[结构组成 2](#_Toc492923523)

[指令执行过程 3](#_Toc492923524)

[微操作与时序 3](#_Toc492923525)

[取指周期 4](#_Toc492923526)

[间址周期(分析指令) 5](#_Toc492923527)

[执行周期 6](#_Toc492923528)

[中断周期 7](#_Toc492923529)

[控制信号的作用 8](#_Toc492923530)

[控制信号 8](#_Toc492923531)

[同步时序控制 9](#_Toc492923532)

[其他时序控制方式 11](#_Toc492923533)

[指令流水 12](#_Toc492923534)

[指令流水 12](#_Toc492923535)

[传统的指令执行方案 12](#_Toc492923536)

[流水线方案 13](#_Toc492923537)

[流水线结构 15](#_Toc492923538)

[流水线性能指标 16](#_Toc492923539)

[影响流水线性能的因素与相应的处理办法 17](#_Toc492923540)

[结构相关(资源相关) 17](#_Toc492923541)

[数据相关 18](#_Toc492923542)

[控制相关 19](#_Toc492923543)

[流水线的多发技术 20](#_Toc492923544)

[超标量技术 20](#_Toc492923545)

[超流水技术 21](#_Toc492923546)

[超长指令字技术 22](#_Toc492923547)

[CISC与RISC介绍 23](#_Toc492923548)

# 结构组成

解释指令：取值、分析、、取操作数、执行、保存结果

保证指令的有序执行

完成一条指令：

取指令：PC寄存器：存放当前欲执行指令的地址，具有技术功能

分析指令IR：存放当前欲执行的指令

执行指令：CU

# 指令执行过程

## 微操作与时序

|  |  |
| --- | --- |
| 微操作 | 微操作是计算机中最基本的、不可再分解的操作  每条指令都可以看作一个微操作序列  由于计算机的指令系统是确定的，其包含的指令也是固定不变的，因此机器有哪些微操作也是确定的  根据计算机的指令系统列出所有的微操作  那么计算机工作其实就是从这些微操作中进行选择并执行。 |
| 时钟周期 | 使用机器主振电路，如晶体振荡器发出的脉冲信号经整形(倍频或分频)，产生时钟信号  时钟信号的频率就是CPU的主频  时钟周期是CPU操作的最基本单位  在一个时钟周期内机器执行一个微操作，或几个同时执行的微操作 |

## 取指周期

指令字长一般都是取存储字长的整数倍；

定长指令字结构，所有指令的取指操作是相同的；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令字长=存储字长 | 微操作 | | | |
| 节拍 | 传输内容 | 传输路径 | 记作 |
| T0 | 指令地址 | PC🡪片内总线🡪MAR | PC🡪MAR |
| 读命令 | CU🡪系统控制总线🡪主存 | 1🡪R |
| T1 | 存储地址 | MAR🡪系统地址总线🡪主存 | M(MAR)🡪MDR |
| 存储数据 | 主存🡪系统数据总线🡪MDR |
| (PC)+1 | PC | (PC)+1🡪PC |
| T2 | 存储数据 | MDR🡪片内总线🡪IR | MDR🡪IR |
| 操作码 | IR🡪片内总线🡪CU | OP(IR)🡪CU |
|  | | | | |
| 指令字长=存储字长×2  需要两次访存 | 微操作 | | | |
| 节拍 | 传输内容 | 传输路径 | 记作 |
| T0 | 指令地址 | PC🡪片内总线🡪MAR | PC🡪MAR |
| 读命令 | CU🡪系统控制总线🡪主存 | 1🡪R |
| T1 | 存储地址 | MAR🡪系统地址总线🡪主存 | M(MAR)🡪MDR |
| 存储数据 | 主存🡪系统数据总线🡪MDR |
| (PC)+1 | PC | (PC)+1🡪PC |
| T2 | 存储数据 | MDR🡪片内总线🡪IR | MDR🡪IR |
| 操作码 | IR🡪片内总线🡪CU | OP(IR)🡪CU |
|  | | | |
| T3 | 指令地址 | PC🡪片内总线🡪MAR | PC🡪MAR |
| 读命令 | CU🡪系统控制总线🡪主存 | 1🡪R |
| T4 | 存储地址 | MAR🡪系统地址总线🡪主存 | M(MAR)🡪MDR |
| 存储数据 | 主存🡪系统数据总线🡪MDR |
| (PC)+1 | PC | (PC)+1🡪PC |
| T5 | 存储数据 | MDR🡪片内总线🡪IR | MDR🡪IR |
| 操作码 | IR🡪片内总线🡪CU | OP(IR)🡪CU |

## 间址周期(分析指令)

触发标志：IND=1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 一次间址周期 | 微操作 | | | |
| 节拍 | 信号内容 | 传输路径 | 记作 |
| T0 | 形式地址码 | IR🡪片内总线🡪MAR | Ad(IR)🡪MAR |
| 读命令 | CU🡪系统控制总线🡪主存 | 1🡪R |
| T1 | 形式地址码 | MAR🡪系统地址总线🡪主存 | M(MAR)🡪MDR |
| 有效地址码 | 主存🡪系统数据总线🡪MDR |
| T2 | 有效地址码 | MDR🡪片内总线🡪IR | MDR🡪Ad(IR) |
|  | | | | |
| 二次间址周期 | 微操作 | | | |
| 节拍 | 信号内容 | 传输路径 | 记作 |
| T0 | 形式地址码1 | IR🡪片内总线🡪MAR | Ad(IR)🡪MAR |
| 读命令 | CU🡪系统控制总线🡪主存 | 1🡪R |
| T1 | 形式地址码1 | MAR🡪系统地址总线🡪主存 | M(MAR)🡪MDR |
| 形式地址码2 | 主存🡪系统数据总线🡪MDR |
| T2 | 形式地址码2 | MDR🡪片内总线🡪IR | MDR🡪Ad(IR) |
|  | | | |
| T3 | 形式地址码2 | IR🡪片内总线🡪MAR | Ad(IR)🡪MAR |
| 读命令 | CU🡪系统控制总线🡪主存 | 1🡪R |
| T4 | 形式地址码2 | MAR🡪系统地址总线🡪主存 | M(MAR)🡪MDR |
| 有效地址码 | 主存🡪系统数据总线🡪MDR |
| T5 | 有效地址码 | MDR🡪片内总线🡪IR | MDR🡪Ad(IR) |

指令的间址周期由一次间址、二次间址、多次间址，因此它们的操作是不同的

存储器间址通过形式地址访存

寄存器间址通过寄存器内容访存

在间址周期阶段，间址次数可以间址周期标志触发器IND确定，当它为0时，表示间址寻址结束

## 执行周期

触发标志：EX=1

在执行周期开头，IR存储的地址码必然是有效地址

不同指令的执行周期的微操作不同，计算机在设计时可以根据需求创建指令，即创建指令所包含的微操做；

下面是一些常用指令的执行周期的微操作序列

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | 微操作 | | | |
| 节拍 | 信号内容 | 传输路径 | 记作 |
| 取操作数  LDA X | T0 | 有效地址码 | IR🡪片内总线🡪MAR | Ad(IR)🡪MAR |
| 读命令 | CU🡪系统控制总线🡪主存 | 1🡪R |
| T1 | 有效地址码 | MAR🡪系统地址总线🡪主存 | M(MAR)🡪MDR |
| 操作数 | 主存🡪系统数据总线🡪MDR |
| T2 | 操作数 | MDR🡪片内总线🡪ACC | MDR🡪ACC |
|  | | | | |
| 存操作数  STA X | T0 | 有效地址码 | IR🡪片内总线🡪MAR | Ad(IR)🡪MAR |
| 写命令 | CU🡪系统控制总线🡪主存 | 1🡪W |
| T1 | 操作数 | ACC🡪片内总线🡪MDR | ACC🡪MDR |
| T2 | 有效地址码 | MAR🡪系统地址总线🡪主存 | MDR🡪M(MAR) |
| 操作数 | MDR🡪系统数据总线🡪主存 |
|  | | | | |
| 加法指令  ADD X | T | 目标操作数 | ACC🡪片内总线🡪ALU | (ACC)+(MDR)🡪ACC |
| 源操作数 | MDR🡪片内总线🡪ALU |
| 结果 | ALU🡪片内总线🡪ACC |
|  | | | | |
| 无条件转移  JMP X | T | 指令地址 | Y🡪片内总线🡪PC | Ad(IR)🡪PC |
| 条件(负)转移  BAN X | T |  |  | 🡪PC |
| 条件(零)转移  BAZ X | T | 指令地址 | Y🡪片内总线🡪PC | 🡪PC |
|  | | | | |
| 清除累加器  CLA | T | 0 | ACC | 0🡪ACC |
| 累加器取反  COM | T |  | ACC | 🡪ACC |
| 算数右移一位  SHR | T |  | ACC | L(ACC)🡪R(ACC)，🡪(符号位不变) |
| 循环左移一位  CSL | T |  | ACC | R(ACC)🡪L(ACC)，🡪[或] |
| 停机  STP | T | 0 | G | 0🡪G |

## 中断周期

触发标志：INT=1

中断周期(如果有的话)，被放在一个指令周期的末尾

在中断周期开头，CPU查询是否有中断请求，如果有则进入中断周期

在中断周期中，中断隐指令会自动完成保护断点、寻找中断服务程序入口地址以及硬件关中断的操作

当CPU采用中断方式其进行信息传输时，CPU在每条指令执行结束前，都要发中断查询信号，如果有中断请求信号，就会进入中断响应阶段，称为中断周期

|  |  |  |  |
| --- | --- | --- | --- |
| 微操作 | | | |
| 节拍 | 信号内容 | 传输路径 | 记作 |
| T0 | (SP)-1 | SP | SP🡪MAR |
| 写命令 | CU🡪系统控制总线🡪主存 | 1🡪W |
| T1 | 程序断点 | PC🡪片内总线🡪MDR | PC🡪MDR |
| T2 | 存储器地址 | MAR🡪系统地址总线🡪主存 | MDR🡪M(MAR) |
| 程序断点 | MDR🡪系统数据总线🡪主存 |
| 0 | G | 向量地址🡪PC |
|  | 0 | EINT | 0🡪EINT |

T6时刻关闭中断，将允许中断触发器清0，该操作可直接由硬件线路完成

|  |  |
| --- | --- |
|  | T1时刻，把用于保存程序断点的存储器特殊地址  这里使用堆栈来保存程序断点的地址  进栈操作是将SP减1，这和传统意义上的进栈操作相反，因为计算机的堆栈中都是向低地址增加，所以进栈操作是减1不是加1 |

# 控制信号的作用

## 控制信号

|  |  |  |
| --- | --- | --- |
| 控制信号与微操作 | 控制信号又称为微操作命令或微命令；  在计算机系统中，每一个微操作，对应一个控制信号，前面已经讲到机器有哪些微操作也是确定的，所以机器有哪些控制信号也是确定的。  微操作是微命令的执行过程，所有微操作的执行都是在控制单元发出的控制信号(微操作命令)控制下完成的，即先由控制单元发出控制信号，再执行微操作。 | |
| 相容性与互斥型 | 相容性微命令：对应的微操作可以同时执行  互斥性微命令：对应的微操作不可以同时执行  相容和互斥都是相对的，一个微命令可以和一些微命令相容，和另一些微命令互斥 | |
| 控制信号的作用 | 控制信号在每个时钟周期的上升沿，由控制单元CU发出；  控制信号必须按一定次序发出，才能使机器有序的工作。  其具体的实现方式可以见《控制单元》 | |
| 控制信号 | 不采用CPU内部总线 | CPU内各个部件之间直接相连，  控制信号控制门电路，门电路控制部件之间的数据传输的通断  如：表示控制MAR，的控制信号  有效：打开MAR送往地址总线的输出门，执行微操作-M(MAR)🡪MDR |
| 采用CPU内部总线 | ：控制信号控制寄存器到总线或总线到寄存器的输入控制门  ：控制从程序计数器到内部总线的输出控制门  表示输入控制，表示输出控制  与不采用片内总线相比，采用片内总线的方式，多用了两个寄存器：Y和Z  这是由于ALU是一个组合逻辑电路，再起运算过程中必须保持两个输入端不变，其中一个输入可以从Y寄存器中获得，另一个输入可以从内部总线上获得。  当CPU内有多个通用寄存器时，由于设置了寄存器Y，可实现任意两个寄存器之间的算术逻辑运算  此外，ALU的输出不能直接与内部总线相连，因为其输出又会通过总线反馈到ALU的输入，影响运算的正确性，故用寄存器Z暂存运算结果，再根据需要送至指定的目标 |
| 读/写信号 | CU发送至内存的控制信号 |
|  |  |

## 同步时序控制

|  |  |
| --- | --- |
| 指令周期 | 指令周期是CPU取出并执行一条指令的全部时间，不同指令的操作不同，指令周期也不同。  指令周期可以划分为4个部分：取指周期，间址周期，执行周期，中断周期  其中取指周期和执行周期必须包含在一个指令周期中，另外两个随情况而定， |
| 冯诺依曼计算机根据指令周期的不同阶段来区分从存储器取出的是指令还是数据，更准确地说：  计算机在不同的阶段对存储器进行读操作取出的代码，有不同的用处。  取指周期取出的是指令，执行周期取出的是数据  控制器可以区分存储单元中存放的是指令还是数据，存储器本身无法区分存储单元中存放的是指令还是数据， |
| 程序是由指令组成的，程序的执行其实就是指令的顺序执行： |
| 机器周期 | 把一个指令划分成多个操作，机器周期是执行其中每一步完整操作所需要的时间  具体的划分方法则要根据该指令系统中的各个指令的执行步骤  目的是要使得机器周期作为所有指令执行过程中的一个基准时间，这样就可以用“x个机器周期”来描述一个指令的执行时间  以存取周期作为基准时间，即内存中读取一个指令字的最短时间作为机器周期。  把通过一次总线事物访问一次主存或I/O的时间定为一个机器周期 |
| 如：设定机器周期是取一次指令所用的时间，那么：  指令字长等于存储字长的2倍的指令，它的取指周期等于机器周期的2倍；  如果指令字长等于存储字长，取值周期等于机器周期 |
| 多级时序系统 | 机器周期、时钟周期组成了多级时序系统  一般来说，CPU的主频越快，机器的运行速度也越快。  在机器周期所含时钟周期数相同的前提下，两级平均指令执行速度比等于两机主频之比  如CPU主频是8MHZ，那么平均指令执行速度是0.8MIPS  若要得到平均指令执行速度是0.4MIPS的机器，则只需要用主频是8×0.4/0.8=4MHZ的CPU即可  实际上机器的速度不仅和主频有关，还与机器周期中所含的时钟周期数以及指令周期中所含的机器周期数有关  同样朱批判的机器，由于机器周期所含有的时钟周期数不同，运行速度也不同  机器周期所含时钟周期数少的机器，速度更快 |

CPI：执行一条指令平均需要的时钟周期数

单周期处理中即指所有指令的指令周期为一个时钟周期，那么每条指令的CPI=1

要考虑比较慢的指令，所以处理器的时钟频率较低

同步控制方式

任何一条指令或指令中任何一个微操作的执行都是事先确定的，并且都是受统一基准时标的时序信号所控制的方式

每个机器周期包含的时钟周期数是相同固定的

如果机器内的存储器存取周期不统一，那么只有把最长的存取周期作为机器周期，才能同步控制，否则取指令和取数时间不同，无法用同一的标准。

又如有些不访存的指令，执行周期的微操作很少，无须这个固定的节拍数所指定的那么多节拍

因此为了提高CPU效率，在同步控制中又有三种方案

|  |  |
| --- | --- |
|  |  |
| 采用定长的机器周期 | 不论指令所对应的微操作序列由多长，也不管微操作的简繁，一律一最长的微操作序列和最复杂的微操作作为标准，  采取完全同一的，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各种不同的指令  这种方案对于微操作序列较短的指令来说，造成时间上的浪费 |
| 采用不定长的机器周期 | 每个机器周期内的节拍数可以不相等  这种控制方式可以解决微操作执行时间不统一的问题。通常把大多数微操作安排在一个较短的机器周期内完成，而对某些复杂的微操作，采用延长机器周期或增加节拍的办法来解决 |
| 中央控制和局部控制相结合 | 将机器的大部分指令安排在统一的、较短的机器周期内完成，称为中央控制  将少数操作复杂的指令中的某些操作，如乘法和浮点运算等，采用局部控制方式  注意：  1.使局部控制的每一个节拍的宽度与中央控制的节拍宽度相同  2.将局部控制节拍作为中央控制中机器节拍的延续，插入到中央控制的执行周期内，是机器已同样的节奏 工作，保证了局部控制和中央控制的同步  的多少克根据情况而定，对于乘法，当操作数位数固定后，的个数也就确定了。而对于浮点运算的对接操作，由于移位次数不是一个固定值，因此的个数不能事先确定 |
| 以乘法指令为例  第一个机器周期采用中央控制的节拍控制取指令的操作，接着仍然用中央控制的、、节拍去完成将操作数从存储器总取出并送至寄存器的操作，然后转局部控制，用局部控制节拍完成重复加和移位的操作 |

假设机器采用同步控制，每个机器周期为3个节拍，

1.有些微操作的次序是不容改变的，故安排微操作节拍必须注意微操作的先后顺序

2.凡是被控制对象不同的微操作，若能在一个节拍内执行，应该尽可能地安排在同一个节拍内，以节省时间

3.若微操做所占时间不长，应该将它们安排在一个节拍内完成，并且允许这些微操作有先后次序

|  |  |  |
| --- | --- | --- |
| 微操作  节拍控制 | 取值周期 | 根据原则2，节拍，安排两个微操作：PC🡪MAR，1🡪R  根据原则2，节拍，安排两个微操作：M(MAR)🡪MAR，(PC)+1🡪PC  节拍：安排MDR🡪IR，考虑到指令译码时间较短，根据原则3，可将指令译码OP(IR)🡪ID也安排在节拍  实际上，(PC)+1🡪PC操作也可以安排在节拍，因为一旦PC🡪MAR后，PC的内容就可以修改 |
| 间址周期 | ：Ad(IR)🡪MAR，1🡪R  ：M(MAR)🡪MDR  ：MDR🡪Ad(IR) |
| 执行周期 | 本例的执行周期只有一个微操作，按照同步的原则，这个微操作可以安排在的任意一个节拍，其余节拍为空  ：空  ：空  ：MDR🡪Ad(IR) |
| 中断周期 | ：Ad(IR)🡪MAR，1🡪R  ：M(MAR)🡪MDR  ：MDR🡪Ad(IR) |

## 其他时序控制方式

|  |  |  |
| --- | --- | --- |
| 异步控制方式 | 不存在基准时标信号，没有固定的周期节拍和严格的时钟同步，执行每条指令和每个微操作需要多少时间就占用多少时间。  这种方式微操作的时序由专门的应答线路控制，即当CU发出执行某一微操作的控制信号后，等待执行部件完成了该操作后发送“回答”信号给CU，CU再开始新的微操作，  这样CPU就没有空闲状态  但因为需要采用各种应答电路，其结构复杂 | |
| 联合控制方式 | 同步控制和异步控制相结合  对各种不同指令的微操作实行大部分同一，小部分区别对待的办法。  如：对每条指令都有的取值操作，采用同步方式控制  对那些时间难以确定的微操作，如I/O操作，则采用异步控制，以执行部件送回的“回答”信号作为本次微操作的结束 | |
| 人工控制方式 | 为了调机和软件开发的需要，在机器面板或内部设置一些开关或按键，来达到人工控制的目的 | |
| 复位键  Reset | 按下Reset键，是计算机处于初始状态。  当机器出现死锁状态或无法继续运行的时候，可按此键  若在机器运行的时候按此键，将会破坏机器内某些状态而引起错误，因此要慎用  有些微型计算机没有此按键，可采用停电再上电的方法 |
| 连续执行和单条执行  转换开关 | 由于调机的需要，有时需要观察执行完一条指令后的机器状态，有时有需要观察连续运行程序后的结果，设置该开关，提供两种选择 |
| 符合停机开关 | 这组开关指示存储器位置，当程序运行到与开关指示的地址相符时，机器便停止运行 |

## 

指令流水

# 指令流水

## 传统的指令执行方案

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 指令1取指 | 指令1执行 | 指令2取指 | 指令2间址 | 指令2执行 | 指令3取指 | …… |

这是传统的指令执行方案，即一条指令执行完毕，才能执行下一条指令

图中，取指令操作可以由指令部件完成，执行指令操作可由执行部件完成，

取指周期：指令部件工作，执行部件空闲，

执行周期：执行部件工作，指令部件空闲，

如果执行周期中不访问主存，那么该执行周期就可以和某取值周期同时进行；可以利用这段时间进行取指操作

|  |  |
| --- | --- |
| 单指令周期 | 对所有指令都选用相同的执行时间来完成，称为单指令周期方案。  每一条指令都在固定的时钟周期内完成，指令之间串行执行，即下一条指令只能在前一条指令执行结束后才能启动。  指令周期取决于执行时间最长的指令的执行时间  对于那些本来可以在更短时间内完成的指令，要使用这个较长的周期来完成，会降低整个系统的运行速度 |
| 多指令周期 | 对不同的指令选用不同的执行步骤来完成，  指令之间串行执行，即下一条指令只能在前一条指令结束之后才能启动  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  而不再要求所有指令占用相同的执行时间 |

## 流水线方案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 为了提高处理机速度，有两种方案：  (1)提高器件的性能  (2)改进系统的结构，开发系统的并行性 | | | | | |
| 器件性能 | 计算机的发展史就是按器件性能分成电子管、晶体管、集成电路、大规模集成电路 4代的  每一次器件的更新换代都可实现突破性进展  如今半导器件的集成度越来越接近物理极限，很难在这方面做出突破 | | | | |
| 并行性 | 定义 | 只要在时间上互相重叠，就存在并行性  并行性包含两个方面：同时性或并发性  同时性：多个事件在同一时刻发生  并发性：多个时间在同一时间段发生 | | | |
| 分级 | 粗粒度  (过程级) | 算法(软件)实现 | 在多个处理机上分别运行读个进程，  由多台处理机合作完成一个程序 | 作业级(程序级) |
| 任务级(进程级) |
| 细粒度  (指令级) | 硬件实现 | 在处理机的操作级和指令级的并行性  指令的流水作业 | 指令之间级 |
| 指令内部级 |

|  |  |  |
| --- | --- | --- |
| 指令流水 | 指令之间可以并行执行的方案  其追求的目标是力争在每个时钟脉冲周期完成一条指令的执行过程(理想情况)  通过在每一个时钟周期启动一条指令，尽量让多条指令同时运行，但各自出在不同的执行步骤中 | |
| 指令预取技术 | 在某条指令的执行周期期间，执行取指操作，取出下一条指令，并将它暂存起来；  等执行部件空闲，将暂存的指令传给执行部件执行  同时指令部件又可取出下一条指令并暂存起来  不采用指令预取技术：每个指令周期都需要取指令 | |
| Cache技术 | 不采用Cache技术：每次取指令都至少要访问内存一次，要求指令字长与存储字长相等且边界对齐 | |
| 工作过程 | 建立时间 |  |
| 稳定状态 | 流水线的各段都处于工作状态 |
| 排空时间 |  |
|  |  |

指令的处理过程分为N个阶段，各段时间相等

|  |  |  |
| --- | --- | --- |
| 二级流水方案 | 取指令 |  |
| 执行指令 |  |
|  |  |
| 五级流水方案 |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| 六级流水方案 | 取值FI | 处理器有6个操作部件可以同时对6条指令进行加工 |
| 指令译码DI |
| 计算操作数地址C0 |
| 取操作数FO |
| 执行指令EI |
| 写操作数WO |

## 流水线结构

|  |  |
| --- | --- |
| 指令流水线 |  |
| 运算流水线 |  |

## 流水线性能指标

|  |  |  |
| --- | --- | --- |
| 吞吐率 | 吞吐率 | 单位时间内流水线完成的指令的数目 |
| 最大吞吐率 | 流水线达到稳定状态后的吞吐率  m段流水线，流水线各段时间： |
| 实际吞吐率 | 考虑建立时间和排空时间、以及其他相关因素使流水线无法连续流动  m段流水线，流水线各段时间：  完成n条指令在m段流水线上花费的时间：  当n>>m时， |
| 加速比 | 流水线的速度与等功能的费流水线的速度之比  m段流水线，流水线各段时间：  完成n条指令在m段流水线上花费的时间：  完成n条指令在非流水线上花费的时间：  当n>>m时，，即当流水线各段时间相等时，最大加速比等于流水线的段数 | |
| 效率E | 流水线中各功能段的利用率  由于流水线有建立时间和排空时间，因此各段功能段的设备不可能移植处于工作状态，总有一段空闲时间  如4段流水线，各段时间均为  是流水线各段处于工作时间的时空区  流水线中隔断总的时空区是  通常用流水线各段处于工作时间的时空区与流水线中各段总的时空区之比 | |

# 影响流水线性能的因素与相应的处理办法

1.并不是每条指令都会经过6个阶段，如去数指令并不需要WO

3.6个阶段实际时间不等

指令的执行周期一般大于取指周期，下条指令的取值周期完成后，还要等一段时间，本条指令的执行周期才结束

## 结构相关(资源相关)

|  |  |  |
| --- | --- | --- |
| 影响因素 | 存储器访问冲突  多条指令进入流水线重叠执行，不同的指令争用同一功能部件，产生资源冲突  硬件资源满足不了要求  通常，指令和数据保存在统一存储器中，且只有一个访问口  如果在某个时钟周期，流水线纪要完成某条指令对操作数的存储访问操作，又要完成另一条指令的取值操作，就会发生访存冲突  如FI、FO、WO都设计存储器的访问，如果出现冲突就不能并行执行 | |
| 解决办法 |  | 让流水线在完成前一条指令对数据的存储器访问时，暂停(一个时钟周期)取后一条指令的操作， |
|  | 设置两个独立的存储器分别存放操作数和指令，以免取指令和去操作数同时进行时互相冲突 |
| 指令预取技术 | 如在8086中设置的指令队列，将指令预先渠道指令队列中排队。  指令预取技术的实现基于访存周期很短的情况，例如，在执行阶段，取数时间很短，因此执行指令时，主存会有空闲，此时，只要指令队列空出，就可以取下一条指令，并放至空出的指令队列中，从而保证在执行第K条指令的同时对第K+1条指令进行译码，实现“执行K”与“分析K+1”重叠 |

## 数据相关

|  |  |  |
| --- | --- | --- |
| 影响因素 | 当多条指令进入流水线重叠执行，后继指令需要用到前面指令执行结果  由于各条指令的重叠操作，可能改变对操作数的读写访问顺序，从而导致了数据相关冲突 | |
| 解决办法 | 后推法 | 遇到数据相关时，就停顿后继指令的运行，知道前面指令的结果已经生成 |
| 定向计数 | 又称为旁路计数或相关专用通路技术  不必带某条指令的执行结果送回到寄存器后，再从寄存器中取出该结果，作为下一条指令的源操作数，  而是直接将执行结果送到其他指令所需要的地方  这样可以使流水线不用发生停顿 |

## 控制相关

|  |
| --- |
| 分支指令和其他改变PC值的指令 |
| 主要由转移指令引起  当转移发生时，将使流水线的连续流动遭到破坏。  执行转移指令时，根据是否发生转移，可能将PC内容改变成转移目标地址。也可能使PC加上一个增量，指向下一个指令的地址  其结果无法预测  本条指令执行周期结束时，才知道转移是否成功  如果成功，PC=转移目标地址  如果不成功，PC=下一个指令的地址  其结果无法预测  为了减少时间的损失而使用猜测法，即当条件转移指令从取指阶段进入执行阶段时，指令部件仍然按顺序预取下一条指令。  如果转移成功，丢掉索取的指令，再取新的指令  如果转移失败，那么正常执行，没有时间损失 |

# 流水线的多发技术

## 超标量技术

## 超流水技术

## 超长指令字技术

CISC与RISC介绍

如前面说过的，CPU其实内部已经含有一些小指令集，我们所使用的软件都要经过CPU内部的微指令集来达成才行。

那这些指令集的设计主要又被分为两种设计理念，这就是目前世界上常见到的两种主要CPU种类： 分别是精简指令集(RISC)和复杂指令集(CISC)系统。

|  |  |
| --- | --- |
| 精简指令集  RISC | Reduced Instruction Set Computing  这种CPU的设计中，微指令集较为精简，每个指令的运行时间都很短，完成的劢作也很单纯，指令的执行效能较佳；  但是若要做复杂的事情，就要由多个指令来完成。  常见的RISC微指令集CPU主要例如升阳(Sun)公司的SPARC系列、 IBM公司的Power Architecture(包括PowerPC)系列、和ARM系列等。  在应用方面，SPARC架构的计算机常用于学术领域的大型工作站中，包括银行金融体系的主服务器也都有这类的计算机架构； 至于PowerPC架构的应用上，例如新力(Sony)公司出产的Play Station 3(PS3)就是使用PowerPC架构的Cell处理器； 那ARM呢？你常使用的各厂牉扃机、PDA、导航系统、网络设备(交换器、路由器等)等，几乎都是使用ARM架构的CPU喔！ 老实说，目前世界上使用范围最广的CPU可能就是ARM呢 |
| 复杂指令集  CISC | Complex Instruction Set Computer  和RISC不同的，CISC在微指令集的每个小指令可以执行一些较低阶的硬件操作，指令数目多而且复杂， 每条指令的长度并不相同。因为指令执行较为复杂所以每条指令花费的时间较长， 但每条个别指令可以处理的工作较为丰富。  常见的CISC微指令集CPU主要有AMD、Intel、VIA等的x86架构的CPU。    由于AMD、Intel、VIA所开发出来的x86架构CPU被大量使用于个人计算机(Personal computer)用途上面， 因此，个人计算机常被称为x86架构的计算机！那为何称为x86架构呢？ 这是因为最早的那颗Intel发展出来的CPU代号称为8086，后来依此架构又开发出80286, 80386……因此这种架构的CPU就被称为x86架构了。  在2003年以前由Intel所开发的x86架构CPU由8位升级到16、32位，后来AMD依此架构修改新一代的CPU为64位， 为了区别两者的差异，因此64位的个人计算机CPU又被统称为x86\_64的架构  那么不同的x86架构的CPU有什么差异呢？除了CPU的整体结构(如第二层快取、每次运作可执行的指令数等)之外， 主要是在于微指令集的不同。  新的x86的CPU大多含有很先进的微指令集， 这些微指令集可以加速多媒体程序的运作，也能够加强虚拟化的效能，而且某些微指令集更能够增加能源效率，降低CPU耗电量 |

在最新的AMD/Intel 的X86架构中有如下这些重要的指令集：

多媒体指令集：MMX、SSE、SSE2、SSE3、SSE4、AMD-3DNow

虚拟化指令集：Intel-VT、AMD-SVM

省电功能：Intel-SpeedStep、AMD-PowerNow

64/32位兼容技术：AMD-AMD64、Intel-EM64T