北京大学信息科学技术学院考试试卷

考试科目：计算机系统导论 **姓名：**_____ **学号：**_____

考试时间：2024 年 11 月 4 日 **小班号：**_____

题号	一	二	三	四	五	**总分**
分数						
阅卷人						

装订线内

不要答题

北京大学考场纪律

1、考生进入考场后，按照监考老师安排隔位就座，将学生证放在桌面上。无学生证者不能参加考试；迟到超过 15 分钟不得入场。在考试开始 30 分钟后方可交卷出场。

2、除必要的文具和主考教师允许的工具书、参考书、计算器以外，其它所有物品（包括空白纸张、手机、或有存储、编程、查询功能的电子用品等）不得带入座位，已经带入考场的必须放在监考人员指定的位置。

3、考试使用的试题、答卷、草稿纸由监考人员统一发放，考试结束时收回，一律不准带出考场。若有试题印制问题请向监考教师提出，不得向其他考生询问。提前答完试卷，应举手示意请监考人员收卷后方可离开；交卷后不得在考场内逗留或在附近高声交谈。未交卷擅自离开考场，不得重新进入考场答卷。考试结束时间到，考生立即停止答卷，在座位上等待监考人员收卷清点后，方可离场。

4、考生要严格遵守考场规则，在规定时间内独立完成答卷。不准交头接耳，不准偷看、夹带、抄袭或者有意让他人抄袭答题内容，不准接传答案或者试卷等。凡有违纪作弊者，一经发现，当场取消其考试资格，并根据《北京大学本科考试工作与学术规范条例》及相关规定严肃处理。

5、考生须确认自己填写的个人信息真实、准确，并承担信息填写错误带来的一切责任与后果。

学校倡议所有考生以北京大学学生的荣誉与诚信答卷，共同维护北京大学的学术声誉。

得分

第一题 单项选择题（每小题 2 分，共 30 分）

注：选择题的回答必须填写在下表中，写在题目后面或其他地方，视为无效。

题号	1	2	3	4	5	6	7	8	9	10
回答										
题号	11	12	13	14	15	16	17	18	19	20
回答						/	/	/	/	/

1.在 x86-64 机器上，产生随机的 32 位有符号整数 x 和 y，则下列表达式中**不恒为真**的是：

A. ((x >> 1) << 1) <= x

B. ((x | -x) >> 31) + 1 == !x

C. (-x < y) == (-y < x)

D. (x ^ y ^ ((~x) - y)) == (y ^ x ^ ((~y) - x))

答案：C，一个反例是 x = Tmin，y = 0。

2.在 x86-64 机器上运行如下代码，输出是：

```
char A[12] = "19260817";
char B[12] = "20241104";
void *x = (void *)&A;
void *y = 2 + (void *)&B;
unsigned short P = *(unsigned short *)x;
unsigned short Q = *(unsigned short *)y;
printf("0x%04x", (unsigned short)(P - Q));
```

提示：‘0’，‘1’，...，‘9’的 ASCII 码分别是 0x30，0x31，...，0x39。

A. 0xff05　　B. 0x04ff　　C. 0x0822　　D. 0x0800

答案：B。

3.我们熟悉的标准浮点格式 float 共有 32 位，其阶码字段和小数字段分别为 k=8 位和 n=23 位。如果我们维持 float 的总位数不变，但阶码字段变为 k'=10 位，则下列说法中，正确的是：

A．修改后能表示的实数值的数量更多了
B．修改后能表示的实数值的数量不变
C．修改后能表示的实数值的数量更少了
D．无法确定

答案：A。
修改后，阶码字段增加，小数字段减少，可以表示的 NaN 的数量减少，所以能表示的实数值增多。

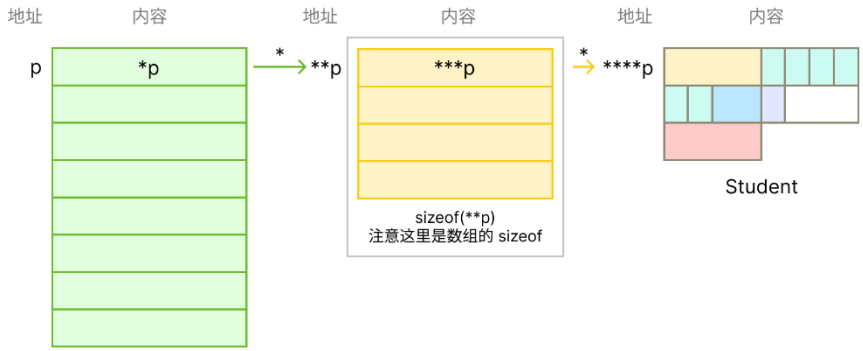
4.在 x86-64 架构、Linux 操作系统下，有如下代码：

```
#include <stdio.h>

struct student {
    int id;
    char name[6];
    short age;
    char gender;
    float score;
};

int main() {
    struct student s;
    struct student* (*p[8])[4];
    printf("%lu\n", sizeof(s));
    printf("%lu\n", sizeof(**p));
    printf("%lu\n", (char*)&p[2] - (char*)p);
    printf("%lu\n", &p[4] - p);
    return 0;
}
```

那么四个输出之和为?
A．72
B．48
C．56
D．84

答案：A。
由于 student 结构体内，最大的数据类型为 int/float，这就要求 student

整体字节数为 4 的倍数。进行内部/外部对齐后，得到 student 的长度为：4(0)+6(0)+2(0)+1(3)+4(0)=20 字节（括号内为对应的内部对齐）。
所以 sizeof(s)=20。
继续看 p 的声明，采用左右法则阅读表达式，可以知道 p 是一个有着 8 个元素的数组，元素类型为指针，指向一个有着 4 个元素的数组，元素类型为指针，指向 student 类型。
所以，我们得到 **p 实际上是一个有着 4 个元素的数组，元素类型为 student 类型的指针，所以 sizeof(**p)=4*8=32。
接着，看 (char*)&p[2] - (char*)p，注意到这里都先强转为了 char* 类型，所以其计算得到的就是 &p[2] 和 p 的实际差值，由于 p 是一个数组，其元素为指针，所以这里的差就是 2 个指针的字节长度，从而 (char*)&p[2] - (char*)p=16。
最后，看 &p[4] - p，与上一行不同，这里没有发生强转，所以实际上计算得到的实际差值会除以步长，从而 &p[4] - p = 4。
综上，此题答案为 20+32+16+4=72。



5.下述关于 x86-64 的栈结构说法错误的是？
A. 被调用者保存寄存器包括：%rbp %rbx %r12 %r13 %r14 %r15
B. 过程的返回地址属于调用者的帧栈
C. 过程传参时，参数 1~6 通过寄存器传递，在参数构造区中存放其他参数，其中，参数 7 最先压栈
D. call 指令的执行过程是：先将下一条指令的地址压栈，随后将 PC 设为目标地址

答案：C。

在参数构造区中存放 7 及以上的参数，参数 7 是最后压栈的。

6.下述说法错误的是？

A. 我们可以使用安全的函数编写程序如 fgets strncpy 指定每次读取的字节数来避免缓冲区溢出，对抗攻击

B. 可以使用地址随机化来随机化程序的内存布局，从而使得攻击代码的位置不再确定，但此处存在权衡：随机的范围必须足够大才足以对抗攻击，但又要足够小避免浪费太多程序空间

C. 可以在过程开始时，从一个特殊的只读段中获取一个金丝雀值放入栈中，在函数返回时，先使用 cmpq 指令来进行比较，随后使用 jle 指令来根据条件码跳转

D. 可以限制哪些内存页内的数据可以当做代码来执行，检查一个页是否可以执行是由硬件来完成的，效率上没有损失

答案：C。

返回时，先使用 xorq 指令进行比较，随后使用 je 指令来根据条件码跳转。

7.在作业题中，我们考察过 GCC 为一个参数和返回值都是结构体的函数产生的汇编代码，此时参数和返回值都是通过栈传递的，以下有关说法错误的是：

A. 返回的结构体的地址被存放在%rax 寄存器中

B. 虽然参数是通过栈传递的，但这里依旧使用到了%rdi

C. 返回的结构体实际上是存放在被调用者的栈帧当中

D. 如果结构体很小的话，GCC 可能会直接用寄存器来传输参数和返回值

答案：C。

C. 返回的结构体实际上存放在调用者的栈帧当中，而非被调用者

8.在 x86-64 中,关于 GCC 对 read1 和 read2 编译的结果,下列说法正确的是：

long read1(long *xp) {
  return (xp ? *xp:0);
}
	long read2(long *xp) {
	  if (xp) return *xp;
	  else return 0;
	}

A. read1 和 read2 的功能一致，并且编译的汇编代码结果也一致

B. read1 在运行时具有安全隐患，当 xp 的值为 NULL 时会导致错误

C. read2 由于可能出现分支预测错误，所以比 read1 的效率更低

D. 对于 cmove(%rdi), %rax 这条指令来说，如果 ZF 为 0，那么就不会报错

答案：A。
B．二者的汇编代码均采用了跳转指令，因此不会出现错误
C．二者汇编代码结果一致，所以效率也是相同的
D．无论条件转移指令的条件是否成立，cmove 都会读取%rdi 指向的内存空间，当%rdi 指向了非法地址时，就会报错

9.在课本中 Y86-64 的 PIPE 处理器设计上执行如下代码片段，只考虑 d_valA 和 d_valB,假设在该段代码执行前和执行后PIPE处理器都执行了足够多的nop指令。请问一共使用到了（）次数据转发/前递（forwarding）。

```
Test:
  xorq %rax, %rax
  irmovq 16, %rbx
  subq %rax, %rbx
  jl .L2
  addq %rax, %rax
  jmp .L3
.L2:
  addq %rbx, %rbx
  addq %rbx, %rax
  addq %rbx, %rcx
.L3:
  ret
```

A．2
B．3
C．4
D．5

4 行使用 2, 3 行
5 行首先预测跳转，9 行进入 D 阶段并使用 4 行，10 行只进入 F 阶段不使用

答案为 3 次（选择 B）。
如果未考虑到预测相关，结果是 2 次。
如果错误理解预测为 not taken，结果为 2 次。
如果不能区分 jl 和 jg，结果为 5 次（10 行和 11 行各使用一次）

10.下列关于 RISC 和 CISC 的表述中，错误的是：
A．RISC 一般没有延迟较长的指令；CISC 有些指令延迟很长

B. RISC 指令编码长度可变；CISC 指令编码长度固定
C. RISC 寻址方式简单；CISC 寻址方式多样
D. 现代指令系统及处理器的设计实现结合了二者思想

答案：B。

11.在 Y86-64 的 SEQ 实现中，对 mem_addr 和 mem_data 的 HCL 补全正确的是：

```
word mem_addr = [
    icode in {IRMMOVQ, IPUSHQ, ICALL, IMRMOVQ} : ____①____;
    icode in {IPOPQ, IRET} : ____②____;
];
word mem_data = [
    icode in {IRMMOVQ, IPUSHQ} : ____③____;
    icode == ICALL : ____④____;
];
```

A. ①valA ②valE ③valE ④valC
B. ①valE ②valA ③valE ④valC
C. ①valA ②valE ③valA ④valP
D. ①valE ②valA ③valA ④valP

答案：D。

12.以下关于存储设备的说法正确的是：
A. SRAM 是双稳态的，只要有电就能保持它的值，所以是非易失性存储器
B. DRAM 每单元所需的晶体管数比 SRAM 少，所以 DRAM 速度一定比 SRAM 更快
C. 随机读写时，SSD 无需像旋转磁盘一样进行多次寻道操作，一般速度更快
D. SSD 有速度快、耗能低等优点，所以任何情况下都应使用 SSD 而不是旋转磁盘

答案：C
A. SRAM 是易失性存储器，非易失性存储器指断电后仍然能保存数据，
B. DRAM 速度一般比 SRAM 慢
C. 正确
D. 旋转磁盘相比 SSD 仍然有价格便宜，不会磨损等优点

13.考虑一个有 x 个盘片，每个扇区 y 字节，每个面 z 条磁道，每条磁道平均 w 个

扇区，旋转速率 a RPM，平均寻道时间 b ms 的磁盘。则下列说法正确的是：

A．磁盘容量为 2wxyz / 2^30 GB

B．访问一个磁盘扇区内容的平均时间为 (60/a * 1000 + 60/a/w * 1000 + b) ms

C．访问一个磁盘扇区中 512 个字节的时间主要是寻道时间和传送时间

D．假设旋转速率未知，在通常情况下我们可以简单估计 a = 60/b * 1/2 * 1000 RPM

答案：D

A. 磁盘容量为 **2wxyz / 10^9 GB**

B. 访问一个磁盘扇区内容的平均时间为 (60/a *** 1/2** * 1000 + 60/a / w * 1000 + b) ms

C. 访问一个磁盘扇区中 512 个字节的时间主要是寻道时间和**旋转延迟**。访问扇区中的 第一个字节用了很长时间，但是访问剩下的字节几乎不用时间。

D. 假设旋转速率未知，在通常情况下我们可以简单估计 a = 60/b * 1/2 * 1000 RPM（书上提到可以用寻道时间的两倍来估计访问时间，故可以用寻道时间估计旋转延迟）

14.考虑通用的高速缓存存储器，下列说法**错误**的是

A．较高的相联度的硬件实现复杂度更大，而且很难使之速度变快。较高的相联度会增加命中时间，因为复杂性增加了。为实现高工作频率，L1 高速缓存一般会选择较低的相联度

B．缓存块越大越好，较大的块能利用程序中的时间和空间局部性，帮助提高命中率

C．在高速缓存的写入策略中，由于局部性，写回能减少总线访问量。考虑缓存一致性，写不命中时不能采用写回，而要采用写分配或写不分配

D．一般而言， 高速缓存越往下层，越可能使用写回而不是直写

答案：B

大的块有利有弊。较大的块能利用程序中可能存在的**空间**局部性，帮助提高命中率。然而大的块对不命中处罚也有负面影响。（较大的块并不能利用时间局部性）

15.假设已有声明 int a, int b, int i, int* pa, int* pb, float fa, float fb, float fc, char s[100], int f(),以及 #include<string.h>，以下优化正确的是：

	原程序	优化程序
A	int c = *pa;	*pb ^= *pa;

	*pa = *pb; *pb = c;	*pa ^= *pb; *pb ^= *pa;

B	a = f();
	b = f();
	int sum = a+b;
		a = f();
		int sum = a+a;

C	int sum = 0;
	for(i = 0; i <
	strlen(s); ++i)
	sum += s[i] - 'a';
		int sum = 0;
		int len = strlen(s);
		for(i = 0; i < len; ++i)
		sum += s[i] - 'a';

D	float ret = fa*fc + fb*fc;	float ret = (fa+fb)*fc;

答案：C

A．pa 和 pb 可能指向同一内存位置

B．f 可能产生副作用

C．正确，string.h 中 strlen 无副作用

D．浮点数运算不满足分配律

得分

第二题（15 分）

GCC 的内联汇编（inline assembly）特性允许在 C 程序中直接嵌入汇编代码，使开发者能够访问常规 C 代码无法直接访问的底层机器特性，例如寄存器的值或特定的处理器指令。通过这种方式，开发者可以对硬件进行精细控制，提升程序性能或实现特定的底层操作。请结合教材第二、三章的相关知识，回答下列问题。

1．下列运行在 x86-64 机器上的 C 代码展示了一个使用内联汇编的例子：

```
float *p = 0x7ffc3608bf40;
float value;
/* inline assembly starts */
__asm__ __volatile__(
    "vmovss (%1), %0"
    : "=x"(value)
    : "r"(p)
);
/* inline assembly ends */
printf("%.2f\n", value);
```

这段代码通过在 C 代码中插入浮点指令 vmovss（Vector Move Scalar Single-Precision Floating-Point）将内存地址 0x7ffc3608bf40 处的浮点数数据读取到 float 类型的变量 value 中。

（1） （2 分）请判断以下说法是否正确（填“是”或“否”）：

a．这段代码也可以在**任何不是** x86-64 的机器上编译、运行。 （ ）

b．插入的内联汇编代码的效果等同于语句”value = *p;”。 （ ）

（2） （2 分）若内存中从地址 0x7ffc3608bf40 开始的 4 个字节的十六进制表示依次为 00 00 70 40，且浮点数的编码方式遵循 IEEE-754 标准，则这段代码的输出为_____。

2．从上一题中可以看出，当需要表示的浮点数的有效位数较少时，使用 IEEE-754 标准编码浮点数会造成一些内存浪费。下面基于 IEEE-754 标准定义一种 8 位的浮点数，符号位、阶码位和尾数位长度分别为 1,5,2。

（1） （4 分）该浮点数格式可表示的**最大的负非规格化数**（不考虑-0）的二进制表示为_____，可表示的**最小的正规格化数**为_____（填十进制数，无需计算出结果中 2 的幂的值）。

（2） （2 分）**只考虑符号位为 0** 的情况，表示 NaN 的浮点数有_____个。

3．接第 1 题，现在我们已经将内存中 0x7ffc3608bf40 处的浮点数数据读取到了变量 value 中，下面的 C 代码使用联合（union）对 value 进行类型转换。

（1） （2 分）请判断以下说法是否正确（填“是”或“否”）：

```
union{
    float f;
    int i;
} converter;
converter.f = value;
printf("%d\n", converter.i);
```

a．这段代码的输出在任何情况下都和语句"printf("%d\n", (int)value);"的输出相同。（ ）

b．float 转换为 int 有可能发生舍入和溢出，但 int 转换为 float 不会发生舍入，也不会出现溢出。（ ）

（2） （3 分）若第 1 题中读取到 value 中的值为 -1104.5，则 printf("%d\n",(int)converter.f+(converter.i >> 31))；的输出为_____。

参考答案：

1．（1） 否；是。 （每空 1 分）

（2） 3.75。 （2 分）

2．（1） 10000001；2^{-14}。 （每空 2 分）

（2） 3。 （2 分）

3．（1） 否；否。 （每空 1 分）

（2） -1105。 （3 分）

解析：

本题题目背景较新颖，将第二、三章内容联系起来，考查学生对有关知识的综合理解。考虑到本题为第一道大题，且覆盖的知识面较广，因此对题目整体难度的要求较低，主要考察学生对基本概念的掌握和理解。

1．（1） a．考查基本概念。该代码使用了 x86-64 指令集特有的 vmovss 指令，只能在 x86-64 架构上运行，其他架构不支持这种特定的汇编指令。

b．考查对指针和指针引用的理解，属于简单题。

（2） 考查浮点数的表示以及 x86-64 的小端法表示，属于基础题。

2．考查对浮点数表示方法的理解，属于基础题。

3．（1） a．考查联合（union）的有关概念。converter.i 和 value 有着相同的位模式，而 (int)value 基于数值对 value 进行类型转换。因此在绝大部分情况下代码的输出和 printf 语句的输出都不同。b． 考查基本概念，属

于简单题。

（2）本题考查浮点数转换为整数的舍入规则（向零舍入）以及有符号整数的基本运算，(int)(converter.f)的值为-1104（向零舍入）。converter.i 和 value 有着相同的位模式，而有符号整数和单精度浮点数的最高位均为符号位，因此(converter.i >> 31)的值为-1。

得分

第三题（15 分）

下列 C 代码实现了一个名为 foo 的函数，请阅读并分析该 C 代码及其对应的 x86 汇编代码，补全代码中缺失的部分，并回答相应问题。（注：第（14）题不要求作答。）

```
void foo(char *p)
{
    int len = strlen(p);
    char tmp = __（1）__;
    *p = __(14)__;
    __(14)__ = '\0';
    if (strlen(p + 1) __（2）__ 1)
    {
        __（3）__;
    }
    __(14)__ = tmp;
}

0000000000001189 <foo>:
    1189:  f3 0f 1e fa          endbr64
    118d:  55                   push   %rbp
    118e:  48 89 e5             mov    %rsp,__（4）__
    1191:  48 83 ec 20          sub    $0x20,%rsp
    1195:  48 89 7d e8          mov    %rdi,-0x18(%rbp)
    1199:  48 8b 45 e8          mov    -0x18(%rbp),%rax
    119d:  48 89 c7             mov    %rax,%rdi
    11a0:  e8 db fe ff ff       call   <strlen@plt>
    11a5:  89 45 fc             mov    %eax,-0x4(%rbp)
    11a8:  48 8b 45 e8          mov    -0x18(%rbp),%rax
    11ac:  0f b6 00             movzbl (%rax),%eax
    11af:  88 45 fb             mov    %al,-0x5(%rbp)
    11b2:  8b 45 fc             mov    -0x4(%rbp),%eax
    11b5:  48 98                cltq
    11b7:  48 8d 50 ff          lea    -0x1(%rax),%rdx
```

```
11bb:  48 8b 45 e8          mov    -0x18(%rbp),%rax
11bf:  48 01 d0             add    %rdx,%rax
11c2:  0f b6 10             movzbl (%rax),%edx
11c5:  48 8b 45 e8          mov    -0x18(%rbp),%rax
11c9:  88 10                mov    %dl,(%rax)
11cb:  8b 45 fc             mov    -0x4(%rbp),%eax
11ce:  48 98                  （5）
11d0:  48 8d 50 ff          lea    -0x1(%rax),%rdx
11d4:  48 8b 45 e8          mov    -0x18(%rbp),%rax
11d8:  48 01 d0             add    %rdx,%rax
11db:  c6 00 00             movb   $0x0,(%rax)
11de:  48 8b 45 e8          mov    -0x18(%rbp),%rax
11e2:  48 83 c0 01          add    $0x1,%rax
11e6:  48 89 c7             mov     （6）  ,%rdi
11e9:  e8 92 fe ff ff       call   <strlen@plt>
11ee:  48 83 f8 01          cmp    $0x1,%rax
11f2:  76 （7）             jbe    1204 <foo+0x7b>
11f4:  48 8b 45 e8          mov    -0x18(%rbp),%rax
11f8:  48 83 c0 01          add    $0x1,%rax
11fc:  48 89 c7             mov    %rax,  （8）
11ff:  e8 85 ff ff ff       call   1189 <foo>
1204:  8b 45 fc             mov    -0x4(%rbp),%eax
1207:  48 98                cltq
1209:  48 8d 50 ff          lea    （9）(%rax),%rdx
120d:  48 8b 45 e8          mov     （10）  ,%rax
1211:  48 01 c2             add    %rax,%rdx
1214:  0f b6 45 fb          movzbl -0x5(%rbp),%eax
1218:  88 02                mov    %al,(%rdx)
121a:  90                   nop
121b:  c9                   leave
121c:  c3                     （11）
```

若在程序中，main 函数调用了 foo 函数，且传递的参数指向字符串"ILOVEICS\0"，则调用该函数后的字符串内容变为（12）。

若 main 函数向 foo 函数传递的参数指向字符串"ICS2024\0"，则 main 函数在调用 foo 函数时，栈中最多会同时存在（13）个 foo 函数的帧。（注意：程序中其他地方不会调用 foo 函数）

（1）_____

（2）_____

（3）_____

（4）_____

（5）_____

（6）_____

（7）_____

（8）_____

（9）_____

（10）_____

（11）_____

（12）_____

（13）_____

参考答案

```
void foo(char *p)
{
    int len = strlen(p);
    char tmp = *p;
    *p = *(p + len - 1);
    *(p + len - 1) = '\0';
    if (strlen(p + 1) > 1)
    {
        foo(p + 1);
    }
    *(p + len - 1) = tmp;
}
```

```
0000000000001189 <foo>:
    1189:  f3 0f 1e fa          endbr64
    118d:  55                   push   %rbp
    118e:  48 89 e5             mov    %rsp,%rbp
    1191:  48 83 ec 20          sub    $0x20,%rsp
    1195:  48 89 7d e8          mov    %rdi,-0x18(%rbp)
    1199:  48 8b 45 e8          mov    -0x18(%rbp),%rax
    119d:  48 89 c7             mov    %rax,%rdi
    11a0:  e8 db fe ff ff       call   <strlen@plt>
    11a5:  89 45 fc             mov    %eax,-0x4(%rbp)
```

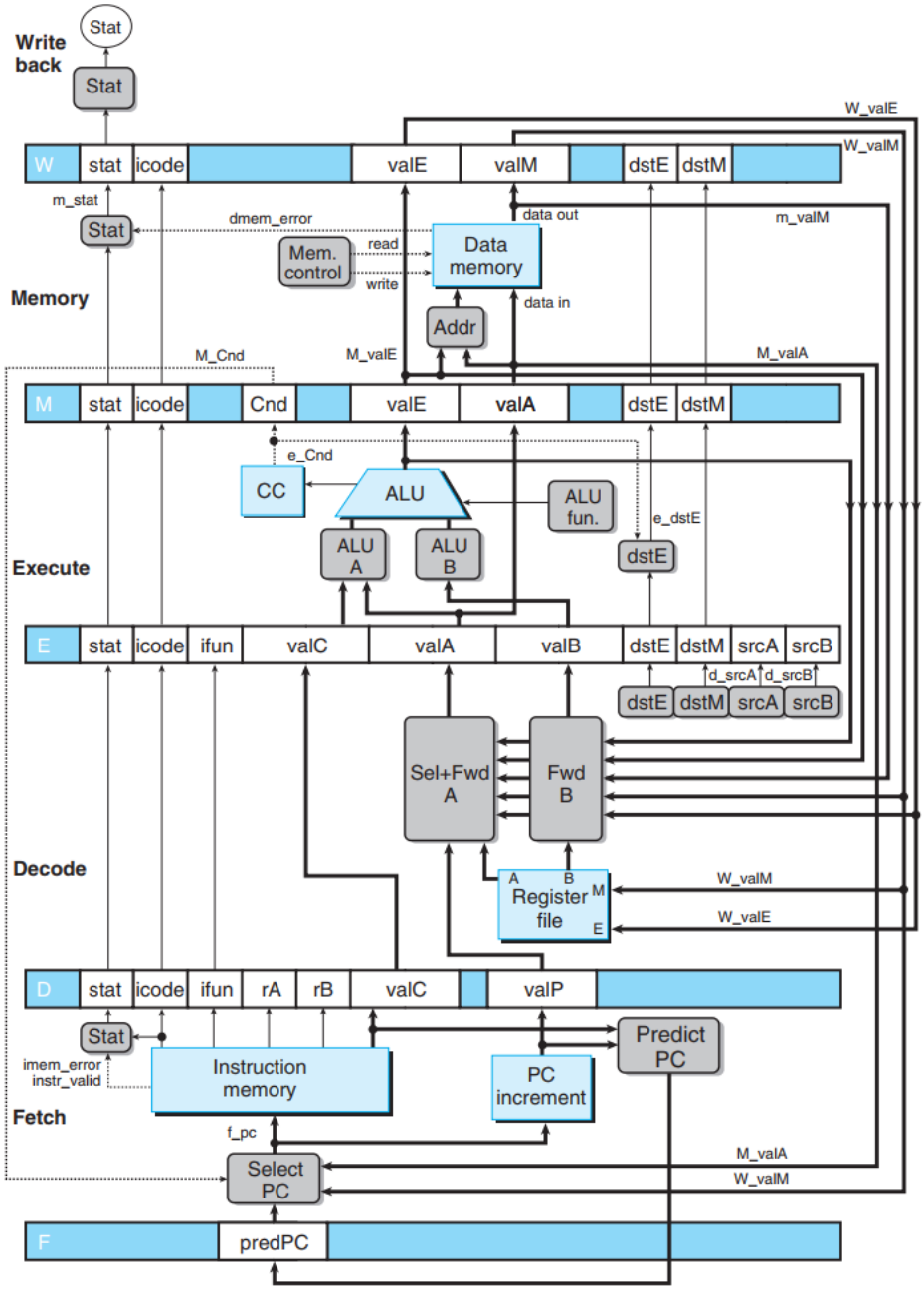
```
11a8:  48 8b 45 e8         mov    -0x18(%rbp),%rax
11ac:  0f b6 00            movzbl (%rax),%eax
11af:  88 45 fb            mov    %al,-0x5(%rbp)
11b2:  8b 45 fc            mov    -0x4(%rbp),%eax
11b5:  48 98               cltq
11b7:  48 8d 50 ff         lea    -0x1(%rax),%rdx
11bb:  48 8b 45 e8         mov    -0x18(%rbp),%rax
11bf:  48 01 d0            add    %rdx,%rax
11c2:  0f b6 10            movzbl (%rax),%edx
11c5:  48 8b 45 e8         mov    -0x18(%rbp),%rax
11c9:  88 10               mov    %dl,(%rax)
11cb:  8b 45 fc            mov    -0x4(%rbp),%eax
11ce:  48 98               cltq
11d0:  48 8d 50 ff         lea    -0x1(%rax),%rdx
11d4:  48 8b 45 e8         mov    -0x18(%rbp),%rax
11d8:  48 01 d0            add    %rdx,%rax
11db:  c6 00 00            movb   $0x0,(%rax)
11de:  48 8b 45 e8         mov    -0x18(%rbp),%rax
11e2:  48 83 c0 01         add    $0x1,%rax
11e6:  48 89 c7            mov    %rax,%rdi
11e9:  e8 92 fe ff ff      call   <strlen@plt>
11ee:  48 83 f8 01         cmp    $0x1,%rax
11f2:  76 10               jbe    1204 <foo+0x7b>
11f4:  48 8b 45 e8         mov    -0x18(%rbp),%rax
11f8:  48 83 c0 01         add    $0x1,%rax
11fc:  48 89 c7            mov    %rax,%rdi
11ff:  e8 85 ff ff ff      call   1189 <foo>
1204:  8b 45 fc            mov    -0x4(%rbp),%eax
1207:  48 98               cltq
1209:  48 8d 50 ff         lea    -0x1(%rax),%rdx
120d:  48 8b 45 e8         mov    -0x18(%rbp),%rax
1211:  48 01 c2            add    %rax,%rdx
1214:  0f b6 45 fb         movzbl -0x5(%rbp),%eax
1218:  88 02               mov    %al,(%rdx)
121a:  90                  nop
121b:  c9                  leave
121c:  c3                  ret
```

12） “SCIEVOLI”

13）3

得分

第四题（20 分）


Write back
Stat
Stat
W_valE
W_valM
W
stat
icode
valE
valM
dstE
dstM
m_stat
Stat
dmem_error
data out
m_valM
Mem. control
read
write
Data memory
Memory
data in
Addr
M_Cnd
M_valE
M_valA
M
stat
icode
Cnd
valE
valA
dstE
dstM
e_Cnd
CC
ALU
ALU fun.
e_dstE
dstE
ALU A
ALU B
Execute
E
stat
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
d_srcA
d_srcB
dstE
dstM
srcA
srcB
Sel+Fwd A
Fwd B
Decode
A
B
Register file
M
E
W_valM
W_valE
D
stat
icode
ifun
rA
rB
valC
valP
Stat
imem_error
instr_valid
Instruction memory
PC increment
Predict PC
Fetch
f_pc
M_valA
W_valM
Select PC
F
predPC

(a)

图(a)是在教材中 Y86-64 的简单流水线处理器（PIPE）实现图。其中部分展示了通过数据前递（forwarding）解决数据冒险(hazard)的流程。

1. （7 分）若当前指令从内存中加载了数据至一个寄存器中，且该寄存器是下一条指令的源寄存器，那么通过数据前递的方式可以减少流水线的暂停（stall）。请根据前递的工作原理，补全下列 HCL 描述语句，以实现正确的 d_valA 的选择。

```
word d_valA = [
     D_icode in {ICALL,IJXX } : ___(1)___;
     d_srcA == e_dstE : ___(2)___;
     d_srcA == M_dstM : ___(3)___;
     d_srcA == M_dstE : ___(4)___;
     d_srcA == W_dstM : ___(5)___;
     d_srcA == W_dstE : ___(6)___;
     1 : ___(7)___;
]
```

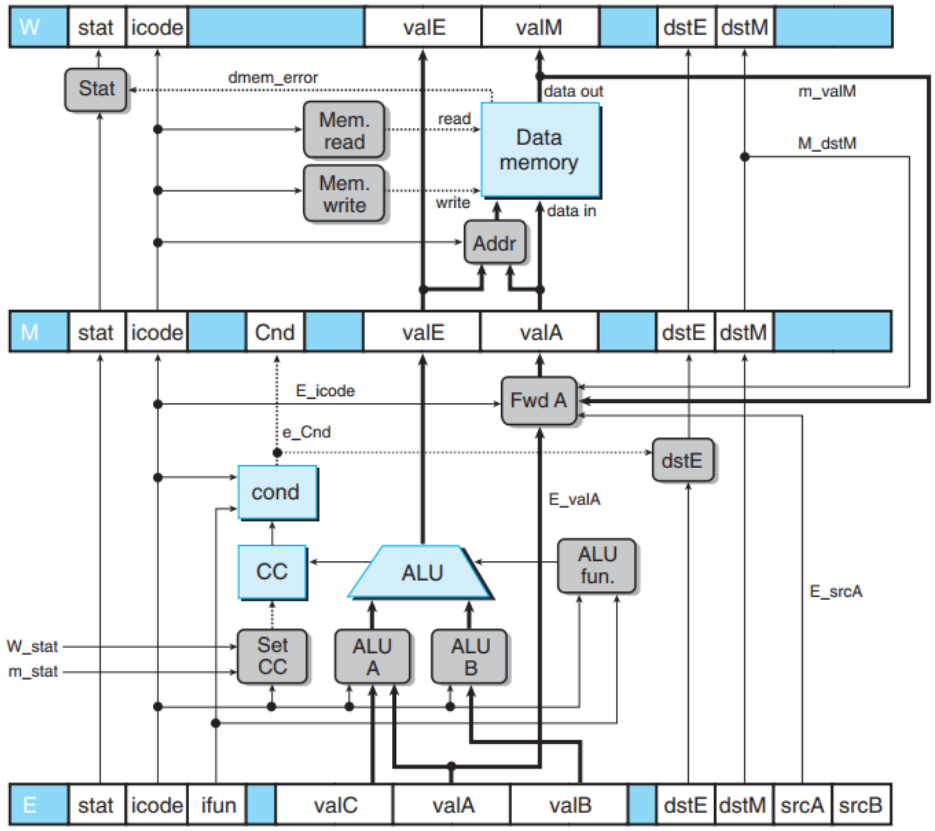
2. （3 分）考虑如下两条 Y86-64 指令系统的指令顺序完成执行：

```
1    mrmovq 0(%rcx),%rdx
2    pushq %rdx
```

这两条指令的执行会导致流水线中出现 Load/Use 数据冒险。当指令 1 执行到(8)阶段时，会获得对应内存地址中的数据，该数据将会以信号(9)进行数据前递。此时按照图(a)的方式，仍然需要停顿来正确执行指令 2，流水线需要停顿(10)个周期来保证程序执行正确。

3. （3 分）针对第 2 小题中的问题，可以通过设计一条额外的数据通路(path)进行前递的方式来解决这种冲突，从而使得指令 1、2 之间无需插入气泡。这条数据通路应当从(11)阶段将数据前递至(12)阶段，用于选择流水线寄存器(13)的值。

4. （3 分）补充通路后的流水线处理器局部如图(b)所示



(b)

请根据图示和设计逻辑，补全下列 HCL 描述语句，以实现正确的 e_valA 的选择。

```
word e_valA = [
      E_icode in { IPUSHQ, IRMMOVQ } && E_srcA ==
    (14)              : (15)                ;
      1 : (16)                ;
]
```

5. （4 分）在补充了第 4 小题中的功能后，流水线依然存在需要暂停才可以处理的数据冒险情况。例如，下两条指令顺序执行：

```
3    popq %rdx
4    rmmovq %rax,0(%rdx)
```

这两条指令会导致流水线中出现 Load/Use 数据冒险，请用 stall、bubble、normal 填写下面的控制信号条件表，说明当前流水线对于这种冒险的处理方式。

Fetch	Decode	Execute	Memory	Write back
(17)	(18)	(19)	(20)	normal

参考答案

1）

```
word d_valA = [
      D_icode in {ICALL,IJXX } : D_valP;
      d_srcA == e_dstE : e_valE;
      d_srcA == M_dstM : m_valM;
      d_srcA == M_dstE : M_valE;
      d_srcA == W_dstM : W_valM;
      d_srcA == W_dstE : W_valE;
      1 : d_rvalA;
]
```

2）

Memory(访存)，m_valM，1；

3）

Memory(访存)，Execute(执行)，M_valA；

4）

```
word e_valA = [
      E_icode in { IPUSHQ, IRMMOVQ } && E_srcA ==
M_dstM : m_valM;
      1 : E_valA;
]
```

5）

Fetch	Decode	Execute	Memory	Write back
stall	stall	bubble	normal	normal

评分标准：每空 1 分

得分

第五题（20 分）

某一计算机的内存空间的大小是32 Bytes，即内存空间的地址范围如下：

0_{10} (00000_2) -- 31_{10} (11111_2)

1. （4分）假设Cache容量大小是8 Bytes，初始状态为空。由于容量过小，需要扩容。若为充分利用原有硬件设计，不希望大幅度修改cache地址解析逻辑，那么最适宜采用的扩容方式是 **C**

   A. 增加set数 B.增大block C.增加相联度 D.以上皆不是

2. （4分）在前题基础上，将Cache容量扩大为16 Bytes。假设对于访问内存地址序列 0_{10},2_{10},4_{10},8_{10},12_{10},14_{10}，其命中率为33.3%，那么请问该Cache的block大小是 **B** ？

   A.2 bytes B.4 bytes C.8 bytes D.以上皆不是

3. 在前题基础上，假设每个Cache Block的大小为4 Bytes（即 B = 4），Cache的结构如下图所示（S=2，E=2），且替换策略为LRU。现有一程序，访问内存地址序列如下所示，单位是Byte。

   2_{10} 23_{10} 13_{10} 9_{10} 20_{10} 15_{10} 13_{10} 10_{10}

   请在下图空白处填入访问上述数据后Cache的状态。（4分）

（TAG使用二进制格式；Data Block使用十进制格式，例：M[4-7]表示地址4_{10}-7_{10}对应的数据）（注：同Set内不同Line的顺序可颠倒）

	V	TAG	Data Block
set0	1	00	M[0-3]
	1	01	M[8-11]

	V	TAG	Data Block
set1	1	10	M[20-23]
	1	01	M[12-15]

上述数据访问一共产生了多少次 Hit（2 分）： 4

4． 在前题基础上，增加一条新规则：地址区间包含10倍数的block将不会被缓存。假设仍旧使用LRU替换策略，请在下图空白处填入访问上述数据后Cache的状态（4分）。（注：同Set内不同Line的顺序可颠倒）

	V	TAG	Data Block
set0	1	00	M[0-3]
	0		

	V	TAG	Data Block
set1	1	01	M[12-15]
	0		

上述数据访问一共产生了多少次Hit(2分)： 2