

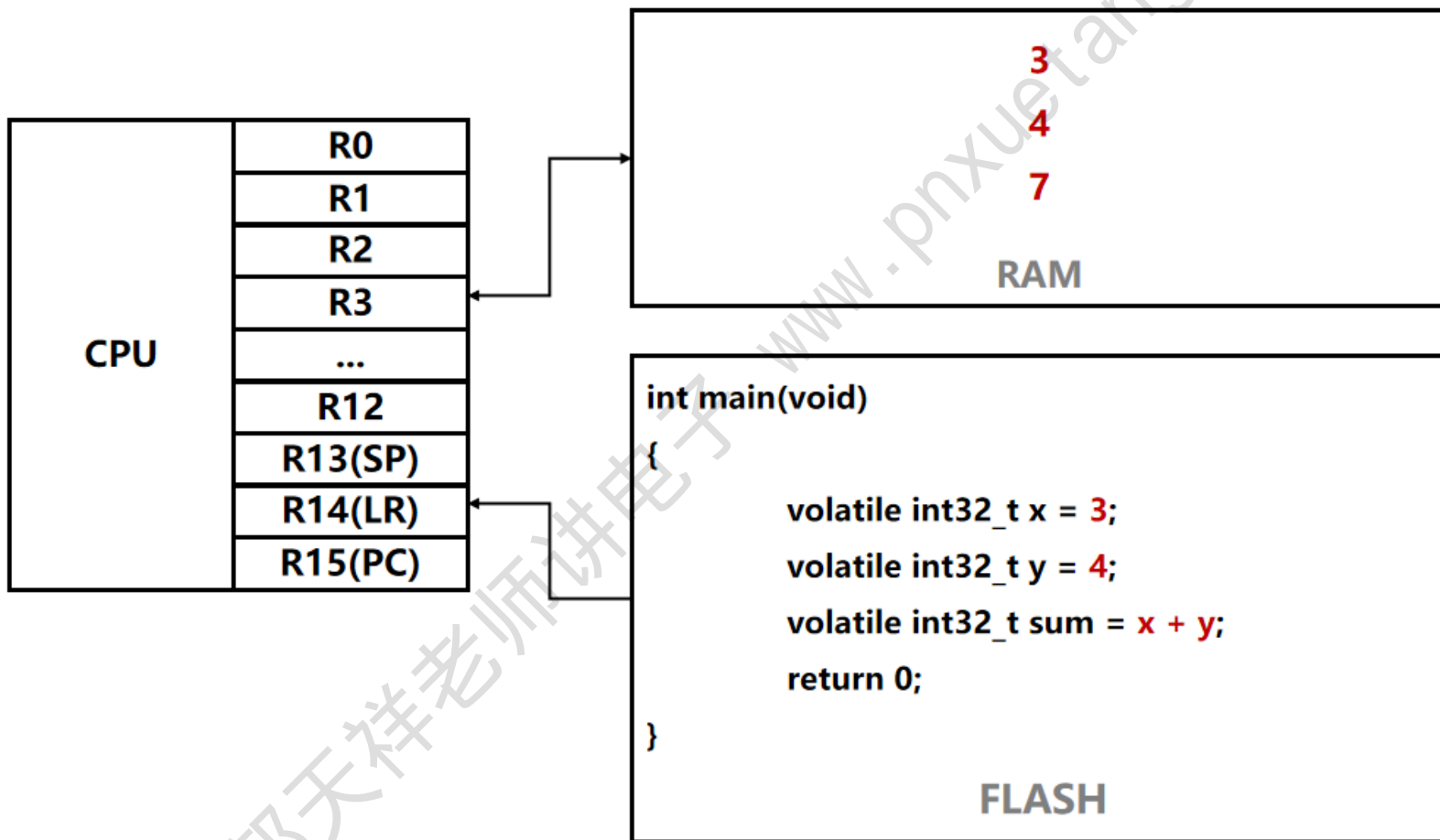
嵌入式C语言之- 指针的工作原理

讲师：叶大鹏

助力你成为优秀的电子工程师！



程序运行时，变量的数据保存在RAM内存中



程序运行时，变量的数据保存在RAM内存中

- 在C语言代码中，通过变量来保存数据，变量的数据被分配在哪个内存地址空间，通常不需要程序员介入，编译器和CPU会按照一定的规则为其分配内存空间。

```
int main(void)
{
    int32_t x = 3;
    int32_t y = 4;
    int32_t sum = x + y;
    return 0;
}
```

```
main
0x000004d0: b50e .. PUSH {r1-r3,lr}
0x000004d2: 2003 . MOVS r0,#3
0x000004d4: 9002 .. STR r0,[sp,#8]
0x000004d6: 2004 . MOVS r0,#4
0x000004d8: 9001 .. STR r0,[sp,#4]
0x000004da: e9dd1001 .... LDRD r1,r0,[sp,#4]
0x000004de: 4408 .D ADD r0,r0,r1
0x000004e0: 9000 .. STR r0,[sp,#0]
```

程序运行时，变量的数据保存在RAM内存中

- 这个例子中，变量x、y和sum保存的是数值，通过它们进行加减乘除运算：

```
int main(void)
{
    int32_t x = 3;
    int32_t y = 4;
    int32_t sum = x + y;
    return 0;
}
```

```
main
0x000004d0: b50e .. PUSH {r1-r3,lr}
0x000004d2: 2003 . MOVS r0,#3
0x000004d4: 9002 .. STR r0,[sp,#8]
0x000004d6: 2004 . MOVS r0,#4
0x000004d8: 9001 .. STR r0,[sp,#4]
0x000004da: e9dd1001 .... LDRD r1,r0,[sp,#4]
0x000004de: 4408 .D ADD r0,r0,r1
0x000004e0: 9000 .. STR r0,[sp,#0]
```

- 但是在某些场景，需要直接操作的不是数值，而是地址，这也是C语言为什么具有软硬结合特点的原因。

通过地址访问寄存器

- 通过地址来访问寄存器，这是嵌入式开发中的一个典型应用。

```
#define GPIOG_CTL0  (* (uint32_t *)0x40012000)
```

```
...
```





```
GPIOG_CTL0 = 0xFFFFFFFF;
```

这个程序能正确执行吗？

➤ 设计实现两个变量互换数值的函数

```
void Swap(int32_t x, int32_t y)
{
    int32_t temp = x;
    x = y;
    y = temp;
}




int main(void)
{
    int32_t a = 8;
    int32_t b = 9;
    Swap(a, b);
    return 0;
}
```

  &x	0x200003F4
  &y	0x200003F8

a 8
0x20000404

等价于赋值运算 $x = a$

x 8
0x200003F4

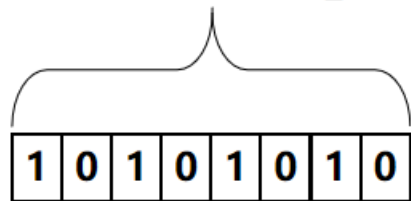
  &a	0x20000404
  &b	0x20000400

指针变量用来保存地址

➤ 首先记住以下概念：

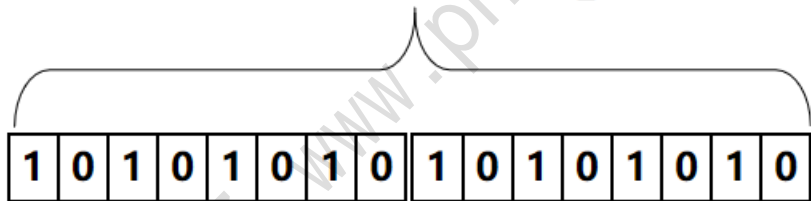
1.单片机的存储空间最小单位是字节，每个字节都有对应的地址：

一个字节保存uint8_t类型



0x20000000

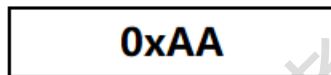
两个字节保存uint16_t类型



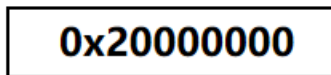
0x20000004

0x20000005

2.存储空间存储的数据既可以是普通数值，也可以是地址：



0x20000000



0x20000004~7

3.保存数值的是普通变量，那么保存地址的是什么变量？是指针变量。

指针变量的定义

- 语法格式:

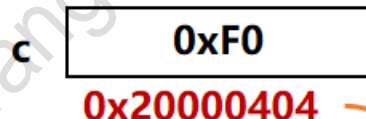
数据类型 *变量名

1. **变量名**，普通变量保存的是数值，指针变量保存的是地址；
2. *****，**指针运算符**，有2种作用，在这的作用是表示定义的是一个指针类型的变量；
3. **数据类型**，指针变量要保存地址，数据类型要和这个地址中保存数据类型保持一致。

指针是如何使用的？

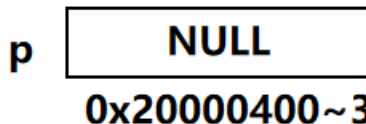
```
uint8_t c = 0xF0;
```

普通变量c会被分配内存空间，保存数值0xF0



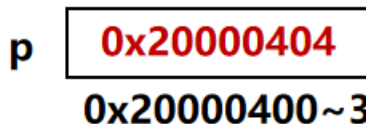
```
uint8_t *p;
```

指针变量p也会被分配内存空间，用来保存地址



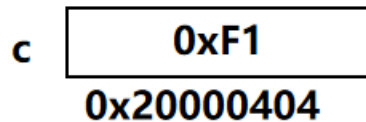
```
p = &c;
```

此时p就保存了0x20000404这个地址，uint8_t *中的uint8_t要和0x20000404保存的数据类型保持一致



```
(*p)++;
```

p, 这里, 是第2种作用，官方称为“解引用”，意思是可以访问指针变量保存的地址了(*)0x20000404，此时*p == c，对*p进行加减乘除运算，也就是对c进行加减乘除，所以也称为“间接访问”



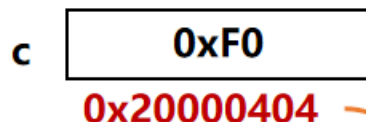
同理，定义其他类型的指针变量：int32_t *p; struct Airquality *p;

指针是如何使用的？

- 指针变量保存的是地址，它占用内存空间大小，取决于硬件平台地址的位数，对于ARM32，是4个字节，与指针变量定义时的数据类型无关：

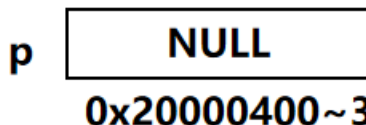
```
uint8_t c= 0xF0;
```

普通变量c会被分配内存空间，保存数值0xF0



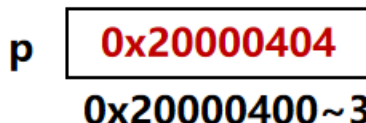
```
uint8_t *p;
```

指针变量p也会被分配内存空间，用来保存地址



```
p = &c;
```

此时p就保存了0x20000404这个地址



指针是如何使用的？

- 指针变量保存的是地址，它占用内存空间大小，取决于硬件平台地址的位数，对于ARM32，是4个字节，与指针变量定义时的数据类型无关：

```
uint32_t c = 0xF0F0F0F0;
```

普通变量c会被分配内存空间，保存数值0xF0

c

0xF0F0F0F0

0x20000404~7

```
uint32_t *p;
```

指针变量p也会被分配内存空间，用来保存地址

p

NULL

0x20000400~3

```
p = &c;
```

此时p就保存了0x20000404这个地址，也就是c的首地址

p

0x20000404

0x20000400~3

指针是如何使用的？

- 定义时注意事项：

`uint32_t *p1, p2;` //此时p1是指针变量，p2是普通变量

`uint32_t *p1, *p2;` //建议书写时*紧挨着变量名字

指针变量作为函数参数

➤ 设计实现两个变量互换数值的函数

```
void Swap(int8_t *x, int8_t *y)
```

```
{
```

```
    int8_t temp = *x;
```

```
    *x = *y;
```

```
    *y = temp;
```

```
}
```

```
int32_t main(void)
```

```
{
```

```
    int8_t a = 8, b = 9;
```

```
    Swap(&a, &b);
```

```
    printf("after swap, a = %d, b = %d \n", a, b);
```

```
    return 0;
```

```
}
```

a 8

0x20000404

参数传递等价于 `int8_t *x = &a`

x 0x20000404

0x200003F4

此时，`*x`表示访问(*)0x20000404，相当于访问a，改变*x，也就是改变a的数据。

通过指针来访问寄存器

- 寄存器也是通过地址来访问，和访问内存一样：

```
#define GPIOG_CTL0 (* (uint32_t *)0x40012000)
```

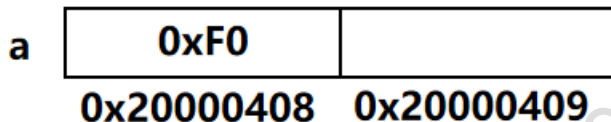
```
...
```

```
GPIOG_CTL0 = 0xFFFFFFFF;
```

1. 0x40012000是一个十六进制数值，此时编译器并不认为它是一个地址；
2. 通过强制类型转换，让编译器认为它是一个地址：(uint32_t *)0x40012000；
uint32_t * 在这里和定义指针变量时作用一样，此时可以将0x40012000理解为定义指针变量时，uint32_t *p中的p；
3. * (uint32_t *)0x40012000，第一个*和*p = 1一样，表示要访问这个地址，接下来就可以对它赋值0xFFFFFFFF。

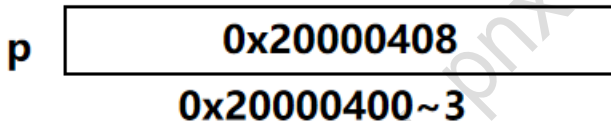
指针变量的运算

```
uint8_t a = 0xF0;
```



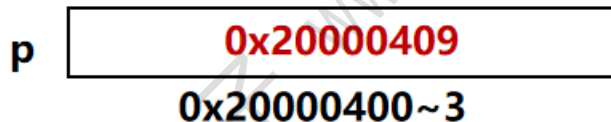
```
uint8_t *p;
```

```
p = &a;
```



```
p++;
```

```
p = ?
```



- 指针变量+1，并不是地址值+1，而是按照指针定义时的数据类型大小为步长进行增加，这里的类型是uint8_t，大小为1个字节。

指针变量的运算

`int32_t a = 0x12345678;`

a	0x78	0x56	0x34	0x12
	0x20000408	409	40A	40B

`int32_t *p;`

`p = &a;`

p	0x20000408
	0x20000400~3

`p++;`

`p = ?`

p	0x2000040C
	0x20000400~3

- 指针变量+1，并不是地址值+1，而是按照指针定义时的数据类型大小为步长进行增加，这里的类型是`int32_t`，大小为4个字节。

指针变量的运算

`int32_t a = 0x12345678;`

a	0x78	0x56	0x34	0x12
	0x20000408	409	40A	40B

➤ 如何使用指针访问a的每一个字节空间呢?

`int8_t *p;`

- 首先定义`int8_t`类型的指针变量

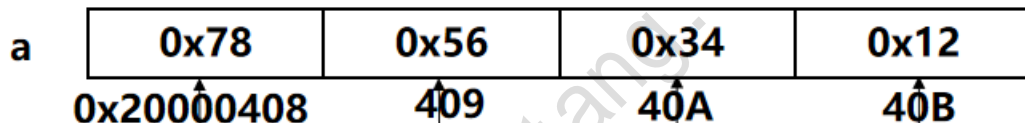
`p = (int8_t *) &a;`

- 赋初值a的地址，由于类型不匹配，需要强制类型转换，`(int8_t *) &a`表示0x20000408里面存放的是`int8_t`类型的数据，这样就可以使用p按照一个字节为步长来访问a的存储空间。

指针变量的运算

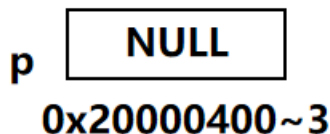
```
int32_t a = 0x12345678;
```

```
&a = 0x20000408
```



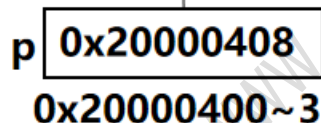
```
int8_t *p
```

```
&p = 0x20000400
```



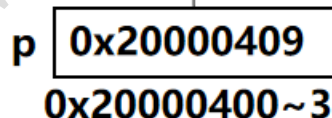
```
p = (int8_t *)&a
```

```
*p = 0x78
```



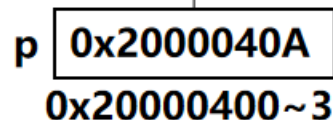
```
p++;
```

```
*p = 0x56
```



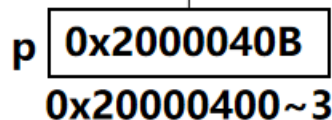
```
p++;
```

```
*p = 0x34
```



```
p++;
```

```
*p = 0x12
```



应用案例1

- 假如在蓝牙项目开发中，2个设备之间进行日期数据传输时，我们先定一个协议，用4个byte也就是uint32_t来表示一个日期，其中byte3表示年份的高位数，byte2表示年份的低位数，byte1表示月份，byte0表示日期。

设备端现在收到另外一台设备传过来的日期数据00010100 00010011 00000110 00011101。那么我要如何解析这个数据来得到实际日期呢？

应用案例

```
/*  
*第一步，获取日期。  
*日期是最后一个byte，也就是最后8位  
*/  
  
uint32_t date = 0x1413061D;  //00010100 00010011 00000110 00011101;  
  
uint8_t day = date;  //(计算结果是00011101，十进制表示是29，也就是日期是29)。  
  
/*  
*第二步，获取月份。  
*月份是倒数第2个byte，此时需要先将最后一个byte砍掉(也就是右移8位)  
*/  
  
date = date >> 8;  //(计算结果是00010100 00010011 00000110)  
  
uint8_t month = date;  //(计算结果是00000110，十进制表示是6，也就是月份是6月)。
```

应用案例

```
/*  
*第三步，获取年份低位。  
*先将最后一个byte砍掉(也就是右移8位)  
*/  
date = date >> 8; //(计算结果是00010100 00010011)  
uint8_t year_low = date; //(计算结果是00010011，十进制表示是19)。  
  
/*  
*第四步，获取年份高位。  
*先将最后一个byte砍掉(也就是右移8位)  
*/  
date = date >> 8; //(计算结果是00010100)  
uint8_t year_high = date; //(计算结果是00010011，十进制表示是20)。
```

➤ 思考：学习完指针课程后，考虑还有没有更优方案？

应用案例

```
uint32_t date = 0x1413061D;
```

```
uint8_t *p;
```

```
p = (uint8_t *)&date;
```

```
uint8_t day = *p;
```

```
p++;
```

```
uint8_t month = *p;
```

```
p++;
```

```
uint8_t year_low = *p;
```

```
p++;
```

```
uint8_t year_high = *p;
```

THANK YOU!