第五章 中央处理器

2022年9月21日 星期三　10:56

CPU的功能和基本结构

CPU的功能	• 指令控制 　• 完成取指令、分析指令和执行指令的操作，即程序的顺序控制。 • 操作控制 　• 管理并产生由内存取出的每条指令的操作信号 　• 把各种操作信号送往相应的部件，从而控制这些部件按指令要求精心的动作。 • 时间控制 　• 对各种操作进行时间上的可控制，要为每一条指令按时间顺序提供应有的控制信号。 • 数据加工 　• 对数据进行算术和逻辑运算。 • 中断处理 　• 对计算机运行过程中出现的异常情况和特殊请求进行处理。
CPU的组成	• CPU = 运算器 + 控制器
运算器	功能 • 接收从控制器送来的命令并执行相应的动作，对数据进行加工和处理 • 运算器是计算机对数据进行加工处理的中心 组成 ALU算术逻辑单元：• 进行算术/逻辑运算 暂存寄存器：• 暂存从主存读来的数据，对应用程序员透明 ACC累加寄存器：• 是一个通用寄存器，暂放ALU运算的结果信息，可作为加法运算的输入端 通用寄存器组：• 如AX，BX，CX，DX，SP • 用于存放操作数和各种地址信息 • SP是堆栈指针，用于指示栈顶的地址 PSW程序状态字寄存器：• 保留由算术逻辑运算指令或测试指令的结果而建立的各种状态信息 • 如溢出标志OF，符号标志SF，零标志ZF，进位标志CF • PSW中的这些位参与并决定微操作的形成 移位器：• 对操作数或运算结果进行移位运算 计数器：• 控制乘除运算的操作步数
控制器	功能 • 负责协调并控制计算机各部件执行程序的指令序列 • 执行指令，每条指令的执行是由控制器发出的一组微操作实现的 工作原理 • 根据指令操作码，指令的执行步骤（微命令序列）和条件信息来形成当前计算机各部件要用到的控制信号 • 计算机各硬件系统在这些控制信号的控制写协同运行，产生预期的执行结果 种类 • 硬步线控制器 • 微操作控制器 组成 PC程序计数器：• 用于指出下一条指令在主存中的存放地址 • CPU根据PC的内容去主存中取指令 • PC有自增功能 IR指令寄存器：• 用于保存当前正在执行的那条指令 指令译码器：• 仅对操作码字段进行译码，向控制器提供特定的操作信号 MDR存储器地址寄存器：• 用于存放要访问的主存单元的地址 MAR存储器数据寄存器：• 用于存放向主存写入的信息或从主存读出的信息 时序系统：• 用于产生各种时序信号，都由统一时钟CLOCK分频得到 微操作信号发生器：• 根据IR的内容(指令)，PSW的内容(状态信息)和时序信号，产生控制计算机系统所需的各种控制信号 • 结构有组合逻辑型和存储逻辑型
CPU寄存器的分类	用户可见的寄存器 • 可对这类寄存器编程 • 通用寄存器，PSW 用户不可见的寄存器 • 不可对这类寄存器编程 • MAR，MDR，IR

控制器的功能和工作原理

控制器的结构和功能

结构

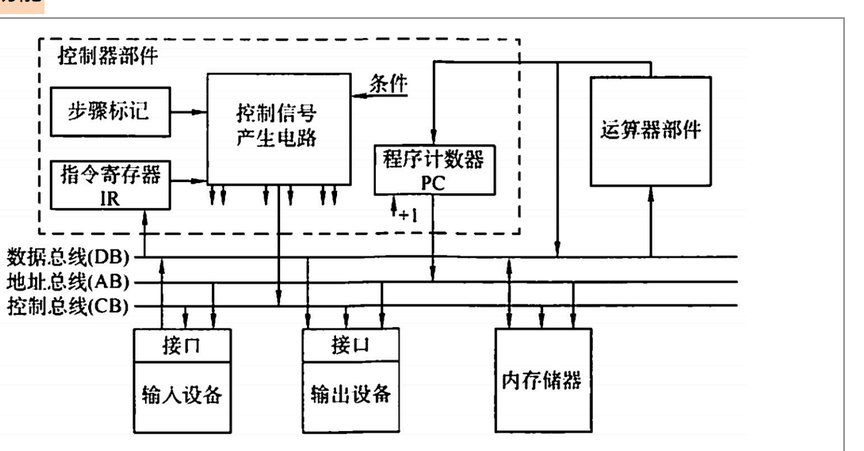


图 5.8　计算机硬件系统和控制器部件的组成

连接关系	• 运算器部件通过数据总线与内存储器、输入设备和输出设备传送数据 • 输入设备和输出设备通过接口电路与总线相连接 • 内存储器、输入设备和输出设备从地址总线接收地址信息，从控制总线得到控制信号，通过数据总线与其他部件传送数据 • 控制器部件从数据总线接受指令信息，从运算器部件接受指令转移地址送出指令地址到地址总线，还要向系统中的部件提供它们运行所需要的控制信号
功能	• 从主存中取出一条命令，并指出下一条指令在主存中的位置 • 对指令进行编码或测试，产生相应的操作控制信号，以便启动规定的动作 • 指挥并控制CPU、主存、输入和输出设备之间的数据流动方向

根据控制器产生微操作信号的方式的不同，控制器可以分为硬布线控制器和微程序控制器

- 2022年之后考的权重下降！，可以不用看设计啥的，就掌握个基本原理和概念吧
- 设计太难了，考大题直接G算了┬┬﹏┬┬

硬布线和微程序控制器的对比

	微程序控制器	硬布线控制器
工作原理	微操作控制信号以微操作的形式存放在控制存储器中，执行指令时读出即可	微操作控制信号由组合逻辑电路根据当前的指令码，状态和时序，即时产生
执行速度	慢	快
规整性	较规则	封锁，不规则
应用场合	CISC CPU	RISC CPU
易扩展性	易扩充修改	困难

硬布线控制器

基本原理	• 根据指令的要求、当前的时序及外部和内部的状态，按时间顺序发送一系列微操作控制信号 • 是由复杂的组合逻辑门电路和一些触发器构成，又称组合逻辑控制器
硬布线控制单元图	CU的输入信号来源 • 经指令编译器译码产生的指令信息 • 时序系统产生的机器周期信号和节拍信号 • 来及执行单元的反馈信号，即标志
硬布线控制器的时序系统及微操作	• 时钟周期：用时钟信号控制节拍发生器，可以产生节拍，每个节拍的宽度正好对应一个时钟周期 • 机器周期：可以视为所有指令执行过程中的一个基准时间 • 指令周期：CPU从主存中取出并执行一条指令的时间称为指令周期 • 微操作命令分析：控制单元发出各种操作命令序列的功能
CPU的控制方式	同步控制方式：• 系统有一个统一的时钟，所有的控制信号都来源于这个统一的时钟信号 • 通常以最长的微操作序列和最繁琐的微操作作为标准 • 采取完全统一的、具有相同时间间隔和相同数目的节拍作为机器周期来运行不同的指令 • 优点：控制电路简单 • 缺点：运行速度慢 异步控制方式：• 不存在基准时标信号，各部件按照自身固有速度工作，通过应答方式进行联络 • 优点：运行速度快 • 缺点：控制电路较为复杂 联合控制方式：• 介于同步、异步之间的一种折中 • 对大部分采用同步控制，小部分采用异步控制
设计步骤(放弃😭)	• 列出微操作命令的操作时间表 • 进行微操作信号综合 • 画出微操作命令的逻辑图

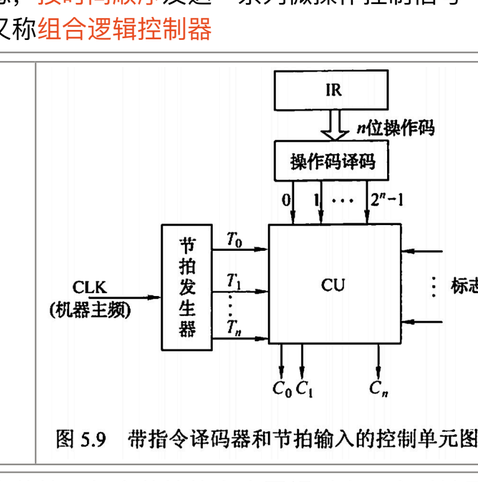


图 5.9　带指令译码器和节拍输入的控制单元图

微程序控制器

基本原理	• 采用存储逻辑实现，即把微操作信号代码化 • 使每条机器指令转化为一段微程序并存入一个专门的存储器（控制存储器）中 • 微操作信号由微指令产生，用寻址用户及其指令的办法来寻址每个微程序中的微指令
基本概念	• 每条及其指令编写成一个微程序，每个微程序包含若干微指令，每条微指令对应一个或几个微操作命令 微命令与微操作：• 一条机器指令可以分解成一个微操作序列 • 微操作是计算机中最基本、不可再分解的操作 • 将控制部件向执行部件发出的各种控制命令称为微命令，是构成控制序列的最小单位。 微指令和微周期：• 微指令是若干微命令的集合 • 存放微指令的控制存储器的单元地址称为微地址 • 微周期指的是从控制存储器中读取一条微指令并执行相应的微操作所需的时间 • 微指令的两个组成部分：微操作码字段和微地址码字段 主存储器与控制存储器：• 主存用于存放程序和数据，在CPU外部，用RAM实现； • CM用于存放微程序，在CPU内部，用ROM实现。 程序与微程序：• 程序是指令的有序集合，用于完成特定的功能； • 微程序是微指令的有序集合，一条指令的功能由一段微程序实现。 微程序和机器指令：• 一般来说，一条机器指令对应一个微程序 • 若指令系统中有n种机器指令，则控制存储器中的微程序数至少是n+1个。 • 寄存器区分表 地址寄存器：存放主存的读写地址 微地址寄存器：存放控制存储器的读写指令的地址 指令寄存器：存放从主存中读出的指令 微指令寄存器：存放从控制存储器中读出的微指令
基本组成	控制存储器：核心部件，用于存放各指令对应的微程序，控制存储器可用ROM构成 微指令寄存器：存放从CM中取出的微指令，位数与微指令字长相等 微地址形成部件：用于产生初始微地址和后继微地址，以保证微指令的连续执行 微地址寄存器：接受微地址形成部件送来的微地址，为在CM中读取微指令做准备

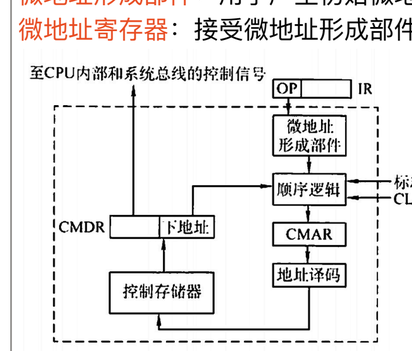


图 5.11　微程序控制器的基本结构

工作过程	1. 执行取微指令公共操作 2. 由及其指令的操作码字段通过微地址形成部件产生该及其指令所对应的微程序的入口地址，并送入CMAR 3. 从CM中逐条取出对应的微指令并执行 4. 执行完对应于一条机器指令的一个微程序后，又回到取指微程序的入口地址，继续第一步
编码方式/控制方式	目的：保证速度的情况下尽量缩短指令字长 直接编码方式：• 无需进行译码，微指令的微命令字段中的每一位都代表一个微命令（选用“1”，不选用“0”） • 优点：简单、直观、执行速度快，操作并行性好 • 缺点：微指令字长过长，n个微指令就要求微指令的操作字段有n位，造成控制存储器容量极大 字段直接编码法：• 将微命令字段分成若干小字段，将互斥性微命令组合在同一字段中 • 把相容性微命令组合在不同字段中，每个字段独立编码 • 每种编码都代表一个微命令且各字段编码含义单独定义，与其他字段无关。 • 优点：可以缩短微指令字长 • 缺点：要通过译码电路之后再发出微命令，速度慢 • 分段原则 ○ 互斥性微命令组合在同一字段中，把相容性微命令组合在不同字段中 ○ 每个小段中包含的信息位不能太多，否则将增加译码电路的复杂性和译码时间 ○ 一般每个小段还要留出一个状态，表示本字段不发出任何微命令 字段间接编码法：• 一个字段的某些微命令需由另一字段中的某些微命令解释 • 由于不是靠字段直接译码发出的微命令，因此称为字段间接编码 • 优点：可进一步缩短微指令字长 • 缺点：削弱了微指令的并行能力
地址形成方式	后继微地址的形成类型 • 直接由微指令的下地址字段指出。格式中设置一个下地址（断定方式）。 • 根据机器指令的操作码形成。机器指令取至IR后，微指令的地址由操作码经微地址形成部件形成。 • 增量计数器法（微地址连续） • 根据各种标志决定微指令分支转移的地址 • 通过测试网络形成 • 由硬件直接产生微程序入口地址
微指令的格式	• 水平型微指令 • 垂直型微指令 • 混合型微指令 • 水平型和垂直型对比 　○ 水平型的并行操作能力强，效率高，灵活性强；垂直型的则较差 　○ 水平型的执行一条指令的时间短；垂直型的长 　○ 水平型对应的微程序，具有微指令字较长但微程序短的特点；垂直型则相反 　○ 水平型微指令用户难以掌握，而垂直型指令与指令较为相似，相对容易掌握
设计步骤(放弃😭)	• 写出对应机器指令的微操作命令及节拍安排 • 确定微指令格式 • 编写微指令码点
动态微程序设计 毫微程序设计	• 动态微程序设计 　• 能根据用户的要求改变微程序，则其具有动态微程序设计功能。 　• 需要可写控制寄存器的支持。可采用EPROM。 • 毫微程序设计 　• 硬件不由微程序直接控制，而是通过存放在第二级控制存储器中的毫微程序来解释的 　• 这个第二级控制存储器就成为毫微存储器，直接控制硬件的就是毫微微指令

指令的执行过程

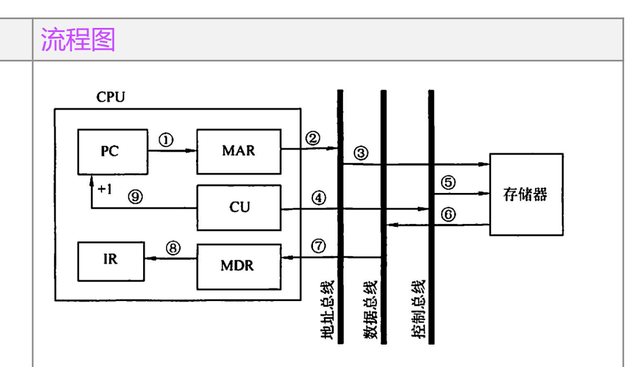
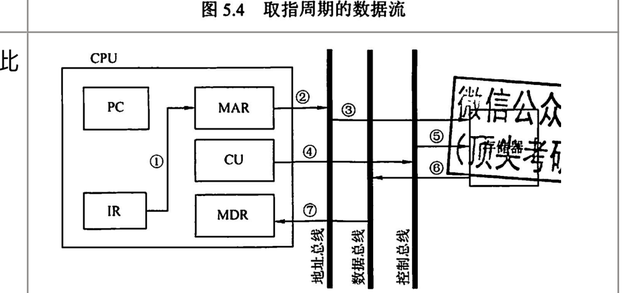
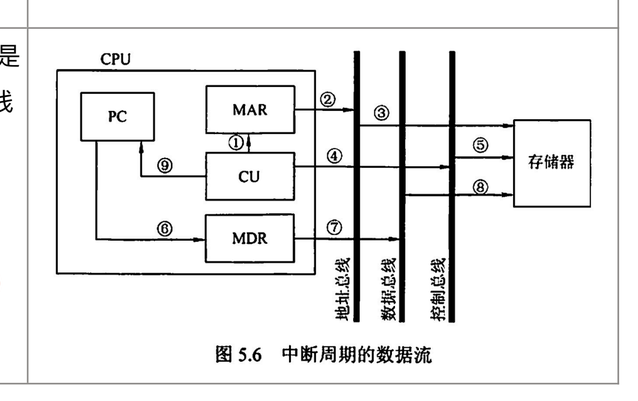
指令周期

- CPU从主存中取出并执行一条指令的时间称为指令周期
- 指令周期最多有4个工作周期：取指周期、间址周期、执行周期、中断周期（都有CPU访存操作）
- 分别对应标志触发器：FE、IND、EX、INT（“1”表示有效，如1-->FE表示有取值周期）
- 计算机工作的最小时间周期是时钟周期

指令类型	指令周期的组成
无条件转移指令JMP X	• 不需要访问主存 • 指令周期 = 取指周期和执行周期
间接寻址的指令	• 为了取操作数，需要先访问一次主存，取出有效地址--->取指周期 • 然后访问主存，取出操作数--->间址周期 • 指令周期 = 取指周期、间址周期和执行周期
当CPU采用中断方式实现主机和I/O设备的信息交换且有中断请求	• CPU在每条指令执行结束前，都要发中断查询信号 • 若有中断请求，则CPU进入中断响应阶段----->中断周期 • 指令周期 = 取指周期、间址周期、执行周期、中断周期 • 📢中断周期进栈操作是将SP-1，计算机的堆栈都是向低地址增加，所有进栈操作减1而不是加1

指令周期的数据流

- 数据流：根据指令要求一次访问的数据序列
- 指令执行不同阶段，访问的数据序列不同
- 不同的指令，数据流也不同

	任务	数据流向	流程图
取值周期	根据PC中的内容从主存中取出指令代码并放在IR中 【取指】	PC中存放的是指令的地址，根据此地址从内存单元中取出的指令，并放在指令寄存器IR中，取指同时，PC+1 1. PC--->MAR--->地址总线--->主存 2. CU(控制单元)发出读命令--->控制总线--->主存 3. 主存--->数据总线MDR--->IR（存放指令） 4. CU发出控制信号--->PC内容加1	 图 5.4　取指周期的数据流
间址周期	取操作数的有效地址 【取有效地址】	以一次间址为例，将指令中的地址码送到MAR并送至地址总线，此后CU向存储器发读命令，以获取有效地址并存至MDR 1. Ad（IR）（或MDR）--->MAR--->地址总线--->主存 2. CU发出读命令--->控制总线--->内存 3. 主存--->数据总线--->MDR（存放有限地址） 4. Ad（IR）表示取出IR中存放的指令字的地址字段。	 图 5.5　一次间址周期的数据流
执行周期	取操作数，并根据IR中的指令字的操作码通过ALU操作产生执行结果。 【取操作数】	无统一的数据流向	
中断周期	处理中断请求 【保存程序断点】	假设程序断点存入堆栈中，并用SP指示栈顶地址，而且进栈操作是先修改栈顶指针，后存入数据；出栈操作是先删除数据，后修改栈顶指针。 1. CU控制将SP减1，SP--->MAR--->地址总线--->主存 2. CU发出写命令--->控制总线--->主存 3. PC--->MDR--->数据总线--->主存（程序断点存入主存） 4. CU（中断服务程序的入口）--->PC	 图 5.6　中断周期的数据流

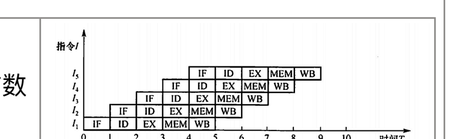
指令的执行方案

- 如何安排指令的执行步骤称为指令执行方案

单指令周期	• 对所有指令都选用相同的执行时间来完成 • 每条指令都在固定的时钟周期内完成，指令之间串行执行 • 指令周期取决于执行时间最长的指令的执行时间
多指令周期	• 对不同类型的指令选用不同的执行步骤 • 指令之间串行执行，但可以选用不同个数的时钟周期来完成不同指令的执行过程 • 指令需要几个周期就为其分配几个周期，不再要求所有指令占用相同的执行时间
流水线方案	• 指令之间可以并行执行的方案 • 目标：力争在每个时间脉冲周期完成一条指令的执行过程（理想情况） • 通过在每个时钟周期启动一条指令，尽量让多条指令同时运行，但各自处在不同的执行步骤中

指令流水线

流水线的基本概念

如何提高处理机的并行性?	1.时间上的并行技术 • 将一个任务拆分成几个不同的子阶段 • 每个阶段在不同的功能部件上并行执行，即流水线技术 2.空间上的并行技术 • 在一个处理机内设置多个执行相同任务的功能部件 • 并让这些功能部件并行工作，这样的处理机称为超标量处理机
指令流水定义	• 将指令执行过程的各阶段视为相应的流水段，则指令的执行过程就构成了一条指令流水
指令流水举例	• 假设一条指令的执行有5条流水段 取指（IF）：从IR或者Cache中取指令 译码/读寄存器（ID）：操作控制器对指令进行译码，同时从寄存器堆中取操作数 执行/计算地址（EX）：执行运算操作或计算地址 访存（MEN）：对存储器进行读写操作 写回（WB）：将指令执行结果写回寄存器堆  图 5.16　一个 5 段指令流水线 • 设用时最长的流水段用时x秒，总执行时间为y秒，系统由N条指令 • 则单处理机用时为X × N，流水线处理机用时为（N+4）× y • 不能缩短单条指令的执行时间，但对于整个程序来说，执行效率得到大幅提升
指令集的特点	• 指令长度尽量一致，有利于简化取指和指令译码操作 • 指令格式尽量规整，尽量保证源寄存器的位置相同，有利于在指令未知时就可取存寄存器操作数 • 采用Load/Store指令，其他指令无法访问存储器 • 这样可以把Load/Store指令的地址计算和运算指令的执行步骤规整到同一个周期中，有利于减少操作步骤 • 数据和指令在存储器中“对齐”存放。这样有利于减少访存次数使所需数据在一个流水段内就可以从存储器中找到
流水线的表示方法	• 采用时空图描述流水线的执行情况  图 5.17　一个 5 段指令流水线时空图

流水线的基本实现

流水线的数据通路

- 一个5段流水线数据通路，实线表示数据信号，虚线表示控制信号

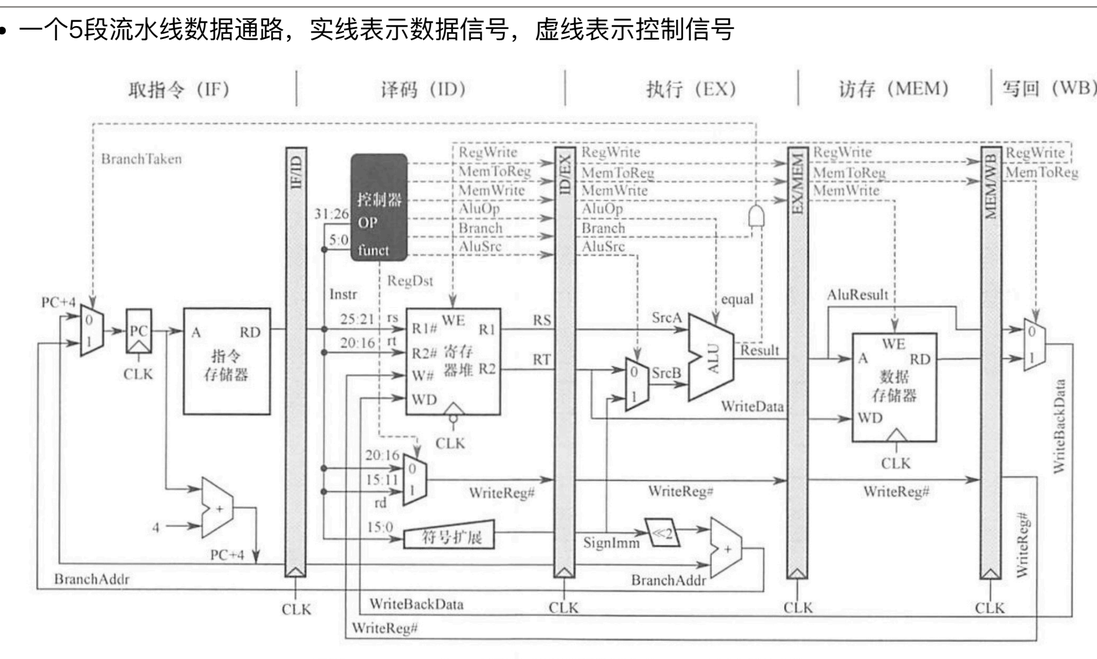


图 5.19　一个 5 段流水线数据通路（及控制信号）

流水线的控制信号

表 5.3　图 5.19 中的控制信号分类

控制信号	位置	来源	功能说明
BranchTaken	IF	EX	分支跳转信号，为 1 表示跳转，由 EX 段的 Branch 信号与 equal 标志进行逻辑与生成
RegDst	ID	ID	写入目的寄存器选择，为 1 时目的寄存器为 rd 寄存器，为 0 时为 rt 寄存器
RegWrite	ID	WB	控制寄存器堆写操作，为 1 时数据需要写回寄存器堆中的指定寄存器
AluSre	EX	EX	ALU 的第二输入选择控制，为 0 时输入寄存器 rt，为 1 时输入扩展后的立即数
AluOp	EX	EX	控制 ALU 进行不同运算，具体取值和位宽与 ALU 的设计有关
MemWrite	MEM	MEM	控制数据存储器写操作，为 0 时进行读操作，为 1 时进行写操作
MemToReg	WB	WB	为 1 时将数据存储器读出数据写回寄存器堆，否则将 ALU 运算结果写回

流水寄存器保存的信息

- 后面流水段要用到的所有数据信息
  - 包括PC+4,、指令、立即数、目的寄存器、ALU运算结果、标志信息等
  - 它们是前面阶段在数据通路中执行的结果
- 后面传递过来的后面各流水段要用到的所有控制信号

流水线的执行过程

取指（IF）	从IR或者Cache中取指令
译码/读寄存器（ID）	操作控制器对指令进行译码，同时从寄存器堆中取操作数
执行/计算地址（EX）	执行运算操作或计算地址
访存（MEN）	对存储器进行读写操作
写回（WB）	将指令执行结果写回寄存器堆

流水线的冒险与处理

- 流水线冒险：在指令流水线中，可能遇到一些情况使得流水线无法正确执行后续指令而引起流水线阻塞或者停顿

	描述	解决方法
结构冒险	• 由于多条指令在同一时刻争用同一资源而形成的冲突	1. 前一指令访存时，使后一条指令（以及其后续指令）暂停一个时钟周期 2. 单独设置数据存储器和指令存储器，使取数和取指令操作各自在不同的存储器中进行（分离结构的Cache）
数据冒险	• 下一条指令会用到当前指令计算出的结果 • 此时这两条指令发生数据冲突 写后读相关：• 当前指令将数据写入寄存器后 • 下一条指令才能从该寄存器读取数据 读后写相关：• 当前指令读出数据后 • 下一条指令才能写入寄存器 写后写相关：• 当前指令写入寄存器后 • 下一条指令才能写入寄存器	1. 把遇到数据相关的指令及其后续指令都暂停一至几个时钟周期知道数据相关问题消失后再继续执行，可分为硬件阻塞和软件插入“NOP”指令两种方法 2. 设置相关专用通路，即不等前一条指令把计算结果写回寄存器组下一条指令也不再会读寄存器组，而直接把前一条指令的ALU计算结果作为自己的输入数据开始计算过程，使本来需要暂停的操作变得可以继续执行，称为数据旁路技术 3. 通过编译器对数据相关的指令编译优化的方法，调整指令顺序来解决数据相关
控制冒险	• 指令通常都是顺序执行的 • 但在遇到改变指令执行顺序的情况就会改变PC值 • 会造成断流，从而引起控制冒险	1. 对转移指令进行分支预测，尽早生成转移目标地址 分为到简单（静态）预测和动态预测 前者总是预测条件不满足，即继续执行分支指令的后续指令 后者根据程序执行的历史情况，进行动态调整，有较高的预测准确率 2. 预取转移成功和不成功两个控制流方向上的目标指令。 3. 加快和提前形成条件码。 4. 提高转移方向的猜准率。

流水线的性能指标

流水线的吞吐率	单位时间内流水线所完成的任务数量，或输出结果的数量
流水线的加速比	完成同样一批任务，不使用流水线和使用流水线所用时间之比

高级流水线技术

- 有以下增加指令级并行的策略

超标量流水线技术	• 动态多发射技术 • 每个时钟周期内可并发多条独立指令，以并行的方式将两条或多条指令编译并执行，为此需要配置多个功能部件
超长指令字技术	• 静态多发射技术 • 由编译程序挖掘出指令减潜在的并行性 • 将多条能并行操作的指令组合成一条具有多个操作码字段的超长指令字，为此需要采用多个处理部件
超流水线技术	• 提高流水线的主频来提升流水线性能 • 但流水线级数（之前划分的5段，还能划分得更细）越多，用于流水寄存器上的开销就大。

数据通路的功能和基本结构

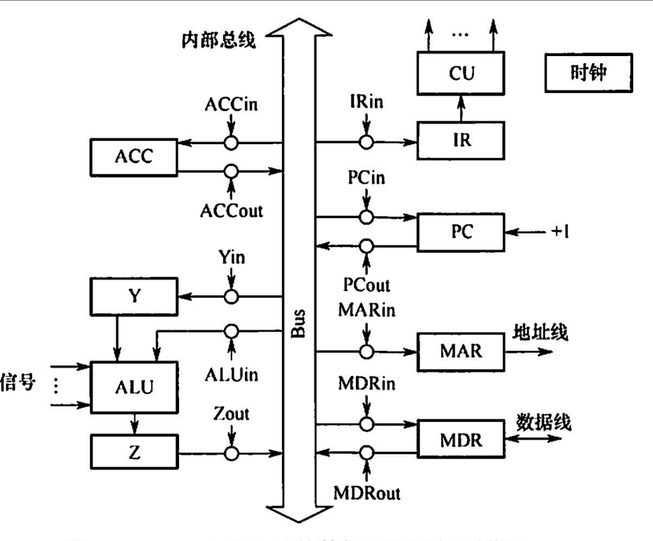
数据通路

- 数据通路：数据在功能部件之间传送的路径，包括数据通路上流经的部件。
- 它描述了信息从什么地方开始，中间经过哪个寄存器或多路开关，最后传送到哪个寄存器
- 数据通路是由控制部件控制，控制部件根据每条指令功能的不同生成对数据通路的控制信号

数据通路的功能

- 实现CPU内部的运算器与寄存器及寄存器之间的数据交换

数据通路的基本结构

CPU内部单总线方式	【定义】将所有寄存器的输入端与输出端都连接到一条公共通路上 【特点】结构比较简单，数据传输存在较多的冲突现象，性能较低
CPU内部三总线方式	【定义】将所有寄存器的输入端与输出端都连接到多条公共通路上 【特点】相较单总线结构，效率提高
专用数据通路方式	【定义】根据指令执行过程中的数据和地址的流动方向安排连接线路 【特点】避免使用共享的总线，性能较好，但硬件总量大
示例图	 图 5.7　CPU 内部总线的数据通路和控制信号 • in表示该部件的允许输入控制信号；out表示该部件的允许输出控制信号 • 📢内部总线是指同一部件之间的线，系统总线是同一台计算机系统各部件之间的线

数据传输举例【以图5.7为例个】

寄存器之间的数据传输	• 寄存器之间的数据传送可以通过CPU内部总线完成 • 以PC寄存器为例，把PC内容送至MAR，实现传送操作的流程的控制信号为 PC--->Bus：PCout有效，PC内容送总线 Bus--->MAR：MARin有限，总线内容送MAR
主存与CPU之间的数据传送	• 主存与CPU之间的数据传送也要借助CPU内部总线完成 • 以CPU从主存读取指令为例，实现传送操作的流程的控制信号为 PC--->Bus--->MAR：PCout和MARin有效，现行指令地址--->MAR 1--->R：CU发读命令 MEN(MAR)--->MDR：MDRin有效 MDR--->Bus--->IR：MDRout和IRin有效，现行指令--->IR
执行算术或逻辑运算	• 以加法运算为例 Ad(IR)--->Bus--->MAR：MDRout和MARin有效 1--->R：CU发出读命令 MEN--->数据线--->MDR：操作数从存储器--->数据线--->MDR MDR--->Bus--->Y：MDRout和Yin有效，有效数--->Y (ACC)+(Y)--->Z：ACCout和ALUin有效，CU向ALU发加命令，结果--->Z Z--->ACC：Zout和ACCin有效，结果--->ACC

异常和中断控制【可以结合OS学习】

基本概念	异常 • 由CPU内部产生的意外事件被称为异常 • 是CPU执行一条命令时，由CPU在其内部检测到的、与正在执行指令相关的同步事件。 中断 • 由来自CPU外部的设备发出的中断请求被称为中断（常用于输入输出） • 典型的由外部设备触发的、与当前正在执行的指令无关的异步事件
异常分类	• 故障和自陷为异常，而终止异常和外中断属于硬件中断 故障：在引起故障的指令启动之后、执行结束前被检测到的异常事件 【举例】指令译码时，出现“非法操作码”；取数据时，发生“缺段”或“缺页”；发现除数为零等 自陷：也称陷阱或陷入，是预先安排的一种“异常事件”，就像预先设置好的“陷阱”一样 【举例】x86机器中，用于程序调试“断点设置”和单步跟踪功能；系统调用指令；条件自陷指令等 终止：若在执行指令的过程中发生了使计算机无法继续执行的硬件故障，那么程序将无法继续执行，只能终止 【举例】控制器出错、存储器校验错，调出中断服务程序来重启系统。
中断分类	• 外部I/O设备通过特定的中断请求信号线向CPU提出中断请求 • CPU每执行完一条指令就检查中断请求信号线，若检测到中断请求，则进入中断响应期 可屏蔽中断：• 通过可屏蔽中断请求线INTR向CPU发出的中断请求 • CPU可以通过在中断控制器中设置相应的屏蔽字来屏蔽或不屏蔽它，被屏蔽的中断信号将不被送到CPU。 不可屏蔽中断：• 通过不可屏蔽中断请求线NMI向CPU发出的中断请求 • 通常是非常紧急的硬件故障，如电源掉电等。
不同点	• “缺页”或“溢出”等异常事件是由特定指令在执行过程中产生的 • 中断不与任何指令相关联，也不阻止任何指令的完成 • 异常的检测由CPU自身完成，不必通过外部的某个信号通知CPU
异常和中断响应过程	• 响应过程不可被打断，整个中断处理过程是软/硬件协同实现的 1.关中断：• 在保存断点和程序状态期间，不能被新的中断打断 • 通常通过设置“中断允许”（IF）触发器来实现 2.保存断点和程序状态：• 为了能在异常和中断处理后正确返回到被中断的程序中继续运行 • 必须将程序的断点（返回地址）送到栈或特定寄存器中 • 通常保存在栈中，这是为了支持异常或中断的嵌套 3.识别异常和中断并转到相应的处理程序中：• 识别有两种方式——软件识别和硬件识别 • 异常常采用软件识别，中断则采用硬件识别

多处理机的基本概念

SISD，SIMD，MIMD的基本概念

- 基于指令流的数量和数据流的数量，将计算机体系结构分为SISD，SIMD，MISD和MIMD

单指令流单数据流结构（SISD）	• 传统的串行计算机结构，通常只包含一个处理器和一个存储器 • 有些使用流水线的方式，所以有时会设置多个功能部件，并采用多模块交叉方式组织存储器
单指令流多数据流结构（SIMD）	• 一个指令流同时对多个数据流进行处理，一般称为数据级并行技术 • 由一个指令控制部件、多个处理单元组成。 • 每个处理单元虽然执行的都是同一条指令 • 但每个单元都有自己的地址寄存器，就有了不同的数据地址 • 一个顺序应用程序被编译之后 • 可能按照SISD组织并运行与串行硬件上，也可能按SIMD组织并运行于并行硬件上 • for循环效率高，但switch或case时效率低 • 向量处理器也是SIMD的变体，是一种实现了直接操作一维数组（向量）指令集的CPU
多指令流单数据流结构（MISD）	• 同时执行多条指令，处理同一个数据，实际上不存在这样的计算机
多指令流多数据流结构（MIMD）	• 同时执行多条指令，处理多个不同的数据 • 分为多计算机系统和多处理器系统 • 多计算机系统 　• 每个计算机节点都具有各自的私有存储器，并且具有独立的主存地址空间 　• 不能通过存取指令来访问不同节点的私有存储器 　• 而要通过消息传递进行数据传送，也称为消息传递MIMD • 多处理器系统 　• 共享存储多处理器（SMP）系统的简称 　• 它具有共享的单一地址空间，通过访存指令来访问系统中的所有存储器，也称共享存储MIMD
联系与区别	• SIMD和MIMD是两种并行计算模式 • SIMD是一种数据级并行模式 • MIMD是一种并行程度更高的线程级并行或线程级以上并行计算模式

硬件多线程的基本概念

- 线程的切换包含很多开销，频繁切换影响系统性能，为了减少开销，便诞生了硬件多线程
- 硬件多线程中必须为每个线程提供单独的通用寄存器组、单独的程序计数器等
- 线程的激活只需要激活选中的寄存器，从而省略了与存储器数据交换的环节，节省了开销

细粒度多线程	• 多个线程之间轮流交叉执行指令，多个线程之间的指令是互不相关的 • 可以乱序并行执行 • 该方式下，处理器能在每个时钟周期切换线程。
粗粒度多线程	• 仅在一个线程出现较大开销的阻塞时，才切换线程 • 如Cache缺失 • 当发生流水线阻塞的时候，必须清除被阻塞的流水线 • 新线程的指令开始执行前需要重载流水线，开销较上一种较大
同时多线程（SMT）	• SMT是上述两种多线程技术的变体 • 它是实现指令级并行的同时，实现线程级的并行 • 即在同一时钟周期内，发射不同线程中的多条指令执行 • Intel处理器中的超线程即是SMT • 在一个单处理器或的那个核中设置了两套线程状态部件，共享高速缓存和功能部件

多核处理器的基本概念

- 将多个处理单元集成到单个CPU中，每个处理单元称为一个核（core）
- 每个核可以有自己的Cache，也可以共享一个Cache
- 所有核一般都是对称的，并且共享主存，因此多核属于共享存储的对称多处理器
- 在多核计算机系统中，若要充分发挥硬件的性能，必须采用多线程执行，使每个核在同一时刻都有线程在执行，这是真正的并行执行。

共享内存多处理器的基本概念

- 具有共享的单一物理地址空间的多处理器被称为共享内存多处理器（SMP）
- 处理器通过存储器中的共享变量相互通信，所有处理器都能通过存取指令访存任何存储器的位置
- 即使这些系统共享同一个物理地址空间，它们仍然可以在自己的虚拟地址空间中单独地运行程序
- 单一地址空间的多处理器分类
  - 统一存储访问（UMA）多处理器
    - 根据处理器与共享存储器之间的连接方式
    - 分为基于总线、基于交叉开关网络和基于多级交换网络连接等几种处理器
    - 每个处理器对所有存储单元的访问时间都是大致相同的
  - 非统一存储访问（NUMA）多处理器
    - 处理器中不带高速缓存时，被称为NC-NUMA
    - 处理器中带有一致性高速缓存时，被称为CC-NUMA（某些访问请求要比其他的快）