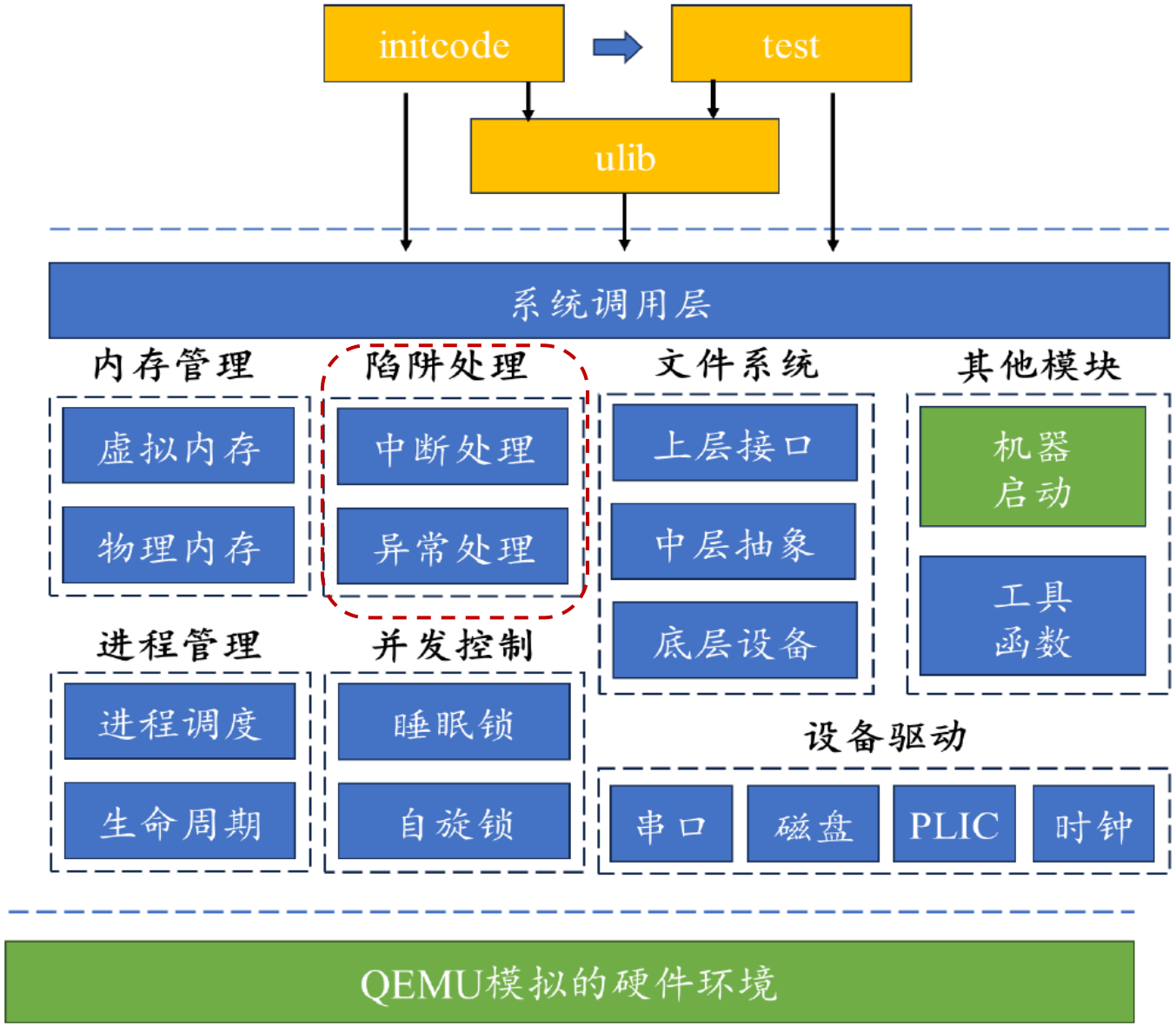
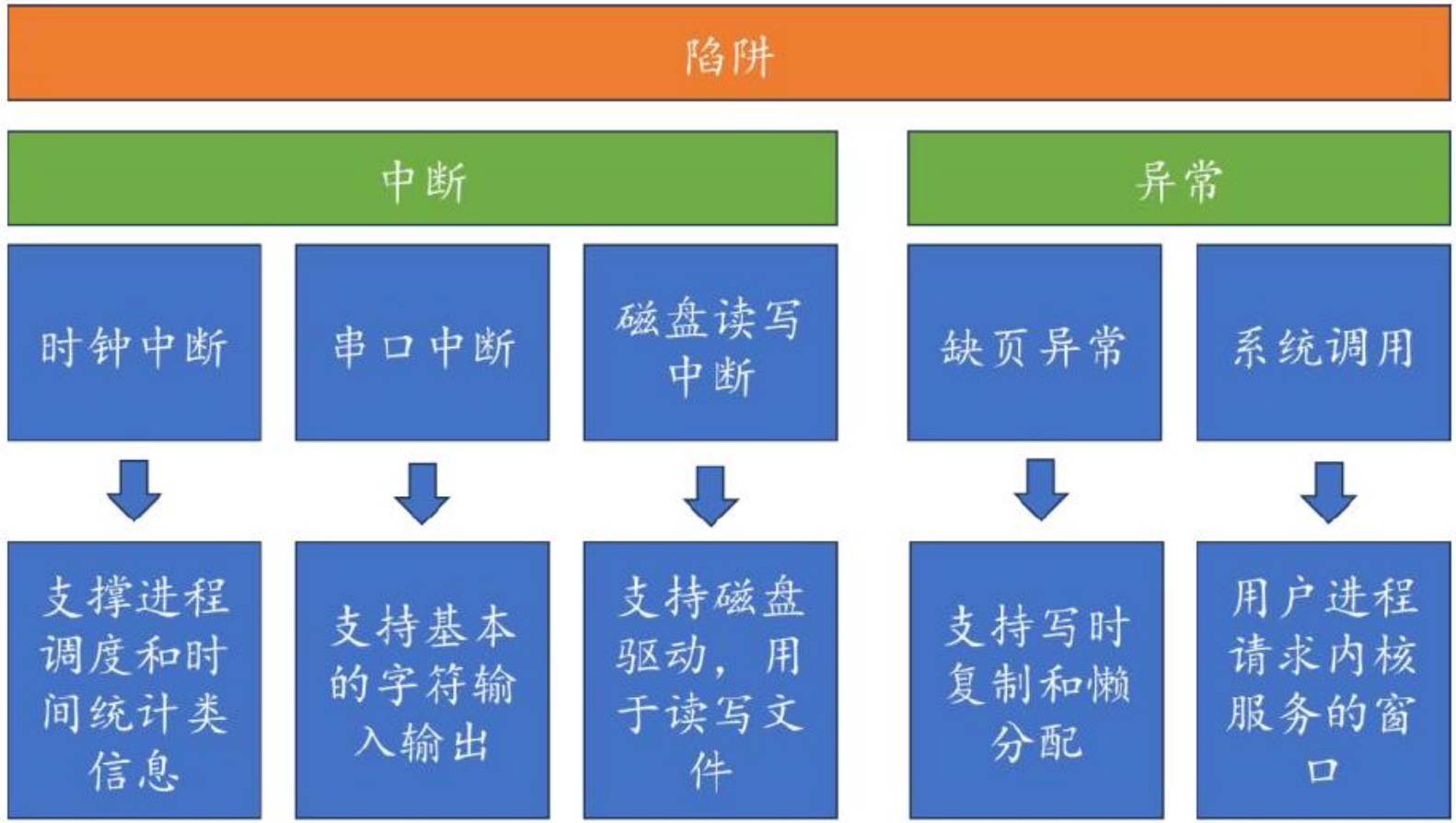
实验三：中断处理与时钟管理



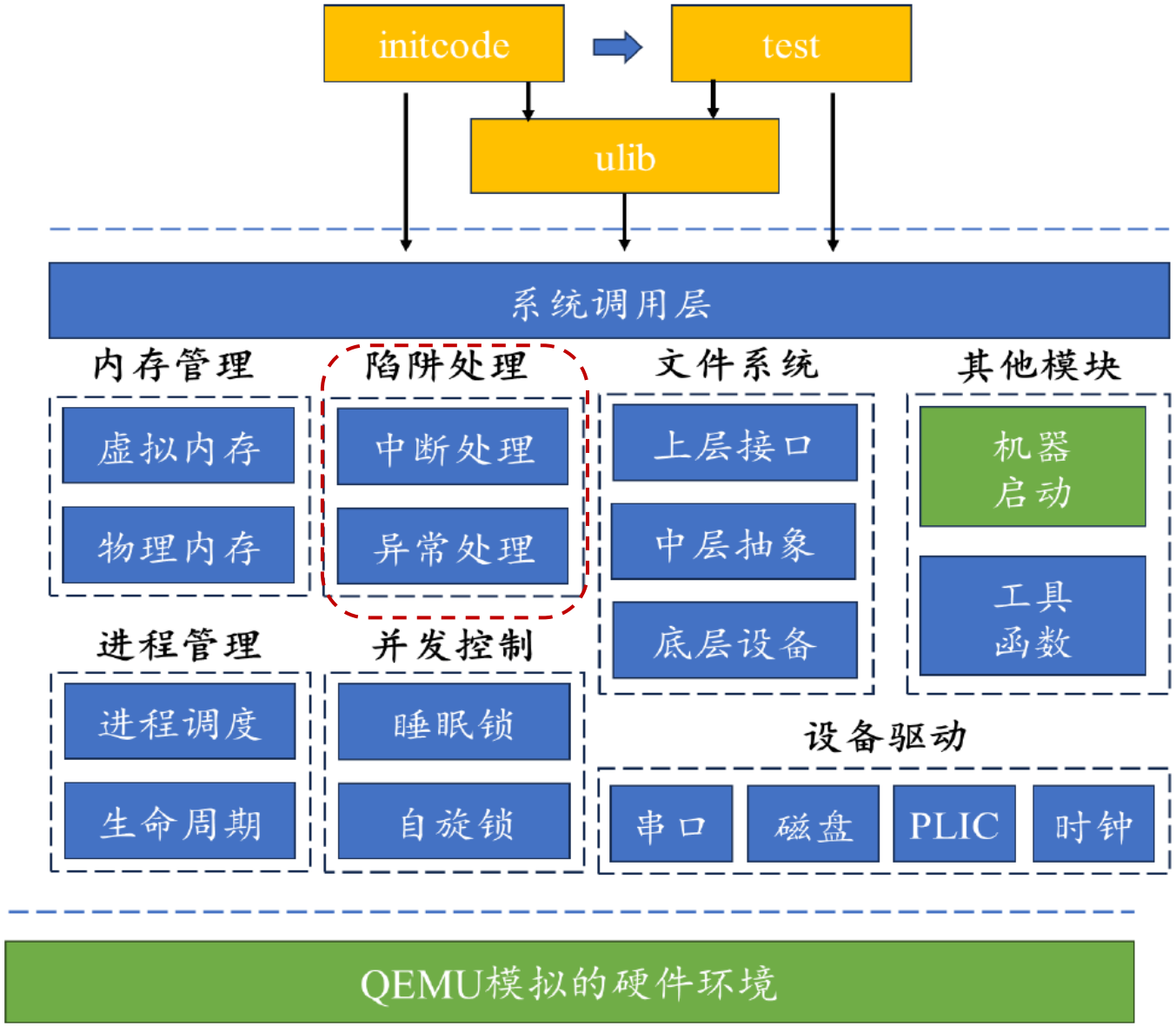
- RISC-V体系结构规定了三种特权模式：用户模式（User mode，U-mode）、监管者模式（Supervisor mode，S-mode）、机器模式（Machine mode，M-mode）。

实验三：中断处理与时钟管理

- 在RISC-V里, 陷阱(trap) 分为 中断(interrupt) 和 异常(exception)



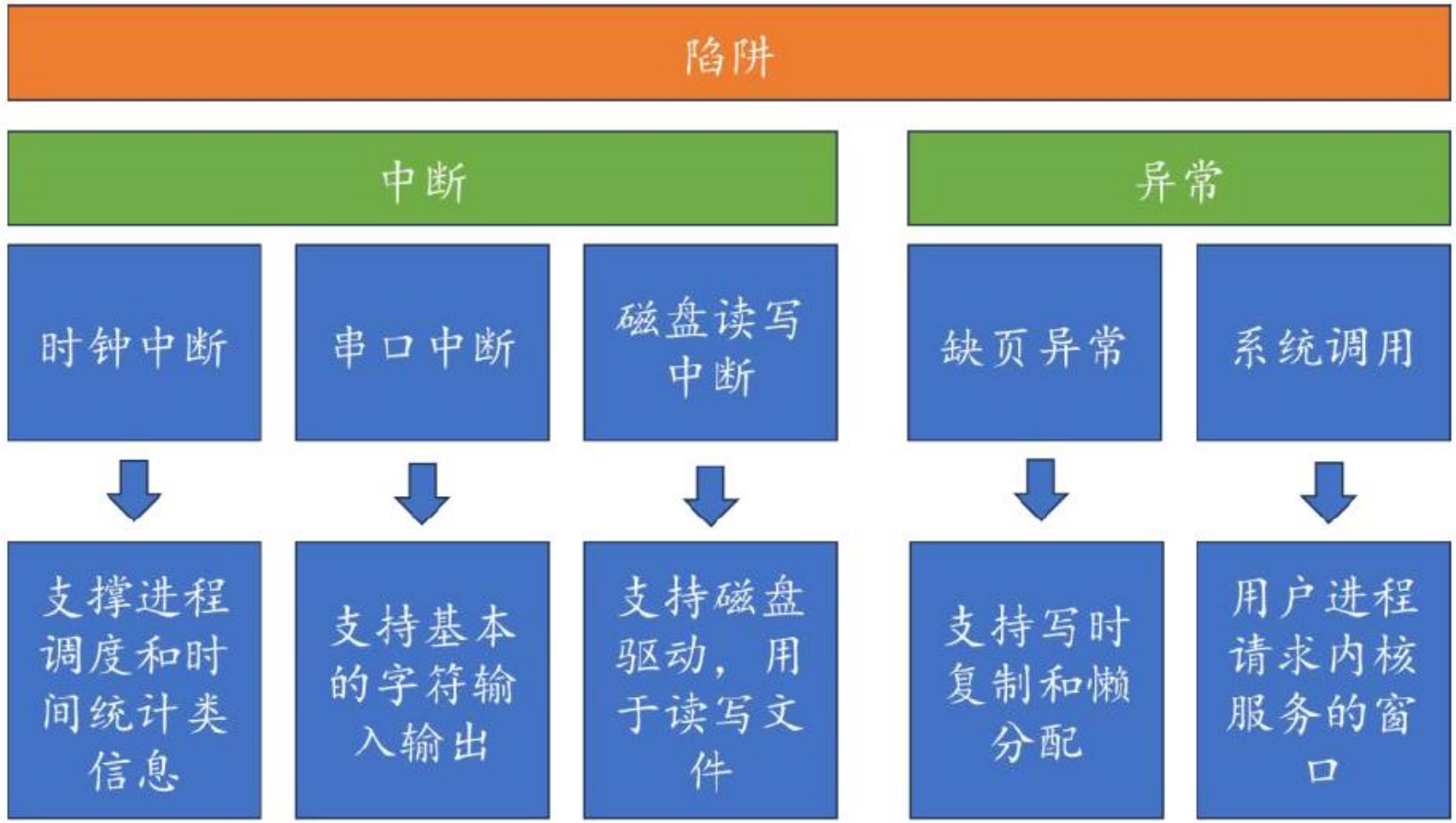
实验三：中断处理与时钟管理



- RISC-V体系结构规定了三种特权模式：用户模式（User mode，U-mode）、监管者模式（Supervisor mode，S-mode）、机器模式（Machine mode，M-mode）。

实验三：中断处理与时钟管理

- 在RISC-V里, 陷阱(trap) 分为 中断(interrupt) 和 异常(exception)



实验三：中断处理与时钟管理

实验任务：支持内核态的中断异常处理，主要是最基本的两种中断：时钟中断 + uart输入中断

- 在RISC_V下，将中断(interrupt)又细分为三种类型：
  - 定时中断(timer Interrupt)：定时中断可以用于产生系统的tick。mip
  - 软件中断（Software Interrupt）：**由软件触发**的中断。通常由一个CPU核心写寄存器来触发另一个CPU核心的中断，用于核间通信（IPI）。msip
  - 中断控制器中断(external Interrupt)：中断控制器则负责所有外设中断。meip

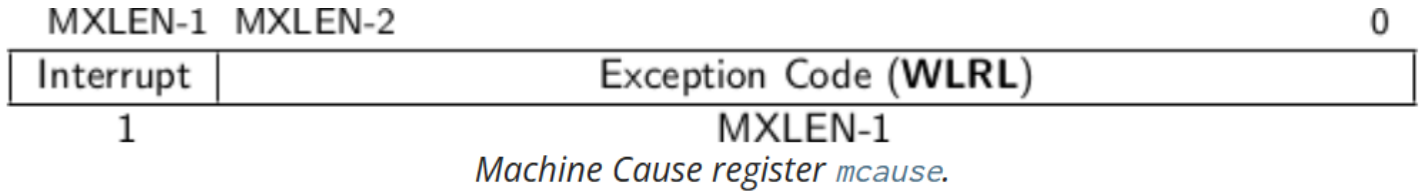
- 在RISC_V中定时器和软件中断是分离出来的，这两个中断被称为CLINT(Core Local Interrupt)，而管理其他外设中断的中断控制器则被称为PLIC(Platform-LevelInterrupt Controller)。每个核都有自己的定时器和产生核间中断的寄存器可以设置，这些寄存器的访问不同于其他的控制状态寄存器，采用的是MMIO映射方式访问.

实验三：中断处理与时钟管理

- M-mode 的寄存器
  - mstatus，mtvec，medeleg，mideleg，mip，mie，mepc，mcause，mtval
- S-mode 的寄存器
- sstatus，stvec，sip，sie，sepc，scause，stval，satp

实验三：中断处理与时钟管理

- mcause



Machine Cause register `mcause`.

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	*Reserved*
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	*Reserved*
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	*Reserved*
1	11	Machine external interrupt
1	≥12	*Reserved*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	≥16	*Reserved*

实验三：中断处理与时钟管理

- mstatus

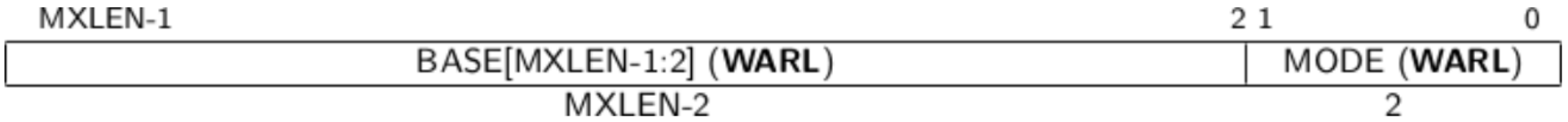
MXLEN-1	MXLEN-2 38	37	36	35 34	33 32	31 23	22	21	20	19	18
SD	**WPRI**	MBE	SBE	SXL[1:0]	UXL[1:0]	**WPRI**	TSR	TW	TVM	MXR	SUM
1	MXLEN-39	1	1	2	2	9	1	1	1	1	1

17	16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
MPRV	XS[1:0]	FS[1:0]	MPP[1:0]	**WPRI**	SPP	MPIE	UBE	SPIE	**WPRI**	MIE	**WPRI**	SIE	**WPRI**
1	2	2	2	2	1	1	1	1	1	1	1	1	1

Machine-mode status register (`mstatus`) for RV64.

实验三：中断处理与时钟管理

- mtvec：记录的是异常处理函数的起始地址



Machine trap-vector base-address register (mtvec).

Value	Name	Description
0	Direct	All exceptions set pc to BASE.
1	Vectored	Asynchronous interrupts set pc to BASE+4×cause.
≥2	—	*Reserved*

实验三：中断处理与时钟管理

默认情况下，各个特权级的陷阱都是在 M-mode被捕捉到，可以通过代码实现将 trap 转发到其它特权级进行处理，为了提高转发的性能在 CPU 级别做了改进并提供两个寄存器：

- medeleg （machine exception delegation）用于指示转发哪些异常到 S-mode；
- mideleg(machine interrupt delegation)用于指示转发哪些中断到 S-mode。

实验三：中断处理与时钟管理

- 中断等待寄存器mip (Machine Interrupt Pending Registers)，可以用于查询中断的等待状态。
- mie 是分别用于保存 pending interrupt enable bits

15 … 12	11	10	9	8	7	6	5	4	3	2	1	0
0	MEIP	0	SEIP	0	MTIP	0	STIP	0	MSIP	0	SSIP	0
4	1	1	1	1	1	1	1	1	1	1	1	1

Standard portion (bits 15:0) of mip.

15 … 12	11	10	9	8	7	6	5	4	3	2	1	0
0	MEIE	0	SEIE	0	MTIE	0	STIE	0	MSIE	0	SSIE	0
4	1	1	1	1	1	1	1	1	1	1	1	1

Standard portion (bits 15:0) of mie.

实验三：中断处理与时钟管理

- mepc：当 trap 陷入到 M-mode 时，mepc 会被 CPU 自动写入引发 trap 的指令的虚拟地址或者是被中断的指令的虚拟地址。
- **mtval：**当 trap 陷入到 M-mode 时，mtval 会被置零或者被写入与异常相关的信息来辅助处理 trap。当触发硬件断点、地址未对齐、access fault、page fault 时，mtval 记录的是引发这些问题的虚拟地址。

实验三：中断处理与时钟管理

- mscratch

  机器模式临时存储器，用于陷阱处理程序的临时存储。它一般用于保存指向机器模式hart本地上下文空间的指针，并在进入M模式陷阱处理程序时与用户寄存器交换。

- msip

  寄存器是一个32位宽的WARL寄存器，高31位被绑定为0，而最低有效位反映在mip 的MSIP位中。

实验三：中断处理与时钟管理

- sstatus

SXLEN-1	SXLEN-2 … 34	33 32	31 … 20	19	18	17	
SD	WPRI	UXL[1:0]	WPRI	MXR	SUM	WPRI	
1	SXLEN-35	2	12	1	1	1	

16 15	14 13	12 … 9	8	7	6	5	4 … 2	1	0
XS[1:0]	FS[1:0]	WPRI	SPP	WPRI	UBE	SPIE	WPRI	SIE	WPRI
2	2	4	1	1	1	1	3	1	1

Supervisor-mode status register (sstatus) for RV64.

- stvec、 sip, sie,sepc, scause, stval 与 m模式 的相应寄存器区别不大。

实验三：中断处理与时钟管理

第一步：时钟中断

- 由于Risc_V的硬件限制，我们必须在M模式下处理定时器中断，但处理程序中仅做简单基础的处理，然后设置sip.SSIP以触发软件中断。也就是说，S模式下的软件中断其实是定时器中断触发的。如此一来，我们即可在S模式下（即系统内核）通过处理软件中断实质性的处理定时器中断。
- 时钟中断的触发，是依靠一个计时器，以及一对寄存器来完成的，即mtime和mtimecmp，
  - mtime 用于反映当前计时器的计数值
  - mtimecmp用于设置计时器的比较值
  - 当mtime 中的计数值，大于或者等于mtimecmp中设置的比较值时，计时器便会产生时钟中断。产生中断后，需要固件/软件，重新写mtimecmp寄存器的值，使其大于mtime 中的值，从而将计时器中断清除。

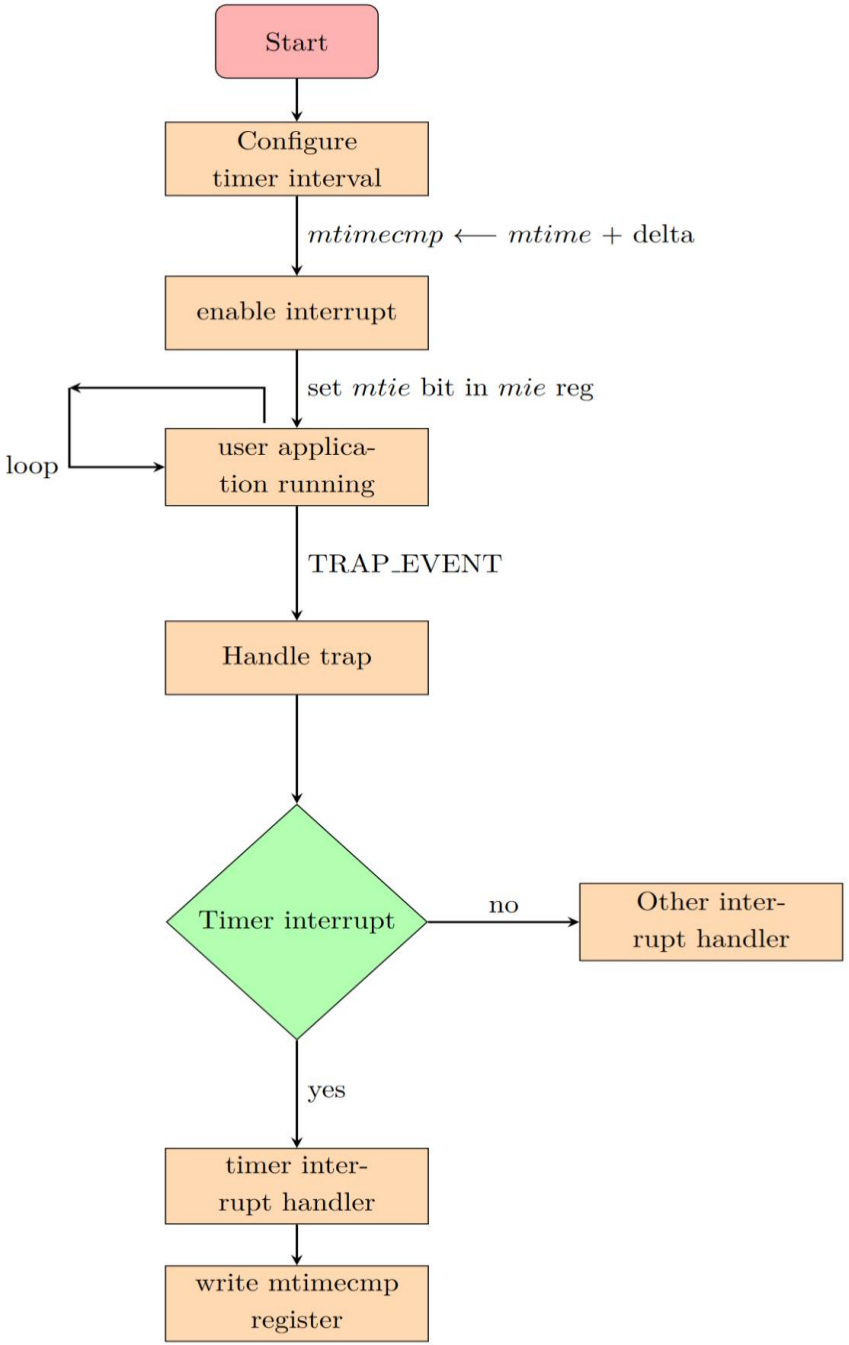
实验三：中断处理与时钟管理

- CLINT中，定义了与核数相同的mtimecmp寄存器。一个mtimecmp 对应一个核，计数值使用同一个mtime寄存器，mtime与所有mtimecmp比较，可以分别对每个核，触发时钟中断。
- 时钟中断触发后，硬件会自动将mip.MTIP=1。

Address	Width	Attr.	Description	Notes
0x0200_0000	4B	RW	msip for hart 0	MSIP Register (1-bit wide)
0x0200_0004			Reserved	
…				
0x0200_3FFF				
0x0200_4000	8B	RW	mtimecmp for hart 0	MTIMECMP Register
0x0200_4008			Reserved	
…				
0x0200_BFF7				
0x0200_BFF8	8B	RW	mtime	Timer Register
0x0200_C000			Reserved	

Table 104: CLINT Memory Map

7.1.3 Timer Interrupt flow chart



实验三：中断处理与时钟管理

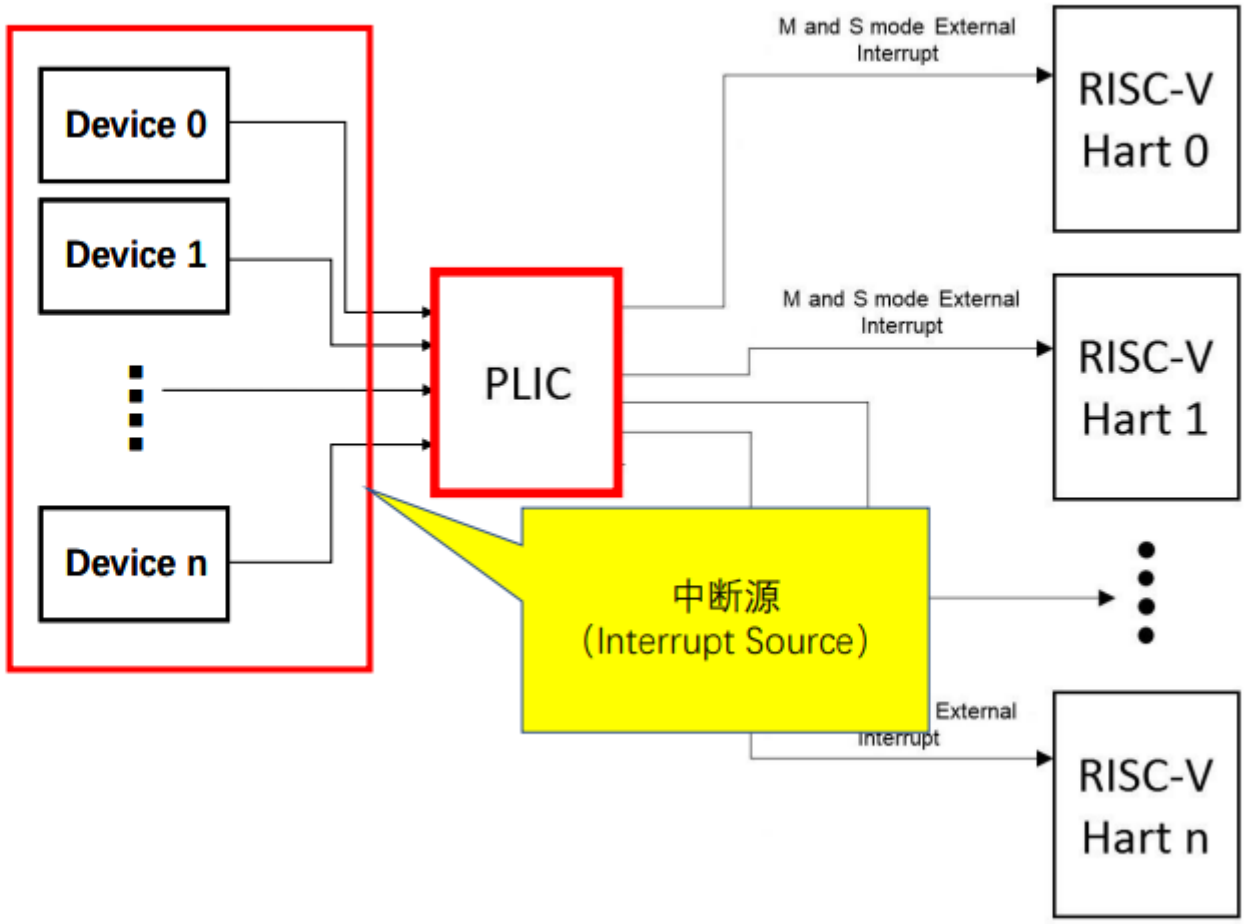
第一步：时钟中断

- xv6关于M模式下的初始化和中断处理，涉及start.c和kernelvec.S 中的timervec。
- 关于S模式下软件中断的处理，涉及main.c、trap.S中的kernelvec、trap.c。其中trap.c中的kerneltrap()和devintr()居于核心调度的地位。

实验三：中断处理与时钟管理

- 外部中断

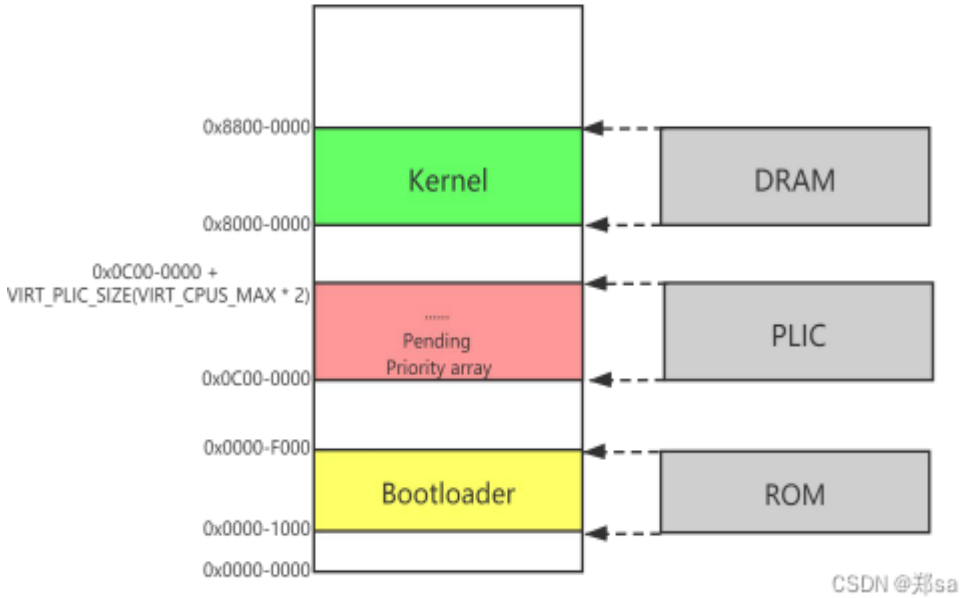
Platform-Level Interrupt Controller



实验三：中断处理与时钟管理

PLIC编程接口-寄存器

- RISC-V 规范规定，PLIC 的寄存器编址采用内存映射（memory map）方式。每个寄存器的宽度为 32-bit。
- 具体寄存器编址采用 base + offset 的格式，且 base 由各个特定 platform 自己定义。针对 QEMU-virt，其 PLIC 的设计参考了 FU540-C000，base 为 0x0c000000。 `#define PLIC_BASE 0x0c000000L`



实验三：中断处理与时钟管理

可编程寄存器	功能描述	内存映射地址
Priority	设置某一路中断源的优先级。	BASE + (interrupt-id) * 4

- 每个 PLIC 中断源对应一个寄存器，用于配置该中断源的优先级。
- QEMU-virt 支持 7 个优先级。 0 表示对该中断源禁用中断。其余优先级，1 最低，7 最高。
- 如果两个中断源优先级相同，则根据中断源的 ID 值进一步区分优先级，ID 值越小的优先级越高。

```
#define PLIC_PRIORITY(id) (PLIC_BASE + (id) * 4)
*(uint32_t*)PLIC_PRIORITY(UART0_IRQ) = 1;
```

实验三：中断处理与时钟管理

可编程寄存器	功能描述	内存映射地址
Pending	用于指示某一路中断源是否发生。	BASE + 0x1000 + ((interrupt-id) / 32) * 4

- 每个 PLIC 包含 2 个 32 位的 Pending 寄存器，每一个 bit 对应一个中断源，如果为 1 表示该中断源上发生了中断（进入 Pending 状态），有待 hart 处理，否则表示该中断源上当前无中断发生。
- Pending 寄存器中断的 Pending 状态可以通过 claim 方式清除。
- 第一个 Pending 寄存器的第 0 位对应不存在的 0 号中断源，其值永远为 0。

实验三：中断处理与时钟管理

可编程寄存器	功能描述	内存映射地址
Enable	针对某个 hart 开启或者关闭某一路中断源。	BASE + 0x2000 + (hart) * 0x80

- 每个 Hart 有 2 个 Enable 寄存器 （Enable1 和 Enable2）用于针对该 Hart 启动或者关闭某路中断源。
- 每个中断源对应 Enable 寄存器的一个 bit，其中 Enable1 负责控制 1 ~ 31 号中断源；Enable2 负责控制 32 ~ 53 号中断源。 将对应的 bit 位设置为 1 表示使能该中断源，否则表示关闭该中断源。

```
#define PLIC_MENABLE(hart) (PLIC_BASE + 0x2000 + (hart) * 0x80)

*(uint32_t*)PLIC_MENABLE(hart)= (1 << UART0_IRQ);
```

实验三：中断处理与时钟管理

可编程寄存器	功能描述	内存映射地址
Threshold	针对某个 hart 设置中断源优先级的阈值。	BASE + 0x200000 + (hart) * 0x1000

- 每个 Hart 有 1 个 Threshold 寄存器用于设置中断优先级的阈值。
- 所有小于或者等于（<=）该阈值的中断源即使发生了也会被 PLIC 丢弃。特别地，当阈值为 0 时允许所有中断源上发生的中断；当阈值为 7 时丢弃所有中断源上发生的中断。

```
#define PLIC_MTHRESHOLD(hart) (PLIC_BASE + 0x200000 + (hart) * 0x1000)

*(uint32_t*)PLIC_MTHRESHOLD(hart) = 0;
```

实验三：中断处理与时钟管理

可编程寄存器	功能描述	内存映射地址
Claim/Complete	详见下描述。	BASE + 0x200004 + (hart) * 0x1000

- Claim 和 Complete 是同一个寄存器，每个 Hart 一个。
- 对该寄存器执行读操作称之为 Claim，即获取当前发生的最高优先级的中断源 ID。Claim 成功后会清除对应的 Pending 位。
- 对该寄存器执行写操作称之为 Complete。所谓 Complete 指的是通知 PLIC 对该路中断的处理已经结束。

```
#define PLIC_MCLAIM(hart) (PLIC_BASE + 0x200004 + (hart) * 0x1000)
#define PLIC_MCOMPLETE(hart) (PLIC_BASE + 0x200004 + (hart) * 0x1000)
```

```
int plic_claim(void)
{
        int hart = r_tp();
        int irq = *(uint32_t*)PLIC_MCLAIM(hart);
        return irq;
}
```

```
void plic_complete(int irq)
{
        int hart = r_tp();
        *(uint32_t*)PLIC_MCOMPLETE(hart) = irq;
}
```

实验三：中断处理与时钟管理

- 第二步： UART输入中断

UART是用于传输、接收系列数据的硬件设备，UART的波特率是38.4KHz，UART在内存的位置起始于 0x10000000，长度为0x100 字节。

UART register to port conversion table

	DLAB = 0		DLAB = 1	
I/O port	Read	Write	Read	Write
base	**RHR** receiver buffer	**THR** transmitter holding	**DLL** divisor latch LSB	
base + 1	**IER** interrupt enable	**IER** interrupt enable	**DLM** divisor latch MSB	
base + 2	**IIR** interrupt identification	**FCR** FIFO control	**IIR** interrupt identification	**FCR** FIFO control
base + 3	**LCR** line control			
base + 4	**MCR** modem control			
base + 5	**LSR** line status	– factory test	**LSR** line status	– factory test
base + 6	**MSR** modem status	– not used	**MSR** modem status	– not used
base + 7	**SCR** scratch			

实验三：中断处理与时钟管理

	A1		REG.		BIT 6		BIT 4		BIT 2		BIT 0
0	0	0	RHR	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	THR	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	1	IER	0	0	0	0	modem status interrupt	receive line status interrupt	transmit holding register interrupt	receive holding register interrupt
0	1	0	FCR	RCVR trigger MSB	RCVR trigger LSB	0	0	DMA mode select	transmit FIFO reset	receiver FIFO reset	FIFO enable
0	1	0	ISR	0/FIFO enabled	0/FIFO enabled	0	0	interrupt prior. bit 2	interrupt prior. bit 1	interrupt prior. bit 0	interrupt status
0	1	1	LCR	divisor latch enable	set break	set parity	even parity	parity enable	stop bits	word length bit 1	word length bit 0
1	0	0	MCR	0	0	0	loop back	OP2	OP1	RTS	DTR
1	0	1	LSR	0/FIFO error	transmit empty	transmit holding empty	break interrupt	framing error	parity error	overrun error	receive data ready
1	1	0	MSR	CD	RI	DSR	CTS	delta CD	delta RI	delta DSR	delta CTS
1	1	1	SPR	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

实验三：中断处理与时钟管理

第二步： UART输入中断

- 外部中断通过PLIC控制器管理。中断处理程序通过读取mcause（或scause、ucause）寄存器，可分辨中断的类型，如果发现是外部中断，则须进一步访问PLIC控制器（plic_claim()）以获取IRQ编号。
- PLIC控制器与外部设备是一对多的关系，多个设备发来中断请求时，PLIC负责仲裁谁被优先处理。PLIC控制器与CPU核心也是一对多的关系，PLIC可能把同一个中断请求发送给多个CPU，当中断处理程序执行plic_claim()，既是为了获取IRQ编号，也是为了认领该中断，晚来的CPU只会获取到0。中断处理结束后，还要再次访问PLIC（plic_complete()），告知本次中断处理结束。

实验三：中断处理与时钟管理

改变部分：

1. main.c 这个文件每次lab基本都会变化
2. start.c

需要写部分：

1. timer.c
2. trap_kernel.c

增加部分：一个子目录trap

timer.h timer.c

plic.h plic.c

trap.h trap.s

实验三：中断处理与时钟管理

第一步：在dev里面增加timer.c

这个文件里的函数被分成了两部分，一部分工作在 **M-mode**，一部分工作在 **S-mode**工作在 **M-mode** 的 timer_init() 要与 `trap.S`里的 timer_vector紧密合作。

时钟中断的原理是：

（1）物理时钟会不停 **di da**，带动当前滴答数寄存器 **MTIME** 不断 +1

（2）还有一个滴答数比较寄存器 **MTIMECMP**，这个寄存器每时每刻都会和 **MTIME** 寄存器做比较；一旦两者相同，就会触发一次时钟中断，程序流跳转到 **mtvec** 寄存器中存储的地址。

实验三：中断处理与时钟管理

（3）可以确定一个滴答数 ****INTERVAL****, 先初始化

****MTIMECMP** = **MTIME** + **INTERVAL****

这样过 ****INTERVAL**** 个滴答后会触发第一次时钟中断。

于是获得了新的时间单元 tick, **1 tick = INTERVAL ×1 di da**

当完成 timer_init() 后, 可以回过头完善 **start.c** 中的 start( )。

之后编写工作在 **S-mode** 的几个函数, 它们只需简单维护系统时钟即可。

实验三：中断处理与时钟管理

第二步, 简单阅读 dev目录下的 plic.c和uart.c的几个函数

第三步, 关注 trap目录下的 trap_kernel.c

首先阅读 **trap.S里面的 **kernel_vector****

相关函数的实现顺序是:

trap_kernel_init()->trap_kernel_inithart() ->trap_kernel_handler()
->timer_interrupt_handler()->external_interrupt_handler()

实验三：中断处理与时钟管理

几个需要注意的地方：

- 目前不处理任何异常，发生了就报错卡死即可。
- xv6 的trap.c写的比较乱，考虑如何重写它的逻辑。
- 建议在trap_kernel_handler()里使用 **trap_id** 进行 switch case 分类处理。
- S-mode的中断默认是关闭的，需要在某个函数里打开(intr_on())。

实验三：中断处理与时钟管理

- 时钟中断默认是在 M-mode 处理，注意它是如何被抛到 S-mode 处理。
- 系统时钟的更新只需要单个CPU去做即可
- external_interrupt_handler()遇到的中断号可能是0，直接忽略即可。
- 发生意料之外的情况时尽量输出有用信息(比如一些寄存器的值，错误原因等)

总而言之，本阶段的关键是理解 kernel_vector与trap_kernel_handler() timer_vector与 timer_init() 如何进行合作，以完成基本的陷阱响应。

实验三：中断处理与时钟管理

- 测试

1. **时钟滴答测试，在合适的地方加一行滴答输出“T”字符。**
2. **时钟快慢测试，在合适的地方加一行ticks输出。**

修改 **INTERVAL**，观察ticks输出速度体会时钟滴答的快慢

3. **UART输入响应测试，验证是否能使用键盘输入字符。**