

一种固定常数相关指令的重命名方法及系统

技术领域

本发明涉及微处理器技术领域，具体涉及一种固定常数相关指令的重命名方法及系统。

背景技术

微处理器经过 50 多年的发展，微处理器的架构伴随着半导体工艺经历了蓬勃的发展。从单核到物理多核及逻辑多核；从顺序执行到乱序执行；从单发射到多发射；……。尤其在服务器领域，不断的追求处理器的性能。随着数据中心和科学计算的要求越来越高，对除法及余数指令的性能要求也越高，同时指令中的除法和余数指令比重逐渐升高。除法和余数指令执行周期比较长，同时执行周期与数据相关，执行周期可变。这些因素对 CPU 的性能影响很大。

现有的微处理器不能对指令进行分类执行，所有的指令统一到执行单元执行，无法消除无效指令的执行。

发明内容

针对现有技术的不足，本发明公开了一种固定常数相关指令的重命名方法及系统，用于解决现有的微处理器不能对指令进行分类执行，所有的指令统一到执行单元执行，无法消除无效指令的执行的执行的问题。

本发明通过以下技术方案予以实现：

第一方面，本发明公开一种固定常数相关指令的重命名方法，包括以下步骤：

S1 解码器根据指令的特点，对指令进行分类，筛选出有固定常数的指令；

S2 在重命名阶段根据不同的固定常数，将 S1 中筛选指令的源寄存器和目的寄存器映射到不同的固定常数物理寄存器；

S3 更新寄存器重命名映射表 SPEC_MAP 和 ARCH_MAP，在指令提交

的时候释放固定常数对应的物理寄存器；

S4 指令根据源寄存器的物理寄存器去读物理寄存器堆或者通过执行单元 Forward 通道的指令执行结果得到源操作数；

S5 指令在执行单元执行完成时，根据目的寄存器的物理寄存器写物理寄存器堆；

S6 将 Forward 指令执行结果到流水线，依赖指令得到指令执行结果，进而实现指令的功能。

更进一步的，所述方法中，每条指令在重命名阶段得到源操作数对应的物理寄存器和目的寄存器物理寄存器，源寄存器的物理寄存器根据指令的相关性从重命名映射表或者空闲物理寄存器队列得到，目的寄存器的物理寄存器为新分配的空闲物理寄存器，从空闲物理寄存器队列得到。

更进一步的，所述方法中，在寄存器重命名之前，根据指令的特点，判断指令的源操作数或目的寄存器是否为一个固定的常数，判断指令中是否经常出现固定常数值 0，常数值 1，常数值，常数值 e，及以常数 e 为底数 2 的对数 $\text{Log}_e(2)$ 。

更进一步的，所述方法中，假设在重命名阶段的指令 inst0, inst1, inst2 和 inst3 同时进行固定常数判断，判断 inst0 的源操作数和目的操作是否为固定常数 CONST0、CONST1、CONST2、……、CONSTN-1，当 inst0 中的 R1_0 或者 R2_0 为固定常数时，R1_0 或者 R2_0 的物理寄存器映射到固定常数的位置；

假设当 R1_0 为固定常数 CONST0，那么物理寄存器的值为 0；

假设当 R1_0 为固定常数 CONST1，那么物理寄存器的值为 1；

假设当 R1_0 为固定常数 CONST2，那么物理寄存器的值为 2；

假设当 R1_0 为固定常数 CONSTN-1，那么物理寄存器的值为 N-1。

更进一步的，所述方法中，判断 R2_0，如果能够判断 inst0 的执行结果为固定常数，则 inst0 不需要申请新的空闲物理寄存器，inst0 将 RD_0 的 SPEC_MAP 和 ARCH_MAP 映射到固定常数的位置，并且释放

RD_0 在 ARCH_MAP 中被退换的物理寄存器；当判断 inst0 的执行结果为常数时，RD_0 的物理寄存器映射到这个固定常数的物理寄存器，在实现具体的指令集时，根据指令的特点，判断指令的源操作数即目的寄存器是否为固定常数。

更进一步的，所述方法中，判断 inst1、判断 inst2 和 inst3 的源操作数和目的操作是否为固定常数 CONST0、CONST1、CONST2、……、CONSTN-1，如果 R1_1 或者 R2_1 为固定常数时，R1_1 或者 R2_1 映射到对应的固定常数，当 R1_1 为固定常数 0，并且 R2_1 不为固定常数时，如果此时 inst1 是一条 MOV 指令，同时正好满足 R2_1 等于 RD_0，及 RD_0 映射到一个固定常数，此时 inst1 的物理寄存器 RD_1 也映射到 RD_0 映射的固定常数，RD_0 和 RD_1 映射到一个相同的固定常数，将 RD_0 和 R2_1 的架构寄存器映射到固定常数的物理寄存器地址，更新 SPEC_MAP 和 ARCH_MAP 中 RD_0 和 R2_1 的映射关系，Inst2 的判断过程与 inst0 和 inst 的完全类似，其需要判断 inst2 是否与 inst0 和 inst1 存在相同的固定常数，inst3 也需要 inst3 是否与 inst0、inst1 和 inst2 存在相同的固定常数。

更进一步的，所述方法中，指令包含源操作数和目的寄存器，指令进入执行单元执行之前，需要得到所有的源操作数，保留站中指令的操作数都准备完成后，发射到执行单元执行，执行完成后，执行结果写到目的寄存器，并且按序提交。

第二方面，本发明公开一种固定常数相关指令的重命名系统，所述系统用于执行第一方面所述的固定常数相关指令的重命名方法，包括寄存器、解码器和执行单元。

更进一步的，所述寄存器包括架构寄存器和物理寄存器，在重命名阶段，指令目的寄存器的架构寄存器映射到一个物理寄存器，并且更新到重命名映射表 SPEC_MAP，指令执行完成后，在重排序缓存中按序提交，指令目的寄存器的架构寄存器更新到 ARCH_MAP。

本发明的有益效果为：

本发明对指令进行分类处理，有固定常数指令不需要进入执行单元，而是通过重命名的方法实现指令的执行，指令执行效率高。

附图说明

为了更清楚地说明本发明实施例或现有技术中的技术方案，下面将对实施例或现有技术描述中所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅是本发明的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其他的附图。

图 1 是本发明实施例指令格式定义图；

图 2 是本发明实施例寄存器重命名映射图；

图 3 是本发明实施例寄存器常数映射图；

图 4 是本发明实施例目的寄存器为固定常数执行流水线图；

图 5 是本发明实施例常数相关指令依赖关系图；

图 6 是本发明实施例 RISC V 指令实例物理寄存器映射图。

具体实施方式

为使本发明实施例的目的、技术方案和优点更加清楚，下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例是本发明一部分实施例，而不是全部的实施例。基于本发明中的实施例，本领域普通技术人员在没有作出创造性劳动前提下所获得的所有其他实施例，都属于本发明保护的范围。

实施例 1

本实施例公开一种通过重命名的方法，实现有固定常数指令的新方法。在解码器根据指令的特点，对指令进行分类，筛选出有固定常数的指令，在重命名阶段根据不同的固定常数，将这类指令的源寄存器和目的寄存器映射到不同的固定常数物理寄存器，更新寄存器重命名映射表 SPEC_MAP 和 ARCH_MAP，在指令提交的时候释放固定常数对应

的物理寄存器，从而实现指令的功能。

不同的指令集定义了不同的指令格式，无论哪种指令集，包含寄存器相关的指令格式如图 1. 所示。指令包含源操作数 src0、src1、……和目的寄存器 rd1、rd2、……。指令进入执行单元执行之前，需要得到所有的源操作数。保留站中指令的操作数都准备完成后，发射到执行单元执行，执行完成后，执行结果写到目的寄存器，并且按序提交。

在支持乱序执行的处理器架构中，每条指令在重命名阶段得到源操作数对应的物理寄存器和目的寄存器物理寄存器。源寄存器的物理寄存器根据指令的相关性从重命名映射表或者空闲物理寄存器队列得到。目的寄存器的物理寄存器为新分配的空闲物理寄存器，即从空闲物理寄存器队列得到。指令根据源寄存器的物理寄存器去读物理寄存器堆或者通过执行单元 Forward 通道的指令执行结果得到源操作数。指令在执行单元执行完成时，根据目的寄存器的物理寄存器写物理寄存器堆，同时 Forward 指令执行结果到流水线，依赖指令得到指令执行结果。

假设指令的架构寄存器分别为 r0、r1、r2、r3、……、r31。在重命名阶段，指令目的寄存器的架构寄存器映射到一个物理寄存器，并且更新到重命名映射表 SPEC_MAP。指令执行完成后，在重排序缓存中按序提交，指令目的寄存器的架构寄存器更新到 ARCH_MAP，如图 2. 所示。

在图 2. 中，在重命名阶段的映射表 SPEC_MAP 保存了架构寄存器 r0、r1、r2、……、r31 最新的物理寄存器映射关系，当出现指令 misprediction 或者 exception 刷新时时，需要恢复。ARCH_MAP 中保存了指令提交之后 r0、r1、r2、……、r31 的物理寄存器映射关系，当重排序缓存处理 exception 时，将 ARCH_MAP 内容全部更新到 SPEC_MAP，SPEC_MAP 恢复到 exception 刷新之前的状态。当出现分支指令的 misprediction 时，根据重排序缓存恢复 SPEC_MAP。

在寄存器重命名之前，根据指令的特点，可以判断指令的源操作

数或目的寄存器是否为一个固定的常数。比如指令中经常出现固定常数值 0, 常数值 1, 常数值, 常数值 e, 及以常数 e 为底数 2 的对数 $\text{Log}_e(2)$ 等。本文提出在物理寄存器堆中分配 N 个专门保存常数的物理寄存器。物理寄存器堆的常数为 CONST0、CONST1、CONST2、……、CONSTN-1, 如图 3. 所示。

当指令的源操作数为固定的一个常数时, 将这个源寄存器的物理寄存器映射为这个固定常数的位置。如果一条指令的执行结果为一个常数, 那么这条指令不需要执行, 而是将这个目的寄存器的物理寄存器映射到固定常数的物理寄存器, 同时这条指令也不需要分配新的物理寄存器。不论这条指令的功能是什么, 因为能够判断这条指令的结果为固定常数, 因此该指令是不需要执行, 而是通过重命名的映射操作, 该指令执行完成, 如图 4. 所示。通过固定常数的重命名机制, 不但节省了物理寄存器资源, 而且指令还不需要执行, 大大提高指令的执行效率。

实施例 2

本实施例公开一种固定常数相关指令的重命名方法, 其实现不限于任何指令集, 为了方便说明原理, 在表 1. 中列举的指令格式为 2 个源操作数和 1 个目的寄存器。同时也不限于任何处理器的架构, 现也假设指令的处理带宽为 4 条指令每个时钟周期。

表格 1 指令判断固定常数

Inst0	R1_0	R2_0	RD_0
Inst1	R1_1	R2_1	RD_1
Inst2	R1_2	R2_2	RD_2
Inst3	R1_3	R2_3	RD_3

假设在重命名阶段的 4 条指令 inst0, inst1, inst2 和 inst3 同时进行固定常数判断:

1. 首先, 判断 inst0 的源操作数和目的操作是否为固定常数 CONST0、CONST1、CONST2、……、CONSTN-1。在图 3 中, 假设 CONST0、

CONST1、CONST2、……、CONSTN-1 在物理寄存器堆的位置为 0、1、2、3、……、N-1。当 inst0 中的 R1_0 或者 R2_0 为固定常数时，R1_0 或者 R2_0 的物理寄存器映射到固定常数的位置。假设当 R1_0 为固定常数 CONST0，那么物理寄存器的值为 0；假设当 R1_0 为固定常数 CONST1，那么物理寄存器的值为 1；假设当 R1_0 为固定常数 CONST2，那么物理寄存器的值为 2；假设当 R1_0 为固定常数 CONSTN-1，那么物理寄存器的值为 N-1。同理，R2_0 也需要做类似的判断。如果能够判断 inst0 的执行结果为固定常数，这些常数为 CONST0、CONST1、CONST2、……、CONSTN-1。那么 inst0 不需要申请新的空闲物理寄存器，即 inst0 只需要将 RD_0 的 SPEC_MAP 和 ARCH_MAP 映射到固定常数的位置，并且释放 RD_0 在 ARCH_MAP 中被退换的物理寄存器。

当判断 inst0 的执行结果为一个常数时，那么 RD_0 的物理寄存器映射到这个固定常数的物理寄存器。例如，inst0 为一条 XOR 指令，R1_0 等于 R_2_0，此时，虽然不需要知道 R1_0 和 R_2_0 的具体值，不需要去读物理寄存器堆，但是就可得到指令的执行结果为 0，RD_0 映射的物理地址为 0。同理，如果 inst0 是一条整数除法指令 DIV，指令执行 RD_0/RD_1，只要当 R1_0 等于 R_2_0，并且 RD_1 的值不是 0，那么该条指令执行结果为 1。

在实现具体的指令集时，根据指令的特点，判断指令的源操作数即目的寄存器是否为固定常数，得到类似的优化。源操作数为固定常数的操作数对应的数据 ready 位也为有效，即指令在保留站的数据准备好一直处于有效状态，只等指令其它非固定常数的源操作数准备好，指令发射到执行单元。R1_0 和 R2_0 的物理寄存器地址分别为 R1_0_phy_reg 和 R2_0_phy_reg。R1_0_phy_reg 和 R2_0_phy_reg 可能是固定常数的物理寄存器也可能是非常数的物理寄存器。

2. 然后，再判断 inst1 的源操作数和目的操作是否为固定常数 CONST0、CONST1、CONST2、……、CONSTN-1。如果 R1_1 或者 R2_1 为固定常数时，R1_1 或者 R2_1 映射到对应的固定常数。当 R1_1 为固定

常数 0, 并且 R2_1 不为固定常数时, 如果此时 inst1 是一条 MOV 指令, 同时正好满足 R2_1 等于 RD_0, 及 RD_0 映射到一个固定常数, 此时 inst1 的物理寄存器 RD_1 也映射到 RD_0 映射的固定常数, 即 RD_0 和 RD_1 映射到一个相同的固定常数。出现这种情况时, RD_0 和 RD_1 不需要申请分配物理寄存器, 而是将 RD_0 和 R2_1 的架构寄存器映射到固定常数的物理寄存器地址, 即更新 SPEC_MAP 和 ARCH_MAP 中 RD_0 和 R2_1 的映射关系。

如图 3. 所示, 存在不同的架构寄存器映射到相同的固定常数物理寄存器。如果当 R2_1 不等于 RD_0 时, 并且 R2_1 结果也为一个常数时, 那么 R2_1 映射到一个不同的固定常数物理寄存器地址。Inst1 做 inst0 类似的判断, 得到源操作数和目的寄存器各种可能的固定常数物理寄存器地址。

3. 其次, 判断 inst2 和 inst3 的源操作数和目的操作是否为固定常数 CONST0、CONST1、CONST2、……、CONSTN-1。Inst2 的判断过程与 inst0 和 inst1 的完全类似, 不同之处在于需要判断 inst2 是否与 inst0 和 inst1 存在相同的固定常数。同理, inst3 也需要 inst3 是否与 inst0、inst1 和 inst2 存在相同的固定常数。具体的判断过程不再赘述。

实施例 3

本实施例以 RISC-V 的整数指令集为例进行说明。RISC V 的整数指令包括 32 个架构寄存器: x0、x1、……、x31。x0 的值固定为 0。假设处理器的带宽为 4 条指令每个时钟周期, 即重命名每个时钟周期重命名 4 条指令。

表格 2 RISC V 指令实例

序号	指令	第一个源操作数	第二个源操作数	目的寄存器	描述
0	ADDI	x3	122	X21	$x3+122 \rightarrow x21$
1	ADD	x3	x5	x11	$x3+x5 \rightarrow x11$
2	SUB	x11	x11	x1	$x11-x11 \rightarrow x1$

3	MUL	x7	x1	x29	$x7 * x1 \rightarrow x29$
4	LOAD	x21	48	x27	$MEM(x21+48) \rightarrow x27$
5	MUL	x27	x29	x31	$x27 * x29 \rightarrow x31$
6	AMOSWAP.W	x27	x31	x24	$MEM(x27) \rightarrow x24$ $x24 \rightarrow MEM(x27)$
7	SLTU	x24	x24	x30	当第1个源操作数小于第2个源操作数时，置 x30 为 1；否则置为 0。
					

序号 0 到序号 7 指令之间的依赖关系如图 5. 所示。按照正常流程，存在依赖的指令，必须得到数据后才能执行，否则只能处于 no-ready 状态。通过指令执行结果是否为常数，并且进行固定常数重命名后，部分指令的依赖关系可以取消。并且通过固定常数的重命名方法完成指令的执行过程。

序号 2 的 SUB 指令进行减法操作，第一个源操作数减去第二个源操作数，由于这 2 个操作数都是 x11，因此，不论 x11 具体的值，SUB 指令的执行结果为 0。SUB 指令可以通过重命名的方式解除了 SUB 与 ADD 指令的依赖关系，如图 5. 所示，并且 SUB 的目的寄存器 x1 不需要分配物理寄存器，更新 SPEC_MAP 表中 x1 映射的物理寄存器为 CONST0，同时把 SUB 在重排序缓存中的执行完成信号置为 1。

在重排序缓存中提交 SUB 时，释放 x1 在 ARCH_MAP 中的物理寄存器，并且把 x1 在 ARCH_MAP 中的物理寄存器更新为 x0。序号 3 的 MUL 指令进行乘法操作，由于当前周期的 SUB 指令执行结果为常数 CONST0，因此，MUL 指令的结果也为常数 CONST0。MUL 目的寄存器 x29 在 SPEC_MAP 的映射关系更新为 CONST0 的物理寄存器地址。并且 MUL 不需要进入乘法执行单元执行，而是直接更新重排序缓存中的执行完成信号，等待提交。

当该 MUL 在重排序缓存提交时，架构寄存器 x29 的映射关系更新为 x0，并且释放 x29 之前的物理寄存器。序号为 5 的乘法指令也解除了和序号为 4 的 LOAD 指令的依赖关系，并且指令执行结果也为 CONST0。

因此，序号为 5 的乘法指令和序号为 3 的乘法指令处理流程类似，不再赘述。序号为 6 的原子操作指令 AMOSWAP.W 也可以得到第 2 个源寄存器 x31 的值为 CONST0。序号为 7 的置位指令 SLTU 也可以判断得到指令执行的结果为 0，因为 2 个源操作数都为 x24，不论 x24 的值为任何值，指令执行结果都为 0。因此，SLTU 也不需要执行，而是将 x30 在 SPEC_MAP 中的映射关系更新为 x0，即为 CONST0，如图 6. 所示。

这些映射到常数 CONST0 的指令，不仅不需要分配新的物理寄存器，而且还不需要分派到保留站、执行单元，从而大大提高了指令的执行效率及处理器能够容纳更多的指令。特别是针对执行时间长的指令，比如 MUL、LOAD 和 AMO 等指令，依赖这些指令的指令可以通过重命名的方式实现。

X86 指令集定义的浮点指令寄存器为一个深度为 8，宽度为 80 的栈。栈用 ST 表示，栈顶为 ST(0)。当对栈进行写操作时，即对栈进行 PUSH 操作。X86 定义的常数相关指令如表 3. 所示。

表格 3X86 浮点常数指令

指令	第一个源操作数	第二个源操作数	目的寄存器	描述
FLDZ	0.0	无	ST(0)	ST 为浮点寄存器栈，栈顶的值写常数 0.0
FLD1	+1.0	无	ST(0)	栈顶的值写常数+1.0
FLDPI	π	无	ST(0)	栈顶的值写常数 π
FLDL2T	$\text{Log}_2(10)$	无	ST(0)	栈顶的值写常数 2 为底 10 的对数
FLDL2E	$\text{Log}_2(e)$	无	ST(0)	栈顶的值写常数 2 为底 e 的对数
FLDLG2	$\text{Log}_{10}(2)$	无	ST(0)	栈顶的值写常数 10 为底 2 的对数
FLDLN2	$\text{Log}_e(2)$	无	ST(0)	栈顶的值写常数 e 为底 2 的对数

X86 浮点指令中的常量宽度和处理器的模式相关，如果这些常量在流水线中跟着指令走完流水线，那么流水线中会消耗大量的硬件资源。将这些常量采用本文提出的重命名方法，不但可以降低硬件资源，而

且还可以节省物理寄存器和提高指令的执行效率。具体的实现方式和 RISC V 指令集实例完全类似，不再赘述。

实施例 4

本实施例公开一种固定常数相关指令的重命名系统，包括寄存器、解码器和执行单元。所述寄存器包括架构寄存器和物理寄存器，在重命名阶段，指令目的寄存器的架构寄存器映射到一个物理寄存器，并且更新到重命名映射表 SPEC_MAP，指令执行完成后，在重排序缓存中按序提交，指令目的寄存器的架构寄存器更新到 ARCH_MAP。

综上，本发明对指令进行分类处理，有固定常数指令不需要进入执行单元，而是通过重命名的方法实现指令的执行，指令执行效率高。

以上实施例仅用以说明本发明的技术方案，而非对其限制；尽管参照前述实施例对本发明进行了详细的说明，本领域的普通技术人员应当理解：其依然可以对前述各实施例所记载的技术方案进行修改，或者对其中部分技术特征进行等同替换；而这些修改或者替换，并不使相应技术方案的本质脱离本发明各实施例技术方案的精神和范围。

权 利 要 求 书

1、一种固定常数相关指令的重命名方法，其特征在于，所述方法包括以下步骤：

S1 解码器根据指令的特点，对指令进行分类，筛选出有固定常数的指令；

S2 在重命名阶段根据不同的固定常数，将 S1 中筛选指令的源寄存器和目的寄存器映射到不同的固定常数物理寄存器；

S3 更新寄存器重命名映射表 SPEC_MAP 和 ARCH_MAP，在指令提交的时候释放固定常数对应的物理寄存器；

S4 指令根据源寄存器的物理寄存器去读物理寄存器堆或者通过执行单元 Forward 通道的指令执行结果得到源操作数；

S5 指令在执行单元执行完成时，根据目的寄存器的物理寄存器写物理寄存器堆；

S6 将 Forward 指令执行结果到流水线，依赖指令得到指令执行结果，进而实现指令的功能。

2、根据权利要求 1 所述的固定常数相关指令的重命名方法，其特征在于，所述方法中，每条指令在重命名阶段得到源操作数对应的物理寄存器和目的寄存器物理寄存器，源寄存器的物理寄存器根据指令的相关性从重命名映射表或者空闲物理寄存器队列得到，目的寄存器的物理寄存器为新分配的空闲物理寄存器，从空闲物理寄存器队列得到。

3、根据权利要求 1 所述的固定常数相关指令的重命名方法，其特征在于，所述方法中，在寄存器重命名之前，根据指令的特点，判断指令的源操作数或目的寄存器是否为一个固定的常数，判断指令中是否经常出现固定常数值 0，常数值 1，常数值，常数值 e，及以常数 e 为底数 2 的对数 $\text{Log}_e(2)$ 。

4、根据权利要求 1 所述的固定常数相关指令的重命名方法，其特征在于，所述方法中，假设在重命名阶段的指令 inst0, inst1, inst2 和 inst3 同时进行固定常数判断，判断 inst0 的源操作数和目的操作

是否为固定常数 CONST0、CONST1、CONST2、……、CONSTN-1，当 inst0 中的 R1_0 或者 R2_0 为固定常数时，R1_0 或者 R2_0 的物理寄存器映射到固定常数的位置；

假设当 R1_0 为固定常数 CONST0，那么物理寄存器的值为 0；

假设当 R1_0 为固定常数 CONST1，那么物理寄存器的值为 1；

假设当 R1_0 为固定常数 CONST2，那么物理寄存器的值为 2；

假设当 R1_0 为固定常数 CONSTN-1，那么物理寄存器的值为 N-1。

5、根据权利要求 4 所述的固定常数相关指令的重命名方法，其特征在于，所述方法中，判断 R2_0，如果能够判断 inst0 的执行结果为固定常数，则 inst0 不需要申请新的空闲物理寄存器，inst0 将 RD_0 的 SPEC_MAP 和 ARCH_MAP 映射到固定常数的位置，并且释放 RD_0 在 ARCH_MAP 中被退换的物理寄存器；当判断 inst0 的执行结果为常数时，RD_0 的物理寄存器映射到这个固定常数的物理寄存器，在实现具体的指令集时，根据指令的特点，判断指令的源操作数即目的寄存器是否为固定常数。

6、根据权利要求 4 所述的固定常数相关指令的重命名方法，其特征在于，所述方法中，判断 inst1、判断 inst2 和 inst3 的源操作数和目的操作是否为固定常数 CONST0、CONST1、CONST2、……、CONSTN-1，如果 R1_1 或者 R2_1 为固定常数时，R1_1 或者 R2_1 映射到对应的固定常数，当 R1_1 为固定常数 0，并且 R2_1 不为固定常数时，如果此时 inst1 是一条 MOV 指令，同时正好满足 R2_1 等于 RD_0，及 RD_0 映射到一个固定常数，此时 inst1 的物理寄存器 RD_1 也映射到 RD_0 映射的固定常数，RD_0 和 RD_1 映射到一个相同的固定常数，将 RD_0 和 R2_1 的架构寄存器映射到固定常数的物理寄存器地址，更新 SPEC_MAP 和 ARCH_MAP 中 RD_0 和 R2_1 的映射关系，Inst2 的判断过程与 inst0 和 inst1 的完全类似，其需要判断 inst2 是否与 inst0 和 inst1 存在相同的固定常数，inst3 也需要 inst3 是否与 inst0、inst1 和 inst2 存在相同的固定常数。

7、根据权利要求1所述的固定常数相关指令的重命名方法，其特征在于，所述方法中，指令包含源操作数和目的寄存器，指令进入执行单元执行之前，需要得到所有的源操作数，保留站中指令的操作数都准备完成后，发射到执行单元执行，执行完成后，执行结果写到目的寄存器，并且按序提交。

8、一种固定常数相关指令的重命名系统，所述系统用于执行如权利要求1-7任一项所述的固定常数相关指令的重命名方法，其特征在于，包括寄存器、解码器和执行单元。

9、根据权利要求8所述的固定常数相关指令的重命名系统，其特征在于，所述寄存器包括架构寄存器和物理寄存器，在重命名阶段，指令目的寄存器的架构寄存器映射到一个物理寄存器，并且更新到重命名映射表 SPEC_MAP，指令执行完成后，在重排序缓存中按序提交，指令目的寄存器的架构寄存器更新到 ARCH_MAP。