83750-5

'84 INTEL

微型计算机系统器件手册

(五)

上海交通大学 微机研究所
科技交流室
1986.12

本册译者：李世祥、韩朔暸、徐子亮、王洪澄、赵正校、
张光耀、王建中
总 审 校：徐子亮、杜毅仁

'84INTEL微型计算机系统器件手册(五)

目　　录

第六章 应 用

概要

UPI—41A 的设计成功，使其能全部占据从低速到中速的广阔变化的外设接口应用的领域，着重地弥补了其它接口在灵活性和易实现性方面的缺陷。

键盘编码器

图6-1图示了利用 UPI 和 8243 I/O 扩展器以扫描 128 键阵列的键盘编码器结构。编码器具有开关阵列扫描逻辑，N-键翻转逻辑，ROM 查表，FIFO 字符缓冲器和作为显示功能的附加输出端，控制键或其它特定的功能。

端口 1(PORT1)和端口 4-7(PORTs4～7)提供了到键盘的接口。端口 1 的通道每置放一次就选择不同的键阵行(键的矩阵行)。

当给某一行加电压的时候，这行中所有的 16 列(即端口4-7的输入端)都被采样以确定在这行中有那一个开关是合上的。因为 UPI 指令集中包含的单独的位设置/清除操作及单字节或双字节指令都能直接对扩展器的端口 4-7 寻址，所以扫描软件是高效的代码。同样，在一个单操作中也能测试累加器位。对128 键扫描的时间大约需要 10ms。每一个阵列点具有一个独一无二的二进制代码。当发现一个键闭合时，这二进制代码就用来对 ROM寻址。ROM的第三页包含着一个具有和每一个键相对应的可用代码(例ASCII，EBCDIC等)的查询表。当发现一个有效键闭合时，为了传送到主处理器，就将与这个键相应的ROM代码存放在 FIFO缓冲数据存贮器中。为了避免丢失干扰和开关的抖动，在确定这个键有效并将代码放到 FIFO 缓冲器之前必须通过二次连贯的扫描以确定键的闭合。一旦按下去多重键而无需考虑何时释放它们时，FIFO 缓冲器允许处理多重键，一个条件辨识作为 N—键的翻转。

这一编码器的基本特征是完全标准的，并仅需要大约 500 字节的存贮器。由于UPI 是可

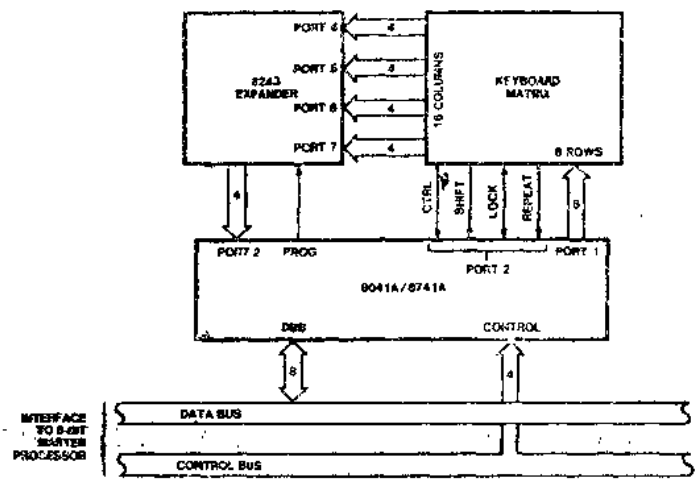

图 6-1 键盘编码器结构

编程的并具有附加存贮器的能力，所以它能处理大量不同任务。例如，特殊的键应被编程以给出一个既能闭又能开的通道。同样，I/O通道对控制16位数的7段显示是有用的。UPI也能被编程以识别象指令那样的特殊符号的组合，然后仅将译好的信息传到主处理器中。

已经展示了关于UPI—41A的完整的键盘应用。在这节中作了描述，在英特 Insite 库中关于应用的代码是有用的(程序AB147)。

点阵打印机接口

在图6-2中的点阵打印机接口是一个关于UPI—41A的典型应用。实际的打印机机械装置可以是为数众多的点阵型打印机的任何一种。大致结构能用鼓轮，球形头，菊形轮及链带印刷器来表示。

总线结构的展示描述了一个8位系统总线结构的概况。为了高效，及二路数据传送，UPI的三态接口端口和异步数据缓冲寄存器允许它直接接到系统的打字机上。

UPI的二个片内I/O端口提供16个输入信号以控制打印机的机械装置。定时器/事件计数器被用来产生一个时序序列，以控制打印头的位置，行推进，托架返回和其它序列。芯片中的程序存贮器提供关于5×7，7×9或其它点阵形式的字符的产生。作为一个附加的特性，64×8位数据存贮器的一部份能用作FIFO缓冲器以致于主处理器能以高的速率发送一个数据块，接着UPI以一个打印机能够接收的速率从缓冲器中输出一个字符。在此期间，主处理器转到其它任务上。

8295打印机控制器是一个8041A作为点阵打印机接口的预编程的例子。

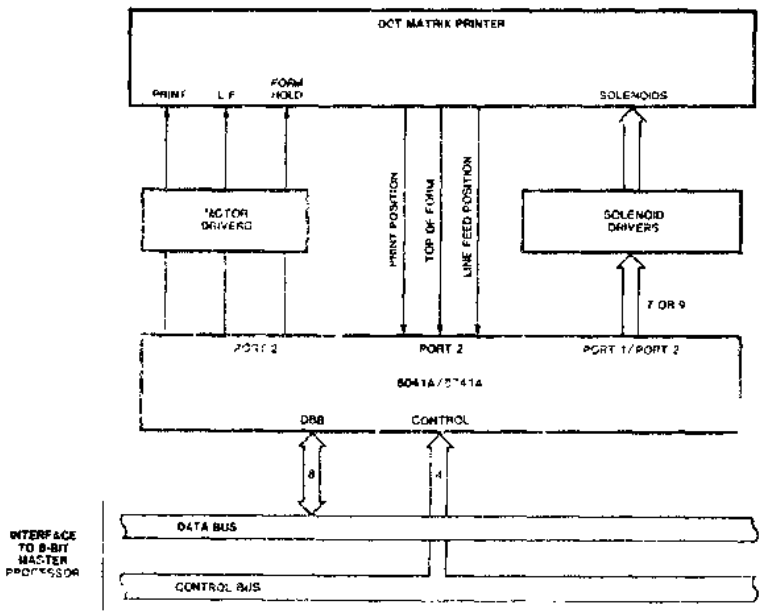

图 6-2 点阵打印机控制器

盒式磁带控制器

图6-3图示了一个由UPI—41A负责完成的数字盒带接口。通过UPI控制磁带传送的两个部分：数字数据/指令逻辑和马达伺服控制。

马达伺服需要一个其宽度与期望速度或比例的单稳脉冲形式的速度基准值。UPI监视

来自磁带的预先录下的时钟，并且用它的在芯片上的内部定时器，在每个时钟的过渡期产生一个符合需要的速度基准。

通过数据/指令逻辑连续不断地提供来自磁带的数据，并通过UPI 变换成8位字，然后传送到主处理器中。以10ips(英寸/秒)的磁带速度，UPI 能容易地处理 8000bps(位/秒)的数据速率。为了记录数据，UPI 用二条到数据/指令逻辑的输入控制线以控制在记录头中的数据流动方向。UPI 也能监视来自磁带传输包括：磁带的结束，插入盒带，忙和写许可的 4 条状态线，通过UPI 的两个 I/O端口能处理所有的控制信号。

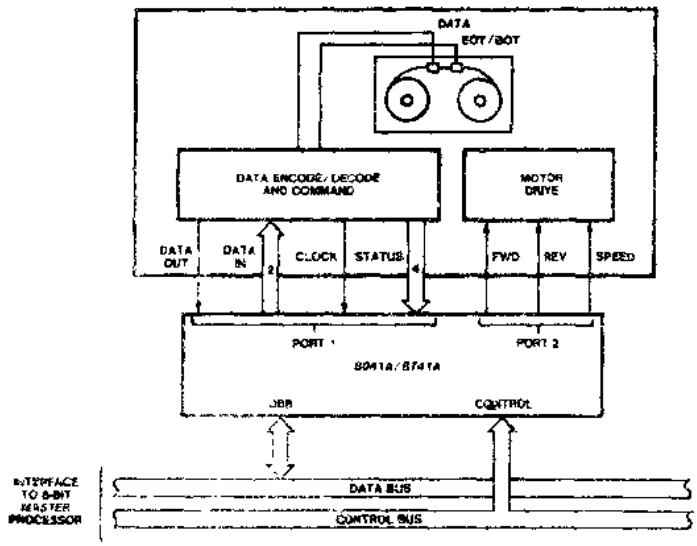

图 6-3　磁带传输控制器

通用I/O接口

图6-4所示为一个基于UPI的I/O接口设计图样。这一结构包括12条平行 I/O线和一个适于1200波特的数据传送串行接口(RS232C)。这一设计型式能用来作为一个主处理器到象一个串行通信通道那样的外设器件板系列的接口。

在这个例子中端口 1(PORT1)严格地用作 I/O，端口 2(PORT2)的引线提供 5 种功能。

- P23—P20　　I/O线 （双向）
- P24　　请求传送 （RTS）
- P25　　清除传送 （CTS）
- P26　　到主处理器的中断
- P27　　串行数据输出

平行的 I/O引线利用UPI 的双向端口结构。任何一个引线都能作为输入端或输出端。通过系统的$\overline{\text{RESET}}$ 脉冲可自动地将所有的端口引线初始到1并保持锁存。一个外部TTL信号连接到一个信号端口的引线上，将使得UPI 的50kΩ 内部提升电阻无效，以使得一条 INPUT 指令能正确对 TTL信号进行采样。

四条端口 2(PORT2)的引线作类似于端口 1(PORT1)的常用 I/O运行。同样，当 UPI

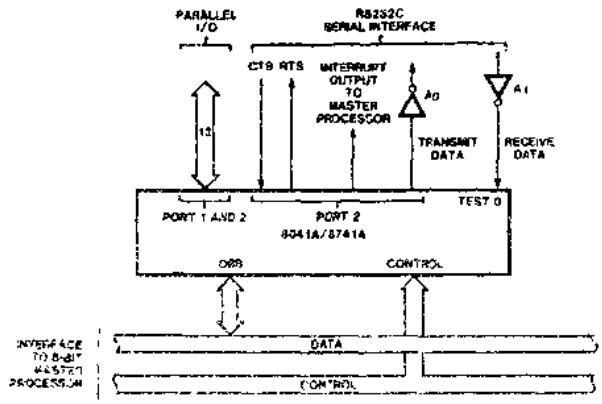

图 6-4 通用I/O接口

有串行数据被发送时，在软件的控制下，在端口2上产生 RTS信号。当允许UPI 传送串行数据时，经过端口 2 来监视 CTS信号。端口 2 的引线也能用作主处理器的软件产生中断。当UPI 有一个要传送的数据字节时或当UPI 准备好服务时，中断将作为一个服务请求。ENFLAGS 指令能用来产生在 P24 和 P25 上的 OBF 和 IBF 中断。

利用TEST0引脚作为服务输入端和端口 2 引脚作为传送输出端来完成 RS232C 接口。外部封装(A_0，A_1)用来提供 RS232C驱动的需要，串行服务软件由中断驱动并用芯片中的是定时器完成时间临界的串行控制。在检测到一个起始位后，能重置间隔定时器以在一个适当的时候为采样串行位流而产生一个中断。这就排除了对软件定时循环的需要，并且允许处理器在串行数的采样之间去处理其它任务(例如并行的 I/O操作)。利用软件标志使主程序能够决定何时驱动中断，以接收有一个汇编了的字符的程序。

结构的这一类型允许系统设计者在作为具有一个装配好的字符串行和并行的 I/O应用中设计常规的 I/O接口方面具有灵活性。例如：如果需要的话，在使用平行 I/O的地方能取代第二或第三串行通道。UPI的数据存贮器能为 4条低速通道(110波特的电传打字机)缓冲数据和指令。

应用说明

如上的应用说明了UPI 系列的各种不同的应用，其它的通过英特文献出版机构能获得包括8048系列应用手册在内的有关文献。

关于 UPI-41A™的介绍

引　言

自从在1974年推出如8080那样的第二代微处理器以来，一个外设接口器件的广阔领域已经展现出来。起先，这些器件解决了一般性的应用问题。例：平行接口(8255)，串行接口(8251)，计数器(8253)，中断控制器(8259)。然而随着LSI(大规模集成电路)技术的速度和集成度的发展，越来越多的智能被集成到外设器件中，这就允许更特殊的应用问题能得到解决，例如，软盘控制器(8271)，CRT控制器(8275)和数据链路控制器(8273)。归属于扩展外设器件智能的系统设计者的优点是许多原来是归于主处理器外设控制任务，现在由外设硬件外部地处理掉了，而无需通过内在于主处理器的软件。这就减少了主处理器的开销，从而增加了系统的能力和减少了软件的复杂性。

尽管有大量的外设器件可用，但由于微机的普及已经开始，以致于单靠提供LSI仍无法满足大量外设控制的应用。"复杂"这个问题是事实。这是因为正在出现新的应用要比对发展新的，致力于外设控制器的有影响的制造业来得快。提出了这些问题以后，便开发了一个新的以微机为基础的通用外设接口(UPI—41A)。

实际上，UPI-41A充当了一个到主系统CPU的从处理器。UPI包含自身的处理器，存贮器和I/O，而且完全是用户可编程的。这就是说，全部的外设控制算法能在UPI中局部地被编程，以取代主处理器的主内存的负荷，这一分配处理的概念允许UPI在主处理器在处理象缓冲器的管理或算术那样的非实时应变任务时去管理象键编码，打印机控制及多路显示那样的实时任务。UPI仅在初始化，基本命令，和数据传送时依靠主处理器。由于对于处理器主UPI和从UPI来说，它们两者的工作是平行的。这一技术导致了在系统效率上的全面增加。

应用说明提出的三个UPU-41A应用可以简单地分成二组：一组是复合的应用，即UPI编码速率的规定允许它们不是以独立的形式存在就是结合作为一个多任务的UPI中的合理的一个任务，另一组应用是一个对于它们自己是一个完整的UPI应用。在第一组中的应用是一个简单的LED显示和传感器阵列控制器，组合的串行/并行I/O器件是第二组应用中的一个，每一种应用表明不同的UPI配置和特征。然而，以前介绍的应用细节中，就包括着UPI和主处理器的协议规定。这些协议规定是UPI软件开发的关键。为了方便起见，UPI的方框图将重现在图1中，并在表1中列出简明的指令集。

UPI—41VS，UPI—41A

UPI—41A是UPI—41的增强性版本。它组合了几个在"非A"器件上找不到的结构特征。

- 独立的数据输入和数据输出的数据总线缓冲寄存器。
- 用户能定义的STATUS(状态)寄存器位。
- 关于OBF和$\overline{\text{IBF}}$标志的可编程主中断。
- 到外部DMA控制器的可编程DMA的接口。

独立的数据入(DBBIN)和数据出(DBBOUT)寄存器大大地简化了与UPI—41兼容的主处理/UPI协议，主处理器仅需要在对DBBIN写之前检查IBF及在读DBBOUT之前检查OBF。需要没有数据总线缓冲器的lock—out。

最有效的状态(STATUS)寄存器的4位字组，在UPI—11中不定义，而是由用户对UPI—41A定义。它可以从最有效的累加器的4位字组直接装入(MOV STS, A)。这些额外的4个状态(STATUS)位对传送附加状态信息到主处理器是有用的，应用说明广泛地利用了这一特征。

一条新的指令(EN, FLAGS)允许在端口2的4号位，(PORT2BIT4)和端口2的5号位(PORT2 BIT5)上独立反映OBF和$\overline{IBF}$。当这些引脚是到主处理器的中断源的时候，这一特征允许中断驱动的数据传送。

通过执行一条EN DMA指令，使得端口2的第六位(PORT2 BIT6)成为一个DRQ(DMA请求)输出端，而端口2的第7位(PORT2 BIT7)则成为$\overline{DACK}$(DMA应答)。设置DRQ为外部DMA控制器请求一个DMA周期。当DMA周期被准许时，DMA控制器将返回一个或是$\overline{RD}$(读)或是$\overline{WR}$(写)的DACK脉冲。DACK自动地强制内部拉低($\overline{CS}$和A_0)，并清除ORQ。这就为DMA传送选择适当的数据缓冲寄存器(DBBOUT适合于DACK和$\overline{RD}$, DBBIN适合于DACK和$\overline{WR}$。)

类似于"非A"、UPI—41A。在ROM(8041A)和EPROM(8741A)程序存贮器版本中都是有效的。由于应用中利用了"A"的增值特性，所以记录下的应用唯一地涉及到UPI—41A。

UPI/主处理器协议

在最紧密地连接的多处理器系统中，各种不同的处理器是通过共享资源来进行通信

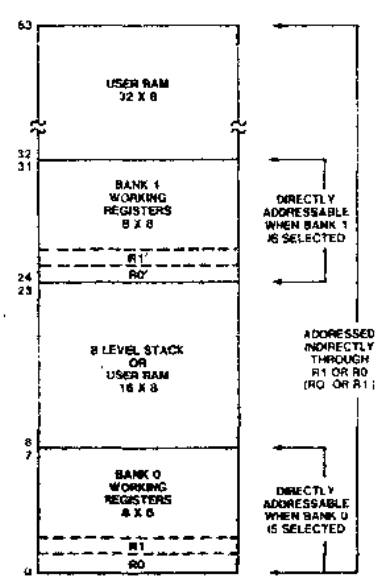

图 1A 程序存贮器映象

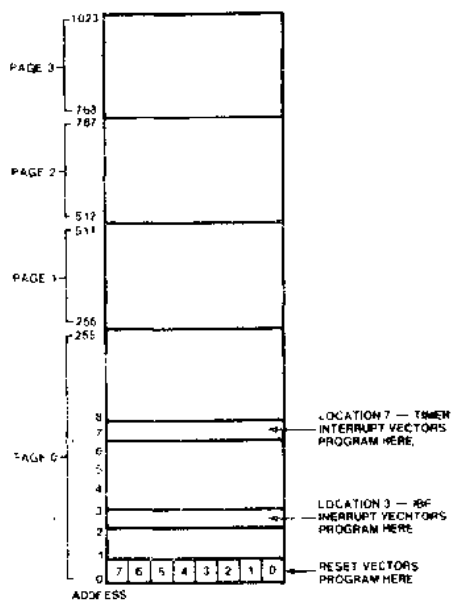

图 1B 数据存贮器映象

的，这个共享资源在 RAM 中或在贯穿传递状态和数据的寄存器中是典型特殊点。在一个主处理器和一个UPI—41A的情况中，共享资源对 UPI 是独立的，主处理器能寻址的和寄存器内部的。这些寄存器是状态寄存器(STATUS)，数据总线缓冲输入寄存器 (DBBIN) 和数据总线输出寄存器(DBBOUT)。[数据总线缓冲器方向与UPI有关]。为了说明这个寄存器的接口，请参阅图2中的 8085A/UPI 系统。

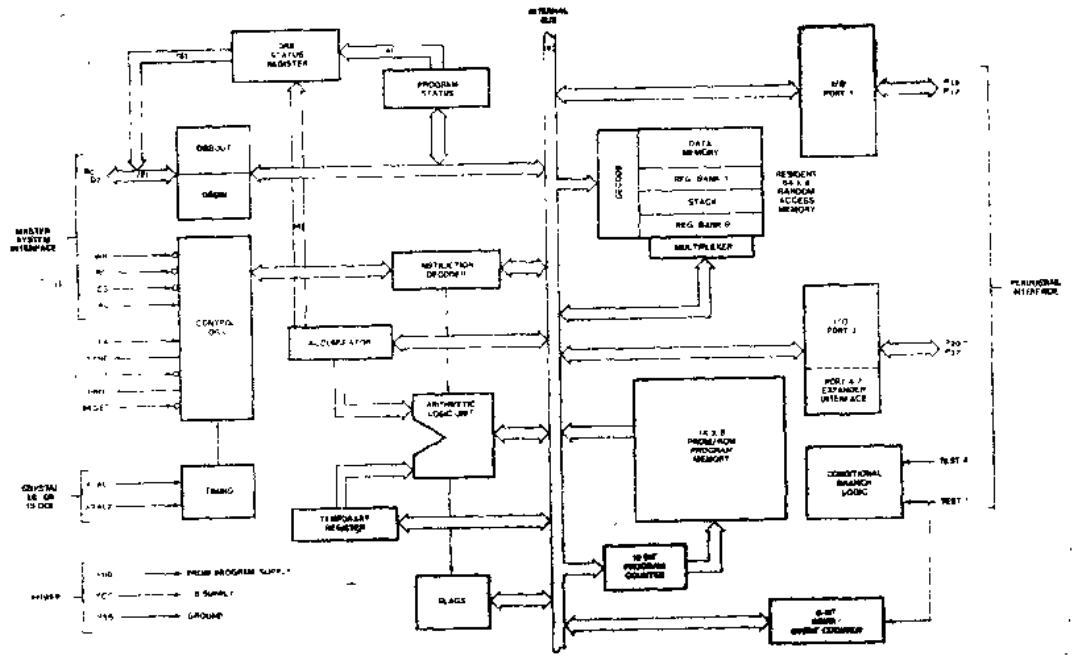

图 1C　UPI—41A方框图

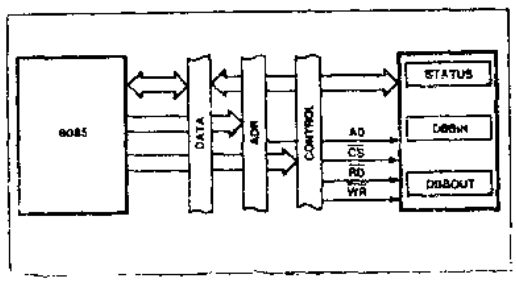

图 2　寄存器接口

从8085A向UPI里面看，8085A 仅看到上面提到的三个寄存器，如果8085A期望向UPI发出一条指令，它同样是依靠表2的译码通过写指令到 DBBIN寄存器来完成。供给UPI 的数据也是经过DBBIN寄存器传递的(UPI 是通过检查A。引脚来区分指令和数据的，合理的方向是做到短期地屏蔽)。来自UPI 供给8085A的数据在 DBBOUT 寄存器中被传递。通过读UPI的状态(STATUS)寄存器，8085可以询问UPI状态。状态(STATUS)寄存器的 4 位可以用作标志和用来传递 UPI 入和出的数据和命令。状态(STATUS)寄存器的形式示于图

3 中：

位BIT0 是OBF(输出缓冲器满)。这一标志为主处理器指明何时数据已经放入UPI 的DBBOUT寄存器中。当 UPI 写数据到 DBBOUT时使OBF置位而主处理器读DBBOUT时清除OBF。仅当 OBF 被置位时，主处理器才找出 DBBOUT 寄存器中有意义的数据。

输入缓冲满(IBF)标志是位BIT1。UPI 利用这个标志作为一个主处理器已经写数据到DBBIN寄存器中的指示器。主处理器用IBF来指示何时UPI 已经收到一个特定的指令或数据字节。在主处理器输出任何东西到UPI之前，都要检查IBF。当主处理器对DBBIN 写的时候使IBF置位，而当UPI 读DBBIN 时则清除 IBF。主处理器必须等到 IBF = 0 以后才能将新的数据或指令写到DBBIN中去。相反地，在UPI读取DBBIN之前都必须保持IBF = 1。

第三个STATUS(状态)寄存器位是 F_0(标志0)。这是一个能被 UPI 置位、清除、和测试的通用标志。其典型用法是向主处理器指出一个UPI 错误或"忙"条件。

标志1(F_1)是提供的最后的状态(STATUS)位。象F_0一样，UPI 也能对这一标志进行置位，清除和测试。然而每当主处理器对DBBIN寄存器进行写的时候 F_1 反映出 A_0 引脚的状态。UPI利用这一标志来描述主处理器的命令和数据写到DBBIN中。

保留的 4 个状态(STATUS)位是用户可定义的。这些位的典型用法是作为一个多任务的UPI 中指出独立的任务的状态或作UPI 的状态产生中断。这些位我们为了一个在即将到来的应用中使用的广阔的领域。

从UPI向8085 看去，UPI 看到的是二个DBB寄存器 加上IBF，OBF 和 F_1 标志。UPI能从它的累加器写到DBBOUT或将DBBIN 读入累加器。UPI 不能直接读取 OBF，IBF或 F_1，但这些标志可以用条件跳转指令来测试。UPI 将完全可以做到在写新数 据到 DBBOUT 之前清除OBF以保持主处理器已经读到先前的DBBOUT数据。由于 DBBIN 数据仅在 IBF 被置位时才是有效的，所以在读DBBIN之前也将测试IBF状态。象先前提到的那样，当 IBF 置位时，UPI 用 F_1 去区别在 DBBIN中的指令和数据的内容。UPI也可以将累加器的高4位写到STATUS(状态)寄存器的高 4 位中去。这些位是用户可定义的。

表 1　指令集概述(参阅第三章的译文中的表)

表 2　寄存器译码

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	寄　存　器
0	0	0	1	读DBBOUT
0	1	0	1	读STATUS
0	0	1	0	写DBBIN[DATA(数据)]
0	1	1	0	写DBBIN[COMMAND(指令)]
1	X	X	X	没有作用

UPI 能在它的内部程序执行期间，随时测试标志位。为了变换有必 要 查 询 STATUS (状态)寄存器。如果主处理器的指令和数据需要快速响应，则可利用 UPI 的内部中断 结

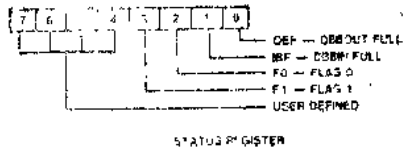

图 3 状态寄存器形式

构。如果允许IBF中断，则主处理器对DBBIN的写入(或是指令或是数据)将使IBF置位，同时产生一个位于程序存贮器中 03H 处的内部调用。在这一点上，利用寄存器组的切换能存放工作寄存器的内容，累加器的内容存放到剩下的工作寄存器中，且 DBBIN 寄存器被读取和服务。有关 IBF 中断的中断逻辑表明在图4中。几个涉及到这一逻辑的评述是适当。注意，如果主处理器对 DBBIN 进行写入时 UPI 仍然在对前一个 IBF 中断进行服务(一条 RETR 指令还没有被执行)，将使IBF 中断挂起线变高，这将引起一个新的到 03H 的调用(CALL)，随即便首先执行一个RETR指令。无需用 EN I(允许中断)指令去重新配置中断逻辑作为8085A或8080系统中的需要；RETR 指令完成这一功能。同样也要注意：执行一条DIS I 指令以屏蔽更进一步的 IBF中断无需清除一个挂起的中断。只有一条调用到位于 03H 的指令或 RESET 信号才清除一个挂起 IBF中断。

记着，实际的主处理器/UPI 协议是根据应用的或许通过例子是说明正确协议的最好方法。让我们考虑利用UPI作为一个简单的并行I/O器件(这是一个平凡的应用，但它体现了所有重要的协议所需涉及到的东西)。由于UPI 内部可被中断和非中断驱动，两者都被考虑到了。

让我们首先采用最容易的结构；利用UPI的端口(PORT 1)作为一个 8 位的输出端口。从UPI的观点来看，由于所需的是UPI 从主处理中输入数据，则其仅是一个输入的应用。

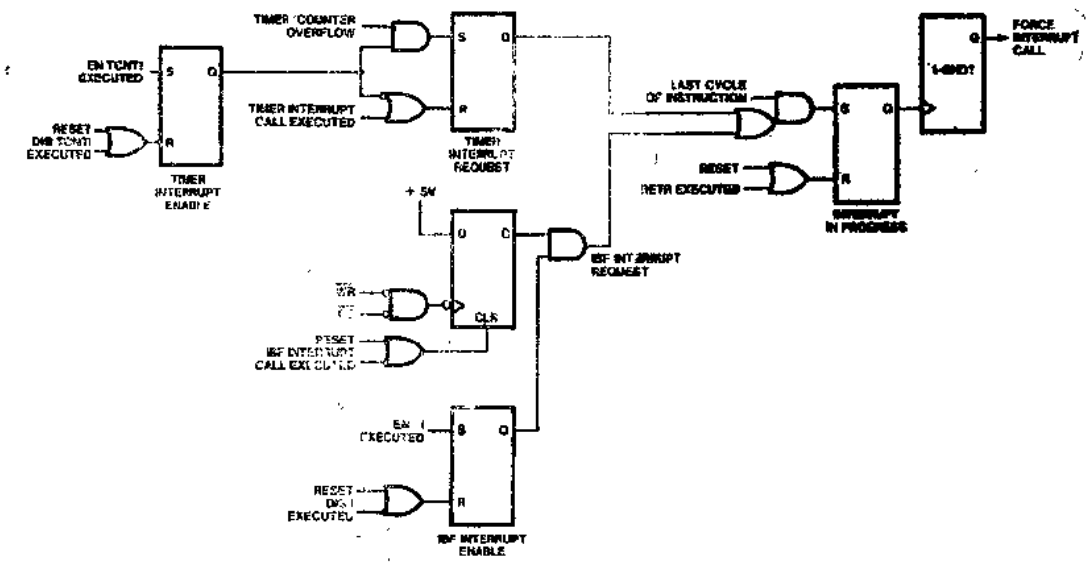

图 4 UPI—41A中断结构

一旦主处理器将数据写入UPI，则UPI读取 DBBIN寄存器并将数据传送到端口1(PORT1)。由于UPI 懂得它只完成一个任务——不需要指令，所以无需测试与数据相对的指令。

非中断驱动的 UPI 软件示于图 5A 中。而图5B说明了基本中断的软件。对于图 5A，UPI 简单地等待直到它看到 IBF 变高，指明主处理器已经将一个数据字节写 到 DBBIN中，UPI 然后读取 DBBIN，将它传送到端口 1(PORT1)，并返回到等待下一数据。对于中 断驱动的UPI，图5B，一旦执行了EN I指令，在处理数据前UPI 简单地等待 IBF中断。在这个等待期间，UPI可以处理其它的任务。当主处理器写数据到DBBIN时，将产生一个IBF 中断，它完成一个到03H处的调用。在这点上，UPI读取 DBBIN(由于一个 IBF 中断意味着 IBF 被置位，所以无需测试 IBF)，将数据传送到端口 1(PORT1)并执行一个将程序 流返回主程序的RETR。

关于主系统8085A的软件包含在图 5C中。对于主系统输出数据到UPI的要求仅是在写新数据之前，要检查UPI 以确定先前的数据已被取走。为完成这一步骤，在写下个数据之前，主处理器简单读取状态寄存器并寻找 IBF = 0。

图6A表示 UPI 端口2(PORT2)用作一个8位输入端口的情况。当主处理器不对UPI 进行写入(输入)而简单地读取STATUS(状态)或DBBOUT寄存器时，这一结构被称为UPI输入。在这个例子中，仅利用OBF状态，OBF发信号给主处理器，说明UPI已经在DBBOUT 放入新的数据。UPI 循环测试 OBF 被清除时，表明主处理器已经读完先前的数 据。然 后 UPI接着读取它的输入端口[端口 2(PORT2)]，将这一数据置放在 DBBOUT 中。接着随着 OBF 等待，直到再次读取输入端口之前由主处理器来取 DBBOUT。当主处理器希望读 取输入端口数据，图6B，在读 DBBOUT 之前，它将简单地检查一下在状态寄存器中的OBF 的设置情况。当这一技术说明适当的协议时，由于主处理器总是得不到端口的最新状态，它将记录下这不是 1 个利用UPI 作为输入端口的好方法。

```
;UPI INPUT ONLY EXAMPLE—PORT 1 USED AS OUTPUT PORT
;       UPI POLLS IBF FOR DATA
;
RESET   JNIBF   RESET       ;WAIT ON IBF FOR INPUT
        IN      A,DBB       ;INPUT THERE SO READ IT
        OUTL    P1,A        ;TRANSFER DATA TO PORT
        JMP     RESE        ;GO WAIT FOR NEXT DATA
```

图 5A 单一输出端口举例——查问

```
;UPI INPUT ONLY EXAMPLE—PORT 1 USED AS OUTPUT PORT
;       DATA INPUT IS INTERRUPT-DRIVEN ON IBF
;
RESET   EN      I           ;ENABLE IBF INTERRUPTS
        JMP     RESET+1     ;LOOP WAITING FOR INPUT
IBFINT  IN      A,DBB       ;READ DATA FROM DBBIN
        OUTL    P1,A        ;TRANSFER DATA TO PORT 1
        RETR                ;RETURN WITH RESTORE
```

图 5B 单一输出端口举例——中断

```
;8085 SOFTWARE FOR UPI INPUT ONLY EXAMPLE
;       DATA FOR OUTPUT IS PASSED IN REG C
;
UPIOUT  IN      STATUS      ;READ UPI STATUS
        ANI     IBF         ;LOOK AT IBF
        JNZ     UPIOUT      ;WAIT FOR IBF=0
        MOV     A,C         ;GET DATA FROM C
        OUT     DBBIN       ;OUTPUT DATA TO DBBIN
        RET                 ;DONE RETURN
```

图 5C 关于单一输出端口举例的8085A代码

```
;UPI OUTPUT ONLY EXAMPLE - PORT 2 USED AS INPUT PORT
;       PORT DATA IS AVAILABLE IN DBBOUT
;
RESET   JOBF    RESET       ;LOOP IF OBF=1 (DATA NOT READ)
        IN      A,P2        ;DBBOUT CLEAR, READ PORT
        OUT     DBB,A       ;TRANSFER PORT DATA TO DBBOUT
        JMP     RESET       ;WAIT FOR MASTER TO READ DATA
```

图 6A 单一输入端口举例

我们能很容易地将上述的例子组合起来。图7表明用端口 1(PORT1 作 为一个输出 端口同时用端口2(PORT2)作为一个输入端口的UPI 软件。程序开始于检查UPI 的 IBF 来察看主处理器是否已经将预定输出端口的数据写到 DBBIN。如果 IBF 被置位，则 UPI 读取

DBBIN并将数据传送到输出端口上[端口1(PORT1)]。如果没有被置位或者是数据传送到输出端口上(如果这个端口是输出端口)，则OBF被测试。如果 OBF 被清除(指出主处理器已经读完DBBOUT)，则输入端被读取[端口2(PORT2)]并传送到DBBOUT。如果OBF 被置位，则主处理器仍然读取 DBBOUT 使程序反复循环测试 OBF。

主处理器软件对独立的输入/输出例子是相同的。在写输出端口数据到DBBIN之前或者从DBBOUT中读取数据到输入端口之前主处理器必须分别地测试IBF或OBF。

```
; 8085 SOFTWARE FOR UPI OUTPUT-ONLY EXAMPLE
;       INPUT DATA RETURNED IN REG. A
;
UPIIN   IN      STATUS      ; READ UPI STATUS
        ANI     OBF         ; LOOK AT OBF
        JZ      UPIIN       ; WAIT UNTIL OBF=1
        IN      DBBOUT      ; READ DBBOUT
        RET                 ; RETURN WITH DATA IN A
```

图 6B 8085A单一输入端口代码

```
;
; UPI INPUT/OUTPUT EXAMPLE—PORT 1 OUTPUT, PORT 2 INPUT
;
RESET:  JNIBF   OUT1        ; IF IBF=0, DO OUTPUT
        IN      A, DBB      ; IF IBF=1, READ DBBIN
        OUTL    P1, A       ; TRANSFER DATA TO PORT 1
OUT1:   JOBF    RESET       ; IF OBF=1, GO TEST IBF
        IN      A, P2       ; IF OBF=0, READ PORT 2
        OUT     DBB, A      ; TRANSFER PORT DATA TO DBBOUT
        JMP     RESET       ; GO CHECK FOR INPUT
```

图 7 组合的输出/输入端口举例

在上述所有三个例子中，UPI 处理来自主处理器唯一作为数据的信息。由于应用不需要指令，因此不需常检查DBBIN 的信息是指令而不是数据。但是端口 1(PORT1)和端口2)PORT2)都用作输出端口怎么办？UPI需要知道放数据到哪个端口上，让我们用指令来选择这个端口吧。

通过DBBIN恢复指令和数据，在对DBBIN写的时候，A_0 引脚的状态被用来区别命令和数据。通过协商决定，具有 $A_0=0$ 的 DBBIN 写是数据，而有 $A_0=1$ 的 DBBIN 写是指令。当DBBIN被写入时，将F_1[标志1(FLAG1)]设置成A_0的状态，UPI 测试 F_1 以决定在 DBBIN 寄存器中的信息是数据还是指令。

关于两个输出端口的情况，让我们假定主处理器在写数据之前选择期望具有指令的端口(我们只能用F_1作为端口的选择，但不能表示指令和数据之间的微妙的区别)。让我们定义端口选择指令：如果下个数据是对于端口1(PORT1)的话，则BIT1=1(写 PORT1=00000010)，而如果下个数据是对于端口2的(PORT2)，则 BIT2=1，被设置位的号码选择端口而忽略其它的任何位。这一分配是完全任意的，我们能用任何指令结构，但这种方式具有简便的优点。

注意，UPI 必须"牢记"，从 DBBIN 写到已经被选择好的写端口上。让我们把 F_0[标志，(FLAG0)]用于这一目的，如果收到一条写端口 1(PORT1)的指令，则清除 F_0。如果指令是端口2(PORT2)，则使 F_0 置位。当UPI 在 DBBIN 中发现数据时，将询问 F_0，并将数据装入到先前选择的端口中。UPI软件表示在图8A中。

开始，UPI简单地等待直到IBF被置位以指出主处理器已经将信息写入 DBBIN。一旦IBF被置位，则读取DBBIN，并对F_1作指令测试，如果$F_1=1$，则DBBIN 字节是指令，假定一条指令，测试第一位(BIT1)以观察是否指令选择端口 1(PORT1)，如果是这样的话，则清除F_0而程序返回以等待数据，如果 BIT1=0 测试 BIT2。如果BIT2被置位，则选择端口2(PRT2)同时也使F_0置位，接着程序反复循环以等待主处理器的下一个输入，这个输入是期望端口的数据。如果BIT2 没有被置位，则 F_0 不被改变，且无动作。

当再次检测到 IBF=1时，将再次对输入的指令或数据进行测试。当它必然是数据时，则读 DBBIN 并测试 F_0，以决定先前选择好的端口，然后将数据输出到那个端口，随后程

序等待下个输入。注意：由于F_0仍然是选择先前的端口，则下个输入能有更多的这个端口的数据。端口选择指令将能被看作一个端口选择触发器控制；一旦产生一个选择，可以再三地写数据到那个端口上，直到选中其它的端口。主处理器软件，见图8B，在对DBBIN写入数据或指令之前，必须简便地检查IBF。否则，主处理机软件是直线向前的。

```
;
; UPI DUAL OUTPUT PORT EXAMPLE—BOTH PORT 1 AND 2 OUTPUTS
;       COMMAND SELECTS DESIRED PORT
;       WRITE PORT 1—0000 0010 (02H)
;       WRITE PORT 2—0000 0100 (04H)
;
;       FLAG 0 USED TO REMEMBER WHICH PORT WAS SELECTED
;       BY LAST COMMAND.
;
RESET:  JNIBF   RESET       ; WAIT FOR MASTER INPUT
        IN      A, DBB      ; READ INPUT
        JF1     CMD         ; IF F1=1, COMMAND INPUT
        JF0     PORT2       ; INPUT IS DATA, TEST F0
        OUTL    P1,A        ; F0=0, SO OUTPUT TO PORT 1
        JMP     RESET       ; WAIT FOR NEXT INPUT
PORT2:  OUTL    P2,A        ; F0=1, SO OUTPUT TO PORT 2
        JMP     RESET       ; WAIT FOR NEXT INPUT
CMD:    JB1     PT1         ; TEST COMMAND BITS (BIT 1)
        JB2     PT2         ; TEST BIT 2
        JMP     RESET       ; NEITHER BIT SET, WAIT FOR INPUT
PT1:    CLR     F0          ; PORT 1 SELECTED, CLEAR F0
        JMP     RESET       ; WAIT FOR INPUT
PT2     CLR     F0          ; PORT 2 SELECTED, SET F0
        CPL     F0
        JMP     RESET       ; WAIT FOR INPUT
```

图 8A　双重输出端口举例

```
;
; 8085 SOFTWARE FOR DUAL OUTPUT PORT EXAMPLE
;       THIS ROUTINE WRITES DATA IN REG. C TO PORT 1
;       (SAME ROUTINE FOR PORT 2—JUST CHANGE COMMAND)
;
PORT1:  IN      STATUS      ; READ UPI STATUS
        ANI     IBF         ; LOOK AT IBF
        JNZ     PORT1       ; WAIT UNTIL IBF=0
        MVI     A, 00000100B ; LOAD WRITE PORT1 CMD
        OUT     UPICMD      ; OUTPUT TO UPI COMMAND PORT
P1:     IN      STATUS      ; READ UPI STATUS AGAIN
        ANI     IBF         ; LOOK AT IBF
        JNZ     P1          ; WAIT UNTIL COMMAND ACCEPTED
        MOV     A, C        ; GET DATA FROM C
        OUT     DBBIN       ; OUTPUT TO DBBIN
        RET                 ; DONE, RETURN
```

图 8B　8085A双重输出端口举例代码

为了完整起见，实现两个输入端口的UPI软件，在图9中给出。这种情况比双重输出的情况简单，这是因为UPI能假定所有对DBBIN写是端口选择指令，所以不需要指令/数据测试，一旦输入口读指令，则读取选择端口且端口数据被放到DBBOUT中。注意：在这种情况下F_0将用作一个UPI错误指示器。如果主处理器偶然地发出一个无效的指令(一条指令没有使BIT1或BIT2置位)，将使F_0置位通知主处理器，UPI不知道怎样解释这条指令。如果主处理器在从先前的指令已经读DBBOUT之前，命令一个端口读，则F_0也要被置位。UPI仅在装DBBOUT之前测试OBF，如果OBF=1，则使F_0置位以指出错误。

```
; P1 DUAL INPUT PORT EXAMPLE—BOTH PORT 1 AND 2 INPUTS
;       COMMAND SELECTS WHICH PORT IS TO BE READ
;       FLAG 0 USED AS ERROR FLAG
;
RESET:  JNIBF   RESET       ; WAIT FOR INPUT
        CLR     F0          ; CLEAR ERROR FLAG
        IN      A, DBB      ; READ INPUT (COMMAND)
        JB1     PT1         ; TEST BIT 1 (PORT 1)
        JB2     PT2         ; TEST BIT 2 (PORT 2)
ERROR:  CPL     F0          ; ERROR—COMPLEMENT F0
        JMP     RESET       ; WAIT FOR INPUT
PT1     IN      A, P1       ; READ PORT 1
        JOBF    ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT     DBB, A      ; LOAD PORT 1 DATA INTO DBBOUT
        JMP     RESET       ; WAIT FOR INPUT
PT2     IN      A, P2       ; READ PORT 2
        JOBF    ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT     DBB, A      ; LOAD PORT 2 DATA INTO DBBOUT
        JMP     RESET       ; WAIT FOR INPUT
```

图 9　双重输入端口举例

所有上面的举例，尽管能容易地收编作为几个从事于UPI处理多重小任务的任务中的一个。但它们自己是相当平凡的UPI应用，我们已经包括了它们起初引进的UPI概念和说明了一些主处理器/UPI的协议。在转到UPI的更现实的应用之前，让我们讨论两个没有直接涉及到主处理器/UPI协议，但大大地提高UPI的传送能力的UPI的特性。

除在状态寄存器中的OBF和IBF位之外，这些标志也能在两个端口引脚上直接有效，然

后这些引脚能作为一个到主处理器的中断源。通过执行一条EN FLAGS指令，端口2(PORT2)的引脚4反映OBF的条件，端口2的引脚5反映IBF的相反的条件($\overline{IBF}$)。于是，这些提供输出口能通过其独立端口位的值被允许或禁止。例P24能反映OBF长达执行一条使P24置位指令(即ORL P2,#10H)。同样的作用施加到$\overline{IBF}$输出，除利用P25外。因而P24可以作为一个DATA AVAILABLE的中断输出服务。同样地，P25可作为一个READY-TO--ACCEPT—DATA的中断。这就大大地简化了中断-驱动，主-从处理器的数据传送。

UPI能提供一个DMA传送接口。如果执行一条EN DMA指令，则端口2(PROT2)引脚6成为一个DMA请求(DRQ)输出端，P27将成为一个高阻抗DMA应答$\overline{DACK}$输入端。任何指令正常地设置P26，即设置DRQ。当DACK变低，且是$\overline{RD}$或是$\overline{WR}$变低时，清除DRQ。当$\overline{DACK}$变低，内部强制$\overline{CS}$和$\overline{A_0}$变低时，允许产生存DBBOUT或DBBIN之间的数据总线传递，依靠$\overline{WR}$或$\overline{RD}$是正确的。当然，运行需要使用一个外部的DMA控制器。

现在我们已经讨论了UPI的协议和数据传递接口的概貌，让我们转到实际的应用上。

应用实例

下列三段的每一段将详细介绍了UPI应用的硬件和软件。每一个应用在最后一段中都用到提及的协议中的一个。第一个例子是简单的数字8LED显示控制器。这一应用仅要求UPI完成从DBBIN中输入的操作；不用到DBBOUT。对于第二个应用正好相反：一个传感器阵列控制器，最后一个应用包含DBBOUT和DBBIN的操作：一个组合的串行/并行I/O装置。

具有开发这些应用的核心主处理器系统是iSBC80/30单板计算机，由于它包含了一个接到UPI—41A奉献的适配器，这一单板机提供一个格外便利的UPI环境。80/30利用8085A作为主处理器，I/O和补充80/30的外设包含12个向量优先级中断(8个在8259可编程中断控制器上，4个在8085A本身中)。一个8253可编程间隔定时器提供三个16位可编程定时器(奉献一个作为可编程的波特率发生器)。通过一个8251可编程的USART提供一个高速串行通道，并由一个8255可编程并行接口来提供24条并行I/O通道。存贮器的定额配置包括采用2117 16k位动态RAM和8202动态RAM控制器，及采用插脚兼容的2716，2758或2332的ROM/EPROM。80/30的RAM采用一个双重端口的结构。这就是说，存贮器能被看作为一个完整的系统资源，无论是来自板中的8085A还是遥远的CPU或是通过MULTIBUS(多路总线)的其它器件都是可以访问的。80/30包含MULTIBUS的控制逻辑，其能允许多达16个80/30或其它总线主处理器共享同一个系统总线(关于iSBC80/30的更详细的信息和其它iSBC产品可以参阅最近的英特尔系统数据目录)。

iSBC80/30的方框图示于图10，UPI接口的详细表述在图11。这一接口以下列格式对UPI寄存器译码。

寄存器	操作
读 STATUS（状态）	IN ␣ E5H
写 DBBIN （指令）	OUT␣ E5H
读 DBBOUT（数据）	IN ␣ E4H
写 DBBIN （数据）	OUT ␣ E4H

8位数字多重LED显示

一个LED显示与一个微处理的接口的传统方法是对每一个显示的数字用一个数据锁存器跟着一个 BCD7 段译码器。这样对于每个数字需要二片集成电路，7个限流电阻和大约45个连接端。当然根据期望显示的数字的总数的需要将成倍地增加。该方法的明显缺点是需要很多的部件，且由于每个数位不断地"接通"，消耗大量的电源能量。用时间多重转换的显示方案代替前面的方法能用来减少部件的数目和电源消耗。

多重转换显示基本上包括以并行连接每个数字的同一段(a, b, c,d,e,f 或g)和驱动每个独立数字的公共数字单元(正极或负极)。如图12所示。显示的各种不同的数字不是立即同时接通的，更确切地说，每次仅给一数字加电。当给每个数字加电时，那个数字的相应段被接通。在这种方法中打开的每个数字，是连续地并是以保持每个数字的接通状态的速度进行：这就意味着显示必须周期间隔地刷新以保持数字的无闪耀。如果 CPU 取得这一处理任务，则将导致中止正常的操作，转到更新显示，然后再返回到它的正常的程序流上。通过UPI能理想地处理这一额外的任务，主处理器只要简单地给UPI字符并让 UPI 完成现行段的译码，多重反复显示和刷新，

作为这一技术的例子，图13表明了UPI控制一个8位数字 LED 显示。所有的数字段都是并行地连接的，并通过UPI的端口1(PORT1)，通过段驱动器驱动它们，端口2(PORT2)的低3位将被输入到一个3—8译码器中以选择一个穿越数字驱动器的独立的数字。利用端口2第四通道作为打开译码器的输入，余下的端口通道加上 TEST0(测试0)和(测试 1)TEST1输入对其它任务是可用的。

实际上，UPI 利用处在间隔定时器模式中的计算器/定时器定义显示刷新的间隔一旦用期望的间隔装入定时器并启动，则UPI 处理其它任务是自由的。仅当定时器溢出中断产生时，UPI 才处理简短的多重反复显示程序。一个完整的中断驱动的后台任务能涉及到多重反复显示，多重反复显示所消耗的时间值等于在UPI 前台中处理一个非定时器任务的充足时间(我们将讨论这一简短的定时)。

当定时器中断产生时，UPI 通过打开译码器断开所有数字，从内部数据存贮器取出下一个数字段的内容，并通过端口1(PORT1)输出到段驱动器中。最后，将下个数字的单元放到端口 2(PORT2 P20-P22)并打开译码器，显示数字的信息直到下一个中断。然后为了下个间隔重新启动定时器，在序列中每个数字连续反复处理。

作为一个讨论 UPI 软件的序曲，让我们检查在这一应用中仅用的内部数据存贮器结构，见图14。这一应用仅需要全部64个数据存贮器单元中的14个，将顶端的 8 个单元奉献给显示映象，每一个数字对应一个单元。这些单元包含关于每个字符的段和小数点信息，仅仅是这样将字符装入进暂时屏蔽的存贮器的一部份中，利用寄存器组1的寄存器 R7作为在中断服务程序期间的暂时累加器存贮。寄存器 R_3 存放下一个要显示数字的数值，R_2 的输入程序期间作为字符的暂存贮器，R_0是一个面向显示图的下一个数字单元的分支指示器。到目前为止，获得了12个单元，剩下的两个单元是两个要求在定时器和输入中断服务程序期间存放返回地址及状态的堆栈单元。剩下的没有使用的单元：所有的寄存器组 0，堆栈的14个字节，寄存器组 1 中的4个和 24 个 RAM 单元，对于由任何前台处理任务的使用都是可用的。

UPI软件仅由三个简短的程序组成，一个INIT严格地用在初始化期间，DISPLA 是一个在定时器中断时调用的多重反复程序，INPUT 是一个 IBF 中断时调用的字符输入处理