**2013**

7、x86体系结构的内存寻址方式有多种格式，请问下列哪些指令是正确的：（ ）

A. movl $34, (%eax)

B. movl (%eax), %eax

C. movl $23, 10(%edx, %eax)

D. movl (%eax), 8(%ebx)

答案：ABC，寻址不支持内存到内存的访问

15、以下哪些程序优化编译器总是可以自动进行？（假设int i, int j, int A[N], int B[N], int m都是局部变量，N是一个整数型常量，int foo(int) 是一个函数）答：（ ）

|  |  |  |
| --- | --- | --- |
|  | 优化前 | 优化后 |
| A. | for (j = 0 ; j < N ; j ++)  m + = i\*N\*j; | int temp = i\*N;  for (j= 0 ; j < N ; j ++)  m + = temp \* j; |
| B. | for (j = 0 ; j < N ; j ++)  B[i] \*= A[j]; | int temp = B[i];  for (j= 0 ; j < N ; j ++)  temp \*= A[j];  B[i] = temp; |
| C. | for (j = 0 ; j < N ; j ++)  m = (m + A[j]) + B[j]; | for (j = 0 ; j < N ; j ++)  m = m + (A[j] + B[j]); |
| D. | for (j = 0 ; j < foo(N) ; j ++)  m ++; | int temp = foo(N);  for (j= 0 ; j < temp ; j ++)  m ++; |

答案：AC

17、关于局部性（locality）的描述，不正确的是：（ ）

A. 数组通常具有很好的时间局部性

B. 数组通常具有很好的空间局部性

C. 循环通常具有很好的时间局部性

D. 循环通常具有很好的空间局部性

答案：A

2）请按IEEE浮点标准的单精度浮点数表示下表中的数值，首先写出形如(-1)s ×M×2E的表达式，然后给出十六进制的表示。（每格1分）

注：单精度浮点数的字段划分如下：

符号位（s）：1-bit；阶码字段（exp）：8-bit；小数字段（frac）：23-bit；偏置值（bias）：127。

|  |  |  |
| --- | --- | --- |
| Value | (-1)s ×M×2E，1<=M<2 | Hex representation |
|  |  |  |
| 2-149 |  |  |

答案

|  |  |  |
| --- | --- | --- |
| Value | (-1)s x M x 2E  1<=M<2 | Hex representation |
|  | (-1)x 1.1 x 20 | 0xBFC00000 |
| 2-149 | 1.0 x 2-149 | 0x00000001 |

## **第四题（10分）**

**阅读下面的汇编代码：**

<f>:

4004c4: push %rbp

4004c5: mov %rsp,%rbp

4004c8: sub $0x10,%rsp

4004cc: mov %edi,-0x4(%rbp)

4004cf: cmpl $0x1,-0x4(%rbp)

4004d3: ja 4004dc <f+0x18>

4004d5: mov $0x1,%eax

4004da: jmp 40052d <f+0x69>

4004dc: mov -0x4(%rbp),%eax

4004df: and $0x1,%eax

4004e2: test %eax,%eax

4004e4: jne 4004f5 <f+0x31>

4004e6: mov 0x200440(%rip),%eax # 60092c <x.1604>

4004ec: add $0x1,%eax

4004ef: mov %eax,0x200437(%rip) # 60092c <x.1604>

4004f5: mov -0x4(%rbp),%eax

4004f8: and $0x1,%eax

4004fb: test %al,%al

4004fd: je 40050e <f+0x4a>

4004ff: mov 0x20042b(%rip),%eax # 600930 <y.1605>

400505: add $0x1,%eax

400508: mov %eax,0x200422(%rip) # 600930 <y.1605>

40050e: mov -0x4(%rbp),%eax

400511: sub $0x1,%eax

400514: mov %eax,%edi

400516: callq 4004c4 <f>

40051b: mov 0x20040f(%rip),%edx # 600930 <y.1605>

400521: lea (%rax,%rdx,1),%edx

400524: mov 0x200402(%rip),%eax # 60092c <x.1604>

40052a: lea (%rdx,%rax,1),%eax

40052d: leaveq

40052e: retq

**1）程序**

main()

{

unsigned int n;

for (n=1; n< 4; n++) {

printf("f(%d) = %x\n", n, f(n));

}

}

**的运行结果为：f(1)=1，f(2)=4e，f(3)=9f，请填写f函数所需要的内容（每空1分）：**

#define N (1)

#define M (2)

struct P1 {char c[N]; char \*d[N]; char e[N]; } P1;

struct P2 {int i[M]; char j[M]; short k[M]; } P2;

unsigned int f(unsigned int n)

{

(3) unsigned int x = sizeof(P1);

(4) unsigned int y = sizeof(P2);

if ( (5) )

return 1;

if ( (6) )

x++;

if ( (7) )

y++;

return (8) ;

}

2、程序

main()

{

printf("%x, %x\n", f(2), f(2));

}

的运行结果为：（2分）

1、答案：

#define N 3

#define M 5

struct P1 {char c[N]; char \*d[N]; char e[N]; } P1;

struct P2 {int i[M]; char j[M]; short k[M]; } P2;

unsigned int f(unsigned int n)

{

static unsigned int x = sizeof(P1);

static unsigned int y = sizeof(P2);

if (n<=1)

return 1;

if ((n & 1) == 0)

x++;

if ((n & 1) == 1)

y++;

return f(n-1) + (y) + (x);

}

1. **答案：4f, 4e （回答4e, 4f给一半的分）**

## 第五题（9分）



在“取指-译码-执行-写回”的四级流水线中，各流水级的工作内容和延迟如上图所示，寄存器的延迟也已标出。数据和指令分别存放在不同的存储器中。Cycle N写入寄存器文件的数据Cycle N+1才可读出。请问：

1）若不考虑流水线填充和清空时间，请计算该处理器的吞吐率。（1分）

2）若将该处理器改造为单周期处理器（SEQ），请计算SEQ处理器的吞吐率。（1分）

3）在上述流水线中，执行阶段包含了访问数据存储的时间。对于如下的Y86程序段，指令间存在哪些数据相关（dependence），会引起哪些数据冒险（hazard）？（5分）

Prog:

irmovl $128, %edx #instr1

irmovl $3, %ecx #instr2

rmmovl %ecx, 0(%edx) #instr3

irmovl $10, %ebx #instr4

mrmovl 0(%edx), %eax #instr5

addl %ebx, %eax #instr6

4）以上的数据冒险，可以通过转发（forward）的方法解决。请结合上述程序代码和流水线结构图逐个说明解决方案。（2分）

答案：

1）1000/(280 + 20) = 1000/300 = 3.33GIPS

2）1000/(80 +60 +280 + 60 +20) = 1000/ 500 = 2 GIPS

3）

Prog：

irmovl $128, %edx ; 1、dx= 128

irmovl $3, %ecx ; 2、cx= 3

rmmovl %ecx, 0(%edx) ; 3、[dx] = cx

irmovl $10, %ebx ; 4、bx= 10

mrmovl 0(%edx), %eax ; 5、ax= [dx]

addl %ebx, %eax ; 6、bx = bx + ax

相关：1-3，2-3, 1-5，4-6，5-6

冒险：1-3，2-3，4-6，5-6

4）2-3，5-6：执行到译码的转发通路解决；1-3,4-6：写回到译码的转发通路解决。

## 第七题（10分）

已知如下的汇编程序实现了函数transform(char\* src, char\* tgt, char delta)

transform:

jmp L2

L1:

add %edx, %eax

add $1, $rdi

mov %al, (%rsi)

add $1, $rsi

L2:

movzbl (%rdi), %eax

test %al, %al

jne L1

movb $0, (%rsi)

retq

参考信息：64位指令集中传递前三个参数分别使用寄存器%rdi, %rsi和%rdx

1）写出transform函数对应的C语言版本（2分）

2）假设读写访存指令延迟为20个时钟周期，其他指令延迟为2个时钟周期，所有分支预测都成功。同时CPU包含足够多的部件来实现指令集并行，那么在最理想情况下CPE最低应该是多少（2分）？为什么（2分）？

3）已知src对应字符串中每个字符c都满足0<c<=80且0<=delta<=5。通过下面的改写，可以把transform程序CPE的理论下限降低一半，请填空。假设程序运行在小端法机器上。（每空1分）

void transform(char\* src, char\* tgt, char delta) {  
 short x = \_\_\_\_\_\_\_\_\_\_\_\_\_\_;  
 while(\*src && \_\_\_\_\_\_\_\_\_\_\_\_) {  
 \*(short\*)tgt = \*(short\*)src + x;  
 src += 2;  
 tgt += 2;  
 }  
 \*(short\*)tgt = \_\_\_\_\_\_\_\_\_\_ ? \*(short\*)src + delta : \*(short\*)tgt &\_\_\_\_\_\_\_\_\_\_\_\_\_;  
}

答案：

1. {while(\*src) \*tgt++ = \*src++ + delta; \*tgt = 0;}
2. 42  
   每次循环的关键路径为 读内存、做加法、写内存，该路径需要42个时钟周期。  
   本题陷阱：同学可能会受书上的例子误导认为做加法可以和下一个时钟周期的读内存并行，使得CPE降到40，但实际上因为src和tgt指向的位置可能重叠，我们不能把下一次迭代的读操作移动到这一次的写操作之前。
3. (((short) delta) << 8) + delta  
   \*(src+1)  
   0xFF00  
   \*src

**2014**

## 第四题（10分）

一个函数如下，其中部分代码被隐去，请通过gdb调试信息补全代码（4分）。

int f(int n, int m) {

if (m > 0) {

if (n > 1) {

int r = f(n - 1, m);

return (r - 1 + m) % n + 1;

}

else if (n == 1) {

return 1;

}

}

return 0;

}

{考察点：x86-64函数调用、参数传递及栈的使用。函数调用通过rdi和rsi传递第一和第二个参数，栈中只记录函数的返回地址。由于是递归调用，需要用栈保存递归过程中参数变量的值。同时考察xor、test、lea、idiv、sete等指令的使用。难点1：两个判断“n > 1”和“n == 1”在汇编代码中只有一次比较，第二次判断相等是通过sete使得n==1时返回值为1，否则返回值为0实现的。难点2：表达式“(r - 1 + m) % n + 1”比较复杂，需要综合多条语句的信息才能分析出来，并且变量r对应于f(n-1,m)的返回值即寄存器%eax，因而不存在相应的赋值指令。}

如下是通过“gcc –g –O2”命令编译后，在gdb中通过“disas f”命令得到的反汇编代码，其中有两个汇编指令不全，请补全这两条汇编指令（2分）。

0x00000000004004e0 <f+0>: mov %rbx,-0x10(%rsp)

0x00000000004004e5 <f+5>: mov %rbp,-0x8(%rsp)

0x00000000004004ea <f+10>: xor %eax,%eax

0x00000000004004ec <f+12>: sub $0x10,%rsp

0x00000000004004f0 <f+16>: test %esi,%esi

0x00000000004004f2 <f+18>: mov %edi,%ebp

0x00000000004004f4 <f+20>: mov %esi,%ebx

0x00000000004004f6 <f+22>: jle 0x400513 <f+51>

0x00000000004004f8 <f+24>: cmp $0x1,%edi

0x00000000004004fb <f+27>: jle 0x400521 <f+65>

0x00000000004004fd <f+29>: lea -0x1(%rbp),%edi

0x0000000000400500 <f+32>: callq 0x4004e0 <f>

0x0000000000400505 <f+37>: lea -0x1(%rax,%rbx,1),%edx

0x0000000000400509 <f+41>: mov %edx,%eax

0x000000000040050b <f+43>: sar $0x1f,%edx

0x000000000040050e <f+46>: idiv %ebp

0x0000000000400510 <f+48>: lea 0x1(%rdx),%eax

0x0000000000400513 <f+51>: mov (%rsp),%rbx

0x0000000000400517 <f+55>: mov 0x8(%rsp),%rbp

0x000000000040051c <f+60>: add $0x10,%rsp

0x0000000000400520 <f+64>: retq

0x0000000000400521 <f+65>: sete %al

0x0000000000400524 <f+68>: movzbl %al,%eax

0x0000000000400527 <f+71>: jmp 0x400513 <f+51>

{考察点：函数中使用到了%rbp和%rbx寄存器，两者都是callee保存的寄存器，使用前需要压栈，函数返回时需要弹栈恢复寄存器的值。通过前后汇编代码的对比，应该可以猜出两个空分别填写什么；但要注意，压栈和弹栈时，%rsp寄存器的值不同，因而对应的地址表示也不同。}

已知在调用函数f(4, 3)时，我们在函数f中指令retq处设置了断点，下面列出的是程序在第一次运行到断点处暂停时时，相关通用寄存器的值。请根据你对函数及其汇编代码的理解，填写当前栈中的内容。如果某些内存位置处内容不确定，请填写X。（4分）

|  |  |
| --- | --- |
| 0x7fffffffe38c | X |
| 0x7fffffffe388 | X |
| 0x7fffffffe384 | X |
| 0x7fffffffe380 | X |
| 0x7fffffffe37c | X |
| 0x7fffffffe378 | X |
| 0x7fffffffe374 | 0x0 |
| 0x7fffffffe370 | 0x00400505 |
| 0x7fffffffe36c | 0x0 |
| 0x7fffffffe368 | 0x4 |
| 0x7fffffffe364 | 0x0 |
| 0x7fffffffe360 | 0x3 |
| 0x7fffffffe35c | 0x0 |
| 0x7fffffffe358 | 0x00400505 |
| 0x7fffffffe354 | 0x0 |
| 0x7fffffffe350 | 0x3 |
| 0x7fffffffe34c | 0x0 |
| 0x7fffffffe348 | 0x3 |
| 0x7fffffffe344 | 0x0 |
| 0x7fffffffe340 | 0x00400505 |
| 0x7fffffffe33c | 0x0 |
| 0x7fffffffe338 | 0x2 |
| 0x7fffffffe334 | 0x0 |
| 0x7fffffffe330 | 0x3 |
| 0x7fffffffe32c | X |
| 0x7fffffffe328 | X |
| 0x7fffffffe324 | X |
| 0x7fffffffe320 | X |

rax 0x1

rbx 0x3

rcx 0x3

rdx 0x309c552970

rsi 0x3

rdi 0x1

rbp 0x2

rsp 0x7fffffffe340

rip 0x400520

{考察点：递归调用的返回地址共三处是明确的，并且相同，值可以从反汇编代码中确定（1分）；三次递归调用程序栈中，压入的%rbx（m）的值不变，压入的%rbp（n）的值为每次减小1（1分）；注意x86-64，栈中的数据都是64位的，但因为数值均比较小，所以这9个位置处的高4字节均为0（1分）；其余位置的内容均是不确定的（1分）。}

**2015**

1. 给定一个实数，会因为该实数表示成单精度浮点数而发生误差。不考虑 NaN

和 Inf 的情况，该绝对误差的最大值为

A. 2 103

B. 2 104

C. 2 230

D. 2 231

参考信息：单精度浮点数阶码 8 位，尾数 23 位

答案：a。计算过程：浮点数阶码 8 位，即最大指数为 127，尾数为 23 位，即最

低位表示的数为2 −23 ∗ 2 127 = 2 104。那么误差在刚好落在最低位的一半的时候最大，

即2 103。其他选项为干扰项，如果求最大指数的时候忘记非规格化数或者对偶数舍

入理解不深会算出 b，如果以为指数为非规格化数会算出 c 或 d。

6. 下列寻址模式中，正确的是：

A. (%eax, , 4)

B. (%eax, %esp, 3)

C. 123 (这是之前没有注意到的)

D. $1(%ebx, %ebp, 1)

选择C. A 中不能省略Ei，B 中比例因子错误，D 中偏移地址表示错误。

7. 假设存储器按“大端法”存储数据对象，已知如下的

C 语言数据结构：union

{ char c[2]; int i; }; 当

c 的值为0x01, 0x23 时，i 的值为：

A. 0x0123

B. 0x2301

C. 0x01230000

D. 不确定

选择D. 因为

i 的两个高地址字节的值未知。

15. 下面关于存储器的说法，错误的是\_\_\_\_

A. SDRAM 的速度比 FPM DRAM 快

B. SDRAM 的 RAS 和 CAS 请求共享相同的地址引脚

C. 磁盘的寻道时间和旋转延迟大致在一个数量级

D. 固态硬盘的随机读写性能基本相当

答案：D

说明：本题考查学生对内存、硬盘基本特性的掌握。

17. 某高速缓存 （E=2, B=4, S=16），地址宽度 14，当引用地址 0x9D28 处的

1 个字节时，tag 位应为：

A. 01110100

B. 001110000

C. 1110000

D. 10011100

答案：A

说明：0x9D28=10*01110100*101000

第二题（20 分）

1.对于下面的每一个表达式，请选择以下选项中的一个或多个（即“不定项”），使

得该表达式恒成立，如果没有满足条件的选项则选 E。

A. < B. > C. == D. != E. none

题目中出现的变量定义如下（浮点数保证不是 NaN 或者 Inf）:

int x, y;

unsigned ux = x;

double d;

1) 如果 x > 0, 则 x + 1 0

2) 如果 x > y, 则 ux y

3) 如果 ((x << 31) >> 31) < 0, 则 x & 1 0

4) 如果((unsigned char)x >> 1) < 64， 则(char)x 0

5) 如果 d < 0， 则 d \* 2 0

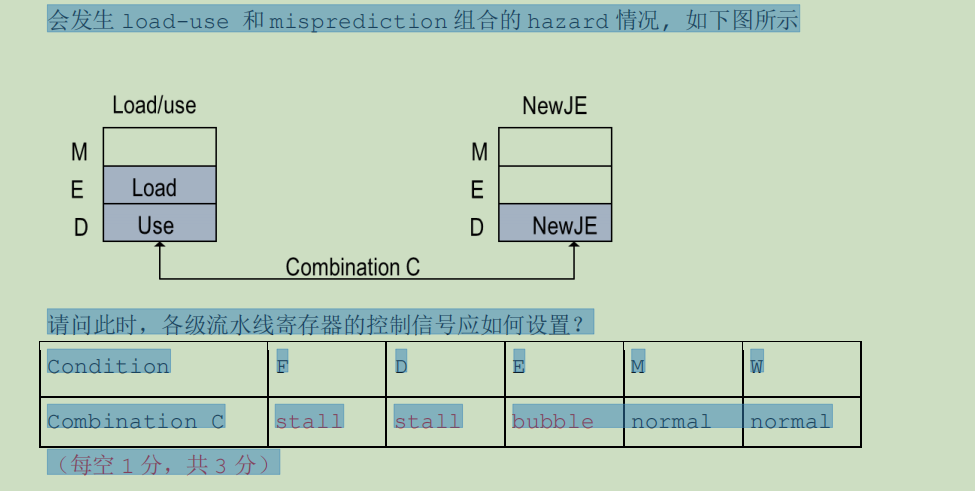
6) 如果 d < 0, 则 d \* d 0

7) x ˆ y ˆ (˜x) - y y ˆ x ˆ (˜y) - x

8) (((!!ux)) << 31) >> 31) (((!!x) << 31) >> 31)

（前 6 题每题一分，漏选错选不得分; 7、8 两题每题 2 分，漏选得 1 分;共 10

分）

****

**2016**

4. 对于如下的 C 语言中的条件转移指令，它所对应的汇编代码中至少包含几条条

件转移指令：if (a > 0 && a != 1 || a < 0 && a != -1) b=a;

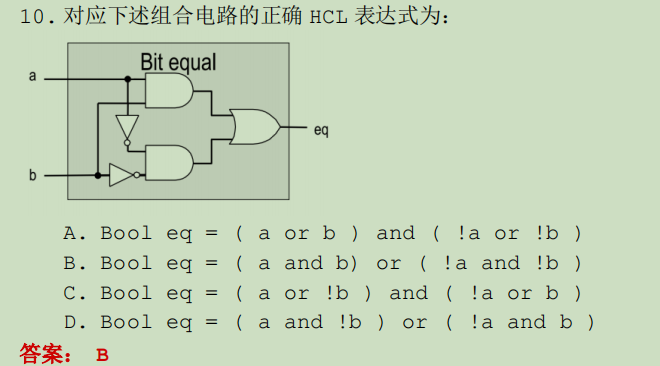
A. 2 条

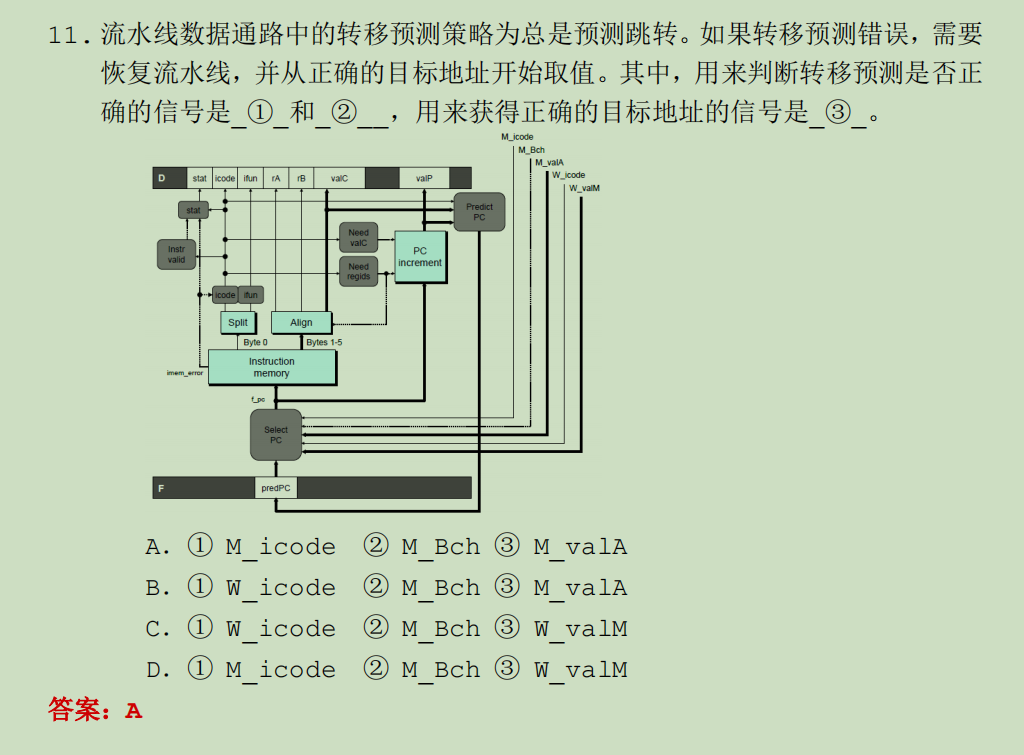
B. 3 条

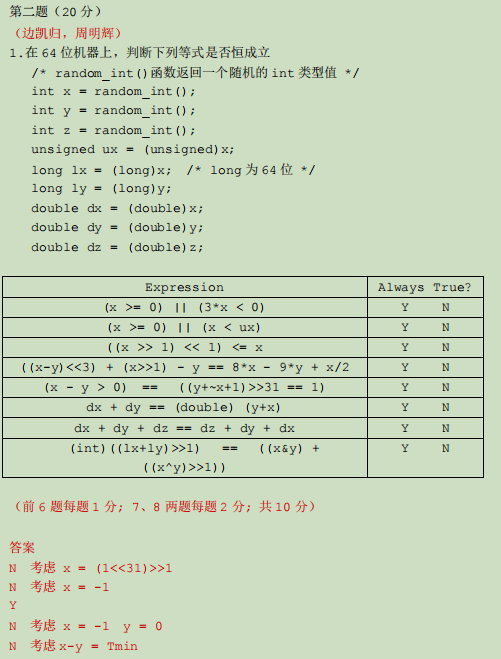
C. 4 条

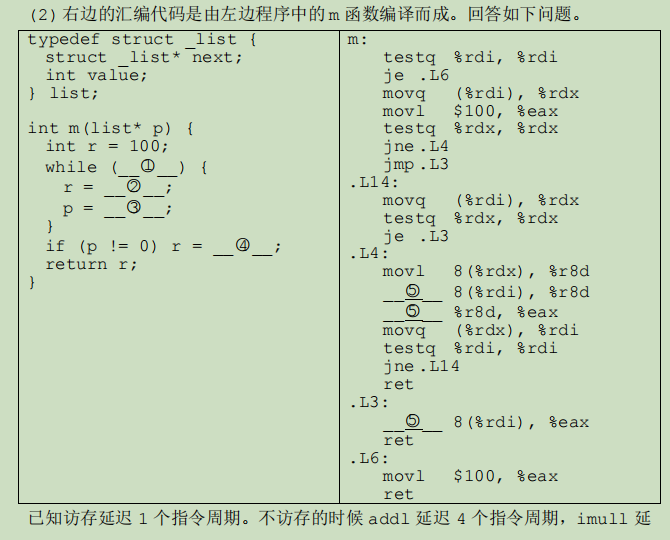
D. 5 条

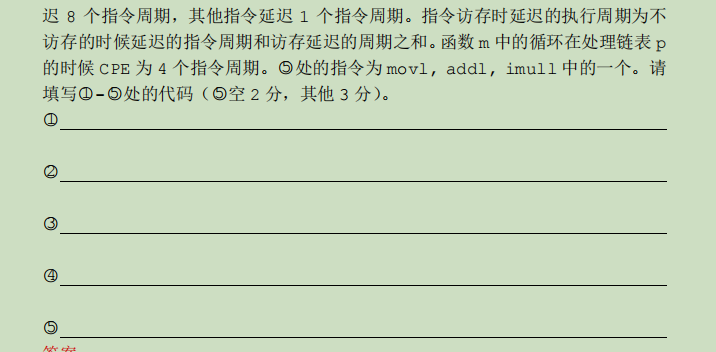
答案：B（这个条件相当于!(a==0 || a==1 || a==-1)）

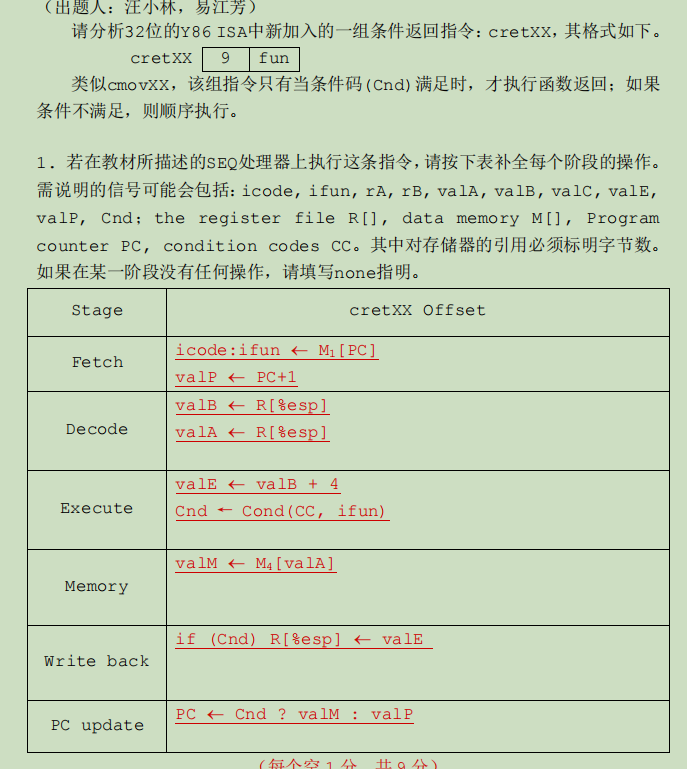




****







2017

1. 有如下代码段：

int func(int x, int y);

int (\*p) (int a, int b);

p = func;

p(0,0);

对应的下列 x86-64 过程调用正确的是：

A. call \*%rax

B. call (%rax)

C. call \*(%rax)

D. call func

答案：A/D 都对

解析：用 O0 编译可以得到 A，用 Og 编译可以得到 D。

7. 有 A 的定义：int A[3][2] = {{1,2}，{3,3}，{2,1}}；

那么 A[2]的值为：

A. &A+16

B. A+16

C. \*A+4

D. \*A+2

答案：C

解析：参见书 P177 页表格，机器在计算指针与常数的运算时，会将常数乘以指针

指向的元素大小。&A 常数扩大的倍数为 sizeof(A[3][2]) = 3\*2\*4; A 常数

扩大的倍数为 sizeof(A[0])=2\*4; \*A 常数扩大的倍数为 sizeof(int) = 4。

正确的答案应为 A+2 或\*A+4，故应选择 C

14. 关于局部性（locality）的描述，正确的是：

A. 数据的时间局部性或数据空间局部性，在任何有意义的程序中都能体现

B. 指令的时间局部性或数据空间局部性，在任何有意义的程序中都能体现

C. 数据的时间局部性，在任何循环操作中都能体现

D. 数据的空间局部性，在任何数组操作中都能体现

答案：B

注：B 选项有笔误（“数据空间局部性”应为“指令空间局部性”），所以所有答案

都不对。按任何选项都对计分。

2018

2. 在采用小端法存储机器上运行下面的代码，输出的结果将会是？

（int，unsigned 为 32 位长，short 为 16 位长，0~9 的 ASCII 码分别是 0x30~0x39）

char \*s = "2018";

int \*p1 = (int \*)s;

short s1 = (\*p1)>>12;

unsigned u1 = (unsigned) s1;

printf("0x%x\n",u1);

A) 0x00002303 B) 0x00032303 C) 0xffff8313 D) 0x00008313

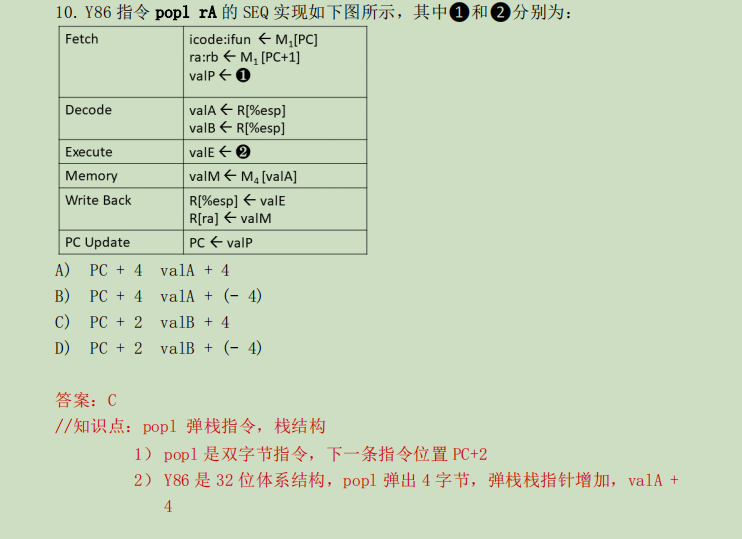
答案：C

本题考查大小端存储，以及整数类型转化。字符串在存储时不区分大小端，前面的

字符存储在低地址，而在被转化成整型的时候低地址被视为低位，因此\*p1 为

0x38313032，s1 为 0x8313。在由 short 转化为 unsigned 的时候，我们要先改变大

小，之后完成有符号到无符号的转换，因此 u1 为



2020

7、下列说法正确的是：C

A. 在SEQ机器中，我们采用预测跳转总是选择（always taken）的策略比从不

选择（never taken）的策略要略好。

B. 流水级划分应尽量均衡，不均衡的流水线会增加控制冒险。

C. 如果一台机器的CPI小于1，则它必然不是普通流水线结构。

D. 由于rrmovq %rax, %rax不影响标记位，所以可使用其代替nop指令。

选项A，SEQ机器无所谓转移预测

选项D，rrmovq会带来潜在的数据相关