

以下是希望大家学完这门  
课之后能够留下的记忆

并非和考试直接相关

# 冯·诺依曼结构

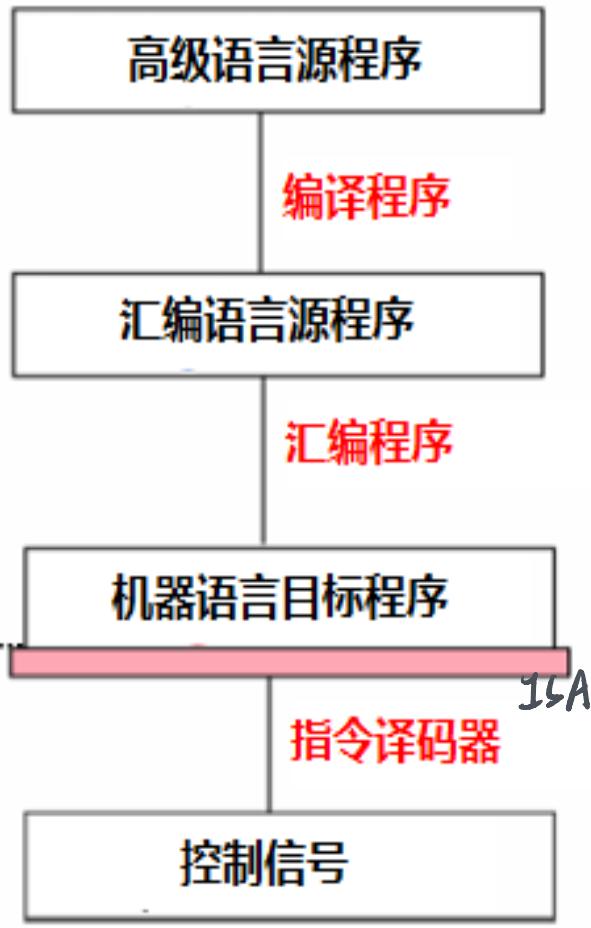
1. 各基本部件的功能是：

- 存储器存放数据、指令，都是二进制，但能区分；
- 控制器自动取出指令来执行；
- 运算器进行算术逻辑等运算；
- 通过输入设备、输出设备和主机进行通信。

2. 采用“**存储程序**”工作方式。

# 不同层次语言之间的等价转换

计算机软件  
计算机硬件



tmp = a[i];  
a[i] = a[i+1];  
a[i+1] = tmp;

lw \$8, 0(\$2)  
lw \$9, 4(\$2)  
sw \$9, 0(\$2)  
sw \$8, 4(\$2)

每条指令由操作码和若干地址码组成

该图展示了机器语言的字节级表示。每行代表一个32位的机器字，由四部分组成：操作码（最左侧，红色）、地址码（中间，黄色）、地址码（中间，绿色）、和高位地址码（最右侧，红色）。图中共有四行，每行的地址码部分都有一个红色的“X”符号，表示这些位在当前讨论的指令中是无效的或未定义的。

100011	00010	01000	0000000000000000
100011	00010	01001	000000000000100
101011	00010	01001	0000000000000000
101011	00010	01000	000000000000100

... , EXTop=1, ALUSelA=1, ALUSelB=11, ALUop=add,  
IorD=1, Read, MemtoReg=1, RegWr=1, ...

任何高级（汇编）语言程序最终都要通过执行机器指令来完成！

高级语言程序和汇编（或机器）语言程序都是一对多的关系

汇编语言程序和机器语言程序肯定是一对一的关系

# 指令集体系结构 (ISA)

- ISA指Instruction Set Architecture, 即指令集体系结构
- ISA规定了如何使用硬件，
  - 核心是指令系统，包括指令格式、操作种类以及每种操作对应的操作数的相应规定；指令可以接受的操作数的类型；指令中操作数的寻址方式；
  - 操作数所能存放的寄存器的名称、编号、长度和用途；
  - 操作数所能存放的存储空间的大小和编址方式；
  - 操作数在存储空间存放时按照大端还是小端方式存放；
  - 指令执行过程的控制方式，包括程序计数器、条件码定义等。

# 数据的编码

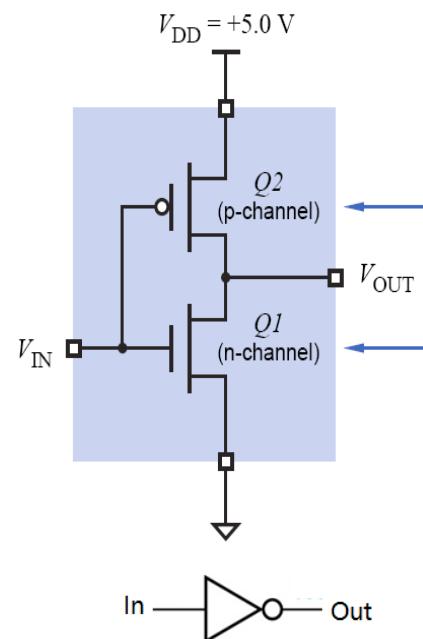
- 在机器内部编码后的数称为**机器数**，其值称为**真值**
- 定义数值数据有三个要素：进制、定点/浮点、编码
- **整数的表示**
  - 无符号数：正整数，就是二进制，用来表示地址等；
  - 带符号整数：用补码表示；
- **浮点数的表示**
  - 符号位；
  - 尾数：定点小数；
  - 指数（阶）：定点整数（基不用表示）
- **浮点数的范围**：与阶码的位数和基的大小有关
- **浮点数的精度**：与尾数的位数和是否规格化有关

# 数据的宽度和存储

- 数据的宽度
  - 位、字节、字（不一定等于字长），k/K/M/G/...
- 数据的存储排列
  - 数据的地址：连续若干单元中最小的地址，即：从小地址开始存放数据
    - 若一个short型（16位）数据si存放在单元0x0100和0x0101中，那么si的地址是什么——0x0100
    - 大端方式：用MSB存放的地址表示数据的地址
    - 小端方式：用LSB存放的地址表示数据的地址

# 数字逻辑基础

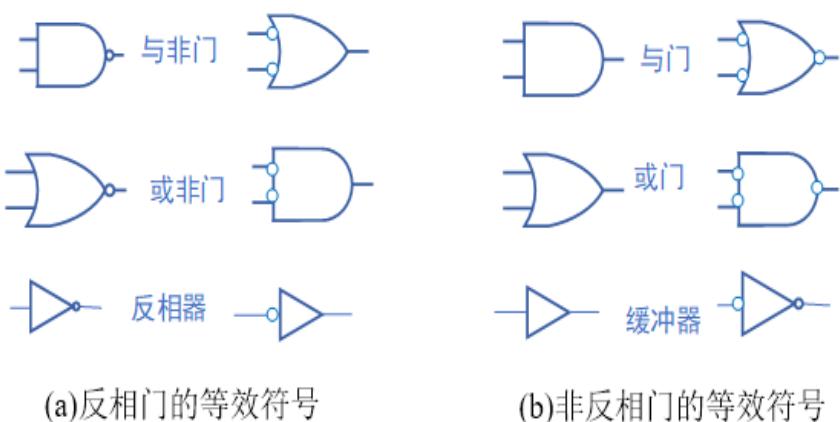
- 逻辑门是最基础的数字电路，可通过CMOS晶体管实现
  - 门符号、逻辑运算符、真值表、逻辑表达式、运算优先级。。
  - 与、或、非、与非、或非、异或
- ◆ CMOS晶体管
  - PMOS和NMOS
  - 常用CMOS门电路
  - 非、与非、或非
  - 与(与非-非)、或
  - 缓冲器、传输门



与	
或	
非	
与非	
或非	
异或	

# 数字逻辑基础

- 最基本的逻辑运算有与、或、非三种运算，对应的逻辑门分别为与门、或门和非门
- 布尔代数、公理系统、对偶定律等基本概念。
- 通常使用真值表、逻辑表达式来描述逻辑变量间的关系
- 可使用代数法、卡诺图等来化简逻辑表达式（参考作业题）
- 在实现数字系统时，为了提高速度、降低成本，通常利用与非门和或非门来构建电路。
- 等效逻辑符号（电路）——
- （德摩根定理的利用）



# 组合逻辑电路

- 数字逻辑电路由若干元件（可以是一个电路）和若干结点互连而成
- 组合逻辑电路的输出值仅依赖于当前输入值
- 组合逻辑电路可以是两级电路或多级电路，两级电路的传输时间短，但占用集成电路物理空间更多，需进行时空权衡
- 组合逻辑电路设计：功能分析-列表-化简-逻辑表达式-画图-评价（参考作业题）
- 无关项指输出取值可任意的项，真值表中用d表示，可用于化简
- 非法值指同时被高、低电平驱动的输出结点的值。
- 高阻态是三态门输出结点的一种非正常逻辑态，相当于“断开”
- 典型组合逻辑部件：译码/编码器、多路选择/分配器、半加/全加器
- 传输延迟：关键路径上所有元件的传输延迟之和
- 最小延迟：最短路径上所有元件的最小延迟之和

# 时序逻辑基本元件

## ◆ SR锁存器

- ✓ 置位端(S)/复位端(R)； 用于设置标志信息

## ◆ D锁存器

- ✓ 控制端C有效时， 锁存数据D

## ◆ D触发器

- ✓ 时钟触发边沿开始后， 经过Clk-Q时间， Q变成D； 输入端D在时钟触发边沿到来前， 须稳定Setup时间； 之后须继续保持hold时间
- ✓ 可带EN控制端、 置位/清零控制端

## ◆ T触发器

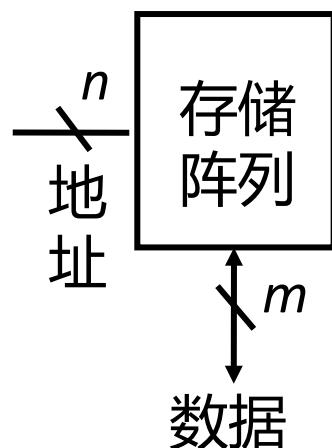
- ✓ 由D触发器构成， T连接Clk， D连接Q， 每个时钟发生状态变化

# 时序逻辑电路

- 时序逻辑电路不仅依赖当前输入，还依赖电路当前的状态
- 可用状态图或状态表描述有限状态机，圈表示状态，有向边表示输入/输出
- 时序电路设计：功能分析-状态图-状态化简和编码-逻辑表达式-画图-评价  
(参考作业题)
- 未用状态分析(挂起/无法自启动)
- 定时分析(clk-Q时间、时钟周期、setup时间、hold时间)——有助于理解后续的CPU设计
- 典型时序逻辑部件：计数器、寄存器/通用寄存器组、移位寄存器

# 存储器的结构和基本概念

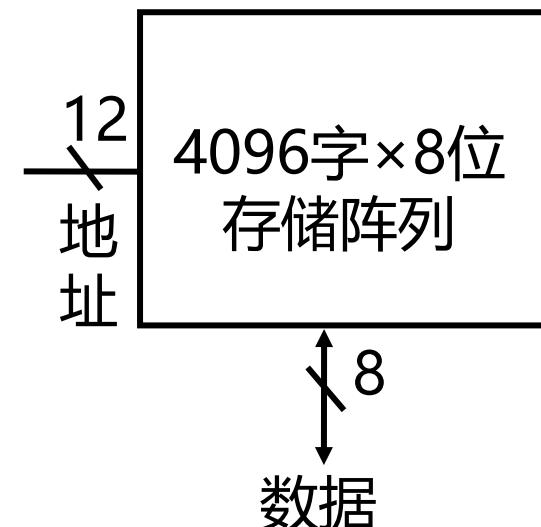
- 存储器可用来存储数字电路中的数据。
  - 寄存器（由触发器构成）用来**存储少量数据**，速度快
  - 存储器阵列用来**存储大量数据**，速度比寄存器慢
- 存储器阵列中每位数据对应一个**记忆单元**（cell），称为**存储元**



$2^n \times m$ 位  
存储阵列

000	0	1	0	0
001	0	0	1	1
010	0	1	0	1
011	1	0	1	1
100	1	1	0	0
101	1	0	1	1
110	0	1	1	0
111	1	0	0	0

8×4位存储器阵列



4KB存储器阵列

# ROM, RAM

- 按功能可分为：只读存储器(Read-only Memory, ROM)和随机存取存储器(Random-access Memory, RAM)
  - ROM属于非易失性存储器，即使电源断电，ROM中存储的数据也不会消失
    - 例如存放BIOS自检启动程序的地方
    - 例如微程序控制器中的CS (控制存储器)
  - RAM属于易失性存储器，一旦电源断电，RAM中存储的数据就消失。
    - 静态RAM (Static RAM, SRAM)
    - 动态RAM (Dynamic RAM, DRAM)

# 运算部件的设计

- ALU的实现
  - 加法器是基础（全加器）
  - 加法器需要进行标志位的计算（有符号数和无符号数都共用同一套标志位计算方法，但标志的用法不一样）
  - 算术逻辑单元ALU：实现基本的加减运算和逻辑运算。
  - 加法运算是所有定点和浮点运算（加/减/乘/除）的基础，  
加法速度至关重要 <sup>△原理</sup>
  - 进位方式是影响加法速度的重要因素
  - 并行进位方式能加快加法速度

# 运算部件的设计

逻辑运算、移位运算、扩展运算等电路简单 (如算术移位、逻辑移位、0扩展、符号扩展等)

## 主要考虑算术运算

- 定点运算涉及的对象  
无符号数；带符号整数(补码)；  
原码小数；移码整数
- 加减运算
  - 补码加/减运算：用于整数加/减运算。符号位和数值位一起运算，减法用加法实现。可判断溢出。
  - 无符号数加/减运算：加法直接加；减法用加负数补码实现。可判断溢出。

# 运算部件的设计

乘法运算：（关注思路，参考作业题）

无符号数乘法：“判断”，“加”，“右移”

原码乘法：符号和数值分开运算，数值部分用无符号数乘法实现，用于浮点数尾数乘法运算。

补码乘法：符号和数值一起运算，采用Booth算法。

- $n$ 位  $\times n$ 位，结果机器数可获得高 $n$ 位和低 $n$ 位。
- 高 $n$ 位可用来判断溢出，也可直接作为乘积的高位（肯定不溢出）。

除法运算：（不考）

无符号数除法：用“加/减” + “左移”，有恢复余数和不恢复余数两种。

原码除法：符号和数值分开，数值部分用无符号数除法实现，用于浮点数尾数除法运算。

补码除法：符号位和数值位一起。

# 指令系统

- ◆ 操作类型
  - 传送 / 算术 / 逻辑 / 移位 / 字符串 / 转移控制 / 调用 / 中断
- ◆ 操作数类型
  - 整数（带符号、无符号、十进制）、浮点数、位、位串
- ◆ 地址码的编码要考虑：
  - 操作数的个数
    - 寻址方式：立即 / 寄存器 / 寄间 / 直接 / 间接 / 偏移 / 堆栈
- ◆ 操作码的编码要考虑：
  - 定长操作码 / 扩展操作码
- ◆ 条件码的生成：四种基本标志：NF (SF) / VF (OF) / CF / ZF
- ◆ 如何控制“周而复始的执行指令”呢？
  - 隐式的自动按顺序取
  - 显式的在指令中给出“下条指令地址”
  - 条件测试后计算出“转移目标地址”
- ◆ 指令设计风格：
  - » 复杂指令集计算机CISC、精简指令集计算机RISC

# 指令系统

## ◆ 指令格式

- 定长指令字：所有指令长度一致
- 变长指令字：指令长度有长有短

## ◆ RV32I

	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		

# 具体指令和RTL

**lui rd, imm20**

**add/sub/or... x10, x12, x14**

**slt x11, x10, x12**

**addi/subi x5, x5,-1**

**ori/andi rd, rs1, imm12**

**slti rd, rs1, imm12**

**jal rd, ,imm20**

**bne/beq x28, x29, imm12**

**lw rd, imm12(rs1),**

**sw rs2, imm12(rs1)**

## RTL规定:

**R[r]: 通用寄存器r的内容**

**M[addr]: 存储单元addr的内容**

**M[R[r]]: 寄存器r的内容所指存储单元的内容**

**PC: PC的内容**

**M[PC]: PC所指存储单元的内容**

**SEXT[imm]: 对imm进行符号扩展**

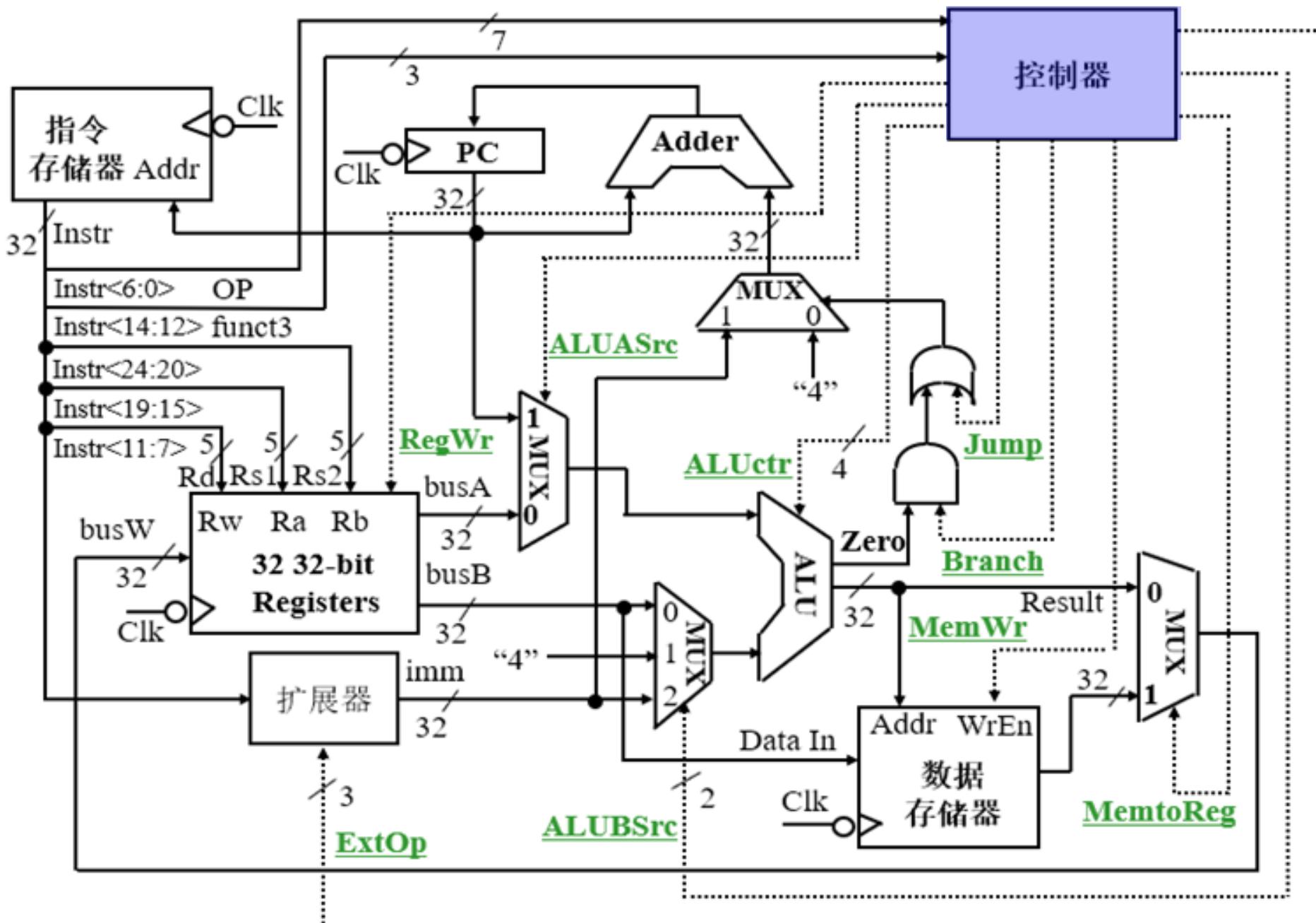
**ZEXT[imm]: 对imm进行零扩展**

**传送方向用←表示, 即传送源在右, 传送目的在左**

# CPU基础

- CPU设计直接决定了时钟周期宽度和CPI，所以对计算机性能非常重要！
- CPU主要由数据通路和控制器组成
  - 数据通路：实现指令集中所有指令的操作功能
  - 控制器：控制数据通路中各部件进行正确操作
- 数据通路中包含两种元件
  - 操作元件（组合电路）：ALU、MUX、扩展器、Adder、Reg/Mem Read等
  - 状态 / 存储元件（时序电路）：PC、Reg/Mem Write
- 数据通路的定时
  - 数据通路中的操作元件没有存储功能，其操作结果必须写到存储元件中
  - 在时钟到达后clk-to-Q时存储元件开始更新状态
- RV32I指令集的一个子集作为CPU的实现目标

## 单周期处理器



# 单周期CPU小结

- 每条指令在一个时钟周期内完成
- 每个时钟到来时，都开始进入取指令操作
  - 经过clk-to-Q，PC得到新值，经过access time后得到当前指令
  - 按三种方式分别计算下条指令地址，在branch / zero / jump的控制下，选择其中之一送到PC输入端，但不会马上写到PC中，一直到下个时钟到达时，才会更新PC。三种下址方式为：
    - branch=jump=0: PC+4
    - branch=zero=1: PC+SEXT(imm12)\*2
    - jump=1: PC+SEXT(imm20)\*2
  - 同时按控制信号完成当前指令所特有的操作
- 汇总每条指令控制信号的取值，生成真值表，写出逻辑表达式，设计控制器逻辑——控制单元对指令进行译码，与指令执行得到的条件码或当前机器的状态、时序信号（时钟）等组合，生成对数据通路进行控制的控制信号。

# 单周期和多周期CPU对比

- 单周期处理器的设计
  - 每条指令都在一个时钟周期内完成
  - 时钟周期以最长的Load指令所花时间为准
  - 无需加临时寄存器存放指令执行的中间结果
  - 同一个功能部件不能重复使用
  - 控制信号在整个指令执行过程中不变，所以控制器设计简单，只要写出指令和控制信号之间的真值表，就可以设计出控制器
- 多周期处理器的设计
  - 每条指令分成多个阶段，每个阶段在一个时钟内完成
  - 时钟周期以最长的阶段所花时间为准
  - 不同指令包含的时钟个数不同
  - 阶段的划分要均衡，每个阶段只能完成一个独立、简单的功能
  - 需加临时寄存器存放指令执行的中间结果
  - 同一个功能部件能在不同的时钟中被重复使用
  - 可用有限状态机表示指令执行流程，并用PLA或微程序方式设计控制器

# 流水线CPU

- **流水线CPU的设计**
  - 将每条指令的执行规整化为若干个同样的流水阶段
  - 每个流水阶段的执行时间一样，都等于一个时钟
  - 理想情况下，每个时钟有一条指令进入流水线，也有一条指令执行结束  
(指令吞吐率增大、**CPI约等于1**、单条指令实际所需时间增大)
  - 每两个相邻流水段之间的**流水段寄存器**，用以记录所有在后面阶段要用到的各种信息 (包括指令译码得到的控制信号)
- **结构冒险 (资源冲突) 及其解决方法**
- **数据冒险 (数据相关) : 前面指令的结果是后面指令的操作数**
  - 软件阻塞 (加nop指令) 、硬件阻塞 (插入 “气泡” )
  - 寄存器前半周期写后半周期读、编译优化、 “转发” (旁路)
  - 对于load-use，采用 “阻塞加转发” 的方式解决数据冒险
- **控制冒险 (控制相关) : 目标指令地址产生前已经有指令被取到流水线中，如果这些指令不该被执行，则发生控制冒险。**
  - 软件阻塞、硬件阻塞、延迟分支技术
  - 采用 “分支预测” 技术：静态预测或动态预测
  - 异常和中断也是一种特殊的控制冒险



# 顾 杈 南京大学 计算机学院 副教授(特聘研究员, 博导)

## ◆ 教育部青年长江学者、达摩院青橙奖获奖者(2023)、Fluid开源社区主席

研究目标: 面向大数据与AI应用的云原生系统基座

### 研究领域背景

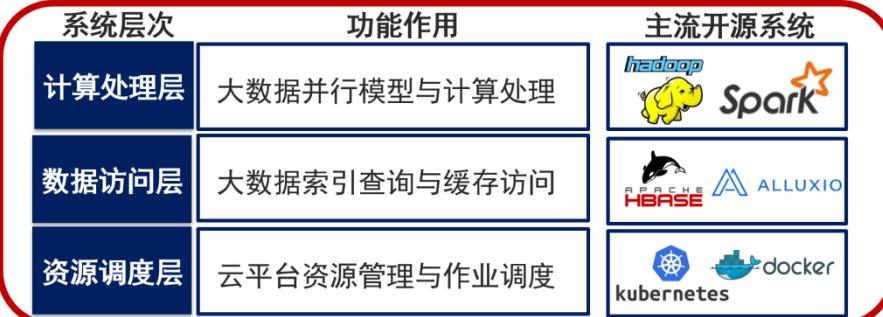
智能化行业  
应用服务



智能化分析  
挖掘算法



大数据与云计算  
系统平台



### 重点研究问题

- AI模型推理服务性能优化
- 大模型在云平台训练加速
- 云原生平台资源管理调度
- 数据库系统查询性能优化
- ...

基础性计算机课程知识:

计算机组成原理、数字逻辑电路、  
操作系统、数据结构、计算机网络、  
大数据系统等

### 学术论文发表



EURO/SYS



SIGMOD



THE TRANSACTIONS ON  
PARALLEL AND  
DISTRIBUTED SYSTEMS



The VLDB Journal  
the International  
Journal of Data  
and Knowledge  
Engineering

### 行业落地应用

