

ICS Seminar Week5 Prep

侯旭森 许珈铭

2023.10.14

Rules

remainder <- ordinal number in WeChat Group % 4

for all questions do

 if question number % 4 == remainder then

 you should work on it

 end

end

Q1

7、关于如何避免缓冲区溢出带来的程序风险，下述错误的做法为？

- A. 编程时定义大的缓冲区数组
- B. 编程时避免使用 `gets`，而采用 `fgets`
- C. 程序运行时随机化栈的偏移地址
- D. 在硬件级别引入不可执行代码段的机制

答：（ ）

Q2

8. 大多数过程的栈帧是_____的，其长度在_____时确定。（注：此处的编译指从高级语言转化为汇编语言的过程）
- A. 定长，编译
 - B. 定长，汇编
 - C. 可变长，汇编
 - D. 可变长，运行

Q3

8. 下面对指令系统的描述中，错误的是：

- A. CISC 指令系统中的指令数目较多，有些指令的执行周期很长；而 RISC 指令系统中通常指令数目较少，指令的执行周期都较短。
- B. CISC 指令系统中的指令编码长度不固定；RISC 指令系统中的指令编码长度固定，这样使得 CISC 机器可以获得了更短的代码长度。
- C. CISC 指令系统支持多种寻址方式，RISC 指令系统支持的寻址方式较少。
- D. CISC 机器中的寄存器数目较少，函数参数必须通过栈来进行传递；RISC 机器中的寄存器数目较多，只需要通过寄存器来传递参数，避免了不必要的存储访问。

D

Q4

9. 下面关于 RISC 和 CISC 的描述中，正确的是：

- A. CISC 和早期 RISC 在寻址方式上相似，通常只有基址和偏移量寻址
- B. CISC 指令集可以对内存和寄存器操作数进行算术和逻辑运算，而 RISC 只能寄存器操作数进行算术和逻辑运算
- C. CISC 和早期的 RISC 指令集都有条件码，用于条件分支检测
- D. CISC 机器中的寄存器数目较少，函数参数必须通过栈来进行传递；RISC 机器中的寄存器数目较多，只需要通过寄存器来传递参数，避免了不必要的存储访问

Q5

9. 请比较 RISC 和 CISC 的特点，回答下述问题：

假设编译技术处于发展初期，程序员更愿意使用汇编语言编程来解决实际问题，那么程序员会更倾向于选用 _____ ISA。

假设你设计的处理器速度非常快，但存储系统设计使得取指令的速度非常慢（也许只是处理单元的十分之一）。这时你会更倾向于选用 _____ ISA。

A) RISC、RISC B) CISC、CISC C) RISC、CISC D) CISC、RISC

Q6

10. 下面有三组对于指令集的描述，它们分别符合 ①____，②____，③____ 的特点。

- ① 某指令集中，只有两条指令能够访问内存。
- ② 某指令集中，指令的长度都是 4 字节。
- ③ 某指令集中，可以只利用一条指令完成字符串的复制，也可以只利用一条指令查找字符串中第一次出现字母 K 的位置。

- A. CISC, CISC, CISC
- B. RISC, RISC, CISC
- C. RISC, CISC, RISC
- D. CISC, RISC, RISC

Q7

11、关于 RISC 和 CISC 的描述，正确的是：

- A. CISC 指令系统的指令编码可以很短，例如最短的指令可能只有一个字节，因此 CISC 的取指部件设计会比 RISC 更为简单。
- B. CISC 指令系统中的指令数目较多，因此程序代码通常会比较长；而 RISC 指令系统中通常指令数目较少，因此程序代码通常会比较短。
- C. CISC 指令系统支持的寻址方式较多，RISC 指令系统支持的寻址方式较少，因此用 CISC 在程序中实现访存的功能更容易。
- D. CISC 机器中的寄存器数目较少，函数参数必须通过栈来进行传递；RISC 机器中的寄存器数目较多，只需要通过寄存器来传递参数。

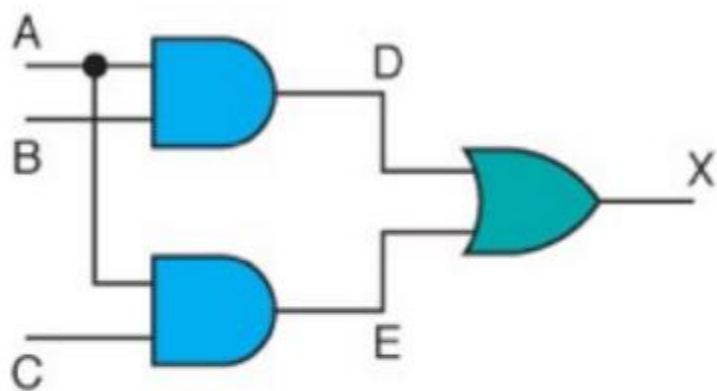
Q8

1. 下列描述更符合（早期）RISC 还是 CISC?

	描述	RISC	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 <code>add R1,R2,R3</code> 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 <code>add S,#ZR,D</code> 进行操作（#ZR 是一个恒为 0 的寄存器），而没有类似于 <code>mov</code> 的指令。		
(12)	已知某个体系结构提供了 <code>xlat</code> 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。		

Q9

9、对应下述组合电路的正确 HCL 表达式为



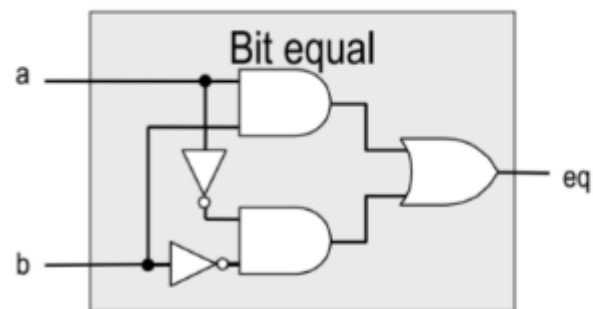
- A. `Bool X = (A || B) && (A || C)`
- B. `Bool X = A || (B && C)`
- C. `Bool X = A && (B || C)`
- D. `Bool X = A || B || C`

答：()

C

Q10

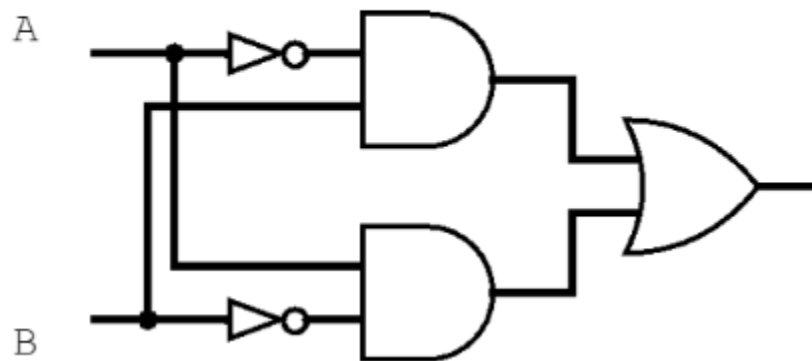
10. 对应下述组合电路的正确 HCL 表达式为:



- A. `Bool eq = (a or b) and (!a or !b)`
- B. `Bool eq = (a and b) or (!a and !b)`
- C. `Bool eq = (a or !b) and (!a or b)`
- D. `Bool eq = (a and !b) or (!a and b)`

Q11

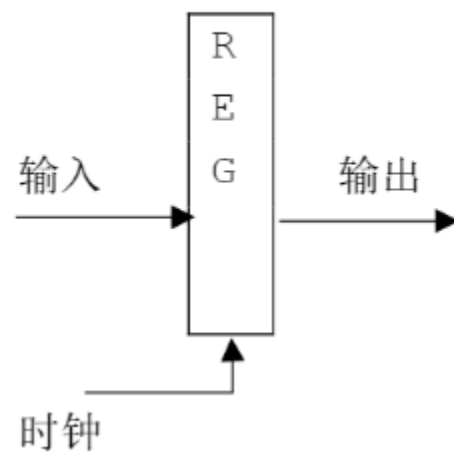
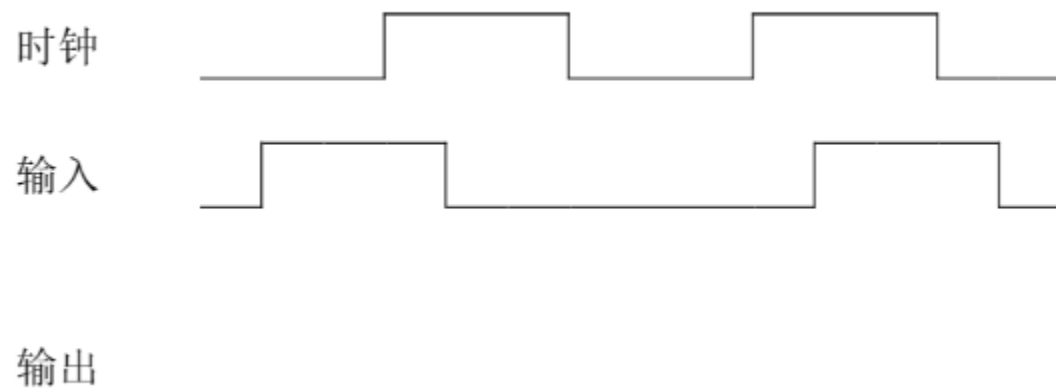
2. 写出下列电路的表达式



$$(!A \ \&\& \ B) \ || \ (A \ \&\& \ !B)$$

Q12

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）



Q13

3. 请查看下文完成如下功能的汇编代码，定位错误语句并进行更正：

给出 n (在 `%ebp+8` 位置, $n \geq 1$), up (在 `%ebp+12` 位置, `ELE*` 类型), 假设以 $*up$ 为头元素 (设 $*up$ 为第 0 个), 由声明中的 `next` 连接形成了一个链表, 请将第 n 个元素 (假设链表足够长) 的 `x` 的值放入 `%eax` 中

X86 代码

```
xorl %ecx,%ecx
movl 8(%edx),%ebp
movl 12(%ebp),%eax
```

```
LOOP:
movl (%eax),%eax
add $1,%ecx
test %ecx,%edx
jne LOOP
```

```
movl (%eax),%eax
```

Y86 代码

```
mrmovl 8(%edx),%ebp
mrmovl 12(%ebp),%eax
irmovl $1,%ecx
```

```
LOOP:
mrmovl (%eax),%eax
test %ecx,%edx
jne LOOP
```

```
mrmovl (%eax),%eax
```

X86代码

```
xorl %ecx,%ecx
movl 8(%edx),%ebp ||应为: movl 8(%ebp),%edx
movl 12(%ebp),%eax
```

LOOP:

```
movl (%eax),%eax
add $1,%ecx
test %ecx,%edx
jne LOOP
```

```
movl (%eax),%eax
```

Y86代码

```
mrmovl 8(%edx),%ebp ||应为: mrmovl 8(%ebp),%edx
mrmovl 12(%ebp),%eax
irmovl $1,%ecx
```

LOOP:

```
mrmovl (%eax),%eax
test %ecx,%edx ||应为: subl %ecx,%edx
jne LOOP
```

```
mrmovl (%eax),%eax
```

Q14

3. 请查看下文完成如下功能的汇编代码，定位错误语句并进行更正：

给出 n (在 `%ebp+8` 位置, $n \geq 1$), up (在 `%ebp+12` 位置, `ELE*` 类型), 假设以 $*up$ 为头元素 (设 $*up$ 为第 0 个), 由声明中的 `next` 连接形成了一个链表, 请将第 n 个元素 (假设链表足够长) 的 `x` 的值放入 `%eax` 中

X86 代码

```
xorl %ecx,%ecx
movl 8(%edx),%ebp
movl 12(%ebp),%eax
```

```
LOOP:
movl (%eax),%eax
add $1,%ecx
test %ecx,%edx
jne LOOP
```

```
movl (%eax),%eax
```

Y86 代码

```
mrmovl 8(%edx),%ebp
mrmovl 12(%ebp),%eax
irmovl $1,%ecx
```

```
LOOP:
mrmovl (%eax),%eax
test %ecx,%edx
jne LOOP
```

```
mrmovl (%eax),%eax
```

X86代码

```
xorl %ecx,%ecx
movl 8(%edx),%ebp || 应为: movl 8(%ebp),%edx
movl 12(%ebp),%eax
```

LOOP:

```
movl (%eax),%eax
add $1,%ecx
test %ecx,%edx
jne LOOP
```

```
movl (%eax),%eax
```

Y86代码

```
mrmovl 8(%edx),%ebp || 应为: mrmovl 8(%ebp),%edx
mrmovl 12(%ebp),%eax
irmovl $1,%ecx
```

LOOP:

```
mrmovl (%eax),%eax
test %ecx,%edx || 应为: subl %ecx,%edx
jne LOOP
```

```
mrmovl (%eax),%eax
```


Q15

第三题 (20 分)

(1) 观察下面C语言函数和它相应的X86-64汇编代码

```
int foo(int x, int i)
{
    switch(i)
    {
        case 1:
            x -= 10;
        case 2:
            x *= 8;
            break;
        case 3:
            x += 5;
        case 5:
            x /= 2;
            break;
        case 0:
            x &= 1;
        default:
            x += i;
    }
    return x;
}
```

```
00000000004004a8 <foo>:
4004a8: mov %edi,%edx
4004aa: cmp $0x5,%esi
4004ad: ja 4004d4 <foo+0x2c>
4004af: mov %esi,%eax
4004b1: jmpq *0x400690(,%rax,8)
4004b8: sub $0xa,%edx
4004bb: shl $0x3,%edx
4004be: jmp 4004d6 <foo+0x2e>
4004c0: add $0x5,%edx
4004c3: mov %edx,%eax
4004c5: shr $0x1f,%eax
4004c8: lea (%rdx,%rax,1),%eax
4004cb: mov %eax,%edx
4004cd: sar %edx
4004cf: jmp 4004d6 <foo+0x2e>
4004d1: and $0x1,%edx
4004d4: add %esi,%edx
4004d6: mov %edx,%eax
4004d8: retq
```

0x400690:	0x00000000004004d1	0x00000000004004b8
0x4006a0:	0x00000000004004bb	0x00000000004004c0
0x4006b0:	0x00000000004004d4	0x00000000004004c3

调用gdb命令x/kg \$rsp 将会检查从rsp中的地址开始的k个8字节字，请填写下面gdb命令的输出（每空一分）。

>(gdb) x/6g 0x400690

0x400690:	0x_____	0x_____
0x4006a0:	0x_____	0x_____
0x4006b0:	0x_____	0x_____

Q16

第三题 (20 分)

(1) 观察下面C语言函数和它相应的X86-64汇编代码

```
int foo(int x, int i)
{
    switch(i)
    {
        case 1:
            x -= 10;
        case 2:
            x *= 8;
            break;
        case 3:
            x += 5;
        case 5:
            x /= 2;
            break;
        case 0:
            x &= 1;
        default:
            x += i;
    }
    return x;
}
```

```
00000000004004a8 <foo>:
4004a8: mov %edi,%edx
4004aa: cmp $0x5,%esi
4004ad: ja 4004d4 <foo+0x2c>
4004af: mov %esi,%eax
4004b1: jmpq *0x400690(,%rax,8)
4004b8: sub $0xa,%edx
4004bb: shl $0x3,%edx
4004be: jmp 4004d6 <foo+0x2e>
4004c0: add $0x5,%edx
4004c3: mov %edx,%eax
4004c5: shr $0x1f,%eax
4004c8: lea (%rdx,%rax,1),%eax
4004cb: mov %eax,%edx
4004cd: sar %edx
4004cf: jmp 4004d6 <foo+0x2e>
4004d1: and $0x1,%edx
4004d4: add %esi,%edx
4004d6: mov %edx,%eax
4004d8: retq
```

0x400690:	0x00000000004004d1	0x00000000004004b8
0x4006a0:	0x00000000004004bb	0x00000000004004c0
0x4006b0:	0x00000000004004d4	0x00000000004004c3

调用gdb命令x/kg \$rsp 将会检查从rsp中的地址开始的k个8字节字，请填写下面gdb命令的输出（每空一分）。

>(gdb) x/6g 0x400690

0x400690:	0x_____	0x_____
0x4006a0:	0x_____	0x_____
0x4006b0:	0x_____	0x_____