Appendix

A RISC-V Tensor Extension Instruction Set

```
cfg.satu       rs
//Configure whether the results of integer calculation
     instructions are saturated and whether symmetric
     saturation is adopted.
cfg.round_mode rs
// Configure the rounding mode during floating-point
     precision conversion or integer right shift
cfg.rsqrt_iter rs
// Configure the number of Newton iterations for rsqrt
     and fdiv instructions
cfg.quant      ts
// Configure the quantization method, supporting per-
     channel quantization and per-tensor quantization
cfg.pad        rs
// Configure the top, bottom, left, and right padding of
     the feature map for convolution and pooling
     instructions
cfg.insrt      rs
// Configure the interpolation of the feature map or
     kernel for convolution and pooling instructions
cfg.stencil    rs
// Configure the sliding window, stride, whether the
     kernel is rotated, and whether the result is passed
     through ReLU for convolution and pooling
     instructions
cfg.kzp        ts
// Configure the zero point of the kernel for convolution
      and matrix multiplication instructions
cfg.dmaidx     rs
// Configure parameters such as the initial value of the
     index or the default write-back value for
     instructions like ls.hscatter, ls.hgather, and ls.
     nzidx
```

Figure 21: CSR Configuration Instructions

```
cfgcr          ca, rs
// Configure the data type and constant value of CR
cfgtr          ta, rs
// Configure the data type and offset address of TR
cfgtr.shape    ta, rs
// Configure TR as N, C, H, W
cfgtr.hwstride  ta, rs
// Configure the H stride and W stride of TR
cfgtr.ncstride  ta, rs
// Configure the N stride and C stride of TR
cfggt          ga, rs
// Configure the data type and address of GT
cfggt.shape     ga, rs
// Configure GT as N, C, H, W
cfggt.hwstride   ga, rs
// Configure the H stride and W stride of GT
cfggt.ncstride   ga, rs
// Configure the N stride and C stride of GT
```

Figure 22: CR, TVR, GVR Configuration Instructions

```
////Data Copy
ls.cp           dst, src
// Copy the source GT or TR to the destination GT or TR
ls.cpbc         dst, src
// Copy the source GT or TR(n, 1, h, w) to the
     destination TR(n, c, h, w)
ls.cpt          dst, src
// Copy the source GT or TR(n, c, h, w) to the
     destination GT or TR(c, n, h, w)
ls.cpr          dst, src, _dim
// Copy the source GT or TR(n, 1, h, w) to the
     destination GT or TR(n, c, h, w) while reversing the
      elements along a specified dimension
```

Figure 23: LOAD & STORE Instructions I

```
////Tensor Load and Store Instructions
ls.ld         td, gs
// Load the source GT into the destination TR. The shapes
      of the source and destination tensors may differ,
     but the total number of elements must remain the
     same
ls.ldt        td, gs
// Load the source GT(c, n, h, w) into the destination TR
     (n, c, h, w), transposing the N and C dimensions
     during loading
ls.ldbc       td, gs
// Load the source GT(n, 1, h, w) into the destination TR
     (n, c, h, w), broadcasting in the C dimension during
      loading
ls.st          gd, ts
// Store the source TR into the destination GT. The
     shapes of the source and destination tensors may
     differ, but the total number of elements must remain
      the same
ls.stt         gd, ts
// Store the source TR(n, c, h, w) into the destination
     GT(c, n, h, w), transposing the N and C dimensions
     during storing
////Matrix Load and Store Instructions
ls.mld        td, gs
// Load the source GT(1, 1, m, n) into the destination TR
     (m, ROUNDUP(n/CU_NUM), 1, w)
ls.mst         gd, ts
// Store the source TR(m, ROUNDUP(n/CU_NUM), 1, w) into
     the destination GT(1, 1, m, n)
//// Mask & Index Instructions
ls.masksel     dst, src, mask, _bigmask
// Select elements from the source tensor at positions
     where the mask tensor has a value of 1, and output
     them to the destination GT. The mask and source
     tensor can be either GT or TR; the mask tensor is of
      integer type
ls.fmasksel    dst, src, mask, _bigmask
// Select elements from the source tensor at positions
     where the mask tensor has a value of 1, and output
     them to the destination GT. The mask and source
     tensor can be either GT or TR; the mask tensor is of
      floating-point type
ls.nzidx       dst, src
// Treat the source tensor as a one-dimensional vector,
     generate indices of non-zero elements, and output
     them to the destination GT; the elements of the
     source tensor are integers
ls.fnzidx      dst, src
// Treat the source tensor as a one-dimensional vector,
     generate indices of non-zero elements, and output
     them to the destination GT; the elements of the
     source tensor are floating-point numbers
//// Gather & Scatter Instructions
ls.hgather     dst, src, idx, _bigh
// Using elements in the index tensor(1,1,h,1) or (1,c,h
     ,1) as coordinates, gather data from the source
     tensor(1,1,ih,1) or (1,c,ih,1) into the destination
     tensor(1,1,h,1) or (1,c,h,1)
ls.hscatter    dst, src, idx, _bigh, _accu
// Sequentially read data from the source tensor(1,1,ih
     ,1) or (1,c,ih,1), and scatter it to the destination
      tensor(1,1,h,1) or (1,c,h,1) using elements in the
     index tensor(1,1,ih,1) or (1,c,ih,1) as coordinates,
      supporting accumulation into an integer destination
      tensor
ls.fhscatter   dst, src, idx, _bigh, _accu
// Sequentially read data from the source tensor(1,1,ih
     ,1) or (1,c,ih,1), and scatter it to the destination
      tensor(1,1,h,1) or (1,c,h,1) using elements in the
     index tensor(1,1,ih,1) or (1,c,ih,1) as coordinates,
      supporting accumulation into a floating-point
     destination tensor
```

Figure 24: LOAD & STORE Instructions II

```
////Integer Arithmetic Instructions
add      out, a, b, shift, _satu
// Integer addition, supporting shifting and saturation
sub      out, a, b, shift, _satu
// Integer subtraction, supporting shifting and saturation
mul      out, a, b, shift, _satu
// Integer multiplication, supporting shifting and saturation
abs      out, a
// Integer absolute value
mac      out, a, b, shift
// Integer multiply-accumulate, supporting shifting and saturation
max      out, a, b
// Integer maximum value
min      out, a, b
// Integer minimum value
selgt    out, a, b, c
// If integer a > b, then out = c; otherwise, 0
seleq    out, a, b, c
// If integer a == b, then out = c; otherwise, 0
sellt    out, a, b, c
// If integer a < b, then out = c; otherwise, 0
cmplt    out, a, b, c, d
// Integer: if a < b then c, else d
cmpeq    out, a, b, c, d
// Integer: if a == b then c, else d
cmpgt    out, a, b, c, d
// Integer: if a > b then c, else d
and      out, a, b
// Logical AND
xor      out, a, b
// Logical XOR
or       out, a, b
// Logical OR
not      out, a
// Logical NOT
lshr     out, a, shift
// Logical right shift
ashr     out, a, shift
// Arithmetic right shift
rshr     out, a, shift
// Rotate right shift
clz      out, a
// Count the number of leading zeros
clo      out, a
// Count the number of leading ones
//// Floating-Point Arithmetic Instructions
fadd     out, a, b
// Floating-point addition with optional saturation
fsub     out, a, b
// Floating-point subtraction with optional saturation
fmul     out, a, b
// Floating-point multiplication with optional saturation
fdiv     out, a, b
// Floating-point division with optional saturation
fabs     out, a
// Floating-point absolute value
fmac     out, a, b
// Floating-point multiply-accumulate
fmax     out, a, b
// Floating-point maximum value
fmin     out, a, b
// Floating-point minimum value
fselgt   out, a, b, c
// If floating-point a > b then out = c, otherwise 0
fseleq   out, a, b, c
// If floating-point a == b then out = c, otherwise 0
fsellt   out, a, b, c
// If floating-point a < b then out = c, otherwise 0
fcmplt   out, a, b, c, d
// Floating-point: if a < b then out = c, else out = d
fcmpeq   out, a, b, c, d
// Floating-point: if a == b then out = c, else out = d
fcmpgt   out, a, b, c, d
// Floating-point: if a > b then out = c, else out = d
```

Figure 25: Compute Instructions I

```
//// Data Type Conversion Instructions
cvt.i2i          out, a
// Integer precision conversion
cvt.i2f          out, a
// Convert integer to float
cvt.f2i          out, a
// Convert float to integer
cvt.f2f          out, a
// Floating-point precision conversion
//// SFU Instructions
sfu.norm         out, a
// Extract exponent part of float and convert to integer
sfu.taylor       out, a, coeff
// Polynomial evaluation
sfu.rsqrt        out, a
// Reciprocal square root
//// Quantization/Dequantization Instructions
dq0              out, a, scale
// The first dequantization: scale and convert to floating-point
rq0              out, a, scale
// The first quantization: scale and convert to low-precision integer
dq1              out, a, scale
// The second dequantization: scale and convert to high-precision integer
rq1              out, a, scale
// The second quantization: scale and convert to low-precision integer
dq2              out, a, scale, _gsize
// The third dequantization: scale and convert to half-precision floating-point
//// Pooling Instructions
pool.avg         out, x, w, _rq
// Integer 2D average pooling
pool.max         out, x
// Integer 2D max pooling
pool.min         out, x
// Integer 2D min pooling
pool.favg        out, x, w
// Floating-point 2D average pooling
pool.fmax        out, x
// Floating-point 2D max pooling
pool.fmin        out, x
// Floating-point 2D min pooling
roipool.avg      out, x, w, roi, _rq
// Integer ROI average pooling
roipool.favg     out, x, w, roi
// Floating-point ROI average pooling
roipool.max      out, x, roi
// Integer ROI max pooling
roipool.fmax     out, x, roi
// Floating-point ROI max pooling
roipool.min      out, x, roi
// Integer ROI min pooling
roipool.fmin     out, x, roi
// Floating-point ROI min pooling
dwconv           out, x, w, bias, _rq
// Integer depthwise convolution
fdwconv          out, x, w, bias
//Floating-point depthwise convolution
//// Convolution Instructions
conv             out, x, w, bias, _rq
// Integer convolution
conva            out, x, w, bias
// Integer convolution accumulate
fconv            out, x, w, bias
// Floating-point convolution
fconva           out, x, w, bias
// Floating-point convolution accumulate
//// CUBE Matrix Multiplication Instructions
mm.<nn|nt|tt>    out, x, w, bias, _rq, _relu
// Integer matrix multiplication
mma.<nn|nt|tt>   out, x, w, bias, _relu
// Integer matrix multiplication accumulate
fmm.<nn|nt|tt>   out, x, w, bias, _relu
// Floating-point matrix multiplication
fmma.<nn|nt|tt>  out, x, w, bias, _relu
//Floating-point matrix multiplication accumulate
```

Figure 26: Compute Instructions II

```
//// Fully Connected Matrix Multiplication Instructions
fcmm.<nn|tn>    out, x, w, bias, _rq, _relu
// Integer matrix multiplication for fully connected
     layers
fcmma.<nn|tn>   out, x, w, bias, _relu
// Accumulated integer matrix multiplication for fully
     connected layers
ffcmm.<nn|tn>   out, x, w, bias, _relu
// Floating-point matrix multiplication for fully
     connected layers
ffcmma.<nn|tn>  out, x, w, bias, _relu
// Accumulated floating-point matrix multiplication for
     fully connected layers
////Cross Comparison Instructions
fvcmax          out, a, b
// Element-wise cross comparison of two floating-point
     vectors to select the maximum value
fvcmin          out, a, b
// Element-wise cross comparison of two floating-point
     vectors to select the minimum value
vcmax           out, a, b
// Element-wise cross comparison of two integer vectors
     to select the maximum value
vcmin           out, a, b
// Element-wise cross comparison of two integer vectors
     to select the minimum value
////Data Copy and Reordering Instructions
cp              out, a
// Tensor copy instruction.
bc              out, a
// Broadcast: replicate a(n, 1, h, w) along the C
     dimension to form out(n, c, h, w)
cwtrans         out, a
// CW-dimension transpose instruction. Transposes the C
     and W dimensions of the input floating-point Tensor
     A and outputs the result to Tensor out.
wctrans         out, a
// WC-dimension transpose instruction. Transpose the W
     and C dimensions of the input floating-point Tensor
     A and outputs the result to Tensor out.
gather.pc       out, a, idx, cs, _bdlimit
// Per-channel gather instruction. Gather data from the W
      dimension of Tensor A into Tensor out based on the
     channel-shared index.
scatter.pc      out, a, idx
// Per-channel scatter instruction. Scatter data from the
      W dimension of Tensor A into Tensor out based on
     the channel-shared index.
gather2d.pc     out, a, idx, cs
// 2D gather instruction. Gather data from the H and W
     dimensions of Tensor A into Tensor out based on 2D
     coordinates shared per channel.
scatter2d.pc    out, a, idx
// 2D scatter instruction. Scatter data from the H and W
     dimensions of Tensor A into Tensor out based on 2D
     coordinates shared per channel.
gather          out, a, idx, cs
// Gather instruction. Gather data from the W dimension
     of Tensor A into Tensor out based on the provided
     index.
scatter         out, a, idx
// Scatter instruction. Scatter data from the W dimension
      of Tensor A into Tensor out based on the provided
     index.
hgatter         out, a, idx, cs
// H-dimension gather instruction. Gather data from the H
      dimension of Tensor A into Tensor out based on the
     provided index.
hscatter        out, a, idx
// H-dimension scatter instruction. Scatter data from the
      H dimension of Tensor A into Tensor out based on
     the provided index.
masksel         out_cnt, out, a, mask
// Mask selection instruction. Write elements from the W
     dimension of Tensor A to Tensor out at positions
     where the mask has a value of 1.
nzidx           out_cnt, out_idx, a
// Non-zero index generation instruction. Write the
     indices of non-zero elements from the W dimension of
      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