No.

Date . .

计算机硬件基础 01绪论

1. 发展历程 EDVAC第一台通用计算机 、IAS计算机

- 冯·诺依曼结构 "存储程序"工作方式 二进制表示
  - ↳运算器、控制器、存储器、输入设备、输出设备
  - 冯：程序和原始数据存一起、共享总线
  - 哈佛：独立，且有2总线

2. 程序编译与加载

.C $\xrightarrow[cpp]{\text{预处理}}$ .i $\xrightarrow[cc1]{\text{编译}}$ .S $\xrightarrow[as]{\text{汇编}}$ .O $\xrightarrow[ld]{\text{链接}}$ 可执行目标程序

插入头文件等　汇编　机器语言指令(二进制)　链接合并一些库函数等

- 加载：加载器把可执行文件装载到内存中.

3. 程序执行的基本过程

处理器：CISC 复杂指令，一个指令多步骤
RISC 精简指令 (MIPS)
↳让编译器搞

机器指令：操作码 + 操作数地址
(操作性能和质)

4. 计算机层次结构

- 运算器　完成基本的逻辑运算.
- 存储器　二进制保存程序和数据
- 控制器　提供各部件工作所需的控制信号，控制其他部件协同工作
- 编程语言：机器语言、汇编语言、现代高级语言

ISA：软件/硬件的接口，计算机组成的抽象 → 是ISA具体实现，怎么设计

ABI：运行在特定ISA的应用程序遵循的一种接口规约
↳应用程序与操作系统间的接口，程序、系统如何交互

5. 资源抽象：一种思维、技术手段 → 向上层提供统一、简洁的逻辑资源

eg. C函数原型、Java类声明

- 进程：文件 → I/O设备　虚拟内存 → 程序存储器
  进程 → 正在运行的程序
- 处理器：指令集 → 真实处理器硬件
- 虚拟机：对整个计算机的抽象.

No.

Date

6. 重要概念

- 并发：逻辑上的并行，物理上交替执行（单核）；多线程、多进程、超线程
  并行：物理上的并行（多核）
- 线程级并行：单/多核　　指令级并行：流水线技术
  超标量并行：多条流水线并行执行　　SIMD　MIMD（都是并行）
- Amdah（阿姆达尔定律）

$$\text{改进后执行时间} = \frac{\text{改进部分执行时间}}{\text{改进部分改进倍数}} + \text{未改进部分执行时间}$$

$$\text{整体改进倍数 } p = \frac{1}{\text{改进部分时间比例 } t / \text{改进倍数 } n + \text{未改进部分时间比例}(1-t)}$$

$$= \frac{1}{\frac{t}{n}+1-t}$$

（能判断改进后最强整体性能大小）

No.

Date

02 数制与运算

一、定点数表示及运算

1. 数值数据表示三要素：进位记数制，定、浮点表示，如何用二进制编码

2. 十进制 → 二进制　　(反方向为 $x_i \cdot 2^n$ 累加，个数0，左加右减)

整数部分：除2取余　　小数部分：乘2取整，顺序排列 (不一定乘得尽)

- 二进制 ↔ 十六进制　　4位二进制数 对应 1位十六进制数.
  - 二→十六：每4位一组，用对应1位十六进制代替

    $(1101\ 0011.1101\ 101)_2 = (D3.DA)_{16}$

    若前不足4位，前补0　　不足4位，后面补0
  - 十六→二：直接替换

3. 无符号数与有符号数
   - 原码：最高位符号位(0正1负)　存在两种零
   - 反码：正数与原码相同，负数为正数取反，仍存在两种0 (很少用)
   - 补码：反码基础上负数加1　零唯一　[-128, 127]
   - 移码：将负数区间+K平移至正数，常用 $K=2^n$　(-5 → 123)

     无符号位，用于表示符点数

补码运算规则：补码符号位和数值一起参与计算，舍弃进位

$[A+B]_{补} = [A]_{补} + [B]_{补} \pmod{2^n}$　　$[A-B]_{补} = [A]_{补} + [-B]_{补} \pmod{2^n}$

$= [A]_{补} + [\overline{B}_{补}] + 1 \pmod{2^n}$

二、浮点数表示及运算　用移码　原码/补码

1. 一般表示法：分为阶码和尾数两个部分

如：$(178.125)_{10} = (10110010.001)_2 = 0.10110010001 \times 2^{01000}$

阶码：01000　尾数 0.1011001000

表示：J_0 $J_1J_2\dots J_m$ S_0 $S_1S_2\dots S_n$

J_0：阶符　$J_1J_2\dots J_m$：阶码数值　S_0：数符　$S_1S_2\dots S_n$：尾数数值

(移码不需要)

No.

Date

2、IEEE 754标准

数符S：0正数 1负数

阶码E：移码表示，偏移 $2^{n-1}-1$

尾数M：尾数必须规格成小数点左侧为1，

eg. $M=1.m$ 且小数点前的1作为隐含位被省略

- 单精度：数符1位 阶8位 尾数(m)23位
- 双精度：数符1位 阶11位 尾数52位

表示公式：
- 单：$(-1)^S \times 1.m \times 2^{(E-127)}$ 可精确7位(十进制) 2^{24}个
- 双：$(-1)^S \times 1.m \times 2^{(E-1023)}$

• IEEE约定：

E	M	浮点数N
$1 \le E \le 254$	$M \neq 0$	$(-1)^S \times 1.m \times 2^{E-127}$
$E=0$	$M=0$	表示N=0
$E=0$	$M \neq 0$	$(-1)^S \times 0.m \times 2^{-126}$ (非规范浮点数)
$E=255$	$M=0$	表示无穷大(±)
$E=255$	$M \neq 0$	不是一个数

规范化浮点数最大值 $1.111\ldots1 \times 2^{127} = (2-2^{-23}) \times 2^{127}$

• 转化IEEE 754浮点数

eg. $(178.125)_{10} = (10110010.001)_2$ 先转为2进制

$= 1.011\,0010001 \times 2^{(111)_2}$ 小数点移到左边一个，移次数为幂

$\Rightarrow S=0$ $E = 00000111 + 01111111 = 10000110$

真实 + 偏移量

$m =$ 01100100010000000000000

小数点后第一位 后补0

• 舍入方式 四种方式 ①就近舍入 非中间值：0舍1入 下位是0 下位是1且后面不全为0 只看最后2位

中间值：强迫为偶数

01舍 11入 10(中间值)：强迫偶数(看是舍/入为偶数)

②朝+∞ ③朝-∞ ④朝0方向

Campus

校验码：奇偶校验，让数据里"1"个数是奇/偶数
校验和

3. 数据的大小端存储　大端存储：最高位存在低地址处
小端存储：最低位存在低地址处

三、非数值数据的表示
1. 数据的纠错：增加若干位校验码，以检错或纠错

四. 布尔代数简介.　逻辑变量集.
1. 逻辑代数　$L=\{K, \vee, \wedge, \neg, 0, 1\}$
• 基本概念：逻辑变量　A. B ...　(反变量) $\bar{A}$. $\bar{B}$ ...
逻辑常量　0.1　逻辑运算　$\wedge$ $\vee$ $\neg$
• 或　与　非

2. 公理、定理、规则　• 与　+或　$\bar{A}$非　⊕异或　⊙同或
• 公理. 交换律　结合律
分配律　$A\cdot(B+C)=A\cdot B+A\cdot C$　$A+B\cdot C=(A+B)\cdot(A+C)$
0-1律　$A+0=A$　$A\cdot 1=A$　$A+1=1$　$A\cdot 0=0$
互补律　$A+\bar{A}=1$　$A\cdot\bar{A}=0$
• 定理　①重叠律 $A+A=A$, $A\cdot A=A$　②吸收律 $A+A\cdot B=A$, $A\cdot(A+B)=A$
③还原律 $\bar{\bar{A}}=A$　④反演律：$\overline{A+B}=\bar{A}\cdot\bar{B}$　$\overline{A\cdot B}=\bar{A}+\bar{B}$
⑤包含律　$A\cdot B+\bar{A}\cdot C+B\cdot C=A\cdot B+\bar{A}\cdot C$
$(A+B)(\bar{A}+C)\cdot(B+C)=(A+B)\cdot(\bar{A}+C)$
⑥吸收律　$A+\bar{A}B=A+B$
• 规则：①代入定理：把某个变量全部换成一个式子，仍成立
②反演定理："·"、"+"互换，"0"、"1"互换，原、反互换，$\bar{F}=F$
(不属于单个变量的非不变)
③对偶定理："·"、"+"互换，"0"、"1"互换，若$F=G$，则$F^*=G^*$

No.

Date . .

3. 逻辑函数表达形式

- 逻辑函数：输入为 $A_1, A_2, \dots A_n$，输出为 F，称 F 为 $A_1, A_2, \dots A_n$ 的逻辑函数，记为 $F = f(A_1, A_2, \dots A_n)$
- 最小项和最大项　　最小项：（与）+（与）+（与）　↗ 包含所有
  最大项：（或）·（或）·（或）

① 最小项推导法：真值表 → 逻辑表达式（输出为1的）
最大项：把输出为0的输入组合写成十式，（0→原变量 1→反变量）

4. 化简（卡诺图）

- 化简方法：　目的：或项个数最少，或项中变量数最少

① 合并乘积项法：利用互补律消去一个变量

eg. $(ABC + A\bar{B}C) = AC(B+\bar{B}) = AC$

② 吸收项法：利用吸收律和包含律减少"与"项

eg. $(AB + \bar{A}\bar{B})CD = \overline{A\bar{B}+\bar{A}B}\cdot CD = CD$

③ 配项法：利用互补律

eg. $AB + \bar{A}\bar{B}C + BC = AB + \bar{A}\bar{B}C + ABC + \bar{A}BC$.

$= (AB + ABC) + (\bar{A}\bar{B}C + \bar{A}BC) = AB(1+C) + \bar{A}C(B+\bar{B})$.

$= AB + \bar{A}C$.

④ 对偶

No.

Date / /

三、数字逻辑：组合、时序

(一) 组合逻辑。 ①真值表 ②列表达示、化简 ③画逻辑电路图

1. 运算单元

1. 加法
- 1位加法
  - 半加器（向上进位，但不考虑下面进位）$A_0 B_0 \to S_0\ C_0$
  - 全加器（考虑低位进位）$C_i\ A_0 B_0 \to S_0\ C_0$
- 多位加法
  - 串行进位 → 进位延时长
  - 并行进位 → 给出每个进位的表达式 ← 复杂

2. 减法 $[A-B]_{补} = [A]_{补} + \overline{[B]_{补}} + 1$

3. 溢出 两种情况
- 符号相同两数加 → 符号变
- 符号不同两数减 → 符号与减数同

判断：两个符号位（00正数，11负数）⇒
- 01 正溢 ⇒ 00往加
- 10 负溢 ⇒ 11往减

4. 乘法 n位阵列乘法（n位数×n位数）
- n(n-1)个全加器
- 2(n-1)最长路径延时

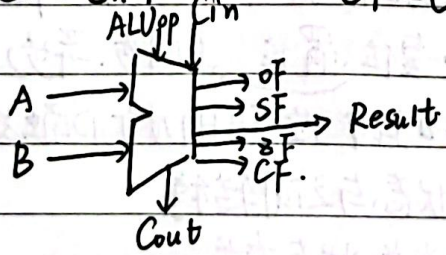
5. 比较器 列真值表
- 4位比较器（7485芯片） ⇒ 多片扩展位数

6. ALU
- 十、逻辑单元
- 标志位：S_f（符号位）、Z_{ero}（零标志）、C_f（进位进，无符号是否溢出）、O_f（溢出标，有符号数是否溢出）
- 运算结果

$OF = C_n \oplus C_{n-1}$　　$CF = C_{out} \oplus C_{in}$　（C都是进位）

$SF = F_{n-1}$

$ZF \Leftrightarrow F=0$



2. 编码、译码

1. 编码器　n位输入信号（只有1位有效）→ m位编码　（真值表）

$n \le 2^m$

↳ 高电平/低电平有效

8421BCD（每一位分别8、4、2、1）、余3编码（+3）→ 译码-3　(12421…)

2. 优先编码器：可以多位有效 → 但编码器只看优先级最高的

No.

Date / /

3、译码器　n线－2^n线　只有一个输出有效（最小项对应）

完全译码器（4→10 多余的不输出）全高电平.

4、显示译码器　真值表

(3) 多路选择器 MUX　$D_0 \sim D_{n-1}$ 输入端　$A_{m-1} \dots A_0$ 选择控制端　Y输出　$(n=2^m)$

$Y = \sum$（D_i对应最小项）　eg. $D_0 \Leftrightarrow \bar{A}_1\bar{A}_1\bar{A}_0 D_0$

特殊：多功能运算：　$D_0 \sim D_{n-1}$ 控制输入　$A_{m-1} \dots A_0$ 输入变量

(2^{2^m}) 2^n种功能：可实现 $A_{m-1} \sim A_0$ 全部组合逻辑

eg. $F(A,B,C) = \bar{A}BC + A\bar{B}\bar{C} + AB = \bar{A}BC + A\bar{B}\bar{C} + ABC + AB\bar{C}$

⇒ D_1、D_2、D_6、D_7 为1，其余为0

(4) 竞争、冒险

逻辑函数可简化为：① 0冒险：$F = A + \bar{A}$（eg. $F = AB + \bar{A}C$）

别的量一定取值下　② 1冒险：$F = A \cdot \bar{A}$

(二) 时序逻辑

(1) 触发器：　当前输出由当前输入与原来状态决定.

无外加信号→保持原态.　有外加信号→变态.

① ~~D触发器~~ 锁存器：电平触发

② D触发器：边沿触发：可以用2个D锁存器做.

·复位、置位（同步、异步）.

③ 寄存器：同一时钟控制的N个D触发器

(2) 有限状态机　有限个状态与之间转移

① Moore型：输出只与当前状态有关

② Mealy型：——、当前状态＋输入信号（输入接组合逻辑上）

(3) 时序关系　$T_C \geq T_{ctq} + T_{cd} + T_{setup}$

别太慢来不及 ⇒　时钟周期　输出稳定时间　组合逻辑最大延迟　建立时间（输入在前需保持）

$T_{ccq} + T_{cd(min)} \geq T_{hold}$

~~反之则~~ ⇒ 输出最小延迟　组合逻辑最小延迟　保持时间（输入在后保持时间）

别太快把上次覆盖

No.

Date / /

四、指令系统与MIPS.

(一)指令格式

1. 概述. (1). 小端: (低). 大端: 最高位存在地址低位

2. 指令格式. 地址数目(0~3). 操作码长度可变 指令长度可变

3. 寻址方式 (1) 种类: 立即.(数在指令里), ~~直接...~~ 寄存器直接.(JC存数)
寄存器间接...(JCQ存数地址) 、相对寻址(相对PC地址)
堆栈...、 基址/变址 (基址(array首地址)+[变址](寄存器)+常数偏移量).

(2) 寻式地址(指令中的) → 有效地址(真实)

(二)指令系统

1. 架构. CISC与RISC: (x86) 指令长、寻址方式多. 功能多 代码少、慢
↳ MIPS 定长指令、寻址少、代码多、效率高、CPI小

2. MIPS架构. (1). 32个寄存器(5位) t0~t7 (8-15) s0~s7 (16-23)
a0打印(立即数) (跳转) V0(syscall) 2 sp栈(29) ra返回跳转(31)

(2) 种类: R、I、J 最多3地址, LW、SW基址(单一)

Op Rs Rt Rd Shamt Func

(3) R型. Op(6) Rs(5) Rt(5) Rd(5) Shamt(5) Func(6)
↓
全为0

①算术、逻辑: $rd = $rs ? $rt. add, addu, sub, subu.
add $rd, $rs, $rt. and、or、nor、xor.

② 移位(不用$rs) $rd = $rt ? shamt. sllv sll、srl、sra.
~~sll $t1, $t2, 10~~ sll $rd, $rt, shamt.

③比较 if($rs < $rt) $rd = 1 (0). slt、sltu
slt $rd, $rs, $rt.

④无条件跳转. jr $rs; 跳到$rs地址
jalr $rd, $rs; $rd为PC+4, 跳到$rs.

(4) I型. Op(6) Rs(5) Rt(5) imme(16).

①逻运算. addi $rt, $rs, imm : $rt = $rs + imm

②load/store: lw $rt, offset($rs) rt ← mem[rs+offset]
(sw).

lb、sb(存取字节).

No.

Date / /

③ beq $rs, $rt, label 相等跳转label (bne).

(5) J型. J target. (jal 跳转时存PC到$ra(31))

op(6) index(26)

(6) syscall ⇒ $v0 = 1 打印int 5 读int 10 exit.

↘ $a0 ↙

(7) 其他 li $t1, imm (赋值)

la $t1, lable (t1赋lable首地址)

No.

Date / /

五、MIPS处理器CPU.

(一)设计概述.

1. 功能　取指、译码、运算、存数.

2. 组成. 执行：ALU、寄存器、数据存储器、MUX、移位器、比较、Extend、Sign

控制：PC、指令寄存器、译码器、时钟.

(二)MIPS模型机.

1. 指令集. (1) 32个32位寄存器、PC、HI LO

(2) ALU、Mux、Signext、Registers、DM、IM、Clock

(三)单周期

1. 哈佛体系结构　IM与DM单独.

指令(PC+4) → Registers → ALU → Mem. → 写Reg.

2. 控制信号：ALUop(ALU控制)、RegDst(Reg写入Rt/Rd)、RegWrite、ALUSrc(ALU第二口用rt/imm)、PCSrc(PC+4/beq)、MemRead、MemWrite、MemtoReg(写回用ALU/DM).

3. 指令周期计算：① 按最长指令决定

② (可变周期) → 按不同指令的比例

No.

Date / /

五、二、流水线CPU

(一)、多周期MIPS通路

1. 普林斯顿结构：指令、数据使用一个存储器，
1个ALU，每个部分中间用寄存器记录指令数据

2. 时钟周期：5×最长步骤　流水线段数 = n+m-1

3. 五个阶段　取指(IF)(PC+4)、译码读Reg(ID)、ALU执行(EX)、访存(Mem)、回写Reg(WB)

4. 流水线寄存器：数据、控制信号。

5. Register File. 上升沿更新时L5写回，L2读L5新值(同周期下)
∴先写后读

6. 性能　·加速比　T流水线 ≥ T单周期 / 指令数　(各阶段相等取等)

·各阶段花费相同→流水线更优

·一件指令周期变大→但指令多了接近一个阶段时间

(二)流水线冒险.

1. 结构冒险　硬件资源冲突

① 单一内存(Im、Dm用一个内存)　多加数据口

② 寄存器同时读写：前半周期写，后半读

2. 数据冒险　※RAW(先后读)、WAR、WAR、RAR

① 旁路转发　都转发回EX(ALU)　MUX{Reg读到值 / 上一条EX值 / 上上条WB值}

~~IF/EX~~ EX/MEM→EX、MEM/WB→EX

先看↗　后匹配↗　看Reg编号一不一样.

·不能做到：lw下一条读

② 阻塞与转发　(1)编译时往指令中加气泡指令(nop)

~~(2)插nop指令(lw后)~~

(2)阻塞：插bubble(硬件逻辑).

当lw进入EX阶段：分析与下条是否冲突

↳(前面为lw且lw的rt与这个rs/rt相同)

No.

Date / /

冲突 ⇒ (1) 冻结 IF/ID　下一条不往前传 → (Control: IF/ID使能不变)
(2) 清除 ID/EX　(插气泡) 指令全设为0 (ID/EX清除)
(3) 冻结 PC 值　PC不变 (新control: PCWrite=0).

• 若DM直接转发至ALU：(两段时间叠加) 周期可能大一倍

3. 控制/分支冒险　beq、bne.　三种办法：① 直接全阻塞 ② 赌博不跳转
若不跳转 → 正常处理
若跳转 → 清除后面3条已执行指令 (因为没有写直接寄存器清0)
③ 可以缩短延迟，在ID里提前比较 → 只用清除1条 (IF/ID)
↑但引起数据冒险 (计算型：nop一条，lw：nop两条)
阻塞+转发　↑因为之前在EX，这次更提前了.

• 优化：所有分支前放一个延迟槽 (nop) → 最多再加一个nop.

(三) 计算机性能评价.

1. 响应时间：完成一个任务所花时间总和
吞吐量：一定时间间隔内完成的任务数.

2. CPU 执行时间 (一个程序在CPU上全部时间)
= CPU 时钟周期数 × 时钟周期长度

CPU 时钟周期数 = 程序指令数 × 平均每条指令时钟周期数.

CPI：平均每条指令时钟周期数.

MIPS：百万指令每秒 = $\frac{主}{CPI \times 10^6}$

六、主存储器

(一) 概述

1. 分类. 介质、访问方式 (RAm. Rom)、功能.
   (random 随机 / read)

2. 性能指标 访问时间 T_A，存储周期 ($T_C > T_A$).

   带宽 = $f(\frac{1}{T_C})$ · 字宽 bps.

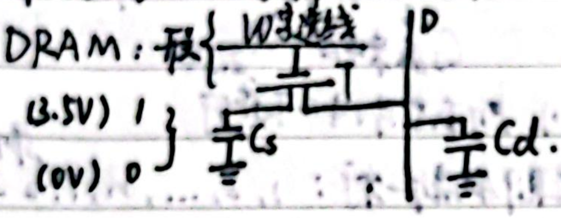
3. 半导体存储方式：
   - RAM
     - SRAM 静态 使用晶体管，不用刷新
     - DRAM 动态 小，用电容，但会漏电.

   DRAM常见：SDRAM. DDR1(双倍). DDR2(双倍×2). DDR3~5.

   - ROM (Read Only 只读)

(二) 存储单元电路 两种状态 0, 1.

DRAM：一般



电容 Cd >> Cs.

写入：D接3.5V→1 / 0V→0
W接高电平→开关开

读入：D充到2.5V
看电流往哪边流 (外→3.5V)

(三) 存储芯片结构

1 Byte (B) = 8 bits (b)   K 2^{10}、M 2^{20}、G 2^{30}

1. 描述：字单元数(字个数) × 每个字位数   eg. 1K×2、64K×8

   $2^n \times m$

   位数 $2^n \times m$ bits 地址线：n根 数据线(字长)：m根
   ↳ 译码 ($n \to 2^n$)

2. 一维地址 ✗

   二维地址. eg. 4096×4 ⇒ $2^7 \times 2^7$

   ∴ 行地址 7位 列地址 5位 ($\frac{2^7}{4}$)

   SRAM： X、Y 分开~~译码~~地址线传输.

   DRAM： X、Y 分别传输，管脚复用 (译码器还是俩)
   ↓
   尽量管脚数一样 [eg. $2^6 \times (2^6 \times 4)$].

H 16进制标志

(四) 存储器扩展

1. 位扩展　1K×4 → 1K×8

两个1K×4，地址线直接连，数据各有一半(线分一半连)

2. 字扩展　1K×8 → 4K×8

地址线、数据线直连 ⇒ 多出来两根地址 → 译码片选信号

3. 混合扩展．分几组(字扩展)，片选选组　4K×4 → 16K×8

每组内部分别连数据线．　4组，每组2个

(五) DRAM刷新

1. $\Delta V = V_d' - V_{pre.} = (V_{cs} - V_{pre}) \times C_s / (C_s + C_d.)$

(ΔV ↓) 需放大　(V_{cs} ↓) C_s原来电压　(V_{pre} →) 2.5V

2. 读操作刷新，按行刷新，所有芯片同时进行

3. 分散刷新　每周期刷新一行：1, 2, …, n, 1, 2 …　刷新T = 行数 × 存储周期

集中：一部分周期工作，另部分刷新全部　T一般2ms

异步：工作几个周期刷新一行　T一般2ms

No.
Date / /

七、Cache高速缓存器

(一) Cache局部性原理

1. 空间局部性(临近单元)、时间局部性(不久再次访问)
   ⇒ CPU与主存之间增加Cache. (寄存器 - Cache - DRAM - 硬盘).
2. Cache结构: Data Block(若干字,内容)、地址标记Tag、有效位V.
   术语: 数据块, 行(block + Tag + v). 组(若干块)、路(组内索引).
   ↳行数
   命中hit率. 缺失miss率
3. 总容量: 组数 × 每组行数 × (每块字节数 × 8 + tag + 1) bits.
   大小(示意)

(二) Cache映射机制.

主存地址 —映射→ Cache.位置.

1. 全相联映射. (多对多). Tag + offset. 主存任一可映射任一位置 (全路组相联)
   ↳查找Tag
   命中率高, 查找复杂
2. 直接映射. 主存每一块有对应Cache块. K = J mod C. (1-路组相联)(行)
   Tag + Index + Offset. Index大小 → Cache数量.
   Cache块
   确定index后,只需与一个tag比较. , 空间利用不充分
3. 组相联. Tag Set offset 主存、Cache都分为K组, 内部用全相联
   组号

正常: Tag Set组号 offset (组相联).
↓每个组只有一行, 有#Cache个组
Tag Index(组号) offset (直接相连)
↓只有一个组, 包含所有块, (没必要有组号)
Tag offset (全相连).

(三) Cache替换策略

1. 访问缺失时, 会把一块都从主存传到Cache.
2. 替换策略: 最低频率LFU: 替换使用次数最少的
   LRU 最近最少、 先进先出 FIFO

No.

Date / /

(四) Cache性能分析

1. Cache实际容量：数据块 + tag + valid bit （用位）
2. 系统访问时间：$T = \text{Cache访问}T_c \times H + (T_c + \text{主存访问}T) \times (1-H)$
   （平均存取时间）　　命中率

八 虚拟存储器.

(一) 辅助存储器

半导体、磁介质、光介质

1. 磁介质： 写入：磁化， 读出：电磁感应

2. 硬磁盘参数： 磁头数 × 磁道数 × 每道扇区数 × 每扇字节数

访问时间 = 寻道时间 T_s + 寻区时间 T_s (除以2：平均)

3. 磁盘类型： RAID 独立冗余磁盘阵列 (多个物理磁盘 → 看成一个)

- RAID0 无备份，性能高
- RAID1 每个盘一个备份
- RAID2 海明校验
- RAID3 奇偶校验(异或) 一块坏可修

(二) 虚拟存储器：操作系统上存储位置 → 把主存当作辅助存储器的高速缓存技术

1. 两种管理方式：可变分区模式(按程度大小分区→最后变零碎)、分页

2. 页表式：内存和进程都分为固定的长度的块(页) → 内存不用连续

(虚拟)逻辑地址 ⟶ 页表 ⟶ 物理地址 ⇒ 具体页

程序：虚页 映射 主存：实页

(1) 页表：建立在内存，每个程序一个，用虚页号索引，得到实页号和有效位(1脏位是否修改)

页表基址寄存器：保存页表在内存中首地址 ⇒ 加虚页号 ⇒ 对应页表位置

虚地址：虚页号 + 页内地址

实地址：实页号 + 页内地址.

(2) 二级页表：节省内存防止页表项太多 一级：页表号 ⇒ 页表地址

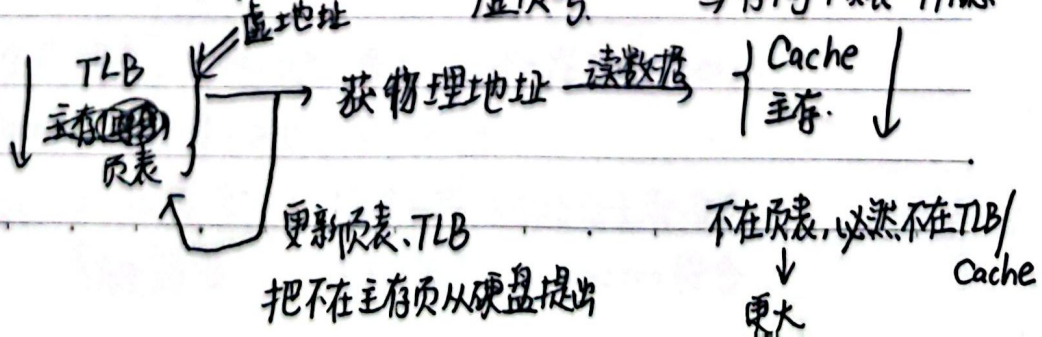
页表号 + 页表偏移 + 页偏移 二级：页表偏移 ⇒ 物理页号

(3) TLB小页表：用Cache存部分活跃页表项(全相联/组相联)

↳ 组相联地址：Tag组内地址 Set组号 Offset

虚页号 ⇒ 存Tag + 页表一样信息

(4) 逻辑：



No.

Date / /

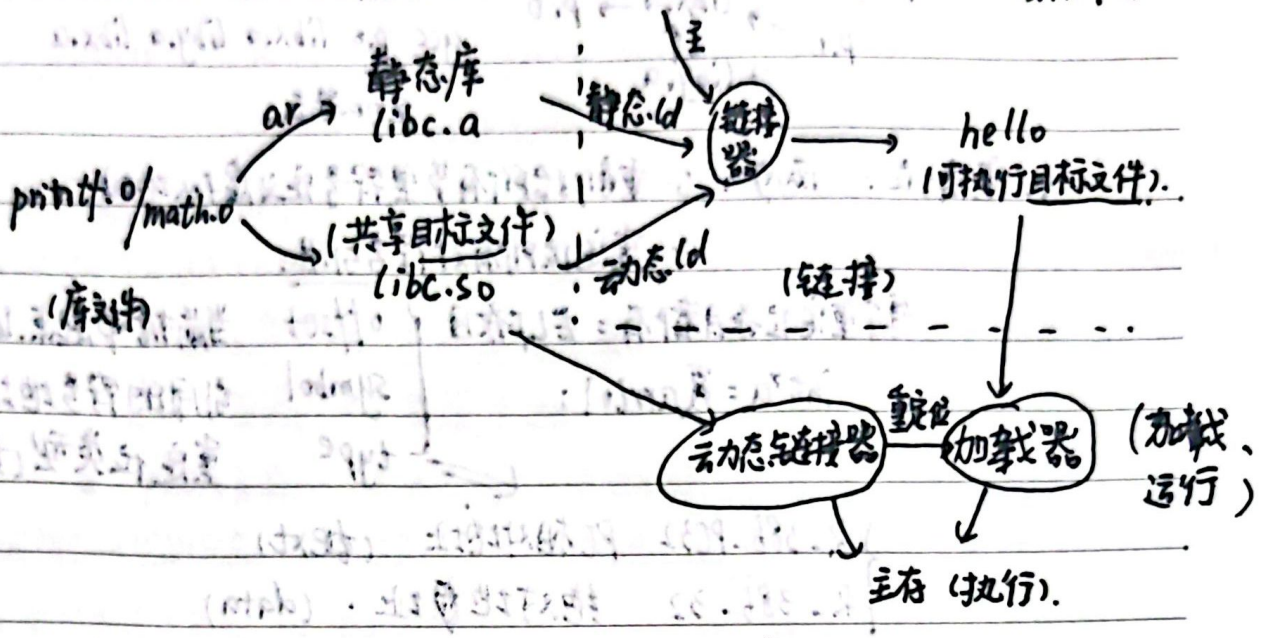
九、链接与运行.

~~1. 静态链接~~ 各种代码、数据组合单一文件. 静态(编译时)

(可重定位目标文件).

hello.c 预处理 hello.i 编译 hello.s 汇编 hello.o (翻译).

源文件



(一) 目标文件: 三种: 可重定位、可执行、共享(动态加载)

- 格式: ELF 格式 (可执行可链接)

ELF Header	运行基本信息
.text	已编译代码
.rodata	只读数据 (常量, eg "hello", 部分 const)
.data	已初始化全局和静态变量 (局部放栈)
.bss	未初始化或为0的全局/静态变量
.symtab	符号表
.rel.text	.text节重定位信息
.rel.data	.data节重定位信息

(二) 符号解析

1. 类型: 全局符号、外部(全局)符号、局部符号(static).

extern: 外部符号定义, 用于连接

2. 符号解析: 强符号: 函数或初始化全局变量

弱符号: 未初始化 / extern符号

No.

Date / /

最多一个强符号，弱符号 解析→ 强符号（无强则指定随机弱）

3. 静态库 .a → 打包多个 .o，找对应的符号的.o加载复制。

↳解析引用。

p.o → libx.a → p.o
p.o → liby.a
libx.a ↔ liby.a

gcc p.o libx.a liby.a libx.a

全加进去。

4. 重定位：两步：① 重新给所有节里符号定义虚拟地址

② 重定位所有对符号引用。

每个重定位条目都有一ELF表目

int *a = &arr[0];

- offset 当前节起点偏移
- symbol 引用的符号地址索引
- type 重定位类型（最多4种类型）

- R_386_PC32 PC相对地址 (text)
- R_386_32 绝对地址 (data)

无法：PC：由当前地址 = section地址 + 偏移量offset

(*refptr)指向地址 = symbol的地址 + *refptr - 当前地址

非PC： *refptr = symbol地址 + *refptr

(三) 运行。映射到内存的分区中。

动态链接库（.so）可被不同文件共享，运行时加载

↳ 动态链接器。

No.

Date / /

十、总线I/O → 只允许单一占用

(一)总线仲裁. 1. 链式(查询)
总线同意信号BG一级级向下传,有前后优先级.

2. 计数器 每个接口一个编号 → 定义优先级.

3. 独立请求 独立请求信号和同意信号.

(二)IO接口: 串行、并行, 同步、异步.
编址:独立编址(x86)、统一编址(MIPS).

(三)程序查询IO方式: 查询IO状态 → 状态寄存器

(四)中断与中断IO方式: (比如按键建盘) → CPU每周期进行中断检查
1. • 主程序中断,去处理紧急事务(中断服务子程序) → 错误、IO响应

分类:可屏蔽、不可屏蔽中断.

2. 中断请求: 触发器、中断标记寄存器(CPU查询)

CPU结束当前指令 → 发出响应 → PC入栈 → 获取中断向量 → (中断程序) 保存其余状态 → 处理 → 恢复状态,PC返回

(五)DMA I/O方式 → DMA控制器帮助CPU,避免中断
外设与内存直接交换

1. 步骤: ① CPU → DMA: 读/写,地址,数量

② DMA数据传输

③ DMA → CPU 发送中断告知完成

2. 传送方式: ① 停止CPU占用,成组传送 ⇒ DMA完全占总线 → 占多个周期

② 周期窃取:等CPU释放总线 → DMA窃取一个周期(多次)

3. 结构: 地址寄存、长度寄存、计数器、控制逻辑、数据缓冲寄存

4. 类型: 选择型:连多个设备,但只可同时一个传输(快)

多路型: 多个通道(慢).