第 33 卷 第 3 期 计 算 机 工 程 2007 年 2 月
Vol.33 No.3 Computer Engineering February 2007
·工程应用技术与实现· 文章编号：1000—3428(2007)03—0251—02 文献标识码：A 中图分类号：TP311

一种高效的基于 ASIPs 的 EPIC 指令编码方法

江山刚，张晓彤，王 沁
(北京科技大学计算机系，北京 100083)

摘 要：EPIC 技术不仅广泛应用于通用 CPU 的设计中，而且它还被应用于专用领域的专用指令集处理器(ASIPs)的设计中。目前使用的 EPIC 技术的性能提高是以程序代码量的急剧膨胀为代价的。为了减少 EPIC 程序代码长度，该文提出了一种新的指令编码方法——动态变长指令编码方法。测试结果表明，对于低编码率的语音编解码领域的 ASIPs，动态变长指令编码方法可以将代码的压缩率提高到 62.8%。
关键词：显示并行指令计算；专用指令集处理器；指令编码；动态变长

High Efficiency Instruction Encoding Method Based on EPIC in ASIPs

JIANG Shangang, ZHANG Xiaotong, WANG Qin
(Dept. of Computer, University of Science and Technology Beijing, Beijing 100083)

【Abstract】EPIC technology is applied not only to the designs of general-purpose CPU, but also to application specification instruction set processors (ASIPs) in application specific domain. But its high performance is at a cost of sharp expansion of EPIC program code. To solve it, this paper presents a new encoding method——dynamic variable-length encoding. Tests indicate that the method can achieves an average static code compression ratio of 62.8% in speech algorithm of low encoding code.
【Key words】Explicitly parallel instruction computing (EPIC); ASIPs; Instruction encoding; Dynamic variable-length

对于专用领域，尤其是数字处理信号领域，对处理器的性能提出了越来越高的要求。由于仅提高主频和增加流水深度不能完全满足要求，而且还会大大地增加设计复杂度，因此并行处理技术成为 IC 设计的必然趋势。EPIC(Explicitly Parallel Instruction Computing)技术提供了良好的指令级的并行性，但同时也带来了程序代码膨胀的问题；而且随着并行度的增加，程序代码膨胀现象越突出。对于嵌入式应用系统，程序量是影响系统成本一个非常重要的因素，减少程序代码量成为控制成本的一个关键因数。

目前有两种途径可以解决程序代码膨胀：压缩定长的指令集和使用变长的指令集。前者因为指令是经过压缩存储的，所以在执行前需要附加的解压过程。无论是对于RISC还是VLIW，前者可以缩减大约 30%~40%的程序代码量[5,6]。但是指令解压部分的电路开销又大大抵消了减少程序代码量所获得的成本优势，同时指令的解压延时阻碍性能的进一步提高。后者通过使用变长的指令集，提高了程序代码存储空间的有效利用率，但还是无法消除多发射指令的空操作(NOP)。

本文在分析了常见的EPIC指令格式的基础上提出了一种可以减少程序代码长度的新的EPIC指令编码方法——动态变长指令编码方法。这种编码方法可以完全消除指令中的NOP，大大减少程序代码量，比变长指令更能提高程序代码存储空间的有效利用率；而且保留定长指令在主存和Cache里的容易检索优点，还避免了压缩指令在执行时的解压延时。

1 常见EPIC指令编码分析

本部分是从减少程序代码角度分析常见的 EPIC 指令的特点；通常 EPIC 的指令构成特点如下：

一个常见的 EPIC 指令束(instruction bundle)通常由一个模板选择域(template select)和后续的多个指令槽(instruction slot)组成，一个指令在指令束中所占的空间称为指令槽。在指令束格式设计时，通常将一些具有互斥性的和对寄存器堆具有大致相当延时和连接关系的指令划分为一个指令组(instruction group)(例如同一个 ALU 单元的加法和减法指令)。指令选择域(instruction select)指示从特定的指令类中的某一特定的指令。一个指令束可以由一个或多个指令组成：一个指令束的模板选择域的值确定了构成当前指令束的指令类型和数量；而指令选择域的值确定了当前指令类型中的特定的指令。指令束选择域的值确定了一条指令束的并发指令数和和并发的指令类型。传统的 EPIC 指令束由以下原因导致程序代码膨胀：

(1)指令长度不等导致指令槽空间浪费

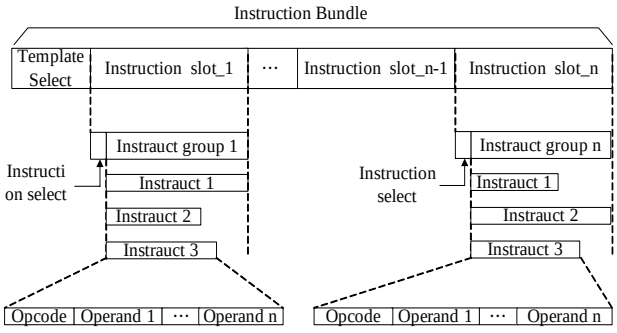

图 1 传统的 EPIC 指令束的组成

作者简介：江山刚(1978－)，男，硕士生，主研方向：IC设计；张晓彤，副教授；王 沁，教授、博导
收稿日期：2006-03-18 **E-mail：**jiangshangang@163.com

对于传统的 EPIC，指令束的长度是定长的。如图 1 所示，由于同一指令组中不同的指令之间长度往往不同，甚至相差很大，但是指令槽的长度为相应的指令组中最常长的指令长度，因此导致一个指令槽的空间浪费很大。

(2)指令束中往往填充大量空指令

对于传统的 EPIC，每种指令束格式的并发指令数是固定的。但在特定的程序中往往由于程序上下文存在数据相关性、资源相关性和跳转分支等因素，导致实际的并发指令数小于指令束允许的并发数，这时就用空指令(No Operation, NOP)填充，使得指令束的长度达到规定长度。

下面举例说明：假设一种 EPIC 的 CPU 有 4 个功能单元：2 个 Load/Store 单元，1 个整数 ALU 单元，1 个分支单元。每个指令束可以并发 4 个指令：Load/Store，Load/Store，Integer ALU 和分支指令，完成相应的指令需要的周期数为 2，2，1，2。对于下面程序的执行：

```
Load R1,A
Load R2,B
Add R3,R1,R2
Store C,R3
```

前两个指令没有数据相关性和资源相关性可以并行执行，第 3 个指令由于数据相关性和 load/store 单元的执行周期数的限制，只能在第 3 个周期执行；第 4 个指令也由于数据相关性只能在第 4 个周期执行，实际执行的 4 条指令束如表 1 所示。

表 1 EPIC 例子中实际执行的 4 条指令束

	指令槽、际执行单元和指令			
指令束编号	Slot1: Ld/St unit 0	Slot2: Ld/St unit 1	Slot3: Integer ALU	Slot4: Branch unit
1	Load R1,A	Load R2,B	NOP	NOP
2	NOP	NOP	NOP	NOP
3	NOP	NOP	Add R3,R1,R2	NOP
4	Store C, R3	NOP	NOP	NOP

在这段程序中由于存在相关性填充了大量的 NOP，使得程序代码急剧膨胀。

2 动态变长指令编码方法及主要特点

2.1 编码方法

如下动态变长指令由如下两大部分组成：

Head			Tail			
length	Template Select	Indicator	Slot 1	…	Slot n	Filling

2.1.1 指令头(Head)

指令头包含了本条指令束的所有控制信息，由以下 3 部分组成：

(1) Length：为长度域，指示当前指令束的实际长度；假设体系支持 N 种长度的指令束，则 Length 域的位宽为$\lceil Log_2N \rceil$，指令束长度变化的粒度以一个或多个字节为单位；

(2) Template Select：为模板域，指示本条指令束允许的并发指令的最大值和并发指令的类型；假设在指令集中共有 N 种指令格式，则模板域的位宽为$\lceil Log_2N \rceil$，对于不同格式的指令束，其支持的并发指令数即指令槽的数量可以不相同；Template 决定了每个指令束的最大长度值；

(3) Indicator：为指示域，对于任意的 Template Select，Indicator 都有 1bit 和每个指令槽一一对应，如果在指令束中某指令槽为 NOP，则这个指令槽对应的指示位为 0，同时该指令槽在指令束中不出现；否则为 1，并在指令槽中填入相应的指令和 Instruction select 值。

指示域的位宽可以是定长，也可以是变长的，指令头的长度也相应可以定长或变长的，但为了从译码器的规模和复杂度的角度来讲，定长的指令头是有利。

2.1.2 指令尾(Tail)

指令尾由 Filling 和一个或多个 Slot 组成：

(1)Slot<i>：一条指令束实际需要的指令槽，它的数量由指令束的模版选择域和指示域共同决定；填充每个指令槽的指令组由指令束的模版选择域决定；每个指令槽的长度由实际的指令所需要的长度决定(或实际长度少许增加值)。

(2)Filling：填充域。为了使得指令束长度规整，需要在实际指令填充一些填充位，其长度为指令束规整化前后的长度差；其平均长度为指令束长度粒度的 1/2。

举例说明：假设在一个特定体系中，其指令束长度支持 64、128、192 和 256 bits；对于指令束格式(Template=01111)，最大可以支持 8 个并发指令(Load1，Load2，store，Alu1，Alu2，Mul1，Mul2，Branch)，此时指令束长度正好为 256bits(LENGTH=11)(无需填充位)；如果这 8 个指令需要全部并发执行，那么 Indicator=11111111，指令编码如表 2。

表 2 指令编码

Length	Template Select	Indicator	Slot0	Slot1	Slot2	Slot3	Slot4	Slot5	Slot6	Slot7
11	01111	1111111	Load1	Load2	store	Alu1	Alu2	Mul1	Mul2	Branch

表 3 指令束实际编码

Length	Template Select	Indicator	Slot0	Slot2	Slot3	Slot5	Slot6	Filling
10	01111	10110110	Load1	store	Alu1	Mul1	Mul2	00

实际时指令束格式不变，但只需要完成 5 个并发指令 Load1、store、Alu1、Alu2、Mul1，Indicator=10110110；同时这 5 个指令的编码长度与指令头的长度之和为 190bits<192bits，需要 2bits 的填充位才能使指令束的长度达到 192bits(Length=10)；这条指令束实际编码如表 3。

2.2 主要特点

动态变长指令编码与变长指令编码不同，主要特点如下：

(1)指令束的长度由指令格式和指示域共同决定，并且是动态变化的；

(2)将单一的指令cache分为 2^n(n为整数)块容量大小相同的指令 cache，它们采用低地址交叉编址；

(3)指令束的长度粒度、PC 值的粒度与指令 cache 的单元地址的粒度三者相同；

(4)每次从 cache 读取指令束时的数据长度为动态变长指令束的最大长度，但程序计数器 PC 值的改变量为当前指令束的实际长度。

3 测试环境及测试结果

本文的的测试数据是在低编码率的语音算法的专用指令集处理器(ASIPs)上完成的。基准测试程序是采用清华大学开发的 4 个正弦激励线性预测(SELP)低速率语音编码算法(其编码率分别为0.6、0.8、1.2 和 2.4Kbps)的基础上完成的。由于采用了合理的激励模型，这些算法都具有很好的重建语音质量。其中 2.4Kbps速率算法的重建语音质量与国际上流行的相同速率的高质量AMBE算法相当，0.6Kbps算法重建语音的可懂度也能达到 90%以上。

由于基准测试程序算法复杂，数据之间有很大的相关性，因此指令束的并行度变化幅度很大。为了最大限度地提高性能和有效地利用程序存储空间，将传统的 EPIC 指令只支持单一规格的指令束长度，改变为指令束可以支持 4 种长度规格(64bits、128bits、192bits 和 256bits)，为方便叙述称之为变长指令编码方法。

为了增加可比性，按照动态变长指令编码方法进行编码

（下转第 255 页）