### 实验四 微程序控制器

一、实验目的

1．理解微程序控制器的控制原理

2．进一步掌握指令流程和功能

3．了解掌握微程序控制器的设计思路和方法

二、实验原理

微程序控制器的设计思想是由英国剑桥大学的威尔克斯（Wilkes）教授于1951年提出来的，即将机器指令的操作（从取指令到执行）分解成若干个更基本的微操作序列，并将有关的控制信号（微命令）按照一定的格式编成微指令，存放到一个只读存储器中，当机器运行时，一条一条地读出这些微指令，从而产生全机所需要的各种操作控制信号，使相应部件执行所规定的操作。

微程序控制器同硬布线控制器相比较，具有规整、灵活和可维护一系列优点，已被广泛应用，在计算机设计中逐渐取代了早期采用的硬布线控制器。在计算机系统中，微程序设计技术是利用软件方法来设计硬件的一门技术。

1．有关的术语和概念

一台计算机基本上可以分为两大部分：控制部件和执行部件。控制器就是控制部件，而运算器、存储器和外部设备相对控制器来说就是执行部件。

（1）微命令。微命令是构成控制信号序列的最小单位，通常是指那些能直接作用于某部件控制门的命令，如打开或关闭某部件通路的控制门的电位以及某寄存器、触发器的打入脉冲等。微命令由控制部件通过控制总线向执行部件发出。

（2）微操作。微操作是由微命令控制实现的最基本的操作。

微命令是微操作的控制信号，微操作是微命令的执行过程。在计算机内部实质上是同一个信号，对控制部件为微命令，对执行部件为微操作。很多情况下两者常常不加区分地使用。

（3）微指令。微指令是一组实现一定操作功能的用二进制编码表示的微命令的组合。

（4）微周期。微周期是从控制存储器读取一条微指令并执行相应的微操作所需的时间。

（5）微程序。微程序是一系列微指令的有序集合。

2. 微程序控制器的组成原理框图

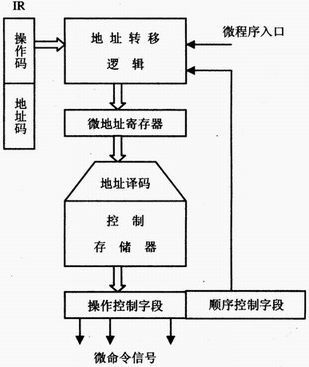
微程序控制器的组成原理控制框图如图4.1所示。它主要由控制存储器、微地址寄存器、微命令寄存器和地址转移逻辑4部分组成。

（1）控制存储器

控制存储器用来实现整个指令系统的所有微程序。一般计算机的指令系统是固定的，所以实现指令系统的微程序也是固定的，因此控制存储器微程序也是固定的，因此控制存储器通常由高速半导体只读存储器构成，其存储容量视机器指令系统而定，即取决于微程序的数量，其字长就是微指令字的长度。

2）微指令寄存器

微指令寄存器用来存放从控制存储器读出的当前微指令。微指令中包含两个字段，即微操作控制字段和微地址字段，微操作控制字段将操作控制信号送到控制信号线上，并提供判别测试字段，微地址字段用于控制下一条微指令地址的形成。

（3）微地址寄存器

微地址寄存器用来存放将要访问的下一条微指令的地址。

（4）地址转移逻辑

地址转移逻辑用来形成将要执行的微指令的地址。其形成方式一般有以下几种：取指令公共操作所对应的微程序一般从控制存储器的0号单元开始存放，所以微程序的人口地址0是由硬件强制规定的；当微程序不出现分支，则微指令从控制存储器读出后直接给出下一条微指令的地址；当微程序出现分支时，通过判别测试字段、微地址字段和执行部件的反馈信息形成后即微地址，包括根据操作码转移的情况。

图4.1 微程序控制器的组成原理框图

3．微程序控制器执行过程描述

在采用微程序控制的计算机中，若整个指令系统对应的微程序已放人控制存储器中，则它的执行过程可描述如下。

（1）从控制存储器中逐条取出“取机器指令”用的微指令，执行取指令公共操作，执行完后，从主存中取出的机器指令就已存入指令寄存器中了。

（2）根据指令寄存器中的操作码，经过微地址形成部件，得到这条指令对应的微程序入口地址，并送人微地址寄存器。

（3）从控制存储器中逐条取出对应的微指令并执行。

（4）执行完对应于一条机器指令的一段微程序后又回到取指微程序的入口地址，继续第l）步，以完成取下一条机器指令的公共操作。

4．微指令编码法

微指令可以分为操作控制字段和下地址字段两大部分。这里所说的微指令编码法就是操作控制字段的编码法，通常有如下3种方法。

（1）直接控制法

所谓直接控制法，就是在微指令的操作控制字段中，每一个微命令都用一位信息表示，这样是否发出某个微命令，只要将控制字段中表示该命令的相应位设置成“l”或“0”，就可打开或关闭某个控制门，因此，微命令的产生不必经过译码，就可从微操作控制字段直接得到，故有时称为不译码法，如图4.2所示。

这种编码法的优点是控制简单、直观，操作并行性最好，从而可以提高速度。其缺点是微指令字太长，控制存储器的容量过大且微指令字利用效率很低。因此这种编码方法只适用于结构简单或速度要求很高的高速数字控制部件。

（2）最短字长编码

这种编码法是将所有的微命令进行统一的二进制编码，用不同的码点去表示不同的微命令，通过译码器产生微操作控制信号，如图4.3所示。

这种编码法的优点是微指令字长很短，但是因为它每次只能产生一个微命令，所有微命令均不能够并行，难以提高微命令的执行效率，故在实际应用中很少采用。

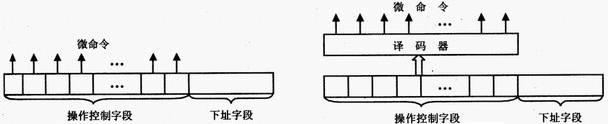


图4.2 直接控制编码 图4.3 最短字长编码

（3）分段直接编码

这种编码法是将微操作控制字段划分为若干个小字段，每个小字段独立译码，每个码点表示一个微命令，其微指令结构如图4.4所示。

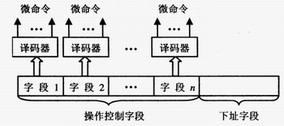


图4.4 分段直接编码的微指令结构

字段直接编码法吸收了直接控制编码和最短字长编码两种方法的优点，既能缩短微指令字长，又有较高的并行性，执行速度比较快，因此得到了广泛的应用。

三、实验内容

实验中，我们设计了一个简单的微处理器，整个设计分成控制器和数据通路，执行简单的直接寻址指令。为了简化微处理器的设计，我们假定只有一条总线，且总线和所有数据通路组件的宽度都是8位。由于单总线可能会被许多不同的组件驱动，每个组件需要使用三态缓冲器以确保在任一时刻仅有一个组件能将有效数据送至总线上。我们用一个时钟驱动所有的时序块以确保设计完全同步。微处理器的模块图如图4.5所示。

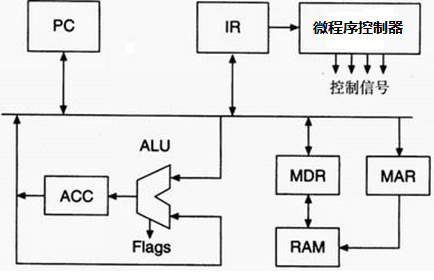


图4.5 CPU的数据通路

由微处理器执行的程序与数据一起存储于存储器中，存储器地址寄存器（MAR）和数据寄存器（MDR）作为地址和数据信号在存储器与总线间的缓冲器。

算术逻辑单元（ALU）执行算术操作（ADD、SUB）。算术逻辑单元是一个组合模块，算术操作的结果保存于一个称为累加器（ACC）的寄存器中。ALU的输入是总线和ACC。ALU可以有更多的输出或者标志来表示ACC中结果的性质，比如为负，这些标志作为控制器的输入。

程序的不同指令在存储器中是顺序存储的。因此需要保存下一条将要执行的指令的地址。这是使用程序计数器（PC）来完成的，如果执行一个分支，程序就要跳出顺序执行，所以必须加载一个新的地址到PC中。

最后，从存储器中读取的指令需要保存与执行。指令寄存器（IR）保存当前指令。操作码的相应位输入至控制器，以产生相应的控制信号。

控制器产生许多控制信号，这些信号决定写总线的组件、读总线的寄存器以及执行的ALU操作。表4.1列出了这个微处理器的控制信号。

表4.1 微处理器的控制信号

| 控制信号 | 描述 |
| --- | --- |
| ACC\_bus | 用ACC的内容驱动总线 |
| load\_ACC | 将总线上的数据载入ACC |
| PC\_bus | 用PC的内容驱动总线 |
| load\_IR | 将总线上的数据装载至IR |
| load\_MAR | 将总线上的数据装载至MAR |
| MDR\_bus | 用MDR的内容驱动总线 |
| load\_MDR | 将总线上的数据装载至MDR |
| ALU\_ACC | 用ALU的结果装载ACC |
| INC\_PC | PC+1并将结果存至PC中 |
| Addr\_bus | 用IR中指令的操作码部分驱动总线 |
| CS | 片选。用MAR的内容设置存储器地址 |
| R\_NW | 读取，不可写。当R\_NW无效且CS有效时，MDR的内容存储于存储器中 |
| ALU\_add | 在ALU中执行加法操作 |
| ALU\_sub | 在ALU中执行减法操作 |

本微处理器的指令系统中包含5条指令：Load、Store、add、sub、bne。整个指令系统对应的微程序存放在控制存储器中，微程序编码见表4.2。

根据微处理器的指令系统和数据通路，绘制出微程序流程图如图4.6所示。图中方框边上的编码对应微程序编码中第一栏的微地址。

微程序的编码采用直接编码方法，有15个控制信号，外加5位地址，所以一共20位。根据图4.6的流程图中的下地址，列出了表4.2中微程序的编码。

地址3的下地址，应该根据指令操作码来形成，所以将其下地址设为01111，表示如果下地址为01111时，下地址的低3位是指令操作码，高2位固定是01（也可用其他的形成方式），因此load指令的开始地址是01000=810，store指令的开始地址是01001=910，同理add的是01010＝1010，sub的是01011＝1110，bne的是01100＝1210。

从图4.6可以明显的看出来。地址12的下地址，也要根据运算器的结果是否为0来判断，所以其下地址设为10000，以决定下地址是13还是14。至于控制信号何时应为1，可以参照图4.6。



图4.6 微程序流程图

四、实验步骤

1. 建立工程cpu，复制代码cpu.vhd、cpu\_def.vhd，编译，
2. 建立仿真文件，并对仿真结果加以说明和分析。

五、实验现象

微处理器的所有数据通路宽度是8位，操作码是3位，只有直接寻址方式，所以说地址是5位。操作码对应的编码：load －> 000，store －> 001，add －> 010，sub －> 011，bne －> 100。内存中存储的指令序列：

| 地址 | 指令 | 描述 |
| --- | --- | --- |
| 0 | load 4 | 从mem(4)取数到ACC中  dao |
| 1 | add 5 | 从mem(5)取数到总线上，与ACC中的数相加 |
| 2 | store 6 | 将相加的结果存储在mem(6)中 |
| 3 | bne 7 | 如果Z\_flag不为0，跳转到内存7地址所指的地址执行指令 |
| 4 | 2 | 数据为2 |
| 5 | 3 | 数据为3 |
| 其他 | 0 | 全为0 |

六、实验报告要求

按照ppt中指出的要点，结合如下的思考题，编制实验报告。

并对仿真结果加以说明和分析

思考题

1. 写出执行的程序。
2. 指令格式怎样？写出五条指令的机器码（用二进制表示）。
3. 分析微码格式，指出操作控制字段采用什么格式？多少位？下址字段多少位？
4. 如何形成后继地址？微码中有两处分支，一是根据OP多路分支，二是根据Z\_flag二路分支，微码设计没有采用P1和P2等测试位的方法，分析代码，回答该微码是如何实现分支的？
5. 解释某一条指令的执行过程。

（为避免抄袭，需解释第n条指令，n为学号除4的余数）

1. 解释某一条微指令的组成。

（为避免抄袭，需解释第n条指令，n为学号除15的余数）

1. 执行完bne指令后程序转到什么地方执行？

附表见下面两页

表1：微程序编码

表2：各寄存器变化过程

表4.2 微程序编码

|  |  |
| --- | --- |
| 微地址 | 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4～0 |
| load\_PC ACC\_bus load\_ACC PC\_bus load\_IR load\_MAR MDR\_bus load\_MDR ALU\_ACC INC\_PC Addr\_bus CS R\_NW ALU\_add ALU\_sub 下地址 |
| 00 | 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 00001 |
| 01 | 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 00010 |
| 02 | 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 00011 |
| 03 | 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 01111 |
| 04 | 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 00000 |
| 05 | 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 00000 |
| 06 | 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 00000 |
| 07 | 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 00000 |
| 08 | 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 00100 |
| 09 | 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 00101 |
| 10 | 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 00110 |
| 11 | 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 00111 |
| 12 | 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 10000 |
| 13 | 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 00000 |
| 14 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 00000 |

表4.3 各寄存器的变化过程

| 指令  周期 | 当前  微地址 | 总线(8位) | 程序  计数器 | (3位)  操作码 | MDR (8位)  数据寄存器 | MAR(5位)  地址寄存器 | IR(8位)  指令寄存器 | 下地址寄存器(5位) | 微码寄存器（20位） | ACC |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0：  load  4 | 0 | 0000 0000 | 0 | 000 | 00000000 | 00000 | 0000 0000 | 00001 | 00010100010000000001 | 0 |
| 1 | 0000 0000 | 1 | 000 | 000 00100 | 00000 | 0000 0000 | 00010 | 00000000000110000010 | 0 |
| 2 | 0000 0100 0 0100 | 1 | 000 | 000 00100 | 00000 | 000 00100 | 00011 | 00001010000000000011 | 0 |
| 3 | 0000 0100 | 1 | 000 | 000 00100 | 00100 | 000 00100 | 01000 | 00000100001000001111 | 0 |
| 8 | 0000 0100 | 1 | 000 | 0000 0010 | 00100 | 000 00100 | 00100 | 00000000000110000100 | 0 |
| 4 | 0000 0010 | 1 | 000 | 0000 0010 | 00100 | 000 00100 | 00000 | 00100010000000000000 | 2 |
| 1：  add  5 | 0 | 0000 0001 | 2 | 000 | 0000 0010 | 00001 | 000 00100 | 00001 | 00010100010000000001 | 2 |
| 1 | 0000 0001 | 2 | 000 | 010 00101 | 00001 | 000 00100 | 00010 | 00000000000110000010 | 2 |
| 2 | 010 00101 | 2 | 000 | 010 00101 | 00001 | 010 00101 | 00011 | 00001010000000000011 | 2 |
| 3 | 000 00101 | 2 | 010 | 010 00101 | 00101 | 010 00101 | 01010 | 00000100001000001111 | 2 |
| 10 | 000 00101 | 2 | 010 | 0000 0011 | 00101 | 010 00101 | 00110 | 00000000000110000110 | 2 |
| 6 | 0000 0011 | 2 | 010 | 0000 0011 | 00101 | 010 00101 | 00000 | 00000010100001000000 | 5 |
| 2：  store  6 | 0 | 0000 0010 | 3 | 010 | 0000 0011 | 00010 | 010 00101 | 00001 | 00010100010000000001 | 5 |
| 1 | 0000 0010 | 3 | 010 | 001 00110 | 00010 | 010 00101 | 00010 | 00000000000110000010 | 5 |
| 2 | 001 00110 | 3 | 010 | 001 00110 | 00010 | 001 00110 | 00011 | 00001010000000000011 | 5 |
| 3 | 000 00110 | 3 | 001 | 001 00110 | 00110 | 001 00110 | 01001 | 00000100001000001111 | 5 |
| 9 | 0000 0101 | 3 | 001 | 0000 0101 | 00110 | 001 00110 | 00101 | 01000001000000000101 | 5 |
| 5 | 0000 0101 | 3 | 001 | 0000 0101 | 00110 | 001 00110 | 00000 | 00000000000100000000 | 5 |
| 3：  bne  7 | 0 | 0000 0011 | 4 | 001 | 0000 0101 | 00011 | 001 00110 | 00001 | 00010100010000000001 | 5 |
| 1 | 0000 0011 | 4 | 001 | 100 00111 | 00011 | 001 00110 | 00010 | 00000000000110000010 | 5 |
| 2 | 100 00111 | 4 | 001 | 100 00111 | 00011 | 100 00111 | 00011 | 00001010000000000011 | 5 |
| 3 | 000 00111 | 4 | 100 | 100 00111 | 00111 | 100 00111 | 01100 | 00000100001000001111 | 5 |
| 12 | 000 00111 | 4 | 100 | 0000 0000 | 00111 | 100 00111 | 1101 | 00000000000110010000 | 5 |
| 13 | 0000 0000 | 0 | 100 | 0000 0000 | 00111 | 100 00111 | 00000 | 10000010000000000000 | 5 |
| 循环 | | | | | | | | | | |

