



[12] 发明专利申请公开说明书

[21] 申请号 95116819.3

[43]公开日 1997年1月15日

[11] 公开号 CN 1140272A

[22]申请日 95.8.31

[30]优先权

[32]94.9.7 [33]US[31]301943

[71]申请人 国际商业机器公司

地址 美国纽约州

[72]发明人 D·T·克朗普

S·T·彭科斯特

[74]专利代理机构 中国专利代理(香港)有限公司

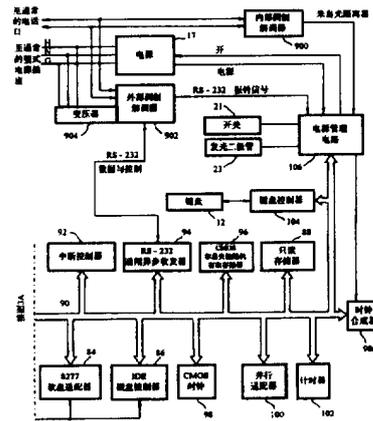
代理人 杜有文 马铁良

权利要求书 3 页 说明书 109 页 附图页数 52 页

[54]发明名称 用于高级电源管理(APM)的自动备份系统

[57]摘要

一种挂起/继续计算机系统，它带有相互作用电路通连的 CPU、非易失性存储器、易失性寄存器和内存数据、电源管理处理器、备份挂起计时器以及电源。电源管理处理器调节电源供给 CPU 的电力。一具有电源管理控制权的操作系统控制着上述挂起/继续系统。备份挂起计时器以独立于操作系统电源管理部分的方式运行。如果操作系统的电源管理部分停止起作用并且系统应被挂起，则备份挂起计时器会使系统挂起。



权利要求书

1. 一种计算机系统, 它包括:

(a) 可执行操作系统代码的一个 CPU, 其特征不在于控制着电源管理状态的转换、应用程序代码以及实现电源管理状态转换的 BIOS 代码;

(b) 电源管理电路, 它与 CPU 作电路通连并用于响应一组预定的挂起事件中的一个而有选择地在正常操作状态与挂起状态之间改变计算机系统的状态;

(c) 一备份挂起计时器, 它与 CPU 作电路通连并以独立于操作系统中用于控制电源管理状态转换的那部分代码的方式执行;

(d) 一电源, 它与上述 CPU 及电源管理电路作电路通连并包括用于响应电源管理电路而有选择地将系统电力从外部电源提供给计算机系统的电路, 此电源的特征是: 它具有第一供电状态和第二供电状态并带有将辅助电力提供给所述电源管理电路的电路;

其中, 所述第一供电状态的特征是: 所述电源从所述外部电源将系统电力提供给所述计算机系统、将辅助电力提供给所述电源管理电路; 以及

所述第二供电状态的特征是: 所述电源不从所述外部电源将系统电力提供给所述计算机系统, 但从所述外部电源将辅助电力提供给所述电源管理电路;

(e) 一非易失性存储器, 它与所述 CPU 作电路通连;

(f) 易失性系统内存, 它与所述 CPU 作电路通连并用于存储

内存数据; 以及

(g) 易失性系统寄存器, 它与所述 CPU 作电路通连并用于存储寄存器数据;

其中:

(1) 所述正常操作状态的特征是: 所述电源处于所述第一供电状态, 并且, 所述计算机系统可以响应用户命令、所述操作系统或所述 BIOS 而执行应用程序;

(2) 所述挂起状态的特征是: 寄存器数据和内存数据均存储在所述非易失性存储器内, 并且, 所述电源处于所述第二供电状态;

(3) 所述正常操作状态与所述挂起状态之间的所述转换包括: 所述 CPU 响应所述预定的挂起事件在所述系统内存、系统寄存器与所述非易失性存储器之间拷贝内存数据和寄存器数据;

(4) 所述正常操作状态与所述挂起状态之间的所述转换还包括, 所述控制部件响应所述预定的挂起事件而使所述电源分别在所述第一供电状态与所述第二供电状态之间转换; 以及

(5) 所述预定事件中的至少一个包括所述备份挂起计时器的到时。

2. 如权利要求 1 所述的计算机系统, 其特征在于, 它还包括一不活动挂起计时器。此计时器设置成在预定时间间隔之后到时, 并且, 所述预定挂定事件中的另一个包括不活动挂起计时器的到时。

3. 如权利要求 1 所述的计算机系统, 其特征在于, 包括一与电源管理处理器作电路通连的瞬时按钮开关, 所述开关响应其被

按下而产生连通事件, 并且, 所述预定挂起事件中的另一个包括所述开关的接通事件。

4. 如权利要求 1 所述的计算机系统, 其特征在于, 所述电源管理处理器提供了所述备份挂起计时器。

5. 如权利要求 1 所述的计算机系统, 其特征在于, 它还包括一瞬时按钮开关, 它与所述电源管理处理器作电路通连并响应按下所述开关而产生接通事件, 并且, 所述预定挂起事件中的另外一个包括所述备份挂起计时器的到时与所述开关的接通事件。

6. 如权利要求 1 所述的计算机系统, 其特征在于, 所述电源管理处理器包括一预编程的微控制器。

7. 如权利要求 1 所述的计算机系统, 其特征在于, 可响应执行中的操作系统的电源管理部分而重新启动或复位所述备份挂起计时器。

8. 如权利要求 4 所述的计算机系统, 其特征在于, 所述电源管理处理器包括一预编程的微控制器。

9. 如权利要求 1 所述的计算机系统, 其特征在于, 所述非易失性存储器包括一固定盘存储器。

说明书

用于高级电源管理 (APM) 的自动备份系统

本发明涉及到下列未决的申请:

申请第 08/097334 号, 1993 年 7 月 23 日提出, 题为“带有单一开关挂起/继续功能的台式计算机”(委托案第 BC9-93-018 (21322/00158) 号);

申请第 08/097250 号, 1993 年 7 月 26 日提出, 题为“具有零电压系统挂起的台式计算机系统”(委托案第 BC9-93-016 (21322/00161) 号);

申请第 08/097246 号, 1993 年 7 月 23 日提出, 题为“保存和恢复在受保护方式下执行代码的 CPU 的状态的方法”(委托案第 BC9-93-017 (21322/00162) 号);

申请第 08/097251 号, 1993 年 7 月 26 日提出, 题为“具有多级电源管理的台式计算机系统”(委托案第 BC9-93-015 号 (21322/00163) 号);

申请第 08/303102 号, 1994 年 9 月 7 日提出, 题为“挂起系统中电源故障的自动排除”(委托案第 BC9-94-043 (21322-00197) 号);

申请第 08/302148 号, 1994 年 9 月 7 日提出, 题为“挂起文件的自动分配”(委托案第 BC9-94-044 (21322/00198) 号);

申请第 08/302147 号, 1994 年 9 月 7 日提出, 题为“用于挂起系统的多功能电源开关和反馈 LED”(委托案第 BC 9-94-108

(21322 - 00202))

申请第 08/302157 号, 1994 年 9 月 7 日提出, 题为“用于计算机系统唤醒的低功率振铃检测” (委托案第 BC9 - 94 - 110 (21322 - 002404) 号);

申请第 08/301464 号, 1994 年 9 月 7 日提出, 题为“断电时用系统管理中断执行系统任务” (委托案第 BC9 - 94 - 112 (21322 - 00206) 号);

申请第 08/302066 号, 1994 年 9 月 7 日提出, 题为“掉电之后用户选择的自动恢复” (委托案第 BC9 - 94 - 113 (21322 - 00207) 号);

申请第 08/303103 号, 1994 年 9 月 7 日提出, 题为“用于防止数据丢失的待机检查点” (委托案第 BC9 - 94 - 114 (21322 - 00208) 号);

本发明在总体上涉及计算机系统的结构, 具体地说, 涉及到一种具有系统挂起/继续能力的台式计算机系统, 该系统带有一多功能开关和受 BIOS 支持的发光二极管 (LED), 所说的开关能进行状态转换, 从而使系统更便于使用。

个人计算机系统在本技术中是周知的。一般的个人计算机系统特别是 IBM 个人计算机系统业已有广泛的应用, 从而为当今社会的许多方面提供了计算机的威力。个人计算机一般界定为台式微机、落地式微机或便携式微机, 它包括: 一系统组件, 此组件带有一个单一的中央处理单元 (CPU) 以及包括所有 RAM 和 BIOS ROM 在内的相关易失性和非易失性存储器, 一系统监视器、一键盘、一个或多个软盘驱动器、一固定的磁盘存储器 (也称为“硬

盘驱动器”)、一称为“鼠标器”的指点设备以及一可选择的打印机。这类系统的显著特点之一是使用母板或系统主板以便将上述组件以电气的方式连接在一起。这类系统最初设计成给单用户以独立的计算能力并且具有不贵的价格,以便个人或小型公司购买。这类个人计算机系统的实例是 IBM 中 PERSONAL COMPUTER AT 和 IBM 的 PERSONAL SYSTEM/1 (IBM PS /1)。

个人计算机通常用于运行软件以执行各种活动,如字处理、通过电子数据表格进行数据处理、在数据库中收集数据并建立数据之间的联系、显示图形、用系统设计软件设计电子或机械系统等等。

上述申请书中的前四个相关的申请公开了一种具有四种电源管理状态的计算机系统,这四种状态是:正常操作状态、待机状态、挂起状态和关机状态。用一个开关在关机状态、正常操作状态以及挂起状态之间进行状态改变。

本发明之计算机系统的正常操作状态实际上与一般台式计算机的正常操作状态相同。用户可以使用应用程序并基本上同其它计算机一样对待这种计算机。一个不同就是存在有电源管理驱动程序,该程序在后台运行(在 BIOS 和操作系统中运行)并且对用户是透明的。该电源管理驱动程序在操作系统(OS)中的那部分是由 Intel 和 Microsoft 公司所写的高级电源管理 (APM) 的新式程序设计接口,这种接口存在于大多数在 Intel 80x 86 系列处理器上运行的操作系统中。上述电源管理驱动程序在 BIOS (APM BIOS) 中的那部分与 APM OS 驱动程序互通。APM OS 驱动程

序和 APM BIOS 例程一道控制着计算机向上述其它三种状态的转换以及来自这三种状态的转换。

第二种状态即待机状态，它比正常操作状态使用更少的电量，但仍使任何应用程序象它们在其它情况下运行一样地运行。一般地说，在待机状态下，通过使各设备处于相应的低功耗状态而节省了电能。例如，在待机状态下通过使磁盘驱动器内的固定式磁盘停止旋转以及通过停止生成视频信号，可以节省电能。

第三种状态是挂起状态。在挂起状态下，计算机系统消耗极少量的电能。挂起的计算机消耗很少的来自壁装电源插座的电量。仅有的功耗是用于维持监控计算机系统内部电池的开关的电路的少量功耗（当计算机系统不接收交流电时）或者是电源在辅助电源线上所产生的少量功耗（当计算机系统接收交流电时）。

通过在电源关闭之前将计算机的状态保存到固定磁盘存储器（硬盘驱动器）中，可以实现少量的用电。为了进入挂起状态，计算机系统中断任何正在执行的代码并把对计算机的控制权交给电源管理驱动程序。该电源管理驱动程序确定计算机系统的状态并将该状态写至固定磁盘存储器。CPU 寄存器、CPU 高速缓冲存储器、系统内存、系统高速缓冲存储器、视频寄存器、视频内存以及其它设备寄存器等的状态全都写至固定磁盘。将系统的整个状态以这样的方式保存起来：在中断不对代码应用程序产生不利影响的情况下将计算机系统的状态保存起来。然后，计算机将数据写入非易失性 CMOS 存储器中以指示系统已挂起。最后，计算机使电源停止供电。计算机的整个状态可靠地保存在固定磁盘存储器中，系统功耗当前处于“停止”，这时，计算机只从电源中接收少

量的稳压电力以供应监控开关的电路。

第四种也是最后一种状态是关机状态。在这种状态下，电源停止向计算机系统提供稳定的电能，计算机系统的状态也不保存在固定磁盘内。这种关机状态实际上等同于按通常的方式关闭一般的台式计算机。

从一种状态到另一种状态的转换是由电源管理驱动程序控制的并且通常是以一个开关、一个标志以及两个计时器即不活动性待机计时器和不活动性挂起计时器的关闭事件为基础的。所述计算机系统带有一个单一的电源开关，此开关用于开启计算机系统，挂起计算机系统的状态、恢复计算机系统的状态以及关闭计算机系统。

上述系统使用了高级电源管理 (APM) 以便于状态转换以及与该状态转换相关的系统维护。APM 是一种工业标准的高级程序编制接口，它的开发是为台式和笔记本式计算机系统内的操作系统和系统 BIOS 提供一种共同管理电源的手段。APM 提供了一种优秀的工具，这种工具因涉及挂起进程中的操作系统而便于使计算机系统挂起和继续并且能使操作系统在挂起进程开始之前准备好所说的系统。

但是，取决操作系统，依靠 APM 来挂起系统会有多种缺陷。在 Microsoft Windows 3.1 和 MS-DOS 中，操作系统的 APM 驱动程序有时常会完全停止对 APM BIOS 的请求进行服务。例如，在显示模态对话框时，Microsoft Windows 的 APM 驱动程序会完全停止对 APM BIOS 的请求进行服务。因此，在显示模态对话框时，基于 Microsoft Windows APM 驱动程序的挂起和继续系统

将不会使计算机系统挂起。这对用户来说是一个很大的问题，因为，计算机系统不会响应用户保存系统状态的请求而挂起。这种结果会把用户搞糊涂并且使用户因挂起和继续系统不可靠而不再信任该系统。

一种解决办法是靠硬件的系统管理中断 (SMI) 来使计算机系统挂起，但这种系统会失去 APM 所提供的优点。例如，APM 驱动程序可以通知所有 APM 可识别的应用软件和硬件计算机系统快要被挂起了，从而这些软件和硬件能作相应的反应。所以，最好具有这样一种系统，它能使计算机系统依靠 APM 驱动程序，但在 APM 驱动程序停止工作时仍然能使计算机系统挂起。

依照本发明，所说的计算机系统配备有一个替代的挂起系统，这种系统能自动地检测 APM 驱动器在何时停止起作用并因此而使计算机系统挂起。本发明的自动备份挂起系统不会影响 OS APM 驱动程序的正常运行，但它在 APM 驱动程序停止对 APM BIOS 的请求进行服务时会自动地接入。当 APM 再度变为活动的时，该备份挂起系统会自动解除。这就会使计算机系统能利用带有上述备份挂起系统所提供的高可靠性的 APM 的优点。

在本发明的备份挂起系统中，使用了一种新型的递减计时器，如果 OS APM 在一段预定的时间内不对 APM BIOS 的请求进行服务，则该计时器就会到时。这种计时器是一种按时间单位如秒来递减的自激式计时器，它可被重新启动，即可被重新加载回原始值。在最佳实施例中，该新型计时器维护在外部硬件内并由诸如系统 BIOS 计时器零中断处理程序或 SMI 中断服务例程之类中断服务例程来服务。另外，也可以由上述中断服务例程单独

来启动并服务该计时器。

最佳的是，由系统 BIOS 计时器零中断处理程序来服务于上述新型计时器，上述处理程序作了改进以便检查是否该新型计时器已到时且用户想要进行挂起（即挂起请求是未决的）。当满足上述条件时，计算机系统在不通知操作系统未将挂起放置到 APM BIOS 事件队列内的情况下挂起。当计算机系统继续时，将一临界的继续事件放置到 APM 事件队列内。当运行 APM 的驱动程序再度开始轮询服务事件时，它会接收到上述临界的继续事件，因此，操作系统可在继续工作之后清除并更新系统时间等。

最佳的是，在 APM 系统 BIOS 中，在来自 OS APM 驱动程序的每次“get event”调用时，复位所述新型递减计时器，即将其置成起始值。这就能阻止该递减计时器在 APM 正对事件进行服务并正常运行时到时。当操作系统的 APM 驱动程序停止对事件进行服务（没有 get event 调用）时，递减计时器不会被复位并将到时。当用户想要挂起时，计算机系统会在递减计时达到零时自动地加以响应。

最佳的是，递减计时器设置在一低成本的预先编程的微控制器内，该微控制器还为计算机系统提供其它的特征。上述微控制器的特征之一就是通用的递减计时器，此计时器用于提供所说的自动备份挂起系统。

每当约 32 个计时器零级中断，系统 BIOS 中断处理程序就会检查是否递减计时器业已到时且用户想要进行挂起。当出现这种情况时，计算机系统被立即挂起。最佳的是，将递减计时器设置为 16 秒以供 APM 内的“死区”（dead zone）如（MS - Windows 下的

全屏 MS - DOS 等)使用,在所说的“死区”内,APM 的“get event” (取事件)轮询率为每 15 秒钟一次。

本发明克服了操作系统内高级电源管理 (APM) 驱动程序的缺陷。当 OS APM 驱动程序不能对 BIOS 事件进行服务时,递减计时器立即开始递减计时。当操作系统的 APM 驱动程序停止对获得事件 (get event) 进行服务时,所说的系统就会自动地接入。同样,当 APM 驱动程序又对事件进行服务时,所说的系统会自动地解除。本发明提高了挂起和继续系统的可靠性。所述备份挂起系统能自动地覆盖已知的死区。该备份挂起系统是全自动的并且无须用户干预。本发明的上述及其它优点将会从对本发明的详细说明中更明显地表现出来。

本发明的上述及其它优点会从对本发明的详细说明中更明显地表现出来。

包括在本说明书中并构成本说明书一部分的附图,连同以上对本发明的总体描述,说明了本发明的实施例,并且,以下给出的详细说明用于例证本发明的原理。

图 1 是体现本发明的个人计算机的透视图;

图 2 是图 1 中个人计算机某些部件的分解透视图,它包括一机箱、一罩盖、一机电直接存取存储器以及一主板并说明了这些部件之间的特定关系;

图 3A 和图 3B 显示了图 1 和图 2 中个人计算机某些组件的框图;

图 4 为本发明之计算机系统的状态图,它显示四种系统状态:正常状态、待机状态、挂起状态和关机状态;

图 5 是显示电源有关部分的框图;

图 6A 为本发明之电源管理电路的概略电路图, 它显示了与其它图各个接口;

图 6B 是使上述电源管理电路与内部调制解调器相连的概略电路图;

图 6C 是显示用于上述电源管理电路的复位电路中各个信号的波形图;

图 6D 是电源故障检修电路第二实施例的概略电路图;

图 7 是由本发明之电源管理处理器所维持的开关状态之一的状态图;

图 8 是总体上显示本发明加电例程的流程图;

图 9A 是管理例程的详细流程图, 操作系统中的 APM 设备驱动器大约每一秒钟调用该管理例程;

图 9B 是显示 APM 处理最近请求例程 (APM Working On Last Request Routine) 细节的流程图;

图 9C 是显示 APM 拒绝最近请示例程 (APM Reject ON Last Request Routine) 细节的流程图;

图 10 是显示本发明之挂起例程 (Suspend Routine) 细节的流程图;

图 11 是显示本发明之引导例程 (Boot - up Routine) 细节的流程图;

图 12 是显示本发明之继续例程 (Resume Routine) 细节的流程图;

图 13 是显示本发明之保存 CPU 状态例程 (Save CPU State

Routine) 细节的流程图;

图 14 是显示本发明之恢复 CPU 状态例程 (Restore CPU State Routine) 细节的流程图;

图 15 是显示本发明之保存 8959 状态例程 (Save 8959 State Routine) 细节的流程图;

图 16 是显示本发明之动态保存文件分配例程 (Dynamic Save File Allocation) 细节的流程图;

图 17 是显示本发明的退出待机状态例程 (Exit Standby Routine) 细节的流程图;

图 18 是显示本发明之进入待机状态例程 (Enter Standby Routine) 细节的流程图; 以及

图 19 是显示本发明之电源管理处理器例程 (Power Management Processor Routines) 细节的流程图。

在以下参照显示本发明之最佳实施例的附图详细说明本发明时, 从以下说明的一开始就应该认识到, 本技术的专家可以改进本文所述的发明, 但仍能获得本发明的良好结果。因此, 应该将以下的说明理解为是以本技术的专家为对象的广义的教导性公开, 并且不限制本发明。本发明涉及到计算机系统的整体设计, 包括但不限于计算机体系结构设计、数字设计、BIOS设计、保护模式的 80486 代码设计、应用程序代码设计、操作系统代码设计以及高级电源管理的新式程序设计界面的使用。

参照附图, 显示了体现本发明的微机系统并总体上用标号 10 表示(图 1)。如上所述, 计算机 10 带有相连的显示器 11、键盘 12、鼠标器 13 以及打印机或绘图仪 14。计算机 10 带有: 一罩盖 15, 此

罩盖由装饰性外部部件 16(图2)以及一内部保护部件 18 构成,此部件与机箱 19 相连从而确定了一封密有屏蔽的空间以便接收加电的数据处理和存储组件,这些组件用于处理和存储数字数据。这些组件中的至少某一些安装在多层主板 20 或母板上,而主板则安装在机箱 19 上并提供了使计算机 10 的组件相互电连接的装置,所说的组件包括上述组件以及其它相关的部件如软盘驱动器,各种形式的直接存取存储器、附件的适配器卡或适配器板以及类似的部件等。正如以后将要详细说明的那样,在主板 20 上设置了朝向或来自微机操作组件的输入/输出信号的通路。

所说的计算机系统带有:一电源 17;一电源按钮 21,下文中称开关 21;以及一电源/反馈 LED23。与一般系统内的通常电源开关不同,电源按钮 21 并不将交流电切换至电源 17 或从该电源 17 切换,这将在以下予以说明。机箱 19 带有:标号 22 所示的底座;标号 24 所示的前部面板;以及标号 25 所示的后部面板(图 2)。前部面板 24 限定了至少一个开放的凹槽(在所示的图中有四个凹槽),以便接收诸如磁盘或光盘驱动器、后备磁带机之类的数据存储装置或类似的设备。在所示的图中设置了一对上部凹槽 26、28 以及一对下部凹槽 29、30。上部凹槽中的一个 26 用于接收第一尺寸的外部设备(如 3.5 英寸的驱动器),另一个 28 则用接收两种尺寸(如 3.5 和 5.25 英寸)中选定的一种驱动器,下部凹槽用于接收一种尺寸(3.5 英寸)的驱动器。图 1 中标号 27 表示一软盘驱动器,此驱动器是一可装卸式直接存取存储器,这种存储器可以接收插进来的软盘并如众所周知的那样用软盘接收、存储和传送数据。标号 31 表示一硬盘驱动器,此驱动器是一固定式直

接存取存储器，这种存储器可以如众所周知的那样存储和传递数据。

在说明上述结构与本发明的关系之前，有必要概述一下个人计算机系统10的通常操作。参照图3A和图3B，显示了个人计算机系统的框图，该框图说明了诸如本发明之系统10之类的计算机系统的各种组件，包括安装在主板20上的组件以及主板与I/O槽和个人计算机系统其它硬件的连接件。与主板相连的是系统处理器40，也即CPU40，它包括一微处理器，该微处理器通过高速CPU局部总线42与内存控制部件46相连，而内存控制部件则又与易失性随机存取存储器(RAM)53相连。内存控制部件46包括内存控制器48、地址多路复用器50以及数据缓冲区52。内存控制部件46还与由四个RAM组件54所表示的随机存取存储器53相连。内存控制器48包括逻辑线路，此逻辑线路用于把去往和来自微处理器40的地址映射到RAM53的特定区域。所说的逻辑线路用于回收先前被BIOS占用的RAM。此外，内存控制器48还生成ROM选择信号(ROMSEL)，此信号用于启动或中止ROM88。虽然任何适当的微处理器都可用作系统处理器40，一种合适的微处理器是INTEL公司出售的80486。Intel 80486带有内部高速缓冲存储器，所以是Intel 80486的任何CPU40都带有CPU高速缓冲存储器41。

虽然在下文中是参照图3A和图3B的系统框图来说明本发明的，但是，在以下说明的一开始就应理解，本发明的设备与方法可和主板的其它硬件配置一道使用。例如，系统处理器40可以是Intel 80286或80386微处理器。正如本文所使用的那样，凡涉及

到 80286、80386 或 80486 一般均指可从 Intel 公司获得的微处理器。但是,最近也有其它生产者业已开发出了能执行 Intel X86 结构的指令集的微处理器,因而所述术语的用法包括了任何能执行该指令集的微处理器。正如本技术的专家所熟知的那样,早期的个人计算机一般把那时流行的 Intel 8088 或 8086 微处理器用作系统处理器。这种处理器具有一兆字节内存的寻址能力。最近,个人计算机一般都使用高速的 Intel 80286、80386 以及 80486 微处理器,这些微处理器能以虚模式或实模式运行来仿真较低速度的 8086 微处理器,或者以保护方式运行,从而对某些机型来说可将寻址范围从一兆字节扩大至四千兆字节。从本质上说,80286、80386 以及 80486 处理器的实模式的特征使得硬件与为 8086 和 8088 微处理器所写的软件相兼容。所述 Intel 系统的微处理器通常只用涉及到整个类型符后三位数字的三位数来表示,如“486”等。

以下回到图 3A 和 3B, CPU 局部总线 42 (包括未显示的数据,寻址和控制组成部分)将微处理器 40、数字协处理器 44 (如果不在 CPU 40 内的话)、视频控制器 56、系统高速缓冲存储器 60 以及高速缓冲存储器控制器 62 连接起来。视频控制器 56 带有与之相联的监视器 (或视频显示终端) 11 以及视频存储器 58。缓冲器 64 也连在 CPU 局部总线 42 上。缓冲器 64 又与 (和 CPU 局部总线 42 相比) 速度较低的系统总线 66 相连,系统总线也包括地址、数据和控制组成部分并且在缓冲器 64 与另一缓冲器 68 之间延伸。系统总线 66 还与总线控制及定时部件 70 和 DMA (直接存储器存取) 部件 71。相连 DMA 部件 71 包括中央仲裁器 82 以及

DMA 控制器 72。辅助缓冲器 74 提供了系统总线 66 与诸如工业标准结构 (ISA) 总线之类的可选的特性总线 76 之间的接口。总线 76 与多个用于接收 ISA 适配器卡 (未显示) 的 I/O 槽 78 相连。ISA 适配器卡以插接的方式与 I/O 槽 78 相连并为系统 10 提供了附加的 I/O 设备或存储器。

仲裁控制总线 80 将 DMA 控制器 72 及中央仲裁器 82 与 I/O 槽 78、磁盘适配器 84 及集成驱动器电子线路 (Integrated Drive Electronics - IDE) 的固定磁盘控制器 86 相连。

虽然显示出微机系统 10 带有基本的四兆字节 RAM 组件 53, 但应该注意, 也可以通过增加可选的高密度存储器组件 54 而如图 3A 和图 3B 所示那样连上附加存储器。仅从说明的角度出发, 参照基本的四兆字节存储器组件来说明本发明。

锁存缓冲器 68 连接在系统总线 66 和主板 I/O 总线 90 之间。主板 I/O 总线 90 包括地址、数据和控制组成部分。沿着主板 I/O 总线 90 连接有多个 I/O 适配器与其它组件, 如磁盘适配器 84, IDE 盘适配器 86, 中断控制器 92, RS - 232 适配器、本文称之为 NVRAM 的非易失性 CMOS RAM96, CMOS 实时时钟 (RTC) 98、并行适配器 100、多个计时器 102、只读存储器 (ROM) 88、8042 104 以及电源管理电路 106。标号 104 所示的 8042 是与键盘 12 与鼠标器 13 相连的从属微处理器。电源管理电路 106 在电路上与电源 17, 电源开关 21、电源/反馈 LED23 以及内部调制解调器 900 和/或外部调制解调器 902 相连。正如本技术的专家所熟知的那样, 外部调制解调器通常与变压器 904 相连, 而该变压器又与壁装电源插座相连。调制解调器 900、902 与通常的电话口

相连。电源管理电路 106 显示在图 6A 及图 6B 中并在与图 6A、图 6B、图 6C 以及图 7 有关的内容中予以更全面的说明。只读存储器 88 包括 BIOS, BIOS 用于连接 I/O 设备与微处理器 40 的操作系统。存储在 ROM88 中的 BIOS 可拷贝到 RAM53 以便减少 BIOS 的执行时间。ROM88 还可响应内存控制器 48 (通过 ROMSEL 信号)。如果用内存控制器 48 启动 ROM88, BIOS 就会从 ROM 中执行。如果内存控制器 48 终止了 ROM, 则 ROM 不响应来自微处理器 40 的地址查询(即 BIOS 从 RAM 中运行)。

实时时钟 98 用于计算日期的时间, NVRAM96 用于存储系统配置数据。也就是说, NVRAM96 包含说明系统当前配置的值。例如, NVRAM96 包括说明固定盘或软盘容量、显示器类型、内存量、时间、日期等的信息。此外, 每当运行诸如 SET 配置之类的特定配置程序时, 上述数据都存储在 NVRAM 中。上述 SET 配置程序的目的是把描述系统配置的值存储到 NVRAM 内。

几乎所有的上述设备都包括易失性寄存器。为了避免对附图作不必要的分类, 特定设备的寄存器总是涉及到该设备。例如, CPU 寄存器称为 CPU40 的寄存器, 视频控制器寄存器称为视频控制器 56 的寄存器。

如上所述, 所说的计算机带有一总体用标号 15 表示的罩盖, 此罩盖与机箱 19 相配合形成一封密有屏蔽的空间, 该空间用于包容微机的上述组件。罩盖 15 最好形成有外部装饰性罩盖部件 16 以及薄金属板衬里 18, 所说的外部装饰性罩盖部件是一成整体的模制组件, 由可模制的合成材料构成, 而薄金属板衬里则呈符合上述装饰性罩盖部件结构的形状。但是, 也可以以其它周知

的方式形成前述罩盖，本发明的实用性并不局限于上述类型的外罩。

操作的状态

参照图 4, 显示了本发明之计算机系统的状态图。本发明的计算机系统 10 具有四种状态: 正常操作状态 150、待机状态 152、挂起状态 154 以及关机状态 156。图 4 所示的状态之间的转换用于说明最佳实施例, 但不限定最佳实施例。因此, 也可以用其它的事件来导致状态转换。

本发明之计算机系统 10 的正常操作状态 150 实际上等同于任何通常的台式计算机的正常操作状态。用户可以使用应用程序并且基本上把上述计算机看作是任何其它的计算机。一个不同点是在操作系统中有在后台运行的电源管理驱动程序 (“APM OS 驱动程序”) 以及多种 APM BIOS 例程, 它们对用户是透明的。APM BIOS 例程将在以后说明, 它们中包括挂起例程、继续例程、引导例程、管理例程、保存 CPU 状态例程以及恢复 CPU 状态例程。所有附图中都未示出的一个 APM BIOS 例程是 APM BIOS 路由选择例程。APM BIOS 路由选择例程主要从 APM OS 驱动程序中接收指令并调用适当的 APM BIOS 例程。例如, 在 APM OS 驱动程序发出 Suspend (挂起) 指令时, APM BIOS 程序选择例程就会调用挂起例程。作为另外一个实例, 每当 APM OS 驱动程序发出 Get Event (获取事件) 指令时, APM BIOS 路由选择例程都会调用管理例程。上述例程处于 BIOS 内并在投影 BIOS 时被投影掉。OS 内的电源管理驱动程序以及 APM BIOS 例程控制着计算机在四种状态之间的转换。虽然上下文可能有所限定, 但

对“APM”一词的引用通常就是指 APM OS 驱动程序。

第二种状态即待机状态 152 要比正常操作状态 150 使用更少的电能，但能使应用程序象在其它情况下运行的那样运行。一般地说，在待机状态 152 下可通过代码将设备设置成各自的低功耗方式而节电。在所述的最佳实施例中，正如以下将会详细说明的那样待机状态 152 通过使固定磁盘存储器 31 内的固定磁盘（未显示）停止转动、通过停止生成视频信号以及通过将 CPU 40 设置成低功耗方式，从而节约了电能。但是，这并不是是一种限制，而是可以使用其它方法如减慢或停止 CPU 时钟等来降低功耗。

在所说的最佳实施例中，以三种独立的方式节约电能。第一，在正常操作状态 150 下，固定磁盘存储器 31 内的固定盘以例如 3600 转/分 (RPM)、4500 转/分 (RPM) 或 5400 转/分 (RPM) 的转速不断旋转。在待机状态 152 下，给 IDE 盘控制器 86 以指令从而使固定磁盘存储器 31 进入低功耗方式（固定磁盘存储器 31 内的固定磁盘停止转动），因此节省了固定磁盘存储器 31 内的电机（未显示）在使固定磁盘旋转时所消耗的电能。

第二，在正常操作状态 150 下，计算机系统的视频控制器 56 不断地产生对应于在视频显示终端 11 所看到的图象的视频信号（诸如本技术中所周知的 HSYNC、VSYNC、R、G、B 等等）。在待机状态 152 下，视频控制器 56 停止产生视频信号，从而节约了通常被视频控制器 56 所消耗掉的电能，HSYNC、VSYNC、R、G、B 等信号均近似为 0.00VDC。使用 VESA（视频电子线路标准协会）监视器会进一步节电，这是因为，当 HSYNC 和 VSYNC 近似为 0.00VDC 时，VESA 监视器会关闭自身。

第三,在正常操作状态 150 下,CPU40 不断地执行指令,因而消耗了电能。在待机状态 152 下,BIOS 响应 AMP CPU 的空调用 (Idle Call) 而发出 HALT (停机) 指令。执行 HALT 指令会明显地降低 CPU 的能耗直至发生下一个硬件中断。CPU 确实空闲时会 在 90% 以上的时间内停止运转。

请注意,某些系统带有“屏幕保护程序”(Screen - savers),这种程序能使屏幕 11 变暗以防止视频显示终端前表面荧光物质老化。在大多数这种系统中,视频控制器 56 仍产生视频信号,只是产生对应于黑暗屏幕或动态显示的视频信号。因此,执行屏幕保护程序的计算机系统仍消耗电能,这些电能是产生上述视频信号所必需的。

第三种状态是挂起状态 154。在挂起状态 154 下,计算机系统消耗非常少的电能。在所述的最佳实施例中,挂起的计算机消耗少于 100 毫瓦的电能。仅有的功耗是由电源 17 的低效率以及电源管理电路 106 使用的少量电能所导致的约为 5 瓦的功耗。

通过在关闭电源之前将计算机系统的状态保存至固定磁盘存储器 (硬盘驱动器) 31,可以实现上述少量的用电。为了进入挂起状态 154,CPU40 中断任何的应用程序并将 CPU 的程序执行控制权转交给电源管理驱动程序。电源管理驱动程序确认计算机系统 10 的状态并将该计算机系统的全部状态写至固定磁盘存储器 31。将 CPU40 的寄存器、CPU 高速缓冲存储器 41、系统 RAM53、系统高速缓冲存储器 60、视频控制器 56 的寄存器、显示器内存 56 以及其它易失性寄存器的状态均写入固定磁盘驱动器 31。以这样的方式保存系统 10 的整个状态,即:可在不明显损失

有效性的情况下恢复该系统的状态。也就是说，用户不必等待系统象通常那样加载操作系统、加载图形用户界面以及应用程序。

然后，所述计算机将数据写入非易失性 CMOS 存储器 96 以指示系统被挂起。最后 CPU40 指令微处理器 40 使电源 17 停止经由 $\pm 5\text{VDC}$ 和 $\pm 12\text{VDC}$ 线向计算机系统提供稳压的电力。这时计算机系统 10 处于降低功耗的状态，而且，该计算机的整个状态被安全地存储到了固定盘存储器 31 上了。

在本文中以两种类似但可能混淆的方式使用“状态”一词。设备可以“处于”特定的状态。四种系统状态—正常 150、待机 152、挂起 154 以及关机 156—均涉及到本发明之计算机系统 10 的整体状态。这些“状态”以通常的方式描述了计算机系统 10。例如，在处于正常操作状态 150 时，CPU40 仍在执行代码并能改变系统 10 内的多个寄存器。同样，在处于待机状态 152 时，会产生类似的活动。因此，当计算机系统 10 处于正常操作状态 150 和待机状态 152 时，系统 10 的内存与寄存器的配置是动态的。

其它设备也可以“处于”某种状态。电源管理电路 106 最好使用一第二处理器作为电源管理处理器，如图 6A 中所示的微控制器 U_2 ，以便实现各种电源管理特征。多种这样的处理器都是适用的，在所述的特定实施例中，电源管理处理器是预先编程了的 83C750 微控制器。正如与图 6A 有关的内容所要说明的那样，微处理器 U_2 的变量和引线可处于好几种状态。

设备的“状态”如“计算机系统 10 的状态”或“CPU40 的状态”与以上内容相对比，设备的“状态”涉及到该设备在特定计算机工作周期内的状况。所有的内存存储单元和寄存器均具有特定的二

进制值。一个设备的“状态”是该设备内容的静态二进制快照。

计算机系统 10 的“状态”是指操作性等价值但不一定是精确的拷贝。例如, 状态 A 下的计算机系统可能在 CPU 高速缓冲存储器 41 或系统高速缓冲存储器 60 内有某些存储内容。可能会将各高速缓冲存储器的内容“倾泻”进系统 RAM53 并将计算机系统设置成状态 B。单纯地说, 计算机在状态 A 下的状态不同于该计算机在状态 B 下的状态, 这是因为, 高速缓冲存储器的内容与系统 RAM 的内容是不同的。但是, 从软件操作性的观点来看, 状态 A 和状态 B 是相同的, 这是因为, 除系统速度略有降低(这是由程序不具有从高速缓冲存储器中执行的有利条件所引起的) 以外, 正在执行的程序不受影响。也就是说, 尽管高速缓冲存储器已被清空的计算机在效率方面略有降低直至用有用的代码重新加载该高速缓冲存储器的区域, 但状态 A 下的计算机与状态 B 下的计算机在软件操作性方面是等价的。

按两种类似但可能混淆的方式使用“Power”一词, “Power”大多数情况下是指电能。但“Power”也偶而指计算的功率。上下文将清楚地表明特定的用法。

“电路”通常指物理上的电子设备或多个相互电连接的设备, 但“电路”一词也包括物理电子设备的等效 CPU 代码。例如, 一方面, 可用 74 LS00 或等价地在可编程设备中实现两个输入的与非门。这两个设备是物理的电子设备。另一方面, 也可以通过使 CPU40 从两个 CPU 可读输入端口读出两个输入、利用 CPU 指令生成与非的结果并将该结果经由 CPU 可写输出端口输出, 从而实现与非门。上述 CPU 接口端口可以是诸如解码的锁存器之类的

简单设备，也可以是等效的可编程设备，或者是诸如外围接口适配器 (PIA) 之类的复杂设备，这些设备在本技术中都是众所周知的。“电路”一词有包括了上述与非门的全部三种实例的足够广泛的含义。“电路”在某些情况下可能仅指电通路。电通路的类型包括印刷电路板上的导线、线路或通道或者是构成一个电连接通路的几种类型电通路的任何组合。

“信号”涉及到单一的电学波形或多个波形。例如，视频控制器产生一视频信号。正如在本技术中周知的那样，上述视频信号实际上是多个电导体上的多个信号：HSYNC、VSYNC、R、G、B 等。

参照图 4，第四种也是最后一种状态是关机状态 156。关机状态 156 实际上等同于在通常意义上业已关闭的通常计算机系统。在这种状态下，电源 17 的初级/稳压部件 172 会停止向计算机系统 10 提供稳压后的电能 (除经由 AUX5 提供少量的电能以外，这将参照图 5 予以详细的说明)，但还未将计算机系统的状态保存至固定磁盘 31。挂起状态 154 和关机状态 156 在电源 17 不再产生稳压电能方面是相似的。它们的不同在于，在关机状态 156 下，不象在挂起状态 154 下那样将计算机系统 10 的状态保存至磁盘驱动器 31。而且，在离开关机状态 156 时，计算机 10 会象开机那样进行“引导”。也就是说，必须由用户或者自动地由诸如 AUTOEXEC. BAT 文件之类的方式启动任何可执行的代码。但是，在离开挂起状态 154 时，计算机 10 则会在它被中断的位置处恢复执行。

图 4 还显示了在四种状态之间引起转换的事件的概况。在参

照与图 6 至图 8 有关的内容中将对这些事件作进一步的说明。但是，作概略的说明可能是有帮助的。电源开关 21、两个计时器（不活动性待机计时器以及不活动性挂起计时器，见图 9 及其所附文字）、唤醒计时器的分钟数以及允许挂起标志（见与图 6A 和图 7 及其有关的内容）均会影响计算机进入何种状态。一般地说，上述两个计时器可以是硬件或者是在 CPU 中作为程序执行的 CPU 代码计时器。在所说的最佳实施例中，上述两个计时器均为从 BIOS 数据段开始执行的 CPU 代码计时器。但是，这两个计时器也可以是硬件计时器，这会是一个比较好的方案，因为这会减少所述系统的总开销。与图 9 有关的内容将更详细地说明上述两个计时器。在计算机 10 处于正常操作状态 150 或待机状态 152 时，上述两个计时器都是活动的。如下所述，计时器与其它例程相联，从而任何一个计时器的到时都会引起转换。可以根据用户的特定要求将一个或两个计时器配置成能在特定的时间之后到时。在所述的最佳实施例中，可将不活动性待机计时器及不活动性挂起计时器设置成在 10 至 90 分钟后到时。也可以使一个或两个计时器停止，即将计时器设置成不会到时。使计时器“停止”实际上是停止了该计时器的增量计数活动或者是忽略了它们的到时。在所述的最佳实施例中，将计时器到时值置为零值会使得不检验该计时器的到时。例如，联网计算机的用户可能不让计算机进入挂起状态 154，因为这样做会使局域网 (LAN) 相对该计算机失效。

从理论上说，计时器可递增计数或递减计数，而且在启动（或重新启动）计时器时，可被复位到预定的状态，该计时器可以期望计数到一种预定的状态，或者使用当前值以及作为边界终止触发

器所计算出的差值或总和，在所述的最佳实施例中，在复位计时器时，将来自实时时钟 98 的分钟数变量的当前值存储起来。通过从所保存的分钟数值中减去当前的分钟数并比较其差值与用户所选定的值，可以检验计时器的到时。

某些系统活动会影响上述两个计时器。例如，在所述的最佳实施例中，正如与图 9 有关的内容所要详细说明的那样，用户按下键盘 12 之按键的活动、移动鼠标器 13 的活动、按下鼠标器 13 之按钮的活动或者硬盘驱动器 31 的活动均会使各个计时器被重新启动；所以，当用户按下键盘 12 的按键或使用鼠标器 13 时，或者当应用程序存取硬盘驱动器 31 时，两个计时器均不会到时。此外，其它的系统活动也会复位所说的计时器。对活动来说，也可以监控任何硬件的中断。因此，最好能使打印 (IRQ5 或 IRQ7) 或通信 (COMM) 端口访问 (IRQ₂ 或 IRQ₃) 阻止系统进入挂起状态 154。

允许挂起标志是微控制器 U₂ 内的 CPU 可控且可读的锁存器，与图 6A 有关的内容将对该标志予以详细的说明。简要地说，将微控制器 U₂ 设置成一种方式会导致按下开关 21 就能将系统 10 置于关机状态 156，而将微控制器 U₂ 设置成另一种方式则会导致按下开关 21 就能将系统 10 置于挂起状态 154。在清掉了写至微控制器 U₂ 的允许挂起标志的情况下，如果计算机系统 10 处于正常操作状态 150 并按下了电源开关 21，那么，计算机系统 10 就会进入关机状态 156，如标号 158 所示。如果计算机系统 10 处于关闭状态 156 且按下电源开关 21，则计算机系统 10 会进入正常操作状态 150，如标号 160 所示。此外，以下将作详细说明的若干“外部事件”也会使计算机系统从关机状态 156 转换至正常操作

状态 150。

如果计算机系统 10 处于正常操作状态 150, 有一种事件可以使计算机进入待机状态, 即: 如果不活动性待机计时器到时, 则计算机系统 10 会转换至待机状态 152, 如标号 162 所示。另外, 所说的计算机系统也可以提供对话框、开关或其它输入设备之类的手段以使用户强迫系统立即进入待机状态。而在待机状态 152 下, 包括用户按下电源开关 21 在内的上述任何用户或系统的活动都会使计算机 10 离开待机状态 152 并重新进入正常操作状态 150, 如标号 164 所示。

按下电源开关 21 会使系统从待机状态 152 转换至正常操作状态 150 从而防止用户弄错。如上所述, 在处于待机状态时, 监视器 11 不作显示, 电源/反馈 LED23 或者是亮的或者是闪烁的, 这取决于微控制器内的允许挂起标志是如何配置的。接近计算机系统的用户会注意到监视器 11 没有显示, 认为系统处于挂起状态 154 或关机状态 156, 于是按下电源开关 21 以试图使系统进入正常操作状态 150。如果按下电源开关 21 使系统进入挂起状态 154 或关机状态 156, 那么, 上述用户恰好关闭或挂起了计算机, 这正与用户的意图相反。所以, 在处于待机状态 152 时按下电源开关 21 应能使计算机系统从待机状态转换至正常操作状态。CPU40 即使是空闲的也会立即检测出是否按下了电源开关。硬件的中断使 CPU40 每秒脱离空闲状态约 20 次, 此后, 在下一个 APM 获得事件 (Get Event) 的过程中, 查询微处理器 U_2 确定是否按下了开关 21。

如果计算机 10 处于正常操作状态 150, 有两个事件会使该计

计算机进入挂起状态 154。首先, 如果不活动性挂起计时器到时, 计算机系统 10 会转换至挂起状态 154, 如标号 166 所示。第二, 在写至微控制器 U_2 的允许挂起标志已被设定的情况下用户可通过按下电源开关 21 立即使计算机 10 进入挂起状态 154, 如标号 166 所示。此外, APM 驱动程序会通过“设置电源状态: 挂起”命令发出挂起请求, 此请求使得 APM BIOS 驱动程序调用挂起例程。在处于挂起状态 154 时用户可通过按下电源开关 21 而转换至正常操作状态 150。如标号 168 所示。

此外, 有若干外部事件可用于使系统 10 从挂起状态 154 转换至正常操作状态 150, 如标号 168 所示, 或从关闭状态 156 转换至正常操作状态 150, 如标号 160 所示。例如, 图 6A 电路中微控制器 U_2 内的电话振铃检测电路配置成能使系统 10 在所连接的电话线铃响时离开关机状态 156 或挂起状态 154 并进入正常操作状态 150。这种特点有助于系统接收电传数据或数字数据。计算机系统响应电话铃声进入正常操作状态, 并执行诸如接收到达的传真发送、上装或下装文件、对系统进行远距离存取等预置的功能, 且响应不活动性挂起计时器的到时而再次进入挂起方式, 同时, 该计算机系统只在处于正常操作状态时才消耗电能。

同样, 微控制器 U_2 实现唤醒告警计数器的分钟数, 它允许告警型事件使系统 10 离开挂起状态 154 或关机状态 156 并进入正常操作状态 150。这种系统对在特定情况下发送电传或数字数据以降低电话使用率来说是有用的, 并且对执行诸如用磁带备份系统备份系统硬盘驱动器 31 之类的系统维护来说也是有用的。在后一种情况下, 设置上述唤醒告警的分钟数以便在调度程序使磁

带备份程序执行之前将机器开启一段固定的时间。另外，APM BIOS 调度程序可用于使磁带备份程序运行。

最后，如果计算机系统 10 处于待机状态 152 并且不活动性挂起计时器到时，那么，计算机系统 10 就会转换至挂起状态 154，如标号 170 所示。计算机系统 10 不能从挂起状态 154 往回转换至待机状态 152，但只能如转换 168 的文字所说明的那样转换至正常操作状态 150。

很明显，计算机系统 10 不能立即改变状态。在相对四种状态之一的每种转换中，都需要特定的时间来进行必要的系统转换。与图 6 至图 15 有文的文字将详细说明每种转换的时间。

系统硬件

在说明 CPU40 执行的代码细节之前，先说明实现上述四种状态所需的硬件是有益的。图 5 显示了电源 17 的框图。电源 17 带有两个部件：控制部件 174 和初级/稳压部件 172。电源 17 带有若干输入：Line - In (线路输入)，它从通常的壁装电源插座接收 115 伏交流电或 220 伏交流电；以 ON(开)，它控制着电源 17 的稳压活动。电源 17 也带有若干输出：AC Line - Out (交流电线路输出)； $\pm 5\text{VDC}$ (± 5 伏直流电输出)； $\pm 12\text{VDC}$ (± 12 伏直流电输出)；AUX5 (辅助输出)；GND (接地) 以及 POWERGOOD (电源良好)。AC Line - Out 是 115 伏交流电，它一般流至视频显示终端 11 的电力输入口 (未显示)。控制部件 174 接收 ON 输入并产生 POWERGOOD 输出。初级/稳压部件 172 有选择地将来自 Line - In 输入的 115 伏交流电稳压降至 ± 5 伏直流电和 ± 12 伏直流电。初级/稳压部件 172 是否在 $\pm 5\text{VDC}$ 和 $\pm 12\text{VDC}$ 线路处

稳压取决于和控制部件 174 相连接的 ON 的值。在所述的最佳实施例中，控制部件 174 应使用诸如适当的光隔离器为产生 ON 信号的电路提供绝缘。

Line - In 输入和 AC Line - Out, $\pm 5\text{VDC}$ 、 $\pm 12\text{VDC}$ 、GND 以及 POWERGOOD 输出在本技术中都是周知的。当电源 17“关闭”，即不从 Line - In 中提供稳压电压时，POWERGOOD 为逻辑 0。当电源 17“接通”时，电源 17 从 115 伏交流电 Line - In 中产生 ± 5 伏和 ± 12 伏的直流稳压电压。正如本技术中所周知的那样，上述四种稳压电压以及相关的 GND 是“系统电力”。当所说的稳压电压达到可接受的容差范围内的水平时，POWERGOOD 信号就会改变成逻辑 1。每当 +5 或 +12 伏线路落在容差范围之外，POWERGOOD 信号都会变成逻辑 0，以便指示这种情况。

AUX5 输出为主板提供辅助的 +5 伏直流电。当把电源 17 插入供给额定 115 伏直流电的通常壁装电源插座内时，无论电源 17 是处于“接通”还是“断开”初级/稳压部件 172 都会在 AUX5 处提供稳压了的 +5 伏直流电。因此，电源 17 在接收交流电时总会经由 AUX5 提供额定 ± 5 伏的直流电。AUX5 输出不同于前述 +5 伏的输出，因为初级/稳压部件 172 只有在电源 17“接通”时才在该 +5 伏输处产生稳压的 +5 伏直流电。AUX5 还在以下方面不同于前述的 +5 伏输出，即：在所述的最佳实施例中，初级/稳压部件 172 经由前述 +5 输出提供电压为 +5 伏多个安培数的直流电，而初级/稳压部件 172 则经由 AUX5 输出提供电压为 +5 伏不超过一安培的直流电。

一般的先有电源使用了高安培数的双掷开关以使得

Line - In 输入与电源的稳压部分接通及断开。本发明的电源 17 不使用高安培数的双掷开关。相反, 开关 21 控制着产生 ON 信号的电路。在所述的最佳实施例中, 开关 21 是瞬时单极单掷按钮开关。但是, 本技术的专家可以使图 6A 的电路适应使用诸如单极双掷开关之类的其它类型开关。AC Line - In 总是从壁装电源插座连接至初级/稳压部件 172。当 ON 是逻辑 1 (近似地说, AUX5 为额定 +5 伏直流电) 时, 初级/稳压部件 172 并不将 115 伏的 AC Line - In 稳压成经由 ± 5 或 ± 12 输出的 ± 5 或 ± 12 伏直流电流。初级/稳压部件 172 只在 AUX5 输出处提供低安培数的额定 +5 伏直流电。另一方面, 当 ON 是逻辑 0 (近似为 GND) 时, 初级/稳压部件 172 将 115 伏的 AC Line - In 分别稳压成经由四个 ± 5 和 ± 12 输出的 ± 5 及 ± 12 伏直流电。因此, 当 ON 为 1 时, 电源 17 是“断开”的, 当 ON 是 0 时, 电源 17 是“接通”的。

如果指定的话, 就可从通常电源供应商那里获得类似上述电源 17、带有 AUX5 输出和 ON 输入的电

源。参照图 6A, 它显示了本发明之计算机系统 10 的概略电路图。图 6A 中的电路用于形成开关 21, 电源/反馈 LED23、电源 17、视频显示终端 11 以及在 CPU40 中执行的代码之间的连接。

如图 6A 所示, 上述电路包括四个集成电路以及电路中各种分立的组件, 上述四个集成电路是: U_1 , 第一预编程的 PAL16L8; U_2 , 预编程的 83C750 微控制器; U_3 , 本技术中众所周知的 74LS05; 以及 U_4 , 第二预编程的 PAL16L8 (未显示)。一般来说, 电路 U_1 和 U_4 (未显示) 将图 3A 和图 3B 的主板 I/O 总线 90 与微控制器 U_2 连接起来, 微控制器与图 6A 的其余电路相连, 而这些电

路则与开关 21、电源17、视频显示终端 11 以及可编程时钟合成器 906 相连。时钟合成器 906 可以是本技术中普通人员所周知的多种这类设备中的一种。其中一种是由Chrontel 公司生产的可从多方面广泛获得的 CH9055A。

图 6A 的电路还包括: 开关 21; 16MHz 晶体检波器 Y_1 ; 十八个电阻 $R_1 - R_{18}$; 八个电容 $C_1 - C_8$; 三个 N 型 MOSFET (金属氧化物半导体场效应晶体管) $Q_1 - Q_3$, 这些 MOSFET 在所述的最佳实施例中是适于起逻辑开关作用的标准低电流 NMOSFET; 以及六个 IN4148 小型信号二极管 $CR_1 - CR_6$, 这些电器件全部如图 6A 所示那样配置且相连。电阻 $R_1 - R_8$ 为 1/4 瓦电阻, 它们具有图 6A 所示的电阻值, 误差为 $\pm 5\%$ 。电容 C_1 是 $10\mu F$ ($\pm 10\%$) 的电解电容。电容 C_2 和 C_3 是 22PF ($\pm 10\%$) 的钽电容。电容 $C_4 - C_8$ 是 $0.1\mu F$ ($\pm 10\%$) 的陶瓷电容。最后, 电容 C_9 是 1000PF ($\pm 10\%$) 的陶瓷电容。

正如在本技术中所周知的那样, 晶体检波器 Y_1 和电容 C_2 及 C_3 产生微控制器 U_2 用来控制操作定时的信号。二极管 CR_1 和 CR_3 , 以及电阻 R_{14} 将 AUX5 的信号隔离于 VBAT 信号, 同时允许 AUX5 信号补充 VBAT 信号, 这是因为, 当电源 17 产生 AUX5 信号时, 电池 171 并未放电。相反, AUX5 的信号经由二极管 CR_1 和 CR_2 降压, 从而转为与 VBAT 相连的设备提供适当的电压。此外, VBAT 的线路绝缘于 AUX5 的线路。

第二 PAL U_4 (未显示) 与地址线 SA (1) 至 SA (15) 以及 AEN

(地址启动)线相连。SA (1) 至 SA (15) 以及 AEN 均是图 3A 和图 3B 所示的主板 I/O 总线 90 的一部分。仅将第二 PAL U_4 编程为一地址解码器, 以便在地址线 SA (1) 至 SA (15) 上出现预定的地址且 AEN (地址启动) 线活动时提供一低电平有效信号 DCD#。在所述的特定实施例中, 对第二 PAL U_4 进行预先编程以便在地址 0ECH 和 0EDH 处对两个接连的 8 位 I/O 端口解码。此外, 正如本技术的专家所周知的那样, 也可以由诸如内存控制器或 ISA 控制器芯片组之类的其它电子设备来产生 DCD# 信号。

对第一 PAL U_1 编程以提供以下几个功能: (i) CPU 与微控制器 U_2 之间的读/写接口以便在 CPU40 与微控制器 U_2 之间传递命令和数据; (ii) 对鼠标器中断 INT12 和键盘中断 INT1 进行逻辑或; 以及 (iii) 响应来自 CPU40 的命令, 复位输出来复位微控制器 U_2 。

第一 PAL U_1 使用了两个接连的 I/O 端口, 本文也称之为“电源管理端口”。第一 PAL U_1 具有来自主板 I/O 总线 90 的八个输入: SD (4)、SD (0)、SA (0)、IOW#、IOR#、RST_DRV、IRQ1 以及 IRQ12。正如本技术的专家所周知的那样, 引线 7 (I6) 处的高电平有效信号 RST_DRV 输入可将第一电路 U_1 复位成已知的初始状态, 而上述 RST_DRV 输入则是由内存控制器 46 所产生的。

微控制器 U_2 的复位线 RST751 位于引线 9 处。复位子电路 920 用于产生 RST751 信号, 并且包括: 四个电阻 R_4 、 R_{16} 、 R_{17} 和 R_{18} ; 两个电容 C_1 和 C_8 ; 以及两个 MOSFET Q_2 和 Q_3 , 这些器件如图 6A 所示与第一 PAL U_1 和微控制器 U_2 呈电路通连。复位子电路

920 将来自第一电路 U_1 的复位输出信号 RESET 转接成微控制器 U_2 的复位输入信号 RST751, 因此, 当 RESET 线处于逻辑 1 时, 会使得 RST751 线也会逻辑 1, 从而能复位微处理器 U_2 。

第一 PAL U_1 响应 CPU40 将逻辑 1 写至控制端口 0EDH 的第 0 位而复位微控制器 U_2 。将逻辑 1 写至控制端口 0EDH 的第 0 位会导致第一 PAL U_1 将 RESET 线置成逻辑 1, 而 RESET 线又将 RST 751 线置成逻辑 1, 从而复位了微控制器 U_2 。CPU40 通过将逻辑 0 写至控制端口 0EDH 的第 0 位而清除复位请求。

此外, 如图 6C 所示, 在通往电源 17 的交流电源“降压”或“消失”时, AUX5 的电压会下降, 此后, AUX5 信号的电压必然会上升一定的量, 不论何时出现这种情况, 复位子电路都会将 RST751 线置为逻辑 1, 从而复位了微控制器 U_2 。83C750 的制造者 Philips 公司建议使用简单的 RC 电路 (阻容电路) 以防止复位问题; 但是, 简单的 RC 电路会使 83C750 在电源降压期间锁住。在图 6A 的特定结构中, 当 AUX5 电压在一段大于由 R_4 、 R_{16} 和 C_1 所决定的时间常数的时间内上升一阈值时, 就会使 RST751 线置成逻辑 1 一段时间, 这段时间是由 R_{17} 和 C_8 所决定的 (从而复位了微控制器 U_2)。这种情况在通常的降压或消失之后会出现。在图 6A 所示的实施例中, 所说的阈值约为直流电 1.5V。

参照图 6C, 它显示了 AUX5 电压随着交流电供给电源 17 而上升那段时间以及发生“降压”那段时间内用于复位电路 920 的波形。在 t_0 之前, 所述电源未产生 AUX5, VBAT 约为 3.3V, Q_3 导通而使 RST751 线接地。在 t_0 时刻, 电源开始产生 AUX5, AUX5

的电压开始以一定的速率上升，所说的速率取决于负荷以及电源内影响 AUX5 的电容。结点 1 即 C_1 与 R_4 之间的结点电容性地连接于 AUX5，所以能随 AUX5 的上升而上升。

在 t_1 时刻，结点 1 达到约为 1.5V，这足以触发 Q_2 ，而 Q_2 则使结点 2 接地。在 t_2 时刻，当结点 2 超过 2.5V 时， Q_3 停止导通，RST751 线借助 R_{18} 突增到 AUX5 的电平并随 AUX5 上升至约为 5V。当 RST751 线变成约 3V 时，就会复位微控制器 U_0 。

在 t_3 时刻，AUX5 停止上升，所以，结点 1 停止上升并开始按 C_1 和 R_4 所确定的速率放电至地电平（第一电路 U_1 的 RESET 线是低的）。在 t_4 时刻，当结点 1 经过约 1.5V 时， Q_2 停止导通，结点 2 按 C_8 和 R_{17} 所确定的速率充电。在 t_5 时刻，当结点 2 超过约 2.5V 时， Q_3 导通并使 RST751 接地。因此，完成了电源接通时的复位，计算机系统通常处于这样的状态：AUX5 为 5V、VBAT 为 3.3V，结点 1 接地且结点 2 处于 VBAT 的状态。

在 t_6 时刻，AUX5 线处开始降压且 AUX5 放电。由于结点 1 电容性地连接于 AUX5，所以它试图随 AUX5 的变化而变化，但却不可能产生这种变化，因为，第一电路 U_1 内的二极管会阻止结点 1 处于大大低于 -0.5V 的状态。在 t_7 时刻，AUX5 处于最低点并开始再度上升。而且，结点 1 随 AUX5 变化并且上升。在 t_8 时刻，结点 1 达到约 1.5V，这足以触发 Q_2 ，而 Q_2 则使结点 2 接地。在 t_9 时刻，当结点 2 超过 2.5V 时， Q_3 停止导通，RST751 线会借助 R_{18} 突增至 AUX5 的电平并随 AUX5 上升至约为 5V。当 RST751 线

变成约为 3V 时, 就会复位微控制器 U_2 。

在 t_{10} 时刻, AUX5 停止上升, 所以, 结点 1 停止上升并开始按 C_1 和 R_4 所确定的速率放电从而接地 (第一 PAL U_1 的 RESET 线是低的)。在 t_{11} 时刻, 当结点 1 通过约为 1.5V 时, Q_2 停止导通, 结点 2 按 C_8 和 R_{17} 所确定的速率充电。在 t_{12} 时刻, 当结点 2 通过约 2.5V 时, Q_3 导通并使 RST751 线接地。因此, 完成了因降压而引起的复位周期。请注意, 在这种特定的降压期间, 结点 1 不会上升至 3V 以上, 因而, 尽管微控制器与 RST751 的引线相连, 结点 1 也不可能复位微处理器。但是, AUX5 的电压会降至 4V 以下, 这足以使微处理器 U_2 进入不确定的状态。

用于触发一次复位的阈值由参照值所限定, 所以, 为了能使阈值电压上升或下降, 就必须使参照值 (在本例中是 VBAT) 上升或下降。复位电路为微控制器 U_2 提供了提高重置保护的好处, 同时又非常廉价并且在不复位微处理器 U_2 时基本上不耗电。

参照图 6A, 微控制器 U_2 通过第一 PAL U_1 与 CPU 相接并具有多个输入、输出以及可内部控制的功能。

SWITCH (开关) 信号在引线 8 (P0.0) 处输入并反映了按钮 21 的当前状态。按钮 21 是常开的。当按钮 21 断开时 SWITCH 线通过电阻 R_1 被置为逻辑 0 (接地)。当按下按钮 21 从而引发闭合事件时, SWITCH 线通过电阻 R_{13} 被置为逻辑 1 (AUX5)。电容 C_6 起作用以消除开关闭合事件弹跳, 正如本技术的专家所周知的那样, 在微控制器 U_2 内通过读取 SWITCH 预定的次数例如 50 次并确认在所有的读取中 SWITCH 线都是相同的, 从而进一步消除

了开关 21 闭合事件的弹跳。

微控制器 U_2 可直接控制对电源 17 的稳压。如图 6A 所示, On (开) 信号在引线 5 (P3.0) 处输出并经由电阻 R_6 与 SWITCH 信号作线“或”以便控制电源的 ON# 信号。当 ON 信号为逻辑 1 时, MOSFET Q_1 导通, 从而使 ON# 线 (JP_2 的引线 2) 成逻辑 0 (接地), 因而使电源 17 开始通过 $\pm 5VDC$ 和 $\pm 12VDC$ 线向计算机系统提供稳压的电力。另一方面, 当 ON 线为逻辑 0 时, MOSFET Q_1 不导通, 所以, ON# 线 (JP_2 的引线 2) 会被电阻 R_7 置成逻辑 1 (AUX5), 从而使得电源 17 停止经由 $\pm 5VDC$ 和 $\pm 12VDC$ 线供给稳压的电力。

微控制器 U_2 响应开关 21 的接通事件以及通过微控制器 U_2 内 CPU40 可写的可写寄存器位响应 CPU40 从而控制 ON 线的状态。AUX5 对微控制器 U_2 加电, 所以, 微控制器 U_2 总是处于加电状态并能执行代码且控制着计算机系统。如果电源 17 不通过 $\pm 5VDC$ 和 $\pm 12VDC$ 线向计算机系统提供稳压的电力并且 (i) 按下开关 21 或者 (ii) 出现了所说的外部事件之一, 那么, 微控制器 U_2 就会确认 ON 信号, 从而使得电源 17 通过 $\pm 5VDC$ 和 $\pm 12VDC$ 线向计算机系统提供稳压的电力。释放开关 21 之后, 微控制器会继续确认 ON 信号。

作为一种后备系统, 电源 17 也可在用户通过按钮 21 的直接控制下接通。如果微控制器 U_2 正如计算机系统未响应按下电源开关 21 而起动所表现出来的那样停止起所希望的作用, 那么, 一般就可以使用上述选择。如图 6A 所示, 开关 21 还通过二极管 CR_2 、

MOSFET Q_1 、电阻 R_7 以及连接器 JP_2 控制电源 17 的 ON# 线。在一般情况下按钮 21 是断开的，SWITCH 线通过 R_1 被置为逻辑 0 并且 MOSFET Q_1 不导通，所以 ON# 线 (JP_2 的引线 2) 通过电阻 R_7 被置为逻辑 1 (AUX5)，因而电源 17 不經由 $\pm 5VDC$ 和 $\pm 12VDC$ 线提供稳压的电力。当用户按下并按住按钮 21 时，SWITCH 线被置为 1，MOSFET Q_1 导通，从而将 ON# 线 (JP_2 的引线 2) 置为逻辑 0 (GND)，因此使得电源 17 开始經由 $\pm 5VDC$ 和 $\pm 12VDC$ 线提供稳压的电力。在继续按住按钮 21 的情况下，计算机系统加电之后，BIOS 会使 CPU40 检测微控制器 U_2 是否还在起作用。如果不起作用，CPU40 就会复位微控制器 U_2 ，而微控制器在被复位之后会检测到开关 21 正被按下。因此，在按钮 21 仍被按住的情况下，微处理器会确认 ON 信号，所以用户在知道了微控制器目前正在控制电源 17 的情况下会最终放开开关 21。为了使用这种后备的选择，用户必须按下按钮 21 数秒的时间即在标志 (logo) 出现之后约为两秒。

微控制器 U_2 只响应 (i) 按下开关 21 或者 (ii) CPU40 命令微控制器关闭系统而关闭计算机系统。对微控制器来说，上述事件都是相同的，这是因为，微控制器配置成开关 21 的接通事件或 CPU40 都能引发一次开关按下，硬件上的按钮按下/释放基本上看成是与软件上的按钮按下/释放一样的。如果微控制器 U_2 中的允许挂起标志被清除，则微控制器 U_2 在没有 CPU 指令的情况下便关闭计算机系统。在这种情况下，当计算机系统处于加电状态且清除了允许挂起标志，那么，微控制器 U_2 就会响应开关 21 的接

通事件而清除 ON 信号，从而使电源 17 停止经由 $\pm 5\text{VDC}$ 和 $\pm 12\text{VDC}$ 线向计算机系统提供稳压的电力。在释放开关 21 之后，ON 信号仍保持在被清除的状态。

微控制器 U_2 也会响应 CPU 在系统状态已成功地保存到硬盘驱动器上 (挂起) 之后必定会发出的命令而关闭计算机系统。微控制器 U_2 响应上述命令而清除 ON 信号，从而使电源 17 停止经由 $\pm 5\text{VDC}$ 和 $\pm 12\text{VDC}$ 线向计算机系统提供稳压的电力。

微控制器 U_2 在某些外部事件发生时也会检测并影响计算机系统。EXT_RING (外部振铃) 信号在引线 7 (P0.1) 处输入并使微处理器 U_2 检测来自加电后外部调制解调器 902 的振铃。正如本技术的专家所周知的那样，当在插塞尖端与振铃电话线上检测到振铃信号时，一般的外部调制解调器会提供振铃信号，该信号会以周知的 RS-232C 格式触发逻辑 1。上述信号经由二极管 CR_6 通向微控制器 U_2 并被电阻 R_{10} 和 R_{11} 分流且最终经由 EXT_RING 线输入微控制器 U_2 。微控制器 U_2 每 25 毫秒对触发的信号进行一次取样并加以分析，从而，对两个连续的样本这一输入是逻辑 1 时，便认为存在有振铃。微控制器 U_2 响应所满足的上述条件确认 ON 信号，从而使电源 17 经由 $\pm 5\text{VDC}$ 和 $\pm 12\text{VDC}$ 线向计算机系统提供稳压的电力。为了用 EXT_RING 信号来检测呼入电话，必须存在有从外部加电的调制解调器 902。

此外，提供符合 RS-232 规范的二进制信号 (或充分接近 RS-232 规范从而足以确认 EXT_RING 信号) 的其它设备可与 EXT_RING 线相接并用于唤醒计算机系统，例如，这些设备可以

是运动传感器、防盗警报器、语音激活传感器、光传感器、红外线传感器、“鸣叫”型传感器等。

如图 6A 和图 6B 所示，本实施例还带有用于检测来自内部调制解调器 900 的电话振铃信号的装置，此装置带有一以振铃检测电路为基础的光隔离器 OPTD1。例如 Hewlett Packard 生产多种合适的光隔离器，这些隔离器可从多方面广泛地获得。可将内部调制解调器 900 设计在系统主板 20 的电路内，或者插在扩展槽 78 上。在后一种情况下，必须对调制解调器 900 加以改进以提供一 Berg 或类似的接头，从而使来自光隔离器 OPTD1 的信号电连接于图 6A 电源管理电路的电路。有许多调制解调器的生产商改进了它们的内部调制解调器以提供适合于和本发明电路一道使用接头。EXT_WAKEUP# (外部唤醒) 信号在微控制器 U_2 的引线 4 (P0.2) 处输入并用于从内部调制解调器中输入来自振铃检测光隔离器 OPTD1 的信号。此信号经过电阻 R_9 和 R_5 二极管 CR_6 以及电容 C_9 ，最后经由 EXT_WAKUP# 线输入微控制器 U_2 。

内部调制解调器 900 的阈值及保护部分 905 与标准的插塞尖端 (Tip) 和振铃电话线相接，并且 (i) 提供防止可能损坏调制解调器 900 的闪电和其它电学事件的保护以及 (ii) 如调制解调器设计技术的专家所周知的那样设置振铃阈值电压。

微控制器 U_2 检测并分析来自光隔离器 OPTD1 的触发信号，从而，每当该信号在 EXT_WAKEUP 上的三个连续信号周期具有 15.1Hz 至 69.1Hz 的频率，就认定存在有振铃。与必须加电才能沿 EXT_RING 提供振铃信号的 EXT_RING 信号不同，对光隔离器 OPTD1 来说，不必为内部调制解调器 900 加电就能沿

EXT_WAKEUP# 线提供适当的信号, 该信号通常可由 R₅ 升至 AUX5。

如果 CPU40 带有系统管理中断 (SMI) (CPU40 不一定要带有 SMI 才能使系统利用本发明的多种优点), 那么, 微控制器 U₂ 就会通过 CPU 的系统管理中断 (SMI) 中断 CPU40。SMI_OUT# (系统管理中断输出) 信号在微控制器 U₂ 的引线 3 (P3.2) 处输出并使得微控制器 U₂ 在不等待操作系统确认或允许中断的情况下立即中断 CPU40。位于微控制器 U₂ 内 CPU40 可写的一个可写寄存器位控制 SMI_OUT# 线的状态。此外, 微控制器 U₂ 可以确认 SMI_OUT# 信号从而 (i) 根据在 ACTIVITY# (活动) 线上所检测到的活动 (ii) 或者在微控制器 U₂ 使电源 17 停止向计算机系统提供稳压电力之前中断 CPU40。从 CPU 到微控制器 U₂ 的命令可以使上述一个或两个事件有效或失效。

对于每个 SMI, CPU40 内的微码都将 CPU 的状态从内存保存至特定的 CPU 状态保存区域或保存至内存。此后, CPU40 执行 SMI 中断处理程序, 该程序执行以下功能。为了恢复 CPU 的状态, SMI 中断处理程序发出 RSM (继续) 指令, 此指令使 CPU40 从上述特定的保存区域中恢复 CPU 自身的状态。

在 CPU40 使微控制器 U₂ 通过 CPU 的 SMI 中断 CPU40 之前, CPU40 将一指示系统管理中断 (SMI) 原因的值写至 CMOS NVRAM 内的变量中。CMOS NVRAM 内的这个值缺省为 00H, 它向 CPU40 指明正如在微控制器 U₂ 使电源 17 停止提供稳压电力之前所出现的那样微控制器 U₂ 正在异步地中断 CPU40。每次

SMI 之后，CPU40 都将 CMOS NVRAM 内的上述变量置为 00H。在假定微控制器 U₂ 即将使计算机系统掉电的情况下，CPU40 会响应上述值去执行某些任务。在 CPU40 通过周期性地重新启动微控制器 U₂ 内的降低功耗延时计时器而在微控制器 U₂ 使计算机系统降低功耗之前，CPU40 可以延时一段时间。

在计算机系统掉电之前的这段时间内，CPU40 可以执行多种任务。例如，由于用户可能改变了影响唤醒警铃的一个或多个参数，所以 CPU 会重新计算出距唤醒的新的分钟数值并将该分钟数值写至微控制器 U₂。此外，CPU 还将以后要写入硬盘驱动器 31 的某些信息如计算机系统从最近一次通电开始运行的时间等写至 CMOS NVRAM。

CPU40 所写的其它值包括：01H，它表明 CPU40 要转至标号 254 处的挂起例程；02H，它表明 CPU40 要转至标号 454 处的继续例程；以及 0FFH，它表明 CPU40 要在 E000H 段数据结构内创建前述特定的 CPU 状态保存区域。

在本实施例中，微控制器进行控制使显示器 11 不显示。DISP _ BLANK (显示器不显示) 信号在微控制器 U₂ 的引线 1 (P3. 4) 处输出并直接控制着显示器 11 的不显示。两个非门 U3D 和 U3E 通过 ESYNC# 和 BLANK# 线与 DISP _ BLANK 信号相连。在 ESYNC# 和 BLANK# 处于逻辑 1 (VCC) 的情况下，视频控制器 56 会产生视频信号。当 ESYNC# 和 BLANK# 线处于逻辑 0 (接地) 时，视频控制器 56 停止产生视频信号。位于微控制器 U₂ 内 CPU40 可写的一个可写寄存器位控制着 DISP _ BLANK 的状态。当计算机系统进入待机状态 152 时，CPU40 会指示微控制器 U₂ 使

显示器不显示。此外，响应开关21的接通事件而依次对 DISP_ BLANK 线置1和清0。与此相似，在本例中为 INT₁和 INT₂的任何一种活动中断处的活动都会使微控制器对 DISP_ BLANK 线清0，从而使得视频控制器56产生视频信号。

此外，微控制器 U₂控制着时钟合成器906所产生的时钟信号的频率。三个 Berg 型跳接器（未显示）JP₀、JP₁以及 JP₂按下述方式控制着时钟合成器：当 JP₀ = 0、JP₁ = 1 且 JP₂ = 0 时，时钟合成器产生 33MHz 的时钟信号；当 JP₀ = 1、JP₁ = 1 且 JP₂ = 0 时，时钟合成器产生 25MHz 的时钟信号；当 JP₀ = 0、JP₁ = 1 且 JP₂ = 0 时，时钟合成器产生 8MHz 的时钟信号。对应于 JP₀、JP₁和 JP₂的三个时钟线 CLK0、CLK1 以及 CLK2 也控制着时钟合成器906。如图6A所示，微控制器 U₂通过 CLK_SLOW# 信号控制着上述时钟线 CLK0、CLK1 和 CLK2，而 CLK_SLOW# 信号则在微控制器 U₂的引线2 (P3.3) 处输出。如图所示，带有开放集电极输出的非门 U3A、U3B 和 U3C 双重反相 CLK_SLOW# 信号。而且，电阻 R₁₅和 R₈都是负载电阻，它们分别用于将 U3A 的集电极开路的输出和朝向时钟合成器906的 CLK0 输入上升至逻辑1。

上述三个时钟信号 CLK0、CLK1 和 CLK2 以及三个跳接器 JP₀、JP₁和 JP₂按如下方式控制着时钟合成器：当 CLK_SLOW# 信号为逻辑1时，CLK1 和 CLK2 信号也为逻辑1，时钟合成器906跳接器 JP₁、JP₂所控制并产生供计算机系统使用的 25MHz 和 33MHz 的较高频时钟信号。另一方面，当 CLK_SLOW# 信号为逻辑0时，CLK1 和 CLK2 信号也为逻辑0，时钟合成器906会产

生供计算机系统使用的 8MHz 较低频信号,从而使计算机系统有较少的功耗。如图 6A 所示,一 Berg 型跳接器将 CLK_SLOW# 线与 CLK0 线分开。如果一个跳接器就位,那么,CLK0 线中就会跟随 CLK_SLOW# 信号。另一方面,如果没有跳接器就位,那么,电阻 R_8 就会在与 CLK_SLOW# 信号无关的情况下将 CLK0 置为逻辑 1。位于微控制器 U_2 内 CPU40 可写的一个可写寄存器位控制着 CLK_SLOW# 线的状态。此外,微控制器 U_2 可以响应 ACTIVITY# 线上的活动将 CLK_SLOW 线清零。正如本技术的专家所知道的那样,本发明也可以使用其它的时钟合成器,这时需改变微控制器 U_2 与时钟合成器之间的相互连接关系以便符合所使用的特定合成器的特殊规格。

另外,微控制器 U_2 还直接控制着电源/反馈 LED23 的发光。LED_CNTRL (LED 控制) 信号在引线 22 (P3.6) 处输出并使得微控制器 U_2 直接控制电源/反馈 LED23。电阻 R_2 和 R_3 以及二极管 CR_4 和 CR_5 根据处于逻辑 0 的 LED_CNTRL 线使 AUX5 电源线或 VCC 电源线驱动电源/反馈 LED23。当 LED_CNTRL 线处于逻辑 1 时,电源/反馈 LED23 不亮。正如以下将要详细说明的那样,微控制器 U_2 响应开关 21 的接通事件、响应唤醒警铃、响应振铃检测输入的一次或多次振铃或者响应处于待机方式下的计算机系统而控制 LED_CNTRL 线的状态。

微控制器 U_2 可将 LED23 控制成简单的电源 LED。这样,在开关 21 使计算机系统从关机状态 156 或挂起状态 154 转变成正常操作状态 150 的接通事件之后,LED23 会发光。同样,在开关 21

使计算机系统从正常操作状态 150 转变成挂起状态 154 或关机状态 156 的释放事件之后,微控制器 U_2 会使 LED23 熄灭。

此外,微控制器 U_2 可以使 LED23 按特定的速率例如每秒一次有选择地闪亮,以指示计算机系统正处于待机状态 152。再有,微控制器 U_2 还可以使 LED23 按不同的速率例如每半秒一次有选择地闪亮,以指示计算机系统被振铃或警铃所唤醒或者计算机系统处于关机或挂起状态。另外,微控制器 U_2 也可以使 LED23 以成组闪亮的方式有选择地闪亮以指示计算机系统因诸如振铃、警铃之类的外部事件而加电以及因不活动性挂起计时器到时而掉电的次数。在这种情况下, BIOS 具有一种或多种功能以允许 OS 及应用程序修改微控制器 U_2 使 LED23 闪亮的次数。例如,如果计算机系统被振铃唤醒并接收到了呼入的传真传输,那么,运程通信应用程序就会调用特定的 BIOS 功能使闪亮数加一。此后, BIOS 使 CPU40 将新的闪亮值写至微控制器 U_2 , 从而使 LED 23 闪亮所命令的次数。

POWERGOOD 信号在微控制器 U_2 的引线 4 (P3.1) 处输入并允许微控制器 U_2 和 CPU40 使用这一信号。具体地说,微控制器使用 POWERGOOD 信号使基于反馈的检错及纠错电路确定电源 17 是否出现故障并排除掉这种故障。正如本说明书其它部分所说明的那样,如果已确立了 ON 信号一段时间(例如三秒)并且 POWERGOOD 信号为逻辑 0 从而指示电源 17 未以适当的电平提供稳定的电压,那么,微控制器 U_2 就假定电源 17 因诸如过载电流而出现故障。因此,为了能够排除该故障,微控制器 U_2 会停止确

立 ON 信号一段时间 (例如 5 秒) 以便排除上述故障。此后, 微控制器 U_2 会重新确立 ON 信号并等待 POWERGOOD 变成逻辑 1, 从而表示电源 17 目前正在向计算机系统提供稳压的电力。在不存在上述基于反馈的检错及纠错的情况下, 电源 17 总是会继续处于出错状态并且微控制器 U_2 总是不断地确立着 ON 信号以试图使电源 17 开始产生稳压的电力。唯一的方案是从上述电源上拔掉交流电以排除所说的故障。

图 6D 显示了电源检错与纠错电路的另一实施例。该实施例使用了四个 FET $Q_{10} - Q_{13}$, 电阻 $R_{20} - R_{23}$ 、电容 C_{20} 以及 74HC132 以检测电源 17 何时出现故障并排除这种故障。当 ON 信号为 HIGH (高), AUX5 加电且 VCC 低于触发 Q_{11} 的阈值时, Q_{12} 将 ON 信号置为 LOW (低) 一段时间, 该时间是由 R_{22} 和 C_{20} 所决定的, 从而排除电源中的故障。

ACTIVITY# 信号在微控制器 U_2 的引线 19 (INT1) 处输入, 微控制器 U_2 使用这一信号以响应键盘 12 和鼠标器 13 上的活动。IRQ₁ 是键盘的硬件中断信号, 它在第一 PALU₁ 的引线 8 (I7) 处输入, 按下键盘 12 上的按键会使 IRQ₁ 信号产生脉动。IRQ₁₂ 是鼠标器的硬件中断信号, 它在第一 PALU₁ 的引线 11 (I9) 处输入, 移动鼠标 13 或按下鼠标器 13 上的按钮会使 IRQ₁₂ 信号产生脉动。IRQ₁ 和 IRQ₁₂ 在第一电路 U_1 内作逻辑“或”并作为 ACTIVITY# 信号输出。使用 ACTIVITY# 信号会使得微控制器 U_2 不会遗漏键盘 12 或鼠标器 13 的活动。

当处于待机状态时, 上述任一种中断的活动都会使微控制器

立刻恢复视频显示。在从待机状态 152 返回至正常操作状态 154 时,按上述方式使用中断 IRQ_1 和 IRQ_{12} 会给用户直接的反馈,该反馈是恢复了的视频显示。否则,正如图 9 中的文字所述的那样,用户可能在几秒钟之后当 APM 检查用户活动时,才接收到反馈。

利用 SD (0)、SD (1)、SD (2)、SD (3)、IO_ STROBE 以及 PROC_ RDY 进行 CPU40 与微控制器 U_2 之间的通讯, SD (0) 在第一 PAL U_1 的引线 18 (I/06) 处输入并经由 RWD0 线输入至微控制器 U_2 , RWD0 线则在第一电路 U_1 的引线 B (I/01) 处输出并在微控制器 U_2 的引线 13 (P1. 0) 处输入, SD (1) 在微控制器 U_2 的引线 14 (P1. 1) 处输入, SD (2) 在微控制器 U_2 的引线 15 (P1. 2) 处输入, SD (3) 在微控制器 U_2 的引线 16 (P1. 3) 处输入, SD (4) 在第一电路 U_1 的引线 6 (I5) 处输入, IO_ STROB# 在微控制器 U_2 的引线 18 (INTO) 处输入, PROC_ RDY 在微控制器 U_2 的引线 20 (P1. 7) 处输出。将第一 PAL U_1 和微控制器 U_2 配置并编程为以便提供:

(i) 从 CPU40 沿 SD (0) 经由 RWD0、SD (1)、SD (2) 以及 SD (3) 到微控制器 U_2 的四位并行写操作, 其中一个地址基本上是用于复位微控制器 U_2 的一位写操作, 而其它地址则是写入微控制器 U_2 的半个字节, 该半个字节只有在数据位 SD (4) 为 HIGH 时才是有效的; 以及 (ii) 由 CPU40 从微控制器 U_2 中沿 SD (0) 经由 RWD0 所进行的串行读操作 (一位), 其中一个地址对应于状态位, 而其它地址则对应于来自微控制器 U_2 的数据位。

参照图 19, 它显示了在微控制器 U_2 内运行的几个例程, 这些

例程始于 1160 处。微控制器 U_2 通常执行两个主例程中的一个：位于任务 1168 至 1216 的通电例程或位于任务 1260 至 1308 处断电例程。当电源 17 在 ± 5 和 ± 12 线处提供稳压的电力，或电源 17 虽不在 ± 5 和 ± 12 线处提供稳压的电力但计算机系统处在通电过程时，微控制器 U_2 就执行通电例程。当电源 17 不在 ± 5 和 ± 12 线处提供稳压的电力或电源 17 虽在 ± 5 和 ± 12 线处提供稳压的电力但计算机系统处于断电过程时，微控制器 U_2 就执行断电例程。此外，还有三种中断驱动例程：一个是在 1220 至 1232 处的用于和 CPU 40 进行通信的例程；一个是在 1236 至 1244 处的用于检测鼠标器 13 或键盘 12 活动的例程；一个是位于 1248 至 1256 处的提供 25 毫秒、半秒、秒以及分钟分辨率的时基的例程。

首先，在 1164 处初始化微控制器 U_2 ，在这期间，初始化所有的变量，初始化计数器变量，初始化计时器中断并使之生效，同时初始化控制通信例程及活动例程的外部中断。

通讯例程是一始于 1220 处的中断驱动例程，它响应被第一 PAL U_1 置为逻辑 0 的 IO_STROBE 而执行，逻辑 0 表示 CPU40 正在开始一命令或查询。简要地说，此例程在 1224 处接收来自 CPU40 的一个或多个半字节命令或查询、在 1128 处执行上述命令或响应上述查询而返回数据、并且在 1232 处将程序执行控制权交还给被中断的代码。

微控制器顺序地接收来自 CPU 构成命令或查询的半字节。在接收一个半字节之后，微控制器将 $PROC_RDY$ 置为 LOW。当微控制器准备好接收下一个半字节时，微控制器再将 $PROC_RDY$ 置为 HIGH。CPU40 一旦检测到在 $PROC_RDY$ 处有上述

从 LOW 到 HIGH 的转换就写下一个命令的半字节。

微控制器 U_2 在执行来自 CPU40 的命令或查询时不能接收另外的命令, 所以, 微控制器 U_2 使 PROC_RDY 保持为逻辑 0, 以便 (通过状态端口的读操作) 向 CPU40 表明微控制器还不能接收下一个命令/查询。当上述工作完成时, PROC_RDY 被置为逻辑 1, 从而 (通过状态端口的读操作) 向 CPU40 表明微控制器 U_2 随时都可以接收下一个命令/查询。

活动例程是一始于 1236 处的中断驱动例程, 它响应被第一 PAL U_1 置为逻辑 0 的 ACTIVITY# 线而执行, 逻辑 0 表示用户已经使用了鼠标器 13 或键盘 12。简要地说, 此例程响应中断的接收而在 1240 处进行下列工作: (i) 设置一个位, 该位表示已存在有鼠标器 13 或键盘 12 的活动; (ii) 如果启动时钟减慢则恢复时钟速度, (iii) 如果屏幕不显示, 则使屏幕显示; (iv) 重新启动故障保险计时器; 以及 (v) 如果有效, 则向 CPU 生成一 SMI。此后, 该例程在 1244 处将程序执行控制权交还给被中断的代码。然后, 如本说明书其它地方所说明的那样, 每当 APM“获得事件”, 管理例程都查询上述例程所设置的位。

计时器例程是一始于 1248 处的中断驱动例程, 它响应内部计时器中断而执行, 内部计时器中断以一个 16 位自由运行的计数器为基础, 该计数器配置成每 25 毫秒产生一次上述内部计时器中断以便为微控制器 U_2 提供时基。计时器例程提供下列时基: 25 毫秒、半秒、秒和分钟。简要地说, 计时器例程在 1252 处接收中断、确定何时出现不同的时间、执行适当的活动并在 1256 处将程序执行控制权交还给被中断的代码。

如果电源未提供稳压的电力并且微控制器配置成响应振铃, 则计时器例程每一瞬间 (25 毫秒) 检查 EXT_RING 线的 RS-232 振铃, 并且, 如果有振铃, 就设置一个位。

如果处于关机状态或挂起状态, 计时器例程每半秒钟就会如本说明书其它地方所说明的那样确定是否应触发 LED23 以便进行外部振铃唤醒指示闪亮。

如果处于待机状态, 计时器例程就会如本说明书其它地方所说明的那样每一秒钟确定是否应触发 LED23 以便进行挂起指示闪亮。

而且, 如果合适的话, 计时器例程每一秒钟都会减小故障保险计时器、减小 APM 故障挂起计时器以及减小电源故障计时器, 同时如果有任何一个计时器到时, 计时器例程还设置一对应的位。故障保险计时器是一个 20 秒的计时器, 它在到时时可使得微控制器切断计算机系统的电源。管理例程响应 APM 的获得事件频繁地重新启动复位故障保险计时器, 所以, 只要 CPU 中运行的代码在正确地执行, 故障保险计时器就不会到时。但是, 如果上述代码真正地停止执行, 故障保险计时器就会到时, 并且, 在假定 BIOS 和其它例程发生故障的情况下, 微控制器 U_2 会响应按下及释放电源按钮 21 使电源 17 停止在 ± 5 和 ± 12 线处提供稳压的电力。

APM 故障挂起计时器是一个 18 秒的计时器, 在故障保险计时器到时以使微控制器关闭计算机系统之前, 当开关 21 处于关闭/释放状态 (表示用户试图关闭计算机系统) 时, APM 故障挂起计时器是有效的, 并且, APM 故障挂起计时器在到时时使计算机

系统设法挂起。同故障保险计时器一样, CPU40 中执行的代码例如 APM Get Events、APM Working On Last Request 以及 APM Reject Last Request 等会频繁地重新启动(复位) APM 故障挂起计时器, 所以, 只要 CPU 40 中执行的代码在正确地执行, APM 故障挂起计时器就不会到时。但是, 如果所说的代码真正停止执行, APM 故障挂起计时器就会到时。

当 APM 故障挂起计时器到时, 微控制器 U_2 会设置一个位。在每次计时器 0 级中断期间检查该位, 正如本技术的专家所周知的那样, 这种情况约每 55 毫秒出现一次。此外, 计时器 0 级中断服务例程重新启动故障计时器。如果计时器 0 级中断服务例程检测到 APM 故障挂起计时器业已到时, 它就会转至挂起例程以试图挂起计算机系统, 如与图 10 有关的文字所述。

由计时器 0 级中断服务例程所引起的中断并不是进行中断的最佳方法。APM 可察觉多种应用程序和适配器, 并且, 这些应用程序和适配器可响应被挂起的计算机系统而执行多种任务。计时器 0 级中断服务例程所引起的中断不能使用 APM 去向这些 APM 可察觉的实体表明即将进行挂起。因此, 在这些实体没有正确地准备好的情况下挂起计算机系统。这样, 计时器 0 级中断服务例程所引起的中断将会把系统保存起来, 所以, 内存中的数据不会丢失, 但是, 在保存了所需数据之后, 用户要重新引导机器以便将计算机系统设置成适当的状态。

APM 故障挂起计时器特别有助于修补 OS 内 APM 驱动程序中的“漏洞”。例如, 当显示 Microsoft Windows 3.1 模态对话框时, Windows APM 驱动程序停止发送 APMget events。因此, 如

果在用户按下电源按钮21以试图挂起系统时有模态对话框显示,那么,计算机系统将不会挂起。微控制器 U_2 会注意到电源开关处于关闭/释放状态,但因为所有的 APM get event 均已停止,所以不会去调用管理例程。按下开关直到用户清除掉模态对话框才会起作用。但是,一旦 APM 故障挂起计时器到时并且计时器 0 级中断服务例程检测到这种到时,有可能在不向 APM 可察觉实体表明计算机系统正在被挂起的范围内将系统的状态保存起来。

每一分钟,计时器例程都会减少唤醒警铃计时器和活动计时器的分钟数。当唤醒计时器的分钟数到时时,如果有效的话,微控制器就会使电源 17 开始在 ± 5 和 ± 12 线处提供稳压的电力。

初始化微控制器 U_2 之后,会在 1168 处测试电源以确定是否断电。如果是通电的,微控制器 17 会在 1172 处检验电源是否有故障。电源 17 有若干种内部保护措施,这些措施会使电源关闭或“出现故障”。微控制器 U_2 按下列方式确定电源 17 是否出现故障:如果微控制器正在运行(表示 AUX5 已加电,即正将交流电提供给电源 17),并且微控制器 U_2 保持着 ON 信号以试图使电源 17 在 ± 5 和 ± 12 线处提供稳压的电力,同时不确立 POWERGOOD 线(表示电源 17 不在 ± 5 和 ± 12 线处提供稳压的电力),那么,就是电源 17 出现了故障并且必须复位该电源。

在任务 1172 处,实际测试电源 17 两次。微控制器 U_2 确立 ON 信号,然后如内部时基所测定的那样等待三秒钟。如果在 ON 信号确立了三秒钟之后未确立 POWERGOOD 信号,那么,微控制器 U_2 就会清除 ON 信号并等待另外的五秒钟。然后,微控制器 U_2 再度确立 ON 信号并等待另外的三秒钟。如果在 ON 信号确立

了三秒钟之后仍未确立 POWERGOOD 信号, 则微控制器 U_2 会清除 ON 信号并认定电源 17 出现了故障。

如果电源出现了故障, 微控制器 U_2 会转至断电例程, 如 1174 处所示。另一方面, 如果电源 17 未出现故障而是关闭了, 则微控制器会在 1175 处使该电源开始在 ± 5 和 ± 12 线上提供稳压的电力并且在 1176 处初始化 I/O 端口、开启 LED23 并使外部中断生效。

图 7 显示了机器在微控制器 U_2 内所维护的开关状态。如该图所示, 所说的状态响应开关 21 的接通事件或其它事件如复位计算机系统 10 以及 CPU40 的写操作等而改变。在电源 17 未提供 AUX5 的情况下, 微控制器 U_2 不被加电, 所以 174 处的开关状态是无意义的。按下开关 21, 来自其它方向的电话振铃、警铃计时器时间数的到时以及来自 CPU40 的命令都会导致微控制器使电源 17 开始向计算机系统提供电力, 如伴随图 6 的文字所述。

如图 7 所示, 开关 21 具有四种由微控制器 U_2 监控的状态: (i) 接通/按下状态 176 (在这种状态下, 用户按着按钮并试图开机); (ii) 接通/释放状态 178 (在这种状态下, 用户已释放了按钮并试图开机), (iii) 断开/按下状态 180 (在这种状态下, 用户按着按钮并试图关机); 以及 (iv) 断开/释放状态 182 (在这种状态下, 用户业已释放了按钮并试图关机)。然后, 在 1180 处微控制器 U_2 测试开关是否处于断开/释放状态, 从而表明用户业已释放了按钮并试图关机。

在处于状态 174 并按下开关 21 时, 微控制器 U_2 进入接通/按

下开关状态 176。释放开关 21 会使微控制器 U_2 进入接通/释放开关状态 178。与此相似, 当复位微控制器 U_2 时, 微控制器 U_2 会进入接通/释放状态 178。再次按下开关 21 会使微控制器 U_2 进入断开/按下开关状态 180。再次释放开关 21 会使微控制器 U_2 进入断开/释放开关状态 182。随后接通开关 21 会使微控制器 U_2 按上述四种状态循环, 如图 7 所示。当计算机系统 10 处于正常操作状态 150 时, 微控制器 U_2 处于接通/释放开关状态 178。处于这种状态时会执行应用程序。在这种状态下, 计算机系统 10 可能会进入和离开待机状态 152。这种状态还对应于用户提出的挂起中止请求。断开/释放开关状态是对应于用户提出的挂起请求的开关状态。也就是说, 从系统处于关闭状态 156 开始, 按下并释放开关 21 一次会使计算机系统处于正常操作状态 150。再次按下并释放开关 21 可产生挂起请求, 该请求由管理例程所读取, 这将在图 9 的文字中予以详细地说明。在计算机系统 10 处于挂起状态 154 之前, 第三次按下并释放开关 21 会产生挂起中止请求, 该请求由挂起例程所读取。

参照图 19, 如果用户业已释放了按钮并试图关机, 那么, 微控制器 U_2 就会转至断电例程, 如 1184 所示。

另一方面, 如果按钮处于断开/按下状态, 这表示用于正接着按钮并试图关机, 那么, 微控制器就会在 1192 处测试开关是否被 BIOS 所屏蔽。BIOS 在进入待机状态的入口处屏蔽开关 21 以防止按下开关使计算机系统从待机状态转换成挂起状态, 从而如本文其它地方所述的那样防止用户搞混。

如果 BIOS 业已屏蔽了开关 21, 那么, 微控制器的代码就会转回任务 1176 并清除屏蔽位以便允许下一次按下开关使得计算机系统进入关机状态或挂起状态。另一方面, 如果未屏蔽开关 21 或者如果开关 21 不处于断开/按下状态, 微控制器就会在 1196 处执行心搏例程 (heartbeat routine)。

心搏例程用于向 CPU40 表明微控制器 U_2 正在正确地发挥着作用。微处理器的 $CMD_STATE\#$ 线的输出 (管脚 17, P1.4) 通常为逻辑 1。每 50 - 60 微秒微控制器 U_2 就会将该线置为逻辑 0 约达 1.5 微秒, 然后使之回到逻辑 1。由于 CPU40 所读取的电源管理状态端口是 $CMD_STATE\#$ 和 PRO_RDY 线的逻辑“与”, 所以 CPU40 能如此频繁地例如在系统引导时监控这种从 HIGH 到 LOW 及返回到 HIGH 的转换, 从而确保了微控制器 U_2 正确地起作用。

然后, 微控制器 U_2 在 1200 处测试 BIOS 是否业已命令断电。CPU40 可以存取并改变微控制器 U_2 内的每一个变量。如果 BIOS 例如在挂起期间系统的状态被写入硬盘驱动器 31 之后业已设置了表示应使计算机系统断电的变量, 那么, 微控制器 U_2 就会转至断电例程, 如 1204 所示。

另一方面, 如果 BIOS 未命令断电, 那么, 微控制器就会在 1208 处执行故障保险例程。故障保险计时器是一个 20 秒的计时器, 当电源 17 在 ± 5 和 ± 12 线处提供稳压的电力时, 该计时器是有效的。故障保险例程检查故障保险计时器是否已到时并且在该计时器业已到时时设置一个二进制位。如果 BIOS 命令重新启动故障保险计时器, 那么, 故障保险例程还会重新启动故障保险计

时器。

然后，在 1212 处，作为一种安全措施并且为了使微控制器与电源 17 同步，微控制器会检查 POWER_GOOD 线以便检测电源 17 是否仍在 ± 5 和 ± 12 线处提供稳压的电力。

如果电源 17 未在 ± 5 和 ± 12 线处提供稳压的电力，那么，微控制器 U_2 就会转至断电例程，如 1216 所示。另一方面，如果电源 17 正在 ± 5 和 ± 12 线处提供稳压的电力，那么，微控制器的代码就会转回 1180 并继续执行。

断电例程始于任务 1260 处。首先，微控制器 U_2 在 1264 处使活动中断无效以防止显示器不变空白。

其次，在 1268 处，微控制器检查 POWER_GOOD 线以检测电源 17 是否仍在 ± 5 和 ± 12 线处提供稳压的电力。如果电源 17 正在 ± 5 和 ± 12 线处提供稳压的电力，那么微控制器 U_2 就在 1272 处测试是否应该使显示器不显示与/或并断开 LED23。如果是这样的话，微控制器 U_2 就使视频控制器 56 停止产生视频信号和/或断开 LED23。

此后，如果 LED 和显示器均未不显示，则微控制器测试：(i) BIOS 是否业已通过设置一个二进制位命令计算机系统应回到开机，或者 (ii) 用户是否业已通过再次按下电源按钮 21 命令计算机系统应回到开机。如果产生上述情况中的任何一种，那么，计算机系统都将回到加电并且微控制器 U_2 会转至通电例程。如 1284 所示。

然后，微控制器确定在来自光隔离器 OPTO1 的 EXT_WAKUP# 线上是否有振铃出现。就 RS-232 线来说，这仅仅是

检查 EXT_ RTNG 线是否为 HIGH。对来自光隔离器 OPTO1 的信号来说, 这包括微控制器 U_2 所作的更多的检查工作。电阻 R_5 通常将 EXT_ WAKEUP# 线置为 HIGH。如果 Tip 和 Ring 两端的电压高于由阈值和保护部分 905 所设置的电压阈值例如在电话振铃时的 60V, 光隔离器 OPTO1 就会将 EXT_ WAKEUP# 线置为 LOW。但是, 测试电话线或电话线上的噪音都能满足上述条件。所以仅等待 EXT_ WAKEUP# 线为低可能使得虚假的“振铃”唤醒计算机系统。

因此, 微控制器通过测定振铃的频率来确定信号是否是振铃。标准的振铃是一在 16Hz 到 58Hz 之间的信号。微控制器 U_2 在 EXT_ WAKEUP# 信号的四个上升边缘之间测定三个时间周期, 如果这三个都对应于 15.1Hz 和 69.1Hz 之间的频率, 则微控制器认定在 EXT_ WAKEUP# 线上出现了适当的振铃并且设置一对应的二进制位。

EXT_ WAKEUP# 线上所检测到的 LOW 信号会启动检查例程。如果对三次接连的读取来说 EXT_ WAKEUP# 线都为 LOW, 那么, 微控制器 U_2 就会等待 EXT_ WAKEUP# 线在三次接续读取的时间内返回到 HIGH。紧接着读取构成计时器中断基础的 16 位计数器并将其值存储起来, 而且, 微控制器 U_2 在三次接连读取的时间内等待 EXT_ WAKEUP# 线转换到 LOW。微控制器然后测试头两个上升边缘之间的时间是否在 15 毫秒与 66 毫秒之间, 这表示信号在 15.1Hz 和 69.1Hz 之间。如果是这样的话, 再次对上述具有高分辨率的计数器进行取样, 微控制器在等待下一次 LOW 至 HIGH 转换时计算两次计数器取样之间的差

值。对 EXT_ WAKEUP# 线上下两次的 LOW 至 HIGH 转换重复上述过程。如果这三个时间周期全都在所说的范围内，那么微控制器 U₂ 就认定在 EXT_ WAKEUP# 线上出现了适当的振铃并设置一对应的二进制位。如果在 EXT_ WAKEUP# 线上不存在 LOW 或者上述三个时间周期中的任何一个超出了所说的范围，微控制器的代码会继续但不设置所说的二进制位。

然后，微控制器在 1286 处测试是否存在有振铃或者唤醒警铃的分钟数业已到时。对 RS-232 振铃、光隔离器振铃或唤醒警铃的分钟数来说，还包括微控制器 U₂ 测试是否设置了相关的二进制位。

如果存在振铃或唤醒警铃的分钟数到时，那么，计算机系统就会回到被加电，微控制器 U₂ 会转至加电例程，如 1287 所示。

此后，在 1288 处，微控制器测试电源 17 是否在 ±5 和 ±12 线处提供稳压的电力。如果未供电，则代码转回至任务 1280 并再次开始循环。另一方面，如果电源 17 在 ±5 和 ±12 线处提供稳压的电力，那么，微控制器 U₂ 在 1292 处执行心搏例程并在 1296 处执行故障保险例程。这两个例程已在任务 1196 和 1208 的文字中作了说明。

微控制器 U₂ 只在下列三种情况下才使得电源 17 停止在 ±5 和 ±12 线处提供稳压的电力：(i) BIOS 业已命令立刻断电，这是在通信例程中实现的；(ii) 故障保险计时器业已到时；或者 (iii) 用户按下电源按钮并且未设置微控制器 U₂ 的允许挂起标志，这是微控制器 U₂ 在每次读取 SWITCH 输入时所检测的条件。所以，微控

制器在 1300 处检测故障保险计时器是否业已到时。如果没有到时, 代码就转回任务 1280 并再次开始循环。

另一方面, 如果故障保险业已到时, 这表示计算机系统将要减少功耗, 那么, 如果有效的话, 微控制器 U_2 就会在 1304 处向 CPU40 生成一 SMI。这就能使 CPU 在假定计算机系统将要在此后立刻断电的情况执行某些任务。例如 CPU40 重新计算唤醒警铃值的更新的分钟数并将该分钟数写至微控制器 U_2 。

如果 CPU40 不执行其它活动, 则在可编程 SMI 计时器到时之后, 微控制器会使计算机系统断电。CPU40 可以通过将适当的值写至微控制器 U_2 重新启动 SMI 计时器, 从而能延长该时间间隔。

此后, 如果 1268 处的测试表明电源未提供合格的电力, 则微控制器 U_2 在 1308 处使计算机系统断电。这包括: (i) 使电源 17 停止在 ± 5 和 ± 12 线处提供稳压的电力; (ii) 使通信中断无效, 这是因为 CPU40 即将掉电; (iii) 将输出端口 (除 ON 端口) 置为 HIGH 以使它们的功耗减至最低 (在这种状态下, 微控制器 U_2 仍可读取 SWITCH, EXT_ RING、EXT_ WAKEUP 等); (iv) 设置断电变量以使得其余的例程知道已切断了通向计算机系统的电力; 以及 (v) 将开关的状态转换成断开/释放状态, 从而下一次按下按键将接通计算机系统。

此后, 代码转回至 1280 并开始再次循环, 以等待振铃、等待按下开关、等待 BIOS 命令该代码唤醒计算机系统或等待唤醒警铃的分钟数到时。

系统软件

业已说明了本发明之计算机系统 10 的硬件方面，以下说明其代码方面。

参照图 8，它显示了增加功耗例程的概况。当 CPU 转至复位矢量 (Reset Vector) 所指示的这一代码并执行该代码时，上述例程就在 200 处开始执行。每当 CPU 加电并且每当复位硬件信号复位了 CPU 或当通过转至上述复位矢量所指示的代码而执行复位指令时，上述情况就会出现。这种复位过程在本技术中是周知的。

首先，增加功耗例程的流程取决于为什么要使计算机系统增加功耗。正如图 11 所要详细说明的那样，计算机系统可能因节电或熄灭而要增加功耗。这样，使计算机系统保存在通电状态是不适合的。所以，增加功耗例程首先在 940 处确定计算机系统是否应处于通电状态。如果计算机系统不恰当地增加功耗，那么，CPU40 就在 942 处命令微控制器 U_2 使电源停止向计算机系统提供稳压的电力。

在确定计算机系统是否仍处于加电状态所进行的一种测试是在计算机系统响应微控制器认为是振铃而使系统加电的情况下确认电话线正在振铃。具体地说，在使计算机系统增加功耗之后，如果响应振铃而唤醒计算机系统，则在计算机系统等待硬盘驱动器 31 内的硬盘起转时，CPU40 会查询已完全加电的调制解调器 900 或 920 是否也检查到了振铃信号。如果未检测到振铃信号，系统就会减少功耗。如果调制解调器 900 或 902 也检测到了振铃信号，则计算机系统使引导过程延续。

假定计算机系统处于加电状态，则从总体上说加电例程的流

程取决于计算机系统是处于关机状态 156 还是处于挂起状态 154。即在 CMOS NVRAM96 内清除还是设置了挂起状态。如 202 处所示，计算机系统通过非易失性 CMOS 存储器 96 读取挂起标志来确定是处于关机状态 156 还是处于挂起状态 154。当计算机系统离开正常操作状态 150 到达关机状态 156 或挂起状态 154 时，每个例程都或者设置或者清除 NVRAM96 中的挂起标志。如果在 NVRAM96 中设置了挂起标志，则计算机系统 10 是处于挂起状态 154，并且将计算机系统 10 的状态存储到固定存储器 31 内。另一方面，如果在 NVRAM96 内清除了挂起标志，则计算机系统 10 是处于关机状态 156，并且不将计算机系统 10 的状态存储到固定盘存储器 31 内。这样，如果在 NVRAM96 内设置了挂起状态，则计算机系统就会执行“正常”的引导例程，如任务 204 - 210 所示。如 204 处所示，第一项任务是通电自检 (POST)，这将在伴随图 11 中的文字中给予详细的说明，从 POST 返回之后，CPU40 调用 PBOOT 例程以加载操作系统，如 206 处所示。

PBOOT 例程是一种在 IBM 计算机上运行的典型例程，但略有变化，这将下文中予以说明，PBOOT 确定从哪里（是从硬盘驱动器 31 还是从软盘驱动器内的磁盘上）引导并加载操作系统，而操作系统则按 CONFIG. SYS 的指令去分析和实现系统的变化并最终执行 AUTOEXEC. BAT 批处理文件。PBOOT 例程在本技术中是周知的。OS 加载 APM 设备驱动程序，该驱动程序向 BIOS 查询 APM 是否认识该 BIOS。如果是这样的话，BIOS APM 例程和 OS APM 例程进行信号交换，此后共同提供本文所述的各种功能。如 210 处所示，操作系统根据用户的指示不定地执行代码。

但是, 向 APM 报告管理例程的结果是 APM BIOS 和 APM OS 使得管理例程与正在运行的程序“并行”地执行, 如 212 处所示。也就是说, 计算机系统 10 是时间多路复用式的多任务系统, 并且可周期性地运行 APM Get Event, 从而能周期性地运行管理例程。最终的结果是, 约每秒钟执行管理例程一次。图 9 中的文字将详细说明管理例程。正常的引导例程 204 - 210 结果之后, 计算机系统 10 处于正常操作状态 150, 如伴随图 4 的文字所述。

参照任务 202, 如果在 NVRAM96 内设置了挂起标志, 则将系统状态保存至硬盘驱动器 31, 并且计算机系统 10 执行继续引导例程, 如 214 - 220 所示。首先, 计算机系统执行简化的 POST, 如 214 处所示。伴随图 11 的文字将详细说明简化的 POST。在简化的 POST 之后, 计算机系统调用继续例程, 如 216 处所示。伴随图 12 的文字将详细说明继续例程。这里只说明继续例程将计算机系统 10 的状态恢复成挂起之前的配置就够了。与任务 204 - 210 所示的正常引导例程不同, 继续引导例程不必向 APM API 报告存在有管理例程, 这是因为, APM 例程必须运行才能使系统挂起, 并且, 在恢复系统状态时, APM 被装回内存。这样, 在继续例程完成恢复计算机系统 10 的状态时, APM 已处于适当的位置并以与所恢复的代码相“并行”地运行, 如 212 和 210 处所示。继续引导例程 214 - 220 结束之后, 计算机系统 10 处于正常的操作状态 150, 如伴随图 4 的文字所示。因此, 执行了正常的引导例程 204 - 210 或继续引导例程 214 - 220 之后, 计算机系统 10 会处于正常的操作状态 150。

图 9 是显示管理例程详细内容的流程图, 该管理例程在“Get

Event”过程中约每秒钟都由 APM 所调用。不同的操作系统以不同的频率 Get Event。

管理例程始于图 9 中的 222 处。以下内容假定计算机系统 10 始于正常的操作状态 150。第一项任务是在 224 处测试用户是否按下了开关 21。正如与图 6A 和图 7 有关的内容所详细说明的那样,通过 CPU 40 查询微控制器而测试开关 21。

如果任务 224 处的测试表明用户按下了开关 21,则管理例程在 950 处继续确定是否先前向 OS 中的 APM 设备驱动程序发出了挂起请求。

如果任务 950 处的测试表明还未将挂起请求发送给 APM 驱动程序,则管理例程在 226 处向 OS 的 APM 设备驱动程序发出“挂起请求”,然后在 228 处返回至 APM 驱动程序。APM 驱动程序响应连接“挂起请求”的 APM 返回代码而广播即将进行的挂起,从而 APM 可识别的设备可以执行任何必要的系统任务(如使硬盘同步等),然后,APM 驱动程序发出“挂起命令”,该命令使得 APM BIOS 路径选择例程调用挂起例程。与图 10 有关的内容将说明挂起例程。挂起例程基本上使计算机系统 10 离开正常操作状态 150 并进入挂起状态 154,而且可将控制权在若干条指令(如果计算机系统尚未准备好挂起)或若干分钟、若干小时、若干天、若干周或若干年(如果计算机被挂起并且被继续)之后交还给管理例程。无论是挂起程序在没有挂起的情况下返回还是在完成挂起和继续之后返回,挂起程序总是会设置“正常继续”的 APM 返回代码。

通常在任务 224 处未按下开关 21,因而管理例程会转至任务

952 以确定是否设置了临界挂起标志。同样, 如果先前向 OS 中的 APM 驱动程序发送了挂起请求, 则管理例程这时也会转至任务 952 以确定是否设置了临界挂起标志。如果设置了临界挂起标志, 则管理例程会继续在 954 处测试是否先前向 APM 驱动程序发出了临界挂起请求。

如果未向 APM 驱动程序发出临界挂起请求, 则管理例程在 956 处引发临界挂起请求 APM 返回代码, 然后在 958 处返回至 APM 驱动程序。APM 驱动程序响应临界挂起请求而在不广播即将进行挂起的情况下立即挂起计算机系统, 所以, APM 所认识的设备不能执行各自的预挂起任务。

如果在 952 处未设置临界挂起标志或者在 954 处已向 OS 中的 APM 驱动程序发送了临界挂起请求, 则管理例程在 957 处确定挂起未决是否超过 15 秒。如果是这样的话, 管理例程在 958 处设置临界挂起标志, 从而在下一次 APM Get Event 过程中进行任务 954 处的测试。

此后或者如果挂起未决不超过 15 秒, 管理例程就在 959 处检查挂起是否是未决的。如果是这样的话, CPU 40 就在 960 处使微控制器 U₂ 重新启动(复位)故障保险计时器以及 APM 故障挂起计时器。

此后或者如果挂起不是未决的, 则管理例程转至任务 230 以确定计算机系统是否刚被继续。如果调用了挂起例程, 那么, 不管挂起例程是在没有挂起的情况下返回, 还是在完成挂起与继续之后返回, 计算机系统认为自己刚刚继续。在 230 处测试这种继续, 并且, 如果计算机系统刚刚继续(或者因 DMA 或文件活动未执

行挂起), 则在 232 处发布 APM 返回代码“正常继续”并在 234 处返回至 APM。据此, APM OS 驱动程序更新系统时钟以及在暂停期间可能变为失去时效的其它值。

通常, 计算机系统 10 不是刚刚继续的, 因而管理例程转至任务 236 以测试任何用户活动。在任务 236 处测试三种类型的用户活动: 硬文件 31 的活动、键盘 12 的活动以及鼠标器 13 的活动。每当有 APM Get Event 时, 管理程序都作以下工作: 从硬盘驱动器 31 读取用于硬文件磁头、柱面和扇区的值; 查询微控制器 U_2 是否在鼠标器中断线或键盘中断线上存在有表示用户活动的活动; 以及从实时时钟 98 中读取分钟数值, 该数值在 0 分至 59 分的范围内并在每小时开始时返回为 0 分。暂时将硬盘驱动器的三个活动变量 (磁头、柱面以及扇区) 以及上述分钟数值存储起来。然后比较这三个硬盘驱动器活动变量和来自先前 Get Event 所保存的硬盘驱动器活动变量。如果当前的三个硬盘驱动器值与来自先前 Get Event 的值相同并且如果在鼠标器中断或键盘中断方面没有活动, 则就是不存在有用户的活动。如果所述硬盘驱动器的值是不相同的, 或者在鼠标器中断或键盘中断方面已存在有活动, 那么, 就是有了用户的活动, 因而将当前的硬盘驱动器活动变量保存起来以便用来与下次 Get Event 过程中所读取的值作比较。

上述活动检测方案是在 CPU 中运行一个例程以确定硬盘驱动器的活动, 并且只监控两个硬件中断的活动。另外, 也可以按硬件方式专门监控活动。例如, 可以监控所有 16 条硬件中断线的活动。

如果存在有活动, 则管理例程在 238 处通过测试待机标志来

确定计算机系统 10 是否处于待机状态 152。如果设置了待机标志,这表示计算机系统 10 处于待机状态 152,那么,管理例程就在 240 处退出待机状态 152 并进入正常操作状态 150。管理例程通过使在进入待机状态 152 时减少功耗的设备增加功耗而退出待机状态 152,如图 18 所示。简要地说,当计算机系统退出待机状态 152 时,管理例程作下列工作:恢复视频信号;使硬盘驱动器 31 内的硬盘起转;恢复系统时钟;使 APM CPU 空调用无效,从而使得来自 APM 驱动程序的 CPU 空调用不再使 CPU 40 停止;以及清除指示计算机系统 10 处于待机状态 152 的标志。

此外,如果存在有活动,则还将来自实时时钟 98 的分钟数值保存起来以便用来与后续的 Get Events 过程中所读取的分钟数值作比较。保存当前分钟数值实际上是在 241 处复位不活动性待机计时器和不活动性挂起计时器。在正常使用过程中,存在有用户的活动,并且,管理例程会在 242 处设置“*No Event*”(无事件) APM 返回代码并在 243 处返回至 APM 调用代码。APM 响应“*No Event*”返回代码而不调用任何例程。

如果任务 236 处的测试表明不存在有用户的活动,则管理例程分别在 245 和 247 处测试不活动性待机计时器以及不活动性挂起计时器是否到时。如果计算机系统 10 处于待机状态 152,则不检查不活动待机计时器的到时情况,相反,在任务 244 处跳过上述测试。

通过从所保存的分钟数值中减去当前的分钟数值以获得与自从有用户活动以来的分钟数值相对应的值,从而可以检查上述两个计时器的到时情况。所说的值在 245 处与不活动待机暂停值

作比较,并在 247 处与不活动挂起暂停值作比较。上述两个暂停值可由用户选择并且能被设置成使计算机系统因两个计时器中的一个到时而不会进入待机状态 152,不会进入挂起状态 154,或者不会进入待机状态 152 中挂起状态 154。将上述任一暂停值设置为零 (0) 表示相应的计时器不会到时。

如果自从最近的用户活动以来的分钟数等于或大于不活动待机暂停值,则管理例程会在 246 处使计算机系统 10 进入待机状态 152。如果不活动的待机计时器尚未到时,则管理程序在 247 处测试不活动挂起计时器的到时情况。另一方面,如果不活动待机计时器业已到时,则管理例程通过使某些组件处于各自的低功率方式而使得计算机系统 10 进入待机状态 152,如图 18 所示。简要地说,在所述的最佳实施例中,管理例程做下列工作:使视频信号不显示;使硬盘驱动器 31 中的硬盘停转;减慢系统时钟;使 APM CPU 空调用有效,因而来自 APM 驱动程序的 CPU 空调用会中止 CPU 40;以及设置表示计算机系统 10 处于待机状态 152 的标志。在使计算机系统 10 进入待机状态 152 之后,管理例程在 247 处测试不活动挂起计时器的到时情况。

管理例程在 247 处测试不活动挂起计时器是否业已到时。如果自从最近的用户活动以来的分钟数等于或大于不活动挂起的暂停值,则管理例程在 248 处设置“挂起请求”APM 返回代码,然后在 243 处返回至 APM。正如以上与任务 226 有关的内容所说明的那样,APM 响应设置“挂起请求”APM 返回代码而执行任何必要的系统任务,然后调用挂起例程。挂起例程将在与图 10 有关的内容中更全面地予以说明,简要地说,挂起例程使计算机系统 10

离开正常操作状态 150 并进入挂起状态 154。正如与任务 226 有关的内容所说明的那样,挂起例程无论是否使计算机系统 10 挂起的情况下都能将控制权交还给管理例程。另一方面,如果不活动挂起计时器尚未到时,则管理例程在 242 处设置 APM 返回代码“*No Event*”并在 243 处返回 APM 调用代码。

尽管最经常的是将 APM 返回代码“*No Event*”返回给 APM,但也可以使各种其它事件返回给 APM。然而,对于每个 APM Get Event 来说,只能指定一个 APM 返回代码。例如,在进入待机状态 152 之后,将“*No Event*”返回给 APM。在离开挂起状态 154 之后,将 APM 返回代码“*Normal Resume*”(正常继续)返回给 APM。为 APM 而排队的特定报文取决于计算机系统的精确特征。管理例程也可以返回 APM 返回代码“*Normal Resume*”或“*Suspend Request*”。

参照图 9B,它显示了 APM 处理最近请求例程 (APM Working On Last Request Routine),此例程始于 916 处。BIOS APM 例程响应所发出的 APM 处理最近请求 (APM Working On Last Request) 而在 962 处重新启动微控制器 U_2 中的 APM 故障挂起计时器以及故障保险计时器、在 963 处重新启动 15 秒的挂起未决计时器,以便在 OS APM 等待计算机为挂起作适当准备时阻止发出临界挂起请求并且在 964 处返回。

参照图 9C,它显示了 APM 拒绝最近请求例程 (APM Reject Last Request Routine),此例程始于 965 处。BIOS APM 响应所发出的 APM 拒绝最近请求 (APM Reject Last Request) 而在 966 处重新启动微控制器 U_2 内的 APM 故障挂起计时器以及故障保险

计时器、在 967 处设置临界挂起标志以便强制立刻挂起并在 968 处返回。

利用对挂起例程的认识可以更好地理解增加功耗和继续例程。所以，应认识到最好按下列次序检验对 APM BIOS 例程的说明：对本发明加电例程的概略说明（如以上图 8 所述）、对管理例程的详细说明（图 9）、对本发明挂起例程的详细说明（图 10）、对本发明增加功耗过程的详细说明（图 11）、对本发明继续例程的详细说明（图 12）、对保存 CPU 状态例程的详细说明（图 13）、对恢复 CPU 状态例程的详细说明（图 14）以及对保存 8259 状态例程的详细说明（图 15）。

应该认识到，虽然因多数例程是相互作用并且挂起/继续过程就是一种连续的循环操作，因而对本发明之计算机系统 10 的任何说明都略微有些循环，但是，最有帮助的还是在引导例程（图 11）或继续例程（图 12）之前说明挂起例程（图 10）。参照图 10，它显示了挂起例程的流程图。回忆一下，在执行了正常引导例程 204 - 210 或继续引导例程 214 - 220 之后，计算机系统 10 处于正常操作状态 150。而且，如以上与图 8 有关的内容所说明的那样，无论是正常引导计算机系统 204 - 210 还是继续引导计算机系统 214 - 220，在这两个例程中任何一个例程结束之后，APM OS 驱动程序都会认识诸如管理例程之类的 APM BIOS 例程，如图 8 所示。因此，APM 约每一秒钟都会轮询管理例程。

图 10 显示了挂起例程，该例程始于 250 处。APM 响应管理例程向 APM 返回 APM 返回代码“Suspend Request”而调用挂起例程。此外，当计算机系统执行如与图 17 和图 18 有关的内容所详

细说明的检查点 (Checkkpoint) 时, 调用并部分地执行挂起例程。首先, 挂起例程的流程在 970 处取决于 CPU 40 是否是带有 SMI 的 S 部件。如果是这样的话, CPU 40 在 972 处使微控制器 U_2 给 CPU 40 生成一 SMI。正如本技术的专家所周知的那样, CPU 40 内的微码会在 974 处将 CPU 40 的状态保存至 E000H 段的数据结构内。

另一方面, 如果 CPU 40 不是带有 SMI 的 S 部件, 则调用保存 CPU 状态例程, 如 252 处所示。与图 13 有关的内容将会详细说明保存 CPU 状态例程。这里, 只说以下内容就足够了: 在初始调用挂起例程时, 无论 CPU 40 处于什么模式, 都会按 CPU 40 处于实模式来执行挂起例程的其余部分, 所以能够在不必担心产生任何错误的情况下执行上述挂起的其余部分, 而所说的错误则是由试图执行容许地址空间之外的指令或试图执行特权指令而引起的。

保存 CPU 状态例程在 253 处以独特的方式将程序控制权交还给挂起例程。从保存 CPU 状态例程“返回”至挂起例程指的是复位 CPU, 这将在以下与图 13 中任务 630 和 632 有关的内容中予以详细说明。挂起例程的重要细节是业已将 CPU 寄存器写至了 E000H 的数据结构, 并且, CPU 40 当前处于实模式。

在保存 CPU 状态例程返回之后, 或者中 CPU 响应 SMI 而保存了自己的状态之后, 挂起例程继续在 254 处确认是否已按下了开关 21。如与图 6 及图 7 有关的内容所说明的那样测试开关 21 的接通情况。如果未按下开关, 则正在进行的挂起是软件挂起并在 CMOS NVRAM 96 中设置软件挂起标志。这就确保了软件挂起不会与因开关接通而引起硬件挂起相混淆。所有的软件挂起都会通过在微控制器 U_2 内设置一二进制位而转化为硬件挂起。将软

件挂起转化为硬件挂起之后,接通开关会中止该挂起。

下一步的任务是在 E000H 段内建立一个堆栈,如 262 处所示。

建立起堆栈之后,挂起例程在 264 处检验 DMA 控制器 72、磁盘适配器 84 以及 IDE 盘控制器 86 以了解当前是否正在分别进行任何的 DMA 传送、软盘驱动器传送或硬文件传送。如果是这样的话,就不能进行挂起,这是因为,这三种类型所特有的性质会阻止进行令人满意的挂起。例如,如果正在进行来自硬盘驱动器 31 的硬文件传送,则 IDE 控制器业已读取了数据,但数据尚未被传送至系统内存 53。CPU 无法充分地存取上述数据,所以,如果在硬文件读操作中间挂起计算机系统,数据就会丢失。因此,如果正在进行上述三种传送中的任何一种,都推迟挂起直至下一次的 APM Get Event,那时,会再一次检测 DMA 和软盘控制器的活动。

因此,必须将在 352、260 和 262 处执行的任务颠倒,从而将控制权转给 APM。首先, BIOS 从读/写变为只读,如 265 处所示。这是通过关闭仍包含有投影数据的 E000H 段而实现的。弹出并恢复在任务 262 处创建的堆栈。最后,在 267 处将控制权转回给 APM 之前,在 266 处由恢复 CPU 状态例程恢复 CPU 的状态。在下次 Get Event 期间 APM 再过约一秒钟再次轮询挂起例程。到那时,可能完成了阻止挂起的传送活动,从而使挂起继续。

参照任务 264,如果当前没有进行 DMA 传送,软盘驱动器传送或硬文件传送,则可以进行挂起。挂起例程在 268 处继续。使故障保险计时器连续地递减计数,当开关 21 处于断开/释放状态

时，故障保险计时器如果到时则会使计算机系统自身关闭。所以，第一项任务就是复位与图 6A 和图 19 有关的内容所述的故障保险计时器，如 268 处所示。

然后，在 270 处保存 8042 协处理器 104 的状态。8042 协处理器 104 的寄存器在本技术中是周知的。CPU 40 可以直接读取上述寄存器并且这些寄存器的值可以直接写入 E000H 的数据结构中。

随后，在 272 处保存 8259 中断控制器 92 的状态。挂起例程调用 8259 保存状态例程，在与图 15 有关的内容中将详细说明上述 8259 保存状态例程。这里，只说下列内容就够了：尽管 8259 中断控制器 92 的某些寄存器是只写的，但 8259 保存状态例程能够确认两个 8259 中断控制器的无名寄存器的内容。该寄存器的值可以直接写至 E000H 的数据结构中。

在保存了中断控制器 92 的状态之后，必须将中断控制器 92 的配置改变成已知的状态以使得挂起例程执行各个中断驱动的任务的正常功能。所以，在 274 处交换 BIOS 数据区与矢量表 (BIOS data Areas & Vector Table)。挂起例程将 0000H 段中的当前状态 BIOS 数据区和矢量表的内容拷贝至 E000H 段中的存储单元。然后，将已知状态 BIOS 数据区和矢量表的内容从 E000H 段中的数据结构拷贝至 0000H 段内的存储单元。在引导例程的任务 414 中将已知状态 BIOS 数据区和矢量表拷贝至 E000H 段，引导例程如图 11 所示并在以下予以说明。最后，将当前状态 BIOS 数据区和矢量表从 0000H 段拷贝至 E000H 段内的数据结构。当 274 处的例程结束时，诸如中断 13H (磁盘读/写中断) 和中断 10H (视频存取中断) 之类的所有中断会象所希望的那

样起作用。

然后，在 276 处将计时器 102 的状态保存起来。该计时器的寄存器在本技术中是周知的。CPU 40 可以直接读取所有的这些寄存器，并且，这些寄存器的值可直接写入 E000H 的数据结构。在 276 处还将 IDE 磁盘控制器 86 的状态保存起来。IDE 磁盘控制器 86 的寄存器在本技术中是周知的。CPU 40 可以直接读取所有的这些寄存器，并且，这些寄存器的值可直接写入 E000H 的数据结构。

下一步是为将系统内存写至硬盘驱动器 31 的挂起文件 (Suspend File) 而做好准备。系统内存包括系统 RAM 53 (它包括主内存和任何扩展内存) 以及视频内存 58。这时，部分 RAM 53 可以处于外部高速缓冲存储器 60 内。在 286 处倾泻 CPU 的高速缓冲存储器，这将在以下与图 13 有关的内容中予以说明。然后，在 286 处倾泻外部高速缓冲存储器并使该存储器提高向硬盘 31 的写的速度。

在计算机系统 10 内执行的代码可能已将 IDE 控制器 86 置入了未知的状态。因此，下一步是在 292 处将 IDE 控制器 86 初始化成已知的状态。这是通过把值直接写至 IDE 控制器 86 内的寄存器而实现的。

然后，在 976 处启动中断驱动的并行线程以读取任何调制解调器的状态并使该状态保存至 E000H 的数据结构中。该例程做以下工作：捕捉对应于 COMM 端口的中断，而 COMM 端口则与特定的调制解调器有关；将命令传递给上述调制解调器以使该调制解调器顺序地传回该调制解调器之寄存器的内容；接收从上述

调制解调器所传送的寄存器内容；以及将上述寄存器的值保存至 E000H 的数据结构中。这一例程将一第一命令传送给调制解调器，然后以中断驱动的方式响应，以便接收调制解调器的应答并响应每个 COMM 端口的中断将下一个命令传递给调制解调器，直至保存了所有调制解调器的寄存器。如果不是作为并行线程运行的，则这一例程给挂起计算机系统所花费的时间增加几秒钟（每个调制解调器为 3 - 5 秒，这取决于特定的调制解调器以及当前的波特率）。由于是中断驱动的并行线程，所以，如果该例程在已将系统状态写至硬盘驱动器 31 之前结束运行，则该例程不给挂起增加时间。

在上述中断驱动并行线程的调制解调器保存例程启动之后，必须在 294 处使挂起文件处在硬盘驱动器 31 的固定盘上。挂起文件的磁头、扇区和柱面均存储在 CMOS 存储器 96 中。一旦确定了挂起文件的位置，就读取该文件的大小与特征。在所说的最佳实施例中，上述特征是任意长度的 ASCII 码，它表示存在有挂起文件。也可以用其它方法实现所说的特征，例如使用二进制串，该串具有能在硬文件系统上被随机查找到的非常低的概率。

取读了挂起文件的文件大小与特征之后，下一步就是在 296 处确认该特征和文件大小是否正确。如果所说的特征不正确，这表明其它程序已修改了挂起文件，或者如果所说的文件大小是不正确的，这表明挂起文件的大小已被修改过，那么，挂起例程就在 298 处调用致命挂起错误例程 (Fatal Suspend Error Routine)，此例程始于图 13 的任务 652 处。如果用户按下开关 17，以退出致命挂起错误例程，程序控制权从任务 299 转至任务

506。

另一方面，如果所说的特征是正确的并且挂起文件足够大，则挂起例程开始将计算机系统的状态写至内存。

在将计算机系统 10 的状态写至硬盘驱动器 31 之前，CPU 40 在 297 处命令微控制器 U_2 重新启动（复位）故障保险计时器并查询微控制器 U_2 以确定是否再次按下了开关 21。如果没有再次按下开关 21，则挂起例程继续。另一方面，如果再次按下了开关 21，则中止挂起。可以在挂起例程内的若干点处重新启动故障保险计时器并测试开关 21 的接通情况。任务 297 仅仅是说明性的，这项应用技术中普通的电路设计者能够确定重新启动故障保险计时器的次数及其间容许的时间。在故障保险计时器到时以使电源 17 “关闭”之前，挂起例程应复位故障保险计时器。同样，应不时检查一下开关 21。如果再度按下开关 21，这表示用户希望中止挂起，那么，上述代码应转至继续例程中的适当点处以便“不进行挂起”并从部分挂起中恢复。

与此类似，在 350 处按下 Ctrl - Alt - Del 键会中止挂起。按下 Ctrl - Alt - Delete 键（同时按下 Control 键、Alt 键和 Delete 键）是复位以 IBM BIOS 和 Intel 80X86 系列 CPU 为基础的一般计算机系统的周知方法。计算机系统 10 用本技术中周知的 BIOS 一号中断处理程序来处理 Ctrl - Alt - Del。计算机系统 10 在 350 处带有略有改变的 1 号中断处理程序，该处理程序在 352 处清除 CMOS 存储器 96 中的挂起标志并在 354 处复位时转至引导例程。

在本发明的计算机系统 10 中，挂起例程正在运行时按下

Ctrl - Alt - Del 会使得计算机系统进入关机状态。这是因为，在开关 21 接通之后，按下 Ctrl - Alt - Del 会调用引导例程，而引导例程则会将微控制器 U_2 初始化成这样的状态：故障保险计时器业已到时且开关仍处于断开/释放状态。因此，在处于挂起例程时按下 Ctrl - Alt - Del 会使计算机系统 10 进入关机状态。

参照任务 300，再次使挂起文件处在硬盘驱动器 31 上，将特征短语在 300 处写至挂起文件的第一字节。然后，在 302 处将 E000H 段中的整个 64K 字节的数据写入挂起文件。E000H 的上述 64K 拷贝实际上只是一个位置保留区并且将会在挂起例程结束时重写这一相同位置。

然后，在 303 处将视频控制器 56 的状态保存起来。视频控制器 56 的寄存器在本技术中是周知的。CPU 40 可以直接读取上述所有的寄存器，而且，这些寄存器的值可直接写至 E000H 中的数据结构内。

随后，将系统内存写至挂起文件，这是由一双缓冲区系统来实现的，该双缓冲区系统从系统内存中读取数据、并压缩该数据且将其写至 E000H 段，最后将已压缩的数据从 E000H 段写至挂起文件。在一时间多路复用结构中两个例程在工作：一个压缩数据并写至 E000H 段，另一个写至挂起文件。前者在前台运行，后者则是在后台运行的中断驱动例程。很明显，由于只有一个 CPU 40，因此在给定的时间内只能执行一个例程，但是，由于后一个例程是由中断驱动的，所以该例程在需要时可中断前一个例程的运行，使向挂起文件传递数据的速度最佳。上述两个缓冲区的每一个都为 8K 字节长，这被认为是可使向硬盘驱动器 31 的传

递时间最佳化。

上述过程始于 304，它读取足够的数据并进行压缩且写至 E000H 段以填充第一个 8K 的缓冲区。利用持续长度编码法压缩所说的数据，但是，也可以使用任何适当的压缩法。这时，在 306 处启动总体上在标号 307 处所示的写自缓冲区例程 (Write from Buffer Routine)。写自缓冲区例程 307 是一种在后台运行的中断驱动例程，包括任务 308 至任务 310。总体上在标号 311 处所示的压缩例程包括任务 312 至任务 318 并且是一种前台例程。首先，写自缓冲区例程 307 在 308 处将刚被任务 304 所填写的缓冲区写至挂起文件。当写自缓冲区例程 307 将缓冲区的内容写至挂起文件时，压缩例程 311 在 312 处继续从系统内存中读取后面的字节并压缩它们且将压缩后的数据写至前述两个 8K 缓冲区中的另一个内。一旦压缩例程 311 用压缩后的数据填满了上述缓冲区，下一步就在 314 处确定是否已压缩了整个系统内存。

IDE 控制器 86 不能很快地将数据写至硬盘驱动器。因此，压缩例程 311 总是会在写自缓冲区例程 307 完成将 8K 缓冲区写至硬盘驱动器 31 之前完成填满未被写至硬盘驱动器 31 的 8K 缓冲区。所以，压缩例程 311 必须等待写自缓冲区例程 307 完成将缓冲区写至硬盘驱动器 31。如果压缩例程 311 尚未完成压缩所有的系统内存并对它们进行写操作，则压缩例程 311 在 316 处等待写自缓冲区例程 307。压缩例程 311 和写自缓冲区例程 307 通过一组标志而相互通信。写自缓冲区例程 307 在完成将当前缓冲区写至挂起文件时会随后转换该缓冲区的标志，以便向压缩例程 311 表明它可以用压缩后的数据填写刚被写至挂起文件的缓冲区。然

后, 在 309 处复位故障保险计时器 C_2 并以与任务 297 有关的内容所述的方式检查开关 21 的接通事件。

然后写自缓冲区例程 307 在 310 处确定刚写至挂起文件的缓冲区是否是最后一个要写的缓冲区。如果不是, 则写自缓冲区例程将刚由压缩例程 311 所填写的缓冲区写至挂起文件。同时, 压缩例程 311 通过检验缓冲区标志而确定是否为另外压缩的系统内存准备好了缓冲区。也就是说, 压缩例程在 316 处等待直至写自缓冲区例程处理完当前的缓冲区, 这时在 312 处继续压缩循环。请注意, 如果支持线性帧缓冲技术, 则可压缩视频内存 58, 但对 VESA 页存取来说却不压缩视频内存 58。相反, 视频控制器 56 利用 VESA 调用读取 VESA 页存取视频内存并且在不用以上详细说明的双缓冲区系统进行压缩的情况下写 VESA 页存取视频内存。

一旦压缩例程 311 完成了压缩所有系统内存, 该例程就会在 318 处等待写自缓冲区例程 307 完成将最后的缓冲区写至挂起文件。一旦写自缓冲区例程 307 结束, 该例程就会从 310 转至 318 并不再存在。这时, 没有后台例程在运行并且主程序在 320 处继续。

然后, 在任务 320 处, 将 DMA 部件 71 (DMA 控制器 72 和中央仲裁器 82)、82077 软盘控制器 84 以及 RS-232 UART (通用异步收发器) 94 的状态保存起来。这些设备均具有在本技术中周知的寄存器。软盘控制器 84 和 UART 94 内的所有寄存器均可由 CPU 40 直接读取, 并且, 这些寄存器的值可直接写至 E000H 内的数据结构中。DMA 部件不带有可读寄存器。相反, 一般在每次

DMA 传送之前都建立起只写寄存器。为此，如果正在进行 DMA 传送，则挂起例程停止挂起。

随后，挂起例程在 978 处测试与任务 976 有关的内容所述的中断驱动的调制解调器状态例程是否结束。如果没有结束，则挂起例程等待上述调制解调器状态例程结束。

应该认识到，一旦计算机系统 10 进入挂起状态 150，就最好能检测到对挂起文件的任何修改。例如，对某些人来说，可能会生成一修改过的挂起文件并将该挂起文件传送至硬盘驱动器 31 且试图使计算机系统 10 恢复成与所保存的状态不同的状态。为此，将一伪随机值放至到 E000H 段的数据结构中。如 328 处所示，在中断驱动的调制解调器状态保存例程结束之后，从高速计时器 102 之一中读出一 16 位的时标。然后将该时标写至 E000H 段的数据结构中。

然后，通过在不考虑进位的情况下将 E000H 中的各个 16 位字加在一起而计算出用于整个 E000H 段的 16 位校验和。该校验和在 330 处写至 E000H 段的数据段中并在 332 处写至 CMOS NVRAM 96 中。此后，在 334 处将所有的工作变量从 CPU 40 写至 E000H 段的数据结构中，并且，在 336 处将整个 E000H 段从挂起文件的特征短语之后开始（紧接在该特征之后）重新写至挂起文件。然后，在 338 处设置 CMOS NVRAM 96 内的挂起标志，以便通知计算机系统 10 已将该计算机系统的状态保存到了挂起文件内。

随后，挂起例程在 980 判定是否已采用了检查点。如果是这样的话，计算机系统就不应降低功耗，反之，计算机系统必须继续

到从刚刚进行的部分挂起中恢复的程度。所以，如果已采用了检查点，挂起例程在 982 处转至继续例程的任务 484 处，然后，继续例程进行部分的继续。

如未采用检查点，则 CPU 40 通过命令微控制器 U_2 将 ON 信号设置成逻辑 0 而“关闭”电源，从而使电源 17 的初级/稳压部件 172 停止沿 ± 5 和 ± 12 线提供稳定的电压。该电压要花几秒钟的时间才能递减至约为 0，从而给 CPU 40 以时间去执行多条命令。所以，在 342 处，CPU 40 在等待电源 17 所产生的系统电压减小时会作无限的循环（“旋转”）直至 CPU 40 停止起作用。

参照图 11，它显示了引导例程的详细内容。与图 8 有关的内容概述了引导的过程。当 CPU 40 转至复位矢量 (Reset Vector) 所指的代码并执行该代码时，引导例程就在 380 处开始。每当 CPU 40 增加功耗并且无论何时 CPU 40 因转至上述复位矢量所指的代码时，都会出现上述情况。复位的过程在本技术中是周知的。

第一项任务是在 382 处测试 CPU 40 并初始化内存控制器 46。CPU 是由 POST 例程来测试的。CPU 的测试部分确定 CPU 40 是否是带有 SMI 的“S”部件。如果是这样的话，就设置一标志以指示这一情况。POST 例程还初始化内存控制器 46。

然后，引导例程在 986 处测试微控制器 U_2 是否在起作用。为此，CPU 顺序地读取电源管理电路 106 的状态端口并等待该端口处从 HIGH 到 LOW 以及从 LOW 回到 HIGH 的转换。这种转换说明微控制器 U_2 的心搏正在起作用，所以，CPU 40 可以在假定微控制器 U_2 按希望的那样起作用的情况下继续进行引导过程。

如果 CPU 在预定的时间如一秒或两秒内没有检测到状态端口处的转换, 则微控制器 U_2 就不具有心搏, CPU 40 就会在 988 处命令第一电路 PAL U_1 复位微控制器 U_2 , 如前所述。然后, CPU 40 在 990 处再次等待状态端口处从 HIGH 到 LOW 的转换。如果 CPU 在一秒或两秒的时间内又没有检测到状态端口的转换, 则微控制器 U_2 不具有心搏, 因而 CPU 40 会于 922 处在假定微控制器 U_2 处于不能复位的状态的情况下使本文所述的电源管理特征无效。

另一方面, 如果微控制器 U_2 正在起作用, 则 CPU 40 在 994 处刷新微控制 U_2 内唤醒警铃值的分钟数。RTC 98 的时基要比微控制器 U_2 的时基精确得多。所以, 为了在不给微控制器 U_2 增加更精确因而是更昂贵的时基的情况下克服上述局限性, BIOS 使上述不太精确的时基与上述更为精确的时基相同步并在每次系统引导时用来自 RTC 98 的更为精确的值更新微控制器 U_2 内唤醒警铃值的分钟数。为了做到这一点, CPU 40 从 CMOS 存储器 96 中读取警铃的绝对日期和时间并计算唤醒警铃值的分钟数且将该分钟数写至微控制器 U_2 。

此后并且如果微控制器 U_2 不起作用从而使所说的电源管理特征无效, 则引导例程在 996 处确定是否因电源 17 的加电而引导了计算机系统。最佳的是, 总是有交流电加载于电源 17 的初级/稳压部件 172 上, 并且, ON# 输入控制着对 ± 5 和 ± 12 线处的电压调节。这样电源 17 可以不断向使电源管理电路 106 通电所需的 AUX5 供电, 并且, 电源管理电路 106 在不转换交流电的情

况下控制电源 17。

但是，正如本技术的专家所周知的那样，某些用户喜欢用开关式配电板(未显示)来对计算机系统加电，从而用一个单一的开关接通和断开整个计算机系统的交流电。这对电源管理电路 106 来说就存在着问题，因为，微控制器 U_2 和其它设备均被配置成要由 AUX5 电源线不间断地供电。所以，计算机系统必须具有能确定它已因接通了交流电而被加电并且相应地活动的方法。

但是，AUX5 线还会遇到上述电压降低和断电的情况。在出现电压降低和断电的情况之后，复位子电路 920 会复位微控制器 U_2 以防止该微控制器因超出容许电压而中止。所以，计算机系统必须能进一步确定在降低电压之后或加载了交流电之后是否唤醒了微控制器。

因此，CPU 在 996 处向微控制器 U_2 查询有关使电源 17 接通的事件。微控制器可以返回下列四种应答之一：(1) 微控制器已被复位，因而使得电源 17 开始在 ± 5 和 ± 12 线处提供稳压的电力；(2) 唤醒警铃的分钟数到时；(3) 在来自光隔离器 OPTO1 的振铃输入或 RS-232 振铃输入处产生了振铃；以及/或者(4) 按下了开关 21。诸如调度程序之类的应用程序可以直接从微控制器 U_2 中读取计算机系统通电的原因，而所说的调度程序则会响应系统增加功耗的原因而运行特定的程序。另外，通过一个或多个 BIOS 调用也可以成为使计算机增加功耗的原因。

除了可被 CPU 40 复位以外，微控制器 U_2 只能被复位子电路 920 所复位，每当 AUX5 线被加电或者故障，上述电路都会复位微控制器。所以，如果微控制器 U_2 被复位或者微控制器返回了一

在 997 处测试的无效唤醒代码, 则 CPU 40 必须要在 998 处确定电源是否应继续在 ± 5 和 $12 \pm$ 线处调节电压。为此, 在 CMOS NVRAM 内使用了一个标为 DEFAULT_ON 的标志。如果设置了该标志, 则电源 17 应在微控制器 U_2 被复位之后继续提供稳压的电力。另一方面, 如果未设置 DEFAULT_ON, 则电源 17 应在微控制器 U_2 被复位之后停止提供稳压的电力, 所以, CPU 40 会在 1000 处命令微控制器 U_2 使电源 17 停止在 ± 5 和 ± 12 线处提供稳压的电力, 此后, 电压会花几秒钟的时间下降至约为零, 从而给 CPU 40 以时间去执行多条命令。因此, CPU 40 在等待电源 17 所产生的系统电压下降时会在 1002 处执行无限循环(“旋转”), 直至 CPU 40 在 1004 处停止起作用。如上所述, 微控制器 U_2 最好由 AUX5 线不间断地供电并连续执行已编程的例程。

此后, 如果微控制器在 997 处返回一有效的唤醒代码, 或者如果虽微控制器 U_2 被复位, 但计算机系统仍在 998 处保持加电状态, 则 CPU 40 会在 1004 处命令微控制器 U_2 在确认应关闭电源而使电源 17 停止在 ± 5 和 ± 12 线处提供稳压电力之前向 CPU 40 产生一 SMI。此外, CPU 还在 1004 处设置 CMOS NVRAM 内的 DEFAULT_ON 位, 因此, 如果断掉交流电, 计算机系统会在重新加载交流电时自己返回接通状态。

然后, 引导例程在 1006 处执行本技术中专家所周知的 First Plug & Plan Resource Allocation Pass (第一次插入及计划资源分配)。

随后, 测试影子内存并将 BIOS 从 ROM 88 拷贝至 RAM 53 的影子内存部分。所执行代码的流程取决于是否在 CMOS

NVRAM 96 内设置了挂起标志。如果设置了挂起标志,则计算机系统 10 处于挂起状态 150,因而应将计算机系统 10 恢复至它被挂起时所处的状态。对E000H 和 F000H 段内的系统 RAM 53 进行简化的测试。为了减少计算机系统继续所花费的时间,只检查上述内存的适当大小并将该内存置零(将 0000H 写至各个存储单元)。

另一方面,如果在 CMOS NVRAM 96 中清除了挂起标志,则对E000H 和 F000H 段中的系统 RAM 53 作标准的深层次的内存测试,包括:(1)粘接位(sticky-bit)测试;(2)双位内存测试;以及(3)交叉地址线测试。这些测试在本技术中都是周知的。

在测试了 E000H 和 F000H 段之后,可投影 BIOS,这包括将 ROM BIOS 88 拷贝至系统 RAM 53 以及将内存控制器配置成能够执行来自 RAM 的 BIOS。投影 BIOS 可以增加系统的速度,并且能提高系统的效率,这是因为,正在运行来自较快的系统 RAM 53 的 BIOS(通常存取时间为 80 毫微秒)而不是来自较慢的 ROM 88 的 BIOS(通常存取时间为 250 毫微秒)。投影 BIOS 包括:将 BIOS 拷贝程序加载至内存低区的地址处;将 BIOS 从 ROM 88 拷贝至系统 RAM 53 的 E000H 段和 F000H 段;以及使影子 RAM 生效。

然后,在 384 处测试并初始化视频控制器 56 同时测试视频内存。这些测试及初始化工作在本技术中都是周知的。

随后,引导例程在 1008 处执行本技术的专家所周知的 Second Plug & Plan resource allocation pas (第二次插入及计划资源分配)。

所执行代码的流程在 386 处取决于是否在 CMOS NVRAM 96 内设置了挂起标志。如果设置了挂起标志, 则象任务 383 那样只检查其余的系统 RAM 53 的大小, 然后将该 RAM 置零。但是, 如果清除了 CMOS NVRAM 96 内的挂起标志, 则在 398 处用与任务 383 有关的内容所说明的三步骤深层次内存测试法来测试其余的系统 RAM 53。

测试内存之后, 在 400 处测试并初始化包括 8259、UART、8042 以及其它设备在内的辅助设备。在 408 处初始化固定盘控制器。

所执行代码的流程在 409 处取决于是否在 CMOS NVRAM 96 内设置了挂起标志。如果设置了挂起标志, 这表示在最近断电时已成功地保存了系统的状态, 那样, 引导例程就会跳过对硬盘驱动器控制器 86 和硬盘驱动器 31 的测试。另一方面, 如果在 CMOS NVRAM 96 内清除了挂起标志, 这表示在最近断电时未保存系统的状态, 那么, 正如在本技术中所周知的那样, 引导例程在任务 410 处对固定盘控制器 86 以及硬盘驱动器 31 作完全的测试。

然后, 在 412 处测试并初始化软盘驱动器控制器 84。

这时, 所有的设备均被初始化并且矢量均指向已知的存储单元, 因此所有的中断例程都会象所希望的那样工作。所以, 引导例程会在 414 处对 BIOS 数据区和矢量表进行快照, 即将 BIOS 数据区和矢量表的拷贝写至 E000H 段的数据结构内。挂起例程在任务 274 处利用 BIOS 数据区和矢量表的上述拷贝将计算机系统 10 置为已知的状态, 同时, 所有的中断均能象所希望的那样工作。

然后,在 416 处如在本技术中所周知那样“扫描进”并初始化 BIOS 的任何扩展部分。BIOS 的扩展部分是 BIOS 代码的程序块,该程序块是由诸如网络适配器之类的外部适配器增加给计算机系统。BIOS 的扩展部分一般位于 ISA 总线 76 的 C000H 和 D000H 段内并且具有相关的“特征”以标识该 BIOS 扩展部分自身。如果检测到 BIOS 的扩展部分,就检查其长度并计算和检查校验和。如果上述特征、长度和校验和都表明存在有有效的 BIOS 扩展部分,则程序控制权转至位于上述特征后面三个字节处的指令,并且, BIOS 扩展部分可执行诸如初始化外围适配器之类的任何所需要的任务。一旦上述 BIOS 扩展部分结束执行,控制权会返回至引导例程,而引导例程则会搜索其它的 BIOS 扩展部分。与上述 BIOS 扩展部分一样处理任何其它的 BIOS 扩展部分。如果没有检测到其它的 BIOS 扩展部分,则引导例程会转至任务 417。

然后, CPU 在 1010 处读取系统加电的时间增量并将该增量加至存储在硬盘驱动器 31 特定分区内的值,同时,将新的加电的总时间重新写回硬盘驱动器 31 的特定分区内。正如在本文其它地方将要说明的那样,微控制器 U_2 在使计算机系统降低功耗之前会通过确立 SMI 线而中断 CPU 40。因此, CPU 会在假定计算机系统即将减少功耗的情况下执行某些特定的任务。最佳的是,这包括将某些信息(如加电时间计时器所测量到的经过时间)加电增量等保存至 CMOS NVRAM。此后, CPU 40 允许微控制器 U_2 使计算机系统减少功耗。

引导例程在 417 处搜索硬盘驱动器 31 上特别为挂起文件而分配的分区。如果在分区表内找到带有 PS/1 标识符“FE”的分区

或带有标识符“84”的冬眠分区，并且该分区足以容纳用于所述特定计算机系统的挂起文件，那么，就将该分区用于挂起文件。因此，将挂起文件的特征 (Suspend File Signature) 写至上述区域的第一字节处，并将该区域的起始磁头、扇区以柱面写至 CMOS NVRAM 96。

然后所执行代码的流程在 418 处做分支转移，这取决于是否在 CMOS NVRAM 96 中设置了挂起标志。如果清除了挂起标志，则引导例程在 420 处将控制权转交给 PBOOT 例程。PBOOT 在本技术中是周知的，它负责从软盘或硬盘驱动器 31 上加载操作系统 (OS) 以及命令解释程序。如果在任务 417 处未找到用于挂起文件的分区，则 OS 执行与图 16 有关内容所说明的 OS 专用驱动程序，该驱动程序检查是否找到了一个分区，并且，如果没有，则在 FAT (文件分配表) 中分配一由连续扇区构成的文件 (如果必要的话，对一区域去除分段)，将所说的特征写至挂起文件的第一个字节并将挂起文件的起始磁头、扇区以柱面写至 CMOS NVRAM 96。

不管在何时分配挂起文件，该文件均应是连续的扇区，从而在挂起和继续期间能够分别快速地写至磁盘以及快速地从磁盘中读出。

然后，OS 按在 CONFIG. SYS 文件中所找到的指令配置计算机系统。最后，OS 运行 AUTOEXEC. BAT 文件，该文件最终将执行控制权返回给操作系统。如果在 CMOS NVRAM 96 中清除了挂起标志，这表明最近断电时未保存计算机系统的状态，那么，就忽略 RESUME. EXE，在与任务 421 有关的内容将详细说明

RESUME.EXE。

参照任务 418, 如果在 CMOS NVRAM 96 中设置了挂起标志, 这表示最近断电时保存了计算机系统的状态, 那么, 所执行代码的流程就会在 419 处作分支转移, 这取决于是否在 CMOS NVRAM 96 内设置了重新初始化适配器标志 (Reinitialize Adapters Flag)。如果设置了重新初始化适配器标志, 则引导例程在 421 处将控制权交给 PBOOT 例程。同通常的 PBOOT 例程一样, 本发明的 PBOOT 加载 OS, OS 则根据在 CONFIG.SYS 和 AUTOEXEC.BAT 文件中找到的命令配置计算机系统, 此外, 还加载驱动程序并按本技术中所周知的方式配置计算机系统。

CONFIG.SYS 和 AUTOEXEC.BAT 中的命令可以初始化计算机系统适配器卡。本申请假定存在有三种类型的适配器卡: I 型适配器不需要初始化; II 型适配器需要初始化但被 BIOS 扩展部分或被每个 CONFIG.SYS 或 AUTOEXEC.BAT 所加载的驱动程序设置成周知的工作状态; III 型适配器由计算机系统执行的代码所修改。可以挂起和恢复包括 I 型及 II 型适配器的计算机系统, 但不能恢复包括含有多网络适配器的 III 型适配器的计算机系统, 除非该适配器卡带有相关的 APM 可认识的设备驱动程序, 该驱动程序在诸如系统断电之类情况发生之后可重新初始化上述适配器。系统可以挂起带有 APM 可认识的设备驱动程序的 III 型适配器卡。

在本实施例中, 将文件 RESUME.EXE 加到 AUTOEXEC.BAT 文件内, RESUME.EXE 负责将程序控制权从 OS 传送给继续例程。OS 在任务 420 忽略 RESUME.EXE 的

存在，但 OS 在任务 421 却执行 RESUME.EXE，在 OS 从 CONFIG.SYS 和 AUTOEXEC.BAT 加载的设备驱动器结束初始化 II 型适配器之后，RESUME.EXE 将控制权转给继续例程。

参照任务 419，如果在 CMOS 96 中清除了重新初始化适配器标志，则 OS 通过 RESUME.EXE 将执行控制权转给继续例程。继续例程从硬盘驱动器上的挂起文件中恢复系统状态，继续例程将在与图 12 有关的内容中予以详细说明。

参照图 12，它显示了从任务 450 至任务 530 的继续例程的详细内容。首先，在 451 处测试 CPU，如果 CPU 带有 SMI，则产生一 CPU 继续 SMI，该 SMI 将 CPU 置为 SMM (系统管理监控) 模式并转至任务 454 处的代码。如果 CPU 不带有 SMI，则停止继续，其中，进行复位，复位处理程序转至任务 454 处的代码。在配置进程中，可能会将 BIOS 数据区和矢量表修改成未知的状态。所以，基本的 BIOS 例程可能不象希望的那样起作用。因此，继续例程会在 454 处使 E000H 段成为可读/写的并且在 456 处调用交换 BIOS 数据区和矢量表例程 (Swap BIOS Data Area & Vector Table Routine)。该例程将在任务 414 处拷贝至 E000H 段的已知且适当的 BIOS 数据区和矢量表与当前在 0000H 段内活动的经过修改的 BIOS 数据区和矢量表作交换。当上述交换例程结束时，上述已知的 BIOS 数据区和矢量表在 E000H 段内是活动的，而上述经过修改的 BIOS 数据区和矢量表则处在 0000H 段内，并且，BIOS 例程会象所希望的那样起作用。

然后，继续例程在 458 处使除了支持键盘和硬盘驱动器的中断以外的所有中断都无效。此后，继续例程在 460 处确定挂起文

件在硬盘驱动器31 上的位置并读取文件的长度和特征, 该特征如上所述是一用于挂起文件的多字节标识符。所执行代码的流程随后在 462 处作分支转移, 这取决于挂起文件是否有正确的长度和特征。如果挂起文件不具有正确的长度和特征, 则挂起例程在 464 处清除 CMOS 内的挂起标志, 程序控制权在 466 处转给处于复位矢量 (Reset Vector) 所指向位置处的代码, 从而使得计算机系统就象从未挂起那样进行引导。另一方面, 如果挂起文件具有正确的长度和特征, 则继续例程在 468 处继续, 它通过将位于挂起文件内上述特征之后的 64K 块 (挂起文件对应于 E000H 段信息的部分) 读至 1000H 段内而使得计算机系统继续。

随后, 在 470 处计算位于 1000H 段内的上述数据块的校验和, 并在 472 处从 CMOS 非易失性存储器 96 中读出先前存储的校验和, 引后, 所执行代码的流程在 474 处作分支转移, 这取决于任务 470 处所计算出的校验和是否与任务 330 处所计算出的校验和相同。如果任务 470 处所计算出的校验和不同于任务 330 处所计算出的校验和, 则挂起文件因某种原因存在有问题 (例如它可能被改变了等等), 因而控制权转至任务 464 处, 该任务如与任务 464 和 466 有关的内容所说明的那样清除挂起标志并复位计算机系统。如果任务 470 处所计算出的校验和与任务 330 处所计算出的校验和相同, 则假定挂起文件与挂起例程所写的挂起文件是同一个文件, 因而在 476 处将 1000H 段内的数据拷贝至 E000H 段。

继续例程在 478 处将一特定的信号屏面写至屏幕, 该信号屏面用于通知用户计算机系统正在恢复以及用户应按下

Ctrl - Alt - Del 键以中止继续。同挂起例程一样，按下 Ctrl - Alt - Del 键会在 526 处清除挂起标志，并在 528 处重新引导计算机系统。因此，在按下 Ctrl - Alt - Del 键且继续例程正在运行时重新正常引导计算机系统。

然后，在 480 和 482 处通过将来自 E000H 数据结构的值分别写至 82077 软盘控制器的寄存器以及 DMA 部件的寄存器中而恢复 82077 软盘控制器 84 和 DMA 部件 71。

随后，在 1020 处启动中断驱动的并行线程，该线程用于从 E000H 的数据结构中恢复任何调制解调器的状态。同任务 976 处的例程一样，调制解调器恢复例程做以下工作：捕获与涉及该特定调制解调器的 COMM 端口相对应的中断；读取来自 E000H 数据结构的值；将命令与值传递给调制解调器以使该调制解调器恢复其中的寄存器。上述例程将一第一命令传递给调制解调器然后以中断驱动的方式进行应答，以便接收调制解调器的应答并响应各 COMM 端口的中断而将下一个值传递给调制解调器，直至恢复了所有的调制解调器的寄存器。同调制解调器保存例程一样，如果上述例程不是作为并行线程执行的，则该例程会在它要使计算机系统继续所花费的时间上增加几秒钟。作为一种中断驱动的并行线程，如果上述例程在从硬盘驱动器 31 中读出系统状态之前全部执行了，则该例程不给前述继续工作增加时间。

当从任务 486 到 500 处的中断驱动的并行线程调制解调器恢复例程开始之后，利用一双缓冲区例程从挂起文件中恢复系统内存，上述双缓冲区例程同与挂起例程中任务 304 至 318 有关的内容所说明的例程相类似。上述双缓冲区系统从挂起文件中读取

压缩数据，将该压缩数据写至E000H段内，对压缩的数据进行解压缩并写至系统内存。有两个例程以时间多路复用的方式进行工作：一个从挂起文件中读取数据并将数据写至E000H段内；另一个对上述数据进行解压缩并将解压缩后的数据写至系统内存。后一个例程在前台运行，前一个例程则是在后台运行的中断驱动的例程。很明显，由于只有一个CPU，所以在给定的时间内只能执行一个例程，但是，由于上述前一个例程是中断驱动的，所以它可以根据需要中断后一个例程的执行，从而使来自挂起文件的数据传送的速度最佳化。上述两个缓冲区均为8K字节长，这认为能使传送时间最佳化。

上述进程始于486处，它从挂起文件中读取数据并将足够的数
据写至E000H段以填充第一个8K的缓冲区。这时，在306处启动读自缓冲区例程(Read from Buffer Routine)，该例程总体上用标号489表示。读自缓冲区例程489是一中断驱动的例程，它在后台运行并包括任务490-492。总体上用标号493所表示的解压缩例程(Decompression Routine)包括任务494-498并且是前台例程。首先，读自缓冲区例程489在490处开始读取挂起文件的下一个8K数据并将其写入现在为当前缓冲区的另一个缓冲区中。当读自缓冲区例程489从挂起文件中读取下一个8K数据并将其写入当前缓冲区中时，解压缩例程493在494处读取由任务486所填写缓冲区、对压缩的数据进行解压缩并将已解压缩的数据写至系统内存。一旦解压缩例程493对上述缓冲区内的所有数据进行了解压缩，则下一步就是在496处确定是否已对整个的系统内存进行了解压缩。

IDE 控制器 86 不能很快地从硬盘驱动器 31 中读取数据。因此，解压缩例程 493 总是在读自缓冲区例程 489 结束将数据从硬盘驱动器 31 中读进当前缓冲区之前结束对未写至硬盘驱动器 31 的 8K 缓冲区的解压缩工作。所以，解压缩例程 493 必须等待读自缓冲区例程 489 结束从硬盘驱动器 31 读出数据。如果解压缩例程 493 尚未解压缩和写所有的系统内存，则解压缩例程 493 在 498 处等待读自缓冲区例程 489。解压缩例程 493 和读自缓冲区例程 489 通过一组标志而相互通信。当读自缓冲区例程 489 结束将数据挂起文件读进当前缓冲区时，该例程 489 随后在 490 处转换上述缓冲区标志，这向解压缩例程 493 表明它可以开始对缓冲区内刚从挂起文件读出的数据进行解压缩。然后，读自缓冲区例程 489 在 492 处确定是否仍要从挂起文件中读取一 8K 的块。如果不是，读自缓冲区例程在 502 处从挂起文件中读取剩下的数据并将该数据写至当前缓冲区。然后读自缓冲区例程在后台停止运行，这实际是在 500 处等待解压缩例程完成对前面的内存所进行的解压缩工作。

同时，解压缩例程 493 通过检验缓冲区的标志而确定缓冲区是否准备好被解压缩至系统内存。也就是说，解压缩例程在 498 处等待，直至读自缓冲区例程结束对当前缓冲区的处理，这时，解压缩循环在 494 处继续。

一旦解压缩例程 493 结束了对所有系统内存的解压缩，则唯一在后台运行的例程是中断驱动的调制解调器恢复例程，而主程序则在 504 处继续，上述恢复例程在与任务 1020 有关的内容中给予了说明。

然后，在 504 和 506 处通过将来自 E000H 段数据结构的值写至视频控制器的寄存器以及 IDE 控制器的寄存器而恢复视频控制器 56 和 IDE 控制器 86。任务 504 也是这样的程序点：如果已采用了检查点，则挂起例程就转至该程序点（见任务 1024）。

此后，继续例程在 1022 处测试与任务 1022 有关的内容所述的中断驱动的调制解调器恢复例程是否结束。如果不是，则继续例程等待上述恢复例程结束。

如 508 处所示，中断驱动的调制解调器状态恢复例程结束之后，通过将适当的值写至 CPU 40 和高速缓冲存储器的控制器 62 而分别使 CPU 高速缓冲存储器 41 以及系统高速缓冲存储器 60 有效。随后，继续例程在 510 至 514 处通过将来自 E000H 段数据结构的值写至计时器控制器 102、8042 键盘接口微处理器 104 以及 8259 中断控制器 92 内的寄存器中而恢复这些设备的状态。

然后，通过在 484 处将来自 E000H 段数据结构的值写至 UART 的寄存器中而恢复 UART 94。

此后，继续例程在 516 处调用交换 BIOS 数据区和矢量表例程。在调用上述例程之前，已知的 BIOS 数据区和矢量表在 0000H 段内是活动的，而从挂起文件中所读出的 BIOS 数据区和矢量表则在 E000H 段数据结构中是不活动的。交换之后，已知的 BIOS 数据区和矢量表在 E000H 段内是不活动的，而由挂起例程所保存的 BIOS 数据区和矢量表则在 0000H 段内是活动的。

最后，继续例程在 518 处转至恢复 CPU 例程 (Restore CPU Routine)，该例程将 CPU 40 恢复至挂起之前状态。在与图 14 有关的内容中将详细说明恢复 CPU 例程。恢复 CPU 例程最终将执

行控制权交还给 APM。

最后, CPU 40 执行 RETURN 指令, 从而使计算机系统返回至 APM。系统当前继续执行代码, 就好像它从未挂起过。从实用角度看, 计算机系统不受挂起/继续过程的影响。

参照图 13, 它显示了保存 CPU 状态例程 (Save CPU State Routine) 的流程。挂起例程在 600 处转至保存 CPU 状态例程。请注意, APM 已使上述例程从中开始执行的 E000H 和 F000H 段成为可读/写的。此外, APM 将 EFLAGS 和八个通用寄存器保存起来, 如 602 处所示。保存 CPU 状态例程在 604 处首先等待任一 DMA 结束并与鼠标器 13 的数据包相同步, 以确保上述例程能在鼠标器数据包传输之间执行。下列步骤能使 DMA 结束并与鼠标器数据包相同步: (1) 使中断生效; (2) 用 7 毫秒等待任一 DMA 结束; (3) 使中断无效; (4) 用 5 毫秒等待鼠标器数据包的边界; (5) 使中断生效; (6) 再用 5 毫秒的时间等待鼠标器数据包的到达; 以及 (7) 使中断无效。这些步骤之后, 所说的代码可在鼠标器数据包之间安全地运行。

然后, 在 606 处将地址线 (Address Line) 20 (I/O 端口 92H) 的状态压至堆栈。

随后所执行代码的流程在 1030 处做分支转移, 这取决于 CPU 40 是否是带有 SMI 的“S”部件。如果是这样的话, CPU 40 在 1032 处命令微控制器 U_2 产生一 SMI 给 CPU 40。CPU 40 内的微码响应 SMI 而在 1034 处将 CPU 40 的状态保存至 E000H 段数据结构中的 E000: FE00H 处。此后, CPU 40 在 1036 处保存浮点协处理器的状态并在 1038 处调用挂起例程 (图 10)。正如本文其它

地方所说明的那样，挂起例程在 1040 处返回并在 1040 处恢复浮点协处理器的状态。然后，在 1042 处，一 RSM (继续) 指令恢复 CPU 状态并转至任务 732。

另一方面，如果 CPU 40 不带有 SMI，则必须在 608 处用图13代码的其余部分将 CPU 的状态保存起来并将算术协处理器 44 的状态压至堆栈。然后，在 610 处设置或清除一标记以分别指示 CPU 是在 32 位模式下运行还是在 16 位模式下运行。

随后，所执行代码的流程在 612 处作分支转移，这取决于 CPU 40 是否在保护模式下运行。如果 CPU 40 不是在保护模式运行，则 CPU 40 必定在实模式运行并且可以非常直接的方式保存其寄存器。首先，在 614 处将机器状态字的值以及 CR3 写至 E000H 段的数据结构中。仍在 614 处将零写至 E000H 段数据结构中与 TR 和 LDTR 相对应的区域内，这是因为，在实模式下 TR 和 LDTR 均为零。

然后，上述代码并入 616 处的共用代码流程，在该流程处，将存储在 GDTR 和 LDTR 内的值写至 E000H 段的数据结构中。随后所执行代码的流程在 618 处做分支转移，这取决于 CPU 40 是否以虚拟 8086 模式运行。如果 CPU 40 未以虚拟 8086 模式运行，则代码沿共用路径继续至任务 620，在此处将排错寄存器 DR₇、DR₆、DR₃、DR₂、DR₁ 以及 DR₀ 压入堆栈。这些寄存器供排错程序及其它例程使用的。在 622 处将 DS、ES、FS 和 GS 压入堆栈。然后将 CS、SS 和 ESP 写至 E000H 段的数据结构中。

这时，已对所有要写至 E000H 段数据结构的值进行了写操作，因此，可在 626 处将影子 RAM 的 E000H 和 F000H 段改回为

只读的。然后在 628 处用 Write - Back 和 Invalidate Cache 指令倾泻 CPU 的高速缓冲存储器。

随后，在 630 处设置 CMOS 非易失性存储器 96 内的一个独特的关机标记 (Shutdown Flag)。最后，保存 CPU 状态例程在 632 处实际上“返回”至挂起例程。所说的“返回”实际上是一种复位，该复位的后面是代码中的分支。通过转至复位矢量所指示的代码而复位 CPU 40。复位 CPU 40 强制 CPU 进入实模式，这种方式下，在不必担心产生保护错误的情况下可以存取所有的设备与内存存储单元。此后，将 CPU 的状态已保存起来，并且，挂起例程必须将计算机系统其余的状态保存起来。

在复位矢量所指向的代码内，程序控制作分支转移，这取决于是否在 CMOS 96 内设置了关机标记。如果清除了关机标记，则计算机会象通常那样引导。另一方面，如果设置了关机标记，则代码转至挂起例程的其余部分，也就是说，执行控制权转至图 10 挂起例程中的任务 253，该任务完成了挂起计算机系统 10。因此，保存 CPU 状态例程在 623 处实际上“返回”至了挂起例程。

参照任务 612，如果 CPU 处于保护模式，则代码在 634 处作分支转移，这取决于 CPU 是否处于虚拟 8086 模式。如果 CPU 不处于虚拟 8086 模式，则代码在 636 处再次作分支转移，这取决于当前的特权级别是否为零。如果当前的特权级别不为零则是一没有适当特权的例程正在执行保存 CPU 状态例程，因而要调用致命挂起错误例程 (Fatal Suspend Error Routine，始于任务 652 处)。以下将详细说明致命挂起错误例程。如果程序控制从致命挂起错误例程返回，则 CPU 必须返回至调用保存 CPU 状态例程之

前的状态,所以,程序执行会转至图 14 中的任务 794,该任务能部分地恢复 CPU。由于 CPU 中很少有变化,所以只需要部分地恢复。

参照任务 636,如果调用代码具有适当的特权级别,则保存工作在 642 处继续,从而将 CR_0 、 CR_3 、TR 以及 LDTR 的值保存至 E000H 段的数据结构中。然后,代码的流程进入 616 处的共用代码流程,在 616 处,如以上所述那样将 GDTR 和 IDTR 的值保存至 E000H 的数据结构。从此处开始,代码沿以上说明过的流程 618 至 632 执行,从而“返回”(复位加上一个分支转移)至其余的挂起例程代码。

参照任务 634,如果 CPU 40 处于虚拟 8086 模式,则在 644 处继续执行,在 644 处,将机器状态字的值(CR_0 的低 16 位)保存至 E000H 的数据结构中,并设置一位于 E000H 段数据结构内的标志(Flag)以指示 CPU 处于虚拟 8086 模式。然后,代码经由转换 646 和 648 进入 616 处的共用代码。在任务 618 处,如果 CPU 处于虚拟 8086 模式,则控制转至 650,在 650 处,将 DS、ES、FS 和 GS 的值保存到 E000H 段的数据结构中。代码再次并入 624 处的共用代码。从此处开始,代码沿以上说明过的流程 624 至 623 执行,从而“返回”(复位加上一个分支转移)至其余的挂起例程代码。

任务 652 至 664 显示了致命挂起错误例程,如果具有不适当特权级别的代码试图保存 CPU 的状态,则调用致命挂起错误例程。首先,在 654 处复位故障保险计时器,然后,在 656 处扬声器以可听见的频率蜂鸣数次,例如以 886Hz 长 0.25 秒蜂鸣三次,蜂鸣

间隔为 1/6 秒。三声蜂鸣警告用户未进行所尝试的挂起。蜂鸣之后，在 658 处再次复位故障保险计时器从而在故障保险计时器到时之前给用户固定的 15 至 18 秒的时间以关闭电源 17。

然后，致命挂起错误例程在 660 至 662 处重复地检查用户是否按下了开关 21，这表示用户要中止挂起。通过 CPU 查询微控制器 U_2 是否产生了接通事件，从而能检查开关的接通情况。如果用户按下了按钮 21，则执行控制返回至前述的任务 640。如果用户未在 15 至 18 秒内按下按钮 21，则故障保险计时器将会到时，微控制器会“关闭”电源 17，因而很明显，CPU 40 所执行的代码在系统电压降至容差范围之外时均会停止。

参照图 14，它显示了恢复 CPU 状态例程 (Restore CPU Routine)，该例程始于 700 处。在其余的硬件及内存均已恢复至它们挂起之前的状态以后，继续例程会调用上述恢复 CPU 状态例程。首先，如果 E000H 段还不是可读/写的，则应在 702 处将该段置成可读/写的。

然后，所执行代码的流程在 704 处作分支转移，这取决于 CPU 40 在挂起时是否是以虚拟 8086 模式运行。如果 CPU 40 在计算机系统 10 挂起时以虚拟 8086 模式在运行，则代码进入任务 706 至 728，任务 706 至 728 是恢复虚拟 8086 CPU 专用的。然后代码并入从任务 730 至 748 的共用流程。

如果在保存 CPU 状态时 CPU 处于虚拟 8086 模式，则用于将 CR_3 、LDTR 及 TR 的值保存至 E000H 段数据结构中的保存 CPU 状态例程无法存取 CR_3 、LDTR 和 TR。所以，必须分别在 706、708 以及 710 处估计 CR_3 、LDTR 以及 TR。一般地说，通过遍

历系统 RAM 31 查找 CR_3 、LDTR 和 TR 所指向的结构, 可估计出它们的值。例如, 查找 GDT 内的 LDT 条目可以确定出 LDTR。

在任务 706 处估计 CR_3 。 CR_3 包括页面目录基址寄存器 (PDBR)。该寄存器包含有页面目录的页帧地址、页面级高速缓冲存储器禁止 (PCD) 位以及页面级写完 (PWT) 位。认识到页面目录一定始于系统 RAM 53 内的 4K 边界处, 知道了由保存 CPU 状态例程保存到 E000H 段数据结构内的 IDTR 值和 GDTR 值并且假定 BIOS 代码从 F000H 段开始执行, 就可以估计出 PDBR。上述假定是合理的, 这是因为, 为了速度的缘故已将 BIOS 代码投影进了影子 RAM。如果操作系统将 BIOS 代码拷贝至不同的区域, 则对 CR_3 的估计就会失败。

利用上述知识和假设, 检测物理内存所有 4K 页面是否有对应于 BIOS 代码段的页转换表。也就是说, 相对上述页面的 03C0H 偏移总是包含值 $000F0 \times \times \times$ 、 $000F1 \times \times \times$ 、 $000F2 \times \times \times$, $000FE \times \times \times$ 。一旦确定了上述页面的位置, 就在系统 RAM 53 内查找一页面目录, 该目录的第一个条目对应于以上确定的页面表的物理地址。上述页面目录的物理地址是 PDBR 的最佳“猜测”值。

然后, 通过确保 PDBR 能正确地转换用于 GDTR 和 IDTR 的地址而验证假设的 PDBR。也就是说, PDBR 用于转换 GDTR 的线性地址, 并且 GDT 的第一个条目被验证是空的 (null) (在任何 CPU 模式下 GDT 的头八个字节都是 00H)。此后, 验证所返回的物理地址是在物理内存的范围内。为了能进行线性至物理转换, 使用模仿 CPU 转换方法的子例程, 如果在物理内存中有该物

理页面存在, 则在 ESI (段结束指示符) 中返回经转换的地址并清除进位标志 CF, 如果内存中不存在该物理页面, 则设置 CF。利用这一转换例程从内存 53 中读取 GDT 的第一个字节。如果 GDT 的第一个条目为空 (null), 则假设的 PDBR 就通过第一次检验, 因而会被再次检验。然后用该 PDBR 转换 IDTR 以便利用上述转移例程查找 IDT。此后证实所返回的物理地址是在物理内存的范围内。如果 IDT 的第一个存储单元出现在物理内存内, 则 PDBR 通过第二次检验。

如果假设的 PDBR 正确地转换成了 GDTR 和 IDTR, 则假定该值是 PDBR, 并将该值写至 E000H 段数据结构内的 CR_3 区域。另一方面, 如果假设的 CR_3 没有通过检验, 则上述例程会再次开始以在系统内搜索另外的 BIOS 代码段页转换表, 该表可能会得出有效的 CR_3 。

对正常的主板操作来说, 总是假定 PCD 和 PWT 固定在 00H。将它们的值置为 0 并同 PDBR 一起写入 E000H 段数据结构内的 CR_3 区。

一旦估算出了 CR_3 , 就在 708 处估算 LDTR。由于已知 LDT 在 GDT 内的某一位置处并且 LDT 一定会出现在内存中, 所以, 在估算出 CR_3 的情况下, 可以估算出 LDTR。为了估算 LDTR, 在 GDT 内查找标记为当前的 LDT。将在物理内存中出现 (用与任务 706 有关的内容所说明的转换例程来测试) 并被标记为当前的第一个 LDT 假定为 LDTR 所指向的表。将该表的起点的物理地址保存至 E000H 段数据结构内的 LDTR 区。

即使是在 OS/2 中有多个 LDT 可被标记为当前的并出现在

物理内存中,上述估算 LDTR 的方法也是足够可靠的,因而可以使用。EMM 386是普通的虚拟 8086 模式 (Virtual 8086 Mode) 例程,因此似乎会有问题,但是,由于 EMM 386 只有一个 CR_3 和一个 LDTR,所以很容易估算出 EMM 386 的 CR_3 和 LDTR。

一旦估算出了 CR_3 和 LDTR,就在 710 处估算 TR。从实质上说,要在 GDT 和 LDT 内的各个任务选择器条目中查找设置了忙位的任务状态选择器。测试各条目的类型字段以确定该条目是忙的 80286 任务状态选择器还是忙的 80486 任务状态选择器。假定忙的 286 TSS (任务状态选择器) 或忙的 486 TSS 中的第一个条目是 TR 所指向的地址。将带有忙的 286 或忙的 486 TSS 的上述条目的物理地址保存至 E000H 段数据结构内的 TR 区。如果没有条目带有忙的 286 或 486 TSS,则将 0 保存到 E000H 段数据结构内的 TR 区。

估算出 CR_3 、LDTR 以及 TR 之后,代码在任务 712 处继续。在 712 处,如果 TR 指向有效的 TSS,则在 714 处清除 TR 所指向的 TSS 内的忙位。不管怎样,代码在 716 处继续,在 716 处,将对 GDT 有效的选择器加载进 DS、ES、FS 和 GS。在 718 处利用来自 E000H 段数据结构的值加载 CR_3 和 CR_0 。然后,在 720 处启动分页,因此,线性地址等于物理地址的区域只有 E000H 和 F000H 段内的区域。随后在 722 处将存储在 E000H 段数据结构内的值加载进 IDTR、GDTR、LDTR 以及 TR。

最后,在 724 和 726 处通过将来自 E000H 段数据结构的对应于 GS、FS、DS、ES、SS、ESP、EFLAGS (设置了 VM 位之后) 以及 CS 的值压入虚拟 8086 中断堆栈 (Virtual 8086 Interrupt Stack)

而建立该堆栈。而且，还在 726 处将对应于任务 730 处的代码的返回地址压入上述堆栈。其后，执行 IRETD 指令以将 CPU 40 设置回虚拟 8086 模式并将执行权转给对应于任务 730 的代码。

任务 730 启动图 14 中各种线程均使用的共用线程。在任务 730 处，从保存在 E000H 段数据结构内的值中恢复协处理器 44。然后地址线 20 (I/O 端口 92H) 在 732 处从堆栈中弹出。任务 732 也是以 SMI 为基础的 CPU 保存状态例程所转至的程序点（见任务 1046）。此后在 734 处使影子 RAM 的 E000H 段再次成为只读。在 736 处通过重新启动故障保险计时器而使 ARM 与硬件相连接，如与图 6A 和图 19 有关的内容所述。在 738 处再次使影子 RAM 的 E000H 和 F000H 段成为只读的。最后，恢复 CPU 状态例程在 740 处设置一标志，此标志表示进行了正常的继续。恢复 CPU 状态例程并不执行任务 742、744 和 746，这些任务只用于表示：在返回到被挂起事件所中断的代码之前的某一时刻，八个通用寄存器会弹出堆栈；使可屏蔽的中断有效（如果代码中断时它们已被置为有效的）；以及所说的标志弹出堆栈。最后，恢复 CPU 状态例程返回至管理例程，管理例程将控制权交还给 APM，APM 又更新任何旧的系统值并将控制权交还给被中断的代码。

参照任务 704，如果 CPU 40 在中断时不处于虚拟 8086 模式，则代码进入 750 至 792 的流程，在 792 处，代码并入任务 730 至 748 的共用线程。在 750 处，如果 E000H 数据结构中的 TR 值显示出 TR 指向有效的 TSS，则在 752 处清除上述 TSS 内的忙位，然后在 754 处将来自 E000H 段数据结构的价值装入 GDTR 和 CR₀。

然后在 756 至 764 处将一空的 (dummy) 页面目录表和页面转换表装入 E000H 段。首先, 在 756 处使影子 RAM 的 E000H 段成为可读/写的。其次, 在地址 0E 000H 处创建一新的页面目录表, 如 758 处所示。第三, 在 760 处更改上述新页面目录表内的第一个条目以使之指向 0E 1000H。第四, 在 0E 1000H 处创建一新的页面转移表, 以使得地址 0E 000H 至 0FFFFFFF 是存在的, 并使得线性地址等于这一地址范围内的物理地址, 如 762 处所示。最后, 将 0E 000H 装入 CR₃ 内的页面目录基址寄存器。因此, 可以通过 0E 0000H 内的新的空 (dummy) 页面目录和页面转换表来进行地址转换。当在任务 754 处加载 754 时, 重新激活页面调度 (如果可行)。

然后, 在 766 处使影子 RAM 的 E000H 和 F000H 段成为可读/写的。此后, 如果 CPU 40 在挂起时正在执行 16 位的代码, 则 CPU 处于 16 位模式 (16 - Bit Mode), 因而在 770 处将指向 16 位代码路径的偏移值保存至 E000H 段的数据结构内。另一方面, 如果 CPU 40 不处于 16 位模式, 则 CPU 处于 32 位模式 (32 - Bit Mode), 因而在 772 处将指向 32 位代码路径的偏移值保存至 E000H 段的数据结构内以代替 16 位的偏移值。无论是哪一种情况, 上述两种代码路径都是平行的, 不同点只在于一个使用 16 位操作数, 另一个使用 32 位操作数。任务 770 和 772 只在上述平行的路径之一中建立所说的偏移值。在以下所述的任务 782 处进入上述路径之一 (与偏移值相对应的那个路径)。

然后, 在 774 处, 将来自 E000H 段数据结构的 CR₃ 值装入 EDX, 将来自 E000H 段数据结构的 SS 值装入 CX, 将来自 E000H

段数据结构的 ESP 值装入 EBP, 将来自 E000H 段数据结构的 TR 值装入 ESI 的高位部分, 将来自 E000H 段数据结构的 LDTR 值装入 ESI 的低位部分 (SI)。这些值被移位进下面各自适当的存储单元。在 776 处将来自 E000H 段数据结构的值装入 GDTR、LDTR 和 CR₀。在 778 处, 将存储在 SI 内的 LDTR 值装入 LDTR。然后代码远程转移至在任务 770 或 772 处设置的偏移值。通过直接将操作码放进源代码内并利用来自任务 770 或 772 的偏移值, 可以将上述远程转移偏码。然后, 代码转至 16 位操作码路径或 32 位操作码路径, 如 782 处所示。

此后, 在 784 处将存储在 EDX 内的 CR₃ 值装入 CR₃, 将存储在 CX 内的 SS 值装入 SS, 将存储在 EBP 内的 ESP 值装入 ESP。在 786 处将 GS、FS、ES 和 DS 弹出截栈。在 788 处, 如果被中断的 CPU 40 正在以受保护模式执行代码, 则在 790 处将存储在 ESI 高位部分内的 TR 值装入 TR。无论是哪一种情况, 代码都在任务 792 处继续, 在任务 792 处, 排错寄存器 DR₀、DR₁、DR₂、DR₃、DR₆ 以及 DR₇ 均弹出堆栈。

这时, 代码的流程并入任务 730 至 748 的共用代码流程, 此代码已在以前作了说明。在 794 处, 错误恢复例程从保存 CPU 状态例程的任务 640 进入上述共用代码流程。

参照图 15, 它显示了保存 8259 状态例程 (Save 8259 State Routine) 的流程图, 该例程始于 800 处。保存 8259 的状态始于在 802 处保存实时时钟 98 所使用的周期性中断值, 以及在 804 处将所有的其它可读寄存器保存至 E000H 段的数据结构。正如在本技术中所周知的那样, 计算机系统 10 的结构需要某些 8259 只读

寄存器具有固定的值。这些值是周知的，并且不需要确定。难以获得的 8259 值是 8259 的基地址、8259 从属地址以及 OS 将两个 8259 设置成能显示出未决的中断还是能显示出进行中的中断。

用图 15 中的其余代码判定上述四个项目。在仅不屏蔽键盘 12 和鼠标器 13 的中断的情况下屏蔽 8259，如 806 处所示。

在 808 处通过将 1K 的底部物理内存拷贝至 E000H 段数据结构而将中断矢量表保存起来。然后，在 810 处，通过加载 256 个独特的空矢量而将一新的“空” (dummy) 中断矢量表装进上述 1K 的底部物理内存，而所说的 256 个空矢量则指向 256 个空中断服务例程，这些例程始于 C800H 段。如 812 处所示，在 C800H 段内生成 256 个空中断服务例程。

然后在 814 处禁止键盘 12 和鼠标器 13 中断。在 816 处确认任何未被确认的键盘 12 和鼠标器 13 的中断。

此后在 818 处产生一键盘中断并在 820 处测试该中断以确定是否将基址 8259 设置为未决的或是使用的。随后将该值写至 E000H 段的数据结构中。代码在 822 处等待对中断的服务。通过调用上述空服务例程中的一个而在 824 处对中断进行服务。调用上述空服务例程来确定 8259 的基地址并确定该 8259 是处于未决模式还是处于工作中模式，将所说的基地址与模式保存至 E000H 段的数据结构中。

对从属的 8259 来说，在任务 826、828、830 和 832 处执行类似的过程。

在 834 处通过将来自 E000H 数据结构的值拷回 1K 的低部物理内存而恢复中断矢量表。然后在 836 处再次将 E000H 段置

为只读的，并且，在 838 处屏蔽所有的中断，以便在 840 处返回调用程序。

参照图 16，它显示了用于动态分配挂起文件的例程。如与任务 1012 有关的内容所说明的那样，分配在 FAT 内的挂起文件应是连续的扇区以便能在挂起与继续过程中分别快速地写至磁盘和快速地从磁盘中读出。此外，正如本技术的专家所知道的那样，挂起文件必须足够大以便存储整个系统状态经压缩的内容。

为此，上述动态分配挂起文件的例程始于 1050。每当计算机系统在不执行继续例程的情况下进行引导时，OS 就执行上述例程，并且该例程是在系统中增加内存之后执行的。首先，图 16 所示的分配例程在 1052 处通过检查 CMOS NVRAM 内的标记来测试是否存在有电源管理电路。如果不存在电源管理硬件 106，则程序在 1054 处退出。如果存在电源管理的硬件 106，则该例程在 1056 处检查确定是否有未决的继续，如果有，则程序在 1058 处退出。

如果继续不是未决的，则分配例程在 1060 处测试是否有保存文件的分区 (Save File Partition)。如果存在有保存文件的分区，则程序在假定该分区足以能存储整个系统状态的情况下而在 1062 处退出。

如果不存在保存文件的分区，则必须在 FAT 中为该保存文件 (Save File) 分配一个文件。首先，在 1064 处确定上述文件的长度。此长度是通过把系统 RAM 53 的长度、视频存储器 58 的长度、其它任何带有较大易失性存储容量的设备的长度以及用于存储诸如 CPU 40 之类的各种设备的寄存器值的 64K 字节区域加在

一起而计算出来的。

计算出所需的保存文件长度之后，分配例程在 1066 处试着在 FAT 内分配该保存文件。如果在硬盘驱动器 31 上没有足够可用的存储空间，则分配例程就在 1070 处调用一个例程，在可能情况下增加硬盘驱动器 31 可用空间长度。

DOS 调用无法保证文件中连续的扇区。所以，如果硬盘驱动器 31 具有足够的空间去存储保存文件，则分配例程在 1072 处确定该空间是否是连续的。如果保存文件是分段的（不连续），则分配例程就在 1074 处调用一个例程，在可能情况下消除硬盘驱动器分段，以便为保存文件提供连续的文件。

如果保存文件不是分段的，则分配例程在 1076 处将特征（“PS/1 Power Management”）写至保存文件的第一个扇区。然后，分配例程在 1078 处将上述文件的 DOS 句柄转换成用于 BIOS 的物理柱面、磁头以及扇区并将这些值写至 CMOS NVRAM。最后，分配例程在 1080 处退出。

在 1074 处调用的用于消除硬盘驱动器 31 的分段的例程始于任务 1082 并继续至任务 1094。首先在 1084 处测试硬盘驱动器 31 以确定是否已用一种本技术专家所周知的硬盘驱动器压缩例程对它进行了压缩。

如果硬盘驱动器 31 未被压缩，则在 1086 处用本技术专家所周知的消除分段实用程序来消除整个硬盘驱动器 31 的分段。此后，上述例程在 1088 处返回以便在 1090 处重新开始分配例程的分配部分。

如果硬盘驱动器 31 是压缩的，则在 1092 处将其压缩部分减

至最少。此后，在 1094 处用本技术专家所周知的消除分段实用程序来消除硬盘驱动器 31 未被压缩的部分的分段。然后，所述例程在 1088 处返回以便在 1090 处重新开始分配例程的分配部分。

在 1070 处调用的用于增加硬盘驱动器 31 上可用空间的例程始于任务 1100 并继续至任务 1110。首先，在 1102 处测试硬盘驱动器 31 以确定是否用一种本技术专家所周知的硬盘驱动器压缩例程对它进行了压缩。

如果硬盘驱动器 31 未被压缩，则它没有可用于保存文件的足够空间，因而在 1104 处显示一报文以通知用户使用挂起或继续特征，用户必须增加额外的硬盘驱动器容量或者从硬盘驱动器 31 上删除一些文件。

如果硬盘驱动器 31 是压缩的，则如果可能的话，在 1108 处增加其未被压缩部分的长度。此后，所述例程在 1110 处返回以便在 1090 处重新开始分配例程的分配部分。

参照图 17，它显示了退出待机状态的例程，该例程始于 1120 处。从概念上讲，计算机系统在退出待机状态 152 时会反向改变系统从正常操作状态 150 转换至待机状态 152 时所产生的变化。简要地说，计算机系统在退出待机状态 152 时会作下列工作：恢复视频信号；使 LED 23 发光；使硬盘驱动器 31 内的硬盘起转；恢复系统时钟；禁止 APM CPU 空闲调用，因此，来自 APM 驱动器的 CPU 空闲调用不再会使 CPU 40 停机；以及清除指示计算机系统 10 处于待机状态 152 的标记。

首先，所述例程在 1122 处测试是否在计算机系统进入待机状态 152 时产生了一检查点 (Checkpoint)。如果是这样的话，则在

1124 处清除检查点采用位以表明该检查点不再有效。在本特定的实施例中，当计算机系统退出待机时，该检查点会失效。如果计算机系统在处于待机状态时失效，则检查点数据只用于使该系统继续，这是因为，多数系统都使用硬盘驱动器上的虚拟交换文件，并且，从该检查点数据的继续工作会将机器置成这样的状态，在这种状态下，上述交换文件完全不同于作为检查点数据而存储起来的系统状态所预期的交换文件。另外，可在下一次磁盘存取之后使检查点数据失效。此外，在系统从检查点数据继续时，可以在磁盘存取一个可能使计算机系统出现问题的文件之后使检查点失效。再有，如果用户理解从检查点数据的继续工作可能使硬盘驱动器 31 上的部分或全部数据丢失，则用户可以随时使用检查点数据。

此后并且如果未采用检查点，则 CPU 40 在 1126 处命令微控制器 U_2 作下列工作：(i) 使视频控制器 56 再次开始生成视频信号；(ii) 使时钟合成器 906 恢复较高的系统时钟频率（25MHz 或 33MHz）；以及 (iii) 使 LED 33 发光。然后，CPU 40 在 1128 处将适当的值写至固定盘控制器 86 以使硬盘驱动器 31 的硬盘开始旋转。随后，在 1130 处禁止 APM CPU 空调用，因此 CPU 不会停机。最后，在 1132 处清除待机标志 (Standby Flag)，这表示计算机系统 10 处于正常操作状态 150，并且，所说的例程在 1140 处返回调用程序。

参照图 18，它显示了进入待机状态例程，该例程始于 1140。简要地说，计算机系统在进入待机状态 152 时会做下列工作：断开视频信号；使 LED 23 闪亮；使硬盘驱动器 31 内的硬盘停转；减

慢系统时钟；使APM CPU 空调用生效，因此，来自 APM 驱动程序的 CPU 空调用会使 CPU 40 停机；以及设置指示计算机系统 10 处于待机状态 152 的标志。

首先，上述例程在 1142 处测试是否采用了一检查点。如果是，就在 1144 处执行挂起例程的大部分内容，以便将计算机系统 10 的状态存储到硬盘驱动器 31。本实施例中，在计算机系统进行待机时采用检查点。另外，可按与图 17 有关的内容中所讨论的注意事项来周期性地采用检查点并用该检查点使计算机系统继续。然后，在 1146 处执行足够的继续例程以便从 1144 处产生的局部挂起中退出。此后，在 1148 处设置检查点采用位以表示采用了有效的检查点。在本实施例中，仅在处于待机状态 152 时系统出现故障的情况下才使用检查点数据。在这种情况下，系统在引导时会从所保存的检查点处继续。

理想地，上述检查点对计算机系统是完全透明的。因此，如果产生硬件中断以防数据丢失，则应使检查点失效。另外，同正常挂起一样，可以忽略任何硬件中断。

此后并且如果未采用检查点，则 CPU 40 在 1150 处命令微控制器做下列工作：(i) 令视频控制器 56 停止生成视频信号；(ii) 使时钟合成器 906 将系统时钟从其高频 (25MHz 或 33MHz) 减慢至 8MHz；以及 (iii) 使 LED 23 闪亮。然后，CPU 40 在 1152 处将适当的值写至固定盘控制器 86 以使硬盘驱动器 31 内的硬盘停转。随后，在 1154 处使 APM CPU 空调用生效，因此，来自 APM 驱动程序的 CPU 空调用会使 CPU 40 停机。最后，在 1156 处设置待机标志，这表示计算机系统处于待机状态 152，并且，所说的例程在

1158 处返回调用程序。

尽管通过本发明的最佳实施例说明了本发明，并且是较详细地说明这些实施例，但是，本申请人并不想以任何方式把后附的权利要求范围局限于上述细节。本技术的专家将非常清楚其它的优点和改进很容易实现。例如，由电源管理电路 106 所执行的多种任务，如对一个或多个中断的硬件监控等，可内装在系统芯片内。所以，从更广泛的方面来说，本发明并不局限于所显示和说明的特定细节、代表性的设备与方法以及示意性的实例。因此，在不脱离本申请人之发明的总体概念的精神和范围的情况下，可以与以上细节不一致。

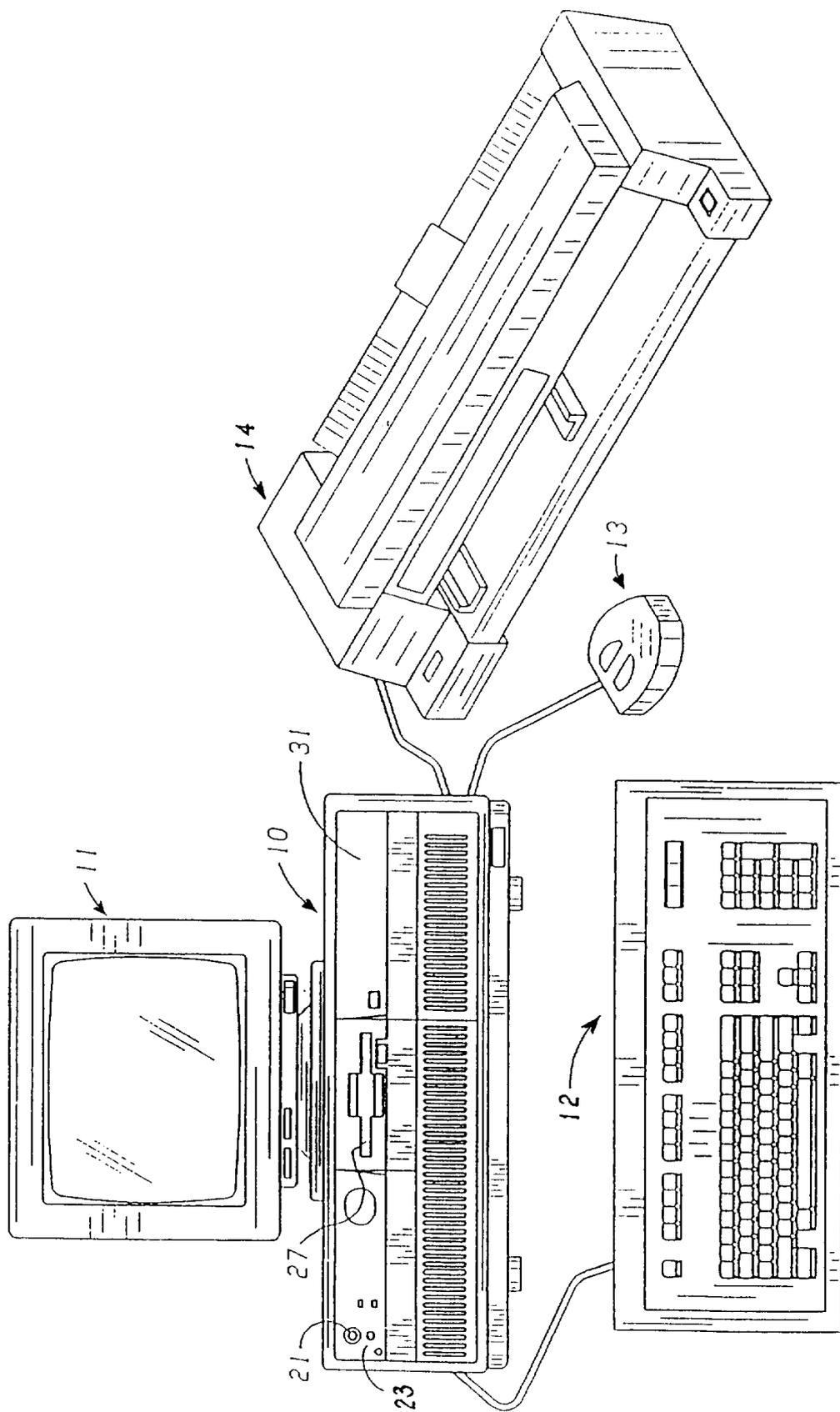


图 1

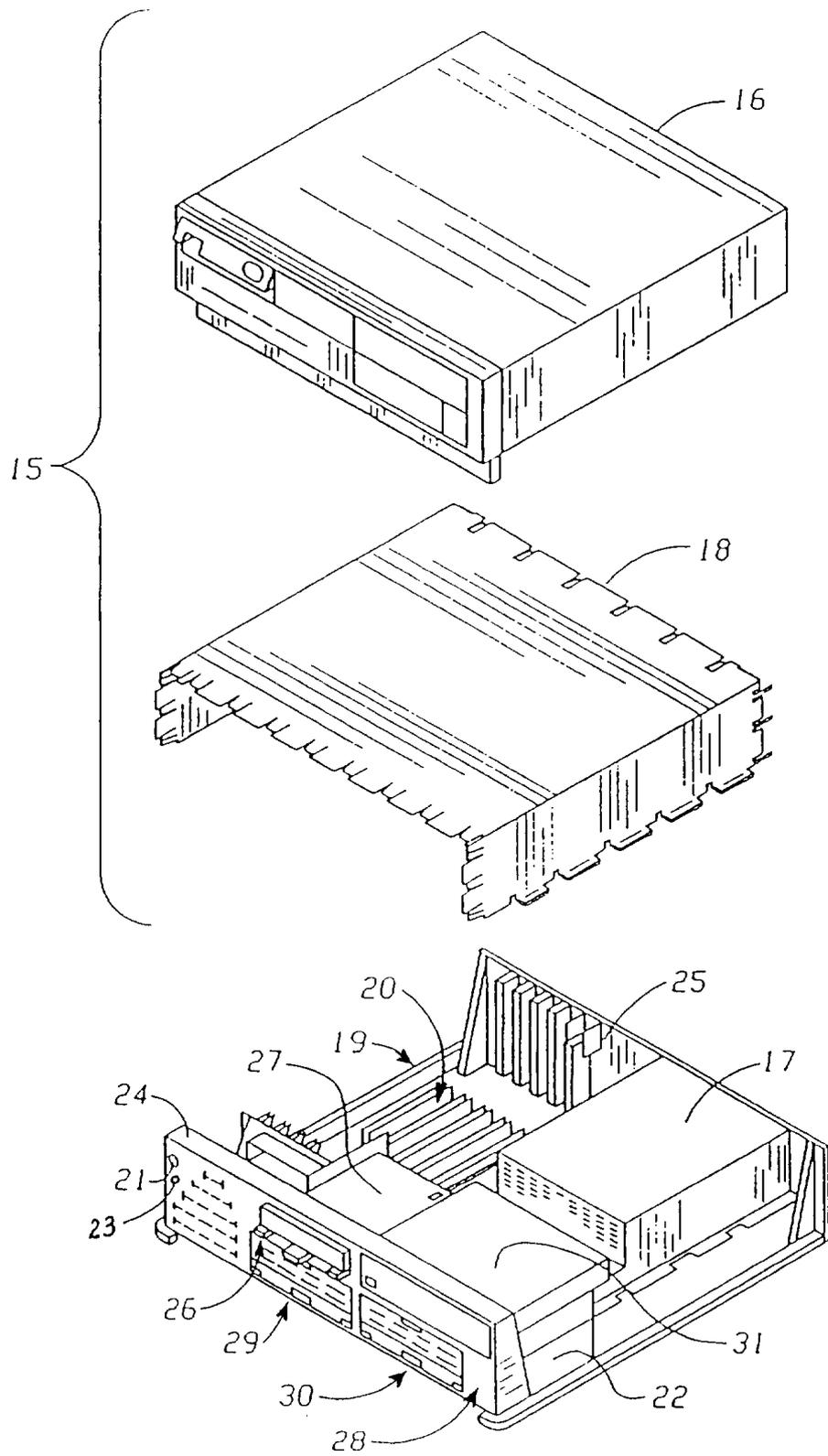


图 2

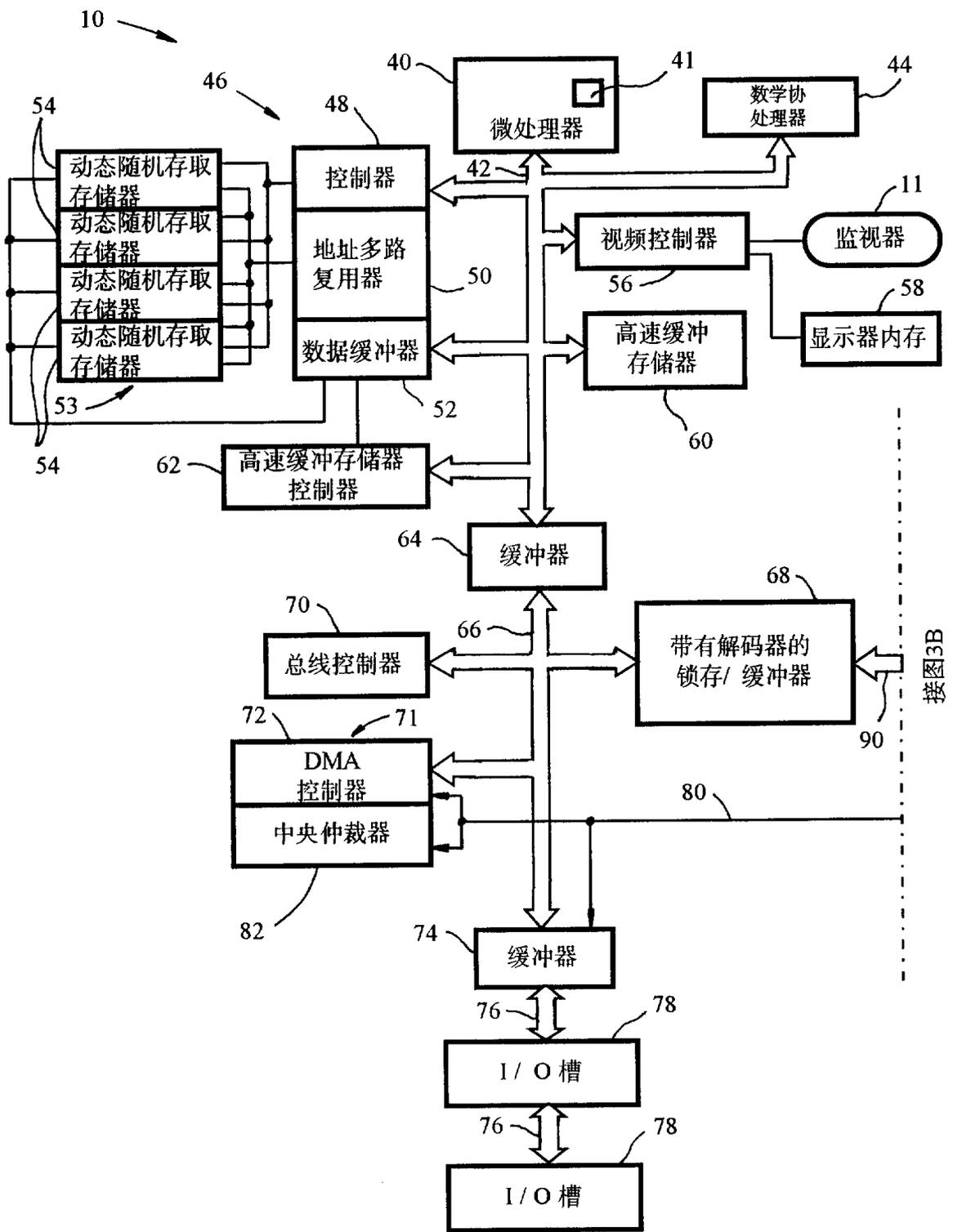


图 3A

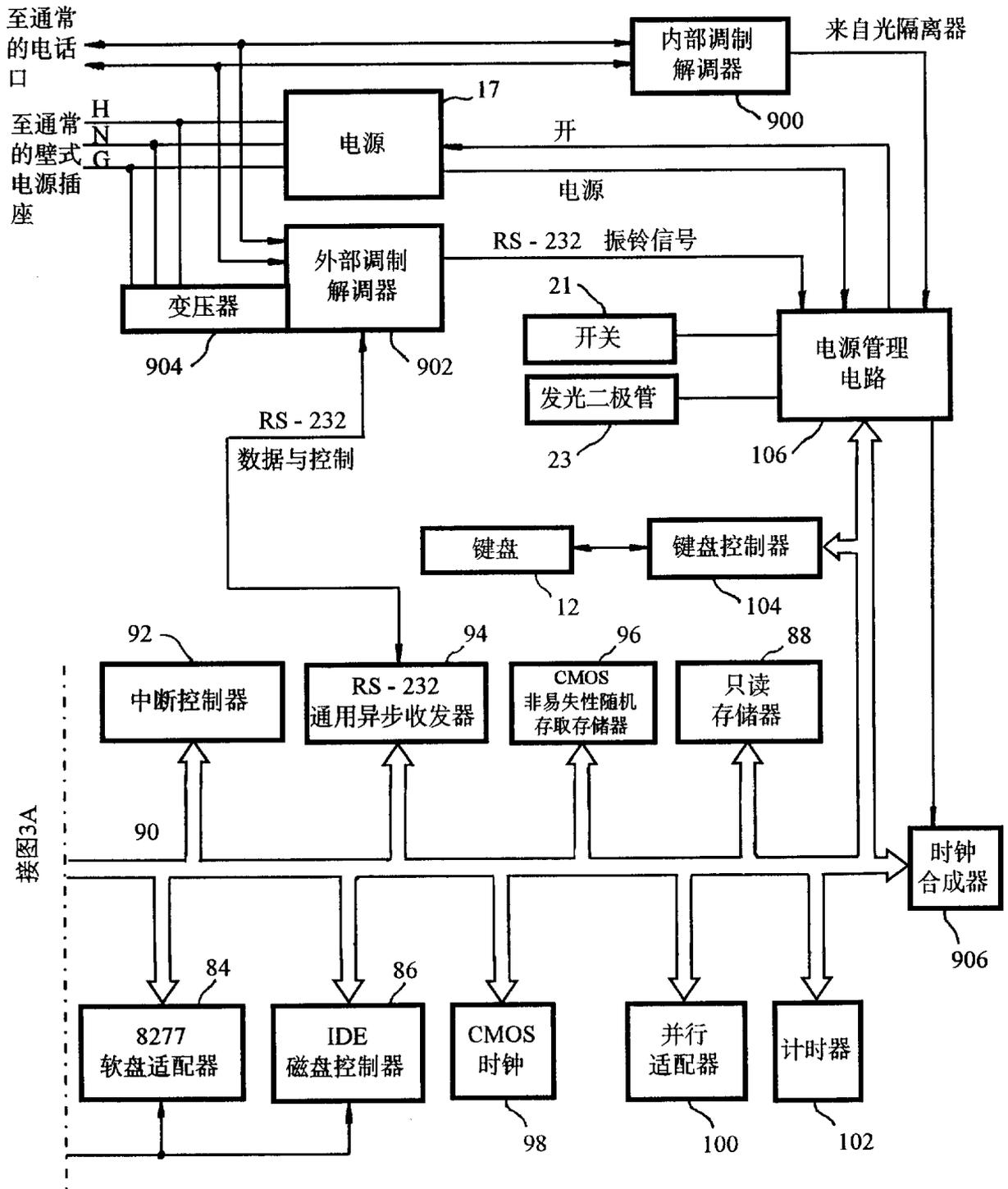


图 3B

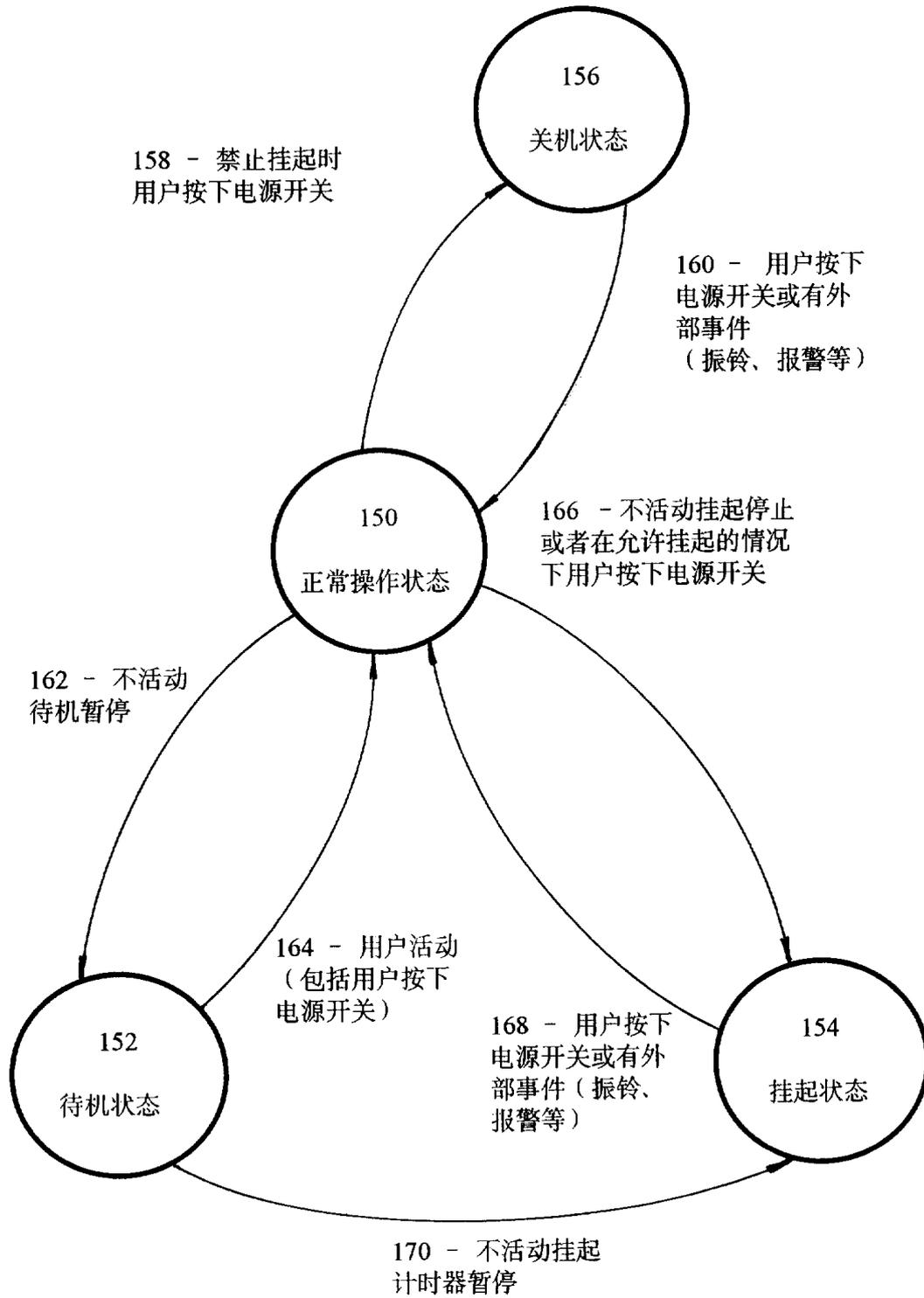


图 4

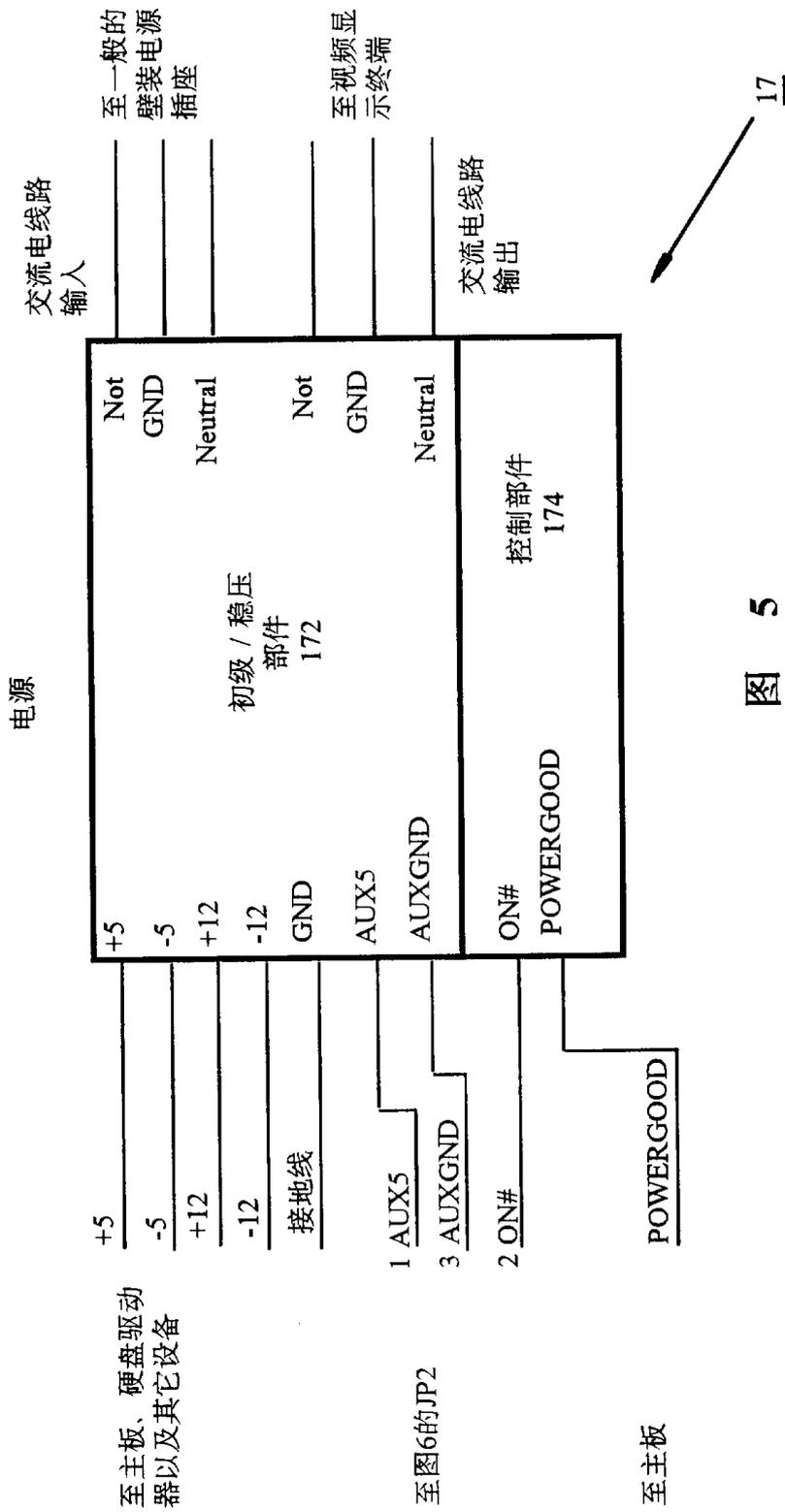


图 5

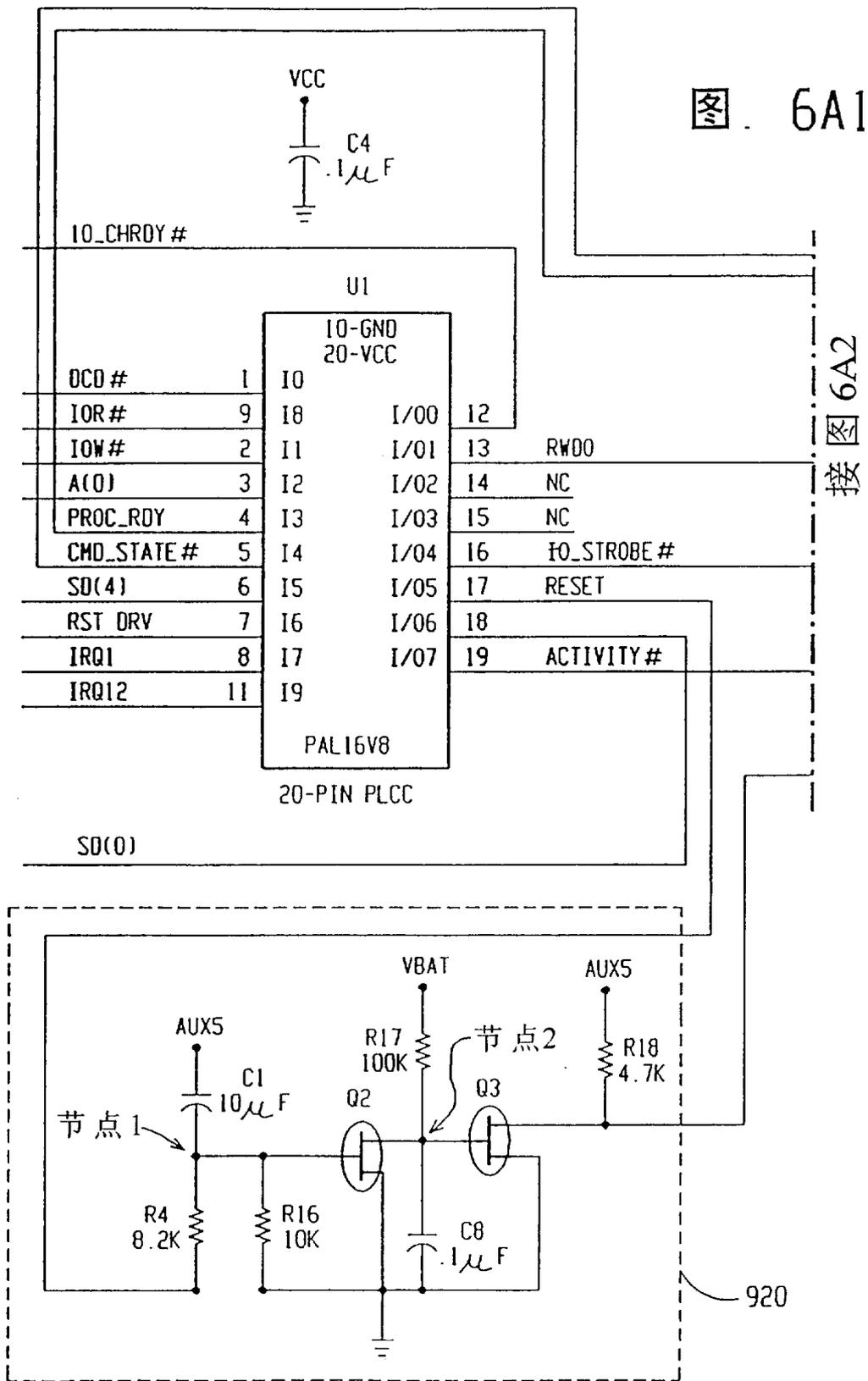


图 6A1

接 图 6A2

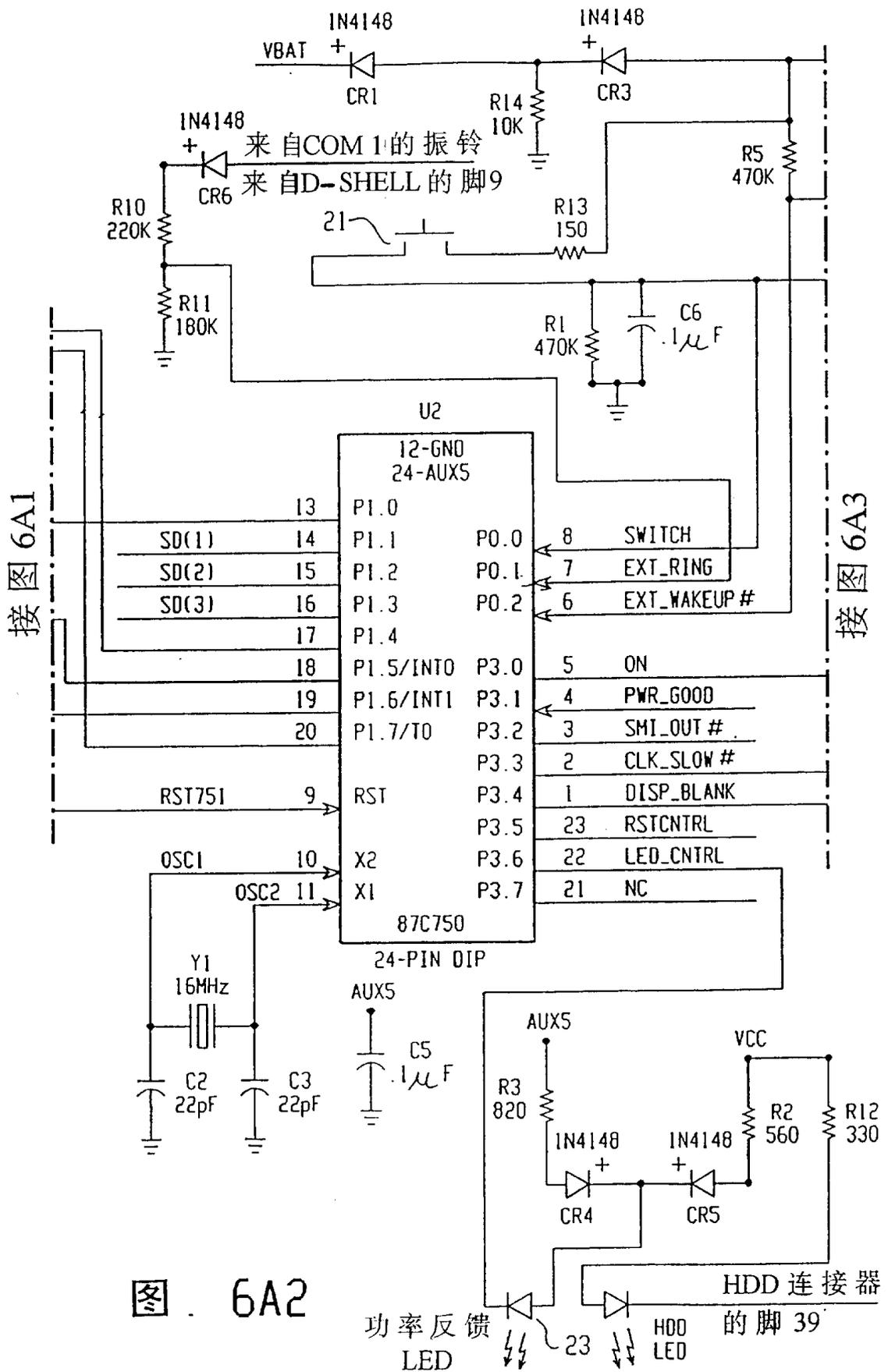


图 6A2

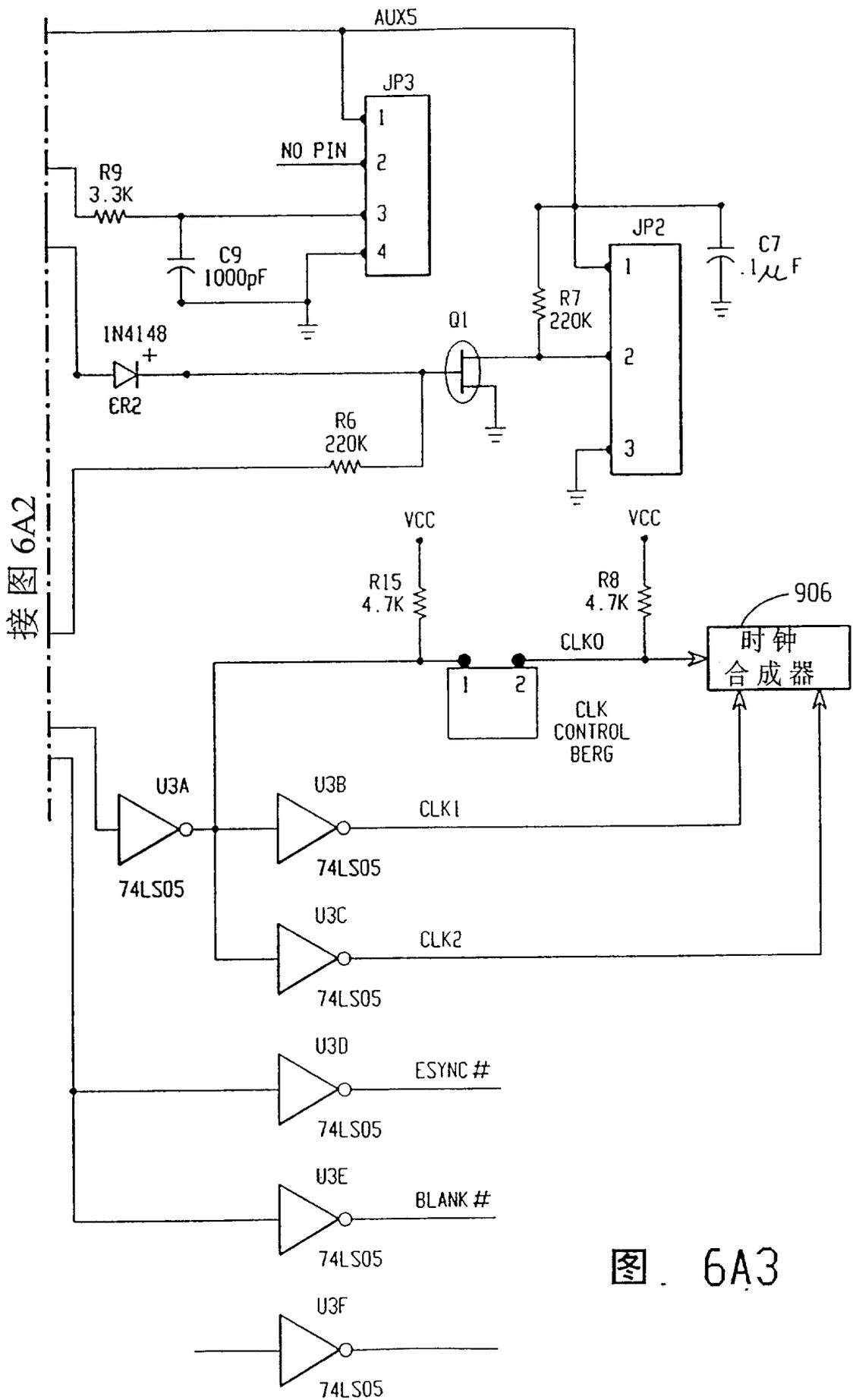


图 6A3

图. 6B

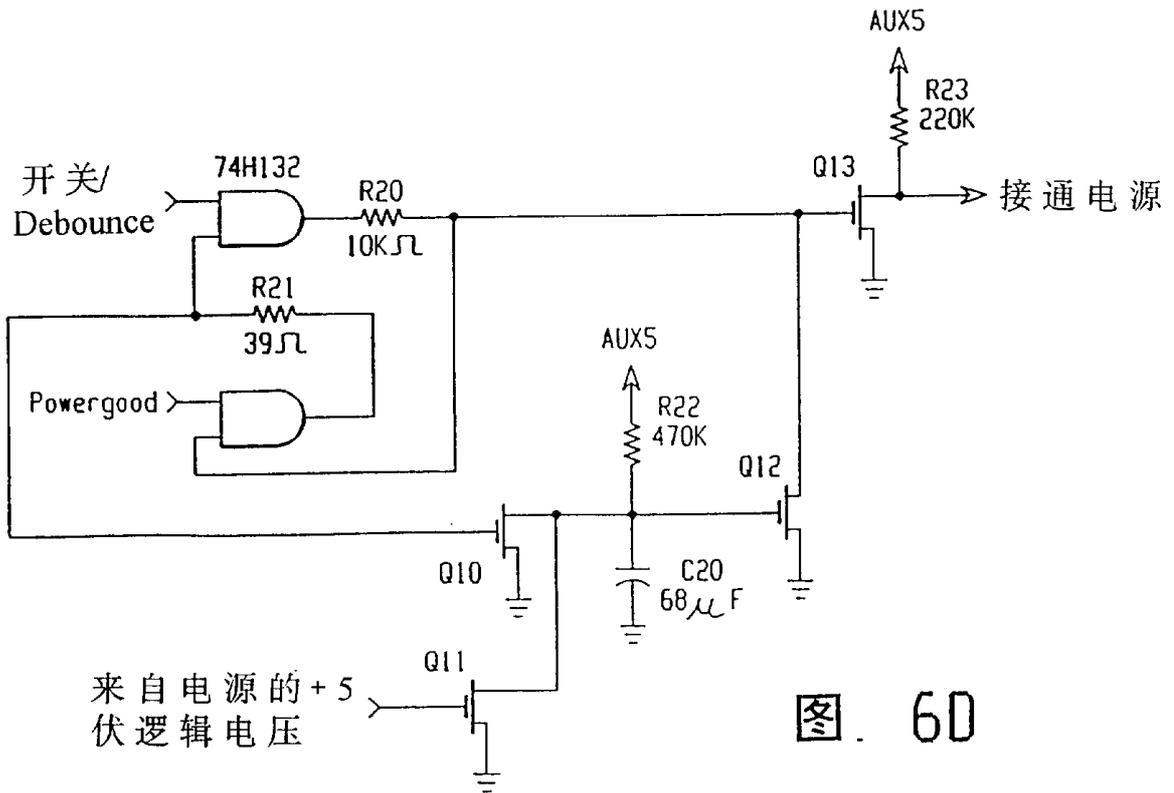
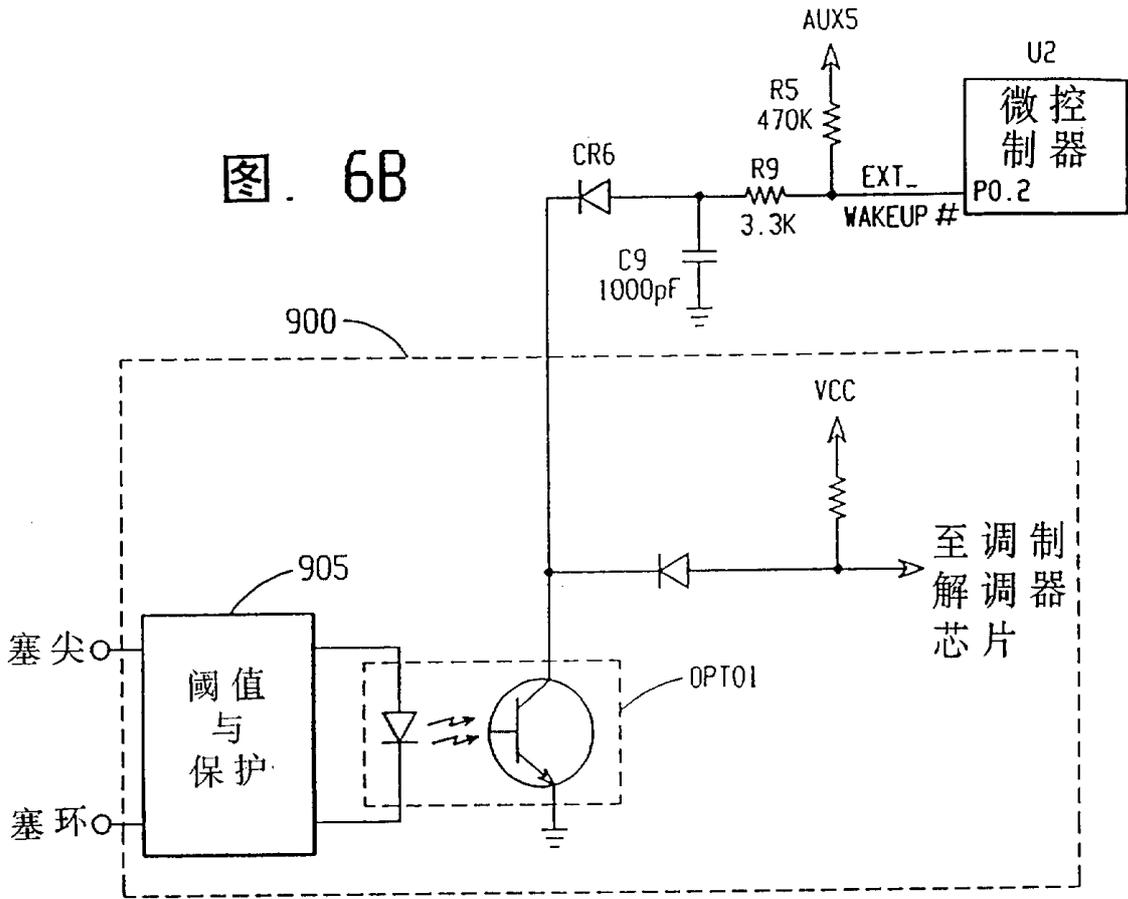


图. 6D

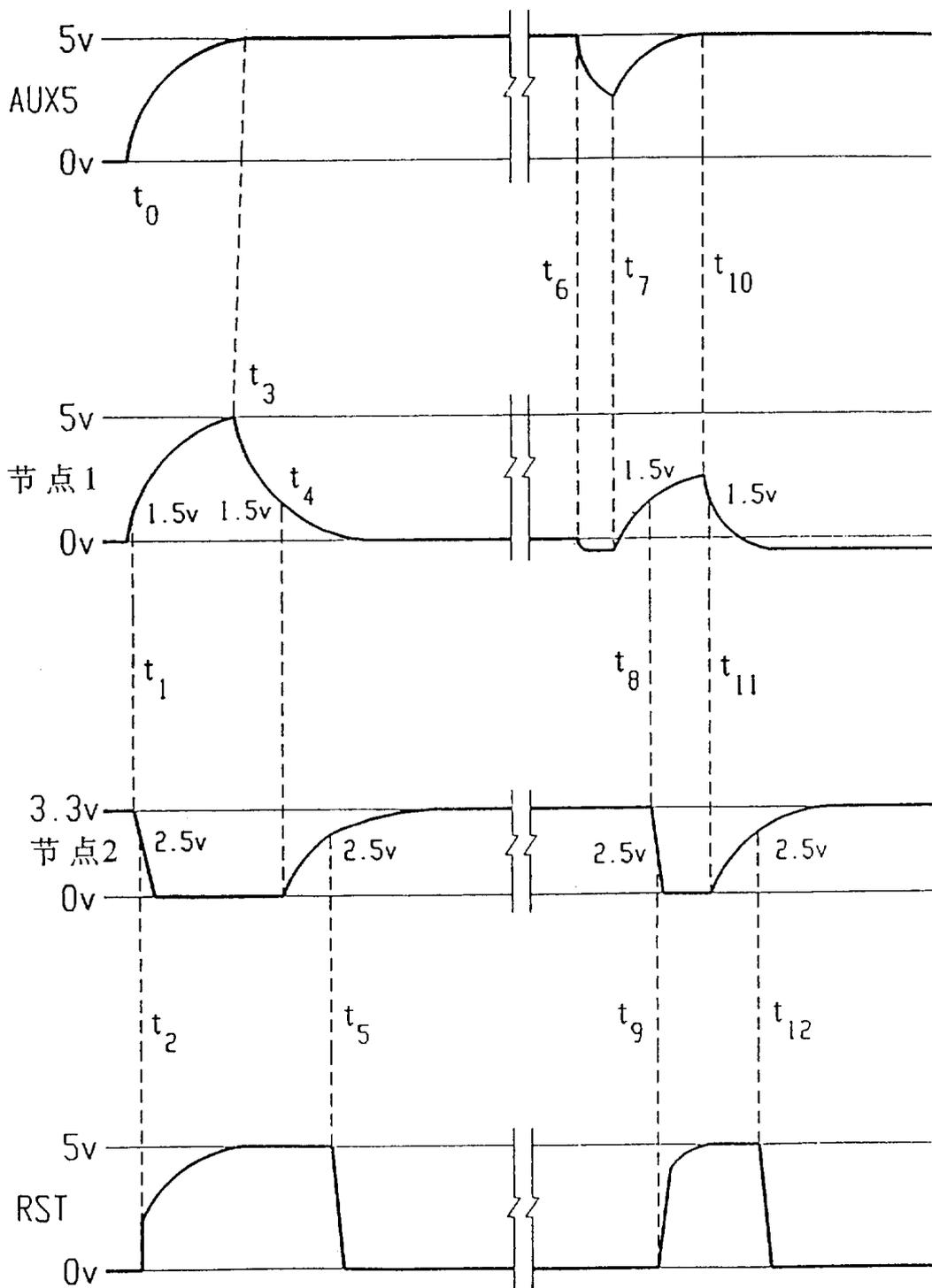


图 . 6C

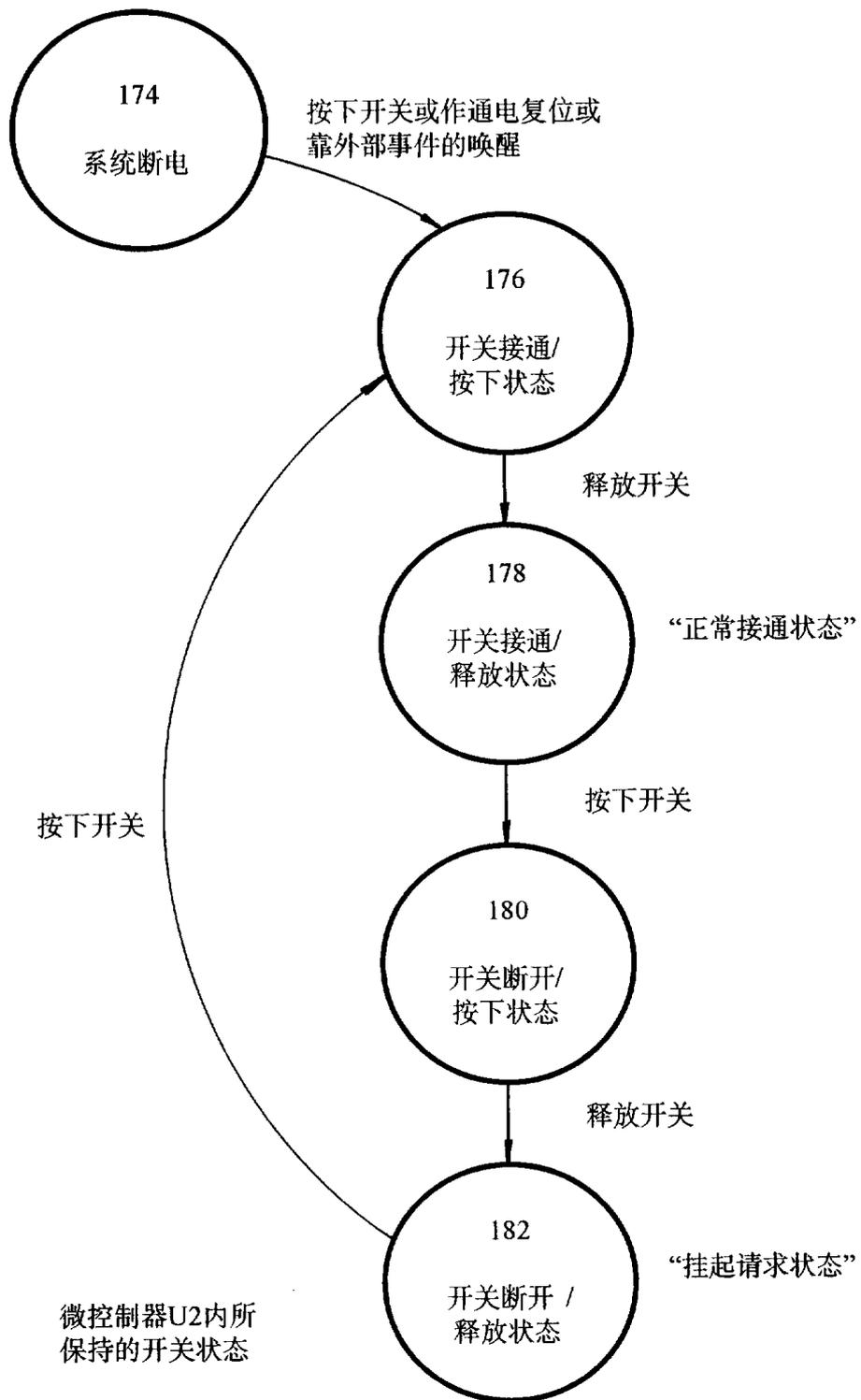


图 7

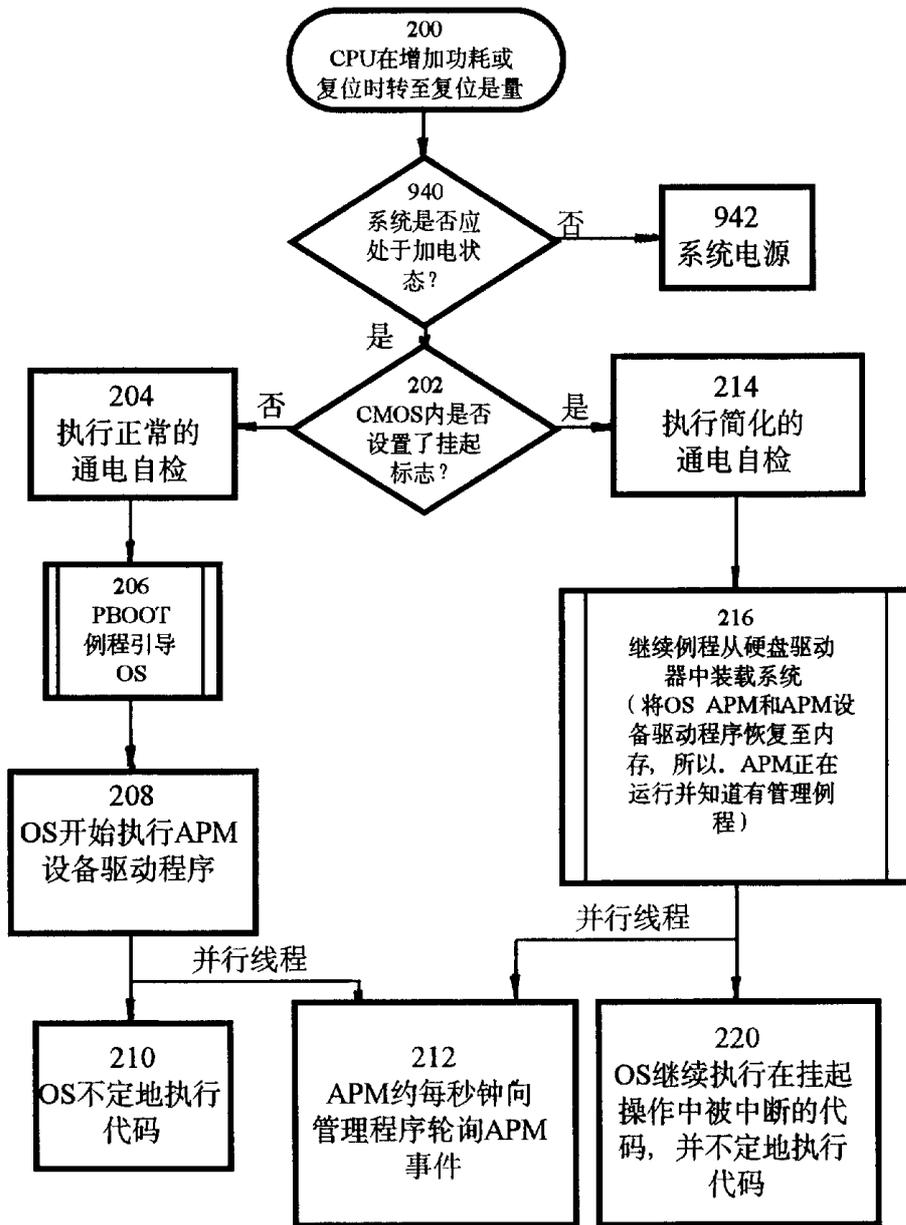


图 8

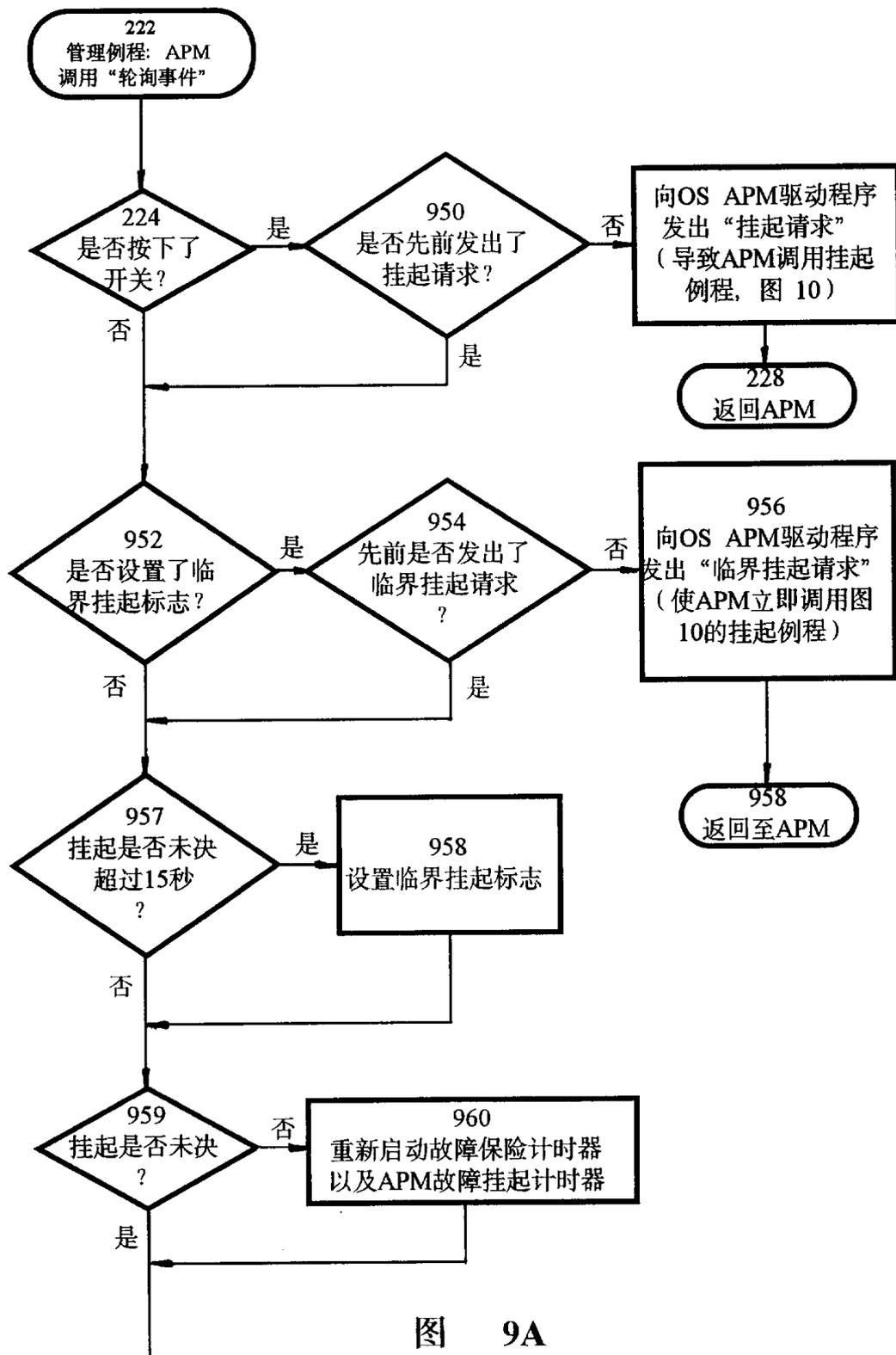


图 9A

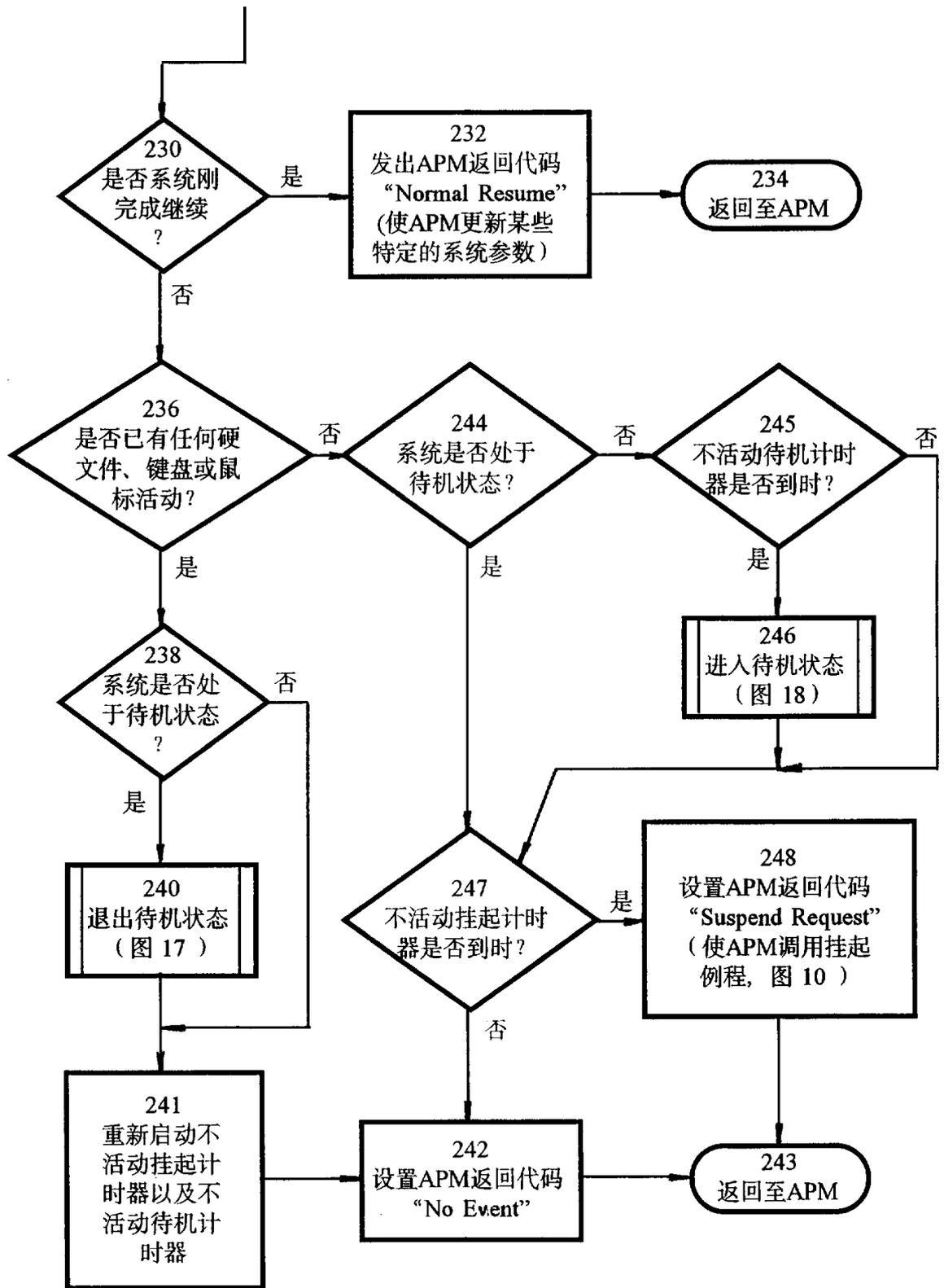


图 9B

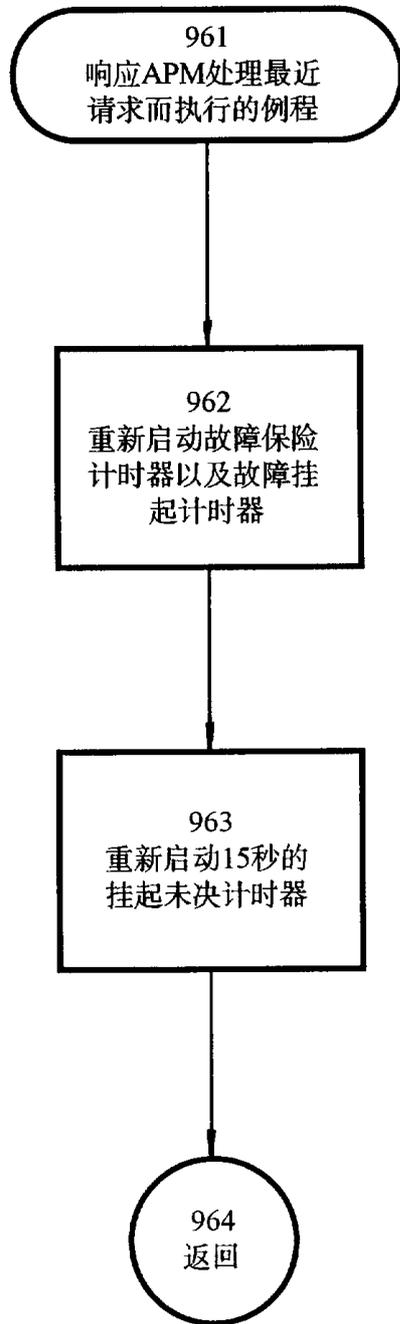


图 9C

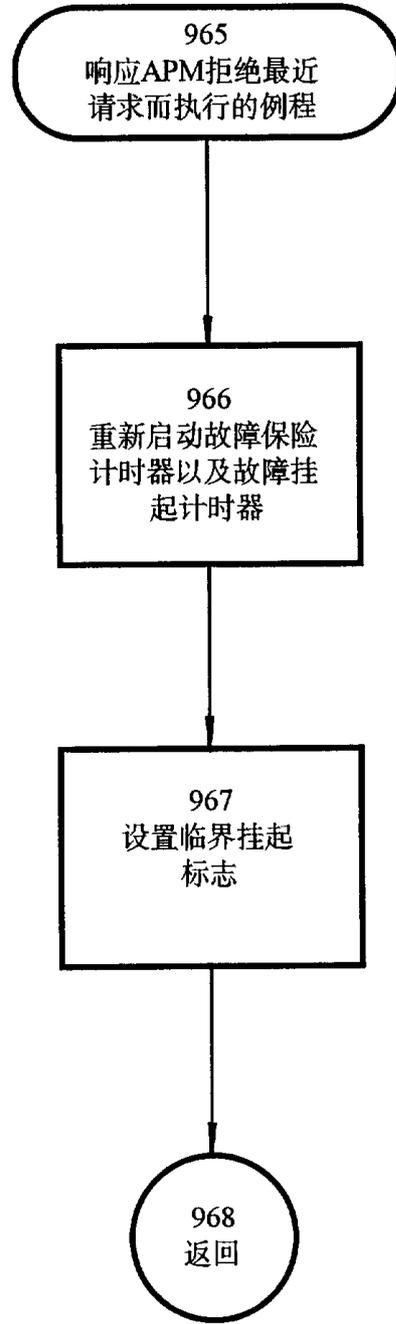


图 9D

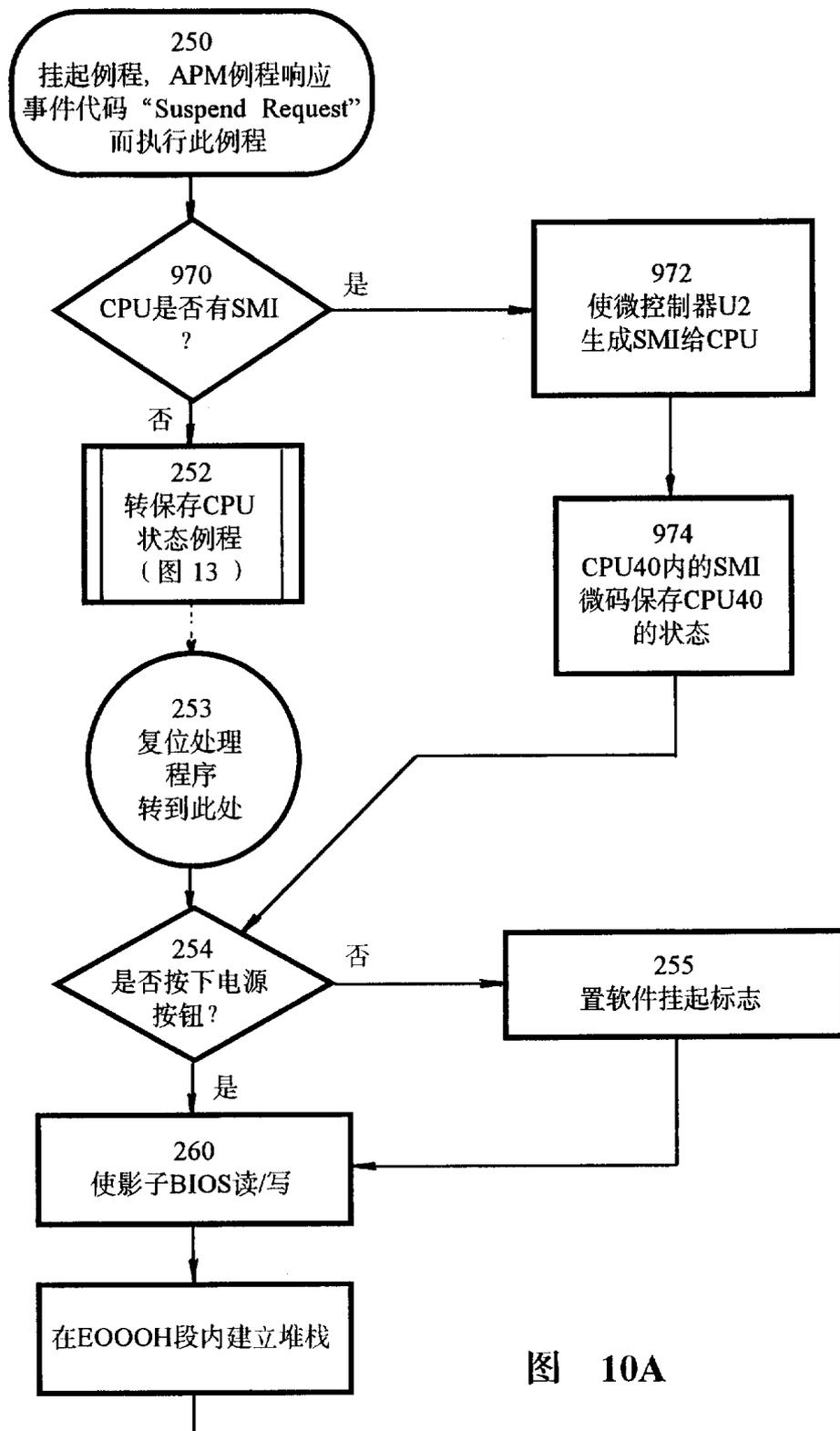


图 10A

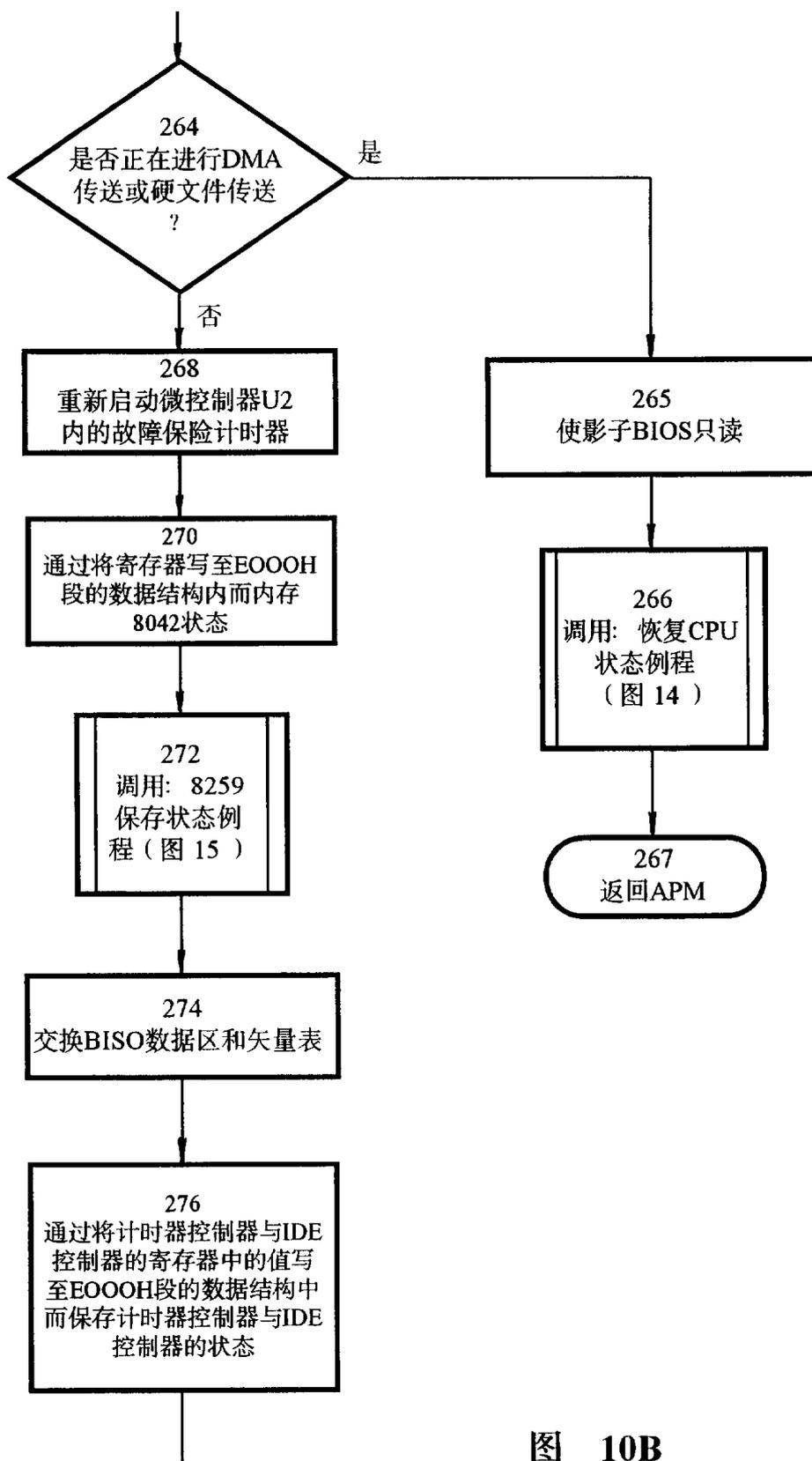


图 10B

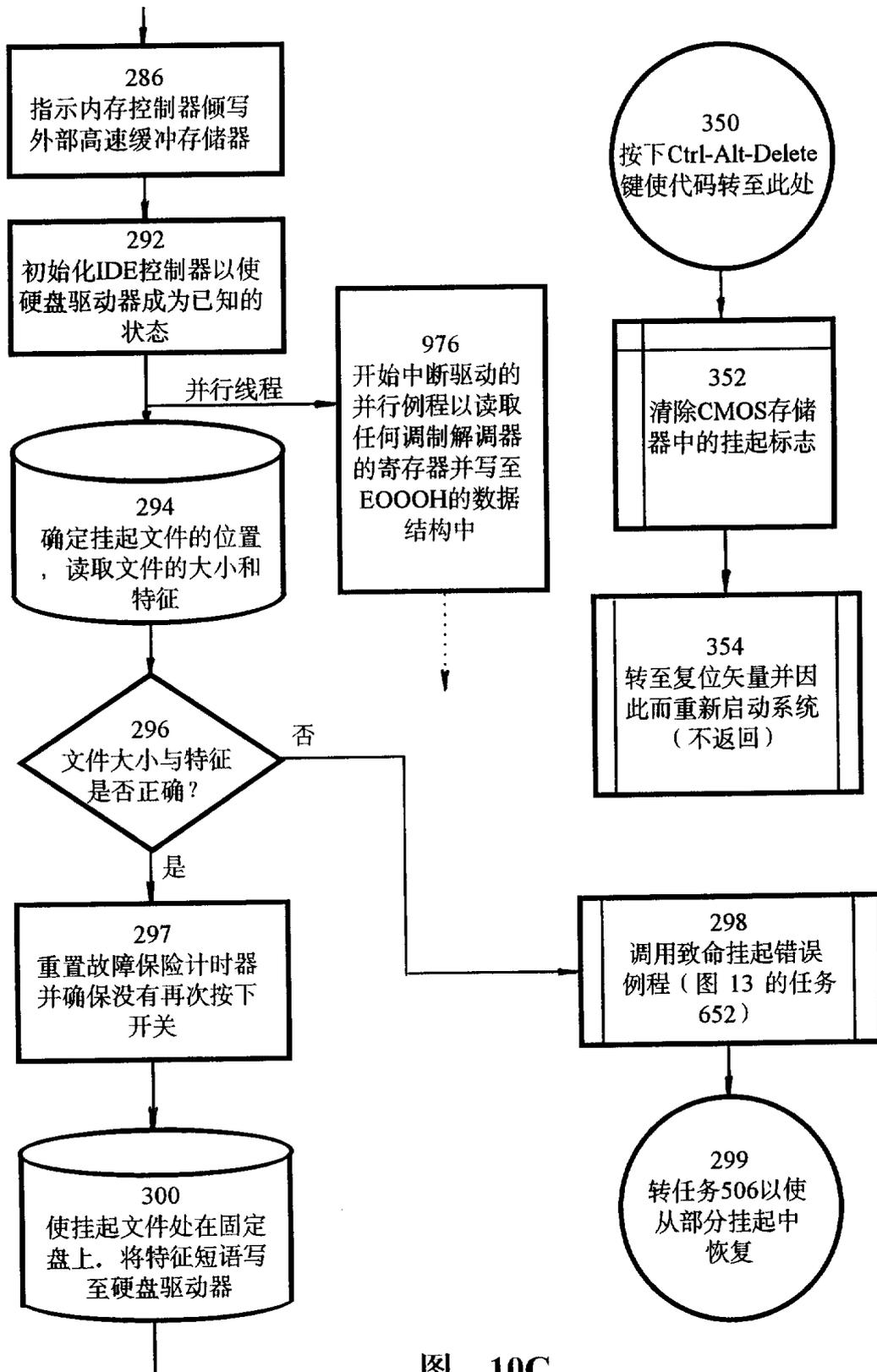


图 10C

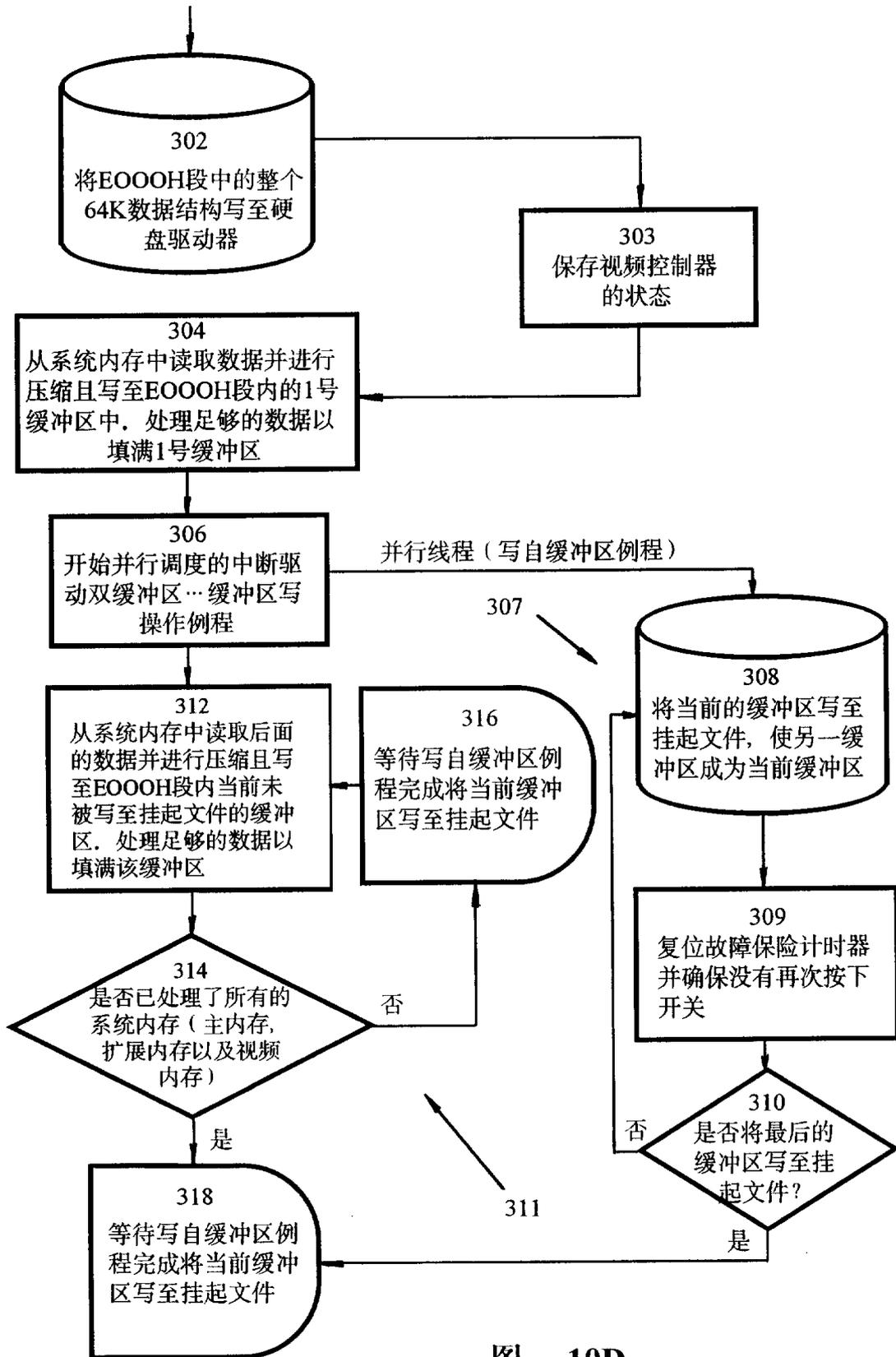


图 10D

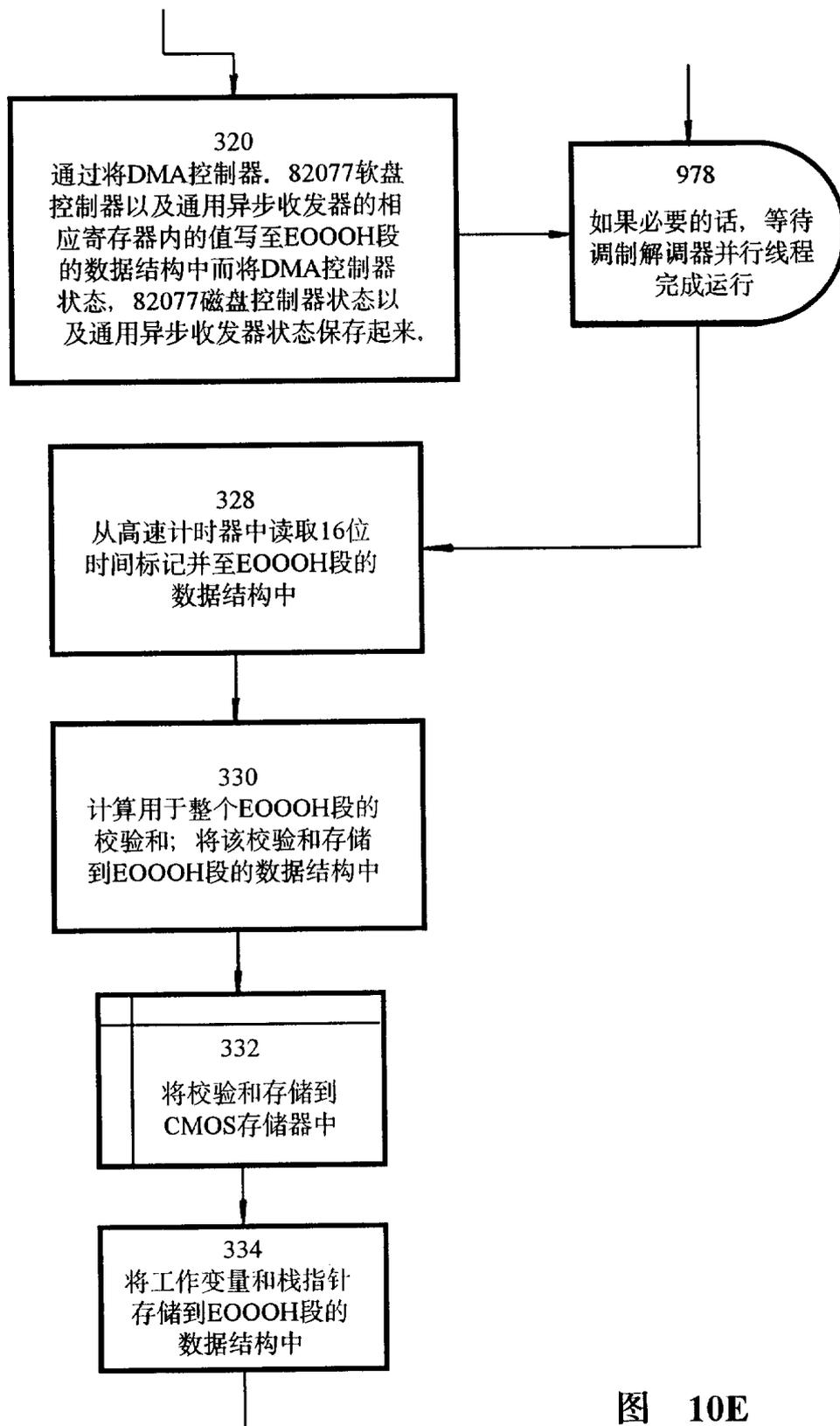


图 10E

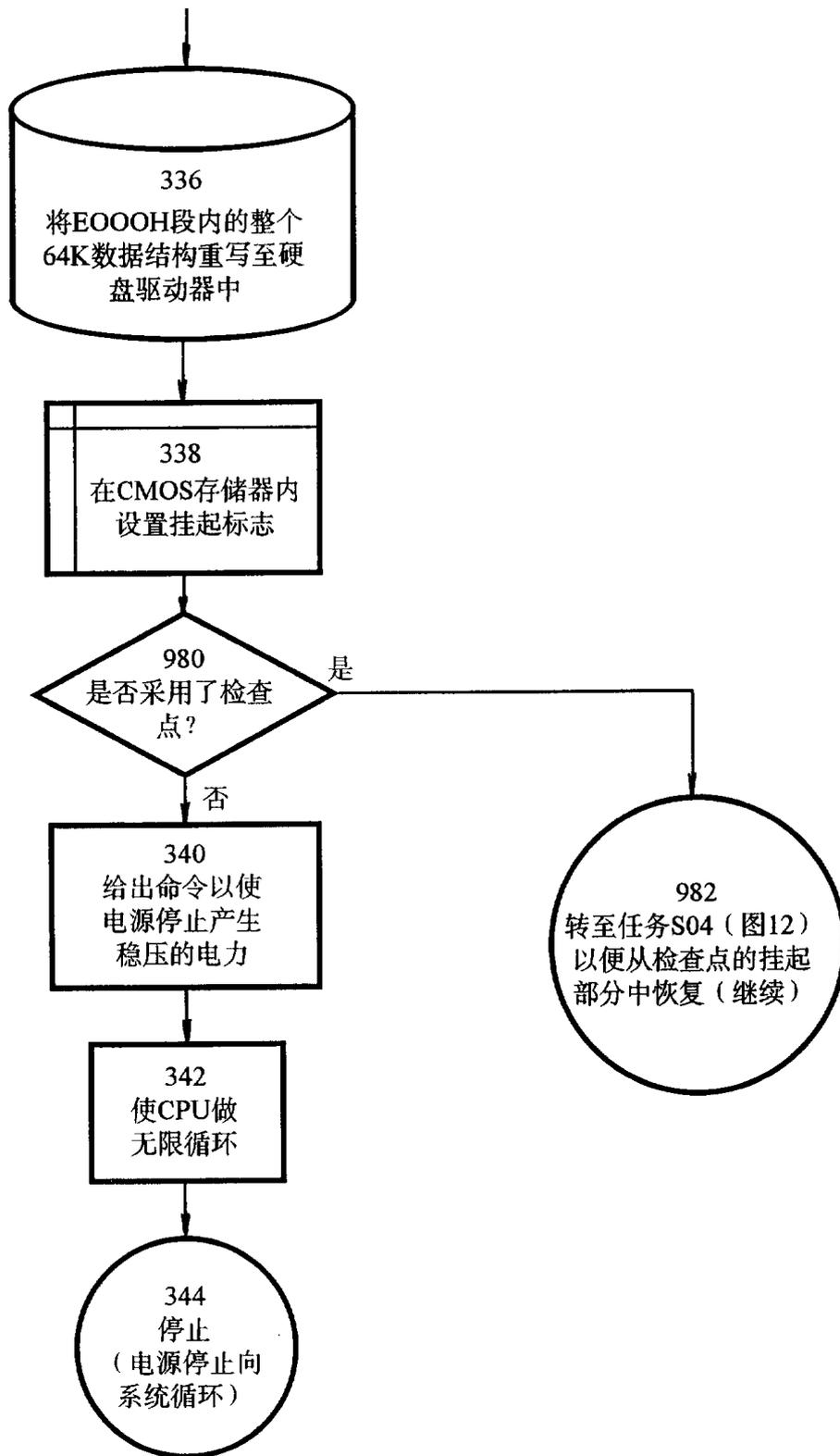


图 10F

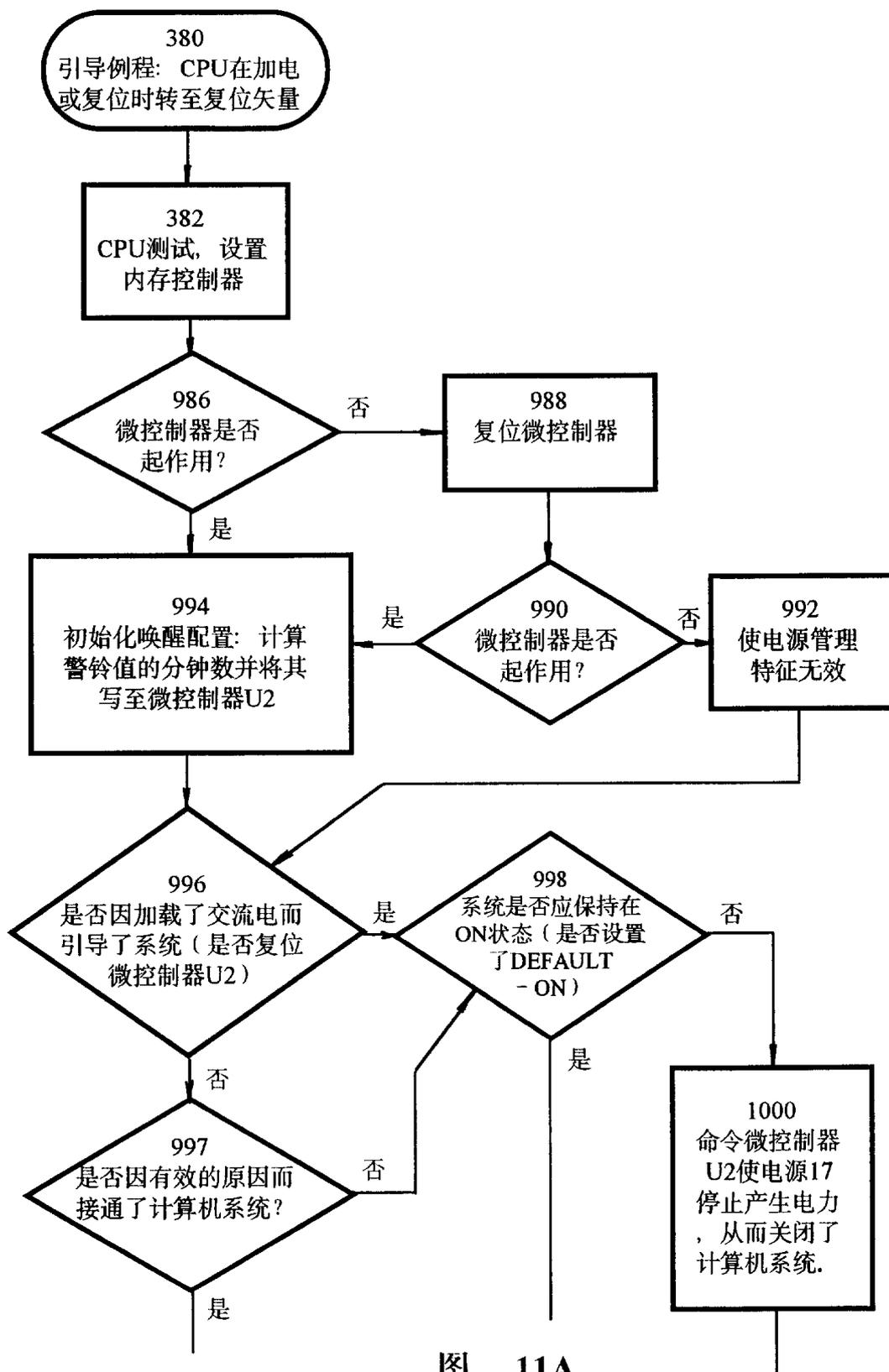


图 11A

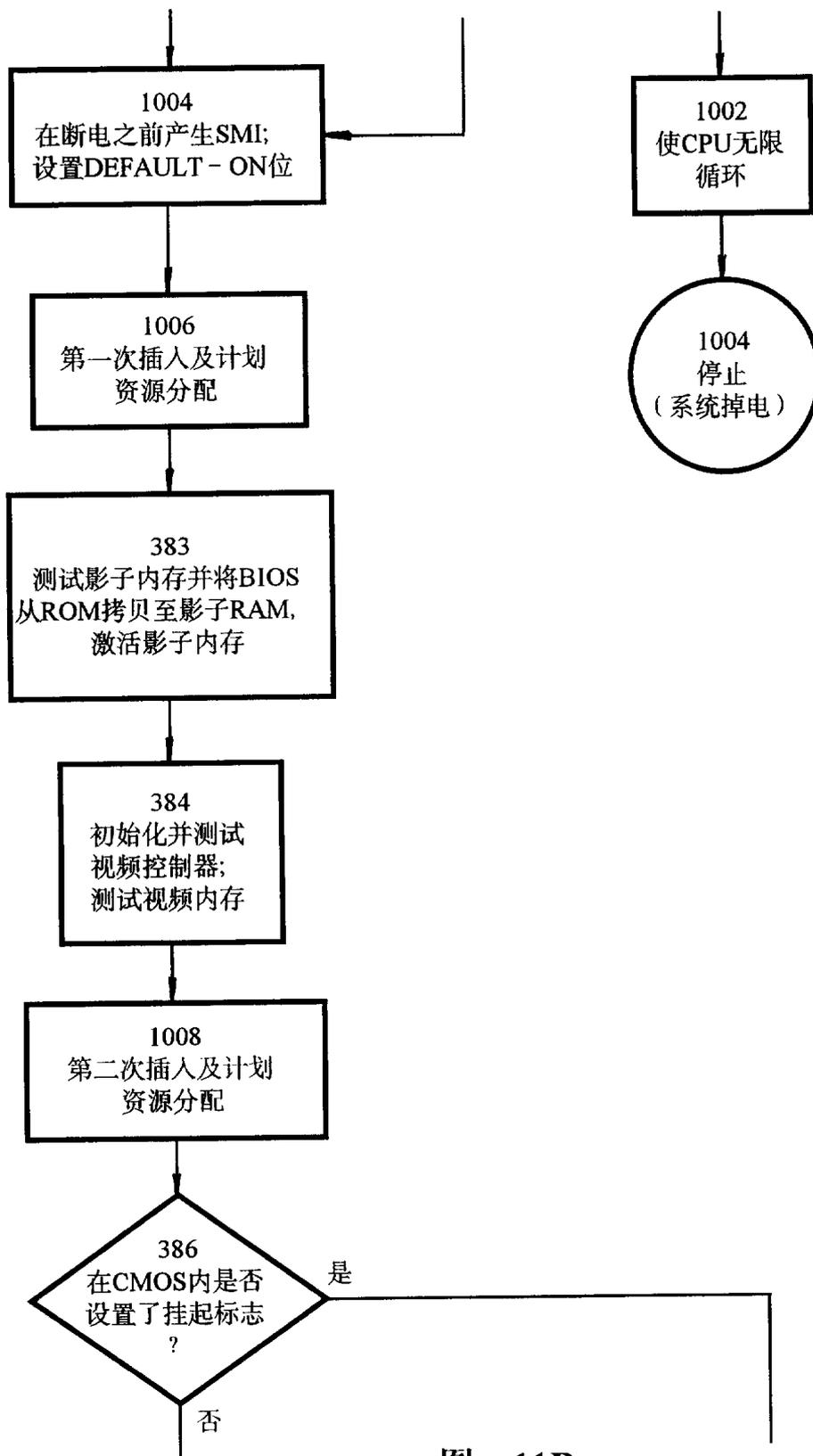


图 11B

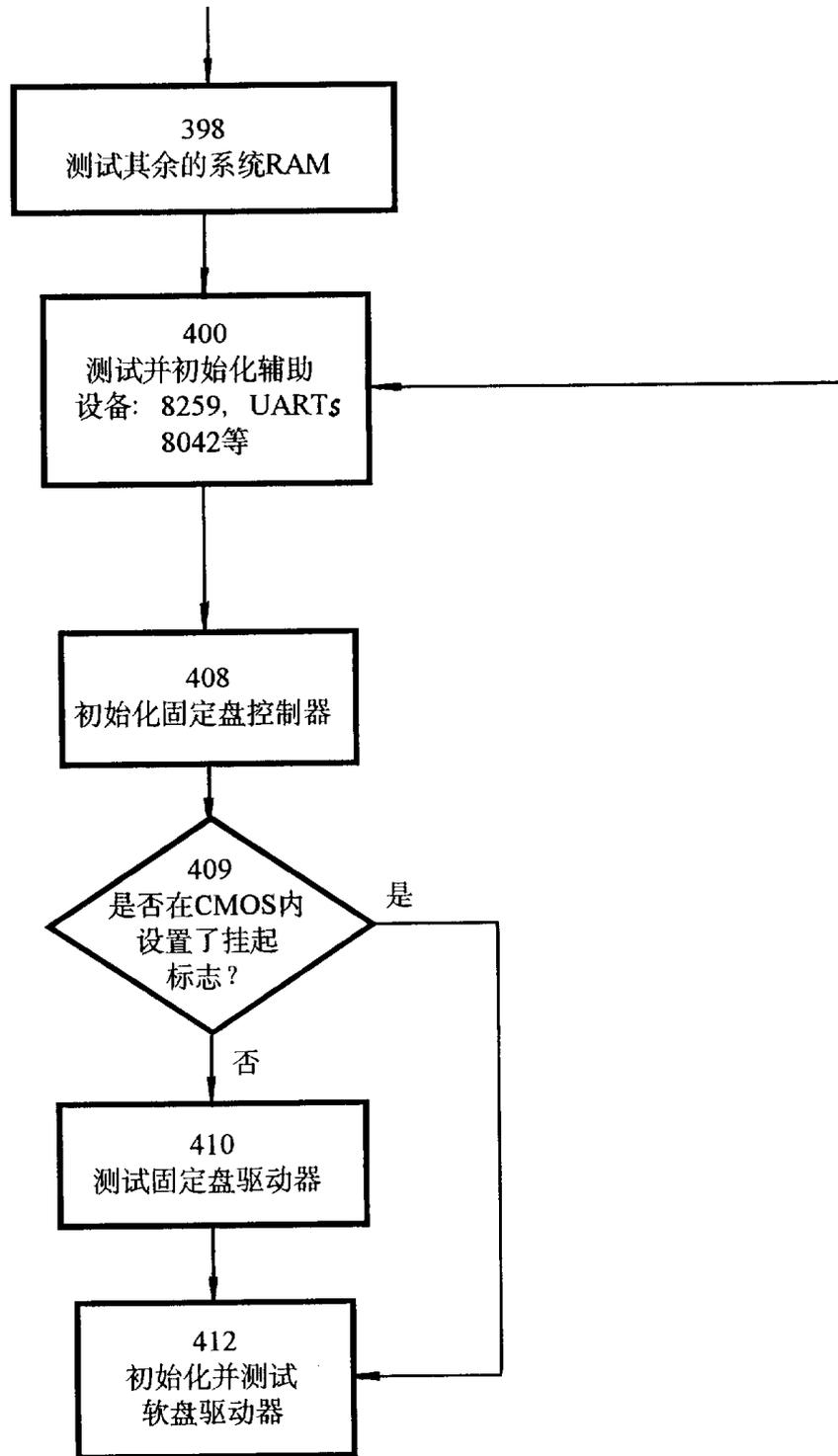


图 11C

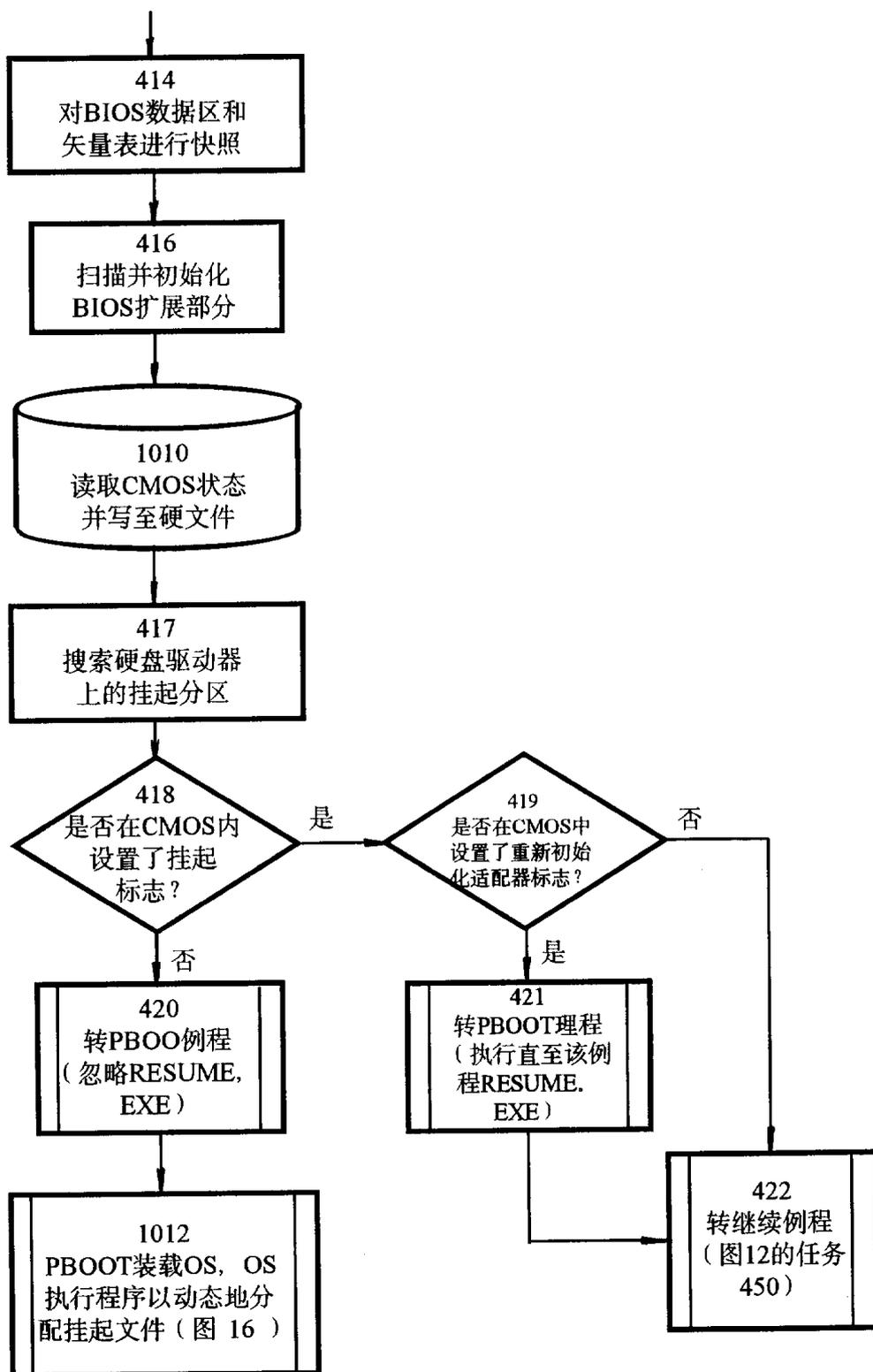


图 11D

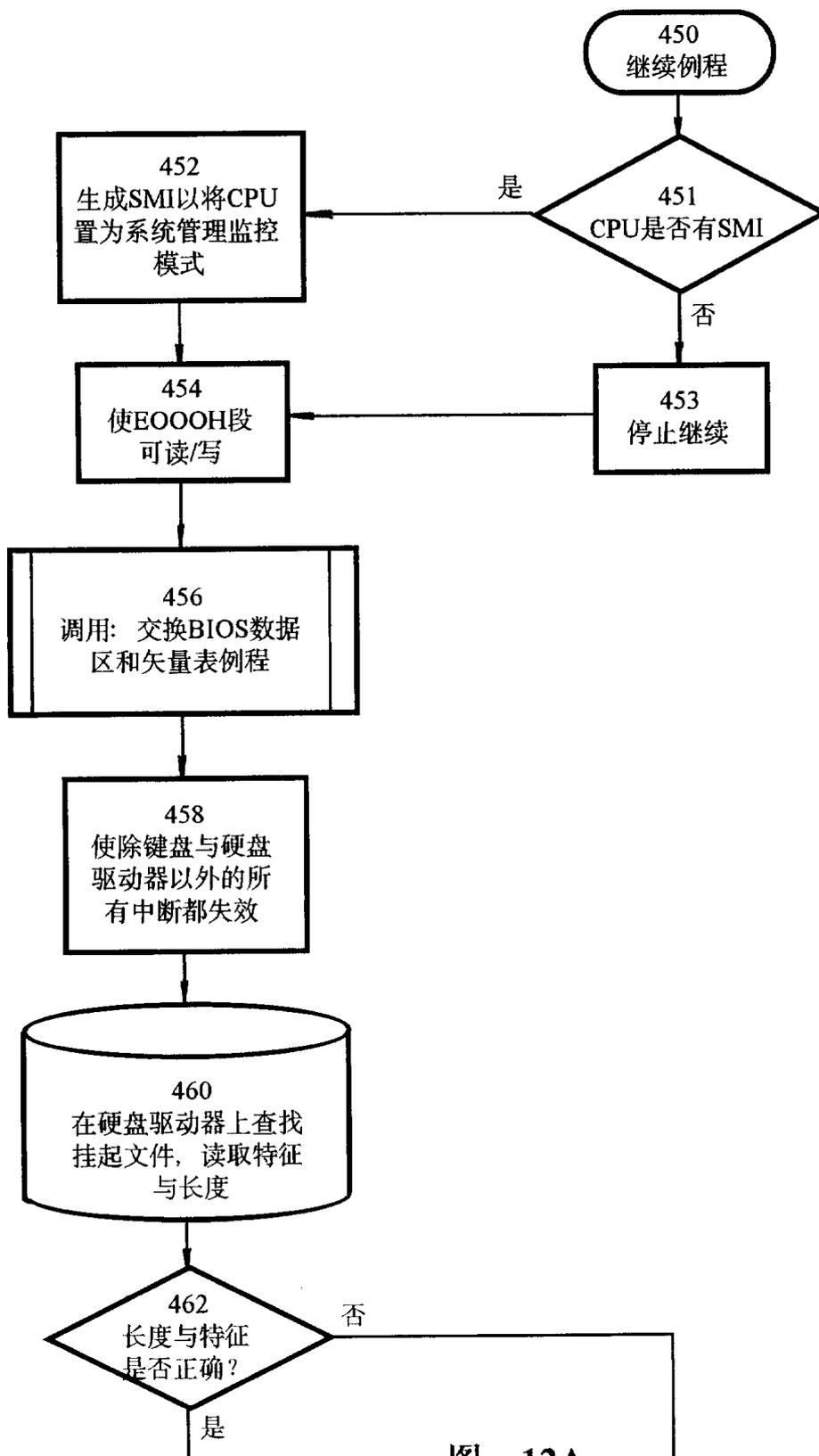


图 12A

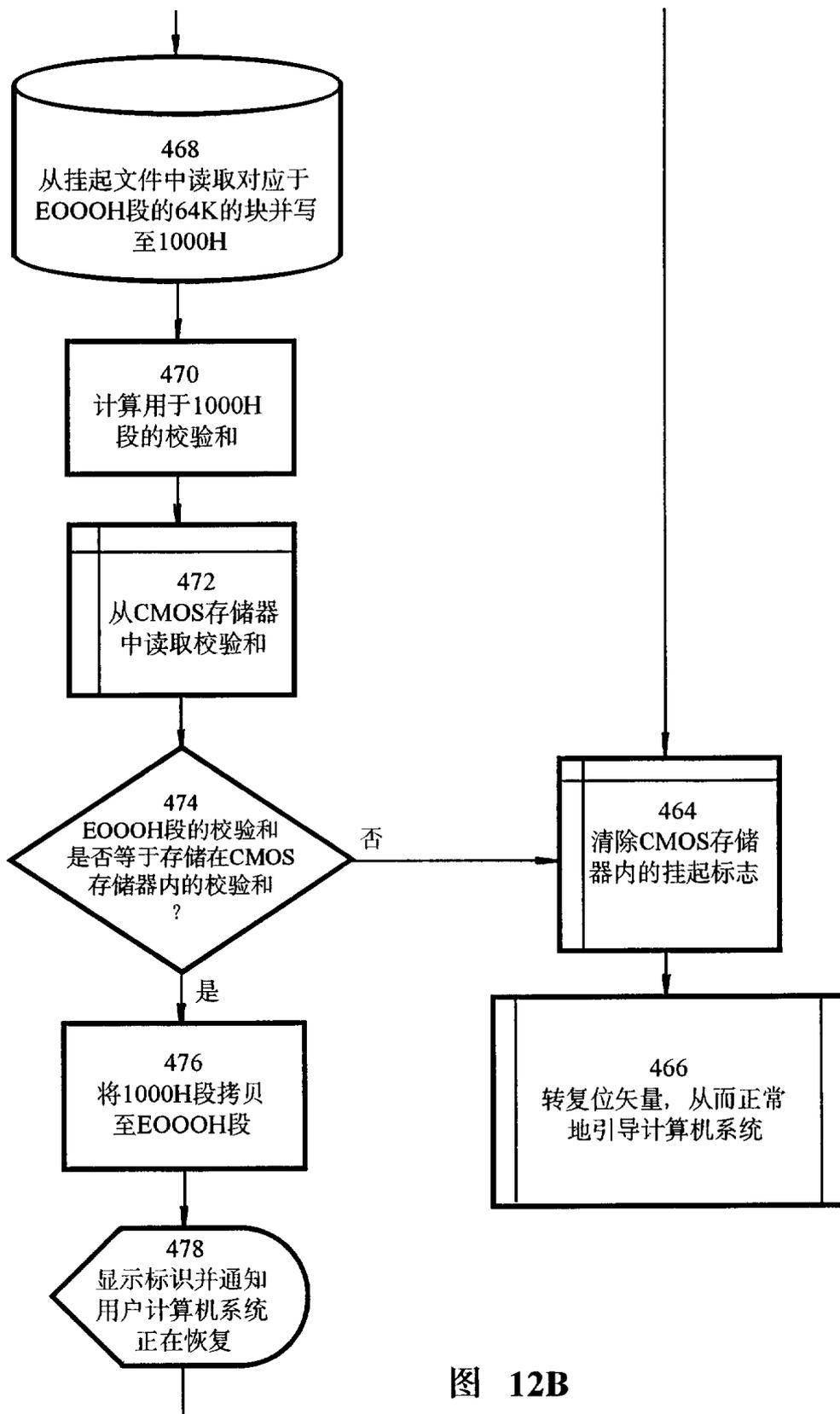


图 12B

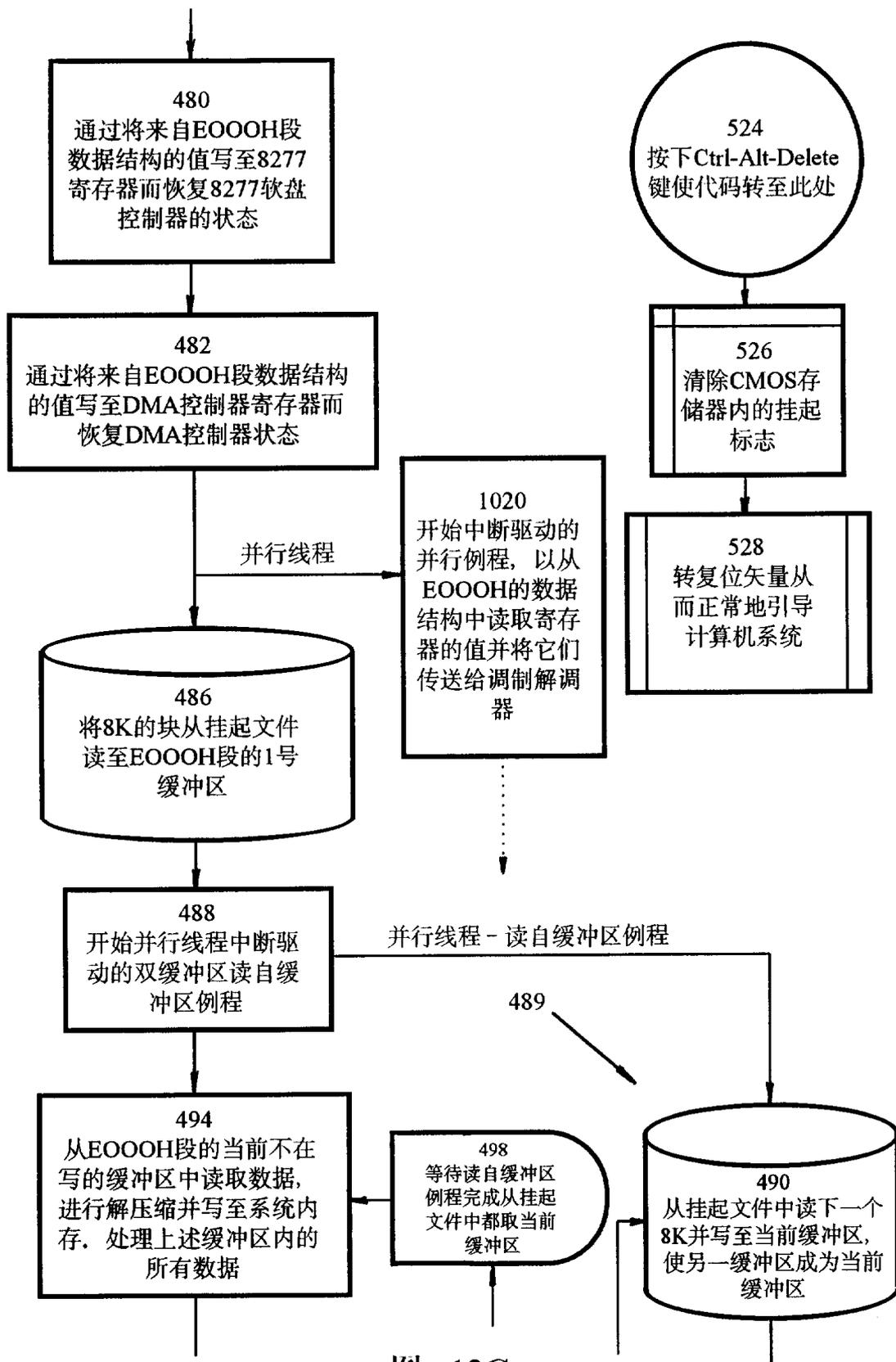


图 12C

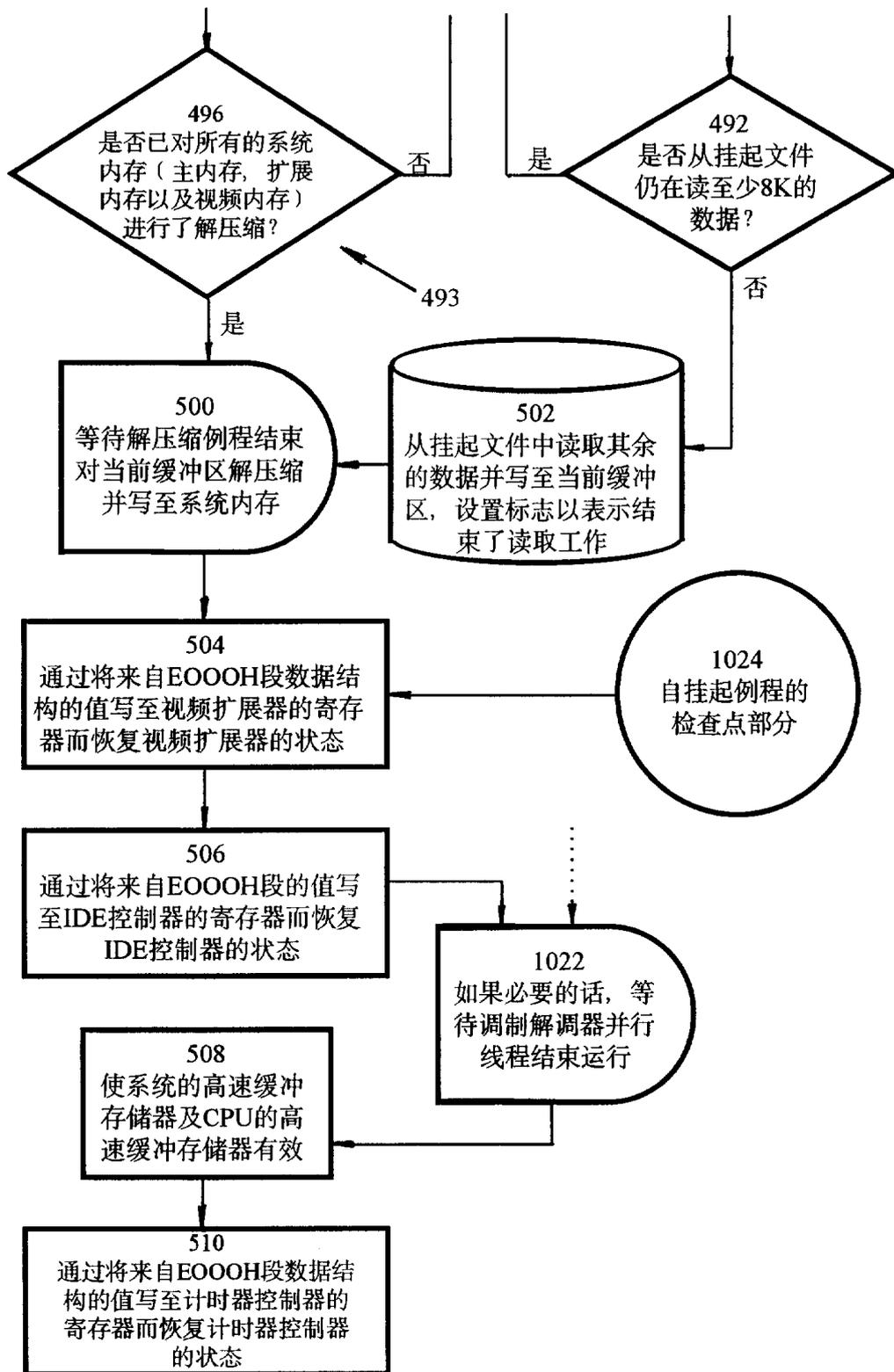


图 12D

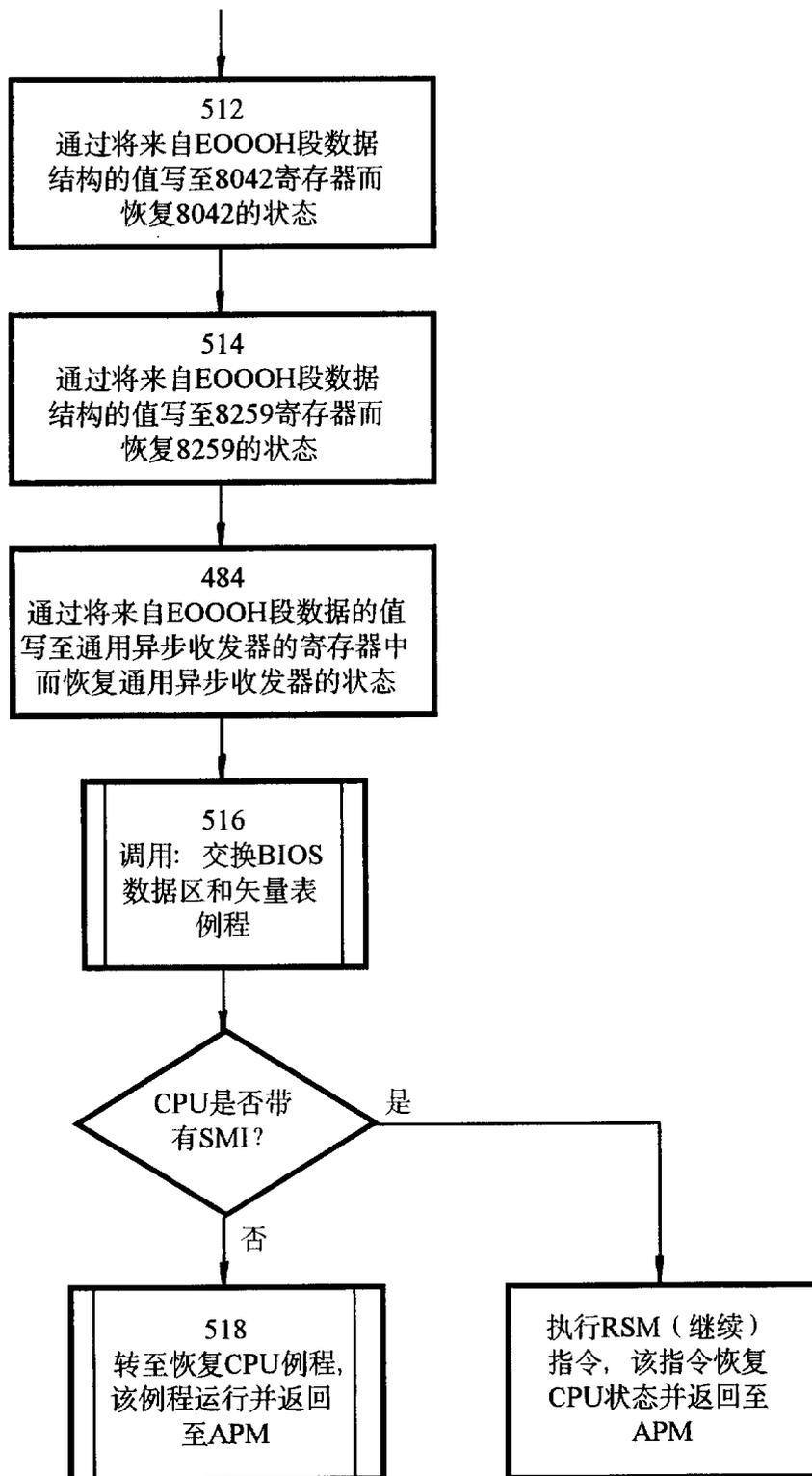


图 12E

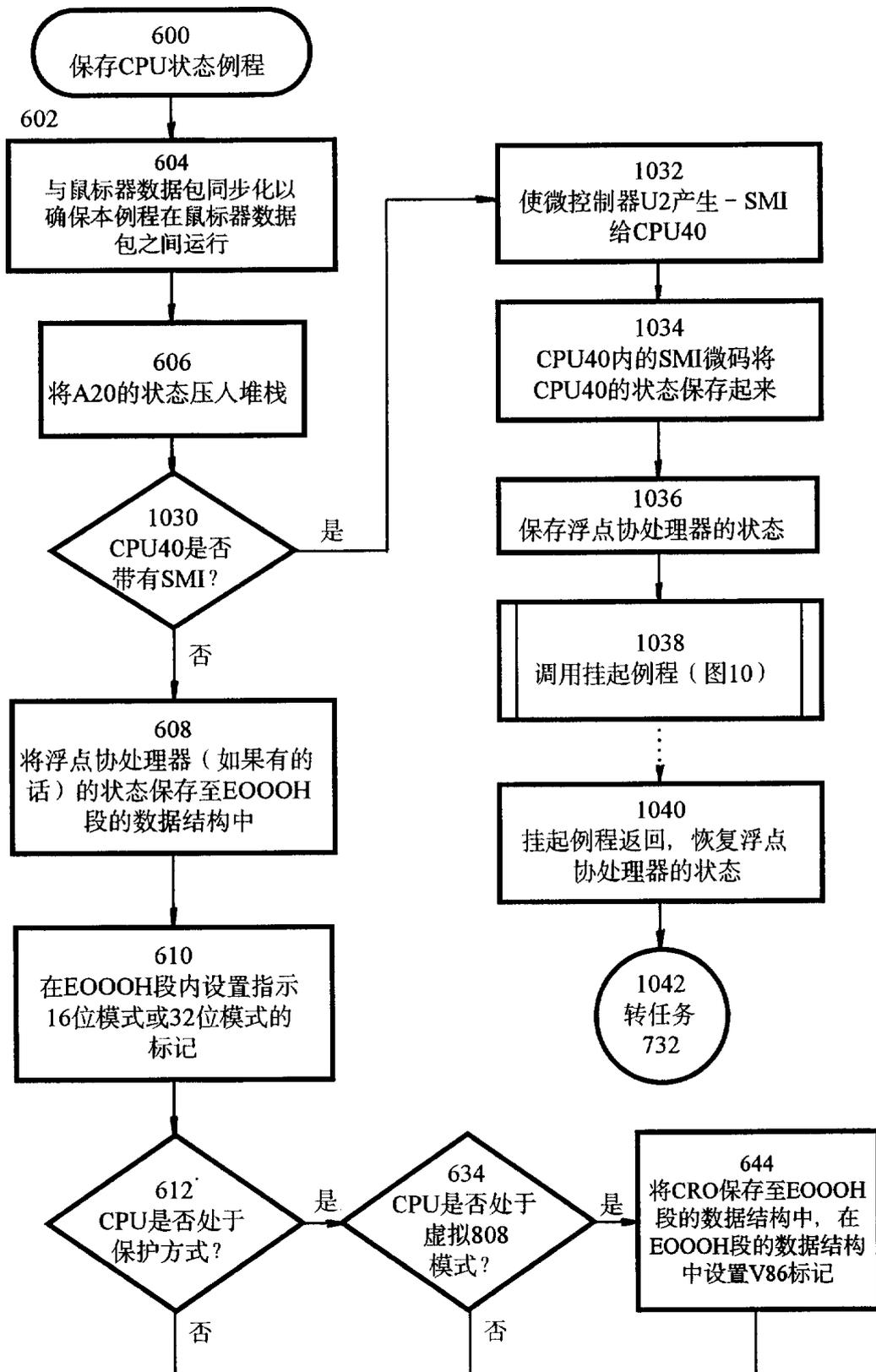


图 13A

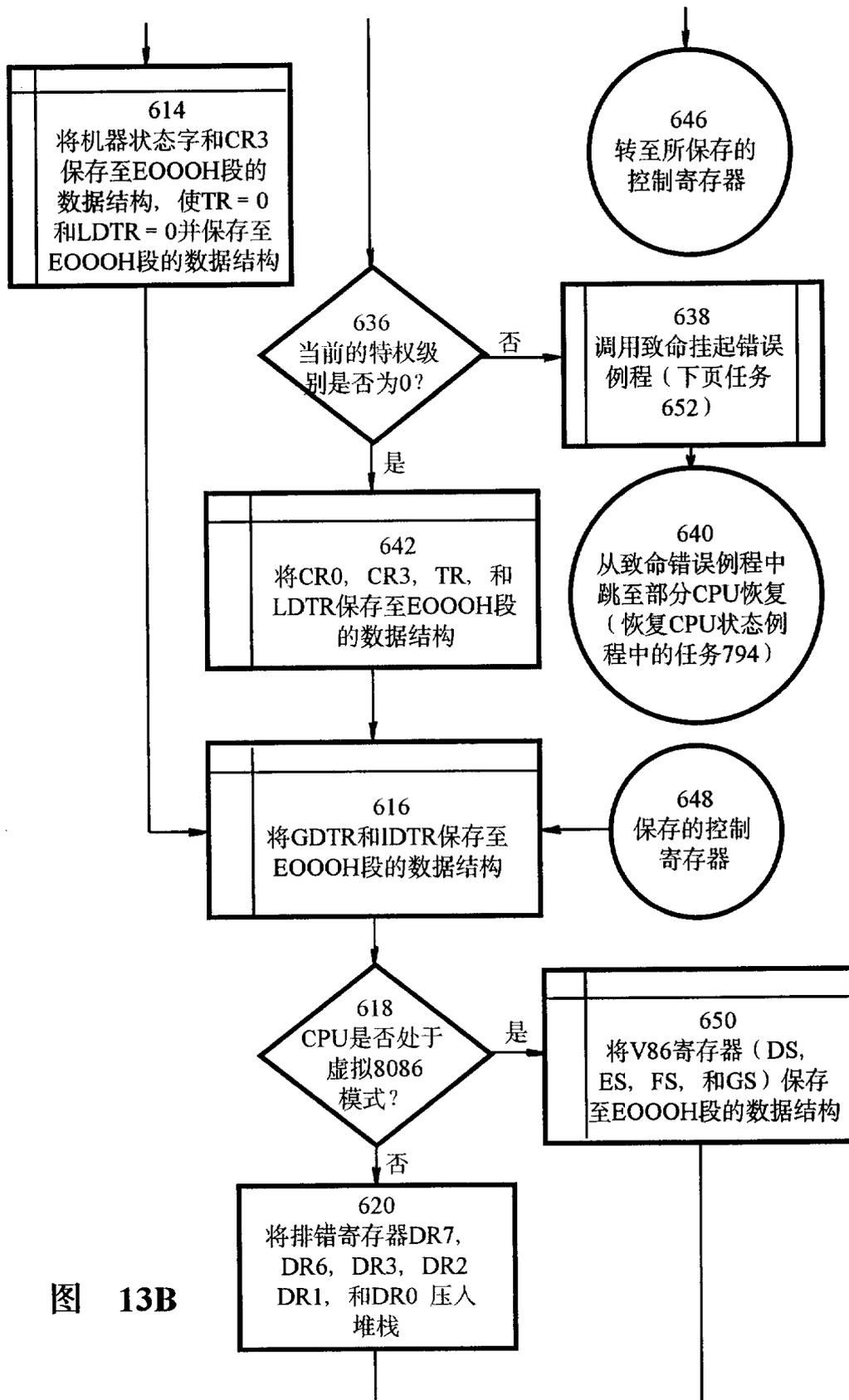


图 13B

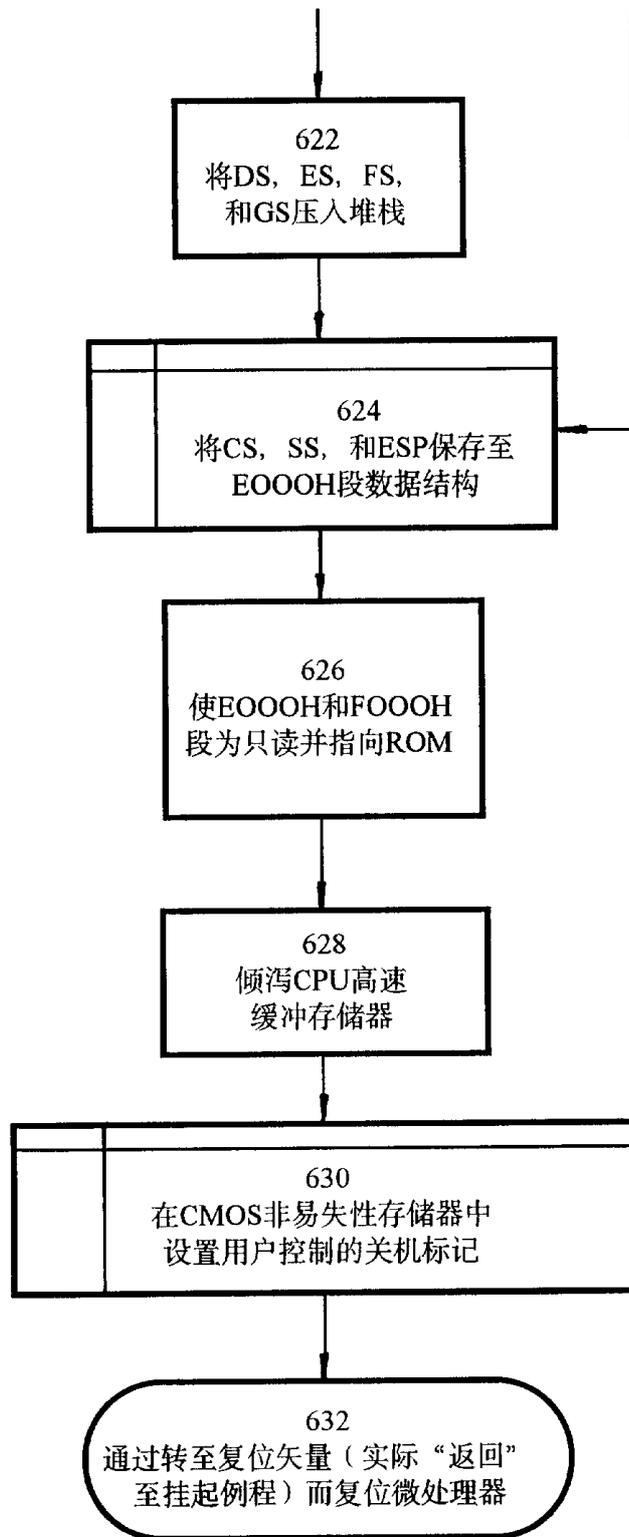


图 13C

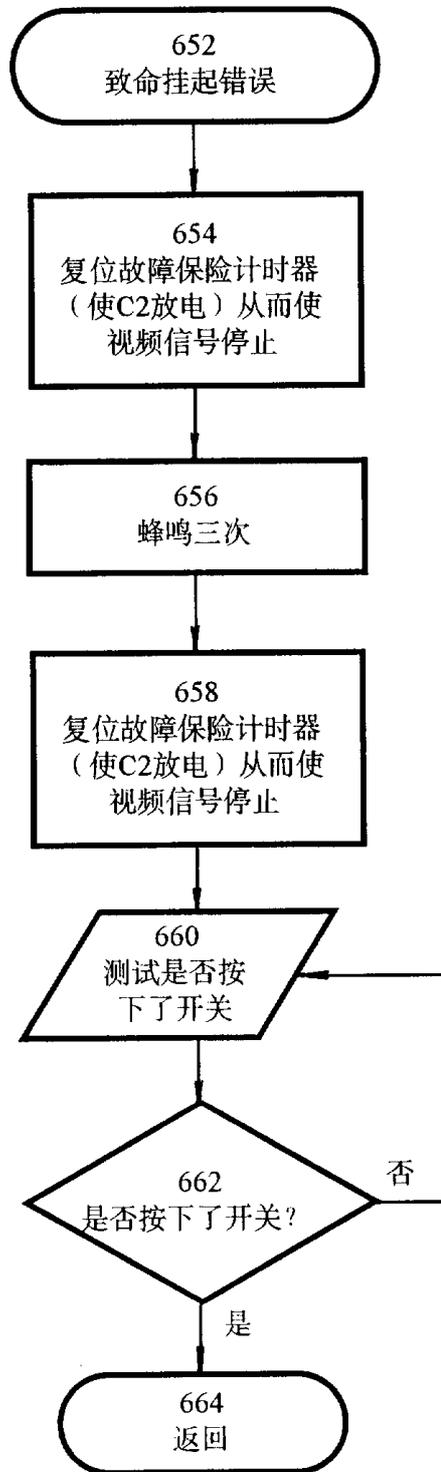


图 13D

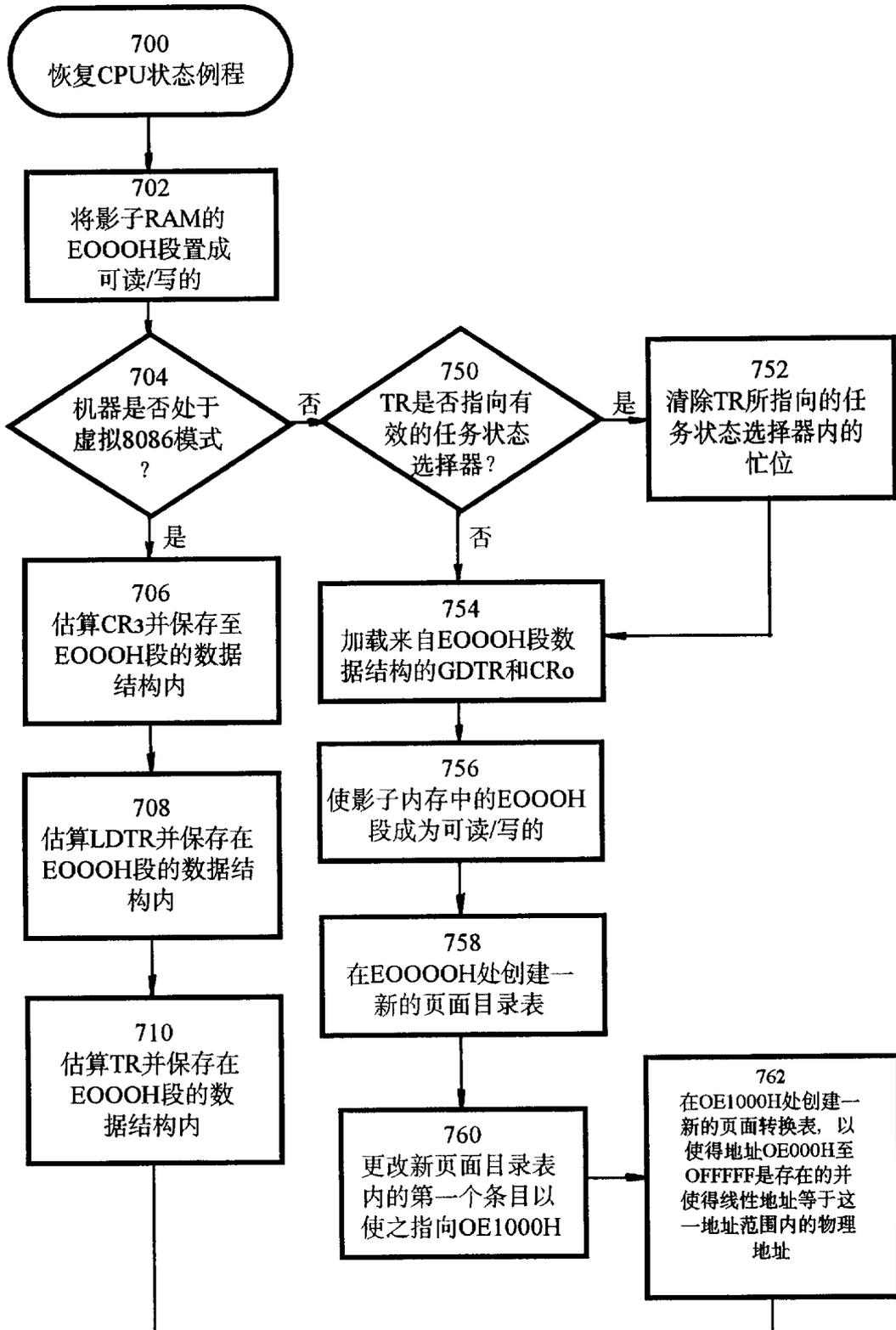


图 14A

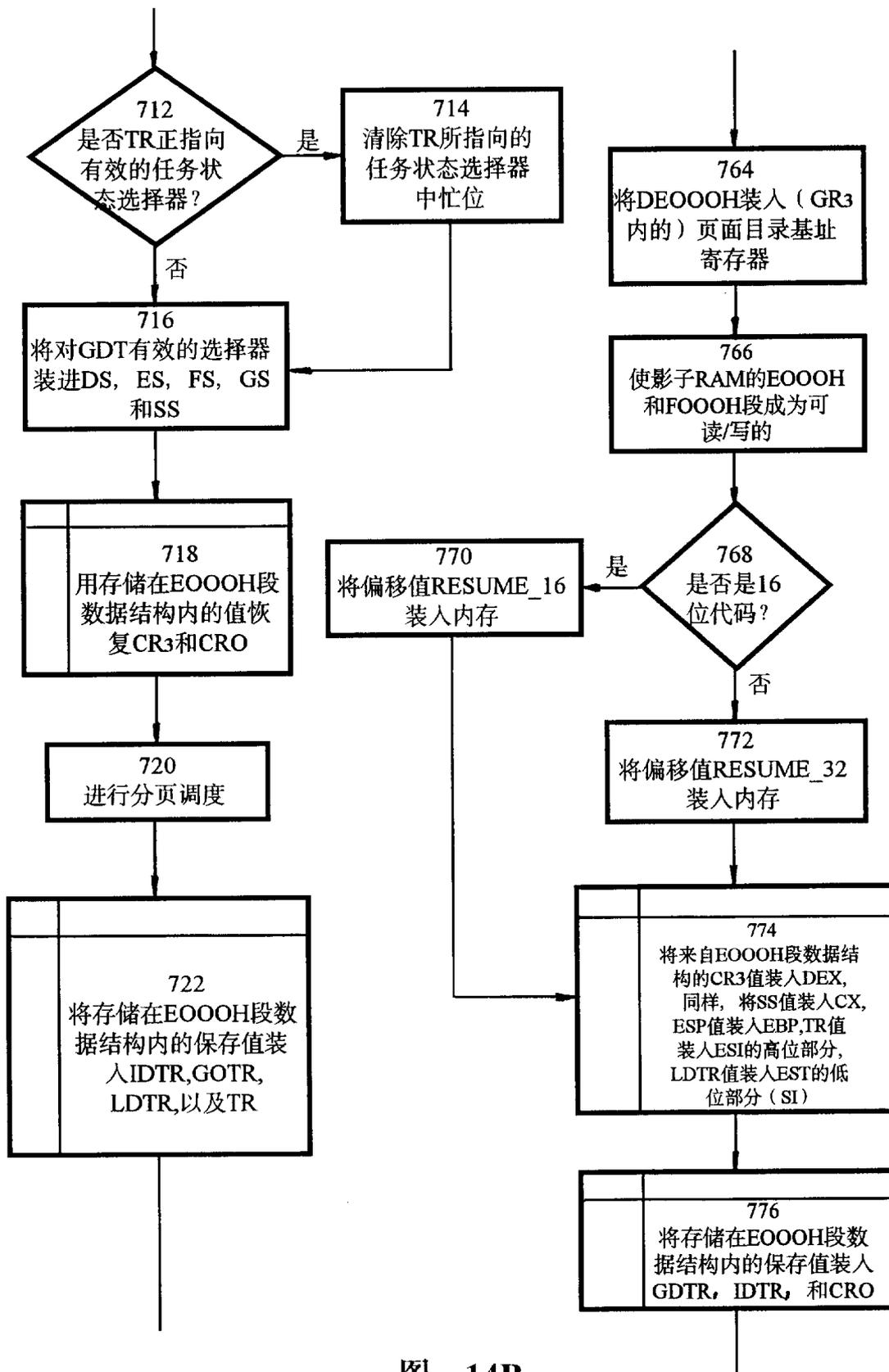


图 14B

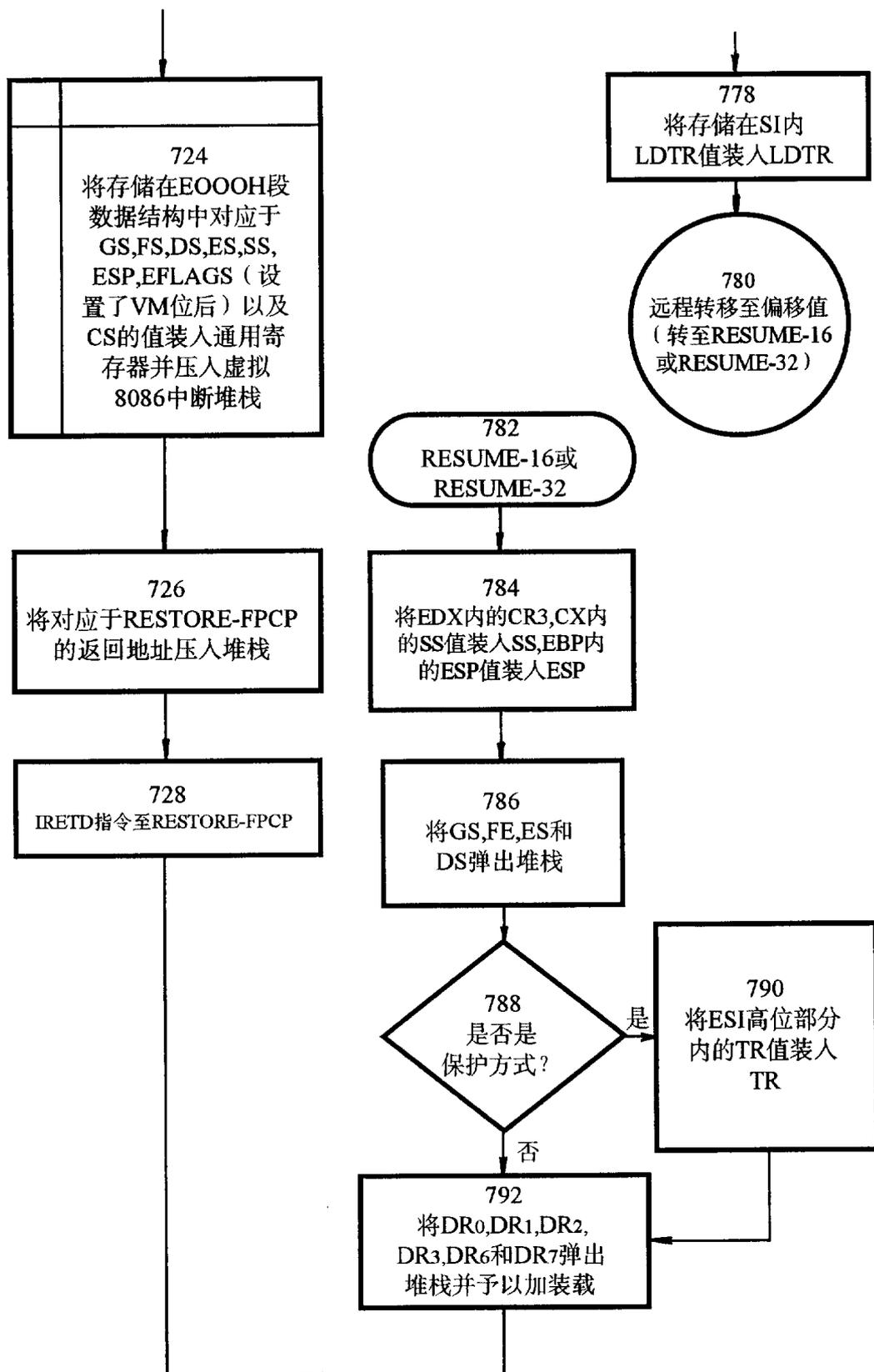


图 14C

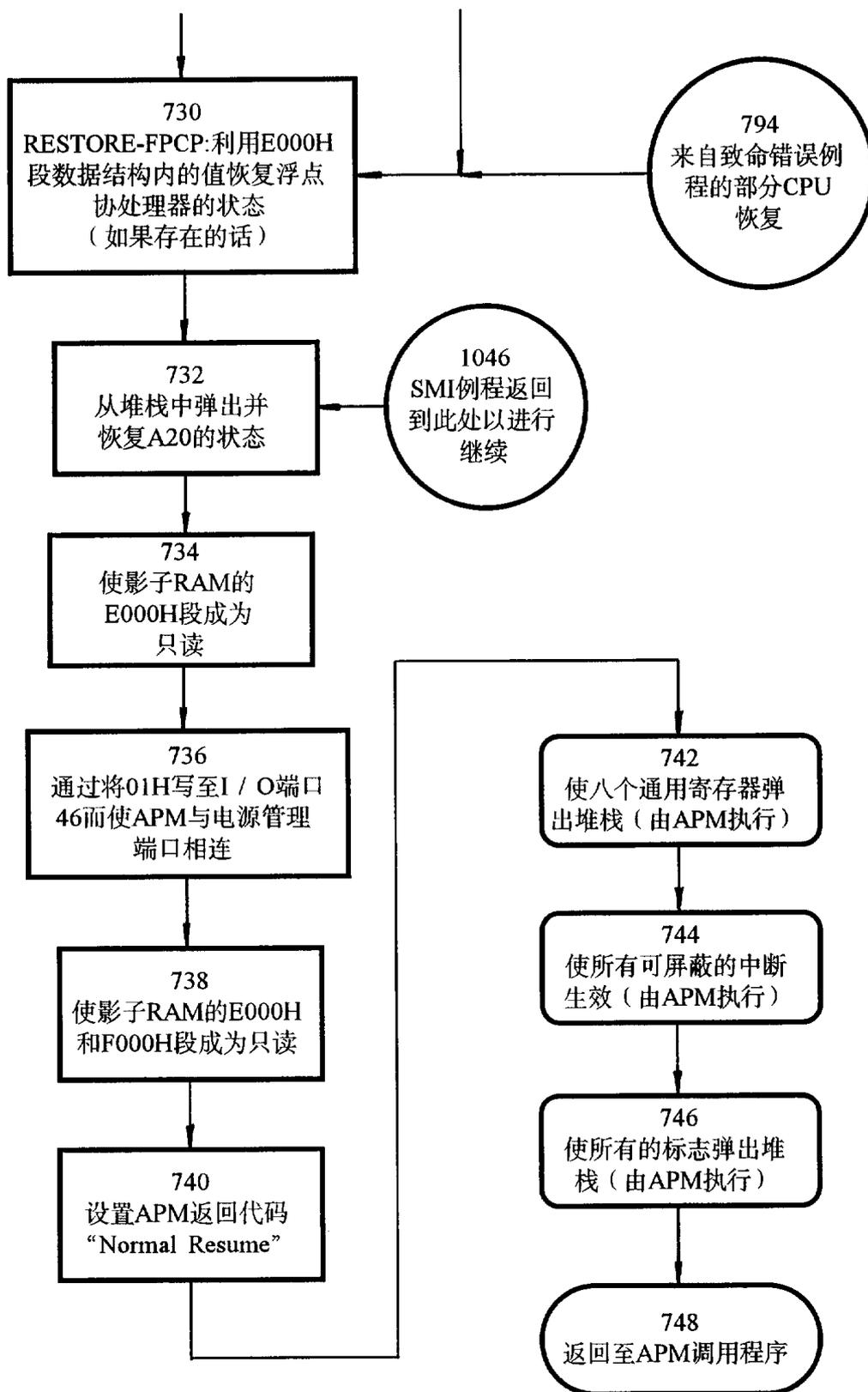


图 14D

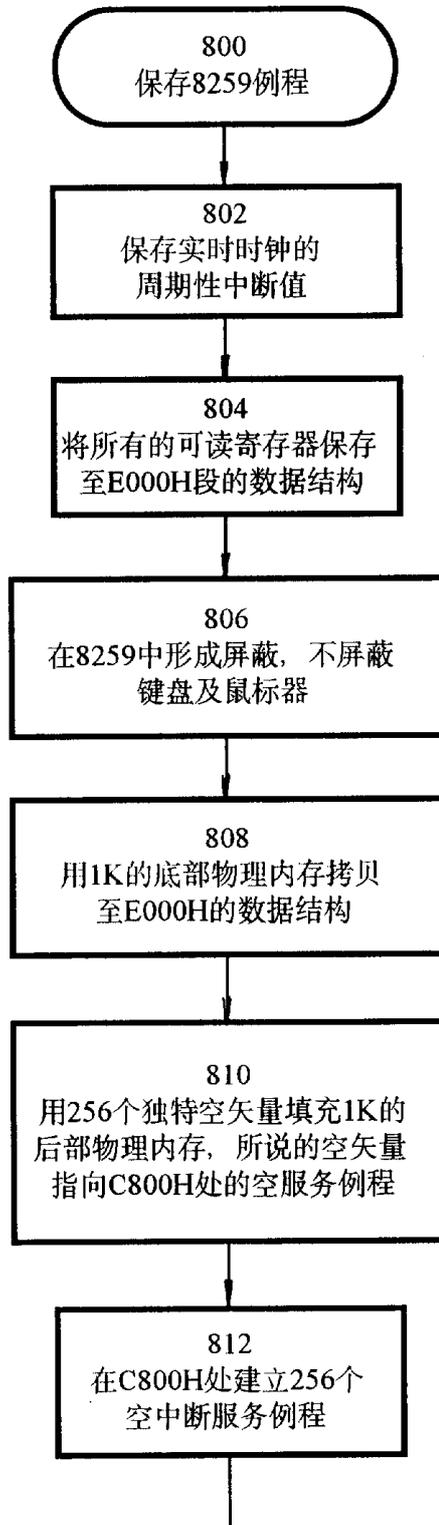


图 15A

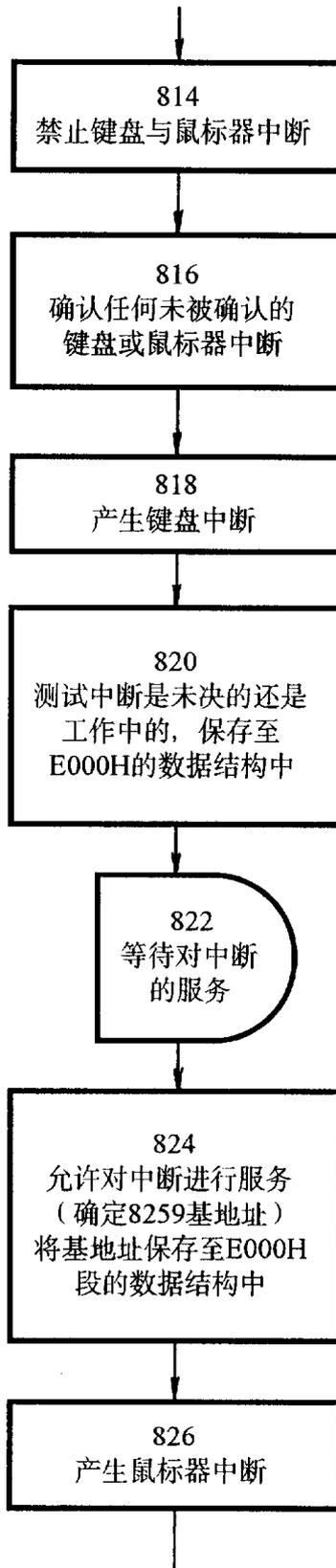


图 15B
41

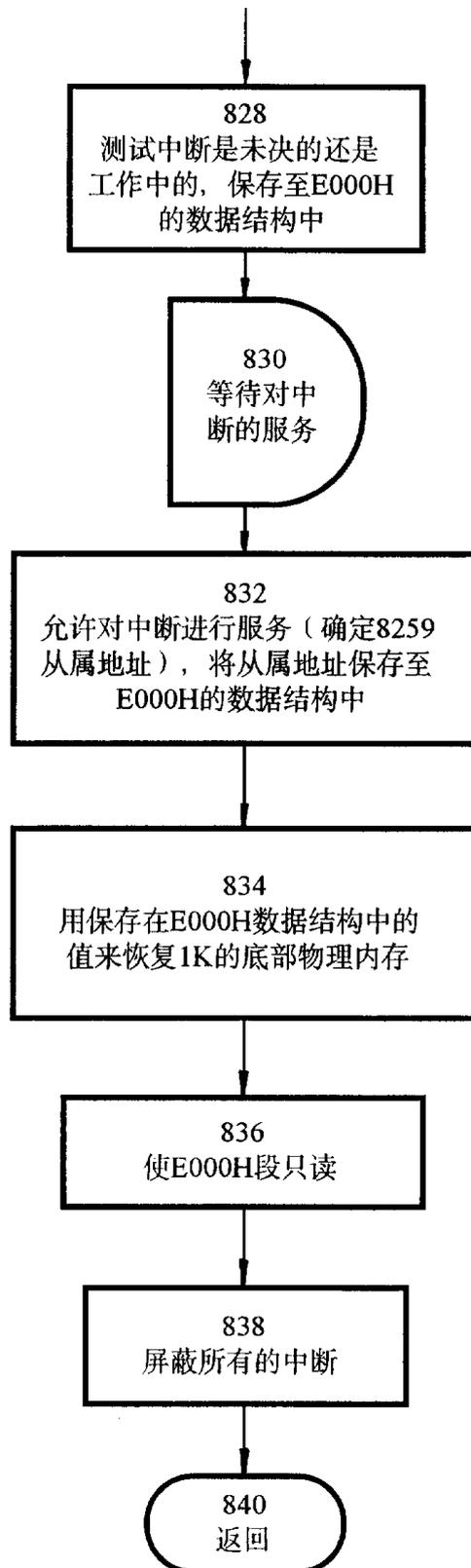


图 15C

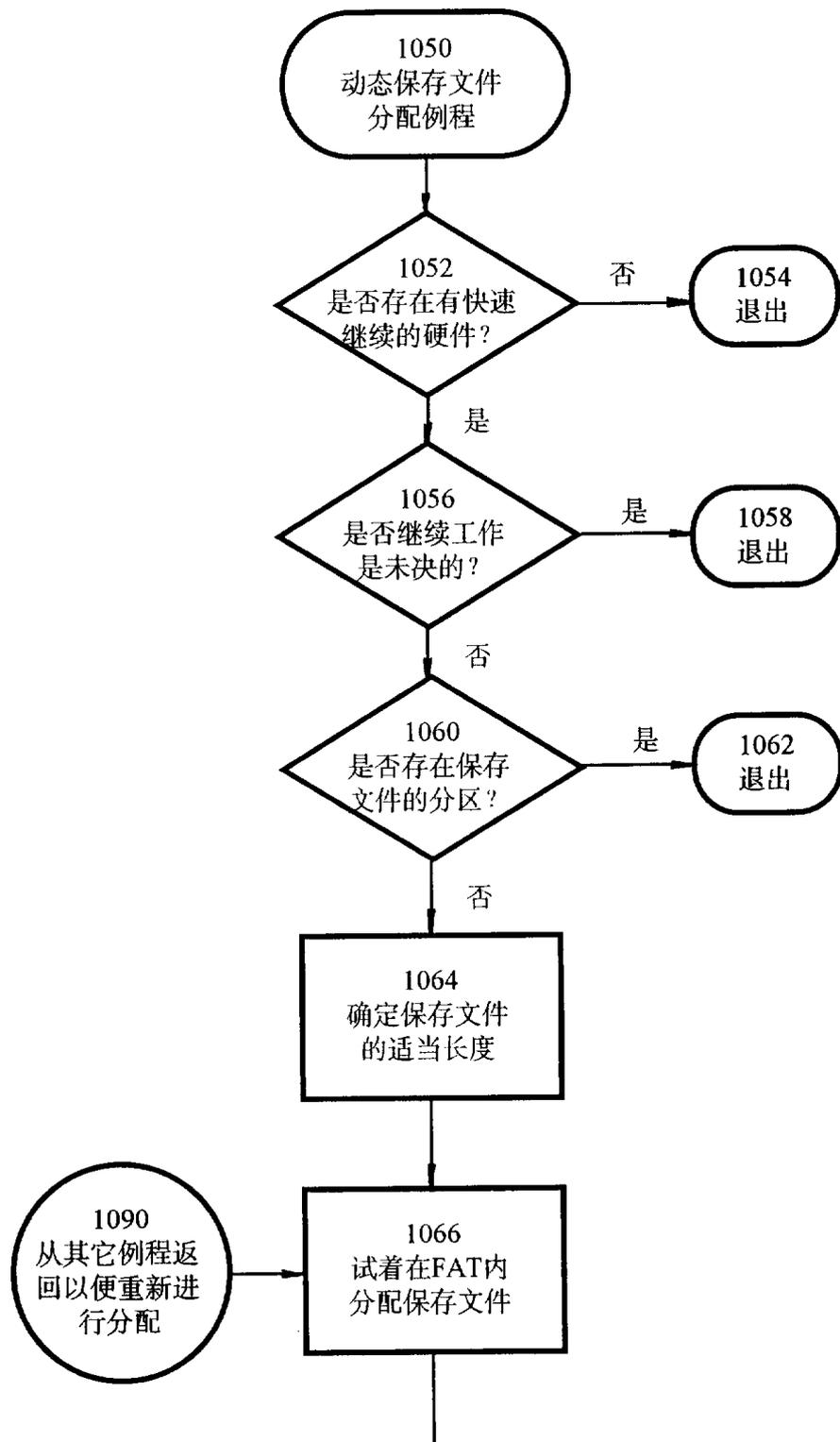


图 16A

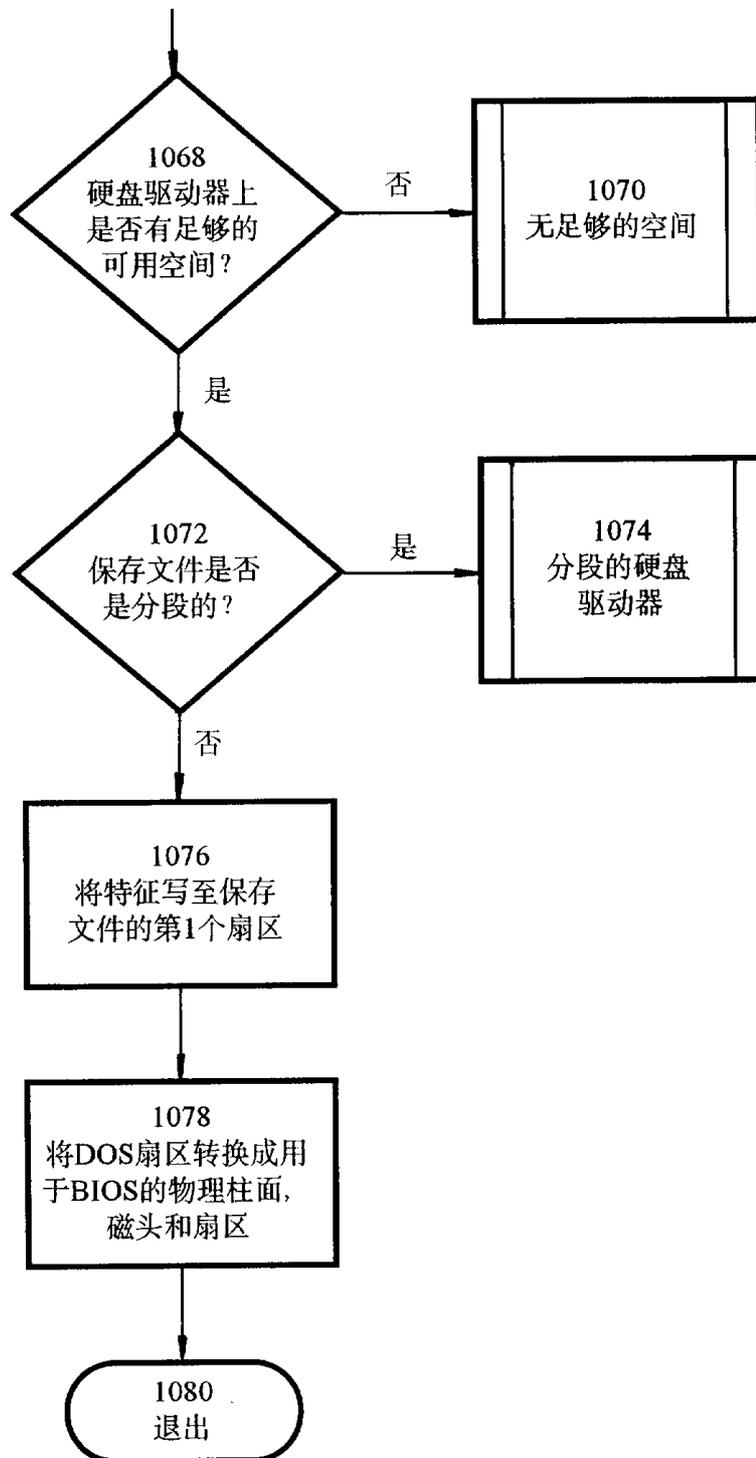


图 16B

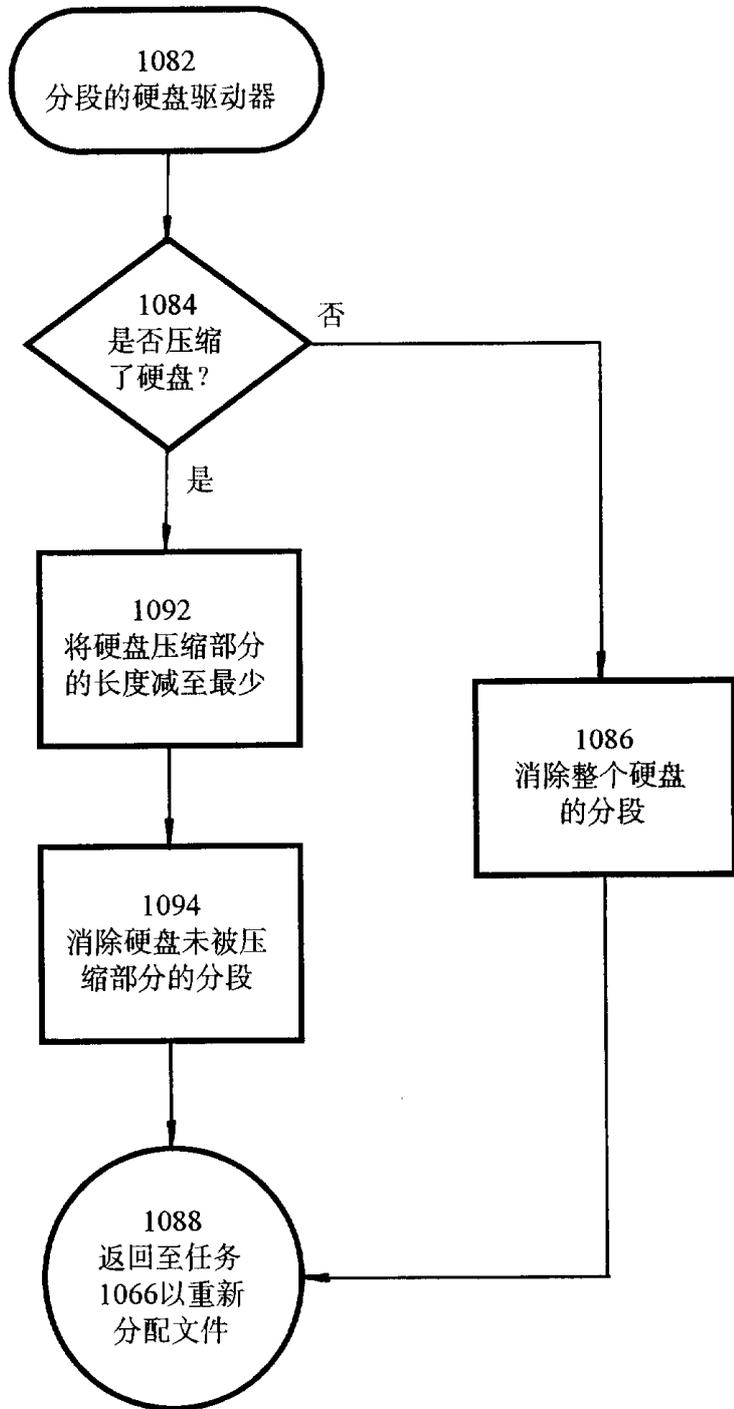


图 16C

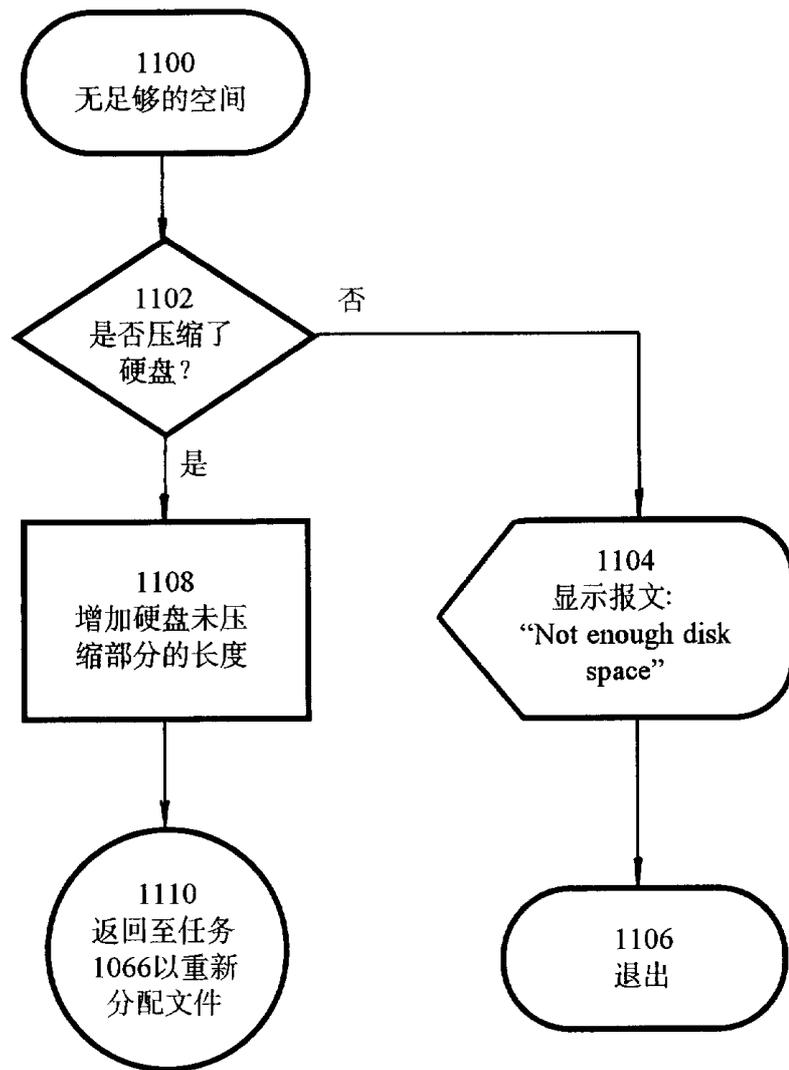


图 16D

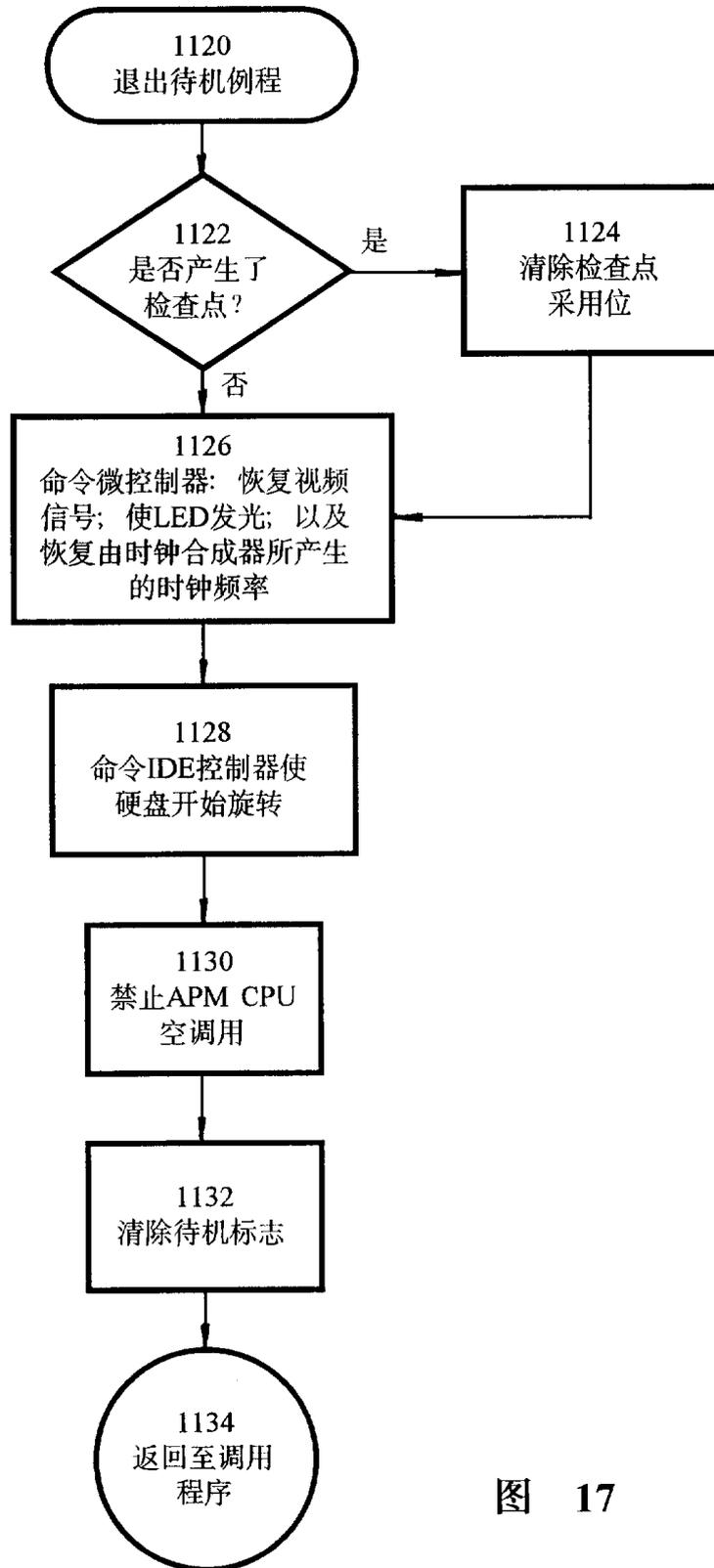


图 17

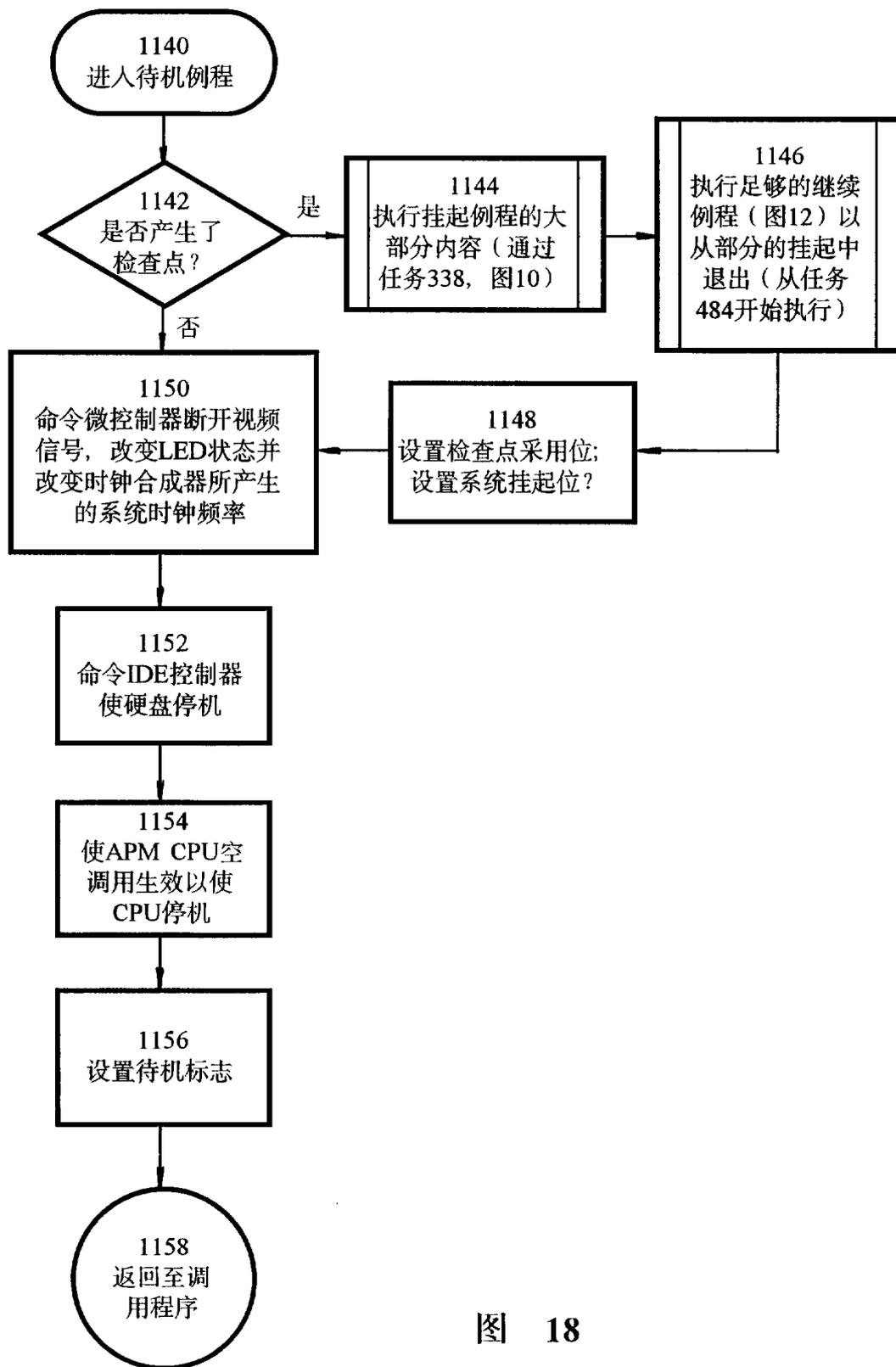


图 18

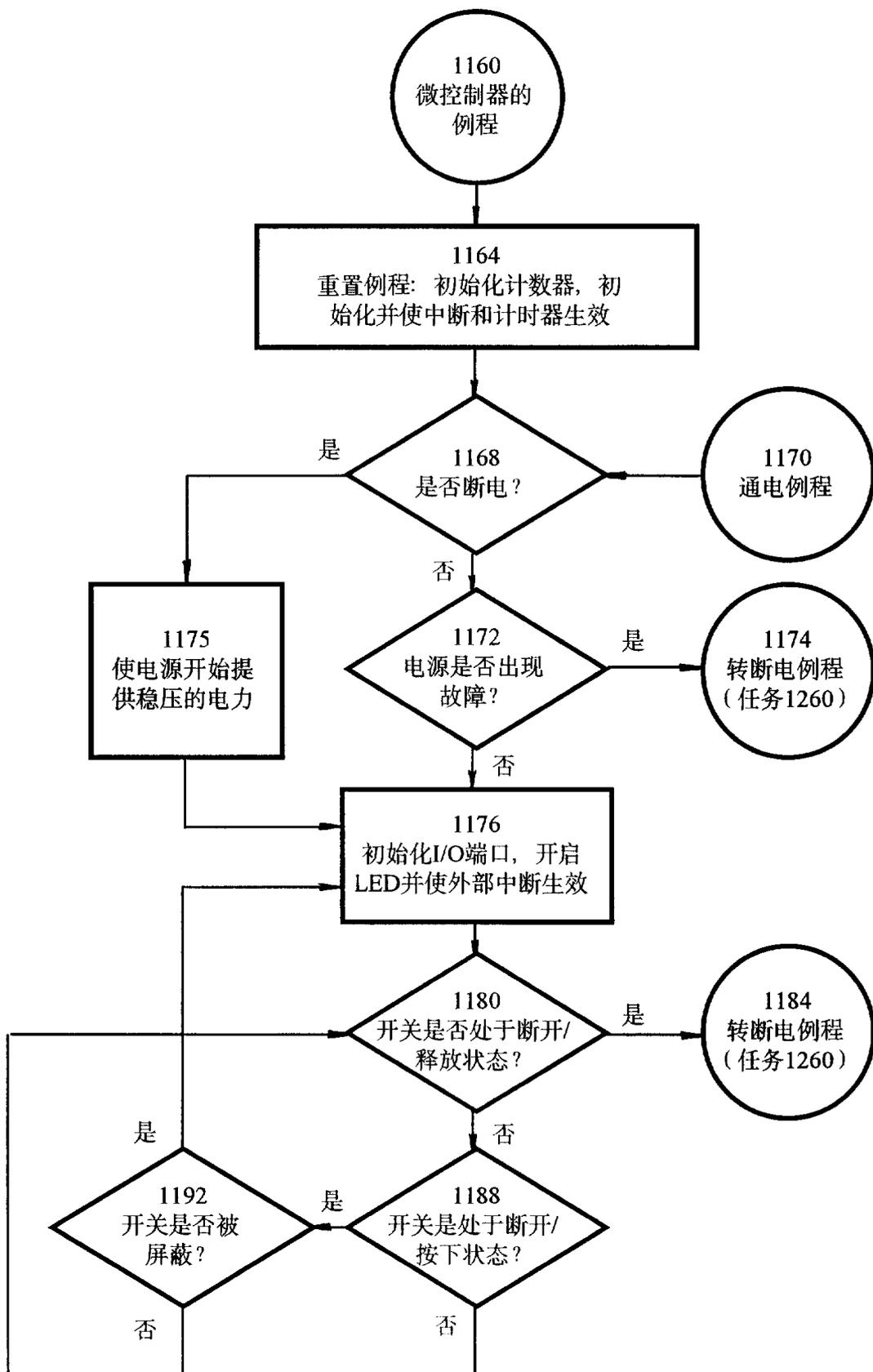


图 19A
49

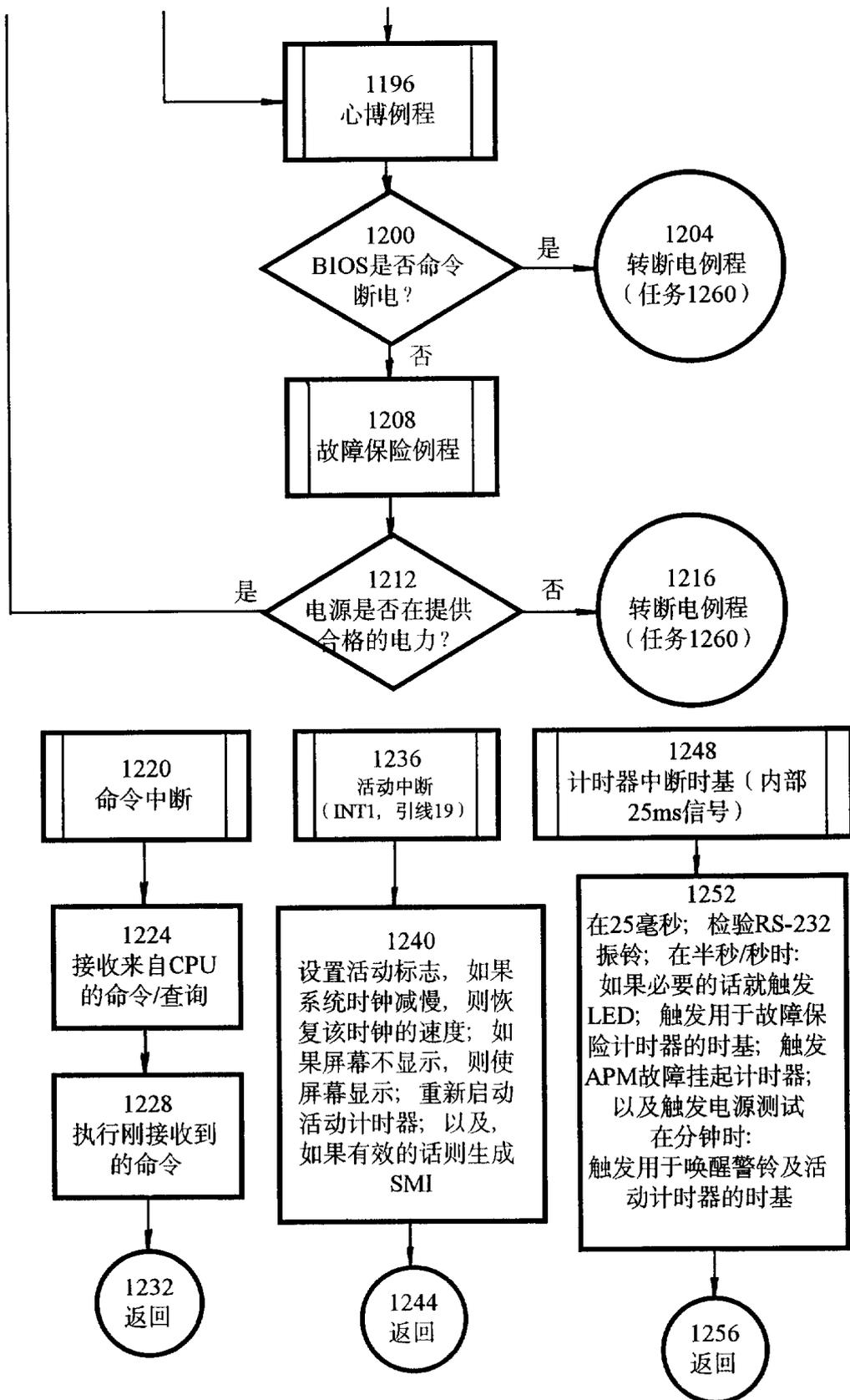


图 19B

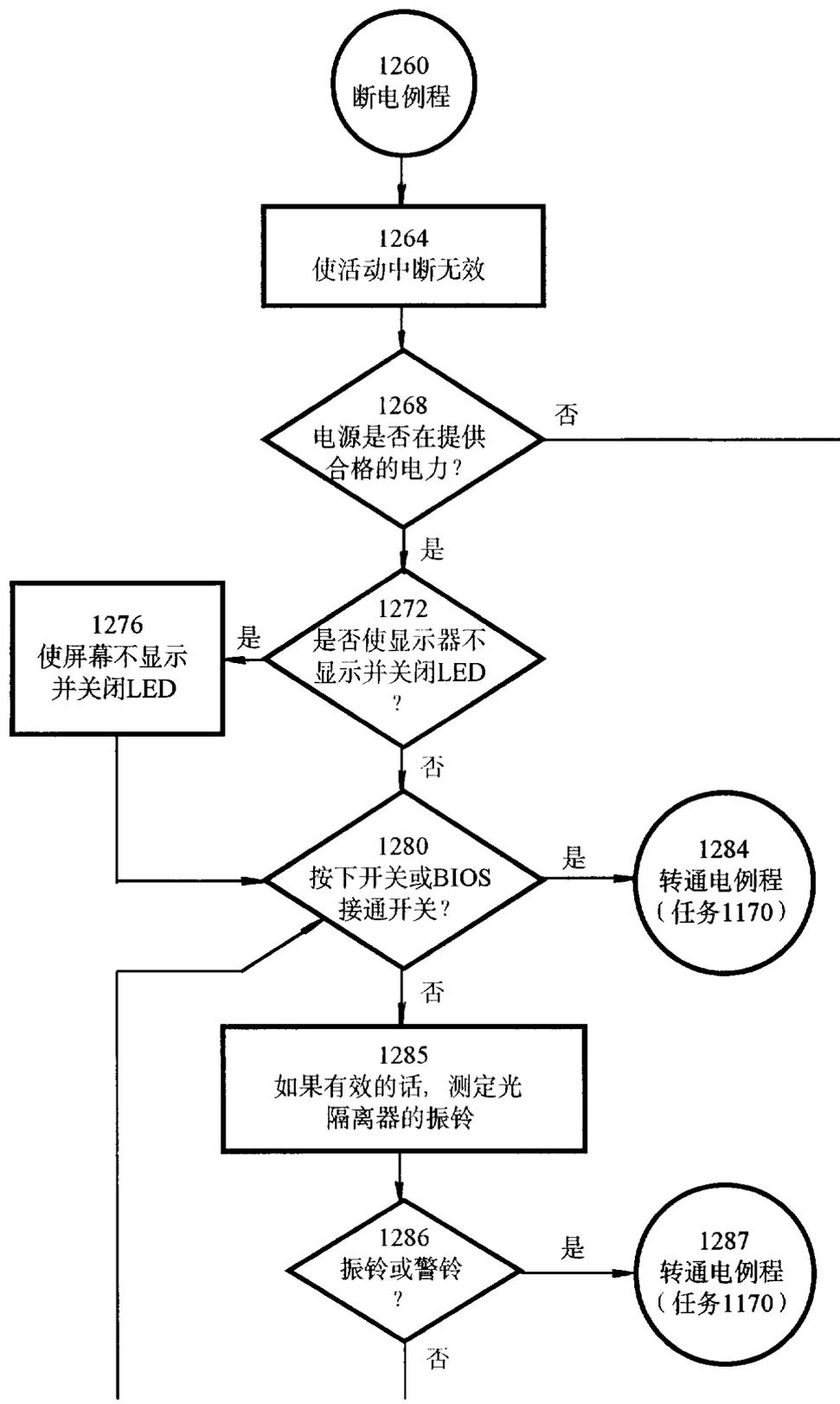


图 19C

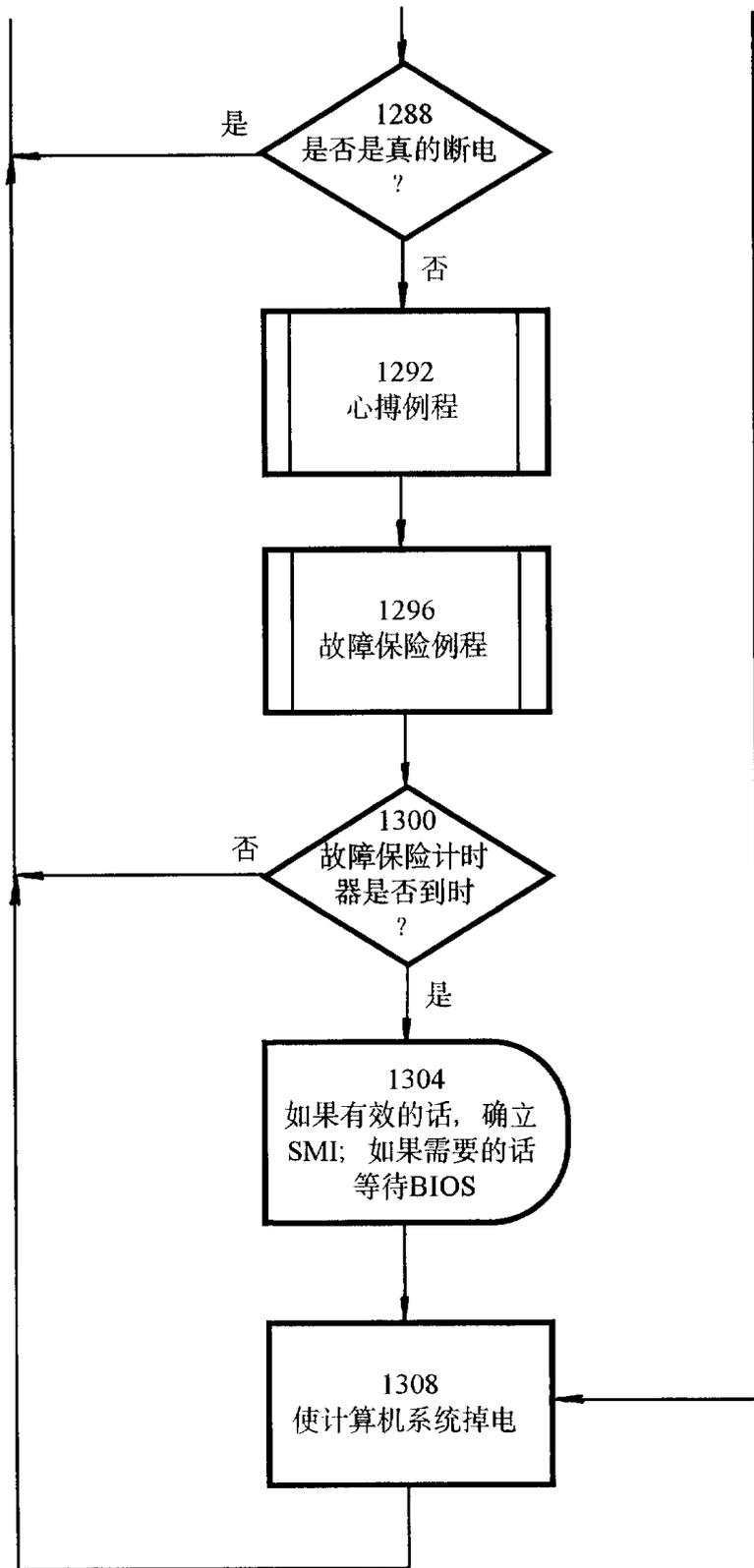


图 19D
52