TP译文

Microcomputer

Z-80的彻底研究

北京工业大学自動化系研究室
北京微型电脑应用分会

1

Z80 的彻底研究

日（八）木广满等著

张学志　译
侯伯文　审校

参加本文审校的还有：冯义波　管保尔同志

目　录

第一章　微型计算机与Z80

单片CPU的结构

在论述Z80之前，先简单介绍一下单片CPU的内部结构。

图1是一般的单片CPU。它包括许多暂存寄存器，在寄存器之间进行运算的运算逻辑单元ALU(Arithmetic Logical Unit)，给出CPU外部存储单元地址的寄存器(MAR)，控制它们动作的控制时钟发生器以及发出控制命令的指令寄存器。

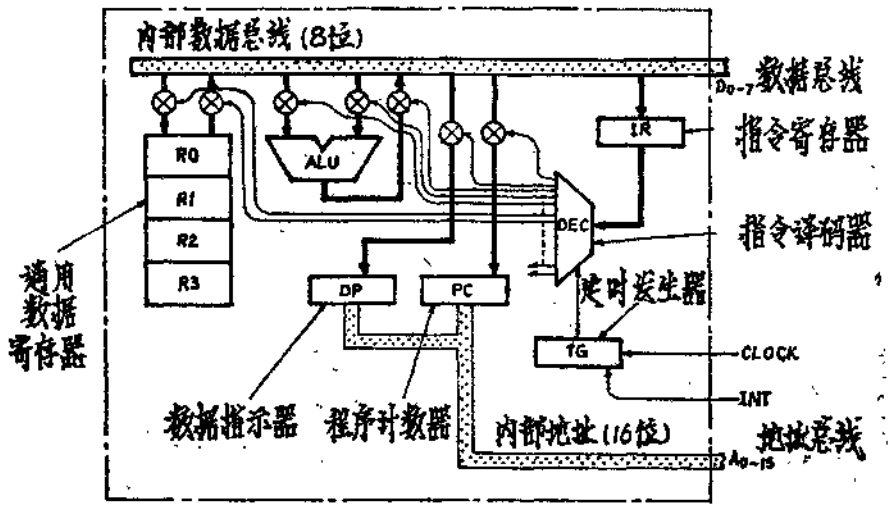

图1　一般单片CUP的内部构成

有三条总线对着外部集成电路：发出存储单元地址信号的地址总线(一束地址信号线)，同存储单元数据有关的数据总线，以及传送控制信号的控制总线。

除此之外，还有时钟、复位、中断用的由CPU控制的信号线。为了对各信号线有一个大致了解，现对其操作顺序作一简单介绍。

要使CPU进行一连串的操作（指令），就要将CPU与内存连接起来，如图2所示。存储器把由地址总线信号指示的地址（场所）的内容（也就是指令）送到数据总线上。处于此种状态的CPU一复位，首先是称作程序计数器的寄存器置0。接着就开始了以下一连串的操作。

①把程序计数器(PC)的内容送入地址总线。

②从存储器通过数据总线，把①中程序计数器所指示地址处的内容（指令）读进指令寄存器(IR)。

③按照指令寄存器的内容进行控制，执行寄存器间的运算、传送等指令。

④程序计数器的内容加1。

在①～④的操作中，①、②、④只是从存储单元中取出指令，也可以说没有完成什么具体工作，这些操作称作“取指令(fetch)”。而③才是有具体目的的操作，这称作“执行指令(Execute)”。

这样看来，一个指令的执行，一般由取指令阶段和执行指令阶段组成。而且①～④的操作一结束，CPU就又回到①上去，就这样一直循环反复地进行着。

①～④的一个回圈叫作指令周期。在Z80中最低要求4个时钟的时间。如果使用4MHz的Z80A就是1

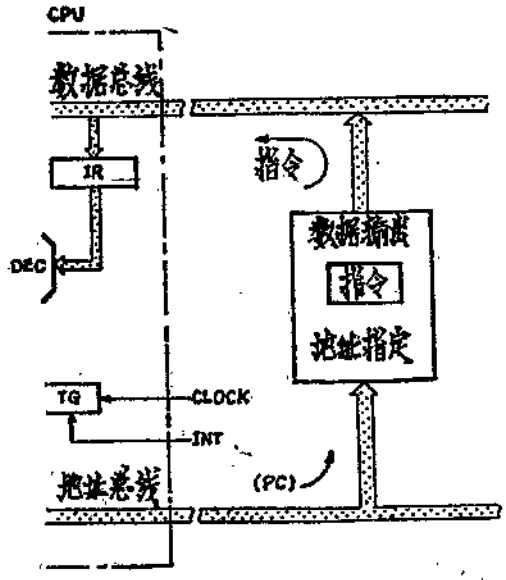

图2　CPU连接存储器

这样，存储器与CPU就能够顺利地进行工作了。但是，如果从CPU—存储器的外围向CPU提出输入输出数据的要求来看，以上CPU只是反复执行着来自存储器的指令，而完全不知道外围方面对它有没有什么要求。

为此，就从外部向“CPU的肩上”加上一根线，这叫中断线。巧妙地使用这根中断线，就能极大地提高CPU的工作能力。

下面来看看Z80是怎样提高它的工作能力的。

8085A与Z80比较

Zilog公司本身是从Intel公司分出来的，因此说Z80是从8080A改进而来的也就可以理解了。

Intel公司为了同Z80竞争，曾设计出8085型的CPU。但在芯片设计方面仍不如Z80的优越。

8085在规格、逻辑线路上还存在着一些缺欠，正因为这一原因，直到目前为止，广大用户还都喜欢使用Z80。

使用Z80—CPU的计算机

Z80概说

图3是Z80的内部结构框图。由14个8位通用寄存器（代名为ABCDEHL），8位加法、减法、移位、逻辑运算的ALU和16位的程序计数器等组成。这同前述的“一般单片CPU”

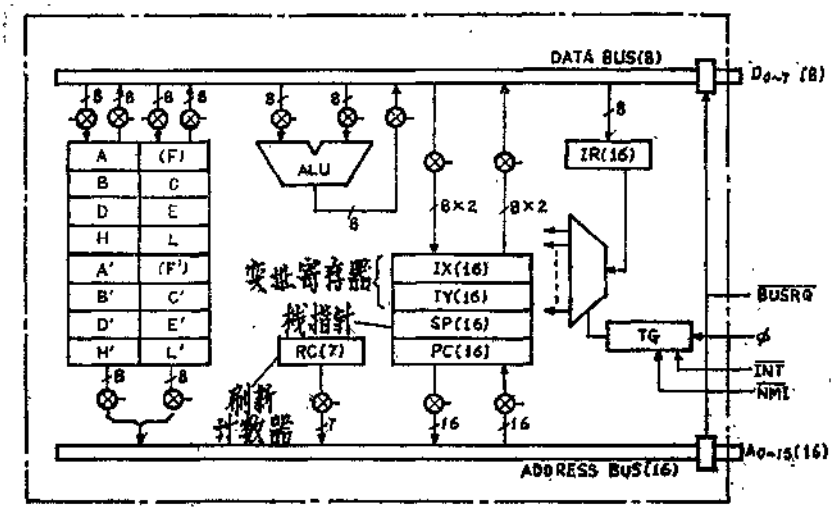

图3　Z80的内部结构

程序计数器(PC)指示的存储器的地址内存着CPU所要执行的指令。而变址寄存器(IX、IY)和堆栈指示器(SP)则是用来指示其他别的存储单元的地址。这是这些寄存器的不同之处，但就指示存储单元地址这一点看来，后者同PC又是相同的。

例如，一条指令把IX指示的存储单元的数据传送给A寄存器，这就要使用PC以外的寄存器了。由此可以看出IX，IY，SP寄存器就相当于图1中所介绍的一般CPU内的数据指示器(DP)。

但是，一般都愿意把相当于数据指示器的寄存器的操作，设计得越简便越好。例如，SP寄存器访问由它指定的地址单元时，一写进数据，SP本身的值就减1，读出就加1。利用这一点虽然意识不到是在存储单元的哪一地址上写入了数据，但知道向存储器内存入和取出数据却是可能的。

除这些寄存器以外，Z80还有BC，DE，HL等寄存器对（8＋8＝16位），这些寄存器对也能作为指示数据地址的指示器使用。由此可见，重要的是CPU应当能够使用最简易的方法来指示存储数据的地址。

此外，Z80还专有一个刷新计数器。它对CPU虽无什么意义。但对使用动态RAM存储器来说则提供了一种非常简便的方法。这种方法能够把单芯片CPU所特有的功能充分表现

出来。

所谓动态RAM，就是在一个小小的电容(0.01pF)上积蓄电荷，借此可以把数据“0”，“1”记忆下来。但是由于电容漏电，也必然会失去所积蓄的电荷。

例如，写进“1”或“0”。元件说明书指明，在最不利条件下，由漏电直到失去正确数据，时间保证为2ms。也就是说，一个存储器元件必须在2ms之内读出原有的内容，并重新将它再写入一次，这叫“刷新”。

此外把室温，适当的电源、电压考虑进去，动态RAM的数据记忆时间，一般能接近1秒。

在小型计算机中，“刷新”的结构比较复杂。必须考虑CPU不向RAM存取（读，写）的时间，否则就会破坏存贮内容。

16K动态RAM一次刷新128位数据。它输出了其中的1位，就会由于这一次的读出动作而对128位自动进行刷新。

就是说用户只能在16K动态RAM低侧的7位地址信号线上，在2ms之内进行刷新(16K÷128=128=2^7)。

在Z80-CPU不向存储单元存取的定时时间间隔内，其内存刷新计数器的值输出到地址总线。同时，CPU向外部送出刷新时钟，然后刷新计数器的值加1。

由于Z80中有了刷新计数器，CPU就能够顺利地进行刷新动作，这也是Z80倍受用户欢迎的一个原因。

另外一个原因是Z80还有2根中断线，1根是可以用指令进行屏蔽（使无效）的可屏蔽中断，另1根是优先权很高的非屏蔽中断。

模型计算机

以上是Z80的一般知识，现在来介绍一下简单的模型计算机。

图4是它的全部电路。虽然只有存储器和CPU，但按下复位开关(SW_3)，就可以从RAM成ROM区域〔可以用开关(SW_0)来更换〕的地址执行。

但是仔细一分析，还是不能向此计算机输入输出数据。因为只有向存储器中写入了“指令（叫程序也许更合适）”之后，才能够输入输出数据。

这种状态就好比一个“人”只有“头脑”而无口、耳、四肢一样。因此首先要想办法从外部向RAM写入“指令”，然后根据此指令来驱动输入输出。这是一个计算机通常所必不可少的条件。

图5是该计算机的I/O（输入输出）。I/O包括输入用的8个开关和输出用的8个LED。当然也可以使用这些开关把程序送入RAM区域。同时本机也还具有把写入RAM里的程序拷贝写入到PROM(2716型)的功能。

那么本机是怎样把程序写进RAM里去的呢？

Z80备有$\overline{BUSRQ}$，$\overline{BUSAK}$引脚。当$\overline{BUSRQ}$引脚处于低电平(0.8V以下)，在此刻进行着的机器周期一结束，就立即把低电平输出到$\overline{BUSAK}$引脚。与此同时，地址总线，数据总线和$\overline{MREQ}$，$\overline{IORQ}$，$\overline{RD}$，$\overline{WR}$，$\overline{RFSH}$等各引脚，都处于高阻抗状态（也就是脱离电源的电平浮动状态。）

当然，此时CPU也就中断了执行指令。等到$\overline{BUSRQ}$返回到高电平(2.0V以上)，CPU

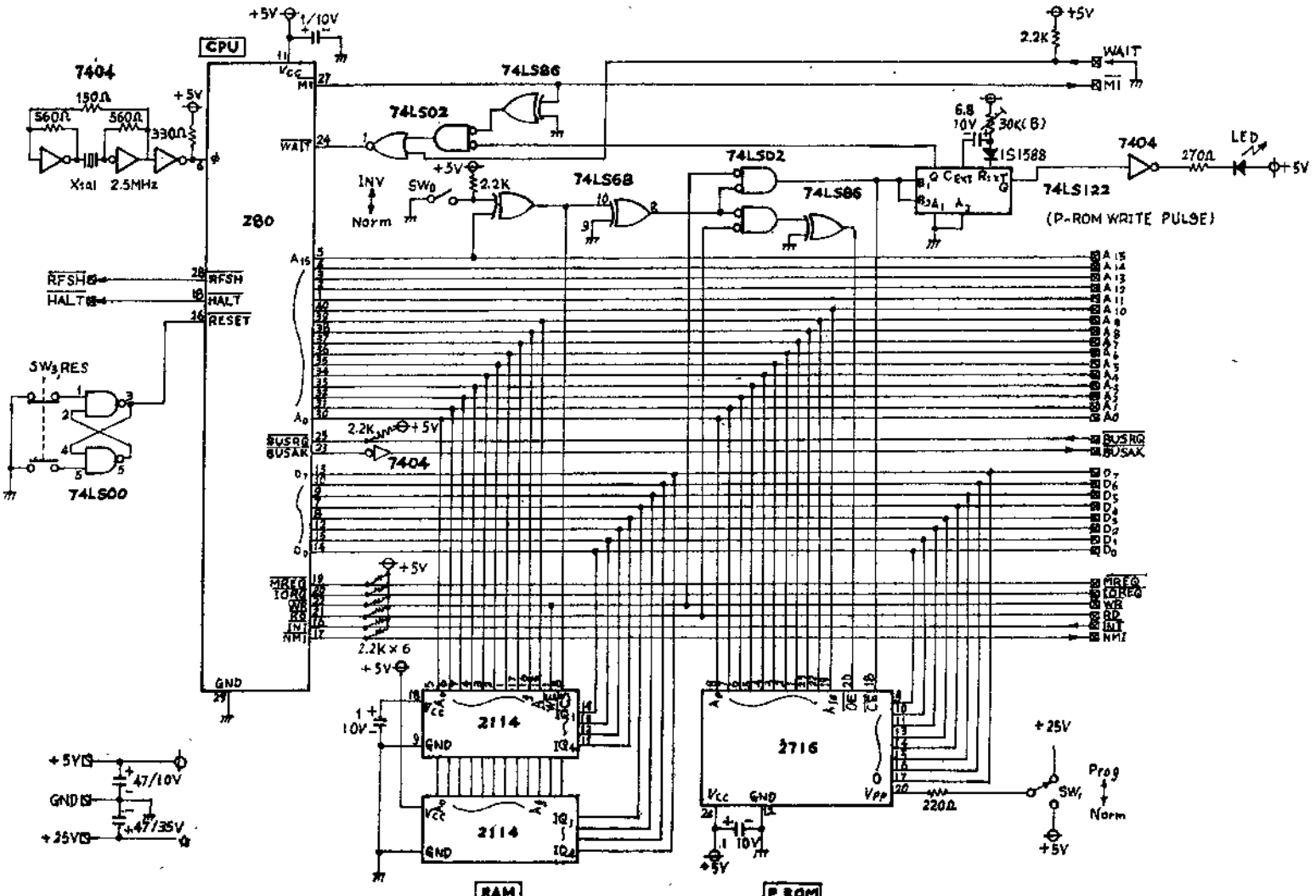

图 4　使用Z80的简单模型计算机

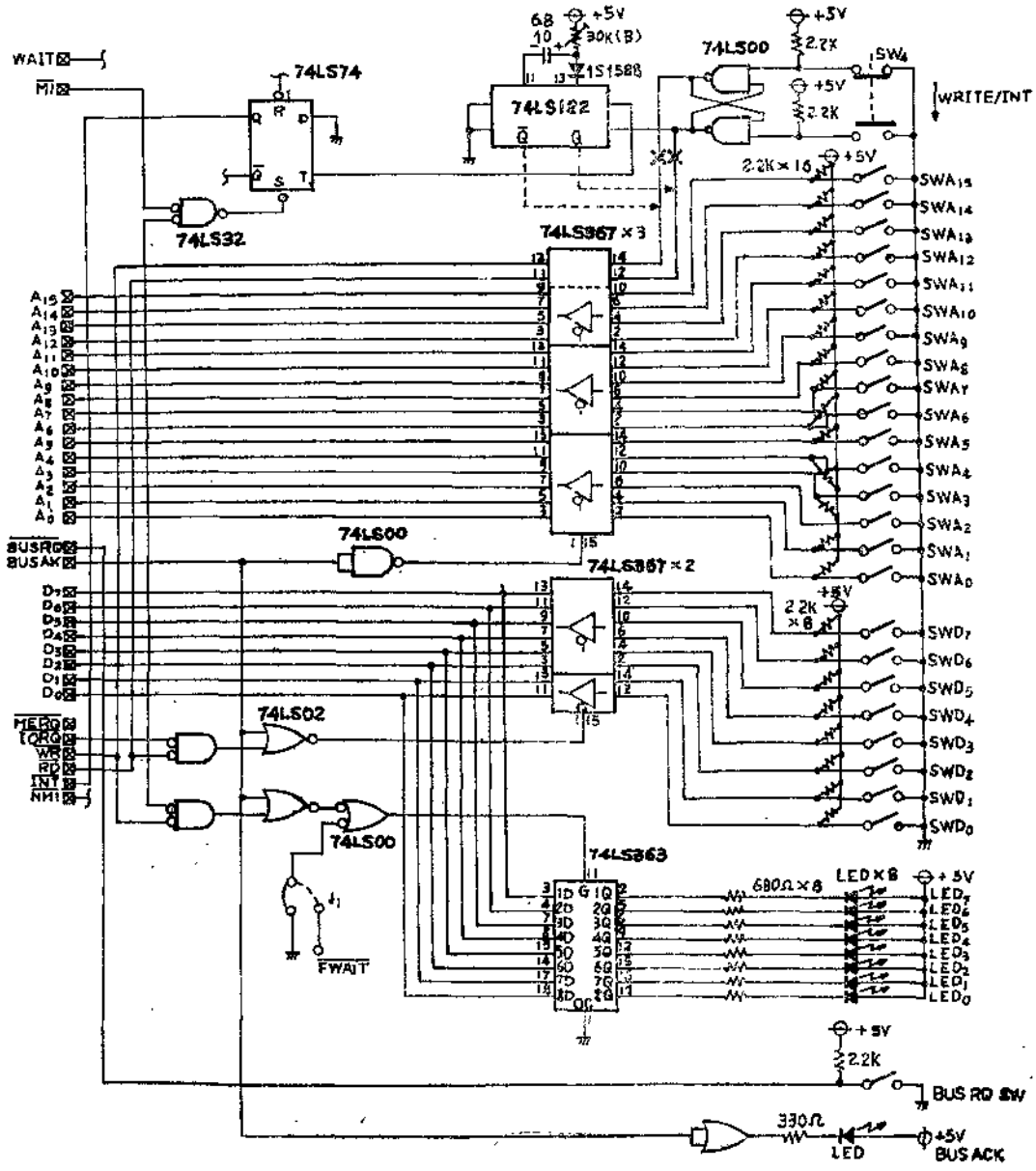

图5　模型计算机的I/O

又开始继续执行指令。换句话说，由于$\overline{\text{BUSRQ}}$变为低电平，CPU就从地址总线以及控制总线上离开电源。利用这一点，就能够向在总线上“悬挂着的”存储器(RAM,ROM)的某一地址写进或读出数据。

如果不考虑通过硬件改变存储器的内容，单靠一条合适的程序是能够准确地进行同样的操作，而且一般情况下也是这样进行的。但是，这里为了理解$\overline{\text{BUSRQ}}$的意义，本例中特意通过硬件的作用及其结构来加以论述。

图6，$\overline{\text{BUSAK}}$成了低电平则Z80与总线分离。根据此信号，开关的各信号送入总线。

因此，按下WR开关(SW_4)，把由SWD_{0-7}所示的数据，写入由SWA_{0-15}所示的存储单元地址中去。

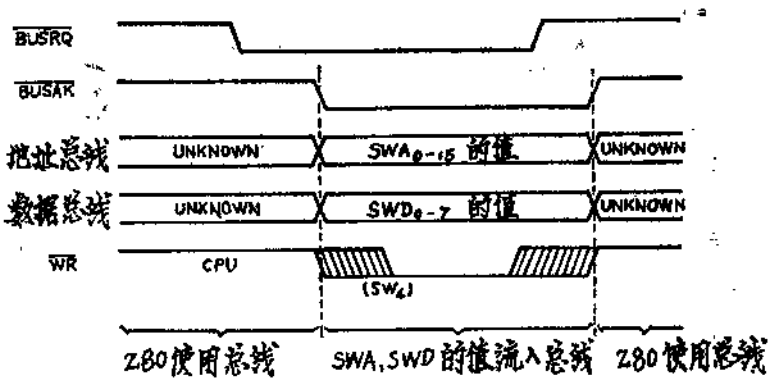

图6　根据$\overline{BUSAK}$开关信号送入总线

可是，代替这个开关，象图7那样连上第2个CPU可以不可以呢？

如果第2个CPU想实现自身对存储器的存取，就对Z80发出$\overline{BUSRQ}$信号，当$\overline{BUSAK}$信号从Z80返回后，第2个CPU就开始了对存储器的存取。

例如，也有这样的情况，就是第2个CPU仅具有把RAM的内容传送给别的输入输出装置(图7)的功能。这第2个CPU叫作DMA(Direct Memory Access)控制器，其结构简单，速度快，并且是单功能，所以比一般的CPU另有派用。当然，在存储器与存储器之间传送数据时也可以使用DMA控制器。

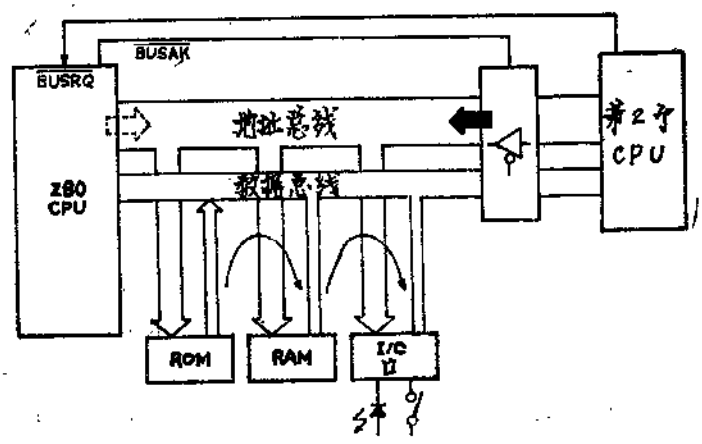

图7　连接第2个CPU(DMA控制器)

这样看来，第2个CPU同存储器那边的CPU同具Z80的优点。实际上在8位CPU发展的初始阶段也曾设想过使用双CPU的方法。但是使用这种方法，必须要有一种手段能控制各CPU之间的总线访问。但目前尚无这样一种手段能够解决这个难题。另一种说法认为单

芯片CPU不宜使用这种方法。

本例就是利用$\overline{BUSRQ}$控制向RAM写入程序（机器码指令）的。只要按下复位开关SW，CPU就从存储器的0地址开始执行。可是RAM的地址分配是从16进制的8000地址（以后写成8000H）开始，因此不论是送什么程序，在复位时，程序计数器的值都必须置于8000H。

因为硬件不能改变程序计数器的值，此模型机复位后，最初所取的指令，只能是在CPU芯片外部，从8000H地址强行读取。在8000H上写进一条把程序计数器值置于8003H的指令*。这样一来，CPU一复位，就如同是从8000H地址执行。做到此点，只需转换A_{15}，使0000～7FFFH地址换成8000H～FFFFH地址即可。

在$\overline{BUSRQ}$处于高电平时，SWD_{0-7}作为CPU的一个输入口工作，而LED_{0-7}作为一个输出口工作。因此首先开始执行这样一个程序，即SWD_{0-7}的值输入后，加上10_{16}，而后把此数输出到LED_{0-7}，下面看看执行的情况。

程序①是用汇编语言书写的程序。执行此程序的全过程是，CPU在数微秒中读1次SWD_{0-7}的内容，把其值加10_{16}，而后输出到LED_{0-7}。

但是以SWD_{0-7}的变化为例，假如1秒钟变化1次，就等于CPU把它能力的99.999%都白白浪费掉了。因为程序①的程序在SWD_{0-7}变化后只执行1次。

因此，CPU通常用接受执行别的（写在别的存储器上的）程序的任务，而当SWD_{0-7}变化了时，再让CPU知道“SW变化内容”。这样，CPU就能得到更有效地利用。

〈程序①〉程序举例，

操作：在SWD_{0-7}上置位的数值上加“10”，其结果输出到LED_{0-7}上

```
ORG     0000
IN      A,(00)      DB    00
ADD     A,10        C6    0A
OUT     (00),A      D3    00
JR      *--3        18    F8
END
```

中断和优先链

中断

为了有效地利用CPU，可以使用NMI，$\overline{INT}$二个中断引脚。虽然这二个都是中断引脚，但对中断的响应略有不同。这一点在后文还会遇到，这里先谈谈$\overline{INT}$引脚。

若$\overline{INT}$引脚处于低电平，则Z80在执行指令周期的结尾时刻即得知从外部发来了中断申请。这时Z80就不再取下一条指令，而是进行以下三种操作中的一种。即，

中断方式0（IM0）：下一条指令是从数据总线读入的，同执行一般指令一样的指令代码。程序计数器的值不变。

中断方式1（IM1）：首先从外部读进指令而暂不用它，但自动执行的是RST38H指令（子程序调用0038H）。

*译者注：即在8000H处，置一条三字节的“JP8003H”指令。

中断方式2（IM2）：由8位I寄存器（其值可由程序更换）同从数据总线读进Z80内的8位数据合成一个16位地址，自动地间接调用此16位地址处的子程序（图8）。

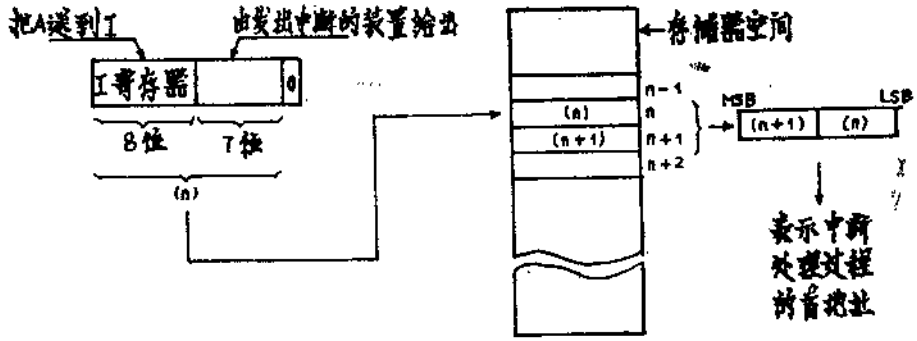

图8　方式2产生的中断矢量

以上三种操作（IM0～2）究竟选用哪一种，要根据指令而定。IM1方式最为简单。首先看看它的使用情况。

CPU执行着与SW，LED完全无关的其他程序（例如某个求π值的程序）。这里把SWD置于某值，按下中断开关SW_4，CPU就中断了正在执行着的程序，调用子程序0038H。BUSAK本例用作写入程序的信号，从0038H开始写入的是以下的程序*。

①此程序首先将A寄存器和程序状态字PSW（F寄存器）的内容（图9）转送给RAM的某地址（存入）。

②SWD的内容读进A寄存器后加10。

③A寄存器的内容输送到LED。

④把原来保存的A寄存器，PSW的内容送回到A寄存器，PSW中去（还原）。

⑤中断许可触发器（IFF1）置位（EI指令），在RET指令中再回到原来执行的程序。

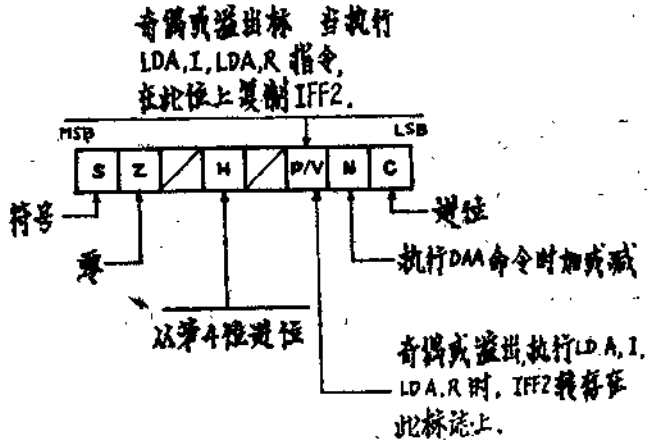

图9　PSW（F寄存器）的内容

*译者注：利用图5中BUSRQ SW每低一次，就将SWD给出的程序机器码写入SWA给出的地址中去，用这种方法一个一个字节写入。

实际上只有②③是有目的的操作，而①④是在执行程序过程中由于发生了中断，为避免误动作所采取的步骤。⑤是为了再次接受中断。此程序如程序②所示。

<程序②>中断处理程序举例

```
ORG       38H
PUSH      AF          F5
IN        A,(00)      DB    00
ADD       A,10        C6    0A
OUT       (00),A      D3    00
POP       AF          F1
EI                    FB
RETI                  ED    4D
```

Z80还有二个中断允许触发器IFF1和IFF2（1位寄存器）。

中断允许触发器置位为"1"；外部有中断申请时则传至内部；若复位为"0"，外部中断信号不能传至内部（图10）。

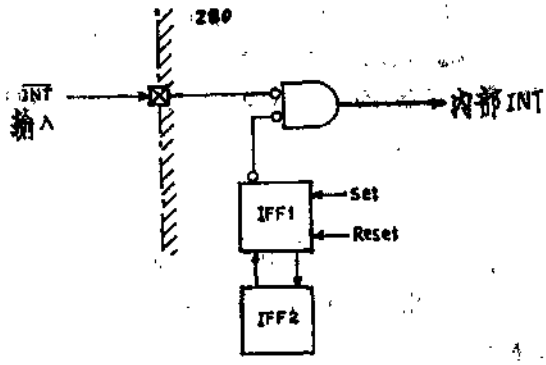

图10　IFF的动作

如不接受中断的情况，当然是IFF处于复位（DI指令）。在接受中断时，IFF1自动复位，直到IFF被置位为止（EI指令），就不再接受其它的中断申请了。另外，如果想真正能接受中断，是从执行EI指令的下一条指令的指令周期才开始的。

EI

RET

也就是在中断处理程序的终了才是有效的（图11）。

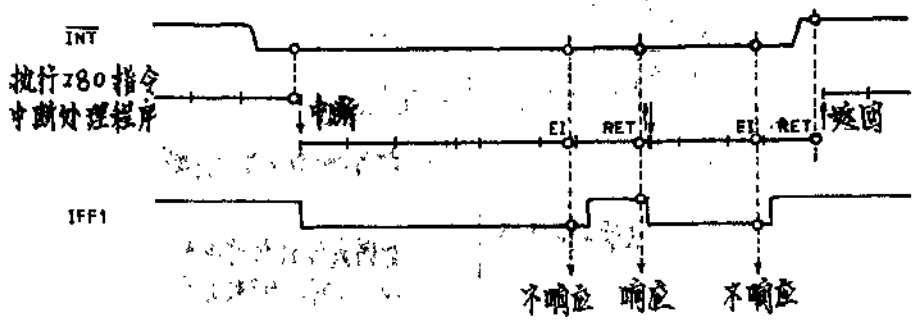

图11　EI执行中不接受中断

开关与指示灯泡的例子也是在按下SW_4时发出中断申请（使$\overline{INT}$为低电平）。也有在1次中断服务结束又紧接着接受另一次中断的情况。这样的情况是在执行EI之后接着中断的。为了把中断返回地址保存起来，就要将它们顺序存入存储器单元中。这一点是很重要

的。

上面介绍了IFF 1 的情况，现在看看IFF 2 有哪些用途。为了说明IFF 2，有必要对另一根中断线$\overline{NMI}$加以说明。

假如在执行前例的中断处理程序时，发生了紧急中断事件（如电源切断），这时对中断允许触发器的位置或复位等状态一概不予考虑，压倒一切的就是立即接受中断。在Z80上就专为此用途增设了一个优先权最高的$\overline{NMI}$（Non Maskable Interrupt非屏蔽中断）引脚。

$\overline{NMI}$处于低电平（正确地说，是向低电平转换，发生中断），在CPU内将IFF 1 的内容向IFF 2 转送，并且不管IFF 1 的值是"1"或是"0"均同$\overline{INT}$一样发生中断。此中断处理后，一执行RETN指令，IFF 2 的内容被送回到IFF 1，这是返回在$\overline{NMI}$成为低电平以前的IFF 1 状态的一种办法，因此才能进行多重中断。对于$\overline{INT}$来说，实际上控制中断允许的是IEF 1，但Z80一经接受了$\overline{NMI}$，就立刻强制IFF 1 复位。因此，从$\overline{NMI}$到返回为止，要记住IFF 1 原来的状态。这样，把IFF 2 作为IFF 1 的保存地点也是合适的。

$\overline{NMI}$一经接受，Z80就在前述IFF 1 ⇒IFF 2 传送的同时，自动调用0066H地址的子程序，这时对任何方式的$\overline{INT}$都不予考虑。

在这里，对Z80系列的中断方式要稍加注意。

如果，Z80—CPU连接 3 个Z80—PIO（Z80系列的一般输入/输出用大规模集成电路（LSI），这 3 个PIO都是通过中断进行输入或输出的装置。

图12表示此装置中中断的接线。从PIO(PIO0,1,2)各发出一条中断申请线与CPU的$\overline{INT}$连接。各PIO$\overline{INT}$的输出因为是开漏极输出，所以当一个以上的PIO输出为低电平时，CPU的$\overline{INT}$引脚就成了低电平，Z80随即接受中断。

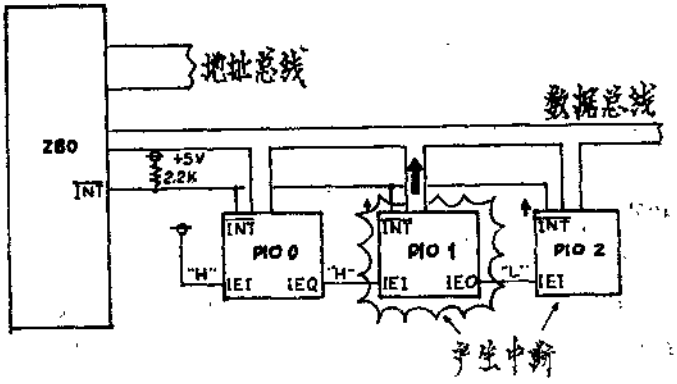

图12　根据优先链Z80系列的中断响应

例如，假定从PIO 1 和PIO 2 发出中断申请，Z80的$\overline{INT}$成了低电平。CPU接受以后，就要对中断进行处理（服务），可是这时的CPU并不知道中断申请是从哪个PIO发出来的。如果说CPU知道，也只是知道$\overline{INT}$处于低电平……。因此Z80必须对PIO进行调查。

首先CPU根据一组控制信号(M1，IORQ)对所有PIO发出"接受中断"的联络。这样，发出中断申请的PIO就会把它的中断处理程序的首地址（矢量地址）送给CPU。因此，CPU就能够知道是从哪一个PIO发来的中断申请。

Z80—CPU对中断的这种响应方式使用的是硬件的，即高效率的IM 2 方式。

可是图12所示的情况，是从PIO1,2二个PIO同时发出中断申请，也就是二个PIO同时向Z80送出矢量地址，这样一来，在数据总线上不是就会发生矢量地址信号冲突的现象吗。

优先链

为了避免在数据总线上发生地址信号的冲突，Z80设计了一种非常巧妙的方法，叫作优先链(Daisy Chain)。使用这种方法可以决定PIO1,2，申请中断的优先权，按其优先权的高低（先后）顺序向Z80输送地址信号，这样一来也就不会发生冲突了。

图13是PIO0-2的优先链部分，在发生中断信号的PIO(PIO1,2)上，IEI是高电平,IEO输出却成了低电平。而在不发出中断申请的PIO上，其IEI的值原样送到IEO。这样，从最左位置的PIO顺序向右依次看下去，对中断优先权的高低也就一目了然了。

只有当PIO的IEI输入高电平，IEO输出低电平时，才能把中断矢量地址送到数据总线上。于是Z80接受了该PIO的中断服务。服务程序一结束，根据RETI指令，PIO的IEO又回到高电平。本例中，首先接受的是PIO1的中断，处理终了，再接受PIO2的中断。

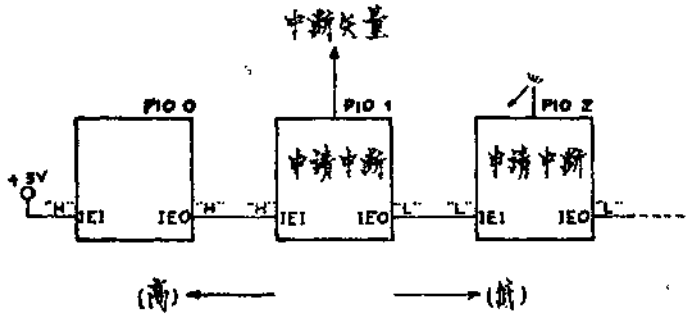

图13 Z80系列的优先链

P—ROM写入器和WAIT信号

上面对中断作了介绍，现在对P—ROM（可编程只读存储器）写入器部分略加说明。

这里使用的P—ROM是叫作2716的FAMOS EP-ROM，写入数据需要+25V的电源。因为进行写入动作的信号全部都是TTL电平，所以也就不需要专门的“P—ROM写入器”的逻辑电路了。

简言之就是Vpp引脚加上+25V，把数据从CPU送进2716RAM（但是$\overline{WR}$信号的宽度必须是每个地址单元50ms）。

图14是2716同2116（16K静态RAM）各引脚对比表。2716要求用一个脉冲(50ms)写进一个地址单元，可是Z80的$\overline{WR}$信号脉冲宽度约360ns（时钟=2.5MHz）。在每个地址上Z80写动作如果达到15万次当然是相当可以了。这里用了单稳多谐振荡器的74LS122制成定时计数线路，Z80在EP—ROM(2716)地址上写进时，想使$\overline{WR}$脉冲宽度自动成为50ms，为此就加入了一个$\overline{WAIT}$信号。

这里解释一下$\overline{WAIT}$信号。Z80从外部存储器读取数据和指令，其内部的读入顺序如

程序时＋25V

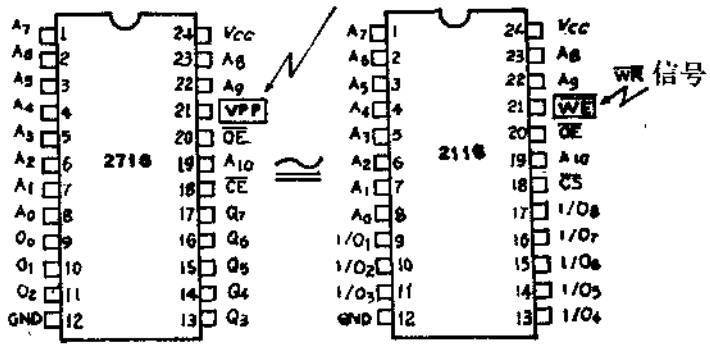

图14　2716(EP-ROM)同2116(16位静态RAM)的引脚比较

图15所示(定时图)，首先发出地址信号，跟着此地址发出表明它是存储器地址的(MREQ)信号，最后把从该存储器地址"读"的信号(RD信号)送到外部。也就是从RD信号送出约520ns以后，开始取进数据。

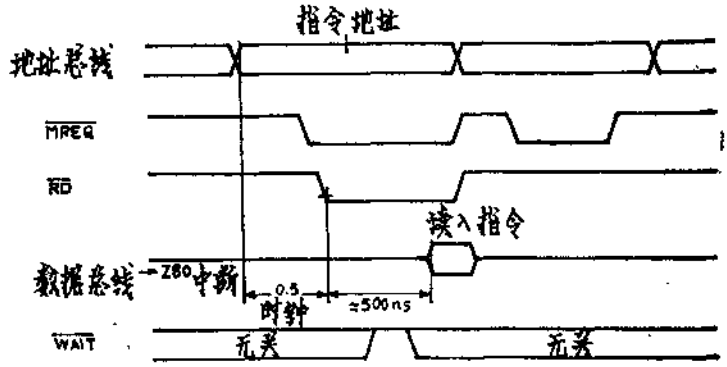

图15　取指令定时

从图上可以看出，必须在500ns时间内把正确的数据送回Z80。可是这对读出时间稍长的存储器（存取时间长的存储器）来说，需要Z80稍予等待。于是就增设了这样一个WAIT引脚。

Z80输出RD信号300～400ns之后，Z80就要检查WAIT引脚是高电平还是低电平。如果是高电平，就要象前面说过的那样，从RD信号送出约500ns之后，从数据总线上向内部读入存储器数据。如果此引脚是低电平，CPU就在每400ns时测试一下（取样）WAIT引脚的输入值，如果是高电平就一直处于等待状态。

因为WAIT引脚使Z80的动作稍予等待，所以就可以适应低速存储器，也就可以进行单步操作（一条条指令执行，停止）（图16）。

可是，在Z80向存储器写进数据时，WAIT引脚的动作也不是一帆风顺。实际上WR（写入）操作与RD的情况稍有不同，因为在判定WAIT值的定时时刻还未发出WR信号

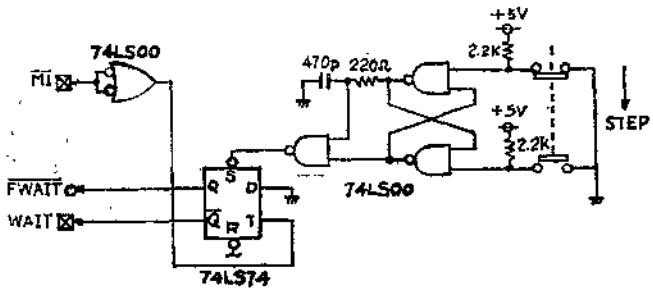

图16　模型计算机的单步电路

（图17）。在进行存储器读取时，靠$\overline{WAIT}$使CPU稍似等待，看来是没有问题的，但是在一个地址上写入数据时加进$\overline{WAIT}$信号，恐怕就不那样简单。

在模型计算机的EP—ROM上写进数据时，希望使$\overline{WAIT}$引脚保持50ms的低电平。因此这里把向EP—ROM写入的指令重复执行2次，根据第一次指令执行时发出$\overline{WR}$信号来驱动50ms的定时计数线路（单稳多谐振荡器），第二次指令的$\overline{WR}$再加上$\overline{WAIT}$信号。

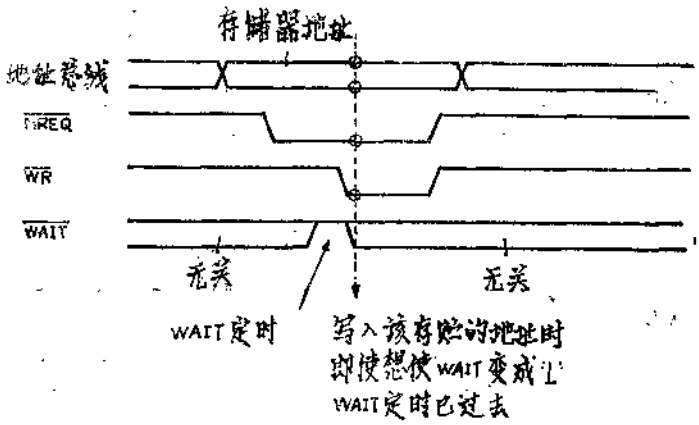

图17　$\overline{WR}$时的WAIT定时

但是，如图18那样，在第一个格的$\overline{WR}$和第二个格的$\overline{WR}$之间，因为有一个要取第二格指令的$\overline{RD}$周期，此$\overline{RD}$周期就会加上$\overline{WAIT}$引脚信号。因此如果在取指令时插入一个封闭$\overline{WAIT}$引脚的逻辑电路，就如同在第二个格的$\overline{WR}$时加上$\overline{WAIT}$引脚目的是完全一样的。

一个存储器读(RD)信号，在读取指令情况下，Z80把$\overline{M1}$引脚置成低电平。正是利用这一点，就有了如图19的逻辑线路。

$\overline{M1}$信号对于Z80系列外围用的集成电路IC也是很重要的。另外，使Z80停止（单步动作）的各项指令也都有效地利用$\overline{M1}$信号。

Z80具有用作存储器地址的0000H～FFFFH的64K字节，用作输出/输入地址的同样的0000H～FFFFH的64K个口*。因此对于各地址和口都能进行$\overline{RD}$，$\overline{WR}$（读/写）组

*译者注：原文如此。