QA 1

1. 简述C、ASM、ML的关系，各自优缺点？
2. Hello.c 经过那些工具、步骤、生成什么类型的文件？
3. 什么是程序可移植性？汇编语言可移植吗？为什么？
4. 编译与解释有什么区别？各举出2个语言的例子
5. 程序优化的目的？
6. 计算机软件与硬件的界面/接口是什么？
7. 简述中间层语言及其运行机制
8. 设计开发一个Java处理器是否可行？
9. 可执行程序的组织与其在内存的一致吗？变量地址呢？
10. 程序执行结果与那些相关呢

QA 2

- 1.8086 CPU内部寄存器组有哪几个？都是多少位的？
- 2.8086 访问存储器的地址有哪部分组成？物理地址怎么形成的？
- 3.BIU与EU是怎么完成指令执行的？
- 4. int x； AX=100； AX=AX+256； x=AX；
- 程序/指令中的常数在内存哪个区域/段？
- 5. 全局变量x以一种什么方式在机器/汇编程序中出现？怎么访问的？至少访几次存储器？
- 6.C语言的数据类型char、short int、int、long、long long、float、double、long double、指针等在32/64位系统中占多少个字节？邮编为什么char[]?

QA 3

- 0. 计算机是64位的是指CPU寄存器是64位的？
- 1.常量表达式是谁来计算的？
- 2.0与‘0'谁大?差多少?空间呢??
- 3.C源程序中的有符号常数，是怎么变成二进制补码的进行和运算的？
- 4.strlen(“1234567 我想毕业\n”)=？怎么算汉字数
- 5.汉字是怎么输入进入源程序的（Win/Linux）？
- 6. OS内核的编码决定了基于其上的系统软件与应用软件的编码要与其一致？不一致怎么办?要转换吗?谁来转换?
- 7.当前的源程序编码是什么? 可以变吗?
- 8.汉字是怎么显示/打印输出的？
- 9. “联通” “洗头” “写”
- 10. int x; x取反 = X ___ _____

QA 4

- IEEE754比整数部分10位+小数部分20位的表示方法有什么优点？ 缺点呢—考虑下?
- float非无穷的最大值，最小值？
- Float的最大绝对值？最小绝对值？
- float数 1，65536，0.4，-1，0的内存表示
- 一个数的Float形式是唯一的吗？（除了0）
- 每一个IEEE754编码对应的数是唯一的吗？
- Float的阶码范围是多少？
- 简述Float数据的浮点数密度分布？
- C语言中除以0一定报错溢出吗？（整数报错，浮点无穷大X/0>Y 可以）

- int与float都占32个二进制位，Float与INT相比谁的个数多？各自是多少个？多多少？ （+-0、nan）
- Float的最大密度区间（非无穷）？Float数多少？密度多少？
- Float的最小密度区间？Float数多少？密度多少？
- Float最大密度区间是最小密度区间的密度的多少倍？
- float最大的负数是多少==最小的正数是多少？
- 怎么判断和定义浮点数的无穷大以及NaN？
- 浮点数的表示，越小精度越高，越大精度越低，这也基本符合数据处理的规律。太大的数据差点没啥，就是个规模而已。如人口、GDP等，没有必要到个、圆角分。
- Float使用注意事项？ 怎么比较两个float数据？

QA 5

- 浮点数的舍入采用什么规则？
- 向偶数舍入方法，尾数会不会产生溢出，如果产生溢出怎么办？阶码会不会溢出？怎么办？
- printf函数把浮点数按照十进制数打印%10.2f，按照什么舍入原则呢？ 到底是什么？不要想当然

- 通用寄存器的通用是什么意思
- 32位CPU怎么实现64位加法
- SI/DI怎么使用（1-2班没讲）？
- 数组、结构、指针各用什么寄存器指示？

QA 6

- 标志寄存器有什么作用？
- 状态标志位有哪几个？加法运算会影响那些？
- 控制标志位 IF/DF作用，怎么改变？
- CF与OF是什么关系？
- 计算机怎么判断两个数相加是否超出了范围？
- 怎么修改IP寄存器？
- 32/64位的cpu中寄存器都是32/64位的？
- 请列出C语言的所有操作/指令，与汇编语言对比，说明汇编语言的优点
- 重点：逻辑操作/位操作

QA 7

- 操作数的寻址方式有哪几种？
- 一/二维数组采用什么寻址方式？
- 结构体的某一个整型成员采取什么寻址方式？
- 结构体的整形数组采用什么寻址方式？
- 一个C生成的执行程序是多少位的是由谁来决定的？
  - （A）CPU　（B）OS　（C）编译器　（D）源程序

QA 8

- gcc在32位与64位执行文件生成过程中，对全局变量是怎么处理的？
- 局部变量呢？
- 32/64位下，参数传递呢？

- 变量作为参数的传值与传地址是怎么实现的？
- 可变个数的参数怎么识别和处理的？

- Printf等格式串，处理成常量还是变量？全局还是局部变量？
- 指针访问怎么处理的？

QA 9

- int a[100]; 在作为全局变量、局部变量、参数时是怎么实现对a[i]元素的访问的？
- int *p;　　*(P++)　与(*p)++实现有什么不同？

if 分支语句的汇编语言实现与C源程序的条件判断之间是_____关系。

- if(y>100)
  - 0x5555555548c9　　mov　-0x20(%rbp),%eax
  - 0x5555555548cc　　cmp　$0x64,%eax
  - 0x5555555548cf　77 _　jle　0x5555555548d8 <main+105>
  - 0x5555555548d1　　movl　$0xffffffff,-0x20(%rbp)
- switch多分支，case条件码不规律时是怎么实现的？

QA 10

- **switch多分支，case条件码规律时是怎么实现的？（注意我们讲的与书上不同，看下不同在哪儿？）**
  - 主要采用的数据结构是什么？
  - 跳转表的元素类型？　　（不同）
  - 跳转表的元素个数怎么计算？
  - 跳转表的初值是什么？什么时候由谁来赋初值的？
  - 元素下标怎么计算？
  - 怎么计算跳转地址？
  - 跳转指令是什么寻址方式？
- **叶子节点与非叶子节点的函数汇编语言实现有什么区别？**
- **递归程序的汇编语言是怎么实现的？**

QA 11

- 静态局部变量是怎么生成汇编语言的？
  - 空间
  - 赋初值
  - 生命周期
- 分析全局变量、局部变量、传值的参数的生命周期
- 简述缓冲器溢出的原理
- 怎么攻击缓冲器溢出的漏洞？
- 怎么防范缓冲器溢出漏洞？
- 结构体的成员在汇编语言层面是怎么操作的？
- 结构体成员作为参数传输是怎么实现的？
- 结构体成员作为返回值是怎么实现的？

QA 12

- CPU设计的主要思想与步骤有哪些？
- ISA设计的原则与注意事项？
- Y86-64CPU的状态存储部件主要有哪些？
- Y86-64的寻址方式有哪几种？
- Y86-64指令系统的算逻操作的操作数与X86-64有什么区别？
- Y86-64的Jxx跳转指令，寻址方式与X86-64（短/近）有什么区别？
- rrmovq rax,rbx的机器指令是？
- Y86-64支持过程调用的指令有哪几个？
- Y86-64标志位怎么改变-副作用，怎么利用？
- 状态码：是怎么产生的？X86为什么看起来没有呢？
  - AOK(1)-正常操作　　HLT(2)-执行Halt
  - ADR(3)-错误的指令或数据地址　INS(4)-无效指令

QA 13

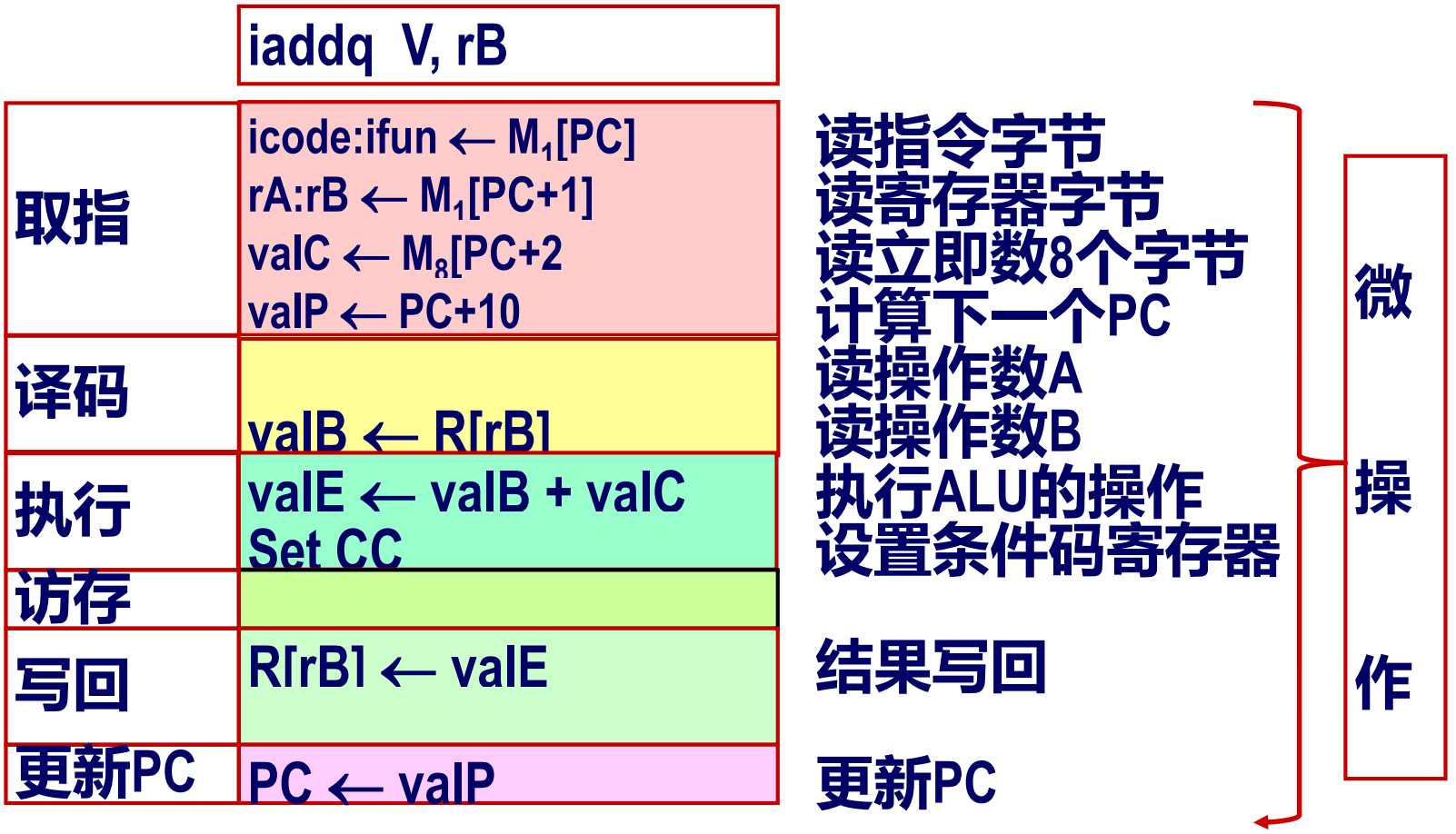
- Y86的SEQ分那几个阶段？各阶段的主要过程？
- Y86的SEQ阶段分析，12条指令的微操作实现的共同点有哪些？
  - 所有的指令有相同的格式，所有的指令都按照6个阶段顺序执行
  - 每一阶段的微操作，按照顺序有3-8种
  - 每一条指令在各阶段执行时，根据指令类型按顺序执行不同的微操作
- Y86的硬件结构包括哪些？
  - 硬件单元（组合、时序逻辑）、控制逻辑、信号连接线
  - 组合逻辑：硬件单元间传播，不用CLK，有延迟。
    - 计算逻辑（ALU、PCINC）读逻辑（ PC、CC、RF、imem、dmem）
  - 硬件单元的时序逻辑-状态单元：都在CLK上升沿时更新。控制阻断
    - 时钟寄存器：PC、CC、STAT的更新
    - 随机访问存储器：RF写、dmem写
- 怎么理解SEQ原则-从不回读？

- 所谓CPU顺序实现SEQ：一个时钟变化，引发一个经过组合逻辑的流，从而执行整个指令。
- 一条新指令的执行，即一个CLK是把上一条指令计算结果-状态单元更新，然后其值进行组合逻辑(新指令的计算)的传播
- 要控制CPU中活动的顺序，只需要用CLK控制寄存器和内存。所有６个阶段的所有步骤的状态更新同时发生（逐级延迟）。且只在时钟上升开始下一个周期时。
- 当前时钟/指令n结束，下一时钟/指令n+1没有到上升沿前，状态单元的数据-状态并不是本指令/时钟n执行的结果，仍然是上一时钟/指令n-1的结果
- 指令n的执行结果，到下一指令n+1的上升沿才更新到转状态单元

- HALT、NOP的微操作？新增C0指令Iraddq V,rB的微操作？
- 取指逻辑的need_valC的HCL

- **硬件单元：**
  - 运算逻辑： ALU、PCINC、
  - 时钟寄存器：PC、CC、STAT、
  - 随机访问存储器：RF、IMEM、DMEM
- **读操作-看成组合逻辑-有地址就有输出**
  - 随机访问存储器：RF、imem/dmem可以用特殊的时钟电路来模拟
  - 时钟寄存器：PC、CC、STAT、
- **写操作：需要时钟信号控制，控制阻断**
  - RF、DMEM、PC、CC、STAT
- **控制逻辑：每一阶段每一步骤的微操作，在硬件上的映射**
  - 在不同硬件单元之间传送数据
  - 操作硬件单元，使得对不同指令执行指定的微操作
- **组合逻辑：不需要任何时序或控制，只要输入变化了，值就通过逻辑门网络传播**
- **时序逻辑：需要时钟信号控制，进行更新-把输入锁存到输出。**

计算序列：iaddq



- V=====valC，指令中V是加数，是ALUA的一个输入
- 这些微操作都有现成的硬件结构，所以不需要改动状态单元，但控制逻辑需要稍微调整。

QA 14

- Y86-64的SEQ顺序结构实现的缺点是什么？
- 依据Y86-64的SEQ实现，STAT可以不用寄存器实现吗？
- 增加iaddq指令，SEQ结构与HCL怎么改进？
- 增加iopq指令，SEQ结构与HCL怎么改进？
- 增加mropq指令呢？
- 增加rmopq指令呢？
- 更新PC阶段是否可以优化为取指阶段完成呢？其输入输出呢？
- 写出每一阶段的输入输出（怎么做呀）？
- 能不能采用C语言描述控制逻辑呢？怎么描述？
- 用Verilog描述Y86-64CPU并能够用口袋板实现。怎么验证？

QA 15

- 为什么用连接器？
- 连接器工作内容与步骤？
- 局部变量链接时按照本地符号进行解析
- 全局变量按照全局符号进行解析
- 链接器对局部变量、参数等符号怎么解析？
- goto L0 的地址符号的强弱？链接时怎么处理？
- 有哪几种目标文件？怎么查看目标文件的各节信息？
- 可执行目标文件中的重定位主要完成哪些工作？
- 目标文件中的数据节有哪几种，分别有什么区别？
- 可执行目标文件在执行时由操作系统把各节按大小在内存分配，并拷贝相应内容到内存区？

QA 16

- 按照从低到高地址，Linux下内存分成哪几个区？
- 全局变量怎么实现重定位（32位/64位）？
- 子程序怎么实现重定位？
- Linux下归档器/库管理程序是什么？
- 库管理中每个函数可以生成单独.o也可以多个函数用一个.o，对应的源程序也是这样的。
- 静态连接时的目标文件生成与库文件的顺序无关。
- 共享库的动态连接有哪几种方法？
- 库打桩的应用有哪几个方面？
- 在什么时候进行库打桩？

QA 17

- 从哪几方面优化程序？在底层与核心技术方面、上层与应用方面、算法与UI方面等优化重点有何区别？
- 程序的性能优化从哪几方面解决？优化性能为何一定要理解“系统”？
- 编译器能做的程序员也通常采用的一般有用的优化有哪几种？
- 妨碍编译器优化的因素有哪些？为什么编译器不能优化呢？
- 妨碍编译器优化的函数调用怎么进行补救或处理？
- 妨碍编译器优化的内存别名怎么进行补救或处理？
- 编译器优化的其他方法与策略？

其他编译相关优化技术

- 寄存器比内存快
- 用快的指令
- 用快的寻址方式
- 位操作比算术运算快
- 整数比浮点运算块
- 宏比函数快
- 简单的结构比复杂结构快
- 算法很重要
- C嵌入汇编快
- 并行比串行快

QA 18

1. 简述现代超标量CPU的结构与执行程序特点
2. 什么容量、延迟界限、发射时间，他们之间有什么关系？
3. x = (x OP d[i]) OP d[i+1] 与x = x OP (d[i] OP d[i+1])哪一个并行性好？
4. 采用x0 = x0 OP d[i]; x1 = x1 OP d[i+1]; 的分离累加器方法与x = x OP (d[i] OP d[i+1])的在流水线利用方面有何区别？
5. 什么是SIMD？为什么会提高整数运算的速度？
6. 对CPU分支预测怎么处理的？程序员与编译器怎么应对？

QA 19

1. 以指令执行为例，说明CPU、内存、外存在数据访问与处理上的区别。
2. 内存MM除了存储单元外，还有哪些硬件？
3. 内存中的行缓冲器通常采用SRAM还是DRAM？
4. SRAM与DRAM物理构成有何区别？
5. DRAM芯片访问时地址信号线是怎么复用的？
6. 内存是4G的，此地址空间不包括BIOS的ROM存储器G
7. 内存提高访问速度的方法有哪几种？
8. 为什么每出现新一代DRAM芯片，容量至少提高４倍？
9. 操作系统对硬盘访问采用逻辑方式(簇)，由谁来实现此块号与物理磁面号/道号/扇区号的访问呢？
10. 内存与硬盘间传输的块方式采用什么硬件实现？

QA 20

1. 存储访问局部性原理是什么意思？
2. Cache访问不命中的种类？
3. Cache怎么组织，其大小怎么计算？
4. 程序怎么利用高速缓存访问存储器？
5. 空间局部性好的程序也常会发生Cache抖动，导致性能大降，是怎么回事？怎么解决？
6. 为什么用中间位不用高位作为组/索引？
7. 你的计算机CPU各级Cache多大？是哪一种？C、S、E、B各是多少？
8. LRU的算法实现？命中行Counter=0，其他行的加1，淘汰计数器最大的-请设计控制逻辑（8路）。
9. I7的CPU，L2Cache为8路的2M容量，则其B=　S=　E=
10. 全相联Cache有冲突不命中吗？为什么
11. 访问高速缓存的地址比内存的地址要短，是吗？

QA 21

- Cache的写命中策略有哪些？
- Cache的写不命中策略有哪些？
- 为什么高速缓存用不命中率而不是命中率来衡量性能？
- 怎么编写高速缓存友好的代码？
- 重新排列以提升程序的_____局部性
- 使用分块以提升程序的_____局部性
- 使用分块提高缓冲器命中率的基本思想与算法？
- 新的计算机cache技术：指令Cache->微指令Cache、追踪缓存TraceCache等， TLB等

QA 22

- 计算机系统的各层发生哪些异常控制？
- 异常处理后有哪三种返回机制？
- 简述异常、异常号、异常处理程序、异常表的关系？
- 在C中通过scanf等IO函数处理IO异常。
- Linux系统调用中的功能号n就是异常号n
- 系统调用是一个特殊的异常处理子程序
- 异步异常需要/只需要CPU执行异常处理子程序，只需要操作系统处理即可。
- 浮点除以0产生异常，导致程序终止Abort退出。
- 浮点数除以0，不产生异常，程序正常执行。
- 任何数除以0产生异常，导致程序终止Abort退出。
- 每个进程的地址空间是整个内存空间的一部分

用“系统思维”分析问题

代码段一：

```
int a = 0x80000000;
int b = a / -1;
printf("%d\n", b);
```

运行结果为-2147483648

objdump反汇编代码，得知除以 -1被优化成取负指令neg，故未发生除法溢出

理解该问题需要知道：

编译器如何优化

机器级数据的表示

机器指令的含义和执行

计算机内部的运算电路

除法错异常的处理

……

代码段二：

```
int a = 0x80000000;
int b = -1;
int c = a / b;
printf("%d\n", c);
```

a/b用除法指令IDIV实现，但它不生成OF标志，那么如何判断溢出异常的呢?

实际上是“除法错”异常#DE（类型0）
Linux中，对#DE类型发SIGFPE信号

运行结果为“Floating point exception”，显然CPU检测到了溢出异常

为什么两者结果不同!

Linux 中异常对应的信号名和处理程序名

类型号	助记符	含义描述	处理程序名	信号名
0	#DE	除法出错	divide_error( )	SIGFPE
1	#DB	单步跟踪	debug( )	SIGTRAP
2		NMI 中断	nmi( )	无
3	#BP	断点	int3( )	SIGTRAP
4	#OF	溢出	overflow( )	SIGSEGV
5	#BR	边界检测（BOUND）	bounds( )	SIGSEGV
6	#UD	无效操作码	invalid( )	SIGILL
7	#NM	协处理器不存在	device_not_available( )	无
8	#DF	双重故障	doublefault( )	无
9	#MF	协处理器段越界	coprocessor_segment_overrun( )	SIGFPE
10	#TS	无效 TSS	invalid_tss( )	SIGSEGV
11	#NP	段不存在	segment_not_present( )	SIGBUS
12	#SS	栈段错	stack_segment( )	SIGBUS
13	#GP	一般性保护错（GPF）	general_protecton( )	SIGSEGV
14	#PF	页故障	page_fault( )	SIGSEGV
15		保留	无	无
16	#MF	浮点错误	coprocessor_error( )	SIGFPE
17	#AC	对齐检测	alignment_check( )	SIGSEGV
18	#MC	机器检测异常	machine_check( )	无
19	#XM	SIMD 浮点异常	simd_coprocessor_error( )	SIGFPE

为何除法错显示却是“浮点异常”的原因！

QA 23

- 并发进程是并行执行的，控制流物理上是不相交的。
- 任务管理器中的进程内是串行、进程间是并行
- fork函数调用一次返回两次，怎么可能呀？
- 进程状态有哪几种？进程终止有几种方法？
- kill—杀死 是终止进程吗？kill某进程是怎么实现的？
- Kill是终止进程并回收进程吗？
- fork的子进程与其父进程有什么相同与区别？
- fork的子进程与其父进程同名的全局变量对应同一物理地址？
- fork生成副本太浪费空间了，能不能先与父进程共享物理内存？特别是代码段！而且fork后还得execve覆盖掉。
- fork7去掉while(1)，执行后再ps，父与子进程能看到哪一个？
- fork8父进程exit为什么ps看不出父进程defunct?

任务/进程数据结构，或称为进程描述符

```
struct task_struct {      /* these are hardcoded - don't touch */
        long state;                     /* -1 unrunnable, 0 runnable, >0 stopped */
        long counter;                   //任务运行时间计数(递减)（滴答数），运行时间片
        long priority;                  //运行优先数。任务开始运行时counter = priority，越大运行越长
        long signal;                    //信号。是位图，每个比特位代表一种信号，信号值=位偏移值+1。
        struct sigaction sigaction[32]; // 信号执行属性结构，对应信号将要执行的操作和标志信息。
        long blocked;                   /* bitmap of masked signals */   //进程信号屏蔽码（对应信号位图）
/* various fields */
        int exit_code;                  //任务执行停止的退出码，其父进程会取。
        unsigned long start_code,end_code,end_data,brk,start_stack; //代码段地址、代码长度（字节数）、 代码长度 + 数据长度（字节数）、 总长度（字节数）、 堆栈段地址。
        long pid,father,pgrp,session,leader;  //进程号、父进程号、父进程组号、会话号、 会话首领
        unsigned short uid,euid,suid;         //用户id、有效用户id、保存的用户id。
        unsigned short gid,egid,sgid;         //组标识号（组id）、有效组id、保存的组id
        long alarm;                           //报警定时值（滴答数）
        long utime,stime,cutime,cstime,start_time;   //用户态运行时间（滴答数）、系统态运行时间、子进程用户态运行时间、子进程系统态运行时间、进程开始运行时刻
        unsigned short used_math;             //标志：是否使用了协处理器
/* file system info */
        int tty;                        //进程使用tty 的子设备号。-1 表示没有使用。
        unsigned short umask;           //文件创建属性屏蔽位。
        struct m_inode * pwd;           //当前工作目录i 节点结构
        struct m_inode * root;          //根目录i 节点结构
        struct m_inode * executable;    //执行文件i 节点结构
        unsigned long close_on_exec;    //执行时关闭文件句柄位图标志。（参见include/fcntl.h）
        struct file * filp[NR_OPEN];    //进程使用的文件表结构
        struct desc_struct ldt[3];      //本任务的局部表描述符。0-空，1-代码段cs，2-数据和堆栈段ds&ss。
        struct tss_struct tss;          //本进程的任务状态段信息结构
};
```

QA 24

- 简述hello从命令行到运行的过程
  - 1fork->2execve->3load->4 dl_init setjmp等 到 _start-(所有C程序的入口点,在ctrl.o中) ->->5_libc_start_main(系统启动函数，在libc.so,初始化执行环境
  - 调用main执行，处理main返回值，控制返回给内核) ->6main ---->exit(0) 或 return 0 终止 --->bash回收
- 信号与异常的区别与联系
- 信号的处理方法有哪几种？
- 批处理、作业、任务、程序、进程？
- 异步异常/中断与信号的处理有何不同？
- 有的中断/异步异常会产生信号？（键盘—Ctrl-C/Z）
- Linux处理“除法错”异常#DE（类型0）发送SIGFPE信号
- Linux用信号处理同步异常，用中断处理异步异常（pushf）
- 信号处理程序与中断处理程序可以各自嵌套与互相嵌套？同一信号/中断同时多次发生呢？ 缺省信号可嵌套中断不嵌套
- 一般的故障类异常-内中断的处理是通过向发生异常的进程发送信号的机制实现异常处理，可尽快完成在内核态的异常处理过程，因为异常处理过程越长，嵌套执行异常的可能性越大，而异常嵌套执行会付出较大的代价。但缺页故障不是
- Linux用两个32/64位数表示信号与其阻塞
- kill函数是终止进程并回收进程
- 延时过程调用与中断服务例程

QA 25

- 计算机的地址空间有哪几种？怎么进行变换？
- MMU怎么将[EBP+ESI*8+4]变换成虚拟地址的？
- Linux是怎么处理Intel的分段管理机制的？
- Intel没有只采用物理寻址的计算机系统。
- 虚拟页面有哪几种状态？
- MMU怎么分配一个新的页面（VM/PM/PTE）？
- 多个进程的PTE项目可能存储同一内容？
- VM是怎么支持连接与进程创建、加载运行的？
- TLB的Block是什么内容？32/64位系统为多少字节？
- TLB怎么实现VA中VP的访问（VPN/PPN/TLBI/TLBT/VPO）
- 多级页表中一级页表与末级页表的表项分别是什么，每张表是不是都必须是1K个元素（32位）？
- 执行MOV EAX,[EBP+ESI*8+4]最多访存几次(TLB2级Cache3级)？

QA 26

- 多级页表中一级页表与末级页表的表项分别是什么，每张表是不是都必须是1K个元素（32位）？
- 执行MOV EAX,[EBP+ESI*8+4]最多访存几次(TLB2级Cache3级)？
- I7 CPU的VA___位，PA___位，VPO___位，PPO____位，VPN___位，PPN____位。采用_____级页表，其页表项数依次为_____，页表空间为____字节，每一页表项占_____字节。TLB采用Cache类型_____，TLB-D1为4路64条，则其TLBI___位，TLBT___位。CacheD1为8路32K，则其_____组，Block为____字节，tag为___位。PA中CT/CI/CO依次为__、___、___位。页表物理基地址为____字节。
- Linux的VM机制对私有写时复制对象是怎么处理的？
- fork的子进程与其父进程同名的全局变量对应同一物理地址？
- fork生成副本太浪费空间了，能不能先与父进程共享物理内存？特别是代码段！而且fork后还得execve覆盖掉。
- execve时当前进程哪些区域是请求二进制零的匿名文件映射？

QA 27

- 举例说明采用显式分配器与隐式分配器的语言。
- 为提高内存利用率，动态分配器会移动较小的已分配块以便空出更多的空闲块，供用户分配。
- 一个内存块中的内部碎片是怎么产生的？
- 如何知道一个指针可以释放多少内存?
- 如何区分空闲块与已分配块？
- 记录空闲块的方法有几种？
- 隐式空闲链如何判断刚释放的块，其前后是否是空闲的呢？
- 隐式空闲链查找空闲块有哪几种方法？优缺点？
- 释放块后的合并空闲块有哪几种情况？
- 显式空闲链表比隐式空闲链表的实现节省空间?
- 显式空闲链表的释放块处理，请比较LIFO与地址顺序法的吞吐率与空间利用率。
- 分离的空闲链表与隐式/显式空闲链表相比时空有什么优缺点？

QA 28

- 垃圾回收时堆使用的内存有向图的根节点有哪几种类型？
- 按照有向图与标记清除方法，所有的垃圾都可以回收？举例
- 除了CPU与RAM，计算机还有啥呀？
- 接口有什么用呀？没有不行吗？
- CPU、存储器、所有的接口等是否都可以集成到一个芯片里？
- 怎么对IO接口芯片和外设进行编程控制呢？
- Linux的文件都有哪些类型？
- Linux的文件操作的编程有几种方式？分别应用于那些场合？
- IO重定向是什么实现的？
- 进程间如何共享打开文件的？
- YN：C的标准IO函数都是带缓冲的，UnixIO函数不带缓冲。
- Unix的IO函数与C标准函数可以混合使用吗？可能会有什么问题？
- 对IO设备与接口芯片的操作，最终会采用什么机器指令完成？
- 带缓冲的IO什么时候真正输入或输出—清空缓冲区？