



UCP4008 Instruction Set Specification

Document number: SL-UCP-000006

Document revision: V1.0

2023/07/27

目录

1. 指令集介绍.....	10
1.1. OVERVIEW	10
1.2. 标量处理单元 SPU	11
1.3. 向量处理单元 MPU.....	12
1.4. 存储架构	13
1.4.1. 存储架构概述	13
1.4.2. DM 读写方式.....	15
1.5. PIPELINE	16
1.5.1. Pipeline Overview	16
1.5.2. SPU Pipeline	17
1.5.3. MPU Pipeline.....	18
2. 指令详细	19
2.1. 指令详细说明页示例	19
2.2. SPU	20
2.2.1. Intrinsics 说明.....	22
2.2.2. SCU	23
2.2.2.1. 定点加法指令	23
2.2.2.2. 定点减法指令	25
2.2.2.3. 定点乘法指令	27
2.2.2.4. 定点除法指令	28
2.2.2.5. 浮点加法指令	30
2.2.2.6. 浮点减法指令	31
2.2.2.7. 浮点乘法指令	32
2.2.2.8. 类型转换指令: 定点 -> 单精度浮点.....	33
2.2.2.9. 类型转换指令: 单精度浮点 -> 定点.....	34
2.2.2.10. 定点绝对值指令	35
2.2.2.11. 浮点绝对值指令	36
2.2.2.12. 读 RFC 标志位指令	37
2.2.2.13. 写 RFC 标志位指令	38
2.2.2.14. 条件寄存器读指令	39

2.2.2.15. 按位与指令	40
2.2.2.16. 按位或指令	41
2.2.2.17. 按位异或指令	42
2.2.2.18. 按位非指令	43
2.2.2.19. 定点等于指令	44
2.2.2.20. 定点不等于指令	45
2.2.2.21. 定点大于等于指令	46
2.2.2.22. 定点小于指令	47
2.2.2.23. 浮点等于指令	48
2.2.2.24. 浮点不等于指令	49
2.2.2.25. 浮点大于等于指令	50
2.2.2.26. 浮点小于指令	51
2.2.2.27. 寄存器左移指令	52
2.2.2.28. 立即数左移指令	53
2.2.2.29. 寄存器右移指令	54
2.2.2.30. 立即数右移指令	55
2.2.2.31. 立即数赋值指令	56
2.2.2.32. BitFilter 指令	57
2.2.2.33. BitExpd 指令	58
2.2.2.34. MergeShift 指令	60
2.2.2.35. Count 指令	61
2.2.2.36. BitReverse 指令	62
2.2.2.37. First 指令	63
2.2.2.38. GetSign 指令	65
2.2.2.39. Select 指令	66
2.2.2.40. 寄存器 Extend 指令	68
2.2.2.41. 立即数 Extend 指令	69
2.2.2.42. CRC 指令	70
2.2.2.43. NOP 指令	72
2.2.3. AGU	73
2.2.3.1. 通用寄存器加载指令	73
2.2.3.2. 通用寄存器存储指令	75
2.2.3.3. 定点加法指令	77
2.2.3.4. 定点减法指令	78
2.2.3.5. SVR 寄存器加载指令	79
2.2.3.6. SVR 寄存器存储指令	81
2.2.3.7. SVR 寄存器加法指令	83
2.2.3.8. SVR 寄存器减法指令	84
2.2.3.9. 向量加载指令	85
2.2.3.10. CSU 向量加载指令	87
2.2.3.11. 向量存储指令	88
2.2.3.12. CSU 向量存储指令	91
2.2.3.13. 通用寄存器立即数偏移加载指令	92

2.2.3.14. 通用寄存器立即数偏移存储指令.....	94
2.2.3.15. 通用寄存器立即数定点加法指令.....	96
2.2.3.16. Merge 指令.....	97
2.2.3.17. MergeR 指令.....	99
2.2.3.18. 同步指令.....	100
2.2.3.19. NOP 指令	101
2.2.4. SEQ	102
2.2.4.1. 绝对跳转指令	102
2.2.4.2. 相对跳转指令	103
2.2.4.3. 绝对函数调用指令	105
2.2.4.4. 相对函数调用指令	106
2.2.4.5. 循环指令	108
2.2.4.6. SPU Stop 指令.....	109
2.2.4.7. Debug Break 指令	110
2.2.4.8. 立即数中断配置指令	111
2.2.4.9. 寄存器中断配置指令	112
2.2.4.10. 读中断配置指令	113
2.2.4.11. 读 PC 指令	114
2.2.4.12. 绝对预取指令	115
2.2.4.13. 相对预取指令	116
2.2.4.14. ICacheConfig 指令	117
2.2.4.15. 中断响应地址配置指令	118
2.2.4.16. 通用寄存器加载指令	119
2.2.4.17. 通用寄存器存储指令	120
2.2.4.18. 通用寄存器立即数偏移加载指令.....	121
2.2.4.19. 通用寄存器立即数偏移存储指令.....	123
2.2.4.20. SeqWord 指令.....	124
2.2.4.21. SeqShort 指令	126
2.2.4.22. SeqByte 指令	128
2.2.4.23. 寄存器传输指令: SVR -> R	130
2.2.4.24. 跳转模式设置指令	131
2.2.4.25. Word 位反序指令	132
2.2.4.26. SVR 位反序指令	133
2.2.4.27. NOP 指令	134
2.2.5. SYN.....	135
2.2.5.1. 立即数调用 MPU 指令	135
2.2.5.2. 寄存器调用 MPU 指令	137
2.2.5.3. MPU 状态查询指令	139
2.2.5.4. 读 FIFO 指令	140
2.2.5.5. 写 FIFO 指令	141
2.2.5.6. 寄存器传输指令: SVR -> MReg	142
2.2.5.7. 单个 MC 配置寄存器配置指令	144
2.2.5.8. 全体 MC 配置寄存器配置指令	146

2.2.5.9. 单个 MC 配置寄存器读取指令	147
2.2.5.10. 寄存器传输指令: KI -> R.....	149
2.2.5.11. 寄存器传输指令: R -> KI.....	151
2.2.5.12. 定点加法指令	153
2.2.5.13. 定点减法指令	154
2.2.5.14. 定点乘法指令	155
2.2.5.15. 定点除法指令	156
2.2.5.16. 读 RFC 标志位指令	158
2.2.5.17. 写 RFC 标志位指令	159
2.2.5.18. 按位与指令	160
2.2.5.19. 按位或指令	161
2.2.5.20. 按位异或指令	162
2.2.5.21. 按位非指令	163
2.2.5.22. 定点等于指令	164
2.2.5.23. 定点不等于指令	165
2.2.5.24. 定点大于等于指令	166
2.2.5.25. 定点小于指令	167
2.2.5.26. 寄存器左移指令	168
2.2.5.27. 立即数左移指令	169
2.2.5.28. 寄存器右移指令	170
2.2.5.29. 立即数右移指令	171
2.2.5.30. 立即数赋值指令	172
2.2.5.31. Select 指令	173
2.2.5.32. 寄存器 Extend 指令	175
2.2.5.33. 立即数 Extend 指令	176
2.2.5.34. NOP 指令	177
2.3. MPU	178
2.3.1. 互联关系	181
2.3.1.1. 功能单元整体互联数据通路	181
2.3.1.2. MReg 不同模式下的互联关系.....	181
2.3.1.3. 各功能单元数据出口	183
2.3.2. Wait 指令	184
2.3.3. Intrinsics 说明	185
2.3.4. MReg	187
2.3.4.1. Read 指令	190
2.3.4.2. Pre Config 指令	193
2.3.4.3. Config MFetch 指令	196
2.3.4.4. Config Latch 指令.....	198
2.3.4.5. Config RPort 指令	200
2.3.4.6. Config WPort 指令	202

2.3.4.7. Config BIU 指令	204
2.3.4.8. Read RPort Config 指令	206
2.3.4.9. Read WPort Config 指令	207
2.3.4.10. Config Depth 指令	208
2.3.4.11. Wait 立即数指令	210
2.3.4.12. Wait KI 指令	211
2.3.4.13. NOP 指令	213
2.3.4.14. Set Condition 指令	214
2.3.5. SHU	215
2.3.5.1. Index Short 指令	216
2.3.5.2. Pre Index byte 指令	220
2.3.5.3. Index byte 指令	221
2.3.5.4. 向量加法指令: 寄存器 + 立即数	227
2.3.5.5. 向量加法指令: 寄存器 + 寄存器	229
2.3.5.6. 向量位移指令	231
2.3.5.7. 向量逻辑运算指令	233
2.3.5.8. PN 序列生成指令	235
2.3.5.9. CRC 校验指令	237
2.3.5.10. Extract Bit 指令	239
2.3.5.11. Insert Bit 指令	241
2.3.5.12. Step Extract 指令	243
2.3.5.13. Reverse Step Extract 指令	245
2.3.5.14. Imm 指令	247
2.3.5.15. Turbo 指令	248
2.3.5.16. Sort 指令	252
2.3.5.17. Wait 立即数指令	256
2.3.5.18. Wait KI 指令	257
2.3.5.19. NOP 指令	259
2.3.5.20. Set Condition 指令	260
2.3.6. BIU	261
2.3.6.1. 加载指令: 普通模式	265
2.3.6.2. 加载指令: 离散模式	268
2.3.6.3. 加载指令: 默认模式	271
2.3.6.4. 存储指令: 普通模式	273
2.3.6.5. 存储指令: 离散模式	275
2.3.6.6. 存储指令: 默认模式	278
2.3.6.7. 传送指令 BIUKG	279
2.3.6.8. 标量加法指令 BIUAddW	281
2.3.6.9. 标量减法指令 BIUSubW	282
2.3.6.10. 标量反序加法指令 BIUAddWR	283
2.3.6.11. 标量反序减法指令 BIUSubWR	284
2.3.6.12. 标量按位与指令 BIUAnd	285
2.3.6.13. 标量按位或指令 BIUOr	286

2.3.6.14. 标量按位异或指令 BIUXor	287
2.3.6.15. 标量按位非指令 BIUInv	288
2.3.6.16. 标量比较指令	289
2.3.6.17. MaskGen 指令	290
2.3.6.18. 标量立即数左移/右移指令	291
2.3.6.19. 标量寄存器左移/右移指令	292
2.3.6.20. 标量立即数赋值指令	293
2.3.6.21. Move 指令	295
2.3.6.22. Bit Reverse 指令.....	297
2.3.6.23. 筛选指令	299
2.3.6.24. 扩展指令	301
2.3.6.25. 有效位生成指令	303
2.3.6.26. load 周期配置指令.....	305
2.3.6.27. 向量寄存器复位指令	306
2.3.6.28. Wait 立即数指令	307
2.3.6.29. Wait KI 指令	308
2.3.6.30. NOP 指令	310
2.3.6.31. Set Condition 指令	311
2.3.7. IMA.....	312
2.3.7.1. 乘法指令 1: Ts0/MR +/- (Ts1,Ts2)*V(Ts3).....	314
2.3.7.2. 乘法指令 2: Ts0/MR +/- Ts1*Ts2.....	319
2.3.7.3. FFT 乘法指令: Ts0 AS Ts1*Ts2.....	327
2.3.7.4. Set MR 指令.....	330
2.3.7.5. Read MR 指令	332
2.3.7.6. Set Shiftmode 指令	334
2.3.7.7. 向量等于指令	337
2.3.7.8. 向量不等于指令	339
2.3.7.9. Read Flag 指令.....	341
2.3.7.10. Set Flag 指令	343
2.3.7.11. RMax 指令.....	344
2.3.7.12. RMin 指令.....	347
2.3.7.13. 向量加法指令	349
2.3.7.14. 向量减法指令	351
2.3.7.15. 向量传送指令	353
2.3.7.16. 向量 Conj 指令	355
2.3.7.17. 向量 RAdd 指令	357
2.3.7.18. 向量 ABS 指令	360
2.3.7.19. 向量 ByteOR 指令	362
2.3.7.20. 向量按位与指令	364
2.3.7.21. 向量按位或指令	366
2.3.7.22. 向量按位异或指令	368
2.3.7.23. 向量按位非指令	370
2.3.7.24. 普通逻辑指令	372

2.3.7.25. 向量比较选择指令(CmpSel0)	374
2.3.7.26. 数据压缩 Cprs 指令	376
2.3.7.27. 向量排序 Index 指令	378
2.3.7.28. 向量除法指令	380
2.3.7.29. 向量 Count1 指令	383
2.3.7.30. 向量 First 指令	385
2.3.7.31. 向量位反序指令	387
2.3.7.32. 向量立即数左移指令	389
2.3.7.33. 向量寄存器左移指令	391
2.3.7.34. 向量立即数右移指令	393
2.3.7.35. 向量寄存器右移指令	395
2.3.7.36. 拼接移位指令	397
2.3.7.37. 比特筛选 BitFilter 指令	399
2.3.7.38. 比特展开 BitExpd 指令	401
2.3.7.39. 加法后取模指令	403
2.3.7.40. 立即数赋值指令	405
2.3.7.41. Wait 立即数指令	407
2.3.7.42. Wait KI 指令	408
2.3.7.43. NOP 指令	410
2.3.7.44. Set Condition 指令	411
2.3.8. MFetch.....	412
2.3.8.1. 定点加法指令	412
2.3.8.2. 定点减法指令	413
2.3.8.3. 定点比较指令: 小于	414
2.3.8.4. 定点比较指令: 小于等于	415
2.3.8.5. 定点比较指令: 等于	416
2.3.8.6. 定点比较指令: 不等于	417
2.3.8.7. 定点左移指令	418
2.3.8.8. 定点右移指令	419
2.3.8.9. 定点按位与指令	420
2.3.8.10. 定点按位或指令	421
2.3.8.11. 定点按位非指令	422
2.3.8.12. 定点传输指令	423
2.3.8.13. 立即数 Repeat 指令	424
2.3.8.14. 寄存器 Repeat 指令	425
2.3.8.15. Loop 指令	426
2.3.8.16. 读 FIFO 指令	427
2.3.8.17. 写 FIFO 指令	428
2.3.8.18. 相对地址跳转指令	429
2.3.8.19. 绝对地址跳转指令	431
2.3.8.20. JumpPreFetchImm	433
2.3.8.21. JumpPreFetchKI	434
2.3.8.22. 传输指令: MFetch -> MReg	435

2.3.8.23. 传输指令: MFetch -> BIU	436
2.3.8.24. 立即数赋值指令	437
2.3.8.25. ConfigJumpMode	438
2.3.8.26. MPU Wait 指令	439
2.3.8.27. MPU Stop 指令	440
2.3.8.28. NOP 指令	441

Smart Logic Confidential For yuyi@ruijie.com.cn

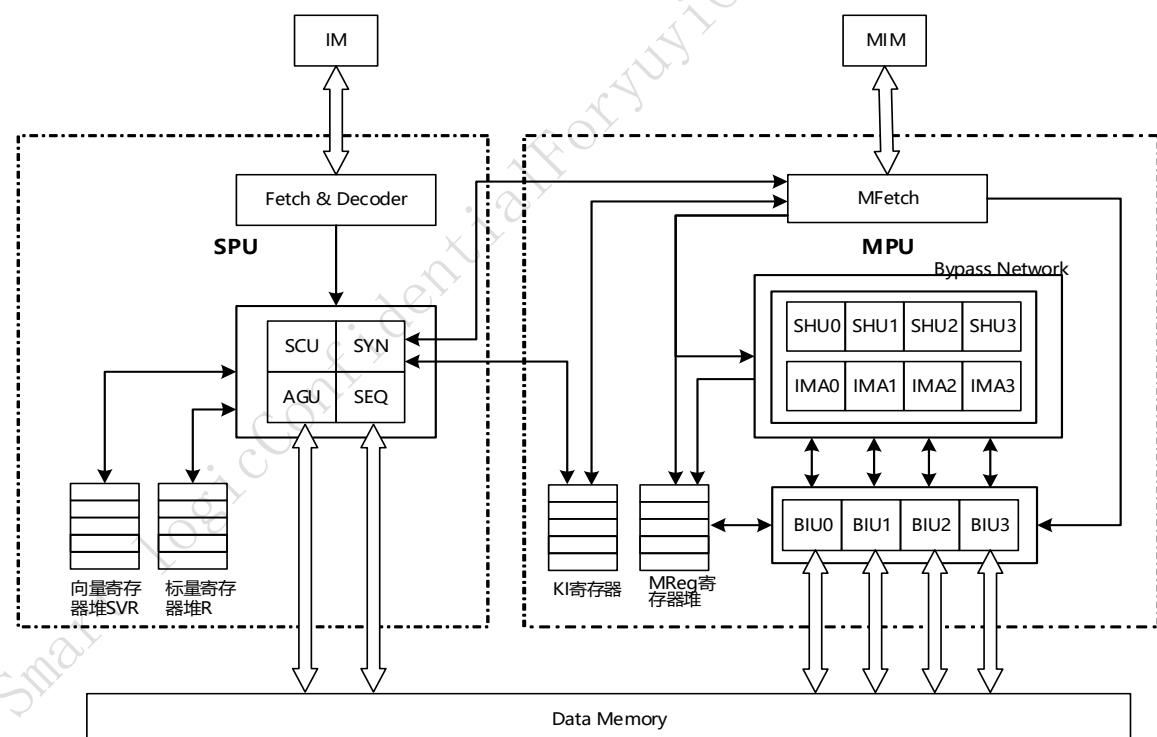
1. 指令集介绍

1.1. Overview

UCP2.0 的一个 APE 核采用 AppAISArc™(Application Algorithm ISA)指令集体体系结构, 它包括:

- 一个标量处理单元 SPU: 4 发射标量流水线;
- 一个向量处理单元 MPU: 21 发射向量流水线;
- 256KB 标量指令存储器 IM;
- 144KB 向量指令存储器 MIM;
- 8 个 DM, 每个 DM 为 256KB (同一个 APC 内的两个 APE 共用);
- 一个通信和同步单元 CSU (同一个 APC 内的两个 APE 共用).

下图是它的结构示意图:



1.2. 标量处理单元 SPU

标量处理单元包括 SCU, SEQ, AGU 和 SYN 四个功能部件, 执行标量指令. 各个功能部件的处理能力如下:

- SCU: 主要进行标量计算
 - 32 位整数加、减、乘、除、移位、比较、逻辑类、求反等操作;
 - IEEE 754 单精度浮点加、减、乘等操作;
 - IEEE 754 单精度浮点与定点数据的类型转换操作;
 - 立即数赋值操作.
- AGU: 主要进行地址计算及 load/store
 - 标量数据加载/存储, 向量数据加载/存储等操作;
 - 读/写 CSU 寄存器.
- SEQ: 主要进行程序流控制
 - 有条件/无条件跳转操作;
 - 有条件/无条件子程序调用操作;
 - 硬件循环;
 - 中断配置.
- SYN: 主要进行与向量流水线通路的同步操作
 - 调用 MPU 程序, 查询 MPU 状态;
 - 配置 MPU 部分寄存器, 通过 FIFO 与 MPU 的 MFetch 进行数据交换;
 - 进行标量计算.

各部件可用的寄存器资源如下:

- R 寄存器堆: 通用寄存器堆, 共有 32 个 32bit 宽度的 R 寄存器, 其中 R0 为只读寄存器, 值为 0, R31 存放中断跳转时的返回地址.
- 特殊向量寄存器 SVR: 存放向量数据, 共有 4 个 512bit 宽度的 SVR 寄存器.

SPU 各个功能部件支持的指令将在 2.2 章进行详细介绍, 预了解指令的具体功能请参见后续章节.

1.3. 向量处理单元 MPU

向量处理单元包括：向量程序控制部件(MFetch), M 寄存器文件堆(MReg), 向量访存部件(BIU0~BIU3), 向量运算部件(IMA0~IMA3), 向量数据交织部件(SHU0~SHU3). 除 MFetch 外, 各个部件处理的数据位宽为 512bit.

- MFetch: 向量程序控制器, 执行向量流水线与标量流水线的同步, 向量的循环跳转等操作. 其内部包含 16 个 24bit 的 KI 寄存器.
- IMA: 执行向量乘累加, 乘法, 加法等运算指令. 每个 IMA 内部包含 6 个 512bit 的普通寄存器 T0~T5, 1 个 512bit 的特殊寄存器 T6(Ttmp), 一个中间结果寄存器 MR.
- SHU: Shuffle 部件, 执行数据广播、抽取、内部交织等指令. 每个 SHU 内部包含 8 个 512bit 的寄存器 T0~T7.
- BIU: 总线接口单元, 执行地址计算, 加载/存储等指令. 每个 BIU 内部包含 4 个 512bit 的寄存器 T0~T3.
- MReg: 矩阵寄存器堆, 为通用向量寄存器堆. MReg 的深度可配置, 详情请参见 2.3.4.

1.4. 存储架构

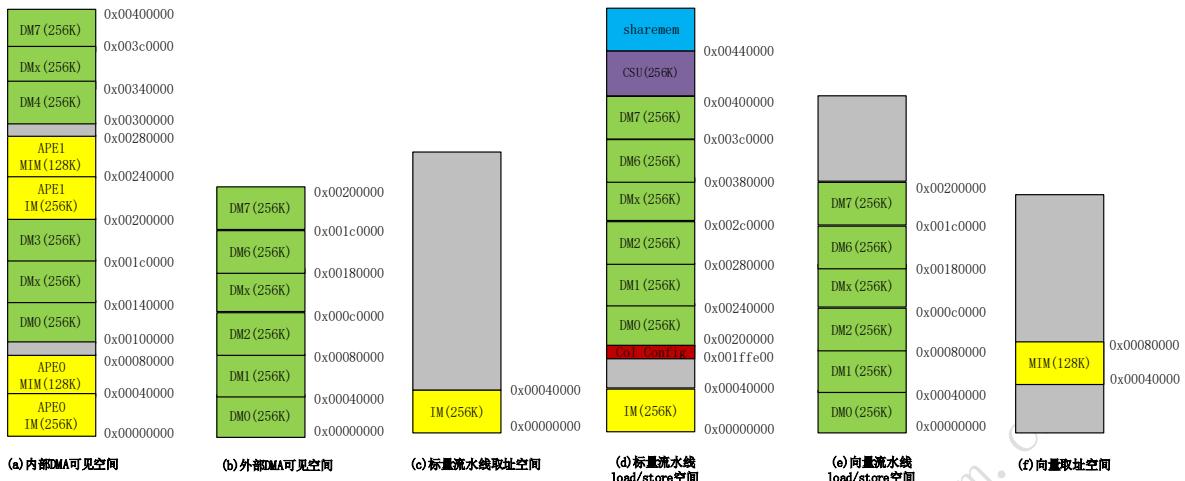
1.4.1. 存储架构概述

一个 APE 含有:

- MIM: 144KB, 用于存储向量指令.
- IM: 256KB, 支持 4 种使用模式:
 - 单纯的 IM: SPU 的指令寻址空间仅为 256KB, IM 中可以存储标量指令和标量数据;
 - 三种可配置 Icache: SPU 的指令寻址空间为 4G, IM 根据配置不同可划分 64KB, 128KB, 256KB 作为 Icache, 剩余空间存放标量数据.
- 8 个 DM, 每个 DM 大小为 256KB. 8 个 DM 为同一个 APC 内的两个 APE 所共用, APE0 的 MPU 仅可访问 DM0~5, APE1 的 MPU 仅可访问 DM2~7, SPU 可访问全部 DM. 访问 DM 的优先级情况如下:
 - SPU 内: AGU 和 SEQ 同时访问同一个 DM 时, AGU 优先级高于 SEQ;
 - MPU 内: 各 BIU 单元可以同时访问同一个 DM, 但会导致 MPU Stall, 访问优先级为 BIU0>BIU1>BIU2>BIU3;
 - APE 内: 在一个 APE 内, MPU 的访问优先级高于 SPU;
 - APC 内的访问优先级: APE0 的 MPU > APE1 的 MPU >APE0 的 AGU > APE1 的 AGU > APE0 的 SEQ > APE1 的 SEQ.

APE 的各个部件可见的数据存储空间如下:

- DMA 可见空间:
 - APE 可以通过 CSU 单元的 DMA 访问外部的 DDR 或 Share Memory;
 - 内部 DMA 可见空间是指: APE 的 DMA 在内部与外部之间搬运数据时, 可见的 APE 内部空间;
 - 外部 DMA 可见空间是指: 其他模块从 APE 搬运数据时, 可见的 APE 地址空间;
- 标量流水线取指空间: 指标量流水线通过取指单元读取指令时, 可见的存储空间.
- 标量流水线 load/store 空间: 指标量流水线加载/存储数据时, 可见的存储空间.
- 向量流水线 load/store 空间: 指向量流水线通过 BIU 单元加载/存储数据时, 可见的存储空间.
- 向量流水线取指空间: 指向量流水线通过 MFetch 读取向量指令时的可见地址空间.
需要注意的是, 向量流水线按 Word 读取 MIM, 而 DMA 按 Byte 读写 MIM.



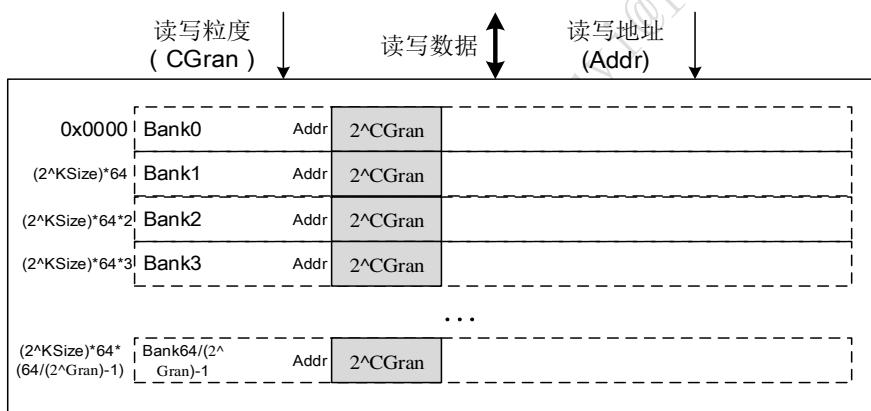
Note:

- APE 加载/存储 IM 和 DM 中的数据, 将按照访问优先级进行响应; 对于同一个功能单元的访问请求, 能确保按照指令的先后顺序进行访问.
- SPU 访问外部存储空间, 要确保同一地址处的数据写入后再被读取, 或读取后再被写入, 则必须在两次访问中间使用 AGU 的同步指令(参见 2.2.3.18).
- SPU 访问外部存储空间, 可以使用带 AT 选项的加载/存储指令进行原子操作(参见 2.2.3.1, 2.2.3.2).
- MIM 中存储的 MPU 指令字长为 64 位, 但仅低 36 位有效. 因此 MIM 大小认为是 144KB, 但 DMA 寻址范围是 0x40000~0x80000.

1.4.2.DM 读写方式

DM 一个时钟周期可读写 64 个 Byte, 支持行模式, 离散模式, 列模式等多种操作模式.

- 行模式可以从起始地址开始向后连续读写 64 个 Byte 数据.
- 离散模式可以访问低 6bit 地址分别为 0~63, 高位地址为寄存器预先配置的 64 个地址离散的 Byte 数据.
- 列模式读写数据时, DM 按 “逻辑 Bank” 寻址, 读写的数据与读写地址、Bank 大小 KSize、数据摆放粒度 Gran 和读写粒度 CGran 有关. KSize 表示每个逻辑块大小为 $(2^{KSize}) * 64$ 个字节; 数据摆放粒度 Gran 表示数据存储时按照支持同时访问 $64 / (2^Gran)$ 个 Bank 的方式存放, 如果想以列模式访问数据则必须将数据以正确的 Gran 存放; 读写粒度 CGran 表示每次读写在一个逻辑 Bank 中连续取 2^{CGran} 个字节, 总共取 $64 / (2^{CGran})$ 个 Bank, CGran 只能等于 Gran 或 6. 逻辑 Bank 内按 Byte 连续编址, 编址模式如下图所示.



DM 在进行粒度为 CGran 的读操作时, 将读写地址 (Addr) 和读写粒度 (CGran) 发送给每个逻辑 Bank, 每个逻辑 Bank 读取从 Addr 起始的连续的 2^{CGran} 个 Byte 并传递给存储器读写端口, $64 / (2^{CGran})$ 个逻辑 Bank 所读取的数据按从左到右的顺序拼接成位宽为 64Byte 的输出数据.

DM 在进行粒度为 CGran 的写操作时, 将存储器读写端口传递过来的数据按从左到右的顺序拆分成 $64 / (2^{CGran})$ 份, 每份数据为 2^{CGran} 个 Byte, 并将第 i 份数据发送给第 i 个逻辑 Bank, 同时将读写地址 (Addr) 和读写粒度 (CGran) 发送给每个逻辑 Bank. 每个逻辑 Bank 以地址 (Addr) 为起始地址, 依次写入 2^{CGran} 个 Byte.

当需要行列转换(即列写行读或行写列读)时, 需要行与列访问时所使用的 Gran 相同, 不支持不同 Gran 之间的互相访问.

当使用 Gran 不等于 6 的读写操作访问逻辑 Bank 的 2^{CGran} 个字节时, 要求这 2^{CGran} 个字节不跨越逻辑 Bank, 如 Gran 等于 6 则无此要求.

1.5. Pipeline

1.5.1. Pipeline Overview

APE 的一条指令包括很多操作，我们将他们组织成特殊的阶段序列，即使指令的动作差异很大，但所有的指令都遵循统一的序列。每一步的具体处理取决于正在执行的指令。下面是各个阶段及其执行的操作的简略描述：

- Fetch(FT): 取指阶段，从内存中读取指令字，地址为程序计数器(PC)的值。根据取回的指令字中的说明段，确定 VLIW 的指令包长度。
- Decode(DC): 译码阶段，根据指令字中的说明段确定哪些功能单元有指令，哪些功能单元为空。
- Dispatch(DP): 指令分发，将指令分发到各个功能单元。
- Execution(EX): 指令执行。在这个阶段，功能单元识别到将要执行的是什么指令，读取寄存器中的源操作数，执行运算操作或访问内存，最终将结果写回。

MPU 和 SPU 的 pipeline 略有差别，下面分别进行说明。

1.5.2. SPU Pipeline



SPU一条指令的执行分为以下阶段:

- Fetch(FT): 从 IM 中读取指令字, 根据指令字中的结束位判断指令包是否结束.
- Dispatch(DP): 将指令字分发到各功能单元, 判断新指令与执行尚未结束的指令是否有数据相关. 如果有数据相关, SPU 会 stall, 直到之前的指令执行结束; 如果没有数据相关, 各功能单元将在这个阶段读取源寄存器中的操作数.
- Execution(EX): 进行计算, 访问内存等操作. 不同指令的 EX 所需要的周期数不一样, 如果指令含有写回操作, 则结果会在最后一个 EX 写入 R 寄存器.

1.5.3. MPU Pipeline



MPU一条指令的执行分为以下阶段:

- Fetch(FT): 从 MIM 中读取指令字, 根据指令字中的说明段(Header)判断指令包长度.
- Decode(DC): 根据指令字中的说明段确定哪些功能单元有指令, 哪些功能单元为空.
执行 MFetch 的指令, 执行所有功能单元的 SetCondition 指令.
- Dispatch(DP):
 - 将指令分发到各个功能单元.
 - DP0: 执行所有功能单元的 wait 指令, 根据指令执行条件判断指令是否执行.
 - DP1: MReg 单元的读指令, 读取计算 MReg 索引所需的寄存器值(MC/latch 等).
- Execution(EX): 在这个阶段, 功能单元识别到将要执行的是什么指令, 读取寄存器中的源操作数, 执行运算操作或访问内存, 最终将结果写回. 不同指令的 EX 所需要的周期数不一样, 每条指令的具体行为请参见其对应的 Pipeline.

2. 指令详细

2.1. 指令详细说明页示例

Syntax 汇编语法说明

Encoding 指令编码说明

Description 指令介绍, 包含指令的功能, 约束等

Execution 指令执行过程的说明, 通常用伪代码表示

Pipeline 指令的流水级说明

Latency 延迟拍数, 表示依赖于该指令的下一条指令至少延后多少拍才能发射.

例如, 指令 B 依赖于指令 A, latency 为 2 表示, 当 A 在第 n 拍发射时, B 最早在第 $n + 2$ 拍才能发射.

Example 汇编示例

2.2. SPU

UCP2.0 SPU 为 4 发射 VLIW 指令集, 每个 VLIW 指令包由最多 4 条指令组成, 分别为 SCU, AGU, SEQ 和 SYN 指令.

当前指令包某 FU 上不发射指令时, 忽略不写.

指令包内指令之间用 || 隔开, 指令包结尾用 ; 标记.

指令包内指令之间没有顺序.

```
// 由 4 条指令组成的 VLIW 指令包, 包内指令无顺序
SCU_Inst || SEQ_Inst || SYN_Inst || AGU;

// 另一种形式的 4 条指令组成的 VLIW 指令包, 包内指令无顺序
AGU_Inst ||
SCU_Inst ||
SYN_Inst ||
SEQ_Inst ;

// 由 3 条指令组成的指令包, 无 SCU, AGU, SYN 指令
SEQ_Inst ||
AGU_Inst ||
SCU_Inst ;
```

汇编用语定义

符号	含义
{...}	...为可选内容
(...)	...为必选内容
X Y	值可以为 X 或者 Y
(X, Y, Z, ...)	标志集合, 可以为(X, Y)或(Y, Z), 但不能表示(), (Y)或(X, Z)
X, Y	标志分隔符, 仅在需要分开两个相邻标志时使用
X.Y	从属关系, 表示 X 中的 Y
X[Y]	索引, 表示 X 中的第 Y 个元素
[...]	表示对内存的引用
	当前指令包未结束
;	当前指令包结束
//...	...为注释

REGd, REGd0, REGd1, ...	目标寄存器, REG 为寄存器类型名, 例如 Rd, SVRd
REGs, REGs0, REGs1, ...	源寄存器, REG 为寄存器类型名, 例如 Rs, SVRs
immD	D 位有符号立即数
%	表示符号引用

编码表通用标志定义

标志	含义
P	表示该指令是否为当前指令包最后一条指令, 即 或; : 0 :: 1

2.2.1. Intrinsics 说明

SPU 所有的 intrinsics 均以 __ucps2 开头, 如 __ucps2_bitFilter 实现比特筛选指令. 具体的 intrinsic 定义请参见每条指令下面的 Intrinsics 一节. 下面对一些通用问题进行说明.

Intrinsics 中的形参参数命名规范如下:

形参参数名	对应指令中
源操作数 s0, s1, s2, ...	Rs0, Rs1, Rs2, ...
type	B/S/W/D/2D/4D/8D/LB/HB 等涉及类型的
flags, 如 col, u, sub1, ...	(Col), (U), (SUB1), ...

用户在传实参时需要遵循如下规范:

形参	实参
type	需带前缀 f_, 如 f_B, f_S, f_W, f_D, f_LB 等
flags	需带前缀 f_, 如 f_Col, f_U, f_SUB1 等, 若不想指定该 flag, 直接传 0 即可

以 First 指令的 intrinsic 为例进行说明:

```
int __ucps2_first (int s0, const int u, const int sub1)
```

会生成指令

```
SCU: Rd = First Rs0 {(U)} {(SUB1)}
```

具体的, 若调用时传递如下参数 __ucps2_first(a, f_U, 0), 其中 a 为 int 型变量, 则会生成 SCU 上的 First 指令, 带 U 选项, 不带 SUB1 选项, 完成前导 0 操作.

此外, 提供了三个 clang 的内建函数:

clang 内建函数	对应指令
int __builtin_popcount(int s0)	SCU: Rd = Count1 Rs0
int __builtin_clz (int s0)	SCU: Rd = First Rs0 (U)
int __builtin_bitreverse32(int s0)	SCU: Rd = BitReverse Rs0

下表列出了可能用到的 flag:

f_B	f_SM
f_S	f_SMR
f_W	f_SMW
f_D	f_Enable
f_2D	f_Disable
f_4D	f_Mode0
f_8D	f_Mode1
f_LB	f_RPORT
f_HB	f_WPORT
f_U	f_Col
f_SUB1	f_Clean

2.2.2. SCU

2.2.2.1. 定点加法指令

Syntax SCU: Rd = Rs0 + Rs1 {(U)} {(T)} {(CI)} {(Flag)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	1	T	U	Flag	CI	COND	0	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 进行 32 位定点数加法运算, 其结果值放入寄存器 Rd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- T 为可选项, 表示结果截断, 否则饱和处理.
- CI 为可选项, 表示溢出标志位参与计算, 否则不参与.
- Flag 为可选项, 表示更新标志位(Flag)寄存器, 否则不更新. 该指令可以更新的 Flag 位宽为 2. 其中, Flag[1]为 CFlag, Flag[0]为 VFlag. CFlag 为 1 时表示结果溢出, 为 0 时表示不溢出; VFlag 为 1 时表示下溢, 为 0 时表示上溢.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: CFlag 为 1 时 VFlag 有效, CFlag 为 0 时 VFlag 无效.

Execution

```
if (COND != null && !COND) {
    return; // 带条件且条件不成立, 指令不执行
}

if (T) {
    if (U) {
        Rd = truncate (CI ?
            ((unsigned)Rs1 + (unsigned)Rs0 + (unsigned)CI)
            : ((unsigned)Rs1 + (unsigned)Rs0));
    } else {
        Rd = truncate (CI ? (Rs1 + Rs0 + CI) : (Rs1 + Rs0));
    }
} else {
```

```
if (U) {
    Rd = saturate (CI ?
                    ((unsigned)Rs1 + (unsigned)Rs0 + (unsigned)CI)
                    : ((unsigned)Rs1 + (unsigned)Rs0));
} else {
    Rd = saturate (CI ? (Rs1 + Rs0 + CI) : (Rs1 + Rs0));
}
}

if (Flag) {
    update_flag(); // 更新 Flag 寄存器
}
```

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R3 + R5;
SCU: R1 = R4 + R6 (CI) (C1)

2.2.2.2. 定点减法指令

Syntax SCU: Rd = Rs0 - Rs1 {(U)} {(T)} {(CI)} {(Flag)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	1	T	U	Flag	CI	COND	0	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 进行 32 位定点数减法运算, 其结果值放入寄存器 Rd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- T 为可选项, 表示结果截断, 否则饱和处理.
- CI 为可选项, 表示溢出标志位参与计算, 否则不参与.
- Flag 为可选项, 表示更新标志位(Flag)寄存器, 否则不更新. 该指令可以更新的 Flag 位宽为 2. 其中, Flag[1] 为 CFlag, Flag[0] 为 VFlag. CFlag 为 1 时表示结果溢出, 为 0 时表示不溢出; VFlag 为 1 时表示下溢, 为 0 时表示上溢.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: CFlag 为 1 时 VFlag 有效, CFlag 为 0 时 VFlag 无效.

Execution

```

if (COND != null && !COND) {
    return; // 带条件且条件不成立, 指令不执行
}

if (T) {
    if (U) {
        Rd = truncate (CI ?
            ((unsigned)Rs0 - (unsigned)Rs1 - (unsigned)CI)
            : ((unsigned)Rs0 - (unsigned)Rs1));
    } else {
        Rd = truncate (CI ? (Rs0 - Rs1 - CI) : (Rs0 - Rs1));
    }
} else {
    if (U) {
        Rd = saturate (CI ?
            ((unsigned)Rs0 - (unsigned)Rs1 - (unsigned)CI)
            : ((unsigned)Rs0 - (unsigned)Rs1));
    }
}

```

```
    } else {
        Rd = saturate (Cl ? (Rs0 - Rs1 - Cl) : (Rs0 - Rs1));
    }
}

if (Flag) {
    update_flag(); // 更新 Flag 寄存器
}
```

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: $R2 = R3 - R5;$
SCU: $R1 = R4 - R6$ (Cl) (C1)

2.2.2.3. 定点乘法指令

Syntax SCU: Rd = Rs0 * Rs1 {(U)} {(T)} {(I)} {(Flag)} {(NM)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	NM	T	U	Flag	I	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 和 Rs1 的值相乘, 其结果值放入寄存器 Rd 中.

- U 为可选项, 表示无符号定点数据格式, 默认为有符号定点数据格式.
- T 为可选项, 整数模式下表示对结果进行截断处理, 默认进行饱和处理, 小数模式下表示对结果进行截断处理, 默认进行就近舍入.
- I 为可选项, 表示进行整数乘, 结果保留低 32 比特, 默认为小数乘, 结果保留高 32 比特.
- NM 为可选项, 存在时表示不移位, 默认为左移一位, I, U 存在时 NM 选项无效.
- Flag 为可选项, 表示更新它的 Flag 寄存器. 该指令可以更新的 Flag 位宽为 2. 其中, Flag[1]为 CFlag, Flag[0]为 VFlag. CFlag 为 1 时表示结果溢出, 为 0 时表示不溢出; VFlag 为 1 时表示下溢, 为 0 时表示上溢.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: CFlag 为 1 时 VFlag 有效, CFlag 为 0 时 VFlag 无效.

Execution

Pipeline FT, DP, EX0, EX1,EX2

Latency 4

Example SCU: R3 = R1 * R2 (NM);
SCU: R4 = R2 * R6 (T)(I)

2.2.2.4. 定点除法指令

Syntax

SCU: $Rd = Rs0 / Rs1 \{(U)\} \{(COND)\}$
 or
 SCU: $Rd = Rs0 \% Rs1 \{(U)\} \{(COND)\}$
 or
 SCU: $Dd = DIVREM Rs0, Rs1 \{(U)\} \{(COND)\}$

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	U	0	ACTION	0	COND	1	1	1		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
ACTION	DIVREM: 00 /: 01 %: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 除法运算指令. Rs0 寄存器的值作为被除数, Rs1 寄存器的值作为除数, 其结果值放入寄存器 Rd(或 Dd)中.

- U 为可选项, 表示为无符号定点除法运算, 默认为有符号定点除法运算.
 - 1)当 U 不存在时, 被除数为 0x8000_0000, 除数为 0xffff_ffff, 此时结果溢出, 商为 0x7fff_ffff, 余数为 0.
 - 2)当除数为 0 时, 结果溢出, U 存在时, 商为 0xffff_ffff, 余数为被除数; U 不存在时, 商为 0x7fff_ffff 与被除数符号位异或的结果, 余数为被除数.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

对于 DIVREM 指令, Dd 为一对 32 位寄存器分别为 Rd 和 Rd+1, 其中 Rd 保存商, Rd+1 保存余数.

Execution

Pipeline FT, DP, EX0, EX1, EX2, ..., EX18

Latency 20

Example SCU: $R3 = R1 / R2;$
SCU: $R4 = R5 \% R6(U);$
SCU: $D5 = \text{DIVREM } R1, R2(C1)$

2.2.2.5. 浮点加法指令

Syntax SCU: Rd = Rs0 + Rs1 (S) {(T)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	T	0	0	0	COND	0	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		Rs1						Rs0				Rd			

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 和 Rs1 中的单精度浮点值相加, 其结果值放入寄存器 Rd 中.

- S 为必选项, 表示单精度浮点数据格式.
- T 为可选项, 表示对结果进行向偶数截断, 默认为向零截断.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1,EX2

Latency 4

Example SCU: R1 = R2 + R3 (S)(T);
SCU: R2 = R4 + R5 (S)(C0)

2.2.2.6. 浮点减法指令

Syntax SCU: Rd = Rs0 - Rs1 (S) {(T)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	T	0	0	0	COND	0	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 和 Rs1 中的单精度浮点值相减, 其结果值放入寄存器 Rd 中.

- S 为必选项, 表示单精度浮点数据格式.
- T 为可选项, 表示对结果进行向偶数截断, 默认为向零截断.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1, EX2

Latency 4

Example SCU: R1 = R2 - R3 (S)(T);
SCU: R2 = R4 - R5 (S)(C0)

2.2.2.7. 浮点乘法指令

Syntax SCU: Rd = Rs0 * Rs1 (S) {(T)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	T	0	0	0	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 和 Rs1 中的单精度浮点值相乘, 其结果值放入寄存器 Rd 中.

- S 为必选项, 表示单精度浮点数据格式.
- T 为可选项, 表示对结果进行向偶数截断, 默认为向零截断.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1,EX2

Latency 4

Example SCU: R1 = R2 * R3 (S)(T);
SCU: R2 = R4 * R5 (S)(C0)

2.2.2.8. 类型转换指令: 定点 -> 单精度浮点

Syntax SCU: Rd = SINGLE Rs0 {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	0	U	0	0	COND	1	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1		Rs0				0	0	0	0	0		Rd			

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 中存放的定点数据转换成单精度浮点数据, 并存放在寄存器 Rd 中. 结果进行就近舍入.

- U 为可选项, 表示无符号定点数据格式, 默认为有符号定点数据格式.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1, EX2

Latency 4

Example SCU: R1 = SINGLE R2 (U);
SCU: R3 = SINGLE R4

2.2.2.9. 类型转换指令: 单精度浮点 -> 定点

Syntax SCU: Rd = INT Rs0 {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	1	0	U	0	0	COND	1	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1								Rs0	0	0	0	0	0		Rd

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 中存放的单精度浮点数据转换成定点数据, 并存放在寄存器 Rd 中. 结果进行就近舍入.

当浮点数据大小超过定点表示的范围时, 定点数据用最大值表示, 即正数为 0x7FFFFFFF, 负数为 0x80000000.

- U 为可选项, 表示转化成无符号定点数据, 默认转换为有符号定点数据.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: 对于一些特殊的单精度浮点数, 转换遵循以下对应关系:

浮点数据 (源数据)	定点数据 (结果)
+INF, NaN(无 U)	0x7fffffff
-INF, NaN(无 U)	0x80000000
NaN(有 U)	0xffffffff
非规格化数	0
阶码 ≥ 31 , 符号位为 0 (无 U)	0x7fffffff
阶码 ≥ 31 , 符号位为 1 (无 U)	0x80000000
阶码 ≥ 32 , 符号位为 0 (有 U)	0xffffffff
符号位为 1 (有 U)	0

Execution

Pipeline FT, DP, EX0, EX1, EX2

Latency 4

Example SCU: R2 = INT R1;
SCU: R3 = INT R2(U)

2.2.2.10. 定点绝对值指令

Syntax SCU: Rd = ABS Rs0 {(T)} {(Flag)} {(COND)}

Note: COND = C0 or C1 or !C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	1	T	0	Flag	0	COND	0	1	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															Rd

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的定点数据进行求绝对值处理, 并把结果放在寄存器 Rd 中.

- T 为可选项, 表示对结果进行截断处理, 默认进行饱和处理.
- Flag 为可选项, 表示更新标志位(Flag)寄存器, 否则不更新. 该指令可以更新的 Flag 位宽为 2. 其中, Flag[1]为 CFlag, Flag[0]为 VFlag. CFlag 为 1 时表示结果溢出, 为 0 时表示不溢出; VFlag 为 1 时表示下溢, 为 0 时表示上溢.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: CFlag 为 1 时 VFlag 有效, CFlag 为 0 时 VFlag 无效.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_abs (int s0)	SCU: Rd = ABS Rs0

int __ucps2_absT (int s0)	SCU: Rd = ABS Rs0 (T)
---------------------------	-----------------------

Example SCU: R1 = ABS R2 (T) (Flag);
SCU: R4 = ABS R2(!C0);

2.2.2.11. 浮点绝对值指令

Syntax SCU: Rd = ABS Rs0 (S) {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	0	0	0	0	COND	0	1	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs0				0	0	0	0	0	0	Rd				

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的单精度浮点数据进行求绝对值处理，并把结果放在寄存器 Rd 中。

- S 为必选项，表示单精度浮点数据格式。
- COND 为可选项，表示该指令的执行条件。当表示执行条件的表达式值为 1，指令执行；否则不执行。不写 COND 表示无条件执行。

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = ABS R1 (S);
SCU: R2 = ABS R1 (S)(C1);

2.2.2.12. 读 RFC 标志位指令

Syntax SCU: Rd = RFC {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	0	0	0	0	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0	0	0	Rd				

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 读标志位寄存器 RFC, 结果写入寄存器 Rd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- SCU 的标志位(Flag) 位宽为 3.
- RFC[0]为定点运算中的 VFlag, 其为 1 时表示下溢, 为 0 时表示上溢.
- RFC[1]为定点运算中的 CFlag, 其为 1 时表示结果溢出, 为 0 时表示不溢出.
- RFC[2]为 AGU 或 SEQ 访问非法地址的标志, 其为 1 表示存在访问非法地址的操作.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_readSCURFC (void)	SCU: Rd = RFC

Example SCU: R4 = RFC;
SCU: R5 = RFC(C0)

2.2.2.13. 写 RFC 标志位指令

Syntax SCU: RFC = Rs0 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	0	0	1	0	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0						Rs0		0	0	0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 读寄存器 Rs0, 结果写入标志位寄存器 RFC 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_writeSCURFC (int s0)	SCU: RFC = Rs0

Example SCU: RFC = R1;
SCU: RFC = R3(C0)

2.2.2.14. 条件寄存器读指令

Syntax SCU:Rd = ReadC0 | ReadC1{(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	C1/C0	0	0	0	0	COND	1	1	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0										Rd				

Encode field:

Field	Value
C1/C0	0:ReadC0 1:ReadC1
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对条件寄存器 Cond 进行读操作，并把结果放在寄存器 Rd 中.

- ReadC0 表示读条件寄存器 C0, ReadC1 表示读条件寄存器 C1.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R4 = ReadC0;
SCU: R2 = ReadC1(C1)

2.2.2.15. 按位与指令

Syntax SCU: Rd{, Cd} = Rs0 & Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
ACTION	&: 01
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 和 Rs1 中的数据进行按位与操作, 结果写到寄存器 Rd 中.

若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 & R4(C1);
SCU: R3, C0 = R5 & R1

2.2.2.16. 按位或指令

Syntax SCU: Rd{, Cd} = Rs0 | Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1						Rs1			Rs0				Rd		

Encode field:

Field	Value
ACTION	! : 00
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 和 Rs1 中的数据进行按位或操作，结果写到寄存器 Rd 中。

若可选项 Cd 存在，结果最低位也同时写入寄存器 Cd 中。

- COND 为可选项，表示该指令的执行条件。当表示执行条件的表达式值为 1，指令执行；否则不执行。不写 COND 表示无条件执行。

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 | R4(C1);

SCU: R3, C0 = R5 | R1

2.2.2.17. 按位异或指令

Syntax SCU: Rd{, Cd} = Rs0 ^ Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
ACTION	^: 10
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 和 Rs1 中的数据进行按位异或操作，结果写到寄存器 Rd 中。若可选项 Cd 存在，结果最低位也同时写入寄存器 Cd 中。

- COND 为可选项，表示该指令的执行条件。当表示执行条件的表达式值为 1，指令执行；否则不执行。不写 COND 表示无条件执行。

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 ^ R4(C1);
SCU: R3, C0 = R5 ^ R1

2.2.2.18. 按位非指令

Syntax SCU: Rd{, Cd} = ~Rs0 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1						Rs0	0	0	0	0	0				Rd

Encode field:

Field	Value
ACTION	~: 11
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的数据进行按位非操作, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = ~R1(C1);
SCU: R3, C0 = ~R5

2.2.2.19. 定点等于指令

Syntax SCU: Rd{, Cd} = Rs0 == Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	1	0	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的有符号定点值, 相等时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 == R5(!C0);
SCU: R3, C1 = R4 == R5

2.2.2.20. 定点不等于指令

Syntax SCU: Rd{, Cd} = Rs0 != Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	1	1	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的有符号定点值, 不相等时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 != R5(!C0);
SCU: R3, C1 = R4 != R5

2.2.2.21. 定点大于等于指令

Syntax SCU: Rd{, Cd} = Rs0 >= Rs1 {(U)} {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	1	1	U	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的定点值, Rs0 大于等于 Rs1 时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 >= R5 (U);
SCU: R3, C1 = R4 >= R5 (!C0)

2.2.2.22. 定点小于指令

Syntax SCU: Rd{, Cd} = Rs0 < Rs1 {(U)} {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	1	0	U	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的定点值, Rs0 小于 Rs1 时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 < R5 (U);
SCU: R3, C1 = R4 < R5 (!C0)

2.2.2.23. 浮点等于指令

Syntax SCU: Rd{, Cd} = Rs0 == Rs1 (S) {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	0	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的单精度浮点值, 相等时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- S 为必选项, 表示单精度浮点数据格式.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 == R5 (S) (!C0);
SCU: R3, C1 = R4 == R5 (S)

2.2.2.24. 浮点不等于指令

Syntax SCU: Rd{, Cd} = Rs0 != Rs1 (S) {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	1	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的单精度浮点值, 不相等时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- S 为必选项, 表示单精度浮点数据格式.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 != R5 (S) (!C0);
SCU: R3, C1 = R4 != R5 (S)

2.2.2.25. 浮点大于等于指令

Syntax SCU: Rd{, Cd} = Rs0 >= Rs1 (S) {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	1	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的单精度浮点值, Rs0 大于等于 Rs1 时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- S 为必选项, 表示单精度浮点数据格式.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 >= R5 (S) (!C0);
SCU: R3, C1 = R4 >= R5 (S)

2.2.2.26. 浮点小于指令

Syntax SCU: Rd{, Cd} = Rs0 < Rs1 (S) {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	0	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的单精度浮点值, Rs0 小于 Rs1 时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- S 为必选项, 表示单精度浮点数据格式.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R2 = R1 < R5 (S) (!C0);
SCU: R3, C1 = R4 < R5 (S)

2.2.2.27. 寄存器左移指令

Syntax SCU: Rd = Rs0 << Rs1 {(U)} {(CL)} {(T)} {(Us1)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	Us1	CL	U	T	0	COND	1	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs0 中的值左移 Rs1 位, 结果写入 Rd 中

- Us1 为可选项, 表示 Rs1 为无符号数, 否则为有符号数 (只看 Rs1 的低 6 位, 根据 Rs1 的第 5bit 判断其正负). 当 Us1 不存在且 Rs1 为负数时, 该指令对寄存器 Rs0 中的数据右移-Rs1, 注意当 Rs1 低 6 位为 6' h20 时, 右移 32 位; 当 Us1 存在时, 或 Us1 不存在且 Rs1 为正数时, 对寄存器 Rs0 中的数据进行左移, 左移的位数取寄存器 Rs1 的低 5 位.
- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 否则不循环.
- T 为可选项, 当左移时, 该选项存在表示截断操作, 不存在表示进行饱和操作; 当右移时, 该选项存在表示对右移结果进行截断, 不存在表示进行就近舍入.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R4 = R2 << R1 (U);
SCU: R5 = R3 << R2 (CL)

2.2.2.28. 立即数左移指令

Syntax SCU: Rd = Rs0 << uimm5 {(U)} {(CL)} {(T)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	CL	U	T	0	COND	1	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	uimm5				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的数据进行左移, 左移的位数为立即数 uimm5, 左移后的数据写到寄存器 Rd 中.

- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 默认不循环. 在非循环移位中, U 存在时, 高位补 0, 否则高位扩展符号位.
- T 为可选项, 表示对结果进行截断, 不存在时进行饱和操作.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R4 = R2 << 12 (U);
SCU: R5 = R3 << 4 (CL)

2.2.2.29. 寄存器右移指令

Syntax SCU: Rd = Rs0 >> Rs1 {(U)} {(CL)} {(T)} {(Us1)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	Us1	CL	U	T	1	COND	1	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	Rs1	Rs0	Rd
---	-----	-----	----

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs0 中的值右移 Rs1 位, 结果写入 Rd 中

- Us1 为可选项, 表示 Rs1 为无符号数, 否则为有符号数 (只看 Rs1 的低 6 位, 根据 Rs1 的第 5bit 判断其正负). 当 Us1 不存在且 Rs1 为负数时, 该指令对寄存器 Rs0 中的数据左移-Rs1, 注意当 Rs1 低 6 位为 6' h20 时, 左移 32 位; 当 Us1 存在时, 或 Us1 不存在且 Rs1 为正数时, 对寄存器 Rs0 中的数据进行右移, 右移的位数取寄存器 Rs1 的低 5 位.
- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 否则不循环.
- T 为可选项, 当左移时, 该选项存在表示截断操作, 不存在表示进行饱和操作; 当右移时, 该选项存在表示对右移结果进行截断, 不存在表示进行就近舍入.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R4 = R2 >> R1 (U);
SCU: R5 = R3 >> R2 (CL)

2.2.2.30. 立即数右移指令

Syntax SCU: Rd = Rs0 >> uimm5 {(U)} {(CL)} {(T)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	CL	U	T	1	COND	1	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	uimm5				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的数据进行右移, 右移的位数为立即数 uimm5, 右移后的数据写到寄存器 Rd 中.

- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 默认不循环. 在非循环移位中, U 存在时, 高位补 0, 否则高位扩展符号位.
- T 为可选项, 表示对结果进行截断, 不存在时进行就近舍入.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R4 = R2 >> 12 (U) (T);
SCU: R5 = R3 >> 4 (CL)

2.2.2.31. 立即数赋值指令

Syntax SCU: Rd = Imm16 {(L)} {(E)} {(U)} {(COND)}

Note:

COND = C0 or C1 or !C0

Imm16 = 16 bits immediate or %Label

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	1	1	Imm16[15: 14]	U	E	L	COND	Imm16[13:11]				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm16[10:0]														Rd	

Encode field:

Field	Value
COND	C0: 00
	C1: 01
	!C0: 10
	null: 11

Description 把 16 位立即数的值传送到 32 位寄存器 Rd(其中 Rd 不能为 R0).

- L 为可选项, 表示存入寄存器 Rd 低 16 位, 否则存入寄存器 Rs 高 16 位.
- U 为可选项, 表示数据为无符号数, 默认为有符号数.
- E 为可选项, 表示进行符号扩展, 默认不扩展.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.
当 E, L 选项同时存在, 立即数放在 Rs 的低 16bit, 高位 16bit 为符号扩展, 无符号数高位为 0; E 选项存在, L 选项不存在, 立即数放在 Rd 的高 16bit, 低 16bit 清 0.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R4 = 12 (L) (E) (U);
SCU: R5 = 6

2.2.2.32. BitFilter 指令

Syntax SCU: Rd = BitFilter Rs0, Rs1 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	1	0	0	0	0	COND	1	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	Rs1	Rs0	Rd
---	-----	-----	----

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对于寄存器 Rs1 中值为 1 的 bit, 将 Rs0 中对应位置的 bit 值筛选出来, 放到结果的低位, 高位补零, 放入寄存器 Rd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: Rs1 中为 0 的位, Rs0 中对应位上的值也应该为 0, 否则结果会出错.

Execution

Pipeline FT, DP, EX0, EX1,EX2

Latency 4

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_bitFilter (int s0, int s1)	SCU: Rd = BitFilter Rs0,Rs1

Example SCU: R4 = BitFilter R1, R2 (C0);

2.2.2.33. BitExpd 指令

Syntax SCU: Rd = BitExpd Rs0, Rs1 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	0	0	0	0	COND	1	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 从寄存器 Rs1 的最低位开始, 若该位是 0, 则 Rs0 对应位填 0, 并将 Rs0 向高位移动一位; 若该位是 1, 则 Rs0 保持原值. 如此依次往高位方向根据 Rs1 的每一比特进行一次操作, 重复 32 次, 结果放入 Rd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

例如:

Rs1: (高位)01011010(低位)

Rs0: (高位)00110110(低位)

结果: (高位)01100000(低位)

1. 从 Rs1 的最低位起, 第 0 位为 0, 所以 Rs0 相应位为 0, 并且以该位为最低位的原子串左移一位并舍弃最高位, 中间结果为 01101100;
2. Rs1 的第 1 位为 1, 所以中间结果不变 01101100;
3. Rs1 的第 2 位为 0, 所以中间结果相应位为 0, 并且以该位为最低位的原子串左移一位并舍弃最高位, 中间结果为 11011000;
4. Rs1 的第 3 位为 1, 所以中间结果不变 11011000;
5. Rs1 的第 4 位为 1, 所以中间结果不变 11011000;
6. Rs1 的第 5 位为 0, 所以中间结果相应位为 0, 并且以该位为最低位的原子串左移一位并舍弃最高位, 中间结果为 10011000;
7. Rs1 的第 6 位为 1, 所以中间结果不变 10011000;
8. Rs1 的第 7 位为 0, 所以中间结果相应位为 0, 并且以该位为最低位的原子串左移一位并舍弃最高位, 最终结果为 00011000;
9. 其他位依此类推

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency 5

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_bitExpd (int s0, int s1)	SCU: Rd = BitExpd Rs0,Rs1

Example SCU: R5 = BitExpd R3, R8 (C1)

2.2.2.34. MergeShift 指令

Syntax SCU: Rd = MergShift Rs0, Rs1, Rs2 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0			Rs2		COND	0	1	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1		Rs1				Rs0				Rd					

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 与寄存器 Rs1 的值拼接成 64 位中间结果, 其中寄存器 Rs0 的值在 64 位的高位, 寄存器 Rs1 的值在 64 位的低位. 根据寄存器 Rs2 值的低 5 位对拼接的结果进行右移, 将移位结果的低 32 位放入寄存器 Rd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_mergeShift (int s0, int s1, int s2)	SCU: Rd = MergShift Rs0, Rs1, Rs2

Example SCU: R4 = MergShift R3, R2, R1

2.2.2.35. Count 指令

Syntax SCU: Rd = CountN Rs0 {(COND)}

Note:

CountN = Count0 or Count1

COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	N	0	0	0	0	COND	0	1	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															Rd

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 数寄存器 Rs0 中 0/1 的个数, 其结果值放入寄存器 Rd 中.

- Count1 表示数 1 的个数, Count0 表示数 0 的个数.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1, EX2

Latency 4

Intrinsics	Intrinsic	Generated Instruction
	int __builtin_popcount(int s0)	SCU: Rd = Count1 Rs0

Example SCU: R6 = Count0 R2;
SCU: R7 = Count1 R3 (!C0)

2.2.2.36. BitReverse 指令

Syntax SCU: Rd = BitReverse Rs0 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1	0	0	0	0	0	COND	0	1	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		Rs0				0	0	0	0	0			Rd		

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs0 中的 32bit 数据反序后放入 Rd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __builtin_bitreverse32(int s0)	SCU: Rd = BitReverse Rs0

Example SCU: R13 = BitReverse R3

2.2.2.37. First 指令

Syntax SCU: Rd = First Rs0 {(U)} {(SUB1)} {(COND)}

Note:

COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	SUB1	U	0	0	COND	0	1	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1		Rs0				0	0	0	0	0		Rd			

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 从高位到低位找第一个 1/0, 其结果值放入寄存器 Rd 中.

- U 为可选项, 表示 Rs0 为无符号数.

当 U 存在时, 找出寄存器 Rs0 中第一个 1 前面 0 的个数(从高位开始查找). 当可选项 SUB1 不存在时, 输出第一个 1 前边 0 的个数; 当 SUB1 存在时, 输出第一个 1 前边 0 的个数减 1 的值. 当不存在 1 时, 即数据为全 0 时, 当可选项 SUB1 不存在时, 结果为 32; 当 SUB1 存在时, 结果为 31. 当第一个 1 之前没有 0 时, SUB1 不存在结果为 0; SUB1 存在时结果为 63.

当 U 不存在时, 表示 Rs0 为有符号数, 通过最高位确定符号位, 从而确定要找的是第一个 0 还是第一个 1. 当符号位为 0 时, 找第一个 1; 反之找第一个 0. 找到第一个 0/1 后, 数其前边 1/0 的个数, 当可选项 SUB1 不存在时, 输出第一个 0/1 前边 1/0 的个数; 当 SUB1 存在时, 输出第一个 0/1 前边 1/0 的个数减 1 的值. 当找不到 0/1 时, 即数据为全 0 或全 1 时, 当可选项 SUB1 不存在时, 结果为 32; 当 SUB1 存在时, 结果为 31.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_first (int s0, const int u, const int sub1)	SCU: Rd = First Rs0 {(U)} {(SUB1)}
	int __builtin_clz (int s0)	SCU: Rd = First Rs0 (U)

Example SCU: R2 = First R3 (SUB1);
SCU: R7 = First R5 (U)

2.2.2.38. GetSign 指令

Syntax SCU: Rd = GetSign Rs0, Rs1 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	0	0	0	0	0	COND	0	1	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 提取 Rs0 寄存器和 Rs1 寄存器中的符号位进行异或, 其结果值放入寄存器 Rd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_getSign (int s0, int s1)	SCU: Rd = GetSign Rs0,Rs1

Example SCU: R3 = GetSign R2, R1 (C1)

2.2.2.39. Select 指令

Syntax SCU: Rd = Select Rs0, Rs1, Rs2 {(T)} {(S|A|I|N|M)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	1	0			Rs2		COND	T	SAINM			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAINM		Rs1				Rs0			Rd						

Encode field:

Field	Value
SAINM	null: 000 S: 001 A: 010 I: 011 N: 100 M: 101
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 根据寄存器 Rs2 的最低 1bit 选择, 若其不为零选择寄存器 Rs1, 否则选择寄存器 Rs0, 结果放入寄存器 Rd 中.

- T 选项为可选项, 存在时表示截断处理, 默认进行饱和处理.
- 选项 S,A,I,N,M 为可选项, 其中:
 - S 选项代表 Sign 操作, 若 Rs0 和 Rs1 符号位的异或值等于选择寄存器的符号位, 则结果为选择寄存器; 若 Rs0 和 Rs1 符号位的异或值不等于选择寄存器的符号位, 则结果为负的选择寄存器的值;
 - A 选项代表选择|Rs1|或者|Rs0|
 - I 选项代表选择|Rs1|或者-|Rs0|
 - N 选项代表选择-Rs1 或者-Rs0
 - M 选项代表选择 Rs1 或者-Rs0
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SCU: R15 = Select R2, R3, R5 (S);
SCU: R14 = Select R1, R3, R5 (T)(A)

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.2.40. 寄存器 Extend 指令

Syntax SCU: Rd = Extend Rs0, Rs1, Rs2 {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1			Rs2		COND	U	0	1		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		Rs1				Rs0			Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs1 的低 5 位的值作为起始位置, 寄存器 Rs2 的低 5 位的值作为终止位置, 从 Rs0 中取出起始位置到终止位置的数据, 并扩展至 32 位后存入 Rd 寄存器中.

- 可选项 U 表示进行无符号扩展, 否则为有符号扩展.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_extend (int s0, int s1, int s2, const int u)	SCU: Rd = Extend Rs0, Rs1, Rs2 {(U)}

Example SCU: R8 = Extend R6, R7, R3 (U);
SCU: R14 = Extend R5, R2, R9

2.2.2.41. 立即数 Extend 指令

Syntax SCU:Rd = Extend Rs0, uimm5s1, uimm5s2 {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	1		uimm5s2		COND	U	1	1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1		uimm5s1		Rs0		Rd									

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 uimm5s1 的值作为起始位置, uimm5s2 的值作为终止位置, 从 Rs0 中取出起始位置到终止位置的数据, 并扩展至 32 位后存入 Rd 寄存器中.

- 可选项 U 表示进行无符号扩展, 否则进行有符号扩展
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_extend (int s0, int imms1, int imms2, const int u)	SCU: Rd = Extend Rs0, uimm5s1, uimm5s2 {(U)}

Note:

The parameter imms1/imms2 should be a constant number between 0~31.

Example SCU: R8 = Extend R6, 2, 17(U);
SCU: R14 = Extend R5, 12, 23

2.2.2.42. CRC 指令

Syntax SCU:Rd = CRC Rs0, Rs1 {(B|S)} {(CRC32C)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	0	0	0	0	CRC3 2C	Type	0	COND	1	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description Rs0 为输入的 CRC 旧值或者 CRC 初始值, Rs1 为需要进行 CRC 的数据, 结果写入 Rd 中.

- 可选项 B 表示 Rs1 为 8 位, S 表示 Rs1 为 16 位, 否则为 32 位.
- 可选项 CRC32C 表示进行 CRC32C 校验, CRC32C 多项式为:
0x1EDC6F41, 否则进行 CRC32 校验, CRC32 多项式为:
0x04C11DB7.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_crc (int s0, int s1, const int type, const int crc32c)	SCU: Rd = CRC Rs0,Rs1, {(B S)} {(CRC32C)}

Note:

The parameter type should be f_B/f_S/f_W.

Example SCU: R6 = CRC R1, R7 (B);

SCU: R21 = CRC R3, R7 (CRC32C)

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.2.43. NOP 指令

Syntax SCU: NOP

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	P	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description 无操作

Execution

Pipeline FT, DP, EX0

Latency 1

Example SCU: NOP;

2.2.3.AGU

2.2.3.1. 通用寄存器加载指令

Syntax AGU: Rd0, Rd1 = [Rs0 + Rs1] {(B)|(S)} {(U)} {(AT)|(NC)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		Rd1			COND [0]	Type	0	ATNC	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Rs0	U	Rs1	Rd0
-----	---	-----	-----

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
ATNC	AT: 11 NC: 10 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令从 Rs0 + Rs1 处 load 数据到 Rd0 中, 同时将 Rs0 + Rs1 的结果写入 Rd1 中, 若 Rd1 为 R0, 则不写入.

- 选项 B, S 表示数据的类型, B 表示 byte 型, S 表示 short 型, 否则表示 word 型.
- U 为可选项, 表示对结果进行无符号扩展后再写入 Rd, 否则进行有符号扩展.
- NC 为可选项, 表示不缓存.
- AT 为可选项, 表示原子操作; AT 选项存在时同样进行普通加法 $Rd1=Rs0+Rs1$ 运算; 原子操作只能访问外部地址空间.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM, CSU 和外部地址空间.

-
- DM 支持非对齐访问，其他存储单元不支持非对齐访问，且访问 CSU 仅支持 word 型数据粒度。对齐访问时，访存地址需与读取数据的字节粒度对齐，即必须是数据字节粒度的倍数。
 - AGU 的 load, Store 类指令，当访问非法地址时，会访问 DM0 的第一个地址，且向 SCU 的 RFC[2]发送访存异常信号。

Execution

Pipeline FT, DP, EX0, EX1,.....,Ex8

Latency Load IM/ShareMemory: 8

Load DM: 10

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_atomicld(void *addr, const int type)	AGU: Rd,R0 = [Rs0+Rs1] {(B) (S)} (AT)
	int __ucps2_load_nocache(void *addr, const int type)	AGU: Rd,R0 = [Rs0+Rs1] {(B) (S)} (NC)

Note:

The parameter type should be f_B/f_S/f_W.

Example AGU: R4, R5 = [R0 + R1] (B) (U);
AGU: R7, R8 = [R2 + R3] (NC)

2.2.3.2. 通用寄存器存储指令

Syntax AGU: [Rs0 + Rs1], Rd = Rs2 {(B)|(S)} {(AT)|(NC)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		Rd		COND [0]	Type	0	ATNC	1				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rs0				0	Rs1				Rs2						

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
ATNC	AT: 11 NC: 10 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令将 Rs2 中的数据 store 到 Rs0 + Rs1 地址处, 同时将 Rs0 + Rs1 的结果写入 Rd 中, 若 Rd 为 R0, 则不写入.

- 选项 B, S 表示数据的类型. B 表示 byte 型, 存储的是 Rs2 的最低 8bit 数. S 表示 short 型, 存储的是 Rs2 的最低 16bit 数. 否则表示 word 型, 存储整个 Rs2.
- NC 为可选项, 表示不缓存.
- AT 为可选项, 表示原子操作; AT 选项存在时不进行普通加法 Rd=Rs0+Rs1 运算, Rd 中存放 store 是否成功的信息, 1 表示 store 成功, 0 表示失败. 原子操作只能访问外部地址空间.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM, CSU 和外部地址空间.
- DM 支持非对齐访问, 其他存储单元不支持非对齐访问, 且访问 CSU 仅支持 word 型数据粒度. 对齐访问时, 访存地址需与读取数据的字节粒度对齐, 即必须是数据字节粒度的倍数.
- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency Store IM: 4
Store DM: 5

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_atomicst(void *addr, int s2, const int type)	AGU: [Rs0+Rs1],Rd = Rs2 {(B) (S)} (AT)
	void __ucps2_store_nocache(void *addr, int s2, const int type)	AGU: [Rs0+Rs1],R0 = Rs2 {(B) (S)} (NC)

Note:

The parameter type should be f_B/f_S/f_W.

Example AGU: [R0 + R1], R4 = R12 (S);
AGU: [R0 + R1], R4 = R12 (AT)

2.2.3.3. 定点加法指令

Syntax AGU: Rd = Rs0 + Rs1 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]	0	0	0	0	0	COND [0]	1	1	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rs0				0	Rs1				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 进行 32 位定点数加法运算, 其结果值放入寄存器 Rd 中.

- 本指令进行有符号计算, 如果计算结果溢出, 则进行截断处理.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example AGU: R2 = R3 + R5;

2.2.3.4. 定点减法指令

Syntax AGU: Rd = Rs0 - Rs1 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]	0	0	0	0	0	COND [0]	1	1	0	0	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rs0				0	Rs1				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 进行 32 位定点数减法运算, 其结果值放入寄存器 Rd 中.

- 本指令进行有符号计算, 如果计算结果溢出, 则进行截断处理.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example AGU: R2 = R3 - R5;

2.2.3.5. SVR 寄存器加载指令

Syntax AGU: Rd0, Rd1 = [SVRs0[k] + SVRs1[j]] {(B)|(S)} {(U)} {(AT)|(NC)}

Note: SVRs0, SVRs1 = SVR0 - SVR3

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	1		Rd1		0	Type	U	ATNC	SVRs0				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVRs0		k		SVRs1		j			Rd0						

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
ATNC	AT: 11 NC: 10 null: 00

Description 本指令从 SVRs0[k] + SVRs1[j] 处 load 数据到 Rd0 中，同时将 SVRs0[k] + SVRs1[j] 的结果写入 Rd1 中，若 Rd1 为 R0，则不写入。

- k, j 为 0~15 的立即数，表示选择源寄存器的第几个 Word.
- 选项 B, S 为数据的类型，B 表示 byte 型，S 表示 short 型，否则表示 word 型.
- U 为可选项，表示对结果进行无符号扩展后再写入 Rd0，否则进行有符号扩展.
- NC 为可选项，表示不缓存.
- AT 为可选项，表示原子操作；AT 选项存在时同样进行普通加法 $Rd1 = SVRs0[k] + SVRs1[j]$ 运算。原子操作只能访问外部地址空间.
- COND 为可选项，表示该指令的执行条件。当表示执行条件的表达式值为 1，指令执行；否则不执行。不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM, CSU 和外部地址空间.
- DM 支持非对齐访问，其他存储单元不支持非对齐访问，且访问 CSU 仅支持 word 型数据粒度。对齐访问时，访存地址需与读取数据的字节粒度对齐，即必须是数据字节粒度的倍数.
- AGU 的 load, Store 类指令，当访问非法地址时，会访问 DM0 的第一个地址，且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1,.....,Ex8

Latency Load IM/ShareMemory: 8
Load DM: 10

Example AGU: R4, R2 = [SVR0[2] + SVR1[10]] (B) (U);
AGU: R6, R0 = [SVR1[3] + SVR2[0]] (AT)

2.2.3.6. SVR 寄存器存储指令

Syntax AGU: [SVRs0[k] + SVRs1[j]], Rd = Rs2 { (B) | (S) } { (AT) | (NC) }

Note: SVRs0, SVRs1 = SVR0 - SVR3

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	0		Rd		0	Type	1	ATNC	SVRs0				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVRs0		k		SVRs1		j					Rs2				

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
ATNC	AT: 11 NC: 10 null: 00

Description 本指令将 Rs2 中数据 store 到 SVRs0[k] + SVRs1[j] 地址处, 同时将 SVRs0[k] + SVRs1[j] 的结果写入 Rd 中, 若 Rd 为 R0, 则不写入.

- k, j 为 0~15 的立即数, 表示选择源寄存器的第几个 Word.
- 选项 B, S 表示数据的类型. B 表示 byte 型, 存储的是 Rs2 的最低 8bit 数. S 表示 short 型, 存储的是 Rs2 的最低 16bit 数. 否则表示 word 型, 存储整个 Rs2.
- NC 为可选项, 表示不缓存.
- AT 为可选项, 表示原子操作; AT 选项存在时不进行普通加法 Rd=SVRs0[k] + SVRs1[j] 运算, Rd 中存放 store 是否成功的信息, 1 表示 store 成功, 0 表示失败. 原子操作只能访问外部地址空间.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM, CSU 和外部地址空间.
- DM 支持非对齐访问, 其他存储单元不支持非对齐访问, 且访问 CSU 仅支持 word 型数据粒度. 对齐访问时, 访存地址需与读取数据的字节粒度对齐, 即必须是数据字节粒度的倍数.
- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2] 发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency Store IM: 4
Store DM: 5

Example AGU: [SVR0[6] + SVR1[8]], R9 = R6 (S) (NC);
AGU: [SVR3[2] + SVR2[9]], R7 = R4 (AT)

2.2.3.7. SVR 寄存器加法指令

Syntax AGU: $Rd = SVRs0[k] + SVRs1[j]$

Note: $SVRs0, SVRs1 = SVR0 - SVR3$

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	1	0	0	0	0	0	0	1	1	0	0	0	SVR s0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

SVR s0	k	SVRs1	j	Rd
-----------	---	-------	---	----

Description 使用 SVRs0 和 SVRs1 寄存器进行 32 位定点数加法运算, 结果放入寄存器 Rd 中.

- k, j 为 0~15 的立即数, 表示选择源寄存器的第几个 Word.
- 本指令进行有符号计算, 如果计算结果溢出, 则进行截断处理.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example AGU: R3 = SVR0[7] + SVR0[9];

2.2.3.8. SVR 寄存器减法指令

Syntax AGU: Rd = SVRs0[k] - SVRs1[j]

Note: SVRs0, SVRs1 = SVR0 - SVR3

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	0	0	0	0	0	0	0	1	1	1	0	0	SVRs0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVRs0		k		SVRs1		j				Rd					

Description 使用 SVRs0 和 SVRs1 寄存器进行 32 位定点数减法运算, 结果放入寄存器 Rd 中.

- k, j 为 0~15 的立即数, 表示选择源寄存器的第几个 Word.
- 本指令进行有符号计算, 如果计算结果溢出, 则进行截断处理.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example AGU: R8 = SVR0[12] – SVR1[4];

2.2.3.9. 向量加载指令

Syntax AGU: SVRd0[k], Rd1 = [Rs0 + Rs1] {(D)|(2D)|(4D)} {(Col)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		Rd1		COND [0]	Type	1	0	Col	1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					Rs0	0		Rs1		SVRd0		k			

Encode field:

Field	Value
Type	D: 00 2D: 01 4D: 10 null: 11
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令从 Rs0 + Rs1 处 load 数据到 SVRd0 中, 同时将 Rs0 + Rs1 的结果写入 Rd1 中, 若 Rd1 为 R0, 则不写入.

- D, 2D, 4D 为数据类型
 - 选项 D, 表示数据类型为 1 个双字, 数据写入 SVRd0 的第 k 个双字, k 取值范围为 0~7.
 - 选项 2D, 表示数据类型为 2 个双字, 数据写入 SVRd0 的第 k 和 k+1 个双字, k 取值范围为 0,2,4,6.
 - 选项 4D, 表示数据类型为 4 个双字, 数据写入 SVRd0 的第 k, k+1, k+2 和 k+3 个双字, k 取值范围为 0,4.
 - 否则表示写入整个 SVRd0, k 取值为 0.
- Col 为可选项, 表示列模式, 默认为行模式. 列模式下 Gran 不能设为 0.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- APC 中该指令只能访问 DM 空间.
- RFM 中, 行模式下, 数据类型为 D 或 2D 时可以访问 DM 和 Share Memory, 访存地址需与读取数据的粒度对齐. RFM 不支持列模式.
- 列模式需要配置 StartAddr, EndAddr, Size, Gran, Valid 五个参数, 配置参数时的地址 bit[21:9]必须为 0x0FFF. 共可配置八组参数, 配

置哪组参数由地址的 bit[8:6]决定, 即八组配置地址分别为:

0x1f_fe00, 0x1f_fe40, 0x1f_fe80, 0x1f_fec0, 0x1f_ff00, 0x1f_ff40, 0x1f_ff80, 0x1f_ffc0. 需特别注意的是, 八组配置参数的地址空间不可有重叠, 否则会出错.

对应的参数摆放位置为:

DMWData[0]	Valid
DMWData[34:32]	Gran
DMWData[99:96]	Size
DMWData[148:128]	EndAddr, 地址 512 位对齐, 只使用高 15 位作为 EndAddr 选通
DMWData[180:160]	StartAddr, 地址 512 位对齐, 只使用高 15 位作为 StartAddr 选通

其中 EndAddr 为屏蔽位, 配置时可选择将实际访问地址的部分位屏蔽, 屏蔽后如果地址等于 StartAddr 则表示使用该参数. 例如将 EndAddr 配置为 0x1c0000, 表示屏蔽地址的低 18 位, StartAddr 配置为 0x000000, 表示屏蔽后地址为 200000, 则当访问的地址范围为 DM0(0x200000~0x23ffff)时将会使用该组参数.

CGran 在列模式时等于 Gran, 行模式时 CGran 为 0x6. Gran 在行模式下也可进行配置, Size 默认值为 0xC.

- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1,.....,Ex8

Latency Load DM: 10

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_loadV(int *addr, const int svr_w_reg, const int type, const int col)	AGU: SVRd[k], Rd = [Rs0+Rs1] {{(D) (2D) (4D)} {(Col)}}
	ucp_vec512_t __ucps2_loadV512(int *addr)	AGU: SVRd[0], R0 = [Rs0+Rs1]

Note:

The parameter type should be f_D/f_2D/f_4D/f_8D.

The parameter svr_w_reg should be SVRx_Wy (0≤x≤3, 0≤y≤15)

Example AGU: SVR3[4], R9 = [R0 + R1] (4D) (Col) ;
AGU: SVR0[0], R1 = [R4 + R5]

2.2.3.10. CSU 向量加载指令

Syntax AGU: SVRd0, Rd1 = [Rs0 + Rs1] {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		Rd1		COND [0]	SVRd0	1	1	0	1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					Rs0	0		Rs1							

Encode field:

Field	Value
COND	C0: 00
	C1: 01
	!C0: 10
	null: 11

Description 本指令从 Rs0 + Rs1 处 load 512bit 数据到 SVRd0 中, 同时将 Rs0 + Rs1 的结果写入 Rd1 中, 若 Rd1 为 R0, 则不写入.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.
- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Note: 该指令只能通过 CSU 访问 ShareMemory. 地址需要 64 字节对齐.

Execution

Pipeline FT, DP, EX0, EX1,.....,Ex6

Latency 8

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_loadV_CSU(int *addr, const int svrid)	AGU: SVRd, Rd = [Rs0+Rs1]
ucp_vec512_t __ucps2_loadV512_CSU(int *addr)	AGU: SVRd, R0 = [Rs0+Rs1]

Note:

The parameter svrid should be SVR0~SVR3

Example AGU: SVR1, R4 = [R9 + R1]

2.2.3.11. 向量存储指令

Syntax AGU: [Rs0 + Rs1], Rd = SVRs2[k] {(D)|(2D)|(4D)} {(Col)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		Rd		COND [0]	Type	1	0	Col	1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					Rs0	1		Rs1		SVRs2		k			

Encode field:

Field	Value
Type	D: 00 2D: 01 4D: 10 null: 11
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令将 SVRs2 中的数据 store 到 Rs0 + Rs1 处, 同时将 Rs0 + Rs1 的结果写入 Rd 中, 若 Rd 为 R0, 则不写入.

- D, 2D, 4D 为数据类型
 - 选项 D, 表示数据类型为 1 个双字, 将 SVRs2 的第 k 个双字写入目标地址, k 取值范围为 0~7.
 - 选项 2D, 表示数据类型为 2 个双字, 将 SVRs2 的第 k 和 k+1 个双字写入目标地址, k 取值范围为 0,2,4,6.
 - 选项 4D, 表示数据类型为 4 个双字, 将 SVRs2 的第 k, k+1, k+2 和 k+3 个双字写入目标地址, k 取值范围为 0,4.
 - 否则表示将整个 SVRs2 写入目标地址, k 取值为 0.
- Col 为可选项, 表示列模式, 默认为行模式. 列模式下 Gran 不能设为 0.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- APC 中该指令只能访问 DM 空间.
- RFM 中, 行模式下, 数据类型为 D 或 2D 时可以访问 DM 和 Share Memory, 访存地址需与写入数据的粒度对齐. RFM 不支持列模式.
- 列模式需要配置 StartAddr, EndAddr, Size, Gran, Valid 五个参数, 配置参数时的地址 bit[21:9]必须为 0x0FFF. 共可配置八组参数, 配

置哪组参数由地址的 bit[8:6]决定, 即八组配置地址分别为:

0x1f_fe00, 0x1f_fe40, 0x1f_fe80, 0x1f_fec0, 0x1f_ff00, 0x1f_ff40, 0x1f_ff80, 0x1f_ffc0. 需特别注意的是, 八组配置参数的地址空间不可有重叠, 否则会出错.

对应的参数摆放位置为:

DMWData[0]	Valid
DMWData[34:32]	Gran
DMWData[99:96]	Size
DMWData[148:128]	EndAddr, 地址 512 位对齐, 只使用高 15 位作为 EndAddr 选通
DMWData[180:160]	StartAddr, 地址 512 位对齐, 只使用高 15 位作为 StartAddr 选通

其中 EndAddr 为屏蔽位, 配置时可选择将实际访问地址的部分位屏蔽, 屏蔽后如果地址等于 StartAddr 则表示使用该参数. 例如将 EndAddr 配置为 0x1c0000, 表示屏蔽地址的低 18 位, StartAddr 配置为 0x000000, 表示屏蔽后地址为 0x200000, 则当访问的地址范围为 DM0(0x200000~0x23ffff)时将会使用该组参数.

CGran 在列模式时等于 Gran, 行模式时 CGran 为 0x6. Gran 在行模式下也可进行配置, Size 默认值为 0xC.

- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency Store DM: 5

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_storeV(int *addr, const int svr_w_reg, const int type, const int col)	AGU: [Rs0+Rs1],Rd = SVRs[k] {{(D) (2D) (4D)} {(Col)}}
	void __ucps2_storeV512(int *addr, ucp_vec512_t svr_var)	AGU: [Rs0+Rs1],R0 = SVRs[0]

Note:

The parameter type should be f_D/f_2D/f_4D/f_8D.

The parameter svr_w_reg should be SVRx_Wy (0≤x≤3, 0≤y≤15)

The parameter svr_var is a vector variable (v16i32)

Example AGU: [R0 + R3], R4 = SVR2[1] (D);

AGU: [R2 + R4], R0 = SVR0[6] (2D) (Col)

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.3.12. CSU 向量存储指令

Syntax AGU: [Rs0 + Rs1], Rd = SVRs2 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		Rd		COND [0]	SVRs2	1	1	0	1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					Rs0	1		Rs1							0

Encode field:

Field	Value
COND	C0: 00
	C1: 01
	!C0: 10
	null: 11

Description 本指令将 SVRs2 中的数据 store 到 Rs0 + Rs1 处, 同时将 Rs0 + Rs1 的结果写入 Rd 中, 若 Rd 为 R0, 则不写入.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.
- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Note: 该指令只能通过 CSU 访问 ShareMemory. 地址需要 64 字节对齐.

Execution

Pipeline FT, DP, EX0, EX1,EX2

Latency 4

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_storeV_CSU(int *addr, const int svrid)	AGU: [Rs0+Rs1],Rd = SVRs
void __ucps2_storeV512_CSU(int *addr, ucp_vec512_t svr_var)	AGU: [Rs0+Rs1],R0 = SVRs

Note:

The parameter svrid should be SVR0~SVR3

The parameter svr_var is a vector variable (v16i32)

Example AGU: [R7 + R9], R5 = SVR2

2.2.3.13. 通用寄存器立即数偏移移加载指令

Syntax AGU: Rd = [Rs + simm9] {(B)|(S)} {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]			simm9[8:4]		COND [0]	Type	1	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Rs	U	0	simm9[3:0]	Rd
----	---	---	------------	----

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令从 Rs0 + simm9 处 load 数据到 Rd 中.

- 对 simm9 进行符号扩展后再进行地址计算.
- 选项 B, S 为数据的类型, B 表示 byte 型, S 表示 short 型, 否则表示 word 型.
- U 为可选项, 表示对结果进行无符号扩展后再写入 Rd, 否则进行有符号扩展.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM, CSU 和外部地址空间.
- DM 支持非对齐访问, 其他存储单元不支持非对齐访问, 且访问 CSU 仅支持 word 型数据粒度. 对齐访问时, 访存地址需与读取数据的字节粒度对齐, 即必须是数据字节粒度的倍数.
- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1,.....,Ex8

Latency Load IM/ShareMemory: 8

Load DM: 10

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_load_ext_mem (void *addr, const int type)	AGU: Rd = [Rs+simm9] {(B) (S)}

Note:

The parameter type should be f_B/f_S/f_W.

Example AGU: R4 = [R1 + 8] (B) (U);
AGU: R6 = [R0 + 9]

2.2.3.14. 通用寄存器立即数偏移存储指令

Syntax AGU: [Rs0 + simm9] = Rs1 {(B)|(S)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		simm9[8:4]		COND [0]	Type	1	0	0	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					Rs0		0	1	simm9[3:0]						Rs1

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令将 Rs1 中数据 store 到 Rs0 + simm9 地址处.

- 对 simm9 进行符号扩展后再进行地址计算.
- 选项 B, S 表示数据的类型. B 表示 byte 型, 存储的是 Rs2 的最低 8bit 数. S 表示 short 型, 存储的是 Rs2 的最低 16bit 数. 否则表示 word 型, 存储整个 Rs2.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM, CSU 和外部地址空间.
- DM 支持非对齐访问, 其他存储单元不支持非对齐访问, 且访问 CSU 仅支持 word 型数据粒度. 对齐访问时, 访存地址需与读取数据的字节粒度对齐, 即必须是数据字节粒度的倍数.
- AGU 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency Store IM: 4

Store DM: 5

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_store_ext_mem (void *addr, int s1, const int type)	AGU: [Rs0+simm9] = Rs1 {(B) (S)}

Note:

The parameter type should be f_B/f_S/f_W.

Example AGU: [R0 + 32] = R1 (B);
AGU: [R4 + 3] = R7

2.2.3.15. 通用寄存器立即数定点加法指令

Syntax AGU: Rd = Rs + simm9 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	0	COND [1]		simm9[8:4]		COND [0]	1	1	1	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs 和 simm9 符号扩展后的值相加, 结果放入寄存器 Rd 中.

- 本指令进行有符号计算, 如果计算结果溢出, 则进行截断处理.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example AGU: R7 = R4 + 8

2.2.3.16. Merge 指令

Syntax AGU: SVRd[k] = Rs3, Rs2, Rs1, Rs0 {(LB)|(HB)|(S)}

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	k[2]		Rs3		1	k[1:0]	1	Type	SVRd [1]				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Rs2	SVRd [0]	Rs1	Rs0
-----	-------------	-----	-----

Encode field:

Field	Value
Type	LB: 00 HB: 01 S: 10 null: 11
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令将 Rs0, Rs1, Rs2, Rs3 中的数据拼接, 结果写入 SVRd[k]中. 当 Rs0, Rs1, Rs2, Rs3 四个寄存器任意一个为 R0 时, 该位置处数据不拼接, 结果中该位置处数据维持原值.

- k 表示结果写入 SVRd 的第几个双字.
- 可选项为:
 - LB: 表示从 Rs0, Rs1, Rs2, Rs3 中取出最低 Byte 的数据, 拼接后的结果为字类型, 写入 SVRd[k]的低 32 位中, k 取值范围: 0~7;
 - HB: 表示从 Rs0, Rs1, Rs2, Rs3 中取出最低 Byte 的数据, 拼接后的结果为字类型, 写入 SVRd[k]的高 32 位中, k 取值范围: 0~7;
 - S: 表示从 Rs0, Rs1, Rs2, Rs3 中取出最低 Short 的数据, 拼接后的结果为双字类型, 写入 SVRd[k]中, k 取值范围: 0~7;
 - 无 LB, HB 和 S 选项时, 默认表示取出 Rs0, Rs1, Rs2, Rs3 的所有数据拼接, 拼接后的结果为 128 位, 写入 SVRd[k]和 SVRd[k+1]中, k 取值范围: 0, 2, 4, 6.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_Merge(const int svr_w_reg, int s3, int s2, int s1, int s0, const int type)	AGU: SVRd[k] = Rs3, Rs2, Rs1, Rs0 {(LB) (HB) (S)}

Note:

The parameter svr_w_reg should be SVRx_Wy ($0 \leq x \leq 3$, $0 \leq y \leq 15$)

The parameter type should be f_LB/f_HB/f_S/f_W

Example AGU: SVR0[5] = R0, R2, R4, R6 (LB) ;
AGU: SVR2[4] = R7, R9, R11, R13

2.2.3.17. MergeR 指令

Syntax AGU: SVRd[Rs1] = Rs0 {(B)|(S)}

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	0	0	0	0	0	0	0	Type	0	SVRd	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	Rs1				Rs0					

Encode field:

Field	Value
Type	B: 01 S: 10 null: 11

Description 将 Rs0 寄存器中的数据传送至 SVRd 寄存器中. Rs1 寄存器中的值表示 SVRd 寄存器下标偏移, 单位为字节.

- B, S 为可选项: 当 B 存在时, 将 Rs0 寄存器的低 8bit 传送至 SVRd 寄存器下标对应的字节处; 当 S 存在时, 下标需 2 字节对齐, 并将 Rs0 寄存器的低 16bit 传送至 SVRd 寄存器对应位置; 默认情况下标单位需 4 字节对齐, 并将整个 Rs0 寄存器的数据传送至 SVRd 寄存器对应位置.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_MergeR(const int svrid, int s1, int s0, const int type)	AGU: SVRd[Rs1] = Rs0 {(B) (S)}

Note:

The parameter svrid should be SVR0~SVR3

The parameter type should be f_B/f_S/f_W

Example AGU: SVR2[R7] = R6 (B);
AGU: SVR0[R1] = R9

2.2.3.18. 同步指令

Syntax AGU: Synch.wait {(SM)|(SMR)|(SMW)}

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	SMx	0	0	0	0	0	0	0	0	0	0	0	0

Encode field:

Field	Value
SMx	SM: 01 SMR: 10 SMW: 11 null: 00

Description 此指令等待之前发送的 load 和 store 指令完成.

可选项为：

- SM 表示等待之前发送的访问 ShareMemory 的所有 load 和 store 指令完成；
- SMR 表示等待之前发送的访问 ShareMemory 的所有 load 指令完成；
- SMW 表示等待之前发送的访问 ShareMemory 的所有 store 指令完成；
- 默认没有可选项表示等待之前发送的访问任何 Memory 的 load 和 store 指令完成.

Execution

Pipeline FT, DP, EX0.....

Latency 拍数不确定

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_synch(const int mode)	AGU: Synch.wait{(SM) (SMR) (SMW)}

Note:

The parameter mode should be f_SM/f_SMR/f_SMW/0

Example AGU: Synch.wait (SM);
AGU: Synch.wait

2.2.3.19. NOP 指令

Syntax AGU: NOP

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description 无操作

Execution

Pipeline FT, DP, EX0

Latency 1

Example AGU: NOP;

2.2.4. SEQ

2.2.4.1. 绝对跳转指令

Syntax SEQ: IF CRs0, JUMP Rs1 {(Enable) | (Disable)}

or

SEQ: JUMP Rs0 {(Enable) | (Disable)}

Note: CRs0 = R0-R29 or !R0-!R29 or C0 or C1 or !C0 or !C1

Encoding

1. SEQ: IF CRs0, JUMP Rs1 {(Enable) | (Disable)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	NOT	Enable	Disable	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0		Rs1			0	0	0	0	0	0

2. SEQ: JUMP Rs0 {(Enable) | (Disable)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	Enable	Disable	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0		Rs1			0	0	0	0	0	0

Encode field:

Field	Value
CRs0 (bits[4:0])	R0-R29: 0 - 29 C0: 30 C1: 31
NOT (bits[24])	R0-R29, C0, C1: 0 !R0-!R29, !C0, !C1: 1

Description 绝对跳转, 改变程序执行流, 跳转到 Rs1 所指示的地址处. 根据 CRs0 的低四位是否为零来决定是否跳转.

- Enable 和 Disable 选项为可选项, 其中选项 Enable 存在时表示使能接收中断, 选项 Disable 存在时使能不接收中断, 默认(Enable 和 Disable 选项都不存在时)表示不改变中断接收状态. 只有当 JUMP 指令跳转时, 可选项才生效, 否则可选项无效.

Execution

Pipeline FT, DP, EX0,EX1,EX2.....

Latency 跳转成功: 至少 5 拍, 跳转不成功: 2 拍

Example SEQ: IF R3, JUMP R11 (Enable);

SEQ: IF !R12, JUMP R1;
SEQ: JUMP R6 (Disable)

2.2.4.2. 相对跳转指令

Syntax SEQ: IF CRs0, JUMP simm16 {(Enable) | (Disable)}

or

SEQ: JUMP simm16 {(Enable) | (Disable)}

Note:

CRs0 = R0-R29 or !R0-!R29 or C0 or C1 or !C0 or !C1

simm16 = 16 bits immediate or %Label

Encoding

1. SEQ: IF CRs0, JUMP simm16 {(Enable) | (Disable)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P 1 1 0 0 0 0 NOT Enable Disable 0 simm16[15:11]												simm16[10:0] CRs0			

2. SEQ: JUMP simm16 {(Enable) | (Disable)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P 1 1 0 0 0 0 1 Enable Disable 0 simm16[15:11]												simm16[10:0] 0 0 0 0 0			

Encode field:

Field	Value
CRs0 (bits[4:0])	R0-R29: 0 - 29 C0: 30 C1: 31
NOT (bits[24])	R0-R29, C0, C1: 0 !R0-!R29, !C0, !C1: 1

Description 相对跳转, 改变程序执行流, 跳转到当前 PC 加 (simm16<<2) 所指的地址. 根据 CRs0 的低四位是否为零来决定是否跳转.

- Enable 和 Disable 选项为可选项, 其中选项 Enable 存在时表示使能接收中断, 选项 Disable 存在时使能不接收中断, 默认(Enable 和 Disable 选项都不存在时)表示不改变中断接收状态. 只有当 JUMP 指令跳转时, 可选项才生效, 否则可选项无效.

Execution

Pipeline FT, DP, EX0,EX1,EX2.....

Latency 跳转成功: 至少 5 拍, 跳转不成功: 2 拍

Example SEQ: IF R3, JUMP 2 (Enable);
SEQ: IF !R12, JUMP %test;
SEQ: JUMP 6 (Disable)

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.4.3. 绝对函数调用指令

Syntax SEQ: IF CRs0, CALL Rs1 {((Enable) | (Disable))}
or
SEQ: CALL Rs1 {((Enable) | (Disable))}

Note: CRs0 = R0-R29 or !R0-!R29 or C0 or C1 or !C0 or !C1

Encoding

1. SEQ: IF CRs0, CALL Rs1 {((Enable) | (Disable))}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	NOT	Enable	Disable	1	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0										

2. SEQ: CALL Rs1 {((Enable) | (Disable))}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	Enable	Disable	1	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0							0	0	0	0

Encode field:

Field	Value
CRs0 (bits[4:0])	R0-R29: 0-29 C0: 30 C1: 31
NOT (bits[24])	R0-R29, C0, C1: 0 !R0-!R29, !C0, !C1: 1

Description 绝对子函数调用, 程序会调用 Rs1 处的子函数, 直到遇见返回指令. 此处返回地址默认存放在 R31 中, 返回时采用的指令为绝对跳转指令 JUMP R31. 根据 CRs0 的低四位是否为零来决定是否调用.

- Enable 和 Disable 选项为可选项, 其中选项 Enable 存在时表示使能接收中断, 选项 Disable 存在时使能不接收中断, 默认)Enable 和 Disable 选项都不存在时)表示不改变中断接收状态.

Execution

Pipeline FT, DP, EX0,EX1, EX2, EX3

Latency 5

Example SEQ: IF R3, CALL R11 (Enable);
SEQ: IF !R12, CALL R1;
SEQ: CALL R6 (Disable)

2.2.4.4. 相对函数调用指令

Syntax SEQ: IF CRs0, CALL simm16 {(Enable) | (Disable)}

or

SEQ: CALL simm16 {(Enable) | (Disable)}

Note:

CRs0 = R0-R29 or !R0-!R29 or C0 or C1 or !C0 or !C1

simm16 = 16 bits immediate or %Label

Encoding

1. SEQ: IF CRs0, CALL simm16 {(Enable) | (Disable)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	0	NOT	Enable	Disable	1		simm16[15:11]			

2. SEQ: CALL simm16 {(Enable) | (Disable)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	0	1	Enable	Disable	1		simm16[15:11]			

Encode field:

Field	Value
CRs0 (bits[4:0])	R0-R29: 0 - 29 C0: 30 C1: 31
NOT (bits[24])	R0-R29, C0, C1: 0 !R0-!R29, !C0, !C1: 1

Description 相对子函数调用, 程序会调用当前 PC 加 (simm16<<2) 处的子函数, 直到遇见返回指令. 此处返回地址默认存放在 R31 中, 返回时采用的指令为绝对跳转指令 JUMP R31. 根据 CRs0 的低四位是否为零来决定是否调用.

- Enable 和 Disable 选项为可选项, 其中选项 Enable 存在时表示使能接收中断, 选项 Disable 存在时使能不接收中断, 默认(Enable 和 Disable 选项都不存在时)表示不改变中断接收状态.

Execution

Pipeline FT, DP, EX0,EX1, EX2, EX3

Latency 5

Example SEQ: IF R3, CALL 3 (Enable);
SEQ: IF !R12, CALL %test;
SEQ: CALL 40 (Disable)

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.4.5. 循环指令

Syntax SEQ: LPTO simm16 BY Rs0 (L0|L1)

Note: simm16 = 16 bits immediate or %Label

Description 循环指令，循环结束地址为%Label 或者当前 PC 加 (simm16<<2)所指的地址. Rs0 为循环的总次数.

- L0 和 L1 选项是指采用哪一套循环硬件资源，必选其一。
 - 进入循环内后不接收任何中断请求。

此指令使用时，有如下约束：

1. 该指令只在 Jump/Loop 模式 0 时支持；
 2. 循环次数为 0(即 Rs0 值为 0)时，循环体也执行一次；
 3. 当出现循环嵌套时，必须使用不同的循环硬件资源，即 L0 和 L1，并且 L0 和 L1 不能指向同一个结束地址。

Execution

Pipeline FT, DP, EX0,EX1,EX2.....

Latency 至少 5 拍

Example SEQ: LPTO %loop_end BY R4 (L0)

2.2.4.6. SPU Stop 指令

Syntax SEQ: SPU.Stop {(Clean)}

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	0	0	0	1	0	clean	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description 执行完程序流中处于 Stop 之前的指令, SPU 核停止工作, 处于 IDLE 状态.

- 可选项 Clean 存在, 表示同时清除 icache 中的指令.

Execution

Pipeline FT, DP, EX0

Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_stop (const int clean)	SEQ: SPU.Stop {(Clean)}

Example SEQ: SPU.Stop (Clean)

2.2.4.7. Debug Break 指令

Syntax SEQ: DbgBreak

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	P	1	1	0	0	0	1	1	0	1	0	1	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description 在调试模式下使用，辅助软断点的实现。

Execution

Pipeline FT, DP, EX0

Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_dbgbreak (void)	SEQ: DbgBreak

Example SEQ: DbgBreak

2.2.4.8. 立即数中断配置指令

Syntax SEQ: IntEn {(Enable) | (Disable)}

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	0	Enable	Disable	0	1	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description 中断配置指令.

- Enable 选项存在时表示允许进行中断操作, Disable 选项存在时表示禁止进行中断操作.

Execution FT, DP, EX0

Pipeline 2

Latency

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_IntEn (const int enable_disable)	SEQ: IntEn {(Enable) (Disable)}

Example SEQ: IntEn (Enable)

2.2.4.9. 寄存器中断配置指令

Syntax SEQ: IntEn Rs0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	1	0	0	1	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Rs0

Description 中断配置指令，根据 Rs0 的值进行中断配置。Rs0 最低位为 1 表示允许进行中断操作，为 0 表示禁止进行中断操作。

Execution

Pipeline FT, DP, EX0

Latency 2

Intrinsics

	Intrinsic	Generated Instruction
	void __ucps2_IntEnRs (int s0)	SEQ: IntEn Rs0

Example SEQ: IntEn R8

2.2.4.10. 读中断配置指令

Syntax SEQ: Rd = ReadIntEn

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	1	0	0	1	0	1	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Description 将中断配置情况读到 Rd 中.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_readIntEn (void)	SEQ: Rd = ReadIntEn

Example SEQ: R9 = ReadIntEn

2.2.4.11. 读 PC 指令

Syntax SEQ: Rd = ReadPC

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	P	1	1	0	0	0	1	1	1	0	0	1	1	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	Rd		

Description 将当前指令 PC 值读到 Rd 中.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_readPC (void)	SEQ: Rd = ReadPC

Example SEQ: R5 = ReadPC

2.2.4.12. 绝对预取指令

Syntax SEQ: PreFetch Rs

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Rs
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----

Description 预取指令, 预取 PC 为 Rs 的指令, 将指令取到 ICache 中.

Execution

Pipeline FT, DP, EX0,EX1.....

Latency 指令拍数不确定

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_preFetch (int s0)	SEQ: PreFetch Rs

Example SEQ: PreFetch R14

2.2.4.13. 相对预取指令

Syntax SEQ: PreFetch simm16

Note:

simm16 = 16 bits immediate or %Label

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	P	1	1	0	0	0	0	0	1	1	0		simm16[15:11]			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	simm16[10:0]															0

Description 预取指令，预取当前 PC 加 (simm16<<2)的指令，将指令取到 ICache 中。

Execution

Pipeline FT, DP, EX0,EX1.....

Latency 指令拍数不确定

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_ preFetch (void *fn)	SEQ: PreFetch simm16

Example SEQ: PreFetch %test

2.2.4.14. ICacheConfig 指令

Syntax SEQ: ICacheConfig (DoubleEnable | DoubleDisable) (BypassEnable | BypassDisable)

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	1	0	0	1	1	1	Doub le	By pa ss
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Encode field:

Field	Value
Double	DoubleDisable: 1 DoubleEnable: 0
Bypass	BypassDisable: 1 BypassEnable: 0

Description

ICache 配置指令.

- DoubleEnable 选项存在时, 预取下一行的 CacheLine 指令到 ICache 中.
- DoubleDisable 选项存在时, 关掉预取下一行, 不再进行预取下一行操作.
- BypassEnable 选项存在时, 开启 ICache 读指令 Bypass 优化.
- BypassDisable 选项存在时, 关闭 ICache 读指令 Bypass 优化.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_iCacheConfig (const int doubleED, const int bypassED)	SEQ:ICacheConfig (DoubleEnable DoubleDisable) (BypassEnable BypassDisable)

Example SEQ: ICacheConfig (DoubleEnable) (BypassEnable)

2.2.4.15. 中断响应地址配置指令

Syntax SEQ: IntAddr Rs0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	0	1	1	1	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Rs0

Description 用 Rs0 寄存器中的数值配置顶层中断响应函数的起始地址. 该中断函数的功能为判断中断源, 并跳转至对应的中断服务程序地址执行.

Execution

Pipeline FT, DP, EX0,EX1,EX2

Latency 4

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_IntAddr (int s0)	SEQ: IntAddr Rs0

Example SEQ: IntAddr R5

2.2.4.16. 通用寄存器加载指令

Syntax SEQ: Rd = [Rs0 + Rs1] {(B)|(S)} {(U)} {(NC)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	1	1	Type	0	U	COND	NC	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	Rs0	Rs1	Rd
---	-----	-----	----

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令从 Rs0 + Rs1 处 load 数据到 Rd0 中.

- 选项 B, S 为数据的类型, B 表示 byte 型, S 表示 short 型, 否则表示 word 型.
- U 为可选项, 表示对结果进行无符号扩展后再写入 Rd, 否则进行有符号扩展.
- NC 为可选项 no cache, 存在时表示不缓存.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM 空间. DM 支持非对齐访问.
- SEQ 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1,.....,Ex8

Latency Load IM: 8

Load DM: 10

Example SEQ: R7 = [R0 + R1] (B) (U);
SEQ: R9 = [R3 + R7] (NC)

2.2.4.17. 通用寄存器存储指令

Syntax SEQ: [Rs0 + Rs1] = Rs2 {(B)|(S)} {(NC)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	1	1	Type	1	0	COND	NC	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs0				Rs1				Rs2						

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs2 中的数据 store 到地址 Rs0+Rs1 中.

- 选项 B, S 表示数据的类型. B 表示 byte 型, 存储的是 Rs2 的最低 8bit 数. S 表示 short 型, 存储的是 Rs2 的最低 16bit 数. 否则表示 word 型, 存储整个 Rs2.
- NC 为可选项 no cache, 存在时表示不缓存.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM 空间. DM 支持非对齐访问.
- SEQ 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency Store IM: 4

Store DM: 5

Example SEQ: [R0 + R15] = R23 (B) (NC);
SEQ: [R13 + R16] = R22

2.2.4.18. 通用寄存器立即数偏移移加载指令

Syntax SEQ: Rd = [Rs0 + simm9] {(B)|(S)} {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	1	0	Type	0	U	COND	simm9[8:6]				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
simm 9 [5]	Rs0				simm9[4:0]				Rd						

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令从 Rs0 + simm9 处 load 数据到 Rd 中.

- 对 simm9 进行符号扩展后再进行地址计算.
- 选项 B, S 为数据的类型, B 表示 byte 型, S 表示 short 型, 否则表示 word 型.
- U 为可选项, 表示对结果进行无符号扩展后再写入 Rd, 否则进行有符号扩展.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM 空间. DM 支持非对齐访问.
- SEQ 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1,.....,Ex8

Latency Load IM: 8

Load DM: 10

Example SEQ: R25 = [R11 + 8] (S) (U);

SEQ: R2 = [R5 + 3]

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.4.19. 通用寄存器立即数偏移存储指令

Syntax SEQ: [Rs0 + simm9] = Rs1 {(B)|(S)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	1	0	Type	1	0	COND	simm9[8:6]				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
simm 9 [5]	Rs0				simm9[4:0]				Rs1						

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 本指令将 Rs1 中数据 store 到 Rs0 + simm9 地址处.

- 对 simm9 进行符号扩展后再进行地址计算.
- 选项 B, S 表示数据的类型. B 表示 byte 型, 存储的是 Rs2 的最低 8bit 数. S 表示 short 型, 存储的是 Rs2 的最低 16bit 数. 否则表示 word 型, 存储整个 Rs2.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- 该指令可以访问 IM, DM 空间. DM 支持非对齐访问.
- SEQ 的 load, Store 类指令, 当访问非法地址时, 会访问 DM0 的第一个地址, 且向 SCU 的 RFC[2]发送访存异常信号.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency Store IM: 4

Store DM: 5

Example SEQ: [R8 + 92] = R19 (B);
SEQ: [R17 + 3] = R27 (S)(C0)

2.2.4.20. SeqWord 指令

Syntax SEQ: Rd = SeqWord SVRs[Addr0 , 0|Addr1, 0|Addr2, 0|Addr3]

Note:

Addr1 = Addr0 + 1

Addr2 = Addr0 + 2

Addr3 = Addr0 + 3

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	0	1	1	1	1	0	0	0	SVRs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVRs	0	Addr3	Addr2	Addr1	0	0	Addr0				Rd				

Description 分发 word 指令, 将寄存器 SVRs 中的 512bit(共 16 个 32bit)数据分发到连续的 Rd 中.

选择 SVRs 中哪段 32bit 数据由 4bit 的 Addr0 决定, 选择 SVRs 中共几个 32bit 数据进行发送由 Addr0, Addr1, Addr2, Addr3 决定.

Addr1 为 0 或 Addr0+1, Addr2 为 0 或 Addr0+2, Addr3 为 0 或 Addr0+3. 如果 Addr1~3 为 0 则不分发其对应的数据, 否则进行分发.

当 Addr1~3 全不为 0 时, 将 Addr0, Addr0+1, Addr0+2, Addr0+3 的数据分发到 Rd, Rd+1, Rd+2, Rd+3 中. 当 Addr1~3 中有一个 0 时, 将 Addr0 和 Addr1~3 中 Addr_n 不为 0 的数据分发到 Rd, Rd+1, Rd+2 中. 当 Addr1~3 中有两个 0 时, 将 Addr0 和 Addr1~3 中 Addr_n 不为 0 的数据分发到 Rd, Rd+1 中. 当 Addr1~3 全为 0 时, 将 Addr0 的数据分发到 Rd 中.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_seqWord (const int svrid, const int addr0, const int addr1, const int addr2, const int addr3)	SEQ: Rd = SeqWord SVRs[Addr0, 0 Addr1, 0 Addr2, 0 Addr3]

Note:

The parameter svrid should be SVR0~SVR3.

The parameter addr0 should be a constant number between 0~15.

Example SEQ: R1 = SeqWord SVR0[2 , 3, 4, 5];
SEQ: Rd = SeqWord SVR3[13 , 0, 15, 0]

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.4.21. SeqShort 指令

Syntax SEQ: Rd = SeqShort SVRs[Addr0 , 0|Addr1, 0|Addr2, 0|Addr3] {(U)}

Note:

Addr1 = Addr0 + 1

Addr2 = Addr0 + 2

Addr3 = Addr0 + 3

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	0	1	1	1	1	0	1	U	SVRs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVRs	0	Addr3	Addr2	Addr1	0	Addr0				Rd					

Description 分发 short 指令, 将寄存器 SVRs 中的 512bit(共 32 个 16bit)数据分发到连续的 Rd 中.

选择 SVRs 中哪段 16bit 数据由 5bit 的 Addr0 决定, 选择 SVRs 中共几个 16bit 数据进行发送由 Addr0, Addr1, Addr2, Addr3 决定.

Addr1 为 0 或 Addr0+1, Addr2 为 0 或 Addr0+2, Addr3 为 0 或

Addr0+3. 如果 Addr1~3 为 0 则不分发其对应的数据, 否则进行分发.

当 Addr1~3 全不为 0 时, 将 Addr0, Addr0+1, Addr0+2, Addr0+3 的数据分发到 Rd, Rd+1, Rd+2, Rd+3 中. 当 Addr1~3 中有一个 0 时, 将 Addr0 和 Addr1~3 中 Addr0 不为 0 的数据分发到 Rd, Rd+1, Rd+2 中. 当 Addr1~3 中有两个 0 时, 将 Addr0 和 Addr1~3 中 Addr0 不为 0 的数据分发到 Rd, Rd+1 中. 当 Addr1~3 全为 0 时, 将 Addr0 的数据分发到 Rd 中.

- U 为可选项, 存在时表示无符号操作, 默认是有符号操作. U 选项存在时 Rd 的高位扩展零, 否则扩展符号位.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_seqShort (const int svrid, const int addr0, const int addr1, const int addr2, const int addr3, const int u)	SEQ: Rd = SeqShort SVRs[Addr0, 0 Addr1, 0 Addr2, 0 Addr3] {(U)}

Note:

The parameter svrid should be SVR0~SVR3.

The parameter addr0 should be a constant number between 0~31.

Example SEQ: R7 = SeqShort SVR1[12 , 13, 14, 15] (U);
 SEQ: R9 = SeqShort SVR2[28 , 0, 30, 31]

2.2.4.22. SeqByte 指令

Syntax SEQ: Rd = SeqByte SVRs[Addr0 , 0|Addr1, 0|Addr2, 0|Addr3] {(U)}

Note:

$$\text{Addr1} = \text{Addr0} + 1$$

$$\text{Addr2} = \text{Addr0} + 2$$

$$\text{Addr3} = \text{Addr0} + 3$$

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	0	1	1	1	1	1	0	U	SVRs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVRs	0	Addr3	Addr2	Addr1	Addr0				Rd						

Description 分发 byte 指令, 将寄存器 SVRs 中的 512bit(共 64 个 8bit)数据分发到连续的 Rd 中.

选择 SVRs 中哪段 8bit 数据由 6bit 的 Addr0 决定, 选择 SVRs 中共几个 8bit 数据进行发送由 Addr0, Addr1, Addr2, Addr3 决定. Addr1 为 0 或 Addr0+1, Addr2 为 0 或 Addr0+2, Addr3 为 0 或 Addr0+3. 如果 Addr1~3 为 0 则不分发其对应的数据, 否则进行分发. 当 Addr1~3 全不为 0 时, 将 Addr0, Addr0+1, Addr0+2, Addr0+3 的数据分发到 Rd, Rd+1, Rd+2, Rd+3 中. 当 Addr1~3 中有一个 0 时, 将 Addr0 和 Addr1~3 中 Addr0 不为 0 的数据分发到 Rd, Rd+1, Rd+2 中. 当 Addr1~3 中有两个 0 时, 将 Addr0 和 Addr1~3 中 Addr0 不为 0 的数据分发到 Rd, Rd+1 中. 当 Addr1~3 全为 0 时, 将 Addr0 的数据分发到 Rd 中.

- U 为可选项, 存在时表示无符号操作, 默认是有符号操作. U 选项存在时 Rd 的高位扩展零, 否则扩展符号位.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_seqByte (const int svrid, const int addr0, const int addr1, const int addr2, const int addr3, const int u)	SEQ: Rd = SeqByte SVRs[Addr0, 0 Addr1, 0 Addr2, 0 Addr3] {(U)}

Note:

The parameter svrid should be SVR0~SVR3.

The parameter addr0 should be a constant number between 0~63.

Example SEQ: R12 = SeqByte SVR1[20 , 21, 22, 23];
 SEQ: R13 = SeqByte SVR2[16 , 17, 18, 0] (U)

2.2.4.23. 寄存器传输指令: SVR -> R

Syntax SEQ: Rd = SVRs [Rs] {(U)} {(B|S)}

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	0	1	1	1	1	1	1	U	SVRs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVRs	0	0	0	Type		Rs				Rd					

Encode field:

Field	Value
Type	B: 00 S: 01 null: 10

Description SVR-R 传输指令, 将 512bit 的 SVRs 寄存器中的数据传送至 Rd 寄存器中.

- Rs 寄存器中的值表示 SVRs 寄存器下标偏移, 下标单位为字节.
- U 选项为可选项, 存在时表示结果高位补 0. 默认结果的高位进行符号扩展.
- B, S 为可选项: 当 B 存在时, 以 Rs 低 6 位作为索引, 将 64 段 8bit 数据中对应的 8bit 数据传送至 Rd 寄存器的低 8 位. 当 S 存在时, 下标需 2 字节对齐(Rs 的最低位应为 0), 以 Rs 的 1 到 5 位作为索引, 将 32 段 16bit 数据中对应的 16bit 数据传送至 Rd 寄存器的低 16 位. 默认情况下下标需 4 字节对齐(Rs 的最低两位应为 0), 以 Rs 的 2 到 5 位作为索引, 将 16 段 32bit 数据中对应的 32bit 数据传送至 Rd 寄存器.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Example SEQ: R23 = SVR2 [R3] (U) (B);
SEQ: R25 = SVR3 [R5]

2.2.4.24. 跳转模式设置指令

Syntax SEQ: SetJumpMode (mode0|mode1)

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	1	1	1	1	0	mode	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description Jump/Loop 模式设置指令, 用于设置 jump 和 loop 的执行模式.

- mode0 进行 jump 和 loop 跳转优化, 通过增加跳转缓存, 减少 jump 和 loop 跳转时的执行周期.
- mode1 进行常规的 jump 和 loop 操作, 当 jump 和 loop 需要跳转时, 重新去指令存储里取指令.

Note: mode0 时, 跳转到下一条指令时, 当下一条指令 PC 为非 16 字对齐时, jump 不做优化处理.

Execution

Pipeline FT, DP, EX0

Latency 2

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_SetJumpMode (const int mode)	SEQ: SetJumpMode(mode0 mode1)

Example SEQ: SetJumpMode (mode0)

2.2.4.25. Word 位反序指令

Syntax SEQ: Rd = BrWord Rs0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	1	1	1	0	1	1	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	0	0	Rs0				Rd						
---	---	---	---	---	---	-----	--	--	--	----	--	--	--	--	--	--

Description 将 Rs0 中的{Byte3,Byte2,Byte1,Byte0}数据以字节为单位, 反序后放入 Rd 中, 反序结果为{ Byte0,Byte1,Byte2,Byte3}.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics

	Intrinsic	Generated Instruction
	int __ucps2_brWord (int s0)	SEQ: Rd = BrWord Rs0

Example SEQ: R24 = BrWord R2

2.2.4.26. SVR 位反序指令

Syntax SEQ: SVRd = BrSVR SVRs0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	1	1	1	0	1	1	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	0	0	0	0	0	SVRs0	0	0	0	0	SVRd	
---	---	---	---	---	---	---	---	---	-------	---	---	---	---	------	--

Description 将 SVRs0 中的{Byte63,Byte62,...,Byte1,Byte0}数据以字节为单位, 反序后放入 SVRd 中, 反序的结果为{Byte0,Byte1,...Byte62,Byte63}.

Execution

Pipeline FT, DP, EX0,EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
ucp_vec512_t __ucps2_brSVR (ucp_vec512_t s0)	SEQ: SVRd = BrSVR SVRs0

Example SEQ: SVR3 = BrSVR SVR0

2.2.4.27. NOP 指令

Syntax SEQ: NOP

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description 无操作

Execution

Pipeline FT, DP, EX0

Latency 1

Example SEQ: NOP;

2.2.5. SYN

2.2.5.1. 立即数调用 MPU 指令

Syntax SYN: Rd = CallM uimm16 {(COND)}

or

SYN: CallM uimm16 (B) {{COND}}

Note:

Imm16 = 16 bits immediate or %Label

COND = C0 or C1 or !C0

Encoding 1. SYN: Rd = CallM Imm16 {{(COND)}}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	1	0	uiimm16[15]	0	uiimm16[14]	COND	uiimm16[13:11]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
uiimm16[10:0]										Rd					

2. SYN: CallM Imm16 (B) {COND}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	1	0	uimm16[15]	1	uimm16[14]	COND	uimm16[13:11]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								uimm16[10:0]			0	0	0	0	0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 调用 MPU 程序, MPU 程序的目的地址用 uimm16 表示.

- 当 B 选项不存在时, 如果 Call 成功则 Rd 返回值的最低位为 0, 如果 Call 没成功则 Rd 返回值的最低位为 1. 当 B 选项存在时, 只有当 MPU 程序调用成功时 SYN 指令槽的指令才能继续执行.

	CallIM 调用成功	CallIM 调用不成功
B 选项存在 (阻塞模式)	SPU 程序继续执行	SYN 执行 CallIM 直到调用成功
B 选项不存在	SPU 程序继续执行	SPU 程序继续执行

-
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency 5

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_callm (void* mpu_fn)	SYN: Rd = CallM uimm16
	void __ucps2_callmb (void* mpu_fn)	SYN: CallM uimm16 (B)

Example SYN: R6 = CallM %m_func;
SYN: CallM %m_func (B)

2.2.5.2. 寄存器调用 MPU 指令

Syntax SYN: Rd = CallM Rs0 {(COND)}

or

SYN: CallM Rs0 (B) {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

1. SYN: Rd = CallM Rs0 {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	0	0	0	0	0	COND	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	Rs0				Rd					

2. SYN: CallM Rs0 (B) {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	0	0	0	1	0	COND	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	Rs0				0	0	0	0	0	0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 调用 MPU 程序, 用 Rs0 寄存器指定 MPU 程序的目的地址.

- 当 B 选项不存在时, 如果 Call 成功则 Rd 返回值的最低位为 0, 如果 Call 没成功则 Rd 返回值的最低位为 1. 当 B 选项存在时, 只有当 MPU 程序调用成功时 SYN 指令槽的指令才能继续执行.

	CallM 调用成功	CallM 调用不成功
B 选项存在 (阻塞模式)	SPU 程序继续执行	SYN 执行 CallM 直到调用成功
B 选项不存在	SPU 程序继续执行	SPU 程序继续执行

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency 5

Example SYN: R7 = CallM R1;
SYN: CallM R3 (B)

2.2.5.3. MPU 状态查询指令

Syntax SYN: Rd = Stat {(B)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	0	0	0	B	0	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Rd

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 查询 MPU 状态, 将 MPU Stat 传送到 Rd 寄存器(无论是否有 B 选项, 结果都会写回 Rd). Stat 为向量状态寄存器, 位宽为 1bit.

- B 为可选项, 表示只有当 Stat 为 0 时返回, 否则停顿等待 (只阻塞 SYN, 其他指令槽不阻塞, SPU 程序继续执行). B 选项不存在表示直接返回.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_getStat (void)	SYN: Rd = Stat
Int __ucps2_getStatB (void)	SYN: Rd = Stat (B)

Example SYN: R7 = Stat;
SYN: R8 = Stat (B)

2.2.5.4. 读 FIFO 指令

Syntax SYN: Rd = FIFO.rx {(B)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	0	0	1	B	0	COND	0	1	rx[1]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rx[0]	0	0	0	0	0	0	0	0	0	0	Rd				

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 FIFO.rx 中的数据读入 Rd 寄存器中.

- B 为可选项, 表示 FIFO.rx 不为空时, 读操作成功; 否则停顿等待 (只阻塞 SYN, 其他指令槽不阻塞, SPU 程序继续执行). 若 B 选项不存在, 读取成功则 Rd 的最高位置为 0, 失败则 Rd 的最高位置为 1.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: 无论读取成不成功, Rd 的 24bit 到 30bit 清 0.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency 5

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_getFIFO(const int fifo_read_id)	SYN: Rd = FIFO.rx
int __ucps2_getFIFOB(const int fifo_read_id)	SYN: Rd = FIFO.rx (B)

Note:

The parameter fifo_read_id should be FIFO_R0/FIFO_R1/FIFO_R2/FIFO_R3.

Example SYN: R11 = FIFO.r0 ;
SYN: R12 = FIFO.r2 (B)

2.2.5.5. 写 FIFO 指令

Syntax SYN: FIFO.wx = Rs0 {(B)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	0	1	1	B	0	COND	0	1	wx[1]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
wx[0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Rs0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs0 寄存器中的数据写入 FIFO.wx 中.

- B 为可选项, 表示 FIFO.wx 为空时, 写操作成功; 否则停顿等待 (只阻塞 SYN, 其他指令槽不阻塞, SPU 程序继续执行). 若 B 选项不存在, 写入成功则 Rs0 的最高位置为 0, 失败则 Rs0 的最高位置为 1.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: 无论写入成不成功, Rs0 的 24bit 到 30bit 清 0.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_setFIFO(const int fifo_write_id, int s0)	SYN: FIFO.wx = Rs0
void __ucps2_setFIFOB(const int fifo_write_id, int s0)	SYN: FIFO.wx = Rs0 (B)

Note:

The parameter fifo_write_id should be FIFO_W0/FIFO_W1/FIFO_W2/FIFO_W3.

Example SYN: FIFO.w1 = R4;
SYN: FIFO.w2 = R5 (B)

2.2.5.6. 寄存器传输指令: SVR -> MReg

Syntax SYN: M[Rs1] = SVRs0 {(COND)}
or
SYN: M[Imm8] = SVRs0 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

1. SYN: M[Rs1] = SVRs0 {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	0	1	0	0	1	COND	1	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0						Rs1	0	0	0	SVRs0

2. SYN: M[Imm8] = SVRs0 {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	0	0	1	1	0	1	COND	1	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0									Imm8	0	0	0	SVRs0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 SVRs0 中的值存入 MReg 中, 其中 Rs1 的低 8 位或者 8 位立即数 Imm8 作为 MReg 的地址索引.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: MReg 深度不为 256 时, 不建议使用该指令.

Execution

Pipeline FT, DP, EX0, EX1,EX2

Latency 4

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_setMReg(const int svrid, int s1)	SYN: M[Rs1] = SVRs or SYN: M[Imm8] = SVRs

void __ucps2_setMReg_var(ucp_vec 512_t svr_svr, int s1)	SYN: M[Rs1] = SVRs or SYN: M[Imm8] = SVRs
---	---

Note:

The parameter svrid should be SVR0~SVR3

The parameter svr_var is a vector variable (v16i32)

Example SYN: M[R9] = SVR0;
 SYN: M[32] = SVR1

2.2.5.7. 单个 MC 配置寄存器配置指令

Syntax SYN: MC[Rs1] = Rs0 (I|S) {(COND)}

or

SYN: MC.rx = Rs0 (I|S) {(COND)}

or

SYN: MC.wx = Rs0 (I|S) {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

1. SYN: MC[Rs1] = Rs0 (I|S) {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	1	1	0	0	0	COND	0	0	I S	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	Rs1				Rs0					

2. SYN: MC.rx = Rs0 (I|S) {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	1	1	1	0	0	COND	0	0	I S	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	rx				Rs0				

3. SYN: MC.wx = Rs0 (I|S) {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	1	1	1	1	0	0	COND	0	0	I S
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	wx				Rs0				

Encode field:

Field	Value
I S	I: 0 S: 1
rx	r0: 0000 r1: 0001 r2: 0010 r3: 0011 r4: 0100 r5: 0101 r6: 0110 r7: 0111
wx	w0: 1000 w1: 1001 w2: 1010 w3: 1011 w4: 1100 w5: 1101 w6: 1110 w7: 1111
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 的数据写入 MC 配置寄存器中. 其中寄存器 Rs1 中的低 4 位或者 rx/wx 表示 MC 配置寄存器的端口号.

- I 或 S 为必选项, 表示配置参数 I 或者参数 S.

- 寄存器 Rs1 的低 4 位数据的含义:

0000: r0	1000: w0
0001: r1	1001: w1
0010: r2	1010: w2
0011: r3	1011: w3
0100: r4	1100: w4
0101: r5	1101: w5
0110: r6	1110: w6
0111: r7	1111: w7

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_setMC_I(int mcregid, int s0)	SYN: MC[Rs] = Rs0 (I) or SYN: MC.rx = Rs0 (I) or SYN: MC.wx = Rs0 (I)
	void __ucps2_setMC_S(int mcregid, int s0)	SYN: MC[Rs] = Rs0 (S) or SYN: MC.rx = Rs0 (S) or SYN: MC.wx = Rs0 (S)

Note:

The parameter mcregid should be MC_R0~MC_R7 or MC_W0~MC_W7 or a variable.

Example SYN: MC[R1] = R5 (I);
SYN: MC.r3 = R7 (S) (C1)

2.2.5.8. 全体 MC 配置寄存器配置指令

Syntax SYN: MC = SVRs0 (RPort|WPort) {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	1	1	1	0	1	COND	0	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	0	0	0	0	0	0	POR	0	0	0	0	SVRs0
T															

Encode field:

Field	Value
PORT	RPort: 1 WPort: 0
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 SVRs0 中的 512bit 数据写入 MC 配置寄存器中.

- 选项 RPort 表示配置读端口, WPort 表示配置写端口.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_setMCSR(const int svrid, const int flag_r_w)	SYN: MC = SVRs (RPort WPort)
void __ucps2_setMCSR_var(ucp_vc512_t svr_var, const int flag_r_w)	SYN: MC = SVRs (RPort WPort)

Note:

The parameter svrid should be SVR0~SVR3.

The parameter svr_var is a vector variable (v16i32)

Example MC = SVR2 (RPort)

2.2.5.9. 单个 MC 配置寄存器读取指令

Syntax SYN: Rd = MC[Rs0] (I|S) {(COND)}

or

SYN: Rd = MC.rx (I|S) {(COND)}

or

SYN: Rd = MC.wx (I|S) {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

1. SYN: Rd = MC[Rs0] (I|S) {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	1	0	0	0	0	COND	0	0	I S	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0						Rd				

2. SYN: Rd = MC.rx (I|S) {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	1	0	1	0	0	COND	0	0	I S	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0					Rd				

3. SYN: Rd = MC.wx (I|S) {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	1	0	1	0	0	COND	0	0	I S	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0					Rd				

Encode field:

Field	Value
I S	I: 0 S: 1
rx	r0: 0000 r1: 0001 r2: 0010 r3: 0011 r4: 0100 r5: 0101 r6: 0110 r7: 0111
wx	w0: 1000 w1: 1001 w2: 1010 w3: 1011 w4: 1100 w5: 1101 w6: 1110 w7: 1111
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 MC 配置寄存器的内容读到寄存器 Rd 中. 其中寄存器 Rs0 的低 4 位或者 rx/wx 表示 MC 配置寄存器的端口号.

- I 或 S 为必选项, 表示配置参数 I 或者参数 S.

- 寄存器 Rs0 的低 4 位数据的含义:

0000: r0	1000: w0
0001: r1	1001: w1
0010: r2	1010: w2
0011: r3	1011: w3
0100: r4	1100: w4
0101: r5	1101: w5
0110: r6	1110: w6
0111: r7	1111: w7

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency 5

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_getMC_I(int mcregid)	SYN: Rd = MC[Rs] (I) or SYN: Rd = MC.rx (I) or SYN: Rd = MC.wx (I)
	int __ucps2_getMC_S(int mcregid)	SYN: Rd = MC[Rs] (S) or SYN: Rd = MC.rx (S) or SYN: Rd = MC.wx (S)

Note:

The parameter mcregid should be MC_R0~MC_R7 or MC_W0~MC_W7 or a variable.

Example SYN: R8 = MC[R0] (I) ;
SYN: R4 = MC.w3 (S);

2.2.5.10. 寄存器传输指令: KI -> R

Syntax SYN: Rd = KI[Rs0] {(COND)}

or

SYN: Rd = KIs0 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

1. SYN: Rd = KI[Rs0] {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	0	0	0	0	0	COND	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	Rs0				Rd					

2. SYN: Rd = KIs0 {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	0	0	1	0	0	COND	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	KIs0				Rd				

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 KI 寄存器中的值传送至 Rd 寄存器中. 其中 Rs0 低 4 位的值或 s0(四位立即数 Imm4)表示传送的 KI 寄存器编号.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1, EX2, EX3

Latency 5

Intrinsics	Intrinsic	Generated Instruction
	int __ucps2_getKI (int ki)	SYN: Rd = KI[Rs] or SYN: Rd = KIs

Note:

The parameter ki should be KI0~KI15 or a variable.

Example SYN: R12 = KI[R12] ;
SYN: R13 = KI13

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.5.11. 寄存器传输指令: R -> KI

Syntax SYN: KI[Rs1] = Rs0 {(COND)}

or

SYN: KId = Rs0 {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

1. SYN: KI[Rs1] = Rs0 {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	0	1	0	0	0	COND	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0			Rs1				Rs0			

2. SYN: KId = Rs0 {(COND)}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	0	1	1	0	1	1	0	0	COND	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0			KId			Rs0			

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs0 寄存器的值配置至 KI 寄存器中. 其中 Rs1 低 4 位的值或 d(四位立即数 Imm4)表示配置的 KI 寄存器编号.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_setKI (int ki, int s0)	SYN: KI[Rs] = Rs0 or SYN: KId = Rs0

Note:

The parameter ki should be KI0~KI15 or a variable.

Example SYN: KI[R10] = R12;
SYN: KI10 = R13 (C0)

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.5.12. 定点加法指令

Syntax SYN: Rd = Rs0 + Rs1 {(U)} {(T)} {(CI)} {(Flag)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	1	T	U	Flag	CI	COND	0	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	Rs1	Rs0	Rd
---	-----	-----	----

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 进行 32 位定点数加法运算, 其结果值放入寄存器 Rd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- T 为可选项, 表示结果截断, 否则饱和处理.
- CI 为可选项, 表示溢出标志位参与计算, 否则不参与.
- Flag 为可选项, 表示更新标志位(Flag)寄存器, 否则不更新. 该指令可以更新的 Flag 位宽为 2. 其中, Flag[1]为 CFlag, Flag[0]为 VFlag. CFlag 为 1 时表示结果溢出, 为 0 时表示不溢出; VFlag 为 1 时表示下溢, 为 0 时表示上溢.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: CFlag 为 1 时 VFlag 有效, CFlag 为 0 时 VFlag 无效.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R3 = R2 + R1 (U) (CI);
SYN: R5 = R2 + R3 (T) (Flag)

2.2.5.13. 定点减法指令

Syntax SYN: Rd = Rs0 - Rs1 {(U)} {(T)} {(CI)} {(Flag)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	1	T	U	Flag	CI	COND	0	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 进行 32 位定点数减法运算, 其结果值放入寄存器 Rd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- T 为可选项, 表示结果截断, 否则饱和处理.
- CI 为可选项, 表示溢出标志位参与计算, 否则不参与.
- Flag 为可选项, 表示更新标志位(Flag)寄存器, 否则不更新. 该指令可以更新的 Flag 位宽为 2. 其中, Flag[1] 为 CFlag, Flag[0] 为 VFlag. CFlag 为 1 时表示结果溢出, 为 0 时表示不溢出; VFlag 为 1 时表示下溢, 为 0 时表示上溢.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: CFlag 为 1 时 VFlag 有效, CFlag 为 0 时 VFlag 无效.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R3 - R5;

SYN: R1 = R4 - R6 (CI) (C1)

2.2.5.14. 定点乘法指令

Syntax SYN: Rd = Rs0 * Rs1 {(U)} {(T)} {(I)} {(Flag)} {(NM)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	NM	T	U	Flag	I	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs0 和 Rs1 的值相乘, 其结果值放入寄存器 Rd 中.

- U 为可选项, 表示无符号定点数据格式, 默认为有符号定点数据格式.
- T 为可选项, 整数模式下表示对结果进行截断处理, 默认进行饱和处理, 小数模式下表示对结果进行截断处理, 默认进行就近舍入.
- I 为可选项, 表示进行整数乘, 结果保留低 32 比特, 默认为小数乘, 结果保留高 32 比特.
- NM 为可选项, 存在时表示不移位, 默认为左移一位, I, U 存在时 NM 选项无效.
- Flag 为可选项, 表示更新它的 Flag 寄存器. 该指令可以更新的 Flag 位宽为 2. 其中, Flag[1]为 CFlag, Flag[0]为 VFlag. CFlag 为 1 时表示结果溢出, 为 0 时表示不溢出; VFlag 为 1 时表示下溢, 为 0 时表示上溢.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note: CFlag 为 1 时 VFlag 有效, CFlag 为 0 时 VFlag 无效.

Execution

Pipeline FT, DP, EX0, EX1,EX2

Latency 4

Example SYN: R3 = R1 * R2 (NM);
SYN: R4 = R2 * R6 (T)(I)

2.2.5.15. 定点除法指令

Syntax SYN: $Rd = Rs0 / Rs1 \{(U)\} \{(COND)\}$
or
SYN: $Rd = Rs0 \% Rs1 \{(U)\} \{(COND)\}$
or
SYN: $Dd = DIVREM Rs0, Rs1 \{(U)\} \{(COND)\}$

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	U	0	ACTION	0	COND	1	1	1		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1															Rd

Encode field:

Field	Value
ACTION	DIVREM: 00 /: 01 %: 10
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 除法运算指令. Rs0 寄存器的值作为被除数, Rs1 寄存器的值作为除数, 其结果值放入寄存器 Rd(或 Dd)中.

- U 为可选项, 表示为无符号定点除法运算, 默认为有符号定点除法运算.
 - 1)当 U 不存在时, 被除数为 0x8000_0000, 除数为 0xffff_ffff, 此时结果溢出, 商为 0x7fff_ffff, 余数为 0.
 - 2)当除数为 0 时, 结果溢出, U 存在时, 商为 0xffff_ffff, 余数为被除数; U 不存在时, 商为 0x7fff_ffff 与被除数符号位异或的结果, 余数为被除数.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

对于 DIVREM 指令, Dd 为一对 32 位寄存器分别为 Rd 和 Rd+1, 其中 Rd 保存商, Rd+1 保存余数.

Execution

Pipeline FT, DP, EX0, EX1, EX2, ..., EX18

Latency 20

Example SYN: $R3 = R1 / R2;$
SYN: $R4 = R5 \% R6(U);$
SYN: $D5 = \text{DIVREM } R1, R2(C1)$

2.2.5.16. 读 RFC 标志位指令

Syntax SYN: Rd = RFC {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	0	0	0	0	0	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0	0	0	Rd				

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 读标志位寄存器 RFC, 结果写入寄存器 Rd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Note:

- SYN 的标志位(Flag) 位宽为 2.
- RFC[0]为定点运算中的 VFlag, 其为 1 时表示下溢, 为 0 时表示上溢.
- RFC[1]为定点运算中的 CFlag, 其为 1 时表示结果溢出, 为 0 时表示不溢出.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
int __ucps2_readSYNRFC (void)	SYN: Rd = RFC

Example SYN: R4 = RFC;
SYN: R5 = RFC(C0)

2.2.5.17. 写 RFC 标志位指令

Syntax SYN: RFC = Rs0 {COND}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	0	0	0	1	0	COND	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0						Rs0		0	0	0

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 读寄存器 Rs0, 结果写入标志位寄存器 RFC 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Intrinsics

Intrinsic	Generated Instruction
void __ucps2_writeSYNRFC (int s0)	SYN: RFC = Rs0

Example SYN: RFC = R1;
SYN: RFC = R3(C0)

2.2.5.18. 按位与指令

Syntax SYN: Rd{, Cd} = Rs0 & Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1						Rs1			Rs0				Rd		

Encode field:

Field	Value
ACTION	&: 01
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 和 Rs1 中的数据进行按位与操作，结果写到寄存器 Rd 中。

若可选项 Cd 存在，结果最低位也同时写入寄存器 Cd 中。

- COND 为可选项，表示该指令的执行条件。当表示执行条件的表达式值为 1，指令执行；否则不执行。不写 COND 表示无条件执行。

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R1 & R4(C1);

SYN: R3, C0 = R5 & R1

2.2.5.19. 按位或指令

Syntax SYN: Rd{, Cd} = Rs0 | Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1						Rs1			Rs0				Rd		

Encode field:

Field	Value
ACTION	!: 00
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 和 Rs1 中的数据进行按位或操作，结果写到寄存器 Rd 中。

若可选项 Cd 存在，结果最低位也同时写入寄存器 Cd 中。

- COND 为可选项，表示该指令的执行条件。当表示执行条件的表达式值为 1，指令执行；否则不执行。不写 COND 表示无条件执行。

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R1 | R4(C1);

SYN: R3, C0 = R5 | R1

2.2.5.20. 按位异或指令

Syntax SYN: Rd{, Cd} = Rs0 ^ Rs1 {COND}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1						Rs1			Rs0				Rd		

Encode field:

Field	Value
ACTION	^: 10
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 和 Rs1 中的数据进行按位异或操作，结果写到寄存器 Rd

中。若可选项 Cd 存在，结果最低位也同时写入寄存器 Cd 中。

- COND 为可选项，表示该指令的执行条件。当表示执行条件的表达式值为 1，指令执行；否则不执行。不写 COND 表示无条件执行。

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R1 ^ R4(C1);

SYN: R3, C0 = R5 ^ R1

2.2.5.21. 按位非指令

Syntax SYN: Rd{, Cd} = ~Rs0 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	ACTION	1	Cd	COND	0	1	0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1						Rs0	0	0	0	0	0				Rd

Encode field:

Field	Value
ACTION	~: 11
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的数据进行按位非操作, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = ~R1(C1);
SYN: R3, C0 = ~R5

2.2.5.22. 定点等于指令

Syntax SYN: Rd{, Cd} = Rs0 == Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	1	0	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的有符号定点值, 相等时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R1 == R5(!C0);
SYN: R3, C1 = R4 == R5

2.2.5.23. 定点不等于指令

Syntax SYN: Rd{, Cd} = Rs0 != Rs1 {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	1	1	0	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的有符号定点值, 不相等时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R1 != R5(!C0);

SYN: R3, C1 = R4 != R5

2.2.5.24. 定点大于等于指令

Syntax SYN: Rd{, Cd} = Rs0 >= Rs1 {(U)} {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	1	1	U	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的定点值, Rs0 大于等于 Rs1 时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R1 >= R5 (U);
SYN: R3, C1 = R4 >= R5 (!C0)

2.2.5.25. 定点小于指令

Syntax SYN: Rd{, Cd} = Rs0 < Rs1 {(U)} {(COND)}

Note:

COND = C0 or C1 or !C0

Cd = C0 or C1

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1	1	0	U	Cd	COND	0	0	0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
Cd	C0: 10 C1: 11 null: 00
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 比较寄存器 Rs0 和 Rs1 的定点值, Rs0 小于 Rs1 时返回 1, 否则返回 0, 结果写到寄存器 Rd 中. 若可选项 Cd 存在, 结果最低位也同时写入寄存器 Cd 中.

- U 为可选项, 表示无符号数, 否则为有符号数.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R2 = R1 < R5 (U);
SYN: R3, C1 = R4 < R5 (!C0)

2.2.5.26. 寄存器左移指令

Syntax SYN: Rd = Rs0 << Rs1 {(U)} {(CL)} {(T)} {(Us1)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	Us1	CL	U	T	0	COND	1	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Rs1				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs0 中的值左移 Rs1 位, 结果写入 Rd 中

- Us1 为可选项, 表示 Rs1 为无符号数, 否则为有符号数 (只看 Rs1 的低 6 位, 根据 Rs1 的第 5bit 判断其正负). 当 Us1 不存在且 Rs1 为负数时, 该指令对寄存器 Rs0 中的数据右移-Rs1, 注意当 Rs1 低 6 位为 6' h20 时, 右移 32 位; 当 Us1 存在时, 或 Us1 不存在且 Rs1 为正数时, 对寄存器 Rs0 中的数据进行左移, 左移的位数取寄存器 Rs1 的低 5 位.
- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 否则不循环.
- T 为可选项, 当左移时, 该选项存在表示截断操作, 不存在表示进行饱和操作; 当右移时, 该选项存在表示对右移结果进行截断, 不存在表示进行就近舍入.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R4 = R2 << R1 (U);
SYN: R5 = R3 << R2 (CL)

2.2.5.27. 立即数左移指令

Syntax SYN: Rd = Rs0 << uimm5 {(U)}{(CL)} {(T)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	0	CL	U	T	0	COND	1	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	uimm5				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的数据进行左移, 左移的位数为立即数 uimm5, 左移后的数据写到寄存器 Rd 中.

- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 默认不循环. 在非循环移位中, U 存在时, 高位补 0, 否则高位扩展符号位.
- T 为可选项, 表示对结果进行截断, 不存在时进行饱和操作.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R4 = R2 << 12 (U);
SYN: R5 = R3 << 4 (CL)

2.2.5.28. 寄存器右移指令

Syntax SYN: Rd = Rs0 >> Rs1 {(U)} {(CL)} {(T)} {(Us1)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	Us1	CL	U	T	1	COND	1	0	0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	Rs1	Rs0	Rd
---	-----	-----	----

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 Rs0 中的值右移 Rs1 位, 结果写入 Rd 中

- Us1 为可选项, 表示 Rs1 为无符号数, 否则为有符号数 (只看 Rs1 的低 6 位, 根据 Rs1 的第 5bit 判断其正负). 当 Us1 不存在且 Rs1 为负数时, 该指令对寄存器 Rs0 中的数据左移-Rs1, 注意当 Rs1 低 6 位为 6' h20 时, 左移 32 位; 当 Us1 存在时, 或 Us1 不存在且 Rs1 为正数时, 对寄存器 Rs0 中的数据进行右移, 右移的位数取寄存器 Rs1 的低 5 位.
- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 否则不循环.
- T 为可选项, 当左移时, 该选项存在表示截断操作, 不存在表示进行饱和操作; 当右移时, 该选项存在表示对右移结果进行截断, 不存在表示进行就近舍入.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R4 = R2 >> R1 (U);
SYN: R5 = R3 >> R2 (CL)

2.2.5.29. 立即数右移指令

Syntax SYN: Rd = Rs0 >> uimm5 {(U)} {(CL)} {(T)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	0	0	CL	U	T	1	COND	1	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	uimm5				Rs0				Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 对寄存器 Rs0 中的数据进行右移, 右移的位数为立即数 uimm5, 右移后的数据写到寄存器 Rd 中.

- U 为可选项, 表示 Rs0 为无符号数, 否则为有符号数.
- CL 为可选项, 表示循环移位, 默认不循环. 在非循环移位中, U 存在时, 高位补 0, 否则高位扩展符号位.
- T 为可选项, 表示对结果进行截断, 不存在时进行就近舍入.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R4 = R2 >> 12 (U) (T);
SYN: R5 = R3 >> 4 (CL)

2.2.5.30. 立即数赋值指令

Syntax SYN: Rd = Imm16 {(L)} {(E)} {U} {(COND)}

Note:

COND = C0 or C1 or !C0

Imm16 = 16 bits immediate or %Label

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	1	1	Imm16[15:14]	U	E	L	COND	Imm16[13:11]				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm16[10:0]														Rd	

Encode field:

Field	Value
COND	C0: 00
	C1: 01
	!C0: 10
	null: 11

Description 把 16 位立即数的值传送到 32 位寄存器 Rd(其中 Rd 不能为 R0).

- L 为可选项, 表示存入寄存器 Rd 低 16 位, 否则存入寄存器 Rs 高 16 位.
 - U 为可选项, 表示数据为无符号数, 默认为有符号数.
 - E 为可选项, 表示进行符号扩展, 默认不扩展.
 - COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.
- 当 E, L 选项同时存在, 立即数放在 Rs 的低 16bit, 高位 16bit 为符号扩展, 无符号数高位为 0; E 选项存在, L 选项不存在, 立即数放在 Rd 的高 16bit, 低 16bit 清 0.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R4 = 12 (L) (E) (U);
SYN: R5 = 6

2.2.5.31. Select 指令

Syntax SYN: Rd = Select Rs0, Rs1, Rs2 {(T)} {(S|A|I|N|M)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	1	0			Rs2		COND	T	SAINM			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAINM			Rs1				Rs0			Rd					

Encode field:

Field	Value
SAINM	null: 000 S: 001 A: 010 I: 011 N: 100 M: 101
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 根据寄存器 Rs2 的最低 1bit 选择, 若其不为零选择寄存器 Rs1, 否则选择寄存器 Rs0, 结果放入寄存器 Rd 中.

- T 选项为可选项, 存在时表示截断处理, 默认进行饱和处理.
- 选项 S,A,I,N,M 为可选项, 其中:
 - S 选项代表 Sign 操作, 若 Rs0 和 Rs1 符号位的异或值等于选择寄存器的符号位, 则结果为选择寄存器; 若 Rs0 和 Rs1 符号位的异或值不等于选择寄存器的符号位, 则结果为负的选择寄存器的值;
 - A 选项代表选择|Rs1|或者|Rs0|
 - I 选项代表选择|Rs1|或者-|Rs0|
 - N 选项代表选择-Rs1 或者-Rs0
 - M 选项代表选择 Rs1 或者-Rs0
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R15 = Select R2, R3, R5 (S);
SYN: R14 = Select R1, R3, R5 (T)(A)

Smart Logic Confidential For yuyi@ruijie.com.cn

2.2.5.32. 寄存器 Extend 指令

Syntax SYN: Rd = Extend Rs0, Rs1, Rs2 {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1			Rs2		COND	U	0	1		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		Rs1				Rs0			Rd						

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将寄存器 Rs1 的低 5 位的值作为起始位置, 寄存器 Rs2 的低 5 位的值作为终止位置, 从 Rs0 中取出起始位置到终止位置的数据, 并扩展至 32 位后存入 Rd 寄存器中.

- 可选项 U 表示进行无符号扩展, 否则为有符号扩展.
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R8 = Extend R6, R7, R3 (U);
SYN: R14 = Extend R5, R2, R9

2.2.5.33. 立即数 Extend 指令

Syntax SYN: Rd = Extend Rs0, uimm5s1, uimm5s2 {(U)} {(COND)}

Note: COND = C0 or C1 or !C0

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
P	1	1	1	0	1		uimm5s2		COND	U	1	1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1		uimm5s1		Rs0		Rd									

Encode field:

Field	Value
COND	C0: 00 C1: 01 !C0: 10 null: 11

Description 将 uimm5s1 的值作为起始位置, uimm5s2 的值作为终止位置, 从 Rs0 中取出起始位置到终止位置的数据, 并扩展至 32 位后存入 Rd 寄存器中.

- 可选项 U 表示进行无符号扩展, 否则进行有符号扩展
- COND 为可选项, 表示该指令的执行条件. 当表示执行条件的表达式值为 1, 指令执行; 否则不执行. 不写 COND 表示无条件执行.

Execution

Pipeline FT, DP, EX0, EX1

Latency 3

Example SYN: R8 = Extend R6, 2, 17(U);

SYN: R14 = Extend R5, 12, 23

2.2.5.34. NOP 指令

Syntax SYN: NOP

Encoding	31	30	29		28	27	26	25	24	23	22	21	20	19	18	17	16
	P	1	1		0	1	0	0	0	0	0	0	1	1	1	1	1
	15	14	13		12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0		0	0	0	0	0	0	0	0	0	0	0	0	0

Description 无操作

Execution

Pipeline FT, DP, EX0

Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucps2_delay (void)	SYN: NOP

Example SYN: NOP;

2.3. MPU

UCP2.0 MPU 为 21 发射 VLIW 指令集, 每个 VLIW 指令包由最多 21 条指令组成, 分别为 MFetch, SHU0-3, IMA0-3, R0~7, BIU0-3 指令.

当前指令包某 FU 上不发射指令时, 忽略不写.

指令包内指令之间用 || 隔开, 指令包结尾用 ; 标记.

指令包内指令之间没有顺序.

```
// 由 21 条指令组成的 VLIW 指令包, 包内指令无顺序
R2_Inst    ||  R3_Inst    ||  BIU3_Inst  ||  SHU1_Inst  ||
SHU3_Inst  ||  IMA0_Inst  ||  IMA3_Inst  ||  R0_Inst    ||
IMA2_Inst  ||  BIU1_Inst  ||  MFetch_Inst ||  BIU0_Inst  ||
SHU2_Inst  ||  SHU0_Inst  ||  R1_Inst    ||  IMA1_Inst  ||
BIU0_Inst  ||  R4_Inst    ||  R5_Inst    ||  R7_Inst    ||
R6_Inst      ;

// 由 6 条指令组成的 VLIW 指令包
// 无 R1, R2, R4-7, SHU1-3, IMA0, IMA2-3, BIU0, BIU2-3 指令
SHU0_Inst    ||
IMA1_Inst    ||
R3_Inst      ||
BIU1_Inst    ||
R0_Inst      ||
MFetch_Inst  ;
```

MPU VLIW 指令包被压缩编码如下表:

	Higher address <-----> Lower address																					
	R7	R6	R5	R4	R3	R2	R1	R0	IMA3	IMA2	IMA1	IMA0	SHU3	SHU2	SHU1	SHU0	BIU3	BIU2	BIU1	BIU0	MFetch	Header
Size (bit)	16	16	16	16	16	16	16	16	40	40	40	40	32	32	32	32	28	28	28	28	24	24

指令包中所有指令按位连续存放. 不发射的指令不编码不占用指令字空间.

MPU 指令字长为 64 位, 但仅低 36 位有效, 高 28 位忽略, 因此一个完整的 VLIW 指令包会被编码成如下形式, 其中 W[n] 表示第 n 个指令字, 数字格 0-15, 每格为 4-bits.

W[15]																W[14]																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ignore								R7				R6				Ignore								R5				R4				R3
W[13]																W[12]																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Ignore		R3	R2		R1	Ignore		R1	R0	IMA3					
W[11]						W[10]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ignore		IMA3			IMA2		Ignore		IMA2			IMA1			
W[9]						W[8]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ignore		IMA1			IMA0		Ignore		IMA0						
W[7]						W[6]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ignore		SHU3			SHU2		Ignore		SHU2			SHU1			
W[5]						W[4]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ignore		SHU1			SHU0		Ignore		SHU0			BIU3			
W[3]						W[2]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ignore		BIU3		BIU2			Ignore		BIU2		BIU1			BIU0	
W[1]						W[0]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ignore		BIU0			MFetch		Ignore		MFetch		Header				

Header 为 24 位指令包说明段, 编码如下:

23	22	21	20	19	18	17	16	15	14	13	12				
Packet Size				R7&R6		R5	R4		R3	R2		R1	R0		IMA3
11	10	9	8	7	6	5	4	3	2	1	0				
IMA2	IMA1	IMA0	SHU3	SHU2	SHU1	SHU0	BIU3	BIU2	BIU1	BIU0	MFetch				

- 19-0 位表示当前指令包发射的指令, 指令包中包含某 FU 的指令时, 相应的位为 1, 否则为 0.
 - 对于第 19 位: R7 和 R6 指令存在时, 19 位为 1. 仅 R7 或 R6 存在时, 19 位也为 1, 但不存在的指令需编为 NOP 并放入指令包相应位置. 仅当 R7 和 R6 都不存在时, 19 位为 0.
- 23-20 位为 Packet Size, 表示当前指令包(包含 Header)的长度, 单位为 double word, 每个指令包 Header 需 64 位对齐.

UCP2.0 MPU 每条指令一般形式为:

SRC -> DST1 & DST2 (COND)

- SRC 为必需部分, 表示当前指令的操作.
- -> DST1 为可选部分, 表示第一目标域.
- & DST2 为可选部分, 表示第二目标域, 部分指令有第二目标域.
- (COND)为可选部分, 表示当前指令执行条件, 可以为 mode0, !mode0 或 mode1.
 - mode1 表示无条件执行, 当 COND 为 mode1 时, 可以省略.

- 为 mode0 时, 表示 mode0 寄存器中表达式成立时执行, 否则不执行.
- 为!mode0 时, 表示 mode0 寄存器中表达式不成立时执行, 成立不执行.

下表为 SHU/BIU/IMA/MReg 单元的 mode0 寄存器中条件表达式值编码表:

	8	7	6	5	4	3	2	1	0
无条件执行	0	0	1	1	1	1	0	1	1
KIn ==/>>= 0 or KI7/KI11/KI15	0	0							
KIn++ ==/>>= 0 or KI7/KI11/KI15 Note: 每次判断后, n++	1	0							
KIn[0] ==/>>= 0 or KI7/KI11/KI15[0] Note: KIn[0]表示 KIn 寄存器第 0 位	0	1							0: 00
KIn[i++] ==/>>= 0 or KI7/KI11/KI15[i++] Note: i 从 0 开始, 每次条件判断后 i++. 条件判断等价于: KI(n+i/24)[i%24] ==/>>= 0 or KI7/KI11/KI15[i%24]	1	1							KI7: 01 KI11: 10 KI15: 11

2.3.1. 互联关系

2.3.1.1. 功能单元整体互联数据通路

SRC	DEST															
	MReg	SHU0	SHU1	SHU2	SHU3	BIU0	BIU1	BIU2	BIU3	IMA0	IMA1	IMA2	IMA3	MReg Latch	MReg MC	MFetch
R0		S_0_0	S_1_1			B_0_3				I_0_1	I_1_2			L_0		
R1		S_0_1	S_1_0			B_1_3				I_0_2	I_1_1				√	
R2			S_2_0	S_3_1		B_2_3						I_2_1	I_3_2	L_1		
R3			S_2_1	S_3_0		B_3_3						I_2_2	I_3_1		√	
R4	S_0_0	S_1_2			B_0_3				I_0_1	I_1_2			L_0			
R5	S_0_2	S_1_0			B_1_3				I_0_2	I_1_1					MF_0	
R6			S_2_0	S_3_2		B_2_3						I_2_1	I_3_2	L_1		
R7			S_2_2	S_3_0					B_3_3				I_2_2	I_3_1		ConfigMrDepth/ ReadMrw
SHU0	W0/W2/W4/W6	√	S_1_0	S_2_0	S_3_0	B_0_0	B_1_0	B_2_0	B_3_0	I0	I0	I2_0	I3_0			
SHU1	W1/W3/W5/W7	S_0_0	√	S_2_0	S_3_0	B_0_0	B_1_0	B_2_0	B_3_0	I1	I1	I5	I5			
SHU2	W0/W2/W4/W6	S_0_0	S_1_0	√	S_3_0	B_0_0	B_1_0	B_2_0	B_3_0	I2	I2	I3	I3			
SHU3	W1/W3/W5/W7	S_0_0	S_1_0	S_2_0	√	B_0_0	B_1_0	B_2_0	B_3_0	I_0_0	I_1_0	I4	I4			
BIU0	W0/W2/W4/W6	S_0_1	S_1_1	S_2_1	S_3_1	√				I0	I0	I3	I3			MF_0
BIU1	W1/W3/W5/W7	S_0_2	S_1_2	S_2_2	S_3_2		√			I1	I1	I4	I4			MF_0
BIU2	W0/W2/W4/W6	S_0_3	S_1_3	S_2_3	S_3_3			√		I2	I2	I5	I5			MF_0
BIU3	W1/W3/W5/W7	S_0_4	S_1_4	S_2_4	S_3_4				√	I2	I2	I5	I5			MF_0
IMA0	W0/W2/W4/W6	S_0_3	S_1_3	S_2_1	S_3_1	B_0_1	B_1_1	B_2_1	B_3_1	√	I_1_0	I_2_2	I_3_2			
IMA1	W1/W3/W5/W7	S_0_4	S_1_4	S_2_2	S_3_2	B_0_1	B_1_1	B_2_1	B_3_1	I_0_0	√	I_2_2	I_3_2			
IMA2	W0/W2/W4/W6	S_0_1	S_1_1	S_2_3	S_3_3	B_0_2	B_1_2	B_2_2	B_3_2	I_0_2	I_1_2	√	I_3_0			
IMA3	W1/W3/W5/W7	S_0_2	S_1_2	S_2_4	S_3_4	B_0_2	B_1_2	B_2_2	B_3_2	I_0_2	I_1_2	I_2_0	√			
MFetch	W5					√	√	√	√						√	
ReadMrw	W7															

特别的是, IMA 的 FFT 乘法指令, 如果它的目标域是写往自身的寄存器, 则会占用两个互通路: IMA0 的 FFT 会占用 IMA0 的√和I_0_2; IMA1 的 FFT 会占用 IMA1 的√和I_1_2; IMA2 的 FFT 会占用 IMA2 的√和I_2_2; IMA3 的 FFT 会占用 IMA3 的√和I_3_2.

2.3.1.2. MReg 不同模式下的互联关系

模式 1(MReg 深度为 256 时):

SRC	DEST			
	M[0]-M[63]	M[64]-M[127]	M[128]-M[191]	M[192]-M[255]
SHU0	M_0_0		M_0_0	
SHU1	M_0_1		M_0_1	
SHU2		√		√
SHU3		M_0_2		M_0_2
BIU0	√	√	√	√
BIU1	√	√	√	√
BIU2	M_0_1		M_0_1	
BIU3		M_0_2		M_0_2
IMA0	√	√	√	√
IMA1	√	√	√	√
IMA2	√	√	√	√
IMA3	√	√	√	√
MFetch	M_0_0		M_0_0	
ReadMrw		M_0_2	M_0_0	M_0_2

模式 2(MReg 深度为 512 时):

SRC	DEST	
	M[0]-M[255]	M[256]-M[511]
SHU0	M_1_0	
SHU1	M_1_1	
SHU2		√
SHU3		M_1_2
BIU0	√	√
BIU1	√	√
BIU2	M_1_1	
BIU3		M_1_2
IMA0	√	√
IMA1	√	√
IMA2	√	√

IMA3	√	√
MFetch		
ReadMrw	M_1_0	M_1_2

模式 3(MReg 深度为 384, 前 128 共用后 256 独立时):

SRC	DEST		
	M[0]~M[63]	M[64]~M[127]	M[128]~M[255]
SHU0	M_2_0		M_2_0
SHU1	M_2_1		M_2_1
SHU2		√	
SHU3		M_2_2	
BIU0	√	√	√
BIU1	√	√	√
BIU2	M_2_1		M_2_1
BIU3		M_2_2	
IMA0	√	√	√
IMA1	√	√	√
IMA2	√	√	√
IMA3	√	√	√
MFetch	M_2_0		M_2_0
ReadMrw		M_2_2	M_2_2

模式 4(MReg 深度为 384, 前 256 独立后 128 共用时):

SRC	DEST			
	M[0]~M[127]	M[128]~M[255]	M[256]~M[319]	M[320]~M[383]
SHU0	M_3_0		M_3_0	
SHU1	M_3_1		M_3_1	
SHU2		√		√
SHU3		M_3_2		M_3_2
BIU0	√	√	√	√
BIU1	√	√	√	√
BIU2	M_3_1		M_3_1	
BIU3		M_3_2		M_3_2
IMA0	√	√	√	√
IMA1	√	√	√	√
IMA2	√	√	√	√
IMA3	√	√	√	√
MFetch	M_3_0		M_3_0	
ReadMrw		M_3_2		M_3_2

查阅说明:

- √表示直连, 每个直连通路都与其他任何通路不存在写入数据冲突
- 有名称的标号表示连接使用的数据通路(Path), 其中:
 - S_开头的为写往 SHU 的通路, 例如 S_0_5
 - B_开头的为写往 BIU 的通路, 例如 B_1_0
 - I_开头的为写往 IMA 的通路, 例如 I_2_2
 - M_开头的为写往 MFetch 的通路, 例如 M_0
 - W0 - W7 为写往 MReg 的通路
 - I0 - I5 为写往 IMA 的独有的数据通路
- 同一个数据通路在同一时刻不能有多于 1 个的数据传输, 例如, 对于通路 I_3_2, 共有 4 个功能单元(R2/R6/IMA0/IMA1)可能使用此通路写往 IMA3, 如下图所示:

SRC	DEST															
	MReg	SHU0	SHU1	SHU2	SHU3	BIU0	BIU1	BIU2	BIU3	IMA0	IMA1	IMA2	IMA3	MReg Latch	MReg MC	MFetch
R0		S_0_0	S_1_1			B_0_3				I_0_1	I_1_2			√		
R1		S_0_1	S_1_0			B_1_3				I_0_2	I_1_1				√	
R2			S_2_0	S_3_1		B_2_3						I_2_1	I_3_2			M_0
R3			S_2_1	S_3_0		B_3_3						I_2_2	I_3_1			
R4		S_0_0	S_1_2		B_0_3					I_0_1	I_1_2			√		
R5		S_0_2	S_1_0		B_1_3					I_0_2	I_1_1				√	
R6			S_2_0	S_3_2		B_2_3						I_2_1	I_3_2			
R7			S_2_2	S_3_0		B_3_3						I_2_2	I_3_1			
SHU0	W0/W2/W4/W6	S_0_5	S_1_5	S_2_5	S_3_5	B_0_0	B_1_0	B_2_0	B_3_0	I0	I0	I2_0	I3_0			
SHU1	W1/W3/W5/W7	S_0_5	S_1_5	S_2_5	S_3_5	B_0_0	B_1_0	B_2_0	B_3_0	I1	I1	I2	I3			
SHU2	W0/W2/W4/W6	S_0_5	S_1_5	S_2_5	S_3_5	B_0_0	B_1_0	B_2_0	B_3_0	I2	I2	I3	I3			
SHU3	W1/W3/W5/W7	S_0_5	S_1_5	S_2_5	S_3_5	B_0_0	B_1_0	B_2_0	B_3_0	I_0_0	I_1_0	I_2_0	I_3_0			
BIU0	W0/W2/W4/W6	S_0_1	S_1_1	S_2_1	S_3_1	√				I0	I0	I3	I3		M_0	
BIU1	W1/W3/W5/W7	S_0_2	S_1_2	S_2_2	S_3_2	√				I1	I1	I4	I4		M_0	
BIU2	W0/W2/W4/W6	S_0_3	S_1_3	S_2_3	S_3_3	√				I2	I2	I5	I5		M_0	
BIU3	W1/W3/W5/W7	S_0_4	S_1_4	S_2_4	S_3_4	√				I2	I2	I5	I5		M_0	
IMA0	W0/W2/W4/W6	S_0_3	S_1_3	S_2_1	S_3_1	B_0_1	B_1_1	B_2_1	B_3_1	√	I_1_0	I_2_2	I_3_2			
IMA1	W1/W3/W5/W7	S_0_4	S_1_4	S_2_2	S_3_2	B_0_1	B_1_1	B_2_1	B_3_1	I_0_0	√	I_2_2	I_3_2			
IMA2	W0/W2/W4/W6	S_0_1	S_1_1	S_2_3	S_3_3	B_0_2	B_1_2	B_2_2	B_3_2	I_0_2	I_1_2	√	I_3_0			
IMA3	W1/W3/W5/W7	S_0_2	S_1_2	S_2_4	S_3_4	B_0_2	B_1_2	B_2_2	B_3_2	I_0_2	I_1_2	I_2_0	√			
MFetch	W2					√	√	√	√						√	

- 在检查数据通路冲突时, 根据源(SRC)和目标(DEST)功能单元搜索数据传输所使用的数据通路, 检查是否有其他的数据传输在同一时刻也使用了该数据通路, 即可知道是否存在数据冲突. 注意数据传输冲突的检查时刻是在每条指令写入数据的时刻, 而非发射时刻. 例如对于 MReg 使用读口 R0 写往 BIU0 的指令, 假如延时为 2 拍, 则应该在该指令发射 2 拍后检查是否存在数据通路冲突.

2.3.1.3. 各功能单元数据出口

对于每一个功能单元, 它的结果都将通过一个数据输出端口输出到目标域. 但是一个功能单元可能会有不同执行周期的指令, 这些指令混合使用时, 需要注意不能同时占用数据出口. 需要特殊说明的是, IMA 单元带有 L 选项的指令(即需要输出扩展的计算结果的情况), 在第三拍和第四拍都会占用本单元的数据出口.

2.3.2. Wait 指令

MReg/SHU/BIU/IMA 均有 Wait 指令. Wait 指令的功能是将对应指令槽的指令延迟发送. 延迟拍数通过立即数或者 K1 进行设置, 拍数被置为 0 时, 对应指令槽的流水线恢复正常.

Note: Wait 指令的下一条 wait 指令的拍数不能与前一个 wait 拍数相同, 否则下一条 wait 会被当成 NOP 指令处理.

例:

```
SHU0:Wait 18(Mode0);
IMA2: T0 + T3 (W) -> BIU3.T2(Mode0);
SHU0: T1 & T3 -> M[1](Mode0);
SHU0:Wait 28(Mode0);
IMA2: T3 - T1 (B) (U) -> BIU0.T1;
SHU0: T1 | T4 -> M[3](Mode0);
SHU0:Wait 0(Mode0);
SHU0:T1 ^ T2 -> M[4](Mode0);
.....
```

在上述程序中, SHU0 的与指令(&)推迟 18 拍发送, 而 IMA 的指令不受影响. 接着, SHU0 的或指令(|)推迟 28 拍发送, 而 IMA 的指令不受影响. Wait 0 指令清除之前的 SHU0 的 wait 配置, 并且 Wait 0 之后的 SHU0 的 28 条指令将被当做 NOP 指令处理(为了补齐之前 wait28 所延迟的时间), 即从 SHU0 的异或指令(^)开始的 28 条 SHU0 的指令不执行, 28 条之后的 SHU0 的指令正常执行. 若 Wait0 换成 Wait 任意小于 28 的数 N, 则之后会有 (28-N) 条指令被当做 NOP 处理.

若程序改为以下形式, 则 Wait 0 指令清除之前的 SHU0 的 wait 配置, 但其之后的所有 SHU0 指令都能正常执行, 而其他指令槽的指令都相当于延迟了 28 拍执行, 即 SHU0 的异或指令(^)和 IMA2 的或指令(|)再次同时开始执行.

```
SHU0:Wait 18(Mode0);
IMA2: T0 + T3 (W) -> BIU3.T2(Mode0);
SHU0: T1 & T3 -> M[1](Mode0);
SHU0:Wait 28(Mode0);
IMA2: T3 - T1 (B) (U) -> BIU0.T1;
SHU0: T1 | T4 -> M[3](Mode0);
SHU0:Wait 0(Mode0) || Mfetch: MPUWait 0;
SHU0: T1 ^ T2 -> M[4](Mode0) || IMA2:T3 | T1 (B) (U) -> BIU0.T1;
.....
```

2.3.3. Intrinsics 说明

MPU 所有的 intrinsics 均以 `_ucpm2` 开头, 如 `_ucpm2_add` 实现 IMA 上的加法指令.

具体的 intrinsic 定义请参见每条指令下面的 Intrinsics 一节. 下面对一些通用问题进行说明. Intrinsics 中的形参参数命名规范如下:

形参参数名	对应指令中
源操作数 s0, s1, s2, ...	Ts0, Ts1, Ts2, ...
type, type_BSW, type_BSW, ...	B/S/W 等涉及类型的
flags, 如 f_P, f_T, f_Flag, ...	(P), (T), (Flag), ...
mode	(!)mode(0 1), 表示指令的执行条件
fu	功能单元, 包括 R0~7, SHU0~3, BIU0~3, IMA0~3, MFetch
pb	(代表指令包未结束) 或者 ; (代表指令包结束)

用户在传实参时需要遵循如下规范:

形参	实参
type	需带前缀 f_, 如 f_B, f_S, f_W 等
flags	需带前缀 f_, 如 f_P, f_T, f_Flag 等, 若不想指定该 flag, 直接传 0 即可
mode	需带前缀 f_, 如 f_MODE0, f_MODE1, f_NMODE0
fu	需带前缀 fu_, 如 fu_R0, fu_SHU1, fu_BIU2, fu_IMA3, fu_Unspecified 等
pb	需带前缀 pb_, 如 pb_END, pb_CONTINUE

以 IMA 向量加法指令的 intrinsic 为例进行说明:

```
ucp_vec512_t __ucpm2_add(ucp_vec512_t s0, ucp_vec512_t s1, const int type_BSW,  
const int f_T, const int f_U, const int f_CI, const int f_Flag, const int mode, const int fu,  
const int pb)
```

会生成指令

```
IMAx: Ts0 + Ts1 (B|S|W) {(T)} {(U)} {(CI)} {(Flag)} -> Dest
```

具体的, 若调用时传递如下参数 `_ucpm2_add(a, b, f_W, f_T, f_U, 0, 0, f_MODE1, fu_IMA0, pb_END)`, 其中 a, b 为 v16u32 向量型变量, 则会生成 IMA0 上的向量加法指令, 数据粒度为 word, 带 T 和 U 选项, 不带 CI 和 Flag 选项, 按 mode1 模式执行, 即无条件执行.

此外, 提供了三个用于 512bits 向量和 1024bits 向量 pair 互相转换的内建函数:

内建函数	含义
<code>ucp_vec512_t __ucpm2_V_Lo(ucp_vec1024_t s0)</code>	取出 1024bits 向量 pair 中的低 512bits
<code>ucp_vec512_t __ucpm2_V_Hi(ucp_vec1024_t s0)</code>	取出 1024bits 向量 pair 中的高 512bits
<code>ucp_vec1024_t __ucpm2_V_Combine(ucp_vec512_t lo, ucp_vec512_t hi)</code>	将两个 512bits 向量合成一个 1024bits 向量 pair

下表列出了可能用到的 flag:

f_NoFlag	f_DIR	f_BIULDl	f_U	f_IndexMode3	f_Update
f_W	f_CR	f_BIUStL	f_C	f_A	f_G0
f_W1	f_NopFlag	f_ML	f_V	f_HH	f_G1
f_DW	f_APP	f_MH	f_AC	f_HL	f_G2
f_DW1	f_I	f_H	f_Every	f_LL	f_G3
f_S	f_SI	f_BIUStH	f_Any	f_SlipMode0	f_Sub
f_D	f_RPC	f_Q	f_TrsMode0	f_SlipMode1	f_ROUND
f_W0	f_EI	f_QL	f_TrsMode1	f_SlipMode2	f_X1
f_W2	f_DI	f_QH	f_TrsMode2	f_UU	f_X2
f_W3	f_L0	f_APP0	f_TrsMode3	f_SS	f_SortMode0
f_W4	f_L1	f_APP1	f_F	f_CprsMode0	f_SortMode1
f_W5	f_RP0	f_E	f_Odd	f_CprsMode1	f_SortMode2
f_W6	f_RP1	f_WB0	f_Step_H	f_CprsMode2	f_SortMode3
f_W7	f_RP2	f_WB1	f_Alpha	f_CprsMode3	f_Enable
f_B	f_RP3	f_WB	f_Beta	f_ExpdMode0	f_Disable
f_WS	f_RPALL	f_TABLE0	f_UDT1	f_ExpdMode1	f_SM0
f_WF	f_IMAS0	f_TABLE1	f_UDT5	f_ExpdMode2	f_SM1
f_BIUS	f_IMAL0	f_IND	f_T5MD	f_ExpdMode3	f_SM2
f_BIUW	f_T	f_V0	f_Index0	f_Max	f_SM3
f_BIUD	f_P	f_V1	f_Index1	f_Min	f_MODE0
f_DD	f_MASK	f_V2	f_Index2	f_IdEn	f_MODE1
f_UFlag	f_BR	f_V3	f_Index3	f_END	f_NMODE0
f_SHIFT	f_R	f_M	f_EN	f_StartID	f_NMODE1
f_I0	f_BIUAS_R	f_O	f_SSS	f_EndH	
f_I1	f_M0	f_Flag	f_USU	f_ACC	
f_I2	f_M1	f_CI	f_USS	f_All	
f_I3	f_M2	f_O1	f_IndexMode0	f_ID	
f_I4	f_M3	f_O2	f_IndexMode1	f_CROSS	
f_I5	f_L	f_BFlag	f_IndexMode2	f_CLEAR	

2.3.4. MReg

MReg 由可配置深度的一堆 512 位宽的向量寄存器(记为 M[0] - M[511])组成, 深度可在 256, 384, 512 之间选择. MReg 地址以 32 为一组, 称为一个 Bank, 4 个 Bank 构成一个簇. 它对外提供 8 个读口(记为 R0-R7)和 8 个写口(记为 W0-W7). 8 个写口不允许在同一周期内访问任意相同 Bank, 8 个读口根据深度不同可以在同一周期由 4 个或 2 个读口访问同一个 Bank.

MReg 在不同深度下, 读写口的访问限制为:

1. 深度为 256: 8 个读口/写口可访问全部 256 个地址. 8 个写口不允许在同一周期内访问任意相同 Bank. 读口 0 和 4, 1 和 5.....不能在同一个周期访问同一个簇, 即读口 0 和 4 不能同时访问 0~127 内的任意两个地址, 但 0 和 1/2/3 可以同时访问 0~31 内的任意两个地址.
2. 深度为 512: 读/写口 0/1/4/5 可以访问地址 0~255, 读/写口 2/3/6/7 可以访问地址 256~511. 此时地址不同, 写口 0/1/4/5 和 2/3/6/7 将不会产生冲突, 但写口之间不能同时访问 0~63, 64~127.....等相邻 Bank. 读口限制与深度 256 时相同.
3. 深度为 384: 实为上两种方案的结合, 可分为前 128 共用后 256 独立, 前 256 独立后 128 共用两种. 前 128 共用后 256 独立时, 读口/写口 0/1/4/5 可访问地址 0~255, 读口/写口 2/3/6/7 可访问地址 0~127/256~383. 前 256 独立后 128 共用时, 读口/写口 0/1/4/5 可访问地址 0~127/256~383, 读口/写口 2/3/6/7 可访问地址 128~383. 访问限制如上.

由上述读写口约束与各处理单元互联系关系可得: 任意处理单元可将数据写入全部 512 个地址, 但仅可从 256 个地址内取得数据.

此外, 其他单元的数据写往 MReg 还应遵循以下限制:

- MReg 深度为 256 时:
 1. 能写往地址 0~63, 128~191 的单元: IMA0, IMA1, IMA2, IMA3, SHU0/MFetch, SHU1/BIU2, BIU0, BIU1/SYN.
 2. 能写往地址 64~127, 192~255 的单元: IMA0, IMA1, IMA2, IMA3, SHU2, SHU3/BIU3/ ReadPortConf, BIU0, BIU1/SYN.
- MReg 深度为 512 时:
 1. 能写往地址 0~255 的单元: IMA0, IMA1, IMA2, IMA3, SHU0/MFetch, SHU1/BIU2, BIU0, BIU1/SYN.
 2. 能写往地址 256~511 的单元: IMA0, IMA1, IMA2, IMA3, SHU2, SHU3/BIU3/ ReadPortConf, BIU0, BIU1/SYN.
- MReg 深度为 384, 前 128 共用后 256 独立时:
 1. 能写往地址 0~63, 128~255 的单元: IMA0, IMA1, IMA2, IMA3, SHU0/MFetch, SHU1/BIU2, BIU0, BIU1/SYN.

-
2. 能写往地址 64~127, 256~383 的单元: IMA0, IMA1, IMA2, IMA3, SHU2, SHU3/BIU3/ ReadPortConf, BIU0, BIU1/SYN.
- MReg 深度为 384, 前 256 独立后 128 共用时:
 1. 能写往地址 0~127, 256~319 的单元: IMA0, IMA1, IMA2, IMA3, SHU0/ MFetch, SHU1/BIU2, BIU0, BIU1/SYN.
 2. 能写往地址 128~255, 320~383 的单元: IMA0, IMA1, IMA2, IMA3, SHU2, SHU3/BIU3/ ReadPortConf, BIU0, BIU1/SYN.
 - 其中 MFetch 和 SYN 将会按照 256 深度模式访问 MReg, 当深度为 512 时将会同时写入两个地址. SYN 数据不会与任何端口冲突.

MReg 无论作为源还是目标域, 都支持下列多种使用方式:

1. 普通模式:

以 M[t]形式, 指定使用整个 M[t]寄存器. 特别的, 某些指令还支持以 M[t][n]的形式指定 M[t]寄存器的第 n 个 word.

2. ASI 索引模式:

MReg 的每个端口都有一个 64 比特的配置寄存器 (MC), 配置寄存器中各参数的位置及含义如下:

Bits	Name	Description
7:0	lStart	l 窗口起始位置
15:8	lStep	l 窗口步进长度
23:16	lSize	l 窗口大小
31:24	lNum	l 窗口内偏移大小
39:32	sStart	s 窗口起始位置
47:40	sStep	s 窗口步进长度
55:48	sSize	s 窗口大小
63:56	sNum	s 窗口内偏移大小

其中, lSize, sSize 取值范围是 1 至 256(全 0 表示 256), lNum, sNum , lStep, sStep, lStart, sStart 取值范围 0 至 255, 且 Num 和 Step 的值需小于 Size 的大小.

M[{S++},{l++},{A++}]根据配置参数产生寄存器索引, 并在计算出索引后更新相应的参数:

模式	本次使用寄存器	配置寄存器变化
M[l++,A++]	M[lNum + lStart]	$lNum = (lNum + lStep) \% lSize$
M[s++,A++]	M[sNum + sStart]	$sNum = (sNum + sStep) \% sSize$
M[A++]	M[(lNum + sNum) \% sSize + lStart]	$lNum = (lNum + lStep) \% lSize$

M[I++, S++, A++]	M[(INum + SNum) % SSize + IStart]	SNum = (SNum + SStep) % SSize INum = 0
M[I++]	M[INum + IStart]	INum = INum
M[S++]	M[SNum + SStart]	SNum = SNum
M[I++, S++]	M[(INum + SNum) % SSize + IStart]	SNum = SNum INum = INum

3. 离散索引模式:

每 4 个读口(R0/R2/R4/R6, R1/R3/R5/R7) 共享 4 个对应的存放访问地址的缓存(Latch), 每 4 个写口(W0/W2/W4/W6, W1/ W3/W5/W7) 共享 4 个对应的存放访问地址的缓存(Latch). 每个 Latch 的位宽为 256 位. 利用 Latch 计算 MReg 索引:

- M[dis{shift})]:

Latch 以字节为单位存放访问地址. 最高位字节表示 Latch 中有多少个字节有效, 最大值为 31. 最低一个字节表示当前的 MReg 访问地址. Shift 选项存在时, 下一拍 Latch 的有效字节先循环右移 8bit, 再使用最低字节作为访问地址. 当循环次数超过最高字节表示的有效字节数时, Latch 将会被恢复为刚被配置的状态. Shift 选项不存在, 则不进行移位.

2.3.4.1. Read 指令

Syntax	SRC	1. Rx: M[t] 2. Rx: M[{S++},{I++},{A++}] 3. Rx: M[dis {(shift)}] {(LatchID)}
		->
	DST1	1. SHUn.Td 2. BIUx.Td 3. IMAx.Td 4. IMAx.Shiftmodep 5. ACC {(S)} {(T)}{(Sub)}
	COND	mode(0 1)

Encoding

1. SRC.1 -> DST1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND	1												t		

2. SRC.2 -> DST1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND		0010										00	A	S	I

3. SRC.3/4 -> DST1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND		0010										SrcMode	Shift	LatchID	

Encode field:

Field	Value																																																	
COND	参见 2.3 章																																																	
SrcMode	dis: 10																																																	
LatchID	对于读口 R0/R2/R4/R6, 可选的 LatchID 为 Latch0/Latch2/Latch4/Latch6, 编码为: 0: Latch0 1: Latch2 2: Latch4 3: Latch6 对于读口 R1/R3/R5/R7, 可选的 LatchID 为 Latch1/Latch3/Latch5/Latch7, 编码为: 0: Latch1 1: Latch3 2: Latch5 3: Latch7 若不写 LatchID, 则默认与 R 口相同, 即 R0 默认 Latch0, R1 默认 Latch1.																																																	
DST1	<table border="1"> <tr> <td></td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr> <tr> <td>IMA0.Tp IMA2.Tp</td><td>0</td><td>0</td><td>0</td><td></td><td>p</td><td></td></tr> <tr> <td>IMA0.Shiftmodep IMA2.Shiftmodep</td><td>0</td><td>0</td><td>1</td><td></td><td>p</td><td></td></tr> <tr> <td>IMA1.Tp, IMA3.Tp</td><td>0</td><td>1</td><td>0</td><td></td><td>p</td><td></td></tr> <tr> <td>IMA1.shifmodep IMA3.Shiftmodep</td><td>0</td><td>1</td><td>1</td><td></td><td>p</td><td></td></tr> <tr> <td>SHU0.Tp, SHU2.Tp</td><td>1</td><td>0</td><td>0</td><td></td><td>p</td><td></td></tr> <tr> <td>SHU1.Tp, SHU3.Tp</td><td>1</td><td>0</td><td>1</td><td></td><td>p</td><td></td></tr> </table>		5	4	3	2	1	0	IMA0.Tp IMA2.Tp	0	0	0		p		IMA0.Shiftmodep IMA2.Shiftmodep	0	0	1		p		IMA1.Tp, IMA3.Tp	0	1	0		p		IMA1.shifmodep IMA3.Shiftmodep	0	1	1		p		SHU0.Tp, SHU2.Tp	1	0	0		p		SHU1.Tp, SHU3.Tp	1	0	1		p	
	5	4	3	2	1	0																																												
IMA0.Tp IMA2.Tp	0	0	0		p																																													
IMA0.Shiftmodep IMA2.Shiftmodep	0	0	1		p																																													
IMA1.Tp, IMA3.Tp	0	1	0		p																																													
IMA1.shifmodep IMA3.Shiftmodep	0	1	1		p																																													
SHU0.Tp, SHU2.Tp	1	0	0		p																																													
SHU1.Tp, SHU3.Tp	1	0	1		p																																													

	BIU.Tp	1	1	1	0	p	
ACC		1	1	0	S	T	Sub

Description 该指令通过 Rx 端口读取 M 寄存器中的数据, 写入到指定目标单元的寄存器中. 其中:

- R0 和 R4 可写入 IMA0, IMA1, SHU0, SHU1, BIU0
- R1 和 R5 可写入 IMA0, IMA1, SHU0, SHU1, BIU1
- R2 和 R6 可写入 IMA2, IMA3, SHU2, SHU3, BIU2
- R3 和 R7 可写入 IMA2, IMA3, SHU2, SHU3, BIU3

对于 SRC

- M[t]: 表示索引为 t 的 M 寄存器.
- M[{S++},{I++},{A++}]: 表示使用 ASI 索引模式产生 M 寄存器索引.
- M[dis {(shift)}] {(LatchID)}: 表示使用内部 latch 寄存器作为索引, 内部 latch 寄存器视为 64 x i8 的向量, 使用 latch[0]作为 M 寄存器索引. 当 Shift 选项存在时, 表示读取后内部 latch 循环右移 8 位.

对于 DST

- SHUn.Td, BIUx.Td, IMAx.Td: 将读出数据写入指定的寄存器.
- IMAX.Shiftmodep: 将读出数据写入指定 IMA 的 Shiftmode 寄存器
- ACC {(S)} {(T)}{(Sub)}: 若无 Sub 选项, 将读出数据与对应写口数据相加后写回 MReg; 有 Sub 选项表示将读出数据与对应写口数据相减后写回 MReg (即:本条指令读出数据 - 写口写回的数据 -> MReg). S 选项表示为 32 x i16 的加法, 否则为 64 x i8 的加法. T 选项表示结果截断, 否则为饱和.

Note: ACC 仅 R0,R2,R4,R6 支持.

Note:

- 指令执行到第 2 拍才读 MReg.
- 若带 ACC 的指令执行到 Ex2 时, 有一条 SHU0/SHU2 的指令结果恰好要通过与其对应的写口写回 MReg, 则执行相加/相减操作, 并将结果写回. 此处的相应端口是指: R0/R4 与 W0/W4 对应, R2/R6 与 W2/W6 对应. 要求读口和写口访问同一个簇, 即读地址与写地址均在同一个 128 以内. 当写口没有进行写操作时该指令不产生任何效果.

Execution

Pipeline | FT | DC | DP0 | DP1 | DP2 | EX0 | EX1 | EX2 | EX3



Latency

- > BIU: 4
- > IMA: 3
- > SHU: 3
- > ACC: 3

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_readMReg_SIA(const int f_S, const int f_I, const int f_A, const int mode, const int fu, const int pb)	Rx: M[{S++},{I++},{A++}] -> Dest
	ucp_vec512_t __ucpm2_readMReg_DIS(const int f_SHIFT, const int latchn, const int mode, const int fu, const int pb)	Rx: M[dis {(shift)}] {(LatchID)} -> Dest

Note:

参数 latchn 应为：Latch0, Latch1, ..., Latch7.

另外，提供三个写 MReg 的 intrinsic，对应其他单元到 MReg 的 move 指令：

令：

Intrinsic	Generated Instruction
void __ucpm2_writeMReg_SIA(const int f_S, const int f_I, const int f_A, ucp_vec512_t s0, const int mode, const int fu, const int pb)	SHUx/BIUx/IMAx: Ts0 -> M[{S++},{I++},{A++}]
void __ucpm2_writeMReg_DIS(const int f_SHIFT, const int latchn, ucp_vec512_t s0, const int mode, const int fu, const int pb)	SHUx/BIUx/IMAx: Ts0 -> M[dis {(shift)}] {(LatchID)}

Example

R1: M[3] -> BIU1.T0 (Mode1);
R6: M[S++, A++] -> BIU2.T3;
R0: M[dis (shift)] (Latch6) -> BIU0.T0 (Mode0);

2.3.4.2. Pre Config 指令

Syntax	SRC	1. Rx: PreConfig (M[t]{[i]}) 2. Rx: PreConfig (M[{S++},{I++},{A++}]{[i]}) 3. Rx: PreConfig (M[dis{(shift)}]{[i]}) {(LatchID)}
	COND	mode(0 1)

Encoding

1. Src1(vector type): Rx: PreConfig (M[t])

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND	0	1	1		0										t

2. Src1(word type): Rx: PreConfig (M[t][i])

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND	0	1	0		i										t

3. Src2(vector type): PreConfig (M[{S++},{I++},{A++}])

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND			0			1			0		0	A	S	I	

4. Src2(word type): Rx: PreConfig (M[{S++},{I++},{A++}][i])

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND			0			0		i		0	A	S	I		

5. Src3(vector type): Rx: PreConfig (M[dis{(shift)}]) {(LatchID)}

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND			0			1			0		1	Shift	LatchID		

6. Src3(word type): Rx: PreConfig (M[dis{(shift)}][i]) {(LatchID)}

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND			0			0		i		1	Shift	LatchID			

Encode field:

Field	Value
COND	参见 2.3 章
LatchID	对于读口 R0/R2/R4/R6, 可选的 LatchID 为 Latch0/Latch2/Latch4/Latch6, 编码为: 0: Latch0 1: Latch2 2: Latch4 3: Latch6 对于读口 R1/R3/R5/R7, 可选的 LatchID 为 Latch1/Latch3/Latch5/Latch7, 编码为: 0: Latch1 1: Latch3 2: Latch5 3: Latch7 若不写 LatchID, 则默认与 R 口相同, 即 R0 默认 Latch0, R1 默认 Latch1.

Description 该指令指定 Rx 端口缓存 M[t]中的向量, 为后续 MReg config 指令 (2.3.4.3 -2.3.4.7) 使用.

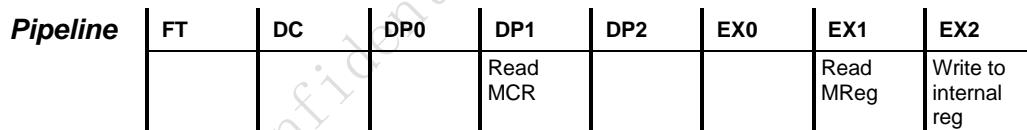
- 当指定 M[t] 时，缓存的向量为 M[t] 的值.
- 当指定 M[t][i] 时，缓存的向量的每一个 word 值为 M[t] 的第 i 个 word 的值.

Note:

- 该指令必须与后续的 MReg config 指令配合使用. 数据可多次使用, 直至出现新的 PreConfig 指令或目标域为 BIU 的读指令 (即 PreConfig 指令和后面配对的 Config 指令中间, 不该有 MReg 写往 BIU 的指令, 否则 PreConfig 的数据无法被后面的 Config 指令正确读取) .
- 指令执行到第 2 拍才读 MReg.

Execution vec512_t saved_config;

```
if ([i] is existed) {
    for (int j = 0; j < 16; j++) {
        saved_config[j] = M[t][i];
    }
} else {
    saved_config = M[t];
}
```



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_preConfigAll(ucp_vec512_t s0, const int mode, const int fu, const int pb)	Rx: PreConfig (M[t])
	void __ucpm2_preConfig(int s0, const int mode, const int fu, const int pb)	Rx: PreConfig (M[t][i])
	void __ucpm2_preConfigAll_SIA(const int f_S, const int f_I, const int f_A, const int mode, const int fu, const int pb)	Rx: PreConfig (M[{S++},{I++},{A++}])
	void __ucpm2_preConfig_SIA(const int f_S, const int f_I, const int f_A, int	Rx: PreConfig (M[{S++},{I++},{A++}][i])

i, const int mode, const int fu, const int pb);	
void __ucpm2_preConfigAll_DIS(const int f_SHIFT, const int latchn, const int mode, const int fu, const int pb)	Rx: PreConfig (M[dis{(shift)}]) {(LatchID)}
void __ucpm2_preConfig_DIS(const int f_SHIFT, const int latchn, int i, const int mode, const int fu, const int pb)	Rx: PreConfig (M[dis{(shift)}][i]) {(LatchID)}

Note :

参数 latchn 应为：Latch0, Latch1, ..., Latch7;

参数 i 应为 0~15 的立即数.

- Example**
- R6: PreConfig (M[7]) (Mode0);
 - R4: PreConfig (M[S++, I++, A++]);
 - R3: PreConfig (M[dis (shift)]) (Latch1);

2.3.4.3. Config MFetch 指令

Syntax	SRC	R5: WriteConf (Mfetch)
		->
	DST1	KI[n n-m]
	COND	mode(0 1)

Encoding 1. R5: WriteConf (Mfetch) -> KI[n-m]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND	0001		1		KI[n-m]							0			

2. R5: WriteConf (Mfetch) -> KI[n]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND	0001		0		n							0			

Encode field:

Field	Value
COND	参见 2.3 章
KI[n-m]	KI[0-3]: 000 KI[4-7]: 001 KI[8-11]: 010 KI[12-15]: 011 KI[0-15]: 100

Description 该指令通过 R5 端口, 将 Pre Config 指令缓存的向量值, 写入 MFetch 指定的 KI 寄存器中.

向量值和 KI 寄存器对应关系如下:

511:480	479:448	447:416	415:384	383:352	351:320	319:288	287:256
KI15	KI14	KI13	KI12	KI11	KI10	KI9	KI8
255:224	223:182	191:160	159:128	127:96	95:64	63:32	31:0
KI7	KI6	KI5	KI4	KI3	KI2	KI1	KI0

Note: 指令执行到第 2 拍才读 Pre Config 指令缓存的向量值.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3
								Read internal reg	Write to MFetch

Latency 8

Intrinsics	Intrinsic	Generated Instruction
-------------------	-----------	-----------------------

void __ucpm2_configMfetch_512(const int mode, const int fu, const int pb)	R5: WriteConf (Mfetch) -> KI[0-15]
void __ucpm2_configMfetch_128(const int ki4regs, const int mode, const int fu, const int pb);	R5: WriteConf (Mfetch) -> KI[0-3]/KI[4-7]/KI[8-11]/KI[12-15]
int __ucpm2_configMfetch_32(const int mode, const int fu, const int pb);	R5: WriteConf (Mfetch) -> KIn

Note:

参数 ki4regs 应为: KI0_3, KI4_7, KI8_11, KI12_15.

Example R5: WriteConf (Mfetch) -> KI[0];

R5: WriteConf (Mfetch) -> KI[8-11] (Mode0);

2.3.4.4. Config Latch 指令

Syntax	Src	Rx: WriteConf (RLatchi / WLatchi) {(H)}
	COND	mode(0 1)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND	0001		i		H	Latch	W/R								0

Encode field:

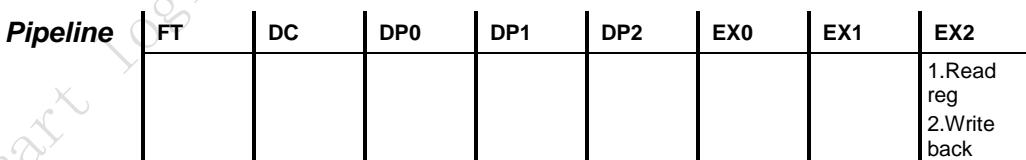
Field	Value
COND	参见 2.3 章
W/R	W: 1 R: 0

Description 该指令通过 Rx 端口, 使用 Pre Config 指令缓存的向量值, 配置 MReg Latch.

- 只有读口 R0/R2/R4/R6 支持该指令, 读口 R0/R4 可配置读/写口 0/1/4/5, 读口 R2/R6 可配置读/写口 2/3/6/7. 不支持读口 R0/R4 或 R2/R6 同时配置读写口.
- 当指定 Latch 时, 使用 Pre Config 缓存的向量值配置 Latch. H 选项存在, 用向量值的高 256bit 配置 Latch; H 不存在, 则用低 256bit 配置 Latch.

Note: 指令执行到第 2 拍才读 Pre Config 指令缓存的向量值.

Execution



- Latency**
- > ReadMReg: 5
 - > SHU Turbo: 3
 - > SHU other instuctions: 4
 - > BIU KG: 4
 - > BIU other instuctions: 1
 - > IMA Div: 1
 - > IMA Mul1/Mul2/FFT & RMax/RMin/RAdd: 2
 - > IMA other instructions: 4

Intrinsics	Intrinsic	Generated Instruction
-------------------	-----------	-----------------------

void __ucpm2_configLatch(const int rwlatch, const int f_H, const int mode, const int fu, const int pb)	Rx: WriteConf (RLatchi / WLatchi) {(H)}
--	---

Note:

参数 rwlatch 应为: RLatch0~7, WLatch0~7;

参数 port 应为: R0~7, W0~7.

Example R0: WriteConf (RLatch1);

2.3.4.5. Config RPort 指令

Syntax	SRC	1. Rx: WriteConf -> MC.Ri (S I SI) 2. Rx: WriteConf -> MC.RAll (S I SI)
	COND	mode(0 1)

Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	COND	0001		i	0	SI	All						0			

Encode field:

Field	Value
COND	参见 2.3 章
SI	I: 00 S: 01 SI:10

Description 该指令通过 Rx 端口, 使用 Pre Config 指令缓存的向量值, 配置 MReg 读端口的配置寄存器 (MC).

- Rx 仅可为 R1 或 R3. 读口 R1 可配置读口 R0/R1/R4/R5, 读口 R3 可配置读口 R2/R3/R6/R7. 当指定 MC.Ri 时, 表示仅配置指定的 Ri. 当指定 MC.RALL 时, 表示同时配置 R0/R1/R4/R5 或 R2/R3/R6/R7.
- 选项 S 表示仅配置 S 参数, I 表示仅配置 I 参数, SI 表示配置 S 和 I 参数.

Note: 指令执行到第 2 拍才读 Pre Config 指令缓存的向量值.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
								1.Read reg 2.Write back

Latency 5

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_configMR(const int mcr, const int si, const int mode, const int fu, const int pb)	Rx: WriteConf -> MC.Ri (S I SI) 或 Rx: WriteConf -> MC.RAll (S I SI)

Note:

参数 mcr 应为: MC_R0, MC_R1, ..., MC_R7, MCR_All;

参数 si 应为: f_S, f_I, f_SI.

Example R3: WriteConf -> MC.R7 (S)(Mode0);

R1: WriteConf -> MC.RAll (SI);

2.3.4.6. Config WPort 指令

Syntax	SRC	1. Rx: WriteConf -> MC.Wi (S I SI) 2. Rx: WriteConf -> MC.WAll (S I SI)
	COND	mode(0 1)

Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	COND	0001		i	1		SI	All					0			

Encode field:

Field	Value
COND	参见 2.3 章
SI	I: 00 S: 01 SI: 10

Description 该指令通过 Rx 端口, 使用 Pre Config 指令缓存的向量值, 配置 MReg 写端口的配置寄存器 (MC).

- Rx 仅可为 R1 或 R3. 读口 R1 可配置写口 W0/W1/W4/W5, 读口 R3 可配置写口 W2/W3/W6/W7. 当指定 MC.Wi 时, 表示仅配置 Wi. 当指定 MC.WALL 时, 表示同时配置 W0/W1/W4/W5 或 W2/W3/W6/W7.
- 选项 S 表示仅配置 S 参数, I 表示仅配置 I 参数, SI 表示配置 S 和 I 参数.

Note: 指令执行到第 2 拍才读 Pre Config 指令缓存的向量值.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
								1.Read reg 2.Write back

- Latency** -> SHU Turbo: 3
-> SHU other instuctions: 4
-> BIU KG: 4
-> BIU other instuctions: 1
-> IMA Div: 1
-> IMA Mul1/Mul2/FFT & RMax/RMin/RAdd: 2
-> IMA other instructions: 4
-> ReadWPort: 5

Intrinsics	Intrinsic	Generated Instruction
-------------------	-----------	-----------------------

<pre>void __ucpm2_configMW(const int mcw, const int si, const int mode, const int fu, const int pb)</pre>	<p>Rx: WriteConf -> MC.Wi (S I SI) 或 Rx: WriteConf -> MC.WAll (S I SI)</p>
---	--

Note:

参数 mcw 应为: MC_W0, MC_W1, ..., MC_W7, MCW_All;

参数 si 应为: f_S, f_I, f_SI.

Example R3: WriteConf -> MC.W5 (SI);
R1: WriteConf -> MC.WAll (I);

2.3.4.7. Config BIU 指令

Syntax	SRC	1. R0/R4: WriteConf -> BIU0.Ts[i] 2. R1/R5: WriteConf -> BIU1.Ts[i] 3. R2/R6: WriteConf -> BIU2.Ts[i] 4. R3/R7: WriteConf -> BIU3.Ts[i]
	COND	mode(0 1)

Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	COND			0		11	1	0		Ts	0		i			

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令通过 Rx 端口, 使用 Pre Config 缓存的向量的第 i 个 word, 配置指定的 BIU 寄存器的第 i 个 word.

- R0 和 R4 可配置 BIU0, R1 和 R5 可配置 BIU1, R2 和 R6 可配置 BIU2, R3 和 R7 可配置 BIU3.

Note: 指令执行到第 2 拍才读 Pre Config 指令缓存的向量值.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
								1.Read reg 2.Write back

Latency

3

Intrinsics	Intrinsic	Generated Instruction
	int __ucpm2_configBIU(const int mode, const int fu, const int pb)	Rx: WriteConf -> BIUn.Ts[i]
	void __ucpm2_configBIU_fix(const int biut_w, const int mode, const int fu, const int pb)	Rx: WriteConf -> BIUn.Ts[i]

Note:

参数 biut_w 应为: BIUxTy_Wz, 其中 $0 \leq x \leq 3, 0 \leq y \leq 3, 0 \leq z \leq 15$.

Example R4: WriteConf -> BIU0.T1[8];

Smart Logic Confidential For yuyi@ruijie.com.cn

2.3.4.8. Read RPort Config 指令

Syntax	SRC	R7: ReadConf (MCR)
		->
	DST1	M[t]
	COND	mode(0 1)

Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	COND	0011			0	0									t	

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令仅可使用 R7 端口, 读所有读端口的配置寄存器, 并写入指定的 M[t] 中.

Execution



Latency

Intrinsics

1	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_readConfigMR(const int mode, const int fu, const int pb)	R7: ReadConf (MCR) -> M[t]

Example R7: ReadConf (MCR) -> M[127] (Mode0);

2.3.4.9. Read WPort Config 指令

Syntax	SRC	R7: ReadConf (MCW)
		->
	DST1	M[t]
	COND	mode(0 1)

Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	COND	0011			0	1							t			

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令仅可使用 R7 端口, 读所有写端口的配置寄存器, 并写入指定的 M[t] 中.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_readConfigMW(const int mode, const int fu, const int pb)	R7: ReadConf (MCW) -> M[t]

Example R7: ReadConf (MCW) -> M[125] (Mode0);

2.3.4.10. Config Depth 指令

Syntax	SRC	R7: WriteConf (Depth, imm2)
	COND	mode(0 1)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND	0001			1	Imm2							0			

Encode field:

Field	Value
COND	参见 2.3 章
Imm2	0: 配置 MReg 深度为 256 1: 配置 MReg 深度为 384, 前 256 独立后 128 共用 2: 配置 MReg 深度为 384, 前 128 共用后 256 独立 3: 配置 MReg 深度为 512

Description 该指令是对 MReg 的深度进行配置.

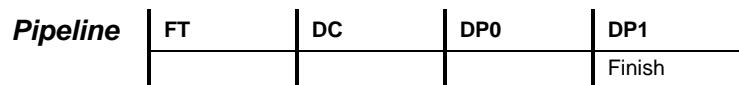
- imm2 为 0, MReg 深度配置为 256, 正常使用 M[0]~M[255]即可.
- imm2 为 1, MReg 深度配置为 384, 前 256 独立后 128 共用. 此时, 读/写口 0/1/4/5 可访问 M[0]~M[127]和 M[256]~M[383], 读/写口 2/3/6/7 可访问 M[128]~M[383]. 用户使用时请注意: 当用读/写口 0/1/4/5 访问 M[256]~M[383]时, 需写成 M[128]~M[255] (即将立即数减去 128); 当用读/写口 2/3/6/7 访问 M[128]~M[383]时, 需写成 M[0]~M[255] (即将立即数减去 128).
- imm2 为 2, MReg 深度配置为 384, 前 128 共用后 256 独立. 此时, 读/写口 0/1/4/5 可访问 M[0]~M[255], 读/写口 2/3/6/7 可访问 M[0]~M[127]和 M[256]~M[383]. 用户使用时请注意: 当用读/写口 2/3/6/7 访问 M[256]~M[383]时, 需写成 M[128]~M[255] (即将立即数减去 128).
- imm2 为 3, MReg 深度配置为 512. 此时, 读/写口 0/1/4/5 可访问 M[0]~M[255], 读/写口 2/3/6/7 可访问 M[256]~M[511]. 用户使用时请注意: 当用读/写口 2/3/6/7 访问 M[256]~M[511]时, 需写成 M[0]~M[255] (即将立即数减去 256).

总结如下表:

模式	端口	实际访问	用户写法
0	所有	M[0]~M[255]	M[0]~M[255]
1	R0/R1/R4/R5	M[0]~M[127]	M[0]~M[127]
		M[256]~M[383]	M[128]~M[255]
2	R0/R1/R4/R5	M[0]~M[255]	M[0]~M[255]

	R2/R3/R6/R7	M[0]~M[127] M[256]~M[383]	M[0]~M[127] M[128]~M[255]
3	R0/R1/R4/R5	M[0]~M[255]	M[0]~M[255]
	R2/R3/R6/R7	M[256]~M[511]	M[0]~M[255]

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_configDepth(const int imm2, const int mode, const int fu, const int pb)	R7: WriteConf (Depth, imm2)

Example R7: WriteConf (Depth, 3);

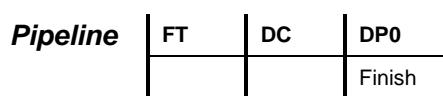
2.3.4.11. Wait 立即数指令

Syntax	SRC	Rx: Wait Imm6
--------	-----	---------------

Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1		0011		01	0	0	0	0						Imm6	

Description 该指令将 Rx 端口的指令延迟 Imm6 拍后执行，最高延迟 62 拍。具体行为参见 2.3.2。

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_mregWaitImm(const int imm6, const int fu, const int pb)	Rx: Wait Imm6

Example R4: Wait 5;

2.3.4.12. Wait KI 指令

Syntax	SRC	Rx: WaitKI
Encoding	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	1 0011 01 0 0 1 0

Description 该指令将使 Rx 端口的指令延迟执行, 最高延迟 62 拍. 具体行为参见 2.3.2.

Wait 的周期数由 Kly[m:n]指定, 其中 y 表示使用哪个 KI, m:n 表示使用该 KI 中的哪些 bit.

Note: 每个 KI 都是 24bit 寄存器, WaitKI 指令使用特定的 6bit 作为 wait 的周期数.

y, m, n 与发射槽的对应关系为:

发射槽	Kly	[m:n]
BIU0	KI8	[5:0]
BIU1	KI8	[11:6]
BIU2	KI8	[17:12]
BIU3	KI8	[23:18]
SHU0	KI9	[5:0]
SHU1	KI9	[11:6]
SHU2	KI9	[17:12]
SHU3	KI9	[23:18]
IMA0	KI10	[5:0]
IMA1	KI10	[11:6]
IMA2	KI10	[17:12]
IMA3	KI10	[23:18]
R0	KI11	[5:0]
R1	KI11	[11:6]
R2	KI11	[17:12]
R3	KI11	[23:18]
R4	KI12	[5:0]
R5	KI12	[11:6]
R6	KI12	[17:12]
R7	KI12	[23:18]

Execution

Pipeline	FT	DC	DP0
			Finish

Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_mregWaitKI(const int fu, const int pb)	Rx: WaitKI

Example R2: WaitKI;

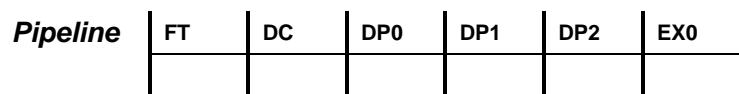
2.3.4.13. NOP 指令

Syntax	SRC	Rx: NOP
--------	-----	---------

Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1		0011		11	0						0				

Description 空指令

Execution



Latency 1

Example R0: NOP;

2.3.4.14. Set Condition 指令

Syntax	Src	Rx: Imm9
		->
	DST1	mode0

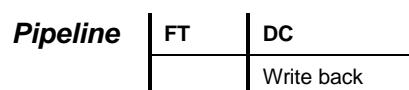
Encoding	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	COND	0011			10											Imm9

Encode field:

Field	Value
COND	参见 2.3 章
Imm9	条件表达式编码, 参见 2.3 章

Description 该指令使用 Imm9 设置条件寄存器 mode0. mode0 寄存器的值对应的条件表达式见 2.3 的条件表达式值编码表.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_mregSetCond(const int imm9, const int fu, const int pb)	Rx: Imm9 -> mode0

Example R6: 172 -> Mode0;

2.3.5. SHU

SHU 公共目标域及编码

No	Assembly	9	8	7	6	5	4	3	2	1	0
1	* -> M[t] {(Wx)}	1	0:W 高 1:W 低						t		
2	* -> M[{S++},{I++},{A++}] {(Wx)}	0	1	1	Wx		0		A++	S++	I++
3	* -> M[dis{(shift)}] {(Wx)}	0	1	1	Wx		10: dis		shift	LatchID	
4	* -> SHUn.Td	0	0	1	0	1	SHUx		Td		
5	* -> BIUx.Td	0	0	1	0	0	BIUx		0	Td	
6	* -> IMA[{0},{1},{2},{3}].Tt	0	0	0	IMA3 Enable	IMA2 Enable	IMA1 Enable	IMA0 Enable		Tt	
15	No destination	0	0	0	0	0	0	0	0	0	0

Note:

1. 对于目标为 Dis 的情况，目标具有可选的 LatchID:

SHU0/SHU2 可选的 LatchID 为: Latch0/Latch2/Latch4/Latch6, 其编码如下:

LatchID	Encoding
Latch0	0
Latch2	1
Latch4	2
Latch6	3

SHU1/SHU3 可选的 LatchID 为: Latch1/Latch3/Latch5/Latch7, 其编码如下:

LatchID	Encoding
Latch1	0
Latch3	1
Latch5	2
Latch7	3

2. Wx 表示使用哪一个写口将结果写入 MReg 中, 但每个 SHU 能够使用的 W 口受到互联关系的限制, 参见 2.3.1.1.

2.3.5.1. Index Short 指令

Syntax	SRC	1. SHUX: Index(Ts0 ,Ts1,Ts2) ($T7 = Ts2 + V(Imm6)$) 2. SHUX: Index(Ts0 ,Ts1,Ts2) ($T7 = Ts2 + V(Ts3)$) 3. SHUX: Index(Ts0 ,Ts1,Ts2) ($T7 = Ts2 + Ts3$) 4. SHUX: Index(Ts0 ,Ts1,Ts2)
		->
	DST	1. IMAX.Td 2. BIUX.Td 3. SHUN.Td 4. M[t {S++},{I++},{A++}] {(Wx)} 5. M[dis{(shift)}] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
	COND	({!}mode(0 1))

Note:

1. Ts0, Ts1 使用 T0~T5, or Ttmp(编码为 0b110), or 0(编码为 0b111)
2. LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

Encoding	1. SRC.1 -> DST1																																																
	<table border="1"> <tr> <td>COND</td><td>0</td><td>0</td><td>Imm6 [5]</td><td>1</td><td>Ts0</td><td>Ts1</td><td>Ts2</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts2</td><td colspan="4">Imm6[4:0]</td><td colspan="18">DST1</td> </tr> </table>	COND	0	0	Imm6 [5]	1	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts2	Imm6[4:0]				DST1																	
COND	0	0	Imm6 [5]	1	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
Ts2	Imm6[4:0]				DST1																																												
	2. SRC.2 -> DST1																																																
	<table border="1"> <tr> <td>COND</td><td>0</td><td>01</td><td>1</td><td>1</td><td>Ts0</td><td>Ts1</td><td>Ts2</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts2</td><td>Ts3</td><td>0</td><td colspan="18">DST1</td> </tr> </table>	COND	0	01	1	1	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts2	Ts3	0	DST1																			
COND	0	01	1	1	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
Ts2	Ts3	0	DST1																																														
	3. SRC.3 -> DST1																																																
	<table border="1"> <tr> <td>COND</td><td>0</td><td>01</td><td>1</td><td>0</td><td>Ts0</td><td>Ts1</td><td>Ts2</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts2</td><td>Ts3</td><td>0</td><td colspan="18">DST1</td> </tr> </table>	COND	0	01	1	0	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts2	Ts3	0	DST1																			
COND	0	01	1	0	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
Ts2	Ts3	0	DST1																																														
	4. SRC.4 -> DST1																																																
	<table border="1"> <tr> <td>COND</td><td>0</td><td>0</td><td>0</td><td>0</td><td>Ts0</td><td>Ts1</td><td>Ts2</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts2</td><td>0</td><td colspan="18">DST1</td> </tr> </table>	COND	0	0	0	0	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts2	0	DST1																				
COND	0	0	0	0	Ts0	Ts1	Ts2	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
Ts2	0	DST1																																															

Encode field:

Field	Value
-------	-------

COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码

Description 该指令将寄存器 Ts0 和 Ts1 中的数据拼接(Ts0 在高, Ts1 在低), 得到 64 个 Short 的中间结果, 然后以 Ts2 中 32 个 Short 的每个 Short 作为索引, 从中间结果中选出 32 个 Short 作为结果, 发往目标域.
Note: 索引值范围是 0~63, 并且要求 Ts2 中每个 Short 的高 Byte 和低 Byte 必须相等.

指令 1~3 会更新 T7 寄存器:

- 对于指令 1: 由 Imm6 复制 64 次得到 512bit, 再将 Ts2 中的数据以 Byte 为粒度与该 512bit 数据分别相加. 计算结果写入 T7.
- 对于指令 2: 由寄存器 Ts3 的最低一个 Byte 复制 64 次得到 512bit, 再将 Ts2 中的数据以 Byte 为粒度与该 512bit 数据分别相加. 计算结果写入 T7.
- 对于指令 3: 将 Ts2 和 Ts3 中的数据以 Byte 为粒度分别相加. 计算结果写入 T7.
- 当计算结果出现溢出时, 做截断处理.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	1.Read regs 2.Write to T7	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)
 - > T7: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_indexShort(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int mode, const int fu, const int pb);	SHUx: Index(Ts0 ,Ts1,Ts2) -> Dest
	ucp_vec512_t __ucpm2_indexShortVImm(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2,	SHUx:Index(Ts0 ,Ts1,Ts2) (T7 = Ts2+V(Imm6)) -> Dest

const int imm6, const int mode, const int fu, const int pb)	
ucp_vec512_t __ucpm2_indexShortVImm_T7(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t _t7, const int imm6, const int mode, const int fu, const int pb)	SHUx:Index(Ts0 ,Ts1,T7) (T7 = T7+V(Imm6)) -> Dest
ucp_vec512_t __ucpm2_indexShortVTt(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int mode, const int fu, const int pb)	SHUx:Index(Ts0 ,Ts1,Ts2) (T7 = Ts2+V(Ts3)) -> Dest
ucp_vec512_t __ucpm2_indexShortVTt_T7(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t _t7, ucp_vec512_t s3, const int mode, const int fu, const int pb)	SHUx:Index(Ts0 ,Ts1,T7) (T7 = T7+V(Ts3)) -> Dest
ucp_vec512_t __ucpm2_indexShortTt(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int mode, const int fu, const int pb)	SHUx:Index(Ts0 ,Ts1,Ts2) (T7 = Ts2+Ts3) -> Dest
ucp_vec512_t __ucpm2_indexShortTt_T7(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t _t7, ucp_vec512_t s3, const int mode, const int fu, const int pb)	SHUx:Index(Ts0 ,Ts1,T7) (T7 = T7+Ts3) -> Dest

Macro	Corresponding Intrinsic
ucpm2_indexShortVImm(s0, s1, s2, _t7, imm6, mode, fu, pb)	__ucpm2_indexShortVImm(s0, s1, s2, imm6, mode, fu, pb); _t7 = __ucpm2_updateT7()
ucpm2_indexShortVImm_T7(s0, s1, _t7, imm6, mode, fu, pb)	__ucpm2_indexShortVImm_T7(s0, s1, _t7, imm6, mode, fu, pb); _t7 = __ucpm2_updateT7()
ucpm2_indexShortVTt(s0, s1, s2, s3, _t7, mode, fu, pb)	__ucpm2_indexShortVTt(s0, s1, s2, s3, mode, fu, pb); _t7 = __ucpm2_updateT7()
ucpm2_indexShortVTt_T7(s0, s1, _t7, s3, mode, fu, pb)	__ucpm2_indexShortVTt_T7(s0, s1, _t7, s3, mode, fu, pb); _t7 = __ucpm2_updateT7()
ucpm2_indexShortTt(s0, s1, s2, s3, _t7, mode, fu, pb)	__ucpm2_indexShortTt(s0, s1, s2, s3, mode, fu, pb);

	<code>_t7 = __ucpm2_updateT7()</code>
<code>ucpm2_indexShortTt_T7(s0, s1, _t7, s3, mode, fu, pb)</code>	<code>__ucpm2_indexShortTt_T7(s0, s1, _t7, s3, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>

Note:

`__ucpm2_updateT7` 是编译器内部用于更新 T7 寄存器的 intrinsic.

Example SHU0: Index(T0 ,T1,T2) (T7 = T2+V(8)) -> BIU0.T1 (Mode0);
SHU0: Index(T3 ,T2,T0) (T7 = T0+T4) -> BIU3.T3;

2.3.5.2. Pre Index byte 指令

Syntax	<table border="1"> <tr> <td>SRC</td><td>SHUx: Index(Ts0,Ts1)</td></tr> <tr> <td>COND</td><td>({!}mode(0 1))</td></tr> </table>	SRC	SHUx: Index(Ts0,Ts1)	COND	({!}mode(0 1))
SRC	SHUx: Index(Ts0,Ts1)				
COND	({!}mode(0 1))				

Note: Ts0 使用 T0~T5, or Ttmp(编码为 0b110), or 0(编码为 0b111)

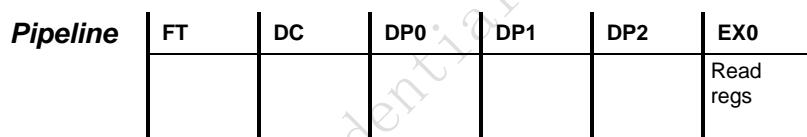
Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	COND	0	11	1	0		Ts0		0		Ts1					
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts1								0							

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令选择寄存器 Ts0 中的数据作为交织的源数据, Ts1 作为交织索引。该指令需要与 Index Byte 指令搭配使用, 两条指令的 Ts1 的值要相等。若该指令与 Index Byte 指令之间插入了其他指令, 则该指令失效。

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_indexPreByte(ucp_vec512_t s0, ucp_vec512_t s1, const int mode, const int fu, const int pb)	SHUx: Index(Ts0,Ts1)

Example SHU0: Index(T0,T4) (Mode0);

2.3.5.3. Index byte 指令

Syntax	SRC	<ol style="list-style-type: none"> 1. SHUX: Index(Ts0,Ts1) ($T7 = Ts1 + V(Imm8)$) 2. SHUX: Index(Ts0,Ts1) ($T7 = Ts1 + V(Ts2)$) 3. SHUX: Index(Ts0,Ts1) ($T7 = Ts1 + Ts2$) 4. SHUX: Index(Ts0,Ts1)
		->
	DST	<ol style="list-style-type: none"> 1. IMAX.Td 2. BIUX.Td 3. SHUn.Td 4. M[t {S++},{I++},{A++}] {(Wx)} 5. M[dis{(shift)}] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
	COND	({!}mode(0 1))

Note:

1. Ts0 使用 T0~T5, or Ttmp(编码为 0b110), or 0(编码为 0b111)
2. LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

Encoding	1. SRC.1 -> DST1																																																															
	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td> </tr> <tr> <td>COND</td><td>0</td><td colspan="2">10</td><td>0</td><td>1</td><td colspan="2">Ts0</td><td colspan="2">Imm8[7:5]</td><td colspan="2">Ts1</td><td></td><td></td><td></td><td></td> </tr> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts1</td><td colspan="5">Imm8[4:0]</td><td colspan="9">DST1</td> </tr> </table>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	COND	0	10		0	1	Ts0		Imm8[7:5]		Ts1						15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts1	Imm8[4:0]					DST1								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																																	
COND	0	10		0	1	Ts0		Imm8[7:5]		Ts1																																																						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																	
Ts1	Imm8[4:0]					DST1																																																										
	2. SRC.2 -> DST1																																																															
	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td> </tr> <tr> <td>COND</td><td>0</td><td colspan="2">10</td><td>1</td><td>1</td><td colspan="2">Ts0</td><td colspan="2">0</td><td colspan="2">Ts1</td><td></td><td></td><td></td><td></td> </tr> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts1</td><td colspan="2">Ts2</td><td colspan="2">0</td><td colspan="9">DST1</td> </tr> </table>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	COND	0	10		1	1	Ts0		0		Ts1						15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts1	Ts2		0		DST1									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																																	
COND	0	10		1	1	Ts0		0		Ts1																																																						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																	
Ts1	Ts2		0		DST1																																																											
	3. SRC.3 -> DST1																																																															
	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td> </tr> <tr> <td>COND</td><td>0</td><td colspan="2">10</td><td>1</td><td>0</td><td colspan="2">Ts0</td><td colspan="2">0</td><td colspan="2">Ts1</td><td></td><td></td><td></td><td></td> </tr> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts1</td><td colspan="2">Ts2</td><td colspan="2">0</td><td colspan="9">DST1</td> </tr> </table>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	COND	0	10		1	0	Ts0		0		Ts1						15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts1	Ts2		0		DST1									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																																	
COND	0	10		1	0	Ts0		0		Ts1																																																						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																	
Ts1	Ts2		0		DST1																																																											
	4. SRC.4 -> DST1																																																															
	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td> </tr> <tr> <td>COND</td><td>0</td><td colspan="2">10</td><td>0</td><td>0</td><td colspan="2">Ts0</td><td colspan="2">0</td><td colspan="2">Ts1</td><td></td><td></td><td></td><td></td> </tr> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>Ts1</td><td colspan="5">0</td><td colspan="9">DST1</td> </tr> </table>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	COND	0	10		0	0	Ts0		0		Ts1						15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ts1	0					DST1								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																																	
COND	0	10		0	0	Ts0		0		Ts1																																																						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																	
Ts1	0					DST1																																																										

Encode field:

Field	Value
-------	-------

COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码

Description 该指令将指定的交织源数据拼接, 得到 128 个 Byte 的中间结果, 然后以 Ts1 中 64 个 Byte 的每个 Byte 作为索引, 从中间结果中选出 64 个 Byte 作为结果, 发往目标域.

交织方式为:

- 若该指令之前有 Pre Index byte 指令, 则将 Pre Index byte 指定的源数据和 Ts0 进行拼接(Ts0 在低位) 得到 128 个 Byte 的中间结果, 然后以 Ts1 中 64 个 Byte 的每个 Byte 作为索引, 从中间结果中选出 64 个 Byte 作为结果.
- 若该指令之前没有 Pre Index byte 指令, 则将两个 Ts0 拼接得到 128 个 Byte 的中间结果, 然后以 Ts1 中 64 个 Byte 的每个 Byte 作为索引, 从中间结果中选出 64 个 Byte 作为结果.

Note: 索引值范围是 0~127.

指令 1~3 会更新 T7 寄存器:

- 对于指令 1: 由 Imm8 复制 64 次得到 512bit, 再将 Ts1 中的数据以 Byte 为粒度与该 512bit 数据分别相加. 计算结果写入 T7.
- 对于指令 2: 由寄存器 Ts2 的最低一个 Byte 复制 64 次得到 512bit, 再将 Ts1 中的数据以 Byte 为粒度与该 512bit 数据分别相加. 计算结果写入 T7.
- 对于指令 3: 将 Ts1 和 Ts2 中的数据以 Byte 为粒度分别相加. 计算结果写入 T7.
- 当计算结果出现溢出时, 做截断处理.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	1.Read regs 2.Write to T7	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)
 - > T7: 1

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_indexByte(ucp_vec512_t s0, ucp_vec512_t s1, const int mode, const int fu, const int pb)</code>	SHUx: Index(Ts0,Ts1) -> Dest
	<code>ucp_vec512_t __ucpm2_indexByteVImm(ucp_vec512_t s0, ucp_vec512_t s1, const int imm8, const int mode, const int fu, const int pb)</code>	SHUx: Index(Ts0,Ts1) (T7 = Ts1+V(Imm8)) -> Dest
	<code>ucp_vec512_t __ucpm2_indexByteVImm_T7(ucp_vec512_t s0, ucp_vec512_t _t7, const int imm8, const int mode, const int fu, const int pb)</code>	SHUx: Index(Ts0,T7) (T7 = T7+V(Imm8)) -> Dest
	<code>ucp_vec512_t __ucpm2_indexByteVTt(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int mode, const int fu, const int pb)</code>	SHUx: Index(Ts0,Ts1) (T7 = Ts1+V(Ts2)) -> Dest
	<code>ucp_vec512_t __ucpm2_indexByteVTt_T7(ucp_vec512_t s0, ucp_vec512_t _t7, ucp_vec512_t s2, const int mode, const int fu, const int pb)</code>	SHUx: Index(Ts0,T7) (T7 = T7+V(Ts2)) -> Dest
	<code>ucp_vec512_t __ucpm2_indexByteTt(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int mode, const int fu, const int pb)</code>	SHUx: Index(Ts0,Ts1) (T7 = Ts1+Ts2) -> Dest
	<code>ucp_vec512_t __ucpm2_indexByteTt_T7(ucp_vec512_t s0, ucp_vec512_t _t7, ucp_vec512_t s2, const int mode, const int fu, const int pb)</code>	SHUx: Index(Ts0,T7) (T7 = T7+Ts2) -> Dest

Macro	Corresponding Intrinsic
<code>ucpm2_indexByteVImm(s0, s1, _t7, imm8, mode, fu, pb)</code>	<code>__ucpm2_indexByteVImm(s0, s1, imm8, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_indexByteVImm_T7(s0, _t7, imm8, mode, fu, pb)</code>	<code>__ucpm2_indexByteVImm_T7(s0, _t7, imm8, mode, fu, pb);</code>

	<code>_t7 = __ucpm2_updateT7()</code>
<code>ucpm2_indexByteVTt(s0, s1, s2, _t7, mode, fu, pb)</code>	<code>__ucpm2_indexByteVTt(s0, s1, s2, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_indexByteVTt_T7(s0, _t7, s2, mode, fu, pb)</code>	<code>__ucpm2_indexByteVTt_T7(s0, _t7, s2, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_indexByteTt(s0, s1, s2, _t7, mode, fu, pb)</code>	<code>__ucpm2_indexByteTt(s0, s1, s2, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_indexByteTt_T7(s0, _t7, s2, mode, fu, pb)</code>	<code>__ucpm2_indexByteTt_T7(s0, _t7, s2, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>

Note:

`__ucpm2_updateT7` 是编译器内部用于更新 T7 寄存器的 intrinsic.

另外，该指令可与 PreIndexByte 指令结合使用：

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_preIndex_indexByte(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int mode, const int fu, const int pb)</code>	<code>SHUx:Index(Ts0,Ts2); SHUx:Index(Ts1,Ts2) -> Dest</code>
	<code>ucp_vec512_t __ucpm2_preIndex_indexByteVI mm(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int imm8, const int mode, const int fu, const int pb)</code>	<code>SHUx:Index(Ts0,Ts2); SHUx:Index(Ts1,Ts2) (T7 = Ts2+V(Imm8)) -> Dest</code>
	<code>ucp_vec512_t __ucpm2_preIndex_indexByteVI mm_T7(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t _t7, const int imm8, const int mode, const int fu, const int pb)</code>	<code>SHUx:Index(Ts0,T7); SHUx:Index(Ts1,T7) (T7 = T7+V(Imm8)) -> Dest</code>
	<code>ucp_vec512_t __ucpm2_preIndex_indexByteVT t(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int mode, const int fu, const int pb)</code>	<code>SHUx:Index(Ts0,Ts2); SHUx:Index(Ts1,Ts2) (T7 = Ts2+V(Ts3)) -> Dest</code>

<code>ucp_vec512_t __ucpm2_prelIndex_indexByteVT t_T7(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t _t7, ucp_vec512_t s3, const int mode, const int fu, const int pb)</code>	<code>SHUx:Index(Ts0,T7); SHUx:Index(Ts1,T7) (T7 = T7+V(Ts3)) -> Dest</code>
<code>ucp_vec512_t __ucpm2_prelIndex_indexByteTt(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int mode, const int fu, const int pb)</code>	<code>SHUx:Index(Ts0,Ts2); SHUx:Index(Ts1,Ts2) (T7 = Ts2+Ts3) -> Dest</code>
<code>ucp_vec512_t __ucpm2_prelIndex_indexByteTt _T7(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t _t7, ucp_vec512_t s3, const int mode, const int fu, const int pb)</code>	<code>SHUx:Index(Ts0,T7); SHUx:Index(Ts1,T7) (T7 = T7+Ts3) -> Dest</code>

Macro	Corresponding Intrinsic
<code>ucpm2_prelIndex_indexByteVIm m(s0, s1, s2, _t7, imm8, mode, fu, pb)</code>	<code>__ucpm2_prelIndex_indexByteVI mm(s0, s1, s2, imm8, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_prelIndex_indexByteVIm m_T7(s0, s1, _t7, imm8, mode, fu, pb)</code>	<code>__ucpm2_prelIndex_indexByteVI mm_T7(s0, s1, _t7, imm8, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_prelIndex_indexByteVTt(s 0, s1, s2, s3, _t7, mode, fu, pb)</code>	<code>__ucpm2_prelIndex_indexByteVT t(s0, s1, s2, s3, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_prelIndex_indexByteVTt_ T7(s0, s1, _t7, s3, mode, fu, pb)</code>	<code>__ucpm2_prelIndex_indexByteVT t_T7(s0, s1, _t7, s3, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_prelIndex_indexByteTt(s0 , s1, s2, s3, _t7, mode, fu, pb)</code>	<code>__ucpm2_prelIndex_indexByteTt(s0, s1, s2, s3, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>
<code>ucpm2_prelIndex_indexByteTt_T 7(s0, s1, _t7, s3, mode, fu, pb)</code>	<code>__ucpm2_prelIndex_indexByteTt _T7(s0, s1, _t7, s3, mode, fu, pb); _t7 = __ucpm2_updateT7()</code>

Note:

`__ucpm2_updateT7` 是编译器内部用于更新 T7 寄存器的 intrinsic.

Example SHU0: Index(T1,T4) ($T7 = T4+V(T3)$) -> BIU3.T2 (Mode0);

2.3.5.4. 向量加法指令: 寄存器 + 立即数

Syntax	SRC	1. SHUx: Ts0 + - V(Imm8) {(T)}
	->	
DST1		1. IMAX.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++},{I++},{A++}] {(Wx)} 5. M[dis{shift})] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
COND		({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01	0	0	0	0	0	T	+ -		Imm8[7:5]		Ts0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0		Imm8[4:0]										DST1			

Encode field:

Field	Value
COND	参见 2.3 章
+ -	+: 0 -: 1
DST1	参见 2.3.5 章, SHU 公共目标域及编码

Description 该指令将寄存器 Ts0 中的数据与 Imm8 进行加减运算, 结果发往目标域.

- +|- 表示进行加法或减法操作.
- T 选项存在, 表示对计算结果进行截断. T 选项不存在, 对计算结果取饱和.
- 首先由 Imm8 复制 64 次得到 512bit, 再将 Ts0 中的数据以 Byte 为粒度与该 512bit 数据分别相加或相减.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_addVImm(ucp_vec512_t s0, int imm8, const int f_T, const int mode, const int fu, const int pb)	SHUx: Ts0 + V(Imm8) {(T)} -> Dest
	ucp_vec512_t __ucpm2_subVImm(ucp_vec512_t s0, int imm8, const int f_T, const int mode, const int fu, const int pb)	SHUx: Ts0 - V(Imm8) {(T)} -> Dest

Example SHU0: T2 - V(7) (T) -> BIU2.T1;

2.3.5.5. 向量加法指令: 寄存器 + 寄存器

Syntax	SRC	1. SHUx: Ts0 + - Ts1 {(T)} {(S)}
		2. SHUx: Ts0 + - V(Ts1) {(T)} {(S)}
	->	
	DST1	1. IMAX.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++}, {I++}, {A++}] {(Wx)} 5. M[dis{(shift)}] {(Wx)} (LatchID) 6. IMA[{0}, {1}, {2}, {3}].Td
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

Encoding

1. SRC.1 -> DST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01	0	1	0	S	T	+ -	0							Ts0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	Ts1	0													DST1

2. SRC.2 -> DST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01	0	1	1	S	T	+ -	0							Ts0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	Ts1	0													DST1

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
+ -	+: 0 -: 1

Description 该指令将寄存器 Ts0 中的数据与 Ts1 中的数据进行加减运算, 结果发往目标域.

- +|- 表示进行加法或减法操作.
- T 选项存在, 表示对计算结果进行截断. T 选项不存在, 对计算结果取饱和.
- S 选项存在, 表示数据粒度为 Short. S 选项不存在, 数据粒度为 Byte.

计算方式如下:

-
- 对于指令 1: Ts0 和 Ts1 中对应粒度的数据分别相加或相减.
 - 对于指令 2: 首先由 Ts1 中最低 1byte 复制得到 512bit, 再将 Ts0 中的数据按照粒度与该 512bit 数据分别相加或相减.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_addVTt(ucp_vec512_t s0, ucp_vec512_t s1, const int f_T, const int f_S, const int mode, const int fu, const int pb)	SHUx: Ts0 + V(Ts1) {(T)} {(S)} -> Dest
	ucp_vec512_t __ucpm2_subVTt(ucp_vec512_t s0, ucp_vec512_t s1, const int f_T, const int f_S, const int mode, const int fu, const int pb)	SHUx: Ts0 - V(Ts1) {(T)} {(S)} -> Dest
	ucp_vec512_t __ucpm2_addTt(ucp_vec512_t s0, ucp_vec512_t s1, const int f_T, const int f_S, const int mode, const int fu, const int pb)	SHUx: Ts0 + Ts1 {(T)} {(S)} -> Dest
	ucp_vec512_t __ucpm2_subTt(ucp_vec512_t s0, ucp_vec512_t s1, const int f_T, const int f_S, const int mode, const int fu, const int pb)	SHUx: Ts0 - Ts1 {(T)} {(S)} -> Dest

Example SHU0: T1 + T2 (S) -> IMA0.T0;

2.3.5.6. 向量位移指令

Syntax	SRC	1. SHUx: Ts0 << Ts1 2. SHUx: Ts0 << Imm3 3. SHUx: Ts0 >> Ts1 {(U)} 4. SHUx: Ts0 >> Imm3 {(U)}
		->
	DST1	1. IMAx.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++}, {I++}, {A++}] {(Wx)} 5. M[dis{(shift)}] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

Encoding

1. SRC.1 -> DST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01	01		0	1		111			0		Ts0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	Ts1		1	0								DST1			

2. SRC.2 -> DST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01	01		0	1		111			0		Ts0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	Imm3		0	0								DST1			

3. SRC.3 -> DST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01	01		0	~U		110			0		Ts0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	Ts1		1	0								DST1			

4. SRC.4 -> DST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01	01		0	~U		110			0		Ts0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	Imm3		0	0								DST1			

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 中的数据以 Byte 为单位进行移位, 结果发往目标域.

- 对于指令 1 和 3, 移位数是寄存器 Ts1 对应 Byte 的低 3 位.
- 对于指令 2 和 4, 移位数是 Imm3.
- 对于右移指令, 若 U 选项存在, 移位后高位补 0; U 选项不存在, 则移位后高位补符号位.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Example SHU0: T4 >> T1(U) -> BIU1.T2;

2.3.5.7. 向量逻辑运算指令

Syntax	SRC	1. SHUx: Ts0 & Ts1 2. SHUx: Ts0 Ts1 3. SHUx: Ts0 ^ Ts1 4. SHUx: ~Ts0
		->
	DST1	1. IMAx.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++}, {I++}, {A++}] {(Wx)} 5. M[dis{(shift)}] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	COND	01		01		1			0						Ts0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0		Ts1/0		OPC										DST1	

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
OPC	00: 01: & 10: ^ 11: ~

Description 该指令进行逻辑运算.

- 对于指令 1~3, 将寄存器 Ts0 和 Ts1 中的值按位进行相应的逻辑运算, 结果发往目标域.
- 对于指令 4, 将寄存器 Ts0 中的值按位取反, 结果发往目标域.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs	Write to MReg, IMA, SHU	Write to BIU

Latency -> BIU: 3

-> MReg: 1

-> IMA: 2

-
- > other SHU: 2
 - > self SHU: 1(bypass)

Example SHU0: T0 | T3 -> SHU0.T4;

Smart Logic Confidential For yuyi@ruijie.com.cn

2.3.5.8. PN 序列生成指令

Syntax	SRC	SHUx: PN (Ts0) {((Update))}
	->	
DST1	1. IMAx.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++},{I++},{A++}] {(Wx)} 5. M[dis{shift})] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td	
COND	({!}mode(0 1))	

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	COND	10	00	0	0	0	0	0	0	0	0	Update	0	0	Ts0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
															DST1	

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码

Description 该指令将寄存器 Ts0 中的 bit48~111 作为数据源(记为 S0~S63), 对其进行一系列操作后生成 112bit 数据(记为 PN0~PN111), 该数据放入 512bit 最终结果的低 112 位, 高位补 0. 最终结果发往目标域.

- Update 选项存在, 表示该指令也会将最终结果发往寄存器 T7.

该指令支持以下两种操作:

- X1:

$$PN0 \sim PN27: PN_x = S_{x+33} \oplus S_{x+36}$$

$$PN28 \sim PN55: PN_x = S_{x+2} \oplus S_{x+8}$$

$$PN56 \sim PN59: PN_x = S_{x+2} \oplus S_{x-54} \oplus S_{x-48}$$

$$PN60 \sim PN111: PN_x = S_{x-60} \oplus S_{x-48}$$

- X2:

$$PN0 \sim PN55: PN_x = S_{x+2} \oplus S_{x+4} \oplus S_{x+6} \oplus S_{x+8}$$

$$PN56 \sim PN83: PN_x = S_{x-29} \oplus S_{x-28} \oplus S_{x-21} \oplus S_{x-20}$$

$$PN84 \sim PN111: PN_x = S_{x-60} \oplus S_{x-56} \oplus S_{x-52} \oplus S_{x-48}$$

Note: 仅 SHU1 和 SHU2 支持该指令. 其中, SHU1 仅支持 X1 操作, SHU2 仅支持 X2 操作.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_pn(ucp_vec512_t s0, const int f_Update, const int mode, const int fu, const int pb)	SHUx: PN (Ts0) {(Update)} -> Dest

Example SHU0: PN (T3) (Update) -> BIU1.T0;

2.3.5.9. CRC 校验指令

Syntax	SHUx: CRC (Ts0)
	->
DST1	7. IMAX.Td 8. BIUx.Td 9. SHUn.Td 10. M[t {S++},{I++},{A++}] {(Wx)} 11. M[dis{shift})] {(Wx)} (LatchID) 12. IMA[{0},{1},{2},{3}].Td
COND	({!}mode(0 1))

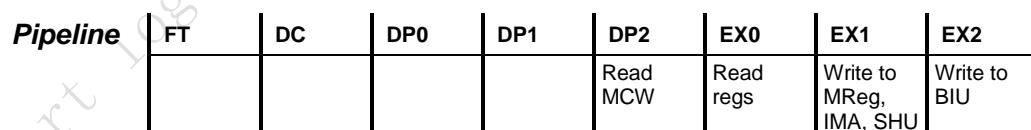
Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	COND	10	00	1	0	Ts0	0	0				0	0		Ts0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
															DST1	

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码

Description 该指令将寄存器 Ts0 中的数据分成 16 个 Word, 将每个 Word 中的 4 个 Byte 互相按位异或, 拼接生成 128bit, 放在 512bit 最终结果的低位, 高位补 0. 最终结果发往目标域.

Execution



Latency

- > BIU: 3
- > MReg: 1
- > IMA: 2
- > other SHU: 2
- > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_crc(ucp_vec512_t s0,	SHUx: CRC (Ts0) -> Dest

const int mode, const int fu, const int pb)	
--	--

Example SHU0: CRC (T4) -> IMA0.T2;

Smart Logic Confidential For yuyi@ruijie.com.cn

2.3.5.10. Extract Bit 指令

Syntax	SRC	SHUx: ExtractBit (Ts0, numbits)
		->
	DST1	13. IMAX.Td 14. BIUX.Td 15. SHUn.Td 16. M[t {S++},{I++},{A++}] {(Wx)} 17. M[dis{(shift)}] {(Wx)} (LatchID) 18. IMA[{0},{1},{2},{3}].Td
	COND	({!}mode(0 1))

Note:

1. numbits = 1 or 2 or 4 or all
2. LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	COND	01		10		1	0		Ts0		0		0		0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			0		numbits							DST1				

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
numbits	00: 1 01: 2 10: 4 11: all

Description 该指令将寄存器 Ts0 中的指定比特抽取出来, 放入目标域.

- 当 numbits=1 时, 将 Ts0 中 64 个 byte 的每个 byte 的最低 1bit 抽取出来, 组成 64bit 数据, 放入目标寄存器的低 64bit 位置 (其他位置为 0).
- 当 numbits=2 时, 将 Ts0 中 64 个 byte 的每个 byte 的最低 2bit 抽取出来, 组成 128bit 数据, 放入目标寄存器的低 128bit 位置 (其他位置为 0).
- 当 numbits=4 时, 将 Ts0 中 64 个 byte 的每个 byte 的最低 4bit 抽取出来, 组成 256bit, 放入目标寄存器的低 256bit 位置 (其他位置为 0).
- 当 numbits=all 时, 将结果分为 8 个 64bit 数据, 则对于第 i 个 64bit 数, 它是从 Ts0 的每个 byte 中抽取第 i 比特拼接得到的. 最终将 512bit 数据全部放入目标域寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_extractBit_1(ucp_vec512_t s0, const int mode, const int fu, const int pb)	SHUx: ExtractBit (Ts0, 1) -> Dest
	ucp_vec512_t __ucpm2_extractBit_2(ucp_vec512_t s0, const int mode, const int fu, const int pb)	SHUx: ExtractBit (Ts0, 2) -> Dest
	ucp_vec512_t __ucpm2_extractBit_4(ucp_vec512_t s0, const int mode, const int fu, const int pb)	SHUx: ExtractBit (Ts0, 4) -> Dest
	ucp_vec512_t __ucpm2_extractBit_All(ucp_vec512_t s0, const int mode, const int fu, const int pb)	SHUx: ExtractBit (Ts0, All) -> Dest

Example SHU0: ExtractBit (T1, all) -> IMA0.T4 (Mode0);

2.3.5.11. Insert Bit 指令

Syntax	SHUx: InsertBit (Ts0, numbits) {({U})}
	->
DST1	1. IMAX.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++},{I++},{A++}] {({Wx})} 5. M[dis{shift})] {({Wx})} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
COND	({!}mode(0 1))

Note:

1. numbits = 1 or 2 or 4 or all
2. LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01		10		0	~U		Ts0		0		0		0	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		0		numbits						DST1					

Encode field:

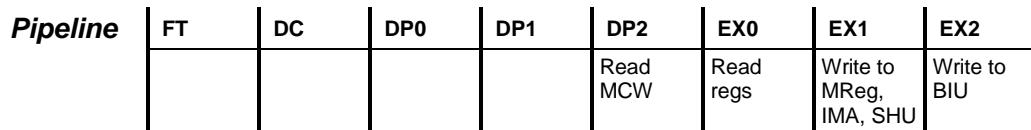
Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
numbits	00: 1 01: 2 10: 4 11: all

Description 该指令对寄存器 Ts0 中的指定比特进行扩展, 结果放入目标域.

- 当 numbits=1 时, 将结果划分为 64 个 byte, 则取 Ts0 中最低 64bit 的第 i 比特放入对应的第 i 个 byte 的最低位(其他位置为 0).
- 当 numbits=2 时, 将结果划分为 64 个 byte, 把 Ts0 中最低 128bit 的每 2bit 分为一组, 取第 i 组放入结果的第 i 个 byte 的最低 2 位(其他位置为 0).
- 当 numbits=4 时, 将结果划分为 64 个 byte, 把 Ts0 中最低 256bit 的每 4bit 分为一组, 取第 i 组放入结果的第 i 个 byte 的最低 4 位(其他位置为 0).
- 当 numbits=all 时, 将 Ts0 的数据分为 8 个 64bit, 则对于第 i 个 64bit 数, 取它的第 k 比特放入结果的第 k 个 byte 的第 i 比特处. 最终将 512bit 数据全部放入目标域寄存器.

-
- U 选项存在, 表示无符号数, 结果的高位扩展 0; 否则为有符号数, 对取出数据的高位进行扩展.

Execution



- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_insertBit_1(ucp_vec512_t s0, const int f_U, const int mode, const int fu, const int pb)	SHUX: InsertBit (Ts0, 1) {(U)} -> Dest
	ucp_vec512_t __ucpm2_insertBit_2(ucp_vec512_t s0, const int f_U, const int mode, const int fu, const int pb)	SHUX: InsertBit (Ts0, 2) {(U)} -> Dest
	ucp_vec512_t __ucpm2_insertBit_4(ucp_vec512_t s0, const int f_U, const int mode, const int fu, const int pb)	SHUX: InsertBit (Ts0, 4) {(U)} -> Dest
	ucp_vec512_t __ucpm2_insertBit_All(ucp_vec512_t s0, const int f_U, const int mode, const int fu, const int pb)	SHUX: InsertBit (Ts0, All) {(U)} -> Dest

Example SHU0: InsertBit (T4, 2) (U) -> M[24](W0);

2.3.5.12. Step Extract 指令

Syntax	SHUx: StepExt(Ts0,Ts1, step) {((Odd)}
	->
DST1	1. IMAx.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++},{I++},{A++}] {((Wx)} 5. M[dis{(shift)}] {((Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
COND	({!}mode(0 1))

Note:

1. step = 1 or 2 or 4 or 8
2. Ts0/Ts1 使用 T0~T5, or Ttmp(编码为 0b110), or 0(编码为 0b111)
3. LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01		11		0	Odd		Ts0		Ts1		0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		0		step						DST1					

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
step	00: 1 01: 2 10: 4 11: 8

Description 该指令将寄存器 Ts0 和 Ts1 拼接得到 1024bit 临时结果, 根据 step 对临时结果进行抽取, 最终结果发往目标域.

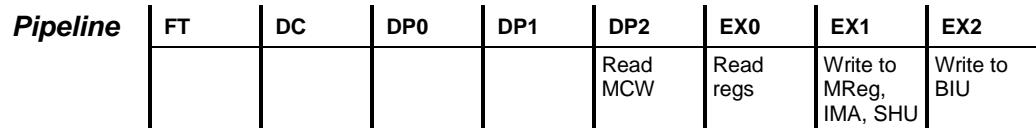
- step 表示抽取间隔.
- 将拼接的数据按照抽取间隔分组, Odd 选项存在, 表示从奇数组开始抽取. Odd 选项不存在, 从偶数组开始抽取.

抽取方式如下:

设抽取间隔为 n. 当 Odd 选项不存在时, 先从 bit0 开始抽取 n 比特, 然后跳过 n 比特, 再取 n 比特, 以此类推. 即最终结果的 bit 构成从低到高为{0,(n-1), 2n,(2n-1), 3n,(3n-1),}. 当 Odd 选项存在时, 先从 bit n 开始取 n 比特, 跳过 n 比特, 再取 n 比特, 以此类推. 即最

终结果的 bit 构成从低到高为{n,.....(2n-1), 3n,.....(4n-1), 5n,(6n-1),}.

Execution



- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_stepExt_1(ucp_vec512_t s0, ucp_vec512_t s1, const int f_Odd, const int mode, const int fu, const int pb)	SHUx: StepExt(Ts0,Ts1, 1) {(Odd)} -> Dest
	ucp_vec512_t __ucpm2_stepExt_2(ucp_vec512_t s0, ucp_vec512_t s1, const int f_Odd, const int mode, const int fu, const int pb)	SHUx: StepExt(Ts0,Ts1, 2) {(Odd)} -> Dest
	ucp_vec512_t __ucpm2_stepExt_4(ucp_vec512_t s0, ucp_vec512_t s1, const int f_Odd, const int mode, const int fu, const int pb)	SHUx: StepExt(Ts0,Ts1, 4) {(Odd)} -> Dest
	ucp_vec512_t __ucpm2_stepExt_8(ucp_vec512_t s0, ucp_vec512_t s1, const int f_Odd, const int mode, const int fu, const int pb)	SHUx: StepExt(Ts0,Ts1, 8) {(Odd)} -> Dest

Example SHU0: StepExt(T2,T1, 2) (Odd) -> SHU0.T3;

2.3.5.13. Reverse Step Extract 指令

Syntax	SHUx: ReStepExt(Ts0,Ts1, step) {(H)}
	->
DST1	1. IMAx.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++},{I++},{A++}] {(Wx)} 5. M[dis{(shift)}] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Tt
COND	({!}mode(0 1))

Note:

1. step = 1 or 2 or 4 or 8
2. Ts0/Ts1 使用 T0~T5, or Ttmp(编码为 0b110), or 0(编码为 0b111)
3. LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	01		11		1	H		Ts0		Ts1		0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		0		step						DST1					

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
step	00: 1 01: 2 10: 4 11: 8

Description 该指令将寄存器 Ts0 和 Ts1 拼接得到 1024bit 临时结果, 根据 step 对临时结果进行抽取, 最终结果发往目标域.

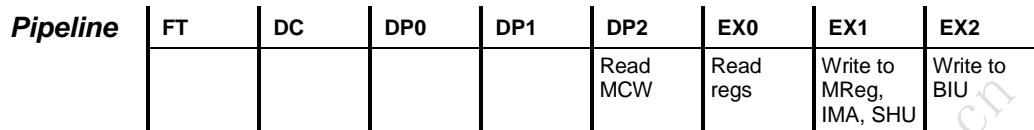
- step 表示抽取间隔.
- H 选项存在, 表示从高一半开始抽取. H 选项不存在, 从低一半开始抽取.

抽取方式如下:

设抽取间隔为 n. 当 H 选项不存在时, 先从 bit0 开始取 n 比特, 然后从 bit512 开始取 n 比特, 拼接, 再从 bit n 开始取 n 比特, 从 bit(512+n)开始取 n 比特, 以此类推. 即最终结果的 bit 构成从低到高为{0,(n-1), 512,(511+n), n,(2n-1), (512+n),(511+2n),}. 当 H 选项存在时, 先从 bit256 开始取 n 比特, 然后从 bit768 开始取 n 比特, 拼

接, 再从 bit (256+n)开始取 n 比特, 从 bit(768+n)开始取 n 比特, 以此类推. 即最终结果的 bit 构成从低到高为{256,(255+n), 768,(767+n), (256+n),(255+2n), (768+n),(767+2n),}.

Execution



- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_reStepExt_1(ucp_vec512_t s0, ucp_vec512_t s1, const int f_H, const int mode, const int fu, const int pb)	SHUX: ReStepExt(Ts0,Ts1, 1) {(H)} -> Dest
	ucp_vec512_t __ucpm2_reStepExt_2(ucp_vec512_t s0, ucp_vec512_t s1, const int f_H, const int mode, const int fu, const int pb)	SHUX: ReStepExt(Ts0,Ts1, 2) {(H)} -> Dest
	ucp_vec512_t __ucpm2_reStepExt_4(ucp_vec512_t s0, ucp_vec512_t s1, const int f_H, const int mode, const int fu, const int pb)	SHUX: ReStepExt(Ts0,Ts1, 4) {(H)} -> Dest
	ucp_vec512_t __ucpm2_reStepExt_8(ucp_vec512_t s0, ucp_vec512_t s1, const int f_H, const int mode, const int fu, const int pb)	SHUX: ReStepExt(Ts0,Ts1, 8) {(H)} -> Dest

Example SHU0: ReStepExt(T5,T4, 4) (H) -> SHU0.T6;

2.3.5.14. Imm 指令

Syntax	Src & Dst COND	1. SHUx: Imm(Imm16) {(H)} -> SHUx.Td[Imm4] 2. SHUx: VImm(Imm16) -> SHUx.Td ({!}mode(0 1))
---------------	-------------------	---

Note: step = 0 or 1 or 2 or 4

Encoding

1. SHUx: Imm(Imm16) {(H)} -> SHUx.Ts[Imm4]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND		11		00		0	0	Imm4[3:2]	H	Imm4[1:0]	Imm16[15]		Td		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. SHUx: VImm(Imm16) -> SHUx.Td

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND		11		00		0	1	01	0	00	Imm16[15]		Td		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

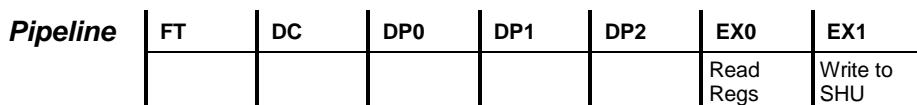
Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令根据 Imm 对目标寄存器的指定位置进行赋值.

- 对于指令 1: 将寄存器视为 16 个 32bit 组成的数据, 然后把 Imm16 写入到目标寄存器 Td 的第 Imm4 个 32bit 处. 当 H 选项存在时, 写入 32bit 的高 16bit. H 选项不存在, 写入 32bit 的低 16bit.
- 对于指令 2: 将 Imm16 复制 32 份, 拼接得到 512bit 结果, 最后把结果写入目标寄存器 Td.

Execution



Latency -> self SHU: 1(bypass)

Example SHU0: Imm(0x1be6) (H) -> SHU0.T4[14];

2.3.5.15. Turbo 指令

Syntax	SRC	SHUx.Turbo(T7, T1, T5) ('Alpha Beta') ('B S') ('Index0 Index1 Index2 Index3') (UpdateT5) {(T5Mode)}
		->
	DST1	1. IMAx.Td 2. BIUx.Td 3. SHUx.T7 4. SHUUn.Td 5. M[t {S++},{I++},{A++}] {(Wx)} 6. M[dis((shift))] {(Wx)} (LatchID) 7. IMA[{0},{1},{2},{3}].Td
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.5: SHU 公共目标域及编码

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COND	10	10	0	A/B	S/B	0	0	UpdateT5	T5Mode	0	11				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	101	Index													DST1

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
Index	00: Index0 01: Index1 10: Index2 11: Index3
A/B	Alpha: 1 Beta: 0
S/B	1:S 0:B

Description 该指令先按指定方式更新寄存器 T5, 然后完成 Turbo 运算, 根据选项 Alpha|Beta 选择完成 Alpha 或 Beta 操作, 将运算结果更新到目标域和寄存器 T7. 只有 SHU1/2 支持该指令, 当目标域为自身 SHU 时, 只能发往 T7, 当非自身 SHU 时, 可发往任意 T. 选项 S 表示操作的数据粒度是 Short, 选项 B 表示操作的数据粒度是 Byte.

对于 SRC

- 寄存器 T7 存放 Alpha 或 Beta 数据, 寄存器 T1/T5 中存放 gamma 的值.

- Alpha/Beta 数据的输入输出摆放规则相同, 如下图, 其中 short 类型表示一组 8×16 bit 数(记为 VA)进行计算, byte 类型表示寄存器中存在两组 8×8 bit 数(记为 VA, VB), 同时进行计算并输出结果..

Alpha 数据的摆放规则:

short 类型

VA[7]	VA[6]	VA[5]	VA[4]	VA[3]	VA[2]	VA[1]	VA[0]
-------	-------	-------	-------	-------	-------	-------	-------

byte 类型

VB[7]	VA[7]	VB[6]	VA[6]	VB[5]	VA[5]	VB[4]	VA[4]	VB[3]	VA[3]	VB[2]	VA[2]	VB[1]	VA[1]	VB[0]	VA[0]
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

Beta 数据的摆放规则:

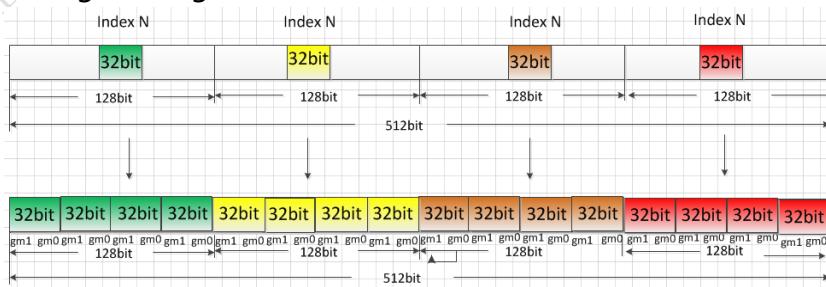
short 类型

VA[3]	VA[7]	VA[6]	VA[2]	VA[1]	VA[5]	VA[4]	VA[0]
-------	-------	-------	-------	-------	-------	-------	-------

byte 类型

VB[3]	VA[3]	VB[7]	VA[7]	VB[6]	VA[6]	VB[2]	VA[2]	VB[1]	VA[1]	VB[5]	VA[5]	VB[4]	VA[4]	VB[0]	VA[0]
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

- Gamma 的选择: 选项 Index0/Index1/ Index2/Index3, 表示在 T1/T5 寄存器中每 128bit 数据选择第几个 32bit 数据作为运算中的 gamma. 当数据类型为 Short 时, 32bit 的低 16bit 为 gamma0/gamma2, 高 16bit 为 gamma1/gamma3 值. 当数据类型为 Byte 时, 32bit 的高 16bit 中高 8bit(高位为另一套运算的 gamma)和低 8bit(低位为第一套运算的 gamma)为 gamma1/gamma3, 低 16bit 中高 8bit 和低 8bit 为 gamma0/gamma2. 每个 128bit 数据中选出的 32bit 数据, 做 4 倍复制扩充成 128bit 数据, 再与其他 4 个 32bit 扩充后的数据拼接成 512bit 数据. 以 Short 类型的 Beta 操作为例, 详细过程如下图 (图中 gm 表示 gamma):



- T7 寄存器的值是第 k-1 次迭代的结果, 即计算公式中的 $\text{beta}[k-1]/\alpha[k-1]$ 值.

Alpha/Beta 运算规则如下:

- Beta 操作: $\text{beta}(k)$ 和 $\text{gamma}(k)$ 表示第 k 次迭代结果, $\text{beta}(k-1)$ 和 $\text{gamma}(k-1)$ 表示第 k-1 次迭代结果, 则:

$$\text{beta}[0](k) = \max(\text{beta}[0](k-1) - \text{gamma}[0] + (k-1), \text{beta}[4](k-1) + \text{gamma}[0] - (k-1));$$

```

beta[1](k) = max(beta[4](k-1) - gamma[0] + (k-1), beta[0](k-1) + gamma[0] - (k-1));
beta[2](k) = max(beta[5](k-1) - gamma[1] + (k-1), beta[1](k-1) + gamma[1] - (k-1));
beta[3](k) = max(beta[1](k-1) - gamma[1] + (k-1), beta[5](k-1) + gamma[1] - (k-1));
beta[4](k) = max(beta[2](k-1) - gamma[1] + (k-1), beta[6](k-1) + gamma[1] - (k-1));
beta[5](k) = max(beta[6](k-1) - gamma[1] + (k-1), beta[2](k-1) + gamma[1] - (k-1));
beta[6](k) = max(beta[7](k-1) - gamma[0] + (k-1), beta[3](k-1) + gamma[0] - (k-1));
beta[7](k) = max(beta[3](k-1) - gamma[0] + (k-1), beta[7](k-1) + gamma[0] - (k-1));

```

Alpha 操作: alpha (k) 和 gamma(k) 表示第 k 次迭代结果, alpha (k-1) 和 gamma(k-1) 表示第 k-1 次迭代结果, 则:

```

alpha[0] (k) = max(alpha[0] (k-1) - gamma[2] + (k-1), alpha[1] (k-1) + gamma[2] - (k-1));
alpha[1] (k) = max(alpha[3] (k-1) - gamma[3] + (k-1), alpha[2] (k-1) + gamma[3] - (k-1));
alpha[2] (k) = max(alpha[4] (k-1) - gamma[3] + (k-1), alpha[5] (k-1) + gamma[3] - (k-1));
alpha[3] (k) = max(alpha[7] (k-1) - gamma[2] + (k-1), alpha[6] (k-1) + gamma[2] - (k-1));
alpha[4] (k) = max(alpha[1] (k-1) - gamma[2] + (k-1), alpha[0] (k-1) + gamma[2] - (k-1));
alpha[5] (k) = max(alpha[2] (k-1) - gamma[3] + (k-1), alpha[3] (k-1) + gamma[3] - (k-1));
alpha[6] (k) = max(alpha[5] (k-1) - gamma[3] + (k-1), alpha[4] (k-1) + gamma[3] - (k-1));
alpha[7] (k) = max(alpha[6] (k-1) - gamma[2] + (k-1), alpha[7] (k-1) + gamma[2] - (k-1));

```

Note: 对数据溢出进行饱和处理.

该指令的执行周期为 2 拍:

- 第一拍: 寄存器 T7 中存放的是 Alpha 或 Beta 数据, 寄存器 T1 中存放的是 gamma 的值, 更新 T5.

更新 T5 的方式:

- 1) T5Mode 选项存在: 当数据粒度为 Short, 寄存器 T7 中的数据, 每 128bit 按 Alpha/Beta 输入摆放顺序从高到低来进行判断, 取出第一个为正数的值 (如无正数, 选择 0), 用其与 gamma0/gamma2 和 gamma1/gamma3 分别相加形成 32bit 结果, 即 gamma0+/gamma2+ 和 gamma1+/gamma3+, 相减形成 32bit 结果, 即 gamma0-/gamma2- 和 gamma1-/gamma3-, 结果做饱和处理后扩展 2 倍, 按照从低到高为 gamma0+/gamma2+, gamma1+/gamma3+, gamma0-/gamma2-, gamma1-/gamma3-.....的顺序存入寄存器 T5. 数据粒度为 Byte 同理.
- 2) T5Mode 选项不存在: 当数据粒度为 Short, 取寄存器 T7 的每 128bit 的最低 16bit 数据, 用其与 gamma0/gamma2 和 gamma1/gamma3 分别相加形成 32bit 结果, 即 gamma0+/gamma2+ 和 gamma1+/gamma3+, 相减形成 32bit 结果, 即 gamma0-/gamma2- 和 gamma1-/gamma3-,

结果做饱和处理后扩展 2 倍, 按照从低到高为
 $\text{gamma0+}/\text{gamma2+}, \text{gamma1+}/\text{gamma3+}, \text{gamma0-}/\text{gamma2-}, \text{gamma1-}/\text{gamma3-}, \text{gamma0+}/\text{gamma2+}, \text{gamma1+}/\text{gamma3+}, \text{gamma0-}/\text{gamma2-}, \text{gamma1-}/\text{gamma3-}$的顺序存入寄存器 T5. 数据粒度为 Byte 同理.

- 第二拍: 寄存器 T7 中存放的是 Alpha 或 Beta 数据, 寄存器 T5 中存放的是 gamma 的值 ($\text{gamma+}(k-1)$ 和 $\text{gamma-}(k-1)$), 根据相应的运算规则计算出结果.

Note: 程序执行过程中的每一拍都会重新读取寄存器 T7 中的数据, 请确保数据的正确性.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3
						Read regs, Read MCW, Write T5	Read T7,T5	Write to MReg, IMA,SHU	Write to BIU

- Latency**
- > BIU: 4
 - > MReg: 2
 - > IMA: 3
 - > other SHU: 3
 - > self SHU: 2(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_turbo(const int type_BS, const int ab, const int f_UDT5, const int f_T5MD, const int index, const int mode, const int fu, const int pb)	SHUx: Turbo(T7, T1, T5) (Alpha Beta) (B S) (Index0 Index1 Index2 Index3) (UpdateT5) {(T5Mode)} -> Dest

Example SHU1.Turbo(T7, T1, T5) (Beta) (B) (Index0) (UpdateT5) -> SHU3.T6;

2.3.5.16. Sort 指令

Syntax	SRC	SHUx:Sort(T1,Ts1)(B S) (SortMode0/SortMode1/SortMode2/SortMode3) {(UpdateT1)} {(UpdateT5)} {(Mask)}
		->
	DST1	1. IMAx.Td 2. BIUx.Td 3. SHUn.Td 4. M[t {S++},{I++},{A++}] {(Wx)} 5. M[dis{shift}] {(Wx)} (LatchID) 6. IMA[{0},{1},{2},{3}].Td
	COND	({!}mode(0 1))

Note:

1. UpdateT1 选项存在时, 且目标域为本 SHU 时, 只能为本 SHU 的 T1, 不能为其他 T 寄存器;
2. UpdateT5 选项只在 SortMode0 下存在;
3. Mask 选项只在目标域为 SHU/MReg 时生效.

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	COND	10	11	1	0	S/ B	0	1	up dat eT 5	Ma sk	up dat eT 1	Ts1				
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts1	0	SortModes													

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.5 章, SHU 公共目标域及编码
SortModes	SortMode0: 0 SortMode1: 1 SortMode2: 2 SortMode3: 3
S/B	B: 0 S: 1

Description 将寄存器 T1 的 64Byte 数据根据 SortMode 分成 16 组, 按有符号数排序, 由寄存器 Ts1 每 Byte 最高位决定是从大到小(1)或从小到大(0)排列, 相等时认为低位为大数. 排序结果发往目标域.

- S 选项存在, 表示数据粒度为 Short. B 选项存在, 数据粒度为 Byte.
- 数据粒度为 Byte 时的分组方式分为四种情况:

-
1. SortMode0: Byte0/2/4/6 为一组、Byte1/3/5/7 为一组.....等相邻 8Byte 分为两组. 结果按分组后的顺序写入目标域.
 2. SortMode1: 每八组中前四组和 SortMode0 一致, 后四组使用前四组的排序结果作为自身排序的索引. 具体为 Byte0/2/4/6 为一组、Byte1/3/5/7 为一组, Byte8/10/12/14 为一组、Byte9/11/13/15 为一组.....等相邻 32Byte 分为八组. 结果按分组后的顺序写入目标域.
 3. SortMode2: Byte0/4/8/12 为一组、Byte1/5/9/13 为一组、Byte2/6/10/14 为一组、Byte3/7/11/15 为一组、Byte16/20/24/28 为一组、Byte17/21/25/29 为一组、Byte18/22/26/30 为一组、Byte19/23/27/31 为一组.....等相邻 32Byte 分为八组. 将第 0 组的排序结果放入 0/ 2/ 16/ 18, 第 1 组的排序结果放入 1/ 3/ 17/ 19, 第 2 组的排序结果放入 4/ 6/ 20/ 22....., 或者将 T5 中的数据带入, 并根据排序结果将结果中的 Byte16~31/48~63 替换为 T5 中的数据.
 4. SortMode3: 每八组中前四组和 SortMode2 一致, 后四组使用前四组的排序结果作为自身排序的索引. Byte0/4/8/12 为一组、Byte1/5/9/13 为一组、Byte2/6/10/14 为一组、Byte3/7/11/15 为一组.....等相邻 32Byte 分为八组. 将第 0 组的排序结果放入 0/ 2/ 8/ 10, 第 1 组的排序结果放入 1/ 3/ 9/11, 第 2 组的排序结果放入 4/ 6/ 12/ 14.....

- 选项 UpdateT1 表示会将结果同时发往 T1, 该选项与目标域为 T1 时作用相同, 该选项有效时目标域不能为本 SHU 的其他 T 寄存器.
- 选项 UpdateT5 表示会根据排序结果, 将 T5 中的数据每 Byte 中低三位根据排序结果进行重新排列, 并写入 T5, 高位不变. 只有 SortMode0 时支持该选项.
- 数据比较的粒度是 Short 时, 要求原始数据按照 8short 一组分为 4 组, 每组中 8 个 short 中每个 short 的低 8bit 取出拼接后放入 T1 寄存器中每组对应的低 64bit, 高 8bit 取出拼接后放入 T1 寄存器中每组对应的高 64bit. 随后根据 T1 中实际的值进行本条指令的计算. 排序之后的结果也按照该分组方式写入目标域. 即该选项仅会在比较时将 Byte0 与 Byte8 合为一个 Short, 将其与 Byte2 与 Byte10、Byte4 与 Byte12、Byte6 与 Byte14 分别组合得到的另三个 Short 比较小. 比较之后的计算过程与 Byte 模式完全相同. 只有 SortMode0、SortMode1 支持 S 选项.

-
- 选项 Mask 表示只发出结果的 0~63、128~191、256~319、384~447bit, 只有目标域为 SHU 和 MReg 时 Mask 选项才生效.
 - 寄存器 Ts1 中的值分为索引部分和排序方向部分, 其中每 byte 中低 6bit 为索引值, 使用方法同交织指令, 即根据分组方式与结果摆放方式决定, SortMode0/1 使用 byte 模式的索引方式; SortMode2/3 使用 short 模式的索引方式. 其中 T1 为交织中的高位, T5 为交织中的低位. 但索引值只有 bit0/3/4/5 有效, 例如: SortMode0 时即为 0/2/4/6/1/3/5/7.....SortMode2 时按 short 索引, 为 0/0/2/2/1/1/3/4/4.....寄存器 Ts1 中的每 byte 中最高 1bit 为排序方向, 和写入位置共同决定选择哪个数据, 即结果的 64Byte 分为 4 种情况, Byte0/4/8/12.....如果对应位置 Ts1 最高位为 1 则选取每组 4byte 数据中最大的数据, 为 0 则选取最小的数据; Byte1/5/9/13.....如果对应 Ts1 最高位为 1 则选取每组 4byte 数据中次大的数据, 为 0 则选取次小的数据; Byte2/6/10/14.....如果对应 Ts1 最高位为 1 则选取每组 4byte 数据中次小的数据, 为 0 则选取次大的数据; Byte3/7/11/15.....如果对应 Ts1 最高位为 1 则选取每组 4byte 数据中最小的数据, 为 0 则选取最大的数据.

例:

数据粒度 Byte, T1 中数据为 10 /20 /5 /9 /3 /8 /12 /14 /90 /0 /60 /40 /23 /35 /95 /18, T5 中数据为 0 /1 /2 /3 /4 /5 /6 /7 /8 /9 /10 /11 /12 /13 /14 /15, 进行 32 选 8 的排序操作. 步骤如下:

1. SortMode0: Ts1 数据为 0x0 /2 /4 /6 /81 /83 /85 /87 /88 /8a /8c /8e /9 /b /d /f, 选项 UpdateT5 使能, 目标发往 T1. T1 中数据变为 3 /5 /10 /12 /20 /14 /9 /8 /95 /90 /60 /23 /0 /18 /35 /40, T5 中数据变为 4 /2 /0 /6 /1 /7 /3 /5 /14 /8 /10 /12 /9 /15 /13 /11.
2. SortMode0: Ts1 数据为 0x0 /2 /4 /6 /81 /83 /85 /87 /88 /8a /8c /8e /9 /b /d /f, 选项 UpdateT5 使能, 目标发往 T1. T1 中数据变为 3 /9 /10 /20 /14 /12 /8 /5 /95 /60 /35 /0 /18 /23 /40 /90, T5 中数据变为 4 /3 /0 /1 /7 /6 /5 /2 /14 /10 /13 /9 /15 /12 /11 /8.
3. SortMode2: Ts1 数据为 0xa0 /a0 /22 /22 /a1 /a1 /23 /23 /a4 /a4 /26 /26 /a5 /a5 /27 /27 /80 /80 /02 /02 /81 /81 /03 /03 /84 /84 /06 /06 /85 /85 /07 /07, 目标发往 T1. T1 中数据变为 95 /23 /18 /60 /40 /20 /35 /90..... 14 /12 /15 /10 /11 /1 /13 /8.
4. SortMode1: Ts1 数据为 0x0 /2 /4 /6 /81 /83 /85 /87 /88 /8a /8c /8e /9 /b /d /f, 目标发往 T1. T1 中数据变为 18 /35 /40 /95 /90 /60 /23 /20..... 15 /13 /11 /14 /8 /10 /12 /1.
5. 后续再进行 1 次 SortMode1 和 1 次 SortMode3 即可完成 32 选 8 操作.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs, Write T5	Write to MReg, IMA, SHU	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > IMA: 2
 - > other SHU: 2
 - > self SHU: 1(bypass)

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_sort(ucp_vec512_t s1, const int type_BS, const int sortmode, const int f_MASK, const int f_UDT1, const int f_UDT5, const int mode, const int fu, const int pb)	SHUx: Sort(T1,Ts1)(B S) (SortMode0/SortMode1/SortMod e2/SortMode3) {{(UpdateT1)} {(UpdateT5)} {(Mask)} -> Dest

Example SHU1: Sort(T1,T2)(B) (SortMode0) (UpdateT5) (Mask) ->SHU0.T5;

2.3.5.17. Wait 立即数指令

Syntax	SHUx: Wait Imm6
Encoding	
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	01 11 10 0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	0 Imm6

Description 该指令将该 SHU 通道的指令延迟 Imm6 拍之后执行，最高延迟 62 拍。
具体行为参见 2.3.2.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_shuWaitImm(const int imm6, const int fu, const int pb)	SHUx: Wait Imm6

Example SHU0: Wait 32;

2.3.5.18. Wait KI 指令

Syntax	SRC	SHUx: WaitKI																																																
Encoding	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 <table border="1"> <tr> <td>01</td><td>11</td><td>10</td><td></td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td>0</td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td>0</td><td></td><td></td> </tr> </table>	01	11	10		0													15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						0				1				0		
01	11	10		0																																														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																			
					0				1				0																																					

Description 该指令将使 SHUx 的指令延迟执行, 最高延迟 62 拍. 具体行为参见 2.3.2.
 Wait 的周期数由 Kly[m:n]指定, 其中 y 表示使用哪个 KI, m:n 表示使用该 KI 中的哪些 bit.
 Note: 每个 KI 都是 24bit 寄存器, WaitKI 指令使用特定的 6bit 作为 wait 的周期数.

y, m, n 与发射槽的对应关系为:

发射槽	Kly	[m:n]
BIU0	KI8	[5:0]
BIU1	KI8	[11:6]
BIU2	KI8	[17:12]
BIU3	KI8	[23:18]
SHU0	KI9	[5:0]
SHU1	KI9	[11:6]
SHU2	KI9	[17:12]
SHU3	KI9	[23:18]
IMA0	KI10	[5:0]
IMA1	KI10	[11:6]
IMA2	KI10	[17:12]
IMA3	KI10	[23:18]
R0	KI11	[5:0]
R1	KI11	[11:6]
R2	KI11	[17:12]
R3	KI11	[23:18]
R4	KI12	[5:0]
R5	KI12	[11:6]
R6	KI12	[17:12]
R7	KI12	[23:18]

Execution

Pipeline | FT | DC | DP0

Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_shuWaitKI(const int fu, const int pb)	SHUx: WaitKI

Example SHU2: WaitKI;

2.3.5.19. NOP 指令

Syntax	SRC	NOP;
--------	------------	------

Encoding	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	COND		11		01							0				
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Encode field:

Field	Value
COND	参见 2.3 章

Description 空指令.

Execution



Latency 1

Example SHU3: NOP;

2.3.5.20. Set Condition 指令

Syntax	SHUx: Imm9 -> DST mode0
---------------	-------------------------------

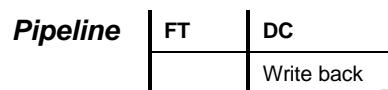
Encoding	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 COND 11 11 0 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0. 0 Imm9
-----------------	---

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令用 9 位立即数对 SHU 单元的 mode0 寄存器进行设置, 即配置指令执行的条件. mode0 寄存器的值对应的条件表达式见 2.3 的条件表达式值编码表.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_shuSetCond(const int imm9, const int fu, const int pb)	SHUx: Imm9 -> mode0

Example SHU2: 172 -> mode0;

2.3.6.BIU

BIU 公共目标域及编码

No	Assembly	9	8	7	6	5	4	3	2	1	0
1	* -> M['t] {(Wx)}	1	1:W 高 0:W 低					t			
2	* -> M[{{S++},{I++},{A++}}] {(Wx)}	0	1	1	Wx		0		A++	S++	I++
3	* -> M[dis{{shift}}] {(Wx)}	0	1	1	Wx		10: dis		shift	LatchID	
4	* -> SHU[{0},{1},{2},{3}].Td	0	1	0	SHU3	SHU2	SHU1	SHU0		Td	
5	* -> IMAX.Td	0	0	1	1	IMAX		0		Td	
6	BIU0: * -> BIU0.Td	0	0	1	0	1		0		Td	
7	* -> MFetch	0	0	1	0	0		0			

Note:

- 对于目标为 Dis 的情况, 目标具有可选的 LatchID:

BIU0/BIU2 可选的 LatchID 为: Latch0/Latch2/Latch4/Latch6, 其编码如下:

LatchID	Encoding
Latch0	0
Latch2	1
Latch4	2
Latch6	3

BIU1/BIU3 可选的 LatchID 为: Latch1/Latch3/Latch5/Latch7, 其编码如下:

LatchID	Encoding
Latch1	0
Latch3	1
Latch5	2
Latch7	3

- Wx 表示使用哪一个写口将结果写入 MReg 中, 但每个 BIU 能够使用的 W 口受到互联关系的限制, 参见 2.3.1.1.

UCP 中每个 APC 共有 2 个 APE 和 8 个 DM, 其中 APE0 的 MPU 可以访问 DM0~5, APE1 的 MPU 可以访问 DM2~7. 每个 APE 有 4 个 BIU 单元, 分别为 BIU0, BIU1, BIU2, BIU3, 其功能完全一样, 每个 BIU 有 4 个 T 寄存器 T0~T3. 各 BIU 单元可以同时访问同一个 DM, 但会导致 MPU Stall, 访问优先级为 BIU0>BIU1>BIU2>BIU3.

BIU 的运算指令均为无符号运算, 如若溢出, 对结果做截断处理.

对于 BIU 的 Load 和 Store 指令而言, 计算 DM 地址所需的 K 参数存放在指定的 T 寄存器中. 根据 Load/Store 操作的模式不同, K 参数在 T 寄存器中的位置也有所区别. 下面分别给出两种模式下 K 参数及其相应的位置信息.

递进模式下 K 参数及相应位置:

K 参数	配置位置	取值范围	行为
KB0	20:0	21bit 0x00000-0x1fffff	一维起始地址
KB1	52:32		二维起始地址
KB2	84:64		三维起始地址
KB3	116:96		四维起始地址

KS0	148:128	21bit 0 -0x1fffff	一维相邻两元素地址差
KS1	180:160		二维相邻两元素地址差
KS2	212:192		三维相邻两元素地址差
KS3	244:224		四维相邻两元素地址差
KI0	271:256	16bit 0x1-0xffff	KIx:当前 x 维度元素总量, 随着 Load/Store 操作, 值依次递减更新
KC0	287:272		KCx:x 维度元素总量, 值不更新
KI1	303:288		
KC1	319:304		
KI2	335:320		
KC2	351:336		
KBNum	405:384	22bit 0-0x3fffff	KB 的数量, 决定 Mask 信息
KG	418:416	3bit 0-6	数据摆放(偏移)粒度: 2^{KG}
KGCurrent	419:419	1bit 0-1	当前读写粒度: $2^{\text{KGCurrent}}$ 1: KGC=6; 0: KGC=KG
KSize	423:420	4bit 0-0xc	$(2^{\text{KSize}})^{*}64$ 个字节表示一个逻辑块的大小
RShiftEn	424	1bit 0-1	Store 时, 结果是否右移 1bit 再存入 DM(移位后高位补符号位).
RShiftS	425	1bit 0-1	Store 时, 右移 1bit 的粒度 1: Short 0: Byte
LoadExtEn	424	1bit 0-1	Load 时, 是否对读取出的数据进行低位扩展
LoadExtS	425	1bit 0-1	LoadExtEn 有效时, 决定扩展的数据粒度: 1: 使用最低 Short 扩展为 512bit 0: 使用最低 Byte 扩展为 512bit
KBNumSize	439:432	8bit 0-0xff	Mask 有效时, KBNum = KBNum - KBNumSize
KMEnable	440	1bit 0-1	KM 使能位
KM	511:448	64bit 0x0-0xffffffffffffffff	Mask 信息: 对应 Load/Store 的 64 字节数据哪些字节有效

循环模式下 K 参数及相应位置:

K 参数	配置位置	取值范围	行为
KB0	20:0	21bit 0x00000-0x1fffff	起始地址
KB0Num	31:21	11bit 0x000-0x7ff	KB0 的数量, Mask 信息
KB1	52:32		起始地址
KB1Num	63:53		KB1 的数量, Mask 信息
.....			
KB12	404:384		起始地址
KB12Num	415:405		KB12 的数量, Mask 信息
KG	418:416	3bit 0-6	数据摆放(偏移)粒度: 2^{KG}

KGCurrent	419:419	1bit 0-1	当前读写粒度: $2^{\text{KGCurrent}}$ 1: KGC=6; 0: KGC=KG
KSize	423:420	4bit 0-0xc	$(2^{\text{KSize}}) * 64$ 个字节表示一个逻辑块的大小
RShiftEn	424	1bit 0-1	Store 时, 结果是否右移 1bit 再存入 DM(移位后高位补符号位).
RShiftS	425	1bit 0-1	Store 时, 右移 1bit 的粒度 1: Short 0:Byte
LoadExtEn	424	1bit 0-1	Load 时, 是否对读取出的数据进行低位扩展
LoadExtS	425	1bit 0-1	LoadExtEn 有效时, 决定扩展的数据粒度: 1: 使用最低 Short 扩展为 512bit 0: 使用最低 Byte 扩展为 512bit
KBNumSize	439:432	8bit 0-0xff	指令中 Mask 有效时, KBNum = KBNum - KBNumSize
KMEnable	440	1bit 0-1	KM 使能位
KM	511:448	64bit 0-0xffffffffffffffffffff	Mask 信息: 对应 Load/Store 的 64 字节数据哪些字节有效

在未对 BIU 进行任何操作前, BIUx 的 T 寄存器中均存放着一组初始 K 参数, 用于起始的 DM load 操作, 其值根据 APE 和 BIU 编号分别为:

寄存器	向量值
APEx.BIUy.Tz	512' h00000000_00000000_00400006_00200000_00000000_00000000_00000000 80008000_00000000_00000000_00000000_00000040_00000000_00000000_00000000 000_00x(y*4)0000

T_z 表示对应 BIU 的 4 个 T 寄存器, 且初始值相同. BIU 的默认配置为 APE0BIU0 访问 DM0, 且初始访存地址为 0x0; APE0BIU1 访问 DM1, 初始访存地址为 0x40000; APE0BIU2 访问 DM2, 初始访存地址为 0x80000; APE0BIU3 访问 DM3, 初始访存地址为 0xc0000; APE1BIU0 访问 DM4, 且初始访存地址为 0x100000; APE1BIU1 访问 DM5, 初始访存地址为 0x140000; APE1BIU2 访问 DM6, 初始访存地址为 0x180000; APE1BIU3 访问 DM7, 初始访存地址为 0x1c0000. 初始步进 KS0=0x40, KC0=KI0=0x8000, KGC=KG=6, KBNumSize=0x40, 即默认为普通行读写模式.

DM 支持多粒度、多种模式的并行访问，BIU 支持所有访问模式。BIU Load/Store 指令的访问模式由具体的选项和 K 参数决定。

Load/Store 基址的产生方式如下：

访问模式	参数及选项	行为
递进方式	无 R 选项, 无 BR 选项	base_addr = KB0
循环方式	R 选项, 无 BR 选项	base_addr = KB0

位反序方式	无 R 选项, BR 选项	base_addr = KB0
-------	---------------	-----------------

当 BIU Load/Store 指令的 A++ 选项存在时, 寄存器中的 K 参数会自动进行递增:

访问模式	参数及选项	行为
递进方式	无 R 选项, 无 BR 选项	$KB = KB + KS$
循环方式	R 选项, 无 BR 选项	$KB0 = KB0 + 2^{KG}$ 当 Mask 有效时, 若 $KB0Num <= 0x40$, K 参数寄存器的 [415:0] 循环右移 32 位; 否则 $KB0Num = KB0Num - 0x40$.
位反序方式	无 R 选项, BR 选项	$KB = \text{inver}(\text{inver}(KB) + \text{inver}(KS))$ (此处的 inver 表示 21bit 位反序, 即: 第 0 位与第 20 位交换, 第 1 位与第 19 位交换, 以此类推.)

Load/Store 操作各种模式的配置方式及行为如下:

模式	参数及选项	行为
普通列模式	$KGCurrent != 6$	$Addr = base_addr$, 从地址 Addr 开始每个逻辑块读/写 2^{KG} 字节粒度数据, 拼为 512bit 数据
普通行模式	$KGCurrent = 6$	$Addr = base_addr$, 从地址 Addr 开始在一个逻辑块中读/写一行 512bit 数据
离散读模式 1	$KGCurrent=KG = 6$; 单指令: QL 选项	$Addr[k] = base_addr \{Ts[k*8+:8],6\{1' b0\}\} k$ (k:0~63) 根据离散地址 Addr[k] 分别读 64 字节, 拼为 512bit 数据
离散读模式 2	$KGCurrent=KG = 6$; 双指令: 1. L 选项 2. QH 选项	1. $\text{Temp_Addr}[k] = base_addr \{Ts[k*8+:8],6\{1' b0\}\}$ 2. $Addr[k] = base_addr \text{Temp_Addr}[k] \{Ts[k*8+:4],8' h0,6\{1' b0\}\} k$ (k:0~63) 根据离散地址 Addr[k] 分别读 64 字节, 拼为 512bit 数据
离散写模式 1	$KGCurrent=KG = 6$; 双指令: 1. L 选项 2. Q 选项	$Addr[k] = base_addr \{Ts[k*8+:8],6\{1' b0\}\} k$ (k:0~63) 将 512 位数据按字节写入每个 k 对应的地址 Addr[k]
离散写模式 2	$KGCurrent=KG = 6$; 三指令: 1. L 选项 2. H 选项 3. Q 选项	1. $\text{Temp_Addr}[k] = \{Ts[k*8+:8],6\{1' b0\}\}$ 2. $Addr[k] = base_addr \text{Temp_Addr}[k] \{Ts[k*8+:4],8' h0,6' h0\} k$ (k:0~63) 将 512 位数据按字节写入每个 k 对应的地址 Addr[k]

Note: 不同数据粒度下访问 DM 时, KSize 的最大取值:

KG	0	1	2	3	4	5	6
KSize	6	7	8	9	a	b	c

2.3.6.1. 加载指令: 普通模式

Syntax	BIUx: Load(Ts0){(A++)}{(Mask)}{(BR)}{(R)}
	->
DST1	1. M[t {S++},{I++},{A++}] {(Wx)} 2. M[dis{(shift)}] {(Wx)} (LatchID) 3. SHUn.Td 4. SHU[{0},{1},{2},{3}].Td 5. IMAx.Td 6. BIUx.Td 7. MFetch
DST2	& IMA.Mx(可选)
COND	({!}mode(0 1))

Note:

- 第二目标域 IMA.Mx 的配置方式和含义见 2.3.6.21, 不指定则默认为 IMA.M0
- LatchID 说明及编码见 2.3.6: BIU 公共目标域及编码
- 当第一目标域为 IMAx.Td 时, 配置的 IMA.Mx 中的 T 寄存器必须与 IMAx.Td 的 Td 相同

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	10	00		Mx	Mask	R	BR	A++	11				
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	00						DST1						

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.6 章, BIU 公共目标域及编码

Description 该指令根据寄存器 Ts0 中存放的 K 参数 (KBx, KSx, KIx, KG, KGCurrent, KM, KSize 等) 对 DM 进行列/行读取, 并将读取数据发往目标域.

关于选项:

- BR 选项存在, 表示基址由位反序方式产生. A++ 选项表示本次操作后, K 参数寄存器自动递增; 当 A++ 不存在时, K 参数不变. 基址产生方式和参数递增方式见 2.3.6 的详细描述.

- K 参数中 KMEnable 表示 KM 是否有效. KMEnable 为 1, KM 有效, 64bitKM 与读取的 64 字节数据一一对应, KM 为 1 的 bit 对应的字节有效. 若 KMEnable 为 0, 则全部 64 字节数据有效. KBNum 表示剩余的数据量, 如 KBNum 大于 0x40 则数据全部有效, 如小于则只有数据中小于 KBNum 的 Byte 才有效. 最终的有效数据为上述两种条件均有效时才有效.
- Mask 选项存在, 更新对应的 KBNum: KBNum = KBNum - KBNumSize; 选项不存在, KBNum 参数不变.
- 当 K 参数中的 LoadExtEn 为 1 时, 取出的数据要进行低位扩展. 若 LoadExtS 为 0, 则使用最低 Byte 扩展为 512bit; 若 LoadExtS 为 1, 则使用最低 Short 扩展为 512bit. LoadExtEn 为 1 时, 认为 512bit 数据都有效.

该条指令支持三种访问模式:

- 递进模式: 无 R 选项, 无 BR 选项, 基址为 KB0. A++ 选项存在, 本次操作后 Ts0 中的 K 参数自动递增.
- 循环模式: R 选项存在, 无 BR 选项, 基址为 KB0. A++ 选项存在, 本次操作后 Ts0 中的 KB0 更新, $KB0 = KB0 + 2^{KGC}$. Mask 选项存在, 若 $KB0Num \leq KBNumSize$, Ts0 的 bit[415:0] 循环右移 32 位; 否则 $KB0Num = KB0Num - KBNumSize$.
- 位反序模式: 无 R 选项, BR 选项存在, 基址为 KB0. A++ 选项存在, 本次操作后 Ts0 中 KB = inver(inver(KB) + inver(KS)).

对于 DST1

- SHUn.Td, BIUx.Td, IMAX.Td: 将 Load 出的数据写入指定的寄存器.
- SHU[{0},{1},{2},{3}].Td: 将 Load 出的数据同时写入多个 SHU 的相同寄存器.
- Mfetch: 将数据写入 KI 寄存器, 向量值和 KI 寄存器对应关系如下:

511:480	479:448	447:416	415:384	383:352	351:320	319:288	287:256
KI15	KI14	KI13	KI12	KI11	KI10	KI9	KI8
255:224	223:182	191:160	159:128	127:96	95:64	63:32	31:0

KI7	KI6	KI5	KI4	KI3	KI2	KI1	KI0
-----	-----	-----	-----	-----	-----	-----	-----

- $M[t | \{S++\}, \{I++\}, \{A++\}] \{(Wx)\}$
- $M[dis\{(shift)\}] \{(Wx)\}$ (LatchID): 将读出数据写入 MReg 的指定地址处, 寻址方式见 2.3.4 的详细描述.

对于 DEST2

- IMA.Mx: 每个 BIU 有四组 IMA 第二目标域寄存器, 由 BIU Move 指令进行配置. 寄存器默认表示结果不发送给第二目标域; 当该寄存器被配置后, 则可以根据寄存器中的值将 Load 结果同时发往第二目标域.

Note: 当第一目标域为 IMA 时, 第二目标域指定的 T 寄存器必须与第一目标域相同. 配置参数的含义见 2.3.6.21 BIUMove 指令.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9
						Read regs			Read DM			Read MCW		Write to BIU, MFetch	Write to SHU, IMA, MReg

Latency

- > BIU: 9
- > M: 9
- > IMA: 10
- > SHU: 10
- > MFetch: 13

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_load(ucp_vec512_t s0, const int f_APP, const int f_MASK, const int f_BR, const int f_R, const int mode, const int fu, const int pb)</code>	BIUx: Load(Ts0){(A++)}{(Mask)}{(BR)}{(R)} -> Dest

Macro	Corresponding Intrinsic
<code>ucpm2_load_postinc(s0, mask, br, r, mode, fu, pb)</code>	<code>__ucpm2_load(s0, f_APP, mask, br, r, mode, fu, pb); s0 = __ucpm2_inc()</code>
<code>ucpm2_load(s0, mask, br, r, mode, fu, pb)</code>	<code>__ucpm2_load(s0, 0, mask, br, r, mode, fu, pb)</code>

Note:

`__ucpm2_inc` 是编译器内部用于地址自增的 intrinsic.

Example

- BIU0: Load(T0) (A++) (BR) -> M[45] & IMA.M0(Mode0);
- BIU0: Load(T0) (A++) (Mask) (R) -> BIU0.T2 & IMA.M1(Mode0);

2.3.6.2. 加载指令: 离散模式

Syntax	SRC	BIUx: DisLoad(Ts0,Ts1) {(A++)} (QL QH L) {(Mask)} {(BR)} {(R)}
		->
	DST1	1. M[t {S++},{I++},{A++}] {(Wx)} 2. M[dis{(shift)}] {(Wx)} (LatchID) 3. SHUn.Td 4. SHU[{0},{1},{2},{3}].Td 5. IMAX.Td 6. BIUx.Td 7. MFetch
	DST2	& IMA.Mx(可选)
	COND	({!}mode(0 1))

Note:

- 第二目标域 IMA.Mx 的配置方式和含义见 2.3.6.21, 不指定则默认为 IMA.M0
- LatchID 说明及编码见 2.3.6: BIU 公共目标域及编码
- 当第一目标域为 IMAX.Td 时, 配置的 IMA.Mx 中的 T 寄存器必须与 IMAX.Td 的 Td 相同

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	10	00		Mx	Mask	R	BR	A++	QHL				
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	Ts1						DST1						

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.6 章, BIU 公共目标域及编码
QHL	00:L 01:QH 10:QL

Description 该指令根据指定寄存器中存放的基址和偏移地址, 生成每个字节的访存地址, 对 DM 进行读取, 并将读取数据发往目标域.

关于选项:

- BR 选项存在, 表示基址 base_addr 由位反序方式产生. A++选项表示本次操作后, K 参数寄存器自动递增; 当 A++不存在时, K 参数不变. 基址产生方式和参数递增方式见 2.3.6 的详细描述.
- R 选项, Mask 选项, K 参数以及目标域的相关说明参见 2.3.6.1 普通模式加载指令.

该指令访存地址的生成有两种方式:

- 单指令模式: 该模式仅需要一条带 QL 选项的指令即可完成读取. 此时, Ts0 中的 KB0 确定 base_addr, Ts1 决定 DM 内部偏移地址的低位, 则第 k 个字节的地址为: $Addr[k] = base_addr | \{Ts1[k*8+:8],6\{1' b0\}\} | k$.
- 双指令模式: 该模式下需要两条指令指定寄存器.
 1. 第一条指令: L 选项存在, Ts1 寄存器决定 DM 内部偏移地址的低位, 得到地址 $Temp_Addr[k] = base_addr | \{Ts1[k*8+:8],6\{1' b0\}\}$. 该条指令不读取数据.
 2. 第二条指令: QH 选项存在, Ts0 中的 KB0 确定 base_addr, Ts1 寄存器决定 DM 内部偏移地址的高位, 则第 k 个字节的地址为: $Addr[k] = base_addr | Temp_Addr[k] | \{Ts1[k*8+:4],8\{1' h0,6\{1' b0\}\}\} | k$. 该条指令读取数据.

Note:

1. 该模式要求两条指令的 Ts0 所指定的 DM 必须是同一个.
2. 双指令离散访问, 如果两条指令间插入了更高优先级的 Load 或者 Store, 并且与它访问相同 DM, 则离散访问会出错.

Note: KB 需配置为 64 字节对齐.

Execution

n

<i>Pipeline</i>	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9
						Read regs			Read DM				Read MCW	Write to BIU, MFetch	Write to SHU, IMA, MReg

Latency -> BIU: 9

- > M: 9
- > IMA: 10
- > SHU: 10
- > MFetch: 13

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_disLoad(ucp_vec512_t s0, ucp_vec512_t s1, const int f_APP, const int qhl, const int f_MASK, const int f_BR, const int f_R, const int mode, const int fu, const int pb)</code>	BIUx: DisLoad(Ts0,Ts1) {(A++)} (QL QH L) {(Mask)} {(BR)} {(R)} -> Dest

Macro	Corresponding Intrinsic
<code>ucpm2_disLoad_postinc(s0, s1, qhl, mask, br, r, mode, fu, pb)</code>	<code>__ucpm2_disLoad(s0, s1, f_APP, qhl, mask, br, r, mode, fu, pb); s0 = __ucpm2_inc()</code>
<code>ucpm2_disLoad(s0, s1, qhl, mask, br, r, mode, fu, pb)</code>	<code>__ucpm2_disLoad(s0, s1, 0, qhl, mask, br, r, mode, fu, pb)</code>

Note:

参数 qhl 应为: f_QL, f_QH, f_L;

`__ucpm2_inc` 是编译器内部用于地址自增的 intrinsic.

Example BIU0: DisLoad(T3,T0) (A++) (QL) (Mask)(R) -> M[0] & IMA0.M0(Mode0);

2.3.6.3. 加载指令: 默认模式

Syntax	BIUx: DefLoad(Ts0){(A++)}{(BR)}
	->
DST1	1. M[t {S++},{I++},{A++}] {(Wx)} 2. M[dis{(shift)}] {(Wx)} (LatchID) 3. SHUn.Td 4. SHU[{0},{1},{2},{3}].Td 5. IMAx.Td 6. BIUx.Td 7. MFetch
DST2	& IMA.Mx(可选)
COND	({!}mode(0 1))

Note:

- 第二目标域 IMA.Mx 的配置方式和含义见 2.3.6.21, 不指定则默认为 IMA.M0
- LatchID 说明及编码见 2.3.6: BIU 公共目标域及编码
- 当第一目标域为 IMAx.Td 时, 配置的 IMA.Mx 中的 T 寄存器必须与 IMAx.Td 的 Td 相同

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	10		01		Mx	0	0	BR	A++		11		
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0	01							DST1					

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.6 章, BIU 公共目标域及编码

Description 该指令根据寄存器 Ts0 中存放的基址 KBO 和复位时使用的默认参数(即:

- KG=KGC=6, KS0=0x40, KMEnable=0)对 DM 进行读取.

关于选项:

- BR 选项存在, 表示基址由位反序方式产生. A++ 选项表示本次操作后, K 参数寄存器自动递增; 当 A++ 不存在时, K 参数不变. 基址产生

方式和参数递增方式见 2.3.6 的详细描述. 目标域的相关说明参见
2.3.6.1 普通模式加载指令.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9
						Read regs			Read DM				Read MCW	Write to BIU, MFetch	Write to SHU, IMA, MReg

Latency

- > BIU: 9
- > M: 9
- > IMA: 10
- > SHU: 10
- > MFetch:13

Example BIU1: DefLoad(T3)(BR) -> IMA2.T3 & IMA.M0(Mode0);

2.3.6.4. 存储指令: 普通模式

Syntax	BIUx: Store(Ts1, Ts0){(A++)}{(Mask)}{(BR)}{(R)}
COND	{(!)mode(0 1)}

Encoding	27 26 25 24 23 22 21 20 19 18 17 16 15 14
	COND 10 10 0 Mask R BR A++ 11
	13 12 11 10 9 8 7 6 5 4 3 2 1 0
	Ts0 Ts1 0

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令根据寄存器 Ts0 中存放的 K 参数 (KBx, KSx, Klx, KG, KGCurrent,

- KM, KSize 等), 将 Ts1 寄存器中的数据按列/行存储到 DM 中.

关于选项:

- BR 选项存在, 表示基址由位反序方式产生. A++ 选项表示本次操作后, K 参数寄存器自动递增; 当 A++ 不存在时, K 参数不变. 基址产生方式和参数递增方式见 2.3.6 开头部分的详细描述.
- K 参数中 KMEnable 表示 KM 是否有效. KMEnable 为 1, KM 有效, 64bitKM 与读取的 64 字节数据一一对应, KM 为 1 的 bit 对应的字节有效. 若 KMEnable 为 0, 则全部 64 字节数据有效. KBNum 表示剩余的数据量, 如 KBNum 大于 0x40 则数据全部有效, 如小于则只有数据中小于 KBNum 的 Byte 才有效. 最终的有效数据为上述两种条件均有效时才有效.
- Mask 选项存在, 更新对应的 KBNum: KBNum=KBNum-KBNumSize; 选项不存在, KBNum 参数不变.
- 当 K 参数中的 RShiftEn 为 1 时, Ts1 中的数据要右移 1bit 再存入 DM. 若 RShiftS 为 0, 则移位的数据粒度是 Byte; 若 RShiftS 为 1, 则移位的数据粒度是 Short. 移位结果的高位补符号位.

该条指令支持三种访问模式:

- 递进模式: 无 R 选项, 无 BR 选项, 基址为 KB0. A++ 选项存在, 本次操作后 Ts0 中的 K 参数自动递增.
- 循环模式: R 选项存在, 无 BR 选项, 基址为 KB0. A++ 选项存在, 本次操作后 Ts0 中的 KB0 更新, $KB0 = KB0 + 2^{KG.C}$. Mask 选项存

在, 若 KB0Num <= KBNumSize, Ts0 的 bit[415:0]循环右移 32 位;
否则 KB0Num = KB0Num – KBNumSize.

- 位反序模式: 无 R 选项, BR 选项存在, 基址为 KB0. A++ 选项存在,
本次操作后 Ts0 中 KB = inver(inver(KB)+inver(KS)).

Note: 当所有数据都 Store(KI=1)完, 但程序中仍有 Store 操作时, 地址
将跳变到 KB3 所指向的地址继续 Store.

Execution



Latency 4

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_store(ucp_vec512_t s1, ucp_vec512_t s0, const int f_APP, const int f_MASK, const int f_BR, const int f_R, const int mode, const int fu, const int pb)	BIUx: Store(Ts1, Ts0){(A++)}{(Mask)}{(BR)}{(R)}

Macro	Corresponding Intrinsic
ucpm2_store_postinc(s1, s0, mask, br, r, mode, fu, pb)	__ucpm2_store(s1, s0, f_APP, mask, br, r, mode, fu, pb); s0 = __ucpm2_inc()
ucpm2_store(s1, s0, mask, br, r, mode, fu, pb)	__ucpm2_store(s1, s0, 0, mask, br, r, mode, fu, pb)

Note:

__ucpm2_inc 是编译器内部用于地址自增的 intrinsic.

Example BIU0: Store(T2, T0)(A++) (BR) (Mode0);

2.3.6.5. 存储指令: 离散模式

Syntax	Src	BlUx: DisStore(Ts1, Ts0)(L H Q){(A++)}{(Mask)}{(BR)}{(R)}
	COND	{(!)mode(0 1)}

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND		10		10		0		Mask	R	BR	A++		LHQ
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0		Ts1						0					

Encode field:

Field	Value
COND	参见 2.3 章
LHQ	00:L 01:H 10:Q

Description 该指令根据指定寄存器中存放的基址和偏移地址, 生成每个字节的访存地址, 将指定寄存器中的数据存储到 DM 中.

BR 选项存在, 表示基址 base_addr 由位反序方式产生. A++ 选项表示本次操作后, K 参数寄存器自动递增; 当 A++ 不存在时, K 参数不变. 基址产生方式和参数递增方式见 2.3.6 开头部分的详细描述.

关于选项:

- 当 K 参数中的 RShiftEn 为 1 时, 存储的数据要右移 1bit 再存入 DM. 若 RShiftS 为 0, 则移位的数据粒度是 Byte; 若 RShiftS 为 1, 则移位的数据粒度是 Short. 移位结果的高位补符号位.
- R 选项, Mask 选项, K 参数的相关说明参见 2.3.6.4 普通模式存储指令.

该指令访存地址的生成有两种方式:

- 双指令模式: 该模式需要两条指令完成存储.
 1. 第一条指令: L 选项存在, Ts0 中的 KB0 确定 base_addr, Ts1 寄存器决定 DM 内部偏移地址的低位, 则第 k 个字节的地址为:

$$\text{Addr}[k] = \text{base_addr} | \{\text{Ts1}[k*8+:8], 6\{1' b0\}\} | k$$
. 该条指令不发送数据.
 2. 第二条指令: Q 选项存在, 将 Ts1 寄存器中的数据发送到第一条指令决定的 DM 地址处.
- 三指令模式: 该模式下需要三条指令完成存储.

- 第一条指令: L 选项存在, Ts1 寄存器决定 DM 内部偏移地址的低位, 得到地址 Temp_Addr[k] = {Ts1[k*8+:8],6{1' b0}}. 该条指令不发送数据.
- 第二条指令: H 选项存在, Ts0 中的 KB0 确定 base_addr, Ts1 寄存器决定 DM 内部偏移地址的高位, 则第 k 个字节的地址为:

$$Addr[k] = base_addr | Temp_Addr[k] | \{Ts1[k*8+:4],8' h0, 6' h0\} | k$$
. 该条指令不发送数据.
- 第三条指令: Q 选项存在, 将 Ts1 寄存器中的数据发送到前两条指令决定的 DM 地址处.

Note:

- KB 需配置为 64 字节对齐.
- Ts0 所指定的 DM 必须是同一个.
- 离散访问, 如果两条指令间插入了更高优先级的 Load 或者 Store, 并且与它访问相同 DM, 则离散访问会出错.

Execution



Latency 4

Intrinsics	Intrinsic	Generated Instruction
	<pre>void __ucpm2_disStore(ucp_vec512_t s1, ucp_vec512_t s0, const int f_APP, const int f_MASK, const int f_BR, const int f_R, const int lhq, const int mode, const int fu, const int pb)</pre>	<pre>BIUx: DisStore(Ts1, Ts0)(L H Q){(A++)}{(Mask)}{(BR)} {(R)}</pre>

Macro	Corresponding Intrinsic
<code>ucpm2_disStore_postinc(s1, s0, mask, br, r, lhq, mode, fu, pb)</code>	<code>__ucpm2_disStore(s1, s0, f_APP, mask, br, r, lhq, mode, fu, pb);</code> <code>s0 = __ucpm2_inc()</code>
<code>ucpm2_disStore(s1, s0, mask, br, r, lhq, mode, fu, pb)</code>	<code>__ucpm2_disStore(s1, s0, 0, mask, br, r, lhq, mode, fu, pb)</code>

Note:

参数 lhq 应为: f_L, f_H, f_Q;

`__ucpm2_inc` 是编译器内部用于地址自增的 intrinsic.

Example 三指令模式:

BIU3: DisStore(T3, T3)(L)(A++) (Mode0);

BIU3: DisStore(T3, T3)(H)(A++) (Mode0);

BIU3: DisStore(T2, T3)(Q)(A++) (Mode0);

Smart Logic Confidential For yuyi@ruijie.com.cn

2.3.6.6. 存储指令: 默认模式

Syntax	SRC	BIUx: DefStore(Ts1, Ts0){(A++)}{(BR)}												
	COND	{(!)mode(0 1)}												
Encoding														
	27	26	25	24	23	22	21	20	19	18	17	16	15	14
		COND	10		11		0	0	0	BR	A++		11	
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Ts0	Ts1						0					

Encode field:

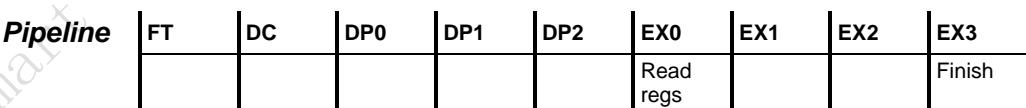
Field	Value
COND	参见 2.3 章

Description 该指令根据寄存器 Ts0 中存放的基址 KB0 和复位时使用的默认参数(即: KG=KGC=6, KS0=0x40, KMEnable=0), 将 Ts1 中的数据写入 DM. BR 选项存在, 表示基址由位反序方式产生. A++ 选项表示本次操作后, K 参数寄存器自动递增; 当 A++ 不存在时, K 参数不变. 基址产生方式和参数递增方式见 2.3.6 的详细描述.

当 K 参数中的 RShiftEn 为 1 时, 存储的数据要右移 1bit 再存入 DM. 若 RShiftS 为 0, 则移位的数据粒度是 Byte; 若 RShiftS 为 1, 则移位的数据粒度是 Short. 移位结果的高位补符号位.

Note: 当所有数据都 Store(KI=1)完, 但程序中仍有 Store 操作时, 地址将跳变到 KB3 所指向的地址继续 Store.

Execution



Latency 4

Example BIU3: DefStore(T3, T1) (A++);

2.3.6.7. 传送指令 BIUKG

Syntax

传送指令 0: 512bit 数据传送

SRC	BIUx: KG(Ts0)(All)
	->
DST1	1. M[t {S++},{I++},{A++}] {(Wx)} 2. M[dis{(shift)}] {(Wx)} (LatchID) 3. SHUn.Td 4. SHU[{0},{1},{2},{3}].Td 5. IMAX.Td 6. BIUx.Td 7. MFetch
COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.6: BIU 公共目标域及编码

传送指令 1: 32bit 扩展为 512bit 数据传送

SRC	BIUx: KG(Ts0[i])
	->
DST1	1. IMAX.Td 2. BIUx.Td
COND	({!}mode(0 1))

传送指令 2: 32bit 数据传送

SRC	BIUx: KG(Ts0[i])
	->
DST1	1. M[t {S++},{I++},{A++}] [k] {(Wx)} 2. M[dis{(shift)}] [k] {(Wx)} (LatchID) 3. KI[x] 4. SHUn.Td[k] 5. SHU[{0},{1},{2},{3}].Td[k]
COND	({!}mode(0 1))

Encoding

传送指令 0: 512bit 数据传送

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	00	1	DST1[9]	DST1[8:7]		0		0		11			
13	12	11	10	9	8	7	6	5	4	3	2	1	0
DST1[6]	1	Ts0		0						DST1[5:0]			

传送指令 1: 32bit 扩展为 512bit 数据传送

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	00	1	DST1[9]	DST1[8:7]		0						11	
13	12	11	10	9	8	7	6	5	4	3	2	1	0
DST1[6]	0	Ts0	i									DST1[5:0]	

传送指令 2: 32bit 数据传送

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	00	1	DST1[9]	DST1[8:7]		k/Klx						11	
13	12	11	10	9	8	7	6	5	4	3	2	1	0
DST1[6]	0	Ts0	i									DST1[5:0]	

Encode field:

Field	Value
COND	参见 2.3 章
DST1	参见 2.3.6 章, BIU 公共目标域及编码

Description 该指令将寄存器 Ts0 中的值传送至目标域.

- 传送指令 0: 将寄存器 Ts0 中的数据全部发送给目标域.
- 传送指令 1: 将寄存器 Ts0 的第 i 个 Word 扩展 16 倍得到 512bit, 然后发往目标域.
- 传送指令 2: 将寄存器 Ts0 的第 i 个 Word 发送给 MReg 或 SHU 寄存器的第 k 个 Word 或者 Klx.
- 目标域的含义参见 2.3.6.1.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
						Read regs, Read MCW	Write to BIU, MFetch	Write to SHU, IMA,MReg

Latency -> BIU: 2

-> MFetch: 6

-> MReg: 2

-> SHU: 3

-> IMA: 3

Example BIU0: KG(T2)(All) -> M[0](Mode0);

BIU1: KG(T1[11]) -> IMA1.T4(Mode0);

2.3.6.8. 标量加法指令 BIUAddW

Syntax	SRC	BIUx: Ts0[i] + Ts1[j]
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
			COND	00	00		Td			k			00	
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				00	Ts0	i		Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 的第 i 个 Word 和 Ts1 的第 j 个 Word 相加, 计算结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU2: T1[0] + T2[8] -> T3[0](Mode0);

2.3.6.9. 标量减法指令 BIUSubW

Syntax	SRC	BIUx: Ts0[i] - Ts1[j]
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	00		00		Td			k			01		
13	12	11	10	9	8	7	6	5	4	3	2	1	0
00		Ts0		i			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 的第 i 个 Word 和 Ts1 的第 j 个 Word 相减, 计算结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU2: T1[1] - T2[7] -> T3[5](Mode0);

2.3.6.10. 标量反序加法指令 BIUAddWR

Syntax	SRC	BIUx: Ts0[i] + Ts1[j] (R)
		->
	DST	Td[k]
	COND	({!}mode(0 1))

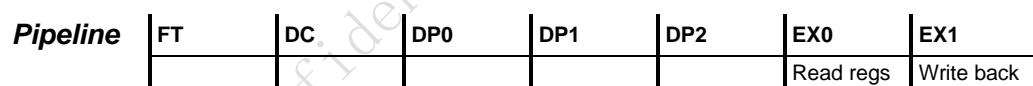
Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND	00		00		Td			k			10		
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00		Ts0		i			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 的第 i 个 Word 和 Ts1 的第 j 个 Word 进行位反序, 然后相加, 计算结果再次进行位反序, 最终结果写入 Td 的第 k 个 Word. 反序的规则为: 高低位对称交换, 即第 0 位与第 31 位交换, 第 1 位与第 30 位交换, 如此依次交换到第 15 位与第 16 位.

Execution



Latency 2

Intrinsics	Intrinsic	Generated Instruction
	int __ucpm2_addR(int s0, int s1, const int f_R, const int mode, const int fu, const int pb)	BIUx: Ts0[i] + Ts1[j] (R) -> Td[k]

Example BIU2: T1[0] + T2[8] (R) -> T3[0](Mode0);

2.3.6.11. 标量反序减法指令 BIUSubWR

Syntax	SRC	BIUx: Ts0[i] - Ts1[j] (R)
		->
	DST	Td[k]
	COND	({!}mode(0 1))

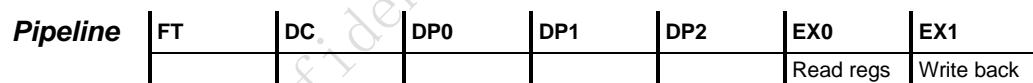
Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND	00		00		Td			k			11		
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00		Ts0		i			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 的第 i 个 Word 和 Ts1 的第 j 个 Word 进行位反序, 然后相减, 计算结果再次进行位反序, 最终结果写入 Td 的第 k 个 Word. 反序的规则为: 高低位对称交换, 即第 0 位与第 31 位交换, 第 1 位与第 30 位交换, 如此依次交换到第 15 位与第 16 位.

Execution



Latency 2

Intrinsics	Intrinsic	Generated Instruction
	int __ucpm2_subR(int s0, int s1, const int f_R, const int mode, const int fu, const int pb)	BIUx: Ts0[i] - Ts1[j] -> Td[k]

Example BIU2: T1[1] - T2[7] (R) -> T3[5](Mode0);

2.3.6.12. 标量按位与指令 BIUAnd

Syntax	SRC	BIUx: Ts0[i] & Ts1[j]
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	01	00		Td		k					00		
13	12	11	10	9	8	7	6	5	4	3	2	1	0
						i		Ts1		j			
00	Ts0												

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 的第 i 个 Word 和 Ts1 的第 j 个 Word 按位与, 结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU2: T1[2] & T2[6] -> T3[0](Mode0);

2.3.6.13. 标量按位或指令 BIUOr

Syntax	SRC	BIUx: Ts0[i] Ts1[j]
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	01		00		Td			k			01		
13	12	11	10	9	8	7	6	5	4	3	2	1	0
00		Ts0		i			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 的第 i 个 Word 和 Ts1 的第 j 个 Word 按位或, 结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU2: T1[2] | T2[6] -> T3[0](Mode0);

2.3.6.14. 标量按位异或指令 BIUXor

Syntax	SRC	BIUX: Ts0[i] ^ Ts1[j]
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	01		00		Td			k				10	
13	12	11	10	9	8	7	6	5	4	3	2	1	0
00		Ts0		i			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 的第 i 个 Word 和 Ts1 的第 j 个 Word 按位异或，结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU2: T1[2] ^ T2[6] -> T3[0](Mode0);

2.3.6.15. 标量按位非指令 BIUInv

Syntax	SRC	BIUx: ~ Ts1[j]
		->
	DST	Td[k]
	COND	({!}mode(0 1))

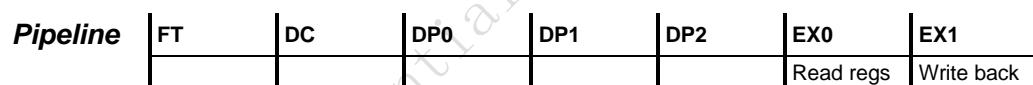
Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND	01		00		Td			k			11		
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00		0		0			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts1 的第 j 个 Word 按位取非, 结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU2: ~T1[2] -> T3[0](Mode0);

2.3.6.16. 标量比较指令

Syntax	SRC	BIUx: Ts0[j] == 1 0
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND	01		11		Td			k			10		
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00	0	Opr		0			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章
Opr	0: == 0 1: == 1

Description 该指令将寄存器 Ts0 的第 j 个 Word 与 0 或 1 进行比较, 结果写入 Td 的第 k 个 Word.

Execution



Intrinsics

Latency 2

Intrinsic	Generated Instruction
int __ucpm2_compare0(int s0, const int mode, const int fu, const int pb)	BIUx: Ts0[j] == 0 -> Td[k]
int __ucpm2_compare1(int s0, const int mode, const int fu, const int pb)	BIUx: Ts0[j] == 1 -> Td[k]

Example BIU0: T3[13] == 0 -> T2[0](Mode0);

2.3.6.17. MaskGen 指令

Syntax	BIUx: MaskGen(Ts0[j])
	->
DST	Td[k]
COND	({!}mode(0 1))

Note: $0 \leq k < 8$

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND	01		11		Td		k		0		11		
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00		01		0			Ts1		j				

Encode field:

Field	Value
COND	参见 2.3 章

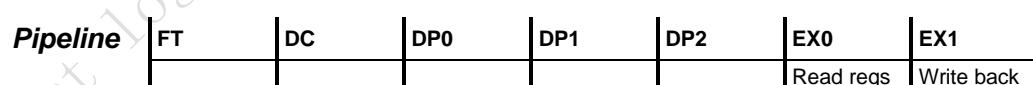
Description 该指令根据寄存器 Ts0 的第 j 个 Word 生成 64bit 的 Mask 信息, 结果写入 Td 的第 k 个 64bit 处.

生成 Mask 信息的规则如下:

Ts0 的第 j 个 Word 的低 16bit 决定起始地址, 高 16bit 决定终止地址.

若起始地址小于等于终止地址, 则 Mask 的 bit 中起始地址~(终止地址-1)为 1, 其余为 0. 若起始地址大于终止地址, 则 Mask 的 bit 中 0~(终止地址-1)和起始地址~63 为 1, 其余为 0.

Execution



Latency 2

Intrinsics	Intrinsic	Generated Instruction
	int __ucpm2_maskGen(int s0, const int mode, const int fu, const int pb)	BIUx: MaskGen(Ts0[j]) -> Td[k]

Example BIU0: MaskGen(T1[8]) -> T3[0](Mode0);

2.3.6.18. 标量立即数左移/右移指令

Syntax	SRC	1. BIUx: Ts0[i] << Imm5 2. BIUx: Ts0[i] >> Imm5
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding

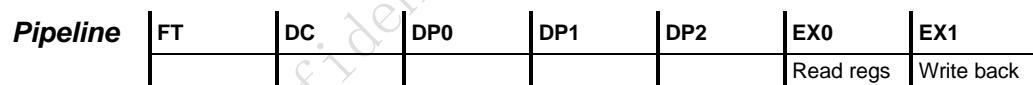
27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	01		10		Td		k		0	Opc			
13	12	11	10	9	8	7	6	5	4	3	2	1	0
00		Ts0		i		0		Imm5					

Encode field:

Field	Value
COND	参见 2.3 章
Opc	0: << 1: >>

Description 该指令将寄存器 Ts0 的第 i 个 Word 左移或右移 Imm5 位, 结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU1: T1[6] << 3 -> T3[7](Mode0);

2.3.6.19. 标量寄存器左移/右移指令

Syntax	SRC	1. BIUx: Ts0[i] << Ts1[j] 2. BIUx: Ts0[i] >> Ts1[j]
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	01		10		Td			k		1	Opc		
13	12	11	10	9	8	7	6	5	4	3	2	1	0

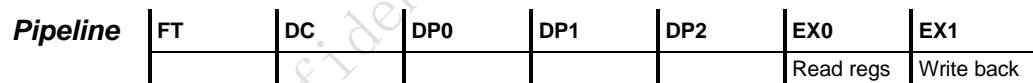
00	Ts0	i	Ts1	j
----	-----	---	-----	---

Encode field:

Field	Value
COND	参见 2.3 章
Opc	0:<< 1:>>

Description 该指令对寄存器 Ts0 的第 i 个 Word 进行左移或右移，移位的位数为 Ts1 第 j 个 Word 的低 5bit 值，结果写入 Td 的第 k 个 Word.

Execution



Latency 2

Example BIU1: T1[6] << T2[3] -> T3[7](Mode0);

2.3.6.20. 标量立即数赋值指令

Syntax	SRC	1. BIUx: Imm16 {(H)} 2. BIUx: Imm16 (E) {(O)}
		->
	DST	Td[k]
	COND	({!}mode(0 1))

Encoding

1. BIUx: Imm16 {(H)} -> Td[k]

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND		11	0	H	Td			k				Imm16	
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm16													

2. BIUx: Imm16 (E) {(O)} -> Td[k]

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND		11	1	O	Td			k				Imm16	
13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm16													

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令用立即数对 BIUx 的寄存器 Td 进行赋值操作, k 为选择第几个字.

1. BIUx: Imm16 {(H)} -> Td[k]: (16bit 立即数赋值)

- H 选项存在时, 将立即数写入 Td 寄存器第 k 个字的高 16 比特.
- H 选项不存在时, 将立即数写入 Td 寄存器第 k 个字的低 16 比特.

2. BIUx: Imm16 (E) {(O)} -> Td[k]: (16bit 扩展为 32bit 立即数赋值)

- O 选项不存在时, 对 Imm16 进行有符号扩展, 写入 Td 寄存器第 k 个字. 即: 将 Imm16 写入 Td 寄存器第 k 个字的低 16 比特, 高 16bit 补 Imm16 的符号位.
- O 选项存在时, 16bit 立即数写入 Td 寄存器第 k 个字的 bit5~20, 高 11bit 和低 5bit 补零.

Execution



Latency 2

Example BIU0: 0x10 (H) -> T1[13](Mode0);
BIU1: 0x800 (E) (O) -> T3[0] (Mode0);

2.3.6.21. Move 指令

Syntax	SRC & DST	1. BIUX: Ts0[i](B)(L ML MH H) -> Td[k](L ML MH H) 2. BIUX: Ts0[i](S)(L MH) -> Td[k](L MH) 3. BIUX: Ts0[i] -> Td[k] 4. BIUX: Ts0 -> Td 5. BIUX: Ts0[i] (B) (L ML MH H) -> IMA.Mx
	COND	({!}mode(0 1))

Encoding

1. BIUX: Ts0[i](B)(L|ML|MH|H) -> Td[k](L|ML|MH|H)

27	26	25	24	23	22	21	20	19	18	17	16	15	14	
COND	01	01		Td			k					00		
13	12	11	10	9	8	7	6	5	4	3	2	1	0	
00		Ts0		i			0		LMH(Td)		LMH(Ts0)			

2. BIUX: Ts0[i](S)(L|MH) -> Td[k](L|MH)

27	26	25	24	23	22	21	20	19	18	17	16	15	14	
COND	01	01		Td			k					00		
13	12	11	10	9	8	7	6	5	4	3	2	1	0	
01		Ts0		i			0		LMH(Td)		LMH(Ts0)			

3. BIUX: Ts0[i] -> Td[k]

27	26	25	24	23	22	21	20	19	18	17	16	15	14	
COND	01	01		Td			k					00		
13	12	11	10	9	8	7	6	5	4	3	2	1	0	
10		Ts0		i			0		0		0		0	

4. BIUX: Ts0 -> Td

27	26	25	24	23	22	21	20	19	18	17	16	15	14	
COND	01	01		Td			0					00		
13	12	11	10	9	8	7	6	5	4	3	2	1	0	
11		Ts0		0			0		0		0		0	

5. BIUX: Ts0[i] (B) (L|ML|MH|H) -> IMA.Mx

27	26	25	24	23	22	21	20	19	18	17	16	15	14	
COND	01	01		Mx			0		0			01		
13	12	11	10	9	8	7	6	5	4	3	2	1	0	
00		Ts0		i			0		0		0		LMH	

Encode field:

Field	Value
COND	参见 2.3 章

LMH	00:L 01:ML 10:MH 11:H
-----	--------------------------------

Description 该指令将寄存器 Ts0 中的数据传送到另一个寄存器.

- 当 B 选项存在时, 传送的数据粒度为 Byte; 当 S 选项存在时, 传送的数据粒度为 Short; 否则传送的数据粒度是 Word. i 指定传送 Ts0 的第几个 Word, k 指定传送到 Td 的第几个 Word. 未指定 i 和 k, 则传输整个 512bit 数据.
- L|ML|MH|H 选项指定待传送数据的位置和数据存入的位置. 对于不同粒度的数据, 选项含义如下:
 - Byte 粒度: L|ML|MH|H 分别表示指定 Word 的第 0, 1, 2, 3 个 Byte.
 - Short 粒度: L|MH 分别表示指定 Word 的第 0, 1 个 Short.
- 对于目标域为 IMA.Mx: 每个 BIU 有 4 组 IMA 第二目标域寄存器, 每个寄存器为 5bit, 高 2bit 表示第二目标域为 IMAx, 低 3bit 表示 IMA 的哪个 T 寄存器. 该指令将指定字节的低 5bit 传送给本单元的第 x 个第二目标域寄存器. 第二目标域寄存器的默认值为 0x6, 即表示不使用第二目标域.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1
						1.Read regs 2.Write 512 bits	Write 8/16/32 bits

Latency 写 512bit 数据: 1
写其他类型数据(8/16/32 bit): 2

Example BIU3: T1[3](B)(ML) -> T3[14](L)(Mode0);

2.3.6.22. Bit Reverse 指令

Syntax	BIUx: BitReverse (Ts0[i]) (L ML MH H) {(B S)}
	->
DST	Td[k] (L ML MH H)
COND	({!}mode(0 1))

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND		01		01		Td			k			10	
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Type		Ts0		i			0		LMH(Td)		LMH(Ts0)		

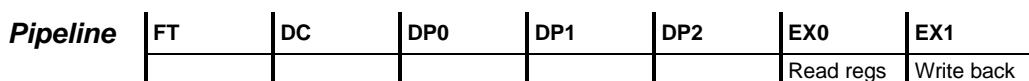
Encode field:

Field	Value
COND	参见 2.3 章
Type	00:B 01:S 10:默认
LMH	00:L 01:ML 10:MH 11:H

Description 该指令将寄存器 Ts0 的第 i 个 Word 指定位置的数据反序, 然后传送到寄存器 Td 的第 k 个 Word 的指定位置.

- 当 B 选项存在时, 传送的数据粒度为 Byte; 当 S 选项存在时, 传送的数据粒度为 Short.
- L|ML|MH|H 选项指定待传送数据的位置和数据存入的位置. 对于不同粒度的数据, 选项含义如下:
 - Byte 粒度: L|ML|MH|H 分别表示指定 Word 的第 0, 1, 2, 3 个 Byte.
 - Short 粒度: 仅可使用 L|MH, 它们分别表示指定 Word 的第 0, 1 个 Short.
- 反序的规则为: 高低位对称交换, 即第 0 位与第 31 位交换, 第 1 位与第 30 位交换, 如此依次交换到第 15 位与第 16 位.

Execution



Latency 2

Intrinsics	Intrinsic	Generated Instruction
	int __ucpm2_biuBR(int s0, const int lmh_s, const int lmh_d, const int type, const int mode, const int fu, const int pb);	BIUx: BitReverse (Ts0[i]) (L ML MH H) {(B S)} -> Td[k] (L ML MH H)

Note:

参数 lmh_s, lmh_d 应为: f_L, f_ML, f_MH, f_H;
参数 type 应为: f_B, f_S, 或 0.

Example BIU3: BitReverse (T3[0]) (MH) (S) -> T1[6] (L)(Mode0);

2.3.6.23. 篩选指令

Syntax	SRC	BIUx: Filter (Ts0[i])
		->
	DST	Td0[k] (L ML MH H) & SHUx.Td1
	COND	({!}mode(0 1))

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND		01		11		Td0			k			00	
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LMH		Ts0		i			SHUx	0			Td1		

Encode field:

Field	Value
COND	参见 2.3 章
LMH	00:L 01:ML 10:MH 11:H

Description 该指令将寄存器 Ts0 中的第 i 个字和第 i+1 个字拼接成 64bit, 进行数据筛选, 存入本单元的 Td0 和 SHUx 的寄存器 Td1.

- 统计 64bit 数中有多少个“1”，并将结果存入寄存器 Td0 的第 k 个 Word 的其中一个 Byte. 选项 L|ML|MH|H 分别表示 Word 的第 0, 1, 2, 3 个 Byte.
- 将 64bit 数中“1”的位置记录下来(从 0 开始计数, 存在一个 byte 中), “0”的位置丢弃, 之后将各个 byte 连续拼接, 剩余高位补 0x40, 得到 512bit 数存入 SHUx 的寄存器 Td1.

例:

输入数据为 0b11010011, 则会将“1”的总个数“5”写入 Td0[k]中, 将 0x.....4040_4007_0604_0100 写入 SHUx.Td1 中.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3
						Read regs	Write to BIU		Write to SHU

Latency -> BIU: 2
-> SHU: 4

Intrinsics	Intrinsic	Generated Instruction
-------------------	-----------	-----------------------

ucp_vec512_t __ucpm2_biuFilterV(int s0, const int mode, const int fu, const int pb)	BIUx: Filter (Ts0[i]) -> Td0[k] (L ML MH H) & SHUx.Td1
int __ucpm2_biuFilterS(int s0, const int lmh, const int mode, const int fu, const int pb)	

Macro	Corresponding Intrinsic
ucpm2_biuFilter(dst1, s0, lmh, mode, fu, pb)	__ucpm2_biuFilterV(s0, mode, fu, pb); dst1 = __ucpm2_biuFilterS(s0, lmh, mode, fu, pb)

Example BIU2: Filter(T2[15]) -> T0[15](MH) & SHU3.T1(Mode0);

2.3.6.24. 扩展指令

Syntax	SRC	BIUx: Expd (Ts0[i])
	->	
	DST	Td0[k] (L ML MH H) & SHUx.Td1
	COND	({!}mode(0 1))

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND		01		11		Td0		k			01		
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LMH		Ts0		i			SHUx	0			Td1		

Encode field:

Field	Value
COND	参见 2.3 章
LMH	00:L 01:ML 10:MH 11:H

Description 该指令将寄存器 Ts0 中的第 i 个字和第 i+1 个字拼接成 64bit, 进行数据筛选, 存入本单元的 Td0 和 SHUx 的寄存器 Td1.

- 统计 64bit 数中有多少个“1”，并将结果存入寄存器 Td0 的第 k 个 Word 的其中一个 Byte. 选项 L|ML|MH|H 分别表示 Word 的第 0, 1, 2, 3 个 Byte.
- 从低到高判断 64bit 数的每一 bit 是否为“1”，如果为“1”，则记录下是第几个“1”（从 0 开始计数，存在一个 byte 中），如果为“0”，则记录为 0x40，从而依序得到 512bit 结果存入 SHUx 的寄存器 Td1.

例:

输入数据为 0b11010011, 则会将 “1”的总个数 “5” 写入 Td0[k]中, 将 0x.....0403_4002_4040_0100 写入 SHUx.Td1 中.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3
						Read regs	Write to BIU		Write to SHU

Latency -> BIU: 2
-> SHU: 4

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_biuExpdV(int s0, const int mode, const int fu, const int pb)	BIUx: Expd (Ts0[i]) -> Td0[k] (L ML MH H) & SHUx.Td1
	int __ucpm2_biuExpdS(int s0, const int lmh, const int mode, const int fu, const int pb)	

Macro	Corresponding Intrinsic
ucpm2_biuExpd(dst1, s0, lmh, mode, fu, pb)	__ucpm2_biuExpdV(s0, mode, fu, pb); dst1 = __ucpm2_biuExpdS(s0, lmh, mode, fu, pb);

Example BIU2: Expd (T2[15]) -> T0[15](L) & SHU3.T1(Mode0);

2.3.6.25. 有效位生成指令

Syntax	SRC	BIUx: Valid (Ts0)
	->	
DST	1. M[t {S++},{I++},{A++}] [k] {(Wx)} 2. M[dis{(shift)}] [k] {(Wx)} (LatchID) 3. SHUn.Td[k] 4. SHU[{0},{1},{2},{3}].Td[k] 5. IMAx.Td 6. BIUx.Td	
COND	({!}mode(0 1))	

27	26	25	24	23	22	21	20	19	18	17	16	15	14
COND	01		11		00			k			10		
13	12	11	10	9	8	7	6	5	4	3	2	1	0
10	Ts0							DST					

Encode field:

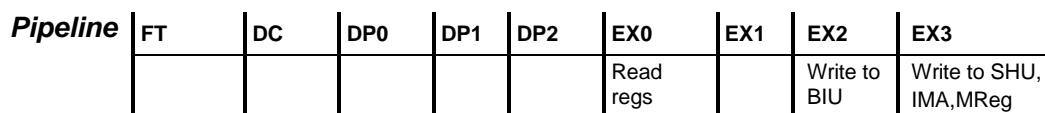
Field	Value
COND	参见 2.3 章

Description 该指令为筛选指令的逆指令, 产生 64bit 结果. 对于寄存器 Ts0 的 64 个 Byte, 判断每个 Byte 的 bit6 是否为 0, 如果为 0, 则使用其 bit0~5 作为索引 Index, 将 64bit 结果的第 Index bit 置为 1, 否则为 0.

关于目标域:

- 当目标域为 MReg, SHU 时, 将 64bit 结果写入对应寄存器的第 k 和 k+1 个 Word, k 只能为偶数.
- 当目标域为 IMA, BIU 时, 会将 64bit 结果扩展 8 倍得到 512bit, 然后发往目标域.

Execution



- Latency**
- > BIU: 3
 - > MReg: 3
 - > SHU: 4
 - > IMA: 4

Intrinsics	Intrinsic	Generated Instruction
-------------------	-----------	-----------------------

ucp_vec512_t __ucpm2_biuValidV(ucp_vec512_t s0, const int mode, const int fu, const int pb)	BIUx: Valid (Ts0) -> 向量目标域
int __ucpm2_biuValidS(ucp_vec512_t s0, const int mode, const int fu, const int pb)	BIUx: Valid (Ts0) -> 标量目标域

Example BIU0: Valid (T3) -> M[21][12](Mode0);

2.3.6.26. load 周期配置指令

Syntax	SRC	BIUx: ConfigLoadCycle (Enable Disable)
	COND	({!}mode(0 1))

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
			COND		01		01		00		0000		11	
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	Enable/Disable									0			

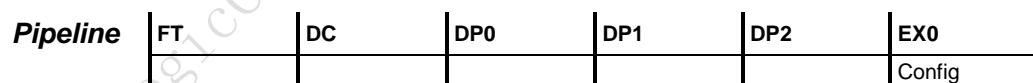
Encode field:

Field	Value
Enable	Enable: 1 Disable: 0
COND	参见 2.3 章

Description 通过该指令可以配置 load 指令发送给目标域的周期数, 但不改变 load 指令从 DM 取数的周期数.

- Disable 表示 load 周期数为默认, Enable 表示 load 周期数比默认少一拍. 初始配置为 Disable.
- 该指令的执行周期为 1 拍. 当 Load 指令目标域为非 MReg 时, 执行完成即生效; 当 Load 指令目标域为 MReg 时, 该指令执行后还需额外增加三条 Nop 指令才能生效.
- 每个 BIU 可分别配置 load 的周期数.

Execution



Latency 1

Example BIU0: ConfigLoadCycle (Enable)(Mode0);

2.3.6.27. 向量寄存器复位指令

Syntax	SRC	BIUx: Reset (Ts0)
	COND	({!}mode(0 1))

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND	00	01		Ts0		0010		0					
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														

Encode field:

Field	Value
COND	参见 2.3 章

Description 该指令将寄存器 Ts0 复位. 寄存器的复位值见 2.3.6.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_reset(ucp_vec512_t s0, const int mode, const int fu, const int pb)	BIUx: Reset (Ts0)

Example BIU1: Reset (T2)(Mode0);

2.3.6.28. Wait 立即数指令

Syntax	BIUx: Wait Imm6																																																								
Encoding	<table border="1"><tr><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td></tr><tr><td>01</td><td></td><td>00</td><td></td><td>01</td><td></td><td>0</td><td></td><td>0100</td><td></td><td>0</td><td></td><td></td><td></td></tr><tr><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Imm6</td></tr></table>	27	26	25	24	23	22	21	20	19	18	17	16	15	14	01		00		01		0		0100		0				13	12	11	10	9	8	7	6	5	4	3	2	1	0							0							Imm6
27	26	25	24	23	22	21	20	19	18	17	16	15	14																																												
01		00		01		0		0100		0																																															
13	12	11	10	9	8	7	6	5	4	3	2	1	0																																												
						0							Imm6																																												

Description 该指令将该 BIU 通道的指令延迟 Imm6 拍之后执行, 最高延迟 62 拍. 具体行为参见 2.3.2.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_biuWaitImm(const int imm6, const int fu, const int pb)	BIUx: Wait Imm6

Example BIU1: Wait 4;

2.3.6.29. Wait KI 指令

Syntax	SRC	BIUx: WaitKI
Encoding	27 26 25 24 23 22 21 20 19 18 17 16 15 14 01 00 01 0 0100 0 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0	

Description 该指令将使 BIUx 的指令延迟执行, 最高延迟 62 拍. 具体行为参见 2.3.2. Wait 的周期数由 KIy[m:n]指定, 其中 y 表示使用哪个 KI, m:n 表示使用该 KI 中的哪些 bit.

Note: 每个 KI 都是 24bit 寄存器, WaitKI 指令使用特定的 6bit 作为 wait 的周期数.

y, m, n 与发射槽的对应关系为:

发射槽	KIy	[m:n]
BIU0	KI8	[5:0]
BIU1	KI8	[11:6]
BIU2	KI8	[17:12]
BIU3	KI8	[23:18]
SHU0	KI9	[5:0]
SHU1	KI9	[11:6]
SHU2	KI9	[17:12]
SHU3	KI9	[23:18]
IMA0	KI10	[5:0]
IMA1	KI10	[11:6]
IMA2	KI10	[17:12]
IMA3	KI10	[23:18]
R0	KI11	[5:0]
R1	KI11	[11:6]
R2	KI11	[17:12]
R3	KI11	[23:18]
R4	KI12	[5:0]
R5	KI12	[11:6]
R6	KI12	[17:12]
R7	KI12	[23:18]

Execution

Pipeline	FT	DC	DP0
			Finish

Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_biuWaitKI(const int fu, const int pb)	BIUx: WaitKI

Example BIU0: WaitKI;

2.3.6.30. NOP 指令

Syntax	SRC	NOP;
Encoding	27 26 25 24 23 22 21 20 19 18 17 16 15 14 COND 00 01 0 1000 0 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0	

Description 空指令.

Execution



Latency 1

Example BIU0: NOP;

2.3.6.31. Set Condition 指令

Syntax	SRC	BIUx: Imm9
		->
	DST	mode0

Note: 本指令无条件执行

Encoding	27	26	25	24	23	22	21	20	19	18	17	16	15	14
	COND	00		01		0		1100		0				
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0								Imm9

Description 该指令用 9 位立即数对 BIU 单元的 mode0 寄存器进行设置, 即配置指令执行的条件. mode0 寄存器的值对应的条件表达式见 2.3 的条件表达式值编码表.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_biuSetCond(const int imm9, const int fu, const int pb)	BIUx: Imm9 -> mode0

Example BIU2: 172 -> mode0;

2.3.7. IMA

IMA 公共目标域及编码, 根据 DST1 的不同有两种格式

1. DST1 为 M[t] 时, 其编码如下:

No	DST1 & DST2 assembly	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IMAn: * -> M[t] (Wx) {& IMAn.Td}	0		Td		Wx				t				
2	IMAn: * -> M[t] (Wx) & SHUx.Td 仅 FFT 乘法指令	0	SHU	Td	x	Wx				t				

各 field 编码如下:

Field	Value	Encode
Td in item 1	T0 - T5	0 - 5
	无 DST2	7
SHUx in item 2	n = 0, x = 0 2	0: n == x
	n = 1, x = 1 3	1: n != x
	n = 2, x = 2 0	
	n = 3, x = 3 1	
Wx	Wh	1
	WI	0
t	0 - 255	0 - 255

2. DST1 不为 M[t] 时, DST1 编码如下:

No	DST1 assembly	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IMAn: * -> M[{{S++},{I++},{A++}}] (Wx)	1		DST2		0	0		Wx	0	A++	S++	I++	
2	IMAn: * -> M[dis({shift})] (Wx) (LatchID)	1		DST2		0	0		Wx	1	shift	LatchID		
3	IMAn: * -> IMA[{0},{1},{2},{3}].Td	1		DST2		1	IMA3	IMA2	IMA1	IMA0			Td	
4	IMAn: * -> SHUx.Td	1		DST2		0	1	0	SHUx	0			Td	
5	IMAn: * -> BIUx.Td	1		DST2		0	1	0	BIUx	1			Td	
6	IMAn: * -> IMAn.MR	1		DST2		0	1	1	0	0	0	0	1	
7	No DST1	1		DST2		0	1	1	0	0	0	0	0	

对于目标为 Dis 的情况, 目标具有可选的 LatchID:

IMA0/IMA2 可选的 LatchID 为: Latch0/Latch2/Latch4/Latch6, 其编码如下:

LatchID	Encoding
Latch0	0
Latch2	1
Latch4	2
Latch6	3

IMA1/IMA3 可选的 LatchID 为: Latch1/Latch3/Latch5/Latch7, 其编码如下:

LatchID	Encoding
Latch1	0
Latch3	1
Latch5	2
Latch7	3

DST2 编码如下:

No	DST2 assembly	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IMAn: * -> DST1 & SHUx.Td	1	1	SHUx	Td									DST1
2	IMAn: * -> DST1 & IMAn.Td	1	0		Td									DST1
3	IMAn: * -> DST1 & IMAn.MR	1	0	1	1	0								DST1
4	No DST2	1	0	1	1	1								DST1

各 field 编码如下:

Field	Value	Encode
Wx in DST1 item 1 & 2	n = 0 2, Wx = W0 2 4 6	W0: 00, W2: 01, W4: 10, W6: 11
	n = 1 3, Wx = W1 3 4 7	W1: 00, W3: 01, W5: 10, W7: 11
SHUx in DST2 item 1	n = 0, x = 0 2	SHU0, SHU1: 0
	n = 1, x = 1 3	SHU2, SHU3: 1
	n = 2, x = 0 2	
	n = 3, x = 1 3	

当两目标域为同一类功能单元时, 其 T 寄存器必须相同. 如当两目标域均为 IMA 单元时, 记 DST1 为 IMAx.Td0, DST2 为 IMAn.Td1, 则 Td0 必须等于 Td1; 当两目标域均为 SHU 单元时, 记 DST1 为 SHUx.Td0, DST2 为 SHUn.Td1, 则 Td0 必须等于 Td1.

所有指令发往 IMA 的 T6 寄存器的结果仅能维持 1 拍.

2.3.7.1. 乘法指令 1: Ts0/MR +/- (Ts1,Ts2)*V(Ts3)

Syntax	SRC	IMAx: Ts0 MR + - (Ts1,Ts2)*V(Ts3)(Shiftmodes)(W S B)(USU USS SSS) {(T)} {(P)} {(L)}
		->
	DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{(shift)}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td 8. IMAX.MR
	DST2	1. & IMAX.Td 2. & SHUn.Td 3. & IMAX.MR
	COND	({!}mode(0 1))

Note:

- Shiftmodes: Shiftmode0/ Shiftmode1/ Shiftmode2/ Shiftmode3
- LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	-/+	WSB	US	T	P	L	0	0	SM		Ts1		Ts3					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0 MR	Ts2																	DST1 & DST2

Encode field:

Field	Value
COND	参见 2.3 章
-/+	1: - 0: +
WSB	00: B 01: S 11: W
US	2: USU 1: USS 0: SSS
SM	0: Shiftmode0 1: Shiftmode1 2: Shiftmode2 3: Shiftmode3
Ts0 MR	0b0-0b101 对应 T0-T5, 0b110 表示 MR, 0b111 表示 0
Ts1~Ts3	0b0-0b101 对应 T0-T5, 0b110 表示 T6(Ttmp), 0b111 表示 0
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令为乘加或乘减指令, 结果发往目标域寄存器. 当使用了 MR 作为源操作数或者目标域指定为 MR 时, 计算产生的中间结果还将存入 MR.

关于选项:

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte. USU/USS/SSS 选项为必选项, 其中 S 表示有符号数, U 表示无符号数, 第一个 S 或者 U 标示 Ts2, Ts1 的符号情况, 第二个 S 或者 U 标示 Ts3, 第三个 S 或者 U 标示 Ts0 和输出结果. 不同粒度的数据相乘将产生不同粒度的中间结果. Word 产生 74bit 粒度的中间结果, Short 产生 42bit 中间结果, Byte 产生 26bit 中间结果. 中间结果可以使用 MR 进行存储.
- Shiftmodes 中也存在选择运算方式的选项, 具体含义和配置方式可见 2.3.7.6. 计算完成的结果还需要根据 ShiftNum(Shiftmode 的 bit0~7) 进行移位, 得到移位后的结果.
- 计算的最终结果由 T, P, L 选项决定. T 选项存在表示对结果进行截断处理, 不存在表示饱和处理. P 选项表示是小数模式, 无 P 选项则认为是整数模式.
- L 选项存在, 表示对结果进行位宽扩展输出. 对于 Word, 截取移位后结果的低 64bit 输出; 对于 Short, 截取结果的低 32bit 输出; 对于 Byte, 截取结果的低 16bit 输出. 截取的结果将分两拍写入目标域的两个连续的寄存器, 先发高位部分, 后发低位部分. 当 L 选项不存在, 截取对应粒度的数据输出到目标域寄存器. 对于 Word, 截取移位后结果的 bit31~62; 对于 Short, 截取结果的 bit15~30; 对于 Byte, 截取结果的 bit7~14.

Note:

- L 选项存在时, MR 结果与低位结果同拍输出. 因此如果使用了 L 选项, 后面还想使用与乘法相关的指令, 则两条指令间至少应该插一条 NOP 指令.
- L 选项存在时, Shiftmode 不允许配置为 0 或左移.
- L 选项存在时, 可认为结果是右移|ShiftNum|-1.
- 结果如果只更新 MR, 则不允许使用 L 选项.

关于运算方式:

Shiftmode 中的选项与指令本身共同决定选择的运算方式, 下表列出了 Shiftmode 的选项配置与运算方式的对应关系:

bit 18	bit 17	bit 16	bit 14: 15	bit 11: 13	bit 10	bit 9	bit 8	运算方式
-----------	-----------	-----------	------------------	------------------	-----------	----------	----------	------

Seco nd	Sec ond	Is KInd	Mind	Ki nd	Co m	Co nj	Nu m	
IsSu b	IsT	ex	ex	ex	pl	ex		
0	1	0	/	0	0	0	0	Ts0 MR +/- (Ts1, Ts2)[MIndex]* Ts3
0	1	1	/	/	0	0	0	Ts0 MR +/- (Ts1, Ts2)[MIndex]* V(Ts3[KIndex])
0	1	0	/	0	1	0	0	Ts0 MR +/- (Ts1, Ts2)[MIndex]* Ts3(C)
0	1	1	/	/	1	0	0	Ts0 MR +/- (Ts1, Ts2)[MIndex]* V(Ts3[KIndex])(C)
0	1	0	/	0	1	1	0	Ts0 MR +/- Conj((Ts1, Ts2)[MIndex])* Ts3(C)
0	1	1	/	/	1	1	0	Ts0 MR +/- Conj((Ts1, Ts2)[MIndex])* V(Ts3[KIndex])(C)
/	1	1	/	/	0	0	1	Ts0 MR +/- (Ts1, Ts2)[MIndex]* V(Ts3[KIndex]) +/- (Ts1, Ts2)MPLUS * VPLUS
/	0	1	/	/	0	0	1	Ts0 MR +/- (Ts1, Ts2)[MIndex]* V(Ts3[KIndex]) +/- Ts1 * VPLUS

上表中 / 表示该位置按需要进行配置.

- Num 为 0, 表示运算次数为 1 次; Num 为 1, 表示运算次数为 2 次. 乘累加 2 次时, IsKIndex 必须为 1, 此时 SecondIsT 和 SecondIsSub 生效. SecondIsT 为 0 选择第二次累加的源操作数是 Ts1, 否则是(Ts1, Ts2) MPLUS. SecondIsSub 为 1, 则第一次做加法, 第二次做减法, 或者正好相反. SecondIsSub 为 0, 则两次运算均为加或均为减.
- Complex 为 1 时, 进行复数计算, 此时只允许进行一次乘累加. 当数据粒度是 Byte 时, 每 32bit 中低 2byte 为实部, 高 2byte 为虚部. 当数据粒度是 Short 时, 每 32bit 中低 16 位为实部, 高 16 位为虚部. Conj 存在表示取共轭, 对(Ts2, Ts1)的结果取共轭.
- (Ts1, Ts2)[MIndex]和 MPLUS 表示 Ts2, Ts1 寄存器中的每 128bit 数据拼接成 256bit 数据 (Ts1 在低位, Ts2 在高位), 从中抽取 128bit, 再进行乘法运算. 对于 Byte, (Ts1, Ts2)[MIndex]从第 MIndex*4 个 byte 开始抽取 128bit 数据, MPLUS 从第 MIndex*4+1 个 byte 开始抽取 128bit 数据; 对于 Short, (Ts1, Ts2)[MIndex]从第 MIndex*2 个 short 开始抽取 128bit 数据, MPLUS 从第 MIndex*2+1 个 short 开始抽取; 对于 Word, (Ts1, Ts2)[MIndex]从第 MIndex 个 word 开始抽取 128bit 数据, 不支持 MPLUS 运算.
- V(Ts3[KIndex])和 VPLUS 表示对 Ts3 的指定数据进行向量扩展后进行乘法运算. 当指令为实数运算时: 对于 Byte, V(Ts3[KIndex])从 Ts3 的

每 128bit 中选取第 KIndex*2 个 byte, 进行 16 倍扩展; VPLUS 从 Ts3 的每 128bit 中选取第 KIndex*2+1 个 byte, 进行 16 倍扩展. 对于 Short, V(Ts3[KIndex]) 从 Ts3 的每 128bit 中选取第 KIndex 个 short, 进行 8 倍扩展; VPLUS 从 Ts3 的每 128bit 中选取第 KIndex+1 个 short, 进行 8 倍扩展. 对于 Word, V(Ts3[KIndex]) 从 Ts3 的每 128bit 中选取第 KIndex/2 个 word, 进行 4 倍扩展; VPLUS 从 Ts3 的每 128bit 中选取第 KIndex/2+1 个 word, 进行 4 倍扩展. 当指令为复数运算时: 对于 Byte, 从 Ts3 的每 128bit 中选取第 KIndex*2 个 byte 作为实部数据, 第 KIndex*2+1 个 byte 作为虚部数据, 进行 8 倍扩展. 对于 Short, 从 Ts3 的每 128bit 中选取第 KIndex 个 short 作为实部数据, 第 KIndex+1 个 short 作为虚部数据, 进行 4 倍扩展.

Note:

- 复数运算, Num 为 1 的两次累加运算均不支持 Word 粒度.
- Shiftmode 的参数中, 每个 Word 的低 8bit 表示的 ShiftNum 都可以不同, 但每 128bit 内各 Word 的其他参数必须一致.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4
					Read ShiftMode	Read regs	Read MCW, ShiftMode	Write to IMA.MR	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 5
 - > MReg: 3
 - > SHU: 4
 - > IMA: 4
 - > MR: 3

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_mulAddV(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 + (Ts1,Ts2)*V(Ts3) (Shiftmodes)(W S B)(USU USS S SS) {(T)} {(P)} -> Dest
	ucp_vec1024_t __ucpm2_mulAddV_L(ucp_vec5 12_t s0, ucp_vec512_t s1,	IMAx: Ts0 + (Ts1,Ts2)*V(Ts3) (Shiftmodes)(W S B)(USU USS S SS) {(T)} {(P)} (L) -> Dest

ucp_vec512_t s2, ucp_vec512_t s3, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const int sss, const int mode, const int fu, const int pb)	
ucp_vec512_t __ucpm2_mulSubV(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 - (Ts1,Ts2)*V(Ts3) (Shiftmodes)(W S B)(USU USS SS) {(T)} {(P)} -> Dest
ucp_vec1024_t __ucpm2_mulSubV_L(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 - (Ts1,Ts2)*V(Ts3) (Shiftmodes)(W S B)(USU USS SS) {(T)} {(P)} (L) -> Dest

Example IMA1: T2 + (T3,T4) * V(T5)(Shiftmode3)(S)(SSS)(T)(L) -> M[0]
(W7)(Mode0);

2.3.7.2. 乘法指令 2: Ts0/MR +/- Ts1*Ts2

Syntax Src1: 实数

SRC	IMAx: Ts0 MR + - Ts1*Ts2(Shiftmodes) (W S B)(USU USS SSS){(T)} {(P)} {(L)} {(Flag)}
	->
DST1	<ol style="list-style-type: none"> 1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{(shift)}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td 8. IMAX.MR
DST2	<ol style="list-style-type: none"> 1. & IMAX.Td 2. & SHUn.Td 3. & IMAX.MR
COND	({!}mode(0 1))

Src2: Byte/Short 类型, 复数

SRC	IMAx: Ts0 MR + - Ts1*Ts2(Shiftmodes) (C)(S B)(USU USS SSS){(T)} {(P)} {(L)} {(Flag)}
	->
DST1	<ol style="list-style-type: none"> 1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{(shift)}] (Wx) (LatchID) 4. IMA[{0}, {1}, {2}, {3}].Td 5. SHUn.Td 6. BIUn.Td 7. IMAX.MR
DST2	<ol style="list-style-type: none"> 1. & IMAX.Td 2. & SHUn.Td 3. & IMAX.MR
COND	({!}mode(0 1))

Src3: Word 类型, 复数第一阶段

SRC	IMAx: Ts0 MR + - Ts1*Ts2(Shiftmodes)(C)(WF)(USU USS SSS) {(P)}
	->
DST	IMAX.MR
COND	({!}mode(0 1))

Src4:Word 类型, 复数第二阶段

SRC	IMAx: MR + - Ts1*Ts2(Shiftmodes) (C)(WS)(USU USS SSS) {(T)} {(P)} {(L)} {(Flag)}
	->
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMA[{0}, {1}, {2}, {3}].Td 5. SHUn.Td 6. BIUn.Td IMAX.MR
DST2	1. & IMAX.Td 2. & SHUn.Td 3. & IMAX.MR
COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

SRC1: 实数

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	0	-/+	WSB	US	T	P	L	Flag	0	SM		Ts1		Ts2					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0 MR		Ts1		DST1 & DST2															

SRC2: Byte/Short 类型, 复数

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	0	-/+	S/B(2bit)	US	T	P	L	Flag	1	SM		Ts1		Ts2					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0 MR		Ts1		DST1 & DST2															

SRC3: Word 类型, 复数第一阶段

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	0	-/+	10	US	0	P	0	0	1	SM		Ts1		Ts2					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0 MR		Ts1		No (DST1 & DST2)															

SRC4: Word 类型, 复数第二阶段

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	0	-/+	11	US	T	P	L	Flag	1	SM		Ts1		Ts2					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
110		Ts1		DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
-/+	1: -

	0: +
WSB, S/B(2bit)	0: B 1: S 3: W
US	2: USU 1: USS 0: SSS
SM	0: Shiftmode0 1: Shiftmode1 2: Shiftmode2 3: Shiftmode3
Ts0 MR	0b0-0b101 对应 T0-T5, 0b110 表示 MR, 0b111 表示 0
Ts1, Ts2	0b0-0b101 对应 T0-T5, 0b110 表示 T6(Ttmp), 0b111 表示 0
{No} (DST1 & DST2)	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令为乘加或乘减指令, 结果发往目标域寄存器. 当使用了 MR 作为源操作数或者目标域指定为 MR 时, 计算产生的中间结果还将存入 MR.

关于选项:

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte. USU/USS/SSS 选项为必选项, 其中 S 表示有符号数, U 表示无符号数, 第一个 S 或者 U 标示 Ts1 的符号情况, 第二个 S 或者 U 标示 Ts2, 第三个 S 或者 U 标示 Ts0 和输出结果. 不同粒度的数据相乘将产生不同粒度的中间结果. Word 产生 74bit 粒度的中间结果, Short 产生 42bit 中间结果, Byte 产生 26bit 中间结果. 中间结果可以使用 MR 进行存储.
- C 选项存在, 表示进行复数计算. C 选项不存在, 进行实数运算.
- Shiftmodes 中也存在选择运算方式的选项, 具体含义和配置方式可见 2.3.7.6. 计算完成的结果还需要根据 ShiftNum(Shiftmode 的 bit0~7)进行移位, 得到移位后的结果.
- 计算的最终结果由 T, P, L 选项决定. T 选项存在表示对结果进行截断处理, 不存在表示饱和处理. P 选项表示是小数模式, 无 P 选项则认为是整数模式.
- L 选项存在, 表示对结果进行位宽扩展输出. 对于 Word, 截取移位后结果的低 64bit 输出; 对于 Short, 截取结果的低 32bit 输出; 对于 Byte, 截取结果的低 16bit 输出. 截取的结果将分两拍写入目标域的两个连续的寄存器, 先发高位部分, 后发低位部分. 当 L 选项不存在, 截取对应粒度的数据输出到目标域寄存器. 对于 Word, 截取移位后结果的 bit31~62; 对于 Short, 截取结果的 bit15~30; 对于 Byte, 截取结果的 bit7~14.
- Flag 表示根据运算结果更新乘法部分的溢出标志寄存器.

Note:

- L 选项存在时, MR 结果与低位结果同拍输出. 因此如果使用了 L 选项, 后面还想使用与乘法相关的指令, 则两条指令间至少应该插一条 NOP 指令.
- L 选项存在时, Shiftmode 不允许配置为 0 或左移.
- L 选项存在时, 可认为结果是右移 $|ShiftNum|-1$.
- 结果如果只会更新 MR, 则不允许使用 L 选项.

对于 Src1 和 Src2, 运算方式:

Shiftmode 中的选项与指令本身共同决定选择的运算方式, 下表列出了 Shiftmode 的选项配置与运算方式的对应关系:

bit1 8	bit17	bit16	bit 14 : 15	bit 11: 13	bit 10	bit 9	bit 8	运算方式
Sec ond IsSu b	Sec ond IsT	Is KInd ex	Mi nd ex	Ki nd ex	Co m pl ex	Co n j	Nu m	
0	0	0	0	0	0	0	0	Ts0 MR +/- Ts1 * Ts2
0	0	1	0	/	0	0	0	Ts0 MR +/- Ts1 * V(Ts2[KIndex])
0	0	0	0	0	1	0	0	Ts0 MR +/- Ts1 * Ts2(C)
0	0	1	0	/	1	0	0	Ts0 MR +/- Ts1 * V(Ts2[KIndex])(C)
0	0	0	0	0	1	1	0	Ts0 MR +/- Conj(Ts1) * Ts2(C)
0	0	1	0	/	1	1	0	Ts0 MR +/- Conj(Ts1) * V(Ts2[KIndex])(C)

上表中 / 表示该位置按需要进行配置.

- 该指令不支持 Num 为 1.
- 指令中存在 C 选项或者 Shiftmode 的 Complex 为 1 时, 均进行复数计算. 当数据粒度是 Byte 时, 每 32bit 中低 2byte 为实部, 高 2byte 为虚部. 当数据粒度是 Short 时, 每 32bit 中低 16 位为实部, 高 16 位为虚部. Conj 存在表示对 Ts1 取共轭.
- V(Ts2[KIndex]) 表示对 Ts2 的指定数据进行向量扩展后进行乘法运算. 当指令为实数运算时: 对于 Byte, 从 Ts2 的每 128bit 中选取第 KIndex*2 个 byte, 进行 16 倍扩展. 对于 Short, 从 Ts2 的每 128bit 中选取第 KIndex 个 short, 进行 8 倍扩展. 对于 Word, 从 Ts2 的每 128bit 中选取第 KIndex/2 个 word, 进行 4 倍扩展. 当指令为复数运算时: 不支持 Byte 粒度. 对于 Short, 从 Ts2 的每 128bit 中选取第 KIndex 个 short 作为实部数据, 第 KIndex+1 个 short 作为虚部数据, 进行 4 倍扩展.

对于 Src3 和 Src4:

这两条指令连续使用可完成 Word 粒度的复数运算. Src3 进行预处理, 中间结果写入 MR. Src4 将运算结果写入目标域. Shiftmode 的配置与 Src1 和 Src2 类似, V(Ts2[KIndex]) 表示从 Ts2 的每 128bit 中选取第 KIndex/2 个 word 作为实部数据, 选取第 KIndex/2+1 个 word 作为虚部数据, 进行乘法运算.

Note:

- Shiftmode 的参数中, 每个 Word 的低 8bit 表示的 ShiftNum 都可以不同, 但每 128bit 内各 Word 的其他参数必须一致.
- Byte 粒度下的复数运算不支持 KIndex 选项.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4
					Read ShiftMode	Read regs	Read MCW, ShiftMode	Write to IMA, MR	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 5

- > MReg: 3
- > SHU: 4
- > IMA: 4
- > MR: 3

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_mulRealAdd(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAX: Ts0 + Ts1*Ts2(Shiftmodes) (W S B)(USU USS SSS){(T)} {(P)} {(Flag)} -> Dest
	ucp_vec1024_t __ucpm2_mulRealAdd_L(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const	IMAX: Ts0 + Ts1*Ts2(Shiftmodes) (W S B)(USU USS SSS){(T)} {(P)} {(Flag)} (L) -> Dest

int f_Flag, const int sss, const int mode, const int fu, const int pb)	
ucp_vec512_t __ucpm2_mulRealSub(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 - Ts1*Ts2(Shiftmodes) (W S B)(USU USS SSS){(T)} {(P)} {(Flag)} -> Dest
ucp_vec1024_t __ucpm2_mulRealSub_L(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int type_BSW, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb);	IMAx: Ts0 - Ts1*Ts2(Shiftmodes) (W S B)(USU USS SSS){(T)} {(P)} {(Flag)} (L) -> Dest
ucp_vec512_t __ucpm2_mulAdd_ComplexBS(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int type_BS, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 + Ts1*Ts2(Shiftmodes) (C) (S B) (USU USS SSS) {(T)} {(P)} {(Flag)} -> Dest
ucp_vec1024_t __ucpm2_mulAdd_ComplexBS_L(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int type_BS, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 + Ts1*Ts2(Shiftmodes) (C) (S B) (USU USS SSS) {(T)} {(P)} {(Flag)} (L) -> Dest
ucp_vec512_t __ucpm2_mulSub_ComplexBS(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int type_BS, const int f_P, const int f_T, const int f_Flag, const int	IMAx: Ts0 - Ts1*Ts2(Shiftmodes) (C) (S B) (USU USS SSS) {(T)} {(P)} {(Flag)} -> Dest

sss, const int mode, const int fu, const int pb)	
ucp_vec1024_t __ucpm2_mulSub_ComplexBS_ L(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int type_BS, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 - Ts1*Ts2(Shiftmodes) (C) (S B) (USU USS SSS) {(T)} {(P)} {(Flag)} (L) -> Dest
ucp_vec512_t __ucpm2_mulAdd_ComplexWF(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int f_WF, const int f_P, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 + Ts1*Ts2(Shiftmodes) (C) (WF) (USU USS SSS) {(P)} -> IMAx.MR
ucp_vec512_t __ucpm2_mulSub_ComplexWF(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int f_WF, const int f_P, const int sss, const int mode, const int fu, const int pb)	IMAx: Ts0 - Ts1*Ts2 (Shiftmodes) (C) (WF) (USU USS SSS) {(P)} -> IMAx.MR
ucp_vec512_t __ucpm2_mulAdd_ComplexWS(/ * the first source is MR */ ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int f_WS, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAx: MR + Ts1*Ts2(Shiftmodes) (C)(WS)(USU USS SSS) {(T)} {(P)} {(Flag)} -> Dest
ucp_vec1024_t __ucpm2_mulAdd_ComplexWS_ L(* the first source is MR */ ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int f_WS, const int f_P, const int f_T, const int f_Flag,	IMAx: MR + Ts1*Ts2(Shiftmodes) (C)(WS)(USU USS SSS) {(T)} {(P)} {(Flag)} (L) -> Dest

const int sss, const int mode, const int fu, const int pb)	
ucp_vec512_t __ucpm2_mulSub_ComplexWS(/ * the first source is MR */ ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int f_WS, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAx: MR - Ts1*Ts2(Shiftmodes) (C)(WS)(USU USS SSS) {(T)} {(P)} {(Flag)} -> Dest
ucp_vec1024_t __ucpm2_mulSub_ComplexWS_ L/* the first source is MR */ ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int f_C, const int f_WS, const int f_P, const int f_T, const int f_Flag, const int sss, const int mode, const int fu, const int pb)	IMAx: MR - Ts1*Ts2(Shiftmodes) (C)(WS)(USU USS SSS) {(T)} {(P)} {(Flag)} (L) -> Dest

Example IMA1: 0 + T2*T3(Shiftmode1) (W)(USS)(T)(P)(Flag) ->
M[56](W3)(Mode0);

2.3.7.3. FFT 乘法指令: Ts0 AS Ts1*Ts2

Syntax	SRC	IMAx: Ts0 AS Ts1*Ts2 (Shiftmodes) (IndexMode0)
		->
	DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID)
	DST2	& SHUn.Td
	COND	({!}mode(0 1))

Note:

1. 2 个目标域为必选
2. LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Syntax	SRC	IMAx: Ts0 AS Ts1*Ts2 (Shiftmodes) (IndexMode2 IndexMode3)
		->
	DST1	IMAx.Td
	COND	({!}mode(0 1))

Note:

1. 仅能写当前 IMA 的 T 寄存器
2. 该指令仅支持 short 类型的计算

Encoding	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
	COND	0	0	0	1	11	0	1	0	IM	1	SM		Ts1		Ts2				
	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Ts0		Ts1										DST1 & DST2						

Encode field:

Field	Value
COND	参见 2.3 章
IM	0: IndexMode0 0: IndexMode2 1: IndexMode3
SM	0: Shiftmode0 1: Shiftmode1 2: Shiftmode2 3: Shiftmode3
Ts0	0b0-0b101 对应 T0-T5, 0b111 表示 0
Ts1 Ts2	0b0-0b101 对应 T0-T5, 0b110 表示 T6(Ttmp), 0b111 表示 0
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令为乘加减指令, 用于 FFT 运算. 当目标域为 SHU 和 MReg 时, 乘加的结果发往 SHU, 乘减的结果发往 MReg. 当目标域为 IMA 时, 乘加的结果发往 IMAx.Td, 乘减的结果发往 IMAx.(Td^1).

关于选项:

- 计算完成的结果需要根据 ShiftNum(Shiftmode 的 bit0~7)进行移位, 得到移位后的结果.
- FFT 默认进行小数计算, 对结果进行饱和处理.
- IndexModes 表示对结果进行交织的方式. IndexMode0 表示不进行交织. 目标域为 IMA 寄存器时, 必须进行交织. 目标域为 SHU 和 MReg 时, 不可以进行交织. 下图以目标域为 IMAX.T0 为例, 说明 IndexMode2|3 交织规则:



- Shiftmode 需要进行对应的配置, 其具体含义和配置方式可见 2.3.7.6. 其中 Shiftmode 寄存器的移位位数取值范围为 -2~5.

Note: FFT 指令结果输出到自身的寄存器时, 会占用两个数据通路, 参见 2.3.1.1.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4		
					Read ShiftMode	Read regs	Read MCW, ShiftMode		Write to MReg, SHU, IMA	Write to BIU		

- Latency**
- > BIU: 5
 - > MReg: 3
 - > SHU: 4
 - > IMA: 4

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec1024_t __ucpm2_fft(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int shiftmode, const int indexmode, const int mode, const int fu, const int pb)	IMAX: Ts0 AS Ts1*Ts2 (Shiftmodes) (IndexMode0) -> Dest

Example IMA1: 0 AS T1*T2 (Shiftmode1) (IndexMode2) -> IMA1.T3(Mode1);
IMA1: T5 AS T3 *T4 (Shiftmode2) (IndexMode0) -> M[dis](W7)(Latch7)
& SHU1.T1(Mode0);

2.3.7.4. Set MR 指令

Syntax	SRC IMAX: SetMR (Ts0) (H L O1 O2)
COND	({!}mode(0 1))

Encoding	39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 COND 1 0 00 HLO 0 0 0 0 1 0 1 111 111 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 Ts0 111 No (DST1 & DST2)
----------	---

Encode field:

Field	Value
COND	参见 2.3 章
HLO	L: 00 H: 01 O1: 10 O2: 11
No DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Tmp: 编码为 110, 表示 T6

Description 该指令是使用寄存器 Ts0 对 MR 进行配置.

n

关于 MR:

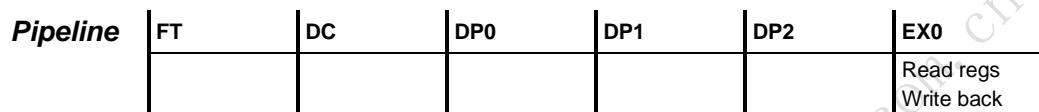
MR 寄存器是 IMA 中间结果寄存器, 常用于临时存放乘累加运算的中间结果. 每个 IMA 有 1 个 MR 寄存器, 可存放 16*104bit 数据, 其中每 104bit 数据由高 40bit 的溢出位和低 64bit 的数据构成.

- 当数据类型为 B(byte)时, MR 寄存器存放 64 个位宽为 26bit 的中间结果数据(10bit 溢出位+8bit 数据位+8bit 数据位). 每 4 个 Byte 的溢出位和数据位分别拼接后存在 MR 的 104bit 中(4 个数据的溢出位依次放在 40bit 溢出位上);
- 当数据类型为 S(short)时, MR 寄存器存放 32 个位宽为 42bit 的中间结果数据(高 10bit 为溢出位+16bit 数据位+16bit 数据位). 每 2 个 Short 的溢出位和数据位分别拼接后存在 MR 的 104bit 中(低位数据的溢出位放在 40bit 溢出位的 bit10~19, 高位数据的溢出位放在 bit30~39, 多出的 20bit 溢出位无效);
- 当数据类型为 W(word)时, MR 寄存器存放 16 个位宽为 74bit 的中间结果数据(高 10bit 为溢出位+32bit 数据位+32bit 数据位) (数据的溢出位放在 40bit 溢出位的 bit30~39, 多出的 30bit 溢出位无效).

关于选项:

-
- L 选项: 用 Ts0 的每个 Word 配置 MR 每 104bit 的低 32bit.
 - H 选项: 用 Ts0 的每个 Word 配置 MR 每 104bit 的 bit32~63.
 - O1 选项: 用 Ts0 的 32 个 Short 的低 10bit, 依次配置 MR 每 104bit 溢出位的低 20bit.
 - O2 选项: 用 Ts0 的 32 个 Short 的低 10bit, 依次配置 MR 每 104bit 溢出位的高 20bit.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_setMR(ucp_vec512_t s0, const int hlo, const int mode, const int fu, const int pb)	IMAx: SetMR (Ts0) (H L O1 O2)

Example IMA0: SetMR (0) (O1) (Mode0);

2.3.7.5. Read MR 指令

Syntax	SRC	IMAx: ReadMR (H L O1 O2)
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td 2. & SHUn.Td	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	00	HLO	0	0	1	0	1	0	1	0	1	111	111				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
111		111		DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
HLO	L: 00 H: 01 O1: 10 O2: 11
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是将 MR 寄存器指定位置的数据取出, 组成 512bit 结果发往目标域.

关于 MR:

MR 寄存器是 IMA 中间结果寄存器, 常用于临时存放乘累加运算的中间结果. 每个 IMA 有 1 个 MR 寄存器, 可存放 16*104bit 数据, 其中每 104bit 数据由高 40bit 的溢出位和低 64bit 的数据构成.

- 当数据类型为 B(byte)时, MR 寄存器存放 64 个位宽为 26bit 的中间结果数据(10bit 溢出位+8bit 数据位+8bit 数据位). 每 4 个 Byte 的溢出位和数据位分别拼接后存在 MR 的 104bit 中(4 个数据的溢出位依次放在 40bit 溢出位上);

-
- 当数据类型为 S(short)时, MR 寄存器存放 32 个位宽为 42bit 的中间结果数据(高 10bit 为溢出位+16bit 数据位+16bit 数据位). 每 2 个 Short 的溢出位和数据位分别拼接后存在 MR 的 104bit 中(低位数据的溢出位放在 40bit 溢出位的 bit10~19, 高位数据的溢出位放在 bit30~39, 多出的 20bit 溢出位无效);
 - 当数据类型为 W(word)时, MR 寄存器存放 16 个位宽为 74bit 的中间结果数据(高 10bit 为溢出位+32bit 数据位+32bit 数据位) (数据的溢出位放在 40bit 溢出位的 bit30~39, 多出的 30bit 溢出位无效).

关于选项:

- L 选项: 取 MR 每 104bit 的低 32bit, 拼接成 512bit 结果.
- H 选项: 取 MR 每 104bit 的 bit32~63, 拼接成 512bit 结果.
- O1 选项: 取 MR 每 104bit 溢出位的低 20bit, 每 10bit 扩展成一个 Short, 拼接成 512bit 结果.
- O2 选项: 取 MR 每 104bit 溢出位的高 20bit, 每 10bit 扩展成一个 Short, 拼接成 512bit 结果.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4
						Read regs	Read MCW	Write to IMA.M R, IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 5
-> MReg: 3
-> SHU: 4
-> IMA: 4
-> MR: 3
-> Ttmp: 3

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_readMR(const int hlo, const int mode, const int fu, const int pb)	IMAx: ReadMR (H L O1 O2) -> Dest

Example IMA0: ReadMR (H) -> BIU3.T1 & SHU2.T1(Mode1);

2.3.7.6. Set Shiftmode 指令

Syntax	SRC	IMAX: SetShiftmode (Ts0)
		->
	DST1	Shiftmoded
	COND	({!}mode(0 1))

Encoding	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
	COND	1	0	00	00	0	1	0	0	1	0	0	1	0	1	0	Shiftmoded	111		
	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	111																No (DST1 & DST2)		

Encode field:

Field	Value
COND	参见 2.3 章
Shiftmoded	Shiftmode0: 00 Shiftmode1: 01 Shiftmode2: 10 Shiftmode3: 11
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
No DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令设置 Shiftmode 寄存器的值.

每个 IMA 单元各有 4 个 Shiftmode 寄存器, 分别为 Shiftmode0-Shiftmode3. 每个 Shiftmode 寄存器共有 16 个 word, 每个 word 宽度为 19bit, 其中低 6bit 是移位的值, 最高 11bit 表示 IMA 计算的配置参数.

该指令将 Ts0 寄存器的每 32bit 数的最低 19bit 的值赋给 Shiftmode 变量, 后续指令通过选用 Shiftmode 变量即可进行相应的移位操作.

- 对于 Shiftmode 寄存器中的每个 word(19bit 数), 低 6bit 用于表示移位的位数, 取值范围为 -8~N-1, 其中负数表示右移, 正数表示左移. N 为数据类型的 bit 数, 即: 对于 byte 类型计算 N=8, short 类型计算 N=16, word 类型计算 N=32.
- 特别的: 对于 FFT 乘法, 移位范围只支持 -8~7.
- 若后续使用 Shiftmode 寄存器的指令为 word(32bit)类型计算, 则每个 word 的移位值由相应的 Shiftmode 寄存器中的值指定, 即: 需要移位的寄存器 Tx 中的每个 word: Tx[0]~Tx[15] 的移位值为 Shiftmode[0]~Shiftmode[15]指定.

- 若后续使用 Shiftmode 寄存器的指令为 short(16bit)类型计算, 需要移位的寄存器 Tx 中的每个 short: Tx[0]~Tx[31]的移位值, 相邻的 2 个 short 由同一个 Shiftmode 的值指定. 例如: Tx[0]~Tx[1]的移位值都是 Shiftmode[0], Tx[30]~Tx[31] 的移位值都是 Shiftmode[15].
- 若后续使用 Shiftmode 寄存器的指令为 byte(8bit)类型计算, 需要移位的寄存器 Tx 中的每个 byte: Tx[0]~Tx[63]的移位值, 相邻的 4 个 byte 由同一个 Shiftmode 的值指定. 例如: Tx[0]~Tx[3]的移位值都是 Shiftmode[0], Tx[60]~Tx[63] 的移位值都是 Shiftmode[15].

对于 Shiftmode 寄存器中的每个 word(19bit 数), 最高 11bit 用于配置 IMA 计算. 注意, 对于每个 Shiftmode 寄存器中 16 个 word, 低 6bit 表示的移位数值可以不同, 但最高 11bit 值必须全部相同. 该 11bit 数 (bit{18-8})的说明如下:

- bit{8}Num, 表示乘累加数是 1 个还是 2 个. 注意有 C 选项(复数计算)时候只能有 1 个乘累加.
- bit{9} Conj 表示乘法部分第一个操作取共轭, Ts0+-Ts2*Ts1, Ts2 虚部取反(实数不变, 虚数取反).
- bit{10} C 表示复数计算, 注意如果指令里本身有 C 选项, 只要有 1 个 C 就表示进行复数计算.
- bit{13-11} KIndex 用法见乘法指令.
- bit{15-14} MIndex 用法见乘法指令.
- bit{16} IsKIndex 表示是不是选 Ts2 的扩展模式.
- bit{17} second is T, {Ts2,Ts1} or Ts2 >>(MIndex*2+1).
- bit{18} second is sub, 进行第二次计算时, 加减操作是不是和前一次相反.

Note:

- 对于使用 ShiftMode 的指令, 应当保证指令结束前 ShiftMode 的值不变.
- Shiftmode 每个 Word 的默认值为 1.

Execution

Pipeline	FT	DC	DPO	DP1	DP2	EX0
						Read regs Write back

Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_setSM0(ucp_vec512_t s0, const int mode, const int fu, const int pb)	IMAx: SetShiftmode (Ts0) -> Shiftmode0
	void __ucpm2_setSM1(ucp_vec512_t s0, const int mode, const int fu, const int pb)	IMAx: SetShiftmode (Ts0) -> Shiftmode1
	void __ucpm2_setSM2(ucp_vec512_t s0, const int mode, const int fu, const int pb)	IMAx: SetShiftmode (Ts0) -> Shiftmode2
	void __ucpm2_setSM3(ucp_vec512_t s0, const int mode, const int fu, const int pb)	IMAx: SetShiftmode (Ts0) -> Shiftmode3

Example IMA3: SetShiftmode (T0) -> Shiftmode3;

2.3.7.7. 向量等于指令

Syntax	IMAx: Ts0 == Ts1 (B S W)
	->
DST1	<ol style="list-style-type: none"> 1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2	<ol style="list-style-type: none"> 1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	Type	0	0	0	0	1	0	1	1	0	Ts1		11				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Ts0		111										DST1 & DST2						

Encode field:

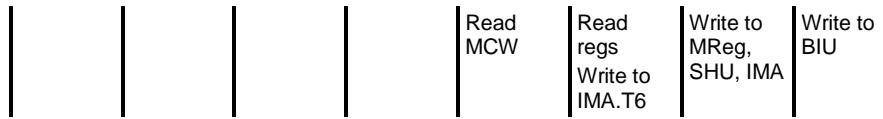
Field	Value
COND	参见 2.3 章
Type	B: 00 S: 01 W: 10
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 和 Ts1 中的数据进行比较, 相等则对应的结果数据为

- n 1, 不等则结果数据为 0. 结果发往目标域寄存器.
 - W 选项存在, 表示比较的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.

Execution

Pipeline | FT | DC | DP0 | DP1 | DP2 | EX0 | EX1 | EX2



Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > MR: 1
- > Ttmp: 1

Example IMA2: T0 == T5 (B) -> BIU3.T0;

2.3.7.8. 向量不等于指令

Syntax	SRC	IMAx: Ts0 != Ts1 (B S W)
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	1	Type	0	0	0	0	1	0	1	1	0	Ts1	11					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Ts0	111														DST1 & DST2			

Encode field:

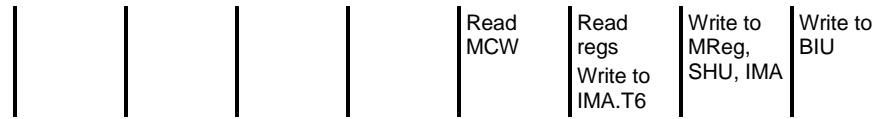
Field	Value
COND	参见 2.3 章
Type	B: 00 S: 01 W: 10
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 和 Ts1 中的数据进行比较, 不相等则对应的结果数据为 1, 相等则结果数据为 0. 结果发往目标域寄存器.

- W 选项存在, 表示比较的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.

Execution

Pipeline | FT | DC | DP0 | DP1 | DP2 | EX0 | EX1 | EX2



Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > MR: 1
- > Ttmp: 1

Example IMA2: T5 != T4 (W) -> BIU2.T3;

2.3.7.9. Read Flag 指令

Syntax	SRC	IMAx: ReadFlag (AC C)
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	0	0	0	Flag	0	0	0	0	1	1	0	1	1	1	1	11	
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	111	111																	DST1 & DST2

Encode field:

Field	Value
COND	参见 2.3 章
Flag	AC: 1 C: 0
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令读取 IMA 单元的溢出标志寄存器的值, 发往目标域.

- AC 表示读 IMA 单元加法部分的溢出标志寄存器, C 表示读 IMA 单元乘法部分的溢出标志寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3

- > MReg: 1
- > SHU: 2
- > IMA: 2
- > MR: 1
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_readFlag(const int cac, const int mode, const int fu, const int pb)	IMAx: ReadFlag (AC C) -> Dest

Example IMA2: ReadFlag (AC) -> SHU3.T0(Mode1);

2.3.7.10. Set Flag 指令

Syntax	SRC	IMAx: SetFlag (Ts0) (AC C)
	COND	({!}mode(0 1))

Encoding	39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20
	COND 1 1 0 0 0 Flag 0 0 0 0 1 1 0 1 1 1 1 11 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
	1 Ts0 111 No DST1 & DST2

Encode field:

Field	Value
COND	参见 2.3 章
Flag	AC: 1 C: 0
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Tmp: 编码为 110, 表示 T6
No DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将 Ts0 寄存器中的值写入 IMA 单元的溢出标志寄存器.

- n - AC 表示写入 IMA 单元加法部分的溢出标志寄存器, C 表示写入 IMA 单元乘法部分的溢出标志寄存器.

Execution



Latency 2

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_setFlag(ucp_vec512_t s0, const int cac, const int mode, const int fu, const int pb)	IMAx: SetFlag (Ts0) (AC C)

Example IMA2: SetFlag (T5) (C);

2.3.7.11. RMax 指令

Syntax	SRC	1. IMAx: Rmax (Ts0) (B S W) {(U)} (SlipMode0 SlipMode1) 2. IMAx: Rmax (Ts0) (B S) {(U)} (SlipMode2)
		无外部目标域, 或:
		->
	DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{(shift)}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
	DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

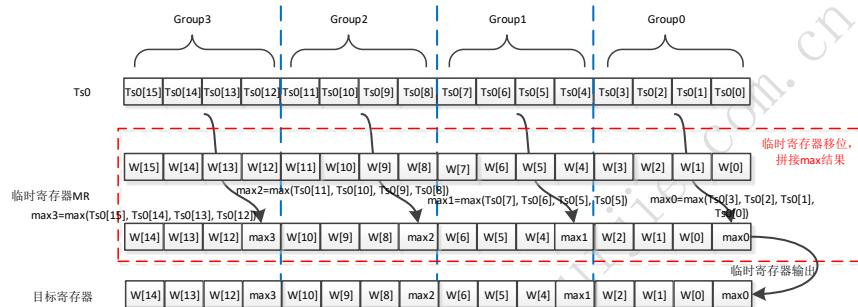
Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	0	U	0	SlipMod	e	1	0	0	0	1	Ts0/111	111				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0		Ts0		{No} (DST1 & DST2)															

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
SlipMode	10: SlipMode0 01: SlipMode1 00: SlipMode2
Ts0/111	编码 Ts0, 如果符合以下情况之一: 1. SlipMode1 2. SlipMode2 3. W 类型 其他情况编码为 0b111
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6

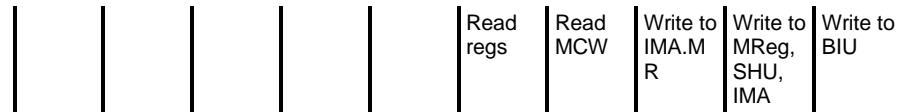
Description 该指令是将 Ts0 寄存器中的的每 2/4/8 个(由 SlipMode0/1/2 决定)个 Byte/Short/Word(由 B/S/W 决定)数据进行比较. 先将临时寄存器 MR 中的数据进行移位, 然后将每组最大的数据写入临时寄存器 MR 的相应位置, 最后将多次求最大值拼接的结果(暂存在 MR 中)存入目标寄存器中. 该指令执行的数据操作如下图所示(以 Slipmode1, word 类型为例):



- B/S/W 指定每个数的类型, 分别为 byte/short/word 类型, 即每个数分别由 1/2/4 个字节组成. 上面示例图中数据为 word 类型, 即每个数包含 4 个 byte, Ts0 共有 16 个 word.
- SlipMode0/1/2 指定每组比较 2/4/8 个数. 上面示例图中为 Slipmode1, 即每组计算有 4 个数, 由于 word 类型时, Ts0 共有 16 个 word, 因此整个计算分为 4 组, 对每组数据执行的操作完全相同.
- 数据分组: 该指令在 B 选项(byte 类型)时, 数据分组与 S/W 选项时不同. S/W 时, 由相邻的数据组成分组; B 选项时, 由间隔 1 个数据的方式划分分组. 例如, 在上面的示例图中, W 类型时每组数据相邻, group0 的数据为: 0,1,2,3, group1 的数据为 4,5,6,7...; 如果是 B 选项, group0 的数据为: 0,2,4,6, group1 的数据为 1,3,5,7, group2 的数据为 8,10,12,14, group3 的数据为 9,11,13,15.....
- U 选项表示无符号数比较.
- 关于临时寄存器 MR: 临时寄存器 MR 在该指令中为隐含寄存器, 程序员不可见, 每次 RMax 指令会自动使用和修改临时寄存器, 该机制可以用于连续使用 RMax 指令, 将最终结果的每个数据都置为每条 RMax 的结果. 第一次使用 RMax 时临时寄存器 MR 的值可能是之前其他指令产生的中间结果, 其对 RMax 指令一般无意义.

Execution

Pipeline | FT | DC | DP0 | DP1 | DP2 | EX0 | EX1 | EX2 | EX3 | EX4



Latency

- > BIU: 5
- > MReg: 3
- > SHU: 4
- > IMA: 4
- > MR: 3

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_rmax(ucp_vec512_t s0, const int type_BSW, const int f_U, const int slipmode, const int mode, const int fu, const int pb)</code>	IMAX: Rmax (Ts0) (B S W) {(U)} (SlipMode0 SlipMode1) -> Dest 或 IMAX: Rmax (Ts0) (B S) {(U)} (SlipMode2) -> Dest
	<code>void __ucpm2_rmaxV(ucp_vec512_t s0, const int type_BSW, const int f_U, const int slipmode, const int mode, const int fu, const int pb)</code>	IMAX: Rmax (Ts0) (B S W) {(U)} (SlipMode0 SlipMode1) 或 IMAX: Rmax (Ts0) (B S) {(U)} (SlipMode2)

Example IMA2: Rmax(T4) (S) (U) (SlipMode2) -> M[219] (W4);

2.3.7.12. RMin 指令

Syntax	SRC	1. IMAx: Rmin (Ts0) (B S W) {(U)} (SlipMode0 SlipMode1) 2. IMAx: Rmin (Ts0) (B S) {(U)} (SlipMode2)
		无外部目标域, 或:
		->
	DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
	DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

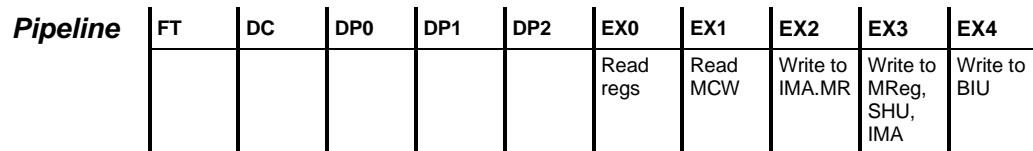
39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	0	U	0	SlipMod e	1	0	1	0	Ts0/111	111						
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	Ts0														{No} (DST1 & DST2)			

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
SlipMode	10: SlipMode0 01: SlipMode1 00: SlipMode2
Ts0/111	编码 Ts0, 如果符合以下情况之一: 1. SlipMode1 2. SlipMode2 3. W 类型 其他情况编码为 0b111
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 参见 2.3.7.11 RMax 指令功能描述, 区别仅为每组数据的比较由选出最大值变为选出最小值.

Execution



Latency

- > BIU: 5
- > MReg: 3
- > SHU: 4
- > IMA: 4
- > MR: 3

Intrinsics	Intrinsic	Generated Instruction
	<pre>ucp_vec512_t __ucpm2_rmin(ucp_vec512_t s0, const int type_BSW, const int f_U, const int slipmode, const int mode, const int fu, const int pb)</pre>	IMAx: Rmin (Ts0) (B S W) {(U)} (SlipMode0 SlipMode1) -> Dest 或 IMAx: Rmin (Ts0) (B S) {(U)} (SlipMode2) -> Dest
	<pre>void __ucpm2_rminV(ucp_vec512_t s0, const int type_BSW, const int f_U, const int slipmode, const int mode, const int fu, const int pb)</pre>	IMAx: Rmin (Ts0) (B S W) {(U)} (SlipMode0 SlipMode1) 或 IMAx: Rmin (Ts0) (B S) {(U)} (SlipMode2)

Example IMA2: RMin(T5) (B) (U) (SlipMode1) -> M[219](W2);

2.3.7.13. 向量加法指令

Syntax	SRC	IMAx: Ts0 + Ts1 (B S W) {(T)} {(U)} {(CI)}{(Flag)}
	->	
DST1		1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2		1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
COND		({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	CI	WSB	flag	U	T	0	0	0	1	0	0	Ts1	111					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	111														DST1 & DST2			

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将寄存器 Ts0 和 Ts1 的数据相加, 结果发往目标域.

关于选项:

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- T 选项存在表示对结果进行截断处理, 不存在表示饱和处理.
- U 选项表示是无符号数, 无 U 选项则认为是有符号数.

-
- CI 选项存在表示溢出标志寄存器参与运算，即运算为 $Ts0 + Ts1 + ACFlag$. 默认不使用.
 - Flag 表示根据运算结果更新加法部分的溢出标志寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<pre>ucp_vec512_t __ucpm2_add(ucp_vec512_t s0, ucp_vec512_t s1, const int type_BSW, const int f_T, const int f_U, const int f_CI, const int f_Flag, const int mode, const int fu, const int pb)</pre>	IMAx: $Ts0 + Ts1 (B S W) \{(T)\}$ $\{(U)\} \{(CI)\} \{(Flag)\} \rightarrow Dest$

Example IMA2: T0 + T3 (W) (U) (CI) (Flag) -> BIU3.T2(Mode0);

2.3.7.14. 向量减法指令

Syntax	SRC	IMAx: Ts0 – Ts1 (B S W) {(T)} {(U)} {(CI)}{(Flag)}
	->	
DST1		1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2		1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
COND		({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	CI	WSB	flag	U	T	0	1	0	1	0	0	0	Ts1		111			
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0		111												DST1 & DST2				

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将寄存器 Ts0 和 Ts1 的数据相减, 结果发往目标域.

关于选项:

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- T 选项存在表示对结果进行截断处理, 不存在表示饱和处理.
- U 选项表示是无符号数, 无 U 选项则认为是有符号数.
- CI 选项存在表示溢出标志寄存器参与运算, 即运算为 Ts0-Ts1-ACFlag. 默认不使用.

-
- Flag 表示根据运算结果更新加法部分的溢出标志寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_sub(ucp_vec512_t s0, ucp_vec512_t s1, const int type_BSW, const int f_T, const int f_U, const int f_CI, const int f_Flag, const int mode, const int fu, const int pb)	IMAx: Ts0 - Ts1 (B S W) {(T)} {(U)} {(CI)} {(Flag)} -> Dest

Example IMA2: T3 - T1 (B) (U) (CI) (Flag) -> BIU0.T1;

2.3.7.15. 向量传送指令

Syntax	SRC	IMAx: Ts0
		->
	DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
	DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	10	0	0	0	0	0	0	0	0	1	0	0	111	111			
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	111	DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
No DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将寄存器 Ts0 中的数据发往目标域.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3
 -> MReg: 1
 -> SHU: 2

-> IMA: 2
-> Ttmp: 1

Example IMA2: T0 ->SHU2.T1(Mode0);

Smart Logic Confidential For yuyi@ruijie.com.cn

2.3.7.16. 向量 Conj 指令

Syntax	SRC	IMAx: CONJ (Ts0) (B S W)
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	00	0	1	0	1	1	1	0	111	111						
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	111		Ts0	DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 中的复数数据进行取共轭操作, 结果发往目标域.

- W 选项存在, 表示计算的数据粒度为 Word, 偶数 Word 为实部, 奇数 Word 为虚部; S 选项存在, 表示数据粒度为 Short, 每 32bit 中低 16 位为实部, 高 16 位为虚部; B 选项存在, 表示数据粒度为 Byte, 每 32bit 中低 2byte 为实部, 高 2byte 为虚部.
- 如果结果溢出, 则按照饱和处理.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_conj(ucp_vec512_t s0, const int type_BSW, const int mode, const int fu, const int pb)	IMAx: CONJ (Ts0) (B S W) -> Dest

Example IMA2: Conj(T5) (B) ->SHU1.T2;

2.3.7.17. 向量 RAdd 指令

Syntax	SRC	IMAx: RAdd (Ts0) (B S W) (SlipMode0 SlipMode1) {(U)} IMAx: RAdd (Ts0) (B S) (SlipMode2) {(U)}
		->
	DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
	DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

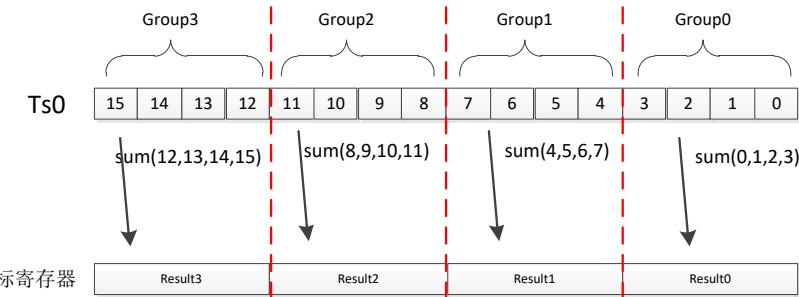
Encoding	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
	COND	1	1	WSB	0	U	1	SlipMode	1	0	0	0	0	Ts0/111	111					
	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Ts0		111										DST1 & DST2						

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
SlipMode	10: SlipMode0 01: SlipMode1 00: SlipMode2
Ts0/111	编码 Ts0, 如果符合以下情况之一: 1. SlipMode1 2. SlipMode2 3. W 类型 其他情况编码为 0b111
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是将 Ts0 寄存器中的的每 2/4/8 个 (由 SlipMode0/1/2 决定) Byte/Short/Word (由 B/S/W 决定) 数据进行相加, 每组的和写入目标寄存器的相应位置. 该指令执行的数据操作如下图所示 (以 Slipmode1,

word 类型为例, 方格中数据的数组表示索引, 0-15 依次表示 Ts0[0]-Ts0[15]):



- U 选项表示无符号计算.
- 数据分组: 该指令在 B 选项(byte 类型)时, Ts0 中的数据分组与 S/W 选项时不同. S/W 时, 由相邻的数据组成分组; B 选项时, 由间隔 1 个数据的方式划分分组. 例如, 在上面的示例图中, W 类型时每组数据相邻, group0 的数据为 0,1,2,3, group1 的数据为 4,5,6,7; 如果是 B 选项, group0 的数据为 0,2,4,6, group1 的数据为 1,3,5,7, group2 的数据为 8,10,12,14, group3 的数据为 9,11,13,15.....
- 对于每组计算, 该指令使用 2 倍于原始数据的宽度存放加法结果, 并在计算结果的宽度范围内进行符号扩展. 每组的计算结果, 写入与源数据对应的低两个位置, 其余位置置 0.
 - 例 1: 对于 byte 类型的计算, 每组的计算结果使用 16bit 存放加法结果, 每组计算结果的低 16bit 做符号扩展; 对于 short 类型的计算, 每组的计算结果使用 32bit 存放加法结果, 每组计算结果的低 32bit 做符号扩展.
 - 例 2: 对于 byte 类型的计算, 使用 Slippmode2, 即 8 个 byte 为一组进行加法, 每一组的结果为 8 个 byte(64bit), 但该指令对于 byte 类型计算使用 16bit 保存计算结果, 所以每组计算结果的 64bit 的低 16bit 为该组原始 8 个 byte 的加法结果(进行 16bit 符号扩展), 每组的高 48bit 全部值 0.

Note: RAdd 指令将会占用 MR, 修改 MR 寄存器的值.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2	EX3	EX4
						Read regs	Read MCW	Write to IMA.MR	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 5

- > MReg: 3
- > SHU: 4
- > IMA: 4
- > MR: 3

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_radd(ucp_vec512_t s0, const int type_BSW, const int slipmode, const int f_U, const int mode, const int fu, const int pb)	IMAx: RAdd (Ts0) (B S W) (SlipMode0 SlipMode1) {(U)} -> Dest 或 IMAx: RAdd (Ts0) (B S) (SlipMode2) {(U)} -> Dest

Example IMA2: RAdd(T5) (W) (SlipMode0) -> M[221](W6)(Mode0);

2.3.7.18. 向量 ABS 指令

Syntax	SRC	IMAx: ABS (Ts0) (B S W)
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	00	0	0	0	1	1	1	0	111	111						
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	111		Ts0	DST1 & DST2															

Encode field:

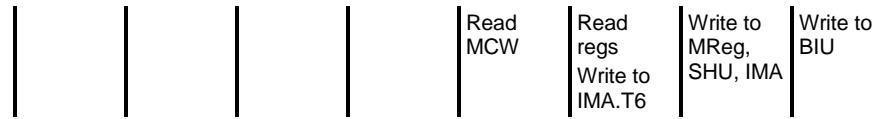
Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 中的数据取绝对值, 结果发往目标域.

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- 如果结果溢出, 则按照饱和处理.

Execution

Pipeline |FT|DC|DP0|DP1|DP2|EX0|EX1|EX2



Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Example IMA2: Abs(T4) (B) -> M[223](W0);

2.3.7.19. 向量 ByteOR 指令

Syntax	SRC	IMAX: ByteOr (Ts0, Ts1)
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAX.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	00	0	1	0	0	1	1	1	1	1	0	111	111				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	Ts1	DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description

该指令将寄存器中的数据分为 8 个 64bit, 在每个 64bit 内进行或操作. 即 64bit 又分为 8 个 8bit 数, 若 Ts1 的 8bit 最低位为 1, 则取出 Ts0 对应位置的 8bit, 然后将取出的数进行或操作, 或的结果放在 512bit 结果中对应 64bit 的低 8bit 位置. 最终结果发往目标域寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3
-> MReg: 1
-> SHU: 2
-> IMA: 2
-> Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_byteOr(ucp_vec512_t s0, ucp_vec512_t s1, const int mode, const int fu, const int pb)	IMAx: ByteOr (Ts0, Ts1) -> Dest

Example IMA2: ByteOr (T0, T5) -> BIU0.T0(Mode0);

2.3.7.20. 向量按位与指令

Syntax	SRC	IMAX: Ts0 & Ts1
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 1. BIUn.Td	
DST2	1. & IMAX.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	00	00	0	1	1	1	1	1	1	1	0	Ts1	111				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Ts0	111	DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将寄存器 Ts0 和 Ts1 中的数据按位与, 结果发往目标域寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3

-> MReg: 1

-> SHU: 2

-> IMA: 2
-> Ttmp: 1

Example IMA2: T4 & T1 -> IMA2.T1 & SHU0.T0(Mode0);

2.3.7.21. 向量按位或指令

Syntax	SRC	IMAx: Ts0 Ts1
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	01	00	0	1	1	1	1	1	1	1	0	Ts0	111				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	111		Ts1	DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将寄存器 Ts0 和 Ts1 中的数据按位或, 结果发往目标域寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3

-> MReg: 1

-> SHU: 2

-
- > IMA: 2
 - > Ttmp: 1

Example IMA2: T4 | T3 -> BIU3.T1 & SHU0.T0(Mode0);

2.3.7.22. 向量按位异或指令

Syntax	SRC	IMAX: Ts0 ^ Ts1
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAX.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20		
COND	1	0	10	00	0	1	1	1	1	1	1	1	0	Ts1	Ts0						
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
		Ts0		Ts1	DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将寄存器 Ts0 和 Ts1 中的数据按位异或, 结果发往目标域寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3

-> MReg: 1

-> SHU: 2

-> IMA: 2
-> Ttmp: 1

Example IMA2: T4 ^ T3 -> IMA2.T1 & SHU0.T0(Mode0);

2.3.7.23. 向量按位非指令

Syntax	SRC	IMAx: ~Ts0
		->
	DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
	DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20					
COND	1	0		11		00	0	1	1	1	1	1	1	0		Ts0		111						
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
		111		111		DST1 & DST2																		

Encode field:

Field	Value
COND	参见 2.3 章
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将寄存器 Ts0 中的数据取反, 结果发往目标域寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3
 -> MReg: 1
 -> SHU: 2

-> IMA: 2
-> Ttmp: 1

Example IMA2: ~T3 -> IMA2.T1 & SHU0.T0(Mode0);

Smart Logic Confidential For yuyi@ruijie.com.cn

2.3.7.24. 普通逻辑指令

Syntax	SRC	IMAX: Logic(Ts0, Ts1, Ts2, Ts3) (G0 G1 G2 G3)
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAX.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	GranFlag	00	0	1	1	1	1	1	1	1	0	Ts2	Ts3				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts1	Ts0																	DST1 & DST2

Encode field:

Field	Value
COND	参见 2.3 章
Ts0~Ts3	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
GranFlag	00: G0 01: G1 02: G2 03: G3
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是一般的逻辑类指令, 该指令的功能描述如下:

设有两个数 Log0 和 Log1, 则:

- G0 时: Log0 为 0, Log1 为 0;
- G1 时: Log0 为 0xFF, Log1 为 0;
- G2 时: Log0 为 0, Log1 为 0xFF;
- G3 时: Log0 为 0xFF, Log1 为 0xFF

对每个 Byte 进行以下逻辑运算, 得到结果:

$(Log0 \mid Ts1) \& (Log1 \wedge Ts2) \mid (Ts0 \& \sim Ts3)$

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Example IMA2: Logic(T2, T3, T4, Ttmp) (G0) -> IMA2.T1 & SHU0.T0(Mode0);

2.3.7.25. 向量比较选择指令(CmpSel0)

Syntax	SRC	IMAX: CompSel(Ts0, Ts1, Ts2, Ts3) (B S W) {(U)}
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAX.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	0	U	0	0	0	1	1	0	1	Ts1	Ts3					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	Ts2	DST1 & DST2																

Encode field:

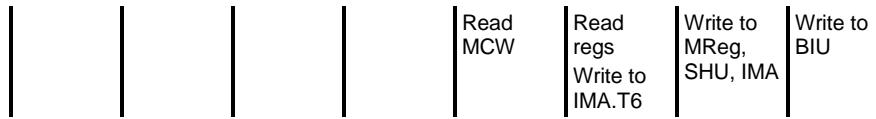
Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0~Ts3	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令比较寄存器 Ts0 和 Ts1 的大小来选择 Ts2 或者 Ts3 作为结果 (Ts0>=Ts1 ? Ts2 : Ts3).

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- U 选项表示是无符号数运算, 无 U 选项则认为是有符号数运算.

Execution

Pipeline | FT | DC | DP0 | DP1 | DP2 | EX0 | EX1 | EX2



Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_compSel(ucp_vec512_t s0, ucp_vec512_t s1, ucp_vec512_t s2, ucp_vec512_t s3, const int type_BSW, const int f_U, const int mode, const int fu, const int pb)</code>	IMAx: CompSel(Ts0, Ts1 , Ts2 , Ts3) (B S W) {(U)} -> Dest.

Example IMA1: CompSel(T5, T3 , T2, T1) (B) (U) -> M[219] (W7) (Mode0);

2.3.7.26. 数据压缩 Cprs 指令

Syntax	SRC	IMAx: CPRS(Ts0)(CprsMode0 CprsMode1){(U)}
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	0	CM	0	U	1	0	1	1	1	1	0	1	111	111			
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	111	DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
CM	0: CprsMode0 1: CprsMode1
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 的数据进行压缩, 结果发往目标域寄存器.

- CprsMode0 表示将 Ts0 的每 16bit 数压缩为 8bit, 放在结果数据每 16bit 的低 8 位.
- CprsMode1 表示将 Ts0 的每 32bit 数压缩为 16bit, 放在结果数据每 32bit 的低 16 位.
- U 选项表示是无符号数, 无 U 选项则认为是有符号数.
- 如果结果溢出, 则按照饱和处理.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_cprs(ucp_vec512_t s0, const int cprsmode, const int f_U, const int mode, const int fu, const int pb)	IMAx: CPRS(Ts0) (CprsMode0 CprsMode1) {(U)} -> Dest

Example IMA0: CPRS(T4) (CprsMode0) (U) -> IMA3.T5(Mode0);

2.3.7.27. 向量排序 Index 指令

Syntax	SRC	IMAX: Index(Ts0, Ts1)
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAX.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	00	00	1	1	0	1	1	0	1	1	0	1	111	111			
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	Ts1	DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description

该指令将寄存器中的数据分为 4 个 128bit, 在每个 128bit 内对数据进行排序操作: 在第 i 个 128bit 中, 取 Ts1 的第 j 个 Byte 的低 4 比特作为索引 Index, 然后取 Ts0 的第 Index 个 Byte, 放入结果的第 i 个 128bit 的第 j 个 Byte 处. 结果发往目标域寄存器.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3
-> MReg: 1
-> SHU: 2
-> IMA: 2
-> Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_index(ucp_vec512_t s0, ucp_vec512_t s1, const int mode, const int fu, const int pb)	IMAx: Index(Ts0, Ts1) -> Dest

Example IMA0: Index(T0, T5) -> BIU3.T3(Mode1);

2.3.7.28. 向量除法指令

Syntax	<table border="1"> <tr> <td>SRC</td><td>1. IMAx: DivStart (Ts0, Ts1) (B S W){(U)}</td></tr> <tr> <td>COND</td><td>({!}mode(0 1))</td></tr> </table>	SRC	1. IMAx: DivStart (Ts0, Ts1) (B S W){(U)}	COND	({!}mode(0 1))			
SRC	1. IMAx: DivStart (Ts0, Ts1) (B S W){(U)}							
COND	({!}mode(0 1))							
SRC	<table border="1"> <tr> <td>2. IMAx: ReadQ (B S W){(U)}</td> </tr> <tr> <td>-></td> </tr> </table>	2. IMAx: ReadQ (B S W){(U)}	->					
2. IMAx: ReadQ (B S W){(U)}								
->								
DST1	<table border="1"> <tr> <td>1. M[t] (Wx)</td> </tr> <tr> <td>2. M[{{S++}, {I++}, {A++}}] (Wx)</td> </tr> <tr> <td>3. M[dis{(shift)}] (Wx) (LatchID)</td> </tr> <tr> <td>4. IMAn.Td</td> </tr> <tr> <td>5. IMA[{0}, {1}, {2}, {3}].Td</td> </tr> <tr> <td>6. SHUn.Td</td> </tr> <tr> <td>7. BIUn.Td</td> </tr> </table>	1. M[t] (Wx)	2. M[{{S++}, {I++}, {A++}}] (Wx)	3. M[dis{(shift)}] (Wx) (LatchID)	4. IMAn.Td	5. IMA[{0}, {1}, {2}, {3}].Td	6. SHUn.Td	7. BIUn.Td
1. M[t] (Wx)								
2. M[{{S++}, {I++}, {A++}}] (Wx)								
3. M[dis{(shift)}] (Wx) (LatchID)								
4. IMAn.Td								
5. IMA[{0}, {1}, {2}, {3}].Td								
6. SHUn.Td								
7. BIUn.Td								
DST2	<table border="1"> <tr> <td>1. & IMAx.Td(可选)</td> </tr> <tr> <td>2. & SHUn.Td(可选)</td> </tr> </table>	1. & IMAx.Td(可选)	2. & SHUn.Td(可选)					
1. & IMAx.Td(可选)								
2. & SHUn.Td(可选)								
COND	({!}mode(0 1))							

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding Src1: DivStart

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	0	U	0	00	1	1	0	0	Ts1		Ts0					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	111		Ts1	No {DST1 & DST2}															

Src2:ReadQ

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	0	U	0	10	1	1	0	0	111		100					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	111		111	DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0, Ts1	T0-T3: 使用 T0-T3 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令为除法相关指令.

n

对于 Src1:

该指令表示除法运算开始. 寄存器 Ts0 作为被除数, Ts1 作为除数.

对于 Src2:

该指令表示将除法的商写往目标域寄存器.

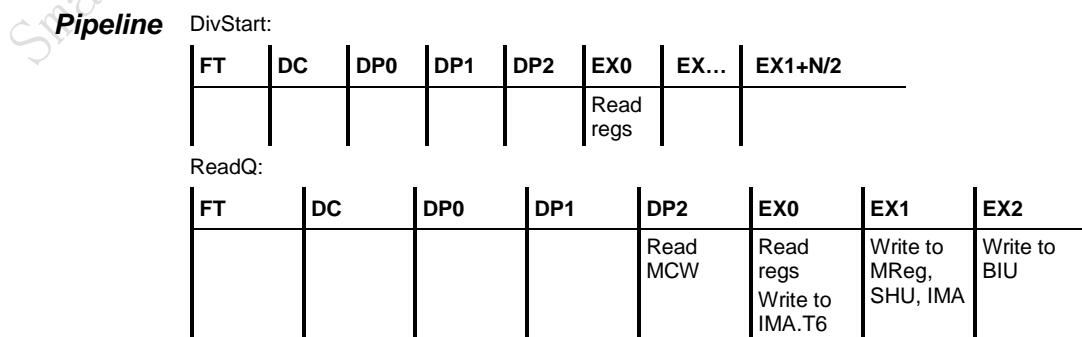
关于选项:

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- U 选项表示是无符号数, 无 U 选项则认为是有符号数.

Note:

- 在做除法期间(从 DivStart 开始到 $2+N/2$ 拍), 将一直占用寄存器 T4, T5, Shiftmode2 和 Shiftmode3 寄存器, 并且除法结果会存储在 T5 中. 因此在做除法期间(从 DivStart 开始到 $2+N/2$ 拍)不能使用 T4, T5, Shiftmode2, Shiftmode3 这几个寄存器作为其他指令的源操作数和目的操作数, 在 ReadQ 之前不能使用 T5 作为其他指令的源操作数和目的操作数.
- DivStart 的 $2+N/2$ 拍以后可以使用 ReadQ 读商(N 表示数据粒度的位宽).
- 在做除法期间(从 DivStart 开始到 $2+N/2$ 拍)不能发射的指令包括: RADD, RMAX, RMIN, FFT 乘法, 不能使用以 MR 作为源或目的寄存器的 byte 类型乘法.
- 发射 DivStart 指令之后, Ts0 和 Ts1 的数据不需要保留.
- 该指令硬件不保证除以 0 的结果.

Execution



Latency DivStart 为 $2+N/2$ 拍, N 为计算数据类型的 bit 数, 即: 对于 byte 类型计算 N=8, short 类型计算 N=16, word 类型计算 N=32.
ReadQ 为(1+传输周期),

传输周期:

- > BIU: 2
- > MReg: 0
- > SHU: 1
- > IMA: 1
- > Ttmp: 0

Example IMA0: DivStart (T0, T1) (B) (U) (Mode1);
MFetch: Repeat @ (5);
IMA0: ReadQ (B) (U) -> M[0] (W0) (Mode1) ;

2.3.7.29. 向量 Count1 指令

Syntax	SRC	IMAx: Count1 (Ts0)
	->	
DST1		1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2		1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
COND		({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	00	00	1	1	1	1	1	0	1	Ts0	111						
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ts0	111	DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是将寄存器 Ts0 分为 64 个 byte, 数每个 byte 中 1 的个数, 并将结果放入目标寄存器的对应 byte 中.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3
-> MReg: 1

-
- > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_count(ucp_vec512_t s0, const int mode, const int fu, const int pb)	IMAx: Count1 (Ts0) -> Dest

Example IMA0: Count1 (T4) -> IMA3.T4 (Mode1);

2.3.7.30. 向量 First 指令

Syntax	SRC	IMAx: First (Ts0) (B S W){(U)}
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	0	U	1	0	0	1	1	1	1	0	111	111				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	111	Ts0	DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令查找寄存器 Ts0 中每个数据的第一个 1/0 的位置, 结果发往目标域.

- W 选项存在, 表示数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- 该指令找出寄存器 Ts0 中每个数据 (byte/short/word) 的第一个 1/0 的位置(从高位开始查找), 并将查找的结果值放入目标寄存器的对应位置上.
- 如果没有 U 选项, 表示有符号数, 将符号位去掉开始计数, 寻找第一个非符号位, 即: 若符号位为 1, 从非符号位中寻找第一个 0 的位置;

若符号位为 0, 从非符号位中寻找第一个 1 的位置. 若有 U 选项, 则找第一个 1 的位置

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_first(ucp_vec512_t s0, const int type_BSW, const int f_U, const int mode, const int fu, const int pb)	IMAx: First (Ts0) (B S W){(U)} -> Dest

Example IMA0: First (T0) (B)(U) -> M[223](W4) & IMA0.T3;

2.3.7.31. 向量位反序指令

Syntax	SRC	IMAx: BitReverse(Ts0)
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	1	00	00	1	0	0	1	1	1	1	0	111	111					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0				DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是对寄存器 Ts0 中的数据进行位反序, 结果发往目标域. 将寄存器 Ts0 分为 64 个 byte, 对每个 byte 进行位反序操作, 得到最终结果, 结果发往目标域.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3

-> MReg: 1

-> SHU: 2

-> IMA: 2

-> Ttmp: 1

Intrinsics

Intrinsic	Generated Instruction
ucp_vec512_t __ucpm2_bitReverse(ucp_vec512_t s0, const int mode, const int fu, const int pb)	IMAx: BitReverse(Ts0) -> Dest

Example IMA0: BitReverse(T1) -> IMA1.T2(Mode0);

2.3.7.32. 向量立即数左移指令

Syntax	SRC	IMAX: Ts0 << Imm5 (B S W) {(U)} {(T)}
	->	
DST1		1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{(shift)}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2		1. & IMAX.Td(可选) 2. & SHUn.Td(可选)
COND		({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	
COND	1	1	WSB	0	U	T	1	0	0	1	1	1	0		Imm5					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ts0		Ts0		DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 中的数据进行左移, 左移的位数由 Imm5 决定, 结果发往目标域.

关于选项:

- W 选项存在, 表示数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- U 选项表示无符号定点数据格式, 默认为有符号定点数据格式.
- T 选项存在表示结果溢出时进行截断处理, 默认饱和处理.

移位操作:

- 对寄存器 Ts0 中的每个数据 (byte/short/word) 进行左移操作.
- 对于 byte(8bit)类型数据: 每个 byte 的移位数由 Imm5 的低 3bit 数据决定, Imm5 的高 2bit 必须是 0, 并且该 3bit 数据为无符号数, 即: $Td[i] = Ts0[i] << (Imm5 \& 0x7)$, $0 \leq i \leq 63$.
- 对于 short(16bit)类型数据: 每个 short 的移位数由 Imm5 的低 4bit 数据决定, Imm5 的高 1bit 必须是 0, 并且该 4bit 数据为无符号数, 即: $Td[i] = Ts0[i] << (Imm5 \& 0xF)$, $0 \leq i \leq 31$.
- 对于 word(32bit)类型数据: 每个 word 的移位数由 Imm5 数据决定, 并且该 5bit 数据为无符号数, 即: $Td[i] = Ts0[i] << Imm5$, $0 \leq i \leq 15$.

Note: byte 类型移位范围为 0~7, short 类型移位范围为 0~15, word 类型移位范围为 0~31.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t</code> <code>__ucpm2_LShiftImm(ucp_vec51</code> <code>2_t s0, int s1, const int</code> <code>type_BSW, const int f_T, const</code> <code>int f_U, const int mode, const int</code> <code>fu, const int pb)</code>	<code>IMAx: Ts0 << Imm5 (B S W) {(U)}</code> <code>{(T)} -> Dest</code>

Example `IMA0:T3 << 13 (S)(T) -> M[dis (shift)](W6) (Latch6) &`
`SHU2.T2(mode1);`

2.3.7.33. 向量寄存器左移指令

Syntax	SRC	IMAX: Ts0 << Ts1 (B S W) {(U)} {(T)}
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAX.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	
COND	1	0	WSB	0	U	T	1	0	0	1	1	1	111		Ts1					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ts0		Ts0		DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 中的数据进行移位操作, 移位的位数由 Ts1 决定, 结果发往目标域.

关于选项:

- W 选项存在, 表示数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- U 选项表示无符号定点数据格式, 默认为有符号定点数据格式.
- T 选项存在表示结果溢出时进行截断处理, 默认饱和处理.

移位操作:

- 对寄存器 Ts0 中的每个数据 (byte/short/word) 进行移位。移位的位数由 Ts1 中对应数据的低 6 位决定，该 6bit 数据为有符号数，正数表示左移，负数表示右移。
- 移位支持的范围：byte 类型移位范围为 -7~7, short 类型移位范围为 -15~15, word 类型移位范围为 -31~31。该指令移位时会使用整个 6bit 数作为移位数，程序员应自行保证其不超出限制范围。

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t</code> <code>__ucpm2_LShiftT(ucp_vec512_t</code> <code>s0, ucp_vec512_t s1, const int</code> <code>type_BSW, const int f_T, const</code> <code>int f_U, const int mode, const int</code> <code>fu, const int pb)</code>	<code>IMAx: Ts0 << Ts1 (B S W) {(U)}</code> <code>{(T)} -> Dest</code>

Example `IMA0:T3 << T4 (B) (U) (T) -> M[128](w0) & IMA0.T2(mode1);`

2.3.7.34. 向量立即数右移指令

Syntax	SRC	IMAx: Ts0 >> Imm5 (B S W) {(U)}
	->	
DST1		1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2		1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
COND		({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	
COND	1	1	WSB	0	U	0	0	0	0	1	1	1	0		Imm5					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ts0		Ts0		DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 中的数据进行右移, 左移的位数由 Imm5 决定, 结果发往目标域.

关于选项:

- W 选项存在, 表示数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- U 选项表示无符号定点数据格式, 默认为有符号定点数据格式.

移位操作:

- 对寄存器 Ts0 中的每个数据 (byte/short/word) 进行右移操作, 右移的位数由 Imm5 决定.
- 对于 byte(8bit)类型数据: 每个 byte 的移位数由 Imm5 的低 3bit 数据决定, Imm5 的高 2bit 必须是 0, 并且该 3bit 数据为无符号数, 即: $Td[i] = Ts0[i] >> (Imm5 \& 0x7)$, $0 \leq i \leq 63$.
- 对于 short(16bit)类型数据: 每个 short 的移位数由 Imm5 的低 4bit 数据决定, Imm5 的高 1bit 必须是 0, 并且该 4bit 数据为无符号数,, 即: $Td[i] = Ts0[i] >> (Imm5 \& 0xF)$, $0 \leq i \leq 31$.
- 对于 word(32bit)类型数据: 每个 word 的移位数由 Imm5 数据决定, 并且该 5bit 数据为无符号数, 即: $Td[i] = Ts0[i] >> Imm5$, $0 \leq i \leq 15$.

Note: byte 类型移位范围为 0~7, short 类型移位范围为 0~15, word 类型移位范围为 0~31.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t</code> <code>__ucpm2_RShiftImm(ucp_vec51</code> <code>2_t s0, int s1, const int</code> <code>type_BSW, const int f_U, const</code> <code>int mode, const int fu, const int</code> <code>pb)</code>	<code>IMAx: Ts0 >> Imm5 (B S W) {(U)}</code> <code>-> Dest</code>

Example IMA2:T3 >> 11 (S) (U) -> IMA0.T5;

2.3.7.35. 向量寄存器右移指令

Syntax	SRC	IMAX: Ts0 >> Ts1 (B S W) {(U)} {(T)}
	->	
DST1		1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2		1. & IMAX.Td(可选) 2. & SHUn.Td(可选)
COND		({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	
COND	1	0	WSB	0	U	T	0	0	0	1	1	1	111						Ts1	
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ts0		Ts0		DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令对寄存器 Ts0 中的数据进行移位操作, 移位的位数由 Ts1 决定, 结果发往目标域.

关于选项:

- W 选项存在, 表示数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- U 选项表示无符号定点数据格式, 默认为有符号定点数据格式.
- T 选项存在表示结果溢出时进行截断处理, 默认饱和处理.

移位操作:

- 对寄存器 Ts0 中的每个数据 (byte/short/word) 进行移位, 移位的位数由 Ts1 中对应数据的低 6 位决定, 并且该 6bit 数据为有符号数, 正数表示右移, 负数表示左移.
- 移位支持的范围: byte 类型移位范围为 -7~7, short 类型移位范围为 -15~15, word 类型移位范围为 -31~31. 该指令移位时会使用整个 6bit 数作为移位数, 程序员应自行保证其不超出限制范围.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency -> BIU: 3

-> MReg: 1

-> SHU: 2

-> IMA: 2

-> Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_RShiftT(ucp_vec512_t s0, ucp_vec512_t s1, const int type_BSW, const int f_T, const int f_U, const int mode, const int fu, const int pb)	IMAX: Ts0 >> Ts1 (B S W) {(U)} {(T)} -> Dest

Example IMA0:T3 >> T4 (B) (U) -> M[S++](w4) & SHU2.T2(mode1);

2.3.7.36. 拼接移位指令

Syntax	SRC	IMAx: SpliceShift(Ts0,Ts1,Ts2)
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	
COND	1	0	00	1	0	0	0	0	0	0	0	1	1	1	111		Ts2			
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ts0		Ts1		DST1 & DST2																

Encode field:

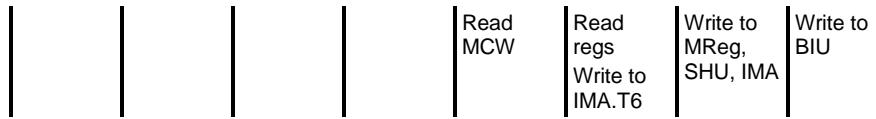
Field	Value
COND	参见 2.3 章
Ts0~Ts2	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是对寄存器 Ts0 和 Ts1 进行拼接移位操作, 移位的位数由 Ts2 决定, 结果发往目标域.

将寄存器 Ts0 和 Ts1 分别划分成 64 个 byte, 再将 Ts0 和 Ts1 的对应 byte 进行拼接(Ts0 为高位, Ts1 为低位), 得到 64 个 short 的中间结果. 然后以 Ts2 中每一个 byte 数据的低 3 位为移位位数, 对中间结果的每个 short 进行右移, 最终取出右移后每个 short 的低 8 位, 拼接成 512bit 作为最终结果.

Execution

Pipeline |FT|DC|DP0|DP1|DP2|EX0|EX1|EX2



Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_spliceShift(ucp_vec51 2_t s0, ucp_vec512_t s1, ucp_vec512_t s2, const int mode, const int fu, const int pb)</code>	IMAx: SpliceShift(Ts0,Ts1,Ts2) -> Dest

Example IMA0: SpliceShift(T0,T5,T2) -> BIU3.T0(Mode0);

2.3.7.37. 比特筛选 BitFilter 指令

Syntax	SRC	IMAx: BitFilter(Ts0,Ts1)
	->	
DST1	1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	1	00	00	1	0	1	1	1	1	1	0	Ts1	111					
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0		DST1 & DST2																	

Encode field:

Field	Value
COND	参见 2.3 章
Ts0,Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description

该指令进行比特筛选操作(polar bit 收集).

将寄存器 Ts0 和 Ts1 中的数据分别划分成 64 个 byte, Ts0 和 Ts1 相对应的每一 byte 中, 将 Ts1 中值为 “1” 的比特所在位置对应的 Ts0 的比特筛选出来, 拼接放到结果的低位, 高位补零.

Note: Ts1 中为 0 的位, Ts0 中对应位上的值也应该为 0, 否则结果会出错.

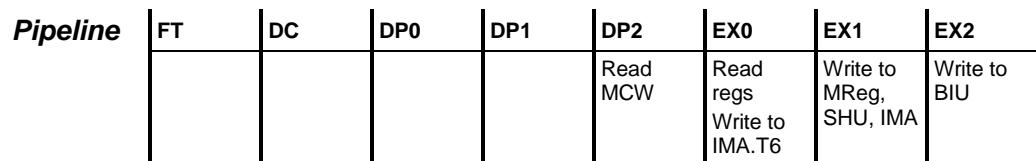
例:

Ts0: (高位)00010010(低位)

Ts1: (高位)10111010(低位)

结果: (高位)00000101(低位)

Execution



- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	ucp_vec512_t __ucpm2_filter(ucp_vec512_t s0, ucp_vec512_t s1, const int mode, const int fu, const int pb)	IMAx: BitFilter(Ts0,Ts1) -> Dest

Example IMA1: BitFilter (T3, T5) -> M[S++](W3) & IMA1.T2;

2.3.7.38. 比特展开 BitExpd 指令

Syntax	SRC	IMAx: BitExpd(Ts0,Ts1)
	->	
DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td	
DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)	
COND	({!}mode(0 1))	

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	
COND	1	0	00	00	1	0	1	1	1	1	1	0	Ts1	111						
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ts0		111		DST1 & DST2																

Encode field:

Field	Value
COND	参见 2.3 章
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是根据寄存器 Ts1 的值对寄存器 Ts0 进行比特扩展, 结果发往目标域.

将寄存器 Ts0 和 Ts1 中的数据分别划分成 64 个 byte, 将 Ts0 的每个 byte 作为中间结果的初始值, 然后从 Ts1 每个 byte 的低位看起, 若该位是 0, 则将以该位为最低位的原子串左移一位并舍弃最高位, 并将中间结果对应位置为 0, 形成新的中间结果; 若该位是 1, 则中间结果保持原值. 如此依次对 Ts1 的所有比特进行一次操作, 得到最终结果.

例:

Ts0: (高位) 00110110 (低位)

Ts1: (高位) 01011010 (低位)

结果: (高位) 00011000 (低位)

- 从 Ts1 的最低位起, 第 0 位为 0, 所以以该位为最低位的原子串左移一位并舍弃最高位, 并且中间结果相应位置为 0, 中间结果变为 01101100;
- Ts1 的第 1 位为 1, 所以中间结果不变 01101100;
- Ts1 的第 2 位为 0, 所以以该位为最低位的原子串左移一位并舍弃最高位, 并且中间结果相应位置为 0, 中间结果变为 11011000;
- 其他位依此类推.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<code>ucp_vec512_t __ucpm2_expd(ucp_vec512_t s0, ucp_vec512_t s1, const int mode, const int fu, const int pb)</code>	IMAx: BitExpd (Ts0,Ts1) -> Dest

Example IMA0: BitExpd(T2, T1) -> M[20] (W0);

2.3.7.39. 加法后取模指令

Syntax	SRC	IMAx: ModAdd(Ts0,Ts1) (B S W) {(U)}
	->	
DST1		1. M[t] (Wx) 2. M[{S++}, {I++}, {A++}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
DST2		1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
COND		{(!)mode(0 1)}

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0	WSB	0	U	1	1	0	1	1	1	0	Ts1		111				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ts0		Ts0		DST1 & DST2															

Encode field:

Field	Value
COND	参见 2.3 章
WSB	00: B 01: S 10: W
Ts0, Ts1	T0-T5: 使用 T0-T5 寄存器 0: 编码为 111, 表示 0 Ttmp: 编码为 110, 表示 T6
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description

该指令进行加法后取模的操作, 即 $Ts0 \geq Ts1 ? (Ts0 - Ts1) : Ts0$.

在 $a+b$ 模 c 操作中, $Ts1$ 中存着 c , $Ts0$ 存的是上一条指令 $a+b$ (截断处理加法) 的结果. 其中 a 和 b 都是在 $0 \sim (c-1)$ 的范围, 则 $Ts0$ 中的数就是在 $0 \sim (2c-2)$ 的范围, 没有超过 c 的两倍.

关于选项:

- W 选项存在, 表示计算的数据粒度为 Word; S 选项存在, 表示数据粒度为 Short; B 选项存在, 表示数据粒度为 Byte.
- U 选项表示是无符号数, 无 U 选项则认为是有符号数.

-
- 计算结果进行饱和处理.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

Latency

- > BIU: 3
- > MReg: 1
- > SHU: 2
- > IMA: 2
- > Ttmp: 1

Intrinsics	Intrinsic	Generated Instruction
	<pre>ucp_vec512_t __ucpm2_modAdd(ucp_vec512_ t s0, ucp_vec512_t s1, const int type_BSW, const int f_U, const int mode, const int fu, const int pb)</pre>	IMAx: ModAdd(Ts0,Ts1) (B S W) {(U)} -> Dest

Example IMA0: ModAdd(T2, T4) (W) (U) -> M[dis](W6)(Latch4);

2.3.7.40. 立即数赋值指令

Syntax	SRC	1. IMAx: V(Imm16) 2. IMAx: VHigh(Ts0, Imm16)
		->
	DST1	1. M[t] (Wx) 2. M[{{S++}, {I++}, {A++}}] (Wx) 3. M[dis{shift}] (Wx) (LatchID) 4. IMAn.Td 5. IMA[{0}, {1}, {2}, {3}].Td 6. SHUn.Td 7. BIUn.Td
	DST2	1. & IMAx.Td(可选) 2. & SHUn.Td(可选)
	COND	({!}mode(0 1))

Note: LatchID 说明及编码见 2.3.7: IMA 公共目标域及编码

Encoding SRC1:

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	0																	
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		111 DST1 & DST2																	

SRC2:

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
COND	1	1																	
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Ts0 DST1 & DST2																	

Encode field:

Field	Value
COND	参见 2.3 章
DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令是用立即数对寄存器进行赋值操作.

- IMAx: V(Imm16): Imm16 扩展成 32 份, 拼接得到 512bit, 写往目标寄存器.

-
- IMAX: VHigh(Ts0, Imm16): 将源寄存器和结果的数据分别分为 16 个 Word, 对于结果的每个 Word, 其低 16bit 为寄存器 Ts0 对应 Word 的低 16bit, 高 16bit 为 Imm16.

Execution

Pipeline	FT	DC	DP0	DP1	DP2	EX0	EX1	EX2
					Read MCW	Read regs Write to IMA.T6	Write to MReg, SHU, IMA	Write to BIU

- Latency**
- > BIU: 3
 - > MReg: 1
 - > SHU: 2
 - > IMA: 2
 - > Ttmp: 1

Example IMA0: VHigh(T3, 0x8778) -> BIU3.T2 & SHU2.T0 (Mode1);

2.3.7.41. Wait 立即数指令

Syntax	SRC	IMAX: Wait Imm6
--------	-----	-----------------

Encoding	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
	01	1	1	00	00	1	1	1	1	1	1	1	1	0						Imm6
	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																				No DST1 & DST2

Encode field:

Field	Value
No DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将 IMAX 的指令延迟 Imm6 拍之后执行, 最高延迟 62 拍. 具体行为参见 2.3.2.

Execution



Latency	1				
Intrinsics					
	<table border="1"><thead><tr><th>Intrinsic</th><th>Generated Instruction</th></tr></thead><tbody><tr><td>void __ucpm2_imaWaitImm(const int imm6, const int fu, const int pb)</td><td>IMAX: Wait Imm6</td></tr></tbody></table>	Intrinsic	Generated Instruction	void __ucpm2_imaWaitImm(const int imm6, const int fu, const int pb)	IMAX: Wait Imm6
Intrinsic	Generated Instruction				
void __ucpm2_imaWaitImm(const int imm6, const int fu, const int pb)	IMAX: Wait Imm6				

Example IMA2: Wait 3;

2.3.7.42. Wait KI 指令

Syntax	SRC	IMAx: WaitKI																																																													
Encoding																																																															
39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20																																																															
<table border="1"> <tr> <td>01</td><td>1</td><td>1</td><td>00</td><td>00</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> <tr> <td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>00</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>No DST1 & DST2</td> </tr> </table>			01	1	1	00	00	1	1	1	1	1	1	0	0									19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	00	0																	No DST1 & DST2
01	1	1	00	00	1	1	1	1	1	1	0	0																																																			
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																												
1	00	0																	No DST1 & DST2																																												

Encode field:

Field	Value
No DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令将使 IMAx 的指令延迟执行, 最高延迟 62 拍. 具体行为参见 2.3.2.

Wait 的周期数由 KIy[m:n]指定, 其中 y 表示使用哪个 KI, m:n 表示使用该 KI 中的哪些 bit.

注意, 每个 KI 都是 24bit 寄存器, waitKI 指令使用特定的 6bit 作为 wait 的周期数.

y, m, n 与发射槽的对应关系为:

发射槽	KIy	[m:n]
BIU0	KI8	[5:0]
BIU1	KI8	[11:6]
BIU2	KI8	[17:12]
BIU3	KI8	[23:18]
SHU0	KI9	[5:0]
SHU1	KI9	[11:6]
SHU2	KI9	[17:12]
SHU3	KI9	[23:18]
IMA0	KI10	[5:0]
IMA1	KI10	[11:6]
IMA2	KI10	[17:12]
IMA3	KI10	[23:18]
R0	KI11	[5:0]
R1	KI11	[11:6]
R2	KI11	[17:12]
R3	KI11	[23:18]
R4	KI12	[5:0]
R5	KI12	[11:6]
R6	KI12	[17:12]
R7	KI12	[23:18]

Execution



Latency	1				
Intrinsics					
	<table><tr><td>Intrinsic</td><td>Generated Instruction</td></tr><tr><td>void __ucpm2_imaxWaitKI(const int fu, const int pb)</td><td>IMAX: WaitKI</td></tr></table>	Intrinsic	Generated Instruction	void __ucpm2_imaxWaitKI(const int fu, const int pb)	IMAX: WaitKI
Intrinsic	Generated Instruction				
void __ucpm2_imaxWaitKI(const int fu, const int pb)	IMAX: WaitKI				

Example IMAX: WaitKI;

2.3.7.43. NOP 指令

Syntax	SRC	NOP;
--------	-----	------

Encoding	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
	01	1	0	1	0	0	0	1	1	1	1	1	1	1	0		0			
	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0		0																No DST1 & DST2

Description 空指令.

Execution



Latency 1

Example IMA0: NOP;

2.3.7.44. Set Condition 指令

Syntax	SRC	IMAX: Imm9
		->
	DST1	mode0

Note: 本指令无条件执行

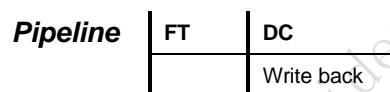
39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20
01		1	0	0	Imm9{8-6}	1	1	1	1	1	1	1	0		Imm9{5:0}				
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																No DST1 & DST2			

Encode field:

Field	Value
No DST1 & DST2	参见 2.3.7 章, IMA 公共目标域及编码

Description 该指令用 9 位立即数对 IMA 单元的 mode0 寄存器进行设置, 即配置指令执行的条件. mode0 寄存器的值对应的条件表达式见 2.3 的条件表达式值编码表.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_immaSetCond(const int imm9, const int fu, const int pb)	IMAX: Imm9 -> mode0

Example IMAX: 172 -> mode0;

2.3.8. MFetch

2.3.8.1. 定点加法指令

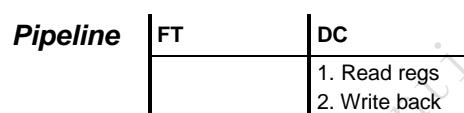
Syntax	SRC	MFetch: Kls1 + Kls0
		->
	DST1	Kld

Note: 本指令无条件执行

23	22	21	20	19	18	17	16	15	14	13	12
0	1	0	0	0	0	0	0	0	0	0	0
11	10	9	8	7	6	5	4	3	2	1	0
Kld				Kls1				Kls0			

Description 将寄存器 Kls1 和 Kls0 的定点值相加, 其结果值放入 Kld 中. 操作为 24bit 加法操作, 结果做截断处理.

Execution



Latency 1

Example MFetch: KI2 + KI1 -> KI3;

2.3.8.2. 定点减法指令

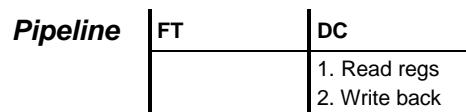
Syntax	SRC	MFetch: Kls1 - Kls0
		->
	DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	1	0	0	0	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 将寄存器 Kls1 和 Kls0 的定点值相减, 其结果值放入 Kld 中. 操作为 24bit 减法操作.

Execution



Latency 1

Example MFetch: KI3 - KI1 -> KI9;

2.3.8.3. 定点比较指令: 小于

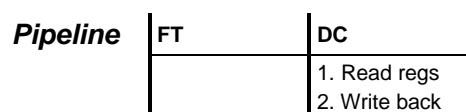
Syntax	MFetch: Kls1 < Kls0
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	0	1	0	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 比较寄存器 Kls1 的定点值是否小于 Kls0, 将其结果值放入 Kld 中.

Execution



Latency 1

Example MFetch: KI3 < KI10 -> KI9;

2.3.8.4. 定点比较指令: 小于等于

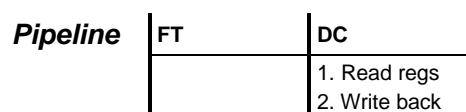
Syntax	MFetch: Kls1 <= Kls0
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	0	1	0	1	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 比较寄存器 Kls1 的定点值是否小于等于 Kls0, 将其结果值放入 Kld 中。

Execution



Latency 1

Example MFetch: KI8 <= KI9 -> KI7;

2.3.8.5. 定点比较指令: 等于

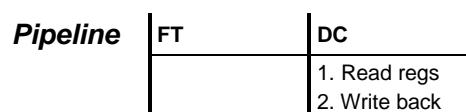
Syntax	MFetch: Kls1 == Kls0
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	0	1	1	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 比较寄存器 Kls1 的定点值是否等于 Kls0, 将其结果值放入 Kld 中.

Execution



Latency 1

Example MFetch: KI8 == KI9 -> KI11;

2.3.8.6. 定点比较指令: 不等于

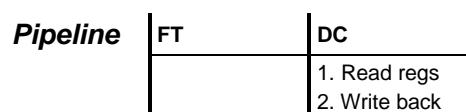
Syntax	MFetch: Kls1 != Kls0
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	0	1	1	1	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 比较寄存器 Kls1 的定点值是否不等于 Kls0, 将其结果值放入 Kld 中.

Execution



Latency 1

Example MFetch: KI1 != KI2 -> KI12;

2.3.8.7. 定点左移指令

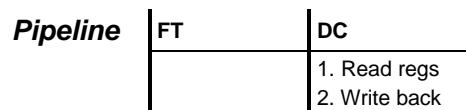
Syntax	SRC	MFetch: Kls1 << Kls0
		->
	DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	0	0	1	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 将寄存器 Kls1 根据 Kls0 的低 5bit 进行左移，其结果值放入 Kld 中。操作作为 24bit 左移操作。若 Kls0 的高 19 位不为 0，则左移结果为 0。

Execution



Latency 1

Example MFetch: KI1 << KI0 -> KI5;

2.3.8.8. 定点右移指令

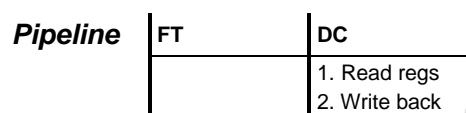
Syntax	MFetch: Kls1 >> Kls0
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	0	0	1	1	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 将寄存器 Kls1 根据 Kls0 的低 5bit 进行右移，高位补 0，其结果值放入 Kld 中。操作为 24bit 右移操作。若 Kls0 的高 19 位不为 0，则右移结果为 0。

Execution



Latency 1

Example MFetch: KI1 >> KI4 -> KI5;

2.3.8.9. 定点按位与指令

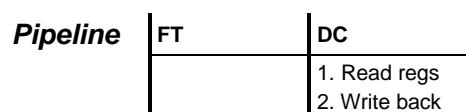
Syntax	MFetch: Kls1 & Kls0
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	1	0	0	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1			Kls0			

Description 将寄存器 Kls1 和 Kls0 进行按位与操作，其结果值放入 Kld 中。

Execution



Latency 1

Example MFetch: KI1 & KI3 -> KI5;

2.3.8.10. 定点按位或指令

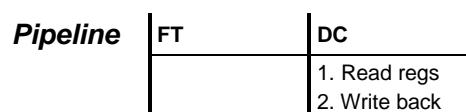
Syntax	MFetch: Kls1 Kls0
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	1	0	0	1	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld					Kls1					Kls0	

Description 将寄存器 Kls1 和 Kls0 进行按位或操作，其结果值放入 Kld 中。

Execution



Latency 1

Example MFetch: KI1 | KI3 -> KI5;

2.3.8.11. 定点按位非指令

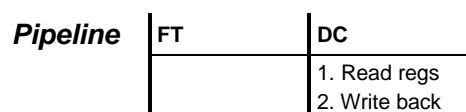
Syntax	MFetch: ~ Kls
	->
DST1	Kld

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	1	0	1	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	Kld				0	0	0	0	Kls			

Description 对寄存器 Kls 进行取非操作，其结果值放入 Kld 中。

Execution



Latency 1

Example MFetch: ~KI1 -> KI5;

2.3.8.12. 定点传输指令

Syntax	SRC	MFetch: KIs
		->
	DST1	KId {(CR)}

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	1	0	1	1	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	KId			CR	0	0	0	KIs				

Description 将寄存器 KIs 值放入 KId 中.

- CR 表示清除 KId 对应的 Loop 循环, 使用该功能时, 必须与 JUMP 指令相结合, 跳转到该 CR 清除语句. 并且当带 CR 选项的指令位于 KId 对应的 Loop 循环(最内层循环)内部时, 该 Loop 所包含的指令数量必须大于三条, 且带 CR 选项的本条指令距离 Loop 末尾的指令编码长度必须大于 64 个 word; 当带 CR 选项的指令位于外层循环时, 不能作为该外层循环的最后一条指令.

Note: 当需要多次跳出 Loop 循环时, 跳转目标指令为该条带 CR 选项的指令, 并且 CR 指令不允许有并行的其他指令.

Execution

Pipeline	FT	DC
		1. Read regs 2. Write back

Latency 1

Example MFetch: KI7 -> KI5;

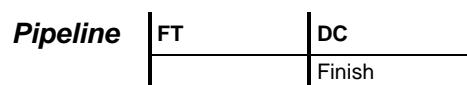
2.3.8.13. 立即数 Repeat 指令

Syntax	SRC	Mfetch: Repeat @(Imm12)
Note: 本指令无条件执行		

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	1	1	0	1		Imm12		
	11	10	9	8	7	6	5	4	3	2	1	0
				Imm12					0	0	0	0

Description 重复执行当前指令行, 其中 Imm12 为所重复执行的次数, 默认为十进制.

Execution



Latency 1

Example Mfetch: Repeat @(7);

2.3.8.14. 寄存器 Repeat 指令

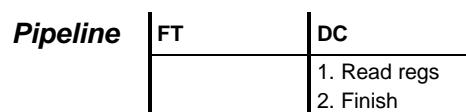
Syntax	SRC Mfetch: Repeat @(Kls Kls - Imm5)
--------	---

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	Imm5[4]	0	1	1	0	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Imm5[3:0]				Kls			

Description 重复执行当前指令行 Kls 次或(Kls-Imm5)次.

Execution



Latency 1

Example Mfetch: Repeat @(Kl2 - 3);

2.3.8.15. Loop 指令

Syntax	SRC	Mfetch: LpTo Label @ (KIs KIs - Imm5)
---------------	------------	---

Note:

本指令无条件执行

Label = 立即数或符号引用, 符号引用以%开头

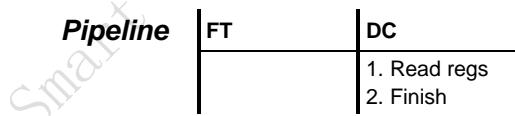
Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	0	Imm5[4]									
	11	10	9	8	7	6	5	4	3	2	1	0
				Label			Imm5[3:0]					KIs

Description 设置循环指令, 循环次数为 KIs 或 KIs-Imm5, 循环起始地址为该指令的下一地址, 循环结束地址为 Label. 其中 KIs 只能为 KI0~KI15, 最多支持 4 重循环.

Note:

- Label 为相对地址.
- 多层 Loop 指令嵌套使用时, loop 指令不能在其他 loop 指令的最后一行上.
- Loop 循环的最后一条指令不可以是带 CR 选项的 MFetch 定点传输指令.
- Jump 指令与 Loop 指令结合使用时, 必须使用带 CR 选项的 MFetch 定点传输指令清除对应循环.
- 不支持在循环体内使用带 CR 选项的 MFetch 定点传输指令清除本重循环.

Execution



Latency 1

Example Mfetch: LpTo %test @ (KI0);

2.3.8.16. 读 FIFO 指令

Syntax	SRC	Mfetch: FIFO.rx (B)
		->
	DST1	KId

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	1	1	1	1	B	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	KId			1	0	rx		0	0	0	0	0

Encode field:

Field	Value
rx	FIFO.r0: 00
	FIFO.r1: 01
	FIFO.r2: 10
	FIFO.r3: 11

Description 该指令是将 FIFO 中的数读到 KId 中.

FIFO.rx 可选择四个 SPU 到 MPU 的 FIFO 中的一个, 其中每个 FIFO 深度为 1. 当指定的 FIFO 为满时, 将 FIFO 的 24bit 数据读入 KId 中; 当 FIFO 为空时, B 选项表示阻塞 MPU 流水线, 直到 FIFO 不为空.

Execution



Latency

1

Intrinsics	Intrinsic	Generated Instruction
	int __ucpm2_readFIFO(const int fifo_r, const int fu, const int pb)	Mfetch: FIFO.rx (B) -> KId

Note:

参数 fifo_r 应为: FIFO_R0, FIFO_R1, FIFO_R2, FIFO_R3.

Example Mfetch: FIFO.r0(B) -> KI1;

2.3.8.17. 写 FIFO 指令

Syntax	Mfetch: Kls -> DST1 FIFO.wx (B)
---------------	--

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	1	1	1	1	B	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1		wx			Kls	

Encode field:

Field	Value
rx	FIFO.w0: 00
	FIFO.w1: 01
	FIFO.w2: 10
	FIFO.w3: 11

Description 该指令是将 Kls 中的数写入 FIFO.

FIFO.wx 可选择四个 MPU 到 SPU 的 FIFO 中的一个, 其中每个 FIFO 深度为 1. 当指定的 FIFO 为空时, 将 Kls 中的数写入 FIFO; 当 FIFO 为满时, B 选项表示阻塞 MPU 流水线, 直到 FIFO 为空.

Execution



Latency

1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_writeFIFO(const int fifo_w, int s0, const int fu, const int pb)	Mfetch: Kls -> FIFO.wx (B)

Note:

参数 fifo_w 应为: FIFO_W0, FIFO_W1, FIFO_W2, FIFO_W3.

2.3.8.18. 相对地址跳转指令

Syntax	SRC Mfetch: IF(Cond) JUMP Label {(RPC)} Or Mfetch: JUMP Label {(RPC)}
---------------	--

Note:

本指令无条件执行

Label = 立即数或符号引用, 符号引用以%开头

Encoding	23 22 21 20 19 18 17 16 15 14 13 12
	1 0 RPC Label
	11 10 9 8 7 6 5 4 3 2 1 0
	Label Cond

Encode field:

Field	Value
Cond	参见 2.3 章, 条件表达式编码表

Description 该指令根据 Label 确定的 13bit 相对地址, 进行跳转.

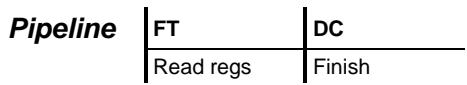
- 可用条件判断位 Cond 进行条件跳转, Cond 为 8 bit. Cond 的语法格式如下:
 - a) KIs== KI15/KI11/KI7/常数 0;
 - b) KIs>= KI15/KI11/KI7/常数 0;
 - c) KIs!= KI15/KI11/KI7/常数 0;
 - d) KIs< KI15/KI11/KI7/常数 0.
- 跳转条件成立时, 当 JumpMode 的 bit0 为 0 时, 程序立即发生跳转; 当其为 1 时, Jump 延迟槽生效, Jump 指令的后一条指令执行后程序跳转.
- RPC 选项表示是否将跳转指令的返回地址存入 KI15 中, 即当 RPC 选项存在且跳转条件成立时保存, RPC 选项不存在或者跳转条件不成立时不保存. 当 JumpMode 的 bit0 为 0 时, 返回地址是下一条指令的地址; 为 1 时, 返回地址为下下条指令的地址.

Note:

- Cond 不能配置为 KI7==KI7, 此种情况被 JumpPrefetch 占用.
- 跳转条件成立且 JumpMode 的 bit0 为 1 时, 该指令不能是 Loop 指令的最后一条指令, 并且若 JUMP 指令的后一条指令是 Loop 或 Jump, 则后一条指令不执行.

-
- 跳转条件成立且 JumpMode 的 bit0 为 1 时, Jump 的下一条指令不能使用带 CR 选项的 MFetch 定点传输指令.

Execution



Latency 1

Example Mfetch: IF(KI3 < 0) JUMP %test (RPC);

2.3.8.19. 绝对地址跳转指令

Syntax	Mfetch: IF(Cond) JUMP KIs { (RPC) } {(EI DI)} or Mfetch: JUMP KIs { (RPC) } {(EI DI)}
---------------	---

Note: 本指令无条件执行

Encoding	<table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="width: 10px;">23</td><td style="width: 10px;">22</td><td style="width: 10px;">21</td><td style="width: 10px;">20</td><td style="width: 10px;">19</td><td style="width: 10px;">18</td><td style="width: 10px;">17</td><td style="width: 10px;">16</td><td style="width: 10px;">15</td><td style="width: 10px;">14</td><td style="width: 10px;">13</td><td style="width: 10px;">12</td></tr> <tr> <td>1</td><td>1</td><td>RPC</td><td>0</td><td>0</td><td>0</td><td colspan="2">EI/DI</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr> <tr> <td colspan="6" style="text-align: center;">KIs</td><td colspan="6" style="text-align: center;">Cond</td></tr> </table>	23	22	21	20	19	18	17	16	15	14	13	12	1	1	RPC	0	0	0	EI/DI		0	0	0	0	11	10	9	8	7	6	5	4	3	2	1	0	KIs						Cond					
23	22	21	20	19	18	17	16	15	14	13	12																																						
1	1	RPC	0	0	0	EI/DI		0	0	0	0																																						
11	10	9	8	7	6	5	4	3	2	1	0																																						
KIs						Cond																																											

Encode field:

Field	Value
Cond	参见 2.3 章
EI/DI	无 Flag: 00 EI: 10 DI: 11

Description 该指令根据 KIs 确定的绝对地址, 进行跳转.

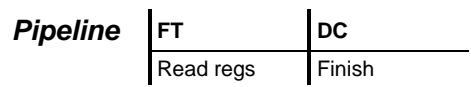
- 可用条件判断位 Cond 进行条件跳转, Cond 为 8 bit. Cond 的语法格式如下:
 - a) KIs == KI15/KI11/KI7/常数 0;
 - b) KIs >= KI15/KI11/KI7/常数 0;
 - c) KIs != KI15/KI11/KI7/常数 0;
 - d) KIs < KI15/KI11/KI7/常数 0
- 跳转条件成立时, 当 JumpMode 的 bit0 为 0 时, 程序立即发生跳转; 当其为 1 时, Jump 延迟槽生效, Jump 指令的下一条指令执行后程序跳转.
- RPC 选项表示是否将跳转指令的下一条指令地址存入 KI15 中, 即当 RPC 选项存在且跳转条件成立时保存, RPC 选项不存在或者跳转条件不成立时不保存. 当 JumpMode 的 bit0 为 0 时, 返回地址是下一条指令的地址; 为 1 时, 返回地址为下下条指令的地址.
- 可选项 EI 表示使能中断, DI 表示禁止中断, 默认情况为不修改.

Note:

- Cond 不能配置为 KI7 == KI7, 此种情况被 JumpPrefetch 占用.
- 跳转条件成立且 JumpMode 的 bit0 为 1 时, 该指令不能是 Loop 指令的最后一条指令, 并且若 JUMP 指令的后一条指令是 Loop 或 Jump, 则后一条指令不执行.

-
- 跳转条件成立且 JumpMode 的 bit0 为 1 时, Jump 的下一条指令不能使用带 CR 选项的 MFetch 定点传输指令.

Execution



Latency 1

Example Mfetch: IF(KI10 != KI11) JUMP KI3 (RPC);

2.3.8.20. JumpPreFetchImm

Syntax	SRC	Mfetch: JumpPreFetch Label
--------	-----	----------------------------

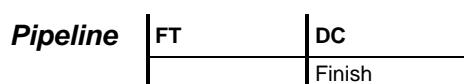
Note: Label = 立即数或符号引用, 符号引用以%开头

Encoding	23	22	21	20	19	18	17	16	15	14	13	12	
	1	0	0										
	11	10	9	8	7	6	5	4	3	2	1	0	
	Label			00111001									

Description 该指令根据 Label 确定的 13bit 相对地址, 将对应的指令取到 Jump 的指令预取槽中.

- Jump 的指令预取槽有两个, 每个深度为 3.
- 使用该指令预取时, 需要提前配置 JumpMode. 若 JumpMode 的 bit1 为 1, 则可以进行指令预取, 否则该指令相当于 NOP.

Execution



Latency 指令周期不确定, 至少是 4

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_jumpPreFetch (void* fn, const int fu, const int pb)	Mfetch: JumpPreFetch Label

Example Mfetch: JumpPreFetch 0x18;

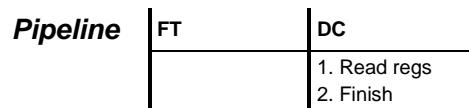
2.3.8.21. JumpPreFetchKI

Syntax	SRC	Mfetch: JumpPreFetch KIs
Encoding	23 22 21 20 19 18 17 16 15 14 13 12 1 1 0 0 11 10 9 8 7 6 5 4 3 2 1 0 KIs 00111001	

Description 该指令根据 KIs 确定的绝对地址, 将对应的指令取到 Jump 的指令预取槽中.

- Jump 的指令预取槽有两个, 每个深度为 3.
- 使用该指令预取时, 需要提前配置 JumpMode. 若 JumpMode 的 bit1 为 1, 则可以进行指令预取, 否则该指令相当于 NOP.

Execution



Latency 指令周期不确定, 至少是 4

Example Mfetch: JumpPreFetch KI5;

2.3.8.22. 传输指令: MFetch -> MReq

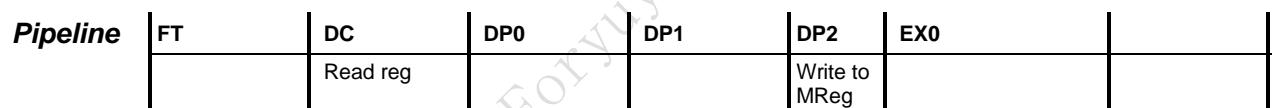
Syntax	SRC	Mfetch: Kls
		->
	DST1	M[t][i]

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	1	1	1	1	1	0		Kls		
	11	10	9	8	7	6	5	4	3	2	1	0
		t							i			

Description 对 KIs 的 24bit 数据高位补 0 后得到 32bit 数据, 写入 MReg[t]的第 i 个 Word 中.

Execution



Latency 1

Example Mfetch: KI12 → M[60][2];

2.3.8.23. 传输指令: MFetch -> BIU

Syntax	SRC	Mfetch: Kls
		->
	DST1	BIUx.Td[i]

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	1	0	1	1	1	0				Kls
	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0		BIUx		Td			i	

Description 对 Kls 的 24bit 数据高位补 0 后得到 32bit 数据, 写入 BIUx 的 Td 寄存器的第 i 个 Word 中.

Execution



Latency 1

Example Mfetch: KI13 -> BIU0.T1[15];

2.3.8.24. 立即数赋值指令

Syntax	SRC	Mfetch: SYM
		->
	DST1	KId (H L)

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	1	1	H L	1		KId			Imm16			
	11	10	9	8	7	6	5	4	3	2	1	0

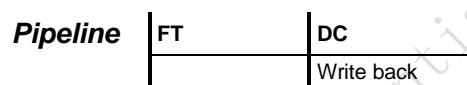
Imm16

Description 该指令使用立即数对 KId 进行赋值.

- SYM 为 Imm16 或符号引用, 符号引用以%开头.
- L 选项存在, 表示将 KId 的低 16 位赋值为立即数 SYM, 高位符号扩展. H 选项存在, 表示 KId 低位不变, 立即数配置其高位.

Note: KId 只能为 KI0~KI14.

Execution



Latency 1

Example Mfetch: 0x1234 -> KI11(L);

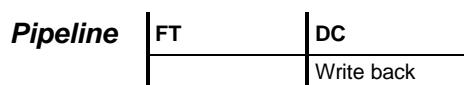
2.3.8.25. ConfigJumpMode

Syntax	SRC	Mfetch: ConfigJumpMode Imm2
Encoding	23 22 21 20 19 18 17 16 15 14 13 12 1 1 0 1 1111 0 11 10 9 8 7 6 5 4 3 2 1 0 0 Imm2	

Description 该指令使用 2bit 立即数对 JumpMode 进行配置.

- JumpMode 是一个 2bit 宽的寄存器, bit0 控制 Jump 的延迟槽, bit1 控制 Jump 的指令预取槽.
- 当 Jump 的延迟槽打开时, Jump 指令不能是循环的最后一条指令.

Execution



Latency

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_configJumpMode(const int imm2, const int fu, const int pb)	Mfetch: ConfigJumpMode Imm2

Example Mfetch: ConfigJumpMode 0;

2.3.8.26. MPU Wait 指令

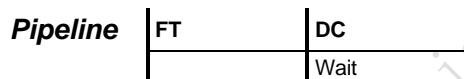
Syntax	SRC Mfetch: MPUWait imm3 {(B)}
--------	---------------------------------------

Note: 本指令无条件执行

Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	0	0	0	0	1	B	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	Imm3

Description 该指令应与其他指令槽的 Wait 指令配合使用。它将所有指令槽的指令按(所有槽的最大延迟+Imm3)进行延迟，使得所有指令槽的流水重新对齐，并且其后的指令不会因 wait 拍数变小而被抛弃。具体行为参见 2.3.2。
B 选项存在时，表示在前面基础之上继续将所有指令槽进行延迟直到所有 BIU 指令都发射完毕。即等效于所有 BIU 指令槽均 Wait 0 来进行延迟，随后将 BIU 指令槽 Wait 还原或根据同行的 BIUWait 指令改为新值。

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_mpuWait(const int imm3, const int f_B, const int fu, const int pb)	Mfetch: MPUWait imm3 {(B)}

Example R0: WaitKI || SHU1: WaitKI;
R0: M[1] -> BIU0.T1(Mode0) || SHU1: ~T0 -> SHU1.T0(Mode0);
.....
R0: Wait 0 || SHU1: Wait 0 || **Mfetch: MPUWait 3;**

2.3.8.27. MPU Stop 指令

Syntax	SRC Mfetch: MPU.Stop
--------	------------------------

Note: 本指令无条件执行

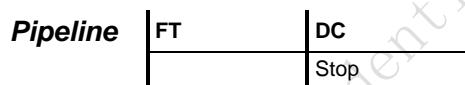
Encoding	23	22	21	20	19	18	17	16	15	14	13	12
	0	1	0	S	1	1	1	0	0	0	0	0
	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	0

Description 向量处理单元 MPU 停止指令, 执行完程序流中该条指令之前的指令, MPU 停止工作, 处于 IDLE 状态.

Note:

- 该指令可能会丢弃指令之前两周期的除 MFetch 和 SetCondition 之外的其他指令, 于是为保证其他指令能够顺利执行该指令之前需要存在至少两条 NOP 指令.
- MPU Stop 指令会清空所有 Wait 指令.

Execution



Latency 1

Intrinsics	Intrinsic	Generated Instruction
	void __ucpm2_mpuStop(const int f_S, const int fu, const int pb)	Mfetch: MPU.Stop

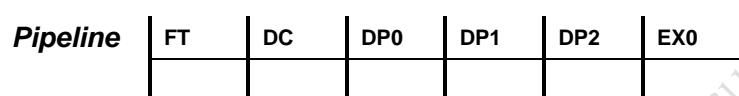
Example Mfetch: MPU.Stop;

2.3.8.28. NOP 指令

Syntax	SRC	Mfetch: NOP																																				
Encoding	23 22 21 20 19 18 17 16 15 14 13 12	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	1	0	0	0	0	1	0	0	0	0	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0																											
11	10	9	8	7	6	5	4	3	2	1	0																											
0	0	0	0	0	0	0	0	0	0	0	0																											

Description 空指令.

Execution



Latency 1

Example Mfetch: NOP;