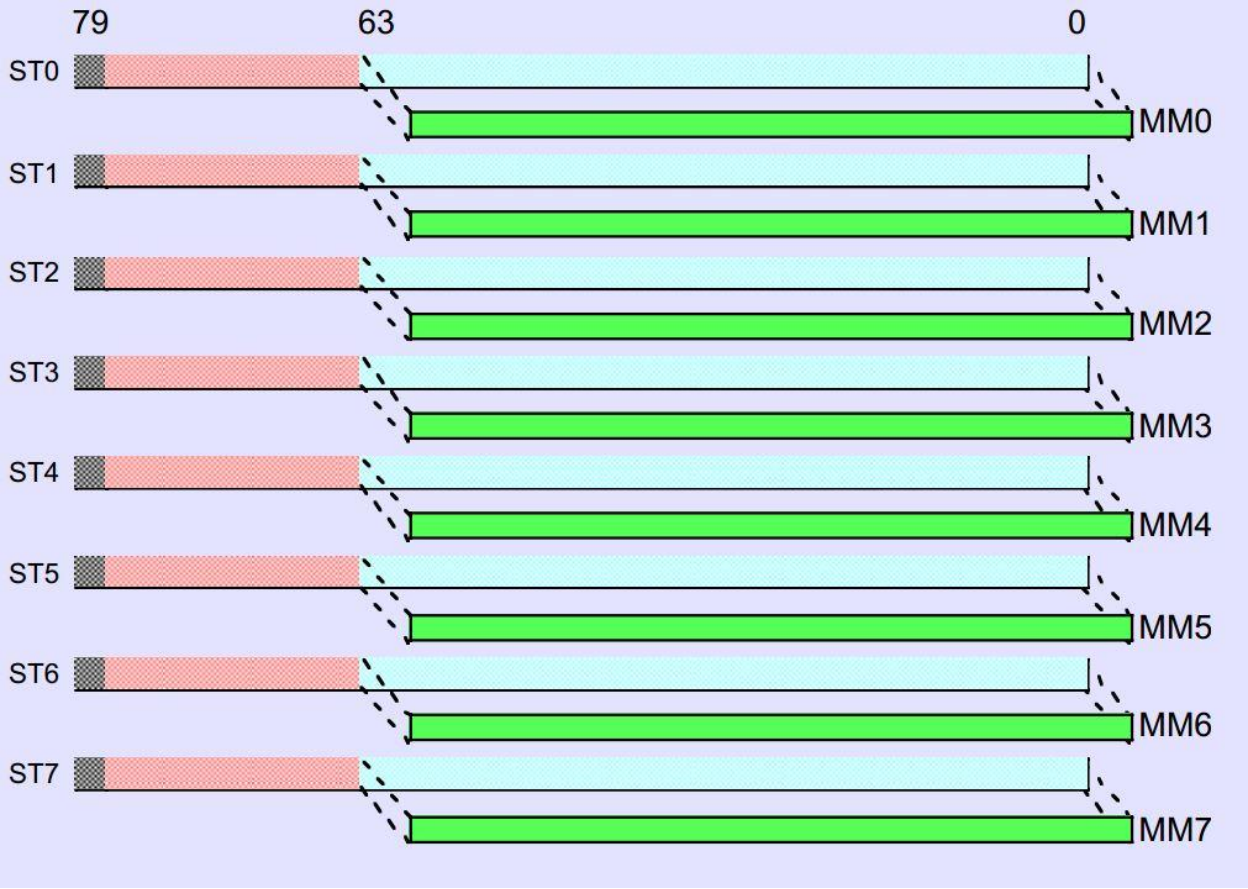
MMX

SIMD

SIMD (Single Instruction, Multiple Data). SIMD describes any extension to microprocessors that allow it to operate on data in parallel. Some common SIMD extensions are MMX, 3DNow!, SSE, and AltiVec (related to VMX). There are many others, but these are the most common ones found in ordinary PCs.

MMX (An Overview)

MMX was the first set of SIMD extensions applied to Intel's 80x86 instruction set. It was introduced in 1997. MMX introduces a number of new instructions that operate on single 64-bit quantities, 2 32-bit quantities, 4 16-bit quantities, or 8 8-bit quantities all at once. It uses the same register space as the FPU, so one cannot use MMX and floating point operations at the same time. It provides the programmer with 8 general-purpose registers, all 64 bits wide (MM0 - MM7). With the exception of emms, movd, and movq, all MMX instructions start with the letter 'p'. Sometimes, MMX is referred to as Matrix Math eXtensions.



The column on the right shows the speed-up obtained moving from scalar C code to MMX code.

Title	Speed-up
Audio	
Audio Echo Effects	5.9x
G.728 Code Book Search	2.7x
Graphics (3D)	
Advanced Procedural Texturing	10x
3D Bilinear Texture Mapping	7x
3D Transform	3.1x
Image Processing	
2X 8-bit Image Scaling	13.5x
Bilinear Interpolation	3.9x
Median Filter	3.8x
Alpha Blending	8x
Video	
IDCT 2D 8x8	3.5x
Absolute Difference	5x
Haar Transform - 2x2	2.2x
Get Bits	2.4x
Video Loop Filter	1.9x

MMX – The Registers

As mentioned above, MMX provides the programmer with 8 64-bit general purpose registers. These registers, called MM0 - MM7, can be used in a number of ways. They can be used as single 64-bit quantities, dual 32-bit quantities, 4 16-bit quantities, or 8 8-bit quantities. When any action is taken on an MMX register, it is applied to all the elements of the register at the same time. This allows software to operate up to 8 times faster (though in real life this never happens).

MMX Registers in FPU's Register Space		
Register	**79 - 64**	**63 - 0**
ST0	**xx**	**MM0**
ST1	**xx**	**MM1**
ST2	**xx**	**MM2**
ST3	**xx**	**MM3**
ST4	**xx**	**MM4**
ST5	**xx**	**MM5**
ST6	**xx**	**MM6**
ST7	**xx**	**MM7**

Notice how the top 16 bits of each 80-bit FPU register are unused in MMX mode.

The many flavors of MMX Registers	
Register	**Description**
63 0	A Single 64-bit Quadword
63 31	2 32-bit Doublewords
63 47 31 15	4 16-bit Words
63 55 47 39 31 23 15 7	8 8-bit Bytes

MMX-State Management

Since MMX and FPU registers occupy the same space it becomes a problem when you try to use floating point code and MMX code at the same time. When the CPU is in MMX mode, it sets the unused fpu bits to invalid values, which will cause any floating point instructions to behave strangely. Entering MMX mode is fairly simple; just execute an MMX instruction. Exiting MMX isn't as simple. We use the emms instruction to perform this.

emms

emms takes no arguments, and can be executed at any time. It restores the fpu so it can operate normally. All MMX code should call emms when it is finished if floating point code is going to be running afterwards. emms stands for Empty MMX State.

MMX-Data Movement

MMX gives us a few new mov instructions to facilitate getting data into and out of MMX registers. These new instructions are movd and movq.

movd (MOVe Doubleword) can move either a 32-bit register or memory location into or out of the bottom 32 bits of an MMX register. When data moves in, the top 32 bits of the MMX register are set to zero.

movq (MOVe Quadword) moves 64-bit quantites between memory and an MMX register or between two MMX register.

MMX-Boolean Logic

MMX is integer-only, so it makes sense for it to offer normal boolean logic operations. These are fairly easy to grasp.

pxor can exclusive-or (XOR) any two MMX registers, an MMX register and memory, or an MMX register and a constant.

por can bitwise-or (OR) any two MMX registers, an MMX register and memory, or an MMX register and a constant.

pand can bitwise-and (AND) any two MMX registers, an MMX register and memory, or an MMX register and a constant.

pandn can bitwise-not-and (NAND) any two MMX registers, an MMX register and memory, or an MMX register and a constant.

These instructions operate the same regardless of how the data is arranged in the register (whether it's a 64-bit value or 8 8-bit values). That's the nature of boolean logic, after all.

There are also a number of shift operations available in MMX.

psllw shifts a specified register left a certain number of bits, operating on words (16 bits).

pslld shifts a specified register left a certain number of bits, operating on doublewords (32 bits).

psllq shifts a specified register left a certain number of bits, operating on a quadwords (64 bits).

psrlw shifts a specified register right a certain number of bits, operating on words (16 bits). This is a logical shift, not arithmetic.

psrld shifts a specified register right a certain number of bits, operating on doublewords (32 bits). This is a logical shift, not arithmetic.

psrlq shifts a specified register right a certain number of bits, operating on a quadwords (64 bits). This is a logical shift, not arithmetic.

psraw shifts a specified register left a certain number of bits, operating on words (16 bits). This one is arithmetic, which means the new top bits are a copy of the original top bit (the sign bit).

psrad shifts a specified register left a certain number of bits, operating on doublewords (32 bits). This one is also arithmetic.

The shift operations *do* distinguish between the various sizes of the register. This is necessary to keep bits in one value from affecting adjacent values.

MMX-Math

MMX has a number of basic math operations included.

paddb adds an MMX register and another MMX register or memory as unsigned 8-bit bytes.

paddsb is just like paddb, except the bytes are signed and the values saturate instead of wrapping around. This instruction saturates at 127 (0x7f) or -128 (0x80).

paddusb is like paddsb but with unsigned bytes. This instruction saturates at 255 (0xff).

paddw add an MMX register and another register or memory as unsigned 16-bit words.

paddsw is just like paddsb except it uses 16-bit words instead of 8-bit bytes. This instruction saturates at 32767 (0x7fff) or -32768 (0x8000).

paddusw adds unsigned words, and saturates at 65535 (0xffff).

paddd adds a register and another register or memory location as unsigned 32-bit doublewords.

psubb subtracts a memory location or MMX register from another register, operating on unsigned 8-bit bytes.

psubsb subtracts a register or memory location from another register, using signed bytes, and saturates at -128 (0x80) or 127 (0x7f).

psubusb subtracts unsigned bytes with saturation, similar to psubsb. This instruction saturates at 0 (0x00).

psubw subtracts unsigned 16-bit words.

psubsw subtracts signed 16-bit words, and saturates at 32767 0x7fff) or -32768 (0x8000).

psubusb subtracts unsigned 16-bit words, saturating at 0 (0x0000).

psubd subtracts a register or memory location from another register using unsigned 32-bit doublewords.

MMX also supplies a few multiply instructions.

pmulhw multiplies a register with another register or memory locatioin using signed 16-bit words. It then stores the upper 16 bits of each 32-bit result.

pmullw multiplies just like pmulhw except it stores the lower 16 bits of each 16-bit result.

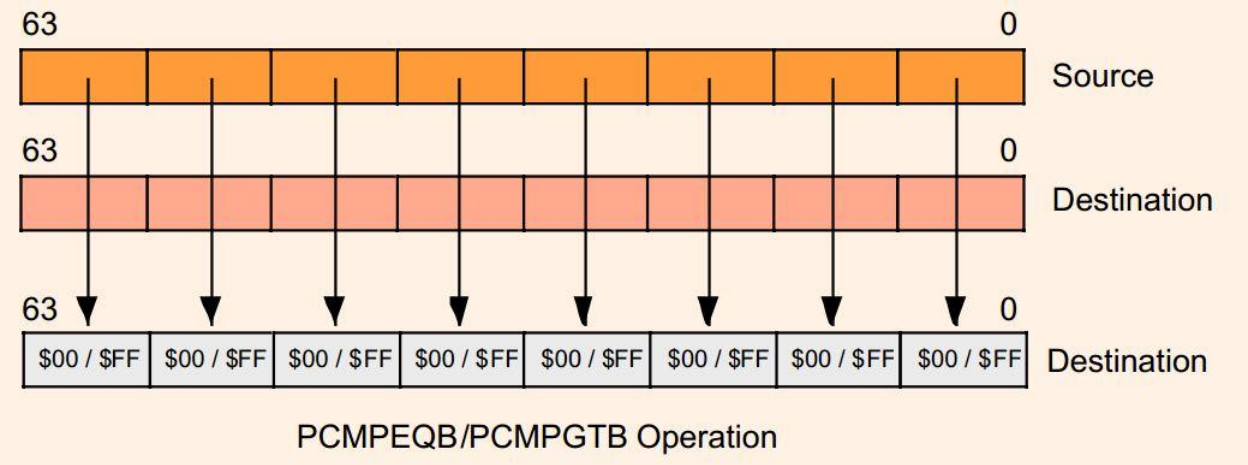
pmaddwd multiplies signed 16-bit words and adds the 32-bit results. It multiplies an MMX register and another register or memory location.

MMX-Comparasons

MMX provides a bunch of instructions for performing various-sized compares.

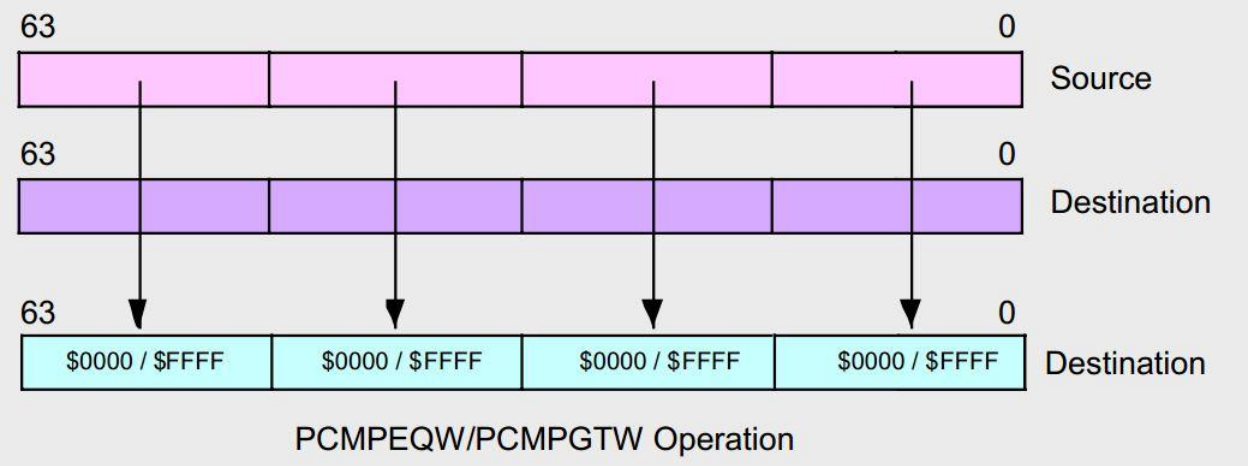
pcmpeqb compares for 8-bit equality, between an MMX register and another register or memory location. For pairs that are equal the result is all ones (0xff), otherwise it is zero (0x00).

pcmpgtb performs a 'Greater Than' 8-bit value compare in the same manner as pcmpeqb. For larger values, the result is all ones (0xff), otherwise zero (0x00).



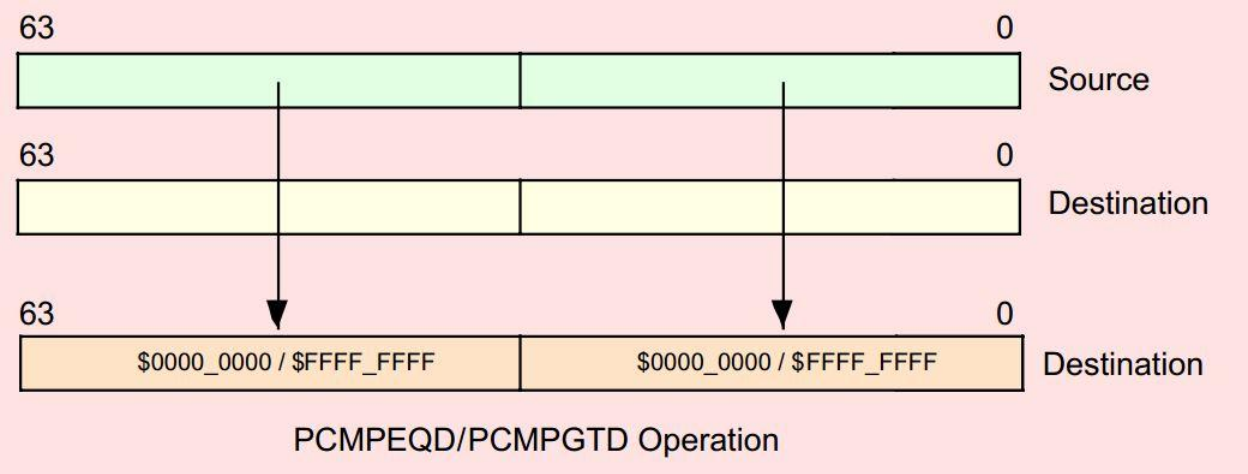
pcmpeqw compares 16-bit words for equality, just like pcmpeqb.

pcmpgtw is the 16-bit equivalent of pcmpgtb.



pcmpeqd compares 32-bit doublewords for equality.

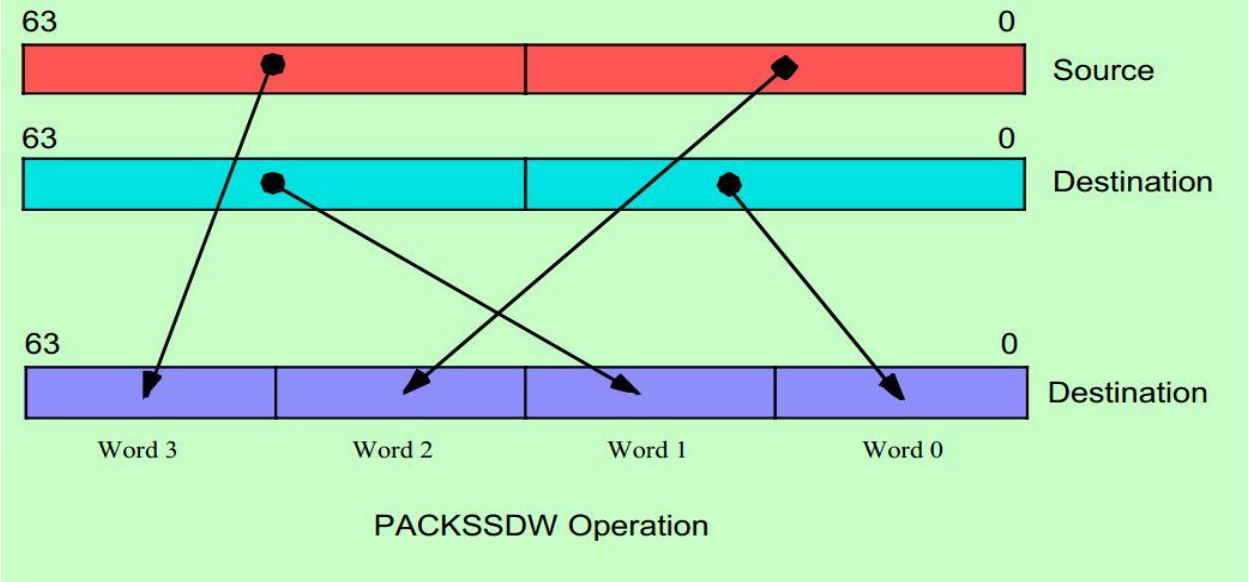
pcmpgtd compares magnitudes of 32-bit doublewords, just like pcmpgtw does for words.



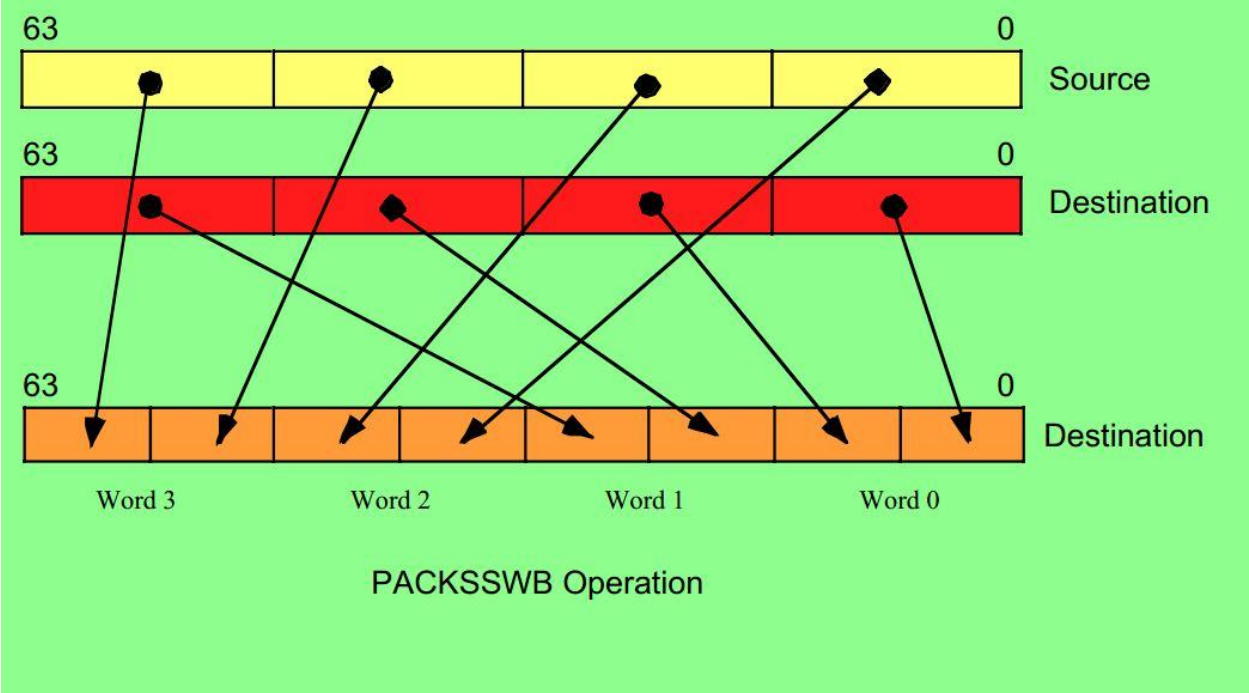
MMX-Data Packing

There are several instructions used for data packing provded by MMX. These instructions generally sign- or zero-extend values, interleave values, and truncate values.

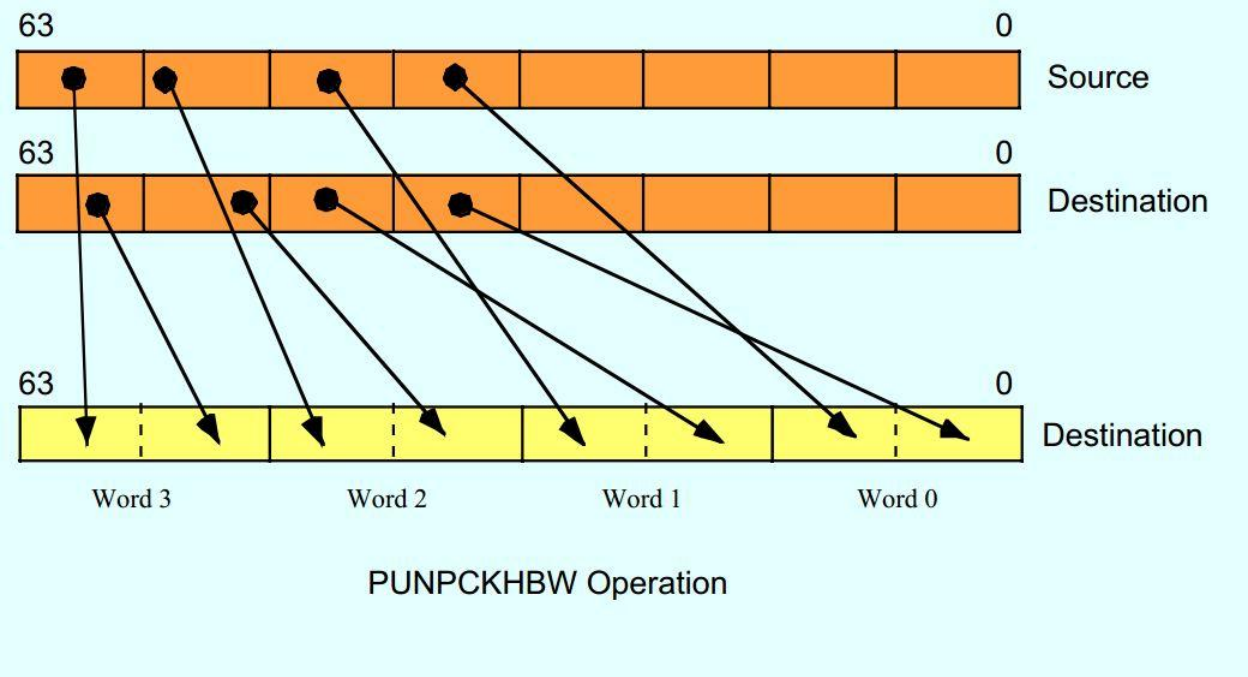
packssdw takes a register and another register or memory location, and saturates the 32-bit doublewords into 16-bit words. The doublewords and word results are signed.



packsswb takes a register and another register or memory location, and saturates the 16-bit signed words from both into signed 8-bit bytes in the first register.

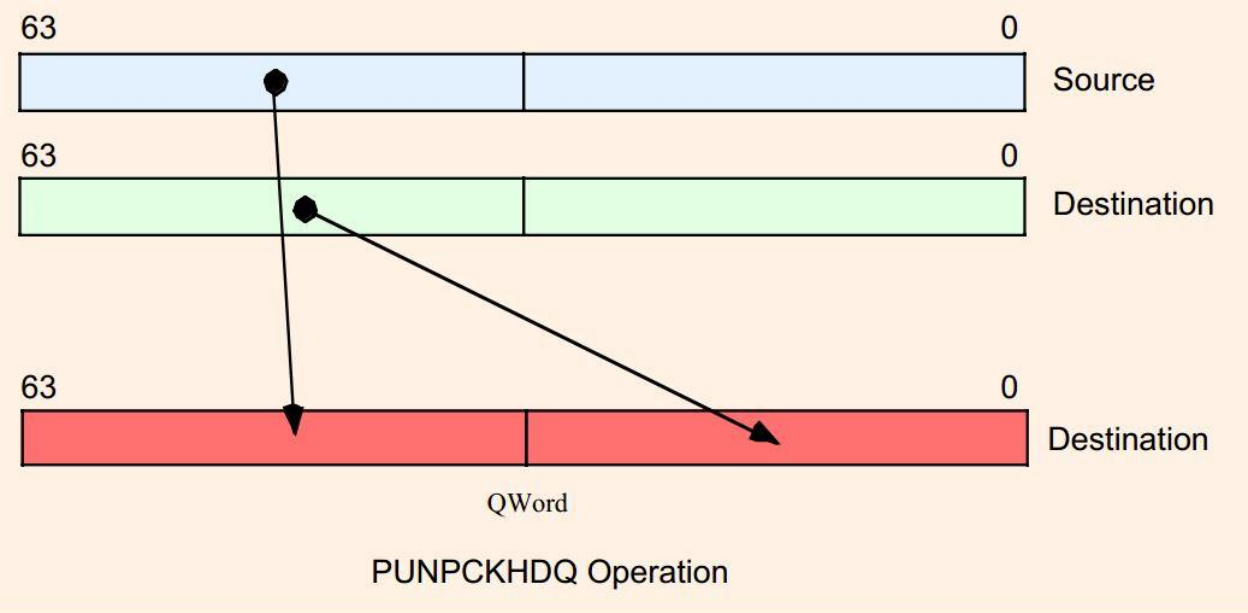


packuswb is similar to `packsswb`, with the source words and resultant bytes being unsigned instead of signed.
punpckhbw unpacks the top 32 bits of 2 MMX registers or a register and a memory location into a destination MMX register. The data is interleaved in 8-bit pieces, with the 2nd operand going to the top halves and the 1st operand going to the bottom halves.



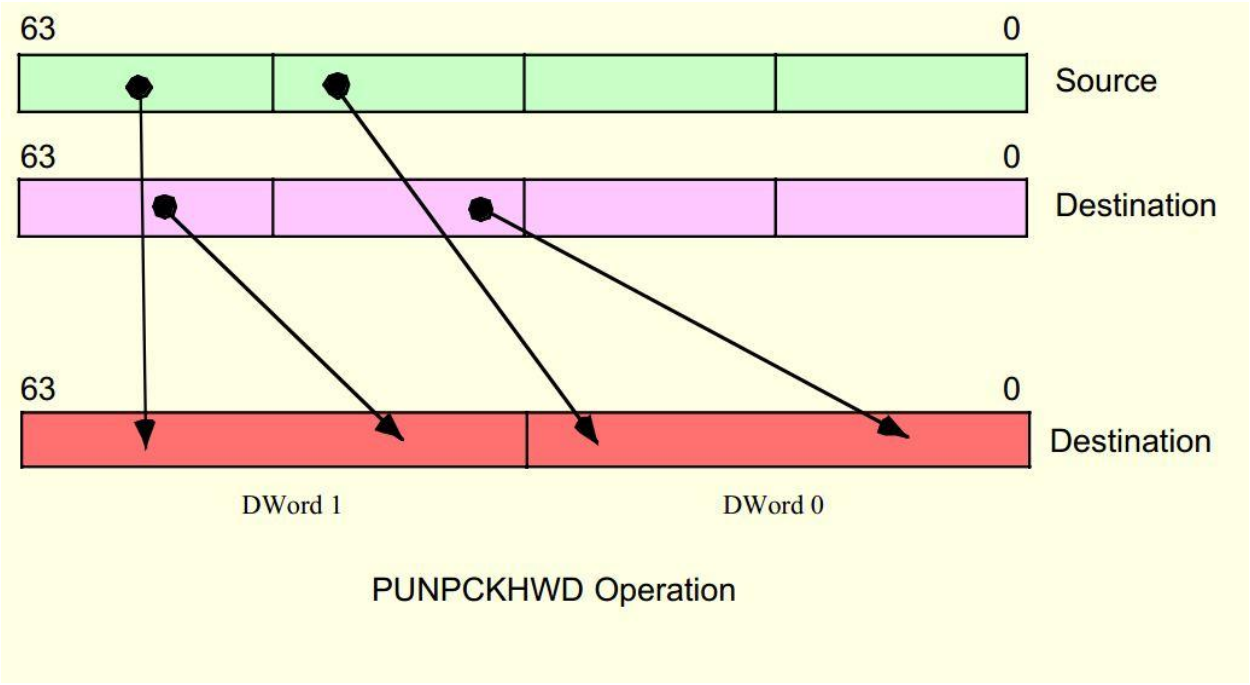
PUNPCKHBW Operation

punpckhdq is similar to `punpckhbw`, except it uses 32-bit pieces instead of 8-bit ones. The 1st operand's top half goes to the bottom half of the destination, and the 2nd operand's top goes to the top half.



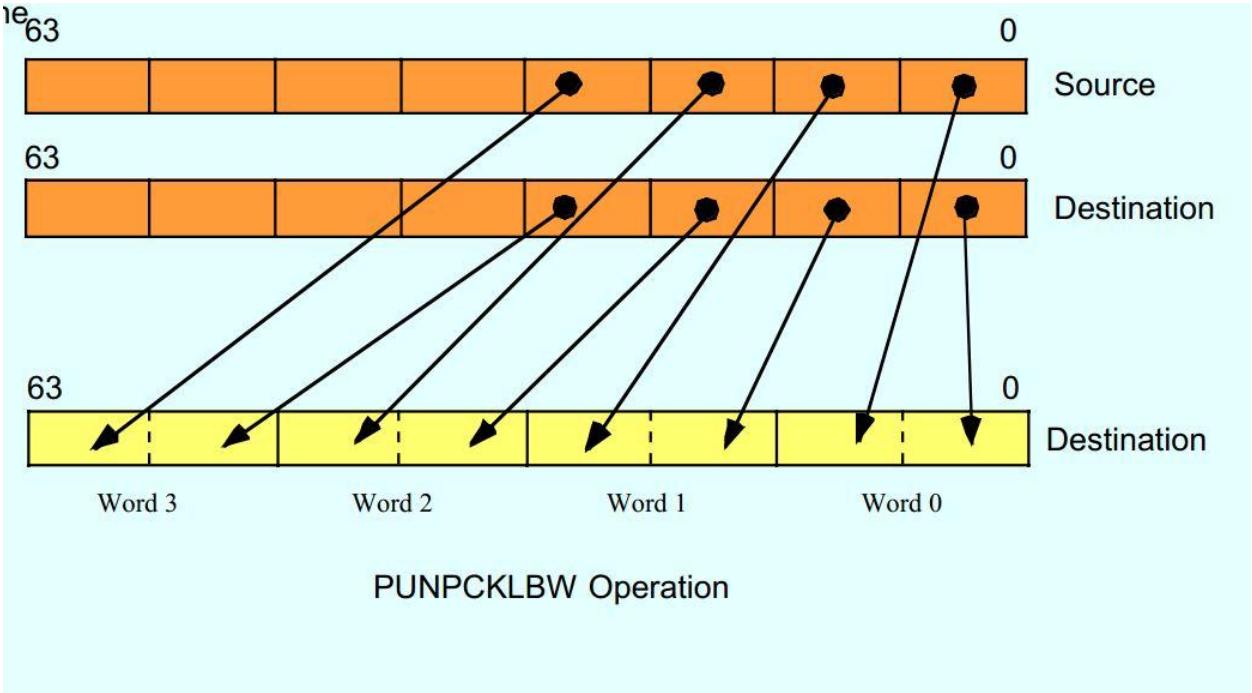
PUNPCKHDQ Operation

punpckhwd is also similar to `punpckhbw`, but with 16-bit pieces instead of 8-bit ones. The 1st operand goes to the top halves, and the 2nd operand goes to the bottom halves.



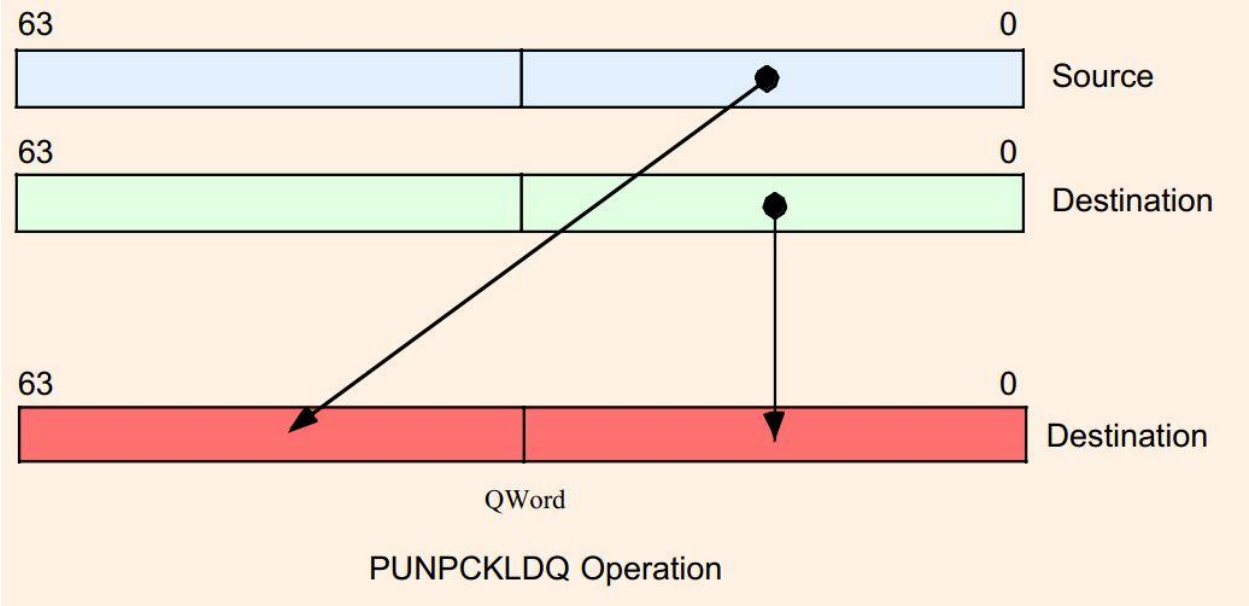
PUNPCKHWD Operation

punpcklbw is like `punpckhbw`, but instead of taking data from the top half of the sources, data is taken from the bottom.

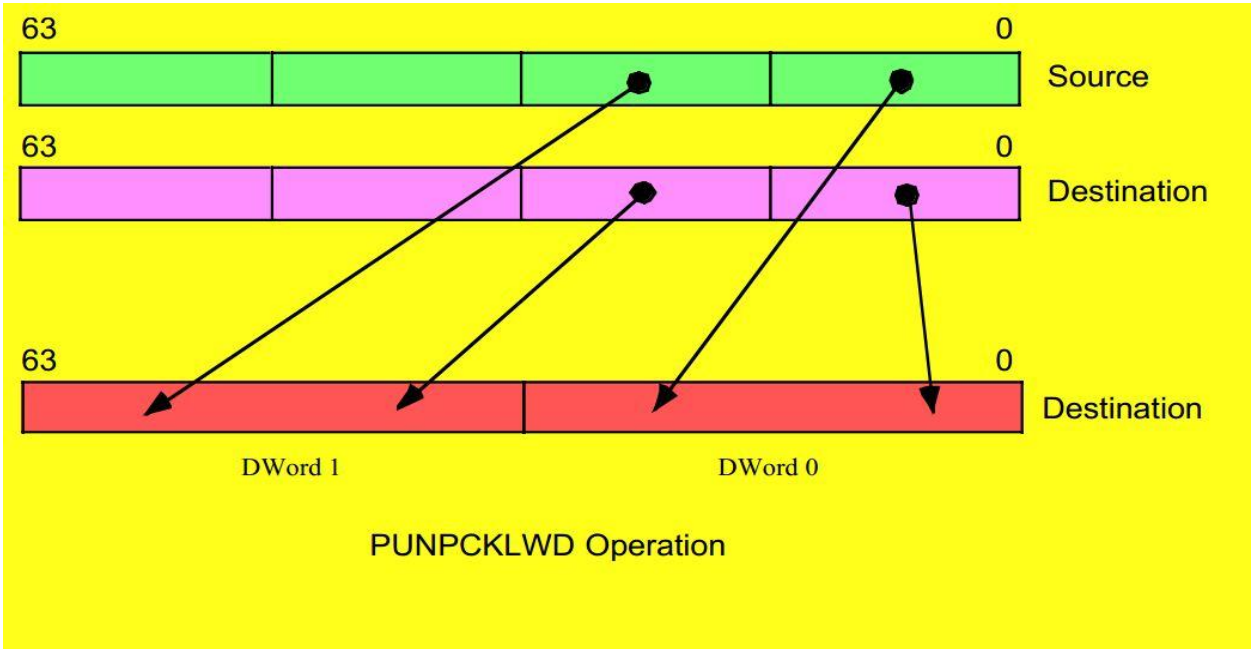


PUNPCKLBW Operation

punpckldq is like `punpckhdq`, but it uses the bottom 32-bit pieces instead of the top 32-bit pieces.



punpcklwd is like `punpckhwd`, but it uses the bottoms of the sources instead of the tops.



Resources

1.http://www.tommesani.com/MMXPrimer.html

2. ORACLE x86 Assembly Language Reference Manual

http://docs.oracle.com/cd/E19963-01/html/821-1608/eojdc.html

3. http://softpixel.com/~cwright/programming/simd/mmx.php

4. http://en.wikipedia.org/wiki/MMX_(instruction_set)

5.The MMX Instruction Set – Chapter Eleven - 2001, By Randall Hyde.