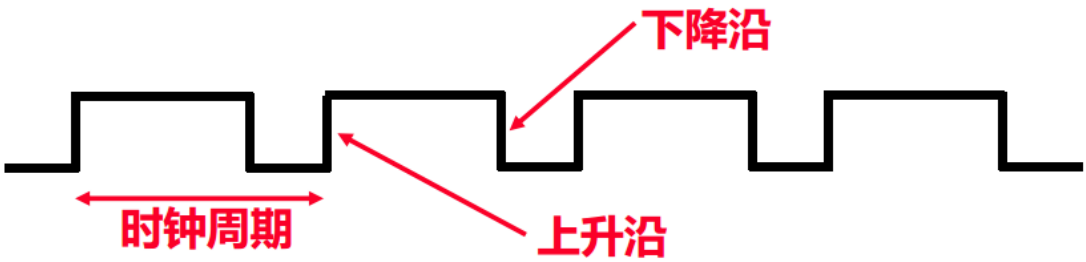
平时作业3

> 姓名：孟启轩
> 学号：2212452
> 专业：计算机科学与技术

问题1：为什么用时钟边缘（下降沿）做控制信号？

状态元件：时序逻辑电路

- 状态（存储）元件的特点：
  - 具有存储功能，在时钟控制下输入被写到电路中，直到下个时钟到达
  - 输入端状态由时钟决定何时被写入，输出端状态随时可以读出
- 定时方式：规定信号何时写入状态元件或何时从状态元件读出
  - 边沿触发（edge-triggered）方式：
    - 状态单元中的值只在时钟边沿改变。每个时钟周期改变一次。
      - 上升沿（rising edge）触发：在时钟正跳变时进行读/写。
      - 下降沿（falling edge）触发：在时钟负跳变时进行读/写。



- 最简单的状态单元（回顾：数字逻辑电路课程内容）：
  - D触发器：一个时钟输入、一个状态输入、一个状态输出

边缘触发（下降沿）能帮助操作系统实现高效、安全且一致的任务调度与中断处理。

为什么用时钟边缘做控制信号

- 精确控制任务切换：边沿触发只在瞬间生效，避免竞态，保证调度准确。
- 可靠保存CPU上下文：寄存器/堆栈在确定瞬间锁存，确保中断和异常处理。安全
- 快速且稳定的中断响应：防止重复或误触发，提升中断处理效率。
- 保障并发一致性：锁存与释放动作精准，保证多核同步和原子性。

为什么选下降沿？

不同模块可以错开操作，提升操作系统调度效率。

问题2：*gmon_start*@plt+16中plt是什么

程序及指令的执行过程

反汇编得到的outputs汇编代码

```
080483e4 : push   %ebp
080483e5 : mov    %esp,%ebp
080483e7 : sub    $0x18,%esp
080483ea : mov    0x8(%ebp),%eax
080483ed: mov     %eax,0x4(%esp)
080483f1 : lea    0xfffffff0(%ebp),%eax
080483f4 : mov    %eax,(%esp)
080483f7 : call   0x8048330 <__gmon_start__@plt+16> what's plt
080483fc : lea    0xfffffff0(%ebp),%eax
080483ff : mov    %eax,0x4(%esp)
08048403: movl    $0x8048500,(%esp)
0804840a: call    0x8048310
0804840f : leave
08048410: ret
```

将strcpy的两个实参入栈

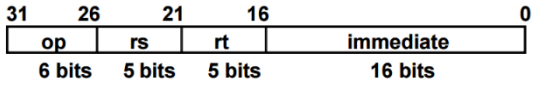
将printf的两个实参入栈

PLT 是过程链接表，用来支持动态链接库函数调用的机制

当程序中调用了动态链接库，比如 `strcpy`、`printf` 这种 libc 里的函数，编译器不会直接在机器码里写死它的真实地址。而是PLT负责第一次调用时跳转到动态链接器，链接器找到真正函数地址，然后后续执行。

问题3：为什么ori指令是0扩展，而load是符号扩展？

RTL: The OR Immediate Instruction



逻辑运算，立即数为逻辑数

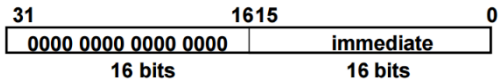
° ori rt, rs, imm16

- M[PC] 取指令（公共操作，取指部件完成）
- R[rt] ← R[rs] or ZeroExt(imm16)

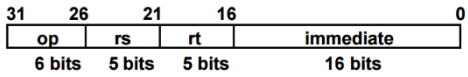
立即数零扩展，并与rs内容做“或”运算

- PC ← PC + 4 计算下地址（公共操作，取指部件完成）

零扩展 ZeroExt(imm16)



RTL: The Load Instruction（装入指令）



° lw rt, rs, imm16

- M[PC] (同加法指令)
- Addr ← R[rs] + SignExt(imm16)

计算数据地址 (立即数要进行符号扩展)

- R[rt] ← M[Addr]

从存储器中取出数据，装入到寄存器中

- PC ← PC + 4 (同加法指令)

与R-type加法指令相比，更复杂！

- ori 指令 —— 零扩展
  - ori 是一种逻辑运算，它处理的是无符号数
  - 逻辑意义：只关心每一位是0还是1，不涉及正负。
  - 16位立即数高位补 0，变成32位。
- lw 指令 —— 符号扩展
  - lw 是装载指令，要用立即数来偏移计算内存地址
  - 有时候偏移是负数，需要支持负偏移。