第六章　微处理器

王晓华

Wxhsnow@163.com

简介

- 微处理器**MPU**是典型的**LSI**、**VLSI**器件
- 其设计中体现了多种**VLSI**设计技术的应用与模块化的结构
- 微处理器模块的设计实质上是数字逻辑模块的设计
- 本章主要讨论如何用**VLSI**设计技术与模块结构实现常规微处理器内核（**Core**）的逻辑模块

wxhsnow@163.com

目录

wxhsnow@163.com

6.1 系统结构概述

- 早期人们使用通用型微处理器，通过软件编程和外围电路的设计实现所需要的功能
- 将功能模块加入微处理器结构，设计内置**ROM**驻留用户程序＝＝》专用微处理器（单片机）
- 专用微处理器的核心部分与通用微处理器相似，同时具有一个功能模块，不必在外围加转换电路，本身就是一个完整的信息处理系统，实现了系统的小型化和微型化

wxhsnow@163.com

传统微处理器设计

- 传统微处理器设计
  - 以8位，16位，32位二进制数为一个字
  - 内部指令的传送、处理都以字来进行
  - 外部数据的输入输出也以字的格式进行
  - 总线机构也是以8的偶数被设定宽度
  - 由于封装技术影响散热，只能以比较少的位数构造微处理器－－>位片微处理器
  - 几个位片组合形成完整字长
  - 1位片，4位片较多用在家电控制领域

微处理器的内核

- 通用微处理器，单片微处理器，位片微处理器的基本内核的构成相近
- 微处理器的内核：进行数字信号处理与运算的逻辑结构
- 数字信号分为数据流与控制流
  - 数据流：待处理和运算的数字信息
  - 控制流：控制数据做何种操作或运算的命令（一组二进制代码）
- 程序的主体是控制流

微处理器处理信号的特点

- 微处理器内核中有一个可变逻辑操作和运算功能的模块
- 通过程序可进行多种逻辑操作和算术运算
  - 普通数字逻辑电路只能完成规定的（不可变的）函数运算
  - 程序必须经过翻译才能被用于控制微处理器各部分

微处理器的主要 组成部分

- 将设计者的规定（程序、中断）“翻译”成具体控制信号的模块
- 执行各种逻辑操作和算术运算的模块
- 传输数据信号和控制信号的总线
- 通常微处理器有两个空间和通信连线组成
  - ✓ 程序空间（控制通道）
  - ✓ 数据空间（数据通道）
  - ✓ 通信连线：总线
- 分离式结构——哈佛结构
- 微处理器采用模块化结构，在每一个空间中由若干的模块分别完成不同的任务

程序空间

- 程序空间主要包括：

➢控制器（**Controller**）：

将用户的需求，与程序，中断，读写控制等翻译成内部的控制信号去“规定”各部分电路的“行为”

➢程序计数器（**PC**）：提取新命令计算地址

➢堆栈（**Stack**）：实现局部信号暂存

➢程序**ROM**：用户程序的载体

wxhsnow@163.com

数据空间

- 数据空间主要包括：

➢ 算术逻辑单元（**ALU**）：

可变逻辑操作和算术运算模块，是操作与运算的核心模块

➢ 累加器（**ACC**）：完成数据的累加

➢ 移位器（**Shifter**）：

完成数据位的左移、右移或循环移位

➢ 寄存器：

完成一些数据的暂存（操作数、地址、指针）

➢ **RAM**：数据暂存

wxhsnow@163.com

总线

- 总线有几种形式：
  - 分离的程序总线
  - 数据总线（双总线、三总线）：
  - 合并总线：程序、数据复用总线

6.2 微处理器单元设计

- 6.2.1 控制器单元
- 6.2.2 算术逻辑单元ALU
- 6.2.3 乘法器
- 6.2.4 移位器
- 6.2.5 寄存器
- 6.2.6 堆栈

wxhsnow@163.com

6.2.1控制器单元

- 控制器是微处理器的主控单元，也是不同微处理器之间差异最大的单元
- 功能：根据指令或直接给予微处理器的控制以及内部反馈信息产生一组或多组信号，去控制相关逻辑单元进行适当的操作和运算
- 最简单的设计方法：根据输入的信息（程序、外部控制、内部反馈等）与输出控制的要求列出真值表，然后转换成逻辑函数，再进行具体逻辑设计

随机逻辑　PLA技术

- 控制器可采用随机逻辑，**PLA**或**ROM**设计
- 控制器早期采用随机逻辑实现，

  运用多种不同基本逻辑单元，版图设计费时，测试和修改困难
- 现多用规则、重复的结构化单元取代随机逻辑:

  PLA技术、微码控制器（**MicroCode Controller**）
- **PLA**技术适合设计小的控制器

微码控制器

- **ROM形式：根据真值表直接得到相应的结构**
- 微码控制器：一块**ROM**和相应的地址发生器组合（包含了全部控制信息）
- 重要特性：具有很宽的控制字输出
- 其结构如图所示
- 微码控制器的组成：
  - 微控制字存储器
  - 下一地址发生器

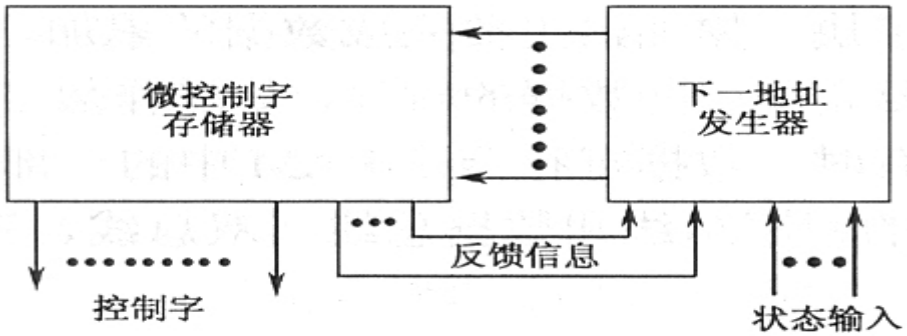


图 6-1　简单微码控制器

微码控制器的组成

- 微控制字存储器：微码**ROM**（**MicroROM**）
  - 实质是一块**ROM**
  - 存放一系列控制字
  - 控制处理器中逻辑模块的行为
  - 其输入为下一地址发生器所馈给的控制字地址
- 下一地址发生器：
  - 根据状态输入信息、从存储器中来的反馈信息生成下一地址

wxhsnow@163.com

微码控制器的工作原理

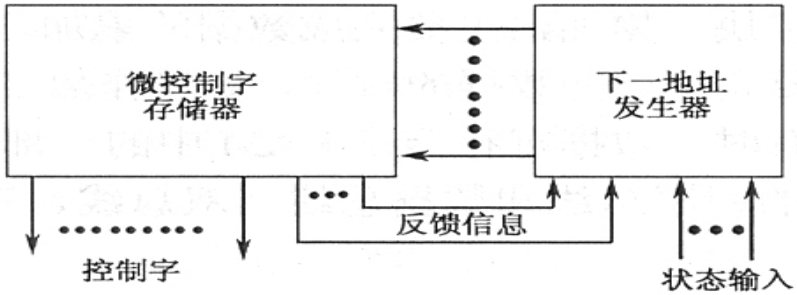


图 6-1　简单微码控制器

- 在一个确定的状态输入控制激励下，下一地址发生器产生一个微码**ROM**地址，选中一个控制字和相应的反馈字

- 控制字输出：选择运算模块
- 反馈字：确定下一地址发生器的下一步动作
  - 在一个状态信息下，运算模块可能会同时或者顺序执行多步动作
  - 这一系列动作和地址发生一直进行到反馈字和内部反馈信息不再产生新的微码**ROM**地址结束

微码控制器的存储器结构示例

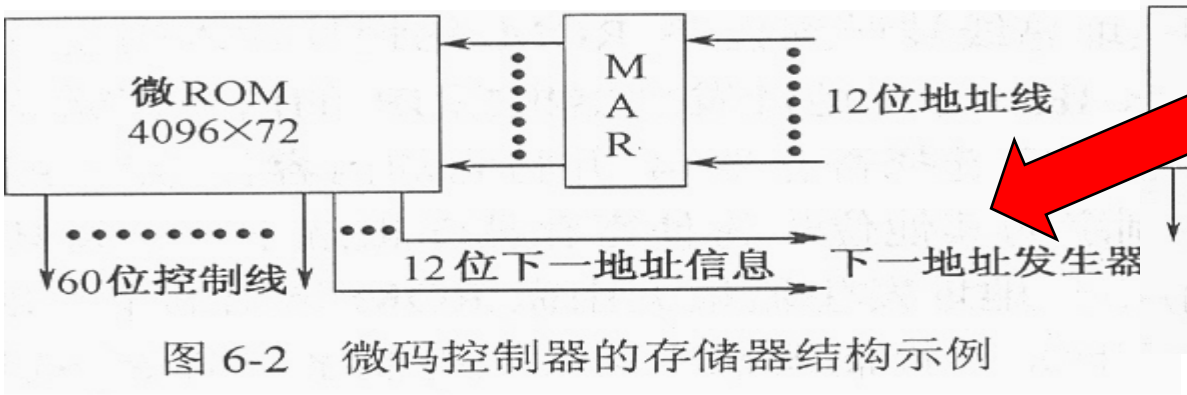


图 6-2　微码控制器的存储器结构示例

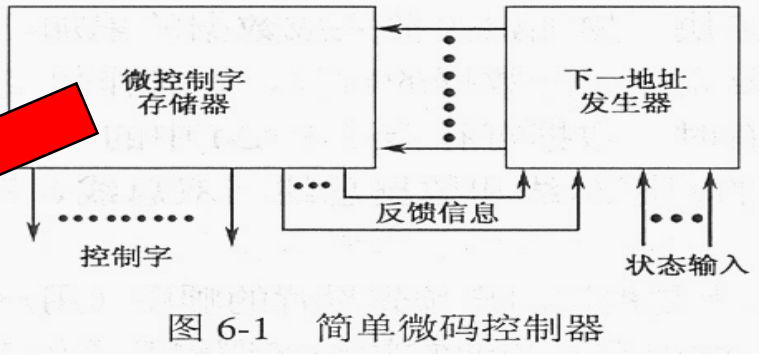


图 6-1　简单微码控制器

- **MAR（Memory Address Register）：**
  存储器地址寄存器：　暂存一条控制字的地址
- **12位地址——4k控制字**
- 每个控制字有**72**位控制信息：
  - **60**位由于控制微处理器组成单元
  - **12**位反馈给下一地址发生器

微码ROM

- 微码ROM数据位多于地址位（微处理器要求的控制信号位数多）
- ＝＝》微码ROM尺寸非常大（29万多个）
- 控制字空间浪费：某些地址根本不会出现
  - 10位1024条，9位512条，513～1024必须用10位
- 控制字重复：不同的地址对应的控制字相同
  - 去除不必要和重复的控制字
  - 多个状态信息对应一个控制字，

    要求：地址映射逻辑

wxhsnow@163.com

减小微ROM尺寸

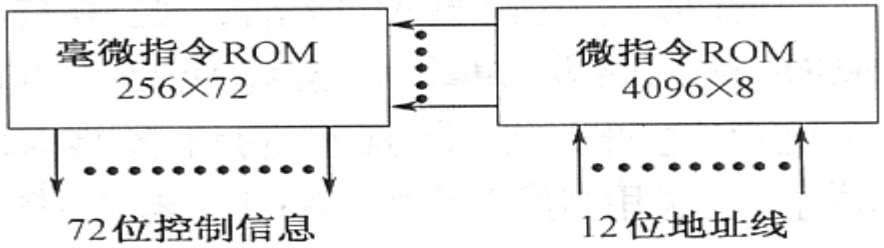


图 6-3　两级微程序存储器结构示意图

- 举例：4096个控制字＝》256个控制字
- 　　14位　　＝》8位
- 微指令ROM：功能类似于译码器
- 毫微指令ROM：存放256个72位的字
- 优点：减小了微ROM的尺寸
- 缺点：增加了电路级数，延迟时间增长

wxhsnow@163.com

微码控制器控制的微处理器的工作过程

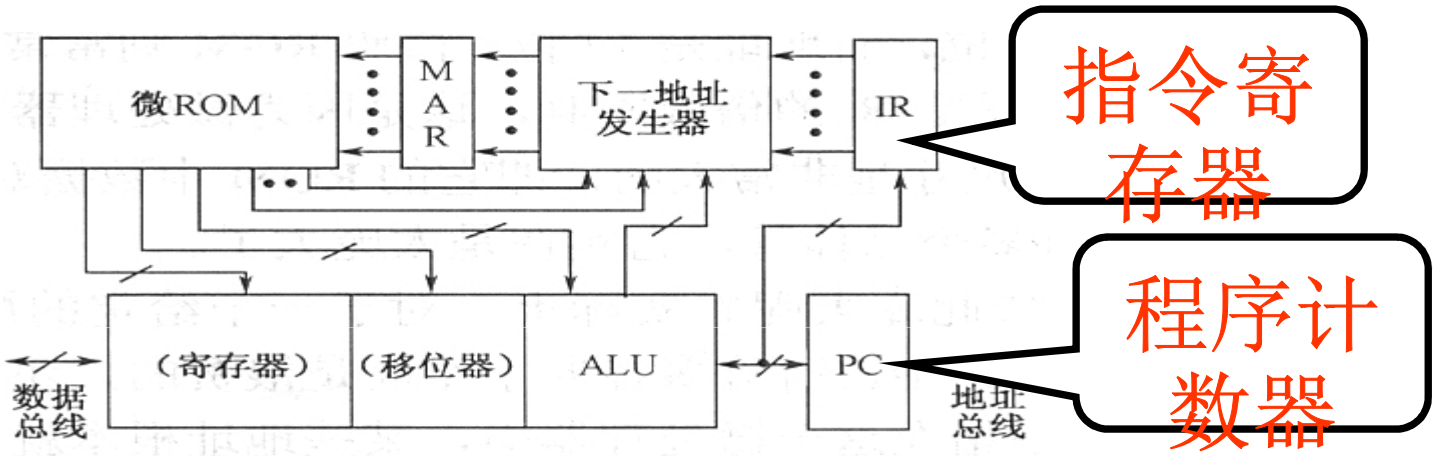


图 6-4　微码控制器控制的运算单元结构示意图

6.2.2算术逻辑单元（ALU）

- 1、信号结构
- 2、全加器Full-adder
- 3、以全加器为核心构造的ALU
- 4、函数发生逻辑电路

ALU的信号结构

- ALU是数据空间的最主要的单元
- 微处理器的核心
- 程序需要的各种主要的算术运算和逻辑操作都通过ALU完成
- 通常的逻辑操作包括：

  逻辑与、逻辑或、逻辑异或、取反、求补等
- 通常的算术操作包括：

  加、减、比较、算术左移等
- 求长运算（乘法运算等）将进行硬件结构扩充

ALU的信号结构

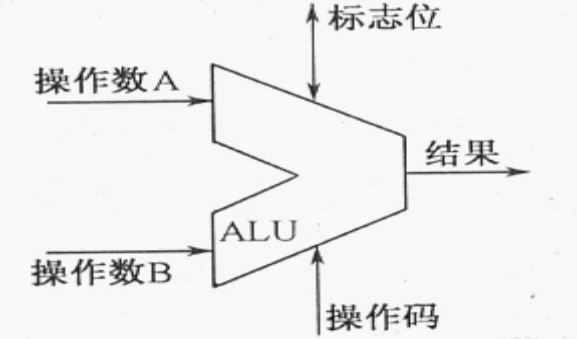


图 6-5　ALU 信号结构

- **ALU内部要求对输入的信息立即产生反应，不需要寄存**
- **操作数的位数由微处理器的基本数据宽度决定**

- 操作码：控制信息，对所需的操作进行选择和控制
- 操作码的位数由所需进行的操作与运算类型数量决定
- 标志位：表示操作的属性

半加器Half-adder

- 半加器：不考虑前级进位输入C_{i-1}

$$S_i = A_i \oplus B_i \qquad C_i = A_i \cdot B_i$$

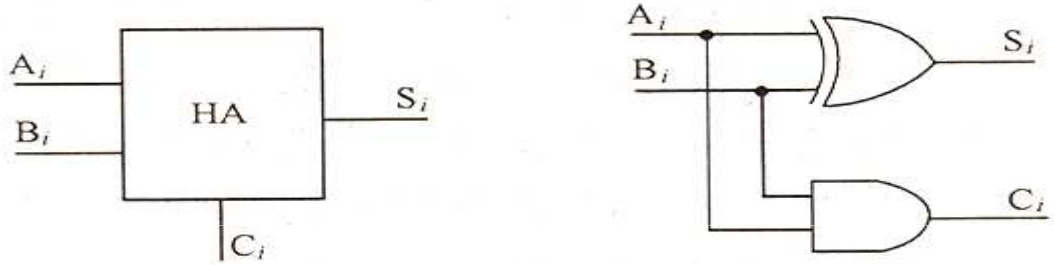


图 6-8　半加器逻辑结构图

- 半加器与函数发生器逻辑配合可实现与、与非、或、或非等逻辑操作
- 在实际设计中，若逻辑输入量只有两个，可用半加器代替全加器

全加器

- 全加器（**Full-adder**）：**ALU** 的核心
- 指可以进行带进位输入和输出的加法运算单元

输　入			输　出	
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

A_i、B_i：需要相加的两个二进制数

C_{i-1}：前级进位输入

S_i：本位和输出

C_i：本级进位输出

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = (A_i + B_i) \cdot C_{i-1} + A_i \cdot B_i$$

全加器

- 全加器可以实现$A_i+B_i+C_{i-1}$，并产生进位
- 若将B_i倒相后输入，则成为减法

$$A-B \Rightarrow A+[-B]_{补}=A+\overline{B}+1$$

$$\therefore \overline{B} \Rightarrow -B-1$$

则：$A_i+\overline{B}_i+C_{i-1} \Rightarrow A_i-B-1+C_{i-1}$

- 若$C_{i-1}=0$，则全加器实现借位的减法A_i-B_i-1
- 若$C_{i-1}=1$，则全加器实现普通减法运算A_i-B_i

前级进位C_{i-1}看作控制信号

- 全加器在不同的控制输入下表现出不同的逻辑操作功能

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$= (A_i \oplus B_i) \cdot \overline{C_{i-1}} + \overline{(A_i \oplus B_i)} \cdot C_{i-1}$$

$$= (A_i \oplus B_i) \cdot \overline{C_{i-1}} + (A_i \odot B_i) \cdot C_{i-1}$$

$$= (\overline{A_i} \cdot B_i + A_i \cdot \overline{B_i}) \cdot \overline{C_{i-1}} + (\overline{A_i} \cdot \overline{B_i} + A_i \cdot B_i) \cdot C_{i-1}$$

前级进位C_{i-1}看作控制信号

$$S_i=(\overline{A_i}\cdot B_i+A_i\cdot\overline{B_i})\cdot\overline{C_{i-1}}+(\overline{A_i}\cdot\overline{B_i}+A_i\cdot B_i)\cdot C_{i-1}$$

当$C_{i-1}=0$时　$S_i=\overline{A_i}\cdot B_i+A_i\cdot\overline{B_i}$　　实现异或操作

若$A_i=0, S_i=B_i$　　传输B_i

若$A_i=1, S_i=\overline{B_i}$　　实现B_i的倒相

其中B_i可为多个变量的组合逻辑

当$C_{i-1}=1$时　$S_i=\overline{A_i}\cdot\overline{B_i}+A_i\cdot B_i$　　实现同或操作

若$A_i=0, S_i=\overline{B_i}$　　实现B_i的倒相

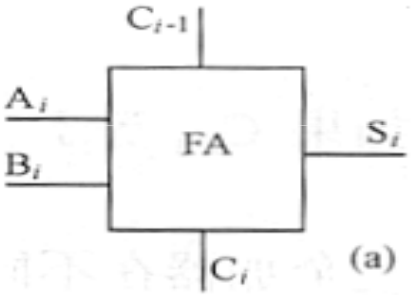
若$A_i=1, S_i=B_i$　　传输B_i

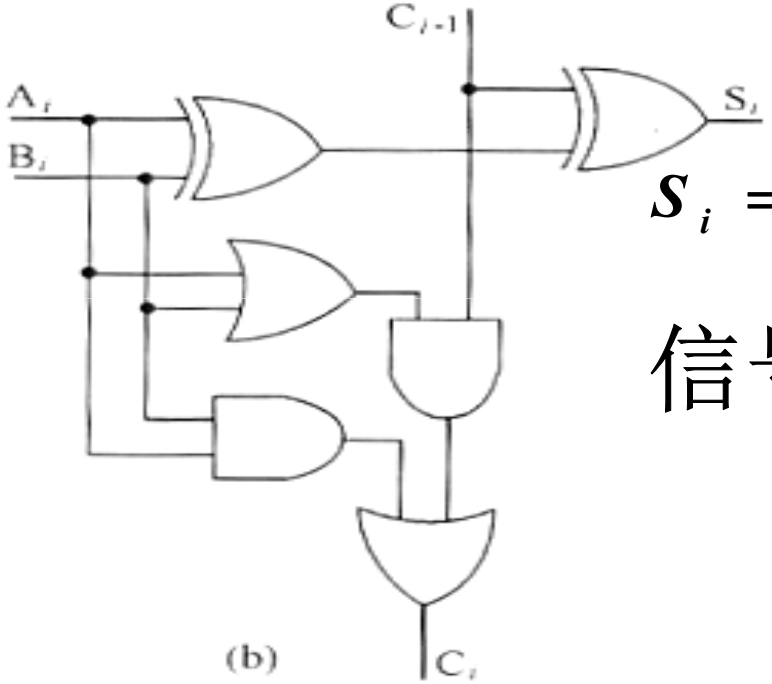
全加器能够在适当的输入与控制下完成不同的算术运算与逻辑操作

wxhsnow@163.com

全加器的逻辑结构图

- 全加器的具体电路结构有多种形式





$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

信号延迟比较大

$$C_i = (A_i + B_i) \cdot C_{i-1} + A_i \cdot B_i$$

MUX实现全加器

- 根据加法器的逻辑可得出真值表

A_i	B_i	S_i
0	0	C_{i-1}
1	1	C_{i-1}
1	0	$\overline{C_{i-1}}$
0	1	$\overline{C_{i-1}}$

A_i	B_i	C_i
0	0	A_i
1	1	A_i
1	0	C_{i-1}
0	1	C_{i-1}

B	A	Z
0	0	C_0
0	1	C_1
1	0	C_2
1	1	C_3

$$S = \overline{A_i} \cdot \overline{B_i} \cdot C_{i-1} + A_i \cdot B_i \cdot C_{i-1} + A_i \cdot \overline{B_i} \cdot \overline{C_{i-1}} + \overline{A_i} \cdot B_i \cdot \overline{C_{i-1}}$$

$$C_i = \overline{A_i} \cdot \overline{B_i} \cdot A_i + A_i \cdot B_i \cdot A_i + A_i \cdot \overline{B_i} \cdot C_{i-1}$$

$$+ \overline{A_i} \cdot B_i \cdot C_{i-1}$$

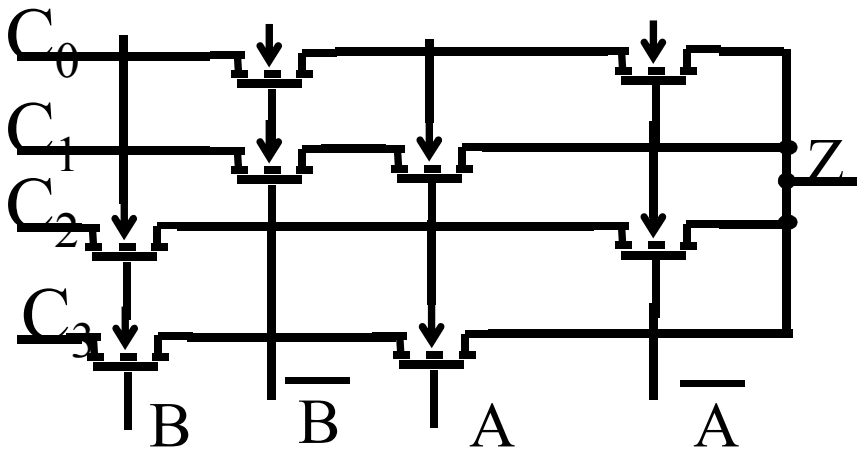
不能化简，防止出现高阻状态

多路转换开关MUX

- NMOS四到一转换开关电路

B	A	Z
0	0	C_0
0	1	C_1
1	0	C_2
1	1	C_3

转换关系

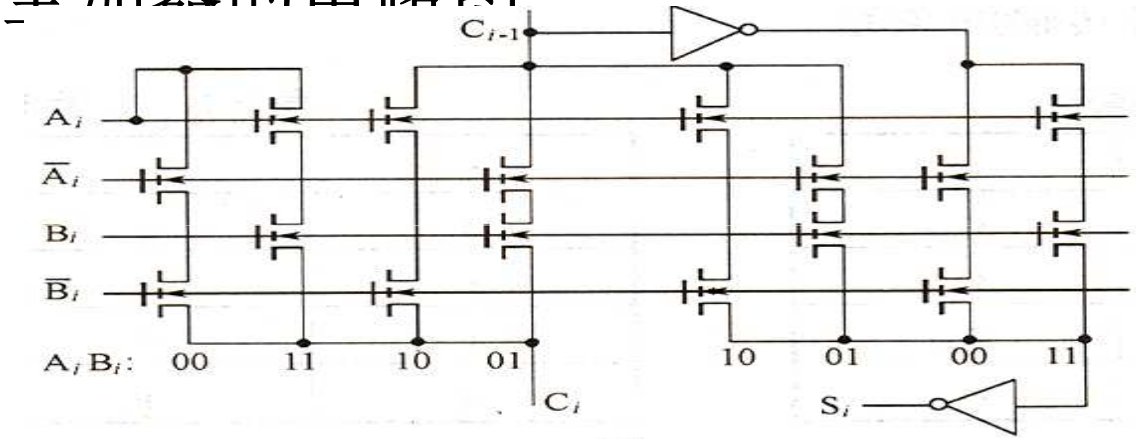


转换电路

全加器

- NMOS结构全加器的电路图

$$C_i = \overline{A_i} \cdot \overline{B_i} \cdot A_i + A_i \cdot B_i \cdot A_i + A_i \cdot \overline{B_i} \cdot C_{i-1} + \overline{A_i} \cdot B_i \cdot C_{i-1}$$



(a)

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$\overline{S_i} = (A_i \oplus B_i) \odot C_{i-1}$$

$$= A_i \cdot \overline{B_i} \cdot C_{i-1} + \overline{A_i} \cdot B_i \cdot C_{i-1} + A_i \cdot B_i \cdot \overline{C_{i-1}} + \overline{A_i} \cdot \overline{B_i} \cdot \overline{C_{i-1}}$$

- 电路结构简单、规则
- 可采用带提升电路的NMOS结构全加器
- 可用CMOS传输逻辑实现

- 例6-1：设计一个实现四种逻辑操作的电路，其中控制信号为K_1、K_0,逻辑输入为A、B.

  当K_1K_0＝00时，实现A、B的与非操作；

  当K_1K_0＝01时，实现A、B的或非操作；

  当K_1K_0＝10时，实现A、B的异或操作；

  当K_1K_0＝11时，实现A的倒相操作；

分析：与非：$S_i = A_i \oplus B_i = A_i\overline{B}_i + \overline{A}_i B_i = \overline{A \cdot B}$ $\Rightarrow A_i = A + B, B_i = B$；

或非：$S_i = A_i \oplus B_i = \overline{A+B}$ $\Rightarrow A_i = \overline{A} + B, B_i = B$；

异或：$S_i = A_i \oplus B_i = A \oplus B$ $\Rightarrow A_i = A, B_i = B$；

倒相A：$S_i = A_i \oplus B_i = \overline{A}$ $\Rightarrow A_i = A, B_i = 1$；

例题

- 根据分析得出输入输出关系

K_1	K_0	A_i	B_i	S_i
0	0	$A+\overline{B}$	B	$\overline{A \cdot B}$
0	1	$\overline{A}+B$	B	$\overline{A+B}$
1	0	A	B	$\overline{A} \cdot B+A \cdot \overline{B}$
1	1	A	1	$\overline{A}$

$$A_i = \overline{K_1} \cdot \overline{K_0}(A+\overline{B}) + \overline{K_1} \cdot K_0(\overline{A}+B) + K_1 \cdot A$$

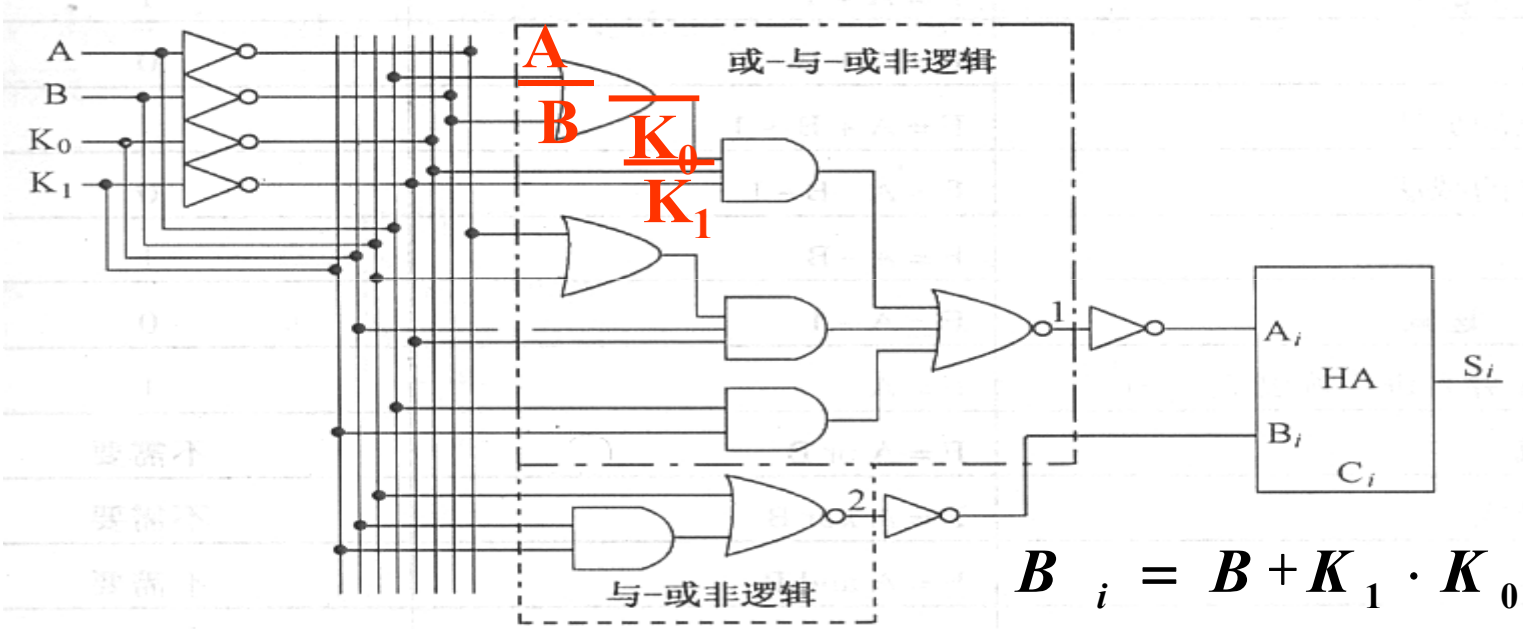
$$B_i = B + K_1 \cdot K_0$$

采用“或—与—或非—倒相器”实现 A_i

采用“与—或非—倒相器”实现 B_i

例题

$$A_i = \overline{K_1} \cdot \overline{K_0}(A + \overline{B}) + \overline{K_1} \cdot K_0(\overline{A} + B) + K_1 \cdot A$$



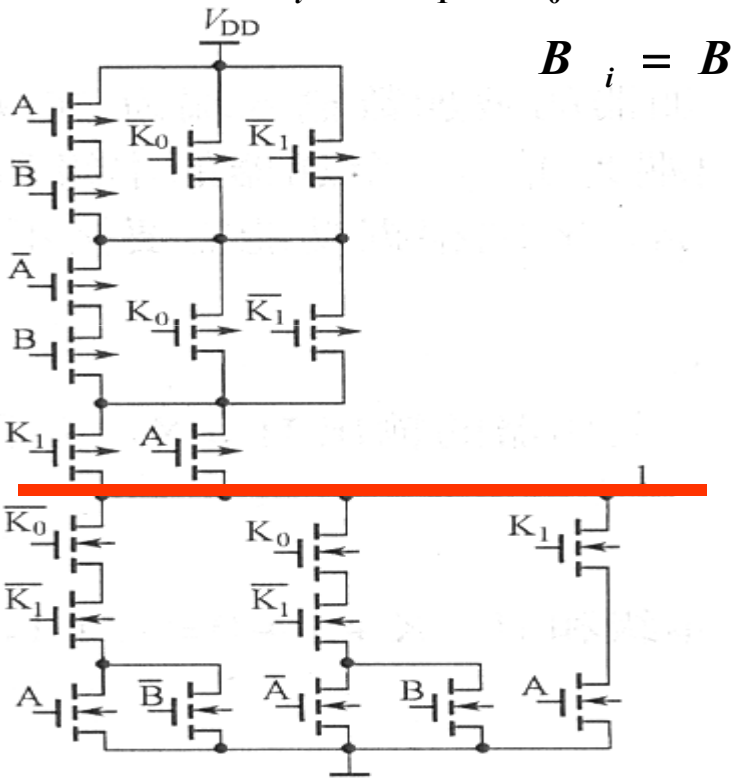
$$B_i = B + K_1 \cdot K_0$$

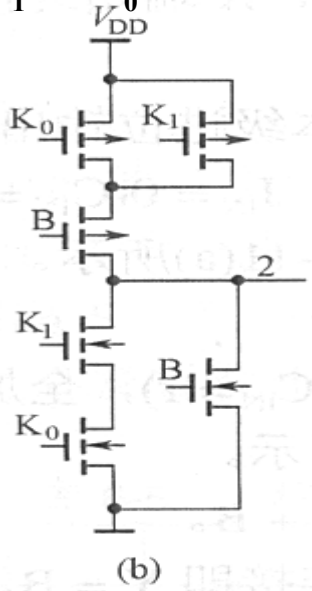
图 6-9　逻辑结构图

组合逻辑门电路

$$A_i = \overline{K_1} \cdot \overline{K_0}(A + \overline{B}) + \overline{K_1} \cdot K_0(\overline{A} + B) + K_1 \cdot A$$

$$B_i = B + K_1 \cdot K_0$$





以全加器为核心构造的ALU

- ALU功能表
- 八种算术运算
- 四种逻辑运算

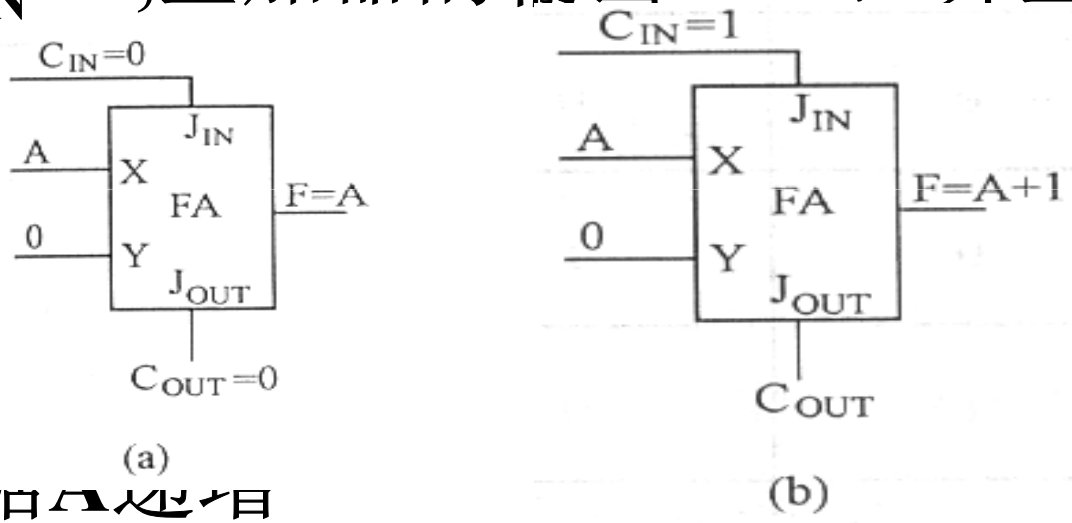
功能要求	函数F	进位输入Cin
传送A并且进位输出 C_{OUT}=0	F=A	0
A加1（递增）	F=A+1	1
加法	F=A+B	0
带进位的加法	F=A+B+1	1
带借位的减法	F=A – B - 1	0
减法	F=A – B	1
A减1（递减）	F=A – 1	0
传送A并且进位输出 C_{OUT}=1	F=A	1
逻辑或	F=A or B	不需要
逻辑异或	F=A xor B	不需要
逻辑与	F=A and B	不需要
逻辑非	F=not A	不需要

ALU实现算术运算的设计

- 为与数据信号加以区别，做以下定义：
  - 被加数输入端为**X**端——接数据**A**；
  - 加数的输入端为**Y**端——根据功能需要接不同的输入；
  - 全加器的本级输出端**H**——结果**F**
  - 前级进位输入端为J_{IN}本级进位输出端J_{OUT}；

ALU实现算术运算的设计（一）

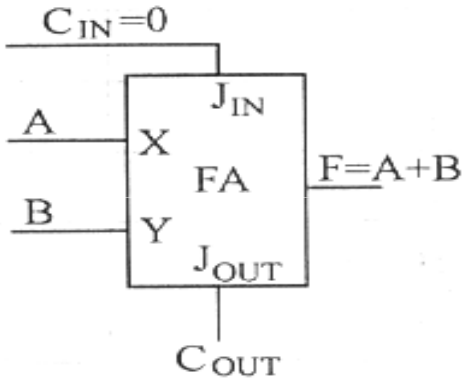
- 1、传送A并且进位输出为0：
  - Y=0, J_{IN}=0,全加器的输出H=X，并且J_{OUT}=0



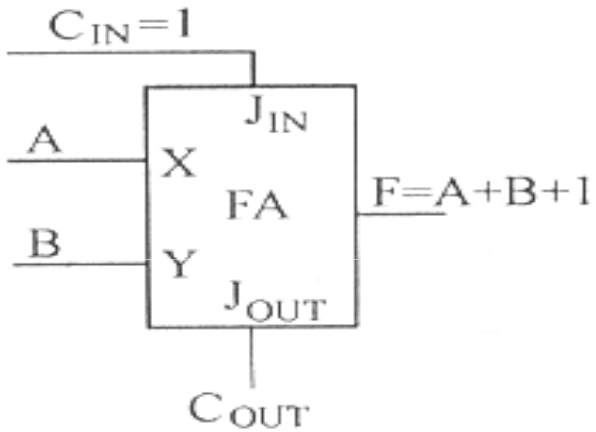
- 2、数据A进行
  - Y=0, J_{IN}=1,全加器的输出H=X＋1

ALU实现算术运算的设计（二）

- 3、加法运算，F=A+B
  - Y=B, J_{IN}=0. 输出H=X＋Y



(c)

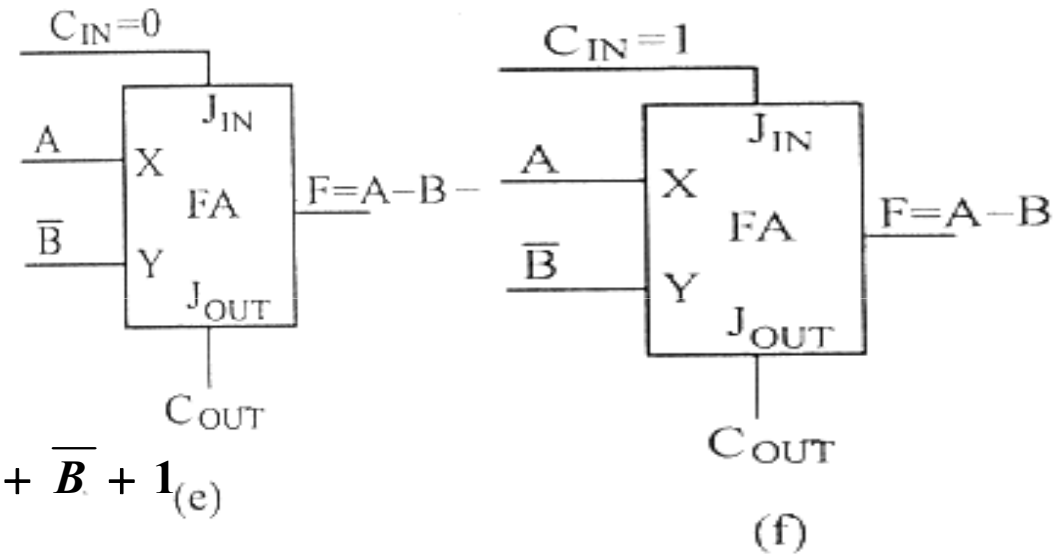


(d)

- 4、带进位的加法:F=A+B+1
  - Y=B, J_{IN}=1, 输出H=X＋Y+1

ALU实现算术运算的设计（三）

- 5、带借位的减法:F=A－B－1 $= A + \overline{B}$
  - Y= $\overline{B}$, J_{IN} =0

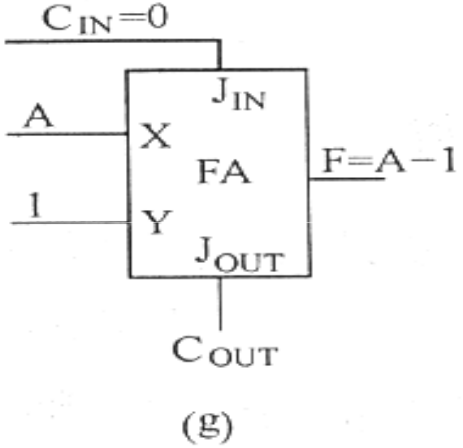


$A - B \Rightarrow A + [-B]_{补} = A + \overline{B} + 1$

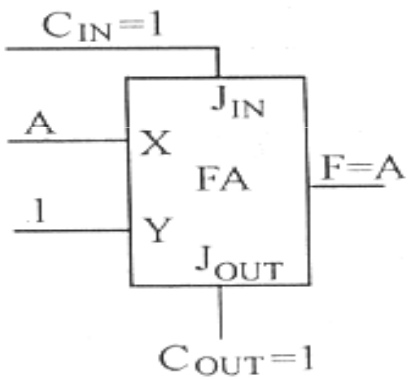
- 6、减法运算： F=A－B
  - Y= $\overline{B}$, J_{IN} =1

ALU实现算术运算的设计（四）

- 7、数据A递减运算：
  - Y=1, J_{IN}=0,全加器的本级和取A的非量
    实现F=A－1



(g)



(h)

- 8、传送A并且进位输出C_{OUT}=1
  - Y=1, J_{IN}=1,全加器的输出H=A

进行一位算术运算的ALU

- 进行一位算术运算的ALU 的功能和信号关系

表 6-6　ALU 算术运算的功能和信号关系

功能要求	函数 F	C_{IN}	S_1	S_0	Y	X
传送 A（$C_{OUT}=0$）	F = A	0	0	0	0	A
A 加 1（递增）	F = A + 1	1				
加法	F = A + B	0	0	1	B	
带进位的加法	F = A + B + 1	1				
带借位的减法	F = A - B - 1	0	1	0	$\overline{B}$	
减法	F = A - B	1				
A 减 1（递减）	F = A - 1	0	1	1	1	
传送 A（$C_{OUT}=1$）	F = A	1				

进行一位算术运算的ALU

- 进行一位算术运算的ALU 的功能和信号关系

表 6-6　ALU 算术运算的功能和信号关系

功能要求	函数 F	C_{IN}	S_1	S_0	Y	X
传送 A（$C_{OUT}=0$）	F = A	0	0	0	0	A
A 加 1（递增）	F = A + 1	1				
加法	F = A + B	0	0	1	B	
带进位的加法	F = A + B + 1	1				
带借位的减法	F = A - B - 1	0	1	0	$\overline{B}$	
减法	F = A - B	1				
A 减 1（递减）	F = A - 1	0	1	1	1	
传送 A（$C_{OUT}=1$）	F = A	1				

ALU实现算术运算的设计

- X端始终接信号A端，算术逻辑运算实际是对Y和J_{IN}的设计。
  - Y的取值有四种可能：0，1，B　$\overline{B}$
- 可通过控制码和相应的逻辑产生这四个值

$$Y = B \cdot S_0 + \overline{B} \cdot S_1$$

表 6-5　Y 状态真值表

S_1	S_0	Y
0	0	0
0	1	B
1	0	$\overline{B}$
1	1	1

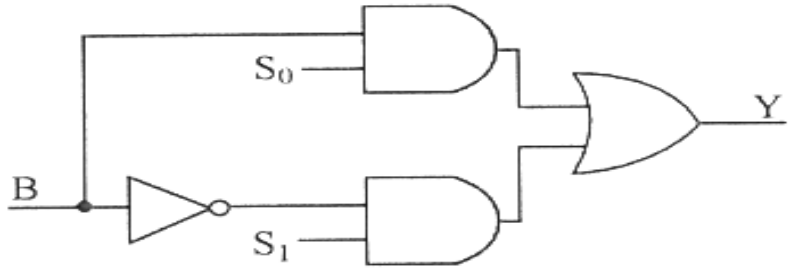


图 6-12　Y 的逻辑图

进行一位算术运算的ALU

- 进行一位算术运算的**ALU** 为：

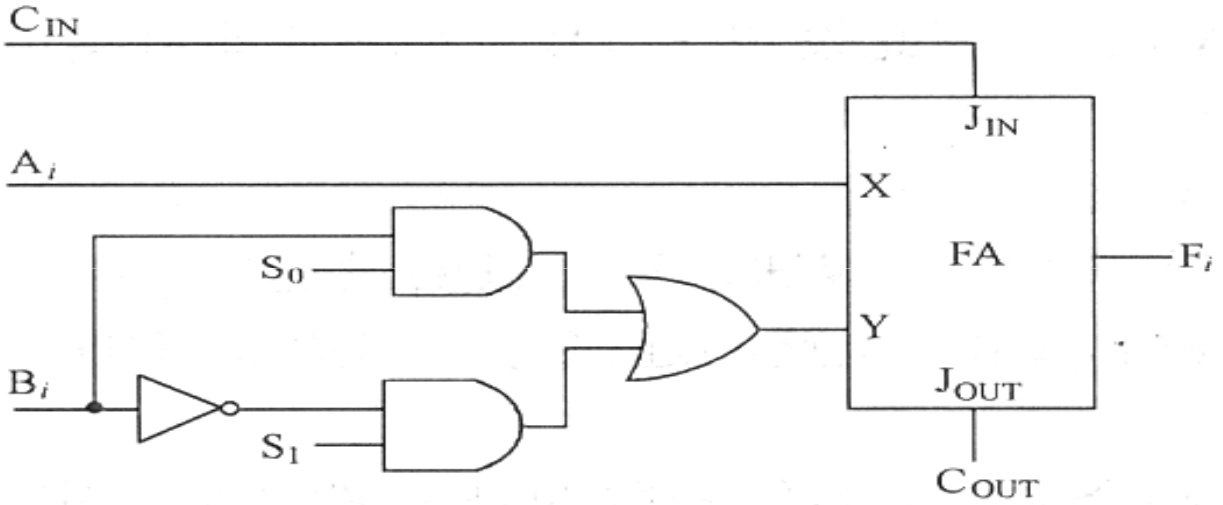


图 6-13　进行一位算术运算的 ALU 逻辑结构

- 由一位算术运算的**ALU** 可构造多位实现算术运算的**ALU**逻辑

ALU实现逻辑运算的设计（一）

- 逻辑操作的控制通过对全加器的X端和J_{IN} 的信号控制实现
  - 逻辑操作不需要考虑进位，应令$J_{IN}=0$
  - 也可以用第三位控制码S_2对J_{IN}的输入进行控制

令 $J_{IN}=\overline{S_2}C_{IN}$

当 $S_2=0$时，$J_{IN}=C_{IN}$；

当 $S_2=1$时，$J_{IN}=0$；

- 进行逻辑操作时，全加器的本位和输出为：

$$F=X\oplus Y\oplus J_{IN}=X\oplus Y$$

wxhsnow@163.com

ALU实现逻辑运算的设计（二）

根据 $J_{IN} = \overline{S_2} C_{IN}$　　$Y = B \cdot S_0 + \overline{B} \cdot S_1$

$F = X \oplus Y \oplus J_{IN}$

- 在各种控制状态下，加法器的功能如下：
  - 当$S_2S_1S_0$=100时,Y=0，令X=A+B,实现或
  - 当$S_2S_1S_0$=101时,Y=B，令X=A,实现异或
  - 当$S_2S_1S_0$=110时,Y= $\overline{B}$ 令X=A+ $\overline{B}$,实现与
  - 当$S_2S_1S_0$=111时,Y=1，令X=A,实现A的倒相

综合算术运算和逻辑操作的要求，可得全加器的输入要求：

$$X = A + (S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot B) + (S_2 \cdot S_1 \cdot \overline{S_0} \cdot \overline{B})$$

$$Y = (S_0 \cdot B) + (S_1 \cdot \overline{B}) \qquad J_{IN} = \overline{S_2} \cdot C_{IN}$$

ALU的逻辑结构

$$X = A + (S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot B)$$
$$+ (S_2 \cdot S_1 \cdot \overline{S_0} \cdot \overline{B})$$

$$Y = (S_0 \cdot B) + (S_1 \cdot \overline{B})$$

$$J_{IN} = \overline{S_2} \cdot C_{IN}$$

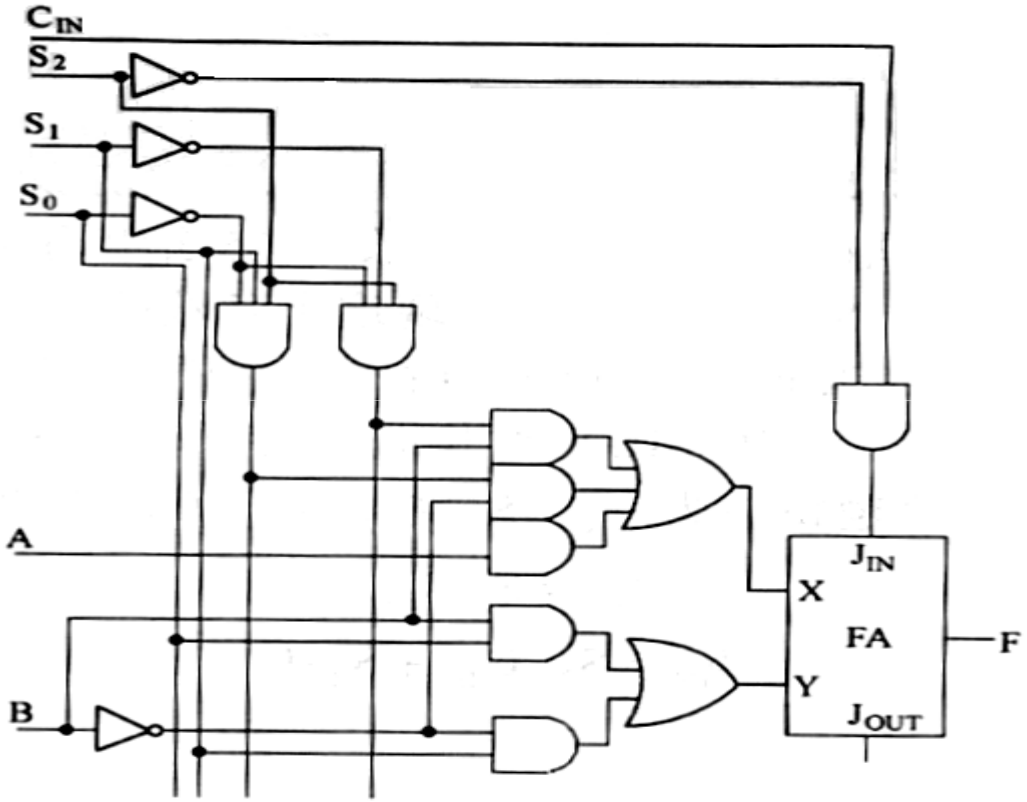


图 6-14　ALU 的局部逻辑结构图

ALU功能与控制信息汇总表

表 6-7　ALU 功能与控制信息汇总表

功能要求	函数 F	S_2	S_1	S_0	C_{IN}
传送 A（$C_{OUT}=0$）	F = A	0	0	0	0
A 加 1（递增）	F = A + 1	0	0	0	1
加法	F = A + B	0	0	1	0
带进位的加法	F = A + B + 1	0	0	1	1
带借位的减法	F = A − B − 1	0	1	0	0
减法	F = A − B	0	1	0	1
A 减 1（递减）	F = A − 1	0	1	1	0
传送 A（$C_{OUT}=1$）	F = A	0	1	1	1
逻辑或	F = A or B	1	0	0	×
逻辑异或	F = A xor B	1	0	1	×
逻辑与	F = A and B	1	1	0	×
逻辑非	F = not A	1	1	1	×

ALU功能与控制信息汇总表

- 表中所列出的控制码的位数是四位，C_{IN}以数据方式设置
- 设计多位ALU必然会使用前级进位端
- 通常情况下，控制位的位数由ALU所要执行的运算功能数量决定

函数发生逻辑电路

- J_{IN}函数发生逻辑
- Y函数发生逻辑
- X函数发生逻辑

➢ 函数发生逻辑可采用组合逻辑门结构，也可采用传输晶体管逻辑

wxhsnow@163.com

J_{IN}函数发生逻辑

$J_{IN} = \overline{S_2} \cdot C_{IN}$

($S_2 = 1$ 时进行逻辑运算，不需要进位)

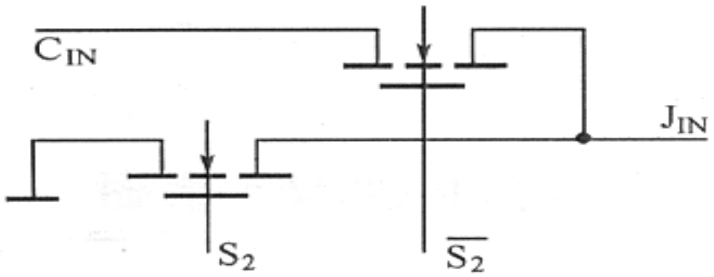


图 6-15　J_{IN} 函数发生电路

- 进位输入端接外部控制输入（或前级进位输出）或者接“0”
- 采用二到一的NMOS MUX实现这个函数
  - 函数中只有一个与项，但采用MUX结构时必须考虑S_2=1情况——表达一种信号传输关系，防止出现高阻

实际的传输函数应为：$J_{IN} = \overline{S_2} \cdot C_{IN} + S_2 \cdot 0$

Y函数发生逻辑

根据 $Y = B \cdot S_0 + \overline{B} \cdot S_1$

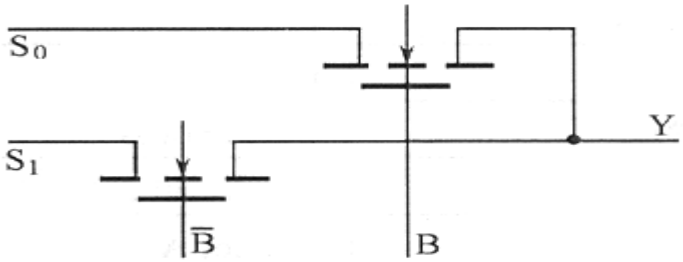


图 6-17　Y 函数的 MUX 结构

将信号 B 作为控制信号，

采用简单的二到一的 MUX 实现 Y 的逻辑

Y函数发生逻辑

表 6-5　Y 状态真值表

S_1	S_0	Y
0	0	0
0	1	B
1	0	$\overline{B}$
1	1	1

$$Y = B \cdot S_0 + \overline{B} \cdot S_1$$

$$= \overline{(B \cdot S_0) + (\overline{B} \cdot S_1)}$$

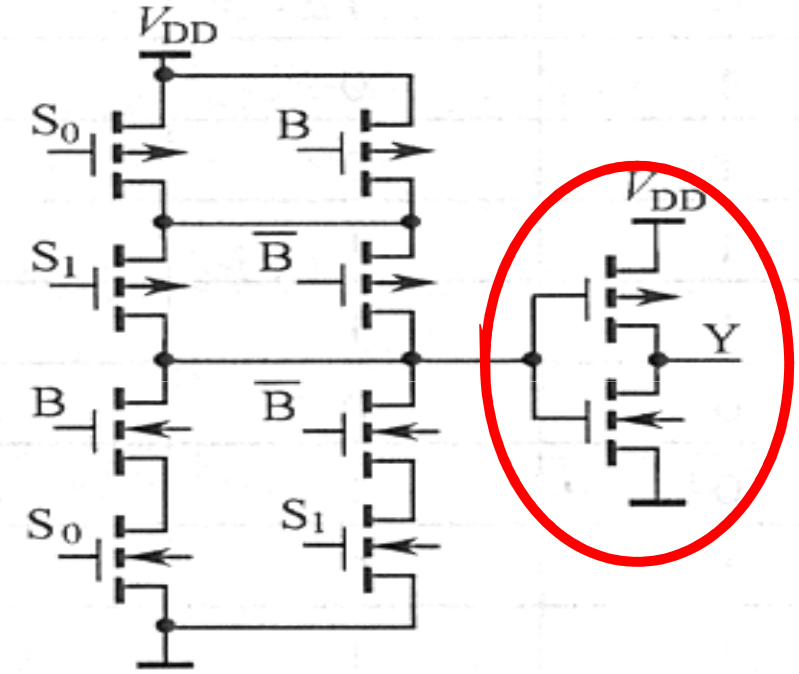


图 6-16　Y 函数发生电路

X函数发生逻辑

X函数逻辑表达式：

$$X = A + (S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot B) + (S_2 \cdot S_1 \cdot \overline{S_0} \cdot \overline{B})$$

进行算术运算时，$S_2 = 0$，控制改变Y

进行逻辑运算时，$S_2 = 1$，控制改变X

- 将简化掉的开关变量补齐

$$X = \overline{S_2} A + S_2 \cdot A + (S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot B) + (S_2 \cdot S_1 \cdot \overline{S_0} \cdot \overline{B})$$

$$X = (\overline{S_2} + S_0) A + S_2 \cdot \overline{S_0} [\overline{S_1}(A + B) + S_1 \cdot \overline{(A + B)}]$$

简单推导 $(\overline{S_2} + S_0) A + S_2 \cdot \overline{S_0} [\overline{S_1} A + S_1 \cdot A] \xrightarrow{?} A$

$$= (\overline{S_2} + S_0) A + S_2 \cdot \overline{S_0} A$$

$$(X + Y) + \overline{X} \cdot \overline{Y} = (X + Y) + \overline{X + Y} = 1$$

wxhsnow@163.com

X函数发生逻辑

$$X = (\overline{S_2} + S_0)\ \overline{A} + S_2 \cdot \overline{S_0}\,[\overline{S_1}(A + B) + S_1 \cdot (A + \overline{B})]$$

$$X = (\overline{S_2} + S_0)\ A + S_2 \cdot \overline{S_0}\,[\overline{S_1}\ \overline{A \cdot B} + S_1 \cdot \overline{A \cdot B}]$$

$$\overline{S_1} \cdot \overline{A} \cdot \overline{B} + S_1 \cdot \overline{A} \cdot B$$

$$\overline{\overline{S_1} \cdot \overline{A} \cdot \overline{B} + S_1 \cdot \overline{A} \cdot B}$$

$$S_1 \cdot A \cdot B + \overline{S_1} \cdot A \cdot B$$

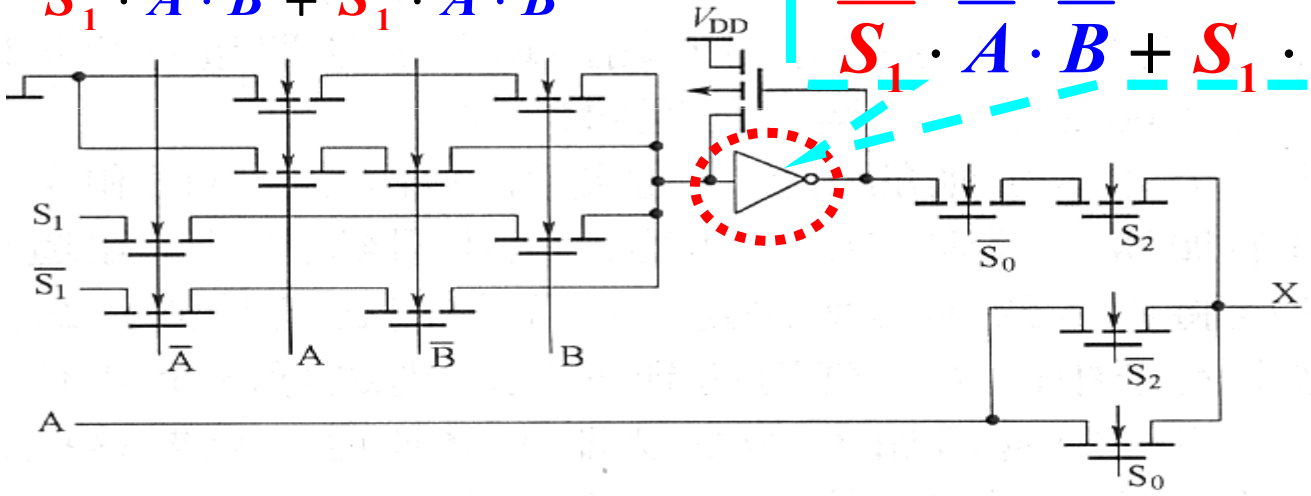


图 6-18　X 函数发生电路

其他方式？

函数发生逻辑

- 采用晶体管逻辑可以使电路结构非常简单
- 缺点：由于阈值电压损耗和串联电阻的作用，将对速度性能产生影响

- **ALU** 的速度将影响整个微处理器的速度

＝＝》超前进位加法器

wxhsnow@163.com

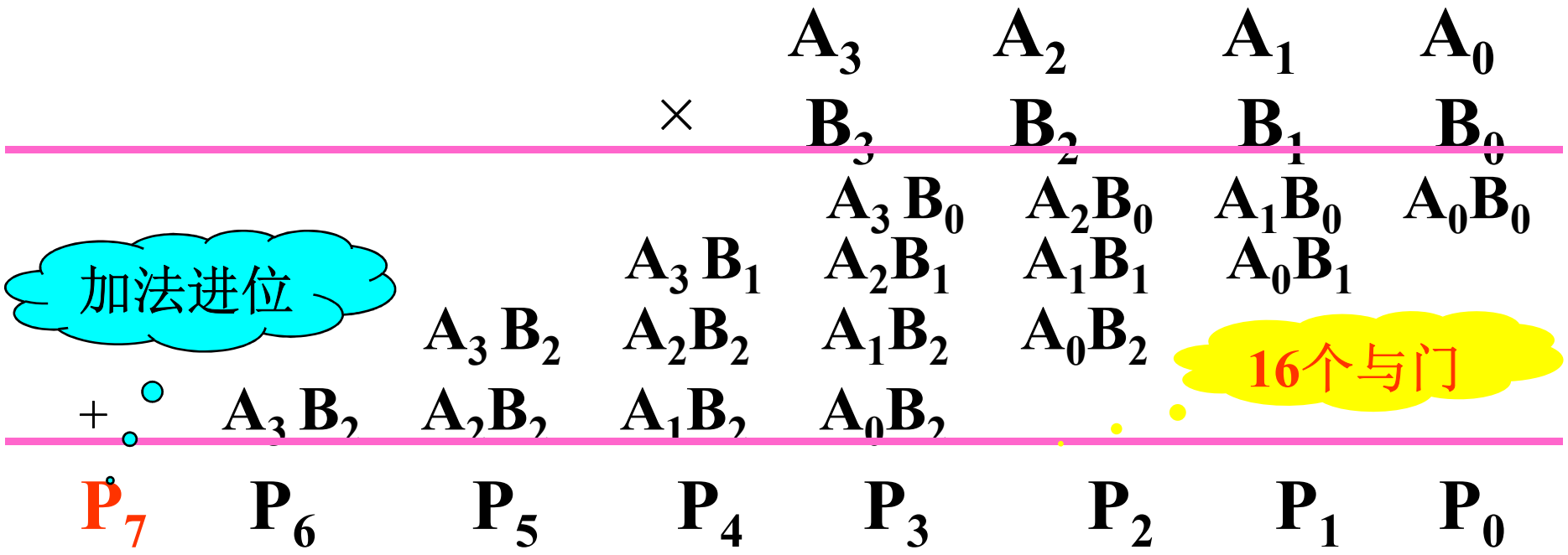
6.2.3 乘法器

wxhsnow@163.com

6.2.3 乘法器

- 乘法器逻辑设计的基本依据：

  算术中的乘法竖式



乘法器的逻辑结构

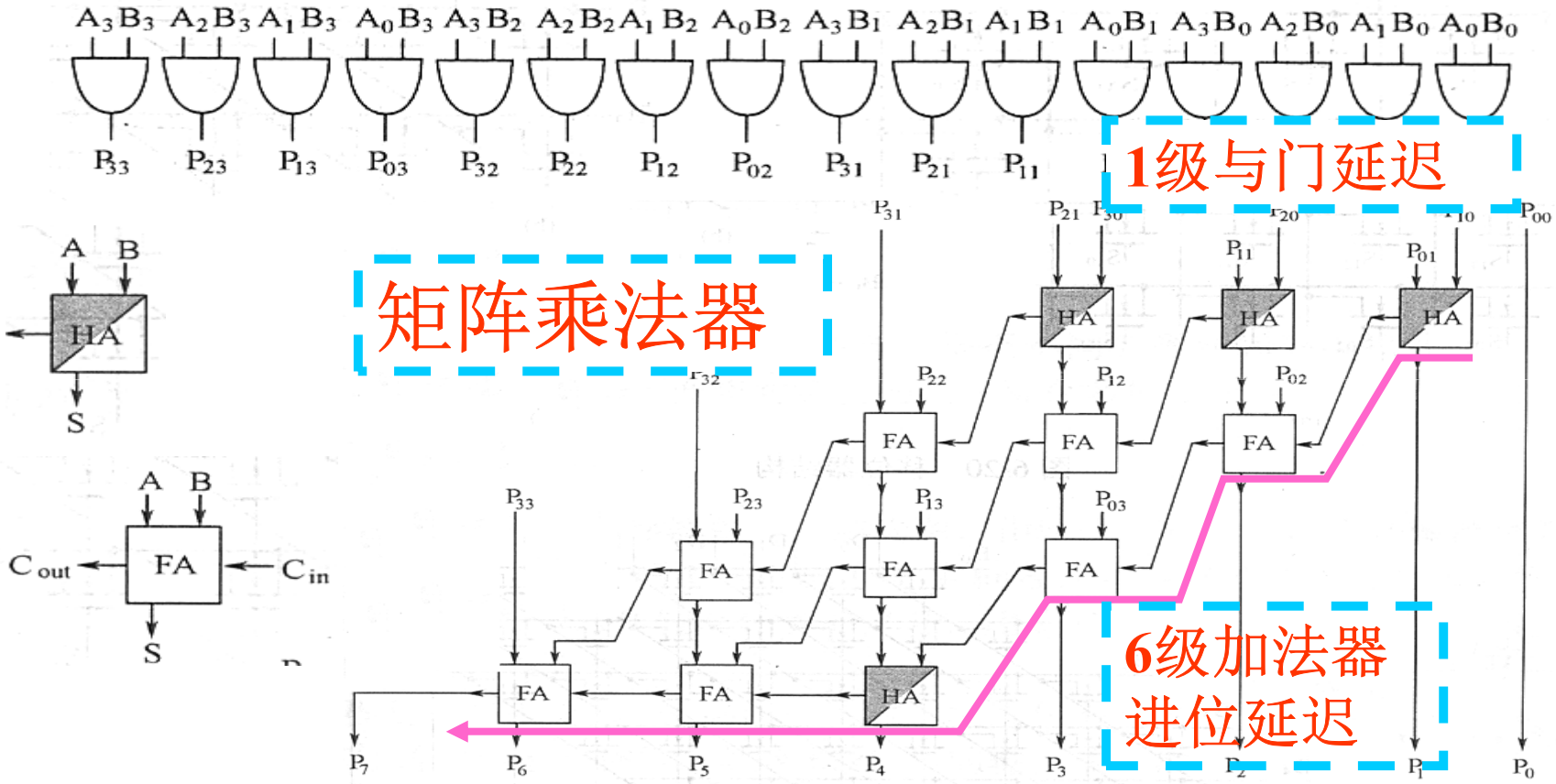


图 6-19　乘法器结构

6.2.3 乘法器

- 延迟时间之和控制在一个指令周期内，即可实现单指令周期的乘法操作
- 考虑取数过程，可能需要两个指令周期完成取数和相乘——*并行可节省时间*
- 数据的位数越多，延迟时间越长，要求硬件的处理越快
- 在需要大量乘法运算的处理器中，专门设计乘法器硬件逻辑，与**ALU**共同完成所需的所有算术运算（如数字信号处理器**DSP**）

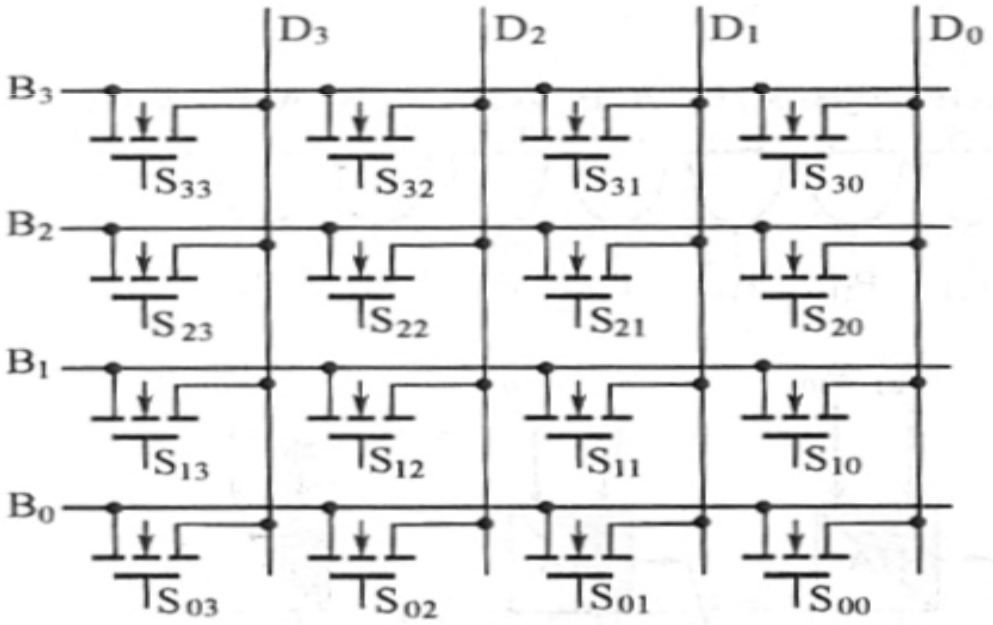
wxhsnow@163.com

指令并行——节省时间

指令1	取指令	取数据	计算	存数据			
指令2		取指令	取数据	计算	存数据		
指令3			取指令	取数据	计算	存数据	
指令4				取指令	取数据	计算	存数据

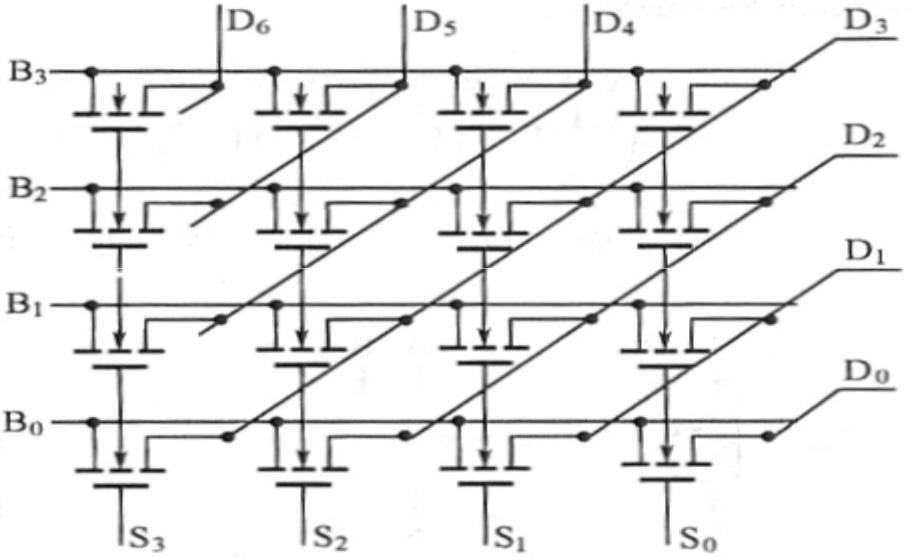
6.2.4 移位器

- 处理器的许多*运算*和*“位操作”*功能通过移位实现
- 移位器通常由多组传输晶体管组成
- 如图所示 4×4的NMOS 开关阵列
- 多路转换开关
- 16个NMOS晶体管
- 16个NMOS控制信号
- 工作原理
  - 不允许两个开关选中同一个输出
  - 控制信号太多



桶型移位器 Barrel Shifter

- 桶型移位器：采用分组控制
- 工作原理：
  - $S_0 \sim S_3$确定移动量
  - S_0=1，B_i对应D_i
  - S_1=1，B_i对应D_{i+1}



- 左移三位如何实现？

循环方式的桶型移位器

- S_0=1，不移位
- S_1=1,右移一位

  或左移7位

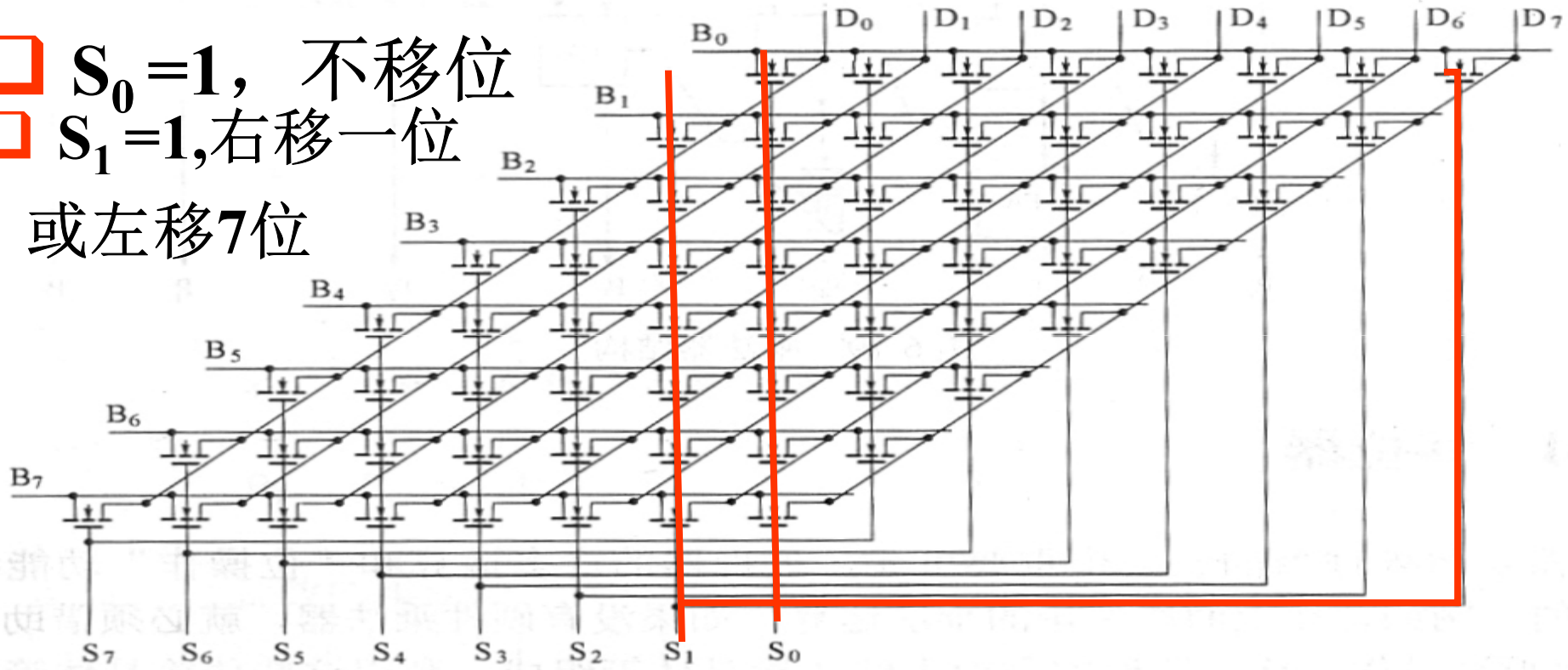


图 6-21　接成循环移位方式的桶型移位器

6.2.4 移位器移位关系列表

表 6-8 移位器移位关系列表

移位控制	$D_7\ D_6\ D_5\ D_4\ D_3\ D_2\ D_1\ D_0$	移位量
$S_0=1$	$B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$	不移位
$S_1=1$	$B_0\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1$	右移 1 位/左移 7 位
$S_2=1$	$B_1\ B_0\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2$	右移 2 位/左移 6 位
$S_3=1$	$B_2\ B_1\ B_0\ B_7\ B_6\ B_5\ B_4\ B_3$	右移 3 位/左移 5 位
$S_4=1$	$B_3\ B_2\ B_1\ B_0\ B_7\ B_6\ B_5\ B_4$	右移 4 位/左移 4 位
$S_5=1$	$B_4\ B_3\ B_2\ B_1\ B_0\ B_7\ B_6\ B_5$	右移 5 位/左移 3 位
$S_6=1$	$B_5\ B_4\ B_3\ B_2\ B_1\ B_0\ B_7\ B_6$	右移 6 位/左移 2 位
$S_7=1$	$B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0\ B_7$	右移 7 位/左移 1 位

- 对照图6-21练习

6.2.4 移位器

- 改变连接可实现：
  - 循环移位方式
  - “左移＋低位补0（补1）”
  - “右移＋高位补0（补1）”
- 移位寄存器不仅需要移位，而且要完成存储功能——寄存器

6.2.5 寄存器

在微处理器中，寄存器的形式多样，根据不同的需要和外围总线结构配合，设置和选择不同的结构形式

- 准静态寄存器
- 双港口寄存器
- 具有开漏晶体管的双港口寄存器
- 开漏结构寄存器的外部连接结构
- 静态存储单元双港口寄存器
- 移位寄存器
- 堆栈

wxhsnow@163.com

准静态寄存器

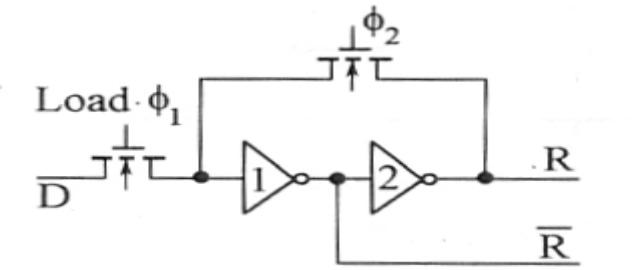


图 6-22　准静态寄存器单元

- 组成：两个倒相器　两个传输晶体管
- 控制：写入控制**Load**ϕ_1、ϕ_2不重叠两相时钟

$Load \cdot \phi_1$控制输入晶体管的打开

ϕ_2控制反馈传输晶体管的导通——锁存信号

准静态寄存器

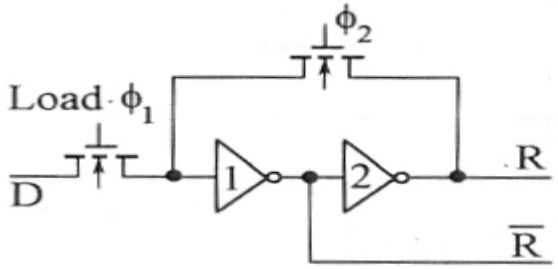


图 6-22　准静态寄存器单元

❑工作过程：

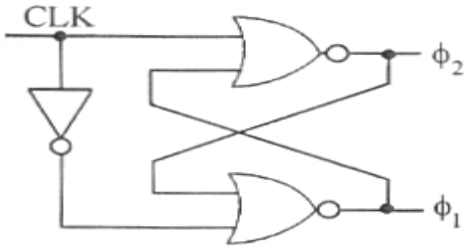
$Load \cdot \phi_1 = 1$时，输入传输晶体管导通，信号D通过传输晶体管、倒相器1、2到达输出端R

$\phi_1 = 0$，$\phi_2 = 1$时，传输晶体管截止，反馈晶体管导通，反馈传输晶体管、两个倒相器形成闭环——锁存信号

*由于ϕ_1、ϕ_2是两相不重叠时钟，从输入晶体管截止到反馈晶体管导通有一段延迟时间，此时存入信号依靠分布电容维持。若延迟时间过长，原先输入的信号可能会因为电容的漏电而丢失

两相不重叠时钟——R-S触发器法

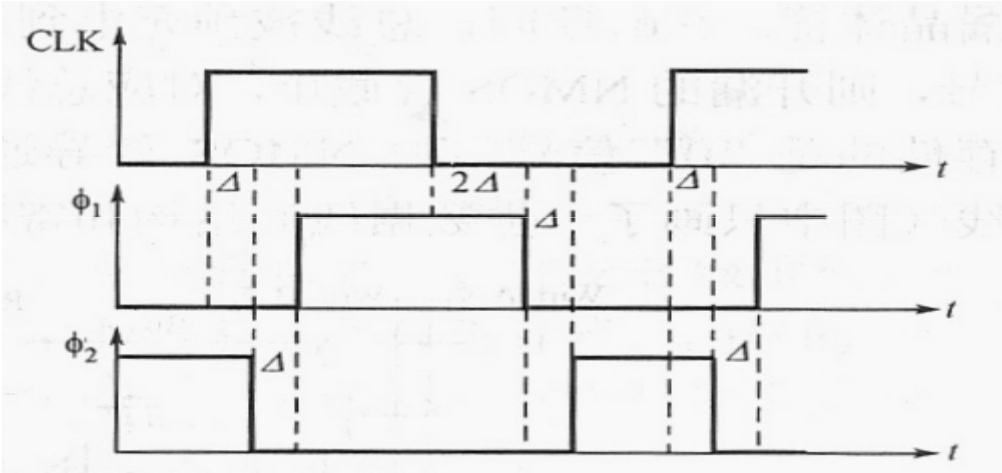
- 假设倒相器、或非门具有相同的时间延迟



初始状态：$CLK = 0$，$\phi_1 = 0$，$\phi_2 = 1$

$$\phi_1 = \overline{\overline{CLK} + \phi_2}$$

$$\phi_2 = \overline{CLK + \phi_1}$$



双港口寄存器

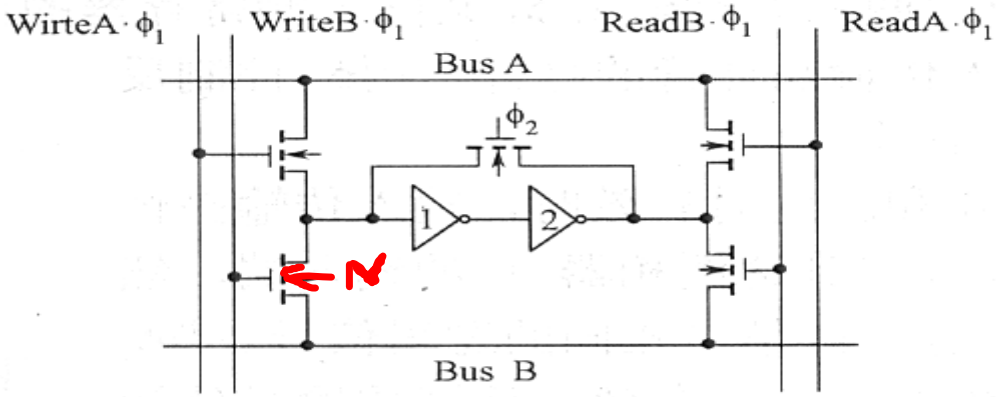


图 6-24　双港口寄存器

- 双港口寄存器：以准静态寄存器为核心
- 双港口：可通过两条数据总线存取数据
- 通过配合可同时从两组数据寄存器读出数据
  ——提高速度

wxhsnow@163.com

双港口寄存器

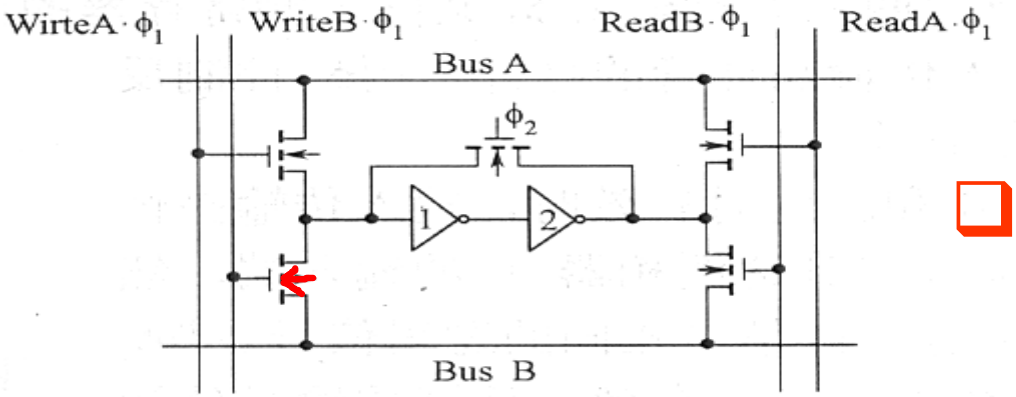


图 6-24　双港口寄存器

- B的工作过程类似

- 工作过程：

$WriteA \cdot \phi_1$有效时，数据从数据总线A存入寄存器

ϕ_2有效时，寄存器通过反馈传输晶体管锁存信号

$ReadA \cdot \phi_1$有效时，数据从寄存器读出到数据总线A

具有开漏晶体管的双港口寄存器

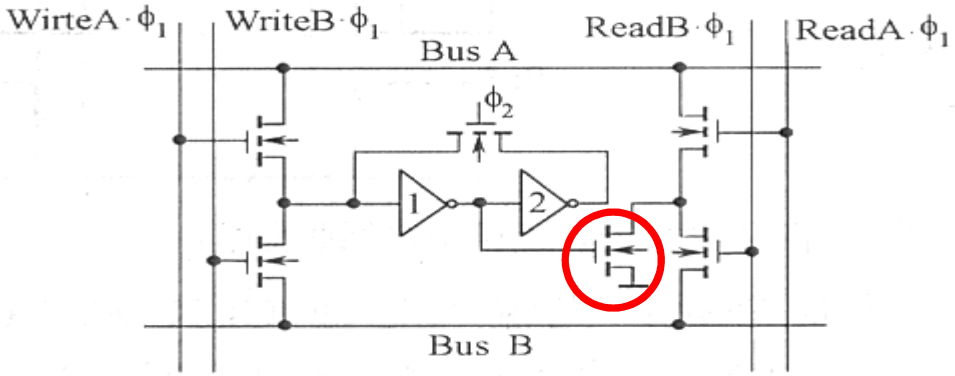


图 6-25　具有开漏晶体管的双港口寄存器

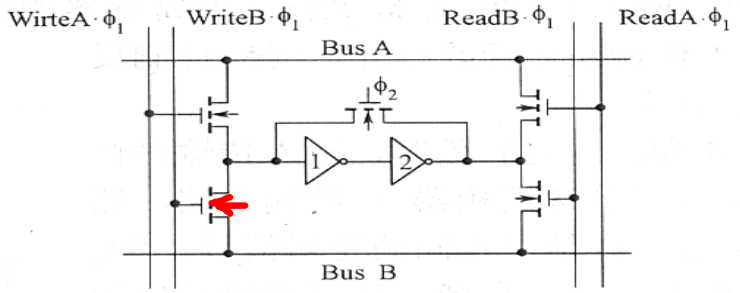


图 6-24　双港口寄存器

- 特点：采用下拉晶体管结构——开漏结构
- 作用：减轻寄存器单元的负载

  ——寄存器只需驱动开漏晶体管

具有开漏晶体管的双港口寄存器

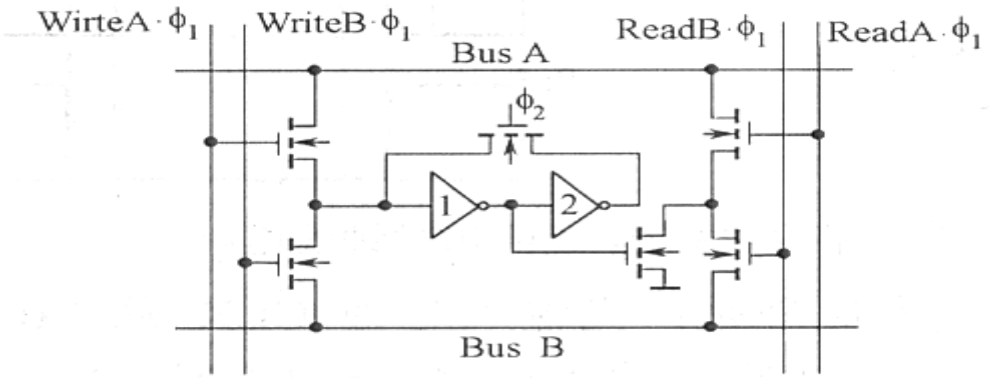


图 6-25　具有开漏晶体管的双港口寄存器

- 工作原理：

ϕ_2有效期间，总线被预充电到高电平

读出数据时：

*若输入信号为1，经倒相器1之后变为0，则开漏*NMOS*截止对应总线的某根位线上保持高电平

*若输入信号为0，经倒相器1后变为1，则开漏*NMOS*导通对应总线的某根位线被放电到低电平

开漏结构寄存器的外部连接结构

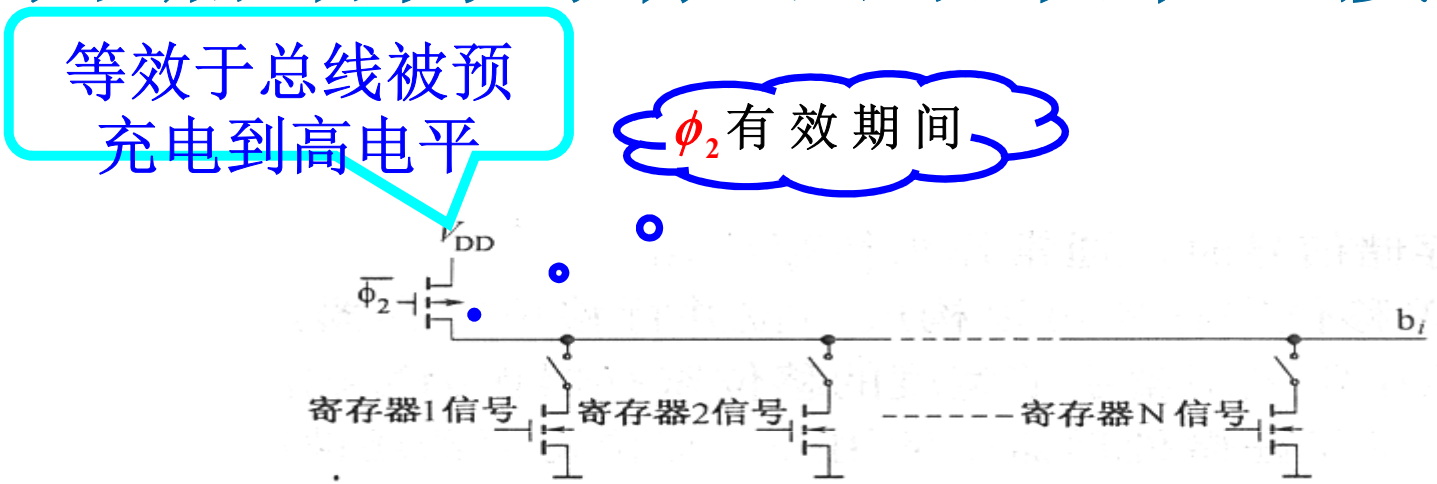


图 6-26　开漏结构寄存器的外部连接结构

- 在数据寄存器组织结构中，寄存器以字为基本单位
- 一个寄存器堆可能有若干个字，每个字的宽度（位数）通常是**8**的偶数倍

静态存储单元双港口寄存器

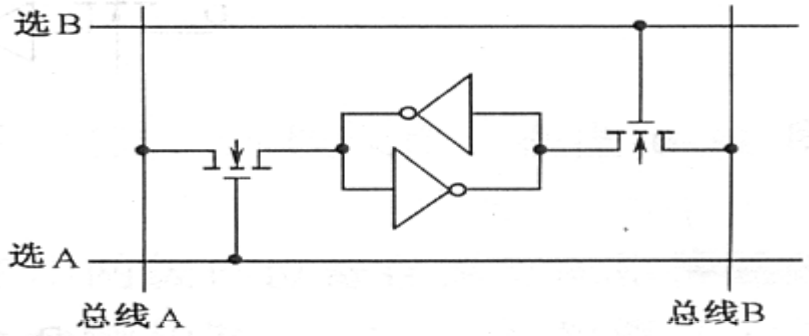


图 6-27　静态存储单元双港口寄存器

- 以静态寄存器为核心
- 两个倒相器连接成闭环结构锁存数据
- 数据流动可双向

- 在同一边写入、读出，读出的数据是原数据
- 在一条总线写入，另一条总线读出，原数据被倒相

移位寄存器

- 数字逻辑的常用单元——暂存数据
  - 采用先进先出移位方式
- 可用做延迟单元——延迟时间以时钟周期数计算
  - 数据在两相不重叠时钟的控制下进行移位

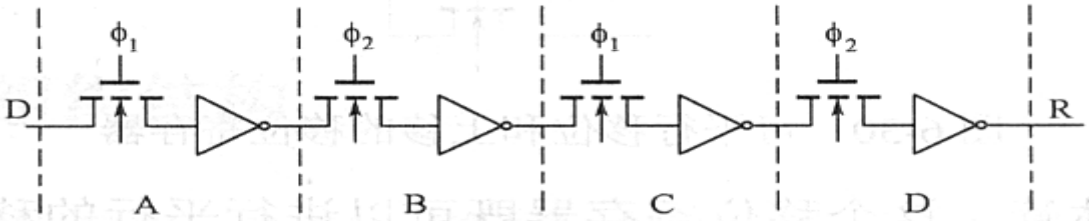


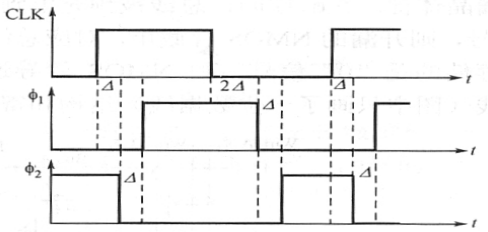
图 6-28　简单移位寄存器

采用两相不重叠信时钟控制，防止重叠时间内多级传送

$\phi_1=1$期间$\phi_2=0$，数据只能进入寄存器的A级

$\phi_2=1$期间$\phi_1=0$，数据继续进入寄存器的B级

N个周期，信号被移动2N级移位寄存器



多位并行移位寄存器

- 多位并行移位寄存器可同步移位若干数据字
- 微处理器中采用整字节宽度的移位寄存器

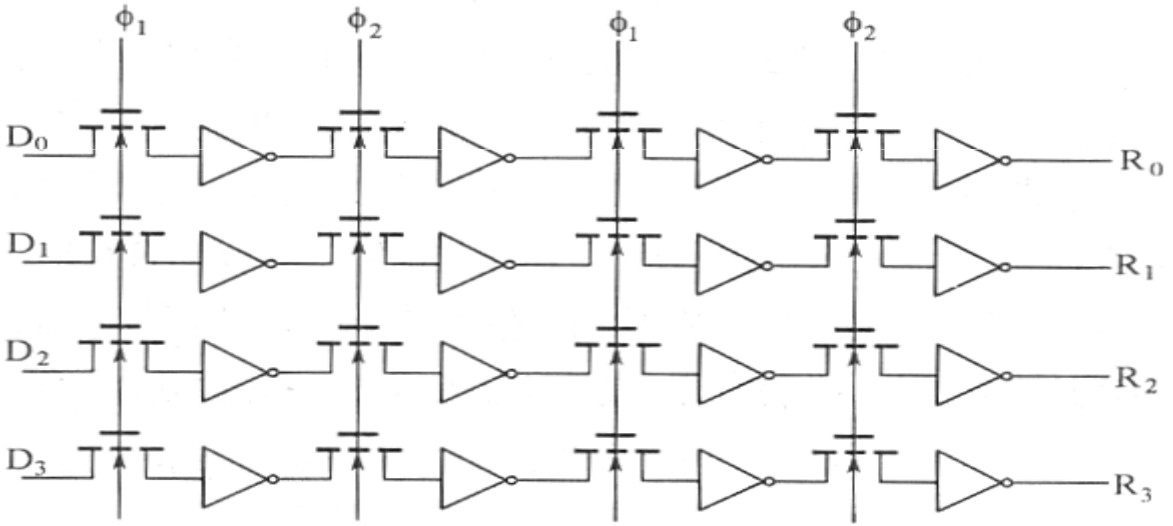


图 6-29　多位并行移位寄存器

移位寄存器

- 可平行移位和上移的移位寄存器

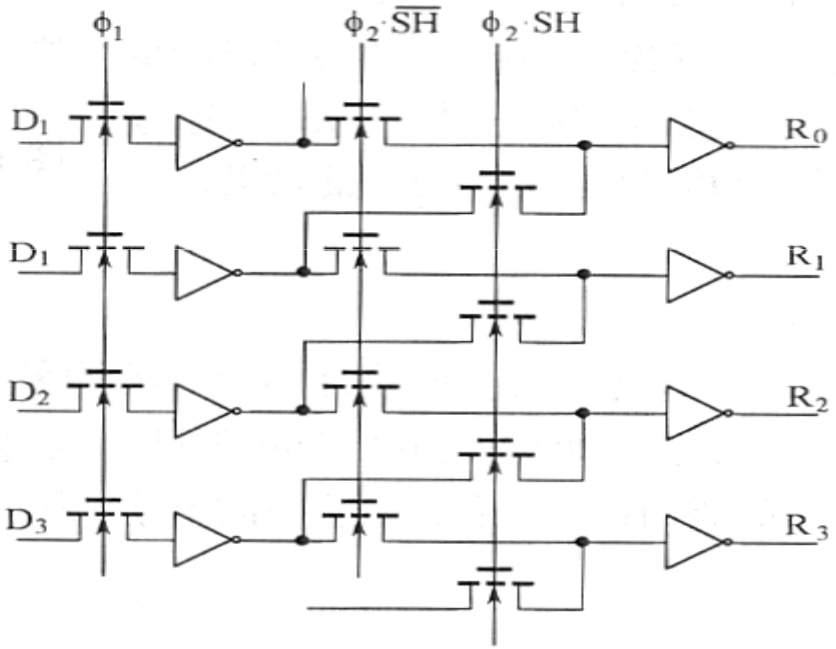


图 6-30　可平行移位和上移的移位寄存器

wxhsnow@163.com

6.2.6 堆栈

- 堆栈：采用先进后出的存储和移位结构
- 基本操作：压栈**PUSH**　弹出**POP**
  - 压栈：将数据存入堆栈

    此时前一次压入的数据往堆栈内部递进一位
  - 弹出：将原先存入堆栈的数据取出

    ——最靠近入口的数据——后进先出
- 堆栈的工作分为：压栈、保持、弹出

一位堆栈结构

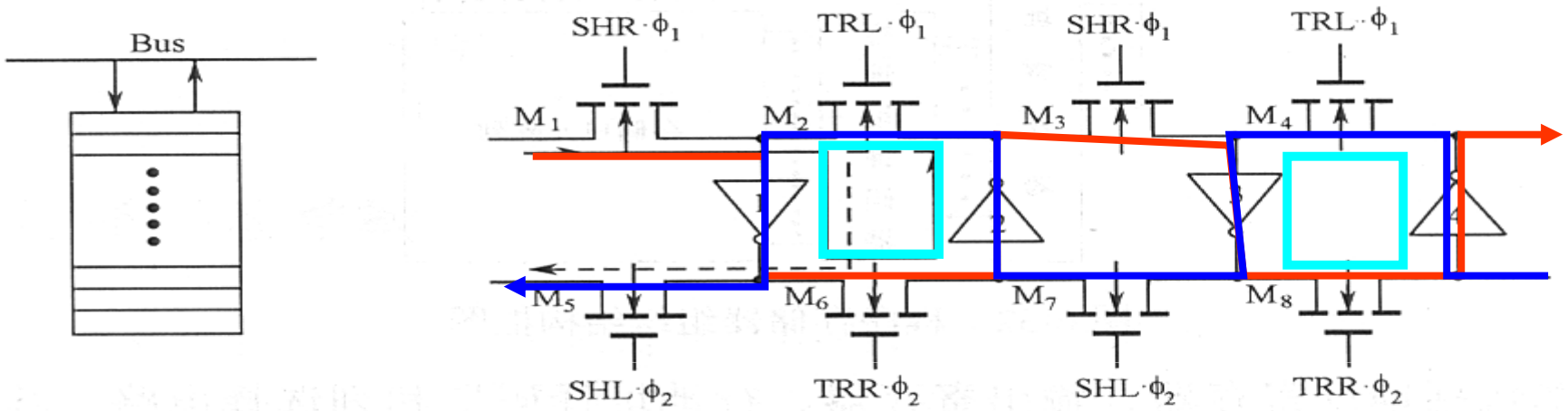


图 6-31　一位堆栈结构

压栈：SHR TRR有效时，ϕ_1、ϕ_2控制压栈

弹出：SHL TRL有效时，ϕ_1、ϕ_2控制下数据经M_5弹出

保持：TRR TRL有效时，ϕ_1、ϕ_2控制保持数据一闭环

6.3 存储器组织

- 6.3.1 存储器组织结构
- 6.3.2 行译码器结构
- 6.3.3 列选择电路结构

wxhsnow@163.com

存储器组织结构

- 存储器作用： 存储数据字、程序（指令）
- 存储器作为数据存入和读出的功能模块，包括两个主要的部分：
  - 记忆体（**ROM**、 **EEPROM**、 寄存器等）
  - 写入/读出控制逻辑

程序存储组织结构框图

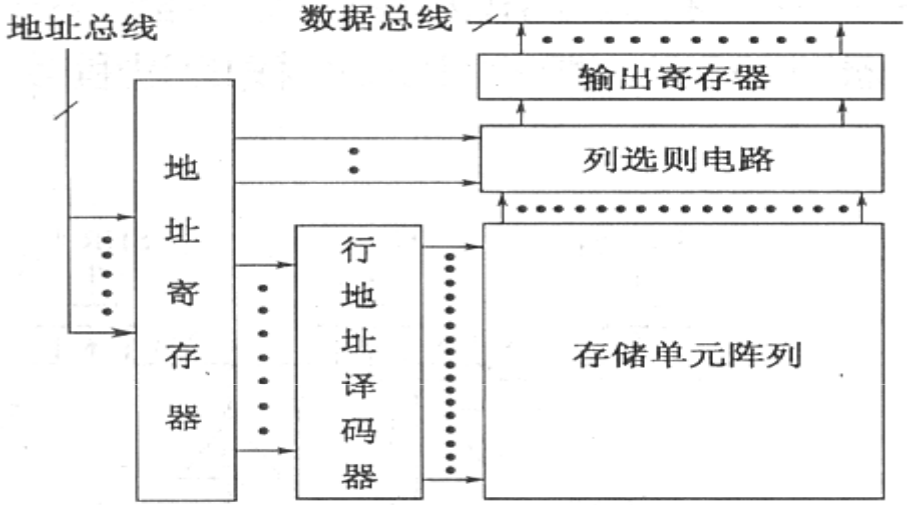


图 6-32　程序存储器组织结构框图

- 主记忆体——存储单元阵列
- 控制逻辑——数据选择读出电路

微处理器的程序存储器

- 存储单元阵列**ROM**
  - 与结构**ROM**（串联，限制，少用）
  - 或结构**ROM**
  - 与－或结构**ROM**

控制逻辑

- 控制逻辑包括地址寄存器、输出寄存器、行地址译码器和列选择电路
  - 地址寄存器:暂存由地址总线传送的所选存储单元的地址，同时隔离存储器与地址总线，防止在译码过程中总线上的信号变化对译码的影响（*准静态结构*）
  - 输出寄存器：暂存由存储器输出的数据字或指令字，同步的将输出字送到数据总线（*准静态结构*）
  - 行地址译码器和列选择电路：对大尺寸**ROM**采用行、列分段译码方式。

N位地址＝＝》2^N个地址值，覆盖全部存储空间

程序存储器组织结构框图

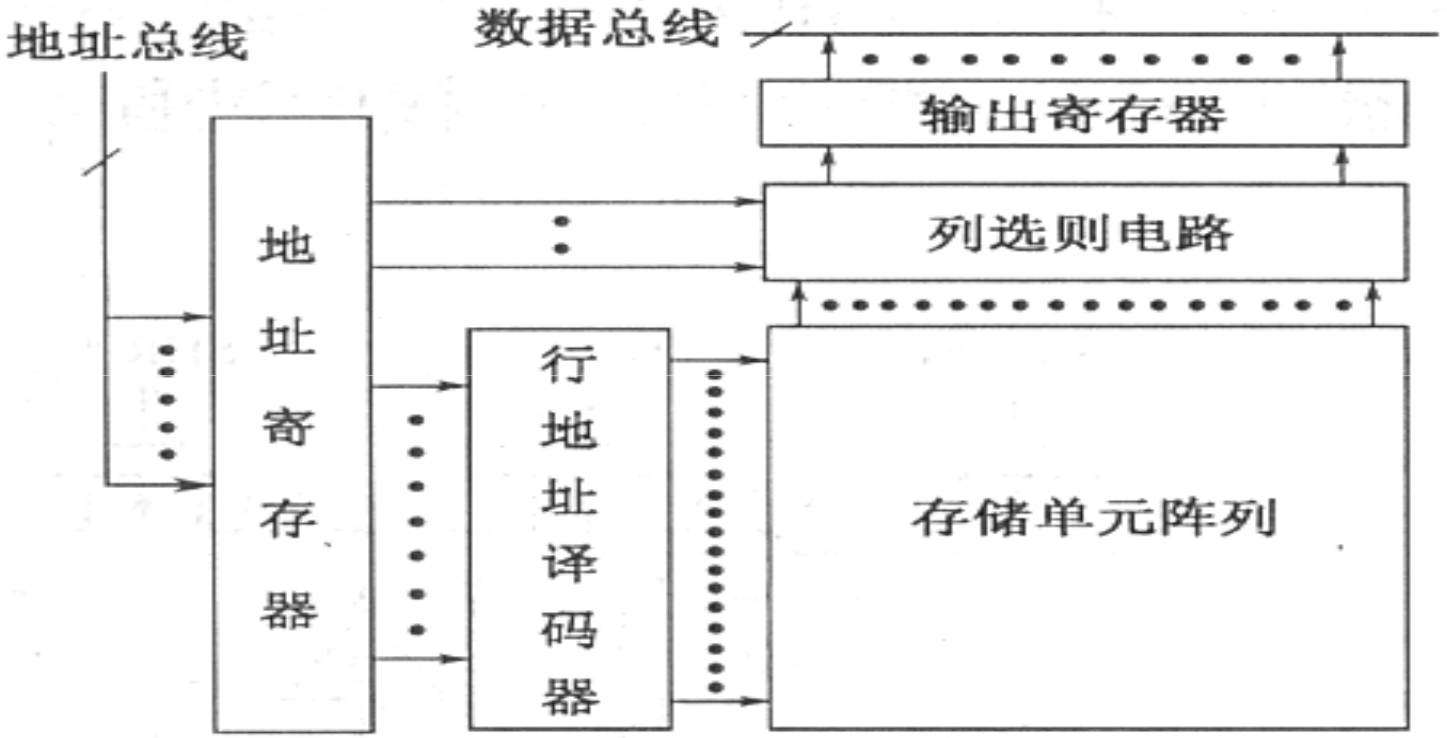


图 6-32　程序存储器组织结构框图

wxhsnow@163.com

16k存储器工作过程

- 16K-- >14位地址，一般分为两段：10位和4位
  - 行译码器：$A_0 \sim A_9$
  - 列译码器：$A_{10} \sim A_{13}$
- *行、列译码器负责的高低位可改变*
- 2^N个字高的瘦长组织结构？？？
- 简化译码器结构，节约存储器面积
  - 根据每个字的位数、字线（行译码器输出）的驱动能力、存储器的大小以及在芯片上的布局来确定地址的分段
  - 举例：

16k存储器存储空间读出顺序

- 存储空间读出顺序：
  - 高四位：0000－－》低十位：0000000000
    ～ 1111111111
  - 高四位：0001－－》低十位：0000000000
    ～ 1111111111

6.3.2 行译码器结构

- 行译码器的结构设计必须考虑**ROM** 的结构形式并与之匹配。

➢ 与结构**ROM**：

选中的字线(输出)为低电平,其他必须为高电平

➢ 或结构**ROM**：

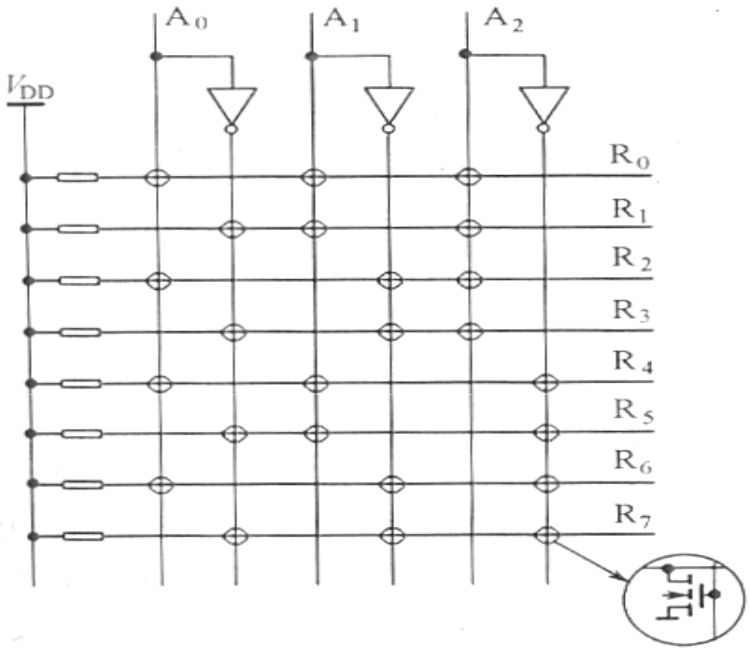
选中的字线(输出)为高电平,其他必须为低电平

➢ 与－或结构**ROM**：

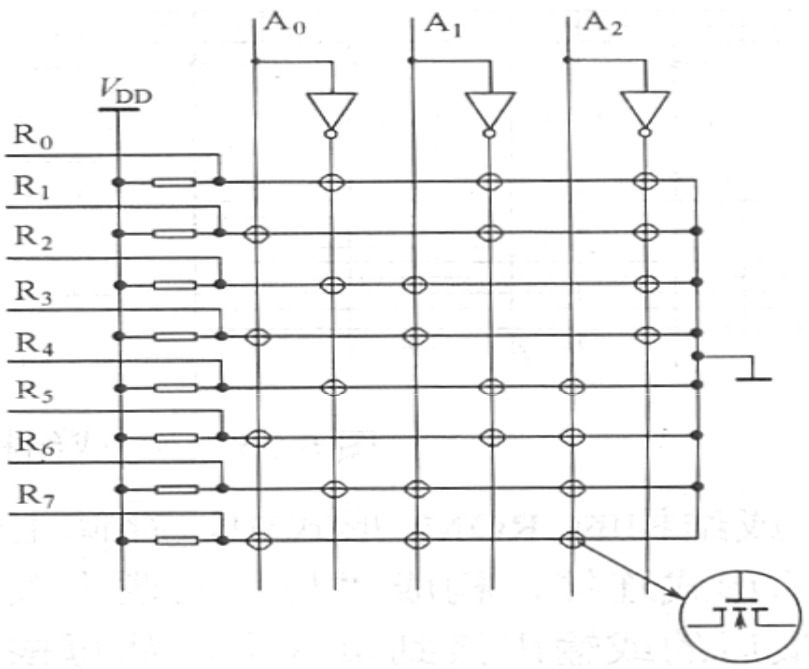
局部与结构遵循与结构的规则

整体或结构遵循或结构的规则

行译码器结构



(a) 或非结构



(b) 与非结构

- 注意选中字线的电平高低
- 负载管、NMOS管数量多

Thanks

wxhsnow@163.com