CS APP 笔记

王康

一、二章

考点一 翻译过程

.c → 预处理 cpp → .i → 编译器 cc1 → .s → 汇编器 as → .o → 链接器 ld → prog

考点二 OS抽象概念

- 文件 → IO设备
- 虚拟内存 → 主存(DRAM)和磁盘I/O设备的抽象(程序存储器)
- 进程 → 处理器、主存和I/O设备的抽象(资源分配的单位),也是对运行程序的抽象
- 虚拟机 → 整个计算机的抽象

考点三 进制转换

整数 — 除基取余法.

小数 — 乘基取整法.

考点四 字节和字节顺序

大端、小端:(我们使用小端法)

考点五 布尔代数

NOT ~ 补集　　AND & 交集　　OR | 并集　　XOR ^ 对称差集

Tips:三个要点:一种是补码除法 $(x<0\ ?\ x+(1<<k)-1\ :\ x)>>k = x/2^k$

一种是浮点数阶码偏置值 $Bias = 2^{k-1}-1$

double 64位:s、e、f → 1, 11, 52

float 32位:s、e、f → 1, 8, 23

一种是舍入机制,非中间值则就近;中间值,则向偶数舍入.

考点六:P.86. 整、浮转换.

三章 程序的机器级表示

考点，一、 可见状态

程序计数器(PC，x86-64中用%rip表示)；16个整型寄存器；条件码寄存器；一组向量寄存器(浮点数寄存器)

考点，二、 gcc 指令

gcc -S xxx.c 产生汇编文件 xxx.s

gcc -c xxx.c 产生目标文件 xxx.o

objdump -d xxx.o 反汇编目标文件

gcc -o prog 产生可执行文件

cpp	main.c	main.i		c→i
cc1	-o	main.i	main.s	i→s
as	-o	main.o	main.s	s→o
ld	-o	prog	[.o .o]	.o → prog

Tips：1. 生成1字节和2字节数字的指令会保持剩下的字节不变；生成4字节数字的指令会把高位4字节置0.

2、传送指令不能都指向内存位置.

3、栈传递参数时，需要按8的倍数对齐，如栈分配局部变量(需要用栈)则按照一般的参数对齐原则.

考点，三 内存越界引用和缓冲区溢出

gets，strcpy，strcat，sprintf 都会导致缓冲区溢出，不进行边界安全检测的函数.

还会改变返回地址，从而改变程序分为.

考点，四 对抗缓冲区溢出攻击

1. 栈随机化 攻击代码在栈中放置时，由于栈随机化，会使改动代码所能运行的位置不确易于预测，因此会需要攻击者对可能的栈地址遍历.

2. 栈破坏检测 在分配缓冲区时，在其尾端设置"金丝雀"值，返回前检测该值是否改变.

3. 通过页表机制，限制不同区域页上的访问权限，

考点，五 变长栈帧. %rbp 为被调用者保存，先压栈，再保存当前栈顶值，之后 %rsp 可随意操作.

回顾 处理器体系结构

数点一：X86-64

1. 现状态：程序寄存器15个（无%r15），条件码CC（ZF|SF|OF）、PC，程序状态 stat、内存

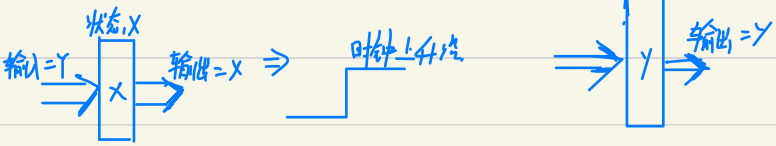
2. 改好内存地址 → 内存地址，立即数 → 内存的传送指令。

3. 小端法操作数。

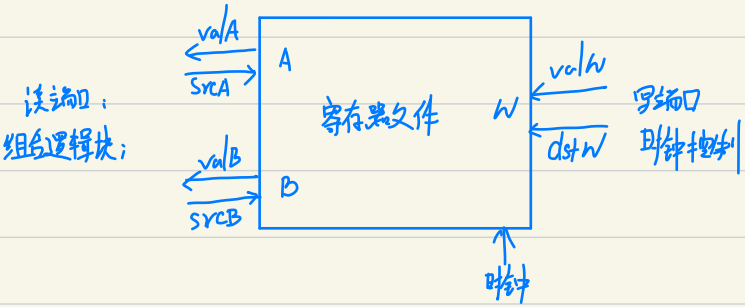
数点二：存储器操作

1. 硬件寄存器：作为电路不同部分中的组合逻辑之间的屏障。

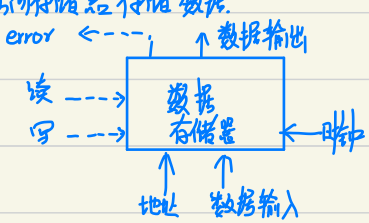
时钟信号 控制 状态值。时钟寄存器



2. 寄存器文件（寄存器阵列）



3. 随机访问存储器 存储数据。



读：组合逻辑

写，时钟控制

考点三 Y86-64 顺序实现

阶段	OPq rA, rB	rrmovq rA, rB	irmovq V, rB
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	icode: ifun ← M_1[PC] rA: rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10
译码	valA ← R[rA] valB ← R[rB]	valA ← R[rA]	
执行	valE ← valA OP valB Set CC	valE ← 0+valA	valE ← 0+valC
访存			
写回	R[rB] ← valE	R[rB] ← valE	R[rB] ← valE
更新PC	PC ← valP	PC ← valP	PC ← valP

阶段	rmmovq rA, D(rB)	mrmovq D(rB), rA
取指	icode: ifun ← M_1[PC] rA: rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10	icode: ifun ← M_1[PC] rA: rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10
译码	valA ← R[rA] valB ← R[rB]	valB ← R[rB]
执行	valE ← valB+valC	valE ← valB+valC
访存	M_8[valE] ← valA	valM ← M_8[valE]
写回		R[rA] ← valM
更新PC	PC ← valP	PC ← valP

阶段	pushq rA	popq 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] valB ← R[%rsp]	valA ← R[%rsp] valB ← R[%rsp]
执行	valE ← valB-8	valE ← valB+8
访存	M_8[valE] ← valA	valM ← M_8[valA]
写回	R[%rsp] ← valE	R[%rsp] ← valE R[rA] ← valM
更新PC	PC ← valP	PC ← valP

阶段	jXX Dest	call Dest	ret
取指	icode:ifun ← M_1[PC] valC ← M_8[PC+1] valP ← PC+9	icode:ifun ← M_1[PC] valC ← M_8[PC+1] valP ← PC+9	icode:ifun ← M_1[PC] valP ← PC+1
译码		valB ← R[%rsp]	valA ← R[%rsp] valB ← R[%rsp]
执行	Cnd ← Cond(CC, ifun)	valE ← valB-8	valE ← valB-8
访存		M_8[valE] ← valP	valM ← M_8[valA]
写回		R[%rsp] ← valE	R[%rsp] ← valE
更新PC	PC ← Cnd ? valC : valP	PC ← valC	PC ← valM

考点四 流水线冒险

1) 指令相关 {数据相关，下一条指令用到这条指令的计算结果
控制相关，一条指令要确定下一条指令的位置。例如，跳转，调用、返回。

2) 冒险 {数据冒险 转发来源元 {五个转发源 e_valE、m_valM、M_valE、W_valM、W_valE
两个转发目的 valA、valB。
控制冒险

加载/使用数据冒险：由于数据加载从内存中为来源时，访存阶段才能真正确定数据值。
暂停 + 转发 处理（加载互锁，会降低流水线吞吐）

控制冒险：（针对F阶段PC值）
(1) ret 所在的 在阶段结束 才知道下一条PC值，
下一条指令在F阶段确定PC值时，需要暂停3个周期（或者插入三个气泡）
(2) 预测错误分支：
默认执行下一条指令，但是真实情况在上一条指令的执行阶段结束 才知道
在取出的预测指令的下个阶段插入气泡，并在下条指令取出正确的指令。

条件	触发条件
处理 ret	
加载/使用冒险	
预测错误分支	
异常	

条件	流水线寄存器
处理 ret 1)	
加载/使用冒险 2)	
预测错误的分支 3)	
1)、3) 组合	
1)、2) 组合	

考点五 异常处理

Stat ADR, HLT, ADR, INS

异常指令之后的指令不应对程序可见的状态产生任何影响（停止更新条件码寄存器，数据内存）

五章 优化程序性能

考点一 编译器能力局限性

内存别名，对全局变量的修改二者不可预测，故编译器无法优化。（编译器只做安全的优化）

考点二 与机器无关的优化

1）消除循环的低效率 （与编译器无关）

方法：代码移动，将循环中需要执行多次但计算结果不会改变的部分移出循环。

2）减少过程调用 — 避免每个循环的边界检查

方法：改变过程调用的形式，将程序中的过程调用修改为更加直接的形式，

3）消除不必要的内存引用

方法：尽可能将计算结果放在累积变量中，以减少内存引用。

考点三 与CPU相关的优化（面向处理器） —— 核心是优化关键路径上的运算。

1）循环展开 k×1

本质上只是减少循环次数，关键路径计算并未减少

改进：k×1a （重新结合变换）

将一些可以在关键路径之外的计算结合。

2）多个累积变量 k×k

对延迟为L，容量为C的操作为言，要求循环展开因子 k ≥ C·L；

Tips：1）不要累积过多，以免寄存器溢出

2）分支预测 ① 不需要过分关心可预测分支；② 编写易于条件传送的代码（非重点）；

考点四（非重点）写/读相关（有空重做一下 P387）

六章 存储器层次结构

考点一：基本的存储技术

SRAM存储器，DRAM存储器，ROM存储器以及旋转的和固态的硬盘。

考点二：总线事务

关于电子电路在CPU与DRAM之间来回传送数据的一系列步骤；读、写事务；

考点三：磁盘。

1）磁盘容量

2）磁盘操作

访问时间 = 寻道时间 + 旋转时间 + 传送时间；

即 $T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$

3）访问磁盘 DMA

考点四：局部性（时间，空间）

步长为K的引用模式 关系 空间局部性；

引用相同变量具有时间局部性；

取指令来说，循环具有良好的局部性；

L_1，L_2，L_3 缓存由内置在缓存中的硬件逻辑管理；

DRAM 是由 OS 软件 和 CPU上的 MMU 共同管理；AFS之类的分布式文件系统的机点，由运行在本地机器上的AFS客户端进程管理；

考点五 高速缓存

1）$C = E \times B \times S = E \times 2^b \times 2^s$ 字节

2）$E=1$，组选择；行匹配；字抽取

（4，1，2，4）

冷不命中，冲突不命中，容量不命中；

3) 1 < E < C/B

组选择；行匹配；字选择.

3) E = C/B，快表（TLB页表项）。不存在冲突不命中.

考点六 自关写的问题

在高速缓存中修改了被缓存的字（写命中）：每个块有一个修改位，用来标志是否更新.

1）直写，立即将M的缓存块写回低一层；

2）写回，替换更新时，再写到低一层，

写不命中时，— 写分配（缓存），写回通常写分配.

— 非写分配（直接写到低层），直写通常为非写分配

考点七 性能影响 P439

考点八 命中率

如果一个高速缓存的块大小为B字节，那么一个步长为k的引用模式，平均每次迭代会有

min(1, (wordsize × k)/B) 次缓存不命中.

局部变量的反复引用，步长为1，都是良好的高速缓存友好代码.　　重新过一下 P.442 的题目.

考点九 重新排列循环

```
k i j
for(k=0; k<n; k++)
  for(i=0; i<n; i++){
    r=A[i][k]
    for(j=0; j<n; j++)
      C[i][j] += r*B[k][j];
}
```

```
i k j
for(i=0; i<n; i++)
  for(k=0; k<n; k++){
    r= A[i][k];
    for(j=0; j<n; j++)
      C[i][j]= B[k][j]*r;
}
```

上面两个都是以步长为1的模式访问 B、C 数组，

2 × wordsize / B

七章

卷一：编译系统各阶段

gcc -Og -o prog main.c sum.c
- cpp main.c /tmp/main.i 预处理形成源程序文本
- cc1 /tmp/main.i -Og -o /tmp/main.s 编译成为汇编语言文本
- as -o /tmp/main.o /tmp/main.s 汇编形成可重定位目标文件
- ld -o prog /tmp/main.o /tmp/sum.o 链接成为可执行文件prog

更多内容见 Xmind 的脑图

第十章

1. Linux shell 创建的每个进程都有3个打开的文件：STDIN_FILENO (0)、STDOUT_FILENO (1)、STDERR_FILENO (2)，

文件位置 k；EOF

2. 文件 int Open(char *filename, int flags, mode_t mode); flags 访问限制，mode 访问者的限制

int close(int fd);

读写 ssize_t read(int fd, void *buf, size_t n);

ssize_t write(int fd, const void *buf, size_t n);

读时 EOF，从描述符文本行 EOF 为行大小；

除了 EOF 读到最后，不会有不足值，写也不会；

fstat stat 文件元数据

RIO 无缓冲 I/O rio_readn, rio_writen

有缓冲输入 rio_readnb, rio_readlineb

初始化缓冲区 rio_readinitb

3. 目录 DIR *opendir(const char *name);

struct dirent *readdir(DIR *dirp);

int closedir(DIR *dirp);

3. 文件共享 描述符表 → 文件表 → v-node（所有进程共享）

父子进程共享

非父子进程的同一个文件单独有对应描述符的文件表。

4. I/O 重定向 int dup2(int oldfd, int newfd);

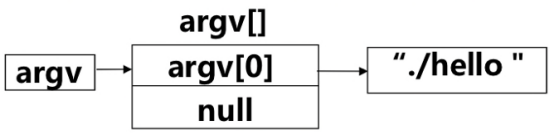
例：1对应文件A，4对应文件B，要将1重定位到4，则 dup2(4,1)，此时复制的是文件表中的指针内容（指针指向文件表）

程序的加载和运行

问题：hello程序的加载和运行过程是怎样的？

Step1：在shell命令行提示符后输入命令：$./hello[enter]

Step2：shell命令行解释器构造argv和envp



BACK

Step3：调用fork()函数，创建一个子进程，与父进程shell完全相同（只读/共享），包括只读代码段、可读写数据段、堆以及用户栈等。

Step4：调用execve()函数,在当前进程（新创建的子进程）的上下文中加载并运行hello程序。将hello中的.text节、.data节、.bss节等内容加载到当前进程的虚拟地址空间（仅修改当前进程上下文中关于存储映像的一些数据结构，不从磁盘拷贝代码、数据等内容）

Step5：调用hello程序的main()函数，hello程序开始在一个进程的上下文中运行。 `int main(int argc, char *argv[], char *envp[]);`