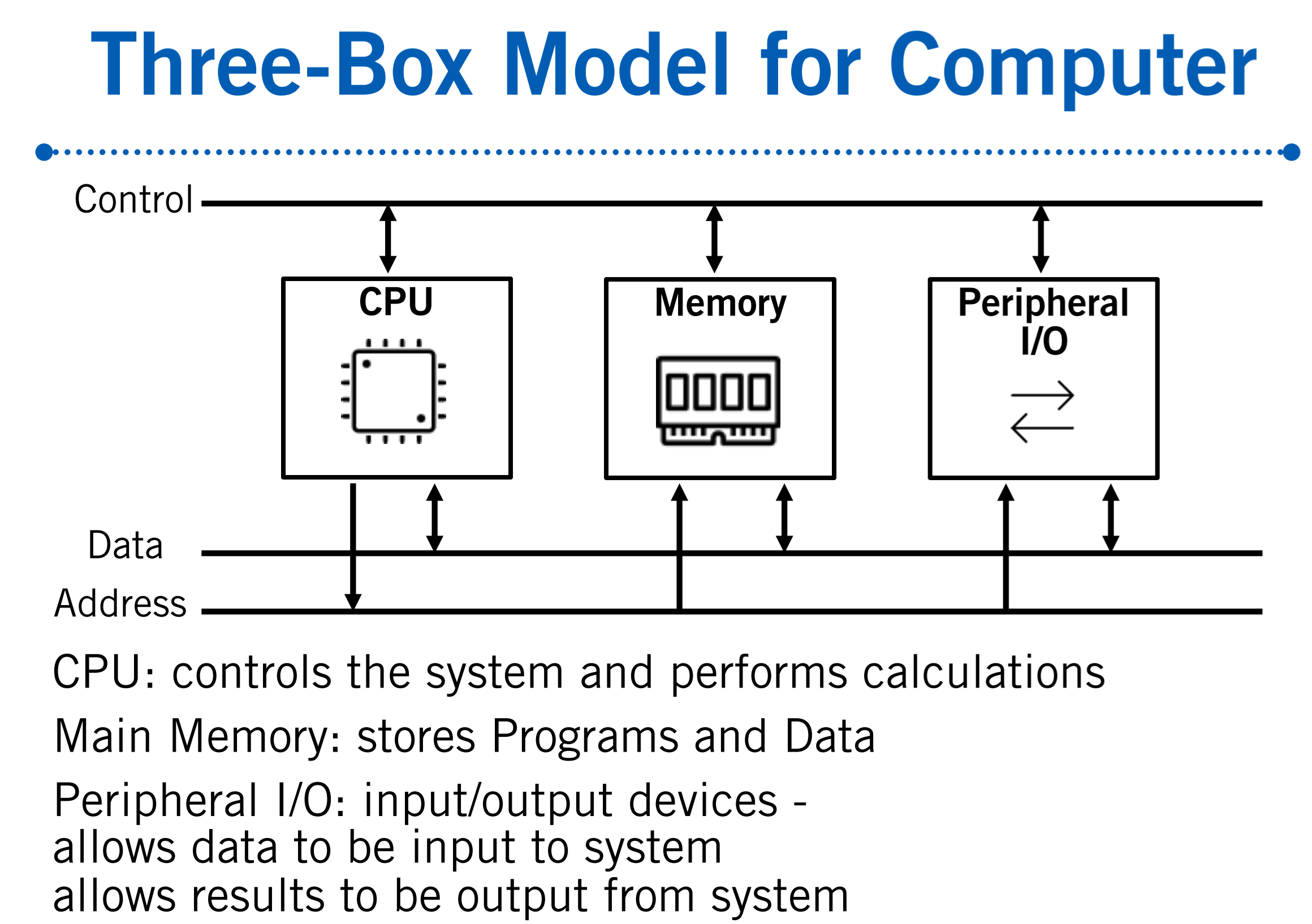
# 微处理器和嵌入式系统

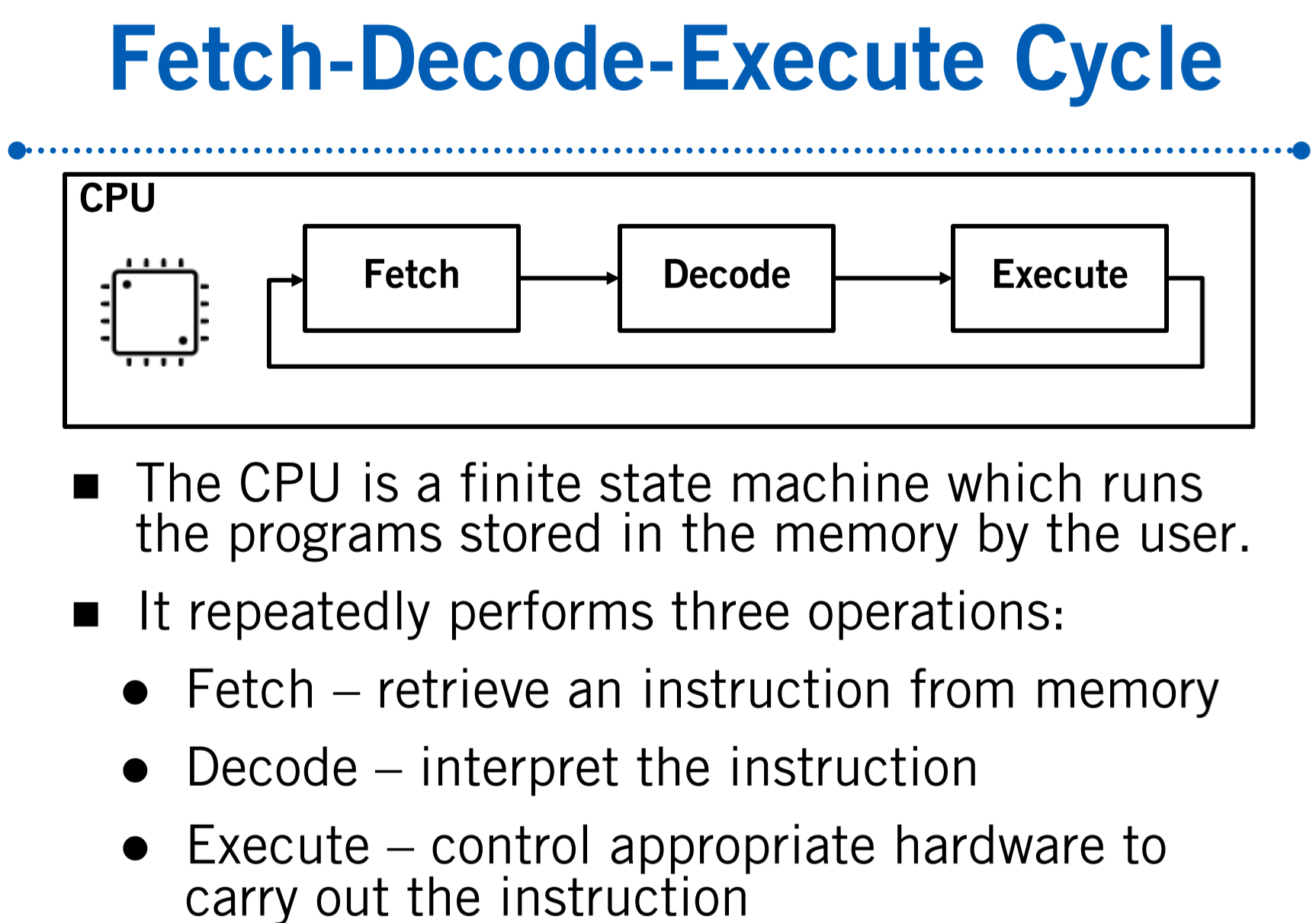
#微处理器结构及简介

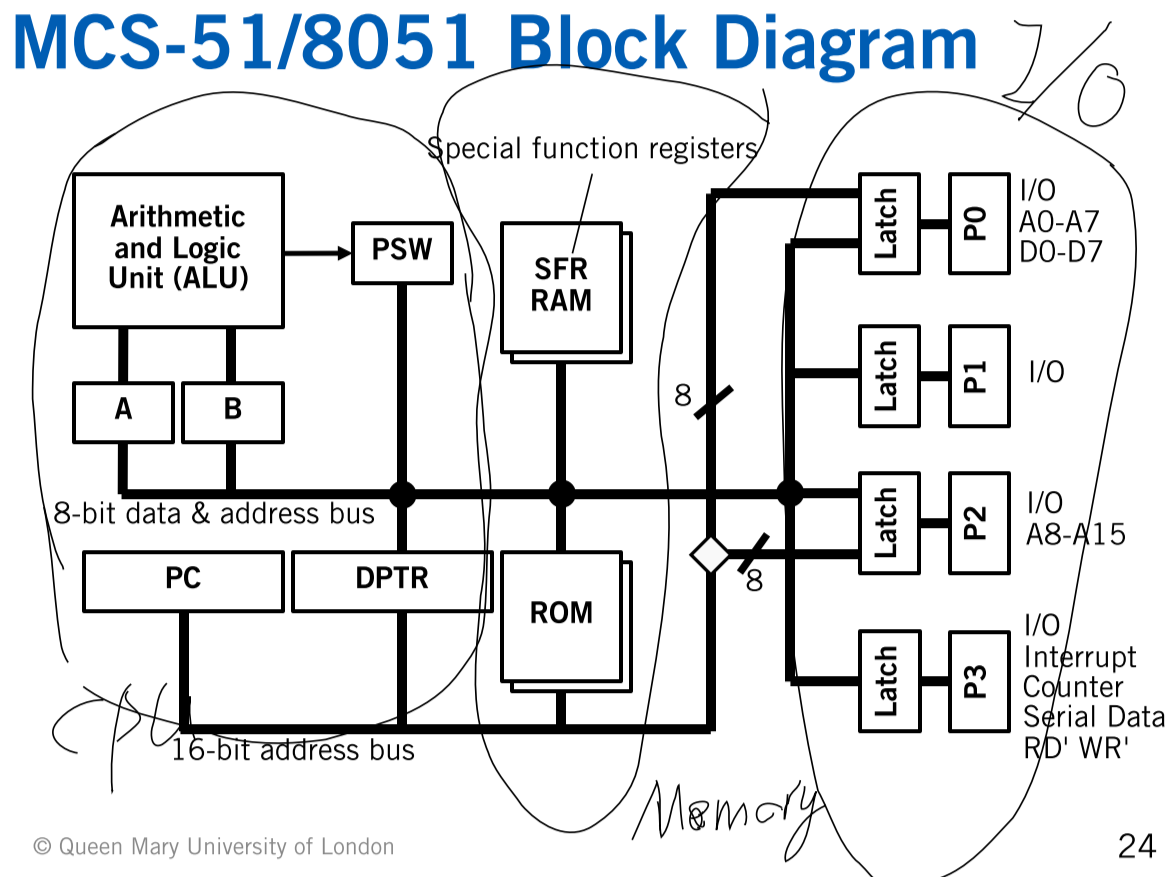
重点！！记忆！！



Ram：用来编程和存储数据

Rom：用来固定程序和常量数据

retrieve检索 interpret翻译



记忆各个原件所在于3盒模型的哪个位置！！

#8051的特性

**CPU为8bit**，即可以同时进行8bit的运算

**地址空间为64KB**，因为有16bit的地址总线，即2^16

**ROM大小为4KB**, for code

**RAM大小为128B**, for data:

32B 分成了4个寄存器Banks，每个8B

16B bit-address memory locations

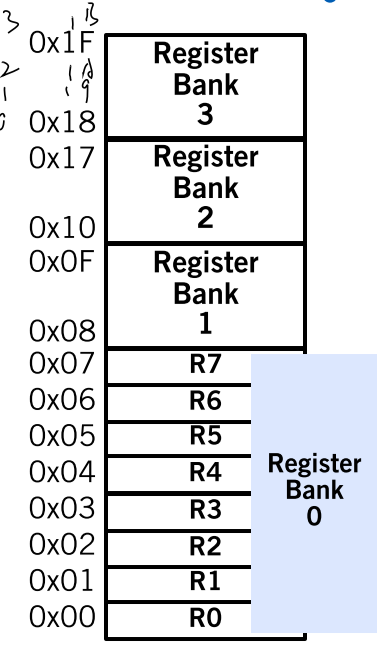
80B 通用数据内存

**硬件计时器Timer为16-bit**，有两个

**IO引脚有32个**，在4个8bit口上（P0-P3）

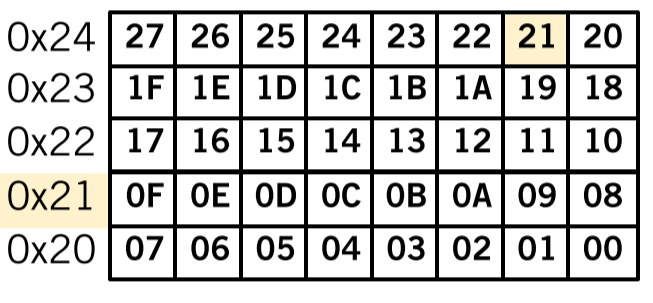
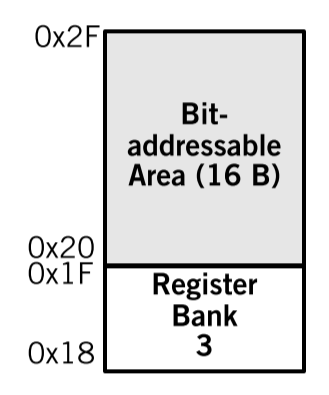
**串行端口1个**

**Tips:** 8051的**RAM地址总线为8bit**，所以最多能有256B的RAM，不过一般8051只有128B。

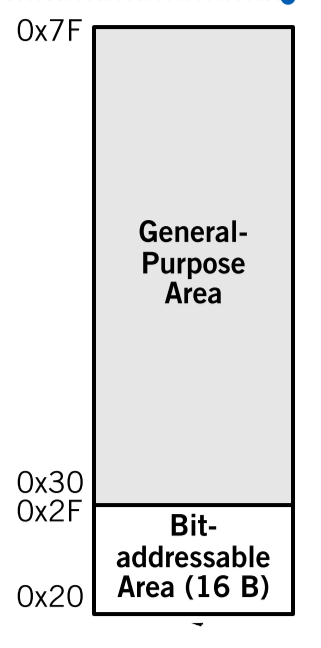
如图所示，这块区域为RAM中的寄存器工作区，共占32B，其中有系统定义好的各个寄存器变量，他们都可直接通过寻址调用。一般用于存储变量。

从00到1F

如图所示，这是RAM中的16B位单元和位地址对应表（右表不全，应该到0x2F）。一般用于临时寻址，进行字节寻址。

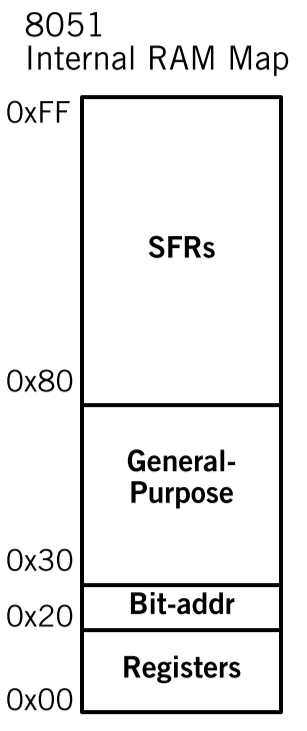


从20到2F

有80B的的一般用途区，也称用户RAM区，用于1）特殊功能寄存，2）更一般用途的RAM区域。

从30到7F

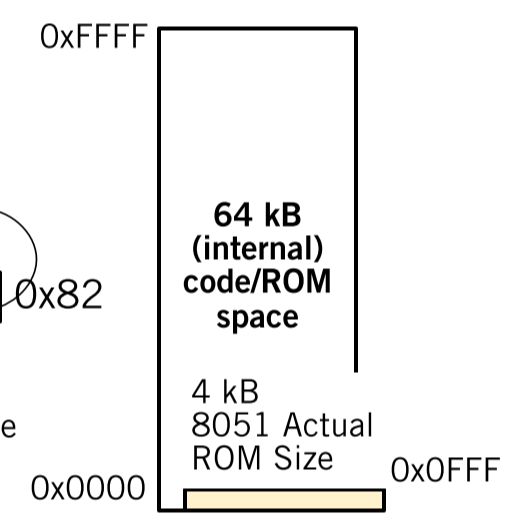
SFR特殊功能寄存器（它不算RAM！！！！只是可以用RAM地址总线寻址！！！！）

一些专门用于控制、管理算数逻辑部件、并行IO口锁存器等功能模块的工作都在SFR中实现。

如

PSW（Program Status Word）

ACC（就是A啦）



ROM

ROM地址总线为16bit，所以总共可以寻64KB（2^16），但是8051只包含其中的1/16，也就是4KB

DPTR

占有两个SFR组成16bit，是一个16bit专用寄存器，其高位字节寄存器用DPH表示，相对的DPL表示低位字节寄存器。它主要用来储存16位ROM地址。

PC

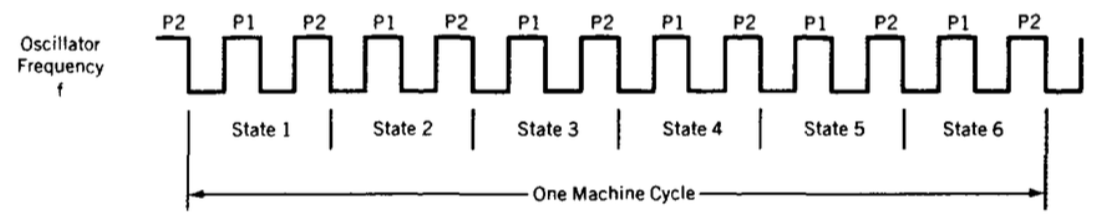
Program counter 也是一个16bitSFR，它是用来存储当前指令的位置。

#振荡器和时钟

\*振荡器和机械频率

1个machine cycle由6个状态组成，1个状态包括2个晶振循环

1 MC=6 States = 12 CC





#指令系统及汇编语言

【标号】 操作码 【目的操作数】【，源操作数】【注释】

LOOP: MOV A , R0 ; A<-(R0)

指令有字节长度的不同，

单字节指令只有1字节，操作码和操作数都包含在1字节中；

双字节指令有2字节，前1字节为操作码，后1字节为操作数；

三字节指令包含3字节，前1字节为操作码，后2字节为操作数

常用指令符号

* + Rn：表示当前选定的工作寄存器组中R0~7寄存器。当前的工作寄存组由PSW中的RS1、RS0决定。
  + #data：表示包含在指令的10位立即数
  + #dataH：表示包含在指令的16位立即数
  + Addr：表示目的地址
  + Direct：表示8位可直接寻址（00~7F）
  + Bit：位寻址，（20~2F）
  + C：表示进位标志
  + @：间接寻址前缀符号，即找到里面的数据当成地址再次寻找其地址中的数据。@A+DPTR表示基址+变址间接寻址

寻址方式：

**立即寻址Immediate addressing：**MOV A, #34H

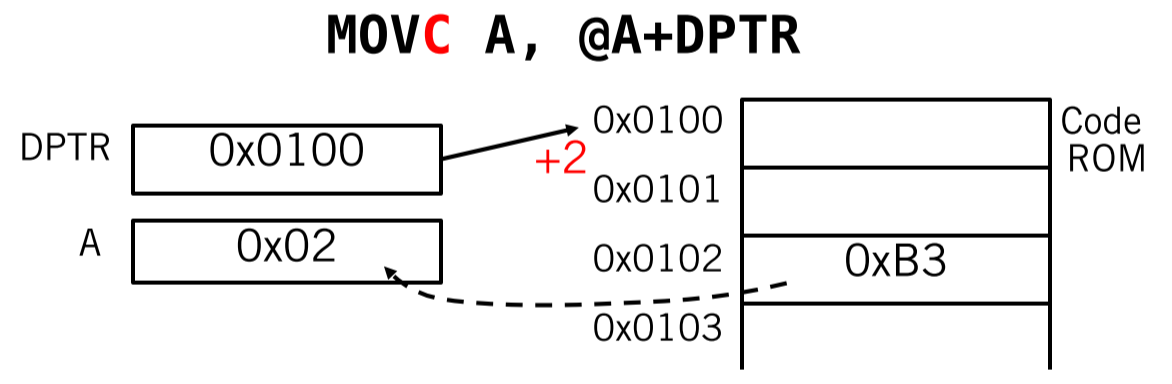
**直接寻址Direct addressing：**MOV A, 30H

**寄存器寻址Register addressing：**MOV A, R0

**寄存器间接寻址Register indirect addressing：**MOV A, @R0

**基址加变址间接寻址Indexed addressing：**MOVC A, @A+DPTR

注意！！！！工作寄存器中只有R0和R1可以作为寄存器间接寻址，特殊寄存器中只有DPTR、PC和SP能够作寄存器间接寻址。



#算数类操作指令

算术类的操作指令24条，可分为加法，带进位的加法，带借位的减法，加一，减一，乘除法，十进制数调整指令。

三种标志：C，AC，OV。

C：是两个无符号操作数相加标明加法指令结果是否有进位的一个标志，若位7有进位，则C=1。

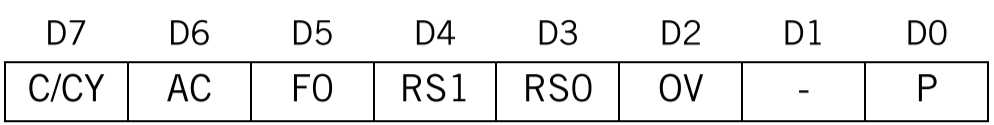
AC：辅助进位运算符，当位3有进位，AC=1。

OV：看用户定义的数是否有符号数，若是两个符号数相加，OV=1表示运算溢出，结果有误。而计算机不会区分是否有符号操作，它只会当成有符号，只要位6**和**位7**其中一个**有进位，OV=1；否则，如果位6和位7**同时有进位或者同时没有进位**，OV=0。

**注意：二进制补码的操作是取反加1**

PSW（Program Status Word）

程序状态字，组成如下：



RS1和RS0用来选择寄存器工作块，00表示bank0，01表示bank1，10表示bank2，11表示bank3。

乘除法：

* 乘法：一般是MUL AB，将结果的低8位存储在A中，高8位存储在B中：A\*B = B:A如果超过255，OV=1
* 除法：一般是DIV AB，A除以B将结果所得商存储在A中，B中存储余数。执行除法指令后，OV和C都会被清0，但如果除数为0，则溢出标志OV=1，A,B内容不确定。

#逻辑运算与移位运算

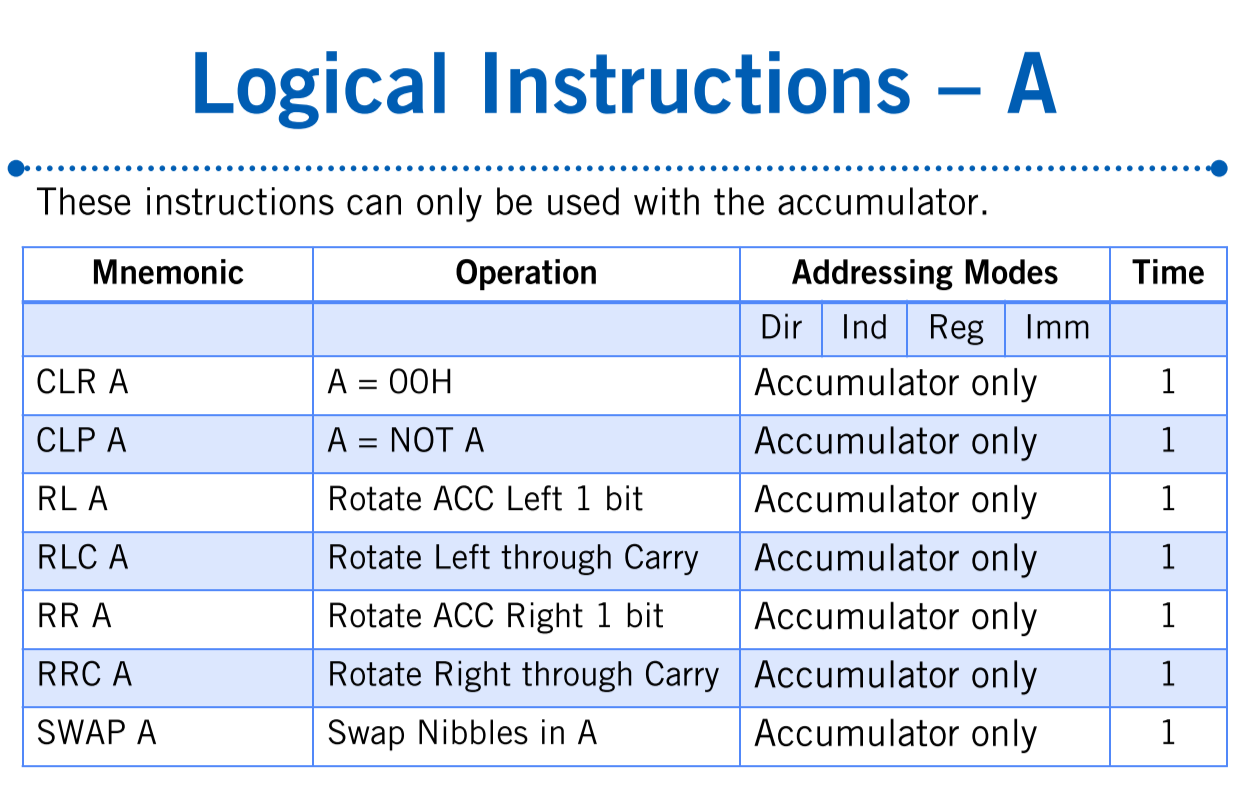
AND-----------ANL;

OR------------ORL;

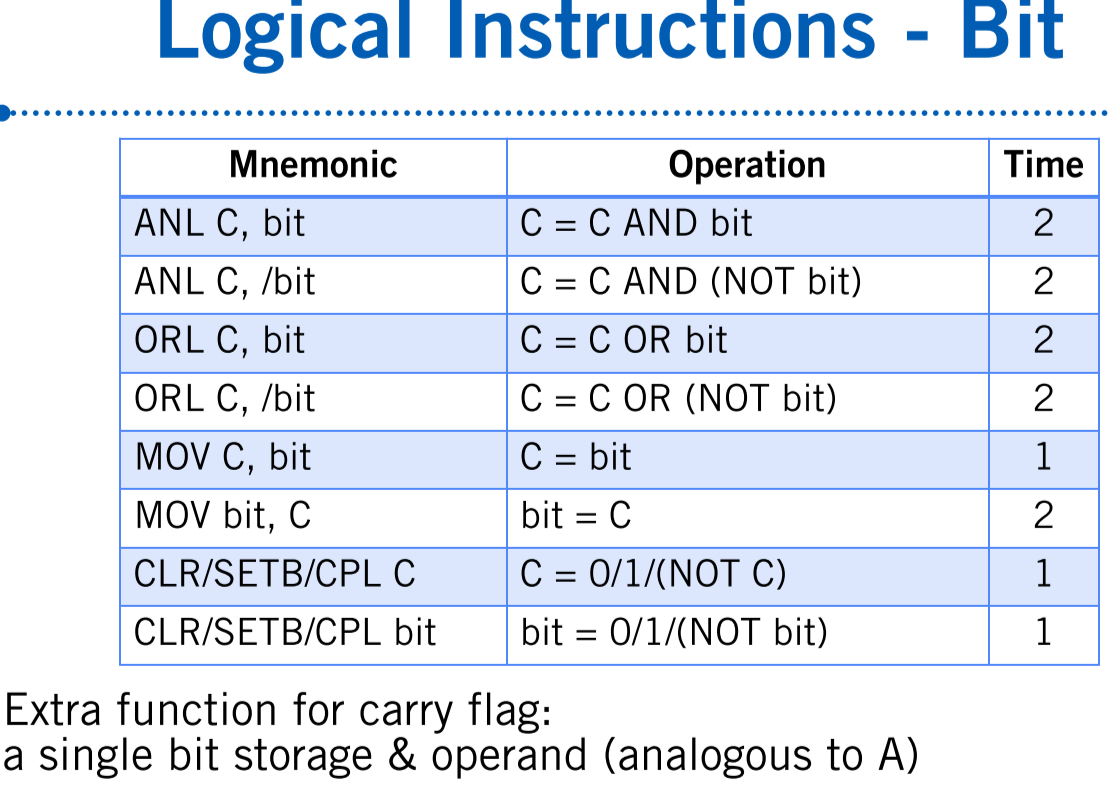
XOR-----------XRL;

NOT-----------CPL.

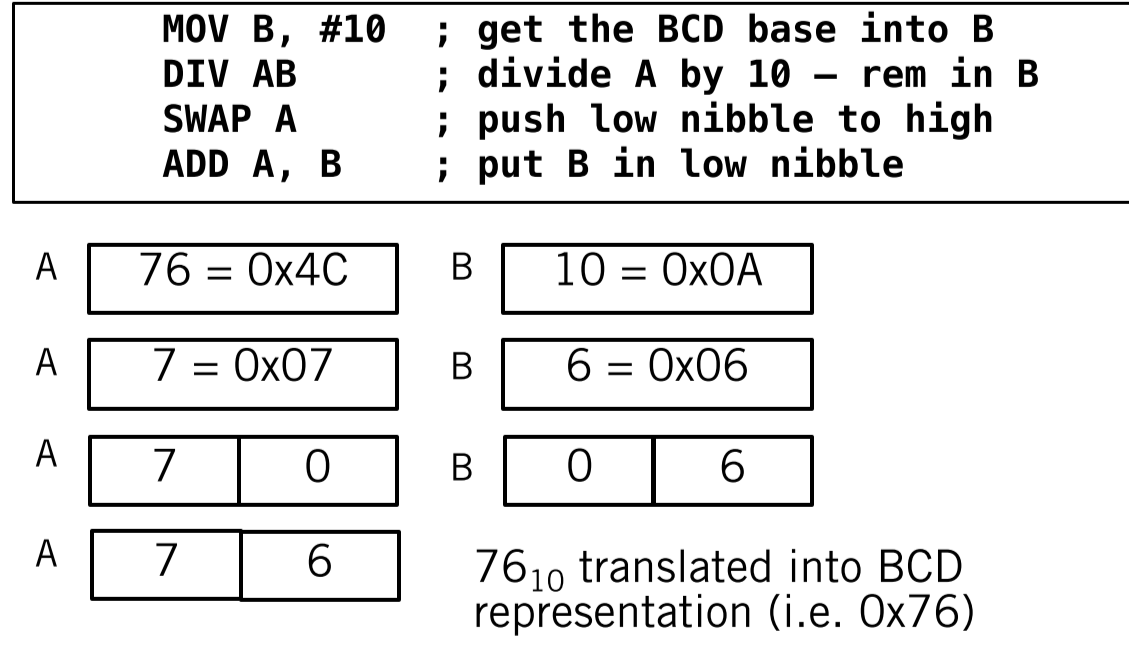
一些A专属的逻辑指令



一些bit指令



一个指令技巧，Binary to BCD



#汇编语言进阶

在8051中jump和call的范围是：

相对转移：-128byte到+127byte

绝对转移：within the same 2KB page

长转移：任意地址空间

\*\*JUMP类

AJUMP addr11;

绝对转移指令，双字节。指令的机器码由指令提供的11位直接地址和5位指令操作码00001组成，并按照，a10,a9,a8,0,0,0,0,1,a7,a6,a5,a4,a3,a2,a1,a0排列。

先由Ajump指令所在位置的地址**PC值加2**（该指令字节数）构成当前的PC值，**然后**把指令中的addr11送入PC10~0，PC15~11不变，形成程序转移的目的地址。

\*LJMP addr16

长转移指令，三字节。指令中提供了16位目标地址，执行时把指令码的第2和第3字节分别装入程序计数器PC的**高位**和**低位**，可无条件的转向addr16指出的目标地址。

\*SJMP rel

是无条件相对转移指令，双字节。指令中的操作数是相对地址偏移量rel，rel是一个带符号的8位二进制的补码数，能够实现向前或向后跳转。如果rel为正，则向后转移，如果rel为负数，则向前转移。执行命令时，先把PC加2，再把rel偏移量加到PC上。

使用了相对转移的语句是：JC; JNC; JB; JNB; CJNE; DJNZ; JZ; JNZ; SJMP。

Bit:

* JC rel: jump carry;
* JNC rel: jump not carry;
* JB bit, rel: jump bit;
* JNB bit, rel: jump not bit;
* JBC bit, rel: jump bit, clear bit;

Byte

JZ rel: jump A = 0;

JNZ rel: jump A !=0;

DJNZ <byte>, rel: decrement and jump if <byte> not zero. **注意！！！先减1，再判断！！**

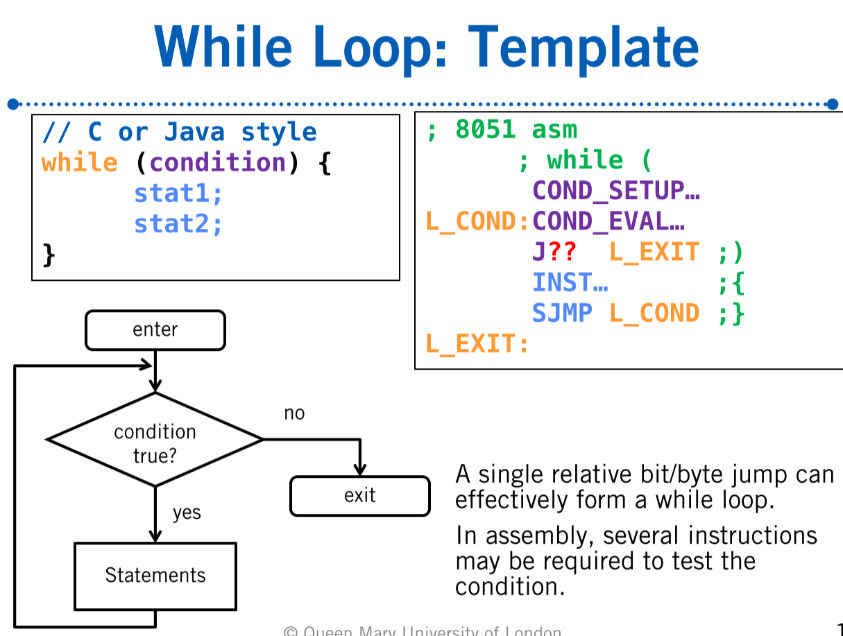
CJNE A, <byte>, rel: jump not equal.

CJNE <byte>, #data, rel

注意！！！CJNE会影响Carry，如果前者小于后者，C=1，其他情况 C=0。（**前小后大C FLAG等于1**）

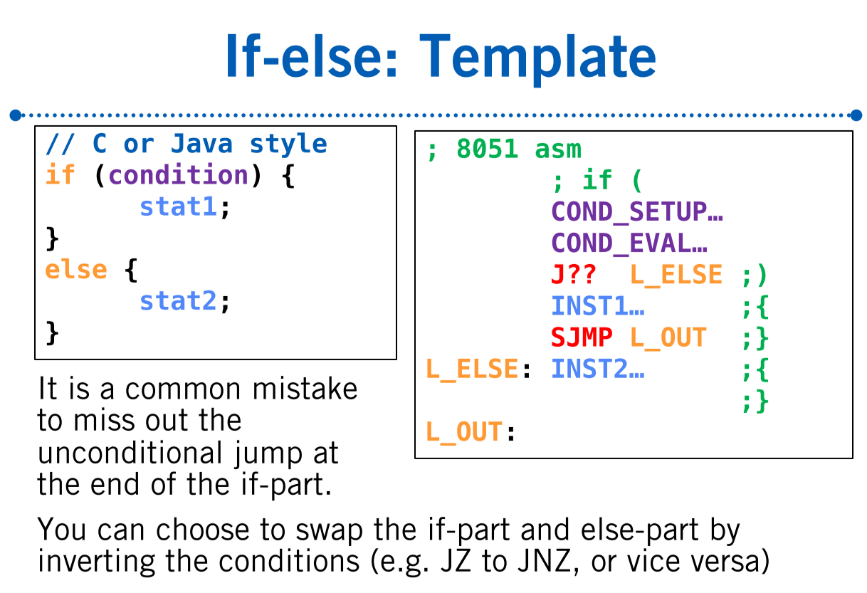
#一些代码的实例

while



要点，最开始写循环头（可能是一些操作并带着判断，判断写后面），然后判断是否可以出循环，再在后面写条回循环头语句

If-else

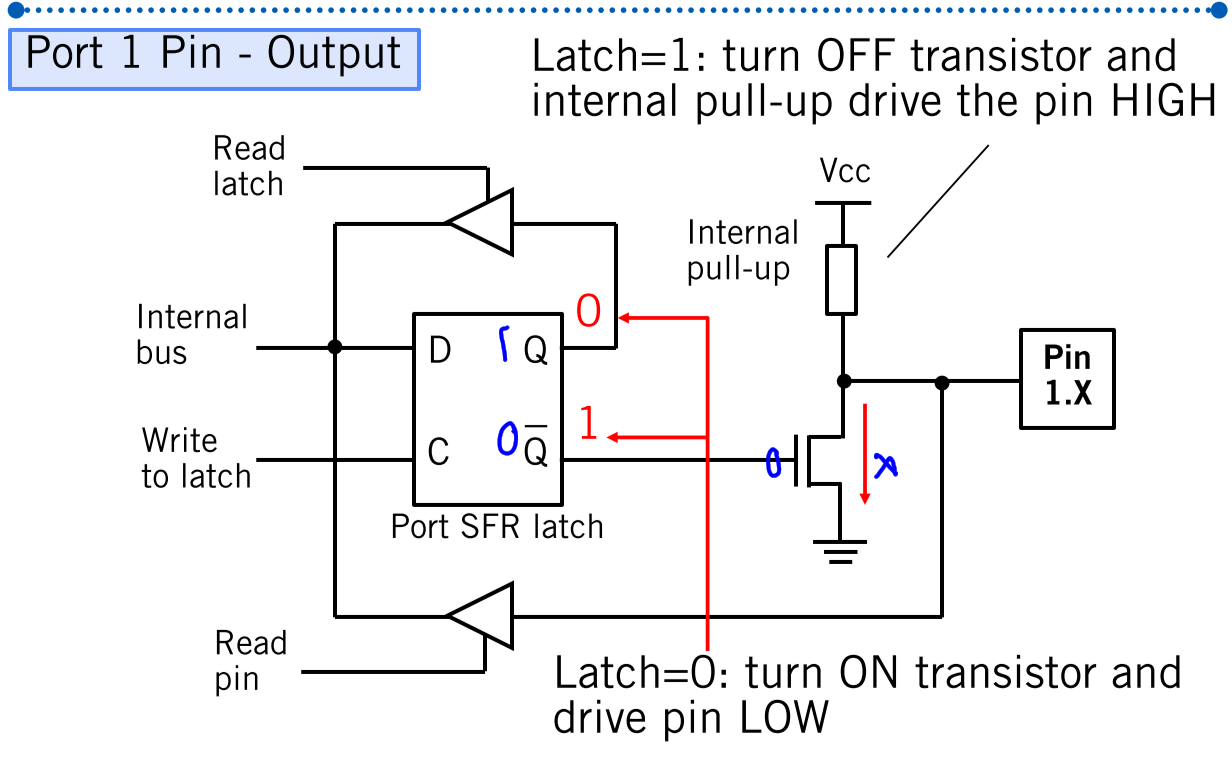


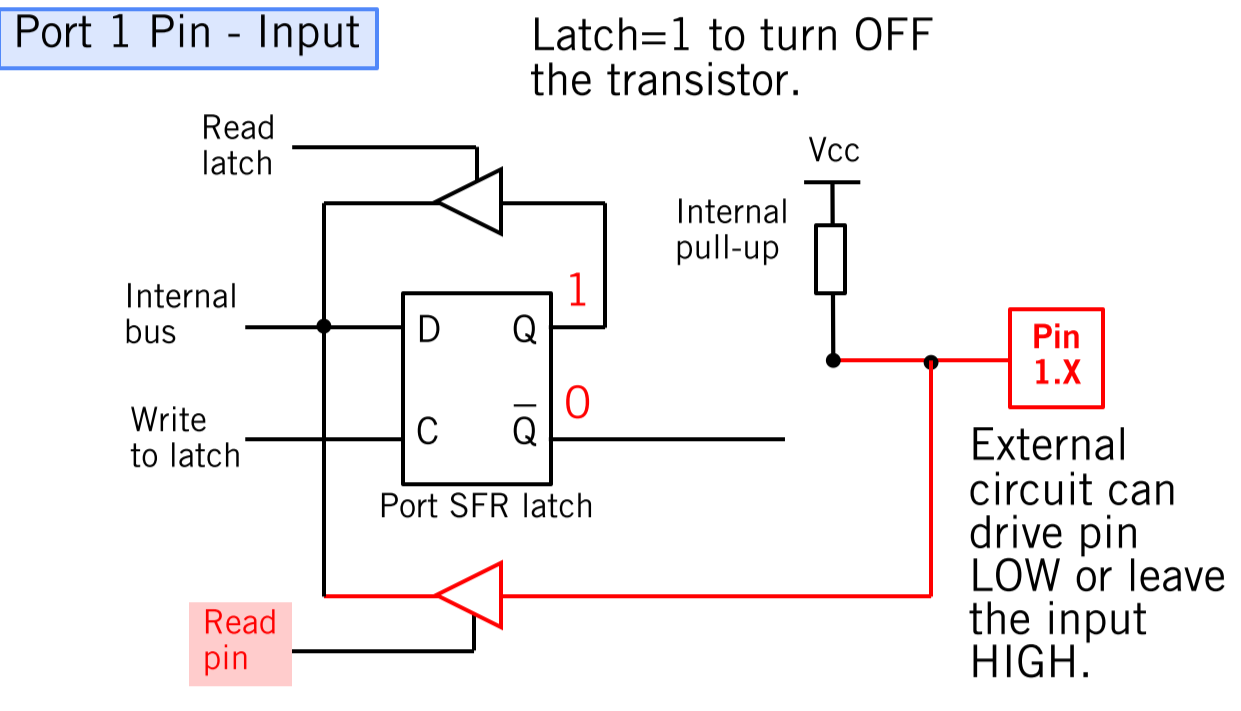
类似与while，但是最后不返回，而是直接写跳入另一个条件。

#IO ports

//我不知道它要怎么考我IOport，先暂时略，明天补一哈

要看的懂图，知道外部输入和SFR latch怎么影响输出的，例子



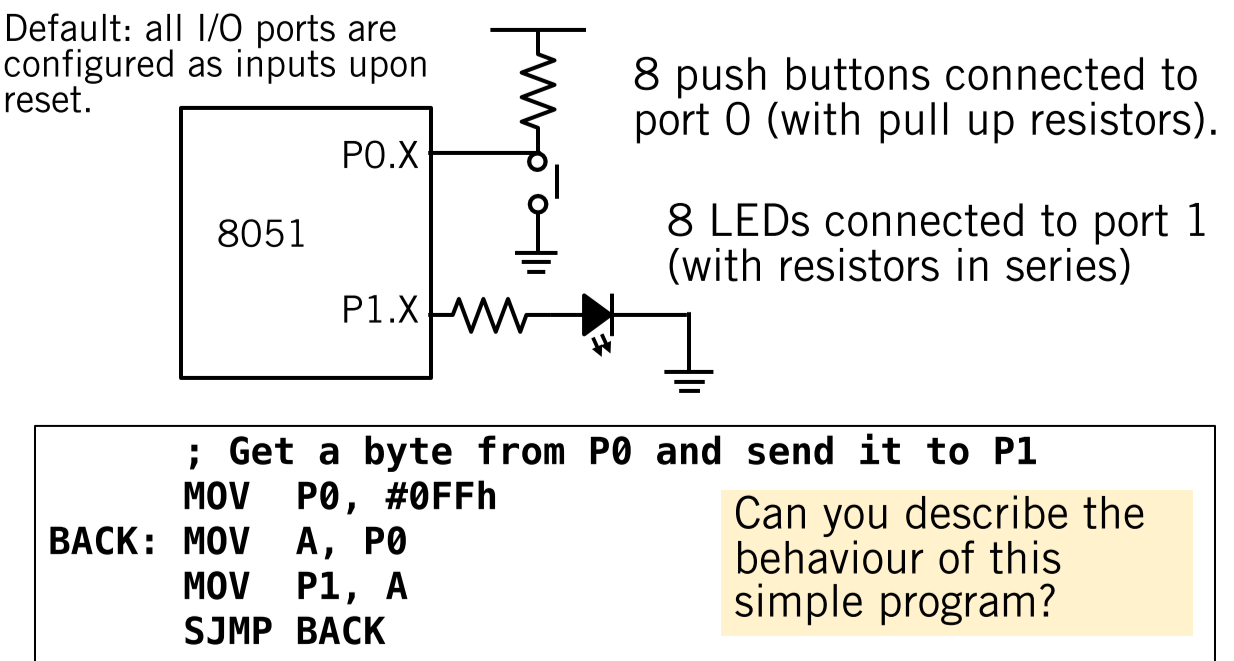


复位情况下，所有端口都初始化为输出端口，如果要让端口配置成输入端口，需要预先将端口的所有位都置1，再进行接受值。原因：将1写入该端口的位中，Q=1，非Q=0，此时晶体管断开，这样才能开始读取外部的电平。

只有端口0作为输入端口时才需要外接上啦电阻（pull-up resistance）

#重要案例—button

当button按下去的时候，端口的值为0。



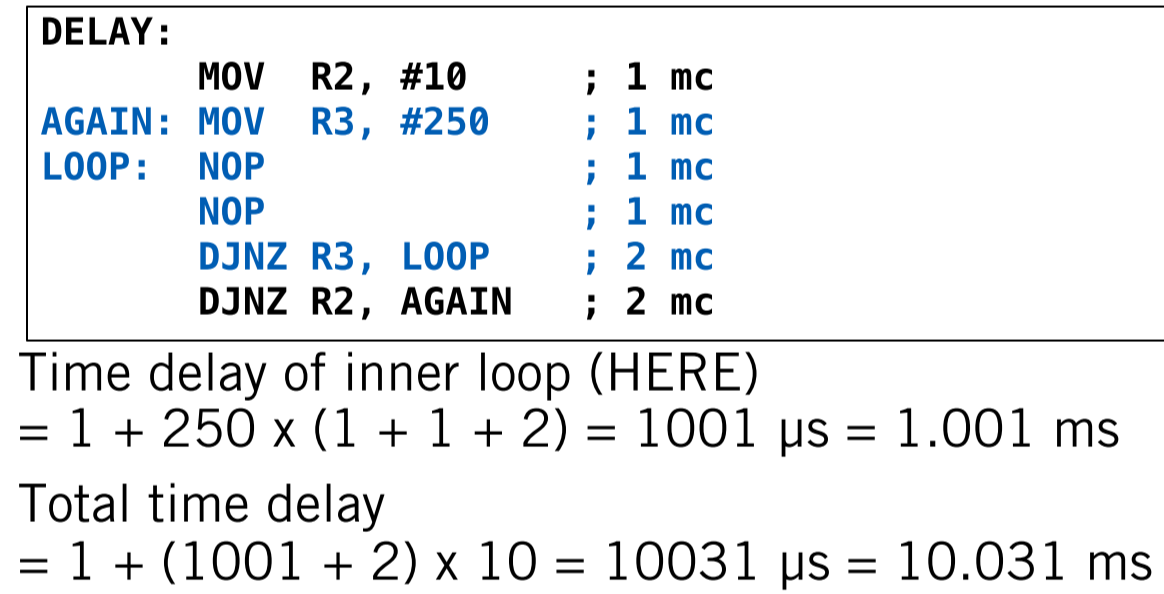
防反跳：当按下去的时候，由于不稳定，电压可能来回变动，我们要避免这种变动，就需要延迟读值，待到稳定的时候读值。

Time Delay结构

假设我们需要一个10ms的延迟，晶振频率为12MHz->一个时钟循环为1/12us，所以一个MC1us

利用循环来设计时延-------

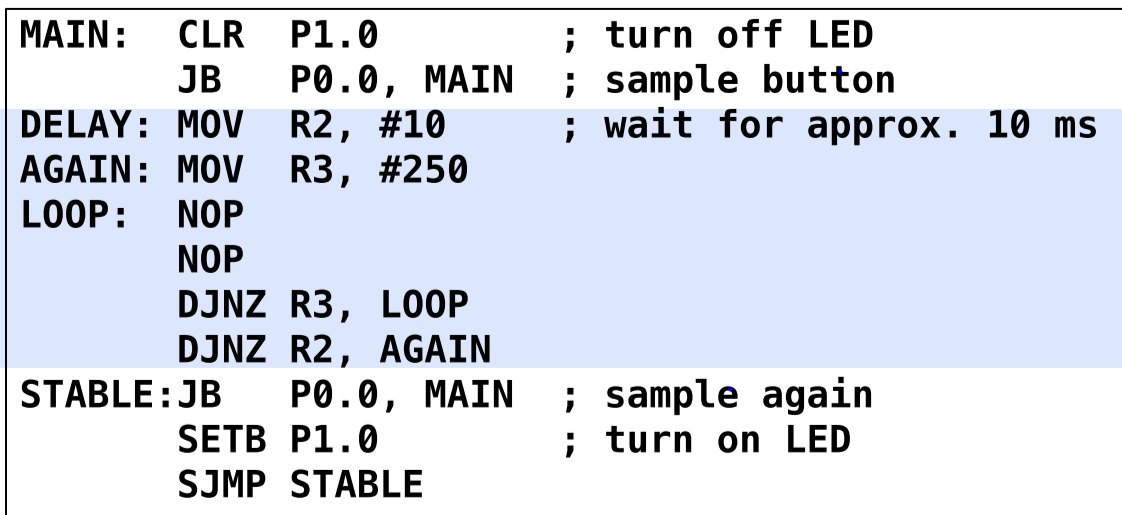
双重循环：



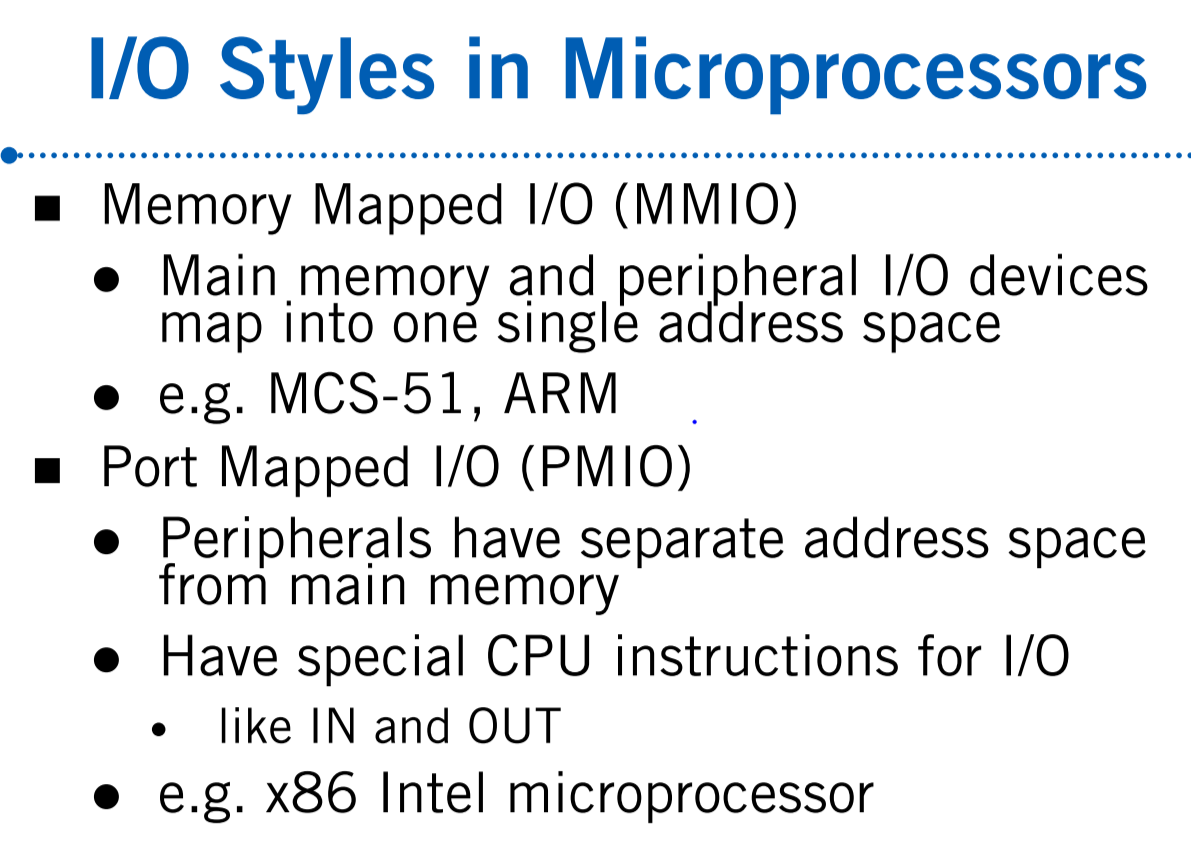
内循环设计目的：为了让一次内循环实现1ms

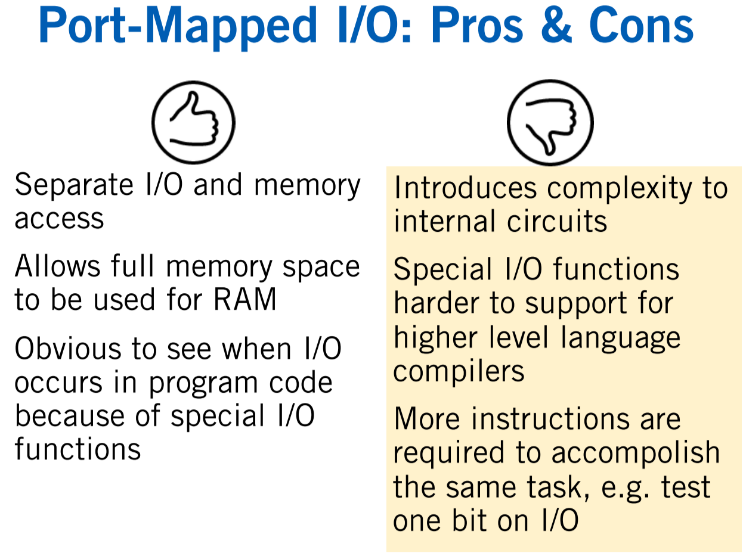
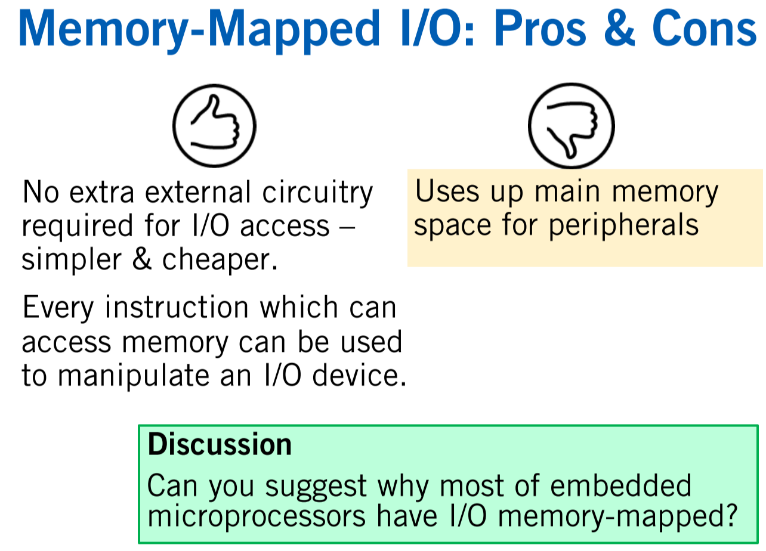
外循环设计目的：让内循环执行10次，共10ms

改过之后的button的代码：



先判断是否有初始0值，如果没有，则不操作，如过有，等待个10ms，再判断是否为0值，如果是，说明不是误触，传值给灯。





#Stack and Subroutine call

什么是Subroutine

A subroutine is a program that may be used many times in execution of a larger program.

单片机在实际程序运行中需要在内部开辟一段缓冲区作为堆栈，以便在子程序调用、中断服务处理等场合保存CPU的运行现场。单片机的堆栈区不是固定的，栈顶的位置由栈顶指针SP指出，复位时，SP为07H单元，堆栈的实际位置从08H开始，即位于工作寄存器内，用户可以给SP赋新值，用来确定堆栈的起始位置。堆栈区最大到80H（不包括）

如何算堆栈的大小：7FH-SP byte

堆栈有3个具体的功能：

1. 保护断点：即保存PC值。单片机在调用子程序操作或执行中断操作后，最后都要返回主程序。因此应该在调用程序之前，预先吧主程序的断点（即当前执行子程序调用命令的吓一跳指令的PC值）保存在堆栈中，为程序的正确返回做好准备。

PUSH PCL; 先将SP++，然后压入PCL

PUSH PCH; 先将SP++，然后压入PCH

2. 现场保护：单片机在执行子程序或中断服务子程序后，需要用到一些RAM单元和寄存器单元，可能会造成原有单元的破坏，这时候就可以使用堆栈先将其压入堆栈，保护起来，要恢复数据时，再POP出来。

3. 临时暂存数据：在程序设计时，有些中间变量或数据需要暂时保存，以备下一步数据处理，待数据处理完成后，就可丢弃这些数据，为了减少开销，可以暂时存储在堆栈中。

\*CALL

ACALL addr11；短调用指令

双字节的绝对调用指令，只能在2KB的范围内调用子程序。先PC自加2，指向下一条指令的地址，然后把PC值压入栈中（SP +2，先压低字节，再压高字节），再然后转向子程序。

LCALL addr16；长调用指令

三字节的绝对调用指令，在64KB范围内调用子程序。PC+3，其他一样。

RET；子程序返回指令

RETI；中断服务程序返回指令

单字节的返回指令，指令的功能是从堆栈中连续两次弹出数据送入PC的高位和低位。

#C51

#Timer

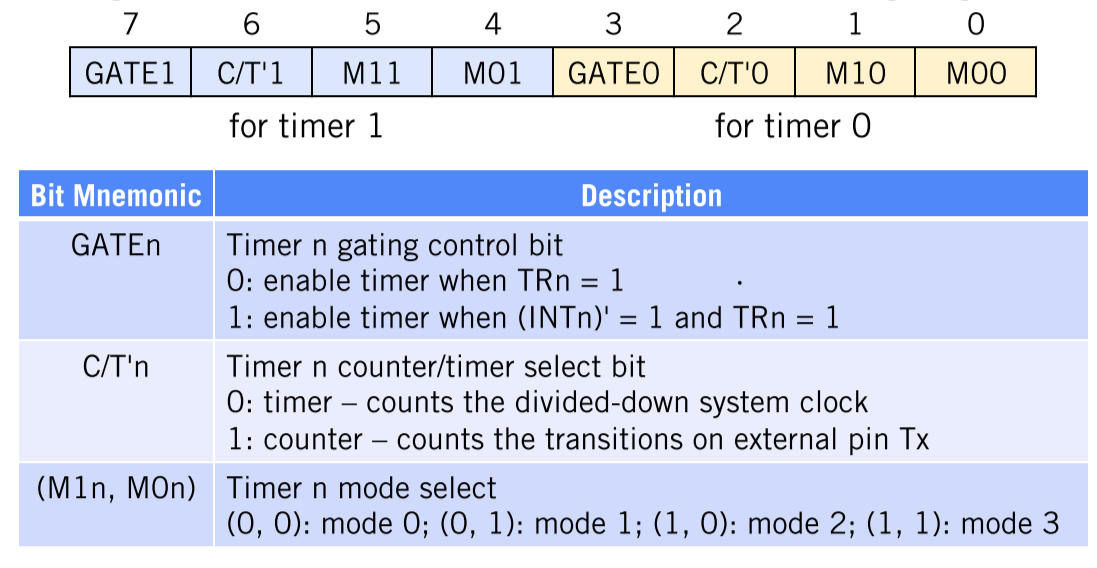
为什么使用定时器

使用循环浪费资源；当循环的时候处理器无法做任何事情；有一个定时器更好。

8051有两个16位加法定时器，**T0由两个8位的特殊功能寄存器TH0、TL0构成。T1由TH1、TH0构成。**当启动定时器时，定数器从其中的初值开始计数。

**TMOD**

TMOD工作方式控制寄存器用于选择T0和T1的工作模式和工作方式，它的字节地址是89H不可寻址。



工作模式有4种，由M1n，M0n决定

0, 0 工作模式为0，13bit计数器。由全部的高8位和5位低位构成。1FFF后溢出。

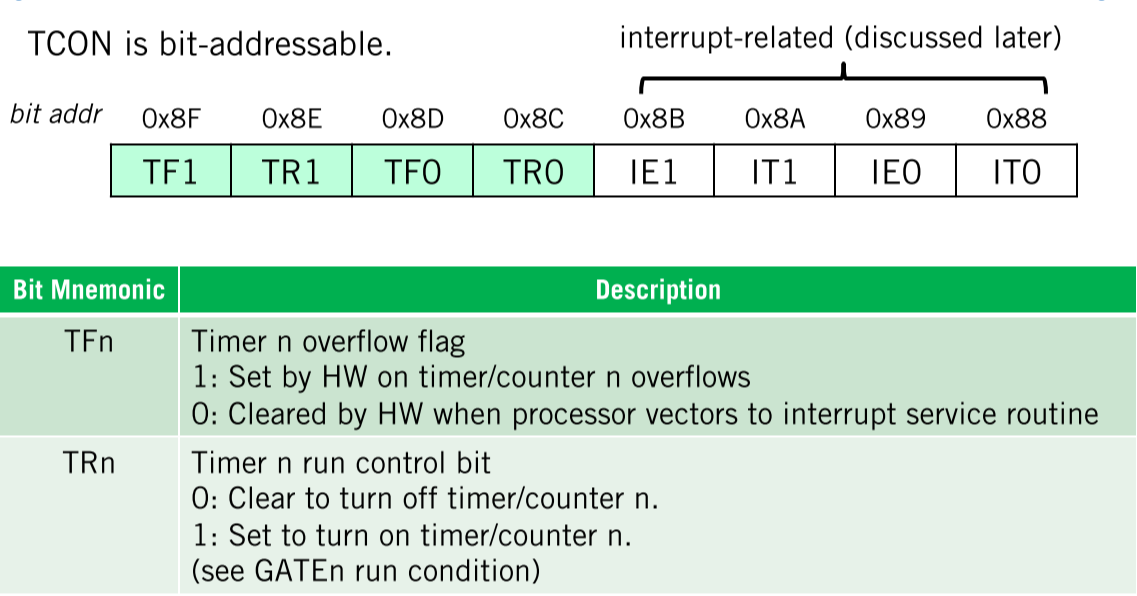
0, 1 工作模式为1，16bit计数器。全部的16位参与运算，FFFF后溢出。

1, 0 工作模式为2，8位计数器，低位运算，高位保留初始值方便重复使用。FF后溢出。

1, 1 工作模式为3，拆成两个独立的8位计时器使用。

**TCON**

TCON控制寄存器的字节地址为88H，可位寻址。

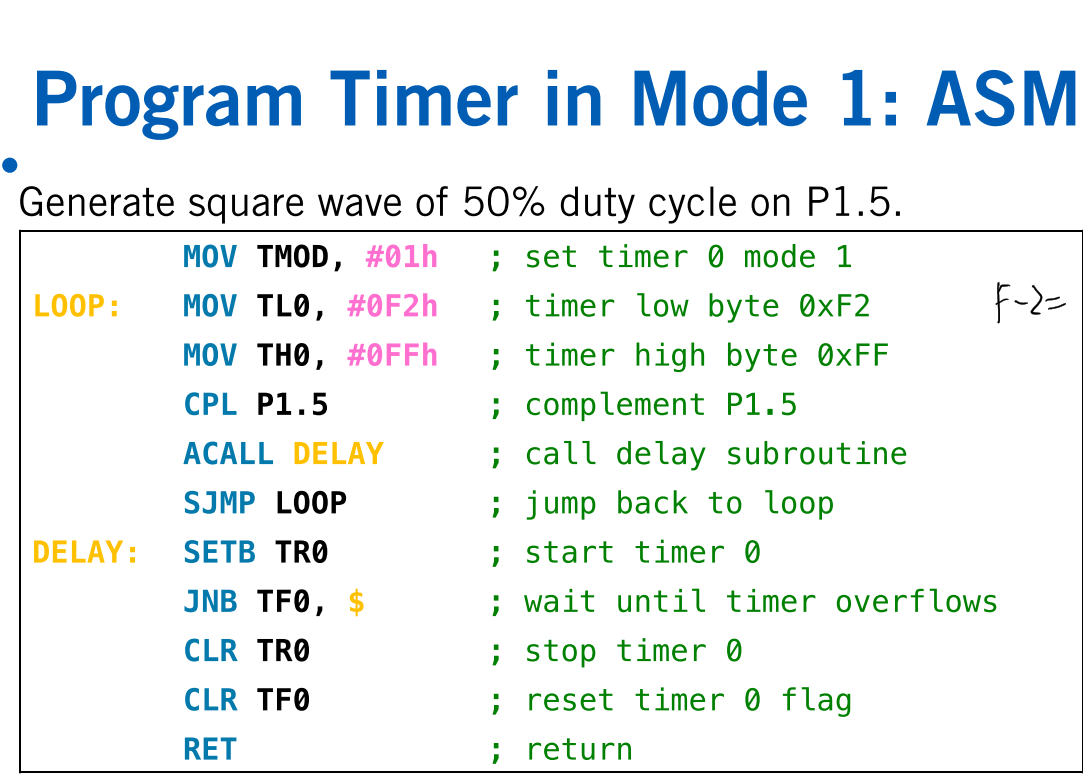


TF1、TF0：T1、T0的溢出标志，溢出时置1。当置1时，可中断或查询。相应后清零。

TR1、TR0：运行控制位，置1表示启动定时器。

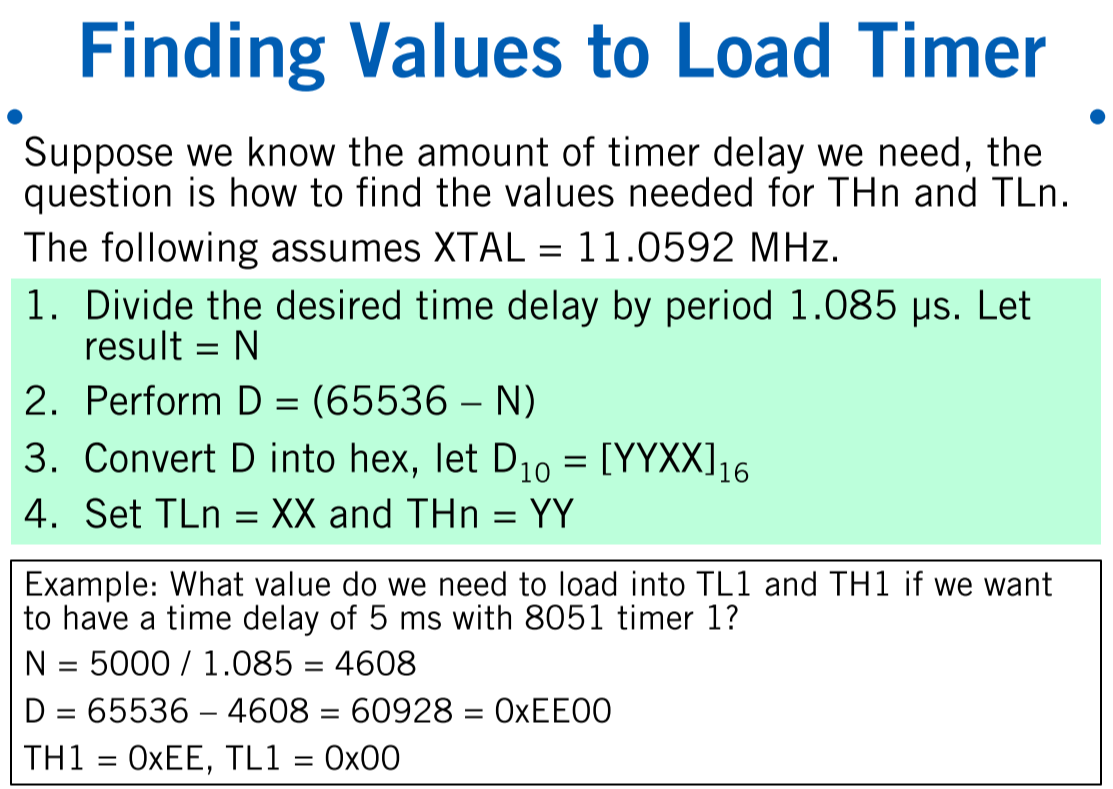
低4位与外部中断有关。

Timer编程示例：



需要记住的是FFFF（16）-0=65536（10）

**\*如何根据自己需要的时间来确定传值TH、TL**



首先根据晶振频率算出时钟频率，时钟频率×12等于机械周期

现在我们有一个机械周期的时长，然后让我们需要的时间除以机械周期，算出我们需要多少个机械周期

然后使用FFFF=65536减去这个机械周期数，得到的结果化为16进制

这个16进制的高8位为TH，低8位为TL。

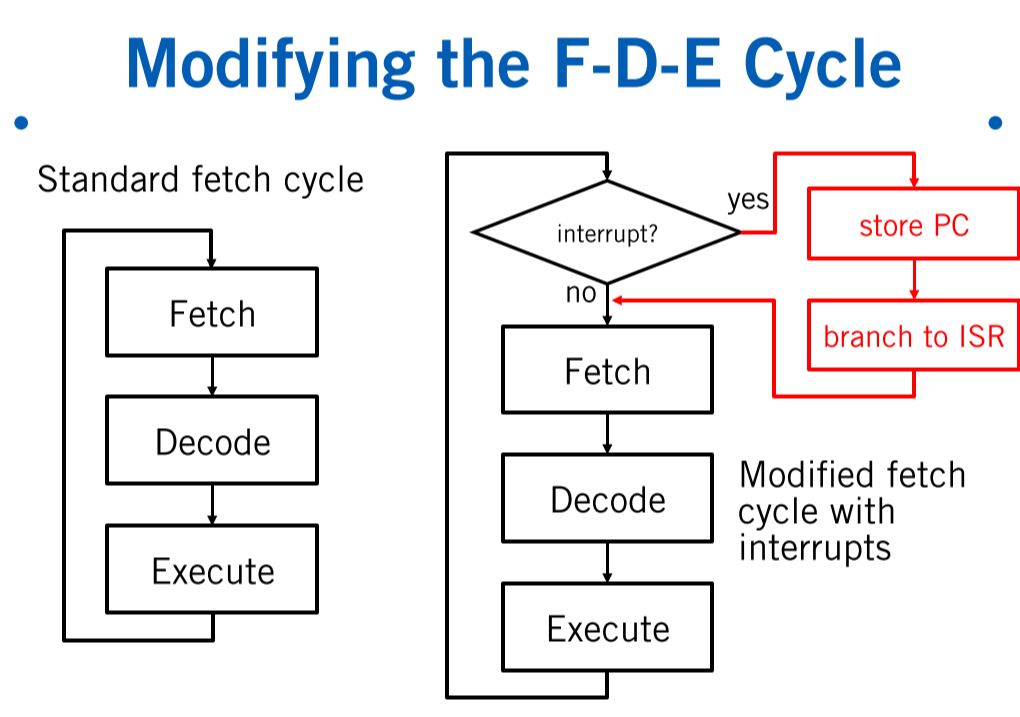
#轮询和中断 polling and interrupt

如何从设备中得到数据？

轮询：不断的询问（用循环实现）

Interrupt

所谓中断，是指当CPU正在处理某件事情时，外部发生的某一事件，请求CPU立刻去处理，于是CPU暂时终止当前的工作，转去处理所发生的事件。中断服务处理完该事件后，再回到原来被终止的地方，继续原来的工作，这样的过程称为中断。



8051中可以引起中断的事件：

1. Reset（程序无法实现）

2. External 0 由INT0引脚输入，中断请求标记为IE0；

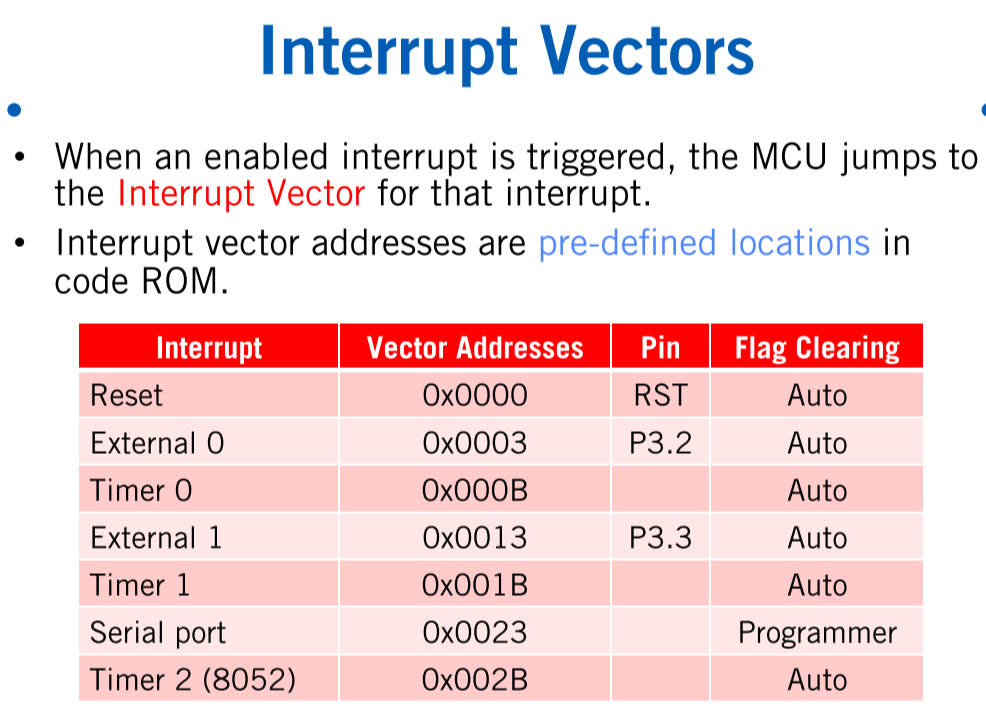
3. External 1 由INT1引脚输入，中断请求标记为IE1；

4. Timer 0 中断请求标记为TF0

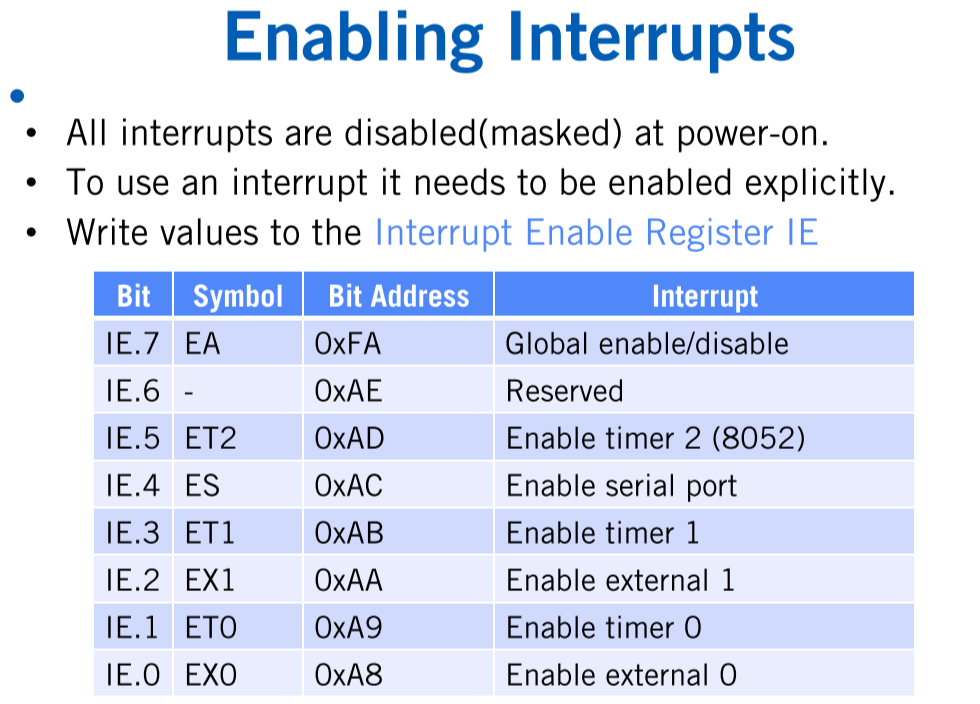
5. Timer 1 中断请求标记为TF1

6. Serial port TX/RX 中断请求标记为T1或R1

当中断发生了，根据中断类型的不同，程序将会调到不同的地址！这些地址被称为中断向量！

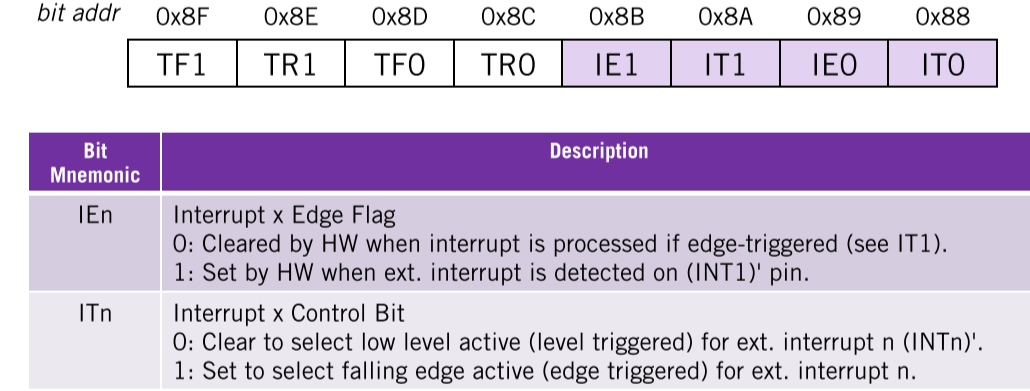


**这个是IE—interrupt enabling的位数对应关系**



比如说IE=10000100B 表示启动EX1中断。记住它的结构，从低位到高位，0到1，外部和定时器交替出现。

让我们回到TCON



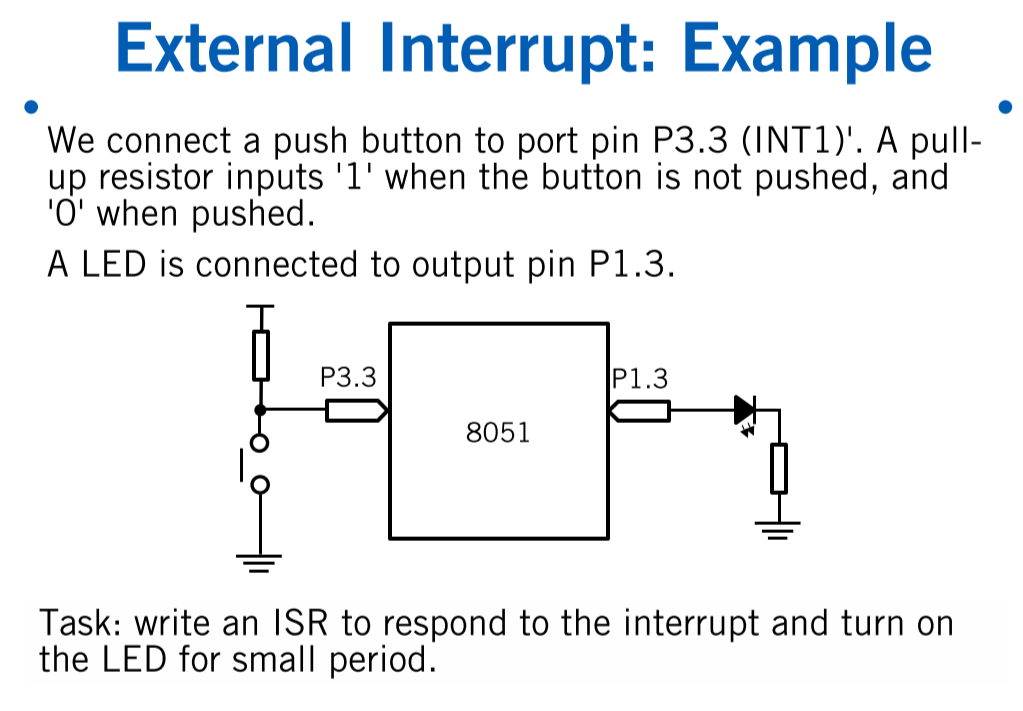
* **IT0：外部中断0中断触发方式选择位。**

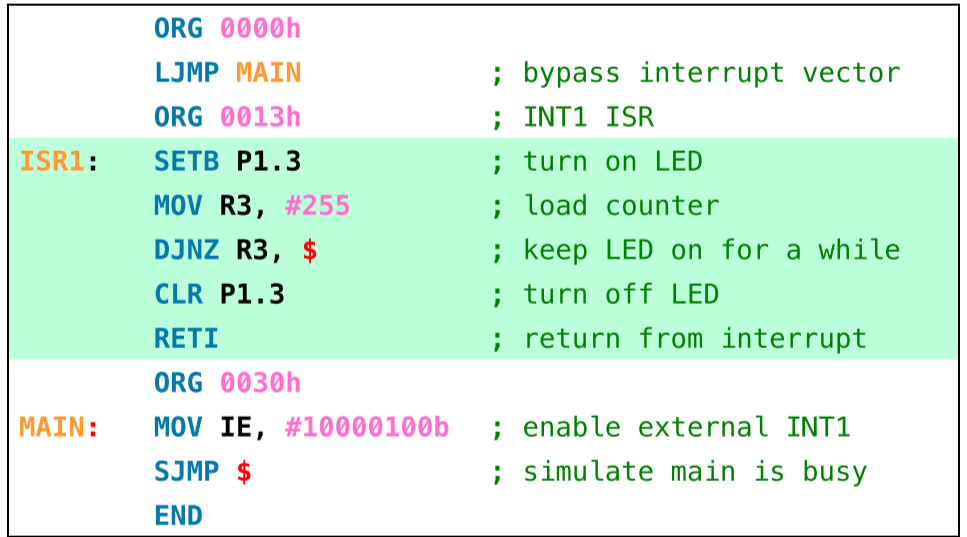
0，电平触发方式，引脚INT0上出现低电平时申请中断；

1，为跳变触发方式，INT0上出现从高电平到低电平的跳变时向CPU申请中断。

* **IE0：外部中断0请求标志位。**1，表示外部中断0正在向CPU进行申请中断。

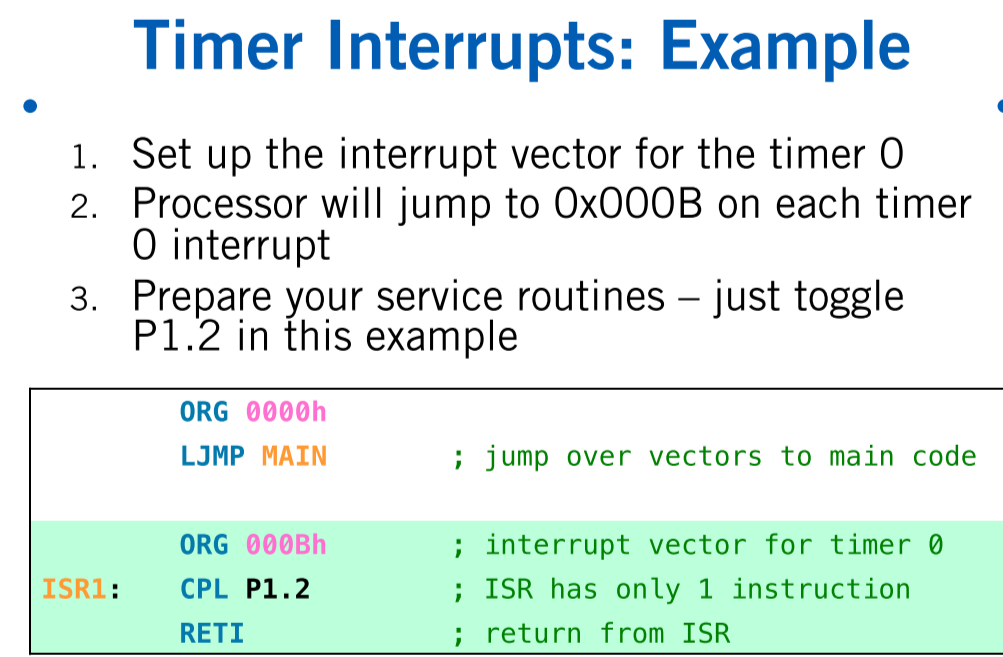
外部中断例子





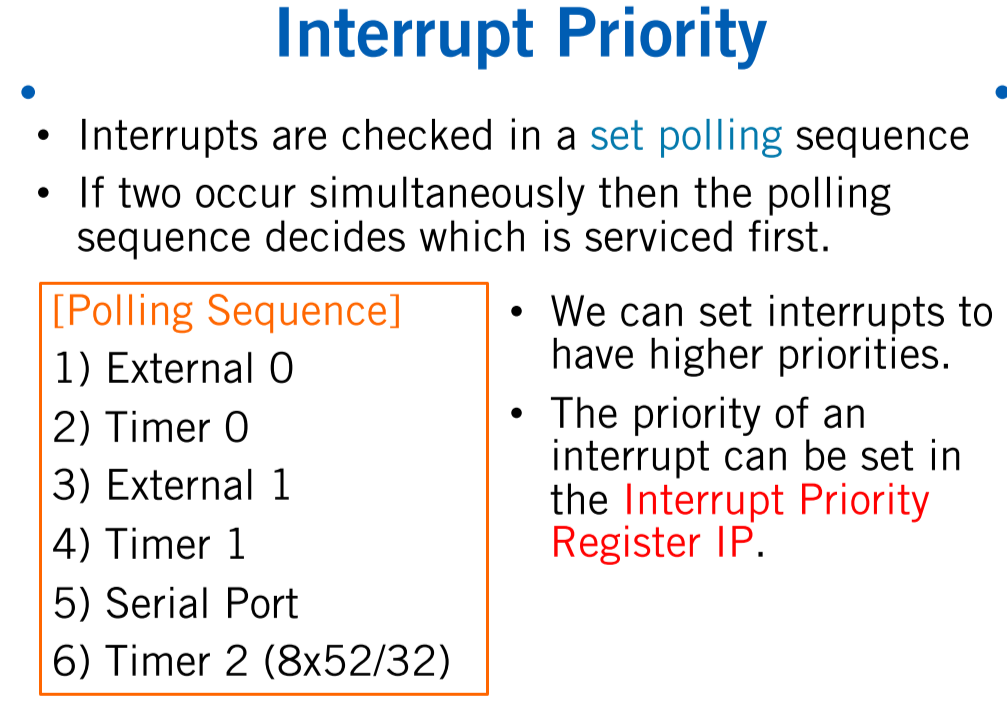
这里的0013H就是外部中断1的中断向量。IE要设置为10000100B

定时器中断例子

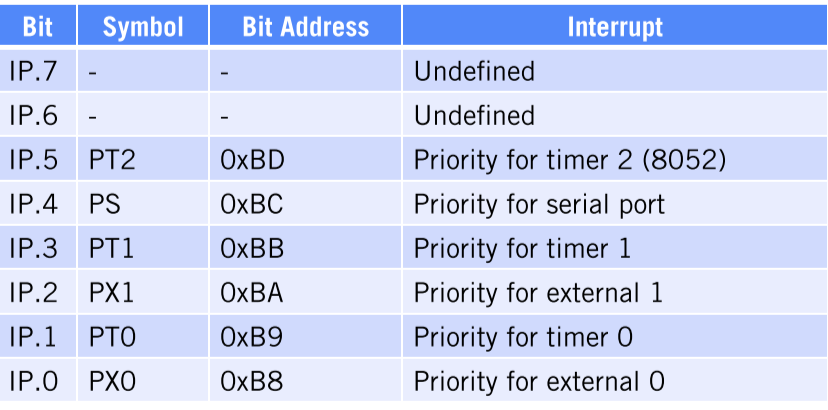


这里的000B就是Timer 0的中断向量。IE要设置为10000010B，别忘记了TMOD设置定时器的模式。

\*中断优先级



设置中断优先级的寄存器—IP

设置顺序与IE一样

#串口通信

Simplex

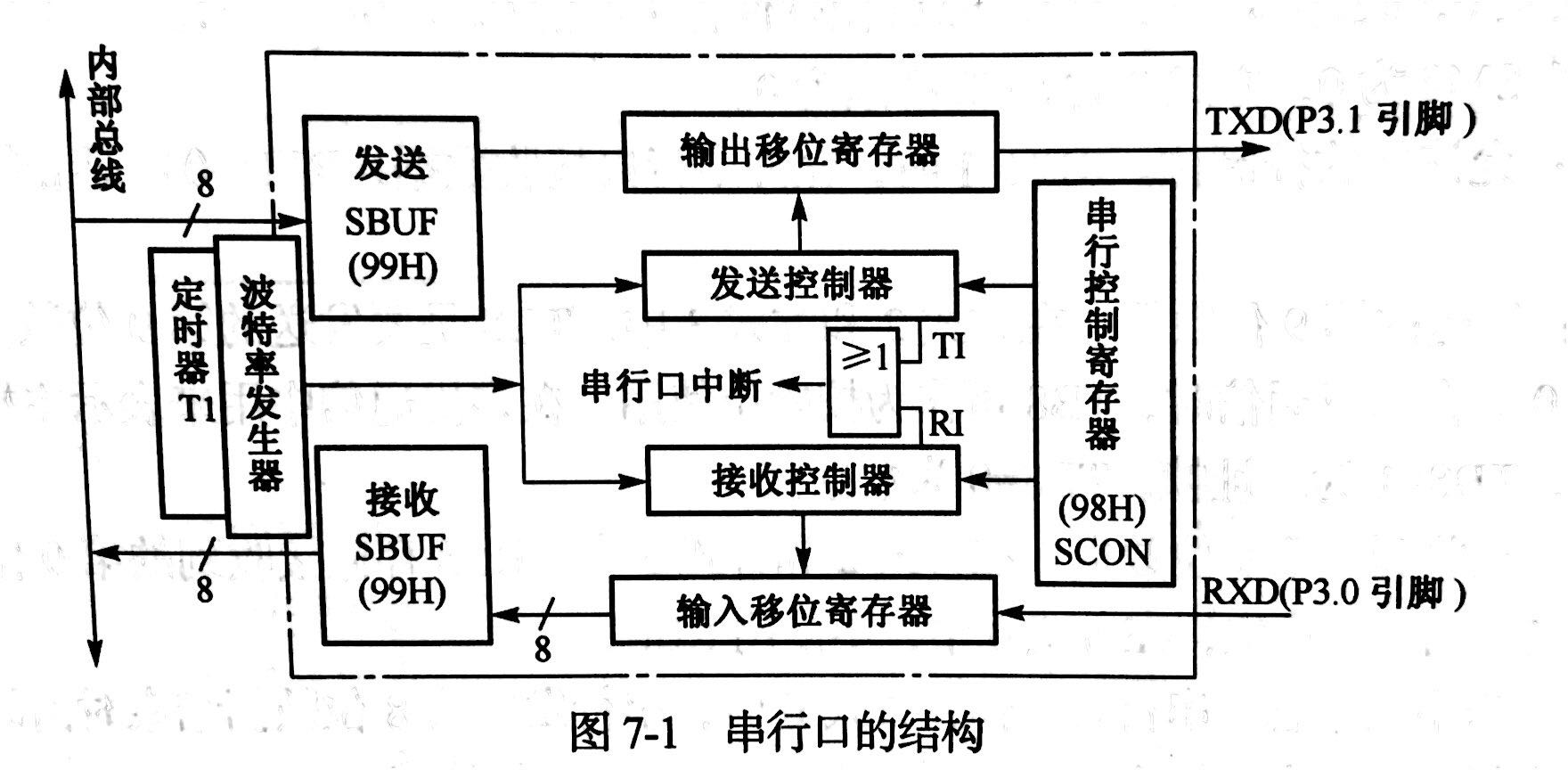
Half duplex

Full duplex

Asynchronous：使用的时钟分开

Synchronous：使用的使用相同

而8051内部有一个强大的全双工异步通信串口



* **SBUF：**

实际上是两个独立的发送缓冲器和接收缓冲器，可同时发送、接收数据，实现全双工串行通信，两个缓冲器共用一个特殊功能寄存器字节地址99H。串行发送时，从内部总线向发送缓冲器SBUF写入数据；串行接收时，从缓冲器SBUF读出数据。

* **串行口控制逻辑：**

1. 接收来自波特率发生器的时钟信号

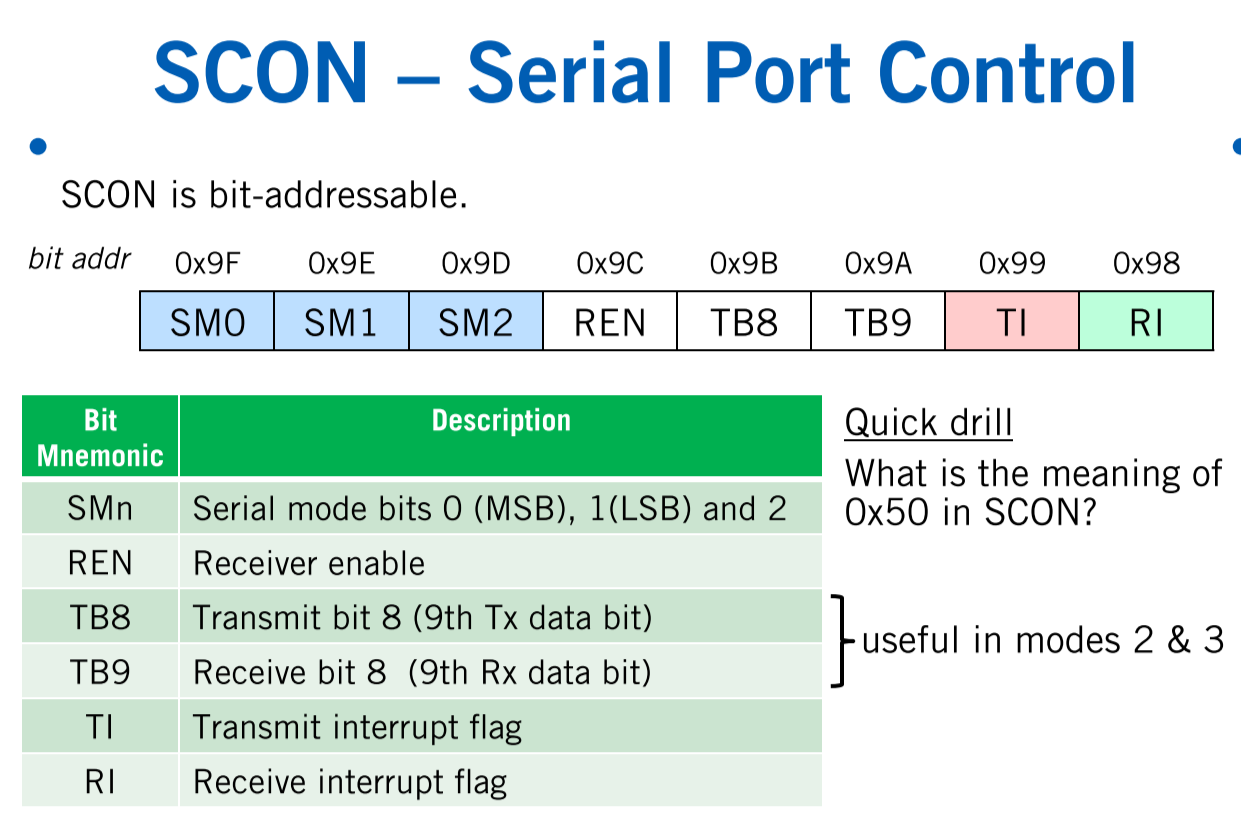
2. 控制内部的输入移位寄存器，将外部的串行数据转换为并行数据送入接收缓冲器SBUF并控制产生接收结束标志RI。

3. 控制内部的输出移位寄存器，将发送缓冲器SBUF中的并行数据转换成串行数据输出，并控制产生发送结束标志TI。

* **控制寄存器：**

控制单片机串行口的控制寄存器主要是串行口控制寄存器SCON，另外还有一个电源管理寄存器PCON。

SCON



SM0, SM1：串行口工作方式控制位。

SM2 ：多机控制位。

REN：Receive enable。允许接收控制位。

TB8：发送的第9位数据。

TB9：接收的第9位数据。

TI：发送中断标志。

RI：接收中断标志。

串行口的工作方式：



模式0不可用在全双工。

模式1：

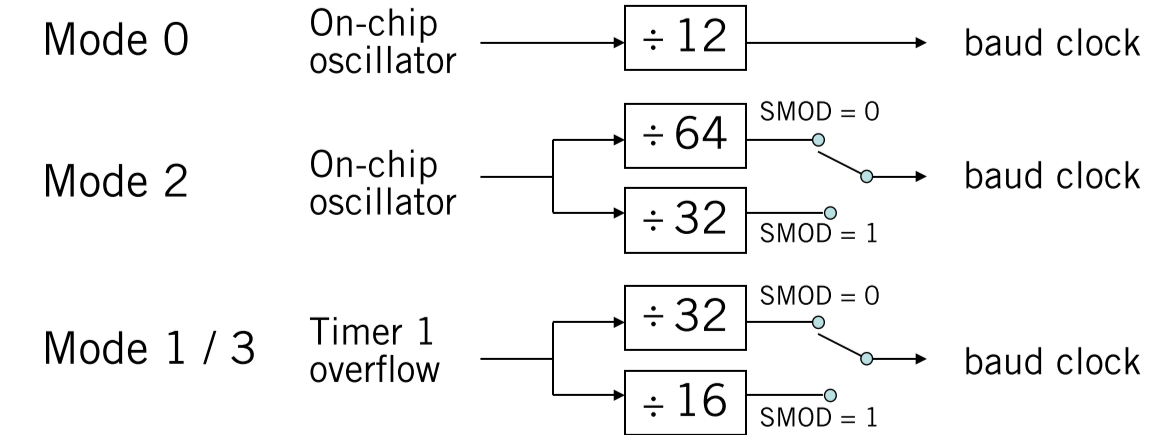
方式一收发的一帧数据为10位：1位起始位，8位数据位，1位停止位，按照先低后高的顺序收发。

模式2和3：

它们的工作原理相似，唯一的区别是模式2的波特率是固定的。发送一帧的数据是11位：1位起始位，8位数据位，1位可预置为1或0的第9位数据，1位停止位。先低后高。

\*波特率的计算

PCON的第7位为SMOD，可取0或1，当其为1时，波特率翻倍，所以SMOD也称为波特率倍增位。



一般选用计时器在工作模式2作为震荡发生器

例题：1200baud/s，芯片为12MHz的晶振频率，SMOD为0，使用方式1，求计时器初始值：

先算出MC：MC=1us，所以1s有1000000个MC也就是1MHz，

1M/N/32=1200 baud

N=26

TH1 = -26 = 0XE6h

补：错误率

实际上的波特率为1M/(26\*32) = 1201.9

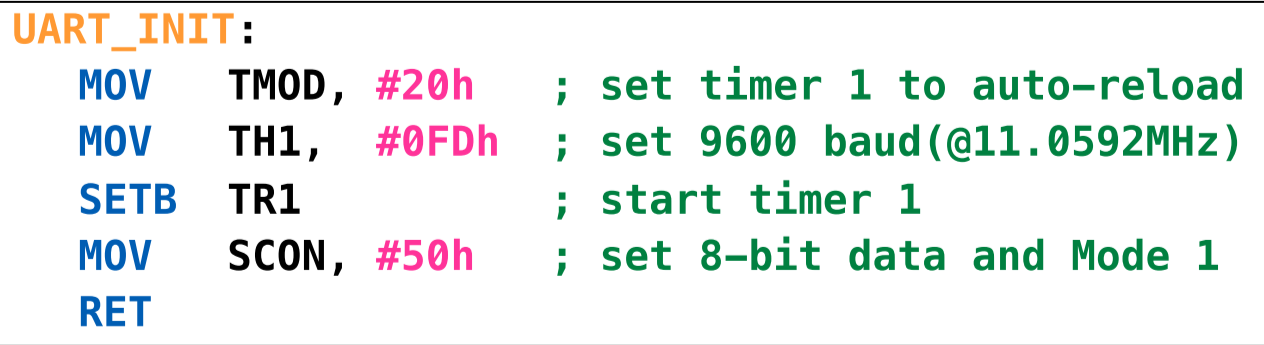
ERR=(1201,9-1200)/1200=0.16%

此时我们可以使用11.0592的晶振频率。

\*串口编程

9600的波特率对应的计时器TH1为#0FDh

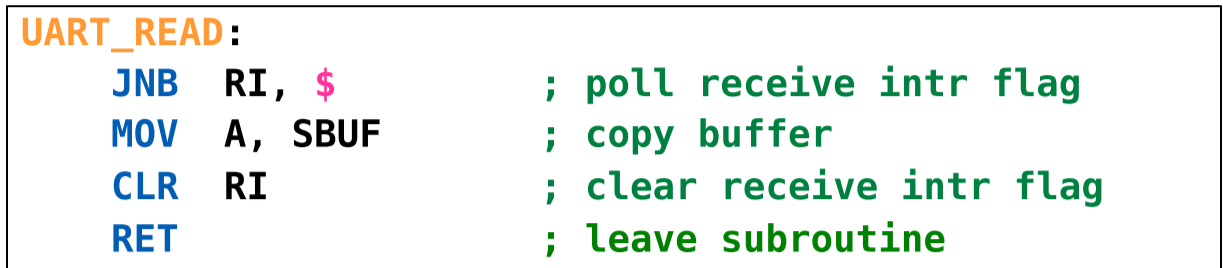
初始化：



传数据

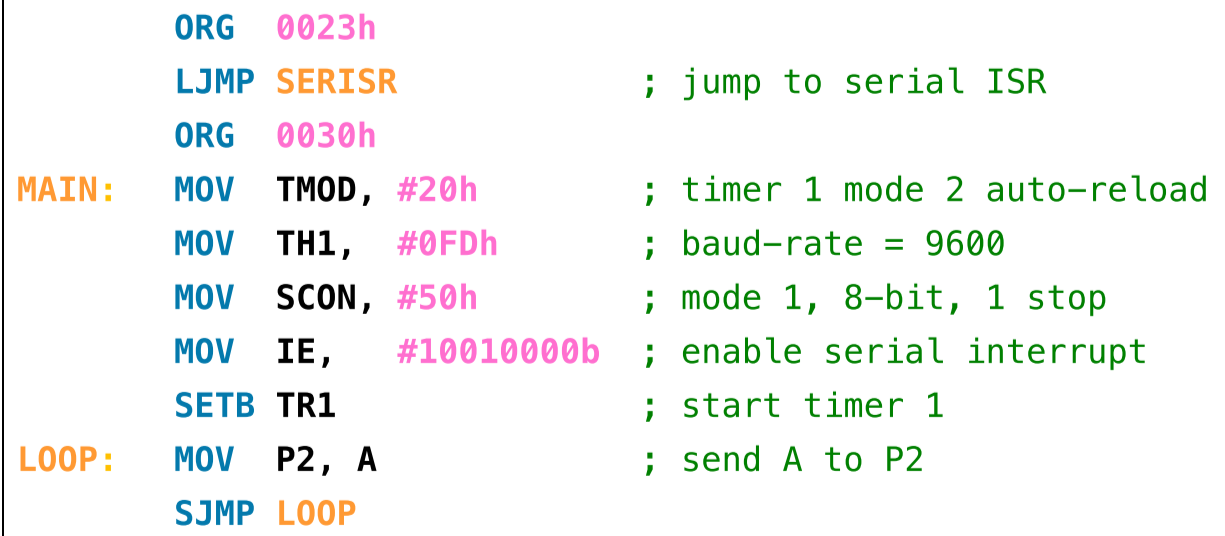


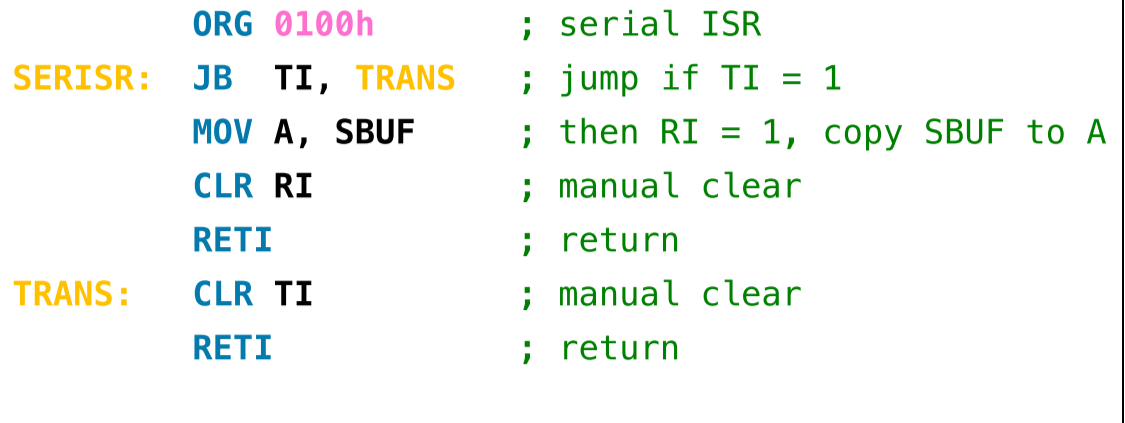
收数据



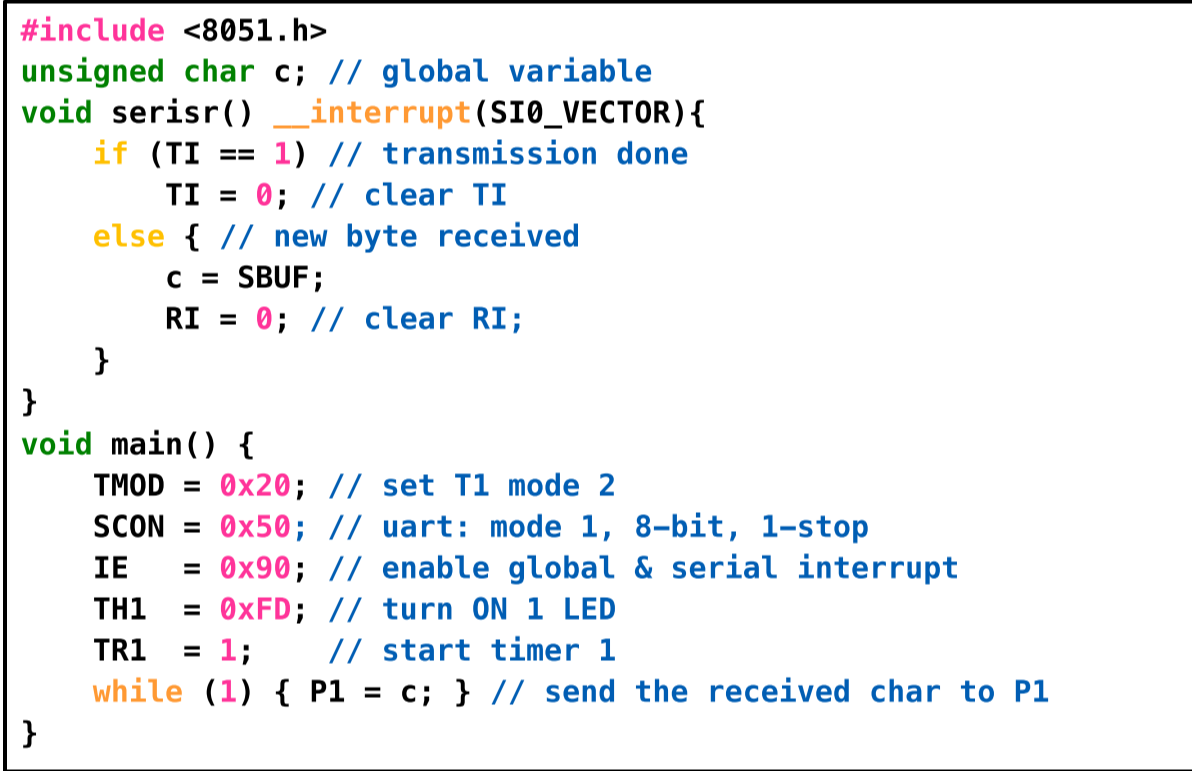
\*串口中断

串口中断向量为0023，TI和RI任意一个为1的情况，都会触发串口中断，并且这两个值必须人为编程置0.



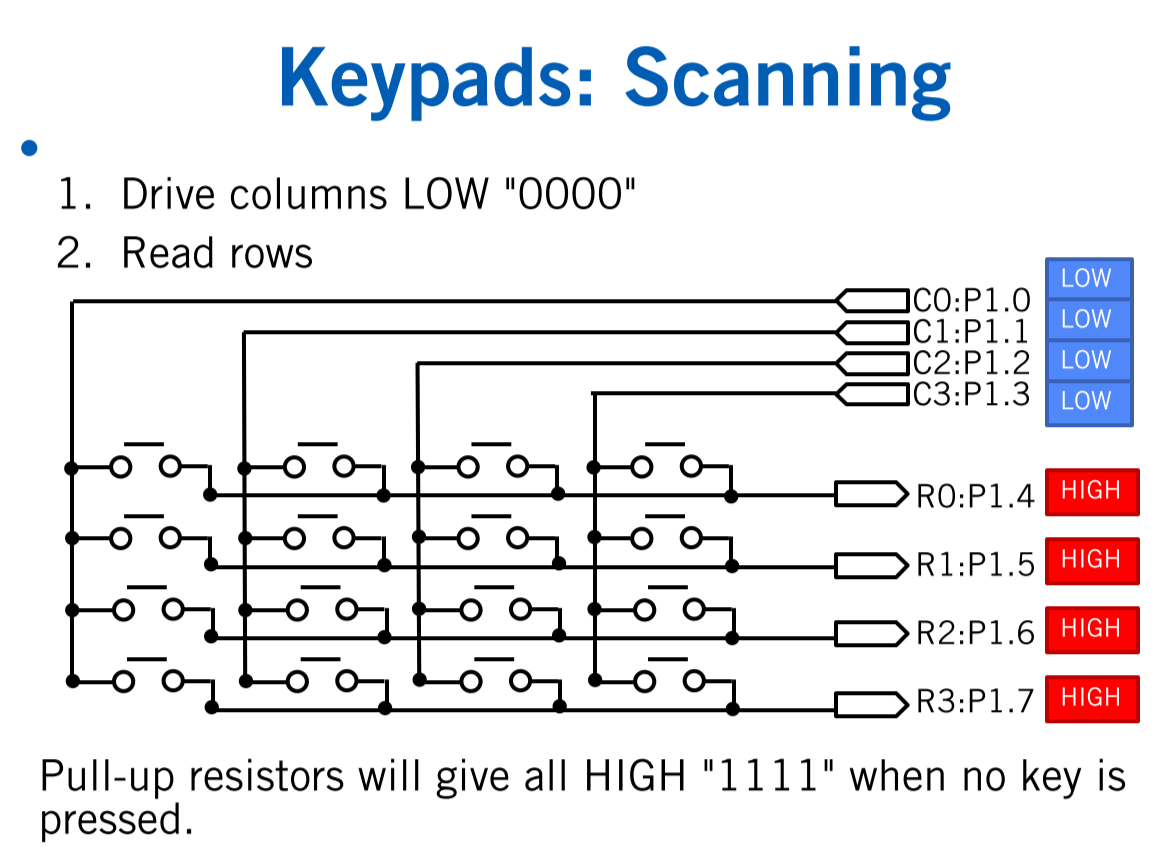


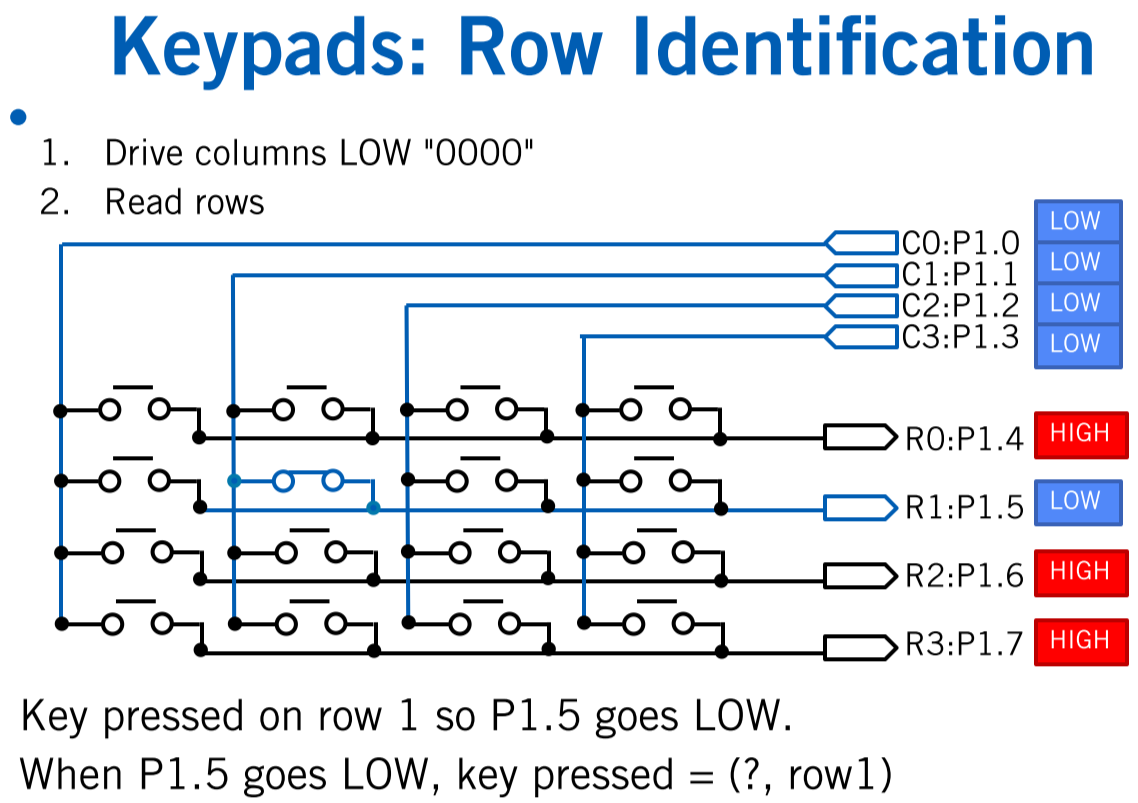
C51代码：

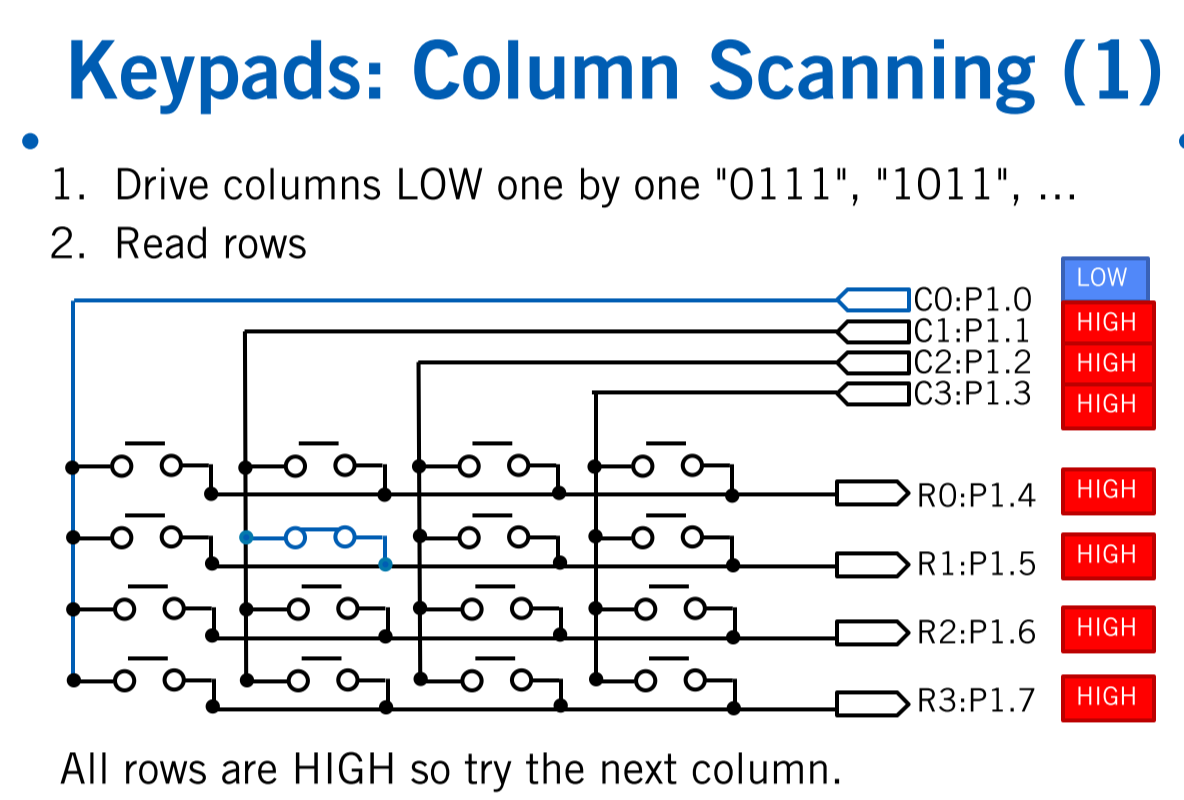


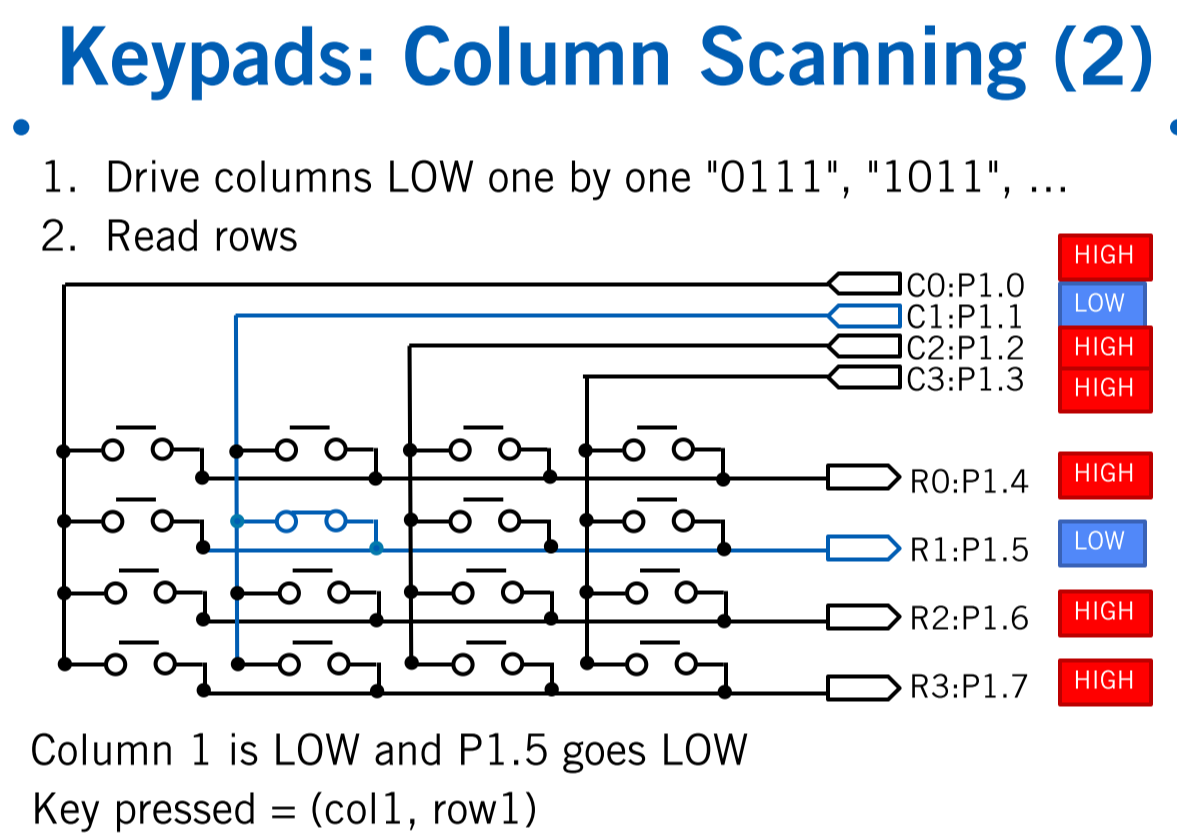
#键盘和LCD

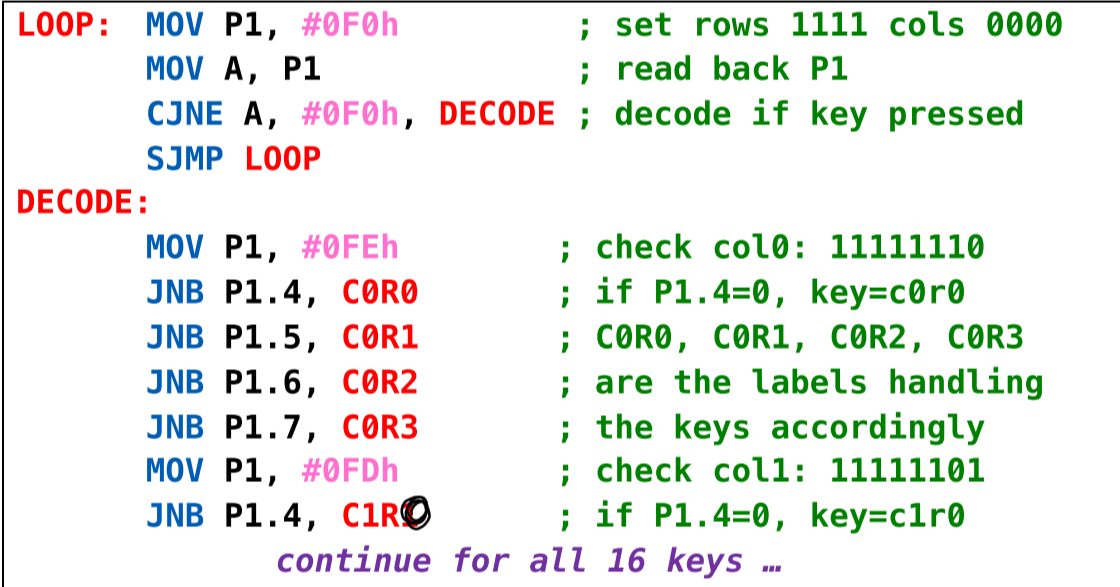
键盘：







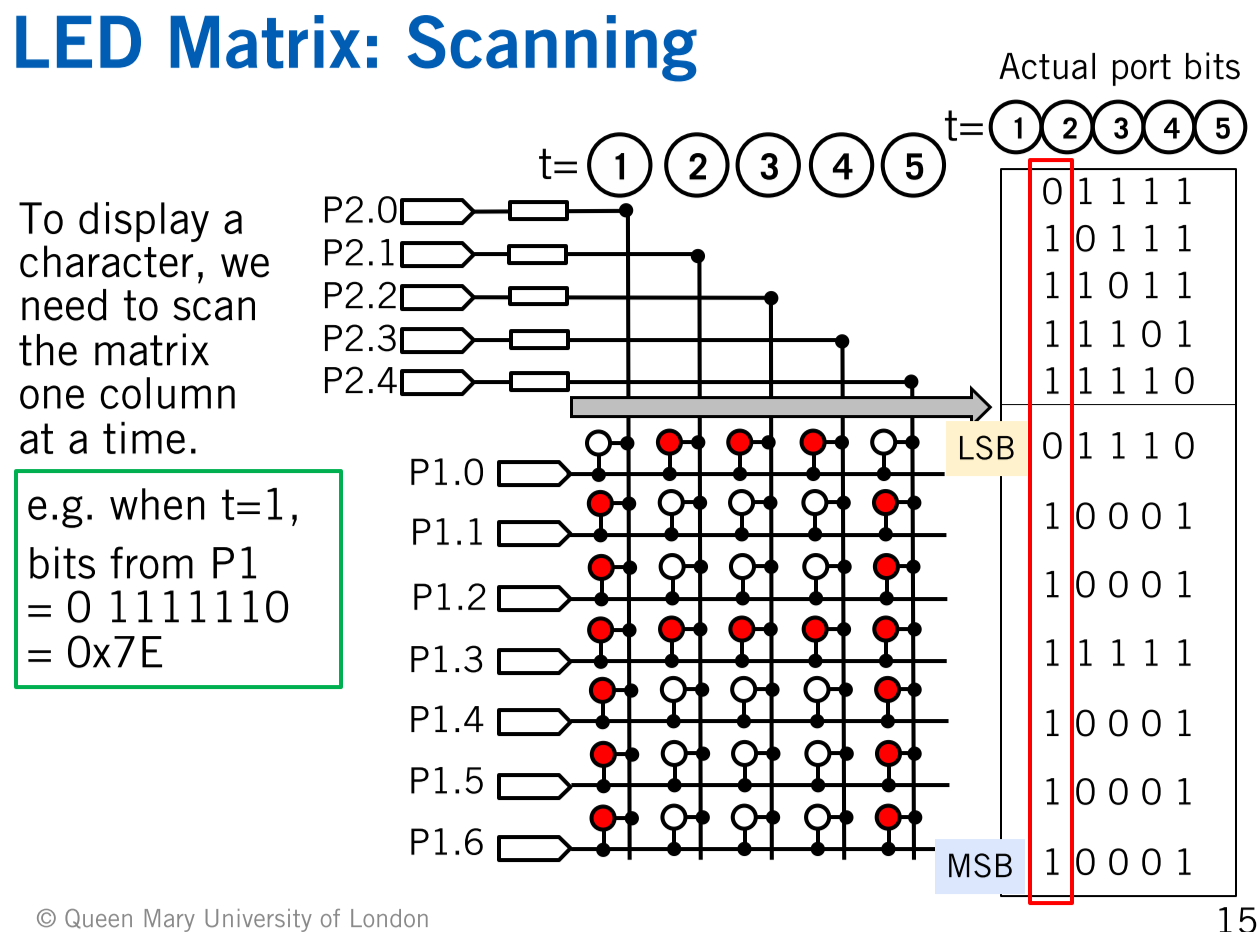


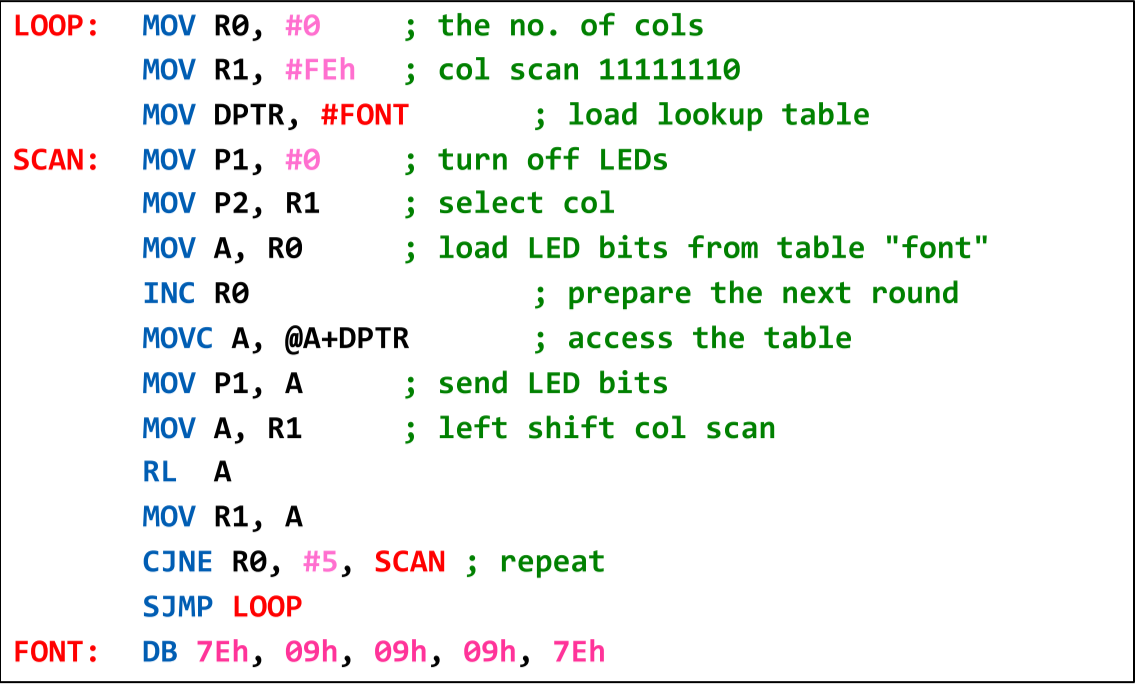


\*Display

首先要明确，LED灯不是同时亮的，通常是同一时间每行只亮一个LED。

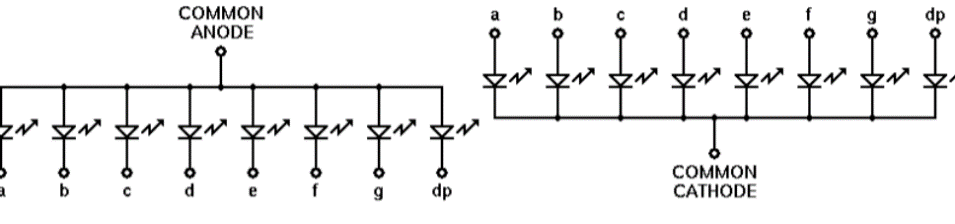
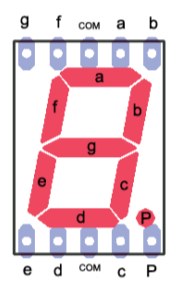
Persistence of vision：视觉残留，当频率大于50Hz的时候，感觉没有闪。





7-segment Display

通常有两种结构，**共阳极common anode（端口低电平点亮）和共阴极common cathode（端口高电平点亮）**



实际结构图：

