(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 102741817 A
(43)申请公布日 2012.10.17

(21)申请号 201080050130.6
(22)申请日 2010.09.03
(30)优先权数据
61/240,059 2009.09.04 US
(85)PCT申请进入国家阶段日
2012.05.04
(86)PCT申请的申请数据
PCT/NL2010/050555 2010.09.03
(87)PCT申请的公布数据
WO2011/028116 EN 2011.03.10
(71)申请人 英特尔贝内卢克斯公司
地址 荷兰艾恩德霍芬
(72)发明人 H·T·J·茨瓦尔坚科特
A·奥古斯特伊恩 郭园青
J·冯厄特尔 J·A·J·莱特恩
E·Y·M·勒特恩阿夫
(74)专利代理机构 永新专利商标代理有限公司 72002
代理人 刘瑜 王英
(51)Int.Cl.
G06F 9/45(2006.01)
G06F 17/50(2006.01)
G06F 9/318(2006.01)

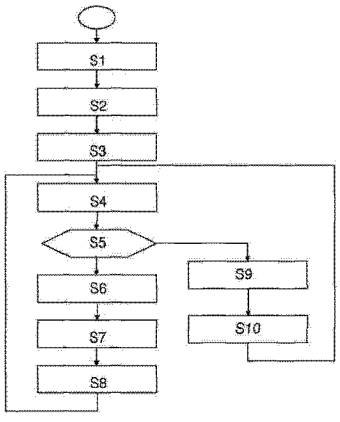
权利要求书 2 页 说明书 23 页 附图 13 页

(54)发明名称

用于生成指令精简方案集的方法、用于根据所生成的集来精简程序的方法，以及能够执行由此精简的程序的可编程处理器

(57)摘要

用于生成针对要由可编程处理器处理的指令的子集的各个指令精简方案的方法包括以下步骤：a)接收代表要在所述可编程处理器上执行的软件的至少一个输入代码样本，所述输入代码包括用于定义第一指令集的多个指令(S1)，b)将被移除指令集初始化为空(S3)，c)确定所述第一指令集的最精简表示(S4)，d)将所述最精简表示的大小与阈值进行比较(S5)，e)如果所述大小大于所述阈值则执行步骤e1至e3，e1)确定所述第一指令集的哪个指令具有最高的编码成本(S6)，e2)从所述第一指令集中移除具有所述最高编码成本的所述指令(S7)，以及e3)将所述指令添加到所述被移除指令集中(S8)，f)重复步骤b-f，其中所述第一指令集由所述被移除指令集形成(S9,S10)。



CN 102741817 A

1. 一种用于生成针对要由可编程处理器处理的指令的子集的各个指令精简方案的方法，包括以下步骤：

a）接收代表要在所述可编程处理器上执行的软件的至少一个输入代码样本，所述输入代码包括用于定义第一指令集的多个指令（S1），

b）将被移除指令集初始化为空（S3），

c）确定所述第一指令集的最精简表示（S4），

d）将所述最精简表示的大小与阈值进行比较（S5），

e）如果所述大小大于所述阈值则执行步骤 e1 至 e3，

e1）确定所述第一指令集的哪个指令具有最高的编码成本（S6），

e2）从所述第一指令集中移除具有所述最高编码成本的所述指令（S7）

e3）将所述指令添加到所述被移除指令集中（S8），

f）重复步骤 b-f，其中所述第一指令集由所述被移除指令集形成（S9，S10）。

2. 根据权利要求 1 所述的方法，包括请求多个指令精简方案和每个指令精简方案的压缩的步骤（S2）。

3. 根据权利要求 1 所述的方法，包括对多个指令精简方案进行迭代并确定为每个指令精简方案所获得的压缩量的步骤。

4. 根据权利要求 1-2 中的其中一个权利要求所述的方法，其中，所述指令包括被单独精简的多个指令字段。

5. 根据权利要求 4 所述的方法，其中，被单独精简的所述指令字段至少包括操作码、指示写端口索引的字段和指示读端口索引的字段。

6. 根据前述任一权利要求所述的方法，其中，针对不同子集的所述指令精简方案具有互不相同的码字宽度，其中所述子集中的至少其中一个子集具有最小码字宽度。

7. 根据权利要求 6 所述的方法，其中，针对每个子集的所述精简方案的码字的大小是整数乘以最小码字宽度，所述整数大于或等于 1。

8. 根据前述任一权利要求所述的方法，其中，互不相同的子集以互不相同的方式进行精简。

9. 根据前述任一权利要求所述的方法，其中，所述子集中的至少其中一个子集被精简为可变长度代码。

10. 根据前述任一权利要求所述的方法，还包括以下步骤：

- 接收包括多个指令的程序，

- 对于每个指令，确定如步骤 a-f 所确定的相应的指令精简，

- 根据所述指令精简来压缩所述指令，

- 提供精简后指令。

11. 根据权利要求 10 所述的方法，包括将所述精简后指令与用于指示所使用的精简类型的至少一个指示符一起提供的步骤。

12. 根据权利要求 10 或 11 所述的方法，其中，所述精简后指令存储在包括多个分段的字中，而且其中每个分段至少包括用于指示所述分段是否是精简后指令的第一分段的指示符。

13. 根据权利要求 10 或 11 所述的方法，其中，所述精简后指令存储在包括多个分段的

字中，其中每个精简后指令包括位于所述精简后指令内的预定位置处的指示符，所述指示符指示用于下一精简后指令的精简类型。

14. 根据权利要求 1 所述的方法，还包括以下步骤：

- 接收可编程处理器的规范，

- 使用所述规范和所生成的各个指令精简方案来确定针对指令解精简器的硬件规范。

15. 被适当编程以实现根据权利要求 1 至 14 中任一权利要求所述的方法的装置。

16. 包括用于促使装置实现根据权利要求 1 至 14 中任一权利要求所述的方法的程序的记录载体。

17. 一种可编程处理器，包括：

- 程序存储器（10），所述程序存储器包括被存储为精简后指令数据的指令序列，所述精简后指令数据至少包括根据第一指令精简方案而被精简为具有 N 个存储器字分段的第一码字的第一指令，以及根据第二指令压缩方案而被精简为具有 M 个存储器字分段的第二码字的第二指令，

- 指令解码器（20），

- 至少一个寄存器文件（40，40a），

- 耦合到所述寄存器文件（40a）的至少一个发布槽（50），

- 指令扩展器（80），所述指令扩展器包括：

- 精简方案识别器（17），用于识别针对从所述程序存储器中获取的精简后指令数据的所述指令精简方案，

- 输入，用于接收程序计数器（PC），

- 存储设备（14），用于临时存储至少一段程序存储器字，

- 选择设备（27），用于从所述程序存储器（10）和所述存储设备（14）中选择精简后指令数据，

- 指令扩展单元（87），用于将所选择的精简后指令扩展成大小为 K 的扩展后指令，

- 控制设备（85），用于响应于所述程序计数器（PC）来生成所述程序存储器的地址（AD），并用于控制所述选择设备，

其中 K、N 和 M 是大于或等于 1 的整数，所述整数 N、M 不大于 K，并且其中 N、M 中的至少一者小于 K。

18. 根据权利要求 17 所述的可编程处理器，其中，所述存储设备（14）是寄存器，并且其中所述选择设备（27）包括多个多路复用模块（27a，…，27d），其中每个多路复用模块选择精简后指令数据的其中一个分段或零个分段。

用于生成指令精简方案集的方法、用于根据所生成的集来精简程序的方法，以及能够执行由此精简的程序的可编程处理器

技术领域

[0001] 本发明涉及用于生成指令精简方案集的方法。

[0002] 本发明还涉及用于根据所生成的指令精简方案集来精简程序的方法。

[0003] 本发明还涉及被适当编程以实现这些方法的装置。

[0004] 本发明还涉及包括用于促使装置实现这些方法中的一种或多种方法的程序的记录载体。

[0005] 本发明还涉及能够执行被如上精简的程序的可编程处理器。

背景技术

[0006] US2002042909 描述了一种用于生成在处理架构中使用的程序指令序列的编译方法，所述处理架构具有用于执行来自相应指令集的指令的架构资源。

[0007] 公知的编译方法输入包括多个源代码指令语句的源文件，所述源代码指令语句包含至少第一种类的指令语句和第二种类的指令语句。

[0008] 该方法选择至少第一指令集和第二指令集。第二指令集是被设计成仅支持由第一指令集所支持的架构资源的子集的精简指令集。通过使用具有不同大小的这至少两个指令集，编译器能够降低被处理的平均代码长度，因为精简码中需要更少的比特来对操作和寄存器进行编码。

[0009] 在该公知的方法中，编译器检测源代码的类型是时间关键代码还是管理代码。被分类为管理代码的代码由第一精简指令集表示，而时间关键代码由第二指令集表示。通过使用具有不同大小的这至少两个指令集，编译器能够降低被处理的平均代码长度，因为精简码中需要更少的比特来对操作和寄存器进行编码。

[0010] 该公知编译器的缺点在于第一和第二指令集的分配仅在时间关键代码和管理代码能够被辨别的情况下才是可能的。

[0011] 本发明的目的是提供一种能够也在更通用的环境中生成一个或多个指令集的方法。

发明内容

[0012] 根据本发明的第一方面，提供一种用于生成针对要由可编程处理器处理的指令的子集的各个指令精简方案的方法，其包括以下步骤：

[0013] a）接收代表要在可编程处理器上执行的软件的至少一个输入代码样本，所述输入代码包括用于定义第一指令集的多个指令，

[0014] b）将被移除指令集初始化为空，

[0015] c）确定第一指令集的最精简表示，

[0016] d）将最精简表示的大小与阈值进行比较，

[0017] e）如果所述大小大于阈值则执行步骤 e1 至 e3，

[0018] e1）确定第一指令集的哪个指令具有最高的编码成本，

[0019] e2）从第一指令集中移除具有最高编码成本的指令，以及

[0020] e3）将所述指令添加到被移除指令集中，

[0021] f）重复步骤 b-f，其中第一指令集由被移除指令集形成。

[0022] 与公知方法相反的是，根据所述第一方面的该方法是通常可应用的。该方法有效地将具有大的互对应性的指令分组到公共群中。选择具有高编码成本的偏离指令（deviating instruction）来形成分离的群。该过程可以重复。

[0023] 对于一些指令，最精简表示可能是原始（未精简）表示。这里使用的原始表示也将被称为“全视图”。

[0024] 指令精简方案的数量以及针对每个指令精简方案的需求压缩可以是固定的。可替换地，可以通过计算考虑多个指令中不同指令的数量以及用于指示最小需求压缩因子的阈值来自动地确定指令精简方案的数量和每个指令精简方案的压缩。在一个实施例中，请求用户指定指令精简方案的数量以及每个指令精简方案的压缩，以便用户能够控制精简过程并试验哪个规范提供最好的结果。

[0025] 在所述方法的实施例中，所述指令包括被单独精简的多个指令字段。通过精简单独的指令字段，与将指令作为整体进行精简的情况相比能够获得更高的压缩。如果两个指令在特定的指令字段中具有相同的值但以其他方式不同，则那个指令字段的值可以由公共代码精简，但是将指令作为整体的精简将需要不同的代码。关于指令字段的知识优选从处理器描述文件中获得。精简方案可以对应于可编程处理器的某个视图。处理器视图被定义为硅蜂窝（silicon hive）编译器的目标，在该编译器中仅处理器资源的子集才是可用的。关于处理器资源的知识从处理器描述文件中获得。

[0026] 在所述实施例的变形中，被单独精简的指令字段至少包括操作码、用于指示写端口索引的字段以及用于指示读端口索引的字段。精简这些字段产生高的代码大小降低。另外，下面的字段也可以被单独精简：用于指示结果端口（总线）的字段、用于指示写端口选择的字段以及包括立即值的字段。

[0027] 在一个实施例中，针对不同子集的指令精简方案具有互不相同的码字宽度，其中这些子集中的至少其中一个子集具有最小码字宽度。在子集被允许具有互不相同的大小的情况中，一些子集可以通过使用更小的码字进行精简，从而节省编码空间。一些子集可以具有互不相同的精简方案，但是可以由具有互相相同大小的码字进行编码。

[0028] 在一个实施例中，针对每个子集的精简方案的码字的大小是大于或等于 1 的整数乘以最小码字宽度。通过该措施简化了指令的读取。部分指令可以不被精简。这些指令的长度可以等于从其中获取被精简或未被精简的指令的指令存储器的宽度，但是可以可替换地更小。优选地，指令不大于指令存储器的宽度以保持短的指令获取时间。

[0029] 在一个实施例中，互不相同的子集以互不相同的方式进行精简。例如，根据第一视图而被精简的指令可以使用采用编译时间可编程寄存器的查找表解精简（扩展），而根据第二视图而被精简的指令可以使用采用硬连线查找表的查找表。如果子集中的至少一个子集被精简为可变长度代码，则是有利的。通过将可变长度代码（VLC）仅应用于指令的子集，一方面，优点在于能够在那个子集中获得高的指令压缩因子，而另一方面，能够保持适中的代

码数量，从而该子集中的代码能够被容易地解精简（被扩展）。精简方案的唯一约束是同一视图中的精简后指令具有小于或等于特定最大长度的大小。因此，长度小于或等于“视图”长度的 VLC 精简后指令将适合于该视图。

[0030] 根据本发明的第二方面，根据第一方面的方法进一步包括以下步骤：

[0031] - 接收包括多个指令的程序，

[0032] - 对于每个指令，确定如步骤 a-f 所确定的相应的指令精简方案，

[0033] - 根据所述指令精简方案来压缩指令，

[0034] - 提供精简后指令。

[0035] 以此方式，被精简的程序可以与用于定义指令精简方案集的程序相同。

[0036] 精简后指令可以存储在特定地址范围中，从而能够根据指令地址而清楚地知道使用什么类型的精简方案。

[0037] 在一个实施例中，根据第二方面的方法进一步包括将精简后指令与用于指示使用的精简方案类型的至少一个指示符一起提供的步骤。这使得能够将精简后指令存储在与用于促成处理的原始程序相同的序列中。另外，精简后指令不需要被对准。

[0038] 在一个实施例中，精简后指令存储在包括多个分段的字中，而且每个分段至少包括用于指示所述分段是否是精简后指令的第一分段的指示符。

[0039] 在另一实施例中，精简后指令存储在包括多个分段的字中，其中每个精简后指令包括位于精简后指令内的预定位置处的指示符，所述指示符用于指示下一精简后指令的视图。其优点在于，即使根据不同视图的指令具有不同的大小，用于对精简后指令进行解精简的指令扩展器仍然能够准确和及时地预获取精简后指令的下一码字。

[0040] 根据本发明的第三方面，提供一种被适当编程以执行根据第一方面或第二方面的方法的装置。

[0041] 根据本发明的第四方面，提供一种包括程序的记录载体，所述程序促使装置执行根据第一方面或第二方面的方法。

[0042] 根据本发明的第五方面，提供一种可编程处理器，其包括：

[0043] - 程序存储器，该程序存储器包括被存储为精简后指令数据的指令序列，精简后指令数据至少包括根据第一精简方案而被精简为具有 N 个存储器字分段的第一码字的第一指令，以及根据第二精简方案而被精简为具有 M 个存储器字分段的第二码字的第二指令，

[0044] - 指令解码器，

[0045] - 至少一个寄存器文件，

[0046] - 耦合到所述寄存器文件的至少一个发布槽（issue slot），

[0047] - 指令扩展器，包括：

[0048] - 精简方案识别器，用于识别针对从程序存储器中获取的精简后指令数据的指令精简方案，

[0049] - 输入，用于接收程序计数器，

[0050] - 存储设备，用于临时存储至少一段程序存储器字，

[0051] - 选择设备，用于从程序存储器和存储设备中选择精简后指令数据，

[0052] - 指令扩展单元，用于将所选择的精简后指令扩展成大小为 K 的扩展后指令，

[0053] - 控制设备，用于响应于程序计数器来生成程序存储器的地址，并用于控制选择设

备，

[0054] 其中 K、N、M 是大于或等于 1 的整数，整数 N、M 不大于 K，并且其中 N、M 中的至少一者小于 K。

[0055] 本发明的第一至第五方面是可以进一步包括设计和测试设备的环境的一部分。

附图说明

[0056] 下面将更详细地描述本发明的这些和其他方面。其中：

[0057] 图 1 示出了传统可编程处理器，

[0058] 图 2 示出了另一传统可编程处理器的一部分，

[0059] 图 3 示意性地示出了图 2 中部分示出的处理器的程序存储器的内容，

[0060] 图 4 示出了用于确定指令精简方案集的方法，

[0061] 图 5 示出了用于生成指令精简方案集的工具，

[0062] 图 6 示出了用于精简程序的工具，

[0063] 图 7 示意性地示出了根据本发明的可编程处理器的第一实施例，

[0064] 图 8 更详细地示出了图 7 的一部分，

[0065] 图 9 更详细地示出了图 8 的一部分，

[0066] 图 10 更详细地示出了图 9 的一部分，

[0067] 图 11 示出了用于在图 7 的处理器中处理指令的方法，

[0068] 图 12 示意性地示出了根据本发明的可编程处理器的第二实施例，

[0069] 图 13 示出了用于在图 12 的处理器中处理指令的方法，

[0070] 图 14 示意性地示出了根据本发明的可编程处理器的第三实施例，

[0071] 图 15 示意性地示出了根据本发明的用于生成可编程处理器的硬件描述的工具。

具体实施方式

[0072] 在下面的详细描述中，阐述了多个具体细节以提供对本发明的透彻理解。然而，本领域技术人员将理解，可以在没有这些具体细节的情况下实践本发明。在其他实例中，没有详细描述公知的方法、过程和部件，以便不模糊本发明的各个方面。

[0073] 下文中，将参照示出了本发明的实施例的附图来更充分地描述本发明。然而，本发明可以被体现为许多不同的形式，并且不应当被解释为局限于本文阐述的实施例。相反地，提供这些实施例，以使得本公开将透彻和完整以及将向本领域技术人员充分地传达本发明的范围。应当理解，当提及一个元件“连接”或“耦合”到另一元件时，该元件能够直接连接或耦合到其他元件，或者可以存在中间元件。相对地，当提及一个元件“直接连接到”或“直接耦合到”另一元件时，不存在中间元件。通篇中，类似的标号指代类似的元件。如本文使用的，术语“和 / 或”包括被关联列出的项中的一个或多个的项的任意和所有组合。

[0074] 应当理解，虽然术语“第一”、“第二”、“第三”等在本文中可以用于描述各种元件、部件和 / 或部分，但是这些元件、部件和 / 或部分不应当受到这些术语的限制。这些术语仅用于将一个元件、部件或部分与另一元件、部件和 / 或部分相区分。因此，下面讨论的第一元件、部件和 / 或部分能够用第二元件、部件和 / 或部分来表述，但是并不背离本发明的教导。

[0075] 除非另有定义，否则本文使用的所有术语(包括技术和科技术语)都具有与本发明所属技术领域中的普通技术人员所通常理解的相同的意义。还应当理解，诸如在普通使用的字典中所定义的那些术语之类的术语应当被解释为具有与它们在相关领域的上下文中所具有的意义相一致的意义，并且不应当以理想或过于正式的意义进行解释，除非在本文中明确地如此定义。本文提到的所有出版物、专利申请、专利和其他参考都通过引用而被全部合并到本文。万一冲突，包括定义的本说明书将进行控制。另外，材料、方法和示例仅是说明性的，而且并非意欲进行限制。

[0076] 图 1 示意性地示出了可编程处理器。在图 1 所示的示例中，可编程处理器是 VLIW 处理器。VLIW 处理器并行地处理被分组成 VLIW 指令字的多个指令字。这些通常由软件开发工具来产生。图 1 所示的 VLIW 处理器包括程序存储器 10 和耦合到程序存储器的指令解码器 20。程序存储器 10 包含 VLIW 指令字。VLIW 处理器还包括经由第一选择元件 30a,...,30m 耦合到多个总线 70 的多个寄存器文件 40a,...,40n。寄存器文件具有一个或多个输入端口和一个或多个输出端口。寄存器端口由数据输入或输出和地址输入构成。

[0077] 清楚起见，图 1 仅示出了单个发布槽 50。实际上，VLIW 处理器将具有多个发布槽。多个发布槽中的每个发布槽处理来自 VLIW 指令字的特定的指令字。发布槽包括能够对输入数据执行有限操作集的一个或多个功能单元。发布槽 50 具有指令解码器 51 和多个功能单元 FU 53a,...,53k，例如乘法器、加法器、移位器等。发布槽 50 还具有第二选择元件 52a,....,52k，用于选择来自各种源(例如寄存器文件)的输入数据和来自解码器 20 的用于提供立即值的输出。功能单元 53a,...,53k 和第二选择元件 52a,...,52k 的操作由操作解码器 51 控制。处理器还包括多个第三选择元件 60a,...,60k，用于选择性地将功能单元 53a,...,53k 耦合到总线 70。

[0078] 指令通常包括多个指令字段。每个字段控制可编程处理器的数据路径的具体项。在该特别示例中，指令可以包括下面的用于操作码、结果端口选择的选择(bus_select)、写端口的选择(wp_select)、写端口的索引的规范(wp_index)、读端口的索引的选择(rp_index)和立即的规范等 6 种指令字段。

[0079] 通常，每个发布槽具有一个操作码指令字段。该字段选择要由发布槽执行哪个操作。该操作由发布槽的其中一个功能单元执行。操作码被解码成功能单元选择信号和操作类型(optype)，以激活特定的 FU 和那个 FU 的特定操作。在一些情况中，操作码可以不存在，例如如果发布槽专用于处理仅一个操作，例如加载立即值。

[0080] 具有不止一个发布槽输出的每个总线具有单独的用于指示哪个发布槽输出被连接到总线的 bus_select 字段。

[0081] 每个寄存器文件输入端口连接到一个或多个总线。如果存在连接到一个写端口的不止一个总线，则多路复用器选择到寄存器文件的输入端口的正确的总线。写端口选择(wp_sel)指令字段包含用于该多路复用器的选择值。专门的代码 wp_sel=“11..11”用于指示在写端口上不应当发生写动作。

[0082] 该指令字段包含被写入寄存器文件的寄存器地址。每个寄存器写端口都具有单独的 wp_index。

[0083] 该指令字段包含从寄存器文件读取的寄存器地址。每个寄存器读端口都具有单独的 rp_index。

[0084] 立即指令字段包含能够被用作发布槽中的其中一个功能单元的输入的值。

[0085] 并非根据本发明的、被称为立即覆盖(immediate overlaying)的代码精简方式依赖于这样的事实,即发布槽中的功能单元的输入可以使用寄存器文件输出或立即字段作为输入。optype 确定输入是什么,而且对于每个指令而言这可以不同。如果操作码指示寄存器文件输出被用于操作,则发布槽的立即字段是冗余的。因此,如果立即值被选择作为输入,则针对连接到发布槽的输入的寄存器输出端口的寄存器索引字段是冗余的。由于该立即字段和该寄存器索引字段从来不在同一指令中使用,所以两个字段可以组合。立即和寄存器索引字段(的一部分)的组合将被称为立即覆盖。

[0086] 代码精简的另一方式利用不同视图的使用。处理器视图被定义为编译器的目标,在该编译器中仅处理器资源的子集才是可用的。子集可通过以下约束来定义:

[0087] - 寄存器文件属性:输入 / 输出端口的数量、地址范围

[0088] - 功能单元属性:立即范围、操作码的数量

[0089] - 总线的数量

[0090] - 完整的发布槽、功能单元、寄存器文件,... 等。

[0091] 就代码精简而言,如果用于控制子集的指令比特的数量明显小于用于完整处理器的指令比特的数量,则处理器视图是有用的。处理器能够具有不止一个视图。

[0092] 用于支持视图机制的硬件如图 2 所示。在第一视图(视图 0)中,所有的处理器资源都是可用的。程序存储器字包括单个指令。在第二视图(视图 1)中,每个程序存储器字包括 2 个精简后指令。在第三视图(视图 2)中,每个程序存储器字包括 4 个精简后指令,如图 3 所示意性示出的那样。

[0093] 注意,全视图并非必须等于程序存储器宽度。有时,选择更宽的程序存储器以使得更小视图能够更好的压缩是有意义的。例如,假设处理器具有 60 比特的全宽,而且最小视图具有 16 比特的宽度。通过采取 60 的程序存储器宽度,最小视图具有 60/16=3.75 的压缩。这必须被四舍五入成 2 的一次幂,这导致压缩因子 2。如果程序存储器的宽度等于 64,则压缩变为因子 4。

[0094] 在图 3 中,针对视图 0 的指令以常规方式放置,从而从地址 0 开始。对于视图 0 指令,PC 等于程序存储器地址。针对视图 1 的指令从程序存储器地址 0x1B 开始。针对视图 1 的第一指令的程序计数器由具有值 0 的一个 LSB (表明程序存储器字 0x1B 的下半区包含指令)、具有值“01”的两个 MSB (表明该指令是视图 1 指令)以及接着等于程序存储器地址 0x1B 的中间比特形成。对于 PC 而言,结果是 0x1036。随后指令的 PC 值能够通过递增该 PC 而找到。从地址 0x2A 开始,针对视图 2 的指令被布置。针对第一视图 2 指令的 PC 值由具有值“00”的 2 个 lsb (用于从所述字中的 4 个精简后指令中选出存储器地址为“0x2A”的第一个精简后指令)和具有值 2 的 2 个 MSB (表示视图号)形成。结果是 0x2150。

[0095] 如图 2 所示,用于支持视图机制的硬件包括第一精简后指令选择器 22 和第二精简后指令选择器 23、第一指令解精简单元 24 和第二指令解精简单元 25 和全指令选择器 26。当运行程序时,处理器指示将用它的程序计数器(PC) 12 来执行指令。程序计数器 12 的输出具有第一部分 12a、第二部分 12b 和第三部分 12c,其中第一部分 12a 用于控制第一指令解精简单元 24 和第二指令解精简单元 25,第二部分 12b 用于处理所需的程序存储器字,以及第三部分 12c 用于控制全指令选择器 26。部分 12c 指示视图选择。在全视图模式中,PC

等于程序存储器地址。针对全视图(视图 0)的程序存储器字确切地包含一个全指令。全指令选择器 26 仅将该指令传递给指令解码器 20。

[0096] 在“精简后视图”模式中,PC 不能直接被映射到程序存储器地址。在那种情况中,PC 的部分 12a 指示程序存储器字中的哪个指令需要被选择。如果由部分 12b 中的地址所指示的程序存储器字被读取,则由被程序计数器的部分 12a 所控制的第一精简后指令选择器 22 和第二精简后指令选择器 23 所选择的精简后指令在指令解精简单元 24 和 25 中被提取。

[0097] 指令解精简单元 24、25 将精简指令转换成全指令。

[0098] 对于每个被实施的精简方案而言,对应于处理器的特定视图来实施指令解精简器 24、25。解精简器 24、25 的输出是全指令选择器 26 的输入。PC 的部分 12c(view_select)确定全指令选择器 26 的哪些输入被选择作为指令选择器 26 的输出。

[0099] 在参照图 2 和图 3 所描述的代码精简方法中,编程器定义需要在哪个视图中运行哪个代码。基本构件块的所有指令应当以一个视图为目标。视图的切换仅能够通过跳转操作(jump operation)的方式来实现。这仅通过跳转指令来实现。在指令选择和调度之后,汇编器定义精简后指令并确定多个精简后指令如何被放置在一个程序存储器字中。链接器定义每个被构建的字的程序存储器地址。通常,针对一个视图的指令被分组在一起。一个组的第一指令总是从程序存储器字的比特 0 开始。同一基本构件块中的随后指令随后被放置在程序存储器字中,如果乘以压缩因子的精简后指令大小小于程序存储器宽度,则在它们之间留下伪比特(dummy bit)。如果字满了,则在下一程序存储器字的比特零处放置下一指令。

[0100] 图 4 示意性地示出了根据本发明的用于生成针对要由可编程处理器处理的指令子集的各个指令精简方案的方法。该方法包括用于接收代表将在可编程处理器上执行的软件的至少一个输入代码样本的第一步骤 S1,所述输入代码包括用于定义第一指令集的多个指令(S1)。

[0101] 在所示的实施例中,该方法包括第二步骤 S2,其中要求用户指定指令精简方案的数量以及针对每个指令精简方案的最小需求压缩。这样,用户能够控制精简过程,并试验哪个规范提供最好的结果。步骤 S2 不是关键的。可替换地,指令精简方案的数量以及每个指令精简方案的需求压缩可以是固定的。在再一实施例中,可以通过计算考虑多个指令中不同指令的数量以及阈值来自动地确定指令精简方案的数量和每个指令精简方案的压缩。

[0102] 在第三步骤(S3)中,定义被移除指令集,而且该被移除指令集被初始化为空集。

[0103] 之后在步骤 S4 中,确定第一指令集的最精简表示。

[0104] 在步骤 S5 中,将所述最精简表示的大小与阈值进行比较。依赖于该比较的结果,在步骤 S5 之后执行步骤 S6 至 S8 或者执行步骤 S9 至 S10。如果最精简表示的大小大于所述阈值,则执行步骤 S6 至 S8,否则执行步骤 S9 至 S10。

[0105] 在步骤 S6 中,确定第一指令集的哪个指令具有最高的编码成本。随后在步骤 S7 中,从第一指令集中移除该指令并且在步骤 S8 中将其添加到移除指令集中。之后程序流转到步骤 S4。

[0106] 在确定最精简表示的大小不大于所述阈值的情况中,第一指令集被重新定义为移除指令集(步骤 S9),而且移除指令集被重新定义为空(步骤 S10)。

[0107] 根据本发明第一方面的该方法可以在如图 5 所示的视图生成工具 VG 中使用。如图所示，视图生成工具从例如 ELF(执行语言格式)的视图无关可再定位对象文件 115 开始。视图无关可再定位对象文件 115 能够从必须由处理器执行的典型软件应用程序中获得。可再定位对象文件 115 在压缩之后理想地确切容纳在所选程序存储器中。对象文件 115 是链接步骤的结果，其中在链接步骤中程序所需的所有模块被合并。文件 115 包含针对跳转目标和数据对象的符号。视图生成工具 VG 应当将符号作为单独值进行处理。应当假设两个不同的符号总是指代不同的值，虽然这或许是不对的。在一个实施例中，视图生成工具 VG 能够执行预再定位，以便识别具有相同值的符号。潜在地，这降低了表中的条目的数量，从而改善了压缩因子。

[0108] 在所示的实施例中，另外地，还提供了处理器描述文件 105。处理器描述文件 105 便于降低用于搜索最佳指令精简方案集的搜索空间，因为其提供了关于能够被单独精简的指令如何被划分成指令字段的信息。处理器的处理器描述文件 105 由第一架构参数提取(APEX)模块 120 转换成时间静止指令格式(TSIF)数据结构 125。APEX 模块 120 提供用于提取在处理器的硬件描述中定义的参数的应用编程接口(API)。当使用硬件构件块库来构建处理器的硬件时使用该 API。TSIF 数据结构 125 和视图无关可再定位对象文件 115 被提供给视图生成模块 130，这将在下面更详细描述。视图生成模块 130 生成视图定义文件 135。

[0109] 视图生成模块 130 的实施例的典型实现方式在下面的伪代码中示出。在该实施例中，唯一地根据可再定位对象文件 115 来确定指令精简方案集。

[0110]

```
minimal_view(instructions) // 返回针对给定指令的最小视图
size(view)     // 用指令比特数量返回视图的大小
Instructions = all program instructions
RemovedInstructions = empty // 被移除指令集
view[ ] // 视图阵列
goal_view_size[ ] // 指示每个视图的期望大小
for v in all_views {
        view[v] = minimal_view(Instructions)
        while (size(view[v]) < goal_view_size[v]) {
        remove_instr = find_most_expensive(Instructions)
  Instructions = remove_one(Instructions, remove_instr)
  RemovedInstructions = add_one(RemovedInstructions, remove_instr)
        view[v] = minimal_view(Instructions)
 }
 Instructions = RemovedInstructions
    }
```

[0111] 视图定义文件 135 优选包括下面的信息。

[0112]

[0113]

Number of views
Number of instruction fields
Full instruction width (number of bits)
Per instruction field
    width (number of bits)
Per view
    instruction width (number of bits)
    per instruction field
        width (number of bits)
        bit indices of the slice in the compacted instruction
        implement table (yes/no)
if yes, table size (number of entries)
NOP value (registers can be saved by making the NOP value Read-Only)

[0114] 图 5 所示的视图生成工具 VG 定义了用于可编程处理器的视图，从而对于意欲在该可编程处理器上运行的代表性程序而言能够达到高压缩因子。用于解精简的硬件应当是不太昂贵的，即没有过多的门数量和 / 或时序图。

[0115] 视图生成工具生成指令精简方案集。在下面的描述中，假设每个指令精简方案对应于处理器的特定视图。可替换地，可以独立于处理器的视图来确定指令精简方案。

[0116] 工具 VG 使用由用户提供的软件内核 115 来生成视图。假定该软件内核代表处理器的（未来）使用。其应当足够大，而且其应当处理处理器的所有方面。例如，如果处理器包含定制操作，则推荐提供用于实际使用这些定制操作的内核。用户能够通过指示视图的数量和每个视图的压缩来调节（tweak）视图生成过程。视图生成工具 VG 的目的在于生成针对运行所提供的软件内核的处理器的最佳视图。最佳视图被认为是提供最高压缩因子的视图。通过采用针对跳转目标的符号，内核应当被提供为例如 ELF（执行语言格式）的单个文件。

[0117] 工具 VG 读取所有指令并存储每个单独的指令字段。之后其确定所谓的最小视图。该最小视图被定义为能够在其中运行所有程序指令的最小视图。通常，最小视图的宽度小于原始全视图的宽度。最小视图由针对每个指令字段的具有一定数量的条目的表构成。log2 个条目等于那个指令字段的精简宽度。精简后指令字段的宽度的和等于精简后指令的宽度。现在，用户能够通过定义压缩因子来对视图的大小设置约束。在给定该压缩因子和指令集的情况下，视图生成工具 VG 的目的是用于定义最佳视图。最佳视图被定义为来自供应集的最大数量的指令能够被映射于其上的视图。

[0118] 在创建了最小视图之后，移除供应集中的一个指令并再次创建最小视图。之后，移除下一指令，并再次再创建最小视图。指令的移除会导致更小的最小视图。移除指令并再次计算最小视图的过程继续，直到最小视图大小达到用户所指示的目标。之后，可以从没有位

于第一视图的指令开始来生成下一视图。该过程的主要问题是：“哪个指令是移除候选？”。

[0119] 可以通过计算每个指令的成本标准并选择具有最高移除成本的一个指令来回答该问题。由于算法的目的是降低最小视图的比特的数量，所以也用比特数量来表示成本。最小视图所需的比特的数量是位于该视图上的所有指令的结果。这些指令具有用于产生表中的不同条目的不同的指令字段值。表中不同条目的数量定义了比特的数量。现在，依赖于指令的指令字段，相对于所有其他指令的指令字段，指令消耗或多或少。

[0120] 根据定义，所有指令的成本等于最小视图所需的比特的数量。单个指令的成本等于其指令字段的成本的和。考虑所有指令中的字段值，指令字段的成本依赖于指令字段的值的频率。如果字段值稀有(频率低)，则成本高，因为该字段的表条目被很少的指令使用。如果字段值常见(频率高)，则成本低，因为许多其他指令使用同一表条目。这两种情况中一个表条目的成本相等，但是对于常见值，该成本与许多指令共享，而在稀有值的情况中，该成本与低数量的指令共享，从而对于每个指令而言，该成本更高。

[0121] 根据定义，所有指令中一个指令字段的成本等于指令字段的宽度。如果一个指令字段值的成本依赖于频率的倒数(reciproke)，则必须用乘数因子进行补偿，该乘数因子对于所有指令上的一个指令字段而言是恒定的。

[0122] 在公式形式中：

[0123] ND_{if} - 所有指令中指令字段 if 的不同值的数量。

[0124] f_{ifv} - 所有指令中指令字段值 ifv 的频率。

[0125] b_{if} - 指令字段 if 的比特数量。

[0126] $cost_{ifv}$- 指令字段值 ifv 的成本。

[0127] $$Cost_{ifv} = \frac{b_{if}}{f_{ifv} \cdot ND_{if}}$$

[0128] 用下面的表 1A、1B 中的示例来解释指令字段值成本函数。

[0129] 表 1A：指令字段值成本　　表 1B：指令字段值成本

[0130]

Instr_nr	Instr_fld_vl	成本	频率 1
0	0	0.15	5
1	1	0.75	1
2	0	0.15	5
3	5	0.375	2
4	0	0.15	5
5	2	0.25	3
6	5	0.375	2
7	0	0.15	5
8	2	0.25	3
9	2	0.25	3
10	0	0.15	5
总指令字段成本		3	

表条目	频率 2
0	5
1	1
2	3
3	0
4	0
5	2
6	0
7	0

[0131] 该示例基于由 11 个指令(instr_nr)构成的小程序。第一个表 1A 示出了针对这 11 个指令的指令字段值。第二个表 1B 指示在列“表条目”中所指示的哪些指令字段值(instr_fld_vl)将被存储在该表中。第二个表还示出了指令字段值的频率(频率 2),即在该程序中多长时间使用一次该条目。第一个表中的频率列(频率 1)是在第二个表中的列“表条目”中查找指令字段值(instr_fld_vl)的频率(频率 2)的结果。通过采用频率(频率 2),能够计算不同条目的数量和字段宽度、每个指令字段的成本,如在第一个表的列“成本”中示出的那样。在这种情况中,至少出现 1 次的不同条目的数量 ND 是 4。字段宽度即字段的比特的数量是 3,因为存在着 8 个可能的指令字段值。第一个表的最后一行示出所有指令字段成本的和。根据定义,这等于字段宽度。

[0132] 如所预期的,具有低频率的指令字段值比具有高频率的值对成本的贡献更大。这两个表还示出每个表条目的成本相同。表条目 5 出现了 2 次,即出现在指令 3 和指令 6 中。两者的成本都等于 0.375,因此该条目的成本等于 2*0.375=0.75。表条目 2 出现了 3 次,即出现在指令 5、8 和 9 中。每个指令的成本是 0.25,从而该条目的成本是 3*0.25=0.75,与条目 5 的成本相同。

[0133] 根据本发明,没有在基本块级上执行视图选择,而是在指令级上执行视图选择。因此,可以单独地为每个指令选择下一视图。通过引入基于指令的视图选择,用于生成视图的搜索空间变得更宽。其中,编程者之前应当对于在哪个视图上编程什么内容有清楚的认识,而现在编程者不再受视图的困扰。这将处理器设计者从生成“逻辑”视图的任务中解脱出

来。替代地，能够通过根据本发明第一方面的方法来自动执行视图的生成。

[0134] 在已经生成指令精简方案集之后，针对可编程处理器的程序能够如图 6 所示那样进行精简。

[0135] 通常，程序由不止一个模块组成。对于每个模块而言，汇编(.s)文件 165a,...,165n 由调度器生成。这些汇编文件被汇编器 170 转换成 ELF 格式的可再定位对象文件 175a,...,175n。汇编器 170 需要用于定义指令的指令字段值的处理器描述 105 和 APEX TSIF 数据结构 125。汇编器 170 的输出包含针对除了立即字段之外的所有指令字段的固定值。立即字段可以包含固定值以及符号。这些符号可以称为分支目标或数据对象。

[0136] 链接器 180 将由汇编器生成的可再定位对象文件 175a,...,175n 合并成一个可再定位对象文件 185。立即字段的可再定位符号仍为符号，仅这些符号的定义可以被修改。

[0137] 精简工具 190 将可再定位对象文件 185 形式的未精简程序转换成精简程序。未精简程序 185 以 ELF 可再定位对象文件的形式进入精简工具 190。精简工具 190 通过 APEX 获得视图定义和全指令格式。APEX 从处理器描述文件 105 收集信息，并形成视图描述文件。类似于视图生成工具 VG，精简工具 190 应当将可再定位符号作为单独值进行处理。在一个实施例中，假设互不同的符号总是指代互不同的值。在优选实施例中，精简工具 190 应用预再定位，以便识别具有相同值的符号。通常，这将改善程序的精简因子。

[0138] 在精简程序时，精简工具 190 将符号放置在表中，好像它们是标准值一样。工具 190 应当对此进行支持。精简过程以两种结果结束：

[0139] 1、精简后程序 195 包含视图 id 和每个指令的(精简后)指令值。

[0140] 2、针对该程序的视图表内容 197。

[0141] 视图表 197 可以包含可再定位符号，而精简后程序 195 则不能。

[0142] 精简后程序 195 被放置在程序存储器映射中。它们的示例随后给出。结果应当被转换成 ELF 对象文件。视图表内容还应当被放置在该对象文件中。

[0143] 链接器 200 执行表内容中的符号的再定位，并将该对象文件传递给二进制表示 205。

[0144] 为了便于可编程处理器识别根据其精简指令的视图，精简后指令数据优选包括视图识别数据。下面描述两个示例。

[0145] 在第一实施例中，向程序存储器字的每个分段中添加额外比特。分段具有最小视图的大小。将被添加的比特的数量依赖于具有相同大小的视图的数量。一个比特(S)指示该分段是否是指令的开始(S=1)。能够基于一系列分段的开始比特来确定精简指令的长度。如果具有相同大小的多个视图被用于软件的精简，则可以添加额外比特来识别正确的视图 id。假设处理器具有大小为 1/8 全视图大小的最小视图，而且最大的两个视图具有相同的大小。则每个分段需要添加 2 个比特。所需要的总的程序存储器比特数量等于 PMsize*8*2=16*PMsize，其中 PMsize 是程序存储器中的存储器字的数量。当视图信息现在在指令中可用时，程序计数器不再需要包括视图信息。替代地，程序计数器等于程序存储器中的开始分段地址。

[0146] 这参照下面的表而被示出。在其中所示的示例中，最小视图具有 1/4 的压缩因子，而最大的两个视图具有相同的大小。而且，在这里，每个分段被添加了两个比特 S、V。表 2 示出针对 11 个指令的序列的一部分的指令信息。第一列示出序列中指令的位置，第二列示

出与指令相对应的视图，第三列示出指令的长度，而第四列示出视图比特。在该示例中，处理器促成包括全视图在内的 6 个视图（0，...，5）。视图 3 和 4 两者都具有相同的长度（3 个分段），视图 1 和 5 也具有相同的长度（1 个分段）。指令的长度可以从开始分段比特获得，对于视图 0 和 2 而言，该长度直接确定了视图。对于该长度之后的视图 1、3、4 和 5，需要视图比特来确定视图。

[0147] 表 2 ：指令信息

[0148]

指令	视图	长度	视图比特
0	1	1	0
1	3	3	0
2	4	3	1
3	2	2	0
4	5	1	1
9	0	4	0
10	0	4	0

[0149] 表 3 示出如何在程序存储器中布置指令。每个分段具有开始分段比特 S，用于指示该分段是否是新指令的开始（1）或者该分段是否包含其开始位于之前分段处的指令的一部分。开始分段比特之后，每个分段具有视图比特 V，用于在指令长度不是总提供指令的视图的情况下指示指令的视图。注意，仅对于开始分段而言才需要该视图比特。对于长于一个分段的指令，可以通过每个分段的视图比特来划分视图信息。如果两个以上的视图具有相同的长度，这会是有用的。

[0150] 表 3 ：程序存储器映射

[0151]

地址	分段 3			分段 2			分段 1			分段 0		
	ID	V	S	ID	V	S	ID	V	S	ID	V	S
0	1	0	0	1	0	0	1	0	1	0	0	1
1	3	0	1	2	1	0	2	1	0	2	1	1
2	...	..	...	...	...	...	4	1	1	3	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
7	9	0	0	9	0	0	9	0	0	9	0	1
8	10	0	0	10	0	0	10	0	0	10	0	1

[0152] 图 7 示意性地示出了根据本发明的可编程处理器。图 7 所示的可编程处理器包括用于存储如上所述的指令序列的程序存储器 10。精简后指令数据至少包括作为具有 N 个

存储器字分段的第一码字的第一指令和作为具有 M 个存储器字分段的第二码字的第二指令，其中第一指令根据第一指令精简方案进行精简，第二指令根据第二指令精简方案进行精简，N 和 M 是大于或等于 1 的互不相同的整数。在这种情况中，精简后指令数据包括根据 6 种不同的格式进行精简的指令。

[0153] 可编程处理器还包括指令解码器 20、至少一个寄存器文件 40 和耦合到寄存器文件 40 的至少一个发布槽 50。另外，可编程处理器包括数据选择元件 30。数据选择元件 30、寄存器文件 40 和至少一个发布槽 50 由指令解码器 20 控制。可编程处理器可以包括更多发布槽，并且每个发布槽可以包括多个功能单元。还可以存在其他的数据处理元件，类似于存储元件、处理元件和选择元件。

[0154] 可编程处理器还包括指令扩展器 80。指令扩展器 80 包括精简方案识别器 17，用于识别哪个指令精简方案被用于从程序存储器 10 中获取的每个精简后指令数据。指令扩展器 80 具有用于接收由指令解码器 20 生成的程序计数器 PC 的输入。可编程处理器还包括存储设备 14 和选择设备（多路复用器单元）27，存储设备 14 用于临时存储至少程序存储器字的分段，而选择设备 27 用于从程序存储器 10 和存储设备 14 中选择精简后指令数据。所选择的精简后指令由指令扩展单元 87 扩展。指令扩展器 80 还包括控制设备 85，该控制设备 85 用于响应于程序计数器 PC 来生成用于程序存储器 10 的地址 AD。控制设备 85 还用信号 Sel 来控制选择设备 27 以及用读取使能信号 Ren 来控制存储设备 14。

[0155] 在根据本发明的可编程处理器中，优选地，针对每个视图的任何（精简）指令能够在存储器字中的任何分段处开始。这通过布置在程序存储器与指令扩展单元 87 之间的存储元件 14 来实现。在这种方式中，支持被存储在两个顺序存储器字上的指令。

[0156] 图 7 中的包括指令扩展器 80 的可编程处理器的第一实施例的一部分在图 8 中更详细地示出。

[0157] 在第一实施例中，程序存储器 10 分别在输出 P0、P1、P2 和 P3 处为分段 0、分段 1、分段 2 和分段 3 提供程序数据。存储元件 14 将由程序存储器 10 提供的程序数据的一部分延迟。在这种情况中，存储元件 14 分别提供来自程序存储器 10 的输出 P3、P2 和 P1 的延迟后程序数据 R2、R1、R0。多路复用单元 27 可控制地选择由程序存储器 10 和存储元件 14 提供的程序数据的一部分。在所示的该实施例中，多路复用器单元 27 包括多个多路复用器模块 27a、27b、27c、27d。多路复用器模块 27a、27b、27c、27d 分别由选择信号 sel0、sel1、sel2、sel3 控制。每个多路复用器模块 27a、27b、27c、27d 耦合到程序存储器 10 的每个输出 P0、P1、P2、P3。第一多路复用器模块 27a 附加地耦合到存储元件 14 的输出 R0、R1、R2。第二多路复用器模块 27b 附加地耦合到存储元件 14 的输出 R1 和 R2。第三多路复用器模块 27c 附加地耦合到存储元件 14 的输出 R2。

[0158] 在所示的实施例中，精简方案识别器 17 直接耦合到程序存储器 10。

[0159] 指令扩展单元 87 的输入由包含（精简后）指令的一个或多个分段形成。该指令被存储在程序存储器 10 中的某个位置处。程序存储器字包含 n 个（这里为 4 个）分段。指令长可以是一个或多个分段，而且一个指令的分段可以位于不同的（但随后的）程序存储器字中。多路复用器单元 27 允许指令扩展单元 87 从程序存储器字中直接读取（其间没有寄存器）。为了支持跨越指令的程序存储器字，寄存器 14 可以存储之前的存储器字。由于最大指令长度是 n 个分段，所以仅需要存储 n-1 个分段。

[0160] 如图 9 所示，输出 O0...On-2 连接到指令扩展单元 87 中的各种指令扩展模块 871-878。O0 被所有指令扩展模块使用，O1...On-2 仅在视图大小需要它时才被连接。On-1 仅用于未精简指令。

[0161] 例如如图 10 中的指令扩展模块 873 所示，指令扩展模块 871-878 中的每个指令扩展模块包括多个指令解精简分段 873a，...，873f。其中，解精简模块 873 包括用于将精简后操作码 opcode-s 转换成解精简后操作码 opcode-w 的操作码解精简分段 873a、用于将精简后立即 immediate-s 转换成解精简后 immediate-s 的立即解精简分段 873b、用于将精简后写端口选择 wp_select-s 转换成解精简后 wp_select-w 的写端口选择解精简分段 873c、用于将精简后总线选择“bus_select-s”解精简成解精简后“bus_select-w”的总线选择解精简分段 873d、用于将精简后写端口索引“wp_index-s”解精简成解精简后“wp_index-w”的写端口索引解精简分段 873e 以及用于将精简后读端口索引“rp_index-s”解精简成解精简后“rp_index-w”的读端口索引解精简分段 873f。

[0162] 操作码解精简分段 873a、总线选择解精简分段 873d 和写端口选择解精简分段 873e 例如通过使用查找表（LUT）来实施。写端口索引解精简分段 873e 和读端口索引解精简分段 873f 可以通过用于在指令字段的最高有效侧处添加一个或多个 0 比特的零扩展分段来实施。立即解精简分段 873b 依赖于针对指令的操作码来执行记号（sign）扩展或零扩展。

[0163] 寄存器 14 中的寄存器分段 Rn-2...R0 仅存储在两个存储器字上进行划分的指令的 lsb 分段。

[0164] 由于根据最小视图的指令的大小等于分段的大小，所以这些指令总是被完全存储在一个单个的程序存储器字中。因此，用于最小视图的多路复用器模块 27d 仅具有直接耦合到程序存储器 10 的输出的输入。针对其指令大小为两个或更多分段的视图的多路复用器 27a、27b、27c 也具有源自寄存器 14 的输入。这是因为这样的事实：这些指令能够在两个程序存储器字上进行划分。多路复用器的寄存器分段输入的数量等于指令分段的数量减一。

[0165] 多路复用器模块 27a、27b、27c、27d 的由控制设备 85 提供的选择信号 sel0，...，seln 依赖于它们的之前值以及之前指令的程序存储器使能信号，即程序存储器输出是新的还是之前已经被使用。指令的长度确定输出 O0，...，On 中的哪些输出被实际使用。该实施例的有利方面在于能够在指令被使用之前的周期中预计算选择信号。在该预计算周期中，能够从寄存器 14 中读取之前值，而且程序存储器使能信号等于用于控制程序存储器 10 的实际信号。如果当前指令表现为跳转指令，则针对下一指令的 PC 的 LSB 能够被认为是选择器的之前值。预计算使得能够使用针对 sel0，...seln 信号的流水线寄存器，以便指令选择的时序将改善。

[0166] 如图 11 的流程图所示，可以从程序存储器 10 中获取指令。

[0167] 在第一步骤 S1 中，将存储器地址 AD 初始化为例如值 0。在第二步骤 S2 中，在这种情况中，将分段计数器 SG 初始化为 0。之后，在第三步骤 S3 中，初始化指令开始地址 BG。指令开始地址由存储器地址分量 AD 和分段分量 SG 构成。在每个存储器字包括 n 个分段的情况中，则将指令开始地址计算为 BG=n*AD+SG。在步骤 S4 中，验证值 SG 是否等于最大值，例如 n-1。如果等于，则在步骤 S5 中将存储器地址 AD 增加 1，并在步骤 S6 中将分段数复位成

0。否则，在步骤 S7 中将分段数增加 1。在步骤 S6 或步骤 S7 之后，无论什么是可应用的，都在步骤 S8 中验证分段比特 S 以确定该分段是否从新的精简后指令开始。如果否，则指令流转到步骤 S4。否则，在步骤 S9 中根据 L=n*AD+SG - BG 来计算之前指令的长度，并且在步骤 S10 中，再次根据 BG=n*AD+SG 来计算 BG 的值。之后，程序流转到步骤 S4。根据长度 L 以及如果必要的话一个或多个附加视图比特 V，确定应当根据哪个视图来精简指令并因此确定哪个视图解码器是可应用的。

[0168] 下面描述如何根据上述表 3 中的程序来产生定序。假设指令 4 包括到指令 9 的跳转。程序从 PC=0 开始。程序存储器地址 0 被读取，并且精简方案识别器 17 识别程序存储器字中的四个分段中的开始分段比特 S。根据这 4 个开始分段比特，精简方案识别器 17 确定该字包含两个指令，其中第一个指令(0)从分段 0 开始并具有 1 个分段的长度，而第二个指令(1)从分段 1 开始并具有 3 或 4 的长度。指令 0 被映射到视图 1，因为分段 0 的视图比特 V 是“0”。其能够通过使用视图 1 指令扩展单元进行解精简，并随后被执行。指令 1 需要被存储，以便获取下一程序存储器字。一旦地址为 1 的下一程序存储器字可用，则精简方案识别器 17 就能够看到该字的分段 0 包含为“1”的开始分段比特 S。这意味着指令 1 具有 3 个分段的长度，并且能够被解精简、解码和执行。另一分段开始比特指示指令 2 的长度为 3。指令 2 和 3 需要被存储，而且地址 2 处的程序存储器字能够被获取。一旦接收到该字，指令 2 就能够被解精简、解码和执行，程序存储器字中的指令 4 和其他信息将被存储并且下一程序存储器字被读取。当指令 4 中的到指令 9 的跳转被执行时，程序计数器被加载有指令 9 的开始分段地址。程序存储器地址被设定为地址 7，而且包含指令 9 的程序存储器字被读取。在下一周期中，表现为映射到全视图的指令 9 能够被执行。

[0169] 在原理上，用于识别代码精简方案的该解决方案并不引入额外的失速周期或额外的分支延迟。如果分支目标在两个程序存储器字上被划分，则会因两个程序存储器字必须被读取以便获取分支目标指令而引入失速周期。但是这与精简方案 / 视图信息被获取的方式不相关。预期该解决方案的关键路径是相当长的，因为确定视图信息和开始分段、选择正确的分段以选择指令和解精简的序列都必须在一个时钟周期的时间内被执行。潜在地，需要在该路径中插入额外的流水线级，以便实现时序目标。

[0170] 在另一实施例中，通过将下一视图 id (nxt_vw_ID)添加到每一(精简后)指令中来识别视图信息。表 4 和 5 中给出了一个示例。该字段可从每个指令的预定位置处获得，例如从每个指令的 lsb 侧获得，而不管指令的长度。该字段的宽度等于视图数量的 log2。当通过程序定序时，之前指令的下一视图 id 和当前指令的位置(其包含在当前 PC 中)确定下一指令在程序存储器中的位置。对于具有 8 个视图和最大压缩因子为 1/8 的处理器而言，如果下一视图 id 字段具有 3 比特的宽度就已经足够了。假设能够用因子 3(所有指令上的每个指令的压缩的平均)来精简程序，则每个程序存储器字的指令的平均数量共计 3 个。所需要的用于识别下一视图 id 的比特的总数量则共计 PMsize*3*3=9*PMsize 个。与使用开始分段比特的解决方案中的 16*PMsize 相比，这是相当小的。

[0171] 表 4 :指令信息

[0172]

指令	视图	长度	nxt_vw_ID
0	1	1	3
1	3	3	4
2	4	3	2
3	2	2	5
4	5	1	0
9	2	2	0
10	0	4	1

[0173] 表 5 :程序存储器映射

[0174]

地址	分段 3	分段 2	分段 1	分段 0
0	1	1	1	0 | 3
1	3 | 5	2	2	2 | 2
2	…	…	4 | 0	3
…	…	…	…	…
7	9 | 0	8	8	8
9	10	10	10 | 0	9
…	…	…	…	…

[0175] 表 4 示出示例性程序中的指令序列，以及表 5 示出现在如何在程序存储器中存储该示例性程序，如上面所讨论的，精简后指令现在包括下一视图 id 字段（nxt_vw_ID）。除了视图和长度列之外，指令表 4 还包含用于指示下一指令的视图 id 的额外的列。

[0176] 图 12 示出根据本发明的可编程处理器的第二实施例的一部分，其用于处理被存储在程序存储器中的如上面的表 5 中所示的精简后程序。

[0177] 第二实施例与第一实施例的不同之处在于，精简方案识别器 17 耦合到选择设备 27 的输出。

[0178] 如图 13 的流程图所示，可以从程序存储器 10 中获取指令。

[0179] 在第一步骤 S20 中，从程序存储器 10 中读取存储器字。在第二步骤 S21 中，精简方案识别器 17 识别用于下一指令的精简方案。假设第一指令的精简方案是已知的，而且其在存储器字中具有预定的对准。在第三步骤 S22 中从表中读取下一指令的长度，并在步骤 S23 中计算下一指令的结束地址。

[0180] 该结束地址由存储器地址分量 AD 和分段分量 SG 构成。在步骤 S24 中，确定该结束地址是否位于下一程序字中。如果否，则将地址计数器 AD 维持在当前值，并在步骤 S27

中将相关分段选为将由可编程处理器解精简和执行的指令的精简后数据。如果在步骤 S24 中确定当前精简后指令的结束地址实际上位于下一程序字中，则在临时存储设备中存储当前程序字中的当前精简后指令的分段，并递增地址 AD。

[0181] 程序计数器由分段地址、程序存储器地址和当前指令的视图 id 构成。在程序执行的开始时，PC 指向指令 0，而且精简方案识别器 17 从 PC 知道第一指令是视图 1 指令，即根据基于第一视图的精简方案而被精简的指令。硬件查找表指示视图 1 指令具有 1 个分段的长度。当程序存储器字在输出处可用时，则能够读取指令 0 的下一字段 id 字段，从而产生值 3。通过索引长度查找表，由精简方案识别器 17 确定被映射到视图 3 的下一指令具有 3 个分段的长度。该信息指示程序存储器 10 的读取能够保持一个时钟周期，因为指令 1 总是在当前程序存储器输出中完全可用。当扩展指令 1 时，程序存储器地址能够被设置成 1，以便读取包含指令 2 和一部分指令 3 的程序存储器字。当指令 2 被扩展并被进一步处理时，控制设备 85 将决定读取下一程序存储器地址，因为指令 2 的下一视图 id 字段指示视图 2 意味着长度为 2 的指令。开始分段位于分段 3 处，这暗示着该下一指令扩展到下一程序存储器字中。指令 4 包含到指令 9 的跳转，指令 9 位于程序存储器地址 7、分段 3 处。除了所述指令的地址之外，跳转指令还传递包括指令 9 的视图 id 2 的 PC。通过采用该信息，控制设备 85 确定两个程序存储器字需要首先被读取，以便解精简指令 9 以进行进一步的执行。当跳转到分支目标时，分支目标 9 在两个程序存储器字上被划分的事实导致失速周期。

[0182] 在条件分支的情况下，分支指令包括下一指令的视图 ID 的指示，而且另外地，所述分支地址包括分段地址、程序存储器地址和被有条件地分支到的地址处的指令的视图 id。

[0183] 如果希望，例如在关键环路中，能够通过将分支目标布置在程序存储器字的分段 0 处以便其永远不被划分在两个存储器字上来避免该失速周期。注意，这仅在分支目标指令被划分在两个程序存储器字上时才是必要的。如果目标不从分段 0 处开始，但是具有其仍然完全容纳在存储器字中的长度，则不需要替换它。

[0184] 将分支目标布置在分段 0 处移除了失速周期但引入了另一问题。在存储器映射中创建了间隙。当不是通过跳转而是通过根据之前指令的定序来达到分支目标时，不得不通过该间隙。提出了可以避免该问题的 5 种解决方案：

[0185] 1、预留下一视图 id 字段（nxt_vw_ID）的一个值，用于指示存储器间隙。如果指令的下一视图字段指示存储器间隙，则下一指令应当从下一程序存储器字处的分段 0 中读取。该指令的视图 id 由该间隙的第一分段的下一视图 id 字段指示。

[0186] 2、将最小视图的“不可能”指令值插入该间隙的第一分段。“不可能”指令值是不能由编译器生成的指令值。一个示例是具有“非 NOP”输入的 NOP 指令。可编程处理器具有用于监控该指令的出现的指令选择器。当被检测到时，其选择下一程序存储器字的分段 0 而非具有不可能指令的分段。分支目标的视图 id 由所添加的不可能指令的下一视图 id 识别。

[0187] 3、全视图的不能由编译器生成的“不可能”指令值（诸如具有“非 NOP”输入的 NOP 指令）如下那样被使用。该指令字段被调整，以便相关指令比特位于全指令的 lsb 分段。该间隙的第一分段用全指令的该 lsb 分段填充。该分段的下一视图 id 指示被对准的分支目标的视图 id。间隙之前的最后指令具有类似于全视图 id 的下一视图 id。基于该全视图 id，

控制设备 85 决定加载下一程序存储器字。当还检测到专用指令时，控制设备 85 决定跳转到下一被对准的分支目标。表 7 示出了该解决方案。在指令 4 之后，程序存储器包含两个分段的间隙。该间隙的第一分段用“不可能”指令（IMP）的 lsb 分段填充，该间隙的第二分段可以是任意值（X），因为其不被读取。

[0188] 表 6 ：指令信息

[0189]

指令	视图	长度	nxt_vw_ID
…	…	…	…
2	4	3	2
3	2	2	5
4	5	1	0
5	4	3	2
6	2	2	4
7	4	3	1
…	…	…	…

[0190] 表 7 ：程序存储器

[0191]

地址	分段 3	分段 2	分段 1	分段 0
…	…	…	…	…
10	3 | 5	2	2	2 | 2
11	X	IMP | 4	4 | 0	3
12	6 | 4	5	5	5 | 2
13	7	7	7 | 1	6
…	…	…	…	…
…	…	…	…	…

[0192] 4、所有关键分支目标被容纳在最小视图中。这可以通过用这些关键分支目标来开始压缩过程并用它们的指令字段值来填充表来实现。

[0193] 5、每个程序存储器字都预留额外比特。如果该比特是“1”，则程序存储器字的 msb 分段指示该间隙的开始点。该间隙之前的最后指令包括被对准的分支目标的视图 id。

[0194] 如表 8 中所示，所呈现的 5 个解决方案可以影响被执行的程序的时序、链接器的复杂度、程序的精简因子和硬件的复杂度和（关键）时序路径。本领域技术人员可以依赖于给予这些因子的权重来选择这些解决方案中的其中一个解决方案。下面的表中示意性地指示了对这些因子的影响。第一列指示如上所述的解决方案，第二列指示该解决方案的可行性。其中，符号“-”、“0”和“+”分别指示可能不可行、不总是可行和可行。第三列指示对精简因

子的影响。“-”指示精简因子被减小。“o”指示措施对精简因子没有实质性影响。最后一列指示就实现所需的门数量方面对硬件复杂度的影响，范围从用于指示相对复杂的硬件实现方式的“-”到用于指示具有相对低复杂度的硬件实现方式的“++”。一般而言，更复杂的硬件实现方式将暗示更长的组合路径，从而具有更慢的时序。

[0195] 表 8 :所呈现的解决方案之间的比较

[0196]

解决方案	可行性	精简	硬件
1	+	-	-
2	O	-	-
3	+	O	O
4	+	-	++
5	+	-	O

[0197] 解决方案 1 是简练的解决方案，但是字段 id 的成本将减小精简因子。解决方案 2 并非总是可行。专门的“不可能”指令在最小视图中可能是不可用的。除此之外，到程序存储器的时序路径使得信号相当长。解决方案 3 没有这些问题，因此是优选的。尽管如此，它的缺陷是指令字段需要被调整。解决方案 4 潜在地降低了精简因子，因为最小视图具有非最优的表条目。解决方案 5 产生小的精简因子降低，但是其并不优于解决方案 3。

[0198] 鉴于上述内容，在全视图中使用专门指令的解决方案 3 是优选的。

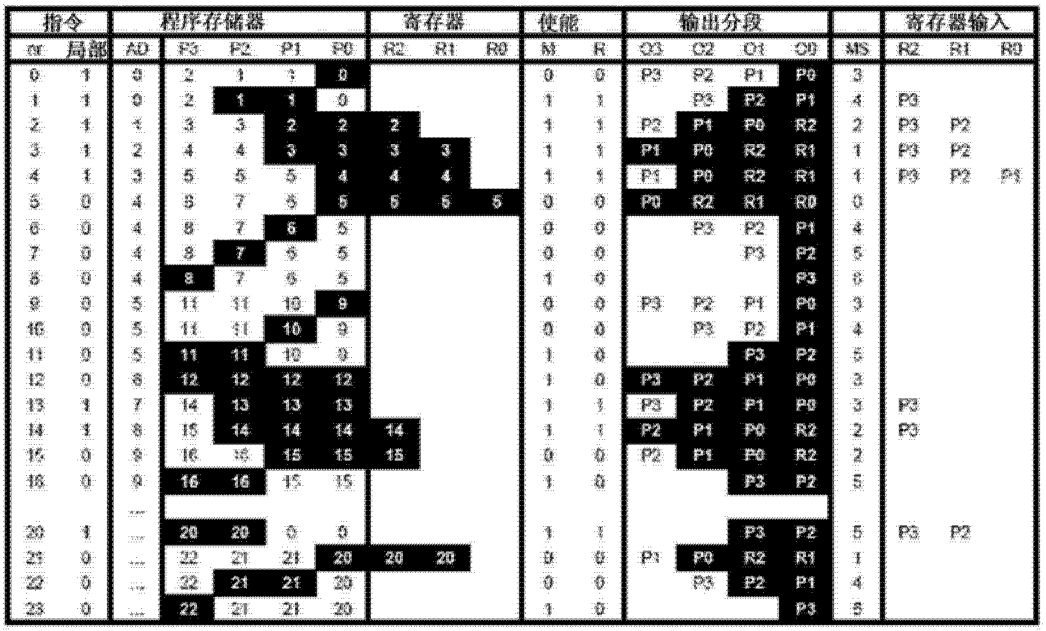
[0199] 对准所有分支目标将大大地降低程序的精简因子。为了避免该情况，应当在被对准的边界处仅放置时间关键的分支目标。通常，时间关键的分支目标是时间关键的环路的一部分。用户或调度器必须指示将其放置在哪里。汇编器必须在(ELF)对象文件中包括该信息，以便精简工具 190 (图 6)能够看到哪些指令应当被放置在被对准的地址处。

[0200] 下面的表 9 示出了针对图 12 的架构的程序执行的示例。在该示例中，程序存储器宽度等于 4 个分段。第一列(nr)指示哪个指令被解精简。如果程序存储器 10 的 msb 分段是在下一程序存储器字中继续的指令的一部分，则第二列(部分)包含“1”。列 P3 向下至 P0 指示程序存储器 10 的输出。每一列表示分段。数字指示指令数。精简后指令 0 表现为具有 1 个分段的宽度，指令 1 是 2 个分段宽等等。列 R2、R1 和 R0 表示存在于寄存器的输出处的指令(或它们的分段)的数量。列 M 和 R 指示程序存储器的地址 AD 是否被递增和 / 或寄存器是否需要被更新。输出分段列 O3..O0 指示从哪个存储器分段 P0,..,P3 或者从哪个寄存器分段 R0,..,R2 中读取指令分段，而且寄存器输入 R2、R1 和 R0 指示在寄存器 14 中存储从程序存储器 10 的哪一部分开始的相关数据。注意，寄存器分段 R0、R1、R2 总是分别从相同的程序存储器分段 P1、P2、P3 中获取它们的数据。列 MS 指示多路复用器 27a,..,27d 的选择值。由该选择值控制所有的多路复用器是有吸引力的。

[0201] 该示例示出寄存器 R0、R1 和 R2 仅在需要是被写入。列 O3,..O0 示出多路复用器总是选择值，即使该值不被使用。仅反向标记的条目才被指令扩展单元 87 实际使用。

[0202] 表 9 :

[0203]



指令		程序存储器					寄存器			使能		输出分段					寄存器输入		
nr	局部	AD	P3	P2	P1	P0	R2	R1	R0	M	R	O3	O2	O1	O0	MS	R2	R1	R0
0	1	0	2	1	1	0				0	0	P3	P2	P1	P0	3			
1	1	0	2	1	1	0				1	1		P3	P2	P1	4	P3		
2	1	1	3	3	2	2	2			1	1	P2	P1	P0	R2	2	P3	P2	
3	1	2	4	4	3	3	3	3		1	1	P1	P0	R2	R1	1	P3	P2	
4	1	3	5	5	5	4	4	4		1	1	P1	P0	R2	R1	1	P3	P2	P1
5	0	4	6	7	6	5	5	5	5	0	0	P0	R2	R1	R0	0			
6	0	4	8	7	6	5				0	0		P3	P2	P1	4			
7	0	4	8	7	6	5				0	0			P3	P2	5			
8	0	4	8	7	6	5				1	0				P3	6			
9	0	5	11	11	10	9				0	0	P3	P2	P1	P0	3			
10	0	5	11	11	10	9				0	0		P3	P2	P1	4			
11	0	5	11	11	10	9				1	0			P3	P2	5			
12	0	6	12	12	12	12				1	0	P3	P2	P1	P0	3			
13	1	7	14	13	13	13				1	1	P3	P2	P1	P0	3	P3		
14	1	8	15	14	14	14	14			1	1	P2	P1	P0	R2	2	P3		
15	0	9	16	16	15	15	15			0	0	P2	P1	P0	R2	2			
16	0	9	16	16	15	15				1	0			P3	P2	5			
		...																	
20	1	...	20	20	0	0				1	1			P3	P2	5	P3	P2	
21	0	...	22	21	21	20	20	20		0	0	P1	P0	R2	R1	1			
22	0	...	22	21	21	20				0	0		P3	P2	P1	4			
23	0	...	22	21	21	20				1	0				P3	5			

[0204] 指令 20 是被划分在两个存储器字上的跳转目标。这导致失速周期。

[0205] 图 14 示出了根据本发明的可编程处理器的第二实施例。该实施例与第一实施例的不同之处在于，寄存器 14 被移位寄存器 15 替换，该移位寄存器 15 经由输入多路复用器单元 16 耦合到程序存储器 10。在所示的该实施例中，输入多路复用器单元 16 包括两个多路复用器模块 16a、16b。在该图中，移位寄存器 15 还被标记为 S2、S1 和 S0。移位寄存器 15 的输入连接到两个输入多路复用器 16a、16b。移位寄存器 15 的一个输入由程序存储器 10 的分段 P3 的输出形成。其他输入连接到移位寄存器，也从 lsb 分段开始。为了避免额外的分支延迟，分支目标应当被放置在单个的程序存储器字中。当分支目标在程序存储器 10 的输出处可用时，其通过输出多路复用器单元 27 被馈送并被指令扩展单元 87 中的适当的指令扩展模块所扩展。之后程序存储器字的其余部分(排除刚被解码的指令)被移位到总地标记为 15 的移位寄存器(S2、S1 和 S0)中，以便下一指令从移位寄存器的 lsb (ls 分段)处开始。下一指令能够从寄存器 15 中读取，并从 lsb 侧开始。如果在移位寄存器 15 中指令仅部分可用，则剩余部分从程序存储器 10 的输出中读取。移位多路复用器 16a、16b 能够被再用于选择程序存储器的适当的分段。

[0206] 在该实施例中，指令获取序列基本上与可编程处理器的第一实施例中的序列相同。用于输出多路复用器的选择信号能够被流水线处理，以便缩短时序路径。

[0207] 下面的表 10 用于说明该设备的操作。

[0208] 第一列中的数字 nr 等于程序计数器值 PC。

[0209] 如果当前 PM 字的 P3 分段是在下一 PM 字中继续的指令的一部分，则第二列包含“1”。

[0210] 第三列指示被编址的程序存储器的存储器地址 AD。

[0211] P0、P1、P2、P3 指示被编址的存储器字的各个分段的指令 id。

[0212] S0、S1、S2 指示从其获取指令数据的程序存储器 10 的分段。

[0213] M 指示下一周期中是否递增存储器地址 AD。

[0214] R 指示寄存器 15 是否被使能以接受来自程序存储器 10 的数据。

[0215] O0,...,O3 分别指示程序存储器分段 P0、P1、P2、P3 和寄存器分段 S0、S1、S2 中的哪个在多路复用器模块 27a,...,27d 的输出处是可见的。

[0216] 在参照下面的表描述的该示例中，观察到下面的内容。

[0217] 在该表的第一行，指令扩展器 80 接收值为 0 的程序计数器，并假设该值为程序存储器地址。在所述存储器地址处，程序存储器在第一分段中包括针对第一指令字的精简后数据(0)，在第二和第三分段中包括针对第二指令字的精简后数据(1)以及在第四分段中包括针对第三指令字的精简后数据的一部分(2)。包括精简后第一指令字的一个分段 P0 被选择并经由输出 O0 被提供给指令扩展单元 87。由于在该情况中被编址的存储器字在分段 P1、P2 中包括完整的精简后第二指令，所以程序计数器 PC 的增加并不跟随有程序存储器地址 AD 的增加。替代地，包括第二指令(1)的两个分段 P1、P2 被选择并分别在输出 O0 和 O1 处被提供给指令扩展单元 87。在同一周期中，在寄存器 15 的分段 S0 中读取输出分段 P3。由于存储器地址 0 仅包括精简后的下一指令(2)的一部分，所以控制信号 M 现在允许存储器地址 AD 在下一周期中增加。因此，在下一周期中，存储器地址 1 被寻址，并且针对指令 2 的指令数据现在分别通过寄存器分段 S0 和程序存储器分段 P0、P1、P2 的选择而被提供在输出 O0、O1、O2、O3 处。在随后的周期中，每次在需要时获取和扩展精简后指令数据的一个或多个分段。

[0218] 表 10：

[0219]

指令		程序存储器					寄存器		使能		输出分段					寄存器输入		
nr	局部	AD	P3	P2	P1	P0	S2	S1	S0	M	R	O3	O2	O1	O0	MS	S2	S1	S0
0	1	0	2	1	1	0				0	0				P0				
1	1	0	2	1	1	0				1	1			P2	P1				P3
2	1	1	3	2	2	2			2	1	1	P2	P1	P0	S0				P3
3	1	2	4	4	3	3			3	1	1		P1	P0	S0			P3	P2
4	1	3	5	5	5	4		4	4	1	1		P0	S1	S0		P3	P2	P1
5	0	4	7	7	6	5	5	5	5	0	0	P0	S2	S1	S0				
6	0	4	7	7	6	5				0	0				P1				
7	0	4	7	7	8	5				1	0			P3	P2				
8	1	5	9	8	8	8				1	1		P2	P1	P0				P3
9	0	6	11	10	10	9			9	0	0			P0	S0				
10	0	6	11	10	10	9				0	0			P2	P1				
11	0	6	11	10	10	9				1	0				P3				
		...																	
20	1	...	20	20	20	0	0	0	0	1	1	0	0	0	0				P3
20	1	...	23	21	21	20	20	20	20	0	0	P0	S2	S1	S0				
21	1	...	23	21	21	20				0	0			P2	P1				
22	1	...	23	21	21	20				1	1				P3				P3
23	1	...	24	18	18	23			23	1	0			P0	S0				
24	0	...	26	26	25	25				0	0		P1	P0	P3				
25	0	...	26	26	26	26				0	0			P1	P0				
26	0	...	26	26	25	25				1	0			P3	P2				
34	1	...	0	0	0	0	0	0	0	0	1				P3				P3

[0220] 对于第一实施例(在图 12 中示出)，下一指令的开始分段能够根据当前指令的开始分段和当前指令的长度进行计算。根据该开始分段，能够计算针对 sel0 的一个热字。sel0 的 LSB 部分被复制到 sel1,...seln。注意，输出分段 O0..On 总是包含数据。指令扩展单元 87 必须考虑依赖于指令长度，仅必须读取 lsb 分段。

[0221] 对于第二实施例(图 14 中示出)，开始分段的计算相同。针对 o sel0 的从开始分段到一个热选择器的转换稍微不同，因为一个热字更小，但是针对 o sel1..o seln 的信号不是仅被移位的版本。S0..Sn 中可用分段的数量影响了 o sel 信号之间的关系。图 12 中所示的实施例使用 2n-1 比特的一个热字来控制多路复用器单元，而图 14 的实施例使用 n+1

比特的一个热字。然而，对于第二实施例，sh_sel 信号也必须被生成。这并非必须被注册，因为从程序存储器 10 到输入移位寄存器 Sn..S0 的时序路径足够短。能够根据下一开始分段生成选择信号 sh_sel0..sh_seln。

[0222] 可以通过使用硬件生成工具来自动地生成根据本发明的可编程处理器的规范。图 15 示出了这样的工具 PC。其中，常规处理器（例如，图 1 中所示的处理器）的视图定义文件 135 和处理器描述文件 105 被提供给第二 APEX 模块 140。该第二 APEX 模块 140 从处理器描述文件 105 和视图定义文件 135 中收集信息，并提供用于提取在处理器的描述中所定义的参数的 API 145。该 API 145 由用于生成规范 155 的硬件构件块库 150 使用。

[0223] 在根据本发明的可编程处理器中，指令或指令字段的解精简典型地通过索引可编程表（这里也称为解精简表）来执行。提供写和读设备，以用于写入和读取解精简表。在一个实施例中，针对解精简表的寄存器仅在条目被写入时才被时钟控制。通常，这在处理器已经被加电之后立即发生。因此，针对解精简表寄存器的时钟门控导致功率使用的明显减小。

[0224] 在一个实施例中，使用至少一个解精简表来根据多个视图来解精简指令。这将不需要位于解精简表上的多个读取端口，因为针对互不相同的精简方案的指令扩展模块并非必须被并行执行。

[0225] 解精简表将总是包括针对指令字段的 NOP 值。为 NOP 代码预留每个表的地址 0 并使其为固定条目而非由寄存器来实施该条目是有意义的。

[0226] 在权利要求中，措辞“包括”并不排除其他元件或步骤，而且不定冠词“一”或“一个”并不排除多个。单个部件或其他单元可以实现权利要求中所引用的若干项的功能。某些措施被引用在互不相同的权利要求中的事实并非指示这些措施的组合不能用于实现有益效果。权利要求中的任何参考标号不应当被解释为限制范围。另外，除非明确以相反地方式指出，否则“或者”指代包括性的“或”而非排他性的“或”。例如，下面各项中的任一项都满足“条件 A 或 B”：A 为真（或存在）而 B 为假（或不存在）；A 为假（或不存在）而 B 为真（或存在）；以及 A 和 B 两者都为真（或存在）。

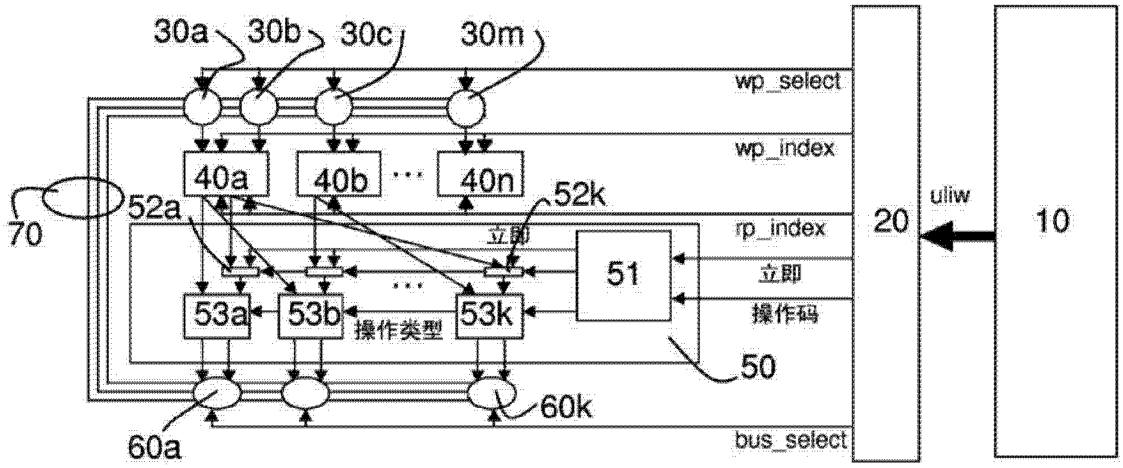

30a
30b
30c
30m
wp_select
wp_index
40a
40b
40n
52a
52k
70
rp_index
立即
51
立即
操作码
53a
53b
53k
操作类型
50
60k
60a
bus_select
20
uliw
10

图 1

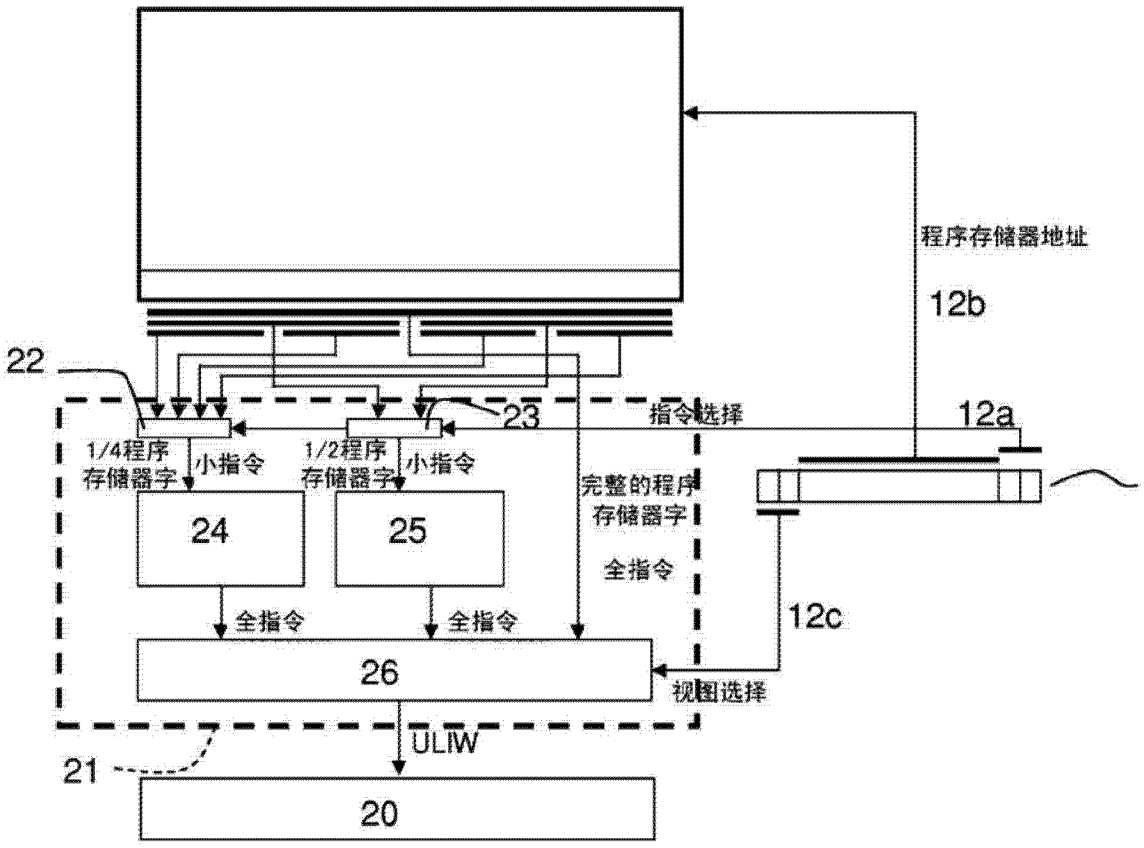

程序存储器地址
12b
22
23
指令选择
12a
1/4程序
存储器字
小指令
1/2程序
存储器字
小指令
完整的程序
存储器字
24
25
全指令
12c
全指令
全指令
26
视图选择
ULIW
21
20

图 2

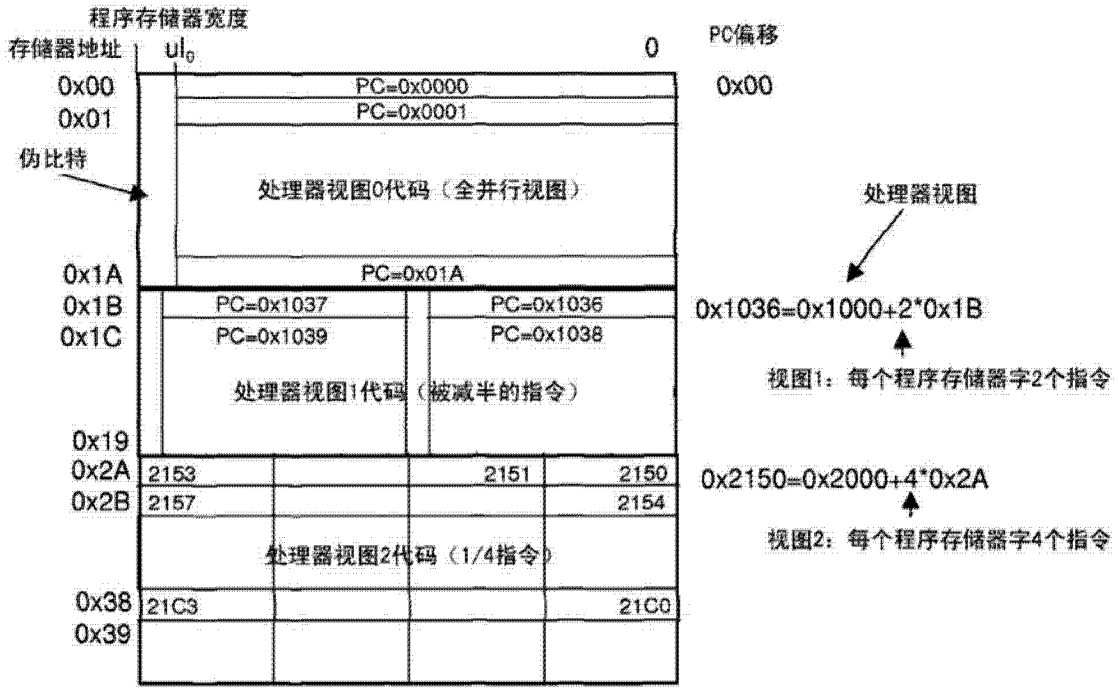

程序存储器宽度
存储器地址
0
PC偏移
0x00
0x01
伪比特
PC=0x0000
PC=0x0001
处理器视图0代码（全并行视图）
处理器视图
0x1A
PC=0x01A
0x1B
0x1C
PC=0x1037
PC=0x1036
PC=0x1039
PC=0x1038
处理器视图1代码（被减半的指令）
0x1036=0x1000+2*0x1B
视图1：每个程序存储器字2个指令
0x19
0x2A
0x2B
2153
2151
2150
2157
2154
处理器视图2代码（1/4指令）
0x2150=0x2000+4*0x2A
视图2：每个程序存储器字4个指令
0x38
0x39
21C3
21C0

图 3

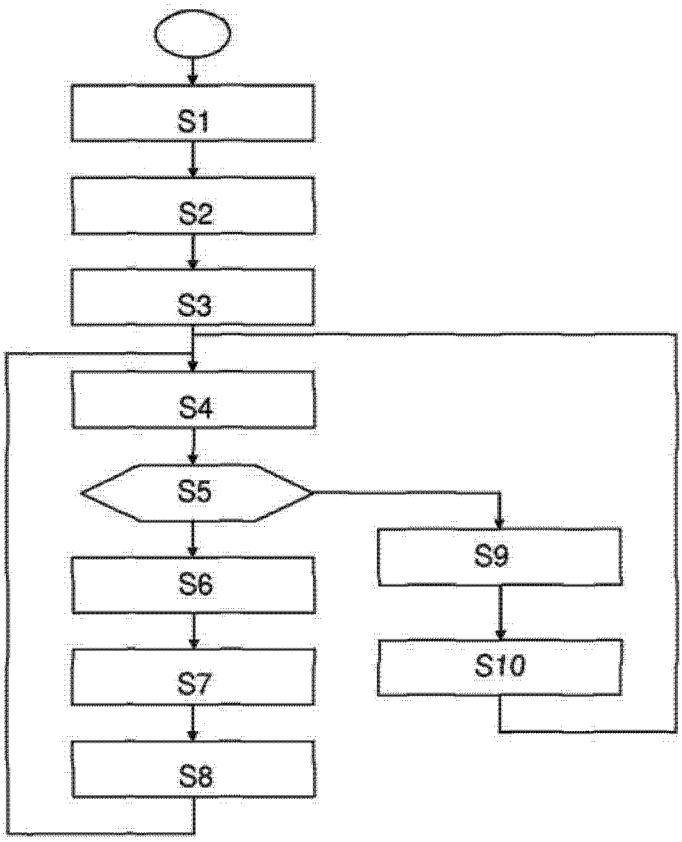

S1
S2
S3
S4
S5
S6
S7
S8
S9
S10

图 4

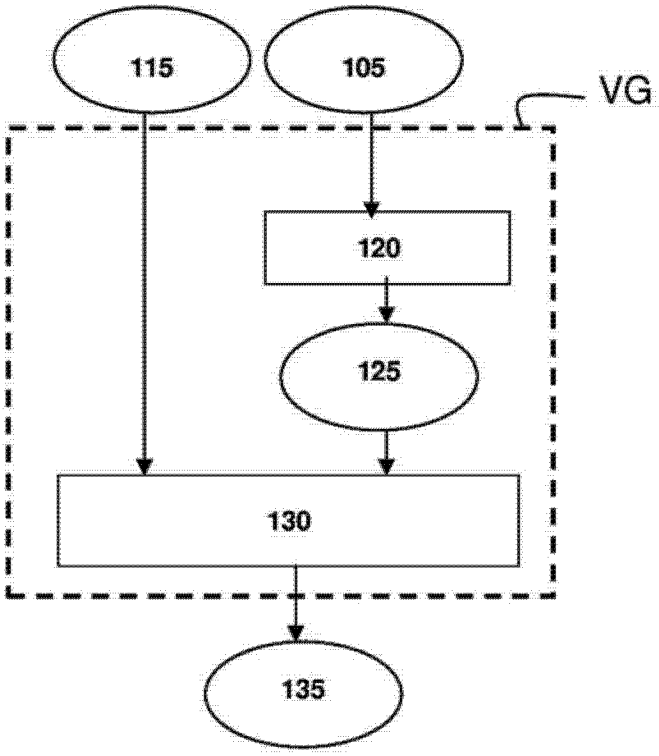


图 5

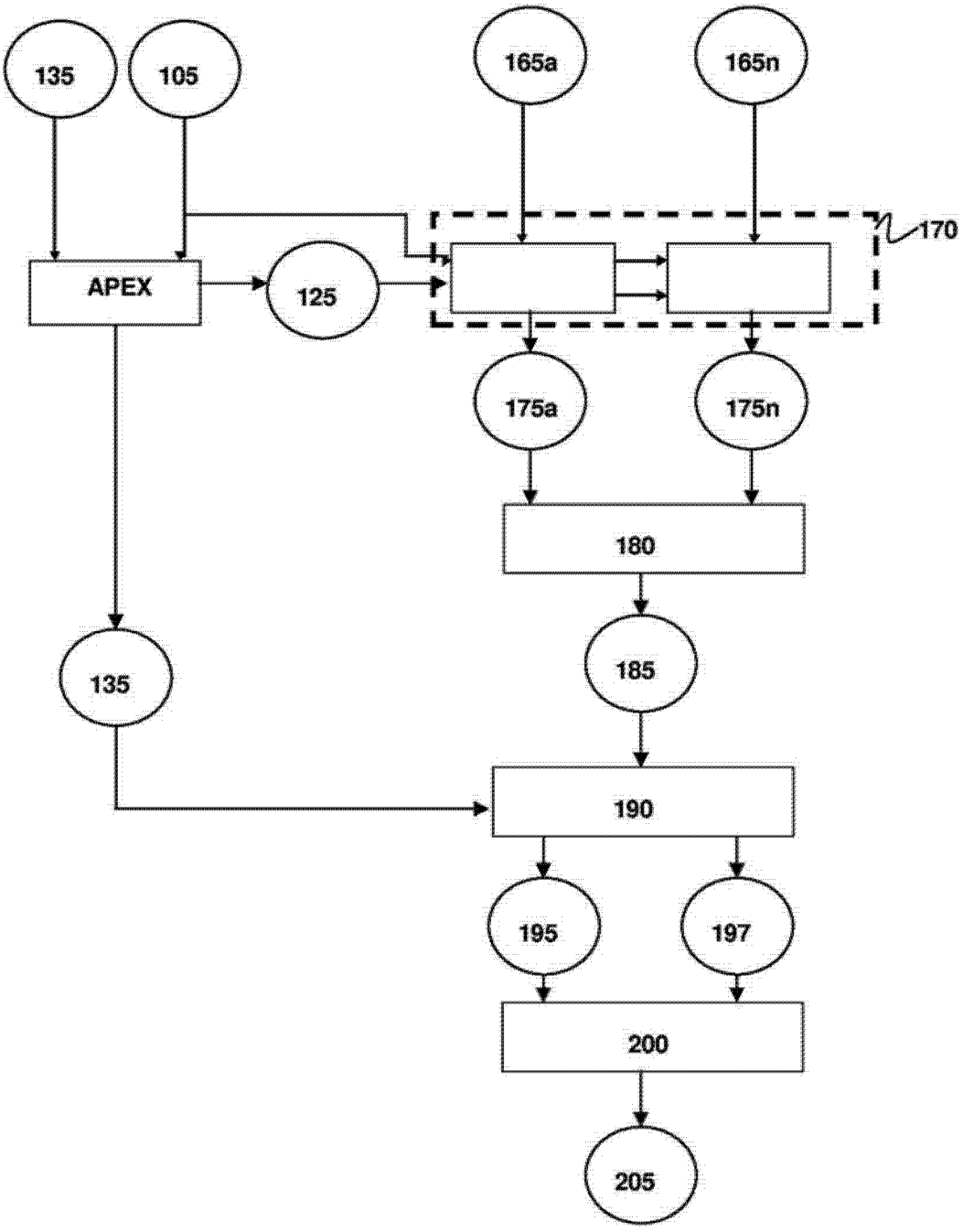

135
105
165a
165n
170
APEX
125
175a
175n
180
135
185
190
195
197
200
205

图 6

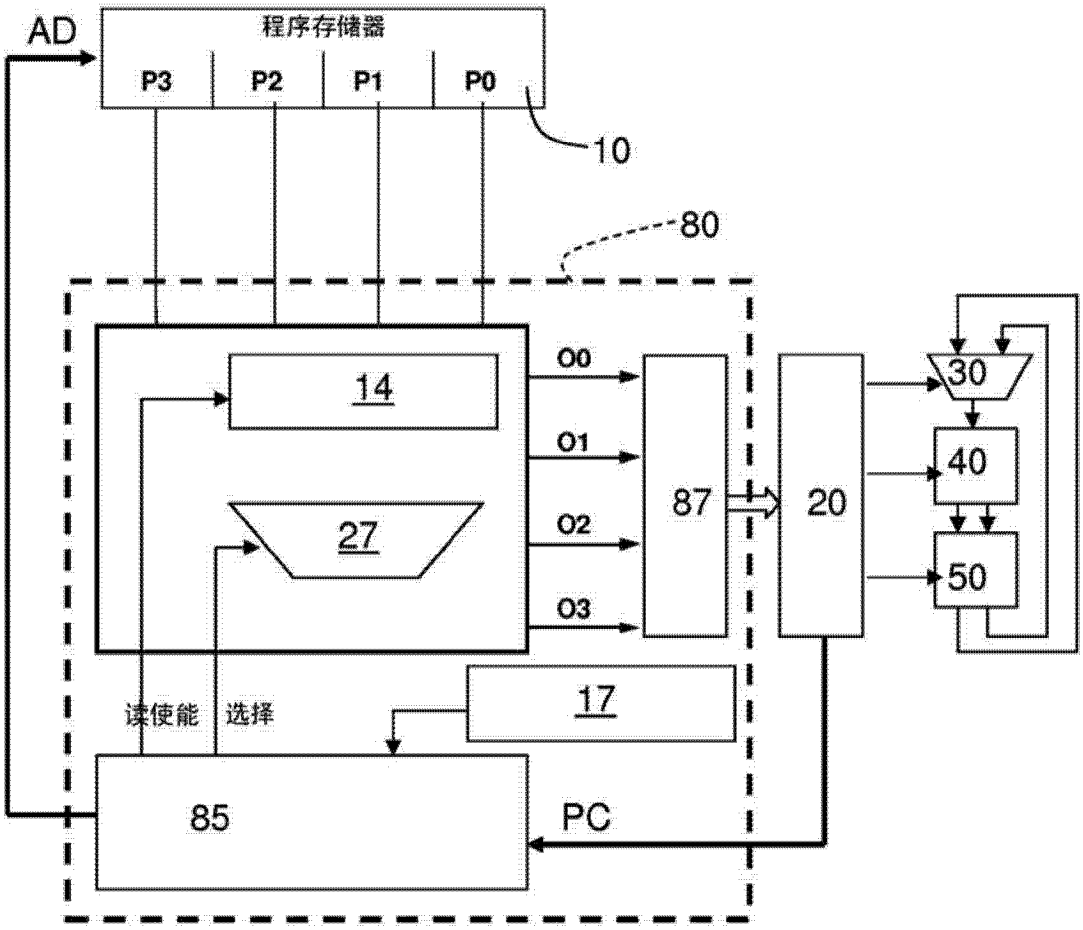

AD
程序存储器
P3
P2
P1
P0
10
80
14
27
O0
O1
O2
O3
87
20
30
40
50
读使能
选择
17
85
PC

图 7

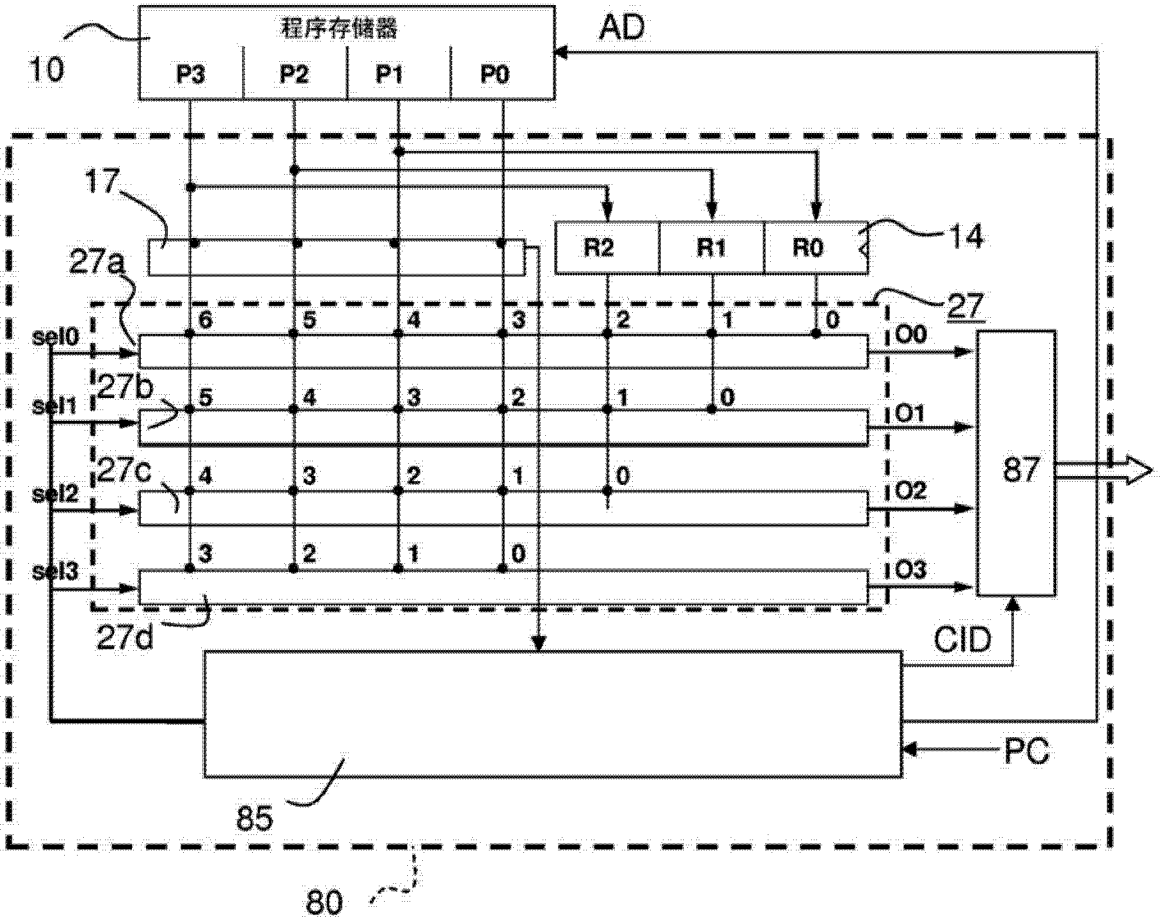

程序存储器
10
P3
P2
P1
P0
AD
17
27a
R2
R1
R0
14
27
6
5
4
3
2
1
0
O0
O1
O2
O3
sel0
sel1
sel2
sel3
27b
27c
27d
87
CID
PC
85
80

图 8

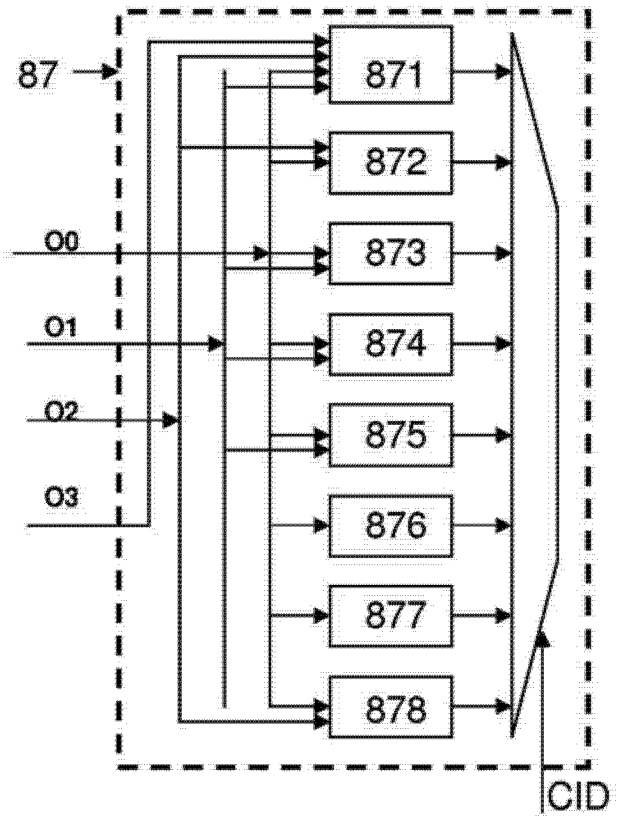


图 9

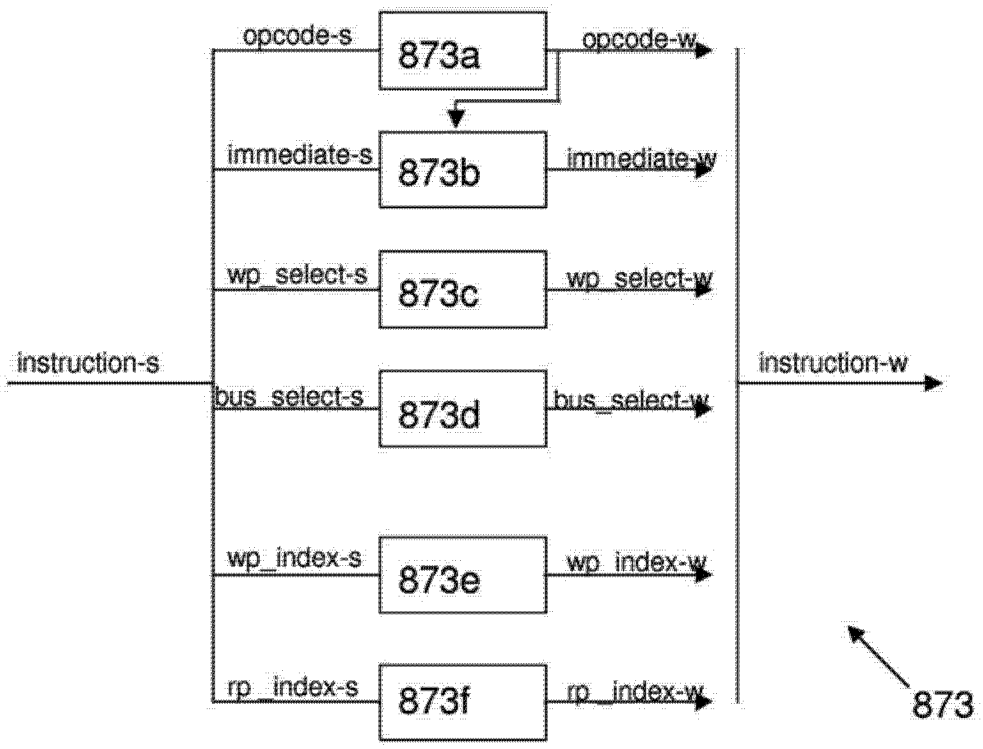

opcode-s
873a
opcode-w
immediate-s
873b
immediate-w
wp_select-s
873c
wp_select-w
instruction-s
bus_select-s
873d
bus_select-w
instruction-w
wp_index-s
873e
wp_index-w
rp_index-s
873f
rp_index-w
873

图 10

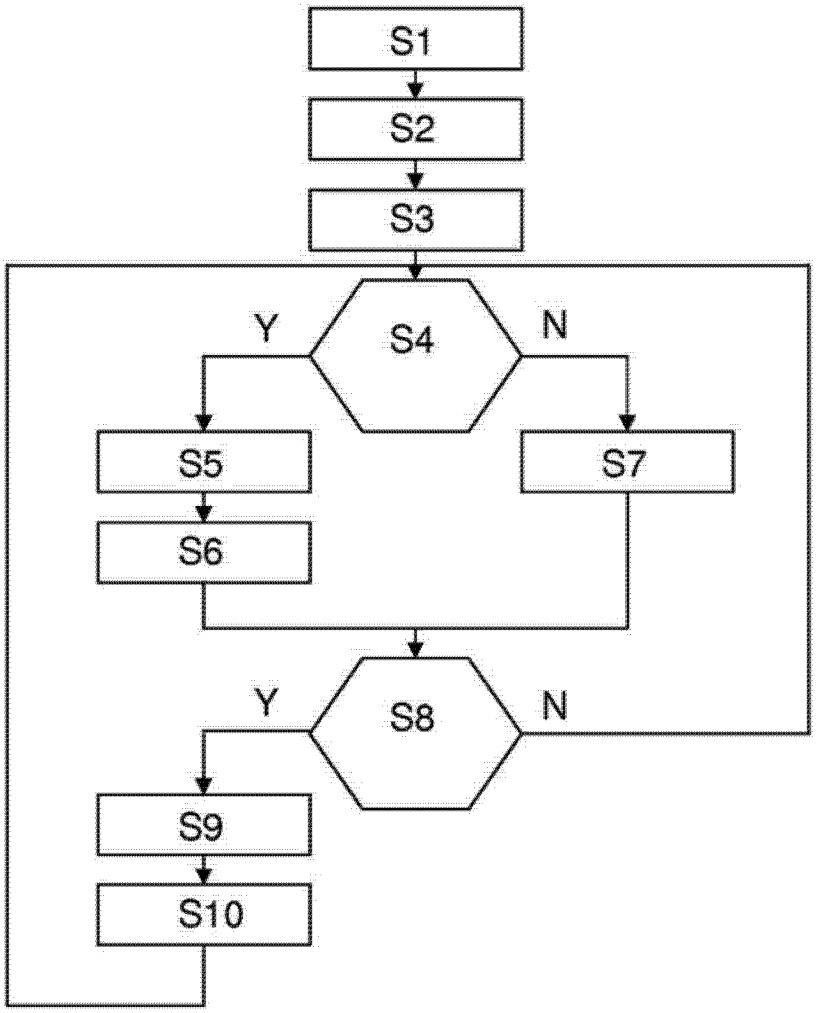


图 11

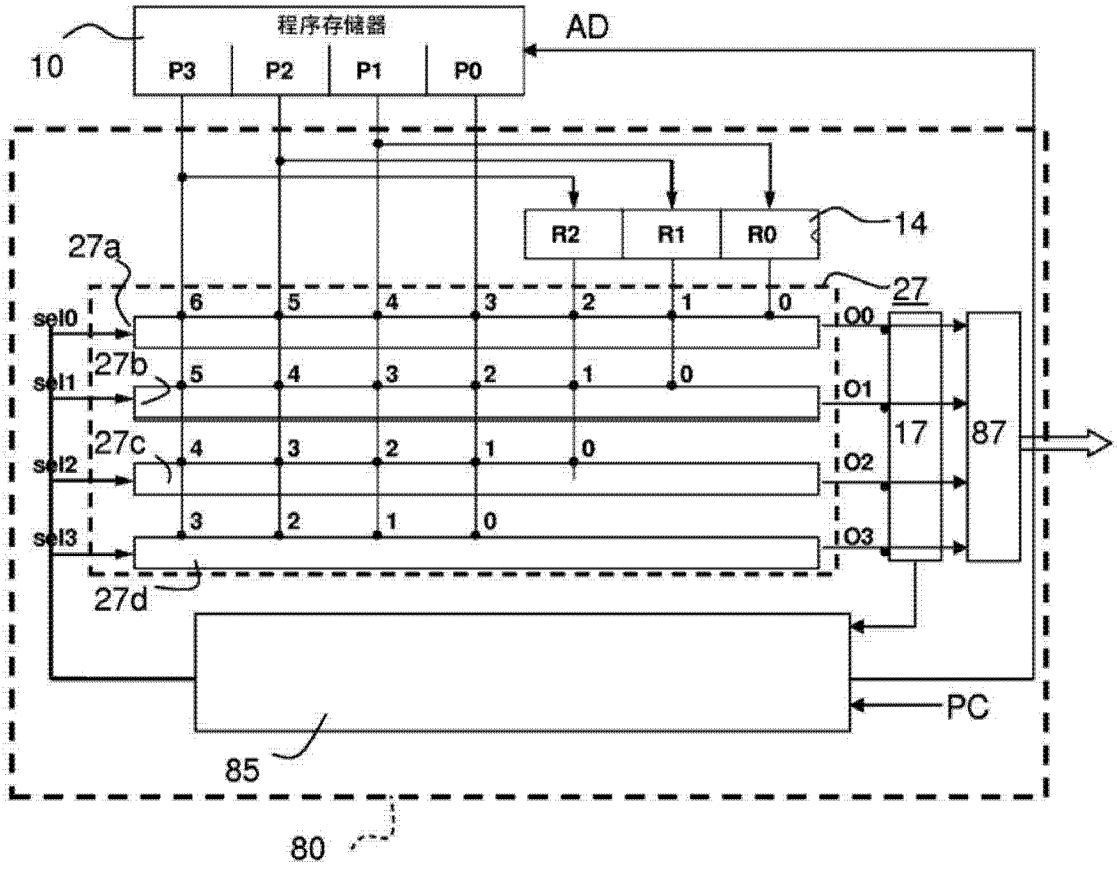

程序存储器
10
P3
P2
P1
P0
AD
R2
R1
R0
14
27a
27
6
5
4
3
2
1
0
sel0
O0
27b
sel1
O1
17
87
27c
sel2
O2
sel3
O3
27d
PC
85
80

图 12

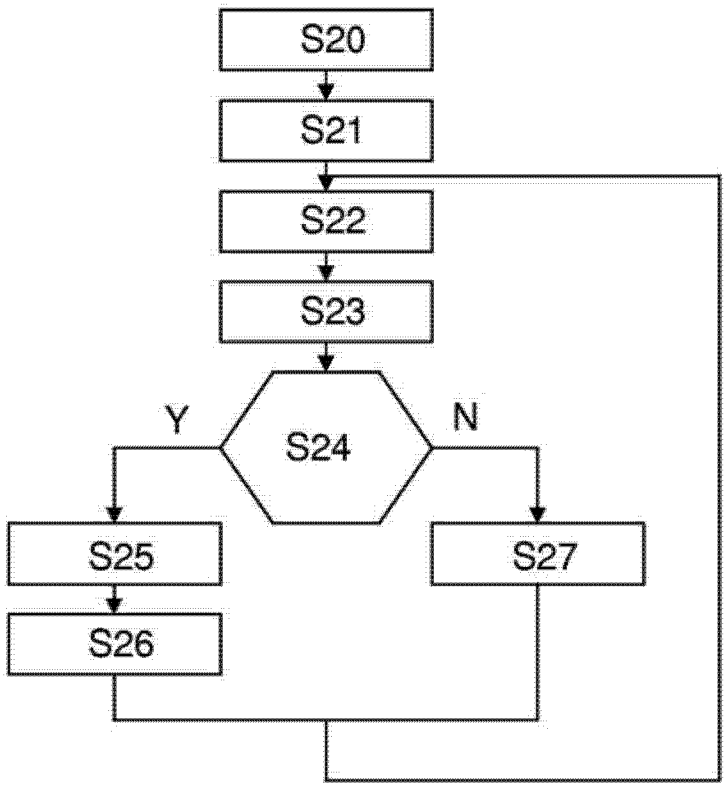


图 13

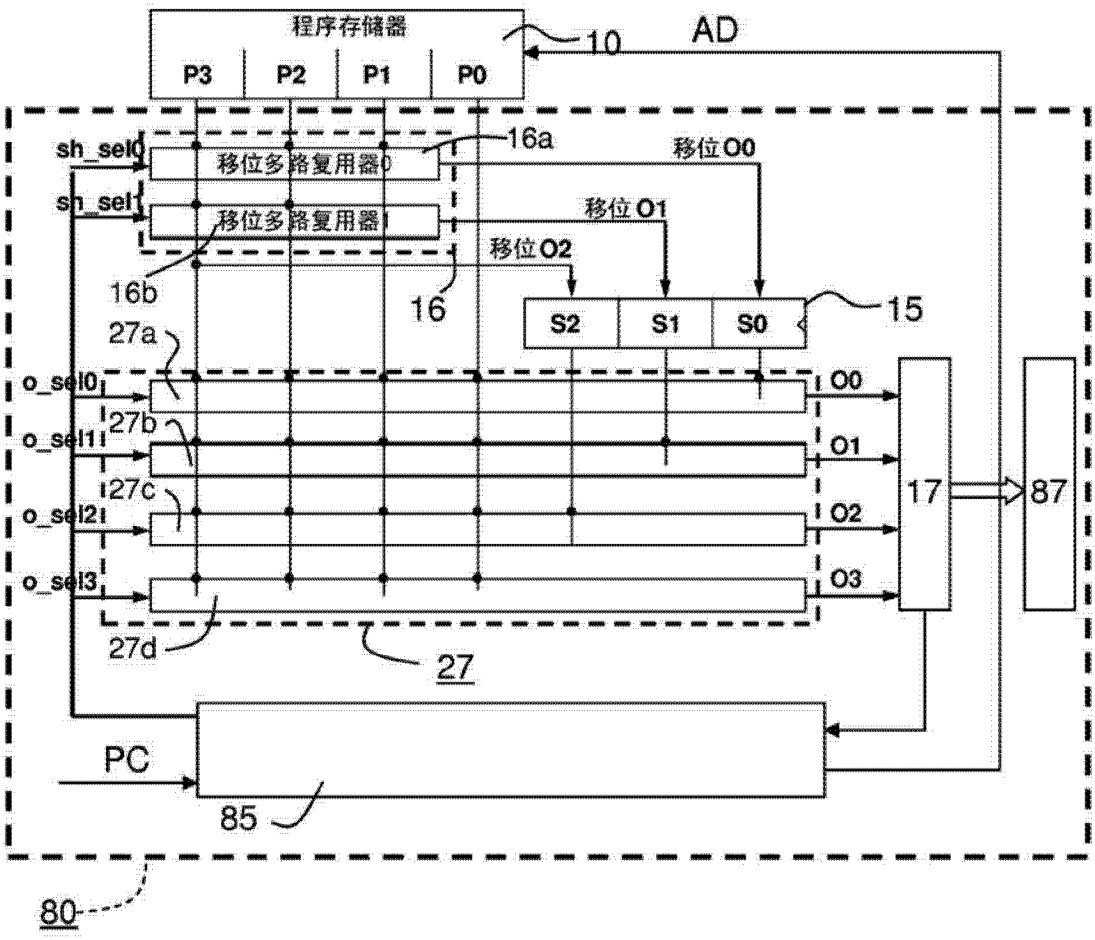

程序存储器
P3
P2
P1
P0
10
AD
sh_sel0
sh_sel1
移位多路复用器0
移位多路复用器1
16a
移位O0
移位O1
移位O2
16b
16
S2
S1
S0
15
27a
o_sel0
27b
o_sel1
27c
o_sel2
o_sel3
27d
27
O0
O1
O2
O3
17
87
PC
85
80

图 14

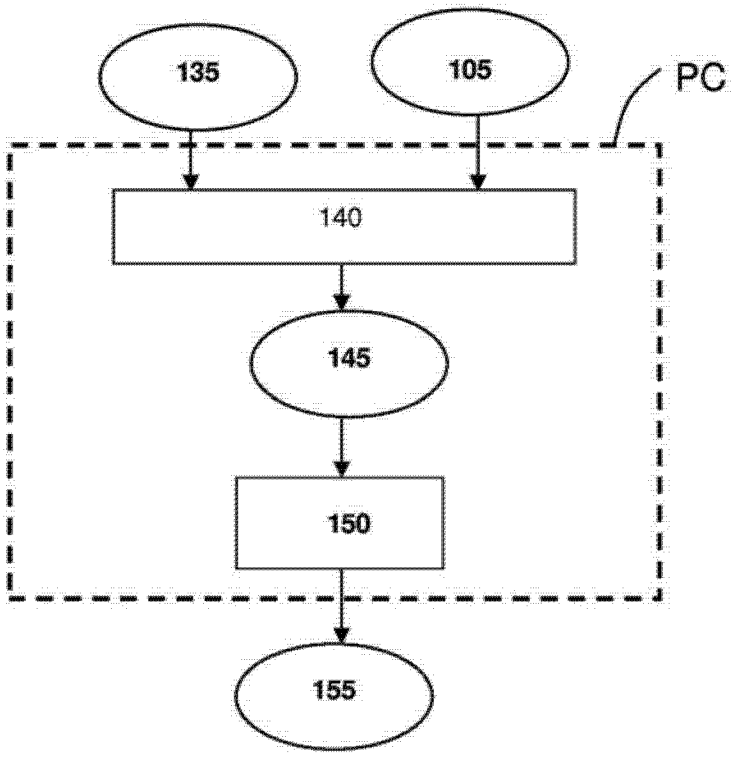


图 15