

# Chapter 01 计算机系统概述

## 1 冯·诺依曼结构计算机

### 1.1 储存程序(stored-program)

- 必须事先编好程序，并将程序和原始数据送入储存器后才能执行程序
- 一旦程序被启动，计算机能在不需要操作人员干预的情况下自动从储存器中逐条取出指令并执行指令

### 1.2 主要思想

- 采用“储存程序”的工作方式
- 计算机由运算器、控制器、储存器、输入设备和输出设备5个基本部件组成
  - 储存器不仅能存放数据，也能存放指令，形式上数据和指令没有区别，但计算机应该能区分它们
  - 控制器应能自动控制指令的执行
  - 运算器应能进行算术运算，也能进行逻辑运算
  - 操作人员可以通过输入/输出设备使用计算机
- 计算机内部以二进制方式表示指令和数据
  - 每条指令由操作码和地址码两部分组成
    - 操作码指出操作类型
    - 地址码指出操作数的地址

通常把控制部件、运算部件和各类寄存器互连组成的电路称为**中央处理器(Central Processing Unit, CPU)**，简称处理器；

把用来存放指令和数据的储存部件称为**主储存器(Main Memory, MM)**，简称主存或内存。

## 1.3 基本部件

### 1.3.1 储存器

- 主存中的每个单元需要编号，称为主存地址
- **主存地址寄存器(Memory Address Register, MAR)**: CPU送到地址线的主存地址
- **主存数据寄存器(Memory Data Register, MDR)**: 发送或从数据线取来的信息

### 1.3.2 运算器

- **算术逻辑部件(Arithmetic and Logic Unit, ALU)**: 算术运算和逻辑运算
- **通用寄存器组(General Purpose Register set, GPRs)**: 临时存放从主存取来的数据或运算的结果
- **标志寄存器**: 存放ALU运算产生的的标志信息

### 1.3.3 控制器

- **控制部件(Control Unit, CU)**: 自动读取指令并译码
- **指令寄存器(Instruction Register, IR)**: 存放从主存取来的指令
- **程序计数器(Program Counter, PC)**: 存放将要执行的下一条指令的主存地址

#### 1.3.4 I/O设备

## 2 程序的表示与执行

### 2.1 指令执行过程

**机器指令(instruction)**和计算机中的程序一样，都用0和1表示。

每条指令的执行过程包括：

- 从主存取指令
  - 以PC为地址访问主存
- 对指令进行译码
- PC增量
- 取操作数并执行
- 将结果送到主存或寄存器保存

程序执行前，首先将程序的起始地址存放在PC中

### 2.2 程序编写语言

#### 2.2.1 机器级语言

**机器代码/机器语言程序**：计算机能直接理解和执行

**汇编语言(assembly language)**：用简短的英文符号和机器指令建立联系

**汇编指令**：机器指令对应的符号

**机器级代码程序**：使用机器级语言编写的程序

#### 2.2.2 高级编程语言

**高级程序设计语言/高级编程语言**：面向算法描述的语言

#### 2.2.3 翻译程序(translator)

**源语言/源程序**：被翻译的语言和程序

**目标语言 / 目标程序**：翻译生成的语言和程序、

翻译程序的分类：

- **汇编程序/汇编器(assembler)**：将汇编语言源程序翻译成机器语言目标程序
- **解释程序/解释器(interpreter)**：将源程序中的语句按照其执行顺序逐条翻译成机器指令并立即执行
- **编译程序/编译器(compiler)**：将高级语言程序翻译成汇编语言或机器语言目标程序

## 3 计算机系统抽象层

计算机系统抽象层及其转换	
软件	应用 (问题)
	算法
	编程 (语言)
	操作系统
过渡	指令集体体系结构 (ISA)
	微体系结构
	功能部件/RTL
	数字逻辑电路
硬件	器件

**语言处理系统**: 提供程序编辑器和各类翻译转换软件的工具

**操作系统**: 对计算机系统结构和计算机硬件的一种抽象, 构成了一台可以供程序员使用的**虚拟机** (*virtual machine*)

**指令集体体系结构(Instruction Set Architecture)**: 架在软件和硬件界面上的桥梁, 有些场合简称为**系统结构(architecture)**或**指令系统(instruction set)**

**计算机组成(computer organization)/微体系结构(micro architecture)简称微架构**: 实现ISA的具体逻辑结构

**功能部件层/寄存器传送级(Register Transfer Level, RTL)层**: 一个特定的微架构由运算器、通用寄存器和储存器等功能部件构成, 功能部件由**数字逻辑电路(digital logic circuit)**实现, 每个基本的逻辑门电路由特定的**器件技术**实现

# Chapter 02 指令系统

## 1 指令系统概述

**指令集体系结构(Instruction Set Architecture, ISA)**: 一台计算机的抽象模型，位于软件和硬件之间的交界面，是构成程序的基本元素，也是硬件设计的依据，简称**指令系统**。

## 2 指令系统设计

### 2.1 操作数和寻址方式

**形式地址A**: 指令中的地址码字段

**有效地址EA**: 指令中给出的操作数所在储存单元的地址

**寻址方式**: 指令给出操作数或操作数地址的方式称为寻址方式

**立即寻址**: 指令中直接给出操作数本身(**立即数**)

**直接寻址**: 形式地址是操作数的有效地址(**直接地址/绝对地址**)， $EA=A$

**间接寻址**: 形式地址是有效地址的地址， $EA=(A)$ ，符号(x)表示寄存器编号x或者储存单元地址x中的内容

**寄存器寻址/寄存器直接寻址**: 形式地址是操作数所在的寄存器编号， $EA=A$

- 寄存器编号比储存器地址短
- 操作数已在CPU中，不用访存，执行速度快

**寄存器间接寻址**: 形式地址是有效地址的寄存器编号， $EA=(A)$

**变址寻址**: 形式地址是**基准地址A**，**变址寄存器I**中存放的是**偏移量/位移量**。有效地址  $EA=A+(I)$

**相对寻址**: 形式地址A给出偏移量，基准地址隐含由PC给出， $EA=(PC)+A$

**基址寻址**: 形式地址A给出偏移量，基准地址由**基址寄存器B**给出， $EA=(B)+A$

变址寻址、相对寻址、基址寻址统称为**偏移寻址**。

### 2.2 操作类型

- **算术和逻辑运算指令**

加(ADD)、减(SUB)、比较(CMP)、乘(MUL)、除(DIV)、与(AND)、或(OR)、取反(NOT)、取负(NEG)、异或(XOR)、加1(INC)、减1(DEC)等

- **移位指令**

算术移位、逻辑移位、循环移位、半字交换

- **数据传送指令**

- MOV: 寄存器之间的传送
- LOAD: 从内存单元读数据到寄存器
- STORE: 从寄存器写数据到内存单元

- **串指令**: 对字符串进行操作的指令
- **顺序控制指令**
  - 有条件转移(BRANCH)、无条件转移(JMP)、调用(CALL)、返回(RET)
  - 通过将转移目标地址送到PC中来实现
    - **绝对转移**: 转移目标地址用直接寻址方式给出
    - **相对转移**: 转移目标地址用相对寻址方式给出
  - **无条件转移指令**: 在任何情况下都执行转移操作
  - **条件转移指令/分支指令**: 在满足特定条件时才执行转移操作
  - **转子指令/调用指令**: 用于子程序调用（**过程调用或函数调用**），必须保存下条指令的地址（称为**返回地址**）
  - **返回指令**: 将实现保存的返回地址传到PC
- **系统指令**
  - 停机、开中断、关中断、系统模式切换、进入特殊处理程序
- **输入输出指令**

## 2.3 操作码编码

**定长操作码**: 操作码为固定长度，译码方便，指令执行速度快，但有信息冗余

**变长操作码**: 操作码按短到长扩展编码

## 2.4 标志信息的生成和使用

**条件码(Condition Code)/状态位(status)**: 标志信息

**程序状态字(Program Status Word, PSW)**: 零标志ZF、溢出标志OF、符号标志SF、进位/借位标志CF，可由专门的**条件码寄存器**（或称**状态寄存器**、**标志寄存器**、**程序状态字寄存器**）来存放

## 2.5 指令系统风格

### 2.5.1 按操作数位置指定风格

- **累加器(Accumulator, AC)型指令系统**
- **栈(Stack)型指令系统**
- **通用寄存器(General Purpose Register)型指令系统**
  - 操作数可以来自：立即数(I)、通用寄存器(R)、储存单元(S)
  - 指令类型可以是：RR型(两个操作数都来自寄存器)、RS型(两个操作数分别来自寄存器和存储单元)、SI型(两个操作数分别来自存储单元和立即数)、SS型(两个操作数都来自储存单元)
- **Load/Store型指令系统**

### 2.5.2 按指令格式的复杂度来分

- **CISC风格指令系统**
  - **复杂指令集计算机(Complex Instruction Set Computer)**: 设计指令时使机器指令的功能接近高级语言语句的功能
- **RISC风格指令系统**
  - **精简指令集计算机(Reduced Instruction Set Computer)**:
    - 指令数目少
    - 指令格式规整
    - 采用Load/Store型指令设计风格

- 采用大量通用寄存器

## 3 指令系统实例：RISC-V架构

### 3.1 32位RISC-V指令格式

	31	27	26	25	24	20	19	15	14	12	11	7	6	0
<b>R</b>	funct7				rs2		rs1	funct3		rd		opcode		
<b>I</b>		imm[11:0]					rs1	funct3		rd		opcode		
<b>S</b>	imm[11:5]				rs2		rs1	funct3	imm[4:0]			opcode		
<b>B</b>	imm[12 10:5]				rs2		rs1	funct3	imm[4:1 11]			opcode		
<b>U</b>		imm[31:12]								rd		opcode		
<b>J</b>		imm[20 10:1 11 19:12]								rd		opcode		

#### 3.1.1 指令类型

- R-型为寄存器操作数指令
- I-型为短立即数操作或取数(Load)指令
- S-型为存数指令
- B-型为条件跳转指令
- U-型为长立即数操作指令
- J-型为无条件跳转指令

#### 3.1.1 指令格式

- 所有指令格式的低7位都为**操作码字段** opcode
- 字段 rd, rs1 和 rs2 给出的是**通用寄存器编号**
  - 因为RISC-V架构共有32个32位通用寄存器x0~x31，因而寄存器编号占5位
  - 0号寄存器x0的内容永远是0
- imm 字段给出的是一个**立即数**，其位数在括号[]中表示
- 字段 funct3 和 funct7 分别表示3位和7位**功能码**，它们和 opcode 一起定义指令的操作功能

## 3.2 RTL语言

寄存器传送级(Register Transfer Level, RTL), RTL语言约定:

- R[r] 表示通用寄存器 r 的内容
- M[addr] 表示储存单元addr的内容
- M[R[r]] 表示寄存器r的内容所指储存单元的内容
- PC 表示PC的内容
- M[PC] 表示PC所指储存单元的内容
- SEXT[imm] 表示对立即数imm进行符号扩展
- ZEXT[imm] 表示对立即数imm进行零扩展
- 传送方向用<--表示，即传送源在右，传送目的在左

## 3.2 基础整数指令集RV32I

### 3.2.1 整数运算类指令

- U型指令一共有两条：

```
lui rd, imm20 #R[rd] <- imm20 || 000H //符号||表示拼接
```

该指令和 addi rd, rs1, imm12 指令结合，可以实现对一个32位变量赋初值。

当 imm12 第一位为1的时候，注意符号扩展带来的影响。

```
auipc rd, imm20 #将立即数imm20加到PC的高20位上，结果存rd
```

可用指令 auipc x10, 0 将PC的值存入寄存器x10中。

- I型指令助剂符都带 i，表示一个操作数为立即数

```
addi rd, rs1, imm12 #R[rd] <- R[rs1] + SEXT[imm12]
```

因为addi指令可以直接加一个负数，因而无需提供subi指令

- R型指令的两个操作数所在的寄存器总是rs1和rs2，结果寄存器为rd

```
sub rd, rs1, rs2 #R[rd] <- R[rs1] - R[rs2]
```

- RV32I指令集提供了(and, andi)、或(or, ori)、异或(xor, xori)三种共6条逻辑运算指令
- 移位指令

```
sll rd, rs1, rs2 #R[rd] <- R[rs1] << R[rs2]  
slli rd, rs1, shamt #R[rd] <- R[rs1] << shamt
```

因为算术左移和逻辑左移的结果完全相同，所以RV32I中没有算术左移指令。

- RV32I提供了4条比较指令：带符号整数小于(slt、slt)和无符号整数小于(sltu、slt)。

```
slt u rd, rs1, imm12 #R[rd] <- R[rs1] < SEXT[imm12]
```

imm12的值应该小于2048，因为slt将imm12符号扩展后按无符号比较。

### 3.2.2 控制转移类指令

- 6条分支指令采用B型格式，其功能为：若rs1和rs2两个寄存器内容比较满足条件，则跳转到目标地址处执行；否则，执行下一条指令。
  - 比较条件包括相等(beq)、不等(bne)、带符号整数小于(blt)、带符号整数大于(bge)、无符号整数小于(bltu)、无符号整数大于(bgeu)。
  - 因指令宽度总是4(RV32G)或2(RV32C)字节，因而总是2的倍数，即指令地址的最低位总是0。因此，转移目标地址 = PC + SEXT[imm[12:1]<<1]。
- RV32I中的两条跳转并链接(Jump & Link)指令jal和jalr分别采用J型指令和I型指令格式。

```
jal rd, imm20 #PC <- PC + SEXT[imm[20:1] << 1]; R[rd] <- PC + 4
```

- 若将返回地址(PC+4)保存到寄存器x1，则可实现过程调用；

- 若目的寄存器rd指令为x0(x0永远为0, 不可更改),则可实现无条件跳转。

```
jalr rd, rs1, imm12 #PC <- R[rs1] + SEXT[imm12]; R[rd] <- PC + 4
```

将目的寄存器rd设为x0时, 可以实现switch-case语句的地址跳转;

若先通过U型指令装入rs1, 则可以实现32位地址空间的绝对或相对跳转。

### 3.2.3 储存器访问类指令

```
lw rd, rs1, imm12 #R[rd] <- M[R[rs1] + SEXT[imm12]]  
sw rs1, rs2, imm12 #M[R[rs1] + SEXT[imm12]] <- R[rs2]
```

### 3.2.4 系统控制指令

# Chapter 03 中央处理器

## 1 CPU概述

### 1.1 CPU基本功能

**指令周期**: CPU取出并执行一条指令的时间

一条指令的执行过程:

- 取指令: 以PC为地址访问主存取出指令送到**指令寄存器(IR)**
- 对IR中的指令操作码译码并计算下一跳指令地址
- 计算源操作数地址并取源操作数
  - **储存器数据**: 需要一次或多次访存
  - **寄存器数据**: 直接从寄存器取数
- 对操作数进行相应的运算
- 计算目的操作数地址并存结果

上述过程中的第一第二步, 所有的指令都一样, 都是取指令、指令译码和修改PC;

剩余步骤由第二步译码得到的控制信号控制。

上述基本操作可以用**寄存器传送级(Register Transfer Level, RTL)**语言描述。

### 1.2 CPU基本组成

两大基本组成部分: **数据通路(datapath)**和**控制器/控制部件(control unit)**。

**执行部件(execution unit)**: 数据通路中专门进行数据运算的部件, 包括**组合逻辑元件/操作元件**和**储存元件/状态元件**。

**数据通路(datapath)**: 由操作元件和储存元件通过总线或分散连接方式连接而成的进行数据存储、处理和传送的路径。

#### 1.2.1 组合逻辑元件

组合逻辑元件属于组合逻辑电路, 输出只取决于当前输入。

常用的组合逻辑元件:

**多路选择器(MUX)**, 需要控制信号Select; **加法器(Adder)**; **算术逻辑部件(ALU)**, 需要控制信号OP

#### 1.2.2 状态元件

状态元件属于时序电路, 具有储存功能, 输入状态在时钟控制下被写入电路中, 并保持电路的输出值不变, 直到下一个时钟到达。输入状态由时钟信号决定何时被写入, 输出状态可以随时读出。

**建立时间(setup time)**: 在时钟下降沿 (以下降沿触发为例) 到达前的一段输入端D的状态必须稳定有效的时间

**保持时间(hold time)**: 时钟下降沿 (以下降沿触发为例) 到达后的一段输入端D的状态必须保持稳定不变的时间

**锁存延迟(Clk-to-Q time)**: 满足建立时间和保持时间约束后输出端状态Q转变为D所需的时间

**暂存寄存器**: 有一个写使能(Write Enable, WE), 当WE为1时, 时钟信号(Clk)边沿到来时, 经过Clk-to-Q时间的延迟, 输出端(DataOut)开始变为输入端(DataIn)的值, 表示输入信息被写入寄存器。如果数据通路中某个寄存器在每个时钟周期到来时都需要写入信息, 则无需WE信号。

**通用寄存器组(General Purpose Register set, GPRs)**: 32个暂存寄存器可以构成一个通用寄存器组, 每个寄存器地址是一个5位二进制编码, 它有两个读口: 32位 busA 和 busB 分别由5位 RA 和 RB 给出地址; 一个写口: 32位 busW 上的信息写入由5位 RW 给出地址的寄存器。读操作为组合逻辑, 经过一个取数时间的延迟, 在busA和busB上的信息便开始有效; 写操作为时序逻辑, 需要时钟信号Clk的控制, 在WE为1时经过Clk-to-Q延迟, 从busW传来的值便开始写入RW指定的寄存器中。

## 1.3 数据通路与时序控制

**早期计算机的三级时序系统**: 机器周期、节拍和脉冲

**现代计算机的时钟信号**: 一个时钟周期就是一个节拍。数据通路可以看成由组合逻辑元件和状态元件交替组成, 只有状态元件能够储存信息, 所有状态元件在同一时钟控制下写入信息。假定各级组合逻辑电路的传输延迟(即最长延迟)为longest delay, 考虑时钟偏移clock skew, 则数据通路的时钟周期应为cycle time = Clk-to-Q + longest delay + setup time + clock skew。假定各级组合逻辑电路的最短传输延迟为shortest delay, 为了使数据通路能够正常工作, 则应满足以下时间约束: Clk-to-Q + shortest delay > hold time。

## 1.4 计算机性能与CPU时间

**吞吐率(throughput)**: 单位时间内所完成的工作量

**带宽(bandwidth)**: 单位时间内传输的信息量

**响应时间(response time)**: 从作业提交开始到作业完成所用的时间

**执行时间(execution time)和等待时间(latency)**: 一个任务所用时间的度量值

用户感觉到的执行时间为: **用户CPU时间/CPU执行时间和其他时间**。

CPU时间计算需要的指标和概念:

- **时钟周期(clock cycle)**: 用于CPU操作同步定时的时钟信号的宽度
- **时钟频率(clock rate)**: 时钟周期的倒数, 又称**主频**
- **CPI(Cycles Per Instruction)**: 执行一条指令所需的时钟周期数

用户CPU时间 = 程序总时钟周期数 / 时钟频率 = 程序总时钟周期数 \* 时钟周期

程序总时钟周期数 = 程序总指令数 \* CPI (指令CPI相同) =  $\Sigma(C * CPI)$

平均CPI = 程序总时钟周期数 / 程序总指令条数 =  $\Sigma(F * CPI)$

## 2 单周期CPU

### 2.1 指令功能描述

**R型指令**

指令	功能
add rd, rs1, rs2	M[PC], PC <- PC + 4 R[rd] <- R[rs1] + R[rs2]
slt rd, rs1, rs2	if (R[rs1] < R[rs2]) R[rd] <- 1 (有符号数比较) else R[rd] <- 0
sltu rd, rs1, rs2	if (R[rs1] < R[rs2]) R[rd] <- 1 (无符号数比较) else R[rd] <- 0

### I-型指令

指令	功能
ori rd, rs1, imm12	R[rd] <- R[rs1]   SEXT(imm12)
lw rd, rs1, imm12	Addr <- R[rs1] + SEXT(imm12) R[rd] <- M[Addr]

### U-型指令

指令	功能
lui rd, imm20	R[rd] <- imm20    000H

### S-型指令

指令	功能
sw rs1, rs2, imm12	Addr <- R[rs1] + SEXT(imm12) M[Addr] <- R[rs2]

### B-型指令

指令	功能
beq rs1, rs2, imm12	Cond <- R[rs1] - R[rs2] if (Cond eq 0) PC <- PC + SEXT(imm12) * 2

### J-型指令

指令	功能
jal rd, imm20	R[rd] <- PC + 4 PC <- PC + SEXT(imm20) * 2

## 2.2 设计

- 扩展部件的设计

实现立即数的扩展操作：immJ、immU、immS、immB、immI

- 算术逻辑部件的设计
- 取指令部件的设计
  - 根据PC从主存取指令
  - 下地址逻辑，顺序加4，跳转看具体指令
- 数据通路的设计：由易到难不断扩展数据通路
  - R-型指令：寄存器 + ALU
  - I-型指令：增加扩展器
  - U-型指令：增加扩展器的扩展方式
  - Load/Store指令：增加扩展器的扩展方式，并且增加数据储存器
  - B-型指令：增加PC、指令储存期和下地址逻辑
  - J-型指令：修改下地址逻辑
- 控制器设计
  - 分析每个指令执行所需的控制信号

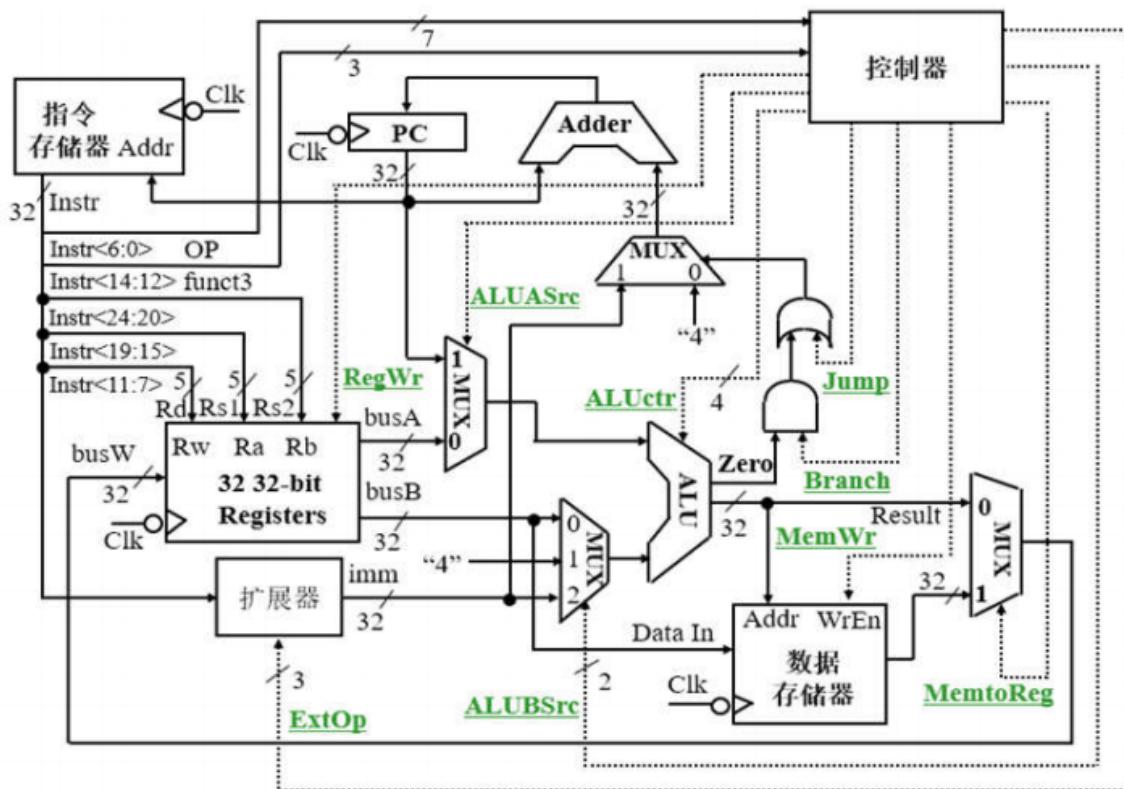


图 1 RV32I 单周期处理器原理图

- 时钟周期的确定

- CPU执行时间由指令数目、时钟周期和CPI决定
- 指令数目由编译器和指令系统决定
- 单周期CPU的CPI为1
- 时钟周期通常取最复杂的指令的执行周期，在给出的9条指令中，最复杂的是lw指令
  - PC所存延迟(Clk-to-Q) + 取指令时间 + 寄存器取数时间 + ALU延迟 + 储存器取数时间 + 寄存器建立时间 + 时钟偏移

## 3 多周期CPU

基本思想：

- 把每条指令的执行分成多个阶段，每个阶段在一个或多个时钟周期内完成
  - 取指令并计算下条指令地址
  - 译码并取数
  - 执行指令
- 每个时钟周期称为一个状态，期间最多完成一次访存或一次寄存器读写或一次ALU操作
- 每个时钟内的执行结果在下个时钟到来时，保存到相应储存元件或稳定地保持在组合电路中
- 时钟周期的宽度以最复杂阶段所用的时间为准

**多周期控制器：**通常采用基于有限状态机的描述和微程序描述两种方式

**有限状态机控制器/组合逻辑控制器/PLA控制器/硬连线控制器：**输入：指令操作码OP + 状态寄存器；输出：控制信号 + 下一状态；输入和输出之间是一个纯组合逻辑，时序逻辑只存在于状态寄存器中

**微程序控制器：**将每条指令的过程用一个**微程序**来表示，每个微程序由若干条**微指令/控制字(Control Word, CW)**/**微命令**(微地址码 + 微操作码)，对应着若干个**微操作**组成，所有指令对应的微程序都存放在一个ROM中，这个ROM称为**控制储存器/控存(Control Storage, CS)**，控存中的信息称为**微代码**。优点：更灵活；缺点：更慢。

**带异常处理的CPU设计：**当CPU在执行当前程序或任务（即用户进程）的第*i*条指令时，若检测到一个异常事件，或在执行第*i*条指令后发现有一个中断请求信号，则CPU会中断当前程序的执行，跳转到操作系统中相应的异常或中断处理程序去执行。若异常或中断处理程序能够解决相应问题，则在异常或中断处理程序的最后，CPU通过执行“异常/中断”返回指令回到被打断的用户进程第*i*条或*i*+1条指令继续执行；若异常或中断处理程序发现的是不可恢复的致命错误，则终止用户进程。

## 4 流水线CPU设计

### 4.1 流水线CPU概述

**指令流水线：**一般由如下5个流水段组成

- **取指令(Instruction Fetch, IF)：**从储存器取指令
- **指令译码(Instruction Decode, ID)：**产生指令所需的控制信号
- **取操作数(Operand Fetch, OF)：**读取储存器操作数或寄存器操作数
- **执行(Execution, EX)：**对操作数完成相应操作
- **写回(Write Back, WB)：**将操作结果写回储存器或寄存器

后一条指令的第*i*步和前一条指令的第*i*+1步同时进行，从而使完成一串指令总的完成时间大为缩短。

例如对于上述五段流水线指令，完成4条，以流水线方式只需8个时钟周期，若以非流水线方式，则需20个时钟周期。

理想情况下，每个时钟都有一条指令进入流水线，每个时钟都有一条指令完成，则每条指令的CPI为1。

规整、简单、一致等特性有利于指令的流水线执行。

## 4.2 指令的流水段分析

- 取指令(IF): 从主存取指令并计算PC+4
- 指令译码(ID): 产生指令执行所需的控制信号，并生成操作数，包括寄存器取数和立即数扩展
- 执行(EX): 对操作数完成指定操作
- 访存(M): 访问储存器
- 写回(WB): 将结果写回寄存器

指令类型	功能段划分
R-型指令	IF + ID + EX + WB
I-型指令	IF + ID + EX + WB
U-型指令	IF + ID + EX + WB
lw指令	IF + ID + EX + M + WB
S-型指令	IF + ID + EX + M
B-型指令	IF + ID + EX + M
J-型指令	IF + ID + EX + WB

## 4.3 流水线数据通路的设计

**流水段寄存器**: 用于存放从当前流水段传到后面所有流水段的信息

IF是公共流水段，不需要控制信号，其余段的控制信号如下：

- ID段
  - 扩展器类型(ExtOp): 3位编码
- EX段
  - ALU的A口来源(ALUASrc): 1, 来源于PC; 0来源于busA
  - ALU的B口来源(ALUBSrc): 00, 来源于busB; 01, 来源于常数4; 10, 来源于扩展器
  - ALU运算类型(ALUctr): 4位编码
- M段
  - 数据储存器DM的写信号(MemWr): S-型指令时为1, 其他指令为0
  - 是否为B-型分支指令(Branch): B-型指令时为1, 其他指令为0
  - 是否为J型跳转指令(Jump): J-型指令时为1, 其他指令为0
- WB段
  - 寄存器的写入源(MemtoReg): 1, DM输出; 0, ALU输出
  - 通用寄存器写信号(RegWr): 结果写寄存器的指令都为1, 其他指令为0

## 5 流水线冒险及其处理

### 5.1 结构冒险

**结构冒险(structural hazard)/硬件资源冲突(hardware resource conflict)**: 同一个部件同时被不同指令所使用的硬件资源竞争

解决：

- 流水线(段划分原则->一个部件每条指令只能用一次)
- 设置多个独立部件(比如将Rrd和Rwr独立开来)

## 5.2 数据冒险

**数据冒险(data hazard)/数据相关(data dependencies):** 后面指令用到前面指令的结果时，前面指令的结果还没产生(比如说，第一条指令的结果要在WB段产生，但是第二条指令的ID段就需要第一条指令的结果了，而此时第一条指令才刚刚运行到EX段，结果还未产生)，这种冒险称为**写后读(Read After Write, RAW)数据冒险**。

可采取的措施：

- 插入空操作指令nop
  - 控制简单
  - 但浪费了指令储存空间和指令执行时间
- 插入气泡(bubble)
  - 通过硬件阻塞(stall)的方式阻止后续指令执行
- 转法(forwarding)/旁路(bypassing)技术
  - 将数据通路中生成的中间数据直接发送到ALU的输入端
- Load-use数据冒险的检测和处理
  - | lw指令之后跟R-型指令或I-型指令的相关性问题
    - 在load指令之前插入nop指令来解决
    - 调整指令顺序以避免Load-use现象

## 5.3 控制冒险

**控制冒险(control hazard):** 由于发生了执行顺序改变而引起的流水线阻塞

**分支冒险(branch hazard):** 由于指令分支而引起的控制冒险

- 软件阻塞：插入nop
- 硬件阻塞：插入stall
- 静态预测(static prediction)/简单预测和动态预测(dynamic prediction)
  - 动态的准确率可达90%
- 延迟分支(delayed branch): 将和分支指令无关的指令放到分支指令之后执行，避免阻塞的情况，由编译器完成此过程
- 异常和中断引起的控制冒险

## 6 高级流水技术

---

**超流水线(super-pipelining)技术：**通过增加流水线技术来使更多的指令同时在流水线中重叠执行

**多发射流水线(multiple issue pipelining)技术：**同时发射多条指令的流水线，数据通路中有多个执行部件

**静态多发射：** VLIW(Very Long Instruction Word, 超长指令字)结构、编译器静态推测

**动态多发射：**超标量结构、硬件动态推测调度



# Chapter 04 运算方法和运算部件

---

## 1 基本运算部件

---

### 1.1 串行进位加法器

**全加器(Full Adder, FA)**: 将两个本位数( $X_n, Y_n$ )和低位进位( $C_{n-1}$ )相加，以生成一位本位和( $F_n$ )以及一位高位进位( $C_n$ )

**行波进位加法器(Carry Ripple Adder, CPA)**:  $n$ 为串行进位加法器，由 $n$ 个全加器串行组成，当 $n$ 较大时，串行进位加法器的速度将显著变慢

### 1.2 并行进位加法器

**进位传递函数**:  $P_i = X_i + Y_i$ ，若 $X_i$ 和 $Y_i$ 有一个为1时，低位进位输入一定会被传递到高位

**进位生成函数**:  $Q_i = X_i * Y_i$ ，当 $X_i$ 和 $Y_i$ 均为1时，不管有无低位进位输入，都会有高位进位

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$$

**先行进位部件(Carry Lookahead Unit, CLU)**: 实现上述逻辑表达式的电路

**先行进位加法器(Carry Lookahead Adder, CLA)**: 通过先行进位方式实现的加法器

### 1.3 带标志加法器

**溢出标志**:  $OF = C_n \text{ Xor } C_{n-1}$

**符号标志**:  $SF = F_{n-1}$

**零标志**:  $ZF = 1 \text{ iff. } F = 0$

**进位/借位标志**:  $CF = Cout \text{ Xor } Cin$

### 1.4 算术逻辑部件

A、B:  $n$ 位操作数输入端

$Cin$ : 进位输入端

ALUop: 操作控制端

## 2 定点数运算

---

## 2.1 补码加减法运算

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

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

## 2.2 定点乘法/除法运算

ALU + (移位)寄存器 + 计数器 + 控制逻辑

# Chapter 05 FPGA设计和硬件描述语言

---

## 1 可编程逻辑器件

---

**可编程逻辑器件(Programmable Logic Device, PLD)**: 主要由与阵列和或阵列构成，逻辑门可以通过编程开关连接，以形成所需要的逻辑电路。

**可编程只读储存器(Programmable Read Only Memory, PROM)**: 与阵列固定，或阵列可编程的简单PLD，将逻辑表达式转化成标准与-或表达式之后可以很容易的由PROM实现。

**可编程逻辑阵列(Programmable Logic Array, PLA)**: 与阵列和或阵列都可以编程的逻辑阵列。用PLA实现逻辑函数时，只需要将逻辑表达式转化成最简与-或表达式即可。

**可编程阵列逻辑(Programmable Array Logic, PAL)**: 与阵列可编程，或阵列固定的PLD。

**通用阵列逻辑(Generic Array Logic, GAL)**: 可擦写、可重复编程、设置加密、输出端设置了可编程的逻辑宏单元(Output Logic Macro Cell, OLMC)。

**复杂可编程逻辑器件(Complex Programmable Logic Device, CPLD)**: **逻辑阵列块(Logic Array Block, LAB)** + I/O控制块 + 可编程互联阵列(PIA)。

**现场可编程门阵列(Field Programmable Gate Array, FPGA)**: 基于查找表(Look-Up Table, LUT)技术构建的集成度更高的CPLD。

## 2 储存器阵列

---

**随机存取储存器(RAM)**:

- **静态RAM(Static RAM, SRAM)**: MOS管较多，占硅片面积大，因而价格高、功耗大、集成度低；但无需刷新和读后再生；特别是它读写速度快，其储存原理可看作RS触发器的读写过程
- **动态RAM(Dynamic RAM, DRAM)**: MOS管少，占硅片面积小，因而价格便宜、功耗小、集成度高；但必须定时刷新和读后再生；特别使它的读写速度相对于SRAM慢，其储存原理可看作电容的充放电过程

**只读存储器(Read Only Memory, ROM)**

## 3 专用集成电路

---

**专用集成电路(Application-Specific Integrated Circuit, ASIC)**: 应特定用户要求和特定电子系统的需要而设计、制造的集成电路



# Chapter 06 时序逻辑电路

## 1 时序逻辑电路概述

### 1.1 时序逻辑与有限状态机

**时序逻辑电路**: 结果输出不仅取决于当前的外部输入，还取决于系统所处的内部状态。

**有限状态机(Finite State Machine, FSM)**: 刻画状态及状态转化的理论工具，常用状态图描述有限状态机。

### 1.2 时序逻辑电路的基本结构

- 状态记忆模块
  - **同步时序逻辑电路**: 状态记忆单元在统一的时钟信号控制下进行状态转换
  - **异步时序逻辑电路**: 状态记忆单元没有统一的时钟信号来控制其状态改变
- 次态激励逻辑模块F
  - **激励函数**: 次态激励逻辑模块对应的函数，是一个组合逻辑
- 输出逻辑模块G
  - **输出函数**: 输出模块对应的函数，是组合逻辑
  - **Mealy型电路**: 输出逻辑不仅依赖于当前的状态，同时还依赖当前的输入
  - **Moore型电路**: 输出逻辑仅依赖当前状态，和当前输入无关

### 1.3 时序逻辑电路的定时

**时钟信号**: 用于触发时序逻辑电路中状态的转换

**时钟周期**: 由高电平和低电平两部分组成，时钟周期的倒数称为**时钟频率**

时序逻辑电路的状态转换多采用时钟边沿触发方式，分为**上升沿触发**和**下降沿触发**。

## 2 锁存器和触发器

**双稳态元件**: 两个非门串联后输出再反馈到输入端，Q高电平时为置位状态， $\sim Q$ 为高电平时为复位状态

**SR锁存器**: 使用一对交叉耦合的或非门构成的储存元件，也称为**置位-复位锁存器**，S是置位(set)输入端，R是复位(reset)输入端。当仅S为1时，Q一定为1；当仅R为1时，Q一定为0；当S和R都为1时，Q和 $\sim Q$ 都为0，是一个禁止状态；当S和R都为0时，保持原先状态不变，如果原先是禁止状态，则此时状态未知

**触发延迟/锁存延迟**: 从输入驱动信号有效开始，到输出达到稳定为止的延时

**D锁存器**: C为使能信号，D为唯一的状态驱动信号。当C为1时，Q与D保持一致；当C为0时，Q状态保持不变

**D触发器**: 采用时钟边沿触发机制，由两个D锁存器依主从结构构成

**建立时间(setup time)**: 在时钟触发边沿到来之前输入端D必须稳定的最短时间

**保持时间(hold time)**: 在时钟触发边沿到来之后输入端必须保持不变的最短时间

**锁存延迟(latch prop or Clk-to-Q time)**: 时钟触发边沿到来之后输入端Q改变为D的当前输入值的时间

**带使能端的D触发器**: En为1时, 和D触发器功能相同; En为0时, 保持原来状态不变

**带复位功能的D触发器**: Rst信号为1时, Q为0

**T触发器**: 每个时钟脉冲的出发边沿到来后都会改变状态, 可用D触发器实现, 冲用于实现分频器或计数器功能

## 3 同步时序电路设计

---

需求分析、状态图/状态表设计、状态化简、状态编码、电路设计、电路分析

**状态图/状态表**: 考虑现态、输入和次态、输出之间的关系; 具有互斥性(从每个状态出发的所有状态的转换条件是互斥的)和完备性(从每个状态出发的所有状态的转换条件的逻辑值或等于1)

**状态化简**: 将等价状态合并, 以达到减少状态个数的目的; 等价指的是两个状态在所有的输入情况下, 输出都相同, 次态都相同或等价

**状态编码**: 假定一个有限状态机的状态数为N, 则状态数M必须满足 $2^M \geq N$ ; M为=位编码中的每一位对应一个状态变量, 每个状态变量的次态都是关于输入变量和所有状态变量的逻辑函数, 即激励函数

## 4 典型时序逻辑部件设计

---

### 4.1 计数器

**行波计数器(ripple counter)**: 串行计数器, 采用多个T触发器串联实现, 慢, 本质上是一个异步时序逻辑。

**并行计数器**: 使用带使能端的T触发器构成, 基本原理是只有当低位都是1时, 本位才会在下一次时钟到来时变化, 本质上是一个同步时序逻辑

**寄存器**: 用来暂存信息的逻辑部件, 由若干个D触发器并行构成

**寄存器堆(register file)/通用寄存器组(General Purpose Register set, GPRs)**: 暂存指令执行过程中用到的中间数据

**移位寄存器**: 由若干个D触发器串行构成, 串行输入, 串行/并行输出

**桶形移位寄存器**: 可以根据移位位数控制端的设置对数据左移或右移指定位数, 是一种组合逻辑, 需要指明算术移位还是逻辑移位



# Chapter 07 组合逻辑电路

---

## 1 组合逻辑概述

**组合逻辑电路(combinational logic circuit)**: 输出值仅依赖于当前输入值

**时序逻辑电路(sequential logic circuit)**: 输出值不仅依赖于当前值，还与之前的输入有关

由若干元件和若干节点构成的电路是组合逻辑电路，应同时满足以下三个规则：

- 每个元件本身是组合逻辑电路
- 不存在一个结点同时是两个元件的输出结点或同时被两个元件的输出信号所驱动
- 不存在从一个输入端经过若干元件和中间节点连到一个输出端，然后又从该输出端连到该输入端的回路

**扇入系数**: 一个逻辑门的输入端个数的最大值

**扇出系数**: 一个逻辑门输出端信号所能驱动的下一级门的数量的最大值

**门延迟(gate delay)**: 从逻辑门的输入信号改变开始到输出信号发生改变所用的时间

**非法值**: 同时被高低电平驱动的节点会使电路不断震荡发热

**三态门(three-state gate)/三态缓冲器**: 输出值除了0和1之外，还可以是高阻态，相当于和所连接的总线断开

## 2 典型组合逻辑部件设计

**译码器(decoder)**: 将二进制编码（如地址码、指令码）翻译为热点(one-hot)编码，最常见的是 $n-2^n$ 译码器

**地址译码器**: 主存中的地址译码器根据输入的地址选择一个对应的输出线进行驱动

**编码器(encoder)**: 输入端通常是多个独立信号，输出端是这些独立信号中的一个有效信号的编码，最常见的编码器就是 $2^n-n$ 编码器，也称二进制编码器

**多路选择器(multiplexer, MUX)/复用器/数据选择器**: 从多个可能的输入中选择一个直接输出，最简单的多路选择器是**二路选择器**，有两个输入端，一个输出端，一个控制端

**多路分配器(demultiplexer, DMUX/DEMUX)**: 把唯一的输入端发送到多个输出端中的某一个

**半加器(Half Adder, HA)**: 不考虑低位进位，只考虑两个加数的一位加法器； $F = A \text{ xOr } B$ ,  $Cout = A * B$

**全加器(Full Adder, FA)**: 同时考虑两个加数和低位进位的一位加法器； $F = A \text{ xOr } B \text{ xOr } Cin$ ,  $Cout = A * B + A * Cin + B * Cin$

## 3 组合逻辑电路的时序分析

**传输延迟(propagation delay, Tpd)**: 从输入端的变化开始到所有输出端得到最终稳定的信号需要的最长时间

**最小延迟(contamination delay, Tcd):** 从输入端的变化开始到任何一个输出开始发生改变所需的最短时间

**关键路径:** 一个组合电路在输入和输出之间经过的最长路径

**组合电路的传输延迟:** 关键路径上所有元件的传输延迟之和

**最短路径:** 一个组合电路在输入和输出之间经过的最短路径

**组合电路的最小延迟:** 最短路径上所有元件的最小延迟之和

**竞争(race):** 某个输入信号经过两条或两条以上的路径作用到输出端，由于每条路径的延迟不同，因而这个输入信号对输出信号就会发生前后不同的影响

**毛刺(glitch):** 电路输出信号中出现的不正确的尖峰信号

**冒险(hazard)/竞争冒险:** 一个组合电路中有毛刺出现，则说明该电路存在冒险



# Chapter 08 数字逻辑基础

---

## 1 逻辑门和数字抽象

---

**逻辑门(logic gate)**: 具有允许或禁止信号传输功能的门电路

**真值表**: 描述输入输出对应关系的二维表

**数字抽象**: 将一定范围内的电压映射到高态和低态，并用0和1来表示，对应关系分**正逻辑(positive logic)**和**负逻辑(negative logic)**，既不能识别为0也不能识别为1的状态为**不确定状态**

## 2 CMOS晶体管

---

**CMOS(Complementary Metal-Oxide Semiconductor)**: 基于金属氧化物半导体场效应晶体管，CMOS晶体管以互补的形式公用一对NMOS和PMOS晶体管

**n沟道晶体管(n-channel MOS(NMOS) transister)**:  $V_{gs}$ 低时， $R_{ds}$ 大； $V_{gs}$ 高时， $R_{ds}$ 小

**p沟道晶体管(p-channel MOS(PMOS) transister)**:  $V_{gs}$ 高时， $R_{ds}$ 大； $V_{gs}$ 低时， $R_{ds}$ 小

**转换时间(transition time)**: 数字电路信号从一种状态改变为另一种状态所需要的时间，分为**上升时间(rise time, tr)**和**下降时间(fall time, tf)**

**传输延迟(propagation delay)**: 从输入信号发生变化到输出信号发生变化所需要的时间，过程中所经历的电气通路称为**信号通路(signal path)**

**静态损耗**: 输出信号保持不变时的功率损耗

**动态损耗**: 输出信号高低状态转换时的功率损耗

## 3 布尔代数

---

**最小项/标准乘积项**: 每个逻辑变量出现且仅出现一次的乘积项

**最大项/标准求和项**: 每个逻辑变量出现且仅出现一次的求和项

**标准与-或表达式(Sum of Product, SOP)**: 函数输出值为1的输入组合最小项之和

**标准或-与表达式(Product of Sum, POS)**: 函数输出值为0的输入组合所对应的最大项之积

# Chapter 09 二进制编码

## 1 进位计数值

可使用后缀字母标识数的进位计数制，一般用B(Binary)表示二进制，用O(Octal)表示八进制，用D(Decimal)表示十进制（十进制数的后缀可以省略），用H(Hexadecimal)表示十六进制，有时也在十六进制数之前用0x作为前缀

R进制数转十进制：“按权展开”

十进制数转R进制：

- 整数部分：“除积取余，上低下高”
- 小数部分：“乘积取整，上高下低”

### 二、十六进制数的相互转换

0H = 0000B	1H = 0001B	2H = 0010B	3H = 0011B
4H = 0100B	5H = 0101B	6H = 0110B	7H = 0111B
8H = 1000B	9H = 1001B	AH = 1010B	BH = 1011B
CH = 1100B	DH = 1101B	EH = 1110B	FH = 1111B

## 2 数值型数据的表示

**机器数**：数值型数据在计算机内部编码表示后的数

**真值**：机器数的在现实世界中带有正负号的真正的值

**原码/“符号-数值”(sign and magnitude)表示法**：由符号位后直接跟数值位构成，原码的优点是与真值的对应关系直观、方便；其缺点是0的表示不唯一，并且原码运算中符号和数值部分必须分开处理

**补码/“2-补码”(two's complement)表示法**： $[X]_{\text{补}} = M + X \pmod{M}$ ，对于正数，符号位取0，其余同真值中的相应各位；对于负数，符号位取1，其余各位由数值部分“各位取反，末位加1”得到。

### IEEE745浮点数格式

- 32位单精度格式：

符号s	阶码e	尾数f
1位	8位	23位

◦ 计算公式： $(-1)^s \times 1.f \times 2^{(e - 127)}$

- 64位双精度格式：

符号s	阶码e	尾数f
1位	11位	52位

- 计算公式:  $(-1)^s \times 1.f \times 2^{(e - 1023)}$

阶码采用移码方式,  $[E]_{\text{移}} = E + \text{偏置常数(bias)}$ , IEE745的偏置常数为 $2^{(n-1)} - 1$ , 即127和1023

**十进制数的二进制编码(Binary Coded Decimal, BCD)**: 有权BCD码即8421码, 用二进制编码十进制数的每个数位

## 3 数据的宽度和储存

---

**比特(bit)**: 一位0或1, 是组成二进制数的最小单位

**字节(Byte)/位组**: 计算机内部二进制信息的计量单位, 一个字节等于8个比特

**字长**: CPU内部用于整数运算的运算器位数和通用寄存器的宽度

**最低有效字节(Least Significant Byte, LSB)**: 小端方式从数据的最低有效字节开始存放, 即变量的地址是LSB所在的地址

**最高有效字节(Most Significant Byte, MSB)**: 大端方式从数据的最高有效字节开始存放, 即变量的地址是MSB所在的地址

# 第1章 二进制编码

作业：习题 3、4、5、6、8、9、17

3. 实现下列各数的转换。

- (1)  $(25.8125)_{10} = (?)_2 = (?)_8 = (?)_{16}$
- (2)  $(101101.011)_2 = (?)_{10} = (?)_8 = (?)_{16} = (?)_{8421}$
- (3)  $(0101\ 1001\ 0110.0011)_{8421} = (?)_{10} = (?)_2 = (?)_{16}$
- (4)  $(4E.C)_{16} = (?)_{10} = (?)_2$

## 【分析解答】

- (1)  $(25.8125)_{10} = (1\ 1001.1101)_2 = (31.64)_8 = (19.D)_{16}$
- (2)  $(101101.011)_2 = (45.375)_{10} = (55.3)_8 = (2D.6)_{16} = (0100\ 0101.0011\ 0111\ 0101)_{8421}$
- (3)  $(0101\ 1001\ 0110.0011)_{8421} = (596.3)_{10} = (1001010100.010011...)_2 = (254.4...)_{16}$
- (4)  $(4E.C)_{16} = (78.75)_{10} = (100\ 1110.11)_2$

4. 假定机器数为 8 位（1 位符号，7 位数值），写出下列各二进制数的原码表示。

+0.1001, -0.1001, +1.0, -1.0, +0.010100, -0.010100, +0, -0

## 【分析解答】

上述各二进制数的原码和补码（小数模为 2，但一般表示整数）表示见下表。

小数的原码和补码表示

数值	原码	补码
+0.1001	0.1001000	0.1001000
-0.1001	1.1001000	1.0111000
+1.0	溢出	溢出
-1.0	溢出	1.0000000
+0.010100	0.0101000	0.0101000
-0.010100	1.0101000	1.1011000
+0	0.0000000	0.0000000
-0	1.0000000	0.0000000

5. 假定机器数为 8 位（1 位符号，7 位数值），写出下列各二进制数的补码和移码表示。

+1001, -1001, +1, -1, +10100, -10100, +0, -0

## 【分析解答】

上述各二进制数的补码和移码表示见下表。

整数的补码和移码表示

数值	补码	移码（偏置常数=1 0000000）
+1001	0 0001001	1 0001001
-1001	1 1110111	0 1110111

+1	0 0000001	1 0000001
-1	1 1111111	0 1111111
+10100	0 0010100	1 0010100
-10100	1 1101100	0 1101100
+0	0 0000000	1 0000000
-0	0 0000000	1 0000000

6. 已知  $[x]_{\text{补}}$ , 求  $x$

- $$(1) [x]_{\text{补}} = 1110\ 0111 \quad (2) [x]_{\text{补}} = 1000\ 0000 \quad (3) [x]_{\text{补}} = 0101\ 0010 \quad (4) [x]_{\text{补}} = 1101\ 0011$$

### 【分析解答】

- (1)  $x = -001\ 1001B = -25$       (2)  $x = -1000\ 0000B = -128$   
 (3)  $x = +101\ 0010B = 82$       (4)  $x = -010\ 1101B = -45$

8. 在 32 位计算机中运行一个 C 语言程序，在该程序中出现了以下变量的初值，请写出它们对应的机器数（用十六进制表示）。

- (1) int x=-32768      (2) short y=522      (3) unsigned z=65530  
(4) char c='@'      (5) float a=-1.1      (6) double b=10.5

### 【分析解答】

- (1)  $-2^{15} = -1000\ 0000\ 0000\ 0000$ B, 故机器数为 1…1 1000 0000 0000 0000=FFFF8000H。

(2) 522=10 0000 1010B, 故机器数为 0000 0010 0000 1010=020AH。

(3)  $65530=2^{16}-1-5=1111\ 1111\ 1111\ 1010$ B, 故机器数为 0000FFFAH。

(4) '@'的 ASCII 码是 40H。

(5)  $-1.1=-1.00011[0011]…$ B= $-1.000\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100$ B, 阶码为 127+0=01111111, 舍入的三位为 110, 因此舍入后尾数末位加 1, 故机器数为 1 01111111 000 1100 1100 1100 1100 1100 1101=BF8CCCCDH。

(6)  $10.5=1010.1$ B= $1.0101$ B $\times 2^3$ , 阶码为  $1023+3=100\ 0000\ 0010$ , 故机器数为 0 100 0000 0010 0101 [0000]=40250000 00000000H。

9. 在 32 位计算机中运行一个 C 语言程序，在该程序中出现了一些变量，已知这些变量在某一时刻的机器数（用十六进制表示）如下，请写出它们对应的真值。

- (1) int x: FFFF0006H    (2) short y: DFFCH    (3) unsigned z: FFFFFFFFAH  
(4) char c: 2AH    (5) float a: C4480000H    (6) double b: C024800000000000H

### 【分析解答】

- (1) FFFF0006H=1…1 0000 0000 0000 0110B, 故  $x = -1111\ 1111\ 1111\ 1010B = -(65535-5) = -65530$ 。  
 (2) DFFCH=1101 1111 1111 1100B=-010 0000 0000 0100B, 故  $y = -(8192+4) = -8196$ 。  
 (3) FFFFFFFAH=1…1 1010B, 故  $z = 2^{32}-6$ 。

(4)  $2AH=0010\ 1010B$ , 故  $c=42$ , 若  $c$  表示字符, 则  $c$  为字符'\*'。

(5)  $C4480000H=1100\ 0100\ 0100\ 1000\ 0\dots 0B$ , 阶码为 10001000, 阶为  $136-127=9$ , 尾数为  $-1.1001B$ ,

故  $a=-1.1001B \times 2^9 = -11\ 0010\ 0000B = -800$ 。

(6)  $C024800000000000H=1100\ 0000\ 0010\ 0100\ 1000\ 0\ 0\dots 0B$ , 阶码为 100 0000 0010, 阶为  $1026-1023=3$ , 尾数为  $1.01001B$ , 故  $b = -1.01001B \times 2^3 = -1010.01B = -10.25$ 。

17. 假定在一个程序中定义了变量  $x$ 、 $y$  和  $i$ , 其中,  $x$  和  $y$  是 float 型变量,  $i$  是 16 位 short 型变量 (用补码表示)。程序执行到某一时刻,  $x = -0.125$ 、 $y = 7.5$ 、 $i = 100$ , 它们都被写到了主存 (按字节编址), 其地址分别是 100, 108 和 112。请分别画出在大端机器和小端机器上变量  $x$ 、 $y$  和  $i$  中每个字节在主存的存放位置。

### 【分析解答】

$x = -0.125 = -0.001B = -1.0B \times 2^{-3}$ , 阶码  $e=127-3=0111\ 1111B$  -11B=0111 1100B, 所以, 用 IEEE 754 单精度浮点数表示为: 1 011 1110 0 000 0000 0000 0000 0000 BE00 0000H。

$y=7.5=111.1B=+1.111B \times 2^2$ , 阶码  $e=127+2=128+1=1000\ 0001$ , 所以, 用 IEEE 754 单精度浮点数表示为: 0 100 0000 1 111 0000 0000 0000 0000 0000 40F0 0000H。

$i = 100 = 110\ 0100B$ , 用 16 位补码表示为 0064H。

上述三个数据在大端机器和小端机器上的存放位置如下表所示。

数据在大端和小端机器中的存放位置

地址	大端机器	小端机器
$\&x (100)$	BEH	00H
$\&x + 1 (101)$	00H	00H
$\&x + 2 (102)$	00H	00H
$\&x + 3 (103)$	00H	BEH
$\&y (108)$	40H	00H
$\&y + 1 (109)$	F0H	00H
$\&y + 2 (110)$	00H	F0H
$\&y + 3 (111)$	00H	40H
$\&i (112)$	00H	64H
$\&i + 1 (113)$	64H	00H

## 第2章 数字逻辑基础

作业：习题 3、5、6（1）、7（5）、7（8）、8（1）、8（2）、8（4）、12、13（2）、13（5）

3. 请用完备归纳法证明定理 T2~T5。

### 【分析解答】

可以直接通过列真值表来证明。

空元素：(T2)  $X+1=1$ ；

证明：当  $X=0$  时，等式左边= $0+1=1$ =右边，等式成立；当  $X=1$  时，等式左边= $1+1=1$ =右边，等式成立。无论  $X$  取 0 或 1，等式都成立，T2 定理得以证明。

同一律：(T3)  $X+X=X$ ；

证明：当  $X=0$  时，等式左边= $0+0=0$ =右边，等式成立；当  $X=1$  时，等式左边= $1+1=1$ =右边，等式成立。无论  $X$  取 0 或 1，等式都成立，T3 定理得以证明。

还原律：(T4)  $\bar{\bar{X}}=X$ 。

证明：当  $X=0$  时，等式左边= $\bar{\bar{0}}=0$ =右边，等式成立；当  $X=1$  时，等式左边= $\bar{\bar{1}}=1$ =右边，等式成立。无论  $X$  取 0 或 1，等式都成立，T4 定理得以证明。

互补律：(T5)  $X+\bar{X}=1$ ；

证明：当  $X=0$  时，等式左边= $0+\bar{0}=1$ =右边，等式成立；当  $X=1$  时，等式左边= $1+\bar{1}=1$ =右边，等式成立。无论  $X$  取 0 或 1，等式都成立，T5 定理得以证明。

5. 有人依据德·摩根定理认为逻辑表达式  $X+Y\cdot Z$  的反是  $\bar{X}\cdot \bar{Y} + \bar{Z}$ 。但当  $XYZ==110$  时，这两个函数运算结果都是 1。对于同样的输入组合，这两个函数结果本应相反，错在哪里？

### 【分析解答】

根据德·摩根定理对逻辑表达式进行转换，不能改变原先表达式中的运算次序。因此逻辑表达式  $X+Y\cdot Z$  的反函数是  $\bar{X}\cdot (\bar{Y} + \bar{Z})$ ，当  $XYZ==110$  时， $X+Y\cdot Z=1+1\cdot 0=1+0=1$ ，而  $\bar{X}\cdot (\bar{Y} + \bar{Z})=0\cdot (0+1)=0\cdot 0=0$ ，结果正好相反。

6. 请用布尔代数定理化简下面的逻辑函数。

$$(1) F = W \cdot X \cdot Y \cdot Z \cdot (\bar{W} \cdot X \cdot Y \cdot Z + W \cdot \bar{X} \cdot Y \cdot Z + W \cdot X \cdot \bar{Y} \cdot Z + W \cdot X \cdot Y \cdot \bar{Z})$$

### 【分析解答】

根据布尔定理 T8 分配律，把公共因子  $W \cdot X \cdot Y \cdot Z$  和括号中每个与项进行与运算，再进行或

运算。可以发现，每一个与项中都包含有同一个变量的原变量和反变量，而根据布尔代数定理 T5，它们的运算结果为 0，则  $F=0$ 。

$$F = W \cdot X \cdot Y \cdot Z \cdot \bar{W} \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y \cdot Z \cdot W \cdot \bar{X} \cdot Y \cdot Z + W \cdot X \cdot Y \cdot Z \cdot W \cdot X \cdot \bar{Y} \cdot Z + W \cdot X \cdot Y \cdot Z \cdot W \cdot X \cdot Y \cdot \bar{Z} \quad (\text{T8})$$

$$F = 0 + 0 + 0 + 0 \quad (\text{T5})$$

$$= 0 \quad (\text{A4D})$$

7. 请写出下面各个逻辑函数的真值表。

$$(5) \quad F = \overline{W \cdot X \cdot \bar{Y} + \bar{Z}}$$

#### 【分析解答】

真值表(5)

WXYZ	F
0000	0
0001	0
0010	0
0011	1
0100	0
0101	0
0110	0
0111	1
1000	0
1001	0
1010	0
1011	1
1100	0
1101	0
1110	0
1111	0

$$(8) \quad F = \overline{\overline{A} + \overline{B} + \overline{C} + D}$$

#### 【分析解答】

真值表(8)

ABCD	F
0000	1
0001	0
0010	1

0011	0
0100	1
0101	0
0110	0
0111	0
1000	1
1001	0
1010	0
1011	0
1100	1
1101	0
1110	0
1111	0

8. 请写出下面各个逻辑函数的标准与-或表达式和标准或-与表达式。

$$(1) F(A,B,C)=\sum m(2,4,6,7)$$

#### 【分析解答】

标准与-或表达式：

$$F(A,B,C)=\bar{A} \cdot B \cdot \bar{C} + A \cdot \bar{B} \cdot \bar{C} + A \cdot B \cdot \bar{C} + A \cdot B \cdot C.$$

根据逻辑函数表达式最小项列表集合和最大项列表集合之间的互补关系，函数 F 的最大项列表为  $F=\prod M(0,1,3,5)$ ，标准或-与表达式为： $F(A,B,C)=(A+B+C) \cdot (A+B+\bar{C}) \cdot (A+\bar{B}+C) \cdot (\bar{A}+B+\bar{C})$ 。

$$(2) F(W,X,Y)=\prod M(0,1,3,4,5)$$

#### 【分析解答】

根据逻辑表达式最小项列表和最大项列表之间的转换关系，函数 F 的最小项列表为  $F(W,X,Y)=\sum m(2,6,7)$ ，标准与-或表达式为： $F(W,X,Y)=\bar{W} \cdot X \cdot \bar{Y} + W \cdot X \cdot \bar{Y} + W \cdot X \cdot Y$ 。

标准或-与表达式为： $F(W,X,Y)=\prod M(0,1,3,4,5)$

$$F = (W + X + Y) \cdot (W + X + \bar{Y}) \cdot (W + \bar{X} + \bar{Y}) \cdot (\bar{W} + X + Y) \cdot (\bar{W} + X + \bar{Y}).$$

$$(4) F = \bar{V} + \overline{\bar{W} + X}$$

#### 【分析解答】

根据德·摩根定理把逻辑表达式转换成两级与-或表达式。

$$F = \bar{V} + \overline{\bar{W} + X} = \bar{V} + \bar{W} \cdot \bar{X} = \bar{V} + W \cdot \bar{X}$$

再根据布尔代数定理组合律 T10，在与项中添加未出现的逻辑变量。

$$F = \bar{V} + W \cdot \bar{X} = \bar{V} \cdot \bar{W} + \bar{V} \cdot W + \bar{V} \cdot W \cdot \bar{X} + V \cdot W \cdot \bar{X}$$

$$F = \bar{V} \cdot \bar{W} \cdot \bar{X} + \bar{V} \cdot \bar{W} \cdot X + \bar{V} \cdot W \cdot \bar{X} + \bar{V} \cdot W \cdot X + \bar{V} \cdot W \cdot \bar{X} + V \cdot W \cdot \bar{X}$$

标准与-或表达式为：

$$F = \bar{V} \cdot \bar{W} \cdot \bar{X} + \bar{V} \cdot \bar{W} \cdot X + \bar{V} \cdot W \cdot \bar{X} + \bar{V} \cdot W \cdot X + V \cdot W \cdot \bar{X}$$

根据布尔代数定理组合律 T10D，在与项中添加未出现的逻辑变量。

$$\begin{aligned} F &= \bar{V} + W \cdot \bar{X} = (\bar{V} + W) \cdot (\bar{V} + \bar{X}) \\ &= (\bar{V} + W + \bar{X}) \cdot (\bar{V} + W + X) \cdot (\bar{V} + \bar{W} + \bar{X}) \cdot (\bar{V} + W + \bar{X}) \end{aligned}$$

标准或-与表达式为：

$$F = (\bar{V} + W + \bar{X}) \cdot (\bar{V} + W + X) \cdot (\bar{V} + \bar{W} + \bar{X})$$

12. 能够实现任何逻辑函数的逻辑门类型的集合称为逻辑门的完全集。例如，2 输入与门、2 输入或门以及反相器构成一个逻辑门完全集。因为任何逻辑函数都能表示为输入信号（以原变量或反变量形式表示）构成的与-或表达式，而且任何超过两个输入端的与门（或门）都能通过2输入端与门（2输入端或门）级联得到。请问2输入与非门能构成逻辑门的完全集吗？请证明你的答案。2输入端异或门呢？

### 【分析解答】

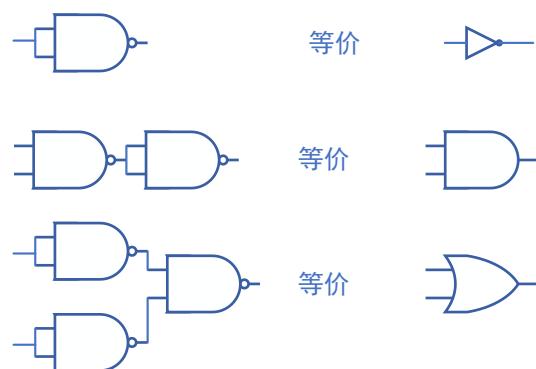
(1) 2输入与非门能构成逻辑门是完全集。这是因为2输入与非门可以通过转换实现与门、或门和非门的功能，从而可以实现任何的逻辑函数。

把与非门的一个输入端接逻辑1或把2个输入端并联，则与非门实现了非门的功能。 $\bar{X} \cdot 1 = \bar{X}$  或  $\bar{X} \cdot \bar{X} = \bar{X}$ 。

把与非门连接到非门，则实现了与门功能。 $\bar{X} \cdot \bar{Y} = X \cdot Y$ 。

把输入信号取反后，再连接到与非门的输入端，则可实现或门功能，如 $\bar{X} \cdot \bar{Y} = X + Y$ 。

可见，与非门通过转换能够实现基本的2输入与门、2输入或门和非门，因而构成逻辑门的完全集。



与非门转换成非门、与门和或门示意图

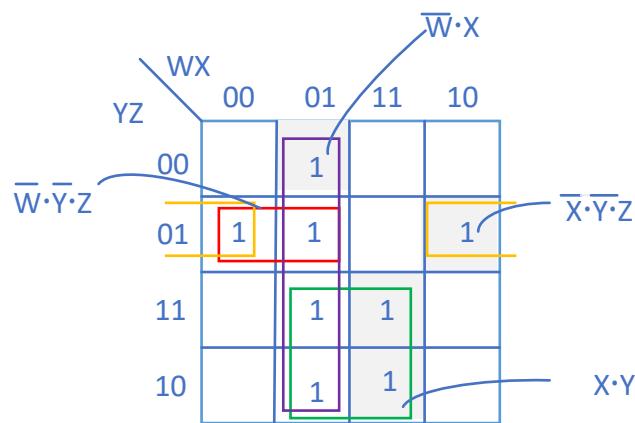
(2) 2 输入端异或门不是逻辑门的完全集。

把异或门的两个输入端并联到一起，输出  $F=0$ ；把其中一个输入端连接到低电平，则  $F=X$ ；把其中一个输入端连接到高电平，则  $F=\bar{X}$ 。可见，异或门可以实现非门的功能，而不能实现或门和与门的功能。

13. 利用卡诺图将下列标准表达式化简得到最简与-或表达式，并把结果转换为与非-与非表达式。

### 【分析解答】

$$(2) F(W,X,Y,Z)=\sum m(1,4,5,6,7,9,14,15)$$



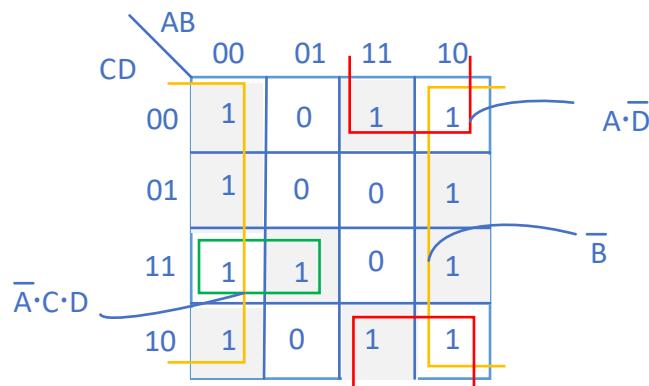
$$\text{最简与-或表达式: } F = \bar{W} \cdot X + X \cdot Y + \bar{X} \cdot \bar{Y} \cdot Z$$

$$\text{与非-与非表达式: } F = \overline{\bar{W} \cdot X + X \cdot Y + \bar{X} \cdot \bar{Y} \cdot Z} = \overline{\overline{\bar{W} \cdot X} \cdot \overline{X \cdot Y} \cdot \overline{\bar{X} \cdot \bar{Y} \cdot Z}}$$

$$(5) F(A,B,C,D)=\prod M(4,5,6,13,15)$$

可根据逻辑函数最大项和最小项列表集合的互补关系，列出该函数的最小项列表：

$$F(A,B,C,D)=\sum m(0,1,2,3,7,8,9,10,11,12,14)$$



$$\text{最简与-或表达式: } F = \bar{B} + A \cdot \bar{D} + \bar{A} \cdot C \cdot D$$

$$\text{与非-与非表达式: } F = \overline{\bar{B} + A \cdot \bar{D} + \bar{A} \cdot C \cdot D} = \overline{\overline{B} \cdot \overline{A \cdot \bar{D}} \cdot \overline{\bar{A} \cdot C \cdot D}}$$

## 第3章 组合逻辑电路

作业：习题3、4、6、7、9、11

3. 写出图3.34所示电路对应的逻辑表达式。

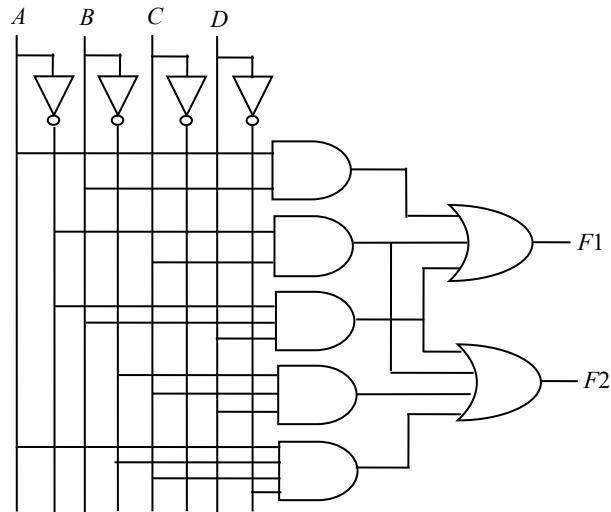


图3.34 习题3所用组合逻辑电路

【分析解答】

$$F1 = A \cdot B + \overline{A} \cdot C + \overline{A} \cdot B \cdot D$$

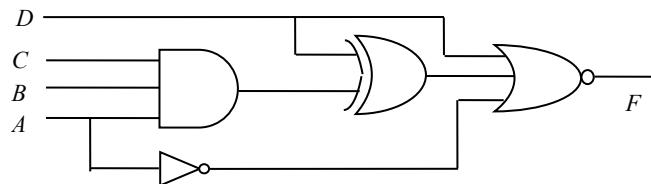
$$F2 = \overline{A} \cdot B \cdot D + \overline{A} \cdot C + \overline{B} \cdot C \cdot D + A \cdot \overline{B} \cdot C \cdot \overline{D}$$

批改情况：正确

4. 假定输出F的逻辑表达式为 $A \cdot B \cdot C \oplus D + \overline{A} + D$ ，画出对应的逻辑电路图，并将该逻辑表达式转换成与-或表达式后，画出对应的两级组合逻辑电路图。

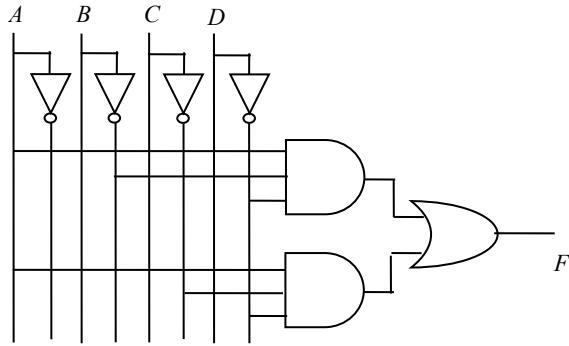
【分析解答】

异或运算的优先级高于或运算，但低于与运算。输出F对应的逻辑电路图如下。



输出F转换为与-或表达式为： $F = A \cdot \overline{B} \cdot \overline{D} + A \cdot \overline{C} \cdot \overline{D}$

与-或表达式对应的逻辑电路图如下：



批改情况：基本正确，少部分同学与-或表达式转换出错

6. 假定一个优先权编码器的输入端为  $I_0, I_1, \dots, I_7$ , 输出端为  $O_0, O_1, O_2$  和 Z, 8 个输入端构成一个 8 位二进制数  $I_0I_1I_2I_3I_4I_5I_6I_7$ , 3 个输出端  $O_0, O_1, O_2$  构成一个 3 位二进制数  $O_0O_1O_2$ 。若输入二进制数  $I_0I_1I_2I_3I_4I_5I_6I_7$  中最左边的 1 所在位为  $I_i$ , 则输出二进制数  $O_0O_1O_2$  的值为  $i$ , Z 为 1; 否则, 若输入二进制数  $I_0I_1I_2I_3I_4I_5I_6I_7$  中最左边的 1 所在位为  $I_i$ , 则输出二进制数  $O_0O_1O_2$  的值为  $i$ , Z 为 0。请用与非门设计该优先权编码器电路, 并说明优先级顺序是什么。

### 【分析解答】

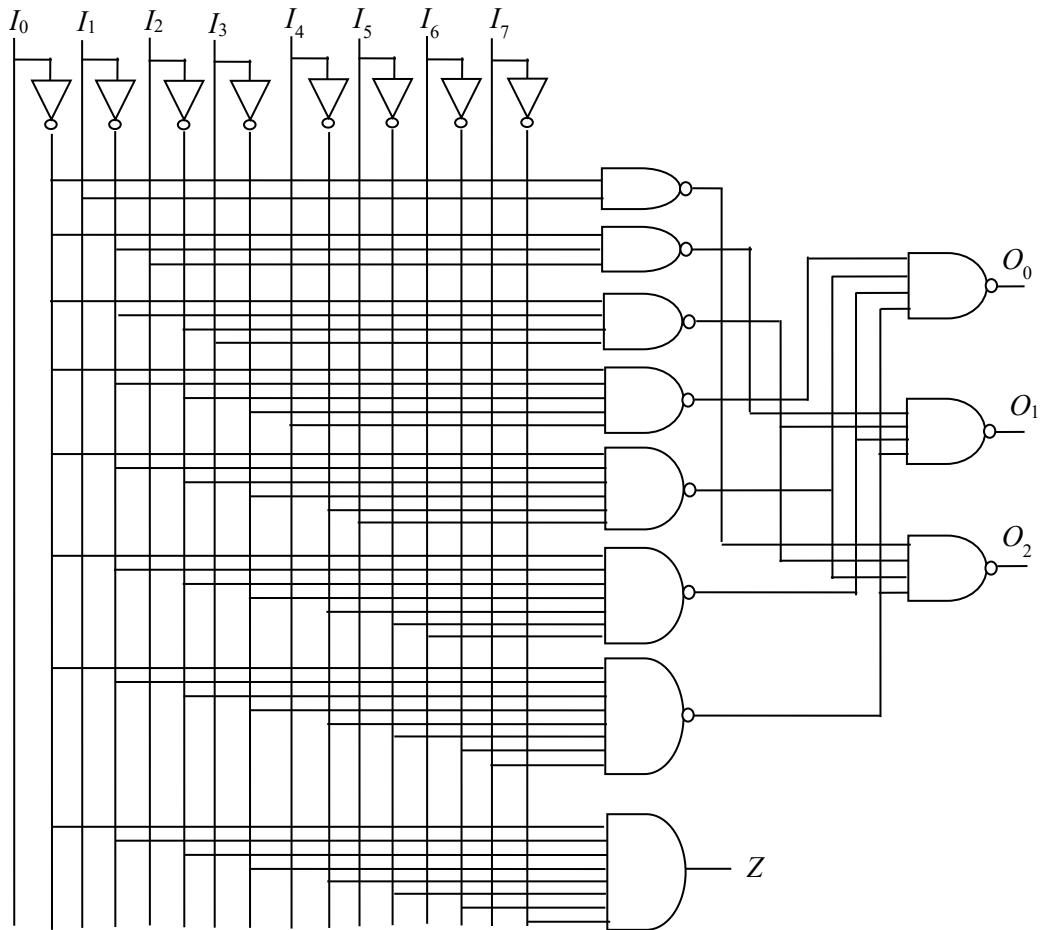
根据题意, 可画出真值表如下:

$I_0$	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$	$O_0$	$O_1$	$O_2$	Z
1	x	x	x	x	x	x	x	0	0	0	0
0	1	x	x	x	x	x	x	0	0	1	0
0	0	1	x	x	x	x	x	0	1	0	0
0	0	0	1	x	x	x	x	0	1	1	0
0	0	0	0	1	x	x	x	1	0	0	0
0	0	0	0	0	1	x	x	1	0	1	0
0	0	0	0	0	0	1	x	1	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	1

根据上述真值表, 可以写出各个输出端的逻辑表达式如下:

$$\begin{aligned}
 O_0 &= \overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot I_4 + \overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot I_5 + \overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot I_6 + \overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot \overline{I_6} \cdot I_7 \\
 &= \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot I_4} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot I_5} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot I_6} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot \overline{I_6} \cdot I_7} \\
 O_1 &= \overline{\overline{I_0} \cdot \overline{I_1} \cdot I_2} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot I_3} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot I_4} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot I_5} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot I_6} \\
 O_2 &= \overline{\overline{I_0} \cdot I_1} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot I_3} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot I_4} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot I_5} \cdot \overline{\overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot I_6} \\
 Z &= \overline{I_0} \cdot \overline{I_1} \cdot \overline{I_2} \cdot \overline{I_3} \cdot \overline{I_4} \cdot \overline{I_5} \cdot \overline{I_6} \cdot \overline{I_7}
 \end{aligned}$$

根据上述表达式, 画出对应的逻辑电路图 (用与非门实现) 如下:



优先权编码器的优先级顺序为:  $I_0 > I_1 > I_2 > I_3 > I_4 > I_5 > I_6 > I_7$

**批改情况:** 逻辑电路图基本绘画正确, 较多同学编码器的优先级没写或优先级颠倒

7. 已知一个组合逻辑电路的功能可用如图 3.35 所示的真值表来描述。分别用下列器件实现该电路。

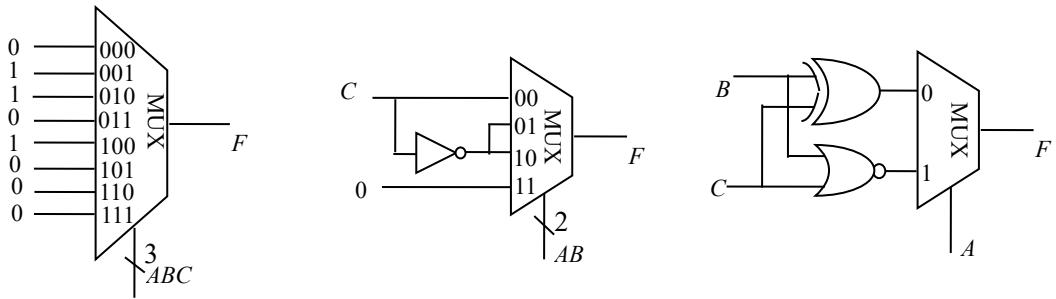
- (1) 一个 8 路选择器。
- (2) 一个 4 路选择器和一个非门。
- (3) 一个 2 路选择器和两个逻辑门。

#### 【分析解答】

用一个 8 路选择器、一个 4 路选择器和一个非门、一个 2 路选择器和两个逻辑门分别实现如下:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

图 3.35 题 7 真值表



**批改情况：**大部分同学电路绘制正确，部分同学三个选择器的输入数据没有准确标明

9. 已知一个组合逻辑电路的功能可用如图 3.36 所示的真值表来描述。要求完成以下任务。

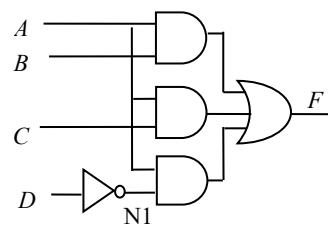
- (1) 利用无关项进行化简，并写出函数 F 的最简逻辑表达式。
- (2) 根据最简逻辑表达式，画出函数 F 对应的逻辑电路图。
- (3) 对于 (2) 中的逻辑电路，请判断是否存在竞争冒险？若存在竞争冒险，则解释在什么情况下会出现毛刺，并画出发生毛刺时的时序图；若不存在竞争冒险，则分析说明其不存在竞争冒险的理由。

A	B	C	D	F
0	0	0	0	d
0	0	0	1	d
0	0	1	0	d
0	0	1	1	0
0	1	0	0	0
0	1	0	1	d
0	1	1	0	0
0	1	1	1	d
1	0	0	0	1
1	0	0	1	0
1	0	1	0	d
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	d
1	1	1	1	1

图 3.36 题 9 真值表

### 【分析解答】

	AB	00	01	11	10
CD	d		1	1	
00	d	d	1		
01		d	1	1	
11			1	1	
10	d		d	d	



$$F = A \cdot B + A \cdot C + A \cdot \overline{D}$$

上面的逻辑电路不存在竞争冒险，因为得到的最简逻辑表达式是积之和表达式，各乘积项中不存在逻辑相反的变量（这个是必要但非充分条件）。

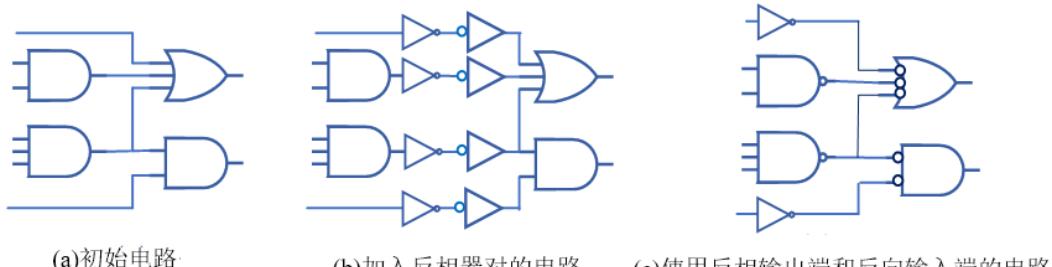
**批改情况：**部分同学卡诺图绘画正确，但根据无关项无法得到最简表达式以至于逻辑电路不是最简，从而对于竞争冒险是否存在判断错误。

11. 根据图 3.37 中给出的逻辑门的传输延迟  $T_{pd}$  和最小延迟  $T_{cd}$ ，

计算下图（2.4.3 节中图 2.30a、2.30b 和 2.30c 中）所示组合逻辑电路的传输延迟和最小延迟，并比较哪个电路的传输延迟最长，哪个电路的传输延迟最短。

逻辑门	$T_{pd}$ (ps)	$T_{cd}$
NOT	15	10
2 输入 OR	40	30
3 输入 OR	55	45
2 输入 AND	30	25
3 输入 AND	40	30
2 输入 NOR	30	25
3 输入 NOR	45	35
2 输入 NAND	20	15
3 输入 NAND	30	25
2 输入 XOR	60	40

图 3.37 门电路延迟



(a)初始电路

(b)加入反相器对的电路

(c)使用反相输出端和反向输入端的电路

### 【分析解答】

电路(a)的传输延迟为  $40+55=95$  ps；最小延迟为 25 ps。

电路(b)的传输延迟为  $40+15+15+55=125$  ps；最小延迟为  $10+10+25=45$  ps。

电路(c)中，反向输入端与门是或非门的等效电路，反向输入端或门是与非门的等效电路，因此，传输延迟为  $30+30=60$  ps；最小延迟为  $10+25=35$  ps。

显然，上述电路中，电路(b)的传输延迟最长，(c)的传输延迟最短。

**批改情况：**电路(c)的错的比较多，部分同学不清楚反向输入端与门是或非门的等效电路，反向输入端或门是与非门的等效电路以至于电路(c)中的传输延迟与最小延迟计算出错

## 4 章 时序逻辑电路

作业：习题 4、5、6、9、11、12

4. 假设 SR 锁存器的输入端  $S$ 、 $R$  的波形如图 4.27 所示，图中信号的上升延迟和下降延迟设为 0，要求画出图 4.27 中输出端  $Q$  和  $\bar{Q}$  的输出波形。

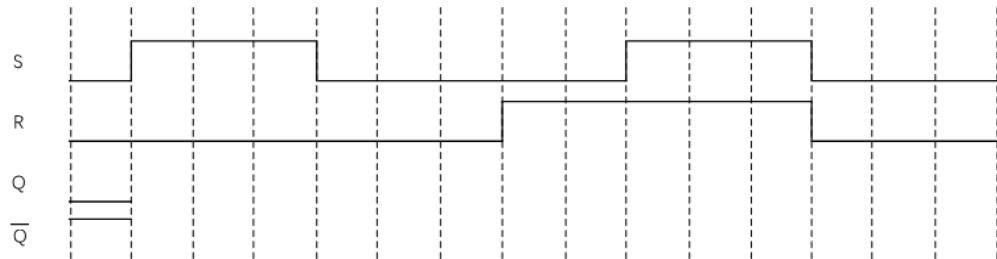
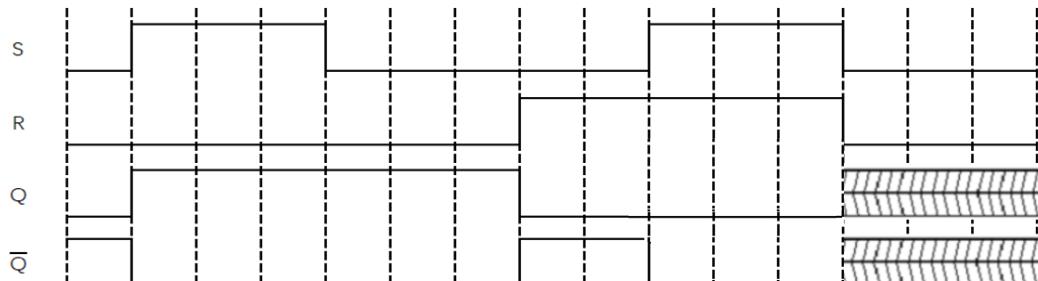


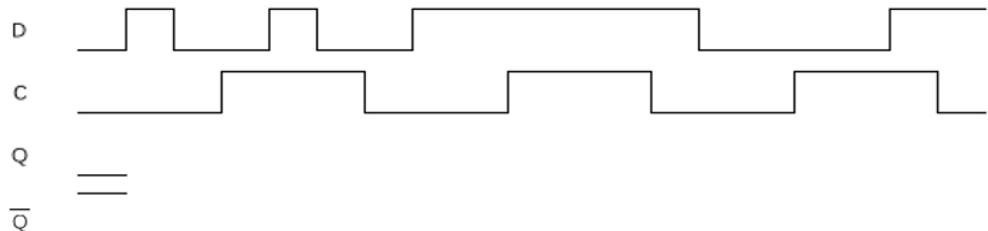
图 4.27 SR 锁存器的波形图

### 【分析解答】

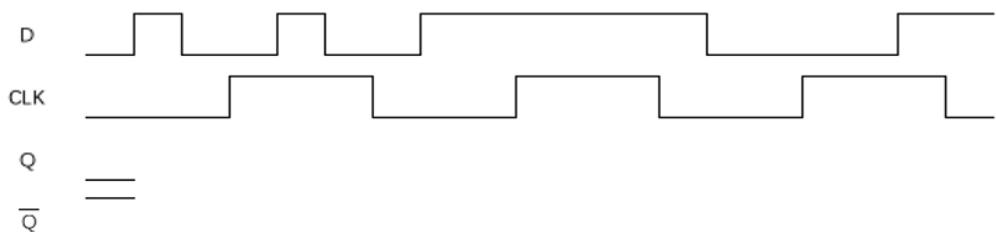
根据  $S$  和  $R$  端的输入情况，输出端  $Q$  和  $\bar{Q}$  的输出波形如下图所示（不考虑门延迟）：



5. 假设 D 锁存器和 D 触发器的各输入端波形分别如图 4.28a 和 b 所示，图中信号的上升延迟和下降延迟设为 0，并且不考虑逻辑门的传输延迟，要求画出图 4.28a 和 b 中输出端  $Q$  和  $\bar{Q}$  的输出波形。



(a) D锁存器

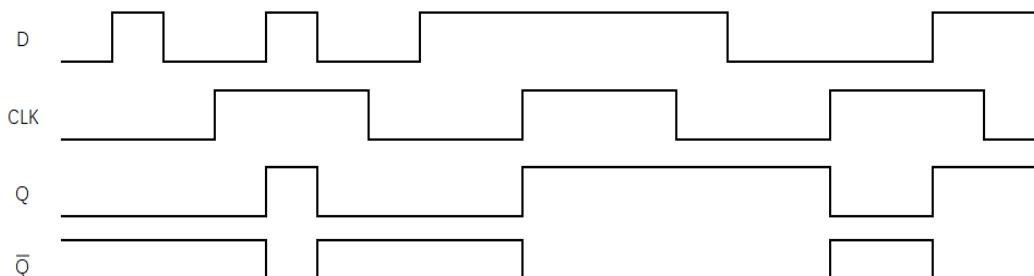


(b) D触发器

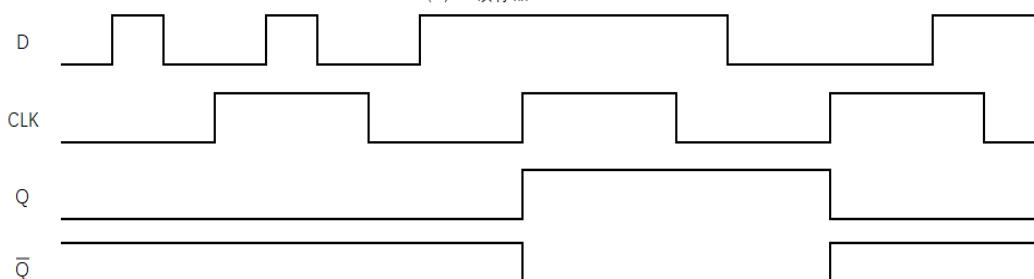
图 4.28 D 锁存器和 D 触发器的波形图

### 【分析解答】

假设 D 锁存器的控制端 C 为高电平有效，D 触发器是上升沿触发，则它们的输出波形图如下：



(a) D锁存器



(b) D触发器

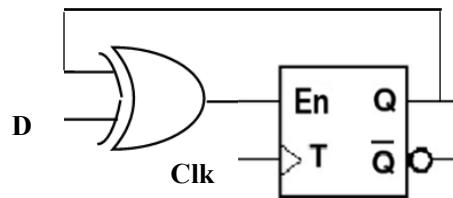
6. 请用带使能端的 T 触发器和组合逻辑构造 D 触发器。

### 【分析解答】

D 触发器的次态方程  $Q^* = D$ ; 使能 T 触发器的次态方程为： $Q^* = \overline{EN} \cdot Q + EN \cdot \overline{Q}$ 。

根据这两个次态方程得到以下表中的结果：

Q	D	Q*	EN
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0



由此可以推得:  $EN = \overline{D} \cdot Q + D \cdot \overline{Q}$ 。(也可以由  $z=x \oplus y \rightarrow x=z \oplus y \rightarrow y=z \oplus x$  推出)

9. 请用尽量少的 D 触发器实现一个能检测输入信号 X 中是否出现“110”序列的电路。

若出现“110”序列，则输出 Z 为 1，否则 Z 为 0。请分析你实现的电路是否能够自启动。如果 D 触发器的个数没有限制，你是否有更简洁的实现方案？

### 【分析解答】

根据题意设计如下状态表:

现态 Y	次态 Y*/输出 Z	
	X=0	X=1
S0 (初态)	S0 / Z=0	S1 / Z=0
S1 (检测到第一位 1)	S0 / Z=0	S2 / Z=0
S2 (检测到两位 11)	S0 / Z=1	S2 / Z=0

根据次优状态分配策略，三个状态都有编码相邻的需求，设置 S0 为 Y0Y1=00，S1 为 01，S2 为 10 编码。得到如下状态转换表:

状态转移表			
Y0	Y1	X	Y0*Y1*Z
0	0	0	0 0 0
0	0	1	0 1 0
0	1	0	0 0 0
0	1	1	1 0 0
1	0	0	0 0 1
1	0	1	1 0 0
1	1	0	d d d
1	1	1	d d d

利用无关项进行化简，可得如下次态函数：

$$Y_0^* = X \cdot Y_1 + X \cdot Y_0 = X \cdot (Y_1 + Y_0)$$

$$Y_1^* = X \cdot \overline{Y_0} \cdot \overline{Y_1}$$

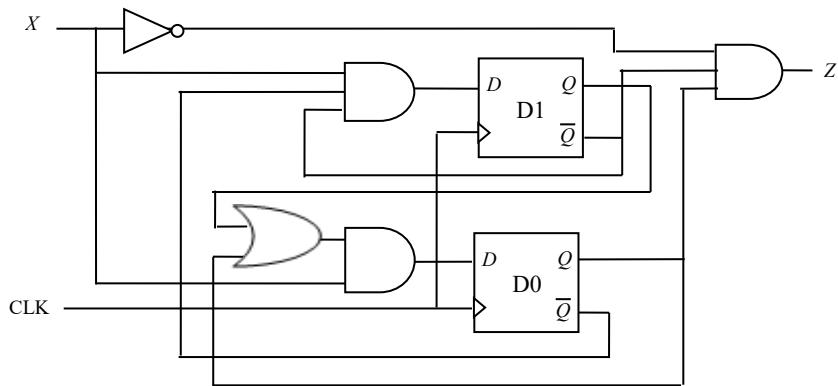
$$Z = \overline{X} \cdot Y_0$$

考察状态变化情况：当电路处于 11 状态时，若  $X=0$ ，则经过一个时钟周期，电路状态回到初态 00。若  $X=1$ ，则经过一个时钟周期，电路变成 10 (S2) 状态，此时若再输入  $X=1$ ，则出现两个连续的 1，变成 S2 状态，状态变化正确；若再输入  $X=0$ ，则经过一个时钟周期，回到初态 00。

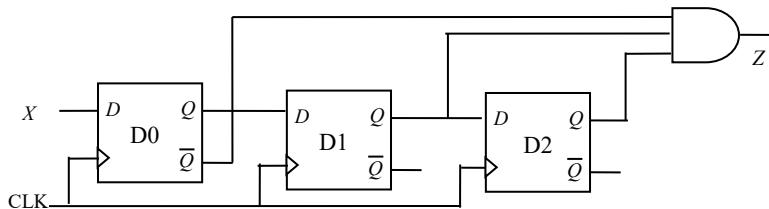
考察输出 Z 的情况：当电路处于 11 状态时，若  $X=0$ ，则输出  $Z=1$ ，输出错误。因此，输出逻辑应调整为：

$$Z = \overline{X} \cdot Y_0 \cdot \overline{Y_1}$$

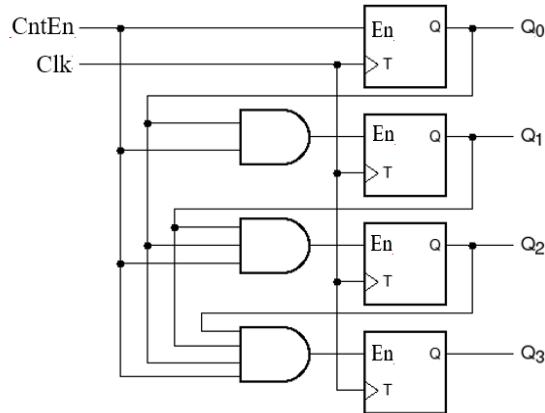
得到电路图如下所示：



如果触发器数目没有限制，可以通过增加一个触发器来简化组合逻辑设计。其实现如下所示：



11. 假设图 4.20 所示的 4 位同步并行加法计数器中 T 触发器的信号传输延迟是  $T_{tq}$ , 与



门的传输延迟为  $T_{and}$ , T 触发器 En 信号的建立时间是  $T_{setup}$ , 请计算该计数器外部时钟 Clk 的最大工作频率。

### 【分析解答】

由时序逻辑电路的时序分析可知:

时钟周期  $t_{clk} >$  触发器锁存延迟  $t_{ffpd} +$  次态激励延迟  $t_{nsd} +$  触发器建立时间  $t_{setup}$

根据题意可知:  $t_{ffpd} = T_{tq}$ ,  $t_{nsd} = T_{and}$ ,  $t_{setup} = T_{setup}$ 。

因此可知, 该计数器最大工作频率为:

$$1 / (T_{tq} + T_{and} + T_{setup})$$

12. 将图 4.25 所示的右移一位寄存器中  $Q_3$  和  $Q_2$  异或后送入输入端  $X$ , 可构成一个线性反馈移位寄存器计数器。请分析该设计中  $Q_3Q_2Q_1Q_0$  构成的状态编码转移情况, 并分析总结其特点。

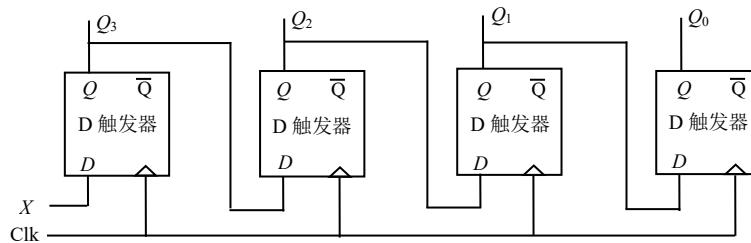


图 4.25 右移一位寄存器

### 【分析解答】

$Q_3Q_2Q_1Q_0$  编码状态转移的情况如下:

$$0000 \rightarrow 0000;$$

$0001 \rightarrow 0000$ ;

$0010 \rightarrow 0001 \rightarrow 0000$ ;

$0011 \rightarrow 0001 \rightarrow 0000$ ;

$0100 \rightarrow 1010 \rightarrow 1101 \rightarrow 0110 \rightarrow 1011 \rightarrow 1101$

$0101 \rightarrow 1010$

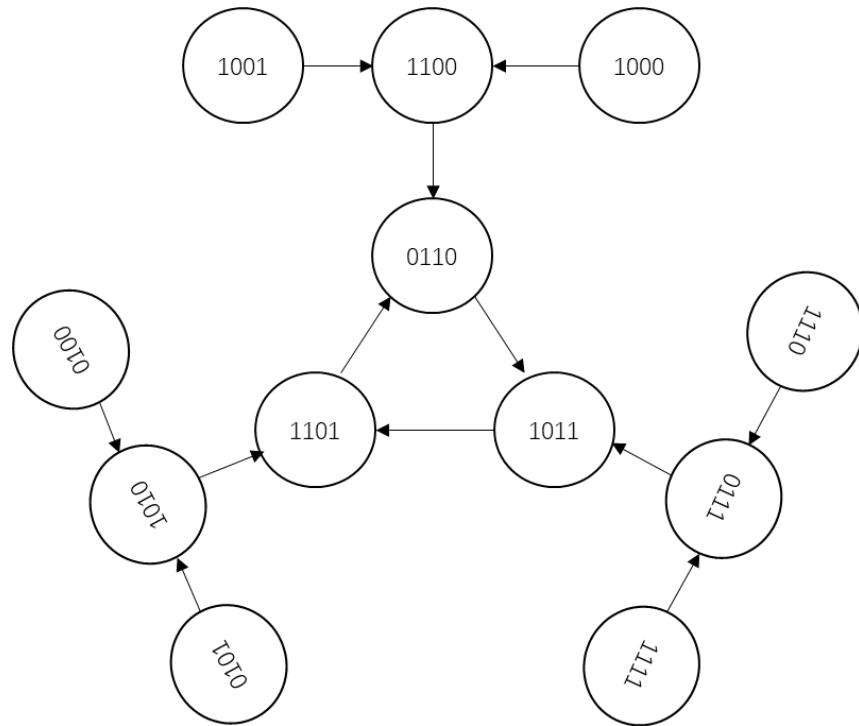
$1000 \rightarrow 1100 \rightarrow 0110$

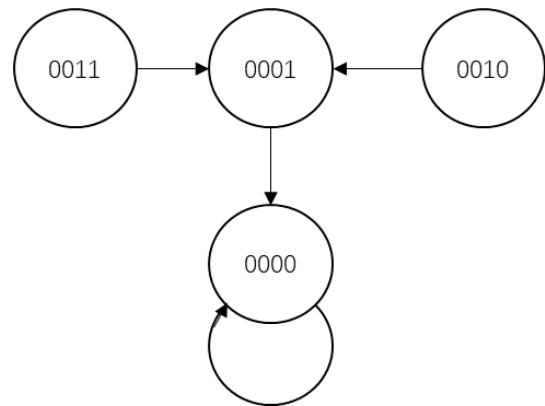
$1111 \rightarrow 0111 \rightarrow 1011$

$1001 \rightarrow 1100$

$1110 \rightarrow 0111$

用编码对应的数字表示状态符号，其状态图如下所示：





图中有两个工作循环，一个是三状态，一个单状态。其它状态都会在有限个时钟周期后，进入工作循环中。

# 第 6 章 运算方法和运算部件

作业：习题 3、4、5、6、7

补充题目（复习第一章的内容）：

考虑下列 C 语言程序代码：

```
int i = 65535;  
short si = (short)i;  
int j = si;
```

假定上述程序段在某 32 位机器上执行，`sizeof (int)=4`，则变量 i、si 和 j 的值分别是多少？为什么？

## 【分析解答】

在一台 32 位机器上执行上述代码段时，i 为 32 位补码表示的定点整数，第 2 行要求强行将一个 32 位带符号数截断为 16 位带符号整数，65535 的 32 位补码表示为 0000 FFFFH，截断为 16 位后变成 FFFFH，它是 -1 的 16 位补码表示，因此 si 的值是 -1。再将该 16 位带符号整数扩展为 32 位时，就变成了 FFFF FFFFH，它是 -1 的 32 位补码表示，因此 j 的值也为 -1。也就是说，i 的值原来为 65535，经过截断、再扩展后，其值变成了 -1。

**批改情况：基本正确，需要注意的是 0000 FFFFH 的值为 65535 的补码而不是 65536，有个别同学出错**

3. 考虑以下 C 语言程序代码：

```
int func1 (unsigned word)  
{  
    return (int) ((word << 24) >> 24);  
}  
int func2 (unsigned word)  
{  
    return ((int) word << 24) >> 24;  
}
```

假设在一个 32 位机器上执行这些函数，`sizeof (int)=4`。说明函数 func1 和 func2 的功能，并填写表 6.2，给出对表中“异常”数据的说明。

表 6.2 题 3 用表

W		func1(w)		func2(w)	
机器数	值	机器数	值	机器数	值
	127				
	128				
	255				

	256				
--	-----	--	--	--	--

### 【分析解答】

函数 func1 的功能是把无符号数高 24 位清零（左移 24 位再逻辑右移 24 位），结果一定是正的带符号整数；而函数 func2 的功能是把无符号数的高 24 位都变成和第 25 位一样，因为左移 24 位后左边第一位变为原来的第 25 位，然后进行算术右移，高位补符号，即高 24 位都变成和原来第 25 位相同。

根据程序执行的结果填表如下表，表中机器数用十六进制表示。

题 3 中填入结果后的表

W		func1(w)		func2(w)	
机器数	值	机器数	值	机器数	值
0000007FH	127	0000007FH	+127	0000007FH	+127
00000080H	128	00000080H	+128	<b>FFFFFF80H</b>	<b>-128</b>
000000FFH	255	000000FFH	+255	<b>FFFFFFFH</b>	<b>-1</b>
00000100H	256	<b>00000000H</b>	<b>0</b>	<b>00000000H</b>	<b>0</b>

因为逻辑左移和算术左移的结果完全相同，所以，函数 func1 和 func2 中第一步左移 24 位得到的结果完全相同，所不同的是右移 24 位后的结果不同。

上述表中，加粗数据是一些“异常”结果。当 w=128 和 255 时，第 25 位正好是 1，因此函数 func2 执行的结果为一个负数，出现了“异常”。当 w=256 时，低 8 位为 00，高 24 位为非 0 值，左移 24 位后使得有效数字被移出，因而发生了“溢出”，使得出现了“异常”结果 0。

**批改情况：基本正确，部分同学审题不仔细没有回答函数 func1 和 func2 的功能**

4. 填写表 6.3，注意对比无符号数和带符号整数的乘法结果，以及截断操作前、后的结果。

表 6.3 题 4 用表

模式	x		y		x×y（截断前）		x×y（截断后）	
	机器数	值	机器数	值	机器数	值	机器数	值
无符号数	110		010					
二进制补码	110		010					
无符号数	001		111					
二进制补码	001		111					
无符号数	111		111					
二进制补码	111		111					

### 【分析解答】

根据无符号数乘法运算和补码乘法运算算法，填写表 6.3 后得到下表。

题 4 中填入结果后的表

模式	x		y		x×y (截断前)		x×y (截断后)	
	机器数	值	机器数	值	机器数	值	机器数	值
无符号数	110	6	010	2	001100	12	100	4
二进制补码	110	-2	010	+2	111100	-4	100	-4
无符号数	001	1	111	7	000111	7	111	7
二进制补码	001	+1	111	-1	111111	-1	111	-1
无符号数	111	7	111	7	110001	49	001	1
二进制补码	111	-1	111	-1	000001	+1	001	+1

对上表中结果分析如下：

① 对于两个相同的机器数，作为无符号数进行乘法运算和作为带符号整数进行乘法运算，因为其所用的乘法算法不同，所以，乘积的机器数可能不同。但是，从表中看出，截断后的乘积是一样的，也即不同的仅是乘积中的高 n 位，而低 n 位完全一样。

② 对于 n 位乘法运算，无论是无符号数乘法还是带符号整数乘法，若截取 2n 位乘积的低 n 位作为最终的乘积，则都有可能结果溢出，即 n 位数字无法表示正确的乘积。虽然表中给出的带符号整数乘积截断后都没有发生溢出，但实际上还是存在溢出的情况，例如， $011 \times 011 = 001001$ ，截断后  $011 \times 011 = 001$ ，显然截断后的结果发生了溢出。

③ 表中加粗的地方是截断后发生溢出的情况。可以看出，对于无符号整数乘法，若乘积中高 n 位为全 0，则截断后的低 n 位乘积不发生溢出，否则溢出；对于带符号整数乘法，若高 n 位中的每一位都等于低 n 位中的第一位，则截断后的低 n 位乘积不发生溢出，否则溢出。

**批改情况：正确**

5. 以下是两段 C 语言代码，函数 arith() 是直接用 C 语言写的，而 optarith() 是对 arith() 函数以某个确定的 M 和 N 编译生成的机器代码反编译生成的。根据 optarith()，可以推断函数 arith() 中 M 和 N 的值各是多少？

```
#define M
#define N
int arith(int x, int y)
{
    int result = 0;
    result = x*M + y/N;
    return result;
}
```

```
int optarith ( int x, int y)
{
    int t = x;
```

```

x <<= 4;
x_ = t;
if (y < 0) y+= 3;
y>>=2;
return x+y;
}

```

### 【分析解答】

对反编译结果进行分析，可知：对于 x，指令机器代码中有一条“x 左移 4 位”指令，即： $x=16x$ ，然后有一条“减法”指令，即  $x=16x-x=15x$ ，根据源程序知 M=15；对于 y，有一条“y 右移 2 位”指令，即  $y=y/4$ ，根据源程序知 N=4。但是，当  $y<0$  时，对于有些 y，执行  $y>>2$  后的值并不等于  $y/4$ 。例如，当  $y=-1$  时，在反编译函数 optarith 中执行  $y>>2$  时，因为 -1 的机器数为全 1，左移两位后还是全 1，也即  $-1>>2=-1$ ，结果为 -1；而原函数 arith 中执行  $y/4$  时，因为  $-1/4=0$ ，得到结果为 0。

对于带符号整数来说，采用算术右移时，高位补符号，低位移出。因此，当符号位为 0 时，与无符号整数相同，采用移位方式和直接相除得到的商完全一样。当符号位为 1 时，若低位移出的是非全 0，则说明不能整除。例如，对于  $-3/2$ ，假定补码位数为 4，则进行算术右移操作  $1101>>1=1110.1B$ （小数点后面部分移出）后得到的商为 -2，而精确商是 -1.5，即整数商应为 -1。显然，算术右移后得到的商比精确商少了 0.5，相当于朝  $-\infty$  方向进行了舍入，而不是朝零方向舍入。因此，这种情况下，移位得到的商与直接相除得到的商不一样，需要进行校正。

校正的方法是，对于带符号整数  $x$ ，若  $x<0$ ，则在右移前，先将  $x$  加上偏移量  $(2^k-1)$ ，然后再右移  $k$  位。例如，上述函数 optarith 中，在执行  $y>>2$  之前加了一条语句 “`if (y < 0) y+= 3;`”，以对 y 进行校正。

**批改情况：正确**

6. 设  $A_4 \sim A_1$  和  $B_4 \sim B_1$  分别是 4 位加法器的两组输入， $C_0$  为低位来的进位。当加法器分别采用串行进位和先行进位时，写出 4 个进位  $C_4 \sim C_1$  的逻辑表达式。

### 【分析解答】

$$\begin{aligned}
\text{串行进位: } C_1 &= A_1 C_0 + B_1 C_0 + A_1 B_1 \\
C_2 &= A_2 C_1 + B_2 C_1 + A_2 B_2 \\
C_3 &= A_3 C_2 + B_3 C_2 + A_3 B_3 \\
C_4 &= A_4 C_3 + B_4 C_3 + A_4 B_4
\end{aligned}$$

$$\begin{aligned}
\text{并行进位: } C_1 &= A_1 B_1 + (A_1 + B_1) C_0 \\
C_2 &= A_2 B_2 + (A_2 + B_2) A_1 B_1 + (A_2 + B_2) (A_1 + B_1) C_0 \\
C_3 &= A_3 B_3 + (A_3 + B_3) A_2 B_2 + (A_3 + B_3) (A_2 + B_2) A_1 B_1 + (A_3 + B_3) (A_2 + B_2) (A_1 + B_1) C_0 \\
C_4 &= A_4 B_4 + (A_4 + B_4) A_3 B_3 + (A_4 + B_4) (A_3 + B_3) A_2 B_2 + (A_4 + B_4) (A_3 + B_3) (A_2 + B_2) A_1 B_1
\end{aligned}$$

$$+(A_4+B_4)(A_3+B_3)(A_2+B_2)(A_1+B_1)C_0$$

**批改情况：**正确，部分同学漏写先行进位时的表达式

7. 请按如下要求计算，并把结果还原成真值。

- (1) 设  $[x]_{\text{补}}=0101$ 、 $[y]_{\text{补}}=1101$ ，求  $[x+y]_{\text{补}}$ ， $[x-y]_{\text{补}}$ 。
- (2) 设  $[x]_{\text{原}}=0101$ 、 $[y]_{\text{原}}=1101$ ，用原码一位乘法计算  $[x \times y]_{\text{原}}$ 。
- (3) 设  $[x]_{\text{补}}=0101$ 、 $[y]_{\text{补}}=1101$ ，用 MBA (基 4 布斯) 乘法计算  $[x \times y]_{\text{补}}$ 。
- (4) 设  $[x]_{\text{原}}=0101$ 、 $[y]_{\text{原}}=1101$ ，用不恢复余数法计算  $[x/y]_{\text{原}}$  的商和余数。
- (5) 设  $[x]_{\text{补}}=0101$ 、 $[y]_{\text{补}}=1101$ ，用不恢复余数法计算  $[x/y]_{\text{补}}$  的商和余数。

### 【分析解答】

(1)  $[x]_{\text{补}}=0101B$ ,  $[y]_{\text{补}}=1101B$ ,  $[-y]_{\text{补}}=0011B$ 。

$[x+y]_{\text{补}} = [x]_{\text{补}} + [y]_{\text{补}} = 0101B + 1101B = (1)0010B$ , 因此,  $x+y=2$ 。

两个不同符号数相加，结果一定不会溢出。验证:  $x=+101B=5$ ,  $y=-011B=-3$ ,  $x+y=2$ 。

$[x-y]_{\text{补}} = [x]_{\text{补}} + [-y]_{\text{补}} = 0101B + 0011B = (0)1000B$ , 因此,  $x-y=-8$ 。

两个正数相加结果为负，发生例了溢出。验证:  $5-(-3)=8>$ 最大可表示数 7，故溢出。

**批改情况：**需要注意是有符号数进行运算，得到的(0)1000B 是补码，转换结果  $x-y$  为负，因此两个正数相加结果为负，发生溢出

(2)  $[x]_{\text{原}} = 0101B$ ,  $[y]_{\text{原}}=1101B$ 。将符号和数值部分分开处理。

乘积的符号为  $0 \oplus 1=1$ ，数值部分采用无符号数乘法算法计算  $101 \times 101$  的乘积。

原码一位乘法过程描述如下：初始部分积为 0，在乘积寄存器前增加一个进位位。每次循环首先根据乘数寄存器中最低位决定  $+X$  还是  $+0$ ，然后将得到的新进位、新部分积和乘数寄存器中的部分乘数一起逻辑右移一位。共循环 3 次，最终得到一个 8 位无符号数表示的乘积  $10011001B$ 。

C	P	Y	说明
0	0 0 0	1 0 1	$P_0=0$
	+ 1 0 1		
0	1 0 1		$y_0=1$ , $+X$
0	0 1 0	1 1 0	$C, P$ 和 $Y$ 同时右移一位 得 $P_1$
	+ 0 0 0		
0	0 1 0		$y_1=0$ , $+0$
0	0 0 1	0 1 1	$C, P$ 和 $Y$ 同时右移一位 得 $P_2$
	+ 1 0 1		
0	1 1 0	0 1 1	$y_2=1$ , $+X$
0	0 1 1	0 0 1	$C, P$ 和 $Y$ 同时右移一位 得 $P_3$

符号位为 1，因此， $[x \times y]_{\text{原}} = 10011001$ ，因此， $x \times y = -25$ 。

若结果取 4 位原码 1 001，则因为乘积数值部分高 3 位为 0011，是一个非 0 数，所以，结果溢出。验证：4 位原码的表示范围为 -7~+7，显然乘积 -25 不在其范围内，结果应该溢出。

**批改情况：**注意这是有符号数的运算用原码一位乘法需要把符号位和数值部分分开运算，错的同学基本都是因为没有把符号位和数值部分分开运算

$$(3) [x]_{\text{补}} = 0101B, [-x]_{\text{补}} = 1011B, [y]_{\text{补}} = 1101B.$$

采用 MBA 算法时，符号和数值部分一起参加运算，在乘数后面添 0，初始部分积为 0，并在部分积前加一位符号位 0。每次循环先根据乘积寄存器中最低 3 位决定执行  $+X$ 、 $+2X$ 、 $-X$ 、 $-2X$ 、还是  $+0$  操作，然后将得到的新的部分积和乘数寄存器中的部分乘数一起算术右移两位。 $-X$  和  $-2X$  分别采用  $+[-x]_{\text{补}}$  和  $+2[-x]_{\text{补}}$  的方式进行。共循环两次。最终得到一个 8 位补码表示的乘积 1111 0001 B。

C	P	Y	$Y_{-1}$	说明
0	0 0 0 0	1 1 0 1	0	$P_0 = 0$
$+0$	0 1 0 1			$y_1 y_0 y_{-1} = 010, +X$
	0 1 0 1			$C, P$ 和 $Y$ 同时右移两位
	0 0 0 1	0 1	1 1	得 $P_1$
$+1$	1 0 1 1			$y_3 y_2 y_1 = 110, -X$
	1 1 0 0			$C, P$ 和 $Y$ 同时右移两位
	1 1 1 1	0 0 0 1		得 $P_2$

$$[x \times y]_{\text{补}} = 1111 0001, \text{ 因此, } x \times y = -15.$$

**批改情况：**基本正确

$$(4) [x]_{\text{原}} = 0101B, [y]_{\text{原}} = 1101B. \text{ 将符号和数值部分分开处理。}$$

将符号和数值部分分开处理。商的符号为  $0 \oplus 1 = 1$ ，数值部分采用无符号数除法算法计算 101B 和 101B 的商和余数。无符号数不恢复余数除法过程描述如下：初始中间余数为 0 000 101 0，其中，最高位为添加的符号位，用于判断余数是否大于等于 0；最后一位 0 为第一次上的商，该位商只是用于判断结果是否溢出，不包含在最终的商中。因为结果肯定不溢出，所以该位商可以直接上 0，并先做一次  $-Y$  操作得到第一次中间余数，然后进入循环。每次循环首先将中间余数和商一起左移一位，然后根据上一次上的商（或余数的符号）决定执行  $+Y$  还是  $-Y$  操作，以得到新的中间余数，最后根据中间余数的符号确定上商为 0 还是 1。 $-Y$  采用  $+[-y]_{\text{补}}$  的方式进行。整个循环内执行的要点是“正、1、减；负、0、加”。共循环 3 次。最终得到一个 3 位无符号数表示的商 0001 和余数 0000，其中第一位商 0 必须去掉，添上符号位后得到最终的商的原码表示为 1001，余数的原码表示为 0000。因此， $x/y$  的商为 -1，余数为 0。

余数寄存器 R	余数/商寄存器 Q	说 明
0 0 0 0	1 0 1	开始 $R_0 = X$
+ 1 0 1 1		$R_1 = X - Y$
1 0 1 1	1 0 1 0	$R_1 < 0$ , 故 $q_3 = 0$ , 没有溢出
0 1 1 1	0 1 0	$2R_1$ ( $R$ 和 $Q$ 同时左移, 空出一位商)
+ 0 1 0 1		$R_2 = 2R_1 + Y$
1 1 0 0	0 1 0 0	$R_2 < 0$ , 则 $q_2 = 0$
1 0 0 0	1 0 0	$2R_2$ ( $R$ 和 $Q$ 同时左移, 空出一位商)
+ 0 1 0 1		$R_3 = 2R_2 + Y$
1 1 0 1	1 0 0 0	$R_3 < 0$ , 则 $q_1 = 0$
1 0 1 1	0 0 0	$2R_3$ ( $R$ 和 $Q$ 同时左移, 空出一位商)
+ 0 1 0 1		$R_4 = 2R_3 - Y$
0 0 0 0	0 0 0 1	$R_4 > 0$ , 则 $q_0 = 1$

商的最高位为 0, 说明没有溢出, 商的数值部分为 001。所以,  $[x/y]_{\text{原}} = 1\ 001$  (最高位为符号位), 余数为 0。

**批改情况: 基本正确, 注意最高位为符号位**

(5)  $[x]_{\text{补}} = 0101B$ ,  $[y]_{\text{补}} = 1101B$ 。

补码不恢复余数除法过程描述如下: 初始中间余数为 0000 0101, 整个循环内执行的要点是“同、1、减; 异、0、加”。共循环 4 次, 得到商 1110 和余数 0010。最终根据情况需要对商和余数进行修正。

余数寄存器 R	余数/商寄存器 Q	说 明
0 0 0 0	0 1 0 1	开始 $R_0 = X$
+ 1 1 0 1		被除数和除数异号, 做加法
1 1 0 1	0 1 0 1	同、1、减
1 0 1 0	1 0 1 1	$2R_1$ ( $R$ 和 $Q$ 同时左移, 空出一位商)
+ 0 0 1 1		$R_2 = 2R_1 - Y$
1 1 0 1	1 0 1 1	同、1、减
1 0 1 1	0 1 1 1	$2R_2$ ( $R$ 和 $Q$ 同时左移, 空出一位商)
+ 0 0 1 1		$R_3 = 2R_2 - Y$
1 1 1 0	0 1 1 1	同、1、减
1 1 0 0	1 1 1 1	$2R_3$ ( $R$ 和 $Q$ 同时左移, 空出一位商)
+ 0 0 1 1		$R_4 = 2R_3 - Y$
1 1 1 1	1 1 1 1	同、1、减
1 1 1 1	1 1 1 1	$2R_4$ ( $R$ 和 $Q$ 同时左移, 空出一位商)

$$\begin{array}{r}
 \underline{0\ 0\ 1\ 1} \\
 0\ 0\ 1\ 0 \quad 1\ 1\ 1\ 0
 \end{array}
 \quad R_5 = 2R_3 - Y$$

异、0、加（最高位商1去掉）

商的修正：最后一次 Q 寄存器左移一位，将最高位  $q_n$  移出，最低位置商  $q_0=0$ 。  
 若被除数与除数同号，Q 中就是真正的商；否则，将 Q 中商的末位加 1。故商为  
 $1110+1=1111B$ 。

余数的修正：若余数符号同被除数符号，则不需修正；否则，按下列规则进行修正：  
 当被除数和除数符号相同时，最后余数加除数；否则，最后余数减除数。故余数为 0010B。

商的为 1111 (-1)，余数为 0010 (2)。验证：“除数  $\times$  商 + 余数 = 被除数”进行验证，得  $(-3) \times (-1) + 2 = 5$ 。

**批改情况：基本正确。**

## 作业：习题 2(4)、2(9)、3、6、7、8、10、11、13、15、17

2. (4) 哪些寻址方式下的操作数在寄存器中？哪些寻址方式下的操作数在存储器中？

### 【分析解答】

寄存器直接寻址的操作数在寄存器中。

寄存器间接、直接、间接、基址、变址、相对这几种寻址方式的操作数都在存储器中。

**批改情况：正确**

2. (9) 转移跳转和调用指令的区别是什么？返回指令是否需要有地址码字段？

### 【分析解答】

转移（跳转）指令执行后，CPU 将跳转到目标指令地址中执行，而调用指令执行后，其返回地址（即调用指令的下条指令的地址）会保存到栈中或特定的寄存器中，然后跳转到被调用过程第一条指令（目标指令）处执行，因此，被调用过程执行结束时会执行一条返回指令，返回指令将取出返回地址并置入 PC，从而使 CPU 返回到调用指令处执行。如果返回地址存放在栈中或特定的寄存器中，则返回指令中可以不需要地址码；如果返回地址存放在某个通用寄存器中，则返回指令中需要给出通用寄存器编号（地址码）。

**批改情况：两种指令区别正确，对于返回指令是否需要地址码字段大部分同学没有考虑  
返回地址分别在特定寄存器、通用寄存器是不同的情况**

3. 假定某计算机中有一条转移指令，采用相对寻址方式，共占两个字节，第一字节是操作码，第二字节是相对位移量（用补码表示），CPU 每次从内存只能取一个字节。

假设执行到某转移指令时 PC 的内容为 258，执行该转移指令后要求转移到 220 开始的一段程序执行，则该转移指令第二字节的内容应该是多少？

### 【分析解答】

因为该计算机的 CPU 每次从内存只能取一个字节，因而它采用字节编址方式。执行到该转移指令时 PC 的内容为 258，因此取出第一字节的操作码后，PC 的内容为 259，取出第二字节的位移量后，PC 的内容为 260。在执行该转移指令计算转移目标地址时，PC 应该已经是 260 了。假定位移量为 x，则根据转移目标地址  $(220) = PC(260) + \text{相对位移量 } (x)$ ，可知  $x = 220 - 260 = -40$ ，用补码表示为 1101 1000B=D8H。

（注：如果 PC 增量在最后做，即执行转移指令的目标地址计算时，PC 为 259，使得结果为-39，也算正确。）

**批改情况：错误较多，①没有考虑 PC 内容在取操作码和位移量后会改变，多数  
同学得到-38；②补码表示错误**

6. 某计算机指令系统采用定长指令字格式，指令字长 16 位，每个操作数的地址码长 6 位。指令分二地址、单地址和零地址三类。若二地址指令有  $k_2$  条，无地址指令有  $k_0$  条，则单地址指令最多有多少条？

**【分析解答】**

假设单地址指令有  $k_1$  条，则  $((16 - k_2) \times 2^6 - k_1) \times 2^6 = k_0$ ，所以  $k_1 = (16 - k_2) \times 2^6 - k_0 / 2^6$

**批改情况：基本正确**

7. 某计算机字长 16 位，存储器存取宽度为 16 位，即每次从存储器取出 16 位。CPU 中有 8 个 16 位通用寄存器。现为该机设计指令系统，要求指令长度为字长的整数倍，至多支持 64 种不同操作，每个操作数都支持 4 种寻址方式：立即（I）、寄存器直接（R）、寄存器间接（S）和变址（X）寻址方式。存储器地址位数和立即数均为 16 位，任何一个通用寄存器都可作变址寄存器，支持以下 7 种二地址指令格式（R、I、S、X 代表上述 4 种寻址方式）：RR 型、RI 型、RS 型、RX 型、XI 型、SI 型、SS 型。请设计该指令系统的 7 种指令格式，给出每种格式的指令长度、各字段所占位数和含义，并说明每种格式指令的功能以及需要的访存次数？

**【分析解答】**

因为至多有 64 种操作，所以操作码字段只需要 6 位；有 8 个通用寄存器，所以寄存器编号至少占 3 位；寻址方式有 4 种，所以寻址方式位至少占 2 位；直接地址和立即数都是 16 位；任何通用寄存器都可作变址寄存器，所以指令中要明显指定变址寄存器，其编号占 3 位；指令总位数是 16 的倍数。此外，指令格式应尽量规整，指令长度应尽量短。按照上述这些要求设计出的指令格式可以有很多种。

以下是采用二地址指令格式的两种指令格式设计方案，RI、XI 和 SI 三种指令格式中添了 3 个 0，是为了补足位数，以使指令长度为 16 的倍数。这两种方案得到的 RR、RS 和 SS 型指令都是 16 位，RI、RX 和 SI 型指令都是 32 位，XI 型指令是 48 位。

指令格式示例 1：如图 1 所示，用专门的“类型”字段（最左 3 位）说明不同指令类型，这样无需对两个操作数的寻址方式分别进行说明。7 种指令类型只要 3 位编码即可，之后的一位总是 0，这一位在需要扩充指令操作类型时可作为 OP 中新的编码位。

RR 型	000 0	OP (6 位)	Rt (3 位)	Rs (3 位)	
RI 型	001 0	OP (6 位)	Rt (3 位)	000	Imm16 (16 位)
RS 型	010 0	OP (6 位)	Rt (3 位)	Rs (3 位)	
RX 型	011 0	OP (6 位)	Rt (3 位)	Rx (3 位)	Offset16 (16 位)
XI 型	100 0	OP (6 位)	Rx (3 位)	000	Offset16 (16 位) Imm16 (16 位)
SI 型	101 0	OP (6 位)	Rt (3 位)	000	Imm16 (16 位)
SS 型	110 0	OP (6 位)	Rt (3 位)	Rs (3 位)	

图 1 第一种指令格式示例

指令格式示例 2：如图 2 所示，用专门的“寻址方式”字段分别说明两个操作数的寻址方式。其定义为 00-立即；01-寄直；10-寄间；11-变址。这种格式相当于用 4 位编码来说明指令格式，比第一种指令格式多用了一位编码，因此可扩展性没有第一种指令格式好。

RR 型	OP (6 位)	01	01	Rt (3 位)	Rs (3 位)	
RI 型	OP (6 位)	01	00	Rt (3 位)	000	Imm16 (16 位)
RS 型	OP (6 位)	01	10	Rt (3 位)	Rs (3 位)	
RX 型	OP (6 位)	01	11	Rt (3 位)	Rx (3 位)	Offset16 (16 位)
XI 型	OP (6 位)	11	00	Rx (3 位)	000	Offset16 (16 位) Imm16 (16 位)
SI 型	OP (6 位)	10	00	Rt (3 位)	000	Imm16 (16 位)
SS 型	OP (6 位)	10	10	Rt (3 位)	Rs (3 位)	

图 2 第二种指令格式示例

存储器存取宽度为 16 位，每次从存储器取出 16 位。因此，读取 16、32 和 48 位指

令分别需要 1、2 和 3 次存储器访问。各类指令的功能和访存次数分别说明如下（指令功能用 RTL 表示，其中  $M[x]$  表示存储器地址  $x$  中的内容， $R[x]$  表示寄存器  $x$  中的内容）。

RR 型指令功能为  $R[Rt] \leftarrow R[Rt] \text{ op } R[Rs]$ , 取指令时访存 1 次;

RI 型指令功能为  $R[Rt] \leftarrow R[Rt] \text{ op Imm16}$ , 取指令时访存 2 次;

RS型指令功能为  $R[Rt] \leftarrow R[Rt] op M[R[Rs]]$ , 取指令和取第2个源操作数各访存1次, 共访存2次;

RX 型指令功能为  $R[Rt] \leftarrow R[Rt] op M[R[Rx]+Offset]$ , 取指令访存 2 次, 取第 2 个源操作数访存 1 次, 共访存 3 次;

XI 型功能为  $M[R[Rx]+Offset] \leftarrow M[R[Rx]+Offset] op Imm16$ , 取指令访存 3 次, 取第一个源操作数访存 1 次, 写结果访存 1 次, 共访存 5 次;

SI 型指令功能为  $M[R[Rt]] \leftarrow M[R[Rt]] \text{ op Imm16}$ , 取指令访存 2 次, 取第一个源操作数和写结果各访存 1 次, 共访存 4 次;

SS 型功能为  $M[R[Rt]] \leftarrow M[R[Rt]] \text{ op } M[R[Rs]]$ , 取指令访存 1 次, 取第一个源操作数、取第二个源操作数和写结果各访存 1 次, 共访存 4 次。

批改情况：指令格式设计基本正确，访存次数错误较多：①没有回答访存次数；②需要知道各种指令的具体功能以及取指令、取操作数、写结果都是需要访问存储器的

8. 某计算机字长为 16 位，主存地址空间大小为 128 KB，按字编址。采用单字长定长指令格式，指令各字段定义如下：



转移指令采用相对寻址方式，相对偏移量用补码表示。寻址方式定义如表 7.4 所示。

表 7.4 题 8 中定义的寻址方式及其含义

Ms / Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=R[Rn]
001B	寄存器间接	(Rn)	操作数=M[R[Rn]]
010B	寄存器间接、自增	(Rn)+	操作数 =M[R[Rn]], R[Rn]←R[Rn]+1
011B	相对	D(Rn)	转移目标地址=PC+R[Rn]

(注:  $M[x]$  表示存储器地址  $x$  中的内容,  $R[x]$  表示寄存器  $x$  中的内容)

请回答下列问题：

(1) 该指令系统最多可有多少条指令？该计算机最多有多少个通用寄存器？存储器地址寄存器（MAR）和存储器数据寄存器（MDR）至少各需要多少位？

(2) 转移指令的目标地址范围是多少？

(3) 若操作码 0010B 表示加法操作（助记符为 add），寄存器 R4 和 R5 的编号分别为 100B 和 101B，R4 的内容为 1234H，R5 的内容为 5678H，地址 1234H 中的内容为 5678H，地址 5678H 中的内容为 1234H，则汇编语句“add (R4), (R5)+”（逗号前为第一源操作数，逗号后为第二源操作数和目的操作数）对应的机器码是什么（用十六进制表示）？该指令执行后，哪些寄存器和存储单元的内容会改变？改变后的内容是什么？

### 【分析解答】

(1) 因为采用单字长指令格式，操作码字段占 4 位，所以最多有 16 条指令；指令中通用寄存器编号占 3 位，所以，最多有 8 个通用寄存器；因为地址空间大小为 128KB，按字编址，故共有 64K 个存储单元，因而地址位数为 16 位，所以 MAR 至少为 16 位；因为字长为 16 位，所以 MDR 至少为 16 位。

(2) 因为地址位数和字长都为 16 位，所以 PC 和通用寄存器位数均为 16 位，两个 16 位数据相加其结果也为 16 位，即转移目标地址位数为 16 位，因而能在整个地址空间转移，即目标转移地址的范围为 0000H~FFFFH。

(3) 要得到汇编语句“add (R4),(R5)+”对应的机器码，只要将其对应的指令代码各个字段拼接起来即可。显然，add 对应 op 字段，为 0010B；(R4)的寻址方式字段为 001B，R4 的编号为 100B；(R5)+的寻址方式字段为 010B，R5 的编号为 101B；因此，对应的机器码为 0010 001 100 010 101B，用十六进制表示为 2315H。指令“add (R4),(R5)+”的功能为： $M[R5] \leftarrow M[R[R5]] + M[R[R4]]$ ， $R[R5] \leftarrow R[R5] + 1$ 。已知  $R[R4]=1234H$ ， $R[R5]=5678H$ ， $M[1234H]=5678H$ ， $M[5678H]=1234H$ ，因为  $1234H + 5678H = 68ACH$ ，所以 5678H 单元中的内容从 1234H 改变为 68ACH，同时 R5 中的内容从 5678H 变为 5679H

**批改情况：基本正确**

10.对于远距离的过程调用，使用伪指令“call offset”作为调用指令，它对应以下两条

真实指令：

```
auipc x1, offset[31:12]+offset[11]    # R[x1]←PC+ (offset[31:12]+offset[11]) <<12  
jalr x1, x1, offset[11:0]             # PC←R[x1]+offset[11:0], R[x1]←PC+4
```

请说明为什么在 auipc 指令中高 20 位的位移量计算时 offset[31:12]需要加上 offset[11]？

### 【分析解答】

因为上述 jalr 指令进行加法运算时，需要对 x1 寄存器中的  
PC+(offset[31:12]+offset[11]) <<12 与 offset[11:0] 符号扩展结果进行相加。

若 offset[11:0] 的最高位 offset[11]（看成是低 12 位数的符号位）是 0，则 PC 所加的高 20 位应该是 offset[31:12]+0；若 offset[11] 是 1，则 offset[11:0] 符号扩展后高 20 位为全 1，此时，PC 所加的高 20 位应为 offset[31:12]+1。综上所述，PC 所加的高 20 位应该为 offset[31:12]+offset[11]。

**批改情况：正确**

11. 除了硬件乘法器外，还可以用移位和加法指令来实现乘法运算。在乘以较小的常数时，这种办法很有效。在不考虑溢出的情况下，假设要将 t0 的内容与 7 相乘，乘积存入 t1 中。请写出一段指令条数最少且不包括乘法指令的 RV32I 代码。

**【分析解答】**

一个数 x 乘以 6，相当于  $8x - x$ ，而  $8x$  可以通过将 x 左移 3 位来实现，这样就只要用一条左移指令和一条减法指令来实现乘 7 操作。以此类推，当乘以一个较小的常数时，只要将这个较小的常数分解成若干个 2 的幂次相加/减，就可以用若干条左移指令和一条加/减法指令来实现乘法运算。可用以下指令序列实现题目要求。

```
sll    t1,  t0,  3      #将t0的内容左移3位，送t1  
sub   t1,  t1,  t0      #将t1和t0的内容相减，送t1
```

**批改情况：基本正确，需要注意题目要求是指令数最少，部分同学功能正确但指令可以化简**

13. 以下程序段是某个过程对应的指令序列。入口参数 int a 和 int b 分别置于 a0 和 a1 中，返回参数是该过程的结果，置于 a0 中。要求为以下 RV32I 指令序列加注释，并简单说明该过程的功能。

```
add   t0, zero, zero  
loop: beq  a1, zero, finish  
      add  t0, t0, a0  
      addi a1, a1, -1  
      j    loop          # “jal x0, loop” 指令的伪指令  
finish: addi t0, t0, 100  
       add  a0, t0, zero
```

**【分析解答】**

```
add   t0, zero, zero      #将寄存器 t0 置 0  
loop: beq  a1, zero, finish    #若 a1 的值等于 0，则转 finish 处
```

```

add t0, t0, a0          #将 t0 和 a0 的内容相加, 送 t0
addi a1, a1, -1        #将 a1 的值减 1
j loop                 #无条件转到 loop 处
finish: addi t0, t0, 100 #将 t0 的内容加 100
add a0, t0, zero       #将 t0 的内容送 a0

```

该过程的功能是计算 “ $a \times b + 100$ ”。

**批改情况：正确**

15. 假定编译器将 a 和 b 分别分配到 t0 和 t1 中, 用一条 RV32I 指令或最短的 RV32I 指令序列实现以下 C 语言语句:  $b=31\&a$ 。如果把 31 换成 65535, 即  $b=65535\&a$ , 则用 RV32I 指令或指令序列如何实现?

**【分析解答】**

只要用一条指令 “andi t1, t0, 31” 就可实现  $b=31\&a$ , 其中 12 位立即数为 0000 0001 1111。但是, 如果把 31 换成 65535, 则不能用一条指令 “andi t1, t0, 65535” 来实现, 因为  $65535 = 1111\ 1111\ 1111\ 1111$ B, 它不能用 12 位立即数表示。可用以下 3 条指令实现  $b=65535\&a$ 。

```

lui t1, 16      #将 0001 0000H 置于寄存器 t1
addi t1, t1, 4095 #将 0001 0000H 与 FFFF FFFFH 相加, 送 t1
and t1, t0, t1    #将 t0 和 t1 的内容进行“与”运算, 送 t1

```

**批改情况：正确**

17. 请说明 RV32I 中 beq 指令的含义, 并解释为什么汇编程序在对下列汇编语言源程序中的 beq 指令进行汇编时会遇到问题, 应该如何修改该程序段?

```

here: beq t0, t2, there
.....
there: addi t1, a0, 4

```

**【分析解答】**

在 RV32I 指令系统中, 分支指令 beq 是 B-型指令, 转移目标地址采用相对寻址方式, 其偏移量为 imm[12:1]乘 2, 相当于在 imm[12:1]后面添一个 0, 再符号扩展为 32 位, 即转移目标地址= $PC+SEXT[imm[12:1]<<1]$ 。

12 位立即数用补码表示, 得到偏移量 offset, 从而计算转移目标地址, 因此 beq 指令执行时若条件满足, 则可能跳转到当前指令前, 也可能跳转到当前指令后, 当 offset 的范围为 0000 0000 0000 0B ~ 0111 1111 1111 0B 时, beq 指令向后(正)跳, 最多向后

跳 1023 条指令；当 offset 的范围为 1000 0000 0000 0B ~ 1111 1111 1111 0B 时，beq 指令向前（负）跳，最多向前跳 1024 条指令。

当上述指令序列中 here 和 there 表示的地址相差超过上述 offset 的范围时，会因为无法得到正确的立即数而使得 beq 指令发生汇编错误。

可将上述指令序列改成以下指令序列，因为 here 和 skip 之间仅相差两条指令，因而 bne 指令中的立即数不会超过可表示范围；而无条件跳转指令 jal x0, there 中可以表示 20 位的立即数，跳转范围为当前指令的前 262 144 到后 262 143 条指令），只要 there 处离 jal 指令在 262 143 条指令范围内，就可以直接跳转到 there。

here: bne t0, t2, skip

jal x0, there

skip:

.....

there: addi t1, a0, 4

批改情况：基本正确，关键在于 beq 指令的跳转范围

## 第8章 中央处理器

作业：习题 3、4、5、6、11、13、14、17、18

3. 某计算机字长 16 位，标志寄存器 Flag 中的 ZF、SF 和 OF 分别是零标志、符号标志和溢出标志，采用双字节定长指令字。假定该计算机中有一条 bgt(大于零转移) 指令，其指令格式如下：第一个字节指明操作码和寻址方式，第二个字节为偏移地址 Imm8，其功能是：

若  $(ZF + (SF \oplus OF) = 0)$  则  $PC = PC + 2 + Imm8$  否则  $PC = PC + 2$

完成如下要求并回答问题：

(1) 该计算机存储器的编址单位是多少位？

增加问题：bgt 指令执行的是带符号整数比较还是无符号整数比较？偏移地址

Imm8 的含义是什么？转移目标地址的范围是什么？

(2) 画出实现 bgt 指令的数据通路。

### 【分析解答】

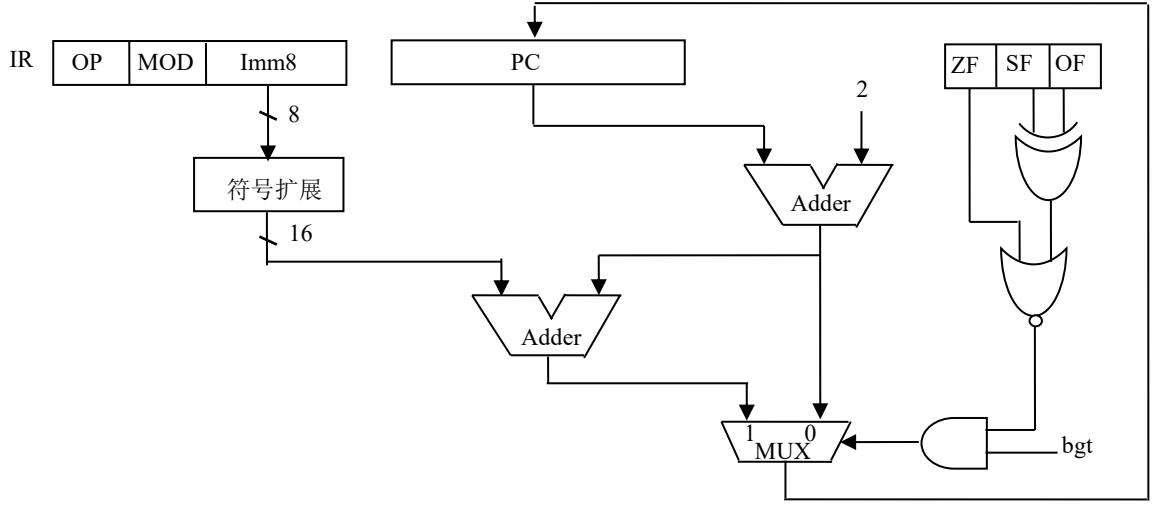
(1) 因为顺序执行时 PC 的增量是 2，且每条指令占 2 个字节，所以编址单位是字节。

增加问题的参考答案：

根据“大于 0”的条件判断表达式 $(ZF + (SF \oplus OF) == 0)$ ，可以看出该 bgt 指令实现的是带符号整数比较。因为无符号整数比较大小时，其判断表达式中应该有借位标志 CF，而没有溢出标志 OF 和符号标志 SF。

偏移地址 Imm8 为补码表示，说明转移目标指令可能在 bgt 指令之前，也可能在 bgt 指令之后。计算转移目标地址时，偏移量为 Imm8，而不是  $Imm8 \times 2$ ，说明 Imm8 是相对地址，而不是相对指令条数。Imm8 占 8 位，即范围为 -128~127，但因为采用双字节定长指令字，所以偏移量不可能是奇数，因此转移目标地址的范围是  $PC + 2 + (-128) \sim PC + 2 + 126$ ，也即转移目标地址的范围是相对于 bgt 指令的前 126 个单元到后 128 个单元之间，用指令条数来衡量的话，就是相对于 bgt 指令的前 63 条指令到后 64 条指令之间。

(2) 实现 bgt 指令的数据通路如下图所示。



bgt 指令的数据通路

4. 假定图 8.18 给出的单周期数据通路对应的控制逻辑发生错误，使得控制信号 RegWr、ALUASrc、Branch、Jump、MemWr、MemtoReg 中某一个在任何情况下总是为 0，则该控制信号为 0 时哪些指令不能正确执行？要求分别对每个控制信号进行讨论。

#### 【分析解答】

若 RegWr=0，则所有需写结果到寄存器的指令（如：R-型指令、I-型运算类指令、load 指令等）都不能正确执行，因为寄存器不发生写操作。

若 ALUASrc=0，则 J 型指令（jal）可能不能正确执行，当指令中的目标寄存器 rd 不是 x0（0 号寄存器）时，需要将 PC+4（返回地址）存入 rd 中，但因为 ALUASrc=0，使得 PC 不能作为 ALU 的 A 输入端，因而在 ALU 中不能完成 PC+2 的计算。

若 Branch=0，则 branch 类（B 型）指令可能出错，因为永远不会发生跳转。

若 Jump=0，则 J 型指令（jal）可能出错，因为永远不会发生跳转。

若 MemWr=0，则 store 指令（sb、sh、sw）不能正确执行，因为存储器不能写入所需数据。

若 MemtoReg=0，则 load 指令（lb、lh、lw）发生错误，因为不能选择将存储器读出的内容送目的寄存器。

5. 假定图 8.18 给出的单周期数据通路对应的控制逻辑发生错误，使得控制信号 RegWr、ALUASrc、Branch、Jump、MemWr、MemtoReg 中某一个在任何情况下总是

为 1，则该控制信号为 1 时哪些指令不能正确执行？要求分别对每个控制信号进行讨论。

### 【分析解答】

若  $\text{RegWr}=1$ ，则所有无需写结果到寄存器的指令（如：store 指令、branch 类（B 型）指令等）都不能正确执行，因为寄存器发生了不该写结果的操作。

若  $\text{ALUASrc}=1$ ，则除了 J 型指令（jal）以外的需要在 ALU 中进行计算的指令都可能执行错误。

若  $\text{Branch}=1$ ，则除 branch 类（B 型）指令以外的指令可能出错，因为可能会发生不必要的跳转。

若  $\text{Jump}=1$ ，则除 J 型指令（jal）以外的指令都可能出错，因为会发生不必要的跳转。

若  $\text{MemWr}=1$ ，则除 store 指令（sb、sh、sw）以外的指令正确错误，因为存储器写入了不该写的数据。

若  $\text{MemtoReg}=1$ ，则除 load 指令（lb、lh、lw）以外的需要写 ALU 运算结果到寄存器的指令都会发生错误，因为选择了将存储器读出的内容送目的寄存器，而不是将 ALU 的结果送目的寄存器。

6. 若要在 RV32I 指令集中增加一条 swap 指令（功能是实现两个寄存器内容的互换），可以有两种方式。一种是采用伪指令（即软件）方式，这种情况下，当执行到 swap 指令时，用若干条已有指令构成的指令序列来代替实现；另一种做法是直接改动硬件来实现 swap 指令，这种情况下，当执行到 swap 指令时，则可直接在 swap 指令对应的数据通路（硬件）上执行。

(1) 写出用伪指令方式实现“swap rs, rt”时的指令序列（提示：伪指令对应的指令序列中不能使用其他额外寄存器，以免破坏这些寄存器的值）。

(2) 假定用硬件实现 swap 指令时会使每条指令的执行时间增加 10%，则 swap 指令在程序中占多大的比例才值得用硬件方式来实现，而不是用（1）中给出的伪指令方式实现？

(3) 采用硬件方式实现时，在不对通用寄存器组进行修改的情况下，能否在单周期数据通路中实现 swap 指令？对于多周期数据通路的情况又怎样？

### 【分析解答】

(1) 若在伪指令“swap rs, rt”的指令序列中使用除 rs 和 rt 以外的额外寄存器，则额外寄存器的内容会被指令序列破坏，因此，伪指令的指令序列中一般不能用额外寄存器。伪指令“swap rs, rt”的指令序列包含以下三条指令，没有用到额外寄存器。

```
xor rs, rs, rt  
xor rt, rs, rt  
xor rs, rs, rt
```

(2) 假定“swap rs, rt”指令所占比例为  $x$  ( $0 \leq x \leq 1$ )，其他指令比例为  $1-x$ ，则用硬件实现该指令时，程序执行时间为原来的  $1.1 \times (x+1-x) = 1.1$  倍。用软件实现该指令时，程序执行时间为原来的  $3x+1-x = 2x+1$  倍。因此，当  $1.1 < 2x+1$  时，硬件实现才有意义，由此可知， $x > 0.05$ ，也即：当“swap rs, rt”指令在程序中的比例大于 5% 时，才值得用硬件方式来实现该指令。

(3) 在单周期数据通路中，所有指令的执行都在一个时钟周期内完成，数据总是在时钟边沿被写入寄存器堆，即本条指令执行的结果总是下条指令开始（即下个时钟到来）时，才开始被写到通用寄存器组中，因此一个时钟周期只能写一次寄存器。而 swap 指令执行过程中需要多次写寄存器，在不对通用寄存器组进行修改的情况下，无法在单周期数据通路中实现 swap 指令；对于多周期数据通路，一个指令周期可以有多个时钟周期，因而可以多次写寄存器，因此，在不对通用寄存器组进行修改的情况下，swap 指令可以在多周期数据通路中实现。

11. 假定在一个 5 级流水线（如图 8.42 所示）处理器中，各主要功能单元的操作时间为：存储单元-200ps；ALU 和加法器-150ps；通用寄存器组的读口或写口-50ps。请问：

- (1) 若执行阶段 EX 所用的 ALU 操作时间缩短 20%，则能否加快流水线执行速度？如果能的话，能加快多少？如果不能的话，为什么？
- (2) 若 ALU 操作时间增加 20%，对流水线的性能有何影响？
- (3) 若 ALU 操作时间增加 40%，对流水线的性能又有何影响？

### 【分析解答】

(1) ALU 操作时间缩短 20% 不能加快流水线指令速度。因为指令流水线的执行速度取决于最慢的功能部件所用时间，最慢的是存储器，只有缩短了存储器的操作时间才可能加快流水线速度。

(2) ALU 操作时间延长 20% 时，变为了 180ps，比存储器所用时间 200ps 还小，因此，对流水线性能没有影响。

(3) ALU 操作时间延长 40% 时，变为了 210ps，比存储器所用时间 200ps 大，因此，在不考虑流水段寄存器延时的情况下，流水线的时钟周期从 200ps 变为 210ps，流水线执行速度降低了  $(210-200)/200=5\%$ 。

13. 假定最复杂的一条指令所用的组合逻辑分成 6 个部分，依次为 A~F，其延迟分别为 80ps、30ps、60ps、50ps、70ps、10ps。在这些组合逻辑块之间插入必要的流水段寄存器就可实现相应的指令流水线，寄存器延迟为 20ps。理想情况下，以下各种方式所得到的时钟周期、指令吞吐率和指令执行时间各是多少？应该在哪里插入流水段寄存器？

- (1) 插入一个流水段寄存器，得到一个两级流水线。
- (2) 插入两个流水段寄存器，得到一个三级流水线。
- (3) 插入三个流水段寄存器，得到一个 4 级流水线。
- (4) 吞吐量最大的流水线。

#### 【分析解答】

(1) 两级流水线的平衡点在 C 和 D 之间，其前面一个流水段的组合逻辑延时为  $80+30+60=170\text{ps}$ ，后面一个流水段的组合逻辑延时为  $50+70+10=130\text{ps}$ 。最长功能段延时为  $170\text{ps}$ ，加上流水段寄存器延时  $20\text{ps}$ ，因而时钟周期为  $190\text{ps}$ ，理想情况下，指令吞吐率为每秒钟执行  $1/190\text{ps}=5.26\text{G}$  条指令。每条指令在流水线中的执行时间为  $2\times190=380\text{ps}$ 。

(2) 两个流水段寄存器分别插在 B 和 C、D 和 E 之间，这样第一个流水段的组合逻辑延时为  $80+30=110\text{ps}$ ，中间第二段的延时为  $60+50=110\text{ps}$ ，最后一个段延时为  $70+10=80\text{ps}$ 。这样，每个流水段所用时间都按最长延时调整为  $110+20=130\text{ps}$ ，故时钟周期为  $130\text{ps}$ ，指令吞吐率为每秒钟执行  $1/130\text{ps}=7.69\text{G}$  条指令，每条指令在流水线中的执行时间为  $3\times130=390\text{ps}$ 。

(3) 三个流水段寄存器分别插在 A 和 B、C 和 D、D 和 E 之间，这样第一个流水段的组合逻辑延时为  $80\text{ps}$ ，第二段延时为  $30+60=90\text{ps}$ ，第三段延时为  $50\text{ps}$ ，最后一段延时为  $70+10=80\text{ps}$ 。这样，每个流水段都以最长延时调整为  $90+20=110\text{ps}$ ，故时钟周期为  $110\text{ps}$ ，指令吞吐率为每秒钟执行  $1/110\text{ps}=9.09\text{G}$  条指令，每条指令在流水线中的执行时间为  $4\times110=440\text{ps}$ 。

(4) 因为各功能部件对应的组合逻辑中最长延时为  $80\text{ps}$ ，所以，流水线的时钟周期肯定比  $80\text{ps}+20\text{ps}=100\text{ps}$  长。为了达到最大吞吐率，时钟周期应该尽量短，因此，最合理的划分方案应该按照每个时钟周期为  $100\text{ps}$  来进行。根据每个功能部件所用时间可知，流水线至少按 5 段来划分，分别把流水线寄存器插入在 A 和 B、B 和 C、C 和 D、D 和 E 之间，这样各段的组合逻辑延时为  $80\text{ps}$ 、 $30\text{ps}$ 、 $60\text{ps}$ 、 $50\text{ps}$  和  $80\text{ps}$ 。其中，最后一个延时  $80\text{ps}$  是 E 和 F 两个阶段的时间相加而得到的。这样时钟周期为  $100\text{ps}$ ，指令吞吐率为每秒钟执行  $1/100\text{ps}=10\text{G}$  条指令，每个指令的执行时间为  $5\times100=500\text{ps}$ 。

通过对上述 4 种情况进行分析，可以得出以下结论：划分的流水段多，时钟周期就变短，指令执行吞吐率就变高，而相应的额外开销（即插入的流水段寄存器的延时）也变大，使得一条指令的执行时间变长。

14. 以下指令序列中，哪些指令对之间发生数据相关？假定采用“取指、译码/取数、执行、访存、写回”5 段流水线方式，那么不用“转发”技术的话，需要在发生数据相关的指令前加入几条 nop 指令才能使这段程序避免数据冒险？如果采用“转发”是否可以完全解决数据冒险？不行的话，需要在发生数据相关的指令前加入几条 nop 指令才能使这段 RV32I 程序不发生数据冒险？

```

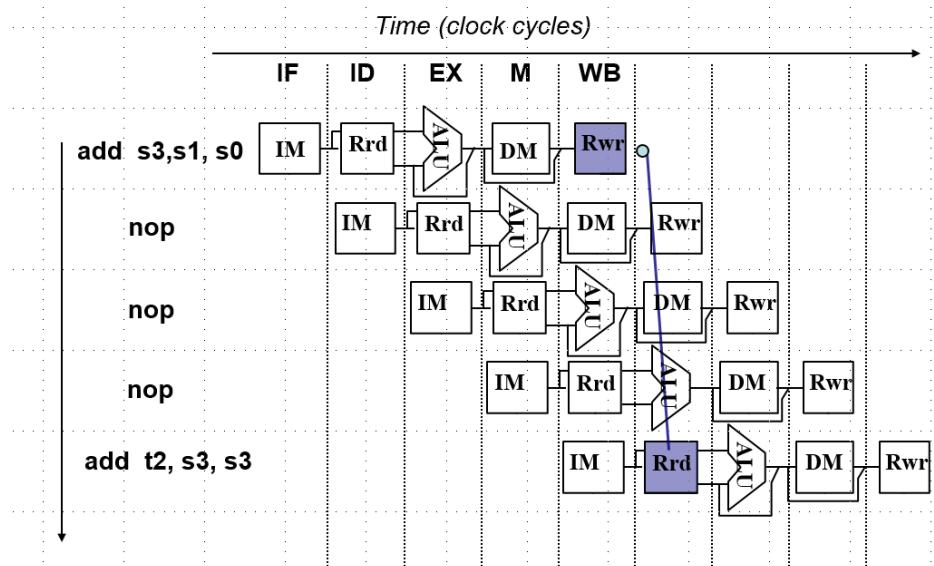
add s3, s1, s0
add t2, s3, s3
lw t1, 0(t2)
add t3, t1, t2

```

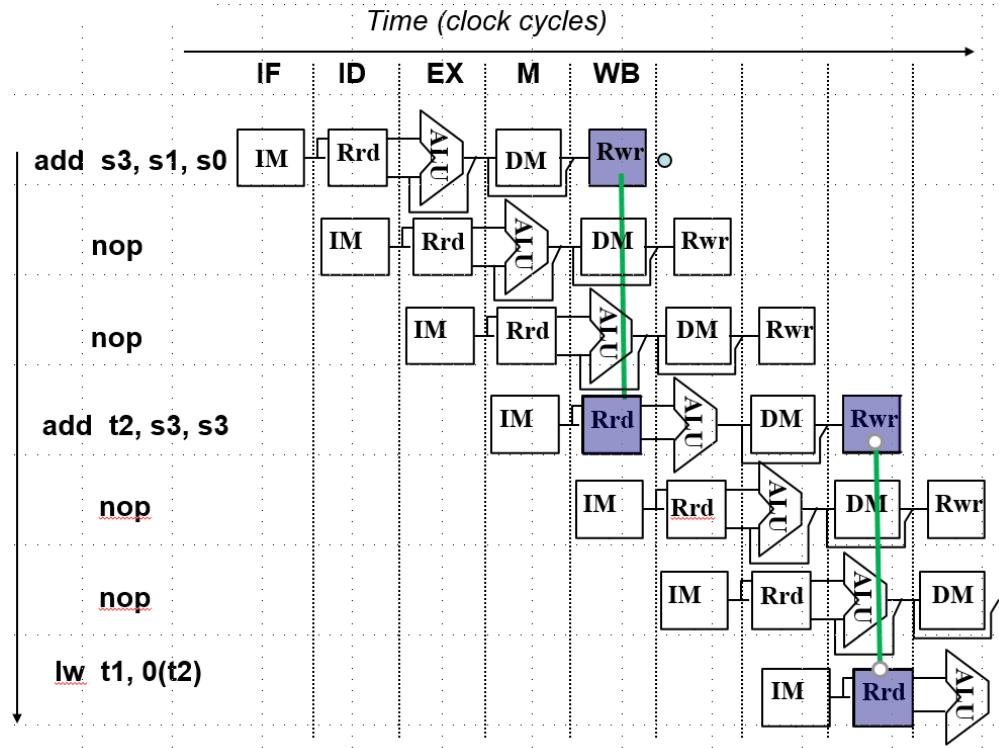
#### 【分析解答】

第 1 和第 2 条指令、第 2 和第 3 条指令、第 2 条和第 4 条指令、第 3 条和第 4 条指令之间发生数据相关。

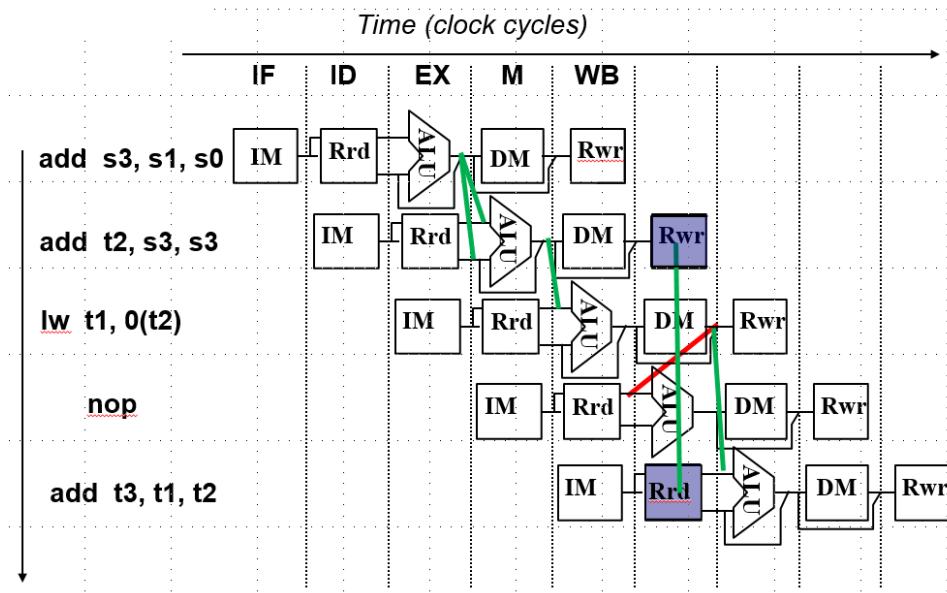
如下图所示，在一个 5 段流水线处理器中，若不采用“转发”技术，则为避免相邻两条指令之间的数据冒险，需要在两条指令之间插入 3 条 nop 指令。



不过，通常在 WB 阶段的前半周期能够写入寄存器，在相邻的后 1 条指令的 ID 阶段的后半周期可以读到寄存器中的新数据（上半周期写，下半周期读），因而在相邻两条数据相关的指令之间插入两条 nop 指令即可。因此，为避免数据冒险，在上述指令序列中，需在前 3 条指令后各插入 2 条 nop 指令（如下图所示）。



可以采用“转发”技术解决第 1 和第 2 条指令、第 2 和第 3 条指令、第 2 和第 4 条指令之间的数据冒险；但是，因为第 3 和第 4 条指令之间是 Load-use 数据冒险，因而不能通过转发技术解决（参见下图中的红线），需要在第 3 条指令后加一条 nop 指令（如下图所示）。



17. 假定在一个带“转发”逻辑的 5 段流水线中执行以下 RV32I 程序段，应怎样调整指令序列使其性能达到最好？

lw s2, 100(s6)

```

add      s2, s2, s3
lw       s3, 200(s7)
add      s6, s4, s7
sub      s3, s4, s6
lw       s2, 300(s8)
beq     s2, s8, Loop

```

### 【分析解答】

因为采用“转发”技术，所以，只要对 load-use 数据冒险进行指令序列调整。从上述指令序列来看，第 1 和第 2 条指令、第 6 和第 7 条指令之间存在 load-use 数据冒险，所以，可将与第 2 和第 3 条指令无关的第 4 条指令插入第 2 条指令之前；将与第 6 和第 7 条无关的第 5 条指令插入第 7 条指令之前。调整顺序后的指令序列如下（粗体部分为变换位置的指令）。

```

1      lw   s2, 100(s6)
4      add  s6, s4, s7
2      add  s2, s2, s3
3      lw   s3, 200(s7)
6      lw   s2, 300(s8)
5      sub  s3, s4, s6
7      beq  s2, s8, Loop

```

18. 在一个采用“取指、译码/取数、执行、访存、写回”的 5 段流水线中，若检测相减结果是否为“零”的操作在执行阶段进行，则分支延迟损失时间片（即分支延迟槽）为多少？以下一段 RV32I 指令序列中，在考虑数据转发的情况下，哪些指令执行时会发生流水线阻塞？各需要阻塞几个时钟周期？

```

loop:  add  t1, s3, s3
        add  t1, t1, t1
        add  t1, t1, s6
        lw   t0, 0(t1)
        bne t0, s5, Exit
        add  s3, s3, s4
        j    loop

```

Exit:

### 【分析解答】

在书中图 8.42 所示的 5 段流水线中，检测结果是否为“零”并更新 PC 的操作在“访存 (M)”阶段进行，这种情况下，分支延迟损失时间片（分支延迟槽）为 3。

若检测结果是否为“零”并更新 PC 的操作提前到在“执行 (EX)”阶段进行，则分支

延迟损失时间片（分支延迟槽）变为 2。

在采用数据转发技术的情况下，上述指令序列中，第 4 条和第 5 条指令之间为 Load-use 冒险，无法通过转发技术解决，此时第 5 条 `bne` 指令的执行被阻塞一个时钟。第 5 条指令是分支指令 `bne`，由于该指令执行时条件检测结果不能马上得到，因此其后指令的执行被阻塞，阻塞的时钟周期数等于分支延迟损失时间片。第 7 条指令为无条件跳转指令 `jal`（`j loop` 是其伪指令），假定“`j loop`”指令更新 PC 的操作在“执行（EX）”阶段进行，则其后指令的执行将被阻塞 2 个时钟周期；假定更新 PC 的操作在“译码（ID）”阶段进行，则被阻塞 1 个时钟周期。