CPU&控制器

1. CPU基本功能与结构

1.1. CPU功能：周而复始执行指令

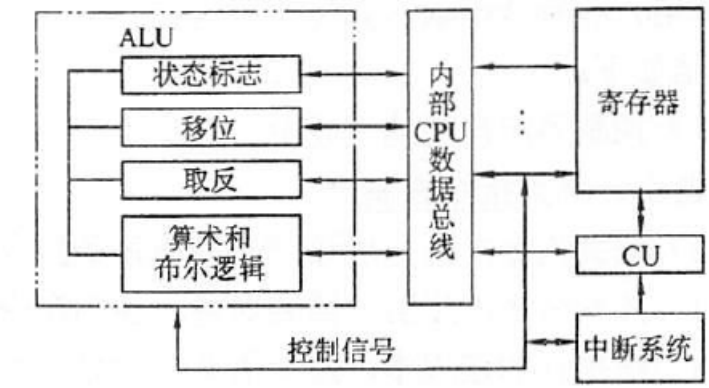
1️⃣由控制单元完成的：

1. 程序控制：控制指令执行顺序
2. 操作控制：根据指令，向相应部件发操作控制信号
3. 时间控制：什么时间执行什么操作

2️⃣由ALU完成的：数据加工，进行算术逻辑运算

3️⃣中断控制

1.2. CPU结构：ALU+CU+寄存器



1.3. CPU中的寄存器

1️⃣运算器中的寄存器

寄存器类别	描述
暂存寄存器	用于暂存从主存读来的数据，对应用程序员透明
累加寄存器ACC	通用寄存器，暂存ALU运算结果
通用寄存器组	存放操作数和地址信息，如AX/BX/CX/DX，对程序员可见
状态条件寄存器（PSW）	保由多个状态条件标志组成

2️⃣控制器中的寄存器

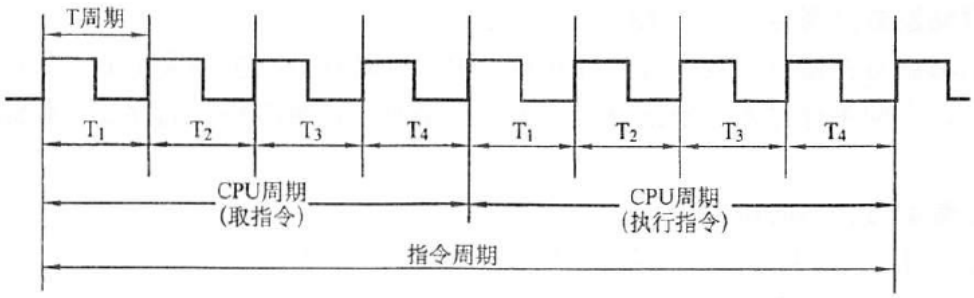
寄存器类别	描述	对程序员可见?
程序计数器PC	定位下一条指令的地址	可见
指令寄存器IR	保存当前正在执行的指令	不可见
存储器数据寄存器MDR	临时存放/中转从主存读出的数据或指令	不可见

寄存器类别	描述	对程序员可见?
存储器地址寄存器MAR	保存当前CPU所访问的内存单元的地址	不可见

2. 指令执行过程

2.1. 指令周期

1 几个周期的概念



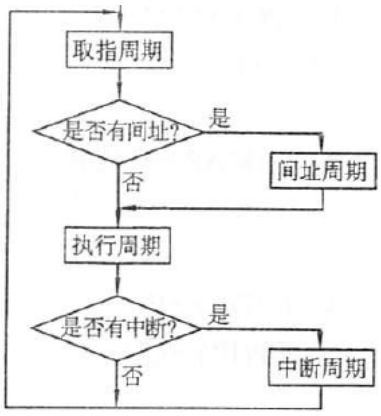
1. 指令周期：CPU完成一条指令的时间
2. CPU周期/基本周期/机器周期：指令周期被划分为多个CPU周期，每个CPU完成指令处理的一步
3. 时钟周期：最基本的单位，如图中的$T_1, T_2 \ldots$，是时钟频率的倒数

2 周期长短

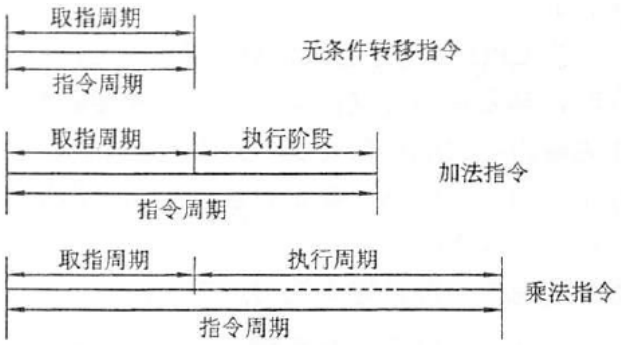
1. 时钟周期：固定不变
2. CPU周期占多少时钟周期：由CPU周期内需要完成什么操作决定
3. 指令周期包含几个时钟周期/包含什么类型的时钟周期：由指令性质决定

3 指令周期组成

1. 完整组成：取指周期+间接寻址周期+执行周期+中断周期，执行流程如下



2. 具体指令的指令周期组成



2.2. 指令执行方案

方案	特点	效率
单指令周期	所有指令相同时间内执行完，串行执行	效率低，所有指令执行时间=最长那个
多指令周期	不同类指令用不同执行步骤，串行执行	指令执行时间更灵活，提高了效率
流水线方案	并行执行，每时钟周期尝试完成一条指令	效率最高

2.3. 指令执行过程&信息流

2.3.1. 取指周期：按PC取指，然后PC递增/跳转

操作	描述
`(PC) -> MAR`	待执行指令的地址$\xrightarrow{\text{送入}}$地址缓冲寄存器
`1 -> R`	发出读命令**(固定写法)**
`M(MAR) -> MDR`	MAR存放的地址$\xrightarrow{\text{在主存中找到}}$待执行的指令$\xrightarrow{\text{将指令送入}}$数据缓冲寄存器
`MDR -> IR`	MDR存放的指令$\xrightarrow{\text{塞入}}$指令寄存器，因此(IR)表示指令本身
`OP(IR) -> CU`	指令的操作码$\xrightarrow{\text{传给}}$控制单元
`PC+1 -> PC`	PC+1，指向下一条指令

2.3.2. 间址周期：取出操作数有效地址

操作	描述
`AD(IR) -> MAR`	指令的(逻辑)地址码$\xrightarrow{\text{传给}}$地址缓冲寄存器
`1 -> R`	发出读取命令
`M(MAR) -> MDR`	MAR存放的逻辑地址$\xrightarrow{\text{在主存中找到}}$有效地址$\xrightarrow{\text{将有效地址送入}}$数据缓冲寄存器

2.3.3. 执行周期&中断周期

详见后面的控制器

3. 数据通路

功能在于：实现CPU内部的ALU-寄存器，寄存器-寄存器的数据交换

3.1. 数据通路结构

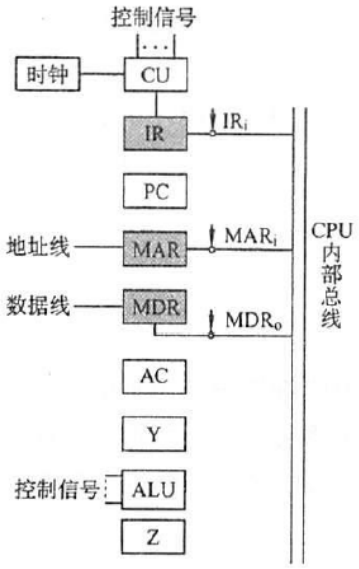


图 5-9　CPU 内部单总线结构

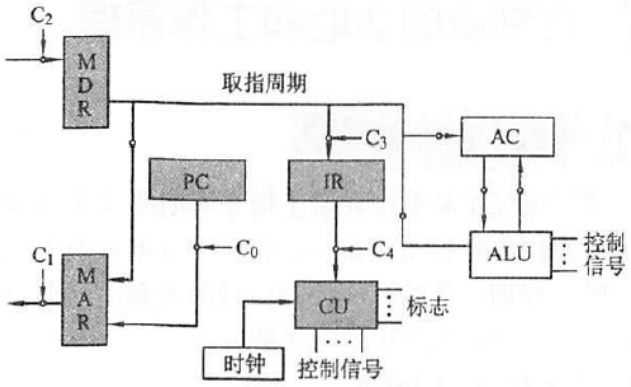


图 5-10　专用数据通路方式

1 CPU内部总线结构

1. 结构：将所有寄存器的输入/输出端连接到总线，总线有多条(多总线)/单条(单总线，如图)
2. 特点：结构简单，传输冲突较多，性能低

PS：图中i的角标表示允许输入控制信号，o表示允许输出控制信号

2 专用数据通路方式：避免总线共享，根据指令执行过程中数据/地址流向安排线路

3.2. 常见的数据传送

1 寄存器→寄存器：由CPU内部总线完成

操作	描述
`PC -> Bus`	PCout有效，PC内容送总线
`Bus -> MAR`	MARin有效，总线内容送MAR

2 主存-CPU数据传送：借助CPU内部总线，以下为CPU从主存读数据

操作	描述
`PC -> Bus -> MAR`	PCout和MARin有效，现行指令地址→MAR
`1 -> R`	发出读取命令
`MEM(MAR) -> MDR`	MDRin有效，根据指令地址，在主存中找出指令内容，送给MDR

操作	描述
`MDR -> Bus -> IR`	MDRout和IRin有效，现行指令送IR

3️⃣执行算术逻辑运算

1. ALU的一端接一个暂存器Y，一个操作数经CPU总线送给Y，Y的内容始终在此端口有效
2. ALU的另一端，从总线接收另一个操作数
3. 最后ALU运算的结果再送入另一个暂存器Z

操作	描述
`Ad(IR) -> Bus -> MAR`	PCout和MARin有效，现行指令地址→MAR
`1 -> R`	发出读取命令
`MEM -> 数据线 -> MDR`	MDRin有效，操作数送给MDR
`MDR -> Bus -> Y`	MDRout和Yin有效，操作数→Y
`(ACC) + (Y) -> Z`	ACCout和ALUin有效，CU向ALU发加命令，结果送往Z
`Z -> ACC`	Zout和ACCin有效，结果送往ACC

4. 控制器

4.1. 控制单元功能

4.1.1. 微操作命令分析

1️⃣执行周期典型例子：**加法指令**，两个操作数一个在ACC，一个在主存单元A中

操作	描述
`Ad(IR) -> MAR`	现行指令地址→MAR
`1 -> R`	启动读取命令
`M(MAR) -> MDR`	通过MAR存储的地址→在主存中找到操作数→送入MDR
`(ACC) + (MDR) -> ACC`	ALU收到加命令后，将ACC内容与MDR内容相加，回送给ACC

2️⃣执行周期典型例子：**存数指令**，比如将以上相加结果存到主存单元A中

操作	描述
`Ad(IR) -> MAR`	现行指令地址→MAR
`1 -> W`	启动写命令
`(ACC) -> MDR`	ACC内容→送入MDR

操作	描述
`(MDR) -> M(MAR)`	将MDR的内容写到所指的主存单元中

3️⃣ 中断周期：

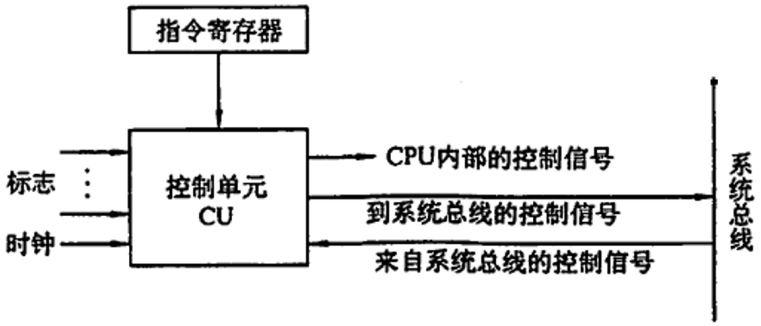
1. 保存现场：假设程序断点(现场)内容保存至主存0单元

操作	描述
`0 -> MAR`	主存0号单元地址→MAR
`1 -> W`	启动写命令
`(PC) -> MDR`	将PC内容(程序断点)→送入MDR
`(MDR) -> M(MAR)`	将MDR的内容写到所指的主存单元中

2. 处理中断：采用硬件向量法寻找入口地址

操作	描述
`向量地址 -> PC`	将向量地址形成部件的输出送至 PC
`0 -> EINT`	关中断，将允许中断触发器清零

4.1.2. 控制单元的功能



1️⃣ 输入CU的内容

输入来源	描述
指令寄存器	将指令的操作码送入CU进行译码。
标志	控制单元根据上条指令的结果来产生相应的控制信号
时钟	通过时钟脉冲来控制每个操作的完成时间和执行顺序
系统控制总线的控制信号	接收中断请求、DMA请求等信号输入

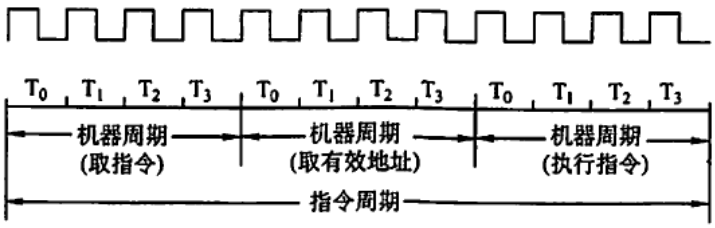
2️⃣ 输出CU的内容

输出目标	描述
CPU内的控制信号	用于CPU寄存器间的传送，控制ALU实现不同的操作

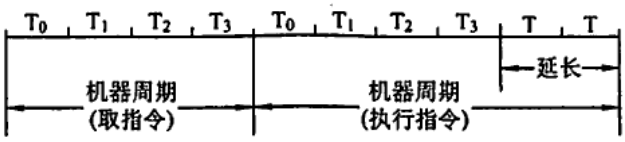
输出目标	描述
系统控制总线	输出信号，让主存或者IO读/写、中断响应等

4.1.3. 控制方式：管理CPU周期和指令执行

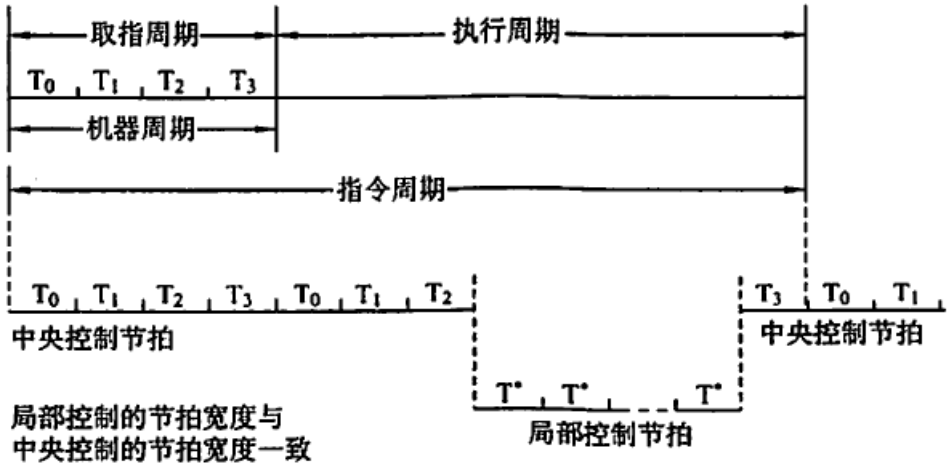
1 同步——定长方式：每个CPU周期内节拍数相同，与所需时长最长的周期对齐



2 同步——不定长方式：每个CPU周期内节拍数可不同，包含数量视指令复杂程度而定



3 同步——中央/局部控制结合方法



1. 指令周期被分为：中央控制节拍+局部控制节拍，二者所包含的节拍长度相同
2. 中央控制：时长短，完成大多简单指令
3. 局部控制：时长长，完成少数复杂指令

4 异步控制：

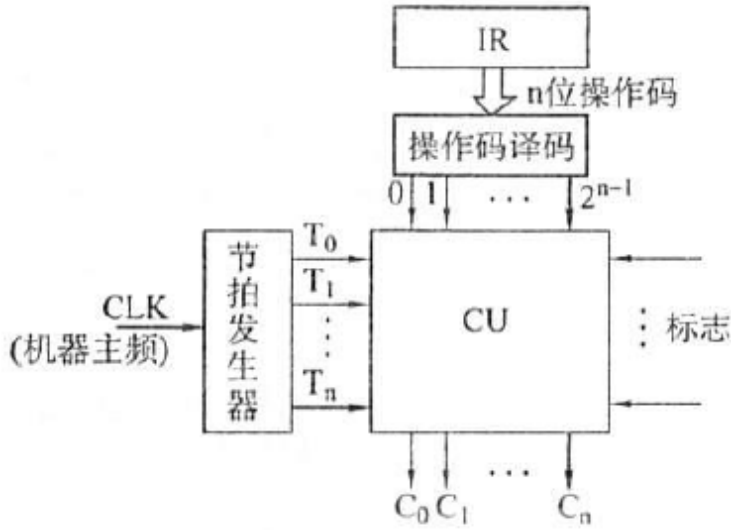
1. 不存在固定节拍/时钟同步
2. CU发出执行微操作信号后，微操作一直执行到完成后回送ACK，CU再发下条执行信号

5 联合控制方式：大部分指令采用同步方式，小部分采用异步

4.2. CU的设计

4.2.1. 组合逻辑设计

组合逻辑控制(硬布线)：由基本门电路组合实现，处理速度快，但是电路复杂



4.2.1.1. 硬布线CU的三种输入信号

1. 指令IR —(指令操作码 / 译码)→ 产生的指令信息 —(输入)→ CU
2. 时序系统 —(节拍发生器)→ 周期信号 —(输入)→ CU
3. 执行单元执行结果 —(修改)→ 标志位 —(输入)→ CU

4.2.1.2. 硬布线CU的微操作

1. 取指周期微操作

操作	描述
`(PC) -> MAR`	待执行指令的地址 —(送入)→ 地址缓冲寄存器
`1 -> R`	发出读命令**(固定写法)**
`M(MAR) -> MDR`	MAR存放的地址 —(在主存中找到)→ 待执行的指令 —(将指令送入)→ 数据缓冲寄存器
`MDR -> IR`	MDR存放的指令 —(塞入)→ 指令寄存器，因此(IR)表示指令本身
`OP(IR) -> CU`	指令的操作码 —(传给)→ 控制单元(译码)
`PC+1 -> PC`	PC+1，指向下一条指令

2. 间接寻址周期微操作

操作	描述
`AD(IR) -> MAR`	指令的(逻辑)地址码$\xrightarrow{\text{传给}}$地址缓冲寄存器
`1 -> R`	发出读取命令
`M(MAR) -> MDR`	MAR存放的逻辑地址$\xrightarrow{\text{在主存中找到}}$有效地址$\xrightarrow{\text{将有效地址送入}}$数据缓冲寄存器

3️⃣ 执行周期微操作

1. 不访问主存的指令

命令	描述	操作
CLA	清空ACC	`0->ACC`
COM	取反	`ACC取反->ACC`
SHR	算术右移	`L(ACC)->R(ACC)，ACC0->ACC0`
CSL	循环左移	`R(ACC)->L(ACC)，ACC0->ACCn`
STP	停机指令	`0->G`

2. 访问主存的指令：加法/存数/取数

指令	地址或数据准备	执行操作
ADD X	`Ad(IR)→MAR，1→RM，(MAR)→MDR`	`(ACC)+(MDR)→ACC`
STA X	`Ad(IR)→MAR，1→W`	`ACC→MDR，MDR→M(MAR)`
LDA X	`Ad(IR)→MAR，1→R，M(MAR)→MDR`	`MDR→ACC`

3. 转移指令

指令	地址或数据准备	执行操作
JMP X	无条件转移	`Ad(IR)→PC`
BAN X	有条件(这里是负数的话)则转移	$A_0 \cdot Ad(IR) + \overline{A_0} \cdot (PC) \rightarrow PC$

4.2.2. 微程序设计

4.2.2.1. 微程序设计概念

1️⃣ 特点：每个机器指令$\xleftrightarrow{\text{对应编写成}}$微程序(存放在CS中)$\xrightarrow{\text{包含}}$多条微指令$\xrightarrow{\text{包含}}$更多的微命令

2️⃣ 控制存储器CS：储存微程序，在CPU内部，用ROM实现

3️⃣ 操作：要执行某指令→从CS中取出微程序→译码产生微命令→执行微命令相关操作

3 特点：CU设计简单，但是速度慢

4.2.2.2. 微程序控制概念

1 微命令&微操作

1. 微操作：计算机执行的原子操作，一条机器指令可分解为系列微操作
2. 微命令：控制单元$\xrightarrow{\text{控制命令aka微命令}}$执行部件，是控制序列的基本单位
3. 二者关系：微操作$\xleftrightarrow{\text{一一对应}}$微命令，微命令是微操作的控制信号，微操作是微命令的执行过程

2 微指令&微周期

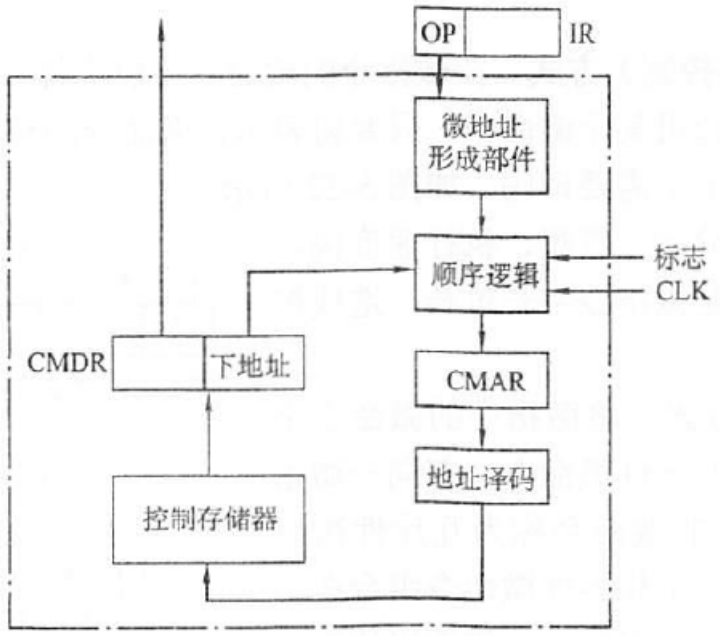
1. 微指令：
   - 概念：多个微命令集合
   - 存储结构：存在CS中，一个微指令对应一个存储单元，存储单元地址称为微地址
   - 组成：操作控制字段(生成执行某步的控制信号)→顺序控制字段(决定下一步执行的步骤)
2. 微周期：从CS读取微指令+执行该微指令的时间

3 机器指令&微指令

1. 机器指令：汇编后形成的，可由CPU直接识别。执行的指令
2. 二者关系：微指令时机器指令的更底层实现，微指令地址可由机器指令操作码生成

4 微程序：微指令的有序集合

4.2.2.3. 微程序CU的基本组成



1 输入输出：输入时钟/标志/指令操作码，将控制信号输出给CPU总线/系统总线

2 控制存储器：核心，存放微程序

3 CMAR/CMDR：类似于主存中的MAR/MDR，缓冲微指令地址/微指令

4 顺序逻辑：控制微指令的序列

4.2.2.4. 微指令格式：<操作控制><顺序控制>

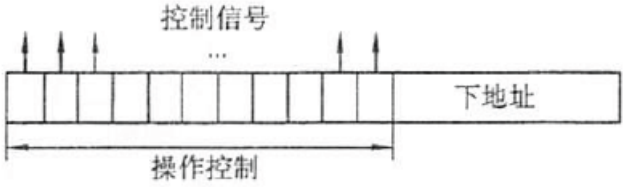
1️⃣操作控制→根据当前指令发出控制信号

2️⃣顺序控制→指出下个指令的地址

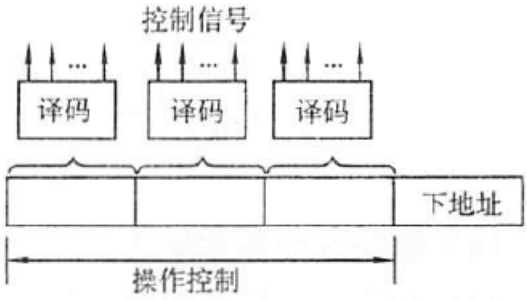
4.2.2.5. 微指令编码：如何编码微指令控制字段，形成控制信号

1️⃣直接编码方式：

1. 含义：操作控制字段中，每一位对应一个微命令，位=1/0表示选择/不选择该微命令
2. 特点：微指令过长，但是实现简单(微指令产生无需译码)

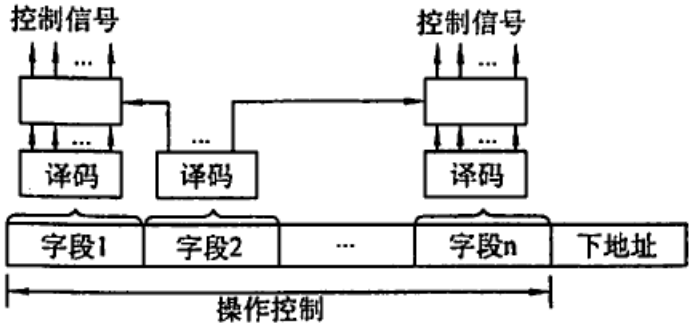


2️⃣字段直接编码方式：



1. 互斥微命令：同一指令周期中不能同时出现的微命令
2. 结构：将操作控制字段分为很多小字段，每个小段单独编码单独译码，代表一个微命令
3. 分段原则
    - 互斥微命令分在同一小字段，相容的却分在不同字段
    - 每小段不能包含太多信息
    - 每个小段定义一个不发出指令的状态，例如3位字段=000表示不操作+7个互斥微命令

3️⃣字段间编码方式(隐式编码)：一字段的某些微命令，可能需另一字段另一些微命令解释



4.2.2.6. 微指令后续地址的生成

1. 断定法：后续指令地址直接由下地址字段给出
2. 通过机器指令操作码形成
   1. 机器指令操作码$\xrightarrow[\text{微地址形成部件}]{\text{译码}}$机器指令微程序首地址
   2. 通过首地址，在微命令寄存器找到对应命令
3. 增量计数器发：类似于PC+1，吓一跳微指令地址通常为CMAR+1
4. 分支转移：
   1. 不可能一直都连续，所以遇到条件转移指令时就会出现分支
   2. 指令格式：`<操作控制字段><转移方式(决定是否转移)><转移地址>`

4.2.2.7. 微指令格式

特性	水平型微指令	垂直型微指令
位字段宽度	宽，同时控制多个微操作	窄，只编码一个/几个微操作
并行性	高	低
占CS的空间	大	小
编码复杂度	复杂，可编码复杂控制策略	简单

5. 指令流水线

5.1. 流水的基本概念

1. 理想情况下，有/无指令流水的时钟对比

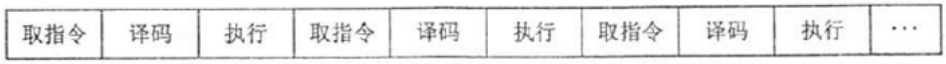


图 5-27　没有采用流水线技术的时钟周期

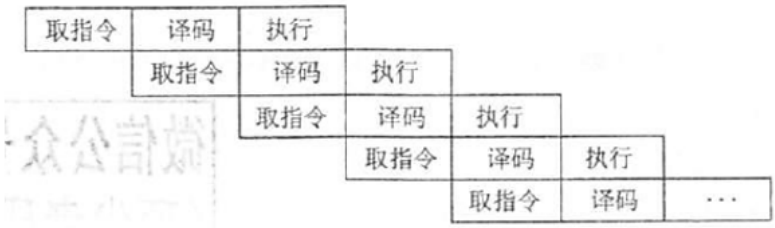
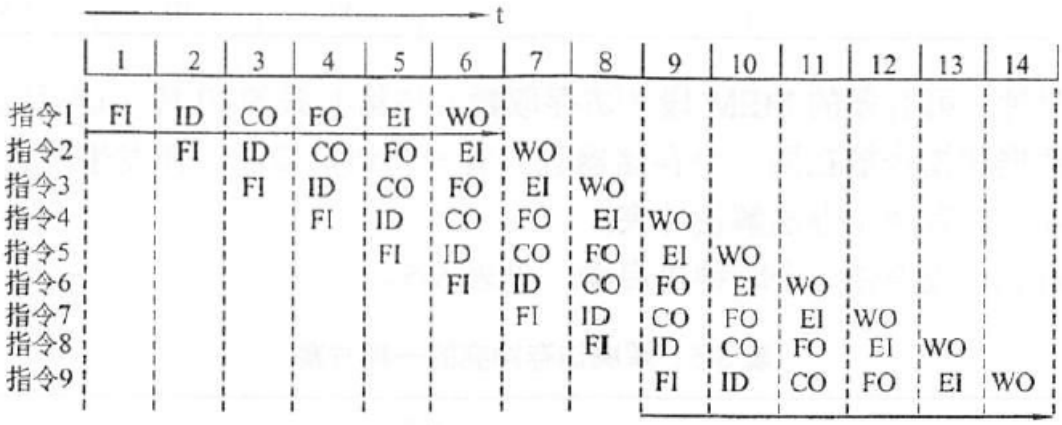


图 5-28　指令流水线示意图

   1. 无流水：指令一条条串行执行，共费时$3N$
   2. 有流水：指令并行执行，共费时$N+2$，当N足够大时一条指令近似只耗时一个时钟周期

2. 一些非理想要素
   1. 指令执行时间>取值时间，所以本指令取指完后，还要等待上条指令执行完
   2. 当遇到条件转移时，要等执行完当前指令，才知道下条指令是什么(根据当前执行结果决定)

3️⃣六级流水线：取指(FI)+译码(ID)+计算操作数地址(CO)+取操作数(FO)+执行指令(EI)+写操作数(WO)



m级流水执行n条指令，共需要$m+n-1$周期

5.2. 指令流水的实现

关键在于解决以下三种相关冲突

5.2.1. 资源相关(结构相关)

1️⃣含义：多条指令进入流水线后，不同指令要在同一周期使用同一功能部件

2️⃣示例：如下是一五级流水线，第四周期中I_1访存取数I_4访存取指令，当数据/指令需要同一内存口访问时，就冲突了

指令	时钟							
	1	2	3	4	5	6	7	8
I_1	FI	ID	EX	MEM	WB			
I_2		FI	ID	EX	MEM	WB		
I_3			FI	ID	EX	MEM	WB	
I_4				FI	ID	EX	MEM	WB
I_5					FI	ID	EX	MEM

3️⃣solution：让I_4停顿一个时钟(整体后移一个时钟)，或增加一个存储器分开存放指令/数据

5.2.2. 数据相关

1️⃣含义：一条指令执行完后，才能执行下一条指令

2️⃣RAW(Read After Write)，WAR，WAW

3️⃣Solution：

1. 简单方法：死等，等上一条读写完，在接下去读写
2. 数据旁路技术：上条指令不再写/下条指令不再读，上条指令结果$\xrightarrow{\text{装用通道}}$下条指令输入

5.2.3. 相关控制

1️⃣诱发因素：转移指令，一局转移条件结果可能执行下条指令/跳转执行新目标地址指令

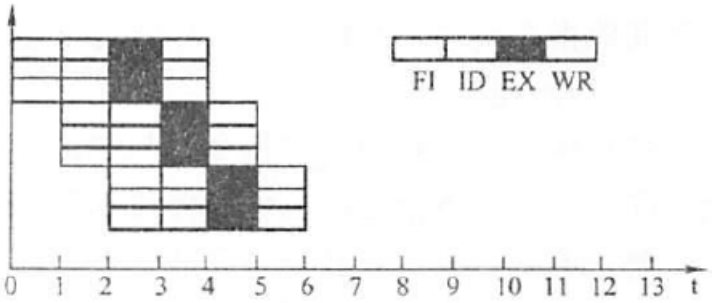
2️⃣Solution：猜测法，从转移的两个分支中先猜一个并按照猜测处理，猜对则流水继续，猜错则预取指令失效

5.3. 流水线性能指标

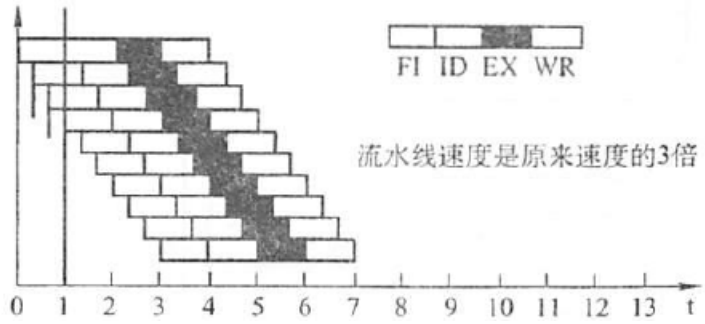
1. 吞吐率：单位时间内流水线完成指令数/输出结果数
2. 加速比：不用流水所用时间 / 用流水后所用时间
3. 流水线效率：$E=\frac{n个任务占时空图的有效面积}{n个任务所用时间与k个流水段所围成的时空区总面积}$

5.4. 流水线的改进

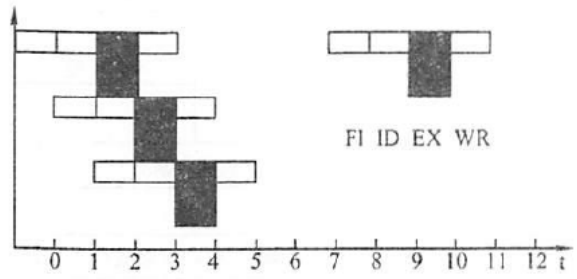
1. 超标量技术：每个时钟周期，每个流水线，可以并发执行多条指令



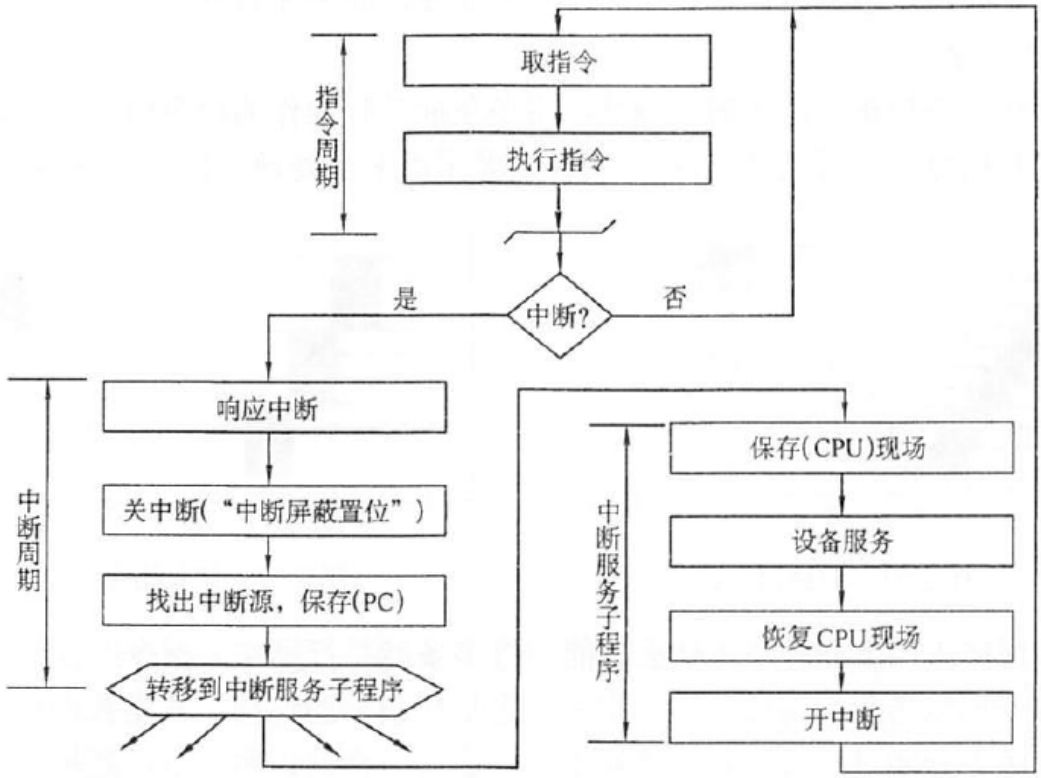
2. 超级流水线：进一步将每部操作细分，使得相邻流水线的进度更靠近



3. 超长指令字：将多条能并行操作的指令，组合成一条超长指令字

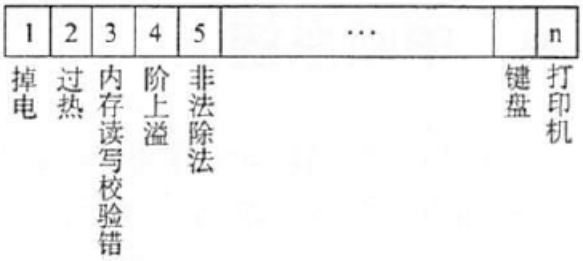


6. 中断系统



6.1. 中断源 响应请求 / 发出请求 CPU过程概述

1 中断请求：中断标记寄存器(INTR)寄存器中一项设为1→对应中断源提出了中断请求



2 CPU查询中断：

1. CPU的一个指令执行周期结束后，就会去查询是否有中断
2. 查询到CPU允许中断触发器(EINT，即关中断)=1，且有中断发出时，CPU才响应中断

3 CPU响应中断：进入中断周期，中断隐指令(基于硬件)完成以下内容

保存断点(PC)→寻找中断服务程序入口→关中断(禁止CPU一个没完就在去响应中断)

4 CPU执行中断

1. 保护现场：隐指令完成断点(PC)保存，中断服务程序完成CPU内各寄存器内容保存
2. 处理中断
3. 屏蔽新的中断请求：
    - 单重中断：要么直接不理会新请求
    - 多重中断：要么按照优先级抢占式调度

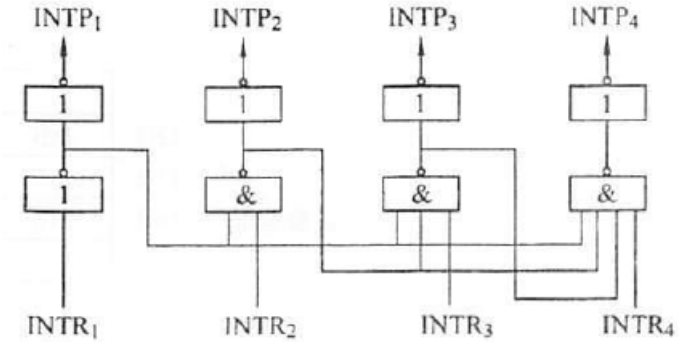
- 屏蔽技术：每个中断源一个屏蔽字，开始处理某一中断时就将其通过屏蔽字屏蔽掉

4. 恢复现场

6.2. 中断判优

1️⃣核心问题：多中断源同时请求，系统只响应一个，中断系统此时会基于优先级，优先响应优先级高的

2️⃣硬件实现：以下电路结构中，只要$INTR_1 = 1$就可以屏蔽右边所有中断，依次类推



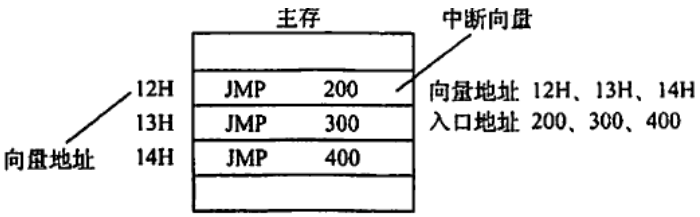
3️⃣软件实现：按优先级高到低挨个问，你中断没？啊你中断了好的我立马响应(没有就去问下一个)

6.3. 如何找到中断服务入口地址

1️⃣Pre：中断操作$\xleftrightarrow{\text{一一对应}}$中断服务程序$\xleftrightarrow{\text{一一对应}}$中断程序入口

2️⃣寻找入口方法1：硬件向量法

1. 中断向量地址形成硬件$\xrightarrow{\text{产生}}$中断向量地址
2. 向量地址$\xrightarrow{\text{找到}}$中断服务程序入口，完成这一过程有两种方法
    - CPU响应中断时，直接把向量地址送给PC，执行命令，得到入口地址
    - 设置向量地址表，放在主存中，存储单元地址=向量地址 / 存储单元内容=入口地址，访问向量地址所指示的存储单元就可获得入口地址



3️⃣方法3：软件查询法