



Netronome Network Flow Processor 6xxx

NFP SDK version 6.0

Microcode Standard Library Reference Manual

- Proprietary and Confidential -

b677.dr6112

**Product code
040-00016-004**

Netronome Network Flow Processor 6xxx: Microcode Standard Library

Reference Manual

Copyright © 2008-2014 Netronome

COPYRIGHT

No part of this publication or documentation accompanying this Product may be reproduced in any form or by any means or used to make any derivative work by any means including but not limited to by translation, transformation or adaptation without permission from Netronome Systems, Inc., as stipulated by the United States Copyright Act of 1976. Contents are subject to change without prior notice.

WARRANTY

Netronome warrants that any media on which this documentation is provided will be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of shipment. If a defect in any such media should occur during this 90-day period, the media may be returned to Netronome for a replacement.

NETRONOME DOES NOT WARRANT THAT THE DOCUMENTATION SHALL BE ERROR-FREE. THIS LIMITED WARRANTY SHALL NOT APPLY IF THE DOCUMENTATION OR MEDIA HAS BEEN (I) ALTERED OR MODIFIED; (II) SUBJECTED TO NEGLIGENCE, COMPUTER OR ELECTRICAL MALFUNCTION; OR (III) USED, ADJUSTED, OR INSTALLED OTHER THAN IN ACCORDANCE WITH INSTRUCTIONS FURNISHED BY NETRONOME OR IN AN ENVIRONMENT OTHER THAN THAT INTENDED OR RECOMMENDED BY NETRONOME.

EXCEPT FOR WARRANTIES SPECIFICALLY STATED IN THIS SECTION, NETRONOME HEREBY DISCLAIMS ALL EXPRESS OR IMPLIED WARRANTIES OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to some users of this documentation. This limited warranty gives users of this documentation specific legal rights, and users of this documentation may also have other rights which vary from jurisdiction to jurisdiction.

LIABILITY

Regardless of the form of any claim or action, Netronome's total liability to any user of this documentation for all occurrences combined, for claims, costs, damages or liability based on any cause whatsoever and arising from or in connection with this documentation shall not exceed the purchase price (without interest) paid by such user.

IN NO EVENT SHALL NETRONOME OR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THE DOCUMENTATION BE LIABLE FOR ANY LOSS OF DATA, LOSS OF PROFITS OR LOSS OF USE OF THE DOCUMENTATION OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, PUNITIVE, MULTIPLE OR OTHER DAMAGES, ARISING FROM OR IN CONNECTION WITH THE DOCUMENTATION EVEN IF NETRONOME HAS BEEN MADE AWARE OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL NETRONOME OR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THE DOCUMENTATION BE LIABLE TO ANYONE FOR ANY CLAIMS, COSTS, DAMAGES OR LIABILITIES CAUSED BY IMPROPER USE OF THE DOCUMENTATION OR USE WHERE ANY PARTY HAS SUBSTITUTED PROCEDURES NOT SPECIFIED BY NETRONOME.

Revision History

Date	Revision	Description
April 2016	004	Updated for NFP SDK 6.0 Beta 1
September 2014	003	Updated for NFP SDK 5.1
January 2014	002	Updated for NFP SDK 5.0 Beta
August 2013	001	Initial release

Table of Contents

1. Introduction	13
1.1. About This Guide	13
1.2. Related Documentation	13
2. Microcode Library	14
2.1. Aggregate Operations	14
2.1.1. Aggregate Operation Macros	14
2.1.2.	14
2.2. Big-Little-endian	18
2.2.1. Big-Little-endian Related Macros	18
2.2.2.	18
2.2.3.	19
2.3. Bitfield Operations	43
2.3.1. Bitfield Operation Macros	43
2.3.2.	43
2.3.3.	43
2.4. Byte Field Manipulation	50
2.4.1. Big / Little-endian Byte Field Extract / Compare / Branch Macros	50
2.4.2.	50
2.5. Cluster Local Scratch	61
2.5.1. Cluster Local Scratch Operation Macros	61
2.5.2.	61
2.5.3.	61
2.6. Common and Global Constants	75
2.6.1.	75
2.7. CRYPTO IPsec Operation	79
2.7.1. CRYPTO IPsec Operation Macros	79
2.7.2.	79
2.7.3.	80
2.8. CRYPTO Operation	101
2.8.1. CRYPTO Operation Macros	101
2.8.2.	102
2.8.3.	103
2.9. CRYPTO Threads Operation	120
2.9.1. CRYPTO Threads Operation Macros	120
2.9.2.	120
2.9.3.	121
2.10. DRAM Access	123
2.10.1. DRAM Access Macros	123
2.10.2.	123
2.11. Event filters and autopush API	135
2.11.1. CLS Filters and Autopush Macros	135
2.11.2.	135
2.12. Fletcher Hash Operations	137
2.12.1. Fletcher Hash Operation Macros	137
2.12.2.	137
2.13. HASH operation	140
2.13.1. HASH operation macros	140
2.13.2.	141

2.13.3.	141
2.14. Limit Operations	145
2.14.1. Limit Operation Macros	145
2.14.2.	145
2.15. Math Operations	153
2.15.1. Math Operation Macros General	153
2.15.2.	153
2.16. Memory Allocation	156
2.16.1. Memory Allocation Macros general info	156
2.16.2.	156
2.17. Memory Queue Operations	161
2.17.1. Local Memory Queue Operation Macros	161
2.17.2.	162
2.17.3.	162
2.18. Microengine CAM Operation	164
2.18.1. Microengine CAM Operation Macros	164
2.18.2.	164
2.19. Microengine CAM Sharing Operation	166
2.19.1. Microengine CAM Sharing Operation Macros	166
2.19.2.	166
2.19.3.	166
2.20. Microengine CRC	171
2.20.1. Microengine CRC Macros	171
2.20.2.	171
2.21. Microengine Standard Macros	176
2.21.1. Microengine Standard Macros	176
2.21.2.	176
2.22. Override Macros	198
2.22.1. Override flags	198
2.22.2.	198
2.23. Ring Utility	223
2.23.1. Ring Utility Macros	223
2.23.2.	224
2.24. SRAM Operation	239
2.24.1. SRAM Operation Macros	239
2.24.2.	240
2.25. Thread Synchronization	250
2.25.1. Thread Synchronization Macros	250
2.25.2.	250
2.26. XBUF Operation	252
2.26.1. XBUF Operation Macros	252
2.26.2.	252

3. Technical Support	267
-----------------------------	------------

List of Tables

1.1. Contents of this Guide	13
2.1. aggregate_zero parameters	14
2.2. aggregate_zero parameters	15
2.3. aggregate_set parameters	15
2.4. aggregate_set parameters	15
2.5. aggregate_copy parameters	16
2.6. aggregate_copy parameters	16
2.7. aggregate_directive parameters	17
2.8. aggregate_directive parameters	18
2.9. Big-Little-endian and Defines	18
2.10. swap parameters	19
2.11. swap01 parameters	20
2.12. swap12 parameters	20
2.13. swap23 parameters	21
2.14. extract0 parameters	21
2.15. extract1 parameters	22
2.16. extract2 parameters	23
2.17. extract3 parameters	23
2.18. extract01 parameters	24
2.19. extract12 parameters	25
2.20. extract23 parameters	25
2.21. extract02 parameters	26
2.22. extract13 parameters	27
2.23. extract03 parameters	27
2.24. merge_extract0 parameters	28
2.25. merge_extract1 parameters	28
2.26. merge_extract2 parameters	29
2.27. merge_extract3 parameters	29
2.28. merge_extract01 parameters	30
2.29. merge_extract12 parameters	31
2.30. merge_extract23 parameters	31
2.31. merge_extract02 parameters	32
2.32. merge_extract13 parameters	32
2.33. comp parameters	33
2.34. comp_ea_eb parameters	33
2.35. comp_ea parameters	34
2.36. comp_ea01 parameters	34
2.37. comp_ea12 parameters	34
2.38. comp_ea23 parameters	35
2.39. Bitfield Operations and Defines	43
2.40. alu_shf_left__sz1 parameters	44
2.41. alu_shf_right__sz1 parameters	45
2.42. bits_clr__sz1 parameters	45
2.43. bits_set__sz1 parameters	46
2.44. bitfield_extract__sz1 parameters	47
2.45. bitfield_clear__sz1 parameters	47
2.46. bitfield_insert__sz2 parameters	48
2.47. bitfield_or__sz1 parameters	49
2.48. bitfield_copy__sz2 parameters	49
2.49. bytefield_decr parameters	50
2.50. bytefield_incr parameters	51

2.51. bytearray_extract parameters	51
2.52. bytearray_dbl_extract parameters	52
2.53. bytearray_insert parameters	53
2.54. bytearray_select parameters	54
2.55. bytearray_br_eq parameters	54
2.56. bytearray_br_gtr parameters	55
2.57. bytearray_br_gtreq parameters	55
2.58. bytearray_br_less parameters	56
2.59. bytearray_br_lesseq parameters	56
2.60. bytearray_comp parameters	57
2.61. bytearray_shf_left_insert parameters	57
2.62. bytearray_shf_right_insert parameters	58
2.63. bytearray_clr_insert parameters	59
2.64. bytearray_clr_shf_left_insert parameters	60
2.65. bytearray_clr_shf_right_insert parameters	60
2.66. Cluster Local Scratch and Defines	61
2.67. cls_read parameters	62
2.68. cls_read_le parameters	62
2.69. cls_write parameters	63
2.70. cls_write_le parameters	64
2.71. cls_write_byte parameters	65
2.72. cls_write_byte_le parameters	66
2.73. cls_bits_clr parameters	66
2.74. cls_bits_set parameters	67
2.75. cls_bits_test_and_clr parameters	67
2.76. cls_bits_test_and_set parameters	68
2.77. cls_incr parameters	69
2.78. cls_decr parameters	69
2.79. cls_add parameters	69
2.80. cls_sub parameters	70
2.81. cls_swap parameters	71
2.82. cls_test_and_add parameters	72
2.83. cls_test_and_decr parameters	73
2.84. cls_test_and_incr parameters	73
2.85. cls_add64_immed_init parameters	74
2.86. cls_add64_immed parameters	75
2.87. cls_memset parameters	75
2.88. Common and Global Constants and Defines	75
2.89. CRYPTO IPsec Operation and Defines	79
2.90. crypto_load_ipsec_enc_pin_A parameters	80
2.91. crypto_gen_ipsec_enc_pin_A parameters	81
2.92. crypto_gen_ipsec_enc_pin_B parameters	82
2.93. crypto_gen_ipsec_enc_pin_C parameters	82
2.94. crypto_gen_ipsec_enc_pin_D parameters	83
2.95. crypto_load_ipsec_dec_pin_A parameters	84
2.96. crypto_gen_ipsec_dec_pin_A parameters	84
2.97. crypto_gen_ipsec_dec_pin_B parameters	85
2.98. crypto_gen_ipsec_dec_pin_C parameters	85
2.99. crypto_gen_ipsec_dec_pin_D parameters	86
2.100. crypto_load_ipsec_enc_strt_pin_A parameters	87
2.101. crypto_gen_ipsec_enc_strt_pin_A parameters	87
2.102. crypto_gen_ipsec_enc_strt_pin_B parameters	88
2.103. crypto_gen_ipsec_enc_strt_pin_C parameters	88

2.104. crypto_gen_ipsec_enc_strt_pin_D parameters	89
2.105. crypto_load_ipsec_enc_cont parameters	90
2.106. crypto_gen_ipsec_enc_cont parameters	90
2.107. crypto_load_ipsec_enc_end parameters	91
2.108. crypto_gen_ipsec_enc_end parameters	91
2.109. crypto_load_ipsec_dec_strt_pin_A parameters	92
2.110. crypto_gen_ipsec_dec_strt_pin_A parameters	93
2.111. crypto_gen_ipsec_dec_strt_pin_B parameters	93
2.112. crypto_gen_ipsec_dec_strt_pin_C parameters	94
2.113. crypto_gen_ipsec_dec_strt_pin_D parameters	94
2.114. crypto_load_ipsec_dec_strt_nw_pin_A parameters	95
2.115. crypto_gen_ipsec_dec_strt_nw_pin_A parameters	96
2.116. crypto_gen_ipsec_dec_strt_nw_pin_B parameters	96
2.117. crypto_gen_ipsec_dec_strt_nw_pin_C parameters	97
2.118. crypto_gen_ipsec_dec_strt_nw_pin_D parameters	97
2.119. crypto_load_ipsec_dec_cont parameters	98
2.120. crypto_gen_ipsec_dec_cont parameters	99
2.121. crypto_load_ipsec_dec_cont_nw parameters	99
2.122. crypto_gen_ipsec_dec_cont_nw parameters	100
2.123. crypto_load_ipsec_dec_end parameters	100
2.124. crypto_gen_ipsec_dec_end parameters	101
2.125. CRYPTO Operation and Defines	102
2.126. crypto_get_core parameters	103
2.127. crypto_free_core parameters	104
2.128. crypto_setup_configs parameters	104
2.129. crypto_setup_configs parameters	105
2.130. crypto_send parameters	105
2.131. crypto_build_xfer_reg parameters	107
2.132. crypto_load_library_sequence parameters	107
2.133. crypto_run_library_sequence parameters	108
2.134. crypto_get_island_and_PPID7 parameters	109
2.135. crypto_send_signal parameters	109
2.136. crypto_send_done_signal parameters	111
2.137. crypto_init_CIB_setup parameters	112
2.138. crypto_library_load_constants parameters	113
2.139. crypto_library_load parameters	115
2.140. crypto_setup_dma_desc_cycle1 parameters	116
2.141. crypto_setup_dma_desc_cycle2 parameters	117
2.142. crypto_dma_out parameters	117
2.143. crypto_dma_in parameters	118
2.144. crypto_non_dma_out parameters	119
2.145. CRYPTO Threads Operation and Defines	120
2.146. crypto_threads_input parameters	122
2.147. crypto_threads_output parameters	122
2.148. dram_mask_write parameters	124
2.149. dram_mask_write parameters	125
2.150. dram_rbuf_read parameters	125
2.151. dram_read parameters	126
2.152. dram_read parameters	127
2.153. dram_read parameters	128
2.154. dram_tbuf_write parameters	128
2.155. dram_write parameters	129
2.156. dram_write parameters	130

2.157. ddr_add64_immed_init parameters	131
2.158. ddr_add64_immed_init parameters	131
2.159. ddr_add64_immed parameters	132
2.160. ddr_add64_immed_sat parameters	132
2.161. dram_memcmp parameters	133
2.162. dram_memcmp parameters	133
2.163. dram_memset parameters	134
2.164. dram_memset parameters	134
2.165. dram_memset parameters	135
2.166. evntm_cls_event_filter_config parameters	135
2.167. evntm_cls_autopush_monitor_config parameters	136
2.168. evntm_cls_autopush_monitor_config parameters	136
2.169. evntm_cls_autopush_monitor_engage parameters	137
2.170. evntm_cls_autopush_user_event parameters	137
2.171. fletcher_hash parameters	138
2.172. jenkins_hash parameters	139
2.173. jenkins_byte_hash parameters	139
2.174. hardware_hash parameters	140
2.175. HASH operation and Defines	141
2.176. hash_init parameters	142
2.177. hash_init_cls parameters	142
2.178. hash_translate parameters	143
2.179. hash_lookup parameters	144
2.180. hash_dual_lookup parameters	145
2.181. limit_min parameters	146
2.182. limit_min parameters	146
2.183. limit_min_cc parameters	147
2.184. limit_max parameters	147
2.185. limit_max parameters	148
2.186. limit_max_cc parameters	149
2.187. limit_min_unsigned parameters	149
2.188. limit_min_unsigned parameters	150
2.189. limit_min_unsigned_cc parameters	150
2.190. limit_max_unsigned parameters	151
2.191. limit_max_unsigned parameters	152
2.192. limit_max_unsigned_cc parameters	152
2.193. limit_align_first_chunk parameters	153
2.194. math_int_div parameters	154
2.195. math_int_div_64 parameters	154
2.196. math_log2 parameters	155
2.197. math_find_highest_set parameters	155
2.198. buf_dram_addr_from_index parameters	156
2.199. buf_dram_addr_from_sram_addr parameters	157
2.200. buf_index_from_dram_addr parameters	158
2.201. buf_index_from_sram_addr parameters	158
2.202. buf_sram_addr_from_index parameters	159
2.203. buf_sram_addr_from_dram_addr parameters	159
2.204. buf_freelist_create parameters	160
2.205. buf_alloc parameters	161
2.206. buf_free parameters	161
2.207. Memory Queue Operations and Defines	162
2.208. lm_handle_alloc parameters	162
2.209. lm_handle_free parameters	163

2.210. lm_handle_verify parameters	163
2.211. incr_lm_base parameters	164
2.212. cam_read_entry parameters	164
2.213. cam_write_entry parameters	165
2.214. cam_read_data parameters	165
2.215. cam_match parameters	166
2.216. Microengine CAM Sharing Operation and Defines	166
2.217. cam_entry_read_state parameters	167
2.218. cam_entry_write_state parameters	167
2.219. cam_entry_read_tag parameters	168
2.220. cam_clearall parameters	169
2.221. cam_entry_write parameters	169
2.222. cam_exit_using_entry parameters	169
2.223. cam_entry_lookup parameters	170
2.224. cam_entry_lookup_with_lm parameters	171
2.225. crc_load_crc10_table parameters	172
2.226. crc_10 parameters	172
2.227. crc_32 parameters	173
2.228. crc_ccitt parameters	174
2.229. crc_read parameters	175
2.230. crc_write parameters	175
2.231. immed32 parameters	176
2.232. immed40 parameters	177
2.233. balr parameters	177
2.234. move parameters	178
2.235. alu_op parameters	178
2.236. add parameters	179
2.237. add_c parameters	180
2.238. sub parameters	180
2.239. shf_right parameters	181
2.240. shf_left parameters	181
2.241. rot_right parameters	182
2.242. rot_left parameters	182
2.243. alu_shf_right parameters	183
2.244. add_shf_right parameters	184
2.245. sub_shf_right parameters	184
2.246. and_shf_right parameters	185
2.247. or_shf_right parameters	185
2.248. alu_shf_left parameters	186
2.249. add_shf_left parameters	187
2.250. sub_shf_left parameters	187
2.251. and_shf_left parameters	188
2.252. or_shf_left parameters	188
2.253. alu_rot_right parameters	189
2.254. alu_rot_left parameters	190
2.255. bitfield_extract parameters	191
2.256. bitfield_insert parameters	191
2.257. bits_clr parameters	192
2.258. bits_set parameters	192
2.259. multiply parameters	193
2.260. divide parameters	194
2.261. array_index_from_elem_addr parameters	195
2.262. elem_addr_from_array_index parameters	195

2.263. arith_shf_right parameters	196
2.264. multiply32 parameters	196
2.265. multiply64 parameters	197
2.266. rand parameters	197
2.267. srand parameters	198
2.268. ov_global_flags parameters	198
2.269. ov_start parameters	199
2.270. ov_set_bits parameters	200
2.271. ov_set_bits parameters	201
2.272. ov_set parameters	202
2.273. ov_set parameters	202
2.274. ov_set_bm_and parameters	203
2.275. ov_set_bm_and parameters	204
2.276. ov_set_extract parameters	205
2.277. ov_set_extract parameters	206
2.278. ov_set_bm_and_extract parameters	208
2.279. ov_set_bm_and_extract parameters	209
2.280. ov_set_use parameters	210
2.281. ov_set_use parameters	211
2.282. ov_set_bm_and_use parameters	212
2.283. ov_set_bm_and_use parameters	213
2.284. ov_set_extract_use parameters	214
2.285. ov_set_extract_use parameters	215
2.286. ov_set_bm_and_extract_use parameters	216
2.287. ov_set_bm_and_extract_use parameters	217
2.288. ov_use parameters	218
2.289. ov_single parameters	219
2.290. ov_recall parameters	221
2.291. ov_resume parameters	222
2.292. ov_dispose parameters	223
2.293. ru_emem_ring_setup parameters	224
2.294. ru_emem_ring_setup parameters	224
2.295. ru_emem_ring_put parameters	225
2.296. ru_emem_ring_get parameters	226
2.297. ru_nn_ring_init parameters	226
2.298. ru_dram_ring_setup parameters	227
2.299. ru_dram_ring_setup parameters	227
2.300. ru_sram_ring_setup parameters	228
2.301. ru_cls_ring_setup parameters	228
2.302. ru_ctm_ring_setup parameters	228
2.303. ru_ring_op parameters	229
2.304. ru_sram_ring_put parameters	230
2.305. ru_dram_ring_put parameters	231
2.306. ru_deq_from_ring parameters	235
2.307. ru_deq_from_ring parameters	236
2.308. ru_cls_ring_put parameters	236
2.309. ru_ctm_ring_put parameters	238
2.310. ru_ctm_ring_get parameters	238
2.311. ru_enq_to_ring parameters	238
2.312. ru_enq_to_ring parameters	239
2.313. sram_read parameters	240
2.314. sram_write parameters	241
2.315. sram_bits_clr parameters	241

2.316. sram_bits_set parameters	242
2.317. sram_bits_test_and_clr parameters	243
2.318. sram_bits_test_and_set parameters	243
2.319. sram_add parameters	244
2.320. sram_decr parameters	244
2.321. sram_incr parameters	245
2.322. sram_sub parameters	245
2.323. sram_swap parameters	246
2.324. sram_test_and_add parameters	247
2.325. sram_test_and_decr parameters	248
2.326. sram_test_and_incr parameters	249
2.327. sram_fast_journal parameters	249
2.328. sram_wr_qdesc parameters	249
2.329. sram_memset parameters	250
2.330. threads_reorder_once parameters	251
2.331. threads_br_ctx parameters	251
2.332. threads_kill parameters	252
2.333. threads_order parameters	252
2.334. xbuf_bind_address parameters	253
2.335. xbuf_deactivate parameters	254
2.336. xbuf_activate parameters	255
2.337. xbuf_alloc parameters	257
2.338. xbuf_free parameters	257
2.339. xbuf_link parameters	258
2.340. xbuf_find parameters	259
2.341. xbuf_param_set parameters	260
2.342. xbuf_xfer_set parameters	261
2.343. xbuf_extract parameters	262
2.344. xbuf_extract_frm_linked_bufs parameters	263
2.345. xbuf_type_extract parameters	263
2.346. xbuf_insert parameters	264
2.347. xbuf_copy parameters	265

1. Introduction

1.1 About This Guide

This document provides a reference to the Microcode Standard Library functions that supports the Netronome Network Flow Processor NFP-6xxx product line.



Note

For simplicity throughout this document, the compiler will be referred to as the *C compiler*, or in some cases simply *the compiler*. Also, the Netronome NFP-6xxx network processor may be referred to as *the network processor*.

Table 1.1. Contents of this Guide

Chapter	Description
Chapter 1	Description of this guide, related documentation, and table of conventions used.
Chapter 2	Microcode Standard Library functions.

1.2 Related Documentation

Further information related to the Netronome Systems NFP-32XX product line is located in:

- *Netronome Network Flow Processor 6xxx: Datasheet*
- *Netronome Network Flow Processor 6xxx: Development Tools User's Guide*
- *Netronome Network Flow Processor 6xxx: Network Flow Assembler System User's Guide*
- *Netronome Network Flow Processor 6xxx: Databook*
- *Netronome Network Flow Processor 6xxx: Network Flow C Compiler User's Guide*
- *Netronome Network Flow Processor 6xxx: Microengine Programmer's Reference Manual*
- *Netronome Network Flow Processor 6xxx: Micro-C Standard Library Reference Manual*

2. Microcode Library

2.1 Aggregate Operations

2.1.1 Aggregate Operation Macros

These macros perform operations on groups of registers

2.1.2.1 `aggregate_zero`

Prototype:

```
aggregate_zero(dst, COUNT)
```

Description:

Zero Aggregate.

Example:

```
.reg write $stat_wr[16]  
aggregate_zero($stat_wr, 16)
```

Table 2.1. `aggregate_zero` parameters

Name	Description
<i>dst</i>	Destination GPRs or write transfer registers to zero
<i>COUNT</i>	CONSTANT number of registers to set to zero

2.1.2.2 `aggregate_zero`

Prototype:

```
aggregate_zero(dst, DST_IDX, COUNT)
```

Description:

Zero Aggregate using Index.

Example:

```
.reg write $stat_wr[16]  
aggregate_zero($stat_wr, 8, 2)
```

Table 2.2. aggregate_zero parameters

Name	Description
<i>dst</i>	Destination GPRs or write transfer registers to zero
<i>DST_IDX</i>	Index of 1st register to zero
<i>COUNT</i>	CONSTANT number of registers to set to zero

2.1.2.3 aggregate_set

Prototype:

```
aggregate_set(dst, scalar_src, COUNT)
```

Description:

Set value to Aggregate.

Example:

```
.reg write $stat_wr[16]
aggregate_set($stat_wr, 0x55, 16)
```

Table 2.3. aggregate_set parameters

Name	Description
<i>dst</i>	Destination GPRs or write transfer registers to set to value
<i>scalar_src</i>	value to set to registers
<i>COUNT</i>	CONSTANT number of registers to set to value

2.1.2.4 aggregate_set

Prototype:

```
aggregate_set(dst, DST_IDX, scalar_src, COUNT)
```

Description:

Set value to Aggregate using Index.

Example:

```
.reg write $stat_wr[16]
aggregate_set($stat_wr, 0x55, 16)
```

Table 2.4. aggregate_set parameters

Name	Description
<i>dst</i>	Destination GPRs or write transfer registers to set to value

Name	Description
<i>DST_IDX</i>	Index of 1st register to set value to
<i>scalar_src</i>	value to set to registers
<i>COUNT</i>	CONSTANT number of registers to set to value

2.1.2.5 aggregate_copy

Prototype:

```
aggregate_copy(dst, src, COUNT)
```

Description:

Copy from Aggregate source to Aggregate destination.

Example:

```
.reg $src[8]
.reg $dst[8]
aggregate_copy($dst, $src, 8)
```

Table 2.5. aggregate_copy parameters

Name	Description
<i>dst</i>	Destination GPRs or write transfer registers for copy
<i>src</i>	Source GPRs or read transfer registers for copy
<i>COUNT</i>	CONSTANT number of registers to copy

2.1.2.6 aggregate_copy

Prototype:

```
aggregate_copy(dst, DST_IDX, src, SRC_IDX, COUNT)
```

Description:

Copy from Aggregate source to Aggregate destination using indices.

Example:

```
.reg $src[8]
.reg $dst[8]
aggregate_copy($dst, 2, $src, 4, 2)
```

Table 2.6. aggregate_copy parameters

Name	Description
<i>dst</i>	Destination GPRs or write transfer registers for copy

Name	Description
<i>DST_IDX</i>	Index of 1st destination register to copy to
<i>src</i>	Source GPRs or read transfer registers for copy
<i>SRC_IDX</i>	Index of 1st source register to copy from
<i>COUNT</i>	CONSTANT number of registers to copy

2.1.2.7 aggregate_directive

Prototype:

```
aggregate_directive(DIRECTIVE, xfer, COUNT)
```

Description:

Generate Assembler Directive for a Register Aggregate.

Example:

```
.reg $xfr[8]
aggregate_directive(.set, $xfr, 8)
```

Table 2.7. aggregate_directive parameters

Name	Description
<i>DIRECTIVE</i>	directive string to generate on transfer registers one of .set, .set_wr, .set_rd, .use, .use_wr, .use_rd, .init
<i>xfer</i>	Transfer registers on which to generate directive
<i>COUNT</i>	CONSTANT number of registers to generate directive on

2.1.2.8 aggregate_directive

Prototype:

```
aggregate_directive(DIRECTIVE, xfer, START, COUNT)
```

Description:

Generate Assembler Directive for a Register Aggregate using index.

Example:

```
.reg $xfr[8]
aggregate_directive(.set, $xfr, 4, 2)
```

Table 2.8. aggregate_directive parameters

Name	Description
<i>DIRECTIVE</i>	directive string to generate on transfer registers one of .set, .set_wr, .set_rd, .use, .use_wr, .use_rd, .init
<i>xfer</i>	Transfer registers on which to generate directive
<i>START</i>	START index of 1st transfer register to generate directive on
<i>COUNT</i>	CONSTANT number of registers to generate directive on

2.2 Big-Little-endian

2.2.1 Big-Little-endian Related Macros

Endian independent internal macros supporting bytefield.uc.Default is BIG_ENDIAN. LITTLE_ENDIAN is not yet supported.These macros perform swaps on operators and stores the result according to the following naming convention: swap ::= e no_character ::= result_op ::= e | <no_character> all_bytes ::= <no_character> left_to_right_bytes ::= 01 | 12 | 23 | <all_bytes> function ::= add | comp aoperand ::= _ea<left_to_right_bytes> | <no_character> boperand ::= _eb<left_to_right_bytes> | <no_character> prev_carry ::= _c | <no_character> unary operator = <result_op><function><left_to_right_bytes> operator = <result_op><function><aoperand><boperand><prev_carry>

Table 2.9. Big-Little-endian and Defines

Defined	Definition
swap32	swap01 Alias for swap01.
swap21	swap12 Alias for swap12.
swap10	swap23 Alias for swap23.
comp_ea32	comp_ea01 Alias for comp_ea01.
comp_ea21	comp_ea12 Alias for comp_ea12.
comp_ea10	comp_ea23 Alias for comp_ea23.

Defined	Definition
add_ea32_eb	add_ea01_eb Alias for add_ea01_eb.
add_ea10_eb	add_ea23_eb Alias for add_ea23_eb.
add_ea10_c	add_ea23_c Alias for add_ea23_c.
add_ea32_c	add_ea01_c Alias for add_ea01_c.

2.2.3.1 swap

Prototype:

```
swap(res, aop, in_load_cc)
```

Description:

Four byte endian swap.

Example:

```
aop = 0xAABBCCDD
res = 0xDDCCBBAA
```

Table 2.10. swap parameters

Name	Description
<i>res</i>	Result register
<i>aop</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.2 swap01

Prototype:

```
swap01(res, aop, in_load_cc)
```

Description:

Two byte endian swap of two left-most bytes.

Example:

```
aop = 0xAABBCCDD
res = 0x0000BBAA
```

Table 2.11. swap01 parameters

Name	Description
<i>res</i>	Result register
<i>aop</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.3 swap12

Prototype:

```
swap12(res, aop, in_load_cc)
```

Description:

Two byte endian swap of two middle bytes.

Example:

```
aop = 0xAABBCCDD
res = 0x0000CCBB
```

Table 2.12. swap12 parameters

Name	Description
<i>res</i>	Result register
<i>aop</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.4 swap23

Prototype:

```
swap23(res, aop, in_load_cc)
```

Description:

Two byte endian swap of two right-most bytes.

Example:

```
aop = 0xAABBCCDD
res = 0x0000DDCC
```

Table 2.13. swap23 parameters

Name	Description
<i>res</i>	Result register
<i>aop</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.5 extract0

Prototype:

```
extract0(dest, source, shift_amt, in_load_cc)
```

Description:

Extract byte 0 from *source*, store in cleared *dest* at bit *shift_amt*.

Example:

```
src = 0xAABBCCDD

extract0(out, src, 24, NO_LOAD_CC) // out == 0xAA000000
extract0(out, src, 8, NO_LOAD_CC) // out == 0x0000AA00
```

Table 2.14. extract0 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value

Name	Description
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0, 8, 16 or 24
<i>in_load_cc</i>	CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are: <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.6 extract1

Prototype:

```
extract1(dest, source, shift_amt, in_load_cc)
```

Description:

Extract byte 1 from *source*, store in cleared *dest* at bit *shift_amt*.

Example:

```
src = 0xAABBCCDD

extract1(out, src, 24, NO_LOAD_CC)    // out == 0xBB000000
extract1(out, src, 8, NO_LOAD_CC)     // out == 0x0000BB00
```

Table 2.15. extract1 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0, 8, 16 or 24
<i>in_load_cc</i>	CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are: <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.7 extract2

Prototype:

```
extract2(dest, source, shift_amt, in_load_cc)
```

Description:

Extract byte 2 from *source*, store in cleared *dest* at bit *shift_amt*.

Example:

```
src = 0xAABBCCDD

extract2(out, src, 24, NO_LOAD_CC)    // out == 0xCC000000
extract2(out, src, 8, NO_LOAD_CC)     // out == 0x0000CC00
```

Table 2.16. extract2 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0, 8, 16 or 24
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.8 extract3

Prototype:

```
extract3(dest, source, shift_amt, in_load_cc)
```

Description:

Extract byte 3 from *source*, store in cleared *dest* at bit *shift_amt*.

Example:

```
src = 0xAABBCCDD

extract3(out, src, 24, NO_LOAD_CC)    // out == 0xDD000000
extract3(out, src, 8, NO_LOAD_CC)     // out == 0x0000DD00
```

Table 2.17. extract3 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0, 8, 16 or 24

Name	Description
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> • NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect • DO_LOAD_CC: Load condition code

2.2.3.9 extract01

Prototype:

```
extract01(dest, source, shift_amt, in_load_cc)
```

Description:

Extract bytes 0 and 1 from *source*, store in cleared *dest* at bit *shift_amt*.

Example:

```
src = 0xAABBCCDD

extract01(out, src, 16, NO_LOAD_CC) // out == 0xAABB0000
extract01(out, src, 8, NO_LOAD_CC) // out == 0x00AABB00
extract01(out, src, 0, NO_LOAD_CC) // out == 0x0000AABB
```

Table 2.18. extract01 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0, 8 or 16
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> • NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect • DO_LOAD_CC: Load condition code

2.2.3.10 extract12

Prototype:

```
extract12(dest, source, shift_amt, in_load_cc)
```

Description:

Extract bytes 1 and 2 from `source`, store in cleared `dest` at bit `shift_amt`.

Example:

```
src = 0xAABBCCDD

extract12(out, src, 16, NO_LOAD_CC) // out == 0xBBCC0000
extract12(out, src, 8, NO_LOAD_CC)  // out == 0x00BBCC00
extract12(out, src, 0, NO_LOAD_CC)  // out == 0x0000BBCC
```

Table 2.19. extract12 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0, 8 or 16
<i>in_load_cc</i>	CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are: <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.11 extract23

Prototype:

```
extract23(dest, source, shift_amt, in_load_cc)
```

Description:

Extract bytes 2 and 3 from `source`, store in cleared `dest` at bit `shift_amt`.

Example:

```
src = 0xAABBCCDD

extract23(out, src, 16, NO_LOAD_CC) // out == 0xCCDD0000
extract23(out, src, 8, NO_LOAD_CC)  // out == 0x00CCDD00
extract23(out, src, 0, NO_LOAD_CC)  // out == 0x0000CCDD
```

Table 2.20. extract23 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0, 8 or 16

Name	Description
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.12 extract02

Prototype:

```
extract02(dest, source, shift_amt, in_load_cc)
```

Description:

Extract bytes 0 to 2 from *source*, store in cleared *dest* at bit *shift_amt*.

Example:

```
src = 0xAABBCCDD

extract02(out, src, 8, NO_LOAD_CC)    // out == 0xAABBCC00
extract02(out, src, 0, NO_LOAD_CC)    // out == 0x00AABBCC
```

Table 2.21. extract02 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0 or 8
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.13 extract13

Prototype:

```
extract13(dest, source, shift_amt, in_load_cc)
```

Description:

Extract bytes 1 to 3 from *source*, store in cleared *dest* at bit *shift_amt*.

Example:

```
src = 0xAABBCCDD

extract13(out, src, 8, NO_LOAD_CC)    // out == 0xBBCCDD00
extract13(out, src, 0, NO_LOAD_CC)    // out == 0x00BBCCDD
```

Table 2.22. extract13 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>shift_amt</i>	Left shift count of the extracted byte. Must be 0 or 8
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.14 extract03

Prototype:

```
extract03(dest, source, dummySHIFT, in_load_cc)
```

Description:

Extract bytes 0 to 3 from *source*, store in cleared *dest* at bit *shift_amt*.

This function is mainly provided for generic coding in other modules.

Table 2.23. extract03 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>dummySHIFT</i>	Ignored
<i>in_load_cc</i>	Ignored, macro behaves as if DO_LOAD_CC is used

2.2.3.15 merge_extract0

Prototype:

```
merge_extract0(dest, source, in_load_cc)
```

Description:

Extract byte 0 from *source*, store in *dest* at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract0(out, src, NO_LOAD_CC)    // out == 0x112233AA
```

Table 2.24. merge_extract0 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.16 merge_extract1

Prototype:

```
merge_extract1(dest, source, in_load_cc)
```

Description:

Extract byte 1 from *source*, store in *dest* at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract1(out, src, NO_LOAD_CC)    // out == 0x112233BB
```

Table 2.25. merge_extract1 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.17 merge_extract2

Prototype:

```
merge_extract2(dest, source, in_load_cc)
```

Description:

Extract byte 2 from `source`, store in `dest` at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract2(out, src, NO_LOAD_CC)    // out == 0x112233CC
```

Table 2.26. merge_extract2 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.18 merge_extract3

Prototype:

```
merge_extract3(dest, source, in_load_cc)
```

Description:

Extract byte 3 from `source`, store in `dest` at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract3(out, src, NO_LOAD_CC)    // out == 0x112233DD
```

Table 2.27. merge_extract3 parameters

Name	Description
<i>dest</i>	Destination register

Name	Description
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.19 merge_extract01

Prototype:

```
merge_extract01(dest, source, in_load_cc)
```

Description:

Extract byte 0 and 1 from *source*, store in *dest* at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract01(out, src, NO_LOAD_CC) // out == 0x1122AABB
```

Table 2.28. merge_extract01 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.20 merge_extract12

Prototype:

```
merge_extract12(dest, source, in_load_cc)
```

Description:

Extract byte 1 and 2 from *source*, store in *dest* at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract12(out, src, NO_LOAD_CC) // out == 0x1122BBCC
```

Table 2.29. merge_extract12 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.21 merge_extract23

Prototype:

```
merge_extract23(dest, source, in_load_cc)
```

Description:

Extract byte 2 and 3 from *source*, store in *dest* at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract23(out, src, NO_LOAD_CC) // out == 0x1122CCDD
```

Table 2.30. merge_extract23 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.22 merge_extract02

Prototype:

```
merge_extract02(dest, source, in_load_cc)
```

Description:

Extract byte 0 to 2 from `source`, store in `dest` at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract02(out, src, NO_LOAD_CC) // out == 0x11AABBCC
```

Table 2.31. merge_extract02 parameters

Name	Description
<i>dest</i>	Destination register
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.23 merge_extract13

Prototype:

```
merge_extract13(dest, source, in_load_cc)
```

Description:

Extract byte 1 to 3 from `source`, store in `dest` at bit 0.

Example:

```
src = 0xAABBCCDD
out = 0x11223344

merge_extract13(out, src, NO_LOAD_CC) // out == 0x11BBCCDD
```

Table 2.32. merge_extract13 parameters

Name	Description
<i>dest</i>	Destination register

Name	Description
<i>source</i>	Input value
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.2.3.24 comp

Prototype:

`comp(aop, bop)`

Description:

Compare *aop* to *bop*.

Table 2.33. comp parameters

Name	Description
<i>aop</i>	Operand A
<i>bop</i>	Operand B

2.2.3.25 comp_ea_eb

Prototype:

`comp_ea_eb(aop, bop)`

Description:

Compare 4 byte endian swapped *aop* to 4 byte endian swapped *bop*.

Table 2.34. comp_ea_eb parameters

Name	Description
<i>aop</i>	Operand A
<i>bop</i>	Operand B

2.2.3.26 comp_ea

Prototype:

`comp_ea(aop, bop)`

Description:

Compare 4 byte endian swapped *aop* to *bop*.

Table 2.35. comp_ea parameters

Name	Description
<i>aop</i>	Operand A
<i>bop</i>	Operand B

2.2.3.27 comp_ea01

Prototype:

`comp_ea01(aop, bop)`

Description:

16 bit big-endian compare *aop* to big-endian *bop*, compare left 2 bytes of *aop* to *bop*.

Table 2.36. comp_ea01 parameters

Name	Description
<i>aop</i>	Operand A
<i>bop</i>	Operand B

2.2.3.28 comp_ea12

Prototype:

`comp_ea12(aop, bop)`

Description:

16 bit big-endian compare *aop* to big-endian *bop*, compare middle 2 bytes of *aop* to *bop*.

Table 2.37. comp_ea12 parameters

Name	Description
<i>aop</i>	Operand A
<i>bop</i>	Operand B

2.2.3.29 comp_ea23

Prototype:

`comp_ea23(aop, bop)`

Description:

16 bit big-endian compare aop to big-endian bop, compare right 2 bytes of aop to bop.

Table 2.38. comp_ea23 parameters

Name	Description
<i>aop</i>	Operand A
<i>bop</i>	Operand B

2.2.3.30 eadd_ea_eb

Prototype:

```
eadd_ea_eb(res, aop, bop)
```

Description:

4 byte endian add of swapped aop and swapped bop, swapped result.

Executes:

```
res = swap( swap(aop) + swap(bop) )
```

2.2.3.31 add_ea_eb

Prototype:

```
add_ea_eb(res, aop, bop)
```

Description:

4 byte endian add of swapped aop and swapped bop.

Executes:

```
res = swap(aop) + swap(bop)
```

2.2.3.32 eadd_ea

Prototype:

```
eadd_ea(res, aop, bop)
```

Description:

4 byte endian add of swapped aop and bop, swapped result.

Executes:

```
res = swap( swap(aop) + bop )
```



Note

bop can be the same register as res.

2.2.3.33 add_ea

Prototype:

```
add_ea(res, aop, bop)
```

Description:

4 byte endian add of swapped aop and bop.

Executes:

```
res = swap(aop) + bop
```



Note

bop can be the same register as res.

2.2.3.34 add_ea01_eb

Prototype:

```
add_ea01_eb(res, aop, bop)
```

Description:

Swap(left 2 bytes of aop) + swap(bop).

Executes:

```
res = swap01(aop) + swap(bop)
```

2.2.3.35 add_ea23_eb

Prototype:

```
add_ea23_eb(res, aop, bop)
```

Description:

Swap(right 2 bytes aop) + swap(bop).

Executes:

```
res = swap23(aop) + swap(bop)
```

2.2.3.36 add_ea_c

Prototype:

```
add_ea_c(res, aop, bop)
```

Description:

4 byte endian carry add of swapped aop and bop.

Executes:

```
res = swap(aop) + bop + previous carry
```



Note

bop can be the same register as res.

2.2.3.37 add_ea23_c

Prototype:

```
add_ea23_c(res, aop, bop)
```

Description:

4 byte endian carry add of swap(right 2 bytes aop) and bop.

Executes:

```
res = swap23(aop) + bop + previous carry
```



Note

bop can be the same register as res.

2.2.3.38 add_ea01_c

Prototype:

```
add_ea01_c(res, aop, bop)
```

Description:

4 byte endian carry add of swap(left 2 bytes aop) and bop.

Executes:

```
res = swap01(aop) + bop + previous carry
```



Note

bop can be the same register as res.

2.2.3.39 incr0

Prototype:

```
incr0(dest, source)
```

Description:

Increment byte 0, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.40 incr1

Prototype:

```
incr1(dest, source)
```

Description:

Increment byte 1, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.41 incr2

Prototype:

```
incr2(dest, source)
```

Description:

Increment byte 2, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.42 incr3

Prototype:

```
incr3(dest, source)
```

Description:

Increment byte 3, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.43 incr01

Prototype:

```
incr01(dest, source)
```

Description:

Increment bytes 0-1.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.44 incr12

Prototype:

```
incr12(dest, source)
```

Description:

Increment bytes 1-2, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.45 incr23

Prototype:

```
incr23(dest, source)
```

Description:

Increment bytes 2-3, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.46 incr02

Prototype:

```
incr02(dest, source)
```

Description:

Increment bytes 0-2.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.47 incr13

Prototype:

```
incr13(dest, source)
```

Description:

Increment bytes 1-3, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.48 incr03

Prototype:

```
incr03(dest, source)
```

Description:

Increment bytes 0-3.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.49 decr0

Prototype:

```
decr0(dest, source)
```

Description:

Decrement byte 0, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.50 decr1

Prototype:

```
decr1(dest, source)
```

Description:

Decrement byte 1, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.51 decr2

Prototype:

```
decr2(dest, source)
```

Description:

Decrement byte 2, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.52 decr3

Prototype:

```
decr3(dest, source)
```

Description:

Decrement byte 3, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.53 decr01

Prototype:

```
decr01(dest, source)
```

Description:

Decrement byte 0-1, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.54 decr12

Prototype:

```
decr12(dest, source)
```

Description:

Decrement byte 1-2, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.55 decr23

Prototype:

```
decr23(dest, source)
```

Description:

Decrement byte 2-3, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.56 decr02

Prototype:

```
decr02(dest, source)
```

Description:

Decrement byte 0-2, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.57 decr13

Prototype:

```
decr13(dest, source)
```

Description:

Decrement byte 1-3, other bytes are unaffected.

Source can be a read transfer register. Destination can be a write transfer register.

2.2.3.58 decr03

Prototype:

```
decr03(dest, source)
```

Description:

Decrement byte 0-3.

Source can be a read transfer register. Destination can be a write transfer register.

2.3 Bitfield Operations

2.3.1 Bitfield Operation Macros

These macros perform bitfield operations using minimum number of instructions

Table 2.39. Bitfield Operations and Defines

Defined	Definition
BF_AML(a,w,m,l)	a[w], m, l
BF_AL(a,w,m,l)	a[w], l
BF_A(a,w,m,l)	a[w]
BF_ML(w,m,l)	m, l
BF_W(w,m,l)	w
BF_M(w,m,l)	m
BF_L(w,m,l)	l
BF_WIDTH(w,m,l)	(m + 1 - l)
_bitfield_insert	bitfield_insert__sz2
_bitfield_clear	bitfield_clear__sz1
_bitfield_or	bitfield_or__sz1
_bitfield_copy	bitfield_copy__sz2
F_AML	BF_AML
F_AL	BF_AL
F_A	BF_A
F_ML	BF_ML
F_W	BF_W
F_M	BF_M
F_L	BF_L

2.3.3.1 alu_shf_left__sz1

Prototype:

```
alu_shf_left__sz1(out_result, in_a, op_spec, in_b, in_shift_amt)
```

Description:

Shift left in_b by in_shift_amt bit positions, then perform operation op_spec on in_a.

Example:

```
.reg output, input
immed[input, 0x05]
alu_shf_left__sz1(output, 0x02, OR, input, 4)
```

Limited to restricted operands and single 8 bit constant parameter or 2 GPRs Use `alu_shf_left()` instead for reduced limitations

Instruction Count: 1

Table 2.40. `alu_shf_left__sz1` parameters

Name	Description
<i>out_result</i>	Destination
<i>in_a</i>	Register or constant
<i>op_spec</i>	ALU operation Legal operators are: <ul style="list-style-type: none"> • B • ~B • AND • ~AND • AND~ • OR
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Constant (0 to 31)

2.3.3.2 `alu_shf_right__sz1`

Prototype:

```
alu_shf_right__sz1(out_result, in_a, op_spec, in_b, in_shift_amt)
```

Description:

Shift right `in_b` by `in_shift_amt` bit positions, then perform operation `op_spec` on `in_a`.

Example:

```
.reg output, input
immed[input, 0x50]
alu_shf_right__sz1(output, 0x20, OR, input, 4)
```

Limited to restricted operands and single 8 bit constant parameter or 2 GPRs Use `alu_shf_left()` instead for more flexibility

Instruction Count: 1

Table 2.41. alu_shf_right__sz1 parameters

Name	Description
<i>out_result</i>	Destination
<i>in_a</i>	Register or constant
<i>op_spec</i>	ALU operation Legal operators are: <ul style="list-style-type: none"> • B • ~B • AND • ~AND • AND~ • OR
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Constant (0 to 31)

2.3.3.3 bits_clr__sz1

Prototype:

```
bits_clr__sz1(io_data, in_start_bit_pos, in_mask)
```

Description:

Clear bits indicated by mask at starting position *in_start_bit_pos*.

Example:

```
.reg reg2, bitpos
bits_clr__sz1(reg2, bitpos, 0x3)
```

Limited to restricted operands, use `bits_clr()` instead for more flexibility

Instruction Count: 1

Table 2.42. bits_clr__sz1 parameters

Name	Description
<i>io_data</i>	Register to modify
<i>in_start_bit_pos</i>	Constant less than 32, starting bit position <i>in_mask</i> is shifted left by this amount
<i>in_mask</i>	Register or constant, mask of bits to clear

2.3.3.4 bits_set__sz1

Prototype:

```
bits_set__sz1(io_data, in_start_bit_pos, in_mask)
```

Description:

Set bits indicated by mask at starting position `in_start_bit_pos`.

Example:

```
.reg reg2, bitpos
bits_set__sz1(reg2, bitpos, 0x3)
```

Limited to restricted operands, use `bits_set()` instead for more flexibility

Instruction Count: 1

Table 2.43. `bits_set__sz1` parameters

Name	Description
<code>io_data</code>	Register to modify
<code>in_start_bit_pos</code>	Constant less than 32, starting bit position <code>in_mask</code> is shifted left by this amount
<code>in_mask</code>	Register or constant, mask of bits to set

2.3.3.5 `bitfield_extract__sz1`

Prototype:

```
bitfield_extract__sz1(out_result, in_src, MSB, LSB)
```

Description:

Extract a bit field from a register.

Example:

```
.reg r_out, r_in
bitfield_extract__sz1(r_out, r_in, 15, 8)
```



Note

Bits are numbered 31-0, left to right.

Limited to restricted operands in some cases, as well as certain combinations of MSB/LSB. Use `bitfield_extract()` instead for more flexibility

Instruction Count: 1

Table 2.44. bitfield_extract__sz1 parameters

Name	Description
<i>out_result</i>	Extracted field
<i>in_src</i>	Source register with multiple fields
<i>MSB</i>	Most significant, left bit defining field
<i>LSB</i>	Least significant, right bit defining field

2.3.3.6 bitfield_clear__sz1

Prototype:

```
bitfield_clear__sz1(io_data, MSB, LSB)
```

Description:

Clear a bit field in a register.

Example:

```
.reg r_val
bitfield_clear__sz1(r_val, 15, 8)
```



Note

Bits are numbered 31-0, left to right.

Limited to restricted operands. Use bits_clr instead for more flexibility

Instruction Count: 1

Table 2.45. bitfield_clear__sz1 parameters

Name	Description
<i>io_data</i>	Register to clear
<i>MSB</i>	Constant, most significant, left bit defining field
<i>LSB</i>	Constant, least significant, right bit defining field

2.3.3.7 bitfield_insert__sz2

Prototype:

```
bitfield_insert__sz2(io_data, MSB, LSB, in_mask)
```

Description:

Clear a bit field in a register, then inserts in_mask.

Example:

```
.reg r_val
bitfield_insert__sz2(r_val, 15, 8, 0x23)
```



Note

Bits are numbered 31-0, left to right.

Limited to restricted operands. Use `bitfield_insert` instead for more flexibility

Instruction Count: 2

Table 2.46. `bitfield_insert__sz2` parameters

Name	Description
<i>io_data</i>	Register to insert value
<i>MSB</i>	Constant, most significant, left bit defining field
<i>LSB</i>	Constant, least significant, right bit defining field
<i>in_mask</i>	Register or constant, bit mask to insert

2.3.3.8 `bitfield_insert__sz1`

Prototype:

```
bitfield_insert__sz1(io_data, MSB, LSB, in_mask)
```

Description:

Retain deprecated aliases for backwards compatibility, but with compiler warnings.

2.3.3.9 `bitfield_or__sz1`

Prototype:

```
bitfield_or__sz1(io_data, LSB, in_mask)
```

Description:

Logical Or bits in an input field into a register.

Example:

```
.reg r_val
bitfield_or__sz1(r_val, 8, 0x23)
```




Note

Bits are numbered 31-0, left to right.

Limited to restricted operands. Use `bitfield_insert` instead for more flexibility

Instruction Count: 1

Table 2.47. `bitfield_or__sz1` parameters

Name	Description
<i>io_data</i>	Register in which to logical or <code>in_mask</code>
<i>LSB</i>	Constant, least significant, right bit defining field
<i>in_mask</i>	Register or constant, bit mask to logical or into <code>io_data</code>

2.3.3.10 `bitfield_copy__sz2`

Prototype:

```
bitfield_copy__sz2(out_reg, OUT_LSB, in_reg, IN_MSB, IN_LSB)
```

Description:

Logical Or a bitfield from one register into another register.

Example:

```
.reg r_val
bitfield_copy__sz2(r_val, 8, 0x23, 15, 0)
```



Note

Bits are numbered 31-0, left to right.

Limited to restricted operands.

Instruction Count: 2

Table 2.48. `bitfield_copy__sz2` parameters

Name	Description
<i>out_reg</i>	Register in which to logical or the bitfield from <code>in_reg</code>
<i>OUT_LSB</i>	Constant, least significant, right bit defining field in <code>out_reg</code>
<i>in_reg</i>	Register containing bit field to extract and insert into <code>out_reg</code>
<i>IN_MSB</i>	Constant, most significant, left bit defining field in <code>in_reg</code>
<i>IN_LSB</i>	Constant, least significant, right bit defining field in <code>in_reg</code>

2.3.3.11 bitfield_copy__sz1

Prototype:

```
bitfield_copy__sz1(out_reg, OUT_LSB, in_reg, IN_MSB, IN_LSB)
```

Description:

Retain deprecated aliases for backwards compatibility, but with compiler warnings.

2.4 Byte Field Manipulation

2.4.1 Big / Little-endian Byte Field Extract / Compare / Branch Macros

Default is BIG-ENDIAN. By defining LITTLE_ENDIAN, the appropriate underlying swaps will be inserted. This will allow users to write code that runs big- or little-endian.

2.4.2.1 bytefield_decr

Prototype:

```
bytefield_decr(out_result, in_src, in_start_byte, in_end_byte)
```

Description:

Decrement byte field from `in_src` and place the result into `out_result`.

Example:

```
bytefield_decr(reg, $xfer[0], 1, 2) // decrement 2 byte fields
```

Instruction Count: 2 to 4

Table 2.49. bytefield_decr parameters

Name	Description
<code>out_result</code>	GPR or write transfer register
<code>in_src</code>	GPR or read transfer reg containing byte field. If GPR, it must be on the opposite bank as <code>out_result</code> .
<code>in_start_byte</code>	Starting byte position 0-3 based on endian mode
<code>in_end_byte</code>	Ending byte position 0-3 based on endian mode. Can be equal to or greater than <code>in_start_byte</code> .

2.4.2.2 bytearray_incr

Prototype:

```
bytefield_incr(out_result, in_src, in_start_byte, in_end_byte)
```

Description:

Increment byte field from `in_src` and place the result into `out_result`.

Example:

```
bytefield_incr(reg, $xfer[0], 1, 3) // increment 3 byte fields
```

Instruction Count: 2 to 4

Table 2.50. bytearray_incr parameters

Name	Description
<code>out_result</code>	GPR or write transfer register
<code>in_src</code>	GPR or read transfer reg containing byte field. If GPR, it must be on the opposite bank as <code>out_result</code> .
<code>in_start_byte</code>	Starting byte position 0-3 based on endian mode
<code>in_end_byte</code>	Ending byte position 0-3 based on endian mode. Can be equal to or greater than <code>in_start_byte</code> .

2.4.2.3 bytearray_extract

Prototype:

```
bytefield_extract(out_result, in_src, in_start_byte, in_end_byte, in_load_cc)
```

Description:

Extract byte field from `in_src` and place the result into `out_result`.

Example:

```
bytefield_extract(reg, $xfer[0], 1, 3, DO_LOAD_CC) // extract 3 byte fields
```

Instruction Count: 1 to 3

Table 2.51. bytearray_extract parameters

Name	Description
<code>out_result</code>	GPR
<code>in_src</code>	GPR or read transfer reg containing byte field. If GPR, it must be on the opposite bank as <code>out_result</code> .
<code>in_start_byte</code>	Starting byte position 0-3 based on endian mode

Name	Description
<i>in_end_byte</i>	Ending byte position 0-3 based on endian mode. Must be greater than <i>in_start_byte</i> .
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> • NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect • DO_LOAD_CC: Load condition code

2.4.2.4 bytearray_dbl_extract

Prototype:

```
bytefield_dbl_extract(out_result, in_a, in_start_byte, in_b, in_end_byte, in_load_cc)
```

Description:

Extract a field that spans two input registers.

Example:

```
bytefield_dbl_extract(reg, $xfer[0], 3, $xfer[1], 1, DO_LOAD_CC) // extract 3 byte fields
```

Instruction Count: 2 to 4

Table 2.52. bytearray_dbl_extract parameters

Name	Description
<i>out_result</i>	GPR
<i>in_a</i>	GPR or read transfer reg, left source longword. If GPR, it must be on the opposite bank as <i>out_result</i> .
<i>in_start_byte</i>	Starting byte position 0-3 of <i>in_a</i> based on endian mode
<i>in_b</i>	GPR or read transfer register, right source longword. If GPR, it must be on the opposite bank as <i>out_result</i> .
<i>in_end_byte</i>	Ending byte position 0-3 of <i>in_b</i> based on endian mode. Must be less than <i>in_start_byte</i> .
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed. Possible values are:</p> <ul style="list-style-type: none"> • NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect • DO_LOAD_CC: Load condition code

2.4.2.5 bytearray_insert

Prototype:

```
bytefield_insert(io_a, in_byte_mask, in_b, in_load_cc)
```

Description:

Insert bytes from `in_b` specified by `in_byte_mask` into `io_a`.

The source and destination registers must have the same endianness.

Example:

```
bytefield_insert(x, 0110, y, DO_LOAD_CC) // insert y bytes 1,2 into x bytes 1,2
```

Instruction Count: 1 to 3

Table 2.53. bytefield_insert parameters

Name	Description
<code>io_a</code>	GPR
<code>in_byte_mask</code>	xxxx, where x = 0 or 1. If 1, insert byte.
<code>in_b</code>	GPR or read transfer register. If GPR, it must be on the opposite bank as <code>io_a</code> .
<code>in_load_cc</code>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed. Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.4.2.6 bytefield_select

Prototype:

```
bytefield_select(out_result, in_byte_mask, in_src, in_load_cc)
```

Description:

Insert bytes from `in_src` specified by `in_byte_mask` into `out_result`.

The output register `out_result` is cleared prior to the insert.

The source and destination registers must have the same endianness.

Example:

```
bytefield_select(x, 0110, y, DO_LOAD_CC) // output bytes 1-2
```

Instruction Count: 1 to 3

Table 2.54. bytefield_select parameters

Name	Description
<i>out_result</i>	GPR or write transfer register
<i>in_byte_mask</i>	xxxx, where x = 0 or 1. If 1, insert byte.
<i>in_src</i>	GPR or read transfer register. If both <i>out_result</i> and <i>in_src</i> are GPR, <i>in_src</i> must be on the opposite bank as <i>out_result</i> .
<i>in_load_cc</i>	CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are: <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.4.2.7 bytefield_br_eq

Prototype:

```
bytefield_br_eq(in_src, in_start_byte, in_end_byte, COMPARE_VAL, target_label)
```

Description:

Compare byte field to a constant and branch if equal.

Example:

```
bytefield_br_eq(test_reg, 0, 2, 0xff, exception_handler#)
```

Instruction Count: 2 to 4

Table 2.55. bytefield_br_eq parameters

Name	Description
<i>in_src</i>	GPR or read transfer reg containing byte field
<i>in_start_byte</i>	Starting byte position 0-3 based on endian mode
<i>in_end_byte</i>	Ending byte position 0-3 based on endian mode. Must be greater than <i>in_start</i> byte.
<i>COMPARE_VAL</i>	Constant value to compare field with
<i>target_label</i>	Label to branch to

2.4.2.8 bytefield_br_gtr

Prototype:

```
bytefield_br_gtr(in_src, in_start_byte, in_end_byte, COMPARE_VAL, target_label)
```

Description:

Compare byte field to a constant and branch if field is greater.

Example:

```
bytefield_br_gtr(test_reg, 0, 2, 0xff, exception_handler#)
```

Instruction Count: 2 to 4

Table 2.56. bytefield_br_gtr parameters

Name	Description
<i>in_src</i>	GPR or read transfer reg containing byte field
<i>in_start_byte</i>	Starting byte position 0-3 based on endian mode
<i>in_end_byte</i>	Ending byte position 0-3 based on endian mode. Must be greater than <i>in_start</i> byte.
<i>COMPARE_VAL</i>	Constant value to compare field with
<i>target_label</i>	Label to branch to

2.4.2.9 bytefield_br_gtreq

Prototype:

```
bytefield_br_gtreq(in_src, in_start_byte, in_end_byte, COMPARE_VAL, target_label)
```

Description:

Compare byte field to a constant and branch if field is greater or equal.

Example:

```
bytefield_br_gtreq(test_reg, 0, 2, 0xff, exception_handler#)
```

Instruction Count: 2 to 4

Table 2.57. bytefield_br_gtreq parameters

Name	Description
<i>in_src</i>	GPR or read transfer reg containing byte field
<i>in_start_byte</i>	Starting byte position 0-3 based on endian mode
<i>in_end_byte</i>	Ending byte position 0-3 based on endian mode. Must be greater than <i>in_start</i> byte.
<i>COMPARE_VAL</i>	Constant value to compare field with
<i>target_label</i>	Label to branch to

2.4.2.10 bytefield_br_less

Prototype:

```
bytefield_br_less(in_src, in_start_byte, in_end_byte, COMPARE_VAL, target_label)
```

Description:

Compare byte field to a constant and branch if field is less.

Example:

```
bytefield_br_less(test_reg, 0, 2, 0xff, exception_handler#)
```

Instruction Count: 2 to 4

Table 2.58. bytefield_br_less parameters

Name	Description
<i>in_src</i>	GPR or read transfer reg containing byte field
<i>in_start_byte</i>	Starting byte position 0-3 based on endian mode
<i>in_end_byte</i>	Ending byte position 0-3 based on endian mode. Must be greater than <i>in_start</i> byte.
<i>COMPARE_VAL</i>	Constant value to compare field with
<i>target_label</i>	Label to branch to

2.4.2.11 bytefield_br_lesseq

Prototype:

```
bytefield_br_lesseq(in_src, in_start_byte, in_end_byte, COMPARE_VAL, target_label)
```

Description:

Compare byte field to a constant and branch if field is less or equal.

Example:

```
bytefield_br_lesseq(test_reg, 0, 2, 0xff, exception_handler#)
```

Instruction Count: 2 to 4

Table 2.59. bytefield_br_lesseq parameters

Name	Description
<i>in_src</i>	GPR or read transfer reg containing byte field
<i>in_start_byte</i>	Starting byte position 0-3 based on endian mode
<i>in_end_byte</i>	Ending byte position 0-3 based on endian mode. Must be greater than <i>in_start</i> byte.
<i>COMPARE_VAL</i>	Constant value to compare field with
<i>target_label</i>	Label to branch to

2.4.2.12 bytefield_comp

Prototype:


```
bytefield_comp(in_src, in_start_byte, in_end_byte, COMPARE_VAL)
```

Description:

Extract byte field based on endian mode.

Compare byte field to COMPARE_VAL.

Condition code is set as result of compare.

Example:

```
bytefield_comp(test_reg, 0, 2, 0xff)
```

Instruction Count: 2 to 4

Table 2.60. bytefield_comp parameters

Name	Description
<i>in_src</i>	GPR or read transfer reg containing byte field
<i>in_start_byte</i>	Starting byte position 0-3 based on endian mode
<i>in_end_byte</i>	Ending byte position 0-3 based on endian mode. Must be greater than <i>in_start</i> byte.
<i>COMPARE_VAL</i>	Constant value to compare field with

2.4.2.13 bytefield_shf_left_insert

Prototype:

```
bytefield_shf_left_insert(io_a, in_byte_mask, in_b, in_shift_amt, in_load_cc)
```

Description:

Insert bytes from *in_b* specified by *in_byte_mask* into *io_a*.

The source and destination registers must have the same endianness.

Example:

```
bytefield_shf_left_insert(x, 0110, y, 8, DO_LOAD_CC) // insert y bytes 2,3 into x bytes 1,2
```

Instruction Count: 1 to 3

Table 2.61. bytefield_shf_left_insert parameters

Name	Description
<i>io_a</i>	GPR
<i>in_byte_mask</i>	xxxx, where x = 0 or 1. If 1, insert byte.
<i>in_b</i>	GPR or read transfer register. If GPR, it must be on the opposite bank as <i>io_a</i> .
<i>in_shift_amt</i>	Shift amount 0-31

Name	Description
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed. Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.4.2.14 bytefield_shf_right_insert

Prototype:

```
bytefield_shf_right_insert(io_a, in_byte_mask, in_b, in_shift_amt, in_load_cc)
```

Description:

Insert bytes from *in_b* specified by *in_byte_mask* into *io_a* after shifting *in_b* left by *in_shift_amt*.

The source and destination registers must have the same endianness.

Example:

```
bytefield_shf_right_insert(x, 0110, y, 8, DO_LOAD_CC) // insert y bytes 0,1 into x bytes 1,2
```

Instruction Count: 1 to 3

Table 2.62. bytefield_shf_right_insert parameters

Name	Description
<i>io_a</i>	GPR
<i>in_byte_mask</i>	xxxx, where x = 0 or 1. If 1, insert byte.
<i>in_b</i>	GPR or read transfer register. If GPR, it must be on the opposite bank as <i>io_a</i> .
<i>in_shift_amt</i>	Shift amount 0-31
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.4.2.15 bytefield_clr_insert

Prototype:

```
bytefield_clr_insert(out_result, in_byte_mask, in_b, in_load_cc)
```

Description:

Insert bytes from `in_b` specified by `in_byte_mask` into `out_result`.

The output register `out_result` is cleared prior to the insert.

The source and destination registers must have the same endianness.

Example:

```
bytefield_clr_insert(x, 0110, y, DO_LOAD_CC) // insert y bytes 1,2 into x bytes 1,2
```

Instruction Count: 1 to 3

Table 2.63. bytefield_clr_insert parameters

Name	Description
<code>out_result</code>	GPR or write transfer register
<code>in_byte_mask</code>	xxxx, where x = 0 or 1. If 1, insert byte.
<code>in_b</code>	GPR or read transfer register. If both <code>out_result</code> and <code>in_b</code> are GPR, <code>in_b</code> must be on the opposite bank as <code>out_result</code> .
<code>in_load_cc</code>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed. Possible values are:</p> <ul style="list-style-type: none"> <code>NO_LOAD_CC</code>: Do not load condition code - suggested value if users only want to select byte(s) without any side effect <code>DO_LOAD_CC</code>: Load condition code

2.4.2.16 bytefield_clr_shf_left_insert

Prototype:

```
bytefield_clr_shf_left_insert(out_result, in_byte_mask, in_b, in_shift_amt, in_load_cc)
```

Description:

Insert bytes from `in_b` specified by `in_byte_mask` into `out_result` after shifting `in_b` left by `in_shift_amt`.

The output register `out_result` is cleared prior to the insert.

The source and destination registers must have the same endianness.

Example:

```
bytefield_clr_shf_left_insert(x, 0110, y, 8, DO_LOAD_CC) // insert y bytes 2,3 into x bytes 1,2
```

Instruction Count: 1 to 3

Table 2.64. bytefield_clr_shf_left_insert parameters

Name	Description
<i>out_result</i>	GPR or write transfer register
<i>in_byte_mask</i>	xxxx, where x = 0 or 1. If 1, insert byte.
<i>in_b</i>	GPR or read transfer register. If both <i>out_result</i> and <i>in_b</i> are GPR, <i>in_b</i> must be on the opposite bank as <i>out_result</i> .
<i>in_shift_amt</i>	Shift amount 0-31
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed. Possible values are:</p> <ul style="list-style-type: none"> • NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect • DO_LOAD_CC: Load condition code

2.4.2.17 bytefield_clr_shf_right_insert

Prototype:

```
bytefield_clr_shf_right_insert(out_result, in_byte_mask, in_b, in_shift_amt, in_load_cc)
```

Description:

Insert bytes from *in_b* specified by *in_byte_mask* into *out_result* after shifting *in_b* left by *in_shift_amt*.

The output register *out_result* is cleared prior to the insert.

The source and destination registers must have the same endianness.

Example:

```
bytefield_clr_shf_right_insert(x, 0110, y, 8, DO_LOAD_CC) // insert y bytes 0,1 into x bytes 1,2
```

Instruction Count: 1 to 3

Table 2.65. bytefield_clr_shf_right_insert parameters

Name	Description
<i>out_result</i>	GPR or write transfer register
<i>in_byte_mask</i>	xxxx, where x = 0 or 1. If 1, insert byte.
<i>in_b</i>	GPR or read transfer register. If both <i>out_result</i> and <i>in_b</i> are GPR, <i>in_b</i> must be on the opposite bank as <i>out_result</i> .
<i>in_shift_amt</i>	Shift amount 0-31

Name	Description
<i>in_load_cc</i>	<p>CONSTANT to specify whether user wants the load ALU condition codes based on the result performed. Possible values are:</p> <ul style="list-style-type: none"> NO_LOAD_CC: Do not load condition code - suggested value if users only want to select byte(s) without any side effect DO_LOAD_CC: Load condition code

2.5 Cluster Local Scratch

2.5.1 Cluster Local Scratch Operation Macros

Default: all memory operation word counts are actual word counts. MEM_WD_COUNT_MIN_1

Table 2.66. Cluster Local Scratch and Defines

Defined	Definition
lscratch_memset	<p>cls_memset</p> <p>Deprecated alias.</p>

2.5.3.1 cls_read

Prototype:

```
cls_read(out_data, in_cls_addr, in_addr_offset, in_lw_count, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Read from Cluster Local Scratch starting at address of first longword.

Example:

```
cls_read($packet[2], addr, 0, LWCOUNT3, SIG_CS, SIG_CS, __)
```



Note

Temporary register usage: Uses 0 to 2 registers if constant addr args > MAX_IMMEDIATE, or register length.

Table 2.67. cls_read parameters

Name	Description
<i>out_data</i>	First transfer register of sequence to read to, array notation must be in xbuf array notation, index range 0-15 for NFP-32xx
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> no_option or ____: Default. Order queue optimize_mem: Mem controller selects cycle to issue priority: High priority

2.5.3.2 cls_read_le

Prototype:

```
cls_read_le(out_data, in_cls_addr, in_addr_offset, in_lw_count, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Read from Cluster Local Scratch starting at address of first longword in little endian.

Example:

```
cls_read_le($packet[2], addr, 0, LWCOUNT3, SIG_CS, SIG_CS, ____)
```



Note

Temporary register usage: uses 0 to 2 registers if constant addr args > MAX_IMMEDIATE, or register length.

Table 2.68. cls_read_le parameters

Name	Description
<i>out_data</i>	First transfer register of sequence to read to, array notation must be in xbuf array notation, index range 0-15 for NFP-32xx
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count

Name	Description
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> • <i>no_option</i> or <i>___</i>: Default. Order queue • <i>optimize_mem</i>: Mem controller selects cycle to issue • <i>priority</i>: High priority

2.5.3.3 cls_write

Prototype:

```
cls_write(in_data, in_cls_addr, in_addr_offset, in_lw_count, REQ_SIG, in_wakeup_sigs,
Q_OPTION)
```

Description:

Write to Cluster Local Scratch.

Example:

```
cls_write($packet[2], addr, 0, LWCOUNT3, SIG_CS, SIG_CS, ___)
```



Note

Temporary register usage: Uses 0 to 2 registers if constant *addr* args > MAX_IMMEDIATE, or register length.

Table 2.69. cls_write parameters

Name	Description
<i>in_data</i>	First transfer register of sequence to write from, array notation must be in xbuf array notation, index range 0-15 for NFP-32xx
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> • <i>no_option</i> or <i>___</i>: Default. Order queue • <i>optimize_mem</i>: Mem controller selects cycle to issue • <i>priority</i>: High priority

2.5.3.4 cls_write_le

Prototype:

```
cls_write_le(in_data, in_cls_addr, in_addr_offset, in_lw_count, REQ_SIG, in_wakeup_sigs,
Q_OPTION)
```

Description:

Write to Cluster Local Scratch (Little Endian).

Example:

```
cls_write_le($packet[2], addr, 0, LWCOUNT3, SIG_CS, SIG_CS, ____)
```



Note

Temporary register usage: Uses 0 to 2 registers if constant addr args > MAX_IMMEDIATE, or register length.

Table 2.70. cls_write_le parameters

Name	Description
<i>in_data</i>	First transfer register of sequence to write from, array notation must be in xbuf array notation, index range 0-15 for NFP-32xx
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> no_option or ____: Default. Order queue optimize_mem: Mem controller selects cycle to issue priority: High priority

2.5.3.5 cls_write_byte

Prototype:

```
cls_write_byte(in_data, in_cls_addr, in_addr_offset, in_lw_count, REQ_SIG, in_wakeup_sigs,
Q_OPTION)
```

Description:

Write bytes to Cluster Local Scratch.

Example:

```
cls_write_byte($packet, addr, 0, BYTE_CNT, SIG_CS, SIG_CS, ____)
```



Note

Temporary register usage: Uses 0 to 2 registers if constant addr args > MAX_IMMEDIATE, or register length.

Table 2.71. cls_write_byte parameters

Name	Description
<i>in_data</i>	First transfer register of sequence to write from, array notation must be in xbuf array notation, index range 0-15 for NFP-32xx
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> no_option or ____: Default. Order queue optimize_mem: Mem controller selects cycle to issue priority: High priority

2.5.3.6 cls_write_byte_le

Prototype:

```
cls_write_byte_le(in_data, in_cls_addr, in_addr_offset, in_lw_count, REQ_SIG,  
in_wakeup_sigs, Q_OPTION)
```

Description:

Write bytes to Cluster Local Scratch (Little Endian).

Example:

```
cls_write_byte_le($packet, addr, 0, BYTE_CNT, SIG_CS, SIG_CS, ____)
```



Note

Temporary register usage: Uses 0 to 2 registers if constant addr args > MAX_IMMEDIATE, or register length.

Table 2.72. `cls_write_byte_le` parameters

Name	Description
<i>in_data</i>	First transfer register of sequence to write from, array notation must be in xbuf array notation, index range 0-15 for NFP-32xx
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> <code>no_option</code> or <code>___</code>: Default. Order queue <code>optimize_mem</code>: Mem controller selects cycle to issue <code>priority</code>: High priority

2.5.3.7 `cls_bits_clr`

Prototype:

```
cls_bits_clr(in_mask, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Clear `in_mask` bits at Cluster Local Scratch longword location.

Table 2.73. `cls_bits_clr` parameters

Name	Description
<i>in_mask</i>	Register or constant, mask of bits to set
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> <code>no_option</code> or <code>___</code>: Default. Order queue <code>optimize_mem</code>: Mem controller selects cycle to issue <code>priority</code>: High priority

2.5.3.8 `cls_bits_set`

Prototype:

```
cls_bits_set(in_mask, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, length,  
Q_OPTION)
```

Description:

Set *in_mask* bits at Cluster Local Scratch longword location.

Example:

```
cls_bits_set(0x111, 0, bit_position, SIG_CS, SIG_CS, ____)
```

Instruction Count: 2 to 6

Table 2.74. *cls_bits_set* parameters

Name	Description
<i>in_mask</i>	Register or constant, mask of bits to set
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>length</i>	Word count, <i>dual_sig_op</i> (for 2-signal ops), or <i>no_wd_count</i>
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> <i>no_option</i> or <i>____</i>: Default. Order queue <i>optimize_mem</i>: Mem controller selects cycle to issue <i>priority</i>: High priority

2.5.3.9 *cls_bits_test_and_clr*

Prototype:

```
cls_bits_test_and_clr(out_data, in_mask, in_cls_addr, in_addr_offset, REQ_SIG,  
in_wakeup_sigs, length, Q_OPTION)
```

Description:

Clear *in_mask* bits at Cluster Local Scratch longword location.

Read contents of Cluster Local Scratch address prior to the write.

Table 2.75. *cls_bits_test_and_clr* parameters

Name	Description
<i>out_data</i>	Read transfer register result
<i>in_mask</i>	Register or constant, mask of bits to set

Name	Description
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>length</i>	Word count, <i>dual_sig_op</i> (for 2-signal ops), or <i>no_wd_count</i>
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> <i>no_option</i> or <i>___</i>: Default. Order queue <i>optimize_mem</i>: Mem controller selects cycle to issue <i>priority</i>: High priority

2.5.3.10 *cls_bits_test_and_set*

Prototype:

```
cls_bits_test_and_set(out_data, in_mask, in_cls_addr, in_addr_offset, REQ_SIG,
in_wakeup_sigs, length, Q_OPTION)
```

Description:

Set *in_mask* bits at Cluster Local Scratch longword location.

Read contents of Cluster Local Scratch address prior to the write.

Example:

```
cls_bits_test_and_set(prev_value, 0x1000, addr0, addr1, SIG_CS, SIG_CS, ___) // test/set bit 3
```

Table 2.76. *cls_bits_test_and_set* parameters

Name	Description
<i>out_data</i>	Read transfer register result
<i>in_mask</i>	Register or constant, mask of bits to set
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>length</i>	Word count, <i>dual_sig_op</i> (for 2-signal ops), or <i>no_wd_count</i>
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> <i>no_option</i> or <i>___</i>: Default. Order queue <i>optimize_mem</i>: Mem controller selects cycle to issue <i>priority</i>: High priority

2.5.3.11 cls_incr

Prototype:

```
cls_incr(in_cls_addr, in_addr_offset)
```

Description:

Increment 32-bit longword at Cluster Local Scratch location.

Table 2.77. cls_incr parameters

Name	Description
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset

2.5.3.12 cls_decr

Prototype:

```
cls_decr(in_cls_addr, in_addr_offset)
```

Description:

Decrement 32-bit longword at Cluster Local Scratch location.

Table 2.78. cls_decr parameters

Name	Description
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset

2.5.3.13 cls_add

Prototype:

```
cls_add(in_data, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Add *in_data* to Cluster Local Scratch location.

Table 2.79. cls_add parameters

Name	Description
<i>in_data</i>	Data to be added to Cluster Local Scratch location specified by <i>in_cls_addr</i> and <i>in_addr_offset</i> . <i>in_data</i> must be a write transfer register i.e. out transfer register.

Name	Description
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> • <i>no_option</i> or ____: Default. Order queue • <i>optimize_mem</i>: Mem controller selects cycle to issue • <i>priority</i>: High priority

2.5.3.14 cls_sub

Prototype:

```
cls_sub(in_data, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Subtract *in_data* from Cluster Local Scratch location.



Note

The sub instruction is not supported in HW, so subtract from 0 and add that number. *in_data* can be GPR or a read transfer register.

Table 2.80. cls_sub parameters

Name	Description
<i>in_data</i>	Data to be added to Cluster Local Scratch location specified by <i>in_cls_addr</i> and <i>in_addr_offset</i>
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> • <i>no_option</i> or ____: Default. Order queue • <i>optimize_mem</i>: Mem controller selects cycle to issue • <i>priority</i>: High priority

2.5.3.15 cls_swap

Prototype:

```
cls_swap(out_data, in_data, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Write *in_data* to Cluster Local Scratch location.

Read contents of Cluster Local Scratch location prior to the operation to *out_data*.

Example:

```
cls_swap(prev_value, new_value, addr0, addr1, SIG_CS, SIG_CS, __) // test/set bit 3
```



Note

If *in_data* and *out_data* is a pair of read/write transfer registers with the same name, eg. \$buffer0, the data from the \$buffer0.write will be written to (*in_cls_addr* + *in_addr_offset*). The data from (*in_cls_addr* + *in_addr_offset*) will be returned in \$buffer0.read.

Table 2.81. cls_swap parameters

Name	Description
<i>out_data</i>	A read/write transfer registers pair. Result is returned in the read part of the read/write transfer registers pair.
<i>in_data</i>	Can be constant, GPR, read transfer register
<i>in_cls_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> • <i>no_option</i> or <i>__</i>: Default. Order queue • <i>optimize_mem</i>: Mem controller selects cycle to issue • <i>priority</i>: High priority

2.5.3.16 cls_test_and_add

Prototype:

```
cls_test_and_add(out_data, in_data, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Read contents of Cluster Local Scratch location to `out_data` then add `in_data` to Cluster Local Scratch location contents.

Example:

```
cls_test_and_add(prev_value, addend, addr0, addr1, SIG_CS, SIG_CS, __) // test/set bit 3
```



Note

If `in_data` and `out_data` is a pair of read/write transfer registers with the same name, eg. `$buffer0`, the data from the `$buffer0.write` will be written to (`in_cls_addr + in_addr_offset`). The data from (`in_cls_addr + in_addr_offset`) will be returned in `$buffer0.read`.

Table 2.82. `cls_test_and_add` parameters

Name	Description
<code>out_data</code>	A read/write transfer registers pair. Result is returned in the read part of the read/write transfer registers pair.
<code>in_data</code>	Can be constant, GPR, read transfer register
<code>in_cls_addr</code>	Register or constant base longword address
<code>in_addr_offset</code>	Register or constant longword address offset
<code>REQ_SIG</code>	Signal associated with this request
<code>in_wakeup_sigs</code>	Signal or signals to wake up on
<code>Q_OPTION</code>	Queue options: <ul style="list-style-type: none"> <code>no_option</code> or <code>__</code>: Default. Order queue <code>optimize_mem</code>: Mem controller selects cycle to issue <code>priority</code>: High priority

2.5.3.17 `cls_test_and_decr`

Prototype:

```
cls_test_and_decr(out_data, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Read contents of Cluster Local Scratch location to `out_data` then decrement Cluster Local Scratch location contents.

Example:

```
cls_test_and_decr(prev_value, addr0, addr1, SIG_CS, SIG_CS, __) // test/set bit 3
```




Note

`out_data` must be a transfer register.

Table 2.83. `cls_test_and_decr` parameters

Name	Description
<code>out_data</code>	A read/write transfer registers pair. Result is returned in the read part of the read/write transfer registers pair.
<code>in_cls_addr</code>	Register or constant base longword address
<code>in_addr_offset</code>	Register or constant longword address offset
<code>REQ_SIG</code>	Signal associated with this request
<code>in_wakeup_sigs</code>	Signal or signals to wake up on
<code>Q_OPTION</code>	Queue options: <ul style="list-style-type: none"> <code>no_option</code> or <code>___</code>: Default. Order queue <code>optimize_mem</code>: Mem controller selects cycle to issue <code>priority</code>: High priority

2.5.3.18 `cls_test_and_incr`

Prototype:

```
cls_test_and_incr(out_data, in_cls_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Read contents of Cluster Local Scratch location to `out_data` then increment Cluster Local Scratch location contents.

Example:

```
cls_test_and_incr(prev_value, addr0, addr1, SIG_CS, SIG_CS, ___) // test/set bit 3
```



Note

`out_data` must be a transfer register.

Table 2.84. `cls_test_and_incr` parameters

Name	Description
<code>out_data</code>	A read/write transfer registers pair. Result is returned in the read part of the read/write transfer registers pair.
<code>in_cls_addr</code>	Register or constant base longword address
<code>in_addr_offset</code>	Register or constant longword address offset

Name	Description
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue options: <ul style="list-style-type: none"> • <i>no_option</i> or ____: Default. Order queue • <i>optimize_mem</i>: Mem controller selects cycle to issue • <i>priority</i>: High priority

2.5.3.19 cls_add64_immed_init

Prototype:

```
cls_add64_immed_init(indirect_ref_reg)
```

Description:

Initialize indirect reference register for use in IXP or NFP indirect reference format mode.

Table 2.85. cls_add64_immed_init parameters

Name	Description
<i>indirect_ref_reg</i>	Indirect GPR register to be initialized

2.5.3.20 cls_add64_immed

Prototype:

```
cls_add64_immed(indirect_ref_reg, val, addr, offset)
```

Description:

Perform immediate add (64-bit) operation in Cluster Local Scratch.

Can add up to a 16-bit value in NFP indirect reference mode and up to 255 in IXP indirect reference format mode. For a 16-bit value, the two most significant bits control sign extension (bits 15 and 14). This means only 14-bits are used for an actual value.

Bits 15 and 14 of a 16-bit value, as from the *Netronome Network Flow Processor 6000 Programmers Reference*, CLS (Atomic Operations):

- 00 indicates no sign extension
- 01 indicates sign extend to 32-bit/64-bit value for 32/64-bit operation respectively
- 10 indicates no sign extension, but duplicate immediate in both high and low 32-bit words (applicable to 64-bit operations only)

- 11 means sign extend to 32-bit word and duplicate value in both high and low 32-bit words (applicable to 64-bit operations only).

Using `val` in the range from 1 to 7 (atomic increment/add), saves one instruction.

Table 2.86. `cls_add64_immed` parameters

Name	Description
<code>indirect_ref_reg</code>	Indirect GPR register initialized with <code>cls_add64_immed_init</code> . Not needed when <code>val</code> is a constant from 0 to 7.
<code>val</code>	Value to be added. Can be constant or GPR. If constant: Values in range 1-7, no indirect and save 1 instruction. Values in range 8-255, indirect reference. Values > 255 must be in GPR and use indirect reference.
<code>addr</code>	Base Address in cluster scratch where immed add operation is done. Byte address. Constant or GPR.
<code>offset</code>	Offset from base. Add is done at (<code>addr + offset</code>). Byte address.

2.5.3.21 `cls_memset`

Prototype:

```
cls_memset(in_cls_addr, in_len, lw_pattern, CHUNK_SIZE)
```

Description:

Fill a region of Cluster Local Scratch memory with a specified pattern.

Table 2.87. `cls_memset` parameters

Name	Description
<code>in_cls_addr</code>	Address to start memory fill from
<code>in_len</code>	Number of bytes to set. Must be a multiple of <code>CHUNK_SIZE</code> .
<code>lw_pattern</code>	32-bit pattern to fill memory region with
<code>CHUNK_SIZE</code>	Chunk size, a multiple of 4 bytes from 4 to 128 in NFP indirect reference format mode and 4 to 64 otherwise. Must be a constant.

2.6 Common and Global Constants

Table 2.88. Common and Global Constants and Defines

Defined	Definition
<code>MEM_WD_COUNT_MIN_1</code>	FALSE Control word count passed to memory access macros.

Defined	Definition
	<p>Values:</p> <ul style="list-style-type: none"> • TRUE: All memory access macros must be called with <code>word count = actual number of word - 1</code>. This will help save 1 cycle. • FALSE: All memory access macros are called with <code>word count = actual number of word</code> <div style="border: 1px solid blue; padding: 10px; margin-top: 10px;">  <p>Note</p> <p>Default is FALSE.</p> </div>
MAX_IMMEDIATE	0xFF
MAX_IMMEDIATE_ADDR	0x7f
	Only 7 bits available in memory operations.
SIG_NONE	0
_____	0
optimize_mem	unordered
FALSE	0
TRUE	1
UNALLOCATED	0
FREELIST0	0
FREELIST1	1
FREELIST2	2
FREELIST3	3
FREELIST4	4
FREELIST5	5
FREELIST6	6
FREELIST7	7
XFRINDEX0	0
XFRINDEX1	1
XFRINDEX2	2
XFRINDEX3	3
XFRINDEX4	4
XFRINDEX5	5
XFRINDEX6	6
XFRINDEX7	7
QWCOUNT1	1

Defined	Definition
QWCOUNT2	2
QWCOUNT3	3
QWCOUNT4	4
QWCOUNT5	5
QWCOUNT6	6
QWCOUNT7	7
QWCOUNT8	8
QWOFFSET0	0
QWOFFSET1	1
QWOFFSET2	2
QWOFFSET3	3
QWOFFSET4	4
QWOFFSET5	5
QWOFFSET6	6
QWOFFSET7	7
LWCOUNT1	1
LWCOUNT2	2
LWCOUNT3	3
LWCOUNT4	4
LWCOUNT5	5
LWCOUNT6	6
LWCOUNT7	7
LWCOUNT8	8
LWOFFSET0	0
LWOFFSET1	1
LWOFFSET2	2
LWOFFSET3	3
LWOFFSET4	4
LWOFFSET5	5
LWOFFSET6	6
LWOFFSET7	7
BYTEOFFSET0	0
BYTEOFFSET1	1
BYTEOFFSET2	2
BYTEOFFSET3	3

Defined	Definition
BYTEOFFSET4	4
BYTEOFFSET5	5
BYTEOFFSET6	6
BYTEOFFSET7	7
BYTEOFFSET8	8
BYTEOFFSET9	9
BYTEOFFSET10	10
BYTEOFFSET11	11
BYTEOFFSET12	12
BYTEOFFSET13	13
BYTEOFFSET14	14
BYTEOFFSET15	15
BYTEOFFSET16	16
BYTEOFFSET17	17
BYTEOFFSET18	18
BYTEOFFSET19	19
BYTEOFFSET20	20
BYTEOFFSET21	21
BYTEOFFSET22	22
BYTEOFFSET23	23
BYTEOFFSET24	24
BYTEOFFSET25	25
BYTEOFFSET26	26
BYTEOFFSET27	27
BYTEOFFSET28	28
BYTEOFFSET29	29
BYTEOFFSET30	30
BYTEOFFSET31	31
PKT_DENY	0x00
PKT_PERMIT	0x01
PKT_QUEUE_TO_CORE	0x02
OP_SIZE_8X24	1
OP_SIZE_16X16	2
OP_SIZE_16X32	3
OP_SIZE_32X32	4

Defined	Definition
BYTES_PER_LW	4
BYTES_PER_QW	8
MU_LOCALITY_HIGH	0
MU_LOCALITY_LOW	1
MU_LOCALITY_DIRECT_ACCESS	2
MU_LOCALITY_DISCARD_AFTER_READ	3

2.7 CRYPTO IPsec Operation

2.7.1 CRYPTO IPsec Operation Macros

This file contains a set of crypto-library sequences. The sequences are designed to be compatible with Netronome's crypto_support facility. They are implemented as 'compressed' sequences, which can be preloaded to the CIB memory space of the bulk core units at initialization time, and then invoked at run-time on a per-packet basis quickly and efficiently. These sequences implement encryption, decryption, and authentication that is intended to be usable on IPsec-formatted packets encapsulated with ESP or AH. The intention is for the ME to issue a single get_core() at initialization time for a specific Crypto Dispatcher context, then keep that core 'pinned' to that context for the life of the program. The selection of which bulk core unit works on a particular packet is thus pushed to earlier in the process, when the packet is assigned to an ME. This technique has the advantage that it avoids the processing delay associated with executing the get_core() and free_core() instructions for every packet. It also eliminates some of the variables that need to be passed to the CIB code sequences, because each core can use a fixed set of pre-allocated buffers that are statically assigned to the ME's. The source file for these sequences is crypto_lib_ipsec.crypt. The source file is processed by the Netronome utility ca2.py to generate the file crypto_lib_ipsec.uc, which is included in the microcode. The macros defined in crypto_lib_ipsec.uc are used by the macros in crypto_lib.uc.

Table 2.89. CRYPTO IPsec Operation and Defines

Defined	Definition
CRYPTO_NFP_MODE	1 SA layout for IPsec-oriented crypto instruction sequences. byte offset description 0x20-0x3F cipher key (up to 32 bytes) 0x40-0x7F authentication key (up to 64 bytes) Based constants: temp0 and temp1 - used for scratch space sa_hmk_X and sa_cik_X - HMAC and cipher key, fixed locations within the SA Note, for 'pinned' operation, each bulk core (crypto context) is 'assigned' four buffer sets. The buffer sets include the SA and the packet. At initialization time, the four SA base addresses

Defined	Definition
	must be passed in as base 1/2/3/4; this set of four addresses is unique to each bulk core. Implementators may choose to use fewer than the four allocated buffer sets.

2.7.3.1 crypto_load_ipsec_enc_pin_A

Prototype:

```
crypto_load_ipsec_enc_pin_A(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

ipsec_enc_pin_A, ipsec_enc_pin_B, ipsec_enc_pin_C, ipsec_enc_pin_D

Sequences to do IPSec-compatible encryption of a packet, using bulk cores that stay 'pinned' to a Dispatcher context. This allows operation without per-packet get_core()/free_core() ops, and also allows preloaded constants to point to fixed addresses within the SA, so that SA-resident parameters do not need to be passed as variables.

Supplied in 4 different versions (_A, _B, _C, _D suffix) to handle up to 4 buffer sets per core. One context will do a get_core() at init time and will hold onto that core for the life of the program. As each packet arrives at the ME, it will assign it to a buffer set and then use its Dispatcher context/core to call the appropriate buffer-set-specific sequence.

For this set of CIB sequences, we also assume that the 'output' (ciphertext) data is at the same crypto SRAM address as the 'input' data. This fits architecturally with the idea of having several buffer sets per core, and using ME threads to do the scheduling. It is also important from a performance perspective, because it keeps the number of parameters down to 8, which can all be transferred to CIB in one cycle, which comes for 'free' as part of the setup_load_CIB() opcode that runs the compressed routine.

Supports AES, DES, and NO encryption with MD5 or SHA-x HMAC auth.

Calling the sequence:

```
crypto_load_ipsec_enc_pin_X (packet, in_len, auth_only_data, iv, auth_length, hmac_keylen, crypt_select,
crypt_modes)
```

Table 2.90. crypto_load_ipsec_enc_pin_A parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM address of the start of plaintext to be encrypted
<i>in_len</i>	length of data to be encrypted
<i>auth_only_data</i>	crypto SRAM address of auth-only data (SPI and seq number)
<i>iv</i>	crypto SRAM address of the Initialization Vector
<i>auth_length</i>	byte length of (SPI/Seq

Name	Description
<i>hmac_keylen</i>	byte length of HMAC key, minus 1 (15 for MD5, 19 for SHA1, etc)
<i>crypt_select</i>	config word 1, produced by <code>crypto_setup_configs</code>
<i>crypt_modes</i>	config word 2, produced by <code>crypto_setup_configs</code> Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.2 `crypto_gen_ipsec_enc_pin_A`

Prototype:

```
crypto_gen_ipsec_enc_pin_A(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in `_crypto_library_load_dynamic` in `crypto_lib.uc`

Table 2.91. `crypto_gen_ipsec_enc_pin_A` parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.3 `crypto_load_ipsec_enc_pin_B`

Prototype:

```
crypto_load_ipsec_enc_pin_B(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.4 `crypto_gen_ipsec_enc_pin_B`

Prototype:

```
crypto_gen_ipsec_enc_pin_B(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in `_crypto_library_load_dynamic` in `crypto_lib.uc`

Table 2.92. crypto_gen_ipsec_enc_pin_B parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.5 crypto_load_ipsec_enc_pin_C

Prototype:

```
crypto_load_ipsec_enc_pin_C(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.6 crypto_gen_ipsec_enc_pin_C

Prototype:

```
crypto_gen_ipsec_enc_pin_C(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.93. crypto_gen_ipsec_enc_pin_C parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.7 crypto_load_ipsec_enc_pin_D

Prototype:

```
crypto_load_ipsec_enc_pin_D(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.8 crypto_gen_ipsec_enc_pin_D

Prototype:

```
crypto_gen_ipsec_enc_pin_D(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in `_crypto_library_load_dynamic` in `crypto_lib.uc`

Table 2.94. crypto_gen_ipsec_enc_pin_D parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.9 crypto_load_ipsec_dec_pin_A

Prototype:

```
crypto_load_ipsec_dec_pin_A(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,  
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

`ipsec_dec_pin_A`, `ipsec_dec_pin_B`, `ipsec_dec_pin_C`, `ipsec_dec_pin_D`

Sequences to do IPSec-compatible decryption of a packet, using bulk cores that stay 'pinned' to a Dispatcher context. This allows operation without per-packet `get_core()/free_core()` ops, and also allows preloaded constants to point to fixed addresses within the SA, so that SA-resident parameters do not need to be passed as variables.

Supplied in 4 different versions (`_A`, `_B`, `_C`, `_D` suffix) to handle up to 4 buffer sets per core. One context will do a `get_core()` at init time and will hold onto that core for the life of the program. As each packet arrives at the ME, it will assign it to a buffer set and then use its Dispatcher context/core to call the appropriate buffer-set-specific sequence.

For this set of CIB sequences, we also assume that the 'output' (plaintext) data is at the same crypto SRAM address as the 'input' data. This fits architecturally with the idea of having several buffer sets per core, and using ME threads to do the scheduling. It is also important from a performance perspective, because it keeps the number of parameters down to 8, which can all be transferred to CIB in one cycle, which comes for 'free' as part of the `setup_load_CIB()` opcode that runs the compressed routine.

Supports AES, DES, and NO encryption with MD5 or SHA-x HMAC auth.

Calling the sequence:

`crypto_load_ipsec_dec_pin_X` (packet, in_len, auth_only_data, iv, auth_length, hmac_keylen, crypt_select, crypt_modes)

Table 2.95. `crypto_load_ipsec_dec_pin_A` parameters

Name	Description
<code>cr_xfr</code>	
<code>cr_ctx</code>	
<code>packet</code>	crypto SRAM address of the start of ciphertext to be decrypted
<code>in_len</code>	length of data to be decrypted
<code>auth_only_data</code>	crypto SRAM address of auth-only data (SPI and seq number)
<code>iv</code>	crypto SRAM address of the Initialization Vector
<code>auth_length</code>	byte length of (SPI/Seq
<code>hmac_keylen</code>	byte length of HMAC key, minus 1 (15 for MD5, 19 for SHA1, etc)
<code>crypt_select</code>	config word 1, produced by <code>crypto_setup_configs</code>
<code>crypt_modes</code>	config word 2, produced by <code>crypto_setup_configs</code> Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.10 `crypto_gen_ipsec_dec_pin_A`

Prototype:

```
crypto_gen_ipsec_dec_pin_A(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in `_crypto_library_load_dynamic` in `crypto_lib.uc`

Table 2.96. `crypto_gen_ipsec_dec_pin_A` parameters

Name	Description
<code>core</code>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<code>desc</code>	sequence 'descriptor', contains sram location address

2.7.3.11 `crypto_load_ipsec_dec_pin_B`

Prototype:

```
crypto_load_ipsec_dec_pin_B(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.12 crypto_gen_ipsec_dec_pin_B

Prototype:

```
crypto_gen_ipsec_dec_pin_B(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.97. crypto_gen_ipsec_dec_pin_B parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.13 crypto_load_ipsec_dec_pin_C

Prototype:

```
crypto_load_ipsec_dec_pin_C(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.14 crypto_gen_ipsec_dec_pin_C

Prototype:

```
crypto_gen_ipsec_dec_pin_C(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.98. crypto_gen_ipsec_dec_pin_C parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.15 crypto_load_ipsec_dec_pin_D

Prototype:

```
crypto_load_ipsec_dec_pin_D(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.16 crypto_gen_ipsec_dec_pin_D

Prototype:

```
crypto_gen_ipsec_dec_pin_D(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.99. crypto_gen_ipsec_dec_pin_D parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.17 crypto_load_ipsec_enc_strt_pin_A

Prototype:

```
crypto_load_ipsec_enc_strt_pin_A(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

ipsec_enc_strt_pin_A, B, C, D

Similar to ipsec_enc_pin_A,B,C,D, but used to begin an encryption sequence that will span multiple buffers, needed to handle jumbo packets.

Calling the sequence:

```
load_ipsec_enc_strt_pin_X (packet, in_len, auth_only_data, iv, auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Table 2.100. crypto_load_ipsec_enc_strt_pin_A parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of start of plaintext to encrypt
<i>in_len</i>	length of data to be encrypted
<i>auth_only_data</i>	crypto SRAM address of auth-only data (SPI and seq number)
<i>iv</i>	crypto SRAM address of the Initialization Vector
<i>auth_length</i>	byte length of (SPI/Seq
<i>hmac_keylen</i>	byte length of HMAC key, minus 1
<i>crypt_select</i>	config word 1, produced by crypto_setup_configs
<i>crypt_modes</i>	config word 2, produced by crypto_setup_configs Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.18 crypto_gen_ipsec_enc_strt_pin_A

Prototype:

```
crypto_gen_ipsec_enc_strt_pin_A(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.101. crypto_gen_ipsec_enc_strt_pin_A parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.19 crypto_load_ipsec_enc_strt_pin_B

Prototype:

```
crypto_load_ipsec_enc_strt_pin_B(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,  
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.20 crypto_gen_ipsec_enc_strt_pin_B

Prototype:

```
crypto_gen_ipsec_enc_strt_pin_B(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.102. crypto_gen_ipsec_enc_strt_pin_B parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.21 crypto_load_ipsec_enc_strt_pin_C

Prototype:

```
crypto_load_ipsec_enc_strt_pin_C(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.22 crypto_gen_ipsec_enc_strt_pin_C

Prototype:

```
crypto_gen_ipsec_enc_strt_pin_C(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.103. crypto_gen_ipsec_enc_strt_pin_C parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.23 crypto_load_ipsec_enc_strt_pin_D

Prototype:

```
crypto_load_ipsec_enc_strt_pin_D(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.24 crypto_gen_ipsec_enc_strt_pin_D

Prototype:

```
crypto_gen_ipsec_enc_strt_pin_D(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.104. crypto_gen_ipsec_enc_strt_pin_D parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.25 crypto_load_ipsec_enc_cont

Prototype:

```
crypto_load_ipsec_enc_cont(cr_xfr, cr_ctx, packet, in_len, dummy1, dummy2, dummy3)
```

Description:

ipsec_enc_cont

Used after ipsec_enc_strt_pin_X, to continue encrypting a packet on a buffer of data following the first part of the packet. Needed to handle jumbo packets. Setup and ending condition from prior use of ipsec_enc_strt_pin_X is required prior to invoking this sequence. In particular, the keys, config registers, hash address, etc. must remain intact when this sequence is started.

Note that this sequence does not use or depend on buffer specific constants, since the buffer specific values and setups are done in ipsec_enc_strt_pin_X

Calling the sequence:

load_ipsec_enc_cont (packet, in_len)

Table 2.105. crypto_load_ipsec_enc_cont parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of the start of plaintext to encrypt
<i>in_len</i>	length of data to encrypt
<i>dummy1</i>	placeholder for consistency with other sequences
<i>dummy2</i>	placeholder for consistency with other sequences
<i>dummy3</i>	placeholder for consistency with other sequences Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.26 crypto_gen_ipsec_enc_cont

Prototype:

```
crypto_gen_ipsec_enc_cont (core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.106. crypto_gen_ipsec_enc_cont parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.27 crypto_load_ipsec_enc_end

Prototype:

```
crypto_load_ipsec_enc_end (cr_xfr, cr_ctx, packet, in_len, hmac_resadr, hmac_keyadr,
hmac_keylen)
```

Description:

ipsec_enc_end

Used after ipsec_enc_strt_pin_x, and possibly ipsec_enc_cont, to complete encrypting a packet on the last buffer of data of the packet. Needed to handle jumbo packets. Setup and ending condition from prior use of

ipsec_enc_strt_pin_x is required prior to invoking this sequence. In particular, the keys, config registers, etc. must remain intact when this sequence is started.

The hash key address and result address for the packet must be specified as parameters to this sequence.

The key address should be equal to the sram location for the hash key corresponding to the buffer (A,B,C or D) used for the 1st part of the packet. For e.g., if the 1st part of the packet was loaded using buffer A, the hash key address would be the same as provided in the constant value sa_hmk_a, since that is where the key was loaded.

The hash key result address should be the sram address corresponding to the hash result for the buffer being used when this macro (ipsec_enc_end) is invoked. For example, if using buffer B, the hash result address would be equal to the constant value set in sa_hmr_b

Calling the sequence:

```
crypto_load_ipsec_enc_end (packet, in_len, hmac_resadr, hmac_keyadr, hmac_keylen)
```

Table 2.107. crypto_load_ipsec_enc_end parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of strt of plaintext to be encrypted
<i>in_len</i>	length of data to be encrypted
<i>hmac_resadr</i>	crypto SRAM address of the HMAC key calculation result
<i>hmac_keyadr</i>	crypto SRAM address of the start of the HMAC key
<i>hmac_keylen</i>	byte length of HMAC key, minus 1 Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.28 crypto_gen_ipsec_enc_end

Prototype:

```
crypto_gen_ipsec_enc_end(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.108. crypto_gen_ipsec_enc_end parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.29 crypto_load_ipsec_dec_strt_pin_A

Prototype:

```
crypto_load_ipsec_dec_strt_pin_A(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

ipsec_dec_strt_pin_A, B, C, D

Similar to ipsec_dec_pin_A,B,C,D, but used to begin a decryption sequence that will span multiple buffers, needed to handle jumbo packets.

Calling the sequence:

```
load_ipsec_dec_strt_pin_X(packet, in_len, auth_only_data, iv, auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Table 2.109. crypto_load_ipsec_dec_strt_pin_A parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of start of ciphertext to decrypt
<i>in_len</i>	length of data to be decrypted
<i>auth_only_data</i>	crypto SRAM address of auth-only data (SPI and seq number)
<i>iv</i>	crypto SRAM address of the Initialization Vector
<i>auth_length</i>	byte length of (SPI/Seq
<i>hmac_keylen</i>	byte length of HMAC key, minus 1
<i>crypt_select</i>	config word 1, produced by crypto_setup_configs
<i>crypt_modes</i>	config word 2, produced by crypto_setup_configs Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.30 crypto_gen_ipsec_dec_strt_pin_A

Prototype:

```
crypto_gen_ipsec_dec_strt_pin_A(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.110. crypto_gen_ipsec_dec_strt_pin_A parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.31 crypto_load_ipsec_dec_strt_pin_B

Prototype:

```
crypto_load_ipsec_dec_strt_pin_B(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.32 crypto_gen_ipsec_dec_strt_pin_B

Prototype:

```
crypto_gen_ipsec_dec_strt_pin_B(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.111. crypto_gen_ipsec_dec_strt_pin_B parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.33 crypto_load_ipsec_dec_strt_pin_C

Prototype:

```
crypto_load_ipsec_dec_strt_pin_C(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.34 crypto_gen_ipsec_dec_strt_pin_C

Prototype:

```
crypto_gen_ipsec_dec_strt_pin_C(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.112. crypto_gen_ipsec_dec_strt_pin_C parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.35 crypto_load_ipsec_dec_strt_pin_D

Prototype:

```
crypto_load_ipsec_dec_strt_pin_D(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.36 crypto_gen_ipsec_dec_strt_pin_D

Prototype:

```
crypto_gen_ipsec_dec_strt_pin_D(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.113. crypto_gen_ipsec_dec_strt_pin_D parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.37 crypto_load_ipsec_dec_strt_nw_pin_A

Prototype:

```
crypto_load_ipsec_dec_strt_nw_pin_A(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

ipsec_dec_strt_nw_pin_A, B, C, D

Similar to ipsec_dec_strt_pin_A,B,C,D, used to begin a decryption sequence that will span multiple buffers, when no cipher is selected, so the sequence does not have a wait for the cipher to complete. use to handle jumbo packets.

Calling the sequence:

```
load_ipsec_dec_strt_nw_pin_X (packet, in_len, auth_only_data, iv, auth_length, hmac_keylen, crypt_select,
crypt_modes)
```

Table 2.114. crypto_load_ipsec_dec_strt_nw_pin_A parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of start of ciphertext to decrypt
<i>in_len</i>	length of data to be decrypted
<i>auth_only_data</i>	crypto SRAM address of auth-only data (SPI and seq number)
<i>iv</i>	crypto SRAM address of the Initialization Vector
<i>auth_length</i>	byte length of (SPI/Seq
<i>hmac_keylen</i>	byte length of HMAC key, minus 1
<i>crypt_select</i>	config word 1, produced by crypto_setup_configs
<i>crypt_modes</i>	config word 2, produced by crypto_setup_configs Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.38 crypto_gen_ipsec_dec_strt_nw_pin_A

Prototype:

```
crypto_gen_ipsec_dec_strt_nw_pin_A(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.115. crypto_gen_ipsec_dec_strt_nw_pin_A parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.39 crypto_load_ipsec_dec_strt_nw_pin_B

Prototype:

```
crypto_load_ipsec_dec_strt_nw_pin_B(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.40 crypto_gen_ipsec_dec_strt_nw_pin_B

Prototype:

```
crypto_gen_ipsec_dec_strt_nw_pin_B(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.116. crypto_gen_ipsec_dec_strt_nw_pin_B parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.41 crypto_load_ipsec_dec_strt_nw_pin_C

Prototype:

```
crypto_load_ipsec_dec_strt_nw_pin_C(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.42 crypto_gen_ipsec_dec_strt_nw_pin_C

Prototype:

```
crypto_gen_ipsec_dec_strt_nw_pin_C(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.117. crypto_gen_ipsec_dec_strt_nw_pin_C parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.43 crypto_load_ipsec_dec_strt_nw_pin_D

Prototype:

```
crypto_load_ipsec_dec_strt_nw_pin_D(cr_xfr, cr_ctx, packet, in_len, auth_only_data, iv,
auth_length, hmac_keylen, crypt_select, crypt_modes)
```

Description:

Prepare transfer regs to load static (aka 'pinned') encryption sequence.

2.7.3.44 crypto_gen_ipsec_dec_strt_nw_pin_D

Prototype:

```
crypto_gen_ipsec_dec_strt_nw_pin_D(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.118. crypto_gen_ipsec_dec_strt_nw_pin_D parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.45 crypto_load_ipsec_dec_cont

Prototype:

```
crypto_load_ipsec_dec_cont(cr_xfr, cr_ctx, packet, in_len, dummy1, dummy2, dummy3)
```

Description:

ipsec_dec_cont

Used after ipsec_dec_strt_pin_X, to continue decrypting a packet on a buffer of data following the first part of the packet. Needed to handle jumbo packets. Setup and ending condition from prior use of ipsec_dec_strt_pin_X is required prior to invoking this sequence. In particular, the keys, config registers, hash address, etc. must remain intact when this sequence is started.

Note that this sequence does not use or depend on buffer specific constants, since the buffer specific values and setups are done in ipsec_dec_strt_pin_X

Calling the sequence:

```
load_ipsec_dec_cont (packet, in_len)
```

Table 2.119. crypto_load_ipsec_dec_cont parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of the start of ciphertext to decrypt
<i>in_len</i>	length of data to decrypt
<i>dummy1</i>	placeholder for consistency with other sequences
<i>dummy2</i>	placeholder for consistency with other sequences
<i>dummy3</i>	placeholder for consistency with other sequences Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.46 crypto_gen_ipsec_dec_cont

Prototype:

```
crypto_gen_ipsec_dec_cont(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in _crypto_library_load_dynamic in crypto_lib.uc

Table 2.120. crypto_gen_ipsec_dec_cont parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.47 crypto_load_ipsec_dec_cont_nw

Prototype:

```
crypto_load_ipsec_dec_cont_nw(cr_xfr, cr_ctx, packet, in_len, dummy1, dummy2, dummy3)
```

Description:

ipsec_dec_cont_nw

Similar to ipsec_dec_cont, but used on no cipher selection, so it does not have a wait for cipher. used for jumbo packets

Note that this sequence does not use or depend on buffer specific constants, since the buffer specific values and setups are done in ipsec_dec_nw_strt_pin_X

Calling the sequence:

```
load_ipsec_dec_cont_nw (packet, in_len)
```

Table 2.121. crypto_load_ipsec_dec_cont_nw parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of the start of ciphertext to decrypt
<i>in_len</i>	length of data to decrypt
<i>dummy1</i>	placeholder for consistency with other sequences
<i>dummy2</i>	placeholder for consistency with other sequences
<i>dummy3</i>	placeholder for consistency with other sequences Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.48 crypto_gen_ipsec_dec_cont_nw

Prototype:

```
crypto_gen_ipsec_dec_cont_nw(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in `_crypto_library_load_dynamic` in `crypto_lib.uc`

Table 2.122. `crypto_gen_ipsec_dec_cont_nw` parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.49 `crypto_load_ipsec_dec_end`

Prototype:

```
crypto_load_ipsec_dec_end(cr_xfr, cr_ctx, packet, in_len, hmac_resadr, hmac_keyadr,
hmac_keylen)
```

Description:

`ipsec_dec_end`

Used after `ipsec_dec_strt_pin_x`, and possibly `ipsec_dec_cont`, to complete decrypting a packet on the last buffer of data of the packet. Needed to handle jumbo packets. Setup and ending condition from prior use of `ipsec_dec_strt_pin_x` is required prior to invoking this sequence. In particular, the keys, config registers, etc. must remain intact when this sequence is started.

The hash key address and result address for the packet must be specified as parameters to this sequence.

The key address should be equal to the sram location for the hash key corresponding to the buffer (A,B,C or D) used for the 1st part of the packet. For e.g., if the 1st part of the packet was loaded using buffer A, the hash key address would be the same as provided in the constant value `sa_hmk_a`, since that is where the key was loaded.

The hash key result address should be the sram address corresponding to the hash result for the buffer being used when this macro (`ipsec_dec_end`) is invoked. For example, if using buffer B, the hash result address would be equal to the constant value set in `sa_hmr_b`

Calling the sequence:

```
crypto_load_ipsec_dec_end (packet, in_len, hmac_resadr, hmac_keyadr, hmac_keylen)
```

Table 2.123. `crypto_load_ipsec_dec_end` parameters

Name	Description
<i>cr_xfr</i>	
<i>cr_ctx</i>	
<i>packet</i>	crypto SRAM addr of strt of ciphertext to be decrypted
<i>in_len</i>	length of data to be decrypted
<i>hmac_resadr</i>	crypto SRAM address of the HMAC key calculation result
<i>hmac_keyadr</i>	crypto SRAM address of the start of the HMAC key

Name	Description
<i>hmac_keylen</i>	byte length of HMAC key, minus 1 Prepare transfer regs to load static (aka 'pinned') encryption sequence

2.7.3.50 crypto_gen_ipsec_dec_end

Prototype:

```
crypto_gen_ipsec_dec_end(core, desc)
```

Description:

Generate cmd sequence as constant data.

used in `_crypto_library_load_dynamic` in `crypto_lib.uc`

Table 2.124. crypto_gen_ipsec_dec_end parameters

Name	Description
<i>core</i>	crypto bulk core, 0 - 3 for NFP3XXX or 0 - 5 for NFP6XXX
<i>desc</i>	sequence 'descriptor', contains sram location address

2.7.3.51 crypto_gen_compr_constants

Prototype:

```
crypto_gen_compr_constants(base0, base1, base2, base3, base4)
```



Note

No description!

2.8 CRYPTO Operation

2.8.1 CRYPTO Operation Macros

Crypto logic support macros facilitating set up and run of precompiled compressed crypto instruction sequences which conform to the crypto-library standard.

Table 2.125. CRYPTO Operation and Defines

Defined	Definition
CRYPTO_DIR_ENCRYPT	0
CRYPTO_DIR_DECRYPT	1
CRYPTO_CIPHER_NONE	0
CRYPTO_CIPHER_NULL_RELOCATE	1
CRYPTO_CIPHER_NULL_DISCARD	2
CRYPTO_CIPHER_NULL_INSERT	3
CRYPTO_CIPHER_AES_128	4
CRYPTO_CIPHER_AES_192	5
CRYPTO_CIPHER_AES_256	6
CRYPTO_CIPHER_3DES	7
CRYPTO_CIPHER_2DES	8
CRYPTO_CIPHER_ARC4	9
CRYPTO_CIPHER_UEA1	10
CRYPTO_CIPHER_UEA2	11
CRYPTO_CIPHER_UIA1	12
CRYPTO_CIPHER_UIA2	13
CRYPTO_CIMODE_ECB	0
CRYPTO_CIMODE_CBC	1
CRYPTO_CIMODE_CFB	2
CRYPTO_CIMODE_OFB	3
CRYPTO_CIMODE_CTR	4
CRYPTO_HASH_NONE	0
CRYPTO_HASH_MD5	1
CRYPTO_HASH_GF128	2
CRYPTO_HASH_GF128_S	3
CRYPTO_HASH_SHA_1	4
CRYPTO_HASH_SHA_256	5
CRYPTO_HASH_SHA_384	6
CRYPTO_HASH_SHA_512	7
CRYPTO_HASH_CRC16	8
CRYPTO_HASH_CRC16_REVI	9
CRYPTO_HASH_CRC16_REVO	10
CRYPTO_HASH_CRC16_REVIO	11
CRYPTO_NDO_DEFAULT	0

Defined	Definition
CRYPTO_NDO_7B	1
CRYPTO_NDO_16LW	2
crypto_load_address(seq)	(seq & 0xFFFF)

2.8.3.1 crypto_get_core

Prototype:

```
crypto_get_core(cr_xfr, cr_ctx)
```

Description:

Allocate a core to a Dispatcher context.

Example:

```
.reg $cr[2]
#define CRYPTO_CTX 0
crypto_get_core ($cr, CRYPTO_CTX)
crypto_send (write_fifo, $cr, 0, 0)
```

This macro prepares transfer regs for get core operation

Table 2.126. crypto_get_core parameters

Name	Description
<i>cr_xfr</i>	Destination transfer register array, 2 registers
<i>cr_ctx</i>	Crypto Dispatcher context whose bulk core will get the data and run command

2.8.3.2 crypto_free_core

Prototype:

```
crypto_free_core(cr_xfr, cr_ctx)
```

Description:

Frees a previously allocated core from a Dispatcher context.

Example:

```
.reg $cr[2]
#define CRYPTO_CTX 0
crypto_free_core ($cr, CRYPTO_CTX)
crypto_send (write_fifo, $cr, 0, 0)
```

Table 2.127. crypto_free_core parameters

Name	Description
<i>cr_xfr</i>	Destination transfer register array, 2 registers
<i>cr_ctx</i>	Crypto Dispatcher context whose bulk core will be released

2.8.3.3 crypto_setup_configs

Prototype:

```
crypto_setup_configs(drn, cipher, mode, keysize, hash)
```

Description:

Create configuration values for setting up the crypto unit for commonly used ciphers and hashes.

Example:

```
crypto_setup_configs (enc, AES, CBC, 256, MD5)
```

Generates the following defined values, and no code:

`_crypt_select` provides the `cfg_pulse_R` and `cfg_sel_R` fields suitable for use in a `Crypt_Setup_Regs_0` instruction; includes bits to clear hash and cipher states `_crypt_select_no_clear` same as `_crypt_select` but does not include cipher and hash state-clearing bits `_crypt_modes` provides the `cfg_Crypt0_R` field for use in a `crypt_setup_regs_0` instruction `_crypt_hmac_len` provides the length of the hmac, -1 byte `_crypt_iv_len` provides the length of the iv

Table 2.128. crypto_setup_configs parameters

Name	Description
<i>drn</i>	CONST, selects direction ('encrypt' or 'decrypt' or 'enc' or 'dec')
<i>cipher</i>	CONST, selects cipher type: 'none', 'null', 'aes', or 'des' supported
<i>mode</i>	CONST, selects block cipher mode: 'ECB' or 'CBC' supported
<i>keysize</i>	CONST, bit length of key: 128, 192, 256 supported for AES; use '--' for DES
<i>hash</i>	CONST, selects hash type: MD5, SHA1, SHA256, SHA384, SHA512 supported

2.8.3.4 crypto_setup_configs

Prototype:

```
crypto_setup_configs(_hlen, _ivlen, _select, _modes, _dir, _cipher, _mode, _hash)
```

Description:

Create configuration values for setting up the crypto unit for commonly used ciphers and hashes.

Example:


```
.reg hlen, ivlen, select, mode
.reg dir, cipher, cimode, hash
move (dir, CRYPTO_DIR_ENCRYPT)
move (cipher, CRYPTO_CIPHER_AES_256
etc.
crypto_setup_configs (hlen, ivlen, select, mode, \
                      dir, cipher, cimode, keylen, hash)
```

Similar to above macro, but uses GPR's instead of constants

Table 2.129. crypto_setup_configs parameters

Name	Description
<i>_hlen</i>	GPR, returns icv/hash length, -1 byte
<i>_ivlen</i>	GPR, returns iv length
<i>_select</i>	GPR, returns cfg_pulse_R and cfg_sel_R fields suitable for use in a Crypt_Setup_Regs_0 instruction; includes bits to clear hash and cipher states
<i>_modes</i>	GPR, returns cfg_Crypt0_R field for use in a Crypt_Setup_Regs_0 instruction
<i>_dir</i>	GPR, selects direction, one of CRYPTO_DIR_*
<i>_cipher</i>	GPR, selects cipher type: one of CRYPTO_CIPHER_*
<i>_mode</i>	GPR, selects block cipher mode: one of CRYPT_CIMODE_*
<i>_hash</i>	GPR, selects hash type: one of CRYPTO_HASH_*

2.8.3.5 crypto_send

Prototype:

```
crypto_send(cmd, xfr, a, b, n, xsig)
```

Description:

Send a specified number of 8-byte words to the Crypto unit.

Example:

```
.sig s
.reg $cmd[4]
#define_eval xfer_length 2
crypto_send (write_fifo, $cmd, --, 0, xfer_length, s)
```

Table 2.130. crypto_send parameters

Name	Description
<i>cmd</i>	type of transfer, must be either "write" or "write_fifo"
<i>xfr</i>	transfer register array to use in the write
<i>a</i>	crypto sram address to be written (only used in "write" case) can be constant or GPR use -- for "write_fifo" operations

Name	Description
<i>b</i>	offset to crypto sram address a (only used in "write" case) must be constant use -- or 0 for "write_fifo" operations
<i>n</i>	optional number of pairs of registers from xfer will be sent to the crypto unit must be constant
<i>xsig</i>	optional ME signal to be used for the I/O operation use -- to generate a local signal and wait for it use a signal name to do sig_done[] on that signal and return without waiting use* nosend* to prevent send

2.8.3.6 crypto_send

Prototype:

```
crypto_send(cmd, xfer, a, b, n)
```

Description:

Send a specified number of 8-byte words to the Crypto unit.

Invokes crypto_send, above, with parameter xsig set to "--"

2.8.3.7 crypto_send

Prototype:

```
crypto_send(cmd, xfer, a, b)
```

Description:

Send a specified number of 8-byte words to the Crypto unit.

Invokes crypto_send, above, with parameters: xsig set to "--" n set to _crypto_transfer_length

_crypto_transfer_length is set by other macros in this library.

2.8.3.8 crypto_send

Prototype:

```
crypto_send(xfer)
```

Description:

Send a specified number of 8-byte words to the Crypto unit.

Invokes crypto_send, above, with parameters: cmd set to "write_fifo" a set to "--" (ignored for write_fifo) b set to "--" (ignored for write_fifo) xsig set to "--" n set to _crypto_transfer_length

_crypto_transfer_length is set by other macros in this library.

2.8.3.9 crypto_build_xfer_reg

Prototype:

```
crypto_build_xfer_reg(res, lo, hi)
```

Description:

Load a 32-bit value in a register from two 16-bit values.

Table 2.131. crypto_build_xfer_reg parameters

Name	Description
<i>res</i>	GPR destination
<i>lo</i>	GPR or CONSTANT, lower 16 bits
<i>hi</i>	GPR or CONSTANT, upper 16 bits

2.8.3.10 crypto_load_library_sequence

Prototype:

```
crypto_load_library_sequence(cr_xfr, seq, core, vars, sigspec)
```

Description:

Load the parameters for a library sequence into crypto SRAM to prepare for running the sequence.

Calling sequence to execute a CIB sequence that has been placed into the compressed library:

crypto_load_<sequence_name>() - used to place parameter data to be passed to the sequence, into transfer registers
 crypto_load_library_sequence() - writes the parameters to crypto SRAM
 crypto_run_library_sequence() - transfers parameters (and dynamic code) to bulk unit and runs the sequence

Table 2.132. crypto_load_library_sequence parameters

Name	Description
<i>cr_xfr</i>	Destination transfer register array (up to 4 registers will be used beginning at cr_xfr[0])
<i>seq</i>	Name of sequence to be run
<i>core</i>	Core number used when the sequence was loaded into the library
<i>vars</i>	Crypto SRAM address to be used to hold parameter variables (static sequences only) Ignored for 'dynamic' loaded sequences (-- can be used in this case) Can be constant or GPR
<i>sigspec</i>	Optional signal specification, determines when the macro returns: -- : creates a local signal and waits for it to be raised signal name : returns immediately, caller is responsible for waiting for named signal missing : same as '--'

2.8.3.11 crypto_load_library_sequence

Prototype:

```
crypto_load_library_sequence(cr_xfr, seq, core, vars)
```

Description:

Load the parameters for a library sequence into crypto SRAM to prepare for running the sequence.

Invokes crypto_load_library_sequence, above, with parameters: sigspec set to "--"

2.8.3.12 crypto_run_library_sequence

Prototype:

```
crypto_run_library_sequence(cr_xfr, cr_ctx, seq, core, vars, base, sigspec)
```

Description:

Run a sequence previously loaded into the compressed library via library_load_static or library_load_dynamic.

Table 2.133. crypto_run_library_sequence parameters

Name	Description
<i>cr_xfr</i>	Destination transfer register array (up to 4 registers will be used beginning at cr_xfr[0])
<i>cr_ctx</i>	Crypto Dispatcher context which will be used to run the sequence
<i>seq</i>	Name of sequence to be run
<i>core</i>	Core number used when the sequence was loaded into the library
<i>vars</i>	Crypto SRAM address to be used to hold parameter variables (static sequences only) Ignored for 'dynamic' loaded sequences (-- can be used in this case) Can be constant or GPR
<i>base</i>	Optional base index (0 or 1) to pass to CIB run instruction (NFP6xxxC only) Use 0 for NFP32xxC processor applications Can be constant or GPR
<i>sigspec</i>	Optional signal specification, determines when the macro returns: -- : creates a local signal and waits for it to be raised signal name : returns immediately, caller is responsible for waiting for named signal missing : same as '--'

2.8.3.13 crypto_run_library_sequence

Prototype:

```
crypto_run_library_sequence(cr_xfr, cr_ctx, seq, core, vars, base)
```

Description:

Run a sequence previously loaded into the compressed library via library_load_static or library_load_dynamic.

Invokes `crypto_run_library_sequence`, above, with parameters: sigspec set to "--"

2.8.3.14 `crypto_run_library_sequence`

Prototype:

```
crypto_run_library_sequence(cr_xfr, cr_ctx, seq, core, vars)
```

Description:

Run a sequence previously loaded into the compressed library via `library_load_static` or `library_load_dynamic`.

Invokes `crypto_run_library_sequence`, above, with parameters: base set to "0" sigspec set to "--"

2.8.3.15 `crypto_get_island_and_PPID7`

Prototype:

```
crypto_get_island_and_PPID7(out_island, out_PPID7)
```

Description:

Generate island and PPID7 values.

Used in support of signal-related macros in this library.

Table 2.134. `crypto_get_island_and_PPID7` parameters

Name	Description
<i>out_island</i>	Destination register to contain the island number on return For NFP3xxxC processors '--' can be used, otherwise the supplied GPR will be set to zero
<i>out_PPID7</i>	Destination register to contain the PPID value, left-shift by 7 bits, on return

2.8.3.16 `crypto_send_signal`

Prototype:

```
crypto_send_signal(cr_xfr, cr_ctx, me_ctx, island, PPID7, sigspec, cmdsigspec)
```

Description:

Enqueue and send a return-signal operation, with flexible signal support.

If sigspec is set to "--", cmdsigspec parameter is ignored and "--" is assumed

Table 2.135. `crypto_send_signal` parameters

Name	Description
<i>cr_xfr</i>	Destination transfer register array (<code>cr_xfr[0]</code> and <code>cr_xfr[1]</code> will be used)

Name	Description
<i>cr_ctx</i>	Crypto Dispatcher context which will generate the signal
<i>me_ctx</i>	ME context to receive the signal can be constant or GPR
<i>island</i>	island number (NFP6xxxC only) [as returned by get_island_and_PPID7()] use -- for NFP3xxxC applications
<i>PPID7</i>	local PPID left-shifted by 7 bits [as returned by get_island_and_PPID7()]
<i>sigspec</i>	ME signal to be raised by the crypto unit use -- to generate a local signal and wait for it use a signal name to enqueue that signal and return without waiting
<i>cmdsigspec</i>	ME signal to be raised when the command has been delivered to the crypto FIFO use -- to generate a local signal and wait for it use a signal name to use that signal and return without waiting

2.8.3.17 crypto_send_signal

Prototype:

```
crypto_send_signal(cr_xfr, cr_ctx, me_ctx, island, PPID7, sigspec)
```

Description:

Enqueue and send a return-signal operation, with flexible signal support.

Invokes crypto_send_signal, above, with parameters: cmdsigspec set to "--"

2.8.3.18 crypto_send_signal

Prototype:

```
crypto_send_signal(cr_xfr, cr_ctx, me_ctx, sigspec, cmdsigspec)
```

Description:

Enqueue and send a return-signal operation to the local thread.

Invokes crypto_send_signal, above, with parameters: signal generated from get_island_and_PPID7 PPID7 generated from get_island_and_PPID7

2.8.3.19 crypto_send_signal

Prototype:

```
crypto_send_signal(cr_xfr, cr_ctx, me_ctx, sigspec)
```

Description:

Enqueue and send a return-signal operation to the local thread.

Invokes `crypto_send_signal`, above, with parameters: `cmdsigspec` set to "--" signal generated from `get_island_and_PPID7` PPID7 generated from `get_island_and_PPID7`

2.8.3.20 `crypto_send_done_signal`

Prototype:

```
crypto_send_done_signal(cr_xfr, cr_ctx, me_ctx, dispatch, island, PPID7, sigspec,
iosigspec)
```

Description:

Enqueue and send a return-signal operation to a selected dispatch queue.

For use with NFP6xxx series processors only.

Table 2.136. `crypto_send_done_signal` parameters

Name	Description
<i>cr_xfr</i>	Destination transfer register array (<code>cr_xfr[0]</code> and <code>cr_xfr[1]</code> will be used)
<i>cr_ctx</i>	Crypto Dispatcher context which will be used to run the sequence
<i>me_ctx</i>	ME context to receive the signal can be constant or GPR 0 = normal / Bulk Core queue, 1 = 'chaining' / DMA Out queue
<i>dispatch</i>	
<i>island</i>	island number [as returned by <code>get_island_and_PPID7()</code>]
<i>PPID7</i>	local PPID left-shifted by 7 bits [as returned by <code>get_island_and_PPID7()</code>]
<i>sigspec</i>	ME signal to be raised by the crypto unit use -- to generate a local signal and wait for it use a signal name to enqueue that signal and return without waiting
<i>iosigspec</i>	ME signal to use for the I/O (write_fifo) operation use -- to generate a local signal and wait for it use a signal name to enqueue that signal and return without waiting

2.8.3.21 `crypto_send_done_signal`

Prototype:

```
crypto_send_done_signal(cr_xfr, cr_ctx, me_ctx, dispatch, island, PPID7, sigspec)
```

Description:

Enqueue and send a return-signal operation to a selected dispatch queue.

Invokes `crypto_send_done_signal`, above, with parameters: `iosigspec` set to "--"

2.8.3.22 `crypto_send_done_signal`

Prototype:

```
crypto_send_done_signal(cr_xfr, cr_ctx, me_ctx, dispatch, sigspec, iosigspec)
```

Description:

Enqueue and send a return-signal operation to a selected dispatch queue.

Invokes `crypto_send_done_signal`, above, with parameters: signal generated from `get_island_and_PPID7` PPID7 generated from `get_island_and_PPID7`

2.8.3.23 `crypto_send_done_signal`

Prototype:

```
crypto_send_done_signal(cr_xfr, cr_ctx, me_ctx, dispatch, sigspec)
```

Description:

Enqueue and send a return-signal operation to a selected dispatch queue.

Invokes `crypto_send_done_signal`, above, with parameters: `iosigspec` set to "--" signal generated from `get_island_and_PPID7` PPID7 generated from `get_island_and_PPID7`

2.8.3.24 `crypto_init_CIB_setup`

Prototype:

```
crypto_init_CIB_setup(stat, dyn, code)
```

Description:

Initialize the process of preparing the bulk core for compressed-library loading and usage.

Table 2.137. `crypto_init_CIB_setup` parameters

Name	Description
<i>stat</i>	starting location in crypto SRAM where 'static' sequences will be staged for transfer to bulk core CIB space. This is a temporary buffer that should be available for exclusive use by one crypto Dispatcher context during the initialization and library loading process. Static sequences reside in the CIB space of each core, for the life of the program, and as such can be accessed immediately on-demand as needed.
<i>dyn</i>	starting location in crypto SRAM where 'dynamic' sequences will be loaded. Dynamic sequences are code sequences that are kept in crypto SRAM long-term (i.e. for the life of the program), and are copied to the core CIB only when needed. They allow the use of a larger compressed library since the compressed code for all the sequences reside in SRAM rather than CIB space.
<i>code</i>	starting location within the CIB space, where static compressed sequences will be loaded. This is a word address (in units of 4-byte words) and must be set to a value beyond the space reserved for constants and variables.

2.8.3.25 crypto_library_load_constants

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx, n, base0, base1, base2, base3, base4, base5)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Constants are presented in the crypto-library file similarly to a 'static' CIB sequence, with a sequence name of 'compr_constants'. library_load_constants() must be called first after init_CIB_setup(), before static-mode sequences are loaded via library_load_static(). Note that library_load_constants() must be called even if no static sequences are going to be loaded, because constants are also used by dynamic sequences.

Table 2.138. crypto_library_load_constants parameters

Name	Description
<i>cr_xfr</i>	Destination transfer register array. Normally 2 registers will be used beginning at cr_xfr[0], more may be used depending on number of constants defined.
<i>cr_ctx</i>	Crypto Dispatcher context to do the transfer (the context must already have a bulk core allocated to it via get_core()).
<i>n</i>	Number of constants to load, valid values 0 - 6
<i>base0</i>	
<i>base1</i>	
<i>base2</i>	
<i>base3</i>	
<i>base4</i>	
<i>base5</i>	

2.8.3.26 crypto_library_load_constants

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Invokes crypto_library_load_constants, above, with parameters: n set to 0 b0-b5 set to --

2.8.3.27 crypto_library_load_constants

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx, b0)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Invokes `crypto_library_load_constants`, above, with parameters: n set to 1 b1-b5 set to --

2.8.3.28 `crypto_library_load_constants`

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx, b0, b1)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Invokes `crypto_library_load_constants`, above, with parameters: n set to 2 b2-b5 set to --

2.8.3.29 `crypto_library_load_constants`

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx, b0, b1, b2)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Invokes `crypto_library_load_constants`, above, with parameters: n set to 3 b3-b5 set to --

2.8.3.30 `crypto_library_load_constants`

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx, b0, b1, b2, b3)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Invokes `crypto_library_load_constants`, above, with parameters: n set to 4 b4-b5 set to --

2.8.3.31 `crypto_library_load_constants`

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx, b0, b1, b2, b3, b4)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Invokes `crypto_library_load_constants`, above, with parameters: `n` set to 5 `b5` set to --

2.8.3.32 `crypto_library_load_constants`

Prototype:

```
crypto_library_load_constants(cr_xfr, cr_ctx, b0, b1, b2, b3, b4, b5)
```

Description:

Load CIB memory of a crypto bulk core unit with constants needed for compressed code sequences.

Invokes `crypto_library_load_constants`, above, with parameters: `n` set to 6

2.8.3.33 `crypto_library_load`

Prototype:

```
crypto_library_load(LOAD_TYPE, xfr, ctx, seq, core)
```

Description:

Add a crypto-library sequence to the run-time-available CIB sequences.

Sequences can be loaded for 'static' or 'dynamic' use models. This macro calls either `_crypto_library_load_static` or `_crypto_library_load_dynamic`, depending on the `LOAD_TYPE` parameter.

Table 2.139. `crypto_library_load` parameters

Name	Description
<code>LOAD_TYPE</code>	Use either <code>CRYPTO_LOAD_STATIC</code> or <code>CRYPTO_LOAD_DYNAMIC</code> to select run type
<code>xfr</code>	
<code>ctx</code>	
<code>seq</code>	Name of sequence to be added to the run-time library.
<code>core</code>	Bulk core number must be a constant. This is only used if a different set of static sequences is being loaded to each core, a non-typical situation. The value must be 0-3 for NFP32xxxC, 0-5 for NFP6xxxC.

2.8.3.34 `crypto_library_load`

Prototype:

```
crypto_library_load(LOAD_TYPE, xfr, ctx, seq)
```

Description:

Add a crypto-library sequence to the run-time-available CIB sequences.

Invokes `crypto_library_load`, above, with parameters: core set to 0

2.8.3.35 `crypto_setup_dma_desc_cycle1`

Prototype:

```
crypto_setup_dma_desc_cycle1(dest_lo, dest_hi, ctx, xlen, prefetch, addr_lo, addr_hi)
```

Description:

Generate a crypto DMA instruction (cycle 1).

Example:

```
.reg $io[4]
.xfer_order $io
crypto_setup_dma_desc_cycle1 ($io[0], $io[1], ...)
crypto_setup_dma_desc_cycle2 ($io[2], $io[3], ...)
crypto [write_fifo, $io[0], x, 0, 2]
```

This macro generatates cycle 1 of an NFPxxxx crypto DMA operation. It is intended to be used in conjunction with the `crypto_setup_dma_desc_cycle2` macro.

Table 2.140. `crypto_setup_dma_desc_cycle1` parameters

Name	Description
<i>dest_lo</i>	register to receive the result, low 32-bits (this can be a transfer register)
<i>dest_hi</i>	register to receive the result, high 32-bits (this can be a transfer register)
<i>ctx</i>	crypto Dispatcher context to receive the resulting DMA instruction (GPR or CONST)
<i>xlen</i>	DMA transfer length, minus 1, in 8-byte words (0 to 255 valid) (0 transfers 8 bytes) (GPR or CONST)
<i>prefetch</i>	set to 1 for prefetch/instruction chaining, 0 otherwise (GPR or CONST)
<i>addr_lo</i>	system memory address, low word (GPR or CONST)
<i>addr_hi</i>	system memory address, high word (GPR or CONST)

2.8.3.36 `crypto_setup_dma_desc_cycle2`

Prototype:

```
crypto_setup_dma_desc_cycle2(dest_lo, dest_hi, ctx, dir, token, action, target, w32,
chain, last, cr_addr)
```

Description:

Generate a crypto DMA instruction (cycle 2).

Example:

```
.reg $io[4]
.xfer_order $io
crypto_setup_dma_desc_cycle1 ($io[0], $io[1], ...)
crypto_setup_dma_desc_cycle2 ($io[2], $io[3], ...)
crypto [write_fifo, $io[0], x, 0, 2]
```

This macro generatates cycle 2 of an NFPxxxx crypto DMA operation. It is intended to be used in conjunction with the crypto_setup_dma_desc_cycle2 macro.

Table 2.141. crypto_setup_dma_desc_cycle2 parameters

Name	Description
<i>dest_lo</i>	register to receive the result, low 32-bits (this can be a transfer register)
<i>dest_hi</i>	register to receive the result, high 32-bits (this can be a transfer register)
<i>ctx</i>	crypto Dispatcher context to receive the resulting DMA instruction (GPR or CONST)
<i>dir</i>	DMA transfer direction; 0=inbound, 1=outbound (GPR or CONST)
<i>token</i>	token field for CPP transactions driven by DMA (GPR or CONST)
<i>action</i>	action field for CPP transactions driven by DMA (GPR or CONST)
<i>target</i>	target field for CPP transactions driven by DMA (GPR or CONST)
<i>w32</i>	target width; 0=64-bit, 1=32-bit (GPR or CONST)
<i>chain</i>	enable instruction-chaining (GPR or CONST)
<i>last</i>	mark this operation as the 'last' DMA in the instruction chain (GPR or CONST)
<i>cr_addr</i>	crypto SRAM address (upper 13 bits only; bottom 3 bits are assumed to be zero) (GPR or CONST)

2.8.3.37 crypto_dma_out

Prototype:

```
crypto_dma_out(cr, dma_ctx, chain, last, sram_addr, addr32_lo, addr32_hi, nbytes, sent_sig)
```

Description:

Set up a Crypto DMA operation from Crypto SRAM to Memory (target ID 7).

Table 2.142. crypto_dma_out parameters

Name	Description
<i>cr</i>	transfer register array to build the instructions into
<i>dma_ctx</i>	crypto dispatcher context which will handle the DMA operation
<i>chain</i>	constant =1 for chained operation, =0 for non-chained
<i>last</i>	constant =1 if this is the last DMA in the chain, =0 otherwise (only valid if chain=1)
<i>sram_addr</i>	address in Crypto SRAM where the data will be copied from (GPR or constant)
<i>addr32_lo</i>	address in DRAM memory where the data will be copied to, low word

Name	Description
<i>addr32_hi</i>	address in DRAM memory where the data will be copied to, high word
<i>nbytes</i>	number of bytes to be transferred (must be a multiple of 8) (GPR or constant)
<i>sent_sig</i>	

2.8.3.38 crypto_dma_out

Prototype:

```
crypto_dma_out(cr, dma_ctx, chain, last, sram_addr, addr32_lo, addr32_hi, nbytes)
```

Description:

Set up a Crypto DMA operation from Crypto SRAM to Memory (target ID 7).

Invokes `crypto_dma_out`, above, with parameters: `sent_sig` set to "--"

2.8.3.39 crypto_dma_in

Prototype:

```
crypto_dma_in(cr, dma_ctx, chain, last, sram_addr, addr32_lo, addr32_hi, nbytes, sent_sig)
```

Description:

Set up a Crypto DMA operation from Memory (target ID 7) to Crypto SRAM.

Table 2.143. crypto_dma_in parameters

Name	Description
<i>cr</i>	transfer register array to build the instructions into
<i>dma_ctx</i>	crypto dispatcher context which will handle the DMA operation
<i>chain</i>	constant =1 for chained operation, =0 for non-chained
<i>last</i>	constant =1 if this is the last DMA in the chain, =0 otherwise (only valid if chain=1)
<i>sram_addr</i>	address in Crypto SRAM where the data will be copied to (GPR or constant)
<i>addr32_lo</i>	address in DRAM memory where the data will be copied from, low word
<i>addr32_hi</i>	address in DRAM memory where the data will be copied from, high word
<i>nbytes</i>	number of bytes to be transferred (must be a multiple of 8) (GPR or constant)
<i>sent_sig</i>	optional signal to set when the instruction has been written to the FIFO use -- to generate an internal signal and wait for it before returning

2.8.3.40 crypto_dma_in

Prototype:

```
crypto_dma_in(cr, dma_ctx, chain, last, sram_addr, addr32_lo, addr32_hi, nbytes)
```

Description:

Set up a Crypto DMA operation from Memory (target ID 7) to Crypto SRAM.

Invokes `crypto_dma_in`, above, with parameters: `send_sig` set to "--"

2.8.3.41 `crypto_non_dma_out`

Prototype:

```
crypto_non_dma_out(_sram_addr, _addr32_lo, _addr32_hi, _nbytes, _MODE)
```

Description:

Transfer data from Crypto SRAM to Memory without using DMA.

default mode does a nonaligned transfer at beginning and end, and 8 byte aligned transfers in between, as needed

max 7 byte mode does a single non-aligned transfer of 1-7 bytes

max 16 lword mode does a single transfer of 1 to 16 lwords, data must be aligned on 8 byte boundary in sram, but may be on any alignment in bulk memory

Needed when memory address is not 8 byte aligned and/or transfer length is not an 8 byte multiple, because Crypto DMA does not support these transfers. Also needed to transfer crypto hash data.

Table 2.144. `crypto_non_dma_out` parameters

Name	Description
<code>_sram_addr</code>	byte address in Crypto SRAM where the data will be copied from (GPR or constant)
<code>_addr32_lo</code>	byte address in memory where the data will be copied to, low word
<code>_addr32_hi</code>	address in memory where the data will be copied to, high word
<code>_nbytes</code>	number of bytes to transfer
<code>_MODE</code>	constant, mode of operation, one of: CRYPTO_NDO_DEFAULT unlimited transfer length & alignment CRYPTO_NDO_7B max 7 byte transfer, any alignment CRYPTO_NDO_16LW max 16 lwords transfer, sram aligned

2.9 CRYPTO Threads Operation

2.9.1 CRYPTO Threads Operation Macros

Crypto logic support macros facilitating encryption, decryption, and authentication of ip packets. Utilizes multiple crypto cores, packet buffers, and threads in a pipeline in order to maximize utilization of crypto hardware and maximize packet processing performance. Intended to be run on one or more microengines in the crypto island of NFP6000

Table 2.145. CRYPTO Threads Operation and Defines

Defined	Definition
CRYPTO_NUM_THREADS	6
CRYPTO_START_CTX	0
CRYPTO_RING_CTM	0
CRYPTO_RING_EMU0	1
CRYPTO_RING_EMU1	2
CRYPTO_RING_EMU2	3
CRYPTO_RING_SIZE_128	128
CRYPTO_RING_SIZE_256	256
CRYPTO_RING_SIZE_512	512
CRYPTO_RING_SIZE_1K	1024
CRYPTO_RING_SIZE_2K	2048
CRYPTO_RING_SIZE_4K	4096
CRYPTO_RING_SIZE_8K	8192
CRYPTO_RING_SIZE_16K	16384
CRYPTO_RING_SIZE_32K	32768
CRYPTO_RING_SIZE_64K	64536
OVERRIDE_RESPONSE_RING	0
RESPONSE_RING_TYPE	CRYPTO_RING_CTM
BUF_RING_TYPE	CRYPTO_RING_CTM
BUF_RING_NUM	0
BUF_RING_ISLAND_ID	0x00
INIT_FLAG_USE_ALLOC_MEM	1
INIT_FLAG_ALLOCATION_FIXED	0
INIT_FLAG_ADDR	0x00000020

Defined	Definition
INIT_FLAG_ISLAND	i12
ENABLE_SA_FLUSH	0
REQ_RING_TYPE	CRYPTO_RING_CTM
REQ_RING_USE_ONE_EMU_RING	0
REQ_RING_SIZE	CRYPTO_RING_SIZE_2K
REQ_RING_NUM	0
REQ_RING_USE_ALLOC_MEM	1
REQ_RING_ALLOCATION_FIXED	0
REQ_RING_ADDR	0x00000000
REQ_RING_DESC_ADDR	(REQ_RING_ADDR + (REQ_RING_SIZE << 2))
_SIZE	(REQ_RING_SIZE << 2)
OVERRIDE_UPDATE_COUNTER	0
COUNTER_MEMORY_TYPE_CLS	0
COUNTER_MEMORY_TYPE_CTM	1
COUNTER_MEMORY_TYPE	COUNTER_MEMORY_TYPE_CLS
COUNTER_USE_ALLOC_MEM	1
COUNTER_ALLOCATION_FIXED	0
COUNTER_BASE_ADDR	0x1000
COUNTER_MEM_SIZE	((CRYPTO_COUNTER_LENGTH/8) * CRYPTO_CNTR_MAX)
ENABLE_JUMBO_PKTS	1
ENABLE_ERROR_CHECKS	0
ENABLE_ANTI_REPLAY	1
ENABLE_DETAILED_COUNTERS	0
ENABLE_DEBUG_COUNTERS	0
HALT_ON_ERROR	0

2.9.3.1 crypto_threads_input

Prototype:

```
crypto_threads_input(_me_ctx, _crypto_ctx, _in_ring_type, _in_ring_num)
```

Description:

Crypto input thread.

Input thread dedicated to a single crypto context / core. Typically run on even #'d thread, i.e. me threads 0,2,4,6 while corresponding output thread is run on odd #'d threads, i.e. me threads 1,3,5,7

Utilizes 1/6 of crypto sram buffer; each crypto core/ctx is allocated 1/6 of the sram buffer. Within each 1/6 of the sram buffer, space is allocated for 4x (2KB packet buffer, SA struct, and temp area)

The 4 packet buffers are used to create a pipeline to keep the crypto core active. The 4 buffers are in one of the following states:

- data being dma'd from system memory into crypto sram buffer
- ready to be operated on by crypto core when current crypto operation done
- being operated on by crypto core
- being dma'd from crypto sram to system memory

Source packet data may be stored contiguously in one buffer, or split into two or three buffers. One, two, or three dma transfers will be performed to move the buffers into crypt sram for encryption/decryption. The buffers will be placed in crypto sram contiguously, starting with data at Start of Packet Address, followed by Continuation of Packet Address, followed by End of Packet Address. If a length field is 0, no dma will occur for the corresponding buffer. (addresses and length are part of the request ring entry, refer to request ring entry in code file for format)

Table 2.146. crypto_threads_input parameters

Name	Description
<code>_me_ctx</code>	GPR, me context of this thread
<code>_crypto_ctx</code>	GPR, crypto core context to be used by this thread. One context is used per core, so context will be equal to the core # in use by this thread
<code>_in_ring_type</code>	CONST, ring type. one of CRYPTO_RING_CTM, CRYPTO_RING_IMU, CRYPTO_RING_EMU
<code>_in_ring_num</code>	CONST or GPR, ring number of input request ring.

2.9.3.2 crypto_threads_output

Prototype:

```
crypto_threads_output(_me_ctx, _crypto_ctx)
```

Description:

Crypto output thread.

Output thread dedicated to a single crypto context / core. Typically run on odd #'d thread, i.e. me threads 1,3,5,7, while corresponding input thread is run on even #'d threads, i.e. me threads 0,2,4,6.

See above re crypto_threads_input for buffer utilization description

Table 2.147. crypto_threads_output parameters

Name	Description
<code>_me_ctx</code>	GPR, me context of this thread

Name	Description
<code>_crypto_ctx</code>	GPR, crypto core context to be used by this thread. One context is used per core, so context will be equal to the core # in use by this thread

2.9.3.3 crypto_threads_init

Prototype:

```
crypto_threads_init()
```

Description:

Initialize crypto threads.

Example:

```
crypto_threads_init()
```

Initializes input request ring, initializes crypto cores, and initializes and starts crypto input and output threads. CRYPTO_NUM_THREADS, CRYPTO_START_CTX determine how many threads are started and what contexts they are started on.

The Request Ring params are defined as REQ_RING_xyz, above.

Typically used via a wrapper microcode file that may override some of the parameters defined above, invokes this macro, and does nothing else.

Threads that wish to send requests to the request ring may use macro: crypto_threads_wait_init_flag() which loops waiting for crypto_threads_init to complete.

2.10 DRAM Access

2.10.1 DRAM Access Macros

DRAM memory specific Access Macros

2.10.2.1 dram_mask_write

Prototype:

```
dram_mask_write(in_data, in_dram_addr, in_addr_offset, in_byte_mask, REQ_SIG,  
in_wakeup_sigs, in_reserved)
```

Description:

Write bytes selected by `in_byte_mask` to a DRAM quadword.



Note

Limitations: Input data must be in transfer registers.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 3 to 8

Table 2.148. dram_mask_write parameters

Name	Description
<code>in_data</code>	Transfer register containing data to be written to DRAM
<code>in_dram_addr</code>	DRAM address. Register or constant. Added to <code>in_addr_offset</code> to form DRAM address used in transfer.
<code>in_addr_offset</code>	DRAM address. Register or constant. Added to <code>in_dram_addr</code> to form DRAM address used in transfer.
<code>in_byte_mask</code>	Register or constant containing an 8 bit mask that indicates which bytes to write. The bits in the mask correspond to bytes, left to right. For example, 0x80 specifies the leftmost byte, and 0x1 specifies the rightmost byte.
<code>REQ_SIG</code>	Requested signal.
<code>in_wakeup_sigs</code>	List of signals causing thread to swap/wakeup.
<code>in_reserved</code>	Reserved for future use. Pass: as the value of this parameter.

2.10.2.2 dram_mask_write

Prototype:

```
dram_mask_write(in_data, in_addr_1, in_addr_2, in_addr_3, in_byte_mask, REQ_SIG,
in_wakeup_sigs, in_reserved)
```

Description:

Write bytes selected by `in_byte_mask` to a DRAM quadword - 40-bit addressing version.



Note

Limitations: Input data must be in transfer registers. **Address specification** takes the form "reg_or_const1, <<8, reg_or_const2" or "reg_or_const1, reg_or_const2, <<8".

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 3 to 8

Table 2.149. dram_mask_write parameters

Name	Description
<i>in_data</i>	Transfer register containing data to be written to DRAM
<i>in_addr_1</i>	Address specification - see note.
<i>in_addr_2</i>	Address specification - see note.
<i>in_addr_3</i>	Address specification - see note.
<i>in_byte_mask</i>	Register or constant containing an 8 bit mask that indicates which bytes to write. The bits in the mask correspond to bytes, left to right. For example, 0x80 specifies the leftmost byte, and 0x1 specifies the rightmost byte.
<i>REQ_SIG</i>	Requested signal.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup.
<i>in_reserved</i>	Reserved for future use. Pass: as the value of this parameter.

2.10.2.3 dram_rbuf_read

Prototype:

```
dram_rbuf_read(in_dram_addr, in_dram_addr_offset, in_rbuf_addr, in_rbuf_addr_offset,
in_qw_count, REQ_SIG, in_wakeup_sigs, in_reserved)
```

Description:

Copy *in_qw_count* quadwords from RBUF to DRAM.

RBUF is the interface buffer for data received from the network.



Note

Limitations: Granularity of transfer count is quadwords.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 2 to 10

Table 2.150. dram_rbuf_read parameters

Name	Description
<i>in_dram_addr</i>	DRAM address. Register or constant. Added to <i>in_dram_addr_offset</i> to form DRAM address used in transfer.
<i>in_dram_addr_offset</i>	DRAM address. Register or constant. Added to <i>in_dram_addr</i> to form DRAM address used in transfer.
<i>in_rbuf_addr</i>	RBUF address. Register or constant. Added to <i>in_rbuf_addr_offset</i> to form RBUF address used in transfer.

Name	Description
<i>in_rbuf_addr_offset</i>	RBUF address. Register or constant. Added to <i>in_rbuf_addr</i> to form RBUF address used in transfer.
<i>in_qw_count</i>	Register or constant. Number of quadwords to transfer from RBUF to DRAM
<i>REQ_SIG</i>	Requested signal.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup.
<i>in_reserved</i>	Reserved for future use. Pass: as the value of this parameter.

Availability:

IXP Indirect Reference Mode

2.10.2.4 dram_read

Prototype:

```
dram_read(out_data, in_dram_addr, in_addr_offset, in_qw_count, REQ_SIG, in_wakeup_sigs, in_reserved)
```

Description:

Read *in_qw_count* quadwords from DRAM.



Note

Limitations: Granularity of transfer count is quadwords.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 1 to 7 (indirect read (count in GPR) and $8 \leq \text{count} \leq 15$)

Table 2.151. dram_read parameters

Name	Description
<i>out_data</i>	Transfer register that will contain read data
<i>in_dram_addr</i>	DRAM address. Register or constant. Added to <i>in_addr_offset</i> to form DRAM address used in transfer.
<i>in_addr_offset</i>	DRAM address. Register or constant. Added to <i>in_dram_addr</i> to form DRAM address used in transfer.
<i>in_qw_count</i>	Register or constant. Number of quadwords to read. The maximum quadword count is 16.
<i>REQ_SIG</i>	Requested signal.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup.
<i>in_reserved</i>	Reserved for future use. Pass: as the value of this parameter.

2.10.2.5 dram_read

Prototype:

```
dram_read(out_data, in_dram_addr, in_addr_offset, in_qw_count, REQ_SIG, in_wakeup_sigs)
```

Description:

Read `in_qw_count` quadwords from DRAM.



Note

Limitations: Granularity of transfer count is quadwords.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 1 to 7 (indirect read (count in GPR) and $8 \leq \text{count} \leq 15$)

Table 2.152. dram_read parameters

Name	Description
<code>out_data</code>	Transfer register that will contain read data
<code>in_dram_addr</code>	DRAM address. Register or constant. Added to <code>in_addr_offset</code> to form DRAM address used in transfer.
<code>in_addr_offset</code>	DRAM address. Register or constant. Added to <code>in_dram_addr</code> to form DRAM address used in transfer.
<code>in_qw_count</code>	Register or constant. Number of quadwords to read. The maximum quadword count is 16.
<code>REQ_SIG</code>	Requested signal.
<code>in_wakeup_sigs</code>	List of signals causing thread to swap/wakeup.

2.10.2.6 dram_read

Prototype:

```
dram_read(out_data, in_addr_1, in_addr_2, in_addr_3, in_qw_count, REQ_SIG, in_wakeup_sigs, in_reserved)
```

Description:

Read `in_qw_count` quadwords from DRAM - 40-bit addressing version.



Note

Limitations: Granularity of transfer count is quadwords. **Address specification** takes the form "`reg_or_const1, <<8, reg_or_const2`" or "`reg_or_const1, reg_or_const2, <<8`".

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 1 to 7 (indirect read (count in GPR) and $8 \leq \text{count} \leq 15$)

Table 2.153. dram_read parameters

Name	Description
<i>out_data</i>	Transfer register that will contain read data
<i>in_addr_1</i>	Address specification - see note.
<i>in_addr_2</i>	Address specification - see note.
<i>in_addr_3</i>	Address specification - see note.
<i>in_qw_count</i>	Register or constant. Number of quadwords to read. The maximum quadword count is 16.
<i>REQ_SIG</i>	Requested signal.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup.
<i>in_reserved</i>	Reserved for future use. Pass: as the value of this parameter.

2.10.2.7 dram_tbuf_write

Prototype:

```
dram_tbuf_write(in_dram_addr, in_dram_addr_offset, in_tbuf_addr, in_tbuf_addr_offset,
in_qw_count, REQ_SIG, in_wakeup_sigs, in_reserved)
```

Description:

Copy *in_qw_count* quadwords from DRAM address to TBUF.

(*in_dram_addr* + *in_dram_addr_offset*) to the TBUF address. (*in_tbuf_addr* + *in_tbuf_addr_offset*).



Note

Limitations: Granularity of transfer count is quadwords.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 2 to 10

Table 2.154. dram_tbuf_write parameters

Name	Description
<i>in_dram_addr</i>	DRAM address. Register or constant. Added to <i>in_dram_addr_offset</i> to form DRAM address used in transfer.
<i>in_dram_addr_offset</i>	DRAM address. Register or constant. Added to <i>in_dram_addr</i> to form DRAM address used in transfer.

Name	Description
<i>in_tbuf_addr</i>	TBUF address. Register or constant. Added to <i>in_tbuf_addr_offset</i> to form TBUF address used in transfer.
<i>in_tbuf_addr_offset</i>	TBUF address. Register or constant. Added to <i>in_tbuf_addr</i> to form TBUF address used in transfer.
<i>in_qw_count</i>	Register or constant. Number of quadwords to write.
<i>REQ_SIG</i>	Requested signal.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup.
<i>in_reserved</i>	Reserved for future use. Pass: as the value of this parameter.

Availability:

IXP Indirect Reference Mode

2.10.2.8 dram_write

Prototype:

```
dram_write(in_data, in_addr_1, in_addr_2, in_addr_3, in_qw_count, REQ_SIG, in_wakeup_sigs)
```

Description:

Write *in_qw_count* quadwords to DRAM memory - 40-bit addressing version.



Note

Limitations: Granularity of transfer count is quadwords. **Address specification** takes the form "reg_or_const1, <<8, reg_or_const2" or "reg_or_const1, reg_or_const2, <<8".

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 1 to 7 (indirect write (count in GPR) and $8 \leq \text{count} \leq 15$)

Table 2.155. dram_write parameters

Name	Description
<i>in_data</i>	Transfer register containing data to write
<i>in_addr_1</i>	Address specification - see note.
<i>in_addr_2</i>	Address specification - see note.
<i>in_addr_3</i>	Address specification - see note.
<i>in_qw_count</i>	Register or constant. Number of quadwords to write. The maximum quadword count is 16.
<i>REQ_SIG</i>	Requested signal.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup.

2.10.2.9 dram_write

Prototype:

```
dram_write(in_data, in_dram_addr, in_addr_offset, in_qw_count, REQ_SIG, in_wakeup_sigs)
```

Description:

Write `in_qw_count` quadwords to DRAM memory.



Note

Limitations: Granularity of transfer count is quadwords.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 1 to 7 (indirect write (count in GPR) and $8 \leq \text{count} \leq 15$)

Table 2.156. dram_write parameters

Name	Description
<i>in_data</i>	Transfer register containing data to write
<i>in_dram_addr</i>	DRAM address. Register or constant. Added to <i>in_addr_offset</i> to form DRAM address used in transfer.
<i>in_addr_offset</i>	DRAM address. Register or constant. Added to <i>in_dram_addr</i> to form DRAM address used in transfer.
<i>in_qw_count</i>	Register or constant. Number of quadwords to write. The maximum quadword count is 16.
<i>REQ_SIG</i>	Requested signal.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup.

2.10.2.10 ddr_add64_immed_init

Prototype:

```
ddr_add64_immed_init(indirect_ref_reg, en_64_bit, ref_cnt, byte_mask_dm_dr)
```

Description:

Alias to `ddr_add64_immed_init`.



Warning

This function is deprecated and may be removed in the future.

Table 2.157. ddr_add64_immed_init parameters

Name	Description
<i>indirect_ref_reg</i>	GPR to be initialized
<i>en_64_bit</i>	Constant Boolean value. <ul style="list-style-type: none"> • 0: Perform 32-bit Add operations • 1: Perform 64-bit Add operations
<i>ref_cnt</i>	Reference count. Constant. Valid values: 0, 1, 2, 3.
<i>byte_mask_dm_dr</i>	Not used, must be 1.

2.10.2.11 ddr_add64_immed_init

Prototype:

```
ddr_add64_immed_init(indirect_ref_reg, en_64_bit, ref_cnt)
```

Description:

Initialize the indirect reference register for ddr_add64_immed.

Using a static register for indirect reference, saves few instructions for every immed add operation.

Table 2.158. ddr_add64_immed_init parameters

Name	Description
<i>indirect_ref_reg</i>	GPR to be initialized
<i>en_64_bit</i>	Constant Boolean value. <ul style="list-style-type: none"> • 0: Perform 32-bit Add operations • 1: Perform 64-bit Add operations
<i>ref_cnt</i>	Reference count. Constant. Valid values: 0, 1, 2, 3.

2.10.2.12 ddr_add64_immed

Prototype:

```
ddr_add64_immed(indirect_ref_reg, val, addr, offset)
```

Description:

Given indirect reference register, value, address and offset, do immed add.

In NFP indirect reference mode, 14-bit values are supported. In IXP indirect reference mode, only 7-bit values are supported.

Table 2.159. ddr_add64_immed parameters

Name	Description
<i>indirect_ref_reg</i>	Indirect reference register initialized using ddr_add64_immed_init macro
<i>val</i>	Value to be added - constant or GPR
<i>addr</i>	DRAM address where add to be performed
<i>offset</i>	Offset from 'addr', where add is to be performed. must by 8-byte aligned.

2.10.2.13 ddr_add64_immed_sat

Prototype:

```
ddr_add64_immed_sat(indirect_ref_reg, val, addr, offset)
```

Description:

Given indirect reference register, value, address and offset, do immed add (saturates at max value).

In NFP indirect reference mode, 14-bit values are supported. In IXP indirect reference mode, only 7-bit values are supported.

Table 2.160. ddr_add64_immed_sat parameters

Name	Description
<i>indirect_ref_reg</i>	Indirect reference register initialized using ddr_add64_immed_init macro
<i>val</i>	Value to be added - constant or GPR
<i>addr</i>	DRAM address where add to be performed
<i>offset</i>	Offset from 'addr', where add to be performed. must by 8-byte aligned.

2.10.2.14 dram_memcmp

Prototype:

```
dram_memcmp(in_cur_addr_1, in_cur_addr_2, in_cur_addr_3, in_src_addr_1, in_src_addr_2,  
in_src_addr_3, in_cur_len, ret_val)
```

Description:

Compare a region of DRAM memory with a specified pattern.

Comparison is done on data from in_cur_dram_addr (labelled A) to data at in_src_dram_addr (labelled B).



Note

Address specification takes the form "reg_or_const1, <<8, reg_or_const2" or "reg_or_const1, reg_or_const2, <<8".

Table 2.161. dram_memcmp parameters

Name	Description
<i>in_cur_addr_1</i>	Address specification - see note.
<i>in_cur_addr_2</i>	Address specification - see note.
<i>in_cur_addr_3</i>	Address specification - see note.
<i>in_src_addr_1</i>	Address specification - see note.
<i>in_src_addr_2</i>	Address specification - see note.
<i>in_src_addr_3</i>	Address specification - see note.
<i>in_cur_len</i>	Number of bytes to compare
<i>ret_val</i>	Set to the first address in A where a mismatch was found otherwise unmodified

2.10.2.15 dram_memcmp

Prototype:

```
dram_memcmp(in_cur_dram_addr, in_cur_dram_offset, in_src_dram_addr, in_src_dram_offset,
in_cur_len, ret_val)
```

Description:

Compare a region of DRAM memory with a specified pattern.

Comparison is done on data from *in_cur_dram_addr* (labelled A) to data at *in_src_dram_addr* (labelled B).

Table 2.162. dram_memcmp parameters

Name	Description
<i>in_cur_dram_addr</i>	Byte address to start comparing from (A)
<i>in_cur_dram_offset</i>	Offset added to <i>in_cur_dram_addr</i> to determine start address
<i>in_src_dram_addr</i>	Byte address to start comparing against (B)
<i>in_src_dram_offset</i>	Offset added to <i>in_src_dram_addr</i> to determine start address
<i>in_cur_len</i>	Number of bytes to compare
<i>ret_val</i>	Set to the first address in A where a mismatch was found or unmodified

2.10.2.16 dram_memset

Prototype:

```
dram_memset(in_dram_addr, in_len, lw_pattern, CHUNK_SIZE)
```

Description:

Fill a region of DRAM memory with a specified pattern.

Compatibility version, which does not use an offset parameter

Table 2.163. dram_memset parameters

Name	Description
<i>in_dram_addr</i>	Address to start memory fill from
<i>in_len</i>	Number of bytes to set. Must be a multiple of <code>CHUNK_SIZE</code> .
<i>lw_pattern</i>	32-bit pattern to fill memory region with
<i>CHUNK_SIZE</i>	Chunk size, a multiple of 8 bytes from 8 to 64. Must be a constant.

2.10.2.17 dram_memset

Prototype:

```
dram_memset(in_dram_addr, in_addr_offset, in_len, lw_pattern, CHUNK_SIZE)
```

Description:

Fill a region of DRAM memory with a specified pattern.

Compatibility version, which does not use an offset parameter

Table 2.164. dram_memset parameters

Name	Description
<i>in_dram_addr</i>	Address to start memory fill from
<i>in_addr_offset</i>	Added to <i>in_dram_addr</i> to form the DRAM address used in transfer
<i>in_len</i>	Number of bytes to set. Must be a multiple of <code>CHUNK_SIZE</code> .
<i>lw_pattern</i>	32-bit pattern to fill memory region with
<i>CHUNK_SIZE</i>	Chunk size, a multiple of 8 bytes from 8 to 64. Must be a constant.

2.10.2.18 dram_memset

Prototype:

```
dram_memset(in_addr_1, in_addr_2, in_addr_3, in_len, lw_pattern, CHUNK_SIZE)
```

Description:

Fill a region of DRAM memory with a specified pattern.



Note

Address specification takes the form "reg_or_const1, <<8, reg_or_const2" or "reg_or_const1, reg_or_const2, <<8".

Table 2.165. dram_memset parameters

Name	Description
<i>in_addr_1</i>	Address specification - see note.
<i>in_addr_2</i>	Address specification - see note.
<i>in_addr_3</i>	Address specification - see note.
<i>in_len</i>	Number of bytes to set. Must be a multiple of <code>CHUNK_SIZE</code> .
<i>lw_pattern</i>	32-bit pattern to fill memory region with
<i>CHUNK_SIZE</i>	Chunk size, a multiple of 8 bytes from 8 to 64. Must be a constant.

2.11 Event filters and autopush API

2.11.1 CLS Filters and Autopush Macros

Cluster Scratch Event Filters & Autopush config Macros

2.11.2.1 evntm_cls_event_filter_config

Prototype:

```
evntm_cls_event_filter_config(filter, mask, match, filter_type)
```

Description:

This macro configures an event filter to a provided mask/match and filter type.

Table 2.166. evntm_cls_event_filter_config parameters

Name	Description
<i>filter</i>	
<i>mask</i>	
<i>match</i>	
<i>filter_type</i>	

2.11.2.2 evntm_cls_autopush_monitor_config

Prototype:

```
evntm_cls_autopush_monitor_config(filter, me_num, ctxt_num, auto_push_xfer_reg, signal)
```

Description:

This macro configures an autopush monitor for a filter with ME number, transfer register and signal.

Table 2.167. evntm_cls_autopush_monitor_config parameters

Name	Description
<i>filter</i>	Filter number to be monitored
<i>me_num</i>	Microengine number to signal and push data to when a FilterStatusMonitor fires. Range[4-15]
<i>ctxt_num</i>	Context to be signaled on event. Range[0-7] ME Transfer register to push event status
<i>auto_push_xfer_reg</i>	
<i>signal</i>	Signal reference to use on auto push.

2.11.2.3 evntm_cls_autopush_monitor_config

Prototype:

```
evntm_cls_autopush_monitor_config(filter, auto_push_xfer_reg, signal)
```

Description:

This macro configures an autopush monitor for a filter.transfer register and signal ME number and Context number are taken from ACTIVE_CTX_STS i.e.

which ever ME & Context is calling this macro.

Table 2.168. evntm_cls_autopush_monitor_config parameters

Name	Description
<i>filter</i>	Filter number to be monitored ME Transfer register to push event status
<i>auto_push_xfer_reg</i>	
<i>signal</i>	Signal reference to use on auto push

2.11.2.4 evntm_cls_autopush_monitor_engage

Prototype:

```
evntm_cls_autopush_monitor_engage(filter, in_xfer, signal, sig_action)
```

Description:

This macro should be called to start monitoring an event filter, after evntm_cls_autopush_monitor_config has been called once.

This macro uses 'one shot acknowledge'

Table 2.169. evntm_cls_autopush_monitor_engage parameters

Name	Description
<i>filter</i>	Filter number to be monitored
<i>in_xfer</i>	Xfer register to be used for I/O
<i>signal</i>	I/O Signal to use
<i>sig_action</i>	THD_SWAP/THD_SPIN/NONE

2.11.2.5 evntm_cls_autopush_user_event

Prototype:

```
evntm_cls_autopush_user_event(event, in_xfer, signal, sig_action)
```

Description:

This macro pushes an event into UserEvent in the CLS event manager.

Table 2.170. evntm_cls_autopush_user_event parameters

Name	Description
<i>event</i>	Event to push
<i>in_xfer</i>	
<i>signal</i>	I/O Signal to use
<i>sig_action</i>	THD_SWAP/THD_SPIN/NONE

2.12 Fletcher Hash Operations

2.12.1 Fletcher Hash Operation Macros

These macros calculate Fletcher, Jenkins, and Hardware based Hash of data in local memory

2.12.2.1 fletcher_hash

Prototype:

```
fletcher_hash(out_f_hash, in_lm_base_addr, in_data_size_lw, in_lm_handle, in_calc_upper)
```

Description:

Calculate 32 bit Fletcher Hash of data in local memory.

Example:

```
localmem_set_address(0, 0, LM_HANDLE_0)
localmem_write8( 0x12345678, \
    0x11111111, \
    0x22222222, \
    0x33333333, \
    0x44444444, \
    0x55555555, \
    0x66666666, \
    0x77777777, \
    0,0)
#define_eval LM_BASE_ADDR 0
#define_eval LM_SIZE_LWORDS 8
.reg r_hash_val
fletcher_hash(r_hash_val, LM_BASE_ADDR, LM_SIZE_LWORDS, 0, 1)
```

refer http://en.wikipedia.org/wiki/Fletcher's_checksum for more information

Table 2.171. fletcher_hash parameters

Name	Description
<i>out_f_hash</i>	Register receives hash calculation result
<i>in_lm_base_addr</i>	Register or Constant, start address of data in local memory
<i>in_data_size_lw</i>	Register or Constant, size of data in lwords
<i>in_lm_handle</i>	Constant, lm handle, either 0 or 1
<i>in_calc_upper</i>	Constant, enable calculation of upper 16 bits if 1 can be set to 0 to save instructions if upper bits not needed

2.12.2.2 jenkins_hash

Prototype:

```
jenkins_hash(out_j_hash, in_lm_base_addr, in_data_size_lw, in_lm_handle)
```

Description:

Calculate Jenkins Hash of data in local memory.

Example:

```
localmem_set_address(0, 0, LM_HANDLE_0)
localmem_write8( 0x12345678, \
    0x11111111, \
    0x22222222, \
    0x33333333, \
    0x44444444, \
    0x55555555, \
    0x66666666, \
    0x77777777, \
    0,0)
#define_eval LM_BASE_ADDR 0
#define_eval LM_SIZE_LWORDS 8
```

```
.reg r_hash_val
jenkins_hash(r_hash_val, LM_BASE_ADDR, LM_SIZE_LWORDS, 0)
```

refer to http://en.wikipedia.org/wiki/Jenkins_hash_function for more information

Table 2.172. jenkins_hash parameters

Name	Description
<i>out_j_hash</i>	Register receives hash calculation result
<i>in_lm_base_addr</i>	Register or Constant, start address of data in local memory
<i>in_data_size_lw</i>	Register or Constant, size of data in lwords
<i>in_lm_handle</i>	Constant, lm handle, either 0 or 1

2.12.2.3 jenkins_byte_hash

Prototype:

```
jenkins_byte_hash(out_j_hash, in_lm_base_addr, in_data_size_lw, in_lm_handle)
```

Description:

Calculate Jenkins Hash of data in local memory, byte wise.

Example:

```
localmem_set_address(0, 0, LM_HANDLE_0)
localmem_write8( 0x12345678, \
                 0x11111111, \
                 0x22222222, \
                 0x33333333, \
                 0x44444444, \
                 0x55555555, \
                 0x66666666, \
                 0x77777777, \
                 0,0)
#define_eval LM_BASE_ADDR 0
#define_eval LM_SIZE_LWORDS 8
.reg r_hash_val
jenkins_byte_hash(r_hash_val, LM_BASE_ADDR, LM_SIZE_LWORDS, 0)
```

refer to http://en.wikipedia.org/wiki/Jenkins_hash_function for more information

Table 2.173. jenkins_byte_hash parameters

Name	Description
<i>out_j_hash</i>	Register receives hash calculation result
<i>in_lm_base_addr</i>	Register or Constant, start address of data in local memory
<i>in_data_size_lw</i>	Register or Constant, size of data in lwords
<i>in_lm_handle</i>	Constant, lm handle, either 0 or 1

2.12.2.4 hardware_hash

Prototype:

```
hardware_hash(out_hw_hash, in_lm_base_addr, in_data_size_lw, in_lm_handle)
```

Description:

Calculate Hash using IXP/NFP32XX hash hardware.

Returns result LSW

Example:

```
localmem_set_address(0, 0, LM_HANDLE_0)
localmem_write4( 0x12345678, \
                 0x11111111, \
                 0x22222222, \
                 0x33333333, \
                 0,0)
#define_eval LM_BASE_ADDR 0
#define_eval LM_SIZE_LWORDS 8
.reg r_hash_val
hardware_hash(r_hash_val, LM_BASE_ADDR, LM_SIZE_LWORDS, 0)
```



Note

Not available for NFP6000. Use hash_init_cls() and hash_translate() instead.

Table 2.174. hardware_hash parameters

Name	Description
<i>out_hw_hash</i>	Register receives hash calculation result
<i>in_lm_base_addr</i>	Register or Constant, start address of data in local memory
<i>in_data_size_lw</i>	Register or Constant, size of data in lwords (only 4 is supported)
<i>in_lm_handle</i>	Constant, lm handle, either 0 or 1

2.13 HASH operation

2.13.1 HASH operation macros

Trie type (TRIE_TYPE) used by hash lookup macros is encoded as:31(8-bits)24(8-bits)16(8-bits)8(4-bits)4(4-bits)0

[PRIM_INDEX_START_BIT|PRIM_INDEX_BIT_LEN|COLL_INDEX_START_BIT|COLL_SHIFT_BITS|LOOKUP_TYPE|

----- For use with CLS, this section
creates a global absolute register: @cls_hash_mask_idx_addr

Table 2.175. HASH operation and Defines

Defined	Definition
HW_HASH_48	1 API identifier for 48 bit hash keys.
HW_HASH_64	2 API identifier for 64 bit hash keys.
HW_HASH_128	3 API identifier for 128 bit hash keys.
CLS_HASH_M4	(1<<0)
CLS_HASH_M36	(1<<1)
CLS_HASH_M53	(1<<2)
CLS_HASH_M63	(1<<3)
CLS_HASH_BASE_ADDR	0x40000
CLS_HASH_MUL_REG_OFFSET	0x000
CLS_HASH_IDX0_64_REG_OFFSET	0x800

2.13.3.1 hash_init

Prototype:

```
hash_init(in_multiplier, MULTIPLIER_SIZE)
```

Description:

Initialize the hash multiplier of the hardware hash translation unit.

Example:

```
xbuf_alloc($multiplier, 2, read_write)
move($multiplier[0], 0)
move($multiplier[1], 1)
hash_init($multiplier, HW_HASH_64)
xbuf_free($multiplier)
```



Note

Not available for NFP6000. Use hash_init_cls instead.

Instruction Count: 2 to 4

Table 2.176. hash_init parameters

Name	Description
<i>in_multiplier</i>	Buffer of write transfer registers with the multiplier. For more description of the multiplier, please refer to the Hash Unit section of the <i>Netronome Network Flow Processor 6000 Databook</i> . Note that this parameter must be supplied using the xbuf_alloc macro. (See Example Usage)
<i>MULTIPLIER_SIZE</i>	Size of multiplier: HW_HASH_48, HW_HASH_64 or HW_HASH_128

2.13.3.2 hash_init_cls

Prototype:

```
hash_init_cls(INDEX, in_mask, MUL_SEL, SB_ENA, NUM_SB)
```

Description:

Initialize the cls hash logic.

Example:

```
#define CLS_HASH_IDX 1
#define CLS_MUL_SEL (CLS_HASH_M4 | CLS_HASH_M36 | CLS_HASH_M53 | CLS_HASH_M63 )
#define CLS_SB_ENA 0
#define CLS_NUM_SB 0
.global_mem hashmask cls 16 8
.init hashmask 0xffffffff, 0x0000ffff, 0x00000000, 0x00000000
hash_init_cls(CLS_HASH_IDX, hashmask, CLS_MUL_SEL, CLS_SB_ENA, CLS_NUM_SB)
```



Note

Must be invoked on NFP6000 prior to using other hash macros.

Instruction Count: 2 to 3

Table 2.177. hash_init_cls parameters

Name	Description
<i>INDEX</i>	Cls Hash Index to use, valid values 0-7
<i>in_mask</i>	Cls global mem with initialized 128 bit mask value. For more description of the mask, please refer to the CLS Hash Unit section of the <i>Netronome Network Flow Processor 6000 Databook</i> .
<i>MUL_SEL</i>	Multiplier selects, logical or of CLS_HASH_M4, CLS_HASH_M36, CLS_HASH_M53, CLS_HASH_M63 as required by user. Refer to <i>Netronome Network Flow Processor 6000 Databook</i> for description
<i>SB_ENA</i>	SBOX Enable, valied values 1 or 0, refer to <i>Netronome Network Flow Processor 6000 Databook</i> for description
<i>NUM_SB</i>	Number of SBOX's to use, value values 0 through 15, refer to <i>Netronome Network Flow Processor 6000 Databook</i> for description.

2.13.3.3 hash_translate

Prototype:

```
hash_translate(io_index, INDEX_SIZE, REQ_SIG, in_wakeup_sigs)
```

Description:

Translate a big index, up to 128 bits, using the hardware hash translation unit.

Example:

```
.sig sig1
hash_translate($big_index, HW_HASH_128, sig1, sig1)
```

Instruction Count: 1 to 2

Table 2.178. hash_translate parameters

Name	Description
<i>io_index</i>	Buffer of read/write transfer registers: <ul style="list-style-type: none"> • Output: buffer of read transfer registers with the translated index • Input : buffer of write transfer registers with the index to be translated
<i>INDEX_SIZE</i>	Size of index: HW_HASH_48, HW_HASH_64 or HW_HASH_128
<i>REQ_SIG</i>	Requested signal. See common section Signal Arguments.
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup. See common section Signal Arguments.

2.13.3.4 hash_lookup

Prototype:

```
hash_lookup(out_index, in_key, KEY_SIZE, trie_base_addr, TRIE_TYPE, KEY_DATA_SD_BASE)
```

Description:

Lookup a table entry using up to 128 bit index.

Uses tables written by core Hash Table Database Manager.

Example:

```
hash_lookup(table_entry_index, $wide_index[0], 102, trie_addr, HASH16_4, 0x100)
```



Note

If HASH_DONT_TRANSLATE_KEYS is defined, the macro will skip hash translation.

Instruction Count: 14 to 16 for `TRIE_TYPE = HASH_16_4` ((1 dram access, 1 dram access) + 11*No of iterations (1 sram access/iteration)), 12 to 19 for `TRIE_TYPE = HASH_FOLD_16` ((1 sram access) + 14*No of iteration (1 dram access/Iteration))

Table 2.179. hash_lookup parameters

Name	Description
<code>out_index</code>	Index of hash table entry if success. 0 if fail (no entry)
<code>in_key</code>	Buffer of write transfer registers with the starting index, up to 128 bits
<code>KEY_SIZE</code>	Size of <code>in_key</code> in bits
<code>trie_base_addr</code>	Address of trie table, which is the sram primary base table
<code>TRIE_TYPE</code>	Index bits used to address the hash trie: <code>HASH_16_4</code> , <code>HASH_FOLD_16</code> <ul style="list-style-type: none"> <code>HASH_16_4</code>: First lookup uses 16 bits of index, subsequent lookups use 4 bits of index <code>HASH_FOLD_16</code>: First lookup XORs the initial index to reduce it by half, then performs a table lookup using 16 bits of half-index, with subsequent chain search until no collision
<code>KEY_DATA_SD_BASE</code>	GPR or CONST, list of available SDRAM space to be used for key/data storage

2.13.3.5 hash_dual_lookup

Prototype:

```
hash_dual_lookup(out_index1, out_index2, in_key1, in_key2, KEY_SIZE, trie_base_addr,
TRIE_TYPE, KEY_DATA_SD_BASE)
```

Description:

Lookup two table entries in parallel using up to 128 bit index each.

The reads of the trie structure are done in parallel. This utilizes trie structure and tables written by core Hash Table Database Manager.

Example:

```
hash_dual_lookup(table_entry_index1, table_entry_index2, $wide_index1[0], wide_index2[0], \
102, trie_addr, HASH16_4, 0)
```



Note

If `HASH_DONT_TRANSLATE_KEYS` is defined the macro will skip hash translation.

Instruction Count: 32 to 37 for `TRIE_TYPE = HASH_16_4` ((2 sram access, 2 dram access) + 22*No of iterations (2 sram access/iteration)), 32 to 40 for `TRIE_TYPE = HASH_FOLD_16` ((2 sram access) + 28*No of iteration (1 dram access/Iteration))

Table 2.180. hash_dual_lookup parameters

Name	Description
<i>out_index1</i>	Corresponding index of hash table entry if success. 0 if fail (no entry).
<i>out_index2</i>	Corresponding index of hash table entry if success. 0 if fail (no entry).
<i>in_key1</i>	Corresponding buffer of write transfer registers with the starting index, up to 128 bits
<i>in_key2</i>	Corresponding buffer of write transfer registers with the starting index, up to 128 bits
<i>KEY_SIZE</i>	Size of <i>in_key1</i> in bits, <i>in_key2</i> must be the same size
<i>trie_base_addr</i>	Address of trie table, which is the sram primary base table
<i>TRIE_TYPE</i>	Index bits used to address the hash trie: HASH_16_4, HASH_FOLD_16 <ul style="list-style-type: none"> HASH_16_4: First lookup uses 16 bits of index, subsequent lookups use 4 bits of index HASH_FOLD_16: First lookup XORs the initial index to reduce it by half, then performs a table lookup using 16 bits of half-index, with subsequent chain search until no collision
<i>KEY_DATA_SD_BASE</i>	GPR or CONST, list of available SDRAM space to be used for key/data storage

2.14 Limit Operations

2.14.1 Limit Operation Macros

These macros perform limit operations

2.14.2.1 limit_min

Prototype:

```
limit_min(out_c, in_a, in_b)
```

Description:

Calculate minimum of two values using signed arithmetic.

Example:

```
.reg minimum, ina, inb
immed[ina,5]
immed[inb,6]
limit_min(minimum, ina, inb)
beq[result_is_zero#]
```

Use *in_a* for the parameter most likely to be the smallest.

Condition Codes: N,Z set based on result in *out_c*

Instruction Count: 4

Cycles: For 3 different parameters: 4 if in_a <= in_b, else 5 Where in_a or in_b is the same as out_c: 5 if out_c already lowest, else 4

Table 2.181. limit_min parameters

Name	Description
<i>out_c</i>	Register written with lower of ina or inb
<i>in_a</i>	Register, value to compare to in_b
<i>in_b</i>	Register, value to compare to in_a

2.14.2.2 limit_min

Prototype:

```
limit_min(io_a, in_b)
```

Description:

Calculate minimum of two values using signed arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,5]
immed[inb,6]
limit_min(ioa, inb)
```

Condition Codes: Data dependent: assume they are clobbered

Instruction Count: 3

Table 2.182. limit_min parameters

Name	Description
<i>io_a</i>	Register, value to compare to in_b, and resultant min value
<i>in_b</i>	Register, value to compare to io_a

2.14.2.3 limit_min_cc

Prototype:

```
limit_min_cc(io_a, in_b)
```

Description:

Calculate minimum of two values using signed arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,0]
immed[inb,6]
limit_min_cc(ioa, inb)
beq[result_is_zero#]
```

Condition Codes: N,Z set based on result in io_a

Instruction Count: 4

Table 2.183. limit_min_cc parameters

Name	Description
<i>io_a</i>	Register, value to compare to in_b, and resultant min value
<i>in_b</i>	Register, value to compare to io_a

2.14.2.4 limit_max

Prototype:

```
limit_max(out_c, in_a, in_b)
```

Description:

Calculate maximum of two values using signed arithmetic.

Example:

```
.reg maximum, ina, inb
immed[ina,5]
immed[inb,6]
limit_max(maximum, ina, inb)
beq[result_is_zero#]
```

Use in_b for the parameter most likely to be the highest.

Condition Codes: N,Z set based on result in out_c

Instruction Count: 4

Cycles: For 3 different parameters: 4 if in_a <= in_b, else 5 Where in_a or in_b is the same as out_c: 5 if out_c already lowest, else 4

Table 2.184. limit_max parameters

Name	Description
<i>out_c</i>	Register written with higher of ina or inb
<i>in_a</i>	Register, value to compare to in_b
<i>in_b</i>	Register, value to compare to in_a

2.14.2.5 limit_max

Prototype:

```
limit_max(io_a, in_b)
```

Description:

Calculate maximum of two values using signed arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,5]
immed[inb,6]
limit_max(ioa, inb)
```

Condition Codes: Data dependent: assume they are clobbered

Instruction Count: 3

Table 2.185. limit_max parameters

Name	Description
<i>io_a</i>	Register, value to compare to <i>in_b</i> , and resultant max value
<i>in_b</i>	Register, value to compare to <i>io_a</i>

2.14.2.6 limit_max_cc

Prototype:

```
limit_max_cc(io_a, in_b)
```

Description:

Calculate maximum of two values using signed arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,5]
immed[inb,6]
limit_max(ioa, inb)
beq[result_is_zero#]
```

Condition Codes: N,Z set based on result in *io_a*

Instruction Count: 4

Table 2.186. limit_max_cc parameters

Name	Description
<i>io_a</i>	Register, value to compare to in_b, and resultant max value
<i>in_b</i>	Register, value to compare to io_a

2.14.2.7 limit_min_unsigned

Prototype:

```
limit_min_unsigned(out_c, in_a, in_b)
```

Description:

Calculate minimum of two values using unsigned arithmetic.

Example:

```
.reg minimum, ina, inb
immed[ina,5]
immed[inb,6]
limit_min_unsigned(minimum, ina, inb)
beq[result_is_zero#]
```

Use in_a for the parameter most likely to be the smallest.

Condition Codes: N,Z set based on result in out_c

Instruction Count: 4

Cycles: For 3 different parameters: 4 if in_a <= in_b, else 5 Where in_a or in_b is the same as out_c: 5 if out_c already lowest, else 4

Table 2.187. limit_min_unsigned parameters

Name	Description
<i>out_c</i>	Register written with lower of ina or inb
<i>in_a</i>	Register, value to compare to in_b
<i>in_b</i>	Register, value to compare to in_a

2.14.2.8 limit_min_unsigned

Prototype:

```
limit_min_unsigned(io_a, in_b)
```

Description:

Calculate minimum of two values using unsigned arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,5]
immed[inb,6]
limit_min_unsigned(ioa, inb)
```

Condition Codes: Data dependent: assume they are clobbered

Instruction Count: 3

Table 2.188. limit_min_unsigned parameters

Name	Description
<i>io_a</i>	Register, value to compare to <i>in_b</i> , and resultant min value
<i>in_b</i>	Register, value to compare to <i>io_a</i>

2.14.2.9 limit_min_unsigned_cc

Prototype:

```
limit_min_unsigned_cc(io_a, in_b)
```

Description:

Calculate minimum of two values using unsigned arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,5]
immed[inb,6]
limit_min_unsigned_cc(ioa, inb)
beq[result_is_zero#]
```

Condition Codes: N,Z set based on result in *io_a*

Instruction Count: 4

Table 2.189. limit_min_unsigned_cc parameters

Name	Description
<i>io_a</i>	Register, value to compare to <i>in_b</i> , and resultant min value
<i>in_b</i>	Register, value to compare to <i>io_a</i>

2.14.2.10 limit_max_unsigned

Prototype:

```
limit_max_unsigned(out_c, in_a, in_b)
```

Description:

Calculate maximum of two values using unsigned arithmetic.

Example:

```
.reg maximum, ina, inb
immed[ina,5]
immed[inb,6]
limit_max_unsigned(maximum, ina, inb)
beq[result_is_zero#]
```

Use in_b for the parameter most likely to be the highest.

Condition Codes: N,Z set based on result in out_c

Instruction Count: 4

Cycles: For 3 different parameters: 4 if in_a <= in_b, else 5 Where in_a or in_b is the same as out_c: 5 if out_c already lowest, else 4

Table 2.190. limit_max_unsigned parameters

Name	Description
out_c	Register written with higher of ina or inb
in_a	Register, value to compare to in_b
in_b	Register, value to compare to in_a

2.14.2.11 limit_max_unsigned

Prototype:

```
limit_max_unsigned(io_a, in_b)
```

Description:

Calculate maximum of two values using unsigned arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,5]
immed[inb,6]
limit_max_unsigned(ioa, inb)
```

Condition Codes: Data dependent: assume they are clobbered

Instruction Count: 3

Table 2.191. limit_max_unsigned parameters

Name	Description
<i>io_a</i>	Register, value to compare to in_b, and resultant max value
<i>in_b</i>	Register, value to compare to io_a

2.14.2.12 limit_max_unsigned_cc

Prototype:

```
limit_max_unsigned_cc(io_a, in_b)
```

Description:

Calculate maximum of two values using unsigned arithmetic.

Example:

```
.reg ioa, inb
immed[ioa,5]
immed[inb,6]
limit_max_unsigned_cc(ioa, inb)
beq[result_is_zero#]
```

Condition Codes: N,Z set based on result in io_a

Instruction Count: 4

Table 2.192. limit_max_unsigned_cc parameters

Name	Description
<i>io_a</i>	Register, value to compare to in_b, and resultant max value
<i>in_b</i>	Register, value to compare to io_a

2.14.2.13 limit_align_first_chunk

Prototype:

```
limit_align_first_chunk(out_chunk, in_size, in_align, in_address)
```

Description:

Calculate size of first chunk of work.

Example:

```
#define_eval BOUNDARY 8
.reg chunk_size, size, address
move(size,200)
```



```
move(address,0x12340004)
limit_align_first_chunk(chunk_size, size, BOUNDARY, addr)
```

Returned size (out_chunk) will not cross alignment boundary even if total size is less than the alignment size

Works with unsigned values

Cycles: 6/7 if in_align constant, else 7/8

Table 2.193. limit_align_first_chunk parameters

Name	Description
<i>out_chunk</i>	Returned size of first chunk of work, so that it will not cross a boundary (mulitple of in_align)
<i>in_size</i>	Total work size
<i>in_align</i>	Alignment boundary, must be power of 2
<i>in_address</i>	Start address of work

2.15 Math Operations

2.15.1 Math Operation Macros General

These macros perform math functions which are not provided by microcode instructions

2.15.2.1 math_int_div

Prototype:

```
math_int_div(out_q, in_numerator, in_denominator)
```

Description:

32 bit unsigned integer divide.

Example:

```
.reg dividend, divisor, quotient
immed[dividend,20]
immed[divisor,4]
math_int_div(quotient, dividend, divisor) // quotient will equal 5
```

if in_numerator and in_denominator are constants, division is done using assembler rather than at runtime

out_q is set to -1 if in_denominator is 0

Table 2.194. math_int_div parameters

Name	Description
<i>out_q</i>	GPR written with quotient (in_numerator / in_denominator)
<i>in_numerator</i>	GPR or CONSTANT, dividend
<i>in_denominator</i>	GPR or CONSTANT, divisor

2.15.2.2 math_int_div_64

Prototype:

```
math_int_div_64(out_q, in_numerator_hi, in_numerator_lo, in_denominator)
```

Description:

64 bit unsigned integer divide (supports 64 bit dividend).

Example:

```
.reg dividend_hi, dividend_lo, divisor, quotient
immed[dividend_hi,20]
immed[dividend_lo,10]
immed[divisor,5723]
math_int_div_64(quotient, dividend_hi, dividend_lo, divisor)
```

out_q is set to -1 if in_denominator is 0

Caution: This macro uses the basic subtract method of division and execution time will be proportionate to the magnitude of the resulting quotient.

Table 2.195. math_int_div_64 parameters

Name	Description
<i>out_q</i>	GPR written with quotient (in_numerator / in_denominator)
<i>in_numerator_hi</i>	GPR upper 32 bits of dividend
<i>in_numerator_lo</i>	GPR lower 32 bits of dividend
<i>in_denominator</i>	GPR divisor

2.15.2.3 math_log2

Prototype:

```
math_log2(out_result, in_arg, IN_ROUND)
```

Description:

calculate base 2 logarithm on unsigned input value.

Example:

```
.reg log2, value
immed[value, 32]
math_log2(log2, value, 0)
```

similar to LOG2 assembler function

Table 2.196. math_log2 parameters

Name	Description
<i>out_result</i>	GPR written with log base 2 of in_arg
<i>in_arg</i>	GPR 32 bit unsigned int
<i>IN_ROUND</i>	Constant, used when result is not a power of 2: round < 0: round result down to next smaller integer round = 0: generate an error (negative output) round > 0: round result up to next larger integer

2.15.2.4 math_find_highest_set

Prototype:

```
math_find_highest_set(out_result, in_arg)
```

Description:

calculate highest set bit in a 32 bit unsigned value.

Example:

```
.reg bit, value
move(value, 0x01234567)
math_find_highest_set(bit, value) // bit should equal 24
```

if no bit is set, out_result is set to -1

Table 2.197. math_find_highest_set parameters

Name	Description
<i>out_result</i>	GPR written with highest bit set of in_arg
<i>in_arg</i>	GPR 32 bit unsigned int

2.16 Memory Allocation

2.16.1 Memory Allocation Macros general info

For some macros a virtual parameter, called the freelisthandle, is used to refer to a specific group of macro parameters. It is composed of: POOL_ID Which freelist: POOL0, POOL1, POOL2 ... D_BASE Base byte address of DRAM buffers. Must be aligned to 32 byte boundary. D_SIZE Byte size of DRAM buffers. Must be power of 2. S_BASE Base byte address of SRAM buffers. Must be aligned to 4 byte boundary. S_SIZE Byte size of SRAM buffers. Must be power of 2. For example:

```
#define BUF_FH BUF_FREE_LIST0, BUF_DRAM_BASE, BUFFER_SIZE, BUF_SRAM_BASE, META_DATA_SIZE
```

All of these addresses are byte addresses, and all of these sizes refer to the number of bytes.

2.16.2.1 buf_dram_addr_from_index

Prototype:

```
buf_dram_addr_from_index(out_address, in_index, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)
```

Description:

Calculate DRAM Buffer Address from Index.

This macro calculates:

```
out_address = (in_index * D_SIZE) + D_BASE
```

Instruction Count: 3 to 4

Table 2.198. buf_dram_addr_from_index parameters

Name	Description
<i>out_address</i>	A DRAM buffer address
<i>in_index</i>	A relative index that identifies DRAM buffer and SRAM buffer descriptor address
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.16.2.2 buf_dram_addr_from_sram_addr

Prototype:

`buf_dram_addr_from_sram_addr(out_dram_addr, in_sram_addr, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)`

Description:

Given a freelist handle and SRAM address, calculate DRAM buffer address.

This macro calculates:

```
Idx = (sram_addr - sram_base) / sram_size
dram_addr = Idx * dram_size

dram_addr =      (sram_addr - sram_base)
                ( ----- ) * dram_size
                  sram_size

dram_addr = (sram_addr - sram_base) * (dram_size/sram_size)
```

Instruction Count: 5 to 6

Table 2.199. buf_dram_addr_from_sram_addr parameters

Name	Description
<i>out_dram_addr</i>	A DRAM buffer address
<i>in_sram_addr</i>	An SRAM buffer descriptor address
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.16.2.3 buf_index_from_dram_addr

Prototype:

`buf_index_from_dram_addr(out_index, in_address, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)`

Description:

Calculate array index from DRAM buffer address.

This macro calculates:

```
out_index = (in_address - D_BASE) / D_SIZE
```

Instruction Count: 2 to 4

Table 2.200. buf_index_from_dram_addr parameters

Name	Description
<i>out_index</i>	A relative index that identifies DRAM buffer and SRAM buffer descriptor addresses
<i>in_address</i>	A DRAM buffer address
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.16.2.4 buf_index_from_sram_addr

Prototype:

```
buf_index_from_sram_addr(out_index, in_address, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)
```

Description:

Calculate Array Index from Buffer Descriptor Address.

This macro calculates:

```
out_index = (in_address - S_BASE) / S_SIZE
```

Instruction Count: 2 to 4

Table 2.201. buf_index_from_sram_addr parameters

Name	Description
<i>out_index</i>	A relative index that identifies DRAM buffer and SRAM buffer descriptor addresses
<i>in_address</i>	An SRAM buffer address
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.16.2.5 buf_sram_addr_from_index

Prototype:

```
buf_sram_addr_from_index(out_address, in_index, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)
```

Description:

Calculate SRAM Buffer Address from Index.

This macro calculates:

```
out_address = (in_index * S_SIZE) + S_BASE
```

Instruction Count: 3 to 4

Table 2.202. buf_sram_addr_from_index parameters

Name	Description
<i>out_address</i>	An SRAM buffer address
<i>in_index</i>	A relative index that identifies DRAM buffer and SRAM buffer descriptor address
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.16.2.6 buf_sram_addr_from_dram_addr

Prototype:

```
buf_sram_addr_from_dram_addr(out_sram_addr, in_dram_addr, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)
```

Description:

Given a freelist handle and DRAM address, calculate SRAM buffer address.

This macro calculates:

```
Idx = (dram_addr - dram_base) / dram_size
sram_addr = Idx * sram_size

sram_addr =      (dram_addr - dram_base)
                ( ----- ) * sram_size
                  dram_size

sram_addr = (dram_addr - dram_base) * (sram_size/dram_size)
```

Instruction Count: 5 to 6

Table 2.203. buf_sram_addr_from_dram_addr parameters

Name	Description
<i>out_sram_addr</i>	An SRAM buffer address
<i>in_dram_addr</i>	A DRAM buffer descriptor address
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...

Name	Description
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.16.2.7 buf_freelist_create

Prototype:

```
buf_freelist_create(in_num_buffers, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)
```

Description:

Create SRAM buffer (descriptor) freelist using sram queues.

The freelist termination is indicated by a return value of 0 from `buf_alloc()`. The freelist is permanent. This library does not make provision for destroying the freelist. Freelist creation should be performed at initialization time only, so as not to interfere with runtime performance.

Instruction Count: (11 to 14) + 7*in_num_buffers

Table 2.204. buf_freelist_create parameters

Name	Description
<i>in_num_buffers</i>	Number of buffers put on the freelist. The minimum value is 1. The maximum value is arbitrary, chosen based on the amount of memory available for buffer space (in_num_buffers x D_SIZE) and (in_num_buffers x S_SIZE).
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.16.2.8 buf_alloc

Prototype:

```
buf_alloc(out_sram_addr, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE, REQ_SIG, in_wakeup_sigs,  
Q_OPTIONS)
```

Description:

Allocate SRAM/DRAM buffers from a buffer freelist identified by the freelist handle.

Table 2.205. buf_alloc parameters

Name	Description
<i>out_sram_addr</i>	Allocated SRAM address.
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.
<i>REQ_SIG</i>	Requested signal
<i>in_wakeup_sigs</i>	List of signals causing thread to swap/wakeup
<i>Q_OPTIONS</i>	Directive for memory controller queue selection. (Currently not applicable: pass the value ____.)

2.16.2.9 buf_free

Prototype:

```
buf_free(in_sram_addr, POOL_ID, D_BASE, D_SIZE, S_BASE, S_SIZE)
```

Description:

Return the specified SRAM address to the buffer pool identified by the freelist handle.

Table 2.206. buf_free parameters

Name	Description
<i>in_sram_addr</i>	An SRAM address
<i>POOL_ID</i>	Which freelist: POOL0, POOL1, POOL2 ...
<i>D_BASE</i>	Base byte address of DRAM buffers. Must be aligned to 32 byte boundary.
<i>D_SIZE</i>	Byte size of DRAM buffers. Must be power of 2.
<i>S_BASE</i>	Base byte address of SRAM buffers. Must be aligned to 4 byte boundary.
<i>S_SIZE</i>	Byte size of SRAM buffers. Must be power of 2.

2.17 Memory Queue Operations

2.17.1 Local Memory Queue Operation Macros

These macros implement queues using local memory Three different mechanisms are used to implement the LM queues. The most appropriate approach is automatically selected depending on the sizes. The most efficient is if the

queue size (number of items * size of item [bytes]) is 256. The worst is if the queue size is not a power-of-2, as conditional branches have to be used. No assumption is made about the alignment of the buffer in local memory. The programmer must ensure that the queue do not overflow. The *_enqueue macros enqueues the new item without checking for overflow. General use: The enqueue/dequeue macros do not directly put/get data. They adjust the head/tail/count variables and initialize the specified LM index to point to the relevant slot. The caller can then read/write the data using the specified LM index. Example use: //Parameters: _NAME_PREFIX, FLAGS, NUM_ITEMS, ITEM_SIZE define SOME_QUEUE _SOME_QUEUE, 0, 16, 4
lmqueue_init(SOME_QUEUE)lmqueue_enqueue(SOME_QUEUE, ACTIVE_LM_ADDR_0) move(*l\$index0[0], 0) move(*l\$index0[1], 100) ... lmqueue_dequeue(SOME_QUEUE, ACTIVE_LM_ADDR_0) move(tmp1, *l\$index0[0]) move(tmp2, *l\$index0[1])

Table 2.207. Memory Queue Operations and Defines

Defined	Definition
_LM_HANDLES_IN_USE	0

2.17.3.1 lm_handle_alloc

Prototype:

```
lm_handle_alloc(HANDLE)
```

Description:

Allocate Handle to local memory control logic.

Example:

```
lm_handle_alloc(SOME_LM_HANDLE)
#define_eval SOME_LM_HANDLE _LM_NEXT_HANDLE
#define_eval SOME_LM_INDEX _LM_NEXT_INDEX
```

Instruction Count: 0

Table 2.208. lm_handle_alloc parameters

Name	Description
<i>HANDLE</i>	Constant handle name

2.17.3.2 lm_handle_free

Prototype:

```
lm_handle_free(HANDLE)
```

Description:

Free Handle allocated with lm_handle_alloc.

Example:

```
lm_handle_free(SOME_LM_HANDLE)
#undef SOME_LM_HANDLE
#undef SOME_LM_INDEX
```

Instruction Count: 0

Table 2.209. lm_handle_free parameters

Name	Description
<i>HANDLE</i>	Constant handle name

2.17.3.3 lm_handle_verify

Prototype:

```
lm_handle_verify(NUM_IN_USE)
```

Description:

Verifies the number of LM handles in use.

Example:

```
lm_handle_verify(1)
```

Instruction Count: 0

Table 2.210. lm_handle_verify parameters

Name	Description
<i>NUM_IN_USE</i>	Constant, expected number of handles in use

2.17.3.4 incr_lm_base

Prototype:

```
incr_lm_base(NAME)
```

Description:

Increment base address of lm block used for queue data.

LM must be manually assigned as parts of the addresses may be used in preprocessor macros.

The lmqueue macros require LM_NAME_BASE and LM_NAME_SIZE.

Note that the blocks must be naturally aligned for some of the uses.

Each LM block can be specified by the triplet:

Example:

```
#define_eval LM_SOMENAME_BASE    (_LM_BASE)
#define_eval LM_SOMENAME_SIZE    256
incr_lm_base(LM_SOMENAME)
```

Instruction Count: 0

The BASE and the SIZE is specified in bytes.

Table 2.211. incr_lm_base parameters

Name	Description
<i>NAME</i>	Constant, Name of queue

2.18 Microengine CAM Operation

2.18.1 Microengine CAM Operation Macros

Microengine CAM Operation Macros

2.18.2.1 cam_clear_all

Prototype:

```
cam_clear_all()
```

Description:

Clear all the CAM entries.

2.18.2.2 cam_read_entry

Prototype:

```
cam_read_entry(out_data, out_state, in_entry_num)
```

Description:

Read the data and state for the specified CAM entry.

Table 2.212. cam_read_entry parameters

Name	Description
<i>out_data</i>	32-bit data read from entry
<i>out_state</i>	4-bit state data associated with this CAM entry
<i>in_entry_num</i>	CAM entry number

2.18.2.3 cam_write_entry

Prototype:

```
cam_write_entry(in_entry_num, in_data, in_state)
```

Description:

Write the data and state for the specified CAM entry.

Table 2.213. cam_write_entry parameters

Name	Description
<i>in_entry_num</i>	CAM entry number
<i>in_data</i>	32-bit data to be written to specified CAM entry
<i>in_state</i>	4-bit state data to be associated with this CAM entry

2.18.2.4 cam_read_data

Prototype:

```
cam_read_data(out_data, in_entry_num)
```

Description:

Read the data for the specified CAM entry.

Table 2.214. cam_read_data parameters

Name	Description
<i>out_data</i>	32-bit data read from entry
<i>in_entry_num</i>	CAM entry number

2.18.2.5 cam_match

Prototype:

```
cam_match(out_state, out_status, out_entry_num, in_data)
```

Description:

Perform content match to get a CAM entry.

Table 2.215. cam_match parameters

Name	Description
<i>out_state</i>	If <i>out_status</i> is 1 (hit), this contains the appropriate state value. If <i>out_status</i> is 0 (miss), this contains 0.
<i>out_status</i>	Match status. A value of 1 means Hit, 0 means Miss.
<i>out_entry_num</i>	If <i>out_status</i> is 1 (hit), this contains the matched entry number. If <i>out_status</i> is 0 (miss), this contains the least recently used (LRU) entry number. The LRU entry number can be used as a hint for writing new data into CAM.
<i>in_data</i>	32-bit data to match

2.19 Microengine CAM Sharing Operation

2.19.1 Microengine CAM Sharing Operation Macros

This API supports: CAM sharing across different microblocks running on same ME. The original LRU CAM implementation (without CAM sharing). To enable CAM sharing support, define `CAM_SHARED` before including this file. For most macros a virtual parameter, called the CAM handle, is used to refer to a specific group of macro parameters. It is composed of: `BID` Unique Microblock Block ID. 4 bits by default. `BID_BIT_SZ` COUNT Number of CAM entries. 4 bits. `FIRST` Number of first CAM entry. 4 bits. For example: `#define RX_CAM_HANDLE 1, 8, 0`. However, the CAM handle is not used if CAM sharing is not enabled. The relevant macro parameters must still be passed though. This section creates the following global absolute registers: `@camsharing_bit_vector`
`@camsharing_ref_cnt1` `@camsharing_ref_cnt2`

Table 2.216. Microengine CAM Sharing Operation and Defines

Defined	Definition
<code>SRC_KEY_BIT_SZ</code>	28 Bit size of the key part of the CAM tag. This plus <code>BID_BIT_SZ</code> must not exceed 32 bits.
<code>BID_BIT_SZ</code>	4 Bit size of the block ID part of the CAM tag. This plus <code>SRC_KEY_BIT_SZ</code> must not exceed 32 bits.

2.19.3.1 cam_init

Prototype:

```
cam_init()
```

Description:

Initializes CAM variables and clears all CAM entries, must be called before using any other macros of this API.

It is assumed that after calling this macro all CAM entries will be initialized to unique values prior to doing CAM lookups.

2.19.3.2 cam_entry_read_state

Prototype:

```
cam_entry_read_state(out_state_val, in_entry_num, in_bid, in_num_entries, in_first_entry)
```

Description:

Reads the state value for the CAM entry (the block specific entry if CAM is shared).

Example:

```
cam_entry_read_state(cam_state, cam_entry, RX_CAM_HANDLE)
```

Table 2.217. cam_entry_read_state parameters

Name	Description
<i>out_state_val</i>	The state value of the CAM entry is placed in bits 11 to 8, other bits are set to 0.
<i>in_entry_num</i>	CAM entry number. May not exceed the total number of CAM entries. It is not range checked.
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.19.3.3 cam_entry_write_state

Prototype:

```
cam_entry_write_state(in_entry_num, in_state_val, in_bid, in_num_entries, in_first_entry)
```

Description:

Writes the state value for the CAM entry (the block specific entry if CAM is shared).

Example:

```
cam_entry_write_state(cam_entry, CAM_STATE_VALID, RX_CAM_HANDLE)
```

Table 2.218. cam_entry_write_state parameters

Name	Description
<i>in_entry_num</i>	CAM entry number. May not exceed the total number of CAM entries. It is not range checked.

Name	Description
<i>in_state_val</i>	The state value to write to the CAM entry.
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.19.3.4 cam_entry_read_tag

Prototype:

```
cam_entry_read_tag(out_tag, in_entry_num, in_bid, in_num_entries, in_first_entry)
```

Description:

Reads the tag for the CAM entry (the block specific entry if CAM is shared).

Example:

```
cam_entry_read_tag(cam_tag, cam_entry, RX_CAM_HANDLE)
```

Table 2.219. cam_entry_read_tag parameters

Name	Description
<i>out_tag</i>	The tag of the CAM entry is returned here.
<i>in_entry_num</i>	CAM entry number. May not exceed the total number of CAM entries. It is not range checked.
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.19.3.5 cam_clearall

Prototype:

```
cam_clearall(in_bid, in_num_entries, in_first_entry)
```

Description:

In CAM sharing mode, clear CAM entries for the specified microblock, otherwise clear all CAM entries.



Note

In CAM sharing mode the CAM tags and state bits of entries are not cleared.

Table 2.220. cam_clearall parameters

Name	Description
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.19.3.6 cam_entry_write

Prototype:

```
cam_entry_write(in_entry_num, in_key, in_state_val, in_bid, in_num_entries, in_first_entry)
```

Description:

Write the tag for the specified CAM entry of the specific microblock.

Example:

```
cam_entry_write(cam_entry, src_key, CAM_STATE_VALID, RX_CAM_HANDLE)
```

Table 2.221. cam_entry_write parameters

Name	Description
<i>in_entry_num</i>	CAM entry number. May not exceed the total number of CAM entries. It is not range checked.
<i>in_key</i>	Source key with which CAM lookup was done.
<i>in_state_val</i>	State value of CAM entry.
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.19.3.7 cam_exit_using_entry

Prototype:

```
cam_exit_using_entry(in_entry_num, in_bid, in_num_entries, in_first_entry)
```

Description:

Decrement the reference count for the specified CAM entry to indicate that a thread is done with the entry.

When the reference count is zero, the entry is marked as free.

Table 2.222. cam_exit_using_entry parameters

Name	Description
<i>in_entry_num</i>	CAM entry number. May not exceed the total number of CAM entries. It is not range checked.

Name	Description
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.19.3.8 cam_entry_lookup

Prototype:

```
cam_entry_lookup(out_lookup_result, in_key, in_bid, in_num_entries, in_first_entry)
```

Description:

Perform a CAM lookup and increment the reference count in CAM sharing mode.

The following algorithm is used to increment the reference count:

<verbatim> Say the @camsharing_ref_cnt1 has: 0011 ... 0100 0011 0011 0001 To increment ref_cnt for entry 1, add: 0000 ... 0000 0000 0001 0000 Result: 0011 ... 0100 0011 0100 0001 </verbatim>

Example:

```
cam_entry_lookup(lookup_result, src_key, RX_CAM_HANDLE)
```

Table 2.223. cam_entry_lookup parameters

Name	Description
<i>out_lookup_result</i>	The lookup result is returned here. For CAM sharing mode the result has the block specific entry number.
<i>in_key</i>	Source key with which CAM lookup should be done.
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.19.3.9 cam_entry_lookup_with_lm

Prototype:

```
cam_entry_lookup_with_lm(out_lookup_result, in_key, in_lm_handle, in_lm_start_addr, in_bid, in_num_entries, in_first_entry)
```

Description:

Perform a CAM lookup, increment the reference count in CAM sharing mode and set the given LM handle to point to corresponding data in Local Memory.

The following algorithm is used to increment the reference count:

<verbatim> Say the @camsharing_ref_cnt1 has: 0011 ... 0100 0011 0011 0001 To increment ref_cnt for entry 1, add: 0000 ... 0000 0000 0001 0000 Result: 0011 ... 0100 0011 0100 0001 </verbatim>

Example:

```
cam_entry_lookup_with_lm(lookup_result, src_key, RXC_LM_HANDLE, RXC_INFO_LM_BASE, \
    RX_CAM_HANDLE)
```

Table 2.224. cam_entry_lookup_with_lm parameters

Name	Description
<i>out_lookup_result</i>	The lookup result is returned here. For CAM sharing mode the result has the block specific entry number.
<i>in_key</i>	Source key with which CAM lookup should be done.
<i>in_lm_handle</i>	LM handle 0 or 1 which will be set to point to the data structure in LM according to the lookup result.
<i>in_lm_start_addr</i>	2-bit LM address where the data structure is located. (0, 1, 2, 3 represents 0, 1024, 2048, 3072 bytes in LM)
<i>in_bid</i>	Microblock Block ID.
<i>in_num_entries</i>	Number of CAM entries for this microblock.
<i>in_first_entry</i>	First CAM entry number of this microblock.

2.20 Microengine CRC

2.20.1 Microengine CRC Macros

This section creates a global absolute register: @crc10_table_base_addr

2.20.2.1 crc_load_crc10_table

Prototype:

```
crc_load_crc10_table(lm_crc10_base_addr)
```

Description:

Load crc_10 lookup table.



Note

The user needs to make sure this area is not corrupted later.

Instruction Count: 130

Table 2.225. `crc_load_crc10_table` parameters

Name	Description
<code>lm_crc10_base_addr</code>	Base address from where CRC lookup table should start in local memory

2.20.2.2 `crc_10`

Prototype:

```
crc_10(out_remainder, in_data, ENDIAN, in_start_byte, in_end_byte)
```

Description:

Perform a crc 10 computation.

The CRC Remainder CSR must contain the remainder intended to be used in this CRC 10 operation (usually it is the remainder of the previous CRC operation).



Note

If `in_start_byte = in_end_byte`, only values 0 or 3 are legal. This means only the byte at either end of `in_data` can be used in the CRC operation.

`in_start_byte` and `in_end_byte` should be consistent with the byte order specified in `ENDIAN`. Example: If you want to use only the least significant byte:

- `ENDIAN = big_endian` ---> `in_start_byte = in_end_byte = 3`
- `ENDIAN = little_endian` ---> `in_start_byte = in_end_byte = 0`

Each time this operation is performed, the new remainder is put in the CRC Remainder CSR.

Instruction Count: 27 for 1 byte, 40 for 2 bytes, 54 for 3 bytes, 68 for 4 bytes

Table 2.226. `crc_10` parameters

Name	Description
<code>out_remainder</code>	Value of remainder. Can be GPR or sram/dram transfer out register (a.k.a. write transfer register).
<code>in_data</code>	Source data on which CRC operation will be performed.
<code>ENDIAN</code>	String with two possible values: <ul style="list-style-type: none"> • <code>little_endian</code>: use <code>crc_le</code> operation (the data will be swaped before being used to calculate CRC remainder) • <code>big_endian</code>: use <code>crc_be</code> operation (no data swapping)
<code>in_start_byte</code>	Byte position where data starts. Must be an immediate value (a number). <code>in_start_byte</code> must be less than or equal <code>in_end_byte</code> .

Name	Description
<i>in_end_byte</i>	Byte position where data ends. Must be an immediate value (a number).

2.20.2.3 crc_32

Prototype:

```
crc_32(out_remainder, in_data, ENDIAN, in_start_byte, in_end_byte)
```

Description:

Perform a crc 32 computation.

The CRC Remainder CSR must contain the remainder intended to be used in this CRC 32 operation (usually it is the remainder of the previous CRC operation).



Note

If `in_start_byte == in_end_byte`, only values 0 or 3 are legal. This means only the byte at either end of `in_data` can be used in the CRC operation.

`in_start_byte` and `in_end_byte` should be consistent with the byte order specified in `ENDIAN`. Example: If you want to use only the least significant byte:

- `ENDIAN = big_endian` ---> `in_start_byte = in_end_byte = 3`
- `ENDIAN = little_endian` ---> `in_start_byte = in_end_byte = 0`

Each time this operation is performed, the new remainder is put in the CRC Remainder CSR.

Instruction Count: 1

Table 2.227. crc_32 parameters

Name	Description
<i>out_remainder</i>	The value of remainder. Can be a GPR or a sram/dram transfer out register (a.k.a. write transfer register).
<i>in_data</i>	Source data on which CRC operation will be performed
<i>ENDIAN</i>	String with two possible values: <ul style="list-style-type: none"> • <code>little_endian</code>: use <code>crc_le</code> operation (the data will be swapped before being used to calculate CRC remainder) • <code>big_endian</code>: use <code>crc_be</code> operation (no data swapping)
<i>in_start_byte</i>	Byte position where data starts. Must be a immediate value (a number). <code>in_start_byte</code> must be less than or equal <code>in_end_byte</code>
<i>in_end_byte</i>	Byte position where data ends. Must be an immediate value (a number).

2.20.2.4 crc_ccitt

Prototype:

```
crc_ccitt(out_remainder, in_data, ENDIAN, in_start_byte, in_end_byte)
```

Description:

Perform a crc_ccitt computation.

The CRC Remainder CSR must contain the remainder intended to be used in this CRC operation (usually it is the remainder of the previous CRC operation).



Note

If `in_start_byte == in_end_byte`, only value 0 or 3 are legal. This means only the byte at either end of `in_data` can be used in the CRC operation.

`in_start_byte` and `in_end_byte` should be consistent with the byte order specified in `ENDIAN`. Example: If you want to use only the least significant byte:

- `ENDIAN = big_endian ---> in_start_byte = in_end_byte = 3`
- `ENDIAN = little_endian ---> in_start_byte = in_end_byte = 0`

Each time this operation is performed, the new remainder is put in the CRC Remainder CSR.

NFP-32xx can perform CRC operation on big or little endian data.

Instruction Count: 1

Table 2.228. crc_ccitt parameters

Name	Description
<code>out_remainder</code>	Value of remainder. Can be a GPR or a sram/dram transfer out register (a.k.a. write transfer register).
<code>in_data</code>	Source data on which CRC operation will be performed
<code>ENDIAN</code>	String with two possible values: <ul style="list-style-type: none"> • <code>little_endian</code>: use <code>crc_le</code> operation (the data will be swapped before being used to calculate CRC remainder) • <code>big_endian</code>: use <code>crc_be</code> operation (no data swapping)
<code>in_start_byte</code>	Byte position where data starts. Must be a immediate value (a number). <code>in_start_byte</code> must be less than or equal <code>in_end_byte</code> .
<code>in_end_byte</code>	Byte position where data ends. Must be an immediate value (a number).

2.20.2.5 crc_read

Prototype:

```
crc_read(out_remainder)
```

Description:

Read the data that is currently in the CSR CRC_Remainder.

Table 2.229. crc_read parameters

Name	Description
<i>out_remainder</i>	Value of remainder. Can be a GPR or a sram/dram transfer out register (a.k.a. write transfer register).

2.20.2.6 crc_write

Prototype:

```
crc_write(in_remainder)
```

Description:

Write the input data into the CRC unit.



Note

This operation needs 3 cycles for data to be updated. Therefore, there must be at least 3 instructions between calling this macro and reading the CRC Remainder or using the remainder via `crc_ccitt` or `CRC_32` operation.

Instruction Count: 1

Table 2.230. crc_write parameters

Name	Description
<i>in_remainder</i>	Value of remainder to be written. Can be a GPR or a sram/dram transfer in register (a.k.a. read transfer register).

2.21 Microengine Standard Macros

2.21.1 Microengine Standard Macros

This paragraph provides a list of generic macros

2.21.2.1 `immed32`

Prototype:

```
immed32(out_result, VAL)
```

Description:

Load `out_result` with 32 bit constant.

If constant fits in 16 bits, expand to a single `immed`, if not, expand into an `immed_w0` and an `immed_w1`. If it is a name (not a number), assume that it is being imported (an import variable).

Example:

```
immed32(output, 0x12345678)
```

Instruction Count: 1 to 2 (3 if output is transfer register)

Table 2.231. `immed32` parameters

Name	Description
<code>out_result</code>	Destination GPR or write transfer register
<code>VAL</code>	Constant

2.21.2.2 `immed40`

Prototype:

```
immed40(out_result_hi, out_result_lo, VAL)
```

Description:

Load the upper byte of the 40-bit immediate value in the upper byte of `out_result_hi` and load the lower 4 bytes into `out_result_lo`. Can be used with immediate constant or import variable.

This macro is specifically for defining a 40-bit addresses

Example: `.reg addr_hi, addr_lo .reg write $wr_buf .sig s`


```
define TEST_BASE_ADDR_SIZE 0x100 define TEST_BASE_ADDR_ALIGN 0x4 .alloc_mem
TEST_BASE_ADDR i26.emem island TEST_BASE_ADDR_SIZE TEST_BASE_ADDR_ALIGN
```

```
immed40(addr_hi, addr_lo, TEST_BASE_ADDR) move($wr_buf, 0x12345678) mem[write8_be, $wr_buf, addr_hi,
<<8, addr_lo, 4], sig_done[s] ctx_arb[s], all
```

Table 2.232. immed40 parameters

Name	Description
<i>out_result_hi</i>	Destination GPR or xfer register
<i>out_result_lo</i>	Destination GPR or xfer register
<i>VAL</i>	40-bit Constant

2.21.2.3 balr

Prototype:

```
balr(in_link_pc, target_label)
```

Description:

Save the current program counter in GPR *in_link_pc* and branch to *target_label*.

Program control can be returned to the current PC (the current location) using the *rtn[in_link_pc]* instruction.

Example:

```
balr(next, target)
```

Instruction Count: 2

Table 2.233. balr parameters

Name	Description
<i>in_link_pc</i>	GPR to hold next PC for use by <i>rtn</i> instruction
<i>target_label</i>	Subroutine start

2.21.2.4 move

Prototype:

```
move(out_result, in_src)
```

Description:

Move the source value into the destination register.

Example:

```
move(output, input)
```

Instruction Count: 1 to 3 (2 if output is GPR, 3 if output is transfer register)

Table 2.234. move parameters

Name	Description
<i>out_result</i>	Destination GPR or write transfer register
<i>in_src</i>	Source GPR, read transfer register or constant

2.21.2.5 alu_op

Prototype:

```
alu_op(out_result, in_a, op_spec, in_b)
```

Description:

Perform ALU operation in_aop_specin_b.

Example:

```
alu_op(output, 0x12345678, +, 10)
```

Instruction Count: 1 to 5 (6 if op_spec = +4)

Table 2.235. alu_op parameters

Name	Description
<i>out_result</i>	Destination GPR or write transfer register
<i>in_a</i>	Register or constant
<i>op_spec</i>	ALU operation Legal operators are: <ul style="list-style-type: none"> • B • ~B • + • +carry • +4 • +8 • +16 • - • B-A • AND • ~AND

Name	Description
	<ul style="list-style-type: none"> • AND~ • OR • XOR
<i>in_b</i>	Register or constant

2.21.2.6 add

Prototype:

```
add(out_result, in_a, in_b)
```

Description:

Perform 32 bit add.

This macro calculates:

```
out_result = in_a + in_b
```

Example:

```
add(output, 0x1234, 0x12345678)
```

Instruction Count: 1 to 5

Table 2.236. add parameters

Name	Description
<i>out_result</i>	Destination GPR or write transfer register
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant

2.21.2.7 add_c

Prototype:

```
add_c(out_result, in_a, in_b)
```

Description:

Perform 32 bit add.

This macro calculates:

```
out_result = in_a + in_b + previous carry
```

Example:

```
add_c(output, source_a, 0x12345678)
```

Instruction Count: 1 to 5

Table 2.237. add_c parameters

Name	Description
<i>out_result</i>	Destination GPR or write transfer register
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant

2.21.2.8 sub

Prototype:

```
sub(out_result, in_a, in_b)
```

Description:

Perform 32 bit subtract.

This macro calculates:

```
out_result = in_a - in_b
```

Example:

```
sub(output, 0x12345678, source)
```

Instruction Count: 1 to 5

Table 2.238. sub parameters

Name	Description
<i>out_result</i>	Destination GPR or write transfer register
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant

2.21.2.9 shf_right

Prototype:

```
shf_right(out_result, in_src, in_shift_amt)
```

Description:

Shift right *in_src* by *in_shift_amt* bit positions.

Example:

```
shf_right(output, input, 3)
```

Instruction Count: 1 to 2

Table 2.239. shf_right parameters

Name	Description
<i>out_result</i>	Destination
<i>in_src</i>	Source
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.10 shf_left

Prototype:

```
shf_left(out_result, in_src, in_shift_amt)
```

Description:

Shift left *in_src* by *in_shift_amt* bit positions.

Example:

```
shf_left(output, input, reg)
```

Instruction Count: 1 to 4

Table 2.240. shf_left parameters

Name	Description
<i>out_result</i>	Destination
<i>in_src</i>	Source
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.11 rot_right

Prototype:

```
rot_right(out_result, in_src, in_shift_amt)
```

Description:

Rotate right *in_src* by *in_shift_amt* bit positions.

Example:

```
rot_right(output, input, 3)
```

Instruction Count: 1

Table 2.241. rot_right parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_src</i>	Source GPR
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.12 rot_left

Prototype:

```
rot_left(out_result, in_src, in_shift_amt)
```

Description:

Rotate left *in_src* by *in_shift_amt* bit positions.

Example:

```
rot_left(output, input, 3)
```

Instruction Count: 1

Table 2.242. rot_left parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_src</i>	Source GPR
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.13 alu_shf_right

Prototype:

```
alu_shf_right(out_result, in_a, op_spec, in_b, in_shift_amt)
```

Description:

Shift right *in_b* by *in_shift_amt* bit positions, then perform operation *op_spec* on *in_a*.

Example:

```
alu_shf_right(output, 0xABCD1234, +, input, 3)
```

Instruction Count: 1 to 4

Table 2.243. alu_shf_right parameters

Name	Description
<i>out_result</i>	Destination
<i>in_a</i>	Register or constant
<i>op_spec</i>	<p>ALU operation Legal operators are:</p> <ul style="list-style-type: none"> • B • ~B • + • +carry • +4 • +8 • +16 • - • -carry • B-A • AND • ~AND • AND~ • OR • XOR
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.14 add_shf_right

Prototype:

```
add_shf_right(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift right *in_b* by *in_shift_amt* bit positions, then ADD *in_a*.

Example:

```
add_shf_right(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.244. add_shf_right parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.15 sub_shf_right

Prototype:

```
sub_shf_right(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift right *in_b* by *in_shift_amt* bit positions, then SUBTRACT *in_a*.

Example:

```
sub_shf_right(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.245. sub_shf_right parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.16 and_shf_right

Prototype:

```
and_shf_right(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift right *in_b* by *in_shift_amt* bit positions, then AND *in_a*.

Example:

```
and_shf_right(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.246. and_shf_right parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.17 or_shf_right

Prototype:

```
or_shf_right(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift right *in_b* by *in_shift_amt* bit positions, then OR *in_a*.

Example:

```
or_shf_right(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.247. or_shf_right parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.18 alu_shf_left

Prototype:

```
alu_shf_left(out_result, in_a, op_spec, in_b, in_shift_amt)
```

Description:

Shift left *in_b* by *in_shift_amt* bit positions, then perform operation *op_spec* on *in_a*.

Example:

```
alu_shf_left(output, 0xABCD1234, +, input, 3)
```

Instruction Count: 1 to 4

Table 2.248. alu_shf_left parameters

Name	Description
<i>out_result</i>	Destination
<i>in_a</i>	Register or constant
<i>op_spec</i>	<p>ALU operation Legal operators are:</p> <ul style="list-style-type: none"> • B • ~B • + • +carry • +4 • +8 • +16 • - • -carry • B-A • AND • ~AND • AND~ • OR • XOR
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.19 add_shf_left

Prototype:

```
add_shf_left(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift left *in_b* by *in_shift_amt* bit positions, then ADD *in_a*.

Example:

```
add_shf_left(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.249. add_shf_left parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.20 sub_shf_left

Prototype:

```
sub_shf_left(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift left *in_b* by *in_shift_amt* bit positions, then SUBTRACT *in_a*.

Example:

```
sub_shf_left(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.250. sub_shf_left parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.21 and_shf_left

Prototype:

```
and_shf_left(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift left *in_b* by *in_shift_amt* bit positions, then AND *in_a*.

Example:

```
and_shf_left(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.251. and_shf_left parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.22 or_shf_left

Prototype:

```
or_shf_left(out_result, in_a, in_b, in_shift_amt)
```

Description:

Shift left *in_b* by *in_shift_amt* bit positions, then OR *in_a*.

Example:

```
or_shf_left(output, 0xABCD1234, input, 3)
```

Instruction Count: 1 to 4

Table 2.252. or_shf_left parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.23 alu_rot_right

Prototype:

```
alu_rot_right(out_result, in_a, op_spec, in_b, in_shift_amt)
```

Description:

Rotate right *in_b* by *in_shift_amt* bit positions, then perform operation *op_spec* on *in_a*.

Example:

```
alu_rot_right(output, input2, +, input, 4)
```

When: *input2* = 5 and *input* = 0x12345678 results in *out_result* = 0x8123456c

Instruction Count: 1 to 3 (4 if op_spec = +4)

Table 2.253. alu_rot_right parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>op_spec</i>	ALU operation Legal operators are: <ul style="list-style-type: none"> • B • ~B • + • +carry • +4 • +8 • +16 • - • B-A • AND • ~AND • AND~ • OR • XOR
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.24 alu_rot_left

Prototype:

```
alu_rot_left(out_result, in_a, op_spec, in_b, in_shift_amt)
```

Description:

Rotate left *in_b* by *in_shift_amt* bit positions, then perform operation *op_spec* on *in_a*.

Example:

```
alu_rot_left(output, input2, +, input, 3)
```

Instruction Count: 1 to 3 (4 if op_spec = +4)

Table 2.254. alu_rot_left parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>in_a</i>	Register or constant
<i>op_spec</i>	<p>ALU operation Legal operators are:</p> <ul style="list-style-type: none"> • B • ~B • + • +carry • +4 • +8 • +16 • - • B-A • AND • ~AND • AND~ • OR • XOR
<i>in_b</i>	Register or constant
<i>in_shift_amt</i>	Register or constant (0 to 31)

2.21.2.25 bitfield_extract

Prototype:

```
bitfield_extract(out_result, in_src, MSB, LSB)
```

Description:

Extract a bit field from a register.



Note

Bits are numbered 31-0, left to right.

Instruction Count: 1 to 6

Table 2.255. bitfield_extract parameters

Name	Description
<i>out_result</i>	Extracted field
<i>in_src</i>	Source register with multiple fields
<i>MSB</i>	Most significant, left bit defining field
<i>LSB</i>	Least significant, right bit defining field

2.21.2.26 bitfield_insert

Prototype:

```
bitfield_insert(out_result, in_a, in_b, MSB, LSB)
```

Description:

Insert bitfield *in_b* into copy of *in_a*.



Note

Bits are numbered 31-0, left to right.

Instruction Count: 2 to 4 (depending on size of bitfield, 2 if mask fits in immediate)

Table 2.256. bitfield_insert parameters

Name	Description
<i>out_result</i>	Merged result
<i>in_a</i>	Register with multiple fields
<i>in_b</i>	Bits to be inserted (This field always start at bit 0. Do not shift to the position to be inserted.)
<i>MSB</i>	Most significant, left bit (bits left to right are 31-0)
<i>LSB</i>	Least significant, right bit

2.21.2.27 bits_clr

Prototype:

```
bits_clr(io_data, in_start_bit_pos, in_mask)
```

Description:

Clear bits indicated by mask at starting position *in_start_bit_pos*.

Example:

```
bits_clr(reg2, bitpos, 0x3)
```

Instruction Count: 4 to 7

Table 2.257. bits_clr parameters

Name	Description
<i>io_data</i>	Register to modify
<i>in_start_bit_pos</i>	Register or constant less than 32, starting bit position <i>in_mask</i> is shifted left by this amount
<i>in_mask</i>	Register or constant, mask of bits to clear

2.21.2.28 bits_set

Prototype:

```
bits_set(io_data, in_start_bit_pos, in_mask)
```

Description:

Set bits indicated by mask at starting position *in_start_bit_pos*.

Example:

```
bits_set(reg2, 5, 0xff)
```

Instruction Count: 3 to 5

Table 2.258. bits_set parameters

Name	Description
<i>io_data</i>	Register to modify
<i>in_start_bit_pos</i>	Register or constant less than 32, starting bit position <i>in_mask</i> is shifted left by this amount
<i>in_mask</i>	Register or constant, mask of bits to set

2.21.2.29 multiply

Prototype:

```
multiply(out_result, in_a, in_b)
```

Description:

32-bit multiplication with 16x16 operand size.

This macro calculates:

```
out_result = in_a * in_b
```




Note

`in_b` is known at compile time, so the pre-processor uses its bit value to construct an optimal multiply via ALU/shift operations. A simple implementation would take every 1-bit position `n` in `in_b`, and shift and add `mpy_in` by `n` to accumulate the `out_result`. So if there were `B` 1-bits in `in_b`, this would take `B` instructions -- up to 32. This implementation also uses subtraction, reducing the maximum number of instructions to 16. For example given `in_b = 123 = 1111011b` $123x = 64x + 32x + 16x + 8x + 2x + 1x$ but also: $123x = 128x - 4x - 1x$

Performance: `n` cycles: 1 to 16 Worst case performance is the number of 1 bits in `in_b`. `0xAAAAAAAA` and `0x55555555` take the maximum 16 cycles. Actual performance depends on `in_b` like so:

1. Spans of two or more 1's cost 2 cycles. 1a) The exception is if there is only 1 span of 1's and it is adjacent to bit 31 -- this costs 1 cycle.
2. Isolated 0's or 1's cost 1 cycle each.

Resources:

- Memory: no accesses, none consumed
- Registers: no additional registers, just `out_result` & input parameters.
- Instructions = `n`: (`n` <= 16), as described Performance.

Limitations: `mpy_in` and `out_result` typically cannot be the same GPR. This is because `add_shf_left` and `sub_shf_left` macros will operate on both `out_result` and `mpy_in` in the same cycle and would result in a compile-time error. However, `mpy_in` and `out_result` can be the same GPR in cases where the macro expands to just a single instruction. For example, if `in_b` is zero or a positive or negative even power of two.

Algorithm: Start scanning the `in_b` from one end looking for the first 1-bit. If it is a single 1-bit, emit an `add_shift` and continue. Otherwise, start counting 1-bits until you find two or more 0-bits in a row. Then add the first of the two zero bits, subtract the first 1-bit, and subtract any single 0-bits along the way.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 3 to 4

Table 2.259. multiply parameters

Name	Description
<code>out_result</code>	Destination GPR
<code>in_a</code>	Multiplicand, GPR
<code>in_b</code>	Multiplier, any 32-bit input constant

2.21.2.30 divide

Prototype:

```
divide(out_result, dividend, DIVISOR)
```

Description:

Divide.

This macro calculates:

```
out_result = dividend / DIVISOR
```



Note

Performance: Variable/CONSTANT: 1 cycle CONSTANT/CONSTANT: 1 - 2 cycles

Resources: Memory: no accesses, none consumed. Registers: no additional registers, just `out_result` and input parameters. Instructions: 1 or 2

Limitations: Power of 2 constant DIVISOR

Algorithm: Determine what DIV_SHIFT yields ($2^{\text{DIV_SHIFT}} == \text{DIVISOR}$). Right shift divisor by DIV_SHIFT.

Condition Codes: Data dependent: assume they are clobbered.

Instruction Count: 1 to 2

Table 2.260. divide parameters

Name	Description
<i>out_result</i>	Destination GPR
<i>dividend</i>	GPR or CONSTANT
<i>DIVISOR</i>	GPR or CONSTANT For variable dividend DIVISOR must be constant power of 2 For CONSTANT dividend DIVISOR can be any CONSTANT

2.21.2.31 array_index_from_elem_addr

Prototype:

```
array_index_from_elem_addr(out_index, in_address, BASE_ADDRESS, ELEMENT_SIZE)
```

Description:

Convert an address into an array index.

This macro calculates:

```
out_index = (in_address / ELEMENT_SIZE) - (BASE_ADDRESS / ELEMENT_SIZE)
```



Note

Limitations: Attempts to compile for non power of 2 `ELEMENT_SIZE` will fail because divide does not currently support non power of 2 divisors.

Instruction Count: 3 (Depends on `ELEMENT_SIZE` and `BASE_ADDRESS`)

Table 2.261. `array_index_from_elem_addr` parameters

Name	Description
<code>out_index</code>	Index of the array element
<code>in_address</code>	Array element's address
<code>BASE_ADDRESS</code>	Memory Address of 1st element in the array
<code>ELEMENT_SIZE</code>	Size of Array Element in address units

2.21.2.32 `elem_addr_from_array_index`

Prototype:

```
elem_addr_from_array_index(out_address, in_index, BASE_ADDRESS, ELEMENT_SIZE)
```

Description:

Convert an array element address to an array index.

This macro calculates:

```
if (ELEMENT_SIZE is power of 2)
    out_address = (in_index << BUF_DAFI_SHIFT) + BASE_ADDRESS
else
    out_address = (in_index * ELEMENT_SIZE) + BASE_ADDRESS
```

Instruction Count: 1 to 21 (Depends on `ELEMENT_SIZE` and `BASE_ADDRESS`). Best: 1 = 1 ALU. Typical: 3 = 2 immeds + 1 ALU. Worst: 21 = 4 immeds + 17 ALU

Table 2.262. `elem_addr_from_array_index` parameters

Name	Description
<code>out_address</code>	Address of the array element
<code>in_index</code>	Index of the array element
<code>BASE_ADDRESS</code>	Memory Address of 1st element in the array
<code>ELEMENT_SIZE</code>	Size of Array Element in address units

2.21.2.33 arith_shf_right

Prototype:

```
arith_shf_right(out_result, in_src, in_shift_amt, SIGN_BIT_POS)
```

Description:

Shift right depend on the following cases:

1. If `SIGN_BIT_POS` is set to 31, no sign extend will occur, just arithmetic shift right.

Table 2.263. arith_shf_right parameters

Name	Description
<i>out_result</i>	Sign-extended shifted-right result
<i>in_src</i>	Source operand.
<i>in_shift_amt</i>	Register of constant right shift amount, values from 0-31 but cannot be greater than <code>SIGN_BIT_POS</code> value
<i>SIGN_BIT_POS</i>	Constant position of sign bit. 31 if <i>in_src</i> is already sign-extended.

2.21.2.34 multiply32

Prototype:

```
multiply32(out_result, in_a, in_b, OPERAND_SIZE)
```

Description:

32 bit multiply optimized for `OPERAND_SIZE` specifications.

This macro calculates:

```
out_result = in_a * in_b
```

Instruction Count: 3 to 4 (3 for `OP_SIZE_8x24` and 4 for `OP_SIZE_16x16`)

Table 2.264. multiply32 parameters

Name	Description
<i>out_result</i>	32 bit multiply result
<i>in_a</i>	Multiplicand
<i>in_b</i>	Multiplier
<i>OPERAND_SIZE</i>	Operand size Valid sizes: <ul style="list-style-type: none"> • <code>OP_SIZE_8X24</code>: Multiplier is 8 bits, multiplicand is 24 bits (8x24) • <code>OP_SIZE_16X16</code>: Multiplier and multiplicand are 16 bits (16x16)

2.21.2.35 multiply64

Prototype:

```
multiply64(out_result_hi, out_result_lo, in_a, in_b, OPERAND_SIZE)
```

Description:

64 bit multiply optimized for OPERAND_SIZE specifications.

This macro calculates:

```
out_result = in_a * in_b
```

Instruction Count: 5 to 7 (5 for OP_SIZE_16x32 and 7 for OP_SIZE_32x32)

Table 2.265. multiply64 parameters

Name	Description
<i>out_result_hi</i>	Most significant 32 bits of multiply result
<i>out_result_lo</i>	Least significant 32 bits of multiply result
<i>in_a</i>	Multiplicand
<i>in_b</i>	Multiplier
<i>OPERAND_SIZE</i>	Operand size Valid sizes: <ul style="list-style-type: none"> OP_SIZE_16X32: Multiplier is 16 bits, multiplicand is 32 bits OP_SIZE_32X32: Multiplier and multiplicand are 32 bits

2.21.2.36 rand

Prototype:

```
rand(out_result)
```

Description:

Generate a pseudo-random number.

Table 2.266. rand parameters

Name	Description
<i>out_result</i>	Random number

2.21.2.37 srand

Prototype:

```
srand(in_src)
```

Description:

Set pseudo-random number seed.

Table 2.267. srand parameters

Name	Description
<i>in_src</i>	Random number seed, constant or register

2.22 Override Macros

2.22.1 Override flags

Names of flags that can be provided to fields that can be provided to `ov_global()`, `ov_start()`, `ov_set*()` and `ov_*use()` to fine-tune the functionality provided by these macros. Flags provided to `ov_global()` are the default flags used every time `ov_start()` is called. `ov_global()` can be called at any time, but the value is only used during a call to `ov_start()`. Flags provided to `ov_start()` override the current global flags, and are used for the entire indirect reference, even if it is stored and later recalled or resumed. Flags provided to `ov_set*()` and `ov_*use()` override the current indirect reference's flags, and are used for only that specific macro.

2.22.2.1 ov_global_flags

Prototype:

```
ov_global_flags(FLAGS)
```

Description:

Set the values of the global flags, from this point forward.

This macro can be called at any time. A copy of these flags is used as a starting point when `ov_start()` is called.

Example:

```
ov_global_flags((OVF_NO_CTX_SWAP_OFF | OVF_RELAXED_ORDER_ON))
```

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

Table 2.268. ov_global_flags parameters

Name	Description
<i>FLAGS</i>	Zero or more flags from <i>Override flags</i> , OR'd together. Flags not specified are assumed to be <code>_OFF</code> .

2.22.2.2 ov_start

Prototype:

```
ov_start(OVERRIDE_TYPES, FLAGS)
```

Description:

Start an indirect reference.

This macro creates the context within which the indirect reference is determined. It is always the first step of an indirect reference, and may only be called if an indirect reference is not already in progress.

Example:

```
ov_start((OV_LENGTH | OV_DATA_REF), OVF_REUSE_ON)
```



Note

The validity of the provided parameters are checked against the indirect reference environment currently being compiled for.



Note

Constants ORd together must be enclosed within round brackets.

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

Table 2.269. ov_start parameters

Name	Description
<i>OVERRIDE_TYPES</i>	One or more override fields from <i>Override fields</i> , ORd together.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , ORd together, which are used for the entire duration of this indirect reference. Flags specified here override the global flags most recently specified using <i>ov_global_flags()</i> .

2.22.2.3 ov_set_bits

Prototype:

```
ov_set_bits(OVERRIDE_TYPE, BIT_MASK, BIT_CONSTANT)
```

Description:

Set bits within the mask of the indicated field to the provided value.

This macro is typically useful to specify constant bits that indicate which subcommand is required.

Example:

```
ov_start(OV_LENGTH)
ov_set_bits(OV_LENGTH, 0b11000, 0b01000) // Clear bit 4 and set bit 3
ov_set(OV_LENGTH, 2, 0, 3)
ov_use()
command[...], indirect_ref
ov_clean()
```



Note

BIT_MASK and *BIT_CONSTANT* must be constant values.

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

Table 2.270. ov_set_bits parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override</i> fields that was specified in <i>ov_start()</i> .
<i>BIT_MASK</i>	Mask indicating which bits must be set. Valid formats are decimal, hexadecimal and binary (prefixed with "0b").
<i>BIT_CONSTANT</i>	Constant indicating value of bit for each bit set in <i>BIT_MASK</i> . Valid formats are decimal, hexadecimal and binary (prefixed with "0b").

2.22.2.4 ov_set_bits

Prototype:

```
ov_set_bits(OVERRIDE_TYPE, DST_MSB, DST_LSB, BIT_CONSTANT)
```

Description:

Set the specific range of bits in the indicated field to the provided value.

This macro is typically useful to specify a range of constant bits that are not used in the instruction that this indirect reference is going to be used with.

Example:

```
ov_start(OV_LENGTH)
ov_set_bits(OV_LENGTH, 4, 3, 0) // Clear bits 4 and 3
ov_set(OV_LENGTH, 2, 0, 3)
ov_use()
command[...], indirect_ref
ov_clean()
```




Note

BIT_CONSTANT is relative to *DST_LSB*, i.e. *BIT_CONSTANT* will be shifted to the left by *DST_LSB* before being used.

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

Table 2.271. *ov_set_bits* parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override</i> fields that was specified in <i>ov_start()</i> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>BIT_CONSTANT</i>	Constant to set the bits <i>DST_MSB:DST_LSB</i> to.

2.22.2.5 *ov_set*

Prototype:

```
ov_set(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, FLAGS)
```

Description:

Set the specific range of bits in the indicated field to the provided value, taking the provided flags into account.

Example:

```
ov_start(OV_LENGTH)
ov_set(OV_LENGTH, 4, 4, 0)
ov_set(OV_LENGTH, 3, 0, 1, OVF_RELAXED_ORDER_ON)
ov_use()
command[...], indirect_ref
ov_clean()
```



Note

in_value is relative to *DST_LSB*, i.e. *in_value* will be shifted to the left by *DST_LSB* before being used.

Condition Codes: Clobbered (*in_value* a register value) or unchanged (*in_value* a constant).

Instruction Count: One or more (*in_value* a register value) or zero (*in_value* a constant).

Table 2.272. ov_set parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override</i> fields that was specified in <i>ov_start()</i> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override</i> flags, ORd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.6 ov_set

Prototype:

```
ov_set(OVERRIDE_TYPE, in_value, FLAGS)
```

Description:

Set the indicated field to the provided value, taking the provided flags into account.

Example:

```
ov_start(OV_LENGTH)
ov_set(OV_LENGTH, 12, OV_SUBTRACT_ONE)
ov_use()
command[...], indirect_ref
ov_clean()
```

Condition Codes: Clobbered (*in_value* a register value) or unchanged (*in_value* a constant).

Instruction Count: One or more (*in_value* a register value) or zero (*in_value* a constant).

Table 2.273. ov_set parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override</i> fields that was specified in <i>ov_start()</i> .
<i>in_value</i>	Value to set field <i>OVERRIDE_TYPE</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override</i> flags, ORd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.7 ov_set_bm_and

Prototype:

```
ov_set_bm_and(in_byte_mask, OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, FLAGS)
```

Description:

Set the OV_BYTE_MASK field to the provided value and also set the specific range of bits in the indicated field to the provided value, taking the provided flags into account.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, FLAGS)
```

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER |
         OV_SIGNAL_MASTER | OV_MASTER_ISLAND))
ov_set(OV_DATA_MASTER, 3, 2, 0)
ov_set_bm_and(byte_mask, OV_DATA_MASTER, 1, 0, data_master, OVF_PARANOID_ON)
ov_set(OV_DATA_REF, data_ref)
ov_set(OV_SIGNAL_MASTER, signal_master)
ov_set(OV_MASTER_ISLAND, mater_island)
ov_set(OV_LENGTH, len, OVF_SUBTRACT_ONE)
ov_use()
command[...], indirect_ref
ov_clean()
```



Note

The parameter *in_byte_mask* is used in its entirety as is. No provided parameters are applicable to it.

in_value is relative to *DST_LSB*, i.e. *in_value* will be shifted to the left by *DST_LSB* before being used.

Condition Codes: Clobbered

Instruction Count: One or more

Table 2.274. ov_set_bm_and parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field OV_BYTE_MASK.
<i>OVERRIDE_TYPE</i>	One override field from <code>Override fields</code> that was specified in <code>ov_start()</code> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> in <i>OVERRIDE_TYPE</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <code>Override flags</code> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.8 ov_set_bm_and

Prototype:

```
ov_set_bm_and(in_byte_mask, OVERRIDE_TYPE, in_value, FLAGS)
```

Description:

Set the OV_BYTE_MASK field to the provided value and also set the indicated field to the provided value, taking the provided flags into account.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set(OVERRIDE_TYPE, in_value, FLAGS)
```

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER |
         OV_SIGNAL_MASTER | OV_MASTER_ISLAND))
ov_set_bm_and(byte_mask, OV_DATA_MASTER, data_master, OVF_PARANOID_ON)
ov_set(OV_DATA_REF, data_ref)
ov_set(OV_SIGNAL_MASTER, signal_master)
ov_set(OV_MASTER_ISLAND, mater_island)
ov_set(OV_LENGTH, len, OVF_SUBTRACT_ONE)
ov_use()
command[...], indirect_ref
ov_clean()
```



Note

The parameter *in_byte_mask* is used in its entirety as is. No provided parameters are applicable to it.

Condition Codes: Clobbered

Instruction Count: One or more

Table 2.275. ov_set_bm_and parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field OV_BYTE_MASK.
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>in_value</i>	Value to set the field <i>OVERRIDE_TYPE</i> to. May be a constant or a register value.

Name	Description
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.9 ov_set_extract

Prototype:

```
ov_set_extract(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

Description:

Set the specific range of bits in the indicated field to the extracted range of bits in the provided value, taking the provided flags into account.

The data is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start(OV_LENGTH)
ov_set(OV_LENGTH, 4, 4, 0)
ov_set_extract(OV_LENGTH, 3, 0, value, 12, 9, OV_RELAXED_ORDER_ON)
ov_use()
command[...], indirect_ref
ov_clean()
```



Note

Bits in the range *SRC_MSB*:*SRC_LSB* are extracted from *in_value*, and are placed in the indicated field relative to *DST_LSB* and , i.e. *in_value* will be masked and effectively shifted to the left by (*DST_LSB* - *SRC_LSB*) bit positions before being used.

Condition Codes: Clobbered (*in_value* a register value) or unchanged (*in_value* a constant).

Instruction Count: Two or more (*in_value* a register value) or zero (*in_value* a constant).

Table 2.276. ov_set_extract parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.

Name	Description
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , ORd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.10 ov_set_extract

Prototype:

```
ov_set_extract(OVERRIDE_TYPE, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

Description:

Set the value of the indicated field to the extracted range of bits in the provided value, taking the provided flags into account.

The data is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start(OV_LENGTH)
ov_set_extract(OV_LENGTH, value, 13, 9)
ov_use()
command[...], indirect_ref
ov_clean()
```



Note

Bits in the range *SRC_MSB*:*SRC_LSB* are extracted from *in_value*, and are placed in the indicated field, i.e. *in_value* will be masked and effectively shifted to the right by *SRC_LSB* bit positions before being used.

Condition Codes: Clobbered (*in_value* a register value) or unchanged (*in_value* a constant).

Instruction Count: Two or more (*in_value* a register value) or zero (*in_value* a constant).

Table 2.277. ov_set_extract parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , ORd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.11 ov_set_bm_and_extract

Prototype:

```
ov_set_bm_and_extract(in_byte_mask, OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

Description:

Set the OV_BYTE_MASK field to the provided value and also set the specific range of bits in the indicated field to the extracted range of bits in the provided value, taking the provided flags into account.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set_extract(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

The data for the indicated field is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER |
OV_SIGNAL_MASTER | OV_MASTER_ISLAND))
ov_set(OV_DATA_MASTER, 3, 2, 0)
ov_set_bm_and_extract(byte_mask, OV_DATA_MASTER, 1, 0, data_master, 5, 4)
ov_set(OV_DATA_REF, data_ref)
ov_set(OV_SIGNAL_MASTER, signal_master)
ov_set(OV_MASTER_ISLAND, mater_island)
ov_set(OV_LENGTH, len, OV_SUBTRACT_ONE)
ov_use()
command[...], indirect_ref
ov_clean()
```



Note

Bits in the range *SRC_MSB*:*SRC_LSB* are extracted from *in_value*, and are placed in the indicated field relative to *DST_LSB* and , i.e. *in_value* will be masked and effectively shifted to the left by (*DST_LSB* - *SRC_LSB*) bit positions before being used.

Condition Codes: Clobbered (*in_value* a register value) or unchanged (*in_value* a constant).

Instruction Count: Two or more (*in_value* a register value) or zero (*in_value* a constant).

Table 2.278. ov_set_bm_and_extract parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field OV_BYTE_MASK.
<i>OVERRIDE_TYPE</i>	One override field from <i>Override</i> fields that was specified in <i>ov_start()</i> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override</i> flags, OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.12 ov_set_bm_and_extract

Prototype:

```
ov_set_bm_and_extract(in_byte_mask, OVERRIDE_TYPE, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

Description:

Set the OV_BYTE_MASK field to the provided value and also set the value of the indicated field to the extracted range of bits in the provided value, taking the provided flags into account.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set_extract(OVERRIDE_TYPE, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

The data for the indicated field is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER
         | OV_SIGNAL_MASTER | OV_MASTER_ISLAND))
ov_set_bm_and_extract(byte_mask, OV_DATA_MASTER, data_master, 7, 4)
ov_set(OV_DATA_REF, data_ref)
ov_set(OV_SIGNAL_MASTER, signal_master)
ov_set(OV_MASTER_ISLAND, mater_island)
ov_set(OV_LENGTH, len, OVF_SUBTRACT_ONE)
ov_use()
command[...], indirect_ref
ov_clean()
```




Note

Bits in the range *SRC_MSB:SRC_LSB* are extracted from *in_value*, and are placed in the indicated field, i.e. *in_value* will be masked and effectively shifted to the right by *SRC_LSB* bit positions before being used.

Condition Codes: Clobbered (*in_value* a register value) or unchanged (*in_value* a constant).

Instruction Count: Two or more (*in_value* a register value) or zero (*in_value* a constant).

Table 2.279. *ov_set_bm_and_extract* parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field OV_BYTE_MASK.
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>in_value</i>	Value to set the bits <i>DST_MSB:DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.13 *ov_set_use*

Prototype:

```
ov_set_use(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, FLAGS)
```

Description:

Set the specific range of bits in the indicated field to the provided value, taking the provided flags into account, and then populates CMD_INDIRECT_REF_0, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, FLAGS)
ov_use()
```

Example:

```
ov_start(OV_LENGTH)
ov_set(OV_LENGTH, 4, 4, 0)
ov_set_use(OV_LENGTH, 3, 0, 1, OVF_RELAXED_ORDER_ON)
command[...], indirect_ref
ov_clean()
```



Note

in_value is relative to *DST_LSB*, i.e. *in_value* will be shifted to the left by *DST_LSB* before being used.

Condition Codes: Clobbered

Instruction Count: One to six

Table 2.280. *ov_set_use* parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override</i> fields that was specified in <i>ov_start()</i> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override</i> flags, OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.14 *ov_set_use*

Prototype:

```
ov_set_use(OVERRIDE_TYPE, in_value, FLAGS)
```

Description:

Set the indicated field to the provided value, taking the provided flags into account, and then populates *CMD_INDIRECT_REF_0*, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OVERRIDE_TYPE, in_value, FLAGS)
ov_use()
```

Example:

```
ov_start(OV_LENGTH)
ov_set_use(OV_LENGTH, 12, OV_SUBTRACT_ONE)
command[...], indirect_ref
ov_clean()
```

Condition Codes: Clobbered

Instruction Count: One to six

Table 2.281. `ov_set_use` parameters

Name	Description
<i>OVERWRITE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <code>ov_start()</code> .
<i>in_value</i>	Value to set field <i>OVERWRITE_TYPE</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.15 `ov_set_bm_and_use`

Prototype:

```
ov_set_bm_and_use(in_byte_mask, OVERWRITE_TYPE, DST_MSB, DST_LSB, in_value, FLAGS)
```

Description:

Set the `OV_BYTE_MASK` field to the provided value and also set the specific range of bits in the indicated field to the provided value, taking the provided flags into account, and then populates `CMD_INDIRECT_REF_0`, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set(OVERWRITE_TYPE, DST_MSB, DST_LSB, in_value, FLAGS)
ov_use()
```

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER))
ov_set(OV_DATA_REF, data_ref)
ov_set(OV_LENGTH, len, OVF_SUBTRACT_ONE)
ov_set(OV_DATA_MASTER, 3, 2, 0)
ov_set_bm_and_use(byte_mask, OV_DATA_MASTER, 1, 0, data_master, OVF_PARANOID_ON)
command[...], indirect_ref
ov_clean()
```



Note

The parameter *in_byte_mask* is used in its entirety as is. No provided parameters are applicable to it.

in_value is relative to *DST_LSB*, i.e. *in_value* will be shifted to the left by *DST_LSB* before being used.

Condition Codes: Clobbered

Instruction Count: One to six

Table 2.282. `ov_set_bm_and_use` parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field <code>OV_BYTE_MASK</code> .
<i>OVERRIDE_TYPE</i>	One override field from <code>Override fields</code> that was specified in <code>ov_start()</code> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> in <i>OVERRIDE_TYPE</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <code>Override flags</code> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.16 `ov_set_bm_and_use`

Prototype:

```
ov_set_bm_and_use(in_byte_mask, OVERRIDE_TYPE, in_value, FLAGS)
```

Description:

Set the `OV_BYTE_MASK` field to the provided value and also set the indicated field to the provided value, taking the provided flags into account, and then populates `CMD_INDIRECT_REF_0`, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set(OVERRIDE_TYPE, in_value, FLAGS)
ov_use()
```

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER))
ov_set(OV_LENGTH, len, OV_SUBTRACT_ONE)
ov_set(OV_DATA_REF, data_ref)
ov_set_bm_and_use(byte_mask, OV_DATA_MASTER, data_master, OVF_PARANOID_ON)
command[...], indirect_ref
ov_clean()
```



Note

The parameter *in_byte_mask* is used in its entirety as is. No provided parameters are applicable to it.

Condition Codes: Clobbered

Instruction Count: Two to six, or more, depending on flags

Table 2.283. *ov_set_bm_and_use* parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field OV_BYTE_MASK.
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>in_value</i>	Value to set the field <i>OVERRIDE_TYPE</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.17 *ov_set_extract_use*

Prototype:

```
ov_set_extract_use(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

Description:

Set the specific range of bits in the indicated field to the extracted range of bits in the provided value, taking the provided flags into account, and then populates CMD_INDIRECT_REF_0, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set_extract(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value,
    SRC_MSB, SRC_LSB, FLAGS)
ov_use()
```

The data is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start(OV_LENGTH)
ov_set(OV_LENGTH, 4, 4, 0)
ov_set_extract_use(OV_LENGTH, 3, 0, value, 12, 9, OV_RELAXED_ORDER_ON)
```

```
command[...], indirect_ref
ov_clean()
```



Note

Bits in the range *SRC_MSB*:*SRC_LSB* are extracted from *in_value*, and are placed in the indicated field relative to *DST_LSB* and , i.e. *in_value* will be masked and effectively shifted to the left by (*DST_LSB* - *SRC_LSB*) bit positions before being used.

Condition Codes: Clobbered

Instruction Count: Two to six, or more, depending on flags

Table 2.284. ov_set_extract_use parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.18 ov_set_extract_use

Prototype:

```
ov_set_extract_use(OVERRIDE_TYPE, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

Description:

Set the value of the indicated field to the extracted range of bits in the provided value, taking the provided flags into account, and then populates *CMD_INDIRECT_REF_0*, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set_extract(OVERRIDE_TYPE, in_value, SRC_MSB, SRC_LSB, FLAGS)
ov_use()
```

The data is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start(OV_LENGTH)
ov_set_extract_use(OV_LENGTH, value, 13, 9)
command[...], indirect_ref
ov_clean()
```



Note

Bits in the range *SRC_MSB*:*SRC_LSB* are extracted from *in_value*, and are placed in the indicated field, i.e. *in_value* will be masked and effectively shifted to the right by *SRC_LSB* bit positions before being used.

Condition Codes: Clobbered

Instruction Count: Two to six, or more, depending on flags

Table 2.285. ov_set_extract_use parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override</i> fields that was specified in <i>ov_start()</i> .
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override</i> flags, OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.19 ov_set_bm_and_extract_use

Prototype:

```
ov_set_bm_and_extract_use(in_byte_mask, OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, SRC_MSB, SRC_LSB, FLAGS)
```

Description:

Set the *OV_BYTE_MASK* field to the provided value and also set the specific range of bits in the indicated field to the extracted range of bits in the provided value, taking the provided flags into account, and then populates *CMD_INDIRECT_REF_0*, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set_extract(OVERRIDE_TYPE, DST_MSB, DST_LSB, in_value, SRC_MSB,
              SRC_LSB, FLAGS)
ov_use()
```

The data for the indicated field is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER))
ov_set(OV_DATA_MASTER, 3, 2, 0)
ov_set(OV_DATA_REF, data_ref)
ov_set(OV_LENGTH, len, OV_SUBTRACT_ONE)
ov_set_bm_and_extract_use(byte_mask, OV_DATA_MASTER, 1, 0, data_master, 5, 4)
command[...], indirect_ref
ov_clean()
```



Note

Bits in the range *SRC_MSB*:*SRC_LSB* are extracted from *in_value*, and are placed in the indicated field relative to *DST_LSB* and , i.e. *in_value* will be masked and effectively shifted to the left by (*DST_LSB* - *SRC_LSB*) bit positions before being used.

Condition Codes: Clobbered

Instruction Count: Two to six, or more, depending on flags

Table 2.286. ov_set_bm_and_extract_use parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field OV_BYTE_MASK.
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>DST_MSB</i>	Most significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>DST_LSB</i>	Least significant bit within field <i>OVERRIDE_TYPE</i> to set.
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.20 ov_set_bm_and_extract_use

Prototype:

```
ov_set_bm_and_extract_use(in_byte_mask, OVERRIDE_TYPE, in_value, SRC_MSB, SRC_LSB, FLAGS)
```


Description:

Set the OV_BYTE_MASK field to the provided value and also set the value of the indicated field to the extracted range of bits in the provided value, taking the provided flags into account, and then populates CMD_INDIRECT_REF_0, if appropriate, and sets the value of previous ALU output.



Note

This macro is functionally equivalent to the following, except that it produces more efficient code:

```
ov_set(OV_BYTE_MASK, in_byte_mask)
ov_set_extract(OVERRIDE_TYPE, in_value, SRC_MSB, SRC_LSB, FLAGS)
ov_use()
```

The data for the indicated field is extracted from *in_value*, meaning that bits set in *in_value* outside the range *SRC_MSB*:*SRC_LSB* have no effect on the indirect reference.

Example:

```
ov_start((OV_LENGTH | OV_BYTE_MASK | OV_DATA_REF | OV_DATA_MASTER))
ov_set(OV_DATA_REF, data_ref)
ov_set(OV_LENGTH, len, OVF_SUBTRACT_ONE)
ov_set_bm_and_extract_use(byte_mask, OV_DATA_MASTER, data_master, 7, 4)
command[...], indirect_ref
ov_clean()
```



Note

Bits in the range *SRC_MSB*:*SRC_LSB* are extracted from *in_value*, and are placed in the indicated field, i.e. *in_value* will be masked and effectively shifted to the right by *SRC_LSB* bit positions before being used.

Condition Codes: Clobbered

Instruction Count: Two to six, or more, depending on flags

Table 2.287. ov_set_bm_and_extract_use parameters

Name	Description
<i>in_byte_mask</i>	Value to be used for override field OV_BYTE_MASK.
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> that was specified in <i>ov_start()</i> .
<i>in_value</i>	Value to set the bits <i>DST_MSB</i> : <i>DST_LSB</i> to. May be a constant or a register value.
<i>SRC_MSB</i>	Most significant bit within field <i>in_value</i> to use.
<i>SRC_LSB</i>	Least significant bit within field <i>in_value</i> to use.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.21 ov_use

Prototype:

```
ov_use(FLAGS)
```

Description:

Use the current indirect reference, in accordance with the provided flags.

This macro populates CMD_INDIRECT_REF_0, if appropriate, and then sets the value of previous ALU output.



Note

The next instruction after calling this macro must be the instruction that requires the indirect reference.

Example:

```
ov_start(OV_LENGTH, OV_REUSE_ON)
ov_set(OV_LENGTH, 12, OV_SUBTRACT_ONE)
ov_use()
command[...], indirect_ref
ov_use(OVF_NO_SWAP_SINCE_USE)
command[...], indirect_ref
ov_clean()
```

Condition Codes: Clobbered

Instruction Count: One to six

Table 2.288. ov_use parameters

Name	Description
<i>FLAGS</i>	Optional: The only flag that may be specified is OVF_NO_SWAP_SINCE_USE. This flag may only be specified on the second or subsequent use of the current indirect reference.

2.22.2.22 ov_clean

Prototype:

```
ov_clean()
```

Description:

Release resources used by the current indirect reference.

After calling this macro, the indirect reference no longer exists.

Example:

```
ov_start(OV_LENGTH)
ov_set(OV_LENGTH, 12, OVF_SUBTRACT_ONE)
ov_use()
command[...], indirect_ref
ov_clean()
```

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

2.22.2.23 ov_single

Prototype:

```
ov_single(OVERRIDE_TYPE, in_value, FLAGS)
```

Description:

Shorthand for an indirect reference containing a single field.

The following shorthand:

```
ov_single(OV_LENGTH, 12, OVF_SUBTRACT_ONE)
command[...], indirect_ref
```

is equivalent to

```
ov_start(OV_LENGTH)
ov_set_use(OV_LENGTH, 12, OVF_SUBTRACT_ONE)
ov_clean()
command[...], indirect_ref
```

Table 2.289. ov_single parameters

Name	Description
<i>OVERRIDE_TYPE</i>	One override field from <i>Override fields</i> .
<i>in_value</i>	Value to set field <i>OVERRIDE_TYPE</i> to. May be a constant or a register value.
<i>FLAGS</i>	Optional: Zero or more flags from <i>Override flags</i> , OR'd together, which, for just this macro call, overrides the indirect reference's flags.

2.22.2.24 ov_store

Prototype:

```
ov_store()
```

Description:

Store the current indirect reference for later use.

After calling this macro, the indirect reference no longer exists.



Note

After calling this macro, the number of the slot in which the indirect reference has been stored is available in the preprocessor define `OV_SLOT`. This must be stored to be able to recall / resume the indirect reference later.

After calling this macro, a new indirect reference may be started using `ov_start()`, or a previously stored macro may be resumed using `ov_resume()` or recalled using `ov_recall()`.

Example:

```
ov_start(OV_LENGTH)
ov_set(OV_LENGTH, 12, OV_SUBTRACT_ONE)
ov_store()
#define_eval IND_REF_ONE OV_SLOT
// ...
ov_resume(IND_REF_ONE)
ov_use()
command[...], indirect_ref
ov_clean()
```

Condition Codes: Unchanged

Instruction Count: Zero to two, depending on whether registers have been implicitly copied or not

See Also:

- Resuming and recalling

2.22.2.25 ov_recall

Prototype:

```
ov_recall(SLOT)
```

Description:

Recall a previously-stored indirect reference for use now.

Recalling makes the previously-stored indirect reference the currently active indirect reference, **but without influencing the stored indirect reference**.



Note

There may be more than one `ov_recall()` corresponding to `ov_store()` for a specific slot.

When the stored indirect reference is no longer required, `ov_destroy()` must be called.

Example:

```

ov_start((OV_LENGTH | OV_SEQ_NUM))
ov_set(OV_LENGTH, 16, OV_SUBTRACT_ONE)
ov_store()
#define_eval REF_LENGTH_SEQ_NUM OV_SLOT

#define COUNT 0
#while (COUNT < 4)

    ov_recall(REF_LENGTH_SEQ_NUM)
    ov_set_use(OV_SEQ_NUM, COUNT)
    command[...], indirect_ref
    ov_clean()

#define_eval COUNT (COUNT + 1)
#endloop

ov_dispose(REF_LENGTH_SEQ_NUM)

```

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

Table 2.290. ov_recall parameters

Name	Description
<i>SLOT</i>	The slot number that was returned in OV_SLOT by ov_store().

See Also:

- Resuming and recalling

2.22.2.26 ov_resume

Prototype:

```
ov_resume(SLOT)
```

Description:

Resume a previously-stored indirect reference.

Resuming carries on with the previously-stored indirect reference, as though nothing had occurred in the interim.



Note

There may be exactly one ov_resume() corresponding to ov_store() for a specific slot.

ov_resume() implicitly destroys the stored indirect reference.

Example:

```

// First indirect reference starts
ov_start((OV_BYTE_MASK | OV_LENGTH))

```

```

ov_set(OV_BYTE_MASK, byte_mask)
ov_store()
#define_eval REF_BYTE_MASK_AND_LENGTH OV_SLOT // Store slot number for later

// Second indirect reference starts
ov_start((OV_DATA_REF | OV_BYTE_MASK))
ov_set(OV_BYTE_MASK, byte_mask)
ov_store()
#define_eval REF_DATA_REF_AND_BYTE_MASK OV_SLOT // Store slot number for later

// First indirect reference resumed
ov_resume(REF_BYTE_MASK_AND_LENGTH)
ov_set_use(OV_LENGTH, len, OVF_SUBTRACT_ONE)
command[...], indirect_ref
ov_clean()

// Second indirect reference resumed
ov_resume(REF_DATA_REF_AND_BYTE_MASK)
ov_set_use(OV_DATA_REF, data)
command[...], indirect_ref
ov_clean()

```

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

Table 2.291. ov_resume parameters

Name	Description
<i>SLOT</i>	The slot number that was returned in OV_SLOT by ov_store().

See Also:

- Resuming and recalling

2.22.2.27 ov_dispose

Prototype:

```
ov_dispose(SLOT)
```

Description:

Destroy the indirect reference previously stored at the indicated slot.

After a previously-stored indirect reference has been destroyed, it can no longer be referred to.



Note

If an indirect reference is recalled using `ov_recall()`, the only way to release the resource is via `ov_dipose()`.

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

Table 2.292. ov_dispose parameters

Name	Description
<i>SLOT</i>	The slot number that was returned in OV_SLOT by <code>ov_store()</code> .

See Also:

- Resuming and recalling

2.22.2.28 ov_sanity

Prototype:

`ov_sanity()`

Description:

Performs sanity checks to ensure correct usage of the indirect reference macros.

Checks that:

- The most recent indirect reference must have been completed with `ov_clean()`.
- Every indirect reference that was stored using `ov_store()` must have been cleared via either `ov_resume()` or `ov_destroy()`.



Note

It is recommended that `ov_sanity` is called at the end of the top-level microcode file. It may be called at any convenient time when there should be no current indirect reference and there should be no stored indirect references.

Condition Codes: Unchanged

Instruction Count: None - only preprocessor

2.23 Ring Utility

2.23.1 Ring Utility Macros

List of macros specific to memory ring utilities

2.23.2.1 ru_emem_ring_setup

Prototype:

```
ru_emem_ring_setup(_in_addr_hi, _in_q_desc_addr_lo, _in_ring_base_addr_lo, _in_ring_num,
_IN_SIZE_LW, _IN_Q_LOC)
```

Description:

Set up a type 2 ring with variable q-descriptor and ring bases addresses.

Table 2.293. ru_emem_ring_setup parameters

Name	Description
<i>_in_addr_hi</i>	Upper byte of the 40-bit address of the ring descriptor in emem0, emem1 or emem2 Note that it is located in the lower byte of <i>_in_addr_hi</i>
<i>_in_q_desc_addr_lo</i>	32-bit address identifying the location of the descriptor
<i>_in_ring_base_addr_lo</i>	32-bit address identifying the location of the rings in emem0, emem1 or emem2
<i>_in_ring_num</i>	Number of emem rings to configure, valid values 0-1023, GPR or constant value
<i>_IN_SIZE_LW</i>	size of ring in long words, must be between 512 and 16M and a power of 2
<i>_IN_Q_LOC</i>	Queue locality mode: <ul style="list-style-type: none"> • MU_LOCALITY_HIGH • MU_LOCALITY_LOW • MU_LOCALITY_DIRECT_ACCESS • MU_LOCALITY_DISCARD_AFTER_READ This macro is also available with fixed q-descriptor and ring addresses (5 parameters)

2.23.2.2 ru_emem_ring_setup

Prototype:

```
ru_emem_ring_setup(_IN_Q_DESC_ADDR, _IN_RING_BASE_ADDR, _in_ring_num, _IN_SIZE_LW,
_IN_Q_LOC)
```

Description:

Set up a type 2 ring with constant q-descriptor and ring bases addresses.

This macro is also available with dynamic q-descriptor and ring addresses (6 parameters)

Table 2.294. ru_emem_ring_setup parameters

Name	Description
<i>_IN_Q_DESC_ADDR</i>	40-bit address identifying the location of the ring descriptor in emem0, emem1 or emem2

Name	Description
<code>_IN_RING_BASE_ADDR</code>	40-bit address identifying the location of the rings in emem0, emem1 or emem2
<code>_in_ring_num</code>	Number of emem rings to configure, valid values 0-1023, GPR or constant value
<code>_IN_SIZE_LW</code>	size of ring in long words, must be between 512 and 16M and a power of 2
<code>_IN_Q_LOC</code>	<p>Queue locality mode:</p> <ul style="list-style-type: none"> • MU_LOCALITY_HIGH • MU_LOCALITY_LOW • MU_LOCALITY_DIRECT_ACCESS • MU_LOCALITY_DISCARD_AFTER_READ <p>Example: Create ring in emem island 26</p> <pre> #define EMEM_RING_NUM 12 #define EMEM_RING_SIZE 512 .alloc_mem RING_BASE_ADDR i26.emem island EMEM_RING_SIZE EMEM_RING_SIZE .alloc_mem Q_DESC_BASE_ADDR i26.emem island 16 .reg \$xdout[2], \$xdin[2] .xfer_order \$xdout, \$xdin .sig g1, g2 ru_emem_ring_setup(Q_DESC_BASE_ADDR, RING_BASE_ADDR, EMEM_RING_NUM, EMEM_RING_SIZE, ru_emem_ring_put(RING_BASE_ADDR, \$xdout[0], EMEM_RING_NUM, 2, g1] ru_emem_ring_get(RING_BASE_ADDR, \$xdin[0], EMEM_RING_NUM, 2, g2]</pre>

2.23.2.3 ru_emem_ring_put

Prototype:

```
ru_emem_ring_put(_in_ring_base_addr, _in_xfer_reg, _in_ring_num, _IN_REF_CNT, _in_sig_name)
```

Description:

Add entries to the tail of the circular buffer in emem based on parameters passed to the macro.

See `ru_emem_ring_setup()` macro for implementation details Some compile time error checking is done.

Table 2.295. ru_emem_ring_put parameters

Name	Description
<code>_in_ring_base_addr</code>	40-bit address identifying the location of the rings in emem0, emem1 or emem2
<code>_in_xfer_reg</code>	Entries to add to the circular buffer
<code>_in_ring_num</code>	Select the ring number to add to, between 0-1023
<code>_IN_REF_CNT</code>	Number of 32-bit words to add to the circular buffer, must be between 1-16
<code>_in_sig_name</code>	Signal to use for ring operation

2.23.2.4 ru_emem_ring_get

Prototype:

```
ru_emem_ring_get(_in_ring_base_addr, _out_xfer_reg, _in_ring_num, _IN_REF_CNT,
_in_sig_name)
```

Description:

Remove entries from the head of the circular buffer in emem based on parameters passed to the macro.

See ru_emem_ring_setup() macro for implementation details Some compile time error checking is done.

Table 2.296. ru_emem_ring_get parameters

Name	Description
<i>_in_ring_base_addr</i>	40-bit address identifying the location of the rings in emem0, emem1 or emem2
<i>_out_xfer_reg</i>	Entries to remove from the circular buffer
<i>_in_ring_num</i>	Select the ring number to remove from, between 0-1023
<i>_IN_REF_CNT</i>	Number of 32-bit words to remove from the circular buffer must between 1-16
<i>_in_sig_name</i>	Signal to use for ring operation

2.23.2.5 ru_nn_ring_init

Prototype:

```
ru_nn_ring_init(NN_EMPTY_THRESHOLD)
```

Description:

Next-neighbor ring initialization.

Table 2.297. ru_nn_ring_init parameters

Name	Description
<i>NN_EMPTY_THRESHOLD</i>	Threshold when NN_Empty asserts. Valid values are 0-3.

2.23.2.6 ru_dram_ring_setup

Prototype:

```
ru_dram_ring_setup(_IN_RING_NUM_, _IN_BASE_ADDR_, _IN_SIZE_LW_, _IN_Q_LOC_, _IN_Q_PAGE_)
```

Description:

Set up a type 2 DRAM ring based on parameters passed to the macro.

Some compile time error checking is done.

Table 2.298. ru_dram_ring_setup parameters

Name	Description
<code>_IN_RING_NUM_</code>	Number of DRAM ring to configure
<code>_IN_BASE_ADDR_</code>	DRAM address where ring starts
<code>_IN_SIZE_LW_</code>	LW size of ring, must be between 512 and 16M and a power of 2
<code>_IN_Q_LOC_</code>	Queue locality mode: <ul style="list-style-type: none"> • <code>MU_LOCALITY_HIGH</code> • <code>MU_LOCALITY_LOW</code> • <code>MU_LOCALITY_DIRECT_ACCESS</code> • <code>MU_LOCALITY_DISCARD_AFTER_READ</code>
<code>_IN_Q_PAGE_</code>	Top two bits of the queue entry addresses.

2.23.2.7 ru_dram_ring_setup

Prototype:

```
ru_dram_ring_setup(_IN_RING_NUM_, _IN_BASE_ADDR_, _IN_SIZE_LW_)
```

Description:

Set up a type 2 DRAM ring based on parameters passed to the macro.

This is an overloaded macro which uses `MU_LOCALITY_HIGH` for `_IN_Q_LOC_` and 0 for `_IN_Q_PAGE_`.

Table 2.299. ru_dram_ring_setup parameters

Name	Description
<code>_IN_RING_NUM_</code>	Number of DRAM ring to configure
<code>_IN_BASE_ADDR_</code>	DRAM address where ring starts
<code>_IN_SIZE_LW_</code>	LW size of ring, must be between 512 and 16M and a power of 2

2.23.2.8 ru_sram_ring_setup

Prototype:

```
ru_sram_ring_setup(_IN_RING_NUM_, _IN_BASE_ADDR_, _IN_SIZE_LW_)
```

Description:

Set up a SRAM ring Based on parameters passed into the macro.

Some compile time error checking is done.

Table 2.300. ru_sram_ring_setup parameters

Name	Description
<code>_IN_RING_NUM_</code>	Number of SRAM ring to configure
<code>_IN_BASE_ADDR_</code>	SRAM address where ring starts (channel value will be extracted)
<code>_IN_SIZE_LW_</code>	LW size of ring, must be between 512 and 64K and a power of 2

2.23.2.9 ru_cls_ring_setup

Prototype:

```
ru_cls_ring_setup(_IN_RING_NUM_, _IN_BASE_ADDR_, _IN_SIZE_LW_)
```

Description:

Set up a single CLS ring based on parameters passed into the macro.

Some compile time error checking is done.

Table 2.301. ru_cls_ring_setup parameters

Name	Description
<code>_IN_RING_NUM_</code>	Number of CLS ring to configure
<code>_IN_BASE_ADDR_</code>	CLS address where ring starts
<code>_IN_SIZE_LW_</code>	LW size of ring, must be between 32 and 1024 and a power of 2

2.23.2.10 ru_ctm_ring_setup

Prototype:

```
ru_ctm_ring_setup(_IN_RING_NUM_, _IN_BASE_ADDR_, _IN_SIZE_LW_, _IN_STATUS_)
```

Description:

Set up a single CTM ring based on parameters passed into the macro.

Some compile time error checking is done.

Table 2.302. ru_ctm_ring_setup parameters

Name	Description
<code>_IN_RING_NUM_</code>	CONST, Number of CTM ring to configure, must be between 0 and 14
<code>_IN_BASE_ADDR_</code>	CONST, CTM address where ring starts, must be aligned to ring size
<code>_IN_SIZE_LW_</code>	CONST, LW size of ring, must be between 128 and 16*1024 and a power of 2
<code>_IN_STATUS_</code>	CONST, status generation control, either "FULL" or "EMPTY"

2.23.2.11 ru_ring_op

Prototype:

```
ru_ring_op(__MEM_TYPE__, _IN_CMD_, in_xfer_reg, _IN_RING_NUM_, _IN_REF_CNT_, in_sig)
```

Description:

Wrapper for CLS/GS/CTM ring commands.

This macro is used to hide some of the internal details such as encoding the ring address.



Note

No swapping on the signal is done, the calling code needs to do this.

Table 2.303. ru_ring_op parameters

Name	Description
<code>__MEM_TYPE__</code>	One of CLS,GS, or CTM. GS not supported for NFP6000. CTM only supported for NFP6000
<code>_IN_CMD_</code>	Ring command to perform (put/get for CLS,GS or ring_put/get for CTM)
<code>in_xfer_reg</code>	Xfer register name to use in command
<code>_IN_RING_NUM_</code>	Ring number where data is to be placed, must be between 0-15
<code>_IN_REF_CNT_</code>	Reference count, must be between 1-16
<code>in_sig</code>	Signal to use for ring operation

2.23.2.12 ru_sram_ring_put

Prototype:

```
ru_sram_ring_put(in_xfer_reg, in_src_op1, in_src_op2, _RING_CHAN_NUM_, _IN_REF_CNT_,  
in_sig_name, sig_action)
```

Description:

Put `_IN_REF_CNT_` words on sram ring.



Note

- If there are no sufficient words in the ring for get and put commands, two signals will be pushed where `sig_name[1]` signals error.

Table 2.304. ru_sram_ring_put parameters

Name	Description
<i>in_xfer_reg</i>	Xfer register name to use in command
<i>in_src_op1</i>	Restricted operands are added (src_op1 + src_op2) to define the following: <ul style="list-style-type: none"> • [31:30]: SRAM channel. • [29:8]: Ignored. • [7:2]: Ring number. • [1:0]: Ignored.
<i>in_src_op2</i>	As per above
<i>_RING_CHAN_NUM_</i>	SRAM channel/bank to use
<i>_IN_REF_CNT_</i>	Reference count in increments of 4 byte words. Valid values are 1 to 8.
<i>in_sig_name</i>	Signal to use for ring operation
<i>sig_action</i>	SIG_NONE or SIG_WAIT

2.23.2.13 ru_sram_ring_get

Prototype:

```
ru_sram_ring_get(out_xfer_reg, in_src_op1, in_src_op2, _RING_CHAN_NUM_, IN_REF_CNT_,
in_sig_name)
```

Description:

Refer to description for ru_sram_ring_put macro.

2.23.2.14 ru_sram_ring_get

Prototype:

```
ru_sram_ring_get(out_xfer_reg, in_src_op1, in_src_op2, _RING_CHAN_NUM_, _IN_REF_CNT_,
in_sig_name, sig_action)
```

Description:

Refer to description for ru_sram_ring_put macro.

2.23.2.15 ru_sram_ring_journal

Prototype:

```
ru_sram_ring_journal(in_xfer_reg, in_src_op1, in_src_op2, _RING_CHAN_NUM_, _IN_REF_CNT_,
in_sig_name, sig_action)
```

Description:

Refer to description for ru_sram_ring_put macro.

2.23.2.16 ru_dram_ring_put

Prototype:

```
ru_dram_ring_put(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,
sig_action)
```

Description:

Put _IN_REF_CNT_ words on dram ring.



Note

- If there are not sufficient words in the ring for get, pop, and put commands, two signals will be pushed where sig_name[1] signals error.
- If the EOP bit is set for get_eop and pop_eop commands, two signals will be pushed, where sig_name[1] signals error.
- If the tag is not matched for get_safe_tag, pop_tag_safe, and journal_tag, two signals will be pushed, where sig_name[1] signals error.

Table 2.305. ru_dram_ring_put parameters

Name	Description
<i>in_xfer_reg</i>	Xfer register name to use in command
<i>in_src_op1</i>	Restricted operands are added (src_op1 + src_op2) to define the following: <ul style="list-style-type: none"> • [23:16]: Memory tag number for XX_tag_XX commands. • [9:0]: Queue array entry number.
<i>in_src_op2</i>	As per above
<i>_IN_REF_CNT_</i>	Reference count in increments of 4 byte words. Valid values are 1 to 16 for NFP6000 or 1 to 8 otherwise. Specified as actual count - 1.
<i>in_sig_name</i>	Signal to use for ring operation
<i>sig_action</i>	SIG_NONE or SIG_WAIT

2.23.2.17 ru_dram_ring_put_tag

Prototype:

```
ru_dram_ring_put_tag(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.18 ru_dram_ring_qadd_work

Prototype:

```
ru_dram_ring_qadd_work(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.19 ru_dram_ring_qadd_thread

Prototype:

```
ru_dram_ring_qadd_thread(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.20 ru_dram_ring_get

Prototype:

```
ru_dram_ring_get(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.21 ru_dram_ring_get

Prototype:

```
ru_dram_ring_get(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.22 ru_dram_ring_get_eop

Prototype:

```
ru_dram_ring_get_eop(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.23 ru_dram_ring_get_safe

Prototype:

```
ru_dram_ring_get_safe(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.24 ru_dram_ring_get_tag_safe

Prototype:

```
ru_dram_ring_get_tag_safe(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.25 ru_dram_ring_pop

Prototype:

```
ru_dram_ring_pop(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.26 ru_dram_ring_pop

Prototype:

```
ru_dram_ring_pop(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.27 ru_dram_ring_pop_eop

Prototype:

```
ru_dram_ring_pop_eop(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.28 ru_dram_ring_pop_safe

Prototype:

```
ru_dram_ring_pop_safe(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.29 ru_dram_ring_pop_tag_safe

Prototype:

```
ru_dram_ring_pop_tag_safe(out_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for ru_dram_ring_put macro.

2.23.2.30 ru_dram_ring_journal

Prototype:

```
ru_dram_ring_journal(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Refer to description for `ru_dram_ring_put` macro.

2.23.2.31 `ru_dram_ring_journal_tag`

Prototype:

```
ru_dram_ring_journal_tag(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,
sig_action)
```

Description:

Refer to description for `ru_dram_ring_put` macro.

2.23.2.32 `ru_deq_from_ring`

Prototype:

```
ru_deq_from_ring(out_req, RING_TYPE, ref_cnt, ring_num, sig, sig_action, __NULL_LABEL__)
```

Description:

Generic ring dequeue operation.

Table 2.306. `ru_deq_from_ring` parameters

Name	Description
<i>out_req</i>	Register aggregate
<i>RING_TYPE</i>	Ring type, One of NN_RING,CLS_RING,GS_RING,DDR_RING,QDR_RING
<i>ref_cnt</i>	Number of operations to perform
<i>ring_num</i>	Ring number
<i>sig</i>	Signal to generate
<i>sig_action</i>	If SIG_WAIT, waits for operation to complete
<i>__NULL_LABEL__</i>	label to branch to on NN_RING empty, or -- to wait for not empty

2.23.2.33 `ru_deq_from_ring`

Prototype:

```
ru_deq_from_ring(out_req, RING_TYPE, ref_cnt, ring_num, ring_chan, sig, sig_action,
__NULL_LABEL__)
```

Description:

Generic ring dequeue operation.

Table 2.307. ru_deq_from_ring parameters

Name	Description
<i>out_req</i>	Register aggregate
<i>RING_TYPE</i>	Ring type, One of NN_RING,CLS_RING,GS_RING,DDR_RING,QDR_RING
<i>ref_cnt</i>	Number of operations to perform
<i>ring_num</i>	Ring number
<i>ring_chan</i>	Ring channel, only applicable to the QDR_RING
<i>sig</i>	Signal to generate
<i>sig_action</i>	If SIG_WAIT, waits for operation to complete
<i>__NULL_LABEL__</i>	label to branch to on NN_RING empty, or -- to wait for not empty

2.23.2.34 ru_cls_ring_put

Prototype:

```
ru_cls_ring_put(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name, sig_action)
```

Description:

Add n 32-bit words to the tail of the ring.

Table 2.308. ru_cls_ring_put parameters

Name	Description
<i>in_xfer_reg</i>	xfer registers
<i>in_src_op1</i>	Ring number (contant/GPR)
<i>in_src_op2</i>	Not used and ignored. Can be "--".
<i>_IN_REF_CNT_</i>	Number of 32-bit words to put on to ring
<i>in_sig_name</i>	Signal to wait on
<i>sig_action</i>	SIG_WAIT or SIG_NONE

2.23.2.35 ru_cls_ring_pop

Prototype:

```
ru_cls_ring_pop(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name, sig_action)
```

Description:

Pop n 32-bit words from tail of the ring (LIFO).

2.23.2.36 ru_cls_ring_pop_safe

Prototype:

```
ru_cls_ring_pop_safe(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Pop n 32-bit words from tail of the ring (LIFO).

If less than n in the ring, return zero for extra words.

2.23.2.37 ru_cls_ring_get_safe

Prototype:

```
ru_cls_ring_get_safe(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name,  
sig_action)
```

Description:

Get n 32-bit words from head of the ring (FIFO).

If less than n in the ring, return zero for extra words.

2.23.2.38 ru_cls_ring_get

Prototype:

```
ru_cls_ring_get(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name, sig_action)
```

Description:

Get n 32-bit words from head of the ring (FIFO).

2.23.2.39 ru_ctm_ring_put

Prototype:

```
ru_ctm_ring_put(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name, sig_action)
```

Description:

Add n 32-bit words to the tail of the ring.

Table 2.309. ru_ctm_ring_put parameters

Name	Description
<i>in_xfer_reg</i>	xfer registers
<i>in_src_op1</i>	Ring number (contant/GPR)
<i>in_src_op2</i>	Not used and ignored. Can be "--".
<i>_IN_REF_CNT_</i>	Number of 32-bit words to put on to ring
<i>in_sig_name</i>	Signal to wait on
<i>sig_action</i>	SIG_WAIT or SIG_NONE

2.23.2.40 ru_ctm_ring_get

Prototype:

```
ru_ctm_ring_get(in_xfer_reg, in_src_op1, in_src_op2, _IN_REF_CNT_, in_sig_name, sig_action)
```

Description:

Get n 32-bit words from head of the ring.

Table 2.310. ru_ctm_ring_get parameters

Name	Description
<i>in_xfer_reg</i>	xfer registers
<i>in_src_op1</i>	Ring number (contant/GPR)
<i>in_src_op2</i>	Not used and ignored. Can be "--".
<i>_IN_REF_CNT_</i>	Number of 32-bit words to get from ring
<i>in_sig_name</i>	Signal to wait on
<i>sig_action</i>	SIG_WAIT or SIG_NONE

2.23.2.41 ru_enq_to_ring

Prototype:

```
ru_enq_to_ring(in_req, RING_TYPE, ref_cnt, ring_num, sig, sig_action, __FULL_LABEL__)
```

Description:

Generic ring enqueue operation, with optional ring channel.

Table 2.311. ru_enq_to_ring parameters

Name	Description
<i>in_req</i>	Register aggregate

Name	Description
<i>RING_TYPE</i>	Ring type, One of NN_RING,CLS_RING,GS_RING(not supported for NFP6000),DDR_RING,QDR_RING
<i>ref_cnt</i>	Number of operations to perform
<i>ring_num</i>	Ring number
<i>sig</i>	Signal to generate
<i>sig_action</i>	If SIG_WAIT, waits for operation to complete
<i>__FULL_LABEL__</i>	label to branch to on NN_RING full, or -- to wait for not full

2.23.2.42 ru_enq_to_ring

Prototype:

```
ru_enq_to_ring(in_req, RING_TYPE, ref_cnt, ring_num, ring_chan, sig, sig_action,
__FULL_LABEL__)
```

Description:

Generic ring enqueue operation, with optional ring channel.

Table 2.312. ru_enq_to_ring parameters

Name	Description
<i>in_req</i>	Register aggregate
<i>RING_TYPE</i>	Ring type, One of NN_RING,CLS_RING,GS_RING,DDR_RING,QDR_RING
<i>ref_cnt</i>	Number of operations to perform
<i>ring_num</i>	Ring number
<i>ring_chan</i>	Ring channel, only applicable to the QDR_RING
<i>sig</i>	Signal to generate
<i>sig_action</i>	If SIG_WAIT, waits for operation to complete
<i>__FULL_LABEL__</i>	label to branch to on NN_RING full, or -- to wait for not full

2.24 SRAM Operation

2.24.1 SRAM Operation Macros

Operations macros specific for SRAM memory (deprecated)

2.24.2.1 sram_read

Prototype:

```
sram_read(out_data, in_sram_addr, in_addr_offset, in_lw_count, REQ_SIG, in_wakeup_sigs,
Q_OPTION)
```

Description:

Read from sram starting at address of first longword.

Instruction Count: 1-5 (1 read SRAM access)

Example:

```
sram_read($packet[2], addr, 0, LWCOUNT3, SIG_SRAM. SIG_SRAM, ____)
```

Table 2.313. sram_read parameters

Name	Description
<i>out_data</i>	First sram xfer reg of sequence to read to, array notation must be in xbuf array notation, index range 0-15
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue priority: high priority Temp register usage: 0-2, uses registers if constant addr args > MAX_IMMEDIATE, or register length

2.24.2.2 sram_write

Prototype:

```
sram_write(in_data, in_sram_addr, in_addr_offset, in_lw_count, REQ_SIG, in_wakeup_sigs,
Q_OPTION)
```

Description:

Write to sram.

Instruction Count: 1-5 (1 write SRAM access)

Example:

```
sram_write($packet[2], addr, 0, LWCOUNT3, SIG_SRAM. SIG_SRAM, ____)
```

Table 2.314. sram_write parameters

Name	Description
<i>in_data</i>	First sram xfer register of sequence to write from, array notation must be in xbuf array notation, index range 0-15
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>in_lw_count</i>	Register or constant longword count
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue priority: high priority Temp register usage: 0-2, uses registers if constant addr args > MAX_IMMEDIATE, or register or length

2.24.2.3 sram_bits_clr

Prototype:

```
sram_bits_clr(in_mask, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Clear in_mask bits at sram longword location.

Table 2.315. sram_bits_clr parameters

Name	Description
<i>in_mask</i>	Register or constant, mask of bits to set
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue

Name	Description
	<ul style="list-style-type: none"> priority: high priority

2.24.2.4 sram_bits_set

Prototype:

```
sram_bits_set(in_mask, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Set in_mask bits at sram longword location.

Instruction Count: 1-8 (1 sram read-modify-write memory access)

Example:

```
sram_bits_set(0x111, 0, bit_position, SIG_SRAM. SIG_SRAM, ____)
```

Table 2.316. sram_bits_set parameters

Name	Description
<i>in_mask</i>	Register or constant, mask of bits to set
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue priority: high priority

2.24.2.5 sram_bits_test_and_clr

Prototype:

```
sram_bits_test_and_clr(out_data, in_mask, in_sram_addr, in_addr_offset, REQ_SIG,  
in_wakeup_sigs, Q_OPTION)
```

Description:

Clear in_mask bits at sram longword location.

Read contents of sram address prior to the write.

Instruction Count: 1-8 (1 sram read-modify-write memory access)

Table 2.317. sram_bits_test_and_clr parameters

Name	Description
<i>out_data</i>	Read xfer register result
<i>in_mask</i>	Register or constant, mask of bits to set
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue priority: high priority

2.24.2.6 sram_bits_test_and_set

Prototype:

```
sram_bits_test_and_set(out_data, in_mask, in_sram_addr, in_addr_offset, REQ_SIG,
in_wakeup_sigs, Q_OPTION)
```

Description:

Set in_mask bits at sram longword location.

Read contents of sram address prior to the write.

Instruction Count: 1-8 (1 sram read-modify-write memory access)

Example:

```
sram_bits_test_and_set(prev_value, 0x1000, addr0, addr1, SIG_SRAM. SIG_SRAM, ____ ) // test/set bit 3
```

Table 2.318. sram_bits_test_and_set parameters

Name	Description
<i>out_data</i>	Read xfer register result
<i>in_mask</i>	Register or constant, mask of bits to set
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue

Name	Description
	<ul style="list-style-type: none"> optimize_mem: mem controller selects cycle to issue priority: high priority

2.24.2.7 sram_add

Prototype:

```
sram_add(in_data, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Add in_data to sram location.

Table 2.319. sram_add parameters

Name	Description
<i>in_data</i>	Register or constant, data to add
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue priority: high priority

2.24.2.8 sram_decr

Prototype:

```
sram_decr(in_sram_addr, in_addr_offset)
```

Description:

Decrement 32-bit longword at sram location.

Table 2.320. sram_decr parameters

Name	Description
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset

2.24.2.9 sram_incr

Prototype:

```
sram_incr(in_sram_addr, in_addr_offset)
```

Description:

Increment 32-bit longword at to sram location.

Table 2.321. sram_incr parameters

Name	Description
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset

2.24.2.10 sram_sub

Prototype:

```
sram_sub(in_data, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Sub *in_data* from sram location.



Note

sub instr is not supported in HW, so we subtract from 0 and add that number. *in_data* must not be a write transfer register.

Instruction Count: 2-6 (1 SRAM read-modify-write access)

Table 2.322. sram_sub parameters

Name	Description
<i>in_data</i>	Register or constant, data to subtract
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue

Name	Description
	<ul style="list-style-type: none"> priority: high priority

2.24.2.11 sram_swap

Prototype:

```
sram_swap(out_data, in_data, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs,
Q_OPTION)
```

Description:

Write `in_data` to sram location.

Read contents of sram location prior to the operation to `out_data`.



Note

`out_data` must be a transfer register.

Instruction Count: 1-8 (1 SRAM read-modify-write access)

Example: `code sram_swap(prev_value, new_value, addr0, addr1, SIG_SRAM. SIG_SRAM, __) // test/set bit 3`

Table 2.323. sram_swap parameters

Name	Description
<code>out_data</code>	Transfer Register, returned previous data value
<code>in_data</code>	Register or constant, data to write
<code>in_sram_addr</code>	Register or constant base longword address
<code>in_addr_offset</code>	Register or constant longword address offset
<code>REQ_SIG</code>	Signal associated with this request
<code>in_wakeup_sigs</code>	Signal or signals to wake up on
<code>Q_OPTION</code>	Queue option <ul style="list-style-type: none"> <code>no_option</code> or <code>__</code> : default. Order queue <code>optimize_mem</code>: mem controller selects cycle to issue priority: high priority

2.24.2.12 sram_test_and_add

Prototype:

```
sram_test_and_add(out_data, in_data, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs,
Q_OPTION)
```

Description:

Read contents of sram location to `out_data`.

Then add `in_data` to sram location contents.



Note

`out_data` must be a transfer register.

Example:

```
sram_test_and_add(prev_value, addend, addr0, addr1, SIG_SRAM. SIG_SRAM, __) // test/set bit 3
```

Instruction Count: 1-8 (1 SRAM read-modify-write access)

Table 2.324. sram_test_and_add parameters

Name	Description
<code>out_data</code>	Transfer Register, returned previous data value
<code>in_data</code>	Register or constant, data to add
<code>in_sram_addr</code>	Register or constant base longword address
<code>in_addr_offset</code>	Register or constant longword address offset
<code>REQ_SIG</code>	Signal associated with this request
<code>in_wakeup_sigs</code>	Signal or signals to wake up on
<code>Q_OPTION</code>	Queue option <ul style="list-style-type: none"> <code>no_option</code> or <code>__</code> : default. Order queue <code>optimize_mem</code>: mem controller selects cycle to issue <code>priority</code>: high priority

2.24.2.13 sram_test_and_decr

Prototype:

```
sram_test_and_decr(out_data, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs, Q_OPTION)
```

Description:

Read contents of sram location to `out_data`.

Then decrement sram location contents.



Note

out_data must be a transfer register.

Instruction Count: 1-5 (1 SRAM read-modify-write access)

Example:

```
sram_test_and_decr(prev_value, addr0, addr1, SIG_SRAM. SIG_SRAM, __) // test/set bit 3
```

Table 2.325. sram_test_and_decr parameters

Name	Description
<i>out_data</i>	Transfer Register, returned previous data value
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or __ : default. Order queue optimize_mem: mem controller selects cycle to issue priority: high priority

2.24.2.14 sram_test_and_incr

Prototype:

```
sram_test_and_incr(out_data, in_sram_addr, in_addr_offset, REQ_SIG, in_wakeup_sigs,  
Q_OPTION)
```

Description:

Read contents of sram location to out_data.

Then increment sram location contents.



Note

out_data must be a transfer register.

Instruction Count: 1-5 (1 SRAM read-modify-write access)

Example:

```
sram_test_and_incr(prev_value, addr0, addr1, SIG_SRAM. SIG_SRAM, __) // test/set bit 3
```


Table 2.326. sram_test_and_incr parameters

Name	Description
<i>out_data</i>	Transfer Register, returned previous data value
<i>in_sram_addr</i>	Register or constant base longword address
<i>in_addr_offset</i>	Register or constant longword address offset
<i>REQ_SIG</i>	Signal associated with this request
<i>in_wakeup_sigs</i>	Signal or signals to wake up on
<i>Q_OPTION</i>	Queue option <ul style="list-style-type: none"> no_option or ____ : default. Order queue optimize_mem: mem controller selects cycle to issue priority: high priority

2.24.2.15 sram_fast_journal

Prototype:

```
sram_fast_journal(in_journal_reg_orig, ring_num)
```

Description:

Macro to do sram ring fast journal logging.

Table 2.327. sram_fast_journal parameters

Name	Description
<i>in_journal_reg_orig</i>	Register with data in lower 24 bits, upper 8 bits clear
<i>ring_num</i>	SRAM journal ring number - must be constant

2.24.2.16 sram_wr_qdesc

Prototype:

```
sram_wr_qdesc(_QID_, _DEBUG_QDESC_SRAM_BASE_)
```

Description:

Given freelist ID and sram address, dump the current sram queue descriptor state to specified sram address.

Table 2.328. sram_wr_qdesc parameters

Name	Description
<i>_QID_</i>	Qdescriptor number to dump
<i>_DEBUG_QDESC_SRAM_BASE_</i>	SRAM address where q-descriptor will be written

2.24.2.17 sram_memset

Prototype:

```
sram_memset(in_sram_addr, in_len, lw_pattern, _CHUNK_SIZE_)
```

Description:

Initialize SRAM memory with given pattern for 'len' bytes.

Table 2.329. sram_memset parameters

Name	Description
<i>in_sram_addr</i>	SRAM start address (constant or mutable GPR)
<i>in_len</i>	Number of bytes to initialize (blocks of 32 bytes) (constant or mutable GPR)
<i>lw_pattern</i>	Pattern to fill (constant)
<i>_CHUNK_SIZE_</i>	

2.25 Thread Synchronization

2.25.1 Thread Synchronization Macros

Thread Synchronization Macros

2.25.2.1 threads_reorder_once

Prototype:

```
threads_reorder_once(in_sig)
```

Description:

Reorder threads.

Threads enabled to run in order 0->7

Example:

```
.sig s
threads_reorder_once(s)
[work to do before next thread runs]
ctx_arb[voluntary] // let next thread run
```

Table 2.330. threads_reorder_once parameters

Name	Description
<i>in_sig</i>	signal

2.25.2.2 threads_br_ctx

Prototype:

```
threads_br_ctx(CTX_MASK, LABEL)
```

Description:

Take a branch if running on thread indicated in bit mask.

Example:

```
.reg r_thd_3_0, r_thd_7_4
#define THD_MASK 0xF0
threads_br_ctx(THD_MASK, thd_7_4_running#)
thd_3_0_running#:
immed[r_thd3_0, 1]
br[done#]
thd_7_4_running#:
immed[r_thd7_4, 1]
done#:
```

Table 2.331. threads_br_ctx parameters

Name	Description
<i>CTX_MASK</i>	
<i>LABEL</i>	Constant, branch target label

2.25.2.3 threads_kill

Prototype:

```
threads_kill(KILL_MASK)
```

Description:

Kill threads indicated in bit mask.

Example:

```
#define THD_MASK 0xAA
threads_kill(THD_MASK) // kill threads 7,5,3,1
```

Table 2.332. threads_kill parameters

Name	Description
<i>KILL_MASK</i>	

2.25.2.4 threads_order

Prototype:

```
threads_order(out_next, in_sig, ORDERED_MASK)
```

Description:

Order threads enabled in bit mask.

Example:

```
.sig s
.reg next_val
#define THD_MASK 0xAA
threads_order(next_val, s, THD_MASK) // order threads 1,3,5,7
```

Table 2.333. threads_order parameters

Name	Description
<i>out_next</i>	GPR, written with
<i>in_sig</i>	signal
<i>ORDERED_MASK</i>	Constant, bit mask, one bit per thread of threads to be ordered

2.26 XBUF Operation

2.26.1 XBUF Operation Macros

XBUF macros

2.26.2.1 xbuf_bind_address

Prototype:

```
xbuf_bind_address(str_xbuf_name, POOLS_BASE, POOL_SIZE, BUFFER_OFFSET)
```

Description:

This macro is used to bind a buffer name with an address in a pool of buffers inside local memory.

Common designs have local memory divided into several blocks of pools. In each block, the pools are contiguous, and the number of pools is equal to the number of active threads so that each thread has one pool. Inside a pool, there can be several buffers.

This call is applicable to LMEM buffers only. It is transparent to other types.

Example:

```
xbuf_alloc(lmem_buf0, 8, READ_WRITE)
xbuf_bind_address(lmem_buf0, 0x100, 0x20, 0x0)
xbuf_activate(lmem_buf0, 0, 0, 1)

Start using lmem_buf0...
ipv4_cksum_verify(lmem_buf0, 0xb)
```

Instruction Count: 0



Note

lmem_buf0, lmem_buf1, ..., lmem_buf15 must be used for local memory buffers.

Table 2.334. xbuf_bind_address parameters

Name	Description
<i>str_xbuf_name</i>	Name of buffer. Name has to start with "lmem". For example, lmem_buf1. If name does not start with "lmem", the macro will assume that buffer is not in local memory and nothing will be done.
<i>POOLS_BASE</i>	Base of all pools in bytes (must be multiple of POOL_SIZE)
<i>POOL_SIZE</i>	Size of each pool in bytes (must be power of 2)
<i>BUFFER_OFFSET</i>	Offset in bytes of the buffer inside its pool. The offset should be aligned on a long-word boundary which is calculated as the next higher power of 2 of the buffer-size.

Macro Defines:

- Set internal constants.
- `_xbuf[i]_lmem_pools_base`
- `_xbuf[i]_lmem_pool_size`
- `_xbuf[i]_lmem_offset` (where $i = 0..15$ since 16 buffers are supported)

2.26.2.2 xbuf_deactivate

Prototype:

```
xbuf_deactivate(str_xbuf_name)
```

Description:

Undefine the binding (set by `xbuf_activate`) between the buffer name and its current Local Memory handle.

This macro is called at the end of a block that defines the scope of the binding. Users have to call `xbuf_activate` again to associate the buffer name with LMEM handle before the buffer can be used.

This call is applicable to Local Memory buffers only. It is transparent to other types.



Note

`lmem_buf0`, `lmem_buf1`, ..., `lmem_buf15` must be used for local memory buffers.

Example 1:

```
#define_eval ipv4_hdr    lmem_buf0
xbuf_alloc(ipv4_hdr, 8, read_write)
xbuf_bind_address(ipv4_hdr, 0x280, 0x20, 0)
xbuf_activate(ipv4_hdr, 0, 3, 1)
// uses of ipv4_hdr
xbuf_deactivate(ipv4_hdr)
xbuf_activate(ipv4_hdr, 1, 3, 1)
// uses of ipv4_hdr
xbuf_deactivate(ipv4_hdr)
xbuf_free(ipv4_hdr)
```

Example 2:

```
#define_eval ipv4_hdr    lmem_buf1
xbuf_alloc(ipv4_hdr, 8, read_write)
xbuf_bind_address(ipv4_hdr, pools_base, pool_size, buf_offset)
#macro first_block(name)
    xbuf_activate(name, 1, 2, 1)    // select handle1
    xbuf_extract(..., name, ...)
    xbuf_deactivate(name)
#endm
#macro second_block(name)
    xbuf_activate(name, 0, 2, 1)    // switch to handle0
    xbuf_extract(..., name, ...)
    xbuf_deactivate(name)
#endm
first_block(ipv4_hdr)
second_block(ipv4_hdr)
```

Instruction Count: 0

Table 2.335. xbuf_deactivate parameters

Name	Description
<i>str_xbuf_name</i>	Name of buffer. Name has to start with "lmem". For example, <code>lmem_buf1</code> . If name does not start with "lmem", the macro will assume that buffer is not in local memory and nothing will be done.

Macro Defines:

- `str_xbuf_name` is no longer associated with any Local Memory handle.

2.26.2.3 xbuf_activate

Prototype:

```
xbuf_activate(str_xbuf_name, INDEX, thread_id, WAIT_ACTION)
```

Description:

To map the specified ACTIVE_LM_ADDRESS CSR to the specified buffer.

The absolute address of the buffer will be put in the ACTIVE_LM_ADDR local CSR. The absolute address is calculated from the parameters supplied from the xbuf_bind_address call. After this macro is called, the specified buffer can be accessed via* l\$index0/1.

This call is applicable to LMEM buffers only. It is transparent to other buffer types.



Note

lmem_buf0, lmem_buf1, ..., lmem_buf15 must be used for Local Memory buffers.

Applications that run with 4 context mode have to explicitly define it:

```
#define CONTEXT_MODE_4
```

It is 8 context mode by default.

Buffer absolute address = (thread_id << [log(2) of (_xbuf[i]_lmem_pool_size)]) + _xbuf[i]_lmem_pools_base + _xbuf[i]_lmem_offset, where _xbuf[i]_lmem is the buffer with the specified name.

Example:

```
#define_eval ipv6_hdr    lmem_buf0

xbuf_alloc(ipv6_hdr, 8, read_write)
xbuf_bind_address(ipv6_hdr, pools_base, pool_size, buf_offset)
xbuf_activate(ipv6_hdr, 0, 3, 1)

Now for thread 3, ipv6_hdr[0], ipv6_hdr[1], ... ipv6_hdr[7]
are associated with *l$index0[0], *l$index0[1], ... *l$index0[7]
```

Instruction Count: 3 - 4 Without wait cycle, 6 - 7 With 3 wait cycles

Table 2.336. xbuf_activate parameters

Name	Description
<i>str_xbuf_name</i>	Name of buffer. Name has to start with "lmem". For example, lmem_buf1. If name does not start with "lmem", the macro will assume that buffer is not in local memory and nothing will be done.
<i>INDEX</i>	<ul style="list-style-type: none"> 0 to map* l\$index0 to the specified buffer

Name	Description
	<ul style="list-style-type: none"> 1 to map* l\$index1 to the specified buffer
<i>thread_id</i>	Run-time value thread ID. Can be either a GPR or a constant.
<i>WAIT_ACTION</i>	<ul style="list-style-type: none"> 1 wait 3 cycles after setting up local CSR (3 NOPs) 0 do not wait (0 NOP)

Macro Defines:

- Set ACTIVE_LM_ADDR_0 or ACTIVE_LM_ADDR_1
- Set _xbuf[i]_lmem_index = INDEX

2.26.2.4 xbuf_alloc

Prototype:

```
xbuf_alloc(str_xbuf_name, NUMBER_OF_REGS, RW_INDICATOR)
```

Description:

Locally declares contiguous 'NUMBER_OF_REGS' sram xfer registers and sets an appropriate .xfer_order.

Registers may be referenced as an array. This macro allows transfer registers to be divided into to a maximum of 4 buffers in IXP compatible indirect reference mode and 8 buffers in NFP indirect reference mode.

Example 1:

```
xbuf_alloc($x, 8, read)           // setup read only xfer regs $x0 thru $x7
sram_read($x0, addr1, addr2, LWCOUNT2, pend) // load the first 2 xfer regs
xbuf_free($x)
```

Example 2: [Using localmem]

```
xbuf_alloc(lmem_buf0, 8, READ_WRITE)
xbuf_bind_address(lmem_buf0, 0x100, 0x20, 0x0)
xbuf_activate(lmem_buf0, 0, 0, 1)

ipv4_cksum_verify(lmem_buf0, 0xb)
...
...
xbuf_deactivate(lmem_buf0)
xbuf_free(lmem_buf0)
```

Instruction Count: 0



Note

For local memory buffers: lmem_buf0, lmem_buf1, ..., lmem_buf15 must be used.

Table 2.337. xbuf_alloc parameters

Name	Description
<i>str_xbuf_name</i>	String uniquely identifying transfer register buffer
<i>NUMBER_OF_REGS</i>	Number of transfer registers to allocate
<i>RW_INDICATOR</i>	String = read, write, or read_write, which will specify the transfer registers as read only, write only, or read/write transfer registers

2.26.2.5 xbuf_free

Prototype:

```
xbuf_free(str_xbuf_name)
```

Description:

Used to deallocate transfer registers.

Example:

```
xbuf_alloc($x, 8)           // setup xfer regs $x0 thru $x7
sram_read($x0, addr1, addr2, LWCOUNT2, pend) // load the first 2 xfer regs
xbuf_free($x)
```

Example 2: [Using Localmem]

```
xbuf_alloc(lmem_buf0, 8, READ_WRITE)
xbuf_bind_address(lmem_buf0, 0x100, 0x20, 0x0)
xbuf_activate(lmem_buf0, 0, 0, 1)

ipv4_cksum_verify(lmem_buf0, 0xb)
...
...
xbuf_deactivate(lmem_buf0)
xbuf_free(lmem_buf0)
```

Instruction Count: 0



Note

For local memory buffers: lmem_buf0, lmem_buf1, ..., lmem_buf15 must be used.

Table 2.338. xbuf_free parameters

Name	Description
<i>str_xbuf_name</i>	Name of transfer register buffer

2.26.2.6 xbuf_link

Prototype:

```
xbuf_link(str_xbuf_name, str_nextxbuf_name)
```

Description:

Link one transfer register buffer to another.

The purpose is to allow processing streams of bytes by continually loading buffers and providing macros that can traverse the buffers.

Example 1:

```
xbuf_alloc($pkt, 4) // setup xfer regs $pkt0 thru $pkt3
// load the xfer buf $pkt, use it
xbuf_alloc($morepkt, 4) // setup xfer regs $x0 thru $x7
xbuf_link($pkt, $morepkt) // logical link pkt to morepkt
// load the second xfer buf $morepkt
ip_verify(exception, $pkt, ip_header_byte_position) // verify ip header that spans the buffers
xbuf_free($pkt)
xbuf_free($morepkt)
```

Example 2: [Using Localmem]

```
xbuf_alloc(lmem_buf0, 8, READ_WRITE)
xbuf_bind_address(lmem_buf0, 0x100, 0x20, 0x0)
xbuf_activate(lmem_buf0, 0, 0, 1)

xbuf_alloc(lmem_buf1, 8, READ_WRITE)
xbuf_bind_address(lmem_buf1, 0x100, 0x20, 32)
xbuf_activate(lmem_buf1, 1, 0, 1)
xbuf_link(lmem_buf0, lmem_buf1)
ipv4_cksum_verify(lmem_buf0, 0xb)
...
...
xbuf_deactivate(lmem_buf0)
xbuf_free(lmem_buf0)
```

Instruction Count: 0



Note

For local memory buffers: lmem_buf0, lmem_buf1, ..., lmem_buf15 must be used.

Table 2.339. xbuf_link parameters

Name	Description
<i>str_xbuf_name</i>	Transfer buffer allocated by xbuf_alloc
<i>str_nextxbuf_name</i>	Another transfer buffer allocated by xbuf_alloc. Next buffer continues stream of bytes.

2.26.2.7 xbuf_find

Prototype:

```
xbuf_find(xfer_name)
```

Description:

Given a transfer register name, removes special characters and array notation and then checks if the XBUF token name is an allocated XBUF.

This macro makes this XBUF the current XBUF.

Example:

```
xbuf_alloc($pkt, 4)                // setup xfer regs $pkt0 thru $pkt3
// load the xfer buf $pkt, use it
xbuf_find($pkt[2])
// load the second xfer buf $morepkt
xbuf_free($pkt)
```

Instruction Count: 0



Note

For local memory buffers: lmem_buf0, lmem_buf1, ..., lmem_buf15 must be used.

Table 2.340. xbuf_find parameters

Name	Description
<i>xfer_name</i>	Transfer buffer allocated by xbuf_alloc

2.26.2.8 xbuf_param_set

Prototype:

```
xbuf_param_set(str_xbuf_name)
```

Description:

Set global internal `_cur_xbuf_name`, `_cur_xbuf_size` and `_cur_xbuf_next` parameters.

This macro is used for following a chain of xbufs. The side-effect of this macro is to modify `_cur_xbuf_name`, `_cur_xbuf_size` and `_cur_xbuf_next`.

Example:

```
xbuf_alloc(a_reg, 4, read_write)
xbuf_alloc(b_reg, 4, read_write)
```

```
immed32(a_reg[0], 0x01000CCC)
immed32(a_reg[1], 0xCCCC0030)
immed32(a_reg[2], 0x788D43B7)
immed32(a_reg[3], 0x0032AAAA)

xbuf_param_set(a_reg)
xbuf_xfer_set(_BUF0, a_reg, 0)
//_BUF0_REG0 should be equal to 0x01000CCC
//_BUF0_REG1 should be equal to 0xCCCC0030
//_BUF0_REG2 should be equal to 0x788D43B7
//_BUF0_REG3 should be equal to 0x0032AAAA
```

Instruction Count: 0



Note

For local memory buffers: lmem_buf0, lmem_buf1, ..., lmem_buf15 must be used.

Table 2.341. xbuf_param_set parameters

Name	Description
<i>str_xbuf_name</i>	Name of sram transfer buffer

2.26.2.9 xbuf_xfer_set

Prototype:

```
xbuf_xfer_set(XFER_TOKEN, str_xbuf_name, BYTE_OFFSET)
```

Description:

Define the register token sequence from the xbufs, starting at the longword indicated by BYTE_OFFSET.

These tokens can then be used as individual identification of transfer registers. The search for transfers can span multiple linked xbufs. The side-effect of this macro is to modify global accessible tokens _XFER0, _XFER1, ..., _XFER7.

Example:

```
xbuf_alloc(a_reg, 4, read_write)
xbuf_alloc(b_reg, 4, read_write)

immed32(a_reg[0], 0x01000CCC)
immed32(a_reg[1], 0xCCCC0030)
immed32(a_reg[2], 0x788D43B7)
immed32(a_reg[3], 0x0032AAAA)

xbuf_param_set(a_reg)
xbuf_xfer_set(_BUF0, a_reg, 0)
//_BUF0_REG0 should be equal to 0x01000CCC
//_BUF0_REG1 should be equal to 0xCCCC0030
```

```
//_BUF0_REG2 should be equal to 0x788D43B7
//_BUF0_REG3 should be equal to 0x0032AAAA
```

Instruction Count: 0

Table 2.342. xbuf_xfer_set parameters

Name	Description
<i>XFER_TOKEN</i>	<div> <div>_S_XFER</div> <div>define tokens _XFER0 through _XFER15/_XFER31</div> </div> <div> <div>_SRD_XFER</div> <div>define tokens _SRD_XFER0 through _SRD_XFER15/_SRD_XFER31</div> </div> <div> <div>_SWR_XFER</div> <div>define tokens _SWR_XFER0 through _SWR_XFER15/_SWR_XFER31</div> </div> <div> <div>_D_XFER</div> <div>define tokens _D_XFER0 through _D_XFER15</div> <div>(only in IXP indirect reference format mode)</div> </div> <div> <div>_DRD_XFER</div> <div>define tokens _DRD_XFER0 through _DRD_XFER15</div> <div>(only in IXP indirect reference format mode)</div> </div> <div> <div>_DWR_XFER</div> <div>define tokens _DWR_XFER0 through _DWR_XFER15</div> <div>(only in IXP indirect reference format mode)</div> </div> <div> <div>_BUF0</div> <div>define tokens _BUF0_REG0 through _BUF0_REG7</div> </div> <div> <div>_BUF1</div> <div>define tokens _BUF1_REG0 through _BUF1_REG7</div> </div> <div> <div>_BUF2</div> <div>define tokens _BUF2_REG0 through _BUF2_REG7</div> </div> <div> <div>_BUF3</div> <div>define tokens _BUF3_REG0 through _BUF3_REG7</div> </div>
<i>str_xbuf_name</i>	Name of first sram transfer buffer
<i>BYTE_OFFSET</i>	Byte offset where _XFER0 is

2.26.2.10 xbuf_extract

Prototype:

```
xbuf_extract(out_byte_field, str_xbuf_name, window_start, field_start, number_of_bytes)
```

Description:

Extract a numeric byte field from a register buffer.



Note

1. No error checking in order to minimize the number of instructions. If users try to extract more than 4 bytes or pass in an field-offset that is bigger than the buffer size, the result will be undefined.
2. If the source buffer is GPRs, users can define a compile option `_EXTRACT_FROM_BANK_B_GPR` if they know for sure that the source is in bank B. When defined, this option makes this macro skip moving the source register to bank B, hence saving 1 to 2 instructions.

Example:

```
xbuf_alloc(wbuf, 2, read_write)
immed32(wbuf[0], 0x01020304)
immed32(wbuf[1], 0x50607080)
```

```
xbuf_alloc(wbuf_next1, 2, read_write)
immed32(wbuf_next1[0], 0x13459851)
immed32(wbuf_next1[1], 0x198428e5)

xbuf_alloc(wbuf_next2, 2, read_write)
immed32(wbuf_next2[0], 0x11111111)
immed32(wbuf_next2[1], 0x22222222)

xbuf_link(wbuf, wbuf_next1)
xbuf_link(wbuf_next1, wbuf_next2)

alu[win_start, --, B, 3]
alu[field_start, --, B, 2]
alu[num_of_bytes, --, B, 4]

xbuf_extract(out_byte_field, wbuf, win_start, field_start, num_of_bytes)
```

Instruction Count: 1-12

Table 2.343. xbuf_extract parameters

Name	Description
<i>out_byte_field</i>	GPR output field, right justified
<i>str_xbuf_name</i>	Name of source to extract from. It can be a buffer of transfer registers, local memory, or GPRs.
<i>window_start</i>	Start byte position of window to extract from. Offset from the beginning of the buffer to the location of the window to extract from.
<i>field_start</i>	Byte offset of the field to be extracted starting from <i>window_start</i> . <i>window_start</i> + <i>field_start</i> -> exact byte location of first byte to extract
<i>number_of_bytes</i>	Number of bytes to extract. Because <i>out_byte_field</i> is a 32-bit GPR, maximum number of bytes to extract is 4.

2.26.2.11 xbuf_extract_frm_linked_bufs

Prototype:

```
xbuf_extract_frm_linked_bufs(out_byte_field, str_xbuf_name, window_start, field_start,
number_of_bytes, DATA_SPREAD)
```

Description:

Extract a numeric byte field from a register buffer.

The source of data may be spread in 2 buffers: *str_xbuf_name* and the buffer that is linked to it. *DATA_SPREAD* indicates whether or not the data spreads 2 buffers. If *DATA_SPREAD* = 0, this macro functions the same way as *xbuf_extract*.

Instruction Count: 1-15

Table 2.344. xbuf_extract_frm_linked_bufs parameters

Name	Description
<i>out_byte_field</i>	GPR output field, right justified
<i>str_xbuf_name</i>	Name of sram read transfer register
<i>window_start</i>	Start byte position of window or header
<i>field_start</i>	Start byte offset of field from window start
<i>number_of_bytes</i>	Number of bytes to extract. Because the <i>out_byte_field</i> is a GPR (32-bit), maximum number of bytes to extract is 4.
<i>DATA_SPREAD</i>	CONSTANT to indicate whether source data spreads 2 buffers <ul style="list-style-type: none"> • 1: If part of data to be extracted spreads into buffer linked to <i>in_src_xbuf</i> • 0: All data to be extracted resides in <i>str_xbuf_name</i>.

2.26.2.12 xbuf_type_extract

Prototype:

```
xbuf_type_extract(out_field, str_xbuf_name, WINDOW_START, FIELD_START, SIZE, DATATYPE)
```

Description:

Extract a specified number of bytes from xbuf.

Perform endian swap if global define `LITTLE_ENDIAN`.

Example:

```
xbuf_alloc(wbuf, 8, read_write)

#define_eval WIN_START      0
#define_eval FIELD_START    0
#define_eval NUM_OF_BYTES   1

immed32(wbuf[0], 0x01020304)

xbuf_type_extract(out_byte_field, wbuf, WIN_START, FIELD_START, NUM_OF_BYTES, BIG_ENDIAN_BYTES)
//out_byte_field should be equal to 1
```

Instruction Count: 8-10

Table 2.345. xbuf_type_extract parameters

Name	Description
<i>out_field</i>	GPR containing field, right justified
<i>str_xbuf_name</i>	Name of sram transfer register buffer
<i>WINDOW_START</i>	Start position (in datatype increments) of window or header
<i>FIELD_START</i>	Start offset (in datatype increments) of field from window start

Name	Description
<i>SIZE</i>	Size of field
<i>DATATYPE</i>	Datatype of field

2.26.2.13 xbuf_insert

Prototype:

```
xbuf_insert(io_str_xbuf_name, in_byte_field, window_start, field_start, number_of_bytes)
```

Description:

Insert specified bytes of a numeric byte field into xbuf buffer at the specified start byte offset.



Note

1. No error checking in order to minimize the number of instruction. If users try to insert more than 4 bytes or pass in an field-offset that is bigger than the buffer size, the result will be undefined.
2. If the source register is GPRs, users can define a compile option `_INSERT_BYTE_FROM_BANK_B_GPR` if they know for sure that the source is in bank B. When defined, this option makes this macro skip moving the source register to bank B, hence save 1 instruction.

Example usage:

```
xbuf_alloc($io_xbuf_dest, 8, read_write)
...
move(in_byte_field, 0x12345678)
//Insert in_byte_field into io_xbuf_dest starting at byte offset 12
xbuf_insert($io_xbuf_dest, in_byte_field, 0, 12, 4)
...
```

Instruction Count: (1 * number of words to insert) for constant offsets and size

Table 2.346. xbuf_insert parameters

Name	Description
<i>io_str_xbuf_name</i>	Name of xbuf register where byte field is to be inserted
<i>in_byte_field</i>	GPR, read transfer register or local memory register that contains byte field to be inserted.
<i>window_start</i>	Start byte position of window to insert to. This is offset from beginning of output buffer to location of window to insert into.
<i>field_start</i>	Byte offset of field to be inserted starting from <i>window_start</i> . <i>window_start</i> + <i>field_start</i> -> exact byte location of the first byte to insert
<i>number_of_bytes</i>	Number of bytes to insert. Because <i>in_byte_field</i> is a 32-bit register, maximum number of bytes is 4.

2.26.2.14 xbuf_copy

Prototype:

```
xbuf_copy(out_dest_xbuf, out_last_element, dest_start_byte, in_src_xbuf, src_start_byte,
in_prepend, total_bytes_to_copy, DATA_SPREAD)
```

Description:

XBUF Copy.

Table 2.347. xbuf_copy parameters

Name	Description
<i>out_dest_xbuf</i>	Name of output buffer. Can be SRAM write registers, DRAM write registers, local memory, or GPR.
<i>out_last_element</i>	GPR to contain last element of destination buffer. This output is very useful when the destination buffer elements are write transfer register and the copy results in an incompleted last element (not all 4 bytes are filled). The need for <i>out_last_element</i> arises when users want to copy another buffer into a paritally-filled destination buffer. Users may find that the last long-word element in the destination is incompleted as the result of the previous copy. In that case, they will need to pass the <i>out_last_element</i> of the previous copy as the <i>in_prepend</i> to the current <i>xbuf_copy</i> . Otherwise, the bytes in the partially-filled write register element will be cleared. If users are sure that they do not need this value, they can pass the constant 0 to save one instruction.
<i>dest_start_byte</i>	Absolute offset in bytes from beginning of output buffer to location where copied data will reside. Can be longword aligned or not ($\text{dest_start_byte} \% 4 = 0, 1, 2, \text{ or } 3$). Can be constant or GPR.
<i>in_src_xbuf</i>	Name of input buffer. Can be SRAM read registers, DRAM read registers, local memory, or GPR. Data to be copied can not spread in several buffers. In other words, all data has to be in <i>in_src_xbuf</i> . In order to have most efficient copy (ie. done with least number of instructions), <i>in_src_xbuf</i> should be buffer that actually contains the data. It is fine if the data spreads beyond <i>in_src_xbuf</i> , but it will take several more instructions if ALL the data does not reside in some buffers that are linked to <i>in_src_xbuf</i> .
<i>src_start_byte</i>	Absolute offset in bytes from beginning of input buffer. Can be longword aligned or not ($\text{src_start_byte} \% 4 = 0, 1, 2, \text{ or } 3$) Can be constant or GPR.
<i>in_prepend</i>	GPR or constant containing bytes to be merged with first word in destination, prepended, if byte alignment of destination is not 0. Bytes to be merged must be at exact byte locations that they will occupy in first word of destination. All other bytes must be 0. In cases <i>in_prepend</i> is not needed, just pass constant 0.
<i>total_bytes_to_copy</i>	Total number of bytes to copy. Can be constant or GPR. Maximum number of bytes is dependent on maximum size of output buffer. <ul style="list-style-type: none"> • If <i>out_dest_xbuf</i> is xfer registers or GPRs buffer: maximum size is 64 bytes. • If <i>out_dest_xbuf</i> is local memory buffer: maximum size is 32 bytes.
<i>DATA_SPREAD</i>	<ul style="list-style-type: none"> • 1: If part of data to be copied spreads into buffer(s) linked to <i>in_src_xbuf</i>

Name	Description
	<ul style="list-style-type: none">• 0: All data to be copied resides in <code>in_src_xbuf</code>. <code>DATA_SPREAD = 0</code> or <code>1</code> are allowed for copy with constant offsets and size. In other words, if ALL offsets and size are constants, data to be copied can completely reside in <code>in_src_xbuf</code>, or be spread between <code>in_src_buf</code> and buffer that is linked to <code>in_src_buf</code>. For copy with run-time offsets and/or run-time size <code>DATA_SPREAD</code> must be <code>0</code>. In other words, data spreading is not allowed for <code>xbuf_copy</code> with any run-time parameters.

3. Technical Support

To obtain additional information, or to provide feedback, please email [<support@netronome.com>](mailto:support@netronome.com) or contact the nearest **Netronome** technical support representative.