第三次作业

作业发布时间：2023/05/29 星期一
本次作业要求如下：
1.截止日期：2023/06/11 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“第三次作业+学号+姓名”， **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 存储器层次结构+虚拟内存

一．填空题

1. 对于一个磁盘，其平均旋转速率是 15000 RPM，平均寻道时间是 4ms，单个磁道上平均扇区数量是 800，则这个磁盘的平均访问时间是 __6.005ms____

2. 对于一个磁盘，其有两个扇片，10000 个柱面，每个磁道平均有 400 个扇区，而每个扇区平均有 512 个字节。那么这个磁盘的的容量为 ___8GB____

3. Cache 为 8 路的 2M 容量，B=64，则其 Cache 组的位数 s= __12_____

4. 在现代计算机存储层次体系中，访问速度最快的是 __寄存器____

5. 若高速缓存的块大小为 B(B>8)字节，向量 v 的元素为 int，则对 v 的步长为 1 的应用的不命中率为__4/B___

6. 某CPU使用32位虚拟地址和4KB大小的页时，需要PTE的数量是 ___2^{20}____（不考虑多级页表情况）

7. 缓存不命中的种类有 __冷不命中____ 、 __冲突不命中_____、 ___容量不命中_____ 。

8. 虚拟页面的状态有 ___未分配_____、已缓存、未缓存共 3 种。

9. Linux 虚拟内存区域可以映射到普通文件和 __匿名文件_____，这两种类型的对象中的一种。

10. 虚拟内存发生缺页时，缺页中断是由 ____MMU____ 触发的。

11. Linux 缺页异常可能的几种原因分别是 ___访问非法内存、在只读段写入、访问 Swap 分区的页_____ 。

二．分析题

1. 假设 1MB 的文件由平均大小为 512 bytes 的逻辑块组成。磁盘的转速为 10000 RPM，平均寻道时间是 5ms，每个磁道上的平均扇区数量为 1000，扇面大小为 4，单个扇区大小为 512 bytes。那么访问这个文件最好的时间和最坏

的时间分别是多少?

平均旋转延迟 $= \frac{1}{2} \times \left(\frac{60}{10000}\right) \times 1000 = 3ms$

数据传输时间 $= \left(\frac{60}{10000}\right) \times 1000 \times \left(\frac{1}{1000}\right) = 0.006ms$/扇区

1MB 的文件需要使用 2000 个扇区。

最好的情况：不需要旋转，传输 1000 扇区+寻道+传输 1000 个扇区，即 $2000 \times 0.006 + 5 = 12 + 5 = 17ms.$

最坏的情况：文件随机分布在不同的扇区，时间为$2000 \times (5 + 3 + 0.006)ms = 16.012s.$

2. 请解释什么是直接内存访问（DMA）。并简述 CPU 发起磁盘读的三条存储指令。

DMA 是一种硬件机制，它允许外围设备和主存之间直接传输 I/O 数据，而不需要系统处理器的参与。使用这种机制可以大大提高与设备通信的吞吐量。

三条存储指令：

1. 发起读请求，包含了有关参数
2. 指明读取的硬盘逻辑块号
3. 指明读取数据对应的内存地址

3. 对于一个机器而言，有如下的假设，内存是字节寻址，并且内存访问是 1 字节的字。地址宽度是 13 位，其高速缓存是 2 路组相联的，块大小是 4 字节，一共有 8 个组。高速缓存的具体内容如图所示。

2路组相联高速缓存

	行0						行1					
组索引	标记位	有效位	字节0	字节1	字节2	字节3	标记位	有效位	字节0	字节1	字节2	字节3
0	09	1	86	30	3F	10	00	0	—	—	—	—
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	—	—	—	—	0B	0	—	—	—	—
3	06	0	—	—	—	—	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	—	—	—	—
6	91	1	A0	B7	26	2D	F0	0	—	—	—	—
7	46	0	—	—	—	—	DE	1	12	C0	88	37

a) 对于该地址格式进行划分，划分出 tag 位，组索引位和块内偏移的区间。

b) 对于地址 0x0E34 ，指出其对应的 tag 位，组索引位和块内偏移的值，并说明高速缓存是否命中，如果命中，写出对应的字节（用 16 进制表示）。

c) 对于地址 0x0DD5，指出其对应的 tag 位，组索引位和块内偏移的值，并说明高速缓存是否命中，如果命中，写出对应的字节（用 16 进制表示）。

a). 如下所示。

Tag	组索引位	块内偏移
12~5	4~2	1~0

b). tag 位是 0x71，组索引为 0x101（5），块内偏移为 0. 高速缓存命中，对应字节为 0x0B。

c). tag 位是 0x6e，组索引为 0x101（5），块内偏移为 1. 高速缓存未命中。

4. 对于一个直接映射的高速缓存系统，假设其大小是 256 字节，块大小是 16 字节，现在定义三个操作，L 为装载操作，S 为数据存储操作，M 为数据更改操作。L 和 S 最多引发一次缓存 miss，而 M 操作可以看作是对于同一个地址先进行了 L 操作，之后进行了 S 操作。分析下面的操作序列，对于高速缓存的

的命中和淘汰情况。（假设高速缓存最开始是空的）

说明：L 10，1 表示对于地址 0x10 位置，进行了一个字节的装载操作。

依题意，组索引有 4 位，0～15. 块内偏移也有 4 位。

L 10，1

组索引 1，块内偏移 0. 缓存未命中，存入组 1.

M 20，1

组索引 2，块内偏移 0. L 操作时缓存未命中，存入组 2，S 操作时缓存命中。

L 22，1

组索引 2，块内偏移 2. 缓存命中。

S 18，1

组索引 1，块内偏移 8. 缓存未命中。

L 110，1

组索引 1，块内偏移 0. 缓存未命中，淘汰原组 1.

L 210，1

组索引 1，块内偏移 0. 缓存未命中，淘汰原组 1.

M 12，1

组索引 1，块内偏移 2. L 操作时缓存未命中，淘汰原组 1，S 操作时缓存命中。

5. 对于著名的存储器山形状图，分析读吞吐量产生如此变化趋势的原因，说明几条山脊和斜坡出现的原因。

由空间局部性原理，步长越小，访问速度越快，吞吐量越大。
由时间局部性原理，工作集越小，固定时间内访问工作集的次数越多，吞吐量越大。
山脊对应的是工作集恰好在 L1 Cache, L2 Cache, L3 Cache 的情况。
斜坡说明步长增大后空间局部性变差，吞吐量随着步长的增大而下降。

6. 在一台具有块大小 16 字节（B=16）、整个大小为 1024 字节的直接映射数据缓存的机器上测量如下代码的高速缓存性能：

假设：

- sizeof(int) = 4。
- grid 从内存地址 0 开始。
- 这个高速缓存开始时是空的。
- 唯一的内存访问是对数组 grid 的元素的访问，变量 i、j、total_x 和 total_y 存放在寄存器中。
- 数据结构定义

```
struct position {
    int x;
    int y;
};

struct position grid[16][16];

int total_x = 0, total_y = 0;
int i, j;
```

A. Test 1

```
for (i = 0; i < 16; i++){
    for(j = 0; j < 16; j++){
        total_x += grid[i][j].x;
    }
}

for (i = 0; i < 16; i++){
    for(j = 0; j < 16; j++){
        total_y += grid[i][j].y;
    }
}
```

1. 读总数是多少?
2. 缓存不命中的读总数是多少?
3. 不命中率是多少?

B. Test 2

```
for (i = 0; i < 16; i++){
    for(j = 0; j < 16; j++){
        total_x += grid[i][j].x;
        total_y += grid[i][j].y;
    }
```

}

1. 读总数是多少?
2. 缓存不命中的读总数是多少?
3. 不命中率是多少?
4. 如果高速缓存有两倍大，那么不命中率是多少?

A. 1. 读总数为 2*16*16=512.

2. 不命中读总数是 256.

3. 不命中率是 50%

B. 1. 读总数是 512.

2. 不命中读总数是 128.

3. 不命中率是 25%

4. 仍然是 25%。

7. 对于一个地址，高速缓存通常使用地址的中间部分作为组索引，为什么不用高位地址作为组索引?

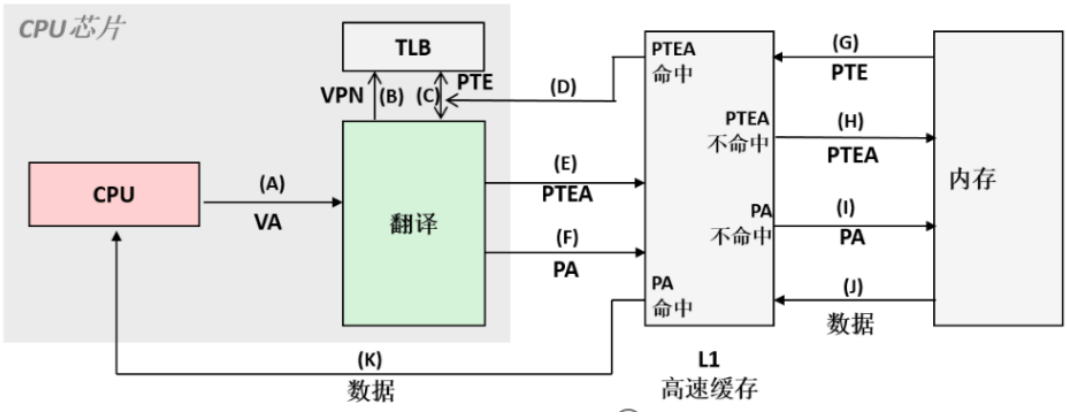
如果使用高位地址作为索引，相邻的内存会被映射到同一块 Cache，在局部顺序访问时，只能利用到一个 cache 块，并且会发生频繁的块替换，降低命中率。

8. 下图展示了一个虚拟地址的访存过程，每个步骤都用不同的字母表示。请针对下面不同的情况，用字母序列表示不同情况下的执行流程。

（1）TLB 命中，缓存物理地址命中。

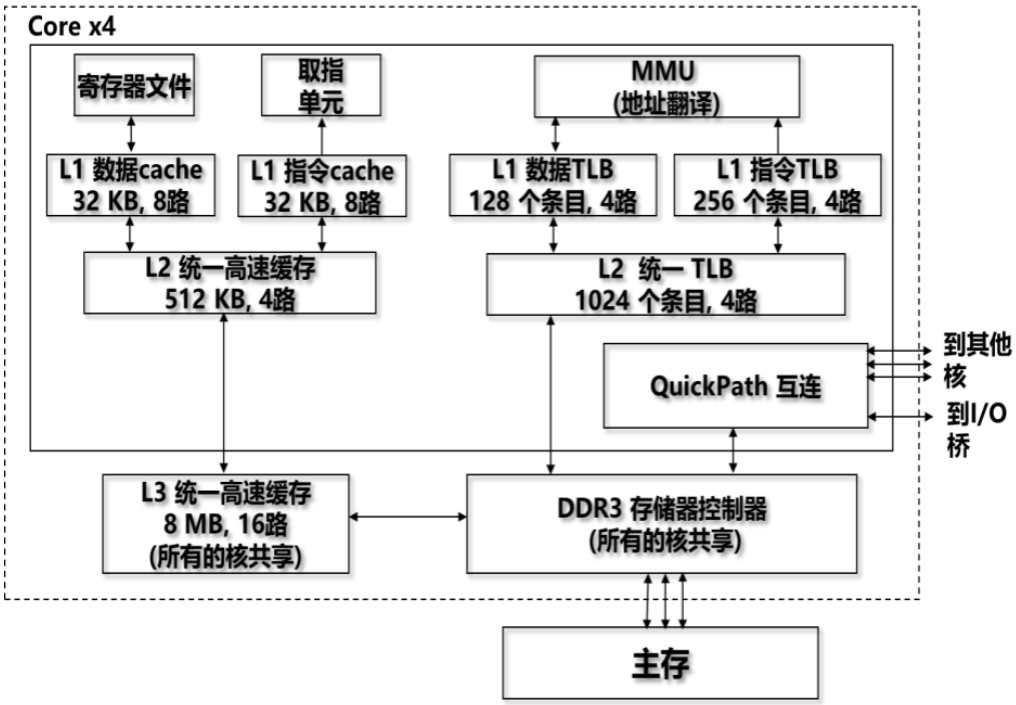
（2）TLB 不命中，缓存页表命中，缓存物理地址命中。

（3）TLB 不命中，缓存页表不命中，缓存物理地址不命中。



（1）。A B C F K
（2）。A B E D F K
（3）。A B E H G D F I J K

7. Intel I7 处理器的虚拟地址为 48 位。虚拟内存的页大小是 4KB，物理地址为 52 位，Cache 块大小为 64B。物理内存按照字节寻址。其内部结构如下图所示，依据这个结构，回答问题。



（1）虚拟地址的 VPN 占多少位，一级页表占多少项？L1 数据 TLB 的组索引位数 TLBI 占多少位？

（2）L1 数据 Cache 有多少组，相应的 Tag 位，组索引位和块内偏移位分别是多少？

（3）对于某指令，其访问的虚拟地址为 0x804849B，则该地址对应的 VPO 为多少？对应的 L1 TLBI 位为多少？（用 16 进制表示）

（4）对于某指令，其访问的物理地址为 0x804849B，则该地址访问 L1 Cache 时，CT 位为多少？CO 位为多少？（用 16 进制表示）

(1. 一页有 4KB，则页内偏移为 12 位，则 VPN 占 36 位。
一级页表占 9 位，有 512 项。
L1 数据 TLB 有 128 条，4 路，因此有 32 组，TLBI 占 5 位。

(2. L1 数据 Cache 有 32KB / (8*64B) = 64 组。Tag 位是 40 位，组索引位是 6 位，块内偏移是 6 位。

(3. L1 指令 TLB 有 256 条目 4 路，因此有 64 组，TLBI 为 6 位。于是 VPO 为 0x49b，TLBI 位为 0x8.

(4. 该物理地址访问 L1 Cache 时，CT 位为 0x8048，CO 位为 0x1B。

Part2 链接器+异常处理+I/O

一．选择题

12. 链接时两个文件同名的弱符号，以（ C ）为基准

A． 连接时先出现的

B． 连接时后出现的

C． 任一个

D． 链接报错

13. 链接时两个同名的强符号，以哪种方式处理？（ D ）

A． 链接时先出现的符号为准

B． 链接时后出现的符号为准

C． 任一个符号为准

D． 链接报错

14. 以下关于程序中链接“符号”的陈述，错误的是（ B ）

A． 赋初值的非静态全局变量是全局强符号

B． 赋初值的静态全局变量是全局强符号

C． 未赋初值的非静态全局变量是全局弱符号

D． 未赋初值的静态全局变量是本地符号

15. C 源文件 m1.c 和 m2.c 的代码分别如下所示，编译链接生成可执行文件后

执行，结果最可能为（A）

$ gcc –o a.out m2.c m1.c ; ./a.out

0x1083020 ; _____ ; _____

A. 0x1083018, 0x108301c B. 0x1083028, 0x1083024

C. 0x1083024, 0x1083028 D. 0x108301c, 0x1083018

```
// m1.c
#include <stdio.h>
int a1 ;
int a2 = 2 ;
extern int a4 ;
void hello()
{
    printf("%p;", &a1);
    printf("%p;", &a2);
    printf("%p\n", &a4);
}
```

```
//m2.c
int a4 = 10 ;
int main()
{
    extern void hello() ;
    hello() ;
    return 0 ;
}
```

16. 对于以下一段代码，可能的输出为： A

```
int count = 0;
int pid = fork();
if (pid == 0){
    printf("count = %d\n",--count);
}
```

```
else{
    printf("count = %d\n",++count);
}
printf("count = %d\n",++count);
```

A.1 2 -1 0　　B.0 0 -1 1

C.1 -1 0 0　　D.0 -1 1 2

17. Linux 进程终止的原因可能是(D)

A.收到一个信号　　B.从主程序返回　　C.执行 exit 函数　　D.以上都是

18. 下列函数中属于系统调用且调用一次，从不返回的是(B)

A.fork　　B.execve　　C.setjmp　　D.longjmp

二．填空题

1. C 语句中的全局变量，在__链接_____阶段被定位到一个确定的内存地址。

2. 子程序运行结束会向父进程发送___SIGCHLD_____信号。

3. 向指定进程发送信号的 linux 命令是____kill____。

4. Unix 内核通过三个表项___描述符表_____、___文件表_____、____V 节点表____表示打开文件。

5. Unix 内核通过调用函数___dup2_____实现 I/O 重定向。

三．分析题

9. 请阅读以下程序，然后回答问题（假设程序中的函数调用都可以正确执行）:

```
int main() {
        printf("A\n");
        if (fork() == 0) {
            printf("B\n");
        }else {
            printf("C\n");
            A
        }
        printf( “D\n"); exit(0);
    }
```

（1）如果程序中的 A 位置的代码为空，列出所有可能的输出结果：

ABCDD / ABDCD / ACBDD / ACDBD

（2）如果程序中的 A 位置的代码为：

waitpid(-1, NULL, 0);

列出所有可能的输出结果：

ABCDD / ABDCD / ACBDD

（3）如果程序中的 A 位置的代码为：

printf(“E\n”);

列出所有可能的输出结果：

ABCEDD / ABDCED / ACEBDD / ACEDBD / ABCDED / ACBDED / ACBEDD

10. 假设一个 C 语言程序有两个源文件：main.c 和 proc1.c，它们的内容如下图所示

```
1  #include <stdio.h>
2  unsigned x=257;
3  short y, z=2;
4  void proc1(void);
5  void main()
6  {
7      proc1();
8      printf("x=%u,z=%d\n", x, z);
9      return 0;
10 }
```

a）main.c 文件

```
1  double x;
2
3  void proc1()
4  {
5      x=-1.5;
6  }
```

b）proc1.c 文件

回答下列问题：

a) 在上述两个文件中出现的符号哪些是强符号？哪些是弱符号？各变量的存储空间分配在哪个节中？各占几个字节？

Main.c：强符号：x，z，main。弱符号：y，proc1.

X 分配在 data 节，占 4B。y 分配在 bss 节，占 2B。z 分配在 data 节，占 2B。

Proc1.c：强符号：proc1．弱符号：x。

b) 程序执行后打印的结果是什么？请分别画出执行第 7 行的 proc1()函数调用前、后，在地址&x 和&z 中存放的内容。

调用前 &x: 0x01 01 00 00 / &z: 0x02 00

调用后 &x: 0x00 00 00 00 / &z: 0x00 00

因此结果为 x=0，z=0.

c) 若 main.c 的第 3 行改为“short y = 1, z = 2;”,结果又会怎样?

此时 y 也存放在 data 段中。且 data 段当中的顺序是 x y z。

调用前:

调用前

&x: 0x01 01 00 00 / &y: 0x01 00 / &z: 0x02 00

调用后

&x: 0x00 00 00 00 / &y: 0x00 00 / &z: 0xf8 bf

因此结果为 x=0，z=-16392

d) 修改文件 proc1, 使得 main.c 能输出正确的结果 (即 x = 257, z = 2)。

要求修改时不能改变任何变量的数据类型和名字。

将proc1.c当中的double x移到proc1函数内部,变成一个局部变量。

11. 两个 C 语言程序 main.c、test.c 如下所示

```
/* main.c */
#include <stdio.h>
int a[4]={-1,-2,2, 3};
extern int val;
int sum();
int main(int argc, char * argv[] )
{
    val=sum();
    printf("sum=%d\n",val);
}
```

```
/* test.c */
extern int a[];
int val=0;
int sum()
{
    int i;
    for (i=0; i<4; i++)
        val += a[i];
    return val;
}
```

用如下两条指令编译、链接，生成可执行程序 test：

gcc -m64 -no-pie -fno-PIC -c test.c main.c

gcc -m64 -no-pie -fno-PIC -o test test.o main.o

运行指令 objdump -dxs main.o 输出的部分内容如下：

```
Contents of section .data:
 0000 ffffffff feffffff 02000000 03000000    ................
Contents of section .rodata:
 0000 73756d3d 25640a00                      sum=%d..
...
Disassembly of section .text:
0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 10             sub    $0x10,%rsp
   8:   89 7d fc                mov    %edi,-0x4(%rbp)
   b:   48 89 75 f0             mov    %rsi,-0x10(%rbp)
   f:   b8 00 00 00 00          mov    $0x0,%eax
  14:   e8 00 00 00 00          callq  19 <main+0x19>
                15: R_X86_64_PC32  sum-0x4
  19:   89 05 00 00 00 00       mov    %eax,0x0(%rip)    # 1f <main+0x1f>
                1b: R_X86_64_PC32  val-0x4
  1f:   8b 05 00 00 00 00       mov    0x0(%rip),%eax    # 25 <main+0x25>
                21: R_X86_64_PC32  val-0x4
  25:   89 c6                   mov    %eax,%esi
  27:   bf 00 00 00 00          mov    $0x0,%edi
                28: R_X86_64_32    .rodata
  2c:   b8 00 00 00 00          mov    $0x0,%eax
  31:   e8 00 00 00 00          callq  36 <main+0x36>
                32: R_X86_64_PC32  printf-0x4
  36:   b8 00 00 00 00          mov    $0x0,%eax
  3b:   c9                      leaveq
  3c:   c3                      retq
```

objdump -dxs test 输出的部分内容如下（■是没有显示的隐藏内容）：

```
SYMBOL TABLE:
0000000000400400 l    d  .text 0000000000000000              .text
00000000004005e0 l    d  .rodata   0000000000000000   .rodata
0000000000601020 l    d  .data     0000000000000000              .data
0000000000601040 l    d  .bss  0000000000000000              .bss
0000000000000000              F *UND*  0000000000000000  printf@@GLIBC_2.2.5
0000000000601044 g          O .bss 0000000000000004              val
0000000000601030 g          O .data      0000000000000010              a
00000000004004e7 g          F .text 0000000000000039              sum
0000000000400400 g          F .text 000000000000002b              _start
0000000000400520 g          F .text 000000000000003d              main
Contents of section .rodata:
 4005e0 01000200 73756d3d 25640a00                  ....sum=%d..
```

```
...
Contents of section .data:
 601020 00000000 00000000 00000000 00000000    ................
 601030 ffffffff feffffff 02000000 03000000    ................
...
00000000004003f0 <printf@plt>:
   4003f0:    ff 25 22 0c 20 00      jmpq    *0x200c22(%rip) # 601018
<printf@GLIBC_2.2.5>
   4003f6:    68 00 00 00 00         pushq   $0x0
   4003fb:    e9 e0 ff ff ff         jmpq    4003e0 <.plt>

Disassembly of section .text:
0000000000400400 <_start>:
   400400:    31 ed            xor     %ebp,%ebp
....
00000000004004e7 <sum>:
   4004e7:    55                   push    %rbp          #①
   4004e8:    48 89 e5             mov     %rsp,%rbp #②
   4004eb:    c7 45 fc 00 00 00 00 movl    $0x0,-0x4(%rbp) #③
   4004f2:    eb 1e                jmp     400512 <sum+0x2b>
   4004f4:    8b 45 fc             mov     -0x4(%rbp),%eax
   4004f7:    48 98                cltq
   4004f9:    8b 14 85 30 10 60 00 mov     0x601030(,%rax,4),%edx
   400500:    8b 05 3e 0b 20 00    mov     0x200b3e(%rip),%eax #601044 <val>
   400506:    01 d0                add     %edx,%eax
   400508:    89 05 36 0b 20 00    mov     %eax,0x200b36(%rip) #601044 <val>
   40050e:    83 45 fc 01          addl    $0x1,-0x4(%rbp)
   400512:    83 7d fc 03          cmpl    $0x3,-0x4(%rbp)#④
   400516:    7e dc                jle     4004f4 <sum+0xd>#⑤
   400518:    8b 05 26 0b 20 00    mov     0x200b26(%rip),%eax # 601044 <val>
   40051e:    5d                   pop     %rbp
   40051f:    c3                   retq

0000000000400520 <main>:
   400520:    55                   push    %rbp
   400521:    48 89 e5             mov     %rsp,%rbp
   400524:    48 83 ec 10          sub     $0x10,%rsp
   400528:    89 7d fc             mov     %edi,-0x4(%rbp)
   40052b:    48 89 75 f0          mov     %rsi,-0x10(%rbp)
   40052f:    b8 00 00 00 00       mov     $0x0,%eax
   400534:    e8(    ①    )       callq   4004e7 <sum>
   400539:    89 05(     ②     )   mov     %eax, ■■■■■(%rip) #601044<val>
   40053f:    8b 05(     ③     )   mov     ■■■■■(%rip),%eax    #601044<val>
   400545:    89 c6                mov     %eax,%esi
```

```
400547:   bf（  ④  ）    mov      ■■■■■,%edi
40054c:   b8 00 00 00 00  mov      $0x0,%eax
400551:   e8（  ⑤  ）    callq    4003f0 <printf@plt>
400556:   b8 00 00 00 00  mov      $0x0,%eax
40055b:   c9              leaveq
40055c:   c3              retq
40055d:   0f 1f 00        nopl     (%rax)
```

1) 阅读的 sum 函数反汇编结果中带下划线的汇编代码（编号①-⑤），解释每行指令的功能和作用。

   i. 将 rb 压入栈中

   ii. 传送指令，将 rsp 的值给 rbp，作为新的栈帧

   iii. 传送指令，将%rbp-4 赋值为 0

   iv. 比较指令，将%rbp-4 的值与 3 相减，判断条件

   v. 跳转指令，如果值<=0，则跳转到 0x4004f4

2) 根据上述信息，链接程序从目标文件 test.o 和 main.o 生成可执行程序 test，对 main 函数中空格①--⑤所在语句所引用符号的重定位结果是什么？以 16 进制 4 字节数值填写这些空格，将机器指令补充完整。

   i. Ae ff ff ff

   ii. 05 0b 20 00

   iii. Ff 0a 20 00

   iv. E4 05 40 00

   v. 9a fe ff ff

3) 在 sum 函数地址 4004f9 处的语句"mov 0x601030(,%rax,4),%edx"中，源操作数是什么类型、有效地址如何计算、对应 C 语言源程序中的什么量(或表达式)？其中，rax 数值对应 C 语言源程序中的哪个量(或表达式)？如何解释数字 4?

源操作数是整型。

有效地址是 0x601030 + %rax*4

对应 C 语言源程序的 a[i]，rax 对应 i。

4 表示比例因子，即 int 类型每个元素有 4B。