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：2021 年 11 月 8 日 **小班号：**_____

题号	一	二	三	四	五	六	**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。

以下为试题和答题纸，共 14 页。

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
回答								
题号	9	10	11	12	13	14	15	
回答								

1. 考虑如下代码在 x86-64 处理器上的运行情况：

```
union U{
  char x[12];
  char *p;
}u;

int main() {
  strcpy(u.x, "I love ICS!");
  printf("%p\n", u.p);
  return 0;
}
```

提示：

1）"I love ICS!"对应的字节序列是 49 20 6c 6f 76 65 20 49 43 53 21

2）%p 用于输出一个指针的值

程序运行的输出是：D

A. 0x49206c6f76652049435321

B. 0x215343492065766f6c2049

C. 0x49206c6f76652049

D. 0x492065766f6c2049

考察大小端法

2. 函数 g 定义如下：

```
int g(float f){
  union {
    float f;
    int i;
  } x;
  x.f = f;
  return x.i ^ ((1u << (x.i < 0 ? 31 : 0)) - 1);
}
```

则对于两个 float 型变量 a,b，有 a<b 是 g(a)<g(b)的:B

A. 充分必要条件

B. 充分不必要条件

C. 必要不充分条件

D. 不必要不充分条件

当 a，b 有意义时，a<b 与 g(a)<g(b)等价，而当 a 与 b 是 NaN 或者正负 0 时，g(a)<g(b)无法推出 a<b。

3.阅读如下一段代码

```
int s = 0;
for (int i = 0; i < 32; i++)
s = s ^ i;
printf("%d", s);
```

问运行这段代码后的输出是什么:A

A. 0

B. 1

C. 4

D. 16

列表找规律即可。

4.阅读如下一段代码

```
int x = 0x2021;
int y = -x;
int z = (x / 2) - (x >> 1) + (y / 2) - (y >> 1);
printf("%d", z);
```

问运行这段代码后的输出是什么:C

A. -1
B. 0
C. 1
D. 2

右移向下取整，而除以 2 则是向 0 取整。

5.在 x86-64 机器上，有关栈的描述中，说法不正确的是：C
A. 通过栈传递参数时，所有的数据大小都向 8 的倍数对齐。
B. 对一个局部变量使用地址运算符‘&’，它的地址可以被放在栈上。
C. 栈是向上增长（由低地址向高地址增长）的。
D. 为了保护返回地址不被破坏，通常会在栈中设置金丝雀值（stack canary）。

栈是向下增长的。

6.以下关于 x86-64 指令的描述，说法正确的是：B
A. 数据传送指令 movabsq $Imm, (%rax) 将以 64 位二进制补码表示的立即数 Imm 放到目的地址(%rax)中。
B. INC 和 DEC 指令会设置溢出标志 OF 和零标志 ZF，但不会改变进位标志 CF。
C. call *%rax 指令以%rax 中的值作为读地址，从内存中读出调用目标。
D. popq %rax 指令的行为等效于 movq %rsp, %rax; addq $8, %rsp。

A 中 movabsq 的目的操作数只能是寄存器，C 中%rax 的值即为跳转目标，D 中等效于 movq (%rsp), %rax; addq $8, %rsp。

7.阅读下列 C 代码和在 x86-64 机器上得到的汇编代码：

```
int a[__A__][__B__];
for (int i = 0; i < __C__; i++)
  a[i][__C__ - i] = 1;

  leaq 40(%rdi), %rax
  addq $440, %rdi
.L2:
  movl $1, (%rax)
  addq $40, %rax
  cmpq %rdi, %rax
  jne  .L2
```

假设 a 的地址初始时放在%rdi 中，假设程序正常运行且没有发生越界问题，则 C 代码中的 A、B、C 处应分别填：A

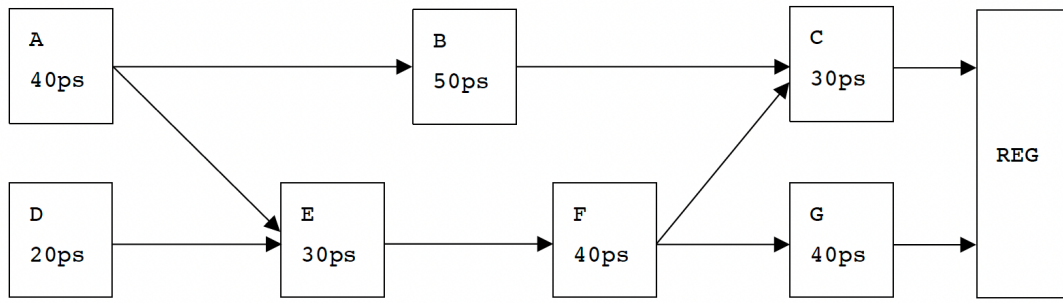
A. 10、11、10

B. 9、11、9

C. 11、10、10

D. 11、11、11

8. A-G 为 7 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所有的依赖关系。寄存器的延迟均为 20ps，在图中以 REG 符号表示。假设流水线寄存器只能添加在有直接依赖关系的基本逻辑单元之间，而不能在 C 或 G 与 REG 之间。以下说法正确的是：D



A. 原电路的吞吐量(throughput)舍入后大约是 1000/150=6.667 GIPS。

B. 将该电路改造成 2 级流水线有 8 种方法

C. 如果将该电路改造成 3 级流水线，延迟最小可以到 80 ps。

D. 不论实现该电路时遇到怎样的数据冒险和控制冒险，一定可以对流水线寄存器使用暂停(stalling)解决。

A. 错。1000/170 = 5.882 而非 1000/150 = 6.667。不要漏掉寄存器。
B. 错。考虑 D-E-F-G 这条路，要么插 D-E，要么 E-F，要么 F-G。如果是 D-E，那么为了使每条极大路径上都恰有一个新插入的寄存器，考虑 D-E-F-C，那么 F-C 之间也不能插入了。但是 A-E-F-G 上又必须有一个，所以只能插 A-E。这样之后不管是插在 A-B 还是 B-C 所得方案都是合法的。对称地，如果是 F-G，结果也是如此。最后考虑 E-F 的情况，此时 A-E，D-E，F-C，F-G 之间均不能再插入，但 A-B 和 B-C 任选一个插入得到的方案都合法。因此一共有 2+2+2=6 种。
C. 错。3 阶段里的最优为 90ps。
D. 对。最朴素的情况每条指令 stall 足够多次，电路回到 SEQ 的状态。

9．在课本 Y86-64 的 PIPE 上执行以下的代码片段，一共使用到了（ D ）次数据转发。假设在该段代码执行前和执行后 PIPE 都执行了足够多的 nop 指令。

```
mrmovq 0(%rdx), %rax
addq   %rbx, %rax
mrmovq 8(%rdx), %rcx
addq   %rcx, %rax
irmovq $10, %rcx
addq   %rcx, %rax
rmmovq %rax, 16(%rdx)
```

A. 3　　B. 4　　C. 5　　D. 6

红色表示使用到数据转发的寄存器

```
mrmovq 0(%rdx), %rax
addq %rbx, %rax (1 stall)
mrmovq 8(%rdx), %rcx
addq %rcx, %rax (1 stall)（Decode 时第二条指令结果还未写回）
irmovq $10, %rcx
addq %rcx, %rax
rmmovq %rax, 16(%rdx)
```

10．在书中 Y86 的 SEQ 实现下，以下哪一条指令是现有信号通路能完成的：C
A. iaddq rA, V：将立即数 V 与 R[rA]相加，其中 rB 域设为 F，结果存入寄存器 rA
B. mmmovq rA, rB：将 R[rA]存的地址开始的 8 字节数据，移动到 R[rB]存的地址
C. leave：相当于先执行 rrmovq %rbp, %rsp，再执行 popq %rbp
D. enter：相当于先执行 pushq %rbp，再执行 rrmovq %rsp, %rbp

A 和 B 还是比较显然的，D 中涉及两次 valE 的写，这是做不到的，两写只能一个是 valE，一个是 valM。

11．下列有关存储器的说法中，正确的是：C

A．DMA 技术是指，当需要磁盘中的数据时，比起将磁盘数据先传送到主存，可以直接将磁盘数据通过总线传送到寄存器中，以提升效率

B．对于只有一个盘片，一个读写头的旋转磁盘，在分配逻辑块时，最好让属于同一逻辑块的扇区均匀分布在不同磁道上，以减少读取一个逻辑块的时间

C．SSD 相比于旋转磁盘，不需要寻道，但更容易磨损，因此常使用平均磨损逻辑以提高 SSD 的寿命

D． SRAM 的电路可以无限期保持其状态，断电之后依旧可以保存信息，而 DRAM 需要周期性地刷新状态，断电后信息将消失

解析：

A. 错误。DMA 技术是将磁盘的内容直接传送到主存，而不是寄存器。

B. 错误。为了减少读取一个逻辑块的时间，最好让属于同一逻辑块的扇区分布在同一个磁道上，以减少寻道时间。

C. 正确。见教材 415 页。

D. 错误。SRAM 属于易失性存储器，断电后不会保存信息。

12．下列有关存储器的说法中，错误的是：D

A．旋转磁盘对扇区的访问时间有三个主要的部分：寻道时间、旋转时间和传送时间

B．在旋转磁盘中，磁盘控制器和逻辑块的设计理念有利于为操作系统隐藏磁盘的复杂性，同时也有利于对损坏的扇区进行管理

C．DRAM 和磁盘的性能发展滞后于 CPU，现代处理器为了弥补 CPU 的访存需求和内存延迟的差距，频繁地使用高速缓存

D．SSD 的闪存芯片包含若干个块，一个块由若干页组成，写入页的时间和该页是否包含数据（是否全为 1）无关

A. 正确。见教材 409 页。

B. 正确。见教材 410 页。

C. 正确。见教材 417 页。

D. 错误。见教材 415 页，如果写操作试图修改一个包含已经有数据的页时，那么这个块中所有带有数据的页都必须被复制到一个新（擦除过的）块，然后才能进行对该页的写。因此，写入页的时间与该页是否包含数据有关。

13．分析下面的 C 程序，以下关于局部性(locality)说法错误的是：D

```
int i = 0, a = 1, b = 1;
for (;i < 100; i++)   {
    a = a + b;
    b = a + b;
}
```

A．体现了数据的时间局部性

B．体现了指令的时间局部性

C．体现了指令的空间局部性

D．以上都没有体现

解析：对于有意义的循环操作，均体现了指令的时间空间局部性。该段程序对 a b 的循环访问也体现了数据的时间局部性。

14．假设已有声明 int i, int j, float x, int y, const int n, int a[n], int b[n], int *p, int *q int *r, int foo(int), 以下哪项程序优化编译器总是可以进行：D

A	for (i = 0; i < n; i++) x = (x+ a[j]) + b[j]	for (i = 0; i < n; i++) x= x + (a[j] + b[j])
B	*p += *q *p += *r	int tmp; tmp = *q + *r *p += tmp

C	for(i = 0; i < foo(n);
	i++)
	sum += i;
		int tmp = foo(n)
		for(i = 0; i < tmp; i++)
		sum += i;

D	for (i = 0; i < n; i++) y = (y * a[j]) * b[j]	for (i = 0; i < n; i++) y = y * (a[j] * b[j])

A．浮点数运算不具有结合性
B．pqr 内存别名引用
C．函数 foo 可能具有副作用

15. 请阅读下面的 Y86-64 代码，并计算它在 PIPE（书中图 4.52）流水线处理器上运行所需要的总周期数以及运行结束后%rax 的值。假设分支预测默认为跳转。只用统计从 MAIN 函数的第一条指令进入 PIPE 到 ret 执行完毕所需要的周期数。(本题无效)

执行周期数和%rax 的值（10 进制）是：C

```
.align 8
Array:
    .quad 0x0
    .quad 0x2
    .quad 0x3
    .quad 0x5

MAIN:
irmovq   Array,   %rdi
irmovq    $4,    %rsi
irmovq    $8,    %r10
irmovq    $0,    %rax
irmovq    $1,    %r8
LOOP:
mrmovq   (%rdi),  %rdx
rrmovq    %rdx,   %r9
subq     %rsi,    %r9
jg        AD
    addq      %rdx,   %rax
    jmp       CHECK
AD:
    addq      %rsi,    %rax
CHECK:
    addq      %r10,   %rdi
    subq      %r8,    %rsi
    jne         LOOP
ret
```

A) 56, 10
B) 55, 10
C) 57, 9
D) 55, 9

得分

第二题 请结合教材第二章“信息的表示和处理”的相关知识,回答下列问题（10分）

1．假设某 x86-64 机器在地址 0x100 和 0x101 处存储的数据用二进制表示分别为$[1010\ 1100]_2$和$[1111\ 1011]_2$.又假设 x 是一个 short 类型的变量,其地址为 0x100.则 x 的十六进制补码表示为 0x____(1)____,这是一个____(2)____(填“正”或“负”)数，其绝对值为____(3)____(用十进制数字表示).

2．设整型变量采用 32 位补码表示,判断正误(填“√”或“×”):

i．设 x,y,z 是整型变量,且 x<y<z<0,则(-y)>(-z)>0．(____(4)____)

ii．表达式“(-5) + sizeof(int) < 0”为真．(____(5)____)

3．考虑一种新的遵从 IEEE 规范的浮点格式系统,该浮点数系统包含 1 位符号,4 位阶码,7 位尾数.

i．该浮点数系统所能表示的最大规格化数(十进制)为____(6)____.

ii．将$-\frac{3}{256}$用该浮点数系统表示,写成十六进制为 0x____(7)____.按照向偶数舍入的原则，把$\frac{3}{256}$的二进制表示舍入到最近的一百二十八分之一,将得到____(8)____(填最简真分数).

4．有下面一段程序:

```
int i, j, k;
for(i = 0; i <= 2147483647 - 2; i ++){
  j = i + 1;
  k = j + 1;
  float *x = (float *)(&i);
  float *y = (float *)(&j);
  float *z = (float *)(&k);
  if(*y - *x != *z - *y){
    printf("0x%08x\n", i);      //输出 8 位 16 进制数
    break;
  }
}
```

其中 float 类型表示 IEEE-754 规定的浮点数，包括 1 位符号，8 位阶码，23 位尾数．请问该程序是否会有输出？____(9)____(填“是”或“否”)．若有输出，请给出输出内容，若没有输出，请说明理由____(10)____．

参考答案：

(1)fbac (2)负 (3)1108 (4)√ (5)× (6)255 (7)860 (8)$\frac{1}{64}$

(9)是 (10)0x00ffffff

解析：本大题共设基础题 5 分，中档题 3 分，难题 2 分．

I．本题考察数据换算和大小端，属基础题．首先通过 x86-64 判断该机器使用小端法，由此得到该短整型的 16 进制表示为**(1)0xfbac**．由于其二进制最高位为 1，因此 x 是一个**(2)负**数．将其取反加一得到 0x0453=1108，因此 x 的绝对值为**(3)1108**．

II．本题考察整数的表示和运算，属基础题．因为 x 严格小于 y，z，所以 y 和 z 都不是 TMin，所以答案为**(4)√**．由于 sizeof()的返回值是无符号数，因此左边的表达式被强制转化为无符号数 UMax，因此填**(5)×**．

III．本题考察 IEEE 浮点数标准，浮点表示和舍入，属中档题．最大规格化数的二进制表示为 011101111111，值为**(6)255**．注意到该浮点系统下最小的负非规格化数为$-\frac{1}{64}$，所以$-\frac{3}{256}$是非规格化数，其 16 进制表示为 0x**(7)860**．$\frac{3}{256}$的二进制表示为 0.00000011，根据向偶数舍入的规则得到(8)$\frac{1}{64}$．

IV．本题考察非规格化浮点数与规格化浮点数的表示方法，属难题．阶码为 1 时的规格化浮点数和非规格化浮点数相邻两个数之间的差是相同的，过渡平滑．因此答案为(9)是，**(10)0x00ffffff**．

得分

第三题（15 分）请阅读并分析下面的 C 语言程序和对应的 x86-64 汇编代码。

1．其中，有一部分缺失的代码（用标号标出），请在标号对应的横线上填写缺失的内容。注:汇编与机器码中的数字用 16 进制数填写。

C 代码如下：

```
long f(long n, long m)
{
  if (n == 0 || ___ (1) ___)
  return m;
  if (___(2)___)
  {
    long ret = ___ (3) ___;
    return ret;
  }
  else
  {
    long ret = f(n - 1, m >> 1);
    return ret;
  }
}
```

x86-64 汇编代码如下（为简单起见，函数内指令地址只给出后四位，需要时可补全）：

```
0x0000555555555149 <f>:
    5149:  f3 0f 1e fa          endbr64
    514d:  55                   push   %rbp
    514e:  48 89 e5             mov    __ (4) __,%rbp
    5151:  48 83 ec 20          sub    $0x20,%rsp
    5155:  48 89 7d e8          mov    %rdi,-0x18(%rbp)
    5159:  48 89 75 e0          mov    %rsi,-0x20(%rbp)
    515d:  48 83 7d e8 00       cmpq   $0x0,-0x18(%rbp)
    5162:  74 __ (5) __          je     __ (6) __
    5164:  48 83 7d e0 01       cmpq   $0x1,-0x20(%rbp)
    5169:  75 06                jne    5171 <f+0x28>
    516b:  48 8b 45 e0          mov    __ (7) __(%rbp),%rax
    516f:  eb 5f                jmp    51d0 <f+0x87>
```

```
5171:  48 8b 45 e0         mov    -0x20(%rbp),%rax
5175:  83 e0 01            and    $0x1,%eax
5178:  48 85 c0            test   %rax,%rax
517b:  74 ??               je     __ (8) __
517d:  48 8b 55 e0         mov    -0x20(%rbp),%rdx
5181:  48 89 d0            mov    %rdx,%rax
5184:  48 01 c0            add    %rax,%rax
5187:  48 01 d0            add    %rdx,%rax
518a:  48 8d 50 01         lea    0x1(%rax),%rdx
518e:  48 8b 45 e8         mov    -0x18(%rbp),%rax
5192:  48 83 e8 01         sub    $0x1,%rax
5196:  48 89 d6            mov    __ (9) __,%rsi
5199:  48 89 c7            mov    %rax,%rdi
519c:  e8 a8 ff ff ff      callq  5149 <f>
51a1:  48 89 45 f8         mov    %rax,-0x8(%rbp)
51a5:  48 8b 45 f8         mov    -0x8(%rbp),%rax
51a9:  eb 25               jmp    51d0 <f+0x87>
51ab:  48 8b 45 e0         mov    -0x20(%rbp),%rax
51af:  48 d1 f8            __ (10) __    %rax
51b2:  48 89 c2            mov    %rax,%rdx
51b5:  48 8b 45 e8         mov    -0x18(%rbp),%rax
51b9:  48 83 e8 01         sub    $0x1,%rax
51bd:  48 89 d6            mov    __ (9) __,%rsi
51c0:  48 89 c7            mov    %rax,%rdi
51c3:  e8 81 ff ff ff      callq  5149 <f>
51c8:  48 89 45 f0         mov    %rax,-0x10(%rbp)
51cc:  48 8b 45 f0         mov    -0x10(%rbp),%rax
51d0:  c9                  leaveq
51d1:  c3                  retq
```

2．已知在调用函数 f(7,6)时，我们在 gdb 中使用 b f 指令在函数 f 处加上了断点，下面是程序某一次运行到断点时从栈顶开始的栈的内容，请在空格中填入相应的值。（U 表示不要求填写）

0x7fffffffe558	0x00005555555551c8
0x7fffffffe550	(11) _____
0x7fffffffe548	U
0x7fffffffe540	U
0x7fffffffe538	U
0x7fffffffe530	(12) _____
0x7fffffffe528	(13) _____
0x7fffffffe520	0x00007fffffffe550
0x7fffffffe518	U
0x7fffffffe510	U
0x7fffffffe508	(14) _____
0x7fffffffe500	0x0000000000000010
0x7fffffffe4f8	0x00005555555551c8

3．运行函数 f(7,6)后得到的值是多少？ (15) _____

答案：

(1) m == 1

(2) m & 1 或 m % 2 == 1 或其余相同语义的表达式

(3) f(n - 1, m * 3 + 1)

(4) %rsp

(5) 07

(6) 516b <f+0x22>

(7) -0x20

(8) 51ab <f+0x62>

(9) %rdx

(10) sar

(11) 0x00007fffffffe580

(12) 0x0000000000000005

(13) 0x00005555555551a1

(14) 0x0000000000000003
(15) 2

得分

第四题（15 分）

请分析 Y86-64 ISA 中加入的一族间接跳转指令：jxx *rB，其格式如下：

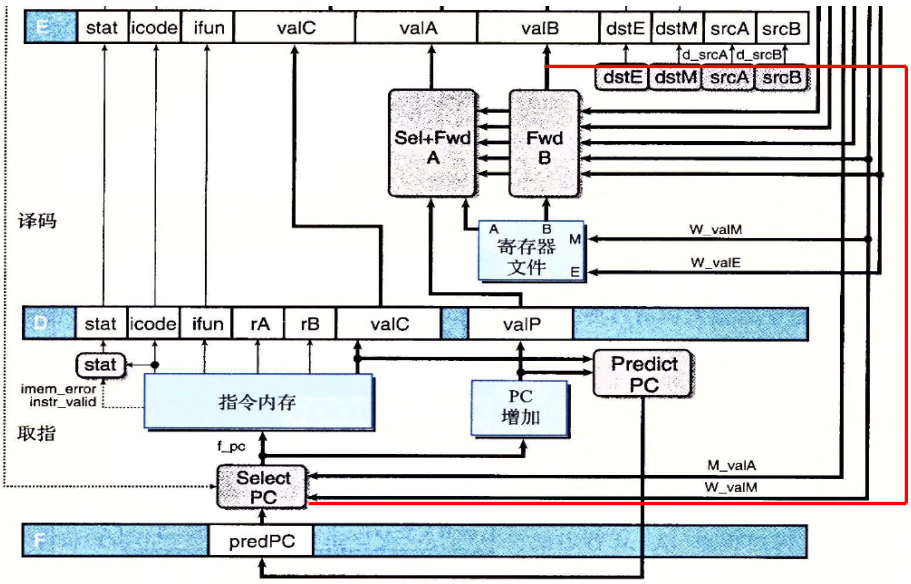
C	Fn	F	rB

该指令的功能是跳转到寄存器 R[rB] 所存放的地址。类似于直接跳转指令，间接跳转指令也包括无条件跳转和条件跳转，通过不同的功能码 Fn 来指示。为了和直接跳转区别，icode 为 IJREGXX。时钟周期适当进行延长，在不修改原有的硬件线路和信号设置的前提下，只增加和新指令有关的逻辑，回答以下问题。

1．在教材中的 SEQ 处理器上实现该指令，请补全下表中每个阶段的操作。需要说明的信号可能有：icode, ifun, rA, rB, valA, valB, valC, valE, valP, Cnd, R[], M[], PC, CC

Stage	jxx *rB
Fetch	icode : ifun ← M_1[PC]
Decode	
Execute	
Memory	
Write Back	
Update PC	PC <- Cnd ? valE: valP

2．考虑在教材中的 Pipeline 处理器上实现该指令，采用总是选择分支（预测下一条指令时使用跳转地址 R[rB]）的预测策略。



d

由于 R[rB]需要到译码阶段才能得到，需要增加一条从 Fwd B 输出信号 d_valB 到 Select PC 的旁路通路，增加线路如图所示。增加旁路后，为了预测 jxx *rB 的下一条 PC，_____（需要/不需要）在该指令和下一条指令间插入气泡。

Select PC 的 HCL 代码如下图所示：

```
word f_pc = [
   ①
   (M_icode == IJXX || M_icode == IJREGXX) && !M_cnd :
  M_valA;
   ②
   W_icode == IRET : W_valM;
   ③
   1 : F_predPC;
   ④
]
```

为了预测下一条 PC，需要修改 Select PC 的 HCL 代码，增加一行 _____ : _____，增加的位置可以是 _____ （写出所有可能的位置，错填不得分，漏填可得部分分）

3．请将该指令预测错误的触发条件，以及此时流水线的控制逻辑补充完整。

触发条件：（如果有多种可能请任写一种）

```
________ == IJREGXX && _______
```

控制逻辑：（如果有多种可能请任写一种）

F	D	E	M	W

4．基于改造后的 Y86-64 PIPE 考虑如下代码片段，回答问题。

```
# Array of 3 elements
array:
.quad return
.quad L1
.quad L2

# void foo(long n, long *arr)
# n in %rdi, arr in %rsi
foo:
rrmovq %rdi, %rdx          # line 1
addq %rdx, %rdx            # line 2
addq %rdx, %rdx            # line 3
addq %rdx, %rdx            # line 4
irmovq array, %rcx         # line 5
addq %rdx, %rcx            # line 6
andq %rdi, %rdi            # line 7
jge *%rcx                  # line 8
return:                    #
ret                    # line 9
L2: #
mrmovq 16(%rsi), %rcx      # line 10
rmmovq %rcx, 8(%rsi)       # line 11
L1:                        #
mrmovq 8(%rsi), %rcx       # line 12
rmmovq %rcx, (%rsi)        # line 13
jmp return                 # line 14
```

在 foo 函数运行过程中，计算以下情况 foo 函数的执行周期数。（周期数计算从执行 foo 第一条指令开始，直到其返回指令 ret 完全通过流水线为止。另外假设 foo 函数开始的若干条指令不会和 foo 函数体外的指令形成冒险。）

n=-1：_____；n=0：_____；n=2：_____

1.

Stage	jxx *rB
Fetch	icode : ifun ← M_1[PC] **rA : rB <- M_1[PC+1]** **valP <- PC+2**
Decode	**valB <- R[rB]**
Execute	**valE <- valB + 0** **Cnd = Cond(CC, ifun)**
Memory	/
Write Back	/
Update PC	PC <- Cnd ? valE: valP

2．**不需要。**间接跳转在 decode 阶段时 SelectPC 正好利用最新转发的 valB 的值作为预测地址访问指令内存。

1 2 3 处都可以插入。插入的内容见下。④是无效的插入位置。注意，原来的 M_XXX 条件和 W_XXX 条件互斥，所以其顺序任意，但它们都不会和新加入的条件冲突。例如 mispredict 发生时，Decode 阶段的指令已经被清空，所以最终不会导致多个条件成立。

```
word f_pc = [
   // D_icode == IJREGXX: d_valB;
   (M_icode == IJXX || M_icode == IJREGXX) && !M_cnd : M_valA;
   // D_icode == IJREGXX: d_valB;
   W_icode == IRET : W_valM;
   D_icode == IJREGXX: d_valB;
   1 : F_predPC;
   ④
]
```

3. **E_icode == IJREGXX && !e_Cnd**

F	D	E	M	W
normal/stall	**bubble**	**bubble**	**normal**	**normal**

分析方法同 IJXX。注意，触发条件是在 E 阶段，因为 E 阶段结束、M 阶段开始时就要控制流水线寄存器。

4. **15 13 20**

n == -1: line 8 分支预测错误，惩罚 2 个周期。一共 9 + 2 + 4 (trailing cycles for ret) = 15。

n == 0: 一共 9 + 4 (trailing cycles for ret) = 13。

n == 1: line 12 和 13 发生 load/use hazard （不考虑加载转发），惩罚 1 个周期。一共 12 + 1 + 4 (trailing cycles for ret) = 17。

n == 2: line 12 和 line 13，以及 line 10 和 line 11 各发生一次 load/use hazard，共惩罚 2 个周期。一共 14 + 2 + 4 (trailing cycles for ret) = 20。

得分

第五题（15 分）说明：本题的 cache 都是写分配的

1. 请从 SRAM 和 DRAM 中，选出更符合描述的一项，并在表格里打✔（每行 1 分，共 3 分）

	SRAM	DRAM
常用于当代个人电脑 CPU 中的高速缓存	✔	
在 10-100 毫秒内会失去电荷，因此需要周期性地刷新		✔
单位容量价格更高	✔	

解析：考察教材 6.1.1 中的“1.静态 RAM”和“2.动态 RAM”的区别
评论：该题都是基本概念，较为简单

2. 在一个 16-bit 地址空间的机器上，有一个总大小为 16 Bytes 的 cache，替换策略为 LRU，回答如下问题（本题不考虑编译优化）（共 12 分）：
（1）若每个 block 的大小为 4 Bytes，采用**直接映射**的方式组织，则 tag 段长度为 12 bits。（3 分）

解析：考察 6.4.1“通用的高速缓存存储器组织结构”
地址长度 m=16 bits
E=1
B=4 Bytes, b=2 bits
S=C/(E*B)=4，s=2 bits
故 tag 长度为 m-s-b=12 bits
评论：该题考查基本概念，计算量小，较为简单

（2）在如下代码中，假设**初始时 cache 为空，只考虑读写数据**引起的 cache 访问（共 5 分）：

```
# define N 4
char x[N];
short y[N];
int main() {
    for (int i = 0; i < N; i++) {
        x[i] = i;
        y[i] = 1 - i;
    }
    return 0;
}
```

① 若数组 x、y 的起始地址分别为 0x0000、0x0008，cache block 大小为 2 Bytes，组相联度为 2，则命中次数为 2 次。（3 分）

② 若数组 x、y 的起始地址与①相同，cache block 大小为 4 Bytes，则无论采取何种方式组织 cache、且无论如何增大 cache 的大小，命中次数至多为 5 次。（2 分）

解析：考察 6.4.3“组相联高速缓存”

（1）中，仅有对 x[1]和 x[3]的访问会命中

（2）中，只需要考虑访问序列分散在几个 block 中（block 大小为 4 Byte），因此去掉地址的后 2 个 bit，分析还有多少不重复的元素即可

评论：这两空是较为基础的计算，如果按照朴素的方法计算，有一定的计算量，难度适中；如果能够找到规律，计算其实较为简单

（3）在如下代码中，假设初始时 cache 为空，只考虑读写数据引起的 cache 访问，假定：

a. Cache 命中的延迟（hit time）为 10 周期，不命中处罚（miss penalty）为 190 周期；

b. 延迟与数据的长度无关；

c. 若数据对应到不同的缓存块，则需进行多次访问，且访问之间不能重叠（即访存不能并行）。

（共 4 分）

```
# define N 4
typedef struct {
    char x;
    short y;
} node;
node p[N];
int main() {
    for (int i = 0; i < N; i++) {
        p[i].x = i;
        p[i].y = 1 - i;
    }
    return 0;
}
```

① 若数组 p 的起始地址为 0x0000，cache block 大小为 8 Bytes，采用直接映射，则 cache 访问的总延迟为 460 周期（注意结构体数据对齐）。（2 分）

② 若结构体不采用数据对齐（即 (void *)&p[i] - (void *)&p[0] == 3 * i），则上述代码在①中的条件下，则 cache 访问的总延迟为 470 周期。（2 分）

解析：考察点与 2.2 相同，探索了数据对齐的作用（现代 cache 允许并行访问，但是本题仍不失为一种启发）

②比①多 10 个周期的原因在于②中对 p[2].y 的访问跨两个缓存块

评论：（3）难度与（2）相比并无明显提升，仍是一道难度适中的题目。

得分

第六题（15 分）

```
struct s_element{
union u_inner{
    float f;
    short s[2];
} u1;
    s_element* next;
};
float func(s_element* p){
    float ans = 0;
    while( p ){
        ans += p->u1.f;
        p = p->next;
    }
    return ans;
}
```

1. 若 tmp 是一个 s_element 类型的变量，则 sizeof (tmp.u1) = _____。（1 分）

2. 简述函数 func 的功能。（2 分）

3. 函数 func 循环部分的汇编代码如下，补全汇编代码：（2 分）

提示：vaddss S1, S2, D 实现单精度浮点数加法 D ← S1+S2

```
L1:
    vaddss     (%rdx), %xmm0, %xmm0
    ____________________________________
L2:
    testq      %rdx, %rdx
    jne L1
```

4．在学习了循环展开后，同学 I 想对 func 函数进行优化。他决定尝试模仿书上 2*2 循环展开的方法，使用两个累积变量以提高代码并行性，请补全他改进后的函数 func1 的代码。（1*4=4 分）

```
float func1(s_element* p){
    float ans1 = 0, ans2 = 0;          //line 1
    while( p && ____①____){            //line 2
        ans1 += p->u1.f;               //line 3
        ans2 += _____②_____;           //line 4
        p = _____③______;              //line 5
    }
    if(_____④______)                   //line 6
        ans1 += p->u1.f;             //line 7
    return ans1 + ans2;              //line 8
}
```

将要补全的代码填在下面：

① _____

② _____

③ _____

④ _____

5．此时，同学 C 指出同学 I 的做法会导致新的错误，原因是_____。（2 分）

6．同学 S 从书上读到，k * k 循环展开在 k 很大时反而可能获得较差的效果，这是因为 k 很大时会导致_____。（2 分）

7．同学 B 使用了同学 I 改进后的 func1 函数实现下面这段代码，并在大端法机器上运行，最终得到的输出是：_____。（2 分）

```
s_element x, y, z;
    x.u1.f = 1.2;
    y.u1.f = 3.55;
    x.next = &y;
    y.next = nullptr;
    z.u1.f = func1(&x);
    printf("0x%x  %d\n",  z.u1.s[0]-z.u1.s[1],  &z.u1.s[0]-
&z.u1.s[1]);
```

答案：

1）4（考察 union 的大小是最长字段 + 末尾 padding）

2）func 函数计算一个 s_element 链表中所有元素 u1.f 的和。（考察源代码阅读）

3）movq 8(%rdx), %rdx （考察第三章汇编代码，同时考察结构体内部对齐的 padding。注意立即数前面没有$）

4）（考察代码阅读能力）

① p->next

② p->next->u1.f

③ p->next->next

④ p

5）浮点数加法不满足结合律，可能会导致 func1 和 func 函数结果不同。（考察浮点数的结合性，重新结合变量在循环展开优化时的限制）

6）寄存器溢出（，过多的循环变量会分配到栈上）（教材 378 页内容，答出寄存器溢出的意思即可，考察对关键路径和循环展开的理解）

7）0x4098 -1（考察浮点数的表示，大小端，同类型指针之差是地址差除以数据差，一半 1 分）