作业发布时间：2023/05/08 星期一
本次作业要求如下：
1.截止日期：2023/05/21 周日晚 24:00
2.后面发布提交作业网址
3.**命名格式**：附件和邮件命名统一为“第一次作业+学号+姓名”，**作业为 PDF 格式**；
4.注意：
(1) 选择、判断和填空只需要写答案，大题要求有详细过程，**过程算分**。
(2) 答案请用另一种颜色的笔回答，便于批改，**否则视为无效答案。**
(3) 大题的过程最好在纸上写了拍照，放到 word 里。
(4) 本次作业由 Part1 和 Part2 两部分，需要全部作答
5.本次作业遇到问题请联系课程群里的助教。

Part1 X86 汇编

1.C 语言程序中的整数常量、整数常量表达式是在（ D ）阶段初始化和计算的。
（A）预处理 （B）执行 （C）连接 （D）编译
2.关于 Intel 的现代 X86-64 CPU 正确的是（ C ）
A. 属于 RISC B. 属于 MISC C. 属于 CISC D. 属于 NISC
3.下列叙述正确的是（ D ）
A.X86-64 指令"mov $-1,%eax"会使%rax 的值变成 0xffffffffffffffff
B.在一条指令执行期间，CPU 不会两次访问内存
C.CPU 不总是执行 CS::RIP 所指向的指令，例如遇到 call、ret 指令时
D.一条 mov 指令不可以使用两个内存地址操作数
4.在 x86-64 系统中，调用函数 int gt (long x, long y)时，保存参数 y 的寄存器是(B)
A. %rdi B.%rsi C.%rax D.%rdx
5.在 x86-64 系统中，调用函数 long gt (long x, long y)时，保存返回值的寄存器是(C)
A. %rdi B.%rsi C.%rax D.%rbx
6.下列传送指令中，哪一条是正确的（ D ）
A.movb $0x105,(%ebx)
B.movl %rdi,%rax
C.movq %rdi,$105
D.movl 4(%rsp),%eax
7.C 语言程序定义了结构体 struct noname{char c; long n; int k; float *p; short a;};若该程序编译成 64 位可执行程序，则 sizeof(noname)的值是 24 。
8.在 X86-64 中，程序的栈存放在内存中，栈顶元素的地址使所有栈中元素中最 小 的，栈指针寄存器 %rsp 保存着栈顶元素的地址。
9.While 循环的两种展开方法分别是什么：（跳到中间法 ）(guarded-do 法)
下面代码是哪种？（跳到中间法 ）

```
long fact_while_jm_goto(long n)
{
    long result = 1;
    goto test;
 loop:
    result *= n;
    n = n-1;
 test:
    if (n > 1)
        goto loop;
    return result;
}
```

10. 已知内存和寄存器中的数值情况如下：

内存地址	值
0x100	0xff
0x104	0xAB
0x108	0x13
0x10c	0x11

寄存器	值
%rax	0x100
%rcx	0x1
%rdx	0x3

请填写下表，给出对应操作数的值：

操作数	值
%rax	0x100
(%rax)	0xff
9(%rax,%rdx)	0x11
0xfc(,%rcx,4)	0xff
(%rax,%rdx,4)	0x11

11. 有下列 C 函数:

```
long max(long x, long y)
{
    long result;
    if(x >= y){
        result = x;
    }else{
        result = y;
    }
    return result;
}
```

请写出红色部分代码使用条件数据传输来实现条件分支的等价形式，并给出对应的条件传送指令(初始执行指令 movq %rdi, %rax) 。

条件数据传输：result = (x>=y)? x : y

movq %rdi, %rax

cmpq %rsi, %rdi

cmovl %rsi, %rax

12.阅读的 sum 函数反汇编结果，解释 1-5 每行指令的 功能和作用

```
4004e7 <sum>:
push %rbp #①
mov %rsp,%rbp #②
movl %rdi,-0x4(%rbp) #③
jmp 400512 <sum+0x2b>
4004f4：
mov -0x4(%rbp),%eax
cltq
mov 0x601030(,%rax,4),%edx
mov 0x200b3e(%rip),%eax #601044 <val>
add %edx,%eax
mov %eax,0x200b36(%rip) #601044 <val>
addl $0x1,-0x4(%rbp)
400512：
cmpl $0x3,-0x4(%rbp)#④
jl 4004f4 <sum+0xd>#⑤
mov 0x200b26(%rip),%eax # 601044 <val>
pop %rbp
Retq
```

① 入栈指令，将 rbp 入栈
② 传送指令，将栈顶指针 rsp 的值传送给 rbp
③ 传送指令，向%rbp-4 的内存位置传送数值%rdi
④ 比较指令：%rbp-4 的内存数值与 3 进行比较
⑤ 条件跳转指令，小于等于则跳转（跳转到 4004f4 处）

13.假设变量 sp 和 dp 被声明为类型：

```
Src_t *sp;
Dest_t *dp;
```

这里的 Src_t 和 Dest_t 是用 typedef 声明的数据类型，我们想使用适当的数据传送指令来实现下面的操作：

```
*dp = (Dest_t) *sp;
```

sp 和 dp 的值分别存储在%rdi 和%rsi 中，对下表的每个表项，请写出合适的两条传送指令（如需用到其他寄存器，使用%rax）。注意：规定如果强制类型转换既涉及大小变化又涉及符号变化时，操作应先改变大小。

Src_t	Dest_t	指令
char	int	① movsbl %rdi, %rax Mov %rax, %rsi
char	unsigned	② movsbl %rdi, %rax Mov %rax, %rsi
unsigned char	long	③ movzbl %rdi, %rax Mov %rax, %rsi
int	char	④ movl %rdi, %rax Movb %al, %rsi
unsigned	unsigned char	⑤ movl %rdi, %rax Movb %al, %rsi

14.设有 C 代码如下：

```
typedef struct {
  int a[2];
  double d;
} struct_t;
double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

其运行时的栈结构如下图所示：

临界状态	6
?4 字节	5
?4 字节	4
d7 ... d4	3
d3 ... d0	2
a[1]	1

a[0]	0

请分别求出：调用 fun（x）的结果，其中 x 分别为 0，1，2，3，4，6 并给出分析过程。

x=0 -> 3.14. 此时 x 在 a 的 index range 内，正常返回。
x=1 -> 3.14. 此时 x 在 a 的 index range 内，正常返回。
X=2 -> 约 3.14. 此时会覆盖 d 的低 32 位，影响尾数。
x=3 -> 约 2. 此时会覆盖 d 的高 32 位，使得浮点数大约为 2.
X=4, 6 -> 报段错误。此时覆盖了 ebp。

15.简述缓冲区溢出攻击的原理以及防范方法（2 种）

向程序输入缓冲区写入特定的数据，用特定的内容覆盖栈中的内容，例如函数返回地址等，使得程序在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：限制字符串操作的长度可编码缓冲区溢出的攻击、栈空间地址的随机偏移、或在栈中某个位置放入特定的金丝雀值。

Part2 处理器体系结构

1.Y86-64 的指令 ret 编码长度为（ A ）。
A.1 字节 B.2 字节 C.9 字节 D.10 字节

2. Y86-64 的 CPU 顺序结构设计与实现中，分成（ B）个阶段
A.5 B.6 C.7 D.8

3. Y86-64 的 CPU 流水线结构设计与实现中，分成（ A ）个阶段
A.5 B.6 C.7 D.8

判断题：
4.Y86-64 的顺序结构实现中，寄存器文件读时是作为时序逻辑器件看待 （ X ）

5.现代超标量 CPU 指令的平均周期接近于 1 个但大于 1 个时钟周期（ X ）

6. 下表是 CPU 的某个场景，解释：加载指令（mrmovq 和 popq）占所有执行指令的 20%，其中 15%会导致 加载/使用 冒险。条件分支指令占所有执行指令的 25%，其中 40%不选择分支。返回指令占所有执行指令的 3%。完成下表：

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. 在 Y86-64 架构的机器上，有下列汇编代码，请指出 jne t 指令之后执行的那条指令的地址为_____0x00b_____（顺序执行，不考虑流水线）。

```
0x000:      xorq %rax,%rax
0x002:      jne  t
…
0x019: t:  irmovq $3, %rdx
0x023:      irmovq $4, %rcx
0x02d:      irmovq $5, %rdx
```

8. 请写出 Y86-64 的 CPU 流水线结构设计与实现中各流水线阶段的名称（注意顺序不要错位）。

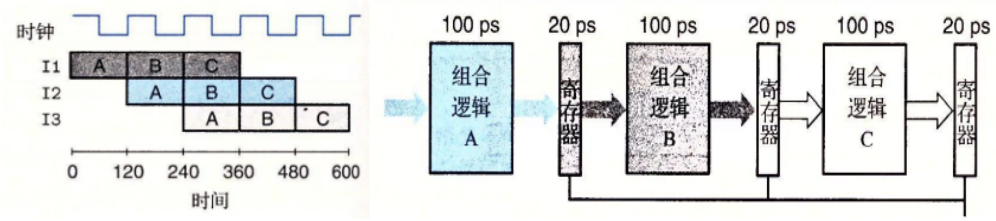
六个阶段：取指、译码、执行、访存、写回、PC 更新。

9. Y86-64 流水线 CPU 中的冒险的种类与处理方法。

数据冒险：
- 暂停：通过在执行阶段插入气泡，使得当前指令执行暂停在译码阶段；
- 数据转发：增加旁路路径，直接送到译码阶段；

控制冒险：
- 在条件为真的地址 target 处的两条指令分别插入一个气泡；
- ret：ret 后插入 3 个气泡；

10. 假设有一个理想的三阶段流水线，执行指令为 I1（Instruction1），I2，I3，其时序图与电路示意图如图所示：（P321）



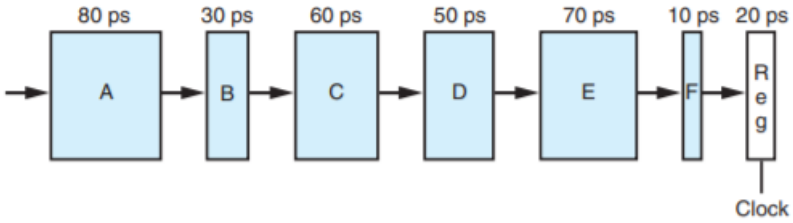
请分析不同时间点各寄存器中保存值所属指令，并完成下表（填入 I1，I2，I3，None。寄存器 1 表示左数第一个寄存器）

	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. 请根据描述在 Y86-64 顺序实现的条件下完成下面表格。
(1)请写出 Y86-64 顺序实现的 push rA 指令在各阶段的微操作。
(2)请写出 Y86-64 顺序实现的 pop rA 指令在各阶段的微操作。

指令	push rA	pop rA
取指	icode:ifun=M1[PC]	icode:ifun=M1[PC]
	rA:rB=M1[PC+1]	rA:rB=M1[PC+1]
	valP=PC+2	valP=PC+2
译码	valA=R[rA]	valA=R[rA]
	valB=R[%rsp]	valB=R[%rsp]
执行	valE=valB-8	valE=valB+8
访存	M8[valE]=valA	MvalM=M8[valA]
写回	R[%rsp]=valE	R[%rsp]=valE
更新 PC	PC=valP	PC=valP

12. 假设我们分析图中的组合逻辑，认为它可以分为 6 个块，以此命名为 A、B、C、D、E、F，延迟分别为 80，30，60，50，70，10（单位 ps），如下图所示。



在这些块之间插入流水线寄存器，就得到这一设计的流水线化的版本。根据在哪里插入流水线寄存器，会出现不同的流水线深度(有多少个阶段)和最大吞吐量的组合。假设每个流水线寄存器的延迟为 20ps。
请根据以上描述，回答下列问题：
(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?

考虑 C、D 之间的两级流水线。前段的总延迟为 170ps，后段总延迟为 130ps。
则插入流水线寄存器之后，最长延迟为 190ps，即时钟周期。
吞吐量为 1/190ps = 5.26Gops，指令的执行时间为 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?

考虑 BC、DE 之间的三级流水线。第一个流水段的延迟为 110ps，第二个为 110ps，第三个为 80ps。
则在 BC、DE 插入流水寄存器之后，最长延迟为 130ps，即时钟周期。
吞吐量为 1/130ps = 7.69Gops，总的执行时间为 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?

考虑 AB、CD、DE 之间的四级流水线。第一个流水段的延迟为 80ps，第二个为 90ps，第三个为 50ps，第四个为 80ps。
则最长的延迟为 110ps，即时钟周期。
吞吐量为 1/110ps = 9.09Gops，总的执行时间为 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。

注意到，要使吞吐量最大，则最长的组合逻辑延时要最小。观察可以发现，最小为 80ps，即 A。为了不超过 80，在 AB、BC、CD、DE 之间都插入流水寄存器。
因此至少有五个阶段，在 AB、BC、CD、DE 之间都插入流水寄存器。
总的延迟为 100ps，即时钟周期。
吞吐量为 1/100ps = 10Gops。总的执行时间为 500ps。

13.下面是一个程序段，请根据 Y86-64 的微指令和流水线数据相关的知识，试解释为什么在 call 指令之前要插入 3 个 nop 指令。

```
0x000:     irmovq Stack,%rsp     # Intialize stack pointer
0x00a:     nop
0x00b:     nop
0x00c:     nop
0x00d:     call p                # Procedure call
0x016:     irmovq $5,%rsi        # Return point
0x020:     halt
```

0x000 处写入%rsp，在 0x00d 处，call p 需要使用%rsp。因此为了防止数据冒险，需要让%rsp 进入写回阶段，于是需要插入三个 nop 指令。