

# 4 CLU

The TigerSHARC processor core contains two computation units known as compute blocks. Each compute block contains a register file and four independent computation units—an ALU, a CLU, a multiplier, and a shifter. The Communications Logic Unit (CLU) is highlighted in [Figure 4-1](#). The CLU takes its inputs from the register file, and returns its outputs to the register file. Most CLU instructions operate on the trellis registers (TR) and trellis history registers (THR).

This unit performs specialized communications functions, primarily to support decoding, CDMA despreading operations, and complex correlation. These functions may also be applied in non-communications algorithms.

This chapter contains:

- [“CLU Operations” on page 4-4](#)
- [“CLU Examples” on page 4-39](#)
- [“CLU Instruction Summary” on page 4-40](#)

The inclusion of the CLU instructions simplifies the programming of these algorithms, yet still retains the flexibility of a software approach. In this way, it is easy to tune the algorithm according to a user’s specific requirements. Additionally, the instructions can be used for a variety of purposes; for example, the TMAX instruction, included to support the decoding of turbo codes, is also very useful in the decoding of low-density parity-check codes.

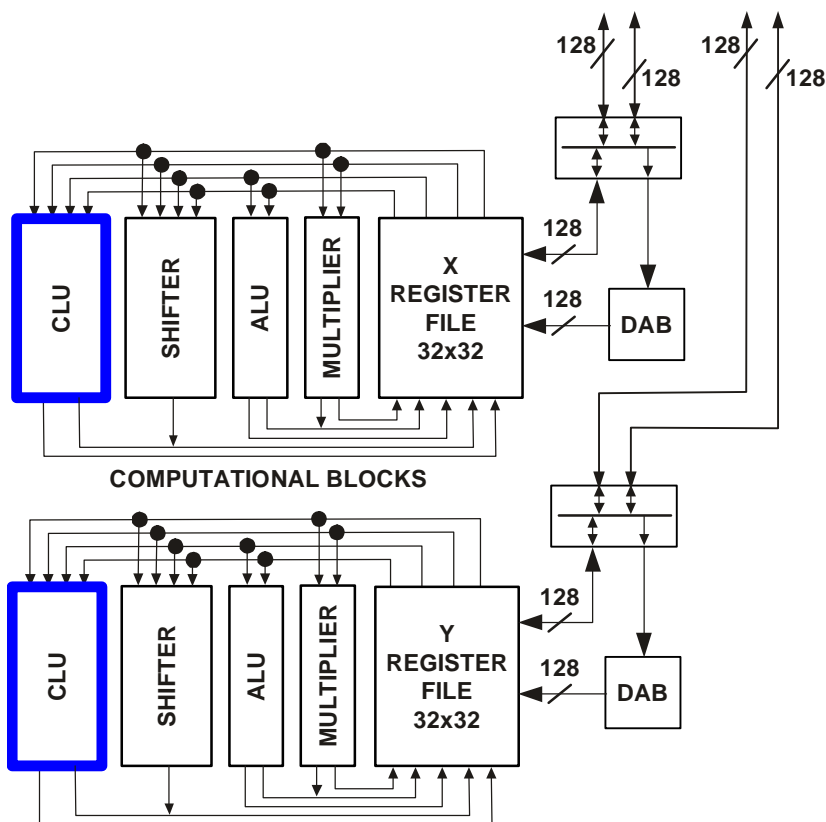


Figure 4-1. CLUs in Compute Block X and Y

The major strength of the TigerSHARC processor is the huge data transfer rate—two 128-bit memory accesses every cycle. For despreading, this enables 16 complex multiply-accumulate operations per cycle of 16-bit complex data (8-bit real, 8-bit imaginary). This enables calculation of a whole 16-bit 64-state trellis calculation every four cycles in both compute blocks together.

CLU operations are applied to fixed-point data. Relating CLU operations and supported real and complex data types shows that the 128-Bit CLU unit within each compute block supports:

- Maximum of values for Viterbi decode (VMAX)
- Jacobian logarithm for turbo decode (TMAX)
- CDMA despreader (DESPREAD)
- CDMA cross correlations (XCORRS)
- Polynomial reordering (PERMUTE)
- Trellis add/compare/select (ACS)

Examining the supported operands for each operation shows that the CLU operations support these data types:

- One or two 32-bit operands
- Two or four 16-bit operands
- Four or eight 8-bit operands
- Output 8-, 16-, or 32-bit results

Within instructions, the register name syntax identifies the input operand and output result data size and type. For more information on data size and type selection for CLU instructions, see [“Register File Registers” on page 2-5](#).

The remainder of this chapter presents descriptions of CLU instructions, options, and results using instruction syntax. For an explanation of the instruction syntax conventions used in CLU and other instructions, see [“Instruction Line Syntax and Structure” on page 1-22](#). For a list of CLU instructions and their syntax, see [“CLU Instruction Summary” on page 4-40](#).

# CLU Operations

The CLU performs communications logic operations on fixed-point data. The ADSP-TS201 processor uses compute block registers for the input operands and output result from CLU operations. The CLU instructions refer to three types of registers:

- $R_{m,n,s}$ —register file (data) registers
- $TR_{m,n,s}$ —32 (trellis) registers that are dedicated to the CLU instructions
- $THR$ —four (trellis history) registers used by the ACS, DESPREAD, and XCORRS instructions for shifted data
- $CMCTL$ —communications control registers used by the XCORRS instruction as an optional source for CUT value

For more information on the register files and register naming syntax for selecting data type and width, see [“Register File Registers” on page 2-5](#).

This section describes:

- [“TMAX Function” on page 4-5](#)
- [“Trellis Function” on page 4-7](#)
- [“Despread Function” on page 4-18](#)
- [“Cross Correlations Function” on page 4-26](#)
- [“CLU Instruction Options” on page 4-37](#)
- [“CLU Execution Status” on page 4-37](#)

For turbo and Viterbi decoding, the communications logic unit (CLU) input data sizes of 8- and 16-bit soft values are supported; output value data sizes of 16 and 32 bits are supported. Care should be taken when choosing the data size to prevent overflow in the calculation.

The `DESPREAD` function works with 16-bit complex numbers. Each 16-bit complex is composed of the real part (bits 7–0) and the imaginary part (bits 15–8). The result is always one or two complex words, each consisting of two shorts. Bits 15–0 represent the real part, and bits 31–16 represent the imaginary part (as complex numbers in the multiplier).

All CLU instructions generate status flags to indicate the status of the result. Because multiple operations are occurring in a parallel instruction, the value of the flag is an ORing of the results of all of the operations. For more information on CLU status, see [“CLU Execution Status” on page 4-37](#).

## TMAX Function

The `TMAX` function is commonly used in the decoding of turbo codes and other high-performance error-correcting codes. The function is:

$$TMAX(a, b) = \max(a, b) + \ln(1 + e^{-|a-b|})$$

The second term is implemented as a table:

$$\ln(1 + e^{-|a-b|})$$

(for large  $|a - b|$ ,  $\ln$  of 1 is 0).

The `TMAX` table input is the result of the subtraction on clock high of the `execute1` (EX1) pipe stage. If the result is negative, it is inverted (assuming that the difference between one’s complement and two’s complement is within the allowed error). The input to [Table 4-1](#) is the seven LSB’s of the compare subtract result. If the compare subtract result is larger than seven bits, the output is zero. Note that the implied decimal point is always placed before the five least significant bits.

## CLU Operations

Table 4-1 shows the  $T_{MAX}$  values. The maximum output error is one LSB.

Table 4-1.  $T_{MAX}$  Values

Negative Input (Binary)	Positive Input (Binary)	Output (Binary)
11..111.1111X	000.0000X	000.10110
11..111.1110X	000.0001X	000.10101
11..111.1101X	000.0010X	000.10100
11..111.1100X	000.0011X	000.10011
11..111.1011X	000.0100X	000.10010
11..111.1010X	000.0101X	000.10001
11..111.100XX	000.011XX	000.10000
11..111.011XX	000.100XX	000.01110
11..111.010XX	000.101XX	000.01101
11..111.001XX	000.110XX	000.01100
11..111.000XX	000.111XX	000.01011
11..110.111XX	001.000XX	000.01010
11..110.110XX	001.001XX	000.01000
11..110.10XXX	001.01XXX	000.00111
11..110.01XXX	001.10XXX	000.00110
11..110.00XXX	001.11XXX	000.00101
11..101.1XXXX	010.0XXXX	000.00011
11..101.0XXXX	010.1XXXX	000.00010
11..100.XXXXX	011.XXXXX	000.00001
<= 11..10XX.XXXXX	>= 1XX.XXXXX	000.00000

## Trellis Function

The trellis diagram (Figure 4-2) is a widely-used tool in communications systems. For example, the Viterbi and turbo decoding algorithms both operate on trellises. The ADSP-TS201 processor provides specialized instructions for trellises with binary transitions and up to eight states. Trellis with larger numbers of states can often be broken up into subtrellises with eight states or fewer and then applied to these instructions.

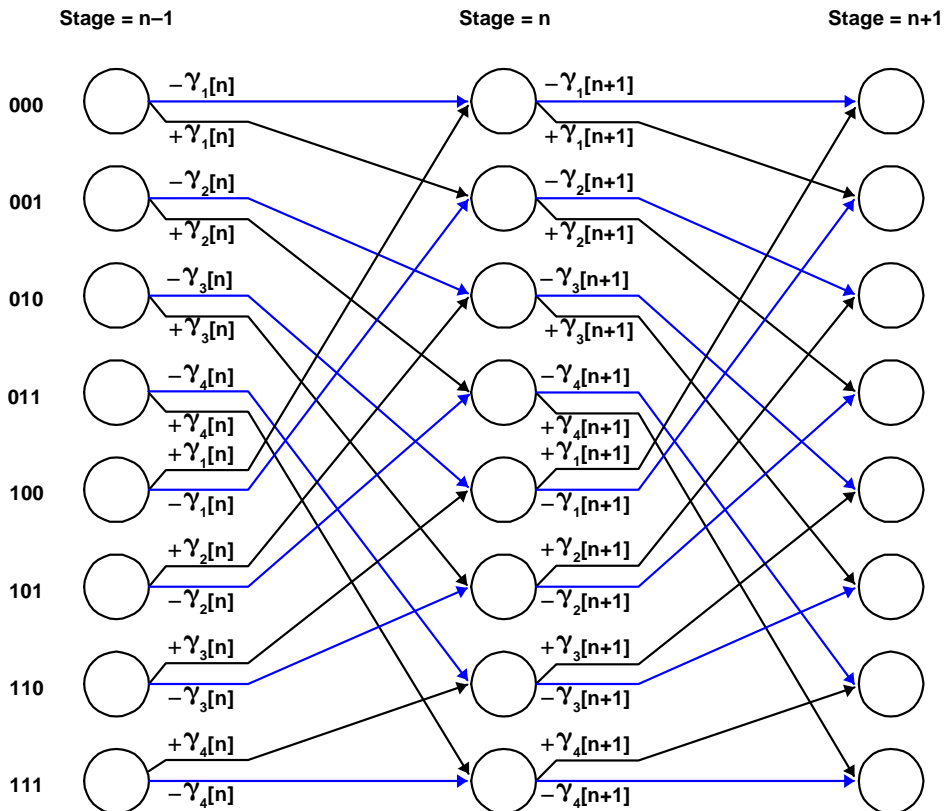


Figure 4-2. Trellis Diagram

## CLU Operations

A typical trellis diagram (shown in [Figure 4-2](#)) represents the set of possible state transitions from one stage to the next.

At each stage  $n$ , we need to compute the accumulated metrics for every state. For a particular state, this value will depend on the metrics of each of its two possible previous states (at stage  $n-1$ ) as well as transition metrics  $\gamma[n]$ . ( $\text{gamma}[n]$ ) corresponding to particular input/output combinations.

The particular symmetry among the transition metrics shown in [Figure 4-2](#) is typical for practical error-correcting codes.

The ACS (Add-Compare-Select) instruction has options for two types of state metric computations. In the Viterbi algorithm, the metrics for each of the two possible previous state are updated, and the one with maximum value is selected. For example, the metric for state "100" (binary for 4) at stage  $n$  is computed as:

$$\begin{aligned} \text{Metrics ("100", } n) = \\ \max ((\text{Metrics("010", } n-1) - \gamma_3(n)), \\ (\text{Metrics("110", } n-1) + \gamma_3(n))) \end{aligned}$$

The ACS instruction computes the metrics for four or eight states in parallel, and additionally records information specifying the selected transitions for use in a trace back routine.

A second option, used in turbo decoding, replaces the MAX operation above with the TMAX operation defined in ["TMAX Function" on page 4-5](#).

### Trellis Function of the Form

$$\text{STRsd} = \text{TMAX}(\text{TRmd} + \text{Rmq}_h, \text{TRnd} + \text{Rmq}_l) ;$$

High part of  $\text{Rmq}$  is added to  $\text{TRmd}$ , low part of  $\text{Rmq}$  is added to  $\text{TRnd}$ , and TMAX function is executed between both add results, as illustrated in [Figure 4-3](#) and [Figure 4-4](#).



Saturation is supported in this instruction. For more details, see [“Saturation Option” on page 3-8](#).

**i** On previous TigerSHARC processors, this instruction could not be executed in parallel to CLU and ALU instructions of the same compute block. This limitation does not apply to the ADSP-TS201 processor.

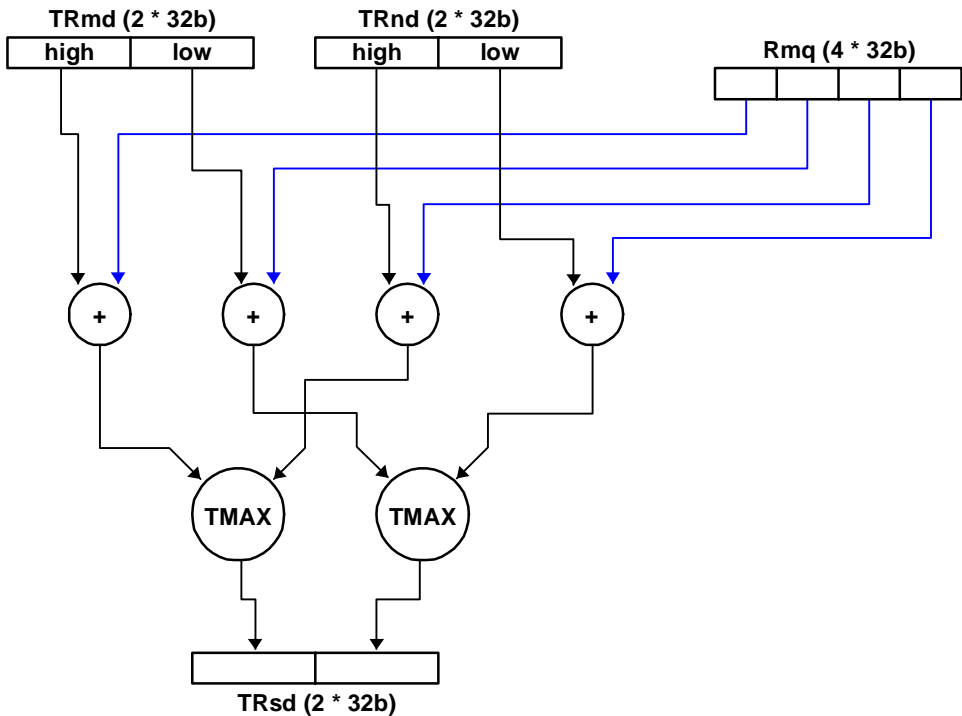


Figure 4-3.  $TRsd = TMAX(TRmd + Rmq\_h, TRnd + Rmq\_l)$

## CLU Operations

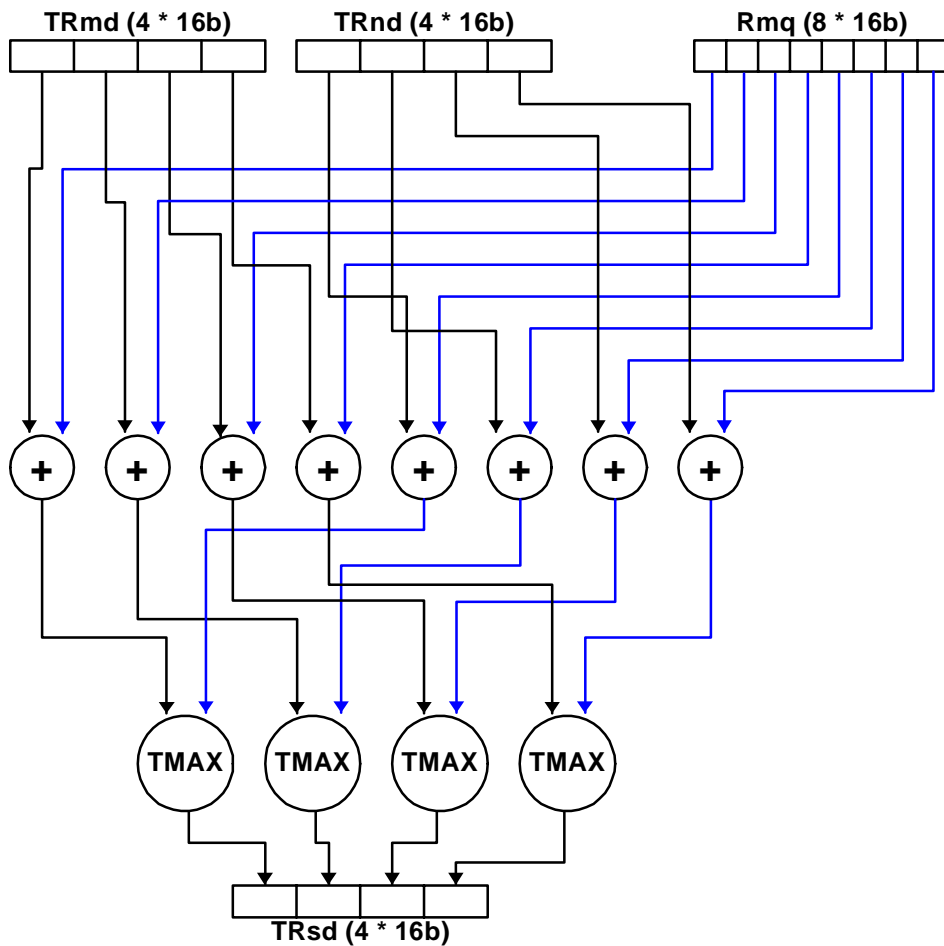



Figure 4-4.  $STRsd = TMAX(TRmd + Rmq\_h, TRnd + Rmq\_l)$

### Trellis Function of the Form

$STRsd = TMAX(TRmd - Rmq\_h, TRnd - Rmq\_l) ;$

The high part of  $Rmq$  is subtracted from  $TRmd$ , low part of  $Rmq$  is subtracted from  $TRnd$ , and  $TMAX$  function is executed between both subtract results, as illustrated in [Figure 4-6](#) and [Figure 4-7](#). For subtraction, the order of operands appears in [Figure 4-5](#).

Saturation is supported in this instruction. For more details, see “[Saturation Option](#)” on page 3-8.

 On previous TigerSHARC processors, this instruction could not be executed in parallel to CLU and ALU instructions of the same compute block. This limitation does not apply to the ADSP-TS201 processor.

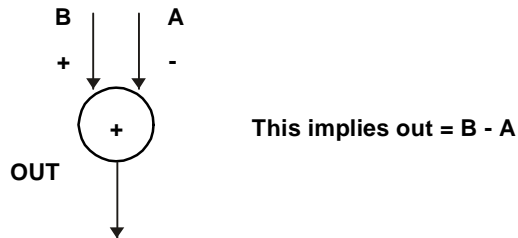


Figure 4-5. Order of Operands for Subtract Diagrams

## CLU Operations

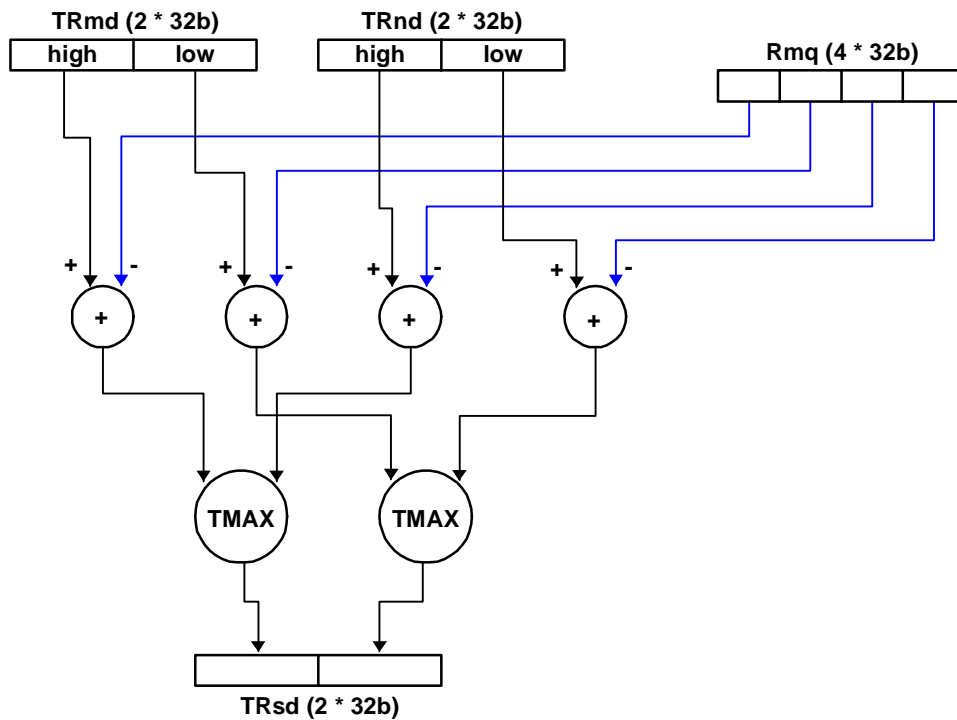


Figure 4-6.  $TRsd = TMAX(TRmd - Rmq\_h, TRnd - Rmq\_l)$

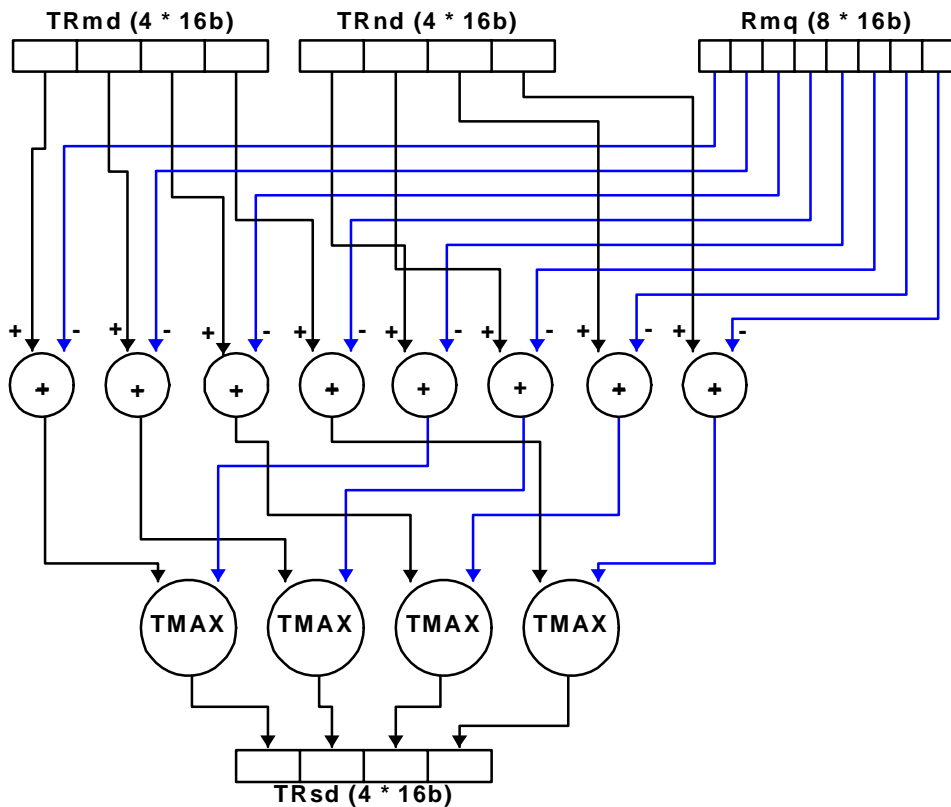


Figure 4-7.  $STRsd = TMAX(TRmd - Rmq\_h, TRnd - Rmq\_l)$

### Trellis Function of the Form

$$SRs = TMAX(TRm, TRn) ;$$

$TMAX$  function is executed between  $TRm$  and  $TRn$ , as illustrated in [Figure 4-8](#) and [Figure 4-9](#).

## CLU Operations

This instruction can be executed in parallel to ALU instructions, shifter instructions, multiplier instructions, and CLU register load. It cannot be executed in parallel to CLU instructions of the same compute block.

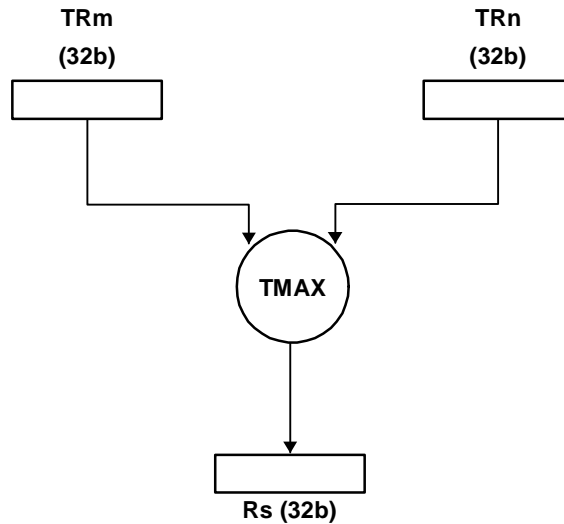


Figure 4-8.  $R_s = \text{TMAX}(\text{TR}_m, \text{TR}_n)$  instruction processing

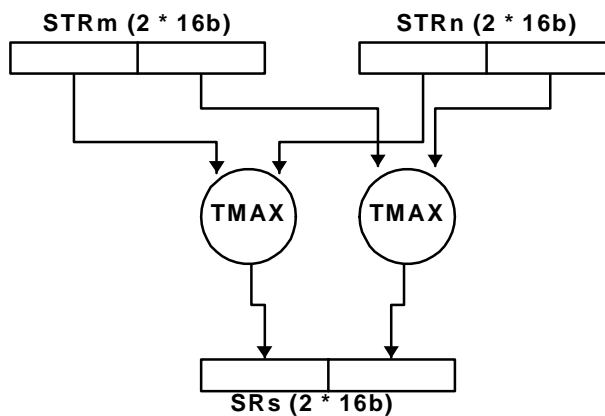


Figure 4-9.  $\text{SR}_s = \text{TMAX}(\text{TR}_m, \text{TR}_n)$

### Trellis Function of the Form

$$STRsd = \text{MAX}(TRmd + Rmq\_h, TRnd + Rmq\_l) ;$$

High part of  $Rmq$  is added to  $TRmd$ , low part of  $Rmq$  is added to  $TRnd$ , and selects the MAX between both add results, as illustrated in [Figure 4-10](#) and [Figure 4-11](#).

Saturation is provided in this instruction. For more details, see “[Saturation Option](#)” on [page 3-8](#).

**i** On previous TigerSHARC processors, this instruction could not be executed in parallel to CLU and ALU instructions of the same compute block. This limitation does not apply to the ADSP-TS201 processor.

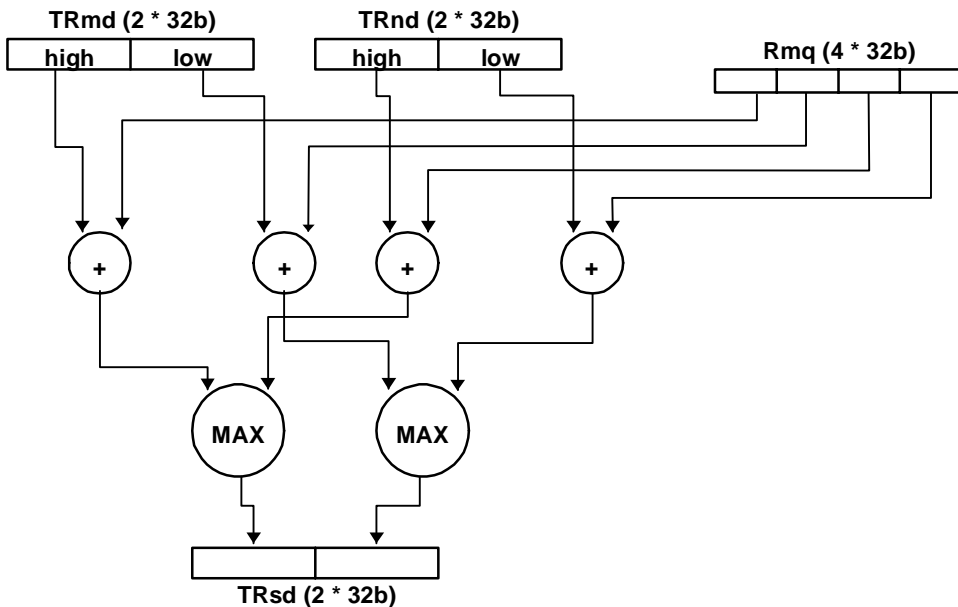


Figure 4-10.  $TRsd = \text{MAX}(TRmd + Rmq\_h, TRnd + Rmq\_l)$

## CLU Operations

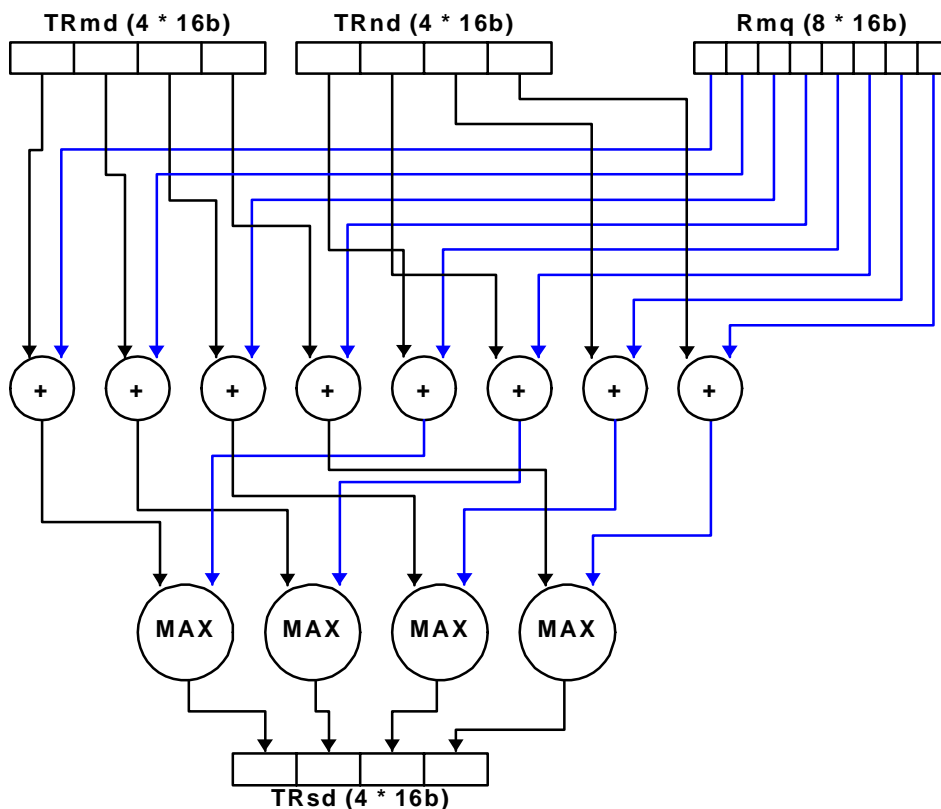


Figure 4-11.  $STRsd = \text{MAX}(TRmd + Rmq\_h, TRnd + Rmq\_l)$

### Trellis Function of the Form

$$STRsd = \text{MAX}(TRmd - Rmq\_h, TRnd - Rmq\_l) ;$$

High part of  $Rmq$  is subtracted from  $TRmd$ , low part of  $Rmq$  is subtracted from  $TRnd$ , and selects the MAX between both subtract results, as illustrated in [Figure 4-12](#) and [Figure 4-13](#).

Saturation is provided in this instruction. For more details, see [“Saturation Option”](#) on page 3-8.



**i** On previous TigerSHARC processors, this instruction could not be executed in parallel to CLU and ALU instructions of the same compute block. This limitation does not apply to the ADSP-TS201 processor.

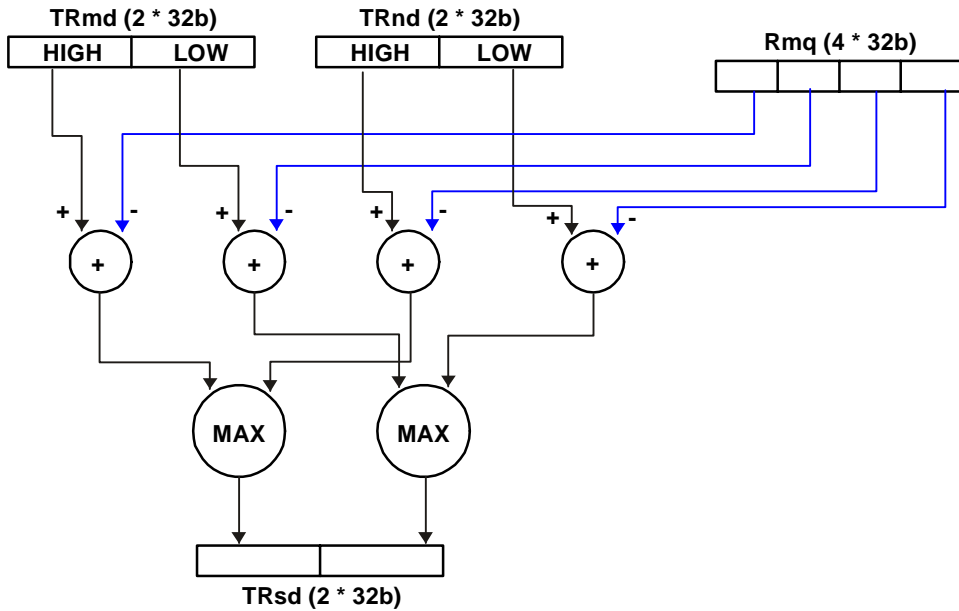


Figure 4-12.  $TRsd = \text{MAX}(TRmd - Rmq\_h, TRnd - Rmq\_l)$

## CLU Operations

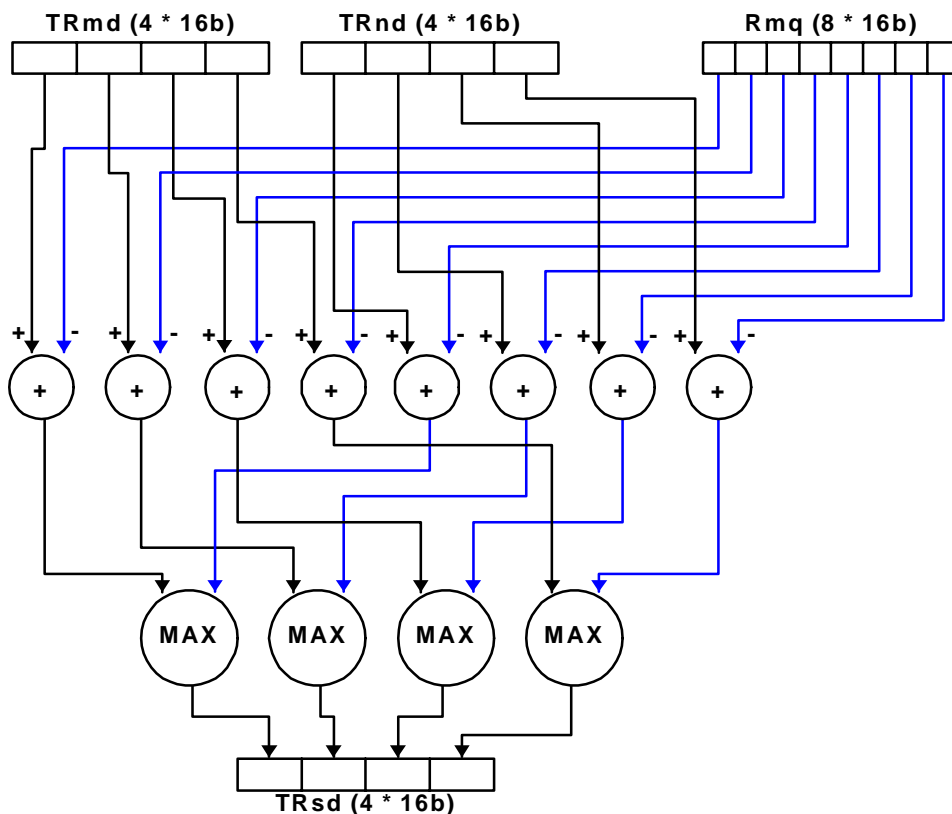


Figure 4-13.  $STRsd = \text{MAX}(TRmd - Rmq\_h, TRnd - Rmq\_l)$

## Despread Function

The `DESPREAD` instruction implements a highly parallel complex multiply-and-accumulate operation that is optimized for CDMA systems. Despreading involves computing samples of a correlation between complex input data and a precomputed complex spreading/scrambling code sequence.

The input data consists of samples with 8-bit real and imaginary parts. The code sequence samples, on the other hand, are always members of  $\{1+j, -1+j, -1-j, 1-j\}$ , and are therefore specified by 1-bit real and imaginary parts. The `DESPREAD` instruction takes advantage of this property and is able to compute eight parallel complex multiply-and-accumulates in each block in a single cycle.

The `DESPREAD` instruction supports accumulations over lengths (spread factors) of four, eight, and multiples of eight samples.

The `DESPREAD` input register *Rmq* is composed of 8 complex shorts - D7 to D0, in which each complex number is composed of 2 bytes. The most significant byte is the imaginary, and the least significant is the real. As shown in [Figure 4-14](#), Dn is composed of DnI and DnQ, where I denotes the real part and Q denotes the imaginary part.

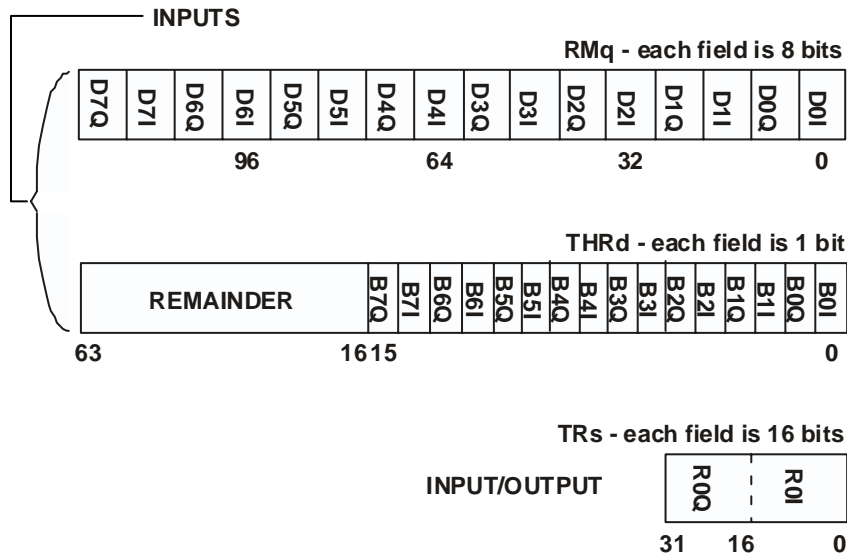


Figure 4-14. Bit field in registers for `TRs += DESPREAD (Rmq, THRd)` ;

## CLU Operations

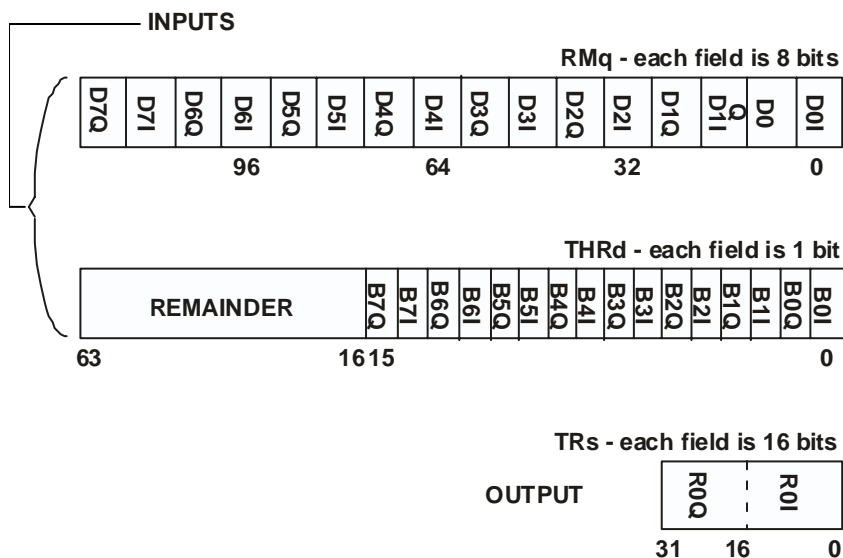


Figure 4-15. Bit field in registers for:  
Rs = TRs, TRs = DESPREAD (Rmq, THRd) ;

The *THRd* register is composed of 8 complex numbers {B7...B0} and 48 Remainder bits. Each complex number is composed of one real bit (least significant) and one imaginary bit. Each bit represents the value of +1 (when clear) or -1 (when set). *THRd* is post-shifted right by 16 bits so that the lowest 16 bits of the remainder may be used for a despread on the next cycle.

Saturation is provided in this instruction. For more details, see [“Saturation Option” on page 3-8](#).

- ❗ On previous TigerSHARC processors, this instruction could not be executed in parallel to ALU instructions of the same compute block. This limitation does not apply to the ADSP-TS201 processor.
- ❗ When you execute this instruction in parallel to `THR` register load the `THR` load instruction takes priority on the `THR` shift in this instruction.

### Despread Function of the Form

**`TRs += DESPREAD (Rmq, TH Rd) + TRs;`**

`TRs` is a complex word, composed of two shorts - real (least significant) and imaginary (most significant).

The function (illustrated in [Figure 4-16](#)) is:

$$\begin{aligned}RsI &= (\text{sum } (n = 0 \text{ to } 7) (BnI * DnI - BnQ * DnQ)) + TRsI \\RsQ &= (\text{sum } (n = 0 \text{ to } 7) (BnI * DnQ + BnQ * DnI)) + TRsQ\end{aligned}$$

The multiplication is by integer ( $\pm 1$ ), the result alignment is to least significant, and the sum is sign extended.

## CLU Operations

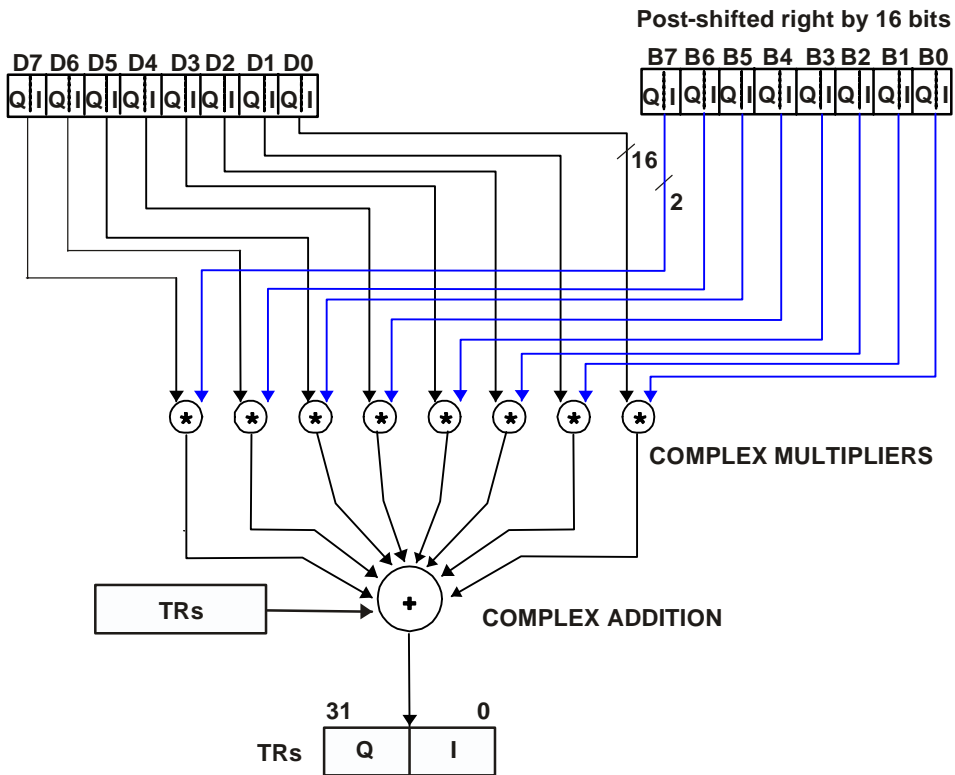


Figure 4-16.  $TRs += \text{DESPREAD}(Rmq, THrd)$  ;

### Despread Function of the Form

$Rs = TRs, TRs = \text{DESPREAD}(Rmq, THrd)$  ;

$TRs$  is a complex word, composed of two shorts - real (least significant) and imaginary (most significant).

The function (illustrated in Figure 4-17) is:

$$RsI = (\text{sum } (n = 0 \text{ to } 7) (BnI * DnI - BnQ * DnQ))$$

$$RsQ = (\text{sum } (n = 0 \text{ to } 7) (BnI * DnQ + BnQ * DnI))$$

The value of  $TRs$  before the operation is stored in  $Rs$ .

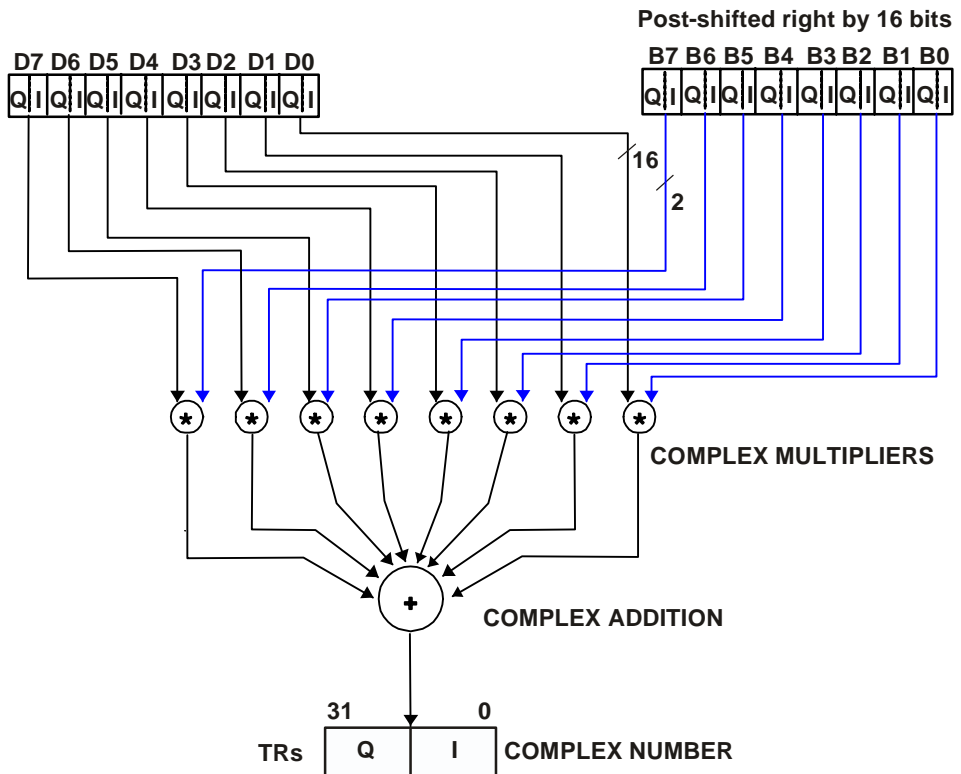


Figure 4-17.  $Rs = TRs$ ,  $TRs = \text{DESPREAD}(Rmq, THrd)$  ;

## CLU Operations

### Despread Function of the Form

$Rsd = TRsd, TRsd = \text{DESPREAD} (Rmq, THRd) ;$

The function (illustrated in [Figure 4-18](#) and [Figure 4-19](#)) is:

$$R0I = (\text{sum } (n = 0 \text{ to } 3) (BnI * DnI - BnQ * DnQ))$$

$$R0Q = (\text{sum } (n = 0 \text{ to } 3) (BnI * DnQ + BnQ * DnI))$$

$$R1I = (\text{sum } (n = 4 \text{ to } 7) (BnI * DnI - BnQ * DnQ))$$

$$R1Q = (\text{sum } (n = 4 \text{ to } 7) (BnI * DnQ + BnQ * DnI))$$

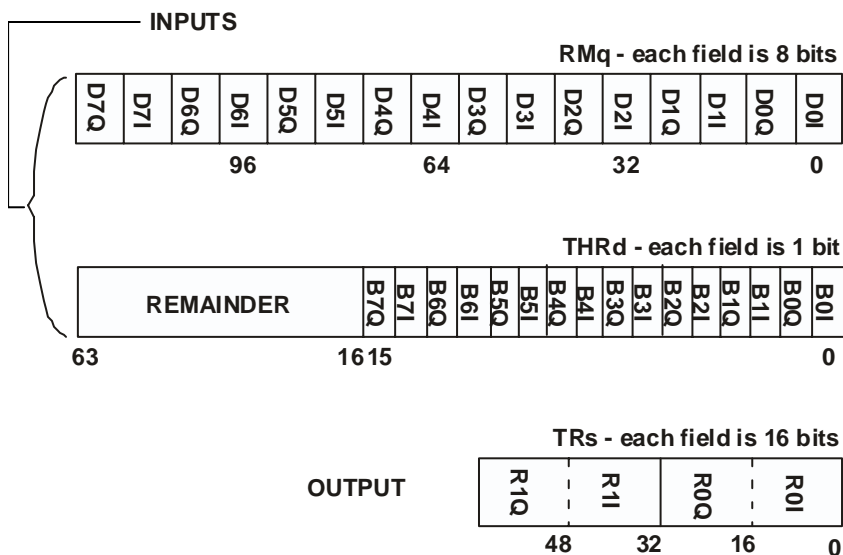


Figure 4-18. Bit fields in registers for:

$Rsd = TRsd, TRsd = \text{DESPREAD} (Rmq, THRd) ;$



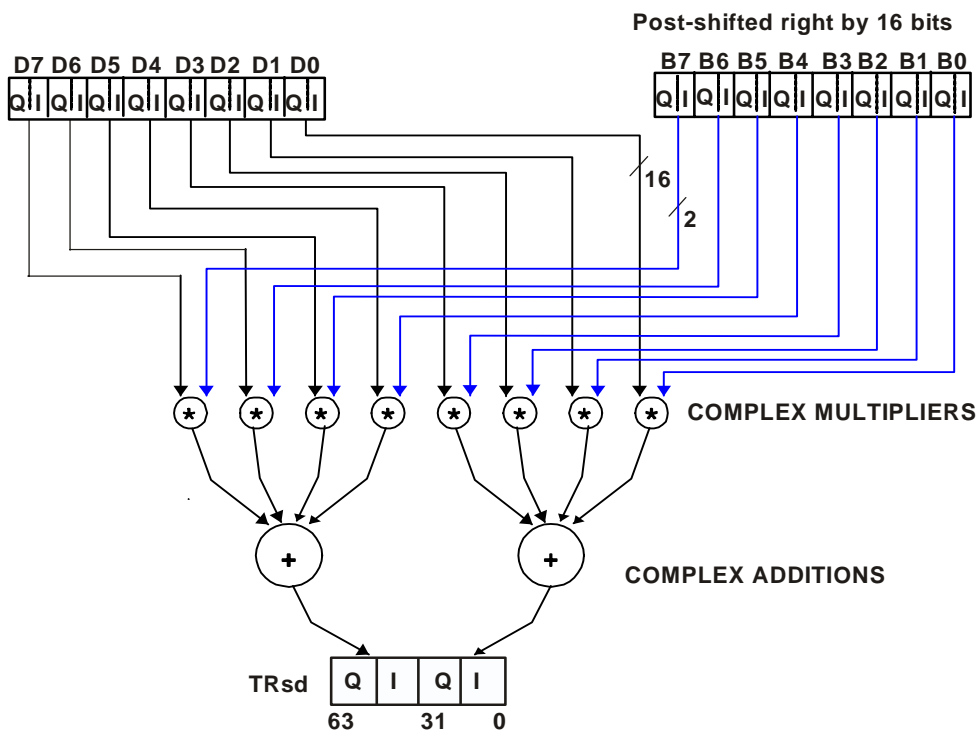


Figure 4-19.  $Rsd = TRsd, TRsd = DESPREAD(Rmq, THrd)$  ;

### Cross Correlations Function

The `XCORRS` instruction correlates long input sequence (such as 2048 complex input numbers of an 8 bit pilot) with a known reference sequence for multiple delays. It is convenient to view the `XCORRS` instruction as a single-cycle execution of 16 parallel `DESPREAD` instructions. Some important features of the `XCORRS` instruction include:

- Clear (`CLR`) option, providing a mechanism to clear the trellis registers before beginning a new cross correlation
- Cut inputs (`CUT`) option, providing a mechanism to discard some input data numbers (for initial and final cycles; lower and upper triangles)
- Extended-precision (`EXT`) option, supporting 16-bit input and 32-bit accumulation (instead of the default 8-bit input and 16-bit accumulation)
- Outputs correlation strength for each delay
- Memory usage; execution considers memory bandwidth and access to multiple blocks

The `XCORRS` equations for every part X or Y performs the operations shown in [Figure 4-20](#) where (depending on the `CUT` value) the inputs are as shown in [Figure 4-21](#). The `CUT` function values are as shown in [Figure 4-22](#).

```

for (k = 0; k ≤ 15; k++)
{
    TR[k] = TR[k] +  $\sum_{m=0}^7 D[m]C[m-k+15]f_{\text{CUT}}[m-k+15]$ ;
}
THR3:0 = THR3:0 >> 16;

```

Figure 4-20. XCORRS Equation

$$f_{\text{CUT}}[m-k+15] = +1 \text{ or } 0$$

$$C[m-k+15] = \pm 1 \pm i \text{ (2 bits)}$$

$$D[m] = (D_{\text{real}}[m] + iD_{\text{imaginary}}[m])(16 \text{ bits})$$

Figure 4-21. XCORRS Equation Inputs

$$f_{\text{CUT}}[w] = \begin{cases} 1 & \text{for } (w < \text{CUT}) \\ 0 & \text{for } (w \geq \text{CUT}) \end{cases}$$

Figure 4-22. XCORRS Equation, CUT Function Values

## CLU Operations

Figure 4-23 and Figure 4-24 show the bit placement for an `XCORRS` instruction using single-precision inputs (default operation. Note that Figure 4-23 and Figure 4-24 represent 16 multiply accumulate operations.

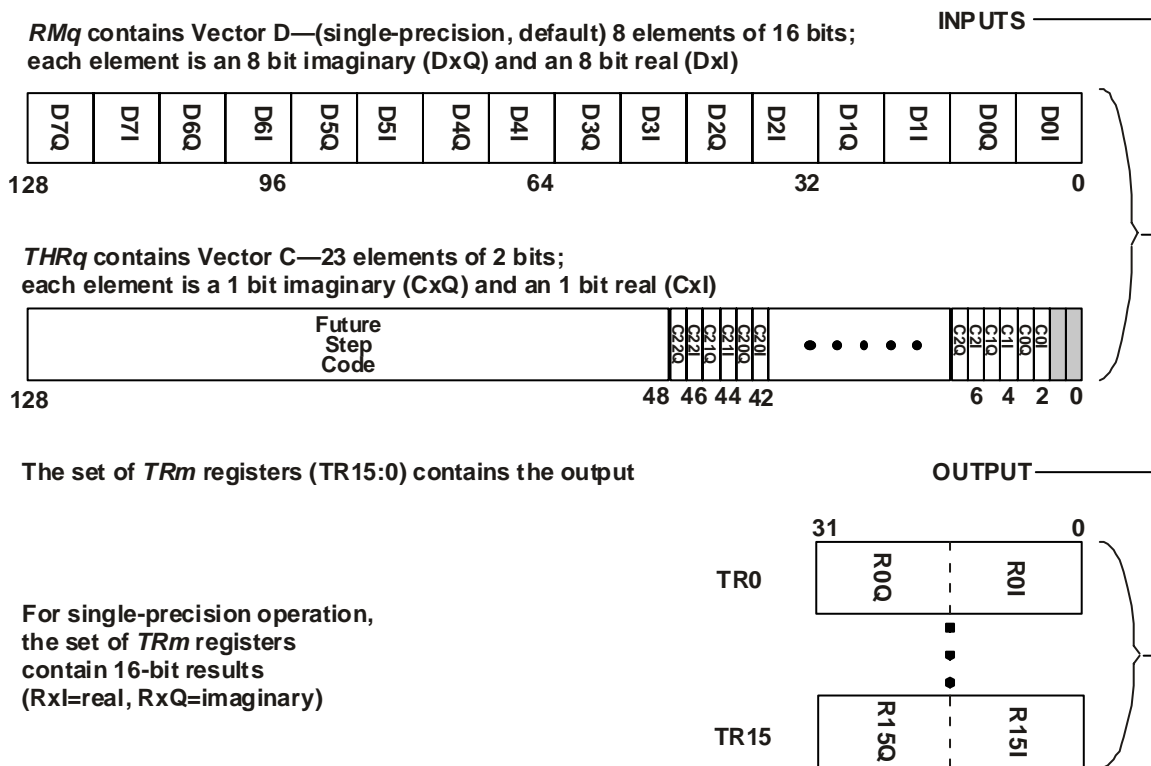


Figure 4-23. Bit fields in registers for:  
 $TRm = XCORRS(Rmq, THRd);$

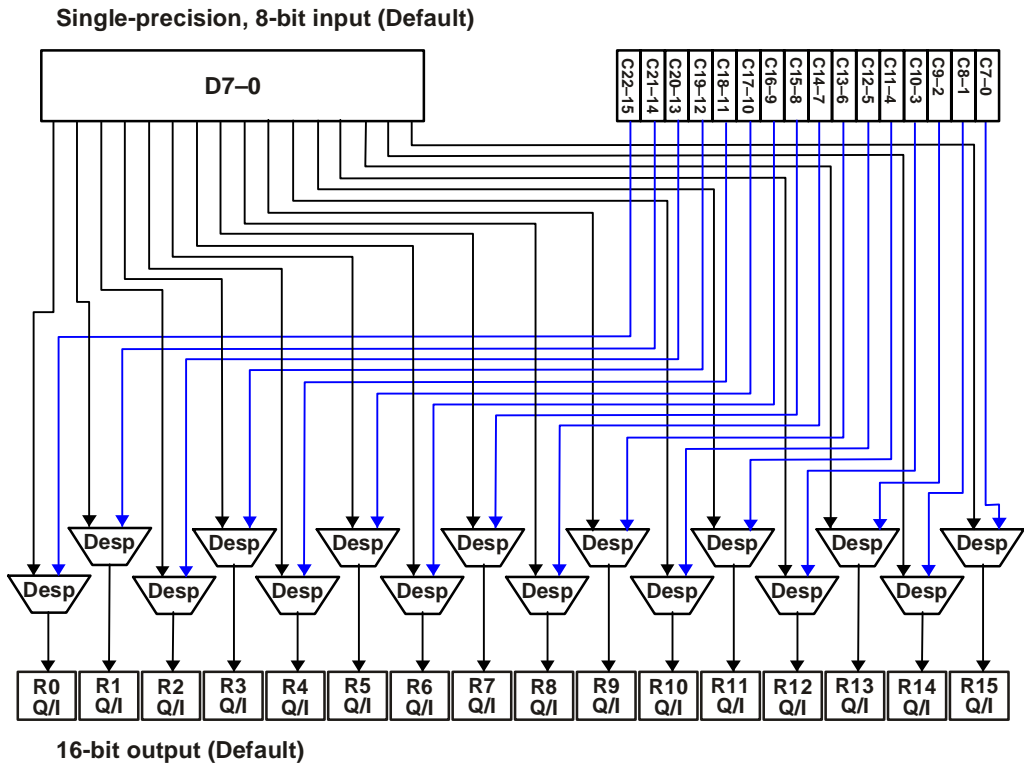


Figure 4-24.  $TR_m = XCORRS(R_{mq}, THR_d)$ ;

## CLU Operations

Figure 4-25 and Figure 4-26 show the bit placement for an `XCORRS` instruction using extended-precision inputs (EXT option operation. Note that Figure 4-25 and Figure 4-26 represent 8 multiply accumulate operations.

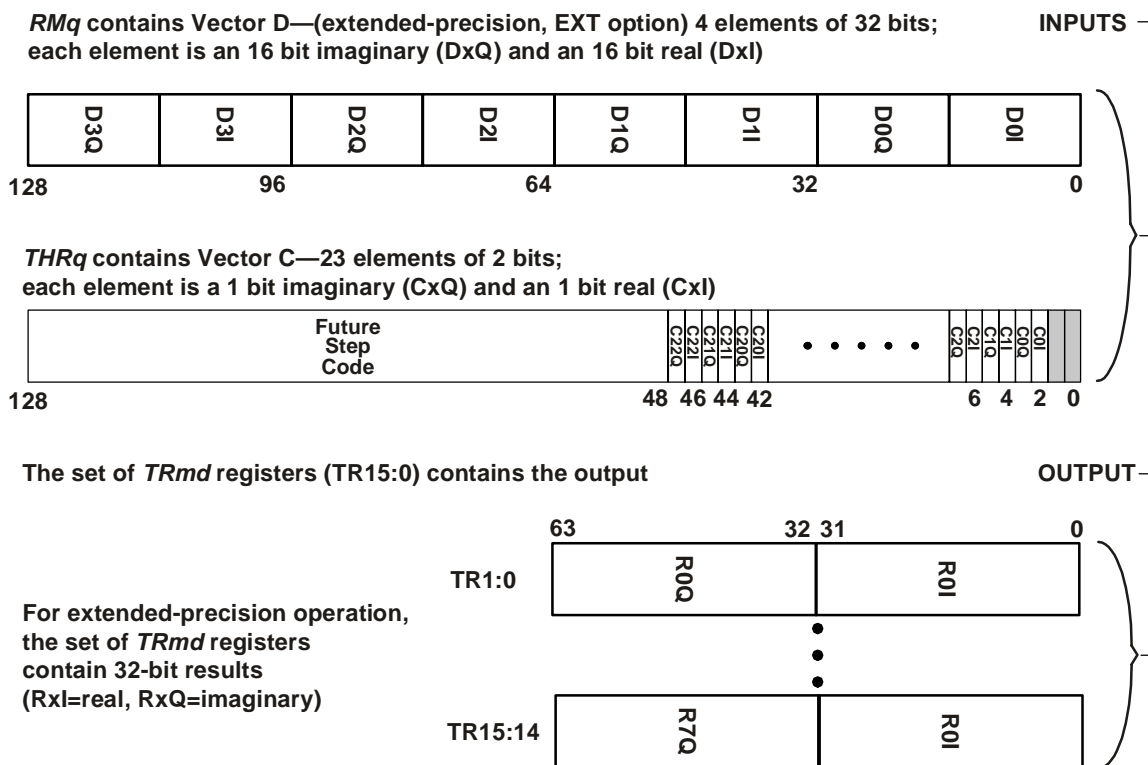


Figure 4-25. Bit fields in registers for:  
 $TRm = XCORRS(Rmq, THRd) (EXT);$

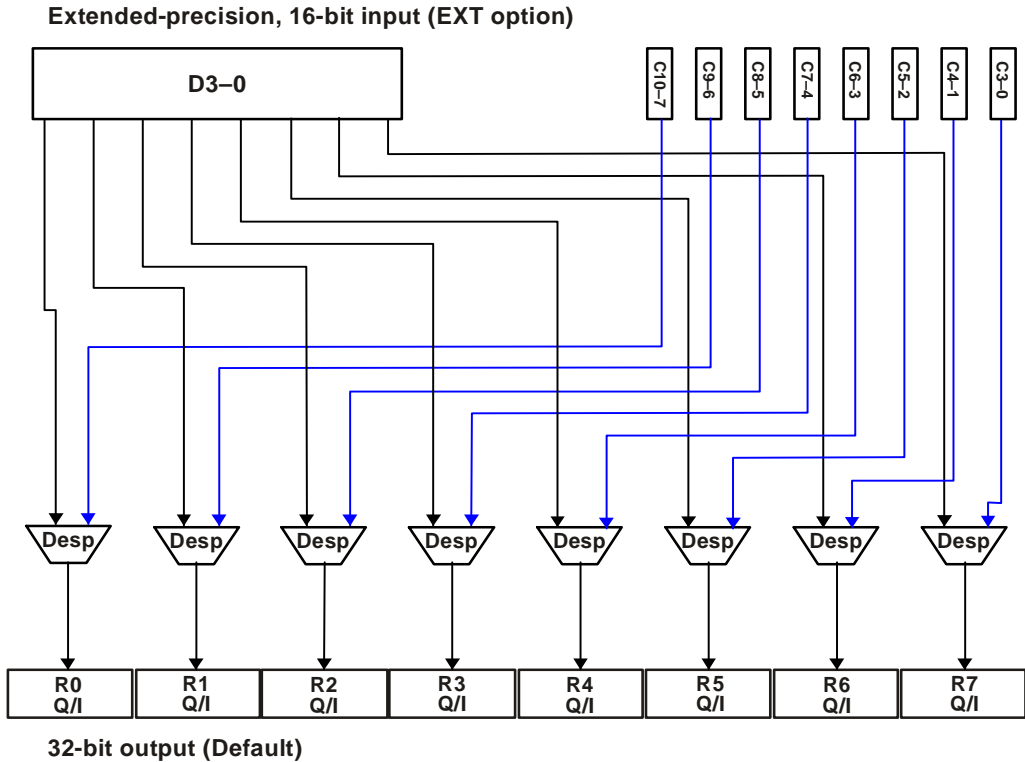


Figure 4-26.  $TR_m = XCORRS(R_{mq}, THR_d)$  (EXT);



When you execute this instruction in parallel to  $THR$  register load the  $THR$  load instruction takes priority on the  $THR$  shift in this instruction.

## CLU Operations

When the `CUT` option is omitted, the `CUT=0`, and the code index is as shown in [Figure 4-27](#). Note that each code index in [Figure 4-27](#) maps to a “C” input in [Figure 4-24 on page 4-29](#).

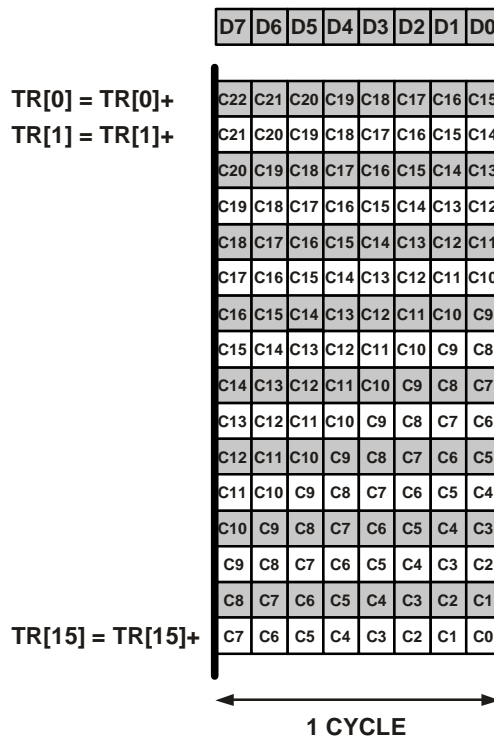


Figure 4-27. `CUT = 0` Definition – Code Index

When the `CUT` option is negative (for example, `CUT = -k`), all multiply-and-accumulate operations under `C[k]` are ignored.



Figure 4-28 shows an example with  $CUT = -7$ .

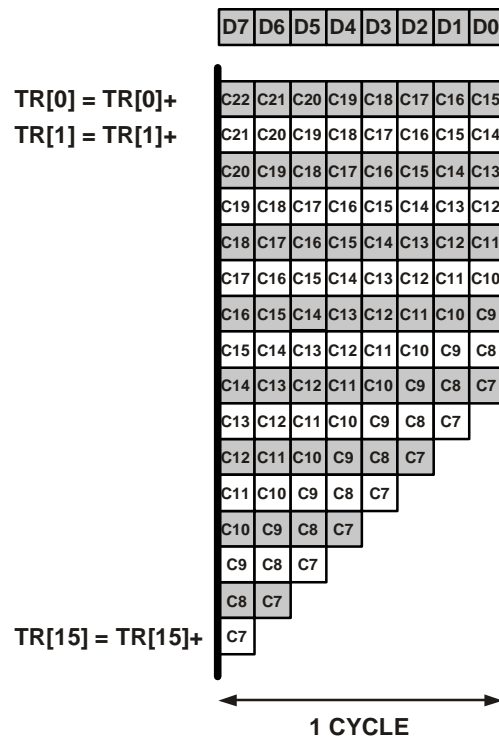


Figure 4-28.  $CUT = \text{Negative Example} - \text{Code Index}$

## CLU Operations

When the CUT option is positive (for example, CUT = +k), all multiply-and-accumulate operations for C[k] and above are ignored.

Figure 4-29 shows an example with  $\text{CUT} = +15$ .

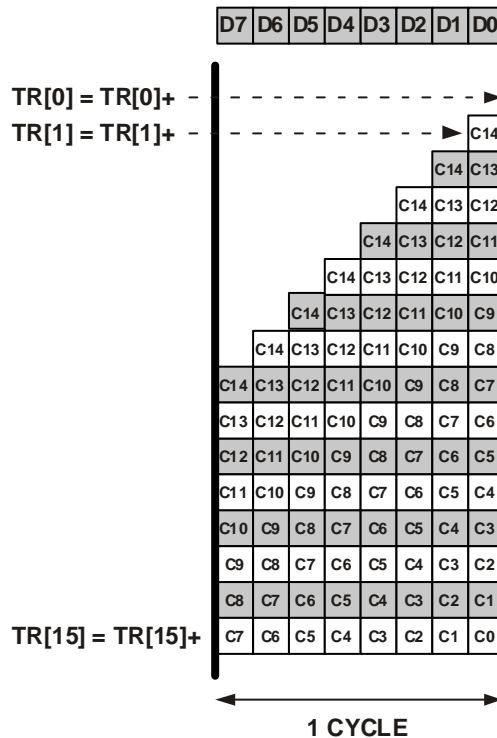


Figure 4-29. CUT = Positive Example – Code Index

When the TR registers are empty (for example, at the beginning of a new series of XCORRS instructions), the data fills the TR registers for the first two cycles as shown in Figure 4-30. This operation is similar to  $CUT = -15$  in cycle one and  $CUT = -7$  in cycle two.

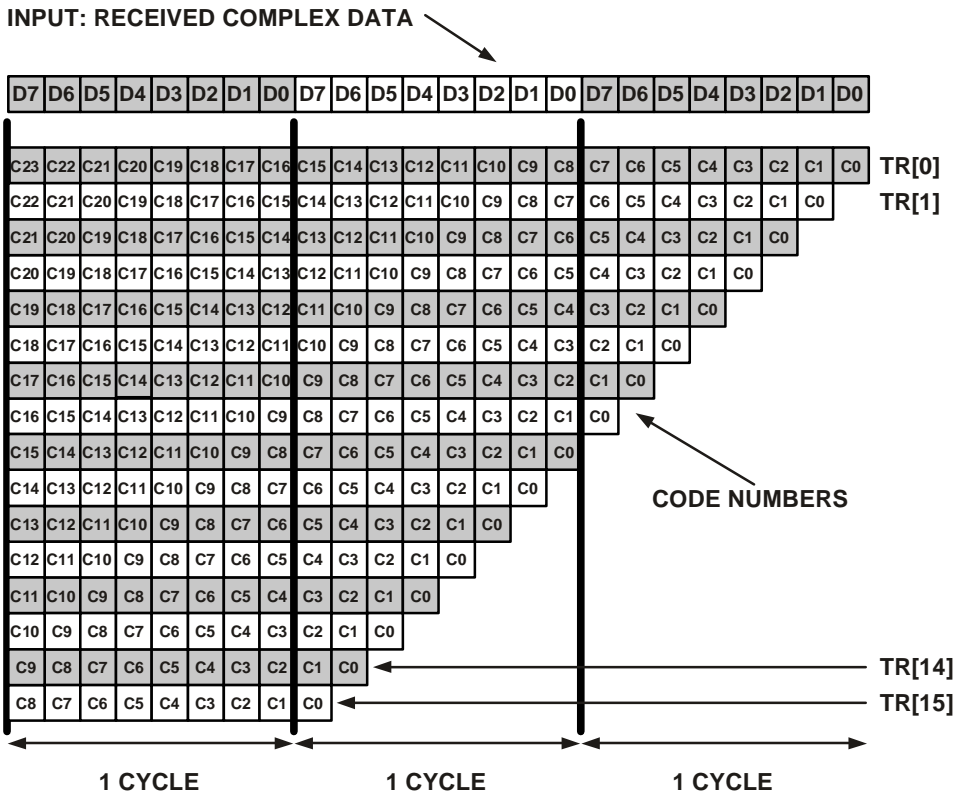


Figure 4-30. Code Index – Example Data Start

## CLU Operations

The `TR` register are emptied at the end of the data set (for example, at the end of a series of `XCORRS` instructions), the data empties the `TR` registers during the last two cycles as shown in [Figure 4-31](#). This operation is similar to `CUT = 16` in the second-to-last cycle one and `CUT = 8` in the last cycle.

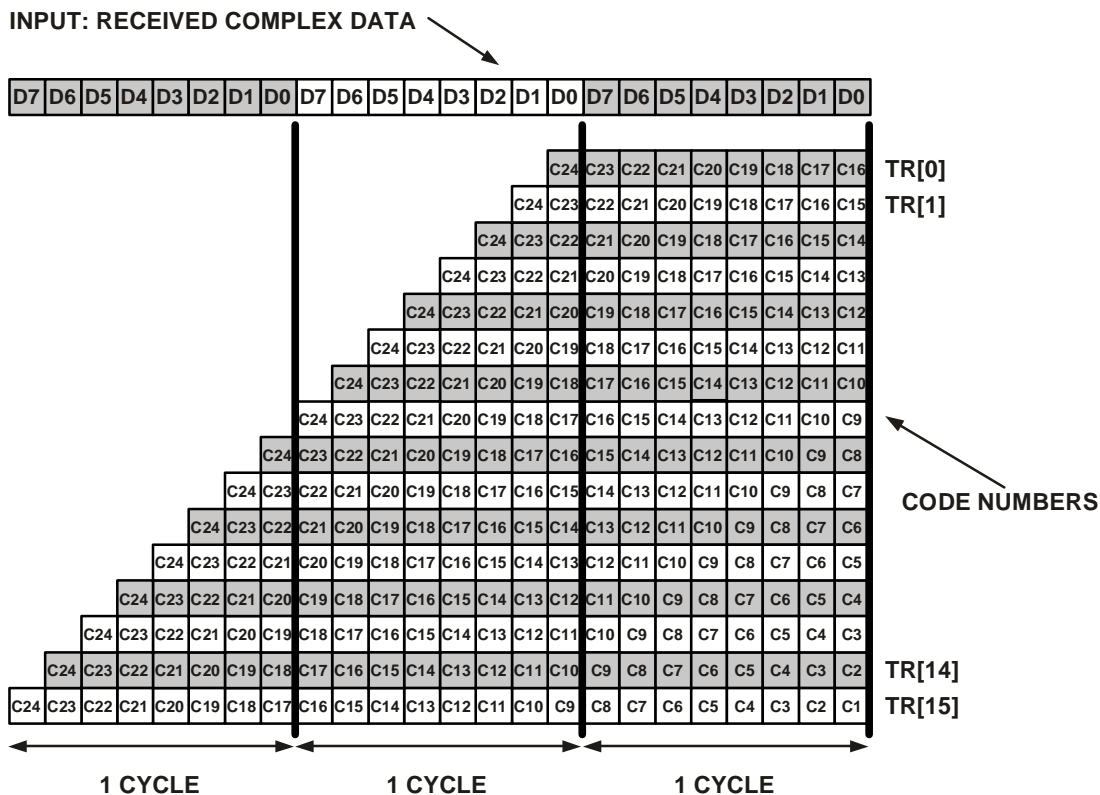


Figure 4-31. Code Index – Example Data End

## CLU Instruction Options

Some of the CLU instructions have options associated with them that permit flexibility in how the instructions execute. It is important to note that these options modify the detailed execution of instructions and options that are particular to a group of instructions—not all options are applicable to all instructions. Instruction options appear in parenthesis at the end of the instruction’s slot. For a list indicating which options apply for particular CLU instructions, see [“CLU Instruction Summary” on page 4-40](#). The CLU instruction options include:

- (TMAX) selects TMAX operation instead of MAX operation<sup>1</sup>
- (CUT *imm*) selects data to cut from cross correlation set<sup>2</sup>
- (CLR) clears the selected trellis register prior to cross correlation summation<sup>2</sup>
- (EXT) selects extended precision for cross correlation input/results<sup>2</sup>

The following examples are CLU instructions that demonstrate arithmetic operations with options applied.

/\* ADD EXAMPLES HERE\*/

## CLU Execution Status

CLU operations update status flags in the compute block’s Arithmetic Status (XSTAT and YSTAT) register (see [Figure 2-2 on page 2-4](#) and [Figure 2-3 on page 2-5](#)). Programs can use status flags to control execution of conditional instructions and initiate software exception interrupts.

---

<sup>1</sup> ACS instruction only

<sup>2</sup> XCORRS instruction only

## CLU Operations

Table 4-2 shows the flags in `XSTAT` or `YSTAT` that indicate CLU status (a 1 indicates the condition) for the most recent CLU operation.

Table 4-2. CLU Status Flags

Flag	Definition	Updated By...
TROV	CLU overflow	All CLU ops

CLU operations also update sticky status flags in the compute block's Arithmetic Status (`XSTAT` and `YSTAT`) register. Table 4-3 shows the flags in `XSTAT` or `YSTAT` that indicate CLU sticky status (a 1 indicates the condition) for the most recent CLU operation. Once set, a sticky flag remains high until explicitly cleared.

Table 4-3. CLU Status Sticky Flags

Flag	Definition	Updated By...
TRSOV	CLU overflow, sticky	All CLU ops

Flag update occurs at the end of each operation and is available on the next instruction slot. A program cannot write the arithmetic status register explicitly in the same cycle that the CLU is performing an operation.

Multi-operand instructions (for example,  $STRsd = TMAX(TRmd + Rmq\_h, TRnd + Rmq\_l)$  ;) produce multiple sets of results. In this case, the DSP determines a flag by ORing the result flag values from individual results.

## CLU Examples

The communications logic unit (CLU) instructions are designed to support different algorithms used for communications applications. These instructions were designed primarily with the following algorithms in mind (although many other uses are possible):

- Viterbi Decoding
- Decoding of turbo codes
- Despreading for code-division multiple access (CDMA) systems

[Listing 4-1](#) provides a number of example CLU arithmetic instructions. The comments with the instructions identify the key features of the instruction, such as input operand size and register usage.

### Listing 4-1. CLU Instruction Examples

```
/*  
    Examples are to be added in future revision of this document.  
*/
```

# CLU Instruction Summary

[Listing 4-2](#) “CLU Instructions” shows the CLU instructions’ syntax. The conventions used in these listings for representing register names, optional items, and choices are covered in detail in [“Register File Registers” on page 2-5](#). Briefly, these conventions are:

- { } – the curly braces enclose options; these braces are not part of the instruction syntax.
- | – the vertical bars separate choices; these bars are not part of the instruction syntax.
- *Rmd* – the register names in italic represent user-selectable single (*Rs*, *Rm*, or *Rn*), double (*Rsd*, *Rmd*, *Rnd*) or quad (*Rsq*) register names.



Each instruction presented here occupies one instruction slot in an instruction line. For more information about instruction lines and instruction combination constraints, see [“Instruction Line Syntax and Structure” on page 1-22](#) and [“Instruction Parallelism Rules” on page 1-26](#).

Listing 4-2. CLU Instructions

```
{X|Y|XY}{S}TRsd = TMAX(TRmd + Rmq_h, TRnd + Rmq_l) ;
{X|Y|XY}{S}TRsd = TMAX(TRmd - Rmq_h, TRnd - Rmq_l) ;
{X|Y|XY}{S}Rs = TMAX(TRm, TRn) ;
{X|Y|XY}{S}TRsd = MAX(TRmd + Rmq_h, TRnd + Rmq_l) ;
{X|Y|XY}{S}TRsd = MAX(TRmd - Rmq_h, TRnd - Rmq_l) ;
{X|Y|XY}Rs = TRm ;
{X|Y|XY}Rsd = TRmd ;
{X|Y|XY}Rsq = TRmq ;
{X|Y|XY}TRs = Rm ;
{X|Y|XY}TRsd = Rmd ;
{X|Y|XY}TRsq = Rmq ;
```



```

{X|Y|XY}Rs = THRm ;
{X|Y|XY}Rsd = THRmd ;
{X|Y|XY}Rsqr = THRmq ;1
{X|Y|XY}THRs = Rm ;
{X|Y|XY}THRsqr = Rmd {(i)} ;
{X|Y|XY}THRsqr = Rmq ;1
{X|Y|XY}TRs = DESPREAD (Rmq, THRd) + TRn ;
{X|Y|XY}Rs = TRs, TRs = DESPREAD (Rmq, THRd) ; (dual op)
{X|Y|XY}Rsd = TRsd, TRsd = DESPREAD (Rmq, THRd) ; (dual op)
{X|Y|XY}TRsa = XCORRS (Rmq, THRnq) {(CUT <Imm>|R)}{(CLR)}{(EXT)};
{X|Y|XY}Rsqr = TRbq, TRsa = XCORRS (Rmq, THRnq)
    {(CUT <Imm>|R)}{(CLR)}{(EXT)} ; (dual operation)
/* where TRsa = TR15:0 or TR31:16 */
{X|Y|XY}{S}TRsqr = ACS (TRmd, TRnd, Rm) {(TMAX)} ;
{X|Y|XY}Rsqr = TRaq, {S}TRsqr = ACS (TRmd, TRnd, Rm)
    {(TMAX)} ; (dual op)
{X|Y|XY}Rsd = PERMUTE (Rmd, Rn) ;
{X|Y|XY}Rsqr = PERMUTE (Rmd, -Rmd, Rn) ;

```

---

<sup>1</sup> Not implemented, but syntax reserved

