# 计算机组成原理

# 期末复习

# 课程介绍

- 课程名称
  - 计算机组成原理（ **Computer Organization**）
- 学时学分
  - 课堂教学：**64**学时**/4**学分
- 课程目的
  - 覆盖了传统的数字逻辑和计算机组成原理**2**门课程的知识。
  - 从原理性的角度出发，以**MIPS**系统为主要研究对象，讲述计算机硬件系统的组成、各部件的结构及其底层硬件工作原理，使学生理解计算机的组织与结构和工作过程，掌握计算机硬件系统的基本设计方法，培养学生分析、设计和开发计算机硬件系统的基本能力，为后续课程打下坚实基础。

- 总体原则：相比往年适当降低试题难度
- 试题类型
  - 选择题 10题10分
  - 简答题 每题5分，共4题
    - 逻辑函数化简
    - I/O
    - 汇编语言
    - VM
  - 大题（综合分析或设计）
    - 组合逻辑、时序逻辑、主存储器、Cache、汇编（15分）、CPU设计（流水线、转发，15分）
- 分数分布
  - 数字逻辑部分： ~30
  - 汇编语言部分：~20
  - 组成原理部分：~50
- 把所有的作业和PPT的例题掌握（包括期中考试的题目）

# 第一讲：计算机组成概述（4学时）

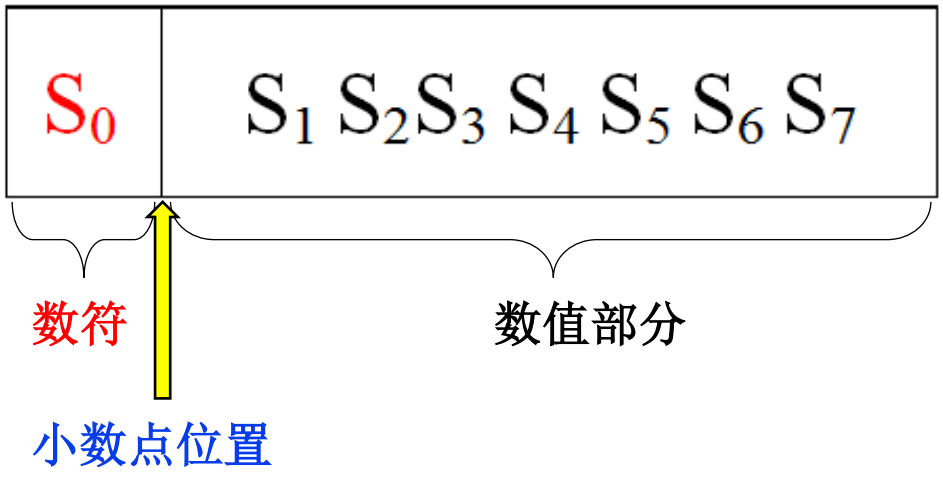
- 目　　标
  - 了解计算机系统的基本功能、组成框架、典型结构及层次关系，掌握计算机中数的表示方法及常用编码。
- 主要内容
  - 计算机系统的基本组成
  - 计算机系统的典型架构与层次关系
  - 计算机中数的表示
    - 定点数的表示（原码、反码、补码）
    - 浮点数的表示
    - 其他编码（格雷码、循环码、ASCII码、汉字编码）
  - 计算机的程序执行原理简介
    - 指令的含义简介
    - 程序的执行过程简介

# 2.1 无符号数和有符号数

## ➢定点小数

如：01100000

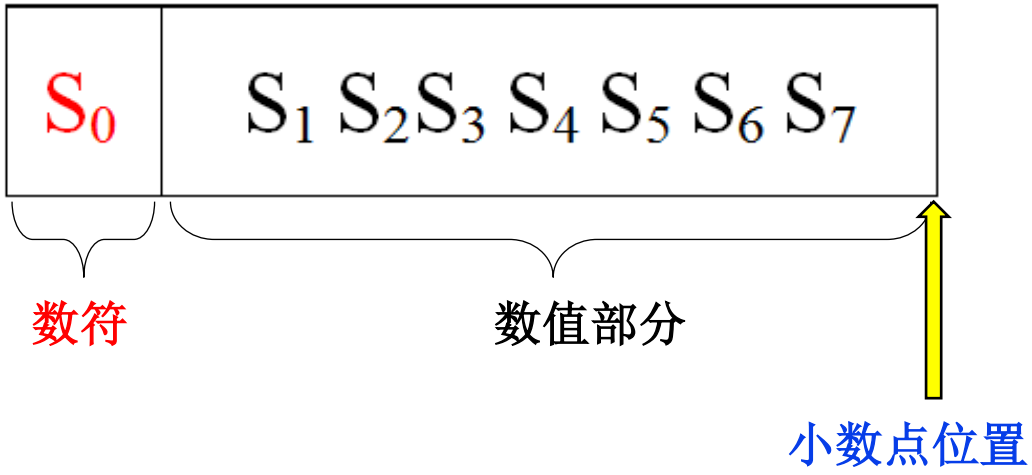
是十进制的0.75




## ➢定点整数

如：01100000

是十进制的192



# 2.2 定点数（定点整数与定点小数）

- 原码
- 反码
- 补码
- 移码

N 位定点整数的原码、反码、补码和移码

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

$$[x]_{反}=\begin{cases}x & 0\le x\le 2^{n-1}-1\\ (2^{n}-1)+x & -(2^{n-1}-1)\le x\le 0\end{cases}$$

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

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

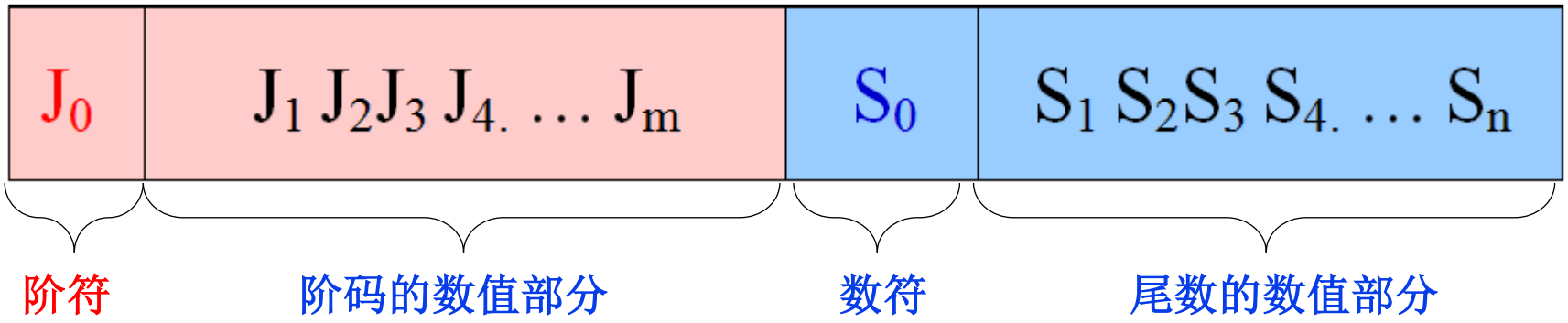
# 2.3 浮点数表示

- 浮点数的一般表示法：分为阶码和尾数两个部分
  - 阶码：采用定点整数表示
  - 尾数：采用定点小数表示
  - 例：$(178.125)_{10}=(10110010.001)_2$

$$=0.10110010001\times 2^{01000}$$

阶码：**01000**

尾数：**0.10110010001**



# 第二讲：组合逻辑设计（8学时）

- 目　　标
  - 了解门电路的基本结构，掌握布尔代数的理论及其门电路实现方法，进而掌握布尔方程表示、转换及化简等方法，以及运算单元、译码器等基本组合逻辑部件设计方法，学习并掌握Verilog HDL。
- 主要内容
  - 逻辑门电路（1学时）
    - 非门、与门、或门、复合逻辑门电路及其性能指标
    - TTL、MOS集成门电路
  - 布尔代数原理及其门电路实现（2学时）
    - 布尔代数基本原理
    - 布尔代数的门电路实现
  - 基本组合逻辑部件设计（3学时）
    - 运算单元电路（加法器、比较器、函数发生器）
    - 多路选择器，译码器，编码器
  - Verilog HDL介绍（2学时）

# 半导体二极管的开关特性

- 半导体器件的开关特性：有导通和截止两种状态，导通状态下允许电信号通过，截止状态下禁止电信号通过。
  - 静态特性（稳态开关特性）：器件稳定在导通和截止两种状态下的特性；

- 二极管单向导电性：外加正向开启电压导通，反向电压截止——受外加电压极性控制的开关
- 二极管的正向开启电压：锗管约为0.2-0.5V；硅管为0.5-0.7V。如无特殊说明，本章中默认为0.7V。
- 当加在二极管上的电压$U_D<V_D$（0.7V）时，二极管截止，电流$I_D=0$；当$U_D>V_D$（0.7V）时，二极管导通，而且一旦导通，则$U_D=V_D$（0.7V）不变。因此，$V_D$称为钳位电压。
- 当二极管的反向电压超过一个阈值($V_Z$)时，二极管会被击穿，此时二极管上的压降是$V_Z$。

# 晶体三极管的开关特性

- 在模拟电路中，晶体三极管主要作为线性放大元件和非线性元件；在数字电路中，主要作为开关元件。
- 晶体管共发射极电路放大能力强，也即控制能力强，只要在输入端加上两种不同幅值的信号，就可以控制晶体管的导通或截止。
- 作为开关电路，晶体三极管主要工作在截止区和饱和区。
- 三极管的稳态开关特性是指三极管稳定在截止和饱和导通两种状态下的特性。

# 晶体三极管三个工作区的特点总结

**放大区：**

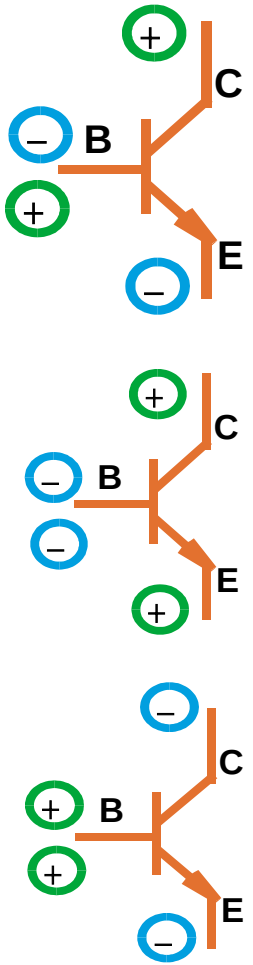
- 发射结正偏，集电结反偏
- 有电流放大作用，$I_C=\beta I_B$
- 输出曲线具有恒流特性

**截止区：**

- 发射结、集电结处于反偏
- 失去电流放大作用，$I_C\approx 0$
- 晶体管C、E之间相当于开路

**饱和区：**

- 发射结、集电结处于正偏
- 失去电流放大作用，$I_C=I_{CS}$，不变
- 晶体管C、E之间相当于短路



# 逻辑代数及其逻辑符号

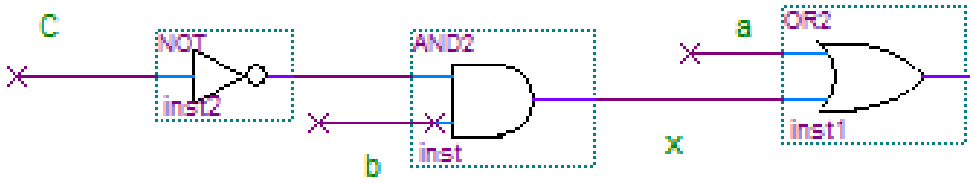
- 与逻辑
- 或逻辑
- 非逻辑
- 与或逻辑
- 与或非逻辑
- 异或逻辑
- 同或逻辑

# 逻辑函数表达式

- 逻辑代数的基本定理
  - 代入定理
  - 反演定理
  - 对偶定理
- 化简的常用公式
  - 四个常用公式
  - 吸收率1、2、3
  - 包含律
- 逻辑函数的常用表达式推导法
  - 最小项推导法
  - 最大项推导法
- 卡诺图的化简

# Verilog HDL模块的结构

- Verilog的基本设计单元是“模块（module）”，实现一个特定功能
- 一个“与门”、“加法器”、ALU都可以是一个模块
- Verilog模块的结构由在module和endmodule关键词之间的4个主要部分组成：




1. 端口定义
2. I/O说明
3. 信号类型声明
4. 功能描述

```
module block1(a,b,c,d );
    input a, b, c;  /* I/O变量缺省为wire变量*/
    output d;
    wire x;
    assign d = a | x; //组合逻辑功能描述
    assign x = ( b & ~c );
endmodule
```

➢结构描述

# Verilog HDL模块

❖ Verilog HDL模块的模板（仅考虑用于逻辑综合的程序）

```
module <顶层模块名> (< 输入输出端口列表>) ;
   output 输出端口列表;
   input 输入端口列表;
//（1）使用assign语句定义逻辑功能
   wire <结果信号名> ;
   assign <结果信号名> = 表达式 ;
//（2）使用always块定义逻辑功能
   always @(<敏感信号表达式>)
      begin
         //过程赋值语句
         //if语句
         //case语句
         //while,repeat,for循环语句
         //task,function调用
      end
```

# 组合逻辑电路的结构和特点

- 按照逻辑功能的不同特点，数字电路分为两大类：
  - 组合逻辑电路和时序逻辑电路
- 组合逻辑电路是将逻辑门以一定的方式组合在一起，使其具有一定逻辑功能的数字电路。
- 它是一种无记忆电路——任一时刻的输出信号仅取决于该时刻的输入信号，而与信号作用前电路原来所处的状态无关。


- 组合逻辑电路的特点
  - 由逻辑门电路组成
  - 输出不能再直接反馈到输入（不能有环路）和存储电路
  - 当时的输出仅由当时的输入决定——速度快

# 编码器

- 为了区分一系列不同的事物，将其中的每个事物用一组二值（0或1）代码表示；或者说，用二进制代码来表示特定信息——编码的含义。
- 将加在电路若干输入端中的某一个输入端的信号变换成相应的一组二进制代码输出的过程叫做编码。
- 实现编码功能的数字电路称为编码器（Encoder）。
- 编码器的作用是将某一时刻仅一个输入有效的多个输入的变量情况用较少的输出状态组合来表达，或者说将输入的每一个高、低电平信号编成一组对应的二进制代码，以便于后续的识别和处理。
- 通常有二进制编码器、BCD码编码器及优先编码器。

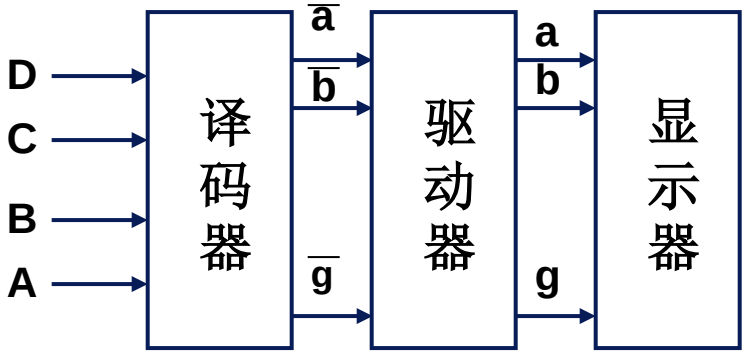
# 译码器

- 将二进制代码所表示的信息翻译成对应输出的高低电平信号的过程称为译码，译码是编码的反操作。实现译码功能的电路称为译码器（Decoder）。
- 常用的译码器有变量译码器、码制变换译码器和显示译码器。
  - 变量译码器（二进制译码器）是用来表示输入变量状态全部组合的译码器。n个输入代码有$2^n$个状态，因此n位二进制译码器有n个输入端和$2^n$个输出端，一般称为n线-$2^n$线译码器。常用的有双2线-4线译码器74××139，3线-8线译码器74××138，4线-16线译码器74××154等。
  - 码制变换译码器是将输入的某个进制代码转换成对应的其他码制输出的译码器。如二-十进制码（8421码）至十进制码译码器（简称BCD译码器）、余3码至十进制码译码器、余3循环码至十进制码译码器等。
  - 显示译码器是将输入代码转换成驱动7段数码显示器各段的电平信号的译码器。常用的有74××47（低电平输出有效）、74××49（高电平输出有效）、74××48（高电平输出有效）等。

# 显示译码器

- 显示译码器用于驱动数码显示器，是一种将二进制代码表示的数字、文字、符号用人们习惯的形式直观显示出来的电路。

（1）电路结构（8421BCD译码显示电路）



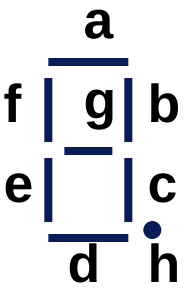

① 显示器件

辉光数码管、7段荧光数码管、液晶显示器

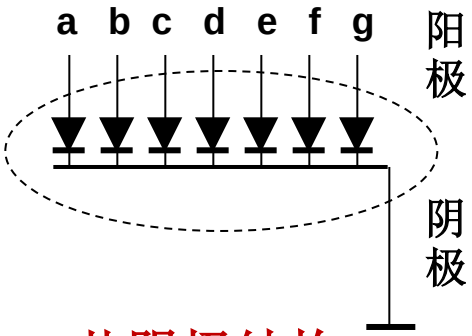
目前广泛使用的显示数字的器件是7段数码显示器（由7段可发光的线段拼合而成），包括半导体数码显示器和液晶显示器两种。

数码显示器人们通常又称为数码管。半导体7段数码管实际上是由7个发光二极管（LED）构成的，因此又称为LED数码管。

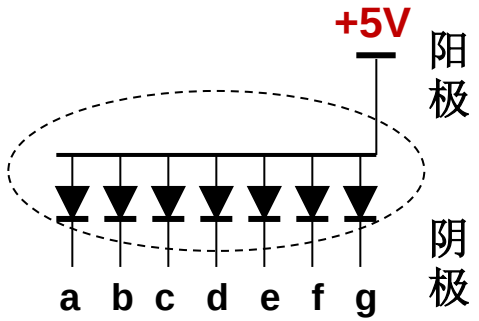
# 半导体7段数码管




半导体7段数码管




共阴极结构



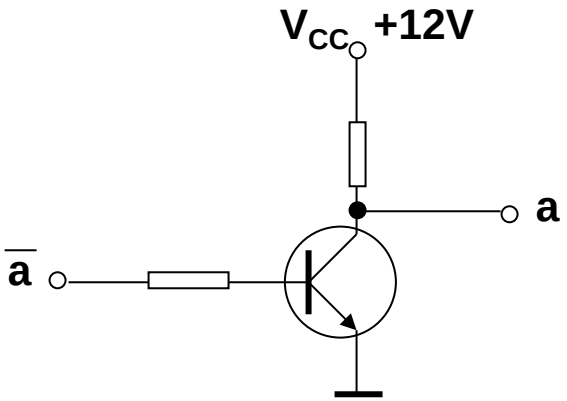

共阳极结构

- 若使用共阴极LED数码管，则显示译码器的输出应为高电平输出有效；若使用共阳极LED数码管，则译码器应为低电平输出有效。

# 驱动电路和译码器

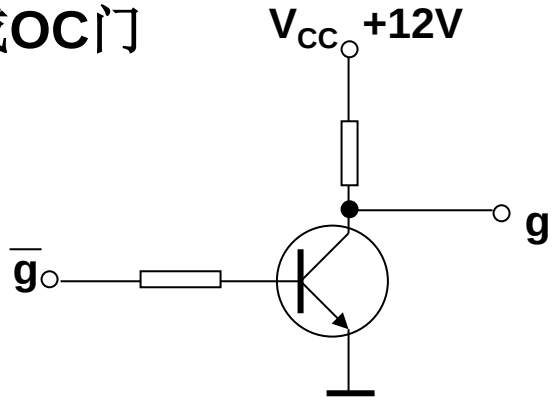
## ② 驱动电路

使译码器输出反相，并提供大的驱动电流。




反相器

或OC门




## ③ 译码器（共阳器件）

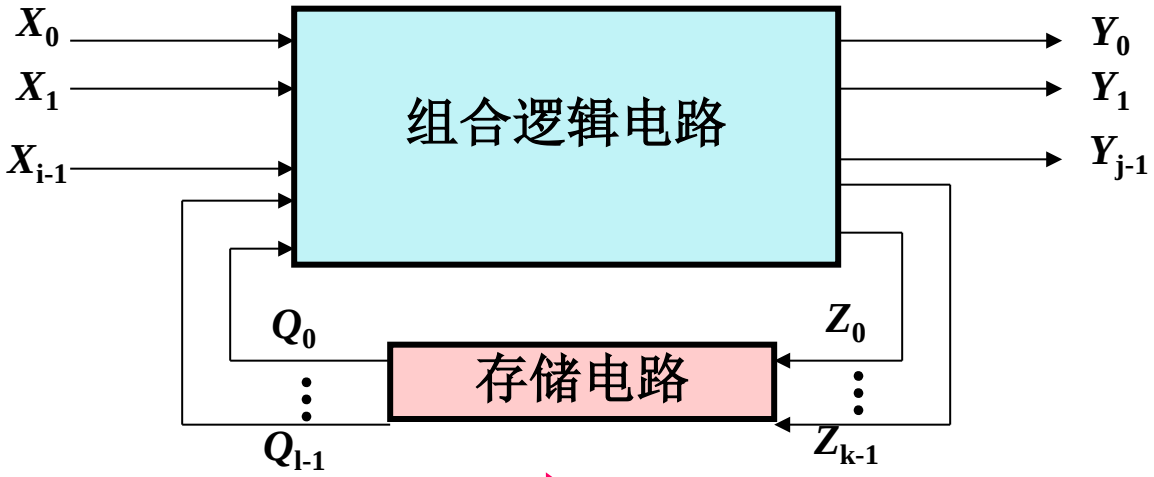
| D C B A | $\overline{a}$ $\overline{b}$ $\overline{c}$ $\overline{d}$ $\overline{e}$ $\overline{f}$ $\overline{g}$ | 显示数字 |
|---|---|---|
| 0 0 0 0 | 0 0 0 0 0 0 1 | 0 |
| 0 0 0 1 | 1 0 0 1 1 1 1 | 1 |
| 0 0 1 0 | 0 0 1 0 0 1 0 | 2 |
| 0 0 1 1 | 0 0 0 0 1 1 0 | 3 |
| 0 1 0 0 | 1 0 0 1 1 0 0 | 4 |
| 0 1 0 1 | 0 1 0 0 1 0 0 | 5 |
| 0 1 1 0 | 1 1 0 0 0 0 0 | 6 |
| 0 1 1 1 | 0 0 0 1 1 1 1 | 7 |
| 1 0 0 0 | 0 0 0 0 0 0 0 | 8 |
| 1 0 0 1 | 0 0 0 1 1 0 0 | 9 |
| 1 0 1 0 | X X X X X X X | |
| 1 0 1 1 | X X X X X X X | |
| 1 1 0 0 | X X X X X X X | |
| 1 1 0 1 | X X X X X X X | |
| 1 1 1 0 | X X X X X X X | |
| 1 1 1 1 | X X X X X X X | |

# 第三讲：时序逻辑设计（8学时）

- 目　　标
  - 掌握触发器、寄存器的结构和工作原理，掌握有限状态机、同步时序逻辑电路的设计方法和分析方法，具备使用仿真工具开发时序逻辑电路的能力。
- 主要内容
  - 锁存器和触发器（2学时）
    - SR锁存器、D锁存器
    - D触发器，JK触发器
    - 基于D触发器的寄存器构造
  - 有限状态机（FSM）　（2学时）
    - Moore型FSM
    - Mealy型FSM
  - 时序逻辑电路设计分析（4学时）
    - 数据寄存器
    - 移位寄存器
    - 计数器

# 时序逻辑电路的特点

- 当时的输出由当时的输入与电路的原来状态决定——具有“记忆”功能
- 结构特点：由组合逻辑电路和存储电路构成




触发器（Flip-Flop，FF）或寄存器

# 钟控JK触发器的逻辑功能表示

## ❖ 电路功能

**CP=0时为保持功能**

**特性表(CP=1)**

| J K $Q^n$ | $Q^{n+1}$ | 功能 |
|---|---|---|
| 0 0 0 | 0 | 保持 |
| 0 0 1 | 1 | |
| 0 1 0 | 0 | 置0 |
| 0 1 1 | 0 | |
| 1 0 0 | 1 | 置1 |
| 1 0 1 | 1 | |
| 1 1 0 | 1 | 翻转 |
| 1 1 1 | 0 | (计数) |

$$Q^{n+1} = \overline{J}\,\overline{K}Q^n + J\,\overline{K}\,\overline{Q}^n + J\,\overline{K}Q^n + JK\,\overline{Q}^n$$
$$= (J\,\overline{K}\,\overline{Q}^n + JK\,\overline{Q}^n) + (\overline{J}\,\overline{K}Q^n + J\,\overline{K}Q^n)$$
$$= J\,\overline{Q}^n + \overline{K}Q^n$$

**简化特性表(CP=1)**

| J K | $Q^{n+1}$ | 功能 |
|---|---|---|
| 0 0 | $Q^n$ | 保持 |
| 0 1 | 0 | 置0 |
| 1 0 | 1 | 置1 |
| 1 1 | $\overline{Q}^n$ | 翻转 |

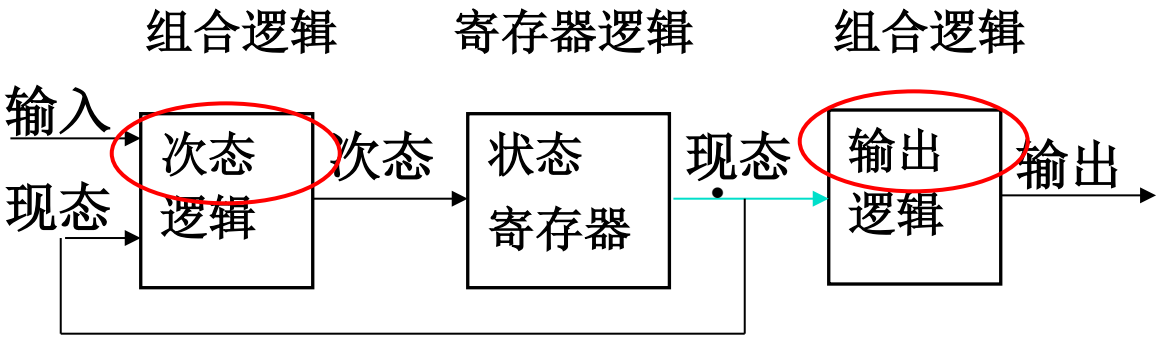
**特性方程**

$$Q^{n+1} = J\overline{Q^n} + \overline{K}\cdot Q^n$$

- **J0K0，输出不变； J0K1，输出为0；**
- **J1K0，输出为1；J1K1，分频计数**

# 有限状态机概述

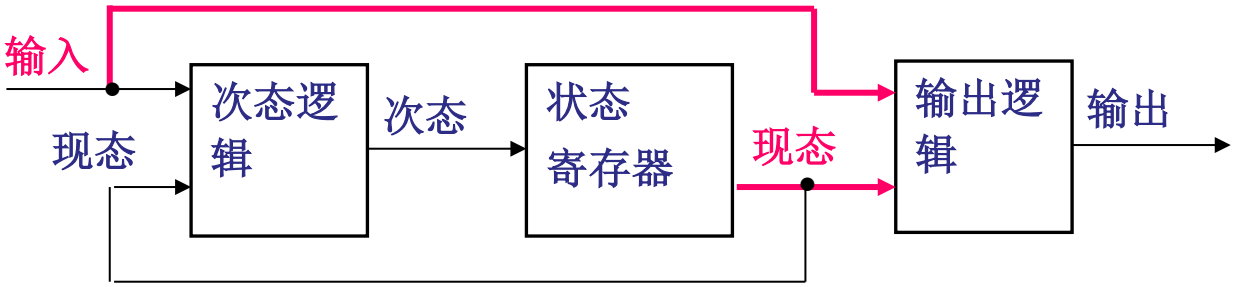
- 有限状态机（Finite State Machine，FSM）是表示有限个状态以及这些状态之间的转移和动作等行为的离散数学模型。
- 有限状态机是组合逻辑和寄存器逻辑的特殊组合。组合逻辑部分包括次态逻辑和输出逻辑，分别用于状态译码和产生输出信号；寄存器逻辑部分用于存储状态。




Moore型状态机典型结构

# 有限状态机的分类

- FSM常用于时序逻辑电路设计，尤其适于设计数字系统的控制模块。具有速度快、结构简单、可靠性高、逻辑清晰、复杂问题简单化的优点。
- 根据输出信号产生的机理不同，状态机可以分成两类：
  - 摩尔(Moore)型状态机--输出信号仅与当前状态有关
  - 米里(Mealy)型状态机--输出信号与当前状态及输入信号有关




Mealy型状态机的典型结构

# 有限状态机的设计方法

- 实用的状态机一般都设计为同步时序逻辑电路，它在同一个时钟信号的触发下，完成各状态之间的转移。
- 状态机设计步骤：

  1. 分析设计要求，列出全部可能状态；
  2. 画出状态转移图；
  3. 用Verilog HDL语言描述状态机，主要采用always块语句，完成3项任务：

  （1）定义起始状态（敏感信号为时钟和复位信号）；

  （2）用case或if-else语句描述出状态的转移（根据现态和输入产生次态）；

  （3）用case或if-else语句描述状态机的输出信号（敏感信号为现态）。

# Moore型有限状态机设计举例

【例2】设计一个序列检测器。要求检测器连续收到串行码{1101}后，输出检测标志为1，否则输出检测标志为0。

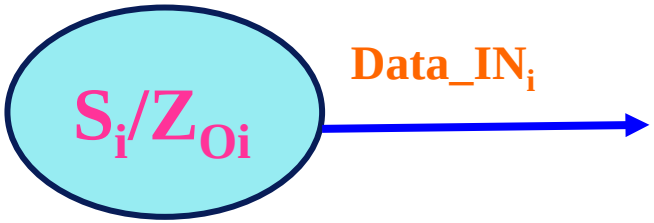
第1步：分析设计要求，列出全部可能状态：

| | |
|---|---|
| 未收到一个有效位（0） | ：S0 |
| 收到一个有效位（1） | ：S1 |
| 连续收到两个有效位（11） | ：S2 |
| 连续收到三个有效位（110） | ：S3 |
| 连续收到四个有效位（1101） | ：S4 |

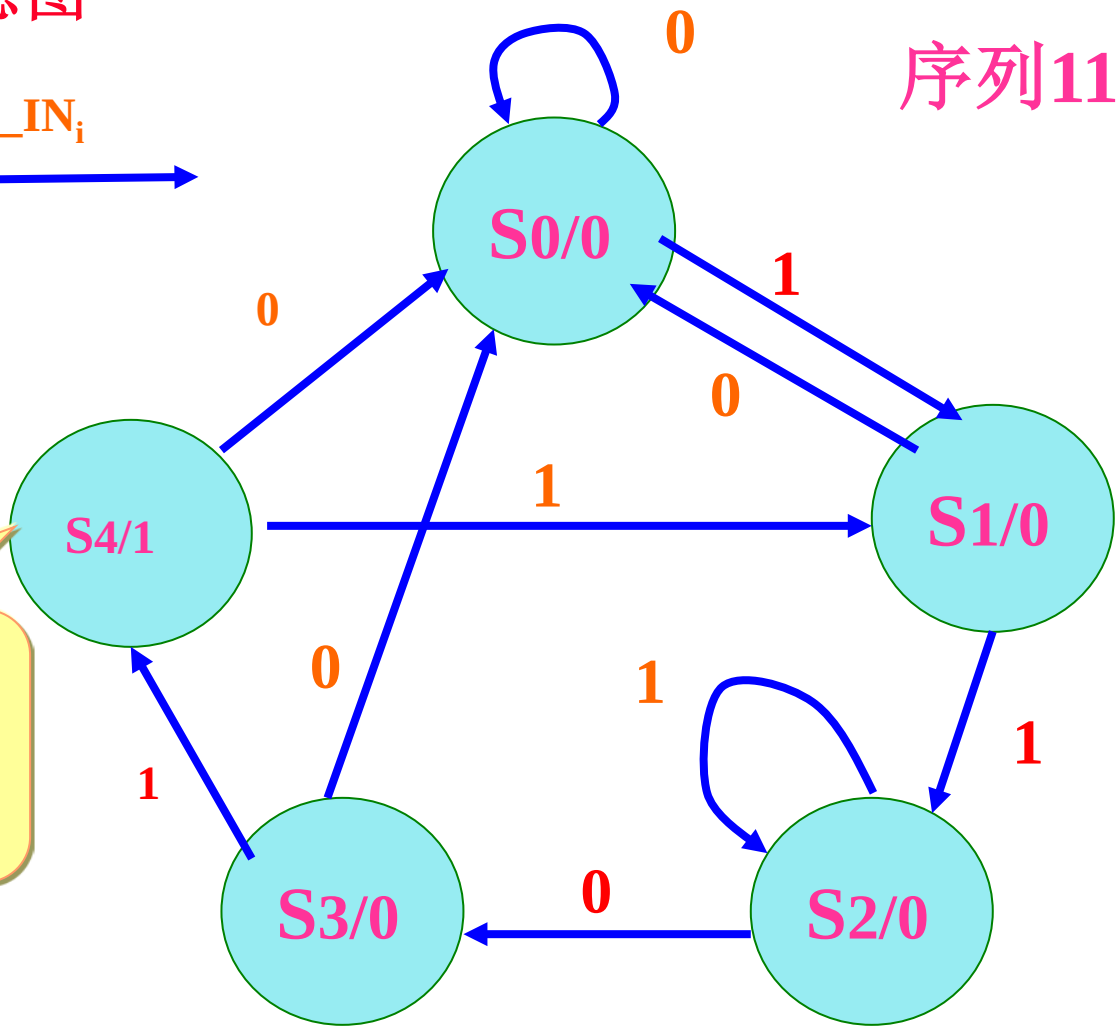
- 由于序列检测器的输出只为状态机当前状态的函数，而与外部输入无关，所以为Moore型状态机

# Moore型有限状态机设计举例

## 第2步：画出状态图




序列1101




只有在S4状态下输出zo置1，其他状态下zo为0，且输出与外部输入无关

# Moore型有限状态机设计举例

## 第3步：用Verilog语言描述状态机

- 在程序的开头定义状态机状态的编码形式
  - 用parameter或｀define语句
- 复位时回到起始状态
  - 敏感信号为时钟和复位信号
- 状态转换描述
  - 用case或if-else语句描述出状态的转移（根据现态和输入产生次态，可与复位时回到起始状态的语句放在同一个always块中，即敏感信号为时钟和复位信号）
- 输出信号描述
  - 用case语句（Mealy型状态机还要用到if-else语句）描述状态机的输出信号（单独放在一个always块中，敏感信号为现态）

# 序列检测器源程序（ Moore型状态机）

```
module monitor(clk,clr,data,zo,state);
   parameter S0=3'b000, S1=3'b001,
   S2=3'b010,S3=3'b011,S4=3'b100; //状态编码的定义
   input clk,clr,data;
   output zo;
   output[2:0] state;                                    //状态机
   reg [2:0] state;
   reg zo;
   always @(posedge clk or posedge clr)
     begin
        if (clr) state=S0; //（1）复位时回到初始状态
        else
          begin
            case (state)//  （2）状态的转移
              S0: if (data==1'b1) state=S1;
                    else state=S0;
               S1: if (data==1'b1) state=S2;
                    else state=S0;
              S2: if (data==1'b0) state=S3;
                    else state=S2;
               S3: if (data==1'b1) state=S4;
                   else state=S0;
              S4: if (data==1'b1) state=S1;
                    else state=S0;
               default: state=S0;
            endcase
            zo=(state==S4)?1'b1:1'b0;  //（3）状态机的输出信号
          end
     end
endmodule
```

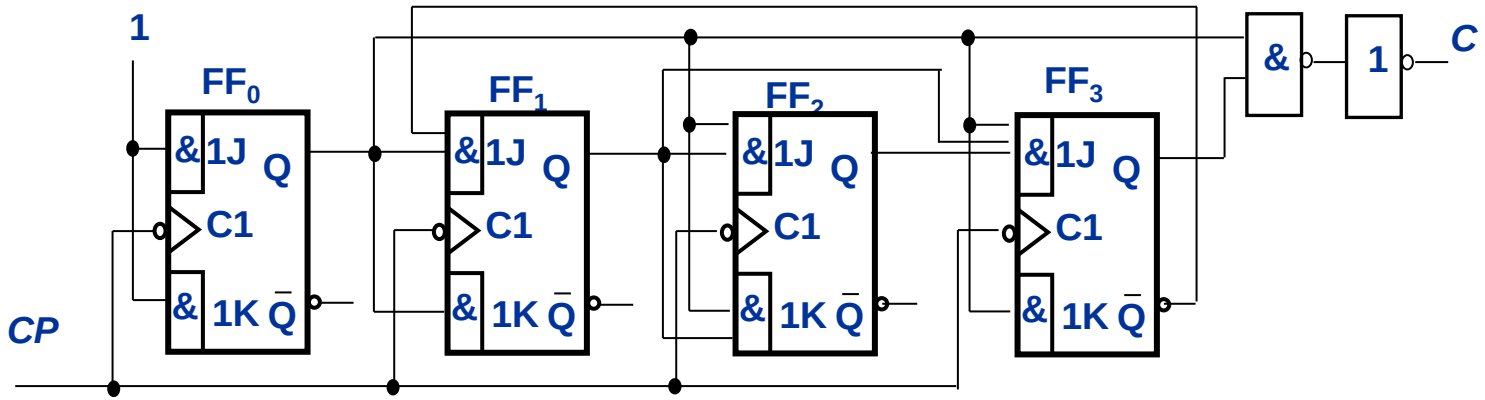
在S4时给zo置1——输出只为状态机当前状态的函数，而与外部输入无关

# 移位寄存器（移存器）

- 在计算机中，常要求寄存器有“移位”功能。例如，移位相加乘法器进行乘法运算时，要求将乘数右移，被乘数左移；除法运算时，要求将余数左移；将并行传递的数转换成串行数据以及将串行传递的数转换成并行数据的过程中，需要移位。
- 具有移位功能的寄存器称为移位寄存器，每来一个时钟脉冲，寄存器中数据就依次向左或向右移一位。
- 分类
  - 左移移位寄存器，右移移位寄存器，双向移位寄存器
- 数据输入方式
  - 串行输入，并行输入
- 数据输出方式
  - 串行输出：右移寄存器、左移寄存器
  - 并行输出：全部触发器的输出作为电路的输出
- 根据数据输入/输出方式，移位寄存器的工作方式有
  - 串入串出、串入并出、并入串出、并入并出

# 同步计数器的分析方法

➢【例8】分析下图电路，说明电路的特点。




➢解：(1)写方程式

$$J_0 = K_0 = 1;$$

$$J_1 = \overline{Q}_3^n Q_0^n, K_1 = Q_0^n;$$

$$J_2 = K_2 = Q_1^n Q_0^n;$$

$$J_3 = Q_2^n Q_1^n Q_0^n, K_3 = Q_0^n$$

$$C = \overline{\overline{Q_3^n Q_0^n}} = Q_3^n Q_0^n$$

$$Q_0^{n+1} = J_0 \overline{Q}_0^n + \overline{K}_0 Q_0^n = \overline{Q}_0^n$$

$$Q_1^{n+1} = J_1 \overline{Q}_1^n + \overline{K}_1 Q_1^n = \overline{Q}_3^n Q_0^n \overline{Q}_1^n + \overline{Q}_0^n Q_1^n$$

$$Q_2^{n+1} = J_2 \overline{Q}_2^n + \overline{K}_2 Q_2^n = Q_1^n Q_0^n \overline{Q}_2^n + \overline{Q_1^n Q_0^n} Q_2^n$$

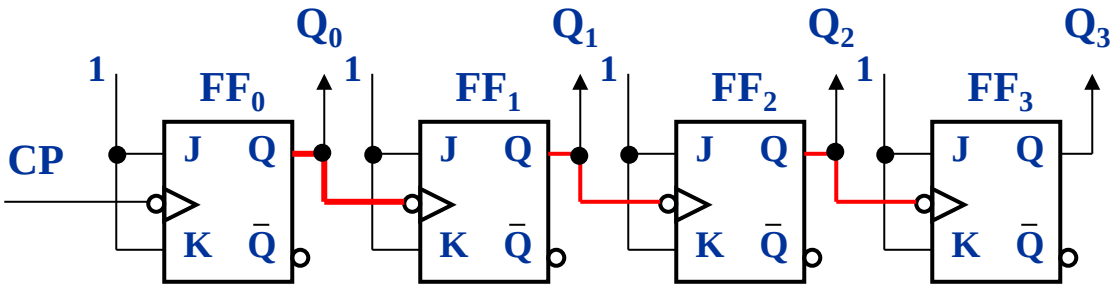
$$Q_3^{n+1} = J_3 \overline{Q}_3^n + \overline{K}_3 Q_3^n = Q_2^n Q_1^n Q_0^n \overline{Q}_3^n + \overline{Q}_0^n Q_3^n$$

$$CP_0 = CP_1 = CP_2 = CP_3 = CP\downarrow$$ ——同步电路

# 异步二进制计数器

- 电路结构特点
  - 全部由JK触发器构成
  - 第一级FF的CP由系统时钟控制，其余各级FF的CP端由前级FF的Q端或/Q端控制

- 【例9】分析异步二进制（M=16）加法计数器电路（N=4）




$FF_1$、$FF_2$、$FF_3$的CP端分别接前级触发器的Q端

- （1）状态方程

$$Q_0^{n+1}=\overline{Q_0^n}\cdot CP\downarrow;\ Q_1^{n+1}=\overline{Q_1^n}\cdot Q_0\downarrow;$$
$$Q_2^{n+1}=\overline{Q_2^n}\cdot Q_1\downarrow;\ Q_3^{n+1}=\overline{Q_3^n}\cdot Q_2\downarrow;$$

# 集成计数器实现M进制计数

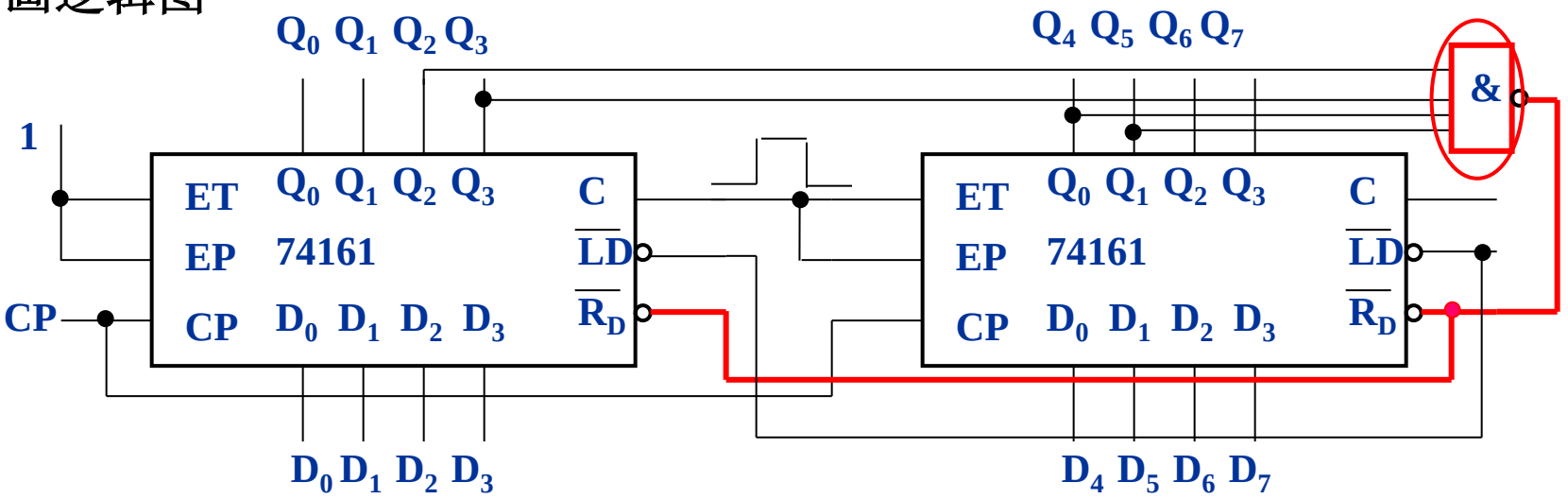
- 假如一个计数器模值为M，则两片级联后，模值变为$M^2$。但有时需要的计数器模值不是$M^2$，例如60进制、24进制
- 反馈复位法或预置法，可以得到任意进制计数器

【例12】用74161（二进制）实现M=60的加法计数器

反馈复位代码 $S_M=(60)_{10}=(111100)_2(N=6即Q_5Q_4Q_3Q_2Q_1Q_0)$

反馈复位逻辑 $\overline{R_D}=\overline{\Pi Q^1}=\overline{Q_5Q_4Q_3Q_2}$

画逻辑图



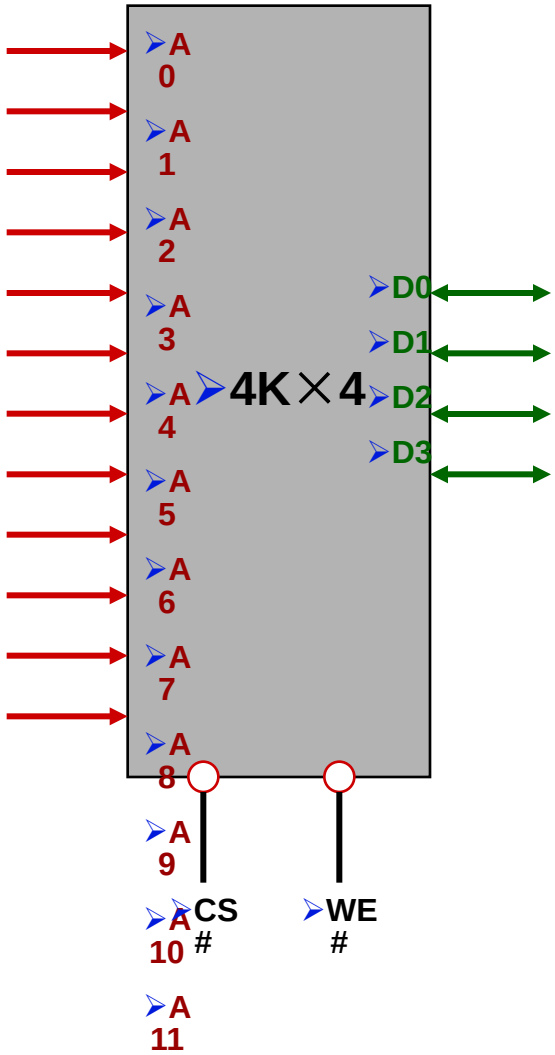
# 第四讲：主存储器（4学时）

- 目　　标
  - 了解存储单元电路的工作原理，掌握主存储器的结构特点、工作原理和构造方法。
- 主要内容
  - 存储单元电路（1学时）
    - SRAM存期单元电路
    - DRAM存储单元电路
    - ROM存储单元电路
  - 主存储器的结构（1学时）
    - SRAM芯片的内部结构
    - DRAM芯片的内部结构
  - 存储器的扩展（2学时）
  - DRAM的刷新

# 4.3 存储器芯片的扩展（混合扩展）

例：4Kx4 SRAM存储芯片构成16Kx8的存储器

- 4K×4芯片：
  - 12个地址管脚 A11～A0
  - 4个数据管脚 D3～D0
  - 1个片选输入管脚 CS#
  - 1个读写控制管脚 WE#
  - 芯片地址空间：000H～FFF H
- CPU向存储器提供：
  - 地址总线14根：AB13～AB0
  - 数据总线8根：DB7～DB0
  - 读写控制信号：MemW
  - 存储器地址空间：0000H～3FFF H
- 需要芯片数：（16K×8) / (4K×4）= 8片
  - 分4组（字扩展），每组2个芯片（位扩展）
- 一个2-4译码器产生4个片选信号
  - 译码器输入：AB13~AB12
  - 译码器输出：分别接4组芯片片选管脚



# CPU与主存的连接（示例）

CPU地址线A15～A0，数据线D7～D0，$\overline{WR}$为读/写信号，$\overline{MREQ}$为访存请求信号。0000H～3FFFH为系统程序区，4000H～FFFFH为用户程序区。用8K×4位ROM芯片和16K×8位RAM芯片构成该存储器，要求说明地址译码方案，并将ROM芯片、RAM芯片与CPU连接。

解：因为0000H～3FFFH为系统程序区，ROM区高两位总是00，低14位为全译码。

ROM区大小为：$2^{14}$×8位=16K×8位=16KB

ROM芯片数为：16K×8位 / 8K×4位 = 2×2 = 8，字方向扩展2倍，位方向扩展2倍

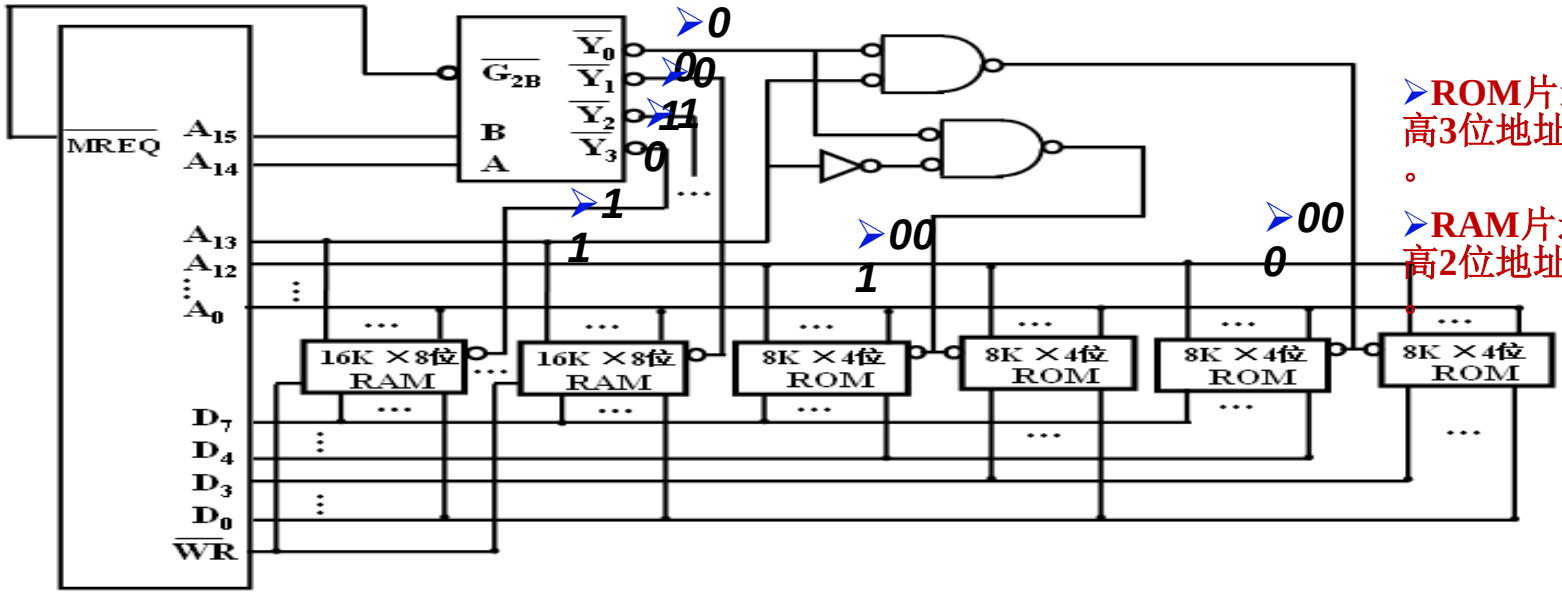
ROM芯片内地址位数为13位，连到CPU低13位地址线A12～A0

因为4000H～FFFFH为用户程序区，RAM区高两位是01、10、11，低14位为全译码。

RAM区大小为：3×$2^{14}$×8位=3×16K×8位= 48KB

RAM芯片数为：48K×8位 / 16K×8位 = 3×1 = 3，字方向上扩展3倍，位方向上不扩展。

RAM芯片内地址位数为14位，连到CPU低14位地址线A13～A0。




- ROM片选由高3位地址确定。
- RAM片选由高2位地址确定

# 第五讲：指令系统与MIPS汇编语言（6学时）

- 目　　标
  - 以X86和MIPS两种指令系统为研究对象，学习并掌握计算机指令系统的格式、寻址方式和设计方法，理解CISC和RISC两种指令系统的特点；学习并掌握MIPS汇编语言编程。
- 主要内容
  - 指令系统概述（1学时）
    - 指令系统的基本要素
    - 指令格式、寻址方式
  - 典型指令系统简介（1学时）
    - MIPS指令系统介绍
    - X86指令系统介绍
    - CISC与RISC的特点
  - MIPS汇编语言编程（4学时）

# 2.2 MIPS指令格式简介

- **MIPS 指令格式**
  - **Op： 6 bits, Opcdoe**
  - **Rs: 5 bits, The first register source operand**
  - **Rt: 5 bits, The second register source operand**
  - **Rd: 5 bits, The register destination operand**
  - **Shamt: 5 bits, Shift amount ( shift instruction)**
  - **Func: 6 bits, function code ( another Opcode)**
    - **R-Type指令OP字段为“000000”，具体操作由func字段给定**

| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |
|---|---|---|---|---|---|---|
| R-Type | Op | Rs | Rt | Rd | Shamt | Func |
| I-Type | Op | Rs | Rt | 16 bit Address or Immediate | | |
| J-Type | Op | 26 bit Address ( for Jump Instruction) | | | | |

# 伪指令

- 取地址　　la　　$s0, table
- 取立即数　　li　　$v0, 10
- 移动　　move $t8, $sp
- 乘　　mul　$t2, $a0, $a1
- 除　　div　$s1, $v1, $t7
- 求余　　rem　$s2, $v1, $t7
- 取反　　neg　$s0, $s0
- 跳转　　bnez $s0, label
-  　　/beq $s0,$zero, label

# 常用指令

- 算术运算 **add/sub/addi**
- 逻辑运算 **and/xor/or/nor/addi/xori**
- 移位指令 **srl/sll/srlv**
- 跳转指令 **beq/bne**
- 移动 **j/jal/jalr/jr**

# 循环

- C中的循环; 假定A[] 是整型数组

```
do {
      g = g + A[i];
      i ++;
while (i != h);
```

重写成

```
Loop:  g = g + A[i];
       i++;
       if (i != h) goto Loop;
```

- 假定使用以下映射

g, h, i, j, A[]的基址分别对应$s1,$s2,$s3,$s4,$s5

- MIPS 代码

```
Loop: sll $t1,$s3,2    #$t1= 4*i
      add $t1,$t1,$s5  #$t1=addr A
      lw  $t1,0($t1)   #$t1=A[i]
      add $s1,$s1,$t1  #g=g+A[i]
      add $s3,$s3,$s4  #i=i+1
      bne $s3,$s2,Loop# goto Loop if i!=h
```

- C中有3种循环结构
  - While；do…while；for
  - 每一种都可以用另外两种中的任一种等价表达, 上面的例子同样适用于 while 和 for 循环

# 第六讲：MIPS处理器设计（16学时）

- 目　　标
  - 以小型MIPS处理器为研究对象，学习并掌握基于指令执行分析的数据通路构造方法、基于与或逻辑阵列为基础的MIPS控制器设计方法，进而掌握MIPS处理器设计方法。
- 主要内容
  - 处理器的功能、组成、一般设计方法等（1学时）
  - MIPS处理器设计概述（1学时）
    - 结构、指令集、数据通路的基本组件
  - 单周期处理器设计（8学时）
    - 单周期数据通路设计（工程方法），
    - 单周期控制器设计、性能分析
  - 多周期处理器设计（6学时）
    - 多周期数据通路设计（工程方法）
    - 多周期控制器设计、性能分析

# 3.1 单周期数据通路设计

- 分析指令执行步骤，确定数据通路所需部件和部件间连接
  - 模型机指令执行过程一般会分为如下几个步骤：
    - 取指令：根据PC访问指令存储器获得指令，然后PC+4；
    - 读寄存器：根据指令格式读取相应寄存器操作数
    - ALU运算：在ALU完成相应的算术逻辑运算
    - 数据存取：LW/SW指令的数据存储器访问
    - 写寄存器：运算类指令和LW指令要把数据写入寄存器
- 使用数据通路设计表格
  - 表格记录数据通路部件输入端的输入来源
  - 暂不考虑控制信号

| 指令 | Adder | | PC | IM Add. | Registers | | | | ALU | | DM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | | | Reg1 | Reg2 | Wreg | Wdata | A | B | Add. | Wdata |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |

# 3.1 单周期数据通路设计——取指与PC自增

## 1. 取指和PC自增数据通路（所有指令）

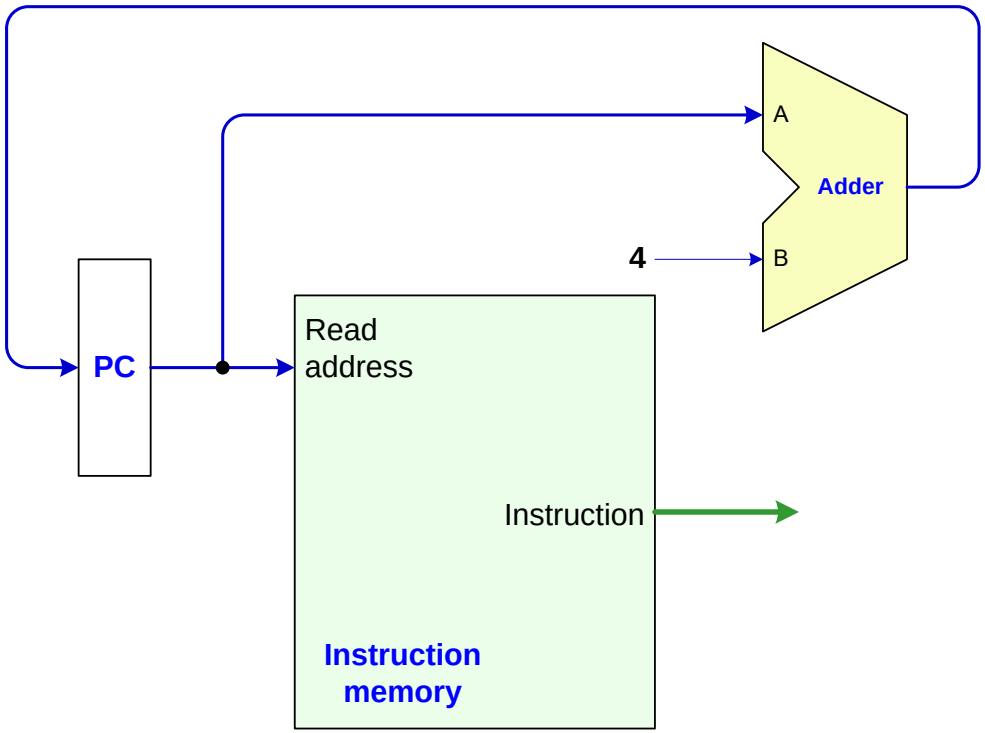
- 功能描述
  - 取指：IM Address ←PC, instruction=IM[PC]
  - PC自增：PC← PC + 4
- 所需部件：PC，Adder（实现PC加4），指令存储器IM

| 指令 | Adder | | PC | IM Add. | Registers | | | | ALU | | DM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | | | Reg1 | Reg2 | Wreg | Wdata | A | B | Add. | Wdata |
| R型指令 | PC | 4 | Adder | PC | | | | | | | | |
| Lw | PC | 4 | Adder | PC | | | | | | | | |
| Sw | PC | 4 | Adder | PC | | | | | | | | |
| Beq | PC | 4 | Adder | PC | | | | | | | | |
| | | | | | | | | | | | | |

# 3.1 单周期数据通路设计——取指与PC自增

## 1. 取指和PC自增数据通路图

- 功能描述
  - 取指：instruction = IM[PC]
  - PC自增：PC← PC + 4
- 所需部件：PC，Adder（实现PC加4），IM



# 3.2 单周期控制器设计

## ❖单周期通路所需控制信号

- ➢ALU控制（ALU Operation）：4位
- ➢其他控制信号：7个

ALU 控制

| 输入 | | ALU operation | ALU运算 |
|---|---|---|---|
| A | B | 0000 | A & B |
| A | B | 0001 | A \| B |
| A | B | 0010 | A + B |
| A | B | 0110 | A – B |

7个控制信号

| 控制信号 | 失效时作用 | 有效时作用 |
|---|---|---|
| RegDst | 寄存器堆写入端地址来选择Rt字段 | 寄存器堆写入端地址选择 Rd字段 |
| RegWrite | 无 | 把数据写入寄存器堆中对应寄存器 |
| ALUSrc | ALU输入端B选择寄存器堆输出R[rt] | ALU输入端B选择Signext输出 |
| PCSrc | PC输入源选择 PC+4 | PC输入选择beq指令的目的地址 |
| MemRead | 无 | 数据存储器DM读数据（输出） |
| MemWrite | 无 | 数据存储器DM写数据（输入） |
| MemtoReg | 寄存器堆写入端数据来自ALU输出 | 寄存器堆写入端数据来自DM输出 |

# 3.2 单周期控制器设计

- 控制器分成两部分：主控单元和ALU控制单元
  - 主控单元
    - 输入：指令操作码字段 Op（指令31:26位）
    - 输出：
      - 7个控制信号
      - ALU控制单元所需的2位输入ALUop
  - ALU控制单元
    - 输入：
      - 主控单元生成的ALUop
      - 功能码字段Func（指令5:0位）
    - 输出：ALU运算控制信号 ALU operation（4位）

ALUOp指明ALU的运算类型
- 00：访存指令所需加法
- 01：beq指令所需减法
- 10：R型指令功能码决定

| Op (31-26) | Rs (25-21) | Rt (20-16) | Rd (15-11) | Shamt (10-6) | Func (5-0) |
|---|---|---|---|---|---|

# 3.2 单周期控制器设计

## ❖主控单元控制信号分析

- **RegDst**
  - R型指令：RegDst=1，选择Rd
  - Lw指令： RegDst=0，选择Rt
  - 其他指令：不关心
- **ALUScr**
  - R型指令：ALUSrc=0，选择寄存器堆的 Read data2 输出
  - Lw指令： ALUSrc=1，选择Signext的输出
  - Sw指令：ALUSrc=1，选择Signext的输出
  - Beq指令（减法运算）： ALUSrc=0，选择 Read data2 输出
- **MemtoReg**
  - R型指令：MemtoReg=0，选择 ALU 输出
  - Lw指令： MemtoReg=1，选择数据存储器DM输出
  - 其他指令： 不关心
- **Branch**
  - Beq指令：Branch=1，此时若Zero=1，PC输入选择加法器Nadd输出（分支指令目的地址），否则选择加法器Add输出（PC+4）
  - 其他指令：Branch=0，PC输入选择加法器Add输出（PC+4）

# 3.2 单周期控制器设计

主控单元真值表

| Input or output | Signal name | R-format | lw | sw | beq |
|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 |
| | Op4 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 |
| | Op2 | 0 | 0 | 0 | 1 |
| | Op1 | 0 | 1 | 1 | 0 |
| | Op0 | 0 | 1 | 1 | 0 |
| Outputs | RegDst | 1 | 0 | X | X |
| | ALUSrc | 0 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | X | X |
| | RegWrite | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 |
| | Branch | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 |
| | ALUOp0 | 0 | 0 | 0 | 1 |

提供给ALU控制单元

# 3.3 单周期数据通路性能分析

## ❖指令执行时间计算

1. 方式一：采用单周期，即所有指令周期固定为单一时钟周期
   - 时钟周期有最长的指令决定（LW指令），为 600ps
   - 指令平均周期 = 600ps
2. 方式二：不同类型指令采用不同指令周期（可变时钟周期）
   - 假设指令在程序中出现的频率
     - lw指令 ： 25%
     - sw指令 ： 10%
     - R类型指令 ： 45%
     - beq指令 ： 15%
     - j指令 ： 5%
   - 平均指令执行时间

     600*25%+550*10%+400*45%+350*15%+200*5% = 447.5ps

—若采用可变时钟周期，时间性能比单周期更高；

—但控制比单周期要复杂、困难，得不偿失。

—改进方法：改变每种指令类型所用的时钟数，即采用多周期实现

## ❖CPI：指令平均执行时钟周期数

- CPI：Clock cycles Per Instruction。
- 不同指令功能不同，所需时间也不同，CPI只是某一机器中一个程序或程序片段每条指令所用时钟周期的平均值。
- 不同指令集的CPI比较没有实际意义。

CPU 时间=CPU 时钟周期数/频率；

CPU 时间=CPU 时钟周期数*时钟周期长；

平均时钟周期数 CPI=CPU 时钟周期数/IC（指令的条数）；

CPU 时间=（IC*CPI）/频率 f；

$$\text{CPU 的时钟周期数} = \sum_{i=1}^{n}(CPI_i \times I_i)$$

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{IC} = \sum_{i=1}^{n}(CPI_i \times \frac{I_i}{IC})$$

# 计算机性能评价

## ❖MIPS：百万指令每秒

- MIPS：Million Instruction Per Second
- 不同指令集的MIPS比较没有实际意义
- 即使同一台机器，用不同的测试程序测出来的MIPS值也可能不一样。

$$\text{MIPS} = \frac{\text{指令条数}}{\text{执行时间} * 10^6} = \frac{f}{CPI * 10^6}$$

## ❖MFLOPS：百万浮点数操作每秒

- MFLOPS： Million Floating point Operations Per Second
- 可以比较不同机器的浮点运算能力, 但有局限性
- MFLOPS不仅和机器有关, 也和所用测试程序有关
- MFLOPS与整数. 浮点操作的比例有关

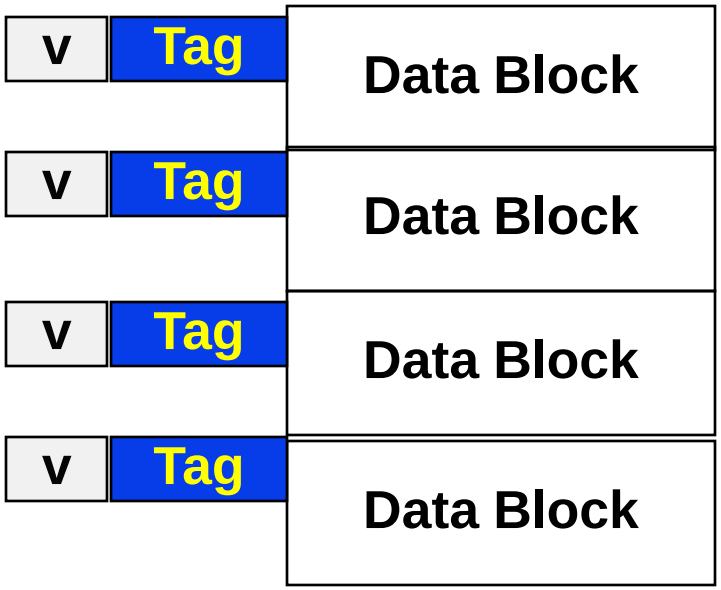
$$\text{MFLOPS} = \frac{\text{程序中的浮点操作次数}}{\text{执行时间} * 10^6}$$

# 第七讲：高速缓存存储器（CACHE）（6学时）

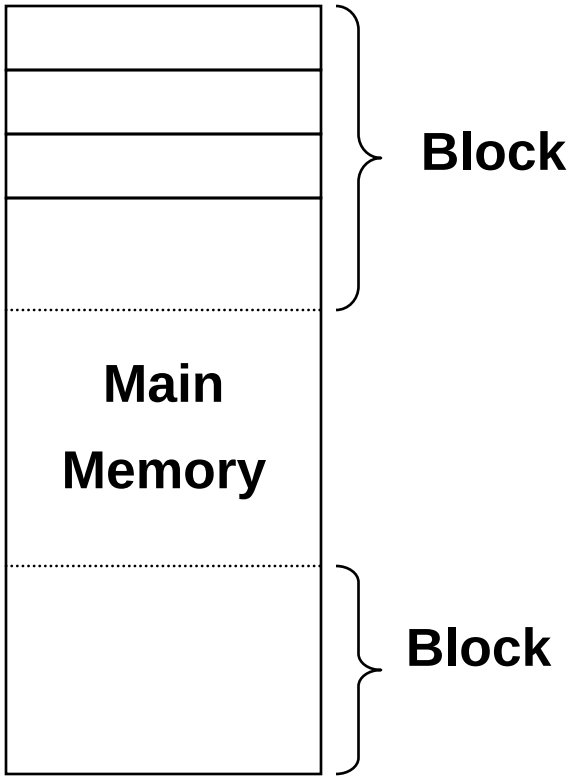
- 目　　标
  - 掌握高速缓存存储器（Cache）的结构特点和工作原理，以及多级Cache层次关系，掌握Cache的映射机制、Cache的命中与缺失分析及其性能计算方法。
- 主要内容
  - 程序执行局部性原理
  - Cache的结构与工作原理
  - Cache的映射机制
    - 直接映射
    - 全相联映射
    - 组相联映射
  - Cache的替换策略
  - Cache性能分析与其他
    - Cache数据一致性问题
    - 命中率与缺失分析
    - 性能计算

# 1.2 高速缓冲存储器(Cache)的原理

- **Cache的基本结构**
  - 存储机构：保存数据，存取数据，一般采用SRAM构成。以Block（若干字）为单位；
  - 地址机构：地址比较机制，地址转换机制，地址标标记（Tag），一个Block具有一个Tag；
  - 替换机构：记录Block的使用情况，替换策略，有效位（v）记录对应数据块中的数据是否有效。




Cache 的基本结构



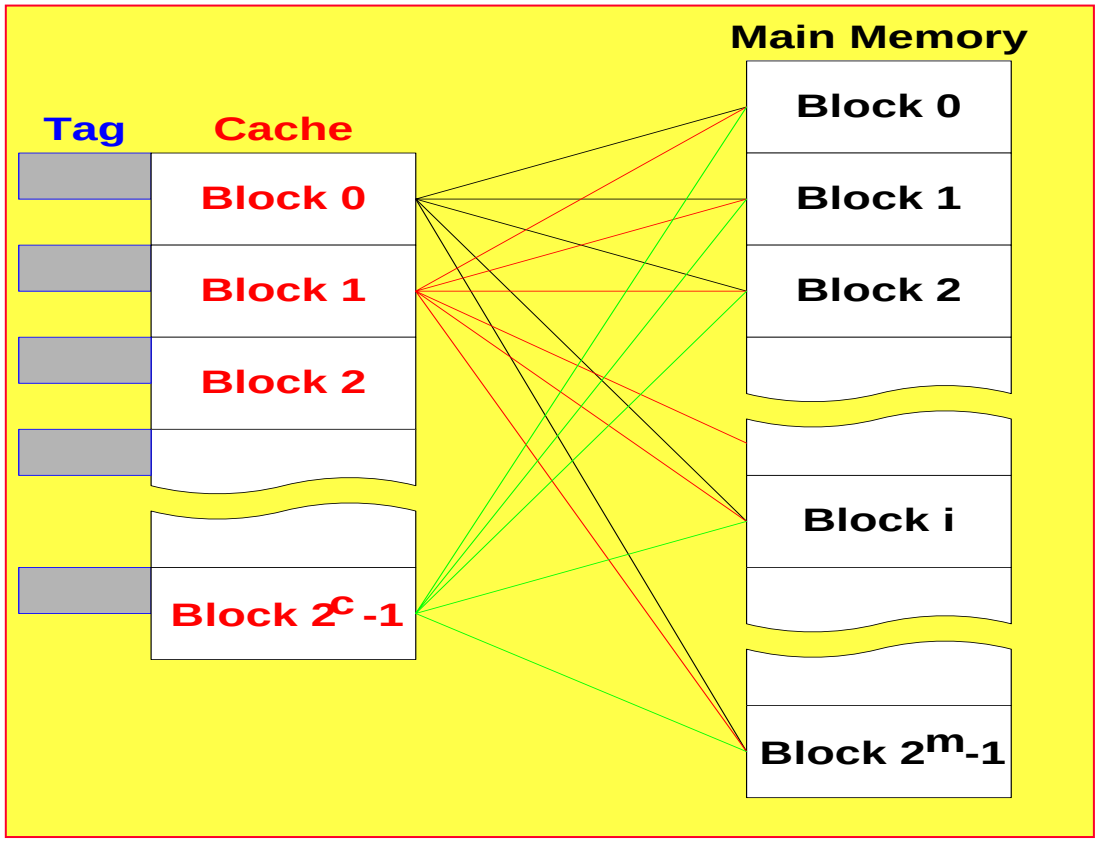
# 1.2 高速缓冲存储器(Cache)的原理

❖ Cache的有关术语

- 数据块（block）：Cache与主存的基本划分单位，也是主存与Cache一次交换数据的最小单位，由多个字节（字）组成，取决与主存一次读写操作所能完成的数据字节数。也表明主存于Cache之间局部总线的宽度。
- 标记（tag）：Cache每一数据块有一个标记字段，用来保存该数据块对应的主存数据块的地址信息。
- 有效位（valid bit）：Cache中每一Block有一个有效位，用于指示相应数据块中是否包含有效数据。
- 行（line )：Cache中 一个block及其 tag、valid bit构成1行。
- 组（set）：若干块(Block)构成一个组，地址比较一般能在组内各块间同时进行。
- 路（way）：Cache相关联的等级，每一路具有独立的地址比较机构，各路地址比较能同时进行（一般与组结合），路数即指一组内的块数。
- 命中率（hit rate）：目标数据在Cache中的存储访问的比例。
- 缺失率（miss rate）：目标数据不在Cache中的存储访问的比例。

# 2.1 Cache与主存之间的映射—全相联

- 全相联（Full Associative）
  - 主存分为若干Block，Cache按同样大小分成若干Block，Cache中的Block数目显然比主存的Block数少得多。
  - 主存中的某一Block可以映射到Cache中的任意一Blcok。



# 2.2 Cache与主存之间的映射—组相联

- 组相联映射
  - 主存的地址格式：

| Tag | Set # | Offset |
|---|---|---|

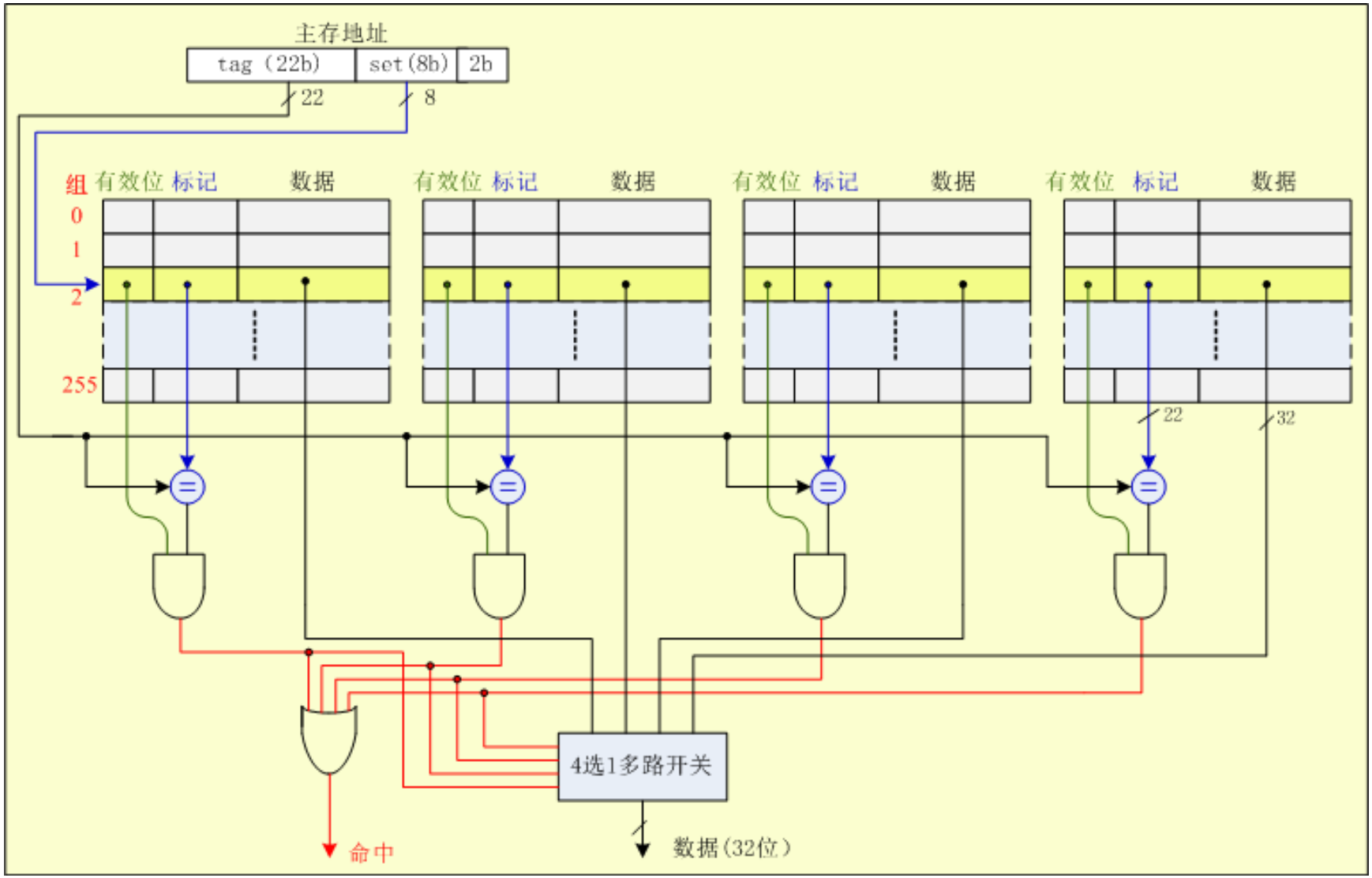
  - Tag的内容：主存中与该Cache数据块对应的数据块的组内块地址。

- 举例
  - 主存容量1M 字节，4路组相联（每组包含4个Block）Cache数据容量16K字节，Block大小256 字节，，有1位有效位，每个字有一个修改位。
  - Cache分多少组？每组包含多少块？
  - Cache的Tag需要多少位？Cache的总容量是多少？

  - 解：
    - Cache 组数＝ $2^{14} \div (2^8 \times 2^2) = 2^4 = 16$ 组
    - 主存每组块数＝ $2^{20} \div (2^8 \times 2^4) = 2^8 = 256$ 块/组
    - 主存地址：20 位，其中高8 位为组内块地址，中间4 位为组地址，低 8位为块内地址
    - Cache的Tag应该为 8 位。
    - Cache总容量：16K/256*（128*8+1+64）

# 2.2 Cache与主存之间的映射—组相联

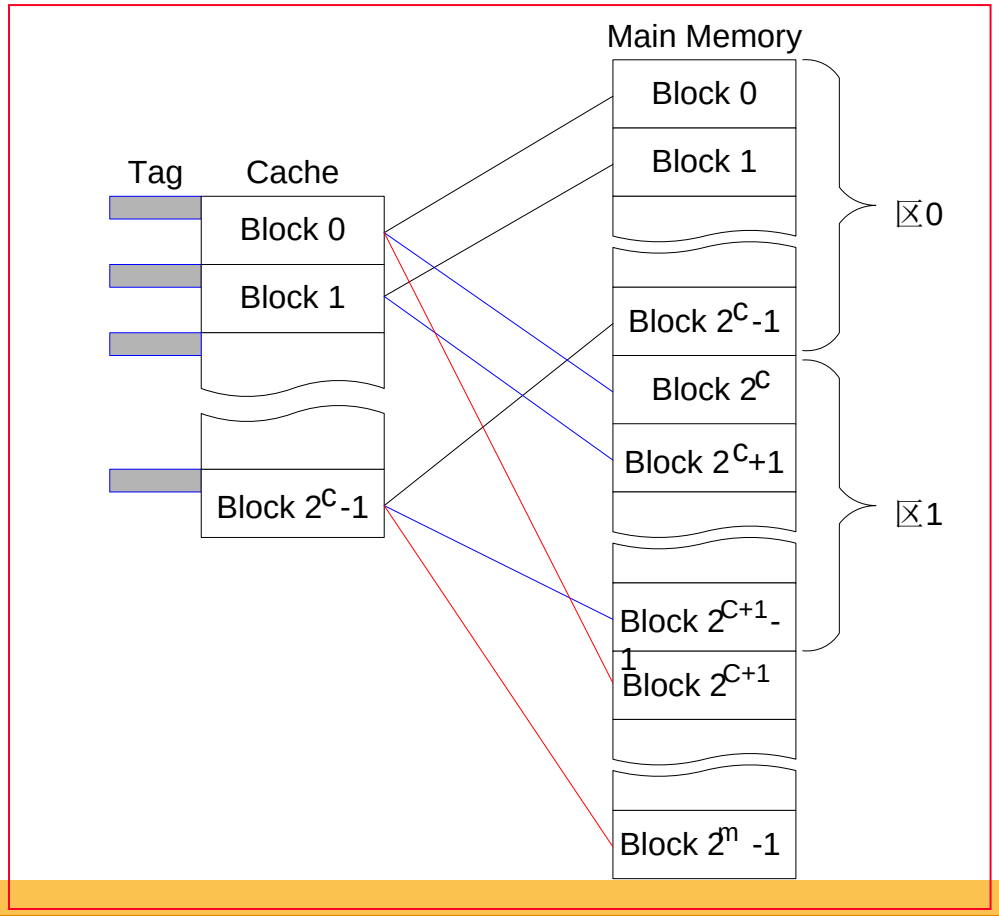
- Cache示例：Cache容量4KB，4路组相联，数据块大小4B，主存地址32位。



# 2.3 Cache与主存之间的映射—直接映射

## ❖直接（Direct）

- 主存中的某一块 J 映射到Cache中的固定块 K， K ＝ J Mod M， 其中M是Cache包含的块数。
- 实际上是将主存按Cache的大小分区，一个区内的各块分别与Cache的对应各块映射。



# 2.3 Cache与主存之间的映射—直接映射

- 直接映射

主存的地址格式：

| Tag | Index | Offset |
|---|---|---|

Cache的Tag内容：主存中与该Cache数据块对应的数据块的区地址（区号）。

Index：区内的块索引号

- 举例
  - 主存容量16M字节，Cache数据容量64K字节，Block大小16 Bytes
  - Cache包含多少块？
  - Tag应该多少位？

解：

Cache: $2^{12}$ Blocks

主存: $2^{20}$ Blocks, 分成$2^{8}$个区，每个区$2^{12}$Blocks

主存地址：24位，其中高 8 位区地址，中间12位为区内块地址，低4位为块内地址

Cache的Tag应该为 8位。

# Cache的替换策略

- 替换算法的实现
  - 一般由硬件电路实现，因此应以简单、便于硬件实现为宜。
- LRU的实现（计数器法）
  - 缓存的每一块都设置一个计数器；
  - 被调入或者被替换的块， 其计数器清 0，而其它的计数器则加 1；
  - 访问命中时，所有块的计数值与命中块的计数值进行比较，如果计数值小于命中块的计数值， 则该块的计数值加 1；如果块的计数值大于命中块的计数值，则数值不变。最后将命中块的计数器清为0。
  - 需要替换时，则选择计数值最大的块被替换。
- FIFO的实现（计数器法）
  - 例如Solar－16/65机Cache采用组相联方式，每组4块，每块都设定一个两位的计数器，当某块被装入或被替换时该块的计数器清为0，而同组的其它各块的计数器均加1，当需要替换时就选择计数值最大的块被替换掉。

# Cache举例

某计算机主存容量 16MB，Cache 容量 16KB，采用 4 路组相联（每组 4 块）映射方式和 LRU 替换策略，每个数据块 16 字节。假设 Cache 中第 8 组（组地址为 8）的 4 个数据块的装入情况及 Tag 内容如下表（装入位为 1 表示数据块已装入）。

| 装入位 | Tag |
|---|---|
| 1 | F40H |
| 1 | 430H |
| 0 | 218H |
| 1 | 030H |

（1） Cache 分多少组？（2 分）

（2） 给出主存的地址格式；（2 分）

（3） 若 CPU 要依次读取主存地址 430082H、2F8086H、03008AH、F40088H、063081H 单元中的数，则读取哪几个地址单元的数会产生 Cache 失效？5 个数读取结束时上表中装入位和 Tag 内容各是多少。

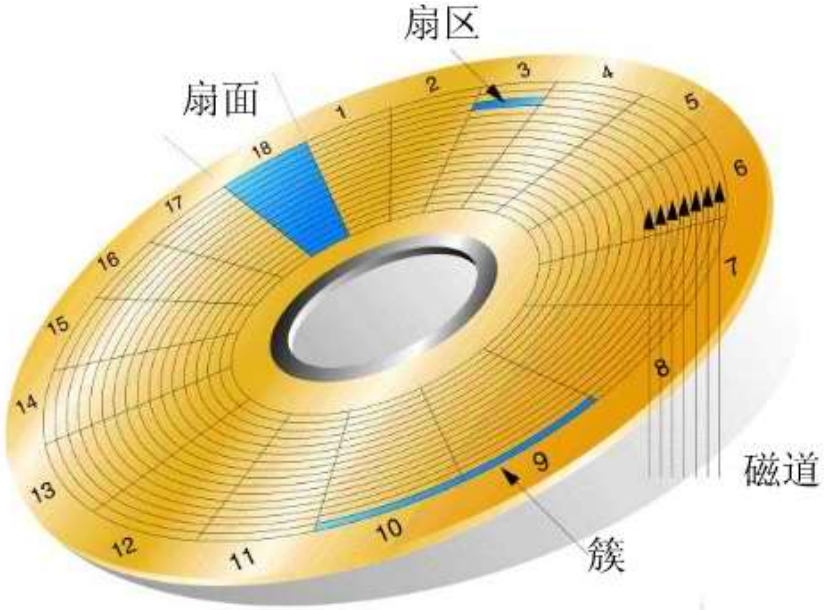
# 第八讲：虚拟存储系统（4学时）

- 目　　标
  - 掌握虚拟存储器工作原理、虚实地址转换与页表工作原理、TLB工作原理，具备进行虚拟存储器性能分析的能力。
- 主要内容
  - 虚拟存储器工作原理
  - 虚实地址转换
  - 页表工作原理
  - TLB工作原理
  - 虚拟存储器性能分析

# 硬磁盘基本结构

## ❖数据结构与格式

- ➢数据结构：
  - 磁道（柱面：Cylinder）
  - 盘面（磁头：Head）
  - 扇区（Sector）
- ➢扇区容量：512 Bytes
- ➢每个磁道包含的扇区数相同
- ➢最小访问单位：扇区
- ➢扇区的地址表示：
- ➢存储容量
- ➢读写速率




磁盘上的磁道、扇区和簇

扇区地址：

| Cylinder# | Head# | Sector# |
|---|---|---|

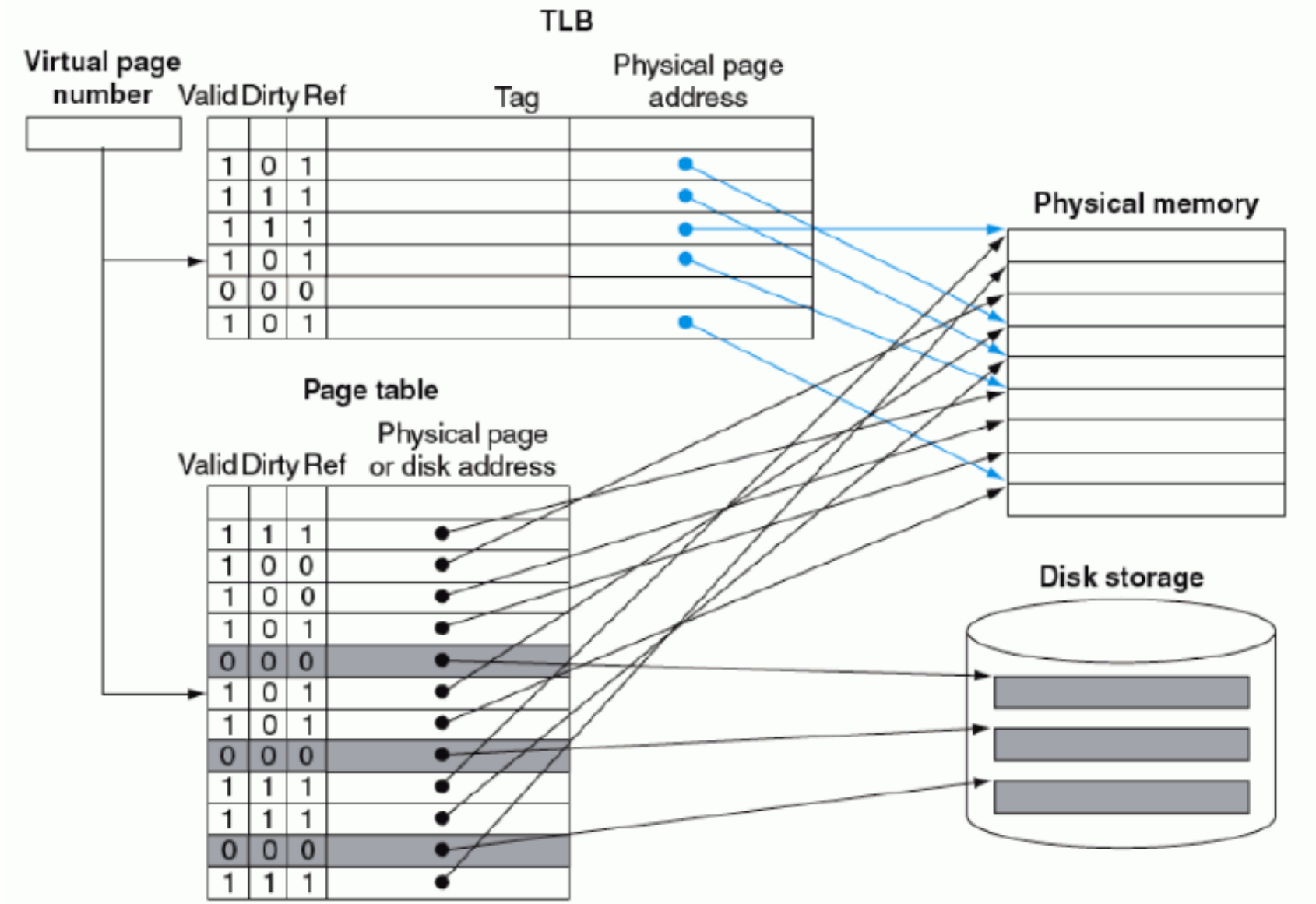
# 概述

- 虚拟存储器的概念
  - 虚拟存储器是一种把主存当做辅助存储器的高速缓存的技术。程序运行时，内存管理采用交换机制，进程保存在辅存中，进程执行时，依据局部性原理，只将其活跃部分调入内存。从逻辑上为用户提供一个比物理内存容量大得多的“主存储器”。
  - **CPU中专门有一个存储器管理部件MMU（Memory Management Unit）协助OS完成存储器访问**

# 第八讲：虚拟存储系统（4学时）

- 虚拟存储技术的动机
  - （早期）采用单道程序运行，系统的主存中包含：
    - 操作系统（常驻监控程序）
    - 正在执行的一个用户程序

    所以无需进行存储管理，即使有也很简单
  - 采用多道程序运行，主存中包括操作系统和若干用户程序（进程）
    - 多道程序（进程）如何有效安全的共享内存？
      - 同时运行的程序对内存需求之和超过计算机实际内存容量；
      - 局部性原理：某个时间点，只需要一部分活跃的程序
    - 如何消除小的主存容量对编程的限制？
      - 编写程序时，不知道程序运行时将和哪些程序共享内存；编程者总是希望把每个程序编译在它自己的地址空间中。
      - 单用户程序大小超过主存容量


TLB
Virtual page number
Valid Dirty Ref
Tag
Physical page address
1 0 1
1 1 1
1 1 1
1 0 1
0 0 0
1 0 1
Physical memory
Page table
Valid Dirty Ref
Physical page or disk address
1 1 1
1 0 0
1 0 0
1 0 1
0 0 0
1 0 1
1 0 1
0 0 0
1 1 1
1 1 1
0 0 0
1 1 1
Disk storage

# 第八讲：虚拟存储系统（4学时）

- TLB，页表，Cache三种缺失的可能性

| TLB | Page table | Cache | Possible? If so, under what circumstance? |
|---|---|---|---|
| hit | hit | miss | 可能，TLB命中则页表一定命中，但实际上不会查页表 |
| miss | hit | hit | 可能，TLB缺失但页表可能命中，信息在主存，就可能在Cache |
| miss | hit | miss | 可能，TLB缺失但页表可能命中，信息在主存，但可能不在Cache |
| miss | miss | miss | 可能，TLB缺失页表可能缺失，信息不在主存，一定也不在Cache |
| hit | miss | miss | 不可能，页表缺失，说明信息不在主存，TLB中一定没有该页表项 |
| hit | miss | hit | 同上 |
| miss | miss | hit | 不可能，页表缺失，说明信息不在主存，Cache中一定也没有该信息 |

最好的情况应该是hit、hit、hit，此时，访问主存几次？ 不需要访问主存！

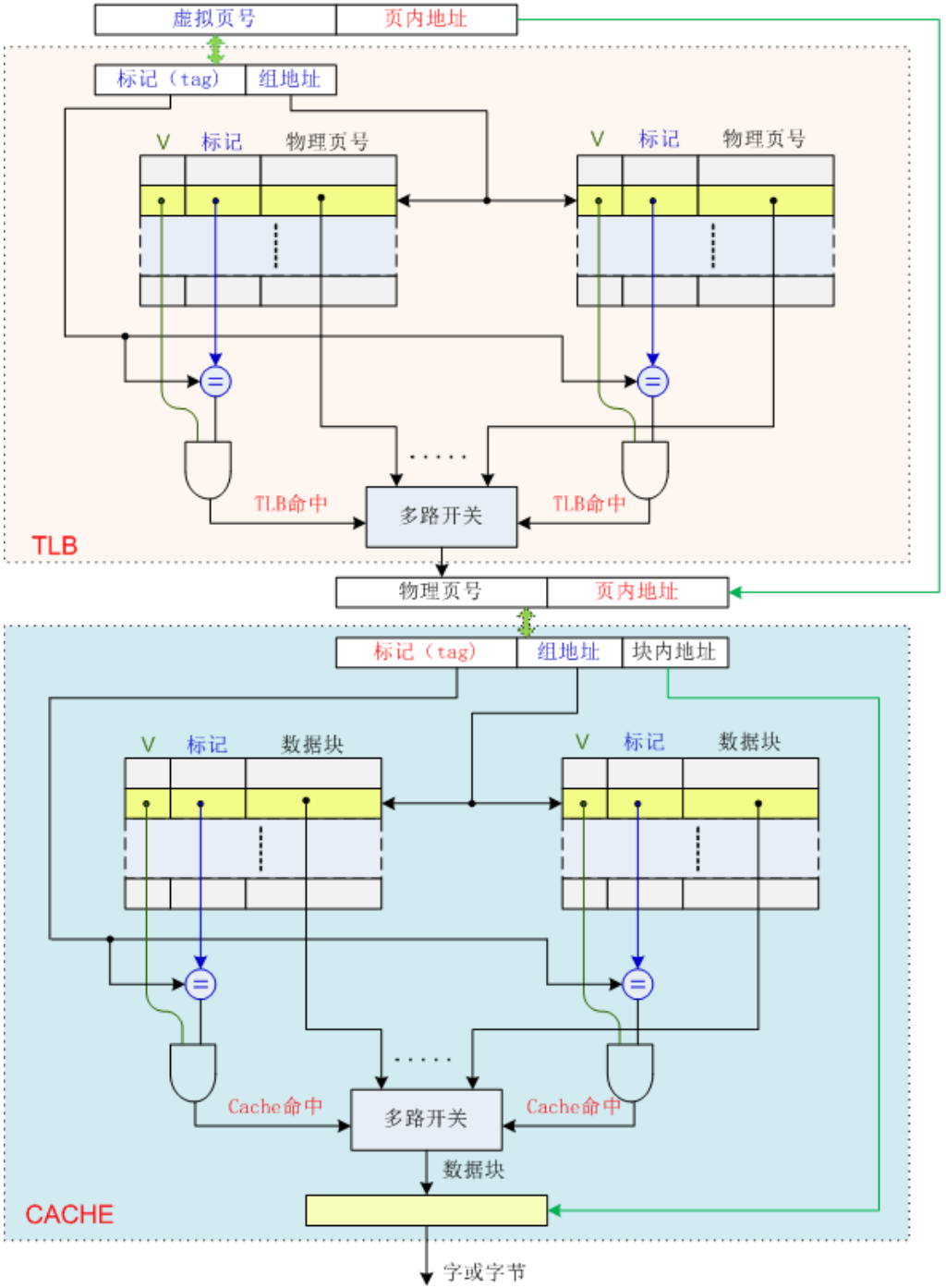
以上组合中，最好的情况是什么？hit、hit、miss和miss、hit、hit 只需访问主存1次

以上组合中，最坏的情况是什么？ miss、miss、miss 需访问磁盘、并访存至少2次

介于最坏和最好之间的是什么？ miss、hit、miss 不需访问磁盘、但访存至少2次

# 页式虚拟存储器

- **CPU通过TLB和Cache访问的全过程**
  - **TLB和Cache都采用组相联**
  - **以虚拟页号为依据访问TLB获取物理页号**
  - **以物理地址为依据访问Cache获取最终数据**



# 举例

- 假定页式虚拟存储系统按字节编址，逻辑地址36位，页大小16KB，物理地址32位，页表中包括有效位和修改位各1位、使用位和存取方式位各2位，且所有虚拟页都在使用中。请问：
  - （1）每个进程的页表大小最多为多少？
  - （2）如果所使用的快表（TLB）总表项数为256项，且采用2路组相联Cache实现，则快表大小至少为多少？

**解答（1）**

页面大小：**16KB=$2^{14}$，页内偏移14位**

虚地址36位：**虚页号=36-14=22位**

实地址32为：**实页号=32-14=18位**

每个进程最多可有：**$2^{22}$个虚页**

每个页表项：**1 +1+2 +2+ 18=24位**

每个页表所占空间：**$2^{22}\times24$=12MB**

**（2）**

**TLB：256个表项，2路组相联，所以共有128组**

**22位虚页号：7位组地址，15位Tag**

**TLB每个表项：15+24=39位**

**TLB容量：39×256=9984位=1248字节**

# 第九讲：外部存储与输入输出方式（4学时）

- 目　　标
  - 掌握程序查询I/O、中断I/O和DMA I/O等输入输出方式的工作原理。
- 主要内容
  - 外部存储器（1学时）
  - I/O方式（3学时）
    - 程序查询I/O方式
    - 中断与中断I/O方式
    - DMA I/O方式
    - I/O通道
  - MIPS的I/O抽象

# I/O方式小结

- I/O方式的演变（CPU从I/O事务中的解放）

① 直接控制方式：CPU直接控制外设，主要用于简单的微处理器控制设备；

② 程序I/O方式：增加控制器和I/O模块，处理器使用编程I/O，使处理器从外设的I/O细节中解脱出来；

③ 中断I/O方式：增加控制器和I/O模块，采用中断I/O方式，处理器不需要浪费时间等待I/O操作完成，提高了处理器的效率；

④ DMA方式： I/O模块通过DMA直接存储存储器，除在传输开始和结束时，传输数据不需要处理器参与；

⑤ I/O通道方式： I/O模块成为有自主控制权的处理器，有处理I/O的专用指令集。CPU指示I/O处理器执行存储器中的I/O程序，I/O处理器不需要CPU干预就能获取并执行I/O指令。这允许CPU指派一系列的I/O活动，并只在整个活动执行完成后才中断CPU；

⑥ I/O处理器方式： I/O模块带局部存储器，成为自治的计算机。这种结构可以控制大量的I/O设备而最小化CPU的干预。

# MIPS IO抽象（内存映射）

- addi $t0, $0, 7　　假定：I/O: 0xffffffff4
- sw $t0, 0xfff4($0)　//写
- Lw $t1, oxfff4($0)　//读