**计算机系统基础期末复习知识点整理**

Reisoul：黄色荧光标记的为考过的题目和最重要的知识点，根据22级和24级考题回忆而成

复习建议：至少考试前一周开始复习，从头到尾捋一遍

计算机系统基础重点考点总览（自查表）：

能全会就是无敌的，全不会也不影响，只要好好复习！（=￣ω￣=）

第一章：

1. 存储程序方式 P3

2. 冯·诺依曼机结构、基本思想 P3

3. 程序执行过程 P5

4. 时钟周期 P6 P19

5. MIPS

6. 从源程序到可执行文件 P9-10

7. ISA（指令集体系结构）P13

8. CPI P20

9. Amdahl Law 阿姆达尔定律 P23

第二章——数据的机器级表示：

1. 补码、反码、原码、移码

2. IEEE 754 浮点数的表示 有关

3. 有/无符号数的加减运算

4. 加法器各运算标志

第三章——程序的转换及机器级表示

1. 汇编代码的阅读

第四章——程序的链接

1. 强弱符号

2. 程序链接的过程

以下是正文笔记（手写于2024年计算机系统基础考前一周）

1. 冯·诺依曼机的五大功能部件及功能：

运算器

输入设备 控制器 输出设备

存储器

• 控制器：自动取出指令来执行

• 存储器：存放指令/数据

2. 可执行目标程序的生成流程：

编译

预处理

汇编语言程序（.s）

源程序（.i）

源程序（.c）

汇编

链接

可执行目标文件（.exe）

可重定位目标文件（.o）

3. 最重要的系统软件是操作系统和语言处理系统（编辑器、预处理程序、编译器、汇编器、链接器、解释程序等）

4. 计算机系统抽象层、计算机系统的层次化结构（24年在第一大题考了，要求写出至少4个）

* 应用（问题）
* 算法

软件层面

* 编程（语言）
* 操作系统/虚拟机
* 指令集体系结构（ISA）→计算机系统的核心部件
* 微体系结构（也叫微架构）

硬件层面

* 功能部件
* 电路
* 器件

5. ISA（指令集体系结构）是什么？

它是一种规约，规定了如何使用硬件。

它是对指令系统的一种规定或结构，具体实现的组织是微体系结构。

不同ISA规定的指令集不同，如 IA-32,MIPS,ARM等同一种ISA可以有不同的计算机组成（理解这句话就行，IA-32,MIPS,ARM有兴趣可以自行了解，作为拓展）

6. ISA规定的内容：（这个24年考了，在第一大题，写几条就可以）

• 可执行的指令的集合：指令格式、操作种类、操作数的规定

• 操作数的类型

• 寄存器组的结构：每个寄存器的名称、编号、长度、用途

• 操作数所能存放的存储空间的大小和编址方式

• 操作数在存储空间是大端存放还是小端存放

• 寻址方式

• 指令执行过程的控制方式，程序计数器，条件码定义等

7. 前端和后端是什么：（不考吧）

高级语言程序

前端，遵循语言规范

词法、语法、语义分析、中间代码生成

中间代码

后端遵循ISA规范和ABI规范（应用程序二进制接口）

目标代码生成及优化

目标代码

8. 计算机性能评价（会考的）

（1）计算机的两种性能：速度和时间（速度指吞吐率和带宽等概念，时间指响应时间、执行时间、等待时间或时延等概念）

（2）基本的性能评价标准是：CPU执行时间

CPU执行时间=总时钟周期数×时钟周期

（3）一些英文名词

• CPI: 执行一条指令所需的时钟周期数（对于一条指令来说是确定的，对于一台机器或者一个程序来说，是一个平均值）

• 时钟频率：就是CPU主频

• MIPS：平均每秒执行多少百万条（定点）指令

• MFLOPS：平均每秒执行多少百万次浮点运算（基于完成的操作次数而非指令数）

• GFLOPS：(10^9\)

• PFLOPS：(10^15)

• TFLOPS：(10^12)

• EFLOPS：(10^18)

一定要弄清楚数量级啊！两个舍友因为数量级搞错了，第二道大题第一问就算错了，之后四问都错了，痛失16分！！对小学生来说很简单，大学生一定要注意口牙！！！

（4）用基准程序评测计算机性能

（5）阿姆达尔定律（考计算题，做书上的例题就很容易理解了）

• 改进后的执行时间=（改进部分执行时间/改进部分的改进倍数）+未改进部分执行时间

• 整体改进倍数=1/[（改进部分时间比例/改进部分的改进倍数）+为改进部分的时间比例]

第二章——数据的机器级表示与处理

1. 补码——模运算（联想时钟，即老师PPT里举的例子）

（1）一个负数的补码=模-该负数的绝对值

（2）对于某一确定的模（这里以模为12举例，联想时钟的12小时计时法）

(x-y)取模后与(x+12-y)取模后相等 (y<=12)

2. 负数补码=正数补码各位取反，然后加一

3. LSB（Least Significant Bit）最低有效位

MSB最高有效位

4. 为什么用补码表示带符号整数？（补码表示的好处）

• 补码运算系统是模运算系统，加减运算统一

• 数0的表示唯一，方便使用

• 比原码和反码多表示一个最小负数

5. 一些补码表示法造成的迷惑判断题（全部顺利做对你就没问题辣！）

这个比较重要，考过

表达式 表达式的值（真为1，假为0） 原因

0==0U（无符号的0） 1

-1<0 1

-1<0U 0 表达式中既有无符号整数又有有符号整数时，C语言会把有符号整数-1类型转换为无符号整型，把111…111(32个1，即-1的补码，也是-1在计算机中存储的形式)解释为无符号整数（即2^32-1），表达式实际上变成了4294967295<0，结果是false，值为0

2147483647>-2147483647-1 1

2147483647U>-2147483647-1 0 表达式实际上是0111…111>111…111

2147483647>(int) 2147483648U 1 实际上是0111…111>1000…000(后者解释为负数)

(unsigned)-1>-2 1 比较无符号111…111和无符号111…110

6. 了解：在有些32位系统上，

-2147483648<2147483647的结果为0

原因：把直接写出来的数值当作无符号整型比较了

但是这样写：

Int i=-2147483648;

I<2147483647

或者写成：

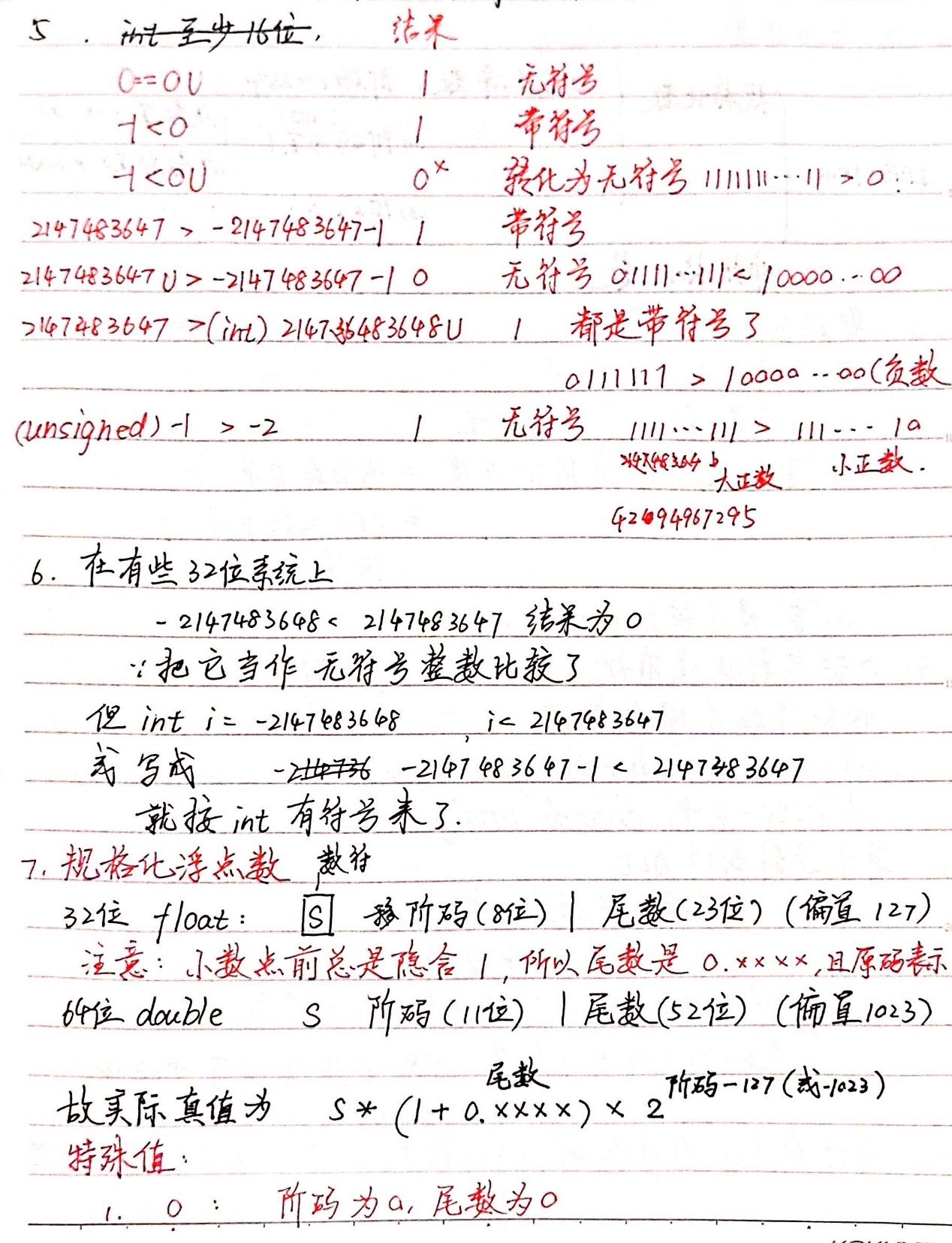
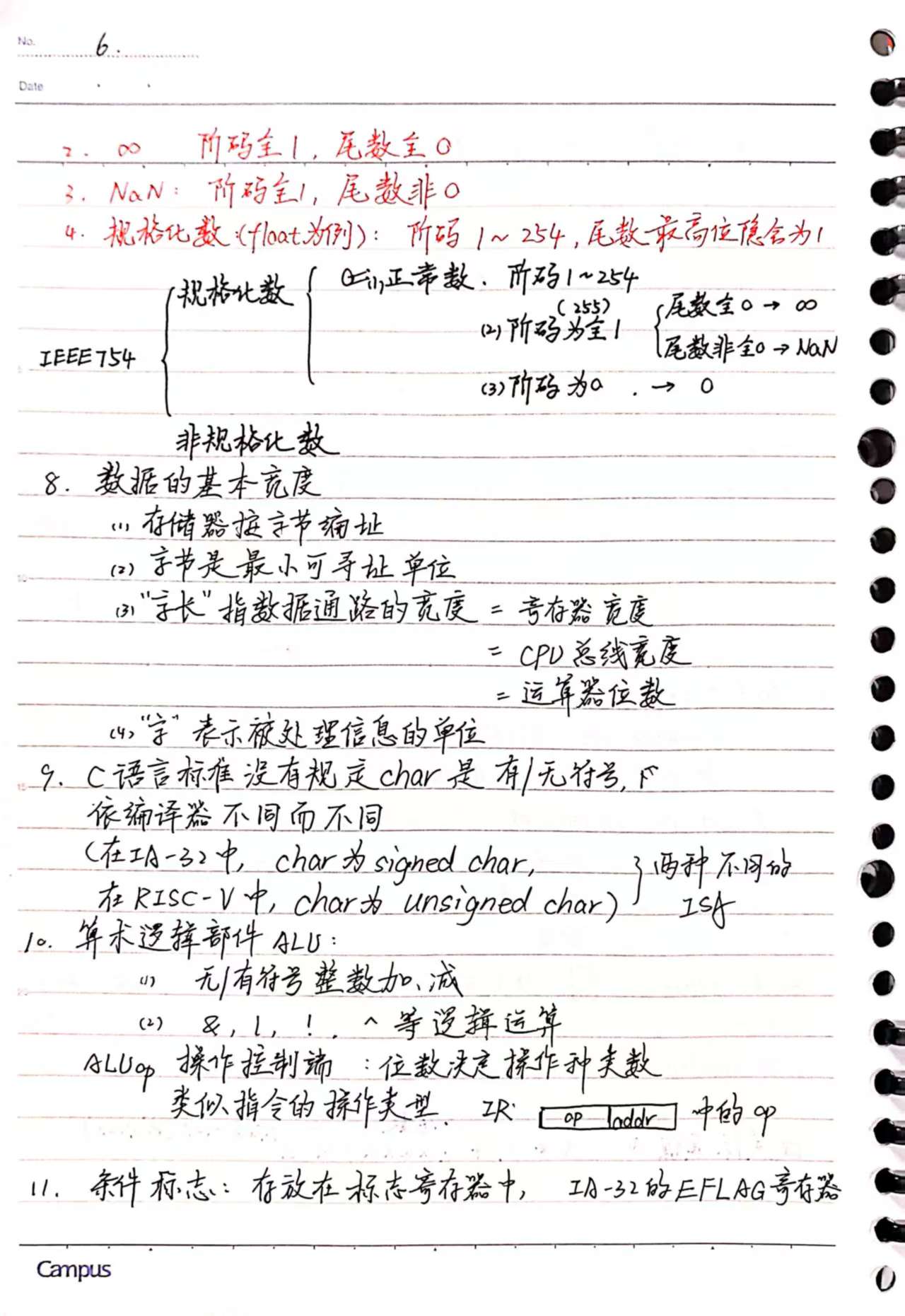
-2147483647-1<2147483647

结果就是1了，因为这样写，就都按int有符号整型比较了。

Reisoul：纯了解一下，考试考这个概率极低（太细了）

7. 浮点数

字丑见谅☹



关于考点：

1. 首先是让你把一个数据转换成float或者double类型，写出二进制表示法，极其重要，24年考了，分值不低。这里推荐直接看B站的教程，课本就别看了，而且考试应该只会考规格化浮点数，转换方法有不止一种，请务必选择一种适合自己的方法理解，不要看太多种方法，会搞混的！
2. 只要能作对课后规格化浮点数的练习题就过关！
3. 考试的时候题干给出了阶码有几位，尾数有几位了

8. 数据的基本宽度，字长，字节和字究竟是什么？（不考吧，只是感觉挺有意思就记下来了）

(1) 存储器按字节编址

(2) 字节是最小可寻址单位

(3) “字长”指数数据通路的宽度 = 寄存器宽度 = CPU总线宽度 = 运算器位数

(4) “字”表示被处理信息的单位

9. C语言标准没有规定char是有/无符号，依编译器不同而不同（这样的一般也不考）

（例如在IA-32中，char为signed char，在RISC-V中，char为unsigned char）

10. 算术逻辑部件ALU的功能（不考吧）

(1) 无/有符号整数加、减

(2) &，|，！，^ 等逻辑运算

ALUop 操作控制端：位数决定操作种类数

类似指令的操作类型

IR寄存器（Instruction Register）op addr 中的 op

Reisoul:我的笔记是这么写的，可我不知道是什么意思了（捂脸），我有空再查查，不会考的

11. 整数的乘运算（没见考）

(1) 无符号：若高n位全0，则不溢出，否则溢出

(2) 有符号：若高n位全0或全1且等于低n位的最高位，不溢出。

12. 整数除法向0舍入

13. 变量与常数之间的除运算，朝0舍入

14. 带符号负整数朝0舍入原理：x / 2^k

先加偏移量（2^k-1），然后右移k位，低位截断

e.g. -14 / 3 = -3

15. 浮点数加减运算

(1) 对阶：阶码小的数尾数右移，将隐含的1也移到小数，移出的低位保留到附加位

(2) 加减

(3) 变为规格化

(4) 判断是否溢出

16. 为何IEEE754加减运算右规时又需要一次？（不考吧）

答：即使两个最大的尾数相加，比如二进制1.111…+1.111…得到的和的尾数是11，故尾数的整数部分最多有两位，保留一个隐含的1后，最多只有一位被右移到小数部分。

(5) 附加位 舍入

尾数保护位 | 附加位

附加位：01 舍

11：入

10：强迫结果为偶数？

e.g. 一个实数值赋给float和double，输出结果不同

float保留7位精确数字，其它位舍入后，可能变大也可能变小。

当结果为 1x10^-127时，不会用规格化浮点数表示（因为这样的话，加上偏置127，阶码是0，把整数部分的1隐藏后，尾数为全0，根据之前的知识，这个是0），而是用非规格化浮点数表示，不是近似表示为0（这个太难了，应该不考）

17. long double 在 IA-32中是 80位扩展精度格式（不考）

18. 强制类型转换的结果： （最好知道）

int → float 不会溢出，但可能舍入（float尾数才23位）

int/float → double 精确值

double → float/int 可能溢出，可能舍入

float/double → int 可能向0截断（当数值不是整数时）

19. 浮点数加法结合律不一定正确（不考）

20．寄存器标志位的判断：

24考了一道题，让你举出一个例子，使得OF和CF有相应的变化，具体忘了（捂脸），但是我以为不会考的，就没复习，结果也不会☹。

知道什么情况下OF，CF，ZF，SF被设置

关于这里，推荐看 BV1Jf4y1y7Nd

第三章.程序的机器级表示

1. 指令的组成

2. 一些汇编指令的含义：

注意：可以不用背，考试的时候在最后会给出对应的意思，而且考的都是简单的指令（我当时不知道，就都在笔记里写了）

以下是一些不常见的指令（考的概率极低）：

INC 增一（影响除CF以外的标志，不区分带/无符号）

DEC 减一（影响标志，若对0取反，则结果为0且CF=0，否则CF=1）

MUL 无符号乘（不区分带/无符号）

IMUL 带符号乘

乘法指令可以有1/2/3个操作数

DIV 无符号除

IDIV 带符号除

逻辑运算：

NOT 非

AND 与

OR 或

XOR 异或

移位运算：SHL/SHR 逻辑左/右移

SAL/SAR 算术左/右移（符号位发生变化，OF=1）

RET：从栈中取返回地址（记为RA），接着程序转到RA处执行

3. 过程调用：（P为调用者，Q为被调用者）（理解这个过程）

(1). P将入口参数（实参）放到 Q能访问到的地方

(2). P保存返回地址，然后将控制转移到 Q（CALL指令）

(3). Q保存 P的现场，并为自己的非静态局部变量分配空间

(4). 执行 Q的过程体（函数体）

(5). G恢复 P的现场，释放局部变量空间

(6). Q取出返回地址，将控制转移到 P

调用者保存寄存器（P要把这里面的值保存）

• EAX、EDX、ECX这几个寄存器Q可以直接使用

被调用者保存寄存器：

• EBX、ESI、EDI Q必须先将它们的值保存到栈中再使用

4. 一个 C 过程：(理解这个过程)

(1). 准备阶段：

• 形成帧帧底 push/mov

• 生成栈帧 sub/and

• 保存现场 push

(2). 过程体：

• 分配局部变量空间，并赋值

• 具体处理逻辑，如果遇到函数调用时

• 在EAX中准备返回值

(3). 结束阶段：

• 退栈

• 取返回地址（下一条要执行的指令地址）

5. IA-32/Linux的存储中

• 只读代码段一般为 0x8048xxx（不需要背，只是老师提了一嘴）

• 栈区地址一般为 0xbffffxxx（不需要背，只是老师提了一嘴）

• 全局变量和静态变量放在可读写数据区

6. 注意：C语言中的函数调用就是过程调用

• 传递参数时从右向左依次压栈

• 全局静态数据区为 0x8049xxx（不需要背，只是老师提了一嘴）

7. 数组：

声明全局数组：

int buf[2]={10,20} 08049908<buf>:`

框内为内存中的样 08049908 0A 00 00 00 14 00 00 00

低地址放前面的元素（千万不要搞错了，有一道大题要填空，给你一个栈，让你放对应地址是什么东西，一旦记错了，全错）

auto 型数组时，分配在栈中

注意： &A[i] - A = (&A[i]) - A

8. 联合体（不要求）

• 数据对齐

9. 显示"Segmentation fault"原因（知道即可，做实验可能出现得比较多）

• ret 指令取到的返回地址跳转到数据区或系统区或其他非法访问的存储区进行执行，造成段错误

10. 缓冲区溢出攻击的防范：（不考的吧）

（1）程序员：使用辅助工具查错

（2）编译器和操作系统：

（3）地址空间随机化

（4）栈破坏检测

（5）可执行代码区域限制：将动态的栈段设为不可执行，防止攻击者执行被植入在输入缓冲区的代码。

第四章 程序的链接

1. 子程序（函数）起始地址和变量起始地址是符号定义和符号引用，调用子程序（函数或过程）和使用变量最终必须链接（合并），合并时须在符号引用处填入定义处的地址。

2. 使用链接的好处：

• 模块化、效率高

• 分成很多源程序文件

• 可构建公共函数库

• 时间上可分开编译，只需重新编译被修改的源程序文件，然后重新链接。

• 空间上无需包含共享库所有代码，源文件中无需包含共享库函数的源码，只需包含所调用函数的代码，不需要包含整个共享库。

3. 链接操作的步骤：（知道这四个步骤名就差不多了）

• 符号绑定：确定标号引用的关系（符号解析）

• 合并相关文件

• 确定每个标号的地址

• 重定位：修改引用（指令中填入新的地址）

4. 符号解析：

• 定义的符号存在一个表里，叫符号表，是一个结构数组，每个表项包含符号名、长度和位置等信息。

• 链接器将每个符号的引用都与一个确定的符号定义相关联。

• 符号解析：将每个模块中引用的符号与某个目标模块中符号绑定的定义符号建立关联，以便在重定位时替换地址

5. 重定位：合并多个代码段和数据为单独的代码段和数据，计算出每个定义的符号在虚拟地址空间中的绝对地址，将可执行文件中符号引用处的地址修改为重定位后的地址。

6. 可执行目标文件与可重定位目标文件的区别：可重定位目标文件中的地址都从0开始，待定位；前者可的地址已经被计算出来了，但仍然是虚拟地址空间的地址。

7. 共享的目标文件：共享库文件，特殊的可重定位目标文件，能在装入或被链接运行时被装入到内存并自动被链接。

8. “链接视图和执行视图”

• 节是ELF文件中具有相同特征的最小可处理单位。

• 节组成：.text、.data、.rodata（代码、数据、只读数据）

• 有节头表

• 段组成，可多个节映射到同一段

• 描述节如何映射到存储段中，如：.data节、.bss节平

• 有程序头表、节头表（可选）

• 在只读数据节中有poff(x=%d^"",x=%d)的x=%d，有switch的跳转表。

9. 程序放在主存里（第一章提到过）

• 节头表在可重定位目标文件中，描述每个节的节名、在文件中的位置。

链接器的作用与符号表

• 链接的本质：合并相同的节

10. 符号表中有三种链接器符号：（24没考）

* 模块内部定义的全局符号
* 外部定义的全局符号
* 本模块的局部符号（半局部变量）

• symtab节记录符号表信息：

• value：对应节中的偏移量

• ndx：在第几节（符号在`.text`/`.data`节...）

• 若符号不是数字，则UND表示未定义，`COM`表示未初始化

• 未初始化的全局变量名是弱符号

• `static int a`不是强符号，而是本地局部符号

11. 多重定义符号

1. 强符号不能多次定义，否则出错

2. 强弱并存，以强符号为准

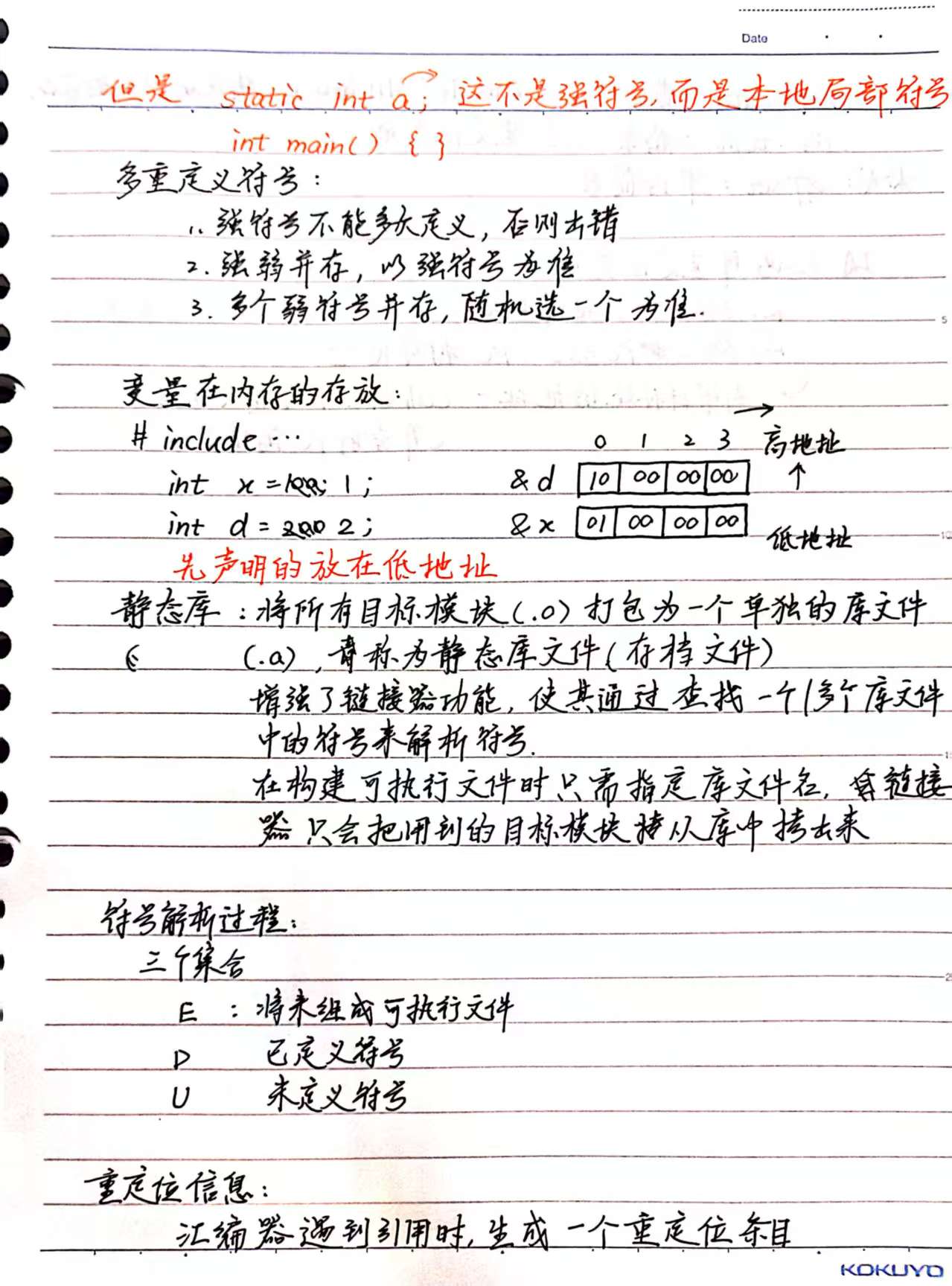
3. 多个弱符号并存，随机选一个为准

12. 变量在内存的存放

先声明的放在低地址(24考了，好几个空)

而且会和大小端存放一起考，记住，我们考的都是小端存放！！！（低有效位放在低地址处！）

例如这段代码：

#include……

int x = 1;

int d = 2;

13. 静态库（24没考）

• 将所有目标模块（.o）打包为一个单独的库文件（.a），称为静态库文件（存档文件）

• 增强了链接器功能，使其通过查找一个（多个）目标文件中的符号来解析符号

• 在构建可执行文件时只需指定库文件名，链接器只会把用到的目标模块从库中拷出来

14. 三个集合：（不考）

• E：将来组成可执行文件

• D: 已定义符号

• U: 未定义符号

15. 重定位信息：（不考）

汇编器遇到引用时，生成一个重定位条目

16. 重定位类型(24没考，实验会用)

• `.rel-data`：数据

• `.rel-text`：指令

• 格式：`offset`：节内偏移

17. 两种重定位类型

1. `R\_386\_32`：绝对地址

2. `R\_386\_PC32`：PC 相对地址

=引用目标处的地址-CALL 指令下条指令地址（即当前 PC 的值）（相当重要，24考了，实验经常要用到）

Reisoul：

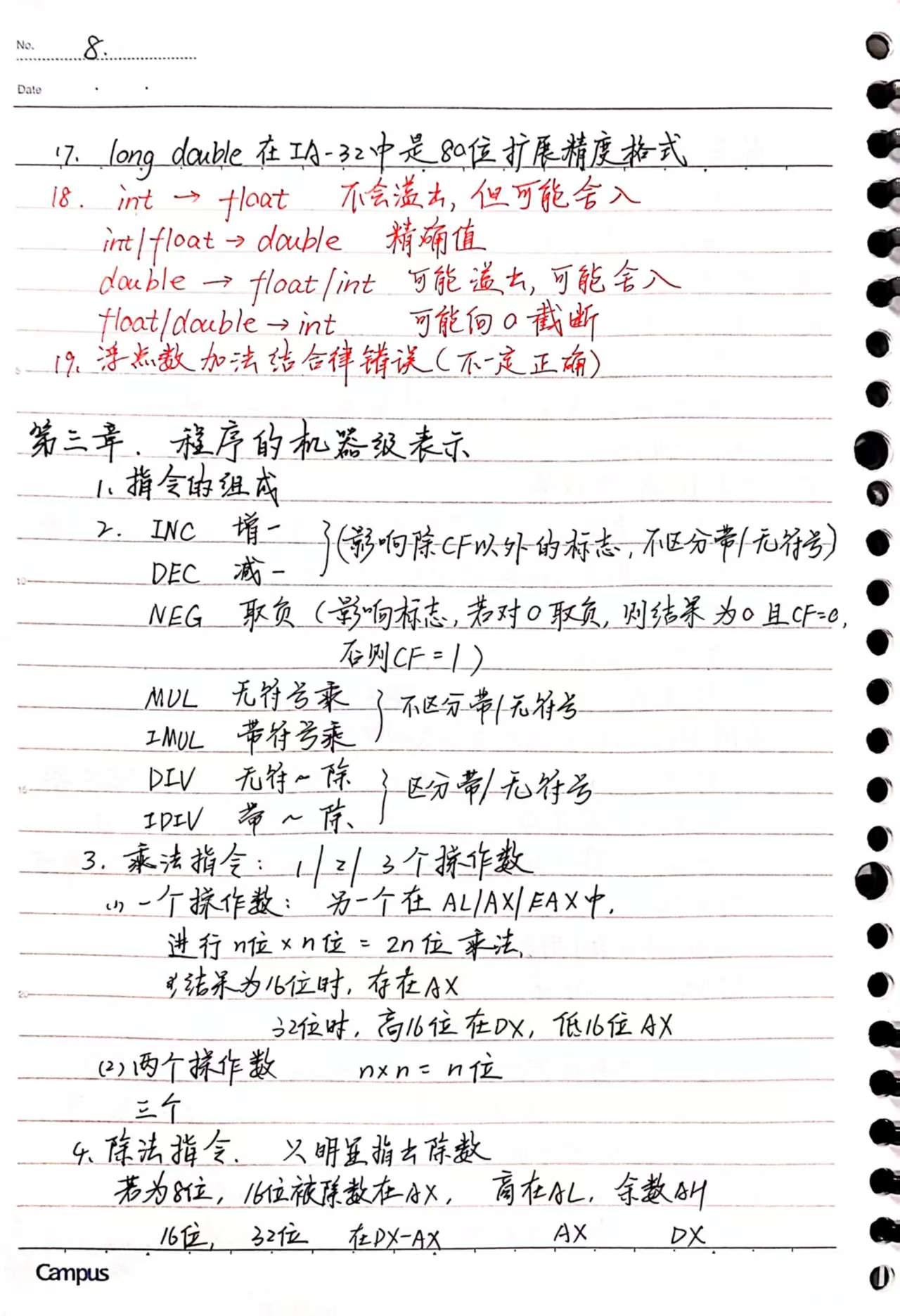
到这里就基本结束啦！恭喜你已经把计算机系统基础认真复习了一遍！

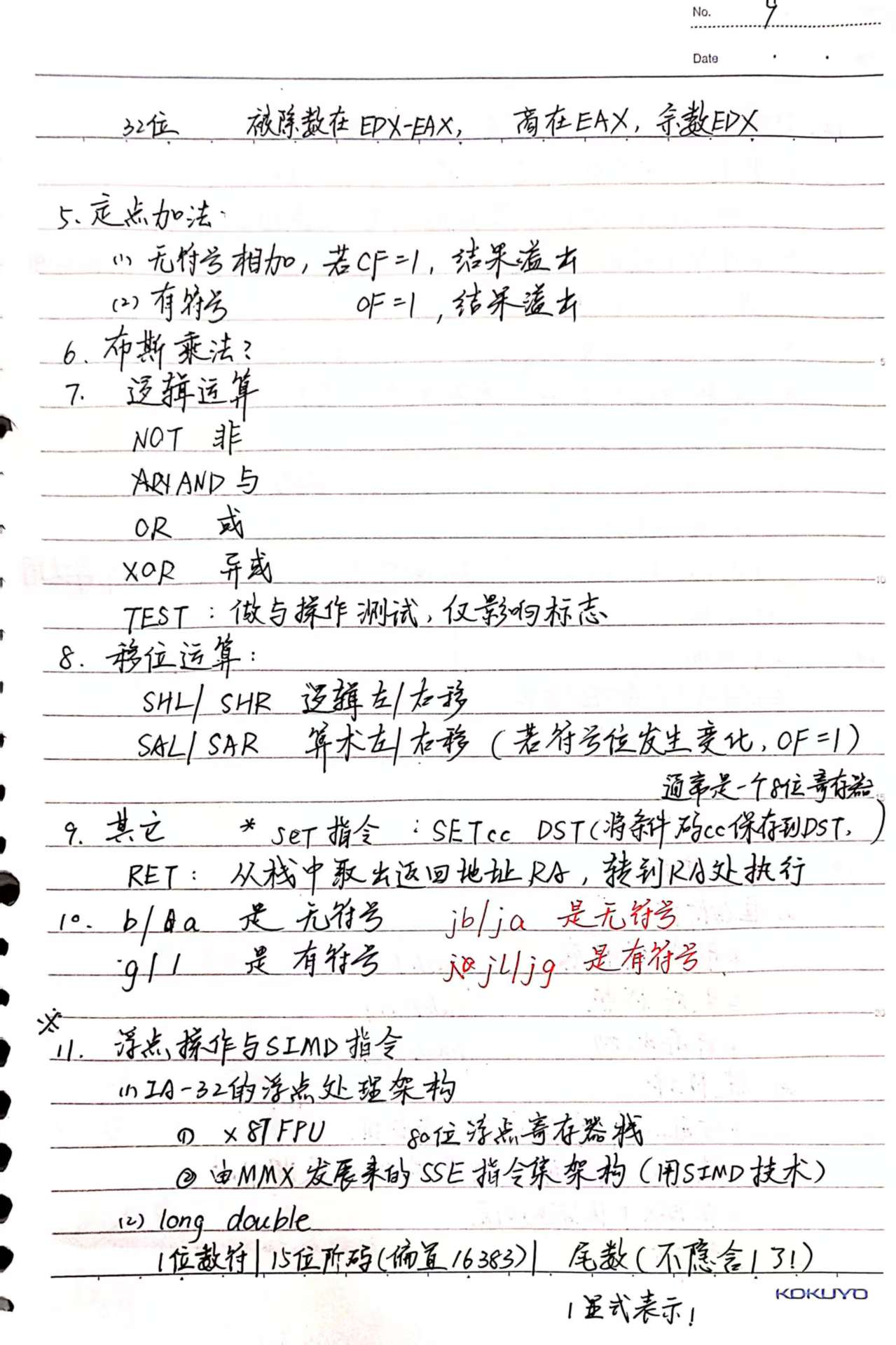
你是认真看到这里的对吧，盯~~(・∀・(・∀・(・∀・\*)

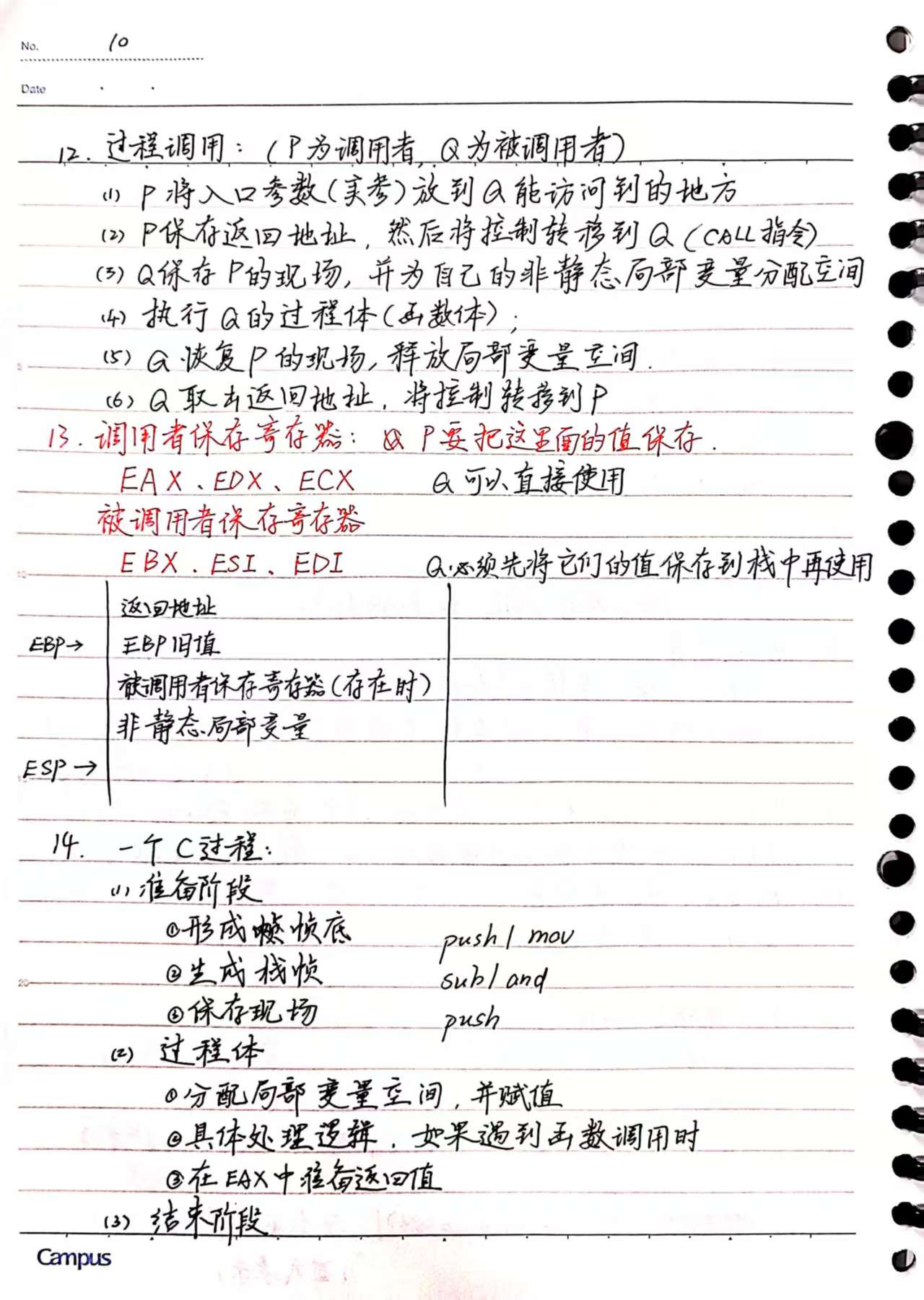
那么就祝你考的顺利，学习天天开心！

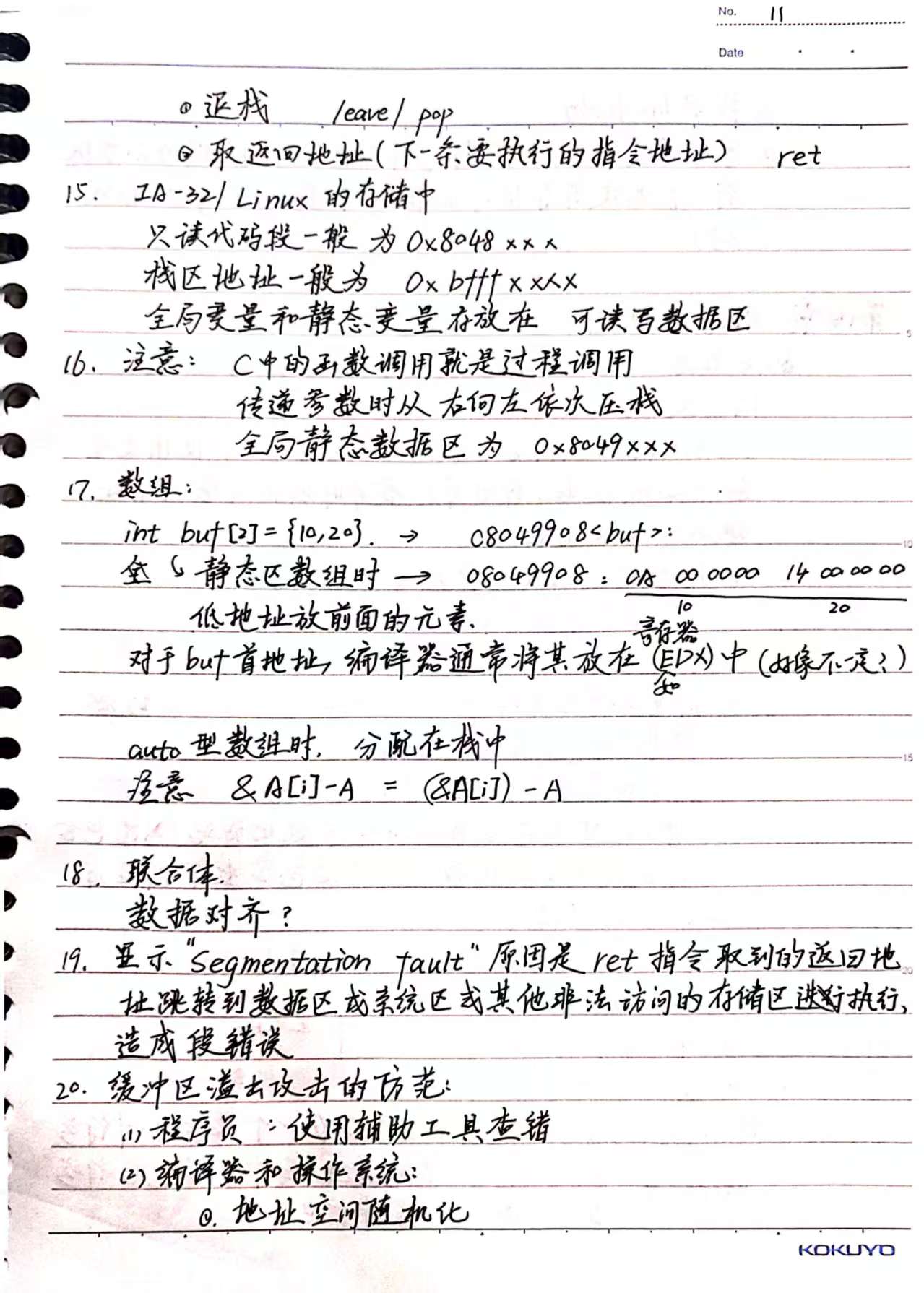
**d=====(￣▽￣\*)b**

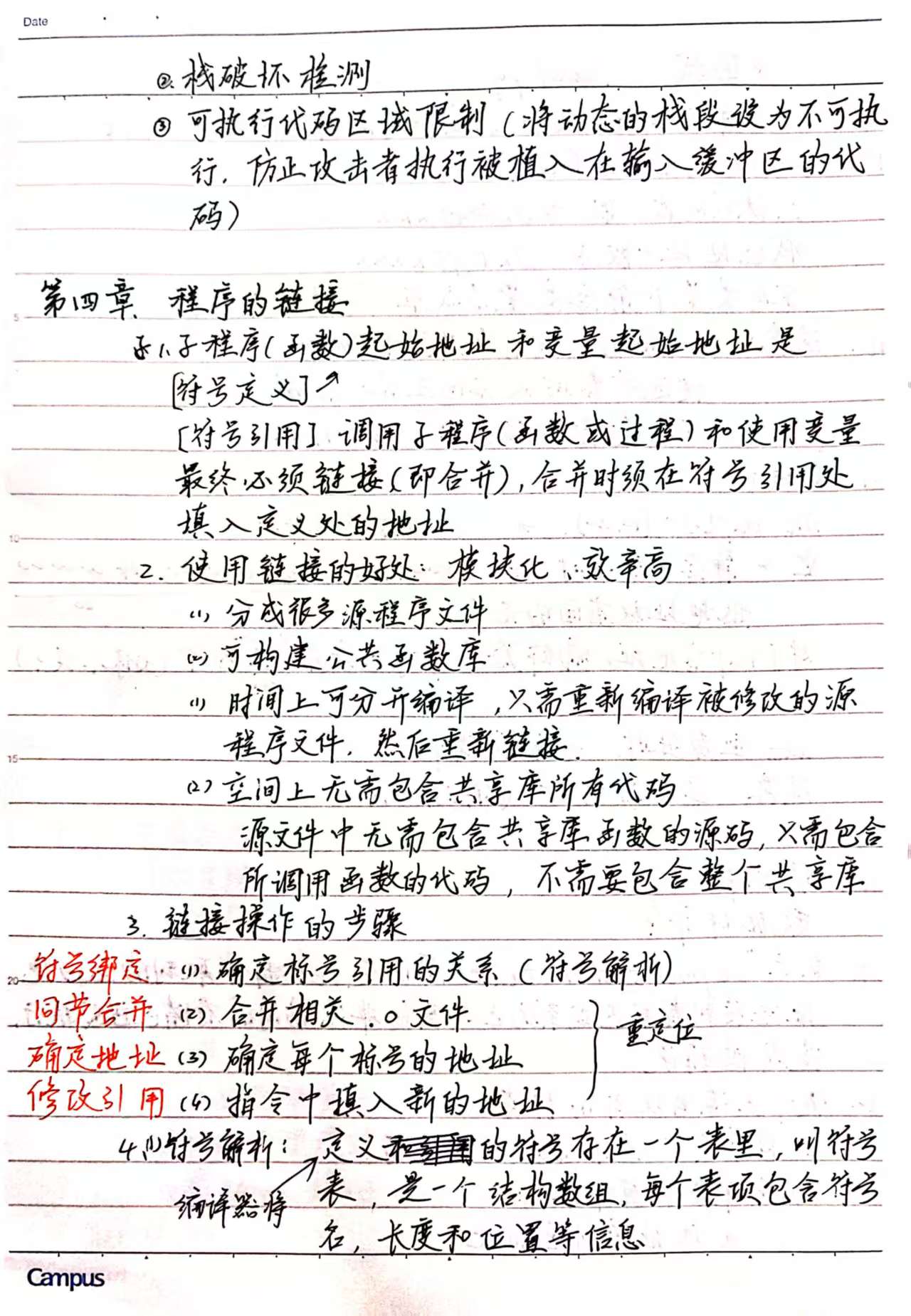
以下为第三章和第四章的手写版本❤

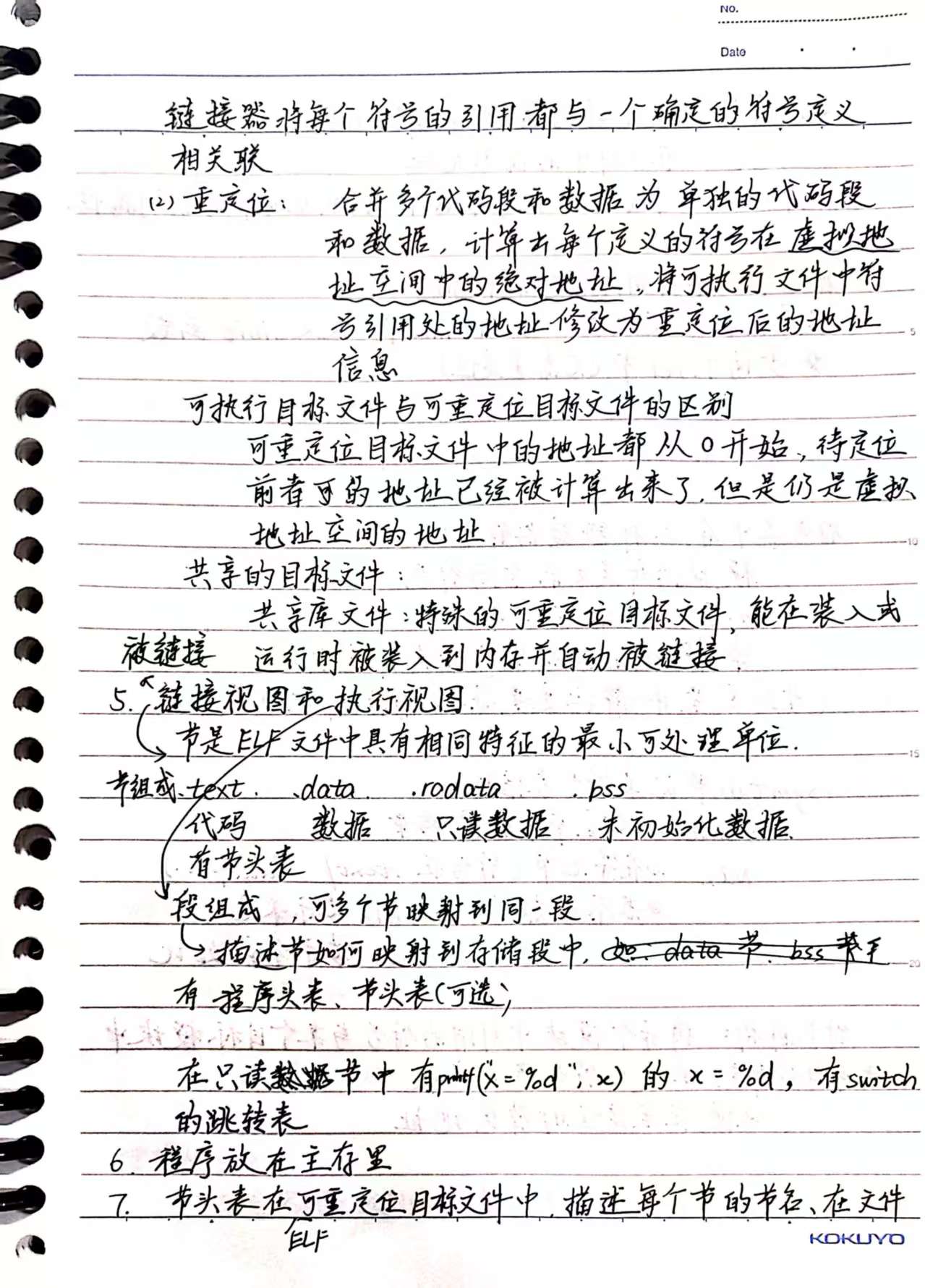


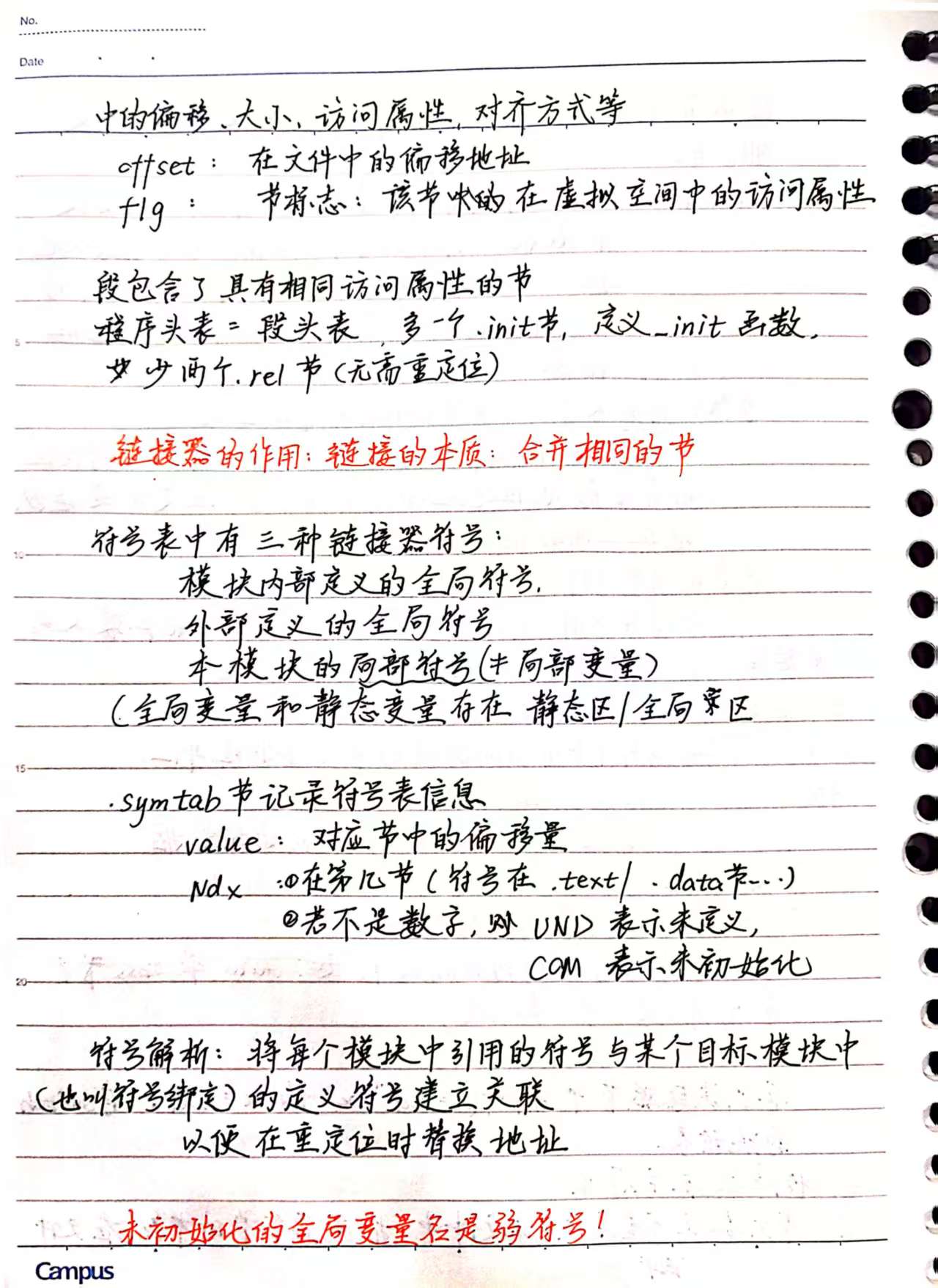


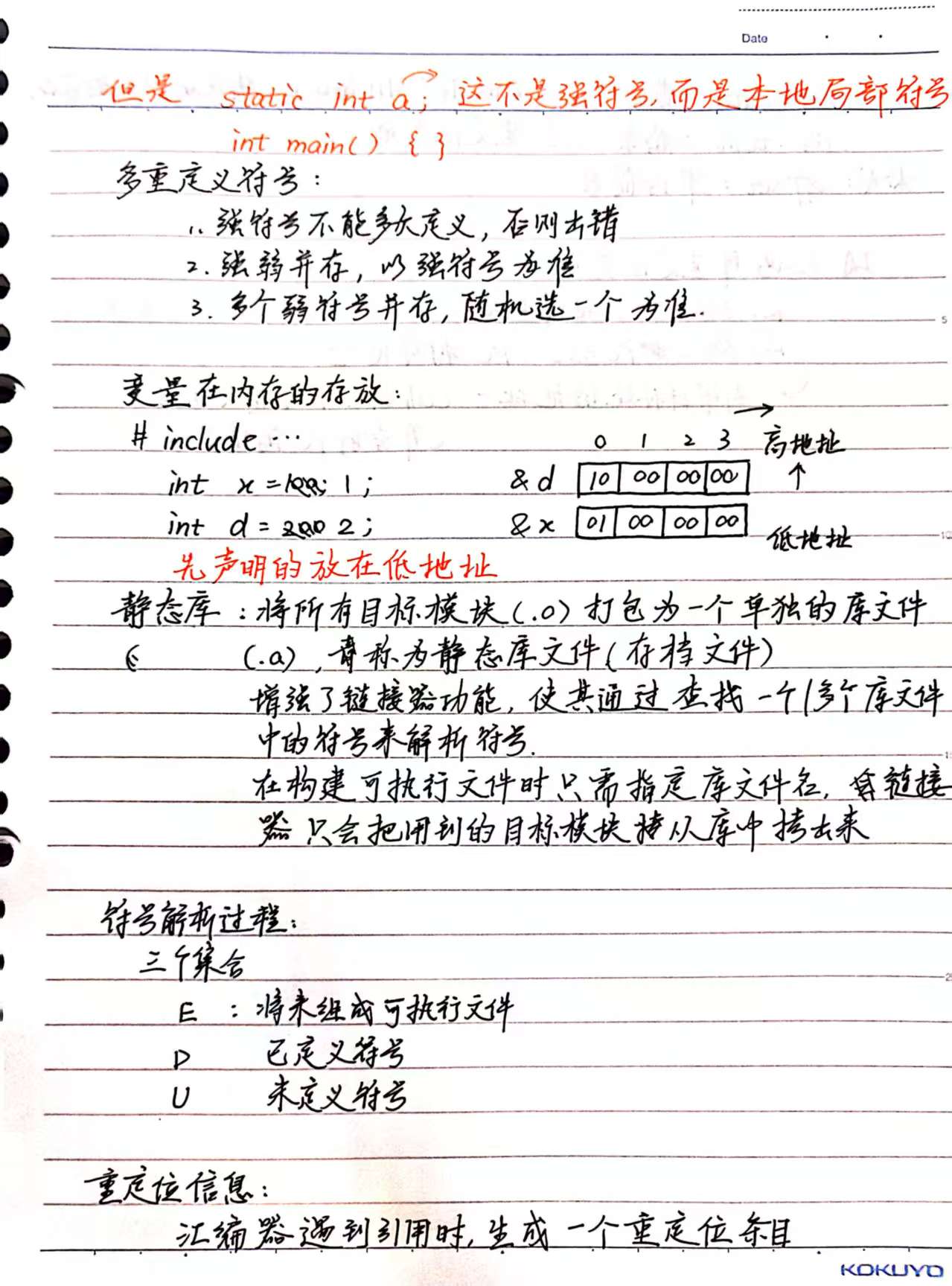


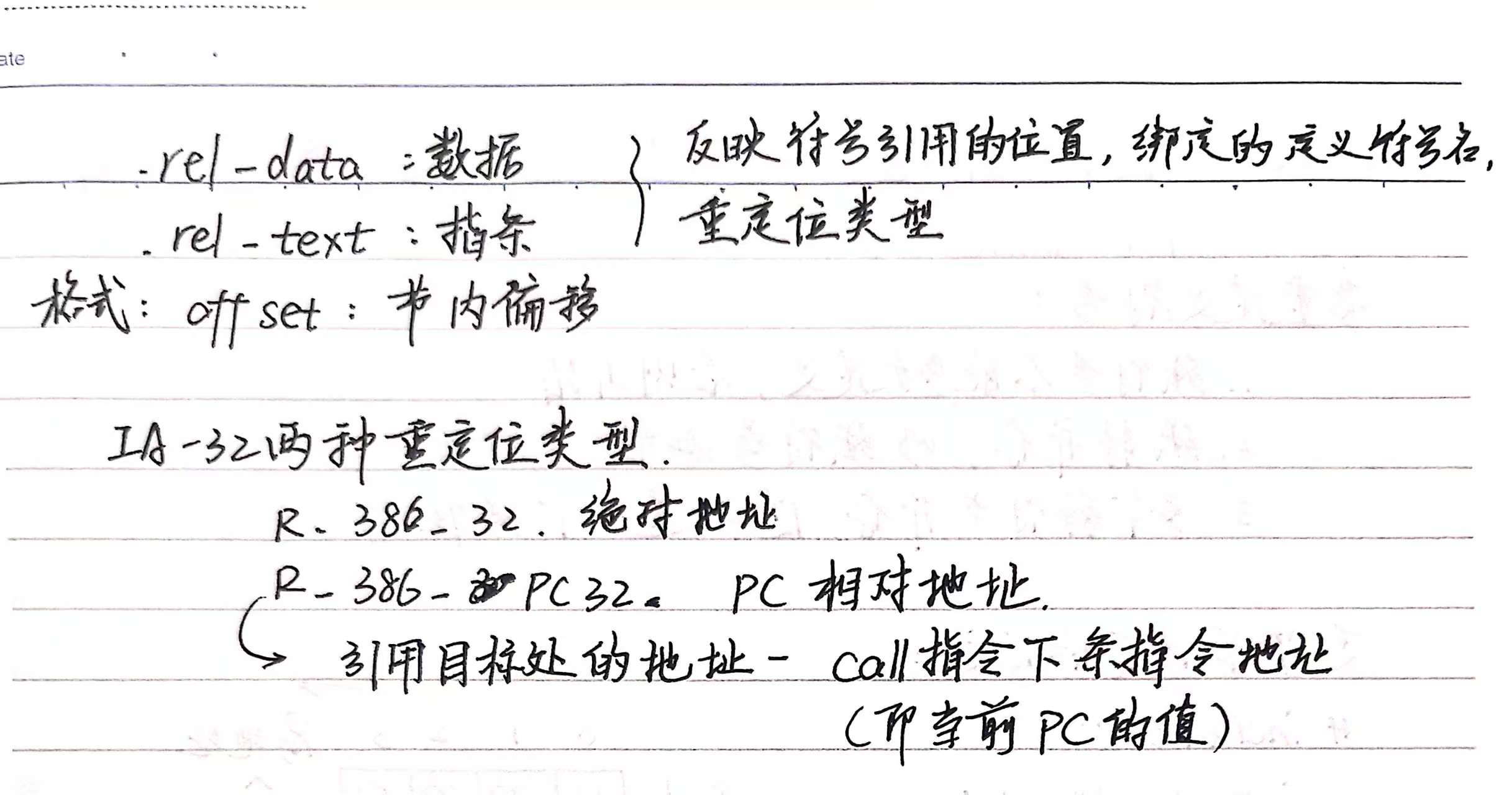












**看到这里了吗？！**

**再次祝考试顺利，学的开心**

**o(\*￣▽￣\*)ブ**