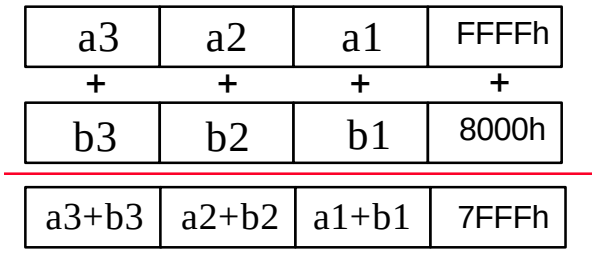
Instruction Examples

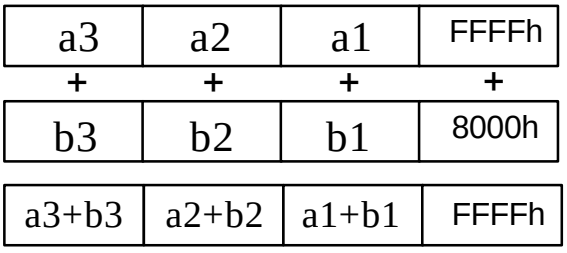
The following section will describe briefly five examples of MMX instructions. For illustration, the data type shown in this section will be the 16-bit word data type; most of these operations also exist for 8-bit or 32-bit packed data types also.

The following example shows a packed add word with wrap around. It performs four additions of the eight, 16-bit elements, each addition independent of the others, in parallel. In this case, the right-most result exceeds the maximum value representable in 16-bits—thus it wraps-around. This is the way regular IA arithmetic behaves. FFFFh + 8000h would be a 17 bit result. The 17th bit is lost because of wrap around, so the result is 7FFFh.



Wrap-around Add

The following example is for a packed add word with unsigned saturation. This example uses the same data values from before. The right-most add generates a result that does not fit into 16 bits; consequently, in this case saturation occurs. Saturation means that if addition results in overflow or subtraction results in underflow, the result is clamped to the largest or the smallest value representable. For an unsigned, 16-bit word, the largest and the smallest representable values are FFFFh and 0x0000; for a signed word the largest and the smallest representable values are 7FFFh and 0x8000. This is important for pixel calculations where this would prevent a wrap-around add from causing a black pixel to suddenly turn white while, for example, doing a 3\\D graphics Gouraud shading loop.

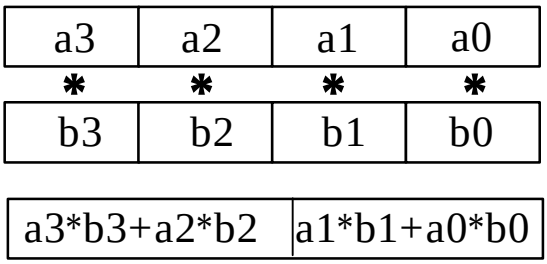


Saturating Arithmetic

The specific instruction here is Packed Add Unsigned Saturation Word (PADDUSW). A complete set of ADD operations exists for signed and unsigned cases. The number FFFFh, treated as unsigned (65,535 decimal), is added to 0x8000 unsigned (32,768), and the result saturates to FFFFh - the largest representable unsigned 16-bit value.

There is no “saturation mode bit” as a new mode bit would require a change to the Operating System (OS). Separate instructions are used to generate wrap-around and saturating results.

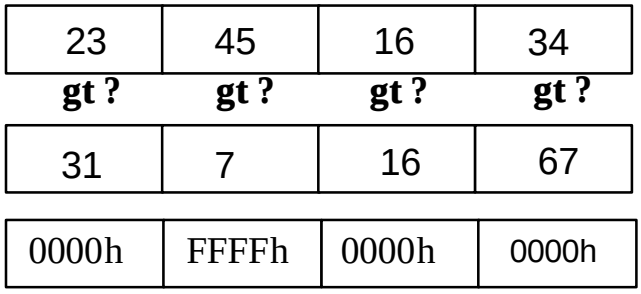
The next example shows the key instruction used for multiply-accumulate operations, which are fundamental to many signal processing algorithms like vector-dot-products, matrix multiplies, FIR and IIR Filters, FFTs, DCTs etc. This instruction is the packed multiply add (PMADD).



16b x 16b -> 32b Multiply Add

The PMADD instruction starts from a 16-bit, packed data type and generates a 32-bit packed, data type result. It multiplies all the corresponding elements generating four 32-bit results, and adds the two products on the left together for one result and the two products on the right together for the other result. To complete a multiply-accumulate operation, the results would then be added to another register which is used as the accumulator.

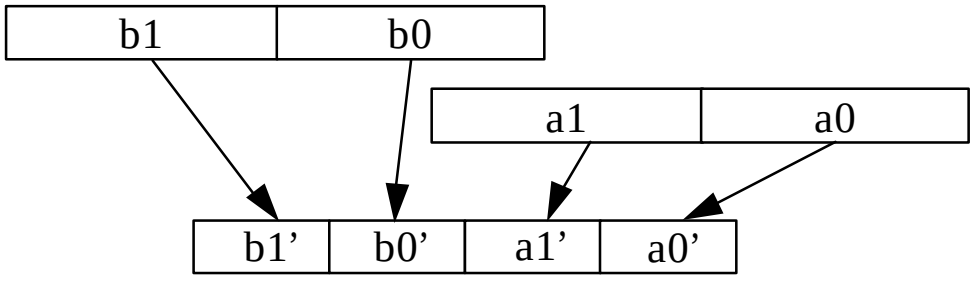
The following example is a packed parallel compare. This example compares four pairs of 16-bit words. It creates a result of true (FFFFh), or false (0000h). This result is a packed mask of ones for each true condition, or zeros for each false condition. The following example shows an example of a compare "greater than" on packed word data. There are no new condition code flags, nor any existing IA condition code flags affected by this instruction.



Parallel Compares

The packed compare result can be used as a mask to select elements from different inputs using a logical operation, eliminating the need for a branch or set of branch instructions. The ability to do a conditional move instead of using branch instructions is an important performance issue in advanced processors that have deep pipelines and employ branch prediction. A branch based on the result of a compare operation on the incoming data is usually difficult to predict, as incoming data in many cases can change randomly. Eliminating branches that are used to perform data selection by using the conditional select capability, together with the parallelism of the MMX instruction set, is an important performance enhancement feature.

Following is an example of a pack instruction. It takes four 32-bit values and packs them into four 16-bit values, performing saturation if one of the 32-bit source values does not fit into a 16-bit result. There are also instructions that perform the opposite - unpack, for example, a packed byte data type into a packed word data type.



The pack and unpack instructions exist to facilitate conversion between the new packed data types. These are especially important when an algorithm needs higher precision in its intermediate calculations, as in image filtering. A filter on an image usually involves a set of multiply operations between filter coefficients and a set of adjacent image pixels, accumulating all the values together. These multiplies and accumulations need more precision than 8-bits, the original data type of the pixels. The solution is to unpack the image's 8-bit pixels into 16-bit words, perform the calculations in 16-bit words without concern for overflow, then pack back to 8-bit pixels before storing the filtered pixels to memory.

APPLICATION EXAMPLES

The following section describes example uses of the MMX instruction set to implement basic coding structures:

Conditional Select

Multimedia applications must process large sets of data. In some cases there is a need to select different data based on a condition query performed on the incoming data. Intel has been able to improve performance in its family of processors by implementing micro-architectural features for faster performance and deeper pipelines. Branch prediction is an important part of making the pipe run efficiently, as a mis-prediction can cause the pipe to flush and degrade performance. The following example shows an efficient way to reduce the need to use branch instructions, especially those that are data dependent, and thus very difficult to predict. The Chroma Keying example demonstrates how conditional selection using the MMX instruction set removes branch mis-predictions, in addition to performing multiple selection operations in parallel. Text overlay on a graphics/video background, or Sprite overlays in games are some of the other operations that would benefit from this technique.

Chroma Keying

Most have seen the television weather man overlaid on the image of a weather map. In this example we use a green screen to overlay an image of a woman on a picture of a Spring blossom. We'll illustrate this example processing four 16-bit pixels in parallel. The instructions also allow the processing of eight 8-bit pixels in parallel, a substantial performance speed-up potential.

First we'll take four pixels from the picture with the woman on a green background. The top row of data below represents pixels that alternate green, not green, green, and not green. The compare instruction builds a mask for that data. That mask is a sequence of words that are all ones or all zeros representing the Boolean values of true and false. We now know what is the unwanted background and what we want to keep. This is shown below using the shadow picture.