原书第十六章

第二十章

控制器操作

20.1微操作

取指周期

间接周期

中断周期

执行周期

指令周期

20.2处理器控制

功能需求

控制信号

控制信号举例

处理器内部的组织

Intel 8085

20.3 硬布线实现

控制器输入

控制器逻辑

20.4 关键词、思考题和习题

学习目标

学习本章后，你应该能够：

解释微操作的概念，并根据此概念定义主要指令周期阶段。

讨论如何组织微操作来控制处理器。

了解硬布线控制单元组织。

我们在第12章讲过，机器指令集很大程度上决定了CPU的功能。若我们知道了机器指令集，包括理解每种操作码的效果，理解寻址方式，以及明确了用户可见的寄存器组，也就知道了CPU必须完成的功能。但事情还不止于此。我们仍必须理解外部接口，即总线，还得要知道中断是如何处理的。根据以上的推断，可以定义CPU所需处理的事项如下：

（1）操作（操作码）

（2）寻址方式

（3）寄存器组

（4）I/O模块接口

（5）内存模块接口

（6）中断

这个列表尽管很普通，但是相当完整。第（1）~（3）项是由指令集定义的；第（4）~（5）项一般是由系统总线定义的；第（6）项部分由系统总线定义，部分由CPU对操作系统的支持类型所定义。

这个6项的列表可以看作是对CPU的功能要求，它们定义了CPU必须做什么。这些也是本书第二部分和第四部分的主要内容。现在我们目光聚焦转到这些功能是如何完成的，更具体地讲就是CPU的各个部件是如何受控以提供这些功能的。下面讨论控制CPU操作的控制器。

**20．1微操作**

我们已经看到在执行程序时，计算机操作是由一系列指令周期组成，每个周期执行一条机器指令。当然因为存在着分支指令的缘故，应记住这个指令周期顺序没必要等同于程序的指令编写顺序（written sequence）。这里我们所说的顺序指的是指令执行的时间顺序（time sequence）。

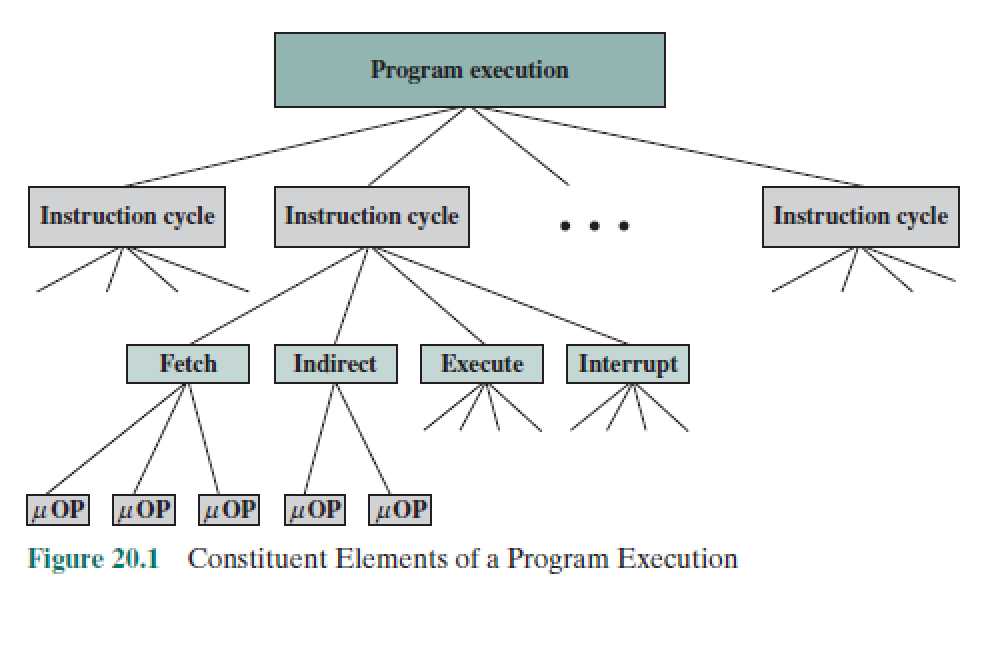
进而我们可以看到，每个指令周期又可以看作是由几个更小的单位组成。一种常见方法是将指令周期分解为取指、间接、执行和中断，其中取指周期和执行周期是必有的。

实际为了设计控制器仍需将此描述进一步向下展开。在第14章讨论流水线时，我们已看到这么做的可能性。实际我们将会看到，每个较小周期又由一系列涉及CPU寄存器操作的更小步骤组成，人们把这些步骤称为**微操作**（micro-operation）。“微”是指这些操作很微小、很简单。图20-1描述了前面讨论的各个概念之间的关系。总而言之，一个程序的执行是由指令的顺序执行组成。每条指令的执行是一个指令周期，每个指令周期由更短的子周期（如取指、间接、执行、中断）组成。每个子周期的完成又涉及一个或多个更短的操作，即微操作。

微操作是CPU最基本的操作，或者说是原子层面的操作。本节将考察微操作，以理解任一指令周期的事件是如何被描述成这样的微操作序列。我们将使用一个简单的例子来说明。本章的其余部分用于说明微操作概念如何应用于控制器设计。

**20．1．1取指周期**

首先了解取指周期，它出现在每个指令周期的开始，从存储器中取出指令。为便于讨论，我们假设使用的是图14-6所描述的组织。它涉及4个寄存器。



**·存储器地址寄存器**（memory address register，MAR）：连接到系统总线的地址线，指定了读、写操作的内存地址。

**·存储器缓冲寄存器**（memory buffer register， MBR）：连接到系统总线的数据线，存放将被存入内存的值或最后一次从内存读出的值。

**·程序计数器**（program counter，PC）：保存待取的下一条指令的地址。

**·指令寄存器**（instruction register，IR）保存最后一次取来的指令。

下面从对CPU寄存器影响的角度来了解取指周期的事件顺序。图20-2是给出一个例子。取指周期开始时，下一条将被执行的指令的地址存放在程序计数器中；此例中的地址是1100100。第一步是将此地址送到存储地址寄存器，因为这是与地址总线相连的唯一寄存器。第二步是装入指令。所要求的地址（在MAR中）被传到地址总线上，控制器发出一个读（READ）命令到控制总线上，于是结果出现在数据总线上并复制到MBR内。我们需对PC递增一个指令的长度，以便为取下一条指令做准备。因为这两个动作（由内存读一个字和递增〕彼此不相干，故可同时完成以节省时间。第三步是将MBR的内容传送到指令寄存器，这也释放了MBR，使其可用于下面可能有的间接周期。

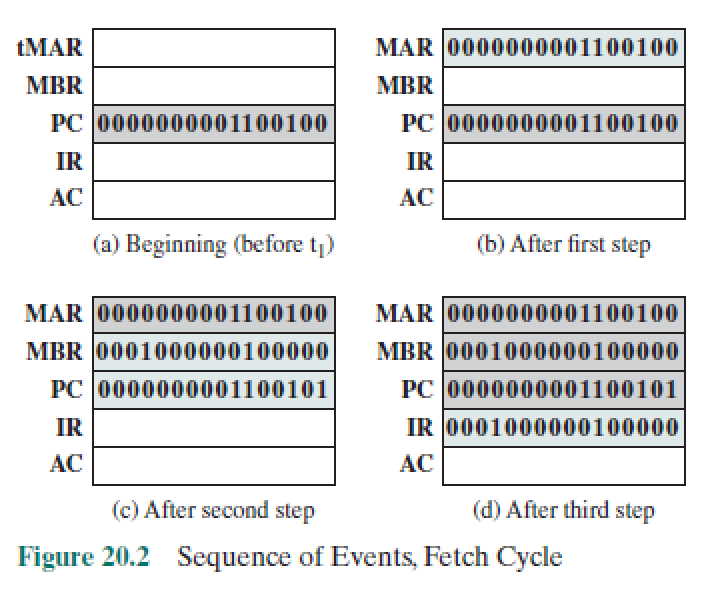
故简单的取指周期实际上由三步和四个微操作组成。每个微操作涉及数据在寄存器之间的传送。只要这些传送彼此互不干扰，那么这几个微操作就可在一步之内同时完成。下面用符号描述此事件顺序。

t1: MAR ← （PC）

t2: MBR ← Memory

PC ← （PC） + *I*

t3: IR ← （MBR）



这里*I*表示指令长度。我们需要对这个操作序列做些解释。为了定时，假定有时钟装置可用，它发出等距的时钟脉冲，每个时钟脉冲定义一个时间单位。于是所有的时间单位都是等长的。每个微操作都能在一个时间单位内完成。（t1, t2, t3）代表连续的时间单位。换句话说，我们有：

·第1个时间单位： PC内容传送到MAR

·第2个时间单位： 被MAR指定的内存中的内容存放到MBR中，PC递增*I*

·第3个时间单位： 传送MBR的内容到IR。

注意，第2个和第3个微操作都是在第2个时间单位同时发生的。第3个微操作也能与第4个微操作组合在一起，并且不影响取指操作。

t1: MAR ← （PC）

t2: MBR ← Memory

t3: PC ← （PC） + *I*

IR ← （MBR）

微操作的分组必须遵守下面两个简单的原则：

（l）事件的流动顺序必须是恰当的。于是，（MAR←（PC））必须先于（MBR←内存），因为内存读操作要使用MAR中的地址。

（2）必须避免冲突。不要试图在一个时间单位里去读、写同一个寄存器，否则结果是不可预料的。例如，（MHR←内存）和（IR←MBR）这两个微操作不应出现在同一时间单位里。

最后值得注意的是，如果有一个微操作涉及加法运算，为避免电路的重复，这个加法应通过ALU完成。根据ALU的功能和CPU的组织，这个ALU的使用可能引起另外的微操作。我们把这个问题放在本章后面讨论。

将这里和下面所讨论的事件与图3-5进行对比会有所裨益。图3-5未涉及微操作，这里的讨论表示了完成指令周期的子周期所需的操作。

**20．1．2间接周期**

取到指令后的下一步是取源操作数。继续使用上述简例，假设使用单地址的指令格式并且支持直接与间接寻址方式。若指令指定是间接寻址，则在指令执行前有一个间接周期。数据流与图14-7有所不同，它包括下述微操作：

t1: MAR ← (IR(Address))

t2: MBR ← Memory

t3: IR(Address) ← (MBR(Address))

指令的地址字段被传送到MAR，再被用于读取操作数的地址。最后，MBH修改IR的地址字段，于是它现在容纳的是操作数的直接地址而不是间接地址。

现在，IR的状态与不使用间接寻址方式时的状态是相同的，并且它已为执行周期准备就绪。先将执行周期的讨论放在一边，下面考虑中断周期

**20．1．3中断周期**

在执行周期完成时，要进行测试以确定是否有允许的中断产生。若有，则出现一个中断周期。这个周期的特性在不同的机器上差异很大。我们给出一个很简单的事件序列，正如图14-8所示，其操作步骤为：

t1: MBR ← (PC)

t2: MAR ← Save\_Address

PC ← Routine\_Address

t3: Memory ← (MBR)

在第一步，PC里的内容被传送到MBR，这样它可作为中断返回地址而保存起来。然后，把MBR中PC内容将要被保存到的内存地址装入到MAR，同时把中断处理子程序的起始地址装入PC。这两个操作可以是单一的微操作。然而大多数CPU提供了多种或多级中断，故在保存地址和子程序起始地址被分别装入PC和MAR中之前，可能会使用一个或多个另外的微操作来取得这两个地址。在任何情况下，一旦得到了这两个地址，最后一步是将包含PC旧值的MBR装入内存。现在CPU为下一个指令周期的开始做好准备了。

**20．1．4执行周期**

取指、间接和中断周期是简单并可预先确定的。每个包括一系列小的、固定序列的微操作，并且每当某周期出现时其相应的一组微操作就被重复一次。

但执行周期不是这样。因为有不同的操作码，就可能会出现不同的微操作序列。让我们来思考几个假设的情况。

首先，考虑一条加法指令；

ADD R1, X

它将存储器x位置的内容加到寄存器R1。该加法指令可能产生如下的微操作序列：

t1: MAR ← (IR(address))

t2: MBR ← Memory

t3: R1 ← (R1) + (MBR)

开始时，IR中装有ADD指令。第一步是将IR的地址部分装人到MAR，然后读出被引用到的存储器位置中的内容。最后，由ALU将R1和MBR的内容相加。同样，这是一个简化的例子。如果从IR中提取出寄存器的引用，以及用某个中间寄存器对ALU的输入和输出进行暂存等，都可能要求另外的微操作。

让我们再来看两个更复杂的例子。一个常用的“递增，若为0则跳步”的指令：

ISZ X

X位置的内容递增1，若结果是0，则跳过下一条指令。可能的微操作序列为：

t1: MAR ←(IR(address))

t2: MBR ←Memory

t3: MBR ←(MBR) + 1

t4: Memory ←(MBR)

If ((MBR) = 0) then (PC ← (PC) + I)

这里引入的新特点是条件操作。若(MBH)=0，则PC递增一个指令长度。这个测试操作可作为一个微操作来实现。还要注意，这个微操作能与将MBR中的修改值写回内存的微操作同时完成。

最后，考虑子程序调用指令。举例，考虑一个“转移并保存地址”指令：

BSA X

此BSA指令之后的指令地址被保存于x位置中，并由x+1位置继续执行。这个被保存的地址稍后会用于返回。这种指令提供对子程序调用直截了当的支持。对应的微操作是：

t1: MAR ← (IR(address))

MBR ← (PC)

t2: PC ← (IR(address))

Memory ← (MBR)

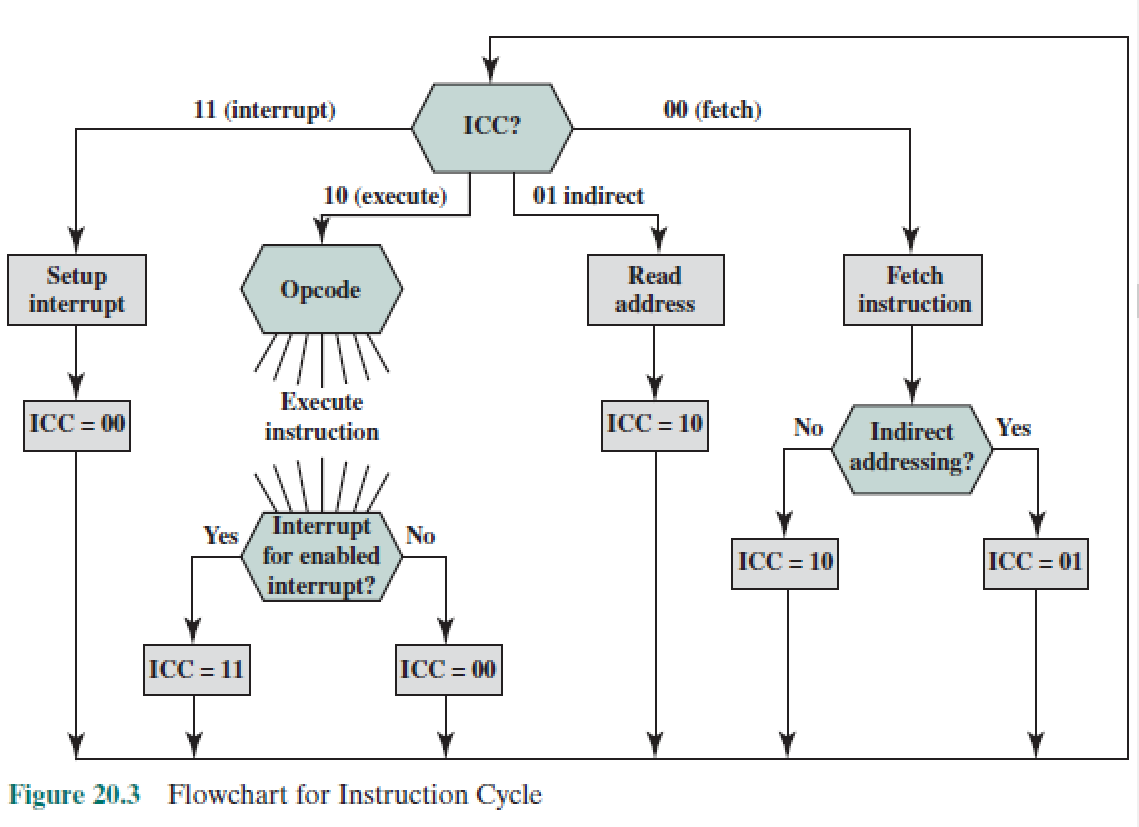
t3: PC ← (PC) + I

指令开始时PC中的地址是下一条指令的地址。它被保存在IR指定的地址位置中此后，PC中的地址也递增I，以提供下一指令周期的指令地址。

**20．1．5 指令周期**

我们已明确指令周期的每个阶段都可分解为一系列的微操作。在本例中，取指、间接、中断周期都各有一个序列，而对于执行周期则是每一操作码有一个序列。

为完善此模型，需要将微操作序列连接在一起，如图20-3所示。我们假设了一个2位的新寄存器，叫做指令周期代码（Instruction Cycle Code，ICC）此ICC定义了CPU处于周期哪一部分的状态：



00：取指

01：间接

10：执行

11：中断

这4个周期每个结束时会对ICC进行相应的设置。间接周期之后总是执行周期，中断周期之后总是取指周期（参见图14-4)。而执行周期和取指周期之后应是什么周期，这取决于系统的状态。

于是，图20-3的流程图定义了微操作的完整顺序，它仅取决于指令序列和中断模式。当然，这是一个简化的例子，实际的CPU流程会更复杂。但无论如何，我们的讨论已说明：CPU的操作被定义为执行一个微操作序列。现在，我们可以来考虑控制器如何发起这个序列的执行。

**20．2．1功能需求**

根据前面的分析结果，可将CPU的行为（功能）分解为称作微操作的基本操作。我们下一个目标是：确定控制器的特性。通过把CPU的操作分解到它的最基本级，就能严格定义控制器必须引起什么动作发生，从而就可以定义控制器的功能需求（functional requirements），即控制器必须完成的功能。这些功能需求的定义是设计和实现控制器的基础。

根据以上信息，如下三步过程能表征控制器；

（1）定义CPU的基本元素。

（2）描述CPU完成的微操作。

（3）确定为了使微作完成，控制器必须具各的功能

至此，已完成第1步和第2步，让我们总结其结果。首先，CPU的基本功能元素有：ALU、寄存器组、内部数据通路、外部数据通路、控制器。

下述思考应该可使读者信服，这是一个完整的列表。ALU是计算机的功能精髓。寄存器组用于CPU内的数据存储。某些寄存器包含用于管理指令顺序执行所需的状态信息（如程序状态字〕，其他寄存器含有来自或去往ALU、内存或I/O模块的数据。内部数据通路用于寄存器之间或寄存器与ALU之间的数据传输。外部数据通路用于将寄存器连接到内存和I/O模块，这通常借助于系统总线。控制器引起CPU内的操作发生。

程序执行由涉及这些CPU元素的操作组成。正如我们已看到的，这些操作由微操作序列组成，回顾20.1节可知，所有的微操作可按如下分类；

·在寄存器之间传送数据。

·将数据由寄存器传送到外部接口（如系统总线）。

·将数据由外部接口传送到寄存器。

·将寄存器作为输入、输出，完成算术或逻辑运算

完成一个指令周期所需的各种微操作，包括执行指令集中任何指令的微操作，都属于上述类型。

现在我们能更明确地说明控制器的功能方式，即控制器完成两项基本任务：

**·定序**(sequencing)：根据正被执行的程序，控制器使CPU以恰当的顺序一步步通过一系列微操作。

**·执行**（execution)：控制器使每个微操作得以完成。

以上是控制器所完成任务的功能描述，控制器如何实现这些功能的关键是对控制信号的使用。

**20．2．2控制信号**

我们已经定义了CPU的组成(ALU、寄存器、数据通路等）及其完成的微操作。为使控制器实现其功能，就必须提供能让它确定系统状态的输入和能让它控制系统行为的输出。这些是控制器的外部规范。至于内部规范，控制器必须包含完成它的定序和执行功能的逻辑。我们把控制内部操作的讨论留到20.3节和第21章.本节的剩余部分用于讨论控制器和CPU其他元素的交互作用。

图20-4是一个控制器的一般模型，图中显示了控制器的所有输入和输出。输入是：

**·时钟**：这是控制器何以“遵守时间”的关键。控制器要在每个时钟脉冲完成一个（或一组同时的）微操作。这有时称为处理器周期时间（processor cycle time）或时钟周期时间（clock cycle time)。

**·指令寄存器**：当前指令的操作码和寻址方式用于确定在执行周期内完成何种微操作

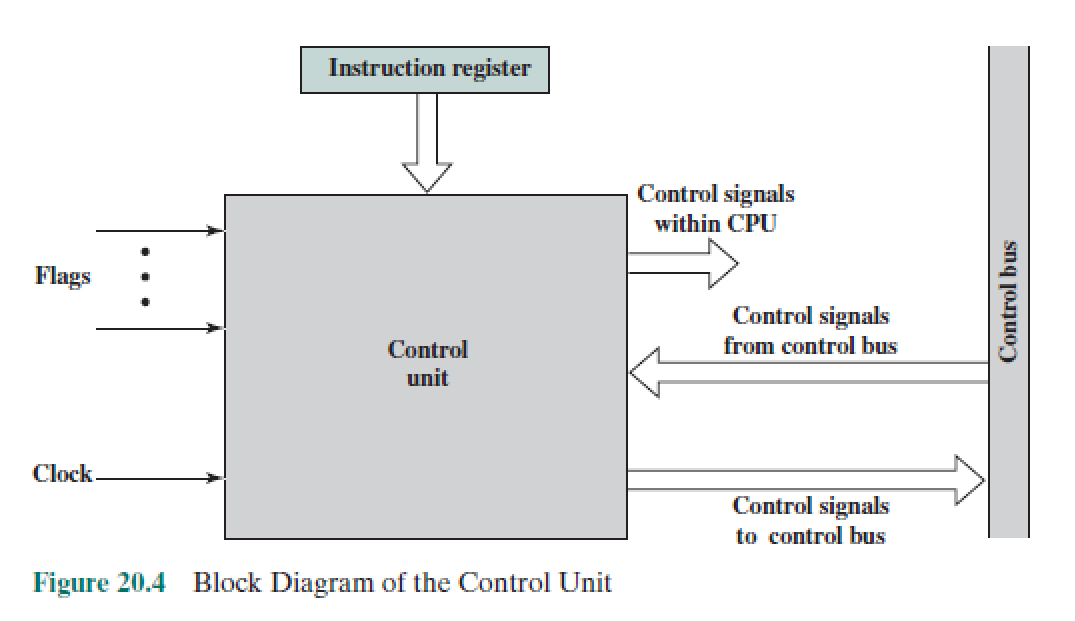
**·标志**: 控制器需要一些标志来确定CPU的状态和前一个ALU操作的结果。例如，对那个“递增，若为0则跳步”（ISZ）指令来说，控制器将依据零标志位是否置位来决定是否递增一个指令长度。

**·来自控制总线的制信号**：系统总线的控制线部分向控制器提供的控制信号。

输出是：

**·CPU内的控制信号**：两类，一类用于寄存器与其他寄存器之间传送数据，另一类用于启动特定的ALU功能

**·到控制总线的控制信号**：这亦有两类，即到存储器的控制信号和到I/O模块的控制信号。



控制信号可分为三类，它们分别是：启动ALU功能的；控制数据路径的；外部系统总线上的或其他外部接口上的控制信号所有这些信号最终作为二进制输入量直接输入到各个逻辑门上。

让我们再次考察取指周期，看看控制器如何维护控制。控制器保持着当前处于指令周期何处的记录。在一个给定时间点，控制器知道下面将要完成的是取指周期。第一步是传送的内容到MA控制器完成这个任务是通过发出控制信号，打开各位与MAR各位之间的门。下一步是由存储器读一个字装人MHR并递增PC。控制器通过发出如下控制信号来完成这个任务。

．控制信号打开逻辑门，以便允许MAR的内容送到地址总线上。

·存储器读控制信号送到控制总线上。

·控制信号打开逻辑门，允许数据总线上的内容存入MBR。

·控制信号对内容加I（指令长度〕并把结果存回PC。

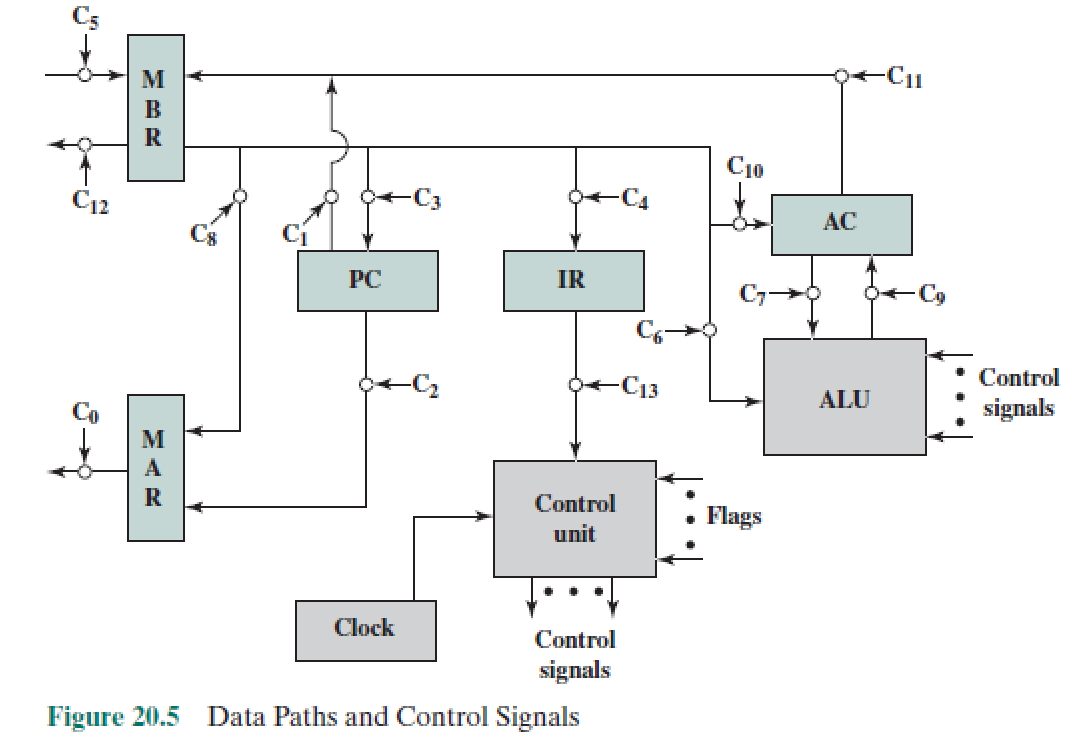
接着，控制器发出打开MBR和IR之间门的控制信号。

这就完成了取指周期，除了一件事：控制器必须判断下面是要完成一个间接周期还是要完成一个执行周期。为此，它要检查IR，看看此指令是否要进行间接存储器访问。

间接周期和中断周期的工作类似。对于执行周期，控制器要检查指令的操作码，并由此判断此周期要完成什么微操作。

**20．2．3控制信号举例**

为说明控制器的功能，让我们考察图20-5所示的例子。这是具有单一累加器（AC）的简单CPU，图中显示了部件之间的数据通路。控制器发出信号的控制通路未画出，但控制信号的终端用一个小圆圈指示并标记有Ci。此控制器接受来自时钟、指令寄存器和标志的输入。每个时钟周期，控制器读取所有的这些输入并发出一组控制信号。控制信号分别送往三个目标。

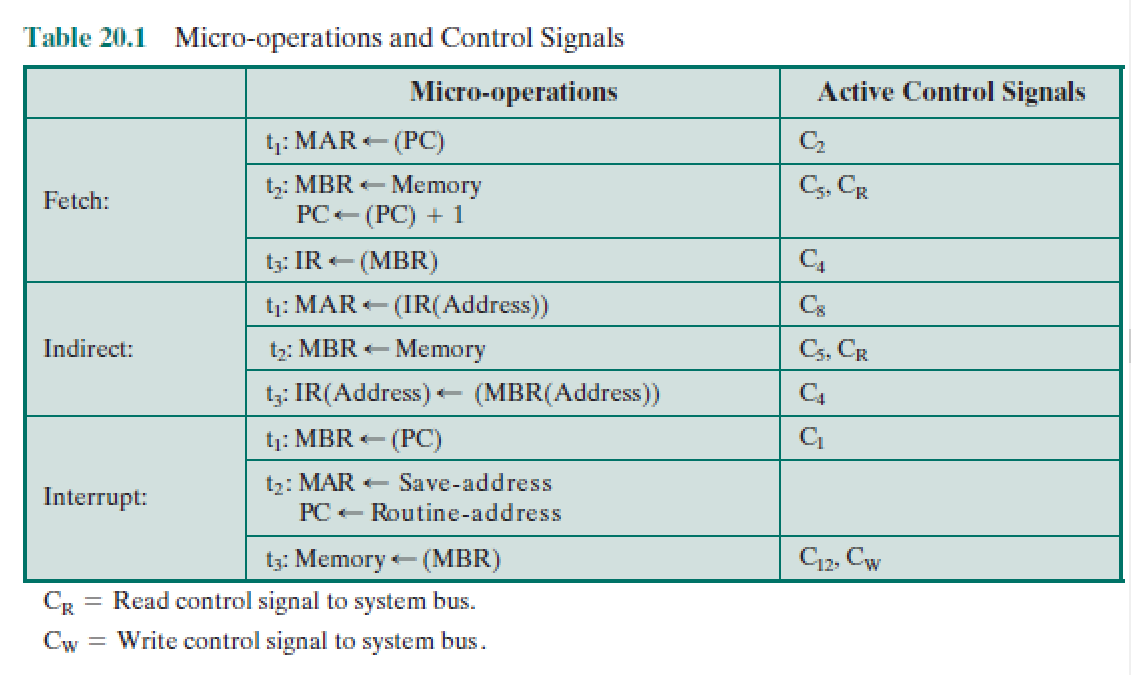


**·数据通路**：控制器控制CPU内部的数据流。例如，取指令时，MBR的内容传送到IR。为了能控制每条通路，通路上都有逻辑门（图中以圆圈表示〕。来自控制器的控制信号可临时打开逻辑门让数据通过：

**·ALU**：控制器以一组控制信号控制ALU的操作。这些信号作用于ALU内的各种逻辑电路和门。

**·系统总线**：控制器发送控制信号到控制总线上（如存储器读信号）。

控制器必须总是知晓它处于指令周期的什么阶段。利用这一信息，并通过读取所有的输入，控制器发送一系列的控制信号使微操作得以产生。控制器使用时钟脉冲确定事件顺序，允许事件之间有一定的时间间隔以使信号电平得以稳定。表20-1表示的控制信号，是前面描述过的某些微操作序列所需的。为了简化，递增的数据和控制通路，以及固定地址装入PC和MAR的数据和控制通路没有给出。



考虑控制器的最小特性是有益的。控制器是整个计算机运行的引擎。它只需要知道将被执行的指令和算术、逻辑运算结果的性质（例如，正的、上溢等）。它不需要知道正被处理的数据或得到的实际结果具体是什么。并且它控制任何操作只是经由少量的送到CPU内的和送到系统总线上的控制信号来实现

**20．2．4处理器内部的组织**

图20-5中使用了不同的数据通路，可以更清楚地观察到这类组织的复杂性。更典型的例子是使用某种内部总线的组织结构，如图14-2所推荐的那样.

使用一个CPU的内部总线。可将图20-5重安排成如图20-6所示，ALU和所有的CPU寄存器都连接到单一的内部总线上。内部总线和寄存器之间有门和控制信号，使得数据可由各寄存器传送到此总线上，或从此总线上被接收。另外的控制线控制着数据和系统（外部）总线的交换以及ALU的操作。

图中所示的内部组织已添加了两个新寄存器，分别标记为Y和Z，这是ALU的一些相应操作所必需的。当ALU的操作涉及两个操作数时，一个可由内部总线得到，但另一个必须要从另外的源得到。累加器AC能实现这个目的，但这限制了系统的灵活性。而且对于有多个通用寄存器的CPU就不能使用累加器的方案。寄存器Y为另一个输入提供了暂时存储。ALU是一个组合逻辑电路（见第11章），其内部无存储电路。这样，当控制信号激活ALU的某个功能时，ALU输入通过ALU的组合逻辑电路被转换为ALU的输出。因此ALU的输出不能直接连到内部总线上，因为这个输出又会被反馈为输入。为此，寄存器Z为这个输出提供了暂时存储。根据如此安排，把存储器的值加到AC的操作将有如下步骤：

t1: MAR ← (IR(address))

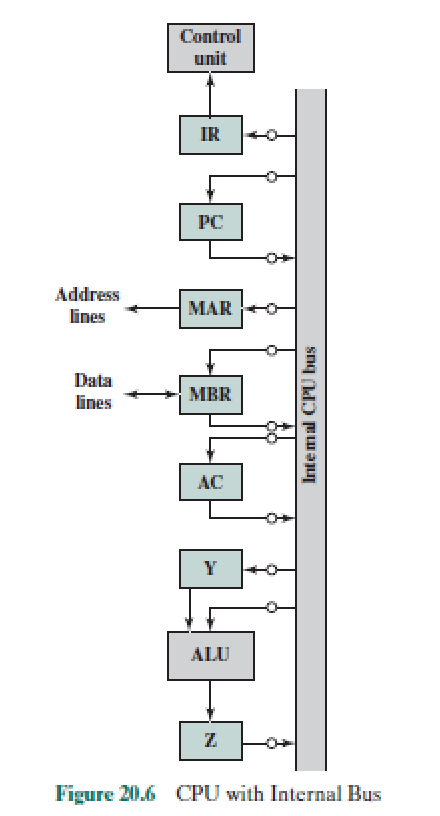
t2: MBR ← Memory

t3: Y ← (MBR)

t4: Z ← (AC) + (Y)

t5: AC ← (Z)

其他的组织形式也是允许的，但通常是要使用某种内部总线或一组内部总线。使用公共数据通路，简化了互连布局和CPU的控制。使用内部总线的另一实际理由是节省芯片面积。



**20．2．5 Intel 8085**

为说明本章至此所介绍的这些概念，让我们考虑Intel 8085，它的内部结构如图20-7所示。几个关键的组件需要稍作解释。

**·递增/递减地址锁存**（incrementer/decrementer address latch）：这是能对栈指针或程序计数器的内容进行加1或减1的逻辑。通过避免了为这些加减法运算使用ALU节省时间。

**·中断控制**(interrupt control)：此模块管理多级中断信号。

**·串行I/O控制**（serial I/O control)：此模块控制与串行设备的接口，串行设备是以每次1位的方式进行通信的设备

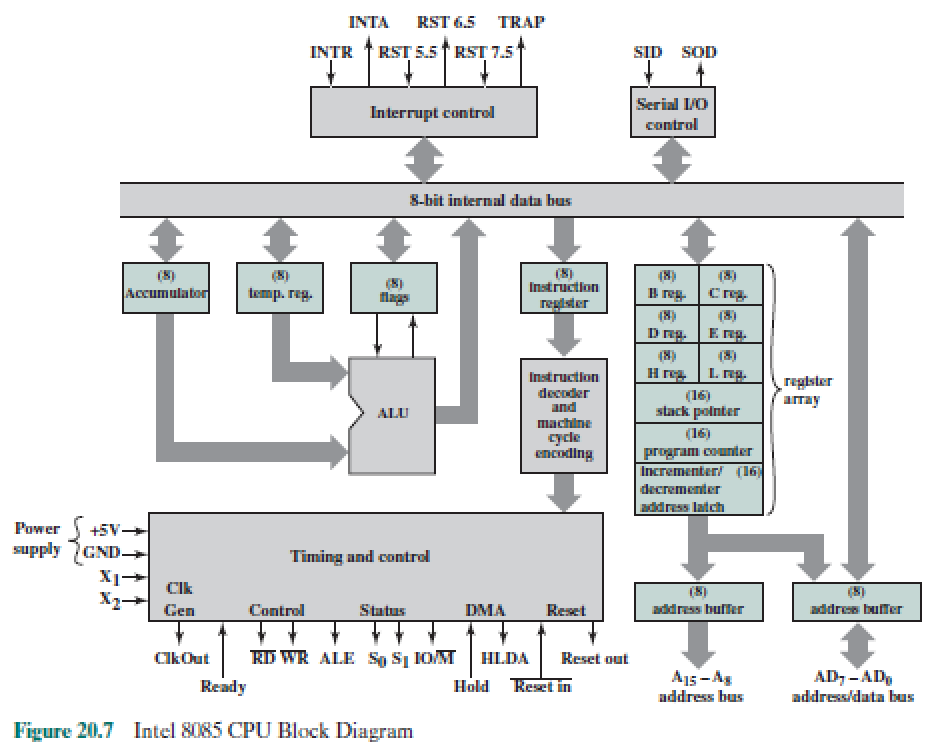
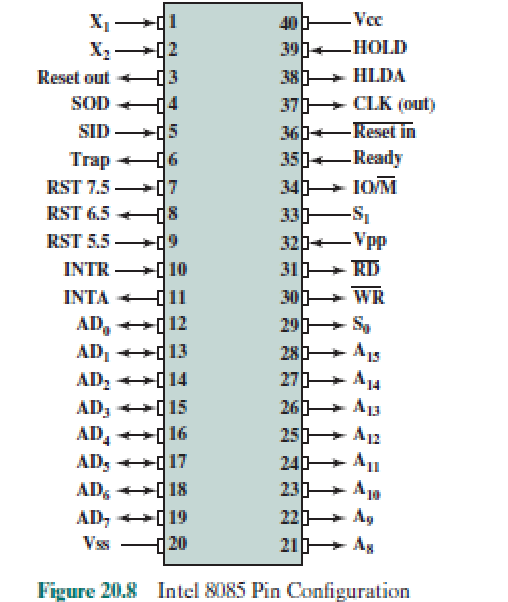


表20-2说明了8085的外部信号，它们连接到外部系统总线上。这些信号是8085处理器与系统其余部分的联系（见图20-8）。



控制器由两个组件组成：（1）指令译码器和机器周期编码（instruction decoder and machine cycle encoding），（2）定时和控制（timing and control）。第一个组件的讨论推迟到下一节。控制器功能主要在于定时和控制各个模块。这个模块包括一个时钟并接受当前指令和某些外部控制信号。它的控制部分输出信号，其中包括送到CPU其他部件的控制信号和送到外部系统总线的控制信号。

CPU操作的时序是与时钟同步的，并受控于控制器的控制信号。每个指令周期又被分为1~5个机器周期，每个机器周期又细分为3~5个状态（states）。每个状态持续一个时钟周期。在一个状态周期内，CPU完成由控制信号所确定的一个或多个同时进行的微操作。

对于给定的一条指令而言，机器周期数目是固定的，但不同指令间该数目有差异。机器周期定义为等同于总线访问。于是，一条指令的机器周期数取决于CPU必须与外部设备通信的次数。例如，若一条指令由两个8位字节组成，则取此指令需要两个机器周期。若此指令涉及1字节的存储器或I/O操作，则还需要第三个机器周期用于执行。

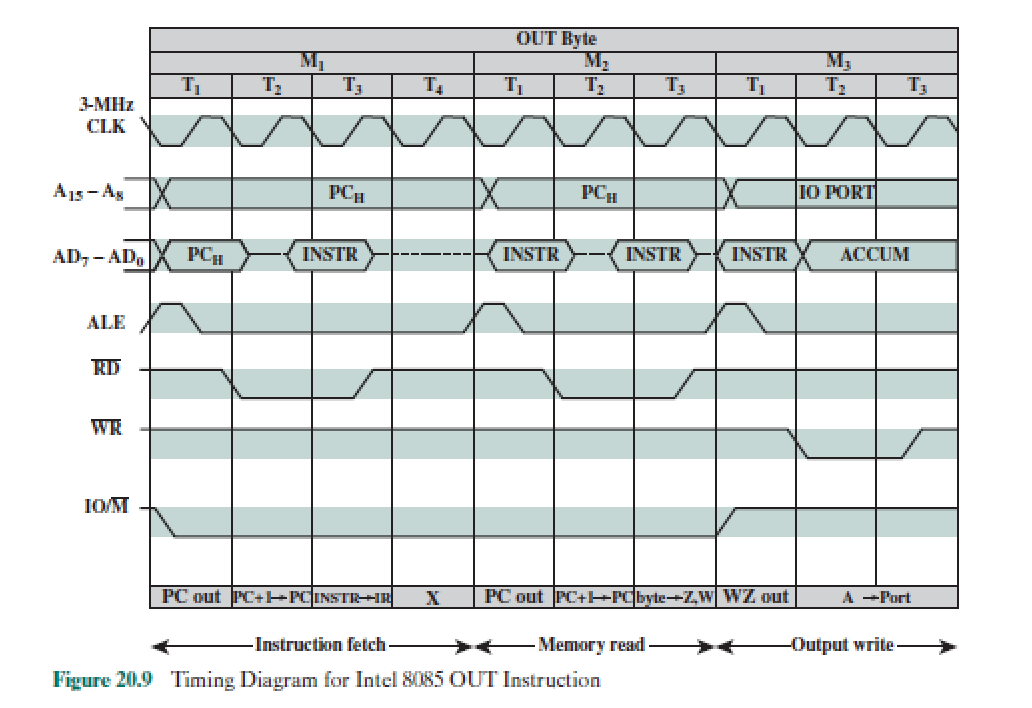


图20-9给出8085时序的一个例子，表示了外部控制信号值的变化。当然与此同时，内部控制信号也正由控制器产生，用于控制内部的数据传输。此图表示的是一条OUT指令的指令周期，它需要三个机器周期（M1,，M2,，M3）。在第一个机器周期，此OUT指令的前一半被取出来；第二个机器周期取来指令的后一半，它含有用于输出的I/O设备号。在第三个机器周期，AC的内容经由数据总线写入所选择的设备。

利用发自控制器的地址锁存使能ALE（address latch enabled）脉冲来指示每一机器周期的开始。此ALE脉冲使外部电路处于待命状态。在机器周期M1的状态T1期间，控制器设置IO/M信号指出这是一个存储器操作。控制器亦使PC内容投放到地址总线（A15~A8）和地址/数据总线（AD7~AD0）上。在ALE脉冲的下降沿，总线上的其他模块会保存地址。

在状态T2期间，存储器模块将被寻址的位置中的内容放到地址／数据总线上。控制器设置读控制RD（read control）信号以指示一个读，但它等待着直到T3才复制总线上的数据。这给存储器模块提供一个时间让它把数据放于总线上，使信号电平得以稳定。最后一个状态T4是一个总线空闲（bus idle）状态，在此期间CPU译码此指令。其余的机器周期也以类似方式进行。

**20．3硬布线实现**

通过对控制器的输入/输出和功能的介绍，我们详细讨论了控制器。现在是讨论控制器实现问题的时候了。已采用的各种技术可分为两大类`

·硬布线实现（hardwired implementation)

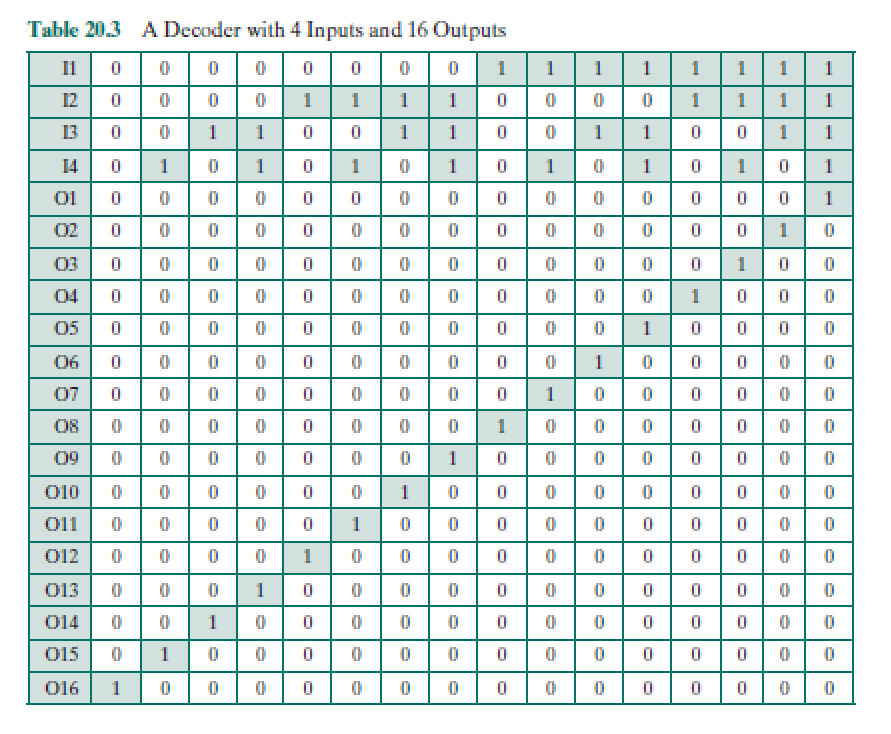
·微程序实现（microprogrammed implementation）

通过**硬布线**方式实现，控制器本质上是一个组合电路。它把输入逻辑信号转换为一组输出逻辑信号，即控制信号。本节考察此方法。微程序实现是第21章的主题

**20．3．1控制器输入**

图20-4给出的是我们至此已讨论过的控制器。其关键输入是指令寄存器、时钟、标志和控制总线信号。对于标志和控制总线信号来说，一般是每个位都有某种意义（例如溢出位)。而另两个输入对于控制器来说其作用不是那么直接。

首先考虑指令寄存器。控制器使用指令的操作码，为不同的指令完成不同的动作（发出不同的控制信号组合）。为简化控制器逻辑，应使每一操作码有一个唯一的逻辑输入。译码器（decoder）能完成这个功能，它接收一个编码了的输入并产生单一的输出。通常，译码器有*n*个输入和2*n*（2的*n*次方）个输出。2*n*个不同输入之一将产生唯一的一个输出。表20-3是当*n* = 4时的一个例子。控制器的译码器更复杂些，它要考虑变长的操作码。第11章给出了使用数字逻辑实现译码器的一个例子。



控制器的时钟部分发出一个重复的脉冲序列。用于度量微操作的持续时间。本质上讲，时钟脉冲周期要足够长，以允许信号能沿着数据通路传播和通过CPU电路。然而正如我们已看到的，在一个指令周期内，控制器要在不同时间单位发送不同的控制信号今于是，我们希望有一个计数器作为控制器的输入，在不同计时步骤T1，T2等发出不同的控制信号。在指令周期结束时，控制器必须通知计数器，以使它从的重新开始。

经过这两点改进，控制器能表示成如图20-10所示的结构。

**20．3．2控制器逻辑**

为定义控制器的硬布线实现方案，剩下的就是讨论控制器的内部逻辑了，它会根据输入信号经函数产生输出控制信号。

基本上，我们要针对每个控制信号导出其布尔表达式作为输入的函数。示例说明更清楚。让我们再一次考察图20-5说明的简单例子。在表20-1中已明确，微操作序列和控制信号都需要对指令周期4个阶段中的3个进行控制。

让我们考虑一个简单的控制信号C5，该信号使外部数据总线上的数据读入MBR。可看出，它在表20-1被使用了两次。重新定义两个新的控制信号P和Q，其组合释义如下：

PQ=OO 取指周期

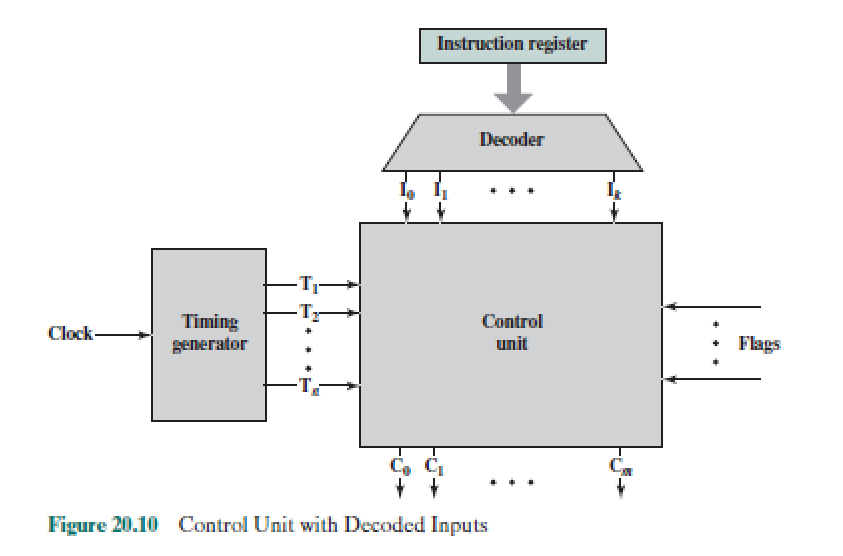
PQ=O1 间接周期

PQ=10 执行周期

PQ=11 中断周期

则如下的表达式定义了C5

C5 = ••T2 + •Q•T2



即控制信号C5，在取指和间接周期的第二个时间单位内有效。

这个表达式仍不完整，执行周期段也需要C5。在这个简例中假定只有LDA、ADD和AND三条指令还需要在执行时读取内存。现可将C5定义为：

C5 =••T2 + •Q•T2 + P••(LDA + ADD + AND)•T2

对CPU产生的任何控制信号重复这样的过程，结果得到一组布尔等式，通过定义了控制器的行为，从而定义了CPU的行为。

为将整个过程组织在一起，控制器必须控制指令周期的状态。正如我们曾提到，在每个子周期（取指、间接、执行、中断）结束时，控制器都要发出一个信号，以使时序发生器重新初始化并发射T1。控制器还必须设置P、Q的相应电平值来定义下面将要完成子周期。

读者应该能想到，对于当代复杂的CPU而言，其控制器实现所需的布尔表达式数量是非常大的。实现这样一个组合电路来满足所有这些布尔表达式的任务将变得异常困难。所以普遍使用的是一种更为简洁的方法：微程序设计方式（microprogramming）。下一章我们会来讨论这一主题。

**20.4关键词、思考题和习题**

**关键词**

control bus：控制总线

control path：控制通路

control signal：控制信号

control unit：控制单元

hardwired implementation：硬布线实现

micro-operations：微操作

**思考题**

20.1 说明指令的编写顺序（written sequence）与时间顺序（time sequence）的区别。

20.2 指令和微操作的关系是什么？

20.3 CPU控制器的总体功能是什么？

20.4 概述表征控制器的三步骤。

20.5 控制器要完成的基本任务有哪些？

20.6 给出一个制器输人输出的典型列表。

20.7 列出三类制信号。

20.8 简要说明制器的硬布线实现是什么意思？

**习题**

20.1 假设有一个ALU不能做减法运算，但它能加两个输入寄存器，并能对两个寄存器的各位取逻辑反。数是以2的补码形式存储的。请列出实现减法的控制器必须完成的微操作。

20.2 若使图20-5中的CPU完成如下指令，请以与表20-1相同的方式列出其微操作和控制信号：

·装载一个数到累加器

·保存累加器内容到存储器  
·加一个数到累加器

·AND一个数到累加器

·跳转  
·若AC = 0则跳转

·累加器取反

20.3 假设图20-6中的沿总线传播、以及通过ALU的信号其传播延迟分别20ns和10ns。由总线将数据拷贝到寄存器需要10ns。那么，必须允许多少时间才能：  
（a）从一个寄存器到另一寄存器传送数据。  
（b）递增程序计数器。

20.4 以图20-6的内部总线结构为例，考虑加一个数到AC，若该数是：  
（a）一个立即数  
（b）一个直接寻址的操作数  
（c）一个间接寻址的操作数  
请写出所需的微操作序列。

20.5 考虑一个按照图10-14来实现的栈，请为（a）由栈弹出（pop）和（b）压入（push）此栈，给出相应的微操作序列。

