2022 计算机系统期末模拟题

ywy_c_asm

免责声明：皆为原创，仅作参考，如有雷同，绝非泄题！

一、单项选择题（每小题 1 分，共 20 分）

1.（ ）是主存和 I/O 设备的抽象表示。

A.操作系统 B.文件 C.虚拟内存 D.进程

2.整数 0x1234567 的低 10 个二进制位构成的整数为（ ）。

A.0x567 B.0x267 C.359 D.361

3.设寄存器 rcx 的初值为 0x1145141919810，执行汇编指令“movw $0x233, %cx”后，rcx 的值为_____，接下来再执行汇编指令“movl $0x19260817, %ecx”后，rcx 的值为_____。

A.0x1145100000233 0x19260817

B.0x1145141919233 0x19260817

C.0x1145141910233 0x19260817

D.0x1145141910233 0x1145119260817

4.编译器常常使用汇编指令“test %rax, %rax”以及条件转移指令来判断寄存器 rax 存的整数在有符号意义下是否为负数，这本质上是利用了 CPU 中的（ ）标志位。

A.SF B.OF C.CF D.ZF

5.在 Y86-64 的顺序实现中，指令（ ）可能会使用到寄存器文件的两个写端口。

A.pushq B.rrmovq C.popq D.call

6.CPU 在访问单核私有的 L1-Cache 时使用的是_____地址，在访问单核私有的 L1-TLB 时使用的是_____地址，在访问所有核共享的 L3-Cache 时使用的是_____地址。

A.物理 物理 虚拟 B.物理 虚拟 虚拟

C.物理 物理 物理 D.物理 虚拟 物理

7.CPU 中集成的 L2-Cache 属于（ ）类型的存储器。

A.DRAM B.SRAM C.EEPROM D.EPROM

8.小 Y 写了一个由两个 C 语言源文件 a.c 和 b.c 构成的程序，这是他的代码：

a.c

```c
double x=233;

void func(int f);

int main(){
    func(1);
    return 0;
}
```

b.c

```c
#include<stdio.h>

int x=666;

void func(int y){
    printf("x+y=%d\n",x+y);
}
```

现在小 Y 准备编译链接他的代码，并运行生成的可执行文件。请你预测一下会发生什么。

A.发生编译错误，无法生成可执行文件。

B.编译成功，但发生链接错误，无法生成可执行文件。

C.可执行文件生成成功，但执行时发生运行时错误，程序意外终止。

D.可执行文件生成成功且运行成功，程序输出“x+y=667”。

9.下列哪项特点不是 Intel Core i7 CPU 所具有的（ ）。

A.具有足够精简的指令集，功耗低，因而在嵌入式领域中广受欢迎。

B.采取超标量技术，使得一个周期内平均执行多个操作成为可能。

C.采取四级页表，并使用速度较高的 TLB 缓存页表条目。

D.支持 SIMD 指令，可以在一条指令中同时操作多个数据。

10.在 C 语言程序中，当数组越界访问时，可能会（ ）。

A.程序仍然能正常运行，无事发生。 B.异常处理程序被执行。

C.当前进程接收到信号。 D.以上都可能会发生。

11.下列哪个不属于进程上下文（ ）。

A.寄存器 rsp 的值 B.CF 标志位 C..text 节 D.进程打开文件的信息

12.关于浮点数，下列说法正确的是（ ）。

A.浮点数-0 是非规格化浮点数。

B.C 语言类型 int 和 float 在内存中所有可能存储的值，int 更多。

C.将一个 double 变量除 0 会导致除 0 故障，进而导致当前进程被终止。

D.浮点数规定的非数 NaN 有两种机器值，+NaN 和-NaN。

13.关于 fork，下列说法正确的是（ ）。

A.它属于异常中的陷阱，并且和 execve 有不同的异常号，属于不同的系统调用。

B.fork 在执行时，为了提高空间效率，会让父子进程共享相同的代码页面。但由于父子进程可能会对数据区作出不同的修改，所以出于保守起见会为子进程直接复制并创建新的数据页面。

C.通过 fork，新创建的子进程可以得知自己的 pid。

D.如果一段程序中连续调用了两次 fork，那么 fork 实际调用的次数可能会多于两次。

14.下列哪种信号的默认行为不会使得进程终止（ ）。

A.SIGCHLD B.SIGINT C.SIGSEGV D.SIGFPE

15.编写信号处理程序时，以下哪种行为是不安全的（ ）。

A.为了避免不易察觉而危险的数据竞争，在操作全局数据结构时，尽量使用 printf 输出调试信息，表明信号处理程序目前正在进行的操作。

B.在操作全局数据结构时，暂时阻塞掉其它可能造成数据竞争的信号，作为保护。

C.在信号处理程序中尽量使用较为简单的操作。

D.尽量不使用信号处理程序对信号进行计数。

16.关于分页机制，下列说法正确的是（ ）。

A.在基于 Intel Core i7 的计算机系统中，物理页的起始地址可以为 8 的倍数，也可以为奇数。

B.Intel Core i7 为了优化地址翻译，当系统内核在设置好页表后，可以通过 CPU 提供的一条特殊指令将整个页表直接加载到处理器的某个寄存器中。

C.在基于 Intel Core i7 的计算机系统中，物理页大小不一定是 4KB，但虚拟页大小有可能是 4KB。

D.为了在分页机制的基础上作出更加灵活的内存操作，用户进程可以指定自己的缺页故障处理程序，就像使用 signal 指定信号处理程序以作出更加灵活的信号处理一样。

17.以下哪条指令在最坏情况下可以有最多的访问存储器次数（ ）。

A. movq %rax, (%rbx)

B. subq %rax, (%rbx)

C. leaq (%rbx,%rcx,4), %rax

D. cmpq %rax, (%rbx)

18.关于缓冲区溢出攻击，下列说法错误的是（ ）。

A.它的原因之一是程序员使用了不安全的未检查缓冲区越界的函数，我们可以通过使用更安全的函数来避免，例如将不安全的 fgets 替换为更安全的 gets。

B.结合虚拟内存以及分页机制，操作系统可以在数据段虚拟页对应的 PTE 中标记该页为禁止执行，防止攻击者在数据段中插入可执行的攻击代码。

C.它的原理是攻击者修改了栈中函数的返回地址。

D.为了防范缓冲区溢出攻击，可以采用基于“金丝雀”的栈破坏检测。

19.关于缓存的不命中，下列说法错误的是（ ）。

A.直接映射高速缓存可能会产生冲突不命中，也可能会产生冷不命中。

B.CPU 在执行内存访问时，可能会涉及到三类缓存的不命中。

C.缓存的不命中既可以完全由硬件处理，也可以由硬件和软件联合处理。

D.虚拟内存作为缓存时，可能会产生冲突不命中。

20.下列 C 语言条件表达式为假的是（ ）。

A.1u > 0　　B.1 > 0　　C.-1 < 0　　D.-1 < 0u

二、填空题（每空 1 分，共 10 分）

21.float 变量 1.0 在内存中从低地址到高地址的字节为_____（十六进制表示）。

22.下列代码定义了一个 struct，它的 sizeof 为_____。

```
struct treenode{
    short val;
    struct data* leftson;
    char ch;
    int sum;
    int size;
    struct data* rightson;
};
```

23.调用一次，可返回超过一次的用户级函数是_____。
24.x86-64 下 C 语言函数调用的第 4 个参数用寄存器_____传递。
25.printf(“Hello World\n”)中使用的字符串在运行前存储在可执行文件的_____节，属于_____段。
26.用户在 shell 中运行程序的时候，输入的命令行参数可以通过 main 的参数_____接收。
27.随着程序用于引用数据的循环的步长增加，程序的_____局部性下降。
28.用户可以在 shell 中按下组合键_____使前台作业停止。
29.某 8 路组相联高速缓存采用 32 位物理地址，总容量为 128Kb，15 位标记位，那么每行的缓存块大小为_____字节。

三、判断题（每小题 1 分，共 10 分，在题前打√×符号）

30.（ ）系统调用 execve 调用后一定不会返回。
31.（ ）在 int 与 float 互相转换的过程中，一定会发生位模式的改变。
32.（ ）在 Y86-64 流水线结构设计中，可以使用转发避免数据冒险。
33.（ ）CPU 为 PTE 提供了专门的缓存 TLB，因而 PTE 只能被 TLB 缓存。
34.（ ）在对固态硬盘进行写操作时，需要先擦除整个页，再向页中写入数据。
35.（ ）循环展开级数越多，指令的并行程度越好，执行时间上的优化也就越明显。
36.（ ）进程在进行系统调用时，可能会发生上下文切换，但不是所有系统调用都会引发上下文切换。
37.（ ）链接器通过重定位关联模块之间的符号定义与符号引用。
38.（ ）存储器层次结构中速度最快的是寄存器。
39.（ ）指令“subq $2, %rax”和“leaq -2(%rax), %rax”的效果是完全一致的。

四、简答题（每小题 10 分，共 30 分）

40.结合你在本课程中的所学知识，简述为什么 C 语言中未赋初值的全局或静态变量在运行时会被自动初始化为 0，而未赋初值的局部变量的值将会是不确定的。

41.现有一大小为 32KB 的直接映射高速缓存，每块大小 128 字节，初始时缓存为空，依次对 6 个 32 位物理地址进行访问：①0x80，②0x0，③0xA8，④0x80E1，⑤0x58058，⑥0xA80C0。在什么时候会缓存命中？什么时候会产生冲突不命中？什么时候会产生冷不命中？请写出详细的分析过程。

42.小 Y 在学完 C 语言的指针后，突发奇想，写出了一个奇怪的程序 me.c：

```c
#include<stdio.h>

const int a=233;

int main(){
    int* ptr=&a;
    *ptr=666;
    printf("a=%d\n",a);
    return 0;
}
```

在这个程序中，他试图通过指针对于 int 型全局**常量** a 进行修改，期望这样能够突破 C 语言对于“**常量不可被修改**”的限制。这个程序确实能够通过编译（尽管编译器抛出了一些警告），但运行时会出错，屏幕显示“**段错误（核心已转储）**”。请你结合链接、异常控制流、虚拟内存等方面的知识，分析这个程序为什么会得到这样的运行结果。

五、系统分析题（每小题 10 分，共 30 分）

43.小 Y 写了一个 C 语言程序：

```c
#include<stdio.h>
#include<stdlib.h>

int a=233;
int b;

int main(){
    char* str="guiheyange";
    int* p=malloc(sizeof(int));
    static int c;
    int d=777;
    return 0;
}
```

已知这个程序没有使用编译器优化，不使用寄存器变量。考虑这个程序中涉及到的几个对象：①变量 a，②变量 b，③变量 str，④字符串“guigeyange”，⑤变量 p，⑥p 指向的对象（即 *p），⑦变量 c，⑧变量 d，⑨整数 777。对它们进行分类：

(1)强符号：_____

(2)弱符号：_____

(3)局部符号：_____

(4)全局符号：_____

(5)存储在栈上：_____

(6)存储在堆上：_____

(7)存储在.text 节：_____

(8)存储在.data 节：_____

(9)存储在.rodata 节：_____

(10)存储在.bss 节：_____

44.小 Y 写了一个 C 语言双参数函数 sum，功能为求一个整数数组的和。以下是它的反汇编代码：

```
1   sum:
2       mov     $0x0,%eax
3       mov     $0x0,%edx
4   label_1157:
5       cmp     %edi,%eax
6       jge     label_1166
7       mov     %eax,%ecx
8       add     (%rsi,%rcx,4),%edx
9       add     $0x1,%eax
10      jmp     label_1157
11  label_1166:
12      mov     %edx,%eax
13      retq
```

这个函数的参数 2 的类型为_____，返回值的类型为_____，第 6 行 jge 指令的转移条件为_____（用标志位表示），解释第 8 行指令的含义与作用_____。

45.某计算机系统的部分参数如下表所示：

名称	参数
页大小	4KB
L1 数据 Cache 容量	64KB
L1 数据 Cache 块大小	64Bytes
L1 数据 Cache 路数	8-way
L1 数据 TLB 容量	512PTEs
L1 数据 TLB 路数	8-way
物理地址	58 位
虚拟地址	54 位

不考虑其它的 Cache 与 TLB，写出下列地址相关参数的位数：

VA_____ PA_____ VPO_____

PPO_____ VPN_____ PPN_____

CI_____ CT_____ CO_____

TLBI_____ TLBT_____

六、综合设计题（每小题 10 分，共 10 分）

46.请你为 Y86-64 添加两条通过寄存器获得目的地址的转移指令 `jrXX rA` 和 `callr rA`，它们的编码格式与功能为：

指令	**功能**	**字节 1**		**字节 2**	
`jrXX rA`	条件转移，目的地址为在寄存器 rA 中	`0xC`	`fn`	`rA`	`0xF`
`callr rA`	过程调用，目的地址在寄存器 rA 中	`0xD`	`0`	`rA`	`0xF`

在下表中设计它们的微操作：

步骤	`jrXX rA`	`callr rA`
取指		
译码		
执行		
访存		
写回		
更新 PC		

答案解析

一、单项选择题（每小题 1 分，共 20 分）

1.（ C ）是主存和 I/O 设备的抽象表示。

A.操作系统 B.文件 C.虚拟内存 D.进程

解析：教材上第一章的图：

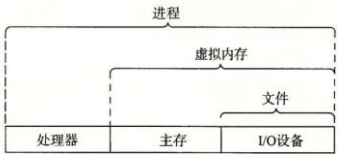


图 1-11 操作系统提供的抽象表示

2.整数 0x1234567 的低 10 个二进制位构成的整数为（ C ）。

A.0x567 B.0x267 C.359 D.361

解析：0x1234567 的低 12 位为 0x567=010101100111，低 10 位为 0101100111=0x167=359

3.设寄存器 rcx 的初值为 0x1145141919810，执行汇编指令“movw $0x233，%cx”后，rcx 的值为_____，接下来再执行汇编指令“movl $0x19260817，%ecx”后，rcx 的值为_____。C

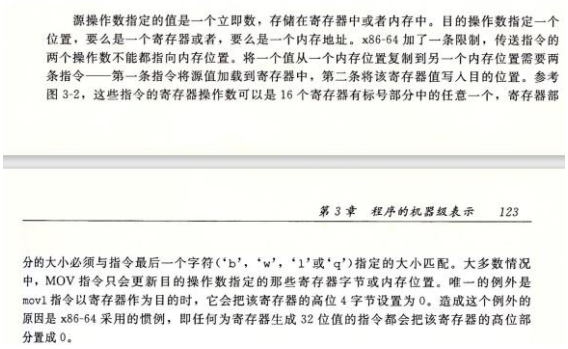
A.0x1145100000233 0x19260817

B.0x1145141919233 0x19260817

C.0x1145141910233 0x19260817

D.0x1145141910233 0x1145119260817

解析：第一条指令设置 cx，相当于直接将 rcx 的低 16 位（也就是低 4 个十六进制位）改为 0x0233，此时 rcx=0x1145141910233，第二条指令设置 ecx，根据 x86-64 的规定，rcx 的高 32 位会自动清零，因此 rcx=ecx=0x19260817。



源操作数指定的值是一个立即数，存储在寄存器中或者内存中。目的操作数指定一个位置，要么是一个寄存器或者，要么是一个内存地址。x86-64 加了一条限制，传送指令的两个操作数不能都指向内存位置。将一个值从一个内存位置复制到另一个内存位置需要两条指令——第一条指令将源值加载到寄存器中，第二条将该寄存器值写入目的位置。参考图 3-2，这些指令的寄存器操作数可以是 16 个寄存器有标号部分中的任意一个，寄存器部

分的大小必须与指令最后一个字符（‘b’，‘w’，‘l’或‘q’）指定的大小匹配。大多数情况中，MOV 指令只会更新目的操作数指定的那些寄存器字节或内存位置。唯一的例外是 movl 指令以寄存器作为目的时，它会把该寄存器的高位 4 字节设置为 0。造成这个例外的原因是 x86-64 采用的惯例，即任何为寄存器生成 32 位值的指令都会把该寄存器的高位部分置成 0。

4.编译器常常使用汇编指令“test %rax，%rax”以及条件转移指令来判断寄存器 rax 存的整数在有符号意义下是否为负数，这本质上是利用了 CPU 中的（ A ）标志位。

A.SF B.OF C.CF D.ZF

解析：如果自己真的是负数，那么符号位是 1，test 自己，得到的结果还是自己，并通过自己设置符号位，这样就能将 SF 置 1，之后就可以用 js 指令转移了。详见教材 P135。

5.在 Y86-64 的顺序实现中，指令（ C ）可能会使用到寄存器文件的两个写端口。

A.pushq B.rrmovq C.popq D.call

解析：popq rA 在写回阶段时，会同时对 rsp 和 rA 进行写入操作，需要两个寄存器文件的写端口。pushq 和 call（注意 PC 不属于寄存器文件）只需要写 rsp，rrmovq 只需要写 rB。详见教材 P266~270。

6.CPU 在访问单核私有的 L1-Cache 时使用的是_____地址，在访问单核私有的 L1-TLB 时使用的是_____地址，在访问所有核共享的 L3-Cache 时使用的是_____地址。D

A.物理 物理 虚拟 B.物理 虚拟 虚拟

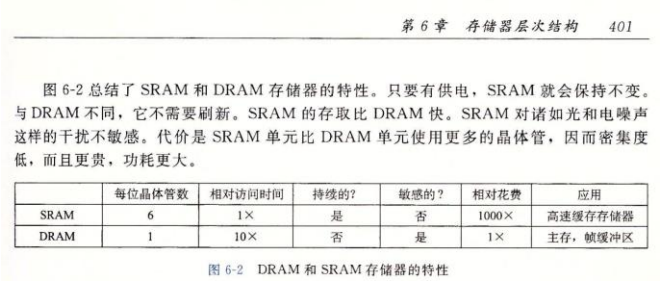
C.物理 物理 物理 D.物理 虚拟 物理

解析：不管 L 几 Cache，都要用物理地址访问，TLB 需要在地址翻译时用虚拟地址访问。经典问题。详见教材 P570。

7.CPU 中集成的 L2-Cache 属于（ B ）类型的存储器。

A.DRAM B.SRAM C.EEPROM D.EPROM

解析：SRAM 速度比 DRAM 高，DRAM 成本比 SRAM 低，因此 SRAM 被用来作高速缓存，DRAM 被用来作主存。



第 6 章 存储器层次结构 401

图 6-2 总结了 SRAM 和 DRAM 存储器的特性。只要有供电，SRAM 就会保持不变。与 DRAM 不同，它不需要刷新。SRAM 的存取比 DRAM 快。SRAM 对诸如光和电噪声这样的干扰不敏感。代价是 SRAM 单元比 DRAM 单元使用更多的晶体管，因而密集度低，而且更贵，功耗更大。

	每位晶体管数	相对访问时间	持续的?	敏感的?	相对花费	应用
SRAM	6	1×	是	否	1000×	高速缓存存储器
DRAM	1	10×	否	是	1×	主存，帧缓冲区

图 6-2 DRAM 和 SRAM 存储器的特性

8.小 Y 写了一个由两个 C 语言源文件 a.c 和 b.c 构成的程序，这是他的代码：

a.c

```c
double x=233;

void func(int f);

int main(){
    func(1);
    return 0;
}
```

b.c

```c
#include<stdio.h>

int x=666;

void func(int y){
    printf("x+y=%d\n",x+y);
}
```

现在小 Y 准备编译链接他的代码，并运行生成的可执行文件。请你预测一下会发生什么。

A.发生编译错误，无法生成可执行文件。

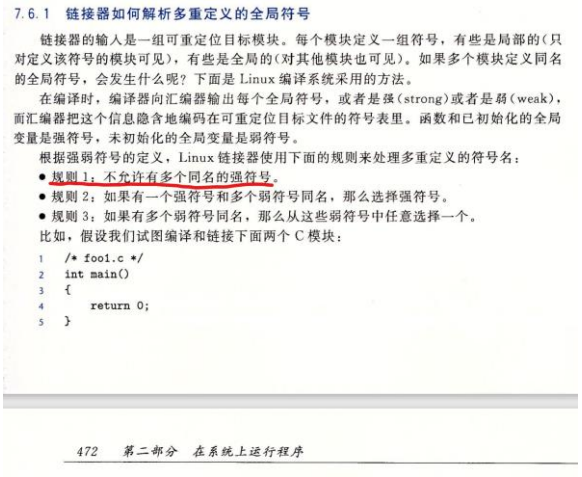
B.编译成功，但发生链接错误，无法生成可执行文件。

C.可执行文件生成成功，但执行时发生运行时错误，程序意外终止。

D.可执行文件生成成功且运行成功，程序输出“x+y=667”。

解析：这两个模块中都声明了已初始化的全局变量 x，属于名称相同的强全局符号，链接器规定不能出现这种情况，会报错。尽管它们类型不同，但编译后，它们就仅仅是个符号或者地址，不再具有类型的属性，链接器也不知道。

有的同学可能会问，模块 1 里声明的 func 的参数名叫 f，为什么它跟实际的 func 的参数名 y 不同，这也不会出错？实际上仅仅是声明函数的话，参数名叫什么是没有意义的，只需要知道这里有一个 int 参数就行了，能够让调用者知道如何传入参数，就算不写参数名也没事……



7.6.1 链接器如何解析多重定义的全局符号

链接器的输入是一组可重定位目标模块。每个模块定义一组符号，有些是局部的（只对定义该符号的模块可见），有些是全局的（对其他模块也可见）。如果多个模块定义同名的全局符号，会发生什么呢？下面是 Linux 编译系统采用的方法。

在编译时，编译器向汇编器输出每个全局符号，或者是强（strong）或者是弱（weak），而汇编器把这个信息隐含地编码在可重定位目标文件的符号表里。函数和已初始化的全局变量是强符号，未初始化的全局变量是弱符号。

根据强弱符号的定义，Linux 链接器使用下面的规则来处理多重定义的符号名：

- 规则 1：不允许有多个同名的强符号。
- 规则 2：如果有一个强符号和多个弱符号同名，那么选择强符号。
- 规则 3：如果有多个弱符号同名，那么从这些弱符号中任意选择一个。

比如，假设我们试图编译和链接下面两个 C 模块：

```
1  /* foo1.c */
2  int main()
3  {
4      return 0;
5  }
```

472 第二部分 在系统上运行程序

9.下列哪项特点不是 Intel Core i7 CPU 所具有的（ A ）。

A.具有足够精简的指令集，功耗低，因而在嵌入式领域中广受欢迎。

B.采取超标量技术，使得一个周期内平均执行多个操作成为可能。

C.采取四级页表，并使用速度较高的 TLB 缓存页表条目。

D.支持 SIMD 指令，可以在一条指令中同时操作多个数据。

解析：众所周知 Intel 的 x86 CPU 都是典型的复杂指令集 CPU，A 选项是以 arm 为代表的 RISC CPU 的特点。关于 RISC 与 CISC 的概念和区别参见教材 P249~250 旁注。

B 选项参见教材 P357 下方，这个以前也被考过。

C 选项参见教材 P576 关于 Core i7 地址翻译的具体讲解，这里的东西必考，务必记住。

D 选项参见教材 P376 关于向量指令的旁注，这也是一种典型的程序优化方式。

10.在 C 语言程序中，当数组越界访问时，可能会（ D ）。

A.程序仍然能正常运行，无事发生。 B.异常处理程序被执行。

C.当前进程接收到信号。 D.以上都可能会发生。

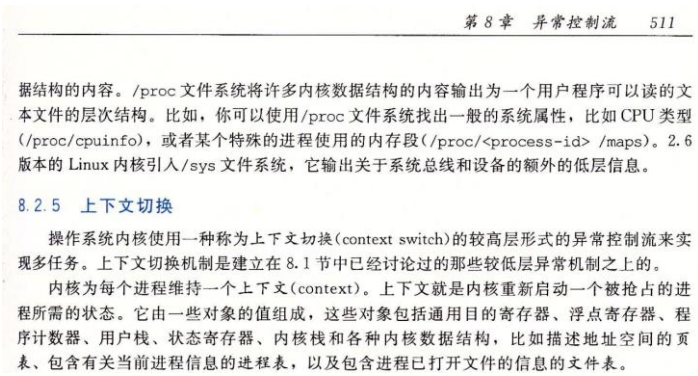
解析：当数组越界越的不太过分的时候，A 选项是可能的，比如局部数组越界，可能仅仅修改了栈帧中其它变量的值，再比如缓冲区溢出攻击，这也不会立刻出现非法的内存访问。对于 BC，如果数组越界越的比较过分，访问了非法的虚拟地址，在地址翻译的时候 MMU 就会触发一般保护故障，从而使得内核的故障处理程序执行，对进程发送 SIGSEGV 信号，默认使得进程终止。这个数组越界的例子曾经被出在系统分析题中。

关于非法内存访问引起的一般保护故障，参见教材 P567。关于 SIGSEGV 信号，参见教材 P527 上方。

11.下列哪个不属于进程上下文（ C ）。

A.寄存器 rsp 的值 B.CF 标志位 C..text 节 D.进程打开文件的信息

解析：数据段和代码段的内容都不算进程上下文，考过好多遍了……



第 8 章 异常控制流 511

据结构的内容。/proc 文件系统将许多内核数据结构的内容输出为一个用户程序可以读的文本文件的层次结构。比如，你可以使用/proc 文件系统找出一般的系统属性，比如 CPU 类型(/proc/cpuinfo)，或者某个特殊的进程使用的内存段(/proc/<process-id> /maps)。2.6 版本的 Linux 内核引入/sys 文件系统，它输出关于系统总线和设备的额外的低层信息。

8.2.5 上下文切换

操作系统内核使用一种称为上下文切换(context switch)的较高层形式的异常控制流来实现多任务。上下文切换机制是建立在 8.1 节中已经讨论过的那些较低层异常机制之上的。

内核为每个进程维持一个上下文(context)。上下文就是内核重新启动一个被抢占的进程所需的状态。它由一些对象的值组成，这些对象包括通用目的寄存器、浮点寄存器、程序计数器、用户栈、状态寄存器、内核栈和各种内核数据结构，比如描述地址空间的页表、包含有关当前进程信息的进程表，以及包含进程已打开文件的信息的文件表。

12.关于浮点数，下列说法正确的是（ A ）。

A.浮点数-0 是非规格化浮点数。

B.C 语言类型 int 和 float 在内存中所有可能存储的值，int 更多。

C.将一个 double 变量除 0 会导致除 0 故障，进而导致当前进程被终止。

D.浮点数规定的非数 NaN 有两种机器值，+NaN 和-NaN。

解析：±0 都是非规格化数，规格化数无法表示 0。如果说能表示的数的多少那肯定是 int 更多，但如果说存储的机器值，int 和 float 都是 32 位数据，两者都有 2^32 种机器值，一样多。浮点除 0 不会发生除 0 故障，整数才会，浮点除 0 会得到 inf 的结果。NaN 的机器值有很多很多种，只要阶码字段全 1 且尾数字段非 0 即可。D 选项是无穷 inf 的性质。

13.关于 fork，下列说法正确的是（ D ）。

A.它属于异常中的陷阱，并且和 execve 有不同的异常号，属于不同的系统调用。

B.fork 在执行时，为了提高空间效率，会让父子进程共享相同的代码页面。但由于父子进程可能会对数据区作出不同的修改，所以出于保守起见会为子进程直接复制并创建新的数据页面。

C.通过 fork，新创建的子进程可以得知自己的 pid。

D.如果一段程序中连续调用了两次 fork，那么 fork 实际调用的次数可能会多于两次。

解析：A 选项注意区分系统功能号和异常号，fork 和 execve 都对应异常号为 0x80 的系统调用，只是功能号不同。参见教材 P506。

B 选项 fork 不会直接为子进程复制新的页面，会采取懒惰的写时复制策略使得二者暂时共享所有页面，需要对其正确理解。参见教材 P584。

C 选项子进程仅仅通过 fork 是无法知道自己的 pid 的，需要在后面使用 getpid。但父进程通过 fork 可以知道子进程的 pid。

D 选项的例子就是嵌套的 fork，比如这段代码中 fork 可以被调用 3 次：

```
int main(){
    fork();
    fork();
}
```

这个经典例子参见教材 P515。

14.下列哪种信号的默认行为不会使得进程终止（ A ）。

A.SIGCHLD B.SIGINT C.SIGSEGV D.SIGFPE

解析：参见教材 P527 的表中信号的默认行为。SIGCHLD 属于少数默认忽略的信号之一。

15.编写信号处理程序时，以下哪种行为是不安全或者不合适的（ A ）。

A.为了避免不易察觉而危险的数据竞争，在操作全局数据结构时，尽量使用 printf 输出调试信息，表明信号处理程序目前正在进行的操作。

B.在操作全局数据结构时，暂时阻塞掉其它可能造成数据竞争的信号，作为保护。

C.在信号处理程序中尽量使用较为简单的操作。

D.尽量不使用信号处理程序对信号进行计数。

解析：printf 不属于异步信号安全的函数，不被建议在异常处理程序中使用。参见教材 P534~535。BC 是教材 P534 提到的典型的安全的信号处理方式。D 是经典结论，信号不可排队，不可计数，如果真的使用信号处理程序进行信号计数，在处理程序被执行时突然来了多个信号，那么只会被视作一个。

16.关于分页机制，下列说法正确的是（ C ）。

A.在基于 Intel Core i7 的计算机系统中，物理页的起始地址可以为 8 的倍数，也可以为奇数。

B.Intel Core i7 为了优化地址翻译，当系统内核在设置好页表后，可以通过 CPU 提供的一条特殊指令将整个页表直接加载到处理器的某个寄存器中。

C.在基于 Intel Core i7 的计算机系统中，物理页大小不一定是 4KB，但虚拟页大小有可能是 4KB。

D.为了在分页机制的基础上作出更加灵活的内存操作，用户进程可以指定自己的缺页故障处理程序，就像使用 signal 指定信号处理程序以作出更加灵活的信号处理一样。

解析：见教材 P576 关于 Core i7 分页与地址翻译的具体讲解，可以设置页大小为 4KB 或 4MB，不过一般设置为 4KB。A 选项参见教材 P578，页的起始地址必须对齐，必须为页大小的倍数（2^12 或 2^22），不可能是奇数。B 选项是我瞎编的，但错的也很离谱，毕竟，CPU 哪有那么大的寄存器能装下整个页表……CPU 只提供了保存页表基地址的寄存器 CR3，这个参见 P576 下方。D 选项很显然用户不能指定异常处理程序，那属于操作系统内核的程序。

17.以下哪条指令在最坏情况下可以有最多的访问存储器次数（ B ）。

A. movq %rax, (%rbx)

B. subq %rax, (%rbx)

C. leaq (%rbx,%rcx,4), %rax

D. cmpq %rax, (%rbx)

解析：这几条指令中，C 不会访问内存（lea 的常见误区，仅仅是地址计算），A 和 D 仅仅会读/写一次内存单元，B 不仅要先读内存单元，计算出结果还需要写回去。这题其实跟地址翻译/Cache 那的过程没太大关系，考的是对汇编指令的理解。

18.关于缓冲区溢出攻击，下列说法错误的是（ A ）。

A.它的原因之一是程序员使用了不安全的未检查缓冲区越界的函数，我们可以通过使用更安全的函数来避免，例如将不安全的 fgets 替换为更安全的 gets。

B.结合虚拟内存以及分页机制，操作系统可以在数据段虚拟页对应的 PTE 中标记该页为禁止执行，防止攻击者在数据段中插入可执行的攻击代码。

C.它的原理是攻击者修改了栈中函数的返回地址。

D.为了防范缓冲区溢出攻击，可以采用基于“金丝雀”的栈破坏检测。

解析：A 选项说的其实是对的，但偷换了概念，应该是把不安全的 gets 替换为更安全的 fgets，见教材 P196，CD 选项都是这里关于缓冲区溢出攻击的基本知识。B 选项参见教材 P579 上方，这是典型的操作系统管理内存访问权限的方式。

19.关于缓存的不命中，下列说法错误的是（ D ）。

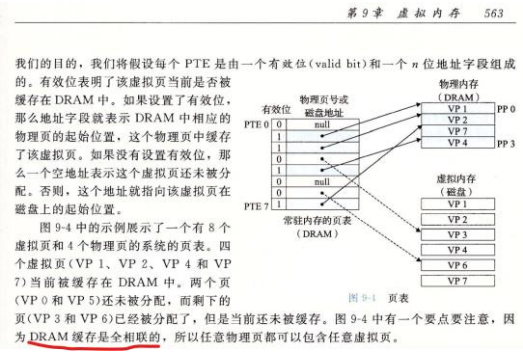
A.直接映射高速缓存可能会产生冲突不命中，也可能会产生冷不命中。

B.CPU 在执行内存访问时，可能会涉及到三类缓存的不命中。

C.缓存的不命中既可以完全由硬件处理，也可以由硬件和软件联合处理。

D.虚拟内存作为缓存时，可能会产生冲突不命中。

解析：D 选项由于虚拟内存作为缓存时，DRAM 主存相当于全相联缓存（因为通过分页机制可以在理论上让任意一个虚拟页对应任意一个物理页，而不是直接映射），不会冲突不命中。



第 9 章 虚拟内存 563

我们的目的，我们将假设每个 PTE 是由一个有效位（valid bit）和一个 n 位地址字段组成的。有效位表明了该虚拟页当前是否被缓存在 DRAM 中。如果设置了有效位，那么地址字段就表示 DRAM 中相应的物理页的起始位置，这个物理页中缓存了该虚拟页。如果没有设置有效位，那么一个空地址表示这个虚拟页还未被分配。否则，这个地址就指向该虚拟页在磁盘上的起始位置。

图 9-4 中的示例展示了一个有 8 个虚拟页和 4 个物理页的系统的页表。四个虚拟页（VP 1、VP 2、VP 4 和 VP 7）当前被缓存在 DRAM 中。两个页（VP 0 和 VP 5）还未被分配，而剩下的页（VP 3 和 VP 6）已经被分配了，但是当前还未被缓存。图 9-4 中有一个要点要注意，因为 DRAM 缓存是全相联的，所以任意物理页都可以包含任意虚拟页。

图 9-4 页表

A 选项显然，参见教材 P423 的不命中种类以及 P427 对于直接映射高速缓存的讲解和示例。B 选项涉及到的三类缓存分别是 TLB、Cache 和缓存物理页的 DRAM 主存。C 选项中 Cache 是纯粹的硬件处理机制，而作为缓存的虚拟内存在缺页（页不命中）时则是由地址翻译硬件+操作系统内核联合处理的机制，属于硬件+软件。

20.下列 C 语言条件表达式为假的是（ D ）。

A.1u > 0　　B.1 > 0　　C.-1 < 0　　D.-1 < 0u

解析：经典的无符号比较涉及到的问题，D 选项的右操作数 0u 是无符号整数，比较为无符号比较，左操作数-1 被解读为无符号数 2^32-1，鉴定为假。

二、填空题（每空 1 分，共 10 分）

21.float 变量 1.0 在内存中从低地址到高地址的字节为 00 00 80 3F（十六进制表示）。
解析：较为简单的浮点数机器值计算，1.0 的尾数就是 1.0，对应在机器值中的字段就是全 0（23 个二进制位 0），阶码为 0，加上偏置为 127，机器值 exp 为 8 位二进制数 01111111，因此二进制表示为 00111111100000000000000000000000，每 8 位对应一个字节，就是 00111111 10000000 00000000 00000000，即十六进制 3F 80 00 00，小端序下从低到高为 00 00 80 3F。

22.下列代码定义了一个 struct，它的 sizeof 为 40 。

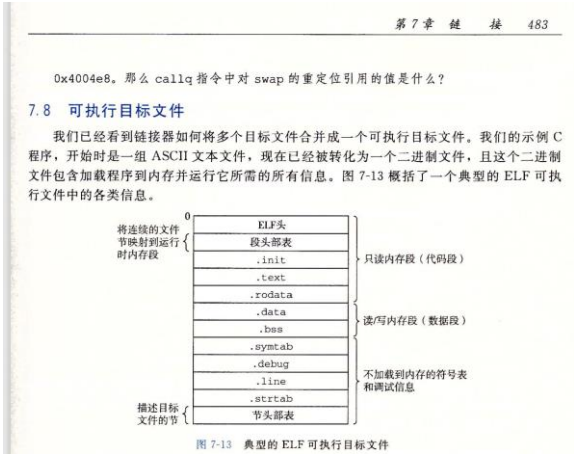
```
struct treenode{
    short val;
    struct data* leftson;
    char ch;
    int sum;
    int size;
    struct data* rightson;
};
```

解析：val 的偏移地址为 0，leftson 为 8 字节指针，偏移地址对齐到 8，ch 的偏移地址为 16，sum 为 4 字节数据，偏移地址对齐到 20，size 的偏移地址为 24，rightson 偏移地址对齐到 32，因此 sizeof(struct treenode)=40。

23.调用一次，可返回超过一次的用户级函数是 setjmp 。
解析：fork 虽然也可以返回超过一次，但它属于系统调用。setjmp 属于用户级函数。

24.x86-64 下 C 语言函数调用的第 4 个参数用寄存器 rcx 传递。
解析：参见教材 P120 关于各个通用寄存器在过程调用中的含义。

25.printf(“Hello World\n”)中使用的字符串在运行前存储在可执行文件的 .rodata 节，属于 代码 段。（或者只读内存段）
解析：这个字符串常量存在.rodata 节中，参见教材 P467 对于各个节的讲解。但要区分节和段，而.rodata 节属于只读的代码段。



26.用户在 shell 中运行程序时，输入的命令行参数可以通过 main 的参数_argv_接收。
解析：基本知识，参见教材 P521 关于 execve 的讲解。

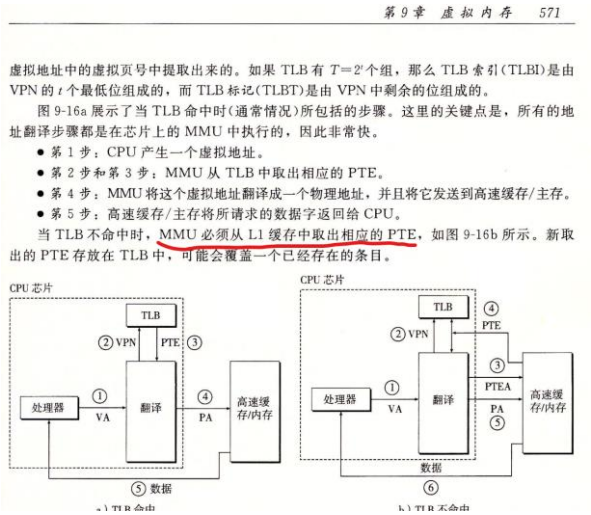
27.随着程序用于引用数据的循环的步长增加，程序的_空间_局部性下降。
解析：参见教材 P418~P420 对于局部性的具体讲解，这是影响空间局部性最主要的因素。

28.用户可以在 shell 中按下组合键_Ctrl+Z_使前台作业停止。
解析：基本知识，并且要区分“终止”和“停止”，参见教材 P529 下方。

29.某 8 路组相联高速缓存采用 32 位物理地址，总容量为 128Kb，15 位标记位，那么每行的缓存块大小为_64_字节。
解析：注意 Kb 和 KB 的区别!!!! 这个以前考试题坑过!!! 小写的 b 表示二进制位，所以 128Kb=16KB=2^14 字节，那么组数就是 2^14/8=2^11，因此地址中的组索引为 11 位，那么物理地址中的块偏移就有 32-15-11=6 位，因此块大小为 2^6=64 字节。
总而言之这种题需要熟悉教材 P434 关于组相联高速缓存关于物理地址的位级结构解析，考试的时候一般都这样给你其它一堆参数，让你用类似方式推出另一个参数，需要知道这东西怎么算出来的。

三、判断题（每小题 1 分，共 10 分，在题前打√×符号）

30.（ × ）系统调用 execve 调用后一定不会返回。
解析：“调用后不返回”只是在调用成功的情况下而言的，如果调用失败，那么会返回。什么你说这个考的太偏难怪？这个 execve 失败返回的例子可是在咱们的 LAB4 的核心代码里出现过的……
31.（ × ）在 int 与 float 互相转换的过程中，一定会发生位模式的改变。
解析：绝大多数情况下肯定要发生改变，但也有反例，比如 0，有意思的是浮点数中的 0（+0）和整数一样，在机器中也是一堆二进制位 0。
32.（ √ ）在 Y86-64 流水线结构设计中，可以使用转发避免数据冒险。
解析：参见教材 P300 关于转发避免数据冒险的讲解。
33.（ × ）CPU 为 PTE 提供了专门的缓存 TLB，因而 PTE 只能被 TLB 缓存。
解析：当 TLB 不命中的时候，需要去主存中查询页表，这个时候也会被 Cache 缓存。



34.（ × ）在对固态硬盘进行写操作时，需要先擦除整个页，再向页中写入数据。
解析：（这个知识点以前被考过）

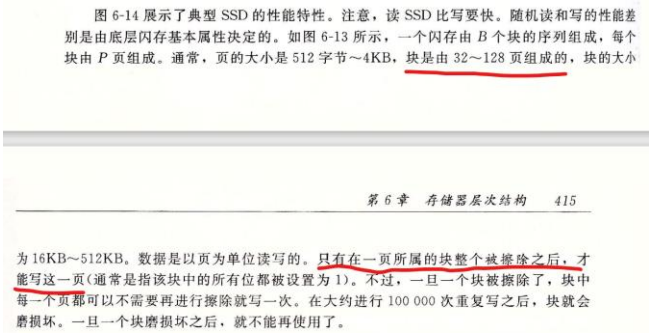


图 6-14 展示了典型 SSD 的性能特性。注意，读 SSD 比写要快。随机读和写的性能差别是由底层闪存基本属性决定的。如图 6-13 所示，一个闪存由 B 个块的序列组成，每个块由 P 页组成。通常，页的大小是 512 字节～4KB，块是由 32～128 页组成的，块的大小

第 6 章　存储器层次结构　415

为 16KB～512KB。数据是以页为单位读写的。只有在一页所属的块整个被擦除之后，才能写这一页（通常是指该块中的所有位都被设置为 1）。不过，一旦一个块被擦除了，块中每一个页都可以不需要再进行擦除就写一次。在大约进行 100 000 次重复写之后，块就会磨损坏。一旦一个块磨损坏之后，就不能再使用了。

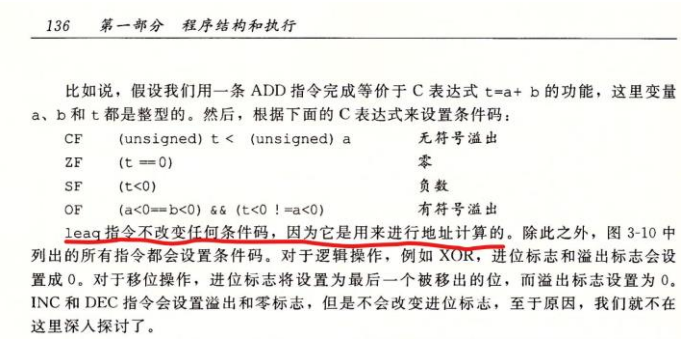
35.（ × ）循环展开级数越多，指令的并行程度越好，执行时间上的优化也就越明显。
解析：理论上循环展开确实越多越好，但实际上，过多的循环展开可能会因过多的局部变量造成寄存器溢出，编译器将不得不把变量分配到栈上，导致不必要的内存访问，反而会拖慢速度。对此教材 P378 有更加详细的例子讲解。

36.（ √ ）进程在进行系统调用时，可能会发生上下文切换，但不是所有系统调用都会引发上下文切换。
解析：参见教材 P511 关于上下文切换的解释，上下文切换必须调度下一个进程，如果当前进程仅简单地进行了系统调用，系统调用立刻返回该进程，那么这不算上下文切换。

37.（ × ）链接器通过重定位关联模块之间的符号定义与符号引用。
解析：这个是符号解析做的事情，参见教材 P470 关于符号解析的概念以及 P478 关于重定位的概念。

38.（ √ ）存储器层次结构中速度最快的是寄存器。
解析：基本知识，参见教材 P421 关于存储器层次结构的概念。

39.（ × ）指令“subq $2, %rax”和“leaq -2(%rax), %rax”的效果是完全一致的。
解析：表面上看，它们都将 rax-=2，但 lea 不会设置标志位，它由地址加法器运算，而 add 由 ALU 执行，这是基本常识。



136　第一部分　程序结构和执行

比如说，假设我们用一条 ADD 指令完成等价于 C 表达式 t=a+ b 的功能，这里变量 a、b 和 t 都是整型的。然后，根据下面的 C 表达式来设置条件码：

CF	(unsigned) t < (unsigned) a	无符号溢出
ZF	(t ==0)	零
SF	(t<0)	负数
OF	(a<0==b<0) && (t<0 !=a<0)	有符号溢出

leaq 指令不改变任何条件码，因为它是用来进行地址计算的。除此之外，图 3-10 中列出的所有指令都会设置条件码。对于逻辑操作，例如 XOR，进位标志和溢出标志会设置成 0。对于移位操作，进位标志将设置为最后一个被移出的位，而溢出标志设置为 0。INC 和 DEC 指令会设置溢出和零标志，但是不会改变进位标志，至于原因，我们就不在这里深入探讨了。

四、简答题（每小题 10 分，共 30 分）

40.结合你在本课程中的所学知识，简述为什么 C 语言中未赋初值的全局或静态变量在运行时会被自动初始化为 0，而未赋初值的局部变量的值将会是不确定的。
C 语言中未赋初值的全局/静态变量对应.bss 节中的区域，在可执行文件被加载时，.bss 节被映射到匿名文件，对应的页面会被内核填充二进制 0，表现为它们在运行时的初值被自动置 0。而局部变量在运行时存在栈上，赋初值需要通过显式的机器指令完成，若不赋初值，那么函数调用时仅仅会为其在栈上分配空间，但由于栈中之前的数据是不确定的，故局部变量的初值也是不确定的。　（.bss 相关参见教材 P467 与 P585）

41.现有一大小为 32KB 的直接映射高速缓存，每块大小 128 字节，初始时缓存为空，依次对 6 个 32 位物理地址进行访问：①0x80，②0x0，③0xA8，④0x80E1，⑤0x58058，⑥0xA80C0。在什么时候会缓存命中？什么时候会产生冲突不命中？什么时候会产生冷不命中？请写出详细的分析过程。

该缓存的组数为 32KB/128B=256=2^8，而 128=2^7，故物理地址的 0~6 位为 7 位块偏移，7~14 位为 8 位组索引，15~31 位为 17 位标记。对这些地址分析二进制位结构，那么：

①0x80：组索引=1，标记=0，发生冷不命中，加载块(0,1)。

②0x0：组索引=0，标记=0，发生冷不命中，加载块(0,0)。

③0xA8：组索引=1，标记=0，命中缓存块(0,1)。

④0x80E1：组索引=1，标记=1，发生冲突不命中，加载块(1,1)，替换(0,1)。

⑤0x58058：组索引=0，标记=11，发生冲突不命中，加载块(11,0)，替换(0,0)。

⑥0xA80C0：组索引=1，标记=21，发生冲突不命中，加载块(21,1)，替换(1,1)。

（参见教材 P427 关于直接映射高速缓存的概念和示例）

42.小 Y 在学完 C 语言的指针后，突发奇想，写出了一个奇怪的程序 me.c：

```c
#include<stdio.h>

const int a=233;

int main(){
    int* ptr=&a;
    *ptr=666;
    printf("a=%d\n",a);
    return 0;
}
```

在这个程序中，他试图通过指针对于 int 型全局**常量** a 进行修改，期望这样能够突破 C 语言对于“**常量不可被修改**”的限制。这个程序确实能够通过编译（尽管编译器抛出了一些警告），但运行时会出错，屏幕显示“**段错误（核心已转储）**”。请你结合链接、异常控制流、虚拟内存等方面的知识，分析这个程序为什么会得到这样的运行结果。

C 语言的常量在编译链接后存在.rodata 节内，程序加载时，代码段的一部分被映射到.rodata 节的数据，这一段区域对应的虚拟页面被内核在 PTE 上标记为只读。当程序试图通过指针修改这个常量时，相当于对代码段进行写操作，指令执行时，MMU 在地址翻译的过程中通过 PTE 发现程序正在尝试对只读页面进行写操作，属于非法的内存访问，引发一般保护故障，内核的故障处理程序对当前进程发送 SIGSEGV 信号，进行将进程终止并显示“段错误（核心已转储）”的默认操作。

（这一典型机制参见教材 P567 关于“虚拟内存作为内存保护的工具”的讲解，SIGSEGV 参见教材 P527 上方）

五、系统分析题（每小题 5 分，共 30 分）

43.小 Y 写了一个 C 语言程序：

```c
#include<stdio.h>
#include<stdlib.h>

int a=233;
int b;

int main(){
    char* str="guiheyange";
    int* p=malloc(sizeof(int));
    static int c;
    int d=777;
    return 0;
}
```

已知这个程序没有使用编译器优化，不使用寄存器变量。考虑这个程序中涉及到的几个对象：①变量 a，②变量 b，③变量 str，④字符串“guigeyange”，⑤变量 p，⑥p 指向的对象（即 *p），⑦变量 c，⑧变量 d，⑨整数 777。对它们进行分类：需要把第七章的各种符号搞清

(1)强符号：①

(2)弱符号：②

(3)局部符号：⑦

(4)全局符号：①②

(5)存储在栈上：③⑤⑧

(6)存储在堆上：⑥

(7)存储在.text 节：⑨（属于指令中的立即操作数）

(8)存储在.data 节：①

(9)存储在.rodata 节：④

(10)存储在.bss 节：②⑦

44.小 Y 写了一个 C 语言双参数函数 sum，功能为求一个整数数组的和。以下是它的反汇编代码：

```
1  sum:
2      mov    $0x0,%eax
3      mov    $0x0,%edx
4  label_1157:
5      cmp    %edi,%eax
6      jge    label_1166
7      mov    %eax,%ecx
8      add    (%rsi,%rcx,4),%edx
9      add    $0x1,%eax
10     jmp    label_1157
11 label_1166:
12     mov    %edx,%eax
13     retq
```

这个函数的参数 2 的类型为_int*/int[]_，返回值的类型为_int/unsigned int_，第 6 行 jge 指令的转移条件为_~(SF^OF)（或 SF 与 OF 相同）_（用标志位表示），解释第 8 行指令的含义与作用_将 rsi+rcx*4 地址处的 32 位整数累加到 edx 中，相当于将基址为 rsi 的 int 数组 arr 的下标为 rcx 的整数累加到 edx 中_。

解析：第一个空的关键在于分析第 8 行中 rsi 的行为，第二个空是因为第 12 行通过 eax 传递返回值，第 3 个空参见教材 P139 关于条件转移指令的讲解，这是有符号>=转移，第 4 个空参见教材 P176 数组访问的讲解，应该也属于基本常识了。

45.某计算机系统的部分参数如下表所示：

名称	参数
页大小	4KB
L1 数据 Cache 容量	64KB
L1 数据 Cache 块大小	64Bytes
L1 数据 Cache 路数	8-way
L1 数据 TLB 容量	512PTEs
L1 数据 TLB 路数	8-way
物理地址	58 位
虚拟地址	54 位

不考虑其它的 Cache 与 TLB，写出下列地址相关参数的位数：

VA___54_____ PA___58_____ VPO___12_____

PPO__12_____ VPN___42_____ PPN___46_____

CI___7_____ CT___45_____ CO___6_____

TLBI_6_____ TLBT__36_____

解析：VA 是虚拟地址，54 位，PA 是物理地址，58 位。VPO 是虚拟页偏移，12 位（页大小 4KB=2^12B）。PPO 是物理页偏移 12 位（等于 VPO）。VPN 是虚拟页号，为虚拟地址除去 VPO 后的高位部分，即 54-12=42 位。PPN 是物理页号，为物理地址除去 PPO 后的高位部分，58-12=46 位。CO 为 Cache 块偏移，6 位（块大小 64B=2^6B）。CI 为 Cache 组索引，Cache 的组数=64KB/(64B*8way)=128=2^7，故 CI 有 7 位。那么物理地址中剩下的 58-7-6=45 位就是 Cache 标记 CT。TLB 的组数=512PTEs/8way=64=2^6，故 VPN 中的 TLB 组索引 TLBI=6 位，剩下的部分 42-6=36 位为 TLB 标记 TLBT。这些缩写考试的时候可能会直接给出，应该都得认识。

六、综合设计题（每小题 10 分，共 10 分）

46.请你为 Y86-64 添加两条通过寄存器获得目的地址的转移指令 jrXX rA 和 callr rA，它们的编码格式与功能为：

指令	功能	字节 1		字节 2	
jrXX rA	条件转移，目的地址为在寄存器 rA 中	0xC	fn	rA	0xF
callr rA	过程调用，目的地址在寄存器 rA 中	0xD	0	rA	0xF

在下表中设计它们的微操作：

步骤	jrXX rA	callr rA
取指	icode:ifun←M_1[PC] rA:rB←M_1[PC+1] valP←PC+2	icode:ifun←M_1[PC] rA:rB←M_1[PC+1] valP←PC+2
译码	valA←R[rA]	valA←R[rA] valB←R[%rsp]
执行	Cnd←Cond(CC,ifun)	valE←valB+(-8)
访存		M_8[valE]←valP
写回		R[%rsp]←valE
更新 PC	PC←Cnd?valA:valP	PC←valA

可仿照教材 P270 给出的 call 和 jXX 的微操作。