作业5

1. 名词解释
2. 寄存器寻址

寄存器寻址是一种指令寻址方式，它用于处理存储在CPU内部寄存器中的操作数。在这种寻址模式下，指令直接从寄存器中读取操作数，而不是从内存中读取。

特点：

速度：由于寄存器位于CPU内部，访问速度非常快，通常比访问内存要快得多。

简单性：寄存器寻址简化了指令格式，因为指令只需要指定寄存器编号，而不需要内存地址。

限制：寄存器的数量有限，这限制了同时处理的数据量。

寄存器寻址的类型：

立即寻址：虽然不是寄存器寻址，但与寄存器寻址类似，它将一个常量（立即数）作为操作数。

直接寻址：与寄存器寻址不同，直接寻址涉及内存地址，指令会直接引用内存中的数据。

间接寻址：在间接寻址中，指令包含一个指向内存地址的寄存器，CPU首先读取该寄存器中的地址，然后再访问该地址中的数据。

应用场景：

寄存器寻址在以下场景中非常有用：

算术和逻辑运算：快速执行加法、减法、乘法、除法以及逻辑运算。

数据传输：在寄存器之间快速移动数据。

条件分支：基于寄存器中的值进行条件判断和跳转。

1. 寄存器间接寻址

寄存器间接寻址是一种寻址方式，它使用寄存器中的值作为内存地址来访问数据。在这种寻址模式下，指令本身并不直接包含内存地址，而是包含一个寄存器的编号，CPU从该寄存器中读取内存地址，然后通过这个地址访问内存中的数据。

特点：

间接性：CPU不直接访问数据，而是间接地通过寄存器中的地址来访问。

灵活性：提供了更大的灵活性，因为内存地址可以在运行时动态确定并存储在寄存器中。

抽象层：为内存访问提供了一个抽象层，允许程序间接地引用数据。

寄存器间接寻址的类型：

完全寄存器间接寻址：仅使用一个寄存器，其内容直接作为内存地址。

基址加偏移寻址：使用基址寄存器和一个偏移量来形成内存地址，即内存地址 = 基址寄存器 + 偏移量。

索引寻址：与基址加偏移寻址类似，但通常用于数组访问，其中索引寄存器的内容作为偏移量。

应用场景：

寄存器间接寻址在以下场景中非常有用：

堆栈操作：在堆栈操作中，堆栈指针寄存器通常用于间接寻址，以实现数据的入栈和出栈。

参数传递：在子程序调用中，参数可以通过寄存器间接寻址传递给子程序。

数组处理：在处理数组时，可以通过寄存器间接寻址快速访问数组元素。

1. 综合题
2. 设系统时钟频率为12M，用定时器T0控制在P1.0口输出周期为500微秒，占空比为50%的方波。写出相应的代码

#include <reg51.h>

#define SYS\_CLK 12000000

#define TIMER\_CLK (SYS\_CLK / 12)

#define MICROSECONDS 500

void Timer0\_Init() {

TMOD &= 0x0F;

TMOD |= 0x01;

TH0 = 0xFF;

TL0 = 0x00;

TR0 = 1;

ET0 = 1;

EA = 1;

}

void Timer0\_ISR() \_\_interrupt 1 {

TH0 = 0xFF;

TL0 = 0x00;

P1 ^= 0x01;

}

void main() {

Timer0\_Init();

while(1) {

}

}

1. 利用UART的方式1，将甲机片内ARM 0x30~0x3f的内容传到乙机片内ARM 0x40~0x4f，写出相应的甲机代码和已机代码。波特率选1200bps，主机频率12M。

甲机：

#include <reg51.h>

#define UART\_BUFF\_SIZE 16

#define START\_ADDR 0x30

#define DATA\_SIZE 16

void UART\_Init() {

SCON = 0x50;

TMOD &= 0x0F;

TMOD |= 0x20;

TH1 = TL1 = 0xFD;

TR1 = 1;

TI = 0;

EA = 1;

ES = 1;

}

void UART\_ISR() interrupt 4 using 1 {

if (TI) {

TI = 0;

SBUF = \*(volatile unsigned char\*)(START\_ADDR + i);

i++;

if (i >= DATA\_SIZE) {

ES = 0;

while (1) {

}

}

}

}

void main() {

int i = 0;

UART\_Init();

while (1) {

if (!TI) {

ES = 1;

}

}

}

乙机

#include <reg51.h>

#define UART\_BUFF\_SIZE 16

#define DEST\_ADDR 0x40

void UART\_Init() {

SCON = 0x50;

TMOD &= 0x0F;

TMOD |= 0x20;

TH1 = TL1 = 0xFD;

TR1 = 1;

EA = 1;

ES = 1;

}

// UART接收中断服务程序

void UART\_ISR() interrupt 4 using 1 {

if (RI) {

RI = 0;

unsigned char data = SBUF;

\*(volatile unsigned char\*)(DEST\_ADDR) = data;

DEST\_ADDR++;

}

}

void main() {

UART\_Init();

while (1) {

}

}

1. uC/OS-II支持的5种任务状态及含义。

就绪：

任务处于就绪状态时，它已经准备好执行，但可能因为其他任务正在运行或有更高优先级的任务而暂时没有被CPU执行。任务在就绪状态时等待CPU资源。

运行：

任务处于运行状态时，它正在被CPU执行。在单核系统中，一次只能有一个任务处于运行状态。

挂起：

挂起状态的任务因为某些原因而暂时不能运行。即使任务的优先级最高，它也不会被调度执行，直到被明确地恢复。

等待：

等待状态的任务正在等待某个事件发生（例如，等待信号量、互斥信号量、邮箱、消息队列等）。任务在等待状态时，它期望获取某种资源或事件的发生，直到该条件满足，它才会变为就绪状态。

空闲：

空闲状态的任务是系统内置的特殊任务，通常具有最低的优先级。当没有其他任务需要执行时，CPU将执行空闲任务。空闲任务通常是执行一些低优先级任务或系统维护工作的地方。