[19]中华人民共和国国家知识产权局

[51]Int.Cl[7]
H04M 3/32
G06F 17/50

[12]发明专利申请公开说明书

[21]申请号 98808032.X

[43]公开日 2000年9月13日

[11]公开号 CN 1266579A

[22]申请日 1998.6.12 [21]申请号 98808032.X
[30]优先权
[32]1997.6.13 [33]US [31]08/874,660
[32]1997.7.3 [33]US [31]08/887,653
[86]国际申请 PCT/EP98/03569 1998.6.12
[87]国际公布 WO98/57484 英 1998.12.17
[85]进入国家阶段日期 2000.2.4
[71]申请人 艾利森电话股份有限公司
地址 瑞典斯德哥尔摩
[72]发明人 S·斯科拜

[74]专利代理机构 中国专利代理(香港)有限公司
代理人 程天正 陈景峻

权利要求书2页 说明书46页 附图页数31页

[54]发明名称 对基于计算机的电信系统的仿真

[57]摘要

一种在主计算机系统(110)上执行以便仿真一个目标电信系统的仿真系统(200)。该仿真系统(200)包括一个仿真内核(231),该仿真核(231)包含多个仿真子系统,每个仿真子系统对应于目标电信系统的子系统。该仿真内核(231)被组合成一个可执行图象,其中的全部仿真子系统都在一个共同的进程关联信息中执行并且共享一个共同的执行线程,仿真系统包括中央处理器仿真器(234),它根据经过用户通信信道输入到仿真核的命令来仿真对目标指令的执行。中央处理器仿真器仿真该目标电信系统的中央处理器的寄存器的使用。该目标电信系统的内部总线借助于各仿真子系统之间的功能调用来进行仿真。

ISSN1008-4274

权　利　要　求　书

1. 一种可以在主计算机系统上执行以便模拟一个目标电信系统的仿真系统，该目标电信系统具有一个中央处理器及多个通过内部总线通信的子系统，该仿真系统包括：

含有多个仿真子系统的仿真内核，这些子系统相应于目标电信系统的各子系统，该仿真内核被汇编成一个可执行的图象，它的所有仿真子系统在一个公共的进程关联信息下执行并共享一个公共的执行线程；

一个通向仿真内核的用于送入命令的用户通信信道；

一个中央处理器仿真器，它根据命令而模拟目标指令的执行，它还模拟对目标电信系统的中央处理器中各寄存器的使用；以及

其中的目标电信系统的内部总线是由各仿真子系统之间的功能调用来模拟的。

2. 权利要求 1 的仿真系统，其特征在于目标电信系统的中央处理器有一个指令处理器、一个信号处理器、以及一个区域处理器控制器、其中中央处理器仿真器有一个用于模拟目标电信系统的指令处理器的仿真指令处理器，以及中央处理器仿真器在接口的级别上模拟信号处理器和区域处理器管理器。

3. 权利要求 1 的仿真系统，其特征在于通过用户通信信道进入的命令用来按照目标电信系统的相应子系统的硬件而配置一个选定的仿真子系统。

4. 权利要求 1 的仿真系统，其特征在于通过用户通信信道进入的命令用来访问各仿真子系统中的至少一个子系统。

5. 权利要求 4 的仿真系统，其特征在于通过用户通信信道进入的用于选定的仿真子系统的命令是通过一个接插座发送到仿真内核并由命令调度程序传递给仿真子系统的。

6. 权利要求 1 的仿真系统，其特征在于每个子系统是按树形分层结构组织的。

7. 权利要求 1 的仿真系统，其特征在于每个仿真子系统具有一个命令解释器。

8. 权利要求 7 的仿真系统，其特征在于每个解释器包括用于管理它的仿真子系统的硬件配置的例行程序。

9. 权利要求 1 的仿真系统，其特征在于每个仿真子系统是可以从用

户通信信道加以控制的，其中每个仿真子系统具有它自己的命令解释器和自己的命令集。

10. 权利要求 9 的仿真系统，其特征在于所有子系统的各个解释器都遵循同样的对命令进行分析的原则。

11. 权利要求 1 的仿真系统，其特征在于各仿真子系统之一是下列各项之一：

（1）用于产生 ISDN 基本速率接入测试的仿真话务发生器；

（2）仿真的集团交换机子系统；

（3）仿真的会议呼叫设备子系统；

（4）仿真的信号转移中央子系统，它和仿真的延伸模块区域处理器设备通信；以及

（5）仿真的信号转移远程子系统。

12. 权利要求 1 的仿真系统，其特征在于各仿真子系统包括一个仿真信号转移中央子系统和仿真信号转移远程子系统，其中该仿真信号转移中央子系统和仿真信号转移远程子系统利用过程调用来转移信号。

13. 权利要求 1 的仿真系统，其特征在于各仿真系统中的至少一个子系统的某些功能是由中央处理器仿真器实现的。

14. 权利要求 1 的仿真系统，其特征在于目标电信系统的各子系统中的一个子系统是一个模块，用户可通过该模块对该电信系统接入，该模块是由接入处理器控制的，其中该仿真系统还包含一个仿真的接入处理器，该仿真的接入处理器模拟接入处理器指令集和接入处理器的寄存器组，该仿真的接入处理器在逻辑级别上和各子系统交互作用。

说　明　书

对基于计算机的
电信系统的仿真

本申请是 1997 年 6 月 13 日提交的题为“计算机处理器的仿真”的美国专利申请序号 08/874,660（代理人文档号 1410－361）的部分继续申请，该申请结合于此以供参考。

1.发明领域

本发明涉及基于计算机系统的测试，具体地说，涉及通过对基于计算机的电信系统进行仿真来进行对这类系统的测试。

2.相关技术和其它考虑

许多大型商业和工业系统使用大量的硬件部件，它们是以计算机为基础的，并且相互合作以实现各种任务。控制这类系统的计算机是复杂的并有多种功能。虽然经常希望对这类系统和控制这些系统的计算机进行测试，但却很少能承担得起这样的测试。因为测试会使系统和/或计算机离线，或者就是要消耗资源。一般来说，对这类系统和/或计算机的测试或者对这类系统的新软件的测试都是复杂和代价昂贵的。

在电信系统中测试的代价尤其显著。一个具体的例子是电话交换机，通过它连接用户的呼叫。这样的电话交换机一般都有一个中央处理器，它连接到多个地区处理器上。地区处理器按照指定给交换机的各种功能来执行各种任务。电信交换机通常是可重新配置的，这意味着其结构是动态的。

系统的仿真或模拟是在不加重系统负担的情况下测试系统的一种方法。在国际专利申请 PCT/US 94/03911“在主计算机中对客座指令进行译码的方法”中叙述了一种具体的仿真技术。

当前的模拟/仿真技术虽然在某些方面是有益的，但一般都具有较差的执行速度，并对存储器有苛刻的要求。此外，用户在仿真中通常都不能完成他在测试中所希望的那样多的要求。另外，要对像电信交换机这样的可以动态配置的系统进行仿真通常是困难的。

因此，所需要的，也是本发明的目的，就是对可动态配置的基于计算机的系统的有效仿真。

发明概要

使一个仿真系统在主计算机系统上执行以便模拟一个目标电信系统。该仿真系统包括一个仿真内核，它含有多个仿真子系统，每一个仿真子系统对应于目标电信系统的各个子系统。仿真内核汇编成一个可执行的图象，其中所有的仿真子系统都在一个共同的进程关联信息（process context）中执行并共享一个共同的执行线程。仿真系统包括一个中央处理器仿真器，它根据通过用户通信信道进入到仿真内核的命令而对执行目标指令进行模拟。中央处理器仿真器模拟对目标电信系统的中央处理器中各寄存器的使用。目标电信系统中内部总线则由各仿真子系统之间的功能调用来模拟。

目标电信系统的中央处理器具有一个指令处理器、一个信号处理器和一个区域处理器控制器。在仿真系统中，中央处理器仿真器有一个仿真的指令处理器，它模拟目标电信系统中的指令处理器。但是，中央处理器仿真器是在接口的级别上模拟信号处理器和区域处理器控制器的。

目标电信系统的一个例子是电信交换机。在模拟电信交换机时，仿真系统包括以下仿真子系统:

（1）一个仿真的话务发生器，用于产生 ISDN 基本速率接入测试;

（2）一个仿真的集团交换机子系统;

（3）一个仿真的会议呼叫设备子系统;

（4）一个仿真的信号转移中央子系统，它和仿真的延伸模块区域处理器设备相通信；和

（5）一个仿真的信号转移远程子系统。

在仿真信号转移中央子系统和仿真信号转移远程子系统之间利用过程调用来转移信号。

对于电信交换机来说，目标系统包括一个模块，用户通过该模块接入到电信系统中，该模块由接入处理器加以控制。这样的接入处理器在仿真系统中是由仿真的接入处理器来模拟的，该仿真的接入处理器模拟接入处理器的指令集和接入处理器的寄存器组。仿真的接入处理器和各子系统以逻辑级别亦即以消息级别相互作用。

用户的通信信道不仅用来向中央处理器仿真器发送命令，并且还用于接入到仿真子系统。例如，对一个选定仿真的接入可以按照目标电信系统中对应子系统的硬件来配置选定的仿真子系统而实现。通过用户通信信道输入的用于选定的仿真子系统的命令是由命令调度器通过插座而

发送到仿真核心并传递到仿真子系统的。

每个子系统按树形层次组织，其中每个仿真子系统有一个命令解释器。每个解释器包括用于处理它的仿真子系统的硬件配置的例程。所有子系统的解释器都遵照同样的用于分析命令的原则。

附图简介

本发明的上述和其它目的、特点和优点从下面的对由附图所表明的优选实施例的更加具体的叙述将变得很明显，图中的参考字符在所有各个不同的示图中都指明相同的部分。这些图不一定按比例表示，相反，重点是放在说明本发明的原理上。

图 1A 是包括一个远程用户站的电话交换系统的原理图。

图 1B 是电话交换系统的原理图，在该交换系统中用户站是包含在父交换局中的。

图 2 是按照本发明的实施例的用于电话交换系统中的中央处理器的原理图。

图 3 是按照本发明的实施例的用于模拟电话交换系统的主计算机系统的原理图。

图 3A 是按照本发明的主机系统的一个实施例的原理图，该实施例有一个跳转表，它能容纳非等长的目标指令。

图 3B 是按照本发明的主机系统的另一个实施例的原理图，该实施例通过提供跳转表的影子表而减少跳转表的大小。

图 3C 是按照本发明的主机系统的另一个实施例的原理图，该实施例通过利用时间印记的压缩技术而减少跳转表的大小。

图 4 是按照本发明的实施例的仿真系统的原理图。

图 5 是按照本发明的实施例的各子系统的分层结构的原理图。

图 6 是按照本发明的一个实施例的 SIGEN 命令分层结构的原理图。

图 7A 是表明按照本发明的一个实施例的集团交换机仿真模块的范围的示意图。

图 7B 是集团交换机仿真模块的原理图。

图 7C 是表明集团交换机仿真模块用的命令的位置示意图。

图 8A 是表明按照本发明的一个实施例的会议呼叫设备（CCD）仿真器的范围的示意图。

图 8B 是会议呼叫设备（CCD）仿真器的原理图。

图 8C 是表明会议呼叫设备（CCD）仿真器用的命令的位置示意图。

图 9A 是表明按照本发明的一个实施例的信号转移控制仿真器的范围的示意图。

图 9B 是信号转移控制仿真器的原理图。

图 9C 是表明信号转移控制仿真器的命令的位置的示意图。

图 10A 是信号转移远程和在其中的信号流的原理图。

图 10B 是表明信号转移远程和它周围的模块合作的原理图。

图 10C 是表明 STR 仿真器怎样和 EMRDs 及 STC 仿真器通信的示意图。

图 11A 是目标系统的各部分的原理图，表明一个信令终端 ST20C 和它的接口及信号流。

图 11B 是信令终端 ST20C 的主要部件的原理图。

图 11C 是按照本发明一个实施例的 ST20C 仿真器的原理图。

图 11D 是描绘使 ISS 68030 适应于在 ST20C 的关联（上下文）中 ISS 68020 的要求的原理图。

图 12A 是在特定的目标系统中提供对基本速率接入（BRA）的 ISDN 控制信号流（D 信道）概观的原理图。

图 12B 是经过实施 DBS 的软件的数据流的原理图。

图 13A 是 SIGEN 图形用户接口窗口的一种格式的示意图。

图 13B 是 SIGEN 命令分层结构的示意图。

图 13C 是 SIGEN 数据结构的示意图。

图 14 是表明由把目标目的代码翻译成仿真指令和执行该仿真指令的仿真监控控制例程所进行的流水线协调的示意图。

图 15 是表明按照本发明的一种模式在目标代码区、跳转表、和其中驻留着翻译例程的翻译指令区之间的相互关系的示意图。

图 16 是按照图 3B 的实施例，显示目标目的、代码区、跳转表、翻译指令区和跳转表影子表之间的相互关系的示意图。

图 17 是表明按照图 3B 的实施例用于编译指令时为翻译程序块所必需的存储器的示意图。

图 18 是按照图 3C 的实施例，表明目标目的代码区、跳转表和时间印记存储区之间的相互关系的示意图。

图 19a 是按照本发明的一种模式，在目标代码模块、跳转表和仿真代码区之间的相互关系的示意图。

图 19b 是表明在线仿真代码的组织的示意图。

图 20 是在修正的目标指令、中间生成表和翻译成仿真代码之间的相互关系的示意图，并说明了把一个特定的目标指令（例如寄存器加寄存器指令）转变成主机指令的过程。

图 21 是在主机存储器中寄存器仿真的示意图。

附图详述

在下面的说明中，作为解释并不是作为限制，提出了具体细节，例如特定的体系结构和技术等，以便为本发明提供完整的理解。但是，对熟悉本项技术的人来说，他们很清楚本发明可以用与这些具体细节并不相同的其它实施例来实现本发明。在其它情况下，众所周知的设备和方法的详细说明都省略了，以免用不必要的细节模糊了对本发明的说明。

本发明涉及用主计算机系统来仿真目标计算机系统。提供的一个具体的但非限制性的目标计算机系统的例子是涉及电话转接用的交换设备。作为讨论所示例子的计算机系统仿真的准备，对电话转接用的交换设备先加以说明。

1.0 目标计算机系统及其环境

图 1A 表示一个电话交换系统 18A，它包括一个本地电话转接用的交换设备 20，交换设备连接到一个远程用户交换设备（RSS）22。转接用的交换设备 20 包括集团交换子系统（GSS）30。集团交换子系统（GSS）30 由一个或多个集团交换区域处理器（GSRP）32 所控制，后者则又被中央处理器（CP）34 所控制。中央处理器（CP）34 和集团交换区域处理器（GSRP）32 通过区域处理器总线（RPB）36 通信。其它的区域处理器和信令终端中心（STC）38 也连接到区域处理器总线（RPB）36。如图 1A 所显示的，每个信令终端（STC）38 都连接到一个相关联的交换终端电路（ETC）40。

远程用户交换机（RSS）22 包括多个线路交换模块（LSM）44，每个线路交换模块（LSM）44 连接到多个用户，如用户 46。线路交换模块（LSM）44 包括众多的部件。至少一个，最好是多个（但不需要全部）线路交换模块（LSM）44 具有一个交换终端板（ETB）48。每个线路交换模块（LSM）44 具有一个延伸模块区域处理器（EMRP）50。每个交换终端板（ETB）48 由通过交换终端回路（ETC）40 中的一个回路的链路连接到集团交换机（GSS）30 的一个端口。延伸模块区域处理器（EMRP）50

连接到两个分开的 EMRP 总线 56A 和 56B。EMRP 总线 56A 还连接到信令终端远程（STR）58A；EMRP 总线 56B 还连接到信令终端远程（STR）58B。信令终端远程（STR）58 连接到远程用户交换机（RSS）22 中的交换终端板（ETB）48。

用户之间的一次呼叫，也称作一个联接，是通过电话交换系统 18 来选择路由的。一次呼叫一般同时包含信令和话务（用户的信息，例如语音或数据）。一次呼叫或联接利用例如在链路 54 上的时分时间片这样的信道，其中的信道 16 一般都被指定用作公共信道信令。

图 1B 表示一个电话交换系统 18B，其中用户级是包括在父交换设备之内的。和图 1A 的系统 18A 相似，电话交换系统 18B 包括一个集团交换子系统（GSS）30，它由集团交换区域处理器（GSRP）32 所控制，后者则又通过区域处理器总线（RPB）36 和中央处理器（CP）34 通信。图 1B 的电话交换系统 18B 和图 1A 的系统的不同之处在于：每一对信令终端中心（STC）38 和交换设备终端板（ETB）40 组合起来而形成区域处理器总线转换器（RPBC）60。区域处理器总线转换器（RPBC）60 同时连接到区域处理器总线（RPB）36 和线路交换模块（LSM）44。此外，在每个线路交换模块（LSM）44 中的交换设备终端板（ETB）被称作为连接线终端回路（JTC）62 的印刷电路板组合所取代。

集团交换子系统（GSS）30 实际上是一个交叉连接的矩阵电路并且因而构成了多个空间交换模块，这里把它集体地作为空间交换模块（SPM）70。空间交换模块的每一行由时间交换模块所控制，这里有多个时间交换模块（每行一个），统一地称为时间交换模块（TSM）72。

当联接信令表明在集团交换子系统（GSS）30 中要建立一次呼叫时，集团交换区域处理器（GSRP）32 通过把信息写入到时间交换模块中一个相应模块的控制存储器中而选择供连接用的经过集团交换子系统（GSS）30 的路径。为了使集团交换子系统（GSS）30 同步，使用了多个时钟模块，在此统一地被称为时钟模块 CLT 74。

用于在网络中建立、选定路由和结束用户的内容信息（即话务）的信令在这里被称作为“联接信令”。联接信令一般是两种形式中的一种：信道相关信令（CAS）或公共信道信令（CS）。这种信令不应和程序信令（例如软件指令）相混淆，后者是在两个或多个处理器之中或之间用来控制交换设备本身的各种功能的。今后在提到“信号”或“信令”时，

除非另有专门的指定，都是指程序信令。

在图 1A 和图 1B 的电话交换系统中，中央处理器（CP）34 有必要和包括在线路交换模块（LSM）44 中的延伸模块区域处理器（EMRP）50 通信。中央处理器（CP）34 利用程序信号和延伸模块区域处理器（EMRP）50 通信。这些程序信号作为在联接交换信道中所携带的信息的一部分而被包括在内。这就是说，在图 1A 的实施例中，在中央处理器（CP）34 和延伸模块区域处理器（EMRP）50 之间的通信是发生在信令终端中心（STC）38 和交换设备终端板（ETB）48 之间的联接信令上的（该信令的路由经过交换设备终端回路(ETC)40），而该交换设备终端板（ETB）48 连接到一个信令终端远程（STR）58 上。每个信令终端远程（STR）58 则又连接到所有的延伸模块区域处理器（EMRP）50。在图 1B 的实施例中，每个区域处理器总线转换器（RPBC）60 连接到所有的延伸模块区域处理器（EMRP）50 上。

图 1A 和图 1B 的电话交换系统的体系结构对熟悉本技术的人来说都是了解的。这种结构的另外的例子在 1994 年 2 月 15 日提交的序号为 08/601,964 的美国专利申请中有叙述，其题目为“电信转接交换设备”，在这里被提及供参考。

中央处理器（CP）34 的一种实施例示于图 2 中。实际上，中央处理器（CP）34 包含两个中央处理器，即特定中央处理器 34A 和中央处理器 34B，它们共同被称为中央处理器 34。每个中央处理器含有指令处理单元（IPU）80，其中指令处理单元 80A 和 80B 由 IPU 更新和匹配总线 82 所连接。每个中央处理器还包含一个信号处理单元（SPU）84，其中信号处理单元 84A 和 84B 由 SPU 更新和匹配总线 86 所连接。信号处理单元 84 管理 CP34 的工作，并准备要由指令处理单元 80 进行的工作，从而使指令处理单元（IPU）连续地执行程序。指令处理单元（IPU）执行程序，而信号处理单元则通知 IPU 程序应被执行的地址。

每个指令处理器（IPU）80 要访问三个存储单元，具体说是程序存储器（PS）87、参考存储器（RS）88、和数据存储器（DS）89。如上所述，指令处理器（IPU）80 执行各项作业，每个作业相当于存放在程序存储器（PS）87 中的一块指令。信号处理器（SPU）84 用来作为指令处理器（IPU）80 的作业的调度器。由于这样一种调度，对于每一个作业信号处理器（SPU）84 收到一个程序“信号”，例如，从外部世界或从

指令处理器（IPU）80 来的信号。一般说来，一个信号是一条指令，它告诉要在一个指令块的特定部分中什么地方去执行，该信号包含在执行该块时要使用的数据。信号在 3、2、3 节中要进一步叙述。信号处理器（SPU）84 分析并准备好进入的信号，并在把它们送到指令处理器（IPU）80 之前给这些信号规定优先级。参考存储器（RS）88 含有描述系统中所用的信号、区段（块）和变量的信息。

维护单元（MAU）90 同时连接到两个指令处理器（IPU）80。维护单元（MAU）90 有若干功能，包括在故障情况下启动各种测试和确定哪一侧（例如中央处理器 34A 还是中央处理器 34B）应被指定为执行单元。

每个中央处理器 34 还包括一个区域处理器控制器 92。区域处理器控制器 92 管理去到各区域处理器或来自各区域处理器的程序信令，这些处理器包括区域处理器总线转换器 60 和集团交换区域处理器 32。区域处理器控制器 94 连接到和它相关联的信号处理器（SPU）84 和区域处理器总线（RPB）36。

一个具体的被称为 APZ 的中央处理器的更进一步的体系结构在 1995 年 12 月 19 日提交的美国专利申请序号 08/574,977 中公布，其题目为“指令处理器的作业调度”，在此引用以供参考。

CP 34 的指令组和寄存器分别在 3、2、1 等和 3、2、2 节中说明。

2.0 主计算机系统

图 3 显示按照本发明的一个实施例的主计算机系统。主计算机系统 110 有一个主处理器，即 CPU112，它通过总线 114 和其它主机组成单元通信。其它主机组成单元包括输入设备（如键盘和指点器〔例如鼠标〕）116；输出设备（如显示器、打印机）118；输入/输出存储设备（如磁盘驱动器）120；只读存储器（ROM）112 和随机存取存储器（RAM）130。熟悉本技术的人知道这些组成单元的各种设备是通过相应的接口连接到总线 114 的。

在所示的实施例中，主计算机是一台 SUN SPARC 工作站。在下面我们假定在主机上用的是 SPARC 的型号 8，也就是说，可以用 imul 和 idiv 指令。除了可以是 32 比特或 64 比特的（合成）指令组外，所有 SPARC 指令是 32 比特的。CPU112 是 110 兆赫的微处理器，RAM 130 是 64 兆字节，而 I/O 存储设备 120 是能容纳 300 兆字节转储的硬盘驱动器。本发明并不局限于应用这种特定的主机类型，因为可以理解本发明的原理同

样也可以用于别的计算机系统。

图 4 表示在主机系统 110 上执行的仿真系统 200。仿真系统 200（也称为“SIMAX”）的主要部分为:

（1） 一组现成的工具，包括工具 210、212、和 214，分别称作 WIOL、Plex 查错程序和 Autosis。

（2） 外加的用户工具 216，称为主控制窗口（MCW），用于硬件配置和低级查错。

（3） 话务发生器 220，用于执行 IGEN 原文。

（4） 查错程序 222（C/C++文本），用于 ST20C 处理器。

（5） 仿真器内核 231（包括若干个仿真子系统，它们部分地相当于 AXE－10 ISDN 基本速率接入系统的实际配置）。

如在图 4 中所应用的含义那样，“子系统”指的是对应于由仿真系统 200 所模拟的实际硬件。

2.1 仿真器内核

关于图 4 的内核 231，仿真子系统包括仿真的集团交换子系统 230；一个或多个仿真的集团交换区域处理器 232；仿真的中央处理器 234；仿真的区域处理器总线 236；仿真的延伸模块区域处理器（EMRPD）250；仿真的区域处理器总线控制器（RPBC）260；仿真的会议呼叫设备（CCD）262；以及仿真的输入/输出系统（IOG 3）264。仿真的中央处理器 234 包括仿真的指令处理单元（IPU）280；仿真的信号处理单元（SPU）284；仿真的程序存储器（PSU）287；仿真的参考存储器（PSU）288；仿真的数据存储单元（DSU）289；以及仿真的区域总线管理器 292。

内核 231 是汇编成一个可执行的 Unix 图象的。所有的在仿真器中的子系统都在共同的 Unix 进程关联信息中执行并共享同一个执行线程。这是由下列理由促成的:

（1）在相互合作的子系统之间的同步可被简化（例如，并不基于任何 Unix 的 IPC 机制）；

（2）不会发生由于 Unix 的“重载”进程调度而引起的性能变坏。

（3）有可能得到对执行的确切控制，甚至达到执行单独指令的水平。

（4）公共的地址空间减少了建立用于在子系统之间传递仿真信号的数据缓冲的复本的需求。

（5）可以在相互合作的子系统之间对执行线程建立高效的分配（极其简单的调度机制）。

（6）把调度程序配置成能在仿真时给出完全确定性行为的可能性，使它有可能严格地重复事件过程。如在这里所用的“调度程序”是指用于模拟内核 231 的不同线程的调度机制。

在 EMRPD 中 Sim APP 是指用 C/C++编写的应用软件。REOS 是 EMRPD 用的 OS（操作系统），并可从 Enea Data 获得。

2.2 在框架内的模块性

仿真器内核 231 的框架为管理和控制在仿真器内核 231 中的不同的子系统建立一个公共的环境。框架还实现和支持若干个由子系统所使用的公共服务。

框架环境由下列各功能建立:

· 登录管理

· 执行线程调度

· 执行控制

· 虚拟时间的产生和管理

· 公共的配置信息处理

附加的公共服务是:

· 出错/跟踪报告实用程序

· 在内核中动态分配的存储器管理

· 分派 Unix 信号

· 简单命令处理

· 进程处理功能

2.2.1 用户交互作用的普遍原则

当仿真系统 200 起动时，称作 SIMAX 命令窗口（SCW）的一个窗口被打开，它成为用户对仿真器核心 231 的通信信道。SCW 是一个纯粹的文本窗口，因为用户的交互作用是命令取向的。命令是逐行输入的，当用户按回车键时，一行完整的命令被转送到命令解释器，由它去实施该命令。在功能测试的一个典型的应用中，配置是通过输入一个准备好的配置原本文件而完成的。

2.2.1.1 仿真系统的配置

一次仿真活动总是从仿真器的配置开始，即用对被仿真的系统（目

标系统）的性质的描述开始。这由两个基本步骤实现。

在第一步中，被仿真系统的硬件组成要加以叙述。各组成部件被建立（被示例），且每一个部件都给出一个身份并详述它的性质。举例来说，可能的性质可以是与中央处理器 34 相关的 PS 87 和 DRS 89 的 RAM 和 ROM 的大小。

在第二步中，在各部件之间的连接要予以说明。例如，在硬件系统中的总线、PCM 链路、和线路接口端口要用仿真部件之间的连接来模拟。

在下面的例子（从 SIGEN 220 的功能描述中借用）中，这两个步骤是涉及 TUBORG（话务发生器子系统）而说明的:

```
FW> ...
FW> . SIGEN
FW/SIGEN> _create TUBORG
FW/SIGEN> TUBORG
FW/SIGEN/TUBORG> _attach DLB2_ONE 1 0
FW/SIGEN/TUBORG> _attach DLB2_ONE 2 1
FW/SIGEN/TUBORG> _attach DLB2_TWO 0 2
FW/SIGEN/TUBORG> up
.
.
```

```
FW/SIGEN> .TUBORG
FW/SIGEN/TUBORG> _detach 0
FW/SIGEN/TUBORG> _detach 1
FW/SIGEN/TUBORG> _detach 2
FW/SIGEN/TUBORG> up
.
.
FW/SIGEN> _delete TUBORG
FW/SIGEN> top
.
.
```

当被仿真的系统正确地建立起来时，仿真就可以开始。

实际上，建立过程很可能是存储在一个原本文件中的，它可以在系统起动时被读入。原本文件基本上是个 ASCII 文本文件，它含有所需的命令。原本文件的名字可以作为命令行的变元给出，也可以作为“插入”命令（见下文）的变元给出。作为命令行的变元给出原本文件相当于键入“插入”命令作为第一条命令。

2.2.1.2 使用被仿真的系统和仿真器

在仿真期间，用户可以用两种方式接入到系统中。一种接入是直接朝向仿真器本身，另一种接入是朝向被仿真的系统。

正常情况下，用户通过一个外部工具（如 Plex View 212 和 WIOL 210）和被仿真系统相互作用。Plex View 是一种观看和查错工具。它处理 PLEX 列表（E-模块）、信号和校正。Plex View 可取代用纸打印的 E-模块列表和信号说明文档。Plex View 向用户提供浏览、搜索和打印功能。WIOL 是一个在 UNIX 操作系统下的交互性人机通信程序。利用 WIOL，用户可以与目标系统建立和关闭连接，并为目标系统准备、编辑和送出命令。

这样，利用这些工具，用户可以实施测试并校验结果。这些工具还允许用户以检验真实系统相同的方式来检验被仿真的系统。对于外部工具而言，被仿真的系统和实际系统是不可分辨的。

还存在一个要和仿真器内核 231 直接交互的需要。这是在主控制窗口中（MCW）完成的。在这样做时，可以检验被仿真系统的其它方面并改变其它性能。例如，可能会希望为调度程序改变伪进程的次序，或者改变伪进程的执行时间片的大小。通过主控制窗口（MCW）的交互作用还可以用作为可与外部工具相比拟附加功能。不过，在这方面可以利用的东西完全取决于不同子系统的设计人员。

2.2.2 命令分层

仿真系统 200 中的每个子系统都有它自己的解释器。子系统和它们的解释器如图 5 所示按“框架”树形层次组织。在图 5 中，各仿真子系统相当于目标电信系统中实际被仿真的硬件。框架的其它部分用来作为管理功能，例如命令处理器等等。子系统仿真的事例是由称作为“工厂”的特定子系统示例的。

利用命令来接入仿真系统 200 的各个不同子系统。每个子系统可以利用导航器命令以到达它的父节点和它的子节点。也可以一步就到达根节点（FrameWork Main）。利用这些导航命令有可能航行遍历整个树。在启动时，FrameWork Main 是当前的活动节点，亦即它的解释器执行任何进入的命令。如果下一个命令是导航命令，那么就改变活动节点，因而也就改变了执行进入命令的解释器。命令有四种基本类型：全局的、局部的、事例的和 Tcl。

全局命令是为所有的解释器定义的并在解释器被建立时加进去。全

局命令在所有的解释器内具有相同的行为。

局部命令由子系统本身定义，而且只有在子系统是活动的子系统时才能使用。局部命令有一个‘_’前缀。

事例命令用于到达子节点并在子系统被建立时加到当前的子系统中。这些命令实际上是局部命令。事例命令有一个‘.’前缀。

可以得到一组 Tcl 定义的命令的子集。这些命令实际上是全局命令。

子系统的分层组织（以及它们的解释器）意味着为了和某个子系统相互作用，用户首先要用导航命令移动到那里，然后再给出所需的命令。要在另外的内核部件中执行命令的另一个可能性是利用“do”（执行）命令，如下面所述。

2.2.2.1 全局命令

全局命令对所有子系统都是公共的。全局命令可以在任何级别上使用，并且在子系统建立时自动加到命令集中。这也意味着全局命令是保留命令并且不能由任何个别的子系统所重新定义。

全局命令中的一组被定义为用于在子系统树形分层结构中导航并执行个别命令:

“up”（上）——在分层中向上移动一个级别。本命令在顶层级别时无作用。

“top”（顶）——直接移动到最上级别。本命令在顶层级别时无作用。

“do”（执行）——在层次中的不同处执行一个命令。例如“do/schd/_status”在调度程序内核部件中执行 Status 命令。

另一个全局命令用于原本文件，具体地是一个“insert file”（“插入文件”）命令。“插入文件”命令从“文件”读出命令，这个文件是一个普通的文本文件，含有提交给当前解释器的命令。它也可以含有嵌套的“插入”命令。

有三个命令用于控制仿真器内核 231 的执行。这三个命令是“halt”（“停止”）、“run[n]”（“执行[n]”）、和“thread”（“线程”）命令。“停止”命令使仿真器内核 231 的执行停止。在停止时，仿真器只响应（从主控制窗口(MCW)来的）命令。“执行[n]”命令使仿真器内核 231 的执行继续进行。这个命令释放内核 231 的伪进程。如果规定了“n”，则调度程序执行“n”个周期。如果没有规定参数，则调度程序

执行到给出“停止”命令为止。“线程”命令强迫对调度程序的控制，使其执行进程线程直到分析了另外的命令。

另外一组命令用于显示被仿真系统中各部件的信息。这些命令是：“showall”（“显示全部”）“info”（“信息”）、“？”、“trace level”（“跟踪级别”）、“prompt”（“提示”）、以及“resetall”（“全部复位”）。“显示全部”命令为已定义的部件调用显示功能（见下面的“_Show”）。然后它把“显示全部”命令转发到所有的子系统。接着每个子节点以同样方式执行命令。“信息”命令显示有关当前事例的与登录有关的信息。而“?”命令为当前子系统列出全部已定义的命令。“跟踪级别”命令为当前的事例/工厂设置其跟踪级别成规定的级别。“提示”命令规定提示字串的格式。“全部复位”命令通过为全部已登录的模块调用复位功能而使仿真系统200复位。

2.2.2.2 局部命令

局部命令可以在登录以后的任意时间加到子系统的解释器中。局部命令有一个“_”前缀，这是在建立时间自动加上去的。有几个预定义的局部命令用于调用已登录的功能，特别是“显示”和“复位”命令。“显示”命令显示关于当前部件的详细信息。由于这种信息随不同的部件而在格式和内容上都有变化，因此每个部件必须有一个与该部件一起登录的显示功能。“复位”命令调用当前事例的复位功能。这个功能一般仅用在硬件仿真模块中。对复位功能的调用应该具有和对被仿真的硬件的发送复位操作时相同的语义。

2.3.2 事例命令

当一个子系统被建立时，对父辈子系统即当前的（活动的）子系统要加一个导航命令。通过调用这个命令，当前子系统被改变成子辈子系统。事例命令有一个前缀‘.’。在分层结构中向上导航的命令（“上”和“顶”）被认为是全局命令。

2.2.4 Tcl命令

由Tcl程序库定义的若干命令可以在每个子系统中获得。这些命令没有前缀并可认为是全局命令。

2.2.5 命令解释器（程序）

如上所述，仿真系统200的内核231的一个普遍设计原则是可以从用户接口加以控制的每个内核部件应该有它自己的命令解释器和自己的

命令集。所有的解释器有必要以相同的方式工作，也就是说，它们遵循相同的分析命令的原则，并且它们以相同的方式把作用（功能）结合到命令上。为此，所有的解释器都使用 Tcl 来实现。这也使得各解释器能够引进每个解释器都应理解的命令。

所有的命令解释器都从命令处理器调用。总是严格地有一个解释器供调用，即当前命令解释器 CCI。来自解释器的任何输出，例如确认，应该使用内存的 Tcl 命令“Tcl_Append Result”（“Tcl－添加结果”）或“Tcl_SetResult”（Tcl_设置结果）。

2.3 CP 仿真器

在所说明的实施例中，CP 仿真器 234（见图 4）仿真 APZ 212 20 处理器，它包括指令处理器（IPU）80 的完整的指令集和寄存器。这使得 APZ 和 APT 软件的真正的作业处理成为可能。信号处理单元（SPU）84 和区域处理管理器（RPH）92 是在接口级别上被仿真的。CP 仿真器 234 是设计成以高速度来执行目标代码（即被仿真系统的代码）的。

如下面将更详细地说明的那样（例如在 3.4 节中），将要去执行的目标系统的代码线程是一步步递增地翻译成为主机（SPARC）二进制代码的。然后二进制代码被执行。二进制代码还被存储起来以便在进入同一线程时再次使用，从而提供例如良好的执行性能。

2.4 输入/输出（IOG）仿真器

输入/输出仿真（IOG3）264（见图 4）支持加载、转储和功能改变的基本功能。它支持 WIOL 210 的用户接口。WIOL 210 和 Autosis 214 是连接到信道上的。Autosis 是一个用来建立和执行测试指令文件的测试系统。Autosis 214 的核心是它的解释器，它能够执行测试指令，即把命令送到正在测试的设备并分析送回的应答。Autosis 214 适合于 AXE 系统，并控制各种话务发生器。

2.4.1 SIGEN 话务发生器

SIGEN 话务发生器 220 和现有的 IGEN_原本和 Autosis 214 完全兼容。

2.4.1.1 概述

SIGEN 命令分成两部分。第一组在 SIGEN 的工厂级别使用，用于建立和删除 SIGEN 事例（建立和删除）同时也用于打印 SIGEN 配置和状态（显示）。第二组在 SIGEN 事例级别上使用，用于在 ISDN 接入和 DLB2

线路之间建立模拟连接（联接和断开）。开头两个命令是在 SIGEN 级别，而后者是在 SIGEN 的每个事例上使用的，如图 6 所示。

2.5 集团交换机仿真器

2.5.1 集团交换机仿真器概览

仿真系统 200 环境下的仿真的集团交换机（SGS）模块 230 可以模仿集团交换机子系统（GSS）30 的功能的一个子集。这种仿真在例如 ISDN 基本速率接入（BRAC）的自动功能测试中是需要的。

仿真集团交换机（SGS）230 的主要目的是维护在集团交换机子系统（GSS）30 的硬件中所建立的途径的信息。这种信息用来模拟由 SIGEN 所仿真的 ISDN 终端之间的端到端的信道和模拟在这些信道中的带内信令，也就是送出和收到的声音和数值。这是在 ISDN BRAC 功能测试中所用的基本手段以便验证被测试的 ISDN 业务和所建立（仿真）的连接。

GS 仿真模块 230 主要考虑 TSM 272 和 CLT 274 块（见图 4）。由于仿真系统 200 提供 CP 仿真器 234 来仿真 CP 34 和 TSM 72、CLT 74 的中心软件，同时由于其它 GSS 块是在仿真的 CP 234 上执行的，因此没有必要在 SGS 中仿真 TSMU 和 CLTU 的功能。图 7A 表示 TSMU 和 CLTU 功能是在 CP 仿真器 234 中执行的，并表示 GS 仿真 230 模拟区域软件（用于 TSMR 和 CLTR）和硬件（用于 TSM 和 CLM）。

2.5.2 集团交换机仿真器的功能

2.5.2.1 内核的被动部分

仿真的集团交换机模块 230 是内核 231 的被动部分。这意味着仿真的集团交换机模块 230 在其内部并不含有任何“进程”，因此它就不是内核 231 的分时策略的一个对象。仿真的集团交换机模块 230 只是在仿真环境中其它部分发出请求时，即从仿真的 CP 34 送出的信号、过程调用、以及低级别命令时才工作。仿真的集团交换机模块 230 的基本元件示于图 7B 中并在下面讨论。

2.5.2.2 集团交换机仿真器的基本元件

仿真的集团交换机模块 230 的基本单元（见图 7B）包括命令解释器 230－1；硬件配置 230－2；TSMR 信号接口 230－3；CLTR 信号接口 230－4；TSMR 仿真器 230－5；集团交换机硬件仿真器 230－6；途径仿真器 230－7；以及 CLTR 仿真器 230－8。

命令解释器 230－1 包括用于处理集团交换机硬件配置命令的例程。

进入到主配置窗口（MCW）的命令是通过一个接插座而传送到内核 231 的，并由一个命令调度器传递给仿真的集团交换机 230。

硬件配置 230－2 是一种功能，用于管理仿真的集团交换机的硬件，即存储分配和为控制它们的仿真 TSM 和区域处理器（RPS）建立数据结构。

TRMR 信号接口 230－3 是一种功能，用于处理送至和来自 TSMR 仿真器 230－5 的 CP－RP 信号。这些信号是在内部分配到正确的区域处理器的。

CLTR 信号接口 230－4 是一种功能，用于处理送至和来自 CLTR 仿真器 230－8 的 CP－RP 信号。

TSMR 仿真器 230－5 是一种功能，用于仿真 TSMR 区域软件所需的元件。所有进入的信号都被接收，但是和连接的建立和取消无关的信号将导致哑响应。

集团交换机硬件仿真器 230－6 包含一些例程，用于仿真集团交换机硬件的各个元件－主要是 TSM 数据存储器。集团交换机硬件仿真器 230－6 由 TSMR 仿真器 230－5 控制。CSC 和 CSAB 存储器含有和在实际 TSM 72 中严格相同的信息，而 SSA 和 SSB 存储着集团交换机中关于带内信令的信息，例如最近送出的声音和字节。

途径仿真器 230－7 是一种功能，用于分析在集团交换机硬件仿真器 230－6 中的途径，通过这些途径可以传播带内信号，也就是说，给出一个“说话方”MUP，本功能将找出连接的“接听方”MUP，并向它送出一个准音调。

CLTR 仿真器 230－8 是一种功能，用于仿真 CLTR 区域软件的基本元件，例如维护 CLM 的状态。所有进入的信号都被接收，但大多数信号都导致哑响应。

2.5.3 用于集团交换机仿真器的通用命令

集团交换机硬件配置 230－2 包括建立和删除区域处理器 232 所用的命令。区域处理器 232 控制 TSM72 和 CLT 74，并且控制用于打印仿真的集团交换机配置和状态的“显示”命令。用于集团交换机仿真器 230 的命令的位置示于图 7C 中。

2.6 会议呼叫设备（CCD）仿真

2.6.1 会议呼叫设备仿真器概览

仿真系统200的仿真会议呼叫设备（CCD）模块（SCCD）262（见图4）对 CCD 功能的那些元件进行仿真，这些元件是 ISDN 基本速率接入（BRAC）的自动功能测试所必需的。

仿真会议呼叫设备（SCCD）262 的主要目的是维护关于在会议呼叫设备（CCD）硬件中建立的会议连接的信息和所产生的去往通过集团交换机而与 CCD RW 相互连接的用户线路上的声音信息。这个信息对于发送和接收像声音或数值这样的在 B 信道上的带内信令进行仿真是必需的，它是在BRAC功能测试中用来验证（仿真的）多方连接的基本手段。

仿真会议呼叫设备 SCCD 262 提供 CCDR 及会议呼叫设备（CCD）硬件的信号接口和选定的功能。CCDU 是在仿真 CP 234 上执行的，因此它不是会议呼叫设备（CCD）仿真的对象。区域软件的仿真是必需的，因为仿真系统 200 并不为区域处理器提供仿真。会议呼叫设备仿真的范围示于图 8A 中。

由于 SCCD 262 同时包含了硬件和区域软件功能，如图 8A 所示，所以就没有必要去仿真在 CCDR 和会议呼叫设备（CCD）硬件之间的硬件接口（即 EM总线）。仿真的 CCDR 直接在仿真的 CCD HW 的存储器上“运行”。RP 总线的功能也被限制在简化了的 CP－RP 信号传递。

2.6.2 会议呼叫设备仿真器的功能

2.6.2.1 内核的被动部分

仿真会议呼叫设备（SCCD）模块 262 是仿真系统 200 的内核 231 的被动部分。这就是说 SCCD 262 在其内部没有任何“进程”，因此它不是内核 231 分时策略的一个对象。SCCD 262 只是在仿真系统 200 的其它部分发出请求时（也就是说，从仿真的 CD 234 送出信号、过程调用和低级别命令时）才起作用。

2.6.2.2 仿真的会议呼叫设备的基本元件

仿真的会议呼叫设备（CCD）262 的总体结构示于图 8B 中。SCCD 模块 262 的基本元件是命令解释器 262－1；硬件配置 262－2；信号接口 262－3；CCDR 仿真器 262－4；硬件仿真器 262－5；以及会议仿真器 262－6。

命令解释器 262－1 包括用于处理 CCD 硬件配置命令的例程。进入主配置窗口（MCW）的命令要通过一个接插口发送到内核 231 并由命令调度器传递到 SCCD 262。

硬件配置 262－2 是一种功能，用于管理 CCD 硬件仿真器 262－5，即为仿真的 CCDS 和控制它们的区域处理器分配存储器和建立数据结构。

信号接口 262－3 是一种功能，它处理送至和来自 CCDR 仿真器 262－4 的 CP－RP 信号。所有进入的信号都被接收，但那些与连接的建立和取消无关的信号将导致哑响应。

CCDR 仿真器 262－4 是一种功能，它模拟区域软件的必需的元件，例如 CCD 连接的建立和取消，送出仿真的声音等。

硬件仿真器 262－5 包括用于仿真 CCD 硬件的各元件的例程－这主要是 CCD 数据存储器，它含有连接的状态、它们的衰减、最近连接的声音（韵律和频率）等等。

会议仿真器 262－6 是一种功能，用于分析在 CCD HW 仿真器 262－5 中的会议连接和把带内信号（例如模拟的声音）传播到它们的参加者去。

2.6.3 会议呼叫设备命令

图 8C 表示会议呼叫设备（CCD）仿真器 262 用的命令的位置。

2.7 STC 仿真

2.7.1 STC 仿真器概览

仿真系统 200 的 STC （信号转移控制）可以仿真模块 238 模拟 STC 功能的一部分，这部分功能是对 ISDN 基本速率接入（BRAC）进行自动功能测试所需要的。在所说明的实施例中，STC 模块是作为仿真系统 200 的内核 231 的一个整体部分而在 SUN Sparc 工作站上在 solaris 2.5 的控制下运行的。

仿真 STC （信号转移控制）238 的主要任务是为加载和与仿真的 EMRPD 250 的信号通信提供手段。STC 是连接到区域处理器总线 36 上的一个特殊类型的区域处理器。在 CP 34 和 EMRPD 50 之间的信号转移（包括加载）要通过区域处理器总线（RPB）36、一对处理器 STC 38 和 STR 58，和由 STR 58 控制的 EMRP 总线（EMRPB）（见图 1A）。

在实际的 AXE 输入/输出系统中，每一对 STC 38 和 STR 58 是由 SS7 链路（ITU－T 公共信道信令系统 7 号，级别 2）连接的，正常情况下是基于一阶 PCM 连接。SS7 链路启动远程用户级的控制。仿真系统 200 对这个链路不作任何仿真，因为在 STC38 和 STR 58 之间的位号可以用过程调用来转移。

图 9A 表示按照本发明的实施例的信号转移控制仿真器的范围。如图 9A 所示，STC 模块 238 包括 STCE（STC 执行程序）和区域程序 CLCR（控制信令链路中心）的功能仿真。STC 执行程序为 RPB 格式和 EMRPB 格式之间的信号转换和在 I/O 转移的协助下 EMRP（D）的加载的管理提供机制。它还为 CLCR 提供作业管理和程序执行的支持并提供 RP 维护所需的功能。

CLCR 程序处理在信号发送、链路测试、和系统再启动时的 SS7 信令链路。CLCR 和 STCP 及 DCI 硬件部件相互配合工作。

2.7.2 STC 仿真器的功能

2.7.2.1 内核的被动部分

STC 仿真模块 238 是仿真系统 200 的内核 231 的被动部分。这就是说 STC 仿真器 238 并不包含任何伪进程，因此，它也不受内核 231 的分时策略的支配。STC 仿真模块 238 仅在被仿真的环境的其它部分发出请求的情况下才动作，即，从仿真的 CP 234 和 EMRPD 250（通过 STR）发来的信号、过程调用、和用户命令。

2.7.2.2 STC 仿真器的基本元件

图 9B 所示的 STC 仿真器 238 的总体结构包括 STC 工厂 238－1；STC 命令解释器 238－2；STC 硬件配置 238－3；RPB 信号接口 238－4；STR 信号接口 238－5；STCE 仿真器 238－6；和 CLCR 仿真器 238－7。

STC 工厂 238－1 是用于处理 STC 事例的一种功能。STC 命令解释器 238－2 是用于处理定义 STC 事例的配置命令的一种功能。这些命令进入到主控制窗口（MCW），发送到内核 231 并由命令调度器传递给 SGS 230。

STC 硬件配置 238－3 是一种管理 STC 事例的配置的一种功能，该事例包括到区域处理控制器（RPH）的链路、到仿真 STR 的链路和由 CP－RP 信号初始化的地址变换表 EMITAB、STCTAB 和 EMOTAB。

RPB 信号接口 238－4 是一种功能，用于处理送至和来自 STCE 仿真器 238－6 的 CP－RP 信号。该接口涉及设备块信号、操作系统信号和 I/O 转移信号。从 CP 234 收到的信号在 STC 中端接或转移到 STR。

STR 信号接口 238－5 是一种功能，用于处理进入的或外出到 STR 仿真器 258 的信令。信号用过程调用发送或接收。在收发两侧都使用一个简单的以确认信号为基础的协议。

STCE 仿真器 238－6 是一种功能，用于仿真 STC 执行程序的各种功

能，主要包括把在 CP 34 和 EMRPB 50 之间转移的信号从 RPB 格式转换到 EMRPB 格式或其反向转换。该功能使用地址转换表 EMITAB、STCTAB、和 EMOTAB。

CLCR 仿真器 238－7 是一种功能，用于响应由 CLCU 送到 CLCR 的 CP－RP 信号。它包括用于链路初始化、测试和阻断的信号。由于 SS7 链路没有被仿真，信号具有哑特性。

2.7.3 STC 命令

图 9C 是表明信号转移控制仿真器的命令位置的示意图。

2.8 STR 仿真

2.8.1 概览

STR 仿真模块 258 模拟 STR 硬件部件 58（见图 1A）的行为。仿真是功能性的，这就是说，被仿真的是 STR 对外部信号的响应，而 STC 58 的内部工作情况则仅仅被仿真到所需要的程度。

如图 1A 所示，STR 58 从 EMRPB 56 和 STC 38 接收信号。这些信号中的一部分由 STR 58 本身来处理，而其余信号则分别转送到 STC 38 和 EMRPB 56。

图 10A 表示 STR 仿真器 258 和它的信号流。具体地说，图 10A 表明从 STC 238 来的信号被存放在输入缓冲 linkin 中并被转送到输入队列 linkib。从这个队列每次取出一个信号并由 STR 执行程序去处理。从 STC 仿真器 238 来的信号可以被转向到 STR 仿真器 258 本身内部的一个模块，也可以进一步送到 EMRPB 256。在后一种情况下，信号在复制到输出缓冲 busout 之前先在缓冲队列 busob 中排队。反过来，从 EMRPB 256 来的信号每次一个存在输入缓冲 busin，然后再在 busib 中排队。然后由 STR 执行程序每次取出一个信号，在 STR 仿真器 258 内部处理，或者经过缓冲队列 linkob 和输出缓冲 sndtab 送到 STC 仿真器 238。

为了使区域处理器能够独立运行，从 EMRPB 256 来的信号也可以立即返回到输出队列 busob 然后再到 EMRPB（以及另一个区域处理器）。这一个功能仅在系统的其它部分发生误操作时才使用，以便区域处理器可以继续运行。

在目标系统（见图 1A）中，STR 58 和 STC38 之间的信号是用 CCITT #7 2 级协议发送的。在仿真系统中，这些模块之间的信号因此也在没有任何协议开销的情况下被缓冲。

在仿真系统中信号的发送和接收也得到了简化。信号的异步发送和接收已被同步的功能调用进行了互换。这样，信号是作为功能调用的参量而发送的。信号将在收到的伪进程下一次允许执行时被接收。不过，到 EMRPB 的信号要在 STR 仿真器 258 中排队并且逐一地送到接收方。

此外，总线管理、电缆选择、和 CRC 计算并不包含在所说明的实施例的 STR 仿真器 258 中。STR 仿真器 258 的结构和它周围模块的合作按照图 10B 所示的原则进行。

2.8.2 STR 仿真器的功能

2.8.2.1 内核子系统

STR 仿真器 258 模块是作为仿真系统 200 的内核 231 中的一个子系统而实现的，并作为一个伪进程独立地执行。因此，它从 EMRP 和 STC 仿真器 238 取出进程并传播信号。这些都是从队列中和缓冲中取出的、这些进程是由周围的模块在它们执行时放在那里的。

2.8.2.2 分配图

图 10C 表明 STR 仿真器 258 是怎样和 EMRP 250 和 STC 仿真器 238 通信的。STR 仿真器 258 向 EMRP 客户提供一个外部可调用的功能接口，客户则反过来为 STR 仿真器 258 规定和功能指针的连接点。与此相似，STR 仿真器 258 向 STC 仿真器 238 提供和功能指针的连接点。

2.8.2.3 接口

朝向 STR 仿真器 258 的接口有三部分。这三部分包括由 STR 仿真器 258 定义的功能（见 2.8.2.4 节）；由 EMRPB 客户提供的功能（这些功能在客户联结到 STR 时被定义）（见 2.8.2.5 节）；以及，由 STR 仿真器 258 向 STC 仿真器 238 提供的功能（这些功能在 STR 仿真器 258 作为客户联结到 STC 仿真器 238 时被定义）（见 2.8.2.5 节）。

2.8.2.4 由 STR 仿真器定义的功能

下列功能由 STR 仿真器 258 定义:

STR_Attach TO EMRPB

本功能把客户附着到 EMRPB 上并把指针转移到能使 STR 控制它的客户的功能上。

STR_ToServer

本功能在客户需要向 STR 发送数据时使用。信号必须用 STR_CreateEMRPBData 建立，数据必须由服务器解除分配。

下面三个功能分别是建立、删除和复制 EMRP 数据用的帮助功能:

STR_CreateEMRPBData

STR_DestroyEMRPBData

STR_CopyEMRPBData

2.8.2.5 由 EMRP 客户提供的功能

由客户提供的功能定义如下(这些功能并不一定要保持下列名字):

STR_ToClient

这是在把数据送到附属客户去时所用功能的指针。数据由服务器解除分配。

STR_ResetClienT

这是指向用于复位客户的功能的指针。

STR_BlockClient

这是指向用于对客户加锁/开锁的功能的指针。

2.8.2.6 向 STC 仿真器提供的功能

STR_FromSerrer 功能是由客户提供的功能,但不一定要保持此名字。

2.9 ST20C 仿真

被仿真的接入处理器(此后称之为 ST20C 251)具有能对目标处理器完全仿真的指令和寄存器。这使得目标模块既能对控制系统也能对应用程序进行加载和执行。ST20C 仿真器 251 和连接的设备在逻辑级别上相互作用。

2.9.1 目标 ST20C 的主接口

图 11A 提供了一个系统概览图,它的重点在于在目标系统即 AXE-10 中用于 BRA 的 ISDN 控制信号流。图 11A 表明有三个主要的 ST20C 251 接口。第一个接口是 EMRPB 56,这是把 EMRPD 50 连接到 CP 34 的总线。第二接口是 DEVCB,是控制所附属的设备板的设备处理器总线。第三接口是 EMRPD BB,是 ST20C 微计算机的局部总线的延伸。这些接口在下列各段中将详细说明。

2.9.1.1 目标 ST20C 用的 EMRPB 接口

EMRP 50 和中央处理器(CP)通过 STR-STC 信令技术对进行通信。STC 38 连接到区域处理器总线(RPB)36 并负责将带有 RP/EM 地址的区域处理器信号转换成带有 EMRP 地址的 EMRP 信号。STR58 是 EMRPB56 的

总线主控器，该总线把所有 EMRP50 都连接到 STR 58 因而也连接到 CP 34。EMRPB 56 是双总线，所有 EMRP 50 连接到两根总线，每根总线各有一个 STR 58。

EMRPB 56 使用一个具有比特取向的协议的同步点　到多点总线。每个 EMRP50 具有一个范围在 0 到 31 之间的独特地址，这是用连接在 EMRP 50 的前面插件上的地址插头来选择的。

STR 58 通过向每个 EMRP 50 的地址顺序地送出查询信号而轮流地查询所有的 EMRP 50。当某址被选中时，该 EMRP 50 如果有某些东西需要送出就用一个发送请求信号来响应。在主控器和选中的 EMRP 50 之间经历某些信号交换之后，EMRP 50 就可以送出它的信号。

如果 STR 58 需要送出信号到一个 EMRP 50，则它暂时停止查询过程并用目的 EMRP 50 的地址送出信号。

STR 58 也可以用一个特殊的广播地址向所有的 EMRP 50 送出信号。

EMRP 50 每次只能在总线的一侧接收和发送信号。具体用哪一侧可以由 STR 58 以 EMRP/STR 协议的较高级别来设置。

为了阻断或复位 EMRP50，STR 58 发出一个特殊的信号事项。

可以在 EMRPB 56 上传输的一帧中的最大数量是 64 字节。

2.9.1.2 目标 ST20C 用的 DEVCB 接口

ST20C 51 和它附属的设备处理器（DEVPS）通过一个称为 DEVCB 的总线进行通信。这是一个异步半双工串行总线，这里 ST20C 是主控器。

在 ST20C 51 上，使用了一个门阵列 LCA－DEVCB 来实施 DEVCB 接口的主控侧。这个单元包括三个主要部分：扫描器、接收器和发送器。

正常情况下，扫描器通过证实各设备的地址来逐一地对所有设备寻址。如果附属设备中的某一个有东西要发送，它确立一个标志信号。这使得总线控制器产生一次中断。如果相应的各种条件都能满足，则 ST20C CPU 送出一个继续进行的信号并等待从被动侧送来的供接收的信号。

当被动侧收到了继续进行的信号时，它可以送出它的信息。主控器对每个收到的字节都产生一次中断以使 CPU 去读出该收到的字节。信息的传送由定时器监控，如果信息中的下一个字节来得太晚或已丢失，则产生一次中断。

从主控侧向被动侧发送信息是通过对设备寻址并逐个字节发送信息而完成的。这个过程也是由中断驱动的。

设备可以通过确立它的地址并在总线上确定一个复位信号来使其复位。

2.9.1.3 目标 ST20C 的 EMRPD BB 接口

EMRPD BB 实际就是 CPU 总线的一个延伸，这使它能够向附属于这个总线的外部板发出常规的存储器访问。

2.9.2 ST20C 仿真

在本节中把重点收缩到 ST20C 本身并且叙述它的体系结构。另外，这个体系结构和被仿真系统的结构之间的映射也在模块的级别上描绘。

2.9.2.1 ST20C 的体系结构图

图 11B 表示信令终端 ST20C 的主要部件以及这些部件的连接。

2.9.2.2 ST20C 对其仿真的映射

被仿真的 ST20C 251 是按照图 11C 组织的。ST20C 在大多数方面反映了图 11B 的硬件体系。不过，根据仿真器的需要，仿真细节的级别可以在不同的 ST20C 部件之间有所不同。这在以下各节中将进一步叙述。

2.9.3 ISS 68020（ST20C 微计算机的指令级别仿真）

ST20C 是以摩托罗拉（Motorola）68020 微处理器为基础的。指令级别的仿真支持一个完整的指令集，同时还支持由微处理器提供的执行模式、中断和包括硬件复位在内的异常处理。

ISS 68020 是以性能优化的体系结构为基础的，它完全实现了摩托罗拉 68030 微处理器的指令集，包括完整的 MMU 仿真。

2.9.4 基本原理

ISS 68030 的一个重要原理在于它的译码和调度技术。这也是有助于它的速度的一个关键特性。这同一个原理也用于 ISS 68020 中。这个技术按如下方式工作。指令代码（实际上是开头的 16 比特）被用作为对调度表的索引，该调度表含有 SPARC 代码序列的地址。这些代码序列模仿 M680X0 指令的行为，从而具有和解释这些指令相同的效果。这可以有效地避免所有不必要的开销。但是，当访问内存时，就必须由特殊的功能进行有效地址的计算。另外，代码序列是从 M680X0 指令的语法和语义的描述而产生的，仿真代码是被自动地收集到表中的。

图 11D 说明了使 ISS 68030 适应于在 ST20C 环境下的 ISS 68020 的需要的情况。

2.10 线路接口板仿真

2.10.1 主要接口

图 12A 提供了在一个以 AXE－10 为范例的特定目标系统中用于 BRA 的 ISDN 控制信号流（D 信道）的概览图。具体而言，图 12A 表示有 5 个 BRA 设备板组合的主要接口。第一个接口是 DSS1，即用户线路接口。第二接口是 DEVSB，是用于在设备和时间开关之间交换数据的总线。第三接口是 DEVCB，是控制附属设备板的设备处理器总线。第四接口是 EMRPD BB，是 ST20C 微计算机的局部总线的延伸。第五接口是称之为“测试总线”的测试设备接入总线。这些接口和设备板将在下列各节中详细说明。

2.10.1.1 电路板 DLB2－5U（ROF 1377828/1 R7B）

电路板 DLB 2－5U 实现用户线路接口。电路板 DLB2－5U 提供 4 条用户线路。在电路板 DLB2－5U 上只实现层次 1（物理层）的功能。这些层次 1 的功能是 NT（网络终端）的电源馈送；2B IQ 转换；B－和 D－信道比特流的在比特层次上的多路复用/多路分解；以及语言总线接口。设备处理器控制该电路板并执行由 EMRPD 通过 DEVCB 下达的各种任务。

2.10.1.2 时间开关 TSW3

时间开关实现在数字用户级的时间切换功能。该电路板包括一个设备处理器，它主要操纵两个软件单元：TSD 和 CDD。在设备处理器和 EMRPD 之间的信令通过设备控制总线（DEVCB）来完成。TSD 和 CDD 使用两个独立的 DEVCB 身份。

2.10.1.3 信令终端板 ST641

ST641 是一块信令技术电路板，用于端接多达 64 个 D 信道的朝向 ISDN 基本速率用户接入的层次 2 的信令数据话务和去到及来自中央处理器的被传送的数据，ST641 用 ST20C 的底板总线接到 ST20C，并由设备语言总线（DEVSB）接到时间开关。

2.10.1.4 DEVCB（设备控制总线）

ST20C 通过一条叫做 DEVCB 的总线和它所附属的设备处理器通信。这是一个异步半双工串行总线，这里 ST20C 是主控方。

2.10.1.5 EMRPD BB

EMRPD BB 其实就是 CPU 总线的延伸，这使它能够向附着在这一总线上的外部电路板发出正常的存储器访问。

2.10.1.6 DEVSB（设备语言总线）

DEVSB 是一个 PCM 总线，它把时间开关和一系列设备相连接，这些

设备可以是任何一种单独地直接连接到 DEVSB 上的硬件。

2.10.1.7 测试总线

通过在 DLB2-5U 上的测试接入继电器的操作，可以把一条线路从用户方断开并连接到一个用于对线路或电路板进行测量的测试接入总线。

2.10.2 设备板仿真器概览

对 BRA 设备板进行仿真的目的是提供一种能被仿真的话务发生器（SIGEN）220 所利用的接口。这个接口支持并模拟考虑到硬件配置的 DLB2－5U 线路接口，也就是考虑到板的位置和电路板实现的是 4 个线路接口中的哪一个接口。这个接口只支持传输 LAPD 层次 2 的 D 信道帧的内容，即相当于在用户侧和交换设备之间的、在打开标志和关闭标志之间的 8 位字节流（没有比特填充）。

一个进入的帧被赋予一个时间片号码，它和在分配给实际电路板位置的 PCM 帧中所用的哪一根用户线路有关。实际电路板的位置是在仿真系统 200 的内核 231 的配置阶段定义的。

为了支持集团交换机的仿真，被仿真的集团交换机 230，即设备板仿真器（DBS）提供一系列功能以便通过设备板组合来解决 B 信道的途径问题。图 12B 表示通过实现 DBS 的软件的数据流。DBS 并不模拟 PCM 链路。途径是根据时间片的赋值来分析的，通过设备板组合可以使含有 LAP－D 帧的缓冲器从它的源移动到它正确的目的地。

2.11 SIGEN 话务发生器

SIGEN 话务发生器 220（见图 4）能在仿真系统 200 中进行 ISDN 基本速率接入的自动功能测试。SIGEN 220 可以执行以 IGEN 原本语言编写入的测试案例（CP 或控制程序）。SIGEN 220 既可以从类似于 IGEN 的命令窗口提供局部控制，也可以从 AUTOSIS 提供远程控制。

从功能上说，SIGEN 220 是现有的 AXE 系统所用的话务发生器即 IGEN 的等同物。两个程序都能够运行同样的基本速率接入测试，且它们的容量是完全相同的。唯一的差别在于主机和目标的环境。IGEN 在 PC 机上运行并用于测试实际的 AXE 系统，而 SIGEN 220 在 SUN SPARC 工作站上运行并用于在仿真系统 200 上执行的测试 AXE 软件。AXE 仿真器也运行在 SUN 工作站上。

2.11.1 话务发生器概览

一般说来，SIGEN 220 有两个主要部分：即 SIGEN－FRONT（它是 SIGEN

220 的通信部分），和 SIGEN－ENGIN（它是 SIGEN 220 的执行部分）。

SIGEN－FRONT 为编辑、加载、和执行 ISDN 功能测试提供全部必要的与用户有关的功能。FRONT 是作为一个独立的 UNIX 进程而实现的。在 FRONT 中实现的用户接口和 IGEN 中现有的接口是相似的。FRONT 还对 AUTOSIS 214 提供一个接口，即具有 AIG（AUTOSIS IGEN 驱动器）的 TENUX 信道。为此目的提供了一个辅助 UNIX 进程，即 TIF。

SIGEN－ENGIN 是 SIMAX 96 的内核 231 的一个整体部分，它和 SIGEN－FRONT 通过一个接插口通信。ENGIN 是一个模块，它编译并执行用 IGEN 原本编写的功能测试，并提供为执行测试所需要的所有的 ISDN 协议（DSSI）的元件。具体说它实现完整的 LAPD（层次 2）协议。

有可能动态地在一个仿真系统 200 中建立一系列的 SIGEN 事例，这相当于若干台正运行 IGEN 的 PC 机连接到一个 AXE 装备中。SIGEN 事例，和仿真系统 200 中所有其它元件一样，是按照由进入到公共的命令窗口的配置而建立的。

2.11.2 接口

SIGEN 220 的图形用户接口（SIGEN GUI）包括一个 X 窗口，这个窗口包括 SIGEN 主菜单、用户命令条、信号和出错登录窗口、访问和 CIN 状态板、计数器监视板。为了在现有的 IGEN 和 SIGEN 之间达到完全一致，SIGEN 窗口的布置和 IGEN 的窗口布置是非常相似的。SIGEN GUI 的方案示于图 13A 中。如图 13A 所示，主菜单含有加载和执行原本的命令，同时也含有与 AUTOSIS 214 建立连接的命令。

和 IGEN 不同，原本编辑功能是放在 SIGEN 窗口以外的，亦即原本的编辑是在一个分开的窗口中完成的。这给 SIGEN 的用户提供了一种自由，可以使用任何在 SUN OS 下可以使用的文字编辑程序，例如 emacs、vl、或 textedit 等。

在许多情况下，AUTOSIS 214 和仿真系统 200 环境可以在同一台工作站上运行。在这些情况下，TCP/IP 的物理和数据链路层是不需要使用的，这当然会提高 TIF－AIG 通信的总体效率。

仿真系统 200 为内核 231 的不同部分的配置和管理提供了一个统一的命令界面。仿真系统 200 的用户可以把命令送入主配置窗口（MCW）。命令是按分层结构安排的，这种层次可以在建立内核模块的新的事例时动态地重新安排。

SIGEN 命令分成两部分。第一部分是用于建立和删除 SIGEN 事例的（“create”和“delete”），而第二部分则用于在 SIGEN ISDN 接入和 DLB2 线路之间建立仿真的硬件连接（“cables”）（“connect”和“disconnect”）。第一部分的两个命令是在 SIGEN 层次上的，而后者可在每个 SIGEN 的事例下使用（有关命令说明的更为详尽细节见 2.11.3 节）。

2.11.3 命令

SIGEN 命令分成两个基本的组。第一组用于建立和删除 SIGEN 事例（“create 和 delete”），而第二组则用于在 ISDN 接入和 DLB2 线路之间建立模拟的连接（“connect”和“disconnect”）。第一组的两个命令是在 SIGEN 层次上的，而第二组则可在每个 SIGEN 事例下使用。

现在说明 SIGEN 命令的使用。建立新的 SIGEN 事例按以下方式完成:

FW.SIGEN>create SIGEN_name

与此相似，去除 SIGEN 事例的方式如下:

FW.SIGEN>delete SIGEN_name

为了在 SIGEN 事例的级别上访问命令，必须由用户送入事例的名字。然后命令提示符会改变以表明当前的事例:

FW.SIGENS SIGEN_name

FW.SIGEN . SIGEN_name>

为了在 SIGEN 接入和 DLB2 线路之间建立仿真的链路，用户应该送入“connect”命令并附有 DLB2 板的名字、线路号码和接入号码:

FW.SIGEN.SIGEN_name>connect DLB2_name line access

为了取消在 SIGEN 接入和 DLB2 线路之间的仿真链路，用户应该送入“disconnect”命令并附有接入号码:

FW.SIGEN.SIGEN_name>disconnect access

通过利用带有 ALL（全部）参数的“disconnect”命令也可以把 SIGEN 事例的所有已建立的链路全部取消:

FW.SIGEN.SIGEN_name>disconnect ALL

按照公共的仿真系统 200 内核规则，SIGEN 还提供“show”命令，它打印有关 SIGEN 事例的信息:

FW.SIGEN.SIGEN_name>show

FW.SIGEN.SIGEN name>……SIGEN info printout

在表 I 中，给出了包括一个名叫“TUBORG”的 SIGEN 事例的建立、配置和取消的命令对话的样本。“up”和“top”是在命令层次中任何一个级别上都可以使用的标准命令。

表 I

命令对话样本

```
FW >...
FW>SIGEN
FW.SIGEN >
FW.SIGEN > create TUBORG
FW.SIGEN > TUBORG
FW.SIGEN.TUBORG >
FW.SIGEN.TUBORG > connect DLB2-ONE 1 0
FW.SIGEN.TUBORG > connect DLB2-ONE 2 1
FW.SIGEN.TUBORG > connect DLB2-TWO 0 2
FW.SIGEN.TUBORG > show
FW.SIGEN.TUBORG >... TUBORG info printout
FW.SIGEN.TUBORG > up
FW.SIGEN >...
FW.SIGEN >...
FW.SIGEN > TUBORG
FW.SIGEN.TUBORG >
FW.SIGEN.TUBORG > disconnect 0
FW.SIGEN.TUBORG > disconnect 1
FW.SIGEN.TUBORG > disconnect 2
FW.SIGEN.TUBORG > up
FW.SIGEN >...
FW.SIGEN > delete TUBORG
FW.SIGEN > top
FW >...
```

2.11.3.1 打印输出

为了提供和现有 IGEN 的完全的兼容性，由 IGEN 所产生的所有打印输出的内容和格式都将保留不变。SIGEN 的打印输出或者送到 SIGEN 窗口（直接模式）或者送到 AUTOSIS 窗口（远程模式）。

有关内部出错和系统配置出错信息将通过主控制窗口中的报告管理程序打印输出。

2.11.3.2 数据和操作

SIGEN 的数据结构的概貌在图 13C 中提供。

3.0 功能说明

3.1 总体说明

如上所述，仿真系统 200 在一台主计算机系统上运行以模拟一个目

标电信系统。仿真系统包括仿真内核 231，它含有多个仿真子系统，每个仿真系统相应于目标电信系统的子系统。

3.2 目标机

如上所述，这里所说明的目标系统的例子是 AXE APZ 212 20。

3.2.1 指令集

目标系统的全部指令（例如 APZ 212 的各指令）都被仿真。这构成了下列指令组：从存储器读出的指令、写入到存储器的指令、寄存器指令、算术指令、移位和逻辑指令、跳转指令、信号传送指令、用于结束程序的指令、搜索指令、FOR 循环指令、从存储器读出的 OS 指令、写入到存储器的 OS 指令、区段转移的 OS 指令、寄存器处理的 OS 指令、送出信号的 OS 指令、特定的 CPS 目的的 OS 指令、特定的 MAS 目的的 OS 指令、以及其它 OS 指令。

3.2.2 寄存器

APZ 212 20 处理器在寄存器存储器（RM）中含有 4 组 192 个 32 比特的寄存器，每个优先级别 TRL、THL、CL 和 DL 各有一组。每一级中只有 64 个寄存器可以由常规软件直接编址。剩下 128 个寄存器只能由微程序或由特定的操作系统指令操纵。另外还存在 224 个由 4 个优先级跨级共享的寄存器。

64 个 32 比特的寄存器的情况如下：1 个变址寄存器（IR）、2 个指针寄存器（PR0、PR1）、1 个比较寄存器（CR）、2 个符号翻译寄存器（SR0、SR1）、24 个信号数据寄存器（DR0－DR23）、4 个算术寄存器（AR0－AR3）、30 个工作寄存器（WR0－WR29）。许多在软件中不能直接访问的寄存器可以用 OS 指令 XMOV 或 XRRM/XWRM 来访问。

为微程序保留的各寄存器中有一个寄存器是结果表示寄存器（RIR），它可以用 XMOV OS 指令从 ASA 被直接访问，但通常情况下当进行分支时（例如用 JOR 或 JZR）只作测试之用。它是一个简单的标志寄存器，它在算术指令完成后起进位标志的作用，在逻辑指令完成后用作为零标志，而在移位指令完成后起“精度损失”的标志的作用。

3.2.3 信号

改变执行流可以用两种方式之一来完成：局部跳转和信号。局部跳转只能在同一个程序块中进行。局部跳转还可以使用“跳转局部链接”指令而起到‘调用子例程’的功能。这些类型的跳转在其它 CPU 如 SPARC

这样的 CPU 中是众所周知的。

APZ 212 指令集并不包括间接跳转，但含有用于 C 类型开关功能的“由寄存器/存储器变址的按表跳转”的结构。

用跳转不能实现跳转到别的程序块去，这时必须利用信号来实现。（如果块号和当前块号相同的话，信号也可以用来实现局部跳转）信号和跳转相似，但它不是规定在同一程序块中的新的偏置值，而是规定一个信号。从一个程序块只能送出固定数量的信号而且必须列出在位于程序块前面的信号发送表中。根据信号指令的类型，可能会发生下列事情之一：

- 信号被立即发送去：程序的执行在目的地立即继续下去。信号可以用链接方式送出而使程序的执行返回到送出信号指令之后的那条指令。

- 信号可以进入一个作业缓冲区以便以后发送。

为了能够找到信号的目的地，要在全局信号分布表中查找各个信号。

当信号被送出时，它最终要变换成一个块号和在接收块中该信号的入口点的偏置值。

3.2.3.1 全局信号分布表

当一个程序块装入到程序存储器 87 时，信号分布表和信号发送表就合并到全局信号分布表中。每个信号也赋给一个全局信号号（GSN）。

如果信号只能被单独一个程序块接收，则它进入独特信号用的全局信号表（GSDT－U）中。如果信号可以被若干块接收，则它进入多个信号用的全局信号表（GSDT－M）中。这个表有一个阵列，它列出能够接收该信号的各程序块。当送出一个信号时，在这个阵列中搜索目的地址。全局信号分布表是 IPU（指令处理单元）80 中的一部分。

3.2.4 作业缓冲区

作业缓冲区用来存储已经送出但尚未被程序块接收的信号。这种情况发生在送出信号的指令要求它被延迟一个特定的时间段的时候，或者是信号已被放在一个特定的作业缓冲区时。作业缓冲区被定时扫描，这个时间是每当收到 CP－CP 或 RP－CP 信号时以及当程序执行结束时。如果信号已经到了要被送到它们目的地的时间，就把它们从作业缓冲区取出并发送。APZ 212 20 CP 含有 9 个作业缓冲区，按增加的优先级为序

分别是DL、CL、THL3、THL2、THLI、TRL3、TRL2、TRL1、和MAL。这些作业缓冲区按顺序扫描。如果高优先级的作业缓冲区中含有尚未送出的信号，则在扫描较低优先级的作业缓冲区之前它们先被送出。作业缓冲区是信号处理单元（SPU）84的一部分。

3.2.5存储器

如前面结合图2所述，下列三种存储器构成了主存储器：参考存储器（RS）88；程序存储器（PS）87；和数据存储器（DS）89。在APZ21220，DS和RS两者都存放在数据参考存储器（DRS）。参考存储器含有指向当前程序块中程序起始地址的指针和指向变量的指针（基地址）。这些指针能使CP 34找到目的地址。从程序块到跳转目的地和变量的所有参考值都是相对地址。CP 34使用参考存储器88以找到在主存储器中的实际地址。这使它实际上不可能访问并不属于本程序块中的数据/代码。和其它程序块交换信息的唯一方法是通过信号。

3.2.6优先级别

中央处理器34每一时刻执行单独一个程序，但是可以在它收到一个中断时切断‘上下文’。切换可以在5种不同的优先级别上进行：

1.MFL（误操作级）－最高级

2.TRL（跟踪级）包括TRL1、TRL2和TRL3

3.THL（话务处理级），包括THL1、THL2、THL3

4.CL（C级）

5.DL（D级）－最低级

作业只能被更高优先级的信号所中断。程序通常情况下在THL和CL级执行。APZ测量在TRL级发生。硬件出错的情况在MFL级处理。DL是用于特殊的非常低级别的进程的。MFL并没有它自己的寄存器组。

3.2.7字节存放次序（endianness）

和SPARC是小次序（little-endiad）不同，APZ 212的体系结构是大次序的（big-endian）。由于APZ指令编址模式的性质，尽管程序存储器87要按小次序来实现（按照16比特编址），目前不能期望这将导致任何性能上的损失。

3.3主机

如前所述，主机是一台SUN SPARC工作站。参见The SPARC Architecture Manual（SPARC体系结构手册），SPARC International，

第 8 版，Addison-wesley（1992），以了解这一结构的说明和提供的指令集。所有 SPARC 指令都是 32 比特，但（合成）指令集是例外，它们可能是 32 或 64 比特的。

3.4 执行模型

如在图 3A 中以原理图显示的那样，主机 110 的 RAM 130 含有一系列的例程和区（例如段或表）。举例而言，目标目的代码区 132，也称为目标指令存储器，用于存放为目标处理器编码的已编程的指令。存放在 RAM 130 的目标目的代码区 132 中的目标指令一般是从一个外部存储介质（例如通过 I/O 存储设备 120）而得到的，并按照需要经过转送（例如经过高速缓存）而进入 RAM 130。所谓“为未加说明的目标处理器而编码的”的意思是目标指令是按照目标处理器的指令集而格式化的，也就是说，目标指令是使用目标处理器的指令集作为汇编级别指令编写的。存放在目标目的代码区 132 中的目标指令是用来使目标处理器执行特定的活动和/或得到特定的结果的。不过，像在下面所说明的，存放在目标目的代码区 132 中的目标指令被改由主计算机 110 翻译成为主计算机 110 能够理解的指令以试图完成相同的特定活动和/或得到特定的结果。

在 RAM 130 中还含有各种由主处理器 112 执行的已编程的例程，包括仿真/模拟监控例程（ESB）133、解码例程 134、翻译例程 135、以及仿真执行例程 136。此外，RAM 130 还在其中存放了跳转表 140 和仿真指令存储区 142。

图 14 表示仿真监控例程是如何协调把目标目的代码翻译成仿真指令和由主机执行仿真指令的流水线式操作的。简单地说，当仿真监控例程确定应该是从目标目的代码区 132 取出指令的时刻时，解码例程 134 就对被取出的目标指令进行分析。解码例程 134 知道目标指令集的结构，分解取出的目标指令并确定它的类型，例如，该取出的指令是否为把两个寄存器内容相加的“加寄存器”指令，或是否为把一个寄存器的内容移到另一个寄存器或地址的“移动”指令，等等。对取出指令的分析的前提是解码例程 134 要理解目标指令集的预先规定的操作码和它的区段。利用由解码例程 134 所进行的分析，翻译程序 135 建造出一组仿真指令以便存储在仿真指令区 142。存放在区 142 中的仿真指令按主机处理器 112 的指令集进行格式化。根据取出的目标指令而产生的仿真指令

被构造成能使主机处理器 112 实现如目标处理器在执行该取出的目标指令时所实现的相同动作。仿真执行例程 136 从区 142 得到仿真指令以供主机处理器 112 去执行。当仿真执行例程 136 确定需要有更多的仿真指令去填充进仿真指令存储区 142 时，就给翻译例程 135 产生一个中断，如图 14 中的线 150 所示。作为对这一中断的响应，更多的目标指令被取出、解码、并翻译以提供给仿真指令存储区 142。

这样，目标处理器指令被动态地翻译成主机指令。这种翻译是按目标指令的序列而进行的。目标指令的这些序列存放在代码缓冲存储器 142 中。翻译是“在空中”进行的。主机指令被存储并供以后执行时重复使用。当执行返回到一个给定点时，指令已经翻译完成。这意味着主机代码是以一种连续流动的方式执行的，并且在主机系统中没有因高速缓存的充溢和流水线的中断而降低执行速度。在翻译过程中执行流是径直进行的。

本发明的一个重要方面是将取出的目标目的代码翻译成为可以由主机处理器 112 执行的仿真指令。

在这方面，更详尽的有关翻译模式的讨论将在下面提供，这包括跳转表 140 的设计和使用。

一个跳转表具有指针项，用于在翻译指令存储器中找到翻译指令集以便为每个不同类型的目标指令产生能由主机执行的仿真指令。

在节 3.4.1 中有更详细说明的一个实施例中，跳转表中每个指针项都有一个项的长度，它不大于最短的目标指令长度，因此能使跳转表去处理指令长度不均匀的目标指令。

在另一个实施例中，其中的目标指令包括目标指令的若干块，块中含有信号处理目标指令，这时跳转表增设一个跳转表影子存储器。跳转表影子存储器有多个跳转表影子表，每个跳转表影子表和具有信号处理目标指令的多个程序块中的一块成为一对。每个跳转表影子表有一个跳转表项的复本，这个项和被包含在一个块中的信号处理目标指令相关联，而这个块则是和跳转表影子表相配对的。当要编译目标指令的一个指定块时，主机处理器就编译对应于指定块的跳转表的一部分和由指定块所产生的信号所调用的任何其它块所需的跳转表影子表。这个实施例在 3.4.3 节中说明。

在另一个实施例中，它在 3.4.4 节中有更详细的说明，跳转表存储

器划分成若干段，它们相应于存储在目标指令存储器中的块。跳转表存储器中被选中的段按照最近被利用程度的判据是不压缩的，而未被选中的段则被压缩。

3.4.1 有可变宽度指令的跳转表

图 15 表示按照本发明的一种模式的在目标目的代码区 132、跳转表 140、和存放翻译例程的翻译指令区 135 之间的相互关系。如图 15 所指出的，存放在目标目的代码区 132 中的目标指令具有不同的指令长度。例如，目标指令 310 是 4 字节指令，目标指令 312 是 2 字节指令，而目标指令 314 则是 4 字节指令。因此，在所说明的实施例中，最短的目标指令长度是 2 字节。

每一条存放在目标目的代码区 132 中的目标指令必须翻译成为称作仿真指令的指令，它们在主机处理器 112 上是可执行的，以便主机处理器 112 能够实现如目标处理器在执行目标指令时相同的作用。为此，翻译例程 135 对于每种类型的目标指令都包含一组翻译指令以产生仿真指令。例如，翻译例程 135 含有一组翻译指令 320，它们适合用来按照目标指令 310 所属的目标指令类型来建立一组仿真指令 330。作为另一个例子，翻译例程 135 含有一组翻译指令 322，它们适合用来按照目标指令 312 所属的目标指令类型来建立一组仿真指令 332。

在指令翻译过程中，有必要在翻译指令区 135 中找到适合于建立仿真指令以便存放在仿真指令存储区 142 中的特定的翻译指令组(例如 320 或 322 组）。为此就要使用跳转表 140。如图 15 所示，跳转表 140 有多个时间片或项 340（1）到 340（n）。跳转表 140 中的每一项 340 的长度为 SL，它不比最短的指令长度更长，也就是说，在本实施例中为 2 字节。

每个在目标目的代码区 132 中的目标指令被解码例程 134 取出并解码后就被赋以一个指针，该指针指向在区 135 中的翻译指令组，它可以用来建立相应的仿真指令组。例如，项 340（3）含有对于目标指令 310 的指针，该项 340（3）的指针访问在区 135 中的翻译指令组 320 的地址。与此相似，项 340（5）含有对于目标指令 312 的指针，项 340（5）的指针访问区 135 中的翻译指令组 332 的地址。每个指针都存放在跳转表 140 中的一个独特的位置。实际上，跳转表 140 和目标目的代码区 132 是这样设计和编址的，它使目标目的代码区 132 中目标指令的地址可以

用来获得在跳转表 140 中的相应指针的地址。具体地说，跟随在目标目的代码区 132 中的目标指令之后的字节地址当被在跳转表 140 中各项的标准长度 SL 相除之后，就产生在跳转表 140 中的相应指针的项号。例如，在所说明的实施例中，目标指令 312（它结束于区 132 的地址 10 上）的指针是存放在跳转表 140 的项 10/SL=10/2=5 上。

这样，图 15 的实施例的跳转表 140 的结构使得利用可变的、也就是非均匀长度的目标指令变得容易，但仍然保持一个有效的变址方式来确定目标指令在跳转表 140 中的指针项。这是通过给跳转表 140 提供一个与最短的目标指令的长度是同长度的分隔度而完成的。但是，跳转表 140 中的每个项仍要求足够长以使它能够包括它所指向的在区 135 中的翻译指令组的地址。因此，跳转表 140 必须具有该目标程序两倍的长度，这个目标程序的目标指令是存放在目标目的代码区 132 中的。对于某些应用来说，跳转表 140 的大小将因此而大得无法接受，从而要求有一个或更多的方法来减少跳转表 140 的大小，这将在下面叙述。

3.4.2 混合执行模型

图 19A 说明了本发明的一个实施例的混合模型的各方面。图 19a 的模型合并了公共执行线程模型的优点和在线模型的优点。具体地说，混合模型实现了跳转目的地址的快速计算和有效地处理从目标到主机代码的程序计数器计算，而这些是纯线程模型所提供的。混合模型也达到了纯在线模型所提供的速度上的优势。

图 19a 的混合系统是在目标目的模块 900 通过跳转表 910 和仿真代码区 912 相互作用下实现这些优点的，这个跳转表 910 含有相关的跳转表指针 910（1）到 910（n）。目标处理器指令是动态地“在空中”翻译成为仿真代码的。但是翻译是按目标指令成块地完成的，这些目标指令块是可连续地执行而无任何跳转的。在这种方式下，主计算机能够按连续流执行代码而不会有不合适的流水线断裂。

然后仿真代码 912 在线地、连续地执行，如图 19a 所示。这就是说，仿真代码能够在没有仿真代码 912（1）到 912（n）的各块之间通常的连接指令的情况下执行。这保证了主计算机能够快速地执行仿真代码，除了在例如地址翻译、中断或断点这样的情况下，不会打断可执行仿真指令的流水线。

如图 19b 所示，连续的主机代码仿真块断裂成为在线的块以便节约

连接指令。在图 19b 的举例的实施例中，每个编译块的边界可以通过查验流动控制指令来确定。当执行到达一个未经编译的目标指令时（这一般发生在进行分支时）就开始递增编译。该块被编译一直到达到一个分支并在该块内部的一个分支处停止。这种产生方式可得到一个在分支的目的地址上开始和结束的仿真代码块。

编译图 19b 中的选定块的示例性规则为:

1. 在分支目的地处的第一条指令使一个块开始，除非该指令已经编译。
2. 该块继续往前直到碰到第一个分支指令，但以下情况除外:
   2.1 如果分支是往回走到当前块的，则该块并不停止而继续编译。
   2.2 局部的（例如小于一个页面的大小）的向前分支并不使该块停止，除非在向前的分支和向前分支的目的地址之间有另一个分支而它又不适用于规则 2.1 和 2.2。
3. 在条件分支之后的第一条指令也是一个块的开始。

这些规则可以简化成下面两条而不会在实践中损失效率:

1. 分支目的地址的第一条指令是一个块的开始，除非该指令已被编译。
2. 该块继续进行直到第一个非条件性分支为止。

例如，在图 19b 中，在分支处的第一条指令是可执行块 920 的开始，而第一条非条件性分支指令使其结束。根据这两个分支，块 920 被定义并存放在仿真代码缓冲区 912。下一个块 921 开始于第一分支并延续到指明的“分支出”。请注意，在块 921 中，中间的“分支返回”并不结束块 921，因为这个“分支返回”是往回分支并且也进入当前块中，因此它可以在仿真代码中执行而没有跳转到外面去或流水线中断的情况。块 921 的“分支返回”因而是符合上述规则 2.1 的，因为它并不结束块 921。

图 19b 的块 922 按照上述的规则 3 开始于“分支入”而结束于条件“分支出”。在这两个分支之间，块 922 还包括一个短的“向前分支”，由于它的相对的大小它并不结束块 922，如规则 2.2 列出的那样。请注意这个规则会导致对代码 924 的编译，这个代码在向前分支指令之后一直到向前分支返回到指令的那一点之间是不被执行的。同时，“条件分

支”按照规则 3 结束并开始块 922 和块 923。

3.4.3 跳转表影子

图 3B 的主机系统 410 实现了在它的跳转表 440 中只用少量的项。图 3B 的主机系统 410 与上面参照图 3A 而说明的主机系统 110 相似，只有在这里所指出的那些不同之处，例如，以不同符号表示的组成元件。主机系统 410 和主机系统 110 之间一个主要的差别是在主机系统 410 的 RAM 130 中提供一个跳转表影子区 444。跳转表 440 和跳转表影子区 444 的使用将参考图 16 来说明和解释。

图 3B 的主机系统 410 特别适合于分组成为若干块的目标指令，其中至少有某些块含有信号处理目标指令。例如，为了简化起见图 16 的目标目的代码区 130 被表示成在其中存储着一个块 510A。组成块 510A 的是各种不同的目标指令，其中包括：用于处理第一信号的目标指令的组 512（1）；用于处理第二信号的目标指令的组 512（2）；以及用于处理最后一个信号的目标指令的组 512（n）。如前面所解释的，一个“信号”是从代码的一个块送到另一块的消息。信号包含一个信号头和一个信号体。信号头含有发送块的识别符和接收块，同时还有信号的号码或类型。信号体含有接收块在执行该信号时要用到的数据。这样，在图 16 所示的例子中，块 510A 可以接收从第一到最后一个信号中的任何一个，在这种情况下块 510A 必须被编译和翻译。此外，块 510A 也可以产生要送到别的块去的信号。

如从前面所了解的，在块 510A 中的每个目标指令（包括包含在组 512（1）到 512（n）中的信号处理目标指令）具有存放在跳转表 540 中的相应指针，利用这些指针再去访问在区 135 中的翻译指令组的地址。例如，组 512（1）有 3 条目标指令，对此有 3 个相应的指针存放在跳转表 540 中。这些在跳转表 540 中的指针分别访问在区 135 中的翻译指令组 535－512（1）－1 到 535－512（1）－3。与此相似，作为另一个例子，在跳转表 540 中的用于被包含在组 512（n）中的信号处理目标指令的各个指针分别对在区 135 中的翻译指令组 535－512（n）－1 到 535－512（n）－3 进行寻址。

图 16 还表明，对于块 510A，在跳转表影子区 444 中包含了跳转表影子表 540－510A。跳转表影子表 540－510A 含有用于块 510A 预期要处理的各个信号的项。在这方面，跳转表影子表 540－510A 含有项 540

－510A－12（1）到 540－510A－12（n），它们相当于被包含在块 510A 中的信号处理目标指令的各个组 512（1）到 512（n）。在跳转表影子表 540－510A 中的每一项都是跳转表 440 中一个项的复本，该项对应于信号处理目标指令的对应组中的第一条目标指令。例如，在跳转表影子表 540－510A 中的第一项（它是用于第一信号的项）是跳转表 540 中的一个指针，该指针指向区 135 的地址 535－512（1）－1。与此相似，跳转表影子表 540－510A 中的最后一项（它是最后一个信号的项）是指向区 135 的地址 535－512（n）－1 的跳转表 540 中的指针的复本。

图 17 表示图 16 的实施例，但具有多个块 510A、510B…510J，每一个都有其对应的跳转表影子表 540－510A、540－510B、……540—510J。这样，具体参考前面的讨论，可以理解，对于每个块 510A、510B、…510J 其跳转表影子表含有由相应的块所处理的各信号的项。在这方面，对于任何给定的块，它的跳转表影子表的项的数量可以从该块的信号分布表来确定。

3.3.4 跳转表分段和时间印记压缩

图 17 还表示翻译目标程序块用的代码的编译过程。从图 17 可以理解，主机处理器 112 没有必要去访问整个跳转表 440，而是只要访问跳转表 440 中要翻译那一块所需的那部分，再加上由被翻译的块所产生的信号相关的其它每一块所用的各跳转表影子表。这样，在图 17 的例子中，其中块 510A 为各块 510B－510N 产生多个信号，而主机处理器 112 只需要访问在图 17 中由线 600 框起来的存储区。

由于主机处理器 112 没有必要访问图 3B 中的整个跳转表 440，因此可以理解，跳转表 440 的其余部分可以存放在 RAM 130 以外的任何地方。例如，跳转表 440 的剩下部分可以存放在硬盘中或其类似部件中，例如由 I/O 存储设备 120 或所用的其它高速缓存技术所处理。

图 3C 说明了另一种节省存储器的主要系统 710 的实施例，具体地是利用对跳转表 740 的数据进行压缩的实施例。图 3C 的主机系统 710 和上面参照图 3A 所说明的主机系统相似，仅仅在这里另外说明的地方有不同，例如，标明不同参考符号的组成元件。主机系统 710 和主机系统 110 之间的一个主要差别是主机系统 710 的 RAM 130 提供了一个时间印记存储区 746。对时间印记存储器 746 结合跳转表 740 中的数据压缩的利用将参考图 18 来说明和解释。

在图 18 中，目标目的代码区 732 有多个块从 710A 到 710N。和以前的实施例一样，每一块的每条目标指令在跳转表 740 中有一个相应的指针。虽然在图 18 中没有说明，但应理解，跳转表 740 中的各指针都指向翻译指令组，其方法和前面结合其它各说明的实施例所说明的相同。

如图 18 所示，块 710A 在跳转表 740 中有它的指针组 740－10A，块 710B 在跳转表 740 中有它的指针组 740－10B，依此类推一直继续到块 710N。跳转表 740 中用虚线表示的各指针组是压缩的，也就是说，具有压缩的数据，而跳转表 740 中用实线表示的各指针组是未经压缩的。

在图 18 中还进一步表明，在跳转表 740 中的每一组指针 740－10A 到 740－10N 具有存放在时间印记存储区 746 中的时间印记。例如，组 740－10A 具有时间印记 746－10A，组 740－10B 具有时间印记 746－10B，依此类推。对于未压缩的组，每个时间印记指明在跳转表 740 中的该组指针未经压缩的相对时间。例如，时间印记 746－10B 指明组 740－10B 在时间 1552 是未压缩的。时间印记 746－10N 指明组 740－10N 在时间 1554 是未压缩的。

这样，在图 3C 的实施例中，跳转表 740 是分裂成段的，目标程序的每一块各有一段。这些段分别地被压缩或未经压缩。当对一个块编译代码时，主机处理器 710 打开跳转表 740 中的相关的段，就可以得到标准的直接的查找方式以便计算局部的分支目的地址。当执行线程离开当前块时，主机处理器 710 重新压缩在跳转表 740 中的相关的段。

更为可取的是，主机处理器 710 只允许预定数量（例如 20 到 50）的跳转表 740 中的段可以同时不加压缩。如上所述，这些未压缩的段是加时间印记的。跳转表 740 中最早使用过的段是可以由时间印记存储区 746 来确定的，在出现需要对新的段解压缩时，它们可以被压缩。最好是在大多数时间内只有少量的块是活动的，使得图 3C 的实施例在这种情况下可以在时间和存储器需求方面有一个良好的折衷。

3.5 指令转换

另外一个方面，它是可供选用的但最好是包括在本发明的现有实施例中去的是一个递增编译程序，用作为从目标代码到仿真代码的代码翻译过程中的一个中间步骤。在目标代码和仿真代码之间的功能指针最初可以由对目标指令的解码和仿真指令的产生等功能的访问来定义。在这

种方式下，对目标计算机系统的改变可以通过更新在翻译步骤中要查阅的表格索引而包含在翻译过程中。

这就是说，在翻译时不是简单地对目标指令解码并产生相应的仿真代码，图 20 的实施例的例子利用了翻译步骤 1003，这一步骤从产生表 1002 摘出它的目标程序并在实际仿真代码产生之前和之后分别实施“前言”和“后记”例程。这些程序在下面将更详细说明。

图 20 的例子表示被解码和再生成为仿真代码的一条加寄存器指令。代码转换过程从读出目标指令 1001 开始。在这个情况下，目标指令是一条在地址 H'5F 的从“r1”到“r2”的加寄存器指令。这条指令装入到地址为 1004 的产生表 1002 中，从而更新产生表 1002 以便在仿真代码产生例程中容纳新的指令步骤。然后翻译步骤 1003 从 1004 摘出该改变指令，执行 1)前言例程，2)仿真代码产生例程，和 3)后记例程。

对一个简单的 AR 指令的解码和代码生成是用下列功能（参阅 3.8.8 节）完成的:

```
GenAR()
{
      register u_char r1 = (instr[0] >> 16) & 0x3f;
      register u_char r2 = (instr[0] >>  8) & 0x3f;

      PROLOGUE(5); /* At most 5 host instructions */
      OUT_READHALF_2_REGS(r1,op1,r2,op2);
      OUT_ADD(op1,op2,op1);
      OUT_SRL(op1,16,regRIR_N);
      OUT_WRITEHALF_REG(op1,r1);
      EPILOGUE(2); /* simulates 2 target cycles*/
```

图 20 是将一个特定目标指令，即寄存器加寄存器指令（AR r1-r2）转换成主机指令的一个过程的示意图。

OUT－宏指令进行实际的代码生成，也就是构成相当于该指令的比特图形并将该图形存入存储器。对于每一组固定宽度的指令，都为 SUN SPARC 存在一个相应的 OUT－宏指令，它存在于叫做 Spare_code.h 的文件中。上面所用的 OUT_ADD 产生一条 SPARC 加法指令的二进制表示。它被定义如下:

```
#define OUT_ADD(RS1,RS2,RD) \
  GEN(0x80000000 | ((RD)<<25)| ((RS1)<<14) | (RS2))
```

OUT_宏指令使用另一条宏指令，它叫做 GEN，其定义如下：

```
#define GEN(instruction) \
    *emulation_code_top++ = (instruction)
```

这条宏指令把指令比特图形存入主机代码缓冲区（在 emulation_code_top）并更新 emulation_code_top 指针。主机代码缓冲区是否溢出将在前言中予以校验（见 3.5.1 节）。

3.5.1 前言

在产生任何代码之前作为转换指令的第一个动作是调用 PROLOGUE（前言）功能。前言可保证在主机代码缓冲区中有足够的可供使用的空间，更新跳转表指针并可用于打印跟踪信息。

```
void PROLOGUE(int ninst)

{
   if (emulation_code_top+ninst+EPILOGUE_MAX_SIZE
   >=      emulation_code_base+emulation_code_size)
     grow_emulation_code_buffer();
    *jt_entry++ = (u_long)emulation_code_top;
   if (sim_convert_trace_on) {
      sim_printf("0x%x: ", emulation_code_top);
      print_instruction((u_long)current_logical);
  }
 }
```

jt_entry 指针是指在转换期间使用的跳转表（见 3.4.2 节）。

所示的前言用于单独的跳步方式。

3.5.2 后记

在为一条指令产生代码后就调用后记功能。这个后记处理几件事情。首先，仿真器含有一个事件机制，它用于几种目的，它必须和仿真

时间打交道。例如，如果用户需要仿真器在经过 950 个周期时中断，那么这就是要在延时 950 个周期后通知仿真器发生一次事件。与此相似，通过作业缓冲来处理延迟信号可以使用这种事件机制。后记对每一个仿真周期（对于在目标机上需要一个周期以上的指令，那就是几个周期）更新主机寄存器 regE。当 regE 和（已分类的）事件队列中的时间印记相匹配时，如有需要，在精确方式下此功能可在 3 个主机指令内完成。在快速方式下，事件校验仅仅在产生的代码的几个地方进行，例如在每个编译单元的结束时。

同时，后记还检查在当前执行分片中被仿真的指令的数量，并且如果这个数量超过预定的阈值（例如 1000 或 10000 条目标指令）时，就使控制返回到环绕的主机调度程序。这也可以使用事件机制来完成。

除了上面的事件检查外，标准的后记功能 EPILOGUE 通过在仿真的程序计数器（IAR）上加上刚刚执行过的指令的大小而更新该仿真程序计数器（IAR）。

存在另外两个后记的变型：SIGNAL－EPILOGUE（信号后记）被各种功能用来为信号发送指令产生代码。它利用信号分配表来寻找新的仿真程序计数器（见 3.2.2 节）。JUMP－EPILOGUE（跳转后记）由各种功能用来模拟分支，并且利用在 3.4.2 节中所述的散列表来计算新的仿真程序计数器。SIGNAL_EPILOGUE 和 JUMP_EPILOGUE 两者都用来产生对 update_jam 功能的调用，该功能更新跳转地址存储器缓冲区。

按照还有另外一个本发明实施例的例子，主计算机用于存放仿真代码的存储器是由主计算机在仿真目标指令被写入于其中时分配到合适的存储块中的。此外，主计算机能够模拟存储分配，这种分配将在目标计算机执行目标指令时被使用。

3.6 存储管理和仿真

发生两种存储管理：（1）管理内部仿真器存储器，例如仿真代码用的存储器；和（2）模拟目标存储器，例如数据存储器 DS。

项目（1）必须使用由主机框架提供的存储器调用来管理。目标存储器模拟涉及 DRS 和 PS 的模拟。DRS 和 PS 用的主机存储器是由仿真器在仿真目标指令写进去时分配到合适的存储块中去的。这种写入是易于被检测到的，因为它们使用特定的指令（XWPS、WS、XFTR……）。由于这些存储器访问无论如何都要由仿真器予以确认，所以这样额外的内部

管理预期不会有明显的开销。例如，可以使用的存储块大小为 16K 或 32K。

当仿真器起动时，通过读入一个预先建立的存储转储而使存储器初始化。这使 DS、PS 和 RS 进入严格定义的状态。

要注意，在最低优先级别（DL）上有一个特别的进程，它不断地扫描整个存储区域以检查硬件是否正确地工作，这个进程可能会对存储器模型带来问题，因为整个 DRS 和 PS 会因此而被读出和写入，从而使整个存储器从硬盘交换到存储器，因此使性能变劣。这个优选级别的进程应该找出来并使其失效。

3.7 寄存器模拟

APZ 在 4 个执行级别的每一级上含有 192 个通用寄存器（见 3.2.2 节和 3.2.6 节）。这些寄存器可以在主机存储器中用指向它们的一个全局主机寄存器 regML 来模拟，如图 21 所示。可以看出，对于每一个 APZ 优先级别有一个这样的存储区。当模拟一个级别变化时，regML 寄存器作相似的变化。

此外，跨越所有优先级别而共享的 224 个寄存器是存放在由寄存器 regML 所指向的单独的主机存储器中的。

3.7.1 寄存器的高速缓存

最常用的目标寄存器可以直接由为此目的而分配的主机寄存器来模拟。read_rey、read_2_regs、readhalf_reg、readhalf_2_reys、write_reg、write_2_regs、writehalf_reg、和 write_2_regs 等所提到的伪指令将处理寄存器高速缓存细节的大部分，例如，read_reg 指令将从用来对它进行高速缓存的主机寄存器取得目标寄存器而不是从存储器装入它。某些特殊目的的寄存器总是被高速缓存的，例如，在 regRIR 的 RIR。

4.0 系统优点

如上所述，仿真中央处理器 CP 234 模拟从目标系统的中央处理器 34 一直低到指令和寄存器级。这就是说，仿真中央处理器 CP 234 模拟 CP34 的指令集的执行并使用 CP 34 的全部寄存器。另一方面，处理这样一些杂务如监控和数据传输（例如数据的成组和分组）的其它处理器则不用全指令集仿真（ISS）来模拟，而是由接口模拟来实现。内部总线则是用各不相同的被仿真子系统之间的功能调用来实现。

从前面所述，可以看出本发明的仿真系统 200 有许多优点。某些优点和好处在下面列出。

4.1 生产率的增长

仿真系统 200 向用户提供生产率的增长，这可从下述得到证实:

（1）新的仿真环境解决了目前接口中的问题。

（2）完整的 PLEX 和 ASA（AXE 汇编）系统可以被装入。

（3）几百 MB 大小的系统可以被执行。

4.2 质量的提高

仿真系统 200 为用户提供质量方面的提高，下面是这方面的例子:

(4)目标系统的功能测试中的主要部分可以用仿真系统 200 来完成。

(5)通过较为容易的对测试规范的配置、装入、起动和执行的处理以及查错而改善可用性。

4.3 节省工作周期

仿真系统 200 为用户提供工作周期的节省，下面是其例子:

（6） 仿真系统 200 提供在工作站上的目标环境，缩短了测试中的工作周期。

（7） 仿真中的执行速度需要短得多的周转时间。周转时间包括装入和起动系统、测试的执行和查错。

（8） 不再需要双份检测指令。

（9） 由于在仿真环境中易于测试程序块和纠错的可能性，支持软件用的开销可减少。

（10） 程序纠错在仿真系统 200 中可以容易地测试。

（11） 转储制作易于实现。

（12） 可以在仿真环境中完成对大型系统的配置然后可以制作转储以便在现场使用。

（13） 对测试装置的需求减少。

虽然本发明已经具体地参考了它的优选实施例而被表明和解释，但那些熟悉本技术的人应理解，在它的形式和细节方面可以进行各种不同的变化而不背离本发明的范围和实质。

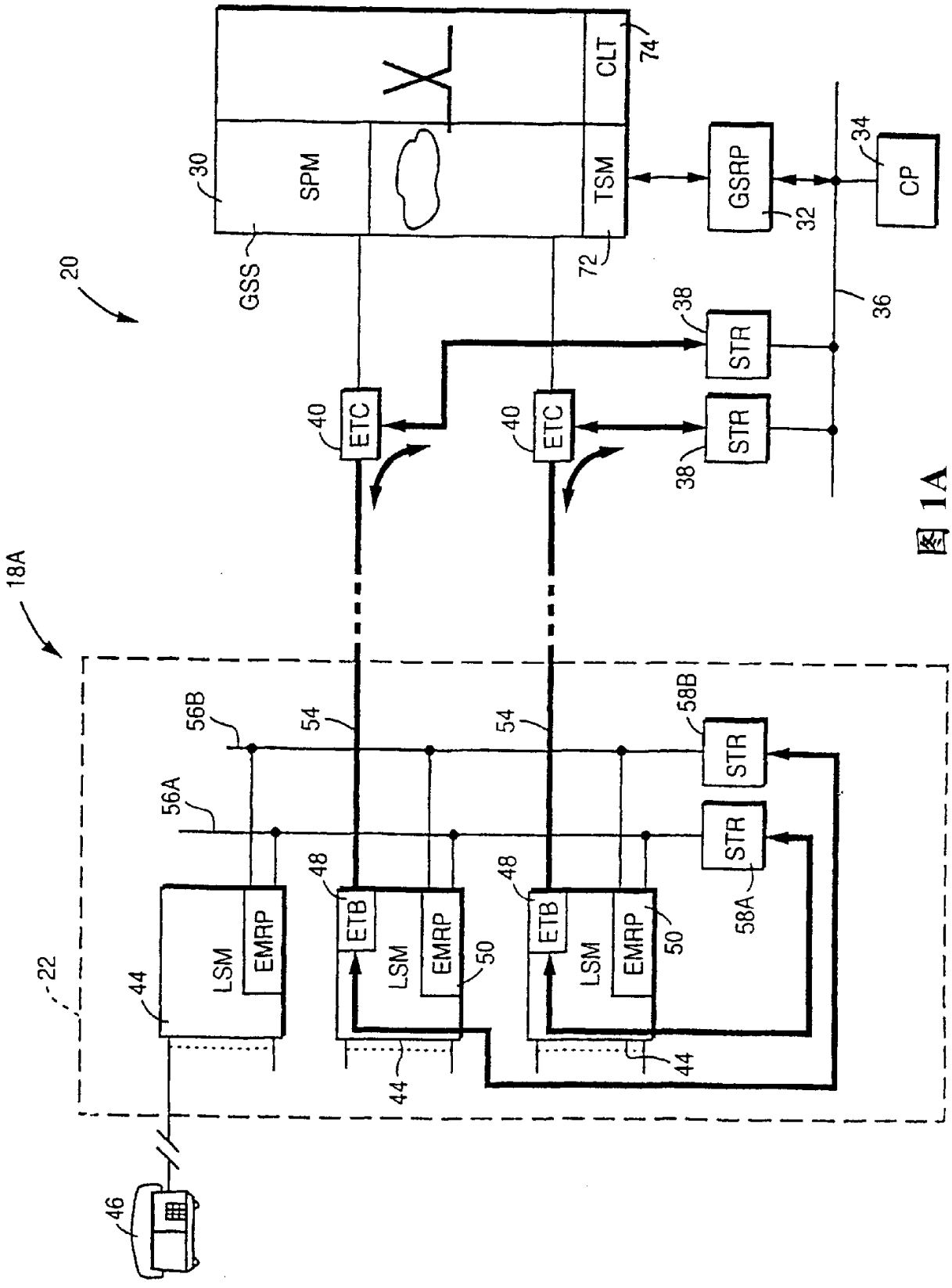

20
GSS
30
SPM
CLT
74
TSM
72
GSRP
32
34
CP
36
38
STR
38
STR
40
ETC
40
ETC
18A
56B
54
54
58B
STR
56A
STR
58A
48
48
ETB
ETB
LSM
EMRP
LSM
EMRP
50
50
22
44
44
44
46

图 1A

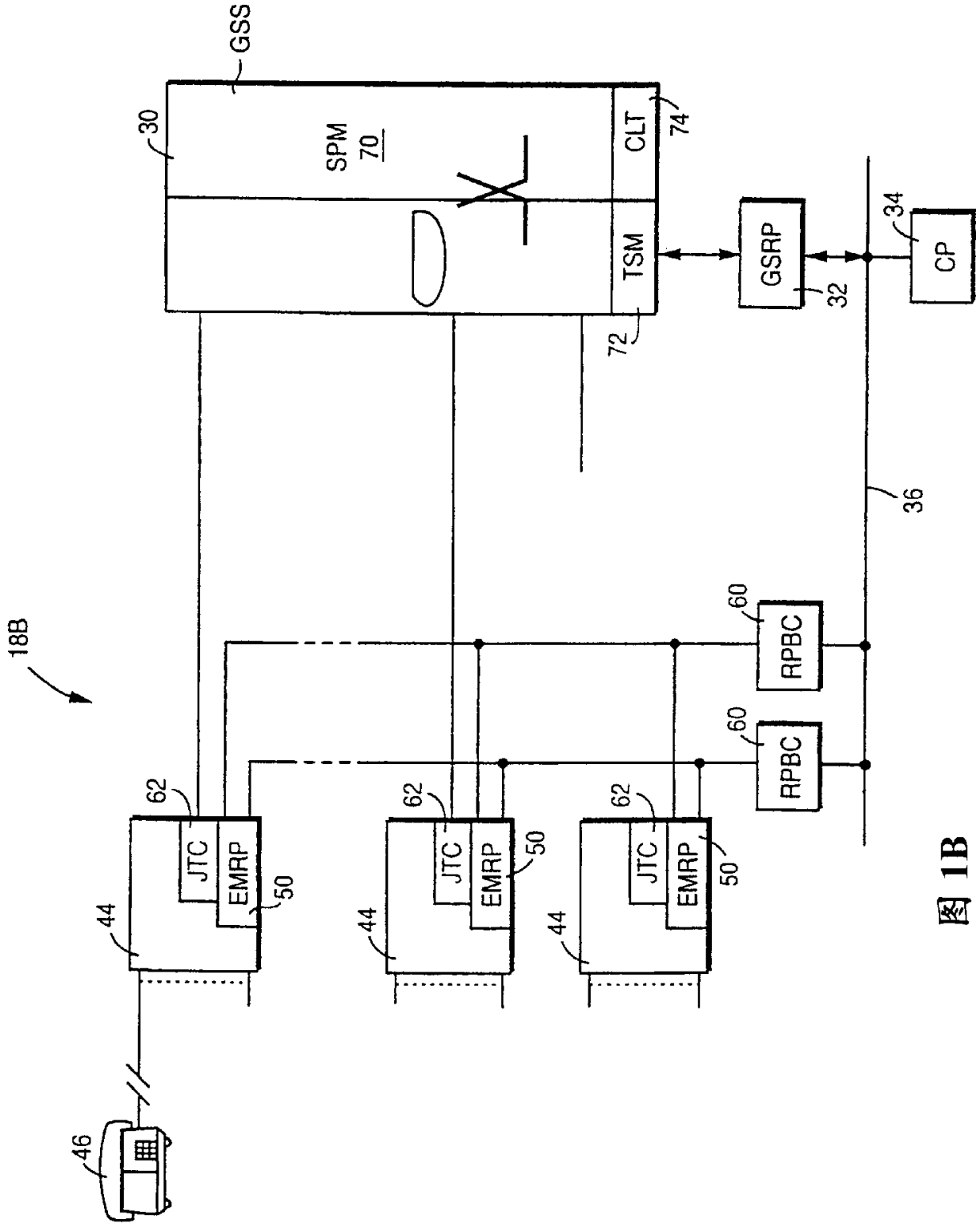

GSS
30
SPM
70
CLT
74
TSM
72
GSRP
32
34
CP
36
18B
60
RPBC
60
RPBC
62
JTC
EMRP
50
44
62
JTC
EMRP
50
44
62
JTC
EMRP
50
44
46

图 1B

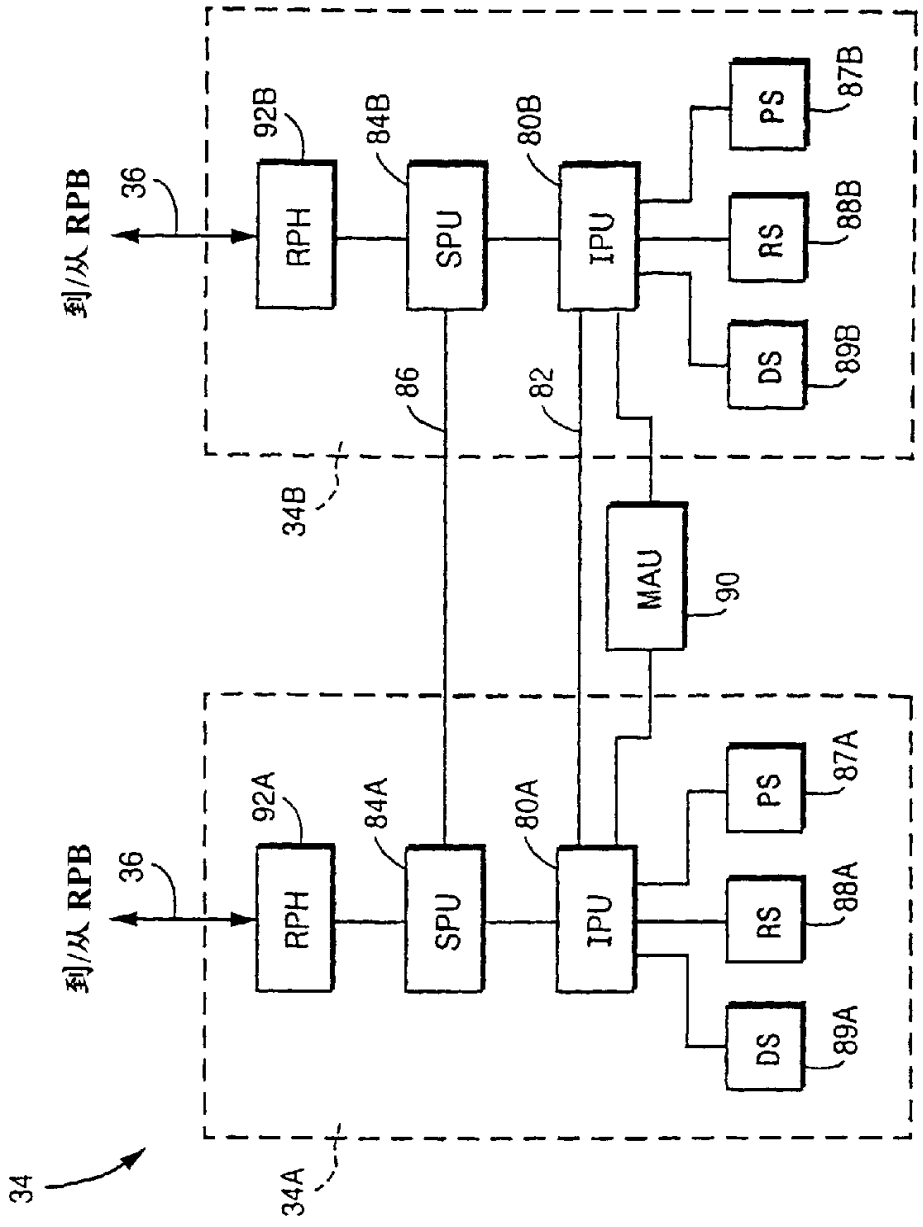

34
34A
34B
到/从 RPB
36
92A
RPH
84A
SPU
80A
IPU
PS
87A
RS
88A
DS
89A
86
82
MAU
90
92B
RPH
84B
SPU
80B
IPU
PS
87B
RS
88B
DS
89B
到/从 RPB
36

图 2

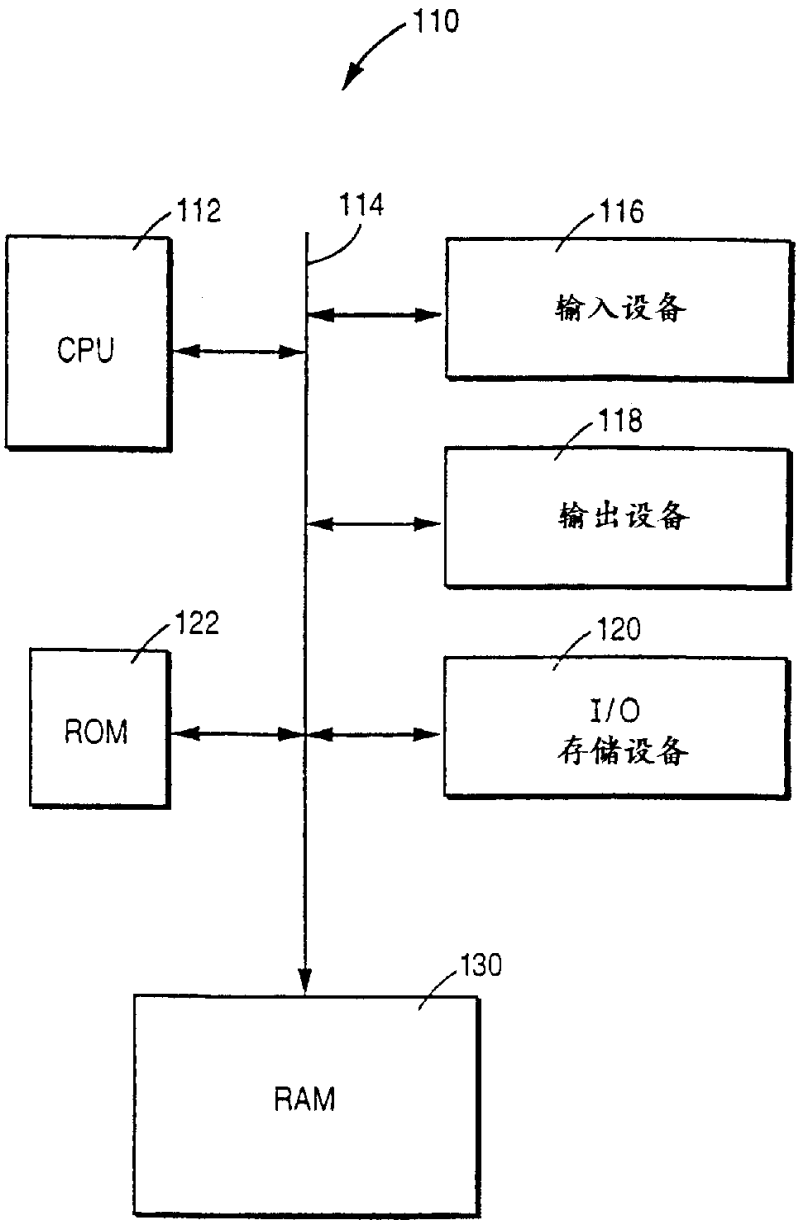


图 3

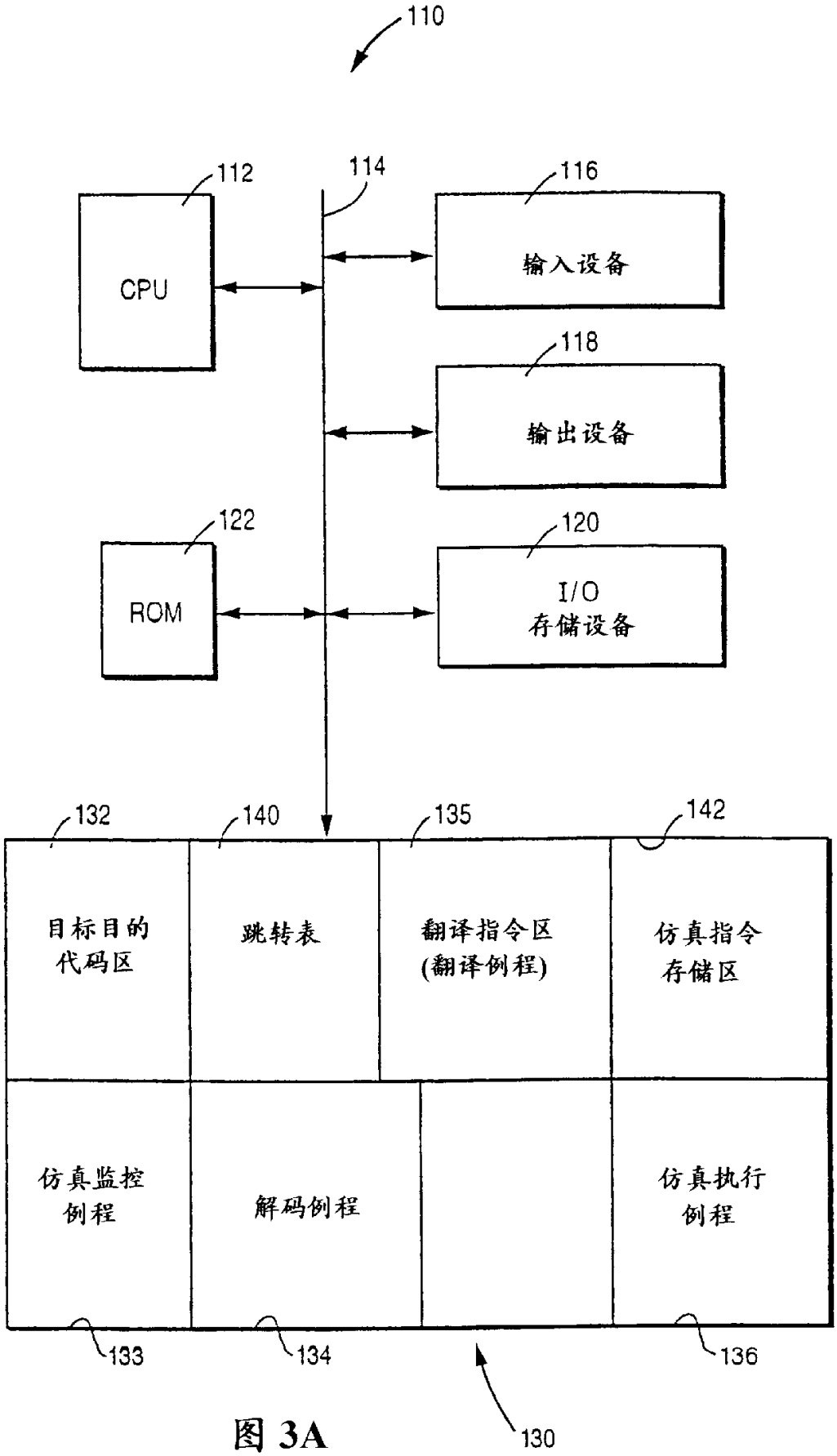

110
112
114
116
CPU
输入设备
118
输出设备
122
120
ROM
I/O
存储设备
132
140
135
142
目标目的
代码区
跳转表
翻译指令区
(翻译例程)
仿真指令
存储区
仿真监控
例程
解码例程
仿真执行
例程
133
134
136
130

图 3A

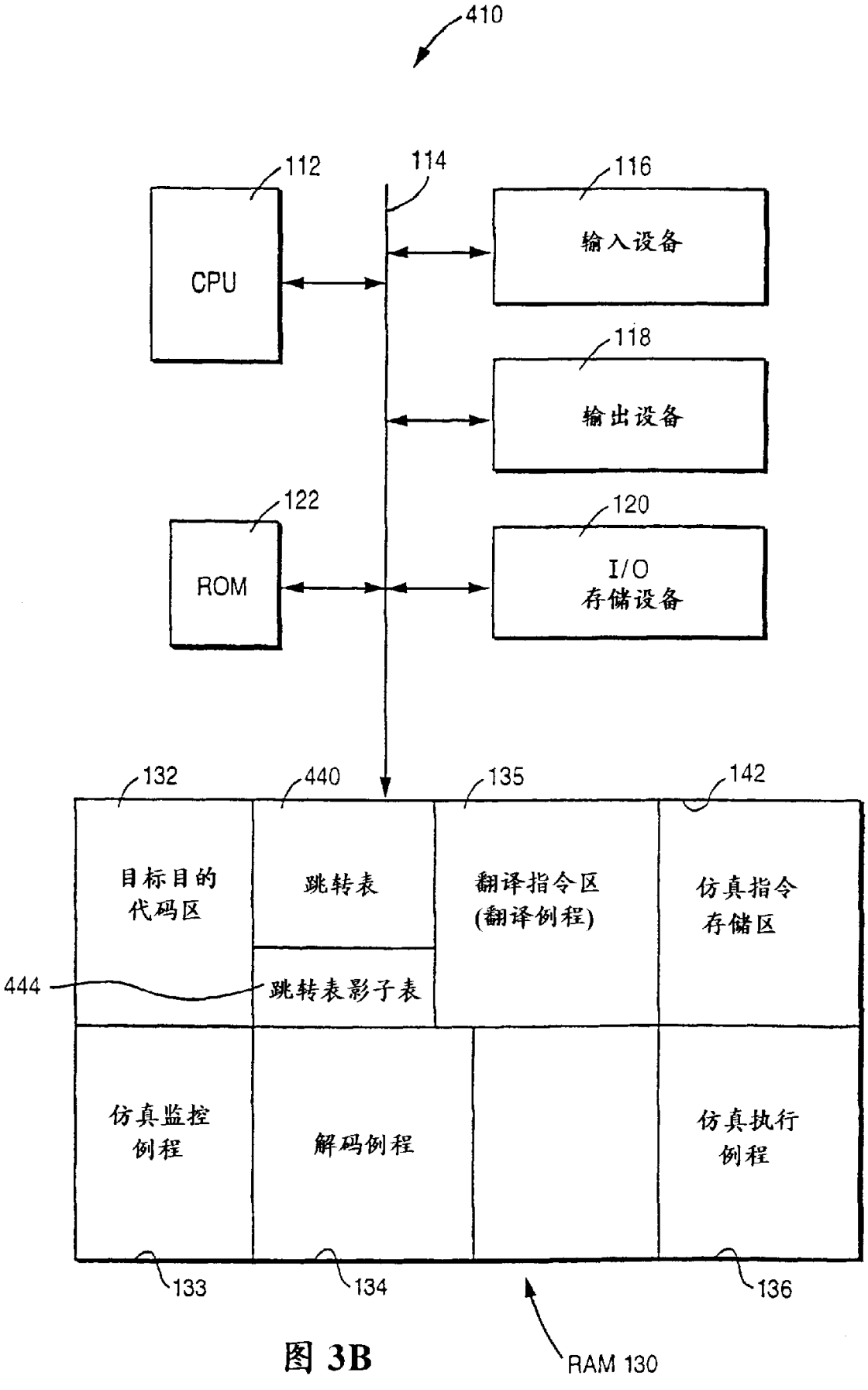

410
112
114
116
CPU
输入设备
118
输出设备
122
120
ROM
I/O
存储设备
132
440
135
142
目标目的
代码区
跳转表
翻译指令区
(翻译例程)
仿真指令
存储区
444
跳转表影子表
仿真监控
例程
解码例程
仿真执行
例程
133
134
136
RAM 130

图 3B

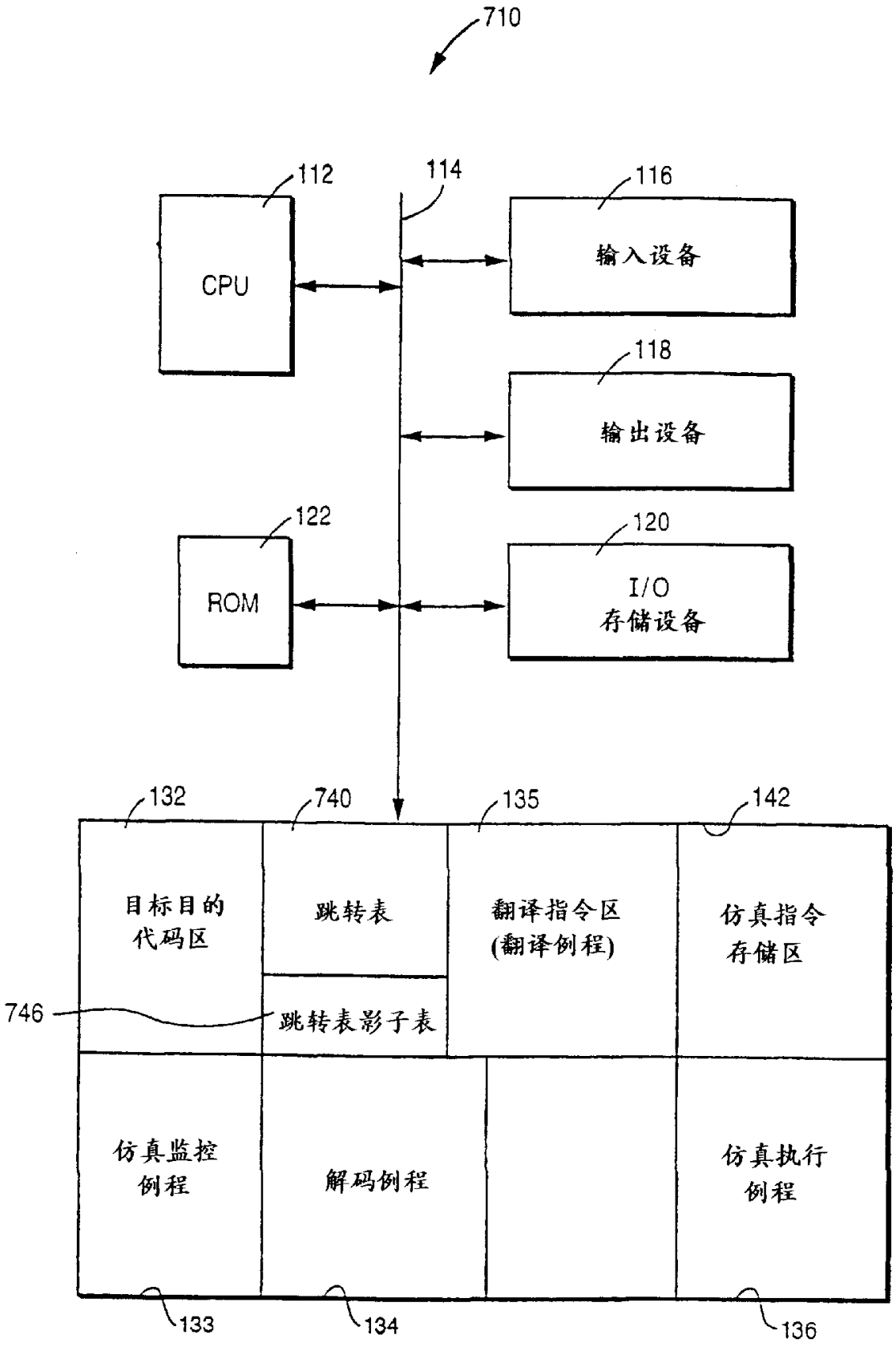

710
112
114
116
CPU
输入设备
118
输出设备
122
120
ROM
I/O
存储设备
132
740
135
142
目标目的
代码区
跳转表
翻译指令区
(翻译例程)
仿真指令
存储区
746
跳转表影子表
仿真监控
例程
解码例程
仿真执行
例程
133
134
136

图 3C

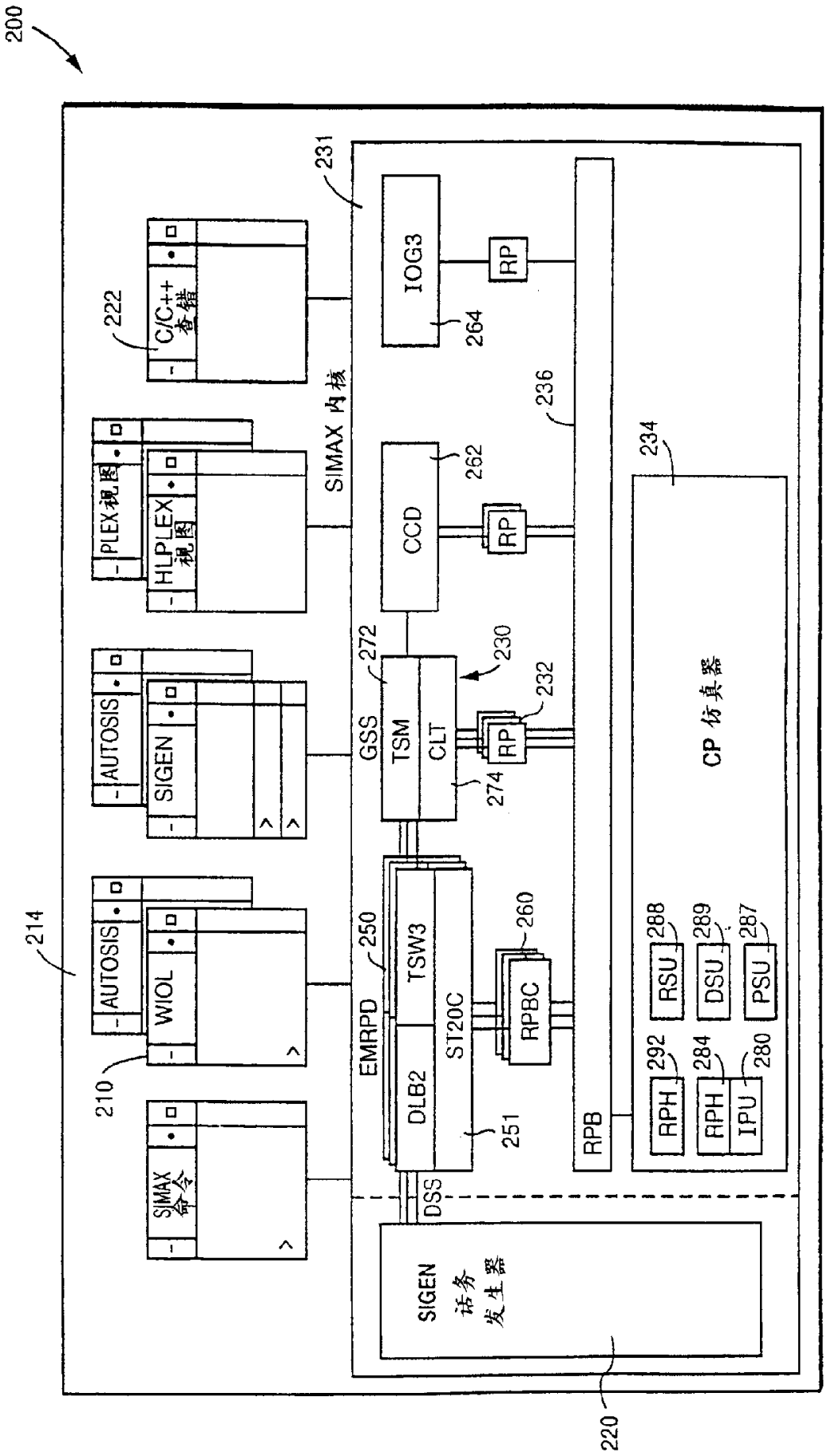

200
214
210
220
231
SIMAX 内核
SIMAX 命令
AUTOSIS
WIOL
AUTOSIS
SIGEN
PLEX视图
HLPLEX 视图
C/C++ 查错
222
EMRPD
250
DLB2
TSW3
ST20C
251
260
RPBC
GSS
272
TSM
CLT
274
230
232
RP
CCD
262
IOG3
264
RP
236
RPB
234
CP 仿真器
RPH
292
RSU
288
RPH
284
IPU
280
DSU
289
PSU
287
DSS
SIGEN
话务
发生器

图 4

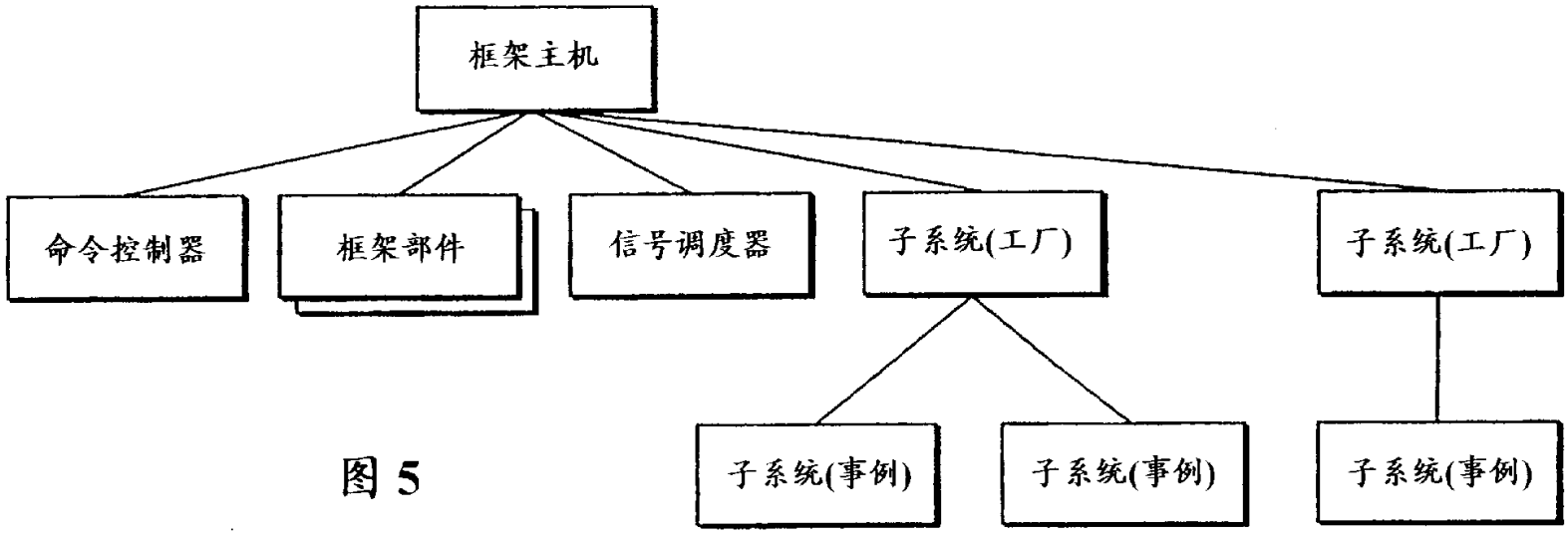

框架主机
命令控制器
框架部件
信号调度器
子系统(工厂)
子系统(工厂)
子系统(事例)
子系统(事例)
子系统(事例)

图 5

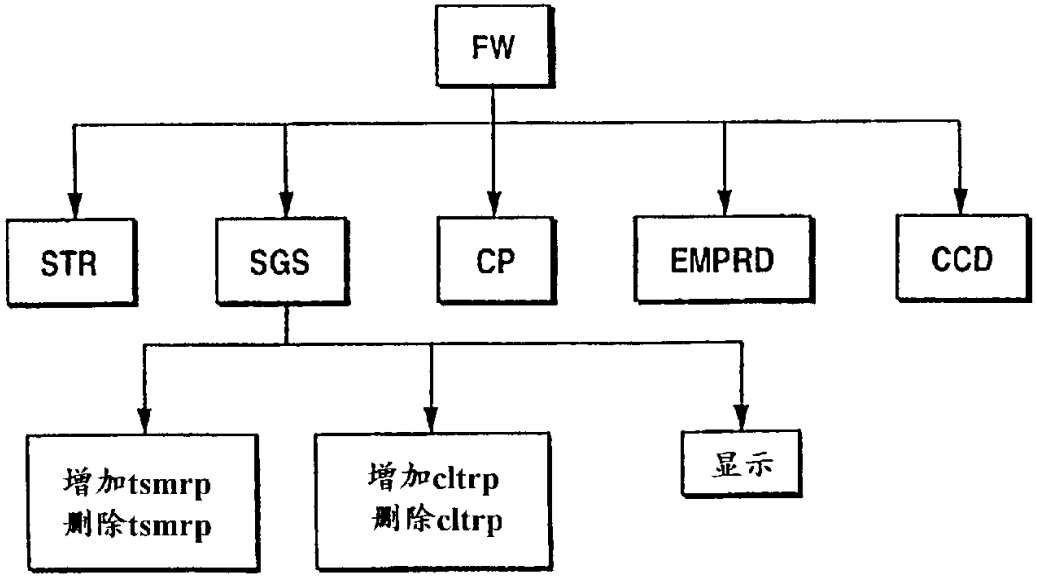

FW
STR
SGS
CP
EMPRD
CCD
增加tsmrp
删除tsmrp
增加cltrp
删除cltrp
显示

图 7C

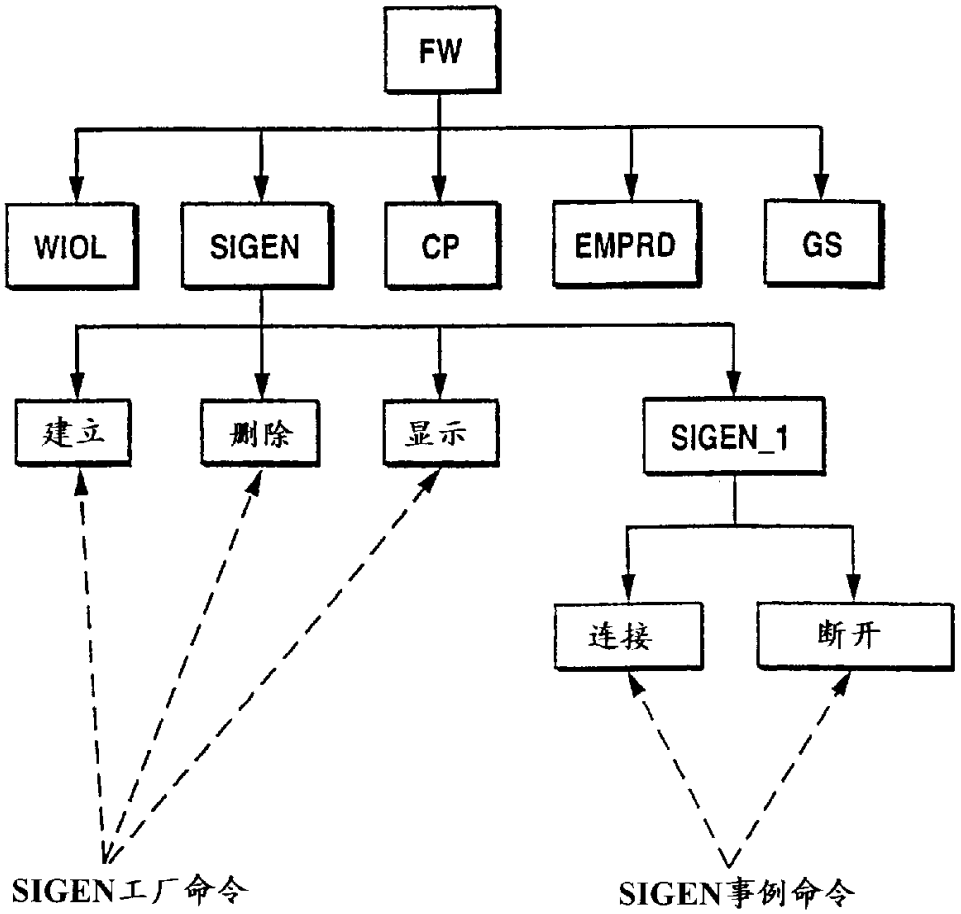

FW
WIOL
SIGEN
CP
EMPRD
GS
建立
删除
显示
SIGEN_1
连接
断开
SIGEN工厂命令
SIGEN事例命令

图 6

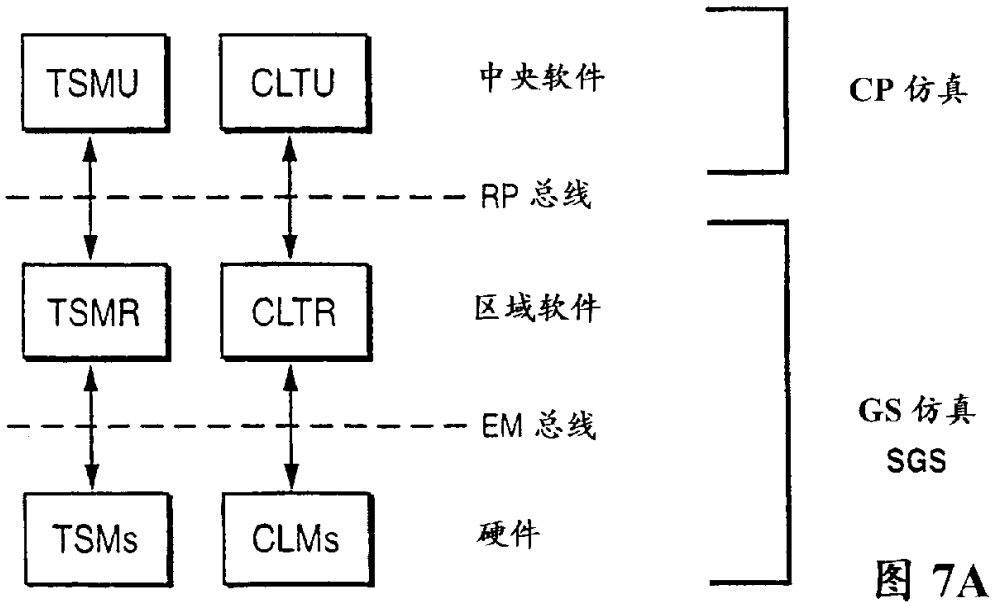

TSMU
CLTU
中央软件
CP 仿真
RP 总线
TSMR
CLTR
区域软件
EM 总线
GS 仿真
SGS
TSMs
CLMs
硬件

图 7A

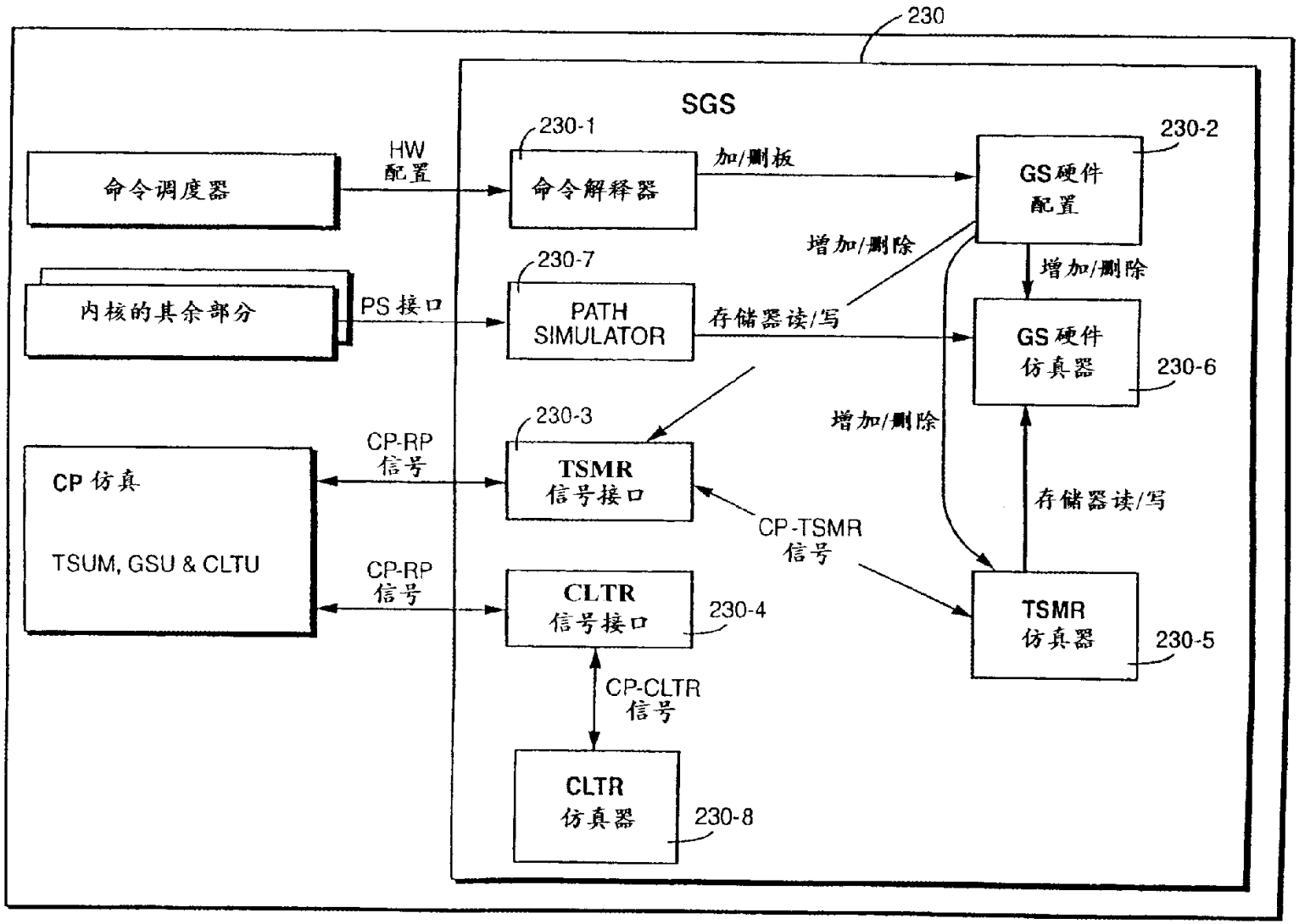

230
SGS
230-1
命令调度器
HW
配置
命令解释器
加/删板
230-2
GS硬件
配置
增加/删除
增加/删除
230-7
内核的其余部分
PS接口
PATH
SIMULATOR
存储器读/写
GS硬件
仿真器
230-6
增加/删除
230-3
CP-RP
信号
CP仿真
TSUM, GSU & CLTU
TSMR
信号接口
CP-TSMR
信号
存储器读/写
CP-RP
信号
CLTR
信号接口
230-4
TSMR
仿真器
230-5
CP-CLTR
信号
CLTR
仿真器
230-8

图 7B

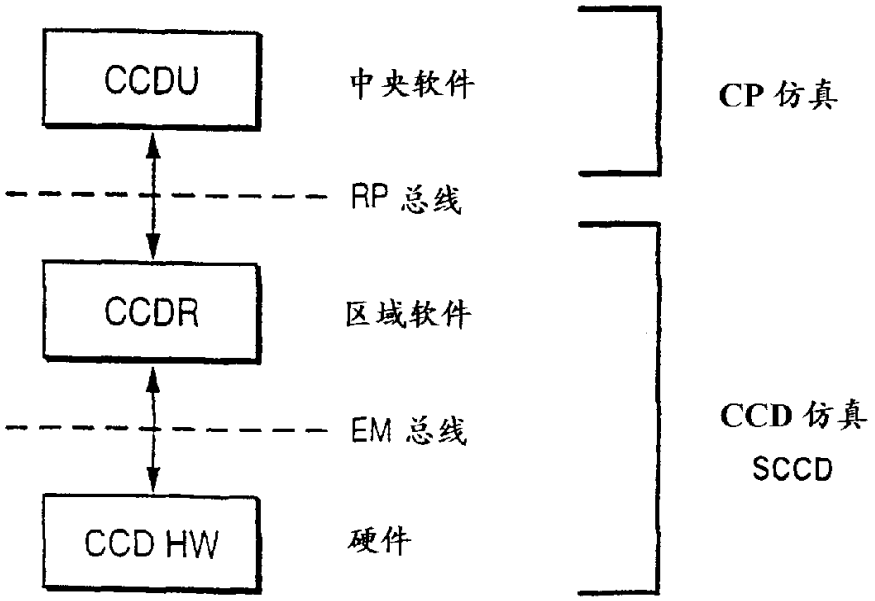

CCDU
中央软件
CP 仿真
RP 总线
CCDR
区域软件
EM 总线
CCD 仿真
SCCD
CCD HW
硬件

图 8A

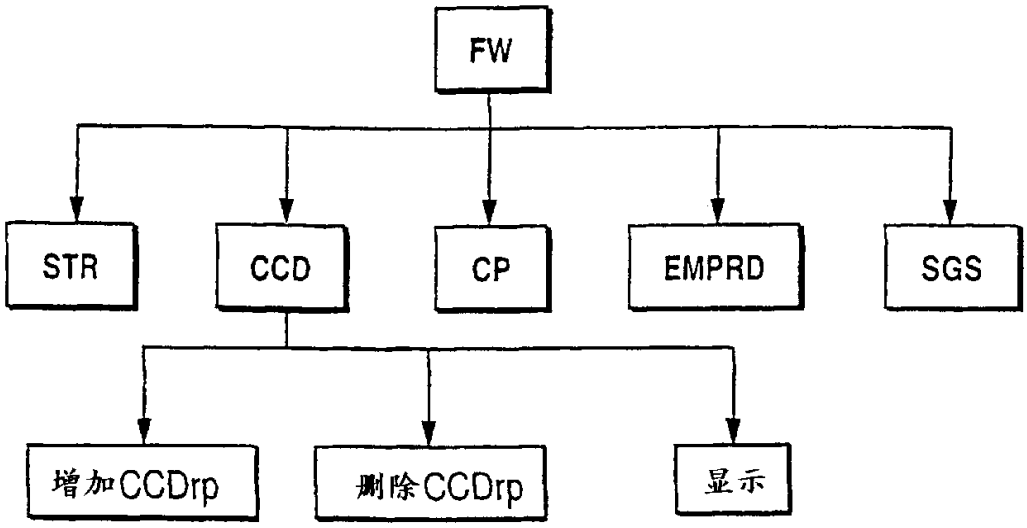

FW
STR
CCD
CP
EMPRD
SGS
增加CCDrp
删除CCDrp
显示

图 8C

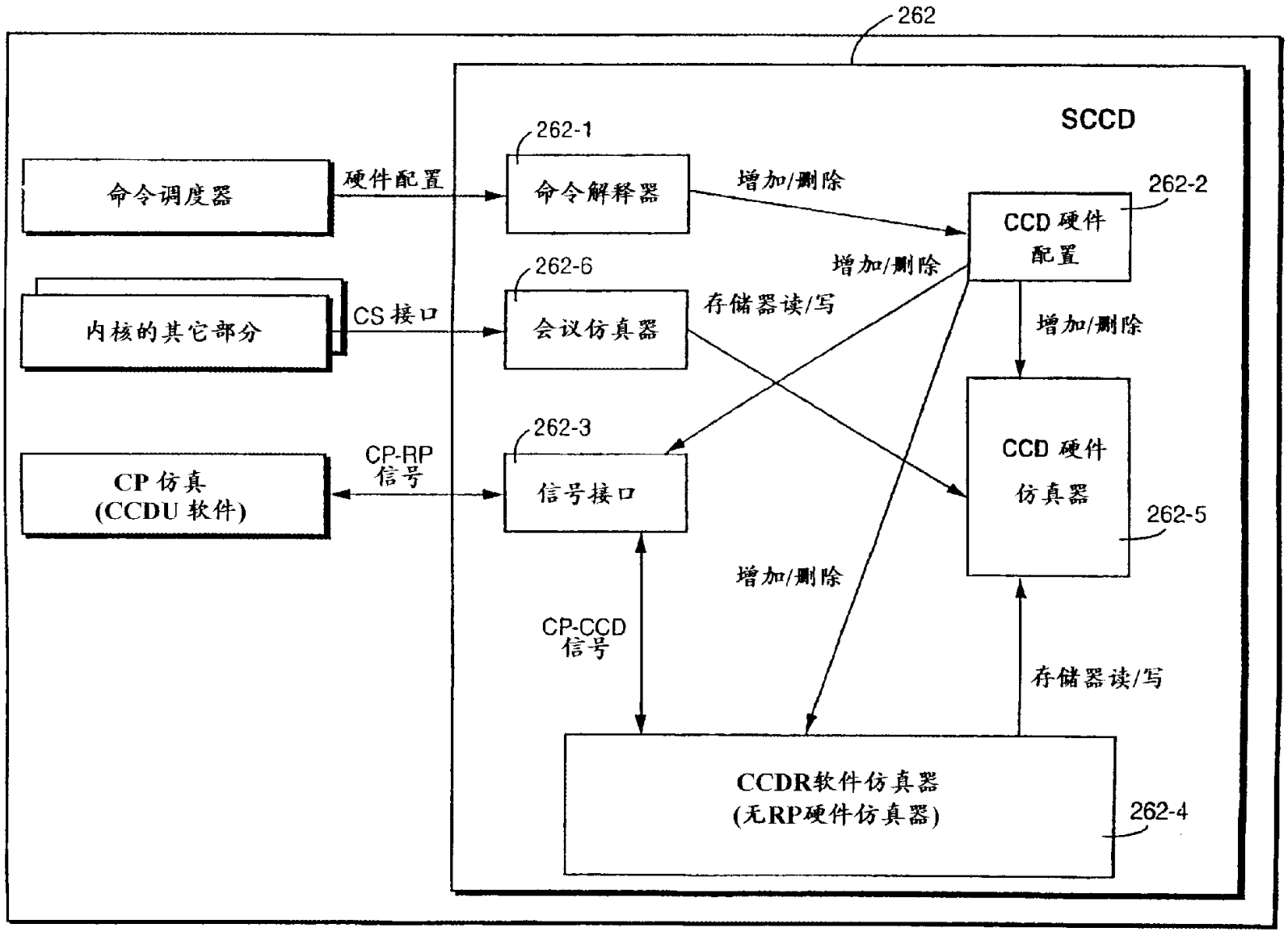

262
SCCD
262-1
命令调度器
硬件配置
命令解释器
增加/删除
262-2
CCD 硬件
配置
增加/删除
262-6
内核的其它部分
CS 接口
会议仿真器
存储器读/写
增加/删除
CCD 硬件
仿真器
262-5
262-3
CP 仿真
(CCDU 软件)
CP-RP
信号
信号接口
增加/删除
CP-CCD
信号
存储器读/写
CCDR软件仿真器
(无RP硬件仿真器)
262-4

图 8B

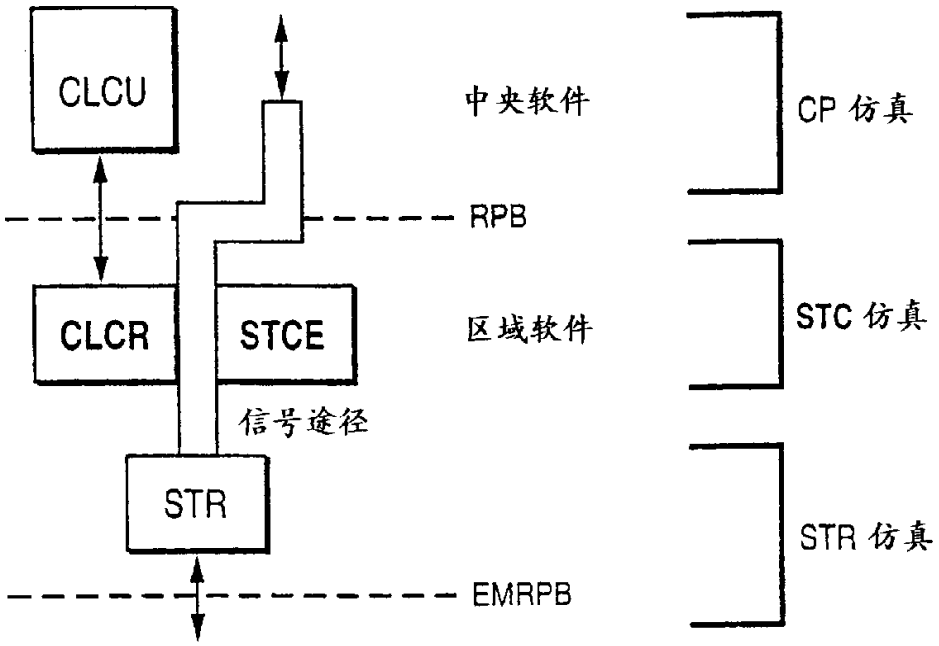

CLCU
中央软件
CP 仿真
RPB
CLCR
STCE
区域软件
STC 仿真
信号途径
STR
STR 仿真
EMRPB

图 9A

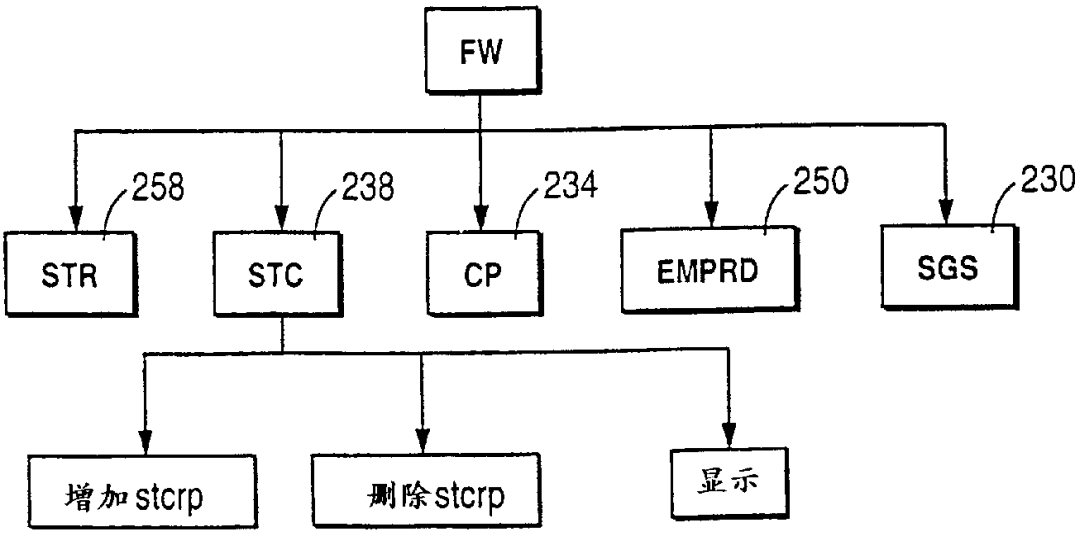

FW
258
238
234
250
230
STR
STC
CP
EMPRD
SGS
增加 stcrp
删除 stcrp
显示

图 9C

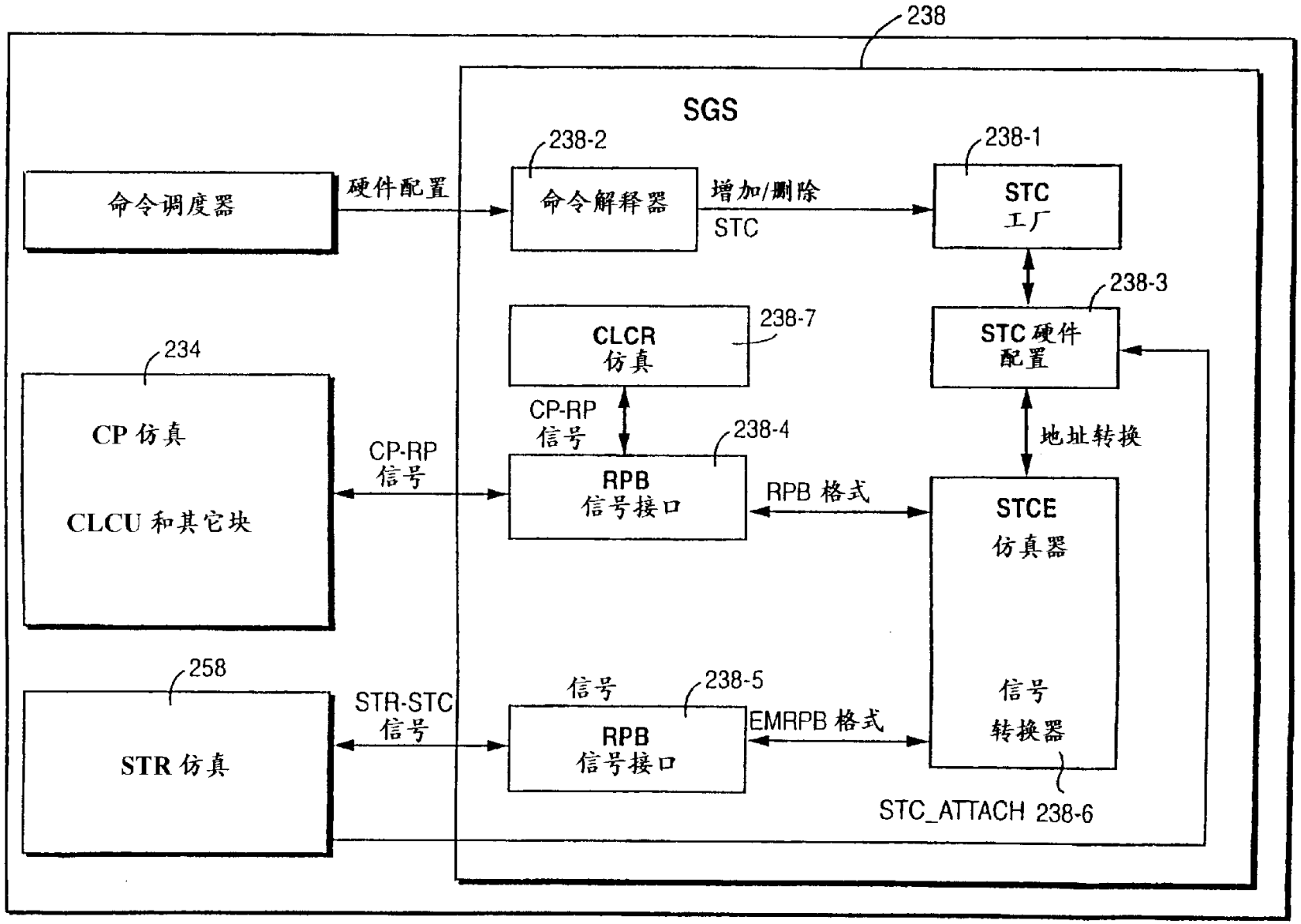

238
SGS
238-2
238-1
命令调度器
硬件配置
命令解释器
增加/删除
STC
STC
工厂
238-3
CLCR
仿真
238-7
STC 硬件
配置
234
CP 仿真
CLCU 和其它块
CP-RP
信号
CP-RP
信号
238-4
地址转换
RPB
信号接口
RPB 格式
STCE
仿真器
258
信号
238-5
STR-STC
信号
STR 仿真
RPB
信号接口
EMRPB 格式
信号
转换器
STC_ATTACH 238-6

图 9B

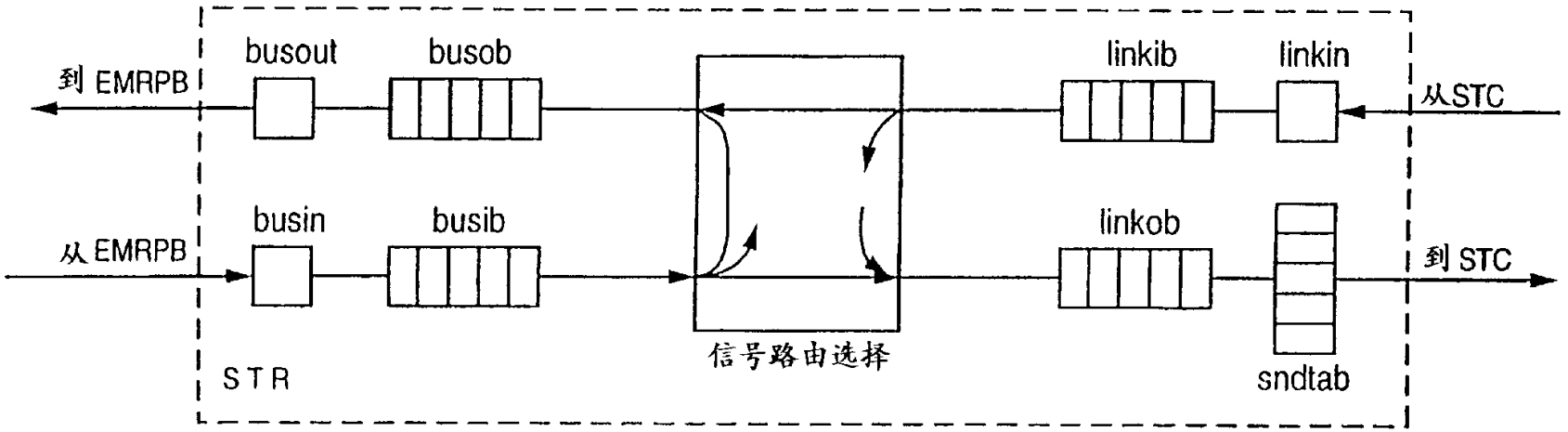

busout
busob
linkib
linkin
到 EMRPB
从STC
busin
busib
linkob
从 EMRPB
到 STC
信号路由选择
STR
sndtab

图 10A

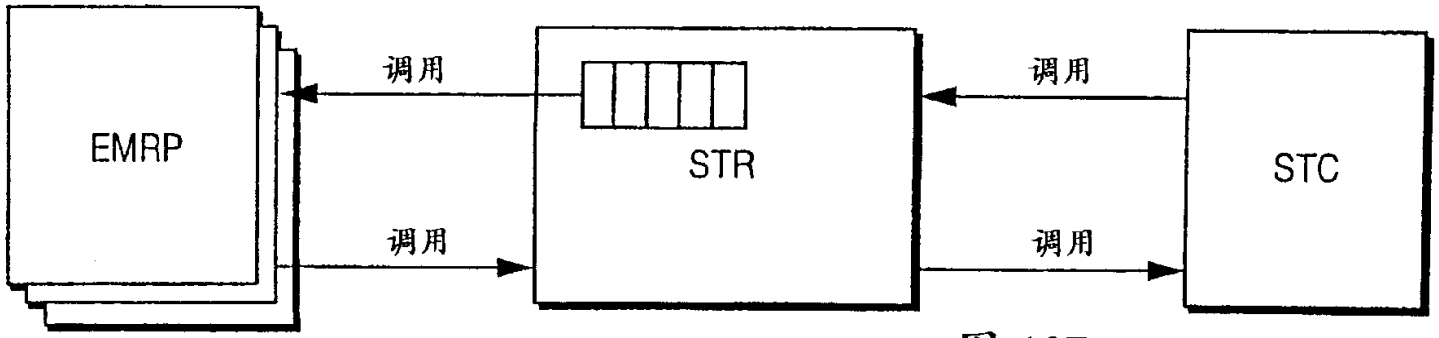

EMRP
调用
STR
调用
调用
调用
STC

图 10B

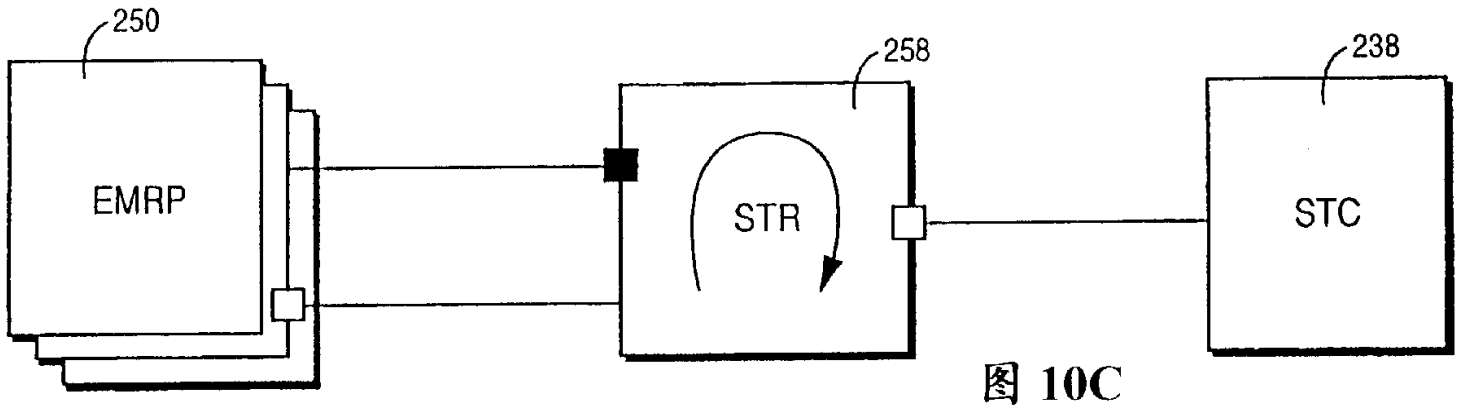

250
EMRP
258
STR
238
STC

图 10C

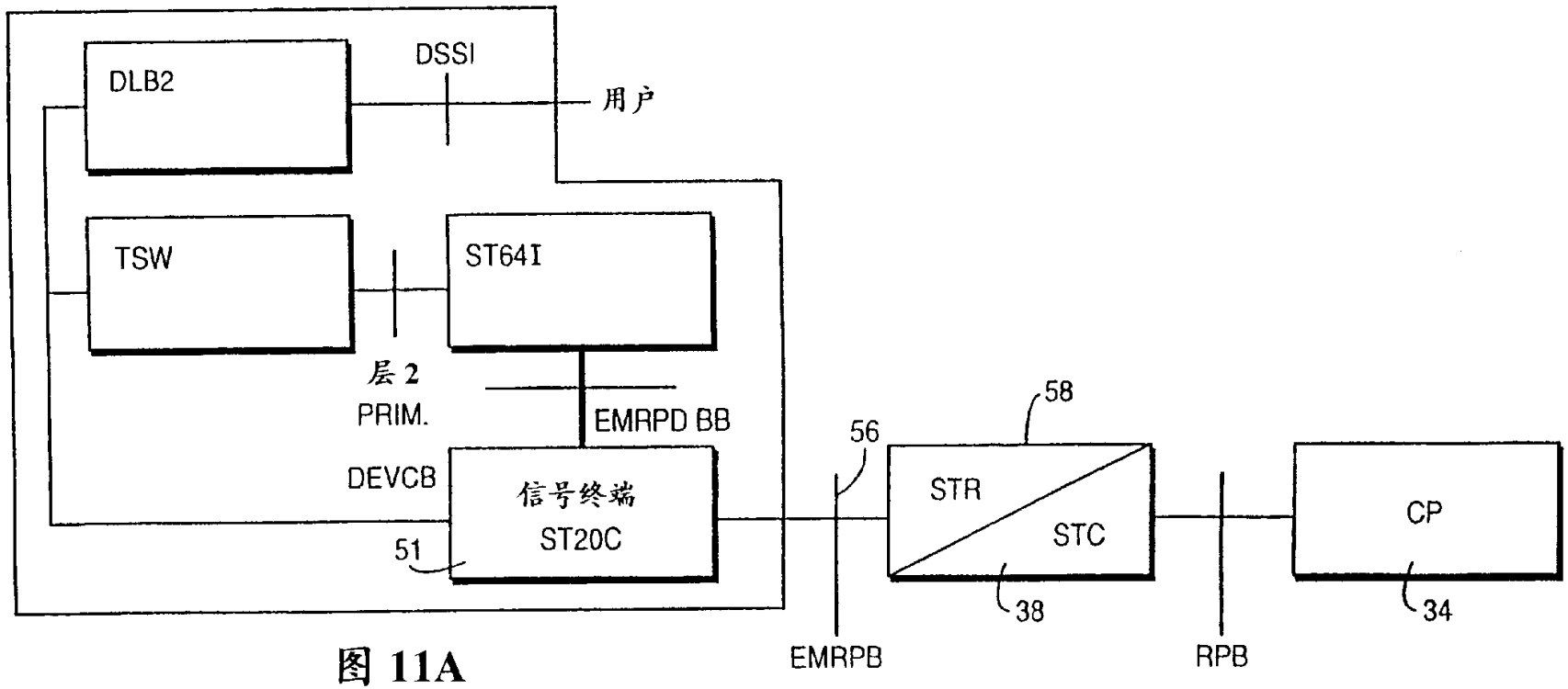

DLB2
DSSI
用户
TSW
ST64I
层 2
PRIM.
EMRPD BB
DEVCB
信号终端
ST20C
51
56
58
STR
STC
38
CP
34
EMRPB
RPB

图 11A

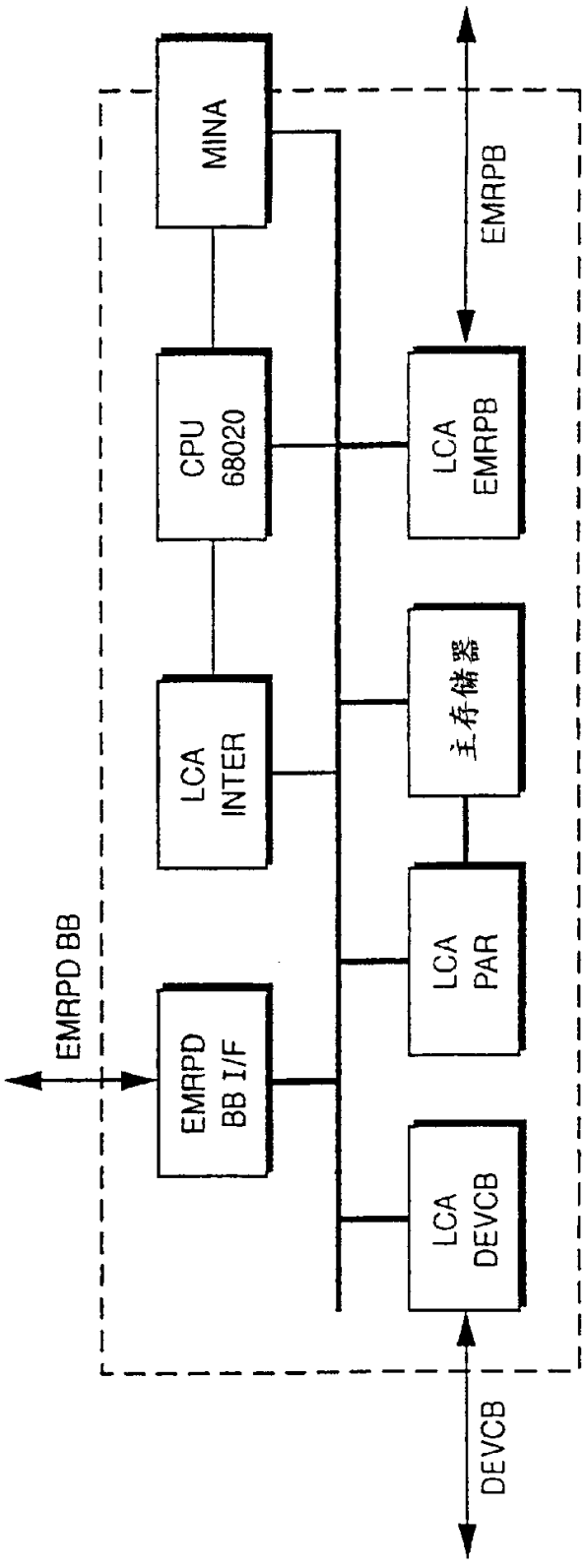

MINA
CPU 68020
LCA INTER
EMRPD BB I/F
LCA EMRPB
主存储器
LCA PAR
LCA DEVCB
EMRPB
EMRPD BB
DEVCB

图 11B

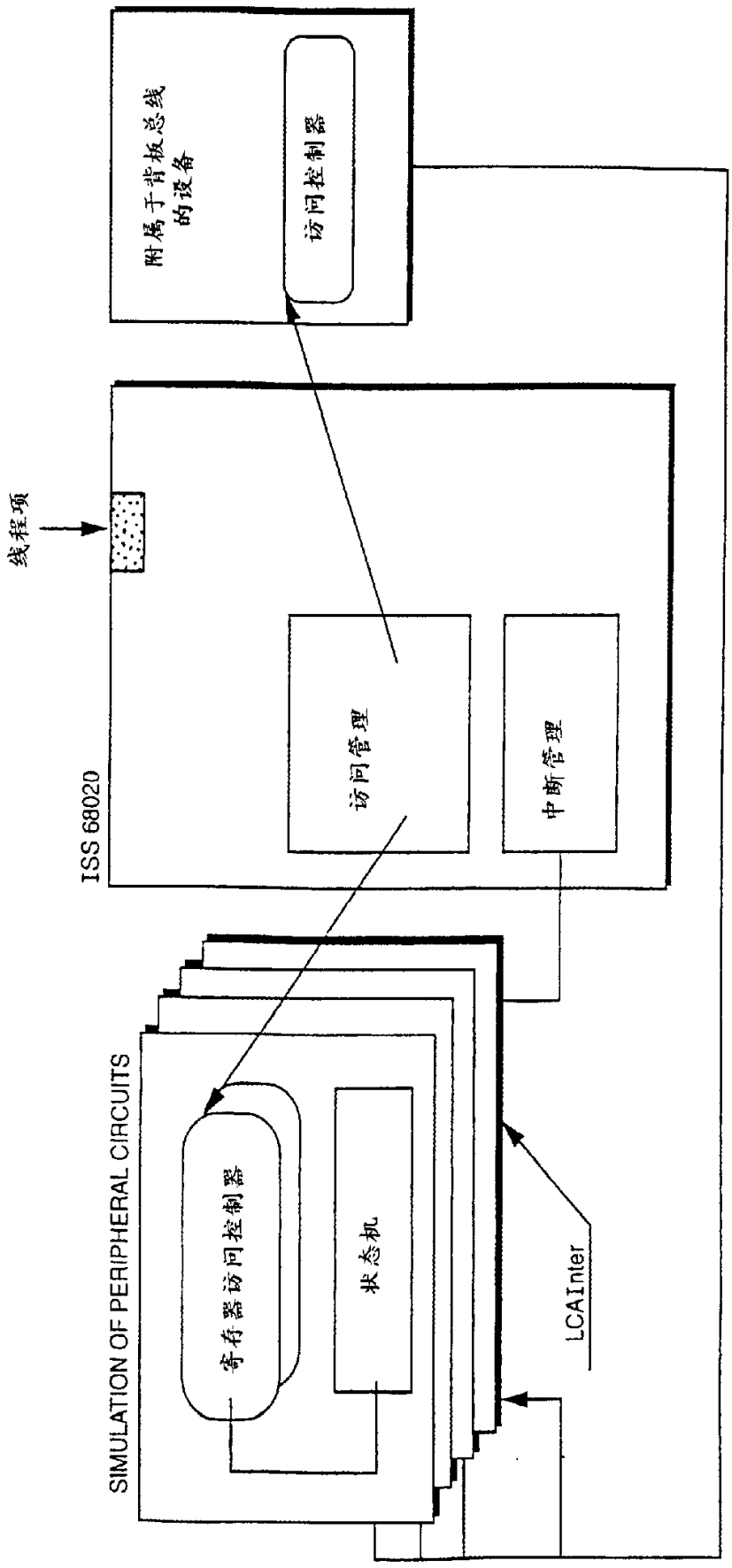

线程项
ISS 68020
访问管理
中断管理
附属于背板总线的设备
访问控制器
SIMULATION OF PERIPHERAL CIRCUITS
寄存器访问控制器
状态机
LCAInter

图 11C

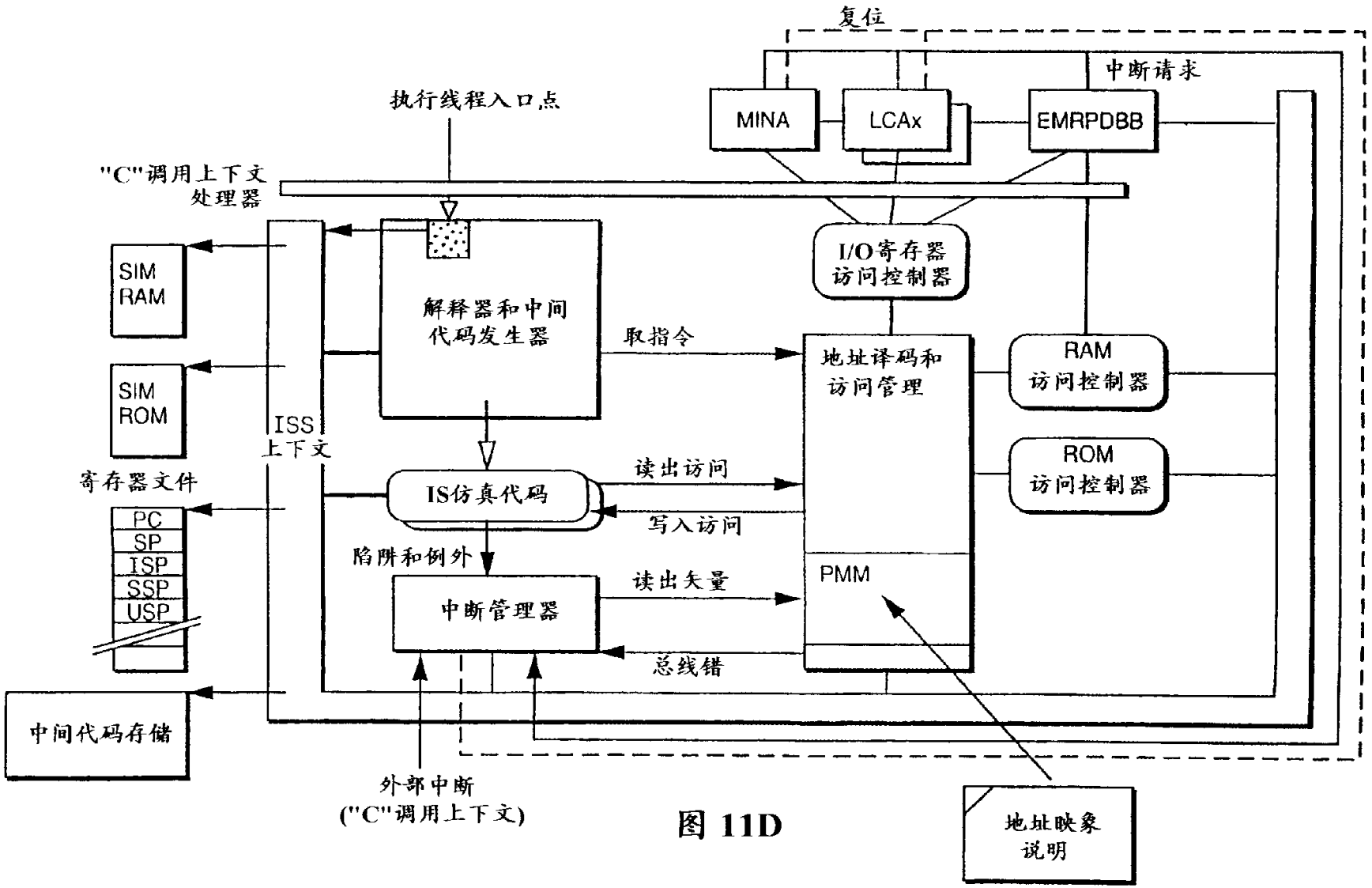

复位
中断请求
执行线程入口点
MINA
LCAx
EMRPDBB
"C"调用上下文
处理器
SIM
RAM
SIM
ROM
寄存器文件
PC
SP
ISP
SSP
USP
ISS
上下文
解释器和中间
代码发生器
I/O寄存器
访问控制器
取指令
地址译码和
访问管理
RAM
访问控制器
ROM
访问控制器
IS仿真代码
读出访问
写入访问
陷阱和例外
中断管理器
读出矢量
PMM
总线错
中间代码存储
外部中断
("C"调用上下文)
地址映象
说明

图 11D

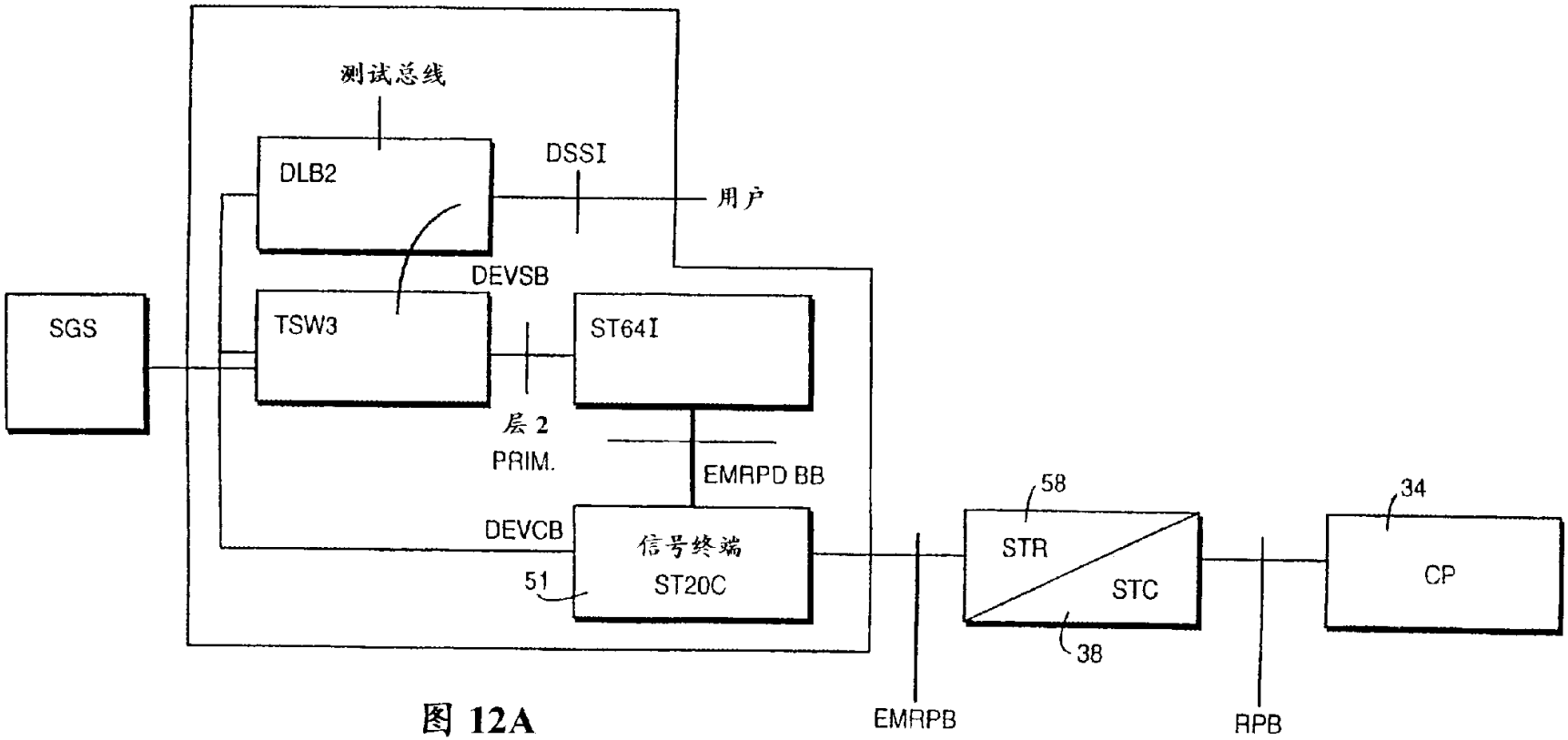

测试总线
DLB2
DSSI
用户
DEVSB
SGS
TSW3
ST64I
层 2
PRIM.
EMRPD BB
DEVCB
信号终端
ST20C
51
58
STR
STC
38
34
CP
EMRPB
RPB

图 12A

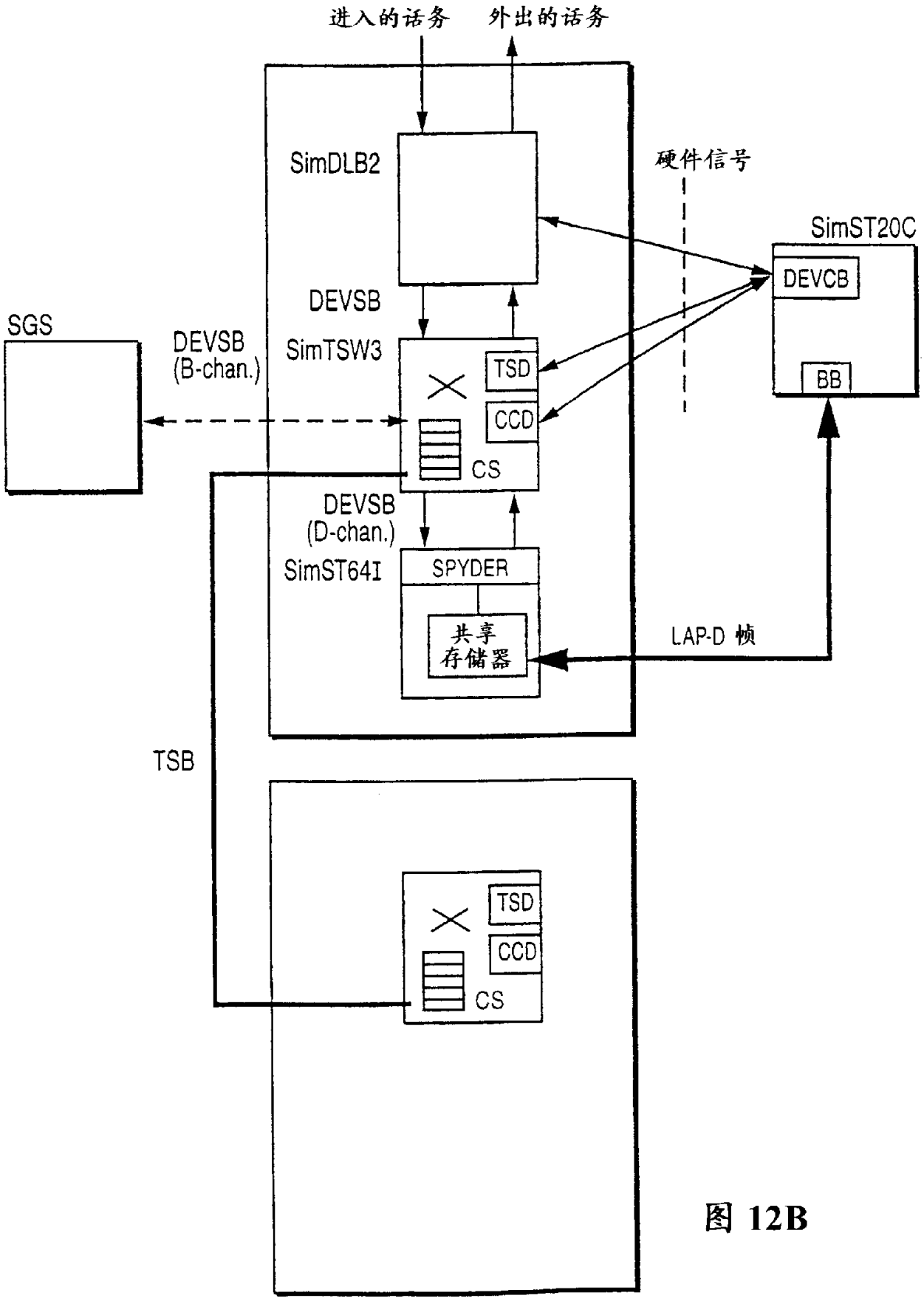

进入的话务
外出的话务
SimDLB2
硬件信号
SimST20C
DEVCB
BB
DEVSB
SGS
DEVSB
(B-chan.)
SimTSW3
TSD
CCD
CS
DEVSB
(D-chan.)
SimST64I
SPYDER
共享
存储器
LAP-D 帧
TSB
TSD
CCD
CS

图 12B

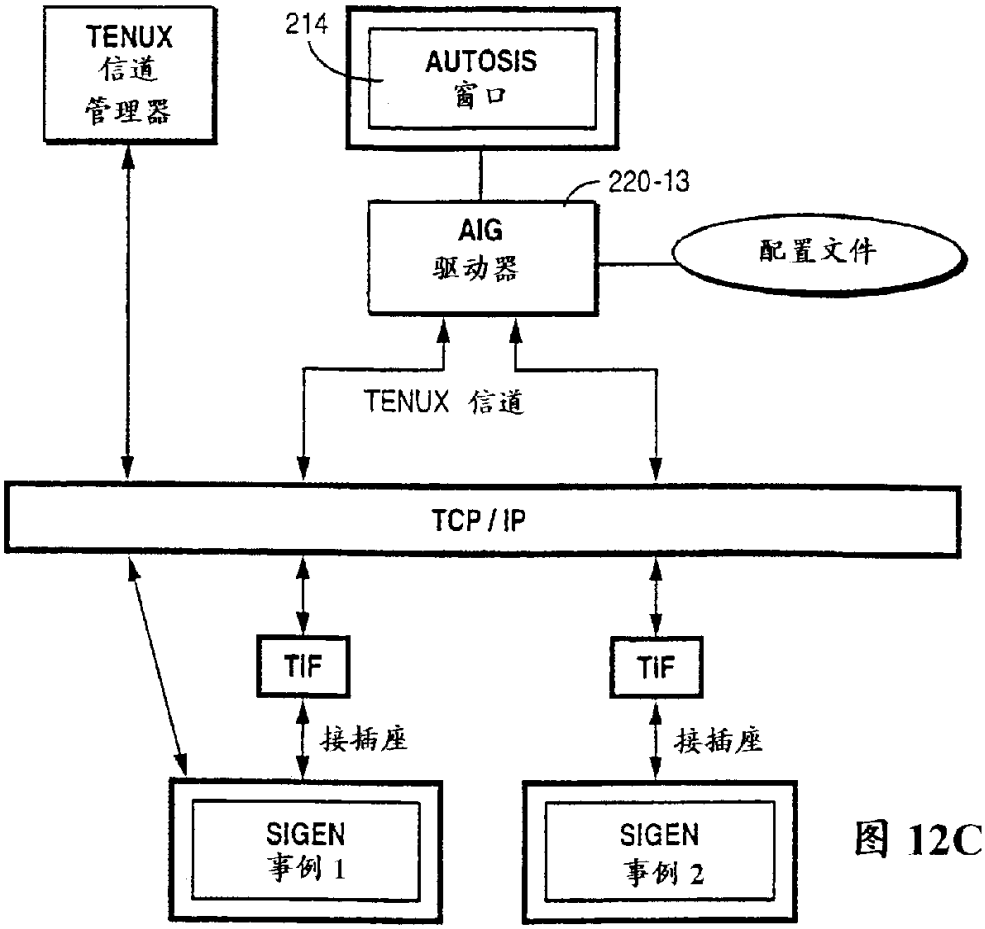

TENUX
信道
管理器
214
AUTOSIS
窗口
220-13
AIG
驱动器
配置文件
TENUX 信道
TCP / IP
TIF
TIF
接插座
接插座
SIGEN
事例 1
SIGEN
事例 2

图 12C

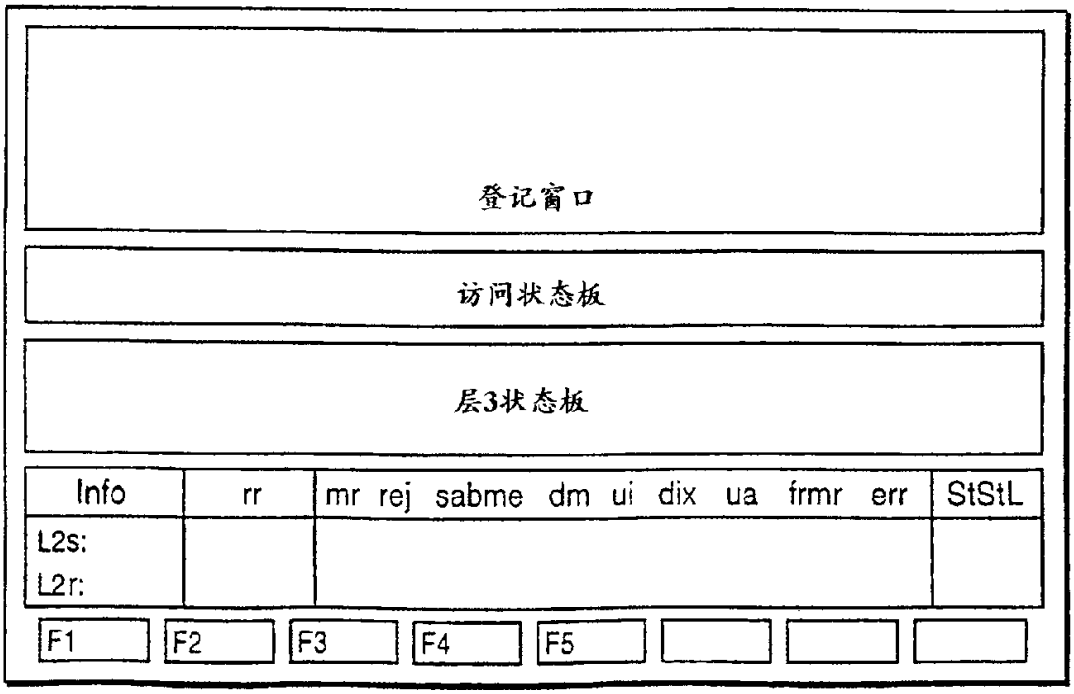

登记窗口
访问状态板
层3状态板
Info
rr
mr rej sabme dm ui dix ua frmr err
StStL
L2s:
L2r:
F1
F2
F3
F4
F5

图 13A

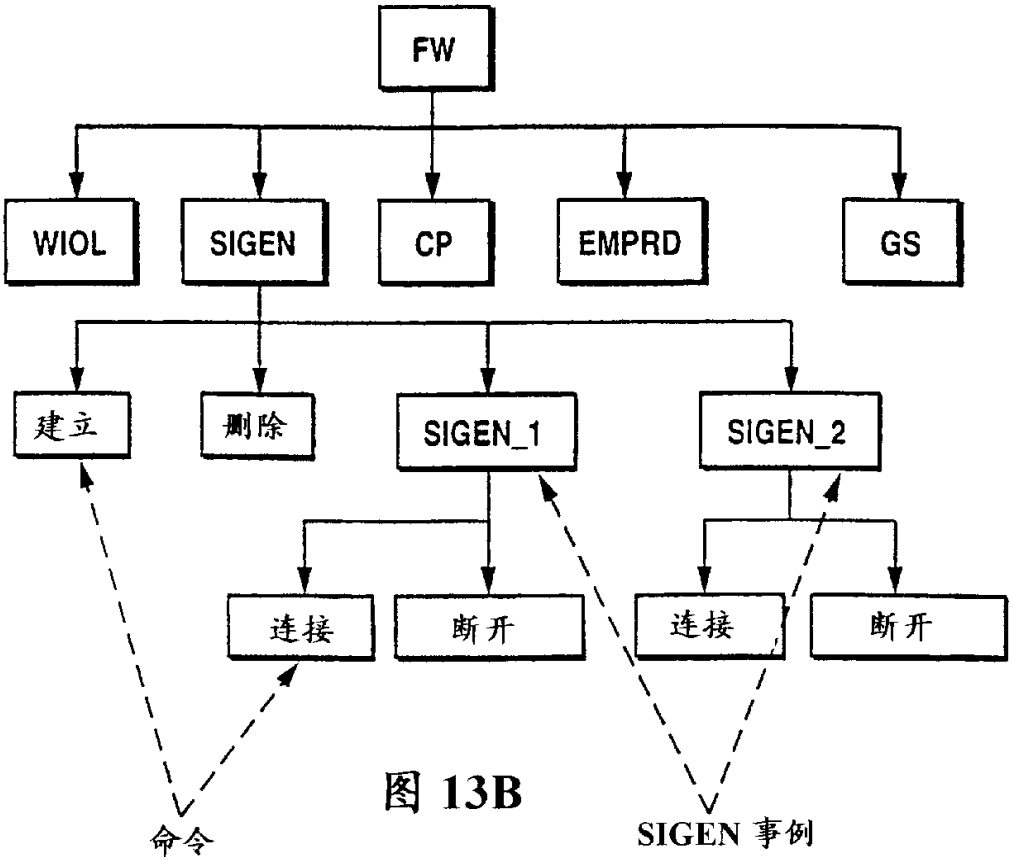

FW
WIOL
SIGEN
CP
EMPRD
GS
建立
删除
SIGEN_1
SIGEN_2
连接
断开
连接
断开
命令
SIGEN 事例

图 13B

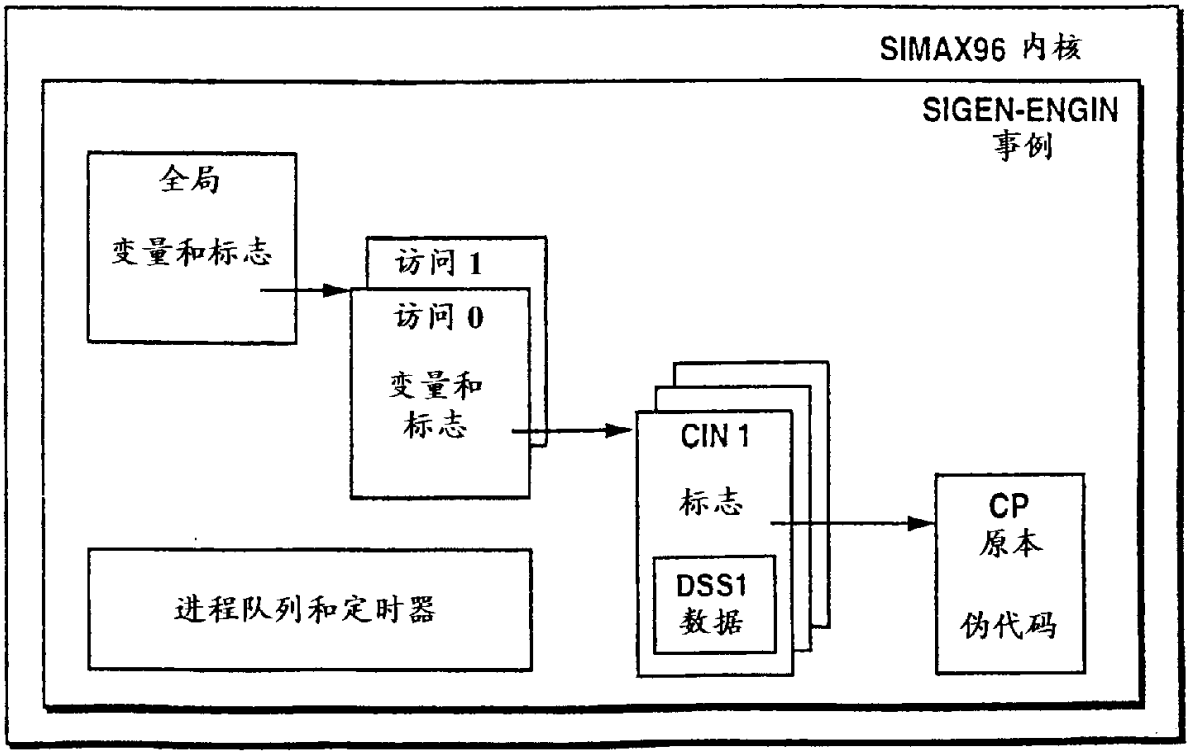

SIMAX96 内核
SIGEN-ENGIN
事例
全局
变量和标志
访问 1
访问 0
变量和
标志
CIN 1
标志
DSS1
数据
CP
原本
伪代码
进程队列和定时器

图 13C

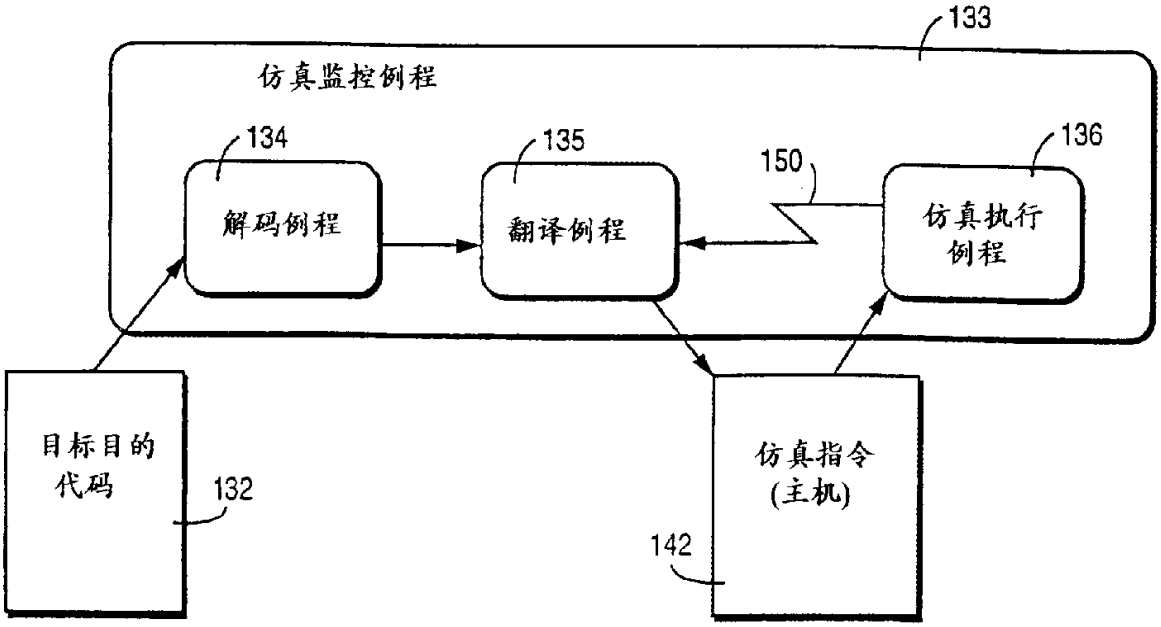

133
仿真监控例程
134
135
136
150
解码例程
翻译例程
仿真执行例程
目标目的代码
132
仿真指令(主机)
142

图 14

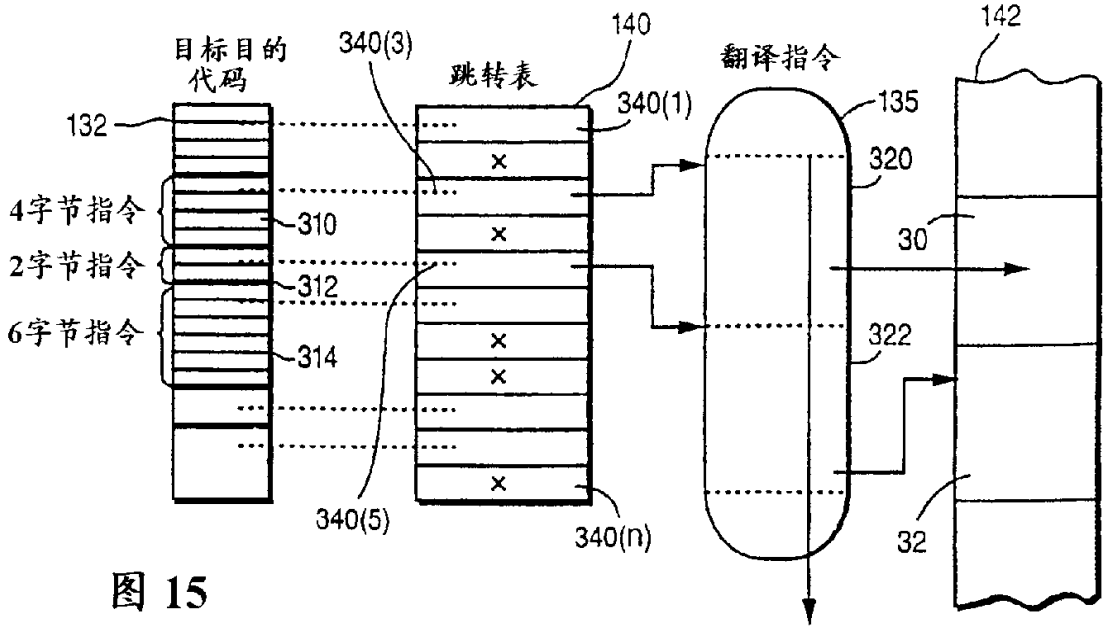

目标目的代码
340(3)
跳转表
140
翻译指令
142
132
340(1)
135
320
4字节指令
310
30
2字节指令
312
6字节指令
314
322
340(5)
340(n)
32

图 15

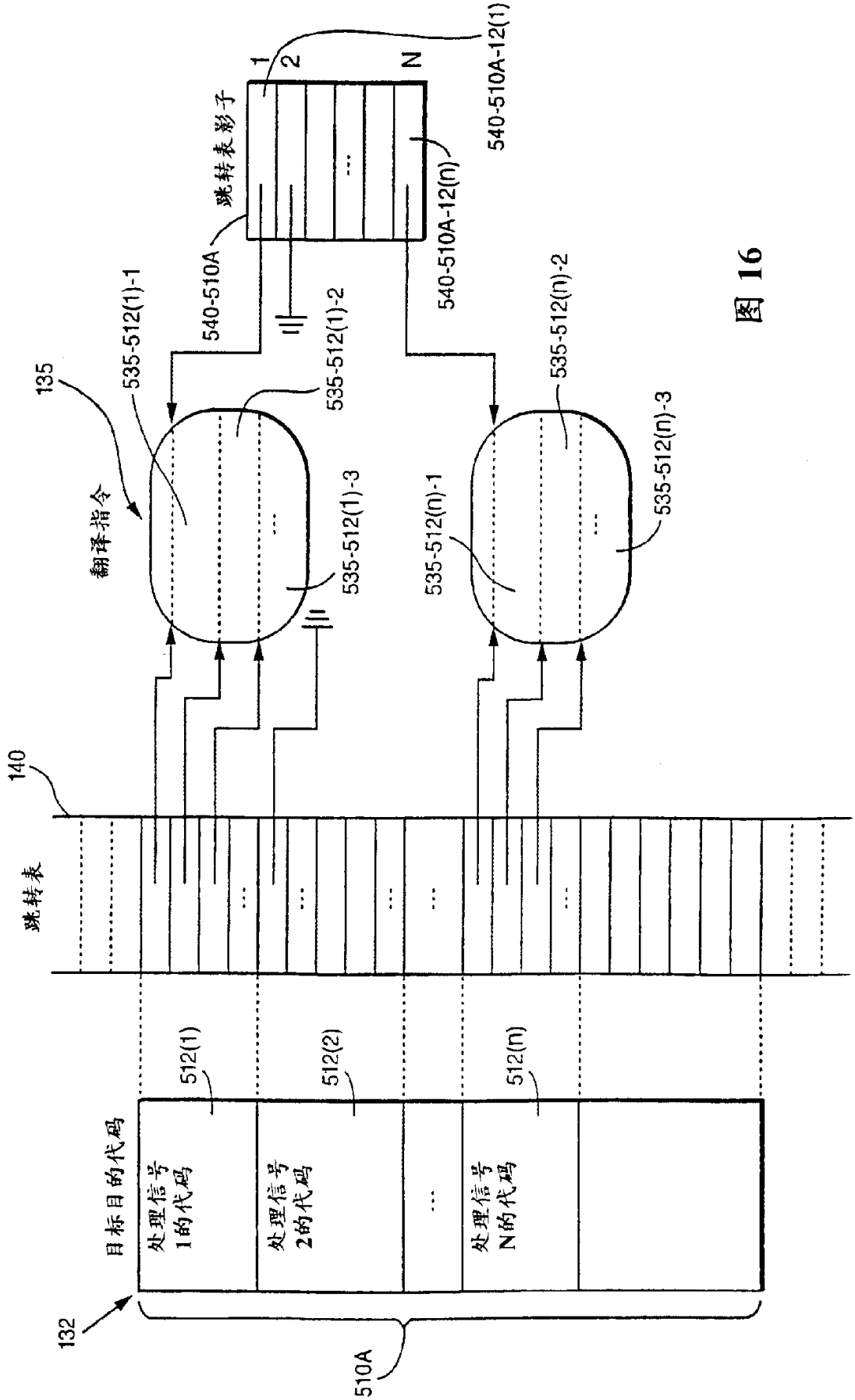

目标目的代码
处理信号1的代码
处理信号2的代码
处理信号N的代码
512(1)
512(2)
512(n)
132
510A
跳转表
140
翻译指令
135
跳转表影子
1
2
N
540-510A
540-510A-12(1)
540-510A-12(n)
535-512(1)-1
535-512(1)-2
535-512(1)-3
535-512(n)-1
535-512(n)-2
535-512(n)-3

图 16

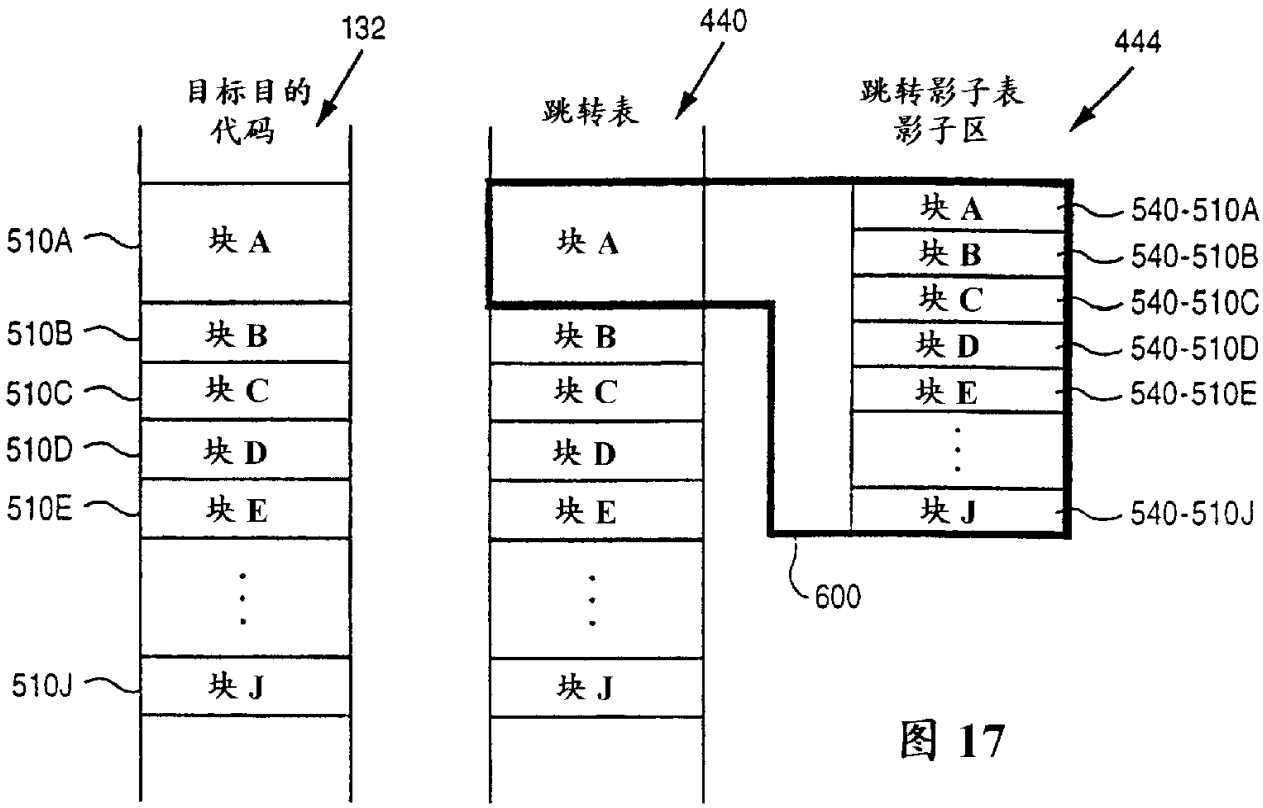

132
目标目的
代码
510A
块 A
510B
块 B
510C
块 C
510D
块 D
510E
块 E
510J
块 J
440
跳转表
块 A
块 B
块 C
块 D
块 E
块 J
444
跳转影子表
影子区
块 A
块 B
块 C
块 D
块 E
块 J
540-510A
540-510B
540-510C
540-510D
540-510E
540-510J
600

图 17

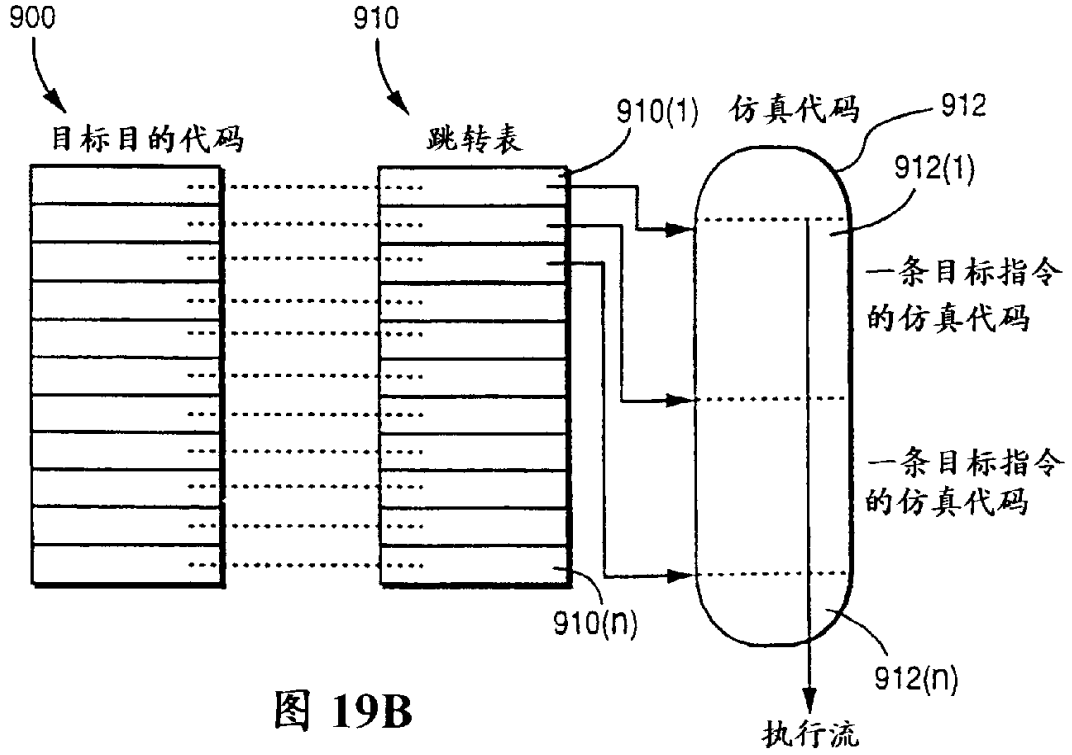
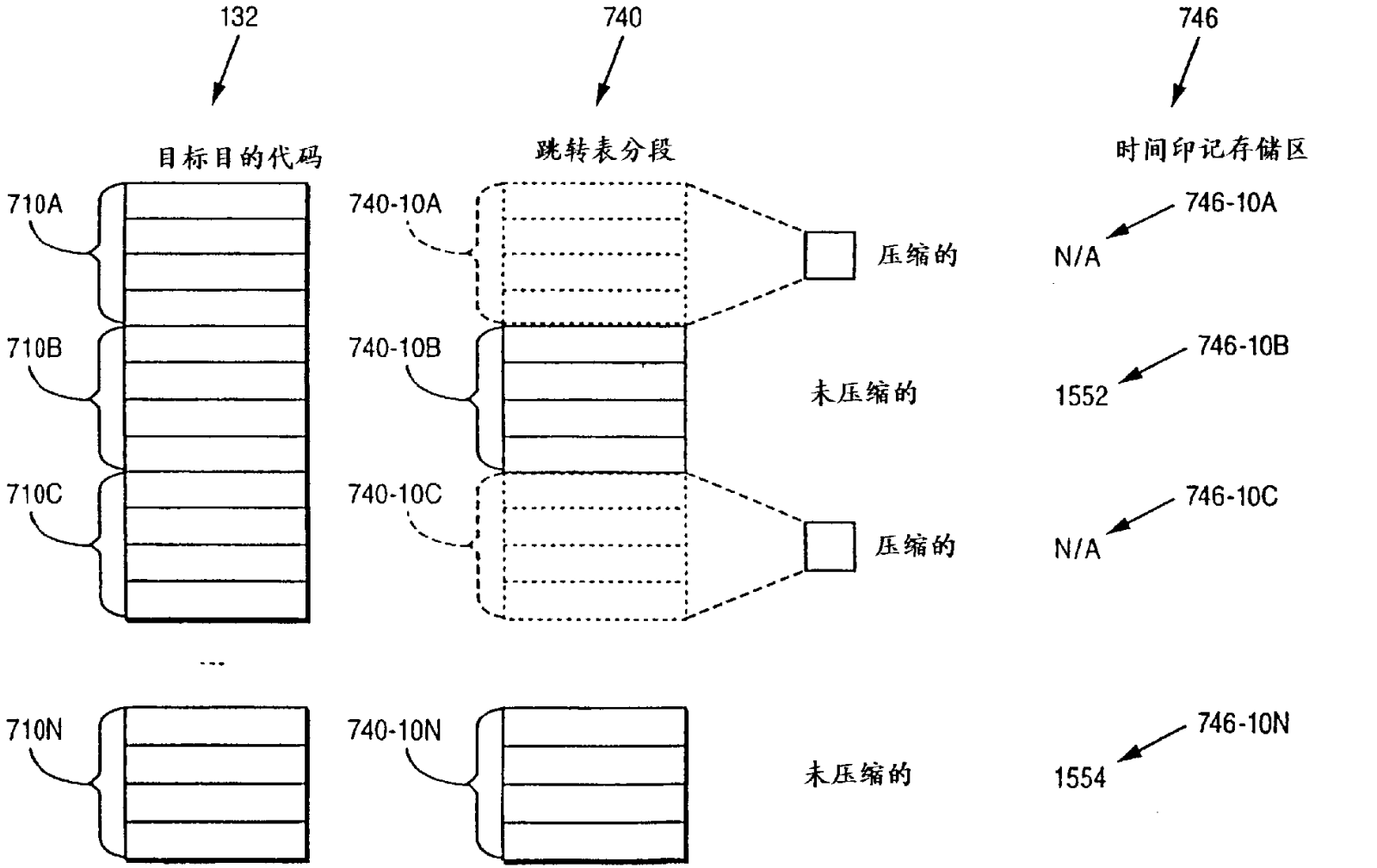

900
目标目的代码
910
跳转表
910(1)
仿真代码
912
912(1)
一条目标指令
的仿真代码
一条目标指令
的仿真代码
910(n)
912(n)
执行流

图 19B

图 18


132
740
746
目标目的代码
跳转表分段
时间印记存储区
710A
710B
710C
710N
740-10A
740-10B
740-10C
740-10N
压缩的
未压缩的
压缩的
未压缩的
746-10A
N/A
746-10B
1552
746-10C
N/A
746-10N
1554

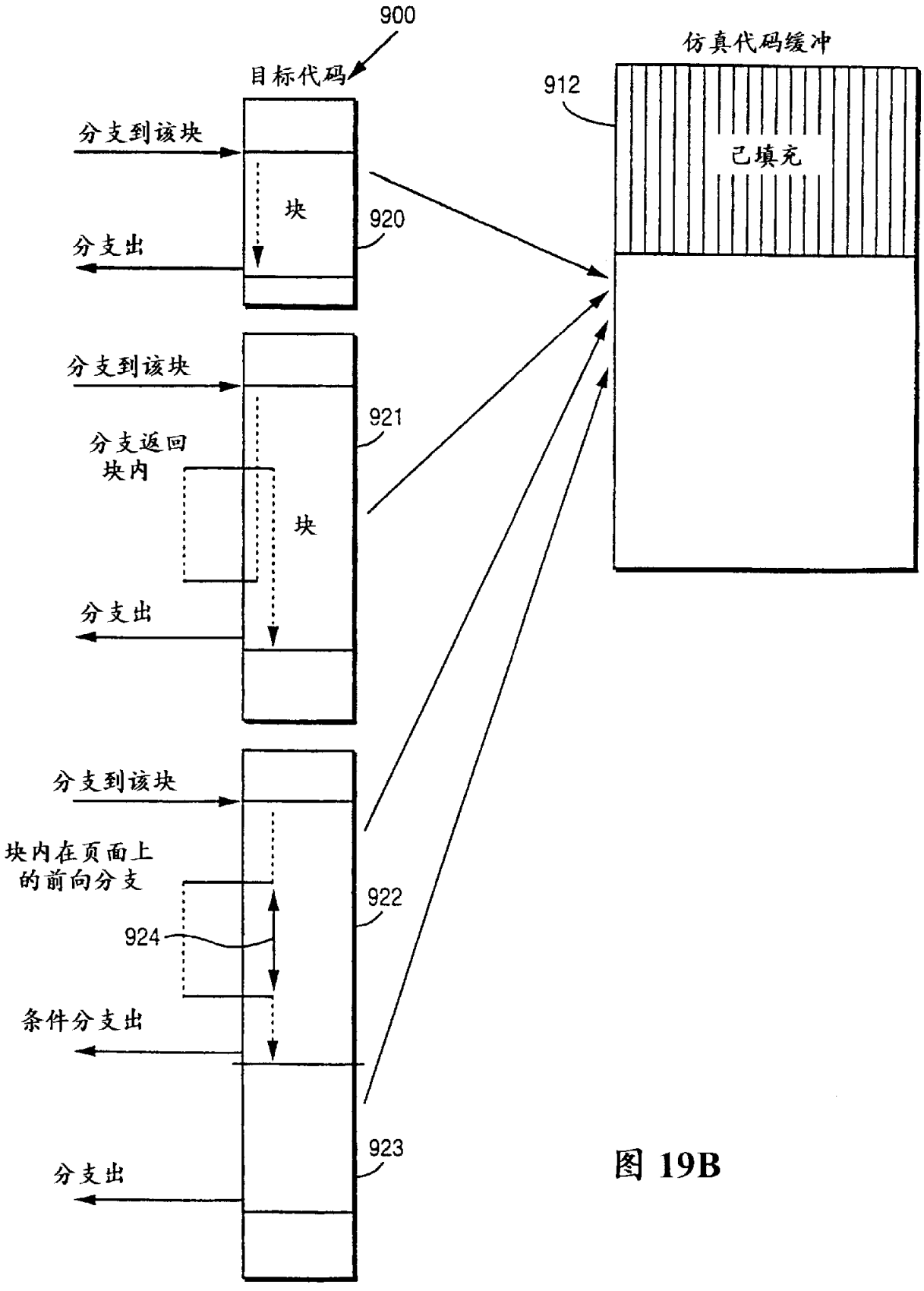

900
目标代码
分支到该块
块
920
分支出
分支到该块
921
分支返回
块内
块
分支出
分支到该块
块内在页面上
的前向分支
922
924
条件分支出
923
分支出
仿真代码缓冲
912
已填充

图 19B

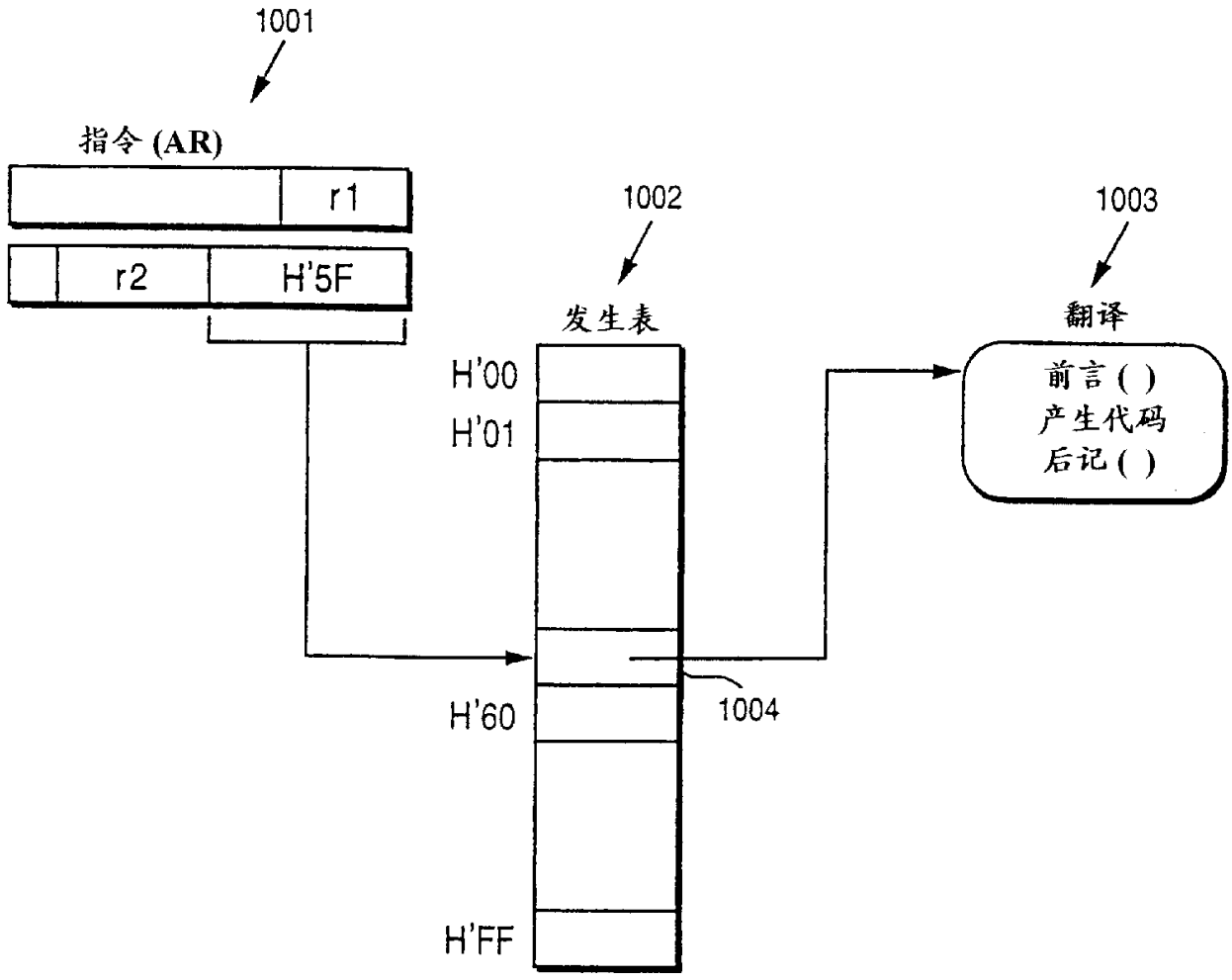

1001
指令 (AR)
r1
r2
H'5F
1002
发生表
H'00
H'01
H'60
H'FF
1004
1003
翻译
前言 ( )
产生代码
后记 ( )

图 20

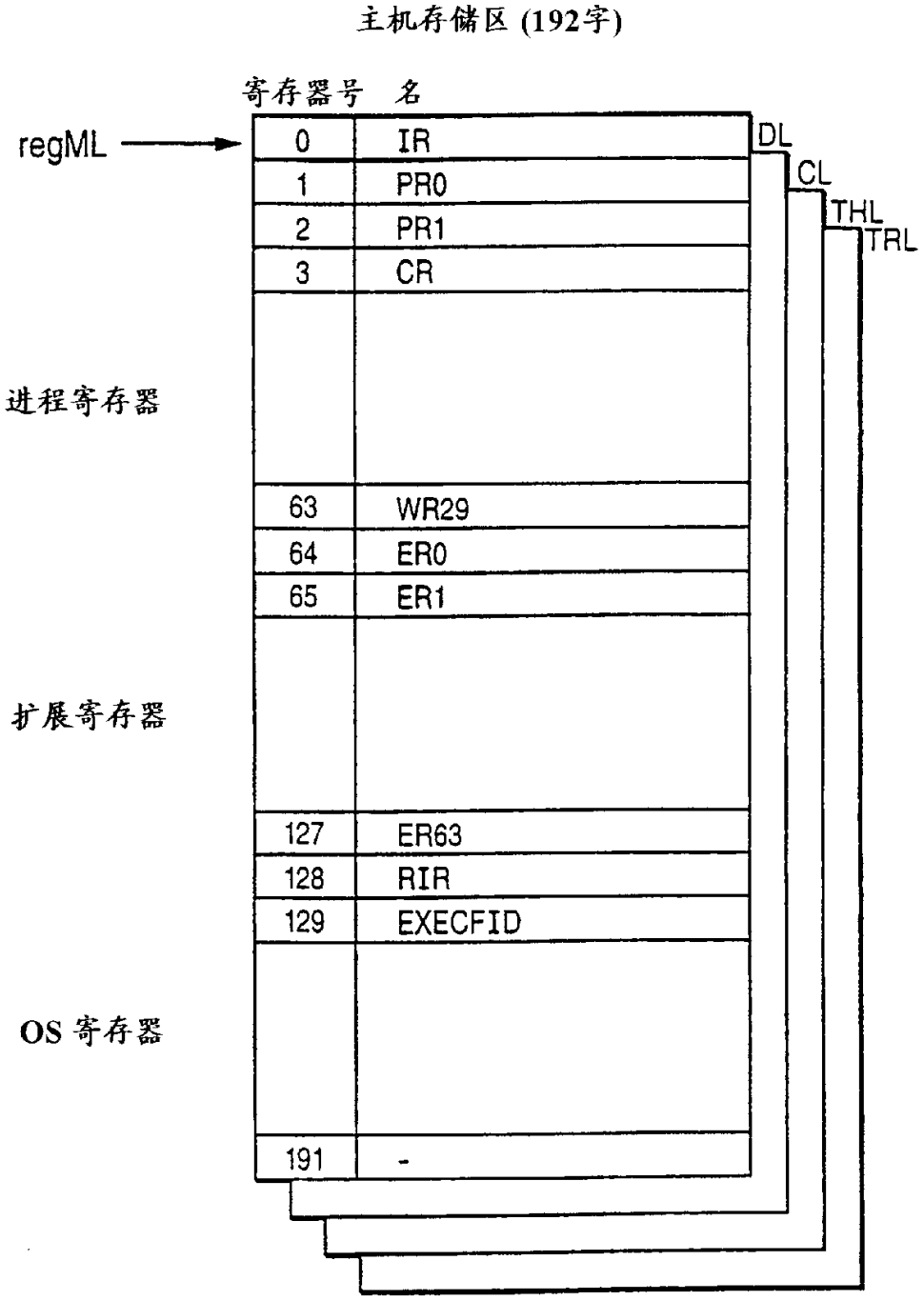

主机存储区 (192字)
寄存器号
名
regML
0
IR
1
PR0
2
PR1
3
CR
进程寄存器
63
WR29
64
ER0
65
ER1
扩展寄存器
127
ER63
128
RIR
129
EXECFID
OS 寄存器
191
-
DL
CL
THL
TRL

图 21