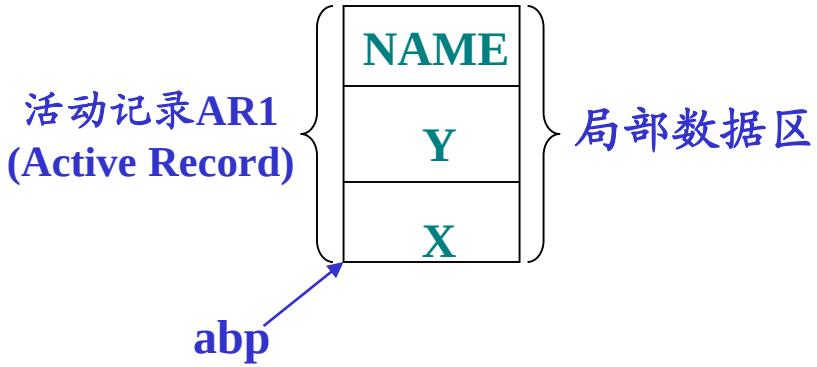
高层（内层）模块可以引用低层（外层）模块中的变量，例如在M1中可引用外层模块中定义的变量Y。

在M1中引用Y时，可通过其display区找到程序块1的活动记录基地址，加上Y在该数据区的相对地址就可以求得y的绝对地址。

```
BBLOCK;
    REAL  X, Y;  STRING   NAME;
(1) M1: PBLOCK( INTEGER IND ) ;
        INTEGER   X;
    (2)
            CALL  M2( IND + 1 );
    END  M1;
    M2:  PBLOCK( INTEGER J );
        BBLOCK;
    (3)     ARRAY  INTEGER F( J );
        (4) LOGICAL TESTI;
        END
    END  M2;
    CALL  M1( X / Y )
END
```

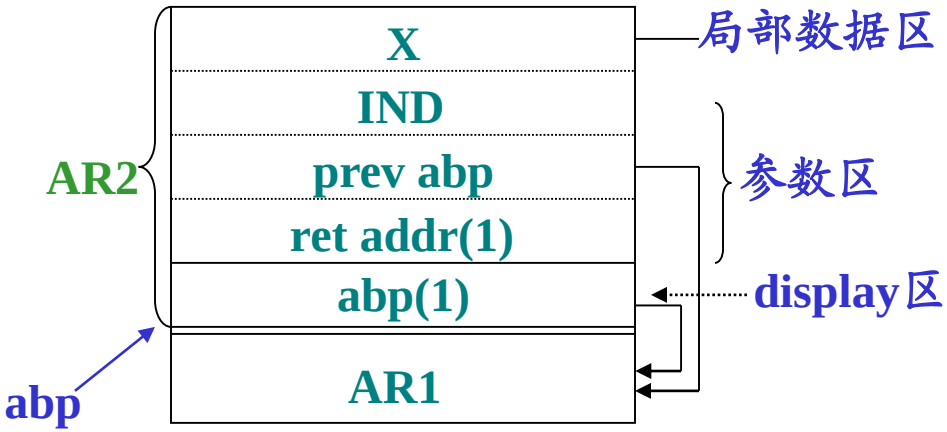
例：下面给出源程序的目标程序运行时，运行栈（数据区栈）的跟踪情况。

```
1  BBLOCK;
       REAL  X, Y;  STRING   NAME;
   2   M1: PBLOCK( INTEGER IND ) ;
           INTEGER   X;
           CALL  M2( IND + 1 );
       END  M1;
   3   M2:  PBLOCK( INTEGER J );
       4       BBLOCK;
                   ARRAY  INTEGER F( J );
                   LOGICAL TESTI;
                    …
               END
       END  M2;
       CALL  M1( X / Y )
   END
```



(a) 进入模块1

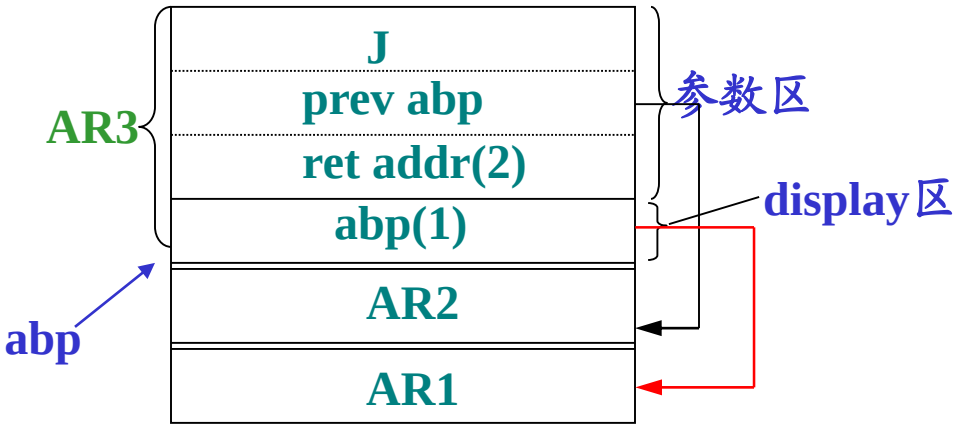
```
1  X, Y,  NAME
   2  M1: (IND);
          X;
      CALL M2;
   3  M2: (J);
      4   ARRAY  F(J);
          TESTI;
   CALL M1
```



(b) M1被调用

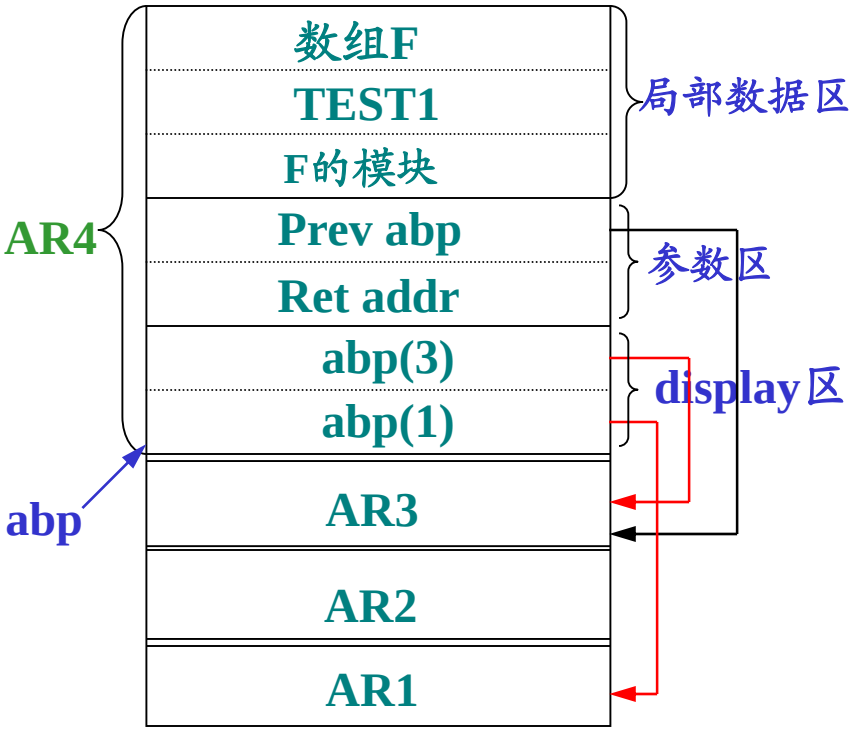
```
1  BBLOCK;
       REAL  X, Y;  STRING   NAME;
   2   M1: PBLOCK( INTEGER IND ) ;
           INTEGER   X;

           CALL  M2( IND + 1 );

       END  M1;
   3   M2:  PBLOCK( INTEGER J );
       4       BBLOCK;
                   ARRAY  INTEGER F( J );
                   LOGICAL TESTI;
                    …
               END
       END  M2;

       CALL  M1( X / Y )

   END
```

```
1  X, Y,  NAME

   2  M1: (IND);
          X;
      CALL M2;

   3  M2: (J);
      4   ARRAY  F(J);
          TESTI;
   CALL M1
```



(c) M2被调用

```
1
BBLOCK;
      REAL  X, Y;  STRING   NAME;
  2
      M1: PBLOCK( INTEGER IND ) ;
            INTEGER   X;

            CALL  M2( IND + 1 );

      END  M1;
  3
      M2:  PBLOCK( INTEGER J );
        4
            BBLOCK;
                  ARRAY  INTEGER F( J );
                  LOGICAL TESTI;
                   …
            END
      END  M2;

      CALL  M1( X / Y )

 END
```



(d) 进入内模块4

```
1
BBLOCK;
      REAL  X, Y;  STRING   NAME;
  2
      M1: PBLOCK( INTEGER IND ) ;
            INTEGER   X;

            CALL  M2( IND + 1 );

      END  M1;
  3
      M2:  PBLOCK( INTEGER J );
        4
            BBLOCK;
                  ARRAY  INTEGER F( J );
                  LOGICAL TESTI;
                   …
            END
      END  M2;

      CALL  M1( X / Y )

 END
```

地址	内容	说明
19	数组F	程序进入第4个模块后。
18	TEST1	
17	F的模块	
16	Prev abp	
15	Ret addr	
14	abp(3)	
abp(4) → 13	abp(1)	
12	J	在模块2中通过执行CALL M2(IND+1)进入第3个模块，但尚未进入第4个内层模块时。
11	Prev abp	
10	Ret addr	
abp(3) → 9	abp(1)	
8	X	通过最外层模块中调用CALL M1(X/Y)进入第2个模块，尚未执行CALL M2(IND+1)语句。
7	IND	
6	Prev abp	
5	Ret addr	
abp(2) → 4	abp(1)	
3	NAME	系统进入第1个模块，尚未执行到CALL M1(X/Y)时
2	Y	
abp(1) → 1	X	

	main的数据	进入main函数的情况
	（空）	
	OS	
abp(0) →	（空）	

```
1
BBLOCK;
      REAL  X, Y;  STRING   NAME;
  2
      M1: PBLOCK( INTEGER IND ) ;
          INTEGER   X;

          CALL  M2( IND + 1 );

      END  M1;
  3
      M2:  PBLOCK( INTEGER J );
        4
            BBLOCK;
                  ARRAY  INTEGER F( J );
                  LOGICAL TESTI;
                   …
            END
      END  M2;

      CALL  M1( X / Y )

 END
```

（e）当模块4执行完，则abp:=prev abp，这样abp恢复到进入模块4时情况，运行栈情况如（c）。

（f）当M2执行完，则abp:=prev abp，这样abp恢复到进入模块M2时情况，运行栈情况如（b）。

（g）当M1执行完，则abp:=prev abp，这样abp恢复到进入模块M1时情况，运行栈情况如（a）。

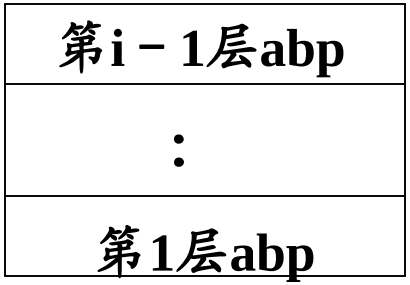
（h）当最外层模块执行完，运行栈恢复到进入模块时的情况，运行栈空。

7.3.2 建造display区的规则

从i层模块进入(调用) j 层模块:

(1) 若j = i + 1

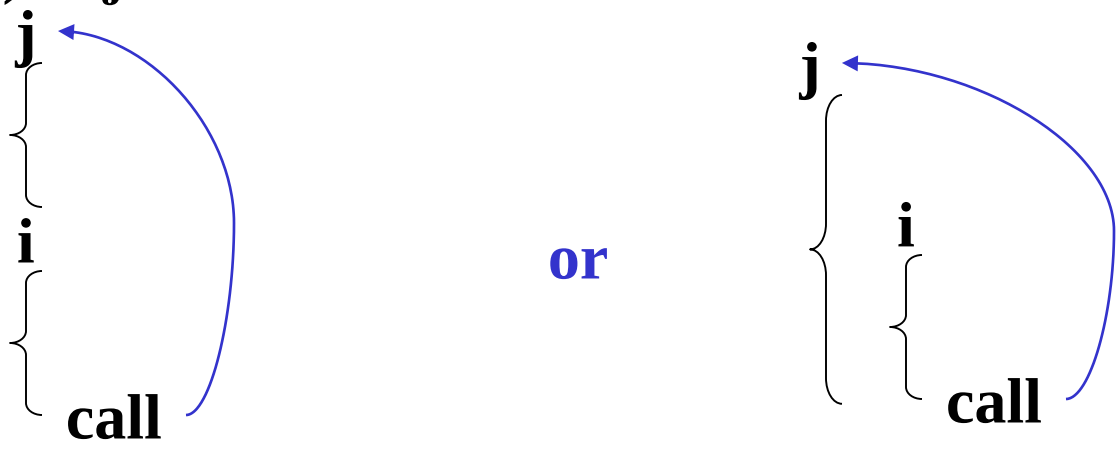
复制 i 层的display，然后增加一个指向 i 层模块记录基地址的指针



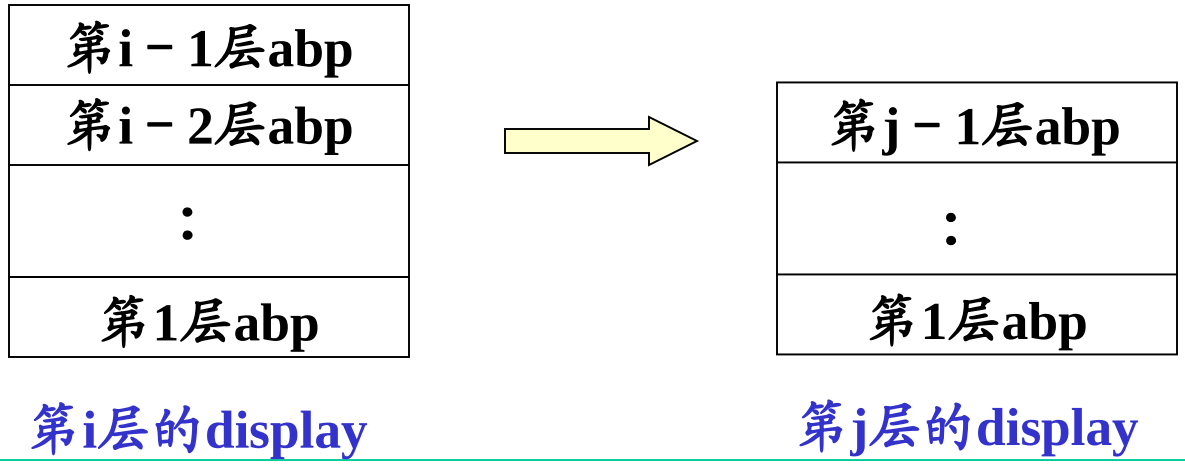
i层模块的display

j 层模块的display

(2) 若j≤i 即调用外层模块或同层模块



将i层模块的display区中的前面j－1个复制到第j层模块的display区



7.3.3 运行时的地址计算

假设要访问的变量的二元地址为：（BL，ON）

- 在LEV层模块中引用BL层模块定义的变量

display区大小　　隐式参数区大小

地址计算公式:

```
if  BL = LEV  then
        addr := abp + (BL - 1) + nip + ON
else if  BL < LEV  then
        addr := display[BL] + (BL - 1) + nip + ON
else
        write("地址错误，不合法的模块层次")
```

作业：P133　第2题