Ingenic®
SIMD/DSP Instruction Set

Revision: 1.0
Date: Apr. 2006
Ingenic SIMD/DSP Instruction Set

Copyright © Ingenic Semiconductor Co. Ltd 2006. All rights reserved.

Release history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr. 2006</td>
<td>1.0</td>
<td>Release-1 (Jz4740 only supports this version)</td>
</tr>
<tr>
<td>Oct. 2007</td>
<td>1.1</td>
<td>Release-2</td>
</tr>
</tbody>
</table>

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Co. Ltd

22th Floor, Building A, Cyber Tower, No.2, Zhong Guan Cun South Avenue
Haidian District, Beijing 100086, China
Tel: 86-10-82511297
Fax: 86-10-82511589
Http://www.ingenic.cn
# Content

1.1 Extended registers for SIMD/DSP Instruction Set .................................................................................................................. 1
   1.1.1 General purpose registers ................................................................................................................................. 1
   1.1.2 MXU control register (MXU_CR) ......................................................................................................................... 1
   1.1.3 Rounding Operation ............................................................................................................................................. 2
   1.1.4 Fraction multiplication ............................................................................................................................................ 2

1.2 Instruction format ..................................................................................................................................................................... 3
   1.2.1 Instruction assembly mnemonic .......................................................................................................................... 3
   1.2.2 Instruction operands .................................................................................................................................................. 4
   1.2.3 Pattern of 16bit Add/Multiply .................................................................................................................................. 5

1.3 Instruction summary ................................................................................................................................................................ 6

1.4 Macro functions for convenience of description .................................................................................................................. 9

1.5 Load/Store ................................................................................................................................................................................... 10
   1.5.1 S32LDDR/S32STDR .............................................................................................................................................. 10
   1.5.2 S32LDD/S32STD .................................................................................................................................................. 11
   1.5.3 S32LDDVR/S32STDVR ....................................................................................................................................... 12
   1.5.4 S32LDDV/S32STDV ............................................................................................................................................ 13
   1.5.5 S32LDIR/S32SDIR ................................................................................................................................................ 14
   1.5.6 S32LDI/S32SDI .................................................................................................................................................... 15
   1.5.7 S32LDIVR/S32SDIVR ......................................................................................................................................... 16
   1.5.8 S32LDIV/S32SDIV ............................................................................................................................................... 17
   1.5.9 LXW ...................................................................................................................................................................... 18
   1.5.10 S16LDD ............................................................................................................................................................. 19
   1.5.11 S16LDI ............................................................................................................................................................... 20
   1.5.12 S16STD ............................................................................................................................................................... 21
   1.5.13 S16SDI ............................................................................................................................................................... 22
   1.5.14 LXH/LXHU ......................................................................................................................................................... 23
   1.5.15 S8LDD ............................................................................................................................................................... 24
   1.5.16 S8LDI ............................................................................................................................................................... 25
   1.5.17 S8STD ............................................................................................................................................................... 26
   1.5.18 S8SDI ............................................................................................................................................................... 27
   1.5.19 LX/H/LXBU ...................................................................................................................................................... 28

1.6 Multiplication with/without accumulation .............................................................................................................................. 29
   1.6.1 S32MUL .............................................................................................................................................................. 29
   1.6.2 S32MULU ............................................................................................................................................................ 30
   1.6.3 S32MADD .......................................................................................................................................................... 31
   1.6.4 S32MADDU ......................................................................................................................................................... 32
   1.6.5 S32MSUB .......................................................................................................................................................... 33
   1.6.6 S32MSUBU ....................................................................................................................................................... 34
   1.6.7 D16MUL .............................................................................................................................................................. 35
   1.6.8 D16MULF ............................................................................................................................................................ 36
   1.6.9 D16MULE ............................................................................................................................................................ 37
# 1.7 SIMD/DSP Instruction Set

## 1.7.1 Add and subtract

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D16MAC</td>
<td>38</td>
</tr>
<tr>
<td>D16MACF</td>
<td>39</td>
</tr>
<tr>
<td>D16MACE</td>
<td>40</td>
</tr>
<tr>
<td>D16MADL</td>
<td>41</td>
</tr>
<tr>
<td>S16MAD</td>
<td>42</td>
</tr>
<tr>
<td>Q8MUL</td>
<td>43</td>
</tr>
<tr>
<td>Q8MULSU</td>
<td>44</td>
</tr>
<tr>
<td>Q8MAC</td>
<td>45</td>
</tr>
<tr>
<td>Q8MCSU</td>
<td>46</td>
</tr>
<tr>
<td>Q8MADL (obsolete)</td>
<td>47</td>
</tr>
<tr>
<td>Q8ADD (obsolete)</td>
<td>48</td>
</tr>
<tr>
<td>Q8ADDE</td>
<td>49</td>
</tr>
<tr>
<td>Q16ADD</td>
<td>50</td>
</tr>
<tr>
<td>Q16ACC</td>
<td>51</td>
</tr>
<tr>
<td>Q16ACCM</td>
<td>52</td>
</tr>
<tr>
<td>D16ASUM</td>
<td>53</td>
</tr>
<tr>
<td>D16CPS</td>
<td>54</td>
</tr>
<tr>
<td>D16SLT</td>
<td>55</td>
</tr>
<tr>
<td>S32MOVN</td>
<td>56</td>
</tr>
<tr>
<td>Q16ADD</td>
<td>57</td>
</tr>
<tr>
<td>Q16ACC</td>
<td>58</td>
</tr>
<tr>
<td>Q16ACCM</td>
<td>59</td>
</tr>
<tr>
<td>D16ASUM</td>
<td>60</td>
</tr>
<tr>
<td>D16CPS</td>
<td>61</td>
</tr>
<tr>
<td>D16SLT</td>
<td>62</td>
</tr>
<tr>
<td>D16MOVZ</td>
<td>63</td>
</tr>
<tr>
<td>S32MOVN</td>
<td>64</td>
</tr>
<tr>
<td>Q16ADD</td>
<td>65</td>
</tr>
<tr>
<td>Q16ACC</td>
<td>66</td>
</tr>
<tr>
<td>Q16ACCM</td>
<td>67</td>
</tr>
<tr>
<td>D16ASUM</td>
<td>68</td>
</tr>
<tr>
<td>D16CPS</td>
<td>69</td>
</tr>
<tr>
<td>D16SLT</td>
<td>70</td>
</tr>
<tr>
<td>D16MOVZ</td>
<td>71</td>
</tr>
<tr>
<td>D16MADL (obsolete)</td>
<td>72</td>
</tr>
<tr>
<td>Q8MUL</td>
<td>73</td>
</tr>
<tr>
<td>Q8MULSU</td>
<td>74</td>
</tr>
<tr>
<td>Q8MAC</td>
<td>75</td>
</tr>
<tr>
<td>Q8MCSU</td>
<td>76</td>
</tr>
<tr>
<td>Q8MADL (obsolete)</td>
<td>77</td>
</tr>
<tr>
<td>Q8ADD (obsolete)</td>
<td>78</td>
</tr>
<tr>
<td>Q8ADDE</td>
<td>79</td>
</tr>
<tr>
<td>Q8AVG</td>
<td></td>
</tr>
<tr>
<td>Q8AVGR</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2005-2007 Ingenic Semiconductor Co., Ltd. All rights reserved.
1.8 Shift ....................................................................................................................................... 80
  1.8.1 D32SLL/D32SLR/D32SAR ............................................................................................ 80
  1.8.2 D32SARL .................................................................................................................. 81
  1.8.3 D32SLLV/D32SLRV/D32SARV .................................................................................. 82
  1.8.4 D32SARW .................................................................................................................. 83
  1.8.5 Q16SLL/Q16SLR/Q16SAR .......................................................................................... 84
  1.8.6 Q16SLLV/Q16SLRV/Q16SARV .................................................................................. 85
  1.8.7 S32EXTR .................................................................................................................. 86
  1.8.8 S32EXTRV .................................................................................................................. 87
1.9 MAX/MIN............................................................................................................................... 88
  1.9.1 S32MAX .................................................................................................................... 88
  1.9.2 S32MIN ..................................................................................................................... 89
  1.9.3 D16MAX .................................................................................................................... 90
  1.9.4 D16MIN .................................................................................................................... 91
  1.9.5 Q8MAX ..................................................................................................................... 92
  1.9.6 Q8MIN ....................................................................................................................... 93
1.10 Bitwise................................................................................................................................... 94
  1.10.1 S32AND .................................................................................................................... 94
  1.10.2 S32OR ....................................................................................................................... 95
  1.10.3 S32XOR ..................................................................................................................... 96
  1.10.4 S32NOR .................................................................................................................... 97
1.11 Register move between GRF and XRF ................................................................................ 98
  1.11.1 S32M2I/S32I2M ......................................................................................................... 98
1.12 Miscellaneous ....................................................................................................................... 99
  1.12.1 S32SFL ..................................................................................................................... 99
  1.12.2 S32ALN/S32ALNI .................................................................................................... 100
  1.12.3 Q16SAT ................................................................................................................... 101
  1.12.4 Q16SCOP ................................................................................................................ 102
  1.12.5 S32LUI .................................................................................................................... 103
This document depicts the JZ SIMD/DSP instructions which serving for MXU – Media Extension Unit. Except Jz4730 MXU is a basic component in Jz47xx series belonging to Xburst uA.

1.1 Extended registers for SIMD/DSP Instruction Set

1.1.1 General purpose registers

In MXU, a dedicated register file named XRF comprises sixteen 32-bit general purpose registers – XR0~XR15. XR0 is a special one, which always is read as zero. Moreover, XR16 is an alias of MXU_CR described below and it can only be accessed by S32I2M/S32M2I instructions.

1.1.2 MXU control register (MXU_CR)

| Name  | Bit 31 | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-------|--------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | LC     | RC   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| RST   | 0      | 0    |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| R/W   |        |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Note  | Reserved|      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |

- MXUEN: Flag of MXU enable. 1 – enable. 0 - disable
- RD_EN: Flag of round enable, 1 – round enable, 0 – round disable. Effective for D16MULF, D16MUL, D16MACF and D16MACE instructions.
- BIAS: Biased rounding and un-biased rounding toggle. 0 – un-bias, 1 – bias.
- RC: Carry out of right lane’s adder
- LC: Carry out of left lane’s adder

Note that only when MXU is enabled (default is disable), executing SIMD/DSP instructions is valid, otherwise, the result is unpredictable. However, S32M2I/S32I2M can be executed without any constraint.

Moreover, when MXUEN is toggled from disable to enable by executing S32I2M, there are at least three non-SIMD/DSP instructions necessary to serve as isolation between the S32I2M instruction and following available SIMD/DSP instructions. However, when MXUEN is disabled, following SIMD/DSP instructions’ being executed will be nullified immediately.
1.1.3 Rounding Operation
Rounding result is an option for 16x16 multiply and accumulation operations. The upper portion (bits 31~16) of the accumulator is rounded according to the contents of the lower portion (bits 15~0) of the accumulator. MXU supports two rounding modes: biased rounding and un-biased rounding.

- **Un-biased rounding (convergent rounding)**
Un-biased rounding (also called round-to-nearest even number) is the default-rounding mode. The traditional rounding method rounds up any lower portion value greater than or equal to the midway point (0x8000) and rounds down any value less than the point. Thus the final results are eventually biased to greater-than-zero direction. Convergent rounding solves the pitfall by rounding down if the number (bit 16) is even (= 0) and rounding up if the number (bit 16) is odd (= 1) for midway point.

- **Biased rounding**
When biased rounding is selected by setting BIAS bit field in the MXU_CR, all lower portion values greater than or equal to the midway point are rounded up and all values less than the midway point are rounded down. Therefore, a small positive bias is introduced.

1.1.4 Fraction multiplication
When execute the D16MULF/D16MULE or D16MACF/D16MACE, MXU takes it for granted that the multiplier operates in fraction mode (Q15 format). Before updating the destination registers, the rounding operation will be done or not indicated by MXU_CR.RD_EN.
1.2 Instruction format

1.2.1 Instruction assembly mnemonic

The SIMD/DSP instruction assembly mnemonic format is as follow

<Operation parallel level><Operand size><Operation abbr.>[<Suffix>]

Operation parallel level: S (single), D (dual) and Q (Quad)
Operand size (bit): 31, 16 and 8
Operation abbr.: LDD, LDI, STD, SDI, MUL, MAC, MAD, ADD, ACC, CPS, AVG, ABD, SLT, SAD, SLL, SLR, SAR, MAX, MIN, M2I, I2M, SAT, ALN, SFL
Suffix (optional): V, L, F, R, E and W

1.2.1.1 Operation abbreviation

<table>
<thead>
<tr>
<th>Abbr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Add or subtract</td>
</tr>
<tr>
<td>ADDC</td>
<td>Add with carry-in</td>
</tr>
<tr>
<td>ACC</td>
<td>Add and accumulate</td>
</tr>
<tr>
<td>ASUM</td>
<td>Sum together to accumulate (add or subtract)</td>
</tr>
<tr>
<td>ASUMC</td>
<td>Sum with carry-in together to accumulate (add or subtract)</td>
</tr>
<tr>
<td>AVG</td>
<td>Average between 2 operands</td>
</tr>
<tr>
<td>ABD</td>
<td>Absolute difference</td>
</tr>
<tr>
<td>ALN</td>
<td>Align data</td>
</tr>
<tr>
<td>AND</td>
<td>a &amp; b</td>
</tr>
<tr>
<td>CPS</td>
<td>Copy sign, used to get absolute value</td>
</tr>
<tr>
<td>EXTR</td>
<td>Extract specific bits from MSB position</td>
</tr>
<tr>
<td>I2M</td>
<td>Move from IU to MXU</td>
</tr>
<tr>
<td>LDD</td>
<td>Load data from memory to XRF</td>
</tr>
<tr>
<td>LDI</td>
<td>Load data from memory to XRF and increase the address base register</td>
</tr>
<tr>
<td>LUI</td>
<td>Load unsigned immediate</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiply</td>
</tr>
<tr>
<td>MULU</td>
<td>Unsigned multiply</td>
</tr>
<tr>
<td>MADD</td>
<td>64-bit operand add 32x32 product</td>
</tr>
<tr>
<td>MSUB</td>
<td>64-bit operand subtract 32x32 product</td>
</tr>
<tr>
<td>MAC</td>
<td>Multiply and accumulate (add or subtract)</td>
</tr>
<tr>
<td>MAD</td>
<td>Multiply and add or subtract</td>
</tr>
<tr>
<td>MAX</td>
<td>Maximum between 2 operands</td>
</tr>
<tr>
<td>MIN</td>
<td>Minimum between 2 operands</td>
</tr>
<tr>
<td>M2I</td>
<td>Move from MXU to IU</td>
</tr>
<tr>
<td>MOVZ</td>
<td>Move if zero</td>
</tr>
<tr>
<td>MOVN</td>
<td>Move if non-zero</td>
</tr>
<tr>
<td>NOR</td>
<td>~ (a</td>
</tr>
<tr>
<td>OR</td>
<td>a</td>
</tr>
<tr>
<td>----------</td>
<td>---</td>
</tr>
<tr>
<td>STD</td>
<td>Store data from XRF to memory</td>
</tr>
<tr>
<td>SDI</td>
<td>Store data from XRF to memory and increase the address base register</td>
</tr>
<tr>
<td>SLT</td>
<td>Set of less than comparison</td>
</tr>
<tr>
<td>SAD</td>
<td>Sum of absolute differences</td>
</tr>
<tr>
<td>SLL</td>
<td>Shift logic left</td>
</tr>
<tr>
<td>SLR</td>
<td>Shift logic right</td>
</tr>
<tr>
<td>SAR</td>
<td>Shift arithmetic right</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturation</td>
</tr>
<tr>
<td>SFL</td>
<td>Shuffle</td>
</tr>
<tr>
<td>SCOP</td>
<td>Calculate x's scope (-1: means x&lt;0, 0: means x==0, 1: means x&gt;0)</td>
</tr>
<tr>
<td>XOR</td>
<td>a ^ b</td>
</tr>
</tbody>
</table>

*: These operations belong to SIMD/DSP release-2.

### 1.2.1.2 Suffix abbreviation

<table>
<thead>
<tr>
<th>Abbr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>Variable instead of immediate</td>
</tr>
<tr>
<td>L</td>
<td>Low part result</td>
</tr>
<tr>
<td>W</td>
<td>Combination of L and V</td>
</tr>
<tr>
<td>F</td>
<td>Fixed point multiplication</td>
</tr>
<tr>
<td>R</td>
<td>Rounding</td>
</tr>
<tr>
<td>E</td>
<td>Expanded results</td>
</tr>
</tbody>
</table>

### 1.2.2 Instruction operands

#### 1.2.2.1 Some immediate number abbreviation

<table>
<thead>
<tr>
<th>Abbr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRD2</td>
<td>Stride of the address index register</td>
</tr>
<tr>
<td>OPTN1,OPTN2</td>
<td>Operand getting pattern</td>
</tr>
<tr>
<td>APTN1, APTN2</td>
<td>Accumulate pattern, how to do the accumulate, add or subtract</td>
</tr>
<tr>
<td>EPTN2</td>
<td>Execute pattern, how to do the execution, add or subtract</td>
</tr>
</tbody>
</table>
1.2.3 Pattern of 16bit Add/Multiply

NOTE: lop denotes left operating lane, rop denotes right operating lane

Above diagram shows the four cases of \texttt{pattern\_MUL} or \texttt{pattern\_ADD}.
## 1.3 Instruction summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load &amp; Store</td>
<td>S32LDD</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32STD</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32LDDV</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S32STDV</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S32LDI</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32SDI</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32LDIV</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S32SDIV</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S32LDDR</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32STDR</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32LDDVR</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S32STDVR</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S32LDIR</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32SDIR</td>
<td>xra, rb, s12</td>
</tr>
<tr>
<td></td>
<td>S32LDIVR</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S32SDIVR</td>
<td>xra, rb, rc, strd2</td>
</tr>
<tr>
<td></td>
<td>S16LDD</td>
<td>xra, rb, s10, eptn2</td>
</tr>
<tr>
<td></td>
<td>S16STD</td>
<td>xra, rb, s10, eptn2</td>
</tr>
<tr>
<td></td>
<td>S16LDI</td>
<td>xra, rb, s10, eptn2</td>
</tr>
<tr>
<td></td>
<td>S16SDI</td>
<td>xra, rb, s10, eptn2</td>
</tr>
<tr>
<td></td>
<td>S8LDD</td>
<td>xra, rb, s8, eptn3</td>
</tr>
<tr>
<td></td>
<td>S8STD</td>
<td>xra, rb, s8, eptn3</td>
</tr>
<tr>
<td></td>
<td>S8LDI</td>
<td>xra, rb, s8, eptn3</td>
</tr>
<tr>
<td></td>
<td>S8SDI</td>
<td>xra, rb, s8, eptn3</td>
</tr>
<tr>
<td></td>
<td>LXW</td>
<td>rd, rs, rt, strd2</td>
</tr>
<tr>
<td></td>
<td>LXH</td>
<td>rd, rs, rt, strd2</td>
</tr>
<tr>
<td></td>
<td>LXHU</td>
<td>rd, rs, rt, strd2</td>
</tr>
<tr>
<td></td>
<td>LXB</td>
<td>rd, rs, rt, strd2</td>
</tr>
<tr>
<td></td>
<td>LXBU</td>
<td>rd, rs, rt, strd2</td>
</tr>
<tr>
<td>MUL &amp; MAC</td>
<td>S32MADD</td>
<td>xra, xrd, rs, rt</td>
</tr>
<tr>
<td></td>
<td>S32MADDU</td>
<td>xra, xrd, rs, rt</td>
</tr>
<tr>
<td></td>
<td>S32SUB</td>
<td>xra, xrd, rs, rt</td>
</tr>
<tr>
<td></td>
<td>S32SUBU</td>
<td>xra, xrd, rs, rt</td>
</tr>
<tr>
<td></td>
<td>S32MUL</td>
<td>xra, xrd, rs, rt</td>
</tr>
<tr>
<td></td>
<td>S32MULU</td>
<td>xra, xrd, rs, rt</td>
</tr>
<tr>
<td></td>
<td>D16MUL</td>
<td>xra, xrb, xrc, xrd, optn2</td>
</tr>
</tbody>
</table>
### SIMD/DSP Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D16MULE</td>
<td>xra, xrb, xrc, optn2</td>
</tr>
<tr>
<td>D16MULF</td>
<td>xra, xrb, xrc, optn2</td>
</tr>
<tr>
<td>D16MAC</td>
<td>xra, xrb, xrc, xrd, aptn2, optn2</td>
</tr>
<tr>
<td>D16MACE</td>
<td>xra, xrb, xrc, xrd, aptn2, optn2</td>
</tr>
<tr>
<td>D16MACF</td>
<td>xra, xrb, xrc, xrd, aptn2, optn2</td>
</tr>
<tr>
<td>D16MADL</td>
<td>xra, xrb, xrc, xrd, aptn2, optn2</td>
</tr>
<tr>
<td>S16MAD</td>
<td>xra, xrb, xrc, xrd, aptn1, optn1</td>
</tr>
<tr>
<td>Q8MUL</td>
<td>xra, xrb, xrc, xrd</td>
</tr>
<tr>
<td>Q8MULSU</td>
<td>xra, xrb, xrc, xrd</td>
</tr>
<tr>
<td>Q8MAC</td>
<td>xra, xrb, xrc, xrd, aptn2</td>
</tr>
<tr>
<td>Q8MACSU</td>
<td>xra, xrb, xrc, xrd, aptn2</td>
</tr>
<tr>
<td>Q8MADL</td>
<td>xra, xrb, xrc, xrd, optn2</td>
</tr>
<tr>
<td>D32ADD</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>D32ADDC</td>
<td>xra, xrb, xrc, xrd</td>
</tr>
<tr>
<td>D32ACC</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>D32ACCM</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>D32ASUM</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>S32CPS</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q16ADD</td>
<td>xra, xrb, xrc, xrd, eptn2, optn2</td>
</tr>
<tr>
<td>Q16ACC</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>Q16ACCM</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>D16ASUM</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>D16CPS</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D16AVG</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D16AVGR</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8ADD</td>
<td>xra, xrb, xrc, eptn2</td>
</tr>
<tr>
<td>Q8ADDE</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>Q8ACCE</td>
<td>xra, xrb, xrc, xrd, eptn2</td>
</tr>
<tr>
<td>Q8ABD</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8SAD</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8AVG</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8AVGR</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D8SUM</td>
<td>xra, xrb, xrc, xrd</td>
</tr>
<tr>
<td>D8SUMC</td>
<td>xra, xrb, xrc, xrd</td>
</tr>
</tbody>
</table>

### Sum & Difference (22)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D32SLL</td>
<td>xra, xrb, xrc, xrd, sft4</td>
</tr>
<tr>
<td>D32SLR</td>
<td>xra, xrb, xrc, xrd, sft4</td>
</tr>
<tr>
<td>D32SAR</td>
<td>xra, xrb, xrc, xrd, sft4</td>
</tr>
<tr>
<td>D32SARL</td>
<td>xra, xrb, xrc, sft4</td>
</tr>
<tr>
<td>D32SLLV</td>
<td>xra, xrb, rb</td>
</tr>
<tr>
<td>D32SLRV</td>
<td>xra, xrb, rb</td>
</tr>
<tr>
<td>Instruction</td>
<td>Format</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>D32SARV</td>
<td>xra, xrb, rb</td>
</tr>
<tr>
<td>D32SARW</td>
<td>xra, xrb, xrc, rb</td>
</tr>
<tr>
<td>Q16SLL</td>
<td>xra, xrb, xrc, xrd, sft4</td>
</tr>
<tr>
<td>Q16SLR</td>
<td>xra, xrb, xrc, xrd, sft4</td>
</tr>
<tr>
<td>Q16SAR</td>
<td>xra, xrb, xrc, xrd, sft4</td>
</tr>
<tr>
<td>Q16SLLV</td>
<td>xra, xrb, rb</td>
</tr>
<tr>
<td>Q16SLRV</td>
<td>xra, xrb, rb</td>
</tr>
<tr>
<td>Q16SARV</td>
<td>xra, xrb, rb</td>
</tr>
<tr>
<td>Compare (16)</td>
<td></td>
</tr>
<tr>
<td>S32MAX</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>S32MIN</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>S32SLT</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>S32MOVZ</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>S32MOVN</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D16MAX</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D16MIN</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D16SLT</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D16MOVZ</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>D16MOVN</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8MAX</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8MIN</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8SLT</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8SLTU</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8MOVZ</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Q8MOVN</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Bitwise (4)</td>
<td></td>
</tr>
<tr>
<td>S32NOR</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>S32AND</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>S32XOR</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>S32OR</td>
<td>xra, xrb, xrc</td>
</tr>
<tr>
<td>Move (2)</td>
<td></td>
</tr>
<tr>
<td>S32M2I</td>
<td>xra, rb</td>
</tr>
<tr>
<td>S32I2M</td>
<td>xra, rb</td>
</tr>
<tr>
<td>Misc (8)</td>
<td></td>
</tr>
<tr>
<td>S32SFL</td>
<td>xra, xrb, xrc, xrd, optn2</td>
</tr>
<tr>
<td>S32ALN</td>
<td>xra, xrb, xrc, rb</td>
</tr>
<tr>
<td>S32ALNI</td>
<td>xra, xrb, xrc, s3</td>
</tr>
<tr>
<td>S32LU1</td>
<td>xra, s8, optn3</td>
</tr>
<tr>
<td>S32EXTR</td>
<td>xra, xrb, rb, bits5</td>
</tr>
<tr>
<td>S32EXTRV</td>
<td>xra, xrb, rs, rt</td>
</tr>
</tbody>
</table>
1.4 Macro functions for convenience of description

This section defines some useful macro functions that will be frequently referenced in later chapters for instruction description.

- **sign_ext32(x)**
  Sign extend x to a 32-bit signed value.

- **zero_ext32(x)**
  Zero extend x to a 32-bit unsigned value.

- **sign_ext16(x)**
  Sign extend x to a 16-bit signed value.

- **zero_ext16(x)**
  Zero extend x to a 16-bit unsigned value.

- **zero_ext10(x)**
  Zero extend x to a 10-bit unsigned value.

- **zero_ext9(x)**
  Zero extend x to a 9-bit unsigned value.

- **signed(x)**
  x represents a signed value.

- **unsigned(x)**
  x represents an unsigned value

- **trunc_8(x)**
  Truncate bits of x except the lower 8 bits.

- **trunc_32(x)**
  Truncate bits of x except the lower 32 bits.

- **usat_8(x)**
  Saturate x to value 0 to 255. That is, if x < 0, then the result is 0; if x > 255, then result is 255. Others, result = x.

- **abs(x)**
  Get the absolution of x.

- **(>>L)**
  Logic right shift.

- **(>>A)**
  Arithmetic right shift.

Note that for all 16bit calculations (S16, D16, Q16), signed operation is default.
1.5 Load/Store

1.5.1 S32LDDR/S32STDR

Syntax:

S32LDDR XRa, rb, S12  
S32STDR XRa, rb, S12

Operation:

S32LDDR:
  tmp32 = (Mem32 [rb + S12]);
  XRa = {tmp32[7:0], tmp32[15:8], tmp32[23:16], tmp32[31:24]};

S32STDR:
  (Mem32 [rb + S12])
  = {XRa[7:0], XRa[15:8], XRa[23:16], XRa[31:24]};

Description:

S32LDDR: Load a word data from the memory address formed by adding rb with S12, reverse the word’s byte sequence and subsequently update XRa with the reversed result.
S32STDR: Store a word to the memory address formed by adding rb with S12, the byte sequence of the word must be reversed first.

NOTE:

1. S12 should be the multiple of 4.
2. The offset range represented by S12 is –2048 ~ 2044.
3. To avoid address error exception, the address formed must be 4-byte aligned.

Example:

S32LDDR XR1, $1, -4  
S32STDR XR15, v0, 0
1.5.2 S32LDD/S32STD

Syntax:

S32LDD XRa, rb, S12
S32STD XRa, rb, S12

Operation:

S32LDD:
XRa = (Mem32 [rb + S12]);

S32STD:
(Mem32 [rb + S12]) = XRa;

Description:

S32LDD: Load a word from the address formed by adding rb with S12.
S32STD: Store a word to the address formed by adding rb with S12.

NOTE:

1. S12 should be the multiple of 4.
2. The offset range represented by S12 is –2048 ~ 2044.
3. To avoid address error exception, the address formed must be 4-byte aligned.

Example:

S32LDD XR11, $5, -4
S32STD XR3, a1, 2044
1.5.3  S32LDDVR/S32STDVR

Syntax:

\[
\begin{align*}
  \text{S32LDDVR} & \quad \text{XRa, rb, rc, STRD2} \\
  \text{S32STDVR} & \quad \text{XRa, rb, rc, STRD2}
\end{align*}
\]

Parameter:

\[
\text{STRD2: 0-2}
\]

Operation:

\[
\begin{align*}
  \text{S32LDDVR:} & \\
  \text{tmp32} & = (\text{Mem32} [\text{rb} + (\text{rc} \ll \text{STRD2})]); \\
  \text{XRa} & = \{\text{tmp32}[7:0], \text{tmp32}[15:8], \text{tmp32}[23:16], \text{tmp32}[31:24]\};
\end{align*}
\]

\[
\begin{align*}
  \text{S32STDVR:} & \\
  (\text{Mem32} [\text{rb} + (\text{rc} \ll \text{STRD2})]) & = \{\text{XRa}[7:0], \text{XRa}[15:8], \text{XRa}[23:16], \text{XRa}[31:24]\};
\end{align*}
\]

Description:

\[
\begin{align*}
  \text{S32LDDVR:} & \quad \text{Load a word from the memory address formed by adding rb with rc<<STRD2, reverse the word’s byte sequence and subsequently update XRa with the reversed result.} \\
  \text{S32STDVR:} & \quad \text{Store a word to the memory address formed by adding rb with rc<<STRD2, reverse the the word’s byte sequence first.}
\end{align*}
\]

NOTE:

To avoid address error exception, the address formed must be 4-byte aligned.

Example:

\[
\begin{align*}
  \text{S32LDDVR} & \quad \text{XR11, $5, v0, 1} \\
  \text{S32STDVR} & \quad \text{XR3, a1, a2, 0}
\end{align*}
\]
1.5.4 S32LDDV/S32STDV

Syntax:

S32LDDV X Ra, rb, rc, STRD2
S32STDV X Ra, rb, rc, STRD2

Parameter:

STRD2: 0~2

Operation:

S32LDDV:
X Ra = (Mem32 [ rb + (rc << STRD2) ]); 

S32STDV:
(Mem32 [ rb + (rc << STRD2) ]) = X Ra;

Description:

S32LDDV: Load a word from the memory address formed by adding rb with rc<<STRD2.
S32STDV: Store a word to the memory address formed by adding rb with rc<<STRD2.

NOTE:

To avoid address error exception, the address formed must be 4-byte aligned.

Example:

S32LDDV XR11, $5, v0, 1
S32STDV XR3, a1, a2, 0
1.5.5 S32LDIR/S32SDIR

Syntax:

S32LDIR XRa, rb, S12  
S32SDIR XRa, rb, S12

Operation:

S32LDIR:

tmp32 = (Mem32 [rb + S12]);
XRa = {tmp32[7:0], tmp32[15:8], tmp32[23:16], tmp32[31:24]};
brb = rb + S12;

S32SDIR:

(Mem32 [rb + S12])
= {XRa[7:0], XRa[15:8], XRa[23:16], XRa[31:24]};
brb = rb + S12;

Description:

S32LDIR: same as S32LDDR in addition to rb is updated by the address formed.  
S32SDIR: same as S32STDR in addition to rb is updated by the address formed.

NOTE:

1. S12 should be the multiple of 4.
2. The offset range represented by S12 is –2048 ~ 2044.
3. To avoid address error exception, the address formed must be 4-byte aligned.

Example:

S32LDIR XR11, v0, 0x10  
S32SDIR XR3, k1, -0x50
1.5.6 S32LDI/S32SDI

Syntax:

S32LDI XRa, rb, S12
S32SDI XRa, rb, S12

Operation:

S32LDI:
XRa = (Mem32 [rb + S12]);
rb = rb + S12;

S32SDI:
(Mem32 [rb + S12]) = XRa;
rb = rb + S12;

Description:

S32LDI: same as S32LDD in addition to rb is updated by the address formed.
S32SDI: same as S32STD in addition to rb is updated by the address formed.

NOTE:

1. S12 should be the multiple of 4.
2. The offset range represented by S12 is –2048 ~ 2044.
3. To avoid address error exception, the address formed must be 4-byte aligned.

Example:

S32LDI XR11, v0, 0x10
S32SDI XR3, k1, -0x50
1.5.7 S32LDIVR/S32SDIVR

Syntax:

S32LDIVR  XRa, rb, rc, STRD2
S32SDIVR  XRa, rb, rc, STRD2

Parameter:

STRD2: 0~2

Operation:

S32LDIVR:

tmp32= (Mem32 [rb + (rc << STRD2)]);
XRa = {tmp32[7:0], tmp32[15:8], tmp32[23:16], tmp32[31:24]};
rb = rb + (rc << STRD2)

S32SDIVR:

(Mem32 [rb + (rc << STRD2)])
= {XRa[7:0], XRa[15:8], XRa[23:16], XRa[31:24]};
rb = rb + (rc << STRD2)

Description:

S32LDIVR: same as S32LDDVR in addition to rb is updated by the address formed.
S32SDIVR: same as S32STDVR in addition to rb is updated by the address formed.

NOTE:

To avoid address error exception, the address formed must be 4-byte aligned.

Example:

S32LDIVR  XR11, $5, v0, 1
S32SDIVR  XR3, a1, a2, 0
1.5.8 S32LDIV/S32SDIV

Syntax:

S32LDIV   XRa, rb, rc, STRD2
S32SDIV   XRa, rb, rc, STRD2

Parameter:

STRD2: 0~2

Operation:

S32LDIV:
XRa = (Mem32 [rb + (rc << STRD2)]);
rb = rb + (rc << STRD2)

S32SDIV:
(Mem32 [rb + (rc << STRD2)]) = XRa;
rb = rb + (rc << STRD2)

Description:

S32LDIV: same as S32LDDV in addition to rb is updated by the address formed.
S32SDIV: same as S32STDV in addition to rb is updated by the address formed.

NOTE:

To avoid address error exception, the address formed must be 4-byte aligned.

Example:

S32LDIV   XR11, $5, v0, 1
S32SDIV   XR3, a1, a2, 0
1.5.9  LXW

Syntax:

\[
\text{LXW} \quad \text{rd, rs, rt, STRD2}
\]

Parameter:

\[
\text{STRD2: 0-2}
\]

Operation:

\[
\text{rd} = \text{(Mem32 [rs + (rt \ll STRD2)])};
\]

Description:

Load a word from the memory address formed by adding rs with rt<<STRD2, update rd with that word

NOTE:

To avoid address error exception, the address formed must be 4-byte aligned.

Example:

\[
\text{LXW} \quad \$1, \$5, \text{v0, 1}
\]
1.5.10 S16LDD

Syntax:

```
S16LDD XRa, rb, S10, OPTN2
```

Parameter:

```
OPTN2: ptn0(000), ptn1(001), ptn2(010), ptn3(011)
```

Operation:

```
tmp16 = (Mem16 [rb + S10]);
switch(OPTN2) {
  case 0: XRa[15:00] = tmp16;
  case 1: XRa[31:16] = tmp16;
  case 2: XRa[31:00] = {{16{sign of tmp16}}, tmp16};
  case 3: XRa[31:00] = {tmp16, tmp16};
}
```

Description:

Load a half word from the memory address formed by adding rb with S10, permutate the half word in terms of specific pattern to form a word to update XRa

NOTE:

1. The valid offset range of S10 is –512 ~ 510, S10 must be multiple of 2
2. Only the specific half word position in XRa is updated for pattern 0~1
3. To avoid address error exception, the address formed must be 2-byte aligned

Example:

```
S16LDD XR11, $5, -512, 3
```
1.5.11 S16LDI

Syntax:

S16LDI XRa, rb, S10, OPTN2

Parameter:

OPTN2: ptn0(00), ptn1(01), ptn2(10), ptn3(11)

Operation:

tmp16 = (Mem16 [rb + S10]);
switch(OPTN2) {
    case 0: XRa[15:00] = tmp16;
    case 1: XRa[31:16] = tmp16;
    case 2: XRa[31:00] = {{16{sign of tmp16}}, tmp16};
    case 3: XRa[31:00] = {tmp16, tmp16};
}
rb = rb + S10;

Description:

Load a half word from the memory address formed by adding rb with S10, permutate the half word in terms of specific pattern to form a word to update XRa, meanwhile rb is updated by the address formed.

NOTE:

1. The valid offset range of S10 is –512 ~ 510, S10 must be multiple of 2
2. Only the specific half word position in XRa is updated for pattern 0~1
3. To avoid address error exception, the address formed must be 2-byte aligned

Example:

S16LDI XRx11, $5, 2, 1
1.5.12 S16STD

Syntax:

\[ \text{S16STD XRa, rb, S10, OPTN2} \]

Parameter:

\[ \text{OPTN2: ptn0(00), ptn1(01), 10~11 are reserved} \]

Operation:

\[
\begin{align*}
\text{switch(OPTN2) } & \{ \\
\text{case 0: tmp16} & = \text{XRa}[15:00] ; \\
\text{case 1: tmp16} & = \text{XRa}[31:16] ; \\
& \}
\end{align*}
\]

\[ \text{Mem16 [rb + S10]} = \text{tmp16} ; \]

Description:

Select a half word from XRa in terms of specific pattern, write the half word to the memory address formed by adding rb with S10.

NOTE:

1. The valid offset range of S10 is –512 ~ 510, S10 must be multiple of 2
2. To avoid address error exception, the address formed must be 2-byte aligned.

Example:

\[ \text{S16STD XR3, a1, -10, ptn0} \]
1.5.13 S16SDI

Syntax:

```
S16SDI XRa, rb, S10, OPTN2
```

Parameter:

```
OPTN2: ptn0(00), ptn1(01), 10~11 are reserved
```

Operation:

```
switch(OPTN2) {
  case 0: tmp16 = XRa[15:00];
  case 1: tmp16 = XRa[31:16];
}
(Mem16 [rb + S10]) = tmp16;
rb = rb + S10;
```

Description:

Select a half word from XRa in terms of specific pattern, write the half word to the memory address formed by adding rb with S10, meanwhile rb is updated by the address formed.

NOTE:

1. The valid offset range of S10 is –512 ~ 510, S10 must be multiple of 2
2. To avoid address error exception, the address formed must be 2-byte aligned.

Example:

```
S16SDI XR3, a1, 14, ptn0
```
1.5.14 LXH/LXHU

Syntax:

LXH    rd, rs, rt, STRD2
LXHU   rd, rs, rt, STRD2

Parameter:

STRD2: 0~2

Operation:

LXH:     rd= sign_ext32(Mem16 [rs + (rt << STRD2)]);
LXHU:    rd= zero_ext32(Mem16 [rs + (rt << STRD2)]);

Description:

Load a half word from the memory address formed by adding rs with rt<<STRD2, make
sign-extension or zero-extension of the half word to update rd.

NOTE:

To avoid address error exception, the address formed must be 2-byte aligned.

Example:

LXH  v1, $5, v0, 1
LXHU t0, t1, t2, 0
1.5.15 S8LDD

Syntax:

    S8LDD   XRa, rb, S8, OPTN3

Parameter:

    OPTN3: ptn0(000), ptn1(001), ptn2(010), ptn3(011), ptn4(100),
          ptn5(101), ptn6(110), ptn7(111)

Operation:

    tmp8 = (Mem8 \[rb + S8\]);
    switch(OPTN3) {
      case 0: XRa[7:0] = tmp8;
      case 1: XRa[15:8] = tmp8;
      case 2: XRa[23:16] = tmp8;
      case 3: XRa[31:24] = tmp8;
      case 4: XRa = {8'b0, tmp8, 8'b0, tmp8};
      case 5: XRa = {tmp8, 8'b0, tmp8, 8'b0};
      case 6: XRa = {{8{sign of tmp8}}, tmp8, {8{sign of tmp8}}, tmp8};
      case 7: XRa = {tmp8, tmp8, tmp8, tmp8};
    }

Description:

    Load a byte from the memory address formed by adding rb with S8, permutate the byte
    in terms of specific pattern to form a word to update XRa

NOTE:

    1. The offset range represented by S8 is –128 ~ 127.
    2. Only the specific byte position in XRa is updated for pattern 0~3

Example:

    S8LDD   XR11, $5, 1, 7
    S8LDD   XR3, a1, -17, ptn7
1.5.16 S8LDI

Syntax:

```
S8LDI XRa, rb, S8, OPTN3
```

Parameter:

```
OPTN3: ptn0(000), ptn1(001), ptn2(010), ptn3(011), ptn4(100),
       ptn5(101), ptn6(110), ptn7(111)
```

Operation:

```
tmp8 = (Mem8 [rb + S8]);
switch(OPTN3) {
    case 0: XRa[7:0] = tmp8;
    case 1: XRa[15:8] = tmp8;
    case 2: XRa[23:16] = tmp8;
    case 3: XRa[31:24] = tmp8;
    case 4: XRa = {8'b0, tmp8, 8'b0, tmp8};
    case 5: XRa = {tmp8, 8'b0, tmp8, 8'b0};
    case 6: XRa = {{8{sign of tmp8}}, tmp8, {8{sign of tmp8}}, tmp8};
    case 7: XRa = {tmp8, tmp8, tmp8, tmp8};
}
rb = rb + S8;
```

Description:

Load a byte from the memory address formed by adding rb with S8, permutate the byte in terms of specific pattern to form a word to update XRa, meanwhile rb is updated by the address formed.

NOTE:

1. The offset range represented by S8 is –128 ~ 127.
2. Only the specific byte position in XRa is updated for pattern 0~3

Example:

```
S8LDI XR11, $5, 13, 3
S8LDI XR3, a1, -1, ptn5
```
1.5.17 S8STD

Syntax:

S8STD XRa, rb, S8, OPTN3

Parameter:

OPTN3: ptn0(000), ptn1(001), ptn2(010), ptn3(011), 100~111 are reserved

Operation:

switch(OPTN3) {
  case 0: tmp8 = XRa[7:0];
  case 1: tmp8 = XRa[15:8];
  case 2: tmp8 = XRa[23:16];
  case 3: tmp8 = XRa[31:24];
}
(Mem8 [rb + S8]) = tmp8;

Description:

Select a byte from XRa in terms of specific pattern, write the byte to the memory address formed by adding rb with S8.

NOTE:

The offset range represented by S8 is –128 ~ 127.

Example:

S8STD XR11, $5, 1, 0
S8STD XR3, a1, -17, ptn7
1.5.18 S8SDI

Syntax:

\[
\text{S8SDI } XRa, \text{ rb, S8, OPTN3}
\]

Parameter:

\[
\text{OPTN3: ptn0(000), ptn1(001), ptn2(010), ptn3(011), 100-111 are reserved}
\]

Operation:

\[
\begin{align*}
\text{switch(OPTN3) } & \{
\text{case 0: tmp8 = XRa[7:0];}
\text{case 1: tmp8 = XRa[15:8];}
\text{case 2: tmp8 = XRa[23:16];}
\text{case 3: tmp8 = XRa[31:24];}
\}
(Mem8[rb + S8]) = \text{tmp8};
\text{rb = rb + S8;}
\end{align*}
\]

Description:

Select a byte from XRa in terms of specific pattern, write the byte to the memory address formed by adding rb with S8, meanwhile rb is updated by the address formed.

NOTE:

The offset range represented by S8 is –128 ~ 127.

Example:

\[
\begin{align*}
\text{S8SDI } XRI1, \text{ $5, 1, 4}
\text{S8SDI } XRI3, \text{ a1, -17, ptn3}
\end{align*}
\]
1.5.19 LXB/LXBU

Syntax:

LXB rd, rs, rt, STRD2
LXBU rd, rs, rt, STRD2

Parameter:

STRD2: 0-2

Operation:

LXB: rd = sign_ext32(Mem8 [rs + (rt << STRD2)]);
LXBU: rd = zero_ext32(Mem8 [rs + (rt << STRD2)]);

Description:

Load a byte from the address formed by adding rs with rt<<STRD2, make sign-extension or zero-extension of the byte to update rd.

Example:

LXB v1, $5, v0, 1
LXBU t0, t1, t2, 0
1.6 Multiplication with/without accumulation

1.6.1 S32MUL

Syntax:

\[ S32MUL \ XRa, \ XRd, \ rs, \ rt \]

Operation:

\[ \text{tmp64} = \text{signed}(rs) \times \text{signed}(rt); \]
\[ \text{XRa} = \text{tmp64}[63:32]; \]
\[ \text{XRd} = \text{tmp64}[31:00]; \]

Description:

Make signed 32x32 multiply for rs and rt to form a 64-bit product, the top 32-bit of the product updates XRa, while the bottom one updates XRd.

Example:

\[ S32MUL \ XR11, \ XR7, \ $1, \ v0 \]
\[ S32MUL \ XR0, \ XR4, \ $1, \ $3 \]
\[ S32MUL \ XR9, \ XR0, \ v0, \ t0 \]
1.6.2 S32MULU

Syntax:

\[ \text{S32MULU} \, XRa, \, XRd, \, rs, \, rt \]

Operation:

\[
\begin{align*}
\text{tmp64} &= \text{unsigh}(rs) \times \text{unsigh}(rt); \\
XRa &= \text{tmp64}[63:32]; \\
XRd &= \text{tmp64}[31:00];
\end{align*}
\]

Description:

Make unsigned 32x32 multiply for rs and rt to form a 64-bit product, the top 32-bit of the product updates XRa, while the bottom one updates XRd.

Example:

\[
\begin{align*}
\text{S32MULU} & \quad \text{XR11, XR7, } S1, \, v0 \\
\text{S32MULU} & \quad \text{XR0, XR4, } S1, \, S3 \\
\text{S32MULU} & \quad \text{XR9, XR0, } v0, \, t0
\end{align*}
\]
1.6.3 S32MADD

Syntax:

\[
\text{S32MADD XRa, XRd, rs, rt}
\]

Operation:

\[
\text{tmp64} = (\text{XRa}, \text{XRd}) + \text{signed}(\text{rs}) \times \text{signed}(\text{rt});
\]
\[
\text{XRa} = \text{tmp64}[63:32];
\]
\[
\text{XRd} = \text{tmp64}[31:00];
\]

Description:

Make signed 32x32 multiply for rs and rt to form a 64-bit product, accumulate the
product to the 64-bit value formed by binding XRa and XRd, the top 32-bit
accumulation updates XRa while the bottom one updates XRd.

NOTE:

The instruction’s executing will stain HI/LO, so take away available values in HI/LO
ahead of time.

Example:

\[
\begin{align*}
\text{S32MADD XR11, XR7, $1, v0} \\
\text{S32MADD XR0, XR4, $1, $3} \\
\text{S32MADD XR9, XR0, v0, t0}
\end{align*}
\]
1.6.4 S32MADDU

**Syntax:**

```
S32MADDU  XRa, XRd, rs, rt
```

**Operation:**

```
tmp64 = (XRa, XRd) + unsigned(rs) * unsigned(rt);
XRa = tmp64[63:32];
XRd = tmp64[31:00];
```

**Description:**

Make unsigned 32x32 multiply for rs and rt to form a 64-bit product, accumulate the product to the 64-bit value formed by binding XRa and XRd, the top 32-bit accumulation updates XRa while the bottom one updates XRd.

**NOTE:**

The instruction’s executing will stain HI/LO, so take away available values in HI/LO ahead of time.

**Example:**

```
S32MADDU  XR11, XR7, $1, v0
S32MADDU  XR0, XR4, $1, $3
S32MADDU  XR9, XR0, v0, t0
```
1.6.5 S32MSUB

Syntax:

    S32MSUB XRa, XRd, rs, rt

Operation:

    tmp64 = (XRa, XRd) - signed(rs) * signed(rt);
    XRa = tmp64[63:32];
    XRd = tmp64[31:00];

Description:

    Make signed 32x32 multiply for rs and rt to form a 64-bit product, Use the 64-bit value
    formed by binding XRa and XRd to subtract the product, the top 32-bit of the minus
    result updates XRa while the bottom one updates XRd.

NOTE:

    The instruction’s executing will stain HI/LO, so take away available values in HI/LO
    ahead of time.

Example:

    S32MSUB XR11, XR7, $1, v0
    S32MSUB XR0, XR4, $1, $3
    S32MSUB XR9, XR0, v0, t0
1.6.6 S32MSUBU

Syntax:

\[ \text{S32MSUBU XRa, XRd, rs, rt} \]

Operation:

\[
\text{tmp64} = (\text{XRa, XRd}) - \text{unsigned}(\text{rs}) \times \text{unsigned}(\text{rt});
\]
\[
\text{XRa} = \text{tmp64}[63:32];
\]
\[
\text{XRd} = \text{tmp64}[31:00];
\]

Description:

Make unsigned 32x32 multiply for rs and rt to form a 64-bit product, Use the 64-bit value formed by binding XRa and XRd to subtract the product, the top 32-bit of the minus result updates XRa while the bottom one updates XRd.

NOTE:

The instruction's executing will stain HI/LO, so take away available values in HI/LO ahead of time.

Example:

\[
\text{S32MSUBU XR11, XR7, $1, v0}
\]
\[
\text{S32MSUBU XR0, XR4, $1, $3}
\]
\[
\text{S32MSUBU XR9, XR0, v0, t0}
\]
1.6.7 D16MUL

Syntax:

D16MUL XRa, XRb, X Rc, XRd, OPTN2

Parameter:

OPTN2: WW(00), LW(01), HW(10), XW(11);

Operation:

\[(L32, R32) = \text{pattern\_MUL}(XRb, X Rc);\]
XRa=L32;
XRd=R32;

Description:

Make \text{pattern\_MUL} multiply for XRb and X Rc according to the operand combinatorial pattern. The left lane product updates XRa, the right lane one updates XRd.

Example:

D16MUL XR1, XR3, XR1, XR7, XW
D16MUL XR6, XR8, XR7, XR7, 0
D16MUL XR6, XR8, XR7, XR7, 2
1.6.8  D16MULF

Syntax:

\[ \text{D16MULFXRa, XRb, X Rc, OPTN2} \]

Parameter:

OPTN2: WW(00), LW(01), HW(10), XW(11);

Operation:

\[ \{L32, R32\} = \text{pattern} \_\text{MUL}(XRb, XRc); \]
\[ \{L32, R32\} = \{L32<<1, R32<<1\}; \]

If (MXU\_CR.RD\_EN)
\[ \{L32, R32\} = \{\text{round}(L32), \text{round}(R32)\}; \]
\[ \text{XRa} = \{L32[31:16], R32[31:16]\}; \]

Description:

Make fractional pattern\_MUL multiply for XRb and XRc according to the operand combinatorial pattern with rounding or not. Pack the two products' top half-word to XRa.

Example:

D16MULF XR1, XR3, XR1, XW
D16MULF XR6, XR8, XR7, 1
1.6.9 D16MULE

Syntax:

D16MULE XRa, XRb, X Rc, XRd, OPTN2

Parameter:

OPTN2: WW(00), LW(01), HW(10), XW(11);

Operation:

\{
L32, R32 \} = \text{pattern}_\text{MUL}(XRb, XRc);
\{
L32, R32 \} = \{L32<<1, R32<<1\};
If (MXU_CR.RD_EN) \{
L32, R32 \} = \{\text{round}(L32), \text{round}(R32)\};
XRa = L32;
XRd = R32;

Description:

Make fractional pattern_MUL multiply for XRb and XRc according to the operand
combinatorial pattern with \text{rounding or not}. The left lane result updates XRa, the right
lane one updates XRd.

Example:

D16MULE XR1, XR3, XR1, XR9, XW
D16MULE XR8, XR8, XR7, XR0, 3
1.6.10 D16MAC

Syntax:

\[ \text{D16MAC} \ XRa, XRb, X Rc, XRd, APTN2, OPTN2 \]

Parameter:

APTN2: AA(00), AS(01), SA(10), SS(11);
OPTN2: WW(00), LW(01), HW(10), XW(11);

Operation:

\[ \{L32, R32\} = \text{pattern}_\text{MUL}(XRb, X Rc); \]
\[ XRa = XRa +/- L32; \]
\[ XRd = XRd +/- R32; \]

Description:

Make \text{pattern}_\text{MUL} multiply for XRb and X Rc according to the operand combinatorial pattern. X Ra subsequently adds or subtracts (directed by left A|S) the left lane product to update itself, while XRd subsequently adds or subtracts (directed by right A|S) the right lane one to update itself.

Example:

\[ \text{D16MAC} \ XR1, XR3, XR1, XR9, SA, XW \]
\[ \text{D16MAC} \ XR3, XR8, XR7, XR4, 3, 2 \]
1.6.11 D16MACF

Syntax:

D16MACF XRa, XRx, XRC, XRd, APTN2, OPTN2

Parameter:

APTN2: AA(00), AS(01), SA(10), SS(11);
OPTN2: WW(00), LW(01), HW(10), XW(11);

Operation:

\{L32, R32\} = \text{pattern}_\text{MUL}(XRb, XRc);
\{L32, R32\} = \{L32<<1, R32<<1\}
L32 = XRa +/- L32;
R32 = XRd +/- R32;
If (MXU_CR.RD_EN)
\{L32, R32\} = \{\text{round}(L32), \text{round}(R32)\}
XRa = \{L32[31:16], R32[31:16]\}

Description:

Make fractional \text{pattern}_\text{MUL} multiply for XRb and XRc according to the operand combinatorial pattern. XRa subsequently adds or subtracts (directed by left A|S) the left lane result, while XRd subsequently adds or subtracts (directed by right A|S) the right lane one, round the two sum results if necessray, finally, pack the two results' top half-word to XRa.

Example:

D16MACF XR1, XR3, XR1, XR9, SA, HW
D16MACF XR3, XR8, XR7, XR4, 2, 0
1.6.12 D16MACE

Syntax:

\[ \text{D16MACF} \ XRa, \ XRb, \ X Rc, \ XRd, \ APTN2, \ OPTN2 \]

Parameter:

\[ \begin{align*}
\text{APTN2:} & \quad \text{AA(00), AS(01), SA(10), SS(11)}; \\
\text{OPTN2:} & \quad \text{WW(00), LW(01), HW(10), XW(11)};
\end{align*} \]

Operation:

\[ \begin{align*}
\{L32, R32\} &= \text{pattern}_\text{MUL}(XRb, XRc); \\
\{L32, R32\} &= \{L32<<1, R32<<1\} \\
L32 &= XRa +/- L32; \\
R32 &= XRd +/- R32; \\
\text{If (MXU CR.RD EN)} & \\
& \quad \{L32, R32\} = \{\text{round}(L32), \text{round}(R32)\} \\
XRa &= L32; \\
XRd &= R32;
\end{align*} \]

Description:

Make fractional \text{pattern}_\text{MUL} multiply for XRb and XRc according to the operand combinatorial pattern. XRa subsequently adds or subtracts (directed by left A|S) the left lane result, while XRd subsequently adds or subtracts (directed by right A|S) the right lane one, round the two sum results if necessray, updates XRa with the left result while XRd with the right one.

Example:

\[ \begin{align*}
\text{D16MACE} & \quad XR1, \ XR3, \ XR1, \ XR9, \ SA, \ XW \\
\text{D16MACE} & \quad XR3, \ XR8, \ XR7, \ XR4, \ 1, \ 3
\end{align*} \]
1.6.13 D16MADL

Syntax:

\[
\text{D16MADL} \quad \text{XRa, XRb, XC, XRd, APTN2, OPTN2}
\]

Parameter:

\[
\text{APTN2: } \text{AA(00), AS(01), SA(10), SS(11)}; \\
\text{OPTN2: } \text{WW(00), LW(01), HW(10), XW(11)};
\]

Operation:

\[
\{\text{L32, R32}\} = \text{pattern_MUL(XRb, XC)}; \\
\text{Short1} = \text{XRa}[31:16] +/\!- \text{L32}[15:00]; \\
\text{Short0} = \text{XRa}[15:00] +/\!- \text{R32}[15:00]; \\
\text{XRd} = \{\text{Short1, Short0}\}
\]

Description:

make \text{pattern_MUL} multiply for XRb and XC according to the operand combinatorial pattern. Subsequently add/subtract two products' bottom half-words with the two 16-bit portions in XRa respectively. Finally, Pack two 16-bit sum results to XRd.

Example:

\[
\text{D16MADL} \quad \text{XR1, XR3, XR1, XR9, SA, XW} \\
\text{D16MADL} \quad \text{XR3, XR8, XR7, XR4, 0, 0}
\]
1.6.14 S16MAD

Syntax:

\[ \text{S16MAD} \ X^Ra, X^Rb, X^Rc, X^Rd, \text{APT}^N1, \text{OPT}^N1 \]

Parameter:

\[ \text{APT}^N1: A(0), S(1); \]
\[ \text{OPT}^N1: 00-\text{XRb.H*XRc.H}, 01-\text{XRb.L*XRc.L}, 10-\text{XRb.H*XRc.L}, 11-\text{XRb.L*XRc.H}; \]

Operation:

\[ \text{If (OPTN2 == 0)} \ X^Rd = X^Ra +/- X^Rb.H \times X^Rc.H; \]
\[ \text{If (OPTN2 == 1)} \ X^Rd = X^Ra +/- X^Rb.L \times X^Rc.L; \]
\[ \text{If (OPTN2 == 2)} \ X^Rd = X^Ra +/- X^Rb.H \times X^Rc.L; \]
\[ \text{If (OPTN2 == 3)} \ X^Rd = X^Ra +/- X^Rb.L \times X^Rc.H; \]

Description:

Make 16x16 multiply for X^Rb and X^Rc according to the operand combinatorial pattern. X^Ra subsequently adds or subtracts the product to update X^Rd.

Example:

\[ \text{S16MAD} \ X^R1, X^R3, X^R1, X^R9, S, 0 \]
\[ \text{S16MAD} \ X^R3, X^R8, X^R7, X^R4, 0, 2 \]
1.6.15 Q8MUL

Syntax:

Q8MUL XRa, XRb, XRc, XRd

Operation:

Short3 = unsigned(XRb[31:24]) * unsigned(XRc[31:24]);
Short2 = unsigned(XRb[23:16]) * unsigned(XRc[23:16]);
Short1 = unsigned(XRb[15:08]) * unsigned(XRc[15:08]);
Short0 = unsigned(XRb[07:00]) * unsigned(XRc[07:00]);
XRa = {Short3, Short2};
XRd = {Short1, Short0};

Description:

Make four parallel unsigned 8x8 multiply for four corresponding byte pairs in XRb and XRc, two 16-bit products generated from the operand’s MSB position update XRa, the other two from the operand’s LSB position update XRd.

Example:

Q8MUL XR1, XR3, XR4, XR9
1.6.16 Q8MULSU

Syntax:

Q8MULSU XRa, XRb, XRc, XRd

Parameter:

Operation:

Short3 = signed(XRb[31:24]) * unsigned(XRc[31:24]);
Short2 = signed(XRb[23:16]) * unsigned(XRc[23:16]);
Short1 = signed(XRb[15:08]) * unsigned(XRc[15:08]);
Short0 = signed(XRb[07:00]) * unsigned(XRc[07:00]);
XRa = {Short3, Short2};
XRd = {Short1, Short0};

Description:

Make four parallel signed 8x8 multiply for four corresponding byte pairs in XRb (contains four signed bytes) and XRc (contains four unsigned bytes), two 16-bit products generated from the operand’s MSB position update XRa, the other two from the operand’s LSB position update XRd.

Example:

Q8MULSU XR1, XR3, XR4, XR9
1.6.17 Q8MAC

Syntax:

Q8MAC XRa, XRb, X Rc, XRd, APTN2

Parameter:

APTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

Short3 = X Ra[31:16] +/- unsigned(XRb[31:24]) * unsigned(XRc[31:24]);
Short2 = X Ra[15:00] +/- unsigned(XRb[23:16]) * unsigned(XRc[23:16]);
Short1 = XRd[31:16] +/- unsigned(XRb[15:08]) * unsigned(XRc[15:08]);
Short0 = XRd[15:00] +/- unsigned(XRb[07:00]) * unsigned(XRc[07:00]);
X Ra = {Short3, Short2};
XRd = {Short1, Short0};

Description:

Make four parallel unsigned 8x8 multiply for four corresponding byte pairs in XRb and
XRc, two 16-bit products generated from the operand’s MSB position accumulate to
X Ra which contains two packed 16-bits, the other two from the operand’s LSB position
accumulate to XRd which contains two packed 16-bits.

Note that the left A|S directs the add/subtract operation of Short3 and Short2 while the
right one for Short1 and Short0.

Example:

Q8MAC XR1, XR3, XR4, XR9, AA
### 1.6.18 Q8MACSU

**Syntax:**

Q8MACSU XRa, XRb, X Rc, XRd, APTN2

**Parameter:**

APTN2: AA(00), AS(01), SA(10), SS(11);

**Operation:**

\[
\begin{align*}
\text{Short3} &= \text{XRa}[31:16] +/\text{signed}(\text{XRb}[31:24]) \times \text{unsigned}(\text{X Rc}[31:24]); \\
\text{Short2} &= \text{XRa}[15:00] +/\text{signed}(\text{XRb}[23:16]) \times \text{unsigned}(\text{X Rc}[23:16]); \\
\text{Short1} &= \text{XRd}[31:16] +/\text{signed}(\text{XRb}[15:08]) \times \text{unsigned}(\text{X Rc}[15:08]); \\
\text{Short0} &= \text{XRd}[15:00] +/\text{signed}(\text{XRb}[07:00]) \times \text{unsigned}(\text{X Rc}[07:00]); \\
\text{XRa} &= \{\text{Short3}, \text{Short2}\}; \\
\text{XRd} &= \{\text{Short1}, \text{Short0}\};
\end{align*}
\]

**Description:**

Make four parallel signed 8x8 multiply for four corresponding byte pairs in XRb (contains four signed bytes) and XRc (contains four unsigned bytes), two 16-bit products generated from the operand’s MSB position accumulate to X Ra which contains two packed 16-bits, the other two from the operand’s LSB position accumulate to XRd which contains two packed 16-bits. Note that left A|S directs the add/subtract operation of Short3 and Short2 while the right one for Short1 and Short0.

**Example:**

Q8MACSU XR1, XR3, XR4, XR9, AA
1.6.19 Q8MADL (obsolete)

Syntax:

Q8MADL XRa, XRb, X Rc, XRd, APTN2

Parameter:

APTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

XRd[31:24] = X Ra[31:24] +/- trunc_8(unsigned(XRb[31:24]) * unsigned(XRc[31:24]));
XRd[15:08] = X Ra[15:08] +/- trunc_8(unsigned(XRb[15:08]) * unsigned(XRc[15:08]));
XRd[07:00] = X Ra[07:00] +/- trunc_8(unsigned(XRb[07:00]) * unsigned(XRc[07:00]));

Description:

Make four parallel unsigned 8x8 multiply for four corresponding byte pairs in XRb and XRc, add each lower 8-bit of the four 16-bit products to the corresponding one in XRa, respectively. The final four 8-bit sum results update XRd.
Note that left A|S directs the add/subtract operation of bit [31:24] and [23:16] while the right one for [15:08] and [07:00].

NOTE:

It is not recommended to use this instruction for it may be discarded in the future.
1.7 Add and subtract

1.7.1 D32ADD

Syntax:

\[ \text{D32ADD XRa, XRb, XRc, XRd, APTN2} \]

Parameter:

\[ \text{APTN2: AA(00), AS(01), SA(10), SS(11);} \]

Operation:

\[
\begin{align*}
\text{Int1} &= \text{XRb} +/- \text{XRc}; \\
\text{Int0} &= \text{XRb} +/- \text{XRc}; \\
\text{XRa} &= \text{Int1}; \\
\text{XRd} &= \text{Int0};
\end{align*}
\]

Description:

Make dual 32-bit add/subtract operations for XRb and XRc, the dual results update XRa and XRd respectively. Note that left A|S directs add/subtract operation of Int1, while the right one for Int0.

NOTE:

The carry-out of the adder for Int1 updates the LC field of MXU_CR when XRa !\= XR0, the carry-out of the adder for Int0 updates the RC field of MXU_CR when XRd !\= XR0.

Example:

\[ \text{D32ADD XR3, XR2, XR1, XR4, SA} \]
1.7.2 D32ADDC

Syntax:

D32ADDC XRa, XRb, XRc, XRd

Operation:

XRa += XRb + LC;
XRd += XRc + RC;

Description:

Adding XRa and XRb together with LC to update XRa, adding XRd and XRc together with RC to update XRd.

NOTE:

The contents of LC and RC fields of MXU_CR retain until an executing of a D32ADD instruction.

Example:

D32ADDC XR3, XR2, XR1, XR4
1.7.3  D32ACC

Syntax:

D32ACC XRa, XRb, XRc, XRd, APTN2

Parameter:

APTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

Int1 = XRb +/- XRc;
Int0 = XRb +/- XRc;
XRa += Int1;
XRd += Int0;

Description:

Make dual 32-bit add/subtract operations for XRb and XRc, the dual results are added to XRa and XRd respectively. Note that left A|S directs add/subtract operation of Int1, while the right one for Int0.

Example:

D32ACC XR3, XR2, XR1, XR4, SA
1.7.4 D32ACCM

Syntax:

D32ACCM XRa, XRb, XRc, XRd, APTN2

Parameter:

APTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

XRa +/-= (XRb + XRc);
XRd +/-= (XRb - XRc);

Description:

XRa adds or subtracts (directed by left A|S) the adding result of XRb and XRc to update XRa, meanwhile XRd adds or subtracts (directed by right A|S) the subtracting result of XRb and XRc to update XRd.

Example:

D32ACCM XR3, XR2, XR1, XR4, SA
1.7.5 D32ASUM

Syntax:

D32ASUM XRa, XRb, X Rc, XRd, APTN2

Parameter:

APTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

XRa +/-= XRb;
XRd +/-= XRc;

Description:

XRa adds or subtracts (directed by left A|S) XRb to update XRa, XRd adds or subtracts (directed by right A|S) XRc to update XRd.

Example:

D32ASUM XR3, XR2, XR1, XR4, SA
1.7.6 S32CPS

Syntax:

S32CPS  XRa, XRb, XRc

Operation:

If (XRc < 0) XRa = 0 – XRb;
Else XRa = XRb;

Description:

Get negative XRb or original XRb according to the sign of XRc to update XRa.

Example:

S32CPS  XR2, XR1, XR3
1.7.7 S32SLT

Syntax:

S32SLT XRa, XRb, XRc

Operation:

XRa = signed(XRb) < signed(XRc)?1:0;

Description:

Update XRa by the bool result of the signed 32-bit less-than comparison

Example:

S32SLT XR3, XR2, XR1
1.7.8 S32MOVZ

Syntax:

\[ \text{S32MOVZ } \text{XR}_a, \text{XR}_b, \text{XR}_c \]

Operation:

If \( \text{XR}_b == 0 \)
\[ \text{XR}_a = \text{XR}_c; \]

Description:

Conditional move \( \text{XR}_c \) to \( \text{XR}_a \) if \( \text{XR}_b \) equals zero.

Example:

\[ \text{S32MOVZ } \text{XR}_3, \text{XR}_2, \text{XR}_1 \]
1.7.9 S32MOVN

Syntax:

S32MOVN XRa, XRb, X Rc

Operation:

If (XRb != 0)
    X Ra = X Rc;

Description:

Conditional move X Rc to X Ra if XRb does not equal zero.

Example:

S32MOVN XR3, XR2, XR1
1.7.10 Q16ADD

Syntax:

Q16ADD XRa, XRb, XRc, XRd, EPTN2, OPTN2

Parameter:

EPTN2: AA(00), AS(01), SA(10), SS(11);
OPTN2: WW(00), LW(01), HW(10), XW(11);

Operation:

{short3, short2} = pattern_SUM(XRb, XRc);
{short1, short0} = pattern_SUM(XRb, XRc);
XRa = {short3, short2};
XRd = {short1, short0};

Description:

Make two parallel pattern_SUM operations to update XRa and XRd. It is determined by
OPTN2 that how the operands (XRb and XRc) are assembled. Note that the left A|S
directs the add/subtract operation of short3 and short2, while the right one for short1
and short0.

Example:

Q16ADD XR3, XR2, XR1, XR4, SA, WX
1.7.11 Q16ACC

Syntax:

Q16ACC XRa, XRb, X Rc, X Rd, EPTN2

Parameter:

EPTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

Short3 = XRb[31:16] +/- X Rc[31:16];
Short2 = XRb[15:00] +/- X Rc[15:00];
Short1 = XRb[31:16] +/- X Rc[31:16];
Short0 = XRb[15:00] +/- X Rc[15:00];
XRa[31:16] += Short3;
XRa[15:00] += Short2;
XRd[31:16] += Short1;
XRd[15:00] += Short0;

Description:

As above expressions. Note that the left A|S directs the sum/difference operation of short3 and short2, while the right one for short1 and short0.

Example:

Q16ACC XR3, XR2, XR1, XR4, SA
1.7.12 Q16ACCM

Syntax:

Q16ACCM XRa, XRb, X Rc, XRd, EPTN2

Parameter:

EPTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

Short3 = XRb[31:16];
Short2 = XRb[15:00];
Short1 = XRc[31:16];
Short0 = XRc[15:00];
XRa[31:16] +/-= Short3;
XRa[15:00] +/-= Short2;
XRd[31:16] +/-= Short1;
XRd[15:00] +/-= Short0;

Description:

Two packed 16-bits in XRb are added or subtracted (directed by left A|S) to corresponding ones in X Ra, respectively.
Two packed 16-bits in XRc are added or subtracted (directed by left A|S) to corresponding ones in XRd, respectively.

Example:

Q16ACCM XR3, XR2, XR1, XR4, SA
1.7.13 D16ASUM

Syntax:

D16ASUM XRa, XRb, X Rc, XRd, EPTN2

Parameter:

EPTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

L32 = sign_ext32(XRb[31:16]) + sign_ext32(XRb[15:00]);
R32 = sign_ext32(XRc[31:16]) + sign_ext32(XRc[15:00]);
XRa +/-= L32;
XRd +/-= R32;

Description:

Make sign_ext32 for two 16-bits in XRb and subsequently add them together to form a 32-bit result to be added or subtracted to XRa (directed by left A|S), same manipulation for XRc and XRd controlled by right A|S.

Example:

D16ASUM XR3, XR2, XR1, XR4, SA
1.7.14 D16CPS

Syntax:

\texttt{D16CPS XRa, XRb, XRc}

Operation:

\texttt{If (XRc[31:16]<0) short1 = 0-XRb[31:16]; \hspace{1cm} else short1 = XRb[31:16]}
\texttt{If (XRc[15:0]<0) short0 = 0-XRb[15:0]; \hspace{1cm} else short0 = XRb[15:0]}
\texttt{XRa = \{short1, short0\};}

Description:

It is a dual 16-bit packed operation. For each 16-bit portion, get negative value or original one in XRb according to the corresponding 16-bit value’s sign in XRc to update corresponding 16-bit portion in XRa.

Example:

\texttt{D16CPS XR2, XR1, XR3}
1.7.15 D16SLT

Syntax:

D16SLT XRa, XRb, XRc

Operation:

XRa[31:16] = signed(XRb[31:16])< signed(XRc[31:16])?1:0;
XRa[15:00] = signed(XRb[15:00])< signed(XRc[15:00])?1:0;

Description:

It is a dual 16-bit packed operation. For each 16-bit portion, make signed 16-bit less-than comparison of XRb and XRc to get 16-bit bool result to update corresponding 16-bit portion in XRa.

Example:

D16SLT XR3, XR2, XR1
1.7.16 D16MOVZ

Syntax:

\[
\text{D16MOVZ } \text{XRa, XRb, XRc}
\]

Operation:

If (XRb[31:16] == 0)
\[
\text{XRa}[31:16] = \text{XRc}[31:16];
\]
If (XRb[15:00] == 0)
\[
\text{XRa}[15:00] = \text{XRc}[15:00];
\]

Description:

It is a dual 16-bit packed operation. For each 16-bit portion, conditional move 16-bit value of XRc to corresponding 16-bit portion in XRa if corresponding 16-it value of XRb equals zero.

Example:

\[
\text{D16MOVZ } \text{XR3, XR2, XR1}
\]
1.7.17 D16MOVN

Syntax:

D16MOVN XRa, XRb, XRc

Operation:

If (XRb[31:16] != 0)
  XRa[31:16] = XRc[31:16];
If (XRb[15:00] != 0)
  XRa[15:00] = XRc[15:00];

Description:

It is a dual 16-bit packed operation. For each 16-bit portion, conditional move 16-bit value of XRc to corresponding 16-bit portion in XRa if corresponding 16-bit value of XRb equals non-zero.

Example:

D16MOVN XR3, XR2, XR1
1.7.18 D16AVG

Syntax:

D16AVG XRa, XRb, XRc

Operation:

Short1 = (XRb[31:16] + XRc[31:16]) >> 1;
Short0 = (XRb[15:00] + XRc[15:00]) >> 1;
XRa = {Short1, Short0};

Description:

Make dual 16-bit average operations for XRb (contain two packed 16-bits) and XRc (contain two packed 16-bits), pack two 16-bit average results to XRa.

Example:

D16AVG XR3, XR2, XR1
1.7.19 D16AVGR

Syntax:

D16AVGR XRa, XRb, XRc

Operation:

Short1 = (XRb[31:16] + XRc[31:16] + 1) >> 1;
Short0 = (XRb[15:00] + XRc[15:00] + 1) >> 1;
XRa = {Short1, Short0};

Description:

Make dual 16-bit average with explicit rounding operations for XRb (contain two packed 16-bits) and XRc (contain two packed 16-bits), pack two 16-bit average results to XRa.

Example:

D16AVGR XR3, XR2, XR1
1.7.20 Q8ADD (obsolete)

Syntax:

Q8ADD XRa, XRb, X Rc, EPTN2

Parameter:

EPTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

Byte3 = XRb[31:24] +/- XRc[31:24];
Byte2 = XRb[23:16] +/- XRc[23:16];
Byte1 = XRb[15:08] +/- XRc[15:08];
Byte0 = XRb[07:00] +/- XRc[07:00];
XRa = {Byte3, Byte2, Byte1, Byte0};

Description:

Make quadruple add/subtract operations for XRb (contain four packed 8-bits) and XRc (contain four packed 8-bits), pack four 8-bit sum results to XRa. Note that the left A|S directs the sum/difference operation of byte3 and byte2, while the right one for byte1 and byte0.

NOTE:

It is not recommended to use this instruction for it may be discarded in the future.

Example:

Q8ADD XR3, XR2, XR1, AA
1.7.21 Q8ADDE

Syntax:

\[ \text{Q8ADDE } \text{XRa}, \text{XRb}, \text{XRc}, \text{XRd}, \text{EPTN2} \]

Parameter:

\[ \text{EPTN2: AA(00), AS(01), SA(10), SS(11)}; \]

Operation:

\[ \begin{align*}
\text{XRa}[31:16] &= \text{zero\_ext16}(\text{XRb}[31:24]) +/\text{zero\_ext16}(\text{XRc}[31:24]); \\
\text{XRa}[15:00] &= \text{zero\_ext16}(\text{XRb}[23:16]) +/\text{zero\_ext16}(\text{XRc}[23:16]); \\
\text{XRd}[31:16] &= \text{zero\_ext16}(\text{XRb}[15:08]) +/\text{zero\_ext16}(\text{XRc}[15:08]); \\
\text{XRd}[15:00] &= \text{zero\_ext16}(\text{XRb}[07:00]) +/\text{zero\_ext16}(\text{XRc}[07:00]);
\end{align*} \]

Description:

It is a quadruple packed 8-bit operation. For each 8-bit portion, zero extends unsigned byte in XRb and XRc to 16-bit to make 16-bit add/subtract operation. Note that the left A|S denotes the dual 16-bit plus(A)/minus(S) operations which results update XRa while the right one for XRd.

Example:

\[ \text{Q8ADDE } \text{XR3}, \text{XR2}, \text{XR1}, \text{XR4}, \text{AS} \]
1.7.22 Q8ACCE

Syntax:

Q8ACCE XRa, XRb, X Rc, XRd, EPTN2

Parameter:

EPTN2: AA(00), AS(01), SA(10), SS(11);

Operation:

XRa[31:16] += zero_ext16(XRb[31:24]) +/- zero_ext16(XRc[31:24]);
XRa[15:00] += zero_ext16(XRb[23:16]) +/- zero_ext16(XRc[23:16]);
XRd[31:16] += zero_ext16(XRb[15:08]) +/- zero_ext16(XRc[15:08]);
XRd[15:00] += zero_ext16(XRb[07:00]) +/- zero_ext16(XRc[07:00]);

Description:

It is a quadruple packed 8-bit operation. For each 8-bit portion, zero extends unsigned byte in XRb and X Rc to 16-bit to make 16-bit add/subtract operation. Note that the left A|S denotes the dual 16-bit plus(A)/minus(S) operations which results are accumulated to XRa while the right one for XRd.

Example:

Q8ACCE XR3, XR2, XR1, XR4, AS
1.7.23 D8SUM

Syntax:

```
D8SUM  XRa, XRx, XRx
```

Operation:

```
XRa[31:16] = zero_ext16(XRb[31:24]) + zero_ext16(XRb[23:16]) +
zero_ext16(XRb[15:8]) + zero_ext16(XRb[7:0]);
XRa[15:0] = zero_ext16(XRc[31:24]) + zero_ext16(XRc[23:16]) +
zero_ext16(XRc[15:8]) + zero_ext16(XRc[7:0]);
```

Description:

Add four unsigned bytes of XRb together to update the upper 16-bit portion of XRa, add four unsigned bytes of XCc together to update the lower 16-bit portion of XRa.

Example:

```
D8SUM  XR3, XRx, XRx
```
1.7.24 D8SUMC

Syntax:

\[ \text{D8SUMC} \ X Ra, \ X Rb, \ X Rc \]

Operation:

\[
\begin{align*}
X Ra[31:16] &= (\text{zero\_ext16}(X Rb[31:24]) + \text{zero\_ext16}(X Rb[23:16]) + 1) \\
& \quad + (\text{zero\_ext16}(X Rb[15:8]) + \text{zero\_ext16}(X Rb[7:0]) + 1); \\
X Ra[15:0] &= (\text{zero\_ext16}(X Rc[31:24]) + \text{zero\_ext16}(X Rc[23:16]) + 1) \\
& \quad + (\text{zero\_ext16}(X Rc[15:8]) + \text{zero\_ext16}(X Rc[7:0]) + 1);
\end{align*}
\]

Description:

Add four unsigned bytes of X Rb together first and subsequently add value 2 to update the upper 16-bit portion of X Ra,
add four unsigned bytes of X Rc together first and subsequently add value 2 to update the lower 16-bit portion of X Ra.

Example:

\[ \text{D8SUMC} \ X R3, \ X R2, \ X R1 \]
1.7.25 Q8ABD

Syntax:

Q8ABD XRa, XRb, XRc

Operation:

XRa[31:24] = abs(unsigned(XRb[31:24]) - unsigned(XRc[31:24]));
XRa[23:16] = abs(unsigned(XRb[23:16]) - unsigned(XRc[23:16]));
XRa[15:08] = abs(unsigned(XRb[15:08]) - unsigned(XRc[15:08]));
XRa[07:00] = abs(unsigned(XRb[07:00]) - unsigned(XRc[07:00]));

Description:

Update XRa with four packed 8-bit absolute differences calculated from four corresponding 8-bit pairs in XRb and XRc

Example:

Q8ABD XR3, XR2, XR1
1.7.26 Q8SLT

Syntax:

Q8SLT XRa, XRb, XRc

Operation:

XRa[31:24] = sign_ext16(XRb[31:24])< sign_ext16(XRc[31:24])?1:0;
XRa[23:16] = sign_ext16(XRb[23:16])< sign_ext16(XRc[23:16])?1:0;
XRa[15:08] = sign_ext16(XRb[15:08])< sign_ext16(XRc[15:08])?1:0;
XRa[07:00] = sign_ext16(XRb[07:00])< sign_ext16(XRc[07:00])?1:0;

Description:

Update XRa with quadruple signed less-than compare results calculated from XRb (contains four 8-bit values) and XRc (contains four 8-bit values)

Example:

Q8SLT XR3, XR2, XR1
1.7.27 Q8SLTU

Syntax:

Q8SLTU XRa, XRb, XRc

Operation:

XRa[31:24] = zero_ext16(XRb[31:24]) < zero_ext16(XRc[31:24]) ? 1:0;
XRa[23:16] = zero_ext16(XRb[23:16]) < zero_ext16(XRc[23:16]) ? 1:0;
XRa[15:08] = zero_ext16(XRb[15:08]) < zero_ext16(XRc[15:08]) ? 1:0;
XRa[07:00] = zero_ext16(XRb[07:00]) < zero_ext16(XRc[07:00]) ? 1:0;

Description:

Update XRa with quadruple unsigned less-than compare results calculated from XRb (contains four 8-bit values) and XRc (contains four 8-bit values)

Example:

Q8SLTU XR3, XR2, XR1
1.7.28 Q8MOVZ

Syntax:

Q8MOVZ XRa, XRb, XRc

Operation:

If (XRb[31:24]== 0)
  XRa[31:24] = XRc[31:24];
If (XRb[23:16]== 0)
  XRa[23:16] = XRc[23:16];
If (XRb[15:08]== 0)
  XRa[15:08] = XRc[15:08];
If (XRb[07:00]== 0)
  XRa[07:00] = XRc[07:00];

Description:

It is a quadruple 8-bit packed operation. For each 8-bit portion, conditional move 8-bit value of XRc to corresponding 8-bit portion in XRa if corresponding 8-bit value of XRb equals zero.

Example:

Q8MOVZ XR3, XR2, XR1
1.7.29 Q8MOVN

Syntax:

Q8MOVN XRa, XRb, X Rc

Operation:

If (XRb[31:24]!= 0)
   XRa[31:24] = X Rc[31:24];
If (XRb[23:16]!= 0)
   XRa[23:16] = X Rc[23:16];
If (XRb[15:08]!= 0)
   XRa[15:08] = X Rc[15:08];
If (XRb[07:00]!= 0)
   XRa[07:00] = X Rc[07:00];

Description:

It is a quadruple 8-bit packed operation. For each 8-bit portion, conditional move 8-bit value of X Rc to corresponding 8-bit portion in X Ra if corresponding 8-bit value of XRb equals non-zero.

Example:

Q8MOVN XR3, XR2, XR1
1.7.30 Q8SAD

Syntax:

Q8SAD  XRa, XRb, X Rc, XRd

Operation:

int3 = abs(zero_ext16(XRb[31:24]) - zero_ext16(XRc[31:24]));
int2 = abs(zero_ext16(XRb[23:16]) - zero_ext16(XRc[23:16]));
int1 = abs(zero_ext16(XRb[15:08]) - zero_ext16(XRc[15:08]));
int0 = abs(zero_ext16(XRb[07:00]) - zero_ext16(XRc[07:00]));
XRa = int3 + int2 + int1 + int0;
XRd += int3 + int2 + int1 + int0;

Description:

Typical SAD operation for motion estimation.

Example:

Q8SAD  XR3, XR2, XR1, XR4
1.7.31 Q8AVG

Syntax:

Q8AVG  XRa, XRb, XRc

Operation:

byte3 = (zero_ext9(XRb[31:24]) + zero_ext9(XRc[31:24])) >> 1;
byte2 = (zero_ext9(XRb[23:16]) + zero_ext9(XRc[23:16])) >> 1;
byte1 = (zero_ext9(XRb[15:08]) + zero_ext9(XRc[15:08])) >> 1;
byte0 = (zero_ext9(XRb[07:00]) + zero_ext9(XRc[07:00])) >> 1;
XRa = {byte3, byte2, byte1, byte0};

Description:

Make quadruple 8-bit average operations for XRb (contain four packed 8-bits) and XRc
(contain four packed 8-bits), pack four 8-bit average results to XRa.

Example:

Q8AVG  XR3, XR2, XR1
1.7.32 Q8AVGR

Syntax:

Q8AVGR XRa, XRb, XRc

Operation:

byte3 = (zero_ext9(XRb[31:24]) + zero_ext9(XRc[31:24]) + 1) >> 1;
byte2 = (zero_ext9(XRb[23:16]) + zero_ext9(XRc[23:16]) + 1) >> 1;
byte1 = (zero_ext9(XRb[15:08]) + zero_ext9(XRc[15:08]) + 1) >> 1;
byte0 = (zero_ext9(XRb[07:00]) + zero_ext9(XRc[07:00]) + 1) >> 1;
XRa = {byte3, byte2, byte1, byte0};

Description:

Make quadruple 8-bit average with explicit rounding operations for XRb (contain four packed 8-bits) and XRc (contain four packed 8-bits), pack four 8-bit average results to XRa.

Example:

Q8AVGR XR3, XR2, XR1
1.8 Shift

1.8.1 D32SLL/D32SLR/D32SAR

Syntax:

D32SLL XRa, XRb, XRc, XRd, SFT4
D32SLR XRa, XRb, XRc, XRd, SFT4
D32SAR XRa, XRb, XRc, XRd, SFT4

Parameter:

SFT4: range 0 ~ 15

Operation:

D32SLL:

XRa = XRb << SFT4;
XRd = XRc << SFT4;

D32SLR:

XRa = XRb (>>L) SFT4;
XRd = XRc (>>L) SFT4;

D32SAR:

XRa = XRb (>>A) SFT4;
XRd = XRc (>>A) SFT4;

Description:

Make two 32-bit parallel shift for XRb and XRc to update XRa and XRd respectively. For logic right shift, filling zero to the left drained bit positions. For arithmetic right shift, filling sign bit to the left drained bit positions.

Example:

D32SAR XR3, XR2, XR5, 11
1.8.2 D32SARL

Syntax:

D32SARL XRa, XRb, XRc, SFT4

Parameter:

SFT4: range 0 ~ 15

Operation:

L32 = XRb (>>A) SFT4;
R32 = XRc (>>A) SFT4;
XRa = {L32[15:00], R32[15:00]};

Description:

Make two 32-bit parallel arithmetic right shift for XRb and XRc, pack two lower 16-bit shift results to update XRa.

Example:

D32SARL XR3, XR2, XR5, 11
1.8.3 D32SLLV/D32SLRV/D32SARV

Syntax:

D32SLLV XRa, XRd, rb
D32SLRV XRa, XRd, rb
D32SARV XRa, XRd, rb

Parameter:

Operation:

D32SLLV:
XRa = XRa << rb[3:0];
XRd = XRd << rb[3:0];

D32SLRV:
XRa = XRa (>>L) rb[3:0];
XRd = XRd (>>L) rb[3:0];

D32SARV:
XRa = XRa (>>A) rb[3:0];
XRd = XRd (>>A) rb[3:0];

Description:

Make two 32-bit parallel shift for XRa and XRd. For logic right shift, filling zero to the left drained bit positions. For arithmetic right shift, filling sign bit to the left drained bit positions.

Example:

D32SARV XR3, XR2, $21
1.8.4 D32SARW

Syntax:

\[ \text{D32SARW XRa, XRb, XRc, rb} \]

Parameter:

Operation:

\[
\begin{align*}
L32 &= \text{XRb (>>A)} \text{ rb}[3:0]; \\
R32 &= \text{XRc (>>A)} \text{ rb}[3:0]; \\
\text{XRa} &= \{L32[15:00], R32[15:00]\};
\end{align*}
\]

Description:

Make two 32-bit parallel arithmetic right shift for XRb and XRc, pack two lower 16-bit shift results to update XRa.

Example:

\[ \text{D32SARW XR3, XR2, XR5, $6} \]
1.8.5  Q16SLL/Q16SLR/Q16SAR

Syntax:

Q16SLL  XRa, XRb, XRc, XRd, SFT4
Q16SLR  XRa, XRb, XRc, XRd, SFT4
Q16SAR  XRa, XRb, XRc, XRd, SFT4

Parameter:

SFT4: range 0 ~ 15

Operation:

Q16SLL:
Short3 = XRb[31:16] << sft4;
Short2 = XRb[15:00] << sft4;
Short1 = XRc[31:16] << sft4;
Short0 = XRc[15:00] << sft4;
XRa = {Short3, Short2}; XRd = {Short1, Short0};

Q16SLR:
Short3 = XRb[31:16] (>>L) sft4;
Short2 = XRb[15:00] (>>L) sft4;
Short1 = XRc[31:16] (>>L) sft4;
Short0 = XRc[15:00] (>>L) sft4;
XRa = {Short3, Short2}; XRd = {Short1, Short0};

Q16SAR:
Short3 = XRb[31:16] (>>A) sft4;
Short2 = XRb[15:00] (>>A) sft4;
Short1 = XRc[31:16] (>>A) sft4;
Short0 = XRc[15:00] (>>A) sft4;
XRa = {Short3, Short2}; XRd = {Short1, Short0};

Description:

Shift two packed 16-bits in XRb to update XRa, and shift two packed 16-bits in XRc to
update XRd. For logic right shift, filling zero to the left drained bit positions. For
arithmetic right shift, filling sign bit to the left drained bit positions.

Example:

Q16SAR  XR3, XR2, XR5, XR9, 11
1.8.6 Q16SLLV/Q16SLRV/Q16SARV

Syntax:

Q16SLLV XRa, XRd, rb
Q16SLRV XRa, XRd, rb
Q16SARV XRa, XRd, rb

Operation:

Q16SLLV:
Short3 = XRa[31:16] << rb[3:0];
Short2 = XRa[15:00] << rb[3:0];
Short1 = XRd[31:16] << rb[3:0];
Short0 = XRd[15:00] << rb[3:0];
XRa = {Short3, Short2}; XRd = {Short1, Short0};

Q16SLRV:
Short3 = XRa[31:16] (>>L) rb[3:0];
Short2 = XRa[15:00] (>>L) rb[3:0];
Short1 = XRd[31:16] (>>L) rb[3:0];
Short0 = XRd[15:00] (>>L) rb[3:0];
XRa = {Short3, Short2}; XRd = {Short1, Short0};

Q16SARV:
Short3 = XRa[31:16] (>>A) rb[3:0];
Short2 = XRa[15:00] (>>A) rb[3:0];
Short1 = XRd[31:16] (>>A) rb[3:0];
Short0 = XRd[15:00] (>>A) rb[3:0];
XRa = {Short3, Short2}; XRd = {Short1, Short0};

Description:

Shift two packed 16-bits in XRa to update XRa, and shift two packed 16-bits in XRd to update XRd. For logic right shift, filling zero to the left drained bit positions. For arithmetic right shift, filling sign bit to the left drained bit positions.

Example:

Q16SARV XR3, XR9, $3
1.8.7 S32EXTR

Syntax:

S32EXTR XRa, XRd, rs, bits5

Operation:

position = rs[4:0];
left = 32 - position;
if (bits5 > left) {
    r_bits = bits5 - left; r_sft = 32 - r_bits;
    tmp = (XRa (<<L) r_bits) | (XRd (>>L) r_sft);
} else {
    r_bits = left - bits5;
    tmp = (XRa (>>L) r_bits);
}
XRa = tmp & ((1<<bits5) - 1);

Description:

Extract several bits (1~31 specified by constant value bits5) from the binding value {XRa, XRd} within bit range from bit position rs[4:0] to rs[4:0] + bits5 – 1, the extracting direction is from MSB of XRa to LSB of XRd.

NOTE:

Valid value range of constant bits5 is 1~31, 0 is unpredictable

Example:

S32EXTR XR3, XR2, $5, 23
1.8.8 S32EXTRV

Syntax:

S32EXTRV XRa, XRd, rs, rt

Operation:

position = rs[4:0];
sft5 = rt[4:0];
left = 32 - position;
if (sft5 > left) {
    r_bits = sft5 - left; r_sft = 32 - r_bits;
tmp = (XRa <<L r_bits) | (XRd >>L r_sft);
} else {
    r_bits = left - sft5;
tmp = (XRa >>L r_bits);
}
XRa = tmp & ((1<<sft5) - 1);

Description:

Extract several bits (1~31 specified by rt[4:0]) from the binding value (XRa, XRd) within bit range from bit position rs[4:0] to rs[4:0] + rt[4:0] – 1, the extracting direction is from MSB of XRa to LSB of XRd.

NOTE:

Valid value range of rt[4:0] is 1~31, 0 is unpredictable

Example:

S32EXTRV XR3, XR2, $5, $1
1.9 MAX/MIN

1.9.1 S32MAX

Syntax:

\[ \text{S32MAX } XRa, XRB, XRC \]

Operation:

\[
\begin{align*}
\text{If } & ((\text{signed})\text{XRb} > (\text{signed})\text{XRC}) \ XRa = \text{XRb}; \\
\text{Else } & \ XRa = \text{XRC};
\end{align*}
\]

Description:

Compare XRB and XRC to choose maximum one to update XRA.

Example:

\[ \text{S32MAX } XR3, XR2, XR1 \]
1.9.2  S32MIN

Syntax:

S32MIN  XRa, XRb, X Rc

Operation:

If ((signed)XRb < (signed)XRc)  XRa = XRb;
Else  X Ra = X Rc;

Description:

Compare XRb and XRc to choose minimum one to update X Ra.

Example:

S32MIN  XR3, XR2, XR1
1.9.3 D16MAX

Syntax:

D16MAX XR, XRb, XRc

Operation:

If ((signed)XRb[31:16] > (signed)XRc[31:16]) short1 = XRb[31:16];
Else short1 = XRc[31:16]);

If ((signed)XRb[15:0] > (signed)XRc[15:0]) short0 = XRb[15:0];
Else short0 = XRc[15:0]);

XRa = {short1, short0};

Description:

Compare two packed signed 16-bit pairs in XRb and XRc to choose two maximum
16-bit values to pack them to XRa.

Example:

D16MAX XR3, XR2, XR1
1.9.4 D16MIN

Syntax:

\[
\text{D16MIN} \ \text{XRa}, \ \text{XRb}, \ \text{XRc}
\]

Operation:

If \((\text{signed})\text{XRb}\[31:16\] \lt (\text{signed})\text{XRc}\[31:16\]) short1 = \text{XRb}\[31:16\];
Else short1 = \text{XRc}\[31:16\]);

If \((\text{signed})\text{XRb}\[15:0\] \lt (\text{signed})\text{XRc}\[15:0\]) short0 = \text{XRb}\[15:0\];
Else short0 = \text{XRc}\[15:0\]);

\text{XRa} = \{\text{short1}, \text{short0}\};

Description:

Compare two packed signed 16-bit pairs in XRb and XRc to choose two minimum 16-bit values to pack them to XRa.

Example:

\[
\text{D16MIN} \ \text{XR3}, \ \text{XR2}, \ \text{XR1}
\]
1.9.5 Q8MAX

Syntax:

Q8MAX XRa, XRb, XRc

Operation:

If ((signed)XRb[31:24] > (signed)XRc[31:24]) byte3 = XRb[31:24];
Else byte3 = XRc[31:24];

If ((signed)XRb[23:16] > (signed)XRc[23:16]) byte2 = XRb[23:16];
Else byte2 = XRc[23:16];

If ((signed)XRb[15:08] > (signed)XRc[15:08]) byte1 = XRb[15:08];
Else byte1 = XRc[15:08];

If ((signed)XRb[07:00] > (signed)XRc[07:00]) byte0 = XRb[07:00];
Else byte0 = XRc[07:00];

XRa = {byte3, byte2, byte1, byte0};

Description:

Compare four packed signed byte pairs in XRb and XRc to choose four maximum bytes to pack them to XRa.

Example:

Q8MAX XR3, XR2, XR1
1.9.6 Q8MIN

Syntax:

Q8MIN XRa, XRb, XRc

Operation:

If ((signed)XRb[31:24] < (signed)XRc[31:24]) byte3 = XRb[31:24];
Else byte3 = XRc[31:24];

If ((signed)XRb[23:16] < (signed)XRc[23:16]) byte2 = XRb[23:16];
Else byte2 = XRc[23:16];

If ((signed)XRb[15:08] < (signed)XRc[15:08]) byte1 = XRb[15:08];
Else byte1 = XRc[15:08];

If ((signed)XRb[07:00] < (signed)XRc[07:00]) byte0 = XRb[07:00];
Else byte0 = XRc[07:00];

XRa = {byte3, byte2, byte1, byte0};

Description:

Compare four packed signed byte pairs in XRb and XRc to choose four minimum bytes to pack them to XRa.

Example:

Q8MIN XR3, XR2, XR1
1.10 Bitwise

1.10.1 S32AND

Syntax:

\[ \text{S32AND XR}_a, \text{XR}_b, \text{XR}_c \]

Operation:

\[ \text{XR}_a = \text{XR}_b \& \text{XR}_c; \]

Description:

Use the result of \( \text{XR}_b \& \text{XR}_c \) to update \( \text{XR}_a \).

Example:

\[ \text{S32AND XR}_2, \text{XR}_5, \text{XR}_7 \]
1.10.2 S32OR

Syntax:

\[ \text{S32OR XR}_a, \text{XR}_b, \text{XR}_c \]

Operation:

\[ \text{XR}_a = \text{XR}_b | \text{XR}_c; \]

Description:

Use the result of \( \text{XR}_b \mid \text{XR}_c \) to update \( \text{XR}_a \).

Example:

\[ \text{S32OR XR}_2, \text{XR}_5, \text{XR}_7 \]
1.10.3 S32XOR

Syntax:

\[
\text{S32XOR } \text{XR}_a, \text{XR}_b, \text{XR}_c
\]

Operation:

\[
\text{XR}_a = \text{XR}_b \oplus \text{XR}_c;
\]

Description:

Use the result of \(\text{XR}_b \oplus \text{XR}_c\) to update \(\text{XR}_a\).

Example:

\[
\text{S32XOR } \text{XR}_2, \text{XR}_5, \text{XR}_7
\]
1.10.4 S32NOR

Syntax:

\[ \text{S32NOR XR}_a, \text{XR}_b, \text{XR}_c \]

Operation:

\[ \text{XR}_a = \neg (\text{XR}_b \lor \text{XR}_c) ; \]

Description:

Use the result of \(\neg (\text{XR}_b \lor \text{XR}_c)\) to update \(\text{XR}_a\).

Example:

\[ \text{S32NOR XR}_2, \text{XR}_5, \text{XR}_7 \]
1.11 Register move between GRF and XRF

1.11.1 S32M2I/S32I2M

Syntax:

\[
\begin{align*}
\text{S32M2I} & \quad \text{XRa, rb} \\
\text{S32I2M} & \quad \text{XRa, rb}
\end{align*}
\]

Operation:

\[
\begin{align*}
\text{S32M2I:} & \\
rb & = \text{XRa}; \\
\text{S32I2M:} & \\
\text{XRa} & = \text{rb};
\end{align*}
\]

Description:

Move value between GRF of IU and XRF of MXU.

Example:

\[
\text{S32M2I} \quad \text{XR2, $3$}
\]

**NOTE:** To access MXU_CR, XR16 is used. For example

\[
\begin{align*}
\text{S32M2I} & \quad \text{XR16, $1$} \quad ; \text{read content of MXU_CR to $1$} \\
\text{ORI} & \quad \text{$1, $1, 1} \\
\text{S32I2M} & \quad \text{XR16, $1$} \quad ; \text{enable MXU}
\end{align*}
\]

Moreover, when MXU is toggled from disable to enable, there are at least three non-MXU instructions necessary between the instruction S32I2M (which enables MXU) and following available MXU instructions.
1.12 Miscellaneous

1.12.1 S32SFL

Syntax:

\[ \text{S32SFL} \ XRa, XRb, X Rc, XRd, OPTN2 \]

Parameter:

OPTN2: ptn0(00), ptn1(01), ptn2(10), ptn3(11)

Operation:

<table>
<thead>
<tr>
<th>ptn0</th>
<th>ptn1</th>
<th>ptn2</th>
<th>ptn3</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRb</td>
<td>XRb</td>
<td>XRb</td>
<td>XRb</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>XRc</td>
<td>XRc</td>
<td>XRc</td>
<td>XRc</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>XRd</td>
<td>XRd</td>
<td>XRd</td>
<td>XRd</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>

Description:

As in the figure above.

Example:

\[ \text{S32SFL} \ X R 3, X R 2, X R 1, X R 5, \text{optn} \]

\[ \text{S32SFL} \ X R 3, X R 2, X R 1, X R 5, \text{ptn0} \]
1.12.2 S32ALN/S32ALNI

Syntax:

S32ALN  XRa, XRb, X Rc, rs
S32ALNI XRa, XRb, X Rc, OPTN3

Parameter:

OPTN3: ptn0(000), ptn1(001), ptn2(010), ptn3(011), ptn4(100), 101~111 are reserved

Operation:

Switch (rs[2:0]|OPTN3) {
  case 0:  X Ra = {XRb[31:0]};
  case 1:  X Ra = {XRb[23:0], X Rc[31:24]};
  case 2:  X Ra = {XRb[15:0], X Rc[31:16]};
  case 3:  X Ra = {XRb[07:0], X Rc[31:08]};
  case 4:  X Ra = {X Rc[31:0]};
  default: Undefined
}

Description:

Concatenate unaligned value of XRb and unaligned value of X Rc to form an aligned word value to update X Ra

Example:

S32ALN  XR3, XR2, XR1, $5
S32ALNI XR5, XR1, XR7, 4
S32ALNI XR5, XR1, XR7, ptn2
1.12.3 Q16SAT

Syntax:

Q16SAT  XRa, XRb, X Rc

Operation:

byte3 = sat_8((signed)XRb[31:16]);
byte2 = sat_8((signed)XRb[15:00]);
byte1 = sat_8((signed)XRc[31:16]);
byte0 = sat_8((signed)XRc[15:00]);
XRa = {byte3, byte2, byte1, byte0};

Description:

XRa is updated by four packed 8-bit saturation results (0 ~ 255) calculated from four
16-bit signed values of XRb and XRc.

Example:

Q16SAT  XR3, XR2, XR1
1.12.4 Q16SCOP

Syntax:

Q16SCOP XRa, XRb, X Rc, XRd

Operation:

short3 = XRb[31:16] < 0? 0xFFFF : (XRb[31:16] > 0? 1 : 0);
short2 = XRb[15:00] < 0? 0xFFFF : (XRb[15:00] > 0? 1 : 0);
short1 = XRc[31:16] < 0? 0xFFFF : (XRc[31:16] > 0? 1 : 0);
short0 = XRc[15:00] < 0? 0xFFFF : (XRc[15:00] > 0? 1 : 0);
XRa = {short3, short2};
XRd = {short1, short0};

Description:

Determine the 16-bit value range, the determination is as follow:
range = x < 0? –1 : (x > 0?) 1 : 0;

Example:

Q16SCOP XR3, XR2, XR1, XR5
1.12.5 S32LUI

Syntax:

S32LUI XRa, S8, OPTN3

Parameter:

OPTN3: ptn0(000), ptn1(001), ptn2(010), ptn3(011), ptn4(100),
       ptn5(101), ptn6(110), ptn7(111)

Operation:

    tmp8 = S8;
    switch(OPTN3) {
      case 0: XRa = {24'b0, tmp8};
      case 1: XRa = {16'b0, tmp8, 8'b0};
      case 2: XRa = {8'b0, tmp8, 16'b0};
      case 3: XRa = {tmp8, 24'b0};
      case 4: XRa = {8'b0, tmp8, 8'b0, tmp8};
      case 5: XRa = {tmp8, 8'b0, tmp8, 8'b0};
      case 6: XRa = {{8{sign of tmp8}}, tmp8, {8{sign of tmp8}}, tmp8};
      case 7: XRa = {tmp8, tmp8, tmp8, tmp8};
    }

Description:

Permute the immediate value (S8) in terms of specific pattern to form a word to
update XRa

Example:

    S32LUI XRl1, 0xF, 6
    S32LUI XRl1, 0xFF, ptn4