第 13 课
指针与链表

薛浩

xuehao0618@outlook.com

阅读

- Programming Abstraction in C++ *Chapter 11,12*

抽象图谱


用户
队列
push()
pop()
抽象数据类型
优先队列
push()
pop()
实现
有序
数据组织策略
二叉
数组
动态内存管理
链表
硬件

今日话题

- 内存与指针
- 链表

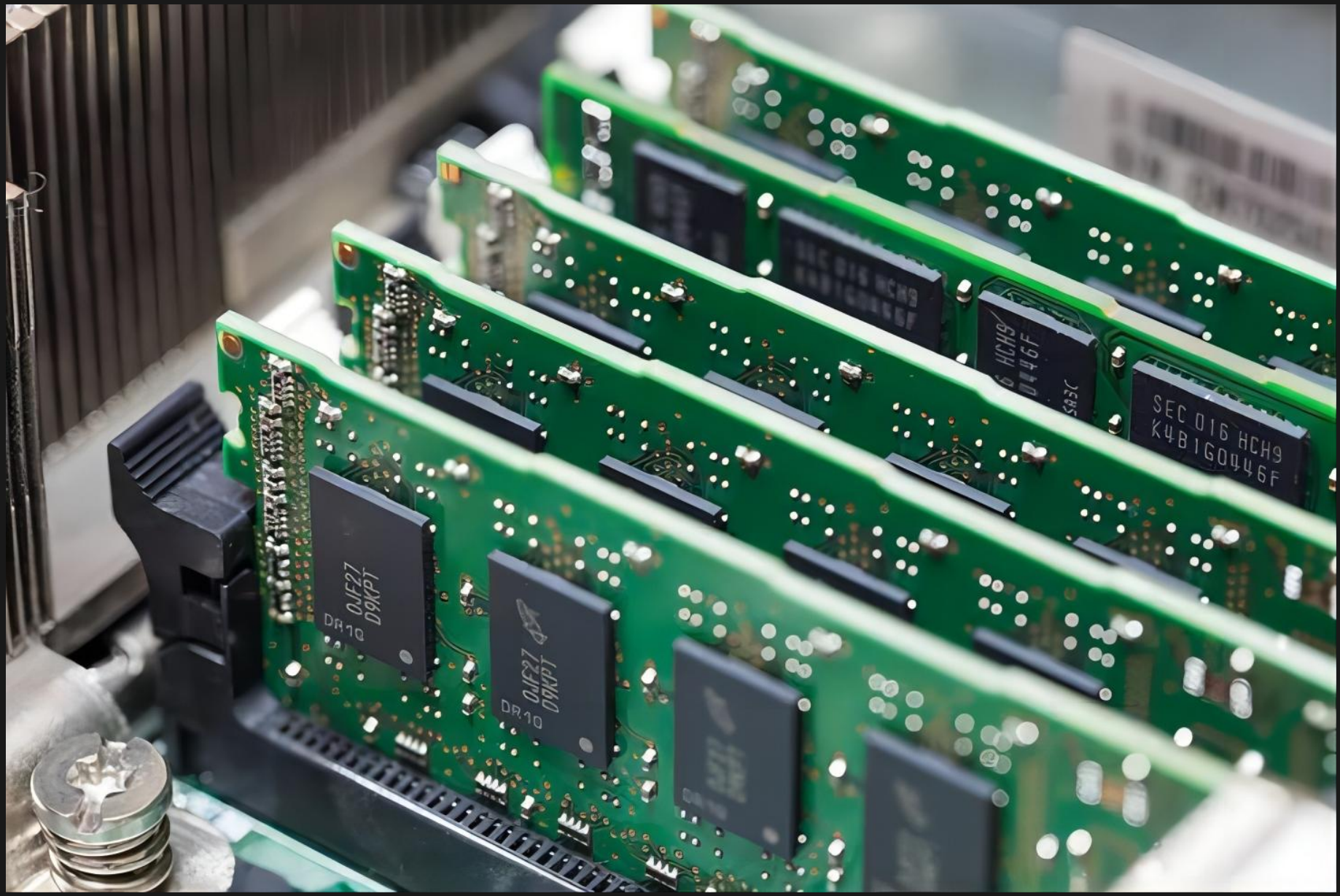
内存与指针

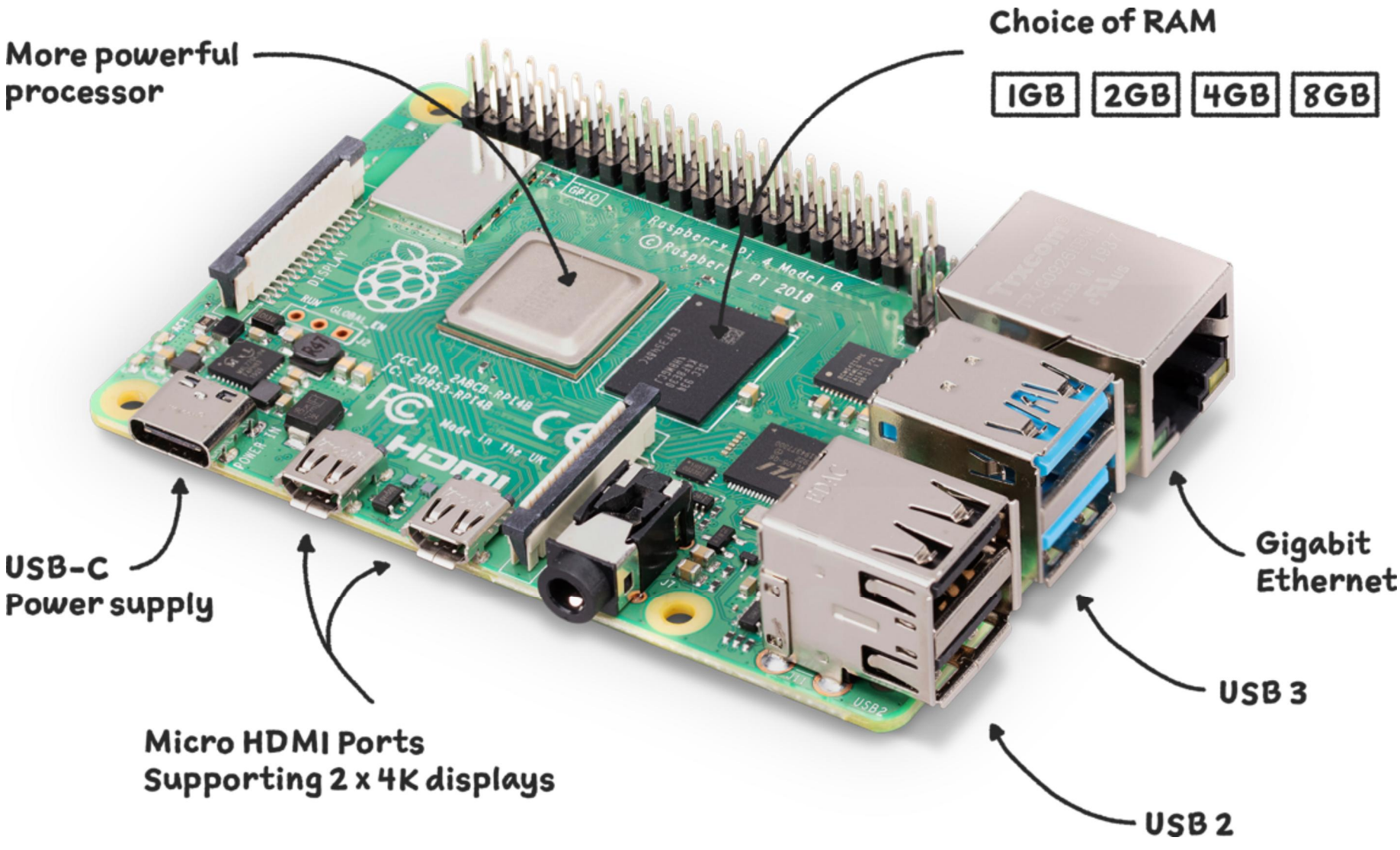
内存

编程的主要过程，就是和内存打交道的过程。

内存

- 代码转变为指令
- 程序执行
- 定义变量
- 读取硬盘文件

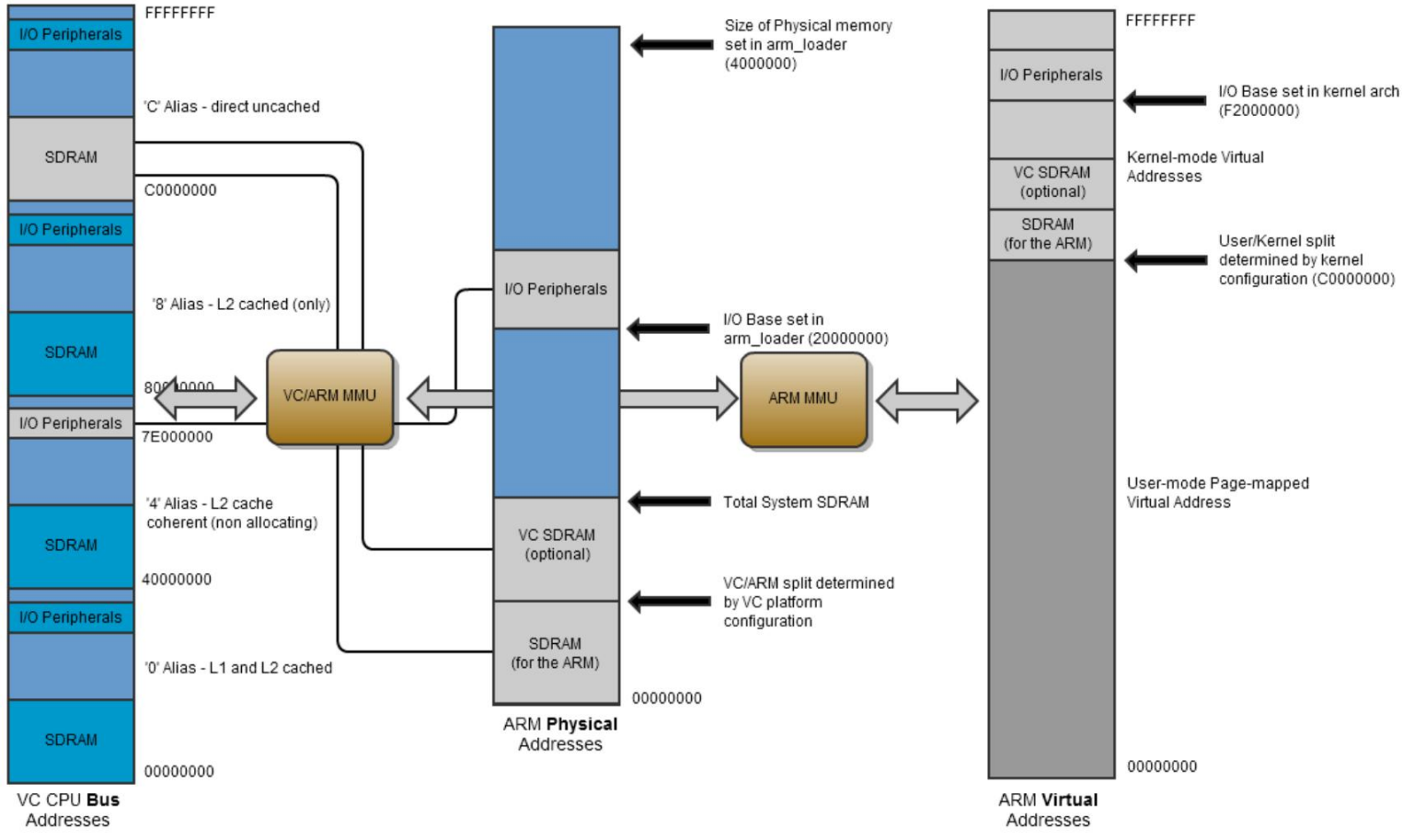

SEC 016 HCH9
K4B1G0446F
OJF27
D9KPT
DA10
OJF27
D9KPT
DA10


More powerful processor
Choice of RAM
1GB
2GB
4GB
8GB
USB-C Power supply
Micro HDMI Ports Supporting 2 x 4K displays
Gigabit Ethernet
USB 3
USB 2

内存

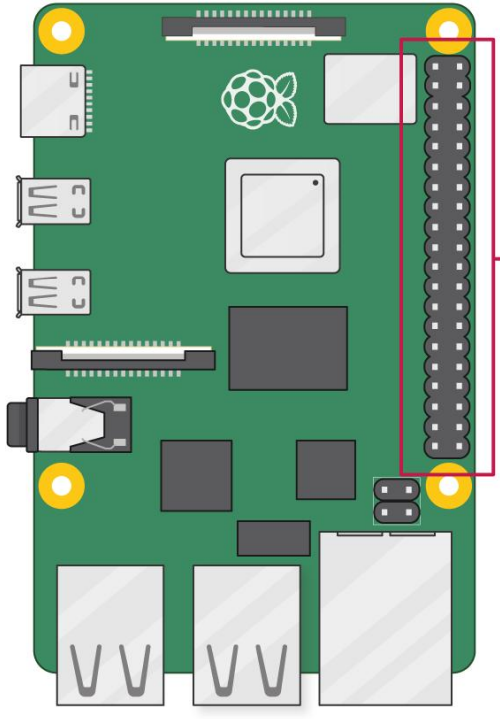
- 由一些比特位构成，单个比特位没什么意义
- 以 8 个位组成的字节块为最小处理单元
- 每个字节有独立编号，用十六进制表示

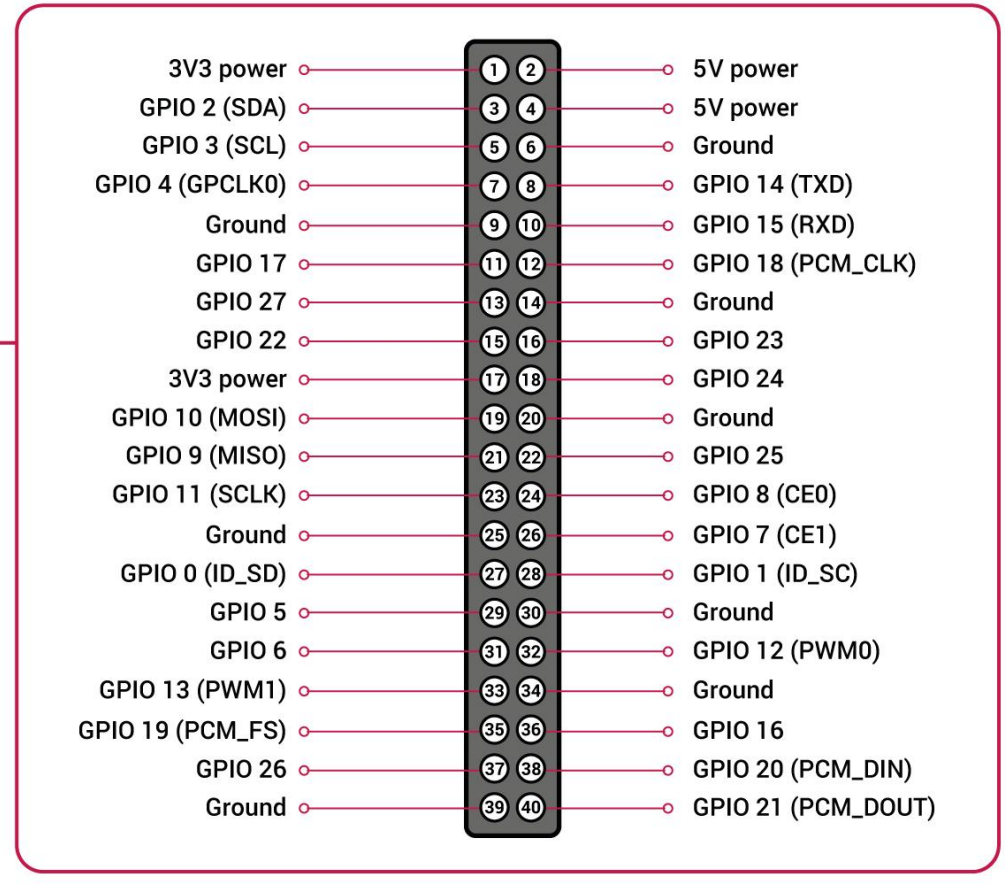
BCM2835 ARM Peripherals

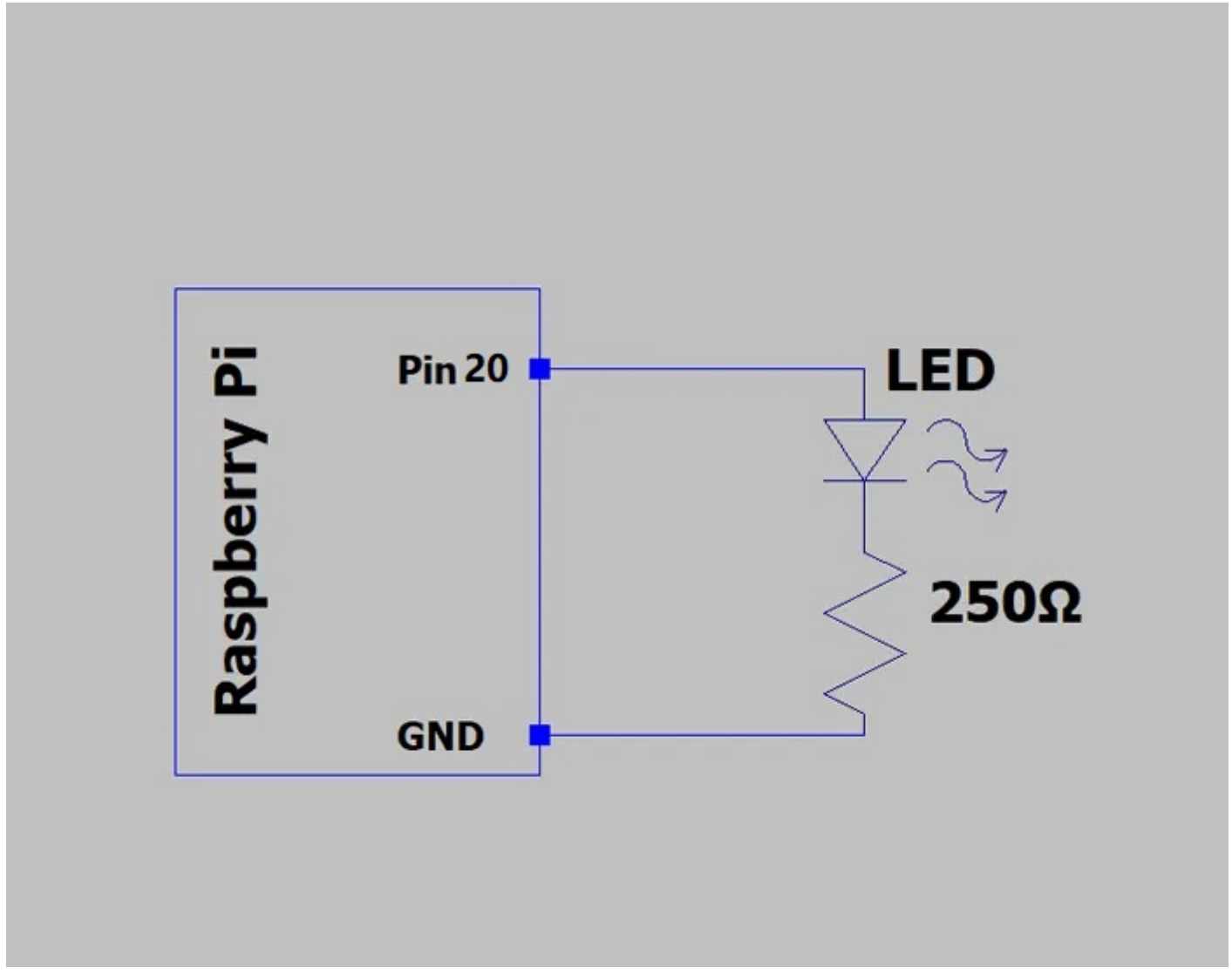


指针

指针是一种特殊数据类型，用于抽象内存的地址。
通过指针才能操作内存的位置。




3V3 power 1 2 5V power
GPIO 2 (SDA) 3 4 5V power
GPIO 3 (SCL) 5 6 Ground
GPIO 4 (GPCLK0) 7 8 GPIO 14 (TXD)
Ground 9 10 GPIO 15 (RXD)
GPIO 17 11 12 GPIO 18 (PCM_CLK)
GPIO 27 13 14 Ground
GPIO 22 15 16 GPIO 23
3V3 power 17 18 GPIO 24
GPIO 10 (MOSI) 19 20 Ground
GPIO 9 (MISO) 21 22 GPIO 25
GPIO 11 (SCLK) 23 24 GPIO 8 (CE0)
Ground 25 26 GPIO 7 (CE1)
GPIO 0 (ID_SD) 27 28 GPIO 1 (ID_SC)
GPIO 5 29 30 Ground
GPIO 6 31 32 GPIO 12 (PWM0)
GPIO 13 (PWM1) 33 34 Ground
GPIO 19 (PCM_FS) 35 36 GPIO 16
GPIO 26 37 38 GPIO 20 (PCM_DIN)
Ground 39 40 GPIO 21 (PCM_DOUT)


Raspberry Pi
Pin 20
GND
LED
250Ω

GPIO 20

- 0x20200008
- 0x2020001c
- 0x20200028

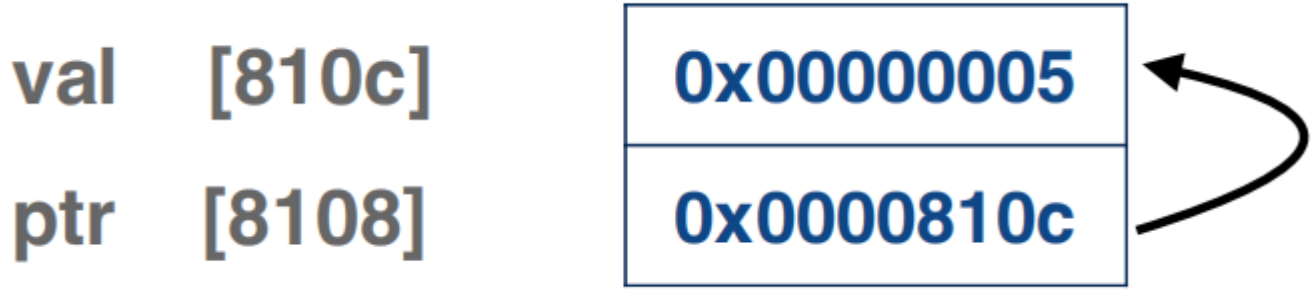
BLINK

```
unsigned *FSEL2 =  (unsigned *)0x20200008;
unsigned *SET0  =  (unsigned *)0x2020001c;
unsigned *CLR0  =  (unsigned *)0x20200028;

void main(void) {
    *FSEL2 = 1;             // 配置 GPIO 20 为输出
    while (true) {
        *SET0 = 1 << 20; // 设置 GPIO 20 为高电平
        delay(1000);        // 等待 1 秒
        *CLR0 = 1 << 20; // 设置 GPIO 20 为低电平
        delay(1000);        // 等待 1 秒
    }
}
```

指针

- 指针也是变量，也要占据内存空间
- 指针占用的空间为一个字（word）
  - 32 位系统中是 4 Byte
  - 64 位系统中是 8 Byte




```
int val = 5;
int *ptr = &val;
```

指针语法

- 指针的声明：使用 typeName * 加变量名。

```
int * val;      // 声明一个 int 指针
char * val;     // 声明一个 char 指针
string * val;   // 声明一个 string 指针
```

- 指针的赋值：空指针 nullptr；使用 & 获取地址

```
// 空指针
char * val = nullptr;
// 取地址
int val = 5;
int *ptr = &val;
```

取地址操作符 &

和引用操作符 & 功能是不一样的

- 记录动态内存块：记录 new 分配的动态内存

```
// 动态分配 5 个 int 大小的堆空间
int* elements = new int[5];
// 动态分配 1 个 Point 大小的堆空间
int* data = new Point;
```

- 删除动态内存块：通过指针删除动态内存

```
// 收回动态分配的数组空间
delete[] elements;
// 收回动态分配的单个空间
delete data;
```

- 读/写变量的值（解引用）：同样使用 * 操作符

```
int val = 5;
int *ptr = &val;
*ptr = 10;          // 修改变量的值，此时 val 内存放的值为 10
int val2 = *ptr;    // 取出 val 内的值 10 初始化另一个变量 val2
```

解引用 * 操作符

无法对空指针作取值操作

今日话题

- ~~内存与指针~~
- 链表

链表

链表

链表和数组都是一种数据结构，

用于存储元素序列。

链表

- 每个元素分开存储
- 元素之间彼此相连（指针）
- 链表结尾有一个特殊标记（空指针）

Linked List Visualization

链表 VS 数组

- [x] 链表元素非连续，比数组更灵活
- [x] 链表更方便添加/删除元素
- [x] 不需要扩容和复制数据
- [ ] 数组可以随机访问
- [ ] 数组占用空间更少

节点的表示

```
struct Node {
    int num;
    Node* next;
}
```

POINT

```
class Point {
public:
    Point *next{nullptr}; // 用于链接另一个 Point 对象

    Point();
    Point(int x, int y);
    int getX();
    int getY();
    string toString() const;
private:
    int x;
    int y;
};
```

小试牛刀

```
Point* list;
list = nullptr;
list = new Point;
cout << (*list).getX() << endl;
cout << (*list).getY() << endl;
cout << list->toString() << endl;
```

小试牛刀

改写 PointStack 类并编写单元测试

- Task 1: 类的三要素

数组版

```
class PointStack {
public:
    PointStack();
    ~PointStack();
    void push(Point pt);
    Point peek() const;
    Point pop();
    void clear();
    int size() const;
    bool isEmpty() const;
    std::string toString() const;
private:
    static const int INIT_SIZE = 8;
    Point *elems;
    int allocatedSize;
```

链表版

```
class PointStack {
public:
    PointStack();
    ~PointStack();
    void push(Point pt);
    Point peek() const;
    Point pop();
    void clear();
    int size() const;
    bool isEmpty() const;
    std::string toString() const;
private:
    Point *elems;
};
```

- Task 2：实现构造器
- Task 3：实现 push
- Task 4：实现 pop
- Task 5：实现 clear 和析构函数

今日话题

- ~~内存与指针~~
- ~~链表~~

下一次课

- 链表

THE END