# 第21章 微程序控制

21.1**基本概念**

  微指令

微程序控制单元

wilkes控制器

优缺点

21.2**微指令排序**

  设计考虑

  定序技术

地址生成

LSI-11微指令定序

21.3**微指令执行**

  微指令的分类

微指令编码

LSI-11微指令执行

IBM 3033微指令执行

21.4**TI 8800**

  微指令格式

微定序器

寄存器ALU

21.5**关键词、思考题和习题**

**学习目标**

学习本章后，你应该能够：

* 介绍微程序控制的基本概念。
* 了解硬连线控制和微程序控制的区别。
* 讨论定序技术的基本分类。
* 介绍微指令的分类概述。

20世纪50年代，M.V.Wilkes最先提出了微程序（microprogram）这个术语[WILK51]。Wilkes提出了一种控制器的设计方法，它是有组织而又有体系的，避免了硬布线实现的复杂性。这个思想引起了很多研究人员的注意，但由于它要求一个快速且不太昂贵的控制存储器而显得不太实际。

Datamation在1964年2月刊上对微程序的技术状况进行了评价。因为那时没有任何微程序式系统在广泛使用，于是一篇论文[HI1.L64]总结了相当流行的看法，说微程序“是有些前途暗淡，没有任何主要厂商表明对此技术感兴趣，尽管可推测他们都曾考察过此技术。”

但是.这种情况在几个月后就发生了急剧变化。IBM的System/360于4月公布，虽然不是全部，但除了它的最大型号之外，其余都是微程序的。尽管360系列是在半导体ROM可以使用之前，但微程序的优点足够令人信服地使IBM这样做。从此，微程序设计成为实现CISC处理器的流行技术。近几年来，微程序设计已较少使用，但它仍是计算机设计的有用工具。例如，正如我们看到的，Pentium 4的机器指令转换为类RISC形式，它们中的大多数不使用微程序来执行；然而，仍有某些指令使用微程序来执行。

## 21.1 基本概念

### **微指令**

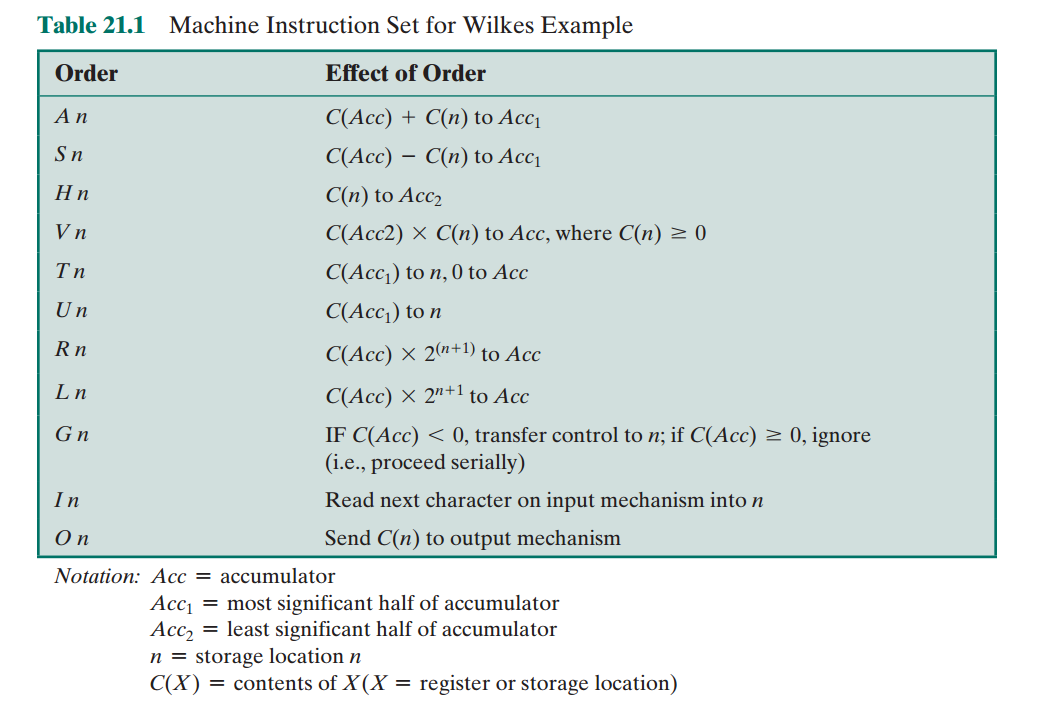
正如前面刚描述过的，控制器像是一个相当简单的设备。但是，若以基本逻辑元件互连来实现一个控制器，却不是一个简单的任务。设计必须包括定序微操作、执行微操作、解释操作码以及根据ALL:的标志来决策等逻辑a设计和測试这样一片硬件是困难的，并且这种设计也不太灵活。例如，改动设计来增加新的机器指令就相当困难。

有一种替代方法，就是**微程序控制器设计方法**，它是很多CISC处理器普遍采用的设计方法。

参见表21-1，除使用控制信号外，每个微操作都以符号表示来描述。这种表示看起来像是一种编程语言。实际上，它确实是一种编程语言，叫做**微程序设计语言**（microprogramming language）。 每行描述一个时间内出现的一组微操作，并称为一条**微指令**（microinstruction）。这种微指令序列被称为**微程序或固件**（firmware）。后一术语反映了这样的事实：微程序是介于硬件与软件之间的。以固件进行设计要比硬件容易，但写一个固件程序要比一个软件程序困难得多。

如何使用微程序概念来实现控制器呢？考虑到对于每个微操作，控制器所做的全部事情就是产生一组控制信号。对任一微操作，控制器发出的每根控制线或开或关。自然，对应这种情况，每根控制线可由一个二进制数字表示。这样，我们就构造了一个控制字（control word），其中每位代表一根控制线，从而每个微操作能用控制字中的不同的0和1的式样来表示。

若能将这些控制字串在一起，就能表示控制器完成的微操作序列了。然而，必须认识到微操作序列不是固定的。有时有间接周期，有时又没有。让我们把控制字放入一个存储器单元中，每个字有自己唯一的地址。现在再给每个控制字添加一个地址字段，以指示若某种条件为真时（例如，存储器访问指令中的间接位为1），将要执行的下一控制字的位置。还有，添加少数几位用于指示条件的真假。



结果是在图21-1a中的所谓**水平微指令**（horizontal microinstruction）。此微指令或控制字的格式说明如下。这里对每一 CPU内控制线和每一系统控制总线都有相应1位。它还有一个指示转移发生条件的条件字段（condition field）和转移目标地址字段（address field），指示将要执行的微指令地址。这样的微指令以如下的方式解释执行。

（1）执行这条微指令的效果是，打开所有位值为1的控制线，关闭所有位值为o的控制线。生成的控制信号会使得一个或多个微操作被完成。

（2）若条件位指示的条件为假，则顺序执行下一条指令。

（3）若条件位指示的条件为真，则地址字段指向的微指令是将被执行的下一条微指令。

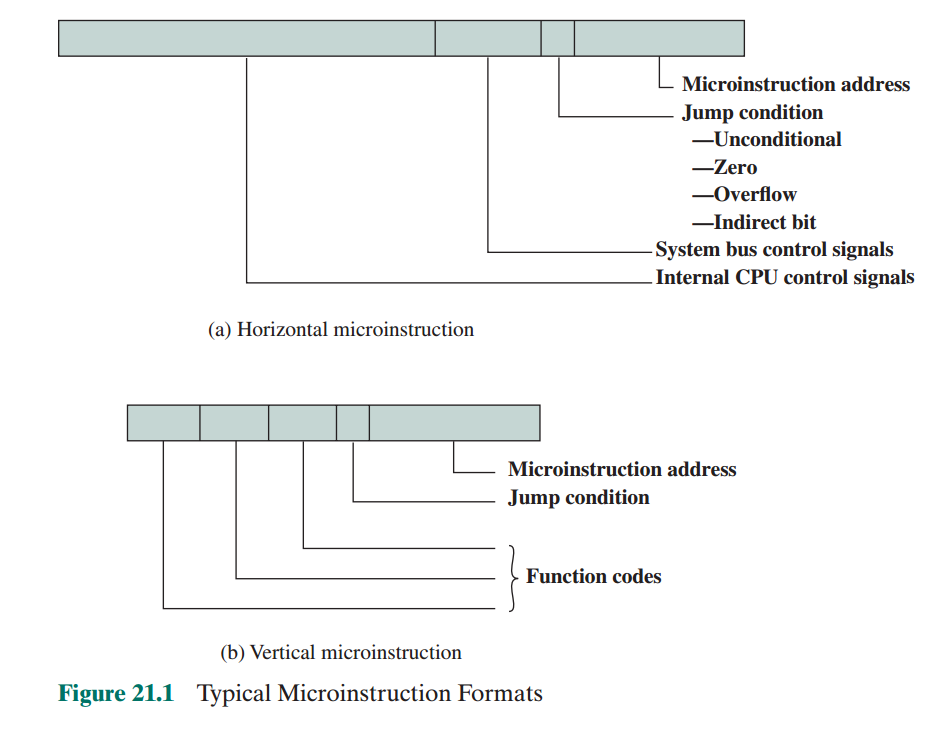
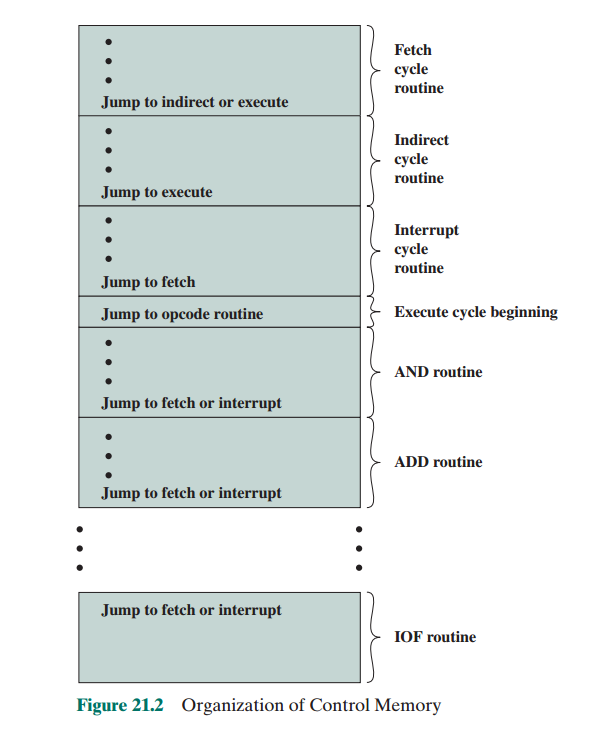


图21-2说明这些控制字或微指令在**控制存储器**（control memory）中是如何安排的。在每个例程中微指令是顺序执行的，在例程的终端有一个分支或转移微指令，指出下面将到何处执行。这里还有一个专门的执行周期例程，它的目的仅在于，根据当前的操作码指明哪一个机器指令（AND、ADD等）例程将被执行。

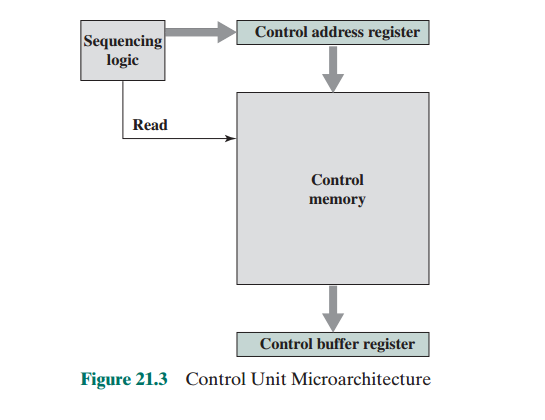
图21-2的控制存储器是控制器整体操作的简明描述，它定义了在每个周期（取指、间接、执行、中断）内将要完成的微操作序列，它也指定了这些周期的顺序。如果没有别的事情，这个表示可作为一个说明具体计算机控制器功能的有用工具。但它还不仅只能作为一个设计说明工具，它亦是实现控制器的一种方式。



### 微程序控制器

图21-2的控制存储器含布描述控制器行为的微程序，下面以执行这个微程序的简单方式来讨论控制器的实现。

图21-3表示了这种控制器实现方式的关键部件：控制存储器（control memory）存有一组微指令；控制地址寄存器（control address register）含有下面即将被读取的微指令地址；当一条微指令由控制存储器读出后，即被传送到控制缓冲寄存器（control buffer register）。此寄存器的左半部分与控制器发出的控制线相连接。于是，由控制存储器读一条微指令等同于执行这条微指令。图中所示的第三个部件是定序器（sequencing unit），它向控制地址寄存器装入地址并发出读命令。



让我们更详细地考察这种结构，如图21-4所示。将此图与图21.3相比，此控制器仍有相同的输入（IR， ALU标志、时钟）和输出（控制信号）。此控制器的功能如下所述。

（1）为执行一条指令，定序逻辑发出一个读命令给控制存储器。

（2）控制地址寄存器指定的一个字读入到控制缓冲寄存器。

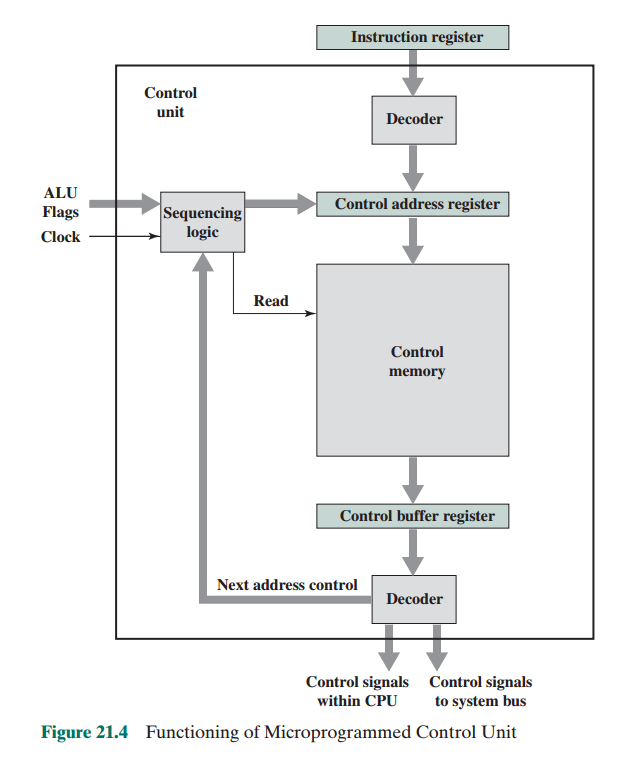
（3）控制缓冲寄存器的内容生成控制信号，并为定序逻辑提供下一条地址信息。

（4）定序逻辑根据这个地址信息和AI.U标志，将一个新地址装入到控制地址寄存器。所有这些事情都发生在一个时钟周期内。

上述的第4个步骤还需进一步说明。在每条微指令结束时，定序逻辑都要将一新的地址装入到控制地址寄存器。这取决于ALU标志和控制缓冲寄存器的内容，它要进行三选一的决策。

* **取顺序下一条微指令：**加1到控制地址寄存器。
* **基于跳转微指令转移到一个新的例程：**将控制缓冲寄存器的地址字段装入控制地址寄存器。
* **转移到一个机器指令例程：**根据IR中的操作码向控制地址寄存器装入机器指令例程的第一条微指令。

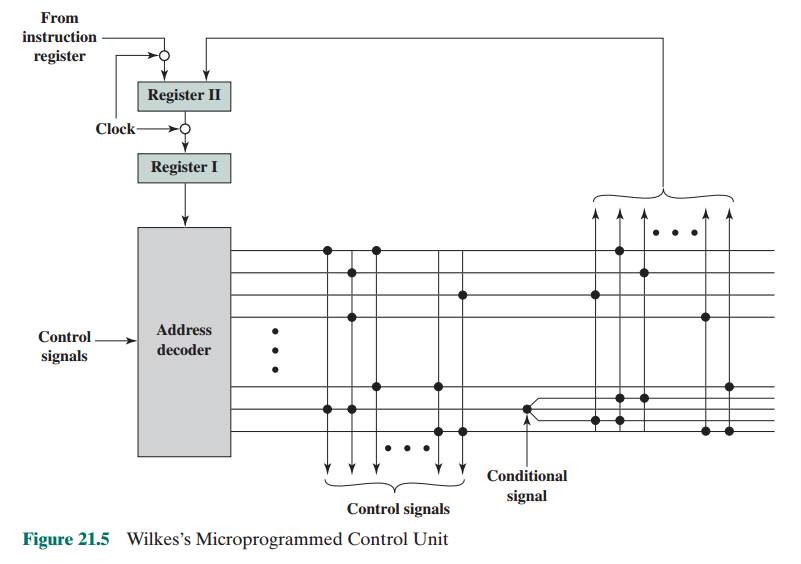
图21-4显示了两个标记有译码器（deader）的模。上方的译码器将IR中的地址码翻译为一个控制存储器地址。下方的译码器不用于水平微指令，而是用于**垂直微指令**（vertical microinstruction）（图21-1b）。正如前面所提到过的，以水平微指令方式，微指令的控制字段中的每一位都接到控制线。以垂直指令方式，一个代码用于表示将被完成的一项动作，例如MAR←（PC），而由此译码器将这个代码转换为对应的控制信号。垂直微指令的优点在于它比水平微指今更紧缩（位数少），代价是一个小量的附加逻辑和时间延迟。



### Wilkes 控制

正如前面所说过的，Wilkes在1951年最早提出使用微程序控制器。这个方案接着被修正并形成更详尽的设计[WTLK53]。现在考察这个最初的方案仍是具有指导意义的。

Wilkes关注的是为设计控制器开发出一种系统化的方法。他提出的配置方案如图21-5所示。系统的核心是一个阵列，其中有些部分连接着二极管。在一个机器周期内，阵列的一行被激活。阵列中连接着二极管的地方（图中以圆点所示）产生信号。每行的前一部分产生控制CPU操作的控制信号，后一部分产生下一周期将要激活的行地址。于是，阵列的每一行是一条微指令，整个阵列则是控制存储器。



机器周期开始时，将要激活的行地址保存在寄存器I中。这个地址输入到译码器，当它被一个时钟脉冲启动时，它激活阵列的某一行。在此周期，或是指令寄存器中的操作码，或是行的后一部分，被传送到寄存器II，这取决于相应的控制信号。然后由一个时钟脉冲打开寄存器II到寄存器I的门。交替的时钟脉冲用于启动阵列行和寄存器II到寄存器I的传送。因为译码器是一个简单的组合电路，故需要这种双寄存器的安排；否则，只使用一个寄存器的话，在一个周期内输出可能会反馈到输入，引起一种不稳定的状况。

这种思想很类似于前面（见图21-1a）所介绍的水平微程序设计方法。主要的不同在于：在前面的介绍中，控制地址寄存器能递增1、得到下一顺序地址；而在Wilkes方案中，下一地址是在微指令中。为准许转移，一行必须包含两个地址部分，受控于一个条件信号（例如标志），如图16-5所示。

为证实所提出的思想，Wilkes提供了一个以这种方式实现控制器的简单机器。这个例子是已知的第一个微程序CPU设汁。这里再一次重复它是有益的，因为它说明了许多当代微程序设计的原则。

此原机型CPU包括如下寄存器：

A被乘数

B累加器（低半部分）

C累加器（高半部分）

D移位寄存器

另外，还有只由控制器可访问的三个寄存器和两个1位的标志。三个寄存器是：  
E用作存储器地址寄存器（MAR）和暂时存储的寄存器  
F程序计数器

G另一个暂时寄存器，用作计数

表21-1列出了这个例子的机器指令集。表21-2是实现控制器的全部微指令集，以符号形式表示。于是，完整定义此系统需要38条微指令。

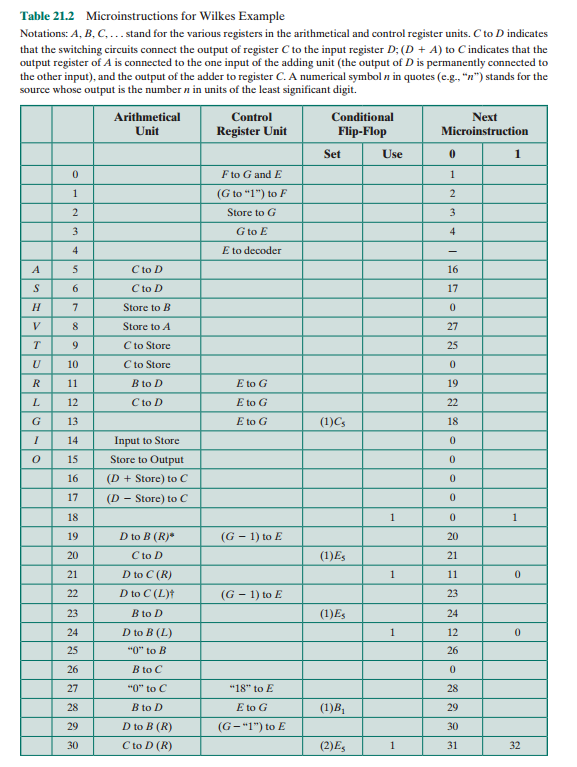
表中第1列给出了每个微指令的地址（行号）。那些对应着操作码的地址被标记出来。例如，当遇到加法指令时，位置5处的微指令被执行。第2列与第3列分别表示ALU和控制寄存器单元发生的动作。每个符号表示都必须翻译为一组控制信号（微指令的位）。第4列与第5列表示两个标志（触发器）的设定和使用。第4列指定此标志的信号置位。例如，（1） C5意味着由寄存器C的符号位来置位标志1。若第5列有一标志识别符，那么第6列与第7列就会含有两个可选的微指令地址。否则，第6列指定将被取的下一条微指令地址。

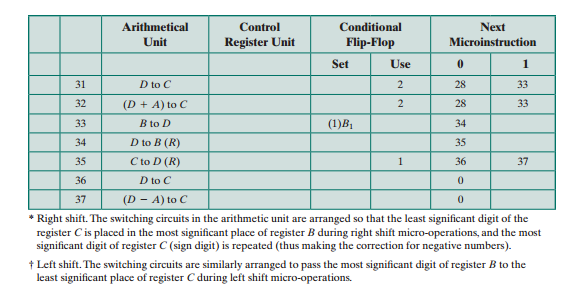
微指令0到4构成了取指周期。微指令4将操作码提交给译码器，译码器产生对应于待取机器指令的微指令地。如果细心研究21-2，读者不难推导出控制器的全部功能。

### 优缺点

使用微程序实现控制器的优点在于，简化了控制器的设计任务，实现起来既成本较低，也能减少出错机会。硬布线控制器需要一个复杂的逻辑，用来使指令周期的众多微操作按序执行。而微程序控制器的译码器和定序逻辑单元是很简单的逻辑电路。

微程序控制器的主要缺点是：要比采用相同或相近半导体工艺的硬布线控制器慢一些。尽管如此，由于它的易实现性，使微程序设计成为当今CISC控制器的主导技术。而对于RISC处理器，由于它们的简单指令格式，一般使用硬布线控制器。下面更详尽地讨论微程序方法。





## 21.2微指令定序

微程序控制器的两个基本任务是：

* **微指令定序（microinsiruction sequencing）：**由控制存储器得到下一条微指令。
* **微指令执行（microinstruction execution）：**产生执行微指令的控制信号。

设计控制器时这些任务必须一起考虑，因为二者都影响微指令格式和控制器时序。这一节将重点

关注定序问题，而尽可能少提及有关格式与时序的问题，后两个问题留待下一节详细讨论。

### 设计考虑

设计微指令定序要考虑到两个问题：微指令的大小和地址生成时间。第一个问题是明显的，减小微指令的大小就能节省控制存储器的成本；第二个问题是尽可能快地执行微指令的最简单的要求。

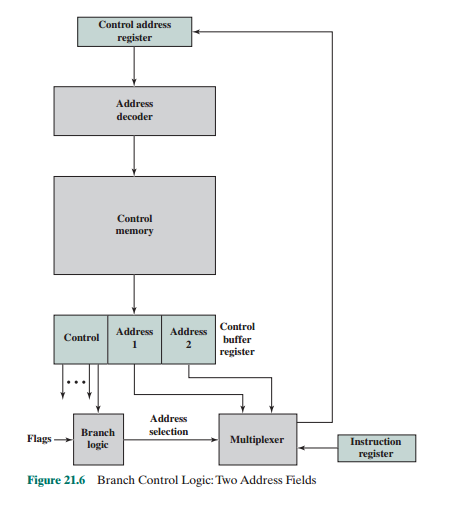
执行微程序时，获得下面将要执行的微指令的地址有如下三种情况：

* 由指令寄存器确定
* 下一顺序地址
* 转移

第一种情況在每指令周期中只出现一次，发生在指令刚刚取来之后。第二种情况在大多数情况下是最普遍的，然而设计不能只为顺序存取来优化来转移，无论是有条件还是无条件的，都是微程序必不可少的部分。而且，微指令序列倾向于短小，每3-4条微指令之后就可能有一个转移发生［SIEW82］。于是，为微指令转移设计一种紧缩的、高时效的技术是重要的。

### 定序技术

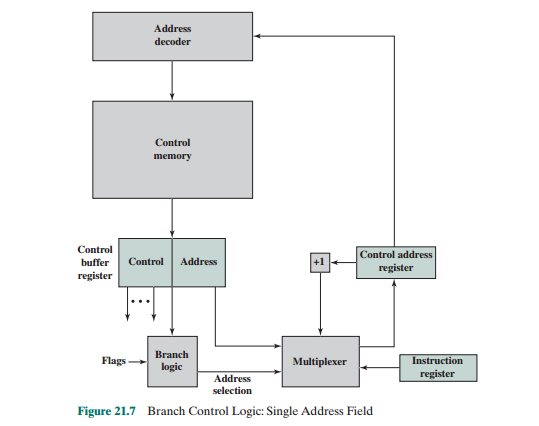
必须根据当前的微指令、条件标志和指令寄存器的内容，产生下一微指令的控制存储器地址。已有很多技术被采用，我们以图21-6到图21-8来说明这些技术的常规分类。以微指令中的地址信息为例，可以大致分为双地址字段、单地址字段、可变格式三类。

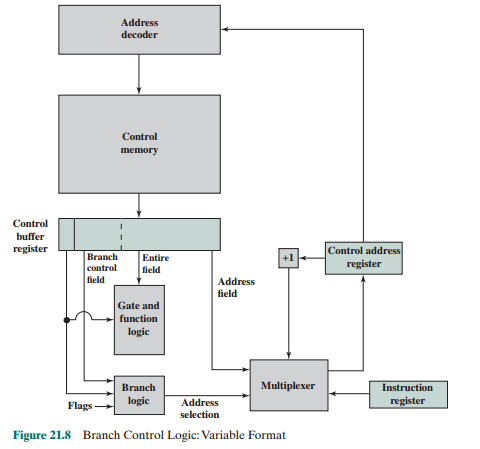


最简单的方法是在每条微指令中提供两个地址字段。图21-6展示了如何使用这些地址宇段，图中使用了一个多路选择器（或多路器），两个地址字段和指令寄存器的内容连接到这个多路选择器作为输入。多路选择器根据地址选择输入，发送两个地址之中的某一个或操作码到控制地址寄存器（CAR）。CAR接着被译码以产生下一微指令地扯。由一转移逻辑（branch logic）模块接收控制器标志和微指令控制部分的输入，向多路选择器提供多路选择信号。

虽然双地址宇段方法简单，但它的微指令比其他方法需要更多的位。添加一些逻辑就可以做到节省。一种普遍的方法是只有单地址字段（参见图21-7）。按照这种方法，下一地处的选择项是：

* 地址字段
* 指令寄存器代码
* 下一顺序地址





地址选择信号确定哪项被选中，这种方法将地址字段减少到1个。然而要注意，一般不会经常用到此地址字段。因此，在微指令编码策略中存在着某种低效因素。

另外的方法是提供两种完全不同的指令格式（见图21-8）。一位字段用于指定哪种格式正被使用。在一种格式中，其余的位用于产生控制信号。在另一种格式中，某些位用于启动转移逻辑模块，剩余的位用来提供地址。以第一种格式，下一地址或是下一顺序地址，或是由指令寄存器来获取的地址。在第二种格式中，指定了一个有条件或无条件转移。这种方法的缺点之一是每一条转移微指令将耗费一个时钟周期。其他方法的地址产生，与控制信号生成在同一周期，从而减少了控制存储器的存取。

刚才介绍的方法只是通用方法。具体的实现常常是这些技术的組合或变异。

### 地址生成

我们已从格式考虑和通常逻辑需求的观点查看了定序问题，另一观点是考虑取得或计算出下一地址的各种方式。

表21-3列出了各种格式生成技术。这些可分成显式（explicit）技术和隐式（implicit）技术两大类。显式技术在微指令中直接给出可用地址，隐式技术要求附加逻辑来产生地址。

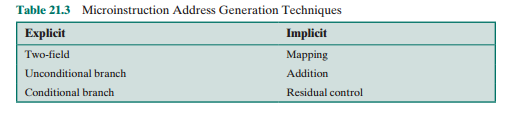
我们已和显式技术打过交道了。对于双地址字段法，每条微指令都有两个可选地址可用。使用单地址字段法或可变格式，能实现各种转移微指令。一个条件转移微指令依赖于如下类型信息：

* ALU标志；
* 机器指令的操作码部分或地址模式字段；
* 选定寄存器的一部分，如符号位；
* 控制器内的状态位。

几种隐式技术也得到普遍采用。其中之一，映射，实际上是所有设计都需要的。机器指令的操作码部分必须映射成微指令地址。这在每指令周期中仅出现一次。

普通的隐式技术是将两个地址部分相加或组合来形成一个完整的地址。IBM S/360系列［TUCK67］以及S/370多种型号都采用这种技术。下面以［BM 3033］作为例子来说明此技术。

IBM 3033的控制地址寄存器是13位长，如图21-9所示。它能分成两个地址部分，高8位(00-07） 一般在一个指令周期到下一个指令周期之间都不会发生改变。微指令执行期间，此8位直接由微指令的8位字段（BA字段）复制到控制地址寄存器的高8位。这就在控制存储器中选定了一个M条微指令的块。控制地址寄存器的剩余5位用于指定此块中待取微指令的具体地址。这5位的每一位都由当前微指令的一个4位字段（有一位是7位字段）所确定；这些字段指定了设置相应位的条件。例  
如，根据最近一次的ALU运算是否出现进位，将控制地址寄存器的一位设置为0或1。



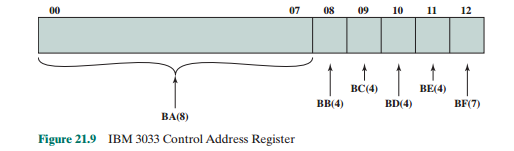


表21-3所列的最后一种方法是剩余控制。这种方法涉及使用先前已保存在控制器内的一个暂存装置中的微指令地址。例如，某些微指令集具有子程序调用功能，一个内部寄存器或寄存器栈用于暂存子程序返回地，使用这一方法的例子是LSI-11机，下面来考察它。

### 微指令定序

LS!-] I是PDP-11的微型机版本，系统的主要部件安装在单板上。LSI-11的实现使用了微程序控制器［SEBE76］。

LSI-11的微指令有22位，控制存储器的容逊是2K🞨22位字。以如下5种方式来确定下一微指令的地址。

* **下一顺序地址：**在不出现另外的情况时，控制器的控制地址寄存器的内容增1。
* **操作码映射：**在每个指令周期开始时，由操作码确定下一微指令地址。
* **子程序机制：**后文有说明。
* **中断测试：**某些微指令有测试中断功能，若中断已出现，它确定下一条微指令的地。
* **转移：**使用了有条件和无条件的转移微指令。

LSI-11控制器提供了单层（one-level）子程序调用的机制。每个微指令都有一位用于此目。当此位置位时，控制地址寄存器更新过的内容装入一个11位的返回寄存器。子程序调用结束时，指示返回的微指令将使返回寄存器的内容装入控制地址寄存器。

此返回是无条件转移指令的一种形式。无条件转移指令的另一种形式是把微指令中的11位地址装入控制地址寄存器中。条件转移指令使用了微指令内的一个4位测试码，以对ALU各种条件码的测试进行决策。若条件不成立，下一顺序地址将被选择；若成立，微指令中的8位被装入控制地址寄存器的低8位。这允许在一个256字的存储器页内进行跳转。

由上可见，LSI-11的控制器内包括了一个强有力的地址定序机制。这就允许微程序设计人员有相当的灵活性，并使微程序设计任务能变得容易些。另一方面，这种方法要比功能简单的控制器需要更多的控制逻辑。

**21.3 Microinstruction Execution**

The microinstruction cycle is the basic event on a microprogrammed processor. Each cycle is made up of two parts: fetch and execute. The fetch portion is determined by the generation of a microinstruction address, and this was dealt with in the preceding section. This section deals with the execution of a microinstruction.

Recall that the effect of the execution of a microinstruction is to generate control signals. Some of these signals control points internal to the processor. The remaining signals go to the external control bus or other external interface. As an incidental function, the address of the next microinstruction is determined.

The preceding description suggests the organization of a control unit shown in Figure 21.10. This slightly revised version of Figure 21.4 emphasizes the focus of this section. The major modules in this diagram should by now be clear. The sequencing logic module contains the logic to perform the functions discussed in the preceding section. It generates the address of the next microinstruction, using as inputs the instruction register, ALU flags, the control address register (for incrementing), and the control buffer register. The last may provide an actual address, control bits, or both. The module is driven by a clock that determines the timing of the microin-struction cycle.



The control logic module generates control signals as a function of some of the bits in the microinstruction. It should be clear that the format and content of the microinstruction determines the complexity of the control logic module.

**A Taxonomy of Microinstructions**

Microinstructions can be classified in a variety of ways. Distinctions that are com-monly made in the literature include the following:

* Vertical/horizontal
* Packed/unpacked
* Hard/soft microprogramming
* Direct/indirect encoding

All of these bear on the format of the microinstruction. None of these terms has been used in a consistent, precise way in the literature. However, an examination of these pairs of qualities serves to illuminate microinstruction design alterna-tives. In the following paragraphs, we first look at the key design issue underlying all of these pairs of characteristics, and then we look at the concepts suggested by each pair.

在Wilkes[WILK51]的最初建议中，微仪器的每个位要么直接产生控制信号，要么直接产生下一个地址的一个位。



M21\_STAL6858\_10\_GE\_C21.indd 769  3/19/15 4:57 PM 

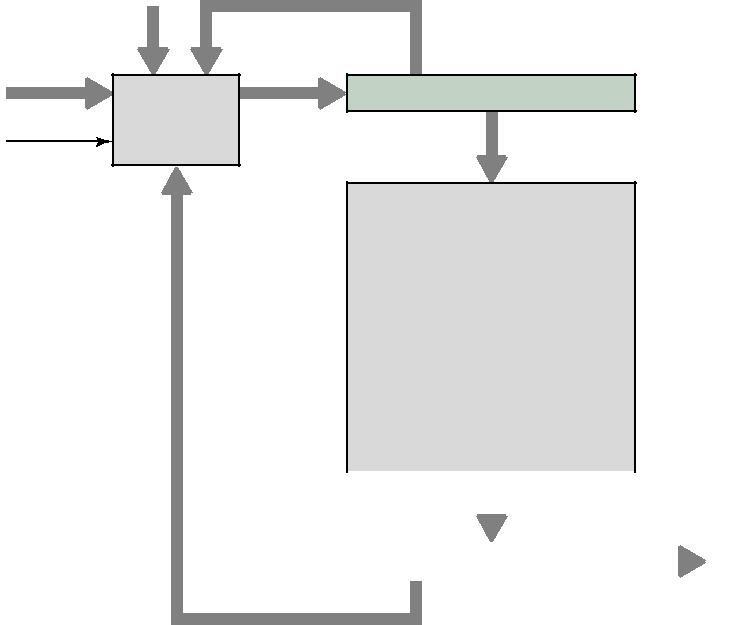




**770 Chapter 21 / 微程序控制**

**Instruction**

**register**

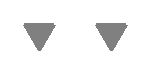


|  |  |  |  |
| --- | --- | --- | --- |
| **ALU** | **Sequencing** | **Control address register** |  |
| **Flags** |  |
| **Clock** | **logic** |  |  |

**Control**

**memory**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **Control** | | | |  |
| **Control buffer register** | | |  |  |  |  | **logic** | | |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |



**Internal External**

**control control**



**signals signals**

**Figure 21.10** Control Unit Organization

在前面的章节中，我们已经看到，使用更少微指令位的更复杂的地址排序方案是可能的。这些方案需要一个更复杂的序列逻辑模块。对于涉及控制信号的微指令部分，存在类似的权衡。通过对控制信息进行编码并随后对其进行解码以产生控制信号，可以节省控制字位。

如何进行编码？要回答这个问题，考虑由控制单元驱动的总共有K个不同的内部和外部控制信号。在威尔克斯的方案中，微指令的K位将专用于这个目的。这允许在任何指令周期中生成所有2K种可能的控制信号组合。但如果我们观察到并非所有可能的组合都将被使用，那么我们可以做得更好。例子包括：

* 两个源不能被门控到相同的目的地（例如，图21.5中的C2和C8）.
* 寄存器不能同时是源和目标（例如，图21.5中的C5和C12）.
* 一次只能向ALU呈现一个控制信号模式.
* 一次只能向外部控制总线呈现一种控制信号模式.



M21\_STAL6858\_10\_GE\_C21.indd 770  3/19/15 4:57 PM 





**21.3 / 微指令执行  771**

因此，对于给定的处理器，可以列出所有可能的控制信号组合，给出一些Q<2K的可能性。这些可以用最小的日志2Q位来编码，而（log2 Q）<K。这将是保持所有允许的控制信号组合的最紧的可能的编码形式。在实践中，不使用这种形式的编码，原因有两个:

* 它和纯解码（Wilkes）方案一样难以编程。这一点目前正在进一步讨论.
* 它需要一个复杂的、因此很慢的控制逻辑模块.

相反，做出了一些妥协。它们有两种:

* 使用比严格必要更多的位来编码可能的组合.
* 一些物理上允许的组合是不可能编码的.

后一种折衷方案具有减少比特数的效果。然而，最终的结果是使用超过Log2 Q位.

在下一小节中，我们将讨论具体的编码技术。本小节的其余部分讨论编码的效果以及用于描述它的各种术语.

基于上述，我们可以看到微指令格式的控制信号部分落在频谱上。在一个极端，每个控制信号都有一个比特；在另一个极端，使用高度编码的格式。表21.4显示微程序控制单元的其他特性也沿着谱下降，并且这些谱大体上由en编码度谱决定.



表中的第二对是相当明显的。纯威尔克斯方案将需要最多的位。显然，这个极端给出了硬件的最详细的视图.

**Table 21.4** The Microinstruction Spectrum

|  |  |
| --- | --- |
|  | **Characteristics** |
|  |  |
| Unencoded | Highly encoded |
| Many bits | Few bits |
| Detailed view of hardware | Aggregated view of hardware |
| Difficult to program | Easy to program |
| Concurrency fully exploited | Concurrency not fully exploited |
| Little or no control logic | Complex control logic |
| Fast execution | Slow execution |
| Optimize performance | Optimize programming |
|  | **Terminology** |
| Unpacked | Packed |
| Horizontal | Vertical |
| Hard | Soft |
|  |  |



M21\_STAL6858\_10\_GE\_C21.indd 771  3/19/15 4:57 PM 





**772 Chapter 21 / 微程序控制**

每个控制信号都由微程序单独控制。编码是以聚集函数或资源的方式进行的，以便微程序员以更高、更不详细的级别查看处理器。此外，为了减轻微程序设计的负担，设计了编码。同样，应该清楚的是，理解和协调所有控制信号的使用是一项艰巨的任务。如上所述，编码的后果之一通常是防止使用某些其他方面允许的组合.

前段从微程序员的角度讨论微指令设计。但是从硬件效果上也可以看出编码的程度。对于纯未编码格式，只需要很少或根本没有解码逻辑；每个位生成一个特定的控制信号。随着使用更紧凑和更聚合的编码方案，需要更复杂的解码逻辑。反过来，这可能会影响性能。通过更复杂的控制逻辑模块的门传播信号需要更多的时间。因此，编码微指令的执行时间比未编码微指令的执行时间长.

因此，表21.4中列出的所有特性都属于设计策略的范围。一般来说，落在规格说明书左端的设计旨在优化控制单元的性能。向右端的设计更关注微程序设计过程的优化。实际上，频谱右端附近的微指令集看起来非常像机器指令集。这方面的一个好例子是LSI-11设计，本节稍后将对其进行描述。通常，当目标仅仅是实现控制单元时，设计将接近于频谱的左端。目前讨论的IBM 3033设计属于这一类。正如我们将在后面讨论的，一些系统允许不同的用户使用相同的微指令设备构造不同的微程序。在后一种情况下，设计可能落在频谱的右端附近.



现在我们可以处理前面介绍的一些术语。表21.4显示了这三对术语如何与微指令频谱相关。本质上，所有这些对都描述了相同的东西，但强调不同的设计特征.

封装程度涉及给定控制任务与特定微指令位之间的识别程度。随着比特变得更加密集，给定数量的比特包含更多的信息。因此，包装意味着编码。术语水平和垂直与微指令的相对宽度有关。[SIEW82]作为经验法则建议垂直微指令的长度在16到40比特的范围内，而水平微指令的长度在40到100比特的范围内。术语”硬微程序设计”和”软微程序设计”用于暗示与底层控制信号和硬件布局的紧密程度。硬微程序通常是固定的，并致力于只读存储器。软微程序更具有可变性，并且暗示了用户微程序设计.

本节开头提到的另一对术语指的是直接编码和间接编码，现在我们将讨论这个问题.

**微指令编码**

实际上，微程序控制单元不是使用纯未编码或水平微指令格式设计的。至少某种程度的编码用于减少控制存储器宽度和简化微编程任务.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M21\_STAL6858\_10\_GE\_C21.indd 772 |  |  | 3/19/15 4:57 PM |  |
|  |  |  |
|  |  |  |  |  |





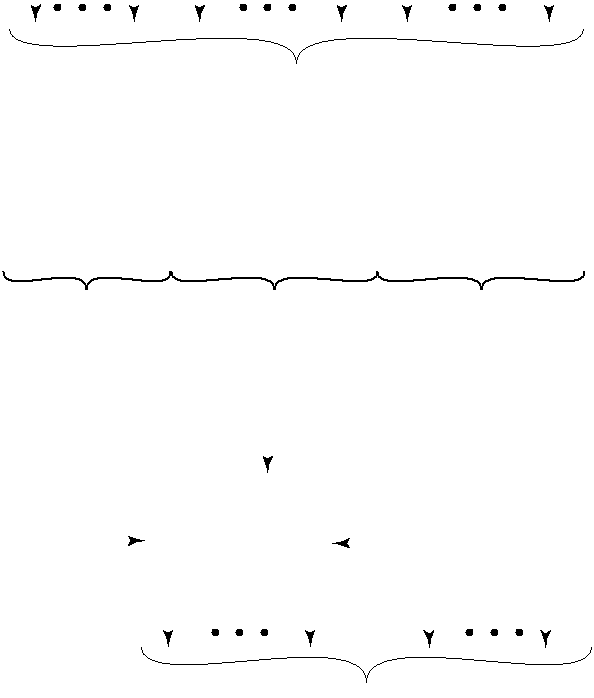
**21.3 / 微指令执行   773**

图21.11a显示了基本的编码技术。微结构被组织为一组字段。每个字段包含一个代码，该代码在解码时激活一个或多个控制信号.

让我们考虑一下这个布局的含义。当执行微指令时，每个字段都被解码并产生控制信号。因此，用n个字段，指定n个同时动作。每个动作导致一个或多个控制信号的激活。通常，但不总是，我们将希望设计格式，以便每个控制信号由不超过一个字段激活。然而，显然，每个控制信号必须能够被至少一个场激活.

现在考虑各个领域。由L位组成的字段可以包含2L代码中的一个，每个代码可以被编码为不同的控制信号模式。因为一次只能在一个字段中出现一个代码，所以代码是相互排斥的，因此，它们引起的操作是相互排斥的.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Field** | |  |  | **Field** | |  |  | **Field** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **Decode** | |  |  | **Decode** | |  |  | **Decode** | |  |  |
|  |  |  | **logic** | |  |  | **logic** | |  |  | **logic** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**Control signals**

(a) Direct encoding

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Field** | | |  |  | **Field** | | | | |  |  |  | **Field** | |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Decode** | | |  |  | **Decode** | | | | |  |  |  | **Decode** | |  |  |  |
|  |  | **logic** | | |  |  | **logic** | | | | |  |  |  | **logic** | |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **Decode** | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **logic** | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**Control signals**

1. Indirect encoding

**Figure 21.11** Microinstruction Encoding



M21\_STAL6858\_10\_GE\_C21.indd 773  3/19/15 4:57 PM 





**774 Chapter 21 / 微程序控制**

编码微指令格式的设计现在可以用简单的术语描述:

* 将格式组织成独立的字段。也就是说，每个字段描述一组动作（控制信号模式），使得来自不同字段的动作可以同时发生.
* 定义每个字段，以便该字段可以指定的替代操作是互斥的。也就是说，一次只能发生为给定字段指定的操作之一.

将编码微指令组织成字段有两种方法：功能和资源。函数编码方法识别机器内的函数，并按函数类型指定字段。例如，如果可变源可用于向累加器传输数据，则可为此指定一个字段，每个代码指定不同的源。资源编码将机器视为由一组独立的资源组成，并为每个资源分配一个字段（例如，I/O、内存、ALU）.

编码的另一个方面是直接还是间接（图21.11b）.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Simple register transfers** | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Special sequencing operations** | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **MDR** | | | |  |  |  |  |  |  | **Register** | | | | | |  |  | |  |  |  | |  |  | |  | |  |  | **CSAR** | | | | | | | |  |  |  |  | **Decoded MDR** | |  |
|  | **0** |  | **0** |  | **0** | **0** |  | **0** |  | **0** |  |  |  |  |  |  |  |  |  |  |  | **0** |  | **1** |  | **0** | **0** |  | **0** |  | **0** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **CSAR** | | | | | | |  |  |  |  |  | **Constant (in next byte)** | |  |
|  | **0** |  | **0** |  | **0** | **0** |  | **0** |  | **1** |  |  |  | **Register** | | | | | | |  |  |  |  |  | **MDR** | | | |  |  | **0** |  | **1** |  | **0** | **0** |  | **0** |  | **1** |  |  |  | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Skip** | | | | | | | | |  |  |  |  |  |  |
|  | **0** |  | **0** |  | **0** | **0** |  | **1** |  | **0** |  |  |  | **MAR** | | | |  |  |  |  |  |  | **Register** | | | | | |  |  | **0** |  | **1** |  | **0** | **0** |  | **1** |  | **0** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **Register select** | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **ALU operations** | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Memory operations** | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  | |  |  | |  | |  |  | **ACC** | | | | | |  |  |  |  | **ACC + Register** | | | |  |
| **Read** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **0** |  | **1** |  | **1** | **0** |  | **0** |  | **0** |  |  |  | | |  |  |
|  | **0** |  | **0** |  | **1** | **0** |  | **0** |  | **0** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **ACC** | | |  |  |  |  |  |  |  | **ACC – Register** | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Write** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **0** |  | **1** |  | **1** | **0** |  | **0** |  | **1** |  |  |  |  |  |  | | |  |  |
|  | **0** |  | **0** |  | **1** | **0** |  | **0** |  | **1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **ACC** | | | |  |  |  |  |  |  | **Register** | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **0** |  | **1** |  | **1** | **0** |  | **1** |  | **0** |  |  |  |  | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Register** | | | | | | | | |  |  |  |  | **ACC** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **0** |  | **1** |  | **1** | **0** |  | **1** |  | **1** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **ACC** | | | | |  |  |  |  |  | **Register + 1** | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **0** |  | **1** |  | **1** | **1** |  | **0** |  | **0** |  |  |  | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  | |  | |  |  | |  | |  |  |  | | | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Register select** | | | | | | | | | | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | (a) Vertical microinstruction format | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | **0** | | | | | **1** |  | **2** |  | **3** | | | **4** | | | | **5** | | **6** |  | **7** | **8** | **9 10 11 12 13 14 15 16 17 18** | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | | | | | | | | | |  |  |  | |  | | | |  | | | |  | |  |  | |  |  |  |  | |  | |  |  | |  | |  |  |  |  |  | | | | | | | | | | | |  |
|  | | | | | | | | | | | | |  |  |  | |  | | | |  | | | |  | |  |  | |  |  |  |  | |  | |  |  | |  | |  |  |  |  |  | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  | **Field** | | | | **1** |  | **2** | |  |  |  |  |  |  |  | **3** |  |  |  | **4** |  | **5** | | |  |  |  |  |  |  | **6** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Field de nition** | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **1 - register transfer** | | | | | | | | | | | | | | | | | |  |  | **4 - ALU operation** | | | | | | | | | | | | | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **2 - memory operation** | | | | | | | | | | | | | | | | | |  |  | **5 - register selection** | | | | | | | | | | | | | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **3 - sequencing operation** | | | | | | | | | | | | | | | | | | |  | **6 - constant** | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



(b) Horizontal microinstruction format

**Figure 21.12** Alternative Microinstruction Formats for a Simple Machine



M21\_STAL6858\_10\_GE\_C21.indd 774  3/19/15 4:57 PM 





**21.3 / 微指令执行  775**

通过间接编码，一个字段用于确定另一个字段的解释. 例如，考虑一个ALU，它能够执行八个不同的算术运算和八个不同的移位运算。1位字段可以用来指示是使用移位操作还是使用算术操作；3位字段可以指示操作。这种技术通常意味着两级解码，增加传播延迟.

图21.12是这些概念的一个简单例子。假设一个处理器具有一个累加器和几个内部寄存器，例如一个程序计数器和一个用于ALU输入的临时寄存器。图21.12A显示高度垂直的格式。前3位表示操作类型，后3位编码操作，最后2位选择内部寄存器。图21.12b是一个更水平的方法，尽管仍然使用编码。在这种情况下，不同的函数出现在不同的字段中.

**LSI-11 微指令执行**

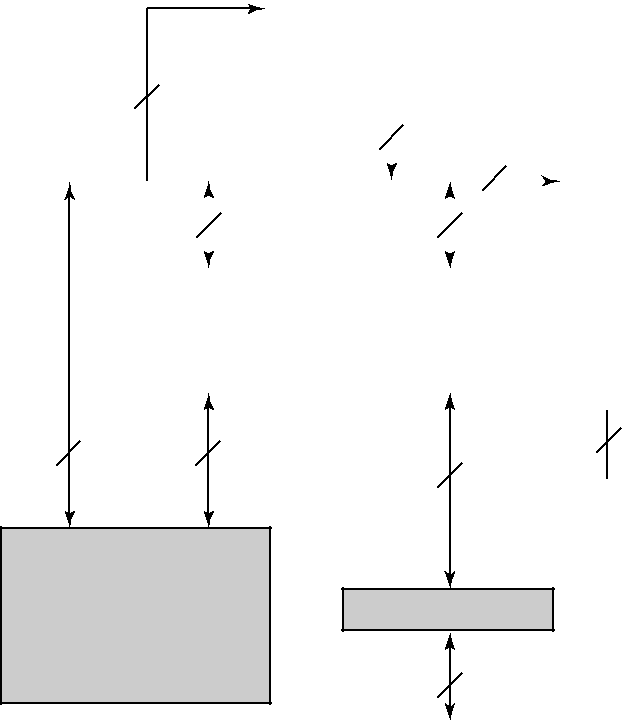
LSI-11[SEBE76]是垂直微指令方法的一个好例子。我们首先看控制单元的组织，然后看微指令格式.

***LSI-11控制单元组织*** LSI-11是作为单板处理器提供的PDP-11家族的第一个成员。该板包含三个LSI芯片，一个称为微指令总线（MIB）的内部总线，以及一些附加的接口逻辑.

图21.13以简化的形式描述了LSI-11前置处理器的组织.



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Control** | | |  |  |  |  |
|  |  |  | **store** | | |  |  |  |  |
|  | **11** |  |  |  |  |  |  |  |  |
|  |  |  |  | **22** |  |  |  |  |  |
|  |  |  |  | **22** | | | | **Microinstruction** |  |
|  |  |  |  | **bus** |  |
|  |  | **18** |  |  | **16** |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | **Control** | |  | **Data** | |  | |  |  |
|  | **chip** | |  | **chip** | |  | |  |  |
|  |  |  |  |  |  |  |  |  |  |



|  |  |  |  |
| --- | --- | --- | --- |
| **4** |  | **With no number indicated,** |  |
| **16** | **a path with multiple signals** |  |
|  |  |
|  |  |  |

**Bus control**

**and other** **Bus logic**

**processor**

**board logic**

**LSI-11 system**

**bus**

**Figure 21.13** Simplified Block Diagram of the LSI-11 Processor



M21\_STAL6858\_10\_GE\_C21.indd 775  3/19/15 4:57 PM 





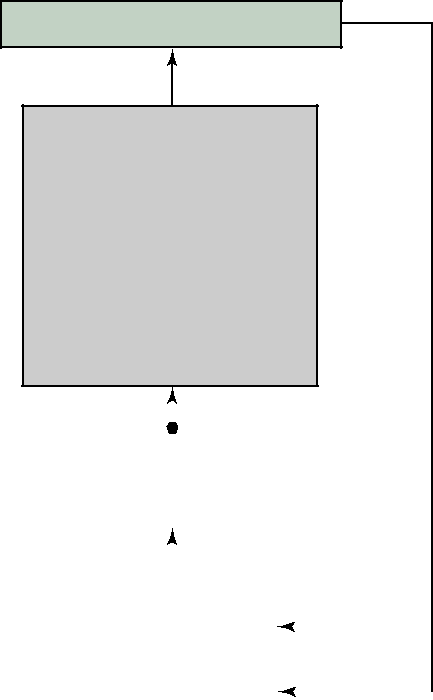
**776 Chapter 21 / 微程序控制**

这三个芯片是数据、控制和控制存储芯片。该数据芯片包含一个8位ALU、26个8位寄存器和若干状态码的存储器。16个寄存器用于实现PDP-11的8个16位通用寄存器。其他包括程序状态字、内存地址寄存器（MAR）和内存缓冲寄存器。因为ALU一次只处理8位，所以需要两次通过ALU来实现16位的PDP-11算术运算。这是由微程序控制的.

控制存储芯片或芯片包含22位宽的控制存储器。控制芯片包含用于排序和执行微指令的逻辑。它包含控制地址寄存器、控制数据寄存器和机器指令寄存器的副本.

MIB将所有组件连接在一起。在微指令提取期间，控制芯片在MIB上产生一个11位的地址。访问控制存储，产生一条22位的微指令，该指令被放置在MIB上。低阶16位进入数据芯片，而低阶18位进入控制芯片。高阶4位控制专用处理器板功能.

图21.14提供了对LSI-11控制单元的简化但更详细的描述：该图忽略了单个芯片的边界。第21.2节中描述的地址排序方案在两个模块中实现.



**Control data register**



**Control**

**store**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **Control address register** | | | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **Microprogram** | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **sequence** | |  | | |  |  |  |  |
|  |  |  |  |  |  |  |  | **control** | |  | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Return register** | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **Translation** | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | **INT** | |  |
| **Instruction register** | | |  |  |  |  |  | **array** | |  |  | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**Figure 21.14** Organization of the LSI-11 Control Unit



M21\_STAL6858\_10\_GE\_C21.indd 776  3/19/15 4:57 PM 





**21.3 / 微指令执行   777**

第21.2节中描述的地址排序方案在两个模块中实现。整个序列控制由微程序序列控制模块提供，该模块能够增加微指令地址寄存器并执行无条件分支。其他形式的地址计算由单独的转换数组执行。这是一个组合电路，它基于微指令、机器指令、微指令程序计数器和中断寄存器来生成地址.

翻译数组在下列情况下起作用:

* 操作码用于确定微例程的开始.
* 在适当的时候，测试微指令的地址模式位以执行适当的寻址.
* 对中断条件进行周期性测试.
* 对条件分支微指令进行评估.

***LSI-11微指令格式*** LSI-11使用非常垂直的微指令格式，只有22位宽。微指令集与其实现的PDP-11机器指令集非常相似。这种设计旨在优化控制单元的性能，在垂直，易于编程设计的约束。表21.5列出了LSI11的一些微指令.

图21.15显示了22位LSI-11微指令格式。高阶4位控制处理器板上的特殊功能。翻译位使翻译数组能够检查挂起的中断。加载返回寄存器位用于微例程的末尾，以便从返回寄存器加载下一个微指令地址.



剩下的16位用于高度编码的微操作。for-mat很像机器指令，具有可变长度的操作码和一个或多个操作数.

**Table 21.5** Some LSI-11 Microinstructions

|  |  |  |
| --- | --- | --- |
| **Arithmetic Operations** | **General Operations** |  |
| Add word (byte, literal) | MOV word (byte) |  |
| Test word (byte, literal) | Jump |  |
| Increment word (byte) by 1 | Return |  |
| Increment word (byte) by 2 | Conditional jump |  |
| Negate word (byte) | Set (reset) flags |  |
| Conditionally increment (decrement) byte | Load G low |  |
| Conditionally add word (byte) | Conditionally MOV word (byte) |  |
| Add word (byte) with carry | **Input/Output Operations** |  |
| Conditionally add digits |  |
| Input word (byte) |  |
| Subtract word (byte) |  |
| Compare word (byte, literal) | Input status word (byte) |  |
| Subtract word (byte) with carry | Read |  |
| Decrement word (byte) by 1 | Write |  |
| **Logical Operations** | Read (write) and increment word (byte) by 1 |  |
| Read (write) and increment word (byte) by 2 |  |
| AND word (byte, literal) | Read (write) acknowledge |  |
| Test word (byte) | Output word (byte, status) |  |
| OR word (byte) |  |  |
| Exclusive-OR word (byte) |  |  |
| Bit clear word (byte) |  |  |
| Shift word (byte) right (left) with (without) carry |  |  |
| Complement word (byte) |  |  |
|  |  |  |



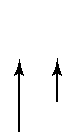
M21\_STAL6858\_10\_GE\_C21.indd 777  3/19/15 4:57 PM 





**778 Chapter 21 / 微程序控制**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **4** | **1** | **1** | **16** |  |
|  |  |  |  |  |
| **Special** |  |  | **Encoded micro-operations** |  |
| **functions** |  |  |  |
|  |  |  |  |



**Load return register**

**Translate**

(a) Format of the full LSI-11 microinstruction

**5** **11**



|  |  |
| --- | --- |
| **Opcode** | **Jump address** |
|  |  |

**Unconditional jump microinstruction format**

|  |  |  |
| --- | --- | --- |
| **4** | **4** | **8** |
|  |  |  |
| **Opcode** | **Test code** | **Jump address** |
|  |  |  |



**Conditional jump microinstruction format**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **4** |  | **8** | |  | **4** |
|  |  |  |  |  |  |
| **Opcode** |  | **Literal value** | |  | **A register** |
|  |  | |  |  |  |
| **Literal microinstruction format** | | | | |  |
| **8** | | **4** | |  | **4** |
|  | |  |  | |  |
| **Opcode** | |  | **B register** |  | **A register** |
|  |  |  |  |  |  |



**Register jump microinstruction format**

1. Format of the encoded part of the LSI-11 microinstruction

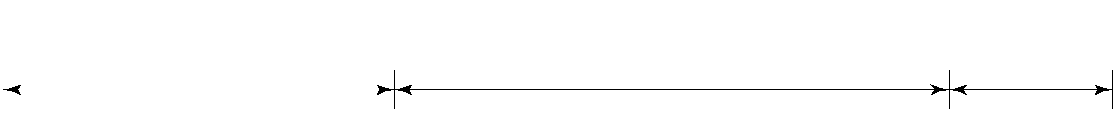
**Figure 21.15** LSI-11 Microinstruction Format

**IBM 3033 微指令执行**



标准IBM 3033控制存储器由4K字组成。其中（0000-07FF）的前半部分包含108位微指令，而其余部分（0800-0FFF）用于存储126位微指令。格式如图21.16所示.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **35** | | |  |  |
|  | **P** |  | **AA** | | | | **AB** |  | **AC** | | **AD** | | | |  |  | **AE** | | | |  |  | **AF** |  |  | **AG** | |  | **AH** |  |  | **AJ** | |  |  |  | **AK** |  | **AL** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **A, B, C, D registers** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  | **Arithmetic** | | | |  |  |  |  |  |  |  | **Shift** | | |  |  |  |  |  |
| **36** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **71** | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **P** |  |  |  |  |  | **BA** |  |  |  | **BB** | | | |  |  | **BC** | | | |  |  | **BD** |  |  | **BE** |  |  | **BF** |  |  |  |  |  |  |  |  | **BH** | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Next address** | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Storage address** | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **72** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **107** | | |  |  |
|  | **P** |  | **BH** | |  |  | **CA** | **CB** |  | **CC** |  |  |  |  | **CD** | | | |  |  |  |  | **CE** |  |  | **CF** | |  |  | **CG** | | | |  |  |  |  | **CH** | | |  |  |
|  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **Shift control** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Miscellaneous controls** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Storage address** | | | | | | | |  | **Local storage** | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **108** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **125** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **P** |  | **DA** | |  | | **DB** |  | **DC** | | **DD** | | | |  |  |  |  |  |  | **DE** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**Testing and condition code setting**

**Figure 21.16** IBM 3033 Microinstruction Format



M21\_STAL6858\_10\_GE\_C21.indd 778  3/19/15 4:57 PM 





**21.4 / TI 8800  779**

**Table 21.6** IBM 3033 Microinstruction Control Fields

|  |  |
| --- | --- |
|  | **ALU Control Fields** |
|  |  |
| AA(3) | Load A register from one of data registers |
| AB(3) | Load B register from one of data registers |
| AC(3) | Load C register from one of data registers |
| AD(3) | Load D register from one of data registers |
| AE(4) | Route specified A bits to ALU |
| AF(4) | Route specified B bits to ALU |
| AG(5) | Specifies ALU arithmetic operation on A input |
| AH(4) | Specifies ALU arithmetic operation on B input |
| AJ(1) | Specifies D or B input to ALU on B side |
| AK(4) | Route arithmetic output to shifter |
| CA(3) | Load F register |
| CB(1) | Activate shifter |
| CC(5) | Specifies logical and carry functions |
| CE(7) | Specifies shift amount |
|  | **Sequencing and Branching Fields** |
| AL(1) | End operation and perform branch |
| BA(8) | Set high-order bits (00–07) of control address register |
| BB(4) | Specifies condition for setting bit 8 of control address register |
| BC(4) | Specifies condition for setting bit 9 of control address register |
| BD(4) | Specifies condition for setting bit 10 of control address register |
| BE(4) | Specifies condition for setting bit 11 of control address register |
| BF(7) | Specifies condition for setting bit 12 of control address register |
|  |  |



虽然这是一种相当水平的格式，但是编码仍然被广泛使用。该格式的关键字段总结在表21.6中.

ALU对来自四个专用的、非用户可见的寄存器A、B、C和D的输入进行操作。微指令格式包含用于从用户可见的寄存器加载这些reg-ister、执行ALU功能和指定一个用户可见的寄存器来存储结果的字段。还有用于在寄存器和存储器之间加载和存储数据的字段.

IBM 3033的测序机制在第21.2节中进行了讨论.

**21.4 TI 8800**

德克萨斯仪器8800软件开发委员会（SDB）是一个可微程序化的32位计算机卡.



M21\_STAL6858\_10\_GE\_C21.indd 779  3/19/15 4:57 PM 





**780 Chapter 21 / Microprogrammed Control**

系统有一个可写的控制存储器，用RAM而不是只读存储器实现。这样的系统不能达到用只读存储器控制存储器的微程序系统的速度或密度。但是，它对于开发原型和用于教育目的很有用。.

8800SDB由以下组件组成（图21.17）:

* 微码存储器
* 微测序仪
* 32位ALU
* 浮点与整数处理器
* 局部数据存储器

两个总线连接系统的内部组件。DA总线将数据从微指令数据字段提供给ALU、浮点处理器或微序列器。在后一种情况下，数据由用于分支指令的地址组成.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | **15** |  | **Next microcode address** | | | | |  |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **Microcode memory** | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **32K** × **128 bits** | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **128 Microinstruction** | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **Microinstruction** | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **pipeline register** | | |  |  |  |  |  |  |  |  |
| **Control and** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **microinstruction** | | | **96** | |  |  | **DA31-DA00** | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | **32** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **ACT8832** | | |  |  |  | **ACT8847** | |  |  | **ACT8818** | | | |  |  |
|  |  | **ƒoating-point and** | | |  |  |  |  |
| **registered ALU** | | |  |  |  |  | **microsequencer** | | | |  |  |
|  |  | **integer processor** | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **32** | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **System Y bus** | | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Local data** | | |  |  |  | **PC/AT** | |  |  |  |  |  |  |  |  |
| **memory** | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **interface** | |  |  |  |  |  |  |  |  |
| **32K** × **32 bits** | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**16**



**Figure 21.17** TI 8800 Block Diagram



M21\_STAL6858\_10\_GE\_C21.indd 780  3/19/15 4:57 PM 





**21.4 / TI 8800  781**

总线也可用于ALU或微序列向其他组件提供数据。系统Y总线通过PC接口将ALU和浮点处理器连接到本地存储器和外部模块。

该板适合于IBM PC兼容的主机计算机。上位机为微代码的组装和调试提供了一个合适的平台.

**微指令格式**

8800的微指令格式由128位组成，分成30个功能字段，如表21.7所示。每个字段由一个或多个位组成，并且这些字段被分成五个主要类别:

* 板的控制
* 8847浮点与整数处理器芯片
* 8832注册ALU
* 8818微测序仪
* WEX-WCS数据场

如图21.17所示，WCS数据字段的32位被馈送到DA总线，作为数据提供给ALU、浮点处理器或微序列器。微指令的其它96位（字段1-27）是直接馈送到适当模块的控制信号。为了简单起见，图21.17中没有显示这些其他连接.

前六个字段处理与板控制相关的操作，而不是控制单个组件。控制操作包括:



* 选择用于定序器控制的条件码。字段1的第一位指示条件标志是要设置为1还是0，其余4位指示要设置哪个标志.
* 发送一个I/O请求到PC/AT.
* 启用本地数据存储器读/写操作.
* 确定系统Y总线的驱动单元。选择连接到总线的四个设备之一（图21.17）.

最后32位是数据字段，它包含特定于特殊微指令的信息.

微指令的其余字段最好在它们控制的设备的上下文中讨论。在本节的其余部分中，我们将讨论微测序器和已注册的ALU。浮点单元没有引入新的概念，因此被跳过.

**微测序仪**

8818微序列器的主要功能是为微程序生成下一个微结构地址。这个15位地址被提供给微代码存储器（图21.17）.

下一个地址可以从五个来源中的一个选择:

1. 微程序计数器（MPC）寄存器，用于重复（重用同一地址）和继续（递增地址1）指令.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M21\_STAL6858\_10\_GE\_C21.indd 781 |  |  | 3/19/15 4:57 PM |  |
|  |  |  |
|  |  |  |  |  |





**782 Chapter 21 / 微程序控制**

**Table 21.7** TI 8800 Microinstruction Format

|  |  |  |
| --- | --- | --- |
| **Field Number** | **Number of Bits** | **Description** |
|  |  |  |
|  |  | **Control of Board** |
| 1 | 5 | Select condition code input |
| 2 | 1 | Enable/disable external I/O request signal |
| 3 | 2 | Enable/disable local data memory read/write operations |
| 4 | 1 | Load status/do no load status |
| 5 | 2 | Determine unit driving Y bus |
| 6 | 2 | Determine unit driving DA bus |
|  | **8847 Floating-Point and Integer Processing Chip** | |
| 7 | 1 | C register control: clock, do not clock |
| 8 | 1 | Select most significant or least significant bits for Y bus |
| 9 | 1 | C register data source: ALU, multiplexer |
| 10 | 4 | Select IEEE or FAST mode for ALU and MUL |
| 11 | 8 | Select sources for data operands: RA registers, RB regis- |
|  |  | ters, P register, 5 register, C register |
| 12 | 1 | RB register control: clock, do not clock |
| 13 | 1 | RA register control: clock, do not clock |
| 14 | 2 | Data source confirmation |
| 15 | 2 | Enable/disable pipeline registers |
| 16 | 11 | 8847 ALU function |
|  |  | **8832 Registered ALU** |
| 17 | 2 | Write enable/disable data output to selected register: |
|  |  | most significant half, least significant half |
| 18 | 2 | Select register file data source: DA bus, DB bus, ALU Y |
|  |  | MUX output, system Y bus |
| 19 | 3 | Shift instruction modifier |
| 20 | 1 | Carry in: force, do not force |
| 21 | 2 | Set ALU configuration mode: 32, 16, or 8 bits |
| 22 | 2 | Select input to 5 multiplexer: register file, DB bus, MQ |
|  |  | register |
| 23 | 1 | Select input to R multiplexer: register file, DA bus |
| 24 | 6 | Select register in file C for write |
| 25 | 6 | Select register in file B for read |
| 26 | 6 | Select register in file A for write |
| 27 | 8 | ALU function |
|  |  | **8818 Microsequencer** |
| 28 | 12 | Control input signals to the 8818 |
|  |  | **WCS Data Field** |
| 29 | 16 | Most significant bits of writable control store data field |
| 30 | 16 | Least significant bits of writable control store data field |
|  |  |  |



M21\_STAL6858\_10\_GE\_C21.indd 782  3/19/15 4:57 PM 





**21.4 / TI 8800  783**

* 该堆栈支持微程序子例程调用以及迭代循环并从中断返回.
* DRA和DRB端口，它们提供来自外部硬件的两条附加路径，通过它们可以生成微程序地址。这两个端口分别连接到DA总线的最高有效位和最低有效位16位。这允许微序列器从当前微指令的WCS数据字段或从ALU计算的结果获得下一个指令地址.
* 寄存器计数器RCA和RCB，可用于附加地址存储.
* 双向Y端口上的外部输入以支持外部中断.

图21.18是8818的逻辑框图。该装置由以下主要官能团组成:

* 一种16位微程序计数器（MPC），由一个寄存器和一个增量器组成.
* 两个寄存器计数器，RCA和RCB，用于计数循环和迭代、存储分支地址或驱动外部设备.
* 一个65字16位的堆栈，允许微程序子程序调用和中断.
* 中断返回寄存器和Y输出使得能够在微指令级进行中断处理.
* Y输出多路复用器，通过它可以从MPC、RCA、RCB、外部总线DRA和DRB或堆栈中选择下一个地址.



***寄存器/计数器*** 寄存器RCA和RCB可以从DA总线加载，或者从当前微指令加载，或者从ALU的输出加载。这些值可以用作控制执行流的计数器，并且可以在访问时自动递减。这些值还可以用作提供给Y输出多路复用器的微指令地址。除了同时减少两个寄存器之外，支持在单个微指令周期中独立控制两个寄存器.

***堆栈*** 堆栈允许多层嵌套调用或中断，并且可以用于支持分支和循环。请记住，这些操作指的是控制单元，而不是整个处理器，并且所涉及的地址是控制存储器中的微指令的地址.

可能存在的六个堆栈操作:

* Clear，将堆栈指针设置为零，清空堆栈;
* Pop，它减少堆栈指针;
* Push，它将MPC、中断返回寄存器或DRA总线的内容放入堆栈，并递增堆栈指针;
* 读取，使得由读取指针指示的地址在Y输出多路复用器处可用;



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M21\_STAL6858\_10\_GE\_C21.indd 783 |  |  | 3/19/15 4:57 PM |  |
|  |  |  |
|  |  |  |  |  |

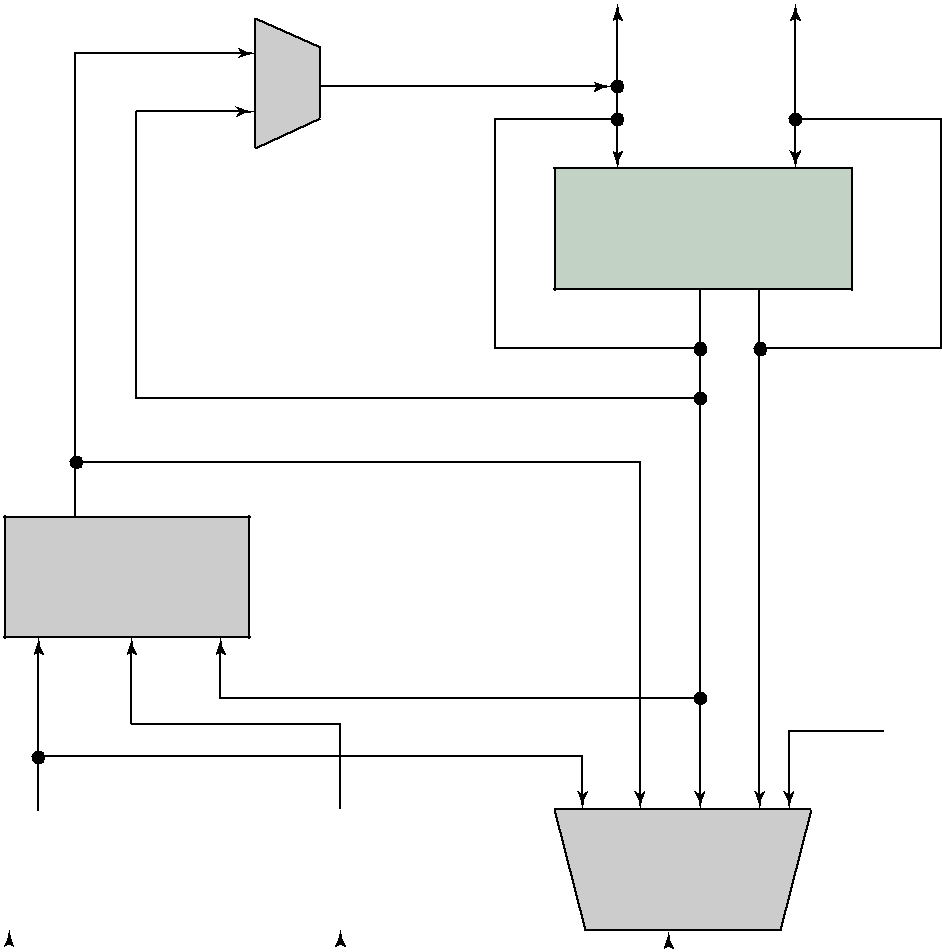




**784 Chapter 21 / 微程序控制**

**DA31-DA16** **DA15-DA00**

**(DRA)** **(DRA)**



**MUX**

**Dual**

**registers/counters**

**Stack**



**B3-B0**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Microprogram** | |  | **Interrupt** | | **Y output** | |  |
| **counter/** | |  | **return** | |  |
|  | **multiplexer** | |  |
| **incrementer** | |  | **register** | |  |
|  |  |  |  |
|  |  |  |  |  |  | **Next microde** |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  | **address** |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |



**Figure 21.18** TI 8818 Microsequencer

* Hold，这使堆栈指针的地址保持不变;
* 加载堆栈指针，它将DRA的七个最低有效位输入堆栈指针.

***微序列器的控制*** 微序列器主要由当前微指令的字段28（表21.7）的12位字段控制。该字段由以下子字段组成:

* **OSEL (1 bit):** 输出选择。确定将哪个值放在多路复用器的输出端，该多路复用器输入DRA总线（图21.18的左上角）。选择输出来自堆栈或寄存器RCA。DRA随后用作Y输出多路复用器或寄存器RCA的输入。



M21\_STAL6858\_10\_GE\_C21.indd 784  3/19/15 4:57 PM 





**21.4 / TI 8800  785**

1. **SELDR (1 bit):** 选择DR总线。如果设置为1，则该位选择外部DA总线作为DRA/DRB总线的输入。如果设置为0，则选择DRA多路复用器到DRA总线（由OSEL控制）的输出以及RCB到DRB总线的内容.
2. **ZEROIN (1 bit):** 用于表示条件分支。然后，微测序器的行为将取决于字段1中选择的条件代码（表21.7）.
3. **RC2–RC0 (3 bits):** 寄存器控制。这些位决定寄存器RCA和RCB内容的变化。每个寄存器可以保持相同、递减或来自DRA/DRB总线的负载.
4. **S2–S0 (3 bits):** 堆栈控件。这些位决定要执行哪个堆栈操作.
5. **MUX2–MUX0:** 输出控件。这些位以及条件码（如果使用的话）控制Y输出多路复用器，因此控制下一个微仪器地址。多路复用器可以从堆栈、DRA、DRB或MPC中选择其输出.

这些位可以由程序员单独设置。然而，这通常是不做的。相反，程序员使用的助记符等同于通常需要的位模式。表21.8列出了字段28的15助记符。微代码汇编程序将这些转换为适当的位模式.



**Table 21.8** TI 8818 Microsequencer Microinstruction Bits (Field 28)

|  |  |  |
| --- | --- | --- |
| **Mnemonic** | **Value** | **Description** |
|  |  |  |
| RST8818 | 000000000110 | Reset Instruction |
| BRA88181 | 011000111000 | Branch to DRA Instruction |
| BRA88180 | 010000111110 | Branch to DRA Instruction |
| INC88181 | 000000111110 | Continue Instruction |
| INC88180 | 001000001000 | Continue Instruction |
| CAL88181 | 010000110000 | Jump to Subroutine at Address Specified by DRA |
| CAL88180 | 010000101110 | Jump to Subroutine at Address Specified by DRA |
| RET8818 | 000000011010 | Return from Subroutine |
| PUSH8818 | 000000110111 | Push Interrupt Return Address onto Stack |
| POP8818 | 100000010000 | Return from Interrupt |
| LOADDRA | 000010111110 | Load DRA Counter from DA Bus |
| LOADDRB | 000110111110 | Load DRB Counter from DA Bus |
| LOADDRAB | 000110111100 | Load DRA/DRB |
| DECRDRA | 010001111100 | Decrement DRA Counter and Branch If Not Zero |
| DECRDRB | 010101111100 | Decrement DRB Counter and Branch If Not Zero |
|  |  |  |



M21\_STAL6858\_10\_GE\_C21.indd 785  3/19/15 4:57 PM 





**786 Chapter 21 / 微程序控制**

例如，如果当前选择的条件代码是1，则指令INC88181用于使下一个按顺序的微结构被选择.

INC88181 = 000000111110

从表21.8中，我们可以直接解码到

* **OSEL** = **0:** 选择RCA作为DRA输出MUX的输出；在这种情况下，选择是不相关的.
* **SELDR** = **0:** 如前所述；同样，这与此指令无关.
* **ZEROIN** = **0:** 结合MUX的值，表明不应采取分支.
* **R** = **000:** 保持RA和RC的电流值.
* **S** = **111:** 保持堆栈的当前状态.
* **MUX** = **110:** 当条件码=1时选择MPC，当条件码=0时选择DRA.

**注册ALU**

8832是具有64个寄存器的32位ALU，可以配置为作为四个8位ALU、两个16位ALU或单个32位ALU操作.

8832由构成微指令的字段17至27的39位控制（表21.7）；这些位作为控制信号提供给ALU。此外，如图21.17所示，8832具有到32位DA总线和32位系统Y总线的外部连接。来自DA的输入可以同时作为输入数据提供给64字寄存器文件和ALU逻辑模块。从系统Y总线的输入被提供给ALU逻辑模块。ALU和移位操作的结果输出到DA总线或系统Y总线。结果还可以反馈到内部寄存器文件.



三个6位地址端口允许在寄存器文件中同时执行两个操作数提取和操作数写入。MQ移位器和MQ正则化器也可以被配置为独立地执行双精度8位、16位和32位移位操作.

每个微指令的字段17至26控制数据在8832内以及在8832与外部环境之间流动的方式。字段如下:

* **写使能。这两个位指定写入32位、16个最高有效位、16个最低有效位，或者不写入寄存器文件。目的地寄存器由字段24定义**.
* **选择寄存器文件数据源。如果要对寄存器文件进行写入，那么这两个位指定源：DA总线、DB总线、ALU输出或系统Y总线**.
* **移位指令修改器。指定有关提供在移位指令期间移位的末端填充位和读取位的选项**.
* **随身携带。该位指示是否为此操作将位携带到ALU**.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M21\_STAL6858\_10\_GE\_C21.indd 786 |  |  | 3/19/15 4:57 PM |  |
|  |  |  |
|  |  |  |  |  |





**21.4 / TI 8800  787**

1. **ALU配置模式。8832可以配置为作为单个32位ALU、两个16位ALU或四个8位ALU操作**.
2. **S输入。ALU逻辑模块输入由两个称为S和R多路复用器的内部多路复用器提供。这个字段选择S多路复用器提供的输入：寄存器文件、DB总线或MQ寄存器。源寄存器由字段25定义**.
3. **R输入。选择要由R多路复用器提供的输入：寄存器文件或DA总线**.
4. **目的地寄存器。寄存器文件中用于目标操作数的寄存器地址**.
5. **源寄存器。由S多路复用器提供的用于源操作数的寄存器文件中的寄存器地址**.
6. **源寄存器。寄存器文件中用于源操作数的寄存器地址，由R多路复用器提供**.

最后，字段27是一个8位的操作码，它指定由ALU执行的算术或逻辑功能。表21.9列出了可以执行的不同操作.

作为用于指定字段17至27的编码的示例，考虑将寄存器1的内容添加到寄存器2并将结果放入regis-ter 3中的指令。符号指令是

CONT11 [17], WELH, SELRYFYMX, [24], R3, R2, R1, PASS + ADD 该指令的各个组成部分可以描述如下:



* CONT11是基本的NOP指令.
* 字段[17]被改为WELH（写使能，低和高），从而写入32位寄存器.
* 将字段[18]改为SELRFYMX以从ALU Y MUX输出中选择反馈.
* 字段[24]被更改为指定目的地寄存器的寄存器R3.
* 字段[25]被改变为指定源寄存器之一的寄存器R2.
* 字段[26]被改变为指定源寄存器之一的寄存器R1.
* 字段[27]被更改为指定ADD的ALU操作。ALU移位器指令是PASS；因此，ALU输出不会被移位器移位.

关于符号记数法可以说几点。没有必要为连续字段指定字段号。也就是说,

CONT11 [17],WELH,[18],SELRFYMX

可以写为

CONT11 [17],WELH,SELRFYMX

因为SELRYMYX在字段18中.

表21.9第1组的ALU指令必须始终用于与第2组的连接。组3到组的ALU指令不能用于组2.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M21\_STAL6858\_10\_GE\_C21.indd 787 |  |  | 3/19/15 4:57 PM |  |
|  |  |  |
|  |  |  |  |  |





**788 Chapter 21 / 微程序控制**

**Table 21.9** TI 8832 Registered ALU Instruction Field (Field 27)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Group 1** | | **Function** |
| ADD |  | H#01 | R + S + Cn |
| SUBR |  | H#02 | (NOT R) + S + Cn |
| SUBS |  | H#03 | R = (NOT S) + Cn |
| INSC |  | H#04 | S + Cn |
| INCNS |  | H#05 | (NOT S) + Cn |
| INCR |  | H#06 | R + Cn |
| INCNR |  | H#07 | (NOT R) + Cn |
| XOR |  | H#09 | R XOR S |
| AND |  | H#0A | R AND S |
| OR |  | H#0B | R OR S |
| NAND |  | H#0C | R NAND S |
| NOR |  | H#0D | R NOR S |
| ANDNR |  | H#0E | (NOT R) AND S |
|  | **Group 2** | | **Function** |
| SRA |  | H#00 | Arithmetic right single precision shift |
| SRAD |  | H#10 | Arithmetic right double precision shift |
| SRL |  | H#20 | Logical right single precision shift |
| SRLD |  | H#30 | Logical right double precision shift |
| SLA |  | H#40 | Arithmetic left single precision shift |
| SLAD |  | H#50 | Arithmetic left double precision shift |
| SLC |  | H#60 | Circular left single precision shift |
| SLCD |  | H#70 | Circular left double precision shift |
| SRC |  | H#80 | Circular right single precision shift |
| SRCD |  | H#90 | Circular right double precision shift |
| MQSRA |  | H#A0 | Arithmetic right shift MQ register |
| MQSRL |  | H#B0 | Logical right shift MQ register |
| MQSLL |  | H#C0 | Logical left shift MQ register |
| MQSLC |  | H#D0 | Circular left shift MQ register |
| LOADMQ |  | H#E0 | Load MQ register |
| PASS |  | H#F0 | Pass ALU to Y (no shift operation) |
|  | **Group 3** | | **Function** |
| SET1 |  | H#08 | Set bit 1 |
| Set0 |  | H#18 | Set bit 0 |
| TB1 |  | H#28 | Test bit 1 |
| TB0 |  | H#38 | Test bit 0 |
| ABS |  | H#48 | Absolute value |
| SMTC |  | H#58 | Sign magnitude/twos-complement |
|  |  |  |  |



M21\_STAL6858\_10\_GE\_C21.indd 788  3/19/15 4:57 PM 





**21.4 / TI 8800  789**

|  |  |  |  |
| --- | --- | --- | --- |
| **Group 3** | |  | **Function** |
| ADDI |  | H#68 | Add immediate |
| SUBI |  | H#78 | Subtract immediate |
| BADD |  | H#88 | Byte add R to S |
| BSUBS |  | H#98 | Byte subtract S from R |
| BSUBR |  | H#A8 | Byte subtract R from S |
| BINCS |  | H#B8 | Byte increment S |
| BINCNS |  | H#C8 | Byte increment negative S |
| BXOR |  | H#D8 | Byte XOR R and S |
| BAND |  | H#E8 | Byte AND R and S |
| BOR |  | H#F8 | Byte OR R and S |
| **Group 4** | |  | **Function** |
| CRC |  | H#00 | Cyclic redundancy character accum. |
| SEL |  | H#10 | Select S or R |
| SNORM |  | H#20 | Single length normalize |
| DNORM |  | H#30 | Double length normalize |
| DIVRF |  | H#40 | Divide remainder fix |
| SDIVQF |  | H#50 | Signed divide quotient fix |
| SMULI |  | H#60 | Signed multiply iterate |
| SMULT |  | H#70 | Signed multiply terminate |
| SDIVIN |  | H#80 | Signed divide initialize |
| SDIVIS |  | H#90 | Signed divide start |
| SDIVI |  | H#A0 | Signed divide iterate |
| UDIVIS |  | H#B0 | Unsigned divide start |
| UDIVI |  | H#C0 | Unsigned divide iterate |
| UMULI |  | H#D0 | Unsigned multiply iterate |
| SDIVIT |  | H#E0 | Signed divide terminate |
| UDIVIT |  | H#F0 | Unsigned divide terminate |
| **Group 5** | |  | **Function** |
| LOADFF |  | H#0F | Load divide/BCD flip-flops |
| CLR |  | H#1F | Clear |
| DUMPFF |  | H#5F | Output divide/BCD flip-flops |
| BCDBIN |  | H#7F | BCD to binary |
| EX3BC |  | H#8F | Excess (3 byte correction |
| EX3C |  | H#9F | Excess (3 word correction |
| SDIVO |  | H#AF | Signed divide overflow test |
| BINEX3 |  | H#DF | Binary to excess - 3 |
| NOP32 |  | H#FF | No operation |
|  |  |  |  |



M21\_STAL6858\_10\_GE\_C21.indd 789  3/19/15 4:57 PM 





**790 Chapter 21 / 微程序控制**

**21.5 关键词、思考题和习题**

**关键词**

|  |  |  |
| --- | --- | --- |
| 控制存储器 | 微指令编码 | 微程序控制器 |
| 控制字 | 微指令执行 | 微程序设计语言 |
| 固件 | 微指令排序 | 软件微程序设计 |
| 硬微程序设计 | 微指令 | 解包微指令 |
| 水平微指令 | 微程序 | 垂直微指令 |
|  |  |  |

**思考题**

**21.1** 控制单元的硬连线实现和微程序实现有什么区别?

**21.2** 水平微指令是如何解释的?

**21.3** 控制地址寄存器和控制缓冲寄存器有什么区别?

**21.4** 执行水平微指令的典型顺序是什么?

**21.5** 水平微指令和垂直微指令有什么区别?

**21.6** 描述微程序单元的优点和缺点.

**21.7** 打包和未打包的微指令有什么区别?



**21.8** 硬微程序和软微程序有什么区别?

**21.9** 功能编码和资源编码有什么区别?

**21.10** 列出了微程序设计的一些常见应用.

**习题**

**21.1** 描述在Wilkes设计的虚拟机中乘法指令的实现。使用叙事和流程图.

**21.2** 假设微指令集包括具有以下符号形式的微指令:

IF (AC0 = 1) THEN CAR d (C0 - 6) ELSE CAR d (CAR) + 1

其中AC0是累加器的符号位，C0-6是微指令的前七位。使用这个微指令，编写一个实现分支寄存器减号（BRM）机器指令的微程序，如果AC为负，则分支寄存器减号（BRM）机器指令。假设微指令的位C1到Cn指定一组并行的微操作。象征性地写出程序.

**21.3** 简单处理器的指令周期有四个主要阶段：获取、间接、执行和中断。在硬连线实现中，两个1位标志指定当前阶段.

* 为什么需要这些flags?
* 为什么在微程序控制单元中不需要它们?

**21.4** 考虑图21.7的控制单元。假设控制存储器为32位宽。微指令格式的控制部分被分成两个字段. 15位的微操作字段指定要执行的微操作。地址选择字段根据标志指定导致微指令分支的条件。有十六个flags.



M21\_STAL6858\_10\_GE\_C21.indd 790  3/19/15 4:57 PM 





**21.5 / 关键词，思考题和习题   791**

1. 地址选择字段中有多少位?
2. 地址字段中有多少位?
3. 控制内存的大小是多少?

**21.5** 在前一个问题的情况下，如何进行无条件分支？如何避免分支；即，描述不指定任何分支（有条件或无条件）的微指令.

**21.6** 我们希望为每个机器指令例程提供8个控制字。机器指令操作码有5位，控制存储器有1024个字。建议从指令寄存器到控制地址寄存器的映射.

**21.7** 将使用编码的微指令格式。显示如何将10位微操作字段划分为子字段以指定78个不同的操作.

**21.8** 处理器有16个寄存器，一个具有16个逻辑和16个算术功能的ALU，以及一个具有8个操作的移位器，所有这些都通过内部处理器总线连接。设计微指令格式来指定处理器的各种微操作.

