Computer Arithmetic and Organization:New Manual for review

author: Chotholly Boss

本文档对 PPT 例题、作业题、学习指导若干题、王道考研的若干题以及笔者个人掺杂私货的押题进行归纳整理,旨在帮助读者在短时间内覆盖足够的知识点以应对考试。

叠甲声明:若考试考到了本文档未覆盖的知识点,笔者概不负责。

Chapter 1: 计算机系统概述

Memorize

1. 计算机发展史:记忆主要器件
   - 第一代:电子管
   - 第二代:晶体管
   - 第三代:集成电路
   - 第四代及以上:超大规模集成电路

   第四代以后的存储器主要为半导体磁盘
2. 计算机体系结构的 7 个伟大思想:1+5+1
   - 1 利用抽象简化设计
   - 5 提高性能
     - 加速经常性事件
     - 并行
     - 流水线
     - 预测
     - 存储层次(层次化存储结构)
   - 1 通过冗余提高可靠性
3. 冯诺依曼架构计算机由 运算器、控制器、存储器、输入设备、输出设备 构成。基于存储程序原理,以运算器为中心,工作方式的基本特点为按地址访问指令,通常情况下顺序执行,特定情况下可以根据条件改变执行顺序。
4. 哈佛架构与冯诺依曼架构的区别是程序空间和数据空间是否为一体的

Example

1. 第三代计算机以 ____ 为主要器件。
2. 系统设计中可以通过 ____ 提高系统的可靠性。
3. 冯诺依曼架构计算机由 ____, ____, ____, ____, ____ 组成.
4. 哈佛架构和冯诺依曼架构的最主要区别是____。

Solution

1. 集成电路
2. 冗余
3. 运算器 控制器 存储器 输入设备 输出设备
4. 程序空间和数据空间是否为一体

Tip

有关性能指标的计算有如下参考步骤:

- 计算 CPI: 用各类指令的 CPI 乘上对应的比例后求和
- 计算 IPS:即 instruction per second 每秒指令数$f \times \frac{1}{CPI}$
  - 频率表示每秒有多少个周期
  - CPI 的倒数表示每周期有多少条指令
  - 相乘即得每秒指令数
- 计算 MIPS: $\text{IPS} \div 10^6$
- 计算程序执行时间: 指令数乘以每条指令所花费的时间
  - $n \times \frac{1}{\text{IPS}}$

重要的是明白缩写的含义,清楚含义的话计算花点时间也能自己发现怎么算。

- CPI:cycles per instruction
- MIPS: Million instructions per second
- IPS: instructions per second

一般来说,`ps` 就是 `per second` 的缩写,`b` 表示 `bit`,`B` 表示 `byte`,`I` 表示 `instruction`,`C` 表示 `cycle`

- bps: bits per second
- Bps: Bytes per second
- Kb/s: K bits per second
- KB/s: K Bytes per second

Example

(PPT 1) 假设一台计算机主频为 1GHz,在其上运行由2×10^5条指令构成的目标代码,代码中各类型指令所占的比例及 CPI 如下表所示

type	CPI	Percent
算术与逻辑	1	60%
Load/Store	2	18%
转移	4	12%
Cache Miss	8	10%

- 求程序的 CPI 和 MIPS
- 求程序的执行时间

Solution

- $\text{CPI} = 1 \times 0.6 + 2 \times 0.18 + 4 \times 0.12 + 8 \times 0.1 = 2.24$
- $\text{IPS} = 1G \times \frac{1}{2.24} = 4.464 \times 10^8$
- $\text{MIPS} = \text{IPS}/1\text{M} = 446.4$
- 执行时间 $= \text{instructions} \times \frac{1}{\text{IPS}} = \frac{2\times10^5}{4.464\times10^8} = 0.448 \times 10^{-3}s = 0.448\text{ms}$
  - 也可通过 CPI 和频率进行计算,即 $T = \text{instructions} \times \text{CPI} \times T, T \times f = 1$

Practice

1. 程序 P 在机器 M 上的执行时间是 20s,编译优化后,P 执行的指令数减少到原来的 70%,而 CPI 增加到原来的 1.2 倍,则 P 在 M 上的执行时间是(16.8s)。

Chapter 2: 计算机的运算方法

本章节大致上可分为数的表示与数的运算两个方面,前者一般通过选填题考察,后者一般通过大题考察。数的运算在选填题中基本不可能出现(无法体现运算过程,理论上直接十进制计算后仍是数的表示问题)。由此,复习时我们采取以下策略:

- 数的表示只需保证结果正确
- 数的计算确定一定的做题步骤,提高做题效率。

> 🔥 Tip
>
> 为快速完成数的表示,我们将数分为正数、负数、浮点数三类进行记忆。
>
> - 正数
>   - 正数的原码反码补码表示都是其自身加个符号位 0,反码将符号位改为 1 即可
>   - 移码:以 4 位正数为例,只需用 10000 加上该数即可。如
>
> $+1010 \to 1,0000+1010=1,1010$
>
> - 负数
>   - 原码:最前面加符号位 1,如
>
> $-1010 \to 1,1010$
>
>   - 补码:最前面加符号位 1,除符号位外按位取反再+1,如
>
> $-1010 \to 1,1010 \to 1,0101+1 \to 1,0110$
>
>   显然,逆过程即为先-1,除符号位外按位取反,把符号位变成符号,如
>
> $1,0110 \to 1,0101 \to 1,1010 \to -1010$
>
>   其他类型的逆转换不再赘述
>   - 反码:最前面加符号位 1,按位取反,如
>
> $-1010 \to 1,1010 \to 0,0101$
>   - 移码:同正数移码,如
>
> $-1010 \to 1,0000-1010=0,0110$
>   - 小数也一样,小数点前的数位为符号位,以补码为例
>
> $-0.1010 \to 1.1010 \to 1.0101 \to 1.0110$
>
> - 浮点数:
>   - 阶码:1 位符号位与数值位,数值位表示整数
>   - 尾数:1 位符号位与数值位,数值位表示小数
>   - 表示范围不难推导,无需记忆
>   - 下溢:取阶码最小(负数最大值),尾数数值为+1
>   - 上溢:取阶码最大,尾数数值最大
>   - 规格化(基 2):尾数真值 $\frac{1}{2} \le |S| < 1$
>   - 原码:不论正负,第一数位为 1
>   - 补码:符号位与第一数位不同(存在特例$-\frac{1}{2},-1$)
>   - 规格化后的表示范围推导方式类似,不赘述

⚠ Warning

- 0 的原码表示按{整数、小数},{正数、负数}共四种情况。
- 被判定为机器零的浮点数:
  - 尾数为 0 时
  - 阶码小于最小阶码时
- 表示范围问题(可能考选择)
  - 原码、反码、移码表示的正负数个数相同
  - 补码表示的负数个数比正数多 1

Fine,经过了漫长的(约 10 分钟左右的快速复习),我们终于可以开始相关的例题了。下面将挑选出部分例子进行快速浏览。

🔥 Tip

寄存器类题目的考察一般选取边界值,以八位寄存器为例,常见的边界值有

- 补码表示的负数最大值: $(-128)_{10} = (1000\_0000)_2$
- 0xFF,即全为 1
  - 若为补码,则表示-1
  - 若为反码,则表示-0
  - 若为原码,则表示-127

非边界值的情况均可直接和真值进行比对

Example

1. (hw1.1)设寄存器内容为 1000_0000, 若他对应的真值是 −128, 则该机器数是采用 ____ 表示的。
2. (hw1.2)设寄存器的内容是 FFH, 若他对应的真值是 −0, 则该机器数是采用 ____ 表示的。

i Solution

1. 补码
2. 反码

🔥 Tip

浮点表示的考察更可能以主观题的形式出现,因为可写的步骤较多,主要为以下两种

- 阶码和尾数从原码、补码、反码、移码中选择两种进行表示
- 给定表示范围设计浮点表示
  - 阶码定范围
  - 尾数定精度,最大精度即尾数位数尽可能多

Example

1. (hw1.3) 在浮点机中, 若要求机器 0(即尾数为 0 且阶码最小的数)在计算机中表示为全“0”, 则阶码应用 ____ 表示。
2. (hw1.7)设浮点数的格式为: 阶码 5 位(含 1 位阶符), 尾数 11 位(含 1 位数符)。写出分数 $-\frac{41}{128}$ 在以下不同要求下对应的机器数:
   - 阶码和尾数均为原码
   - 阶码和尾数均为补码
   - 阶码为移码, 尾数为补码
3. (hw1.5)设 32 位长浮点数, 其中, 阶码 12 位(含 1 位阶符), 尾数 20 位(含 1 位数符), 均采用补码表示, 且尾数采用规格化形式, 则:
   - 能表示的最大正数是?
   - 非 0 最小正数是?
   - 绝对值最大的负数是?
   - 绝对值最小的负数是?

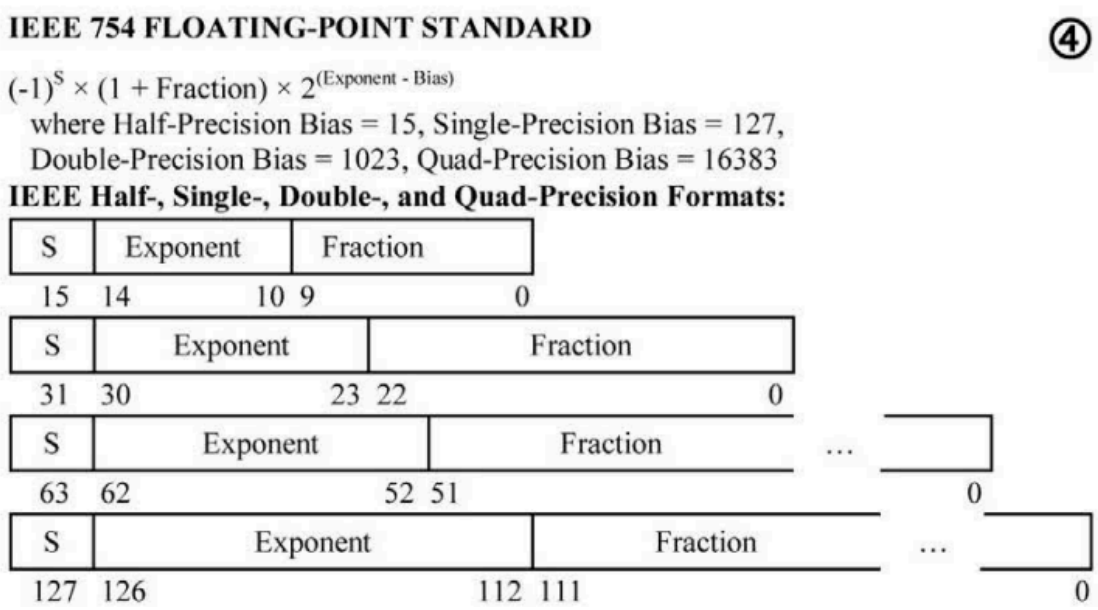
   均采用十进制表示, 无需化简
4. (hw1.6)设浮点字长为 32 位, 欲表示 ±6万之间的十进制数, 在保证数的最大精度条件下, 除阶符和数符各取一位外:
   - 阶码和尾数各取几位
   - 按这样分配, 该浮点数溢出的条件是什么

i Solution

1. 移码
2. 按如下步骤进行
   - 进制转换
     - 分母定位数,$128 = 2^7$,故小数点后有 7 位
     - 分子定表示,$41 = 32 + 8 + 1 = 101001$
     - 转换结果为 $-\frac{41}{128} = -0.0101001$
   - 移动小数点
     - $-0.0101001 = -0.101001 \times 2^{-1}$
   - 按要求进行转换
     - 均为原码,则有阶码$-1 = 1,0001$, 尾数$-0.101001 = 1,1010010000$
     - 均为补码,则有
       - 阶码$-1 \to 1,0001 \to 1,1110 + 1 \to 1,1111$
       - 尾数$-0.101001_0000 \to 1,101001_0000 \to 1,010110_1111 + 1 \to 1,010111_0000$
     - 阶码为移码,尾数为补码,则有
       - 阶码$-1 \to 1,0000 - 1 \to 0,1111$
       - 补码由上一问得 $1,0101110000$
3. 即求表示范围,注意尾数规格化要求,补码的规格化符号位与第一数位不同
   - 正数:尾数形如 0.1*形式,最小即为*取 0、阶码取最小,最大即为*取最大,阶码取最大
   - 负数:尾数形如 1.0*形式,由于是补码表示,绝对值最小即为*取最大、阶码取最小,绝对值最大即为*取 0,阶码取最大
   - 结果如下
     - 正数:$\frac{1}{2} \times 2^{-2^{11}} \sim \left(1 - 2^{-19}\right) \times 2^{2^{11}-1}$
     - 负数:$(-1) \times 2^{2^{11}-1} \sim \left(-\frac{1}{2} - 2^{-19}\right) \times 2^{-2^{11}}$
4. 此类题解答可快速求解。最大精度即尾数位数尽可能多
   - 阶码取最接近表示范围的位数,$1024 \times 64 = 2^{16}$是很容易快速得到的,故阶码除符号位外,为表示 16,数值至少为 5 位,由最大精度要求设为 5 位数值位
   - 尾数位数除符号位外有 $32 - 5 - 2 = 25$位
   - 上溢即阶码大于最大阶码+31,下溢即阶码小于最小阶码-31

Tip

IEEE754 的题目与浮点数表示并无太多不同,且可从 RISC-V 卡片上查看格式



可按如下步骤进行:

- 符号位填充到 S 上
- 进制转换
- 小数点移动使得小数点前的数为 1
- 计算指数 exponent = bias + value

Example

1. (hw1.4.1)十进制数 −0.875 用 IEEE754 单精度浮点数表示为 ____ H 。
2. (hw1.4.2)十进制数 −412 用 IEEE754 单精度浮点数表示为 ____ H 。
3. (hw1.4.3)给定 IEEE 754 单精度浮点数 C401_0000H, 该数的十进制表示为 ____。

Solution

1. 按步骤进行即可
   - 符号为负,S=1
   - 进制转换,$(0.875)_{10} = (0.5 + 0.25 + 0.125)_{10} = (0.111)_2$
   - 小数点移动, $0.111 \rightarrow 1.11 \times 2^{-1}$
   - 计算指数 exponent = 127 + (-1) = 126 = $(0111\_1110)_2$
   - 最终结果为 1_01111110_110⋯0,化为十六进制即为 0xBF60_0000
2. 按步骤进行即可
   - S = 1
   - $(412)_{10} = (256 + 128 + 16 + 8 + 4)_{10} = (110011100)_2$
   - $110011100 = 1.10011100 \times 2^8$
   - $127 + 8 = 128 + 7 = (1000\_0111)_2$
   - res = $(1\_10000111\_10011100...0)_2 = (C3CE\_0000)_{16}$

下面对本章节必考的计算题进行归纳整理。由不知出处的参考资料明确了除法不考大题,故本部分略去对除法的讨论,如有任何顾虑请自行复习。

Tip

首先,定点数计算题出现的只可能是原码和补码计算,故反码和移码以下不做考虑。 运算部分除了基本的数的表示和运算知识外,还有单双符号位的概念。

- 单符号位:与先前的用法一致,不赘述
- 双符号位
  - 最高位为真正的符号位
  - 另一符号位用于判断溢出,结果的双符号位相同则未溢出,不相同则溢出(可能会问)
  - 转换方式类似
    - 正数不论原码补码,在最前面加上 00
    - 负数
      - 原码则在最前面加上 11 即可
      - 补码在最前面加上 11,除了两位符号位外按位取反,再加上 1

Tip

加减运算题目步骤

- 化补码计算
- 结果按要求表示,多为用十进制表示

Example

(hw2.1)设机器数长 8 位(含 1 位符号位), 已知 $A = \frac{11}{64}, B = -\frac{13}{32}$,采用(单符号位)补码运算规则计算 $A - B$

Solution

- 进制转换
  - $A=\left(\frac{11}{64}\right)_{10}=(0.0010110)_2$
  - $B=\left(-\frac{13}{32}\right)_{10}=(-0.0110100)_2$
- 化补码
  - $[A]_{补}=0.0010110$
  - $[-B]_{补}=0.0110100$
- 计算
  - $[A-B]_{补}=[A]_{补}+[-B]_{补}=0.1001010$
  - $A-B=+0.1001010$
- 结果转换,可直接在上一步完成
  - $+0.1001010=\frac{37}{64}$

Tip

乘法运算为考察重点,思维难度不大但需要谨慎,本部分通过例子讲解

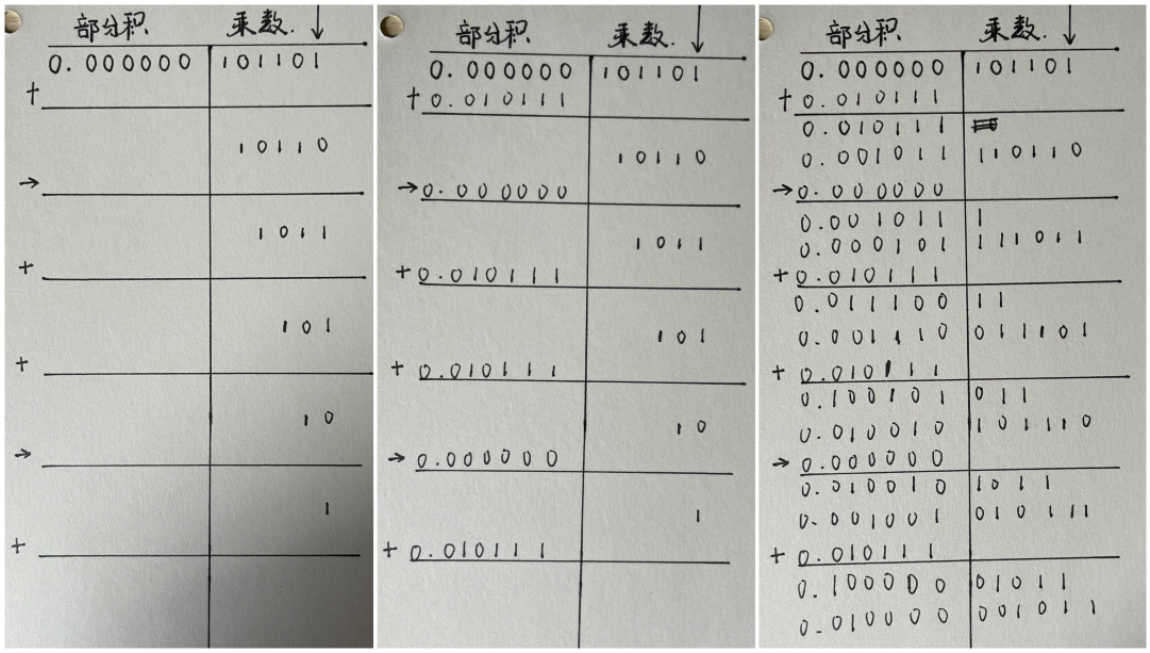
Example

(hw2.2)用原码一位乘计算 $x\times y$

- $x=0.010111$
- $y=-0.101101$

Solution

原码乘只需将数值部分进行相乘,符号位单独处理。乘法的关键在于每一步是加上被乘数还是加 0.下面给出笔者的做题方式,仅供参考。



- 乘数为 6 位,一共算 6 次,画好分界线
- 每一步是加被乘数还是 0 由末位决定,故先列出每一步的操作
  - + 表示加上被乘数,→ 表示加上 0,即仅移位
- 填上要加的数,把→改成+(此处忘改了)
- 从上到下,加一次移一次,加一次移一次
- 得到最终结果,添加符号位即可
- 答案为−0.010000_001011

Example

(hw2.3)用 Booth 算法,采用双符号位计算 $[x \times y]_{补}$

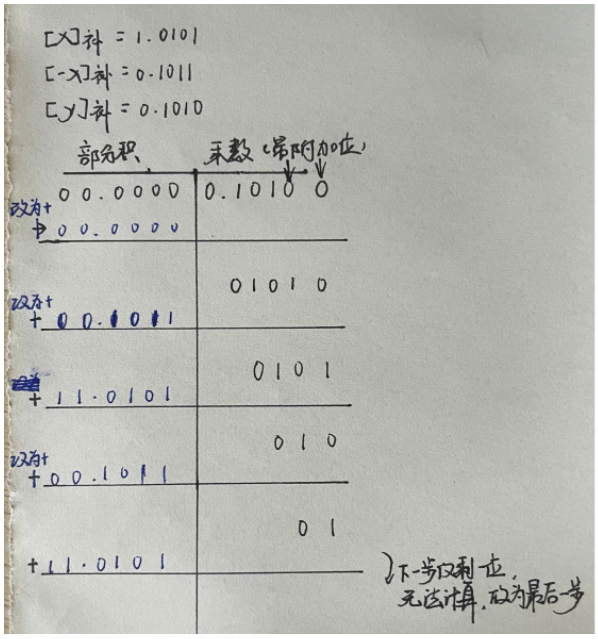
- $x = -0.1011$
- $y = 0.1010$

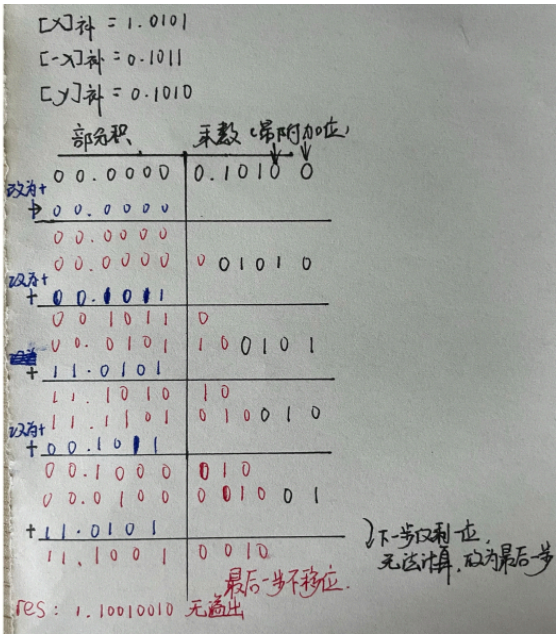
Tip

Booth 乘法操作类似,主要有以下细节不同

- 乘数需要补上一个 0
- 操作由末两位决定
  - 00 或 11 表示+0,10 表示-,01 表示+
- 最后一步不移位

Solution





补充几点说明:

- 补码相乘时符号位需参与运算,即乘数带符号位
- 双符号位运算只需要部分积的部分运算带双符号即可,乘数无需双符号
- 整数乘法和小数乘法类似,不赘述。
  - 但整数除法和小数除法有略微区别,不考所以不讲

Tip

浮点运算的步骤较为固定,一般不要求做舍入处理,但为防止出题人发电,本部分会略微提及相关内容。主要按照以下步骤进行:

- 对阶:小阶向大阶看齐
- 尾数求和
- 规格化

Example

1. (hw2.6)假设阶码取 3 位, 尾数取 6 位(均不包括符号位),采用双符号位计算
   - $[2^5 \times \frac{11}{16}] + [2^4 \times (-\frac{9}{16})]$

Solution

1. 按部就班进行
   - $x = 2^5 \times \frac{11}{16}, y = 2^4 \times -\frac{9}{16}$
   - 转补码表示:$[x]_{补} = 00,101;00.101100 [y]_{补} = 00,100;11.011100$
   - 小阶向大阶看齐,阶差为 1,y 尾数右移一位,阶码+1。即$[y]_{补} = 00,101;11.101110$,该阶码即为未规格化前的阶码
   - 尾数求和 $[S_x]_{补} + [S_y]_{补} = 00.011010$
   - 规格化
     - 若无溢出,左规直到符号位与第一数位相同.本题即无溢出,左规一次即可,尾数左移 1 位,阶码-1, 结果为 $00,100;00.110100$
     - 若溢出,右规,尾数右移 1 位,阶码+1,舍入如下:
       1. 若无需做任何舍入处理,直接右移
       2. 恒置 1 法:右规多出的位数至少有 1 个 1,则末位恒置 1.
       3. 0 舍 1 入法:末位后面的一位若为 0,则丢弃; 若为 1,则向末位进 1
       4. 例如,如果上面的结果的尾数为01.110100,则发生溢出,需要进行右规。右移一位得到$00.111010(0)$,括号中的 0 即为右规多出的位数。如果括号中为 1 且采取 0 舍 1 入法,则右规结果为00.111011。当然,如果向末位加 1 后导致了溢出,则需要再一次右规。
   - 无舍入,无溢出 $x + y = 00,100;00.110100 = 2^4 \times \frac{13}{16}$

Example

补充一道右规的题目

设$x = 0.1101 \times 2^{10}, y = 0.1011 \times 2^{01}$,求 x + y. (除阶符 2 位、数符 2 位外,阶码取 3 位,尾数取 6 位)

Solution

- $[x]_{补} = 00,010;00.110100, [y]_{补} = 00,001;00.101100$
- 阶差为 1,小阶对大阶。$[y]_{补} = 00,010;00.010110$
- 尾数求和:$[S_x]_{补} + [S_y]_{补} = 01.001010$
- 符号位为 01,发生溢出,需右规。尾数右移,阶码+1.
  - $x + y = 00,011;00.100101(0)$,采用 0 舍 1 入法直接舍去
- $x + y = 0.100101 \times 2^{11} = \frac{37}{64} \times 2^3$

Chapter 3: 指令系统与汇编

Memorize

1. CISC & RISC
   - ISA 的功能设计的任务是确定硬件支持哪些操作,采用统计的方法。
   - CISC
     - 强化指令功能以减少指令条数,提高系统性能
     - 面向目标程序、高级语言和编译器的优化
   - RISC
     - 简化指令系统,用高效的方法实现最常用的指令
     - 充分发挥流水线的效率,降低 CPI
2. 在汇编语言中,用助记符代替机器指令的操作码,用地址符号或标号代替指令或操作数的地址
   - RISC-V 指令格式:`op dst,src1,src2`
   - RV32I 和 RV64I 指令集的区别在通用寄存器的位数
   - RISC-V 是小端机
3. 指令系统
   - IS: Instruction Set 指令集 — CPU 设计的依据
   - ISA: Instruction Set Architecture 指令集系统架构
   - 系列机是指基本指令系统相同、基本系统结构相同的计算机
4. 评价指令系统的指标有方便硬件设计、方便编译器实现、性能更优、成本功耗更低。
   - 性能要求中关于规整性,x86 还包括对称性、匀齐性、一致性的定义
5. RISC 的特点有
   - 指令条数少,保留使用频率最高的指令
   - Load/Store 架构
   - 指令定长、格式简单、寻址方式简单
   - CPU 设置大量寄存器
   - CPU 采用硬布线控制
   - 一个时钟周期完成一条指令

Example

1. IS 是指 ____ ,ISA 是指 ____ 。系列机是指 ____ ,____ 的计算机
2. 评价指令系统的指标有 ____, ____, ____, ____.
3. RISC 的特点有哪些
4. 比较 CISC 和 RISC
5. 在汇编语言中,用____代替机器指令的操作码,用地址符号或标号代替____。

Tip

指令的汇编应借助 RISC-V 考试卡片进行。下面对如何利用卡片进行简单概括:

1. 首先在 opcode 里面找到相应的指令,在指令的前面做个标记。

OPCODES IN NUMERICAL ORDER BY OPCODE

MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT6/7 OR IMM	HEXADECIMAL
lb	I	0000011	000		03/0
lh	I	0000011	001		03/1
lw	I	0000011	010		03/2
ld	I	0000011	011		03/3
lbu	I	0000011	100		03/4
lhu	I	0000011	101		03/5
lwu	I	0000011	110		03/6
fence	I	0001111	000		0F/0
fence.i	I	0001111	001		0F/1
addi	I	0010011	000		13/0
slli	I	0010011	001	000000	13/1/00
slti	I	0010011	010		13/2
sltiu	I	0010011	011		13/3
xori	I	0010011	100		13/4
srli	I	0010011	101	000000	13/5/00
srai	I	0010011	101	010000	13/5/20
ori	I	0010011	110		13/6
andi	I	0010011	111		13/7
auipc	U	0010111			17
addiw	I	0011011	000		1B/0
slliw	I	0011011	001	000000	1B/1/00
srliw	I	0011011	101	000000	1B/5/00
sraiw	I	0011011	101	010000	1B/5/20
sb	S	0100011	000		23/0
sh	S	0100011	001		23/1
sw	S	0100011	010		23/2
sd	S	0100011	011		23/3
add	R	0110011	000	0000000	33/0/00
sub	R	0110011	000	0100000	33/0/20
sll	R	0110011	001	0000000	33/1/00
slt	R	0110011	010	0000000	33/2/00
sltu	R	0110011	011	0000000	33/3/00
xor	R	0110011	100	0000000	33/4/00
srl	R	0110011	101	0000000	33/5/00
sra	R	0110011	101	0100000	33/5/20
or	R	0110011	110	0000000	33/6/00
and	R	0110011	111	0000000	33/7/00
lui	U	0110111			37
addw	R	0111011	000	0000000	3B/0/00
subw	R	0111011	000	0100000	3B/0/20
sllw	R	0111011	001	0000000	3B/1/00
srlw	R	0111011	101	0000000	3B/5/00
sraw	R	0111011	101	0100000	3B/5/20
beq	SB	1100011	000		63/0
bne	SB	1100011	001		63/1
blt	SB	1100011	100		63/4
bge	SB	1100011	101		63/5
bltu	SB	1100011	110		63/6
bgeu	SB	1100011	111		63/7
jalr	I	1100111	000		67/0
jal	UJ	1101111			6F

2. 然后根据找到的指令对应的 FMT 类型查找对应的机器码格式,在前面做个标记。

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Tip

3. 然后根据以下指令格式,完善相关参数(卡片没有,必须记忆)
   - R 型指令:`op rd,rs1,rs2`
   - I 型指令:
     - 算术或逻辑运算: `op rd,rs1,imm`
     - Load: `op rd,offset(rs1)`
   - S 型指令: `op rs2,offset(rs1)`
   - SB 型指令: `op rs1,rs2,label`
   - U 型指令: `op rd,imm`
   - UJ 型指令: `op rd,label`

为集中注意力,可以先完成第三步,再完成第一二步。

Warning

注意 S 型指令 `op rs2,offset(rs1)`,第一个出现的寄存器是 `rs2`,不是 `rs1`

Example

(hw3.2) 给定 RISC-V 汇编指令如下

```
L1:
  addi  x11,x11,-1
  beq   x10,x11,L1
```

其中 `beq x10,x11,L1` 对应的机器指令表示为 ____。

Solution

先确定寄存器和立即数的内容后再去查表

- beq 为条件跳转类 SB 型,格式为 `beq rs1,rs2,label`,因此可得到:
  - `rs1 = x10 = 0b01010`
  - `rs2 = x11 = 0b01011`
  - `imm = -4 = 0b1100(补码,扩展可等需要时再扩)`
- 查表得到 `beq` 的 OPCODE 为 1100011,FUNCT3 为 000
- SB 型指令格式为| imm[12|10:5] | rs2 | rs1 | funct3 | imm[4:1|11] | opcode |
- 填空得 1_111111_01011_01010_000_1110_1_1100011
- 每四位进行划分:1111_1110_1011_0101_0000_1110_1110_0011
- 写成十六进制得:0xFEB5_0EE3

Tip

指令的反汇编可从下图得到启发:

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

- 用指令的低 7 位查找 OPCODES IN NUMERICAL ORDER。由于该表已排序,故能很快找到对应的指令
- 根据指令对应的类型进行反汇编
  - 同一个 opcode 若有多个匹配指令,则进一步使用 funct3 进行鉴别。

Example

(hw3.6) 给定十六进制机器指令 0x0010_80A3,查表写出其对应的汇编指令。

Solution

- 不建议一开始就直接转二进制,第一步应着重于快速鉴别指令。
- 由低 8 位 0xA3 得 `opcode=0100011`, 查表得到其对应四条指令 `sb,sh,sw,sd`,均为 S 型指令,故按照 S 型指令的格式进行拆分
- 0010_80A3
- 0000000_00001_00001_000_00001_0100011
- 进一步得到如下信息
  - `rs2 = 1`
  - `rs1 = 1`
  - `funct3 = 000` — `sb` 指令
  - `imm = 1`
- 最终结果为 `sb x1,1(x1)`

Tip

另一种考察方式是阅读代码,人工执行代码。这一部分在记住各类指令基本格式的基础上,有不懂的就查卡片指令中的 `Description(in verilog)`即可

Example

(hw3.5) 考虑以下 RISC-V LOOP:

```
LOOP:
  beq   x6,x0,DONE
  addi  x6,x6,-1
  addi  x5,x5,2
  jal   x0,LOOP
DONE:
```

- 假设 x_6 初值为 10,x_5初值为 0,则x_5最终的十进制值为 ____。
- 假设 x_6 的初值为 N, 总共执行了 ____ 条指令。

Solution

人工执行,一般没什么问题。下面为解题示例,仅供参考。

- 首先关注标签的起止指令
  - `beq x6,x0,DONE`: 当 `R[x6] == 0` 时跳转到 DONE 标签处,循环体内应对 `x6` 进行更新,否则会产生死循环
  - `jal x0,LOOP`:无条件跳转至 LOOP 处
- 其次关注内部指令,不难发现每轮循环 `x6` 自减 1, `x5` 自增 2
- 因此,当 `x6` 初值为 10 时,循环体执行 10 次,`x5 = 20`.
- 每次循环执行 4 条指令,共执行 N 次循环,加上退出循环的测试指令,共执行 `4N + 1` 条指令。

Tip

更进一步的考察方式是利用汇编进行编程,考虑到如果让同学们自己写一个程序,试卷上的答案难以验证正确性(除非题目极其简单),较大概率可能考察程序填空题。为了方便批改,每一条指令可能都进行了注释,如作业题所做的一样。

Example

(hw3.7)对于以下的 C 语句,按要求补充相应的 RISC-V 汇编代码。

```
// C 语言
// long long int A[1024],B[1024];
B[g] = A[f] + A[f-1];
```

- 变量 f,g 分别分配给寄存器 x5,x6
- 数组 A,B 的地址分别在寄存器 x10,x11 中
- 注意:`long long int` 是 64 位

```
# RISC-V 汇编语言
slli  x30,x5,3    # x30 = f * 8
add  x30,x10,x30 # x30 = &A[f]
slli  x31,x6,3    # x31 = g * 8
_____________________ # 计算 B[g]的地址,x31 为目标寄存器
_____________________ # 载入 A[f]数据,x5 为目标寄存器
_____________________ # 计算 &A[f-1],x12 为目标寄存器
_____________________ # 载入 A[f-1],x30 为目标寄存器
_____________________ # 加法运算,x30 为目标寄存器
_____________________ # 将计算结果存到 B[g]
```

Solution

- 访存操作
  - 计算地址:`add addr,baseAddr,offset`
  - 加载数据:`ld  dst,0(addr)`
    - 使用的加载指令视数据大小而定

```
// C 语言
// long long int A[1024],B[1024];
B[g] = A[f] + A[f-1];
```

- 变量 f,g 分别分配给寄存器 x5,x6
- 数组 A,B 的地址分别在寄存器 x10,x11 中
- 注意:`long long int` 是 64 位

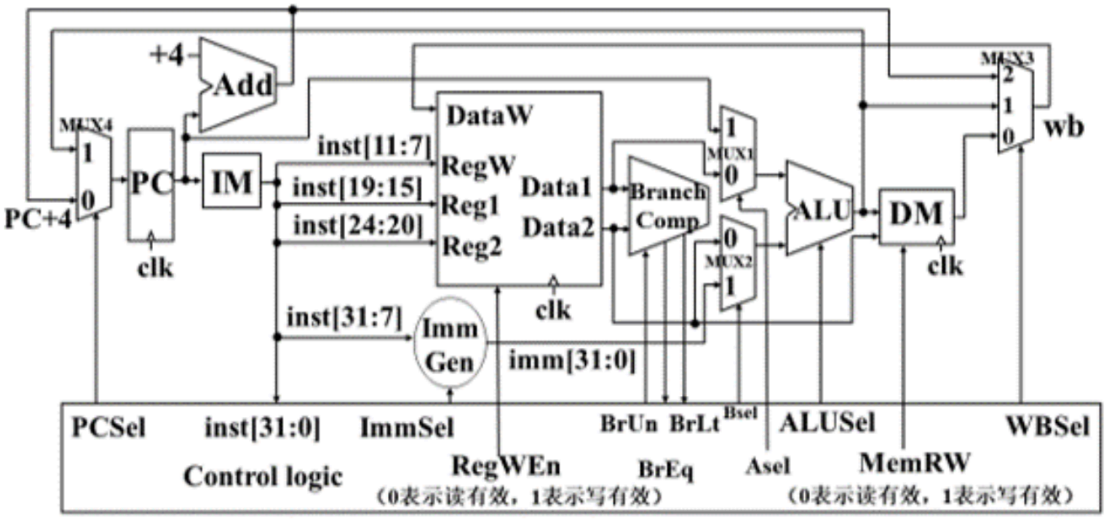
```
# RISC-V 汇编语言
slli  x30,x5,3    # x30 = f * 8
add   x30,x10,x30 # x30 = &A[f]
slli  x31,x6,3    # x31 = g * 8
# Answers
add   x31,x11,x31 # 计算 B[g]的地址,x31 为目标寄存器
ld    x5, 0(x30)  # 载入 A[f]数据,x5 为目标寄存器
addi  x12, x30,-8 # 计算 &A[f-1],x12 为目标寄存器
ld    x30, 0(x12) # 载入 A[f-1],x30 为目标寄存器
add   x30, x30,x5 # 加法运算,x30 为目标寄存器
sd    x30, 0(x31) # 将计算结果存到 B[g]
```

Chapter 4: 处理器设计

Tip

最可能考察的内容为根据数据通路图分析指令在处理器中的执行。

- 控制信号的值
- 哪些指令经过哪些部件



上图为作业题图,该图与 PPT 图的不同之处在于每一个多路选择器 MUX 都标上了 0 和 1,对于控制信号的含义也进行了注释,由 MUX 不同的值对应的信号线可知

- ASel: ALU 的操作数 1
  - 为 0 选择 data1
  - 为 1 选择 PC,多为跳转目标计算
- BSel: ALU 的操作数 2
  - 为 0 选择 data2
  - 为 1 选择立即数扩展
- WBSel:无需写回时该信号无所谓,即填*即可
  - 0 表示取数据存储器的指令,一般为 load 指令
  - 1 表示取 ALU 运算得到的结果
  - 2 表示取下一条指令的地址,一般为 jal 指令导致的
- PCSel:0 表示 Pc 自增 4,即顺序取下一条指令；1 表示取跳转指令的目标地址。
- RegWEn:0 表示读有效,1 表示写有效,无需写回的指令应保持为 0.
- MemRW:0 表示读有效,1 表示写有效,非 Store 指令应保持为 0。

Example

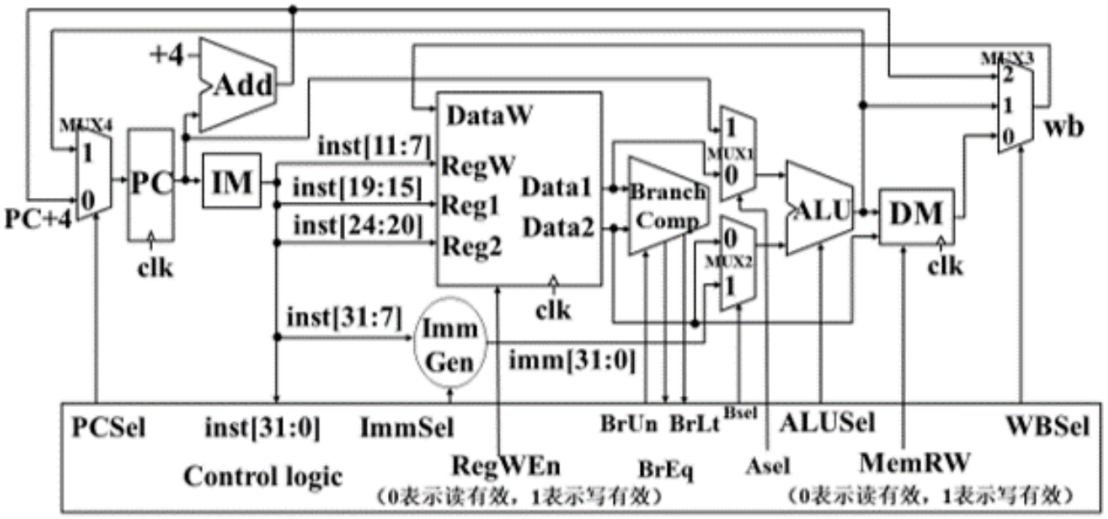


Figure 8: data path

(hw4)根据上面的数据通路图回答下列问题

1. 根据以下指令,填写数据通路控制信号(0,1,2,*)

指令	ASel	BSel	WBSel	PCSel	RegWEn	MemRW
1 `beq x0,x0,LOOP`	-	-	-	-	-	-
2 `lw x5.0(x29)`	-	-	-	-	-	-

2. 上面两条指令中
   - 发生数据访存的有 ____.
   - 发生指令访存的有 ____.
   - 使用符号扩展的有 ____.
3. 对于指令 2,如果 `MemRW` 控制线出错且 `MemRW=1`,则执行该指令时会发生什么?

Solution

1. 在了解了各个控制信号的含义后,只要了解指令的含义,就可直接填表。
   - `beq rs1,rs2,label`
     - 从寄存器堆读出两个数进行比较,若相等则跳转,两个寄存器都是 `x0`,显然相等。
     - ALU 用于计算跳转地址,操作数分别为 PC 值和立即数
     - 无需写回

指令	ASel	BSel	WBSel	PCSel	RegWEn	MemRW
1 `beq x0,x0,LOOP`	1	1	*	1	0	0

   - `lw x5,0(x29)`
     - 从寄存器堆读出 1 个数
     - ALU 用于计算访存地址,操作数为寄存器 data1 和立即数
     - 需写回

指令	ASel	BSel	WBSel	PCSel	RegWEn	MemRW
2 `lw x5.0(x29)`	0	1	0	0	1	0

2. 参照下面的 tips
   - 仅 load 类型指令会发生数据访存,故填 2
   - 所有指令都需进行指令访存,故填 1、2
   - 除 U 型指令外,涉及立即数的指令均需符号扩展,故填 1、2
3. `MemRW=1` 表示写数据存储器,而指令 2 为读数据存储器,故会将 `rs2` 寄存器的值写入数据存储器,即 `0(x29)` 中会被写入 data2 的值(如果需要,可将指令进行汇编得到 data2 是哪个寄存器的值),且向 `x5` 中写入不确定值(因为没读到想要的值)。

Tip

RISC-V 的六种基本指令格式
- R 型 :用于寄存器-寄存器操作
- I 型 :用于短立即数和取数 load 操作,需进行符号扩展
- S 型 :用于存数 store 操作,需进行符号扩展
- B 型 :条件跳转操作,偏移量最低位为 0,需进行符号扩展
- U 型 :长立即数操作,不需符号扩展
- J 型 :立即跳转操作,需符号扩展
  - jalr 是 I 型指令

Chapter 5: 流水线处理器

本章节的考察多以选填或者大题中的小问出现。流水线处理器的控制信号不太可能作为一道大题单独考察(如果有请问候出题人)。故本部分只针对小题进行复习。

> Tip

流水线的用途就是为了提高性能,因此性能计算是该板块的重要考点。需要掌握以下几点:

- 流水线的时钟周期由时钟周期最长的阶段所决定,拆分时也一般拆时间最长的阶段
- 任何指令都会经过流水线的各个阶段,即使部分阶段无需任何处理
  - 以 `add` 指令为例,在单周期处理器中,其不需访存,由于是组合逻辑,所以其执行时间不需要包含访存阶段的延迟
  - 但在流水线处理器中,该指令将经过各个流水寄存器,到达写回阶段必须经过访存-写回的流水寄存器,故执行时间为所有阶段之和。
- 流水线通过提高指令吞吐量以提高性能。计算加速比时遵循如下公式:
  - $\text{rate} = \frac{t_{\text{single}}}{t_{\text{pipeline}}}$,即单周期执行时间除以流水线执行时间。

> Example

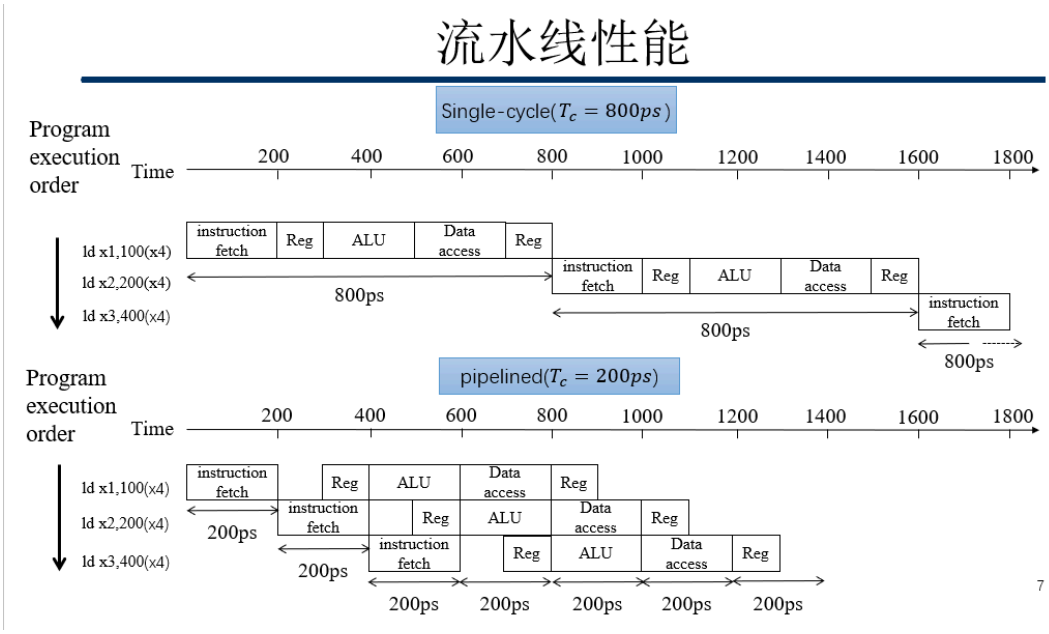
1. (hw4.2)假设指令由取指,译码、执行三个部件完成,每个部件的执行时间为 t ,对于流水线处理器,连续执行 8 条指令,则流水线加速比是 ____。
2. (hw4.5) 假设数据通路各流水线阶段延迟如下:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

   - 单周期处理器中,时钟周期为____ps.
   - 流水线处理器中,时钟周期为____ps.
   - 在单周期和流水线处理器中,一条 ld 指令的完整执行时间分别为 ____ps, ____ps.
   - 如果要将一流水级拆分为两个流水级,应拆分____ .

i Solution

1. 最好会画流水图,防止出现时间不等长的题目



当然,对于时间等长的题目可以直接用公式计算,但使用流水图分析是最可靠的。

- 本题用公式计算为 $\frac{3t\times 8}{3t+(8-1)t} = 2.4$
- 本题用流水图分析如下

-	-	-	-	-	-	-	-	-	-
IF	ID	EX	-	-	-	-	-	-	-
-	IF	ID	EX	-	-	-	-	-	-
-	-	IF	ID	EX	-	-	-	-	-
-	-	-	IF	ID	EX	-	-	-	-
-	-	-	-	IF	ID	EX	-	-	-
-	-	-	-	-	IF	ID	EX	-	-
-	-	-	-	-	-	IF	ID	EX	-
-	-	-	-	-	-	-	IF	ID	EX

实际上只需要分析左下方的斜对角线再加上最后一条指令的剩余阶段即可。

- 流水线耗时:$8t + 2t = 10t$
- 单周期耗时:$3t \times 8 = 24t$
- 相除即得 2.4

Tip

单周期处理器中,不同类型的指令执行时间是不同的。

- add 指令无访存延迟
- beq 指令无访存和写回延迟
- jal 指令无访存延迟,有写回延迟
- S 型指令无写回延迟

详细参考上一章内容

Tip

流水线另一个可考的就是冒险的判断和消除,题目一般为五级流水,即取指-译码-执行-访存-回写。记住以下几点:

- 数据冒险可通过停顿和前递两种方式进行处理
  - R-R 型冒险

```
1 addi  x1,x0,1
2 addi  x1,x1,1
```

指令 2 在译码阶段需要读出 x1,但指令 1 此时处于执行阶段,发生数据冒险。

  - 通过前递解决,即将执行阶段的结果前递至译码阶段
  - 通过停顿解决,即加空泡。

```
1 addi  x1,x0,1
  nop
  nop
2 addi  x1,x1,1
```

指令 2 处于译码阶段时,指令 1 处于回写阶段,若一个周期内寄存器可先写后读,则指令 2 能获得正确的数据。

  - Load-to-Use 型冒险

```
1 lw    x1,0(x0)
2 addi  x1,x1,1
```

指令 2 在译码阶段需要读出 x1,但指令 1 此时处于执行阶段,发生数据冒险。不仅如此,指令 1 在执行阶段无法得到结果,故仅有前递无法解决问题,必须停顿+前递

Example

1. (hw4.1)下面语句一定存在 ____ 冒险

```
ld  x31,0(x20)
add x31,x31,x21
```

2. 在下面的代码中添加最少的 nop 指令,让他无需处理数据冒险也能正确的运行在流水线处理器上(同一周期内,寄存器可先写后读)

```
addi  x11,x12,5
add   x13,x11,x12
addi  x14,x11,15
add   x15,x13,x12
```

Solution

1. load-to-use 数据冒险
2. 五级流水线中,无前递硬件支持时,数据相关的指令需要间隔两条指令

```
addi  x11,x12,5
nop
nop
add   x13,x11,x12
addi  x14,x11,15
nop
add   x15,x13,x12
```

- 可能的变式有:
  - 多周期访存,即IF-ID-EX-MEM1-MEM2-WB。关键点在于读写寄存器所间隔的流水级数
  - 添加了一定的前递支持,如可将 EX 阶段的数据前递。

Chapter 6: 存储器

Memorize

1. 缓存是为了加速主存的性能,辅存是为了扩大主存的容量(虚拟存储器)
   - Cache 由硬件实现、虚拟存储器由硬件和 OS 协同
2. 程序的局部性原理
   - 时间局部性:最近被访问的很可能再次访问
   - 空间局部性:访问的程序和数据往往集中在一小片存储区
   - 访问顺序:指令顺序执行比转移执行的可能性大
3. DRAM 的刷新有集中刷新、分散刷新和异步刷新三种方式
4. TLB 可加速主存和辅存间访问速度
5. RAID 技术中,RAID0 的容错性最差但性能最好,RAID1 的容错性最好。

Example

1. DRAM 的刷新有哪三种方式,为什么要刷新
2. (hw5.1)比较动态 RAM 与静态 RAM。
3. (hw3.1)一个 32 位字长的数据 0xabcd_ef12 连续存储在地址 0x0000 −0x0003 中
   - 若采用小端存储,则 0x0002 存储的数据是 ____。
   - 若采用大端存储,则 0x0003 存储的数据是 ____。

Solution

1. 集中刷新、分散刷新、异步刷新。存储电荷的电容放(漏)电
2.

-	DRAM	SRAM
基本存储元件	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

3. 分别写出两种方式的存储

-	0000	0001	0002	0003
大端法(高位数据存低位地址)	ab	cd	ef	12
小端法(低位数据存低位地址)	12	ef	cd	ab

然后根据要求选出对应数据即可。即 cd, 12

Tip

汉明码的考察题型较为固定

1. 给定数据,要求生成汉明码
   - 根据数据的位数 n ,用$2^k \geq n+k+1$确定冗余校验位的数目
   - 根据 position of $C_p = 2^p$放置冗余校验位
   - 二进制数中从最低有效位起第 $k(k=1,2,...)$ 位为 1 的数位一起配奇数个 1 或偶数个 1(配奇或配偶)
   - 实操请看例题解答
2. 给定汉明码,判断出错位置并恢复原始数据
   - 按配偶原则配置还是配奇原则,下面以配奇原则为例
   - 二进制数中从最低有效位起第 $k(k=1,2,...)$ 位为 1 的数位视为一个整体,如果有奇数个 1,则$P_{2^{k-1}}) = 0$,反之为 1,表示出错。
   - 将P_k从高到低拼成二进制数,该数的值即为出错的数位
   - 将出错的数位更正后删除所有校验位

Example

1. (hw5.3) 1000 按配偶原则生成的汉明码是,按配奇原则生成的汉明码是____。
2. (hw5.4) 接收到的按配偶原则配置的汉明码为 1100100,其中第____位出错,欲传输的原始代码是____.

Solution

1. 按照 tip 中的步骤进行
   - 1000 有 4 位,由$2^k \geq 4 + k + 1$得到 $k = 3$,即 3 位校验位
   - 放置校验位,得到

1	2	3	4	5	6	7
C_1	C_2	1	C_4	0	0	0

   - 开始配偶
     - 二进制最低有效位起第 1 位为 1 的数配偶,即 1,3,5,7 位配偶数个 1,故$C_1 = 1$
     - 二进制最低有效位起第 2 位为 1 的数配偶,即 2,3,6,7 位配偶数个 1,故$C_2 = 1$
     - 二进制最低有效位起第 3 位为 1 的数配偶,即 4,5,6,7 位配偶数个 1,故$C_4 = 0$
   - 得到汉明码 1110000
   - 配奇类似,此处省略,结果为 0011000
2. 本题为配偶
   -

1	2	3	4	5	6	7
1	1	0	0	1	0	0

   - 开始配偶
     - 二进制最低有效位起第 1 位为 1 的数配偶,即 1,3,5,7 位,此处确为偶数个 1,故$P_1 = 0$
     - 二进制最低有效位起第 2 位为 1 的数配偶,即 2,3,6,7 位配偶数个 1,此处为奇数个 1,故$P_2 = 1$
     - 二进制最低有效位起第 3 位为 1 的数配偶,即 4,5,6,7 位配偶数个 1,此处为奇数个 1,故$P_4 = 1$
   - 拼成二进制数 $P_4P_2P_1 = (110)_2 = (6)_{10}$故为第 6 位出错。
   - 更正并删除校验码如下
     -

1	2	3	4	5	6	7
-	-	0	-	1	1	0

   - 原始数据为 0110

Tip

Cache 的设计与计算类题目也较为固定,一般来说不会出现对 Cache 行替换的考察,复习的时候只需要掌握以下内容

- 根据指定的参数划分主存地址字段
  - 根据寻址方式确定主存地址宽度
  - 根据字块大小和字大小确定字块内地址宽度
  - 根据 Cache 容量和 Cache 参数确定组地址宽度
  - 确定主存字块标记宽度
- 计算命中率和与主存进行速度比较
  - 命中率 = (访存总数 - 不命中数) / 访存总数
  - 设 Cache 命中一次用时t_c,主存访存一次用时t_m,Cache 命中率为r
    - 提高倍数为 $\frac{t_m}{t_c\times r+t_m\times(1-r)}-1$
    - Warning

      提高了多少倍而不是提高到多少倍,注意 −1, 作业题此处不 −1 会扣分。

Example

(hw5.6) 某机主存容量 16MB,按字节寻址。Cache 容量 16KB,每字块有 8 个字,每字 32 位,四路组相连。

1. 画出主存地址中各字段的位数
2. 地址为 0xabcdef 的主存单元会被映射到 Cache 中的第几组?(用十进制表示,设起始组为第 0 组)
3. 设 Cache 初态为空,CPU 从主存第 0,1,2,⋯,89 号单元依次读出 90 个字,按此次序重复读 8 次,求 Cache 的命中率
4. 若 Cache 的读取速度为主存的 6 倍,试问有 Cache 和无 Cache 相比,速度约提高多少倍?

Solution

1. 按部就班操作即可
   - 主存 16MB,按字节寻址,由$16\text{MB} \div 1\text{B} = 16\text{M} = 2^{24}$得主存地址 24 有 24 位
   - 每字块 8 个字且字长 32 位得$8 \times 4\text{B} = 32\text{B} = 2^5 B$,即字块内地址为 5 位
     - 不管主存如何寻址,字块内偏移的基本单位始终是字节
   - 四路组相联得组大小为 $4 \times 32B = 2^7 B$,得组数为 $16\text{KB} \div 2^7 B = 2^7$,故组地址宽度为 7 位
   - 字块标记宽度为 $24 - 5 - 7 = 12$位
   - 因此,各字段位数如下图

主存字块标记	组地址	字块内地址
12 位	7 位	5 位

2. 不建议整个数转二进制,由上一问结果可得只需看低 12 位的高 7 位,故只看 0xdef
   - 0xdef = 0b1101_1110_1111
   - 组地址为 0b1101111,转成十进制为 $0b1100000 + 0b1111 = 2^6 + 2^5 + 15 = 64 + 32 + 15 = 111$
3. 一共有 90×8 个读请求,其中会发生读缺失的为每个块的第一个字,即读$0, 8, 16, ..., 88$时会发生缺失,且仅在第一次读取的时候会发生缺失,共 12 次缺失。由此可直接计算命中率为 $\frac{90\times8-12}{90\times8} = 98.33\%$
4. 根据 tips 中的公式可得 $\frac{t_m}{t_c\times0.9833+t_m\times(1-0.9833)} - 1 = \frac{1}{\frac{1}{6}\times0.9833+1-0.9833} - 1 = 4.54$

Tip

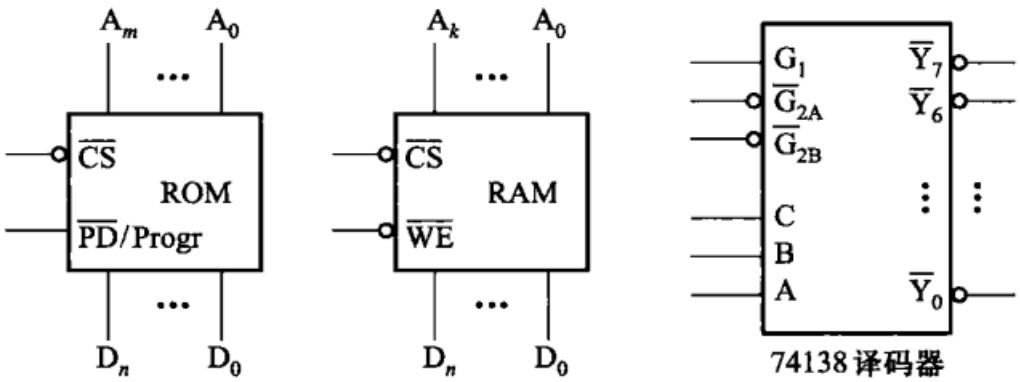
CPU-Memory 的连接类题目考察内容较为固定,弄清楚以下几点即可

- 字扩展或位扩展
  - 数据宽度满足要求但地址宽度不满足
  - 地址宽度满足要求但数据宽度不满足
- 74138 译码器的用法

弄清以上两点后,做题的流程较为固定

- 画出地址分配表
- 选择用于片选的地址信号及芯片
- 连接 CPU 和芯片
  - 片选连线
  - 地址连线
  - 数据连线

Tip



作几点说明:

- 头上带横杠的信号均为低电平有效,其余为高电平有效
- $\overline{CS}$: Chip Select,片选信号
- $\overline{PD}$: Programmed,不用管,接地就行。
- $\overline{WE}$: Write Enable 写使能信号
- 74138 译码器
  - 控制信号G_1为高电平有效,$\overline{G_{2A}}$和$\overline{G_{2B}}$为低电平有效,三个信号同时有效时译码器正常工作
  - 输出为低电平有效,对接到芯片使能时需注意
  - ABC 从二进制数的最低有效位开始排起,即$A=1,B=0,C=0$时,输出$\overline{Y_1}=0$
  - 一个 74138 译码器可判断三位地址,当对片选造成影响的地址信号多于 3 位时,需要另外的逻辑门进行辅助,必要的时候可以使用 2 个 74138 译码器

Example

(hw5.5)根据以下要求,选择合适的芯片,画出 CPU 与存储芯片的连接图

- CPU: 16 根地址线, 8 根数据线
- $\overline{MREQ}$:访存控制信号,低电平有效
- R/$\overline{W}$: 读写命令信号,高电平为读,低电平为写
- 可选芯片
  - ROM: 2K × 8 位,4K×4 位,8K × 8 位
  - RAM: 1K × 4 位,2K×8 位,4K × 8 位
  - 74138 译码器
  - 门电路任选
- 最小 4K 地址为系统程序区,4096-16383 地址范围为用户程序区

i Solution

解题的步骤较为固定

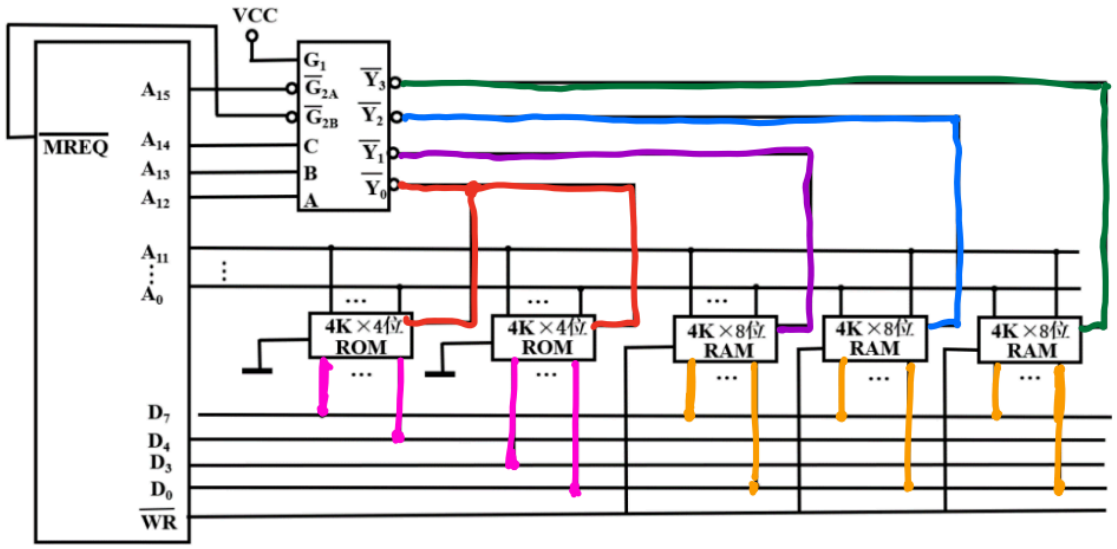
- 画地址分配表(熟练之后这一步可省略,但或许错的时候能有过程分吧)
  - 16 位地址
  - 系统程序区为最小 4K 地址, $4K = 2^{12}$,即需要地址的低 12 位。
  - 内存地址范围为 0-16383,即$16K = 2^{14}$,即需要地址的低 14 位
  - 起始地址为 0x0,终止地址为 0x0011_1111_1111_1111

A15	A14	A13	A12	A11	A10	⋯	A0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
-	-	-	-	-	-	-	-
0	0	0	1	0	0	0	0
0	0	0	1	1	1	1	1
-	-	-	-	-	-	-	-
0	0	1	0	0	0	0	0
0	0	1	0	1	1	1	1
-	-	-	-	-	-	-	-
0	0	1	1	0	0	0	0
0	0	1	1	1	1	1	1

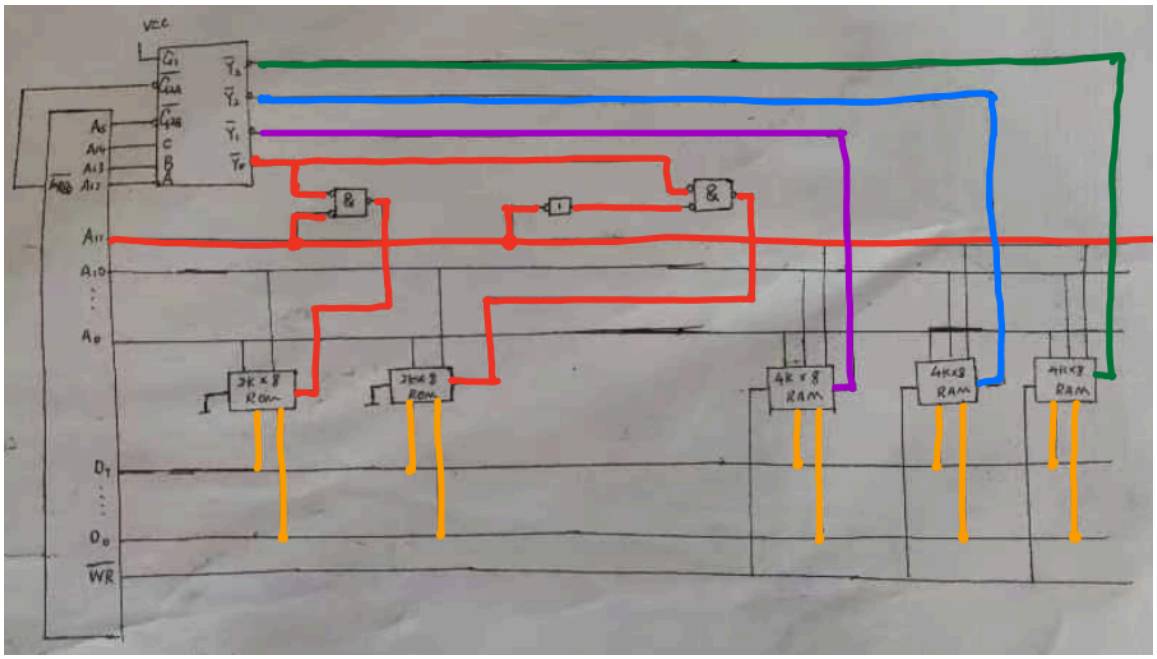
- 选取芯片
  - 由地址表可知, A13 和 A12 可用于对 4K 地址空间的片选
  - 可选用不同的芯片进行组合,系统程序只读,选择 ROM,用户程序可读可写,选 RAM
    1. 系统程序空间
       - ROM 2K × 8 位两片,进行字扩展,在 A13 和 A12 为 00 的情况下,即 74138 译码器输出 000 时,另外用 A11 进行对 ROM 的片选
       - ROM 4K × 4 位 两片,进行位扩展,将数据线分为 D0-D3, D4-D7 两组分别接到对应的 ROM 上
       - ROM 8K × 8 位,理论上可行,只要 ROM 的高地址位接地即可,但是画图麻烦,不建议采用大于所需地址空间的芯片
    2. 用户程序空间
       - 选用 3 片 4K × 8 位 RAM 的图最好画
       - 理论上选用 6 片 2K × 8 位 RAM 也是可以的,只需要另外用 A11 进行 4K 程序空间内部的 2K 片选即可。
    3. 多余的地址线(如果剩余较多的话)可用或逻辑门或一起后接到 74138 译码器的低电平有效控制信号。
- 有序画图
  - 先把控制信号连接上,即片选逻辑和读写使能信号。
  - 再把地址信号和数据信号连上

i Solution

以下图片摘自顾老师发的作业详解



如果选用 2 片 2K × 8 位的 ROM,则连接图如下:



Tip

此类题目为重点题型,变式较多,下面针对几种变式做一个简单浏览。

1. 系统程序工作的时候一般会产生临时数据,这一部分数据通常放在系统程序工作区中。同样的,用户程序也有自己的用户程序工作区。由此自然可以在芯片数量上进行变式
2. 程序和程序工作区也可以分开,即系统和用户程序分配低位地址,系统和用户的数据分配高位地址,提高空间局部性。

变式主要为以上两种,即

- 加量:增加程序工作区,选用更多的 RAM
- 换位:分配高位地址进行使用

Example

1. 40. 设 CPU 共有 16 根地址线,8 根数据线,并用$\overline{MREQ}$作为访存控制信号(低电平有效),用$\overline{WR}$作为读/写控制信号(高电平为读,低电平为写)。现有芯片及各种门电路(门电路自定),如图 4.16 所示。画出 CPU 与存储器的连接图,要求:

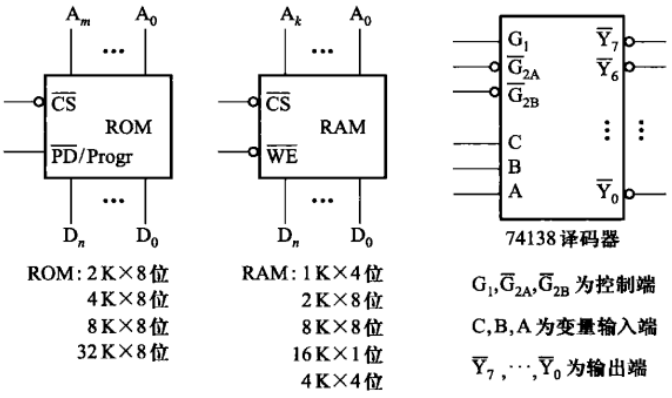


图 4.16 第 40 题芯片图

(1) 存储芯片地址空间分配:最小 4K 地址空间为系统程序区;相邻的 4K 地址空间为系统程序工作区;与系统程序工作区相邻的是 24K 用户程序区。

(2) 指出选用的存储芯片类型及数量。

(3) 详细画出片选逻辑。

2. 38. 在 36 题给出的条件下,画出 CPU 与存储芯片的连接图,要求:

(1) 存储芯片地址空间分配:0 ~ 8191 为系统程序区;8192 ~ 32767 为用户程序区;最大 4K 地址空间为系统程序工作区。

(2) 指出选用的存储芯片类型及数量。

(3) 详细画出片选逻辑。

i Solution

1.

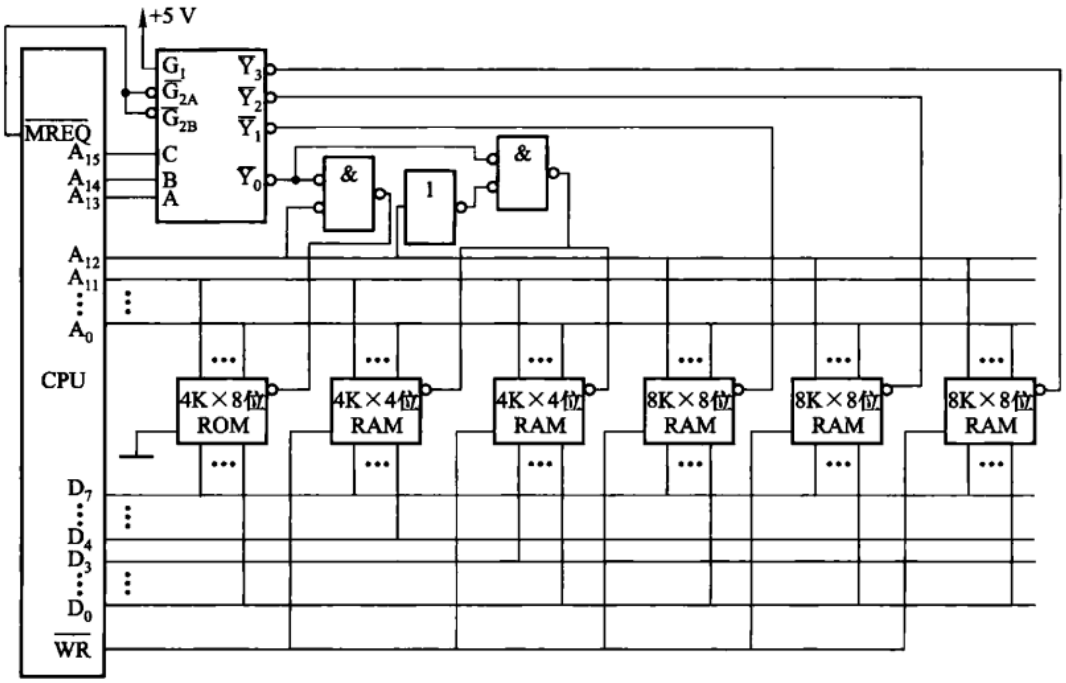


图 4.29　第 40 题答图

- 74138 译码器用以对最高三位地址进行判断,片选需要的其他地址位则可用逻辑门进行保证

2.

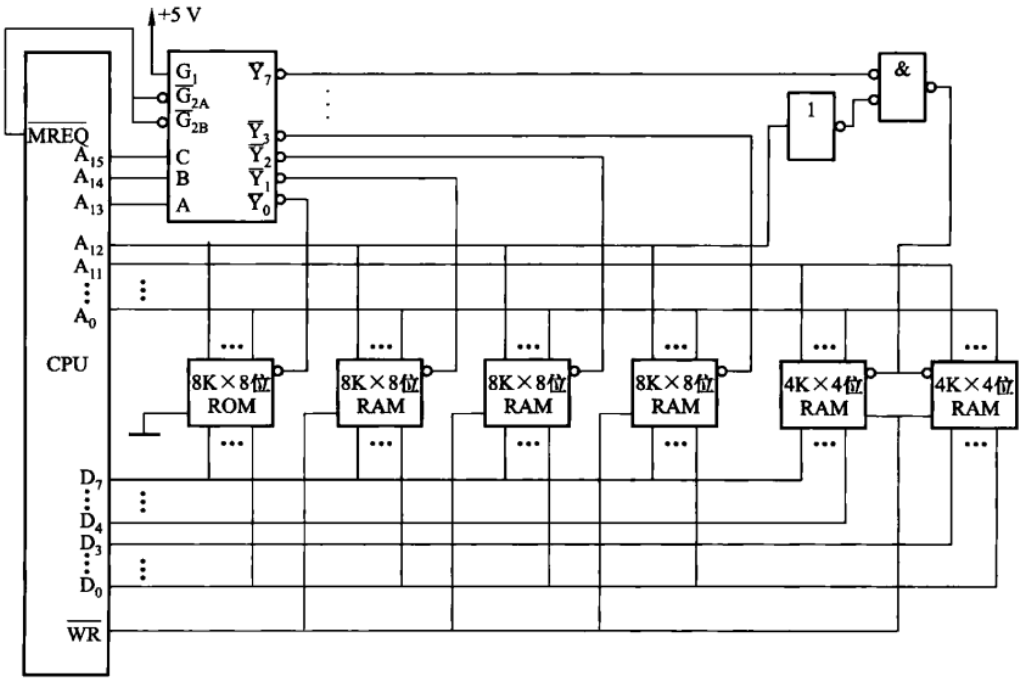


图 4.27　第 38 题答图

- 最大 4K 即 0x1111_0000_0000_0000 — 0x1111_1111_1111_1111,74138 译码器保证最高三位均为 1,再加上 A12 为 1 即可选中该片地址区域

Chapter 7: 系统总线

> ! Memorize

1. 总线是连接各个部件的信息传输线,是各个部件共享的传输介质.
2. 总线可分为片上总线、系统总线、通信总线
   - 片上总线:芯片内部的总线
   - 系统总线
     - 数据:双向
     - 地址:单向
     - 控制:有入有出
3. 主设备可控制总线、从设备响应总线命令
4. 总线传输
   - 目的:解决通信双方协调配合问题
   - 申请 → 寻址 →传输 →结束
5. 总线判优控制的集中方式
   - 链式查询:固定优先级
   - 计数器定时查询
     - 每次从 0 开始:序号越低优先级越高
     - 从上一次结束的地方开始:相同优先级
   - 独立请求
     - 优先级可灵活变化,根据排队逻辑确定。
6. 总线通信方式有同步通信、异步通信、半同步通信、分离式通信四种
   - 同步通信
   - 异步通信
     - 不互锁: 单机
     - 半互锁: 多机
     - 全互锁: 网络通信
   - 半同步通信
   - 分离式通信:充分提高总线利用率

Tip

总线部分的考察多以选填的形式出现,复习过程中应侧重概念的理解记忆。另外,通信方式常用现实例子进行类比考察,需要着重理解。

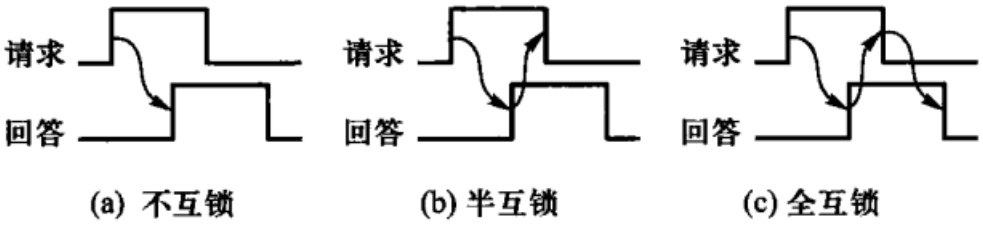


图 3.4 异步通信的三种方式

1. 请求发出后才有回答,这恒河里,不互锁。
2. 没有回答前一直保持请求,即对方不回消息我就刷屏。(我舔对方,对方锁我)
3. 请求没撤销前一直保持回答,即我没说收到对方就要一直刷屏。(对方舔我,我锁对方)

分离式通信可类比为寄信,只需要把信放到送给邮递员就可以了,一切交给邮局处理。

Example

1. (3.4.1.30)在单机系统中,CPU 向存储器写信息,通常采用____类型的联络方式。
2. (3.4.1.29)在多机系统中,某个 CPU 需访问共享存储器(供所有 CPU 访问的存储器),通常采用____类型的联络方式实现通信。
3. (3.4.1.31)在____通信方式中,总线上所有模块都可以成为主模块。
4. 在做手术过程中,医生经常将手伸出,等护士将手术刀递上,待医生握紧后,护士才松手。如果把医生和护士看作是两个通信模块,上述一系列动作相当于 ____ 通信中的 ____ 方式

Solution

1. 不互锁
2. 半互锁
3. 分离式
4. 异步 全互锁
    1. 医生伸出手,发出请求
    2. 护士递上刀,响应请求
    3. 护士没递上刀医生不能放手,护士锁医生
    4. 医生没握紧护士不能放手,医生锁护士
    5. 因此为全互锁

Chapter 8: 输入输出系统

> **Memorize**

1. I/O 接口电路通常具有 [选址] 、 [传送命令] 、 [传送数据] 和 [反映设备状态] 功能
2. CPU 响应中断时要保护现场,包括对 [程序断点] 和 [寄存器内容] 的保护,前者通过[中断隐指令] 实现,后者可通过 [中断服务程序] 实现
3. 主机与 I/O 交换信息的三种控制方式分别为程序查询方式、程序中断方式、DMA 方式。
   - 程序查询方式:检查状态标记→准备就绪?→交换数据 (串行工作)
   - 程序中断方式:CPU-设备并行
   - DMA 方式:效率最高,CPU-设备并行,传送-主程序并行
4. 一次中断处理过程大致分为中断请求、中断判优、中断响应、中断服务、中断返回五个阶段。
5. 中断服务程序有保护现场、中断服务、恢复现场、中断返回四个阶段。

   > **Warning**
   >
   > 中断返回后的指令是下一条指令而不是当前指令。

6. 中断的响应优先级不可改变,处理优先级可通过屏蔽技术改变。
7. DMA 的工作过程有预处理、数据传送、后处理三个阶段
8. 在 DMA 方式中,CPU 和 DMA 控制器通常采用三种方法来分时使用主存,它们是[停止 CPU 访问主存] 、 [周期挪用] 和 [DMA 和 CPU 交替访问主存] 。
9. 周期窃取是指当 DMA 要求访存时,CPU 暂停一个或多个存储周期,将总线控制权让给 DMA,数据传送结束后,CPU 继续运行。CPU 现场没有变动,仅延缓了指令的执行。
   - 窃取一个个存储周期
10. 比较 DMA 方式和中断方式

-	中断方式	DMA 方式
数据传送	程序	硬件
响应时间	指令执行结束	存取周期结束
处理异常情况	能	不能
中断请求	传送数据	后处理
优先级	低	高

11. 中断请求触发器 INTR
    - 分散在各中断源的接口电路中
    - 集中在 CPU 中断系统内
    - 多个 INTR 组成中断请求寄存器
12. 中断服务程序的入口地址(中断向量地址)查询方法有软件查询法和硬件向量法
13. 实现多重中断的条件有
    - 提前设置开中断指令
    - 优先级高的中断源有权中断优先级低的

Tip

此部分除了中断屏蔽有较大的可能考大题以外,基本只会以记忆型题目出现。即仅要求掌握相关概念而不要求进行计算。复习时建议着重看选填和概念的简答题,不建议看 PPT 的计算相关例题。

Example

1. (5.4.3.17)解释周期挪用,分析周期挪用会出现几种情况
2. (5.4.3.24)试从五个方面比较中断方式和 DMA 方式的区别

Solution

1. 周期挪用是指 DMA 传送方式中
   - 当 I/O 设备没有 DMA 请求时,CPU 按程序的要求访问主存
   - I/O 设备有 DMA 请求并与 CPU 访存发生冲突时,CPU 要暂停一个存取周期访存,把总线控制权让给 DMA。
   - 这就好比 I/O 设备挪用了 CPU 的访存周期,故称周期挪用.

   设备提出 DMA 请求可能出现三种情况
   - CPU 不需要访存,无冲突,不需要进行周期挪用
   - CPU 正在访存,必须等待存取周期结束后才能进行 I/O 访存
   - CPU 和 DMA 同时提出访存请求,DMA 优先,CPU 延缓一个存取周期进行访存
2. DMA 方式和程序中断方式的区别
   - 从数据传送看,程序中断方式靠程序传送,DMA 式靠硬件传送
   - 从 CPU 响应时间看,程序中断方式在一条指令执行结束时响应,而 DMA 方式存取周期结束时 CPU 才能响应,即将总线控制权让给 DMA 传送
   - 程序中断方式有处理异常事件的能力,DMA 方式没有这种能力;
   - 程序中断方式需要中断现行程序,故需保护现场,DMA 方式不必中断现行程序,无需保护现场
   - DMA 的优先级比程序中断高。

Tip

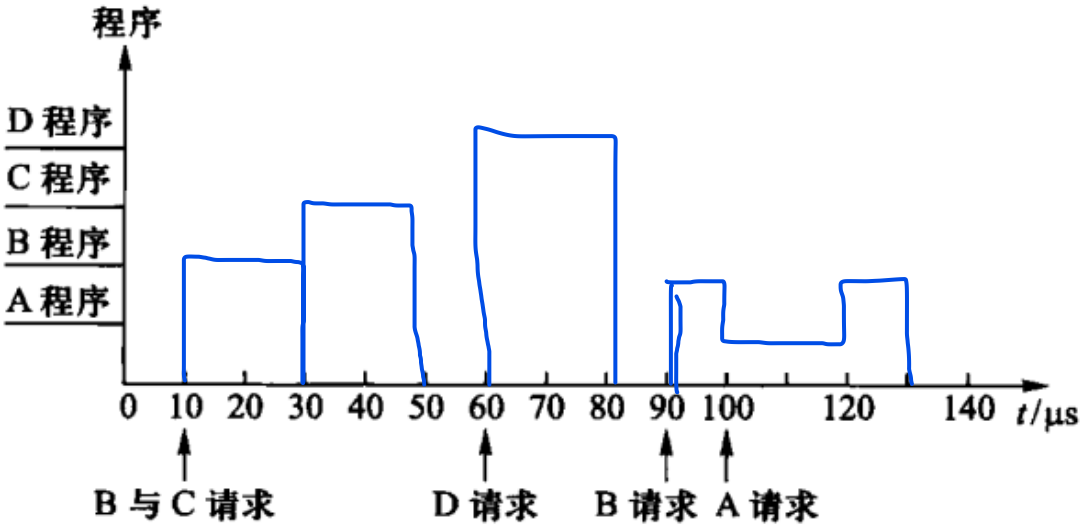
中断屏蔽的考察多要求给出处理一系列中断的次序,或以文字表述或画图表示,解题关注以下两点:

- 响应优先级:当多个中断同时发生时,处理哪一个中断
- 处理优先级:由中断屏蔽字表,判断当前发生的中断是否可中断当前正在处理的中断
  - 中断屏蔽字表中对应中断下若为 1,则表示可屏蔽该中断
  - 中断 A 必须能屏蔽其自身

更为顺畅的思考方式是由中断屏蔽字表判断哪个中断源可中断当前中断源服务程序

Example

1. 某机有 4 个中断源,优先顺序为 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$,若想将中断处理顺序改为 $3 \rightarrow 1 \rightarrow 4 \rightarrow 2$, 则 1、2、3、4 中断源对应的屏蔽字分别是 ____,____, ____, ____。
2. (diy) 现有 4 个中断源 A、B、C、D, 优先顺序为 $A > B > C > D$。中断服务程序为 $20\mu s$,请根据下图时间轴给出的中断源请求时刻,画出 CPU 执行程序的轨迹。



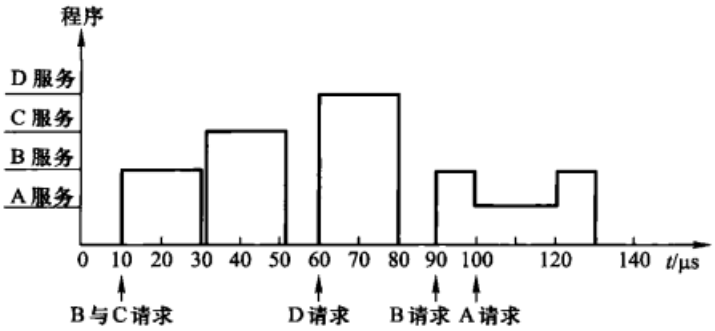
Solution

1. 思考方式应为“可以被谁中断”,而不是“屏蔽谁的中断”
   - 3 最优先,谁都不能中断他,故为 1111
   - 1 可以被 3 中断,故为 1101
   - 4 可以被 3 和 1 中断,故为 0101
   - 2 可以被 3,1,4 中断,故为 0100

   故答案为 1101,0100,1111,0101
2. 此类题目的解题大致可分为以下步骤
   - 多个中断同时发生,按优先顺序先跳转至对应中断处理程序
   - A 中断处理过程中若有新中断 B
     - 若 B 能中断 A,则跳至 B 处理程序后,B 处理完再 A
     - 若 B 不能中断 A,A 处理完后再处理 B

   本题中按如下分析:
   - B、C 同时请求,B 的响应优先级更高,先跳至 B 的处理程序
     - C 的处理优先级比 B 低,所以应该在 B 处理完后再处理 C
   - D 处理过程无事发生
   - B 处理过程出现 A 请求
     - A 请求的处理优先级比 B 高,故转至 A 程序
     - A 结束后,继续处理 B 程序

   如下图所示:



   按上述分析,很容易对给出中断屏蔽字的情况下进行变式处理。 例如
   - C 的处理优先级比 B 高,则 $10\mu s$处转至 B 后应立即转至 C

Warning

- 不能直接画到 C 服务,必须体现响应优先级

- A 的处理优先级比 B 的低,则后半段应在 B 处理完后再处理 A