致尊敬的顾客

关于产品目录等资料中的旧公司名称

NEC电子公司与株式会社瑞萨科技于2010年4月1日进行业务整合（合并），整合后的新公司暨“瑞萨电子公司”继承两家公司的所有业务。因此，本资料中虽还保留有旧公司名称等标识，但是并不妨碍本资料的有效性，敬请谅解。

瑞萨电子公司网址：http://www.renesas.com

2010年4月1日
瑞萨电子公司

【发行】瑞萨电子公司（http://www.renesas.com）

【业务咨询】http://www.renesas.com/inquiry

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Software Manual

32

SH-4A

瑞萨32位RISC单片机

SuperH™ RISC engine系列

瑞萨电子

www.renesas.com

Rev.1.00 2008.10

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

注意

本文只是参考译文，前页所载英文版“Cautions”具有正式效力。

关于利用本资料时的注意事项

1. 本资料是为了让用户根据用途选择合适的本公司产品的参考资料，对于本资料中所记载的技术信息，并非意味着对本公司或者第三者的知识产权及其他权利做出保证或对实施权力进行的承诺。
2. 对于因使用本资料所记载的产品数据、图、表、程序、算法及其他应用电路例而引起的损害或者对第三者的知识产权及其他权利造成侵犯，本公司不承担任何责任。
3. 不能将本资料所记载的产品和技术用于大规模破坏性武器的开发等目的、军事目的或其他的军需用途方面。另外，在出口时必须遵守日本的《外汇及外国贸易法》及其他出口的相关法令并履行这些法令中规定的必要手续。
4. 本资料所记载的产品数据、图、表、程序、算法以及其他应用电路例等所有信息均为本资料发行时的内容，本公司有可能在未做事先通知的情况下，对本资料所记载的产品或者产品规格进行更改。所以在购买和使用本公司的半导体产品之前，请事先向本公司的营业窗口确认最新的信息并经常留意本公司通过公司主页(http://www.renesas.com)等公开的最新信息。
5. 对于本资料中所记载的信息，制作时我们尽力保证出版时的精确性，但不承担因本资料的叙述不当而致使顾客遭受损失等的任何相关责任。
6. 在使用本资料所记载的产品数据、图、表等所示的技术内容、程序、算法及其他应用电路例时，不仅要对所使用的技术信息进行单独评价，还要对整个系统进行充分的评价。请顾客自行负责，进行是否适用的判断。本公司对于是否适用不负任何责任。
7. 本资料中所记载的产品并非针对万一出现故障或是错误运行就会威胁到人的生命或给人体带来危害的机器、系统(如各种安全装置或者运输交通用的、医疗、燃烧控制、航天器械、核能、海底中继用的机器和系统等)而设计和制造的，特别是对于品质和可靠性要求极高的机器和系统等（将本公司指定用于汽车方面的产品用于汽车时除外）。如果要用于上述的目的，请务必事先向本公司的营业窗口咨询。另外，对于用于上述目的而造成的损失等，本公司概不负责。
8. 除上述第7项内容外，不能将本资料中记载的产品用于以下用途。如果用于以下用途而造成的损失，本公司概不负责。
   1）生命维持装置。
   2）植埋于人体使用的装置。
   3）用于治疗（切除患部、给药等）的装置。
   4）其他直接影响到人的生命的装置。
9. 在使用本资料所记载的产品时，对于最大额定值、工作电源电压的范围、放热特性、安装条件及其他条件请在本公司规定的保证范围内使用。如果超出了本公司规定的保证范围使用时，对于由此而造成的故障和出现的事故，本公司将不承担任何责任。
10. 本公司一直致力于提高产品的质量和可靠性，但一般来说，半导体产品总会以一定的概率发生故障、或者由于使用条件不同而出现错误运行等。为了避免因本公司的产品发生故障或者错误运行而导致人身事故和火灾或造成社会性的损失，希望客户能自行负责进行冗余设计、采取延烧对策及进行防止错误运行等的安全设计（包括硬件和软件两方面的设计）以及老化处理等，这是作为机器和系统的出厂保证。特别是单片机的软件，由于单独进行验证很困难，所以要求在顾客制造的最终的机器及系统上进行安全检验工作。
11. 如果把本资料所记载的产品从其载体设备上卸下，有可能造成婴儿误吞的危险。顾客在将本公司产品安装到顾客的设备上时，请顾客自行负责将本公司产品设置为不容易剥落的安全设计。如果从顾客的设备上剥落而造成事故时，本公司将不承担任何责任。
12. 在未得到本公司的事先书面认可时，不可将本资料的一部分或者全部转载或者复制。
13. 如果需要了解关于本资料的详细内容，或者有其他关心的问题，请向本公司的营业窗口咨询。

产品使用时的注意事项

本文对适用于单片机所有产品的 “使用时的注意事项”进行说明。有关个别的使用时的注意事项请参照正文。此外，如果在记载上有与本手册的正文有差异之处，请以正文为准。

1. 未使用的引脚的处理
   【注意】将未使用的引脚按照正文的 “未使用引脚的处理”进行处理。
   CMOS产品的输入引脚的阻抗一般为高阻抗。如果在开路的状态下运行未使用的引脚，由于感应现象，外加LSI周围的噪声，在LSI内部产生穿透电流，有可能被误认为是输入信号而引起误动作。
   未使用的引脚，请按照正文的 “未使用引脚的处理”中的指示进行处理。

2. 通电时的处理
   【注意】通电时产品处于不定状态。
   通电时， LSI内部电路处于不确定状态，寄存器的设定和各引脚的状态不定。通过外部复位引脚对产品进行复位时，从通电到复位有效之前的期间，不能保证引脚的状态。
   同样，使用内部上电复位功能对产品进行复位时，从通电到达到复位产生的一定电压的期间，不能保证引脚的状态。

3. 禁止存取保留地址 （保留区）
   【注意】禁止存取保留地址 （保留区）
   在地址区域中，有被分配将来用作功能扩展的保留地址 （保留区）。因为无法保证存取这些地址时的运行，所以不能对保留地址 （保留区）进行存取。

4. 关于时钟
   【注意】复位时，请在时钟稳定后解除复位。
   在程序运行中切换时钟时，请在要切换成的时钟稳定之后进行。复位时，在通过使用外部振荡器（或者外部振荡电路）的时钟开始运行的系统中，必须在时钟充分稳定后解除复位。另外，在程序运行中，切换成使用外部振荡器 （或者外部振荡电路）的时钟时，在要切换成的时钟充分稳定后再进行切换。

5. 关于产品间的差异
   【注意】在变更不同型号的产品时，请对每一个产品型号进行系统评价测试。
   即使是同一个群的单片机，如果产品型号不同，由于内部ROM、版本模式等不同，在电特性范围内有时特性值、动作容限、噪声耐量、噪声辐射量等也不同。因此，在变更不认同型号的产品时，请对每一个型号的产品进行系统评价测试。

本书的结构

本书按以下结构制作而成：

1.产品的一般注意事项
2.本书的结构
3.前言
4.目录
5.概要
6.各功能模块的说明
- CPU及系统控制体系
- 内部外围模块
  各模块的功能说明结构因模块的不同而各异，一般由
  ①特点、②输入/输出引脚、③寄存器说明、④运行说明、⑤使用注意事项等章节构成。

设计采用本 LSI 的应用系统时，请充分确认注意事项以后再进行设计。 各章的正文中有关于说明的注意事项，各章的结尾有使用注意事项。必须仔细阅读。（使用注意事项是根据需要记载的。）

7.寄存器一览表
8.索引

前言

SH-4A 是瑞萨 科技独创的 RISC 方式的 CPU 内核。

对象　　本手册以设计使用了 SH-4A 的应用系统的用户为对象。
使用本手册的读者需具备电气电路、逻辑电路、单片机及与汇编 /C 语言编程等相关基础知识。

目的　　本手册以理解 SH-4A 的指令为目的。有关硬件的内容，请参考该产品的硬件手册。

阅读方法

- 希望理解整体功能时
  → 请按照目录阅读。
  本书按顺序大体分为CPU功能、系统控制功能及各指令的说明。
- 希望理解各指令时
  → 指令体系及基本运行在“**第3章 指令系统**”进行说明。各指令的详细操作请阅读“**第10章 各指令的说明**”。

凡例　寄存器的表示　：串行通信等相同或类似的功能存在于多个通道时，使用下列描述方式：
XXX_N （XXX 为基本的寄存器名称， N 为通道号）
位的表示　：按照左侧为高位，右侧为低位的顺序描述。
数字的表示　：2 进制数表示为 B'XXXX，16 进制数表示为 H'XXXX，10 进制数表示为 XXXX。
符号的表示　：低电平有效信号带有上划线 （$\overline{\text{XXXX}}$）。

省略语的说明

ALU	Arithmetic Logic Unit 算术逻辑单元
ASID	ASIDAddress Space Identifier 地址空间标识符
CPU	Central Processing Unit 中央处理器
FPU	Floating Point Unit 浮点运算设备
LRU	Least Recently Used 最近最少使用
LSB	Least Significant Bit 最低位
MMU	Memory Management Unit 存储器管理单元
MSB	Most Significant Bit 最高位
PC	Program Counter 程序计数器
RISC	Reduced Instruction Set Computer 精简指令系统计算机
TLB	Translation Lookaside Buffer 变换旁查缓冲器

目　录

第 1 章 概要

1.1 SH-4A 的特点

SH-4A 为 32 位 RISC （精简指令系统计算机）单片机。与 SH-1、 SH-2、 SH-3、 SH-4 单片机在指令系统级高位兼容。通过 16 位固定长的指令系统、与 32 位指令相比可将程序代码长度大体缩小 50%。

SH-4A 的特点如**表** 1.1 所示。

表 1.1 SH-4A 的特点

项目	特点
CPU	• 瑞萨科技独创体系结构 • 32 位内部数据总线 • 通用寄存器文件： 16 个 32 位通用寄存器 （及 8 个 32 位影像寄存器） 7 个 32 位控制寄存器 4 个 32 位系统寄存器 • RISC 型指令系统 （与 SH-1、 SH-2、 SH-3、 SH-4 高位兼容）： 指令长度：用于改善代码效率的 16 位固定长度 加载存储结构 延迟转移指令 带条件执行 基于 C 语言的指令系统 • 包含 FPU 的 2 条指令同时执行型超标量 • 指令执行时间： 最多 2 条指令 / 周期 • 虚拟地址空间： 4G 字节 • 空间标识符 ASID：8 位、 256 个虚拟地址空间 • 内置乘法器 • 7 段流水线

项目	特点
浮点单元 （FPU）	• 内置浮点协处理器 • 支持单精度 （32 位）及双精度 （64 位） • 支持遵照 IEEE754 的数据类型及异常 • 舍入模式：向接近方向及 0 方向舍入 • 非规格化数的处理：舍为 0 或产生以 IEEE754 为依据的中断 • 浮点寄存器：32 位 ×16 字 ×2 个存储体 （单精度 ×16 字或双精度 ×8 字） ×2 个存储体 • 32 位 CPU-FPU 浮点通信寄存器 （FPUL） • 支持 FMAC （乘法及累加）指令 • 支持 FDIV （除法） /FSQRT （平方根）指令 • 支持 FLDI0/FLDI1 （加载常数 0/1）指令 • 指令执行时间 等待时间 （FADD/FSUB）：3 个周期 （单精度）、 5 个周期 （双精度） 等待时间 （FMAC/FMUL）：5 个周期 （单精度）、 7 个周期 （双精度） 节距 （FADD/FSUB）：1 个周期 （单精度 / 双精度） 节距 （FMAC/FMUL）：1 个周期 （单精度）、 3 个周期 （双精度） 【注】：FMAC 只支持单精度。 • 3D 图形指令 （仅单精度）： 4 维向量转换及矩阵运算 （FTRV）、 4 个周期 （节距）、 8 个周期 （等待时间） 4 维向量 （FIPR）的内积、 1 个周期 （节距）、 5 个周期 （等待时间） • 10 段流水线
存储器管理单元 （MMU）	• 4G 字节的地址空间、 256 地址空间标识符 （ASID 8 位） • 单一虚拟内存模式和多重虚拟内存模式 • 支持多种页大小： 1K、 4K、 64K、 1M 字节 • 对指令的 4 个入口的全相联 TLB • 对指令及操作数的 64 个入口的全相联 TLB • 支持通过软件的更换方法及随机计数器方式更换算法 • 通过地址映射可直接存取 TLB 的内容
高速缓冲存储器	• 指令高速缓存 （IC） 4 路成组相联 32 字节块长度 • 操作数高速缓存 （OC） 4 路成组相联 32 字节块长度 可选择的写入方式 （备份 / 直写） • 存储队列 （32 字节 ×2 个入口） 【注】关于指令高速缓存、操作数高速缓存的高速缓存容量，请参考产品的硬件手册。
L 存储器	• 2 个独立的读取 / 写入端口 从 CPU 进行 8/16/32/64 位存取 根据外部请求进行 8/16/32/64 位及 16/32 字节存取 【注】关于 L 存储器的容量，请参考产品的硬件手册。

1.2 从 SH-4 到 SH-4A 的变更点

本手册以章、节为单位，说明从 SH-4 到 SH-4A 的变更点。

表 1.2 从 SH-4 到 SH-4A 的变更点

章号、章名	节号	节名	变更点
第 1 章 概要	—	—	全面变更 （个别变更点在第 2 章以后的差分中有所记述。）
第 2 章 编程模型	2.2	寄存器结构	浮点状态 / 控制寄存器 （FPSCR） 追加 SZ=1, PR=1 时的运行
第 3 章 指令系统	3.3	指令系统	追加 9 条指令作为 CPU 指令
			追加 3 条指令作为 FPU 指令
第 4 章 流水线操作	4.1	流水线	将流水线段数从 5 变更为 7
	4.2	并列执行性	追加 9 条指令作为 CPU 指令
			追加 3 条指令作为 FPU 指令
			变更指令的组划分及并列执行组合
	4.3	执行周期	变更执行周期数
第 5 章 异常处理	—	—	无
第 6 章 浮点单元 （FPU）	6.3.2	浮点状态 / 控制寄存器（FPSCR）	追加 SZ=1, PR=1 时的运行，追加每个字节序的运行说明
	6.5	浮点异常	设定允许 FPU 异常时，变更 FPU 异常检测条件的规格
第 7 章 存储器管理单元（MMU）	7.1.1	地址空间	变更 P4 区域的结构
			删除内部 RAM 空间
	7.2	寄存器的说明	删除页表入口辅助寄存器 （PTEA）
			追加物理地址空间控制寄存器
	7.2.6	物理地址空间控制寄存器（PASCR）	新追加
	7.2.7	指令再取禁止控制寄存器（IRMCR）	新追加
	7.3	TLB 的功能	从 TLB 删除空间属性位 （SA[2:0] ）、时序控制位（TC）
	7.4.5	避免同义问题	变更高速缓存大小，删除变址模式，相应位也随之发生变更
	7.5.1、7.5.4	指令 TLB 多重命中异常及数据 TLB 多重命中异常	在 ITLB 未命中运转导致的 UTLB 检索期间，变为多重命中时，也变更为指令 TLB 多重命中异常
	7.6	存储器分配 TLB 的结构	删除 ITLB 及 UTLB 的数据阵列 2
	7.6.3	UTLB 地址阵列	对 UTLB 地址阵列的联想写入变更为不发生数据 TLB 多重命中异常
			将存储器分配地址从 H'F600 0000 ～ H'F6FF FFFF 变更为 H'F600 0000 ～ H'F60F FFFF
	7.6.4	UTLB 数据阵列	将存储器分配地址从 H'F700 0000 ～ H'F77F FFFF 变更为 H'F700 0000 ～ H'F70F FFFF
	7.7	32 位地址扩展模式	新追加

章号、章名	节号	节名	变更点
第 8 章 高速缓存	8.1	特点	将指令高速缓存的容量变更为 32K 字节
			将方式变更为 4 路成组相联
	8.2	寄存器的说明	追加内部存储器控制寄存器
	8.2.1	高速缓存控制寄存器（CCR）	变更内容
	8.2.4	内部存储器控制寄存器（RAMCR）	新追加
	8.3	操作数高速缓存的运行说明	删除 RAM 模式及 OC 变址模式
	8.3.6	OC 2 路模式	新追加
	8.4	指令高速缓存的运行说明	删除 IC 变址模式
	8.4.3	IC 2 路模式	新追加
	8.5.1	高速缓存与外部存储器的一致性	追加 ICBI 指令、 PREFI 指令及 SYNCO 指令
	8.6	存储器分配高速缓存的结构	随着容量的变更及 4 路成组相联化，变更入口位及路位
	8.8	使用 32 位地址扩展模式时的注意事项	新追加
第 9 章 L 存储器	—	—	新追加
第 10 章 各指令的说明	—	—	追加 9 条指令作为 CPU 指令
			追加 3 条指令作为 FPU 指令

第 2 章 编程模型

本章就 SH-4A 的编程模型进行记述。 SH-4A 具有以下所示的寄存器及数据格式。

2.1 数据格式

SH-4A 所支持的数据格式如图 2.1 所示。

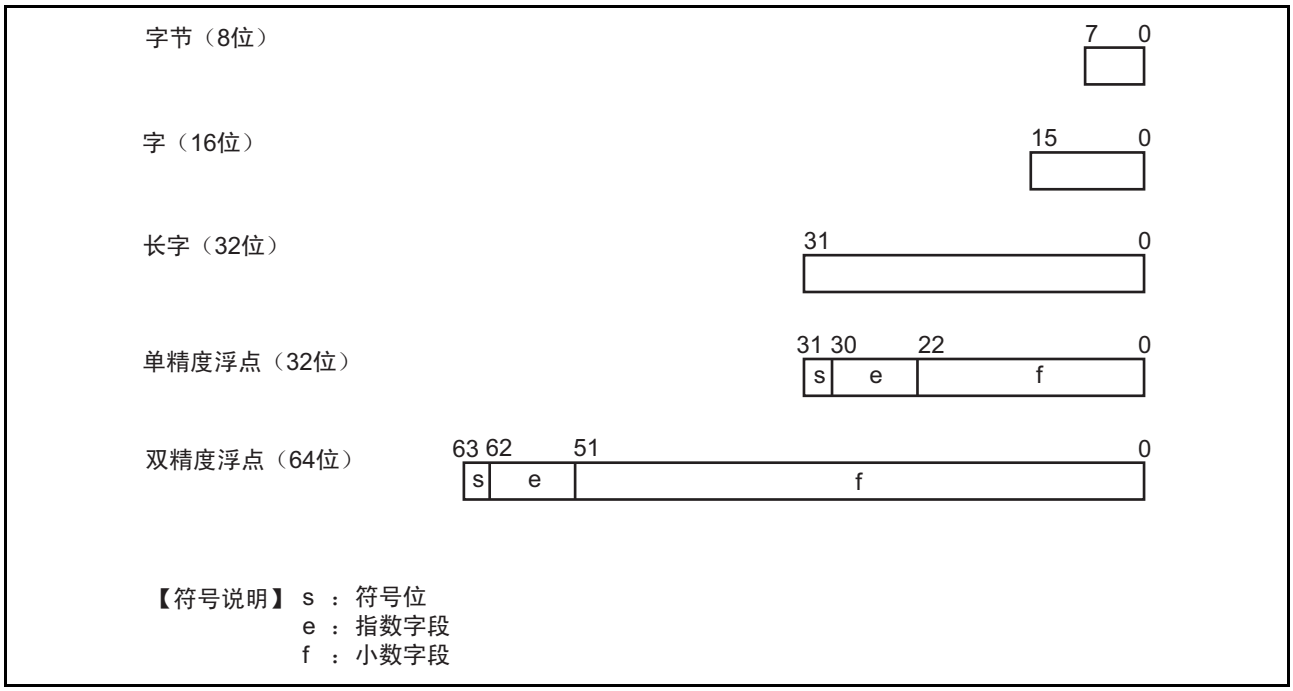


图 2.1 数据格式

2.2 寄存器的结构

2.2.1 特权模式与存储体

(1) 处理模式

处理模式有用户模式与特权模式 2 种。通常是在用户模式下运行，发生异常或接受中断时变为特权模式。寄存器分为通用寄存器、系统寄存器、控制寄存器及浮点寄存器。在不同的处理模式下可存取的寄存器也不同。

(2) 通用寄存器

通用寄存器具有从 R0 到 R15 共 16 个寄存器。通用寄存器 R0 到 R7 为存储体寄存器，可通过处理模式切换。

- 特权模式时
  通过状态寄存器 （SR）的寄存器存储体位 （RB）、确定可作为通用寄存器进行存取的寄存器与不可作为通用寄存器进行存取的寄存器。不可作为通用寄存器进行存取的寄存器可通过控制寄存器的加载指令 （LDC）与存储指令 （STC）进行存取。
  RB位为1时、即选择存储体1时，存储体1的通用寄存器R0_BANK1到R7_BANK1与同存储体无关的R8到R15共计16个寄存器，可作为通用寄存器R0到R15进行存取，存储体0的通用寄存器R0_BANK0到R7_BANK0这8个寄存器可通过LDC/STC指令进行存取。
  RB位为0时、即选择存储体0时，存储体0的通用寄存器R0_BANK0到R7_BANK0与同存储体无关的R8到R15共计16个寄存器，可作为通用寄存器R0到R15进行存取，存储体1的通用寄存器R0_BANK1到R7_BANK1这8个寄存器可通过LDC/STC指令进行存取。
- 用户模式时
  存储体0的通用寄存器R0_BANK0到R7_BANK0与同存储体无关的R8到R15共计16个寄存器，可作为通用寄存器R0到R15进行存取，存储体1的通用寄存器R0_BANK1到R7_BANK1这8个寄存器不可进行存取。

(3) 控制寄存器

在处理模式中，控制寄存器有共用的全局基址寄存器 （GBR）和状态寄存器 （SR）以及只能在特权模式中存取的保存状态寄存器 （SSR）、保存程序计数器 （SPC）、向量基址寄存器 （VBR）、保存通用寄存器 15 （SGR）、调试基址寄存器 （DBR）。状态寄存器有只能在特权模式中存取的位 （如 RB 位）。

(4) 系统寄存器

系统寄存器包含乘加寄存器 （MACH/MACL）、过程寄存器 （PR）、程序计数器 （PC），并且与处理模式无关。

(5) 浮点寄存器与 FPU 相关的系统寄存器

浮点寄存器有 FR0 ～ FR15、 XF0 ～ XF15 共计 32 个寄存器。可选择是否将 FR0 ～ FR15、 XF0 ～ XF15 分别分配至 FPR0_BANK0 ～ FPR15_BANK0、 FPR0_BANK1 ～ FPR15_BANK1 的任一存储体。

另外， FR0 ～ FR15 可作为 8 个 DR0/2/4/6/8/10/12/14 （双精度浮点寄存器或寄存器对）或 4 个 FV0/4/8/12 （寄存器向量）使用、 XF0 ～ XF15 可作为 8 个 XD0/2/4/6/8/10/12/14 （寄存器对）或 1 个 XMTRX （寄存器矩阵）使用。

FPU 相关的系统寄存器包含浮点交流寄存器 （FPUL）与浮点状态 / 控制寄存器 （FPSCR），用于进行 FPU-CPU 间的通信、异常处理的设定。

复位后寄存器的值如**表 2.1** 所示。

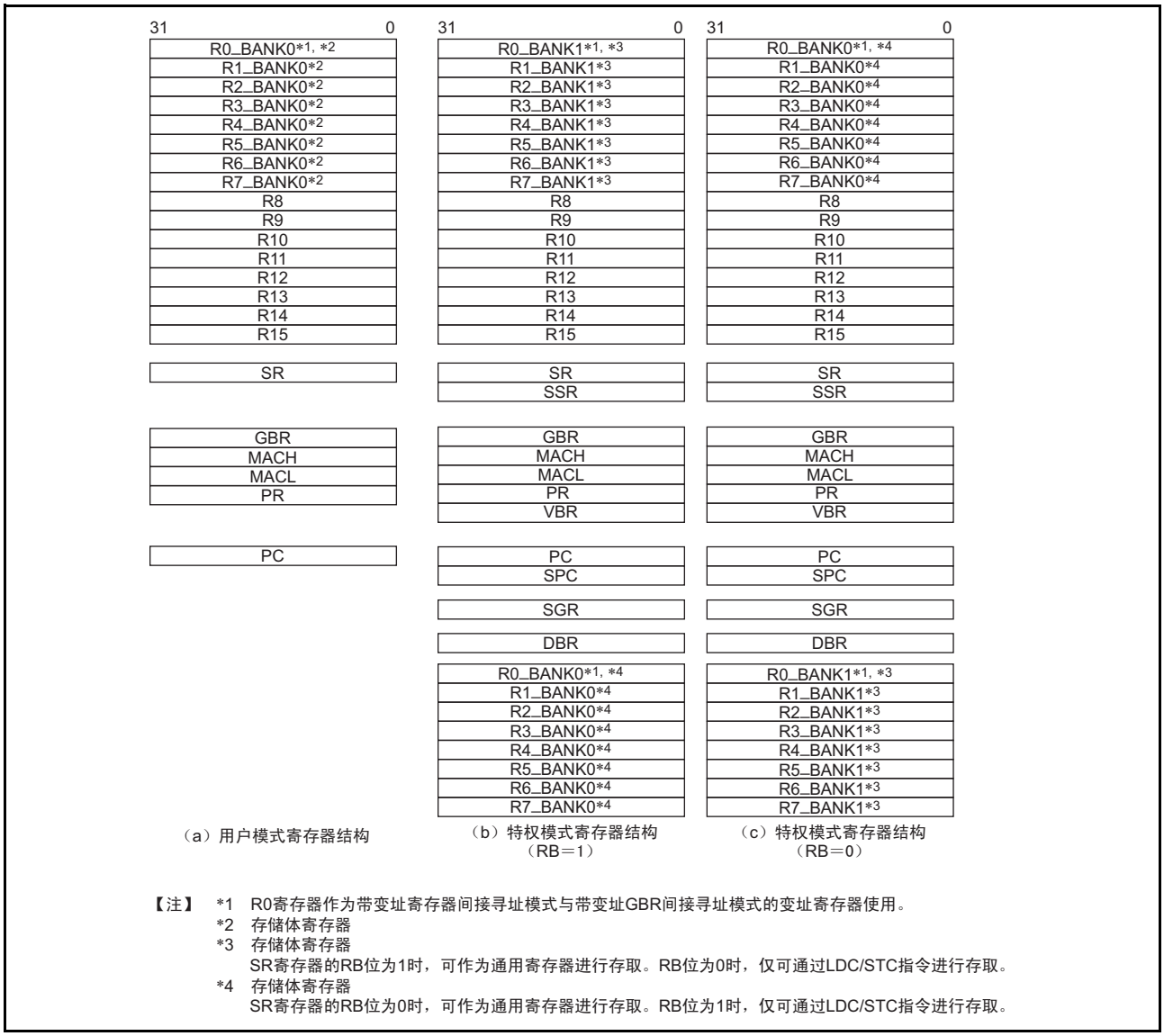
表 2.1 寄存器初始值

区分	寄存器	初始值 *
通用寄存器	R0_BANK0 ～ R7_BANK0、 R0_BANK1 ～ R7_BANK1、 R8 ～ R15	不定
控制寄存器	SR	MD 位为 1、 RB 位为 1、 BL 位为 1、 FD 位为 0、 IMASK 为 B'1111、保留位为 0、其它为不定
	GBR、 SSR、 SPC、 SGR、 DBR	不定
	VBR	H'00000000
系统寄存器	MACH、 MACL、 PR	不定
	PC	H'A0000000
浮点寄存器	FR0 ～ FR15、 XF0 ～ XF15、 FPUL	不定
	FPSCR	H'00040001

【注】 * 通过上电复位、手动复位被初始化。

不同处理模式下的 CPU 寄存器结构如**图 2.2** 所示。

通过状态寄存器的处理模式位 （MD）切换用户模式与特权模式。



【注】 *1 R0寄存器作为带变址寄存器间接寻址模式与带变址GBR间接寻址模式的变址寄存器使用。
*2 存储体寄存器
*3 存储体寄存器
SR寄存器的RB位为1时，可作为通用寄存器进行存取。RB位为0时，仅可通过LDC/STC指令进行存取。
*4 存储体寄存器
SR寄存器的RB位为0时，可作为通用寄存器进行存取。RB位为1时，仅可通过LDC/STC指令进行存取。

图 2.2 不同处理模式下的 CPU 寄存器结构

2.2.2 通用寄存器

处理模式与通用寄存器的关系如图 2.3 所示。 SH-4A 具有 24 个 32 位通用寄存器 （R0_BANK0 ～ R7_BANK0、R0_BANK1 ～ R7_BANK1、R8 ～ R15）。但是其中仅有 16 个寄存器在一种处理模式下可作为通用寄存器 R0 ～ R15 进行存取。 SH-4A 有特权模式与用户模式 2 种处理模式。根据这 2 种模式如下分配 R0 ～ R7。

- R0_BANK0～R7_BANK0
  用户模式 （SR.MD=0）下、常分配至R0～R7。
  特权模式 （SR.MD=1）下、仅在 （SR.RB=0）时分配至R0～R7。
- R0_BANK1～R7_BANK1
  用户模式下不可存取。
  在特权模式下、仅在 （SR.RB=1）时分配至R0～R7。

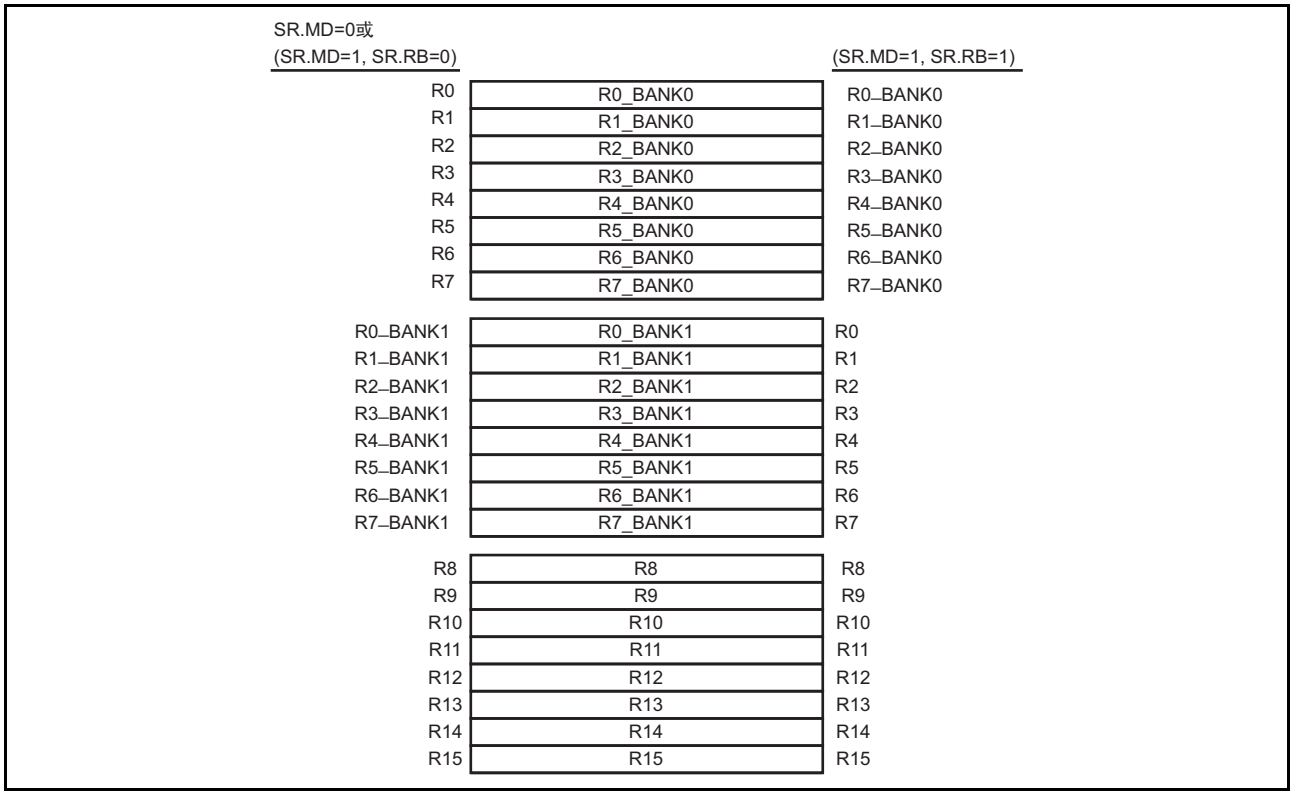


图 2.3 通用寄存器

【编程中的注意事项】

因为用户模式的R0～R7分配至R0_BANK0～R7_BANK0，异常/中断后的R0～R7分配至R0_BANK1～R7_BANK1，所以中断处理程序无需保存或恢复用户模式的R0～R7 （R0_BANK0～R7_BANK0）。

2.2.3 浮点寄存器

浮点寄存器如图 2.4 所示。具有 32 个 32 位浮点寄存器。这些寄存器由 2 种存储体构成，即：FPR0_BANK0 ～ FPR15_BANK0、 FPR0_BANK1 ～ FPR15_BANK1。将这 32 个寄存器作为 FR0 ～ FR15、DR0/2/4/6/8/l0/12/14、 FV0/4/8/12、 XF0 ～ XF15、 XD0/2/4/6/8/l0/12/14、 XMTRX 进行参考。通过 FPSCR 的 FR 位确定 FPRn_BANKi 与参考名称的对应。请参考图 2.4。

(1) 浮点寄存器 FPRn_BANKi（32 个寄存器）

FPR0_BANK0、 FPR l_BANK0、 FPR2_BANK0、 FPR3_BANK0、
FPR4_BANK0、 FPR5_BANK0、 FPR6_BANK0、 FPR7_BANK0、
FPR8_BANK0、 FPR9_BANK0、 FPR10_BANK0、 FPR11_BANK0、
FPR12_BANK0、 FPR13_BANK0、 FPR14_BANK0、 FPR15_BANK0
FPR0_BANK1、 FPR1_BANK1、 FPR2_BANK1、 FPR3_BANK1、
FPR4_BANK1、 FPR5_BANK1、 FPR6_BANK1、 FPR7_BANK1、
FPR8_BANK1、 FPR9_BANK1、 FPR10_BANK1、 FPR11_BANK1、
FPR12_BANK1、 FPR13_BANK1、 FPR14_BANK1、 FPR15_BANK1

(2) 单精度浮点寄存器 FRi（16 个寄存器）

FPSCR.FR=0 时，将 FR0 ～ FR15 分配至 FPR0_BANK0 ～ FPR15_BANK0。
FPSCR.FR=1 时，将 FR0 ～ FR15 分配至 FPR0_BANK1 ～ FPR15_BANK1。

(3) 双精度浮点寄存器或单精度浮点寄存器对 DRi（8 个寄存器）

DR 寄存器由 2 个 FR 寄存器构成。
DR0={FR0、 FR1}、 DR2={FR2、 FR3}、
DR4={FR4、 FR5}、 DR6={FR6、 FR7}、
DR8={FR8、 FR9}、 DR10={FR10、 FR11}、
DR12={FR12、 FR13}、 DR14={FR14、 FR15}

(4) 单精度浮点向量寄存器 FVi（4 个寄存器）

FV 寄存器由 4 个 FR 寄存器构成。
FV0={FR0、 FR1、 FR2、 FR3},
FV4={FR4、 FR5、 FR6、 FR7},
FV8={FR8、 FR9、 FR10、 FR11},
FVl2={FR12、 FR13、 FR14、 FR15}

(5) 单精度浮点扩展寄存器 XFi（16 个寄存器）

FPSCR.FR=0 时，将 XF0 ～ XF15 分配至 FPR0_BANK1 ～ FPR15_BANK1。
FPSCR.FR=1 时，将 XF0 ～ XF15 分配至 FPR0_BANK0 ～ FPR15_BANK0。

(6) 单精度浮点扩展寄存器对 XDi（8 个寄存器）

XD 寄存器由 2 个 XF 寄存器构成。
XD0={XF0、 XF1}、 XD2={XF2、 XF3}、
XD4={XF4、 XF5}、 XD6={XF6、 XF7}、
XD8={XF8、 XF9}、 XD10={XF10、 XF11}、
XD12={XF12、 XF13}、 XD14={XF14、 XF15}

(7) 单精度浮点扩展寄存器矩阵 XMTRX

XMTRX 由 16 个 XF 寄存器构成。

$$XMTRX=\begin{pmatrix} XF0 & XF4 & XF8 & XF12 \\ XF1 & XF5 & XF9 & XF13 \\ XF2 & XF6 & XF10 & XF14 \\ XF3 & XF7 & XF11 & XF15 \end{pmatrix}$$

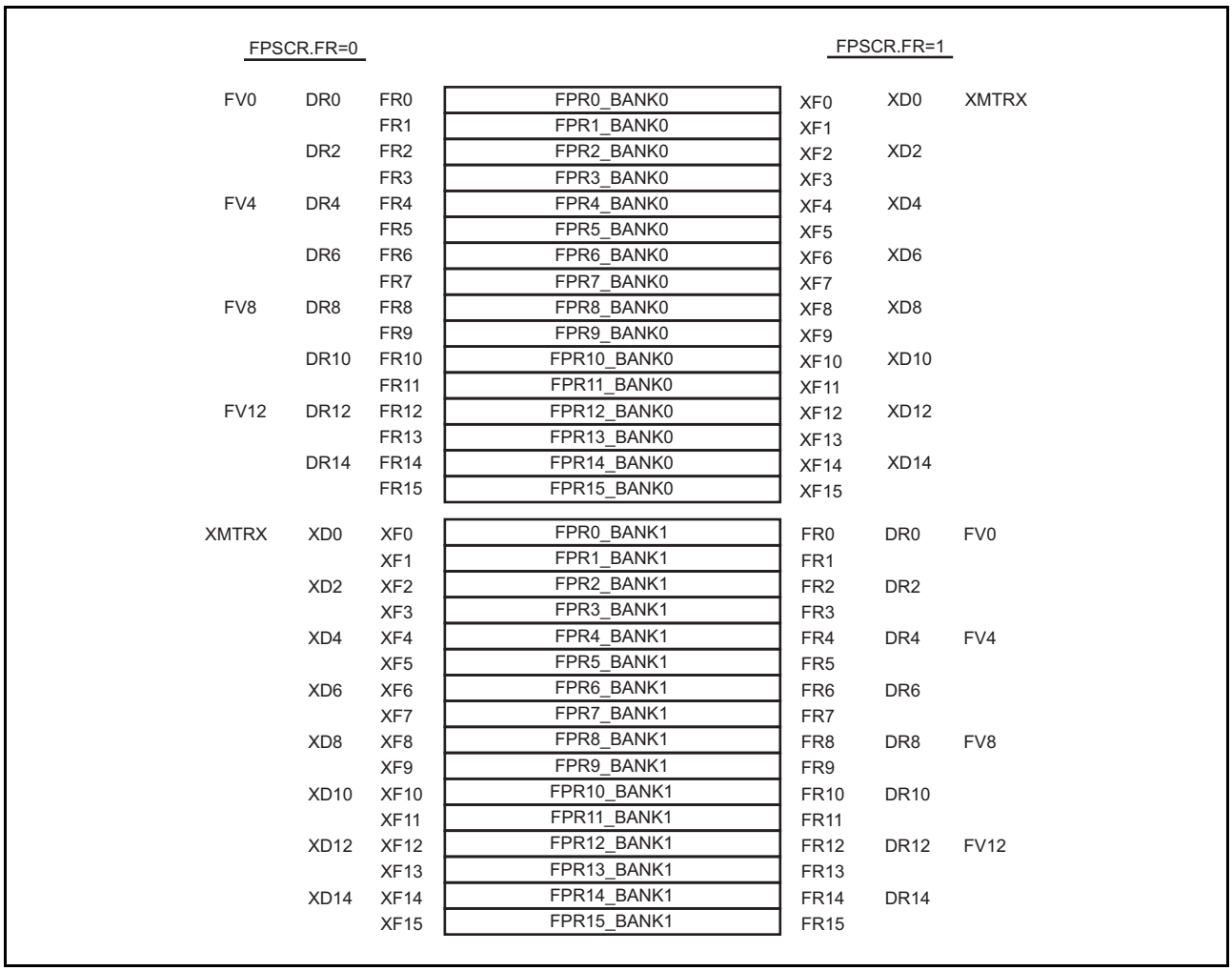
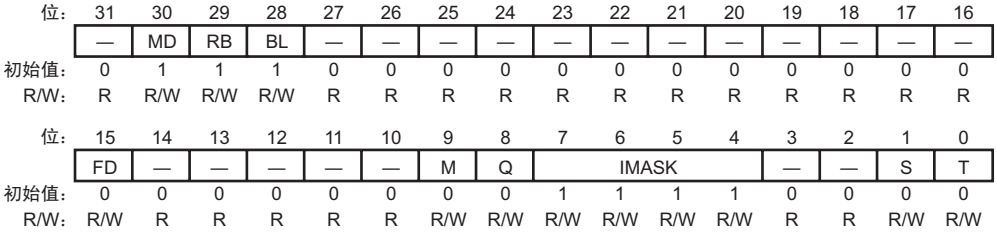


图 2.4 浮点寄存器

2.2.4 控制寄存器

(1) 状态寄存器 （SR）



位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	MD	RB	BL	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R/W	R/W	R/W	R	R	R	R	R	R	R	R	R	R	R	R

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	FD	—	—	—	—	—	M	Q	IMASK				—	—	S	T
初始值：	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
R/W：	R/W	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R/W	R/W

位	位名称	初始值	R/W	说明
31	—	0	R	保留位 关于此位的读取 / 写入请参考 “产品使用时的注意事项”。
30	MD	1	R/W	处理模式 选择处理模式。 0：用户模式 （指令中存在不可执行的指令。另外，资源中存在不可存取的资源。） 1：特权模式 通过异常或中断该位置 1。
29	RB	1	R/W	特权模式下的通用寄存器存储体指定位 0：R0_BANK0 ～ R7_BANK0 可作为通用寄存器 R0 ～ R7 进行存取， R0_BANK1 ～ R7_BANK1 可通过 LDC/STC 指令进行存取。 1：R0_BANK1 ～ R7_BANK1 可作为通用寄存器 R0 ～ R7 进行存取， R0_BANK0 ～ R7_BANK0 可通过 LDC/STC 指令进行存取。 通过异常或中断该位置 1。
28	BL	1	R/W	异常 / 中断阻塞位 该位为 1 时，屏蔽中断请求。发生用户中断以外的普通异常时，处理器转移至复位状态。 通过异常或中断该位置 1。
27 ～ 16	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
15	FD	0	R/W	FPU 禁止位 该位为 1 时， FPU 指令产生普通 FPU 禁止异常， FPU 指令在延迟槽时发生槽 FPU 禁止异常 （FPU 指令：H'F*** 令、对于 FPUL/FPSCR 的 LDS(.L)/STS(.L) 指令）。
14 ～ 10	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。

位	位名称	初始值	R/W	说明
9	M	0	R/W	M 位 DIV0S、 DIV0U、 DIV1 指令时使用此位。
8	Q	0	R/W	Q 位 DIV0S、 DIV0U、 DIV1 指令时使用此位。
7 ～ 4	IMASK	均为 1	R/W	中断屏蔽级 屏蔽 IMASK 以下级别的中断。另外，产生中断后，可使用 CPU 运行模式寄存器 （CPUOPM）切换 IMASK 是否变为中断接受级别。 CPUOPM 的运行请参考 “附录 A. CPU 运行模式寄存器（CPUOPM）”。
3、 2	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
1	S	0	R/W	S 位 指定 MAC 指令的饱和运行。
0	T	0	R/W	T 位 表示真 / 伪条件、进位、借位、上溢或下溢等。 详细内容请参考 “第 3 章 指令系统”。

(2) 保存状态寄存器 （SSR）（32 位、特权保护、初始值 = 不定）

发生异常或中断时将 SR 的内容保存至 SSR。

(3) 保存程序计数器 （SPC）（32 位、特权保护、初始值 = 不定）

将发生异常或中断的指令地址保存至 SPC。

(4) 全局基址寄存器 （GBR）（32 位、初始值 = 不定）

将 GBR 作为 @(disp,GBR)、 @(R0,GBR) 寻址的基址被参考。

(5) 向量基址寄存器 （VBR）（32 位、特权保护、初始值 =H'0000 0000）

发生异常及中断时将 VBR 作为转移目标基址被参考。详细内容请参考 “第 5 章 异常处理”。

(6) 保存通用寄存器 15 （SGR）（32 位、特权保护、初始值 = 不定）

发生异常或中断时将 R15 的内容保存至 SGR。

(7) 调试基址寄存器 （DBR）（32 位、特权保护、初始值 = 不定）

将用户中断调试功能设定为有效时 （CBCR.UBDE=1）、 DBR 代替 VBR 作为用户中断处理程序的转移目标地址被参考。

2.2.5 系统寄存器

(1) 乘加高位寄存器 （MACH）（32 位、初始值 = 不定）、
乘加低位寄存器 （MACL）（32 位、初始值 = 不定）

将 MACH/MACL 作为 MAC 指令的加法值使用。另外，也用于保存 MAC 指令、 MUL 指令的运算结果。

(2) 过程寄存器 （PR）（32 位、初始值 = 不定）

使用 BSR、BSRF、JSR 指令调用子程序时的返回地址保存至 PR。通过子程序返回指令 （RTS）参考 PR。

(3) 程序计数器 （PC）（32 位、初始值 =H'A000 0000）

PC 表示执行中的指令地址。

(4) 浮点状态 / 控制寄存器 （FPSCR）

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	FR	SZ	PR	DN	Cause	
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Cause				Enable（EN）					Flag					RM	
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 22	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
21	FR	0	R/W	浮点寄存器组 0：将 FPR0_BANK0 ～ FPR15_BANK0 分配至 FR0 ～ FR15，将 FPR0_BANK1 ～ FPR15_BANK1 分配至 XF0 ～ XF15。 1：将 FPR0_BANK0 ～ FPR15_BANK0 分配至 XF0 ～ XF15，将 FPR0_BANK1 ～ FPR15 ～ BANK1 分配至 FR0 ～ FR15。
20	SZ	0	R/W	传送长度模式 0：FMOV 指令的数据长度为 32 位。 1：FMOV 指令的数据长度为 32 位对或 64 位。 SZ 位、 PR 位与字节序的关系请参考图 2.5。
19	PR	0	R/W	精度模式 0：将浮点指令作为单精度运算执行。 1：将浮点指令作为双精度运算执行 （图形支持指令未定义）。 PR 位、 SZ 位与字节序的关系请参考图 2.5。
18	DN	1	R/W	非规格化模式 0：将非规格化数作为非规格化数处理。 1：将非规格化数作为 0 处理。

位	位名称	初始值	R/W	说明
17 ～ 12	Cause	均为 0	R/W	FPU 异常源字段 FPU 异常允许字段 FPU 异常标志字段 执行 FPU 运算指令时， FPU 异常源字段最初设定为 0。之后发生 FPU 异常时，将 FPU 异常源字段与 FPU 异常标志字段的相应位置 1。 FPU 异常标志字段保持最后清除 FPU 异常标志字段后所发生的异常状态。 有关各字段位的分配请参考表 2.2。
11 ～ 7	Enabl（EN）	均为 0	R/W	
6 ～ 2	Flag	均为 0	R/W	
1、 0	RM	01	R/W	舍入模式 选择舍入方法。 00：向接近方向舍入 01：向 0 方向舍入 10：保留 （禁止设定） 11：保留 （禁止设定）

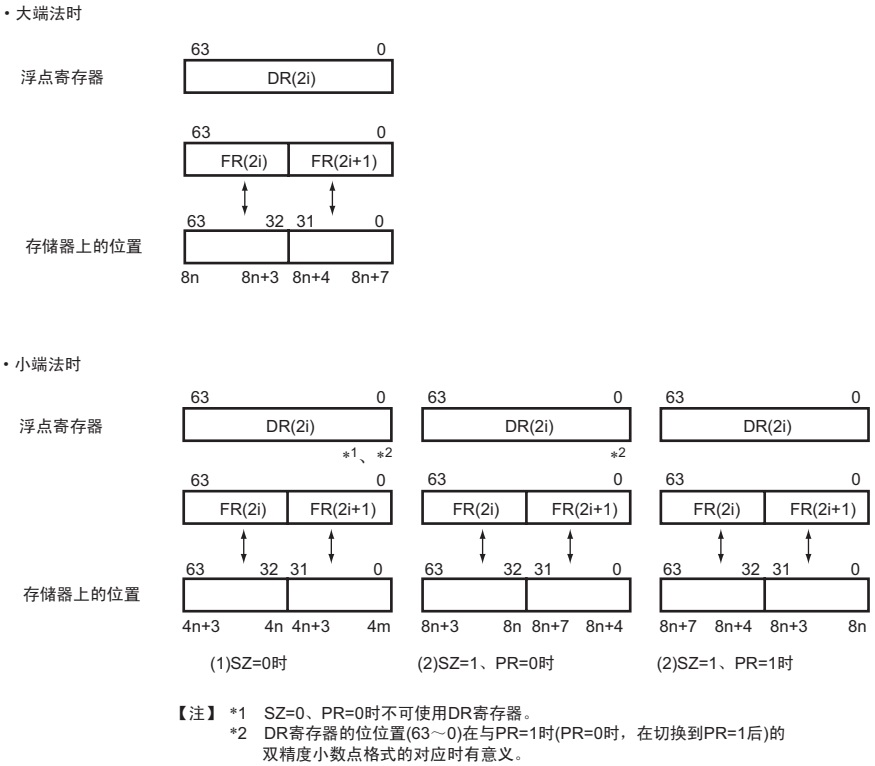


图 2.5 SZ 位与字节序的关系

表 2.2 FPU 异常处理相关位的分配

		FPU 错误（E）	无效运算（V）	0 除法（Z）	上溢（O）	下溢（U）	不正确（I）
Cause	FPU 异常源字段	bit17	bit16	bit15	bit14	bit13	bit12
Enable	FPU 异常允许字段	无	bit11	bit10	bit9	bit8	bit7
Flag	FPU 异常标志字段	无	bit6	bit5	bit4	bit3	bit2

(5) 浮点通信寄存器 （FPUL）（32 位、初始值 = 不定）

通过 FPUL 进行 FPU 寄存器与 CPU 寄存器间的数据传送。

2.3 存储器分配寄存器

某些控制寄存器被映射到以下的存储区域。分配到这些存储区域的寄存器有 2 个地址。

H'1C00 0000 ～ H'1FFF FFFF

H'FC00 0000 ～ H'FFFF FFFF

以上 2 个区域的使用如下。

- H'1C00 0000～H'1FFF FFFF
  必须使用MMU地址转换功能存取该区域。通过将该区域的页码设定到TLB的相应字段可对存储器分配寄存器进行存取。不使用MMU地址转换功能存取该区域时，不保证运行。
- H'FC00 0000～H'FFFF FFFF
  在用户模式下，如对区域H'FC00 0000～H'FFFF FFFF进行存取则发生地址错误。用户模式下，在地址转换的存取过程中可参考存储器分配寄存器。

【注】 请勿存取 2 个区域中未分配寄存器的地址。对未分配寄存器的地址进行存取时运行不稳定。另外，必须以固定的数据长度对存储器分配寄存器进行存取。以不正确的长度进行存取时运行也不稳定。

2.4 寄存器的数据格式

寄存器操作数的数据长度通常为长字 （32 位）。向寄存器加载存储器中的数据时，如存储器操作数的数据长度为字节 （8 位）或字 （16 位），则将其扩展 （符号扩展）为长字并保存至寄存器。

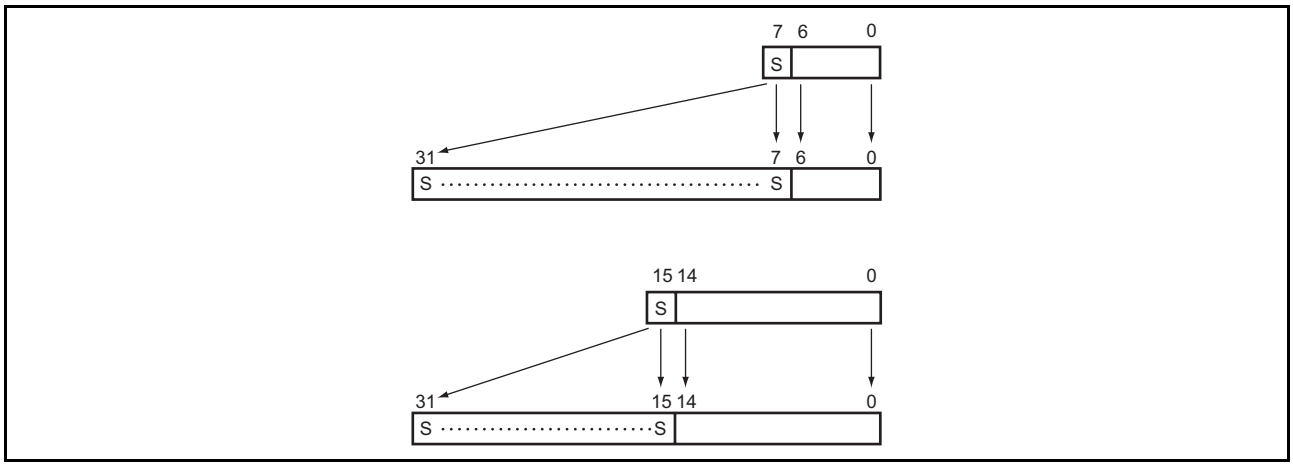


图 2.6 字节数据、字数据在寄存器中的数据格式

2.5 存储器中的数据格式

存储器中的数据格式有字节、字、长字。可通过 8 位字节、 16 位字、 32 位长字格式对存储器进行存取。未满 32 位的存储器操作数，进行符号扩展后保存至寄存器。

字操作数请从字边界 （2 字节单位的偶数地址：地址 2n）进行存取，长字操作数请从长字边界 （4 字节单位的偶数地址：地址 4n）进行存取。假如不遵守此规则，将会导致地址错误。从任意地址均可存取字节操作数。

数据格式可选择大端法或小端法字节顺序。上电复位时请用外部引脚设定字节序。不可动态地变更字节序。但是，位的位置通常是按从最高位 （most-significant）向最低位 （least-significant）、从左向右减少的顺序编号。即：在 32 位的长字中，最左边的位、 bit31 为最高位，最右边的位、 bit0 为最低位。

存储器中的数据格式如图 2.7 所示。

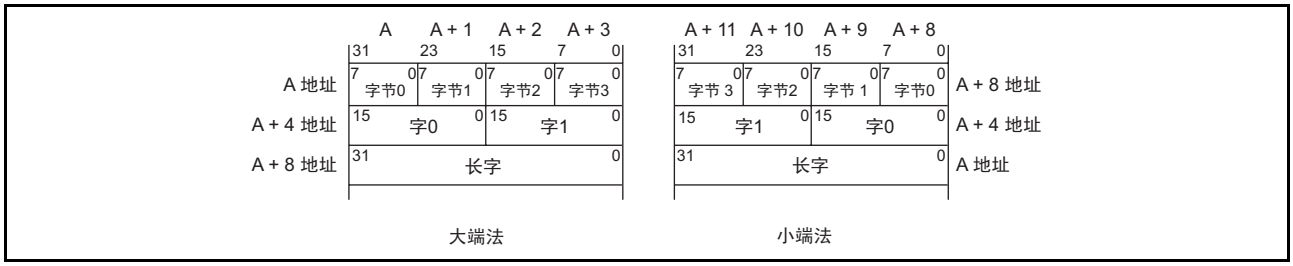


图 2.7 存储器中的数据格式

关于 64 位的数据格式请参考图 2.5。

2.6 处理状态

处理状态大体分为复位状态、指令执行状态、低功耗状态这 3 种。

(1) 复位状态

复位状态即复位 CPU 的状态。复位状态分为上电复位状态与手动复位状态。

上电复位状态下，初始化 CPU 的内部状态与内部外围模块的寄存器。手动复位状态下，初始化部分内部外围模块的寄存器与 CPU 的内部状态。详细内容请参考硬件手册各章的寄存器结构。

(2) 指令执行状态

指令执行状态即 CPU 依次执行程序的状态。指令执行状态分为普通程序执行状态与异常处理状态。

(3) 低功耗状态

低功耗状态即停止 CPU 运行降低功耗的状态。通过睡眠指令变为低功耗状态。低功耗状态有两种模式即：睡眠模式、待机模式。低功耗状态的详细内容请参考该产品硬件手册的 “低功耗模式”。

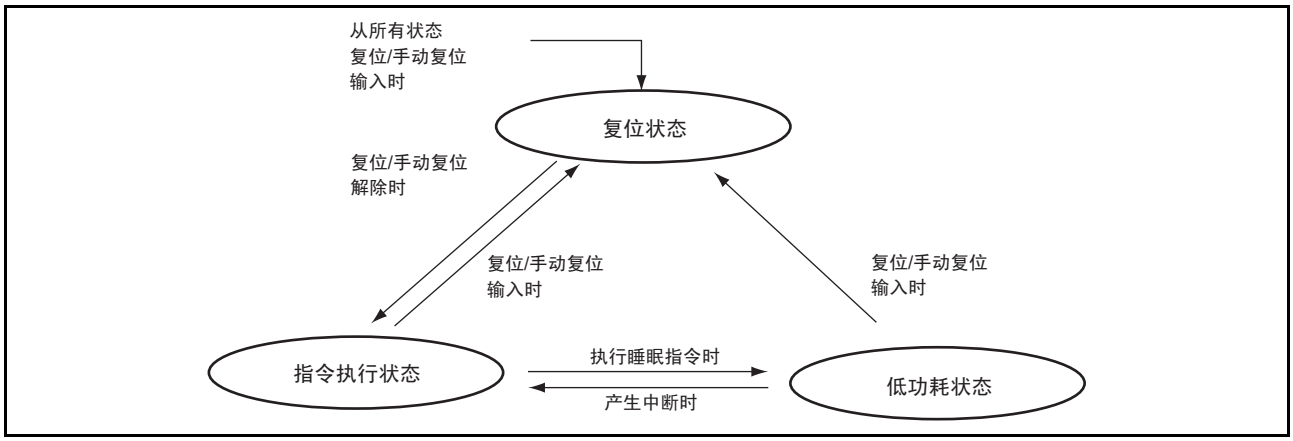


图 2.8 处理状态转移图

2.7 使用时的注意事项

2.7.1 自我改写码注意事项

SH-4A 为了实现高速处理而进行指令的预先读取。因此改写存储器中的指令列后，立即准备执行该指令时，有可能执行保存于预先读取缓冲器的更新前的指令。另外，为装载指令 / 操作数分离式的高速缓存，也需注意一致性。为了确切地反映变更，请在进行改写的指令与执行被改写的指令之间执行以下的指令列。

(1) 改写的指令存在于不可进行高速缓存操作的区域时

```
SYNCO
ICBI @Rn
```

通过 ICBI 指令的 Rn 指定的地址可为无地址错误范围内的任意地址。

(2) 改写的指令列存在于可进行高速缓存操作 （直写）的区域时

```
SYNCO
ICBI @Rn
```

通过 ICBI 指令使对应改写后指令列的指令高速缓存区域全部无效。ICBI 以线为单位执行。1 条线为 32 个字节。

(3) 改写的指令列存在于可进行高速缓存操作 （备份）的区域时

```
OCBP @Rm 或 OCBWB @Rm
SYNCO
ICBI @Rn
```

通过 OCBP 指令或 OCBWB 指令，将对应改写后指令列的操作数高速缓存区域全部回写至主存储器，此后请通过 ICBI 指令使对应的指令高速缓存区域无效。 ICBI/OCBP/OCBWB 以线为单位执行。 1 条线为 32 个字节。

第 3 章 指令系统

以固定长度 16 位指令实现 SH-4A 的指令系统。 SH-4A 可使用字节 （8 位）、字 （16 位）、长字 （32 位）、四字 （64 位）的数据长度对存储器进行存取。单精度浮点数据 （32 位）可用长字或四字长度与存储器进行互换。双精度浮点数据 （64 位）可用四字长度与存储器进行互换。 SH-4A 将字节长度及字长度的数据从存储器转移至寄存器时，将数据进行符号扩展。

3.1 执行环境

(1) PC

PC 表示指令自身的指令地址。

(2) 加载 / 存储体系结构

SH-4A 将通过寄存器执行基本运算的加载 / 存储体系结构作为特点。除了在存储器直接执行的逻辑 AND 运算那样的位操作运算之外，将需进行存储器存取的运算加载至寄存器后，在寄存器执行。

(3) 延迟转移

除 BF、 BT 这 2 个转移指令之外， SH-4A 的转移指令及 RTE 均为延迟转移。延迟转移中，转移指令的下一个指令在转移目标指令之前执行。

(4) 延迟槽

将延迟转移后的此执行槽称为 “延迟槽”。例如：BRA 执行顺序如下。

表 3.1 延迟转移指令的执行顺序

指令列				执行顺序
	BRA	TARGET	（延迟转移指令）	BRA
	ADD		（延迟槽）	↓
	:			ADD
	:			↓
TARGET	target-inst		（转移目标指令）	target-inst

有些指令在延迟槽执行时可能发生槽非法指令异常。详细内容请参考 **“第 5 章 异常处理”**。转移未成立的 BF/S、 BT/S 的下一个指令也是延迟槽指令。

(5) T 位

状态寄存器 （SR）的 T 位用于表示比较运算的结果等，通过带条件的转移指令参考此位。例如：带条件的转移指令例如下所示：

```
ADD       #1, R0     ; 通过 ADD 运算 T 位未变更。
CMP/EQ    R1, R0     ; R0=R1 时 T 位置 1。
BT        TARGET     ; T 位 =1 （R0=R1）时，转移至 TARGET。
```

在 RTE 延迟槽中、如下参考状态寄存器 （SR）位。指令存取中，在变更前使用 MD 位，数据存取中，存取变更后的 MD 位。为执行延迟槽指令而使用变更后的其它 S 、T、M、Q、FD、BL、RB 位。STC、STC.L SR 指令对变更后的所有 SR 位进行存取。

(6) 常数值

可通过操作码、立即值指定 8 位常数值。另外可在存储器中定义 16 位、 32 位常数值，可通过 PC 相对加载指令对其进行参考。

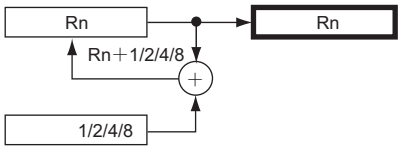
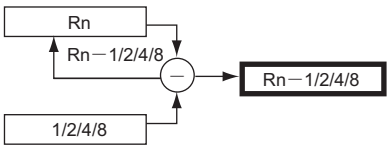
```
MOV.W     @(disp, PC), Rn
MOV.L     @(disp, PC), Rn
```

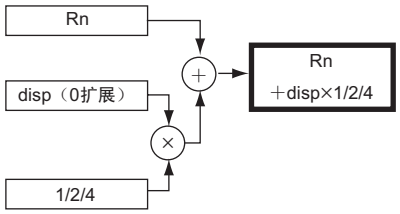
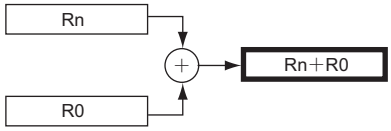
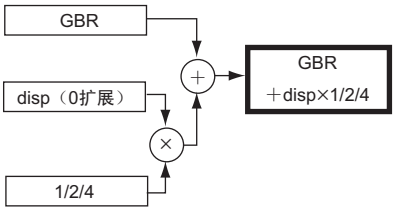
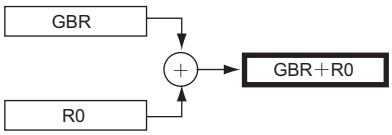
无对于浮点的 PC 相对加载指令。但是对于单精度浮点寄存器，可通过使用 FLDI0、 FLDI1 指令设定为 0.0 或 1.0。

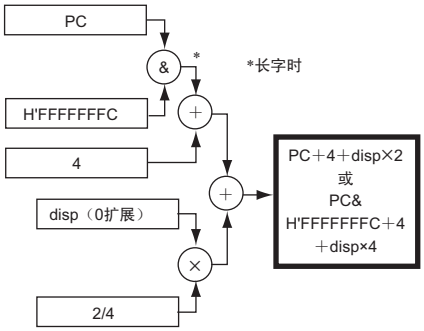
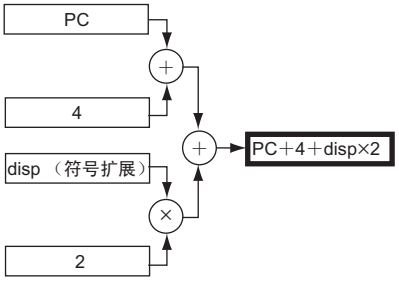
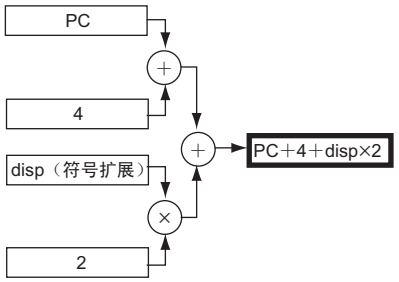
3.2 寻址方式

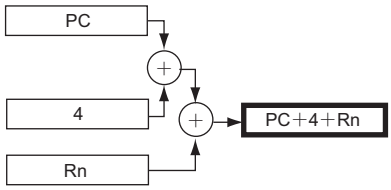
寻址方式与有效地址的计算如**表 3.2** 所示。存取虚拟地址空间的某些位置时 （MMUCR.AT=1）、有效地址转换为物理地址。选择多个虚拟存储空间系统时 （MMUCR.SV=0），也将 PTEH 的最低位作为存取 ASID 进行参考。请参考 “**第 7 章 存储器管理单元** （MMU）”。

表 3.2 寻址方式与有效地址

寻址方式	指令格式	有效地址的计算方法	计算公式
寄存器直接寻址	Rn	有效地址为寄存器 Rn。 （操作数为寄存器 Rn 的内容。）	—
寄存器间接寻址	@Rn	有效地址为寄存器 Rn 的内容。 	Rn→EA （EA：有效地址）
带后增量的寄存器间接寻址	@Rn+	有效地址为寄存器 Rn 的内容。指令执行后给 Rn 加上常数。操作数长度为字节时常数为 1、为字时常数为 2、为长字时常数为 4、为四字时常数为 8。 	Rn→EA 指令执行后 字节：Rn+1→Rn 字：Rn+2→Rn 长字：Rn+4→Rn 四字：Rn+8→Rn
带先减量的寄存器间接寻址	@– Rn	有效地址为预先减去常数的寄存器 Rn 的内容。操作数长度为字节时常数为 1、为字时常数为 2、为长字时常数为 4、为四字时常数为 8。 	字节：Rn–1→Rn 字：Rn–2→Rn 长字：Rn–4→Rn 四字：Rn–8→Rn Rn→EA （用计算后的 Rn 执行指令）

寻址方式	指令格式	有效地址的计算方法	计算公式
带偏移量的寄存器间接寻址	@(disp:4, Rn)	有效地址为寄存器 Rn 加上 4 位偏移量 disp 的内容。 Disp 进行零扩展后，根据操作数长度增加，操作数长度是字节时为 1 倍、是字时为 2 倍、是长字时为 4 倍。 	字节：Rn+disp→EA 字：Rn+disp×2→EA 长字：Rn+disp×4→EA
带变址的寄存器间接寻址	@(R0, Rn)	有效地址为寄存器 Rn 加上 R0 的内容。 	Rn+R0→EA
带偏移量的 GBR 间接寻址	@(disp:8, GBR)	有效地址为寄存器 GBR 加上 8 位偏移量 disp 的内容。 Disp 进行零扩展后，根据操作数长度增加。操作数长度是字节时为 1 倍、是字时为 2 倍、是长字时为 4 倍。 	字节：GBR+disp→EA 字：GBR+disp×2→EA 长字：GBR+disp×4→EA
带变址的 GBR 间接寻址	@(R0, GBR)	有效地址为寄存器 GBR 加上 R0 的内容。 	GBR+R0→EA

寻址方式	指令格式	有效地址的计算方法	计算公式
带偏移量的 PC 相对寻址	@(disp:8, PC)	有效地址为 PC+4 加上 8 位偏移量 disp 的内容。 Disp 进行零扩展后，根据操作数长度增加。操作数长度是字时为 2 倍、是长字时为 4 倍。并且为长字时屏蔽 PC 的低 2 位。 	字：PC+4+disp×2→EA 长字： PC & H'FFFFFFFC+4+disp×→EA
PC 相对寻址	disp:8	有效地址为将 8 位偏移量 disp 进行有符号扩展后乘以 2 再加上 PC+4 的内容。 	PC+4+disp×2→Branch-Target
	disp:12	有效地址为将 12 位偏移量 disp 进行有符号扩展后乘以 2 再加上 PC+4 的内容。 	PC+4+disp×2→Branch-Target

寻址方式	指令格式	有效地址的计算方法	计算公式
PC 相对寻址	Rn	有效地址为 PC+4 加上 Rn 的内容。 	PC+4+Rn→Branch-Target
立即值	#imm:8	将 TST、 AND、 OR、XOR 指令的 8 位立即数 imm 零扩展。	—
	#imm:8	将 MOV、 ADD、 CMP/EQ 指令的 8 位立即数 imm 符号扩展。	—
	#imm:8	将 TRAPA 指令的 8 位立即数据 imm 零扩展后再乘以 4。	—

【注】 下列带偏移量 （disp）的寻址方式中，为明确 LSI 的运行，本操作手册的汇编程序记述中所写入的是执行定标（此定标取决于操作数长度。 ×1、 ×2、 ×4）前的值。实际的汇编程序记述请参考各汇编程序的记述规则。

@ （disp:4, Rn） ;带偏移量的寄存器间接寻址

@ （disp:8, GBR） ;带偏移量的GBR间接寻址

@ （disp:8, PC） ;带偏移量的PC相对寻址

disp: 8, disp:12 ;PC相对寻址

3.3 指令系统

表 3.4 ～表 3.13 所示的用于 SuperH 指令说明的符号如表 3.3 所示。

表 3.3 指令列表的记述

项目	格式	说明
指令助记符	OP.Sz SRC,DEST	OP：操作码 Sz：长度 SRC：源操作数 DEST：源及 / 或目标操作数 Rm：源寄存器 Rn：目标寄存器 imm：立即数 disp：偏移量
运算的要点		→、←：传送方向 (xx)：存储器操作数 M/Q/T：SR 的标志位 &：各位的逻辑积 |：各位的逻辑和 ^：各位的逻辑异 ~：各位的逻辑非 <<n,>>n：n 位移位
操作码	MSB←→LSB	mmmm：寄存器号码 （Rm、 FRm） nnnn：寄存器号码 （Rn、 FRn） 0000： R0, FR0 0001： R1, FR1 ： 1111： R15, FR15 mmm：寄存器号码 （DRm、 XDm、 Rm_BANK） nnn： 寄存器号码 （DRn、 XDn、 Rn_BANK） 000： DR0、 XD0、 R0_BANK 001： DR2、 XD2、 R1_BANK ： 111： DR14、 XD14、 R7_BANK mm：寄存器号码 (FVm) nn：寄存器号码 (FVn) 00： FV0 01： FV4 10： FV8 11： FV12 iiii：立即值 dddd：偏移量
特权模式	—	记述为 “特权”时，仅可在特权模式下执行。
T 位	指令执行后的 T 位值	—：无变更
新指令	—	记述为 “新指令”时，是 SH-4A 中新追加的指令。

【注】 根据指令操作数的长度执行定标 （×1、 ×2、 ×4、 ×8）。

表 3.4 定点传送指令

指令		运行	操作码	特权	T 位	新指令
MOV	#imm,Rn	imm→ 符号扩展 →Rn	1110nnnniiiiiiii	—	—	—
MOV.W	@(disp*,PC),Rn	(disp×2+PC+4)→ 符号扩展 →Rn	1001nnnndddddddd	—	—	—
MOV.L	@(disp*,PC),Rn	(disp×4+PC&H'FFFFFFFC+4)→Rn	1101nnnndddddddd	—	—	—
MOV	Rm,Rn	Rm→Rn	0110nnnnmmmm0011	—	—	—
MOV.B	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0000	—	—	—
MOV.W	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0001	—	—	—
MOV.L	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0010	—	—	—
MOV.B	@Rm,Rn	(Rm)→ 符号扩展 →Rn	0110nnnnmmmm0000	—	—	—
MOV.W	@Rm,Rn	(Rm)→ 符号扩展 →Rn	0110nnnnmmmm0001	—	—	—
MOV.L	@Rm,Rn	(Rm)→Rn	0110nnnnmmmm0010	—	—	—
MOV.B	Rm,@-Rn	Rn-1→Rn, Rm→(Rn)	0010nnnnmmmm0100	—	—	—
MOV.W	Rm,@-Rn	Rn-2→Rn, Rm→(Rn)	0010nnnnmmmm0101	—	—	—
MOV.L	Rm,@-Rn	Rn-4→Rn, Rm→(Rn)	0010nnnnmmmm0110	—	—	—
MOV.B	@Rm+,Rn	(Rm)→ 符号扩展 →Rn, Rm+1→Rm	0110nnnnmmmm0100	—	—	—
MOV.W	@Rm+,Rn	(Rm)→ 符号扩展 →Rn, Rm+2→Rm	0110nnnnmmmm0101	—	—	—
MOV.L	@Rm+,Rn	(Rm)→Rn, Rm+4→Rm	0110nnnnmmmm0110	—	—	—
MOV.B	R0,@(disp*,Rn)	R0→(disp+Rn)	10000000nnnndddd	—	—	—
MOV.W	R0,@(disp*,Rn)	R0→(disp×2+Rn)	10000001nnnndddd	—	—	—
MOV.L	Rm,@(disp*,Rn)	Rm→(disp×4+Rn)	0001nnnnmmmmdddd	—	—	—
MOV.B	@(disp*,Rm),R0	(disp+Rm)→ 符号扩展 →R0	10000100mmmmdddd	—	—	—
MOV.W	@(disp*,Rm),R0	(disp×2+Rm)→ 符号扩展 →R0	10000101mmmmdddd	—	—	—
MOV.L	@(disp*,Rm),Rn	(disp×4+Rm)→Rn	0101nnnnmmmmdddd	—	—	—
MOV.B	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0100	—	—	—
MOV.W	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0101	—	—	—
MOV.L	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0110	—	—	—
MOV.B	@(R0,Rm),Rn	(R0+Rm)→ 符号扩展 →Rn	0000nnnnmmmm1100	—	—	—
MOV.W	@(R0,Rm),Rn	(R0+Rm)→ 符号扩展 →Rn	0000nnnnmmmm1101	—	—	—
MOV.L	@(R0,Rm),Rn	(R0+Rm)→Rn	0000nnnnmmmm1110	—	—	—
MOV.B	R0,@(disp*,GBR)	R0→(disp+GBR)	11000000dddddddd	—	—	—
MOV.W	R0,@(disp*,GBR)	R0→(disp×2+GBR)	11000001dddddddd	—	—	—
MOV.L	R0,@(disp*,GBR)	R0→(disp×4+GBR)	11000010dddddddd	—	—	—
MOV.B	@(disp*,GBR),R0	(disp+GBR)→ 符号扩展 →R0	11000100dddddddd	—	—	—
MOV.W	@(disp*,GBR),R0	(disp×2+GBR)→ 符号扩展 →R0	11000101dddddddd	—	—	—
MOV.L	@(disp*,GBR),R0	(disp×4+GBR)→R0	11000110dddddddd	—	—	—
MOVA	@(disp*,PC),R0	disp×4+PC&H'FFFFFFFC+4→R0	11000111dddddddd	—	—	—
MOVCO.L	R0,@Rn	LDST→T if(T==1)R0→(Rn) 0→LDST	0000nnnn01110011	—	LDST	新指令
MOVLI.L	@Rm,R0	1→LDST (Rm)→R0 但是，发生中断 / 异常时 0→LDST	0000mmmm01100011	—	—	新指令

指令	运行	操作码	特权	T 位	新指令
MOVUA.L @Rm,R0	(Rm)→R0 非边界调整数据的加载	0100mmmm10101001	—	—	新指令
MOVUA.L @Rm+,R0	(Rm)→R0,Rm+4→Rm 非边界调整数据的加载	0100mmmm11101001	—	—	新指令
MOVT Rn	T→Rn	0000nnnn00101001	—	—	—
SWAP.B Rm,Rn	Rm→ 低位 2 字节的 上下字节交换 →Rn	0110nnnnmmmm1000	—	—	—
SWAP.W Rm,Rn	Rm→ 上下字交换 →Rn	0110nnnnmmmm1001	—	—	—
XTRCT Rm,Rn	Rm:Rn 的中间 32 位 →Rn	0010nnnnmmmm1101	—	—	—

【注】 * 在瑞萨的汇编程序中，将定标后 （×1、 ×2、 ×4）的值设定为 disp。

表 3.5 算数运算指令

指令	运行	操作码	特权	T 位	新指令
ADD Rm,Rn	Rn+Rm→Rn	0011nnnnmmmm1100	—	—	—
ADD #imm,Rn	Rn+imm→Rn	0111nnnniiiiiiii	—	—	—
ADDC Rm,Rn	Rn+Rm+T→Rn, 进位 →T	0011nnnnmmmm1110	—	进位	—
ADDV Rm,Rn	Rn+Rm→Rn, 上溢 →T	0011nnnnmmmm1111	—	上溢	—
CMP/EQ #imm,R0	R0=imm 时 1→T 除此之外时 0→T	10001000iiiiiiii	—	比较结果	—
CMP/EQ Rm,Rn	Rn=Rm 时 1→T 除此之外时 0→T	0011nnnnmmmm0000	—	比较结果	—
CMP/HS Rm,Rn	无符号 Rn ≥ Rm 时 1→T 除此以外时 0→T	0011nnnnmmmm0010	—	比较结果	—
CMP/GE Rm,Rn	有符号 Rn ≥ Rm 时 1→T 除此以外时 0→T	0011nnnnmmmm0011	—	比较结果	—
CMP/HI Rm,Rn	无符号 Rn > Rm 时 1→T 除此以外时 0→T	0011nnnnmmmm0110	—	比较结果	—
CMP/GT Rm,Rn	有符号 Rn > Rm 时 1→T 除此以外时 0→T	0011nnnnmmmm0111	—	比较结果	—
CMP/PZ Rn	Rn ≥ 0 时 1→T 除此以外时 0→T	0100nnnn00010001	—	比较结果	—
CMP/PL Rn	Rn > 0 时 1→T 除此以外时 0→T	0100nnnn00010101	—	比较结果	—
CMP/STR Rm,Rn	任意字节相等时 1→T 除外以外时 0→T	0010nnnnmmmm1100	—	比较结果	—
DIV1 Rm,Rn	1 步进式除法 (Rn÷Rm)	0011nnnnmmmm0100	—	计算结果	—
DIV0S Rm,Rn	Rn 的 MSB→Q, Rm 的 MSB→M, M^Q→T	0010nnnnmmmm0111	—	计算结果	—
DIV0U	0→M/Q/T	0000000000011001	—	0	—
DMULS.L Rm,Rn	有符号 Rn×Rm→MAC, 32×32→64 位	0011nnnnmmmm1101	—	—	—
DMULU.L Rm,Rn	无符号 Rn×Rm→MAC, 32×32→64 位	0011nnnnmmmm0101	—	—	—
DT Rn	Rn-1→Rn, Rn 为 0 时 1→T Rn 为 0 以外时 0→T	0100nnnn00010000	—	比较结果	—

指令		运行	操作码	特权	T 位	新指令
EXTS.B	Rm,Rn	将 Rm 从字节符号扩展 →Rn	0110nnnnmmmm1110	—	—	—
EXTS.W	Rm,Rn	将 Rm 从字符号扩展 →Rn	0110nnnnmmmm1111	—	—	—
EXTU.B	Rm,Rn	将 Rm 从字节零扩展 →Rn	0110nnnnmmmm1100	—	—	—
EXTU.W	Rm,Rn	将 Rm 从字零扩展 →Rn	0110nnnnmmmm1101	—	—	—
MAC.L	@Rm+,@Rn+	有符号 (Rn)×(Rm)+MAC→MAC Rn+4→Rn, Rm+4→Rm 32×32+64→64 位	0000nnnnmmmm1111	—	—	—
MAC.W	@Rm+,@Rn+	有符号 (Rn)×(Rm)+MAC→MAC Rn+2→Rn, Rm+2→Rm 16×16+64→64 位	0100nnnnmmmm1111	—	—	—
MUL.L	Rm,Rn	Rn×Rm→MACL 32×32→32 位	0000nnnnmmmm0111	—	—	—
MULS.W	Rm,Rn	有符号 Rn×Rm→MACL 16×16→32 位	0010nnnnmmmm1111	—	—	—
MULU.W	Rm,Rn	无符号 Rn×Rm→MACL 16×16→32 位	0010nnnnmmmm1110	—	—	—
NEG	Rm,Rn	0-Rm→Rn	0110nnnnmmmm1011	—	—	—
NEGC	Rm,Rn	0-Rm-T→Rn, 借位 →T	0110nnnnmmmm1010	—	借位	—
SUB	Rm,Rn	Rn-Rm→Rn	0011nnnnmmmm1000	—	—	—
SUBC	Rm,Rn	Rn-Rm-T→Rn, 借位 →T	0011nnnnmmmm1010	—	借位	—
SUBV	Rm,Rn	Rn-Rm→Rn, 下溢 →T	0011nnnnmmmm1011	—	下溢	—

表 3.6 逻辑运算指令

指令		运行	操作码	特权	T 位	新指令
AND	Rm,Rn	Rn & Rm→Rn	0010nnnnmmmm1001	—	—	—
AND	#imm,R0	R0 & imm→R0	11001001iiiiiiii	—	—	—
AND.B	#imm,@(R0,GBR)	(R0+GBR) & imm→(R0+GBR)	11001101iiiiiiii	—	—	—
NOT	Rm,Rn	~Rm→Rn	0110nnnnmmmm0111	—	—	—
OR	Rm,Rn	Rn ｜ Rm→Rn	0010nnnnmmmm1011	—	—	—
OR	#imm,R0	R0 ｜ imm→R0	11001011iiiiiiii	—	—	—
OR.B	#imm,@(R0,GBR)	(R0+GBR) | imm→(R0+GBR)	11001111iiiiiiii	—	—	—
TAS.B	@Rn	(Rn) 为 0 时 1→T 除此以外时 0→T 对于两者 1→(Rn) 的 MSB	0100nnnn00011011	—	测试结果	—
TST	Rm,Rn	Rn & Rm, 结果为 0 时 1→T 除此以外时 0→T	0010nnnnmmmm1000	—	测试结果	—
TST	#imm,R0	R0 & imm, 结果为 0 时 1→T 除此以外时 0→T	11001000iiiiiiii	—	测试结果	—
TST.B	#imm,@(R0,GBR)	(R0+GBR)&imm, 结果为 0 时 1→T 除此以外时 0→T	11001100iiiiiiii	—	测试结果	—
XOR	Rm,Rn	Rn ^ Rm→Rn	0010nnnnmmmm1010	—	—	—
XOR	#imm,R0	R0 ^ imm→R0	11001010iiiiiiii	—	—	—
XOR.B	#imm,@(R0,GBR)	(R0+GBR) ^ imm→(R0+GBR)	11001110iiiiiiii	—	—	—

表 3.7 移位指令

指令		运行	操作码	特权	T 位	新指令
ROTL	Rn	T←Rn←MSB	0100nnnn00000100	—	MSB	—
ROTR	Rn	LSB→Rn→T	0100nnnn00000101	—	LSB	—
ROTCL	Rn	T←Rn←T	0100nnnn00100100	—	MSB	—
ROTCR	Rn	T→Rn→T	0100nnnn00100101	—	LSB	—
SHAD	Rm, Rn	Rm ≥ 0 时 Rn<<Rm→Rn, Rm<0 时 Rn>>Rm→[MSB→Rn]	0100nnnnmmmm1100	—	—	—
SHAL	Rn	T←Rn←0	0100nnnn00100000	—	MSB	—
SHAR	Rn	MSB→Rn→T	0100nnnn00100001	—	LSB	—
SHLD	Rm, Rn	Rm ≥ 0 时 Rn<<Rm→Rn, Rm<0 时 Rn>>Rm→[0→Rn]	0100nnnnmmmm1101	—	—	—
SHLL	Rn	T←Rn←0	0100nnnn00000000	—	MSB	—
SHLR	Rn	0→Rn→T	0100nnnn00000001	—	LSB	—
SHLL2	Rn	Rn<<2→Rn	0100nnnn00001000	—	—	—
SHLR2	Rn	Rn>>2→Rn	0100nnnn00001001	—	—	—
SHLL8	Rn	Rn<<8→Rn	0100nnnn00011000	—	—	—
SHLR8	Rn	Rn>>8→Rn	0100nnnn00011001	—	—	—
SHLL16	Rn	Rn<<16→Rn	0100nnnn00101000	—	—	—
SHLR16	Rn	Rn>>16→Rn	0100nnnn00101001	—	—	—

表 3.8 转移指令

指令		运行	操作码	特权	T 位	新指令
BF	label	T=0 时 disp×2+PC+4→PC,T=1 时 nop	10001011dddddddd	—	—	—
BF/S	label	延迟转移 ,T=0 时 disp×2+PC+4→PC, T=1 时 nop	10001111dddddddd	—	—	—
BT	label	T=1 时 disp×2+PC+4→PC,T=0 时 nop	10001001dddddddd	—	—	—
BT/S	label	延迟转移 ,T=1 时 disp×2+PC+4→PC,T=0 时 nop	10001101dddddddd	—	—	—
BRA	label	延迟转移 ,disp×2+PC+4→PC	1010dddddddddddd	—	—	—
BRAF	Rn	延迟转移 ,Rn+PC+4→PC	0000nnnn00100011	—	—	—
BSR	label	延迟转移 ,PC+4→PR,disp×2+PC+4→PC	1011dddddddddddd	—	—	—
BSRF	Rn	延迟转移 ,PC+4→PR,Rn+PC+4→PC	0000nnnn00000011	—	—	—
JMP	@Rn	延迟转移 ,Rn→PC	0100nnnn00101011	—	—	—
JSR	@Rn	延迟转移 ,PC+4→PR,Rn→PC	0100nnnn00001011	—	—	—
RTS		延迟转移 ,PR→PC	0000000000001011	—	—	—

表 3.9 系统控制指令

指令		运行	操作码	特权	T 位	新指令
CLRMAC		0→MACH,MACL	0000000000101000	—	—	—
CLRS		0→S	0000000001001000	—	—	—
CLRT		0→T	0000000000001000	—	0	—
ICBI	@Rn	使通过逻辑地址 Rn 表示的指令高速缓存无效	0000nnnn11100011	—	—	新指令
LDC	Rm,SR	Rm→SR	0100mmmm00001110	特权	LSB	—
LDC	Rm,GBR	Rm→GBR	0100mmmm00011110	—	—	—
LDC	Rm,VBR	Rm→VBR	0100mmmm00101110	特权	—	—
LDC	Rm,SGR	Rm→SGR	0100mmmm00111010	特权	—	新指令
LDC	Rm,SSR	Rm→SSR	0100mmmm00111110	特权	—	—
LDC	Rm,SPC	Rm→SPC	0100mmmm01001110	特权	—	—
LDC	Rm,DBR	Rm→DBR	0100mmmm11111010	特权	—	—
LDC	Rm,Rn_BANK	Rm→Rn_BANK(n=0 ～ 7)	0100mmmm1nnn1110	特权	—	—
LDC.L	@Rm+,SR	(Rm)→SR,Rm+4→Rm	0100mmmm00000111	特权	LSB	—
LDC.L	@Rm+,GBR	(Rm)→GBR,Rm+4→Rm	0100mmmm00010111	—	—	—
LDC.L	@Rm+,VBR	(Rm)→VBR,Rm+4→Rm	0100mmmm00100111	特权	—	—
LDC.L	@Rm+,SGR	(Rm)→SGR,Rm+4→Rm	0100mmmm00110110	特权	—	新指令
LDC.L	@Rm+,SSR	(Rm)→SSR,Rm+4→Rm	0100mmmm00110111	特权	—	—
LDC.L	@Rm+,SPC	(Rm)→SPC,Rm+4→Rm	0100mmmm01000111	特权	—	—
LDC.L	@Rm+,DBR	(Rm)→DBR,Rm+4→Rm	0100mmmm11110110	特权	—	—
LDC.L	@Rm+,Rn_BANK	(Rm)→Rn_BANK,Rm+4→Rm	0100mmmm1nnn0111	特权	—	—
LDS	Rm,MACH	Rm→MACH	0100mmmm00001010	—	—	—
LDS	Rm,MACL	Rm→MACL	0100mmmm00011010	—	—	—
LDS	Rm,PR	Rm→PR	0100mmmm00101010	—	—	—
LDS.L	@Rm+,MACH	(Rm)→MACH,Rm+4→Rm	0100mmmm00000110	—	—	—
LDS.L	@Rm+,MACL	(Rm)→MACL,Rm+4→Rm	0100mmmm00010110	—	—	—

指令		运行	操作码	特权	T 位	新指令
LDS.L	@Rm+,PR	(Rm)→PR,Rm+4→Rm	0100mmmm00100110	—	—	—
LDTLB		PTEH/PTEL→TLB	0000000000111000	特权	—	—
MOVCA.L	R0,@Rn	(未获取高速缓存块)R0→(Rn)	0000nnnn11000011	—	—	—
NOP		无操作	0000000000001001	—	—	—
OCBI	@Rn	使操作数高速缓存块无效	0000nnnn10010011	—	—	—
OCBP	@Rn	回写操作数高速缓存块并使之无效	0000nnnn10100011	—	—	—
OCBWB	@Rn	回写操作数高速缓存块	0000nnnn10110011	—	—	—
PREF	@Rn	(Rn)→ 操作数高速缓存	0000nnnn10000011	—	—	—
PREFI	@Rn	将 32 字节的指令块读入指令高速缓存	0000nnnn11010011	—	—	新指令
RTE		延迟转移 ,SSR/SPC→SR/PC	0000000000101011	特权	—	—
SETS		1→S	0000000001011000	—	—	—
SETT		1→T	0000000000011000	—	1	—
SLEEP		睡眠或待机	0000000000011011	特权	—	—
STC	SR,Rn	SR→Rn	0000nnnn00000010	特权	—	—
STC	GBR,Rn	GBR→Rn	0000nnnn00010010	—	—	—
STC	VBR,Rn	VBR→Rn	0000nnnn00100010	特权	—	—
STC	SSR,Rn	SSR→Rn	0000nnnn00110010	特权	—	—
STC	SPC,Rn	SPC→Rn	0000nnnn01000010	特权	—	—
STC	SGR,Rn	SGR→Rn	0000nnnn00111010	特权	—	—
STC	DBR,Rn	DBR→Rn	0000nnnn11111010	特权	—	—
STC	Rm_BANK,Rn	Rm_BANK→Rn(m=0 ～ 7)	0000nnnn1mmm0010	特权	—	—
STC.L	SR,@-Rn	Rn-4→Rn,SR→(Rn)	0100nnnn00000011	特权	—	—
STC.L	GBR,@-Rn	Rn-4→Rn,GBR→(Rn)	0100nnnn00010011	—	—	—
STC.L	VBR,@-Rn	Rn-4→Rn,VBR→(Rn)	0100nnnn00100011	特权	—	—
STC.L	SSR,@-Rn	Rn-4→Rn,SSR→(Rn)	0100nnnn00110011	特权	—	—
STC.L	SPC,@-Rn	Rn-4→Rn,SPC→(Rn)	0100nnnn01000011	特权	—	—
STC.L	SGR,@-Rn	Rn-4→Rn,SGR→(Rn)	0100nnnn00110010	特权	—	—
STC.L	DBR,@-Rn	Rn-4→Rn,DBR→(Rn)	0100nnnn11110010	特权	—	—
STC.L	Rm_BANK,@-Rn	Rn-4→Rn,Rm_BANK→(Rn) (m=0 ～ 7)	0100nnnn1mmm0011	特权	—	—
STS	MACH,Rn	MACH→Rn	0000nnnn00001010	—	—	—
STS	MACL,Rn	MACL→Rn	0000nnnn00011010	—	—	—
STS	PR,Rn	PR→Rn	0000nnnn00101010	—	—	—
STS.L	MACH,@-Rn	Rn-4→Rn,MACH→(Rn)	0100nnnn00000010	—	—	—
STS.L	MACL,@-Rn	Rn-4→Rn,MACL→(Rn)	0100nnnn00010010	—	—	—
STS.L	PR,@-Rn	Rn-4→Rn,PR→(Rn)	0100nnnn00100010	—	—	—
SYNCO		在本指令以前的数据操作完成之前不开始本指令以后的指令	0000000010101011	—	—	新指令
TRAPA	#imm	imm<<2→TRA,PC+2→SPC, SR→SSR,R15→SGR, 1→SR.MD/BL/RB,H'160→EXPEVT, VBR+H'0100→PC	11000011iiiiiiii	—	—	—

表 3.10 浮点单精度指令

指令		运行	操作码	特权	T 位	新指令
FLDI0	FRn	H'00000000→FRn	1111nnnn10001101	—	—	—
FLDI1	FRn	H'3F800000→FRn	1111nnnn10011101	—	—	—
FMOV	FRm,FRn	FRm→FRn	1111nnnnmmmm1100	—	—	—
FMOV.S	@Rm,FRn	(Rm)→FRn	1111nnnnmmmm1000	—	—	—
FMOV.S	@(R0,Rm),FRn	(R0+Rm)→FRn	1111nnnnmmmm0110	—	—	—
FMOV.S	@Rm+,FRn	(Rm)→FRn,Rm+4→Rm	1111nnnnmmmm1001	—	—	—
FMOV.S	FRm,@Rn	FRm→(Rn)	1111nnnnmmmm1010	—	—	—
FMOV.S	FRm,@-Rn	Rn-4→Rn,FRm→(Rn)	1111nnnnmmmm1011	—	—	—
FMOV.S	FRm,@(R0,Rn)	FRm→(R0+Rn)	1111nnnnmmmm0111	—	—	—
FMOV	DRm,DRn	DRm→DRn	1111nnn0mmm01100	—	—	—
FMOV	@Rm,DRn	(Rm)→DRn	1111nnn0mmmm1000	—	—	—
FMOV	@(R0,Rm),DRn	(R0+Rm)→DRn	1111nnn0mmmm0110	—	—	—
FMOV	@Rm+,DRn	(Rm)→DRn,Rm+8→Rm	1111nnn0mmmm1001	—	—	—
FMOV	DRm,@Rn	DRm→(Rn)	1111nnnnmmm01010	—	—	—
FMOV	DRm,@-Rn	Rn-8→Rn,DRm→(Rn)	1111nnnnmmm01011	—	—	—
FMOV	DRm,@(R0,Rn)	DRm→(R0+Rn)	1111nnnnmmm00111	—	—	—
FLDS	FRm,FPUL	FRm→FPUL	1111mmmm00011101	—	—	—
FSTS	FPUL,FRn	FPUL→FRn	1111nnnn00001101	—	—	—
FABS	FRn	FRn & H'7FFF FFFF→FRn	1111nnnn01011101	—	—	—
FADD	FRm,FRn	FRn+FRm→FRn	1111nnnnmmmm0000	—	—	—
FCMP/EQ	FRm,FRn	FRn=FRm 时 1→T 除此以外时 0→T	1111nnnnmmmm0100	—	比较结果	—
FCMP/GT	FRm,FRn	FRn ＞ FRm 时 1→T 除此以外时 0→T	1111nnnnmmmm0101	—	比较结果	—
FDIV	FRm,FRn	FRn/FRm→FRn	1111nnnnmmmm0011	—	—	—
FLOAT	FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	—	—	—
FMAC	FR0,FRm,FRn	FR0×FRm+FRn→FRn	1111nnnnmmmm1110	—	—	—
FMUL	FRm,FRn	FRn×FRm→FRn	1111nnnnmmmm0010	—	—	—
FNEG	FRn	FRn ^ H'80000000→FRn	1111nnnn01001101	—	—	—
FSQRT	FRn	sqrt(FRn)→FRn*	1111nnnn01101101	—	—	—
FSUB	FRm,FRn	FRn - FRm→FRn	1111nnnnmmmm0001	—	—	—
FTRC	FRm,FPUL	(long)FRm→FPUL	1111mmmm00111101	—	—	—

【注】 * sqrt(FRn) 表示 FRn 的平方根。

表 3.11 浮点双精度指令

指令		运行	操作码	特权	T 位	新指令
FABS	DRn	DRn&H'7FFF FFFF FFFF FFFF→DRn	1111nnn001011101	—	—	—
FADD	DRm,DRn	DRn+DRm→DRn	1111nnn0mmm00000	—	—	—
FCMP/EQ	DRm,DRn	DRn=DRm 时 1→T 除此以外时 0→T	1111nnn0mmm00100	—	比较结果	—
FCMP/GT	DRm,DRn	DRn ＞ DRm 时 1→T 除此以外时 0→T	1111nnn0mmm00101	—	比较结果	—
FDIV	DRm,DRn	DRn/DRm→DRn	1111nnn0mmm00011	—	—	—
FCNVDS	DRm,FPUL	double_to_float(DRm)→FPUL	1111mmm010111101	—	—	—
FCNVSD	FPUL,DRn	float_to_double(FPUL)→DRn	1111nnn010101101	—	—	—
FLOAT	FPUL,DRn	(float)FPUL→DRn	1111nnn000101101	—	—	—
FMUL	DRm,DRn	DRn×DRm→DRn	1111nnn0mmm00010	—	—	—
FNEG	DRn	DRn ^ H'8000 0000 0000 0000→DRn	1111nnn001001101	—	—	—
FSQRT	DRn	sqrt(DRn)→DRn*	1111nnn001101101	—	—	—
FSUB	DRm,DRn	DRn– DRm→DRn	1111nnn0mmm00001	—	—	—
FTRC	DRm,FPUL	(long)DRm→FPUL	1111mmm000111101	—	—	—

【注】 * sqrt(DRn) 表示 DRn 的平方根。

表 3.12 浮点控制指令

指令		运行	操作码	特权	T 位	新指令
LDS	Rm,FPSCR	Rm→FPSCR	0100mmmm01101010	—	—	—
LDS	Rm,FPUL	Rm→FPUL	0100mmmm01011010	—	—	—
LDS.L	@Rm+,FPSCR	(Rm)→FPSCR,Rm+4→Rm	0100mmmm01100110	—	—	—
LDS.L	@Rm+,FPUL	(Rm)→FPUL,Rm+4→Rm	0100mmmm01010110	—	—	—
STS	FPSCR,Rn	FPSCR→Rn	0000nnnn01101010	—	—	—
STS	FPUL,Rn	FPUL→Rn	0000nnnn01011010	—	—	—
STS.L	FPSCR,@-Rn	Rn-4→Rn,FPSCR→(Rn)	0100nnnn01100010	—	—	—
STS.L	FPUL,@-Rn	Rn-4→Rn,FPUL→(Rn)	0100nnnn01010010	—	—	—

表 3.13 浮点图形强化指令

指令		运行	操作码	特权	T 位	新指令
FMOV	DRm,XDn	DRm→XDn	1111nnn1mmm01100	—	—	—
FMOV	XDm,DRn	XDm→DRn	1111nnn0mmm11100	—	—	—
FMOV	XDm,XDn	XDm→XDn	1111nnn1mmm11100	—	—	—
FMOV	@Rm,XDn	(Rm)→XDn	1111nnn1mmmm1000	—	—	—
FMOV	@Rm+,XDn	(Rm)→XDn,Rm+8→Rm	1111nnn1mmmm1001	—	—	—
FMOV	@(R0,Rm),XDn	(R0+Rm)→XDn	1111nnn1mmmm0110	—	—	—
FMOV	XDm,@Rn	XDm→(Rn)	1111nnnnmmm11010	—	—	—
FMOV	XDm,@-Rn	Rn-8→Rn,XDm→(Rn)	1111nnnnmmm11011	—	—	—
FMOV	XDm,@(R0,Rn)	XDm→(R0+Rn)	1111nnnnmmm10111	—	—	—
FIPR	FVm,FVn	inner_product(FVm,FVn)→FR[n+3]	1111nnmm11101101	—	—	—
FTRV	XMTRX,FVn	transform_vector(XMTRX,FVn)→FVn	1111nn0111111101	—	—	—
FRCHG		～ FRSCR.FR→FRSCR.FR	1111101111111101	—	—	—
FSCHG		～ FPSCR.SZ→FPSCR.SZ	1111001111111101	—	—	—
FPCHG		～ FPSCR.PR→FPSCR.PR	1111011111111101	—	—	新指令
FSRRA	FRn	1/sqrt(FRn)→FRn*	1111nnnn01111101	—	—	新指令
FSCA	FPUL,DRn	sin(FPUL)→FRn cos(FPUL)→FR[n+1]	1111nnn011111101	—	—	新指令

【注】 * sqrt(FRn) 表示 FRn 的平方根。

第 4 章 流水线操作

SH-4A 为 2 条指令并行型 （2-ILP, Instruction-Level-Parallelism）超标量流水线处理微处理器。能以流水线方式执行指令，并可并行执行 2 条指令。

4.1 流水线

基本流水线如**图 4.1** 所示。通常，流水线由取指令 （I1、 I2）、解码 / 寄存器读取 （ID）、执行 （E1、E2、 E3）、回写 （WB）这 7 个阶段构成。将 1 条指令作为基本流水线组合来执行。

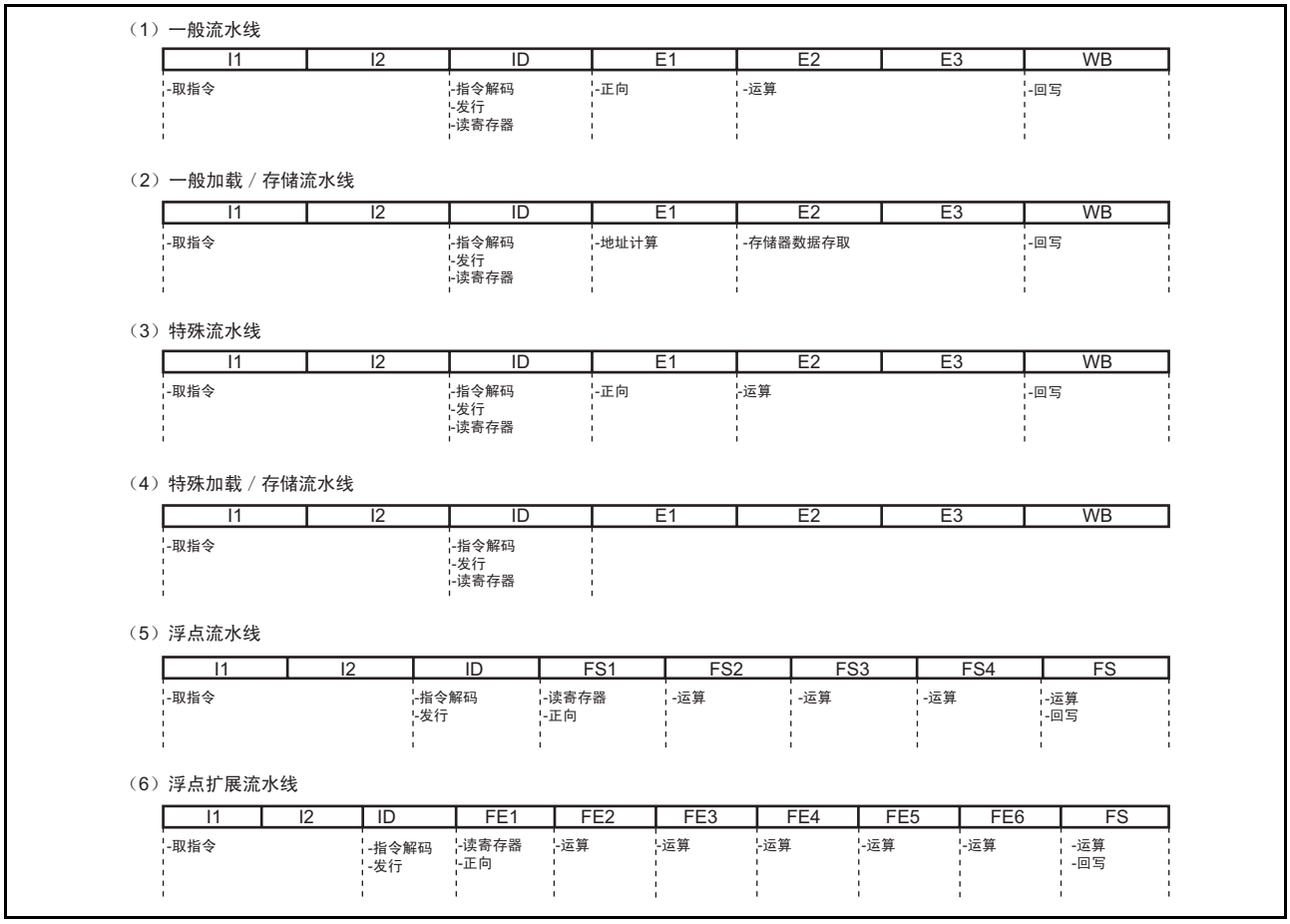


图 4.1 基本流水线

指令执行模式如图 4.2 所示。图 4.2 所使用的记述及其含义如下所示。

表 4.1 指令执行模式记述说明

记 述	含 义
E1 E2 E3 WB	占有 CPU EX 管道
S1 S2 S3 WB	占有 CPU LS 管道 （有存储器存取时）
s1 s2 s3 WB	占有 CPU LS 管道 （没有存储器存取时）
E1/S1	占有 CPU EX 或 LS
E1S1 、E1s1	占有 CPU EX 与 LS
M2 M3 MS	占有 CPU MULT 运算器
FE1 FE2 FE3 FE4 FE5 FE6 FS	占有 FPU-EX 管道
FS1 FS2 FS3 FS4 FS	占有 FPU-LS 管道
ID	锁定 ID 阶段
└──	占有 CPU 与 FPU 的管道

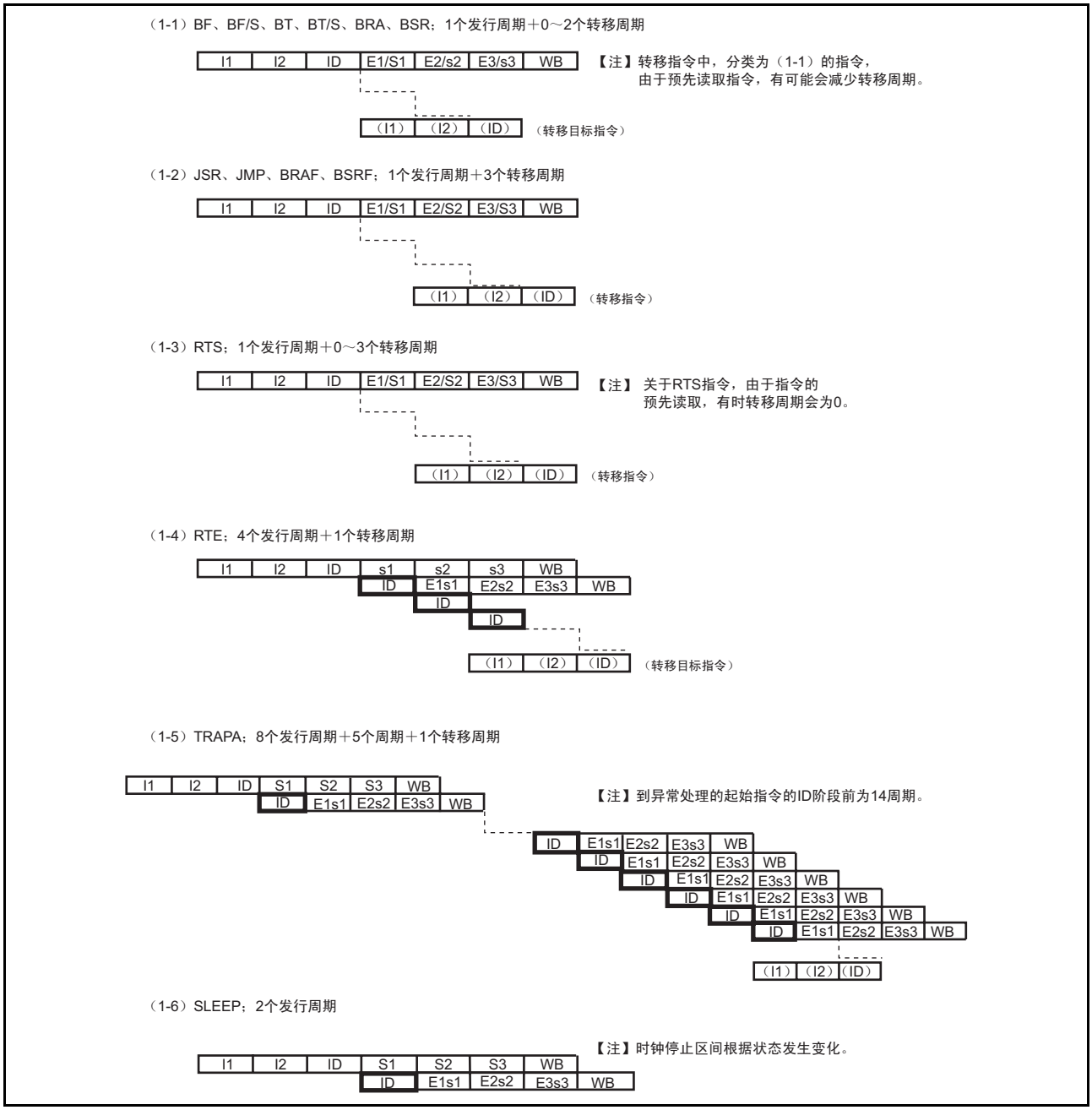

（1-1）BF、BF/S、BT、BT/S、BRA、BSR：1个发行周期＋0～2个转移周期
I1 I2 ID E1/S1 E2/s2 E3/s3 WB
【注】转移指令中，分类为（1-1）的指令，
由于预先读取指令，有可能会减少转移周期。
（I1） （I2） （ID）
（转移目标指令）
（1-2）JSR、JMP、BRAF、BSRF：1个发行周期＋3个转移周期
I1 I2 ID E1/S1 E2/S2 E3/S3 WB
（I1） （I2） （ID）
（转移指令）
（1-3）RTS：1个发行周期＋0～3个转移周期
I1 I2 ID E1/S1 E2/S2 E3/S3 WB
【注】 关于RTS指令，由于指令的
预先读取，有时转移周期会为0。
（I1） （I2） （ID）
（转移指令）
（1-4）RTE：4个发行周期＋1个转移周期
I1 I2 ID s1 s2 s3 WB
ID E1s1 E2s2 E3s3 WB
ID
ID
（I1） （I2） （ID）
（转移目标指令）
（1-5）TRAPA：8个发行周期＋5个周期＋1个转移周期
I1 I2 ID S1 S2 S3 WB
ID E1s1 E2s2 E3s3 WB
【注】到异常处理的起始指令的ID阶段前为14周期。
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
（I1） （I2） （ID）
（1-6）SLEEP：2个发行周期
【注】时钟停止区间根据状态发生变化。
I1 I2 ID S1 S2 S3 WB
ID E1s1 E2s2 E3s3 WB

图 4.2 指令执行模式 （1）

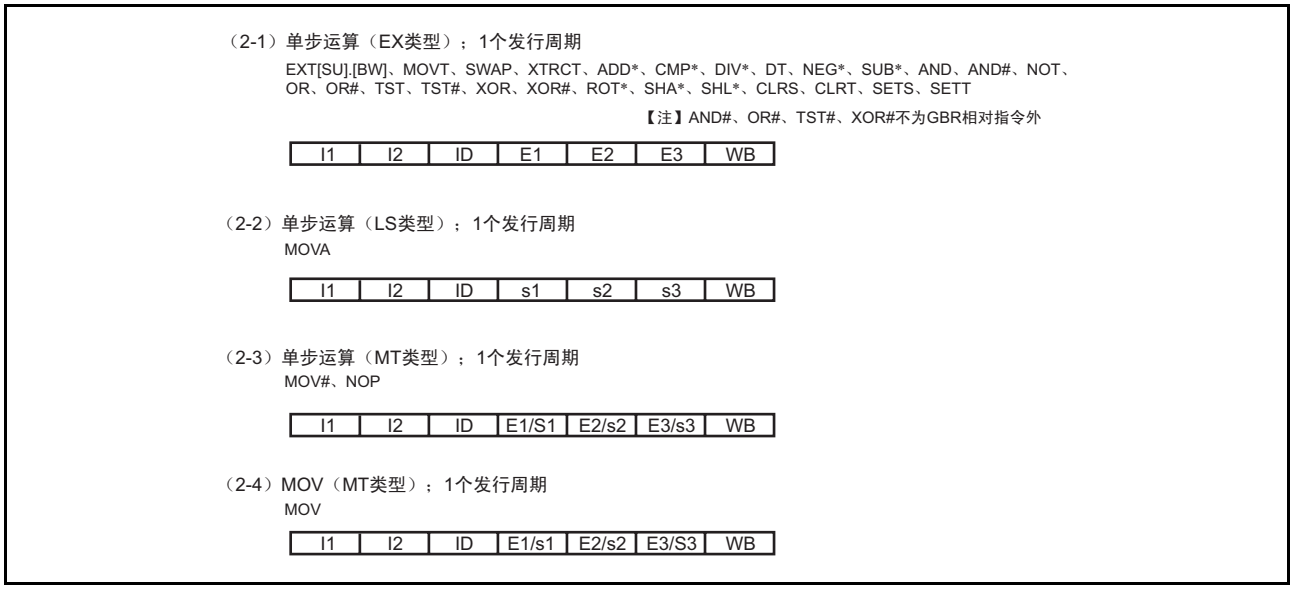

（2-1）单步运算（EX类型）；1个发行周期
EXT[SU].[BW]、MOVT、SWAP、XTRCT、ADD*、CMP*、DIV*、DT、NEG*、SUB*、AND、AND#、NOT、
OR、OR#、TST、TST#、XOR、XOR#、ROT*、SHA*、SHL*、CLRS、CLRT、SETS、SETT
【注】AND#、OR#、TST#、XOR#不为GBR相对指令外
I1 I2 ID E1 E2 E3 WB
（2-2）单步运算（LS类型）；1个发行周期
MOVA
I1 I2 ID s1 s2 s3 WB
（2-3）单步运算（MT类型）；1个发行周期
MOV#、NOP
I1 I2 ID E1/S1 E2/s2 E3/s3 WB
（2-4）MOV（MT类型）；1个发行周期
MOV
I1 I2 ID E1/s1 E2/s2 E3/S3 WB

图 4.2 指令执行模式 （2）

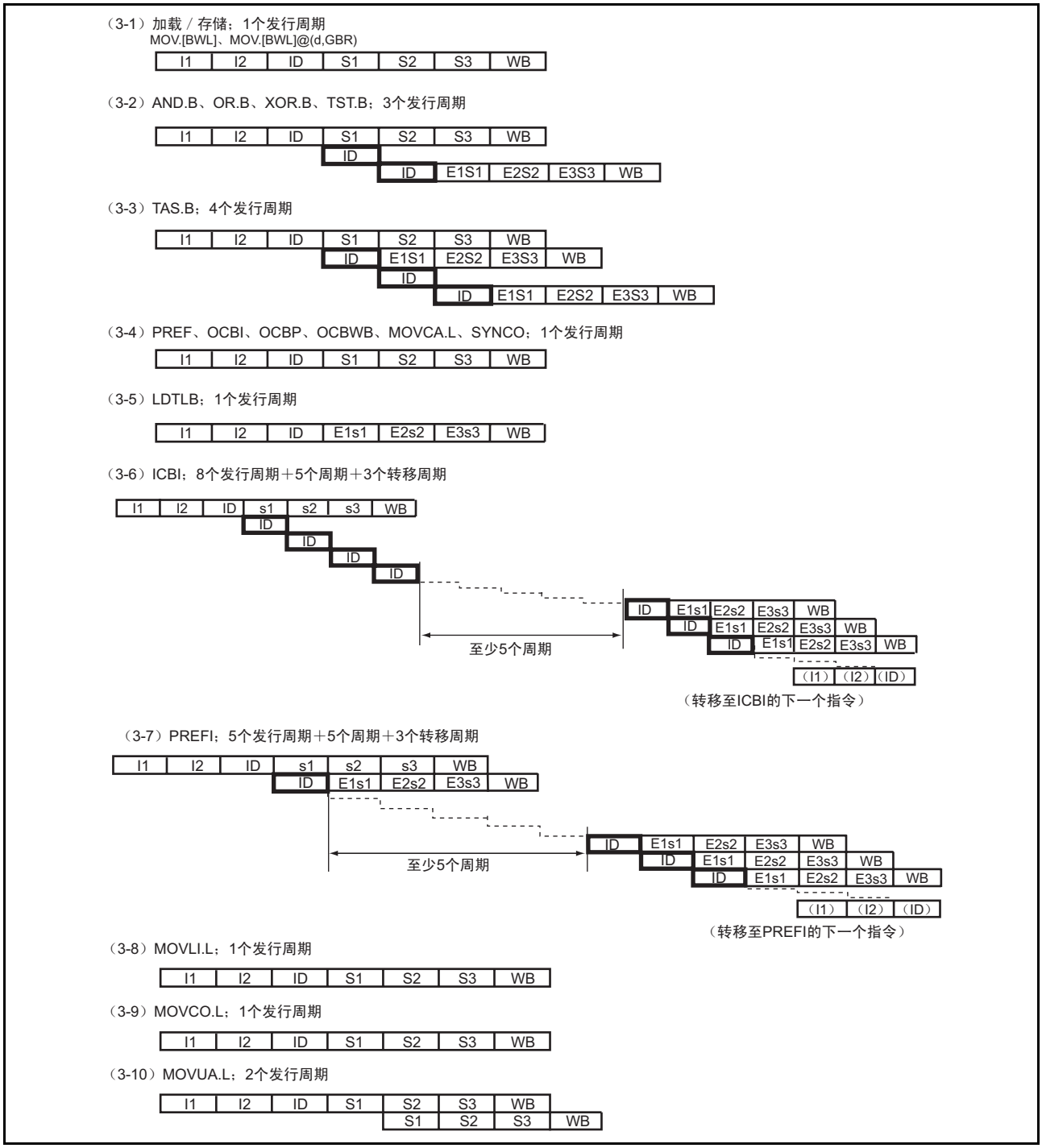

（3-1）加载 / 存储：1个发行周期
MOV.[BWL]、MOV.[BWL]@(d,GBR)
I1 I2 ID S1 S2 S3 WB
（3-2）AND.B、OR.B、XOR.B、TST.B：3个发行周期
I1 I2 ID S1 S2 S3 WB
ID
ID E1S1 E2S2 E3S3 WB
（3-3）TAS.B：4个发行周期
I1 I2 ID S1 S2 S3 WB
ID E1S1 E2S2 E3S3 WB
ID
ID E1S1 E2S2 E3S3 WB
（3-4）PREF、OCBI、OCBP、OCBWB、MOVCA.L、SYNCO：1个发行周期
I1 I2 ID S1 S2 S3 WB
（3-5）LDTLB：1个发行周期
I1 I2 ID E1s1 E2s2 E3s3 WB
（3-6）ICBI：8个发行周期＋5个周期＋3个转移周期
I1 I2 ID s1 s2 s3 WB
ID
ID
ID
ID
至少5个周期
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
（I1） （I2） （ID）
（转移至ICBI的下一个指令）
（3-7）PREFI：5个发行周期＋5个周期＋3个转移周期
I1 I2 ID s1 s2 s3 WB
ID E1s1 E2s2 E3s3 WB
至少5个周期
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
ID E1s1 E2s2 E3s3 WB
（I1） （I2） （ID）
（转移至PREFI的下一个指令）
（3-8）MOVLI.L：1个发行周期
I1 I2 ID S1 S2 S3 WB
（3-9）MOVCO.L：1个发行周期
I1 I2 ID S1 S2 S3 WB
（3-10）MOVUA.L：2个发行周期
I1 I2 ID S1 S2 S3 WB
S1 S2 S3 WB

图 4.2 指令执行模式 （3）

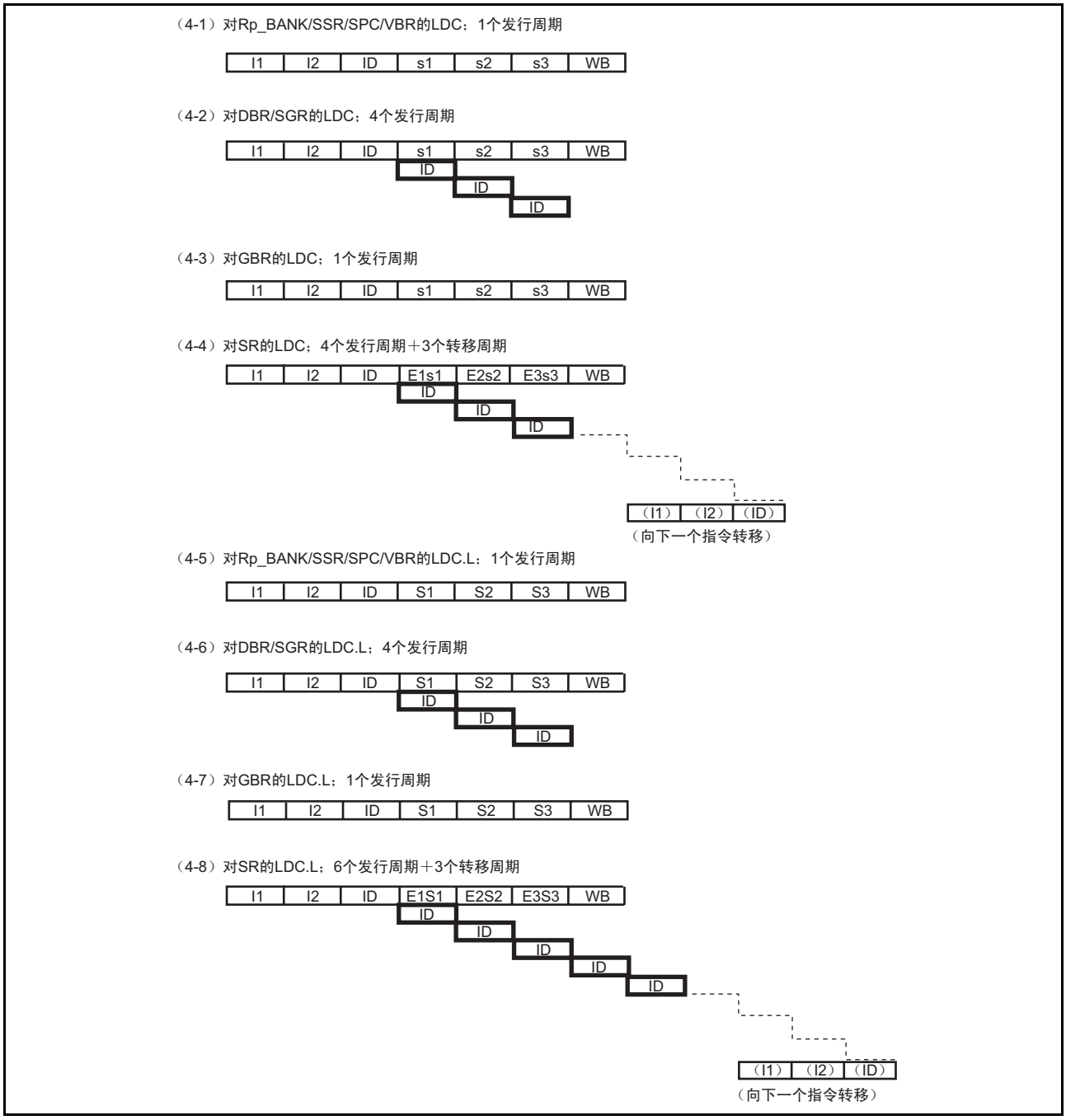

（4-1）对Rp_BANK/SSR/SPC/VBR的LDC：1个发行周期
I1 I2 ID s1 s2 s3 WB
（4-2）对DBR/SGR的LDC：4个发行周期
I1 I2 ID s1 s2 s3 WB
ID
ID
ID
（4-3）对GBR的LDC：1个发行周期
I1 I2 ID s1 s2 s3 WB
（4-4）对SR的LDC：4个发行周期＋3个转移周期
I1 I2 ID E1s1 E2s2 E3s3 WB
ID
ID
ID
（I1） （I2） （ID）
（向下一个指令转移）
（4-5）对Rp_BANK/SSR/SPC/VBR的LDC.L：1个发行周期
I1 I2 ID S1 S2 S3 WB
（4-6）对DBR/SGR的LDC.L：4个发行周期
I1 I2 ID S1 S2 S3 WB
ID
ID
ID
（4-7）对GBR的LDC.L：1个发行周期
I1 I2 ID S1 S2 S3 WB
（4-8）对SR的LDC.L：6个发行周期＋3个转移周期
I1 I2 ID E1S1 E2S2 E3S3 WB
ID
ID
ID
ID
ID
（I1） （I2） （ID）
（向下一个指令转移）

图 4.2 指令执行模式 （4）

（4-9）来自DBR/GBR/Rp_BANK/SSR/SPC/VBR/SGR的STC；1个发行周期

I1	I2	ID	s1	s2	s3	WB

（4-10）来自SR的STC；1个发行周期

I1	I2	ID	E1s1	E2s2	E3s3	WB

（4-11）来自DBR/GBR/Rp_BANK/SSR/SPC/VBR/SGR的STC.L；1个发行周期

I1	I2	ID	S1	S2	S3	WB

（4-12）来自SR的STC.L；1个发行周期

I1	I2	ID	E1S1	E2S2	E3S3	WB

（4-13）来自PR的LDS；1个发行周期

I1	I2	ID	s1	s2	s3	WB

（4-14）来自PR的LDS.L；1个发行周期

I1	I2	ID	S1	S2	S3	WB

（4-15）来自PR的STS；1个发行周期

I1	I2	ID	s1	s2	s3	WB

（4-16）来自PR的STS.L；1个发行周期

I1	I2	ID	S1	S2	S3	WB

（4-17）BSRF、BSR、JSR的延迟槽指令（PR设定）；0个发行周期

（I1）	（I2）	（ID）	（??1）	（??2）	（??3）	（WB）

【注】PR值在延迟槽指令的E3阶段被更新。
在延迟槽使用来自PR的STS、STS.L指令时，使用更新后的PR值。

图 4.2 指令执行模式 （5）

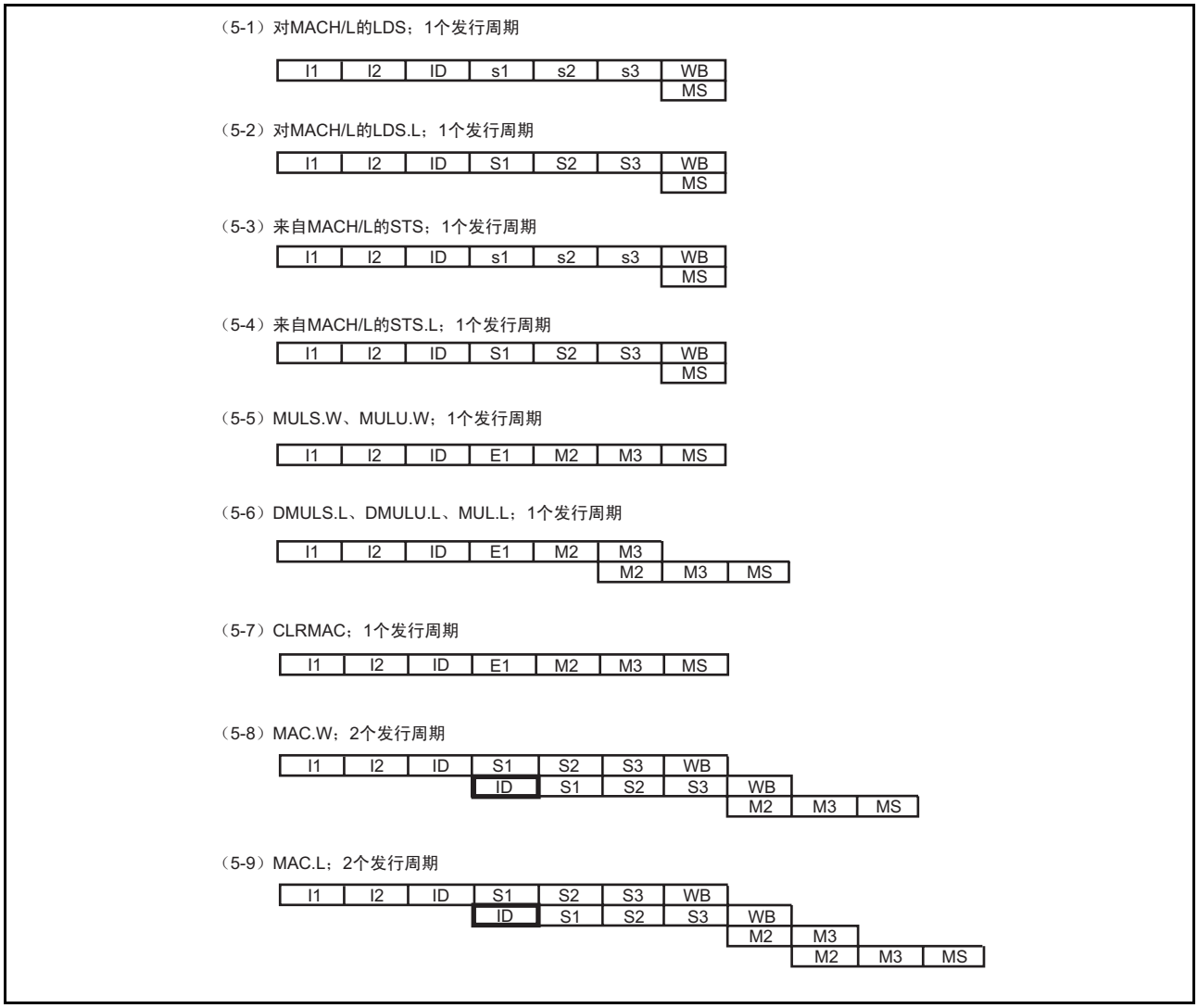

（5-1）对MACH/L的LDS；1个发行周期
I1 I2 ID s1 s2 s3 WB
MS
（5-2）对MACH/L的LDS.L；1个发行周期
I1 I2 ID S1 S2 S3 WB
MS
（5-3）来自MACH/L的STS；1个发行周期
I1 I2 ID s1 s2 s3 WB
MS
（5-4）来自MACH/L的STS.L；1个发行周期
I1 I2 ID S1 S2 S3 WB
MS
（5-5）MULS.W、MULU.W；1个发行周期
I1 I2 ID E1 M2 M3 MS
（5-6）DMULS.L、DMULU.L、MUL.L；1个发行周期
I1 I2 ID E1 M2 M3
M2 M3 MS
（5-7）CLRMAC；1个发行周期
I1 I2 ID E1 M2 M3 MS
（5-8）MAC.W；2个发行周期
I1 I2 ID S1 S2 S3 WB
ID S1 S2 S3 WB
M2 M3 MS
（5-9）MAC.L；2个发行周期
I1 I2 ID S1 S2 S3 WB
ID S1 S2 S3 WB
M2 M3
M2 M3 MS

图 4.2 指令执行模式 （6）

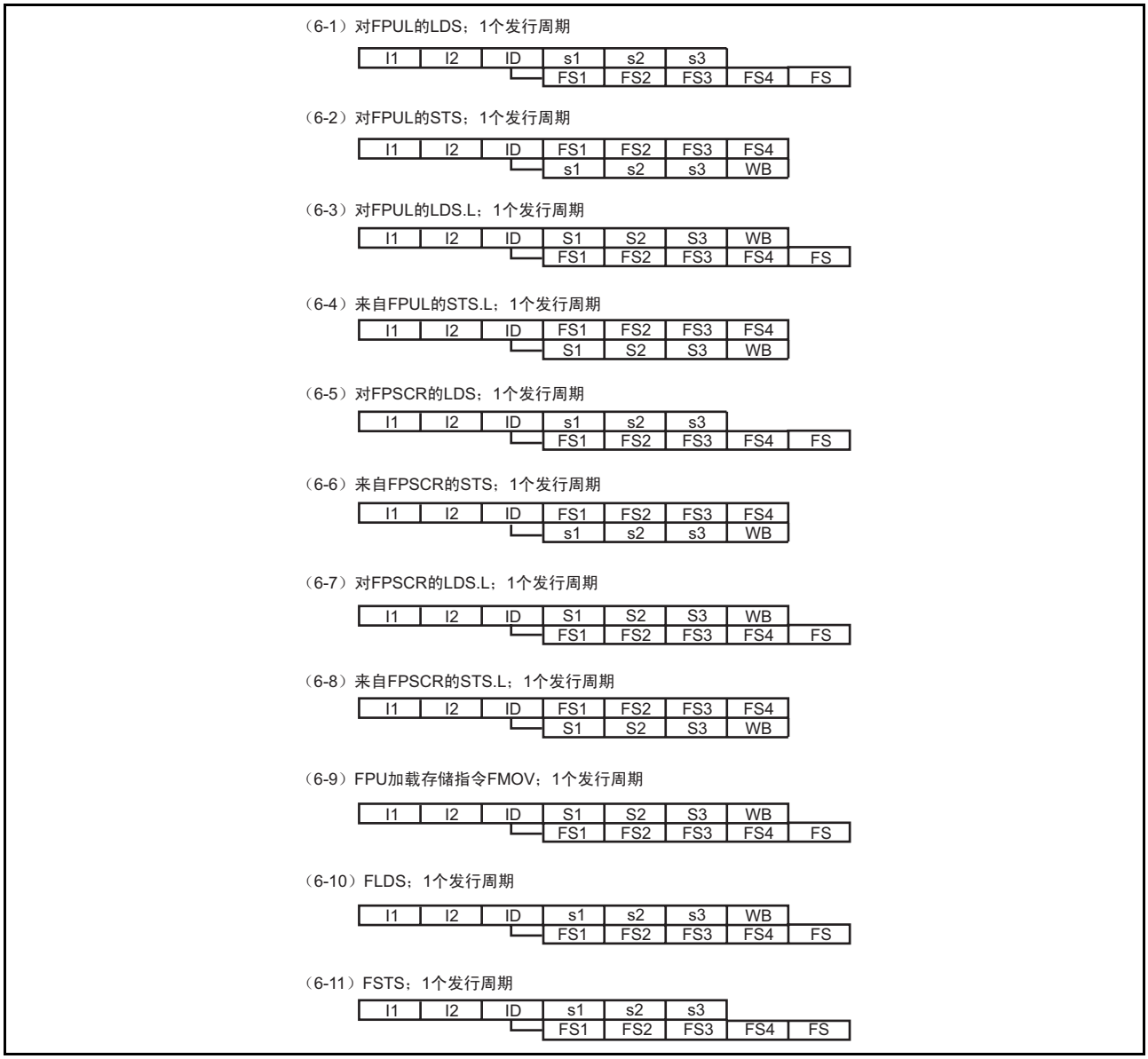

（6-1）对FPUL的LDS：1个发行周期
I1 I2 ID s1 s2 s3
FS1 FS2 FS3 FS4 FS
（6-2）对FPUL的STS：1个发行周期
I1 I2 ID FS1 FS2 FS3 FS4
s1 s2 s3 WB
（6-3）对FPUL的LDS.L：1个发行周期
I1 I2 ID S1 S2 S3 WB
FS1 FS2 FS3 FS4 FS
（6-4）来自FPUL的STS.L：1个发行周期
I1 I2 ID FS1 FS2 FS3 FS4
S1 S2 S3 WB
（6-5）对FPSCR的LDS：1个发行周期
I1 I2 ID s1 s2 s3
FS1 FS2 FS3 FS4 FS
（6-6）来自FPSCR的STS：1个发行周期
I1 I2 ID FS1 FS2 FS3 FS4
s1 s2 s3 WB
（6-7）对FPSCR的LDS.L：1个发行周期
I1 I2 ID S1 S2 S3 WB
FS1 FS2 FS3 FS4 FS
（6-8）来自FPSCR的STS.L：1个发行周期
I1 I2 ID FS1 FS2 FS3 FS4
S1 S2 S3 WB
（6-9）FPU加载存储指令FMOV：1个发行周期
I1 I2 ID S1 S2 S3 WB
FS1 FS2 FS3 FS4 FS
（6-10）FLDS：1个发行周期
I1 I2 ID s1 s2 s3 WB
FS1 FS2 FS3 FS4 FS
（6-11）FSTS：1个发行周期
I1 I2 ID s1 s2 s3
FS1 FS2 FS3 FS4 FS

图 4.2 指令执行模式 （7）

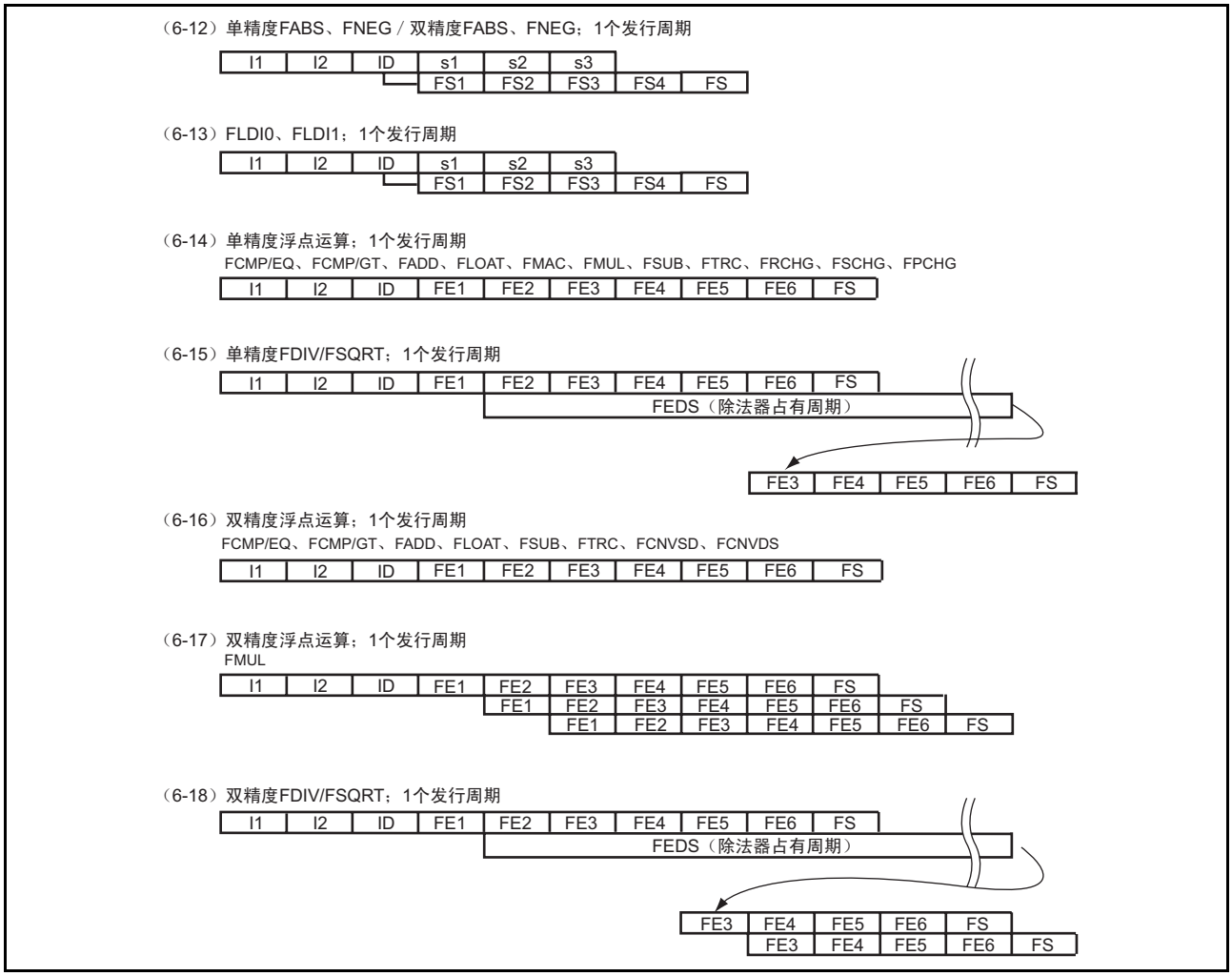


图 4.2 指令执行模式 （8）

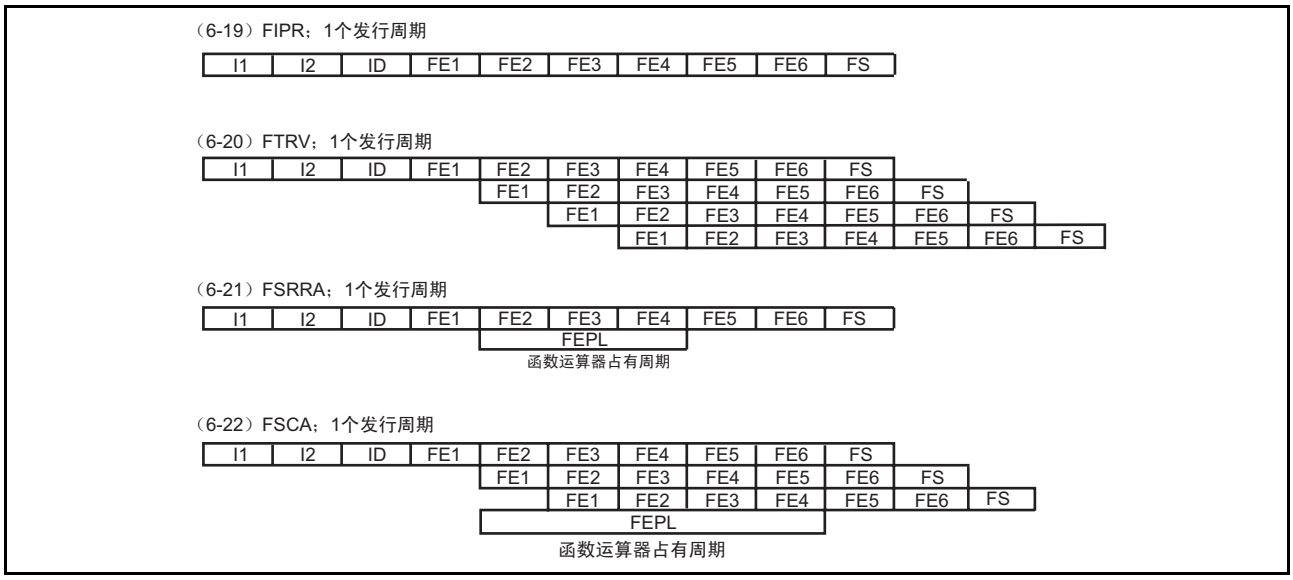


图 4.2 指令执行模式 （9）

4.2 并行执行性

按照所使用的内部功能块，将指令划分为如表 4.2 所示的组。表 4.3 中以组来表示可并行执行的 2 条指令的组合。例如：可并行执行 EX 组的 ADD 与 BR 组的 BRA。

表 4.2 指令组

指令组	指 令			
EX	ADD	DT	ROTL	SHLR8
	ADDC	EXTS	ROTR	SHLR16
	ADDV	EXTU	SETS	SUB
	AND #imm,R0	MOVT	SETT	SUBC
	AND Rm,Rn	MUL.L	SHAD	SUBV
	CLRMAC	MULS.W	SHAL	SWAP
	CLRS	MULU.W	SHAR	TST #imm,R0
	CLRT	NEG	SHLD	TST Rm,Rn
	CMP	NEGC	SHLL	XOR #imm,R0
	DIV0S	NOT	SHLL2	XOR Rm,Rn
	DIV0U	OR #imm,R0	SHLL8	XTRCT
	DIV1	OR Rm,Rn	SHLL16	
	DMUS.L	ROTCL	SHLR	
	DMULU.L	ROTCR	SHLR2	
MT	MOV #imm,Rn	MOV Rm,Rn	NOP	
BR	BF	BRAF	BT	JSR
	BF/S	BSR	BT/S	RTS
	BRA	BSRF	JMP	
LS	FABS	FMOV.S FR,@adr	MOV.[BWL] @adr,R	STC CR2,Rn
	FNEG	FSTS	MOV.[BWL] R,@adr	STC.L CR2,@-Rn
	FLDI0	LDC Rm,CR1	MOVA	STS SR2,Rn
	FLDI1	LDC.L @Rm+,CR1	MOVCA.L	STS.L SR2,@-Rn
	FLDS	LDS Rm,SR1	MOVUA	STS SR1,Rn
	FMOV @adr,FR	LDS Rm,SR2	OCBI	STS.L SR1,@-Rn
	FMOV FR,@adr	LDS.L @adr,SR2	OCBP	
	FMOV FR,FR	LDS.L @Rm+,SR1	OCBWB	
	FMOV.S @adr,FR	LDS.L @Rm+,SR2	PREF	
FE	FADD	FDIV	FRCHG	FSCA
	FSUB	FIPR	FSCHG	FSRRA
	FCMP （S/D）	FLOAT	FSQRT	FPCHG
	FCNVDS	FMAC	FTRC	
	FCNVSD	FMUL	FTRV	

指令组	指 令			
CO	AND.B#imm,@(R0,G	LDC.L @Rm+,SR	PREFI	TRAPA
	BR)	LDTLB	RTE	TST.B
	ICBI	MAC.L	SLEEP	#imm,@(R0,GBR)
	LDC Rm,DBR	MAC.W	STC SR,Rn	XOR.B
	LDC Rm,SGR	MOVCO	STC.L SR,@-Rn	#imm,@(R0,GBR
	LDC Rm,SR	MOVLI	SYNCO	
	LDC.L @Rm+,DBR	OR.B	TAS.B	
	LDC.L @Rm+,SGR	#imm,@(R0,GBR)		

【符号说明】 R ：Rm/Rn
@adr ：地址
SR1 ：MACH/MACL/PR
SR2 ：FPUL/FPSCR
CR1 ：GBR/Rp_BANK/SPC/SSR/VBR
CR2 ：CR1/DBR/SGR
FR ：FRm/FRn/DRm/DRn/XDm/XDn

仅在以下情况可同时执行 2 条指令：
addr （先行指令）与 addr+2 （后行指令）这 2 条指令不超出 1K 字节 （最小的页大小）。
表 4.3 （先行指令 / 后行指令交配表）中可同时执行的指令 （显示为○）。
addr 中的指令与此前的指令无数据命中。
addr+2 中的指令与此前的指令无数据命中。
2 条指令均有效。

表 4.3 先行指令 / 后行指令交配表

		先行指令 （addr）					
		EX	MT	BR	LS	FE	CO
后行指令 （addr+2）	EX	×	○	○	○	○	
	MT	○	○	○	○	○	
	BR	○	○	×	○	○	
	LS	○	○	○	×	○	
	FE	○	○	○	○	×	
	CO						×

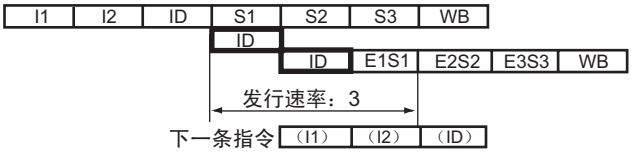
4.3 发行速率与执行状态

指令的发行速率与执行状态如**表 4.4** 所示。**表 4.4** 中的指令组与**表 4.2** 中的分类相对应。另外，本节所示的发行速率与执行状态未考虑因流水线停顿造成的损失周期。

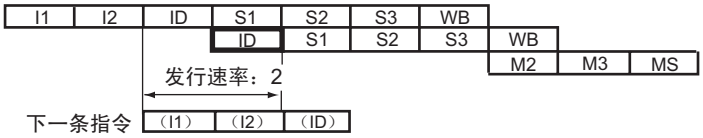
（1）发行速率

发行速率表示一条指令的发行与下一条指令发行的间隔时间。

（例）AND.B指令



（例）MAC.W指令

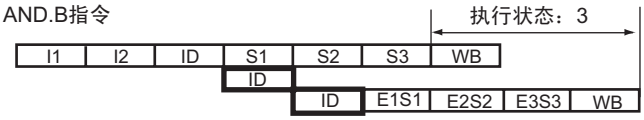


（2）执行状态

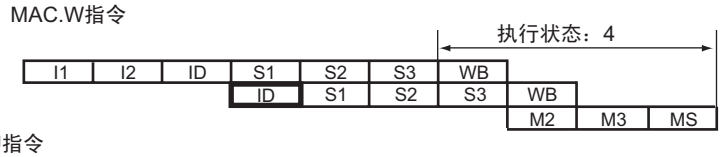
执行状态是用以下基准表示指令占有流水线的周期数。

●CPU指令

（例）AND.B指令

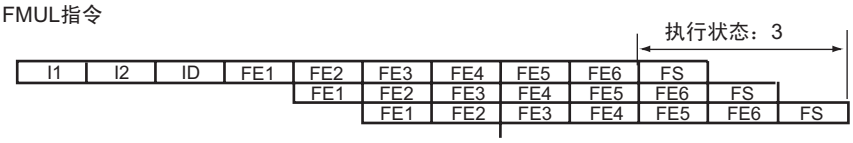


（例）MAC.W指令



●FPU指令

（例）FMUL指令



（例）FDIV指令

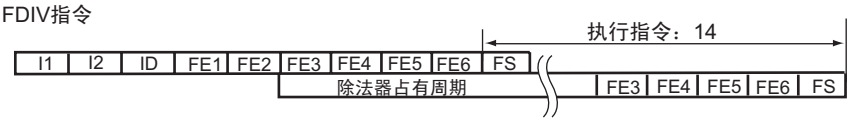


表 4.4 发行速率与执行状态

功能分类	No	指令		指令组	发行速率	执行状态	执行模式
数据传送指令	1	EXTS.B	Rm,Rn	EX	1	1	2-1
	2	EXTS.W	Rm,Rn	EX	1	1	2-1
	3	EXTU.B	Rm,Rn	EX	1	1	2-1
	4	EXTU.W	Rm,Rn	EX	1	1	2-1
	5	MOV	Rm,Rn	MT	1	1	2-4
	6	MOV	#imm,Rn	MT	1	1	2-3
	7	MOVA	@(disp,PC),R0	LS	1	1	2-2
	8	MOV.W	@(disp,PC),Rn	LS	1	1	3-1
	9	MOV.L	@(disp,PC),Rn	LS	1	1	3-1
	10	MOV.B	@Rm,Rn	LS	1	1	3-1
	11	MOV.W	@Rm,Rn	LS	1	1	3-1
	12	MOV.L	@Rm,Rn	LS	1	1	3-1
	13	MOV.B	@Rm+,Rn	LS	1	1	3-1
	14	MOV.W	@Rm+,Rn	LS	1	1	3-1
	15	MOV.L	@Rm+,Rn	LS	1	1	3-1
	16	MOV.B	@(disp,Rm),R0	LS	1	1	3-1
	17	MOV.W	@(disp,Rm),R0	LS	1	1	3-1
	18	MOV.L	@(disp,Rm),Rn	LS	1	1	3-1
	19	MOV.B	@(R0,Rm),Rn	LS	1	1	3-1
	20	MOV.W	@(R0,Rm),Rn	LS	1	1	3-1
	21	MOV.L	@(R0,Rm),Rn	LS	1	1	3-1
	22	MOV.B	@(disp,GBR),R0	LS	1	1	3-1
	23	MOV.W	@(disp, GBR),R0	LS	1	1	3-1
	24	MOV.L	@(disp, GBR),R0	LS	1	1	3-1
	25	MOV.B	Rm,@Rn	LS	1	1	3-1
	26	MOV.W	Rm,@Rn	LS	1	1	3-1
	27	MOV.L	Rm,@Rn	LS	1	1	3-1
	28	MOV.B	Rm,@-Rn	LS	1	1	3-1
	29	MOV.W	Rm,@-Rn	LS	1	1	3-1
	30	MOV.L	Rm,@-Rn	LS	1	1	3-1
	31	MOV.B	R0,@(disp,Rn)	LS	1	1	3-1
	32	MOV.W	R0,@(disp,Rn)	LS	1	1	3-1
	33	MOV.L	Rm,@(disp,Rn)	LS	1	1	3-1
	34	MOV.B	Rm,@(R0,Rn)	LS	1	1	3-1
	35	MOV.W	Rm,@(R0,Rn)	LS	1	1	3-1

功能分类	No.	指令		指令组	发行速率	执行状态	执行模式
数据传送指令	36	MOV.L	Rm,@(R0,Rn)	LS	1	1	3-1
	37	MOV.B	R0,@(disp,GBR)	LS	1	1	3-1
	38	MOV.W	R0,@(disp,GBR)	LS	1	1	3-1
	39	MOV.L	R0,@(disp,GBR)	LS	1	1	3-1
	40	MOVCA.L	R0,@Rn	LS	1	1	3-4
	41	MOVCO.L	R0,@Rn	CO	1	1	3-9
	42	MOVLI.L	@Rm,R0	CO	1	1	3-8
	43	MOVUA.L	@Rm,R0	LS	2	2	3-10
	44	MOVUA.L	@Rm+,R0	LS	2	2	3-10
	45	MOVT	Rn	EX	1	1	2-1
	46	OCBI	@Rn	LS	1	1	3-4
	47	OCBP	@Rn	LS	1	1	3-4
	48	OCBWB	@Rn	LS	1	1	3-4
	49	PREF	@Rn	LS	1	1	3-4
	50	SWAP.B	Rm,Rn	EX	1	1	2-1
	51	SWAP.W	Rm,Rn	EX	1	1	2-1
	52	XTRCT	Rm,Rn	EX	1	1	2-1
定点算术指令	53	ADD	Rm,Rn	EX	1	1	2-1
	54	ADD	#imm,Rn	EX	1	1	2-1
	55	ADDC	Rm,Rn	EX	1	1	2-1
	56	ADDV	Rm,Rn	EX	1	1	2-1
	57	CMP/EQ	#imm,R0	EX	1	1	2-1
	58	CMP/EQ	Rm,Rn	EX	1	1	2-1
	59	CMP/GE	Rm,Rn	EX	1	1	2-1
	60	CMP/GT	Rm,Rn	EX	1	1	2-1
	61	CMP/HI	Rm,Rn	EX	1	1	2-1
	62	CMP/HS	Rm,Rn	EX	1	1	2-1
	63	CMP/PL	Rn	EX	1	1	2-1
	64	CMP/PZ	Rn	EX	1	1	2-1
	65	CMP/STR	Rm,Rn	EX	1	1	2-1
	66	DIV0S	Rm,Rn	EX	1	1	2-1
	67	DIV0U		EX	1	1	2-1
	68	DIV1	Rm,Rn	EX	1	1	2-1
	69	DMULS.L	Rm,Rn	EX	1	2	5-6
	70	DMULU.L	Rm,Rn	EX	1	2	5-6
	71	DT	Rn	EX	1	1	2-1
	72	MAC.L	@Rm+,@Rn+	CO	2	5	5-9

功能分类	No.	指令		指令组	发行速率	执行状态	执行模式
定点算术指令	73	MAC.W	@Rm+,@Rn+	CO	2	4	5-8
	74	MUL.L	Rm,Rn	EX	1	2	5-6
	75	MULS.W	Rm,Rn	EX	1	1	5-5
	76	MULU.W	Rm,Rn	EX	1	1	5-5
	77	NEG	Rm,Rn	EX	1	1	2-1
	78	NEGC	Rm,Rn	EX	1	1	2-1
	79	SUB	Rm,Rn	EX	1	1	2-1
	80	SUBC	Rm,Rn	EX	1	1	2-1
	81	SUBV	Rm,Rn	EX	1	1	2-1
逻辑指令	82	AND	Rm,Rn	EX	1	1	2-1
	83	AND	#imm,R0	EX	1	1	2-1
	84	AND.B	#imm,@(R0,GBR)	CO	3	3	3-2
	85	NOT	Rm,Rn	EX	1	1	2-1
	86	OR	Rm,Rn	EX	1	1	2-1
	87	OR	#imm,R0	EX	1	1	2-1
	88	OR.B	#imm,@(R0,GBR)	CO	3	3	3-2
	89	TAS.B	@Rn	CO	4	4	3-3
	90	TST	Rm,Rn	EX	1	1	2-1
	91	TST	#imm,R0	EX	1	1	2-1
	92	TST.B	#imm,@(R0,GBR)	CO	3	3	3-2
	93	XOR	Rm,Rn	EX	1	1	2-1
	94	XOR	#imm,R0	EX	1	1	2-1
	95	XOR.B	#imm,@(R0,GBR)	CO	3	3	3-2
移位指令	96	ROTL	Rn	EX	1	1	2-1
	97	ROTR	Rn	EX	1	1	2-1
	98	ROTCL	Rn	EX	1	1	2-1
	99	ROTCR	Rn	EX	1	1	2-1
	100	SHAD	Rm,Rn	EX	1	1	2-1
	101	SHAL	Rn	EX	1	1	2-1
	102	SHAR	Rn	EX	1	1	2-1
	103	SHLD	Rm,Rn	EX	1	1	2-1
	104	SHLL	Rn	EX	1	1	2-1
	105	SHLL2	Rn	EX	1	1	2-1
	106	SHLL8	Rn	EX	1	1	2-1
	107	SHLL16	Rn	EX	1	1	2-1
	108	SHLR	Rn	EX	1	1	2-1
	109	SHLR2	Rn	EX	1	1	2-1

功能分类	No.	指令		指令组	发行速率	执行状态	执行模式
移位指令	110	SHLR8	Rn	EX	1	1	2-1
	111	SHLR16	Rn	EX	1	1	2-1
转移指令	112	BF	disp	BR	1+0 ～ 2	1	1-1
	113	BF/S	disp	BR	1+0 ～ 2	1	1-1
	114	BT	disp	BR	1+0 ～ 2	1	1-1
	115	BT/S	disp	BR	1+0 ～ 2	1	1-1
	116	BRA	disp	BR	1+0 ～ 2	1	1-1
	117	BRAF	Rm	BR	1+3	1	1-2
	118	BSR	disp	BR	1+0 ～ 2	1	1-1
	119	BSRF	Rm	BR	1+3	1	1-2
	120	JMP	@Rn	BR	1+3	1	1-2
	121	JSR	@Rn	BR	1+3	1	1-2
	122	RTS		BR	1+0 ～ 3	1	1-3
系统控制指令	123	NOP		MT	1	1	2-3
	124	CLRMAC		EX	1	1	5-7
	125	CLRS		EX	1	1	2-1
	126	CLRT		EX	1	1	2-1
	127	ICBI	@Rn	CO	8+5+3	13	3-6
	128	SETS		EX	1	1	2-1
	129	SETT		EX	1	1	2-1
	130	PREFI		CO	5+5+3	10	3-7
	131	SYNCO	@Rn	CO	不定	不定	3-4
	132	TRAPA	#imm	CO	8+5+1	13	1-5
	133	RTE		CO	4+1	4	1-4
	134	SLEEP		CO	不定	不定	1-6
	135	LDTLB		CO	1	1	3-5
	136	LDC	Rm,DBR	CO	4	4	4-2
	137	LDC	Rm,SGR	CO	4	4	4-2
	138	LDC	Rm,GBR	LS	1	1	4-3
	139	LDC	Rm,Rp_BANK	LS	1	1	4-1
	140	LDC	Rm,SR	CO	4+3	4	4-4
	141	LDC	Rm,SSR	LS	1	1	4-1
	142	LDC	Rm,SPC	LS	1	1	4-1
	143	LDC	Rm,VBR	LS	1	1	4-1
	144	LDC.L	@Rm+,DBR	CO	4	4	4-6
	145	LDC.L	@Rm+,SGR	CO	4	4	4-6

功能分类	No.	指令		指令组	发行速率	执行状态	执行模式
系统控制指令	146	LDC.L	@Rm+,GBR	LS	1	1	4-7
	147	LDC.L	@Rm+,Rp_BANK	LS	1	1	4-5
	148	LDC.L	@Rm+,SR	CO	6+3	4	4-8
	149	LDC.L	@Rm+,SSR	LS	1	1	4-5
	150	LDC.L	@Rm+,SPC	LS	1	1	4-5
	151	LDC.L	@Rm+,VBR	LS	1	1	4-5
	152	LDS	Rm,MACH	LS	1	1	5-1
	153	LDS	Rm,MACL	LS	1	1	5-1
	154	LDS	Rm,PR	LS	1	1	4-13
	155	LDC.L	@Rm+,MACH	LS	1	1	5-2
	156	LDC.L	@Rm+,MACL	LS	1	1	5-2
	157	LDC.L	@Rm+,PR	LS	1	1	4-14
	158	STC	DBR,Rn	LS	1	1	4-9
	159	STC	SGR,Rn	LS	1	1	4-9
	160	STC	GBR,Rn	LS	1	1	4-9
	161	STC	Rp_BANK,Rn	LS	1	1	4-9
	162	STC	SR,Rn	CO	1	1	4-10
	163	STC	SSR,Rn	LS	1	1	4-9
	164	STC	SPC,Rn	LS	1	1	4-9
	165	STC	VBR,Rn	LS	1	1	4-9
	166	STC.L	DBR,@-Rn	LS	1	1	4-11
	167	STC.L	SGR,@-Rn	LS	1	1	4-11
	168	STC.L	GBR,@-Rn	LS	1	1	4-11
	169	STC.L	Rp_BANK,@-Rn	LS	1	1	4-11
	170	STC.L	SR,@-Rn	CO	1	1	4-12
	171	STC.L	SSR,@-Rn	LS	1	1	4-11
	172	STC.L	SPC,@-Rn	LS	1	1	4-11
	173	STC.L	VBR,@-Rn	LS	1	1	4-11
	174	STS	MACH,Rn	LS	1	1	5-3
	175	STS	MACL,Rn	LS	1	1	5-3
	176	STS	PR,Rn	LS	1	1	4-15
	177	STC.L	MACH,@-Rn	LS	1	1	5-4
	178	STC.L	MACL,@-Rn	LS	1	1	5-4
	179	STC.L	PR,@-Rn	LS	1	1	4-16
单精度浮点指令	180	FLDI0	FRn	LS	1	1	6-13
	181	FLDI1	FRn	LS	1	1	6-13
	182	FMOV	FRm,FRn	LS	1	1	6-9

功能分类	No.	指令		指令组	发行速率	执行状态	执行模式
单精度浮点指令	183	FMOV.S	@Rm,FRn	LS	1	1	6-9
	184	FMOV.S	@Rm+,FRn	LS	1	1	6-9
	185	FMOV.S	@(R0,Rm),FRn	LS	1	1	6-9
	186	FMOV.S	FRm,@Rn	LS	1	1	6-9
	187	FMOV.S	FRm,@-Rn	LS	1	1	6-9
	188	FMOV.S	FRm,@(R0,Rn)	LS	1	1	6-9
	189	FLDS	FRm,FPUL	LS	1	1	6-10
	190	FSTS	FPUL,FRn	LS	1	1	6-11
	191	FABS	FRn	LS	1	1	6-12
	192	FADD	FRm,FRn	FE	1	1	6-14
	193	FCMP/EQ	FRm,FRn	FE	1	1	6-14
	194	FCMP/GT	FRm,FRn	FE	1	1	6-14
	195	FDIV	FRm,FRn	FE	1	14	6-15
	196	FLOAT	FPUL,FRn	FE	1	1	6-14
	197	FMAC	FR0,FRm,FRn	FE	1	1	6-14
	198	FMUL	FRm,FRn	FE	1	1	6-14
	199	FNEG	FRn	LS	1	1	6-12
	200	FSQRT	FRn	FE	1	30	6-15
	201	FSUB	FRm,FRn	FE	1	1	6-14
	202	FTRC	FRm,FPUL	FE	1	1	6-14
	203	FMOV	DRm,DRn	LS	1	1	6-9
	204	FMOV	@Rm,DRn	LS	1	1	6-9
	205	FMOV	@Rm+,DRn	LS	1	1	6-9
	206	FMOV	@(R0,Rm),DRn	LS	1	1	6-9
	207	FMOV	DRm,@Rn	LS	1	1	6-9
	208	FMOV	DRm,@-Rn	LS	1	1	6-9
	209	FMOV	DRm,@(R0,Rn)	LS	1	1	6-9
双精度浮点指令	210	FABS	DRn	LS	1	1	6-12
	211	FADD	DRm,DRn	FE	1	1	6-16
	212	FCMP/EQ	DRm,DRn	FE	1	1	6-16
	213	FCMP/GT	DRm,DRn	FE	1	1	6-16
	214	FCNVDS	DRm,FPUL	FE	1	1	6-16
	215	FCNVSD	FPUL,DRn	FE	1	1	6-16
	216	FDIV	DRm,DRn	FE	1	14	6-18
	217	FLOAT	FPUL,DRn	FE	1	1	6-16

功能分类	No.	指令		指令组	发行速率	执行状态	执行模式
双精度浮点指令	218	FMUL	DRm,DRn	FE	1	3	6-17
	219	FNEG	DRn	LS	1	1	6-12
	220	FSQRT	DRn	FE	1	30	6-18
	221	FSUB	DRm,DRn	FE	1	1	6-16
	222	FTRC	DRm,FPUL	FE	1	1	6-16
FPU 系统控制指令	223	LDS	Rm,FPUL	LS	1	1	6-1
	224	LDS	Rm,FPSCR	LS	1	1	6-5
	225	LDS.L	@Rm+,FPUL	LS	1	1	6-3
	226	LDS.L	@Rm+,FPSCR	LS	1	1	6-7
	227	STS	FPUL,Rn	LS	1	1	6-2
	228	STS	FPSCR,Rn	LS	1	1	6-6
	229	STS.L	FPUL,@-Rn	LS	1	1	6-4
	230	STS.L	FPSCR,@-Rn	LS	1	1	6-8
图形强化指令	231	FMOV	DRm,XDn	LS	1	1	6-9
	232	FMOV	XDm,DRn	LS	1	1	6-9
	233	FMOV	XDm,XDn	LS	1	1	6-9
	234	FMOV	@Rm,XDn	LS	1	1	6-9
	235	FMOV	@Rm+,XDn	LS	1	1	6-9
	236	FMOV	@(R0,Rm),XDn	LS	1	1	6-9
	237	FMOV	XDm,@Rn	LS	1	1	6-9
	238	FMOV	XDm,@-Rn	LS	1	1	6-9
	239	FMOV	XDm,@(R0,Rn)	LS	1	1	6-9
	240	FIPR	FVm,FVn	FE	1	1	6-19
	241	FRCHG		FE	1	1	6-14
	242	FSCHG		FE	1	1	6-14
	243	FPCHG		FE	1	1	6-14
	244	FSRRA	FRn	FE	1	1	6-21
	245	FSCA	FPUL,DRn	FE	1	3	6-22
	246	FTRV	XMTRX,FVn	FE	1	4	6-20

第 5 章 异常处理

5.1 概要

异常处理就是在检测出复位、普通异常、中断时，采用与通常不同的程序进行必要的处理。例如：执行中的指令发生异常结束时，需要一个控制功能，此功能通过适当的处理使之返回原程序或报告异常并结束异常。为支持这样的功能，对于异常结束，产生异常处理请求，将控制流程传送给用户作成的异常处理程序等统称为异常处理。

SH-4A 的异常处理分为复位、普通异常、中断 3 种。

5.2 寄存器说明

异常处理相关的寄存器结构如表 5.1 所示。

表 5.1 寄存器结构

名称	简称	R/W	P4 区域地址 *	区域 7 地址 *	存取长度
TRAPA 异常寄存器	TRA	R/W	H'FF00 0020	H'1F00 0020	32
异常事件寄存器	EXPEVT	R/W	H'FF00 0024	H'1F00 0024	32
中断事件寄存器	INTEVT	R/W	H'FF00 0028	H'1F00 0028	32

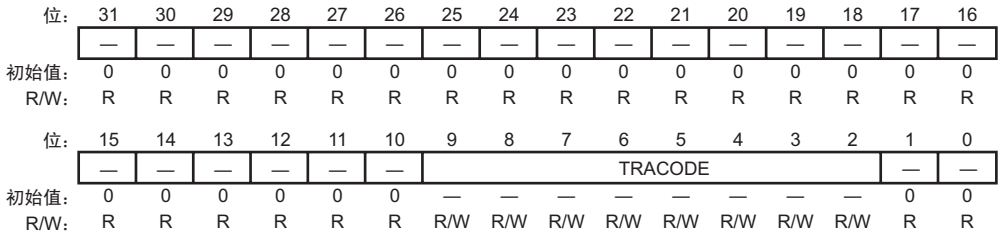
【注】 * P4 区域地址为使用虚拟地址空间的 P4 区域的地址。区域 7 地址为使用 TLB 从物理地址空间的区域 7 进行存取的地址。

表 5.2 各处理模式下的寄存器状态

名称	简称	上电复位	手动复位	睡眠	待机
TRAPA 异常寄存器	TRA	不定	不定	保持	保持
异常事件寄存器	EXPEVT	H'0000 0000	H'0000 0020	保持	保持
中断事件寄存器	INTEVT	不定	不定	保持	保持

5.2.1 TRAPA 异常寄存器 （TRA）

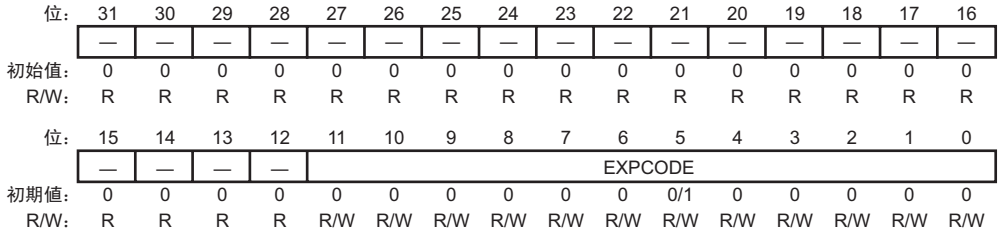
TRAPA 异常寄存器 （TRA）为设定 TRAPA 指令的 8 位立即数 （imm）的寄存器。执行 TRAPA 指令时通过硬件自动设定 TRA。 TRA 也可通过软件来变更。



位	位名称	初始值	R/W	说明
31 ～ 10	—	均为 0	R	保留位 有关本位的读取 / 写入请参考 “产品使用时的注意事项”。
9 ～ 2	TRACODE	不定	R/W	TRAPA 码 设定 TRAPA 指令的 8 位立即数。
1、 0	—	均为 0	R	保留位 有关本位的读取 / 写入请参考 “产品使用时的注意事项”。

5.2.2 异常事件寄存器 （EXPEVT）

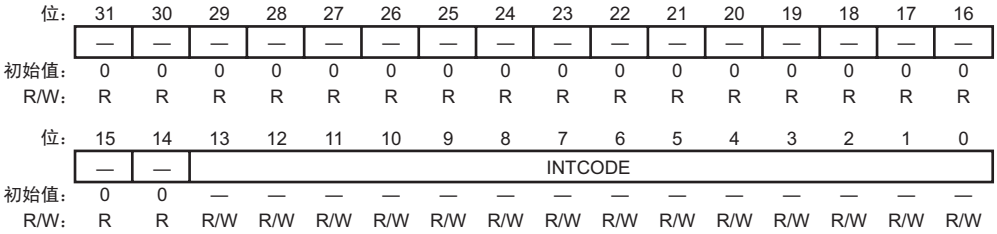
异常事件寄存器 （EXPEVT）设定由 12 位的复位与普通异常事件产生的异常码。接受异常时，通过硬件自动设定异常码。 EXPEVT 也可通过软件来变更。



位	位名称	初始值	R/W	说明
31 ～ 12	—	均为 0	R	保留位 有关本位的读取 / 写入请参考 “产品使用时的注意事项”。
11 ～ 0	EXPCODE	H'000 或 H'020	R/W	异常码 设定复位、普通异常的异常码。详细内容请参考表 5.3。

5.2.3 中断事件寄存器 （INTEVT）

中断事件寄存器 （INTEVT）设定由中断请求产生的 14 位异常码。接受异常时通过硬件自动设定异常码。INTEVT 也可通过软件来变更。



位	位名称	初始值	R/W	说明
31 ～ 14	—	均为 0	R	保留位 有关本位的读取 / 写入请参考 “产品使用时的注意事项”。
13 ～ 0	INTCODE	不定	R/W	异常码 设定中断的异常码。详细内容请参考表 5.3。

5.3 异常处理功能

5.3.1 异常处理的流程

异常处理中，程序计数器 （PC）、状态寄存器 （SR）、 R15 的内容分别保存至保存程序计数器 （SPC）、保存状态寄存器 （SSR）、保存通用寄存器 （SGR），根据向量地址开始执行对应的异常处理程序。所谓异常处理程序就是由用户根据各个异常内容编成的程序。为了结束异常处理程序、返回原程序，执行从异常处理返回指令 （RTE）。通过此指令，恢复 PC 与 SR 的内容，发生异常等时可返回正常处理程序。另外，用 RTE 指令不能将 SGR 的内容回写至 R15。

基本的异常处理流程如下。 SR 位含义的详细内容请参考 “第 2 章 编程模型”。

1. PC、 SR及R15的内容分别保存至SPC、 SSR及SGR。
2. SR的阻塞位 （BL）置1。
3. SR的模式位 （MD）置1。
4. SR的寄存器存储体位 （RB）置1。
5. 复位时， SR的FPU禁止位 （FD）置0。
6. 将异常码写入异常源的异常事件寄存器 （EXPEVT）或中断事件寄存器 （INTEVT）的bit13～0。
7. 转移至已确定的异常处理的向量地址，开始异常处理程序。

5.3.2 异常处理向量地址

复位向量地址固定为 H'A000 0000。异常、中断的向量地址为向量基址加上各事件偏移值的地址。通过软件将向量基址设定至向量基址寄存器 （VBR）。例如，因为 TLB 未命中异常的偏移为 H'0000 0400，所以如果预先将 H'9C08 0000 设定至 VBR，则异常处理向量地址为 H'9C08 0400。在异常处理向量地址再次发生异常时，为双重异常，难以恢复，因此向量地址请指定非地址转换对象的 P1、 P2 区域的地址。

5.4 异常的种类与优先顺序

异常的种类、优先顺序、向量地址及异常 / 中断码如**表 5.3** 所示。

表 5.3 异常一览表

异常分类	执行形态	异常	优先级	优先顺序	异常转移目标		异常码
					向量基址	偏移量	
复位	中断型	上电复位	1	1	H'A000 0000	—	H'000
		手动复位	1	2	H'A000 0000	—	H'020
		H-UDI 复位	1	1	H'A000 0000	—	H'000
		指令 TLB 多重命中异常	1	3	H'A000 0000	—	H'140
		数据 TLB 多重命中异常	1	4	H'A000 0000	—	H'140
普通异常	再执行型	指令执行前用户中断 *	2	0	（VBR/DBR）	H'100/—	H'1E0
		指令地址错误	2	1	（VBR）	H'100	H'0E0
		指令 TLB 未命中异常	2	2	（VBR）	H'400	H'040
		指令 TLB 保护违反异常	2	3	（VBR）	H'100	H'0A0
		普通非法指令异常	2	4	（VBR）	H'100	H'180
		槽非法指令异常	2	4	（VBR）	H'100	H'1A0
		普通 FPU 禁止异常	2	4	（VBR）	H'100	H'800
		槽 FPU 禁止异常	2	4	（VBR）	H'100	H'820
		数据地址错误 （读取）	2	5	（VBR）	H'100	H'0E0
		数据地址错误 （写入）	2	5	（VBR）	H'100	H'100
		数据 TLB 未命中异常 （读取）	2	6	（VBR）	H'400	H'040
		数据 TLB 未命中异常 （写入）	2	6	（VBR）	H'400	H'060
		数据 TLB 保护违反异常 （读取）	2	7	（VBR）	H'100	H'0A0
		数据 TLB 保护违反异常 （写入）	2	7	（VBR）	H'100	H'0C0
		FPU 异常	2	8	（VBR）	H'100	H'120
		初始页写入异常	2	9	（VBR）	H'100	H'080
	完成型	无条件陷阱 （TRAPA）	2	4	（VBR）	H'100	H'160
		指令执行后用户中断 *	2	10	（VBR/DBR）	H'100/—	H'1E0
中断	完成型	非屏蔽中断	3	—	（VBR）	H'600	H'1C0
		普通中断请求	4	—	（VBR）	H'600	—

优先权： 首先按优先级排序，同一级别内按优先顺序排序 （较小的数值优先权高）。

异常转移目标： 复位时控制转移至 H'A000 0000，其它情况下控制转移至 （VBR+ 偏移）。

异常码： 复位、普通异常时保存至 EXPEVT，中断时保存至 INTEVT。

【注】 * CBCR.UBDE=1 时 PC=DBR。其它情况下 PC=VBR+H'100

5.5 异常流程

5.5.1 异常流程

指令执行与异常处理的基本运行大致如**图 5.1** 所示。在此为便于说明，以逐个执行指令为基础进行说明。异常类别 （复位、普通异常、中断）间的优先顺序如**图 5.1** 所示。**图 5.1** 中，异常成立时的寄存器设定仅限 SSR、SPC、SGR、EXPEVT/INTEVT、SR、及 PC，但也有根据异常通过硬件被自动设定的寄存器。详细内容请参考 **“5.6 各异常的说明”**。另外，有关延迟转移指令与延迟槽指令执行中的异常处理及产生 2 次数据存取的指令，请参考 **“5.6.4 发生多次异常时的优先顺序”**。

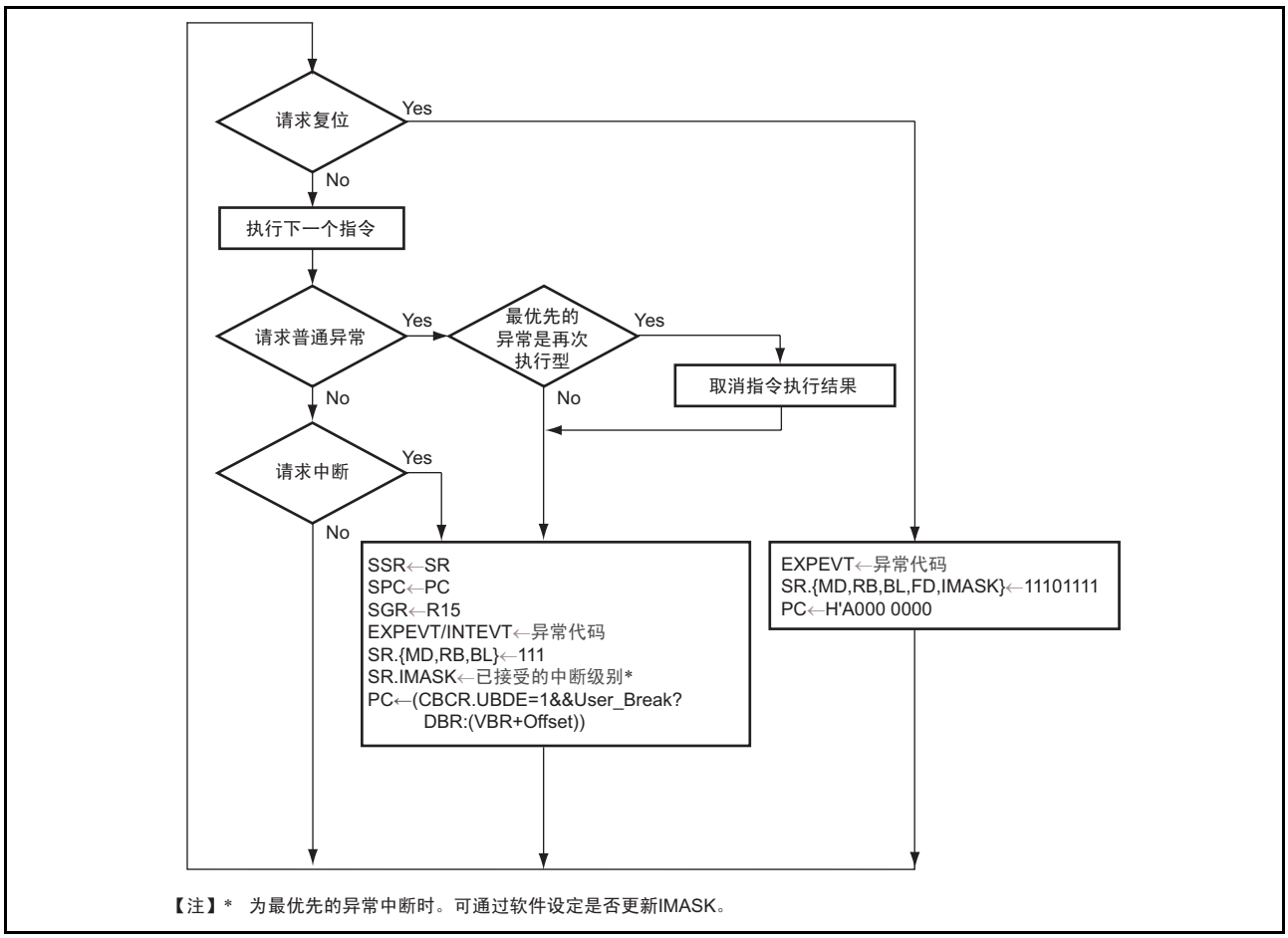


图 5.1 指令执行与异常处理

5.5.2 接受异常源

同时发生至少 2 个异常时，为决定所接受的异常而给所有的异常确定优先顺序。普通异常中的 5 种异常：普通非法指令异常、槽非法指令异常、普通 FPU 禁止异常、槽 FPU 禁止异常、无条件陷阱异常是在各自的指令分析过程中被检测出、在指令流水线中不同时发生的异常。因此优先顺序为相同值。普通异常是通过指令执行顺序检测出的，但是，异常处理则依据指令流程顺序（程序顺序）进行处理。即：与后续指令的异常相比优先接受先前指令的异常。普通异常的接受顺序例如图 5.2 所示。

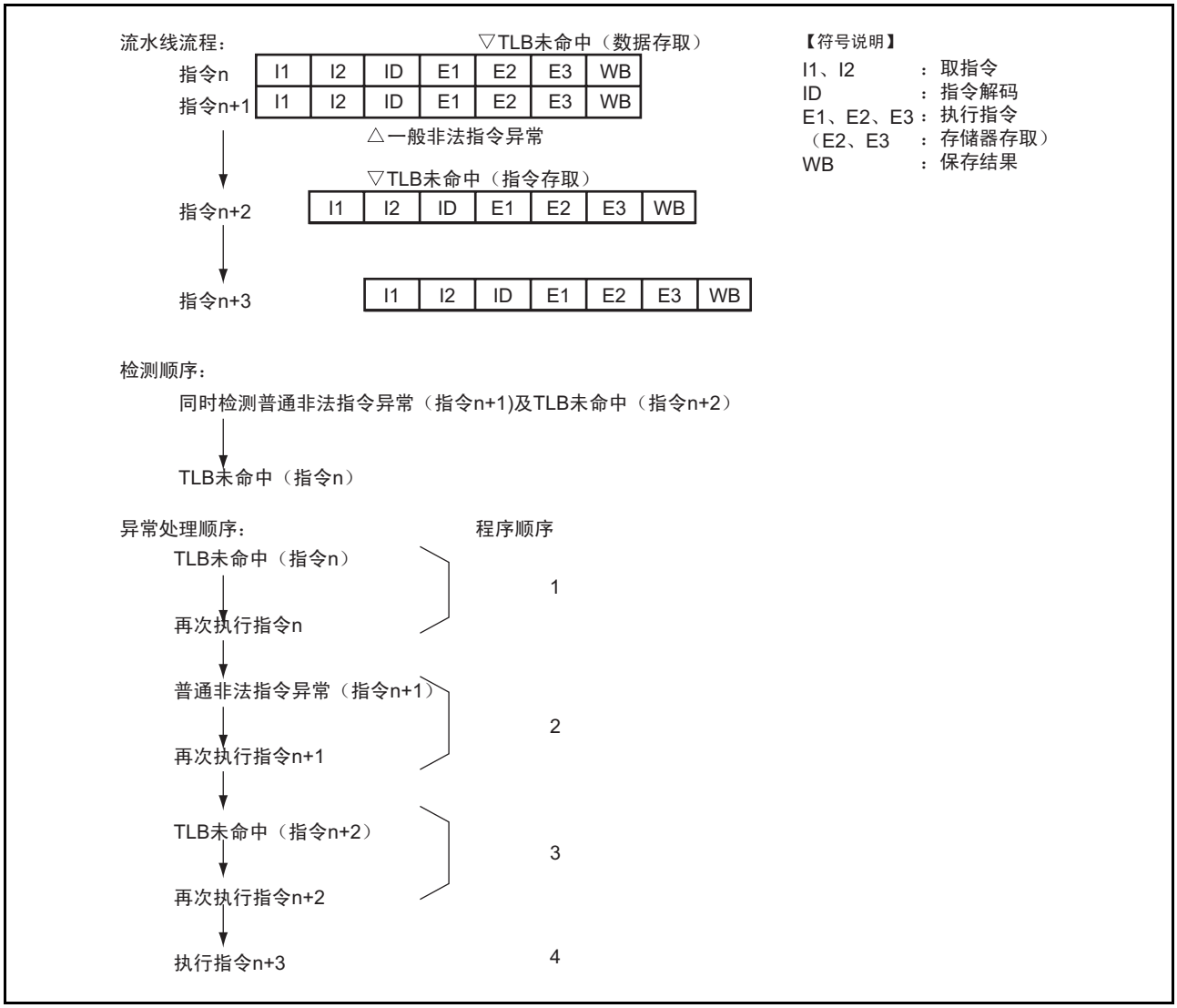


图 5.2 普通异常的接受序列

5.5.3 异常请求与 BL 位

SR 的 BL 位为 0 时，接受异常、中断。

SR 的 BL 位为 1 时，如果发生除用户中断之外的异常， CPU 内部寄存器、其他模块的寄存器变为手动复位后的状态，并转移至与复位相同的地址 （H'A000 0000）。产生用户中断时的运行请参考该产品硬件手册的“用户中断控制器”。另外，产生普通中断时，保留中断请求，通过软件将 BL 位清 0 后接受中断请求。产生非屏蔽中断 （NMI）时，可通过软件设定保留或接受。

这样，通常为了可多重接受异常状态，保存 SPC 与 SSR，此后将 SR 的 BL 位清 0。

5.5.4 从异常处理返回

使用 RTE 指令从异常处理返回。通过 RTE 指令， SPC 恢复至 PC， SSR 恢复至 SR，转移至 SPC 的地址后，从异常处理返回。将 SPC、 SSR 保存至存储器时，请在将 SR 的 BL 位置 1 后，恢复 SPC 与 SSR，并发行 RTE 指令。

5.6 各异常的说明

各个异常处理运行说明了发生源、发生时的转移目标地址、转移时的处理器运行。

5.6.1 复位

(1) 上电复位

- 条件：
  上电复位请求
- 运行：
  将H'000设定至EXPEVT，初始化CPU及内部外围模块后，转移至复位向量 （H'A0000000）。详细内容请参考各章的寄存器说明。通电时必须进行上电复位。

(2) 手动复位

- 条件：
  手动复位请求
- 运行：
  将H'020设定至EXPEVT，初始化CPU及内部外围模块后，转移至复位向量 （H'A0000000）。上电复位与手动复位中初始化的寄存器是不同的。详细内容请参考各章的寄存器说明。

(3) H-UDI 复位

- 源：SDIR. TI[7:4]为B'0110 （否定）或B'0111 （指定）
- 转移目标地址：H'A000 0000
- 转移时的运行：
  将异常码H'000设定至EXPEVT。初始化VBR、SR，并转移至PC=H'A000 0000。
  初始化CPU及内部外围模块。详细内容请参考硬件手册各章的寄存器说明。

(4) 指令 TLB 多重命中异常

- 源：ITLB地址多重匹配
- 转移目标地址：H'A000 0000
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将异常码H'140设定至EXPEVT。初始化VBR、 SR，并转移至PC=H'A000 0000。
  与手动复位时相同，进行CPU及内部外围模块的初始化、详细内容请参考硬件手册各章的寄存器说明。

(5) 数据 TLB 多重命中异常

- 源：UTLB地址多重匹配
- 转移目标地址：H'A000 0000
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将异常码H'140设定至EXPEVT。初始化VBR、SR，并转移至PC=H'A000 0000。与手动复位时相同，进行CPU及内部外围模块的初始化。详细内容请参考硬件手册各章的寄存器说明。

5.6.2 普通异常

(1) 数据 TLB 未命中异常

- 源：UTLB的地址比较结果，地址不匹配
- 转移目标地址：VBR + H'0000 0400
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  读取时将异常码H'040设定至EXPEVT，写入时将异常码H'060设定至EXPEVT。将SR的BL位、 MD位、 RB位置1并转移至PC=VBR+H'0400。
  为使TLB未命中处理高速化，区分其他异常与偏移量。

```
Data_TLB_miss_exception()
{
            TEA=EXCEPTION_ADDRESS;
            PTEH.VPN=PAGE_NUMBER;
             SPC=PC;
             SSR=SR;
             SGR=R15;
             EXPEVT= read_access ? H'00000040 : H'00000060;
             SR.MD=1;
             SR.RB=1;
             SR.BL=1;
             PC=VBR + H'00000400;
}
```

(2) 指令 TLB 未命中异常

- 源：ITLB地址比较的结果，地址不匹配
- 转移目标地址：VBR + H'0000 0400
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。将异常码H'040设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0400。
  为使TLB未命中处理高速化，区分其他异常与偏移量。

```
ITLB_miss_exception()
{
            TEA=EXCEPTION_ADDRESS;
            PTEH.VPN=PAGE_NUMBER;
            SPC=PC;
            SSR=SR;
            SGR=R15;
            EXPEVT=H'00000040;
            SR.MD=1;
            SR.RB=1;
            SR.BL=1;
            PC=VBR + H'00000400;
}
```

(3) 初始页写入异常

- 源：在存储存取中命中TLB，但脏位D=0
- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将发生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。将异常码H'080设定至EXPEVT。
  将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。

```
Initial_write_exception()
{
            TEA=EXCEPTION_ADDRESS;
            PTEH.VPN=PAGE_NUMBER;
            SPC=PC;
            SSR=SR;
            SGR=R15;
            EXPEVT=H'00000080;
            SR.MD=1;
            SR.RB=1;
            SR.BL=1;
            PC=VBR + H'00000100;
}
```

(4) 数据 TLB 保护违反异常

- 源：存取违反了以下所示的UTLB保护信息 （PR位）。

PR	特权模式	用户模式
00	仅可读取	不可存取
01	可读取 / 写入	不可存取
10	仅可读取	仅可读取
11	可读取 / 写入	可读取 / 写入

- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  读取时将异常码H'0A0设定至EXPEVT，写入时将异常码H'0C0设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。

```
Data_TLB_protection_violation_exception()
{
            TEA=EXCEPTION_ADDRESS;
            PTEH.VPN=PAGE_NUMBER;
            SPC=PC;
            SSR=SR;
            SGR=R15;
            EXPEVT=read_access ? H'000000A0 : H'000000C0;
            SR.MD=1;
            SR.RB=1;
            SR.BL=1;
            PC=VBR + H'00000100;
}
```

(5) 指令 TLB 保护违反异常

- 源：存取违反了以下所示的ITLB保护信息 （PR位）。

PR	特权模式	用户模式
0	可存取	不可存取
1	可存取	可存取

- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时的R15保存至SGR。
  将异常码H'0A0设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。

```
ITLB_protection_violation_exception()
{
            TEA=EXCEPTION_ADDRESS;
            PTEH.VPN=PAGE_NUMBER;
            SPC=PC;
            SSR=SR;
            SGR=R15;
            EXPEVT=H'000000A0;
            SR.MD=1;
            SR.RB=1;
            SR.BL=1;
            PC=VBR + H'00000100;
    }
```

(6) 数据地址错误

- 源：
  - 从字边界以外 （2n+1）存取字数据
  - 从长字数据边界以外 （4n+1、 4n+2、 4n+3）存取长字数据
  - 从四字数据边界以外 （8n+1、 8n+2、 8n+3、 8n+4、 8n+5、 8n+6、 8n+7）存取四字
  - 存取用户模式下的区域H'8000 0000～H'FFFF FFFF
    但是，可设定分别从用户模式存取H'E000 0000～H'E3FF FFFF及H'E500 0000～H'E5FF FFFF。详细内容请参考 “第7章 存储器管理单元 （MMU）”及 “第9章 L存储器”。
- 转移目标地址：VBR+ H'0000 0100
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  读取时将异常码H'0E0设定至EXPEVT，写入时将异常码H'100设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。详细内容请参考 “第7章 存储器管理单元 （MMU）”。

```
Data_address_error()
{
            TEA=EXCEPTION_ADDRESS;
            PTEH.VPN=PAGE_NUMBER;
            SPC=PC;
            SSR=SR;
            SGR=R15;
            EXPEVT=read_access? H'000000E0: H'00000100;
            SR.MD=1;
            SR.RB=1;
            SR.BL=1;
            PC=VBR + H'00000100;
}
```

(7) 指令地址错误

- 源：
  — 从字边界以外 （2n+1）取指令
  — 从用户模式下的区域H'8000 0000～H'FFFF FFFF取指令
  — 但是，可设定从用户模式存取H'E500 0000～H'E5FF FFFF。详细内容请参考 “第9章 L存储器”。
- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将产生此异常的虚拟地址 （32位）设定至TEA，将对应的虚拟页码 （22位）设定至PTEH[31:10]。
  PTEH的ASID表示发生此异常时的ASID。
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  将异常码H'0E0设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。详细内容请参考 “第7章 存储器管理单元 （MMU）”。

```
Instruction_address_error()
{
            TEA = EXCEPTION_ADDRESS;
            PTEH.VPN = PAGE_NUMBER;
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'000000E0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(8) 无条件陷阱

- 源：执行TRAPA指令
- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  由于是处理完成型异常，将TRAPA指令的下一个指令的PC保存至SPC。执行TRAPA指令时的SR、R15保存至SSR、 SGR。将TRAPA指令中的8位立即数乘以4，并设定至TRA[9:0]。
  将异常码H'160设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC＝VBR+H'0100。

```
TRAPA_exception()
{
            SPC = PC + 2;
            SSR = SR;
            SGR = R15;
            TRA = imm << 2;
            EXPEVT = H'00000160;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(9) 普通非法指令异常

- 源：
  — 对延迟槽以外的未定义指令进行解码
  延迟转移指令：JMP、 JSR、 BRA、 BRAF、 BSR、 BSRF、 RTS、 RTE、 BT/S、 BF/S
  未定义指令：H'FFFD
  — 在用户模式下对延迟槽以外的特权指令进行解码
  特权指令：LDC、 STC、 RTE、 LDTLB、 SLEEP、
  但是，除了以LDC、 STC存取GBR的指令
- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  将异常码H'180设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。另外，解码H'FFFD以外的未定义码时不保证运行。

```
General_illegal_instruction_exception()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000180;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(10) 槽非法指令异常

- 源：
  — 对延迟槽中的未定义指令进行解码
  延迟转移指令：JMP、 JSR、 BRA、 BRAF、 BSR、 BSRF、 RTS、 RTE、 BT/S、 BF/S
  未定义指令：H'FFFD
  — 对改写延迟槽内PC的指令进行解码
  改写PC的指令：JMP、JSR、BRA、BRAF、BSR、BSRF、RTS、RTE、BT、BF、BT/S、BF/S、
  TRAPA、 LDC Rm,SR、 LDC.L @Rm+,SR、 ICBI、 PREFI
  — 在用户模式下对延迟槽内的特权指令进行解码
  特权指令：LDC、 STC、 RTE、 LDTLB、 SLEEP
  但是，除了以LDC、 STC存取GBR的指令
  — 对延迟槽内的PC相对MOV指令、 MOVA指令进行解码
- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将之前的延迟转移指令的PC保存至SPC。发生此异常时的SR、 R15保存至SSR、 SGR。
  将异常码H'1A0设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC＝VBR+H'0100。
  另外，解码H'FFFD以外的未定义指令时不保证运行。

```
Slot_illegal_instruction_exception()
{
            SPC = PC - 2;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'000001A0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(11) 普通 FPU 禁止异常

- 源：在SR.FD=1时对延迟槽以外的FPU指令*进行解码
- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  将异常码H'800设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。

```
General_fpu_disable_exception()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000800;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

【注】 * 所谓 FPU 指令就是操作码的最初 4 位为 F 的指令 （但是，除了未定义指令 H'FFFD）与对于 FPUL、 FPSCR 的 LDS、 STS、 LDS.L、 STS.L 指令。

(12) 槽 FPU 禁止异常

- 源：在SR.FD=1时对延迟槽中的FPU指令进行解码
- 转移目标地址：VBR + H'0000 0100
- 转移时的运行：
  将之前的延迟转移指令的PC保存至SPC。发生此异常时的SR、 R15保存至SSR、 SGR。
  将异常码H'820设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。

```
Slot_fpu_disable_exception()
{
            SPC = PC - 2;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000820;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

(13) 指令执行前用户中断 / 指令执行后用户中断

- 源：用户断点控制器中设定的中断条件成立
- 转移目标地址：VBR + H'0000 0100或DBR
- 转移时的运行：
  为指令执行后中断时，将设定断点的指令之后的指令的PC保存至SPC。为指令执行前中断时，将设定断点的指令的PC保存至SPC。
  产生中断时的SR、 R15保存至SSR、 SGR。将异常码H'1E0设定至EXPEVT。
  将SR的BL位、 MD位、 RB位置1，并转移至PC＝VBR+H'0100。但是，也可转移至PC=DBR。
  有关设定数据中断时的PC等详细内容请参考该产品硬件手册的 “**用户断点控制器**”。

```
User_break_exception()
{
            SPC = (pre_execution break? PC : PC + 2);
            SSR = SR;
            SGR = R15;
            EXPEVT = H'000001E0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = (CBCR.UBDE==1 ? DBR : VBR + H'00000100);
}
```

(14) FPU 异常

- 源：执行浮点运算引发的异常
- 转移目标地址：VBR+ H'0000 0100
- 转移时的运行：
  将产生此异常的指令的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  将异常码H'120设定至EXPEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0100。

```
FPU_exception()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            EXPEVT = H'00000120;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            PC = VBR + H'00000100;
}
```

5.6.3 中断

(1) NMI（非屏蔽中断）

- 源：NMI引脚的边沿检测
- 转移目标地址：VBR+H'0000 0600
- 转移时的运行：
  将接受此中断的指令之后的PC、 SR分别保存至SPC、 SSR，此时将R15保存至SGR。
  将异常码H'1C0设定至INTEVT。将SR的BL位、 MD位、 RB位置1，并转移至PC=VBR+H'0600。 SR的BL位为0时不通过SR的中断屏蔽位屏蔽此中断，而是被最优先接受。 SR的BL位为1时，可通过软件设定屏蔽此中断或接受此中断。详细内容请参考该产品硬件手册的 “中断控制器”。

```
NMI()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            INTEVT = H'000001C0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            If(cond)SR.IMASK = B'1111;
            PC = VBR + H'00000600;
}
```

(2) 普通中断请求

- 源：
  SR的中断屏蔽位比中断请求的中断级别低，并且SR的BL为0 （在指令的断开处接受）。
- 转移目标地址：VBR+ H'0000 0600
- 转移时的运行：
  将已接受的指令之后的PC设定至SPC。接受时的SR、 R15设定至SSR、 SGR。
  将对应各中断源的代码设定至INTEVT。将SR的BL位、 MD位、 RB位置1，并转移至VBR+H'0600。
  详细内容请参考该产品硬件手册的 “中断控制器”。

```
Module_interruption()
{
            SPC = PC;
            SSR = SR;
            SGR = R15;
            INTEVT = H'00000400 ~ H'00003FE0;
            SR.MD = 1;
            SR.RB = 1;
            SR.BL = 1;
            if(cond)SR.IMASK = level_of_accepted_interrupt();
            PC = VBR + H'00000600;
}
```

5.6.4 发生多次异常时的优先顺序

在对存储器进行 2 次存取的指令、不可分的带延迟转移指令及延迟槽指令等，发生多次异常。此时，由于与普通的异常优先顺序不同，所以必须注意。

(1) 对存储器进行 2 次存取的指令

虽然 MAC 指令、存储器与存储器间的逻辑运算指令、 TAS 指令、 MOVUA 指令是 1 条指令，但是有 2 次数据传送，所以在各自的数据传送时检测出异常。因此，用以下顺序进行判断。

1. 第1次数据传送的数据地址错误
2. 第1次数据传送的TLB未命中
3. 第1次数据传送的TLB保护违反
4. 第1次数据传送的初始页写入异常
5. 第2次数据传送的数据地址错误
6. 第2次数据传送的TLB未命中
7. 第2次数据传送的TLB保护违反
8. 第2次数据传送的初始页写入异常

(2) 不可分的带延迟的转移指令与延迟槽指令

由于带延迟的转移指令与延迟槽指令为不可分割，所以作为 1 个指令来处理。因此，有关各自指令中的异常、优先顺序也与通常不同。延迟槽指令仅具有 1 次数据传送时的顺序如下所示。

1. 检验带延迟的转移指令中的优先级1、 2的中断型及再执行型异常。
2. 检验延迟槽指令中的优先级1、 2的中断型及再执行型异常。
3. 检验带延迟的转移指令中的优先级2的完成型异常。
4. 检验延迟槽指令中的优先级2的完成型异常。
5. 检验带延迟的转移指令中的优先级3与延迟槽指令中的优先级3 （无2者之间的优先顺序）。
6. 检验带延迟的转移指令中的优先级4与延迟槽指令中的优先级4 （无2者之间的优先顺序）。

延迟槽指令有第 2 次数据传送时，在 2. 中，按照 （1）进行 2 次检验。

接受的异常 （优先权最高的异常）为延迟槽指令的再执行型异常时，不禁止转移指令的 PR 寄存器写入操作 （BSR、 BSRF、 JSR 的 PC→PR 运行）。但是，此时不保证 PR 寄存器的内容。

5.7 注意事项

(1) 从异常处理返回

1. 请用软件检验SR的BL位。 SPC、 SSR保存至存储器时，请将SR的BL位置1后再恢复。
2. 请发行RTE指令。通过RTE指令，将SPC设定至PC，将SSR设定至SR，转移至SPC的地址后，从异常处理返回。

(2) SR.BL=1 时发生异常或中断的情况

1. 异常
   发生除用户中断以外的异常时，发生手动复位。此时EXPEVT为H'0000 0020， SPC、 SSR的各寄存器为不定值。
2. 中断
   产生普通中断时，保留中断请求，通过软件将SR的BL位清0后接受中断请求。产生非屏蔽中断（NMI）时，可通过软件设定保留或接受。
   但是，睡眠或待机状态下，即使SR的BL位为1，也接受中断。

(3) 发生异常时的 SPC

1. 再执行型异常
   将产生异常的指令的PC设定至SPC，从异常处理返回后重新执行。但是，在延迟槽指令发生异常时，不管之前的延迟转移指令的条件是否成立，都将延迟转移指令的PC设定至SPC。
2. 完成型异常、中断
   将产生异常的指令的下一个指令的PC设定至SPC。但是，在带延迟槽的转移指令发生异常时，将转移目标的PC设定至SPC。

(4) RTE 指令的延迟槽

1. 保存至SSR的值返回至SR后执行RTE指令的延迟槽中所配置的指令。指令存取相关的异常的接受判断根据返回前的SR值来确定，其它异常的接受判断根据由返回后的SR的处理模式、 BL位来决定。虽然完成型的异常是在执行RTE的转移目标前接受，但是，如果发生再执行型异常，则不保证运行。
2. RTE指令的延迟槽中所配置的指令不接受用户中断。

(5) 变更 SR 寄存器值与接受异常

1. 通过LDC指令改变SR寄存器的MD、 BL位时，从下一个指令开始，通过SR寄存器的新值重新判断异常的接受*。在完成型异常中，在下一个指令执行后接受异常，但是完成型异常中的中断在下一个指令执行前接受。

【注】 * 执行对于 SR 的 LDC 指令时，再次执行对后续指令的取指令，并通过 SR 的新值进行取指令异常的重新评价。

第 6 章 浮点单元 （FPU）

6.1 概要

FPU 具有以下特点：

- 依据IEEE754规格
- 32个单精度浮点寄存器 （也可作为16个双精度寄存器来参考）
- 2个舍入模式：向接近方向及0方向的舍入
- 2个非规格化数处理模式：向0的闪存与非规格化数的处理
- 6个异常源：
  FPU错误、无效运算、被0除、上溢、下溢、不正确
- 包括的指令：
  单精度、双精度、图形支持、系统控制
- 在SH-4A中追加以下3个指令：
  FSRRA、 FSCA、 FPCHG

将 SR 的 FD 位置 1 时，不可使用浮点单元 （FPU），如果打算执行 FPU 指令，则发生 FPU 禁止异常（普通 FPU 禁止异常或槽 FPU 禁止异常）。

6.2 数据格式

6.2.1 浮点格式

浮点由以下 3 个字段构成：

- 符号位 （s）
- 指数字段 （e）
- 小数字段 （f）

SH-4A 可采用图 6.1 与图 6.2 所示的格式，处理单精度、双精度浮点。

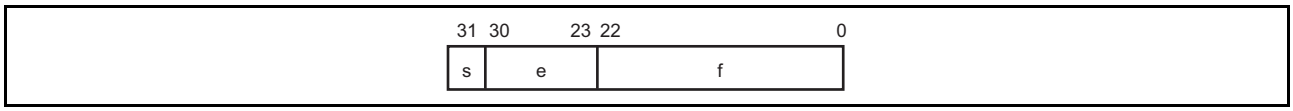


图 6.1 单精度浮点格式

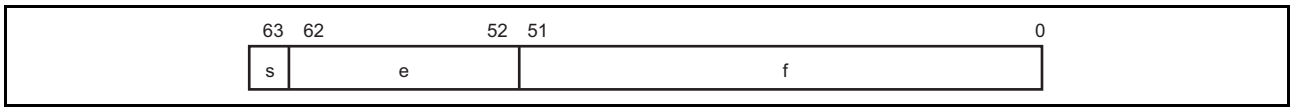


图 6.2 双精度浮点格式

指数表示如下 （附带偏置）：

e ＝ E + 偏置

无偏置指数 E 的范围从 E_{min} – 1 到 E_{max} +1。 E_{min} –1 与 E_{max} +1 两个值区别如下：E_{min} –1 表示 0 （正、负双方的符号）与非规格化数， Emax +1 表示正或负的无穷大或非数 （NaN）。浮点格式与参数如表 6.1 所示。

表 6.1 浮点的格式与参数

参数	单精度	双精度
总位宽度	32 位	64 位
符号位 （s）	1 位	1 位
指数字段 （e）	8 位	11 位
小数字段 （f）	23 位	52 位
精度	24 位	53 位
偏置	+127	+1023
E_{max}	+127	+1023
E_{min}	–126	–1022

浮点数值 v 可如下决定：

- E = E_{max} + 1 且 f ≠ 0 时， v 为与符号 s 无关的非数 （NaN）。
- E = E_{max} + 1 且 f = 0 时， v 为 $(-1)^s$ （无穷）“正无穷或负无穷”。
- E_{min} ≤ E ≤ E_{max} 时， v 为 $(-1)^s 2^E$ （1.f）“规格化数”。
- E = E_{min} – 1 且 f ≠ 0 时， v 为 $(-1)^s 2^{Emin}$ （0.f）“非规格化数”。
- E = E_{min} – 1 且 f = 0 时， v 为 $(-1)^s 0$ “正 0 或负 0”。

16 进制数各类型的范围如**表 6.2** 所示。关于信令非数与静态非数请参考 “**6.2.2 非数 （NaN）**”，关于非规格化数请参考 “**6.2.3 非规格化数**”。

表 6.2 浮点的范围

类型	单精度	双精度
信令非数	H'7FFFFFFF ～ H'7FC00000	H'7FFFFFFF FFFFFFFF ～ H'7FF80000 00000000
静态非数	H'7FBFFFFF ～ H'7F800001	H'7FF7FFFF FFFFFFFF ～ H'7FF00000 00000001
正的无穷大	H'7F800000	H'7FF00000 00000000
正的规格化数	H'7F7FFFFF ～ H'00800000	H'7FEFFFFF FFFFFFFF ～ H'00100000 00000000
正的非规格化数	H'007FFFFF ～ H'00000001	H'000FFFFF FFFFFFFF ～ H'00000000 00000001
正的零	H'00000000	H'00000000 00000000
负的零	H'80000000	H'80000000 00000000
负的非规格化数	H'80000001 ～ H'807FFFFF	H'80000000 00000001 ～ H'800FFFFF FFFFFFFF
负的规格化数	H'80800000 ～ H'FF7FFFFF	H'80100000 00000000 ～ H'FFEFFFFF FFFFFFFF
负的无穷大	H'FF800000	H'FFF00000 00000000
静态非数	H'FF800001 ～ H'FFBFFFFF	H'FFF00000 00000001 ～ H'FFF7FFFF FFFFFFFF
信令非数	H'FFC00000 ～ H'FFFFFFFF	H'FFF80000 00000000 ～ H'FFFFFFFF FFFFFFFF

6.2.2 非数 （NaN）

非数 （NaN）的位格式如图 6.3 所示。以下情况下的值为 NaN。

- 符号位 ：don't care
- 指数字段：所有位为1
- 小数字段：至少1位为1

小数字段的 MSB 为 1 时， NaN 为信令非数 （sNaN）。小数字段为 0 时， NaN 为静态非数 （qNaN）。

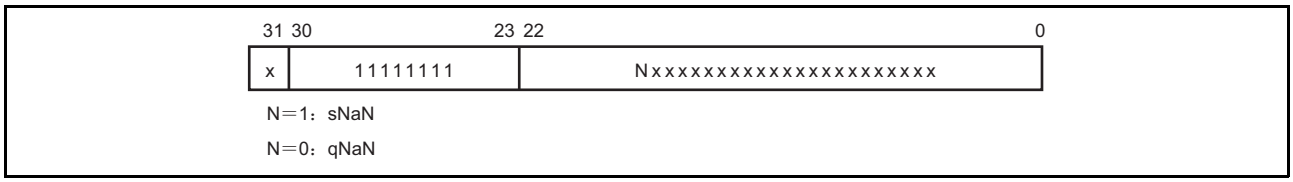


图 6.3 单精度的 NaN 位组合

假设 sNaN 为生成寄存器与寄存器之间的传送指令 FABS 或 FNEG 以外的浮点值的运算输入数据时，

- FPSCR 寄存器的 EN.V 位为 0 时，运算结果 （输出）为 qNaN。
- FPSCR 寄存器的 EN.V 位为 1 时，发生无效运算异常。此时，运算的目标寄存器的内容不变。

寄存器与寄存器之间的传送指令具有如下 3 条指令：

- FMOV FRm,FRn
- FLDS FRm,FPUL
- FSTS FPUL,FRn

通过生成浮点值的运算，输入 qNaN，却未在此运算中输入 sNaN 时，与 FPSCR 寄存器的 EN.V 位的设定无关，输出通常为 qNaN。此时，不发生异常。

作为运算结果， SH-4A 生成的 qNaN 值通常为如下的值：

- 单精度 qNaN：H'7FBFFFFF
- 双精度 qNaN：H'7FF7FFFF FFFFFFFF

关于输入非数 （NaN）时的浮点运算的详细内容请参考 “第 10 章 各指令的说明”。

6.2.3 非规格化数

非规格化数的浮点值表示为：指数字段设定为 0、小数字段设定为 0 以外的值。

FPU 的状态寄存器 FPSCR 的 DN 位为 1 时，通过生成 （寄存器与寄存器之间的传送指令、 FNEG、 FABS 以外的运算的）值的浮点运算，非规格化数 （源操作数或运算结果）为正的零或负的零。

FPSCR 的 DN 位为 0 时，非规格化数 （源操作数或运算结果）直接进行处理。关于输入非规格化数时的浮点运算的详细内容请参考 **“第 10 章 各指令的说明”**。

6.3 寄存器

6.3.1 浮点寄存器

浮点寄存器的结构如**图 6.4** 所示。具有 32 个 32 位浮点寄存器。这些寄存器由 2 个存储体构成，即：FPR0_BANK0 ～ FPR15_BANK0、 FPR0_BANK1 ～ FPR15_BANK1。另外，这 32 个寄存器作为 FR0 ～ FR15、DR0/2/4/6/8/l0/12/14、FV0/4/8/12、XF0 ～ XF15、XD0/2/4/6/8/l0/12/14、XMTRX 来参考。由 FPSCR 的 FR 位决定 FPRn_BANKi 与参考名称的对应。

(1) 浮点寄存器 FPRn_BANKi （32 个寄存器）

FPR0_BANK0 ～ FPR15_BANK0
FPR0_BANK1 ～ FPR15_BANK1

(2) 单精度浮点寄存器 FRi （16 个寄存器）

FPSCR.FR=0 时， FR0 ～ FR15 分配至 FPR0_BANK0 ～ FPR15_BANK0。
FPSCR.FR=1 时， FR0 ～ FR15 分配至 FPR0_BANK1 ～ FPR15_BANK1。

(3) 双精度浮点寄存器或单精度浮点寄存器对 DRi （8 个寄存器）

DR 寄存器由 2 个 FR 寄存器构成。
DR0={FR0、 FR1}、 DR2={FR2、 FR3}、
DR4={FR4、 FR5}、 DR6={FR6、 FR7}、
DR8={FR8、 FR9}、 DR10={FR10、 FR11}、
DR12={FR12、 FR13}、 DR14={FR14、 FR15}

(4) 单精度浮点向量寄存器 FVi （4 个寄存器）

FV 寄存器由 4 个 FR 寄存器构成。
FV0={FR0、 FR1、 FR2、 FR3},
FV4={FR4、 FR5、 FR6、 FR7},
FV8={FR8、 FR9、 FR10、 FR11},
FVl2={FR12、 FR13、 FR14、 FR15}

(5) 单精度浮点扩展寄存器 XFi （16 个寄存器）

FPSCR.FR=0 时， XF0 ～ XF15 分配至 FPR0_BANK1 ～ FPR15_BANK1。
FPSCR.FR=1 时， XF0 ～ XF15 分配至 FPR0_BANK0 ～ FPR15_BANK0。

(6) 单精度浮点扩展寄存器对 XDi （8 个寄存器）

XD 寄存器由 2 个 XF 寄存器构成。

XD0={XF0、 XF1}、 XD2={XF2、 XF3}、

XD4={XF4、 XF5}、 XD6={XF6、 XF7}、

XD8={XF8、 XF9}、 XD10={XF10、 XF11}、

XD12={XF12、 XF13}、 XD14={XF14、 XF15}

(7) 单精度浮点扩展寄存器矩阵 XMTRX

XMTRX 由 16 个 XF 寄存器构成。

$$XMTRX=\begin{pmatrix} XF0 & XF4 & XF8 & XF12 \\ XF1 & XF5 & XF9 & XF13 \\ XF2 & XF6 & XF10 & XF14 \\ XF3 & XF7 & XF11 & XF15 \end{pmatrix}$$

FPSCR.FR=0				FPSCR.FR=1		
FV0	DR0	FR0	FPR0 BANK0	XF0	XD0	XMTRX
		FR1	FPR1 BANK0	XF1		
	DR2	FR2	FPR2 BANK0	XF2	XD2	
		FR3	FPR3 BANK0	XF3		
FV4	DR4	FR4	FPR4 BANK0	XF4	XD4	
		FR5	FPR5 BANK0	XF5		
	DR6	FR6	FPR6 BANK0	XF6	XD6	
		FR7	FPR7 BANK0	XF7		
FV8	DR8	FR8	FPR8 BANK0	XF8	XD8	
		FR9	FPR9 BANK0	XF9		
	DR10	FR10	FPR10 BANK0	XF10	XD10	
		FR11	FPR11 BANK0	XF11		
FV12	DR12	FR12	FPR12 BANK0	XF12	XD12	
		FR13	FPR13 BANK0	XF13		
	DR14	FR14	FPR14 BANK0	XF14	XD14	
		FR15	FPR15 BANK0	XF15		
XMTRX	XD0	XF0	FPR0 BANK1	FR0	DR0	FV0
		XF1	FPR1 BANK1	FR1		
	XD2	XF2	FPR2 BANK1	FR2	DR2	
		XF3	FPR3 BANK1	FR3		
	XD4	XF4	FPR4 BANK1	FR4	DR4	FV4
		XF5	FPR5 BANK1	FR5		
	XD6	XF6	FPR6 BANK1	FR6	DR6	
		XF7	FPR7 BANK1	FR7		
	XD8	XF8	FPR8 BANK1	FR8	DR8	FV8
		XF9	FPR9 BANK1	FR9		
	XD10	XF10	FPR10 BANK1	FR10	DR10	
		XF11	FPR11 BANK1	FR11		
	XD12	XF12	FPR12 BANK1	FR12	DR12	FV12
		XF13	FPR13 BANK1	FR13		
	XD14	XF14	FPR14 BANK1	FR14	DR14	
		XF15	FPR15 BANK1	FR15		

图 6.4 浮点寄存器

6.3.2 浮点状态 / 控制寄存器 （FPSCR）

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	FR	SZ	PR	DN	Cause	
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Cause				Enable（EN）					Flag					RM	
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 22	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
21	FR	0	R/W	浮点寄存器存储体 0：FPR0_BANK0 ～ FPR15_BANK0 分配至 FR0 ～ FR15， FPR0_BANK1 ～ FPR15_BANK1 分配至 XF0 ～ XF15。 1：FPR0_BANK0 ～ FPR15_BANK0 分配至 XF0 ～ XF15， FPR0_BANK1 ～ FPR15 ～ BANK1 分配至 FR0 ～ FR15。
20	SZ	0	R/W	传送长度模式 0：FMOV 指令的数据长度为 32 位。 1：FMOV 指令的数据长度为 32 位对或 64 位。 SZ 位及 PR 位与字节序的关系请参考图 6.5。
19	FR	0	R/W	精度模式 0：作为单精度运算执行浮点指令。 1：作为双精度运算执行浮点指令 （图形支持指令未定义）。 PR 位及 SZ 位与字节序的关系请参考图 6.5。
18	DN	1	R/W	非规格化模式 0：作为非规格化数处理非规格化数。 1：作为 0 处理非规格化数。
17 ～ 12	Cause	均为 0	R/W	FPU 异常源字段
11 ～ 7	Enable(EN)	均为 0	R/W	FPU 异常允许字段
6 ～ 2	Flag	均为 0	R/W	FPU 异常标志字段 执行 FPU 运算指令时， FPU 异常源字段最初设定为 0。然后发生 FPU 异常时， FPU 异常源字段及 FPU 异常标志字段的相应位置 1。FPU 异常标志字段保持最后清除 FPU 异常标志字段以后发生的异常的状态。 有关各字段的位的分配请参考表 6.3。
1、 0	R	B'01	R/W	舍入模式 选择舍入的方法。 00：向接近方向的舍入 01：向 0 方向的舍入 10：保留 （禁止设定） 11：保留 （禁止设定）

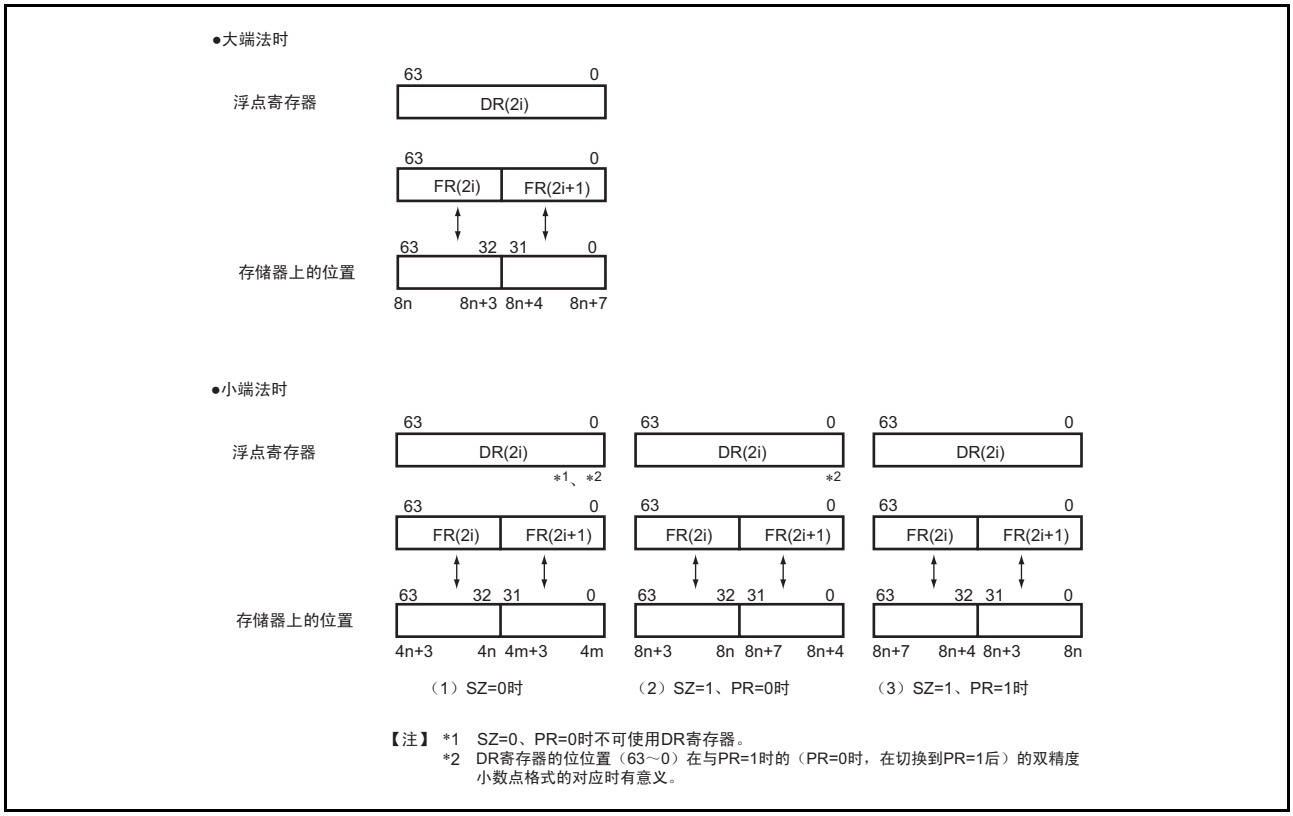


图 6.5 SZ 位与字节序的关系

表 6.3 FPU 异常处理相关位的分配

		FPU 错误（E）	无效运算（V）	被 0 除（Z）	上溢（O）	下溢（U）	不正确（I）
Cause	FPU 异常源字段	bit17	bit16	bit15	bit14	bit13	bit12
Enable	FPU 异常允许字段	无	bit11	bit10	bit9	bit8	bit7
Flag	FPU 异常标志字段	无	bit6	bit5	bit4	bit3	bit2

6.3.3 浮点通信寄存器 （FPUL）

通过 FPUL 寄存器进行 FPU 与 CPU 间的信息传递。 FPUL 寄存器为 32 位系统寄存器，也通过 LDS、 STS 指令从 CPU 存取。例如，将保存至通用寄存器 R1 的整数转换为单精度浮点的处理流程如下：

R1 → （LDS 指令）→ FPUL → （单精度 FLOAT 指令） → FR1

6.4 舍入

浮点指令中，从中间结果生成最终运算结果时进行舍入。因此，如 FMAC、 FTRV、 FIPR 的组合指令的结果，与仅使用 FADD、FSUB、FMUL 等基本指令的结果不同。这是因为 FMAC 进行 1 次舍入， FADD、FSUB 及 FMUL 进行 2 次舍入。

舍入有 2 种方法，使用方法由 FPSCR 的 RM 字段决定。

RM =00：向接近方向的舍入

RM =01：向 0 方向的舍入

(1) 向接近方向的舍入

将运算结果舍入至最接近的可表现的值。有 2 个最接近的可表现的值时，选择 LSB 为 0。

如果舍入前的值大于等于 $2^{Emax}(2-2^{-p})$，则为与舍入前相同符号的无穷。在此， E_{max}、 p 在单精度时分别为 127、 24，在双精度时分别为 1023、 53。

(2) 向 0 方向的舍入

可舍去舍入前的值的舍入位以下的位。

但是，舍入前的值大于可表现的最大绝对值时，为与舍入前相同符号的可表现的最大绝对值。

6.5 浮点异常

FPU 相关的异常如下：

(1) 普通 FPU 禁止 / 槽 FPU 禁止异常

SR.FD=1 时，如果执行 FPU 指令，则发生普通 FPU 禁止异常 / 槽 FPU 禁止异常。 FPU 指令存在于延迟槽以外时，发生普通 FPU 禁止异常， FPU 指令存在于延迟槽时，发生槽 FPU 禁止异常。

(2) FPU 异常

异常源如下所示：

- FPU错误 （E）：
  FPSCR.DN＝0且输入非规格化数时
- 无效运算 （V）：
  如NaN输入一样的无效运算时
- 被0除 （Z）：
  以0为除数的除法
- 上溢 （O）：
  运算结果上溢时
- 下溢 （U）：
  运算结果下溢时
- 不精确异常 （I）：
  发生舍入时

FPSCR 的 FPU 异常源字段中包含上述 E、V、Z、O、U、I 的所有相应的位，FPSCR 的标志及允许字段包含 V、 Z、 O、 U、 I 相应的位，但不包含 E 相应的位。这样，不可将 FPU 错误设定为禁止。

如果发生 FPU 异常， FPU 异常源字段的相应位置 1，与 FPU 异常标志字段相应的位累积 1。未发生 FPU 异常时， FPU 异常源字段的相应位清 0，与 FPU 异常标志字段相应的位不变。

(3) FPU 异常处理

下列情况下发生 FPU 异常：

- FPU错误 （E）：
  FPSCR.DN=0且对不可处理非规格化数的指令输入非规格化数时
- 无效运算 （V）：
  FPSCR.EN.V=1且 （指令=FTRV或无效运算）时
- 被0除 （Z）：
  FPSCR.EN.Z=1且以0为除数的除法或FSRRA的输入为0时
- 上溢 （O）：
  FPSCR.EN.O=1且运算结果有可能上溢时
- 下溢 （U）：
  FPSCR.EN.U=1且运算结果有可能下溢时
- 不精确异常 （I）：
  FPSCR.EN.I=1且运算结果有可能为不正确的指令

源于 FPU 运算的所有异常事件作为相同的异常事件被分配。使用软件通过读取系统寄存器 FPSCR、解释保持的信息来决定异常的含义内容。

另外，无论通过任何 FPU 异常处理运行，都不变更目标寄存器。

在上述情况以外产生 FPU 异常源时，将 V、 Z、 O、 U、 I 相应的位置 1，作为运算结果生成默认值。

- 无效运算 （V）：
  作为结果生成qNaN。
- 被0除 （Z）：
  生成带与舍入前相同符号的无穷大。
- 上溢 （O）：
  向0方向舍入时，生成带与舍入前相同符号的最大规格化数。
  向接近方向舍入时，生成带与舍入前相同符号的无穷大。
- 下溢 （U）：
  FPSCR.DN=0时，生成带与舍入前相同符号的非规格化数或带与舍入前相同符号的0 。
  FPSCR.DN=1时，生成带与舍入前相同符号的0。
- 不精确异常 （I）：
  生成不正确的结果。

6.6 图形支持功能

SH-4A 支持 2 种图形功能。一个为几何运算用的指令，另一个为可进行快速数据传送的一对单精度传送指令。

6.6.1 几何运算指令

因为几何运算指令能够通过最小的硬件进行快速运算， SH-4A 忽视 4 个乘法的部分运算结果中相对小的值。因此，运算结果中产生如下误差：

最大误差 =MAX(乘法结果 $\times 2^{-MIN(\text{乘数的有效数字位数}-1,\ \text{被乘数的有效数字位数}-1)}$)+ MAX(结果值 $\times 2^{-23}$、 2^{-149})

但是，对于规格化数，有效数字位数为 24，对于非规格化数有效数字位数为 23 （小数部分的前导零的位数）。

保证今后的 SuperH 系列中的运算误差 , 但不保证不同的处理器内核之间相同的运算结果。

(1) FIPR FVm,FVn （m、 n：0、 4、 8、 12）

此指令的用途例如下所示：

- 内积 （m≠n）：
  通常，此运算用于判断多边形表面的亮度及表面 / 里面。
- 各要素的平方和 （m=n）：
  通常，此运算用于获取向量长度。

由于 FIPR 指令未检测不精确异常，所以执行指令时， FPU 异常源字段及 FPU 异常标志字段的不精确异常 (I) 位通常置 1。因此，如果将 FPU 异常允许字段的 I 位置位，则进行 FPU 异常处理。

(2) FTRV XMTRX, FVn （n：0、 4、 8、 12）

此指令的用途例如下所示：

- 矩阵 （4×4）• 向量 （4）：
  通常，此运算用于视点变更、角度变更或移动的向量转换 （4维）。基本上，用于角度 + 平行移动的防射转换处理需要 4×4 矩阵。因此， SH-4A 支持 4 维运算。
- 矩阵 （4×4）• 矩阵 （4×4）：
  为进行此运算，需要执行 4 次 FTRV 指令。

由于 FIRV 指令未检测不精确异常，所以执行指令时， FPU 异常源字段及 FPU 异常标志字段的不精确异常（I）位通常置 1。因此，如果允许字段的 I 位置位，则进行 FPU 异常处理。另外，执行 FTRV 指令时，执行前不可检查寄存器内所有数据类型。如果将 FPU 异常允许字段的 V 位置位，则进行 FPU 异常处理。

(3) FRCHG

此指令变更存储体寄存器。例如，使用 FTRV 指令时，需要在背后的存储体中设定矩阵要素。但是，作成转换矩阵的要素本体时，使用前面的存储体寄存器的方法比较简单。使用对 FPSCR 的 LDS 指令时，此指令为了维持 FPU 的状态，需要 4 ～ 5 个周期。 FRCHG 指令可通过 1 个周期进行 FPSCR.FR 位的变更。

6.6.2 对单精度数据传送

包含强大的几何运算指令， SH-4A 支持快速数据传送指令。
FPSCR.SZ=1 时，进行通过对单精度数据传送指令的数据传送。

- FMOV DRm/XDm、DRn/XDRn(m、n：0、 2、 4、 6、 8、 10、 12、 14)
- FMOV DRm/XDm、 @Rn(m：0、 2、 4、 6、 8、 10、 12、 14、 n：0～15)
  通过这些指令，可传送2个单精度 （2×32位）数据。即，这些指令的传送性能为2倍。

- FSCHG
  此指令变更FPSCR的SZ位的值。可快速转换是否进行对单精度数据的传送。

第 7 章 存储器管理单元 （MMU）

SH-4A 可从 8 位地址空间标识符与 32 位虚拟地址空间处理 29 位物理地址空间。使用内置于 SH-4A 的存储器管理单元 （MMU：Memory Management Unit）进行从虚拟地址向物理地址的地址转换。 MMU 通过将用户编制的地址转换表的信息向转换旁路缓冲器 （TLB：Translation Lookaside Buffer）高速缓存的操作，快速进行地址转换。

SH-4A 内置 4 个指令 TLB （ITLB）入口、 64 个共用 TLB （UTLB）入口，通过硬件将 UTLB 的备份保存至 ITLB。地址转换方式为分页方式，支持 4 种 （1K/4K/64K/1M 字节）页面大小。另外，可分别在特权模式、用户模式下，设定对虚拟地址空间的存取权，并进行存储保护。

7.1 MMU 的概要

所谓 MMU 就是为了有效地利用物理存储器而设想的功能。如图 7.1(0) 所示，程序的大小比物理存储器小时，可将所有程序映射至物理存储器。但是，增大程序的大小、物理存储器保存不下时，需分割程序，并将执行时必需的部分随时映射至物理存储器 （图 7.1(1)）。从程序本身考虑，执行向此物理存储器的映射，增大了程序的负担。为了减轻此负担，总括处理向物理存储器的映射，所产生的考虑方法为虚拟存储方式（图 7.1(2)）。虚拟存储方式与物理存储器相比，具备了足够大的虚拟存储器。程序映射至此虚拟存储器。因此，程序仅考虑在虚拟存储器中的运行即可。从虚拟存储器向物理存储器的映射中，可使用 MMU。通常， OS 管理 MMU，为了使程序所需的虚拟存储器顺利地映射至物理存储器，进行物理存储器的转换。物理存储器的转换可在 2 次存储等之间进行。

由此产生的虚拟存储方式在多个程序同时运行的分时系统 （TSS）中发挥巨大作用 （图 7.1(3)）。由于在 TSS 中运行的多个程序一边识别向各自物理存储器的映射一边运行，所以效率并未提高。为了提高效率、减轻各程序的负担，可使用虚拟存储方式 （图 7.1(4)）。此虚拟存储方式中可将虚拟存储器分配至每个程序。MMU 有效地将多个虚拟存储器映射至物理存储器。而且，为了使某个程序不会错误地存取其他程序的物理存储器， MMU 还具备存储保护功能。

使用 MMU 从虚拟存储器向物理存储器进行地址转换时，有时此转换信息未注册到 MMU，有时会错误地存取不同程序的虚拟存储器。此时， MMU 产生异常，并变更物理存储器的映射，注册新的地址转换信息。

仅通过软件也可实现 MMU 功能，但由于程序每次存取物理存储器都通过软件进行转换，所以效率极低。因此，在硬件中备有用于地址转换的缓冲器 （TLB），频繁使用的地址转换信息预先设置于 TLB。 TLB 可称为用于地址转换信息的高速缓存。但是，与高速缓存不同，地址转换失败时，即通常通过软件转换异常发生时的地址转换信息。因此可通过软件灵活地进行存储器管理。

MMU 作为从虚拟存储器向物理存储器映射的方式，具有使用固定长度的地址转换方式 （分页方式）与使用可变长度的地址转换方式 （分段方式）。分页方式下被称为固定大小的页面的地址空间为转换单位。

以下，在 SH-4A 中将虚拟存储器中的地址空间称为虚拟地址空间，将物理存储器中的地址空间称为物理地址空间。

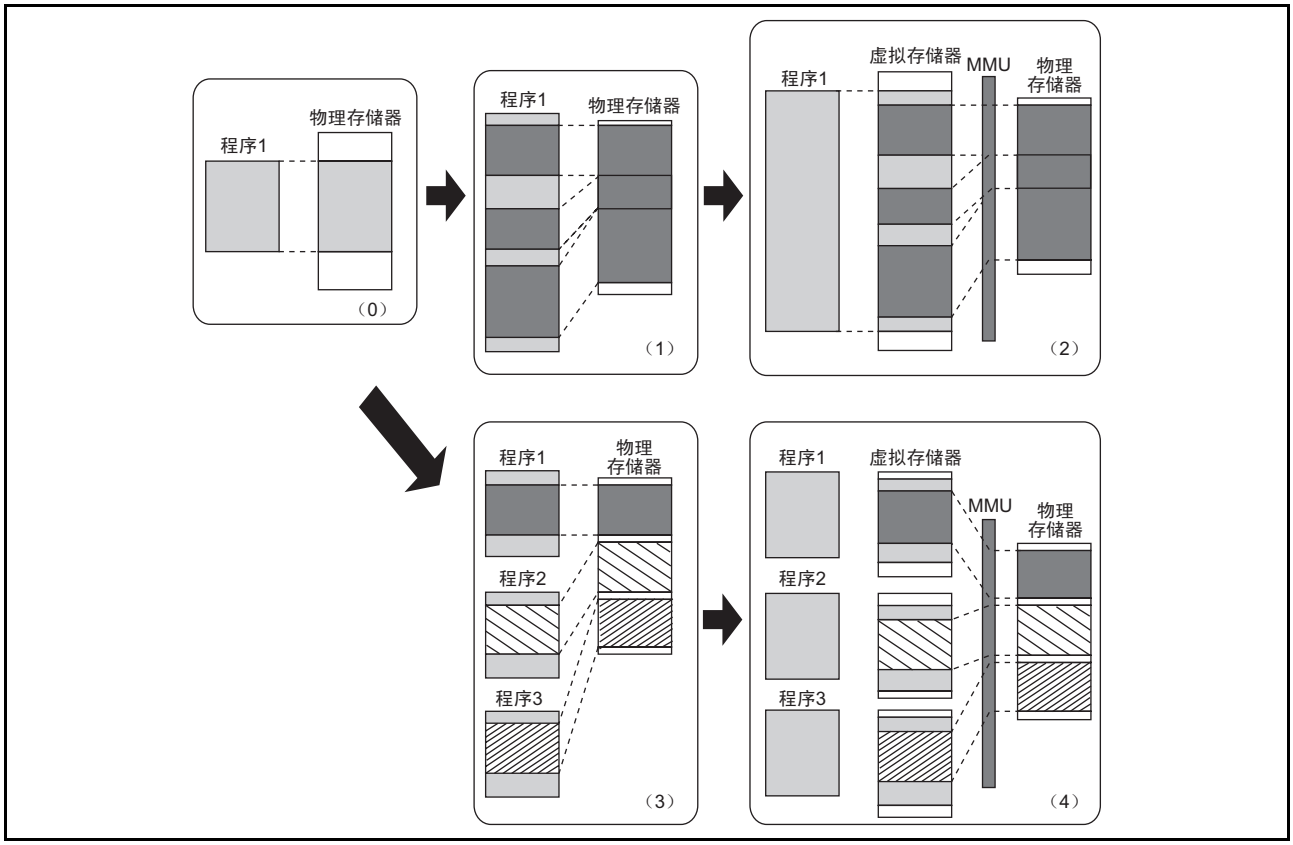


图 7.1 MMU 的作用

7.1.1 地址空间

(1) 虚拟地址空间

SH-4A 支持 32 位虚拟地址空间，可存取 4G 字节的地址空间。虚拟地址空间如图 7.2、图 7.3 所示，可分为若干区域。特权模式下可存取从 P0 区域到 P4 区域的 4G 字节空间。用户模式下可存取 U0 区域的 2G 字节空间。另外， MMU 控制寄存器 （MMUCR）的 SQMD 位为 0 时，也可存取存储队列区域的 64M 字节空间，内部存储器控制寄存器 （RAMCR）的 RMD 位为 1 时，也可存取内部存储器区域的 16M 字节空间。用户模式下存取 U0 区域、存储队列区域、内部存储器区域以外时，则为地址错误。

将 MMUCR 的 AT 位置 1、MMU 设定为允许时，在这些区域中， P0、P3、U0 区域能够以 1K/4K/64K/1M 字节为页单位映射至任意的物理地址空间。另外，通过使用 8 位地址空间标识符可将 P0、 P3、 U0 区域增加至 256 个。进行从虚拟地址空间向 29 位物理地址空间的映射时使用 TLB。

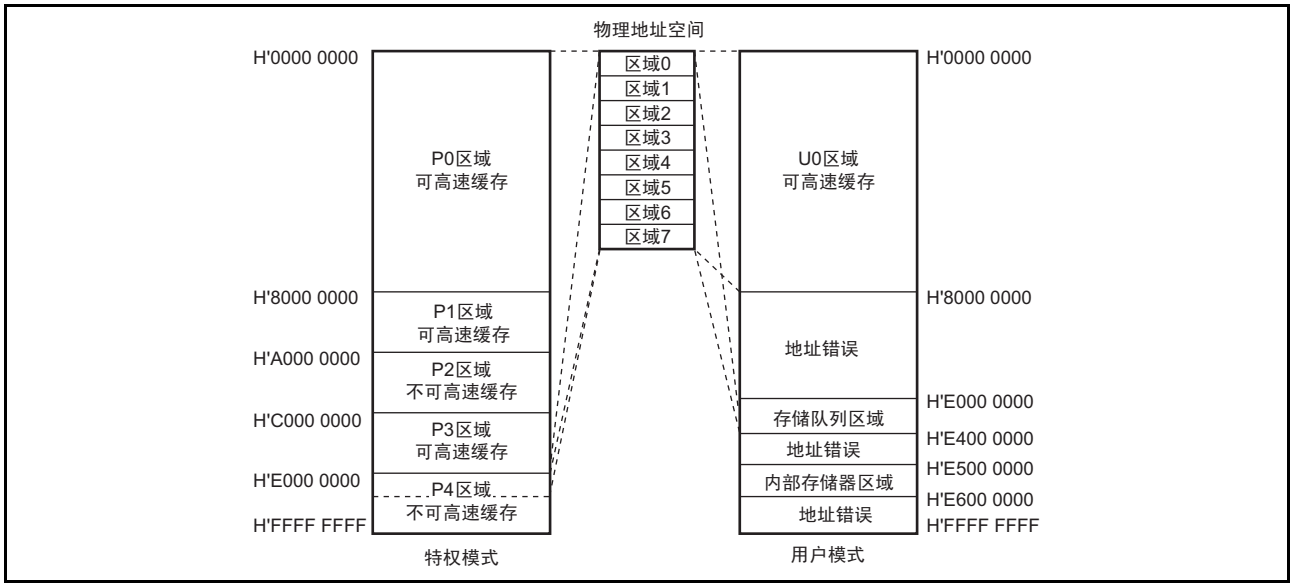


图 7.2 虚拟地址空间 （MMUCR.AT=0）

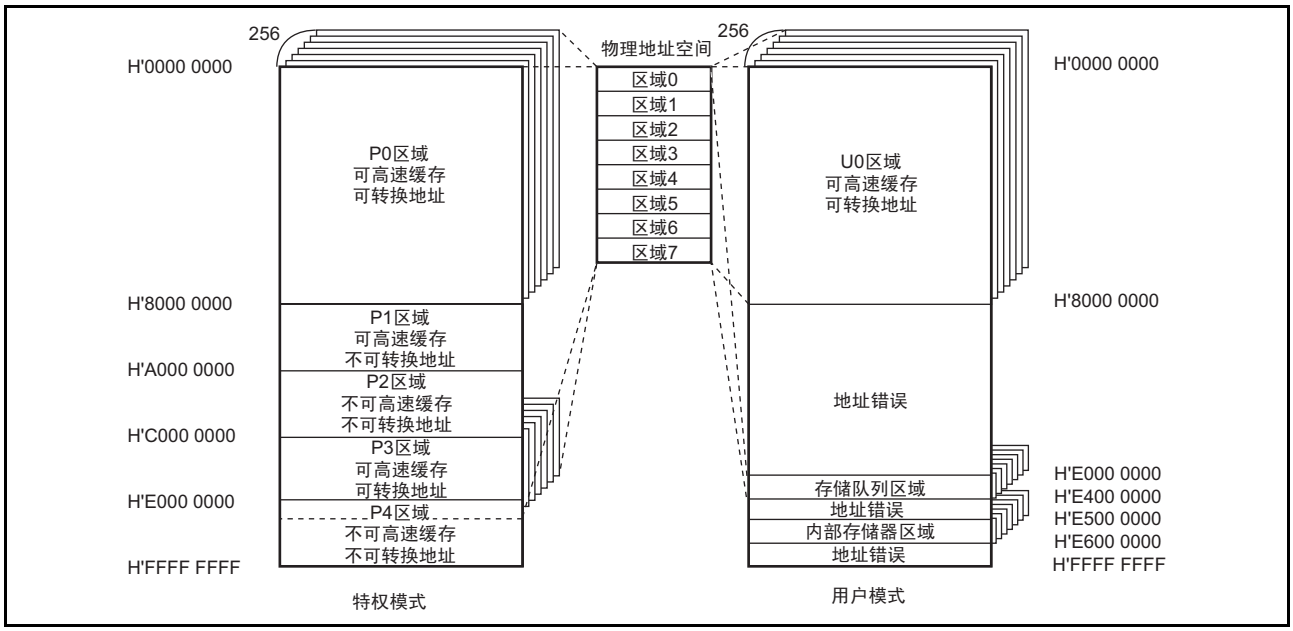


图 7.3 虚拟地址空间 （MMUCR.AT=1）

(a) P0、 P3、 U0 区域

P0、 P3、 U0 区域可使用 TLB 进行地址转换也可使用高速缓存进行存取。

MMU 禁止时，地址的高 3 位置 0 的地址为对应的物理地址空间的地址。是否使用高速缓存依据高速缓存控制寄存器。使用高速缓存时，写存取中的备份方式与直写方式的转换依据 CCR 的 WT 位。

MMU 允许时，这些区域可使用 TLB，以 1K/4K/64K/1M 字节为页单位映射至任意的物理地址空间。 CCR 为高速缓存允许状态，而且 TLB 入口的该页的可进行高速缓存操作位 （C 位）为 1 时，可进行使用高速缓存的存取。使用高速缓存时，写存取中的备份方式与直写方式的转换依据 TLB 的 WT 位。

通过 TLB 将这些区域映射至存在于物理地址空间的区域 7 中的控制寄存器区域时，请将该页的 C 位清 0。

(b) P1 区域

P1 区域不可进行使用 TLB 的地址转换，但为可使用高速缓存的存取区域。

无论 MMU 是否允许，地址的高 3 位置 0 的地址均为对应的物理地址空间的地址。是否使用高速缓存依据 CCR。使用高速缓存时，写存取中的备份方式与直写方式的转换依据 CCR 的 CB 位。

(c) P2 区域

P2 区域为不可进行使用 TLB 的地址转换与使用高速缓存的存取的区域。

无论 MMU 是否允许，将地址的高 3 位置 0 的地址均为对应的物理地址空间的地址。

(d) P4 区域

P4 区域为映射至 SH-4A 的内部资源的区域。此区域除存储队列与内部存储器区域，不可进行使用 TLB 的地址转换。另外，此区域不可进行使用高速缓存的存取。 P4 区域的详细内容如**图 7.4** 所示。

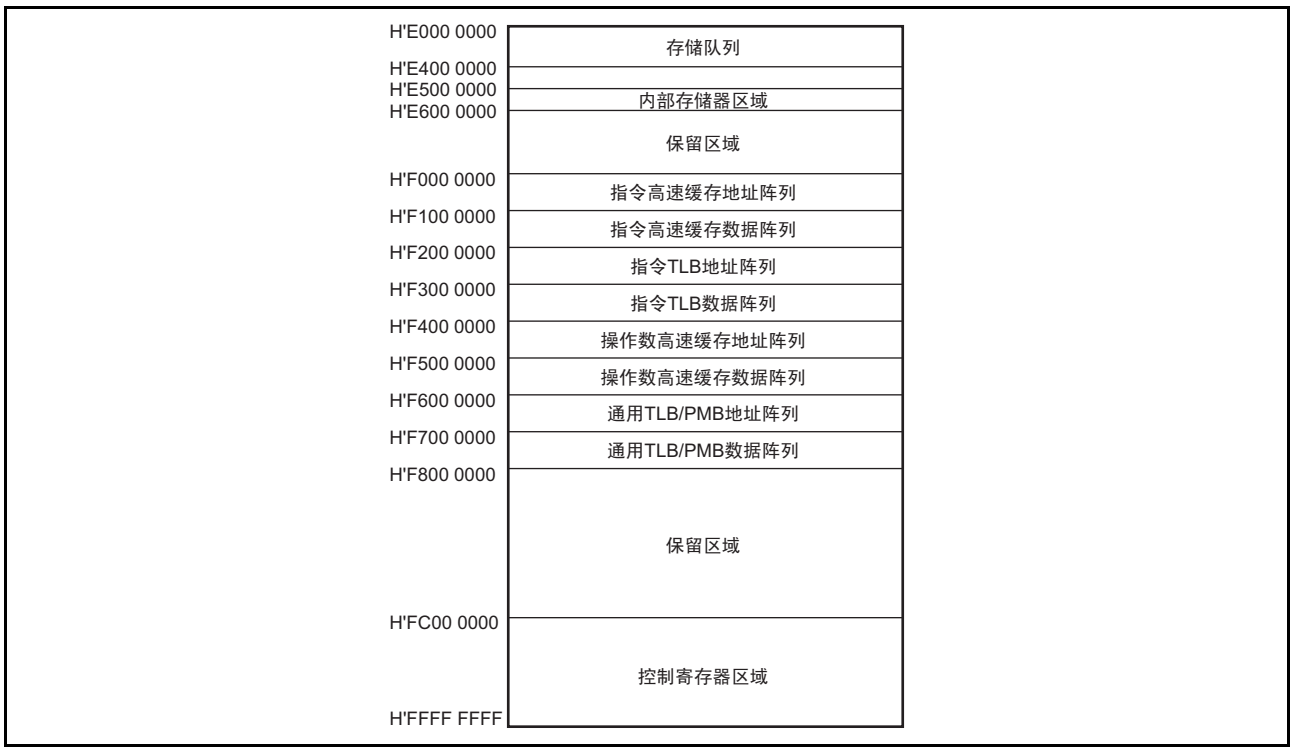


图 7.4 P4 区域

H'E000 0000 ～ H'E3FF FFFF 为用于存取存储队列 （SQ）的区域。通过 MMUCR 的 SQMD 位指定用户模式下的存取权。详细内容请参考 “8.7 存储队列”。

H'E500 0000 ～ H'E5FF FFFF 为用于存取内部存储器的区域。通过 RAMCR 寄存器的 RMD 位指定用户模式下的存取权。详细内容请参考 “第 9 章 L 存储器”。

H'F000 0000 ～ H'F0FF FFFF 为用于直接存取指令高速缓存的地址阵列的区域。详细内容请参考 “8.6.1 IC 地址阵列”。

H'F100 0000 ～ H'F1FF FFFF 为用于直接存取指令高速缓存的数据阵列的区域。详细内容请参考 “8.6.2 IC 数据阵列”。

H'F200 0000 ～ H'F2FF FFFF 为用于直接存取指令 TLB 的地址阵列的区域。详细内容请参考 “7.6.1 ITLB 地址阵列”。

H'F300 0000 ～ H'F37F FFFF 为用于直接存取指令 TLB 的数据阵列的区域。详细内容请参考 “7.6.2 ITLB 数据阵列”。

H'F400 0000 ～ H'F4FF FFFF 为用于直接存取操作数高速缓存的地址阵列的区域。详细内容请参考“8.6.3 OC 地址阵列”。

H'F500 0000 ～ H'F5FF FFFF 为用于直接存取操作数高速缓存的数据阵列的区域。详细内容请参考“8.6.4 OC 数据阵列”。

H'F600 0000 ～ H'F60F FFFF 为用于直接存取共用 TLB 的地址阵列的区域。详细内容请参考 “7.6.3 UTLB 地址阵列”。

H'F610 0000 ～ H'F61F FFFF 为用于直接存取 PMB 的地址阵列的区域。详细内容请参考 “7.7.5 存储器分配 PMB 的结构”。

H'F700 0000 ～ H'F70F FFFF 为用于直接存取共用 TLB 的数据阵列的区域。详细内容请参考 “7.6.4 UTLB 数据阵列”。

H'F710 0000 ～ H'F71F FFFF 为用于直接存取 PMB 的数据阵列的区域。详细内容请参考 “7.7.5 存储器分配 PMB 的结构”。

H'FC00 0000 ～ H'FFFF FFFF 为内部外围模块的控制寄存器的区域。详细内容请参考硬件手册各章节的寄存器说明的项。

(2) 物理地址空间

SH-4A 支持 29 位物理地址空间。物理地址空间如图 7.5 所示，分为 8 个区域。区域 7 为保留区域。详细内容请参考相关产品硬件手册的 “总线状态控制器”的章节。

仅限使用 TLB 存取物理地址空间的区域 7 时，区域 7 的 H'1C00 0000 ～ H'1FFF FFFF 区域不为保留区域，与包含于虚拟地址空间的 P4 区域的控制寄存器区域等效。

图 7.5 物理地址空间

(3) 地址转换

使用 MMU 时，将虚拟地址空间分割为页单位，并以此页为单位转换为物理地址。将与虚拟地址对应的物理地址及存储保护码等附加信息保存至外部存储器中的地址转换表，为使地址转换高速化，将外部存储器中的地址转换表的内容向 TLB 高速缓存。在 SH-4A 中指令存取时使用 ITLB，数据存取时使用 UTLB。发生向 P4 区域以外的存取时，此已存取的虚拟地址转换为物理地址。此虚拟地址属于 P1、 P2 区域时，不存取 TLB，仅决定物理地址。此虚拟地址属于 P0、 U0、 P3 区域时，通过虚拟地址检索 TLB，此虚拟地址注册在 TLB 时，为 TLB 命中，从 TLB 读取对应的物理地址。另外，已存取的虚拟地址未注册在 TLB 时，发生 TLB 未命中异常，处理转移至 TLB 未命中异常处理程序。在 TLB 未命中异常处理程序中，检索外部存储器中的地址转换表，并将对应的物理地址、页面管理信息注册于 TLB。而且从异常处理程序返回后，重新执行使 TLB 未命中异常发生的指令。

(4) 单虚拟存储模式与多重虚拟存储模式

虚拟存储方式具有单虚拟存储方式与多重虚拟存储方式，可通过 MMUCR 的 SV 位选择。单虚拟存储方式中，多个程序一边排他性地使用虚拟地址空间一边同时运行，仅决定与某个虚拟地址对应的物理地址。多重虚拟存储方式中，由于多个程序一边共用虚拟地址空间一边运行，所以某个虚拟地址可通过程序转换为不同的物理地址。单虚拟存储方式与多重虚拟存储方式运行的不同，仅为 TLB 地址比较的方式 （参考 **“7.3.3 地址转换方式”**）。

(5) 地址空间标标识符 （ASID）

多重虚拟存储模式时， 8 位地址空间标识符 （ASID）可用于区别一边共用虚拟地址空间一边同时运行的多个程序。 ASID 为 8 位，软件在 MMU 内的 PTEH 中设定当前运行的程序的 ASID。另外，通过 ASID 转换程序时无需清除 TLB。

单虚拟存储模式时， ASID 用于一边排他性地使用虚拟地址空间一边同时运行的多个程序的存储保护。

【注】 单虚拟存储模式的设定中，不可将具有不同 ASID 的同一虚拟页码 （VPN）的多个入口同时设定至 TLB。

7.2 寄存器说明

有关 MMU 处理的寄存器如下所示。

表 7.1 寄存器结构

名称	简称	R/W	P4 区域地址 *	区域 7 地址 *	大小
页表入口高位寄存器	PTEH	R/W	H'FF00 0000	H'1F00 0000	32
页表入口低位寄存器	PTEL	R/W	H'FF00 0004	H'1F00 0004	32
转换表基址寄存器	TTB	R/W	H'FF00 0008	H'1F00 0008	32
TLB 异常地址寄存器	TEA	R/W	H'FF00 000C	H'1F00 000C	32
MMU 控制寄存器	MMUCR	R/W	H'FF00 0010	H'1F00 0010	32
物理地址空间控制寄存器	PASCR	R/W	H'FF00 0070	H'1F00 0070	32
重新取指令禁止控制寄存器	IRMCR	R/W	H'FF00 0078	H'1F00 0078	32

【注】 * P4 区域地址为使用虚拟地址空间的 P4 区域时的地址。区域 7 地址为使用 TLB 从物理地址空间的区域 7 进行存取的地址。

表 7.2 各处理状态下的寄存器状态

名称	简称	上电复位	手动复位	睡眠	待机
页表入口高位寄存器	PTEH	不定	不定	保持	保持
页表入口低位寄存器	PTEL	不定	不定	保持	保持
转换表基址寄存器	TTB	不定	不定	保持	保持
TLB 异常地址寄存器	TEA	不定	保持	保持	保持
MMU 控制寄存器	MMUCR	H'0000 0000	H'0000 0000	保持	保持
物理地址空间控制寄存器	PASCR	H'0000 0000	H'0000 0000	保持	保持
重新取指令禁止控制寄存器	IRMCR	H'0000 0000	H'0000 0000	保持	保持

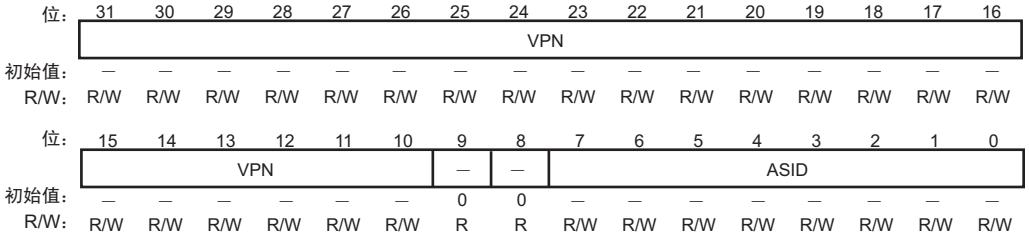
7.2.1 页表入口高位寄存器 （PTEH）

PTEH 由虚拟页码 （VPN）与地址空间标识符 （ASID）构成。 VPN 在产生 MMU 异常或地址错误异常时，通过硬件设定已产生异常的虚拟地址的 VPN。 VPN 因页面大小而不同，但异常发生时通过硬件设定的 VPN 为已产生异常的虚拟地址的高 22 位。也可通过软件设定 VPN。通过软件在 ASID 设定当前执行中的程序编号。 ASID 不通过硬件进行更新。通过 LDTLB 指令将此 VPN 与 ASID 注册至 UTLB。

请在更新 PTEH 寄存器的 ASID 字段后，对使用更新后的 ASID 值的 P0、 P3、 U0 区域进行存取 （含取指令）前，执行以下 1 ～ 3 中的任意一项：

1. 请执行通过RTE指令产生的转移。此时，转移目标也可为P0、 P3、 U0区域。
2. 请对任意地址 （不能进行高速缓存操作的区域也可以）执行ICBI指令。
3. PTEH更新前预先设定为IRMCR.R2=0 （初始值）时，无需执行特定的指令。但是因为此方法，必须从取指令重新执行PTEH更新指令的下一个指令， CPU的处理效率会降低，所以请注意。

但是，在今后的 SuperH 系列中有可能不保证方法 3。为保证在今后的 SuperH 系列中的兼容性，推荐使用 1 或 2。



位	位名称	初始值	R/W	说明
31 ～ 10	VPN	—	R/W	虚拟页码
9、 8	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
7 ～ 0	ASID	—	R/W	地址空间标识符

7.2.2 页表入口低位寄存器 （PTEL）

PTEL 用于保存通过 LDTLB 指令注册至 UTLB 的物理页码与页面管理信息。除非软件指示，否则本寄存器就不会变更内容。

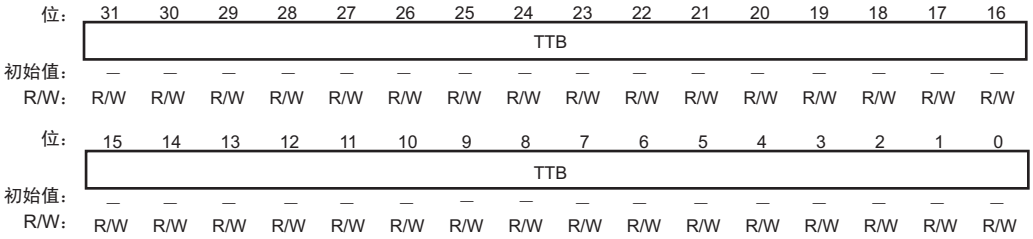
位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	PPN												
初始值：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PPN						—	V	SZ1	PR1	PR0	SZ0	C	D	SH	WT
初始值：	—	—	—	—	—	—	0	—	—	—	—	—	—	—	—	—
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 29	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
28 ～ 10	PPN	—	R/W	物理页码
9	—	0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
8	V	—	R/W	页面管理信息 各位的含义与共用 TLB （UTLB）的对应位相同。 详细内容请参考 “7.3 TLB 功能”。
7	SZ1	—	R/W	
6 5	PR1 PR0	—	R/W	
4	SZ0	—	R/W	
3	C	—	R/W	
2	D	—	R/W	
1	SH	—	R/W	
0	WT	—	R/W	

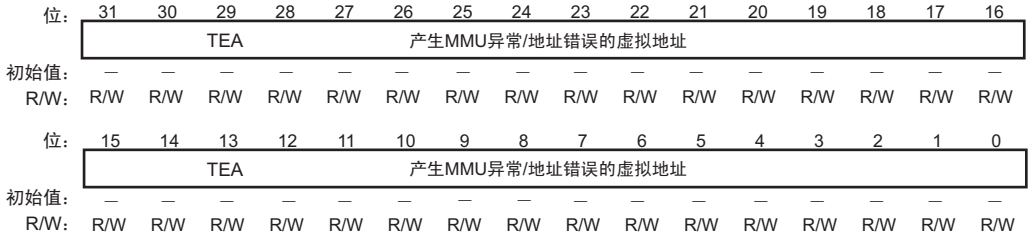
7.2.3 转换表基址寄存器 （TTB）

TTB 是用于保存当前使用的页表基址等的寄存器。除非软件指示，否则 TTB 就不会变更内容。本寄存器可通过软件自由地使用。



7.2.4 TLB 异常地址寄存器 （TEA）

TEA 在 MMU 异常或地址错误异常发生后保存已产生异常的虚拟地址。可通过软件变更此寄存器。



7.2.5 MMU 控制寄存器 （MMUCR）

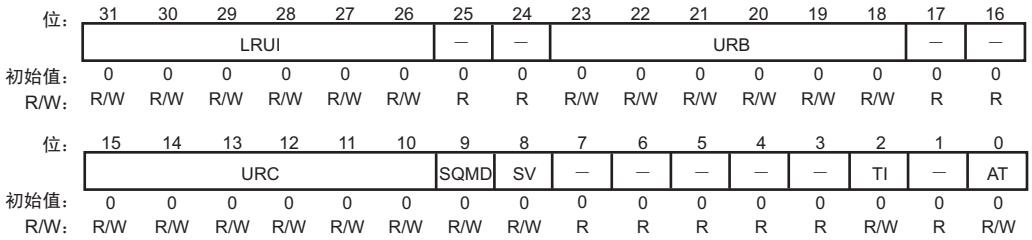
MMUCR 的各位如下进行 MMU 的设定。因此请通过 P1、 P2 区域的程序改写 MMUCR。

请在更新 MMUCR 寄存器后，对 P0、 P3、 U0、存储队列区域存取 （含取指令）前，执行以下 1 ～ 3 项中的任意一项：

1. 请执行通过RTE指令产生的转移。此时，转移目标也可为P0、 P3、 U0区域。
2. 请对任意地址 （不能进行高速缓存操作的区域也可以）执行ICBI指令。
3. MMUCR更新前预先设定为IRMCR.R2=0 （初始值）时，无需特定的指令序列。但是因为此方法，必须从取指令重新执行MMUCR更新指令的下一个指令， CPU的处理效率会降低，所以请注意。

但是，在今后的 SuperH 系列中有可能不保证方法 3。为保证在今后 SuperH 系列的兼容性，推荐使用 1 或 2。

可通过软件变更 MMUCR。但是也可通过硬件更新 LRUI 位与 URC 位。



位	位名称	初始值	R/W	说明
31 ～ 26	LRUI	均为 0	R/W	表示进行转换的 ITLB 入口的 LRU 位 为了在 ITLB 未命中发生时决定转换的 ITLB 入口，使用 LRU 方式 （Least Recently Used）。可使用 LRUI 位，确定 ITLB 的清除入口。 LRUI 可通过以下算法进行更新： 以下的 “x” 表示不进行更新。 000xxx：使用 ITLB 的入口 0 时 1xx00x：使用 ITLB 的入口 1 时 x1x1x0：使用 ITLB 的入口 2 时 xx1x11：使用 ITLB 的入口 3 时 xxxxxx：上述以外 另外， LRUI 为以下状态时，通过 ITLB 未命中更新对应的 ITLB 入口。请勿通过硬件设定下表中禁止设定的值。因为上电复位、手动复位后， LRUI 初始化为 0，所以通过更新硬件， LRUI 不为禁止设定的值。 以下的 “x” 表示 Don't care。 111xxx：更新 ITLB 的入口 0 0xx11x：更新 ITLB 的入口 1 x0x0x1：更新 ITLB 的入口 2 xx0x00：更新 ITLB 的入口 3 上述以外：禁止设定
25、 24	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
23 ～ 18	URB	均为 0	R/W	表示进行转换的 UTLB 入口边界的位 URB ≠ 0 时有效。
17、 16	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
15 ～ 10	URC	均为 0	R/W	用于表示通过 LDTLB 指令进行转换的 UTLB 入口的随机计数器 每次产生对 UTLB 的存取就会进行递增。但是当 URB ＞ 0 时，如果 URC=URB 的条件成立，则 URC 清 0。另外，通过软件将 URC ＞ URB 的值写入 URC 时，最初，到 URC=H'3F 前，超过 URB 进行递增，所以请注意。 URC 不通过 LDTLB 指令进行累加计数。
9	SQMD	0	R/W	存储队列模式位 指定对存储队列的存取权。 0：可进行用户 / 特权存取 1：可进行特权存取 （用户存取时为地址错误异常）
8	SV	0	R/W	单虚拟存储模式 / 多重虚拟存储模式转换位 变更此位时，必须将 1 也写入 TI 位。 0：多重虚拟存储模式 1：单虚拟存储模式
7 ～ 3	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。

位	位名称	初始值	R/W	说明
2	TI	0	R/W	TLB 无效位 将 1 写入此位时， UTLB/ITLB 的有效位全部清 0。读取时通常读取为 0。
1	—	0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
0	AT	0	R/W	地址转换有效位 指定 MMU 的允许 （有效）与禁止 （无效）。 0：设定为 MMU 禁止 1：设定为 MMU 允许 AT 位为 0 的状态下不产生 MMU 异常。因此在不使用 MMU 的软件中，请在 AT 位为 0 的状态下使用。

7.2.6 物理地址空间控制寄存器 （PASCR）

PASCR 控制物理地址空间的运行。

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	UB							
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 8	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
7 ～ 0	UB	均为 0	R/W	每个区域 （64M 字节）的缓冲写入控制 在完成不使用高速缓存的写入的总线存取前，每个区域指定是否等待来自 CPU 的下一个总线存取。 0：CPU 不等待写入的总线存取完成就进行下一个总线存取 1：CPU 等待写入的总线存取完成后进行下一个总线存取 UB[7]：对应控制寄存器区域 UB[6]：对应区域 6 UB[5]：对应区域 5 UB[4]：对应区域 4 UB[3]：对应区域 3 UB[2]：对应区域 2 UB[1]：对应区域 1 UB[0]：对应区域 0

7.2.7 重新取指令禁止控制寄存器 （RMCR）

变更特定资源时， IRMCR 控制是否从取指令重新执行下一个指令。所谓特定资源就是指控制寄存器的一部分、 TLB、高速缓存。

初始状态下资源变更后，设定为重新执行下一个指令的取指令。但是此状态下，每进行一次资源的变更就会引起取指令的重新执行，并且 CPU 的处理效率降低。因此，推荐将 IRMCR 的各位设定为 1，集中进行必要的资源变更后，执行特定指令后，转移至执行使用变更后的资源的程序。

有关特定的顺序请参考各资源说明。

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	—	—	R2	R1	LT	MT	MC
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 5	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
4	R2	0	R/W	寄存器变更后重新取禁止 2 变更 MMUCR、 PASCR、 CCR、 RAMCR、 PTEH 的各寄存器时，控制是否重新取下一个指令。 0：重新取 1：不重新取
3	R1	0	R/W	寄存器变更后重新取禁止 1 变更存在于地址 H'FF200000 ～ H'FF2FFFFF 的寄存器时，控制是否重新取下一个指令。 0：重新取 1：不重新取
2	LT	0	R/W	LDTLB 执行后重新取禁止 执行 LDTLB 指令后，控制是否重新取下一个指令。 0：重新取 1：不重新取
1	MT	0	R/W	存储器分配 TLB 写入后重新取禁止 MMUCR.AT=1 状态下，进行存储器分配 ITLB/UTLB 写入后，控制是否重新取下一个指令。 0：重新取 1：不重新取
0	MC	0	R/W	存储器分配 IC 写入后重新取禁止 CCN.ICE=1 状态下，进行存储器分配 IC 写入后，控制是否重新取下一个指令。 0：重新取 1：不重新取

7.3 TLB 功能

7.3.1 共用 TLB （UTLB）的结构

UTLB 用于以下 2 个目的：

1. 数据存取时，将虚拟地址转换为物理地址。
2. 指令TLB未命中时，注册至ITLB的地址转换信息表。

因此被称为共用 TLB。将外部存储器中设置的地址转换表信息高速缓存到 UTLB。将虚拟页码与地址空间标识符、与之对应的物理页码与页面管理信息保存至地址转换表。 UTLB 的结构如图 7.6 所示。 UTLB 由全相联方式的 64 个入口构成。页面大小与地址的关系如图 7.7 所示。

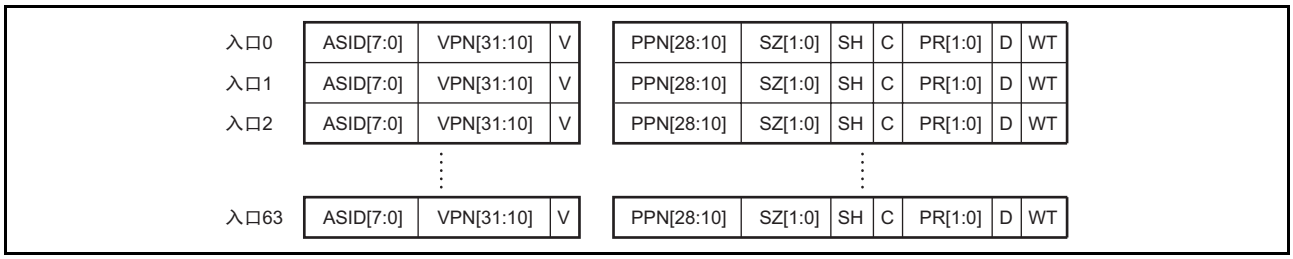


图 7.6 UTLB 的结构

【符号说明】

VPN：虚拟页码	1K 字节页时，虚拟地址的高 22 位 4K 字节页时，虚拟地址的高 20 位 64K 字节页时，虚拟地址的高 16 位 1M 字节页时，虚拟地址的高 12 位
ASID：地址空间标识符	表示可存取虚拟页的程序。 单虚拟存储模式且用户模式或多重虚拟存储模式时，如果 SH 位为 0，则在地址比较时与 PTEH 中的 ASID 进行比较。
SH：共有状态位	0：不在多个程序共有页。 1：在多个程序共有页。
SZ[1:0]：页面大小位	指定页面大小。 00：1K 字节页 01：4K 字节页 10：64K 字节页 11：1M 字节页
V：有效位	表示入口是否有效。 0：无效 1：有效 上电复位时清 0。 手动复位时不变。

PPN：物理页码	物理地址的高 22 位 1K 字节页时 PPN[28:10] 有效。 4K 字节页时 PPN[28:12] 有效。 64K 字节页时 PPN[28:16] 有效。 1M 字节页时 PPN[28:20] 有效。 另外，在 PPN 的设定中请注意同义问题 （参考 “7.4.5 同义问题的避免”）。
PR[1:0]：保护键数据	以代码表示页存取权的 2 位数据 00：特权模式下仅可读取 01：特权模式下可读取 / 写入 10：特权 / 用户模式下仅可读取 11：特权 / 用户模式下可读取 / 写入
C：可进行高速缓存操作位	表示页是否可以进行高速缓存操作。 0：不可进行高速缓存操作。 1：可进行高速缓存操作。 进行控制寄存器空间的映射时，请将此位置 0。
D：脏位	表示是否写入页。 0：未写入。 1：写入。
WT：直写位	指定向高速缓存的写入模式。 0：备份模式 1：直写模式

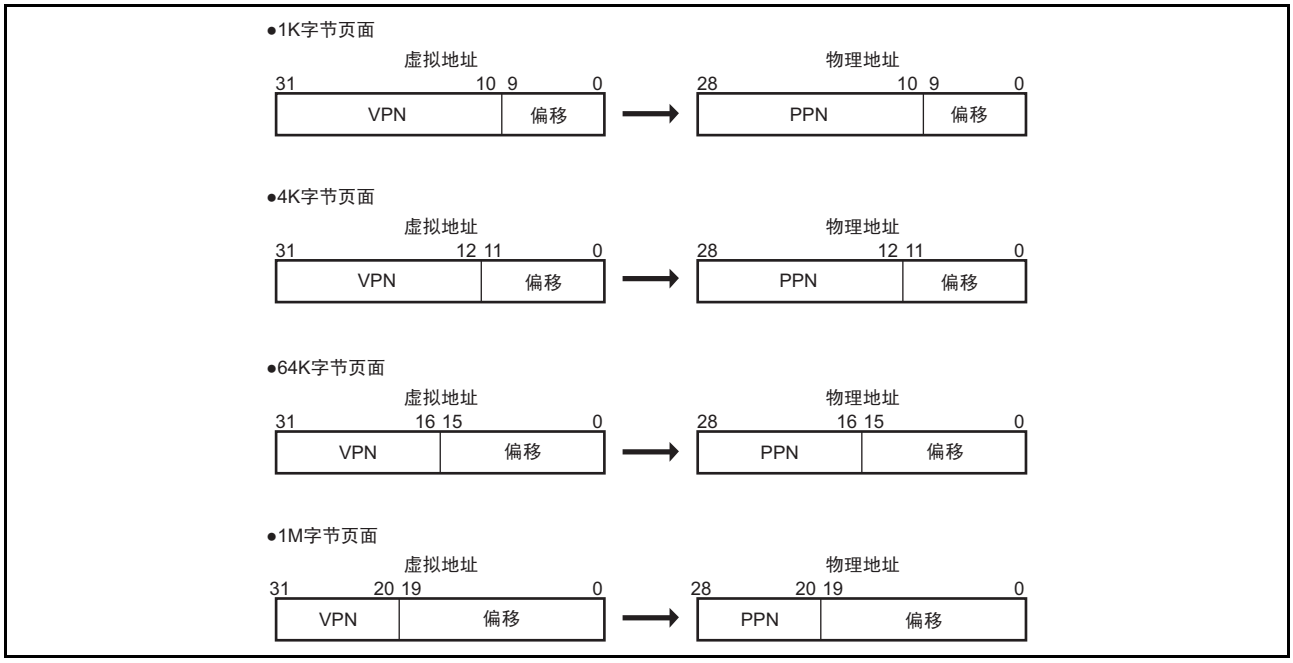


图 7.7 页面大小与地址的关系

7.3.2 指令 TLB （ITLB）的结构

ITLB 用于指令存取时将虚拟地址转换为物理地址。将 UTLB 中设置的地址转换表的信息高速缓存到 ITLB。 ITLB 的结构如图 7.8 所示。 ITLB 由全相联的 4 个入口构成。

入口0	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR
入口1	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR
入口2	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR
入口3	ASID[7:0]	VPN[31:10]	V	PPN[28:10]	SZ[1:0]	SH	C	PR

【注】 1. 不支持D、W位。
2. PR位变为1位时，与UTLB的PR位的高1位对应。

图 7.8 ITLB 的结构

7.3.3 地址转换方式

使用 UTLB 的存储器存取流程如**图 7.9** 所示。

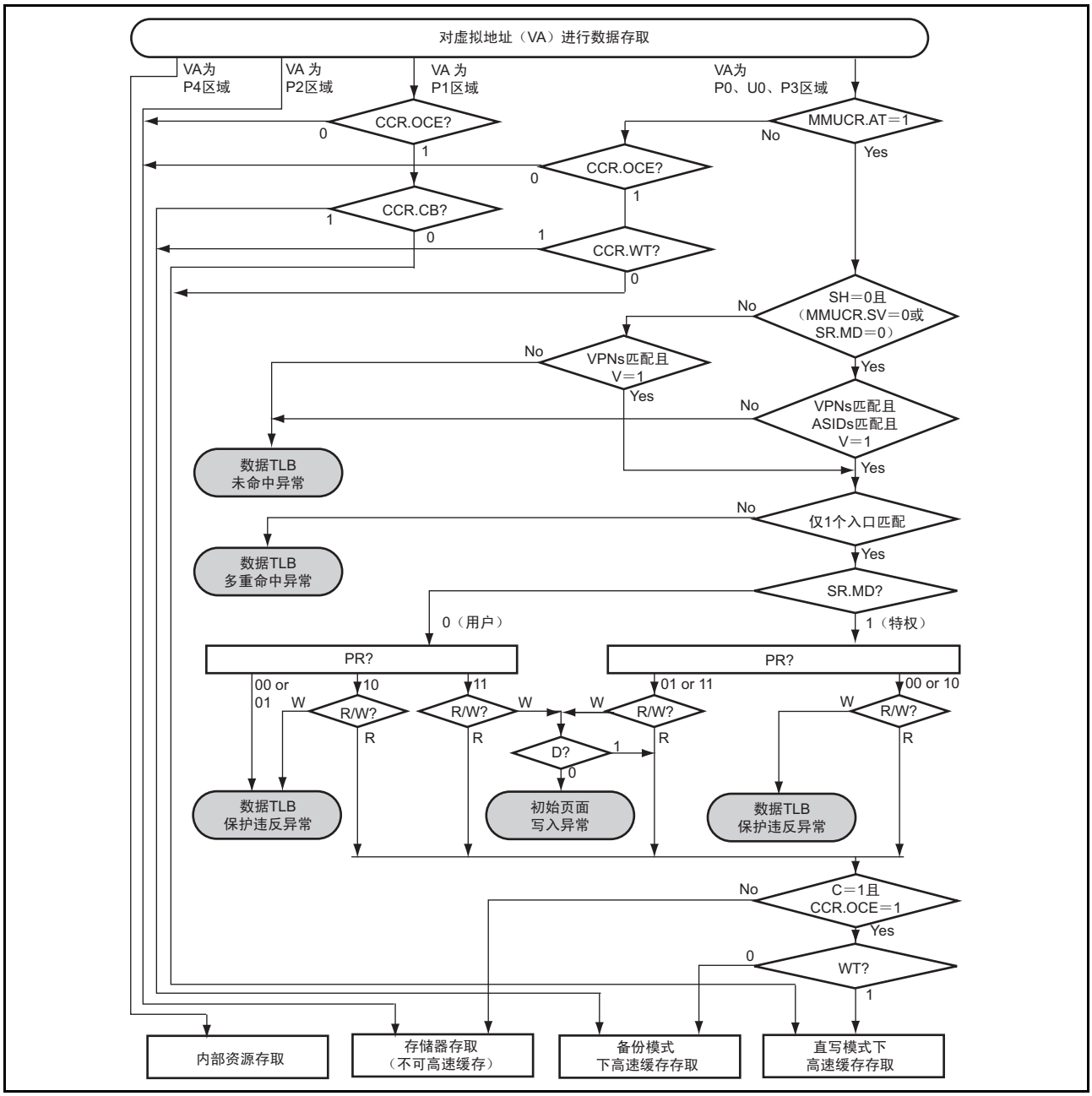


图 7.9 使用 UTLB 的存储器存取流程

使用 ITLB 的存储器存取流程如 7.10 所示。

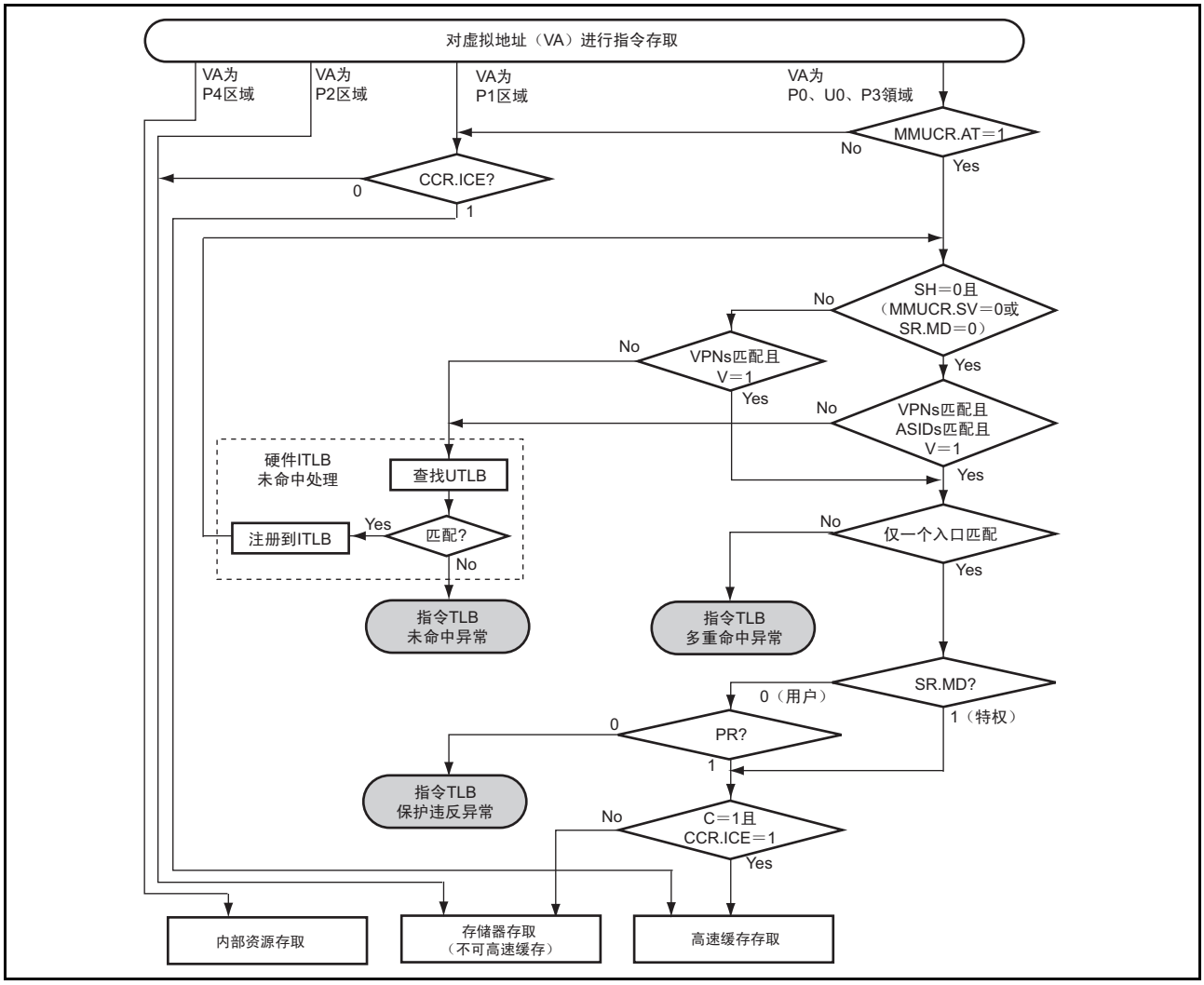


图 7.10 使用 ITLB 的存储器存取流程

7.4 MMU 功能

7.4.1 MMU 的硬件管理

SH-4A 支持如下所示的 MMU 功能：

1. 解码软件存取的虚拟地址，依据MMUCR的设定控制UTLB、 ITLB，并进行地址转换。
2. 以地址转换时读取的页面管理信息为基础，判断对高速缓存的存取状态 （C、 WT位）。
3. 数据存取、指令存取中不能正常地进行地址转换时，通过产生MMU异常来通知软件。
4. 指令存取中未在ITLB注册地址转换信息时，检索UTLB。在UTLB注册必要的地址转换信息时，依据MMUCR的LRUI位，将该地址转换信息复制到ITLB。

7.4.2 MMU 的软件管理

对 MMU 的软件处理如下：

1. MMU相关寄存器的设定。也可通过一部分硬件自动进行更新。
2. TLB入口的注册、删除、读取。 UTLB入口的注册有使用LDTLB指令的方法与直接写入存储器分配UTLB的方法。 ITLB入口的注册仅有直接写入存储器分配ITLB的方法。 UTLB、 ITLB入口的删除与读取可通过存取存储器分配UTLB、 ITLB。
3. MMU异常处理。产生MMU异常时，以硬件所设定的信息为基础进行处理。

7.4.3 MMU 指令 （LDTLB）

作为注册 UTLB 入口的指令具有 TLB 加载指令 （LDTLB）。发行 LDTLB 指令时， SH-4A 将 PTEH 与 PTEL 的内容复制到 URC 位所指示的 UTLB 入口。通过 LDTLB 指令不可更新 ITLB 入口，所以从 UTLB 入口清除的地址转换信息有可能残存于 ITLB 入口。由于 LDTLB 指令为变更地址转换信息的指令，因此必须通过 P1、 P2 区域的程序来发行。请在 LDTLB 指令执行后、 TLB 对有效区域进行存取 （含取指令）前，执行以下 1. ～ 3. 中的任意一项：

1. 请执行通过RTE指令产生的转移。此时，转移目标也可为TLB有效的区域。
2. 请对任意地址 （不能进行高速缓存操作的区域亦可）执行ICBI指令。
3. LDTLB指令执行前预先设定IRMCR.LT=0 （初始值）时，无需特定的指令序列。但是此方法，必须从取指令重新执行LDTLB指令的下一个指令， CPU的处理效率会降低，所以请注意。

但是，在今后的 SuperH 系列中有可能不保证方法 3.。为保证在今后 SuperH 系列中的兼容性，推荐使用 1. 或 2.。

LDTLB 指令的运行如图 7.11 所示。

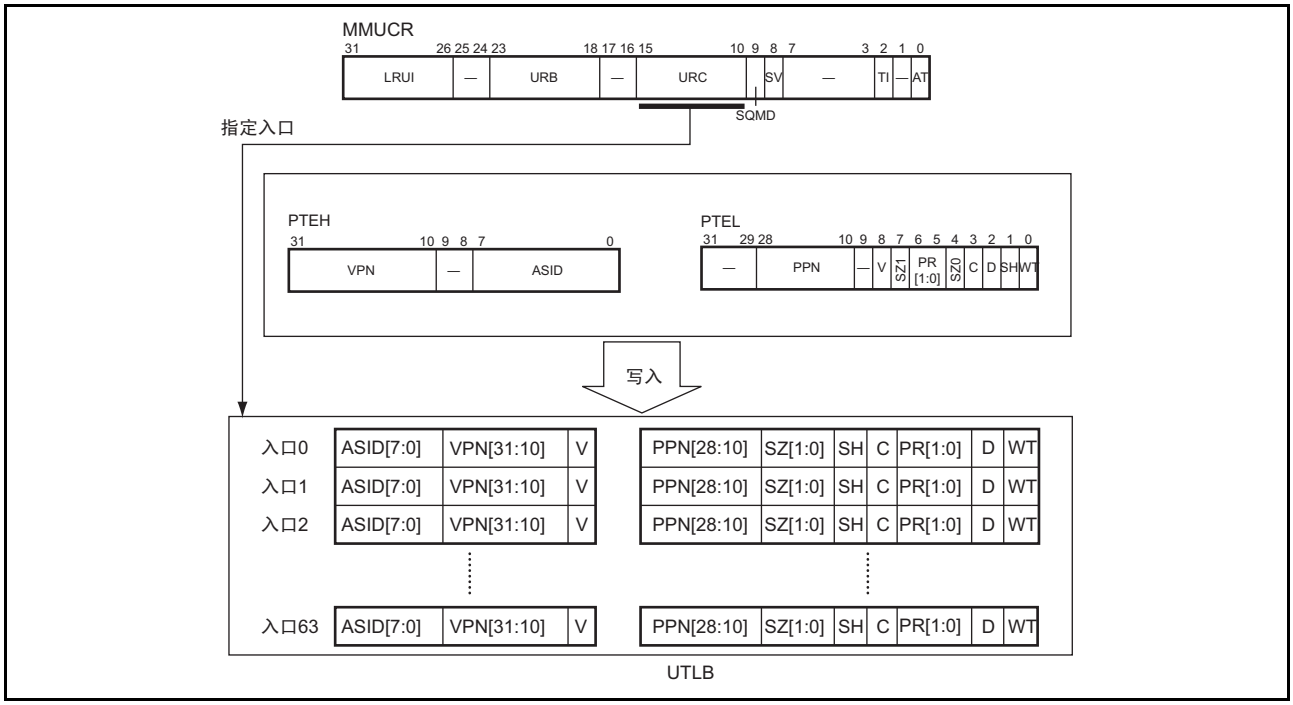


图 7.11 LDTLB 指令的运行

7.4.4 硬件 ITLB 未命中处理

存取指令时，在检索 ITLB 且未找到必要的地址转换信息 （ITLB 未命中）的情况下， SH-4A 通过硬件检索 UTLB，如果有必要的地址转换信息，则注册至 ITLB。将此称为硬件 ITLB 未命中处理。即使检索 UTLB 仍未找到必要的地址转换信息时，发生指令 TLB 未命中异常，将处理转移至软件。

7.4.5 同义问题的避免

在 TLB 入口注册 1K、 4K 字节页时，有可能产生同义问题。所谓同义问题就是指在多个虚拟地址映射至 1 个物理地址时，在高速缓存的多个入口注册完成相同物理地址数据，而不能保证数据一致性的问题。因为通过指令 TLB 与指令高速缓存仅可读取数据，所以不会产生此问题。为了在 SH-4A 中实现操作数高速缓存的高速运行，使用虚拟地址的 [12:5] 指定入口。但是 1K 字节页中虚拟地址的 [12:10] 与 4K 字节页中虚拟地址的 [12] 为地址转换的目标。因此虚拟地址的 [12:10] 有可能与转换后的物理地址的 [12:10] 不同。

因此向 UTLB 入口的地址转换信息的注册具有以下限制：

1. 将多个1K字节页的UTLB入口转换至相同物理地址的地址转换信息注册至UTLB时， VPN[12:10]必须相同。
2. 将多个4K字节页的UTLB入口转换至相同物理地址的地址转换信息注册至UTLB时， VPN[12]必须相同。
3. 请勿在不同页面大小的UTLB入口使用1K字节页的UTLB入口的物理地址。
4. 请勿在不同页面大小的UTLB入口使用4K字节页的UTLB入口的物理地址。

在进行使用高速缓存的存取时限定上述限制。

【注】 为了以后的 SuperH RISC engine 族扩展，在多个地址转换信息使用相同的物理存储器时，请将 VPN[20:10] 设定为相同。另外，请勿使不同页面大小的地址转换信息使用相同物理地址。

7.5 MMU 异常

MMU 异常具有指令 TLB 多重命中异常、指令 TLB 未命中异常、指令 TLB 保护违反异常、数据 TLB 多重命中异常、数据 TLB 未命中异常、数据 TLB 保护违反异常、初始页写入异常等的 7 种异常。有关各异常的发生条件请参考图 7.9 与图 7.10。

7.5.1 指令 TLB 多重命中异常

存在多个与指令存取的虚拟地址匹配的 ITLB 入口时，发生指令 TLB 多重命中异常。通过硬件 ITLB 未命中处理检索 UTLB 时，在 UTLB 多重命中发生的情况下，也发生指令 TLB 多重命中异常。

指令 TLB 多重命中异常发生时为复位，不保证高速缓存的相关性。

- 硬件处理

  指令TLB多重命中异常时，硬件进行以下处理：

  1. 将已产生异常的虚拟地址设定至TEA。
  2. 将异常码H'140设定至EXPEVT。
  3. 转移至复位处理程序 （H'A000 0000）。

- 软件处理 （复位程序）

  通过复位程序确认已发生多重命中的ITLB入口。由于此异常在程序调试时使用，所以通常请勿发生此异常。

7.5.2 指令 TLB 未命中异常

由于硬件 ITLB 未命中处理在 UTLB 入口未找到与指令存取的虚拟地址所对应的地址转换信息时，发生指令 TLB 未命中异常。通过指令 TLB 未命中异常的硬件进行的处理与通过软件进行的处理如下所述。这与数据 TLB 未命中异常时的处理相同。

- 硬件处理

  指令TLB未命中异常时，硬件进行以下处理：

  1. 将已产生异常的虚拟地址的VPN设定至PTEH。
  2. 将已产生异常的虚拟地址设定至TEA。
  3. 将异常码H'040设定至EXPEVT。
  4. 将指向异常产生的指令地址的PC值设定至SPC。如果在延迟槽发生异常，则将指向延迟转移指令地址的PC值设定至SPC。
  5. 将发生异常时的SR内容设定至SSR。将此时的R15设定至SGR。
  6. 将SR的MD位置1，转换至特权模式。
  7. 将SR的BL位置1，屏蔽此后的异常请求。
  8. 将SR的RB位置1。
  9. 转移至将偏移H'0000 0400添加到VBR内容的地址，并开始指令TLB未命中异常处理程序。

- 软件处理 （指令TLB未命中异常处理程序）

  软件负责检索外部存储器的页表、分配必要的页表入口。为了查找到必要的页表入口并进行分配，软件进行如下处理：

  1. 将记录在外部存储器地址转换表的页表入口的PPN、PR、SZ、C、D、SH、V、WT各位的值写入PTEL。
  2. 通过软件指定在入口调换中被调换的入口时，将该值写入MMUCR的URC。此时，在URC超过URB的情况下，请在LDTLB指令发行后变更为适当的值。
  3. 执行LDTLB指令，并将PTEH、 PTEL的内容写入TLB。
  4. 最后，请执行从异常处理返回的指令 （RTE），终止异常处理程序，将控制返回正常流程。但是，请在LDTLB指令的下一个指令以后发行RTE指令。

7.5.3 指令 TLB 保护违反异常

尽管与进行指令存取的虚拟地址匹配的地址转换信息存在于 ITLB 入口，但在通过 PR 位所指定的存取权不允许实际的存取类型时，发生指令 TLB 保护违反异常。指令 TLB 保护违反异常的硬件处理与软件处理如下所述：

- 硬件处理
  指令TLB保护违反异常时，硬件进行以下处理：
  1. 将已产生异常的虚拟地址的VPN设定至PTEH。
  2. 将已产生异常的虚拟地址设定至TEA。
  3. 将异常码H'0A0设定至EXPEVT。
  4. 将指向发生异常的指令地址的PC值设定至SPC。如果异常在延迟槽发生，则将指向延迟转移指令的地址的PC值设定至SPC。
  5. 将发生异常时的SR的内容设定至SSR。将此时的R15设定至SGR。
  6. 将SR的MD位置1，转换至特权模式。
  7. 将SR的BL位置1，屏蔽此后的异常请求。
  8. 将SR的RB位置1。
  9. 转移至将偏移H'0000 0100添加至VBR内容的地址，并开始指令TLB保护违反异常处理程序。
- 软件处理 （指令TLB保护违反异常处理程序）
  请解决指令TLB保护违反、执行从异常处理返回的指令 （RTE）、终止异常处理程序、使控制返回正常流程。但是，请在LDTLB指令的下一个指令以后发行RTE指令。

7.5.4 数据 TLB 多重命中异常

存在多个与数据存取的虚拟地址匹配的 UTLB 入口时，发生数据 TLB 多重命中异常。

数据 TLB 多重命中异常发生时为复位，不保证高速缓存的相关性。另外，有时会损坏异常发生前的 UTLB 中的 PPN 内容。

- 硬件处理
  数据TLB多重命中异常时，硬件进行以下处理：
  1. 将已产生异常的虚拟地址设定至TEA。
  2. 将异常码H'140设定至EXPEVT。
  3. 转移至复位处理程序 （H'A000 0000）。
- 软件处理 （复位程序）
  通过复位处理程序确认发生多重命中的UTLB入口。由于此异常在程序调试时使用，所以通常请勿发生此异常。

7.5.5 数据 TLB 未命中异常

在 UTLB 中未找到进行数据存取的虚拟地址所对应的地址转换信息时，发生数据 TLB 未命中异常。数据 TLB 未命中异常的硬件进行的处理与软件进行的处理如下所述：

- 硬件处理

  数据TLB未命中异常时，硬件进行以下处理：

  1. 将已产生异常的虚拟地址的VPN设定至PTEH。
  2. 将已产生异常的虚拟地址设定至TEA。
  3. 读取时将异常码H'040设定至EXPEVT，写入时将异常码H'060设定至EXPEVT （OCBP、OCBWB：读取， OCBI、 MOVCA.L：写入）。
  4. 将指向发生异常的指令地址的PC值设定至SPC。如果异常在延迟槽发生，则将指向延迟转移指令的地址的PC值设定至SPC。
  5. 将发生异常时的SR的内容设定至SSR。将此时的R15设定至SGR。
  6. 将SR的MD位置1，转换至特权模式。
  7. 将SR的BL位置1，屏蔽此后的异常请求。
  8. 将SR的RB位置1。
  9. 转移至将偏移H'00000400添加至VBR内容的地址，并开始数据TLB未命中异常处理程序。

- 软件处理 （数据TLB未命中异常处理程序）

  软件负责检索外部存储器的页表、分配必要的页表入口。为了查找到必要的页表入口并进行分配，软件进行如下处理：

  1. 将在外部存储器的地址转换表中记录的页表入口的PPN、PR、SZ、C、D、SH、V、WT各位的值写入PTEL。
  2. 通过软件指定在入口调换中被调换的入口时，并将该值写入MMUCR的URC。此时，在URC超过URB的情况下，请在LDTLB指令发行后变更为适当的值。
  3. 执行LDTLB指令，将PTEH、 PTEL的内容写入UTLB。
  4. 最后，请执行从异常处理返回的指令 （RTE）、终止异常处理程序、使控制返回正常流程。但是，请在LDTLB指令的下一个指令以后发行RTE指令。

7.5.6 数据 TLB 保护违反异常

尽管与进行数据存取的虚拟地址匹配的地址转换信息存在于 UTLB 入口，但在通过 PR 位所指定的存取权不允许实际的存取类型时，发生数据 TLB 保护违反异常。数据 TLB 保护违反异常的硬件进行的处理与软件进行的处理如下所述：

- 硬件处理

  数据TLB保护违反异常时，硬件进行以下处理：

  1. 将已产生异常的虚拟地址的VPN设定至PTEH。
  2. 将已产生异常的虚拟地址设定至TEA。
  3. 读取时将异常码H'0A0设定至EXPEVT写入时将异常码H'0C0设定至EXPEVT （OCBP、 OCBWB：读取， OCBI、 MOVCA.L：写入）。
  4. 将指向发生异常的指令地址的PC值设定至SPC。如果异常在延迟槽发生，则将指向延迟转移指令的地址的PC值设定至SPC。
  5. 将发生异常时的SR的内容设定至SSR。将此时的R15设定至SGR。
  6. 将SR的MD位置1，转换至特权模式。
  7. 将SR的BL位置1，屏蔽此后的异常请求。
  8. 将SR的RB位置1。
  9. 转移至将偏移H'0000 0100添加至VBR内容的地址，并开始数据TLB保护违反异常处理程序。

- 软件处理 （数据TLB保护违反异常处理程序）

  请解决数据TLB保护违反、执行从异常处理返回的指令 （RTE）、终止异常处理程序、使控制返回正常流程。但是，请在LDTLB指令的下一个指令以后发行RTE指令。

7.5.7 初始页写入异常

尽管与进行数据存取 (写入) 的虚拟地址匹配的地址转换信息存在于 UTLB 入口中，并且也允许存取权，但 D 位为 0 时发生初始页写入异常。初始页写入异常的硬件进行的处理与软件进行的处理如下所述：

- 硬件处理

  初始页写入异常时，硬件进行以下处理：

  1. 将已产生异常的虚拟地址的VPN设定至PTEH。
  2. 将已产生异常的虚拟地址设定至TEA。
  3. 将异常码H'080设定至EXPEVT。
  4. 将指向发生异常的指令地址的PC值设定至SPC。如果异常在延迟槽发生，则将指向延迟转移指令的地址的PC值设定至SPC。
  5. 将发生异常时的SR的内容设定至SSR。将此时的R15设定至SGR。
  6. 将SR的MD位置1，转换至特权模式。
  7. 将SR的BL位置1，屏蔽此后的异常请求。
  8. 将SR的RB位置1。
  9. 转移至将偏移H'0000 0100添加至VBR内容的地址，并开始初始页写入异常处理程序。

- 软件处理 （初始页写入异常处理程序）

  软件负责进行如下处理：

  1. 从外部存储器查找到必要的页表入口。
  2. 请将1写入外部存储器的页表入口的D位。
  3. 将保存在外部存储器的页表入口的PPN、 PR、 SZ、 C、 D、 WT、 SH、 V位的值写入PTEL。
  4. 通过软件指定在入口调换中被调换的入口时，将该值写入MMUCR的URC。此时，在URC超过URB的情况下，请在LDTLB指令发行后变更为适当的值。
  5. 执行LDTLB指令，将PTEH、 PTEL的内容写入UTLB。
  6. 最后，执行从异常处理返回的指令 （RTE），终止异常处理程序、使控制返回正常流程。但是，请在LDTLB指令的下一个指令以后发行RTE指令。

7.6 存储器分配 TLB 的结构

为了通过软件来管理 ITLB 及 UTLB，在特权模式时，可通过 MOV 指令从 P2 区域的程序读取、写入 ITLB 与 UTLB 的内容。从其他区域的程序进行存取时，不保证运行。

请在存储器分配 TLB 存取后、进行对 P2 区域以外的存取 （含取指令）前，执行以下 1 ～ 3 中的任意一项：

1. 请执行通过RTE指令产生的转移。此时，转移目标也可为P2区域以外。
2. 请对任意地址 （不能进行高速缓存操作区域的亦可）执行ICBI指令。
3. 存储器分配TLB存取前预先设定为IRMCR.MT=0 （初始值）时，无需特定的指令顺序。但是因为此方法必须从取指令重新执行MMUCR更新指令的下一个指令， CPU的处理效率会降低，所以请注意。

但是，在今后的 SuperH 系列中有可能不保证方法 3。为保证在今后的 SuperH 系列中的兼容性，推荐使用 1 或 2。

将 ITLB 及 UTLB 分配至虚拟地址空间的 P4 区域。在 ITLB 中可将 VPN、 V、 ASID 作为地址阵列来存取，将 PPN、 V、 SZ、 PR、 C、 SH 作为数据阵列来存取。

在 UTLB 中可将 VPN、D、V、ASID 作为地址阵列来存取，将 PPN、V、SZ、PR、C、D、WT、SH 作为数据阵列来存取。 V 与 D 可从地址阵列与数据阵列双方进行存取。存取长度仅限长字长度。不可对此区域进行取指令。对于保留位请作为写入值指定 0。不保证读取值。

7.6.1 ITLB 地址阵列

将 ITLB 的地址阵列分配至 P4 区域的 H'F200 0000 ～ H'F2FF FFFF。存取地址阵列时，需要指定 32 位地址部分 （读取 / 写入时）与 32 位数据部分 （写入时）。在地址部分指定用于选择存取的入口的信息，在数据部分指定写入地址阵列的 VPN、 V、 ASID。

地址部分中， [31:24] 为表示 ITLB 地址阵列的 H'F2，通过 [9:8] 选择入口。为了进行长字存取，地址部分 [1:0] 请指定 0。

数据部分中， [31:10] 表示 VPN， [8] 表示 V， [7:0] 表示 ASID。

可对 ITLB 地址阵列进行以下 2 种操作：

1. ITLB地址阵列　读取
   从设定于地址部分的入口所对应的ITLB入口，将VPN、 V、 ASID读取至数据部分。
2. ITLB地址阵列　写入
   对设定于地址部分的入口所对应的ITLB入口，写入在数据部分指定的VPN、 V、 ASID。

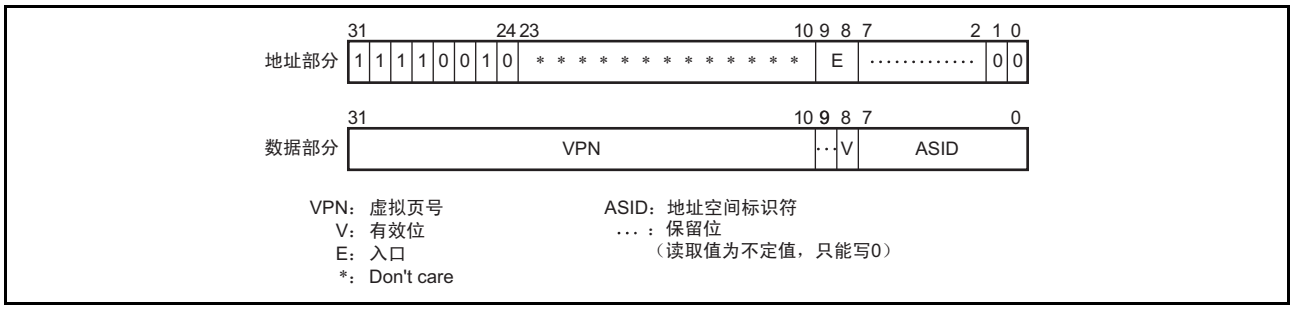


图 7.12 存储器分配 ITLB 地址阵列

7.6.2 ITLB 数据阵列

将 ITLB 的数据阵列分配至 P4 区域的 H'F300 0000 ～ H'F37F FFFF。存取数据阵列时，需要指定 32 位地址部分 （读取 / 写入时）与 32 位数据部分 （写入时）。在地址部分指定用于选择存取的入口的信息，在数据部分指定写入数据阵列 1 的 PPN、 V、 SZ、 PR、 C、 SH。

地址部分中， [31:23] 为表示 ITLB 数据阵列的 H'F30，通过 [9:8] 来选择入口。

数据部分中， [28:10] 表示 PPN， [8] 表示 V， [7]、 [4] 表示 SZ， [6] 表示 PR， [3] 表示 C， [1] 表示 SH。

可对 ITLB 数据阵列进行以下 2 种操作：

1. ITLB数据阵列　读取
   从设定于地址部分的入口所对应的ITLB入口，将PPN、 V、 SZ、 PR、 C、 SH读取至数据部分。
2. ITLB数据阵列　写入
   对设定于地址部分的入口所对应的ITLB入口，写入在数据部分指定的PPN、 V、 SZ、 PR、 C、 SH。

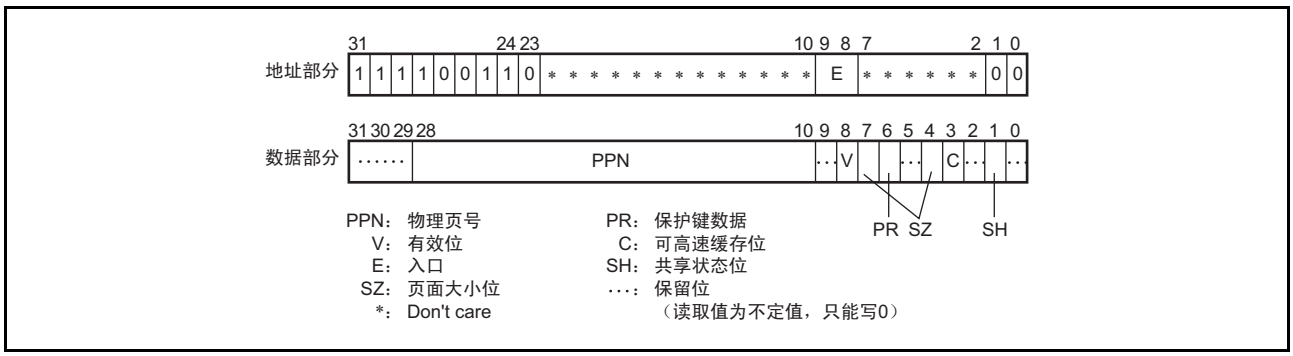


图 7.13　存储器分配 ITLB 数据阵列

7.6.3 UTLB 地址阵列

将 UTLB 的地址阵列分配至 P4 区域的 H'F600 0000 ～ H'F60F FFFF。存取地址阵列时，需要指定 32 位地址部分 （读取 / 写入时）与 32 位数据部分 （写入时）。在地址部分指定用于选择存取的入口的信息，在数据部分指定写入地址阵列的 VPN、 D、 V、 ASID。

地址部分中， [31:20] 为表示 UTLB 地址阵列的 H'F60，通过 [13:8] 来选择入口。地址部分 [7] 的联想位（A 位）指定有无写入 UTLB 地址阵列时的地址比较。

数据部分中， [31:10] 表示 VPN， [9] 表示 D， [8] 表示 V， [7:0] 表示 ASID。

可对 UTLB 地址阵列进行以下 3 种操作：

1. UTLB地址阵列 读取
   从设定于地址部分的入口所对应的UTLB入口，将VPN、 D、 V、 ASID读取至数据部分。读取时，地址部分所指定的联想位无论为1还是为0，均不进行联想操作。
2. UTLB地址阵列 写入 （无联想）
   对设定于地址部分的入口所对应的UTLB入口，写入数据部分所指定的VPN、 D、 V、 ASID。请将地址部分的A位置0。
3. UTLB地址阵列 写入 （有联想）
   地址部分的A位写入1时，使用在数据部分中指定的VPN与PTEH.ASID，并与UTLB的所有入口之间进行比较。比较依据普通地址比较规则，但是UTLB产生错误时，不发生异常，为空操作。通过比较，在数据部分指定的VPN所对应的UTLB入口存在时，对该入口写入数据部分所指定的D与V。此联想操作也可对ITLB同时进行， ITLB中存在匹配的入口时，对该入口写入V。通过在UTLB的比较，即使为空操作，如果在ITLB匹配，则仅写入ITLB。另外， UTLB与ITLB双方匹配时， UTLB的信息也可写入ITLB。

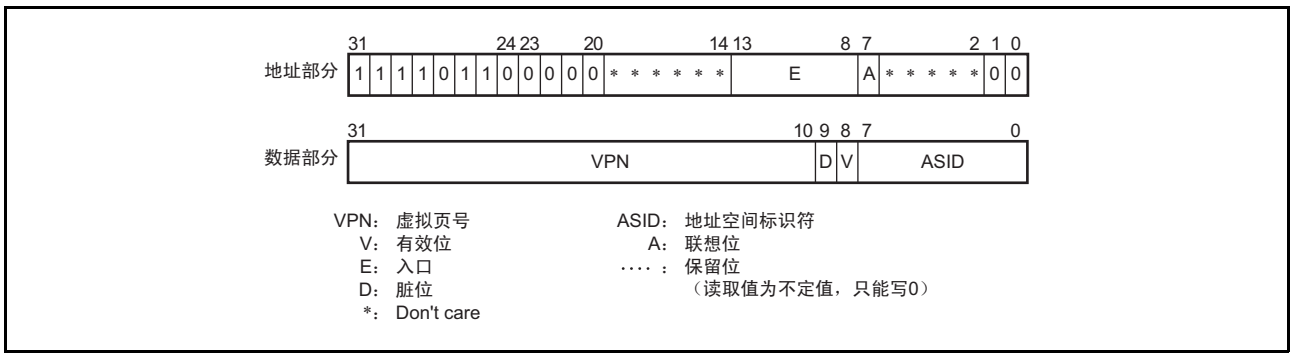


图 7.14 存储器分配 UTLB 地址阵列

7.6.4 UTLB 数据阵列

将 UTLB 的数据阵列分配至 P4 区域的 H'F700 0000 ～ H'F70F FFFF。存取数据阵列时，需要指定 32 位地址部分 （读取 / 写入时）与 32 位数据部分 （写入时）。在地址部分指定用于选择存取的入口的信息，在数据部分指定写入数据阵列的 PPN、 V、 SZ、 PR、 C、 D、 SH、 WT。

地址部分中， [31:20] 为表示 UTLB 数据阵列的 H'F70，通过 [13:8] 来选择入口。

数据部分中，[28:10] 表示 PPN，[8] 表示 V，[7]、[4] 表示 SZ，[6:5] 表示 PR，[3] 表示 C，[2] 表示 D，[1] 表示 SH， [0] 表示 WT。

可对 UTLB 数据阵列进行以下 2 种操作：

1. UTLB数据阵列　读取
   从设定于地址部分的入口所对应的UTLB入口，将PPN、V、SZ、PR、C、D、SH、WT读取至数据部分。
2. UTLB数据阵列　写入
   对设定于地址部分的入口所对应的UTLB入口，写入数据部分所指定的PPN、 V、 SZ、 PR、 C、 D、 SH、 WT。

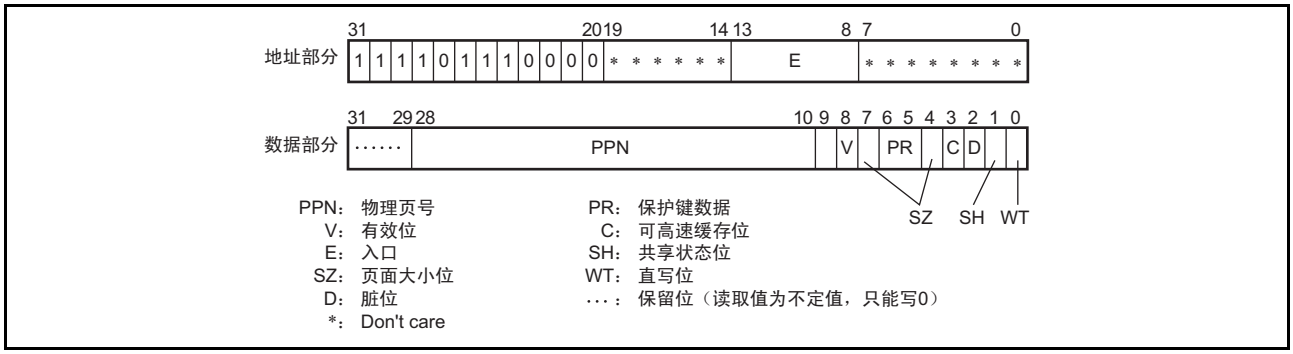


图 7.15　存储器分配 UTLB 数据阵列

7.7 32 位地址扩展模式

通过将 PASCR 寄存器的 SE 位置 1， SH-4A 可从处理 29 位物理地址空间的 29 位地址模式变更为处理 32 位物理地址空间的 32 位地址扩展模式。

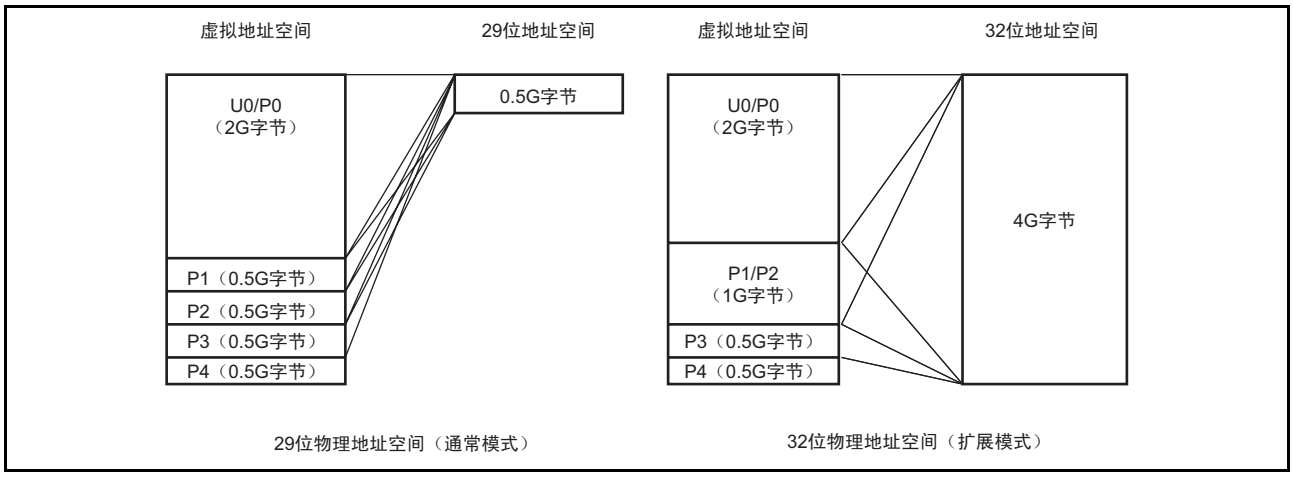


图 7.16 物理地址空间 （32 位地址扩展模式）

7.7.1 32 位地址扩展模式概要

32 位地址扩展模式下，导入将 29 位地址模式下为地址转换目标以外的 P1/P2 区域的虚拟地址映射至 32 位物理地址空间的特权空间映射缓冲器 （PMB）。另外，有关现有的 TLB （UTLB/ITLB）地址转换目标区域，扩展 UTLB/ITLB 的 PPN 字段的高 3 位， TLB 转换后的地址可处理 32 位物理地址。

另外，至于高速缓存操作， 29 位地址模式下 P1 区域可固定地进行高速缓存， P2 区域不可进行高速缓存，但是 32 位地址扩展模式下， P1、 P2 区域均依据 PMB 的 C 位及 WT 位。

7.7.2 向 32 位地址扩展模式的转换

SH-4A 在上电复位后为 29 位地址模式。通过将 1 写入 PASCR 寄存器的 SE 位，向 32 位地址扩展模式转移。 32 位地址扩展模式下 MMU 操作如下：

1. MMUCR.AT=0 时， U0/P0/P3 区域的虚拟地址直接为 32 位物理地址。 P1/P2 区域的地址依据 PMB 映射的信息进行地址转换。
2. MMUCR.AT=1 时， U0/P0/P3 区域的虚拟地址依据 TLB 转换信息转换为 32 位物理地址。 P1/P2 区域的地址依据 PMB 映射的信息进行地址转换。
3. 控制寄存器区域 （H'FC00 0000～H'FFFF FFFF）中，不管 MMUCR.AT，物理地址的 [31:29] 均为 B'111。将控制寄存器区域注册至 UTLB 并进行存取时，请将 B'111 设定至 PPN[31:29]。

7.7.3 特权空间映射缓冲器 （PMB）结构

32 位地址扩展模式下， P1/P2 区域的虚拟地址依据 PMB 映射信息进行地址转换。 PMB 有 16 个入口，各入口结构如下。

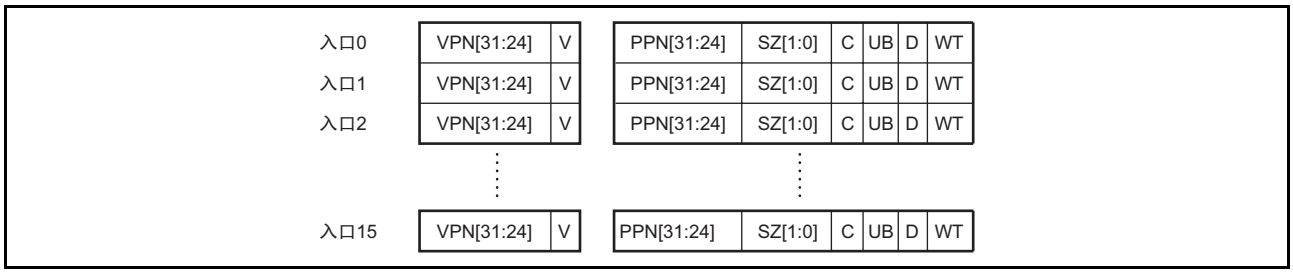


图 7.17 PMB 的结构

【符号说明】

VPN： 虚拟页码
16M 字节页时，虚拟地址的高 8 位
64M 字节页时，虚拟地址的高 6 位
128M 字节页时，虚拟地址的高 5 位
512M 字节页时，虚拟地址的高位 3 位

SZ： 页面大小位
指定页面大小。
00：16M 字节页
01：64M 字节页
10：128M 字节页
11：512M 字节页

V： 有效位
表示入口是否有效。
0：无效
1：有效
上电复位时清 0。
手动复位时不变。

PPN： 物理页码
物理地址的高 8 位
16M 字节页时， PPN[31:24] 有效
64M 字节页时， PPN[31:26] 有效
128M 字节页时， PPN[31:27] 有效
512M 字节页时， PPN[31:29] 有效

C： 可高速缓存位
表示页面可否进行高速缓存。
0：不可进行高速缓存
1：可进行高速缓存

WT： 直写位
指定向高速缓存的写入模式。
0：备份模式
1：直写模式

UB： 缓冲写入位
指定是否进行缓冲写入。
0：缓冲写入 （不等待写入完成就开始后续指令的数据存取）
1：无缓冲写入 （等待写入完成再开始后续指令的数据存取）

7.7.4 PMB 功能

SH-4A 支持以下的 PMB 功能：

1. 仅可通过存储器分配写入进行向PMB的写入。不可注册至LDTLB。
2. 存取PMB映射目标的P1/P2区域的地址必须通过软件来保证PMB注册。在PMB中无转换信息的P1/P2区域的地址有存取时， SH-4A为TLB复位。此时，在TEA保存对TLB复位原因的P1/P2区域存取的地址，在EXPEVT保存代码H'140。
3. SH-4A不保证PMB引起多重命中时的运行。特别注意软件并注册PMB映射信息。
4. PMB中无联想写入功能。
5. PMB中不存在PR字段，不可实施读/写保护。由于PMB的地址转换目标为P1/P2地址，所以通过用户模式下的存取产生地址错误异常。
6. 通过硬件ITLB未命中处理将UTLB与PMB双方的入口混合注册至ITLB。但是，可通过VPN[31:30]是否为10来识别从UTLB注册或从PMB注册。 PMB的入口注册至ITLB时，将H'00注册至不存在于PMB中的字段的ASID，将01注册至PR，将1注册至SH。

7.7.5 存储器分配 PMB 的结构

为通过软件管理 PMB，特权模式时，可通过 MOV.L 指令从 P1/P2 区域的程序读取、写入 PMB 的内容。将 PMB 的地址阵列分配至 P4 区域的 H'F610 0000 ～ H'F61F FFFF，将 PMB 的数据阵列分配至 P4 区域的 H'F710 0000 ～ H'F71F FFFF。在 PMB 中，可将 VPN、 V 作为地址阵列进行存取，可将 PPN、 V、 SZ、 C、 WT、 UB 作为数据阵列进行存取。 V 可从地址阵列与数据阵列双方进行存取。进行 PMB 存储器分配存取的程序配置于设定为 PMB.C=0 的页面区域。

1. PMB地址阵列读取
   作为地址，在[31:20]指定表示PMB地址阵列的H'F61，在[11:8]指定入口后，进行存储器读取时，在[31:24]读取VPN，在[8]读取V。
2. PMB地址阵列写入
   作为地址，在[31:20]指定表示PMB地址阵列的H'F61，在[11:8]指定入口；作为数据，在[31:24]指定VPN，在[8]指定V后，进行存储器写入时，可写入指定的入口。
3. PMB数据阵列读取
   作为地址，在[31:20]指定表示PMB数据阵列的H'F71，在[11:8]指定入口后，进行存储器读取时，在[31:24]读取PPN，在[9]读取UB，在[8]读取V，在[7]、 [4]读取SZ，在[3]读取C，在[0]读取WT。
4. PMB数据阵列写入
   作为地址，在[31:20]指定表示PMB数据阵列的H'F71，在[11:8]指定入口；作为数据，在[31:24]指定PPN，在[9]指定UB，在[8]指定V，在[7]、 [4]指定SZ，在[3]指定C，在[0]指定WT后，进行存储器写入时，可写入指定的入口。

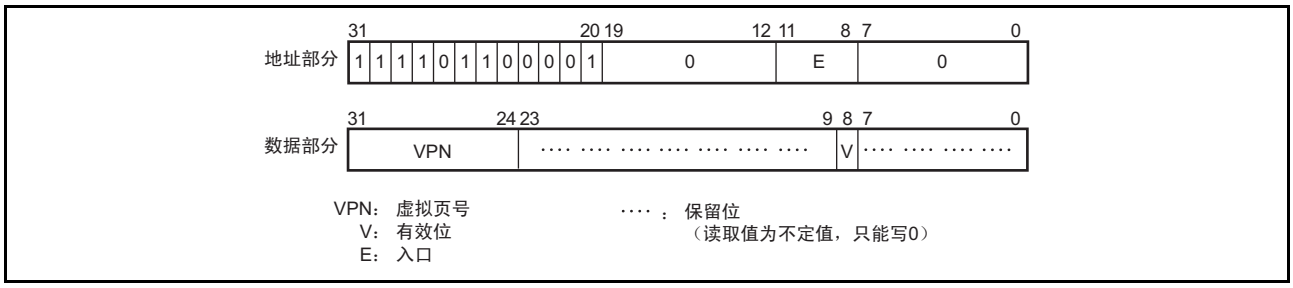


图 7.18 存储器分配 PMB 地址阵列

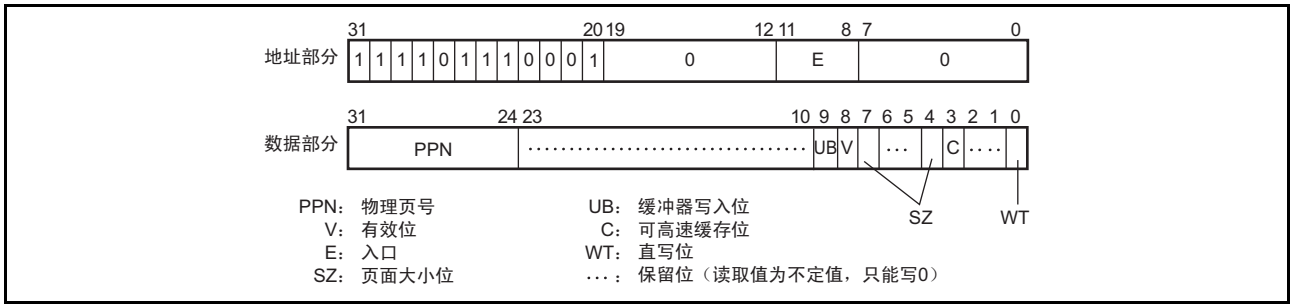


图 7.19 存储器分配 PMB 数据阵列

7.7.6 使用 32 位地址扩展模式时的注意事项

使用 32 位地址扩展模式时，本章节中记述的事项如下进行扩展或变更，所以请注意。

(1) PASCR.SE

在控制寄存器 PASCR[31] 追加 SE 位。另外， UB[6:0] 无效 （UB[7] 即使在 32 位地址扩展模式仍然有效）。

对 P1/P2 区域写入时通过 PMB 的 UB 位控制是否为缓冲写入。 MMU 允许时，通过 TLB 的 UB 位控制对 P0/P3/U0 区域的写入。 MMU 禁止时，通常为缓冲写入。

位	位名称	初始值	R/W	说明
31	SE	0	R/W	地址模式 0：29 位地址模式 1：32 位地址扩展模式
30 ～ 8	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
7 ～ 0	UB	均为 0	R/W	每个区域 （64M 字节）的缓冲写入控制 不使用高速缓存写入时，对每个区域指定 CPU 是否等待写入完成。 0：CPU 不等待写入完成 1：CPU 在写入完成前停止并等待。 UB [7]：控制寄存器区域的缓冲写入控制 UB [6:0]：每个区域 （64M 字节）的缓冲写入控制 （32 位地址扩展模式下无效）

(2) ITLB

ITLB 的 PPN 字段向 [31:10] 扩展。

(3) UTLB

在 UTLB 的各入口追加与 PMB 的 UB 位相同含义的 UB 位。

UB：缓冲写入位

指定是否进行缓冲写入。

0：缓冲写入 （不等待写入完成就开始后续处理）

1：无缓冲写入 （等待写入完成再开始后续处理）

在存储器分配 TLB 存取中， UB 位可通过数据阵列的位 [9] 进行读取 / 写入。

(4) PTEL

与 UTLB 同样，在 PTEL 寄存器的位 [9] 追加与 PMB 的 UB 位相同含义的 UB 位。

可通过 LDTLB 指令将此 UB 位写入 UTLB 的 UB 位。另外， PPN 字段扩展为 [31:10]。

(5) CCR.CB

CCR 寄存器的 CB 位无效。对 P1 区域的可高速缓存写入为备份模式还是直写模式，依据 PMB 的 WT 位。

(6) IRMCR.MT

IRMCR 的 MT 位即使对存储器分配 PMB 写入仍然有效。

(7) QACR0、 QACR1

QACR0、 QACR1 寄存器的 AREA0[4:2]、 AREA[4:2] 分别扩展为 AREA0[7:2]、 AREA1[7:2]，对应物理地址 31 ～ 26。

(8) LSA0、 LSA1、 LDA0、 LDA1

L0SADR、 L1SADR、 L0DADR、 L1DADR 分别扩展为 [31:0]。

另外，使用 32 位地址模式时，软件注意以下几点：

1. 通过上电复位或手动复位后的引导程序， SE 位的转换仅支持从 0 到 1 的转换。
2. SE 位转换后，由于配置此程序的区域本身为 PMB 地址转换目标，所以需要在转换 SE 位之前注册至 PMB。异常处理程序等，有可能对 P1/P2 区域存取的地址也必须注册至 PMB。
3. 转换 SE 位的 MOV.L 指令前，某个操作数存储器存取引起外部存储器存取时，请将在两个地址模式下存取的外部存储空间地址设定为相同。
4. 注册 PMB 时，请注意 V 位映射至地址阵列与数据阵列双方的情况。即第一次向一方写入时选择 V=0，第二次向另一方写入时选择 V=1。

第 8 章 高速缓存

SH-4A 内置用于指令的 32K 字节指令的高速缓存 （IC）与用于数据的 32K 字节的操作数高速缓存（OC）。

【注】 有关指令高速缓存、操作数高速缓存的容量，请参考产品的硬件手册。本手册说明了各个 32K 字节的情况。

8.1 特点

高速缓存的特点如表 8.1 所示。

SH-4A 为了进行向外部存储器的快速写入，支持 32 字节 ×2 存储队列 （SQ）。 SQ 的特点如 8.2 所示。

表 8.1 高速缓存的特点

项目	指令高速缓存	操作数高速缓存
容量	32K 字节高速缓存	32K 字节高速缓存
方式	4 路成组相联、虚拟地址变址 / 物理地址标记符	4 路成组相联、虚拟地址变址 / 物理地址标记符
线长度	32 字节	32 字节
入口数	256 个入口 / 通路	256 个入口 / 通路
写入方式	—	可选择备份 / 直写
置换方式	LRU （Least Recently Used）算法	LRU （Least Recently Used）算法

表 8.2 存储队列的特点

项目	存储队列
容量	2×32 字节
地址	H'E000 0000 ～ H'E3FF FFFF
写入	存储指令 （1 个周期写入）
回写	预取指令 （PREF 指令）
存取权	禁止 MMU 时：由 MMU 控制寄存器 （MMUCR）的 SQMD 位决定 允许 MMU 时：由各页 PR 决定

SH-4A 的操作数高速缓存为 4 路成组相联方式，各个通路均由 256 个高速缓存线构成。操作数高速缓存的结构如图 8.1 所示。

指令高速缓存为 4 路成组相联方式，各个通路均由 256 个高速缓存线构成。指令高速缓存的结构如图 8.2 所示。

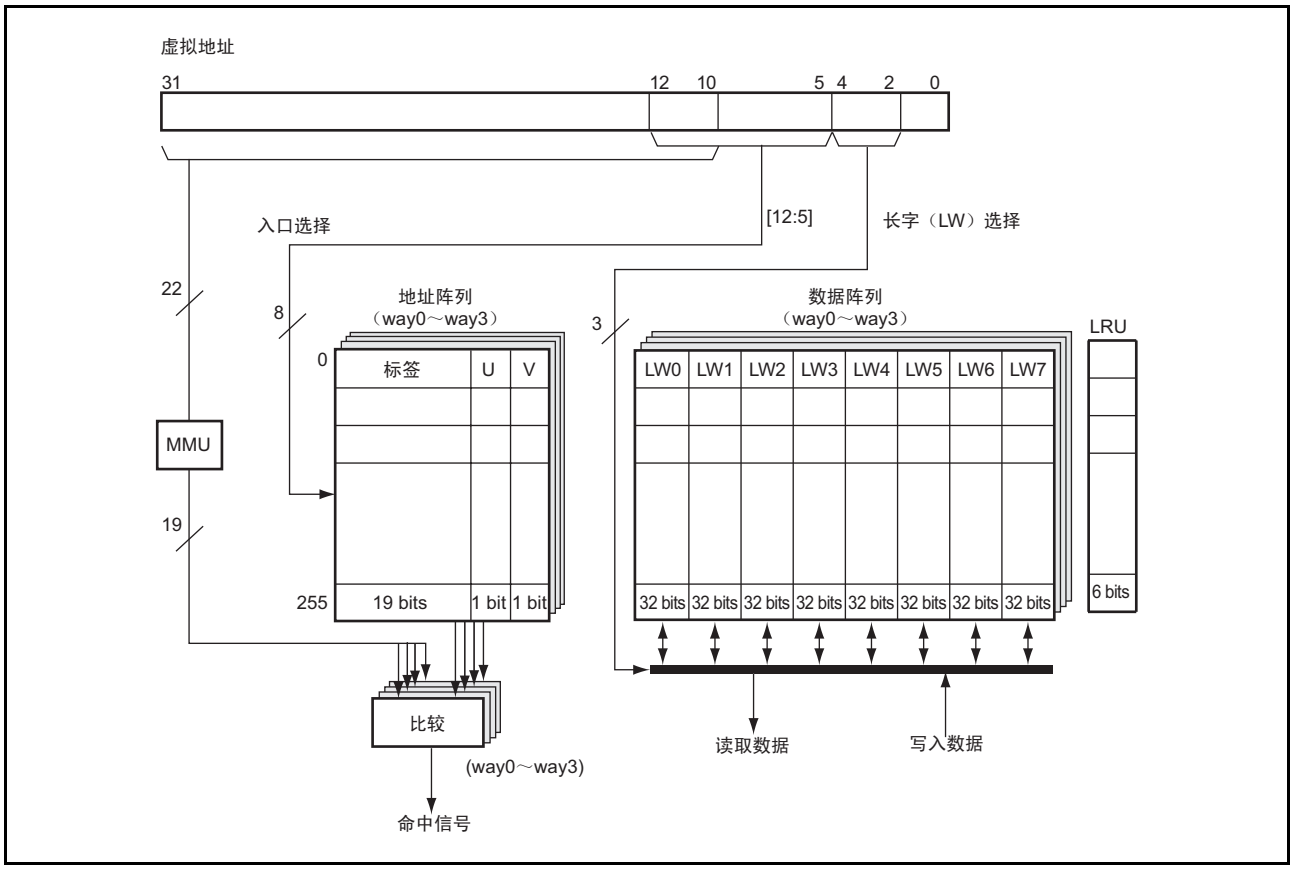


图 8.1 操作数高速缓存的结构

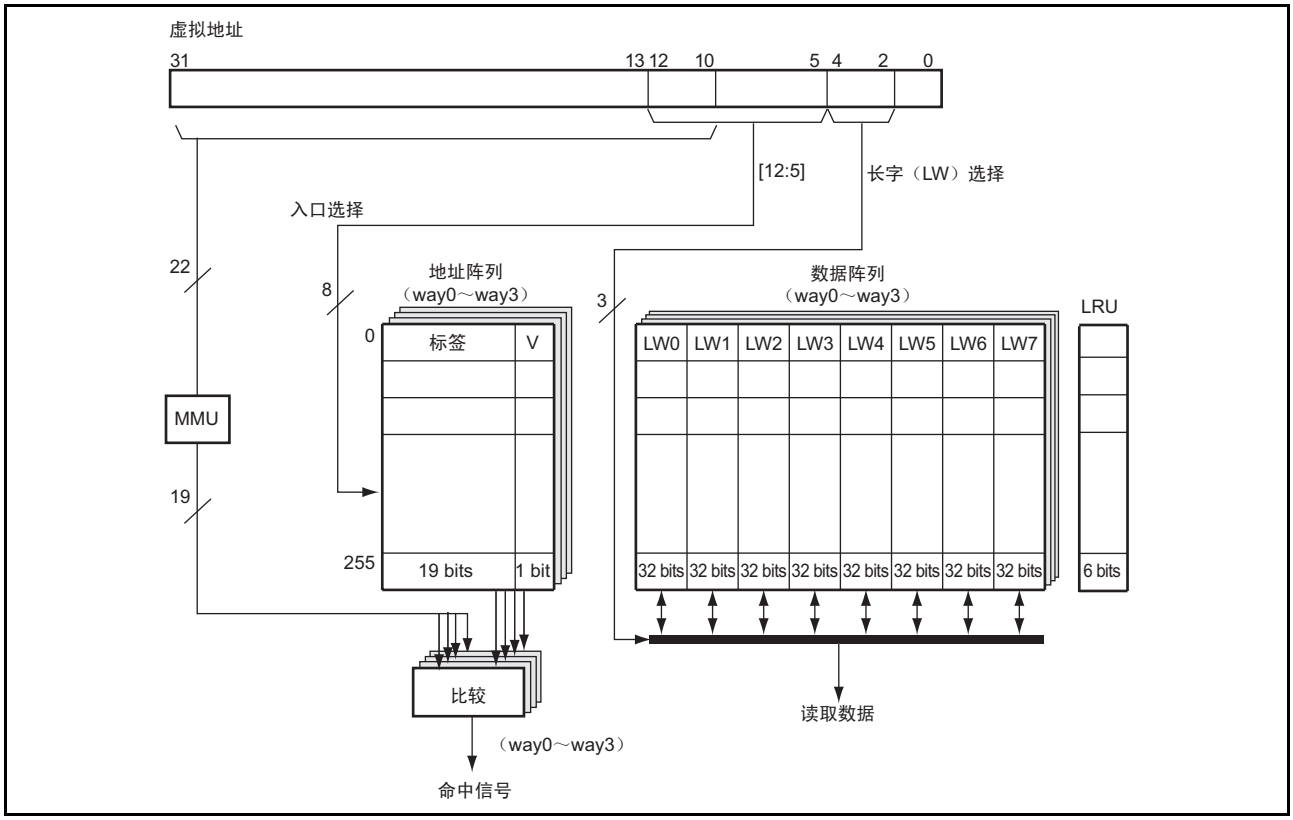


图 8.2 指令高速缓存的结构

(1) 标记符

保存进行高速缓存的数据线的物理地址 29 位的高 19 位。不能通过上电复位、手动复位初始化标记符。

(2) V 位 （有效位）

表示是否将有效数据保存至高速缓存线。该位为 1 时，此高速缓存线的数据有效。通过上电复位将 V 位初始化为 0，通过手动复位保持值。

(3) U 位 （脏位）

在备份模式下使用高速缓存的过程中，将数据写入高速缓存线时， U 位为 1。即：U 位表示高速缓存线中的数据与外部存储器中的数据不一致。通过存取存储器分配高速缓存 （参考 **“8.6 存储器分配高速缓存的结构”**），只要不改写 U 位，在直写模式下使用高速缓存的过程中， U 位就不会为 1。通过上电复位将 U 位初始化为 0，通过手动复位保持值。

(4) 数据部分

将每条高速缓存线的 32 字节 （256 位）数据保存至数据部分。不能通过上电复位及手动复位初始化数据阵列。

(5) LRU 部分

4 路成组相联方式下，入口地址可在高速缓存中最多注册 4 个相同的数据。注册入口时， LRU 位表示注册到 4 个通路中的哪个通路。 LRU 位由各入口 6 位构成，通过硬件来控制。作为通路选择的算法，使用选择最初存取的通路的 LRU （Least Recently Used）算法。通过上电复位将 LRU 位初始化为 0，不通过手动复位初始化。不能通过软件读取 LRU 位。

8.2 寄存器说明

高速缓存相关的寄存器如下所示：

表 8.3 寄存器结构

名称	简称	R/W	P4 区域地址 *	区域 7 地址 *	大小
高速缓存控制寄存器	CCR	R/W	H'FF00 001C	H'1F00 001C	32
队列地址控制寄存器 0	QACR0	R/W	H'FF00 0038	H'1F00 0038	32
队列地址控制寄存器 1	QACR1	R/W	H'FF00 003C	H'1F00 003C	32
内部存储器控制寄存器	RAMCR	R/W	H'FF00 0074	H'1F00 0074	32

【注】 * P4 区域地址为使用虚拟地址空间的 P4 区域时的地址。区域 7 地址为使用 TLB，从物理地址空间的区域 7 存取的地址。

表 8.4 各处理模式下的寄存器状态

名称	简称	上电复位	手动复位	睡眠	待机
高速缓存控制寄存器	CCR	H'0000 0000	H'0000 0000	保持	保持
队列地址控制寄存器 0	QACR0	不定	不定	保持	保持
队列地址控制寄存器 1	QACR1	不定	不定	保持	保持
内部存储器控制寄存器	RAMCR	H'0000 0000	H'0000 0000	保持	保持

8.2.1 高速缓存控制寄存器 （CCR）

CCR 进行高速缓存运行模式的选择、高速缓存所有入口的无效化及向高速缓存的写入模式的选择。

必须通过不可进行高速缓存操作的 P2 区域的程序来改写 CCR。请在 CCR 更新后，存取可执行高速缓存操作的区域 （包含取指令）之前，执行以下 1 ～ 3 中的任意一项。

1. 请执行通过RTE指令产生的转移。此时，转移目标亦可为可高速缓存区域。
2. 请对任意地址 （亦可为不可执行高速缓存操作的区域）执行ICBI指令。
3. CCR更新前预先设定为IRMCR.R2=0 （初始值）时，无需特定的指令顺序。但由于此方法必须从取指令开始重新执行CCR更新指令的下一个指令，会降低CPU的处理功能，所以请注意。

但在今后的 SuperH 系列中有可能不保证方法 3。为保证在今后的 SuperH 系列中的兼容性，推荐使用 1 或 2。

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	ICI	—	—	ICE	—	—	—	—	OCI	CB	WT	OCE
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R/W	R	R	R/W	R	R	R	R	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 12	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
11	ICI	0	R/W	IC 无效化位 给该位写入 1 时，将 IC 的所有入口的 V 位置 0。读取时通常读取为 0。
10、9	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
8	ICE	0	R/W	IC 有效位 选择使用 IC。但进行地址转换时，如果页面管理信息的 C 位也不为 1，则不可使用 IC。 0：不使用 IC 1：使用 IC
7 ～ 4	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
3	OCI	0	R/W	OC 无效化位 给该位写入 1 时，将 OC 的所有入口的 V、 U 位置 0。读取时通常读取为 0。
2	CB	0	R/W	备份位 表示向 P1 区域高速缓存的写入模式。 0：直写模式 1：备份模式
1	WT	0	R/W	直写模式 表示向 P0、 U0、 P3 区域高速缓存的写入模式。但进行地址转换时，优先页面管理信息的 WT 位的值。 0：备份模式 1：直写模式
0	OCE	0	R/W	OC 有效位 选择使用 OC。但进行地址转换时，如果页面管理信息的 C 位也不为 1，则不可使用 OC。 0：不使用 OC 1：使用 OC

8.2.2 队列地址控制寄存器 0 （QACR0）

MMU 为禁止状态时， QACR0 设定映射存储队列 0 （SQ0）的区域。

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	—	—	AREA0			—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	—	—	—	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R	R

位	位名称	初始值	R/W	说明
31 ～ 5	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
4 ～ 2	AREA0	不定	R/W	MMU 为禁止状态时，生成 SQ0 的物理地址 28 ～ 26。
1、 0	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。

8.2.3 队列地址控制寄存器 1 （QACR1）

MMU 为禁止状态时， QACR1 设定映射存储队列 1 （SQ1）的区域。

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	—	—	AREA1			—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	—	—	—	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R/W	R/W	R/W	R	R

位	位名称	初始值	R/W	说明
31 ～ 5	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
4 ～ 2	AREA1	不定	R/W	MMU 为禁止状态时，生成 SQ1 的物理地址 28 ～ 26。
1、 0	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。

8.2.4 内部存储器控制寄存器 （RAMCR）

RAMCR 控制 IC 及 OC 的通路数。

必须通过不可进行高速缓存操作的 P2 区域的程序来改写 RAMCR。请在 RAMCR 更新后，存取可执行高速缓存操作的区域或 L 存储器区域 （包含取指令）之前，执行以下 1 ～ 3 中的任意一项。

1. 请执行通过RTE指令产生的转移。此时，转移目标亦可为不可执行高速缓存操作的区域或L存储器区域。
2. 对任意地址 （亦可为不可执行高速缓存操作的区域）执行ICBI指令。
3. RAMCR更新前，预先设定为IRMCR.R2=0 （初始值）时，无需特定的指令顺序。但由于此方法必须从取指令开始重新执行RAMCR更新指令的下一个指令，会降低CPU的处理功能，所以请注意。

但在今后的 SuperH 系列中有可能不保证方法 3。为保证在今后的 SuperH 系列中的兼容性，推荐使用 1 或 2。

位：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

位：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	RMD	RP	IC2W	OC2W	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R	R	R	R	R	R

位	位名称	初始值	R/W	说明
31 ～ 10	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
9	RMD	0	R/W	内部存储器存取模式位 详细内容请参考 “9.4 L 存储器的保护功能”。
8	RP	0	R/W	内部存储器保护有效位 详细内容请参考 “9.4 L 存储器的保护功能”。
7	IC2W	0	R/W	IC 2 路模式位 0：IC 为 4 路运行 1：IC 为 2 路运行 详细内容请参考 “8.4.3 IC 2 路模式”。
6	OC2W	0	R/W	OC 2 路模式位 0：OC 为 4 路运行 1：OC 为 2 路运行 详细内容请参考 “8.3.6 OC 2 路模式”。
5 ～ 0	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。

8.3 操作数高速缓存的运行说明

8.3.1 读取操作

操作数高速缓存 （OC）有效 （CCR.OCE=1）且从可执行高速缓存操作的区域读取数据时， OC 进行如下操作：

1. 从通过虚拟地址的位[12:5]进行变址的各通路的高速缓存线读取标记符、 V位、 U位及LRU位。
2. 将通过MMU转换虚拟地址的物理地址的位[28:10]与从各通路读取的标记符作比较：
   - 标记符一致且V位为1的通路存在时 → 3.
   - 标记符一致且V位为1的通路不存在，并且通过LRU位选择的置换目标通路的U位为0时 → 4.
   - 标记符一致且V位为1的通路不存在，并且通过LRU位选择的置换目标通路的U位为1时 → 5.
3. 高速缓存命中
   从发生命中的通路的数据部分，根据存取长度通过虚拟地址的位[4:0] 读取进行变址的数据。另外，为了使发生命中的通路为最新，需更新LRU位。
4. 高速缓存未命中 （无回写）
   从对应虚拟地址的物理地址空间，向置换目标通路的高速缓存线读取数据。从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据，在相应的数据到达高速缓存时，返回向CPU读取的数据。读取剩余的高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位，将0写入U位。另外，为了使置换的通路为最新，需更新LRU位。
5. 高速缓存未命中 （有回写）
   将置换目标通路的高速缓存线的标记符与数据部分保存至回写缓冲器。此后，从对应虚拟地址的物理地址空间向置换目标通路的高速缓存线读取数据。从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据，在相应的数据到达高速缓存时，返回向CPU读取的数据。读取剩余的高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位，将0写入U位。另外，为了使置换的通路为最新，需更新LRU位。此后，将回写缓冲器的数据回写至外部存储器。

8.3.2 预取操作

操作数高速缓存 （OC）有效 （CCR.OCE=1）且从可执行高速缓存操作的区域将数据预取至 OC 时， OC 进行如下操作：

1. 从通过虚拟地址的位[12:5]进行变址的各通路的高速缓存线读取标记符、 V位、 U位及LRU位。
2. 将通过MMU转换虚拟地址的物理地址的位[28:10]与从各通路读取的标记符作比较：
   - 标记符一致且V位为1的通路存在时 → 3.
   - 标记符一致且V位为1的通路不存在，并且通过LRU位选择的置换目标通路的U位为0时 → 4.
   - 标记符一致且V位为1的通路不存在，并且通过LRU位选择的置换目标通路的U位为1时 → 5.
3. 高速缓存命中
   为了使发生命中的通路成为最新，需更新LRU位。
4. 高速缓存未命中 （无回写）
   从对应虚拟地址的物理地址空间，向置换目标通路的高速缓存线读取数据。从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据，预取操作下， CPU不等待数据的到达，在读取高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位，将0写入U位。另外，为了使置换的通路为最新，需更新LRU位。
5. 高速缓存未命中 （有回写）
   将置换目标通路的高速缓存线的标记符与数据部分保存至回写缓冲器。此后，从对应虚拟地址的物理地址空间向置换目标通路的高速缓存线读取数据。从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据，预取操作中， CPU不等待数据到达，在读取高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位，将0写入U位。另外，为了使置换的通路为最新，需更新LRU位。此后，将回写缓冲器的数据回写至外部存储器。

8.3.3 写入操作

操作数高速缓存 （OC）有效 （CCR.OCE=1）且对可执行高速缓存操作的区域写入数据时， OC 进行如下操作：

1. 从通过虚拟地址的位[12:5]进行变址的各通路的高速缓存线读取标记符、 V位、 U位及LRU位。
2. 将通过MMU转换虚拟地址的物理地址的位[28:10]与从各通路读取的标记符作比较，及从成为目标区域的属性开始

	备份	直写
• 标记符一致且V位为1的通路存在时	→ 3.	→ 4.
• 标记符一致且V位为1的通路不存在，通过LRU位选择的置换目标通路的U位为0时	→ 5.	→ 7.
• 标记符一致且V位为1的通路不存在，通过LRU位选择的置换目标通路的U位为1时	→ 6.	→ 7.

3. 高速缓存命中 （备份）
   对于通过发生命中的通路的数据部分的、虚拟地址的位[4:0]进行变址的数据位置，根据存取长度进行写入。另外，将1写入U位时，为了使发生命中的通路为最新，需更新LRU位。
4. 高速缓存命中 （直写）
   对于通过发生命中的通路的数据部分的、虚拟地址的位[4:0]进行变址的数据位置，在根据存取长度进行写入的同时，也对对应虚拟地址的外部存储器进行写入。另外，为了使发生命中的通路为最新，需更新LRU位。此时，不更新U位。
5. 高速缓存未命中 （备份、无回写）
   对于通过置换目标通路的数据部分的、虚拟地址的位[4:0]进行变址的数据位置，根据存取长度进行写入。另外，从对应虚拟地址的物理地址空间，向置换目标通路的高速缓存线读取数据 （但除过已写完的发生高速缓存未命中的数据）。从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据。读取高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位，将1写入U位。另外，为了使置换的通路为最新，需更新LRU位。
6. 高速缓存未命中 （备份、有回写）
   将置换目标通路的高速缓存线的标记符与数据部分保存至回写缓冲器。对于通过置换目标通路的数据部分的、虚拟地址的位[4:0]进行变址的数据位置，根据存取长度进行写入。另外，从对应虚拟地址的物理地址空间，向置换目标通路的高速缓存线读取数据 （但除过已写完的发生高速缓存未命中的数据）。从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据。读取高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位，将1写入U位。另外，为了使置换的通路为最新，需更新LRU位。然后，将回写缓冲器的数据回写外部存储器。
7. 高速缓存未命中 （直写）
   通过指定的存取长度，向对应虚拟地址的外部存储器进行写入。此时，不向高速缓存进行写入。也不更新标记符、 V位、 U位、 LRU位。

8.3.4 回写缓冲器

通过高速缓存未命中，需要向外部存储器清除脏高速缓存的入口时，为了提高优向在高速缓存读取数据的功能， SH-4A 内置用于保存清除的高速缓存线的数据的回写缓冲器。回写缓冲器由高速缓存 1 条线的数据与清除目标的物理地址构成。

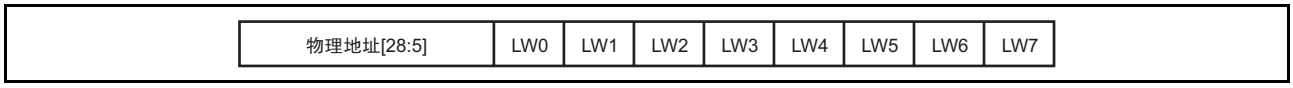


图 8.3 回写缓冲器的结构

8.3.5 直写缓冲器

在直写模式下进行数据写入及对不可执行高速缓存操作的区域进行写入操作时， SH-4A 内置用于保持写入数据的 64 位缓冲器。因此，完成向直写缓冲器的写入时， CPU 不等待向外部存储器的写入完成，就转移至下一个操作。

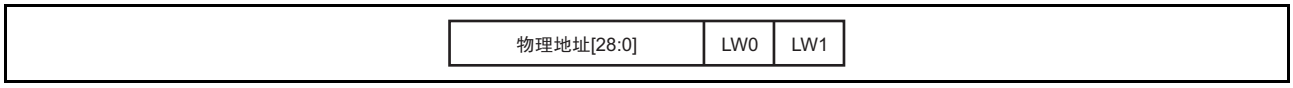


图 8.4 直写缓冲器的结构

8.3.6 OC 2 路模式

将 RAMCR 寄存器的 OC2W 位置 1 时，为仅使用 OC 的通路 0 及通路 1 的 OC 2 路模式，可降低功耗。本模式也包含存储器分配 OC 存取，仅使用通路 0 与通路 1。

请通过 P2 区域的程序来改写 OC2W 位。另外，改写时，已在 OC 注册有效线的情况下，改写 OC2W 位之前，根据需要通过软件进行回写之后，请将 1 写入 CCR 寄存器的 OCI 位，并使 OC 的所有入口无效。

8.4 指令高速缓存的操作说明

8.4.1 读取操作

指令高速缓存 （IC）有效 （CCR.ICE=1）且从可执行高速缓存操作的区域执行取指令时， IC 执行以下操作：

1. 从通过虚拟地址的位[12:5]进行变址的各通路的高速缓存线读取标记符、 V位及LRU位。
2. 将通过MMU转换虚拟地址的物理地址的位[28:10]与从各通路读取的标记符作比较，
   - 标记符一致且V位为1的通路存在时。 → 3.
   - 标记符一致且V位为1的通路不存在时。 → 4.
3. 高速缓存命中
   从发生命中的通路的数据部分通过虚拟地址的位[4:3]将进行变址的数据作为指令读取。另外，为了使发生命中的通路为最新，需更新LRU位。
4. 高速缓存未命中
   从对应虚拟地址的物理地址空间，向通过LRU位选择的置换目标通路的高速缓存线读取数据。数据的读取从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据，相应数据到达高速缓存时，将读取的数据作为指令返回CPU。读取剩余的高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位。另外，为使置换的通路为最新，需更新LRU位。

8.4.2 预取操作

指令高速缓存 （IC）有效 （CCR.ICE=1）且从可执行高速缓存操作的区域将指令预取至 IC 时， IC 执行以下操作：

1. 从通过虚拟地址的位[12:5]将进行变址的各通路的高速缓存线读取标记符、 V位及LRU位。
2. 将通过MMU变换虚拟地址的物理地址的位[28:10]与从通路读取的标记符作比较，
   - 标记符一致且V位为1的通路存在时。 → 3.
   - 标记符一致且V位为1的通路不存在时。 → 4.
3. 高速缓存命中
   为使发生命中的通路为最新，需更新LRU位。
4. 高速缓存未命中
   从对应虚拟地址的物理地址空间，向置换目标通路的高速缓存线读取数据。从包含高速缓存未命中的数据的四字 （8字节）开始按顺序以环绕的方式读取数据。预取操作中， CPU不等待数据的到达，在读取高速缓存1条线的数据时， CPU可进行下一个处理。在高速缓存读完1条线的数据时，注册通过物理地址产生的标记符，将1写入V位。另外，为使置换的通路为最新，需更新LRU位。

8.4.3 IC 2 路模式

将 RAMCR 寄存器的 IC 2W 位置 1 时，为仅使用 IC 的通路 0 与通路 1 的 IC 2 路模式，可降低功耗。本模式也包含存储器分配 IC 存取，仅使用通路 0 与通路 1。

通过 P2 区域的程序进行 IC2W 位的改写。另外，改写时，已在 IC 注册有效线的情况下，改写 IC2W 位之前，请将 1 写入 CCR 寄存器的 ICI 位，并使 IC 的所有入口无效。

8.5 高速缓存操作指令

8.5.1 高速缓存与外部存储器的相关性

请通过软件保证高速缓存与外部存储器的相关性。在 SH-4A，作为操作高速缓存的指令支持以下 6 条指令。各指令的详细内容请参考 “**第 10 章 各指令的说明**”。

- 操作数高速缓存无效化指令：OCBI @Rn
  使操作数高速缓存无效 （无回写）
- 操作数高速缓存清除指令：OCBP @Rn
  使操作数高速缓存无效 （有回写）
- 操作数高速缓存回写指令：OCBWB @Rn
  操作数高速缓存的回写
- 操作数高速缓存分配指令：MOVCA.L R0,@Rn
  操作数高速缓存的确保
- 指令高速缓存无效化指令：ICBI @Rn
  使指令高速缓存无效
- 操作数存取同步指令：SYNCO
  等待数据传送完成

另外，为控制操作数高速缓存的相关性，可接受来自 SuperHyway 总线的 PURGE 及 FLUSH 处理。可通过 PURGE/FLUSH 处理提供的地址为物理地址。因此，允许 MMU 时，为避免高速缓存同义问题，请注意以下限制事项：

- 请勿使用1K字节的页面大小。

(1) PURGE 处理

允许操作数高速缓存时，检索操作数高速缓存，使已发生命中的入口无效。如果被无效的线为脏，则回写至外部存储器。发生未命中时为空操作。

(2) FLUSH 处理

允许操作数高速缓存时，检索操作数高速缓存，有已发生命中的入口，且如果为脏，则回写至外部存储器。不使已发生命中的入口无效。发生未命中或发生命中的入口不为脏时为空操作。

8.5.2 预取操作

为减少通过高速缓存未命中发生的高速缓存满的损失， SH-4A 支持预取指令。通过读取操作、写入操作获知发生高速缓存未命中时，通过预取指令预先将数据填充至高速缓存，可防止在读取操作、写入操作过程中发生高速缓存未命中。以此可提高软件功能。对已保存至高速缓存的数据执行预取指令，或者准备预取的地址在 UTLB 发生错误时或违反保护时，为空操作，不发生异常。预取指令的详细内容请参考 **“第 10 章 各指令的说明”**。

- 预取指令 （OC）：PREF @Rn
- 预取指令 （IC）：PREFI @Rn

8.6 存储器分配高速缓存的结构

为通过软件管理 IC、 OC，特权模式下，可通过 MOV 指令从 P2 区域的程序读取 / 写入 IC 的内容。不保证从其他区域的程序的存取。此时，请采用以下 1 ～ 3 的任意一种方法来进行向 P0、 U0、 P1、 P3 区域的转移。

1. 请执行通过RTE指令产生的转移。
2. 对任意地址 （亦可为不可执行高速缓存操作的区域），执行ICBI指令后，请进行向P0、 U0、 P1、 P3区域的转移。
3. 在对存储器分配IC存取之前，预先设定为IRMCR.MC=0 （初始值）时，无需特定的指令顺序。但由于此方法必须从取指令重新执行存储器分配IC存取指令的下一个指令，会降低CPU的处理功能，所以请注意。

但在今后的 SuperH 系列有可能不保证方法 3，为保证今后 SuperH 中系列的兼容性，推荐使用 1 或 2。

另外，特权模式下，可通过 MOV 指令从 P1、 P2 区域的程序读取 / 写入 OC 的内容。不保证从其他区域的程序的存取。将 IC、OC 分配至虚拟地址空间的 P4 区域。 IC 的地址阵列 / 数据阵列、OC 的地址阵列 / 数据阵列均仅为可进行数据存取，存取长度固定为长字。对此区域不可执行取指令。请将保留位设定为 0。保留位的读取值为不定。

8.6.1 IC 地址阵列

将 IC 的地址阵列分配至 P4 区域的 H'F000 0000 ～ H'F0FF FFFF。地址阵列的存取需要指定 32 位地址部分（读取 / 写入时）与 32 位数据部分。在地址部分指定存取的通路与入口，在数据部分指定写入的标记符与 V 位。

地址部分中， [31:24] 为表示 IC 地址阵列的 H'F0，通过 [14:13] 指定通路，通过 [12:5] 指定入口。地址部分 [3] 的联想位 （A 位）在向 IC 地址阵列写入时，指定是否进行联想。因为存取固定为长字长度，所以请将地址部分 [1:0] 指定为 0。

数据部分中， [31:10] 表示标记符， [0] 表示 V 位。由于 IC 地址阵列的标记符为 19 位，所以在不进行联想的写入时不使用数据部分 [31:29]。仅在进行联想写入时，数据部分 [31:29] 用于指定虚拟地址。

可对 IC 地址阵列进行以下 3 种操作：

(1) IC 地址阵列 读取

从在地址部分设定的通路与入口所对应的 IC 入口向数据部分读取标记符及 V 位。读取时，在地址部分指定的联想位无论为 1 还是为 0，均不执行联想操作。

(2) IC 地址阵列 写入 （无联想）

对于在地址部分设定的通路与入口所对应的 IC 入口，写入在数据部分指定的标记符与 V 位。请将地址部分的 A 位置 0。

(3) IC 地址阵列 写入 （有联想）

将 1 写入地址部分的 A 位时，可在地址部分所指定的入口保存的各通路的标记符与数据部分所指定的标记符之间进行匹配判断。不使用地址部分 [14:13] 的通路号。此时，如果允许 MMU，则使用 ITLB 将在数据部分 [31:10] 指定的虚拟地址转换为物理地址之后再进行匹配判断。如果地址匹配且该通路的 V 位为 1，则将在数据部分指定的 V 位写入 IC 的入口。除此以外均为空操作。本操作用于使 IC 特定入口无效。地址转换时，如果发生 ITLB 未命中或通过匹配判断确定为不匹配，则不发生异常，为空操作，并且不进行写入。

【注】 在今后的 SuperH 系列中有可能不支持本功能。推荐进行 ITLB 未命中处理或指令 TLB 未命中异常的通知，推荐使用可正确执行 IC 操作的 ICBI 指令。

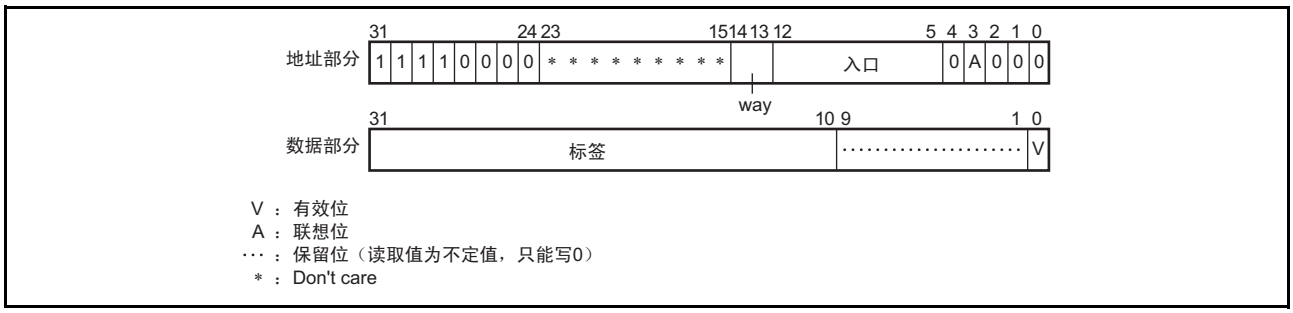


图 8.5 存储器分配 IC 地址阵列

8.6.2 IC 数据阵列

将 IC 的数据阵列分配至 P4 区域的 H'F100 0000 ～ H'F1FF FFFF。数据阵列的存取需要指定 32 位地址部分（读取 / 写入时）与 32 位数据部分。在地址部分指定存取的通路与入口，在数据部分指定写入的长字数据。

地址部分中， [31:24] 为表示 IC 数据阵列的 H'F1，通过 [14:13] 指定通路，通过 [12:5] 指定入口。地址部分 [4:2] 用于指定入口中的长字数据。因为存取固定为长字长度，所以请指定地址部分 [1:0] 为 0。

数据部分用于指定长字数据。

可对 IC 数据阵列执行以下 2 种操作：

(1) IC 数据阵列 读取

在地址部分设定的通路与入口所对应的 IC 入口中，从通过地址部分的长字指定位所指定的数据向数据部分读取长字数据。

(2) IC 数据阵列 写入

在地址部分设定的通路与入口所对应的 IC 入口中，对通过地址部分的长字指定位所指定的数据，写入在数据部分指定的长字数据。

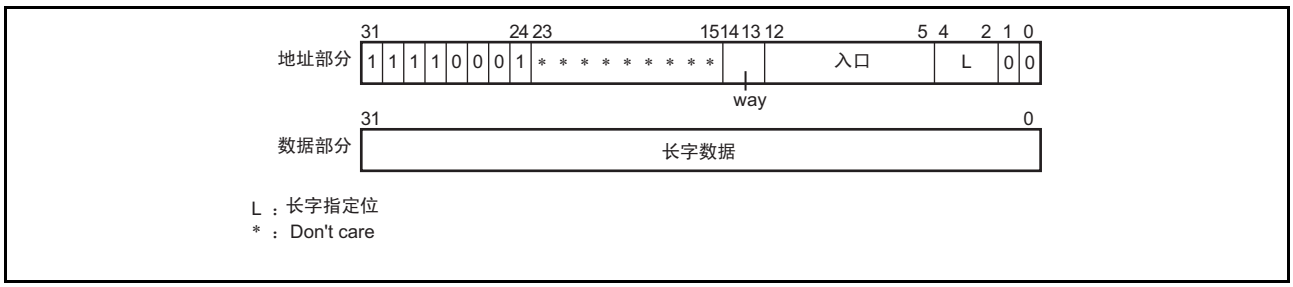


图 8.6 存储器分配 IC 数据阵列

8.6.3 OC 地址阵列

将 OC 的地址阵列分配至 P4 区域的 H'F400 0000 ～ H'F4FF FFFF。地址阵列的存取需要指定 32 位地址部分 （读取 / 写入时）与 32 位数据部分。在地址部分指定存取的通路与入口，在数据部分指定写入的标记符、U 位及 V 位。

地址部分中， [31:24] 为表示 OC 地址阵列的 H'F4，通过 [14:13] 指定通路，通过 [12:5] 指定入口。地址部分 [3] 的联想位 （A 位）在向 OC 地址阵列写入时，指定是否进行联想。因为存取固定为长字长度，所以地址部分 [1:0] 请指定为 0。

数据部分中， [31:10] 表示标记符， [1] 表示 U 位， [0] 表示 V 位。由于 OC 地址阵列的标记符为 19 位，所以不进行联想写入时，不使用数据部分 [31:29]。数据部分 [31:29] 仅在进行联想写入时用于指定虚拟地址。

可对 OC 地址阵列执行以下 3 种操作：

(1) OC 地址阵列 读取

从在地址部分设定的通路与入口所对应的 OC 入口中，向数据部分读取标记符、 U 位及 V 位。读取时，在地址部分指定的联想位无论为 1 或 0，均不进行联想操作。

(2) OC 地址阵列 写入 （无联想）

对在地址部分设定的通路与入口所对应的 OC 入口中，写入在数据部分指定的标记符、 U 位及 V 位。请将地址部分 A 位置 0。

对 U 位为 1、 V 位为 1 的高速缓存线进行写入时，进行该高速缓存线的回写后，写入在数据部分指定的标记符、 U 位及 V 位。

(3) OC 地址阵列 写入 （有联想）

将 1 写入地址部分的 A 位时，可在地址部分所指定的入口保存的各通路的标记符与在数据部分指定的标记符之间进行匹配判断。不使用位 [14:13] 的通路号。此时，允许如果 MMU，则使用 UTLB 将在数据部分 [31:10] 指定的虚拟地址转换为物理地址之后，进行匹配判断。如果地址匹配且该通路的 V 位为 1，则将在数据部分指定的 U 位与 V 位写入 OC 的入口。除此以外为空操作。本操作用于使 OC 特定入口无效。此时如果 OC 的入口的 U 位为 1， V 位为 0 或将 0 写入 U 位，则发生回写。地址转换时，在发生 UTLB 未命中或通过匹配判断确定为不匹配时，不发生异常，为空操作，并且不进行写入。

【注】 在今后的 SuperH 系列中有可能不支持本功能。推荐进行数据 TLB 未命中异常的通知，使用可正确执行 OC 操作的 OCBI/OCBP/OCBWB 指令。

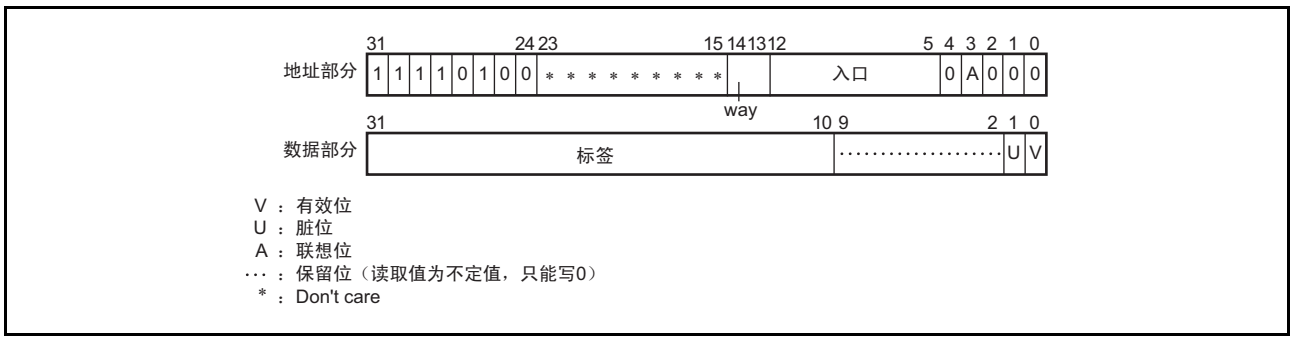


图 8.7 存储器分配 OC 地址阵列

8.6.4 OC 数据阵列

将 OC 的数据阵列分配至 P4 区域的 H'F500 0000 ～ H'F5FF FFFF。数据阵列的存取需要指定 32 位地址部分 （读取 / 写入时）与 32 位数据部分。在地址部分指定存取的通路与入口，在数据部分指定写入的长字数据。

地址部分中， [31:24] 为表示 OC 数据阵列的 H'F5，通过 [14:13] 指定通路，通过 [12:5] 指定入口。地址部分 [4:2] 用于指定入口中的长字数据。因为存取固定为长字长度，所以地址部分 [1:0] 请指定为 0。

数据部分用于指定长字数据。

可对 OC 数据阵列执行以下 2 种操作：

(1) OC 数据阵列 读取

在地址部分设定的通路与入口对应的 OC 入口中，从通过地址部分的长字指定位所指定的数据向数据部分读取长字数据。

(2) OC 数据阵列 写入

在地址部分设定的通路与入口对应的 OC 入口中，对通过地址部分的长字指定位所指定的数据，写入在数据部分指定的长字数据。通过该写入，地址阵列方的 U 位不为 1。

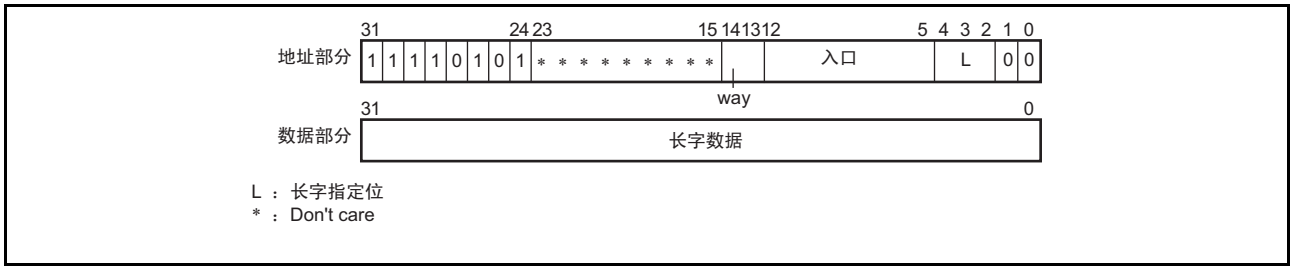


图 8.8 存储器分配 OC 数据阵列

8.7 存储队列

为向外部存储器进行快速写入， SH-4A 支持 32 字节 ×2 的存储队列 （SQ）。

8.7.1 SQ 的结构

SQ 如图 8.9 所示，由 32 字节的 SQ0 与 32 字节的 SQ1 构成。可分别独立设定 SQ0、 SQ1。

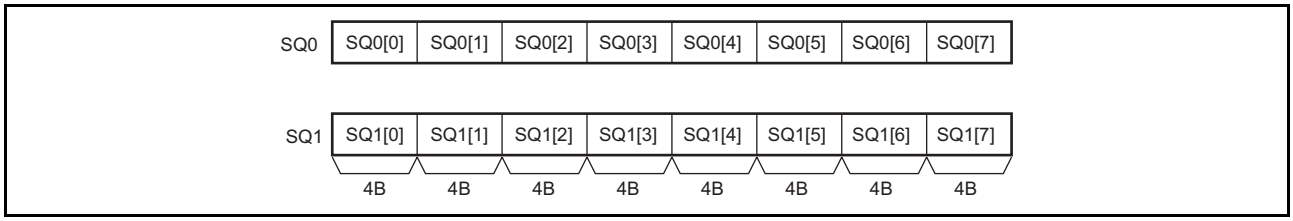


图 8.9 存储队列的构成

8.7.2 向 SQ 的写入

可通过对 P4 区域的 H'E000 0000 ～ H'E3FF FFFC 的存储指令来进行向 SQ 的写入。存取长度可为长字或四字。该地址具有以下含义：

[31:26]	： 111000	： 指定存储队列
[25:6]	： Don't care	： 用于向外部存储器的传送 / 存取权
[5]	： 0/1	： 0: 指定 SQ0 1: 指定 SQ1
[4:2]	： LW 指定	： 指定 SQ0、 SQ1 中的长字位置
[1:0]	： 00	： 固定为 0

8.7.3 向外部存储器的传送

可通过预取指令 （PREF）来进行从 SQ 向外部存储器的传送。通过对 P4 区域的 H'E000 0000 ～ H'E3FF FFFC 发行 PREF 指令，开始进行从 SQ 向外部存储器的传送。传送固定为 32 字节，起始地址必须为 32 字节边界。向外部存储器传送一方 SQ 的过程中，可无损失周期地进行向另一方 SQ 的写入，但在向外部存储器传送的过程中，向 SQ 的写入可等待至向外部存储器的传送完成。

SQ 的传送目标的物理地址 [28:0] 通过允许 / 禁止 MMU 进行如下指定：

(1) 允许 MMU （MMUCR.AT=1）时

在 UTLB 的 VPN 设定 SQ 区域 （H'E000 0000 ～ H'E3FF FFFF），在 PPN 设定传送目标的物理地址。ASID、V、SZ、SH、PR、D 位与普通的地址转换具有相同的含义，但 C、WT 位不具有与该页面相关的含义。

发行向 SQ 区域的预取指令时，进行地址转换，根据 SZ 位的指定生成物理地址 [28:10]。物理地址的 [9:5]，与禁止 MMU 相同，均从地址转换前的地址生成。物理地址的 [4:0] 固定为 0。可对该地址进行从 SQ 向外部存储器的传送。

(2) 禁止 MMU （MMUCR.AT=0）时

在发行 PREF 指令的地址指定 SQ 区域 （H'E000 0000 ～ H'E3FF FFFF）。该地址 [31:0] 具有以下含义：

[31:26]	：111000	：指定存储队列
[25:6]	：地址	：传送目标物理地址 [25:6]
[5]	：0/1	：0: 指定 SQ0　1: 指定 SQ1　且　传送目标物理地址 [5]
[4:2]	：Don't care	：预取时无含义
[1:0]	：00	：固定为 0

从 QACR0、 QACR1 生成从上述地址无法生成的物理地址 [28:26]。

QACR0[4:2]	：SQ0 的物理地址 [28:26]
QACR1[4:2]	：SQ1 的物理地址 [28:26]

由于突发传送的起始为 32 字节边界，所以物理地址的 [4:0] 通常固定为 0。

8.7.4 SQ 存取的异常判断

可通过允许 / 禁止 MMU 来如下进行向 SQ 写入及向外部存储器传送 （PREF 指令）的异常判断。在向 SQ 写入时发生异常的情况下， SQ 的内容保证为原值。在从 SQ 向外部存储器传送时发生异常的情况下，禁止向外部存储器传送。

(1) 允许 MMU （MMUCR.AT=1）时

依据在 UTLB 注册的地址转换信息与 SQMD 位。将向 SQ 的写入作为写入类型、从 SQ 向外部存储器的传送 （PREF 指令）作为读取类型来进行异常判断，发生 TLB 未命中异常及保护违反异常。但，仅在特权模式下允许通过 SQMD 位对 SQ 存取时，用户模式下即使地址转换成功也为地址错误。

(2) 禁止 MMU （MMUCR.AT=0）时

根据 SQMD 位。

0：可特权 / 用户存取

1：可特权存取

SQMD 位为 1 时，用户模式下如果存取 SQ 区域，则发生地址错误。

8.7.5 从 SQ 的读取

特权模式下， SH-4A 可通过对 P4 区域的 H'FF00 1000 ～ H'FF00 103C 的加载指令进行从 SQ 的读取。存取长度仅为长字存取。

[31:26]	：H'FF00 1000	：指定存储队列
[5]	：0/1	：0：指定 SQ0、 1：指定 SQ1
[4:2]	：LW 指定	：指定 SQ0、 SQ1 中的长字位置
[1:0]	：00	：固定为 0

8.8 32 位地址扩展模式使用注意事项

32 位地址扩展模式下，对本章已叙述的事项进行如下扩展：

1. IC及OC的标记符从[28:10]的19位开始，扩展为[31:10]的22位。
2. 配置操作IC指令 （存储器分配IC存取及CCR.ICI写入）的区域为P1或P2区域，请将PMB的该入口的可执行高速缓存位 （C位）设定为0。
3. QACR0寄存器的AREA0位及QACR1寄存器的AREA1位分别从[4:2]的3位扩展为[7:2]的6位。

第 9 章 L 存储器

SH-4A 内置 L 存储器模块，可存储指令或数据。

【注】 关于 L 存储器的容量，请参考产品的硬件手册。

9.1 特点

- 容量：
  L 存储器总计可从 16K 字节、 32K 字节、 64K 字节及 128K 字节选择。
- 页面：
  L 存储器分为 2 个页面 （页面 0 及页面 1）。
- 存储器映射：
  本存储器将虚拟地址空间、物理地址空间都分配在表 9.1 所示的地址。

表 9.1 存储器地址

页面	存储器大小 （总计 2 个页面）			
	16K 字节	32K 字节	64K 字节	128K 字节
L 存储器页面 0	H'E500E000 ～ H'E500FFFF	H'E500C000 ～ H'E500FFFF	H'E5008000 ～ H'E500FFFF	H'E5000000 ～ H'E500FFFF
L 存储器页面 1	H'E5010000 ～ H'E5011FFF	H'E5010000 ～ H'E5013FFF	H'E5010000 ～ H'E5017FFF	H'E5010000 ～ H'E501FFFF

- 端口：
  各页面具有 3 个独立的读取/写入端口，与各个总线连接。取指令使用指令总线，操作数存取使用操作数总线，从 SuperHyway 总线主控器模块的存取使用 SuperHyway 总线。
- 优先顺序：
  对于相同的页面，从不同的总线同时产生存取请求时，根据优先顺序来处理存取。优先顺序按由高到低依次为 SuperHyway 总线、操作数总线、指令总线。

9.2 寄存器说明

与 L 存储器相关的寄存器如下所示。

表 9.2 寄存器结构

名称	简称	R/W	P4 区域地址 *	区域 7 地址	大小
内部存储器控制寄存器	RAMCR	R/W	H'FF00 0074	H'1F00 0074	32
L 存储器传送源地址寄存器 0	LSA0	R/W	H'FF00 0050	H'1F00 0050	32
L 存储器传送源地址寄存器 1	LSA1	R/W	H'FF00 0054	H'1F00 0054	32
L 存储器传送目标地址寄存器 0	LDA0	R/W	H'FF00 0058	H'1F00 0058	32
L 存储器传送目标地址寄存器 1	LDA1	R/W	H'FF00 005C	H'1F00 005C	32

【注】 * P4 区域地址为使用虚拟地址空间的 P4 区域时的地址。区域 7 地址为使用 TLB、从物理地址空间的区域 7 存取的地址。

表 9.3 各处理状态下的寄存器状态

名称	简称	上电复位	手动复位	睡眠	待机
内部存储器控制寄存器	RAMCR	H'0000 0000	H'0000 0000	保持	保持
L 存储器传送源地址寄存器 0	LSA0	不定	不定	保持	保持
L 存储器传送源地址寄存器 1	LSA1	不定	不定	保持	保持
L 存储器传送目标地址寄存器 0	LDA0	不定	不定	保持	保持
L 存储器传送目标地址寄存器 1	LDA1	不定	不定	保持	保持

9.2.1 内部存储器控制寄存器 （RAMCR）

RAMCR 控制 L 存储器的保护功能。

位名：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

位名：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	RMD	RP	IC2W	OC2W	—	—	—	—	—	—
初始值：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W：	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R	R	R	R	R	R

位	位名称	初始值	R/W	说明
31 ～ 10	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
9	RMD	0	R/W	内部存储器存取模式位 指定从虚拟地址空间对 L 存储器存取的存取权。 0：可进行特权存取 （用户存取时发生地址错误异常） 1：可进行用户 / 特权存取
8	RP	0	R/W	内部存储器保护有效位 对于从虚拟地址空间对 L 存储器的存取，选择是否使用 ITLB、 UTLB 的保护功能。 0：不使用保护功能 1：使用保护功能 详细内容请参考 “9.4 L 存储器的保护功能”。
7	IC2W	0	R/W	IC 2 路模式位 详细内容请参考 “8.4.3 IC 2 路模式”。
6	OC2W	0	R/W	OC 2 路模式位 详细内容请参考 “8.3.6 OC 2 路模式”。
5 ～ 0	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。

9.2.2 L 存储器传送源地址寄存器 0 （LSA0）

MMUCR.AT=0 或 RAMCR.RP=0 时，在向 L 存储器页面 0 进行块传送的情况下， LSA0 指定传送源的物理地址。

位名：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	L0SADR												
初始值：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位名：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	L0SADR						—	—	—	—	L0SSZ					
初始值：	—	—	—	—	—	—	0	0	0	0	—	—	—	—	—	—
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 29	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
28 ～ 10	L0SADR	不定	R/W	L 存储器页面 0 块传送源地址 MMUCR.AT=0 或 RAMCR.RP=0 时，指定作为对 L 存储器页面 0 进行块传送的传送源的物理地址。
9 ～ 6	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
5 ～ 0	L0SSZ	不定	R/W	L 存储器页面 0 块传送源地址选择位 MMUCR.AT=0 或 RAMCR.RP=0 时，关于作为对 L 存储器页面 0 进行块传送的传送源的物理地址中的 bit15 ～ 10，选择使用操作数地址或使用 L0SADR 的值。 L0SSZ[5:0] 对应传送源物理地址的 [15:10]。 0：传送源物理地址使用操作数地址。 1：传送源物理地址使用 L0SADR 的值。 • 可设定的值 111111 以 1K 字节为单位设定传送源的物理地址时 111110 以 2K 字节为单位设定传送源的物理地址时 111100 以 4K 字节为单位设定传送源的物理地址时 111000 以 8K 字节为单位设定传送源的物理地址时 110000 以 16K 字节为单位设定传送源的物理地址时 100000 以 32K 字节为单位设定传送源的物理地址时 000000 以 64K 字节为单位设定传送源的物理地址时 上述以外，禁止设定。

9.2.3 L 存储器传送源地址寄存器 1（LSA1）

MMUCR.AT=0 或 RAMCR.RP=0 时，在向 L 存储器页面 1 进行块传送的情况下， LSA1 指定传送源的物理地址。

位名：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	L1DADR												
初始值：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位名：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	L1DADR						—	—	—	—	L1DSZ					
初始值：	—	—	—	—	—	—	0	0	0	0	—	—	—	—	—	—
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 29	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
28 ～ 10	L1SADR	不定	R/W	L 存储器页面 1 块传送源地址 MMUCR.AT=0 或 RAMCR.RP=0 时，指定作为对 L 存储器页面 1 进行块传送的传送源的物理地址。
9 ～ 6	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
5 ～ 0	L1SSZ	不定	R/W	L 存储器页面 1 块传送源地址选择位 MMUCR.AT=0 或 RAMCR.RP=0 时，关于作为对 L 存储器页面 1 进行块传送的传送源的物理地址中的 bit15 ～ 10，选择使用操作数地址或使用 L1SADR 的值。 L1SSZ[5:0] 对应传送源物理地址的 [15:10]。 0：传送源物理地址使用操作数地址。 1：传送源物理地址使用 L1SADR 的值。 • 可设定的值 111111 以 1K 字节为单位设定传送源的物理地址时 111110 以 2K 字节为单位设定传送源的物理地址时 111100 以 4K 字节为单位设定传送源的物理地址时 111000 以 8K 字节为单位设定传送源的物理地址时 110000 以 16K 字节为单位设定传送源的物理地址时 100000 以 32K 字节为单位设定传送源的物理地址时 000000 以 64K 字节为单位设定传送源的物理地址时 上述以外，禁止设定。

9.2.4 L 存储器传送目标地址寄存器 0 （LDA0）

MMUCR.AT=0 或 RAMCR.RP=0 时，在向 L 存储器页面 0 进行块传送的情况下，LDA0 指定传送目标的物理地址。

位名：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	L0DADR												
初始值：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位名：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	L0DADR						—	—	—	—	L0DSZ					
初始值：	—	—	—	—	—	—	0	0	0	0	—	—	—	—	—	—
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 29	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
28 ～ 10	L0DADR	不定	R/W	L 存储器页面 0 块传送目标地址 MMUCR.AT=0 或 RAMCR.RP=0 时，指定作为对 L 存储器页面 0 进行块传送的传送目标的物理地址。
9 ～ 6	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
5 ～ 0	L0DSZ	不定	R/W	L 存储器页面 0 块传送目标地址选择位 MMUCR.AT=0 或 RAMCR.RP=0 时，关于作为对 L 存储器页面 0 进行块传送的传送目标的物理地址中的 bit15 ～ 10，选择使用操作数地址或使用 L0DADR 的值。 L0DSZ[5:0] 对应传送目标物理地址的 [15:10]。 0：传送目标物理地址使用操作数地址。 1：传送目标物理地址使用 L0DADR 的值。 • 可设定的值 111111 以 1K 字节为单位设定传送目标的物理地址时 111110 以 2K 字节为单位设定传送目标的物理地址时 111100 以 4K 字节为单位设定传送目标的物理地址时 111000 以 8K 字节为单位设定传送目标的物理地址时 110000 以 16K 字节为单位设定传送目标的物理地址时 100000 以 32K 字节为单位设定传送目标的物理地址时 000000 以 64K 字节为单位设定传送目标的物理地址时 上述以外，禁止设定。

9.2.5 L 存储器传送目标地址寄存器 1（LDA1）

MMUCR.AT=0 或 RAMCR.RP=0 时，在向 L 存储器页面 1 进行块传送的情况下，LDA1 指定传送目标的物理地址。

位名：	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	L1DADR												
初始值：	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W：	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

位名：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	L1DADR						—	—	—	—	L1DSZ					
初始值：	—	—	—	—	—	—	0	0	0	0	—	—	—	—	—	—
R/W：	R/W	R/W	R/W	R/W	R/W	R/W	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W

位	位名称	初始值	R/W	说明
31 ～ 29	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
28 ～ 10	L1DADR	不定	R/W	L 存储器页面 1 块传送目标地址 MMUCR.AT=0 或 RAMCR.RP=0 时，指定作为对 L 存储器页面 1 进行块传送的传送目标的物理地址。
9 ～ 6	—	均为 0	R	保留位 关于本位的读取 / 写入请参考 “产品使用时的注意事项”。
5 ～ 0	L1DSZ	不定	R/W	L 存储器页面 1 块传送目标地址选择位 MMUCR.AT=0 或 RAMCR.RP=0 时，关于作为对 L 存储器页面 1 进行块传送的传送目标的物理地址中的 bit15 ～ 10，选择使用操作数地址或使用 L1DADR 的值。 L1DSZ[5:0] 对应传送目标物理地址的 [15:10]。 0：传送目标物理地址使用操作数地址。 1：传送目标物理地址使用 L1DADR 的值。 • 可设定的值 111111 以 1K 字节为单位设定传送目标的物理地址时 111110 以 2K 字节为单位设定传送目标的物理地址时 111100 以 4K 字节为单位设定传送目标的物理地址时 111000 以 8K 字节为单位设定传送目标的物理地址时 110000 以 16K 字节为单位设定传送目标的物理地址时 100000 以 32K 字节为单位设定传送目标的物理地址时 000000 以 64K 字节为单位设定传送目标的物理地址时 上述以外，禁止设定。

9.3 运行说明

9.3.1 从 CPU 及 FPU 存取

通过虚拟地址，从指令总线或操作数总线进行从 CPU 及 FPU 的存取。只要不发生页面竞争，就为 1 个周期存取。

9.3.2 从 SuperHyway 总线主控器模块存取

从 DMAC 等的 SuperHyway 总线主控器模块对本存储器的存取，是从作为物理地址总线的 SuperHyway 总线的存取，请使用与虚拟地址相同的地址。

9.3.3 块传送

在 L 存储器与外部存储器之间，可以不通过高速缓存，而通过块传送进行高速数据传送。

可通过预取指令 （PREF）进行从外部存储器向 L 存储器的传送。通过对虚拟地址空间的 L 存储器区域的地址发行 PREF 指令，开始从外部存储器向 L 存储器进行块传送。

可通过回写指令 （OCBWB）进行从 L 存储器向外部存储器的传送。通过对虚拟地址空间的 L 存储器区域的地址发行 OCBWB 指令，开始从 L 存储器向外部存储器进行块传送。

由于任何一个传送的传送长度均固定为 32 字节，开始地址也必须为 32 字节边界，所以忽视通过寄存器 Rn 所指示地址的低 5 位，通常均作为 0 来处理。另外，任何时候进行块传送时，都有可能对其他页面及高速缓存进行存取，但在存取传送中的页面时，到传送结束之前， CPU 会停止。

通过允许 / 禁止 MMU，如下指定 L 存储器与进行传送的外部存储器的物理地址 [28:0]。

(1) 允许 MMU （MMUCR.AT=1）且 RAMCR.RP=1 时

给 UTLB 的 VPN 字段设定 L 存储器区域的地址，给 PPN 字段设定传送源 （PREF 指令时）或传送目标（OCBWB 指令时）的物理地址。ASID、V、SZ、SH、PR、D 位与通常地址转换具有同样的意思，但是，C、WT 位不具有与该页面相关的意思。

向 L 存储器区域发行 PREF 指令时，进行地址转换，并根据 SZ 位的指定生成物理地址 [28:10]。物理地址的 [9:5] 从地址转换前的虚拟地址生成。物理地址的 [4:0] 固定为 0。从通过此物理地址指定的外部存储器向 L 存储器进行块传送。

向 L 存储器区域发行 OCBWB 指令时，进行地址转换，并根据 SZ 位的指定生成物理地址 [28:10]。物理地址的 [9:5] 从地址转换前的虚拟地址生成。物理地址的 [4:0] 固定为 0。从 L 存储器向通过该物理地址指定的外部存储器进行块传送。

PREF 指令、 OCBWB 指令作为读取类型，判断 MMU 异常，按照需要发生 TLB 未命中异常、保护违反异常。发生异常时，禁止块传送。

(2) MMU 禁止 （MMUCR.AT0）或 RAMCR.RP=0 时

给 LSA0 寄存器的 L0SADR 位设定作为向 L 存储器页面 0 进行块传送的传送源的物理地址，通过软件设定， L0SSZ 位使用通过 PREF 指令作为传送源的物理地址的 bit15 ～ 10 指定的虚拟地址还是使用 L0SADR 的值。即：能够以 1K 字节～ 64K 字节为单位来设定传送源区域。

给 LDA0 寄存器的 L0DADR 位设定作为从 L 存储器页面 0 进行块传送的传送目标的物理地址，通过软件设定， L0DSZ 位使用通过 OCBWB 指令作为传送目标的物理地址的 bit15 ～ 10 指定的虚拟地址还是使用 L0DADR 的值。即：能够以 1K 字节～ 64K 字节为单位设定传送目标区域。

L 存储器页面 1 的块传送的设定与页面 0 相同，也是对 LSA1 及 LDA1 进行设定。

向 L 存储器区域发行 PREF 指令时，根据 LSA0 寄存器或 LSA1 寄存器的指定生成物理地址 [28:10]。物理地址的 [9:5] 从虚拟地址生成。物理地址的 [4:0] 固定为 0。从通过该物理地址指定的外部存储器向 L 存储器进行块传送。

向 L 存储器区域发行 OCBWB 指令时，根据 LDA0 寄存器或 LDA1 寄存器的指定生成物理地址 [28:10]。物理地址的 [9:5] 从虚拟地址生成。物理地址的 [4:0] 固定为 0。从 L 存储器向通过该物理地址指定的外部存储器进行块传送。

9.4 L 存储器的保护功能

SH-4A 对于 L 存储器，使用内部存储器控制寄存器 RAMCR 的内部存储器存取模式位 （RMD）与内部存储器保护有效位 （RP），实现以下保护功能。

- 对从CPU及FPU存取的保护功能
  RAMCR.RMD=0时，将用户模式下的存取判断为地址错误异常。
  另外， MMUCR.AT=1且RAMCR.RP=1时，包含地址错误异常的判断，作为P4区域一部分的L存储器区域也与P0/P3/U0区域相同，进行MMU异常的判断。
  上述汇总在表9.4。

表 9.4 由于对 L 存储器存取的保护功能而产生的异常

MMUCR.AT	RAMCR.RP	SR.MD	RAMCR.RMD	必须产生的异常	可能产生的异常
0	*	0	0	地址错误异常	—
			1	—	—
		1	*	—	—
1	0	0	0	地址错误异常	—
			1	—	—
		1	*	—	—
	1	0	0	地址错误异常	—
			1	—	MMU 异常
		1	*	—	MMU 异常

【符号说明】*：Don't care

9.5 使用注意事项

9.5.1 页面竞争

对于相同的页面，从不同的总线同时产生存取请求时，为页面竞争。虽然各存取正确完成，但这种竞争会导致存储器存取的功能降低。因此，为了尽量不引起竞争，推荐使用软件对策。例如，如果各总线存取不同的页面，就不会发生竞争。

9.5.2 L 存储器的相关性

将指令分配至 L 存储器时，给 L 存储器写入指令后，请执行以下顺序后转移至改写后的指令。

- SYNCO
- ICBI @Rn

此时， ICBI 指令的对象在非地址错误异常的范围内，可为任意地址（L 存储器的地址亦可），也可为高速缓存命中 / 未命中。

9.5.3 睡眠模式

睡眠模式中，不可从 DMAC 等的 SuperHyway 总线主控器模块对本存储器存取

9.6 32 位地址扩展模式使用注意事项

32 位地址扩展模式下，LSA0 寄存器的 L0SADR 位、 LSA1 寄存器的 L1SADR 位、 LDA0 寄存器的 L0DADR 位、 LDA1 寄存器的 L1DADR 位分别从 [28:10] 的 19 位扩展为 [31:10] 的 22 位。

第 10 章 各指令的说明

本章说明所支持指令的操作。使用以下格式，以指令单位按照字母顺序进行说明。

指令名称　　指令功能（英文）　　指令分类

指令功能

（表示延迟转移指令或特权指令）

格式	操作概略	操作码	执行状态	T位
以汇编程序的输入形式表示。imm、disp变为数值、公式或符号。	表示操作概略。	按照MSB<=>LSB的顺序表示。	不等待时的值。	表示指令执行后T位的状态。

（1）说明

操作的说明。

（2）注意

说明使用指令时必须注意的事项。

（3）操作内容

以C语言表示操作内容。

（4）使用例

以汇编程序助记符为例表示指令执行前后的状态。
斜体字（例：*.align*）表示为汇编程序控制指令。汇编程序控制指令所表示的意思如下。详情请参考「C/C++编译器、汇编程序、优化连接编辑程序」用户手册。

.org	设定位置计数器
.data.W	确保字整数数据
.data.1	确保长字整数数据
.sdata	确保字符串数据
.align 2	调整2字节边界
.align 4	调整4字节边界
.align 32	调整32字节边界
.arepeat 16	16次重复展开
.aerpeat 32	32次重复展开
.aendr	指定次数重复展开完成

【注】：SH系列交叉汇编程序Ver1.0中，不支持带条件的汇编功能。

（5）有可能发生的异常 （无使用说明例时为（4项））

列举执行指令时有可能发生的异常。但是因为执行任何指令的时候都有可能发生指令TLB多重命中异常、指令TLB未命中异常、指令TLB保护违反异常、指令地址错误，所以此处省略。另外详细说明了有关上溢/下溢异常的产生条件。

10.1 CPU 指令

【注】 “10.2 CPU 指令 （FPU 相关）”中记载了 CPU 指令中支持 FPU 的 CPU 指令及与 SH-4A、 SH4AL-DSP 中一部分功能不同的指令，本节记载其他指令。

在对使用了 CPU 指令的 C 语言描述的操作内容进行说明时，使用以下资源与函数。

```
char       8-bit integer
short     16-bit integer
int       32-bit integer
long      64-bit integer
float     single precision floating point number(32 bits)
double    double precision floating point number(64 bits)
```

这些为数据类型。

```
unsigned char  Read_Byte(unsigned long Addr);
unsigned short Read_Word(unsigned long Addr);
unsigned long  Read_Long(unsigned long Addr);
```

返回地址 Addr 各自长度的内容。从 2n 地址以外的地址所读取的字，或从 4n 地址以外的地址所读取的长字被检测为地址错误。

```
unsigned char Write_Byte(unsigned long Addr, unsigned long Data);
unsigned short Write_Word(unsigned long Addr, unsigned long Data);
unsigned long Write_Long(unsigned long Addr, unsigned long Data);
```

根据各自的长度将数据 Data 写入地址 Addr。向 2n 地址以外的地址写入字或向 4n 地址以外的地址写入长字时，都检测为地址错误。

```
Delay_Slot(unsigned long Addr)
```

转移至执行地址 （Addr）的槽指令。

```
unsigned long R[16];
unsigned long SR,GBR,VBR;
unsigned long MACH,MACL,PR;
unsigned long PC;
```

各个寄存器

```
struct SR0 {
    unsigned long dummy0:22;
    unsigned long     M0:1;
    unsigned long     Q0:1;
    unsigned long     I0:4;
    unsigned long dummy1:2;
    unsigned long     S0:1;
    unsigned long     T0:1;
};
```

SR 结构定义

```
#define M ((*(struct SR0 *)(&SR)).M0)
#define Q ((*(struct SR0 *)(&SR)).Q0)
#define S ((*(struct SR0 *)(&SR)).S0)
#define T ((*(struct SR0 *)(&SR)).T0)
```

SR 中位的定义

```
Error( char *er );
```

错误显示函数

10.1.1 ADD ADD binary 算术运算指令

二进制加法运算

格式	操作概略	操作码	执行状态	T 位
ADD Rm,Rn	Rn+Rm→Rn	0011nnnnmmmm1100	1	—
ADD #imm,Rn	Rn+imm→Rn	0111nnnniiiiiiii	1	—

(1) 说明

将通用寄存器 Rn 的内容与 Rm 相加，并将结果保存在 Rn。
通用寄存器 Rn 与 8 位立即数也可相加。
因为 8 位立即数进行符号扩展后为 32 位，所以可与减法运算复用。

(2) 注意事项

无

(3) 操作内容

```
ADD(long m, long n) /* ADD Rm,Rn */
{
    R[n] += R[m];
    PC += 2;
}

ADDI(long i, long n) /* ADD #imm,Rn */
{
    if((I & 0x80) == 0)
          R[n] += (0x000000FF & (long)i);
    else R[n] += (0xFFFFFF00 | (long)i);
    PC += 2;
}
```

(4) 使用示例

```
ADD   R0,R1             ;执行前 R0=H'7FFFFFFF,R1=H'00000001
                        ;执行后 R1=H'80000000
ADD   #H'01,R2          ;执行前 R2=H'00000000
                        ;执行后 R2=H'00000001
ADD   #H'FE,R3          ;执行前 R3=H'00000001
                        ;执行后 R3=H'FFFFFFFF
```

10.1.2 ADDC ADD with Carry 算术运算指令

带进位的二进制加法运算

格式	操作概略	操作码	执行状态	T 位
ADDC Rm,Rn	Rn+Rm+T→Rn, 进位 →T	0011nnnnmmmm1110	1	进位

(1) 说明

将通用寄存器 Rn 的内容与 Rm 及 T 位相加，并将结果保存在 Rn。根据运算结果在 T 位反映进位。进行超过 32 位的加法运算时，使用该指令。

(2) 注意事项

无

(3) 操作内容

```
ADDC(long m, long n)    /* ADDC Rm,Rn */
{
        unsigned long tmp0,tmp1;

        tmp1 = R[n] + R[m];
        tmp0 = R[n];
        R[n] = tmp1 + T;
        if(tmp0>tmp1) T = 1;
        else T = 0;
        if(tmp1>R[n]) T = 1;
        PC += 2;
}
```

(4) 使用示例

```
CLRT                    ;R0:R1(64 位 )+R2:R3(64 位 )=R0:R1(64 位 )
ADDC  R3,R1             ;执行前 T=0,R1=H'00000001,R3=H'FFFFFFFF
                        ;执行后 T=1,R1=H'00000000
ADDC  R2,R0             ;执行前 T=1,R0=H'00000000,R2=H'00000000
                        ;执行后 T=0,R0=H'00000001
```

10.1.3 ADDV ADD with (Vflag) overflow check 算术运算指令

带上溢的二进制加法运算

格式	操作概略	操作码	执行状态	T 位
ADDV Rm,Rn	Rn+Rm→Rn, 上溢 →T	0011nnnnmmmm1111	1	上溢

(1) 说明

将通用寄存器 Rn 的内容与 Rm 相加，并将结果保存在 Rn。发生上溢时， T 位置位。

(2) 注意事项

无

(3) 操作内容

```
ADDV(long m, long n)    /* ADDV Rm,Rn */
{
        long dest,src,ans;

        if((long)R[n]>=0) dest = 0;
        else dest = 1;
        if((long)R[m]>=0) src = 0;
        else src = 1;
           src += dest;
        R[n] += R[m];
        if((long)R[n]>=0) ans = 0;
        else ans = 1;
        ans += dest;
        if(src==0 || src==2) {
          if(ans==1) T = 1;
          else T = 0;
        }
        else T = 0;
        PC += 2;
}
```

(4) 使用示例

```
ADDV  R0,R1          ;执行前 R0=H'00000001,R1=H'7FFFFFFE,T=0
                     ;执行后 R1=H'7FFFFFFF,T=0
ADDV  R0,R1          ;执行前 R0=H'00000002,R1=H'7FFFFFFE,T=0
                     ;执行后 R1=H'80000000,T=1
```

10.1.4 AND AND logical 逻辑运算指令

逻辑 “与” 运算

格式		操作概略	操作码	执行状态	T 位
AND	Rm,Rn	Rn&Rm→Rn	0010nnnnmmmm1001	1	—
AND	#imm,R0	R0&imm→R0	11001001iiiiiiii	1	—
AND.B	#imm,@(R0,GBR)	(R0+GBR)&imm→(R0+GBR)	11001101iiiiiiii	3	—

(1) 说明

获取通用寄存器 Rn 的内容与 Rm 的逻辑 “与”，并将结果保存在 Rn。

可实现通用寄存器 R0 与零扩展后的 8 位立即数的逻辑 “与” 运算，或在带变址的 GBR 间接寻址方式下 8 位存储器与 8 位立即数的逻辑 “与” 运算。

(2) 注意事项

通常以 AND #imm,R0 将运算结果、 R0 的高 24 位清除。

(3) 操作内容

```
AND(long m, long n)    /* AND Rm,Rn */
{
      R[n] &= R[m];
      PC += 2;
}

ANDI(long i)    /* AND #imm,R0 */
{
      R[0] &= (0x000000FF & (long)i);
      PC += 2;
}

ANDM(long i)    /* AND.B #imm,@(R0,GBR) */
{
      long temp;

      temp = (long)Read_Byte(GBR+R[0]);
      temp &= (0x000000FF & (long)i);
      Write_Byte(GBR+R[0],temp);
      PC += 2;
}
```

(4) 使用示例

```
AND    R0,R1             ;执行前 R0=H'AAAAAAAA,R1=H'55555555
                         ;执行后 R1=H'00000000
AND    #H'0F,R0          ;执行前 R0=H'FFFFFFFF
                         ;执行后 R0=H'0000000F
AND.B #H'80,@(R0,GBR)    ;执行前 @(R0,GBR)=H'A5
                         ;执行后 @(R0,GBR)=H'80
```

(5) 有可能发生的异常

仅执行 AND.B 指令时，可能会发生以下异常。

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

异常处理中，将通过该指令进行的数据存取作为字节加载、字节存储来处理。

10.1.5 BF Branch if False 转移指令

条件转移

格式	操作概略	操作码	执行状态	T 位
BF label	T=0 时 PC+4+disp ×2→PC, T=1 时 nop	10001011dddddddd	1	—

(1) 说明

该指令为参考 T 位的带条件的转移指令。 T=1 时，不转移。相反， T=0 时，转移。转移目标为地址（PC+4+ 位移量 × 2）。 PC 源值为 BF 指令地址。因为符号扩展后要给 8 位位移量乘以 2，所以与转移目标的相对距离在 -256 字节～ +254 字节范围内。

(2) 注意事项

不能到达转移目标时，需要结合 BRA、 JMP 指令来进行相应处理。

(3) 操作内容

```
BF(int d)    /* BF disp */
{
    int disp;

    if((d&0x80)==0)
          disp = (0x000000FF & d);
    else disp = (0xFFFFFF00 | d);
    if(T==0)
          PC = PC + 4 + (disp<<1);
    else PC += 2;
}
```

(4) 使用示例

```
     CLRT                          ; 通常 T=0
    BT    TRGET_T                  ; 因为 T=0，所以不转移。
    BF    TRGET_F                  ; 因为 T=0，所以向 TRGET_F 转移。
    NOP                            ;
    NOP                            ;
TRGET_F:                           ;←BF 指令的转移目标
```

(5) 有可能发生的异常

- 槽非法指令异常

10.1.6 BF/S Branch if False with delay Slot 转移指令

带延迟的条件转移 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
BF/S label	T=0 时 PC+4+disp× 2→PC, T=1 时 nop	10001111dddddddd	1	—

(1) 说明

该指令为参考 T 位的带延迟的条件转移指令。 T=1 时，执行下一条指令，不转移。 T=0 时，执行下一条指令后进行转移。

转移目标为地址 （PC+4+ 位移量 ×2）。 PC 源值为 BF/S 指令地址。因为符号扩展后要给 8 位位移量乘以 2，所以与转移目标的相对距离在 -256 字节～ +254 字节范围内。

(2) 注意事项

因为该指令是延迟转移指令，所以发生转移时，先执行该指令的下一条指令，然后执行转移目标指令。

该指令与其下一条指令之间，不接受中断。

下一条指令为转移指令时，将其识别为槽非法指令。

该指令被分配在延迟转移指令紧接着的延迟槽时，识别为槽非法指令。

未到达转移目标时，需要结合 BRA、 JMP 指令来进行相应处理。

(3) 操作内容

```
BFS(int d)    /* BFS disp */
{
      int disp;
      unsigned int temp;

      temp = PC;
      if((d&0x80)==0)
            disp = (0x000000FF & d);
      else disp = (0xFFFFFF00 | d);
      if(T==0)
            PC = PC + 4 + (disp<<1);
      else PC += 4;
      Delay_Slot(temp+2);
}
```

(4) 使用示例

```
CLRT                    ; 通常 T=0
BT/S  TRGET_T           ; 因为 T=0，所以不转移。
NOP                     ;
BF/S  TRGET_F           ; 因为 T=0，所以转移至 TRGET-F。
ADD   R0,R1             ; 在转移前执行。
NOP                     ;
TRGET_F:                ;←BF/S 指令的转移目标
```

(5) 有可能发生的异常

- 槽非法指令异常。

10.1.7 BRA BRAnch 转移指令

无条件转移 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
BRA label	PC+4+disp ×2→PC	1010dddddddddddd	1	—

(1) 说明

该指令为无条件延迟转移指令。转移目标为地址 （PC+4+ 位移量 × 2）。 PC 源值为 BRA 指令地址。因为符号扩展后要给 12 位位移量乘以 2，所以与转移目标的相对距离在 -4096 字节～ +4094 字节范围内。未到达转移目标时，可通过 JMP 指令实现此转移。

(2) 注意事项

因为该指令为延迟转移指令，所以先执行该指令的下一条指令，然后执行转移目标指令。
该指令与其下一条指令之间，不接受中断。下一条指令为转移指令时，将其识别为槽非法指令。

(3) 操作内容

```
BRA(int d)    /* BRA disp */
{
      int disp;
      unsigned int temp;

      temp = PC;
      if((d&0x800)==0)
            disp = (0x00000FFF & d);
      else disp = (0xFFFFF000 | d);
      PC = PC + 4 + (disp<<1);
      Delay_Slot(temp+2);
}
```

(4) 使用示例

```
        BRA   TRGET          ; 向 TRGET 转移。
        ADD   R0,R1          ; 转移前，先执行 ADD。
        NOP                  ;
TRGET:                       ;←BRA 指令的转移目标
```

(5) 有可能发生的异常

- 槽非法指令异常

10.1.8 BRAF BRAnch Far 转移指令

无条件转移 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
BRAF Rn	PC+4+Rn→ PC	0000nnnn00100011	1	—

(1) 说明

该指令为无条件延迟转移指令。转移目标为地址 （PC+4+Rn）。转移目标地址为 PC 加上 4 与通用寄存器 Rn 的 32 位内容后的地址。

(2) 注意事项

因为是该指令是延迟转移指令，所以先执行该指令的下一条指令，然后执行转移目标指令。
该指令与其下一条指令之间，不接受中断。下一条指令为转移指令时，将其识别为槽非法指令。

(3) 操作内容

```
BRAF(int n)   /* BRAF Rn */
{
     unsigned int temp;

     temp = PC;
     PC = PC + 4 + R[n];
     Delay_Slot(temp+2);
}
```

(4) 使用示例

```
            MOV.L #(TRGET-BRAF_PC),R0      ; 设定位移量。
        BRAF  R0                       ; 向 TRGET 转移。
        ADD   R0,R1                    ; 转移之前，先执行 ADD。
      BRAF_PC:                         ;
        NOP
        TRGET:                         ;←BRAF 指令的转移目标
```

(5) 有可能发生的异常

- 槽非法指令异常

10.1.9 BT Branch if True 转移指令

条件转移

格式	操作概略	操作码	执行状态	T 位
BT label	T=1 时 PC+4+disp ×2→PC, T=0 时 nop	10001001dddddddd	1	—

(1) 说明

该指令为参考 T 位的带条件的转移指令。 T=1 时，转移。相反， T=0 时，不转移。

转移目标为地址（PC+4+ 位移量 ×2）。 PC 源值为 BT 指令地址。因为符号扩展后要给 8 位位移量乘以 2，所以与转移目标的相对距离在 -256 字节～ +254 字节范围内。

(2) 注意事项

未到达转移目标时，需要结合 BRA、 JMP 指令来进行相应处理。

(3) 操作内容

```
BT(int d)    /* BT disp */
{
    int disp;

    if((d&0x80)==0)
          disp = (0x000000FF & d);
    else disp = (0xFFFFFF00 | d);
    if(T==1)
          PC = PC + 4 + (disp<<1);
    else PC += 2;
}
```

(4) 使用示例

```
     SETT                          ; 通常 T=1
     BF    TRGET_F                 ; 因为 T=1，所以不转移。
     BT    TRGET_T                 ; 因为 T=1，所以向 TRGET_T 转移。
     NOP                           ;
     NOP                           ;
TRGET_T:                           ;←BT 指令的转移目标
```

(5) 有可能发生的异常

- 槽非法指令异常

10.1.10 BT/S Branch if True with delay Slot 转移指令

带延迟的条件转移 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
BT/S label	T=1 时 PC+4+disp× 2→PC, T=0 时 nop	10001101dddddddd	1	—

(1) 说明

该指令为参考 T 位的带延迟的条件转移指令。 T=1 时，转移。 T=0 时，不转移。

PC 源值为 BT/S 指令地址。因为符号扩展后要给 8 位位移量乘以 2，所以与转移目标的相对距离在 -256 字节～ +254 字节范围内。

(2) 注意事项

因为该指令是延迟转移指令，所以在发生转移时，先执行该指令的下一条指令，然后执行转移目标指令。

该指令与其下一条指令之间，不接受中断。

下一条指令为转移指令时，将其识别为槽非法指令。未到达转移目标时，需要结合 BRA、 JMP 指令进行相应处理。

(3) 操作内容

```
BTS(int d)    /* BTS disp */
{
    int disp;
    unsigned temp;

    temp = PC;
    if((d&0x80)==0)
          disp = (0x000000FF & d);
    else disp = (0xFFFFFF00 | d);
    if(T==1)
          PC = PC + 4 + (disp<<1);
    else PC += 4;
    Delay_Slot(temp+2);
}
```

(4) 使用示例

```
    SETT                ; 通常 T=1
    BF/S TRGET_F        ; 因为 T=1，所以不转移。
    NOP                 ;
    BT/S TRGET_T        ; 因为 T=1，所以转移至 TRGET_T。
    ADD R0,R1           ; 在转移之前执行。
    NOP                 ;
  TRGET_T:              ;←BT/S 指令的转移目标
```

(5) 有可能发生的异常

- 槽非法指令异常

10.1.11 CLRMAC CLeaR MAC register 系统控制指令

清除 MAC 寄存器

格式	操作概略	操作码	执行状态	T 位
CLRMAC	0→MACH,MACL	0000000000101000	1	—

(1) 说明

清除 MACH、 MACL 寄存器。

(2) 注意事项

无

(3) 操作内容

```
CLRMAC( )    /* CLRMAC */
{
     MACH = 0;
     MACL = 0;
     PC += 2;
}
```

(4) 使用示例

```
CLRMAC                  ; 清除 MAC 寄存器后进行初始化。
MAC.W  @R0+,@R1+        ; 乘法累加运算
MAC.W  @R0+,@R1+        ;
```

10.1.12 CLRS CLeaR Sbit 系统控制指令

清除 S 位

格式	操作概略	操作码	执行状态	T 位
CLRS	0→S	0000000001001000	1	—

(1) 说明

将 S 位清 0。

(2) 注意事项

无

(3) 操作内容

```
CLRS( )    /* CLRS */
{
     S = 0;
     PC += 2;
}
```

(4) 使用示例

```
     CLRS              ;执行前 S=1
                       ;执行后 S=0
```

10.1.13 CLRT CLeaR Tbit 系统控制指令

清除 T 位

格式	操作概略	操作码	执行状态	T 位
CLRT	0→T	0000000000001000	1	—

(1) 说明

清除 T 位。

(2) 注意事项

无

(3) 操作内容

```
CLRT( )    /* CLRT */
{
     T = 0;
     PC += 2;
}
```

(4) 使用示例

```
CLRT            ;执行前 T=1
                ;执行后 T=0
```

10.1.14 CMP/cond CoMPare conditionally 算术运算指令

比较

格式	操作概略	操作码	执行状态	T 位
CMP/EQ Rm,Rn	Rn=Rm 时 1→T、 此外 0→T	0011nnnnmmmm0000	1	比较结果
CMP/GE Rm,Rn	有符号且 Rn ≥ Rm 时 1→T、 此外 0→T	0011nnnnmmmm0011	1	比较结果
CMP/GT Rm,Rn	有符号且 Rn > Rm 时 1→T、 此外 0→T	0011nnnnmmmm0111	1	比较结果
CMP/HI Rm,Rn	无符号且 Rn > Rm 时 1→T、 此外 0→T	0011nnnnmmmm0110	1	比较结果
CMP/HS Rm,Rn	无符号且 Rn ≥ Rm 时 1→T、 此外 0→T	0011nnnnmmmm0010	1	比较结果
CMP/PL Rn	Rn > 0 时 1→T、 此外 0→T	0100nnnn00010101	1	比较结果
CMP/PZ Rn	Rn ≥ 0 时 1→T、 此外 0→T	0100nnnn00010001	1	比较结果
CMP/STR Rm,Rn	任意字节相等时 1→T、 此外 0→T	0010nnnnmmmm1100	1	比较结果
CMP/EQ #imm,R0	R0=imm 时 1→T、 此外 0→T	10001000iiiiiiii	1	比较结果

(1) 说明

比较通用寄存器 Rn 与 Rm，被指定的条件（cond）成立时， T 位置位。条件不成立时清除 T 位。 Rn 的内容不变化。可指定 9 个条件。关于 PZ 与 PL 这 2 个条件，为 0 与 Rn 的比较。

关于 EQ 条件，可对符号扩展后的 8 位立即数与 R0 进行比较。 R0 的内容不变化。

助记符		说明
CMP/EQ	Rm,Rn	Rn=Rm 时 T=1
CMP/GE	Rm,Rn	作为有符号值 Rn ≥ Rm 时 T=1
CMP/GT	Rm,Rn	作为有符号值 Rn > Rm 时 T=1
CMP/HI	Rm,Rn	作为无符号值 Rn > Rm 时 T=1
CMP/HS	Rm,Rn	作为无符号值 Rn ≥ Rm 时 T=1
CMP/PL	Rn	Rn > 0 时 T=1
CMP/PZ	Rn	Rn ≥ 0 时 T=1
CMP/STR	Rm,Rn	任意字节相等时 T=1
CMP/EQ	#imm,R0	R0=imm 时 T=1

(2) 注意事项

无

(3) 操作内容

```
CMPEQ(long m, long n)    /* CMP_EQ Rm,Rn */
{
    if(R[n]==R[m]) T = 1;
    else T = 0;
    PC += 2;
}
CMPGE(long m, long n)    /* CMP_GE Rm,Rn */
{
    if((long)R[n]>=(long)R[m]) T = 1;
    else T = 0;
    PC += 2;
}

CMPGT(long m, long n)    /* CMP_GT Rm,Rn */
{
    if((long)R[n]>(long)R[m]) T = 1;
    else T = 0;
    PC += 2;
}

CMPHI(long m, long n)    /* CMP_HI Rm,Rn */
{
    if((unsigned long)R[n] > (unsigned long)R[m]) T = 1;
    else T = 0;
    PC += 2;
}

CMPHS(long m, long n)    /* CMP_HS Rm,Rn */
{
    if((unsigned long)R[n]>=(unsigned long)R[m]) T = 1;
    else T = 0;
    PC += 2;
}

CMPPL(long n)              /* CMP_PL Rn */
{
    if((long)R[n]>0) T = 1;
    else T = 0;
    PC += 2;
}
```

```
CMPPZ(long n)    /* CMP_PZ Rn */
{
    if((long)R[n]>=0) T = 1;
    else T = 0;
    PC += 2;
}

CMPSTR(long m, long n)    /* CMP_STR Rm,Rn */
{
    unsigned long temp;
    long HH,HL,LH,LL;

    temp = R[n] ^ R[m];
    HH = (temp & 0xFF000000) >> 24;
    HL = (temp & 0x00FF0000) >> 16;
    LH = (temp & 0x0000FF00) >> 8;
    LL = temp & 0x000000FF;
    HH = HH && HL && LH && LL;
    if(HH==0) T = 1;
    else T = 0;
    PC += 2;
}

CMPIM(long i)    /* CMP_EQ #imm,R0 */
{
    long imm;

    if((i&0x80)==0) imm = (0x000000FF & (long i));
    else imm = (0xFFFFFF00 | (long i));
    if(R[0]==imm) T = 1;
    else T = 0;
    PC += 2;
}
```

(4) 使用示例

```
CMP/GE          R0,R1           ;R0=H'7FFFFFFF,R1=H'80000000
BT              TRGET_T         ; 因为 T=0，所以不转移。
CMP/HS          R0,R1           ;R0=H'7FFFFFFF,R1=H'80000000
BT              TRGET_T         ; 因为 T=1，所以转移。
CMP/STR         R2,R3           ;R2="ABCD",R3="XYCZ"
BT              TRGET_T         ; 因为 T=1，所以转移。
```

10.1.15 DIV0S DIVide(step0) as Signed 算术运算指令

初始化带符号的除法运算

格式	操作概略	操作码	执行状态	T 位
DIV0S Rm,Rn	Rn 的 MSB→Q, Rm 的 MSB→M, M^Q→T	0010nnnnmmmm0111	1	计算结果

(1) 说明

该指令进行带符号的除法运算的初始设定。结合紧接着该指令的进行 1 位除法运算的 DIV1 等指令，反复进行除法运算，并求商。详细内容请参考 DIV1 的说明。

(2) 注意事项

无

(3) 操作内容

```
DIV0S(long m, long n)    /* DIV0S Rm,Rn */
{
    if((R[n] & 0x80000000)==0) Q = 0;
    else Q = 1;
    if((R[m] & 0x80000000)==0) M = 0;
    else M = 1;
    T = !(M==Q);
    PC += 2;
}
```

(4) 使用示例

请参考 DIV1 的使用示例。

10.1.16 DIV0U DIVide (step0) as Unsigned 算术运算指令

初始化无符号的除法运算

格式	操作概略	操作码	执行状态	T 位
DIV0U	0→M/Q/T	0000000000011001	1	0

(1) 说明

该指令进行无符号的除法运算的初始设定。结合紧接着该指令的进行 1 位除法运算的 DIV1 指令，反复进行除法运算，并求商。详细内容请参考 DIV1 的说明。

(2) 注意事项

无

(3) 操作内容

```
DIV0U( )        /* DIV0U */
{
    M = Q = T = 0;
    PC += 2;
}
```

(4) 使用示例

请参考 DIV1 的使用示例。

10.1.17 DIV1 DIVide 1 step 算术运算指令

除法运算

格式	操作概略	操作码	执行状态	T 位
DIV1 Rm,Rn	1 步除法运算 (Rn÷Rm)	0011nnnnmmmm0100	1	计算结果

(1) 说明

该指令为用 Rm 的内容 （除数）对通用寄存器 Rn 的 32 位内容 （被除数）进行 1 位除法运算 （1 步除法运算）的指令。该指令单独或与其他指令结合并反复执行后求商。反复执行时，请勿改写指定的寄存器与 M、Q、 T 位。

1 步除法运算是指将被除数左移 1 位，然后减去除数，并根据结果是正是负，在 Q 位反映商位。

通过除法运算求余数时，请使用 DIV1 指令求得商后，按照以下公式

（余数） = （被除数） — （除数） × （商）

求余数。

不提供零除法运算与上溢的检测。进行除法运算前，请检查是否是零除法运算与上溢除法运算。不提供余数运算。求除数与求得的商的乘积，然后从被除数减去所得的乘积，再求余数。

先以 DIV0S 或 DIV0U 进行初始设定。反复执行除数的位数次的 DIV1。需要商大于等于 17 位时，将 ROTCL 放在 DIV1 之前。除法运算的具体顺序请参考使用示例。

(2) 注意事项

无

(3) 操作内容

```
DIV1(long m, long n)    /* DIV1 Rm,Rn */
{
   unsigned long tmp0, tmp2;
   unsigned char old_q, tmp1;

   old_q = Q;
   Q = (unsigned char)((0x80000000 & R[n]) != 0);
   tmp2 = R[m];
   R[n] <<= 1;
   R[n] |= (unsigned long)T;

    switch(old_q){
      case 0:  switch(M){
                    case 0:  tmp0 = R[n];
                             R[n] -= tmp2;
                             tmp1 = (R[n]>tmp0);
                             switch(Q){
                               case 0:  Q = tmp1;
                                        break;
                               case 1:  Q = (unsigned char)(tmp1==0);
                                        break;
```

```
                        }
                        break;
                    case 1:  tmp0 = R[n];
                        R[n] += tmp2;
                        tmp1 = (R[n]<tmp0);
                        switch(Q){
                           case 0:  Q = (unsigned char)(tmp1==0);
                                    break;
                           case 1:  Q = tmp1;
                                    break;
                        }
                        break;
                }
                break;
        case 1:  switch(M){
                   case 0:  tmp0 = R[n];
                            R[n] += tmp2;
                            tmp1 = (R[n]<tmp0);
                            switch(Q){
                              case 0:  Q = tmp1;
                                       break;
                              case 1:  Q = (unsigned char)(tmp1==0);
                                       break;
                            }
                            break;
                   case 1:  tmp0 = R[n];
                            R[n] -= tmp2;
                            tmp1 = (R[n]>tmp0);
                            switch(Q){
                              case 0:  Q = (unsigned char)(tmp1==0);
                                       break;
                              case 1:  Q = tmp1;
                                       break;
                            }
                            break;
                }
                break;
    }
    T = (Q==M);
    PC += 2;
}
```

(4) 使用示例

- 例1

```
                                ;R1(32 位 ) ÷ R0(16 位 )=R1(16 位 ): 无符号
SHLL16          R0              ; 将除数设定为高 16 位，将低 16 位置 0
TST             R0,R0           ; 检查零除法运算
BT              ZERO_DIV        ;
CMP/HS          R0,R1           ; 检查上溢
BT               OVER_DIV        ;
DIV0U                           ; 初始化标志
.arepeat        16              ;
DIV1            R0,R1           ; 反复 16 次
.aendr                          ;
ROTCL           R1              ;
EXTU.W          R1,R1           ;R1= 商
```

- 例2

```
                                 ;R1:R2(64 位 ) ÷ R0(32 位 )=R2(32 位 ): 无符号
 TST            R0,R0            ; 检查零除法运算
BT              ZERO_DIV        ;
CMP/HS          R0,R1           ; 检查上溢
BT  OVER_DIV                     ;
DIV0U                           ; 初始化标志
.arepeat        32              ;
ROTCL           R2              ; 反复 32 次
DIV1R0,R1;
.aendr ;
ROTCL           R2              ;R2= 商
```

- 例3

```
                                    ;R1(16 位 ) ÷ R0(16 位 )=R1(16 位 ): 带符号
SHLL16      R0              ; 将除数设定为高 16 位，将低 16 位置 0
EXTS.W      R1,R1           ; 将被除数符号扩展至 32 位
XOR         R2,R2           ;R2=0
MOV         R1,R3           ;
ROTCL       R3              ;
SUBC        R2,R1           ; 被除数为负时， -1。
DIV0S       R0,R1           ; 初始化标志
.arepeat16;
DIV1        R0,R1           ; 反复 16 次
.aendr    ;
EXTS.W      R1,R1           ;
ROTCL       R1              ;R1= 商 （1 的补码）
ADDC        R2,R1           ; 商的 MSB 为 1 时， +1，然后转换为 2 的补码。
EXTS.W      R1,R1           ;R1= 商 （2 的补码）
```

- 例4

```
                                        ;R2(32 位 ) ÷ R0(32 位 )=R2(32 位 ): 带符号
MOV         R2,R3           ;
ROTCL       R3              ;
SUBC        R1,R1           ; 将被除数符号扩展至 64 位 （R1:R2）
XOR         R3,R3           ;R3=0
SUBC        R3,R2           ; 被除数为负时， -1，然后转换为 1 的补码。
DIV0S       R0,R1           ; 初始化标志
.arepeat    32              ;
ROTCL       R2              ; 反复 32 次
DIV1        R0,R1           ;
.aendr                          ;
ROTCL       R2              ;R2= 商 （1 的补码）
ADDC        R3,R2           ; 商的 MSB 为 1 时， +1，然后转换为 2 的补码
                            ;R2= 商 （2 的补码）
```

10.1.18 DMULS.L Double-length MULtiply as Signed 算术运算指令

带符号的双精度乘法运算

格式	操作概略	操作码	执行状态	T 位
DMULS.L Rm,Rn	带符号 Rn×Rm→MAC	0011nnnnmmmm1101	2	—

(1) 说明

对通用寄存器 Rn 的内容与 Rm 进行 32 位乘法运算，将 64 位结果保存在 MACH 寄存器与 MACL 寄存器。以带符号的算术运算进行运算。

(2) 注意事项

无

(3) 操作内容

```
DMULS(long m, long n)  /* DMULS.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;
    long tempm,tempn,fnLmL;

    tempn = (long)R[n];
    tempm = (long)R[m];
    if(tempn<0) tempn = 0-tempn;
    if(tempm<0) tempm = 0-tempm;
    if((long)(R[n]^R[m])<0) fnLmL = -1;
    else fnLmL = 0;

    temp1 = (unsigned long)tempn;
    temp2 = (unsigned long)tempm;

    RnL = temp1 & 0x0000FFFF;
    RnH = (temp1>>16) & 0x0000FFFF;
    RmL = temp2 & 0x0000FFFF;
    RmH = (temp2>>16) & 0x0000FFFF;

    temp0 = RmL*RnL;
    temp1 = RmH*RnL;
    temp2 = RmL*RnH;
    temp3 = RmH*RnH;

    Res2 = 0;
    Res1 = temp1 + temp2;
    if(Res1<temp1) Res2 += 0x00010000;
    temp1 = (Res1<<16) & 0xFFFF0000;
```

```
      Res0 = temp0 + temp1;
      if(Res0<temp0) Res2++;

      Res2 = Res2 + ((Res1>>16) & 0x0000FFFF) + temp3;

      if(fnLmL<0) {
      Res2 = ~Res2;
     if(Res0==0) Res2++;
     else  Res0 = (~Res0)+1;
     }

      MACH = Res2;
      MACL = Res0;
      PC += 2;
  }
```

(4) 使用示例

```
  DMULS.L          R0,R1           ; 执行前 R0=H'FFFFFFFE,R1=H'00005555
                                   ; 执行后 MACH=H'FFFFFFFF,MACL=H'FFFF5556
  STS              MACH,R0         ; 获得运算结果（高位）
  STS              MACL,R1         ; 获得运算结果（低位）
```

10.1.19 DMULU.L Double-length MULtiply as Unsigned 算术运算指令

无符号双精度乘法运算

格式	操作概略	操作码	执行状态	T 位
DMULU.L Rm,Rn	无符号 Rn×Rm→MAC	0011nnnnmmmm0101	2	—

(1) 说明

对通用寄存器 Rn 的内容与 Rm 进行 32 位乘法运算，将 64 位结果保存在 MACH 寄存器与 MACL 寄存器。以无符号算术运算进行运算。

(2) 注意事项

无

(3) 操作内容

```
DMULU(long m, long n)  /* DMULU.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;

    RnL = R[n] & 0x0000FFFF;
    RnH = (R[n]>>16) & 0x0000FFFF;

    RmL =R [m] & 0x0000FFFF;
    RmH = (R[m]>>16) & 0x0000FFFF;

    temp0 = RmL*RnL;
    temp1 = RmH*RnL;
    temp2 = RmL*RnH;
    temp3 = RmH*RnH;

    Res2=0
    Res1 = temp1 + temp2;
    if(Res1<temp1) Res2 += 0x00010000;

    temp1 = (Res1<<16)&0xFFFF0000;
    Res0 = temp0 + temp1;
    if(Res0<temp0) Res2++;

    Res2 = Res2 + ((Res1>>16)&0x0000FFFF) + temp3;

    MACH = Res2;
    MACL = Res0;
    PC += 2;
}
```

(4) 使用示例

```
DMULU.L     R0,R1          ;执行前 R0=H'FFFFFFFE,R1=H'00005555
                           ;执行后 MACH=H'00005554,MACL=H'FFFF5556
STS         MACH,R0        ;获得运算结果 （高位）
STS         MACL,R1        ;获得运算结果 （低位）
```

10.1.20 DT Decrement and Test 算术运算指令

递减测试

格式	操作概略	操作码	执行状态	T 位
DT Rn	Rn － 1→Rn,Rn 为 0 时 1→T Rn 不为 0 时 0→T	0100nnnn00010000	1	比较结果

(1) 说明

将通用寄存器 Rn 的内容递减 1，将结果与 0 （零）进行比较。结果为 0 时，将 T 位置 1。结果不为 0 时，将 T 位置 0。

(2) 注意事项

无

(3) 操作内容

```
DT(long n)/* DT Rn */
{
    R[n]--;
    if(R[n]==0) T = 1;
    else T = 0;
    PC += 2;
}
```

(4) 使用示例

```
        MOV #4,R5           ;设定循环次数。
LOOP:
        ADD R0,R1           ;
        DT  R5              ;递减 R5 的值，判断是否为 0。
        BF  LOOP            ;如果 T=0，则向 LOOP 转移 （此例中循环 4 次）。
```

10.1.21 EXTS EXTend as Signed 算术运算指令

符号扩展

格式	操作概略	操作码	执行状态	T 位
EXTS.B Rm,Rn	将 Rm 从字节进行符号扩展 →Rn	0110nnnnmmmm1110	1	—
EXTS.W Rm,Rn	将 Rm 从字进行符号扩展 →Rn	0110nnnnmmmm1111	1	—

(1) 说明

将通用寄存器 Rm 的内容进行符号扩展，并将结果保存在 Rn。

进行字节指定时，将 Rm 的 bit7 的内容传送至 Rn 的 bit8 ～ bit31。进行字指定时，将 Rm 的 bit15 的内容传送至 Rn 的 bit16 ～ bit31。

(2) 注意事项

无

(3) 操作内容

```
EXTSB(long m, long n)/* EXTS.B Rm,Rn */
{
    R[n] = R[m];
    if((R[m]&0x00000080)==0) R[n] &= 0x000000FF;
    else R[n] |= 0xFFFFFF00;
    PC += 2;
}
EXTSW(long m, long n)/* EXTS.W Rm,Rn */
{
    R[n] = R[m];
    if((R[m]&0x00008000)==0) R[n] &= 0x0000FFFF;
    else R[n] |= 0xFFFF0000;
    PC += 2;
}
```

(4) 使用示例

```
EXTS.B      R0,R1          ;执行前 R0=H'00000080
                           ;执行后 R1=H'FFFFFF80
EXTS.W      R0,R1          ;执行前 R0=H'00008000
                           ;执行后 R1=H'FFFF8000
```

10.1.22 EXTU EXTend as Unsigned 算术运算指令

零扩展

格式	操作概略	操作码	执行状态	T 位
EXTS.B Rm,Rn	将 Rm 从字节进行零扩展 →Rn	0110nnnnmmmm1100	1	—
EXTS.W Rm,Rn	将 Rm 从字进行零扩展 →Rn	0110nnnnmmmm1101	1	—

(1) 说明

将通用寄存器 Rm 的内容进行零扩展，并将结果保存在 Rn。
进行字节指定时，将 0 传送至 Rn 的 bit8 ～ bit31。进行字指定时，将 0 传送至 Rn 的 bit16 ～ bit31。

(2) 注意事项

无

(3) 操作内容

```
EXTUB(long m, long n)/* EXTU.B Rm,Rn */
{
    R[n] = R[m];
    R[n] &= 0x000000FF;
    PC += 2;
}

EXTUW(long m, long n)/* EXTU.W Rm,Rn */
{
    R[n] = R[m];
    R[n] &= 0x0000FFFF;
    PC += 2;
}
```

(4) 使用示例

```
EXTU.B          R0,R1           ;执行前 R0=H'FFFFFF80
                                ;执行后 R1=H'00000080
EXTU.W          R0,R1           ;执行前 R0=H'FFFF8000
                                ;执行后 R1=H'00008000
```

10.1.23 ICBI Instruction Cache Block Invalidate 数据传输指令

指令高速缓存块的无效化

格式	操作概略	操作码	执行状态	T 位
ICBI @Rn	使逻辑地址 Rn 所示的指令高速缓存无效	0000nnnn11100011	13	—

(1) 说明

将 Rn 的内容作为有效地址，存取指令高速缓存。命中高速缓存命中时，将对应的高速缓存块设定为无效（V bit=0）。此时，不进行回写。未命中高速缓存时，如果对不可高速缓存区进行存取，则不使其无效。

(2) 注意事项

无

(3) 操作内容

```
ICBI(int n)/* ICBI @Rn */
{
    invalidate_instruction_cache_block(R[n]);
    PC += 2;
}
```

(4) 使用示例

在 RAM 上改写程序，在执行的程序中，为了使指令高速缓存不在保持改写前的原有指令状态下执行，以 ICBI 指令将该指令高速缓存块设定为无效。

(5) 有可能发生的异常

即使不执行无效化时，也可能发生下列异常。

- 指令TLB多重命中异常
- 指令TLB未命中异常
- 指令TLB保护违反异常
- 指令地址错误
- 槽非法指令异常

10.1.24 JMP JuMP 转移指令

无条件转移 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
JMP @Rn	Rn→PC	0100nnnn00101011	1	—

(1) 说明

无条件地向 Rn 所指定的地址进行延迟转移。

(2) 注意事项

因为该指令为延迟转移指令，所以先执行该指令的下一条指令，然后执行转移目标指令。
该指令与其下一条指令之间，不接受中断。下一条指令为转移指令时，将其识别为槽非法指令。

(3) 操作内容

```
JMP(int n)/* JMP @Rn */
{
    unsigned int temp;

    temp = PC;
    PC = R[n];
    Delay_Slot(temp+2);
}
```

(4) 使用示例

```
                    MOV.L    JMP_TABLE,R0      ;R0=TRGET 地址
        JMP         @R0                ; 向 TRGET 转移。
        MOV         R0,R1              ; 转移之前，先执行 MOV。
        .align      4
JMP_TABLE:.data.l   TRGET              ; 跳转表
     • • • • • •
TRGET:  ADD         #1,R1              ; ← 转移目标
```

(5) 有可能发生的异常

- 槽非法指令异常

10.1.25 LDC LoaD to Control register 系统控制指令

加载到控制寄存器 （特权指令）

格式	操作概略	操作码	执行状态	T 位
LDC Rm, GBR	Rm→GBR	0100mmmm00011110	1	—
LDC Rm, VBR	Rm→VBR	0100mmmm00101110	1	—
LDC Rm, SGR	Rm→SGR	0100mmmm00111010	4	—
LDC Rm, SSR	Rm→SSR	0100mmmm00111110	1	—
LDC Rm, SPC	Rm→SPC	0100mmmm01001110	1	—
LDC Rm, DBR	Rm→DBR	0100mmmm11111010	4	—
LDC Rm, R0_BANK	Rm→R0_BANK	0100mmmm10001110	1	—
LDC Rm, R1_BANK	Rm→R1_BANK	0100mmmm10011110	1	—
LDC Rm, R2_BANK	Rm→R2_BANK	0100mmmm10101110	1	—
LDC Rm, R3_BANK	Rm→R3_BANK	0100mmmm10111110	1	—
LDC Rm, R4_BANK	Rm→R4_BANK	0100mmmm11001110	1	—
LDC Rm, R5_BANK	Rm→R5_BANK	0100mmmm11011110	1	—
LDC Rm, R6_BANK	Rm→R6_BANK	0100mmmm11101110	1	—
LDC Rm, R7_BANK	Rm→R7_BANK	0100mmmm11111110	1	—
LDC.L @Rm+, GBR	(Rm)→GBR, Rm+4→Rm	0100mmmm00010111	1	—
LDC.L @Rm+, VBR	(Rm)→VBR, Rm+4→Rm	0100mmmm00100111	1	—
LDC.L @Rm+, SGR	(Rm)→SGR, Rm+4→Rm	0100mmmm00110110	4	—
LDC.L @Rm+, SSR	(Rm)→SSR, Rm+4→Rm	0100mmmm00110111	1	—
LDC.L @Rm+, SPC	(Rm)→SPC, Rm+4→Rm	0100mmmm01000111	1	—
LDC.L @Rm+, DBR	(Rm)→DBR, Rm+4→Rm	0100mmmm11110110	4	—
LDC.L @Rm+, R0_BANK	(Rm)→R0_BANK, Rm+4→Rm	0100mmmm10000111	1	—
LDC.L @Rm+, R1_BANK	(Rm)→R1_BANK, Rm+4→Rm	0100mmmm10010111	1	—
LDC.L @Rm+, R2_BANK	(Rm)→R2_BANK, Rm+4→Rm	0100mmmm10100111	1	—
LDC.L @Rm+, R3_BANK	(Rm)→R3_BANK, Rm+4→Rm	0100mmmm10110111	1	—
LDC.L @Rm+, R4_BANK	(Rm)→R4_BANK, Rm+4→Rm	0100mmmm11000111	1	—
LDC.L @Rm+, R5_BANK	(Rm)→R5_BANK, Rm+4→Rm	0100mmmm11010111	1	—
LDC.L @Rm+, R6_BANK	(Rm)→R6_BANK, Rm+4→Rm	0100mmmm11100111	1	—
LDC.L @Rm+, R7_BANK	(Rm)→R7_BANK, Rm+4→Rm	0100mmmm11110111	1	—

(1) 说明

把源操作数保存在控制寄存器 GBR、 VBR、 SSR、 SPC、 DBR、 SGR 或 R0_BANK ～ R7_BANK。

(2) 注意事项

LDC, LDC.L 指令 （除 LDC Rm, GBR 与 LDC.L @Rm+,GBR 外），只可在特权模式下使用。如果在用户模式下使用，则发生非法指令异常。但是， LDC Rm,GBR 与 LDC.L @Rm+,GBR 也可在用户模式下使用。

SR 寄存器的 RB 位为 0 时， LDC Rm, Rn_BANK, LDC.L @Rm, Rn_BANK 指令的 Rn_BANK 表示 Rn_BANK1； RB 位为 1 时， LDC Rm, Rn_BANK, LDC.L @Rm, Rn_BANK 指令的 Rn_BANK 表示 Rn_BANK0。

(3) 操作内容

```
LDCGBR(int m)     /* LDC Rm,GBR */
{
      GBR = R[m];
       PC += 2;
}
LDCVBR(int m)     /* LDC Rm,VBR : Privileged */
{
       VBR = R[m];
       PC += 2;
}
 LDCSGR(int m)     /* LDC Rm,SGR : Privileged */
{
       SGR = R[m];
       PC += 2;
}
LDCSSR(int m     )/* LDC Rm,SSR : Privileged */
{
       SSR = R[m];
       PC += 2;
}
LDCSPC(int m)     /* LDC Rm,SPC : Privileged */
{
       SPC = R[m];
       PC += 2;
}
LDCDBR(int m)     /* LDC Rm,DBR : Privileged */
{
       DBR = R[m];
       PC += 2;
}
LDCRn BANK(int m)     /* LDC Rm,Rn BANK : Privileged */
                      /* n=0-7 */
{
       Rn BANK = R[m];
```

```
        PC += 2;
}
LDCMGBR(int m)      /* LDC.L @Rm+,GBR */
{
        GBR = Read Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDCMVBR(int m)      /* LDC.L @Rm+,VBR : Privileged */
{
        VBR = Read Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDCMSGR(int m)      /* LDC.L @Rm+,SGR : Privileged */
{
        SGR = Read Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDCMSSR(int m)      /* LDC.L @Rm+,SSR : Privileged */
{
        SSR = Read Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDCMSPC(int m)      /* LDC.L @Rm+,SPC : Privileged */
{
        SPC = Read Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDCMDBR(int m)      /* LDC.L @Rm+,DBR : Privileged */
{
        DBR = Read Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDCMRn BANK(Long m)      /* LDC.L @Rm+,Rn BANK : Privileged */
                         /*n=0-7 */
{
        Rn BANK=Read Long(R[m]);
        R[m] += 4;
        PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 普通非法指令异常
- 槽非法指令异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.1.26 LDS LoaD to System register 系统控制指令

加载到系统寄存器

格式	操作概略	操作码	执行状态	T 位
LDS Rm,MACH	Rm→MACH	0100mmmm00001010	1	—
LDS Rm,MACL	Rm→MACL	0100mmmm00011010	1	—
LDS Rm,PR	Rm→PR	0100mmmm00101010	1	—
LDS.L @Rm+,MACH	(Rm)→MACH, Rm+4→Rm	0100mmmm00000110	1	—
LDS.L @Rm+,MACL	(Rm)→MACL, Rm+4→Rm	0100mmmm00010110	1	—
LDS.L @Rm+,PR	(Rm)→PR, Rm+4→Rm	0100mmmm00100110	1	—

(1) 说明

将源操作数保存至系统寄存器 MACH、 MACL、 PR。

(2) 注意事项

无

(3) 操作内容

```
LDSMACH(long m)      /* LDS Rm,MACH  */
{
       MACH = R[m];
       PC += 2;
}
LDSMACL(long m)     /* LDS Rm,MACL  */
{
       MACL = R[m];
       PC += 2;
}
LDSPR(long  m)     /* LDS Rm,PR  */
{
       PR = R[m];
       PC += 2;
}
LDSMMACH(long  m)     /* LDS.L @Rm+,MACH  */
{
       MACH = Read_Long(R[m]);
       R[m] += 4;
```

```
        PC += 2;
}
LDSMMACL(long  m)     /* LDS.L @Rm+,MACL  */
{
        MACL = Read_Long(R[m]);
        R[m] += 4;
        PC += 2;
}
LDSMPR(long  m)     /* LDS.L @Rm+,PR  */
{
        PR=Read_Long(R[m]);
        R[m] += 4;
        PC += 2;
}
```

(4) 使用示例

```
LDS          R0、 PR            ;执行前 R0=H'12345678、 PR=H'00000000
                                ;执行后 PR=H'12345678
LDS.L        @R15+,MACL         ;执行前 R15=H'10000000
                                ;执行后 P15=H'10000004,MACL=(H'10000000)
```

(5) 有可能发生的异常

仅执行 LDS.L 指令时，可能会发生以下异常：

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.1.27 LDTLB LoaD PTEH/PTEL to TLB 系统控制指令

加载到 TLB （特权指令）

格式	操作概略	操作码	执行状态	T 位
LDTLB	PTEH/PTEL→TLB	0000000000111000	1	—

(1) 说明

将 PTEH/PTEL 寄存器内容保存至 MMUCR.URC （MMC 控制寄存器的随机计数器区域）所指定的 TLB（Translation Lookaside Buffer）。

LDTLB 指令为特权指令，只可在特权模式下使用。如果在用户模式下使用，则发生非法指令异常。

(2) 注意事项

因为该指令为将 PTEH/PTEL 寄存器加载到 TLB 的指令，所以请在 MMU 为禁止状态或 MMU 为允许状态时在逻辑空间的 P1 或 P2 区域使用该指令 （详细内容请参考 **“第 7 章 存储器管理单元 （MMU）”**）。发行该指令后，与 LDTLB 指令和对区域 P0、 U0、 P3 进行存取相关的指令，即必须在发行 BRAF、 BSRF、 JMP、 JSR、 RTS、 RTE 期间，至少有一条指令。

(3) 操作内容

```
LDTLB( )   /*LDTLB */
{
    TLB[MMUCR. URC] .ASID=PTEH & 0x000000FF;
    TLB[MMUCR. URC] .VPN=(PTEH & 0xFFFFFC00) >> 10;
    TLB[MMUCR. URC] .PPN=(PTEH & 0x1FFFFC00) >> 10;
    TLB[MMUCR. URC] .SZ=(PTEL & 0x00000080) >> 6 | (PTEL & 0x00000010) >> 4;
    TLB[MMUCR. URC] .SH=(PTEH & 0x00000002) >> 1;
    TLB[MMUCR. URC] .PR=(PTEH & 0x00000060) >> 5;
    TLB[MMUCR. URC] .WT=(PTEH & 0x00000001);
    TLB[MMUCR. URC] .C=(PTEH & 0x00000008) >> 3;
    TLB[MMUCR. URC] .D=(PTEH & 0x00000004) >> 2;
    TLB[MMUCR. URC] .V=(PTEH & 0x00000100) >> 8;

    PC += 2;
}
```

(4) 使用示例

```
MOV     @R0,R1          ;将页表入口高位加载到 R1
MOV     R1,@R2          ;将 R1 加载到 PTEH， R2 为 PTEH 的地址 (H'FF000000)
LDTLB                   ;将 PTEH,PTEL 寄存器加载到 TLB
```

(5) 有可能发生的异常

- 普通非法指令异常
- 槽非法指令异常

10.1.28 MAC.L Multiply and ACcumulate Long 算术运算指令

双精度乘法累加运算

格式	操作概略	操作码	执行状态	T 位
MAC.L @Rm+,@Rn+	带符号 (Rn)×(Rm)+MAC→MAC Rn+4→Rn, Rm+4→Rm	0000nnnnmmmm1111	5	—

(1) 说明

该指令进行带符号的 32 位操作数 （以通用寄存器 Rm 与 Rn 的内容为地址）的乘法运算，将 64 位结果与 MAC 寄存器的内容相加，并将其结果保存至 MAC 寄存器。每次读取各个操作数时，分别给 Rm+4，给 Rn+4。

S 位为 0 时，将 64 位结果保存至连接的 MACH、 MACL 寄存器。

S 位为 1 时，与 MAC 寄存器的加法运算在从 LSB 开始的第 48 个位变为饱和运算。饱和运算中，仅 MAC 寄存器的低 48 位有效，将结果的范围限制在 H'FFFF800000000000 （最小值）～ H'00007FFFFFFFFFFF （最大值）。

(2) 注意事项

无

(3) 操作内容

```
MACL(long m, long n)   /* MAC.L @Rm+,@Rn+ */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;
    long tempm,tempn,fnLmL;

    tempn = (long)Read_Long(R[n]);
    R[n] += 4;
    tempm = (long)Read_Long(R[m]);
    R[m] += 4;

    if((long)(tempn^tempm)<0) fnLmL = -1;
    else fnLmL = 0;
    if(tempn<0) tempn = 0-tempn;
    if(tempm<0) tempm = 0-tempm;

    temp1 = (unsigned long)tempn;
    temp2 = (unsigned long)tempm;

    RnL = temp1 & 0x0000FFFF;
    RnH = (temp1>>16) & 0x0000FFFF;
    RmL = temp2 & 0x0000FFFF;
    RmH = (temp2>>16)& 0x0000FFFF;
```

```
    temp0 = RmL * RnL;
    temp1 = RmH * RnL;
    temp2 = RmL * RnH;
    temp3 = RmH * RnH;

    Res2 = 0;

Res1 = temp1 + temp2;
if(Res1<temp1) Res2 += 0x00010000;

temp1 = (Res1<<16) & 0xFFFF0000;
Res0 = temp0 + temp1;
if(Res0<temp0) Res2++;

Res2 = Res2 + ((Res1>>16) & 0x0000FFFF) + temp3;

if(fnLmL<0){
    Res2 = ~Res2;
    if(Res0==0) Res2++;
    else Res0 = (~Res0)+1;
}
if(S==1){
    Res0 = MACL + Res0;
    if(MACL>Res0) Res2++;
    if(MACH&0x00008000);
    else Res2 += MACH | 0xFFFF0000;
          Res2 += MACH & 0x00007FFF;

    if(((long)Res2<0)&&(Res2<0xFFFF8000)){
        Res2 = 0xFFFF8000;
        Res0 = 0x00000000;
    }
    if(((long)Res2>0)&&(Res2>0x00007FFF)){
        Res2 = 0x00007FFF;
        Res0 = 0xFFFFFFFF;
    };

    MACH = (Res2 & 0x0000FFFF) | (MACH & 0xFFFF0000);
    MACL = Res0;
}
    else {
        Res0 = MACL + Res0;
        if(MACL>Res0) Res2++;
        Res2 += MACH;

        MACH = Res2;
```

```
            MACL = Res0;
        }
        PC += 2;
    }
```

(4) 使用示例

```
                    MOVA      TBLM,R0          ; 获取表地址
                MOV       R0,R1            ;
                 MOVA      TBLN,R0           ; 获取表地址
                CLRMAC                     ; 初始化 MAC 寄存器
                MAC.L     @R0+,@R1+        ;
                MAC.L     @R0+,@R1+        ;
                STS       MACL,R0          ; 在 R0 中得出结果
                ・・・・・・・
                .align    2                ;
    TBLM        .data.l   H'1234ABCD       ;
                .data.l   H'5678EF01       ;
    TBLN        .data.l   H'0123ABCD       ;
                .data.l   H'4567DEF0       ;
```

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.1.29 MAC.W Multiply and ACcumulate Word 算术运算指令

乘法累加运算

格式		操作概略	操作码	执行状态	T 位
MAC.W	@Rm+,@Rn+	带符号 (Rn)×(Rm)+MAC→MAC Rn+2→Rn, Rm+2→Rm	0100nnnnmmmm1111	4	—
MAC	@Rm+,@Rn+				

(1) 说明

该指令进行带符号的 16 位操作数 （以通用寄存器 Rm 与 Rn 的内容为地址）的乘法运算，给 MAC 寄存器的内容加上 32 位结果，并将结果保存至 MAC 寄存器。每次读取各个操作数时，分别给 Rm+2，给 Rn+2。

S 位为 0 时，为 16×16+64→64 位的乘法累加运算，将 64 位结果保存至连接的 MACH、 MACL 寄存器。

S 位为 1 时，为 16×16+32→32 位的乘法累加运算，与 MAC 寄存器的加法运算变为饱和运算。饱和运算时，仅 MACL 寄存器有效，将结果的范围限制在 H'80000000 （最小值）～ H'7FFFFFFF （最大值）。发生上溢时，将 MACH 寄存器的 LSB 置 1。结果向负方向上溢时，将 H'80000000 （最小值）保存至 MACL 寄存器；结果向正方向上溢时，将 H'7FFFFFFF （最大值）保存至 MACL 寄存器。

(2) 注意事项

S 位为 0 时，进行 16×16+64→64 位的乘法累加运算。

(3) 操作内容

```
MACW(long m, long n)   /* MAC.W @Rm+,@Rn+ */
{
    long tempm,tempn,dest,src,ans;
    unsigned long templ;
    tempn = (long)Read_Word(R[n]);
    R[n] += 2;
    tempm = (long)Read_Word(R[m]);
    R[m] += 2;
    templ=MACL;
    tempm = ((long)(short)tempn*(long)(short)tempm);
    if((long)MACL>=0) dest = 0;
    else dest = 1;
    if((long)tempm>=0) {
        src=0;
        tempn = 0;
    }
    else {
        src=1;
        tempn = 0xFFFFFFFF;
    }
    src += dest;
    MACL += tempm;
    if((long)MACL>=0) ans = 0;
```

```
    else ans = 1;
    ans += dest;
    if(S==1) {
        if(ans==1) {
            if(src==0) MACL = 0x7FFFFFFF;
            if(src==2) MACL = 0x80000000;
        }
    }
    else {
        MACH += tempn;
        if(templ>MACL) MACH += 1;

    }
    PC += 2;
}
```

(4) 使用示例

```
                MOVA    TBLM,R0         ; 获取表地址
        MOV     R0,R1           ;
        MOVA    TBLN,R0         ; 获取表地址
        CLRMAC                  ; 初始化 MAC 寄存器
        MAC.W   @R0+,@R1+       ;
        MAC.W   @R0+,@R1+       ;
        STS     MACL,R0         ; 在 R0 中得出结果
        ・・・・・・
        .align  2               ;
TBLM    .data.w H'1234          ;
        .data.w H'5678          ;
TBLN    .data.w H'0123          ;
        .data.w H'4567          ;
```

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.1.30 MOV MOVe data 数据传输指令

数据传输

格式		操作概略	操作码	执行状态	T 位
MOV	Rm,Rn	Rm→Rn	0110nnnnmmmm0011	1	—
MOV.B	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0000	1	—
MOV.W	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0001	1	—
MOV.L	Rm,@Rn	Rm→(Rn)	0010nnnnmmmm0010	1	—
MOV.B	@Rm,Rn	(Rm)→ 符号扩展 →Rn	0110nnnnmmmm0000	1	—
MOV.W	@Rm,Rn	(Rm)→ 符号扩展 →Rn	0110nnnnmmmm0001	1	—
MOV.L	@Rm,Rn	(Rm)→Rn	0110nnnnmmmm0010	1	—
MOV.B	Rm,@-Rn	Rn-1→Rn, Rm→(Rn)	0010nnnnmmmm0100	1	—
MOV.W	Rm,@-Rn	Rn-2→Rn, Rm→(Rn)	0010nnnnmmmm0101	1	—
MOV.L	Rm,@-Rn	Rn-4→Rn, Rm→(Rn)	0010nnnnmmmm0110	1	—
MOV.B	@Rm+,Rn	(Rm)→ 符号扩展 →Rn, Rm+1→Rm	0110nnnnmmmm0100	1	—
MOV.W	@Rm+,Rn	(Rm)→ 符号扩展 →Rn, Rm+2→Rm	0110nnnnmmmm0101	1	—
MOV.L	@Rm+,Rn	(Rm)→Rn, Rm+4→Rm	0110nnnnmmmm0110	1	—
MOV.B	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0100	1	—
MOV.W	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0101	1	—
MOV.L	Rm,@(R0,Rn)	Rm→(R0+Rn)	0000nnnnmmmm0110	1	—
MOV.B	@(R0,Rm),Rn	(R0+Rm)→ 符号扩展 →Rn	0000nnnnmmmm1100	1	—
MOV.W	@(R0,Rm),Rn	(R0+Rm)→ 符号扩展 →Rn	0000nnnnmmmm1101	1	—
MOV.L	@(R0,Rm),Rn	(R0+Rm)→Rn	0000nnnnmmmm1110	1	—

(1) 说明

将源操作数传送至目标。操作数为存储器时，可将传输的数据长度指定在字节 / 字 / 长字范围内。源操作数为存储器时，将已加载的数据符号扩展为长字后，将其保存至寄存器。

(2) 注意事项

无

(3) 操作内容

```
MOV(long m, long n)   /* MOV Rm,Rn */
{
    R[n] = R[m];
    PC += 2;
}

MOVBS(long m, long n) /* MOV.B Rm,@Rn */
{
     Write_Byte(R[n],R[m]);
     PC += 2;
}
```

```
MOVWS(long m, long n) /* MOV.W Rm,@Rn */
{
     Write_Word(R[n],R[m]);
     PC += 2;
}

MOVLS(long m, long n) /* MOV.L Rm,@Rn */
{
     Write_Long(R[n],R[m]);
     PC += 2;
}

MOVBL(long m, long n) /* MOV.B @Rm,Rn */
{
     R[n] = (long)Read_Byte(R[m]);
     if((R[n]&0x80)==0) R[n] &= 0x000000FF;
     else R[n] |= 0xFFFFFF00;
     PC += 2;
}

MOVWL(long m, long n) /* MOV.W @Rm,Rn */
{
     R[n] = (long)Read_Word(R[m]);
     if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
     else R[n] |= 0xFFFF0000;
     PC += 2;
}

MOVLL(long m, long n) /* MOV.L @Rm,Rn */
}
     R[n] = Read_Long(R[m]);
     PC += 2;
}

MOVBM(long m, long n) /* MOV.B Rm,@-Rn */
{
     Write_Byte(R[n]-1,R[m]);
     R[n] -= 1;
     PC += 2;
}

MOVWM(long m, long n) /* MOV.W Rm,@-Rn */
{
     Write_Word(R[n]-2,R[m]);
     R[n] -= 2;
```

```
     PC += 2;
}

MOVLM(long m, long n)   /* MOV.L Rm,@-Rn */
{
     Write_Long(R[n]-4,R[m]);
     R[n] -= 4;
     PC += 2;
}

MOVBP(long m, long n) /* MOV.B @Rm+,Rn */
{
     R[n] = (long)Read_Byte(R[m]);
     if((R[n]&0x80)==0) R[n] &= 0x000000FF;
     else R[n] |= 0xFFFFFF00;
     if(n != m) R[m] += 1;
     PC += 2;
}
MOVWP(long m, long n)    /* MOV.W @Rm+,Rn */
{
     R[n] = (long)Read_Word(R[m]);
     if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
     else R[n] |= 0xFFFF0000;
     if(n != m) R[m] += 2;
     PC += 2;
}
MOVLP(long m, long n)   /* MOV.L @Rm+,Rn */
{
     R[n] = Read_Long(R[m]);
     if(n != m) R[m] += 4;
     PC += 2;
}

MOVBS0(long m, long n) /* MOV.B Rm,@(R0,Rn) */
{
     Write_Byte(R[n]+R[0],R[m]);
     PC += 2;
}

MOVWS0(long m, long n) /* MOV.W Rm,@(R0,Rn) */
{
     Write_Word(R[n]+R[0],R[m]);
    PC += 2;
}

MOVLS0(long m, long n) /* MOV.L Rm,@(R0,Rn) */
```

```
{
     Write_Long(R[n]+R[0],R[m]);
     PC += 2;
}

MOVBL0(long m, long n) /* MOV.B @(R0,Rm),Rn */
{
    R[n] = (long)Read_Byte(R[m]+R[0]);
    if((R[n]&0x80)==0) R[n] &= 0x000000FF;
    else R[n] |= 0xFFFFFF00;
    PC += 2;
}

MOVWL0(long m, long n)   /* MOV.W @(R0,Rm),Rn */
{
    R[n] = (long)Read_Word(R[m]+R[0]);
    if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
    else R[n] |= 0xFFFF0000;
    PC += 2;
}

MOVLL0(long m, long n)   /* MOV.L @(R0,Rm),Rn */
{
    R[n] = Read_Long(R[m]+R[0]);
    PC += 2;
}
```

(4) 使用示例

```
MOV       R0,R1          ; 执行前 R0=H'FFFFFFFF,R1=H'00000000
                         ; 执行后 R1=H'FFFFFFFF
MOV.W     R0,@R1         ; 执行前 R0=H'FFFF7F80
                         ; 执行后 (R1)=H'7F80
MOV.B     @R0,R1         ; 执行前 (R0)=H'80,R1=H'00000000
                         ; 执行后 R1=H'FFFFFF80
MOV.W     R0,@- R1       ; 执行前 R0=H'AAAAAAAA,(R1)=H'FFFF7F80
                         ; 执行后 R1=H'FFFF7F7E,(R1)=H'AAAA
MOV.L     @R0+,R1        ; 执行前 R0=H'12345670
                         ; 执行后 R0=H'12345674,R1=(H'12345670)
MOV.B     R1,@(R0,R2     ; 执行前 R2=H'00000004,R0=H'10000000
          )              ; 执行后 (H'10000004)=R1
MOV.W     @(R0,R2),R     ; 执行前 R2=H'00000004,R0=H'10000000
          1              ; 执行后 R1=(H'10000004)
```

(5) 有可能发生的异常

可能会发生以下异常， MOV Rm,Rn 指令除外。

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误
- 初始页写入异常 （仅限向存储器写入的指令）

10.1.31 MOV MOVe constant value 数据传输指令

立即数的传输

格式		操作概略	操作码	执行状态	T 位
MOV	#imm,Rn	imm→ 符号扩展 →Rn	1110nnnniiiiiiii	1	—
MOV.W	@(disp*,PC),Rn	(disp 2+PC+4)→ 符号扩展 →Rn	1001nnnnddddddd	1	—
MOV.L	@(disp*,PC),Rn	(disp 4+PC&H'FFFFFFFC+4)→Rn	1101nnnnddddddd	1	—

【注】 * 在瑞萨的汇编程序中，将定标后（×1、×2、×4）的值设定为 disp。

(1) 说明

将符号扩展为长字后的立即数保存至通用寄存器 Rn。数据为字或长字时，从地址（PC+4+ 位移量 ×2）或（PC+4+ 位移量 × 4）的存储器位置保存数据。

因为数据为字时，零扩展后要给 8 位位移量乘以 2，所以与表的相对距离为到 PC+4+510 字节前的范围。PC 为该指令的指令地址。因为数据为长字时，零扩展后要给 8 位位移量乘以 4，所以与操作数的相对距离为到 PC+4+1020 字节前的范围。PC 为该指令的指令地址，将低 2 位修改为 B'00 的值用于地址计算。

(2) 注意事项

在延迟槽执行 PC 相对加载指令时，发生槽非法指令异常。

(3) 操作内容

```
MOVI(int i, int n)  /* MOV #imm,Rn */
{
      if((i&0x80)==0) R[n]=(0x000000FF & i);
      else R[n] = (0xFFFFFF00 | i);
      PC += 2;
}

MOVWI(d, n) /* MOV.W @(disp,PC),Rn */
{
      unsigned int disp;

      disp = (unsigned int)(0x000000FF & d);
      R[n] = (int)Read_Word(PC+4+(disp<<1));
      if((R[n]&0x8000)==0) R[n] &= 0x0000FFFF;
      else R[n] |= 0xFFFF0000;
      PC += 2;
}

      MOVLI(int d, int n)/* MOV.L @(disp,PC),Rn */
{
      unsigned int disp;

      disp = (unsigned int)(0x000000FF & (int)d);
      R[n] = Read_Long((PC&0xFFFFFFFC)+4+(disp<<2));
      PC += 2;
}
```

(4) 使用示例

地址

```
1000          MOV       #H'80,R1        ;R1=H'FFFFFF80
1002          MOV.WI    MM,R2           ;R2=H'FFFF9ABC  IMM 意为 (PC+4+H'08)
1004          ADD       #-1,R0;
1006          TST       R0,R0
1008          MOV.L     @(3*,PC),R3     ;R3=H'1234567
100A          BRA       NEXT            ; 延迟转移指令
100C          NOP
100E. IMM     .data.w   H'9ABC          ;
1010          .data.w   H'1234          ;
1012  NEXT    JMP       @R3             ;BRA 的转移目标
1014          CMP/EQ    #0,R0           ;
              .align    4               ;
1018          .data.l   H'12345678      ;
101C          .data.l   H'9ABCDEF0      ;
```

【注】 * 在瑞萨的汇编程序中，将定标后（×1、×2、×4）的值设定为 disp。

(5) 有可能发生的异常

只有在执行 PC 相对加载指令时，才可能会发生以下异常。

- 数据TLB多重命中异常
- 槽非法指令异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.1.32 MOV MOVe global data 数据传输指令

全局数据的传输

格式		操作概略	操作码	执行状态	T 位
MOV.B	@(disp*,GBR),R0	(disp+GBR)→ 符号扩展 →R0	11000100dddddddd	1	—
MOV.W	@(disp*,GBR),R0	(disp ×2+GBR)→ 符号扩展 →R0	11000101dddddddd	1	—
MOV.L	@(disp*,GBR),R0	(disp×4+GBR)→R0	11000110dddddddd	1	—
MOV.B	R0,@(disp*,GBR)	R0→(disp+GBR)	11000000dddddddd	1	—
MOV.W	R0,@(disp*,GBR)	R0→(disp×2+GBR)	11000001dddddddd	1	—
MOV.L	R0,@(disp*,GBR)	R0→(disp×4+GBR)	11000010dddddddd	1	—

【注】 * 在瑞萨的汇编程序中，将定标后 （× 1、 × 2、 × 4）的值设定为 disp。

(1) 说明

将源操作数传送至目标。可将数据长度指定在字节、字或长字范围内，寄存器固定为 R0。传输数据长度为字节时，仅将 8 位位移量进行零扩展，因此可指定高达 +255 字节的范围。长度为字时，将 8 位位移量进行零扩展后，再乘以 2，因此可指定高达 +510 字节的范围。长度为长字时，将 8 位位移量零扩展后再乘以 4，因此可指定高达 +1020 字节的范围。

源操作数为存储器时，将加载的数据符号扩展为长字后再保存至寄存器。

(2) 注意事项

加载时，目标寄存器固定为 R0。

(3) 操作内容

```
MOVBLG(int d)  /* MOV.B @(disp,GBR),R0 */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        R[0] = (int)Read_Byte(GBR+disp);
        if((R[0]&0x80)==0) R[0] &= 0x000000FF;
        else R[0] |= 0xFFFFFF00;
        PC += 2;
}

MOVWLG(int d)  /* MOV.W @(disp,GBR),R0 */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);

        R[0] = (int)Read_Word(GBR+(disp<<1));
        if((R[0]&0x8000)==0) R[0] &= 0x0000FFFF;
        else R[0] |= 0xFFFF0000;
        PC += 2;
}
```

```
MOVLLG(int d)  /* MOV.L @(disp,GBR),R0 */
{

        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        R[0] = Read_Long(GBR+(disp<<2));
        PC += 2;
}
MOVBSG(int d) /* MOV.B R0,@(disp,GBR) */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        Write_Byte(GBR+disp,R[0]);
        PC += 2;
}

MOVWSG(int d) /* MOV.W R0,@(disp,GBR) */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & d);
        Write_Word(GBR+(disp<<1),R[0]);
        PC += 2;
}

MOVLSG(int d) /* MOV.L R0,@(disp,GBR) */
{
        unsigned int disp;

        disp = (unsigned int)(0x000000FF & (long)d);
        Write_Long(GBR+(disp<<2),R[0]);
        PC += 2;
}
```

(4) 使用示例

```
MOV.L    @(2*,GBR),R0    ; 执行前 (GBR+8)=H'12345670
                         ; 执行后 R0=H'12345670
MOV.B    R0,@(1*,GBR)    ; 执行前 R0=H'FFFF7F80
                         ; 执行后 (GBR+1)=H'80
```

【注】 * 在瑞萨的汇编程序中，将定标后 （×1、×2、×4）的值设定为 disp。

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 槽非法指令异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误
- 初始页写入异常 （仅限向存储器写入的指令）

10.1.33 MOV MOVe structure data 数据传输指令

结构体数据传输

格式	操作概略	操作码	执行状态	T 位
MOV.B R0,@(disp*,Rn)	R0→(disp+Rn)	10000000nnnndddd	1	—
MOV.W R0,@(disp*,Rn)	R0→(disp×2+Rn)	10000001nnnndddd	1	—
MOV.L Rm,@(disp*,Rn)	Rm→(disp×4+Rn)	0001nnnnmmmmdddd	1	—
MOV.B @(disp*,Rm),R0	(disp+Rm)→ 符号扩展 →R0	10000100mmmmdddd	1	—
MOV.W @(disp*,Rm),R0	(disp ×2+Rm)→ 符号扩展 →R0	10000101mmmmdddd	1	—
MOV.L @(disp*,Rm),Rn	(disp×4+Rm)→Rn	0101nnnnmmmmdddd	1	—

【注】 * 在瑞萨的汇编程序中，将定标后 （×1、×2、×4）的值设定为 disp。

(1) 说明

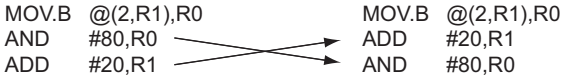
将源操作数传送至目标。适用于结构体、堆栈内的数据存取。可将数据长度指定在字节、字或长字范围内，字节或字时，寄存器固定为 R0。

数据长度为字节时， 4 位位移量只进行零扩展，因此可指定达 +15 字节的范围。长度为字时， 4 位位移量进行零扩展后再乘以 2，因此可指定达 +30 字节的范围。长度为长字时， 4 位位移量进行零扩展后再乘以 4，因此可指定达 +60 字节的范围。未达到存储器操作数时，需要使用上述 @(R0,Rn) 方式。

源操作数为存储器时，将加载的数据符号扩展为长字后再保存至寄存器。

(2) 注意事项

加载字节 / 字数据时，目标寄存器固定为 R0。因此，如果下一条指令试图参考 R0，则需要在执行完加载指令之前一直等待。可通过改变指令的顺序将其优化。



(3) 操作内容

```
MOVBS4(long d), long n   /* MOV.B R0,@(disp,Rn) */
{
     long disp;
     disp = (0x0000000F & (long)d);
     Write_Byte(R[n]+disp,R[0]);
     PC += 2;
}

MOVWS4(long d, long n)  /* MOV.W R0,@(disp,Rn) */
{
     long disp;

     disp = (0x0000000F & (long)d);
     Write_Word(R[n]+(disp<<1),R[0]);
     PC += 2;
}

MOVLS4(long m, long d, long n)/* MOV.L Rm,@(disp,Rn) */
{
     long disp;

     disp = (0x0000000F & (long)d);
     Write_Long(R[n]+(disp<<2),R[m]);
     PC += 2;
}

MOVBL4(long m, long d)  /* MOV.B @(disp,Rm),R0 */
{
     long disp;

     disp = (0x0000000F & (long)d);
     R[0] = Read_Byte(R[m]+disp);
     if((R[0]&0x80)==0) R[0] &= 0x000000FF;
     else R[0] |= 0xFFFFFF00;
     PC += 2;
}

MOVWL4(long m, long d) /* MOV.W @(disp,Rm),R0 */
{
   long disp;

   disp = (0x0000000F & (long)d);
   R[0] = Read_Word(R[m]+(disp<<1));
   if((R[0]&0x8000)==0) R[0] &= 0x0000FFFF;
```

```
    else R[0] |= 0xFFFF0000;
    PC += 2;
}

MOVLL4(long m, long d, long n) /* MOV.L @(disp,Rm),Rn */
{
    long disp;

    disp = (0x0000000F & (long)d);
    R[n] = Read_Long(R[m]+(disp<<2));
    PC += 2;
}
```

(4) 使用示例

```
MOV.L    @(2*,R0),R1     ;执行前 (R0+8)=H'12345670
                         ;执行后 R1=H'12345670
MOV.L    R0,@(H'F*,R1    ;执行前 R0=H'FFFF7F80
                         ;执行后 (R1+60)=H'FFFF7F80
```

【注】 * 在瑞萨的汇编程序中，将定标后（×1、×2、×4）的值设定为 disp。

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 槽非法指令异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误
- 初始页写入异常（仅限向存储器写入的指令）

10.1.34 MOVA MOVe effective Address 数据传输指令

有效地址的传输

格式		操作概略	操作码	执行状态	T 位
MOVA	@(disp*,PC),R0	Disp×4+PC&H'FFFFFFFC+4→R0	11000111dddddddd	1	—

【注】 * 在瑞萨的汇编程序中，将定标后 （× 1、×2、×4）的值设定为 disp。

(1) 说明

将源操作数的有效地址保存至通用寄存器 R0。 8 位位移量进行零扩展后再乘以 4。 PC 为该指令的指令地址，将低 2 位修改为 B'00 的值用于地址计算。

(2) 注意事项

在延迟槽中执行该指令时，发生槽非法指令异常。

(3) 操作内容

```
MOVA(int d)                /* MOVA @(disp,PC),R0 */
{
          unsigned int disp;

          disp = (unsigned int)(0x000000FF & d);
          R[0] = (PC&0xFFFFFFFC) + 4 + (disp<<2);
          PC += 2;
}
```

(4) 使用示例

```
      地址              .org          H'1006
      1006              MOVA          STR*,R0         ;STR 的地址 →R0
      1008              MOV.B         @R0,R1          ;R1="X"  ←PC 低 2 位修改后的位置
      100A              ADD           R4,R5
                        .align        4
      100C    STR:      .sdata        "XYZP12"
```

【注】 * 在瑞萨的汇编程序中，将定标后 （× 1、 × 2、 × 4）的值设定为 disp。

(5) 有可能发生的异常

- 槽非法指令异常

10.1.35 MOVCA.L MOVe with Cache block Allocation 数据传输指令

确保高速缓存块

格式	操作概略	操作码	执行状态	T 位
MOVCA.L R0,@Rn	（不获取高速缓存块） R0→(Rn)	0000nnnn11000011	1	—

(1) 说明

将通用寄存器 R0 保存至有效地址 Rn 所示的存储器。该指令与其它保存指令在以下方面有所不同。

在被存取的存储器选择了回写方式，且发生高速缓存未命中时，虽然确保了高速缓存块，但是不读取块，将 R0 的数据写入该高速缓存块。其它高速缓存块的内容还未定义。

(2) 注意事项

无。

(3) 操作内容

```
MOVCAL(int n)      /*MOVCA.L  R0,@Rn */
{
     if((is_write_back_memory(R[n]))
            && (look_up_in_operand_cache(R[n]) == MISS))
                     allocate_operand_cache_block(R[n]);
     Write_Long(R[n],R[0]);
     PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

10.1.36 MOVCO MOVe COnditional 数据传输指令

完成原子 Read-Modify-Write 时，带保存条件的保存

格式	操作概略	操作码	执行状态	T 位
MOVCO.L R0,@Rn	LDST→T if(T==1) R0→(Rn) 0→LDST	0000nnnn01110011	1	LDST

(1) 说明

结合 MOVLI 指令与 MOVCO 指令，实现单处理器上的原子 Read-Modify-Write。

该指令将 LDST 标志的值复制至 T 位，T 为 1 时，将 R0 的值保存至地址 Rm，T 为 0 时，不保存。最后将 LDST 标志清 0。发生中断或异常时， LDST 标志清 0，因此在 MOVLI 指令与 MOVCO 指令之间未发生中断或异常时，保存 MOVCO 指令。

(2) 注意事项

无

(3) 操作内容

```
MOVCO(long n) /* MOVCO Rn,@Rn */
{
     T = LDST;
     if(T==1)
Write_Long(R[n],R[0]);
     LDST = 02;
     PC += 2
}
```

(4) 使用示例

```
Retry:  MOVLI.L   @Rn,R0      ; 原子递增
        ADD       #1,R0
        MOVCO.L   R0,@Rn
        BF        Retry       ;MOVLI、 MOVCO 之间发生中断 / 异常时，重新执行
        NOP
```

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

10.1.37 MOVLI MOVe LInked 数据传输指令

开始加载原子 Read-Modify-Write

格式	操作概略	操作码	执行状态	T 位
MOVLI.L @Rm,R0	1→LDST, (Rm)→R0 但是，发生中断 / 异常时 0→LDST	0000nnnn01100011	1	LDST

(1) 说明

结合 MOVLI 指令与 MOVCO 指令，实现单处理器上原子 Read-Modify-Write。

该指令将 LDST 标志置 1，将 Rm 指向的 4 字节数据读取到 R0。发生中断或异常时， LDST 清 0。

被 MOVLI 指令置 1 的 LDST 通过中断或异常不被清 0 而执行 MOVCO 指令时， MOVCO 指令执行保存并将 T 位置 1。 LDST 清 0 时如果执行了 MOVCO 指令，则不执行保存，而将 T 位置 0。

(2) 注意事项

无

(3) 操作内容

```
MOVLINK(long m) /* MOVLI Rm,@Rn */
{
     LDST = 1;
     R[0] = Read_Long(R[m]);
     PC += 2
}
```

(4) 使用示例

请参考 MOVCO 指令。

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.1.38 MOVT MOVe Tbit 数据传输指令

T 位的传送

格式	操作概略	操作码	执行状态	T 位
MOVT Rn	T→Rn	0000nnnn00101001	1	—

(1) 说明

将 T 位保存至通用寄存器 Rn。 T=1 时， Rn=1 ； T=0 时 Rn=0。

(2) 注意事项

无

(3) 操作内容

```
MOVT(long n)                /* MOVT Rn */
{
        R[n] = (0x00000001 & SR);
        PC += 2;
}
```

(4) 使用示例

```
XOR             R2,R2;              ;R2=0
CMP/PZ          R2                  ;T=1
MOVT            R0;                 ;R0=1
CLRT                                ;T=0
MOVT            R1                  ;R1=0
```

10.1.39 MOVUA MOVe UnAligned 数据传输指令

非边界调整数据传送

格式	操作概略	操作码	执行状态	T 位
MOVUA.L @Rm,R0	(Rm)→R0 加载非边界调整数据	0100nnnn10101001	2	—
MOVUA.L @Rm+,R0	(Rm)→R0、 Rm+4→Rm 加载非边界调整数据	0100nnnn11101001	2	—

(1) 说明

Rm 的内容作为有效地址，将存储器中的长字数据加载至 R0。不仅可从长字边界的地址 （4n）加载长字数据，还可从不是长字边界的地址 （4n+1、 4n+2、 4n+3）加载长字数据。即使存取不是长字边界的地址（4n+1、 4n+2、 4n+3）也不发生数据地址错误异常。

(2) 注意事项

无

(3) 操作内容

```
MOVUAL(int m) /* MOVUA.L Rm,R0*/
{
     Read_Unaligned_Long(R0,R[m]);
        PC += 2;
}
MOVUALP(int m) /* MOVUA.L Rm+,R0*/
{
       Read_Unaligned_Long(R0,R[m]);
       if(m != 0) R[m] += 4;
       PC += 2;
}
```

(4) 使用示例

```
MOVUA.L     @R1,R0      ; 执行前 R1=H'00001001、 R0=H'00000000
                        ; 执行后 R0=(H'00001001)
MOVUA.L     @R1+,R0     ; 执行前 R1=H'00001007、 R0=H'00000000
                        ; 执行后 R1=H'0000100B、 R0=(H'00001007)
; 源操作数为 @R0 的特殊情况
MOVUA.L     @R0,R0      ; 执行前 R0=H'00001001
                        ; 执行后 R0=(H'00001001)
MOVUA.L     @R0+,R0     ; 执行前 R0=H'00001001
                        ; 执行后 R0=(H'00001001)
```

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误异常 （用户模式下已存取特权区域时）

10.1.40 MUL.L MULtiply Long 算术运算指令

双精度乘法运算

格式	操作概略	操作码	执行状态	T 位
MUL.L Rm,Rn	Rn×Rm→MACL	0000nnnnmmmm0111	2	—

(1) 说明

对通用寄存器 Rn 的内容与 Rm 进行 32 位乘法运算，将结果的低 32 位保存至 MACL 寄存器。 MACH 的内容不变。

(2) 注意事项

无

(3) 操作内容

```
MULL(long m, long n) /* MUL.L Rm,Rn */
{
     MACL = R[n]*R[m];
     PC += 2;
}
```

(4) 使用示例

```
MUL.L     R0,R1      ; 执行前  R0=H'FFFFFFFE,R1=H'00005555
                     ; 执行后  MACL=H'FFFF5556
STS       MACL,R0    ; 获取运算结果
```

10.1.41 MULS.W MULtiply as Signed Word 算术运算指令

带符号的乘法运算

格式		操作概略	操作码	执行状态	T 位
MULS.W	Rm,Rn	带符号 Rn ×Rm→MACL	0010nnnnmmmm1111	1	—

(1) 说明

对通用寄存器 Rn 的内容与 Rm 进行 16 位乘法运算，将 32 位结果保存在 MACL 寄存器。以带符号算术运算进行运算。 MACH 的内容不变。

(2) 注意事项

无

(3) 操作内容

```
MULS(long m, long n) /* MULS Rm,Rn */
{
     MACL = ((long)(short)R[n]*(long)(short)R[m]);
     PC += 2;
}
```

(4) 使用示例

```
          MULS.W      R0,R1       ;执行前 R0=H'FFFFFFFE,R1=H'00005555
                                  ;执行后 MACL=H'FFFF5556
          STS         MACL,R0     ;获取运算结果
```

10.1.42 MULU.W MULtiply as Unsigned Word 算术运算指令

无符号乘法运算

格式	操作概略	操作码	执行状态	T 位
MULU.W Rm,Rn	无符号 Rn ×Rm→MACL	0010nnnnmmmm1110	1	—

(1) 说明

对通用寄存器 Rn 的内容与 Rm 进行 16 位乘法运算，将 32 位结果保存在 MACL 寄存器。以无符号算术运算进行运算。 MACH 的内容不变。

(2) 注意事项

无

(3) 操作内容

```
MULU(long m, long n) /* MULU Rm,Rn */
{
     MACL = ((unsigned long)(unsigned short)R[n]*
     (unsigned long)(unsigned short)R[m];
       PC += 2;
}
```

(4) 使用示例

```
MULU.W      R0,R1       ; 执行前 R0=H'00000002,R1=H'FFFFAAAA
                        ; 执行后 MACL=H'00015554
STS         MACL,R0     ; 获取运算结果
```

10.1.43 NEG NEGate 算术运算指令

符号取反

格式	操作概略	操作码	执行状态	T 位
NEG Rm,Rn	0-Rm→Rn	0110nnnnmmmm1011	1	—

(1) 说明

获取通用寄存器 Rm 的内容的 2 的补码，将结果保存在 Rn。即从 0 减去 Rm，将结果保存在 Rn。

(2) 注意事项

无

(3) 操作内容

```
NEG(long m, long n) /* NEG Rm,Rn */
{
     R[n] = 0-R[m];
     PC += 2;
}
```

(4) 使用示例

```
NEG          R0,R1       ;执行前 R0=H'00000001
                         ;执行后 R1=H'FFFFFFFF
```

10.1.44 NEGC NEGate with Carry 算术运算指令

带借位的符号取反

格式	操作概略	操作码	执行状态	T 位
NEGC Rm,Rn	0-Rm-T→Rn, 借位 →T	0110nnnnmmmm1010	1	借位

(1) 说明

从 0 减去通用寄存器 Rm 的内容与 T 位，并将结果保存在 Rn。根据运算结果，在 T 位反映借位。对超过 32 位的值的符号进行取反时使用该指令。

(2) 注意事项

无

(3) 操作内容

```
NEGC(long m, long n) /* NEGC Rm,Rn */
{
      unsigned long temp;

      temp = 0-R[m];
      R[n] = temp-T;
      if(0<temp) T = 1;
      else T = 0;
      if(temp<R[n]) T = 1;
      PC += 2;
}
```

(4) 使用示例

```
CLRT                    ;将 R0:R1(64 位 ) 的符号取反
NEGC        R1,R1       ;执行前 R1=H'00000001,T=0
                        ;执行后 R1=H'FFFFFFFF,T=1
NEGC        R0,R0       ;执行前 R0=H'00000000,T=1
                        ;执行后 R0=H'FFFFFFFF,T=1
```

10.1.45 NOP No OPeration 系统控制指令

无操作

格式	操作概略	操作码	执行状态	T 位
NOP	无操作	0000000000001001	1	—

(1) 说明

仅进行 PC 的递增，然后转移到下一条指令的执行。

(2) 注意事项

无

(3) 操作内容

```
NOP(  ) /* NOP */
{
     PC += 2;
}
```

(4) 使用示例

```
NOP             ;经过 1 个执行状态的时间。
```

10.1.46 NOT NOT-logical complement 逻辑运算指令

位取反

格式	操作概略	操作码	执行状态	T 位
NOT Rm,Rn	～ Rm→Rn	0110nnnnmmmm0111	1	—

(1) 说明

获取通用寄存器 Rm 的内容的 1 的补码，将结果保存在 Rn。取反 Rm 的位并保存在 Rn。

(2) 注意事项

无

(3) 操作内容

```
NOT(long m, long n) /* NOT Rm,Rn */
{
        R[n] =  R[m];
        PC += 2;
}
```

(4) 使用示例

```
        NOT         R0,R1       ; 执行前 R0=H'AAAAAAAA
                                ; 执行后 R1=H'55555555
```

10.1.47 OCBI Operand Cache Block Invalidate 数据传输指令

高速缓存块的无效化

格式	操作概略	操作码	执行状态	T 位
OCBI @Rn	使操作数高速缓存区无效	0000nnnn10010011	1	—

(1) 说明

使用有效地址 Rn 所示的内容来存取数据。高速缓存命中时，使对应的高速缓存区无效 （Vbit=0）。此时，即使在回写模式下，或在有未写入信息 （Ubit=1）时，也不回写。高速缓存未命中时或对非高速缓存区存取时，不运行。

(2) 注意事项

无

(3) 操作内容

```
OCBI(int n)        /* OCBI @Rn */
{
  invalidate_operand_cache_block(R[n]);
  PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

请注意，即使 OCBI 不运行时，也会发生上述异常。

10.1.48 OCBP Operand Cache Block Purge 数据传输指令

高速缓存块的清除

格式	操作概略	操作码	执行状态	T 位
OCBP @Rn	回写操作数高速缓存块，并使其无效	0000nnnn10100011	1	—

(1) 说明

使用有效地址 Rn 所示内容来存取数据。高速缓存命中且有未写入信息 （Ubit=1）时，将对应的高速缓存块回写至外部存储器，然后使该块无效 （Vbit=0）。此时，无未写入信息 （Ubit=0）时，只使该块无效。高速缓存未命中时，或对非高速缓存区存取时，不运行。

(2) 注意事项

无

(3) 操作内容

```
OCBP(int n)        /* OCBP @Rn */
{
  if(is_dirty_block(R[n]))  write_back(R[n])
  invalidate_operand_cache_block(R[n]);
  PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

请注意，即使 OCBP 不运行，也会发生上述异常。

10.1.49 OCBWB Operand Cache Block Write Back 数据传输指令

高速缓存块的回写

格式	操作概略	操作码	执行状态	T 位
OCBWB @Rn	回写操作数高速缓存块	0000nnnn10110011	1	—

(1) 说明

使用有效地址 Rn 所示的内容来存取数据。高速缓存命中且有未写入信息 （Ubit=1）时，将对应的高速缓存区回写至外部存储器，然后清除该块 （Ubit=0）。其它情况下，高速缓存未命中或已清除时，对非高速缓存区存取时，不运行。

(2) 注意事项

无

(3) 操作内容

```
OCBWB(int n)       /* OCBWB @Rn */
     {
       if(is_dirty_block(R[n]))  write_back(R[n]);
       PC += 2;
     }
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

请注意，即使 OCBWB 不运行，也会发生上述异常。

10.1.50 OR OR logical 逻辑运算指令

逻辑 “或” 运算

格式	操作概略	操作码	执行状态	T 位
OR Rm,Rn	Rn ∣ Rm → Rn	0010nnnnmmmm1011	1	—
OR #imm,R0	R0 ∣ imm → R0	11001011iiiiiiii	1	—
OR.B #imm,@(R0,GBR)	(R0+GBR) ∣ imm → (R0+GBR)	11001111iiiiiiii	3	—

(1) 说明

获取通用寄存器 Rn 的内容与 Rm 的逻辑 “或”，将结果保存在 Rn。

可对通用寄存器 R0 与零扩展后的 8 位立即数进行逻辑 " 或 "，或可用带变址的 GBR 间接寻址方式对 8 位存储器与 8 位立即数进行逻辑 " 或 "。

(2) 注意事项

无

(3) 操作内容

```
OR(long m, long n) /* OR Rm,Rn */
{
        R[n] |= R[m];
        PC += 2;
}

ORI(long i)  /* OR #imm,R0 */
{
        R[0] |= (0x000000FF & (long)i);
        PC += 2;
}

ORM(long i) /* OR.B #imm,@(R0,GBR) */
{
       long temp;

       temp = (long)Read_Byte(GBR+R[0]);
       temp |= (0x000000FF & (long)i);
       Write_Byte(GBR+R[0],temp);
       PC += 2;
}
```

(4) 使用示例

```
OR          R0,R1               ; 执行前 R0=H'AAAA5555,R1=H'55550000
                                ; 执行后 R1=H'FFFF5555
OR          #H'F0,R0            ; 执行前 R0=H'00000008
                                ; 执行后 R0=H'000000F8
OR.B        #H'50,@(R0,GBR)     ; 执行前 (R0+GBR)=H'A5
                                ; 执行后 (R0+GBR)=H'F5
```

(5) 有可能发生的异常

仅执行 OR.B 指令时，可能会发生以下异常。

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

异常处理中，将通过该指令进行的数据存取作为字节加载、字节保存来处理。

10.1.51 PREF PREFetch data to cache 数据传输指令

预取数据高速缓存

格式	操作概略	操作码	执行状态	T 位
PREF @Rn	(Rn)→ 操作数高速缓存	0000nnnn10000011	1	—

(1) 说明

将从 32 字节边界开始的 32 字节的数据块读取到操作数高速缓存中。将 Rn 所指定地址的低 5 位屏蔽为 0。

不会因为该指令发生数据地址错误。也不发生 MMU 异常，数据 TLB 多重命中异常除外。符合这些异常条件时，作为 NOP （无操作）指令来处理。

(2) 注意事项

无

(3) 操作内容

```
PREF(int n) /* PREF @Rn */
{
        PC += 2;
}
```

(4) 使用示例

```
                MOV.L  SOFT_PF,R1    ;R1 的地址为 SOFT_PF
                PREF   @R1           ; 将 SOFT_PF 的数据加载到内部高速缓存

                   .align            32
SOFT_PF:           .data.l           H'12345678
                   .data.l           H'9ABCDEF0
                   .data.l           H'AAAA5555
                   .data.l           H'5555AAAA
                   .data.l           H'11111111
                   .data.l           H'22222222
                   .data.l           H'33333333
                   .data.l           H'44444444
```

(5) 有可能发生的异常

- 数据TLB多重命中异常

10.1.52 PREFI PREFetch Instruction cache block 数据传输指令

预取指令高速缓存块

格式	操作概略	操作码	执行状态	T 位
PREFI @Rn	将 32 字节的指令读取至指令高速缓存	0000nnnn11010011	10	—

(1) 说明

将从 32 字节边界开始的 32 字节的指令块读取至指令高速缓存。 Rn 所指定地址的低 5 位视为 0。

不会因为该指令发生数据地址错误及 MMU 异常。符合这些异常条件时，作为 NOP （无操作）指令来处理。

想预取的地址未命中 UTLB 时或违反了保护时，则作为 NOP （无操作）指令来处理，不发生 TLB 异常。

(2) 注意事项

无

(3) 操作内容

```
PREFI(int n) /* PREFI @Rn */
{
      prefetch_instruction_cache_block(R[n]);
      PC += 2;
}
```

(4) 使用示例

```
            MOVA        WakeUp,R0     ;WakeUp 地址 →0
            PREFI       @R0           ; 预取 SLEEP 指令解除后的指令
            SLEEP
WakeUp:
            NOP
```

发行 SLEEP 指令前，从 SLEEP 状态恢复时，将执行的指令预取到高速缓存中。

(5) 有可能发生的异常

- 槽非法指令异常

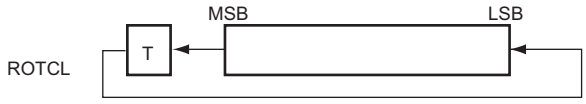
10.1.53 ROTCL ROTate with Carry Left 移位指令

带 T 位
向左旋转 1 位

格式	操作概略	操作码	执行状态	T 位
ROTCL Rn	T←Rn←T	0100nnnn00100100	1	MSB

(1) 说明

将通用寄存器 Rn 的内容向左旋转 1 位 （包括 T 位），将结果保存在 Rn。旋转后将超出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
ROTCL(long n) /* ROTCL Rn */
{
        long temp;

        if((R[n]&0x80000000)==0) temp = 0;
        else temp = 1;
        R[n] <<= 1;
        if(T==1) R[n] |= 0x00000001;
        else R[n] &= 0xFFFFFFFE;
        if(temp==1) T = 1;
        else T = 0;
        PC += 2;
}
```

(4) 使用示例

```
ROTCL   R0      ;执行前 R0=H'80000000,T=0
                ;执行后 R0=H'00000000,T=1
```

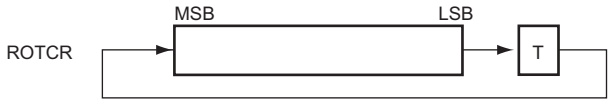
10.1.54 ROTCR ROTate with Carry Right 移位指令

带 T 位
向右旋转 1 位

格式		操作概略	操作码	执行状态	T 位
ROTCR	Rn	T→Rn→T	0100nnnn00100101	1	LSB

(1) 说明

将通用寄存器 Rn 的内容向右旋转 1 位 （包括 T 位），将结果保存在 Rn。旋转后将超出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
ROTCR(long n) /* ROTCR Rn */
{
        long temp;

        if((R[n]&0x00000001)==0) temp = 0;
        else temp = 1;
        R[n] >>= 1;
        if(T==1) R[n] |= 0x80000000;
        else R[n] &= 0x7FFFFFFF;
        if(temp==1) T = 1;
        else T = 0;
        PC += 2;
}
```

(4) 使用示例

```
        ROTCR    R0      ;执行前 R0=H'00000001,T=1
                         ;执行后 R0=H'80000000,T=1
```

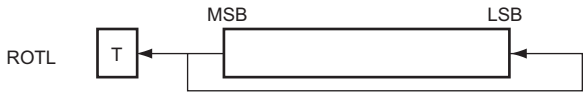
10.1.55 ROTL ROTate Left 移位指令

向左旋转 1 位

格式	操作概略	操作码	执行状态	T 位
ROTL Rn	T←Rn←MSB	0100nnnn00000100	1	MSB

(1) 说明

将通用寄存器 Rn 的内容向左旋转 1 位，将结果保存在 Rn。旋转后将超出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
ROTL(long n)  /* ROTL Rn */
{
        if((R[n]&0x80000000)==0) T = 0;
        else T = 1;
        R[n] <<= 1;
        if(T==1) R[n] |= 0x00000001;
        else R[n] &= 0xFFFFFFFE;
        PC += 2;
}
```

(4) 使用示例

```
ROTCL    R0      ;执行前 R0=H'80000000,T=0
                 ;执行后 R0=H'00000001,T=1
```

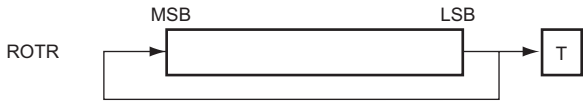
10.1.56 ROTR ROTate Right 移位指令

向右旋转 1 位

格式	操作概略	操作码	执行状态	T 位
ROTR Rn	LSB→Rn→T	0100nnnn00000101	1	LSB

(1) 说明

将通用寄存器 Rn 的内容向右旋转 1 位，将结果保存在 Rn。旋转后将超出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
ROTR(long n) /* ROTR Rn */
{
      if((R[n]&0x00000001)==0) T = 0;
      else T = 1;
      R[n]>>=1;
      if(T==1) R[n] |= 0x80000000;
      else R[n] &= 0x7FFFFFFF;
      PC += 2;
}
```

(4) 使用示例

```
ROTR R0          ;执行前 R0=H'00000001,T=0
                 ;执行后 R0=H'80000000,T=1
```

10.1.57 RTE ReTurn from Exception 系统控制指令

从异常处理返回 （特权指令）
延迟转移指令

格式	操作概略	操作码	执行状态	T 位
RTE	SSR→SR, SPC→PC	0000000000101011	4	—

(1) 说明

从异常、中断处理程序返回。从 SPC 与 SSR 恢复 PC 与 SR 值。从恢复的 PC 值所指定的地址继续执行程序。因为 RTE 指令为特权指令，所以只可在特权模式下使用。如果在用户模式下使用，则会发生非法指令异常。

(2) 注意事项

因为该指令是延迟转移指令，所以先执行 RTE 指令的下一条指令，然后执行转移目标指令。

该指令与其下一条指令之间，不接受中断。不通过该指令延迟槽内的指令发生异常。下一条指令为转移指令时，将其识别为槽非法指令。将该指令分配至紧接着延迟转移指令的延迟槽时，识别为槽非法指令。通过 RTE 延迟槽中的指令来存取的 SR 的内容为通过 RTE 从 SSR 恢复的值。但是，在执行 RTE 之前已定义的 SR、MD 值用于取 RTE 延迟槽内的指令。

(3) 操作内容

```
RTE( ) /* RTE */
{
        unsigned int temp;
        temp = PC;
        SR = SSR;
        PC = SPC;
        Delay_Slot(temp+2);
}
```

(4) 使用示例

```
RTE             ; 返回至源程序。
ADD #8,R14      ; 在转移之前执行。
```

【注】 在延迟转移中，转移操作是在槽指令执行后发生的；指令的执行 （寄存器的更新等），实际上是按照延迟转移指令 → 延迟槽指令的顺序进行的。例如，即使在延迟槽变更转移目标地址所保存的寄存器，变更前的寄存器内容还是转移目标地址。

(5) 有可能发生的异常

- 槽非法指令异常
- 普通非法指令异常

10.1.58 RTS ReTurn from Subroutine 转移指令

从子程序过程返回 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
RTS	PR→PC	0000000000001011	1	—

(1) 说明

从子程序过程返回。即，从 PR 恢复 PC，从恢复的 PC 所示的地址继续执行处理。通过该指令，可从被 BSR 及 JSR 指令调用的子程序过程返回至调用源。

(2) 注意事项

因为该指令是延迟转移指令，所以先执行该指令的下一条指令，然后执行转移目标指令。
该指令与其下一条指令之间，不接受中断。下一条指令为转移指令时，将其识别为槽非法指令。
必须在执行 RTS 指令之前执行恢复 PR 的指令。该恢复指令不能为 RTS 延迟槽。

(3) 操作内容

```
RTS( ) /* RTS */
{
        unsigned int temp;

        temp = PC;
        PC = PR;
        Delay_Slot(temp+2);
}
```

(4) 使用示例

```
                MOV.L       TABLE,R3    ;R3=TRGET 地址
                JSR         @R3         ; 转移到 TRGET。
                NOP                     ; 转移前，先执行 NOP。
                ADD         R0,R1       ;← 从子程序过程返回的目标 (PR 内容 )
                ・・・・・・
TABLE:          .data.l     TRGET       ; 跳转表
                ・・・・・・
TRGET:          MOV         R1,R0       ;← 过程的入口
                RTS                     ;PR 内容 →PC
                MOV         #12,R0      ; 转移前，先执行 MOV。
```

(5) 有可能发生的异常

- 槽非法指令异常

10.1.59 SETS SET Sbit 系统控制指令

S 位置位

格式	操作概略	操作码	执行状态	T 位
SETS	1→S	0000000001011000	1	—

(1) 说明

将 S 位置 1。

(2) 注意事项

无

(3) 操作内容

```
SETS( ) /* SETS */
{
        S=1;
        PC += 2;
}
```

(4) 使用示例

```
        SETS        ;执行前 S=0
                    ;执行后 S=1
```

10.1.60 SETT SET Tbit 系统控制指令

T 位置位

格式	操作概略	操作码	执行状态	T 位
SETT	1→T	0000000000011000	1	1

(1) 说明

置位 T 位。

(2) 注意事项

无

(3) 操作内容

```
SETT( )  /* SETT */
{
        T = 1;
        PC += 2;
}
```

(4) 使用示例

```
        SETT        ;执行前 T=0
                    ;执行后 T=1
```

10.1.61 SHAD SHift Arithemetic Dynamically 移位指令

动态算术移位

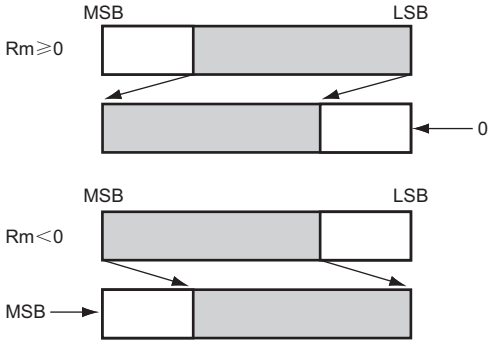
格式	操作概略	操作码	执行状态	T 位
SHAD Rm, Rn	Rm ≥ 0 时、 Rn<<Rm→Rn Rm < 0 时、 Rn>>Rm→[MSB→Rn]	0100nnnnmmmm1100	1	—

(1) 说明

将通用寄存器 Rn 的内容进行算术移位。通用寄存器 Rm 指定移位方向与移位的位数。

Rm 寄存器值为正时左移，为负时右移。右移时在高位追加 MSB。

由 Rm 寄存器的低 5 位 （bit4 ～ 0）来指定移位的位数。值为负 （MSB=1）时，以 2 的补码表示 Rm 寄器。左移的移位量为 0 ～ 31，右移的移位量为 1 ～ 32。



(2) 注意事项

无

(3) 操作内容

```
SHAD(int m,n) /*SHAD Rm,Rn */
{
        int sgn=R[m] & 0x80000000;
        if(sgn==0)
                R[n] <<= (R[m] & 0x1F);
        else if((R[m] & 0x1F) == 0) {
                if((R[n] & 0x80000000) == 0)
                    R[n] = 0;
                else
                    R[n] = 0xFFFFFFFF;
           }
        else  R[n] = (long)R[n] >> ((~R[m] & 0x1F)+1);
        PC += 2;
}
```

(4) 使用示例

```
SHAD        R1,R2       ; 执行前 R1=H'FFFFFFEC,R2=H'80180000
                        ; 执行后 R1=H'FFFFFFEC,R2=H'FFFFF801
SHAD        R3,R4       ; 执行前 R3=H'00000014,R4=H'FFFFF801
                        ; 执行后 R3=H'00000014,R4=H'80100000
```

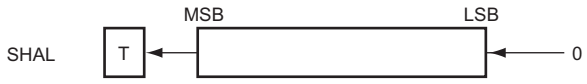
10.1.62 SHAL SHift Arithmetic Left 移位指令

算术左移 1 位

格式	操作概略	操作码	执行状态	T 位
SHAL Rn	T←Rn←0	0100nnnn00100000	1	MSB

(1) 说明

将通用寄存器 Rn 的内容算术左移 1 位，将结果保存在 Rn。移位后将移出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
SHAL(long n) /* SHAL Rn (Same as SHLL) */
{
        if((R[n]&0x80000000)==0) T = 0;
        else T = 1;
        R[n] <<= 1;
        PC += 2;
}
```

(4) 使用示例

```
SHAL R0         ; 执行前 R0=H'80000001,T=0
                ; 执行后 R0=H'00000002,T=1
```

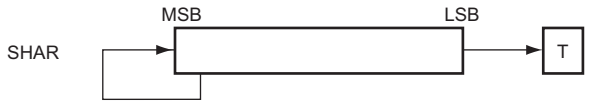
10.1.63 SHAR SHift Arithmetic Right 移位指令

算术右移 1 位

格式	操作概略	操作码	执行状态	T 位
SHAR Rn	MSB→Rn→T	0100nnnn00100001	1	LSB

(1) 说明

将通用寄存器 Rn 的内容算术右移 1 位，将结果保存在 Rn。移位后将移出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
SHAR(long n) /* SHAR Rn */
{
        long temp;

        if((R[n]&0x00000001)==0) T = 0;
        else T = 1;
        if((R[n]&0x80000000)==0) temp = 0;
        else temp = 1;
        R[n] >>= 1;
        if(temp==1) R[n] |= 0x80000000;
        else R[n] &= 0x7FFFFFFF;
        PC += 2;
}
```

(4) 使用示例

```
SHAR R0         ;执行前 R0=H'80000001,T=0
                ;执行后 R0=H'C0000000,T=1
```

10.1.64 SHLD SHift Logical Dynamically 移位指令

动态逻辑
移位

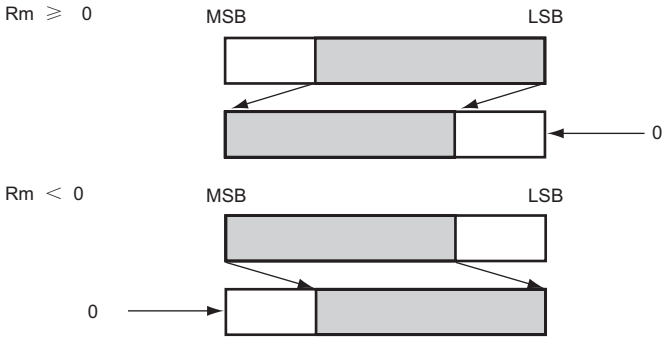
格式	操作概略	操作码	执行状态	T 位
SHLD Rm, Rn	Rm ≥ 0 时、 Rn<<Rm→Rn Rm < 0 时、 Rn>>Rm→[0→Rn]	0100nnnnmmmm1101	1	—

(1) 说明

将通用寄存器 Rn 的内容进行逻辑移位。通用寄存器 Rm 指定移位方向与移位的位数。

Rm 寄存器值为正时左移，为负时右移。右移时，在高位追加 0。

由 Rm 寄存器的低 5 位 （bit4 ～ 0）指定移位的位数。值为负 （MSB=1）时，以 2 的补码表示 Rm 寄存器。左移的移位量为 0 ～ 31，右移的移位量为 1 ～ 32。



(2) 注意事项

无

(3) 操作内容

```
SHLD(int m,n)/*SHLD Rm,Rn */
{
     int sgn = R[m] & 0x80000000;
     if(sgn == 0)
               R[n] <<= (R[m] & 0x1F);
     else if((R[m] & 0x1F) == 0)
               R[n] = 0;
     else
               R[n] = (unsigned)R[n] >> ((~R[m] & 0x1F) + 1);
     PC += 2;
}
```

(4) 使用示例

```
          SHLD          R1, R2     ; 执行前R1=H'FFFFFFEC, R2=H'80180000
                                   ; 执行后R1=H'FFFFFFEC, R2=H'00000801
          SHLD          R3, R4     ; 执行前R3=H'00000014, R4=H'FFFFF801
                                   ; 执行后R3=H'00000014, R4=H'80100000
```

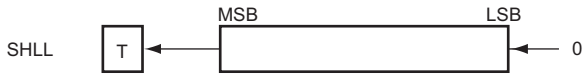
10.1.65 SHLL SHift Logical Left 移位指令

逻辑左移 1 位

格式	操作概略	操作码	执行状态	T 位
SHLL Rn	T←Rn←0	0100nnnn00000000	1	MSB

(1) 说明

将通用寄存器 Rn 的内容逻辑左移 1 位，将结果保存在 Rn。移位后将移出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
SHLL(long n) /* SHLL Rn  (Same as SHAL) */
{
    if((R[n]&0x80000000)==0) T = 0;
    else T = 1;
    R[n] <<= 1;
    PC += 2;
}
```

(4) 使用示例

```
SHLL R0        ;执行前    R0=H'80000001,T=0
               ;执行后    R0=H'00000002,T=1
```

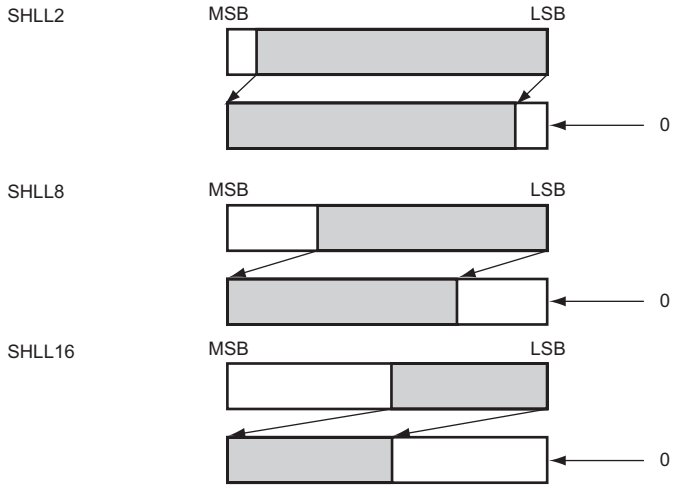
10.1.66 SHLLn n bits SHift Logical Left 移位指令

逻辑左移 n 位

格式	操作概略	操作码	执行状态	T 位
SHLL2 Rn	Rn<<2 → Rn	0100nnnn00001000	1	—
SHLL8 Rn	Rn<<8 → Rn	0100nnnn00011000	1	—
SHLL16 Rn	Rn<<16 → Rn	0100nnnn00101000	1	—

(1) 说明

将通用寄存器 Rn 的内容逻辑左移 2/8/16 位，并将结果保存在 Rn。舍弃移位后移出操作数的位。



(2) 注意事项

无

(3) 操作内容

```
SHLL2(long n) /* SHLL2 Rn */
{
    R[n] <<= 2;
    PC += 2;
}

SHLL8(long n) /* SHLL8 Rn */
{
    R[n] <<= 8;
    PC += 2;
}
SHLL16(long n) /* SHLL16 Rn */
{
    R[n] <<= 16;
    PC += 2;
}
```

(4) 使用示例

```
SHLL2     R0      ; 执行前  R0=H'12345678
                  ; 执行后  R0=H'48D159E0
SHLL8     R0      ; 执行前  R0=H'12345678
                  ; 执行后  R0=H'34567800
SHLL16    R0      ; 执行前  R0=H'12345678
                  ; 执行后  R0=H'56780000
```

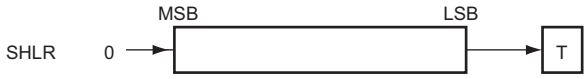
10.1.67 SHLR SHift Logical Right 移位指令

逻辑右移 1 位

格式	操作概略	操作码	执行状态	T 位
SHLR Rn	0→Rn→T	0100nnnn00000001	1	LSB

(1) 说明

将通用寄存器 Rn 的内容逻辑右移 1 位，将结果保存在 Rn。移位后将超出操作数的位传送至 T 位。



(2) 注意事项

无

(3) 操作内容

```
SHLR(long n)   /* SHLR Rn */
{
    if ((R[n]&0x00000001)==0) T = 0;
    else T = 1;
    R[n] >>= 1;
    R[n] &= 0x7FFFFFFF;
    PC += 2;
}
```

(4) 使用示例

```
SHLR      R0      ; 执行前  R0=H'80000001,T=0
                  ; 执行后  R0=H'40000000,T=1
```

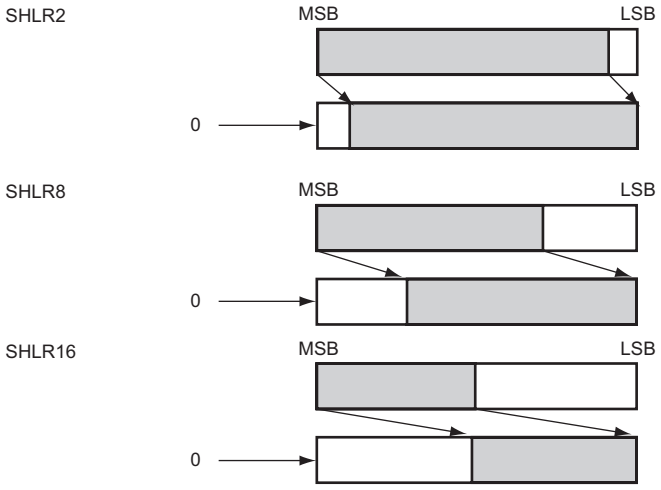
10.1.68 SHLRn n bits SHift Logical Right 移位指令

逻辑右移 n 位

格式	操作概略	操作码	执行状态	T 位
SHLR2 Rn	Rn>>2→Rn	0100nnnn00001001	1	—
SHLR8 Rn	Rn>>8→Rn	0100nnnn00011001	1	—
SHLR16 Rn	Rn>>16→Rn	0100nnnn00101001	1	—

(1) 说明

将通用寄存器 Rn 的内容逻辑右移 2/8/16 位，并将结果保存在 Rn。舍弃移位后移出操作数的位。



(2) 注意事项

无

(3) 操作内容

```
SHLR2(long n)          /* SHLR2 Rn */
{
     R[n] >>= 2;
     R[n] &= 0x3FFFFFFF;
     PC += 2;
}
SHLR8(long n)          /* SHLR8 Rn */
{
     R[n] >>= 8;
     R[n] &= 0x00FFFFFF;
     PC += 2;
}

SHLR16(long n)         /* SHLR16 Rn */
{
     R[n] >>= 16;
     R[n] &= 0x0000FFFF;
     PC += 2;
}
```

(4) 使用示例

```
          SHLR2         R0          ;执行前  R0=H'12345678
                                    ;执行后  R0=H'048D159E
          SHLR8         R0          ;执行前  R0=H'12345678
                                    ;执行后  R0=H'00123456
          SHLR16        R0          ;执行前  R0=H'12345678
                                    ;执行后  R0=H'00001234
```

10.1.69 SLEEP SLEEP 系统控制指令

转移至低功耗模式 （特权指令）

格式	操作概略	操作码	执行状态	T 位
SLEEP	睡眠或待机	0000000000011011	不定	—

(1) 说明

将 CPU 设定为低功耗状态。

低功耗模式下，保持 CPU 的内部状态，停止执行下一条指令，等待产生中断请求。产生请求时，退出低功耗状态。

SLEEP 指令为特权指令，因此只可在特权模式下使用。如果在用户模式下使用，会发生非法指令异常。

(2) 注意事项

SLEEP 性能取决于 STBCR （待机控制寄存器）。请参考该产品硬件手册的 “低功耗模式”。

(3) 操作内容

```
SLEEP( )    /* SLEEP */
{
    Sleep_standby();
}
```

(4) 使用示例

```
SLEEP               ;转移至低功耗模式
```

(5) 有可能发生的异常

- 槽非法指令异常
- 普通非法指令异常

10.1.70 STC STore Control register 系统控制指令

从控制寄存器保存 （特权指令）

格式	操作概略	操作码	执行状态	T 位
STC GBR, Rn	GBR→Rn	0000nnnn00010010	1	—
STC VBR, Rn	VBR→Rn	0000nnnn00100010	1	—
STC SSR , Rn	SSR→Rn	0000nnnn00110010	1	—
STC SPC, Rn	SPC→Rn	0000nnnn01000010	1	—
STC SGR, Rn	SGR→Rn	0000nnnn00111010	1	—
STC DBR , Rn	DBR→Rn	0000nnnn11111010	1	—
STC R0_BANK, Rn	R0_BANK→Rn	0000nnnn10000010	1	—
STC R1_BANK, Rn	R1_BANK→Rn	0000nnnn10010010	1	—
STC R2_BANK, Rn	R2_BANK→Rn	0000nnnn10100010	1	—
STC R3_BANK, Rn	R3_BANK→Rn	0000nnnn10110010	1	—
STC R4_BANK, Rn	R4_BANK→Rn	0000nnnn11000010	1	—
STC R5_BANK, Rn	R5_BANK→Rn	0000nnnn11010010	1	—
STC R6_BANK, Rn	R6_BANK→Rn	0000nnnn11100010	1	—
STC R7_BANK, Rn	R7_BANK→Rn	0000nnnn11110010	1	—
STC.L GBR, @-Rn	Rn-4→Rn, GBR→(Rn)	0100nnnn00010011	1	—
STC.L VBR, @-Rn	Rn-4→Rn, VBR→(Rn)	0100nnnn00100011	1	—
STC.L SSR , @-Rn	Rn-4→Rn, SSR→(Rn)	0100nnnn00110011	1	—
STC.L SPC, @-Rn	Rn-4→Rn, SPC→(Rn)	0100nnnn01000011	1	—
STC.L SGR, @-Rn	Rn-4→Rn, SGR→(Rn)	0100nnnn00110010	1	—
STC.L DBR , @-Rn	Rn-4→Rn, DBR→(Rn)	0100nnnn11110010	1	—
STC.L R0_BANK, @-Rn	Rn-4→Rn, R0_BANK→(Rn)	0100nnnn10000011	1	—
STC.L R1_BANK, @-Rn	Rn-4→Rn, R1_BANK→(Rn)	0100nnnn10010011	1	—
STC.L R2_BANK, @-Rn	Rn-4→Rn, R2_BANK→(Rn)	0100nnnn10100011	1	—
STC.L R3_BANK, @-Rn	Rn-4→Rn, R3_BANK→(Rn)	0100nnnn10110011	1	—
STC.L R4_BANK, @-Rn	Rn-4→Rn, R4_BANK→(Rn)	0100nnnn11000011	1	—
STC.L R5_BANK, @-Rn	Rn-4→Rn, R5_BANK→(Rn)	0100nnnn11010011	1	—
STC.L R6_BANK, @-Rn	Rn-4→Rn, R6_BANK→(Rn)	0100nnnn11100011	1	—
STC.L R7_BANK, @-Rn	Rn-4→Rn, R7_BANK→(Rn)	0100nnnn11110011	1	—

(1) 说明

将控制寄存器 GBR、VBR、SSR、SPC、SGR、DBR、Rm_BANK(m=0 ～ 7) 保存在目标。由 SR 寄存器的 RB 位指定 Rm_BANK 操作数。 RB 位为 1 时，通过 STC/STC.L 指令存取 Rm_BANK0 寄存器； RB 位为 0 时，通过 STC/STC.L 指令存取 Rm_BANK1 寄存器。

(2) 注意事项

STC/STC.L 指令只可在特权模式下使用， STC GBR,Rn/STC.L GBR,@-Rn 除外。在用户模式下使用时，会发生非法指令异常。

(3) 操作内容

```
STCGBR(int n)/* STC GBR,Rn */
{
     R[n] = GBR;
     PC += 2;
}
STCVBR(int n)/* STC VBR,Rn : Privileged */
{
     R[n] = VBR;
     PC += 2;
}
STCSSR(int n)/* STC SSR,Rn : Privileged */
{
     R[n] = SSR;
     PC += 2;
}
STCSPC(int n)/* STC SPC,Rn : Privileged */
{
     R[n] = SPC;
     PC += 2;
}
STCSGR(int n)/* STC SGR,Rn : Privileged */
{
     R[n] = SGR;
     PC += 2;
}
STCDBR(int n)/* STC DBR,Rn : Privileged */
{
     R[n] = DBR;
     PC += 2;
}
STCRm_BANK(int n)/* STC Rm_BANK,Rn : Privileged */
   /* m=0-7 */
{
     R[n] = Rm_BANK;
     PC += 2;
}
STCMGBR(int n)/* STC.L GBR,@-Rn */
{
     R[n] -= 4;
     Write_Long(R[n],GBR);
     PC += 2;
}
STCMVBR(int n)/* STC.L VBR,@-Rn : Privileged */
{
```

```
        R[n] -= 4;
        Write_Long(R[n],VBR);
        PC += 2;
}
STCMSSR(int n)/* STC.L SSR,@-Rn : Privileged */
{
        R[n] -= 4;
        Write_Long(R[n],SSR);
        PC += 2;
}
STCMSPC(int n)/* STC.L SPC,@-Rn : Privileged */
{
        R[n] -= 4;
        Write_Long(R[n],SPC);
        PC += 2;
}
STCMSGR(int n)/* STC.L SGR,@-Rn : Privileged */
{
        R[n] -= 4;
        Write_Long(R[n],SGR);
        PC += 2;
}
STCMDBR(int n)/* STC.L DBR,@-Rn : Privileged */
{
        R[n] -= 4;
        Write_Long(R[n],DBR);
        PC += 2;
}
STCMRm_BANK(int n)/* STC.L Rm_BANK,@-Rn : Privileged */
          /* m=0-7 */
{
        R[n] -= 4;
        Write_Long(R[n],Rm_BANK)
        PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 普通非法指令异常
- 槽非法指令异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

10.1.71 STS STore System register 系统控制指令

从系统寄存器保存

格式	操作概略	操作码	执行状态	T 位
STC MACH,Rn	MACH→Rn	0000nnnn00001010	1	—
STC MACL,Rn	MACL→Rn	0000nnnn00011010	1	—
STC PR,Rn	PR→Rn	0000nnnn00101010	1	—
STC.L MACH,@-Rn	Rn-4→Rn, MACH→(Rn)	0100nnnn00000010	1	—
STC.L MACL,@-Rn	Rn-4→Rn, MACL→(Rn)	0100nnnn00010010	1	—
STC.L PR,@-Rn	Rn-4→Rn, PR→(Rn)	0100nnnn00100010	1	—

(1) 说明

将系统寄存器 MACH、 MACL、 PR 保存在目标。

(2) 注意事项

无

(3) 操作内容

```
STSMACH(int n)      /* STC MACH,Rn */
     {
       R[n] = MACH;
       PC += 2;
     }
STSMACL(int n)      /* STC MACL,Rn */
     {
       R[n] = MACL;
       PC += 2;
     }
STSPR(int n)     /* STS PR,Rn */
     {
       R[n] = PR;
       PC += 2;
     }
STSMACH(int n)      /* STS.L MACH,@-Rn */
     {
       R[n] -= 4;
       Write_Long(R[n],MACH);
PC += 2;
     }
STSMACL(int n)      /* STS.L MACL,@-Rn */
     {
       R[n] -= 4;
       Write_Long(R[n],MACL);
PC += 2;
     }
STSMPR(int n)     /* STS.L PR,@-Rn */
     {
       R[n] -= 4;
Write_Long(R[n],PR);
PC += 2;
     }
```

(4) 使用示例

```
STS       MACH,R0      ; 执行前 R0=H'FFFFFFFF,MACH-H'00000000
                       ; 执行后 R0=H'00000000
STS.L     PR,@-R15     ; 执行前 R15=H'10000004
                       ; 执行后 R15=H'10000000,(R15)=PR
```

(5) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

10.1.72 SUB SUBtract binary 算术运算指令

二进制减法运算

格式	操作概略	操作码	执行状态	T 位
SUB Rm,Rn	Rn-Rm→Rn	0011nnnnmmmm1000	1	—

(1) 说明

从通用寄存器 Rn 的内容减去 Rm，并将结果保存在 Rn。使用 ADD #imm,Rn 进行与立即数的减法运算。

(2) 注意事项

无

(3) 操作内容

```
SUB(long m, long n) /* SUB Rm,Rn */
{
   R[n] -= R[m];
   PC += 2;
}
```

(4) 使用示例

```
SUB       R0,R1        ; 执行前  R0=H'00000001,R1=H'80000000
                       ; 执行后  R1=H'7FFFFFFF
```

10.1.73 SUBC SUBtract with Carry 算术运算指令

带借位的二进制减法运算

格式	操作概略	操作码	执行状态	T 位
SUBC Rm,Rn	Rn-Rm-T→Rn, 借位 →T	0011nnnnmmmm1010	1	借位

(1) 说明

从通用寄存器 Rn 的内容减去 Rm 与 T 位，将结果保存在 Rn。根据运算结果将借位反映在 T 位。进行超过 32 位的减法运算时使用该指令。

(2) 注意事项

无

(3) 操作内容

```
SUBC(long m, long n) /* SUBC Rm,Rn */
{
     unsigned long tmp0,tmp1;

     tmp1 = R[n]-R[m];
     tmp0 = R[n];
     R[n] = tmp1-T;
     if(tmp0<tmp1) T = 1;
     else T = 0;
     if(tmp1<R[n]) T = 1;
     PC += 2;
}
```

(4) 使用示例

```
CLRT          ;R0:R1(64 位 )-R2:R3(64 位 )=R0:R1(64 位 )
SUBC  R3,R1   ; 执行前  T=0,R1=H'00000000,R3=H'00000001
              ; 执行后  T=1,R1=H'FFFFFFFF
SUBC  R2,R0   ; 执行前  T=1,R0=H'00000000,R2=H'00000000
              ; 执行后  T=1,R0=H'FFFFFFFF
```

10.1.74 SUBV SUBtract with (Vflag)underflow check 算术运算指令

带下溢的二进制减法运算

格式	操作概略	操作码	执行状态	T 位
SUBV Rm,Rn	Rn-Rm→Rn, 下溢 →T	0011nnnnmmmm1010	1	下溢

(1) 说明

从通用寄存器 Rn 的内容减去 Rm，将结果保存在 Rn。发生下溢时，置位 T 位。

(2) 注意事项

无

(3) 操作内容

```
SUBV(long m, long n) /* SUBV Rm,Rn */
{
     long dest,src,ans;

     if((long)R[n]>=0) desT = 0;
     else desT = 1;
     if((long)R[m]>=0) src = 0;
     else src = 1;
     src += dest;
     R[n] -= R[m];
     if((long)R[n]>=0) anS = 0;
     else ans = 1;
     ans += dest;
     if(src==1) {
          if(ans==1) T = 1;
          else T = 0;
     }
     else T = 0;
     PC += 2;
}
```

(4) 使用示例

```
SUBV  R0,R1   ;执行前  R0=H'00000002,R1=H'80000001
              ;执行后  R1=H'7FFFFFFF,T=1
SUBV  R2,R3   ;执行前  R2=H'FFFFFFFE,R3=H'7FFFFFFE
              ;执行后  R3=H'80000000,T=1
```

10.1.75 SWAP SWAP register halves 数据传送指令

高 / 低位的交换

格式	操作概略	操作码	执行状态	T 位
SWAP.B Rm,Rn	Rm→ 低位 2 字节的高 / 低字节交换 →Rn	0110nnnnmmmm1000	1	—
SWAP.W Rm,Rn	Rm→ 高 / 低字交换 →Rn	0110nnnnmmmm1001	1	—

(1) 说明

交换通用寄存器 Rm 内容的高位与低位，并将结果保存在 Rn。

字节指定时，交换 Rm 的 bit15 ～ bit8 这 8 位与 bit7 ～ bit0 这 8 位。直接将 Rm 的高 16 位传送至 Rn 的高 16 位。

字指定时，交换 Rm 的 bit31 ～ bit16 这 16 位与 bit15 ～ bit0 这 16 位。

(2) 注意事项

无

(3) 操作内容

```
SWAPB(long m, long n)      /* SWAP.B Rm,Rn */
{
   unsigned long temp0,temp1;

   temp0 = R[m]&0xFFFF0000;
   temp1 = (R[m]&0x000000FF)<<8;
   R[n] = (R[m]&0x0000FF00)>>8;
   R[n] = R[n]|temp1|temp0;
   PC += 2;
}

SWAPW(long m, long n)      /* SWAP.W Rm,Rn */
{
   unsigned long temp;

   temp = (R[m]>>16)&0x0000FFFF;
   R[n] = R[m]<<16;
   R[n] |= temp;
   PC += 2;
}
```

(4) 使用示例

```
SWAP.B      R0,R1       ; 执行前  R0=H'12345678
                        ; 执行后  R1=H'12347856
SWAP.W      R0,R1       ; 执行前  R0=H'12345678
                        ; 执行后  R1=H'56781234
```

10.1.76 SYNCO SYNChronize data Operation 数据传送指令

数据操作的同步

格式	操作概略	操作码	执行状态	T 位
SYNCO	优先于该指令的总线存取完成之前，都不会开始通过该指令之后的指令进行的数据存取。	0000000010101011	不定	—

(1) 说明

该指令用于同步数据操作。执行该指令时，等优先于该指令的总线存取完成后，再开始通过该指令后的指令进行的数据存取。

(2) 注意事项

通过总线外的方法给 CPU 通知通过总线存取后的数据更新结果时，即关于地址映射的硬件寄存器的反映等，只通过插入 SYNCO 指令不能保证排序。此时，关于保证顺序的条件请个别参考各个寄存器的项目。

(3) 操作内容

```
SYNCO /* SYNCO*/
{
      synchronize_data_operaiton();
      PC += 2;
}
```

(4) 使用示例

1. 对与其他存储器用户共用的存储器的存取进行排序
2. 清除所有的写入缓冲器
3. 防止存储器存取的合并或存储器存取的消失
4. 等待高速缓存操作指令的完成

10.1.77 TAS Test And Set 逻辑运算指令

存储器测试与位设定

格式	操作概略	操作码	执行状态	T 位
TAS.B @Rn	(Rn) 为 0 时 1→T, 此外 0→T 两者均为 1→(Rn) 的 MSB	0100nnnn00011011	4	测试结果

(1) 说明

对于通用寄存器 Rn 的内容所指定的存储区，该指令清除对应的高速缓存块，读取该地址所示的字节数据，此数据为零时， T=1，不为零时 T=0。此后，将 bit7 置 1 并写入相同地址。此期间，不释放总线权。

此时，清除操作的执行如下。

清除操作中，作为有效地址通过通用寄存器 Rn 的内容存取数据。发生高速缓存命中，且对应的高速缓存块脏 （U 位 =1）时，将此高速缓存块的内容回写到外部存储器后，高速缓存块为无效 （V 位 =0）。发生高速缓存命中，且对应的高速缓存块干净 （U 位 =0）时，只是高速缓存块为无效 （V 位 =0）。发生高速缓存未命中时，或存取的存储器位置为禁止高速缓存时，不执行清除。

自动执行 TAS.B 的两个存储器存取。 TAS.B 的两个存取之间，不执行其它存储器存取。

(2) 注意事项

无

(3) 操作内容

```
TAS(int n) /* TAS.B @Rn */
{
     int temp;

     temp = (int)Read_Byte(R[n]); /* Bus Lock */
     if(temp==0) T = 1;
     else T = 0;
     temp |= 0x00000080;
     Write_Byte(R[n],temp);     /* Bus unlock */
     PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

异常处理中，将通过该指令进行的数据存取作为字节加载、字节保存来处理。

10.1.78 TRAPA TRAP Always 系统控制指令

陷阱异常处理

格式	操作概略	操作码	执行状态	T 位
TRAPA #imm	Imm<<2→TRA, PC+2→SPC, SR→SSR,R15→SGR, 1→SR.MD/BL/RB, H'160→EXPEVT, VBR+H'0100→PC	11000011iiiiiiii	13	—

(1) 说明

开始陷阱异常处理。将 （PC+2）、 SR 及 R15 的值保存在 SPC、 SSR 及 SGR，将 8 位立即数保存在 TRA 寄存器 （bit9 ～ 2）。将处理模式切换为特权模式 （SR 的 MD 位为 1）， SR 的 BL 位与 RB 位为 1。结果，屏蔽异常与中断请求使其不被接受，而选择 BANK1 寄存器 （R0_BANK1 ～ R7_BANK1）。将异常代码 H'160 写入 EXPEVT 寄存器 （bit11 ～ 0）。程序转移至以 VBR 寄存器与偏移 H'00000100 的和所示的地址 （VBR+H'00000100）。

(2) 注意事项

无

(3) 操作内容

```
TRAPA(int i) /* TRAPA #imm */
{
      int imm;
      imm = (0x000000FF & i);
      TRA = imm << 2;
      SSR = SR;
      SPC = PC + 2;
      SGR = R15;
      SR.MD = 1;
      SR.BL = 1;
      SR.RB = 1;
      EXPEVT = H'00000160;
      PC = VBR + H'00000100;
}
```

(4) 有可能发生的异常

- 槽非法指令异常
- 无条件陷阱

10.1.79 TST TeST logical 逻辑运算指令

逻辑 “与”运算的 T 位置位

格式	操作概略	操作码	执行状态	T 位
TST Rm,Rn	Rn & Rm、结果为 0 时 1→T 此外 0→T	0010nnnnmmmm1000	1	测试结果
TST #imm,R0	R0 & imm、结果为 0 时 1→T 此外 0→T	11001000iiiiiiii	1	测试结果
TST.B #imm,@(R0,GBR)	(R0+GBR)&imm、结果为 0 时 1→T 此外 0→T	11001100iiiiiiii	3	测试结果

(1) 说明

获取通用寄存器 Rn 的内容与 Rm 的逻辑 “与”，结果为零时 T 位置位。结果不为零时，清除 T 位。不改变 Rn 的内容。

可进行通用寄存器 R0 与零扩展后的 8 位立即数的逻辑 “与”，或可在带变址的 GBR 间接寻址方式下进行 8 位存储器与 8 位立即数的逻辑 “与”。不改变 R0 或存储器的内容。

(2) 注意事项

无

(3) 操作内容

```
TST(long m, long n) /* TST Rm,Rn */
{
     if((R[n]&R[m])==0) T = 1;
     else T = 0;
     PC += 2;
}

TSTI(long i)/* TST #imm,R0 */
{
     long temp;

     temp = R[0] & (0x000000FF & (long)i);
     if(temp==0) T = 1;
     else T = 0;
     PC += 2;
}
TSTM(long i) /* TST.B #imm,@(R0,GBR) */
{
     long temp;

     temp = (long)Read_Byte(GBR+R[0]);
     temp &= (0x000000FF & (long)i);
     if(temp==0) T = 1;
     else T = 0;
     PC += 2;
}
```

(4) 使用示例

```
TST         R0,R0           ; 执行前 R0=H'00000000
                            ; 执行后 T=1
TST         #H'80,R0        ; 执行前 R0=H'FFFFFF7F
                            ; 执行后 T=1
TST.B       #H'A5,@(R0,GBR) ; 执行前 (R0,GBR)=H'A5
                            ; 执行后 T=0
```

(5) 有可能发生的异常

可能只在执行 TST.B 指令时会发生以下异常。

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

异常处理中，将通过该指令进行的数据存取作为字节加载、字节保存来处理。

10.1.80 XOR eXclusive OR logical 逻辑运算指令

“异”运算

格式	操作概略	操作码	执行状态	T 位
XOR Rm,Rn	Rn ^ Rm → Rn	0010nnnnmmmm1010	1	—
XOR #imm,R0	R0 ^ imm → R0	11001010iiiiiiii	1	—
XOR.B #imm,@(R0,GBR)	(R0+GBR)^imm → (R0+GBR)	11001110iiiiiiii	3	—

(1) 说明

进行通用寄存器 Rn 的内容与 Rm 的 “异”运算，将结果保存在 Rn。

可进行通用寄存器 R0 与零扩展后的 8 位立即数的 “异”运算，或可在带变址的 GBR 间接寻址方式下进行 8 位存储器与 8 位立即数的 “异”运算。

(2) 注意事项

无

(3) 操作内容

```
XOR(long m, long n) /* XOR Rm,Rn */
{
     R[n] ^= R[m];
     PC += 2;
}

XORI(long i)  /* XOR #imm,R0 */
{
     R[0] ^= (0x000000FF & (long)i);
     PC += 2;
}

XORM(long i)  /* XOR.B #imm,@(R0,GBR) */
{
     int temp;

     temp = (long)Read_Byte(GBR+R[0]);
     temp ^= (0x000000FF & (long)i);
     Write_Byte(GBR+R[0],temp);
     PC += 2;
}
```

(4) 使用示例

```
XOR     R0,R1                   ;执行前  R0=H'AAAAAAAA,R1=H'55555555
                                ;执行后  R1=H'FFFFFFFF
XOR     #H'F0,R0                ;执行前  R0=H'FFFFFFFF
                                ;执行后  R0=H'FFFFFF0F
XOR.B   #H'A5,@(R0,GBR)         ;执行前  (R0,GBR)=H'A5
                                ;执行后  (R0,GBR)=H'00
```

(5) 有可能发生的异常

仅执行 XOR.B 指令时，可能会发生以下异常。

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

异常处理中，将通过该指令进行的数据存取作为字节加载、字节保存来处理。

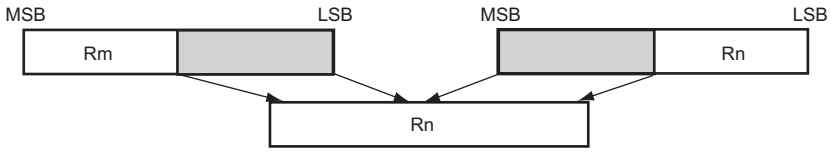
10.1.81 XTRCT eXTRaCT 数据传送指令

抽取连接寄存器的中间位

格式	操作概略	操作码	执行状态	T 位
XTRCT Rm,Rn	Rm:Rn 的中间 32 位 →Rn	0010nnnnmmmm1101	1	—

(1) 说明

从连接通用寄存器 Rm 与 Rn 的 64 位内容中抽取中间的 32 位，并将结果保存在 Rn。



(2) 注意事项

无

(3) 操作内容

```
XTRCT(long m, long n)     /* XTRCT Rm,Rn */
{
     unsigned long temp;

     temp = (R[m]<<16) & 0xFFFF0000;
     R[n] = (R[n]>>16) & 0x0000FFFF;
     R[n] |= temp;
     PC += 2;
}
```

(4) 使用示例

```
        XTRCT       R0,R1          ;执行前  R0=H'01234567,R1=H'89ABCDEF
                                   ;执行后  R1=H'456789AB
```

10.2 CPU 指令 （FPU 相关）

本节记述 CPU 指令中支持 FPU 的 CPU 指令及与 SH-4A、 SH4AL-DSP 中一部分功能不同的指令。

10.2.1 BSR Branch to SubRoutine 转移指令

转移至子程序过程 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
BSR label	PC+4→PR, PC+4+disp×2→PC	1011ddddddddddd	1	—

(1) 说明

转移至地址 （PC+4+ 位移量 ×2），将地址 （PC+4）保存在 PR。 PC 源值为 BSR 指令地址。因为 12 位位移量在符号扩展后要乘以 2，所以与转移目标的相对距离在 –4096 字节～ +4094 字节范围。未到达转移目标时，可通过 JSR 指令实现此转移。

(2) 注意事项

因为该指令是延迟转移指令，所以先执行该指令的下一条指令，然后再执行转移目标指令。
该指令与其下一条指令之间，不接受中断。下一条指令为转移指令时，将其识别为槽非法指令。

(3) 操作内容

```
BSR(int d)   /* BSR disp */
{
    int disp;
    unsigned int temp;

    temp = PC;
    if((d & 0x800) == 0)
       disp =  (0x00000FFF & d);
        elsedisp =  (0xFFFFF000 | d);
    PR = PC + 4;
    PC = PC + 4 + (disp << 1);
    Delay_Slot(temp + 2);
}
```

(4) 使用示例

```
        BSR     TRGET       ; 转移至 TRGET。
        MOV     R3,R4;      ; 转移之前，先执行 MOV。
        ADD     R0,R1       ; 从子程序过程返回的目标 (PR 的内容 )。
    • • • • •
    • • • • •
TRGET:                      ;← 过程的入口
        MOV     R2,R3;      ;
        RTS                 ; 返回至上述 ADD 指令。
        MOV     #1,R0       ; 转移之前，先执行 MOV。
```

(5) 有可能发生的异常

- 槽非法指令异常

10.2.2 BSRF Branch to SubRoutine Far 转移指令

转移至子程序过程 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
BSRF Rn	PC+4→PR, PC+4+Rn→PC	0000nnnn00000011	1	—

(1) 说明

转移至地址 （PC+4+Rn），将地址 （PC+4）保存在 PR。 PC 源值为 BSRF 指令地址。转移目标是给 PC+4 加上通用寄存器 Rn 的内容的 32 位数据后的地址。

(2) 注意事项

因为该指令是延迟转移指令，所以先执行该指令的下一条指令，然后再执行转移目标指令。

该指令与其下一条指令之间，不接受中断。下一条指令为转移指令时，将其识别为槽非法指令。

(3) 操作内容

```
BSRF(int n)     /* BSRF Rn */
{
     unsigned int temp;

     temp = PC;
     PR = PC + 4;
     PC = PC + 4 + R[n];
     Delay_Slot(temp + 2);
}
```

(4) 使用示例

```
          MOV.L       #(TRGETBSRF_PC),R0    ; 设定位移量。
          BRSF        R0                    ; 转移至 TRGET。
          MOV         R3,R4                 ; 转移之前，先执行 MOV。
BSRF_PC:                                    ;
          ADD         R0,R1                 ;
          ......
 TRGET:                                     ;← 过程的入口
          MOV         R2,R3                 ;
          RTS                               ; 返回至上述 ADD 指令。
          MOV         #1,R0                 ; 转移之前先执行 MOV。
```

(5) 有可能发生的异常

- 槽非法指令异常

10.2.3 JSR Jump to SubRoutine 转移指令

转移至子程序过程 延迟转移指令

格式	操作概略	操作码	执行状态	T 位
JSR @Rn	PC+4→PR, Rn→PC	0100nnnn00001011	1	—

(1) 说明

执行该指令的下一条指令之后，向指定地址的子程序过程进行延迟转移。将返回目标地址 （PC+4）保存在 PR，向通用寄存器 Rn 所示的地址转移。与 RTS 组合，用于子程序过程调用。

(2) 注意事项

因为该指令是延迟转移指令，所以先执行该指令的下一条指令，然后执行转移目标指令。
该指令与其下一条指令之间，不接受中断。下一条指令为转移指令时，将其识别为槽非法指令。

(3) 操作内容

```
JSR(int n)/* JSR @Rn */
{
    unsigned int temp;

    temp = PC;
    PR = PC + 4;
    PC = R[n];
    Delay_Slot(temp + 2);
}
```

(4) 使用示例

```
            MOV.L       JSR_TABLE,R0    ;R0=TRGET 地址
            JSR         @R0             ; 转移至 TRGET。
            XOR         R1,R1           ; 转移之前，先执行 XOR。
            ADD         R0,R1           ;¨ 是从过程的返回目标
            • • • • • •                 (PR 的内容 )。
            .align      4
JSR_TABLE:  .data.l     TRGET           ; 跳转表
TRGET:      NOP                         ;¨ 过程的入口
            MOV         R2,R3           ;
            RTS                         ; 返回至上述 ADD 指令。
            MOV         #70,R1          ;RTS 之前，先执行 MOV。
```

(5) 有可能发生的异常

- 槽非法指令异常

10.2.4 LDC LoaD to Control register 系统控制指令

加载到控制寄存器 （特权指令）

格式	操作概略	操作码	执行状态	T 位
LDC Rm, SR	Rm→SR	0100mmmm00001110	4	LSB
LDC.L @Rm+, SR	(Rm)→SR, Rm+4→Rm	0100mmmm00000111	4	LSB

(1) 说明

将源操作数保存在控制寄存器 SR。

(2) 注意事项

只可在特权模式下使用。如果在用户模式下使用，则会发生非法指令异常。

(3) 操作内容

```
LDCSR(int m)      /* LDC Rm,SR : Privileged */
{
    SR = R[m] & 0x700083F3;
    PC += 2;
}
LDCMSR(int m)      /* LDC.L @Rm+,SR: Privileged */
{
    SR = Read Long(R[m]) & 0x700083F3;
    R[m] += 4;
    PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 普通非法指令异常
- 槽非法指令异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.2.5 LDS LoaD to FPU System register 系统控制指令

加载到 FPU 系统寄存器

格式	操作概略	操作码	执行状态	T 位
LDS Rm,FPUL	Rm→FPUL	0100mmmm01011010	1	—
LDS.L @Rm+,FPUL	(Rm)→FPUL, Rm+4→Rm	0100mmmm01010110	1	—
LDS Rm,FPSCR	Rm→FPSCR	0100mmmm01101010	1	—
LDS.L @Rm+,FPSCR	(Rm)→FPSCR, Rm+4→Rm	0100mmmm01100110	1	—

(1) 说明

将源操作数保存在 FPU 系统寄存器 FPUL、 FPSCR。

(2) 注意事项

无

(3) 操作内容

```
#define FPSCR_MASK 0x003FFFFF

LDSFPUL(int  m,int *FPUL)/* LDS Rm,FPUL  */
{
          *FPUL = R[m];
              PC += 2;
}
LDSMFPUL(int  m,int *FPUL)/* LDS.L @Rm+,FPUL  */
{
              *FPUL = Read_Long(R[m]);
              R[m] += 4;
              PC += 2;
}
LDSFPSCR(int  m)/* LDS Rm,FPSCR  */
{
              FPSCR = R[m] & FPSCR_MASK;
              PC += 2;
}
LDSMFPSCR(int  m)/* LDS.L @Rm+,FPSCR  */
{
              FPSCR = Read_Long(R[m]) & FPSCR_MASK;
              R[m] += 4;
              PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 数据地址错误

10.2.6 STC STore Control register 系统控制指令

从控制寄存器保存 （特权指令）

格式	操作概略	操作码	执行状态	T 位
STC SR, Rn	SR→Rn	0000nnnn00000010	1	—
STC.L SR, @-Rn	Rn-4→Rn, SR→(Rn)	0100nnnn00000011	1	—

(1) 说明

将控制寄存器 SR 保存在目标。

(2) 注意事项

只可在特权模式下使用。如果在用户模式下使用，则发生非法指令异常。

(3) 操作内容

```
STCSR(int n)      /* STC SR,Rn : Privileged */
{
      R[n] = SR;
      PC += 2;
}
STCMSR(int n)     /* STC.L SR,@-Rn : Privileged */
{
      R[n] -= 4;
      Write_Long(R[n],SR);
      PC += 2;
}
```

(4) 有可能发生的异常

- 数据TLB多重命中异常
- 普通非法指令异常
- 槽非法指令异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

10.2.7 STS Store from FPU System register 系统控制指令

从 FPU 系统寄存器保存

格式	操作概略	操作码	执行状态	T 位
STS FPUL,Rn	FPUL→Rn	0000nnnn01011010	1	—
STS FPSCR,Rn	FPSCR→Rn	0000nnnn01101010	1	—
STS.L FPUL,@-Rn	Rn-4→Rn, FPUL→(Rn)	0100nnnn01010010	1	—
STS.L FPSCR,@-Rn	Rn-4→Rn, FPSCR→(Rn)	0100nnnn01100010	1	—

(1) 说明

将 FPU 系统寄存器 FPUL、 FPSCR 保存在目标。

(2) 注意事项

无

(3) 操作内容

```
STSFPUL(int  n, int *FPUL)/* STS FPUL,Rn */
{
          R[n] = *FPUL;
          PC += 2;
}
STSMFPUL(int  n, int *FPUL)/* STS.L FPUL,@-Rn  */
{
          R[n] -= 4;
          Write_Long(R[n],*FPUL) ;
          PC += 2;
}
STSFPSCR(int  n)/* STS FPSCR,Rn  */
{
          R[n] = FPSCR & 0x003FFFFF;
             PC += 2;
}
STSMFPSCR(int  n)/* STS.L FPSCR,@-Rn  */
{
          R[n] -= 4;
          Write_Long(R[n],FPSCR & 0x003FFFFF)
          PC += 2;
}
```

(4) 使用示例

```
STS
Example 1:
MOV.L       #H'12ABCDEF,  R12
LDS         R12, FPUL
STS         FPUL, R13
                          ; After executing the STS instruction:
                          ; R13 = 12ABCDEF
Example 2:
STS         FPSCR, R2     ; After executing the STS instruction:
                          ; The current content of FPSCR is stored in register R2
STS.L
Example 1:
MOV.L       #H'0C700148,R7
STS.L       FPUL, @-R7
                          ; Before executing the STS.L instruction:
                          ; R7 = 0C700148
                          ; After executing the STS.L instruction:
                          ; R7 = 0C700144, and the content of FPUL is saved at
                          memory
                          ; location 0C700144.
Example 2:
MOV.L       #H'0C700154,R8
STS.L       FPSCR, @-R8

                          ; After executing the STS.L instruction:
                          ; The content of FPSCR is saved at memory location
                          0C700150.
```

(5) 可能会发生的异常

- 数据TLB多重命中异常
- 数据TLB未命中异常
- 数据TLB保护违反异常
- 初始页写入异常
- 数据地址错误

10.3 FPU 指令

对使用了 FPU 指令的 C 语言描述的操作内容进行说明时，除对说明 CPU 指令的操作内容时所使用的资料或函数外，还使用以下资料与函数。

- 为浮点用的定义描述。

```
#define PZERO                   0
#define NZERO                   1
#define DENORM                  2
#define NORM                    3
#define PINF                    4
#define NINF                    5
#define qNaN                    6
#define sNaN                    7
#define EQ                      0
#define GT                      1
#define LT                      2
#define UO                      3
#define INVALID                 4
#define FADD                    0
#define FSUB                    1

#define CAUSE                   0x0003f000          /* FPSCR(bit17-12) */
#define SET_E                   0x00020000          /* FPSCR(bit17) */
#define SET_V                   0x00010040          /* FPSCR(bit16,6) */
#define SET_Z                   0x00008020          /* FPSCR(bit15,5) */
#define SET_O                   0x00004010          /* FPSCR(bit14,4) */
#define SET_U                   0x00002008          /* FPSCR(bit13,3) */
#define SET_I                   0x00001004          /* FPSCR(bit12,2) */
#define ENABLE_VOUI             0x00000b80          /* FPSCR(bit11,9-7) */
#define ENABLE_V                0x00000800          /* FPSCR(bit11) */
#define ENABLE_Z                0x00000400          /* FPSCR(bit10) */
#define ENABLE_OUI              0x00000380          /* FPSCR(bit9-7) */
#define ENABLE_I                0x00000080          /* FPSCR(bit7) */
#define FLAG                    0x0000007C          /* FPSCR(bit6-2) */

#define FPSCR_FR                FPSCR>>21&1
#define FPSCR_PR                FPSCR>>19&1
#define FPSCR_DN                FPSCR>>18&1
#define FPSCR_I                 FPSCR>>12&1
#define FPSCR_RM                FPSCR&1
#define FR_HEX                  frf.l[ FPSCR_FR]
#define FR                      frf.f[ FPSCR_FR]
#define DR_HEX                  frf.l[ FPSCR_FR]
#define DR                      frf.d[ FPSCR_FR]
#define XF_HEX                  frf.l[~FPSCR_FR]
#define XF                      frf.f[~FPSCR_FR]
#define XD                      frf.d[~FPSCR_FR]


union {
        int     l[2][16];
        float   f[2][16];
        double d[2][8];
} frf;
int FPSCR;
```

```
int sign_of(int n)
{
      return(FR_HEX[n]>>31);
}
int data_type_of(int n) {
int abs;
     abs = FR_HEX[n] & 0x7fffffff;
     if(FPSCR_PR==0) {/* 单精度 */
            if(abs<0x00800000){
                     if((FPSCR_DN == 1) || (abs == 0x00000000)){
                             if(sign_of(n) == 0){zero(n, 0); return(PZERO);}
                            else                          {zero(n, 1); return(NZERO);}
                             }
                            elsereturn(DENORM);
        }
        else if(abs<0x7f800000)                    return(NORM);
   else if(abs==0x7f800000) {
                      if(sign_of(n)==0)                 return(PINF);
                      else                                       return(NINF);
              }
              else if(abs<0x7fc00000)              return(qNaN);
                     else                                        return(sNaN);
         }
   else {      /* 双精度 */
        if(abs<0x00100000){
                 if((FPSCR_DN==1) || ((abs==0x00000000)
                        && (FR_HEX[n+1]==0x00000000){
                        if(sign_of(n)==0)     {zero(n,0); return(PZERO);}
                        else                            {zero(n,1); return(NZERO);}
          }
          else return(DENORM);
         }
}
         else if(abs < 0x7ff00000)         return(NORM);
         else if((abs = =0x7ff00000)     &&
                  (FR_HEX[n+1]= =0x00000000)) {
          if(sign_of(n) = =0)   return(PINF);
          else                              return(NINF);
       }
       else if(abs < 0x7ff80000) return(qNaN);
       else                       return(sNaN);
     }
}
void register_copy(int m,n)
{
```

```
                        FR[n] = FR[m];
        if(FPSCR_PR == 1)FR[n+1] = FR[m+1];
}
void normal_faddsub(int m,n,type)
{
union {
      float f;
      int l;
}     dstf,srcf;
union {
      double d;
      int l[2];
}     dstd,srcd;
union {              /* "long double" 格式 :*/
      long double x;/*1-bit 符号*/
      int l[4]; /*  15-bit 指数*/
}     dstx;      /*     112-bit 小数*/
      if(FPSCR_PR == 0) {
          if(type == FADD) srcf.f =  FR[m];
          else             srcf.f = -FR[m];
          dstd.d = FR[n];  /* 从单精度向双精度转换 */
          dstd.d += srcf.f;
       if(((dstd.d == FR[n]) && (srcf.f != 0.0)) ||
          ((dstd.d == srcf.f) && (FR[n] != 0.0))) {
          set_I();
          if(sign_of(m) ^ sign_of(n)) {
                 dstd.l[1] -= 1;
             if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
          }
       }
       if(dstd.l[1] & 0x1fffffff) set_I();
       dstf.f += srcf.f; /* 向接近方向舍入 */
       if(FPSCR_RM == 1) {
          dstd.l[1] &= 0xe0000000; /* 向 0 舍入 */
          dstf.f = dstd.d;
       }
       check_single_exception(&FR[n],dstf.f);
   } else {
       if(type == FADD)   srcd.d =  DR[m>>1];
       else                  srcd.d = -DR[m>>1];
       dstx.x = DR[n>>1]; /* 从双精度向扩展双精度转换 */
       dstx.x += srcd.d;
       if(((dstx.x == DR[n>>1]) && (srcd.d != 0.0)) ||
          ((dstx.x == srcd.d) && (DR[n>>1] != 0.0)) ) {
          set_I();
          if(sign_of(m)^ sign_of(n)) {
```

```
                dstx.l[3] -= 1;
                if(dstx.l[3] == 0xffffffff) {dstx.l[2] -= 1;
                if(dstx.l[2] == 0xffffffff) {dstx.l[1] -= 1;
                if(dstx.l[1] == 0xffffffff) {dstx.l[0] -= 1;}}}
            }
        }
        if((dstx.l[2] & 0x0fffffff) || dstx.l[3]) set_I();
        dst.d += srcd.d; /* 向接近方向舍入 */
            if(FPSCR_RM == 1) {
            dstx.l[2] &= 0xf0000000; /* 向 0 舍入 */
            dstx.l[3]  = 0x00000000;
            dst.d = dstx.x;
        }
        check_double_exception(&DR[n>>1] ,dst.d);
    }
}
void normal_fmul(int m,n)
{
union {
        float f;
        int l;
}       tmpf;
union {
        double d;
        int l[2];
}         tmpd;
union {
        long double x;
        int l[4];
}       tmpx;
        if(FPSCR_PR == 0) {
            tmpd.d = FR[n]; /* 从单精度到双精度 */
            tmpd.d *= FR[m]; /* 准确完成 */
            tmpf.f *= FR[m]; /* 向接近方向舍入 */
            if(tmpf.f != tmpd.d) set_I();
            if((tmpf.f > tmpd.d) && (FPSCR_RM == 1)) {
              tmpf.l -= 1; /* 向 0 舍入 */
            }
            check_single_exception(&FR[n],tmpf.f);
    } else {
        tmpx.x = DR[n>>1]; /* 从单精度到双精度 */
        tmpx.x *= DR[m>>1]; /* 准确完成 */
        tmpd.d *= DR[m>>1]; /* 向接近方向舍入 */
        if(tmpd.d != tmpx.x) set_I();
        if(tmpd.d > tmpx.x) && (FPSCR_RM == 1)) {
            tmpd.l[1] -= 1; /* 向 0 舍入 */
```

```
            if(tmpd.l[1] == 0xffffffff) tmpd.l[0] -= 1;
        }
        check_double_exception(&DR[n>>1], tmpd.d);
    }
}
void fipr(int m,n)
{
union {
      double d;
      int l[2];
}     mlt[4];

float dstf;
      if((data_type_of(m) == sNaN) || (data_type_of(n) == sNaN) ||
       (data_type_of(m+1) == sNaN) || (data_type_of(n+1) == sNaN) ||
             (data_type_of(m+2) == sNaN) || (data_type_of(n+2) == sNaN) ||
       (data_type_of(m+3) == sNaN) || (data_type_of(n+3) == sNaN) ||
       (check_product_invalid(m,n)) ||
       (check_product_invalid(m+1,n+1)) ||
       (check_product_invalid(m+2,n+2)) ||
       (check_product_invalid(m+3,n+3)) )  invalid(n+3);
      else if((data_type_of(m) == qNaN)|| (data_type_of(n) == qNaN)||
       (data_type_of(m+1) == qNaN) || (data_type_of(n+1) == qNaN) ||
       (data_type_of(m+2) == qNaN) || (data_type_of(n+2) == qNaN) ||
       (data_type_of(m+3) == qNaN) || (data_type_of(n+3) == qNaN)) qnan(n+3);
      else if(check_ positive_infinity() &&
          (check_ negative_infinity()) invalid(n+3);
         else if(check_ positive_infinity())inf(n+3,0);
      else if(check_ negative_infinity())inf(n+3,1);
   else {
       for(i=0;i<4;i++) {
          /* FPSCR_DN == 1 时，置 0) */
                    if(data_type_of(m+i) == PZERO)FR[m+i] = +0.0;
                    else if(data_type_of(m+i) == NZERO)FR[m+i] = -0.0;
                    if(data_type_of(n+i) == PZERO)FR[n+i] = +0.0;
                    else if(data_type_of(n+i) == NZERO)FR[n+i] = -0.0;
                    mlt[i].d = FR[m+i];
                    mlt[i].d *= FR[n+i];

   /* 准确地说，因为在 FIPR 下舍入低 18bit，所以此处描述与硬件不同，相对来说更简单一些。 */
       mlt[i].l[1] &= 0xff000000;
       mlt[i].l[1] |= 0x00800000;
       }
       mlt[0].d += mlt[1].d + mlt[2].d + mlt[3].d;
       mlt[0].l[1] &= 0xff800000;
       dstf = mlt[0].d;
```

```
        set_I();
        check_single_exception(&FR[n+3],dstf);
    }
}
void check_single_exception(float *dst,result)
{
union {
      float f;
         int l;
      }tmp;
float abs;
      if(result < 0.0)tmp.l = 0xff800000; /* - 无穷大 */
      else            tmp.l = 0x7f800000; /* + 无穷大 */
      if(result == tmp.f) {
         set_O(); set_I();
         if(FPSCR_RM == 1){
             tmp.l -= 1; /* 规格化数的最大值 */
             result = tmp.f;
         }
      }
      if(result < 0.0)abs = -result;
      else            abs =  result;
      tmp.l = 0x00800000; /* 规格化数的最小值 */
      if(abs < tmp.f) {
          if((FPSCR_DN == 1) && (abs != 0.0)) {
          set_I();
          if(result < 0.0)result = -0.0; /* 将非规格化数置 0。 */
          else              result =  0.0;
       }
       if(FPSCR_I == 1) set_U();
    }
    if(FPSCR & ENABLE_OUI) fpu_exception_trap();
    else                  *dst = result;
}
void check_double_exception(double *dst,result)
{
union {
      double d;
      int l[2];
      }tmp;
double abs;
      if(result < 0.0)tmp.l[0] = 0xfff00000; /* - 无穷大 */
      else                  tmp.l[0] = 0x7ff00000; /* + 无穷大 */
                        tmp.l[1] = 0x00000000;
      if(result == tmp.d)
         set_O(); set_I();
```

```
          if(FPSCR_RM == 1) {
             tmp.l[0] -= 1;
                    tmp.l[1] = 0xffffffff;
                    result = tmp.d; /* 规格化数的最大值 */
        }
    }
    if(result < 0.0)abs = -result;
    else                      abs =  result;
    tmp.l[0] = 0x00100000; /* 规格化数的最小值 */
    tmp.l[1] = 0x00000000;
    if(abs < tmp.d) {
       if((FPSCR_DN == 1) && (abs != 0.0)) {
          set_I();
          if(result < 0.0)result = -0.0; /* 将非规格化数置 0。 */
          else            result =  0.0;
       }
       if(FPSCR_I == 1) set_U();
    }
    if(FPSCR & ENABLE_OUI)fpu_exception_trap();
       else                     *dst = result;
}
int check_product_invalid(int m,n)
{
       return(check_product_infinity(m,n)  &&
          ((data_type_of(m) == PZERO) || (data_type_of(n) == PZERO) ||
           (data_type_of(m) == NZERO) || (data_type_of(n) == NZERO)));
}
int check_ product_infinity(int m,n)
{
    return((data_type_of(m) == PINF) || (data_type_of(n) == PINF) ||
                     (data_type_of(m) == NINF) || (data_type_of(n) == NINF));
}
int check_ positive_infinity(int m,n)
{
 return(((check_ product_infinity(m,n) && (~sign_of(m)^ sign_of(n))) ||
    ((check_ product_infinity(m+1,n+1) && (~sign_of(m+1)^ sign_of(n+1))) ||
    ((check_ product_infinity(m+2,n+2) && (~sign_of(m+2)^ sign_of(n+2))) ||
    ((check_ product_infinity(m+3,n+3) && (~sign_of(m+3)^ sign_of(n+3))));
}
int check_ negative_infinity(int m,n)
{
  return(((check_ product_infinity(m,n) && (sign_of(m)^ sign_of(n))) ||
    ((check_ product_infinity(m+1,n+1) && (sign_of(m+1)^ sign_of(n+1))) ||
    ((check_ product_infinity(m+2,n+2) && (sign_of(m+2)^ sign_of(n+2))) ||
    ((check_ product_infinity(m+3,n+3) && (sign_of(m+3)^ sign_of(n+3))));
}
```

```
void clear_cause () {FPSCR &= ~CAUSE;}
void set_E() {FPSCR |= SET_E; fpu_exception_trap();}
void set_V() {FPSCR |= SET_V;}
void set_Z() {FPSCR |= SET_Z;}
void set_O() {FPSCR |= SET_O;}
void set_U() {FPSCR |= SET_U;}
void set_I() {FPSCR |= SET_I;}
void invalid(int n)
{
      set_V();
      if((FPSCR & ENABLE_V) == 0 qnan(n);
      elsefpu_exception_trap();
}

void dz(int n,sign)
{
      set_Z();
      if((FPSCR & ENABLE_Z) == 0 inf(n,sign);
      else      fpu_exception_trap();
}
void zero(int n,sign)
{
      if(sign == 0)  FR_HEX [n]   = 0x00000000;
      else           FR_HEX [n]   = 0x80000000;
      if(FPSCR_PR==1)FR_HEX [n+1] = 0x00000000;
}
void inf(int n,sign) {
     if(FPSCR_PR==0) {
        if(sign == 0) FR_HEX [n]   = 0x7f800000;
        else          FR_HEX [n]   = 0xff800000;
   } else {
        if(sign == 0) FR_HEX [n]   = 0x7ff00000;
        else          FR_HEX [n]   = 0xfff00000;
                      FR_HEX [n+1] = 0x00000000;
   }
}
void qnan(int n)
{
      if(FPSCR_PR==0) FR[n]   = 0x7fbfffff;
       else {            FR[n]   = 0x7ff7ffff;
                         FR[n+1] = 0xffffffff;
   }
}
```

10.3.1 FABS Floating - point ABSolute value 浮点指令

浮点绝对值

PR	格式	操作概略	操作码	执行状态	T 位
0	FABS FRn	FRn & H'7FFFFFFF→FRn	1111nnnn01011101	1	—
1	FABS DRn	DRn & H'7FFFFFFFFFFFFFFF →DRn	1111nnn001011101	1	—

(1) 说明

将浮点寄存器 FRn/DRn 的内容的最高位清 0，将结果保存在 FRn/DRn。
不更新 FPSCR 的 cause/flag 部分。

(2) 注意事项

无

(3) 操作内容

```
void FABS (int n){
     FR[n] = FR[n] & 0x7fffffff;
     pc += 2;
}
/* 不取决于精度，进行相同操作。 */
```

(4) 有可能发生的异常

无

10.3.2 FADD Floating - point ADD 浮点指令

浮点加法运算

PR	格式	操作概略	操作码	执行状态	T 位
0	FADD FRm,FRn	FRn+FRm→FRn	1111nnnnmmmm0000	1	—
1	FADD DRm,DRn	DRn+DRm→DRn	1111nnn0mmm00000	1	—

(1) 说明

- FPSCR.PR=0时
  对FRn与FRm的内容的2个单精度浮点数进行算术加法运算，将结果保存在FRn。
- FPSCR.PR=1时
  对DRn与DRm的内容的2个双精度浮点数进行算术加法运算，并将结果保存在DRn。
  如果设定为FPSCR.enable.I，发生FPU异常陷阱，但与异常的发生无关。如果设定为FPSCR.enable.O/U，当实际产生FPU异常源，及满足指令固有的特殊条件时，发生FPU异常陷阱。该特殊条件将在本节末进行描述。发生异常时，在FPSCR.cause FPSCR.flag反映正确的异常信息，且不更新FRn/DRn。请通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FADD (int m,n)
{
      pc += 2;
      clear_cause();
      if((data_type_of(m)==sNaN) || (data_type_of(n)==sNaN)) invalid(n);
      else if((data_type_of(m)==qNaN) || (data_type_of(n)==qNaN)) qnan(n);
            else if((data_type_of(m)==DENORM)
               || (data_type_of(n)==DENORM)) set_E();
            else switch (data_type_of(m)){
                  case NORM:switch (data_type_of(n)){
                              case NORM:normal_faddsub(m,n,ADD);
                                        break;
                              case PZERO:
                              case NZERO:register_copy(m,n);
                                         break;
                              default:break;
                              }
                              break;
                        case PZERO:switch (data_type_of(n)){
                               case NZERO:zero(n,0);break;
                               default:   break;
                               }
                              break;
                   case NZERO:break;
                  case PINF:switch (data_type_of(n)){
                            case NINF:invalid(n); break;
                            default:inf(n,0);break;
```

```
                    }
                    break;
                case NINF:switch (data_type_of(n)){
                    case PINF:invalid(n);break;
                    default:inf(n,1);break;
                    }
                    break;
                }
    }
```

FADD 特殊情况

FADD	FRn,DRm							
FRm,DRm	+NORM	-NORM	+0	-0	+inf	-inf	qNaN	sNaN
+NORM	FADD							
-NORM								
+0			+0					
-0				-0		-inf		
+inf					+inf	invalid		
-inf				-inf	invalid	-inf		
qNaN							qNaN	
sNaN								invalid

【注】 FPSCR.DN=1 时，将 DENORM 作为 0 来处理。
FPSCR.DN=0时， DENORM的计算与NORM相同。

(4) 有可能发生的异常与发生上溢 / 下溢异常陷阱的条件

- FPU错误
- 无效运算
- 上溢

  发生上溢异常陷阱的条件
  - FPSCR.PR=0时：FRn与FRm为相同符号且至少一方的指数为H'FE
  - FPSCR.PR=1时：DRn与DRm为相同符号且至少一方的指数为H'7FE
- 下溢

  发生下溢异常陷阱的条件
  - FPSCR.PR=0时：FRn与FRm为不同的符号且双方的指数均小于等于H'18
  - FPSCR.PR=1时：DRn与DRm为不同的符号且双方的指数均小于等于H'035
- 不精确异常

10.3.3 FCMP Floating - point CoMPare 浮点指令

浮点比较

PR	格式	操作概略	操作码	执行状态	T 位	PR
1	0	FCMP/EQ FRm,FRn	FRn=FRm 时 1→T 此外 0→T	1111nnnnmmmm0100	1	1/0
2	1	FCMP/EQ DRm,DRn	DRn=DRm 时 1→T 此外 0→T	1111nnn0mmm00100	1	1/0
3	0	FCMP/GT FRm,FRn	FRn ＞ FRm 时 1→T 此外 0→T	1111nnnnmmmm0101	1	1/0
4	1	FCMP/GT DRm,DRn	DRn ＞ DRm 时 1→T 此外 0→T	1111nnn0mmm00101	1	1/0

(1) 说明

1. FPSCR.PR=0时：对FRn与FRm的内容的2个单精度浮点数进行算术比较，相等时将1保存在T位，否则保存0。
2. FPSCR.PR=1时：对DRn与DRm的内容的2个双精度浮点数进行算术比较，相等时将1保存在T位，否则保存0。
3. FPSCR.PR=0时：对FRn与FRm的内容的2个单精度浮点数进行算术比较，FRn>FRm时，将1保存在T位，否则保存0。
4. FPSCR.PR=1时：对DRn与DRm的内容的2个双精度浮点数进行算术比较，DRn>DRm时，将1保存在T位，否则保存0。

(2) 注意事项

无

(3) 操作内容

```
void FCMP_EQ(int m,n) /* FCMP/EQ  FRm,FRn */
{
     pc += 2;
     clear_cause();
     if(fcmp_chk(m,n) == INVALID) fcmp_invalid();
     else if(fcmp_chk(m,n) == EQ)T = 1;
     else T = 0;
}
void FCMP_GT(int m,n) /* FCMP/GT  FRm,FRn */
{
     pc += 2;
     clear_cause();
     if((fcmp_chk(m,n)==INVALID) || (fcmp_chk(m,n)==UO)) fcmp_invalid();
     else if(fcmp_chk(m,n)==GT)  T = 1;
else                        T = 0;
}
int fcmp_chk (int m,n)
{
     if((data_type_of(m) == sNaN) ||
         (data_type_of(n) == sNaN))        return(INVALID);
```

```
        else if((data_type_of(m) == qNaN) ||
                   (data_type_of(n) == qNaN))      return(UO);
        else switch(data_type_of(m)){
                case NORM: switch(data_type_of(n)){
                                case PINF:  return(GT); break;
                                case NINF:  return(LT); break;
                                default:     break;
                           }      break;
                case PZERO:
                case NZERO: switch(data_type_of(n)){
                                case PZERO:
                                case NZERO: return(EQ); break;
                                default:     break;
                             }      break;
              case PINF :   switch(data_type_of(n)){
                                case PINF: return(EQ);  break;
                                default:    return(LT);  break;
                             }      break;
              case NINF:    switch(data_type_of(n)){
                                case NINF:  return(EQ);  break;
                                default:     return(GT);  break;
               }      break;
             }
        if(FPSCR_PR==0) {
    if(FR[n]==FR[m]) return(EQ);
    else if(FR[n] > FR[m]) return(GT);
          else return(LT);
        }
        else {
    if(DR[n>>1] == DR[m>>1]) return(EQ);
    else if(DR[n>>1] > DR[m>>1]) return(GT);
          else return(LT);
        }
    }
    void fcmp_invalid()
    {
    set_V();
    if((FPSCR&ENABLE_V) == 0)  T = 0;
    else fpu_exception_trap();
```

FCMP 特殊情况

FCMP/EQ	FRn,DRn							
FRm,DRm	NORM	DNORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	CMP							
DNORM								
+0			EQ					
-0								
+INF					EQ			
-INF						EQ		
qNaN							!EQ	
sNaN								Invalid

【注】 DN=1 时，将非规格化数的值作为 0 来处理。

FCMP/GT	FRn,DRn							
FRm,DRm	NORM	DNORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	CMP				GT	GT		
DNORM								
+0			!GT					
-0								
+INF	!GT				!GT			
-INF	GT					!GT		
qNaN							UO	
sNaN								Invalid

【注】 DN=1 时，将非规格化数的值作为 0 来处理。

UO 意为无序。无序当作 !GT 处理。

(4) 有可能发生的异常

- 无效运算

10.3.4 FCNVDS Floating - point CoNVertDouble to Single precision 浮点指令

双精度转换为单精度

PR	格式	操作概略	操作码	执行状态	T 位
0	—	—	—	—	—
1	FCNVDS DRm,FPUL	(float)DRm →FPUL	1111mmm010111101	1	—

(1) 说明

FPSCR.PR=1 时：将 DRm 内的双精度浮点数转换为单精度浮点数，并保存在 FPUL。

如果设定为 FPSCR.enable.I，发生 FPU 异常陷阱，但与异常的发生无关。如果设定 FPSCR.enable.O/U，当实际产生了 FPU 异常源，以及满足指令固有的特殊条件时，发生 FPU 异常陷阱。将在本节末描述对应的特殊条件。

发生异常时，在 FPSCR.cause FPSCR.flag 反映正确的异常信息，且不更新 FPUL。请通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FCNVDS(int m, float *FPUL){
     case((FPSCR.PR){
          0:  undefined_operation();   /* reserved */
          1:  fcnvds(m, *FPUL);  break;  /* FCNVDS */
     }
}
void fcnvds(int m, float *FPUL)
{
     pc += 2;
     clear_cause();
     case(data_type_of(m)){
          NORM  :
          PZERO :
          NZERO :     normal_ fcnvds(m, *FPUL);  break;
          DENORM :   set_E();
          PINF  :     *FPUL = 0x7f800000; break;
          NINF  :     *FPUL = 0xff800000; break;
          qNaN  :     *FPUL = 0x7fbfffff; break;
          sNaN  :set_V();
if((FPSCR & ENABLE_V) == 0) *FPUL = 0x7fbfffff;
else fpu_exception_trap();   break;
     }
}
void normal_fcnvds(int m,float *FPUL)
{
int sign;
float abs;
union {
float f;
int l;
```

```
}dstf,tmpf;
union {
double d;
int l[2];
}dstd;
dstd.d = DR[m>>1];
if(dstd.l[1] & 0x1fffffff)) set_I();
if(FPSCR_RM == 1) dstd.l[1] &= 0xe0000000; /* round toward zero */
dstf.f = dstd.d;
check_single_exception(FPUL,dstf.f);
}
```

FCNVDS 特殊情况

DRn	+NORM	– NORM	+0	–0	+INF	– INF	qNaN	sNaN
FCNVDS(DRn FPUL)	FCNVDS	FCNVDS	+0	–0	+INF	– INF	qNaN	Invalid

【注】 DN=1 时，将非规格化数的值作为 0 来处理。

(4) 有可能发生的异常与发生上溢 / 下溢异常陷阱的条件

- FPU错误
- 无效运算
- 上溢

  发生上溢异常陷阱的条件
  - DRn的指数大于等于H'47E
- 下溢

  发生下溢异常陷阱的条件
  - DRn的指数小于等于H'380
- 不精确异常

10.3.5 FCNVSD Floating - point CoNVertSingle to Double precision 浮点指令

单精度转换为双精度

PR	格式	操作概略	操作码	执行状态	T 位
0	—	—	—	—	—
1	FCNVSD FPUL,DRn	(double)FPUL→DRn	1111nnn010101101	1	—

(1) 说明

FPSCR.PR=1 时：将 FPUL 的内容解释为单精度浮点数，并转换为双精度浮点数，将结果保存在 DRn。

(2) 注意事项

无

(3) 操作内容

```
void FCNVSD(int n, float *FPUL){
     pc += 2;
     clear_cause();
     case((FPSCR_PR){
          0:  undefined_operation();    /* reserved */
          1:  fcnvsd(n,*FPUL);  break;  /* FCNVSD */
     }
}
void fcnvsd(int n, float *FPUL)
{
     case(fpul_type(*FPUL)){
          PZERO :
          NZERO :
          PINF  :
          NINF  :DR[n>>1] = *FPUL; break;
          DENORM :set_E();break;
          qNaN  :qnan(n);break;
          sNaN  :invalid(n);break;
     }
}
int fpul_type(int *FPUL)
{
int abs;
abs = *FPUL & 0x7fffffff;
if(abs < 0x00800000){
if((FPSCR_DN==1) || (abs==0x00000000)){
if(sign_of(src) == 0)return(PZERO);
elsereturn(NZERO);
}
elsereturn(DENORM);
}
else if(abs<0x7f800000)return(NORM);
else if(abs==0x7f800000) {
if(sign_of(src) == 0)return(PINF);
elsereturn(NINF);
```

```
}
else if(abs<0x7fc00000)return(qNaN);
elsereturn(sNaN);
}
```

FCNVSD 特殊情况

FRn	+NORM	– NORM	+0	–0	+INF	– INF	qNaN	sNaN
FCNVSD(FPUL FRn)	+NORM	– NORM	+0	–0	+INF	– INF	qNaN	Invalid

【注】 DN=1 时，将非规格化数的值作为 0 来处理。

(4) 有可能发生的异常

- FPU错误
- 无效运算

10.3.6 FDIV Floating - point DIVide 浮点指令

浮点除法运算

PR	格式		操作概略	操作码	执行状态	T 位
0	FDIV	FRm,FRn	FRn/FRm→FRn	1111nnnnmmmm0011	14	—
1	FDIV	DRm,DRn	DRn/DRm→DRn	1111nnn0mmm00011	30	—

(1) 说明

- FPSCR.PR=0时
  对FRn与FRm的内容的2个单精度浮点数进行算术除法运算，将结果保存在FRn。
- FPSCR.PR=1时
  对DRn与DRm的内容的2个双精度浮点数进行算术除法运算，将结果保存在DRn。
  如果设定为FPSCR.enable.I，发生FPU异常陷阱，但与异常的发生无关。如果设定为FPSCR.enable.O/U，当实际产生FPU异常源，及满足指令固有的特殊条件时，发生FPU异常陷阱。将在本节末描述对应的特殊条件。
  发生异常时，在FPSCR.cause FPSCR.flag反映正确的异常信息，不更新FRn/DRn。请通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FDIV(int m,n)     /* FDIV FRm,FRn */
{
     pc += 2;
     clear_cause();
     if((data_type_of(m) == sNaN) || (data_type_of(n) == sNaN)) invalid(n);
     else if((data_type_of(m) == qNaN) || (data_type_of(n) == qNaN)) qnan(n);
          else switch (data_type_of(m)){
               case NORM: switch (data_type_of(n)){
                          case PINF:
                          case NINF:inf(n,sign_of(m)^sign_of(n));
                                    break;
                          case PZERO:
                          case NZERO:zero(n,sign_of(m)^sign_of(n));
                                                              break;
                          case DENORM:set_E();
                                      break;
                          default:normal_fdiv(m,n);
                                  break;
                            } break;
                  case PZERO: switch (data_type_of(n)){
                              case PZERO:
                              case NZERO:invalid(n);
                                         break;
                              case PINF:
                              case NINF:break;
                              default:  dz(n,sign_of(m)^sign_of(n));
                                        break;
                            } break;
                  case NZERO: switch (data_type_of(n)){
                  case PZERO:
                  case NZERO:invalid(n);break;
                  case PINF:inf(n,1);break;
                  case NINF:inf(n,0);break;
default                     :dz(FR[n],sign_of(m)^sign_of(n));
                             break;
                  } break;
       case DENORM:set_E();
                   break;
       case PINF :
       case NINF : switch (data_type_of(n)){
                  case DENORM:set_E();
                              break;
                  case PINF:
                  case NINF:invalid(n);
                         break;
                 default:zero(n,sign_of(m)^sign_of(n))
                         break
                 } break;
               }
       }
void normal_fdiv(int m,n)
```

```
{
union {
       float f;
       int l;
}      dstf,tmpf;
union {
       double d;
       int l[2];
}      dstd,tmpd;
union {
       int double x;
       int l[4];
}      tmpx;
       if(FPSCR_PR == 0) {
          tmpf.f = FR[n];            /* save destination value */
          dstf.f /= FR[m];           /* round toward nearest or even */
          tmpd.d = dstf.f;           /* convert single to double */
          tmpd.d *= FR[m];
          if(tmpf.f != tmpd.d) set_I();
          if((tmpf.f < tmpd.d) && (FPSCR_RM == 1)) dstf.l -= 1;
                                /* round toward zero */
          check_single_exception(&FR[n], dstf.f);
}
else {
          tmpd.d = DR[n>>1]; /* save destination value */
          dstd.d /= DR[m>>1];/* round toward nearest or even */
          tmpx.x = dstd.d; /* convert double to int double */
          tmpx.x *= DR[m>>1];
          if(tmpd.d != tmpx.x) set_I();
          if((tmpd.d < tmpx.x) && (FPSCR_RM == 1)) {
             dstd.l[1] -= 1; /* round toward zero */
             if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
          }
          check_double_exception(&DR[n>>1], dstd.d);
      }
}
```

FDIV 特殊情况

FDIV	FRn,DRn									
FRm,DRm	+NORM	– NORM	+DENORM	– DENORM	+0	–0	+inf	– inf	qNaN	sNaN
+NORM	FDIV				+0	–0	+inf	– inf		
– NORM					–0	+0	– inf	+inf		
+DENORM					+0	–0	+inf	– inf		
– DENORM				Error	-0	+0	– inf	+inf		
+0							+inf	– inf		
–0				DZ		invalid	– inf	DZ+inf		
+inf	+0	–0	+0	–0	+0	–0				
– inf	–0	+0	–0	+0	–0	+0		invalid		
qNaN									qNaN	
sNaN										invalid

【注】 FPSCR.DN=1 时，将 DENORM 作为 0 来处理。

(4) 有可能发生的异常与发生上溢 / 下溢异常陷阱的条件

- FPU错误
- 无效运算
- 被0除
- 下溢

  发生下溢异常陷阱的条件
  - FPSCR.PR=0时：FRn的指数－FRm的指数+H'7F大于等于H'FF
  - FPSCR.PR=1时：DRn的指数－DRm的指数+H'3FF大于等于H'7FF
- 下溢

  发生下溢异常陷阱的条件
  - FPSCR.PR=0时：FRn的指数－FRm的指数+H'7F小于等于H'01
  - FPSCR.PR=1时：DRn的指数－DRm的指数+H'3FF小于等于H'001
- 不精确异常

10.3.7 FIPR Floating - point Inner PRoduct 浮点指令

浮点内乘积

PR	格式	操作概略	操作码	执行状态	T 位
0	FIPR FVm,FVn	inner_product(FVm, FVn)→FR[n+3]	1111nnmm11101101	1	—
1	—	—	—	—	—

【注】 FV0 = {FR0, FR1, FR2, FR3}
FV4 = {FR4, FR5, FR6, FR7}
FV8 = {FR8, FR9, FR10, FR11}
FV12 = {FR12, FR13, FR14, FR15}

(1) 说明

- FPSCR.PR=0时

  对FVn与FVm所示的4维单精度浮点数向量进行算术内乘积，将结果保存在FR[n+3]。 FIPR指令为追求正确度、更追求高速化的指令。因此，结合FADD或FMUL使用时，结果不同。按照以下步骤执行FIPR。

1. 对各项分别进行乘法运算。结果为28位。
2. 调整这些结果。此时，舍入为30位以内。
3. 加上调整后的结果。
4. 进行规格化与舍入。

以下情况下，为特殊处理。

5. 输入值中有sNaN时，发生无效异常。
6. 乘法运算的输入值中有0与无穷大的组合时，发生无效异常。
7. 此外，输入值包括qNaN时，结果为qNaN。
8. 此外，输入值中有无穷大时
   - 如果乘法运算后的结果为2个以上无穷大，且符号不同时，发生无效异常。
   - 此外，保存正确的无穷大。
9. 输入值不包括sNaN、 qNaN、无穷大时，进行通常处理。

如果设定 FPSCR.enable.U/I 时，发生 FPU 异常陷阱，与是否发生异常无关。如果设定为 FPSCR.enable.O/U，当实际产生 FPU 异常源，及满足指令固有的特殊条件时，发生 FPU 异常陷阱。将在本节末描述对应的特殊条件。

发生异常时，在 FPSCR.cause FPSCR.flag 反映正确的异常信息，不更新 FR[n+3]。

通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FIPR(int m,n)   /* FIPR FVm,FVn */
{
         if(FPSCR_PR == 0) {
        pc += 2;
        clear_cause();
        fipr(m,n);
     }
    else  undefined_operation();
}
```

(4) 有可能发生的异常与发生上溢异常陷阱的条件

- 无效运算
- 上溢

  发生上溢异常陷阱的条件

  以下数至少一个大于等于H'FC
  - FRn的指数+FRm的指数
  - FR(n+1)的指数+FR(m+1)的指数
  - FR(n+2)的指数+FR(m+2)的指数
  - FR(n+3)的指数+FR(m+3)的指数
- 下溢
- 不精确异常

10.3.8 FLDI0 Floating - point LoaD Immediate 0.0 浮点指令

0.0 加载

PR	格式	操作概略	操作码	执行状态	T 位
0	FLDI0 FRn	0x00000000→FRn	1111nnnn10001101	1	—
1	—	—	—	—	—

(1) 说明

FPSCR.PR=0 时，将浮点数 0.0 （0x00000000）保存至 FRn。

(2) 注意事项

无

(3) 操作内容

```
void FLDI0(int n)
{
     FR[n] = 0x00000000;
     pc += 2;
}
```

(4) 有可能发生的异常

无

10.3.9 FLDI1 Floating - point LoaD Immediate 1.0 浮点指令

1.0 加载

PR	格式	操作概略	操作码	执行状态	T 位
0	FLDI1 FRn	0x3F800000→FRn	1111nnnn10011101	1	—
1	—	—	—	—	—

(1) 说明

FPSCR.PR=0 时，将浮点数 1.0 （0x3F800000）保存至 FRn。

(2) 注意事项

无

(3) 操作内容

```
void FLDI1(int n)
{
     FR[n] = 0x3F800000;
     pc += 2;
}
```

(4) 有可能发生的异常

无

10.3.10 FLDS Floating - point LoaD toSystem register 浮点指令

向系统寄存器传送

格式	操作概略	操作码	执行状态	T 位
FLDS FRm,FPUL	FRm → FPUL	1111mmmm00011101	1	—

(1) 说明

将浮点寄存器 FRm 的内容保存至系统寄存器 FPUL。

(2) 注意事项

无

(3) 操作内容

```
void FLDS(int m,float *FPUL)
{
     *FPUL = FR[m];
     pc += 2;
}
```

(4) 有可能发生的异常

无

10.3.11 FLOAT Floating - point convert from integer 浮点指令

整数转换浮点数

PR	格式	操作概略	操作码	执行状态	T 位
0	FLOAT FPUL,FRn	(float)FPUL→FRn	1111nnnn00101101	1	—
1	FLOAT FPUL,DRn	(double)FPUL→DRn	1111nnn000101101	1	—

(1) 说明

FPSCR.PR=0 时：把 FPUL 的内容看作 32bit 整数，将其转换为单精度浮点数，并将结果保存在 FRn。

FPSCR.PR=1 时：把 FPUL 的内容看作 32bit 整数，将其转换为双精度浮点数，并将结果保存在 DRn。

FPSCR.enable.I=1 且 FPSCR.PR=0 时，发生 FPU 异常陷阱，但与异常的发生无关。发生异常时，在 FPSCR.cause FPSCR.flag 反映正确的异常信息，不更新 FRn。请通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FLOAT(int n,float *FPUL)
{
union {
      double d;
      int l[2];
}     tmp;
      pc += 2;
      clear_cause();
      if(FPSCR.PR==0){
        FR[n] = *FPUL; /* convert from integer to float */
        tmp.d = *FPUL;
        if(tmp.l[1] & 0x1fffffff) inexact();
      } else {
        DR[n>>1] = *FPUL; /* convert from integer to double */
      }
}
```

(4) 有可能发生的异常

- 不精确异常：FPSCR.PR = 1时，不发生。

10.3.12 FMAC Floating - point Multiply and Accumulate 浮点指令

浮点乘加

PR	格式	操作概略	操作码	执行状态	T 位
0	FMAC FR0,FRm,FRn	FR0×FRm+FRn→FRn	1111nnnnmmmm1110	1	—
1	—	—	—	—	—

(1) 说明

FPSCR.PR=0 时：对 FR0 与 FRm 的内容的 2 个单精度浮点数进行算术乘法运算，将 FRn 的内容进行算数加法运算，并将结果保存在 FRn。

如果设定为 FPSCR.enable.I，发生 FPU 异常陷阱，但与异常的发生无关。如果设定为 FPSCR.enable.O/U，当实际产生 FPU 异常源，及满足指令固有的特殊条件时，发生 FPU 异常陷阱。将在本节末描述对应的特殊条件。

发生异常时，在 FPSCR.cause FPSCR.flag 反映正确的异常信息，不更新 FRn。请通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FMAC(int m,n)
{
      pc += 2;
      clear_cause();
      if(FPSCR_PR == 1) undefined_operation();
      else if((data_type_of(0) == sNaN) ||
              (data_type_of(m) == sNaN) ||
              (data_type_of(n) == sNaN)) invalid(n);
      else if((data_type_of(0) == qNaN) ||
              (data_type_of(m) == qNaN)) qnan(n);
      else if((data_type_of(0) == DENORM) ||
              (data_type_of(m) == DENORM)) set_E();
else switch (data_type_of(0)){
       case NORM: switch (data_type_of(m)){
       case PZERO:
       case NZERO: switch (data_type_of(n)){
      case DENORM:set_E();break;
      case qNaN:  qnan(n);break;
      case PZERO:
      case NZERO: zero(n,sign_of(0)^ sign_of(m)^sign_of(n));  break;
      default:    break;
      }
  case PINF:
case NINF: switch (data_type_of(n)){
      case DENORM:set_E();break;
      case qNaN:qnan(n);break;
      case PINF:
      case NINF:if(sign_of(0)^ sign_of(m)^sign_of(n))invalid(n);
```

```
                else inf(n,sign_of(0)^ sign_of(m));break;
        default:       inf(n,sign_of(0)^ sign_of(m));break;
        }
    case NORM: switch (data_type_of(n)){
                case DENORM:set_E();          break;
                case qNaN:  qnan(n);break;
                case PINF:
                case NINF:  inf(n,sign_of(n));break;
                case PZERO:
                case NZERO:
                case NORM:normal_fmac(m,n);break;
              }   break;
              case PZERO:
              case NZERO: switch (data_type_of(m)){
                             case PINF:
                             case NINF:  invalid(n); break;
                             case PZERO:
                             case NZERO:
                             case NORM: switch (data_type_of(n)){
                             case DENORM: set_E();    break;
                             case qNaN:   qnan(n);    break;
                             case PZERO:
                case NZERO: zero(n,sign_of(0)^ sign_of(m)^sign_of(n));  break;
                             default:    break;
                             }      break;
        }      break;
               case PINF :
               case NINF : switch (data_type_of(m)){
                     case PZERO:
                     case NZERO:invalid(n);break;
                     default: switch (data_type_of(n)){
                     case DENORM:set_E();break;
                     case qNaN:qnan(n);break;
                     default:  inf(n,sign_of(0)^sign_of(m)^sign_of(n));break
                     }      break;
               }   break;
        }
    }
    void normal_fmac(int m,n)
    {
    union {
          int double x;
          int l[4];
          }dstx,tmpx;
    float dstf,srcf;
          if((data_type_of(n) == PZERO)|| (data_type_of(n) == NZERO))
                 srcf = 0.0; /* flush denormalized value */
          else srcf = FR[n];
          tmpx.x = FR[0]; /* convert single to int double */
          tmpx.x *= FR[m]; /* exact product */
          dstx.x = tmpx.x + srcf;
          if(((dstx.x == srcf) && (tmpx.x != 0.0)) ||
           ((dstx.x == tmpx.x) && (srcf != 0.0))) {
```

```
        set_I();
        if(sign_of(0)^ sign_of(m)^ sign_of(n))  {
           dstx.l[3] -= 1; /* correct result */
           if(dstx.l[3] == 0xffffffff) dstx.l[2] -= 1;
           if(dstx.l[2] == 0xffffffff) dstx.l[1] -= 1;
           if(dstx.l[1] == 0xffffffff) dstx.l[0] -= 1;
      }
      else  dstx.l[3] |= 1;
}
if((dstx.l[1] & 0x01ffffff) || dstx.l[2] || dstx.l[3]) set_I();
if(FPSCR_RM == 1) {
          dstx.l[1] &= 0xfe000000; /* round toward zero */
          dstx.l[2]  = 0x00000000;
          dstx.l[3]  = 0x00000000;
     }
     dstf = dstx.x;
     check_single_exception(&FR[n],dstf);
}
```

FMAC 特殊情况

FMAC		FRm							
FRn	FR0	+NORM	– NORM	+0	–0	+inf	– inf	qNaN	sNaN
NORM	+NORM	FMAC				+inf	– inf		
	– NORM					– inf	+inf		
	+0					invalid			
	–0								
	+inf	+inf	-inf	invalid		+inf	– inf		
	– inf	-inf	+inf			– inf	+inf		
+0	+NORM	FMAC		+0		+inf	– inf		
	– NORM					– inf	+inf		
	+0	+0				invalid			
	–0								
	+inf	+inf	-inf	invalid		+inf	– inf		
	– inf	-inf	+inf			– inf	+inf		
-0	+NORM	FMAC		+0	-0	+inf	– inf		
	– NORM			-0	+0	– inf	+inf		
	+0	+0	-0	+0	-0	invalid			
	-0	-0	+0	-0	+0				
	+inf	+inf	– inf	invalid		+inf	– inf		
	– inf	– inf	+inf			– inf	+inf		
+inf	+NORM	+inf				+inf	invalid		
	– NORM					invalid	+inf		
	+0					invalid			
	–0								
	+inf	+inf	invalid	invalid		+inf	invalid		
	– inf	invalid	+inf			invalid	+inf		
-inf	+NORM	– inf				invalid	– inf		
	– NORM					– inf	invalid		
	+0					invalid			
	-0								
	+inf	invalid	– inf	invalid		invalid	– inf		
	– inf	– inf	invalid			– inf	invalid		
qNaN	+NORM								
	– NORM								
	+0					invalid			
	-0								
	+inf			invalid					
	– inf								
!sNaN	qNaN							qNaN	
all types	sNaN								
sNaN	all types								invalid

【注】 FPSCR.DN=1 时，将 DENORM 作为 0 来处理。
FPSCR.DN=0时， DENORM的计算与NORM相同。

(4) 有可能发生的异常与发生上溢 / 下溢异常陷阱的条件

- FPU 错误
- 无效运算
- 上溢
  发生上溢异常陷阱的条件
  以下数至少一方大于等于 H'FD
  - FR0 的指数 +FRm 的指数
  - FRn 的指数
- 下溢
  发生下溢异常陷阱的条件
  - 以下数至少一方小于等于 H'2E
  - FR0 的指数 +FRm 的指数
  - FRn 的指数
- 不精确异常

10.3.13 FMOV Floating - point MOVe 浮点指令

浮点传送

No.	SZ	格式	操作概略	操作码	执行状态	T 位
1	0	FMOV FRm,FRn	FRm→FRn	1111nnnnmmmm1100	1	—
2	1	FMOV DRm,DRn	DRm→DRn	1111nnn0mmm01100	1	—
3	0	FMOV.S FRm,@Rn	FRm→(Rn)	1111nnnnmmmm1010	1	—
4	1	FMOV DRm,@Rn	DRm→(Rn)	1111nnnnmmm01010	1	—
5	0	FMOV.S @Rm,FRn	(Rm)→FRn	1111nnnnmmmm1000	1	—
6	1	FMOV @Rm,DRn	(Rm)→DRn	1111nnn0mmmm1000	1	—
7	0	FMOV.S @Rm+,FRn	(Rm)→FRn,Rm+4→Rm	1111nnnnmmmm1001	1	—
8	1	FMOV @Rm+,DRn	(Rm)→DRn,Rm+8→Rm	1111nnn0mmmm1001	1	—
9	0	FMOV.S Rm,@-Rn	Rn-4→Rn,FRm→(Rn)	1111nnnnmmmm1011	1	—
10	1	FMOV DRm,@-Rn	Rn-8→Rn,DRm→(Rn)	1111nnnnmmm01011	1	—
11	0	FMOV.S @(R0,Rm),FRn	(R0+Rm)→FRn	1111nnnnmmmm0110	1	—
12	1	FMOV @(R0,Rm),DRn	(R0+Rm)→DRn	1111nnn0mmmm0110	1	—
13	0	FMOV.S FRm, @(R0,Rn)	FRm→(R0+Rn)	1111nnnnmmmm0111	1	—
14	1	FMOV DRm, @(R0,Rn)	DRm→(R0+Rn)	1111nnnnmmm00111	1	—

(1) 说明

1. 将FRm的内容传送至FRn。
2. 将DRm的内容传送至DRn。
3. 将FRm的内容传送至Rn所示地址的存储器。
4. 将DRm的内容传送至Rn所示地址的存储器。
5. 将Rm所示地址的存储器的内容传送至FRn。
6. 将Rm所示地址的存储器的内容传送至DRn。
7. 将Rm所示地址的存储器的内容传送至FRn，并给Rm加4。
8. 将Rm所示地址的存储器的内容传送至DRn。并给Rm加8。
9. 从Rn减4，并将FRm的内容传送至Rn所示地址的存储器。
10. 从Rn减8，将DRm的内容传送至Rn所示地址的存储器。
11. 将 （R0+Rm）所示地址的存储器的内容传送至FRn。
12. 将 （R0+Rm）所示地址的存储器的内容传送至DRn。
13. 将FRm的内容传送至 （R0+Rn）所示地址的存储器。
14. 将DRm的内容传送至 （R0+Rn）所示地址的存储器。

(2) 注意事项

无

(3) 操作内容

```
void FMOV(int m,n)                  /* FMOV FRm,FRn */
{
     FR[n] = FR[m];
     pc += 2;
}
void FMOV_DR(int m,n)            /* FMOV DRm,DRn */
{
     DR[n>>1] = DR[m>>1];
     pc += 2;
}
void FMOV_STORE(int m,n)        /* FMOV.S FRm,@Rn */
{
     store_int(FR[m],R[n]);
     pc += 2;
}
void FMOV_STORE_DR(int m,n)   /* FMOV DRm,@Rn */
{
     store_quad(DR[m>>1],R[n]);
     pc += 2;
}
 void FMOV_LOAD(int m,n)        /* FMOV.S @Rm,FRn */
{
     load_int(R[m],FR[n]);
     pc += 2;
}
void FMOV_LOAD_DR(int m,n)    /* FMOV @Rm,DRn */
{
     load_quad(R[m],DR[n>>1]);
     pc += 2;
}
void FMOV_RESTORE(int m,n)     /* FMOV.S @Rm+,FRn */
{
     load_int(R[m],FR[n]);
     R[m] += 4;
     pc += 2;
}
void FMOV_RESTORE_DR(int m,n) /* FMOV @Rm+,DRn */
{
     load_quad(R[m],DR[n>>1]) ;
     R[m] += 8;
     pc += 2;
}
void FMOV_SAVE(int m,n)         /* FMOV.S FRm,@-Rn */
{
     store_int(FR[m],R[n]-4);
     R[n] -= 4;
     pc += 2;
}
```

```
void FMOV_SAVE_DR(int m,n)    /* FMOV DRm,@-Rn */
{
     store_quad(DR[m>>1],R[n]-8);
     R[n] -= 8;
     pc += 2;
}
void FMOV_INDEX_LOAD(int m,n)  /* FMOV.S @(R0,Rm),FRn */
{
     load_int(R[0] + R[m],FR[n]);
     pc += 2;
}
void FMOV_INDEX_LOAD_DR(int m,n) /*FMOV @(R0,Rm),DRn */
{
     load_quad(R[0] + R[m],DR[n>>1]);
     pc += 2;
}
void FMOV_INDEX_STORE(int m,n)  /*FMOV.S FRm,@(R0,Rn)*/
{
     store_int(FR[m],R[0]+R[n]);
     pc += 2;
}
void FMOV_INDEX_STORE_DR(int m,n)/*FMOV DRm,@(R0,Rn)*/
{
     store_quad(DR[m>>1],R[0]+R[n]);
     pc += 2;
}
```

(4) 有可能发生的异常

- 数据TLB未命中异常
- 数据保护违反异常
- 初始页写入异常
- 数据地址错误

10.3.14 FMOV Floating - point MOVe extension 浮点指令

浮点传送

No.	SZ	格式		操作概略	操作码	执行状态	T 位
1	1	FMOV	XDm,@Rn	XRm→(Rn)	1111nnnnmmm11010	1	—
2	1	FMOV	@Rm,XDn	(Rm)→XDn	1111nnn1mmmm1000	1	—
3	1	FMOV	@Rm+,XDn	(Rm)→XDn,Rm+8→Rm	1111nnn1mmmm1001	1	—
4	1	FMOV	XDm,@-Rn	Rn-8→ Rn,XDm→(Rn)	1111nnnnmmm11011	1	—
5	1	FMOV	@(R0,Rm),XDn	(R0+Rm)→XDn	1111nnn1mmmm0110	1	—
6	1	FMOV	XDm,@(R0,Rn)	XDm→(R0+Rn)	1111nnnnmmm10111	1	—
7	1	FMOV	XDm,XDn	XDm→XDn	1111nnn1mmm11100	1	—
8	1	FMOV	XDm,DRn	XDm→DRn	1111nnn0mmm11100	1	—
9	1	FMOV	DRm,XDn	DRm→XDn	1111nnn1mmm01100	1	—

(1) 说明

1. 将XDm的内容传送至Rn所示地址的存储器。
2. 将Rm所示地址的存储器的内容传送至XDn。
3. 将Rm所示地址的存储器的内容传送至XDn，并给Rm加上8。
4. 从Rn减去8，并将XDm的内容传送至Rn所示地址的存储器。
5. 将(R0+Rm)所示地址的存储器的内容传送至XDn。
6. 将XDm的内容传送至(R0+Rn)所示地址的存储器。
7. 将XDm的内容传送至XDn。
8. 将XDm的内容传送至DRn。
9. 将DRm的内容传送至XDn。

(2) 注意事项

无

(3) 操作内容

```
void FMOV_STORE_XD(int m,n)      /* FMOV XDm,@Rn */
{
     store_quad(XD[m>>1],R[n]);
     pc += 2;
}
void FMOV_LOAD_XD(int m,n)      /* FMOV @Rm,XDn */
{
     load_quad(R[m],XD[n>>1]);
     pc += 2;
}
void FMOV_RESTORE_XD(int m,n)   /* FMOV @Rm+,XDn */
{
     load_quad(R[m],XD[n>>1]);
     R[m] += 8;
     pc += 2;
}
void FMOV_SAVE_XD(int m,n)      /* FMOV XDm,@-Rn */
```

```
{
     store_quad(XD[m>>1],R[n]-8);
     R[n] -= 8;
     pc += 2;
}
void FMOV_INDEX_LOAD_XD(int m,n)/* FMOV @(R0,Rm),XDn */
{
     load_quad(R[0] + R[m],XD[n>>1]);
     pc += 2;
}
void FMOV_INDEX_STORE_XD(int m,n)/* FMOV XDm,@(R0,Rn) */
{
     store_quad(XD[m>>1], R[0] + R[n]);
     pc += 2;
}
 void FMOV_XDXD(int m,n)/* FMOV XDm,XDn */
{
     XD[n>>1] = XD[m>>1];
     pc += 2;
}
void FMOV_XDDR(int m,n)/* FMOV XDm,DRn */
{
     DR[n>>1] = XD[m>>1];
     pc += 2;
}
void FMOV_DRXD(int m,n)/* FMOV DRm,XDn */
{
     XD[n>>1] = DR[m>>1];
     pc += 2;
}
```

(4) 有可能发生的异常

- 数据TLB未命中异常
- 数据保护违反异常
- 初始页写入异常
- 数据地址错误

10.3.15 FMUL Floating - point MULtiply 浮点指令

浮点乘法运算

PR	格式	操作概略	操作码	执行状态	T 位
0	FMUL FRm,FRn	FRn×FRm→FRn	1111nnnnmmmm0010	1	—
1	FMUL DRm,DRn	DRn×DRm→DRn	1111nnn0mmm00010	3	—

(1) 说明

- FPSCR.PR=0时
  对FRn与FRm的内容的2个单精度浮点数进行算术乘法运算，将结果保存在FRn。
- FPSCR.PR=1时
  对DRn与DRm的内容的2个双精度浮点数进行算术乘法运算，将结果保存在DRn。

如果设定为 FPSCR.enable.I，发生 FPU 异常陷阱，但与异常的发生无关。如果设定为 FPSCR.enable.O/U，当实际产生 FPU 异常源，及满足指令固有的特殊条件时，发生 FPU 异常陷阱。将在本节末描述对应的特殊条件。

发生异常时，在 FPSCR.cause FPSCR.flag 反映正确的异常信息，不更新 FRn/DRn。请通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FMUL(int m,n)
{
    pc += 2;
      clear_cause();
      if((data_type_of(m) == sNaN) ||
         (data_type_of(n) == sNaN)) invalid(n);
      else if((data_type_of(m) == qNaN) ||
              (data_type_of(n) == qNaN)) qnan(n);
      else if((data_type_of(m) == DENORM) ||
              (data_type_of(n) == DENORM)) set_E();
      else switch (data_type_of(m){
           case NORM: switch (data_type_of(n)){
           case PZERO:
           case NZERO:zero(n,sign_of(m)^sign_of(n));  break;
           case PINF:
           case NINFinf(n,sign_of(m)^sign_of(n));  break;
           default:normal_fmul(m,n);break;
  }        break;
     case PZERO:
     case NZERO: switch (data_type_of(n)){
           case PINF:
           case NINF:invalid(n);break;
           default:  zero(n,sign_of(m)^sign_of(n));break;
       }      break;
           case PINF :
           case NINF : switch (data_type_of(n)){
```

```
            case PZERO:
            case NZERO:invalid(n);break;
            default:  inf(n,sign_of(m)^sign_of(n));break
        }      break;
    }
    }
```

FMUL 特殊情况 （FPSCR.PR=0 时）

FMUL	FRn							
FRm	+NORM	– NORM	+0	–0	+inf	– inf	qNaN	sNaN
+NORM	FMUL		+0	–0	+inf	– inf		
– NORM			–0	+0	– inf	+inf		
+0	+0	–0	+0	–0	invalid			
–0	–0	+0	–0	+0				
+inf	+inf	– inf	invalid		+inf	– inf		
– inf	– inf	+inf			– inf	+inf		
qNaN	qNaN							
sNaN	invalid							

【注】 FPSCR.DN=0 时， DENORM 的计算与 NORM 相同。

FMUL 特殊情况 （FPSCR.PR=1 时）

FMUL	DRn									
DRm	+NORM	– NORM	+DENORM	– DENORM	+0	– 0	+inf	– inf	qNaN	sNaN
+NORM	FMUL		Error		+0	– 0	+inf	– inf		
– NORM					– 0	+0	– inf	+inf		
+DENORM					+0	– 0	+inf	– inf		
– DENORM					– 0	+0	– inf	+inf		
+0	+0	– 0	+0	– 0	+0	– 0	invalid			
– 0	– 0	+0	– 0	+0	– 0	+0				
+inf	+inf	– inf	+inf	– inf	invalid		+inf	– inf		
– inf	– inf	+inf	– inf	+inf			– inf	+inf		
qNaN	qNaN									
sNaN	invalid									

【注】 FPSCR.DN=1 时，将 DENORM 作为 0 来处理。

(4) 有可能发生的异常与发生上溢 / 下溢异常陷阱的条件

- FPU错误
- 无效运算
- 上溢

  发生上溢异常陷阱的条件
  - FPSCR.PR=0时：FRn的指数+FRm的指数－H'7F大于等于H'FE
  - FPSCR.PR=1时：DRn的指数+DRm的指数－H'3FF大于等于H'7FE
- 下溢

  发生下溢异常陷阱的条件
  - FPSCR.PR=0时

    FRn、 FRm均为规格化数时：FRn的指数+FRm的指数－H'7F小于等于H'00

    FRn、 FRm中至少一方为非规格化数时：FRn的指数+FRm的指数－H'7F小于等于H'18
  - FPSCR.PR=1时：DRn的指数+DRm的指数－H'3FF小于等于H'000
- 不精确异常

10.3.16 FNEG Floating - point NEGate value 浮点指令

取反浮点符号

PR	格式	操作概略	操作码	执行状态	T 位
0	FNEG FRn	FRn ^ H'80000000→FRn	1111nnnn01001101	1	—
1	FNEG DRn	DRn ^ H'8000000000000000→DRn	1111nnn001001101	1	—

(1) 说明

取反浮点寄存器 FRn/DRn 的内容的最高位 （符号位），将结果保存在 FRn/DRn。
不更新 FPSCR 的 cause/flag 部分。

(2) 注意事项

无

(3) 操作内容

```
void FNEG (int n){
     FR[n] = -FR[n];
     pc += 2;
}

/* 不取决于精度，并进行相同操作。 */
```

(4) 有可能发生的异常

无

10.3.17 FPCHG Pr-bit ChanGe 浮点指令

取反 PR 位

格式	操作概略	操作码	执行状态	T 位
FPCHG	～ FPSCR.PR → FPSCR.PR	1111011111111101	1	—

(1) 说明

取反浮点状态寄存器 FPSCR 的 PR 位。转换 FPSCR 的 PR 位时，可切换单精度运算与双精度运算。

(2) 注意事项

无

(3) 操作内容

```
void FPCHG(){/* FPCHG */}
{
          FPSCR ^= 0x00080000; /* bit 19 */
          PC += 2;
}
```

(4) 有可能发生的异常

无

10.3.18 FRCHG FR-bit CHanGe 浮点指令

取反 FR 位

PR	格式	操作概略	操作码	执行状态	T 位
0	FRCHG	～ FRSCR.FR→FRSCR.FR	1111101111111101	1	—
1	—	—	—	—	—

(1) 说明

取反浮点状态寄存器 FPSCR 的 FR 位。转换 FPSCR 的 FR 位时， FPR0_BANK0 ～ FPR15_BANK0 及 FPR0_BANK1 ～ FPR15_BANK1 中， FR0 ～ FR15 为 XR0 ～ XR15， XR0 ～ XR15 为 FR0 ～ FR15。 FPSCR.FR=0 时， FPR0_BANK0 ～ FPR15_BANK0 与 FR0 ～ FR15 对应， FPR0_BANK1 ～ FPR15_BANK1 与 XR0 ～ XR15 对应。 FPSCR.FR=1 时， FPR0_BANK1 ～ FPR15_BANK1 与 FR0 ～ FR15 对应， FPR0_BANK0 ～ FPR15_BANK0 与 XR0 ～ XR15 对应。

(2) 注意事项

无

(3) 操作内容

```
void FRCHG()   /* FRCHG */
{
          if(FPSCR_PR == 0){
              FPSCR ^= 0x00200000; /* bit 21 */
    PC += 2;
   }
   else undefined_operation();
}
```

(4) 有可能发生的异常

无

10.3.19 FSCA Floating Point Sine And Cosine Approximate 浮点指令

PR	格式	操作概略	操作码	执行状态	T 位
0	FSCA FPUL,DRn	sin(FPUL)→FRn cos(FPUL)→FR[n+1]	1111nnn011111101	3	—
1	—	reserved	1111nnnn11111101	—	—

(1) 说明

作为单精度浮点来计算 FPUL 所示角度的 sin 及 cos 的近似值 （绝对误差在 ±2 ^ -21 以内），并将 sin 值写入 FRn，将 cos 值写入 FR[n+1]。因为是近似运算指令，所以总是发生不精确异常 （即使输入为 0，结果还是不精确）。

如果设定了为 FPSCR.enable.I，发生 FPU 异常陷阱。发生异常时，在 FPSCR.cause、 FPSCR.flag 反映正确的异常信息，不更新 FRn/FR[n+1]。请通过软件进行适当的处理。

(2) 注意事项

无

(3) 操作内容

```
void FSCA(int n)
{
      float angle;
      long offset = 0x00010000;
      long fraction = 0x0000ffff;
       case(FPSCR_PR){
              0: clear_cause ();
                        set_I();
  /* extract sub-rotation (fraction) part */
                    fraction &= FPUL;
  /* convert to float */
                    angle = fraction;
  /* convert to radian */
                    angle = 2*M_PI*angle / offset;
                     FR[n] ~= sin(angle);
                     FR[n+1] ~= cos(angle);
                     pc += 2; break;
       1: undefined_operation();     /* reserved */
    }
}
```

(4) 源操作数的数据格式：角度的指定方法

以作为 2 的补码的带符号小数点数所表示的旋转数来指定角度。 sin/cos 的结果为单精度浮点数。

0x7FFF FFFF ～ 0x0000 0001 ：$360 \times 2^{15} - 360/2^{16}$ ～ $360/2^{16}$ 度
0x0000 0000 ：0 度
0xFFFF FFFF ～ 0x8000 0000 ：$-360/2^{16}$ ～ -360×2^{15} 度

(5) 有可能发生的异常

- 不精确异常

10.3.20 FSCHG Sz-bit CHanGe 浮点指令

取反 SZ 位

PR	格式	操作概略	操作码	执行状态	T 位
0	FSCHG	～ FPSCR.SZ→FPSCR.SZ	1111001111111101	1	—

(1) 说明

取反浮点状态寄存器 FPSCR 的 SZ 位。转换 FPSCR 的 SZ 位时，将 FMOV 指令的数据传送切换为 1 个或 2 个单精度数据。 FPSCR.SZ=0 时， FMOV 指令传送 1 个单精度数据。 FPSCR.SZ=1 时， FMOV 指令成对传送 2 个单精度数据。

(2) 注意事项

无

(3) 操作内容

```
void FSCHG()   /* FSCHG */
{
     if(FPSCR_PR == 0){
       FPSCR ^= 0x00100000; /* bit 20 */
       PC += 2;
    }
    else undefined_operation();
}
```

(4) 有可能发生的异常

无

10.3.21 FSQRT Floating - point SQuare RooT 浮点指令

浮点平方根

PR	格式	操作概略	操作码	执行状态	T 位
0	FSQRT FRn	sqrt(FRn)*→FRn	1111nnnn01101101	14	—
1	FSQRT DRn	sqrt(DRn)*→DRn	1111nnn001101101	30	—

【注】 * sqrt(FRn) 表示 FRn 的平方根， sqrt(DRn) 表示 DRn 的平方根。

(1) 说明

- FPSCR.PR=0时
  计算FRn的内容的单精度浮点数的算术平方根，并将结果保存在FRn。
- FPSCR.PR=1时
  计算DRn的内容的双精度浮点数的算术平方根，并将结果保存在DRn。

如果设定了为 FPSCR.enable.I，发生 FPU 异常陷阱，但与异常的发生无关。发生异常时，在 FPSCR.cause FPSCR.flag 反映正确的异常信息，不更新 FRn/DRn。请通过软件进行适当处理。

(2) 注意事项

无

(3) 操作内容

```
void FSQRT(int n){
     pc += 2;
     clear_cause();
     switch(data_type_of(n)){
          case NORM  :if(sign_of(n) == 0) normal_fsqrt(n);
                      else     invalid(n); break;
          case DENORM:if(sign_of(n) == 0) set_E();
                      else     invalid(n); break;
             case PZERO :
             case NZERO :
             case PINF  :break;
             case NINF  :invalid(n); break;
             case qNaN  :qnan(n); break;
             case sNaN  :invalid(n); break;
    }
}
void normal_fsqrt(int n)
{
union {
       float f;
       int l;
}      dstf,tmpf;
union {
       double d;
```

```
        int l[2];
    }   dstd,tmpd;
    union {
        int double x;
        int l[4];
    }   tmpx;
        if(FPSCR_PR == 0) {
           tmpf.f = FR[n]; /* save destination value */
           dstf.f = sqrt(FR[n]); /* round toward nearest or even */
           tmpd.d = dstf.f; /* convert single to double */
           tmpd.d *= dstf.f;
           if(tmpf.f != tmpd.d) set_I();
           if((tmpf.f < tmpd.d) && (FPSCR_RM == 1))
              dstf.l -= 1; /* round toward zero */
           if(FPSCR & ENABLE_I)fpu_exception_trap();
           else                      FR[n] = dstf.f;
         } else {
           tmpd.d = DR[n>>1]; /* save destination value */
           dstd.d = sqrt(DR[n>>1]); /* round toward nearest or even */
           tmpx.x = dstd.d; /* convert double to int double */
           tmpx.x *= dstd.d;
           if(tmpd.d != tmpx.x) set_I();
           if((tmpd.d < tmpx.x) && (FPSCR_RM == 1)) {
               dstd.l[1] -= 1; /* round toward zero */
               if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
           }
           if(FPSCR & ENABLE_I)fpu_exception_trap();
           else              DR[n>>1] = dstd.d;
       }
    }
```

FSQRT 特殊情况

FRn	+NORM	– NORM	+DENORM	– DENORM	+0	– 0	+INF	– INF	qNaN	sNaN
FSQRT(FRn)	SQRT	Invalid	Error	Error	+0	– 0	+INF	Invalid	qNaN	Invalid

【注】 DN=1 时，将非规格化数的值作为 0 来处理。

(4) 有可能发生的异常

- FPU错误
- 无效运算
- 不精确异常

10.3.22 FSRRA Floating Ponit Square Reciprocal Approximate 浮点指令

PR	格式	操作概略	操作码	执行状态	T 位
0	FSRRA FRn	1/sqrt(FRn)*→FRn	1111nnnn01111101	1	—
1	—	reserved	1111nnnn01111101		

【注】 * sqrt(FRn) 表示 FRn 的平方根。

(1) 说明

将作为 FRn 内容的单精度浮点的算术平方根的倒数的近似值 （绝对误差在 ±2 ^ -21 以内）写入 FRn。因为是近似值运算指令，所以输入为规格化数时，发生不精确异常。其他情况下，不发生不精确异常。

如果设定了为 FPSCR.enable.I，发生 FPU 异常陷阱。发生异常时，在 FPSCR.cause、 FPSCR.flag 反映正确的异常信息，不更新 FRn/DRn。请通过软件进行适当处理。

(2) 注意事项

无

(3) 操作内容

```
vaid FSRRA(int n){
     case(FPSCR_PR){
         0:fsrra_single(n);break;
         1:undefined_operation(); break;
     }
     PC += 2;
}
fsrra_single(int n)
{    clear_cause();
     case(data_type_of(n)){
          NORM:  if(sign_of(n)==0) {
                    Set_I();
                    FR[n] ~= 1/sqrt(FR[n]);
          }
          else invalid(n);     break;
       DENORM:if(sign_of(n)==0)
          fpu_error();         break;
          else invalid(n);     break;
       PZERO:
             NZERO:dz(n,sign_of(n));     break;
             PINF:FR[n]=0;               break;
             NINF:invalid(n);            break;
```

```
                qNAN:qnan(n);       break;
                sNANinvalid(n);     break;
        }
}
```

FSRRA 特殊情况

FRn	+NORM	– NORM	+DENORM	– DENORM	+0	–0	+INF	– INF	qNaN	sNaN
FSRRA(FRn)	1/SQRT	Invalid	Error	Invalid	DZ	DZ	+0	Invalid	qNaN	Invalid

【注】 DN=1 时，将非规格化数的值作为 0 来处理。

(4) 有可能发生的异常

- FPU错误
- 无效运算
- 被0除
- 不精确异常

10.3.23 FSTS Floating - point StoreSystem register 浮点指令

从系统寄存器传送

格式	操作概略	操作码	执行状态	T 位
FSTS FPUL,FRn	FPUL→FRn	1111nnnn00001101	1	—

(1) 说明

将系统寄存器 FPUL 的内容传送至浮点寄存器 FRn。

(2) 注意事项

无

(3) 操作内容

```
void FSTS(int n, float *FPUL)
{
     FR[n] = *FPUL;
     pc += 2;
}
```

(4) 有可能发生的异常

无

10.3.24 FSUB Floating - point SUBtract 浮点指令

浮点减法运算

PR	格式	操作概略	操作码	执行状态	T 位
0	FSUB FRm,FRn	FRn-FRm→FRn	1111nnnnmmmm0001	1	—
1	FSUB DRm,DRn	DRn-DRm→DRn	1111nnn0mmm00001		

(1) 说明

- FPSCR.PR=0时
  对FRn与FRm的内容的2个单精度浮点数进行算术减法运算，将结果保存在FRn。
- FPSCR.PR=1时
  对DRn与DRm的内容的2个双精度浮点数进行算术减法运算，将结果保存在DRn。

如果设定为 FPSCR.enable.I，发生 FPU 异常陷阱，但与异常的发生无关。如果设定为 FPSCR.enable.O/U，当实际产生 FPU 异常源，及满足指令固有的特殊条件时，发生 FPU 异常陷阱。将在本节末描述对应的特殊条件。

发生异常时，在 FPSCR.cause FPSCR.flag 反映正确的异常信息，不更新 FRn/DRn。请通过软件进行适当处理。

(2) 注意事项

无

(3) 操作内容

```
void FSUB (int m,n)
{
     pc += 2;
     clear_cause();
     if((data_type_of(m) == sNaN) ||
        (data_type_of(n) == sNaN)) invalid(n);
     else if((data_type_of(m) == qNaN) ||
             (data_type_of(n) == qNaN)) qnan(n);
     else if((data_type_of(m) == DENORM) ||
             (data_type_of(n) == DENORM)) set_E();
     else switch (data_type_of(m)){
          case NORM: switch pe_of(n)){
          case NORM:normal_faddsub(m,n,SUB); break;
          case PZERO:
          case NZERO:register_copy(m,n); FR[n] = -FR[n];break;
          default:break;
     }            break;
     case PZERO:break;
     case NZERO: switch (data_type_of(n)){
       case NZERO:zero(n,0); break;
       default:   break;
     }            break;
     case PINF: switch (data_type_of(n)){
          case PINF:invalid(n);     break;
          default:  inf(n,1);       break;
```

```
        }          break;
        case NINF: switch (data_type_of(n)){
          case NINF:     invalid(n);     break;
          default:inf(n,0);              break;
            }       break;
        }
    }
```

FSUB 特殊情况

FSUB	FRn,DRn							
FRm,DRm	+NORM	– NORM	+0	– 0	+inf	– inf	qNaN	sNaN
+NORM	FSUB							
– NORM								
+0			+0	– 0				
– 0					+inf			
+inf	– inf				invalid	– inf		
– inf					+inf	invalid		
qNaN							qNaN	
sNaN								invalid

【注】 FPSCR.DN=1 时，把 DENORM 作为 0 来处理。
FPSCR.DN=0时， DENORM的计算与NORM相同。

(4) 有可能发生的异常与发生上溢 / 下溢异常陷阱的条件

- FPU错误
- 无效运算
- 上溢
  发生上溢异常陷阱的条件
  - FPSCR.PR=0时：FRn与FRm符号相异且至少一方的指数为H'FE
  - FPSCR.PR=1时：DRn与DRm符号相异且至少一方的指数为H'7FE
- 下溢
  发生下溢异常陷阱的条件
  - FPSCR.PR=0时：FRn与FRm符号相同且指数均小于等于H'18
  - FPSCR.PR=1时：DRn与DRm符号相同且指数均小于等于H'035
- 不精确异常

10.3.25 FTRC Floating - point Truncate and Convert to integer 浮点指令

转换为整数

PR	格式	操作概略	操作码	执行状态	T 位
0	FTRC FRm,FPUL	(long)FRm→FPUL	1111mmmm00111101	1	—
1	FTRC DRm,FPUL	(long)DRm→FPUL	1111mmm000111101	1	—

(1) 说明

- FPSCR.PR=0时
  将FRm的内容的单精度浮点数转换为32位整数，将结果保存在FPUL。
- FPSCR.PR=1时
  将DRm的内容的双精度浮点数转换为32位整数，将结果保存在FPUL。
  通常将舍入模式设定为下舍入。

(2) 注意事项

无

(3) 操作内容

```
#define N_INT_SINGLE_RANGE 0xcf000000 & 0x7fffffff  /* -1.000000 * 2^31 */
#define P_INT_SINGLE_RANGE 0x4effffff  /* 1.fffffe * 2^30 */
#define N_INT_DOUBLE_RANGE 0xc1e0000000200000 & 0x7fffffffffffffff
#define P_INT_DOUBLE_RANGE 0x41e0000000000000

void FTRC(int m,int *FPUL)
{
     pc += 2;
     clear_cause();
     if(FPSCR.PR==0){
          case(ftrc_single_ type_of(m)){
          NORM:     *FPUL = FR[m];break;
          PINF:     ftrc_invalid(0, *FPUL);break;
          NINF:     ftrc_invalid(1, *FPUL);break;
          }
     }
     else{                  /* case FPSCR.PR=1 */
          case(ftrc_double_type_of(m)){
          NORM:     *FPUL = DR[m>>1];break;
          PINF:     ftrc_invalid(0, *FPUL);break;
          NINF:     ftrc_invalid(1, *FPUL);break;
          }
     }
}
int ftrc_single_type_of(int m)
{
if(sign_of(m) == 0){
if(FR_HEX[m] > 0x7f800000)return(NINF);/* NaN */
else if(FR_HEX[m] > P_INT_SINGLE_RANGE)
 return(PINF);/* out of range,+INF */
else return(NORM);/* +0,+NORM          */
} else {
if((FR_HEX[m] & 0x7fffffff) > N_INT_SINGLE_RANGE)
 return(NINF);/* out of range ,+INF,NaN*/
else return(NORM);/* -0,-NORM              */
     }
}
int ftrc_double_type_of(int m)
{
if(sign_of(m)==0){
if((FR_HEX[m] > 0x7ff00000) ||
   ((FR_HEX[m] == 0x7ff00000) &&
    (FR_HEX[m+1] != 0x00000000)))   return(NINF);      /* NaN */
else if(DR_HEX[m>>1] >= P_INT_DOUBLE_RANGE)
 return(PINF);/* out of range,+INF */
else return(NORM);/* +0,+NORM          */
} else{
if((DR_HEX[m>>1] & 0x7fffffffffffffff) >= N_INT_DOUBLE_RANGE)
 return(NINF);/* out of range ,+INF,NaN*/
else return(NORM);/* -0,-NORM              */
}
```

```
}
void ftrc_invalid(int sign,int *FPUL)
{
     set_V();
     if((FPSCR&ENABLE_V) == 0){
if(sign == 0) *FPUL = 0x7fffffff;
else             *FPUL = 0x80000000;
     }
     else fpu_exception_trap();
}
```

FTRC 特殊情况

FRn,DRn	NORM	+0	– 0	Positive Out of Range	Negative Out of Range	+INF	– INF	qNaN	sNaN
FTRC (FRn,DRn)	TRC	0	0	Invalid +MAX	Invalid – MAX	Invalid +MAX	Invalid – MAX	Invalid +MAX	Invalid – MAX

【注】 DN=1 时，将非规格化数的值作为 0 来处理。

(4) 有可能发生的异常

- 无效运算

10.3.26 FTRV Floating - point TRansform Vector 浮点指令

转换向量

PR	格式	操作概略	操作码	执行状态	T 位
0	FTRV XMTRX,FVn	transform_vector(XMTRX, FVn)→	1111nn0111111101	4	—
1	—	FVn	–	—	—
		—			

(1) 说明

- FPSCR.PR=0时

  将XMTRX所示的浮点寄存器XF0～XF15的内容作为4行4列的矩阵，将FVn所示浮点寄存器FR[n]～FR[n+3]的内容作为4维向量，进行矩阵与向量的乘法运算，将结果保存在FVn。

XMTRXFVnFVn

$$\begin{pmatrix} XF[0] & XF[4] & XF[8] & XF[12] \\ XF[1] & XF[5] & XF[9] & XF[13] \\ XF[2] & XF[6] & XF[10] & XF[14] \\ XF[3] & XF[7] & XF[11] & XF[15] \end{pmatrix} \times \begin{pmatrix} FR[n] \\ R[n+1] \\ FR[n+2] \\ FR[n+3] \end{pmatrix} \rightarrow \begin{pmatrix} FR[n] \\ FR[n+1] \\ FR[n+2] \\ FR[n+3] \end{pmatrix}$$

FTRV 指令追求正确度更追求高速。因此，结合 FADD 或 FMUL 使用时，结果不同。按照下列步骤执行FTRV：

1. 分别对各项进行乘法运算。结果为28位。
2. 调整结果。此时，舍入为30位以内。
3. 加上调整后的结果。
4. 进行规格化与舍入。

下列情况时，为特殊处理。

1. 输入值中有sNaN时，产生无效异常。
2. 进行乘法运算的输入值有0与无穷大的组合时，为无效运算异常。
3. 上述以外，输入值包括qNaN时，结果为qNaN。
4. 上述以外，输入值包括无穷大时
   - 如果乘法运算后的结果为2个以上无穷大，符号相异时，为无效运算异常。
   - 上述以外时，保存正确的无穷大。
5. 输入值不包括sNaN、 qNaN、无穷大时，进行通常处理。

   如果设定为FPSCR.enable.V/O/U/I，发生FPU异常陷阱，与是否发生异常无关。发生异常时，在FPSCR.cause FPSCR.flag反映正确的异常信息，不更新FVn。请通过软件进行适当处理。

(2) 注意事项

无

(3) 操作内容

```
void FTRV (int n)     /* FTRV FVn */
{
float saved_vec[4],result_vec[4];
int saved_fpscr;
int dst,i;
      if(FPSCR_PR == 0){
          PC += 2;
          clear_cause();
          saved_fpscr = FPSCR;
          FPSCR &= ~ENABLE_VOUI; /* mask VOUI enable */
          dst = 12 - n;/* select other vector than FVn */
          for (i=0;i<4;i++)saved_vec [i] = FR[dst+i];
          for (i=0;i<4;i++){
              for(j=0;j<4;j++)FR[dst+j] = XF[i+4j];
              fipr(n,dst);
              saved_fpscr |= FPSCR & (CAUSE|FLAG) ;
              result_vec[i] = FR[dst+3];
          }
          for (i=0;i<4;i++)FR[dst+i] = saved_vec[i];
          FPSCR = saved_fpscr;
          If(FPSCR & ENABLE_VOUI) fpu_exception_trap();
          Else for (i=0;i<4;i++)FR[n+i] = result_vec[i];
      }
      else undefined_operation();
}
```

(4) 有可能发生的异常

- 无效运算
- 上溢
- 下溢
- 不精确异常

第 11 章 寄存器一览

本章以一览表的形式汇总了各章说明的有关内部 I/O 寄存器内容。

1. 寄存器地址一览（按各功能模块均、手册章编号的顺序）
   - 按各功能模块均、手册章编号的顺序。
   - 请勿存取本列表未记载的保留地址。

2. 各运行模式下的寄存器状态
   - 按照“寄存器地址一览（按各功能模块均、手册章编号的顺序）”的排列，描述寄存器的状态。
   - 初始化时各个位的状态请参考对应章的寄存器说明。
   - 表示基本的运行模式时的寄存器状态，有内部模块固有的复位时，请参考对内部模块进行说明的章。

11.1 寄存器地址一览（按各功能模块均、手册章编号的顺序）

【注】 禁止存取未定义及保留地址。由于无法保证存取这些寄存器时的运行及连续运行，所以请勿存取。

模块名称	名称	简称	R/W	P4 区域地址 *	区域 7 地址 *	存取大小
异常处理	TRAPA 异常寄存器	TRA	R/W	H'FF00 0020	H'1F00 0020	32
	异常事件寄存器	EXPEVT	R/W	H'FF00 0024	H'1F00 0024	32
	中断事件寄存器	INTEVT	R/W	H'FF00 0028	H'1F00 0028	32
MMU	页表入口高位寄存器	PTEH	R/W	H'FF00 0000	H'1F00 0000	32
	页表入口低位寄存器	PTEL	R/W	H'FF00 0004	H'1F00 0004	32
	转换表基址寄存器	TTB	R/W	H'FF00 0008	H'1F00 0008	32
	TLB 异常地址寄存器	TEA	R/W	H'FF00 000C	H'1F00 000C	32
	MMU 控制寄存器	MMUCR	R/W	H'FF00 0010	H'1F00 0010	32
	物理地址空间控制寄存器	PASCR	R/W	H'FF00 0070	H'1F00 0070	32
	重新取指令禁止控制寄存器	IRMCR	R/W	H'FF00 0078	H'1F00 0078	32
高速缓存	高速缓存控制寄存器	CCR	R/W	H'FF00 001C	H'1F00 001C	32
	队列地址控制寄存器 0	QACR0	R/W	H'FF00 0038	H'1F00 0038	32
	队列地址控制寄存器 1	QACR1	R/W	H'FF00 003C	H'1F00 003C	32
	内部存储器控制寄存器	RAMCR	R/W	H'FF00 0074	H'1F00 0074	32
L 存储器	L 存储器传送源地址寄存器 0	LSA0	R/W	H'FF00 0050	H'1F00 0050	32
	L 存储器传送源地址寄存器 1	LSA1	R/W	H'FF00 0054	H'1F00 0054	32
	L 存储器传送目标地址寄存器 0	LDA0	R/W	H'FF00 0058	H'1F00 0058	32
	L 存储器传送目标地址寄存器 1	LDA1	R/W	H'FF00 005C	H'1F00 005C	32

【注】 * P4 区域地址为使用虚拟地址空间的 P4 区域的地址。区域 7 地址为使用 TLB、从物理地址空间的区域 7 存取的地址。

11.2 各运行模式下的寄存器状态

【注】 * 寄存器的初始值请参考对各模块进行说明的章。另外，关于初始值不稳定的寄存器，无法保证其值，所以表现为初始化。

模块名称	名称	简称	上电复位	手动复位	睡眠	待机
异常处理	TRAPA 异常寄存器	TRA	不定	不定	保持	保持
	异常事件寄存器	EXPEVT	H'0000 0000	H'0000 0020	保持	保持
	中断事件寄存器	INTEVT	不定	不定	保持	保持
MMU	页表入口高位寄存器	PTEH	不定	不定	保持	保持
	页表入口低位寄存器	PTEL	不定	不定	保持	保持
	转换表基址寄存器	TTB	不定	不定	保持	保持
	TLB 异常地址寄存器	TEA	不定	保持	保持	保持
	MMU 控制寄存器	MMUCR	H'0000 0000	H'0000 0000	保持	保持
	物理地址空间控制寄存器	PASCR	H'0000 0000	H'0000 0000	保持	保持
	重新取指令禁止控制寄存器	IRMCR	H'0000 0000	H'0000 0000	保持	保持
高速缓存	高速缓存控制寄存器	CCR	H'0000 0000	H'0000 0000	保持	保持
	队列地址控制寄存器 0	QACR0	不定	不定	保持	保持
	队列地址控制寄存器 1	QACR1	不定	不定	保持	保持
	内部存储器控制寄存器	RAMCR	H'0000 0000	H'0000 0000	保持	保持
L 存储器	L 存储器传送源地址寄存器 0	LSA0	不定	不定	保持	保持
	L 存储器传送源地址寄存器 1	LSA1	不定	不定	保持	保持
	L 存储器传送目标地址寄存器 0	LDA0	不定	不定	保持	保持
	L 存储器传送目标地址寄存器 1	LDA1	不定	不定	保持	保持

附录

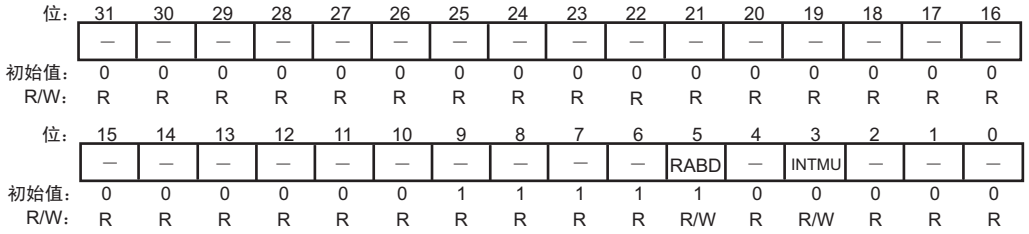
附录 A. CPU 运行模式寄存器 （CPUOPM）

CPUOPM 用于切换 CPU 的运行模式。本寄存器能够以 32 位长度从 P4 区域的 H'FF2F0000 或区域地址的 H'1F2F0000 读取 / 写入。写入本寄存器时，必须要将初始值写入保留位。不保证将初始值以外的值写入保留位时的运行。

CPUOPM 的更新不是从 CPU 以外的 SuperHyway 总线主控器进行存取，而通过 CPU 的存储指令来进行。另外， CPUOPM 更新后，读取一次 CPUOPM 后，请执行以下 1. 或 2. 中的任意一项。

1. 请执行使用RTE指令的转移。
2. 对于任意地址 （不可高速缓存操作的区域亦可），请执行ICBI指令。

执行 1. 或 2. 之后， CPU 保证使用更新后的 CPUOPM 值的运行



位	位名称	初始值	R/W	说明
31 ～ 6	—	H'000000F	R	保留位 写入时，必须要写入初始值。
5	RABD	1	R/W	子程序返回投机执行 0：从子程序返回时，投机发行取指令。将本位设定为 0 时，请参考 “附录 C. 子程序返回投机执行”。 1：从子程序返回时，不投机发行取指令。
4	—	0	R	保留位 写入时，必须要写入初始值。
3	INTMU	0	R/W	中断运行模式切换位 0：即使接受中断， SR.IMASK 的值也不发生变化。 1：接受中断时，自动地将接受的级别设定为 SR.IMASK 的值。
2 ～ 0	—	000	R	保留位 写入时，必须要写入初始值。

附录 B. 关于预取指令及其副作用

SH-4A 在内部设置保存预先读取的指令的缓冲器，通常预先读取指令。因此，请不要在各存储器空间的最终 64 字节区域分配程序。将程序分配至该区域时，会预先读取超过存储器区域的指令，从而发生总线存取。

因此造成的问题事例如下。

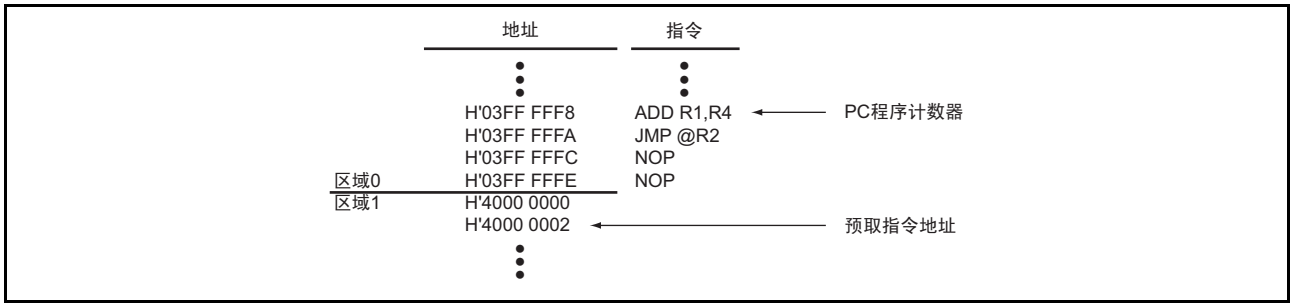


图 B.1 预取指令

图 B.1 假设 PC （程序计数器）指示的指令 （ADD）及 H'0400 0002 地址的取指令同时执行，另外，假设程序在后续的 JMP 指令及延迟槽指令执行后，转移至区域 1 以外区域。

此时，有可能会发生从程序流程对无法预测的区域 1 进行总线存取 （预取指令）。

(1) 预取指令的副作用

1. 可以考虑为预取指令引起的外部总线存取，是造成给该区域连接的FIFO等外部设备而发生误动作的原因。
2. 响应预取指令引起的外部总线请求的设备不存在时，会发生死机。

(2) 避免方法

1. 通过使用MMU，可避免这些非法的取指令。
2. 不给各区域最终64字节的区域分配程序，可避免发生问题。

附录 C. 子程序返回投机执行

SH-4A 内部具有从子程序返回时投机发行取指令的结构。通过从子程序返回时投机发行取指令，可缩短返回时的执行周期。将 CPU 运行模式寄存器 （CPUOPM）的 bit5 （RABD）的值设定为 0 时，该功能有效。但，从子程序返回时如果投机发行取指令，则对于程序上不该存取的地址有时会执行取指令。其结果有可能会发生对无法预测的区域的总线存取，或发生内部的指令地址错误从而引起误动作。由于发生对无法预测区域的总线存取而产生的副作用请参考 “图 B.1 预取指令”。

使用条件：

子程序返回投机执行的功能有效时，对于通过 JSR/BSR/BSRF 指令在 PR 设定的返回地址，请使用 RTS 指令来从子程序返回。因此，可禁止存取程序上不该存取的地址，并可避免误动作。

附录 D. 版本寄存器 （PVR、 PRR）

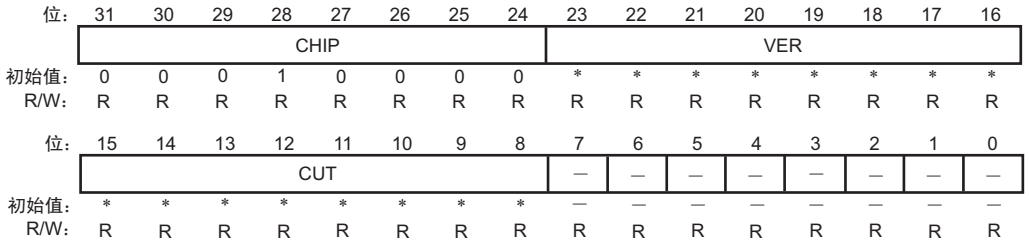
SH-4A 内置表示处理器内核版本与产品版本的只读寄存器。通过使用这些寄存器的值，可从软件区分处理器的版本及产品版本，并可构筑扩展性高的系统。由于版本寄存器的值因产品不同而各异，所以详细内容请参考各产品的硬件手册或咨询瑞萨科技、瑞萨销售及专卖店。

【注】 必须屏蔽 PVR 寄存器的 bit7 ～ bit0 及 PRR 寄存器的 bit3 ～ bit0 的值，请不要影响软件

表 D.1 寄存器结构

名称	简称	R/W	P4 区域地址	区域 7 地址	大小
处理器版本寄存器	PVR	R	H'FF00 0030	H'1F00 0030	32
产品寄存器	PRR	R	H'FF00 0044	H1F00 0044	32

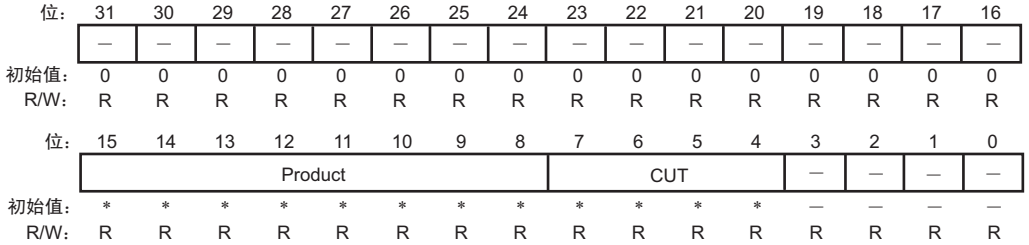
- PVR



位	位名称	初始值	R/W	说明
31 ～ 24	CHIP	H'10	R	表示处理器族的类别。 为 SH-4A 系列时，必须读取 H'10。
23 ～ 16	VER	*	R	表示版本。 SH-4A 系列中，进行大幅度功能扩展的情况下变更此值。 在本手册中 VER 仅将 H'20 作为目标。
15 ～ 8	CUT	*	R	表示版本。 SH-4A 系列中，进行小规模修订的情况下变更此值。 本位根据产品不同而各异。
7 ～ 0	—	不定	R	读取不定值。 从软件读取后，必须屏蔽以后再使用。

【注】 * 本位的值根据产品不同而各异。

- PRR



位	位名称	初始值	R/W	说明
31 ～ 16	—	均为 0	R	均固定为 0。
15 ～ 8	Product	*	R	表示产品类别。 本位的值因产品不同而各异。
7 ～ 4	CUT	*	R	表示版本。 产品进行小规模修订时，变更此值。本位因产品不同而各异。
3 ～ 0	—	不定	R	读取不定值。 从软件读取后，必须屏蔽以后再使用。

【注】 * 本位的值根据产品不同而各异。

索引

数字

B

C

D

F

G

H

J

K

L

N

P

S

T

W

X

Y

Z

修订记录	SH-4A 软件手册

Rev.	发行日	修订内容	
		页	修订处
1.00	2008.10.27	—	初版发行

瑞萨 32 位 RISC 单片机
软件手册
SH-4A

Publication Date:	Rev1.00, Oct.27, 2008
Published by:	Sales Strategic Planning Div.
	Renesas Technology Corp.
Edited by:	Customer Support Department
	Global Strategic Communication Div.
	Renesas Solutions Corp.

Renesas Technology Corp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan

RENESAS SALES OFFICES

http://www.renesas.com

Refer to "**http://www.renesas.com/en/network**" for the latest and detailed information.

Renesas Technology America, Inc.
450 Holger Way, San Jose, CA 95134-1368, U.S.A
Tel: <1> (408) 382-7500, Fax: <1> (408) 382-7501

Renesas Technology Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.
Tel: <44> (1628) 585-100, Fax: <44> (1628) 585-900

Renesas Technology (Shanghai) Co., Ltd.
Unit 204, 205, AZIACenter, No.1233 Lujiazui Ring Rd, Pudong District, Shanghai, China 200120
Tel: <86> (21) 5877-1818, Fax: <86> (21) 6887-7858/7898

Renesas Technology Hong Kong Ltd.
7th Floor, North Tower, World Finance Centre, Harbour City, Canton Road, Tsimshatsui, Kowloon, Hong Kong
Tel: <852> 2265-6688, Fax: <852> 2377-3473

Renesas Technology Taiwan Co., Ltd.
10th Floor, No.99, Fushing North Road, Taipei, Taiwan
Tel: <886> (2) 2715-2888, Fax: <886> (2) 3518-3399

Renesas Technology Singapore Pte. Ltd.
1 Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: <65> 6213-0200, Fax: <65> 6278-8001

Renesas Technology Korea Co., Ltd.
Kukje Center Bldg. 18th Fl., 191, 2-ka, Hangang-ro, Yongsan-ku, Seoul 140-702, Korea
Tel: <82> (2) 796-3115, Fax: <82> (2) 796-2145

Renesas Technology Malaysia Sdn. Bhd
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No.18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: <603> 7955-9390, Fax: <603> 7955-9510

SH-4A

RENESAS

瑞萨电子株式会社

RCJ09B0060-0100