8.1 CPU 的结构

一、CPU 的功能

1.控制器的功能：

（1）取指令；

（2）分析指令；

（3）执行指令，发出各种操作命令；

（4）控制程序输入及输出结果；

（5）总线管理；

（6）处理异常请求和特殊情况。

2.运算器的功能：

实现算术运算和逻辑运算。

总的来说，CPU 有以下功能：

指令控制、操作控制、时间控制、处理中断、数据加工。

二、CPU 结构框图

1.CPU 与系统总线

如图 1.1 所示。

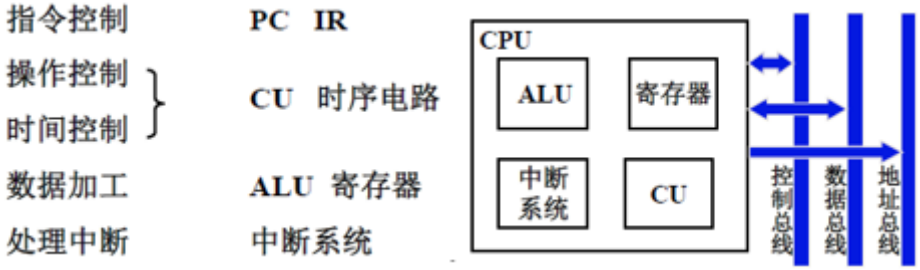

图 1.1 CPU 与系统总线

三、CPU 的寄存器

1.用户可见的寄存器

（1）通用寄存器：存放操作数，可作某种寻址方式所需的专用寄存器，例如 8086 中的 AX、DX 等。

（2）数据寄存器：存放操作数（满足各种数据类型），两个寄存器拼接存放双倍字长数据。

（3）地址寄存器：存放地址，其位数应该满足最大的地址范围，用于特殊的寻址方式，段基址、栈指针。

（4）条件码寄存器：存放条件码，可作程序分支的依据，如正、负、零、溢出、进位等。

2.控制和状态寄存器

（1）控制寄存器

PC->MAR->M->MDR->IR

控制 CPU 操作。其中 MAR/MDR/IR 用户不可见，PC 用户可见。

（2）状态寄存器

状态寄存器用来存放条件码；PSW 寄存器存放程序状态字。

四、控制单元 CU 和终端系统

1.CU 产生全部指令的微操作命令序列。实现方法有组合逻辑设计（硬连线逻辑，速度

快）、微程序设计（存储逻辑，结构简单，适用于比较复杂的地方）。

2.中断系统，后续讲述。

五、ALU

已经在第 6 章讲述。

8.2 指令周期

一、指令周期的基本概念

1.指令周期

取出并执行一条指令所需的全部时间。完成一条指令包括两部分：一是取指、分析（取指周期），二是执行（执行周期）。**不同的 CPU、不同的设计集等对指令周期的分法是不一样的**，这里只是课本上如此说。

2.即使是同一个 CPU 中，每条指令的指令周期也不一定相同。

3.具有间断寻址的指令周期。如图 2.1 上所示。

4.带有间接寻址和中断周期的指令周期，如图 2.1 下所示。

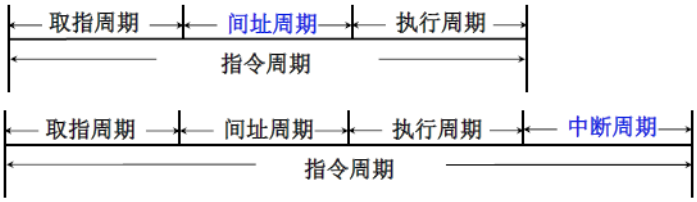

图 2.1 指令周期的图

5.指令周期流程。如图 2.2 所示。

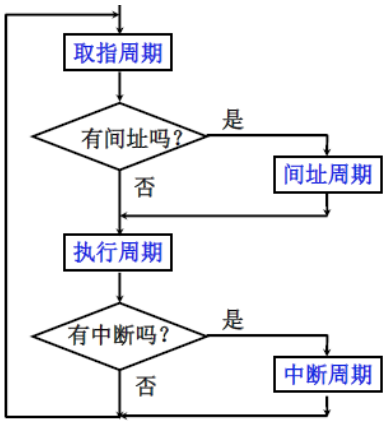

图 2.2 指令周期流程图

首先要取指令，然后判断指令是否采用间址，若采用间址，则进入间址周期，获取操作数所在的地址，然后再到所给的地址去寻找操作数，然后进入执行周期，判断是否有中断，如果有中断则进入中断，否则完成该指令周期。

6.CPU 工作周期的标志

CPU 访存有四种性质：

取指令	取指周期
取地址	间址周期
存取操作数或结果	执行周期

存程序断点	中断周期

二、指令周期的数据流

1.取指周期数据流

如图 2.3 所示。在取指阶段，用 PC，PC 告诉你下一条地址，地址保存在 MAR 中，数据保存在 MDR 中，指令保存在 IR 中，控制部分是 CU 中。

数据流从 PC 开始。PC 把地址传送给 MAR，然后送到地址总线，再送给存储器，存储器知道了地址，CU 把控制信号送到控制总线，然后送到存储器，读到指令送到 MDR，然后送到 IR。CU 执行+1 再传给 PC，为下一条指令做准备（无跳转的情况下）。

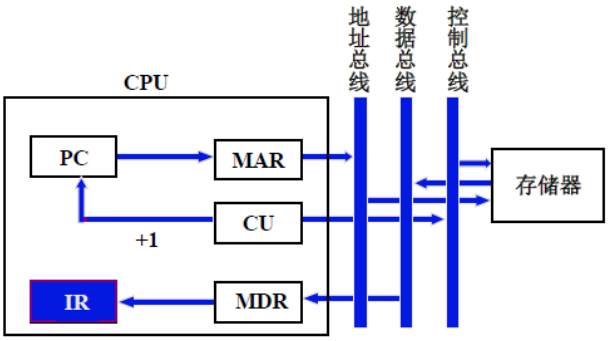

图 2.3 取指周期数据流

2.间址周期数据流

如图 2.4 所示。首先要取出操作数的地址。开始从 IR 或者 MDR 都可以。假设从 MDR 开始，那么先把 MDR 中的地址码给 MAR，然后 MAR 送到地址总线，再送给存储器，CU 发送控制命令给控制总线，然后发送给存储器，开始从存储器读该指令的地址，把数据送到数据总线，然后送到 MDR，这时 MDR 的就保存了我们需要的东西。

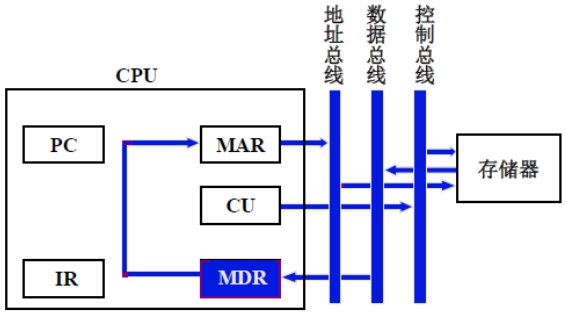

图 2.4 间址周期数据流

3.执行周期数据流：不同的指令的执行周期数据流差异非常大，见第 9 章。

4.中断周期数据流

如图 2.5 所示。在中断周期我们做了保存断点、形成中断服务程序入口地址硬件关中断操作。其中 PC 当前的内容必须保存起来，以待执行完中断服务程序后可以准确返回到该程序的间断处。

图中由 CPU 把用于保存程序断点的存储器特殊地质（如栈指针的内容）送往 MAR，并送到地址总线上，然后由 CU 向存储器发写命令，将 PC 中的当前内容送到 MDR，最终使程序断点经数据总线送入存储器。此外，CU 还需将中断服务程序的入口地址送至 PC，为下一个指令周期的取指周期做好准备。

8.3 指令流水

一、如何提高机器速度

1.提高访存速度：高速芯片、Cache、多体并行。

2.提高 I/O 和主机之间的传送速度：中断、DMA、通道、I/O 处理机、多总线。

3.提高运算器速度：高速芯片、改进算法、快速进位链。

4.**提高整机处理能力：高速器件，改进系统结构、开发系统的并行性**。

二、系统的并行性

1.并行的概念

并行指：①并发：两个或两个以上事件在同一**时间段**发生；②同时：两个或两个以上事件在同一**时刻**发生。

2.并行性的等级

过程级（程序之间、进程之间） 粗粒度 软件实现

指令级（指令之间、指令内部） 细粒度 硬件实现

三、指令流水原理

1、指令的串行执行

取指令 1	执行指令 1	取指令 2	执行指令 2	……

取指令是由取指令部件完成，执行指令是由执行指令部件完成，这种方式，总有一个部件空闲。

2.指令的二级流水

取指令 1	执行指令 1		
	取指令 2	执行指令 2	
		取指令 3	执行指令 3

若**取指令**和**执行阶段**时间上**完全重叠**，指令周期**减半**，速度提高 1 倍。

3.影响指令流水效率加倍的因素

（1）执行时间>取指时间

解决方法：

取指令部件->指令部件缓冲区->执行指令部件

取出来的指令临时放到缓冲区，当执行的指令结束后，再把缓冲区的指令执行。

（2）条件转移指令对指令流水的影响

必须等**上条**指令执行结束后，才能确定**下条**指令的地址，造成时间损失。例如可以采用分支预测的方法等，非常多。

4.指令的六级流水

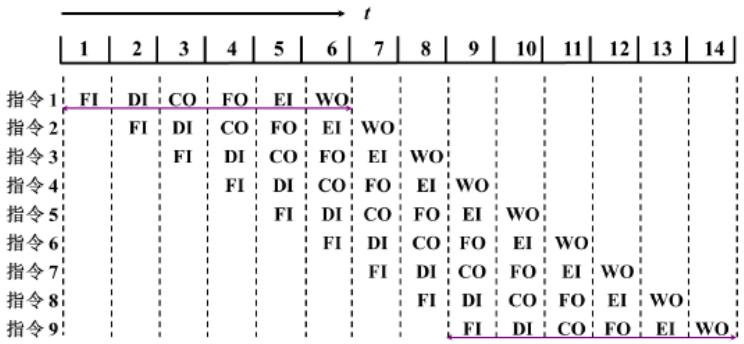

图 3.1 指令的六级流水

如图 3.1 所示，这 6 级指令分别是**取指令、指令译码、生成操作数地址、取操作数、执行**和**结果的写回**。假设 6 级流水线每一段时间都是相同的，如果我们采用串行流水，完成一条指令就需要 6 个时间单位，9 条指令就需要 **54** 个时间单位。但是如果采用 6 级流水，则只需要 **14** 个时间单位。

四、影响指令流水线性能的因素

1.结构相关：不同指令争用同一功能部件产生资源冲突。例如在第 4 个时间单位，指令 1 中的 FO 和指令 4 的 FI 同时访问内存，产生冲突；第 5 个时间段，指令 2 的 FO 操作和第 5 条指令的 FI 会产生冲突；第 6 个时间段，指令 1 的 WO 和指令 3 的 FO 以及指令 6 的 FI 同时访问内存，会产生冲突……

解决方法：（1）停顿，可以指令 4 的冲突指令往后推。

（2）指令存储器和数据存储器分开（现代计算机采用这个方案）。

（3）指令预取技术（适用于访存周期短的情况）。

2.数据相关：不同指令因重叠操作，可能改变操作数的读/写访问顺序。

• 写后读相关（RAW）

SUB R1,R2,R3;　　R2-R3->R1

ADD R4,R5,R1;　　R5+R1->R4

必须完成减后才能进行加，否则的话数据出错。

• 读后写相关（WAR）

STA M, R2;　　R2->M 存储单元

ADD R2, R4, R5;　　R4+R5->R2

必须先把 R2 里面的数据放到 M 后，再去把 R4+R5 结果放在 R2 中，否则数据出错。

• 写后写相关（WAW）

MUL R3,R2,R1;　　R2*R1->R3

SUB R3,R4,R5;　　R4-R5->R3

必须按照顺序执行，否则 R3 中数据会出错。

解决方法：

（1）后推法；

（2）采用**旁路技术**。

3.控制相关

由转移指令引起的。例如：

	LDA #0
	LDX #0
M	ADD X,D
	INX
	CPX #N
	BNE M
	DIV #N
	STA ANS

BNE 指令必须等 CPX 指令的结果才能判断出是转移还是顺序执行。借助图 3.1 来理解，加入指令 3 是转移指令。那么指令 3 在第 3 个时间单位开始执行，第 4 个时间单位指令 4 开始执行，第 5 个时间单位指令 5 开始执行，第 6 个时间单位指令 6 开始执行，第 7 个时间单位指令 7 开始执行。但是在第 7 个时间单位结束的时候，也就是指令 3 执行完成之后，我们才知道要转移到指令 15 执行，那么指令 4 到指令 7 作废，浪费了执行时间。

五、流水线性能

1.吞吐率：**单位**时间内**流水线所完成指令**或**输出结果**的**数量**。

设 m 段的流水线各段时间为 Δt 。我们有两个衡量的指标：

（1）最大吞吐率：没有任何冲突，满负载运行、没有转移指令等时候的吞吐率，公式为：

$$T_{p\max}=\frac{1}{\Delta t}$$

（2）实际吞吐率：连续处理 n 条指令的吞吐率为

$$T_p=\frac{n}{m\cdot\Delta t+(n-1)\cdot\Delta t}$$

2.加速比 Sp

m 段的流水线的速度与等功能的非流水线的速度之比，设流水线各段时间为 Δt ，完成 n 条指令在 m 段流水线上共需

$$T=m\cdot\Delta t+(n-1)\cdot\Delta t$$

完成 n 条指令在等效的非流水线上共需

$$T'=nm\cdot\Delta t$$

则

$$Sp=\frac{T'}{T}=\frac{nm}{m+n-1}$$

3.效率：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**，因此各功能段的设备**不可能**一直处于工作状态。

效率=流水线各段处于工作时间的时空区/流水线各段总的时空区=$\frac{mn\Delta t}{m(m+n-1)\Delta t}$，如图 3.2 所示。

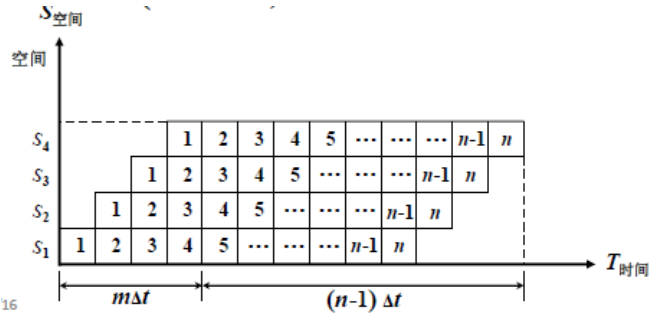

图 3.2 时空图

六、流水线的多发技术

1.超标量技术

每个时钟周期内可**并发多条独立**指令，配置多个功能部件。

不能调整指令的**执行顺序**，通过编译优化技术，把可并行执行的指令搭配起来。

2.超流水线技术

在**一个时钟周期**内再**分段**（例如 3 段），在一个时钟周期内**一个功能部件使用多次**（3 次）。

不能调整指令的**执行顺序**，考编译程序解决优化问题。这里的流水线速度是原来速度的

3 倍。如图 3.3 所示。

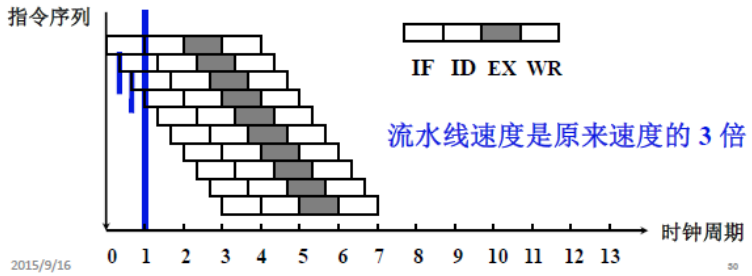

图 3.3 超流水线技术

3.超长指令字技术

由编译程序**挖掘**出指令间**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**（可达几百位），采用**多个处理部件**。如图 3.4 所示。

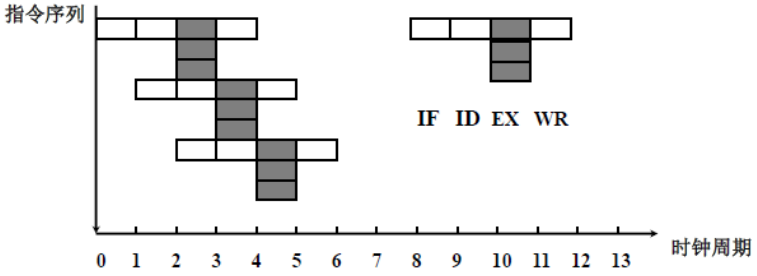

图 3.4 超长指令字技术

七、流水线的结构

1.指令流水线结构

完成一条指令分 6 段，每段需 1 个时钟周期，如下所示：

取指令部件->（锁存）->指令译码部件->（锁存）->地址形成部件->（锁存）->取操作数部件->（锁存）->操作执行部件->（锁存）->回写结果部件

若流水线不出现断流，一个时钟周期就会有 1 个结果。如果不采用流水技术，6 个时钟周期出 1 个结果。

理想情况下，6 级流水的速度是不采用流水技术的 **6 倍**。

2.运算流水线

完成浮点加减运算可分为：**对阶**、**尾数求和**和**规格化**三部分。如图 3.5 所示。注意：分段时每段**操作时间**尽量**一致**。

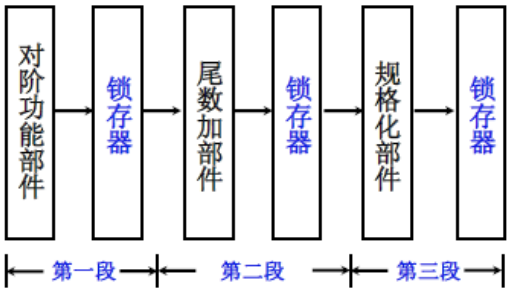

图 3.5 运算流水线

8.4 中断系统

一、概述

1.引起中断的各种因素：

（1）认为设置的中断；（2）程序事故；（3）硬件故障；（4）I/O 设备；（5）外部事件等。

二、中断请求标记和中断判优逻辑

1.中断请求标记：INTR

一个请求源：**一个 INTR** 中断请求标记触发器。

多个 INTR：组成**中断请求标记寄存器**。

INTR **分散**在各个中断源的**接口电路**中。INTR **集中**在 CPU 的中断系统**内**。

2.中断判优逻辑

（1）硬件实现（排队器）

①分散在各个中断源的接口电路中的情况下，**链式排队器**。第五章已经讲解，不再累述。

②集中在 CPU 内。如图 4.1 所示。

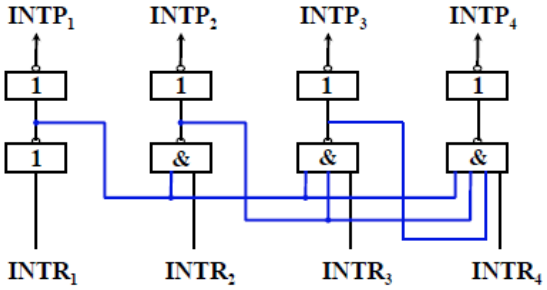

图 4.1 集中在 CPU 内的示例

INTR1/INTR2/INTR3/INTR4 优先级降序排列。

（2）软件实现（程序查询）

A/B/C 优先级按降序排列，程序先查询是否是 A 的请求，是则进入 A 的服务程序；否则查询是否是 B 的请求，是则进入 B 的服务程序；否则查询是否是 C 的请求，是则进入 C 的服务程序，否则继续往下或结束。

三、中断服务程序入口地址的寻找

1.硬件向量法

用硬件的方式找到入口地址。如图 4.2 所示。排队器的输出只有 1 个高电平，其它都是低电平。向量地址形成部件的输入指出了申请的所有中断源中哪一个优先级最高。给出方法有两种，第一种如图 4.2 中图所示，在主存单元中存在一个跳转指令，这个跳转指令跳转到入口地址；第二种如图 4.2 右所示，在主存单元中保存着入口地址，CPU 要取回来，然后再去该地址。

在这个图中，12H/13H/14H 就是向量地址，200/300/400 就是入口地址。

这种方法速度快，但是灵活性低。

图 4.2 硬件向量法

2.软件查询法

假如我们有 8 个中断源，按降序排列。中断识别程序如下（入口地址为 M）

表 4.1 中断识别程序

地址	指令	说明
M	SKP DZ1# JMP 1#SR SKP DZ2# JMP 2#SR …… SK DZ8# JMP 8#SR	1# D=0 跳（D 为完成触发器） 1# D=1 转 1#服务程序 2# D=0 跳 2# D=1 转 2#服务程序 8# D=0 跳 8# D=1 转 8#服务程序

四、中断响应

1.响应中断条件

允许中断触发器 **EINT=1**，且**有中断请求**。

2.响应中断的时间：指令执行周期**结束**时刻由 CPU 发查询信号。

3.中断隐指令（计算机硬件完成，不是在具体指令下完成）

（1）保护程序断点（为将来的中断返回做准备）

方法有以下两种：

①断点存于特定地址（0 号地址）内。

②断点进栈。

保护内容：①保护程序断点；②保护程序运行的软硬件状态

（2）寻找服务程序入口地址

硬件向量法：向量地址直接给 PC；软件向量法：中断识别程序入口地址 M->PC。

（3）硬件关中断

把中断允许触发器置为 0。

单重中断：执行中断服务程序时不允许再发生中断；

多重中断：保护程序软硬件状态的过程中，不允许发生中断。

五、保护现场和恢复现场

1.保护现场：①**断点**：**中断隐指令**完成；②**寄存器**内容：**中断服务程序**完成。

2.恢复现场：**中断服务程序**完成。

如图 4.3 所示，中断服务程序由下面几部分内容组成。

图 4.3 中断服务程序

六、多重中断

1.多重中断的概念（省略）

2.实现多重中断的条件

（1）**提前**设置**开中断**指令。

（2）优先级别**高**的中断源**有权**中断优先级别**低**的中断源。

3.屏蔽技术

（1）屏蔽触发器的作用

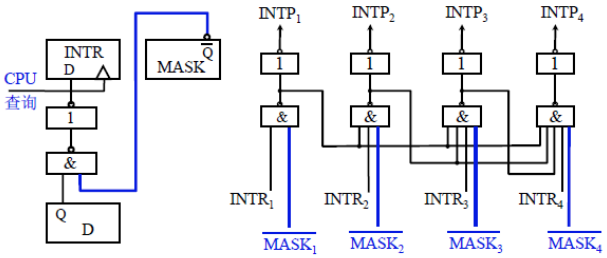

图 4.4 屏蔽触发器

如图 4.4 左图所示，MASK=0（未屏蔽），INTR 能被置 1。若 MASK=1（屏蔽），INTP=0（不能被排队选中）。

图 4.4 右侧是一个排队器和屏蔽触发器组合使用。

（2）屏蔽字

假设 16 个中断源 1,2,3,…,16 按**降序**排列。每个中断源的屏蔽字表示当该中断源执行的时候，是否允许另一个中断源提出的中断服务请求进入到中断排队器排队。如果对应值为 1 则表示屏蔽，0 是允许。

表 4.2 屏蔽字

优先级	屏蔽字															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
……	……															
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

（3）屏蔽技术可改变**处理**优先等级

响应优先级是由硬件电路给出，不可改变；处理优先级可以通过重新设置中断屏蔽字的方式来改变优先级。如表 4.3 所示，响应优先级为 A->B->C->D 降序排列，处理优先级为 A->D->C->B 降序排列。

表 4.3 修改优先等级示例

中断源	原屏蔽字	新屏蔽字
A	1 1 1 1	1 1 1 1
B	0 1 1 1	0 1 0 0
C	0 0 1 1	0 1 1 0
D	0 0 0 1	0 1 1 1

在修改之前，CPU 执行程序轨迹如图 4.5 所示，按照次序完成，没有什么可以解释的。修改之后 CPU 执行程序轨迹如图 4.6 所示。首先 A 中断，然后进入 B 中断，B 并不能屏蔽其他任何中断，突然这个时候来了 C 中断，就需要响应 C 中断，但是 C 中断不能屏蔽 A 和 D 中断，这个时候中断 D 又来了，就去响应 D 中断，D 处理完成后回去处理 C，C 处理完成后回去处理 B，直到最后完成。

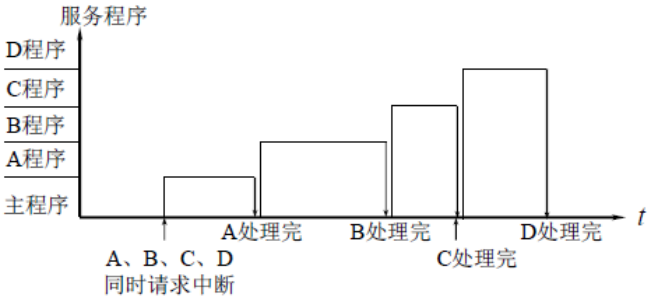

图 4.5 修改前 CPU 执行轨迹

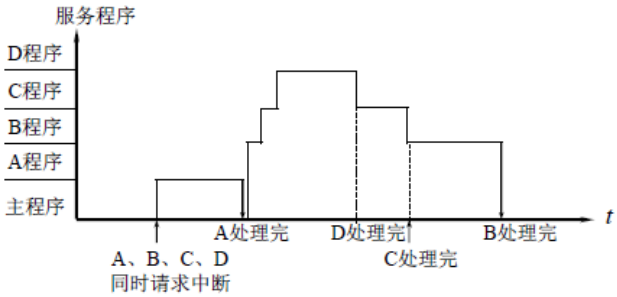

图 4.6 修改后 CPU 执行轨迹

（4）屏蔽技术的其他作用

可以**人为地屏蔽**某个中断源的请求。

（5）新屏蔽字的设置

如图 4.7 所示。

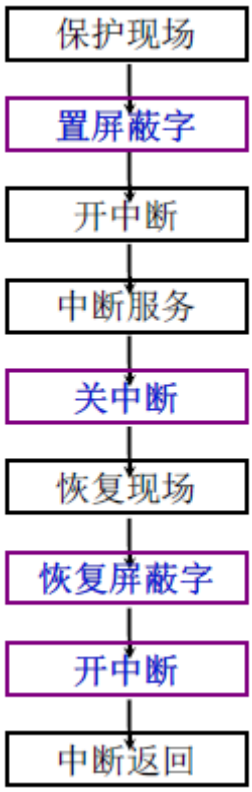

图 4.7 新屏蔽字设置

4.多重中断的断点保护

两种方法：

（1）断点进栈（**中断隐指令**完成）；

（2）断点存入“0”地址（**中断隐指令**完成）。

过程如下：0（假设断点地址是这里）->MAR，命令存储器写，然后 PC->MDR（PC 中保存断点地址），MDR 中的断电地址再保存到主存储器当中。

多次中断，多个断点都存入“0”地址，如何保证断点不丢失呢？如（3）所示。

（3）程序断点存入“0”地址的断点保护

如图 4.8 所示。0 地址是原来存程序断点的地方，然后到地址 5（向量地址），跳转到 SERVE（中断服务程序的入口地址），下面进行保存现场。在多重中断的第 2 个中断来之前，把 0 地址内转存到 RETURN 中，然后开中断，进行其他服务内容。然后恢复现场，然后间址返回。**设置中断屏蔽字必须在开中断之前。恢复中断字必须在 JMP 之前**。

地　址	内　容	说　明
0	××××	存程序断点
5	JMP　SERVE	5 为向量地址
SERVE	STA　SAVE	保护现场
	⋮	
	LDA　0	} 0 地址内容转存
置屏蔽字	STA　RETURN	
	ENI	开中断
	⋮	} 其他服务内容
	LDA　SAVE	恢复现场
	JMP　@　RETURN	间址返回
SAVE	××××	存放 ACC 内容
RETURN	××××	转存 0 地址内容

图 4.8 多重中断“0”地址的保护