

# 寄存器堆互连的 VLIW 结构及其指令调度算法

周志雄 何 虎 杨 旭 张延军 孙义和

(清华大学微电子学研究所 北京 100084)

**摘 要** 超长指令字(Very Long Instruction Word, VLIW)处理器一般采用总线互连的多簇结构,每个簇中的功能单元共享一个本地寄存器堆,簇间采用总线传输数据,以避免功能单元增多时,全连通结构的延时、面积和功耗的快速增长;但簇间数据共享时的拷贝和延时,使得处理器在性能上有所下降.文中提出了一种寄存器堆互连的多簇 VLIW 结构,采用寄存器堆来连接各个簇,从而可以避免簇间数据传输的延时和额外的数据拷贝操作.同时也提出了针对这种结构的指令调度算法,以提高指令调度的性能.实验结果表明,与全连通的 VLIW 结构相比,寄存器堆互连结构在性能上仅有 13% 左右的性能下降,代码长度则基本不变;这都优于总线互连的多簇结构.

**关键词** 超长指令字;指令调度;寄存器堆

**中图法分类号** TP314

## Instruction Scheduling Algorithm for Register File Connectivity Clustered VLIW Architecture

ZHOU Zhi-Xiong HE Hu YANG Xu ZHANG Yan-Jun SUN Yi-He

(Institute of Microelectronics, Tsinghua University, Beijing 100084)

**Abstract** Generally VLIW(Very Long Instruction Word) processors are implemented as bus-connectivity clustered architecture, in which the function units in a cluster only access the corresponding local registers and different clusters are connected by buses. This architecture can avoid aggressive growing of delay, area and power in full-connectivity VLIW processors when function units increase. However, performance degradation is induced by its copy operations and latency of communications between clusters. This paper presents a new clustered architecture, in which a register file is used to connect all the clusters so as to turn copy and latency away. This paper also gives instruction scheduling algorithm to improve the performance. The experimental results indicate that this new architecture under the help of this scheduling algorithm shows only 13% performance degradation and little code size increase in average compared with those of fully connectivity VLIW architecture, which prevails that of bus-connectivity clustered VLIW architecture.

**Keywords** VLIW; instruction scheduling; register file

## 1 引 言

超长指令字(Very Long Instruction Word,

VLIW)结构处理器一般含有多个功能单元和较大的寄存器数.所有功能单元都可以访问每个寄存器的全连通结构(Full-Connectivity VLIW, FC-VLIW),需要一个多端口的大寄存器堆,这使得处

收稿日期:2006-02-27;最终修改稿收到日期:2007-08-27.本课题得到国家自然科学基金(60236020)资助.周志雄,男,1979年生,博士研究生,研究方向为面向多媒体应用处理器的自动设计及其编译技术. E-mail: zhouzx02@mails. tsinghua. edu. cn. 何 虎,男,1974年生,助理研究员,研究方向为 DSP 设计方法学、LSI/VLSI 测试方法学和可测性设计. 杨 旭,男,1982年生,博士研究生,研究方向为低功耗编译. 张延军,男,1978年生,博士研究生,研究方向为面向多媒体应用的处理器设计. 孙义和,男,1945年生,教授,博士生导师,研究领域为 LSI/VLSI 测试方法学、可测性设计和多媒体 VLSI 设计技术.

理器的时钟频率下降,面积和功耗增加.因此,VLIW 处理器往往实现为多簇结构<sup>[1]</sup>,每个簇中的功能单元共享一个较小的本地寄存器堆,簇间用总线互连,以消除多端口大寄存器堆对时钟频率、面积和功耗的负面影响.

多簇 VLIW 处理器的一个关键问题就是簇间的数据传输.大部分处理器采用簇间的总线和开关网络来实现数据传输,如 TI 公司的 TMS320C6000 系列处理器<sup>[2]</sup>、Analog 公司的 TigerSharc 处理器<sup>[3]</sup>等,这种结构我们称之为总线互连的多簇 VLIW (Bus-Connectivity Clustered VLIW, BCC-VLIW).总线互连的数据传输机制往往会带来传输的延时以及额外的数据拷贝操作.传输的延时使数据产生后不能马上被属于另一个簇的功能单元所使用,额外的拷贝操作会占用功能单元的某个周期,因而,处理器的性能将有所下降.

我们提出了一种寄存器堆互连的多簇 VLIW 结构 (Register File Connectivity Cluster VLIW, RFCC-VLIW)<sup>[4]</sup>,它采用寄存器堆连接各个簇,以实现簇间的数据传输.当一个簇产生的结果需要被另一个簇使用时,就将结果存入这个互连寄存器堆中,可以直接被其它簇使用,而不需要任何的拷贝操作,也不存在任何的延时.

VLIW 处理器所能达到的性能很大程度上依赖于编译器.相对 FC-VLIW 结构,多簇 VLIW 处理器的编译在指令调度算法上需要增加簇分配的过程.如何合理地将指令分配到各个簇执行,以最大程度地减少传输延时和数据拷贝带来的处理器性能下降,是多簇 VLIW 处理器调度算法的关键<sup>[1,5]</sup>.以 BCC-VLIW 结构处理器为目标,已经有多种调度算法被提出,如 BUG 算法<sup>[6]</sup>、LC-VLIW 算法<sup>[7]</sup>、UAS 算法等<sup>[8]</sup>,其中 UAS 算法效果较好.

RFCC-VLIW 结构处理器不存在簇间数据传输时的拷贝和延时,但由于互连寄存器堆端口数的限制,使同簇中的功能单元不能同时都访问互连寄存器堆,也会造成处理器性能相对 FC-VLIW 结构的下降.我们以 RFCC-VLIW 结构为目标,提出了相应的指令调度算法,以减少端口数限制引起的性能下降.

我们在本文的第 2 节对 RFCC-VLIW 结构及需要在调度算法上处理的问题进行介绍;第 3 节阐述对 RFCC-VLIW 结构的调度算法;第 4 节给出实验结果;第 5 节总结全文.

## 2 RFCC-VLIW 结构

VLIW 结构处理器内部包含多个功能单元,在

一个周期内可以发射包含多条指令的指令字,以提高处理器的指令级并行性.多个功能单元要求处理器包含更多的寄存器.但寄存器堆的尺寸过大,以及必须向每个功能单元都提供读写端口,会造成处理器频率下降,面积和功耗的增加.设处理器中功能单元数目为  $N$ ,随着  $N$  的增加,FC-VLIW 结构中寄存器堆的延时、面积和功耗分别以  $N^{3/2}$ 、 $N^3$  和  $N^3$  的比例增加<sup>[9]</sup>.

改善这个问题的方法一般是将大寄存器堆划分成数个小寄存器堆,每个小寄存器堆只可被部分功能单元访问,从而减小每个寄存器堆的尺寸和端口数目.小寄存器堆和可以访问它的功能单元被称为簇.处于不同簇的功能单元需要交换数据时,一般通过连接各簇的总线来完成.这种 BCC-VLIW 结构中寄存器堆及互连总线的延时、面积和功耗分别与  $N$ 、 $N^2$  和  $N^2$  成比例<sup>[9]</sup>.

我们提出的 RFCC-VLIW 结构不同于 BCC-VLIW 用总线连接处理器中的各个簇,而是采用寄存器堆.这个寄存器堆称为互连寄存器堆,它向每个簇提供一组访问端口,所有的功能单元都可以通过复用访问它.当一个簇产生的数据需要被其它簇使用时,就将数据存入互连寄存器堆,其它簇中的功能单元可以直接从互连寄存器堆将数据读出.这样就避免了数据从一个簇传到另一个簇时可能会有的延时,而且也不需要拷贝操作来完成数据传送.图 1 为 RFCC-VLIW 结构的一个例子.

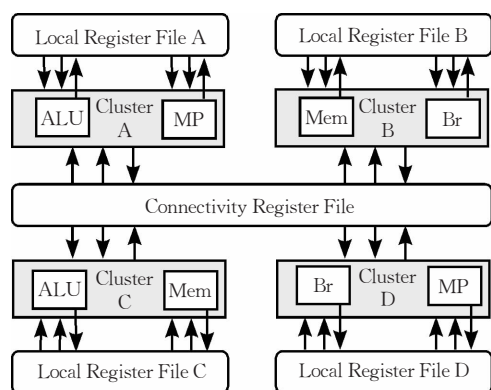


图 1 RFCC-VLIW 结构处理器的一个实例

RFCC-VLIW 结构中,每个簇中的功能单元越多,则本地寄存器的延时、面积和功耗等代价会越大;簇的数目越多,则互连寄存器所需代价越大.我们取簇的数目和每个簇中功能单元的数目都为  $N^{1/2}$ ,因为此时本地和互连寄存器堆的代价近似相同.随着  $N$  的增加,每个簇可以看作一个簇内功能单元数随  $N^{1/2}$  增加的全连通结构,因此,RFCC-

VLIW 结构处理器中寄存器堆面积和功耗的变化为  $(1+N^{1/2})(N^{1/2})^3$ , 这与 BCC-VLIW 相当. 本地寄存器堆的延时为  $(N^{1/2})^{3/2} = N^{3/4}$ , 而互连寄存器堆受到连线延时的影响较大, 连线延时与  $N$  成正比, 因此互连寄存器堆延时与  $N$  成正比. 可见, RFCC-VLIW 结构中寄存器堆的延时、面积和功耗分别与  $N, N^2$  和  $N^2$  成比例.

在 RFCC-VLIW 结构中, 互连寄存器堆仅向每个簇提供一组读写端口, 每个簇的功能单元必须复用这一组读写端口. 这种复用带来了调度算法实现上的问题——互连寄存器冲突, 即处于同一个簇中执行的指令不能够同时利用某一端口访问互连寄存器堆. 以三操作数的指令为例, 若两条指令同一周期在同一个簇中执行, 那么它们对应的操作数不可以同时都是互连寄存器. 下面是这种冲突的一个例子:

SUB A1, A2, G2 (1)

AND A5, A6, G3 (2)

其中, G 开头的寄存器表示是互连寄存器, A 开头的寄存器表示处于本地寄存器堆 A 中. SUB 指令第 2 个源操作数 G2 和 AND 指令第 2 个源操作数 G3 都是互连寄存器, 若 SUB 和 AND 同一周期在同一个簇中执行, 那么它们是冲突的, 这样的操作将产生错误的结果.

在第 3 节中, 将介绍 RFCC-VLIW 结构的调度算法, 以充分利用这种处理器结构的优点, 并解决这种新产生的互连寄存器冲突问题.

### 3 RFCC-VLIW 结构的调度算法

在 RFCC-VLIW 结构中, 不存在数据传输的延时和额外的拷贝操作, 但却存在互连寄存器的冲突, 这种互连寄存器的冲突也会引起处理器性能的下降. 基于 BCC-VLIW 结构的调度算法的目的是尽量减少由于传输延时和数据拷贝带来的处理器性能下降, 那么 RFCC-VLIW 结构的调度算法的目的也就在于尽量减少由于互连寄存器冲突而引起的性能下降. 调度算法流程图如图 2 所示, 主要分为两个部分, 一是簇分配和指令调度, 二是互连寄存器冲突检测和冲突消除.

簇分配和指令调度部分主要参考 UAS 算法. 对于多簇的 VLIW 结构, 合理地进行簇分配非常重要. UAS 算法把簇分配和指令调度合在一个过程之中, 在进行指令调度的同时可以考虑到各簇中的功能单元使用情况, 获得了较好的调度效果. 我们的算

法中簇分配和指令调度部分和 UAS 算法基本相同, 只是不用考虑到是否需要拷贝操作和拷贝操作能否调度. 无需拷贝操作这一点, 使得在调度后代码长度相对全连通结构不会增加. 在这一步完毕后得到调度的中间结果里, 被不同簇中的功能单元使用的操作数就会被分配互连寄存器. 下面我们着重讨论对互连寄存器的冲突检测和消除.

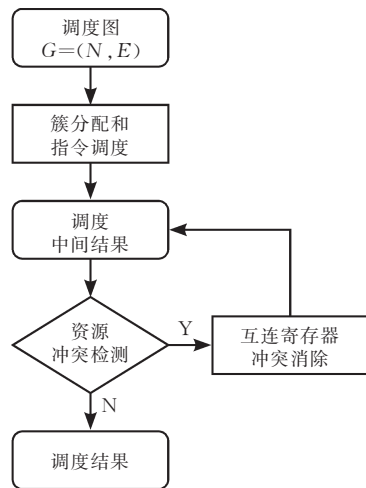


图 2 RFCC-VLIW 调度算法流程图

解决互连寄存器冲突的基本思想, 是按照在调度中间结果里指令执行的周期顺序遍历所有簇, 对每一个周期的每一个簇检测是否有冲突的存在, 若存在, 则根据一定的方法消除冲突; 若不存在, 继续下一个周期或者下一个簇, 直到最后一个周期.

互连寄存器的冲突有两种情况: 源操作数冲突, 目的操作数冲突. 式(1)和式(2)的互连寄存器冲突就属于源操作数冲突. 只要两个属于同一簇的操作在同一周期执行, 就有可能产生源操作数的冲突. 但目的操作数的冲突发生在两条指令写寄存器的周期上, 这就与每条指令产生结果的延时有关. 例如下面的两条指令:

ADD G2, A1, A2 (3)

MUL G3, A4, A5 (4)

ADD 指令产生结果的延时为 1 个周期, MUL 指令产生结果的延时为 2 个周期, 它们的目的操作数都是互连寄存器. 这两条指令在同一周期执行时, 不会产生互连寄存器的冲突; 而仅当 MUL 指令在 ADD 指令的前一个周期执行才会产生冲突.

互连寄存器冲突检测的伪代码如图 3 中进程 CheckResConf 所示. 该进程可以检测出在第  $i$  个周期第  $j$  个簇是否有互连寄存器冲突的存在. 对源操作数的冲突, 只要检查在本周期中该簇里的所有操

作是否有两个或两个以上的对应源操作数都是互连寄存器即可. 对目的操作数, 只需检测与第  $i$  个周期以后指令产生结果之前执行的指令之间的冲突即可, 因为与第  $i$  周期以前的指令的冲突必然已经被遍历过程中以前的步骤所消除.

```
procedure CheckResConf( $i, j$ ) return BOOL
 $i$ : in integer || Number of cycle
 $j$ : in integer || Number of Cluster
 $ops[i][j]$ : set of operations in Cycle  $i$  and Cluster  $j$ 
begin
  || Check source operands conflict
  while not all source operands checked do
    begin
      if at least two corresponding source operands of  $ops[i][j]$  are
        global registers then
        return TRUE;
    end;
  || Check destination operands conflict
  for each  $op1$  in  $ops[i][j]$  do
    for each  $k$  in  $i$  to  $i+op1.latency-1$  do
      for each  $op2$  in  $ops[k][j]$  do
        if both destination operands of  $op1$  and  $op2$  are global regis-
          ter &&.  $k+op2.latency=i+op1.latency$  then
          return TRUE;
      end;
    end;
  end;
  return FALSE;
end; || CheckResConf
```

图 3 互连寄存器冲突检测伪代码

解决互连寄存器冲突的方法有 4 种:

(1) 对于源操作数的冲突, 如果某条指令其源操作数被交换而语义不会改变, 可以尝试通过交换其源操作数的顺序来解决. 例如式(1)和式(2)的互连寄存器冲突, 交换式(2)中的两个源操作数, 变成

AND A5, G3, A6 (5)

那么这两条指令不再存在互连寄存器冲突.

(2) 对于处于非关键路径上的指令, 在调度后可能有一定的活动空间. 调度器总是将指令放在最早可以调度的位置, 这个位置为最早周期. 该指令最早调度的后继指令所在位置的前一个周期, 称为最迟周期. 指令位于其最早和最迟周期之间的任一位置, 不会影响调度长度. 但是这种位置的改变却可能解决互连寄存器冲突问题. 因此, 冲突消除的第二个方法, 就是将冲突的指令放入最早周期的后一个周期和最迟周期之间的位置, 当然, 其所属的簇不能改变, 在不产生新冲突及功能单元资源允许的前提下, 尝试是否可以消除互连寄存器冲突.

(3) 对于所有的操作数都是互连寄存器或者立即数的指令, 其所属簇可以随意改变, 而不影响互连寄存器的分配情况和调度长度. 若这类指令存在冲突, 可以将它放入最早周期和最迟周期期间的任一

置的任一簇中, 以解决互连寄存器的冲突.

(4) 在冲突的位置后插入一个新的周期. 对于源操作数冲突, 插入一个新周期, 并将其中一条指令放入到这个新周期中的相同功能单元中即可. 而对目的操作数冲突, 若两条冲突的指令在同一周期执行, 可以按照与源操作数相同的方法; 若不在同一周期执行, 则只需插入一个空周期, 而不能将这条指令放入新周期中. 这种情形如图 4 所示, 其中 MUL 指令和 ADD 指令都在第 3 个周期写互连寄存器堆, 它们在同一簇中执行则发生冲突. 在第 2 个周期插入空指令后, 如果将 MUL 指令放入第 2 个周期, 那么它们仍然冲突. 当然, 这种插入新周期的解决办法会增加代码运行的周期数, 使处理器的性能下降.

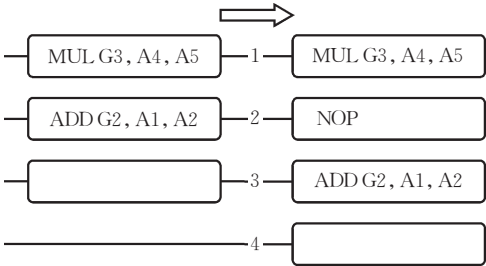


图 4 插入空指令以消除互连寄存器冲突

4 实验结果及分析

我们在 ORC<sup>①</sup> 编译器的框架上实现了 RFCC-VLIW 结构的调度算法, 并以四簇八发射的处理器为目标编译了 UTDSP<sup>②</sup> 的测试程序. 目标 RFCC-VLIW 结构处理器结构如图 1 所示, 包含 8 个功能单元, 算术逻辑单元 (ALU)、乘法单元 (MP)、分支单元 (BR)、内存操作单元 (MEM) 各 2 个, 其中 BR 和 MEM 也可以完成一些算术操作. 我们以图 1 所示的 RFCC-VLIW 结构处理器和具有相同功能单元的 FC-VLIW 处理器为目标, 对这套测试程序用修改的 ORC 编译器进行了编译, 然后将这两种不同结构处理器产生的实验结果进行比较.

图 5 和图 6 分别为 RFCC-VLIW 结构处理器相对 FC-VLIW 结构处理器的性能下降和代码长度的变化. 从图中可以看出, RFCC-VLIW 处理器性能的下降, 大部分在 15% 以内, 平均值仅为 13% 左右. 代码长度的变化都很小, 变化都在 4% 以内, 而且有

① Open research compiler for Itanium processor family, <http://sourceforge.net/projects/open64>  
② Lee C, Stoodley M. UTDSP benchMark suite. <http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.tar.gz>, 1992



几个程序的代码长度甚至有所减少,平均代码长度的增长仅约 0.4%.

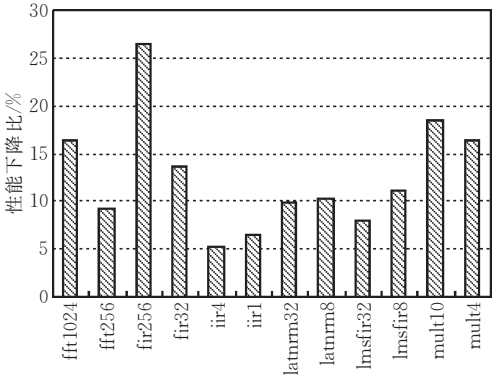


图 5 RFCC-VLIW 处理器相对相应的 FC-VLIW 处理器性能下降百分比

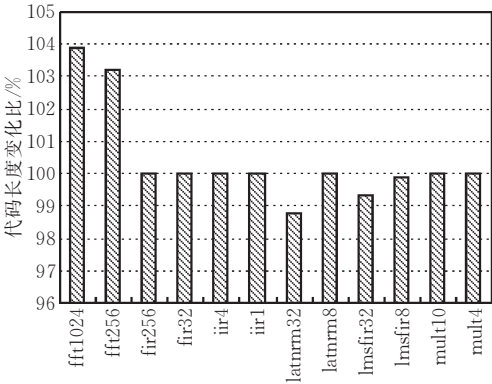


图 6 RFCC-VLIW 处理器相对相应的 FC-VLIW 处理器代码长度变化

从图 6 中可以看出,RFCC-VLIW 有些程序代码长度变长,也有些变得更小,而大部分的测试程序代码长度没有变化.从两种结构的编译过程来看,对 RFCC-VLIW 处理器的编译中增加了本文第 3 节所描述的互连寄存器冲突消除过程,对代码进行了调整——移动指令和插入空指令,但并未增加任何其它的指令.因而代码长度变长是由于在这一过程中插入了一些空指令.而代码长度变小则与处理器的结构和编译器的优化有关.首先,由于 ORC 目标处理器和 RFCC-VLIW 处理器结构上的差异,在 ORC 编译器中的有些优化手段,如函数调用时的参数传输机制等,在我们的编译过程中未能应用,故出现了较多的空指令.其次,处理器的指令集中并未设计连续空指令(如 NOP 3 表示 3 条空指令),当连续多个周期不执行有效操作时,需要有多条空指令.因此,在互连寄存器冲突消除的过程中当有些指令在从一个周期移到另一个周期时,若是替代了空指令,会导致这条空指令被消除,从而减少了总的代码长度.而

且,这种代码长度的减小程度甚微,仅 1% 左右.

表 1 给出了以 FC-VLIW 结构为标准,RFCC-VLIW 处理器和 BCC-VLIW 处理器性能和代码长度<sup>[6-8]</sup>的比较.从表中可以看出,本文提出的 RFCC-VLIW 结构及相应的调度算法的指令调度结果,在性能和代码长度方面都优于 BCC-VLIW 结构的各种算法.

表 1 RFCC-VLIW 和 BCC-VLIW 调度结果的比较				
结构	互连方式	调度算法	性能下降比/%	代码长度增加比/%
BCC-VLIW	总线互连	LCVLIW	57~69	-
		BUG	32	-
		UAS	19	11.0
RFCC-VLIW	寄存器互连		13	0.4

BCC-VLIW 结构处理器相对 FC-VLIW 结构性能上的下降是由于簇间数据传输的延时所引起;而 RFCC-VLIW 结构处理器在性能上的下降是由于互连寄存器堆的冲突所引起的. BCC-VLIW 结构簇间数据传输的延时是不可避免的,只能通过调度手段来隐藏这种延时,但有时这种延时是无法隐藏的(如关键路径上的指令).而 RFCC-VLIW 结构可以通过合适的指令调度和分配,复用互连寄存器堆提供的端口,尽量减少这种冲突的发生,从而减少性能的下降.因此,本文提出的 RFCC-VLIW 结构及相应的调度算法获得了更好的处理器性能.

BCC-VLIW 结构相对 FC-VLIW 结构代码长度上的增加是由簇间数据传输的拷贝操作所引起;而 RFCC-VLIW 结构在代码长度上的变化则是由于互连寄存器冲突消除过程中的指令调整所引起. RFCC-VLIW 结构在簇间数据传输无需拷贝操作的结构特点决定了在代码长度的变化上要明显优于 BCC-VLIW 结构处理器.

## 5 结 论

本文提出了一种新的多簇 VLIW 处理器结构——寄存器堆互连的多簇 VLIW. 这种结构具有簇间数据传输无延时、无需拷贝操作的优点,寄存器堆的延时、面积和功耗的增加和总线互连的 BCC-VLIW 结构相当,其代价是引起了互连寄存器冲突问题. 本文又提出了针对这种结构的指令调度算法,以充分利用这种结构的优点,同时尽量避免由于互连寄存器冲突而引起的性能下降. 实验结果表明,采用本文提出的 RFCC-VLIW 处理器配合本文所述的调度算法,在性能上的下降和代码长度的增加都

明显优于 BCC-VLIW 结构效果最好的 UAS 算法.

## 参 考 文 献

- [1] Joseph A. Fisher, Paolo Faraboschi and Cliff Young. Embedded computing. California: Morgan Kaufmann, 2005
- [2] Texas Instrument Inc. TMS320C62x/67x CPU and instruction set reference guide. 1998
- [3] Fridman J, Greefield Z. The TigerShare DSP architecture. IEEE Micro, 2000, 20(1): 66-76
- [4] Zhang Yan-Jun, He Hu, Sun Yi-He. A new register file access architecture for software pipelining in VLIW processors// Proceedings of the ASP-DAC. Shanghai, 2005, 1: 627-630
- [5] Faraboschi P, Finsher J A, Young C. Instruction scheduling for instruction level parallel processors. Proceedings of the

IEEE, 2001, 89(11): 1638-1659

- [6] Ellis J R. Bulldog: A Compiler for VLIW Architectures. London: The MIT Press, 1986
- [7] Capitanio A, Dutt N, Nicolau A. Partitioned register files for VLIWs: A preliminary analysis of tradeoffs//Proceedings of the 25th International Symposium on Microarchitecture, 1992: 292-300
- [8] Ozer E, Banerjia S, Conte T M. Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures//Proceedings of the 31st International Symposium on Microarchitecture. Dallas, TX, 1998: 308-315
- [9] Rixner S, Dally W J, Khailany B et al. Register organization for media processing//Proceedings of the 6th International Symposium on High-Performance Computer Architecture. Toulouse, 2000: 375-286



**ZHOU Zhi-Xiong**, born in 1979, Ph.D. candidate. His research interests include auto-design and compiler technology of media specific processors.

**HE Hu**, born in 1974, assistant researcher. His research interests include DSP design methodology, LSI/VLSI

test methodology and design for test.

**YANG Xu**, born in 1982, Ph.D. candidate. His research interests focus on compiling for energy reduction.

**ZHANG Yan-Jun**, born in 1978, Ph.D. candidate. His research interests focus on media specific processor design.

**SUN Yi-He**, born in 1945, professor, Ph.D. supervisor. His research interests include LSI/VLSI test methodology, design for test and media specific VLSI design technology.

## Background

This work is supported by the National Natural Science Foundation under grant No. 60236020. VLIW architecture is adopted in most high performance media processors, such as TI C6000 series. Generally they are implemented as clustered VLIW so as to achieve less area and lower power. The authors present a new clustered VLIW architecture which can give comparative area and power but higher performance and

shorter code size. Up to now the authors have finished designing of a DSP based on this new architecture. Performance of VLIW processors depends on the compiler, so it is necessary to make research on compiler technology aimed to this new architecture. This paper gives overview and analysis of this new architecture, and then presents instruction scheduling algorithm to improve the performance.