复习内容：第一章、第二章、第三章课本内容、PPT内容、作业习题。

**第一章**

冯诺依曼型计算机五大功能部件；

计算机系统的层次化结构；

计算机系统的性能评价：响应时间（执行时间）和吞吐率；

MIPS、CPI等；

响应时间（执行时间）=时钟周期数\*时钟周期=指令数\*CPI\*时钟周期=指令数\*CPI/时钟频率；

**第二章**

十进制与二进制之间转换；（包括整数转二进制和小数转二进制）

原码、补码、反码、移码转换；

补码的优点；移码的优点；

IEEE 754标准中单精度浮点、双精度浮点格式；

C语言中int、short、char、float等各种常见数据类型所占字节数；

大小端机器数据在内存中的存储顺序差别，注意x86是小端方式；

什么是对齐？为什么要对齐？

**第三章**

IA-32中通用寄存器的名称、宽度；（注意EAX,AX,AH,AL的位宽差别）

IA-32中EFLAGS、EIP的功能；

IA-32中各类寻址方式及有效地址的计算方式；

了解IA-32中各类传输指令功能，包括MOV、PUSH/POP、LEA、PUSHF/POPF等；

了解IA-32中各类定点算术运算指令的功能，包括ADD/SUB，INC/DEC、NEG、CMP、MUL/DIV、IMUL/IDIV；

了解IA-32中各类逻辑运算指令的功能，包括NOT、AND、OR、XOR、TEST等；

了解IA-32中各类移位指令的功能，包括SHL/SHR、SAL/SAR等。

了解IA-32中各类控制转移指令的功能，包括JMP、Jcc等；

了解IA-32中调用和返回指令的功能，包括CALL、RET等。

理解C语言中过程调用的执行步骤；

理解堆、栈、栈帧的概念，能画出过程调用中栈帧的变化；

牢记EAX是过程调用return的返回值；

牢记在栈帧中EBP+8一般是指向第一个入口参数，EBP+4指向返回地址，EBP指向EBP的旧值，EBP-4指向被调用者的第一个非静态局部变量；

理解过程调用中按值传递参数和按地址传递参数的本质区别；

了解常用C语言与汇编语言之间的对应关系：if else；switch；条件表达式；while；for等。

难点：能在C语言与汇编语言之间做手动转换，重点是读懂汇编程序，反汇编为对应的C代码。也能够根据C语言，手动汇编为对应的汇编程序。参考作业题：13和14等。

了解数组元素、结构体数据、联合体数据在存储空间中的存放方式及访问机制。

了解越界访问和缓冲器溢出攻击机器及其防范方法。

注意：不用记忆汇编语言格式，考试时会给出汇编指令功能提示，类似下面的表格（统一按AT&T格式给出）：（一）

凡是考试中涉及的指令均会给出功能提示。

栈帧格式记忆图：（二）

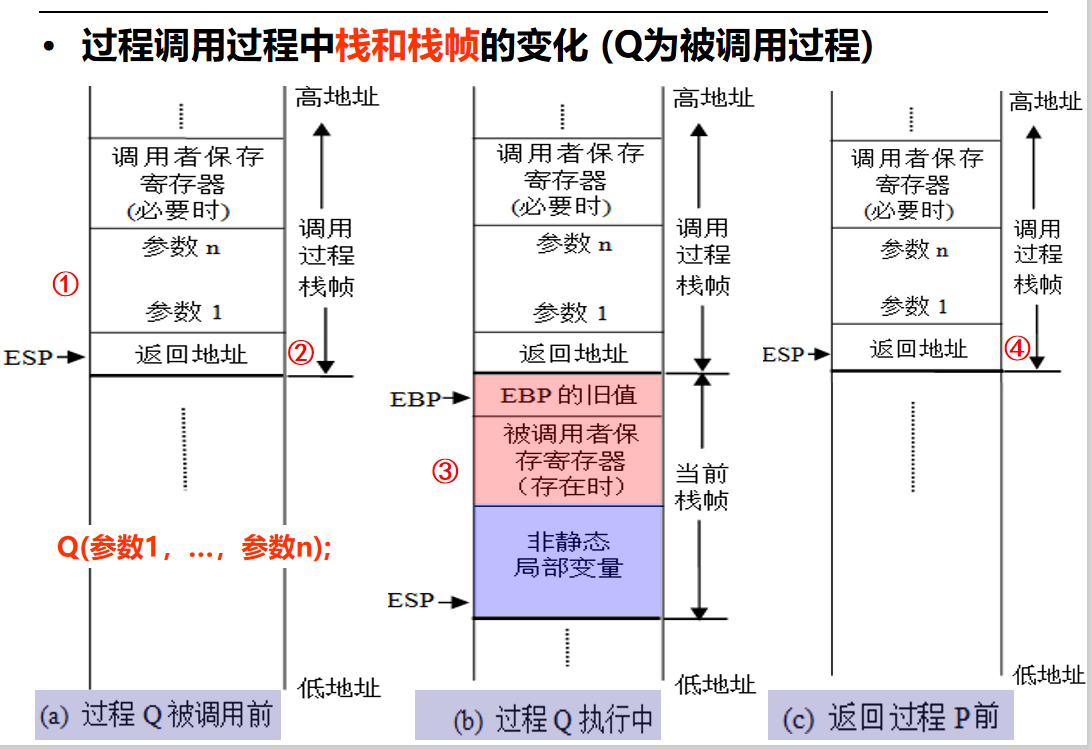
**第四章**

本章未布置作业，仅作一般了解。

了解ELF可重定位目标文件的格式。

了解ELF可执行文件的格式。

了解静态链接与动态链接的概念。



|  |  |
| --- | --- |
| 汇编指令 | 指令功能 |
| Movl %eax, %ebx | R[ebx]=R[eax] |
| Call Label | Push %eip  R[eip]=Label |
| sub $1, %edx | R[edx]=R[edx]-1 |
| cmpl %edx, %eax | 根据R[eax]-R[edx]减法结果改变EFLAGS，但R[eax]值不改写 |
| Mulb %bl | R[ax]=R[al]\*R[bl] |
| salw $2, %ax | R[ax]=R[ax]<<2 |
| cmpl %edx，%eax ，jg Lable | 带符号整数%edx>%eax时跳转 |

**AT&T的汇编语法，以及GCC的内嵌汇编语法。**

**1.寄存器引用**

引用寄存器要在寄存器号前加百分号%,如“movl %eax, %ebx”。

80386有如下寄存器：

* 8个32-bit寄存器 %eax，%ebx，%ecx，%edx，%edi，%esi，%ebp，%esp；
* 8个16-bit寄存器，它们事实上是上面8个32-bit寄存器的低16位：%ax，%bx，%cx，%dx，%di，%si，%bp，%sp；
* 8个8-bit寄存器：%ah，%al，%bh，%bl，%ch，%cl，%dh，%dl。它们事实上是寄存器%ax，%bx，%cx，%dx的高8位和低8位；
* 6个段寄存器：%cs(code)，%ds(data)，%ss(stack), %es，%fs，%gs；
* 3个控制寄存器：%cr0，%cr2，%cr3；
* 6个debug寄存器：%db0，%db1，%db2，%db3，%db6，%db7；
* 2个测试寄存器：%tr6，%tr7；
* 8个浮点寄存器栈：%st(0)，%st(1)，%st(2)，%st(3)，%st(4)，%st(5)，%st(6)，%st(7)。

**2. 操作数顺序**

操作数排列是从源（左）到目的（右），如“movl %eax(源）, %ebx(目的）”

**3. 立即数**

使用立即数，要在数前面加符号$, 如“movl $0x04, %ebx”

或者：

para = 0x04

movl $para, %ebx

指令执行的结果是将立即数04h装入寄存器ebx。

**4. 符号常数**

符号常数直接引用 如

value: .long 0x12a3f2de

movl value , %ebx

指令执行的结果是将常数0x12a3f2de装入寄存器ebx。

引用符号地址在符号前加符号$, 如“movl $value, % ebx”则是将符号value的地址装入寄存器ebx。

**5. 操作数的长度**

操作数的长度用加在指令后的符号表示b(byte, 8-bit), w(word, 16-bits), l(long, 32-bits)，如“movb %al, %bl”，“movw %ax, %bx”，“movl %eax, %ebx ”。

如果没有指定操作数长度的话，编译器将按照目标操作数的长度来设置。比如指令“mov %ax, %bx”，由于目标操作数bx的长度为word，那么编译器将把此指令等同于“movw %ax, %bx”。同样道理，指令“mov $4, %ebx”等同于指令“movl $4, %ebx”，“push %al”等同于“pushb %al”。对于没有指定操作数长度，但编译器又无法猜测的指令，编译器将会报错，比如指令“push $4”。

**6. 符号扩展和零扩展指令**

绝大多数面向80386的AT&T汇编指令与Intel格式的汇编指令都是相同的，符号扩展指令和零扩展指令则是仅有的不同格式指令。

符号扩展指令和零扩展指令需要指定源操作数长度和目的操作数长度，即使在某些指令中这些操作数是隐含的。

在AT&T语法中，符号扩展和零扩展指令的格式为，基本部分"movs"和"movz"（对应Intel语法的movsx和movzx），后面跟上源操作数长度和目的操作数长度。movsbl意味着movs （from）byte （to）long；movbw意味着movs （from）byte （to）word；movswl意味着movs （from）word （to）long。对于movz指令也一样。比如指令“movsbl %al, %edx”意味着将al寄存器的内容进行符号扩展后放置到edx寄存器中。

其它的Intel格式的符号扩展指令还有：

* cbw -- sign-extend byte in %al to word in %ax；
* cwde -- sign-extend word in %ax to long in %eax；
* cwd -- sign-extend word in %ax to long in %dx:%ax；
* cdq -- sign-extend dword in %eax to quad in %edx:%eax；

对应的AT&T语法的指令为cbtw，cwtl，cwtd，cltd。

**7. 调用和跳转指令**

段内调用和跳转指令为"call"，"ret"和"jmp"，段间调用和跳转指令为"lcall"，"lret"和"ljmp"。

段间调用和跳转指令的格式为“lcall/ljmp $SECTION, $OFFSET”，而段间返回指令则为“lret $STACK-ADJUST”。