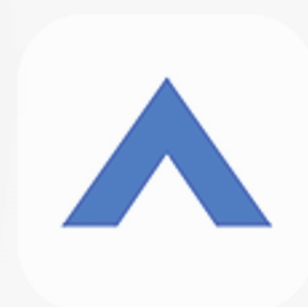




1. CPU的设计与结构



函数调用的底层处理

船说：计算机基础



1. CPU的设计与结构

1.6.3 函数调用的CPU底层处理

01 函数的调用与返回（r14的作用）

02 函数参数的传递

03 函数的返回值

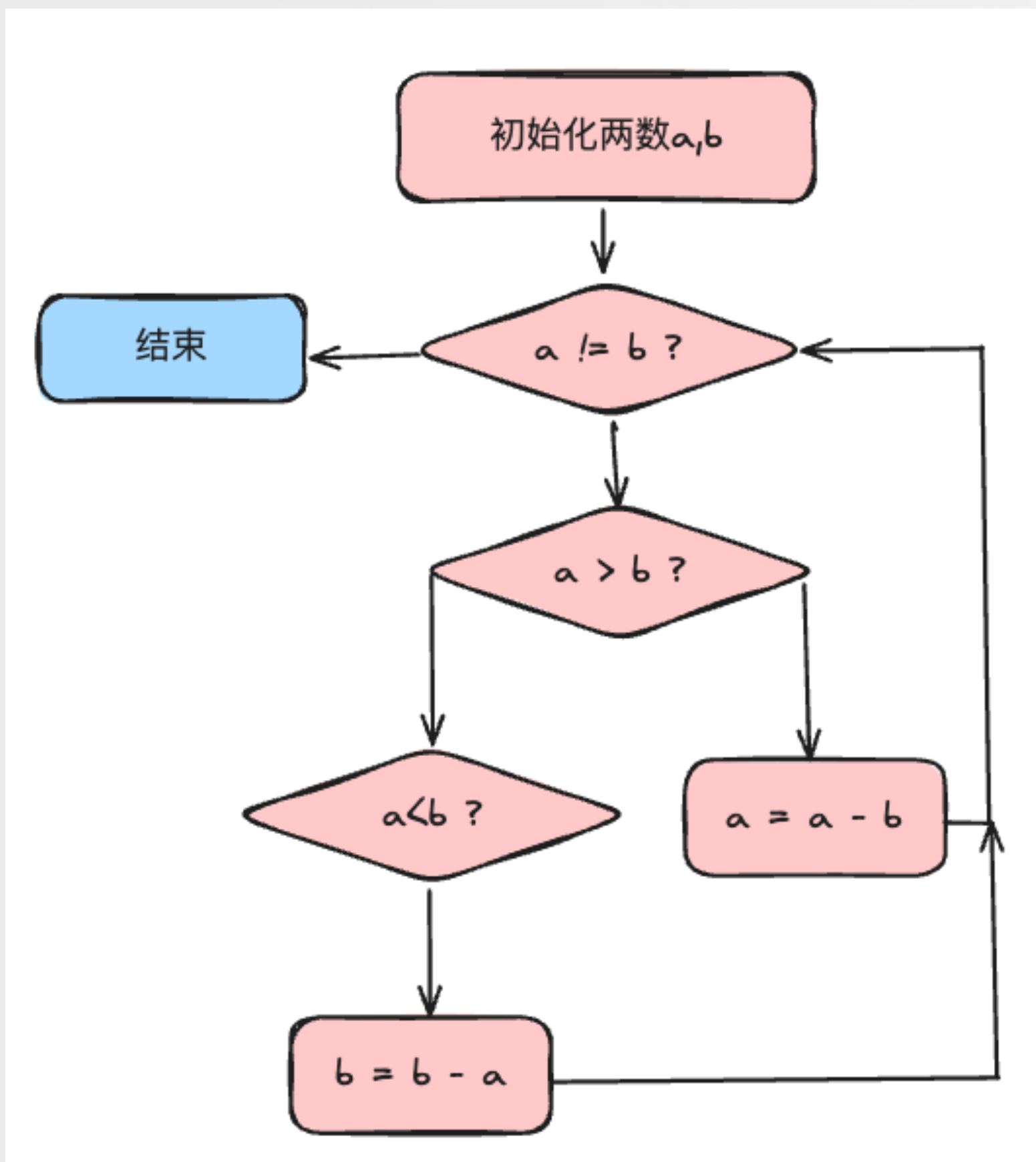
04 函数中的临时变量





1. CPU的设计与结构

补充函数的概念



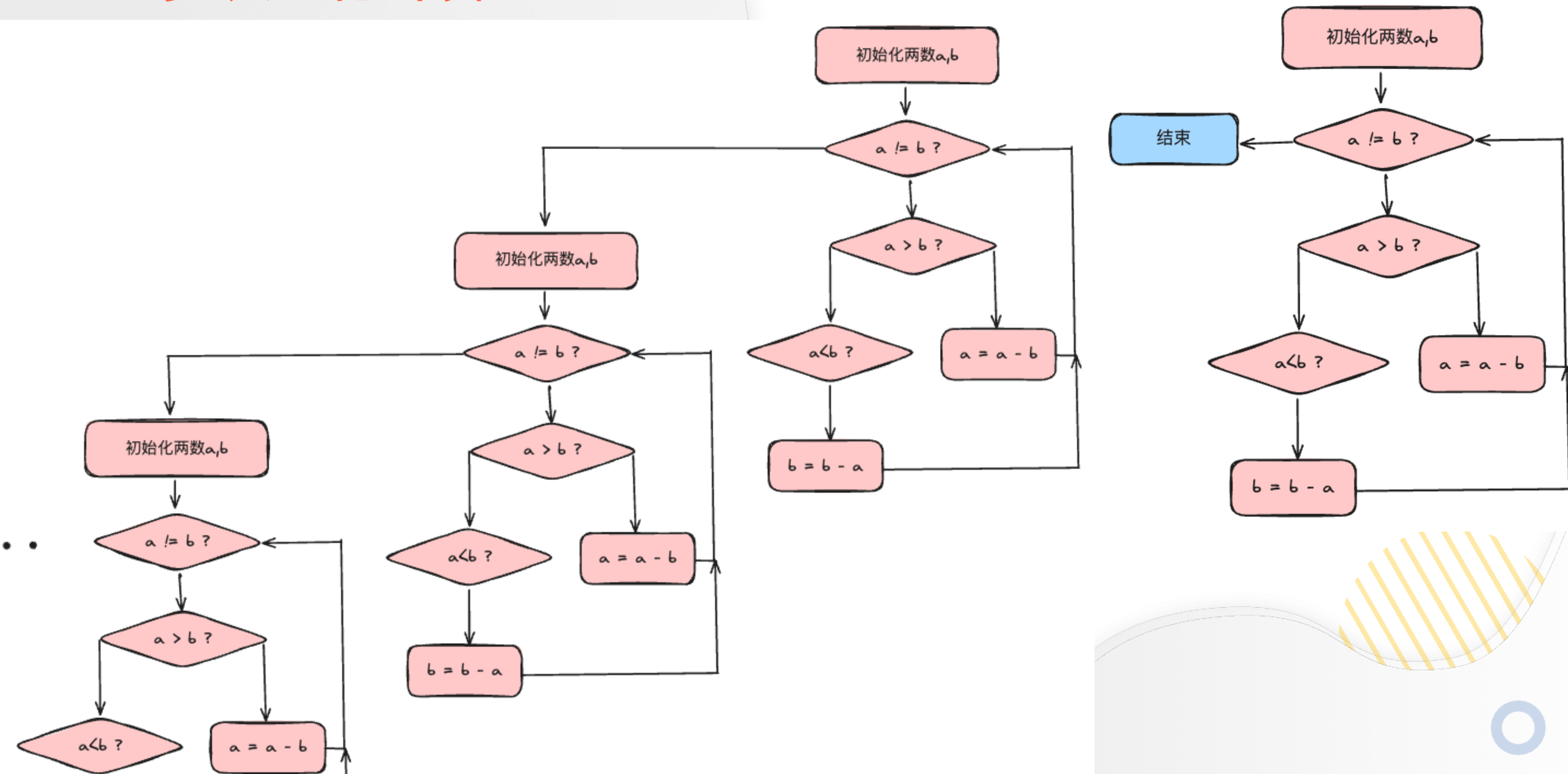
问题：如果有多组数据需要计算出最大公约数，怎么办？





1. CPU的设计与结构

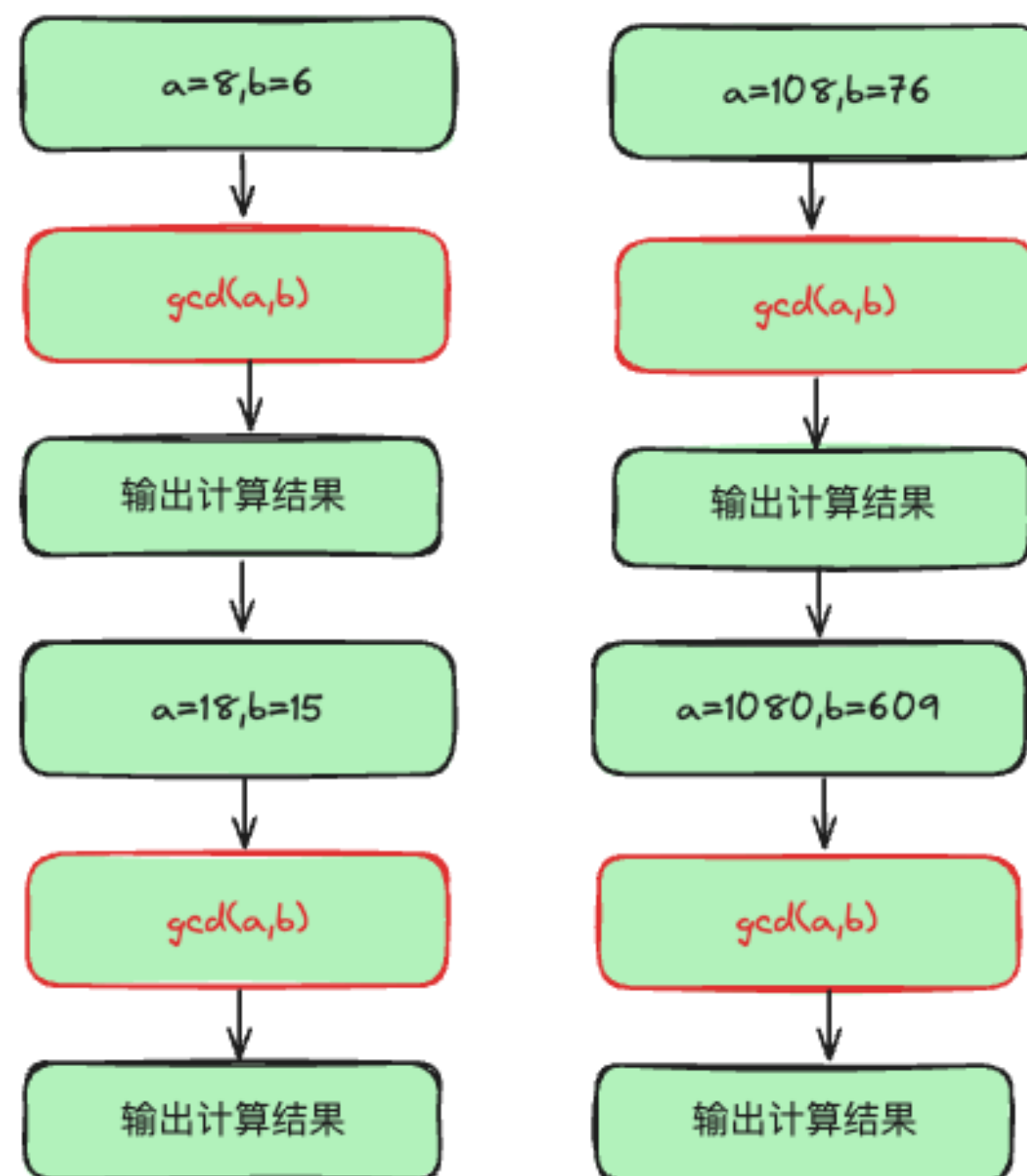
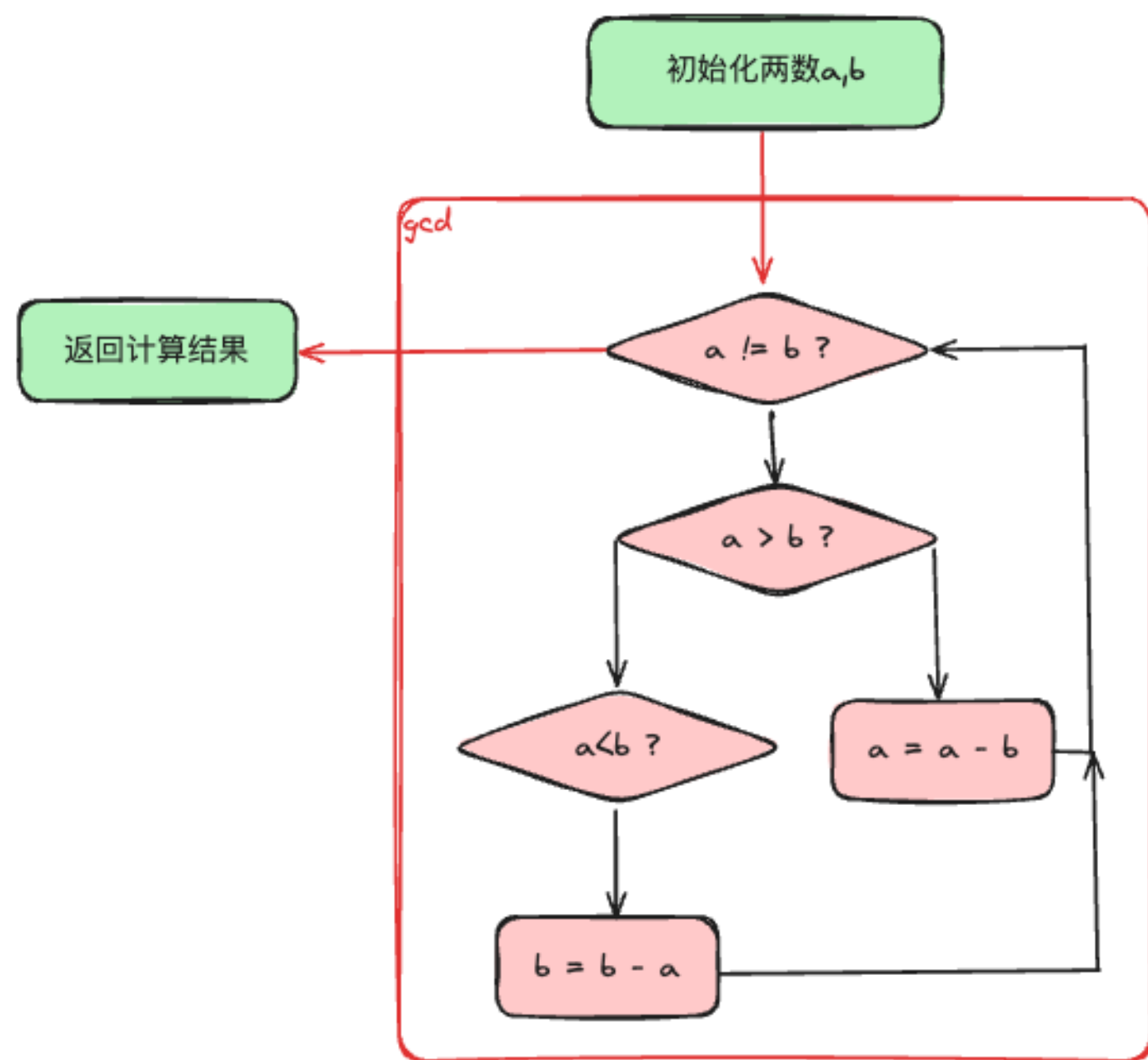
多次进行计算





1. CPU的设计与结构

封装成一个功能模块，重复使用





1. CPU的设计与结构

这个功能模块，通常叫函数

```
1  #include <stdio.h>
2  int gcd(int x , int y)
3  {
4      while(x != y){
5          if( x > y)
6              x = x - y;
7          else
8              y = y - x;
9      }
10     return x;
11 }
12
13 int main()
14 {
15     int a = 18;
16     int b = 15;
17     int result;
18     result = gcd(18,15);
19     printf("gcd(%d,%d) = %d \n", a,b,result);
20
21     return 0;
22 }
```



1. CPU的设计与结构

没有参数的调用与返回

1. bl 指令同时影响两个寄存器pc (r15) 和 lr(r14)
2. mov r15,r14 指令将把lr里保存的指令地址恢复到pc中，实现函数返回。



1. CPU的设计与结构

函数参数传递

1. 少于等于三个参数时，直接使用r0,r1,r2存放第一、第二、第三个参数。
2. 再多的参数通过保存到栈空间，使用时从栈空间取出。



1. CPU的设计与结构


函数返回数据处理

1. 有返回值，通过r0寄存器传递。



1. CPU的设计与结构

寄存器中的数据保存

1. 为了避免影响，在函数内使用的寄存器，使用前要先保存，使用后返回前要恢复
2. 如果函数内还要调用函数，要先对lr进行保存，
 然后再使用bl 跳转，第二级调用返回后先恢复lr，再进行第一级调用的返回。



1. CPU的设计与结构

数据保存到内存

1. 先要知道保存到内存的位置（即地址）
2. 使用STR可以把寄存器的数据保存到指定内存
3. 使用LDR可以将指定内存中的数据读到寄存器



1. CPU的设计与结构

1.6.3 本节总结

1. 使用bl指令在跳转的同时会保存跳转前将要执行的下一条指定
2. 函数的返回，其实就是将bl跳转时保存的地址给PC
3. 如果要在函数中使用使用寄存器，需要先保存原来的值，并在返回前恢复这些值
4. 一般情况下，在处理C语言函数的返回值时，默认使用r0保存返回值。



欢迎参与学习

WELCOME FOR YOUR JOINING

船说：计算机基础