

第一章 计算机系统概论

计算机系统由硬件系统和软件系统组成

硬件系统由运算器、控制器、存储器、输入设备、输出设备

冯诺依曼计算机的工作原理：

1.存储程序：将程序存放在计算机的存储器里

访问存储器的容量：容量 $2^n \rightarrow n$ 根地址线数量

按地址顺序存放

2.程序控制：按指令地址访问存储器并取出指令，经译码依次产生指令执行所需控制信号，实现对计算的控制，完成指令的功能。方式：顺序执行、跳跃执行

输入设备：将信息转换成机器能识别的形式

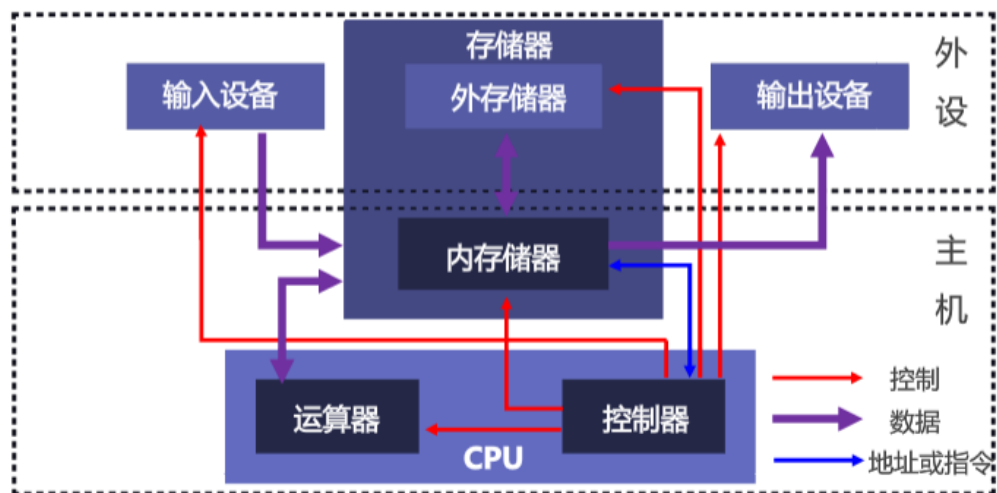
存储器：存放数据和程序

运算器：进行算术运算、逻辑运算

控制器：指挥程序运行

输出设备：将结果转换成人们熟悉的形式

1)硬件系统（总体图）



主机：CPU（运算器 + 控制器）、内存

外设：输入设备、输出设备、外存储器

总线：地址线、数据线、控制线

两种信息流：信息流、控制流

信息流：可双向可单向，分散流向各个部件

控制流：控制器到其他四个，单向，它受控制信息的控制，从一个部件流向另一个部件，在流动的过程

被相应的部件加工处理。

计算机工作原理：取指、执行
跳跃执行

计算机的层次结构：

微程序级：由机器硬件直接执行微指令。

一般机器（机器语言）级：是软件系统和硬件系统的界面，一条机器指令的功能由微程序机器级的一段微型程序的功能实现。

操作系统级：调度计算机中的软件和硬件资源。由操作系统程序实现，这些操作系统由机器指令和广义指令组成，这些广义指令是操作系统定义和解释的软件指令。

汇编语言级：给程序人员提供一种符号形式的语言，以降低程序编写的复杂性。将用户编写的接近人类语言的程序，翻译成能在机器上运行的目标程序

高级语言级：完全面向用户，为方便用户编写应用程序而设置的。由各种高级语言编译程序支持。

软硬件分界线是软硬件的接口，是指令操作硬件的接口。

基本性能指标：

非时间指标：

1.字长。计算机的字长一般指一次参与运算数据的基本长度，用二进制位数的长度来衡量。

2.主存容量：主存能储存的最大信息量

时间指标：

1.时钟周期：时钟周期是时间频率（主频）的倒数，是处理操作最基本的时间单位。

2.CPI：执行每条指令所需要的平均时钟周期数。 $CPI = \text{程序执行所需要的 CPU 时钟周期总数} / \text{程序所包含的指令条数}$

3.CPU 时间：执行某个任务是 CPU 实际消耗的时间。

$\text{CPU 时间} = \text{程序中所有指令的 CPU 时钟周期数之和} \times \text{CPU 的时钟周期}$

$\text{CPU 时间} = \text{程序中所有指令的 CPU 时钟周期数之和} / \text{CPU 时钟频率}$

$\text{CPU 时间} = \text{CPU 时钟周期} \times \text{CPI} \times \text{指令条数} = \text{CPI} \times \text{指令条数} / \text{时钟频率}$ （CPU 全性能公式）

4.MIPS：每秒百万条指令。

$\text{MIPS} = \text{程序中指令的条数} / (\text{程序 CPU 时间} \times 10^6) = \frac{\text{时钟频率}}{CPI \times 10^6}$

第二章 数据表示

数据信息的表示方法

计算机中表示地址使用无符号数

原码：最高位为 0 时表示为正数，为 1 时表示为负数。

1) 纯小数，设 $x = x_0.x_1x_2\cdots x_{n-1}$ ，其中 x_0 为符号位，共 n 位字长。

$$[x]_{\text{原}} = \begin{cases} x & , 0 \leq x \leq 1 - 2^{-(n-1)} \\ 1 - x & , -(1 - 2^{-(n-1)}) \leq x \leq 0 \end{cases}$$

2) 纯整数，设 $x = x_0x_1x_2\cdots x_{n-1}$ ，其中 x_0 为符号位，共 n 位字长。

$$[x]_{\text{原}} = \begin{cases} x & , 0 \leq x \leq 2^{n-1} - 1 \\ 2^{n-1} - x & , -(2^{n-1} - 1) \leq x \leq 0 \end{cases}$$

$$[+0]_{\text{原}} = 00000000, [-0]_{\text{原}} = 10000000$$

减负数相当于加正数，负数的原码就是在最高位前加个 1

补码：除符号位取反加一

1) 纯小数时

$$[x]_{\text{补}} = \begin{cases} x & , 0 \leq x \leq 1 - 2^{-(n-1)} \\ 2 + x & , -1 \leq x \leq 0 \end{cases}$$

2) 纯整数时

$$[x]_{\text{补}} = \begin{cases} x & , 0 \leq x \leq 2^{n-1} - 1 \\ 2^n + x & , -2^{n-1} \leq x \leq 0 \end{cases}$$

$$[0]_{\text{补}} = 00000000$$

补码的补码是原码

$$\text{小数-1: } [-1]_{\text{补}} = 1.0000000$$

$$\text{整数-1: } [-1]_{\text{补}} = 11111111$$

反码：除符号位取反

1) 纯小数时

$$[x]_{\text{补}} = \begin{cases} x & , 0 \leq x \leq 1 - 2^{-(n-1)} \\ 2 - 2^{-(n-1)} + x & , -1 \leq x \leq 0 \end{cases}$$

2) 纯整数时

$$[x]_{\text{补}} = \begin{cases} x & , 0 \leq x \leq 2^{n-1} - 1 \\ 2^n - 1 + x & , -2^{n-1} \leq x \leq 0 \end{cases}$$

移码：补码的符号位取反

$$[x]_{\text{移}} = 2^{n-1} + [x]_{\text{补}}$$

定点小数:

原码范围:

负数:

$$1.1111111 \sim 1.0000001, -(1 - 2^{-7}) \sim -2^{-7}$$

正数:

$$0.0000001 \sim 0.1111111, 2^{-7} \sim 1 - 2^{-7}$$

补码范围:

负数:

$$1.0000000 \sim 1.1111111, -1 \sim -2^{-7}$$

正数:

$$0.0000001 \sim 0.1111111, 2^{-7} \sim 1 - 2^{-7}$$

定点整数:

原码范围:

负数:

$$11111111 \sim 1000001, -127 \sim -1$$

正数:

$$00000001 \sim 01111111, 1 \sim 127$$

补码范围:

负数:

$$10000000 \sim 11111111, -128 \sim -1$$

正数:

$$00000001 \sim 01111111, 1 \sim 127$$

因为补码 0 唯一, 所以补码可以表示 2^n 个数, 而原码只能 $2^n - 1$ 个数。

数的浮点表示:

浮点表示法把字长分为阶码 (表示指数) 和尾数 (表示数值) 两部分

阶码 = 阶符 + 阶码值

尾数 = 数符 + 尾数值

设阶码为 E, 尾数为 D, 则 $X = D \times 2^E$ 。

阶码采用补码或移码定点整数形式、尾数通常用补码定点小数形式。

浮点表示法两种格式:

1. 阶符 + 阶码值 + 数符 + 尾数值

2. 数符 + 阶符 + 阶码值 + 尾数值

提高浮点数精度: 1. 增加尾数尾数 2. 浮点数规格化

浮点数的规格化：

通过调整阶码，使尾数满足以下形式：

1.原码规格化：

正数为 $0.1 \times \dots \times$ 的形式，负数为 $1.1 \times \dots \times$ 的形式。

2.补码规格化：

正数为 $0.1 \times \dots \times$ 的形式，负数为 $1.0 \times \dots \times$ 的形式。

定点数溢出判断是对数值本身进行判断，浮点数是对规格化后的阶码进行判断。但一个浮点数阶码大于机器的最大阶码，称为上溢；小于最小阶码时，称为下溢。机器产生上溢时，不能再继续运算，一般要进行中断处理；出现下溢时，一般规定把浮点数各位强迫为零，机器仍可继续计算。

IEEE754：符号位 S + 指数部分 E + 尾数部分 M

IEEE754 阶码为移码，尾数为原码

$$N = (-1)^S \times 2^{E-127} \times 1.M$$

单精度格式（32位）：符号位 1 位，指数部分 8 位，尾数部分 23 位。

约定小数点左边隐藏一位 1。

阶码部分采用移码表示，即加上 127。

阶码 255，尾数为 0：无穷大

阶码 255，尾数不为 0：非数值 NaN

阶码为 0，尾数为 0：0

阶码为 0，尾数不为 0：非规格化数（0 为隐含位）

正负 0，非规格化数的隐含位为 0，不是 1。

汉字编码：

汉字输入码（外码）：是为了将汉字输入计算机而编制的代码，是代表某一汉字的一串键盘符号。

区位码：4 位 10 进制，94 * 94 矩阵，国家标准局公布的 6763 个两级汉字分为 94 个区，每个区分 94 位，实际上把汉字表示成二维数组，每个汉字在数组中的下标就是区位码。

国标码：区位码加 2020H

GB2312 汉字机内码 = 区位码 + 0xA0A0

机内码：计算机内存存储字符时使用的编码

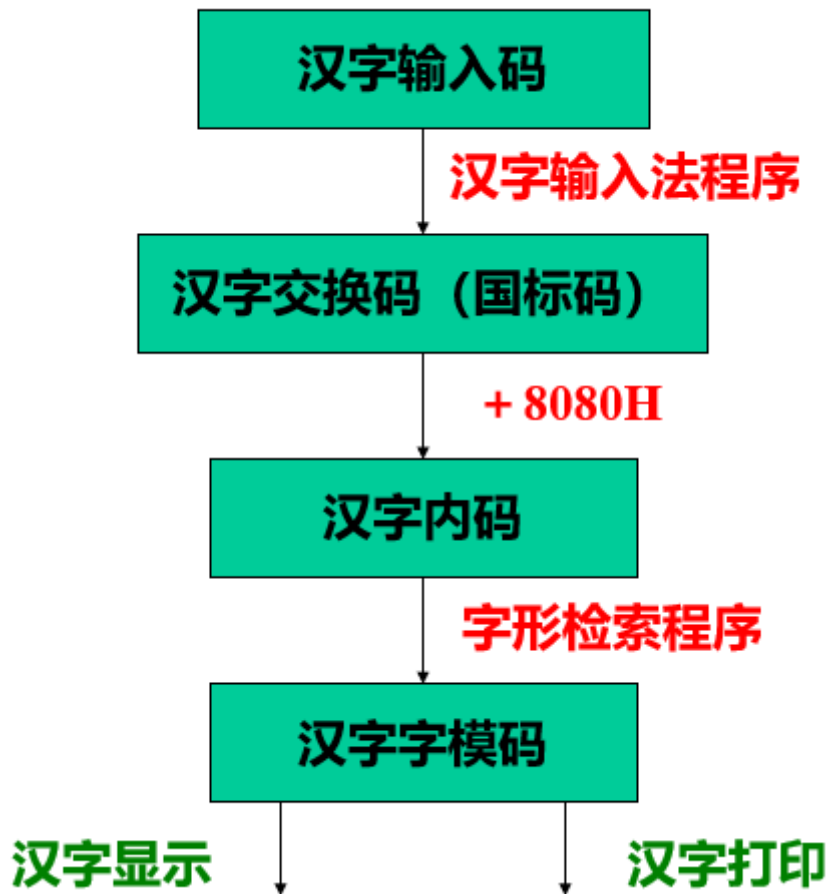
英文字符：ASCII 单字节 有效位 7 位，MSB（最高有效位）为 0

汉字字符：GB2312 双字节 有效位 14 位 MSB（最高有效位）为 1

汉字字形（模）码：将汉字字形经过点阵数字化后形成的一串二进制数，用于汉字的显示和打印。

点阵信息量大，所占存储空间大，用来构成汉字字库，不用于机内存储

具体过程：



数据校验的基本原理

数据校验：

有效信息 k 位 + 校验信息 r 位

码距（海明距离）：同一编码中，任意两个合法编码之间不同二进制位数的最小值（或者异或后数 1 的数目）。

增加冗余项的目的是为了增大码距。

码距与检错或纠错能力的关系：（ e 为检错的数目， t 为纠错的数目）

码距 $\geq e + 1$ ，可检测 e 个错误

码距 $\geq 2t + 1$ 可纠正 t 个数目

码距 $\geq e + t + 1$ 可纠正 t 个数目，同时检测 e 个错误（ $e \geq t$ ）

码距越大，抗干扰能力越强，纠错能力越强，数据冗余越大，编码效率越低，编码电路也越复杂。

奇偶校验：

有效信息（ k 位）+ 校验信息（ $r = 1$ 位）

偶校验：使 1 的个数为偶数个

奇校验：使 1 的个数为奇数个

奇校验检错码：， $G = \overline{C \oplus x_1 \oplus x_2 \oplus x_3 \oplus \cdots \oplus x_n}$ ，0 表示正常，否则出错

偶校验检错码：， $G = C \oplus x_1 \oplus x_2 \oplus x_3 \oplus \cdots \oplus x_n$ ，0 表示正常，否则出错

奇偶校验码距为 2

奇偶校验不能检测偶数位错误，无错结论不可靠，无纠错能力

CRC 校验：

有效信息（k 位）+ 校验信息（r 位）， $N = k + r \leq 2^r - 1$

选择生成多项式生成冗余码

利用生成多项式对收到的编码进行模 2 除法运算，根据余数判断出错位置

CRC 校验的检错与纠错：

编码不同数位出错对应不同的余数（1 位出错）

CRC 校验无错结论不可信

循环特性：若余数不为 0，对余数补 0 继续做模 2 除，同时将被检测的校验码循环左移，当余数为 101 时，出错位移到了 A1 位置，通过异或运算修改，再循环左移和进行模 2 除法，将出错位转回原位，这样就不需要为每一位提供纠错电路。

海明校验：

有效信息（k 位）+ 校验信息（r 位）， $N = k + r \leq 2^r - 1$

第 i 位校验码 P_i 在第 2^{i-1} 位上。

第 H_j 位的数据被编号小于 j 的若干个海明位号之和等于 j 的校验位所校验。

由此可以采用偶校验计算出校验位的值。

当 N = 11 时：

$$P_1 = b_1 \oplus b_2 \oplus b_4 \oplus b_5 \oplus b_7$$

$$P_2 = b_1 \oplus b_3 \oplus b_4 \oplus b_6 \oplus b_7$$

$$P_3 = b_2 \oplus b_3 \oplus b_4$$

$$P_4 = b_5 \oplus b_6 \oplus b_7$$

指错字：

$$G_1 = P_1 \oplus b_1 \oplus b_2 \oplus b_4 \oplus b_5 \oplus b_7$$

$$G_2 = P_2 \oplus b_1 \oplus b_3 \oplus b_4 \oplus b_6 \oplus b_7$$

$$G_3 = P_3 \oplus b_2 \oplus b_3 \oplus b_4$$

$$G_4 = P_4 \oplus b_5 \oplus b_6 \oplus b_7$$

$G_4 G_3 G_2 G_1$ 为 0 则表示无错误，否则表示出错的位数

海明校验无错结论不可信，也无法区别是 1 位错还是 2 位错

第三章 运算方法与运算器

ALU 能完成算术与逻辑运算

运算器由数据总线、ALU、状态寄存器组成

定点数加减法

$$[x]_{\text{补}} + [y]_{\text{补}} = [x + y]_{\text{补}}$$

$$[x]_{\text{补}} - [y]_{\text{补}} = [x - y]_{\text{补}} = [x + (-y)]_{\text{补}} = [x]_{\text{补}} + [-y]_{\text{补}}$$

补码运算规则：

1. 参与运算的数都用补码
2. 数据的符号和数据一样都参与运算
3. 求差时将减数求补，用求和代替求差
4. 运算结果为补码，符号位为 0，正数；符号位为 1，负数
5. 符号位的进位若为模值，则应丢掉

溢出判断：

两个符号相同的数相加，其运算结果的符号应与被加数符号相同，两个符号相异的数相减，其运算结果应与被减数符号相同。

判断方法：

1. 双符号位法：

00 表示正号

11 表示负号

01 表示正向溢出

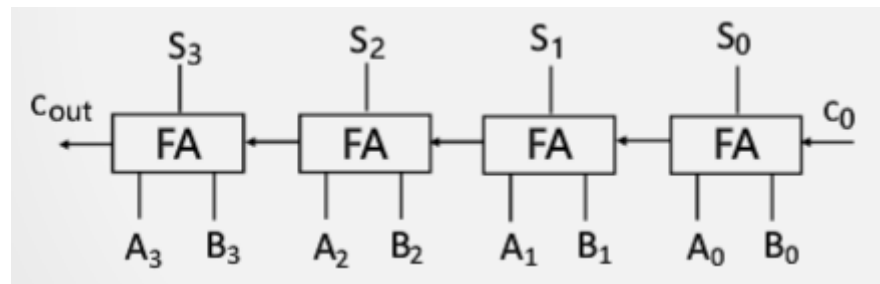
10 表示负向溢出

两位取异或为 1 时有溢出，为 0 时无。

2. 进位判断法：

当两个单符号位的补码进行加减运算时，若最高数值位向符号位的进位值 C 与符号位产生的进位输出值 S 相同，则没有溢出，反之则有。即 $V = S \oplus C$

串行加法器：每一个全加器接受当前位与上一位的进位，输出结果，并向下一位提供进位



加/减法器：若 y 为负数，取其异或值

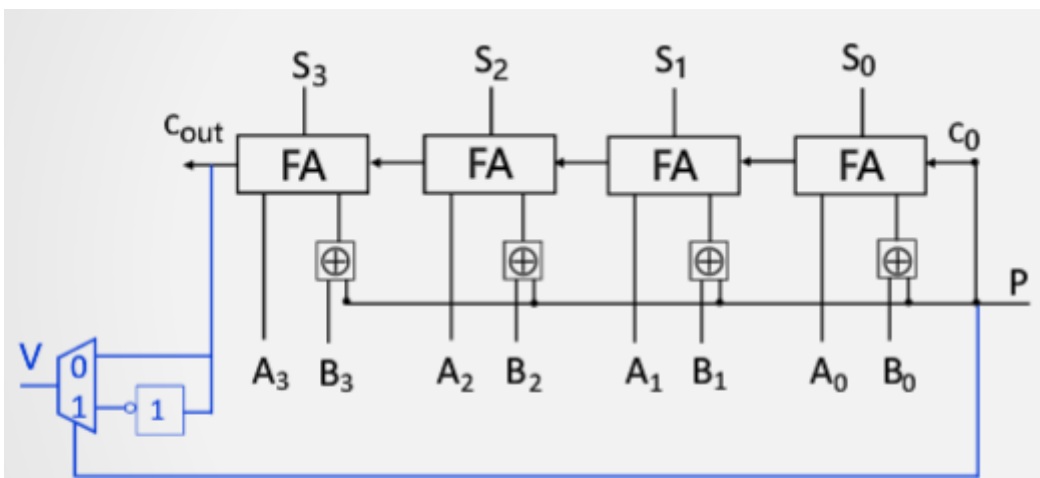
[illegible]

The circuit diagram illustrates a 4-bit ripple carry adder. It consists of four Full Adders (FA) labeled FA₀, FA₁, FA₂, and FA₃. The inputs are A₀, B₀, A₁, B₁, A₂, B₂, A₃, and B₃. The outputs are S₀, S₁, S₂, S₃, C₀, and C_{out}. The carry propagation signal P is generated from the first two bits. The carry lookahead logic uses AND gates and OR gates to generate the carry signals. The final output V is the sum of the carry-out and the product of the carry-in and the propagate signal.

$$V = X_0 Y_0 \bar{S}_0 + \bar{X}_0 \bar{Y}_0 S_0$$

$V = C_0 \oplus C_1$

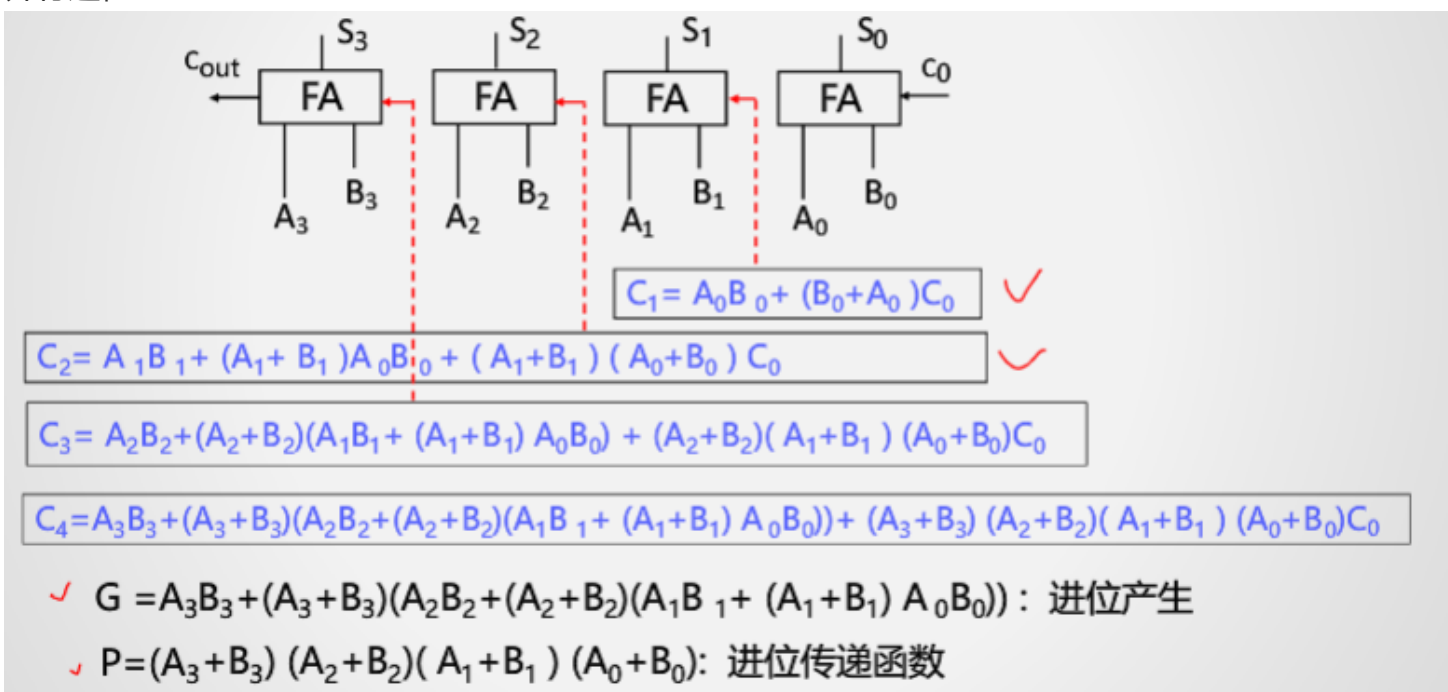
无符号:



P=1 选择无符号数减法溢出 (借位)

P=0时, 选择无符号加法溢出 (进位)

并行进位:



定点乘法运算

逻辑左移: 数据为整体左移一位, 最高位被移出至进位位, 最低位补 0

算数左移: 数据为整体左移一位, 最高位被移出至进位位, 最低位补 0

逻辑右移: 数据位整体右移一位, 最高位补 0, 最低位被移出

算术右移: 数据位整体右移一位, 最高位复制原最高位, 最低位被移出

原码一位乘法运算规则: $x = x_0 x_1 x_2 \cdots x_n$

(1) 被乘数和乘数均去绝对值参与运算, 符号位单独考虑

(2) 被乘数取**双符号**，部分积的长度与被乘数的长度相同，初值为 0

(3) 从乘数的最低位的 y_n 位开始对乘数进行判断：若 $y_n = 1$ ，则部分积加上被乘数 $|x|$ ，然后右移一位；若 $y_n = 0$ ，则部分积加上 0，然后右移一位。

(4) 重复 (3) 的判断 n 次。

需要多输入的全加器，需要长 $2n$ 的积寄存器

例：

0.010
× 0.101
—
0010
→ 0010
+ 0000
—
00010
→ 00010
+ 0010
—
001010
→ 001010
+ 0000
—
0001010

例1 已知 $X = 0.110$ $Y = -0.101$ 用原码一位乘法求 $X \cdot Y$
解: $[X]_{\text{原}} = 0.110$ $[Y]_{\text{原}} = 1.101$
部分积
00.000
+ 00.110
—
00.110
→ 00.011
+ 00.000
—
00.011
→ 00.001
+ 00.110
—
00.111
→ 00.011
 $X \cdot Y = (0 \oplus 1).011110 = 1.011110$

说明
 $Y_3 = 1$ 部分积 + $|X|$
每次运算结果右移1位
 $Y_3 = 0$ 部分积 + 0
 $Y_3 = 1$ 部分积 + $|X|$
 Y_0
 Y_0
 Y_0

补码一位乘法的运算规则： $x = x_0x_1x_2 \cdots x_n$

$$[x \cdot y]_{\text{补}} = [x]_{\text{补}} \cdot \sum_{i=0}^n (y_{i+1} - y_i) 2^{-i}$$

(1) **符号位参与运算**，运算的数均以补码表示

(2) 被乘数一般取**双符号位**参与运算，部分积初值为 0

(3) 乘数取可取单符号位，以决定最后一步是否需要校正，即是否需要加 $[-x]_{\text{补}}$

(4) 乘数末尾增设附加位 y_{n+1} ，且初值设为 0

(5) 如果 $y_{n+1} = y_n$ ，部分积加 0，部分积算术右移 1 位；如果 $y_{n+1}y_n = 10$ ，部分积加 $[x]_{\text{补}}$ ，部分积算术右移 1 位；如果 $y_{n+1}y_n = 01$ ，部分积加 $[-x]_{\text{补}}$ ，部分积算术右移 1 位；

(6) 重复 $n+1$ 次，第 $n+1$ 次不移位，仅根据 y_0 与 y_1 的比较结果做相应运算即可

其中 y_{n+1} 是在乘数寄存器后增加一位

结果需要再取补码获得原码

例：

例1 已知 $X = +1101$ $Y = +1011$ 用补码一位乘法求 $X \times Y$

解： $[X]_{\text{补}} = 01101$ $[Y]_{\text{补}} = 01011$ $[-X]_{\text{补}} = 10011$

	部分积	乘数	说明
	000000	<u>010110</u>	$Y_{n+1} < Y_n$ 部分积 $+ [-X]_{\text{补}}$
+	<u>110011</u>		
	110011		
→	111001	<u>101011</u>	结果右移一位, $Y_{n+1} = Y_n$ 部分积 $+ 0$
+	<u>000000</u>		
	111001		
→	111100	<u>110101</u>	结果右移一位, $Y_{n+1} > Y_n$ 部分积 $+ [X]_{\text{补}}$
+	<u>001101</u>		
	001001		

部分积	乘数	说明
→ 000100	<u>111010</u>	将结果右移一位, $Y_{n+1} < Y_n$ 部分积 $+ [-X]_{\text{补}}$
+	<u>110011</u>	
	110111	
→ 111011	<u>111101</u>	将结果右移一位, $Y_{n+1} > Y_n$ 部分积 $+ [X]_{\text{补}}$
+	<u>001101</u>	
	001000	

$\therefore [X \cdot Y]_{\text{补}} = 010001111$

$\therefore X \cdot Y = 010001111$

定点数除法运算

除法可以用减法实现，需要长度为 $2n$ 的余数寄存器

原码恢复余数法：（使用**双符号位**）

商的符号通过除数与被除数的符号异或求得，符号位不参与运算

利用减法，通过余数符号位判断够不够减。

1. 余数为正数，商上 1，将余数算术左移一位，继续操作

2.余数为负数，商上 0，加上除数，将余数算术左移一位，继续操作。
重复此操作知道商达到所需要的位数为止。

原码不恢复原数法：

- (1) 符号位**不参与运算**，、
- (2) 先用被除数减去除数，余数为正时，商上 1，余数左移一位，再减去除数，进行比较；余数为负时，余数减取一位，再加上除数，进行比较
- (3) 当第 $n + 1$ 步余数为负时，需要加上 $|y|$ 得到第 $n + 1$ 步正确的余数

例：

已知 $X = 0.1001$, $Y = - 0.1011$, 用原码一位除法求 X/Y

被除数/余数	商	上商位	说明
00.1001			减Y比较
$+[-Y]_{补}$ 11.0101			
0 11.1110		0	余数 < 0 商上零
11.1110	0		左移一位
$+ [Y]_{补}$ 00.1011			加Y比较
1 00.0111		1	余数 > 0, 商上1
00.1110	0.1		左移一位
$+ [-Y]_{补}$ 11.0101			减Y比较
1 00.0011		1	余数 > 0, 商上1
00.0110	0.11		左移一位
$+ [-Y]_{补}$ 11.0101			减Y比较
0 11.1011		0	余数 < 0 商上零
11.0110	0.110		左移一位
$+ [Y]_{补}$ 00.1011			加Y比较
1 00.0001	0.1101	1	余数 > 0, 商上1, 移商

最后结果：

商 $Q = X_0 \oplus Y_0.1101 = 1.1101$

余数 $R = 0.0001 * 2^{-4}$

被除数/余数	商	上商位	说明
00.1001			减Y比较
$+ [-Y]_{补}$ 11.0101			
11.1110		0	余数 < 0, 商上0
00.1011			加Y恢复余数
00.1001			左移一位
01.0010	0		减Y比较
$+ [-Y]_{补}$ 11.0101			
00.0111		1	余数 > 0, 商上1
00.1110	0.1		左移一位
$+ [Y]_{补}$ 11.0101			减Y比较
00.0011		1	余数 > 0, 商上1
00.0110	0.11		左移一位
$+ [-Y]_{补}$ 11.0101			减Y比较
11.1011		0	余数 < 0, 商上0
00.1011			加Y恢复余数
00.0110			左移一位
00.1100	0.110		减Y比较
$+ [-Y]_{补}$ 11.0101			
00.0001	0.1101	1	余数 > 0, 商上1, 移商

浮点数加减运算

浮点数加减运算方法及步骤：

- (1) 对阶：求阶差，右移阶码小的浮点数的尾数并同步增加其阶码，使两阶码相等
- (2) 尾数加/减
- (3) 规格化结果
- (4) 右移规格化时舍入：
0 舍 1 入：若右移出的是 1，则在最低位加 1
恒置 1：只要数字位 1 被移掉，就将最后 1 位恒置为 1.
- (5) 溢出处理：阶码为 01，上溢；阶码为 10，下溢。

例：

例 设 $x = 2^{010} \times 0.11011011$ $y = 2^{100} \times (-0.10101100)$ 求 $x+y$

解：先用补码形式表示 x 和 y

$$[X]_{\text{补}} = 00\ 010 \quad , \quad 00.11011011$$

$$[Y]_{\text{补}} = 00\ 100 \quad , \quad 11.01010100$$

(1) 对阶

$$[\Delta E]_{\text{补}} = [Ex]_{\text{补}} + [-Ey]_{\text{补}} = 00010 + 11100 = 11\ 110$$

$\therefore \Delta E = -2$ x 的阶码 小于 y 的阶码

将 x 的尾数向右移动2位，同时阶码加 2，对阶后的 x 为：

$$[X]_{\text{补}} = 00\ 100 \quad , \quad 00.0011011011$$

2) 尾数运算

$$\begin{array}{r} 00.00110110\ 11 \\ +\ 11.01010100 \\ \hline 11.1000101011 \end{array}$$

3) 尾数规格化处理

分析发现，只左移一次即可达到规格化要求。规格化后的结果为：

$$[X+Y]_{\text{补}} = 00\ 011 \quad , \quad 11.000101011$$

4) 舍入（0舍1入）

在结果尾数的最低位加1，最后的结果为：

$$[X+Y]_{\text{补}} = 00\ 011 \quad , \quad 11.00010110 \quad \quad X+Y = -0.11101010 \times 2^{011}$$

第四章 存储系统

存储系统层次结构

基本存储体系原理：

存储程序

- 1) 输入设备将程序与数据写入主存
- 2) CPU 取指令
- 3) CPU 之心指令期间读数据

4) CPU 写回运算结果

5) 输出设备输出结果

程序控制

主存速度慢的原因：

主存增速与 CPU 增速不同步

指令执行期间多次访问存储器

Cache 解决 CPU 与主存速度不匹配的矛盾

辅助存储器：解决主存容量不足与高成本的矛盾

存储体系的层次化结构：

Cache、主存、辅助存储器

CPU 访问到的存储系统具有 Cache 的速度吗，辅存的容量的价格

哈佛结构：指令存储与数据存储在分开的存储器结构

局部性原理：

时间局部性：现在被访问的信息在不久的将来还会被再次访问（循环结构）

空间局部性：现访问一个信息，下次访问附近的信息（顺序结构）

主存中的数据组织

存储字长：主存的一个存储单元所包含的二进制位数

数据存储与边界的关系：

1.按边界对齐

2.未按边界对齐（节省空间，增加了访问次数）

双字长数据边界的起始地址最末三位是 000

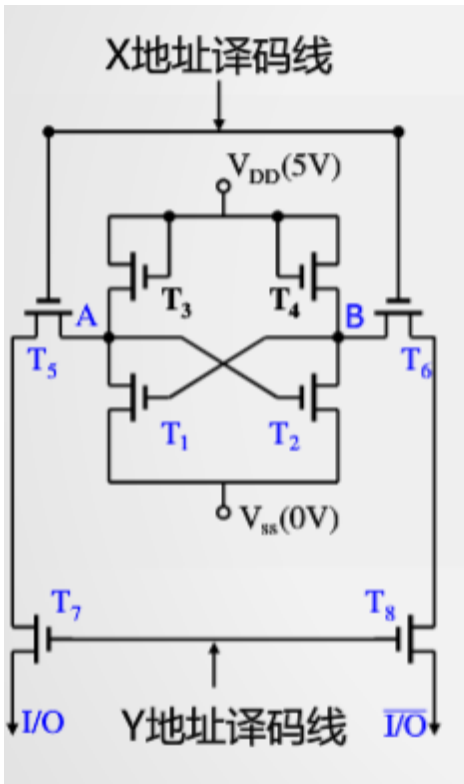
单字长边界对齐的起始地址的末二位为 00

半字长边界对齐的起始地址的最末一位为 0

大端存储方式：最高字节地址是数据地址

小端存储方式：最低字节地址是数据地址

静态存储器工作原理



- 工作管： T_1 、 T_2 (保存数据)
- 负载管： T_3 、 T_4 (补充电荷)
- 门控管： T_5 、 T_6 、 T_7 、 T_8 (开关作用)

X 地址选通 (行选通)： T_5 、 T_6 管导通，A 点与位线相连

Y 地址选通 (列选通)： T_7 、 T_8 管导通，A 点电位输出到 I/O 端

无论读/写，都要求 X 和 Y 译码线同时有效

静态 ram 工作原理：

静态 ram 是由两个 MOS 反相器交叉耦合而成的触发器。

写操作： T_1 管导通， T_2 管截止表示写 0； T_1 管截止， T_2 管导通表示写 1。

读操作：通过外接于 I/O 与 $\overline{I/O}$ 间的电流放大器中的电流方向可判断读出的是 1 还是 0。

保持：负载管 T_3 、 T_4 分别为工作管 T_1 、 T_2 提供工作电流，保持其稳定互锁状态不变。

地址线： n 条 $\rightarrow 2^n$ 字节存储空间

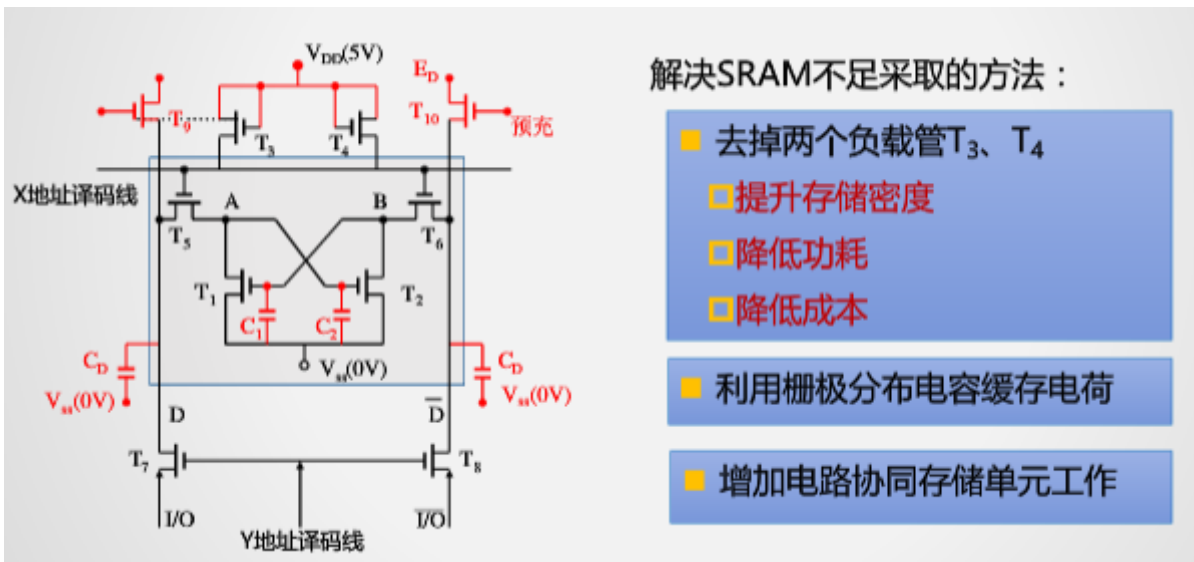
数据线： n 条 \rightarrow 每个单元 n 位数据

2114 SRAM：10 根地址线，1K 存储空间，每个单元 4 位数据， \overline{WE} 为低电平时写操作，反之为读操作。片选线低电平有效

单译码结构： n 位地址，寻址 2^n 个存储单元， 2^n 根译码线

双译码结构： n 位地址，寻址 2^n 个存储单元， $2^{n/2+1}$ 根译码线。

动态存储器工作原理



X 地址选通（行选通）： T_5, T_6 管导通，位线与 C_1, C_2 相连

Y 地址选通（列选通）： T_7, T_8 管导通，

I/O 端数据写入到位线

无论读/写，都要求 X 和 Y 译码线同时有效

动态 ram:

动态 ram 利用栅极分布电容缓存电荷。

写操作： T_1 管导通， T_2 管截止电容 C_1 充电，表示写 0； T_1 管截止， T_2 管导通电容 C_2 充电，表示写 1。

读操作：给出预充信号， T_9, T_{10} 导通给 C_d 充电，撤除预充信号后右 C_d 给 C_1 充电，左 C_d 给 C_2 充电，左右 C_d 间形成放电电流来进行读写操作

保持：栅极自行维持，但可持续时间很短

刷新：给出预充信号， T_9, T_{10} 导通给 C_d 充电，撤除预充信号后右 C_d 给 C_1 充电，左 C_d 给 C_2 充电。

刷新周期：两次刷新之间的时间间隔

双译码结构的 DRAM 刷新按行进行，需要知道 DEAM 芯片存储矩阵的行数。

刷新地址由刷行地址计数器给出。

刷新方式：

1.集中刷新：

假定刷新周期为2ms, DRAM 内部128行, 读写周期0.5 μ s

集中刷新



采用集中刷新的存储器平均读写周期

$$\overline{T} = 2\text{ms} / (4000 - 128) = 0.5165\mu\text{s}$$

刷新期间 CPU 不可访问存储器, 存在死时间

2.分散刷新:

假定刷新周期为2ms, DRAM 内部128行, 读写周期0.5 μ s

分散刷新



$$\overline{T} = 1\mu\text{s}$$

刷新过多对存储器性能影响较大

3.异步刷新:

假定刷新周期为2ms, DRAM 内部128行, 读写周期0.5 μ s

分散刷新



$$\overline{T} = 1\mu\text{s}$$

2116 DRAM: 地址线复用, n 条地址线 $\rightarrow 2^{2n}$ 字节存储单元

DRAM 比相同工艺的 SRAM 要慢, 原因:

1.DRAM 需要刷新

2.DRAM 读写过程中对其地址分行、列分时传送, 读操作要事先进行预充操作

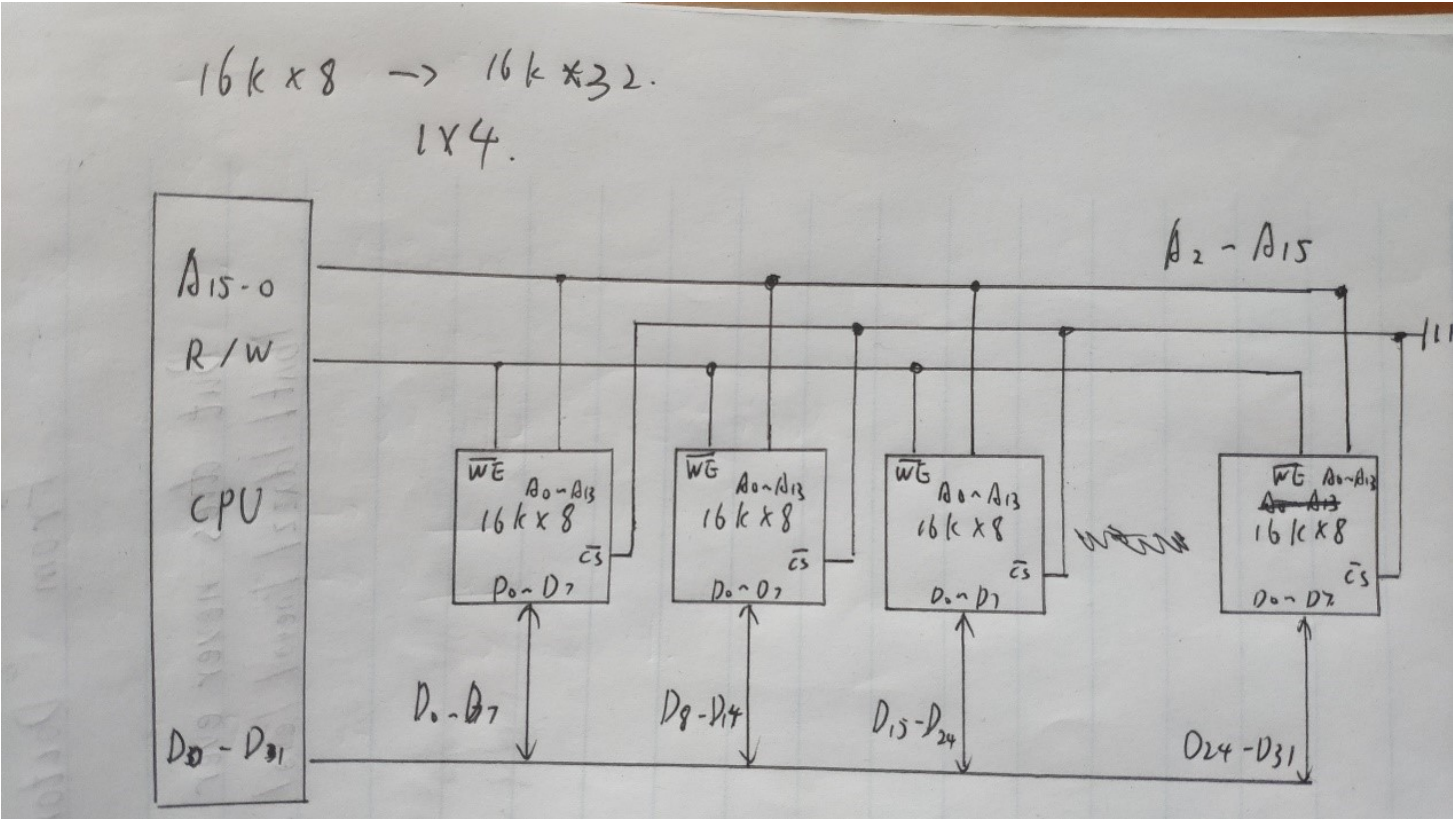
SRAM 与 DRAM 比较：

比较内容	SRAM	DRAM
存储方式	双稳态触发器	栅极分布电容
电源不掉电时	信息稳定	信息丢失
刷新	不需要	需要
集成度	低	高
容量	小	大
价格	高	低
速度	快	慢
适用场合	Cache	主存

存储扩展

- 位扩展：所有芯片并行工作，贡献位
- 字扩展：多余地址线作为片选择码输入
- 字位扩展

例：用 16K * 8 的存储芯片构造 16K * 32 的存储器



例：用 $16K \times 8$ 的存储芯片构建 $128K \times 8$ 的存储器，其中 $08000H \sim 0BFFFH$ 存储空间保留不用

$16k \times 8 \rightarrow 128k \times 8$.

8×1 .

$128k \rightarrow 17$ 根地址线

$16k \rightarrow 14$ 根地址线.

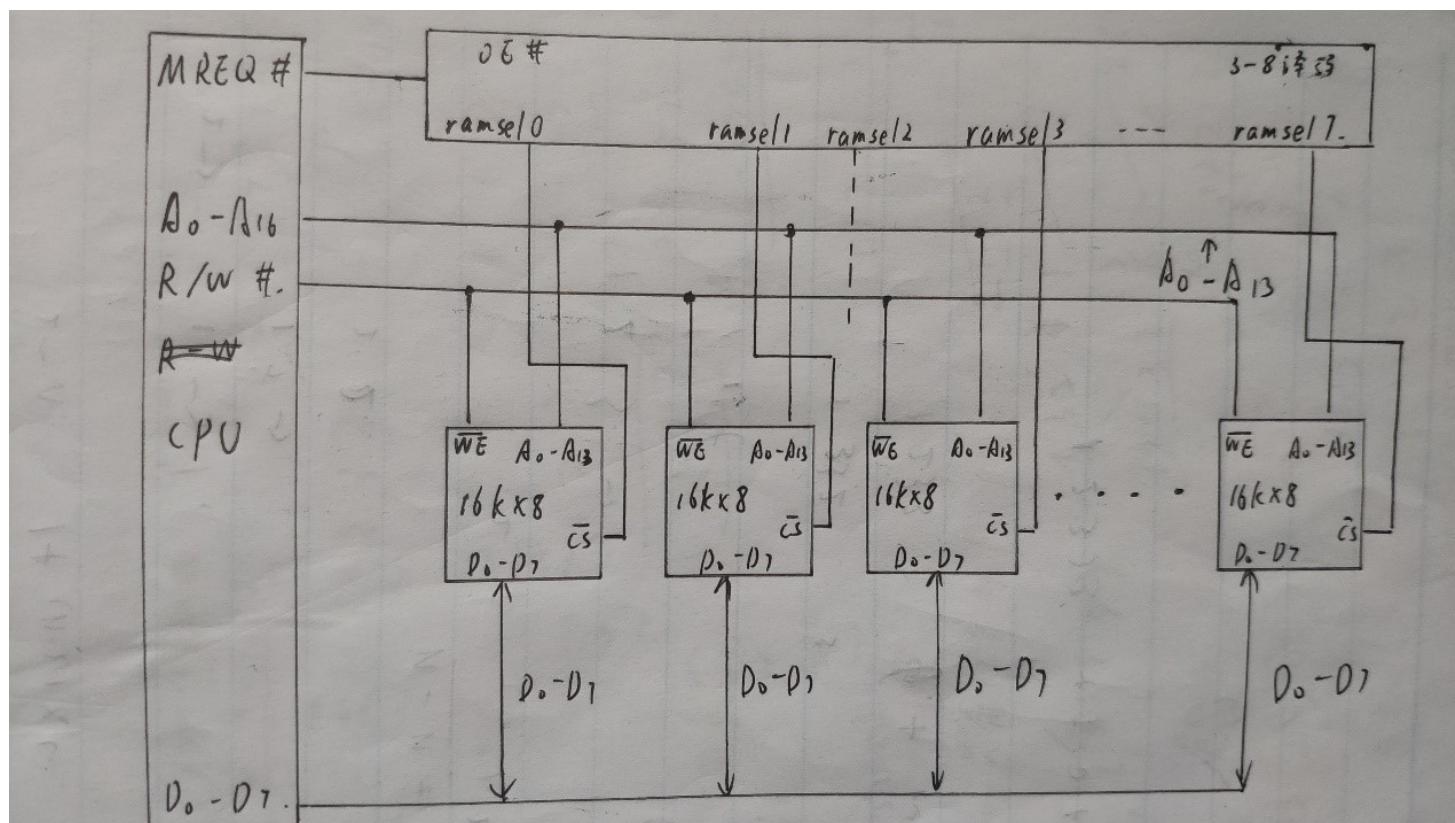
多余 3 根用于作为片选译码输入.

$08000H \rightarrow 010\ 00\ 0000\ 0000\ 0000$

$0BFFFH \rightarrow 010\ 11\ 1111\ 1111\ 1111$

第 2 个片选不用.

共 7×1 片.



RAM (random access memory) 即随机存储内存，这种存储器在断电时将丢失其存储内容，故主要用于存储短时间使用的程序。

ROM (Read-Only Memory) 即只读内存, 是一种只能读出事先所存数据的固态半导体存储器。RAM、ROM都采用随机存取的方式进行访问, RAM 是易失性存储器, ROM 是非失性存储器。