



# Using MMX™ Instructions to Compute the AbsoluteDifference in Motion Estimation

Information for Developers and ISVs

From Intel® Developer Services  
[www.intel.com/IDS](http://www.intel.com/IDS)

March 1996

*Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.*

*Note: Please be aware that the software programming article below is posted as a public service and may no longer be supported by Intel.*

Copyright © Intel Corporation 2004

\* Other names and brands may be claimed as the property of others.

# **Using MMX™ Instructions to Compute the Absolute Difference in Motion Estimation**

---

March 1996

## **CONTENTS**

1.0 INTRODUCTION

2.0. MOTION ESTIMATION

2.1. Absolute Difference

2.2. Optimized MMX™ Code

2.3. Optimized Scalar

3.0. PERFORMANCE GAINS

4.0. ABSOLUTE DIFFERENCE

FUNCTION CODE LISTING

## 1.0. INTRODUCTION

The media extensions to the Intel Architecture (IA) instruction set include single-instruction, multiple-data (SIMD) instructions. This document describes an MMX™ technology implementation of a procedure to perform an absolute difference on a 16x16 block of pixels. This procedure can be an integral part of a motion estimation kernel, as will be described.

## 2.0. MOTION ESTIMATION

An important technique used in video compression is to try and predict movement between consecutive frames. In many cases a moving object stays the same from frame to frame and only moves across the viewing field. A substantially better compression ratio can be achieved by producing displacement vectors of the object from frame to frame, instead of compressing the object for every frame. Calculating these vectors is called motion estimation and requires the calculation of an absolute difference between blocks of the frames.

### 2.1. Absolute Difference

The motion estimation function sums the absolute differences (or squared differences) between the pixel values of two different 16x16 blocks, and finds the best match. In MPEG1 for example, the calculation can be made in four ways. Either the absolute differences between the pixel values is summed ( $L_1$  norm) or the square of the differences ( $L_2$  norm). Orthogonal to the difference equation, this sum can be accumulated with respect to a reference block that has been shifted either by some number of whole pixel positions or by some number of half-pixel positions.

A C Language code example for a 16x16 pixel full pel motion estimation using absolute differences is given in Example 1. The code has a *fast-out* so that if the difference accumulated across some rows is more than the current best match, it aborts the rest of the absolute difference.

*Example 1: C Code of 16x16 Motion Estimation*

```
char *bptr; /* pointer to start row of 16x16 pixel block being compressed.*/
char *cptr; /* pointer to start row of 16x16 pixel reference block.*/
val=0;
for(i=0;i<16;i++)
{
    for(j=0;j<16;j++)
    {
        data=*(bptr++)-*(cptr++);
        if (data<0) {val-=data;}
        else      {val+=data;}
    }
}
/* Fast out after this row if we've exceeded best match*/
if (val > best_value) break;
/* update pointer to next row*/
bptr += (rm->width - 16);
/* update pointer to next row */
cptr += (cm->width - 16);
}
```

### 2.2. Optimized MMX™ Code

The flow of the motion estimation inner loop code when using MMX instructions is shown in Figure 1. The operation uses a PSUBUSB (packed-byte-subtract-unsigned-with-saturation) to generate the absolute differences without requiring 16-bit precision to perform the operation.

Usually, subtraction of two 8-bit unsigned numbers produces a 9-bit signed result. Thus, in order to keep the precision it may seem that a conversion to 16 bits is needed before the subtract operation. This becomes unnecessary by using the PSUBUSB instruction. By subtracting source 2 from source 1 and then

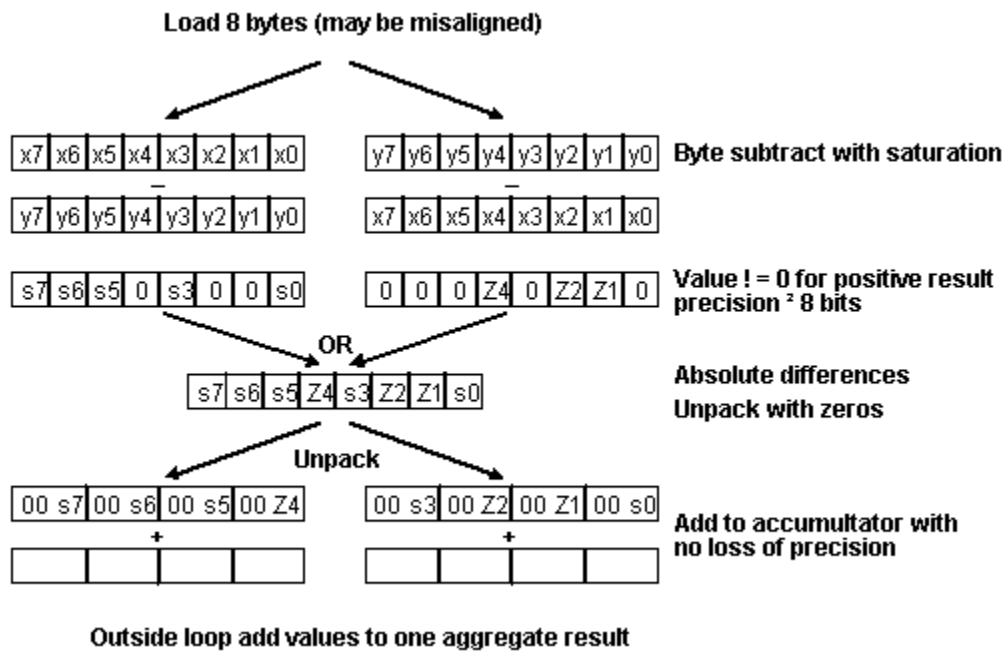
# Using MMX™ Instructions to Compute the AbsoluteDifference in Motion Estimation

March 1996

doing the opposite operation on a copy of one of the sources, each result register has an absolute difference value. This is only true when an element in source 1 is larger than an element in source 2; if not, the result is zero because a negative result saturates to zero. The same thing happens for the opposite subtraction. This takes care of generating the absolute differences for the cases in which the elements in source 2 are larger than the elements in source 1.

Using the OR command on the results of the two subtractions generates the eight desired absolute difference results. This enables us to find the absolute difference in byte precision substantially faster than if done in 16 bit precision.

Figure 1. Motion Estimation Inner Loop Using Packed Data



The MMX code for the inner-most computation of absolute differences is presented in Example 2. This code does not have the *fast out* option as does the C Language code of Example 1. In general, a *fast out* is not relevant after each 16x1 estimation as the overhead cost of adding the *fast out* for every iteration of the MMX instruction loop is prohibitive.

Example 2. Inner Core of Absolute Difference for 16x16 Block

```

estim Proc C Public uses m1:DWORD, m2:DWORD
                pxor                mm6, mm6
                movq                mm0, [m1]
                movq                mm1, [m2]
                movq                mm2, mm0
                movq                mm3, 8[m1]
                psubusb             mm0, mm1
                psubusb             mm1, mm2
                movq                mm4, 8[m2]
                por                 mm0, mm1
                movq                mm5, mm3
                movq                mm1, mm0
                psubusb             mm3, mm4
                punpckbw            mm0, mm6
                psubusb             mm4, mm5
    
```

## Using MMX™ Instructions to Compute the Absolute Difference in Motion Estimation

March 1996

```
    psrlq          mm1, 32
    por           mm3, mm4
    punpckbw mm1, mm6
    movq         mm4, mm3
    punpckbw mm3, mm6
    paddw        mm7, mm0
    psrlq          mm4, 32
    paddw        mm7, mm1
    punpckbw mm4, mm6
    paddw        mm7, mm3
    paddw        mm7, mm4
    ret
estim EndP
```

Note that the misalignment penalty of memory accesses is an important factor in the motion estimation loop. Thus, minimizing the number of memory accesses is very important. For this reason, it is faster to load the data once and copy it to avoid destruction inherent to the IA two-operand model than to base the code on MMX instructions with one of the operands from memory.

### 2.3. Optimized Scalar

It is possible to perform a version of the absolute difference using the same technique as the MMX instructions, but this can only be done on one data element at a time instead of utilizing the parallelism that MMX instructions can provide. This results in more optimized scalar code and is listed below in Example 3.

#### *Example 3. Optimized Scalar Code*

```
estim Proc C Public uses m1: DWORD, m1:DWORD
    pushl        esi
    pushl        edi
    pushl        ebx
    pushl        m1
    pushl        m2
    popl         edi
    popl         esi
    xorl         ecx, ecx
    xorl         edx, edx
    xorl         ebx, ebx
    xorl         eax, eax
    movb         bl, [esi]
    movb         cl, [edi]
    subl         ecx, ebx
    movb         bl, 1[esi]
    xorb         cl, ch
    movb         dl, 1[edi]
    subb         cl, ch
    subl         edx, ebx
    andl         ecx, 0xff
    xorb         dl, dh
    addl         eax, ecx
    subb         dl, dh
    andl         edx, 0xff
    movb         bl, 2[esi]
    addl         eax, edx
    movb         cl, 2[edi]
    subl         ecx, ebx
    movb         bl, 3[esi]
    xorb         cl, ch
    movb         dl, 3[edi]
    subb         cl, ch
    subl         edx, ebx
```

## Using MMX™ Instructions to Compute the Absolute Difference in Motion Estimation

---

March 1996

```
    andl    ecx,    0xff
    xorb    dl,     dh
    addl    eax,    ecx
    subb    dl,     dh
    andl    edx,    0xff
    movb    bl,     4[esi]
    addl    eax,    edx
    movb    cl,     4[edi]
    subl    ecx,    ebx
    movb    bl,     5[esi]
    xorb    cl,     ch
    movb    dl,     5[edi]
    subb    cl,     ch
    subl    edx,    ebx
    andl    ecx,    0xff
    xorb    dl,     dh
    addl    eax,    ecx
    subb    dl,     dh
    andl    edx,    0xff
    movb    bl,     6[esi]
    addl    eax,    edx
    movb    cl,     6[edi]
    subl    ecx,    ebx
    movb    bl,     7[esi]
    xorb    cl,     ch
    movb    dl,     7[edi]
    subb    cl,     ch
    subl    edx,    ebx
    andl    ecx,    0xff
    xorb    dl,     dh
    addl    eax,    ecx
    subb    dl,     dh
    andl    edx,    0xff
    movb    bl,     8[esi]
    addl    eax,    edx
    movb    cl,     8[edi]
    subl    ecx,    ebx
    movb    bl,     9[esi]
    xorb    cl,     ch
    movb    dl,     9[edi]
    subb    cl,     ch
    subl    edx,    ebx
    andl    ecx,    0xff
    xorb    dl,     dh
    addl    eax,    ecx
    subb    dl,     dh
    andl    edx,    0xff
    movb    bl,     10[esi]
    addl    eax,    edx
    movb    cl,     10[edi]
    subl    ecx,    ebx
    movb    bl,     11[esi]
    xorb    cl,     ch
    movb    dl,     11[edi]
    subb    cl,     ch
    subl    edx,    ebx
    andl    ecx,    0xff
    xorb    dl,     dh
    addl    eax,    ecx
    subb    dl,     dh
    andl    edx,    0xff
    movb    bl,     12[esi]
    addl    eax,    edx
    movb    cl,     12[edi]
```

## Using MMX™ Instructions to Compute the AbsoluteDifference in Motion Estimation

---

March 1996

```
    subl    ecx,    ebx
    movb   bl,    13[esi]
    xorb   cl,    ch
    movb   dl,    13[edi]
    subb   cl,    ch
    subl   edx,    ebx
    andl   ecx,    0xff
    xorb   dl,    dh
    addl   eax,    ecx
    subb   dl,    dh
    andl   edx,    0xff
    movb   bl,    14[esi]
    addl   eax,    edx
    movb   cl,    14[edi]
    subl   ecx,    ebx
    movb   bl,    15[esi]
    xorb   cl,    ch
    movb   dl,    15[edi]
    subb   cl,    ch
    subl   edx,    ebx
    andl   ecx,    0xff
    xorb   dl,    dh
    addl   eax,    ecx
    subb   dl,    dh
    andl   edx,    0xff
    popl   ebx
    addl   eax,    edx
    popl   edi
    popl   esi
    ret
estim EndP
```

### 3.0. PERFORMANCE GAINS

We compared the MMX technology technique to the C Language implementation as well as to the optimized scalar version. On a Pentium® processor implementation, the performance improvements are as follows:

- MMX technology performance gains over C implementation **five times**
- MMX technology performance gains over optimized scalar **two times**

## 4.0. ABSOLUTE DIFFERENCE FUNCTION CODE LISTING

```
.586
include MMX .inc
ASSUME ds:FLAT, cs:FLAT, ss:FLAT
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
_TEXT ENDS
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
; basic flow outlined in appnote.
; performing saturated subtractions and merging results of differences (por).
; unpacking is then performed to 16-bit precision to accumulate differences without
; overflow. Accumulation into mm7 is then done and returned in mm7.
estim Proc Near C Public m1:PTR DWORD, m2:PTR DWORD
    pxor        mm6, mm6
    movq        mm0, DWORD PTR [m1] ; grab 1st half of row from bptr
    movq        mm1, DWORD PTR [m2] ; grab 1st half of row from cptr
    movq        mm2, mm0           ; make a copy for abs diff operation
    movq        mm3, DWORD PTR 8[m1] ; grab 2nd half of row from bptr
    psubusb    mm0, mm1           ; do subtraction one way (w/saturation)
    psubusb    mm1, mm2           ; do subtraction the other way (w/saturation)
    movq        mm4, DWORD PTR 8[m2] ; grab 2nd half of row from cptr
    por        mm0, mm1           ; merge results of 1st half
    movq        mm5, mm3           ; make a copy for abs diff operation
    movq        mm1, mm0           ; keep a copy
    psubusb    mm3, mm4           ; do subtraction one way (w/saturation)
    punpcklbw  mm0, mm6           ; unpack to higher precision for accumulation
    psubusb    mm4, mm5           ; do subtraction the other way (w/saturation)
    psrlq      mm1, 32            ; shift results for accumulation
    por        mm3, mm4           ; merge results of 2nd half
    punpcklbw  mm1, mm6           ; unpack to higher precision for accumulation
    movq        mm4, mm3           ; keep a copy
    punpcklbw  mm3, mm6           ; unpack to higher precision for accumulation
    paddw      mm7, mm0           ; accumulate difference...
    psrlq      mm4, 32            ; shift results for accumulation
    paddw      mm7, mm1           ; accumulate difference...
    punpcklbw  mm4, mm6           ; unpack to higher precision for accumulation
    paddw      mm7, mm3           ; accumulate difference...
    paddw      mm7, mm4           ; accumulate difference...
    ret
estim EndP
_TEXT ENDS
END
```