申威 411 处理器
高级用户手册

2017 年 10 月

成都申威科技有限责任公司

成都市华府大道四段电子科大科技园 D22 栋
Building D22, National University Science and technology park,
Section 4, Huafu Avenue, Chengdu

免责声明

本文档仅提供阶段性信息，所含内容可根据产品的实际情况随时更新，恕不另行通知。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

成都申威科技有限责任公司

Chengdu Sunway Technology Corporation Limited

地址：成都市华府大道四段电子科大科技园 D22 栋

Building D22, National University Science and technology park,
Section 4, Huafu Avenue, Chengdu

Mail：sales@swcpu.cn

Tel ： 028-68769016

Fax：028-68769019

阅读指南

《申威 411 处理器数据手册》主要描述了申威 411 处理器程序员优化方法、管理员优化方法、编译优化方法、调试支持等内容。

文档修订

文档更新记录	文档名	申威 411 处理器高级用户手册
	版本号	V1.0
	创建人	研发部
	创建日期	2017-10-8

版本更新

版本号	更新内容	更新日期
V1.0	初稿	2017-10-8

技术支持

可通过邮箱或问题反馈网站向我司提交产品使用的问题，并获取技术支持。

售后服务邮箱：sales@swcpu.cn

问题反馈网址：http://www.swcpu.cn/

目　　录

1 程序员优化

申威411处理器的一部分指令在具体实现时，结合处理器的微结构，进行了适当的改进和优化，在此对这些指令及其具体实现进行更详细的补充说明，以便程序员进行代码优化。

1.1 转移指令

申威411处理器支持条件转移、无条件转移、跳转指令三类转移指令，其中条件转移和无条件转移使用转移指令格式，跳转指令使用存储器指令格式。为了提高程序运行的效率，申威411处理器实现了编译器通过转移指令格式对转移预测进行显式提示：可能的转移目标指令地址、返回指令地址栈操作和条件转移的方向。无条件转移指令的转移方向是明确的，因此不需要对转移预测进行提示。

1.1.1 跳转指令

对于跳转类指令，指令中携带了16位偏移值（disp[15:0]），申威411处理器在进行具体实现时，将16位的偏移用于预测跳转目标指令的地址，预测的方法是：将跳转指令的顺序地址（NextPC）加上偏移值的符号扩展，作为下一次取指的PC。表1-1为这类指令与转移预测的关系。

表 1-1：跳转指令与转移预测的关系

助记符	转移目标地址预测	返回指令地址栈操作
JMP	NextPC+4*SEXT(disp[15:0])	—
CALL	NextPC+4*SEXT(disp[15:0])	原 PC 值压栈
RET	返回指令地址堆栈	新 PC 值从栈顶弹出

这些指令所完成的操作完全相同，唯一的区别在于对可能的转移预测逻辑的提示不同。16位偏移值可用于转移目标指令地址的预测，合理设置能够提高性能，但并不是正确运行所必须的。软件在进行编译优化时，如果能确定跳转的目标地址，建议将转移目标地址与NextPC的差值填入到跳转指令的16位偏移中，以便提高跳转指令的转移预测成功率。

1.1.2 条件转移指令

对于条件转移指令，指令中携带了 21 位有符号的指令偏移（disp[20:0]），可在前后 1M 条指令范围内实现跳转。申威 411 处理器在装填指令 Cache（ICache）时，如果检测到条件转移指令，会根

据条件转移的方向初始化用于判断预测转移是否发生的饱和计数器初值。转移指令的偏移为正数时（disp[20]为“0”），饱和计数器初始化为“01”，偏移为负数时（disp[20]为“1”），饱和计数器初始化为“10”。当饱和计数器的值是“10”或“11”时，转移预测器将预测转移发生，当饱和计数器的值是“00”或“01”时，转移预测器将预测转移不发生。

另外，申威 411 处理器核心在硬件上支持指令预取，即始终按程序的顺序地址进行预取，每次预取 32 条指令。为了使指令预取能有效地提高指令 Cache 的命中率，可以通过重排代码，使顺序路径成为条件转移指令的两条执行路径中经常选择的路径。

由于申威 411 处理器核心采用动态转移预测，重排代码不会影响转移预测的性能，因此基于上述原则，建议将代码按如下方式进行重排序：

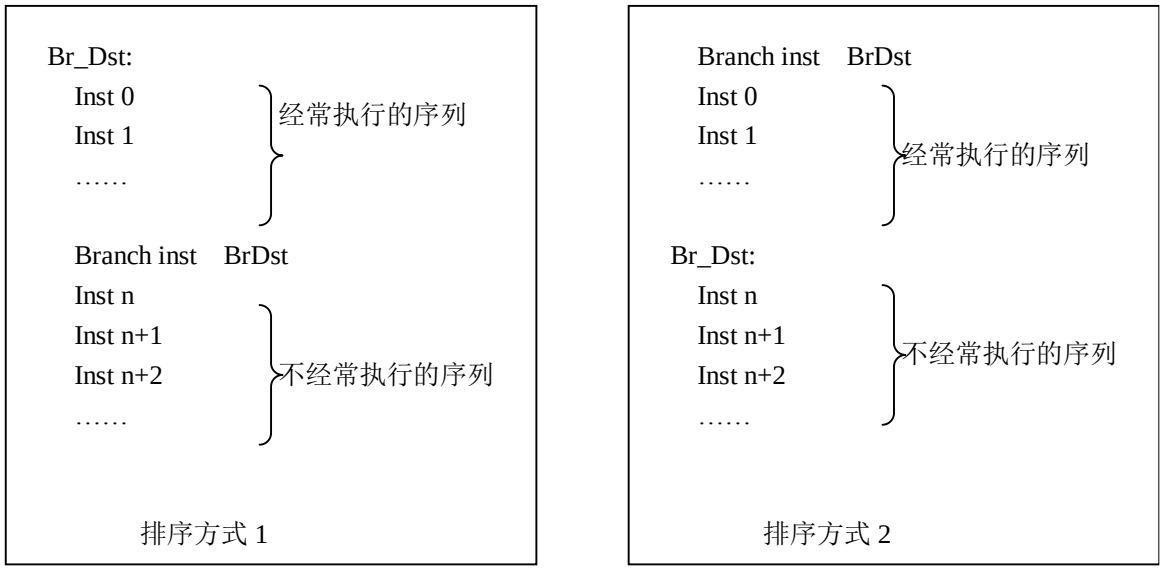


图 1-1：指令序列排序示意图

1） 排序方式 1：将经常执行的分支安排在往后跳的路径，不经常执行的路径安排在顺序路径；

2） 排序方式 2：将经常执行的分支安排在顺序路径，将不经常执行的路径安排在往前跳路径。

两种排序方式都可以适当提高指令执行的效率，具体采用哪种排序方式可根据程序的执行特征来选择。

1.2 Cache 控制指令

申威411处理器中，当指令的目标寄存器为R31、F31或V31时一般当作空指令处理，但是部分存储器装入指令目标寄存器为R31、F31或者V31时，作为Cache控制指令，如表1-2所示。

表 1-2：Cache 控制指令列表

序号	指 令	操 作 码	描 述
1	LDBU	0x20	目标寄存器为 R31 时，实现 Cache 行刷新功能

			（FLUSHD）。
2	LDHU	0x21	目标寄存器为 R31 时，实现将指定地址的 Cache 行淘汰出所有核心私有 Cache 功能（EVICTDG）。
3	LDW	0x22	目标寄存器为 R31 时，作为装填提示指令，提示将数据装填到核心二级 Cache，且为清洁共享状态（S_FILLCS）。
4	LDL	0x23	目标寄存器为 R31 时，作为装填提示指令，提示将数据装填到核心二级 Cache，且为脏独占状态（S_FILLDE）。
5	LDL_U	0x24	目标寄存器为 R31 时，实现将指定地址的 Cache 行淘汰出本核心功能（EVICTDL）。
6	FLDS	0x26	目标寄存器为 F31 时，作为装填提示指令，提示将数据装填到核心一级 Cache，且为脏独占（FILLDE）。
7	FLDD	0x27	目标寄存器为 F31 时，作为装填提示指令，提示将数据装填到核心一级 Cache，且为脏独占，且标记为优先淘汰（FILLDE_E）。
8	LDWE	0x09	目标寄存器为 F31 时，作为装填提示指令，提示将数据装填到核心一级 Cache，且为清洁共享（FILLCS）。
9	LDSE	0x0A	目标寄存器为 F31 时，作为装填提示指令，提示将数据装填到核外 Cache，且为清洁共享（E_FILLCS）。
10	LDDE	0x0B	目标寄存器为 F31 时，作为装填提示指令，提示将数据装填到核心一级 Cache，且为清洁共享，且标记为优先被淘汰（FILLCS_E）。
11	VLDS	0x0C	目标寄存器为 F31 时，作为装填提示指令，提示将数据装填到核外 Cache，且为脏独占（E_FILLDE）。

格式： 助记符 disp.ab(Rb.ab) ！存储器指令格式

操作： CASE

S_FILLCS: LDW R31, disp (Rb)

S_FILLDE: LDL R31, disp (Rb)

FILLDE: FLDS F31, disp (Rb)

FILLDE_E: FLDD F31, disp (Rb)

FILLCS: LDWE F31, disp (Rb)

```
FILLCS_E:       LDDE   F31, disp (Rb)
E_FILLCS:       LDSE   F31, disp (Rb)
E_FILLDE:       VLDS   F31, disp (Rb)
FLUSHD:         LDBU   R31, disp (Rb)
EVICTDL:        LDL_U  R31, disp (Rb)
EVICTDG:        LDHU   R31, disp (Rb)
ENDCASE
```

异常： 跟具体实现相关

助记符：

S_FILLCS	提示装填数据进核心二级Cache，为清洁共享
S_FILLDE	提示装填数据进核心二级Cache，为脏独占
FILLDE	提示装填数据进核心一级Cache，为脏独占
FILLDE_E	提示装填数据进核心一级Cache，为脏独占，并标记为优先淘汰
FILLCS	提示装填数据进核心一级Cache，为清洁共享
FILLCS_E	提示装填数据进核心一级Cache，为清洁共享，并标记为优先淘汰
E_FILLCS	提示装填数据进核外Cache，为清洁共享
E_FILLDE	提示装填数据进核外Cache，为脏独占
FLUSHD	Cache行刷新，具体定义与C3实现相关
EVICTDL	将指定地址的Cache行淘汰出本核心私有Cache
EVICTDG	将指定地址的Cache行淘汰出所有核心私有Cache

指令描述：

表1-2中所列出的指令是将部分装入指令在目标寄存器为R31、F31或V31时，当作Cache控制指令，按功能可大致分为Cache装填（预取）、Cache刷新和Cache淘汰三类，合理使用该类指令可以提升存储访问性能。另外，当Cache控制指令到达访存部件时，也可能因某种原因被当作空指令，具体要根据访存部件内部状态来决定。

Cache装填指令是对处理器的一个提示，即一个Cache行可能在将来要访问，提前将其装填到核心一级Cache（DCache）、二级Cache（SCache）或核外Cache（ECache）中，以提高各级Cache命中率，降低平均访存延时，提高有效访存带宽。如果Cache装填指令的地址映射到I/O空间或到不可Cache空间，则该指令被当作空指令处理。Cache装填指令访问数据流地址转换缓冲（DTB）发生脱靶时，是否进入脱靶处理流程受CSR：DC_CTL[F_DTB_MISS]控制，2.2.1节中有详细描述。Cache装填指令在执行中遇到其它存储管理异常，则不产生任何异常自陷，直接被当作空指令处理。

Cache刷新指令不进行地址转换，也不进行地址符合扩展检查和对界检查，但收到非法响应时，可以报数据流故障。C3中，FLUSHD指令用刷新地址的Vaddr[29]位进行区分，为“0”则表示为二级Cache刷新指令，用于刷新二级Cache（SCache）中指定路和指定行；为“1”则表示为核外Cache刷新指令，用于刷新核外Cache（ECache）中指定路和指定行。由于申威411处理器的每个核心中的二

级Cache（SCache）容量为512KB，组成为8路组相联结构，因此二级Cache刷新指令的地址译码为：Vaddr[18:16]表示淘汰的SCache组号，Vaddr[15:7]表示淘汰的SCache行号，其它位没有意义。当CSR：SC_CTL[ECACHE_FLUSH_En]为“1”时，核外Cache刷新指令有意义，其地址由核外根据ECache的容量和相联度进行定义；（4A芯片中核外ECache的容量为6MB，因此地址译码为：Vaddr[24:20]表示淘汰的ECache组号，Vaddr[19:7]表示淘汰的ECache行号，其它位没有意义。）当CSR：SC_CTL[ECACHE_FLUSH_En]为“0”时，核外Cache刷新指令被当作空指令处理。

Cache 淘汰指令不进行地址对界检查，产生其它异常时，是否进入异常处理流程受 CSR：DC_CTL[EVICTD_ERR_EN]控制。如果允许这类指令产生异常，则在产生存储管理异常时，进入相应的异常特权程序，否则该指令被当作空指令处理。如果这类指令的地址映射到 I/O 空间或不可 Cache 空间，则该指令被当作空指令处理。另外需要注意的是，当满足以下条件之一时，浮点指令将产生浮点屏蔽故障（FEN）：

1）CSR：UPCR[FPE]和 CSR：UPCR[FCE]中任意一位为“0”，执行浮点指令或 SIMD 扩展指令；

2）CSR：UPCR[SCE]为“0”，执行 SIMD 扩展指令。

只有合法且非空指令才会产生浮点屏蔽故障，上述目标寄存器为F31的浮点指令被当作Cache控制类指令，则不产生浮点屏蔽故障。

针对一级Cache装填请求，如果DBOX收到带ECC多错标志的响应，则不设置CSR：DC_STAT[MEM_MERR]为“1”，也不进入数据流故障处理入口执行，只是装填DCache，并正常执行；如果DBOX收到带控制错标志的响应，则不设置CSR：DC_STAT[ACK_ERR]为“1”，不进入机器检查错中断处理入口执行，而是正常执行。针对二级Cache装填请求，如果SBOX收到带ECC多错标志的响应或带控制错标志的响应，也不在任何CSR中登记错误，只装填SCache。

硬件实现时，并没有保证Cache刷新和淘汰指令严格按指令的顺序执行，因此当为如下指令序列时，可能出现淘汰不干净的情况：

Load/store r,pa

evictdl pa

如果软件需要确保淘汰干净，可在淘汰指令之前增加存储栏栅指令。

2 管理员优化

2.1 大页模式优化说明

申威 411 处理器的指令流和数据流地址转换表（TLB）中常用的页面粒度是 8KB（小页模式），额外还提供了 256KB、8MB 和 256MB 三种页面粒度（大页模式）。

在执行某些指令流地址或数据流地址跨度比较大的课题时，操作系统可利用申威 411 处理器提供的大页模式，选择适当的页面粒度进行页面管理，以降低虚实地址转换时 TLB 的脱靶率，提高程序运行效率。

具体实现时，申威 411 处理器在指令流或数据流地址转换缓冲的每个条目中隐式存放两位页面粒度指示位，并在地址转换缓冲装填和查询过程中，使用该页面粒度指示位进行比较判断。

2.1.1 指令流地址转换

2.1.1.1 粒度指示位

在 CSR：ITB_PTE 中增加 2 位粒度指示位（GH[1:0]），如下表 2-1。

表2-1：ITB_PTE寄存器域补充描述

名 称	范围	类型	描 述
GH[1:0]	[6:5]	WO	页面粒度，值为“00”、“01”、“10”或“11”，分别指示该页为1、32、1024或32768个连续的8KB页（分别对应8KB、256KB、8MB或256MB地址对界的连续空间）。

同时，在指令地址转换缓冲（ITB）的条目中，也增加2位粒度指示位（GH[1:0]），含义与CSR：ITB_PTE[GH]相同。

2.1.1.2 地址匹配

写 CSR：ITB_IS 刷新 ITB 中指定条目时，将 CSR：ITB_IS 与 ITB 中所有有效条目的虚地址标记进行比较，比较的位数要根据粒度（GH[1:0]）指示位来确定，即 GH 为“00”、“01”、“10”或“11”时，比较的位数分别为 ITB_IS [42:13]、[42:18]、[42:23]或[42:28]位。

执行指令流虚实地址转换时，将指令流虚地址（IVA）与 ITB 中所有有效条目的虚地址标记进行比较，比较的位数也要根据粒度 GH[1:0]来确定，即 GH 为“00”、“01”、“10”或“11”时，比较的位数分别为 IVA[42:13]、[42:18]、[42:23]或[42:28]位。

当 ITB 命中时，根据命中条目的粒度 GH[1:0]来获得物理地址（IPA），具体方法如下：

1） 若 GH 为“00”，则 IPA[38:0]由 ITB 中 PA[38:13]和 IVA[12:0]组成；

2） 若 GH 为“01”，则 IPA[38:0]由 ITB 中 PA[38:18]和 IVA[17:0]组成；

3） 若 GH 为“10”，则 IPA[38:0]由 ITB 中 PA[38:23]和 IVA[22:0]组成；

4） 若 GH 为“11”，则 IPA[38:0]由 ITB 中 PA[38:28]和 IVA[27:0]组成。

2.1.2 数据流地址转换

2.1.2.1 粒度指示位

在 CSR：DTB_PTE 中增加 2 位粒度指示位（GH[1:0]），如下表 2-2。

表2-2：DTB_PTE寄存器域补充描述

名 称	范围	类型	描 述
GH[1:0]	[6:5]	WO	页面粒度，值为“00”、“01”、“10”或“11”，分别对应1、32、1024或32768个连续且对界的8KB页（分别对应8KB、256KB、8MB或256MB地址对界的连续空间）。

同时，在一级数据地址转换缓冲（L1DTB）的条目中，也增加 2 位粒度指示位（GH[1:0]），含义与 CSR：DTB_PTE[GH]相同。二级数据地址转化缓冲（L2DTB）由两个地址表组成，一个是 512 条目的小页表（L2SDTB），一个是 32 条目的大页表（L2BDTB），其中 L2BDTB 的条目中，也增加 2 位粒度指示位（GH[1:0]），其值只能为“01”、“10”或“11”，分别对应 32、1024 或 32768 个连续且对界的 8KB 页（分别对应 256KB、8MB 或 256MB 地址对界的连续空间）。

需要注意的是：当装填的 DTB 条目的地址和粒度与两级 DTB 中已有条目的地址和粒度有重叠时，硬件不保证将原 DTB 中的条目删除，因此软件必须保证：当需要更改某个页面的粒度或虚实地址对应关系时，必须对 DTB 进行软件刷新，否则硬件在进行 DTB 访问时，可能因命中多个条目而出错。

2.1.2.2 地址匹配

写 CSR：DTB_IS 刷新 DTB 中指定条目时，将 CSR：DTB_IS 与 DTB 中所有有效条目的虚地址标记进行比较，比较的位数要根据粒度（GH[1:0]）指示位来确定，即 GH 为“00”、“01”、“10”或“11”时，比较的位数分别为 DTB_IS [42:13]、[42:18]、[42:23]或[42:28]位。

执行数据流虚实地址转换时，将数据流虚地址（DVA）与 DTB 中所有有效条目的虚地址标记进行比较，比较的位数也要根据粒度 GH[1:0]来确定，即 GH 为“00”、“01”、“10”或“11”时，比较的位数分别为 DVA[42:13]、[42:18]、[42:23]或[42:28]位。

当 DTB 命中时，根据命中条目的粒度 GH[1:0]来获得物理地址（DPA），具体方法如下：

1） 若 GH 为“00”，则 DPA[39:0]由 DTB 中 PA[39:13]和 DVA[12:0]组成；

2） 若 GH 为“01”，则 DPA[39:0]由 DTB 中 PA[39:18]和 DVA[17:0]组成；

3） 若 GH 为“10”，则 DPA[39:0]由 DTB 中 PA[39:23]和 DVA[22:0]组成；

4） 若 GH 为“11”，则 DPA[39:0]由 DTB 中 PA[39:28]和 DVA[27:0]组成。

2.2 Cache 控制指令的访问特性

本章对Cache控制指令的访问特性进行补充说明，包括对存储器空间和I/O空间的访问特性，以及访问合并原则，另外，还对如何使用Cache控制指令实现申威411处理器的数据流访问容错功能进行说明。

2.2.1 DC_CTL 补充定义

CSR：DC_CTL为数据Cache控制寄存器，在该寄存器中补充定义Cache淘汰相关的控制信息位，如下表2-3。

表2-3：DC_CTL寄存器域补充描述

名　称	范围	类型	描　述
F_DTB_MISS	[16]	RW,0	强制装填指令处理DTB miss；该位有效时，装填指令访问DTB发生脱靶时，需要进入脱靶处理流程。
EVICTD_ERR_EN	[5]	RW,0	Cache淘汰错误检查使能。

CSR：DC_CTL[EVICTD_ERR_EN]位只对 Cache 淘汰指令（EVICTDL 和 EVICTDG）有影响，当 Cache 淘汰指令在地址转换过程中产生存储管理故障时，或在向核外发出淘汰请求后收到错误的响应时，如果该位为“0”，则 Cache 淘汰指令仍然正常退出；否则 Cache 淘汰指令进入异常处理流程。

2.2.2 访问特性

Cache 控制指令可以对存储器空间进行访问，对于 FLUSHD 指令，总是访问可 Cache 的存储器空间，PA[39:37]无意义；对于 Cache 装填指令和 Cache 淘汰指令，如果 PA[39]为“1”或 PA[38:37]不等于本核组的编号，则当作空指令。

Cache 控制指令不可以对 I/O 空间进行访问，如果 Cache 控制指令访问的数据流地址为 I/O 空间，则作为空指令。

2.2.3 访问合并原则

处理对可 Cache 存储器空间的访问时，新产生的请求与之前产生且未完成处理的请求进行地址比较，如果满足条件则可将两个请求合并。具体规则见表 2-4。

表 2-4：访问共享可 Cache 空间的合并规则

		已产生且未完成处理的请求					
		LDx	**STx**	**FETCHDx**	**FLUSHD**	**EVICTDx**	**指令流**
新产生的请求	**LDx**	合并	合并	合并	—	—	—
	STx	合并	合并	合并	—	—	—
	FETCHDx	合并	合并	合并	—	—	—
	FLUSHD	—	—	—	—	—	—
	EVICTDx	—	—	—	—	—	—
	指令流	—	—	—	—	—	合并

说明：

1） 已有的请求相关的指令与新产生的请求相关的指令是按程序顺序定义的；

2） 表中指示“合并”的位置，表示两个请求可以合并；

3） 表中指示“—”的位置，表示不存在或不允许请求合并。

4） 锁装入和锁存储指令产生的访存请求不允许与任何访存请求合并。

2.2.4 容错处理

数据 Cache 的地址标记（DTag）采用偶校验，当处理 Load/Store 类指令时，对从数据 Cache 中读出的地址标记进行偶校验，若产生校验错，则在 CSR：DC_CTL[DCTAG_PAR_EN]为“1”时，设置 CSR：DS_STAT[TAG_PERR]为“1”。该错误硬件无法纠正，需要进入数据流故障（DFAULT），由特权程序来进行错误恢复。

1） 特权程序读 CSR：DS_STAT，确定引起 DFAULT 的原因，如果不存在其它数据流故障且 DS_STAT[TAG_PERR]为“1”，则可以确定是数据 Cache 的地址标记偶校验错；

2） 设置 CSR：DC_CTL[DCTAG_PAR_EN]为“0”，关闭数据 Cache 的地址标记校验使能；

3） 再从 CSR：DVA 中获得出现错误的数据流物理地址；

4） 根据这个物理地址[15:7]位，形成 8 个二级 Cache 索引地址（地址[15:7]位）相同，二级 Cache 组号不同的淘汰地址，使用 FLUSHD 指令将出现故障的数据 Cache 内容淘汰到主存。

由于出现地址标记偶校验错的数据 Cache 行被淘汰时，由二级 Cache 的地址标记（STAG）提供地址信息，因此数据 Cache 的地址标记偶校验错被清除，且对应的 Cache 数据不会受到破坏。

2.3 事件统计机制

事件统计是对程序运行过程中发生的特定事件进行统计，给用户提供程序运行信息，以达到调试、修改、优化程序，提高程序执行效率的目的。

申威 411 处理器为每个核心设置 2 个事件统计器，在相关 CSR 的控制下，实现对申威 411 处理器内部各种事件的计数，计数器溢出可分别产生事件统计 1 中断和事件统计 2 中断。计数器的初值、计数使能控制、计数的事件选择都通过 CSR 来控制。系统通过事件统计中断服务程序，实现事件统计功能。

2.3.1 相关的 CSR

表 2-5 与表 2-6 分别给出了与事件统计相关的 CSR 的索引及定义。

表 2-5：与事件统计相关的 CSR 的索引

序号	名称	符号	索引	记分板位	存取特性
1	中断使能寄存器	IER3	0011 1011	0	RW
2	中断状态寄存器	INT_STAT3	0011 0011	0	RO
3	指令流控制寄存器	IS_CTL	0001 0001	0	RW
4	事件统计与控制寄存器 0	PC0_CR	0010 0100	0	RW
5	事件统计与控制寄存器 1	PC1_CR	0010 0101	0	RW
6	中断向量映射寄存器	INT_VEC	0001 1111	0	RW

其中，IER3、INT_STAT3、IS_CTL、INT_VEC 是在原有的 CSR 上进行补充定义，PC0_CR 与 PC1_CR 是为事件统计新增的 CSR，这两个 CSR 缺省使用的记分板位和存取特性等与申威 411 处理器中其它指令部件内的 CSR 一样。

表 2-6：与事件统计相关的 CSR

序号	符号	相关的域	含义
1	IER3	PCEN[1:0]	事件统计中断使能。
2	INT_STAT3	PC[1:0]	事件统计中断请求。该域通过 INT_VEC 进行定义。
3	IS_CTL	PC_CM[1:0]	指定进行事件统计的处理器模式。
		CM_PCE	在指定处理器模式下事件统计使能。
		AM_PCE	在任何处理器模式下事件统计使能。

		PC0_EN	事件统计器 0 使能。
		PC1_EN	事件统计器 1 使能。
4	PC0_CR	PC0_SEL[3:0]	选择事件统计器 0 的计数事件。
		PC0[57:0]	事件统计器 0，可设置初值。
5	PC1_CR	PC1_SEL[5:0]	选择事件统计器 1 的计数事件。
		PC1[57:0]	事件统计器 1，可设置初值。
6	INT_VEC	PC0_VEC[5:0]	指定事件统计 0 中断对应的中断向量位。
		PC1_VEC[5:0]	指定事件统计 1 中断对应的中断向量位。

2.3.1.1 IER 与 INT_STAT

CSR：INT_STAT3 作为核心中断状态寄存器，也用于记录申威 411 处理器的核心收到的事件统计中断 1 和事件统计中断 0。IER3 作为核心中断使能寄存器，也包括事件统计中断 1 和事件统计中断 0 的使能位，具体位置与 CSR：INT_STAT3 一一对应，可以由软件通过写 CSR：INT_VEC 来指定。下表 2-7 给出了复位结束后，事件统计中断在 CSR：INT_STAT3 中对应的位置。

表2-7：INT_STAT3寄存器复位后对应的中断

名称	范围	类型	描述
SLEEP	[63]	RO	睡眠中断请求。
MCHK_INT	[62]	RO	机器检查错中断请求。
CR	[61]	RO	已纠正错中断请求。
PC1	[60]	RO	事件统计1中断请求。
PC0	[59]	RO	事件统计0中断请求。
TIMER_INT	[58]	RO	定时器中断。
其它	—	—	保留。

2.3.1.2 IS_CTL

CSR：IS_CTL 中补充定义了一些与事件统计相关的控制信息位，具体定义如下表 2-8。

表2-8：IS_CTL寄存器域补充描述

名　称	范围	类型	描　述
PC1_EN	[15]	RW,0	事件统计器1使能。
PC0_EN	[14]	RW,0	事件统计器0使能。
PC_CM	[13:12]	RW,0	指定在何种模式下进行事件统计：“00”为硬件模式；“01”为虚拟模式；“10”为核心模式；“11”为用户模式。

CM_PCE	[11]	RW,0	指定模式事件统计使能。
AM_PCE	[10]	RW,0	任何模式事件统计使能。
其它	—	—	保留。

其中，PC0_EN、PC1_EN、AM_PCE、CM_PCE、PC_CM 是事件统计相关的控制位，当 PC0_EN 为“1”，且 AM_PCE 为“1”或 CM_PCE 为“1”时，事件统计器 0 使能；当 PC1_EN 为“1”，且 AM_PCE 为“1”或 CM_PCE 为“1”时，事件统计器 1 使能。AM_PCE 和 CM_PCE 不能同时为“1”，如果同时为“1”，则按所有模式进行事件统计。

当 CM_PCE 为“1”时，硬件只对 PC_CM 中指定的模式下发生的事件统计。

2.3.1.3 PC0_CR

CSR：PC0_CR为事件统计与控制寄存器0，可读写。用于读写事件统计器0的计数值和设置事件统计器0的计数事件。软件在复位时需将该寄存器清为“0”，具体定义如下表2-9。

表2-9：PC0_CR寄存器域描述

名　称	范围	类型	描　述
PC0_SEL[3:0]	[63:60]	RW	事件统计器0的计数事件选择，具体见表2-12。
PC0[57:0]	[57:0]	RW	事件统计器0。
其它	—	—	保留。

2.3.1.4 PC1_CR

CSR：PC1_CR为事件统计与控制寄存器1，可读写。用于读写事件统计器1的计数值和设置事件统计器1的计数事件。软件在复位时需将该寄存器清为“0”，具体定义如下表2-10。

表2-10：PC1_CR寄存器域描述

名　称	范围	类型	描　述
PC1_SEL[5:0]	[63:58]	RW	事件统计器1的计数事件选择，具体见表2-13。
PC1[57:0]	[57:0]	RW	事件统计器1。

2.3.1.5 INT_VEC

CSR：INT_VEC 作为中断向量映射寄存器，也用于记录事件统计 1 中断和事件统计 0 中断在 INT_STAT 中的向量位置，与事件统计相关的信息具体定义如下表 2-11。

表2-11：INT_VEC寄存器域补充描述

名称	范围	类型	描述

PC0_VEC[5:0]	[29:24]	RW,0xFB	指定事件统计0中断对应的向量。
PC1_VEC[5:0]	[23:18]	RW,0xFC	指定事件统计1中断对应的向量。

2.3.2 事件统计中断

申威 411 处理器设置了两个事件统计中断，用于实现事件统计。当硬件对指定模式下的特殊事件进行事件统计并产生溢出时，产生事件统计 0 中断或事件统计 1 中断，如果 CSR：IER 中对应的事件统计中断使能，则硬件自动在 CSR：INT_STAT 中登记，登记的位置由 CSR：INT_VEC[PC0_VEC]或 CSR：INT_VEC[PC1_VEC]决定，并进入一般中断处理特权程序入口 1（INTERRUPT1）。当事件统计中断服务程序处理结束后，软件可通过写 CSR：INT_CLR 中对应位来清除事件统计 0 或事件统计 1 中断。

事件统计中断的中断优先级可归为一般中断。

2.3.3 事件统计的模式

事件统计分成 2 类，一是所有处理器模式下的事件统计，即在任何模式下都对所选事件进行计数；二是指定处理器模式下的事件统计，即在指定处理器模式下对所选事件进行计数。两者的控制有所不同，前者的事件统计使能由 CSR：IS_CTL[AM_PCE]决定，后者则由 CSR：IS_CTL[CM_PCE]决定。一般两类事件统计只允许一种事件统计使能，否则事件统计的结果不能正确反映程序运行的情况。

申威 411 处理器设置了两个事件统计器（PC0 和 PC1），分别受各自的使能信号控制，可以单独计数，也可以同时计数。

当事件统计使能（即 CSR：IS_CTL[PC0_EN]或 CSR：IS_CTL[PC1_EN]有效、且 CSR：IS_CTL[CM_PCE]或 CSR：IS_CTL[AM_PCE]有效）时，事件统计器 0 或 1 即对所选择的事件进行计数。计数器的初值可通过写 CSR：PC0_CR 和 CSR：PC1_CR 进行设置。当某事件统计器计数溢出时，若事件统计中断使能（即 CSR：IER[PCEN(1:0)]有效），则核心中断状态寄存器 CSR：INT_STAT[PC(1:0)]的对应位为“1”，触发对应的事件统计中断。事件统计器在溢出之后将从零开始继续计数，需要事件统计中断服务程序重新设置初值。计数器的具体数值可在中断服务程序中通过读 CSR：PC0_CR 和 CSR：PC1_CR 来获得。

事件统计器 0 的计数事件由 CSR：PC0_CR[PC0_SEL(3:0)]决定，具体定义如表 2-12。

表 2-12：事件统计器 0 的计数事件

PC0_SEL[3:0]	事件统计器 0 的计数事件
0x0	完成处理并退出的指令计数（不包含空指令和 IMEMB 指令）。
0x1	完成处理并退出的访存指令计数（含 EVICTD、FETCHD、FLUSHD 等指令）。

0x2	完成处理并退出的读访存指令计数（含 EVICTD、FLUSHD 指令）。
0x3	执行站台执行的转移指令计数。
0x4	执行站台执行的条件转移指令计数。
0x5	执行站台执行的无条件转移指令计数。
0x6	执行站台执行的 CALL 和 JSR 指令计数。
0x7	执行站台执行的 RET 指令计数。
0x8	处理器周期计数。
0x9	ITB 访问次数计数。
0xA	DTB 访问次数计数。
0xB	指令 Cache 读访问次数计数。
0xC	数据 Cache 读访问次数计数。
0xD	二级 Cache 访问次数计数，仅指指令流和数据流请求读 STAG 的次数。
0xE	主存访问次数计数（只包含指令流和数据流读主存的次数）。
0xF	Cache 一致性请求次数计数（只包含多 Cache 一致性处理产生的请求）。

事件统计器 1 的计数事件由 CSR：PC1_CR[PC1_SEL(5:0)]决定，具体定义如表 2-13。

表 2-13：事件统计器 1 的计数事件

PC1_SEL[5:0]	事件统计器 1 计数事件
0x0	指令流水线空周期计数（流水线空含义是指令译码、寄存器重命名、发射和退出站台都为空）。
0x1	重命名站台因整数重命名寄存器不足引起指令流水线停顿周期计数。
0x2	重命名站台因浮点重命名寄存器不足引起指令流水线停顿周期计数。
0x3	重命名站台因整数等待队列（IWQ）满引起指令流水线停顿周期计数。
0x4	重命名站台因浮点等待队列（FWQ）满引起指令流水线停顿周期计数。
0x5	重命名站台因重排序缓冲（ROB）满引起指令流水线停顿周期计数。
0x6	发射站台因装入队列（LQ）满引起指令发射停顿周期计数。
0x7	发射站台因存储队列（SQ）满引起指令发射停顿周期计数。
0x8	发射站台空的周期计数。
0x9	整数指令发射队列 IQ0 有阻塞情况的周期计数：
0xA	ROB 头部为访存类指令，且无法退出的周期计数。
0xB	执行站台判断出所有转移指令预测失败的次数。
0xC	执行站台判断出条件转移指令预测失败的次数。
0xD	执行站台判断出无条件转移指令预测失败的次数。

0xE	执行站台判断出 CALL、JMP 指令预测失败的次数。
0xF	执行站台判断出 RET 指令预测失败的次数。
0x10	二级 Cache 脱靶次数计数，仅指指令流和数据流读 STAG 发生 Miss 的次数。
0x11	二级 Cache 部件记录访存请求直接命中存储器页次数计数,根据存储控制器的响应 ReadDataDirty 和 ReadData 携带的信息计数。
0x12	二级 Cache 部件记录访存请求命中存储器页次数计数,根据存储控制器的响应 ReadDataDirty 和 ReadData 携带的信息计数。
0x13	二级 Cache 部件记录访存请求不命中存储器页次数计数,根据存储控制器的响应 ReadDataDirty 和 ReadData 携带的信息计数。
0x14	Cache 一致性请求请求命中二级 Cache 次数（包括命中回写缓冲）,仅指多 Cache 一致性处理产生的一致性请求。
0x15	BPU 误预测次数，即指令流水线译码站台看到 BPU 预测的指令类型与实际的指令类型不同的次数。
0x16	指令 Cache 读访问脱靶次数计数。（即不命中 ICache，也不命中 ICache 装填缓冲。）
0x17	ITB 脱靶次数计数。
0x18	L1DTB Miss，L2DTB 命中或偶校验错的计数
0x19	保留
0x1A	同站台旁路冲突计数
0x1B	MAF 满冲突计数
0x1C	同站台索引冲突计数
0x1D	DCache 端口 0 冲突计数
0x1E	DCache 端口 1 冲突计数
0x1F	写寄存器端口 1 冲突计数
0x25	访存请求从其它核心的 Cache 读取数据的情况。
0x26	整数指令发射队列 IQ1 有阻塞情况的周期计数。
0x27	整数指令发射队列 IQ2 有阻塞情况的周期计数。
0x28	访存指令地址发射队列 AQ0 有阻塞情况的周期计数。
0x29	访存指令地址发射队列 AQ1 有阻塞情况的周期计数。
0x2a	整数存储指令数据发射队列 ISQ 有阻塞情况的周期计数。
0x2b	浮点存储指令数据指令发射队列 FSQ 有阻塞情况的周期计数。
0x2c	浮点指令发射队列 FQ0 有阻塞情况的周期计数。
0x2d	浮点指令发射队列 FQ1 有阻塞情况的周期计数。

0x2e	重命名站台因访存等待队列（AWQ）满引起指令流水线停顿周期计数。
0x30	DTB SingleMiss 次数计数。
0x31	DTB Double Miss 次数计数。
0x32	所有访存指令读访问数据 Cache 脱靶次数计数。
0x33	“读－读”重发自陷次数计数。
0x34	“写－读”重发自陷次数计数。
0x35	保留
0x36	存储访问部件重发自陷总次数计数。
0x37	不含预取请求的访存指令访问数据 Cache 脱靶次数计数。
其它	保留。

选择处理器周期计数时，每个时钟周期计数器都“+1”；选择其它周期计数时，指定的计数条件成立时，每个时钟周期计数器“+1”。选择次数计数时，指定的事件每发生一次，计数器“+1”，需注意有些事件在一个周期里可能发生多次。

2.3.4 事件统计的流程

事件统计的一般流程如下：

1） 配置相关的 CSR，先通过写 CSR：PC0_CR[PC0_SEL]与 CSR：PC1_CR[PC1_SEL]选择事件统计事件，写 CSR：PC0_CR[PC0]与 CSR：PC1_CR[PC1]设置事件统计器的初值，然后写 CSR：IS_CTL[PC0_EN]或 CSR：IS_CTL[PC1_EN]指定事件统计中断 0 或 1 使能，再写 CSR：IS_CTL[CM_PCE]和 CSR：IS_CTL[PC_CM]，或只写 CSR：IS_CTL[AM_PCE]指定事件统计事件所在模式，最后写 CSR：IER[PCEN]，打开事件统计中断使能；(注意:如果希望准确计数，则需要先将 CSR：IS_CTL[PC0_EN]或 CSR：IS_CTL[PC1_EN]关闭，配置好 CSR：PC0_CR 和 CSR：PC1_CR 后再将计数使能打开。)；

2） 计数器计数：事件统计使能打开后，硬件就会根据相应 CSR 选择的计数事件，每发生一次对应事件，CSR：PC0_CR[PC0]或 CSR：PC1_CR[PC1]“+1”；

3） 计数器溢出：当 PC0_CR 或 PC1_CR 中的计数器溢出时，若事件统计中断使能，则产生事件统计中断请求，硬件置 CSR：INT_STAT 中 PC[1:0]有效；

4） 硬件中断：核心响应事件统计中断，进入中断特权处理程序；

5） 中断处理：事件统计器溢出后从 0 开始重新计数，中断服务程序需根据需要重新设置事件统计器的初值，为以后的计数作准备。

2.3.5 常用事件统计内容与方法

软件可根据需要，让 PC0 和 PC1 选择不同的事件同时计数，将一个计数器的值作为分母，另一个计数器的值作为分子，可计算出比值后用于衡量申威 411 处理器的相关性能。

在进行事件统计分析时，应尽可能使得需要的事件统计值在申威 411 处理器上运行一次得到，以提高事件统计的准确性。

下面列举了一些常用的事件统计内容及其计数事件的配置方法。

1） 转移预测事件统计的配置方法如表 2-14 所示。

表 2-14：转移预测事件统计的配置方法

性能测试事件	事件统计器 0	事件统计器 1
总转移预测失败率	PC0_SEL＝0x3 执行站台执行的转移指令计数	PC1_SEL＝0xB 执行站台判断出所有转移指令预测失败的次数
条件转移指令的转移预测失败率	PC0_SEL＝0x4 执行站台执行的条件转移指令计数	PC1_SEL＝0xC 执行站台判断出条件转移指令预测失败的次数
无条件转移指令的转移预测失败率	PC0_SEL＝0x5 执行站台执行的无条件转移指令计数	PC1_SEL＝0xD 执行站台判断出无条件转移指令预测失败的次数
JMP、CALL 的转移预测失败率	PC0_SEL＝0x6 执行站台执行的 JMP、CALL 指令计数	PC1_SEL＝0xE 执行站台判断出 JMP、CALL 指令预测失败的次数
RET 的转移预测失败率	PC0_SEL＝0x7 执行站台执行的 RET 指令计数	PC1_SEL＝0xF 执行站台判断出 RET 指令预测失败的次数
条件转移指令占转移指令比例	PC0_SEL＝0x3 执行站台执行的转移指令计数	PC1_SEL＝0x4 执行站台执行的条件转移指令计数
无条件转移指令占转移指令比例	PC0_SEL＝0x3 执行站台执行的转移指令计数	PC1_SEL＝0x5 执行站台执行的无条件转移指令计数
JMP、CALL 指令占转移指令比例	PC0_SEL＝0x3 执行站台执行的转移指令计数	PC1_SEL＝0x6 执行站台执行的 CALL 和 JSR 指令计数

RET 指令占转移指令比例	PC0_SEL＝0x3 执行站台执行的转移指令计数	PC1_SEL＝0x7 执行站台执行的 RET 指令计数
BPU 错误预测率	PC0_SEL＝0x3 执行站台执行的转移指令计数	PC1_SEL＝0x15 预测的指令类型与实际的指令类型不同的次数

2） 指令流水线事件统计的配置方法如表 2-15 所示。

表 2-15：指令流水线事件统计的配置方法

性能测试事件	事件统计器 0	事件统计器 1
CPU 空闲比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x0 指令流水线空闲周期计数
整数寄存器不足引起的 Map 阻塞的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x1 重命名站台因整数重命名寄存器不足引起 Map 停顿周期计数
浮点寄存器不足引起的 Map 阻塞的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x2 重命名站台因浮点重命名寄存器不足引起 Map 停顿周期计数
IQ 容量不足引起的流水线阻塞的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x3 重命名站台因整数发射队列（IQ）满引起流水线停顿周期计数
FQ 容量不足引起的 Map 阻塞的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x4 重命名站台因浮点发射队列（FQ）满引起 Map 停顿周期计数
ROB 容量不足引起的 Map 阻塞的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x5 重命名站台因重排序缓冲（ROB）满引起 Map 停顿周期计数
LQ 容量不足引起的 Issue 阻塞的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x6 发射站台因装入队列（LQ）满引起流水线停顿的周期计数
SQ 容量不足引起的 Issue 阻塞的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x7 发射站台因存储队列（SQ）满的周期计数停顿的周期计数
取指和译码速度不足引起 Issue 断流的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x8 发射站台因缺少指令而无法发射的周期计数

Issue 因各种原因无法全速发射的比例	PC0_SEL＝0x8 周期计数	PC1_SEL＝0x9 发射站台阻塞发射周期计数

3） 访存流水线事件统计的配置方法如表 2-16 所示。

表 2-16：监测访存流水线性能的配置方法

性能测试事件	事件统计器 0	事件统计器 1
ITB 脱靶率	PC0_SEL＝0x9 ITB 访问次数计数	PC1_SEL＝0x17 ITB 脱靶次数计数
DTB Single 脱靶率	PC0_SEL＝0xA DTB 访问次数计数	PC1_SEL＝0x18 DTB SingleMiss 次数计数
DTB Double 脱靶率	PC0_SEL＝0xA DTB 访问次数计数	PC1_SEL＝0x19 DTB Double Miss 次数计数
指令 Cache 脱靶率	PC0_SEL＝0xB 指令 Cache 访问次数计数	PC1_SEL＝0x16 指令 Cache 脱靶次数计数
数据 Cache 脱靶率	PC0_SEL＝0xC 数据 Cache 访问次数计数	PC1_SEL＝0x1A 数据 Cache 脱靶次数计数
SCache 脱靶率	PC0_SEL＝0xD SCache 访问次数计数	PC1_SEL＝0x10 SCache 脱靶次数计数
主存直接页命中率	PC0_SEL＝0xE 主存访问次数计数	PC1_SEL＝0x11 访存请求直接命中存储器页次数计数
主存页命中率	PC0_SEL＝0xE 主存访问次数计数	PC1_SEL＝0x12 访存请求命中存储器页次数计数
主存页脱靶率	PC0_SEL＝0xE 主存访问次数计数	PC1_SEL＝0x13 访存请求不命中存储器页次数计数
访存请求从其它核心的 Cache 读取数据的概率。	PC0_SEL＝0xE 主存访问次数计数	PC1_SEL＝0x25 访存请求从其它核心的 Cache 读取数据的情况。
CTag 副本精确率	PC0_SEL＝0xF Probe 请求次数计数	PC1_SEL＝0x14 Probe 请求查询命中 SCache 次数
读读重发自陷发生比例	PC0_SEL＝0x1 退出的访存指令计数	PC1_SEL＝0x1B 读－读重发自陷次数计数
写读重发自陷发生比例	PC0_SEL＝0x1 退出的访存指令计数	PC1_SEL＝0x1C 写－读重发自陷次数计数
SQ 数据旁路自陷发生比例	PC0_SEL＝0x1 退出的访存指令计数	PC1_SEL＝0x1D SQ 数据旁路自陷次数计数

DBOX 重发自陷发生比例	PC0_SEL＝0x1 退出的访存指令计数	PC1_SEL＝0x1E DBOX 重发自陷总次数计数

3 编译优化

3.1 转移预测优化

申威 411 处理器包含转移预测器，通过转移目标缓冲（BTB）和返回地址堆栈（RAS）来对转移方向和转移目标指令地址进行预测。

1） 指令组：申威 411 处理器在取指令时，一次取出地址自然对界的 4 条指令（即 128 位），将这地址对界的 4 条指令称之为指令组；

2） 指令组前段与指令组后段：申威 411 处理器进行指令译码、寄存器映射时，一次处理 2 条指令。对指令组中四条指令，分 2 次进行处理，前 2 条指令称为指令组前段，后 2 条指令称为指令组后段；

3） 指令槽：指令组前段或指令组后段的 2 条指令中，第一条指令位置称为指令槽 0，第二条指令位置称为指令槽 1；

4） 申威 411 处理器使用转移目标缓冲（BTB）来预测条件转移、无条件转移、CALL、JMP 等指令的转移方向和目标指令地址，使用返回地址堆栈（RAS）来预测 RET 指令的目标指令地址。

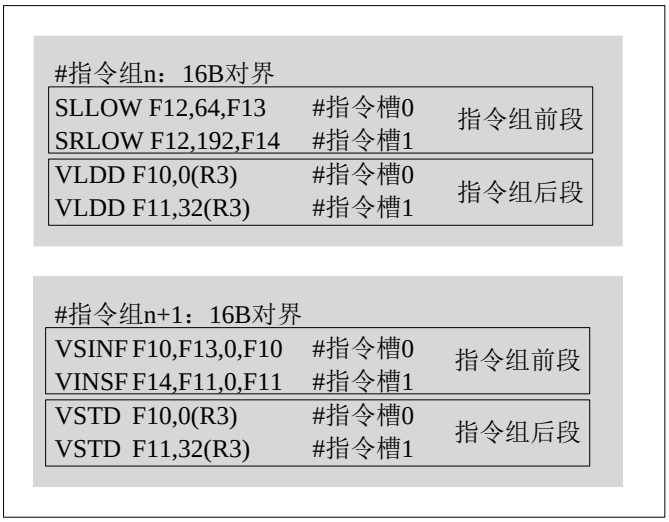


图 3-1：指令组和指令槽

3.1.1 转移目标地址的约束

通常要求程序的起始地址以及转移指令（包括跳转指令、无条件转移指令和条件转移指令）的

目标指令地址的低四位尽可能为全“0”，即程序的起始指令或跳转的目标指令最好是一个指令组的第一条指令，可以通过插入空指令来达到这个目的。

空指令在申威 411 处理器的指令流水线译码站台提前退出（Retire），所以空指令引起的主要开销是占用指令 Cache 的空间和取指令的带宽。

3.1.2 转移路径的选择

申威 411 处理器在初始化转移预测表时，会根据条件转移的方向初始化用于判断预测转移是否发生的饱和计数器初值。转移指令的偏移为正数时，饱和计数器初始化为“01”，偏移为负数时，饱和计数器初始化为“10”。当饱和计数器的值是“10”或“11”时，转移预测器将预测转移发生，当饱和计数器的值是“00”或“01”时，转移预测器将预测转移不发生。通过代码重排，将第一次转移发生概率较大的转移指令安排为往回跳转（偏移为负数），将第一次转移发生概率较小的转移指令安排为往前跳转（偏移为正数），可提高转移预测的成功率。

申威 411 处理器在硬件上支持指令预取，即始终按程序的顺序地址进行预取，每次预取 32 条指令。为了使指令预取能有效地提高指令 Cache 的命中率，可以通过重排代码，使顺序路径成为条件转移指令的两条执行路径中经常选择的路径。由于申威 411 处理器采用动态转移预测，重排代码不会影响转移预测的性能。

3.1.3 转移指令位置的优化

BTB 用于转移方向和转移目标地址的预测，BTB 中每个条目对应一个指令组，而不是一条指令。如果一个指令组中只有一条转移指令，则 BTB 条目中的转移历史信息更为准确；如果一个指令组中存在多条转移指令，则 BTB 条目中的转移历史信息被多条转移指令更改和共用，将会导致转移预测成功率明显下降。因此，一个指令组中只出现一条转移指令，可使得 BTB 的转移预测更为准确。

正常情况下，指令流访问命中指令 Cache，取指令站台每隔两个时钟周期向译码站台发送 4 条指令，译码站台需要两个时钟周期处理 4 条指令，即在前一个时钟周期处理指令组前段，在后一个时钟周期处理指令组后段。由于译码站台每个时钟周期只能处理一条转移指令，因此如果在指令组前段或后段同时存在两条转移指令，则译码站台的处理时间需要增加一个时钟周期，从而影响指令流水的速度。因此，转移指令放在一个指令组的尾部，即作为指令组中最后一条指令，可以获得更好的指令执行效率。

3.1.4 跳转指令使用的约束

RET 指令使用返回地址堆栈（RAS）来进行目标指令地址预测，当译码站台处理 BSR 和 CALL

指令时，将顺序指令地址压入 RAS，当译码站台处理 RET 指令时，就将 RAS 弹出的地址作为预测的目标指令地址，所以返回地址的压栈和弹出必须配对，才能保证 RAS 预测的正确性。

申威 411 处理器中设置了 8 个条目的 RAS，若返回地址堆栈不满，指令地址压入 RAS 时，分配一个新条目来保留返回地址，作为返回地址堆栈的栈顶条目。如果返回地址堆栈满，则将条目中的内容依次向下移动，返回地址堆栈的底部条目被覆盖，新的入栈地址写入栈顶条目。对于一个子程序调用，如果在其匹配的指令返回之前嵌套了多于 8 次的子程序调用，则 RAS 中含返回地址的对应条目会被覆盖，当执行 RET 指令时，部分 RET 指令的返回指令地址将无法准确预测。因此，子程序调用的嵌套次数尽量不超过 8 次。

3.2 寄存器和指令的使用

3.2.1 浮点寄存器的使用

在申威 1610 处理器中，向量寄存器（V0~V31）与浮点寄存器（F0~F31）共用，向量寄存器的低 64 位即是浮点寄存器。在申威 1610 处理器的汇编程序中，向量寄存器 Vi 采用对应的浮点寄存器 Fi 的命名来描述（i=0~31），由指令助记符来确定 Fi 为向量寄存器还是浮点寄存器。如图 3-2 中所示的扩展指令“VADDD F1,F2,F3”中，其向量寄存器描述符分别为 F1、F2 和 F3。

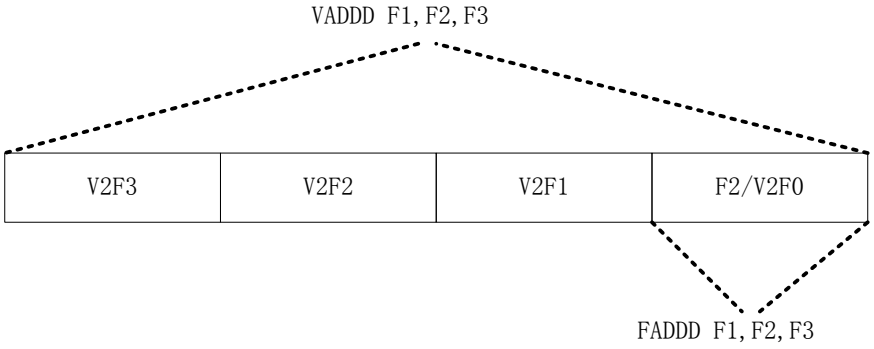


图 3-2：浮点寄存器和向量寄存器在使用中的差别

浮点寄存器既可以作为基本指令中浮点运算的源寄存器和目标寄存器，也可作为扩展指令的源寄存器和目标寄存器。浮点寄存器既可以用来保存基本指令和扩展指令与浮点运算相关的浮点数据，也可用来保存扩展指令与整数运算相关的整数数据。如图 3-2 所示，当浮点寄存器 F2 用于指令 FADDD 时，F2 为浮点寄存器，即寄存器 F2 的最低 64 位参与运算，当 F2 用于指令 VADDD 时，全部 256 位都参与运算。注意图 3-2 中的 V2F1、V2F2、V2F3 只是为了描述方便使用的符号，在指令中不能直接读取向量寄存器 F2 中 V2F1、V2F2 或 V2F3 中数据，即 F2 中四个 64 位数据总是同时被读出。

在使用浮点寄存器时，软件必须保证寄存器中存放数据的类型与指令功能的一致性，硬件对其

中的数据不作区别和检查。软件也应确保对寄存器中数据所作的运算是有意义的，否则运算的结果将不可预测。浮点寄存器文件中保存的整数数据不能直接与浮点数据进行运算，需要时，可通过浮点与整数的转换指令进行处理后，才能进行运算。

例如：假设向量寄存器 F1 和 F2 均为浮点数据，F3 为整数数据。F1 和 F2 相乘，其结果保存到 F4。若将 F3 与 F4 进行运算，由于 F4 为浮点数据，而 F3 为整数数据，F3 与 F4 运算是没有意义的，运算的结果也是不可预测的。

另外，基本指令中的浮点运算只对浮点寄存器进行操作，即执行结果写入命名相同的向量寄存器的低 64 位，而该向量寄存器的高 192 位内容不可预测，后续任何处理不能依赖高 192 位内容。同样，扩展指令的运算结果为向量数据时，结果写入向量寄存器（256 位），命名相同的浮点寄存器（即此向量寄存器的低 64 位）原有内容也被覆盖。若需要保留该浮点寄存器内容，则需要程序显式将向量寄存器中低 64 位的浮点寄存器内容保存到其它寄存器或存储器中。

例如：两条指令“VADDD F1，F2，F3”和“FADDD F3，F4，F3”完成处理并退出后，第一条指令产生的结果中，只有向量寄存器 F3 的低 64 位内容参与第二条指令的计算，第二条指令因寄存器更名，使得原保留在向量寄存器中的 V3F1、V3F2、V3F3 内容丢失，第二条指令完成并退出后，V3F1、V3F2、V3F3 中的内容将变得不可预测。

3.2.2 扩展指令的数据

由于向量寄存器与浮点寄存器共用，一般将向量寄存器分成两个部分，其低 64 位称之为主浮点寄存器，高 192 位称之为浮点扩展寄存器。扩展指令同时读写主浮点寄存器和浮点扩展寄存器，而基本指令只使用主浮点寄存器，因此扩展指令涉及的寄存器与基本指令涉及的浮点寄存器不能混用，否则会使浮点扩展寄存器中的数据丢失，产生不可预测的结果。建议的使用方法如下：

1） 扩展指令之间可以任意使用浮点寄存器作为源操作数寄存器或目标寄存器，此时的浮点寄存器被当做向量寄存器来使用；

2） 使用基本指令中的浮点装入指令读入的数据或浮点运算指令产生的结果数据作为扩展指令的源操作数前，应使用 VINSF/VINSW/VCPYF 指令，将浮点寄存器中数据扩展为 256 位向量数据；

3） 扩展指令产生的结果存放在主浮点寄存器和浮点扩展寄存器中，需要作为基本指令中的浮点运算的源操作数时，只有对应的主浮点寄存器内容可以直接使用，而浮点扩展寄存器的内容不能直接使用，应使用 VEXTF/VEXTW 指令，将浮点扩展寄存器内容传送到主浮点寄存器后再使用；

4） 如果扩展指令的目标寄存器又被作为基本指令的浮点运算目标寄存器，则这个目标寄存器对应的浮点扩展寄存器内容是不可预测的，使得该目标寄存器原有的浮点扩展寄存器内容丢失，如果这个内容是有意义的，必须在作为基本指令的浮点运算目标寄存器之前进行保

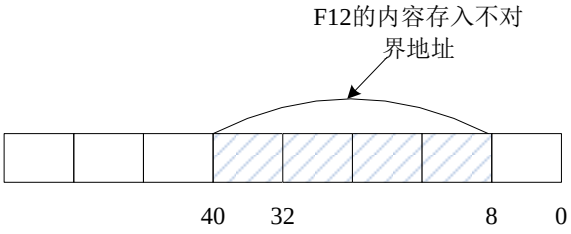
留。

3.2.3 扩展存储装入指令

在申威 1610 处理器的扩展指令中，包含 12 条存储器装入指令（VLDS、VLDD、LDWE、LDSE、LDDE、VLDW_UL、VLDW_UH、VLDS_UL、VLDS_UH、VLDD_UL、VLDD_UH、VLDD_NC）和 9 条存储器存储指令（VSTS、VSTD、VSTW_UL、VSTW_UH、VSTS_UL、VSTS_UH、VSTD_UL、VSTD_UH、VSTD_NC），这些指令也称之为向量存储装入指令，这些指令都是对主浮点寄存器和浮点扩展寄存器同时进行写或读。

由于申威 1610 处理器不支持向量存储装入指令产生的对 I/O 空间的数据流访问请求，若扩展存储装入指令产生的数据流地址对应的物理地址为 I/O 空间（即物理地址[39]位为“1”），则作为存取越权而产生数据流故障。另外，申威 1610 处理器也不支持大端（Big-endian）寻址模式，所有向量存储装入指令都只能按小端(Little-endian)模式寻址。

在使用扩展指令进行数据处理时，如果存储器中的向量数据存在地址不对界情况，可以使用不对界向量存储装入指令来提高性能。如图 3-3(a)中，要将 F12 的内容存入不对界地址的存储器中，该地址的最低四位为 0x08，如果使用不对界存储指令，只需简单两条指令，如图 3-3(b)，否则需要八条指令来实现相同功能，如图 3-3(c)。



(a)

```
VSTD_UL     F12,8(R3)
VSTD_UH     F12,40(R3)
```

(b)

```
SLLOW       F12,64,F13
SRLOW       F12,192,F14
VLDD        F10,0(R3)
VLDD        F11,32(R3)
VINSF       F10,F13,0,F10
VINSF       F14,F11,0,F11
VSTD        F10,0(R3)
VSTD        F11,32(R3)
```

(c)

图 3-3：使用不对界访存指令提升性能

3.2.4 不可 Cache 访存指令

申威 1610 处理器支持不可 Cache 访存指令，包括：LDW_NC、LDL_NC、LDD_NC、STW_NC、STL_NC、STD_NC、VLDD_NC、VSTD_NC，使用这些指令访问存储器空间时，数据不进入二级 Cache 和数据 Cache。

如果程序段中核心访问的数据没有时间局部性，即写入的数据很长时间内不会被用到，则比较适合使用不可 Cache 指令，一方面使用不可 Cache 指令可以避免将有时间局部性的数据从 Cache 中淘汰，另一方面也可以省掉处理可 Cache 写指令时先将数据装入到 Cache 中的开销。

图 3-4 中是不可 Cache 指令的一种典型应用，当 N 较大，两个数组的数据量超过二级 Cache 容量大小时，使用不可 Cache 存储指令可减少存储器访问次数，提高访存带宽，从而提升性能。

当需要对可 Cache 空间的相同存储器地址既使用可 Cache 指令进行访问又使用不可 Cache 指令进行访问时，对不可 Cache 访存指令的使用需要遵循一定的约束。由于硬件不保证对本核组的相同存储器空间可 Cache 访问与不可 Cache 访问之间的顺序，因此软件在对同一个存储器位置进行访问方式切换时，必须通过插入 MEMB 指令来保证两者之间的访问顺序。

```
void copy(double *a, double *c)
{
      register int i;

      for(i=0;i<N;i++)
            c[i]=a[i];
}
```

(a)

```
L_copy:
      VLDD    F13,0(R5)
      VLDD    F12,32(R5)
      VLDD    F11,64(R5)
      VLDD    F10,96(R5)
      ADDW    R4,4,R4
      CMPEQ   R4,R7,R17
      VSTD_NC F13,0(R6)
      VSTD_NC F12,32(R6)
      VSTD_NC F11,64(R6)
      VSTD_NC F10,96(R6)
      LDI     R5,128(R5)
      LDI     R6,128(R6)
      BEQ     R17,.L_copy
```

(b)

图 3-4：不可 Cache 指令典型应用

3.2.5 整数乘法

对于有符号的 32 位整数乘法，MULW 指令给出不溢出时的 32 位结果。当指令处理产生溢出时，MULL 指令可以给出全部的 64 位结果。

对于有符号的 64 位整数乘法，MULL 指令给出不溢出时的 64 位结果。当指令处理产生溢出时，需要使用 UMULH 指令来计算完整的 128 位结果。UMULH 指令计算两个无符号的 64 位源操作数乘积的高 64 位结果。有符号数乘积的完整 128 位结果如下：

1）高 64 位：使用 UMULH 指令可得高 64 位结果；

2）低 64 位：使用 MULL 指令可得低 64 位结果。

3.2.6 原子操作指令

原子操作指令包括“取并加一”、“取并减一”、“取并置一”三种指令，可用于实现多核之间的同步互斥操作。例如操作系统核心中常用的 spinlock 可用图 3-5 的代码来实现。

```
spin_lock:
        LDI R0,lockvar          //装入lockvar地址
        LDW_SET R0,0(R0)  //对lockvar取并置1
        NOP                     //空指令
        BNE R0,spin_lock      //加锁不成功，跳回

unlock:
        LDI R0,lockvar          //装入lockvar地址
        STW R31,0(R0)           //存入0，解锁
```

图 3-5：用原子操作实现的 spinlock

通过 spinlock 可以实现其它较复杂的临界区、信号灯等互斥机制，从而实现多个处理器核心之间对关键资源的互斥访问。基于 spinlock 还可实现多核间的各种同步机制。

另外，由于操作系统核心中大量的“计数器”也需要在多个处理器核心之间互斥进行，这些“计数器”的临界区内的代码只是简单的“加 1”或“减 1”的操作，如果全都通过临界区或 spinlock 锁操作实现，开销会比较大，使用原子操作指令可以大大简化这些操作。

3.2.7 锁指令

申威 411 处理器包含 6 条指令可用于实现软件中的各种锁操作。这 6 条指令分别是 LLDW、LLDL、LSTW、LSTL、WR_F、RD_F。

其中，锁装入指令（LLDW/LLDL）从指定存储器单元读取对应粒度的数据（字整数/长字整数）

到目标寄存器中，再将锁地址（经 TLB 映射后的物理地址）及数据粒度写入锁寄存器，并设置锁有效位（lock_valid）为“1”。

WR_F 指令用于将源寄存器中数据的最低位写入锁标志寄存器（lock_flag）。

锁存储指令（LSTW/LSTL）判断锁有效位（lock_valid）和锁标志位（lock_flag）是否都为“1”，并与锁寄存器中记录的锁地址和数据粒度是否匹配，还需用锁地址查询数据 Cache 判断是否命中数据 Cache 且状态为可写，这四个条件同时满足则完成写操作并置锁状态位（lock_success）为“1”，否则丢弃写操作并置 lock_success 为“0”。无论是否加锁成功，锁存储指令总是清除 lock_valid 和 lock_flag。

RD_F 指令用于将锁状态位（lock_success）读入目标寄存器中。

在操作系统中对共享资源的置位清零操作，如果都基于使用原子操作指令的 spinlock 实现，开销非常大，使用锁指令可大大减少这种开销，提升系统运行效率。图 3-6 中给出了使用锁指令实现置位操作的一个示例。

```
#对指定地址处的64位数据的某一位置“1”
#R16:指定地址
#R17:1<<n
#指令组：16B对界
set_bit:
    LLDL        R11,0x0(R16)
    LDI         R12,1
    WR_F        R12
    BIS         R11,R17,R11

    LSTL        R11,0x0(R16)
    RD_F        R11
    NOP
    BEQ         R11,set_bit
```

图 3-6：用锁指令实现的置位操作

在申威 1610 处理器中，为了简化硬件设计，对锁存储指令（LSTW/LSTL）和 RD_F 指令的使用进行了约束：锁存储指令（LSTW/LSTL）和 RD_F 必须配对使用，且锁存储指令（LSTW/LSTL）位于一个指令组中的指令槽 0 位置，RD_F 位于指令槽 1 位置，如果在编程中违背了这种约束，申威 1610 处理器将报非法指令异常。图 3-7 中列出了违反约束使用锁指令的所有情况。

```
#指令组n：16B对界
BIS     R0,R0,R0   #指令槽0
LSTW    R1,0x0(R2) #指令槽1;LSTW指令在指令槽1，LSTW报非法
BIS     R0,R0,R0   #指令槽0
BIS     R0,R0,R0   #指令槽1
                                (a)

#指令组n：16B对界
BIS     R0,R0,R0   #指令槽0
BIS     R0,R0,R0   #指令槽1
LSTL    R1,0x0(R2) #指令槽0;LSTL指令后跟的不是RD_F，LSTL报非法
BIS     R0,R0,R0   #指令槽1
                                (b)

#指令组n：16B对界
RD_F    R2         #指令槽0;RD_F指令在指令槽0，RD_F报非法
BIS     R0,R0,R0   #指令槽1
BIS     R0,R0,R0   #指令槽0
BIS     R0,R0,R0   #指令槽1
                                (c)

#指令组n：16B对界
BIS     R0,R0,R0   #指令槽0
BIS     R0,R0,R0   #指令槽1
BIS     R0,R0,R0   #指令槽0
RD_F    R2         #指令槽1;RD_F指令对应指令槽0不是锁存储指令，
                   #RD_F报非法
                                (d)
```

图 3-7：锁指令违反约束使用情况

申威 1610 处理器中，锁指令的使用还有另一个约束：配对使用的锁装入指令（LLDW/LLDL）和锁存储指令（LSTW/LSTL）之间不能使用转移指令，否则将造成锁存储指令不成功。图 3-8(a)中将加锁不成功，正确的写法如图 3-8(b)所示。

```
LLDW   R11,0x0(R16)          LLDW   R11,0x0(R16)
BNE    R11,locktry           XOR    R11,1,R11
LDI    R11,0x1               WR_F   R11
WR_F   R11                   LDI    R11,1

LSTW   R11,0x0(R16)          LSTW   R11,0x0(R16)
RD_F   R11                   RD_F   R11
BEQ    R11,locktry           BEQ    R11,locktry
MEMB                         MEMB
        (a)                          (b)
```

图 3-8：锁指令使用约束

另外，下列情况下锁存储和锁装入指令会发生异常，从而导致锁存储指令写操作不成功：

1） 锁存储或锁装入指令发生存取越权；

2） 锁装入指令访问一个具有 FOR（Fault On Read）属性的页面；

3） 锁存储指令访问一个具有 FOW（Fault On Write）属性的页面；

4） 锁存储或锁装入指令在计算地址时发生符号扩展错；

5） 锁存储或锁装入指令访问地址不对界；

6） 锁存储或锁装入指令访问 I/O 空间或不可 Cache 空间（其它核组的存储器空间）。

因此，程序员使用锁指令时需要保证锁地址的合法性。

3.3 避免重发自陷或冲突重试

申威 411 处理器可以同时处理多达 32 条访存指令，由于访存指令的乱序执行，访存指令之间的顺序会因违反存储一致性（Memory consistency）而引起正确性问题，例如 Load-Load 乱序、Load-Store 乱序，此时发生乱序的指令以及后续指令会被重新取指执行，这就是重发自陷。重发自陷是一种硬件机制，与其它异常或自陷不同，不是一种真正的错误。重发自陷的主要开销是必须等待被中止指令之前的所有指令正常退出（Retire），然后才能继续执行发生自陷的指令以及后继的指令。重发自陷对性能的影响主要是：

1） 产生重发自陷的指令等待前面的指令都正常退出；

2） 流水线的重新启动存在一定的运行时间开销。

为避免重发自陷产生较大的开销，申威 411 处理器有一种内部重试方式，称为冲突重试。该方式不需要重新取指令执行，只在核心的数据 Cache 控制部件内部重试，且重试请求的优先级低于正常的访存请求，对性能影响较低，但仍然存在一定的影响。

避免重发自陷或冲突重试的方法有：

1） 避免重发自陷或冲突重试最根本的办法是尽可能地减少对相同地址或相同 Cache 索引地址的连续读写访存。如存储的数据会马上被重新使用，应尽量避免使用访存指令，而是将数据保存在寄存器中，如果出现寄存器不够用的情形，可以使用 FIMOVx 或 IFMOVx 指令，在浮点寄存器和整数寄存器之间传输数据，从而避免这种重发自陷的发生；

2） 将两条可能引起重发自陷或冲突重试的指令分开可以直接避免重发自陷或冲突重试。如果数据在二级 Cache 中，则两条指令之间需要间隔至少 40 条指令才可以避免重发自陷，如果数据不在二级 Cache 而是在 DDR3 存储器中，需要间隔的指令条数则更多。

下面将介绍几种主要的重发自陷或冲突重试，说明其产生的原因与相关的影响，以便于进行程序优化。

3.3.1 写-读重发自陷

在程序中，访问相同地址的存储器写指令和读指令，写指令在前（年老），读指令在后（年轻），可能由于乱序发射导致年轻的存储器读指令比年老的写指令提早发射、执行，使得存储器读指令得

到的数据不是最新数据，从而引起写-读重发自陷。在申威 411 处理器中，产生写-读重发自陷的指令是年老的存储器写指令。

浮点存储器写指令需要特别注意，因为这类指令要求先在浮点指令队列中检查浮点源寄存器是否准备好，如果准备好并被选中，则两拍后才可能从整数队列中真正发射到执行部件，因此后续相同地址的存储器读指令（包括整数和浮点存储器读指令）更易被提早发射从而产生写-读重发自陷。

3.3.2 读-读重发自陷

为保证申威 1610 处理器内多核之间的存储一致性，需要保证最新的存储器读能够读到最新值。在程序中，对相同地址的两条存储器读指令，由于乱序发射会导致年轻的存储器读指令比年老的存储器读指令提早发射、执行，从而引起读-读重发自陷。在申威 1610 处理器中，产生读-读重发自陷的指令是年老的存储器读指令。

3.3.3 数据旁路产生的冲突重试

年老的存储器写指令先进入存储指令队列（SQ:Store Queue），后继年轻的相同地址的存储器读指令可以从 SQ 中旁路取得所需的数据，但是在某些情况下这种旁路获得的数据可能是错误的，为保证正确性，在发生这些情况时，硬件需要通过冲突重试拦住年轻的读指令，直到写数据写入数据 Cache 为止。

在下列情况下会发生冲突重试：

1） 存储器写指令的粒度小于存储器读指令的粒度，无法为读指令提供完整的数据，读指令产生冲突重试；

2） 有多条存储器写指令与存储器读指令地址相同，可旁路的存储器写指令有多条，读指令产生冲突重试。

在发生上述冲突重试时，只有与冲突重试相关的数据写入数据 Cache 后，冲突条件才会消失，因此，数据旁路产生的冲突重试等待时间较长。

3.3.4 读不命中产生的冲突重试

年轻的存储器读指令到达装入指令队列（LQ：Load Queue）时，发现年老的存储器读指令不命中数据 Cache，则年轻的存储器读指令产生冲突重试。一般类似石蕊测试的程序易发生这种冲突重试。

3.3.5 映射到同一 Cache 行的冲突重试

即使没有发生年轻指令超越年老指令提前发射的情况，为避免产生对相同索引 Cache 的装填和淘汰同时发生，存储器读写如果有如下情况，则年轻指令将产生冲突重试：

1） 读-读：两条存储器读指令的 Cache 索引地址相同，且年老的存储器读指令不命中数据 Cache；
2） 读-写：年老的存储器读指令和年轻的存储器写指令的 Cache 索引地址相同，且年老的存储器读指令不命中数据 Cache；
3） 写-读：年老的存储器写指令和年轻的存储器读指令的 Cache 索引地址相同，不论年老的存储器写地址是否命中数据 Cache；
4） 写-写：两条存储器写指令 Cache 索引地址相同，不论年老的存储器写指令是否命中数据 Cache。

当上述两条指令访问同一个 Cache 行，且满足合并条件时，不会发生冲突重试。

3.4 数据流访问

3.4.1 数据流访问的相关数据

申威 1610 处理器中与数据流访问有关的结构及技术参数如下：

1） Load 类指令产生的数据流访问命中数据 Cache 时，与此 Load 类指令的目标寄存器真相关的下一条指令的最小发射间隔为 4 个时钟周期；
2） 数据 Cache 容量为 32KB，相联度为 4，Cache 行大小为 128 字节；
3） Load 类指令产生的数据流访问命中二级 Cache 时，与此 Load 类指令的目标寄存器真相关的下一条指令的最小发射间隔为 14 个时钟周期；
4） 二级 Cache 容量为 512KB，相联度为 8，Cache 行大小为 128 字节；
5） 数据流访问命中二级 Cache，读取数据要连续占用内部数据总线 4 个时钟周期，在占用数据总线期间，不允许其它存储器访问；
6） 装入指令队列（LQ）有 32 个条目，最多存放 16 个读请求，读请求可以在 MAF 中合并；
7） 存储指令队列（SQ）有 16 个条目，最多存放 16 个写请求，写请求也可以在 MAF 中合并；
8） 不命中请求地址缓冲（MAF：Miss Address File）有 8 个条目，最多可以发出 8 个不同地址的存储器访问。

3.4.2 数据流访问合并

3.4.2.1 可 Cache 空间的合并

申威 411 处理器可以对可 Cache 空间的读写指令进行合并，合并窗口是 MAF 条目的有效周期，即从 MAF 分配条目到响应返回并删除 MAF 条目，这期间如果新产生的请求地址与已产生的请求地址属于同一个 Cache 行、地址不重叠，则新请求可以合并到已有请求中。对可 Cache 空间的访问，不同数据长度、不同数据类型的可 Cache 读写指令都可以合并，可 Cache 读指令和可 Cache 写指令也可以合并。

对可 Cache 空间的不可 Cache 指令的合并规则与 3.4.2.2 节“不可 Cache 空间的合并”中的规则相同。

3.4.2.2 不可 Cache 空间的合并

申威 411 处理器可以对不可 Cache 读写指令进行合并。

不可 Cache 读指令的合并窗口是 MAF 条目的有效周期，即从 MAF 分配条目到响应返回并删除 MAF 条目，这期间如果新产生的请求地址与已产生的请求地址属于同一个 Cache 行、地址不重叠，则新请求可以合并到已有请求中。

不可 Cache 写指令的合并窗口可通过 CSR:MERGE_OVTIME 配置，缺省值为 256 个核心时钟周期，最多可将对连续 128B 数据的写请求合并成一条写请求。不可 Cache 写的具体合并规则如下：

1） 不同数据长度、不同数据类型的 Store 指令不允许合并；

2） 按程序顺序，可以对前后若干条访问数据长度为字节的 Store 指令在自然对界的 16 字节范围内进行合并；

3） 按程序顺序，可以对前后若干条访问数据长度为半字的 Store 指令在自然对界的 32 字节范围内进行合并；

4） 按程序顺序，可以对前后若干条访问数据长度为字整数或单精度浮点的 Store 指令在自然对界的 64 字节范围内进行合并；

5） 按程序顺序，可以对前后若干条访问数据长度为长字整数或双精度浮点的 Store 指令在自然对界的 128 字节范围内进行合并；

6） 按程序顺序，可以对前后若干条访问数据长度为单精度浮点向量的 Store 指令在自然对界的 128 字节范围内进行合并；

7） 按程序顺序，可以对前后若干条访问数据长度为字整数向量或双精度浮点向量的 Store 指令在自然对界的 128 字节范围内进行合并；

8） 按程序顺序，可以对前后若干条不对界字向量或单精度浮点向量 Store 指令在自然对界的 64

字节范围内进行合并；

9） 按程序顺序，可以对前后若干条不对界双精度浮点向量 Store 指令在自然对界的 128 字节范围内进行合并。

由于不可 Cache 读可以关闭不可 Cache 写的合并窗口，不可 Cache 写也可以关闭不可 Cache 读的合并窗口。因此，连续产生的不可 Cache 写可以合并，中间不要插入不可 Cache 读；不可 Cache 读也同样需要连续产生，中间不要插入不可 Cache 写，这样可以提高带宽利用率。

在编程中，对大块连续空间的访问，访存指令的访存地址应顺序改变，避免一个 Cache 行的合并窗口被另一个 Cache 行的访问关闭。如图 3-9 所列的两段代码中，每个循环的访存地址总是升序的，但在一个循环体内，图 3-9(a)中的 VSTD_UH 和 VSTD_UL 两条指令的访存地址是降序排列，图 3-9(b)中的 VSTD_UH 和 VSTD_UL 两条指令的访存地址是升序排列，当多个循环展开后，访存地址可能出现跨 Cache 行，此时图 3-9(a)不能合并，而图 3-9（b）一直可以合并，因此图 3-9(b)的实际性能要比图 3-9(a)的性能高。

```
L0:
ADDW      R4,0x1,R4
VSTD_UH   F12,40(R3)
VSTD_UL   F12,8(R3)
LDI       R3,32(R3)
CMPEQ     R4,R5,R6
BEQ       R6,L0
```
(a)

```
L0:
ADDW      R4,0x1,R4
VSTD_UL   F12,8(R3)
VSTD_UH   F12,40(R3)
LDI       R3,32(R3)
CMPEQ     R4,R5,R6
BEQ       R6,L0
```
(b)

图 3-9：有效利用合并缓冲

3.4.2.3 I/O 空间的合并

申威 1610 处理器不支持对 I/O 空间读指令的合并。

申威 1610 处理器支持对 I/O 空间写指令的合并，I/O 空间写指令的合并超时时间可通过 CSR:MERGE_OVTIME 配置，缺省值为 256 个核心时钟周期。如果前后若干条长字 I/O 写指令产生的数据流地址是升序且不重叠（不一定连续），则允许这些指令在自然对界的 128 字节范围内进行合并。具体合并规则如下：

1） 字节或半字的 I/O 写指令不允许合并，不同数据长度的 I/O 写指令不允许合并；

2） 按程序顺序，如果前后若干条字 I/O 写指令产生的数据流地址是升序且不重叠（不一定连续），则可以对这些指令在自然对界的 64 字节范围内进行合并；

3） 按程序顺序，如果前后若干条长字 I/O 写指令产生的数据流地址是升序且不重叠（不一定连续），则可以对这些指令在自然对界的 128 字节范围内进行合并；

4） MEMB 指令、I/O 读指令或不能合并的 I/O 写指令，将关闭 I/O 写指令的合并窗口。

申威 1610 处理器对 I/O 空间的合并可用于提升处理器核心访问 PCIE 的 EP 设备空间的性能。

3.5 低功耗优化

3.5.1 浮点执行部件的功耗控制

申威 411 处理器的浮点执行部件中，包括四条运算流水线，其中主浮点运算流水线对应主浮点寄存器，三条从浮点运算流水线对应浮点扩展寄存器。由于浮点执行部件在核心中所占功耗比例较大，其中从浮点运算流水线在浮点执行部件中所占功耗最大，因此在执行基本指令中的浮点运算指令时，仅使用主浮点运算流水线，执行扩展指令的向量运算指令时，才使用主、从四条浮点运算流水线。

申威 411 处理器支持浮点执行部件的动态功耗管理，通过 CSR：UPCR 可以控制浮点执行部件以及从浮点运算流水线的工作时钟。

针对不使用扩展指令中向量运算指令的程序，可以通过 CSR：UPCR 关闭从浮点运算流水线的时钟，降低核心的运行功耗。

针对不使用基本指令中浮点运算指令和扩展指令中向量运算指令的程序，可以通过 CSR：UPCR 关闭浮点执行部件的时钟，进一步降低核心的运行功耗。

即使程序中存在少量的浮点运算指令或者向量运算指令，在性能允许的前提下，也可以关闭浮点执行部件或者从浮点运算流水线时钟，此时执行这些指令将产生自陷，可以通过软件仿真方式完成这些指令的处理。

3.5.2 核心睡眠

申威 1610 处理器还支持核心的浅睡眠和深睡眠功能，程序员可通过操作系统提供的相关功能进一步实施低功耗优化。

当申威 1610 处理器的某个核心短时间里没有可执行的任务时，可在非硬件模式下，通过执行 HALT 指令（硬件模式下此指令被当作空指令），使核心进入浅睡眠状态。此状态下，核心的指令流水线不再进行取指令，也不发射指令，可大大降低该核心的运行功耗。处于浅睡眠状态下的核心仍然支持 Cache 一致性操作，核心内的寄存器和 Cache 中的数据不会丢失。通过任何中断可将核心浅睡眠状态唤醒。

当申威 1610 处理器的某个核心长时间里没有可执行的任务时，可由该核心或其它核心向该核心发送睡眠中断，使核心进入深睡眠状态。此状态下，核心处于复位状态，工作频率为维护时钟的八分频，可最大限度降低核心的运行功耗。处于深睡眠状态下的核心不能接收睡眠唤醒中断之外的其它中断，也不支持 Cache 一致性操作，核心内的寄存器和 Cache 中的数据会丢失，因此系统软件必须在核心进入深睡眠之前，保留寄存器中的内容，刷新数据 Cache 和二级 Cache。当睡眠的核心需

要恢复运行时，可以通过其它核心在硬件模式下向该核心发送睡眠唤醒中断，唤醒处于睡眠状态的核心，并恢复寄存器中的内容，然后进入正常的运行状态。

4 调试支持

申威 411 处理器的可测性设计，包括以下几个方面：

1） 存储器自测试；

2） 状态扫描。

4.1 存储器自测试

4.1.1 自测试类型

申威 411 处理器的存储器自测试包括 Debug、BIST、BISR 和 ScanBISR 四种测试模式，如表 4-9 所示。

表 4-1：BIST 测试类型说明

测试类型	含义	备注
ScanBISR 测试	灵敏放大器值自动迭代的自测试方式，且支持自修复。	普通流程，具体类型与具体存储阵列相关。
BISR 测试	指定灵敏放大器值或实现方式不同于灵敏放大器的自测试方式，且支持自修复。	
BIST 测试	指定灵敏放大器值或实现方式不同于灵敏放大器的自测试方式，但不支持自修复。	
Debug 测试	指定指定灵敏放大器值进行自测试，自测试过程的出错信息（包括出错地址、出错数据）可以被记录，便于分析。	特殊流程，用于 Debug 测试。

ScanBISR、BISR 和 BIST 三种普通模式，可以按照默认流程进行，也可以通过配置维护 IOR：INIT_CTL 指定自测试类型。在默认流程中，上电复位、冷复位会启动片内所有存储器阵列的存储器自测试，核心睡眠唤醒会启动相应核心内所有存储器阵列的存储器自测试。存储器自测试结束以后，测试结果会登记在维护接口 IOR：BIST_STAT、CGx_BIST_STAT，可以通过维护命令读取该测试结果。

Debug 测试，测试过程中记录的错误信息通过维护接口以交互的方式获得。首先芯片通过 BIST_STOP_H（TEST_OUT[3]_H 引脚，需要 IOR：TESTSEL 配置为零）通知维护系统，发现 BIST

测试错误，维护系统利用维护命令扫描 BIST 测试 Debug 信息，完成扫描后通过 BIST_GOON_L 引脚（MT_RESET_L 复用）通知芯片继续 Debug 测试。具体的 Debug 信息见状态扫描。

4.1.2 测试状态输出引脚

存储器自测试结果会通过测试结果输出。表 3-1 给出了测试结果的具体信号以及含义。BIST_DONE、BIST_FAIL 和 BIST_Repair 是电平信号，表示 BIST 测试的最终结果。BIST_STOP_H 只在 Debug 测试时有意义，用于发生 BIST 错误时和系统进行交互。系统收到 BIST_STOP_H 信号，会利用 SCAN 扫出命令得到发生错误的地址和错误数据，而后利用 BIST_GOON_L 通知芯片继续进行 BIST 测试。

表 4-2：BIST 测试结果引脚输出

IOR：TESTSEL	内部信号	对应的 TEST_OUT_H	含义说明
0（默认）	BIST_DONE	TEST_OUT_H [0]	BIST 测试完成，高电平信号有效。
	BIST_FAIL	TEST_OUT_H [1]	TEST_OUT_H[2:1]具体含义如下： 为 2’b00，表示测试无错； 为 2’b01，表示有错不可修复； 为 2’b10，表示有错可修复； 为 2’b11，表示测试时间超时。
	BIST_Repair	TEST_OUT_H [2]	
	BIST_STOP_H	TEST_OUT_H [3]	指示在存储器 Debug 测试方式下，发现错误而暂停存储器 Debug 测试，此时可通过状态扫描获得具体的错误信息，有效一个维护时钟周期（在维护时钟的下降沿输出）。
	保留	TEST_OUT_H [4]	保留。

4.1.3 BIST FUSE 信息

除维护的 IOR：BIST_STAT 和 CGx_BIST_STAT 外，针对各灵敏放大器值的测试结果、最终的选择的灵敏放大器的值，以及修复地址都记录在各存储器阵列底层。这些信息可以在申威 411 处理器内部存储器阵列的存储器自测试完成以后，通过维护接口的扫描命令得到，具体信息见状态扫描。

4.2 状态扫描

申威 411 处理器的扫描信息按 16MB 的地址空间进行编址。维护命令（SCAN 扫出、扫入命令）根据 24 位的地址，对各位进行扫出、扫入操作。申威 411 处理器大部分信息只支持扫出操作。

4.2.1 扫描地址分配

表 3-2 给出了申威 411 处理器监测信息的编址说明。

表 4-3：核心级地址说明

扫描内容	预留的数量	预留的地址宽度	地址高位[12:9]	部件输入地址	占用数据位数
IBOX	4096	12	0XXX	[11:0]	[0]
ICache	128	7	00XX	[6:0]	[1]
EBOX	256	8	010X	[7:0]	[1]
FBOX	512	9	011X	[8:0]	[1]
S_SBOX	4096	12	0XXX	[11:0]	[2]
S_SCDATA0	512	9	0000	[8:0]	[3]
S_SCDATA1	512	9	0001	[8:0]	[3]
S_SCDATA2	512	9	0010	[8:0]	[3]
S_SCDATA3	512	9	0011	[8:0]	[3]
S_SCDATA4	512	9	0100	[8:0]	[3]
S_SCDATA5	512	9	0101	[8:0]	[3]
S_SCDATA6	512	9	0110	[8:0]	[3]
S_SCDATA7	512	9	0111	[8:0]	[3]
S_STAG	4096	12	1XXX	[11:0]	[3]
DC_BIST_TOP	1024	10	0XXX	[9:0]	[4]
DBOX	2048*4	11	1XXX	[10:0]	[7:4]

4.2.2 环振测试控制信息

表 4-14 给出了申威 411 处理器环振测试控制信息。

表 4-4：环振测试控制信息

字节内数据位	核心内地址[12:0]	简称	含义	操作类型
[4]	0	Ring_En	环振测试使能。	读写
[4]	1	Ring_Type	环振测试类型选择，“0”表示 HVT，“1”表示 SVT。	读写
[4]	3~2	Ring_Sel[1:0]	环振测试链选择，环振被放在核心的不同位置，共四个。	读写

4.2.3 Icache 自测试信息

表 4-给出了申威 411 处理器 Icache 自测试信息。ICData 四路共用一个修复列。

表 4-5：Icache 自测试信息

字节内数据位	核心内地址[12:0]	简称	含义	操作类型
[0]	3073~3072	Icache_Fuse[1:0]	Icache 的 SAEN 信息。	读写
[0]	3074	Icache _FuseCtl	Icache 的 SAEN 选择策略。	读写
[0]	3082~3080	Icache_SAEN0_STAT[2:0]	Icache 的 SAEN0 的自测试结果。	只读
[0]	3085~3083	Icache_SAEN1_STAT[2:0]	Icache 的 SAEN1 的自测试结果。	只读
[0]	3088~3086	Icache_SAEN2_STAT[2:0]	Icache 的 SAEN3 的自测试结果。	只读
[0]	3091~3089	Icache_SAEN3_STAT[2:0]	Icache 的 SAEN3 的自测试结果。	只读
[0]	3094~3092	Icache_SAEN_STAT[2:0]	Icache 的自测试结果（对应 Icache_Fuse）。	只读
[0]	3097~3095	BistMode_STAT	Icache Bist 模式	只读
[0]	3104	Icache_BisrFlag_Scan_Stat	Bisr 标志	只读
[0]	3105	Icache_BistFlag_Scan_Stat	Bist 标志	只读
[0]	3106	Icache_BisiFlag_Scan_Stat	Bisi 标志	只读
[0]	3107	Icache_DebugFlag_Scan_Stat	Debug 标志	只读
[0]	3111~3108	Icache_Fail_All	测试失败标志，每位对应一个 ICSubData	只读
[0]	3115~3112	Icache_Repair_All	测试修复标志，每位对应一个 ICSubData	只读
[0]	3116	Icache_TestFail_Scan_Stat	总的失败标志	只读
[0]	3117	Icache_TestRepair_Scan_Stat	总的修复标志	只读

[0]	444~278	ICAHE_Monitor[166:0]	ICAHE 的 Debug 信息。 [164:163]：路号； [162:154]：地址； [153:149]：状态机状态； [148]：期望数据； [147:111]:针对修复粒度 3 读出的数据，为 1 表示该位数据出错； [110:74]：针对修复粒度 2 读出的数据，为 1 表示该位数据出错； [73:37]：针对修复粒度 1 读出的数据，为 1 表示该位数据出错； [36:0]：针对修复粒度 0 读出的数据，为 1 表示该位数据出错。	只读
[1]	0	Icache_W0G0_RepaireV	Icache 的修复粒度 0 的列冗余有效位。	读写
[1]	6~1	Icache_W0G0_Repaire[5:0]	Icache 的修复粒度 0 的列地址。	读写
[1]	8	Icache_W0G1_RepaireV	Icache 的修复粒度 1 的列冗余有效位。	读写
[1]	14~9	Icache_W0G1_Repaire[5:0]	Icache 的修复粒度 1 的列地址。	读写
[1]	16	Icache_W0G2_RepaireV	Icache 的修复粒度 2 的列冗余有效位。	读写
[1]	22~17	Icache_W0G2_Repaire[5:0]	Icache 的修复粒度 2 的列地址。	读写
[1]	24	Icache_W0G3_RepaireV	Icache 的修复粒度 3 的列冗余有效位。	读写
[1]	30~25	Icache_W0G3_Repaire[5:0]	Icache 的修复粒度 3 的列地址。	读写

4.2.4 DCACHE 自测试信息

表 4-1 给出了申威 411 处理器 DCACHE 自测试信息。

表 4-1：DCACHE 自测试信息

字节内数据位	核心内地址[12:0]	简称	含义	操作类型

[4]	33~32	DCACHE_Fuse[1:0]	DCACHE 的 SAEN 信息。	读写
[4]	34	DCACHE _FuseCtl	DCACHE 的 SAEN 选择策略。	读写
[4]	42~40	DCACHE_SAEN0_STAT[2:0]	DCACHE 的 SAEN0 的自测试结果。	只读
[4]	45~43	DCACHE_SAEN1_STAT[2:0]	DCACHE 的 SAEN1 的自测试结果。	只读
[4]	48~46	DCACHE_SAEN2_STAT[2:0]	DCACHE 的 SAEN3 的自测试结果。	只读
[4]	51~49	DCACHE_SAEN3_STAT[2:0]	DCACHE 的 SAEN3 的自测试结果。	只读
[4]	54~52	DCACHE_SAEN_STAT[2:0]	DCACHE 的自测试结果（对应 DCACHE_Fuse）。	只读
[4]	57~55	DCACHE_BIST_MODE[2:0]	给 BIST_CTRL 的测试模式	只读
[4]	128	DCACHE_BISR_FLAG	DCACHE 自测试模式为 BISR	只读
[4]	129	DCACHE_BIST_FLAG	DCACHE 自测试模式为 BIST	只读
[4]	130	DCACHE_DEBUG_FLAG	DCACHE 自测试模式为 DEBUG	只读
[4]	131	DCACHE_DONE_FLAG	DCACHE 自测试完成标志	只读
[4]	139~132	DCACHE_G[7:0]_Fail	DCACHE 修复粒度[7:0]的有错不可修标志。	只读
[4]	147~140	DCACHE_G[7:0]_Repair	DCACHE 修复粒度[7:0]的有错可修标志。	只读
[4]	472~160	DCACHE_Monitor[312:0]	DCACHE 的 Debug 信息。 [312]:1’b0； [311:302]:地址[12:5]+组号[1:0]； [301:297]:状态机状态； [296]:期望数据； [295:0]:错误数据（读出数据）。	只读
[4]	576	DCACHE_G0_RepaireV	DCACHE 的修复粒度 0 的列冗余有效位。	读写
[4]	582~577	DCACHE_G0_Repaire[5:0]	DCACHE 的修复粒度 0 的列地址。	读写
[4]	584	DCACHE_G1_RepaireV	DCACHE 的修复粒度 1 的列冗余有效位。	读写
[4]	590~585	DCACHE_G1_Repaire[5:0]	DCACHE 的修复粒度 1 的列地址。	读写
[4]	592	DCACHE_G2_RepaireV	DCACHE 的修复粒度 2 的列冗余有效位。	读写
[4]	598~593	DCACHE_G2_Repaire[5:0]	DCACHE 的修复粒度 2 的列地址。	读写

[4]	600	DCACHE_G3_RepaireV	DCACHE 的修复粒度 3 的列冗余有效位。	读写
[4]	606~601	DCACHE_G3_Repaire[5:0]	DCACHE 的修复粒度 3 的列地址。	读写
[4]	608	DCACHE_G0_RepaireV	DCACHE 的修复粒度 4 的列冗余有效位。	读写
[4]	614~609	DCACHE_G0_Repaire[5:0]	DCACHE 的修复粒度 4 的列地址。	读写
[4]	616	DCACHE_G1_RepaireV	DCACHE 的修复粒度 5 的列冗余有效位。	读写
[4]	622~617	DCACHE_G1_Repaire[5:0]	DCACHE 的修复粒度 5 的列地址。	读写
[4]	624	DCACHE_G2_RepaireV	DCACHE 的修复粒度 6 的列冗余有效位。	读写
[4]	630~625	DCACHE_G2_Repaire[5:0]	DCACHE 的修复粒度 6 的列地址。	读写
[4]	632	DCACHE_G3_RepaireV	DCACHE 的修复粒度 7 的列冗余有效位。	读写
[4]	638~633	DCACHE_G3_Repaire[5:0]	DCACHE 的修复粒度 7 的列地址。	读写

4.2.5 SCACHE 自测试信息

SCACHE 数据部分从高到低分为 SCSUBDATA[i]、SCBANK[j]、SCSET[k]共 3 个层次，每一个 SCSET 是一个行列修复粒度，其中 i、j 和 k 的取值范围分别为 7:0、1:0 和 7:0，因此 SCACHE 的数据阵列共包含 128 个修复粒度。若为每一个修复粒度分配一个编号 N，则 N 等于 i×16+j×8+k，N 的取值范围为 127:0。扫描逻辑为每一个修复粒度分配 16 位的修复结果信息，用于记录行列修复粒度的结果、列修复有效位和列修复地址，其中高 7 位保留。每个修复粒度的 16 位信息连续编址，修复粒度所分配的地址按 3 个层次从低到高连续编址。扫描逻辑为 SCACHE 数据阵列的各个修复粒度的行修复有效和地址分配独立的 16 位信息位（在 STAG 内实现），其中高 8 位保留。

表 4-2：SCACHE 数据阵列自测试信息

字节内数据位	核心内地址[12:0]	简称	含义	操作类型
[2]	2805	SC_TestDone	SCACHE 数据阵列的测试结束标志。	只读
[2]	2804	SC_TestFail	SCACHE 数据的测试结果， 1 表示测试失败。	只读
[2]	2803	SC_TestRepair	SCACHE 数据测试结束标志， 1 表示需修复。	只读
[2]	3126~1818	SC_Monitor[311:3]	SCACHE 数据阵列的 DEBUG 信息：	只读

			[311:308]：debug 测试时发现错误的 SCSUBDATA 的 Way 号,其中[3:1]位表示路号，0 位表示 BANK 号。 [307:297]：debug 测试时发现错误的行地址。 [296:292]：错误时的主状态机状态。 [291] ：期望的数据。 [290:255]：读出的 SCD7 的非冗余列数据。 [254:219]：读出的 SCD6 的非冗余列数据。 [218:183]：读出的 SCD5 的非冗余列数据。 [182:147]：读出的 SCD4 的非冗余列数据。 [146:111]：读出的 SCD3 的非冗余列数据。 [110: 75]：读出的 SCD2 的非冗余列数据。 [74: 39]：读出的 SCD1 的非冗余列数据。 [38:3]：读出的 SCD0 的非冗余列数据。	
[2]	2753~2752	SC_SAEN[1:0]	SCACHE 数据阵列的 SAEN 的值。	读写
[2]	2754	SC_SAENCtrl	SCACHE 数据阵列的 SAEN 的选择策略。	读写
[2]	2762~2760	SC_SAEN0 [2:0]	SCACHE 数据阵列在 SAEN=0 的测试结果。	只读
[2]	2765~2763	SC_SAEN1 [2:0]	SCACHE 数据阵列在 SAEN=1 的测试结果。	只读
[2]	2768~2766	SC_SAEN2 [2:0]	SCACHE 数据阵列在 SAEN=2 的测试结果。	只读
[2]	2771~2769	SC_SAEN3 [2:0]	SCACHE 数据阵列在 SAEN=3 的测试结果。	只读

[3]	N×16+8	SC_G[N]_TestFail	SC 粒度 N 测试结果，为 1 表示测试失败。	只读
[3]	N×16+7	SC_G[N]_TestRepair	SC 粒度 N 测试结果，为 1 表示需要修复。	只读
[3]	N×16+6	SC_G[N]_RepColValid	SC 粒度 N 列修复有效位。	读写
[3]	N×16+5 ~N×16	SC_G[N]_RepColAddr	SC 粒度 N 列修复地址。	读写
[3]	6144+N×16+7	SC_G[N]_RepRowValid	粒度 N 行修复有效位。	读写
[3]	6144+N×16+6 ~6144+N×16	SC_G[N]_RepRowAddr	粒度 N 行修复地址。	读写

注：其中 N=0~127。

表 4-3：STAG 的自测试信息

字节内数据位	核心内地址[12:0]	简称	含义	操作类型
[3]	4658	STAG_TestDone	STAG 的测试结束标志。	只读
[3]	4657	STAG_TestFail	STAG 的测试结果， 1 表示测试失败。	只读
[3]	4656	STAG_TestRepair	STAG 测试结束标志， 1 表示需修复。	只读
[3]	4715~4660	STAG_Monitor[58:3]	STAG 的 DEBUG 信息： [58:56]：debug 测试时发现错误的路号； [55:52]：保留位 ； [51:43]：debug 测试时发现错误的地址； [42:38]：错误时的主状态机状态； [37]：期望的数据； [36:34]：读出的淘汰指针； [33: 3]：读出的非冗余数据 。	只读
[3]	4609~4608	STAG_SAEN[1:0]	STAG 的 SAEN 的值。	读写
[3]	4610	STAG_SAENCtrl	STAG 的 SAEN 的选择策略。	读写
[3]	4618~4616	STAG_SAEN0[2:0]	STAG 在 SAEN=0 的测试结果。	只读
[3]	4621~4619	STAG_SAEN1[2:0]	STAG 在 SAEN=1 的测试结果。	只读
[3]	4624~4622	STAG_SAEN2[2:0]	STAG 在 SAEN=2 的测试结果。	只读

[3]	4627~4625	STAG_SAEN3[2:0]	STAG 在 SAEN=3 的测试结果。	只读
[3]	4767	STAG_G0_TestFail	STAG 粒度 0 测试结果，为 1 表示测试失败。	只读
[3]	4766	STAG_G0_TestRepair	STAG 粒度 0 测试结果，为 1 表示需要修复。	只读
[3]	4742	STAG_G0_RepColValid	STAG 粒度 0 列修复有效位。	读写
[3]	4741~4736	STAG_G0_RepColAddr	STAG 粒度 0 列修复地址。	读写
[3]	4799	STAG_G1_TestFail	STAG 粒度 1 测试结果，为 1 表示测试失败。	只读
[3]	4798	STAG_G1_TestRepair	STAG 粒度 1 测试结果，为 1 表示需要修复。	只读
[3]	4774	STAG_G1_RepColValid	STAG 粒度 1 列修复有效位。	读写
[3]	4773~4768	STAG_G1_RepColAddr	STAG 粒度 1 列修复地址。	读写
[3]	4831	STAG_G2_TestFail	STAG 粒度 2 测试结果，为 1 表示测试失败。	只读
[3]	4830	STAG_G2_TestRepair	STAG 粒度 2 测试结果，为 1 表示需要修复。	只读
[3]	4806	STAG_G2_RepColValid	STAG 粒度 2 列修复有效位。	读写
[3]	4805~4800	STAG_G2_RepColAddr	STAG 粒度 2 列修复地址。	读写
[3]	4863	STAG_G3_TestFail	STAG 粒度 3 测试结果，为 1 表示测试失败。	只读
[3]	4862	STAG_G3_TestRepair	STAG 粒度 3 测试结果，为 1 表示需要修复。	只读
[3]	4838	STAG_G3_RepColValid	STAG 粒度 3 列修复有效位。	读写
[3]	4837~4832	STAG_G3_RepColAddr	STAG 粒度 3 列修复地址。	读写

4.2.6 核心观测信息

表 4-给出了核心的观测信息，表中的地址是部件输入地址。

表 4-9：核心观测信息

字节内 数据位	部件 输入地址	简称	含义
一级指令 Cache（IBOX）			

0	563	r_RdDataValid	读 ICache 站台有效。
0	565~564	r_RdDataMode[1:0]	读 ICache 站台模式。
0	607~567	r_RdDataAddr_Scan[42:2]	读 ICache 站台地址。
0	608	r_RdDataHold_Scan	读 ICache 站台流水线阻塞。
0	652~650	i_PipeState[2:0]	流水线取指状态。
0	654~653	i_FlushState[1:0]	Icache 刷新状态。
0	659~655	r_QueryState[4:0]	取指查询 Icache 状态。
0	662~660	i_RecvState[2:0]	装填状态机状态。
0	663	i_PRABFree	取指悬挂缓冲空。
0	664	i_PRABFull	取指悬挂缓冲满。
0	1211~1208	i_InstValid[3:0]	4 条指令有效信息。
0	1217~1212	i_CurState[5:0]	状态机状态。
0	1218	i_MapHalt	指示 Map 站台该周期指令需要停顿，下一周期不接收新指令。
0	1791~1536	gr_IER_Scan[255:0]	其它中断使能寄存器。
0	1794~1792	gr_II_IER[2:0]	核间中断使能寄存器。
0	1810~1795	gr_II0_Scan[5:0]	核间中断寄存器 0。
0	2066~1811	r_INT_STAT[255:0]	中断状态寄存器。
0	2098~2067	gr_EXC_SUM_Scan[31:0]	异常摘要寄存器。
0	2141~2099	gr_EXC_PC_Scan[42:0]	异常地址寄存器。
0	2565~2560	r_InumCounter_Scan[5:0]	Inum 计数器，分配 Inum 的基准。
0	2574	i_MapInst0Valid	指令 0 有效。
0	2577~2575	i_Inst0Type[2:0]	指令 0 类型。
0	2580～2578	i_Inst1Type[2:0]	指令 1 类型。
0	2581	i_MapInst1Valid	指令 1 有效。
0	2587~2582	i_MapFreeIntReg[5:0]	空闲的整数寄存器。
0	2593~2588	i_MapFreeFltReg[5:0]	空闲的浮点寄存器。
0	2594	i_DisableCSRIssue	CSR 记分板控制，不能发射。
0	2595	c_DisableFltIssue	FPCR 记分板控制，浮点不能发射。
0	2596	c_DisableRenameIssue	重命名冲突不能发射。
0	2597	c_DisableSpecInst	ROB 非空，读 FPCR 不能发射。
0	2598	c_DisableMemIssue	访存指令不能发射。
0	2599	i_FPCROff	FPCR 锁记分板。

0	3072	ROB_Empty	表示重排序缓冲（ROB）空。
0	3073	ROB_Full	表示重排序缓冲（ROB）满。
0	3074	ROB_Retire_TimeOut	重排序缓冲中的指令退出超时（指该指令有效维持了 2^{16} 个核心时钟）。
0	3080~3075	ROB_QT[5:0]	表示重排序缓冲（ROB）尾指针。
0	3121~3081	ROB_PC[42:2]	最近退出的一条指令的虚地址。
0	3123~3122	ROB_CurMode[1:0]	最近退出的一条指令所在的处理器模式。
0	3130~3124	ROB_QH[6:0]	表示重排序缓冲（ROB）头指针。
0	3171~3031	c_ROBQHInstAddr[42:3]	准备退出的指令 PC 值。
0	3173~3072	c_ROBQHInstCurPM	准备退出的指令模式。
0	3176~3174	ROB0QH0InstType	表示重排序缓冲（ROB）头指针 0 位置指令类型。 000：整数运算； 001：浮点运算； 010：读访存指令； 011：写访存指令； 其它：保留。
0	3177	ROB0QH0Complete	表示重排序缓冲（ROB）头指针 0 位置指令完成标志。
0	3178	ROB0QH0NoFault	表示重排序缓冲（ROB）头指针 0 位置指令错误标志。
0	3181~3179	ROB1QH0InstType	表示重排序缓冲（ROB）头指针 1 位置指令类型。 000：整数运算； 001：浮点运算； 010：读访存指令； 011：写访存指令； 其它：保留。
0	3182	ROB1QH0Complete	表示重排序缓冲（ROB）头指针 1 位置指令完成标志。
0	3183	ROB1QH0NoFault	表示重排序缓冲（ROB）头指针 1 位置指令错误标志。
0	3595~3584	IQValid[11:0]	IQ 发射队列有效。

0	3607~3596	IQUCanIssue[11:0]	IQ 发射队列上部件可以发射。
0	3619~3608	IQLCanIssue[11:0]	IQ 发射队列下部件可以发射。
0	3629~3620	FQValid[9:0]	FQ 发射队列有效。
0	3639~3630	FQCanIssue[9:0]	FQ 发射队列可以发射。
一级数据 Cache（DBOX）			
7	0	CSR2OPU_FOW	指示在数据流写访问一个具有 FOW 属性的页面而产生的写故障。
6	0	CSR2OPU_FOR	指示在数据流读访问一个具有 FOR 属性的页面而产生的读故障。
5	0	CSR2OPU_ACV0	指示在数据流访问过程中出现存取越权。
4	0	CSR2OPU_DATERR	访存指令读 DCache 的数据阵列时产生 ECC 校验错。
7	1	CSR2OPU_DAMATCH	指示数据流地址匹配 IPR 寄存器 DA_MATCH 内容。
6	1	CSR2OPU_ACV1	指示 I/O 空间的访问数据超长，即 VLDS、VLDD、VSTS、VSTD 指令产生的数据流地址为 I/O 空间地址。
5	1	CSR2OPU_BADDVA	指示访存类指令的数据流地址产生符号扩展错。
4	1	CSR2OPU_TAGPERR	指示访存指令查询 DCache 的 TAG 产生偶校验错。
[7:4]	2	CSR2OPU_SETEN[3:0]	DTAG 的组使能。
7	3	CSR2OPU_CSRErr	一条写 CSR 指令未退出时 Dbox 又接收到一条新的 CSR 写指令。
6	3	CSR2OPU_DATMErr	SQ 中写指令在读出 DCache 中数据进行“读修改写”时，产生不能被写数据完全覆盖的 ECC 不可纠正多错。
5	3	CSR2OPU_ProbeErr	Cbox 送 Mbox 的查询请求与 MAF、LQ 或 SQ 中的请求状态不一致。
4	3	CSR2OPU_AckErr	访存指令产生请求，通过 Cbox 发给系统，从系统返回 Cbox 的响应是错误的响应。
4	4	LQ2OPU_FstValid	Load 指令队列（LQ）头条目有效位。

[7:5]	4	LQ2OPU_FstInum[2:0]	Load 指令队列（LQ）头条目年龄[2:0]。
[7:4]	5	LQ2OPU_FstInum[6:3]	Load 指令队列（LQ）头条目年龄[6:3]。
4	6	LQ2OPU_FstComplete	Load 指令队列（LQ）头条目完成标志。
5	6	LQ2OPU_FstRetire	Load 指令队列（LQ）头条目可退出标志。
6	6	LQ2OPU_FstErrorFlag	Load 指令队列（LQ）头条目错误标志。
7	6	LQ2OPU_FstReplay	Load 指令队列（LQ）头条目重放标志。
4	7	LQ2OPU_FstIsIOSpace	Load 指令队列（LQ）头条目 IO 指示。
5	7	LQ2OPU_FstIsAtom	Load 指令队列（LQ）头条目原子操作指示。
6	7	LQ2OPU_FstMAFValid	Load 指令队列（LQ）头条目 MAF 有效标志。
7	7	LQ2OPU_FstNCCache	Load 指令队列（LQ）头条目不可 Cache 访问标志。
4	8	SQ2OPU_FstValid	Store 指令队列（SQ）头条目有效位。
[7:5]	8	SQ2OPU_FstInum[2:0]	Store 指令队列（SQ）头条目年龄[2:0]。
[7:4]	9	SQ2OPU_FstInum[6:3]	Store 指令队列（SQ）头条目年龄[6:3]。
4	10	SQ2OPU_FstComplete	Store 指令队列（SQ）头条目完成标志。
5	10	SQ2OPU_FstDirty	Store 指令队列（SQ）头条目可写标志。
6	10	SQ2OPU_FstRetire	Store 指令队列（SQ）头条目可退出标志。
7	10	SQ2OPU_FstErrorFlag	Store 指令队列（SQ）头条目错误标志。
4	11	SQ2OPU_FstReplay	Store 指令队列（SQ）头条目重放标志。
5	11	SQ2OPU_FstIsIOSpace	Store 指令队列（SQ）头条目 IO 指示。
6	11	SQ2OPU_FstIsNCpace	Store 指令队列（SQ）头条目不可 Cache 访问指示。
7	11	SQ2OPU_FstMAFValid	Store 指令队列（SQ）头条目 MAF 有效标志。
[7:6]	12	PPU2OPU_ProbeState[1:0]	Probe 的状态。
[5:4]	12	SQ2OPU_SQMBFSMStat[1:0]	指示 Mbox 写合并缓冲状态。
7	13	MAF2OPU_MAFReqValid	表示 MAF 给 SBX 的请求有效
6	13	MAF2OPU_MAFEmptyFlag	MAF 空标志
[5:4]	13	PPU2OPU_ProbeState[3:2]	Probe 的状态。

7	14	PFH2OPU_Full	指示流预取缓冲满
6	14	PFH2OPU_Empty	指示流预取缓冲空
5	14	CSR2OPU_MEM_MERR	指示对存储器执行读操作时，监测到了 ECC 多错
4	14	CSR2OPU_SIMD_MV	指示普通 SIMD 访存指令的地址，对指令访存粒度而言不对界，但对元素访问粒度而言对界
7	15	SQ2OPU_FstNCCache	指示存储队列（SQ）头条目不可 Cache 访问标志。
6	15	CSR2OPU_ECBErrEn	ECB 报错使能。
5	15	CSR2OPU_ECCErrEn	ECC 报错使能。
4	15	CSR2OPU_PRB_CTRL	Probe 控制使能。
[7:4]	323~320	r_HeadIndex_Scan[15:0]	LQ 头指针指示。
[7:4]	331~328	r_LQRBValidBits_Scan[15:0]	LQ 有效位指示。
[7:4]	339~336	r_IORReplayIndex_Scan[15:0]	LQ 重放指针。
[7:4]	883~880	r_SQHeadIndex_Scan[15:0]	SQ 头指针。
[7:4]	871~868	r_SQCUValidBits_Scan[15:0]	SQ 有效位。
[7:4]	879~876	r_ReplayReqIndex_Scan[15:0]	重放指针。
二级 Cache（S_SBOX）			
2	59	SPBQ_ProbeValid	二次请求队列输出请求有效。
2	60	SPBQ_BadProbe	二次请求队列保留的二次请求编码。
2	61	SPBQ_OverFlow	二次请求队列队列溢出。
2	62	SPBQ_curempty	二次请求队列队列空。
2	63	SPBQ_curfull	二次请求队列队列满。
2	702	SVAF1full	SVA1 缓冲满。
2	703	SVAF1Empty	SVAF1 缓冲空。
2	988	SVAF2ProbeValid	SVAF2 存在用于一致性二次请求的条目有效。
2	989	SVAF2full	SVAF2 缓冲满。
2	990	SVAF2Empty	SVAF2 缓冲空。
2	991	WrBackMissVAF2	淘汰的二次请求不命中 SVAF2 错。
2	1534	SMAFfull	SMA 缓冲满。
2	1535	SMAFEmpty	SMA 缓冲空。

2	1821	SRDQEmpty	SRDQ 空信号。
2	1822	SRDQNearFull	SRDQ 几乎满。
2	1823	SRDQFull	SRDQ 满信号。
2	2206	RetryValid	重试请求有效。
2	2400	r_IntReq_Valid_Scan	中断请求有效。
2	2401	c_AllSetEn	所有路使能。
2	2402	r_TagSErrLock	STAG 校验单错信号，不受使能控制。
2	2403	r_TagMErrLock	STAG 校验多错信号，不受使能控制。
2	2404	r_DatSErrLock	SCACHE 数据单错信号，不受使能控制。
2	2405	r_DatMErrLock	SCACHE 数据多错信号，不受使能控制。
2	2406	r_DvicSErrLock	DCACHE 回写数据单错，不受使能控制。
2	2407	r_DvicMErrLock	DCACHE 回写数据多错，不受使能控制。
2	2408	r_CacheStErrLock	SCACHE 状态错误标志，不受使能控制。
2	2409	r_WrErrorLock	写请求地址非法错误，不受使能控制。
2	2410	r_SYSACK_ECCERRLock	外部装填控制校验错，不受使能控制。
2	2411	r_SYSDAT_SERRLock	外部装填数据校验单错，不受使能控制。
2	2412	r_SYSDAT_MERRLock	外部装填数据校验多错，不受使能控制。
2	2413	r_SYSREQ2_ECCERRLock	外部二次请求校验错，不受使能控制。
2	2414	r_SCTAGErrEn_Scan	STAG 校验错误使能。
2	2415	r_SCDATErrEn_Scan	SCACHE 和 DCACHE 数据校验使能。
2	2416	r_CacheStErrEn_Scan	SCACHE 非法状态报错使能。
2	2417	r_MBCtl_Scan	MB 对 SVAF2 淘汰的控制使能。
2	2418	r_IsFillCtl_Scan	指令装填控制，1 表示不装填 scache。
2	2419	r_WrErrorEn_Scan	写越界报错使能。
2	2420	r_AckChkEn_Scan	ACK 类型匹配报错使能。
2	2421	r_SYSACK_ERR_EN_Scan	ACK 包头偶校验错误使能。

2	2422	r_SYSDAT_ERR_EN_Scan	响应数据 ECC 校验错误使能。
2	2423	r_SYSREQ_ERR_EN_Scan	二次请求偶校验错误使能。
2	2424	r_VictimFirst_Scan	淘汰优先发送标志。
2	2425	r_CVBEnable_Scan	清洁淘汰使能。
2	2426	r_CTDSelClean_Scan	清洁置脏选择。
2	2427	r_ELBEnable_Scan	EvictLocal 使能。
2	2428	r_EGBEnable_Scan	EvictGlobal 使能。
2	2429	r_MembEnable_Scan	MemB 外部使能。
2	2430	r_CleanInter_Scan	独占清洁内部确认使能。
2	2449~2440	r_MBCnt_Scan[9:0]	Memb 计数器。
2	2450	gr_MembMarkDone	MB 标志的二次请求处理完成。
2	2451	gr_MBDone	MB 被外部确认标志。
2	2452	r_DBX2SBX_MAFIsEmpty	表示 Dbox 的 MAF 空。
2	2453	r_IBX2SBX_PipeLineNoMem	表示 Map 后的站台和缓冲无访存指令。
2	2454	r_IBX2SBX_MapNoMem	表示 Map 站台的访存请求空。
2	2455	r_IBX2SBX_MBReq	MB 请求有效。
2	2456	o_MembReq	MB 请求有效。
2	2457	r_MBCntUnderFlow	MB 计数器下溢出。
2	2458	o_MBCntEmpty	MB 计数器空。
2	2459	r_MBCntFull	MB 计数器满。
2	2463~2460	r_MBState_Scan[3:0]	Memb 控制状态机。
2	2555	o_ScmdValid	系统命令队列输出有效。
2	2556	r_SCMQOverFlow	系统命令队列写溢出。
2	2557	o_CmdQEmpty	系统命令队列供流水线用的条目空。
2	2558	o_CmdQNearFull	系统命令队列供流水线用条目几乎满。
2	2559	o_CmdQFull	系统命令队列供流水线用的条目满。
2	2589	r_SRPQOverFlow	回答队列溢出。
2	2590	o_SRPQEmpty	回答队列空。
2	2591	c_SRPQFull	回答队列满。
2	2743	c_AckHeadQFull	ACK 响应包头队列满。
2	2744	c_AckHeadQEmpty	ACK 响应包头队列空。
2	2745	c_AKDQFull	ACK 响应数据队列满。

2	2746	c_AKDQEmpty	ACK 响应数据队列空。
2	2747	r_AckHeadQVFlow	ACK 包头队列溢出。
2	2749	r_AckDataBufUFlow	ACK 数据缓冲下溢出。
2	2751	r_BadSysAck	发现保留的 ACK 类型编码。
2	3659	c_Req1NotEqReq1	SMAF 一次请求与 ACK 携带的不同。
2	3660	c_HitVAF1VAF2	同时命中 SVAF1 和 SVAF2 错误。
2	3663~3661	c_RQ2ErrCode[2:0]	二次请求之间错误编码。
2	3664	c_RQ2HitRQ2Error	二次请求之间错误。
2	3669~3665	c_ObsSC_ErrCode[4:0]	SBOX 错误类型编码。
2	3670	c_ObsIllegalScState	STAG 非法状态。
2	3671	c_ObsSVAF1MAFErr	SVAF1 或者 SMAF 控制错误。
2	3672	c_ObsDVic_MisMatch	淘汰与 STAG 状态不匹配。
2	3673	c_ObsDS_MisMatch	数据流与 STAG 状态不匹配。
2	3674	c_ObsMCRQ2_MisMatchSC	二次请求与 STAG 状态不匹配。
2	3675	c_ObsMCAck_MisMatchSC	ACK 与 STAG 状态不匹配。
2	3676	c_ObsCNF_HitVAF2	SVAF2 控制错误。
2	3677	c_ObsMCAck_MisMatchCmd	响应类型与一次请求不匹配。
2	3678	c_ObsDSMatchDS	数据流之间冲突。
2	3679	c_ObsMCAck_MissMAF	响应悬挂缓冲不匹配。