

Appendix

A RISC-V Tensor Extension Instruction Set

```

1513 cfg.satu      rs
1514 //Configure whether the results of integer calculation
1515 //instructions are saturated and whether symmetric
1516 //saturation is adopted.
1517 cfg.round_mode rs
1518 // Configure the rounding mode during floating-point
1519 // precision conversion or integer right shift
1520 cfg.rsqrt_iter rs
1521 // Configure the number of Newton iterations for rsqrt
1522 // and fdiv instructions
1523 cfg.quant      ts
1524 // Configure the quantization method, supporting per-
1525 // channel quantization and per-tensor quantization
1526 cfg.pad        rs
1527 // Configure the top, bottom, left, and right padding of the
1528 // feature map for convolution and pooling
1529 // instructions
1530 cfg.insrt      rs
1531 // Configure the interpolation of the feature map or
1532 // kernel for convolution and pooling instructions
1533 cfg.stencil    rs
1534 // Configure the sliding window, stride, whether the
1535 // kernel is rotated, and whether the result is passed
1536 // through ReLU for convolution and pooling
1537 // instructions
1538 cfg.kzp        ts
1539 // Configure the zero point of the kernel for convolution
1540 // and matrix multiplication instructions
1541 cfg.dmaidx     rs
1542 // Configure parameters such as the initial value of the
1543 // index or the default write-back value for
1544 // instructions like ls.hscatter, ls.hgather, and ls.
1545 // nzidx

```

Figure 21: CSR Configuration Instructions

```

1540 cfgcr         ca, rs
1541 // Configure the data type and constant value of CR
1542 cfgtr         ta, rs
1543 // Configure the data type and offset address of TR
1544 cfgtr.shape   ta, rs
1545 // Configure TR as N, C, H, W
1546 cfgtr.hwstride ta, rs
1547 // Configure the H stride and W stride of TR
1548 cfgtr.ncstride ta, rs
1549 // Configure the N stride and C stride of TR
1550 cfggt         ga, rs
1551 // Configure the data type and address of GT
1552 cfggt.shape   ga, rs
1553 // Configure GT as N, C, H, W
1554 cfggt.hwstride ga, rs
1555 // Configure the H stride and W stride of GT
1556 cfggt.ncstride ga, rs
1557 // Configure the N stride and C stride of GT

```

Figure 22: CR, TVR, GVR Configuration Instructions

```

1555 //Data Copy
1556 ls.cp         dst, src
1557 // Copy the source GT or TR to the destination GT or TR
1558 ls.cpb        dst, src
1559 // Copy the source GT or TR(n, 1, h, w) to the
1560 // destination TR(n, c, h, w)
1561 ls.cpt        dst, src
1562 // Copy the source GT or TR(n, c, h, w) to the
1563 // destination GT or TR(c, n, h, w)
1564 ls.cpr        dst, src, _dim
1565 // Copy the source GT or TR(n, 1, h, w) to the
1566 // destination GT or TR(n, c, h, w) while reversing the
1567 // elements along a specified dimension

```

Figure 23: LOAD & STORE Instructions I

```

1567 //Tensor Load and Store Instructions
1568 ls.ld         td, gs
1569 // Load the source GT into the destination TR. The shapes
1570 // of the source and destination tensors may differ,
1571 // but the total number of elements must remain the
1572 // same
1573 ls.ldt        td, gs
1574 // Load the source GT(c, n, h, w) into the destination TR
1575 // (n, c, h, w), transposing the N and C dimensions
1576 // during loading
1577 ls.ldbc       td, gs
1578 // Load the source GT(n, 1, h, w) into the destination TR
1579 // (n, c, h, w), broadcasting in the C dimension during
1580 // loading
1581 ls.st         gd, ts
1582 // Store the source TR into the destination GT. The
1583 // shapes of the source and destination tensors may
1584 // differ, but the total number of elements must remain
1585 // the same
1586 ls.stt        gd, ts
1587 // Store the source TR(n, c, h, w) into the destination
1588 // GT(c, n, h, w), transposing the N and C dimensions
1589 // during storing
1590 //Matrix Load and Store Instructions
1591 ls.mld        td, gs
1592 // Load the source GT(1, 1, m, n) into the destination TR
1593 // (m, ROUNDUP(n/CU_NUM), 1, w)
1594 ls.mst        gd, ts
1595 // Store the source TR(m, ROUNDUP(n/CU_NUM), 1, w) into
1596 // the destination GT(1, 1, m, n)
1597 //Mask & Index Instructions
1598 ls.maskssel   dst, src, mask, _bigmask
1599 // Select elements from the source tensor at positions
1600 // where the mask tensor has a value of 1, and output
1601 // them to the destination GT. The mask and source
1602 // tensor can be either GT or TR; the mask tensor is of
1603 // integer type
1604 ls.fmaskssel  dst, src, mask, _bigmask
1605 // Select elements from the source tensor at positions
1606 // where the mask tensor has a value of 1, and output
1607 // them to the destination GT. The mask and source
1608 // tensor can be either GT or TR; the mask tensor is of
1609 // floating-point type
1610 ls.nzidx      dst, src
1611 // Treat the source tensor as a one-dimensional vector,
1612 // generate indices of non-zero elements, and output
1613 // them to the destination GT; the elements of the
1614 // source tensor are integers
1615 ls.fnzidx     dst, src
1616 // Treat the source tensor as a one-dimensional vector,
1617 // generate indices of non-zero elements, and output
1618 // them to the destination GT; the elements of the
1619 // source tensor are floating-point numbers
1620 //Gather & Scatter Instructions
1621 ls.hgather    dst, src, idx, _bigh
1622 // Using elements in the index tensor(1,1,h,1) or (1,c,h
1623 // ,1) as coordinates, gather data from the source
1624 // tensor(1,1,ih,1) or (1,c,ih,1) into the destination
1625 // tensor(1,1,h,1) or (1,c,h,1)
1626 ls.hscatter   dst, src, idx, _bigh, _accu
1627 // Sequentially read data from the source tensor(1,1,ih
1628 // ,1) or (1,c,ih,1), and scatter it to the destination
1629 // tensor(1,1,h,1) or (1,c,h,1) using elements in the
1630 // index tensor(1,1,ih,1) or (1,c,ih,1) as coordinates,
1631 // supporting accumulation into an integer destination
1632 // tensor
1633 ls.fhscatter  dst, src, idx, _bigh, _accu
1634 // Sequentially read data from the source tensor(1,1,ih
1635 // ,1) or (1,c,ih,1), and scatter it to the destination
1636 // tensor(1,1,h,1) or (1,c,h,1) using elements in the
1637 // index tensor(1,1,ih,1) or (1,c,ih,1) as coordinates,
1638 // supporting accumulation into a floating-point
1639 // destination tensor

```

Figure 24: LOAD & STORE Instructions II

```

1625  ///Integer Arithmetic Instructions
1626  add    out, a, b, shift, _satu
1627  // Integer addition, supporting shifting and saturation
1628  sub    out, a, b, shift, _satu
1629  // Integer subtraction, supporting shifting and
1630  // saturation
1631  mul    out, a, b, shift, _satu
1632  // Integer multiplication, supporting shifting and
1633  // saturation
1634  abs    out, a
1635  // Integer absolute value
1636  mac    out, a, b, shift
1637  // Integer multiply-accumulate, supporting shifting and
1638  // saturation
1639  max    out, a, b
1640  // Integer maximum value
1641  min    out, a, b
1642  // Integer minimum value
1643  selgt  out, a, b, c
1644  // If integer a > b, then out = c; otherwise, 0
1645  seleq  out, a, b, c
1646  // If integer a == b, then out = c; otherwise, 0
1647  sellt  out, a, b, c
1648  // If integer a < b, then out = c; otherwise, 0
1649  cmplt  out, a, b, c, d
1650  // Integer: if a < b then c, else d
1651  cmpeq  out, a, b, c, d
1652  // Integer: if a == b then c, else d
1653  cmpgt  out, a, b, c, d
1654  // Integer: if a > b then c, else d
1655  and    out, a, b
1656  // Logical AND
1657  xor    out, a, b
1658  // Logical XOR
1659  or     out, a, b
1660  // Logical OR
1661  not    out, a
1662  // Logical NOT
1663  lshr   out, a, shift
1664  // Logical right shift
1665  ashr   out, a, shift
1666  // Arithmetic right shift
1667  rshr   out, a, shift
1668  // Rotate right shift
1669  clz    out, a
1670  // Count the number of leading zeros
1671  clo    out, a
1672  // Count the number of leading ones
1673  /// Floating-Point Arithmetic Instructions
1674  fadd   out, a, b
1675  // Floating-point addition with optional saturation
1676  fsub   out, a, b
1677  // Floating-point subtraction with optional saturation
1678  fmul   out, a, b
1679  // Floating-point multiplication with optional saturation
1680  fdiv   out, a, b
1681  // Floating-point division with optional saturation
1682  fabs   out, a
1683  // Floating-point absolute value
1684  fmac   out, a, b
1685  // Floating-point multiply-accumulate
1686  fmax   out, a, b
1687  // Floating-point maximum value
1688  fmin   out, a, b
1689  // Floating-point minimum value
1690  fselgt out, a, b, c
1691  // If floating-point a > b then out = c, otherwise 0
1692  fseleq out, a, b, c
1693  // If floating-point a == b then out = c, otherwise 0
1694  fsellt out, a, b, c
1695  // If floating-point a < b then out = c, otherwise 0
1696  fcmlt  out, a, b, c, d
1697  // Floating-point: if a < b then out = c, else out = d
1698  fcmpeq out, a, b, c, d
1699  // Floating-point: if a == b then out = c, else out = d
1700  fcmlt  out, a, b, c, d
1701  // Floating-point: if a > b then out = c, else out = d

```

Figure 25: Compute Instructions I

```

1683  /// Data Type Conversion Instructions
1684  cvt.i2i    out, a
1685  // Integer precision conversion
1686  cvt.i2f    out, a
1687  // Convert integer to float
1688  cvt.f2i    out, a
1689  // Convert float to integer
1690  cvt.f2f    out, a
1691  // Floating-point precision conversion
1692  /// SFU Instructions
1693  sfu.norm    out, a
1694  // Extract exponent part of float and convert to integer
1695  sfu.taylor  out, a, coeff
1696  // Polynomial evaluation
1697  sfu.rsqrt   out, a
1698  // Reciprocal square root
1699  /// Quantization/Dequantization Instructions
1700  dq0         out, a, scale
1701  // The first dequantization: scale and convert to
1702  // floating-point
1703  rq0         out, a, scale
1704  // The first quantization: scale and convert to low-
1705  // precision integer
1706  dq1         out, a, scale
1707  // The second dequantization: scale and convert to high-
1708  // precision integer
1709  rq1         out, a, scale
1710  // The second quantization: scale and convert to low-
1711  // precision integer
1712  dq2         out, a, scale, _gsize
1713  // The third dequantization: scale and convert to half-
1714  // precision floating-point
1715  /// Pooling Instructions
1716  pool.avg    out, x, w, _rq
1717  // Integer 2D average pooling
1718  pool.max    out, x
1719  // Integer 2D max pooling
1720  pool.min    out, x
1721  // Integer 2D min pooling
1722  pool.favg   out, x, w
1723  // Floating-point 2D average pooling
1724  pool.fmax   out, x
1725  // Floating-point 2D max pooling
1726  pool.fmin   out, x
1727  // Floating-point 2D min pooling
1728  roipool.avg out, x, w, roi, _rq
1729  // Integer ROI average pooling
1730  roipool.favg out, x, w, roi
1731  // Floating-point ROI average pooling
1732  roipool.max  out, x, roi
1733  // Integer ROI max pooling
1734  roipool.fmax out, x, roi
1735  // Floating-point ROI max pooling
1736  roipool.min  out, x, roi
1737  // Integer ROI min pooling
1738  roipool.fmin out, x, roi
1739  // Floating-point ROI min pooling
1740  dwconv      out, x, w, bias, _rq
1741  // Integer depthwise convolution
1742  fdwconv     out, x, w, bias
1743  // Floating-point depthwise convolution
1744  /// Convolution Instructions
1745  conv        out, x, w, bias, _rq
1746  // Integer convolution
1747  conva       out, x, w, bias
1748  // Integer convolution accumulate
1749  fconv       out, x, w, bias
1750  // Floating-point convolution
1751  fconva      out, x, w, bias
1752  // Floating-point convolution accumulate
1753  /// CUBE Matrix Multiplication Instructions
1754  mm.<nn|nt|tt> out, x, w, bias, _relu
1755  // Integer matrix multiplication
1756  mma.<nn|nt|tt> out, x, w, bias, _relu
1757  // Integer matrix multiplication accumulate
1758  fmm.<nn|nt|tt> out, x, w, bias, _relu
1759  // Floating-point matrix multiplication
1760  fmma.<nn|nt|tt> out, x, w, bias, _relu
1761  // Floating-point matrix multiplication accumulate

```

Figure 26: Compute Instructions II

```

1741 // Fully Connected Matrix Multiplication Instructions
1742 fcmml.<nn|tn> out, x, w, bias, _rq, _relu
1743 // Integer matrix multiplication for fully connected
1744 layers
1745 fcmmla.<nn|tn> out, x, w, bias, _relu
1746 // Accumulated integer matrix multiplication for fully
1747 connected layers
1748 ffcmm.<nn|tn> out, x, w, bias, _relu
1749 // Floating-point matrix multiplication for fully
1750 connected layers
1751 ffcmma.<nn|tn> out, x, w, bias, _relu
1752 // Accumulated floating-point matrix multiplication for
1753 fully connected layers
1754 // Cross Comparison Instructions
1755 fvcmax out, a, b
1756 // Element-wise cross comparison of two floating-point
1757 vectors to select the maximum value
1758 fvcmin out, a, b
1759 // Element-wise cross comparison of two floating-point
1760 vectors to select the minimum value
1761 vcmax out, a, b
1762 // Element-wise cross comparison of two integer vectors
1763 to select the maximum value
1764 vcmin out, a, b
1765 // Element-wise cross comparison of two integer vectors
1766 to select the minimum value
1767 // Data Copy and Reordering Instructions
1768 cp out, a
1769 // Tensor copy instruction.
1770 bc out, a
1771 // Broadcast: replicate a(n, 1, h, w) along the C
1772 dimension to form out(n, c, h, w)
1773 cwtrans out, a
1774 // CW-dimension transpose instruction. Transposes the C
1775 and W dimensions of the input floating-point Tensor
1776 A and outputs the result to Tensor out.
1777 wctrans out, a
1778 // WC-dimension transpose instruction. Transpose the W
1779 and C dimensions of the input floating-point Tensor
1780 A and outputs the result to Tensor out.
1781 gather.pc out, a, idx, cs, _bdlimit
1782 // Per-channel gather instruction. Gather data from the W
1783 dimension of Tensor A into Tensor out based on the
1784 channel-shared index.
1785 scatter.pc out, a, idx
1786 // Per-channel scatter instruction. Scatter data from the
1787 W dimension of Tensor A into Tensor out based on
1788 the channel-shared index.
1789 gather2d.pc out, a, idx, cs
1790 // 2D gather instruction. Gather data from the H and W
1791 dimensions of Tensor A into Tensor out based on 2D
1792 coordinates shared per channel.
1793 scatter2d.pc out, a, idx
1794 // 2D scatter instruction. Scatter data from the H and W
1795 dimensions of Tensor A into Tensor out based on 2D
1796 coordinates shared per channel.
1797 gather out, a, idx, cs
1798 // Gather instruction. Gather data from the W dimension
1799 of Tensor A into Tensor out based on the provided
1800 index.
1801 scatter out, a, idx
1802 // Scatter instruction. Scatter data from the W dimension
1803 of Tensor A into Tensor out based on the provided
1804 index.
1805 hgatter out, a, idx, cs
1806 // H-dimension gather instruction. Gather data from the H
1807 dimension of Tensor A into Tensor out based on the
1808 provided index.
1809 hscatter out, a, idx
1810 // H-dimension scatter instruction. Scatter data from the
1811 H dimension of Tensor A into Tensor out based on
1812 the provided index.
1813 masksel out_cnt, out, a, mask
1814 // Mask selection instruction. Write elements from the W
1815 dimension of Tensor A to Tensor out at positions
1816 where the mask has a value of 1.
1817 nzidx out_cnt, out_idx, a
1818 // Non-zero index generation instruction. Write the
1819 indices of non-zero elements from the W dimension of
1820 Tensor A into Tensor out.

```

Figure 27: Compute Instructions III

```

mov.t.v vs, (rs)
// Load a set of VRF data into LMEM; rs specifies the
// starting relative address in LMEM
mov.dist.v vs, (rs)
// Distribute a set of VRF data evenly to all LMEMs; rs
// specifies the starting relative address in LMEM
mov.t.vv vs2, vs1, (rs)
// Load two sets of VRF data into LMEM; rs specifies the
// starting relative address in LMEM
mov.dist.vv vs2, vs1, (rs)
// Distribute two sets of VRF data evenly to all LMEMs;
// rs specifies the starting relative address in LMEM
mov.v.t vd, (rs)
// Store data from LMEM into a set of VRF; rs specifies
// the starting relative address in LMEM
mov.v.coll vd, (rs)
// Collect data from the same relative offset in each
// LMEM and store it into a set of VRF; rs specifies
// the starting relative address in LMEM

```

Figure 28: MOVE Instructions

```

sync.i rs, _engine
// rs holds the tag. After executing this instruction,
// the tag is written into CSR.sync_tag. The CPU can
// read this register to determine whether the
// instruction has finished executing, and once read,
// the register is automatically cleared.
msgsend rs
// Send the information in rs to the message queue.
msgwait rs
// Block instruction issue until a message is received
// from the message queue, then store the message in rs
.

```

Figure 29: Synchronization Instructions