Part1 信息的表示和处理+程序优化

（总分 54 分，附加分 6 分）

一.选择题（每题 1 分，共 15 分）

1.A 2.B 3.A 4.D 5.A

6.A 7.C 8.A 9.D 10.D

11.C 12.B 13.B 14.C 15.A

二.填空题（每题 1 分，共 5 分）

1. 1111111111111110

2. n&0x40 或 n&0x80 或(n>>6)&1 或(n>>7)&1

3. 0X00 0X00 0X80 0XC4

4. 64

5. 0XF9 0XFF 0XFF 0XFF

三.判断题（每题 2 分，共 14 分）

1.X 2. √ 3. √ 4.X 5. √ 6.X 7.X

四.分析题（每题 10 分，共 20 分，注意：最后一题如果有代码测试再附加 6 分）

1. 解答：

(1)首先进行十进制到二进制的转化：

-1010.0001100110011[0011]…

(2)表示为科学记数法为：

-1.01000011001100110011001100 x 2^3

(3)指数部分的+127 移码为

3+127=130，其二进制形式为 10000010

(4)尾数部分 23 位的编码为 0100 0011 0011 0011 0011 001 1001100

向上舍入 0100 0011 0011 0011 0011 010

(5) IEEE 754 编码为：符号位 1 （16 进制）

1 10000010 0100 0011 0011 0011 0011 010 = C1 21 99 9A

(6) 内存中倒序：小端模式 0X9A 0X99 0X21 0XC1

（注意：浮点数计算中需要舍入时，是向偶舍入）

2. 解答：

```
void vector_sum(vec *v, float *sum)
{
    long int i;
    *sum = 0;
    int length = vec_length(v);
    float *d = v->data;
    float acc = 0;//局部变量累计结果
    for (i = 0; i < length; i++)
        acc = acc + d[i];//消除不必要的内存引用
```

```
    *sum = acc;
}
```

优化方法：

（1）代码移动，将 vec_length()移到循环外。

（2）消除不必要的函数调用，将 get_vec_element()转换为通过 d[i]来获取每次循环的元素。

（3）用局部变量 acc 做累加，消除不必要的内存引用

（4）可做循环展开

测试优化性能：可以通过多次调用 vector_sum 函数放大函数执行时间进行比较，也可以用一些程序性能测试工具进行测试（有测试会额外+6 分）。

Part2 X64 汇编

（总分 34 分）

选择题（**每题 1 分，共 6 分**）

1.D　2. C　3. D　4. B　5. C　6. D

填空题（**第 7 题~12 题，每空 1 分，共 18 分**）

7. 40

8. 低　%rsp

9. 跳转到中间　guarded-do　跳转到中间

10. 0x100，0xff，0x11，0xff，0x11

11. （1）result = x>=y? x:y;（2）cmpq %rsi, %rdi　cmovl %rsi，%rax

注意：本题 2 分，只答对第二问的两个汇编语句即可得 2 分

12. ①movsbl （%rdi），%eax　movl %eax，（%rsi）

②movsbl （%rdi），%eax　movl %eax，（%rsi）

③movzbl （%rdi），%eax　movq %rax，（%rsi）另解：movzbq (%rdi), %rax　movq %rax, (%rsi)

④movl （%rdi），%eax　movb %al，（%rsi）

⑤movl （%rdi），%eax　movb %al，（%rsi）

13. （共 10 分）答案：

攻击原理（3 个采分点，6 分）：

1）程序输入缓冲区写入特定的数据（如攻击代码），例如在 gets 读入字符串时

2）使位于栈中的缓冲区数据溢出，用特定的内容覆盖栈中的内容，例如函数返回地址等

3）使得程序在读入字符串，结束函数 gets 从栈中读取返回地址时，错误地返回到特定的位置，执行特定的代码，达到攻击的目的。

防范方法(答对 2 个方法即可得 4 分)：

1）代码中避免溢出漏洞：例如使用限制字符串长度的库函数。

2）随机栈偏移：程序启动后，在栈中分配随机数量的空间，将移动整个程序使用的栈空间地址。

3）限制可执行代码的区域

4）进行栈破坏检查——金丝雀

Part3 Y86 处理器体系结构

（总分 75 分）

选择题+判断（**每空 1 分，共 5 分**）

1. A　2. B　3. A　4. 错（X）　5. 错（X）

6. 填表**（每空 1 分，共 5 分）**

原因	名称	指令频率	条件频率	气泡数	总处罚	CPI
加载使用	lp	0.20	0.15	1	0.32	1.32
预测错误	mp	0.25	0.40	2		
返回	rp	0.03	1.00	3		

7. **（1 分，等价的答案都给分）**0x00b/0xb

8. **（5 分）**取指、译码、执行、访存、写回。（注意：流水线结构中没有更新 PC）

9. （8 分，其中 3 种冒险共 3 分，解决办法 5 个采分点共 5 分）

(1)数据冒险：指令使用寄存器 R 为目的，瞬时之后使用 R 寄存器为源。

解决方法有：

①暂停：通过在执行阶段插入气泡（bubble/nop），使得当前指令执行暂停在译码阶段；

②数据转发（前递）：增加 valM/valE 的旁路路径，直接送到译码阶段；

(2) 加载使用冒险：

解决方法有：指令暂停在取指和译码阶段，在执行阶段插入气泡（bubble/nop）；

(3) 控制冒险：

解决办法：

①分支预测错误：在条件为真的地址 target 处的两条指令分别插入 1 个 bubble。

②ret：在 ret 后插入 3 个 bubble。

10. （12 分，每空 1 分）

	寄存器 1	寄存器 2	寄存器 3
239	I1	None	None
241	I2	I1	None
300	I2	I1	None
361	I3	I2	I1

11. （19 分，每个采分点 1 分）

指令	push rA	pop rA
取指	icode:ifun <- M_1[PC]	icode:ifun ← M_1[PC]
	rA:rB <- M_1[PC+1]	rA:rB ← M_1[PC+1]
	valP <- PC+2	valP ← PC+2

译码	valA <- R[rA]	valA ← R[%rsp]
	valB <- R[%rsp]	valB ← R[%rsp]
执行	valE <- valB+(-8)	valE ← valB+8
访存	M_8[valE] <- valA	valM← M_8[valA]
写回	R[%rsp] <- valE	R[%rsp] ← valE R[rA] ← valM
更新 PC	PC <- valP	PC ← valP

12. (共 10 分，前三小题各 2 分，最后小题 1+3=4 分)

(1).只插入一个寄存器，得到一个两阶段的流水线。要使吞吐量最大化，该在哪里插入寄存器呢?吞吐量和延迟是多少?

12-1 解析：对一个两阶段流水线来说，最好的划分是块 A、B 和 C 在第一阶段，块 D、E 和 F 在第二阶段。第一阶段的延迟为 170ps,所以时钟周期是 170+20= 190ps. 因此最大吞吐量为 5.26GIPS，而延迟为 380ps。

(2).要使一个三阶段的流水线的吞吐量最大化，该将两个寄存器插在哪里呢?吞吐量和延迟是多少?

12-2 解析：对一个三阶段流水线来说，应该使块 A 和 B 在第一阶段，块 C 和 D 在第二阶段，而块 E 和 F 在第三阶段。前两个阶段的延迟均为 110ps,所以时钟周期为 130ps,而吞吐量为 7.69GIPS，延迟是 390ps。

(3).要使一个四阶段的流水线的吞吐量最大化，该将三个寄存器插在哪里呢?吞吐量和延迟是多少?

12-3 解析：对于一个四阶段流水线来说，块 A 为第一阶段， 块 B 和 C 在第二阶段，块 D 是第三阶段，而块 E 和 F 在第四阶段。第二阶段需要 90ps,所以时钟周期是 110ps，而吞吐量为 9.09 GIPS，延迟是 440ps。

(4).要得到一个吞吐量最大的设计，至少要有几个阶段?描述这个设计及其吞吐量和延迟。

12-4 解析：最优的设计应该是五阶段流水线，除了 E 和 F 处于第五阶段以外，其他每个块是一个阶段，周期时长为 80+ 20 = 100ps，吐量为大约 10.00 GIPS，延迟为 500ps。变成更多的阶段也不会有帮助了，因为不可能使流水线运行得比以 100ps 为一周期还要快了。

13. (共 10 分，按照解释的正确性给分，参考答案中的表和图不是必须的)

解析：

指令	irmovq Stack,%rsp	call p
取指	icode : ifun <- M1[PC]	icode:ifun <- M1[PC]
	rA:rB <- M1[PC+1]	
	valC <- M8[PC+2]	valC <- M8[PC+1]
	valP <- PC+10	valP <- PC+9
译码		
		valB <- R[%rsp]
执行	valE <- 0 + valC	valE <- valB+(-8)

访存		M8[valE] <- valP
写回	R[rB] <- valE (注意这里 rB 就是%rsp)	R[%rsp] <- valE
更新 PC	PC <- valP	PC <- valC

由上图可知数据相关产生的原因：irmovq Stack,%rsp 指令在写回阶段才修改%rsp，call p 指令要在译码阶段读取%rsp，发生了数据相关，需要让 call p 的译码阶段发生在 irmovq Stack,%rsp 的写回阶段之后。因此流水线中插入 3 个 nop 如下：