AM2900
位片式计算机设计

中国船舶工业总公司
第七研究院第七〇九研究所

位片式计算机设计

作者：（美）丁·米克　丁·布里克

译者：（按所译章节顺序为序）

尹守峻　高修忠　何家喜　叶其寿

柳开忠　刘象庆　程家骥　王仁华

付　扬　吴泰昫　张鸿海

总校　[illegible]　[illegible]宁

中国船舶工业总公司第七研究院第七〇九研究所

译 者 的 话

为了满足和适应国内位片式计算机设计的需要，中国科学院计算技术研究所和第七研究院第七〇九研究所部分同志共同翻译了“**位片式计算机设计**”一书。该书作者丁·米克和丁·布里克是AMD公司Am2900系列器件的两位主要设计师。本书是经过自76年以来多次修改，于80年最新定稿的。原书约400页，近400张图表，全书图文完整。该书系统地介绍了Am2900系列各主要功能器件的设计原理，使用原则及使用实例，对于位片式计算机设计者是一本难得的针对性很强的参考书。对于计算机科学工作者或计算机专业的教师与学生来说，在了解位片式计算机、掌握近代计算机体系功能设计技术方面也是一本很好的参考书。

全书共分九章。第一章介绍了计算机的基本结构，第二章介绍了微程序设计的原则，微程序控制器Am 2910 等器件设计原理及使用规则，第三章与第四章介绍了计算机的数据通路的设计及Am2903、Am2904等计算机的核心部件位片的工作原理及使用技巧，第五章介绍程序控制器 Am2930 的设计原理及使用方法，第六章介绍中断系统的原理及向量中断优先排队控制器 Am2914 器件的工作原理和使用规则，第七章介绍了直接存储器存取方式的原理及其相应的位片器件的工作原理，第八章介绍了一个16位的小型机HEX-29的设计资料，第九章介绍了超级16位微型机的设计资料。

本书在译校过程中，对原书中的一些错误作了更正。由于水平所限，仍可能有不少错误和不妥之处，恳请读者批评指正。

序　言

介绍新的集成电路常常先引用许多理论和数据图表，然后提出应用说明。现在，我们介绍可编微程序的LSI器件Am2901和它配套的IC，也坚持了这一做法。根据以往介绍比较复杂的有固定指令系统的MOS微处理器的成功经验，我们觉得这就足够了。但是，位片微处理器设计表明，它比原先认识的更为复杂，有更多的困难之处。难处之一就是要了解各器件之间更密切的关系。这些设计要求设计者要能挑选器件：设计需要多少个位片？应该选择什么样的微程序序列器和/或控制器？需要先行进位发生器吗？等等等等。所有这些器件必须一起发挥作用，而不是只靠那一个器件就能完成的。

通过设计上的努力，用户得到了更快的传输速度和极大的灵活性。灵活性给设计的简易性造成障碍。用户必须自己设计指令系统以及硬件和应用程序。他们不再享有现成的固定指令系统。另一方面，他们可以除掉那些不必要的指令，并能在以后容易改变和增添新的指令，或者仿真较慢的CPU的现有指令系统。

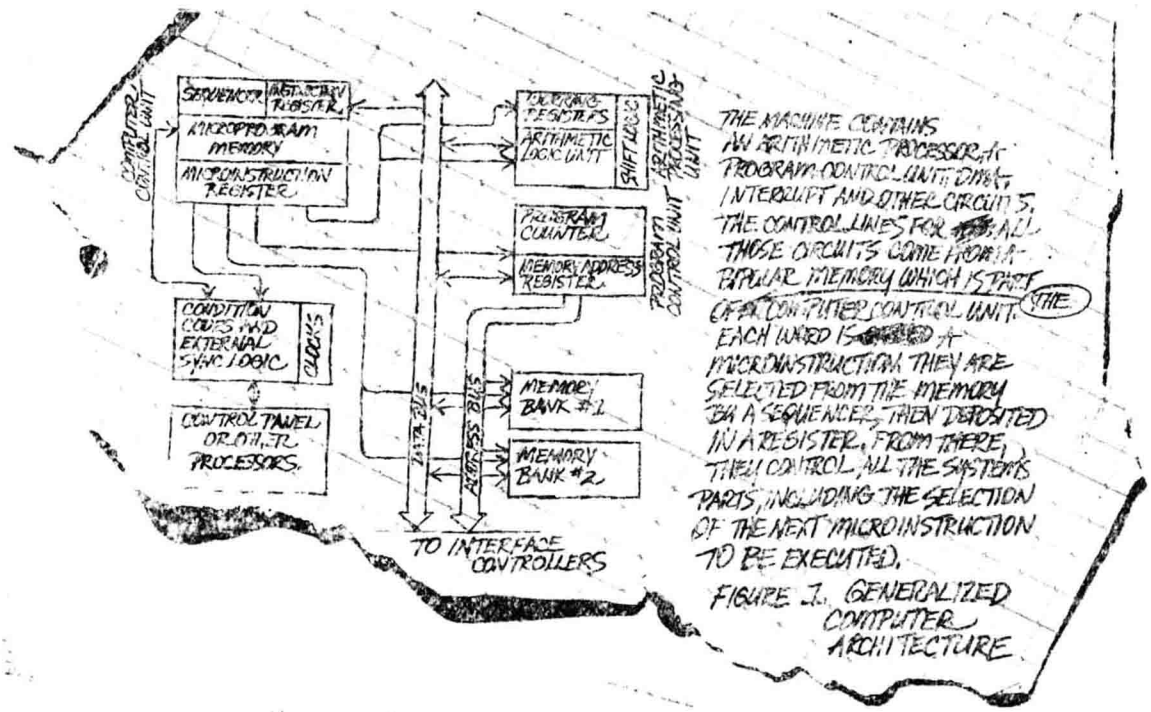
复杂的是，2900系列不是一下就整个跃入世界的，它的许多器件是在工程和处理手段的支持下经过多年才被引入和重新设计的。这种发展过程仍在继续。

为了减少事情的麻烦，AMD公司宣布了一个有九个部分组成的关于可编微程序处理机设计的教程，每个部分是相对独立的。当然，后一部分要以前一部分为基础。

自这个教程问世以来，事实说明，本教程所选用的材料都是非常必要的。实际上，双极型可编微程序LSI器件的市场在前两年中都是每年翻一番，并且尚无下降的迹象。因此，当压缩我们的教材时，我们只是想使它们能自然地容纳于一本书之内。这本书就是这样产生的。

我们认为，花费这么多时间和精力是有价值的。

目　　录


COMPUTER CONTROL UNIT
SEQUENCER
INSTRUCTION REGISTER
MICROPROGRAM MEMORY
MICROINSTRUCTION REGISTER
CONDITION CODES AND EXTERNAL SYNC LOGIC
CLOCKS
CONTROL PANEL OPERATOR PROCESSORS
DATA BUS
ADDRESS BUS
GENERAL REGISTERS
ARITHMETIC LOGIC UNIT
SHIFT LOGIC
ARITHMETIC PROCESSING UNIT
PROGRAM COUNTER
MEMORY ADDRESS REGISTER
PROGRAM CONTROL UNIT
MEMORY BANK #1
MEMORY BANK #2
TO INTERFACE CONTROLLERS
THE MACHINE CONTAINS AN ARITHMETIC PROCESSOR, A PROGRAM CONTROL UNIT, DMA, INTERRUPT AND OTHER CIRCUITS. THE CONTROL LINES FOR ALL THOSE CIRCUITS COME FROM A BIPOLAR MEMORY WHICH IS PART OF THE COMPUTER CONTROL UNIT. EACH WORD IS A MICROINSTRUCTION. THEY ARE SELECTED FROM THE MEMORY BY A SEQUENCER, THEN DEPOSITED IN A REGISTER. FROM THERE, THEY CONTROL ALL THE SYSTEM'S PARTS, INCLUDING THE SELECTION OF THE NEXT MICROINSTRUCTION TO BE EXECUTED.
FIGURE 1. GENERALIZED COMPUTER ARCHITECTURE

第 一 章

计 算 机 结 构

第一章　计算机结构

前　言

在此绪言性的章节中我们打算：

1）介绍以后各章的一些公用术语。

2）介绍有关程序存储计算机设计的若干题目。

3）确定计算机结构方面的某些（将在下面几章要解决的）问题。

为了达到这些目标，我们应从计算机的基础知识开始。必须强调指出，尽管可选择解决问题的途径和方法可能与本章和下面几章中所介绍的有所不同，但所论述的基本思想对于进一步熟悉微程序的位片式器件以便利用它们组成任何结构来说还是适用的。

回顾一下计算机基础

所谓程序存储计算机，就是它能根据事先确定的规则（指令）处理数据，而程序（指令的集合）和数据则存储在它的存储器中（图1）。没有与外部设备通讯的某些手段就既不能把程序和数据装入存储器，也无法把结果取出来。因此，输入/输出设备是必需的。如图2所示。

存储器通常是按字组织的，每字含有N位的信息。给每个字分配一个唯一的地址，以此确定该字与其他字的相对位置。中央处理部件（CPU）通常是通过对存储器的寻址访问每次读出或写入一个字，当存储器准备好就可读取这个字的内容或向这个字写入新的内容。为完成这个操作，通常需要两个寄存器：存储器地址寄存器（MAR）用来保存地址和存储器数据寄存器（MDR）用来保存数据（图3）。

因为访问存储器（读出或写入）常常很慢，所以如果在CPU内有几个可以快速读写的存储单元那是有利的。这些存储单元一般称为累加器或工作寄存器。有了这些内部快速存取的寄存器，CPU（图4）不必去访问存储器（通过MAR和MDR）就能执

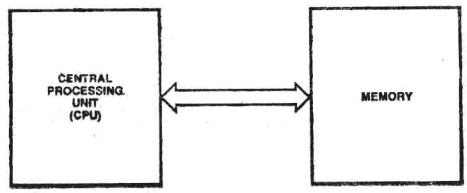


图1　程序存储计算机的基本构成

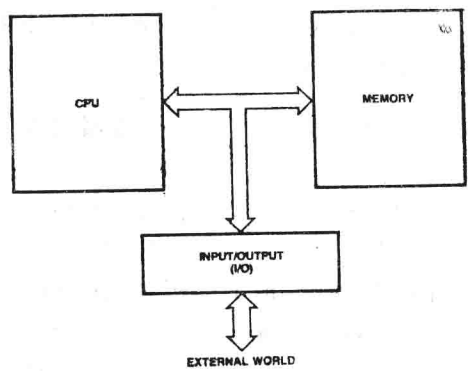


图2　添加了I/O设备的存储计算机

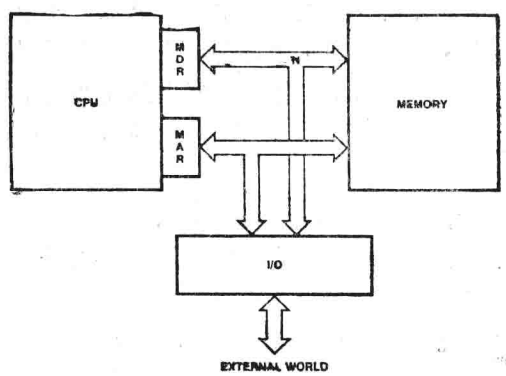


图3　标出了MAR和MDR的程序存储计算机

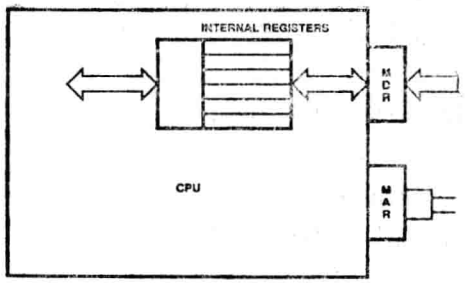


图4　带有内部高速寄存器的CPU

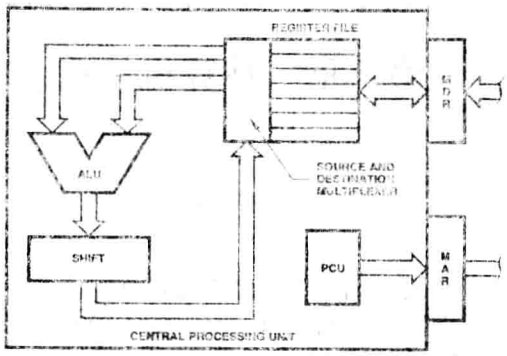


图5　添加了ALU和移位器的CPU设计

行许多操作。因此，这些操作就能快速执行。

实际执行数据加工的部件叫做算术及逻辑部件（ALU）。它有两个操作数的入口和一个结果出口。它一般是对一个字的所有位进行并行操作的。ALU能够执行下列操作的全部或一部分：

算术操作	逻辑操作
加　(Add)	或　(OR)
求补(Complement)	与　(AND)
减　(Subtract)	异或(XOR)
加1 (Increment)	非与(NAND)
减1 (Decrement)	非或(NOR)
	异非或(XNOR)
	求补(Complement)

在某些结构中，操作数之一总是在一个专用寄存器（累加器）中，而ALU操作的结果也总是送到这个寄存器去。在更通用的CPU中，通用内部寄存器中的任何两个均可保存操作数，而ALU操作的结果，可送入其中一个。

CPU另一个十分有用的特点是它能把寄存器或ALU的输出内容作一位或多位的左右移位。（如图5所示）。

现在我们有了进行数据处理所需的各个构件，但我们还需要一个部件，它能正确地给MAR置值，以便在存储器里找到程序中的下一条指令及其相关的数据。这个部件叫作程序控制部件（PCU），它的作用是把正确的地址装入MAR，以便找到下一条指令或数据项，或是指出数据字将要写入存储器的那个单元。

程序的各步(指令，数据)常常是存放在存储器的相邻的单元中，这些单元从零地址开始或从任何一个予先指定地址开始。PCU部件在每次存储器访问之后能够简单地进行加“1”，从而指向下一条指令或数据的地址，这种计数器型的PCU，灵活性很小。有时，我们想要改变这种“正规”的指令流，特别是当我们想要计算机能够根据当前执行点产生的条件“作出决定”时更觉如此。例如，我们可以根据最后完成操作的结果来执行两个不同指令序列中的一个指令序列。这是通过给MAR装入新值（即下条要执行的指令的地址）来实现，而不是给MAR加1。这个操作称作“转移”（BRANCH）或“跳转”（JUMP）。它可以是无条件的（允许执行不相邻的指令串），也可以是有条件的（例如，根据最后操作结果是否为零，是正或是负，是1或是0，等等）。

要得到更大的灵活性可以利用堆栈（一组内部的或外部的暂存单元）来存放重要的数据。堆栈指针用来指出当前的堆栈栈顶的

存储单元的地址。间接和相对寻址方式以及其他的更复杂的寻址方式(均可由 PCU 执行)将在后面讨论。同时，图5把 PCU 表示为CPU 的一部分。

在我们的计算机中执行任何一条指令现在需要下列步骤：

a) PCU 把下一条指令的地址装入 MAR，并向存储器发出信号表示请求“读出”。PCU 可以和长度等于地址宽度的程序计数器一样简单。存储器把被访问单元的内容装入 MDR。

b) CPU 把指令译码；即（假设操作数均在内部寄存器中）选择对应的寄存器向 ALU 送数，选择相应的要 ALU 执行的功能，如果需要可把结果进行移位，选定存放结果的寄存器。

c) ALU 执行所选择的功能。

d) 计算结果装入目的寄存器。

e) 检查结果，确定是否需要执行“转移”(BRANCH)。

f) PCU 计算出下条指令的地址。(一般叫做“取指”FETCH)。

如果操作数都不在内部寄存器或结果不装入它们中的一个内部寄存器，这个过程还更复杂。让我们选一个采用相对寻址的例子看看：

从本条指令的下一个字和 R1 的内容之和所指向的单元取出第一操作数；从本条指令后第二个字与 R2 内容之和所指定的单元取出第二操作数；把这两个操作数相加，并把所得结果放入到由本条指令后第三字与 R3 内容之和所指定的单元中去。然后，如果相加的结果有进位，则执行由本条指令后的第四条字与 R4 内容之和所指定单元中的指令，否则按顺序继续下去。

完成这条指令所需的步骤如下：

a) PCU 把下条指令的地址装入 MAR，向存储器发出“读出”访问信号。存储器将被寻址单元的内容装入 MDR。

b) CPU 对指令进行译码，也就是启动下列各个步骤。

c) PCU 自动加“1”并从存储器读出下一个字。

d) 把 R1 寄存器和 MDR 选作源寄存器，MAR 选作为目的寄存器。

e) ALU 执行 ADD 并把结果放入 MAR。

f) 第一个操作数从存储器中取出，并放入到譬如 R5 中。

g) PCU 自动“加1”并从存储器读出下一个字。

h) R2 寄存器和 MDR 再选作源寄存器，MAR 选作目的寄存器。

i) ALU 执行 ADD 并把结果放入 MAR。

j) 第二操作数从存储器中取出并放入譬如 R6 中。

k) PCU 自动“加1”并把下个字从存储器读出。

l) R3 和 MDR 选作源寄存器，MAR 选作目的寄存器。

m) ALU 执行“ADD”并把结果放入MAR，它是操作数之和将存入的单元。

n) 寄存器 R5 和 R6 选作源寄存器(它们保存着操作数)，MDR 现在是目的寄存器。

o) ALU 执行“ADD”，结果放入 MDR。

p) 产生一个存储器写入周期，MDR 的内容存入指定的地址中去。

q) 检查有无进位以便确定将要执行的下一步。假设没有进位。

r) PCU 进行再次加 1（以便跳到本条指令后的第五个字）。它现在指出了下一条指令的地址。

可以看出，为了完成一个采用相对寻址

方式的加法就用了18步。很明显，我们的CPU需要某种协调器，它能够

1） 对存储器取出的指令进行译码。
2） 启动需要执行的对应节拍周期。
3） 建立每步所需的各种控制。
4） 按照排好的顺序执行这些步骤。
5） 根据各种（条件）信号的状态作出判定。

我们把这个协调器叫做计算机控制部件CCU并示于图6中。现在，我们的CPU（或多或少）是比较完整了，下面我们将把它搞得更详细。

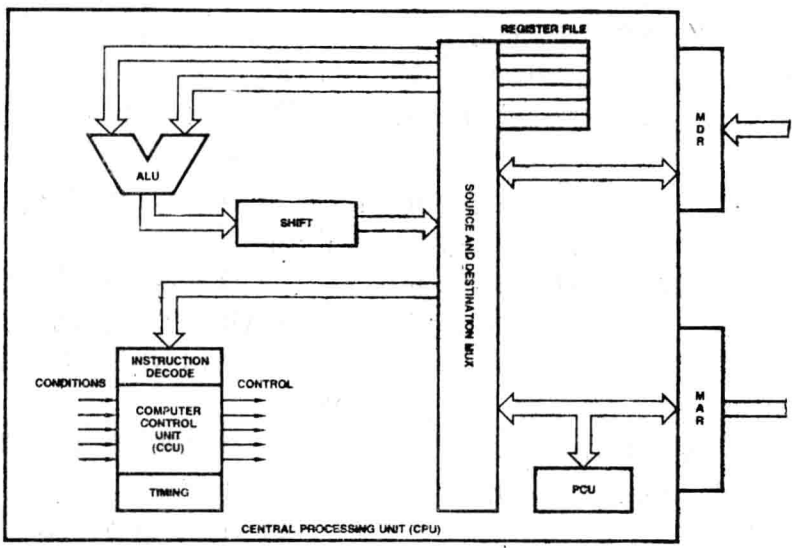


图6 在CPU内的计算机控制部件(CCU)

存 储 器

现在让我们来讨论一下存储器。存放在存储器中的数据是按字组织的，每个字含有N位。N对于小型处理机可以小到8，对于功能强的机器可以大到64。数字N叫作存储器的宽度，MDR的位数明显地也是N，等于存储器的宽度。

存储器的深度是它所含字的个数。对于有K位的MAR，可以对2^k个连续的存储单元进行寻址。地址区间是从零开始直到2^k-1。

由CPU进行的直接访问的存储器的读出时间是指从存储器的地址稳定后到数据准确地存入MDR这一段时间。这个取数时间可因存储器的不同类型小到几十毫微秒大到几个微秒。采用高速存储器来改善计算机的性能就是减少花费在存储器响应的等待时间。通常，存储器的速度越快其成本就越高，印制板面积较大和消耗更多的功率，从而产生更多的热量。一个32位宽，2K (2048) 字，50ns存取时间的存储器要求 +5V 电源供给10安培电流和需要10"×6"印制板面积。这已经是十分小的存储器空间。

一般说来，也不是非要有大容量快速存储器。不是所有的程序及其数据都必须同时常驻于存储器中。我们只要把当前的程序或是它的一部分放在内存中，而其余的程序或数据文件可以放在别处，并且当需要时，再

把相应的部分调到内存即可。这个"别处"可以是磁带，盒式磁带，磁盘，软磁盘等等，我们称它们为大容量外存。大容量外存的特点是：

1） 非常大的容量

2） 非易失性（不用时，能保存好使用时的信息）

3） 不能随机访问

4） 读取时间较长

5） 价廉（每位）

通常，外存设备是串行访问的，也就是第一个字的读取时间很长，但随后顺序的字的读取就相对快多了。

在稍后一章中将详细讨论主存与外存之间最有效的通讯过程，称为直接存贮器存取（DMA）。

外 部 设 备

在任何一个实际可用的机器中，需要某些与外部通讯的设备。它可能是键盘，CRT显示，卡片读入机，纸带穿孔机，在某些过程控制器中，还可能是读出传感器或定位调节器等。所有输入/输出设备的共同特点是它们的速度大大地慢于 CPU，因此而产生了一个时间配合问题，即 CPU 必须知道 I/O 设备何时准备数据传送。通常，为了引起 CPU 的注意，由外设向 CPU 发送一个信号。现在 CPU 要执行下列两个操作中的一个：

1） 周期性地检测这个信号，并且当它出现时就跳到处理数据传送的程序上去。这种操作方式叫做"定时询问"（Polling）。这种方法有两个主要缺点：第一个是把宝贵的机器时间花费在周期性的检测上，而且大多数检测是无效的（即没有出现"准备"信号）。第二个是 CPU 对该信号的识别要延迟到 CPU 按定时询问顺序询问到该设备时为止。

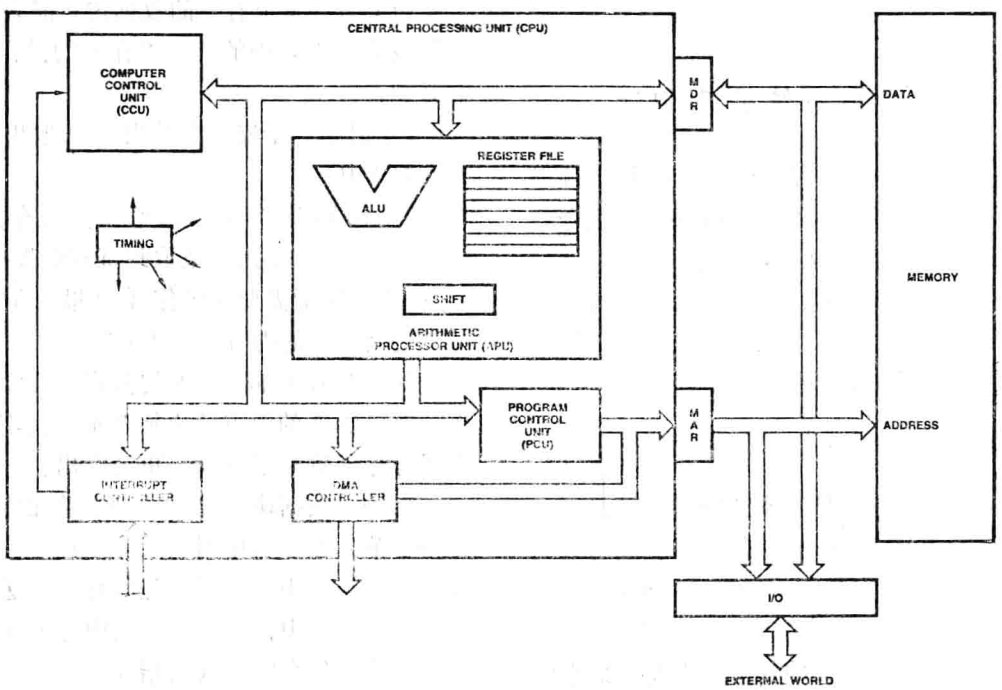


图 7 补充了 DMA 和中断控制的程序存储计算机

可以想象，当I/O设备数量很大时，将会出现什么情况。在很多I/O设备同时工作（忙态）时，等待时间会变得很长。

2）在CPU中增设某些硬件，使其能够感知“准备”信号的出现，并能中断正常的指令流和当有请求时就强迫计算机跳到I/O服务程序上去。它可根据被检测到的是谁的“准备”标志使CPU去处理对应I/O设备的不同的程序。当多于一个以上的外设同时要求使用CPU时，要对这些不同设备建立优先权。此外，在程序控制下，当计算机在某个时刻不允许中断时，这个电路可以不理睬某些或是全部请求信号。很明显，花费很少的硬件，我们在计算机性能上得到很大收益。我们把这部分硬件叫做“中断控制器”并且稍后将对它进行充分讨论。

现在，我们把计算机画在图7中。我们把ALU，内部寄存器组和移位电路包括在一个部件内，并称之为“运算处理部件”。

在后面的章节中，我们将对机器的每一部分进行更详尽的论述。

关于指令系统的几句话

CPU的内部结构在某种程度上依赖于计算机要执行的指令系统。如果这个指令系统很大，并且某些指令很复杂，则该机器的功能更强，速度更快和效率更高。另一方面，内部电路也就更复杂。下面就是对此进行折衷选择时要考虑的某些因素。

ALU处理能力

尽管只要有三种基本功能（add，complement，OR/AND）就可以完成所有的算术和逻辑操作，但大多数处理器还是做成能完成subtract，NAND，XOR等操作。这可能是如何在机器的复杂性上花费最小的代价而在性能和速度方面得到较大收益的一个最明显的例子。增加了这些功能后，XOR（异或）操作就可用一条指令完成，而不是原来的5条。

数据传送

假设下面有四种不同数据传送能力的计算机：

机器A。字从存储器读出只能装入寄存器A。寄存器A的内容可以写入或送入其他寄存器。任何一个寄存器的内容均可复制到寄存器A中去。

机器B。任何一个寄存器的内容可以复制到其余任何一个寄存器中去或者写入内存。从内存读出的字可以装入任何寄存器。

机器C。同机器B，但还有从内存一个单元中读取一个字，并把该字写入内存另一单元中去的能力。

机器D。同机器C，并且存储器到存储器的操作对连续地址可重复执行。要传送的字数（或地址上限和地址下限）由指令指定。

机器A具有很有限的数据传送能力。为了完成存于内存中的两个操作数的操作，我们必须：

1）把第一个操作数从内存中取出放到寄存器A中。

2）将A的内容复制到另一个寄存器中。

3）把第二个操作数取到寄存器A中。

4）完成所需要的操作（结果在A中）。

5）将A的内容存入内存。

如果对有中间结果的数进行连续的操作，则机器A的缺点就更加严重，特别是在内部寄存器数量很少时，更为突出。

将一个数据块从内存的一个单元传送到另一个单元在D机中用一条指令即可完成，而在A、B机中，则要求先将每个字送到一个内部寄存器中，然后再送到内存的新单元中去（传送每个字要两条指令）。

很明显，从A机到D机，计算机的译码，

多路转接和定序的复杂性都随之增大。我们应根据机器的性能速度和软件（程序设计）来调整硬件的复杂性。

寻址方式

参与一个操作的操作数可以用几种方法找到：

- 操作数就是指令的明显一部分（立即型）。
- 操作数的地址是指令的明显的一部分（直接型）。
- 操作数的地址在某个内部寄存器中，而该寄存器由指令指出（RR 型）。
- 操作数的地址是由指令指定的某内部寄存器的内容与本指令的一部分中所示之数（称为位移量）的和（RX型）。
- 把某内部寄存器的内容与由本条指令所指定的某地址内存放的数相加，所得之和就是操作数的地址（间接型）。
- 把某内部寄存器的内容与本指令显式指出的某单元数相加，所得之和指出操作数地址所在的单元。
- 等等

通过对上述操作方式组合或链接可以组成更多的寻址方式。每一种情况的“有效地址”必须通过计算和（或）存储器的访问才能得到。我们可以采用更复杂的寻址方式来增强机器性能，但我们也得增加机器的复杂性（特别是控制部分的复杂性）并为此付出代价。

定时、定序、控制

在上一节中我们指出了，我们可以通过在控制器（CCU）内增加硬件使之执行更复杂的指令来增强机器的功能。我们已指出“Add”操作一例就需要18个控制步骤。假如我们想将某些步同时完成，那我们就需要如图8所示的多相时钟来控制它们。我们首先将一条指令的第一个字（即指令的操作码）如图8那样装入指令寄存器。利用指令寄存器的输出（IR_0 到 IR_{n-1}），不同的时钟相位和各种条件输入到CCU，我们可以写出满足我们指令系统的全体指令的所有各步(steps)要求的逻辑等式。然后，我们可用卡诺图或其他方法简化这些等式并利用AND，OR，INVERT（反相器）等门和触发器来实现它们。这看来很简单，但可以设想一台高级的计算机及其调整过程是相当复杂的。

直接提出的问题是：是否有结构性更强和更易于理解的方法来实现复杂的控制呢？或者可以说我们能否有某种“微机器”，它能完成我们这个计算机的定时、定序和控制方面的所有工作——一个位于我们计算机内

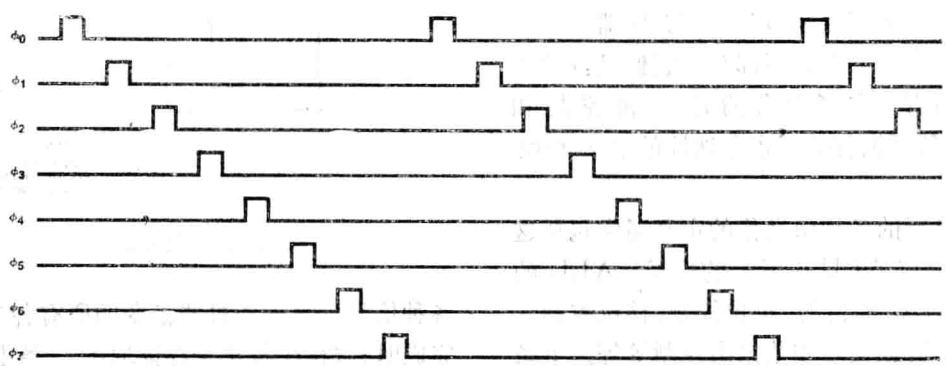


图 8　8相时钟脉冲

的计算机呢？由于Am2900系列——新的双极型LSI器件的出现，回答是：是的，我们能够这样做！

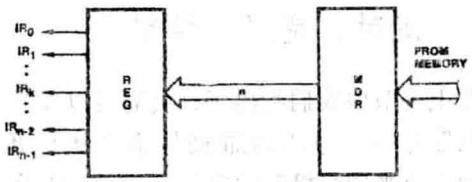


图 9　指令寄存器的各位

微　机　器

我们需要一个能够执行若干规定序列的机器。这使我们想起，这不正是程序存贮计算机的含义吗？我们的微机器与通用计算机的唯一区别是，在通用机中所执行的程序是从一个作业到另一个作业改变着的，而在我们的微机器中，程序则是固定的。这就允许可用PROM来代替在通用机中必需的RAM存储器。采用微机器的CCU控制部件如图10所示。

微程序控制的机器基本上是一种利用一套连贯的微指令序列来完成机器所需要的各种命令的机器。如果这机器是计算机，则每个微指令的序列可以用来完成一条机器指令。在执行机器指令时，由机器完成的所有这些微小的基本的作业叫做指令。而这些微指令的存储区通常叫做微程序存储器。

微指令一般分为两部分。它们是：(1)要执行的所有基本微操作的定义和控制和(2)下一条要执行的微指令地址的定义和控制。

要执行的各种微操作的定义通常包括这些内容：如ALU源操作的选择，ALU的功能，ALU的目的，进位控制，移位控制，中断控制，数据入和数据出控制等等。下条微指令功能的定义通常包括：下条微指令地址的源选择的区分以及在某些场合对微指令地址提供实际值。

微程序控制的机器与非微程序机器通常有下列不同：较早的非微程序的机器实现控制是借助于把门和触发器用某种随机的方式组合以产生机器所需的定时和控制信号。另一方面，微程序控制的机器，通常看做是十分规则化的并在控制功能方面更加结构化。按最简单的定义，微程序控制部件由微程序存储器和确定下条微指令地址所必须的结构组成。

操作码（计算机要执行的指令类型）装入指令寄存器并由指令译码器将其译码。实际上，它产生微地址，这个微地址是那条指令留驻在微程序存贮器中执行序列的第一步。随后Am2910序列器产生下条微指令的微地址。微程序的数据提供为控制计算机所有各部分（其中包括序列器本身）所需要的

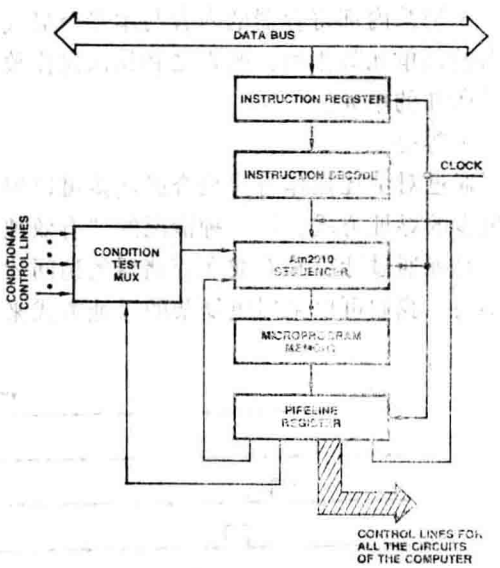


图 10　微机器

各种信号。当一条机器指令的所有各步均已完成时，微程序将从主存读取下一条机器指令。典型的是，计算机控制部件CCU取出

指令并把它译码，即用PROM将操作码（op code）取出指令变换成对应执行该条指令所需的微指令序列的头地址。CCU也可读取机器指令所需的所有操作数，并把它们交给ALU进行处理。典型的计算机CCU的工作流程之一表示于图11。

假设机器指令的操作码有8位宽。这允许我们至少可执行256条不同的指令。又假设完成这些指令需要的步数平均为6个。如果我们采用独立的微程序存储器单元，则这个微程序存贮器的深度仅为1.5k（k=1024）。但在这种场合，序列器完全可用简单的计数器代替，通常我们总设法在不同的指令中间共享某些微程序。稍加努力我们就能够缩小图10的微程序存储器的深度，使其小于$\frac{1}{2}$k。当然，序列器将要稍微复杂些，它将能完成条件转移和微子程序调用，但对于微程序控制我们尚不需要如在机器指令系统那一节中所谈到的那样复杂的寻址方式。

另一方面，我们微程序存储器的宽度可以较大，可能是60到100位。这依赖于我们计算机所需的控制线的数目。这不是很大的缺点，因为PROM的成本正在继续下降。在以后各章中我们将讨论减小微程序存储器深度和宽度的方法，以便降低成本。

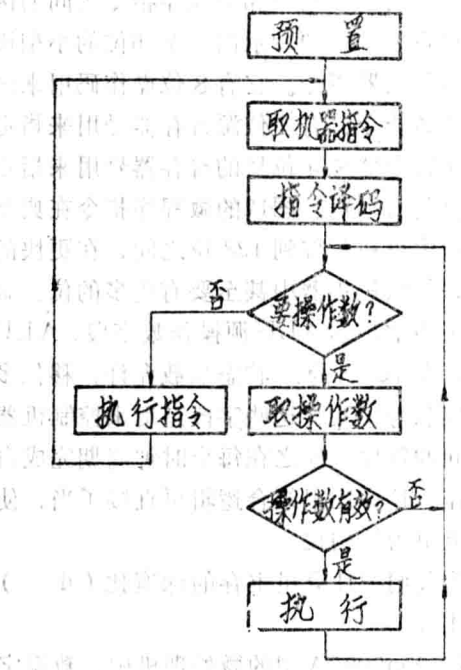


图 11 计算机的控制功能流程图

机 器 指 令

操 作 码								目的寄存器 R_1				源寄存器 R_2			
15							8	7			4	3			0

微 程 序 指 令

转移地址	Am2910指令	条件码多路器	IR加载	Am2903 A.B地址	Am2903的源	Am2903的ALU	Am2903的目的	状态加载	移位多路器	其它

32到128位

机器指令有16位长，它由操作码、源寄存器和目的寄存器编码组成。

微程序指令确定了控制机器各部分需要的所有的基本信号。

图 12

弄清机器级指令和微程序指令之间的区别是很重要的。图12示出一个16位的小型机的典型的机器指令。它有8位操作码用来区分256条指令，有4位源寄存器号用来指定16个源寄存器和4位目的寄存器号用来指定16个目的寄存器。图12的微程序指令在典型的设计中可在32位到128位之间，在更快的高度并行微码机器中甚至要有更多的位。微指令字通常包含ALU源操作数字段，ALU功能，ALU目的，状态加载允许，移位多路器控制等字段。这些字段是用来控制机器中不同的器件，使之在每个时钟周期完成自己动作。这样比用组合逻辑更直接了当，使设计也更为结构化。

让我们把计算机主存的深宽比（d/w）作一比较。

在Am9080A型的微处理机中，数据字段宽是8位和地址字段是16位，可寻址64k单元。其d/w比是8k。在某些小型机中，数据字段宽是16～32位，地址容量是64～128k。其d/w比大约与上面的相同。在更大的计算机中，有32～64位的数据的宽度，我们看到256～512k深的存储器或者更深些，其d/w比仍是至少8k。

另一方面，微程序存储器的d/w比很少是大于几十的。即使我们假设深度为2k，宽度为64位，我们得到的d/w比也仅是32，通常它在10左右。控制一个d/w为10到20的机器比控制d/w为8k的机器要容易多了。

再说几句

我们已经建议用“微机器”代替CCU的随机逻辑的实现方法。我们称它为“微程序控制的结构”。这种结构的最大优点是建立控制序列结构的简易性。我们可以指定微程序存储器的一位或一组位来控制某些功能的实现（如ALU源寄存器的选择，ALU的功能，ALU目的寄存器的选择，条件选择，下一地址计算选择，MDR目的寄存器选定，MAR源寄存器选择等等）并对于每一个微步，我们可以把这些位相应的状态（LOW－HIGH）写入存储器。稍后我们将会看到用来完成写入微程序的那个自动化的而又复杂的工具。这种工具之一是在系统29中使用的AMDASM（AMD微汇编缩写）。但这不是微程序结构的唯一优点。

由于不能做到万无一失，某些被忽视的错误可能滑入到设计中去。在随机逻辑结构中，我们必须重新设计，这通常导致重造整个计算机。而在微程序中与此有关的位就能够解决问题了。这是十分容易的，如果在研制和调整阶段将PROM用RAM代替的话，当然，我们必须借助某些外部手段能够向这个存储器加载（装入）微程序。AMD的System/29TM是十分有效的工具。

最后，让我们面对现实：通常在你完成了80%的逻辑设计时，经销人员往往要改变它们的要求（即指令系统）。现在，你必须从头开始。不必如此！你只要改变一下某些微码或许还有很少硬件即可。当只需要向已有的指令系统填加一些指令时，就更为方便了。按照新的思想向你的微程序再填加几行就行了，这只要占用System29几分钟时间就完成了。这就是最大的灵活性！顺便指出，千万别告诉经销人员这样容易修改设计，否则你的产品将永远推销不出去。

小　结

框图13示出了一个典型的16位小型机结构。在这张图上标出了可以用在这些部件中的Am2900系列的各种元件。同样的设计既可用4个Am2901A也可用4个Am2903来组成ALU数据通路。一个Am2910作为微程序序列器可用来控制容量可达4k字的微程序存