# CHAPTER 10 SSE4.2 AND SIMD PROGRAMMING FOR TEXT-PROCESSING/LEXING/PARSING

String/text processing spans a discipline that often employs techniques different from traditional SIMD integer vector processing. Much of the traditional string/text algorithms are character based, where characters may be represented by encodings (or code points) of fixed or variable byte sizes. Textual data represents a vast amount of raw data and often carrying contextual information. The contextual information embedded in raw textual data often requires algorithmic processing dealing with a wide range of attributes, such as character values, character positions, character encoding formats, subsetting of character sets, strings of explicit or implicit lengths, tokens, delimiters; contextual objects may be represented by sequential characters within a pre-defined character subsets (e.g. decimal-valued strings); textual streams may contain embedded state transitions separating objects of different contexts (e.g. tag-delimited fields).

Traditional Integer SIMD vector instructions may, in some simpler situations, be successful to speed up simple string processing functions. SSE4.2 includes four new instructions that offer advances to computational algorithms targeting string/text processing, lexing and parsing of either unstructured or structured textual data.

## 10.1 SSE4.2 STRING AND TEXT INSTRUCTIONS

SSE4.2 provides four instructions, PCMPESTRI/PCMPESTRM/PCMPISTRI/PCMPISTRM that can accelerate string and text processing by combining the efficiency of SIMD programming techniques and the lexical primitives that are embedded in these 4 instructions. Simple examples of these instructions include string length determination, direct string comparison, string case handling, delimiter/token processing, locating word boundaries, locating sub-string matches in large text blocks. Sophisticated application of SSE4.2 can accelerate XML parsing and Schema validation.

Processor's support for SSE4.2 is indicated by the feature flag value returned in ECX [bit 20] after executing CPUID instruction with EAX input value of 1 ( i.e. SSE4.2 is supported if CPUID.01H:ECX.SSE4_2 [bit 20] = 1). Therefore, software must verify CPUID.01H:ECX.SSE4_2 [bit 20] is set before using these 4 instructions. (Verifying CPUID.01H:ECX.SSE4_2 = 1 is also required before using PCMPGTQ or CRC32. Verifying CPUID.01H:ECX.POPCNT[Bit 23] = 1 is required before using the POPCNT instruction.)

These string/text processing instructions work by performing up to 256 comparison operations on text fragments. Each text fragment can be 16 bytes. They can handle fragments of different formats: either byte or word elements. Each of these four instructions can be configured to perform four types of parallel comparison operation on two text fragments.

The aggregated intermediate result of a parallel comparison of two text fragments become a bit patterns:16 bits for processing byte elements or 8 bits for word elements. These instruction provide additional flexibility, using bit fields in the immediate operand of the instruction syntax, to configure an unary transformation (polarity) on the first intermediate result.

Lastly, the instruction's immediate operand offers a output selection control to further configure the flexibility of the final result produced by the instruction. The rich configurability of these instruction is summarized in Figure 10-1.

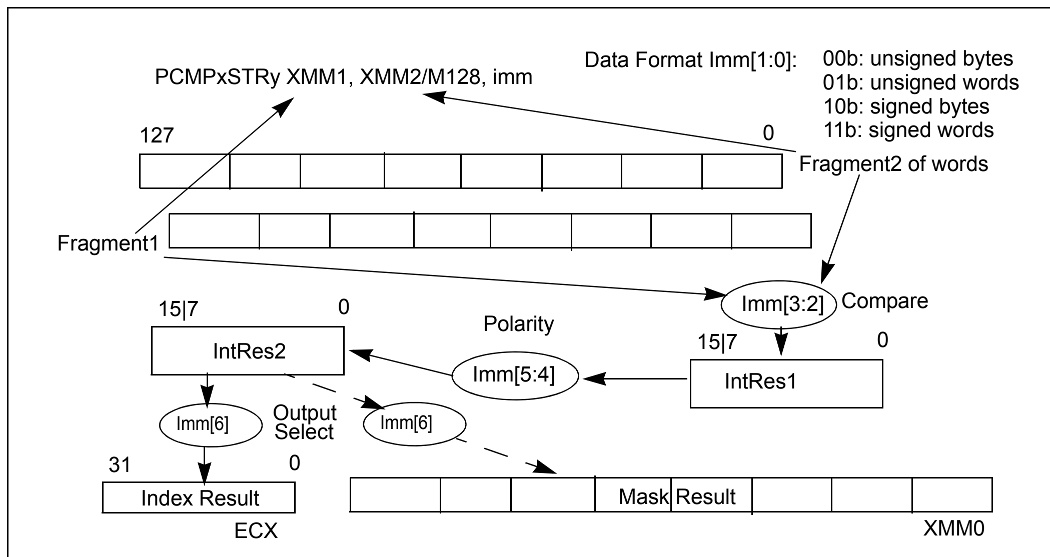


Figure 10-1. SSE4.2 String/Text Instruction Immediate Operand Control

The PCMPxSTRI instructions produce final result as an integer index in ECX, the PCMPxSTRM instructions produce final result as a bit mask in the XMM0 register. The PCMPISTRy instructions support processing string/text fragments using implicit length control via null termination for handling string/text of unknown size. the PCMPESTRy instructions support explicit length control via EDX:EAX register pair to specify the length text fragments in the source operands.

The first intermediate result, IntRes1, is an aggregated result of bit patterns from parallel comparison operations done on pairs of data elements from each text fragment, according to the imm[3:2] bit field encoding, see Table 10-1.

Table 10-1. SSE4.2 String/Text Instructions Compare Operation on N-elements

| Imm[3:2] | Name | IntRes1[i] is TRUE if | Potential Usage |
|---|---|---|---|
| 00B | Equal Any | Element i in fragment2 matches any element j in fragment1 | Tokenization, XML parser |
| 01B | Ranges | Element i in fragment2 is within any range pairs specified in fragment1 | Subsetting, Case handling, XML parser, Schema validation |
| 10B | Equal Each | Element i in fragment2 matches element i in fragment1 | Strcmp() |
| 11B | Equal Ordered | Element i and subsequent, consecutive valid elements in fragment2 match fully or partially with fragment1 starting from element 0 | Substring Searches, KMP, Strstr() |

Input data element format selection using imm[1:0] can support signed or unsigned byte/word elements.

The bit field imm[5:4] allows applying a unary transformation on IntRes1, see Table 10-2.

**Table 10-2. SSE4.2 String/Text Instructions Unary Transformation on IntRes1**

| Imm[5:4] | Name | IntRes2[i] = | Potential Usage |
|---|---|---|---|
| 00B | No Change | IntRes1[i] | |
| 01B | Invert | -IntRes1[i] | |
| 10B | No Change | IntRes1[i] | |
| 11B | Mask Negative | IntRes1[i] if element i of fragment2 is invalid, otherwise - IntRes1[i] | |

The output selection field, imm[6] is described in Table 10-3.

**Table 10-3. SSE4.2 String/Text Instructions Output Selection Imm[6]**

| Imm[6] | Instruction | Final Result | Potential Usage |
|---|---|---|---|
| 0B | PCMPxSTRI | ECX = offset of least significant bit set in IntRes2 if IntRes2 != 0, otherwise<br>ECX = number of data element per 16 bytes | |
| 0B | PCMPxSTRM | XMM0 = ZeroExtend(IntRes2); | |
| 1B | PCMPxSTRI | ECX = offset of most significant bit set in IntRes2 if IntRes2 != 0, otherwise<br>ECX = number of data element per 16 bytes | |
| 1B | PCMPxSTRM | Data element i of XMM0 = SignExtend(IntRes2[i]); | |

The comparison operation on each data element pair is defined in Table 10-4. Table 10-4 defines the type of comparison operation between valid data elements (last row of Table 10-4) and boundary conditions when the fragment in a source operand may contain invalid data elements (rows 1 through 3 of Table 10-4). Arithmetic comparison are performed only if both data elements are valid element in fragment1 and fragment2, as shown in row 4 of Table 10-4.

**Table 10-4. SSE4.2 String/Text Instructions Element-Pair Comparison Definition**

| fragment1 element | fragment2 element | Imm[3:2]= 00B, Equal Any | Imm[3:2]= 01B, Ranges | Imm[3:2]= 10B, Equal Each | Imm[3:2]= 11B, Equal Ordered |
|---|---|---|---|---|---|
| invalid | invalid | Force False | Force False | Force True | Force True |
| invalid | valid | Force False | Force False | Force False | Force True |
| valid | invalid | Force False | Force False | Force False | Force False |
| valid | valid | Compare | Compare | Compare | Compare |

The string and text processing instruction provides several aid to handle end-of-string situations, see Table 10-5. Additionally, the PCMPxSTRy instructions are designed to not require 16-byte alignment to simplify text processing requirements.

**Table 10-5. SSE4.2 String/Text Instructions Eflags Behavior**

| EFLAGs | Description | Potential Usage |
|---|---|---|
| CF | Reset if IntRes2 = 0; Otherwise set | When CF=0, ECX= #of data element to scan next |
| ZF | Reset if entire 16-byte fragment2 is valid | likely end-of-string |
| SF | Reset if entire 16-byte fragment1 is valid | |
| OF | IntRes2[0]; | |