操作系统

Cynthia

2024-11-27

Table of Contents

Chapter 1: 第一章 计算机系统概述

1.1. 计算机的分类

$$\text{电子计算机}\begin{cases}\text{电子模拟计算机（数值连续）}\\ \text{电子模拟计算机（按位运算）}\end{cases} \tag{1.1}$$

1.2. 计算机发展简史

1.2.1. 计算机性能指标

1. CPU 执行时间: CPU 执行时间 = CPU 时钟周期数 (N_C) × CPU 时钟周期 (T)
2. CPI: 表示每条指令的周期数
   CPI = 执行某程序所需 CPU 时钟周期数 / 程序包含的指令条数
   $\text{CPI}=\frac{N_c}{I_N}=\sum_{i=1}^{n}\text{CPI}_i\times\frac{I_i}{I_N}$
   CPI_i 表示 i 指令所需的平均时间周期数，I_i 表示 i 指令在程序中执行的次数
3. MIPS: 平均每秒执行的百万条定点指令数
   MIPS= 指令数 / （程序执行时间 $(t_{\text{CPU}})\times 10^6$ ）

1.3. 计算机的硬件

$$\text{计算机硬件:}\begin{cases}\text{运算器}\\ \text{存储器}\\ \text{控制器}\\ \text{适配器}\\ \text{输入/输出设备}\end{cases} \tag{1.2}$$

1.3.1. 控制器

1. 指令: 操作码 + 地址码
2. 控制器的基本任务: 按照计算程序所排的指令序列，先从存储器取出一条指令放到控制器，译码器对指令操作码进行判别，根据指令性质，执行指令。
3. 取指周期: 取指令的时间
4. 执行周期: 执行指令的时间

运算器和控制器通常被组合于一个集成电路芯片中，合称中央处理器(CPU)。如今 CPU 中加入了存储器

1.4. 计算机的软件

$$\text{计算机软件:}\begin{cases}\text{系统程序}\\ \text{应用程序}\end{cases} \tag{1.3}$$

1. 编译程序: 将源程序编译为机器语言程序
2. 编译器: 编译程序+运行系统

3. 链接器: 将编译程序生成的目标程序与库程序链接成为可执行程序
4. 加载器: 将可执行程序装入内存并执行
5. 操作系统: 管理计算机硬件和软件资源的计算机程序
6. 数据库: 实现有组织地、动态地存储大量数据，方便多用户访问的计算机软、硬件组成的系统

$$\text{高级语言} \xrightarrow{\text{编译器}} \text{汇编语言} \xrightarrow{\text{汇编器}} \text{机器语言} \xrightarrow{\text{链接器}} \text{可执行代码} \tag{1.4}$$

Chapter 2: 运算方法与运算器

2.1. 不同数制间的转换

2.1.1. 小数部分精度要求

当小数部分不能整除为二进制时，则乘以 2 取整的过程中，积不会为 0；或者当小数部分转化为二进制位数很长，这时由精度来决定二进制位数。

$(114.35)_{10} = (?)_2$，要求精度大于10%

要求“ = ”左右两边的十进制值的差的绝对值 $< 10\%$；因为 $10\% > 2^{-4}$，所以只需取 4 位二进制小数即可满足要求。$(114.35)_{10} = (1110010.0101)_2$

2.2. 数据与文字的表示方法

2.2.1. 机器数

1. 特点:
   - 表示的数值范围受计算机字长的限制;
   - 机器数的符号位必须被数值化为二进制 0 和 1;
   - 机器数的小数点是用隐含规定的方式来表达的。
2. 真值：机器数所真正表示的数值，一般使用数值（二进制或十进制）前冠以“+”、“ - ”符号这种方法来书写。

2.2.2. 数的机器码表示

1. 原码
   - 符号位：正数为“0”，负数为“1”
   - 其余数值位不变
2. 反码
   - 正数：与原码一致
   - 负数：除符号位，其余取反
3. 补码
   - 正数：与原码相同
   - 负数：$[X]_{\text{补}} = 2^{n+1} + X$

     除符号位，其余按照反码+1（or 在原码的基础上，从后往前，保留第一个 1，其余取反）
4. 移码
   - 正数：同补码相比，符号位取反
   - 负数：符号位取反后+1（对真值的数值位从后往前第一个 1 不变，其余取反，包括符号位）

5. 逻辑移位 vs 算术移位
   - 逻辑移位：左移或右移补零
   - 算术移位：左移补零，右移时符号位补齐

2.2.3. 浮点数的机器表示

浮点机器数用于表示实数，其小数点的位置由其中的阶码规定，因此是浮动的

浮点数 N 的构成：$N = M \times R^E$

M_s（数符）	E_s（阶符）	$E_1E_2...E_m$ 阶码数值	$M_1M_2...M_n$（尾数数值）

1. 尾数 M
   - 为定点小数。尾数的符号代表了浮点数的正负，称为“数符”。
   - 尾数一般用原码和补码表示
2. 阶码 E
   - 为定点正数。数值大小表示浮点数实际小数点与尾数小数点之间的偏移量
   - 阶码一般用移码和补码表示
3. 阶码的底 R
   隐含规定，一般为 2、8 或 16

IEEE754 国际标准，隐含阶码底数为 **2**，常用浮点数格式有 3:

1. 短实数（单精度浮点数）
2. 长实数（双精度浮点数）
3. 临时实数
   主要用于进行浮点数运算时保存临时的计算结果

M_s	E	M
数符	阶码	尾数
0:正数 1:负数	2^n-1的移码	原码表示， 单双精度的整数位“1”隐藏，临时实数无隐藏位

类型	总位数	尾数（含 1 位数符）	阶码位数	真值计算
短实数	32	24	8	$N=(-1)^{M_s}\times(1.M_1M_2...M_n)\times 2^{E-127}$
长实数	64	53	11	$N=(-1)^{M_s}\times(1.M_1M_2...M_n)\times 2^{E-1023}$
临时实数	80	65	15	

对 **IEEE754(32 位)**标准转十进制，解题步骤:

1. 转为二进制
2. 得到 S, E, M，M 前面隐含了 1

3. 得到 $e = E - 127$
4. 求算 $X = (-1)^S \times 1.M \times 2^e$，转十进制

e.g. X 的 754 标准$(41360000)_{16} = (?)_{10}$

$$(41360000)_{16} = \left(\overbrace{0}^{\text{S}}\overbrace{10000010}^{\text{E}}\overbrace{01101100000000000000}^{\text{M}}\right)_2$$

$\therefore e = E - 127 = 130 - 127 = (3)_{10}$

$1.M = 1.011011$

$\therefore X = (-1)^S \times 1.M \times 2^e = (1.011011) \times 2^3 = 1011.011 = (11.375)_{10}$

将十进制转为 IEEE754 标准的 32 位浮点数的二进制存储格式，解题步骤:

1. 转为二进制
2. 得到 $1.M, e$
3. 得到 $E = e + 127$
4. 得到 X 的二进制存储格式

e.g. 将 $(20.59375)_{10}$ 转换成 754 标准的 32 位浮点数的二进制存储格式

$(20.59375)_{10} = (10100.10011)_2 = 1.010010011 \times 2^4$

$\therefore e = 4, M = +010010011$

$\therefore E = e + 127 = (131)_{10} = (10000011)_2$

$\therefore X = \overbrace{01000001101001001100000000000}^{\text{E}} = (41A4C00)_{16}$

2.2.4. 浮点机器数的规格化表示

对尾数规格化:

1. 原码表示: 尾数最高有效位(M_1)为 1
2. 补码表示: 尾数的数符和最高有效位相异，即$M_S \oplus M_1 = 1$

对于非规格化浮点数，可以通过修改阶码和左右移尾数来规格化

1. 左规 尾数左移，阶码减一
2. 右规 尾数右移，阶码加一

2.2.5. 浮点数的表示范围

- 上溢: 大于最大正数 or 小于最小负数
- 下溢: 位于最大负数和最小正数之间，计算机直接视其为机器零

- 浮点数在机器上不均匀分布

2.2.6. 字符编码

ASCII 码 规定最高位为 0，剩余 7 位给出 128 个编码

2.2.7. 汉字的表示方法

1. 汉字的输入编码
   - 数字编码 常用国标区位码
   - 拼音码
   - 字形编码
2. 汉字内码 两个字节表示，最高位规定为 1
3. 汉字字模码 用点阵表示汉字字形

2.2.8. 校验码

1. 奇偶校验
   - 奇校验 将数字按位异或，当有奇数个 1 时，结果为 1，即加 0
   - 偶校验 将数字按位异或，当有偶数个 1 时，结果为 0，即加 0
2. 海明码
3. CRC 校验

2.3. 定点加法、减法运算

2.3.1. 补码加法

补码加法公式:

$$[x]_{补}+[y]_{补}=[x+y]_{补} \pmod{2^{n+1}} \tag{2.1}$$

e.g $x=+1011, y=-0101$，求$x+y$

$[x]_{补}=01011, \quad [y]_{补}=11011$

$[x+y]_{补}=[x]_{补}+[y]_{补}=100110$

$\therefore x+y=+0110$

2.3.2. 补码减法

补码减法公式:

$$\begin{gathered}[x-y]_{补}=[x]_{补}-[y]_{补}=[x]_{补}+[-y]_{补} \\ [-y]_{补}=-[y]_{补} \pmod{2^{n+1}} \\ [-y]_{补}=\rightharpoonup [y]_{补}+2^{-n}\end{gathered} \tag{2.2}$$

符号 $\rightharpoonup$ 表示对 $[y]_{补}$ 作包括符号位在内的求反操作，$+2^{-n}$ 表示在最末位加 1

e.g. 已知 $x_1 = -1110$，求$[x_1]_{补}$，$[-x_1]_{补}$

$[x_1]_{补} = 10010$

$[-x_1]_{补} =\rightharpoonup [x_1]_{补} + 2^{-4} = 01101 + 00001 = 01110$

2.3.3. 溢出概念和检测方法

溢出：运算过程中出现了大于字长绝对值的现象。定点整数机器中，数的表示范围为 $|x| < (2^n - 1)$

正溢：两个正数相加，结果大于机器字长所能表示的最大正数

负溢：两个负数相加，加过小于机器所能表示的最小负数

图 2.1　定点整数表示范围

检测溢出是否发生的两种方法:

1. 双符号位法:
   顾名思义，有两个符号位。采用变形补码，使得模 2^{n+1}的补码能够表示的数值大一倍，变为模 2^{n+2}
   - 两个符号位都看作数码来计算
   - 舍弃最高符号位上产生的进位

   采用变形补码后:
   - 正数符号位为“00”
   - 负数符号位为“11”

   最高符号位表示结果的正确符号位

   e.g $x = +1100$，　$y = +1000$，求 $x + y$
   $[x]_{补} = \mathbf{00}1100$，　$[y]_{补} = \mathbf{00}1000$
   $[x + y]_{补} = 001100 + 001000 = \mathbf{01}0100$
   $\therefore$ 正溢出，结果大于 $+2^n$

2. 单符号位法
   - 正溢：当最高有效位有进位而符号位无进位时
   - 负溢：当最高有效位无进位而符号位有进位时

2.3.4. 基本的二进制加法/减法器

2.4. 定点乘法运算

2.4.1. 原码一位乘法算法

假设 $[X]_{原}=X_SX_1...X_n$，$[Y]_{原}=Y_sY_1...Y_n$，$P=X\cdot Y$，P_s 为积的符号

1. 符号位单独处理 $P_s=X_s\oplus Y_s$
2. 绝对值进行数值运算 $|P|=|X|\cdot|Y|$
3. 部分积初始化为 0。$Y_i=1$，部分积加 $|X|$；$Y_i=0$，部分积加 0，累加结果右移一位，得到新的部分积
4. 总共累加右移 n 次

$[X]_{原}=0,1011$

$[Y]_{原}=1,1101$

则 $P_s=0\oplus 1=1$

计算 $|P|=|X|\cdot|Y|$

部分积	乘数	操作
$0,0000$ $+\quad 0,1011$		

2.5. 浮点加减运算

1. 0 操作数检查：以尽可能的简化操作。
2. 对阶：原则是小阶对向大阶
   - 求阶差$\Delta E=E_X-E_Y$，若 ΔE≠0，即 EX≠EY 时需要对阶。
   - 若$\Delta E>0$，则$E_X>E_Y$，MY 每右移一位，EY+1，直至 EY=EX 。
   - 若$\Delta E<0$，则$E_X<E_Y$，MX 每右移一位，EX+1，直至 EX=EY 。
3. 尾数相加减
4. 结果规格化：尾数运算的结果可能出现两种非规格化情况：
   - 尾数溢出：需要右规（1 次），即尾数右移 1 位，阶码 + 1
   - |尾数| $<2^{-1}$：有前导零。需要左规，即尾数左移 1 位，阶码 − 1，左规可能多次，直到尾数变为规格化形式。
5. 舍入：可采用截断法、0 舍 1 入法、末位恒置 1。
6. 溢出：判断阶码溢出与否

Chapter 3: 存储系统

3.1. 存储体系概述

3.2. 主存储器

主存储器主要由 DRAM 构成，SRAM 和 ROM 也充当其部分

动态 RAM（主存）与静态 RAM（缓存）的比较

	DRAM	SRAM
存储原理	电容	触发器
集成度	高	低
芯片引脚	少	多
功耗	小	大
价格	低	高
速度	慢	快
刷新	有	无

表 3.1　DRAM 与 SRAM 的比较

地址线与数据线

地址线总数决定了存储容量，数据线总数决定了存储器字长

- 每根地址线传输 0/1，可以选择两个地址块；n 根地址线可以选择 2^n 个地址块
- 每个地址块中存储了数据，存储器字长指明每个地址块中的数据长度。（若存储了 0010，则需要 4 根地址线进行输出）

> 一个 $16K \times 32$位 的存储器，其地址线和数据线总共多少根？
>
> $16K$ 为存储容量，说明 $2^n = 16K$，即 地址线总数 $n = \log_2(16K) = 14$
> 32 bit 为存储字长，说明数据线总数为 32

3.2.1. 存储器容量的扩展

存储器芯片存储量有限，为满足实际容量需求，对存储器进行扩展

1. 位扩展
   - 存储器字长较大，对所选芯片进行位扩展（增加其能提供的存储字长）
   - 对所有位扩展芯片使用同一片选信号
2. 字扩展
   - 地址线较多，所选芯片不足以提供相应数量的地址块，需要进行字扩展（增加其存储容量）
   - 对进行字扩展的芯片使用不同片选信号

3. 字、位扩展
   同时进行字、位扩展。当然，实际上我们优先位扩展

用 $1K \times 4$ bit 的存储芯片组成 $4K \times 8$ bit 的存储器

所需存储芯片数量为 $\frac{4K \times 8 \text{ bit}}{1K \times 4 \text{ bit}} = (4 \times 2) = 8$ 片
其中，需要进行字位扩展：每两片为一组，进行位扩展，共 4 组，进行字扩展

3.2.2. 主存储器和 CPU 的连接

连接步骤:

1. 写出系统程序区（放 ROM）和用户程序区（放 RAM）的二进制地址码
2. 确定芯片类型和数量
3. 分配地址线
4. 确定片选信号

某机字长 16 位，CPU 地址总线 18 位，数据总线 16 位，存储器按字编址，CPU 的 控制信号线有：$\overline{\text{MREQ}}$（存储器访问请求，低电平有效），$R\overline{W}$（读写控制，低电平为写信号，高电平为读信号）。试问:
（1）该机可以配备的最大主存容量为?

（2）该机主存采用 $64K \times 1$ bit 的 DRAM 芯片（内部为 4 个 128×128 阵列）构成最大主 存空间，则共需多少个芯片?
若采用异步刷新方式，单元刷新间隔为 2ms，则刷新信号 的周期为?

（3）若为该机配备 $2K \times 16$ 位的 Cache，每块 8 字节，采用 2 路组相联映象，试写出对 主存地址各个字段的划分（标出各个字段的位数）；若主存地址为 462EH，则该地址可映象 到 Cache 的哪一组?

（4）已知该机已有 $8K \times 16$ 位的 ROM 存储器，地址处于主存的最高端；现在再用若干 个 $16K \times 8$ 位的 SRAM 芯片形成 $128K \times 16$ 位的 RAM 存储区域，起始地址为 $00000H$，假设 SRAM 芯片有 $\overline{\text{CS}}$（片选，低电平有效）和 $\overline{\text{WE}}$（写使能，低电平有效）信号控制端；试写出 RAM、ROM 的地址范围，并画出 SRAM、ROM 与 CPU 的连接图，请标明 SRAM 芯片个数、译码器的输入输出线、地址线、数据线、控制线及其连接。

---

(1) $S = 2^{18} \times 16 \text{ bit} = 512KB$

(2) 易知 DRAM 行数为 128
需要芯片 $\therefore n = \frac{2^{18} \times 16 \text{ bit}}{64K \times 1 \text{ bit}} = 4 \times 16 = 64$ 片
刷新信号周期为 $T = \frac{2ms}{128} = 15.625\mu s$

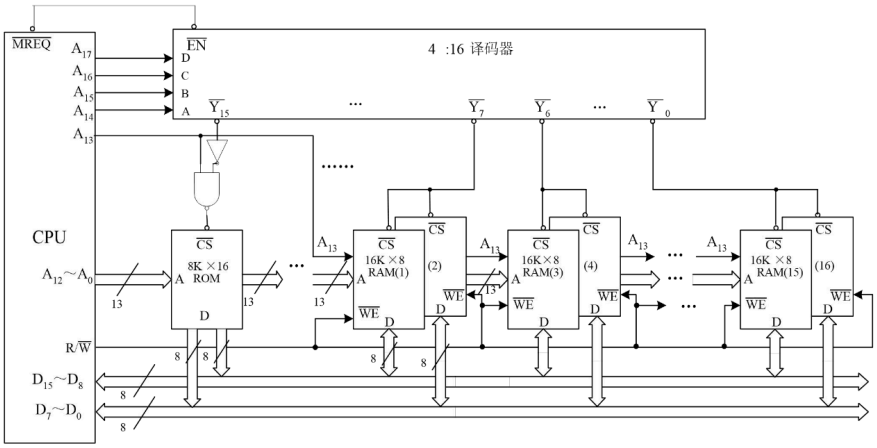
(3)

(4)

- ROM 放在最高端，前面都是 1； RAM 从 $\mathrm{hex}(00000) = 0$ 开始放置
- 需要 RAM $\frac{128K\times16\text{ bit}}{16K\times8\text{ bit}} = 8\times2 = 16$ 片，需要进行字位扩展，每组两片进行位扩展，共 8 组。ROM 和 RAM 共 9 组，需 4 位片选信号
- 地址线分配

A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$128K$ RAM
								…	…									
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
								…	…									
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	$8K$ ROM
								…	…									
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
D	C	B	A															

  - 画图



3.3. SRAM

3.4. DRAM

3.4.1. DRAM 刷新方式

刷新周期：从上一次刷新结束到下一次对整个 DRAM 全部刷新一遍为止，这一段时间间隔称为刷新周期。

刷新操作：即是按行来执行内部的读操作。由刷新计数器产生行地址，选择当前要刷新的行，读即刷新，刷新一行所需时间即是一个存储周期。

刷新行数：单个芯片的单个矩阵的行数。

- 对于内部包含多个存储矩阵的芯片，各个矩阵的同一行是被同时刷新的。

- 对于多个芯片连接构成的DRAM, DRAM控制器将选中所有芯片的同一行来进行逐行刷新。

  单元刷新间隔时间: DRAM 允许的最大信息保持时间，一般为 2ms。

  刷新方式: 集中式刷新、分散式刷新和异步式刷新。

1. 集中刷新
   先集中地访存，访完后集中刷新。刷新期间无法访存，“死区”。
2. 分散刷新
   访一会刷一会，对一行的操作分两步，宏观来看无“死区”。
   - 在任何一个存储周期内，分为访存和刷新两个子周期。
   - 存储周期为存储器存储周期的 2 倍（因为原先是 $T_{存储}=T_{\text{read}}$，现在 $T_{存储}=T_{\text{read}}+t_{\text{refresh}}$）
   - 刷新周期缩短: $T_{\text{refresh}}=T_{存储}*n(行数)$，$2ms$刷新了 $\frac{2ms}{T_{\text{refresh}}}$ 次
3. 异步刷新
   - 异步刷新采取折中的办法，在 $2ms$ 内分散地把各行刷新一遍。
   - 避免了分散式刷新中不必要的多次刷新，提高了整机速度；同时又解决了集中式刷新中“死区”时间过长的问题。
   - 刷新信号的周期为 $\frac{2ms}{n(行数)}=15.625\mu s$。让刷新电路每隔$15\mu s$（刷新信号周期）产生一个刷新信号，刷新一行

3.5. ROM

3.6. 高速存储器

3.7. 高速缓冲存储器 Cache

3.7.1. Cache 命中率

在一个程序执行期间，设 N_C 表示 Cache 完成存取的总次数，N_M 表示主存完成存取的总次数，则命中率 h:

$$h=\frac{N_C}{N_C+N_M}\times 100\% \tag{3.1}$$

若 t_c 表示 Cache 的访问时间（存储周期），t_m 表示主存的访问时间，则 Cache/主存系统的平均访问时间 t_a 为:

$$t_a=h\times t_c+(1-h)\times t_m \tag{3.2}$$

Cache/主存系统的访问效率 e:

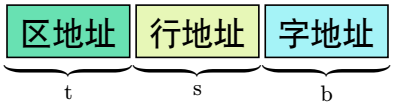
$$e=\frac{t_c}{t_a} \tag{3.3}$$

3.7.2. Cache 和主存 RAM 的映射关系

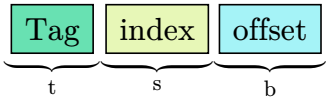
Cache 的行和主存的块长度一致

映射关系

1. 直接相联
   主存中块的块只能映射到 Cache 中固定的行 $2^n + r$ 行映射到 r 行（当 Cache 有 n 行）。即，Cache line index $= M_l \bmod C_l$
   则主存地址可分为三部分：区地址、块地址和字地址
   设 Cache 有 $n = 2^s$ 块，主存 DRAM 每 n 块划为一个区，共有 $m = 2^t$ 个区

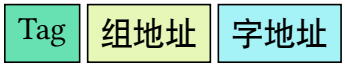
   $\Rightarrow$

   数据查找流程如下：
   a. 根据行地址查找 Cache 中的行
   b. 根据区地址（Tag）以及一些标志位判断其是否与主存中要查找的区对应
   c. if Tag 一致，输出该行内容，并根据字地址选择最终字的输出
   d. if Tag 不一致，从主存中载入相应地址内容到 Cache，再由 Cache 传给 CPU
2. 全相联
   主存中的任意一块（有连续的字）可以放在 Cache 中的任意行中，没有分区的概念

3. 组相联
   一个组分有几路，主存中的块需要顺序映射到组，但是路随机选择。

3.8. 虚拟存储器

主存和辅存依靠辅助软硬件支持构成虚拟存储器，扩展了可用空间，让程序看到一个连续的、大得多的“假装存在的内存空间”，即使实际物理内存没那么大。

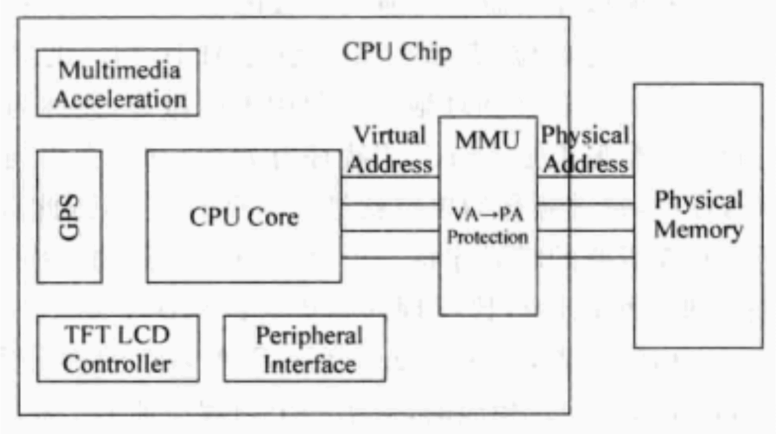


图 3.1　使用虚存的系统架构

MMU 为内存管理单元，用于将虚拟地址转为物理地址

程序进行虚地址到实地址转换的过程称为程序的再定位

术语	含义
逻辑地址	虚拟存储器提供的地址
物理地址	CPU 用于访问主存的地址

3.8.1. 虚拟存储器的管理方式

1. 段式管理

   程序按逻辑结构分段，段的大小不一，主存按段来分配存储管理方式。

   - 虚拟地址:

逻辑段号	段内地址

     段表首地址+逻辑段号（偏移量）得到段表项地址

   - 表项:

实存段首地址	装入位	段长

     若装入位=1，实地址 = 实存段首地址 + 段内地址

     若装入位=0，需从辅存将该段调入主存

   - 优缺点:
     - 优: 便于多道程序共享
     - 缺: 段间零碎存储空间不好利用

2. 页式管理

   辅存和主存空间分为等长页，主存按页来分配管理方式。

   - 虚拟地址:

逻辑页号	页内地址

页内地址根据页的大小计算。

操作系统维护页表，存储逻辑页号与对应主存块号。每个进程对应一个页表。

- 表项:

控制位	实页号

由于每个进程所需页数不固定，因此页表长度可变。$\Longrightarrow$ 通常将页表的基地址（指明页表在主存的位置）保存在寄存器中，而页表本身放在主存中。

- 优缺点
  - 优: 造页表方便、主存空间利用率高
  - 缺: 程序的处理、保护和共享不便

3. 段页式管理
   - 虚拟地址

虚段号	段内虚页号	页内地址

   - 地址变换:

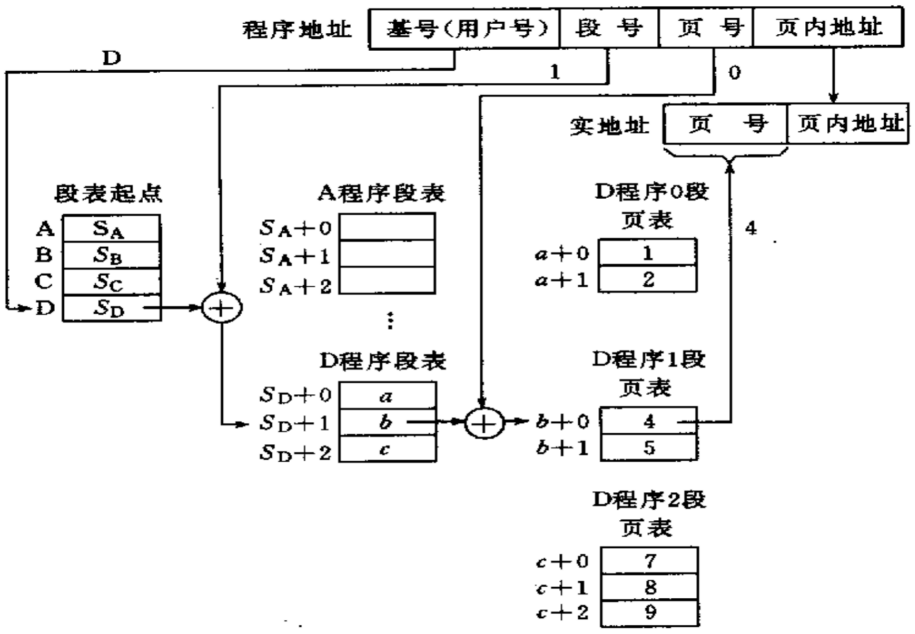


图 3.2 段页式虚拟存储器的地址变换

3.9. 外存储器