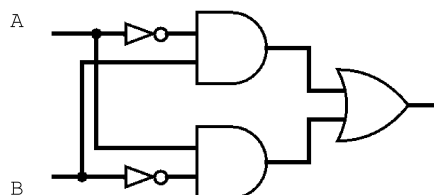


一、体系结构基础

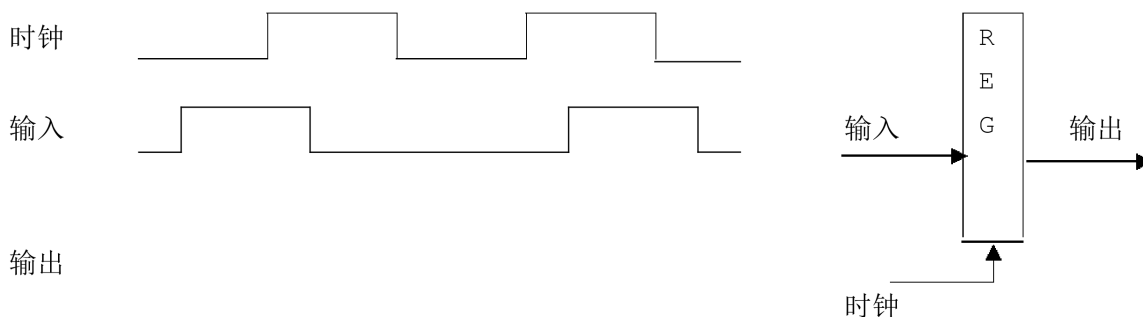
1. 下列描述更符合（早期）RISC 还是 CISC?

- (1) 指令机器码长度固定
- (2) 指令类型多、功能丰富
- (3) 不采用条件码
- (4) 实现同一功能，需要的汇编代码较多
- (5) 译码电路复杂
- (6) 访存模式多样
- (7) 参数、返回地址都使用寄存器进行保存
- (8) x86-64
- (9) MIPS
- (10) 广泛用于嵌入式系统
- (11) 已知某个体系结构使用 `add R1,R2,R3` 来完成加法运算。当要将数据从寄存器 S 移动至寄存器 D 时，使用 `add S,#ZR,D` 进行操作（#ZR 是一个恒为 0 的寄存器），而没有类似于 `mov` 的指令。
- (12) 已知某个体系结构提供了 `xlat` 指令，它以一个固定的寄存器 A 为基地址，以另一个固定的寄存器 B 为偏移量，在 A 对应的数组中取出下标为 B 的项的内容，放回寄存器 A 中。

2. 写出下列电路的表达式



3. 下列寄存器在时钟上升沿锁存数据，画出输出的电平（忽略建立/保持时间）



二、顺序处理器

1. 根据 32 位 Y86 模型完成下表

		call Dest	jXX Dest
Fetch	icode,ifun	icode:ifun $\leftarrow M_1[PC]$	icode:ifun $\leftarrow M_1[PC]$
	rA, rB	\	\
	valC	valC $\leftarrow M_8[PC+1]$	valC $\leftarrow M_8[PC+1]$
	valP	valP $\leftarrow PC+9$	valP $\leftarrow PC+9$
Decode	valA,srcA		
	valB,srcB		
Execute	valE		
	Cond Code		
Memory	valM		
Write back	dstE		
	dstM		
PC	PC		

2. 已知 valC 为指令中的常数值, valM 为访存得到的数据, valP 为 PC 自增得到的值, 完成以下的 PC 更新逻辑:

```
int new_pc = [
    icode == ICALL : ____;
    icode == IJXX && Cnd: ____;
```

```

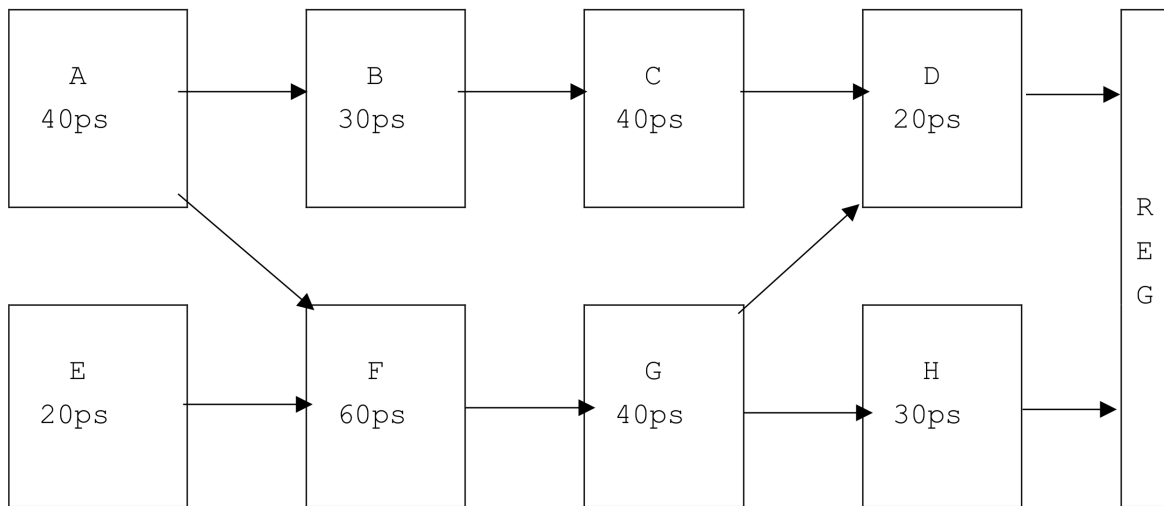
    icode == IRET : ____;
5   1: ____;
]

```

三、流水线的基本原理

1. 判断下列说法的正确性

- (1) () 流水线的深度越深，总吞吐率越大，因此流水线应当越深越好。
 - (2) () 流水线的吞吐率取决于最慢的流水级，因此流水线的划分应当尽量均匀。
 - (3) () 假设寄存器延迟为 20ps，那么总吞吐率不可能达到或超过 50 GIPS。
 - (4) () 数据冒险总是可以只通过转发来解决。
 - (5) () 数据冒险总是可以只通过暂停流水线来解决。
2. 一条三级流水线，包括延迟为 50ps, 100ps, 100ps 的三个流水级，每个寄存器的延迟为 10ps。那么这条流水线的总延迟是 _____ps，吞吐率是 _____GIPS。
3. A~H 为 8 个基本逻辑单元，下图中标出了每个单元的延迟，以及用箭头标出了单元之间的数据依赖关系。寄存器的延迟均为 10ps。



- (1) 计算目前的电路的总延迟
- (2) 通过插入寄存器，可以对这个电路进行流水化改造。现在想将其改造为两级流水线，为了达到尽可能高的吞吐率，问寄存器应插在何处？获得的吞吐率是多少？
- (3) 现在想将其改造为三级流水线，问最优改造所获得的吞吐率是多少？

四、流水线处理器

1. 一个只使用流水线暂停、没有数据前递的 Y86 流水线处理器，为了执行以下的语句，至少需要累计停顿多少个周期？

<pre> irmovl \$1, %eax irmovl \$2, %ebx addl %eax, %ecx addl %ebx, %edx s halt </pre>	<pre> rrmovl %eax, %edx mrmovl (%ecx), %eax addl %edx, %eax halt </pre>	<pre> irmovl \$0x40, %eax mrmovl (%eax), %ebx subl %ebx, %ecx halt </pre>
---	--	--

2. 考虑 Y86 中的 ret 与 jXX 指令。jXX 总是预测分支跳转。

(1) 写出流水线需要处理 ret 的条件 (ret 对应的常量为 IRET):

(2) 写出发现上述条件以后, 流水线寄存器应设置的状态

	Fetch	Decode	Execute	Memory	Writeback
处理 ret					

(3) 写出流水线需要处理 jXX 分支错误的条件 (jXX 对应的常量为 IJXX)

(4) 写出发现上述条件以后, 流水线寄存器应设置的状态

	Fetch	Decode	Execute	Memory	Writeback
处理 ret					

(5) 写出下一条指令地址 f_pc 的控制逻辑

```

int f_pc = [
    M_icode == IJXX && !M_Cnd : _____;
    W_icode == IRET : _____;
    1 : F_predPC;
s ];

```

已知有如下的代码, 其中 valC 为指令中的常数值, valM 为访存得到的数据, valP 为 PC 自增得到的值:

```

int f_predPC = [
    f_icode in { IJXX, ICALL } : f_valC;
    1 : f_valP;
];
s int d_valA = [
    D_icode in { ICALL, IJXX } : D_valP; # Use incremented PC
    # ...省略部分数据前递代码
    1 : d_rvalA; # Use value read from register file
];

```