Using MMX™ Instructions to Implement a Modem Passband Canceler

Information for Developers and ISVs
From Intel® Developer Services
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.
CONTENTS

1.0. INTRODUCTION
2.0. WHAT IS ECHO CANCELING
3.0. THE PASSBAND ECHO CANCELER ALGORITHM
4.0. PASSBAND ECHO CANCELER IMPLEMENTATION METHODOLOGY
5.0. SIMPLE IMPLEMENTATION - psecimpl.asm
   5.1. Steps to Reduce Instruction Count
   5.2. Overview of psecimpl.asm
   5.3. Potential Improvements
6.0. ALIGNED IMPLEMENTATION - psecalin.asm
   6.1. Steps to Align the Code
   6.2. Data Alignment Analysis of psecimpl.asm
   6.3. Overview of psecalin.asm
7.0. OPTIMIZED IMPLEMENTATION - psecopt.asm
   7.1. Steps to Fully Optimize the Code
   7.2. Pairing Methodology
   7.3. Overview of psecopt.asm
   7.4. Potential Improvements
   7.5. Performance Numbers

APPENDIX A
APPENDIX B
APPENDIX C
1.0. INTRODUCTION

The Intel Architecture (IA) media extensions include single-instruction, multi-data (SIMD) instructions. This application note presents an implementation of a common modem algorithm that takes advantage of the new Intel Architecture media extensions. It describes the steps followed to optimize the assembly code, and gives suggestions for potential further improvements.

2.0. WHAT IS ECHO CANCELING

There are two sources of echo in a modem. The near end (NE) echo signal is a combination of the reflection of the transmitted signal due to the impedance mismatches of the lines at the hybrid transformer on the modem board, and the mismatches at the Public Switch Telephone Network (PSTN). The far end echo signal is a combination of the reflection of the transmitted signal due to impedance mismatches of the lines at the far end hybrid transformer on the receiving modem board, and the mismatches at the far end of the PSTN (Figure 1). The near end echo has a much larger amplitude than the far end echo. The algorithm is the same for both the near end and far end echo cancelers. The difference is in the amount of delay in the buffer that holds the transmitted data used in the filter calculation: the delay is longer for the far end.

*Figure 1. Echo Origin*

The passband echo canceler is an adaptive filter that effectively cancels out the near and far end echoes allowing the transmitted signal from the remote modem to arrive more cleanly at the receiver. The echo canceler can adapt because it knows the characteristics of the transmitted signal, which also appears in the echoes and can therefore be subtracted from the combined signal.
3.0. THE PASSBAND ECHO CANCELER ALGORITHM

The passband echo canceler algorithm has three functions (Figure 2):

1. Computation of the adaptive filter output
2. Computation of the difference between the filter's output and $s$ (error)
3. Adaptation of the real ($I$) and imaginary ($Q$) adaptive filter coefficients.

The real received data is of type 16-bit signed integer. Filter coefficients are 32 bit signed integers, split into a 16-bit low-order part and a 16-bit high-order part. Only the filter's higher 16 bits are used in the computation of the filter's output, but the full 32 bits are used in the filter adaptation.

Each baud received by the modem is represented by three real samples, stored in the $s$ array. As a result, the passband echo canceler is really a set of three complex, 32 bit, filters. These filters are represented by a set of four 16-bit, real and imaginary arrays, with the three filters sharing the same arrays. In each array, the order is: filter1, filter2, and filter3.

The actual pseudocode for the passband echo canceler function implemented in this application note is listed in Example 1.

**Example 1. C Pseudocode for Passband Echo Canceler**

```c
void psEchoCanceler (short *dI, short *dQ, short *s, short *hIH, short *hQH,
short *hIL, short *hQL, short h_Leng, short s_Leng)
{
    long y, adapt;
    short hnum, filtnum, snum, MU=3;
    for (snum=0; snum<s_Leng; snum++) // signal length
    {
        for (filtnum=0; filtnum<3; filtnum++) // 3 filters
        {
            //---------- cancel echo -----------
            for (y=0, hnum=h_Leng-1; hnum>=0; hnum--)
            {
                y+=dI[hnum+snum]*(long)hIH[hnum+filtnum*h_Leng] -
                dQ[hnum+snum]*(long)hQH[hnum+filtnum*h_Leng];
            }
            s[filtnum+3*snum]=s[filtnum+3*snum]-(short)(y>>14);
            //------ canceler adaption ---------
            for (hnum=0; hnum<h_Leng; hnum++)
            {
                adapt=(((long)hIH[hnum+filtnum*h_Leng])<<16) |
                       ((long)hIL[hnum+filtnum*h_Leng])&0x0000ffff);
                adapt=adapt+(s[filtnum+3*snum]*(long)dI[hnum+snum])>>MU;
                hIH[hnum+filtnum*h_Leng]=adapt>>16;
            }
        }
    }
}
```
hIL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
adapt=(((long)hQH[hnum+filtnum*h_Leng])<<16)
    |(((long)hQL[hnum+filtnum*h_Leng])&0x0000ffff);
adapt=adapt-((s[filtnum+3*snum]*(long)dQ[hnum+snum])>>MU);
hQH[hnum+filtnum*h_Leng]=adapt>>16;
hQL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
4.0. PASSBAND ECHO CANCELER IMPLEMENTATION METHODOLOGY

The final MMX™ code for the passband echo canceler was developed in stages. To reach that goal, the following steps were taken:

1. Reviewed the C implementation to minimize computational complexity.
2. Developed a simple functional version with a minimum number of MMX instructions (psecimpl.asm, listed in Appendix A).
3. Added code to align all memory accesses (psecalin.asm, shown in Appendix B)
4. Completely reordered the code to pair a maximum number of instructions (psecopt.asm, listed in Appendix C).

Each of the steps is described in a separate section below.
5.0. SIMPLE IMPLEMENTATION - psecimpl.asm

Refer to Appendix A when reading this section.

5.1. Steps to Reduce Instruction Count

The goal of the first MMX code was to give the same functionality as the C code, with a minimum number of instructions.

The following principles were used to reduce instruction count:

- Gave the inner-most loop the highest priority while minimizing instruction count (CancelEchoLoop, CancelerAdaptionLoop).
- Made the array index also have the role of loop counter and adjusted the size of the loop step and the counter limit accordingly (hnun variable as array index and loop count, loop step of 4*DATASIZE in CancelEchoLoop, loop step of 2*DATASIZE in CancelerAdaptionLoop).
- Made the loop counter decrement to 0, because in the Intel architecture, the sub instruction modifies the condition codes according to the result's value compared to 0.
- Computed array offsets as early as possible and combined them with all other non-varying offsets before the loop (combine hIH_base and filtnum*h_Leng*DATASIZE pointers into one variable before CancelEchoLoop).
- Left the varying array index (hnun) as a displacement to add in real time to the rest of the address of the array element (hIH_base+filtnum*h_Leng*DATASIZE), because the Pentium® processor version with MMX technology hides one displacement computation for free in the pipeline.
- Used variable names instead of registers (through text macro definition) to disconnect registers from variables, to optimize register usage more easily, and to make the code more readable. Proper register management is extremely important in the Intel Architecture because only six general purpose registers are available. Pros and cons of symbolic register names are discussed below.
- Chose to compute hI and hQ adaptation in one loop and share a register between dI and dQ. I could have split the two computations into two separate loops, but I preferred adding extra loads rather than adding extra branches (for which misprediction costs several cycles).
- Computed (-y) instead of y in CancelEchoLoop to minimize the number of instructions needed to compute s.

Pros and cons of the symbolic register naming convention used in the assembly code of this application note are discussed in Table 1 below. In the end, using symbolic register names is largely a matter of taste.

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meaning of computations easier to see.</td>
<td>Actual register used is hidden. Harmful if the code relies on special properties of the register (which is not the case in this application note).</td>
</tr>
<tr>
<td>Register renaming is easy, which helps simplify pairing and reduce register pressure.</td>
<td>Many programmers are unfamiliar with this coding style; a learning curve is involved.</td>
</tr>
</tbody>
</table>
5.2. Overview of psecimpl.asm

The following lines describe the implementation decisions made. Code portions are identified by their label in bold.

**PsEchoCancelerMMX:** All variables are allocated on the stack, to make the code reentrant, so that several versions of the same code can run at the same time on the machine.

**SnumLoop:** The 16-bit data size is hardcoded in the form of an add. A MUL instruction would have been more portable, but much slower.

**SnumLoop:** As a convention, all variables that must be defined before a loop entry point are listed just below the loop label, along with their associated register or memory location, under the name "Preconditions". Expected loop results are listed under the name "Postconditions". To emphasize the expected output of loops and also to limit the amount of comments, only loop results are listed as postconditions, not the variables that are supposed to live through the loop. Those variables are shown in the next precondition section. Hence, a postcondition section and the following preconditions section do not match as they theoretically should.

**CancelEchoLoop:** To reduce the number of instructions, the loop count is also used as a displacement, the loop count decrements to 0, and MMX instructions access memory.

**s computation:** The computation of the local variable stored in curs_mem could be improved. It would pair better (fewer register conflicts, easier merge with other MMX instructions later in the flow) if done in the MMX technology unit instead of in the integer side of the machine. Also, there is a three-cycle penalty for the first 16-bit memory access.

**ComputAdapt:** The ComputAdapt macro was developed to show that computations for hI and hQ are almost identical. In the macro, X means I or Q depending on the call.

**CancelerAdaptionLoop:** There are seven variables needed to perform memory accesses and loop control, and only 6 integer registers in the Intel Architecture, which forces two variables (dI_ and dQ_) to share the same register. Another solution would have been to split the loop in two, one loop to compute hI, one loop to compute hQ. The implementation chosen works well, because the variable swaps were successfully hidden in vacant slots of the pipeline.

**FiltnumLoop** termination: All array pointers are updated to reflect the filter change.

5.3 Potential Improvements

After going through code optimization, I realized that instruction pairing was limited by my implementation's heavy usage of the shift unit. Since only one shift, pack or unpack instruction can be executed in one cycle, several instructions of the CancelerAdaptionLoop did not pair in psecopt.asm.
While doing MMX code development, I recommend the code produced be inspected for its usage of the multiply and shift units, to balance the use of these units and potentially move computations to the ALU.
6.0. ALIGNED IMPLEMENTATION - psecalin.asm

Refer to Appendix B when reading this section.

6.1. Steps to Align the Code

The goal of this implementation is to align all memory accesses to avoid large data read and store latencies. To reach that goal, I:

- Aligned all local variables to their natural boundaries
- Created aligned versions of the dI and dQ arrays
- Computed quads instead of doubles in each iteration of CancelerAdaptionLoop.

6.2. Data Alignment Analysis of psecimpl.asm

Assumptions: the input arrays dI, dQ, hIH, hQH, hIL, and hQL are aligned to 64 bit boundaries (quadwords) in the calling code (typically in C). I met this constraint by declaring the arrays on the heap, declaring them as arrays of elements of type double (64 bit of size in Win32), and using the default Microsoft* MSVC 2.0 compiler option for Struct Data Alignment: 8 bytes. This way, the compiler aligned the arrays to a double boundary (64 bits). The double array pointers were then copied to short (16-bit) pointers. The s array is not a concern because all versions of the code do single data accesses for s, which are always aligned. Finally, the array length h_Leng must be a multiple of 4 * DATASIZE = 4 * 16 bits = 64 bits = quadword.

In the C code for the passband echo canceler (Figure 3), the following array offsets are used in array accesses:

hnum + snum

hnum + filtnum * h_Leng

Figure 3. Creation of the Aligned Versions of dI and dQ

Since there is only one memory access for each quadword, the loop counter hnum decrements by quad multiples, so array_base + hnum is always quad aligned if array_base is quad aligned. Since h_Leng is assumed to be a multiple of a quadword, filtnum * h_Leng is also a multiple of a quadword and array_base + hnum + filtnum * h_Leng is always a multiple of a quadword.
Therefore, snum is the only offset that generates unaligned accesses in the psecimpl.asm code. The only arrays using (hnum + snum) as offset are dI and dQ. To align all accesses to these arrays in psecalin.asm, dI and dQ are first copied into modified arrays in which aligned accesses can replace the unaligned accesses needed in the original code. The relationship between the original and the derived arrays is illustrated in Figure 3. Each element of the original array appears four times in the new array, as the 4th, 3d, 2d and 1st element of successive quads. The quad at (hnum + snum) in the original array is at (hnum + snum * 4) in the new array, and the new access is quadword aligned.

The Copy macro in Appendix B implements the copy of the old array into the new array. Also, the snum loop index is incremented by 4*DATASIZE in the SnumLoop termination code in Appendix B.

6.3. Overview of psecalin.asm

The following lines describe implementation decisions. Code portions are identified by their label in bold. Only differences with psecimpl.asm are mentioned.

Constants definition: The constant SLMAX is added. It is used in memory allocation for the new dI and dQ arrays. SLMAX should be chosen such that it is greater than (s_Leng + h_Leng) for all calls to PsEchoCancelerMMX.

PsEchoCancelerMMX: All variables are allocated on the heap here, instead of on the stack as in psecimpl.asm, because it is easier to align data structures on the heap than on the stack. Data structure alignment on the stack can be achieved through the STRUCT directive in Microsoft* Assembler (MASM) and runtime alignment of the base of the STRUCT. Since no benefit was seen in running several versions of the Echo Canceler on the same machine at the same time, code simplicity was selected over versatility. Therefore, psecalin.asm and psecopt.asm are not re-entrant.

Also, all variables are initialized to specific bit patterns, for debug purposes.

ArrayCopy loop: Creates aligned versions of dI and dQ. See "Data Annalysis of psecimpl.asm," Appendix A, for further details on the justification for this module. In the Copy macro, the original array is parsed in quads from the end to the beginning. dX1 contains the previous quad, dI0 is loaded with the new quad. dI4 is used to merge portions of dI0 and dX1. dI0 and dI4 are shared by the dI and dQ iterations of the macro.

ComputAdapt macro: This macro differs from the original one in psecimpl.asm in that quadwords instead of double words of data are processed in one macro call. This way, all data accesses are quadword aligned. Instructions are numbered to help visualize the flow of instructions in psecopt.asm.
7.0. OPTIMIZED IMPLEMENTATION - psecopt.asm

Refer to Appendix C when reading this section.

7.1. Steps to Fully Optimize the Code

The goal of this implementation is to fully optimize the code. To reach that goal, I:

- Remapped some variables to different registers, so as to move code pieces around.
- Reordered the code to utilize both the U and V pipes of the MMX technology unit, and to hide multiplies latencies.

7.2. Pairing Methodology

Typically, loops consist of three functionally distinct blocks:

1. loads from memory
2. computations
3. stores to memory

Either through loop enrolling or because the same succession of operations is performed on different data sets (I and Q), the loop can be modified to have the following format:

1. loads from memory (data set A)
2. computations (data set A)
3. stores to memory (data set A)
4. loads from memory (data set B)
5. computations (data set B)
6. stores to memory (data set B)

Since only independent instructions can be paired, one goal is to mix instructions from data set A and data set B. However, there are many constraints on the types of MMX instructions that can be paired, as is typically the case for superscalar processors. Currently, only one instruction in the MMX technology unit can access memory, the one in the U pipe. A solution is to move the load block for data set A to the end of the loop.

loads from memory (data set A) loop:
1. computations (data set A) merged w/ loads from memory(data set B)
2. stores to memory (data set A) merged w/ computations (data set B)
3. stores to memory (data set B)
4. loads from memory (data set A) for next iteration

This reordering implies the creation of a duplicate load block before the loop, and loads from potentially unimplemented memory before or after data set A (depending if the array is accessed from the end or the beginning) on the last iteration of the loop. This is why psecopt.asm (Appendix C) makes assumptions on the existence of padding areas before and after the input arrays.

In practice, computations, loads and stores do not use the same number of instructions, and some instructions can move between blocks of the same data set. Therefore, loads for data set A and stores for data set B may pair with instructions of computations of A and B, as it is the case in CancelerAdaptionLoop in Appendix C.
The method described above was used as a starting point to make pairing easier. From there, I used a trial and error approach to best pair instructions. The major difficulties to pairing were the PMUL and PMADD latency, and the single shift unit. While pairing, I found it critical to identify the longest string of mutually dependent instructions, and to work around it, filling the empty instruction slots with instructions from the next most constraining succession of instructions.

7.3. Overview of psecopt.asm

The following lines describe implementation decisions. Code portions are identified by their label **bold**. Only differences with Appendix B are mentioned.

**Assumptions:** Several new assumptions are present, due to the move of memory accesses to different loop iterations. Refer to Section 7.2, "Pairing Methodology," for a justification of the memory instructions move.

**PsEchoCancelerMMX:** Padding is also added before the arrays dIalign_mem and dQalign_mem because of the loop optimizations.

**ArrayCopy** loop setup: Instructions for snum loop setup and ArrayCopy loop setup are interleaved for better pairing.

**ArrayCopy:** The first step to pairing was to reorder instructions for better pairing within each data set. The following code shows how ArrayCopy instructions were reordered within each data set, to facilitate the global pairing of all instructions in the loop:

```plaintext
; Notation for loop instruction numbering:
;   qa03
;   ||V____ instruction number: from 01 to last instruction of loop
;   ||_____ iteration number: a for 1st iteration, b for 2d iteration
;   |______ data set: i for dI, q for dQ
;   I-set code numbering in place in
;       psecopt.asm psecalin.asm
; moved to next loop iteration: (if different)
mov itmp, [dI_+qcount*4-16*DATASIZE] ;ib01
movq dI0, [oldI+qcount-4*DATASIZE] ;ib02       ;i03
movdf [dI_+qcount*4-6*DATASIZE], dI1 ;ib03       ;i02
movq [dI_+qcount*4-16*DATASIZE], dI0 ;ib04
movq dI4, dI1 ;ib05
psllq dI4, 48 ;ib06
; I-set code in current loop

psrlq dI0, 16 ;ia07
por dI4, dI0 ;ia08
psrlq dI0, 16 ;ia09       ;i10
psllq dI1, 16 ;ia10       ;i13
movdf [dI_+qcount*4+8*DATASIZE], dI0 ;ia11
movq [dI_+qcount*4+4*DATASIZE], dI4 ;ia12       ;i09
psrlq dI0, 16 ;ia13       ;i12
por dI1, dI0 ;ia14
movq [dI_+qcount*4+12*DATASIZE], dI1 ;ia15
movq dI1, [oldI+qcount] ;ia16
; Q-set code
; Q-set computations in current set:
```
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

mov itmp, [dQ_+qcount*4] ;qa01
movdf [dQ_+qcount*4+10*DATASIZE], dQ1 ;qa02
movq dQ0, [oldQ+qcount] ;qa03
movq dQ4, dQ1 ;qa04 ;q05
movq [dQ_+qcount*4], dQ0 ;qa05 ;q04
psllq dQ4, 48 ;qa06
psrlq dQ0, 16 ;qa07
por dQ4, dQ0 ;qa08
psrlq dQ0, 16 ;qa09 ;q10
movq [dQ_+qcount*4+4*DATASIZE], dQ4 ;qa10 ;q09
psllq dQ1, 16 ;qa11 ;q13
movdf [dQ_+qcount*4+8*DATASIZE], dQ0 ;qa12 ;q11
psrlq dQ0, 16 ;qa13 ;q12
por dQ1, dQ0 ;qa14
movq [dQ_+qcount*4+12*DATASIZE], dQ1 ;qa15
movq dQ1, [oldQ+qcount] ;qa16

Memory accesses ia 01 through ia04, plus two additional instructions (ia05, ia06) are moved to the preceding loop iteration. Separate MMX registers are assigned to dI0, dI4, dQ0, and dQ4, to allow for instruction reordering. Array offsets of instructions moved from a loop iteration to another are adjusted accordingly.

**FiltnumLoop:** The life of the register variable filtnum loop is shortened to very limited places within the loop, to free up the register for other usage.

**CancelEchoLoop:** The loop is unrolled once and all I-related loads are moved to the preceding loop iteration. The loop is limited by all the memory accesses necessary to perform the computation. Here, several loads and computations could probably be broken into separate instructions at no cycle cost, because of the three vacant V pipe slots.

**Computation of s,** loop setup for CancelerAdaptionLoop: Computation of s and CancelerAdaptionLoop setup are interleaved. For that, sptr, tmp, sdl, sdQI, sdlh, and sdQh were remapped to different registers. The computation of the local variable stored in curs_mem could be improved. It would pair better (fewer register conflicts, easier merge with other MMX instructions later in the flow) if done in the MMX technology unit instead of on the integer side. Also, there is a three-cycle penalty for the first 16-bit memory access.

**CancelerAdaptionLoop:** The approach taken was to first pair the I-data set instructions among themselves as much as possible, by reordering instructions within the I-data set as done in the ArrayCopy loop above, and by moving sdI1 computations to the previous iteration. The instruction numbering relates to the original unpaired and unordered version of ComputAdapt in Appendix B. Then, remaining unpaired I and Q instructions were paired by combining I and Q instructions. 6 slots were left vacant, due to the unique shifter and the PMADD latency. Better results could potentially be achieved by considering the shifter, the multiply unit as well as memory accesses when choosing the instructions to move from one iteration to another.

### 7.4 Potential Improvements

Most loops still have a few slots left, so pairing could potentially be better. I recommend starting again from psecalin.asm (Appendix B), re-pairing the loops you think you could improve, and comparing your results to the ones obtained for psecopt.asm (Appendix C) using Wdis.
7.5. Performance Numbers

The routine PsEchoCancelerMMX was called 200 times, including the cache warmup call, with the following parameters passed to the routine:

\[ s\_Leng = 40 \]

\[ h\_Leng = 48 \]

PsEchoCancelerMMX took 134 cycles to execute on average over the 200 calls. This number is slightly less than the number given by Wdis (137), because Wdis counts penalties for instruction dependencies crossing loop labels, when the actual code incurs such penalties only during the first pass through the loop. Since psecopt.asm has 212 instructions, instructions took 0.63 cycles to execute on average, which indicates that instructions paired well.
APPENDIX A

; File: psecimpl.asm
;
; DESCRIPTION:
; This is the simple version of the Passband Echo Canceler, showing
; the functionality of the code.
;
;**************************************************************************
;
; ASSUMPTIONS:
; ============
; + array sizes: dI,dQ [h_leng + s_Leng]
; s [3 * s_leng]
; hIH,hQH [3 * h_leng]
; hIL,hQL [3 * h_leng]
;
; + h_Leng is a multiple of 4 * DATASIZE, because 4 elements are accessed at
; one time in CancelEchoLoop. The arrays hIH and hQH must be
; padded with zeroes if necessary.
;
; + s_Leng is a multiple of 4 * DATASIZE, because 4 elements are accessed at
; one time in CancelEchoLoop. The arrays dI, dQ and s must be
; padded with zeroes if necessary.
;
; + s Leng, h Leng must have positive values, because loops are executed
; at least once before these variables are checked.
;
;
; TYPE OF COMPUTATIONS
; ====================
;
; All computations are integer computations.
;
;**************************************************************************/
;
; PSEUDO CODE OF ALGORITHM
; ===============
;
; void psEchoCanceler(short *dI,short *dQ, short *s,
;                     short *hIH, short *hQH, short *hIL, short *hQL,
;                     short h_Leng, short s_Leng)
; {
;     long y,adapt;
;     short hnum,filtnum, snum, MU=3;
;     for (snum=0; snum<s_Leng; snum++) // signal length
;     {
;         for (filtnum=0; filtnum<3; filtnum++) // 3 filters
;         {
;             //--------- cancel echo --------
;             for (y=0, hnum=h_Leng-1; hnum>=0; hnum--)
;             {
;                 y+=dI[hnum+snum]*(long)hIH[hnum+filtnum*h_Leng]
;                 -dQ[hnum+snum]*(long)hQH[hnum+filtnum*h_Leng];
;             }
;         }
;     }
; }
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

```c
;     s[filtnum+3*snum]=s[filtnum+3*snum]-(short)(y>>14);
;
; //----- canceler adaption ---------
; for (hnum=0; hnum<h_Leng; hnum++)
; {
;    adapt=((long)hIH[hnum+filtnum*h_Leng])<<16
;     |((long)hIL[hnum+filtnum*h_Leng])&0x0000ffff;
;    adapt=adapt+((s[filtnum+3*snum]*(long)dI[hnum+snum])>>MU);
;    hIH[hnum+filtnum*h_Leng]=adapt>>16;
;    hIL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
;
;    adapt=((long)hQH[hnum+filtnum*h_Leng])<<16
;     |((long)hQL[hnum+filtnum*h_Leng])&0x0000ffff);
;    adapt=adapt-((s[filtnum+3*snum]*(long)dQ[hnum+snum])>>MU);
;    hQH[hnum+filtnum*h_Leng]=adapt>>16;
;    hQL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
; 
; }
; }

/***************************************************************************/

; PSEUDO CODE OF SIMPLE IMPLEMENTATION
; ====================================
; Notation: `[addr]16' means value of 16 bit element at address addr
;
; void psEchoCanceler(short *dI,short *dQ, short *s,
;                     short *hIH, short *hQH, short *hIL, short *hQL,
;                     short h_Leng, short s_Leng)
; {
;    long y,adapt;
;    short hnum,filtnum, snum;
;    #define MU=3, DATASIZE=2, TOTFILTS=3;
;
;    s_LengDS_mem = s_Leng * DATASIZE;
;    h_LengLQ_mem = (h_Leng-4) * DATASIZE;
;    h_LengLD_mem = (h_Leng-2) * DATASIZE;
;
;    //------------- SnumLoop ----------------------------
; for (snum=0; snum<s_LengDS; snum=snum+DATASIZE)
; {
;     //--------- FiltnumLoop --------------------------
;     for (filtnum= 0; filtnum<TOTFILTS; filtnum=filtnum+DATASIZE)
;     {
;         //----- CancelEchoLoop -------------
;         for (y=0, hnum=h_LengLQ; hnum>=0; hnum - 4 * DATASIZE)
;         { // 4 hnum values processed in one loop iteration
;             y += -(dI+hnum+snum)16 * [hIH+hnum+filtnum*h_Leng]16
;             +[dQ+hnum+snum]16 * [hQH+hnum+filtnum*h_Leng]16
;             -[dI+hnum+snum+DATASIZE]16 * [hIH+hnum+filtnum*h_Leng +
;             DATAIZE]16
;             +[dQ+hnum+snum+DATASIZE]16 * [hQH+hnum+filtnum*h_Leng +
;             DATAIZE]16
;             -[dI+hnum+snum+2*DATASIZE]16 *
;             [hIH+hnum+filtnum*h_Leng+2*DATASIZE]16
;             
;         }
;     }
; }
; }```

16
MACRO: ComputAdapt(dX, hXH, hXL, sign)
    adapt32 = ((([hXH+hnum+filtnum*h_Leng]16)<<16)
    |([hXL+hnum+filtnum*h_Leng]16)&0x0000ffff))32;
    if (sign=="+") adapt32 = adapt32 +(([s+filtnum+3*snum]16 * [dX+hnum+snum]16)>>MU);
    else adapt32 = adapt32 -(([s+filtnum+3*snum]16 * [dX+hnum+snum]16)>>MU);
    [hXH+hnum+filtnum*h_Leng]=(adapt32>>16)16;
    [hXL+hnum+filtnum*h_Leng]=(adapt32&0x0000ffff)16;
ENDMACRO

PsEchoCancelerMMX    PROC C uses ebx ecx edx esi edi,
    dI_base:PTR WORD,  dQ_base:PTR WORD,  s_base:PTR WORD,
    hIH_base:PTR WORD, hQH_base:PTR WORD,
    hIL_base:PTR WORD, hQL_base:PTR WORD,
    h_Leng:DWORD, s_Leng:DWORD
LOCAL   snum_mem:    DWORD      ; loop count value

PsEchoCancelerMMX
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

LOCAL   sptr_mem: DWORD         ; s[3 * snum] - should be quad aligned for fast access
LOCAL   filtnum_mem: DWORD      ; loop count value
LOCAL   s_LengDS_mem: DWORD     ; s_Leng * DATASIZE to speed up loop iterations
LOCAL   h_LengDS_mem: DWORD     ; h_Leng * DATASIZE for hIH...hQL updates
LOCAL   h_LengLD_mem: DWORD     ; pointer to last doublet:
    ; (h_Leng-2) * DATASIZE to speed up loop iterations
LOCAL   h_LengLQ_mem: DWORD     ; pointer to last quad:
    ; (h_Leng-4) * DATASIZE to speed up loop iterations
LOCAL   dI_mem:   DWORD         ; needed because of register conflict between
LOCAL   dQ_mem:   DWORD         ; dI_ and dQ_ in canceler adaption
LOCAL   curs_mem: QWORD         ; repository for 16 bit value of current
    ; 64 bit value is:   0  |  s  | 0  |  s
    ; Should be quad aligned for fast access
LOCAL   hIL_mem:  DWORD         ; storage for hIL[filtnum*h_Leng]
LOCAL   hQL_mem:  DWORD         ; storage for hQL[filtnum*h_Leng]

;=================== snum loop ===========================

; Locals:
; name         x86 register              life span
; snum     TEXTEQU <edi>         ; lives at entrance and exit of snum loop
; itmp     TEXTEQU <esi>         ; lives when filtnum does not
; dI_     TEXTEQU <eax>         ; lives until middle of CancelerAdaption
; dQ_     TEXTEQU <ebx>         ; lives until beginning of CancelerAdaption

;------------------- LOOP SETUP ---------------------------
mov     itmp, s_base    ; initialize sptr_mem
mov     sptr_mem, itmp
mov     itmp, s_Leng
add     itmp, itmp        ; DATASIZE = 2  is hardcoded here
mov     s_LengDS_mem, itmp    ; s_LengDS_mem = s_Leng * DATASIZE
mov     itmp, h_Leng
add     itmp, itmp        ; DATASIZE = 2  is hardcoded here
mov     h_LengDS_mem, itmp    ; h_LengDS_mem = h_Leng * DATASIZE
sub     itmp,     ; DATASIZE + DATASIZE
mov     h_LengLD_mem, itmp    ; h_LengLD_mem = (h_Leng-2) * DATASIZE
sub     itmp,     ; DATASIZE + DATASIZE
mov     h_LengLQ_mem, itmp    ; h_LengLQ_mem = (h_Leng-4) * DATASIZE

DATASIZE

; initialize loop counter
mov     snum, 0

SnumLoop:  ;-------- LOOP START ---------------------------

; Preconditions:
; Name    value          register
; snum    new value of loop counter    edi
; dI_     undefined        eax
; dQ_     undefined        ebx
;
mov     dI_, dI_base
mov     dQ_, dQ_base
add     dI_, snum        ; create dI_[snum] pointer
add     dQ_, snum        ; create dQ_[snum] pointer
mov     dI_mem, dI_      ; store dI_ and dQ_ for canceler
mov     dQ_mem, dQ_      ; adaption: they use the same physical
    ; register there
mov     snum_mem, snum   ; store the counter, now can modify the register
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

mov     itmp, hIL_base    ; create hIL[filtnum*h_Leng]
mov     hIL_mem, itmp
mov     itmp, hQL_base    ; create hQL[filtnum*h_Leng]
mov     hQL_mem, itmp

================= filtnum loop ===================

Locals:
  filtnum TEXTEQU <esi>         ; replaces itmp in this loop
  hIH     TEXTEQU <ecx>
  hQH     TEXTEQU <edx>

------------------- LOOP SETUP -------------------
mov     hIH, hIH_base
mov     hQH, hQH_base

; initialize loop counter
mov     filtnum, 0

FiltNumLoop: ;- LOOP START -----------------------------
  ; Preconditions:
  ; Name         value                        register
  ; hIH       hIH[filtnum * h_Leng]              ecx
  ; hQH       hQH[filtnum * h_Leng]              edx
  ; filtnum    new value                          esi
  ; dI_       dI_[snum]                          eax
  ; dQ_       dQ_[snum]                          none - in memory: dQ_mem

  mov     dQ_, dQ_mem     ; destroyed by reg conflict with dI_
mov     filtnum_mem, filtnum ; store variable

================= cancel echo loop ===========================

Locals:
  hnum    TEXTEQU <edi>   ;; loop counter and index
  y       TEXTEQU <mm0>
dIhIH    TEXTEQU <mm1>
dQhQH    TEXTEQU <mm2>

------------------- LOOP SETUP -------------------
mov     hnum, h_LengLQ_mem          ; point to last quad:
        h_LengLQ_mem = (h_Leng-4) * DATASIZE
pxor    y,y                     ; y = 0

CancelEchoLoop: ;-- LOOP START -----------------------------
  ; Preconditions:
  ; Name         value                        register
  ; hIH       hIH[filtnum * h_Leng]              ecx
  ; hQH       hQH[filtnum * h_Leng]              edx
  ; dI_       dI_[snum]                          eax
  ; dQ_       dQ_[snum]                          ebx
  ; filtnum    current filter                    esi
  ; hnum      current loop count value            edi
  ; y         cumulative result so far            mm0

  ; Line numbering used to improve
  ; readability of paired implementation
  ; in psecopt.asm ____________________

  Data flow in 64 bit MMX register:

  movq    dIhIH, [dI_ + hnum] ;1a;   dI3 | dI2 | dI1 | dI0
movq    dQhQH, [dQ_ + hnum] ;3a;   dQ3 | dQ2 | dQ1 | dQ0
psubd dQhQH,dIHhI     ;5a;
padd y, dQhQH         ;6a; S2;3[dQ.hQH-dI.hIH] | S0;1[dQ.hQH-dI.hIH]
sub hnum, 4 * DATASIZE  ; decrement pointer
jge CancelEchoLoop

; CancelEchoLoop -- LOOP END -----------------------------

; Postconditions:
; Name      value        register
; y          result of computation mm0

; Locals:
sptr   TEXTEQU <edi> ; replaces hnum
itmp   TEXTEQU <ebx> ; replaces dQ_ 
itmpshort TEXTEQU <bx> ; lower 16 bits of itmp
tmp   TEXTEQU <mm7> ; for local computations

; compute final y, compute s
movq tmp, y
psrlq tmp, 32
padd y, tmp                ; S          | S[...]
srld y, 14                  ; y >> 14
movdf itmp, y
mov sptr, [sptr_mem]
add itmpshort, [sptr + filtnum] ; y + s

; 16 bit computation = s
mov [sptr + filtnum], itmpshort ; 16 bit mem access
mov filtnum_mem, filtnum ; store variable
; for canceler adaption, store s[filtnum+3*snum] as:
; 0 | s | 0 | s
and itmp, 0FFFFh             ;                           0   |   s
mov DWORD PTR curs_mem, itmp
mov DWORD PTR curs_mem[4], itmp   ;     0  |   s   |   0   |   s

;================= canceler adaption loop =======================

; Register mapping in this loop:
; Name      value        register
; hIH     hIH[filtnum * h_Leng]        ecx
; hQH     hQH[filtnum * h_Leng]       edx
; dI_    dI_[snum]                    eax | 1st part of loop
; dQ_    dQ_[snum]                    eax | 2d part of loop
; filtnum  current filter            esi | before loop
; hIL     hIL[filtnum * h_Leng]     esi  | in loop
; hQL     hQL[filtnum * h_Leng]      ebx
; hnum   hnum

; Locals:

hnum   TEXTEQU <edi> ; loop counter and index
hIL TEXTEQU <esi>
hQL TEXTEQU <ebx> ; 2d part of loop
dQ_ TEXTEQU <eax> ; conflicts with dI_: runtime swaps w/ mem

; saves and restores

;------------------- LOOP SETUP ---------------------------

mov hQL, hQL_mem ; load hQL[filtnum * h_Leng] pointer
; trick: hIL reg = filtnum reg
; filtnum terminates its life here, hIL starts its
mov hIL, hIL_mem ; load hIL[filtnum * h_Leng] pointer
; initialize hnum loop counter and index variable
mov hnum, h_LengLD_mem ; point to last pair

; h_LengLD_mem = (h_Leng-2) * DATASIZE

mov dQ_, TEXTEQU <eax> ; conflicts with dI_: runtime swaps w/ mem

sdI TEXTEQU <mm0>
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

sdQ   TEXTEQU <mm0>         ; also maps to mm0 because used only
     ; after death of sdI

// Global variables:
newhXH TEXTEQU <mm2>
adapt TEXTEQU <mm3>

;---------- MACRO DEFINITION START ---------------------------------
ComputAdapt MACRO  dX_:REQ, sdX:REQ, hXH:REQ, hXL:REQ, iter:REQ
    ; Global variables:
    ; newhXH is defined
    movdt   adapt, [hXL + hnum]     ;           ....    | hIL1 | hIL0
    movdt   sdX, [dX_ + hnum]       ;           ....    |  dI1 |  dI0
    ; adapt computation
    punpcklwd adapt, [hXH + hnum]   ;       hIH1 | hIL1 | hIH0 | hIL0
    ; =        hI1     |     hI0
    punpcklwd sdX, sdX              ;        dI1 |  dI1 |  dI0 |  dI0
    pmaddwd sdX, DWORD PTR curs_mem ;          s.dI1    |   s.dI0
    psrad   sdX, MU                 ;     s.dI1 >> MU   |  s.dI0 >> MU
    % IF iter EQ 1
    ;; dI_        paddd   adapt, sdX              ;      hIH1 |  hIL1 | hIH0 | hIL0  : RESULT
    ELSE        ;; dQ_        psubd   adapt, sdX
    ENDDIF
    ; now save the result
    movq    newhXH, adapt
    psrlq   newhXH, 32              ;       ... |  ...  | hIH1 | hIL1
    punpcklwd adapt, newhXH         ;      hIH1 | hIH0 | hIH1 | hIL0
    movdf   [hXL + hnum], adapt
    psrlq   adapt, 32               ;       ... |  ...  | hIH1 | hIH0
    movdf   [hXH + hnum], adapt
EXITM <>
ENDM   ;; end of ComputAdapt
;---------- MACRO DEFINITION END ---------------------------------
CancelerAdaptionLoop: ;-- LOOP START -----------------------------
    ; Preconditions:
    ; Name         value                        register
    ; hIH         hIH[filtnum * h_Leng]             ecx
    ; hQH         hQH[filtnum * h_Leng]             edx
    ; dI_        dI_[snum]                          eax
    ; dQ_        undefined                           (eax)
    ; filtnum     undefined                           (esi)
    ; hIL         hIL[filtnum * h_Leng]             esi
    ; hQL         hQL[filtnum * h_Leng]             ebx
    ; hnum        new hnum value                     edi

    ComputAdapt(dI_, sdI, hIH, hIL,1)
    mov    dQ_, dQ_mem                          ; restore dQ_, kill dI_  
    Compu...
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

; hQH_base new filter coefficients hQH_base
; hIL_base new filter coefficients hIL_base
; hQL_base new filter coefficients hQL_base
;
    add    hIH, h_LengDS_mem            ; update hIH[filtnum*h_Leng] pointer
    add    hQH, h_LengDS_mem            ; update hQH[filtnum*h_Leng] pointer
    add    hIL, h_LengDS_mem            ; update hIL[filtnum*h_Leng] pointer
    add    hQL, h_LengDS_mem            ; update hQL[filtnum*h_Leng] pointer
    mov    hIL_mem, hIL                ; store hIL, hQL values for next
    mov    hQL_mem, hQL                ; iteration
    ; FiltnumLoop termination
    mov    filtnum, filtnum_mem        ; access loop counter in memory
    add    filtnum, DATASIZE            
    cmp    filtnum, TOTFILTS * DATASIZE
    jl     FiltnumLoop
    ; FiltnumLoop ------------- LOOP END ------------------------------
    mov    itmp, sptr_mem
    add    itmp, 3 * DATASIZE            
    mov    sptr_mem, itmp               ; = s_[3*snum/4]
    ; SnumLoop termination
    mov    snum, snum_mem               ; access loop counter in memory
    add    snum, DATASIZE
    cmp    snum, s_LengDS_mem
    jl     SnumLoop
    ; SnumLoop -------------- LOOP END ------------------------------
    emms
    ret
PsEchoCancelerMMx EndP
END
APPENDIX B

; File: psecalin.asm
;
; DESCRIPTION:
; This is the aligned version of the Pass Band Echo Canceler, derived
; from the simple implementation psecimpl.asm. All memory accesses are aligned.
;
;******************************************************************************
;aisonments:
; ============
; array      size
; + array sizes:  dI,dQ   \[h_leng + s_Leng\]
; s       \[3 * s_leng\]
; hIH,hQH [3 * h_leng]
; hIL,hQL [3 * h_leng]
;
; + h_Leng is a multiple of 4 * DATASIZE, because 4 elements are accessed at
; one time in CancelEchoLoop. The arrays hIH and hQH must be
; padded with zeroes if necessary.
;
; + s_Leng is a multiple of 4 * DATASIZE, because 4 elements are accessed at
; one time in CancelEchoLoop. The arrays dI, dQ and s must be
; padded with zeroes if necessary.
;
; + s Leng, h Leng must have positive values, because loops are executed
; at least once before these variables are checked.
;
; + the input arrays dI, dQ, hIH, hQH, hIL, hQL are aligned on a 4 * DATASIZE
; boundary.
;
; + the input array s is aligned on a DATASIZE boundary.
;
;
; TYPE OF COMPUTATIONS
; ====================
; All computations are integer computations.
;
;******************************************************************************
;
; PSEUDO CODE OF ALGORITHM
; ================
;
; void psEchoCanceler(short *dI, short *dQ, short *s,
;                      short *hIH, short *hQH, short *hIL, short *hQL,
;                      short h_Leng, short s_Leng)
; {
;   long y,adapt;
;   short hnum,filtnum, snum, MU=3;
;   for (snum=0; snum<s_Leng; snum++) // signal length
;   {
;     for (filtnum=0; filtnum<3; filtnum++) // 3 filters
;     {
;       //------- cancel echo ---------
Using MMX™ Instructions to Implement a Modem Passband Canceler
March 1996

PSEUDO CODE OF ALIGNED IMPLEMENTATION

```c
void psEchoCanceler(short *dI, short *dQ, short *s,
                     short *hIH, short *hQH, short *hIL, short *hQL,
                     short h_Leng, short s_Leng)
{
    long y, adapt;
    short hnum, filtnum, snum, MU=3;

    ArrayCopy(dI);
    ArrayCopy(dQ);
    for (snum=0; snum<s_Leng; snum++) // signal length
    {
        for (filtnum=0; filtnum<3; filtnum++) // 3 filters
        {
            //------ cancel echo -----
            for (y=0, hnum=h_Leng-1; hnum>=0; hnum--)
            {
                y+=dI[hnum+snum]*(long)hIH[hnum+filtnum*h_Leng]
                -dQ[hnum+snum]*(long)hQH[hnum+filtnum*h_Leng];
            }
            s[filtnum+3*snum]=s[filtnum+3*snum]-(short)(y>>14);
        }

        //----- canceler adaptation -------
        for (hnum=0; hnum<h_Leng; hnum++)
        {
            adapt=(((long)hIH[hnum+filtnum*h_Leng])<<16)
                   |(((long)hIL[hnum+filtnum*h_Leng])&0x0000ffff);
            adapt=adapt+((s[filtnum+3*snum]*(long)dI[hnum+snum])>>MU);
            hIH[hnum+filtnum*h_Leng]=adapt>>16;
            hIL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
        }
    }
}
```

Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

;     hIH[hnum+filtnum*h_Leng]=adapt>>16;
;     hIL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
;
;     adapt=(((long)hQH[hnum+filtnum*h_Leng])<<16)
;             |(((long)hQL[hnum+filtnum*h_Leng])&0x0000ffff);
;     adapt=adapt-((s[filtnum+3*snum]*(long)dQ[hnum+snum])>>MU);
;     hQH[hnum+filtnum*h_Leng]=adapt>>16;
;     hQL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;

} })

where:

ArrayCopy(oldX, dX)
{
    for (qcount = h_Leng + s_Leng -1; qcount >= 0 ; qcount--)
    {
        dX[qcount    *4  ] = oldX[qcount];
        dX[(qcount-1)*4+1] = oldX[qcount];
        dX[(qcount-2)*4+2] = oldX[qcount];
        dX[(qcount-3)*4+3] = oldX[qcount];
    }
}
title             PsEchoCancelerMMX

Constants:

MU       = 3         ;; adaptation step size
DATASIZE = 2         ;; in bytes. Revisit beginning of snum loop if this number
                    ;; changes (hardcoded there).
TOTFILTS = 3         ;; number of filters
SLMAX    = 88        ;; upper limit of: s_Leng + h_Leng

.data      ; linker should align this segment to a 32 byte boundary
ALIGN 16
dI_algn_mem WORD SLMAX*4 DUP (011111H)    ; aligned version of dI_ for all accesses
ALIGN 16
dQ_algn_mem WORD SLMAX*4 DUP (0AAAAAH)    ; aligned version of dQ_ for all accesses
ALIGN 8
curs_mem  QWORD (0EEEEEEEEH)    ; repository for 16 bit value of current
s[snum]   WORD (011111111H)        ;; must be quad aligned for fast access
s[snum]   DWORD (0222222222H)      ;; snum loop count value
filt[num] DWORD (011111111H)       ;; filt[num] loop count value
s_LengQ_mem DWORD (0222222222H)    ;; s_Leng * DATASIZE * 4 to speed up loop

iterations
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

h_LengDS_mem DWORD (011111111H) ; h_Leng * DATASIZE for hIH...hQL updates
h_LengLQ_mem DWORD (011111111H) ; pointer to last quad:
                        ; (h_Leng-4) * DATASIZE to speed up loop

iterations
dI_mem    DWORD (022222222H)            ; needed because of register conflict between
dQ_mem    DWORD (011111111H)            ; dI_ and dQ_ in canceler adaption
hIL_mem   DWORD (022222222H)            ; storage for hIL[filtnum*h_Leng]
hQL_mem   DWORD (011111111H)            ; storage for hQL[filtnum*h_Leng]
        .code
PsEchoCancelerMMX PROC C uses ebx ecx edx esi edi,
             dI_base:PTR WORD,  dQ_base:PTR WORD,  s_base:PTR WORD,
             hIH_base:PTR WORD, hQH_base:PTR WORD,
             hIL_base:PTR WORD, hQL_base:PTR WORD,
             h_Leng:DWORD, s_Leng:DWORD
;=================== snum loop ===========================
; Locals:
snum     TEXTEQU <edi>         ; lives at entrance and exit of snum loop
itmp     TEXTEQU <esi>         ; lives when filtnum does not
    dI_ TEXTEQU <eax>         ; lives until middle of CancelerAdaption
    dQ_ TEXTEQU <ebx>         ; lives until beginning of CancelerAdaption
qcount   TEXTEQU <ecx>         ; loop counter for ArrayCopy
;------------------- LOOP SETUP ---------------------------
mov     itmp, s_base    ; initialize sptr_mem
mov     sptr_mem, itmp
mov     qcount, h_Leng
add     qcount, qcount    ; DATASIZE = 2  is hardcoded here
mov     h_LengDS_mem, qcount  ; h_LengDS_mem = h_Leng * DATASIZE
sub     qcount, DATASIZE + DATASIZE
sub     qcount, DATASIZE + DATASIZE
mov     h_LengLQ_mem, qcount  ; h_LengLQ_mem = (h_Leng-4) * DATASIZE
;================= ArrayCopy loop =========================
;    creation of aligned versions of dI and dQ
;------------------- LOOP SETUP ---------------------------
mov     oldI, dI_base
mov     oldQ, dQ_base
lea     dI_, dIalgn_mem
lea     dQ_, dQalgn_mem
; initialize the counter
mov     itmp, s_Leng
add     itmp, itmp      ; DATASIZE = 2  is hardcoded here
add     qcount, qcount    ; (h_Leng+s_Leng-4) * DATASIZE
add     itmp, itmp
add     itmp, itmp
mov     s_LengQ_mem, itmp  ; s_LengQ_mem = s_Leng * DATASIZE * 4
pxor    dI1, dI1
pxor    dQ1, dQ1
;-------------- MACRO DEFINITION START ---------------------
Copy  MACRO  dX_: REQ, dX1:REQ, oldX:REQ
    ; Notation for loop instruction numbering:
    ; i03
    
EndPsEchoCancelerMMX
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

; |V___ instruction number: from 01 to last instruction of loop
; |_____ data set: i for dI, q for dQ
; bring the 32 byte line into the cache before writing to it
; because the cache does not allocate on writes
mov    itmp, [dX_+qcount*4] ;i01
        ;dI1= d7|  d6  |  d5  |  d4
movdf  [dX_+qcount*4+10*DATASIZE], dX1 ;i02; W d5 |  d4 | - | -
movq   dI0, [oldX+qcount] ;i03;
movq   [dX_+qcount*4], dI0 ;i04; W d3 |  d2 | d1 | d0
movq   dI4, dX1 ;i05;
psslq  dI4, 48 ;i06; d4 |  0  |  0  |  0
psrlq  dI0, 16 ;i07;  0  |  d3  |  d2  |  d1
por    dI4, dI0 ;i08;  d4  |  d3  |  d2  |  d1
movq   [dX_+qcount*4+4*DATASIZE], dI4 ;i09; W d4 |  d3 |  d2 |  d1
psrlq  dI0, 16 ;i10
movdf  [dX_+qcount*4+8*DATASIZE], dI0 ;i11; W -  |  -  |  d3 |  d2
psrlq  dI0, 16 ;i12;  0  |  0  |  0  |  d3
psrlq  dX1, 16 ;i13;  d6  |  d5  |  d4  |  0
por    dX1, dI0 ;i14;
movq   [dX_+qcount*4+12*DATASIZE], dX1 ;i15; W d6 |  d5 |  d4 |  d3
movq   dX1, [oldX+qcount] ;i16; for next iteration
EXITM>
ENDM

;-------------- MACRO DEFINITION END ----------------------

ArrayCopy: ;---------- LOOP START -------------------------

; Preconditions:
; Name       value                          register
; qcount     current counter value          ecx
; dI_        pointer to new, aligned dI     eax
; dQ_        pointer to new, aligned dQ     ebx
; itmp       temporary variable             esi
; oldI       pointer to old dI              edx
; oldQ       pointer to old dQ              edi    maps over snum
; dI0        undefined                       (mm0)
; dI1        content of quad 1 of dI values (mm1)
; dQ1        content of quad 1 of dQ values (mm2)
; dI4        undefined                       (mm3)
;
    Copy(dI_, dI1, oldI)
    Copy(dQ_, dQ1, oldQ)
sub     qcount, 4 * DATASIZE
jge    ArrayCopy

;ArrayCopy ---------- LOOP END ------------------------

; Postconditions:
; Name       value                          memory location
; dIalgn_mem 4 copies of the original dI     dIalgn_mem
; dQalgn_mem 4 copies of the original dQ     dQalgn_mem
; initialize loop counter
mov     snum, 0

SnumLoop: ;-------- LOOP START -----------------------

; Preconditions:
; Name       value                          register
; snum       new value of loop counter       edi
; dI_        undefined                       edx
; dQ_        undefined                       ebx
;
    ;-- compute actual snum
lea    dI_, dIalign_mem
lea    dQ_, dQalign_mem
add    dI_, snum         ; create dI_[snum] pointer
add    dQ_, snum         ; create dQ_[snum] pointer
mov    dI_mem, dI_       ; store dI_ and dQ_ for canceler
mov    dQ_mem, dQ_       ; adaption: they use the same physical
                       ; register there
mov    snum_mem, snum    ; store the counter, now can modify the register
mov    itmp, hIL_base    ; create hIL[filtnum*h_Leng]
mov    hIL_mem, itmp
mov    itmp, hQL_base    ; create hQL[filtnum*h_Leng]
mov    hQL_mem, itmp

;================= filtnum loop ============================

; Locals:
filtnum        TEXTEQU <esi>         ; replaces itmp in this loop
hIH             TEXTEQU <ecx>
hQH             TEXTEQU <edx>

;----------------- LOOP SETUP ---------------------------
mov    hIH, hIH_base
mov    hQH, hQH_base
; initialize loop counter
mov    filtnum, 0
FiltnumLoop: ;-- LOOP START ---------------------------

; Preconditions:
; Name         value                        register
; hIH       hIH[filtnum * h_Leng]              ecx
; hQH       hQH[filtnum * h_Leng]              edx
; filtnum    new value                          esi
; dI_       dI_[snum]                          eax
; dQ_       dQ_[snum]                          ebx

mov    dQ_, dQ_mem     ; destroyed by reg conflict with dI_
                       ; in adaption loop
mov    filtnum_mem, filtnum ; store variable

;================= cancel echo loop ===========================

; Locals:
hnum        TEXTEQU <edi>   ;; loop counter and index
y            TEXTEQU <mm0>
dIhIH        TEXTEQU <mm1>
dQhQH        TEXTEQU <mm2>

;----------------- LOOP SETUP ---------------------------
mov    hnum, h_LengLQ_mem          ; point to last quad:
                      ; h_LengLQ_mem = (h_Leng-4) * DATASIZE
pxor    y,y                     ; y = 0
CancelEchoLoop: ;-- LOOP START ---------------------------

; Preconditions:
; Name         value                        register
; hIH       hIH[filtnum * h_Leng]              ecx
; hQH       hQH[filtnum * h_Leng]              edx
; dI_       dI_[snum]                          eax
; dQ_       dQ_[snum]                          ebx
; filtnum    current filter                    esi
; hnum    current loop count value              edi
; y        cumulative result so far              mm0


Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

; Line numbering used to improve readability of paired implementation
; in psecotp.asm ____________________
;                                     | Data flow in 64 bit MMX register:
;                                     v
movq    dIhIH, [dI_ + hnum*4];1a;        dI3 | dI2 | dI1 | dI0  
movq    dQhQH, [dQ_ + hnum*4];3a;        dQ3 | dQ2 | dQ1 | dQ0  
psubd   dQhQH,dIhIH          ;5a;  
paddd   y, dQhQH             ;6a; S2;3[dQ.hQH-dI.hIH] | S0;1[dQ.hQH-dI.hIH]  
sub     hnum, 4 * DATASIZE      ; decrement pointer  
je     CancelEchoLoop

; CancelEchoLoop -- LOOP END -----------------------------
; Postconditions:
;  Name         value                        register
;  y         result of computation             mm0
; Locals:

; compute final y, compute s
movq    tmp, y  
psrlq   tmp, 32  
padd   y, tmp                  ;           ......      |  S[dQ.hQH-dI.hIH]  
psrld   y, 14                   ;           ......      |    y >> 14
movdf   itmp, y
mov     sptr, [sptr_mem]  
add     itmpshort, [sptr + filtnum]   ;      ......     |   ...   | y + s
; 16 bit computation  
 mov     [sptr + filtnum], itmpshort ; 16 bit mem access  
 mov     filtnum_mem, filtnum ; store variable  
 ; for canceler adaption, store s[filtnum+3*snum] as:  
 ; 0 | s | 0 | s  
 and     itmp, 0FFFFh            ;                           0   |   s
mov     DWORD PTR curs_mem, itmp
mov     DWORD PTR curs_mem[4], itmp

;================= canceler adaption loop =======================
; Register mapping in this loop:
; Name         value                        register
;  hIH       hIH[filtnum * h_Leng]              ecx
;  hQH       hQH[filtnum * h_Leng]              edx
;  dI_       dI_[snum]                          eax  | 1st part of loop
;  dQ_       dQ_[snum]                          eax  | 2d part of loop
;  filtnum   current filter                     esi  | before loop
;  hIL       hIL[filtnum * h_Leng]              edi  | in loop
;  hQL       hQL[filtnum * h_Leng]              ebx
;  hnum      hnum                                edi
; Locals:

;------------------- LOOP SETUP ---------------------------

mov     hQL, hQL_mem ; load hQL[filtnum * h_Leng] pointer

; Postconditions:
;  Name         value                        register
;  y         result of computation             mm0
; Locals:

; compute final y, compute s
movq    tmp, y  
psrlq   tmp, 32  
padd   y, tmp                  ;           ......      |  S[dQ.hQH-dI.hIH]  
psrld   y, 14                   ;           ......      |    y >> 14
movdf   itmp, y
mov     sptr, [sptr_mem]  
add     itmpshort, [sptr + filtnum]   ;      ......     |   ...   | y + s
; 16 bit computation  
 mov     [sptr + filtnum], itmpshort ; 16 bit mem access  
 mov     filtnum_mem, filtnum ; store variable  
 ; for canceler adaption, store s[filtnum+3*snum] as:  
 ; 0 | s | 0 | s  
 and     itmp, 0FFFFh            ;                           0   |   s
mov     DWORD PTR curs_mem, itmp
mov     DWORD PTR curs_mem[4], itmp

;================= canceler adaption loop =======================
; Register mapping in this loop:
; Name         value                        register
;  hIH       hIH[filtnum * h_Leng]              ecx
;  hQH       hQH[filtnum * h_Leng]              edx
;  dI_       dI_[snum]                          eax  | 1st part of loop
;  dQ_       dQ_[snum]                          eax  | 2d part of loop
;  filtnum   current filter                     esi  | before loop
;  hIL       hIL[filtnum * h_Leng]              edi  | in loop
;  hQL       hQL[filtnum * h_Leng]              ebx
;  hnum      hnum                                edi
; Locals:

;------------------- LOOP SETUP ---------------------------

mov     hQL, hQL_mem ; load hQL[filtnum * h_Leng] pointer

; Postconditions:
;  Name         value                        register
;  y         result of computation             mm0
; Locals:

; compute final y, compute s
movq    tmp, y  
psrlq   tmp, 32  
padd   y, tmp                  ;           ......      |  S[dQ.hQH-dI.hIH]  
psrld   y, 14                   ;           ......      |    y >> 14
movdf   itmp, y
mov     sptr, [sptr_mem]  
add     itmpshort, [sptr + filtnum]   ;      ......     |   ...   | y + s
; 16 bit computation  
 mov     [sptr + filtnum], itmpshort ; 16 bit mem access  
 mov     filtnum_mem, filtnum ; store variable  
 ; for canceler adaption, store s[filtnum+3*snum] as:  
 ; 0 | s | 0 | s  
 and     itmp, 0FFFFh            ;                           0   |   s
mov     DWORD PTR curs_mem, itmp
mov     DWORD PTR curs_mem[4], itmp

;================= canceler adaption loop =======================
; Register mapping in this loop:
; Name         value                        register
;  hIH       hIH[filtnum * h_Leng]              ecx
;  hQH       hQH[filtnum * h_Leng]              edx
;  dI_       dI_[snum]                          eax  | 1st part of loop
;  dQ_       dQ_[snum]                          eax  | 2d part of loop
;  filtnum   current filter                     esi  | before loop
;  hIL       hIL[filtnum * h_Leng]              edi  | in loop
;  hQL       hQL[filtnum * h_Leng]              ebx
;  hnum      hnum                                edi
; Locals:

;------------------- LOOP SETUP ---------------------------

mov     hQL, hQL_mem ; load hQL[filtnum * h_Leng] pointer

; Postconditions:
;  Name         value                        register
;  y         result of computation             mm0
; Locals:
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

; trick: hIL reg = filtnum reg
; filtnum terminates its life here, hIL starts its
mov    hIL, hIL_mem ; load hIL[filtnum * h_Leng] pointer
; initialize hnum loop counter and index variable
mov    hnum, h_LengLQ_mem ; point to last quad
    ; h_LengLQ_mem = (h_Leng-4) * DATASIZE

; dQ_ TEXTEQU <eax> ; conflicts with dI_: runtime swaps w/ mem
sdI1  TEXTEQU <mm0>
sdQ1  TEXTEQU <mm0>
    ; also maps to mm0 because used only
    ; after death of sdI
sdIh  TEXTEQU <mm5>
sdQh  TEXTEQU <mm5>
    ; also maps to mm0 because used only
    ; after death of sdI
tmp   TEXTEQU <mm1>
hXH1  TEXTEQU <mm2>
hXHh  TEXTEQU <mm3>
adaptl TEXTEQU <mm4>
adapth TEXTEQU <mm7>
odd   TEXTEQU <mm6>

;---------- MACRO DEFINITION START -----------------------------
ComputAdapt MACRO  dX_:REQ, sdXl:REQ, sdXh:REQ, hXH:REQ, hXL:REQ, iter:REQ
    ; Global variables:
    ; newhXH is defined
    ;
    ; x means either i or q ___________
    ;
    ;    movq  adaptl, [hXL + hnum] ;x01; hIL3 | hIL2 | hIL1 | hIL0
    ;    movq  adapth, adaptl ;x02
    ;    movq  sdXl, [dX_ + hnum*4] ;x03; di3 | di2 | di1 | di0
    ;    movq  sdXh, sdXl ;x04
    ; adapt computation
    ;    punpcklwd adaptl, [hXH + hnum] ;x05; hIH1 | hIH0 | hIL0
    ;        = hI1 | hI0
    ;    punpckhwd adapth, [hXH + hnum] ;x06; hIH3 | hIH2 | hIL2
    ;        = hI3 | hI2
    ;    punpcklwd sdXl, sdXl ;x07; di1 | di1 | di0 | di0
    ;    pmaddwd sdXl, DWORD PTR curs_mem;x08; s.dI1 | s.dI0
    ;    psrad  sdXl, MU ;x09; s.dI1 >> MU | s.dI0 >> MU
    ;    punpckhwd sdXh, sdXh ;x10; di3 | di3 | di2 | di2
    ;    pmaddwd sdXh, DWORD PTR curs_mem;x11; s.dI3 | s.dI2
    ;    psrad  sdXh, MU ;x12; s.dI3 >> MU | s.dI2 >> MU
% IF iter EQ 1
    ; dI_
padd  adaptl, sdXl ;i13; hIH1 | hIL1 | hIH0 | hIL0 :
RESULT
padd  adapth, sdXh ;i14; hIH3 | hIL3 | hIH2 | hIL2 :
RESULT
ELSE    ; dQ_
psubd  adaptl, sdXl ;q13;
psubd  adapth, sdXh ;q14;
ENDIF

; now save the result
movq  hXH1, adaptl ;x15
psrlq hXH1, 32 ;x16; ... | ... | hIH1 | hIL1
punpcklwd adaptl, hXH1 ;x17; hIH1 | hIH0 | hIL1 | hIL0
movdf  [hXL + hnum], adaptl ;x18
psrlq adaptl, 32 ;x19; ... | ... | hIH1 | hIH0
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

movdf  [hXH + hnum], adapt1  ;x20
movq   hXHh, adaptph  ;x21
psrlq  hXHh, 32  ;x22;       ... | ... | hIH3 | hIL3
punpcklwd adaptph, hXHh  ;x23;   hIH3 | hIH2 | hIL3 | hIL2
movdf  [hXL+hnum+2*DATASIZE], adaptph  ;x24
psrlq   adaptph, 32  ;x25;       ... | ... | hIH3 | hIH2
movdf  [hXH+hnum+2*DATASIZE], adaptph  ;x26
EXITM <>
ENDM   ;; end of ComputAdapt
;--------- MACRO DEFINITION END --------------------------------

CancelerAdaptionLoop: ;-- LOOP START -----------------------------
; Preconditions:
;   Name         value                        register
;   hIH       hIH[filtnum * h_Leng]              ecx
;   hQH       hQH[filtnum * h_Leng]              edx
;   dI_       dI_[snum]                          eax
;   dQ_       undefined                         (eax)
;   filtnum   undefined                         (esi)
;   hIL       hIL[filtnum * h_Leng]              esi
;   hQL       hQL[filtnum * h_Leng]              ebx
;   hnum      new hnum value                     edi
;
; ComputAdapt(dI_, sdIl, sdIh, hIH, hIL,1)
; mov     dQ_, dQ_mem             ;i27; restore dQ_, kill dI_
; ComputAdapt(dQ_, sdQl, sdQh, hQH, hQL,2)
; mov     dI_, dI_mem             ;q27; restore dI_, kill dQ_
; CancelerAdaptionLoop termination
; sub     hnum, 4 * DATASIZE      ; decrement pointer
; je     CancelerAdaptionLoop
; CancelerAdaptionLoop -- LOOP END -----------------------------
; Postconditions:
;   Name         value                        memory location
;   hIH_base   new filter coefficients           hIH_base
;   hQH_base   new filter coefficients           hQH_base
;   hIL_base   new filter coefficients           hIL_base
;   hQL_base   new filter coefficients           hQL_base
;   add     hIH, h_LengDS_mem           ; update hIH[filtnum*h_Leng] pointer
;   add     hQH, h_LengDS_mem           ; update hQH[filtnum*h_Leng] pointer
;   add     hIL, h_LengDS_mem           ; update hIL[filtnum*h_Leng] pointer
;   add     hQL, h_LengDS_mem           ; update hQL[filtnum*h_Leng] pointer
;   mov     hIL_mem, hIL            ; store hIL, hQL values for next
;   mov     hQL_mem, hQL            ; iteration
;   FiltnumLoop termination
; mov     filtnum, filtnum_mem       ; access loop counter in memory
; add     filtnum, DATASIZE
; cmp     filtnum, TOTFILTS * DATASIZE
; jl     FiltnumLoop
; FiltnumLoop ----- LOOP END -----------------------------
"
Using MMX™ Instructions to Implement a Modem Passband Canceler
March 1996

emms
ret
PsEchoCancelerMMx EndP
END
APPENDIX C

; File: psechopt.asm
;
; DESCRIPTION:
;   This is the optimized version of the Pass Band Echo Canceler. All
;   memory accesses are aligned, loops are unrolled and instructions
;   paired. Comments show pairing constraints.
;
;**************************************************************************
;
; ASSUMPTIONS:
; ==============
;   array      size
;   + array sizes:       dI,dQ   [h_leng + s_Leng]
;   s       [3 * s_Leng]
;   hIH,hQH [3 * h_leng]
;   hIL,hQL [3 * h_leng]
;
;   + h_Leng is a multiple of 8 * DATASIZE. The arrays hIH and hQH must be
;     padded with zeroes if necessary.
;
;   + s_Leng is a multiple of 4 * DATASIZE. The arrays dI, dQ and s must be
;     padded with zeroes if necessary.
;
;   + s_Leng, h_Leng must have positive values, because loops are executed
;     at least once before these variables are checked.
;
;   + the input arrays dI, dQ, hIH, hQH, hIL, hQL are aligned on a 4 * DATASIZE
;     boundary.
;
;   + the input array s is aligned on a DATASIZE boundary.
;
;   + there is 32 bytes of space at memory (dI - 32 bytes) and (dQ - 32 bytes),
;     for optimization in CancelerAdaptionLoop.
;
;   + there is a quadword of space at memory (hIH_base - 8 bytes) for
;     optimization purposes in CancelEcho loop.
;
;   + there is a quadword of space at memory (hQH_base - 8 bytes) and at
;     hQH_base[3*h_Leng] for optimization purposes in CancelerAdaption loop.
;
;
;
; TYPE OF COMPUTATIONS
; =========
;
; All computations are integer computations.
;**************************************************************************/
;
; PSEUDO CODE OF ALGORITHM
; ================
;
; void psEchoCanceler(short *dI, short *dQ, short *s,
;                   short *hIH, short *hQH, short *hIL, short *hQL,
;                   short h_Leng, short s_Leng)
{
; long y, adapt;
; short hnum, filtnum, snum, MU=3;
;
; for (snum=0; snum<s_Leng; snum++)  // signal length
; {
    for (filtnum=0; filtnum<3; filtnum++)  // 3 filters
    {
        //-------- cancel echo ---------
        for (y=0, hnum=h_Leng-1; hnum>=0; hnum--)
        {
            y+=dI[hnum+snum]*(long)hIH[hnum+filtnum*h_Leng]
            -dQ[hnum+snum]*(long)hQH[hnum+filtnum*h_Leng];
        }
        s[filtnum+3*snum]=s[filtnum+3*snum]-(short)(y>>14);
    }

    //----- canceler adaption --------
    for (hnum=0; hnum<h_Leng; hnum++)
    {
        adapt=(((long)hIH[hnum+filtnum*h_Leng])<<16)
            |(((long)hIL[hnum+filtnum*h_Leng])&0x0000ffff);
        adapt=adapt+((s[filtnum+3*snum]*(long)dI[hnum+snum])>>MU);
        hIH[hnum+filtnum*h_Leng]=adapt>>16;
        hIL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;

        adapt=(((long)hQH[hnum+filtnum*h_Leng])<<16)
            |(((long)hQL[hnum+filtnum*h_Leng])&0x0000ffff);
        adapt=adapt-((s[filtnum+3*snum]*(long)dQ[hnum+snum])>>MU);
        hQH[hnum+filtnum*h_Leng]=adapt>>16;
        hQL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
    }
}

; PSEUDO CODE OF OPTIMIZED IMPLEMENTATION
; ============================================
; void psEchoCanceler(short *oldI, short *oldQ, short *s,
;                      short *hIH, short *hQH, short *hIL, short *hQL,
;                      short h_Leng, short s_Leng)
; {
    long y, adapt;
    short hnum, filtnum, snum, MU=3;

    ArrayCopy(oldI, dI);
    ArrayCopy(oldQ, dQ);
    for (snum=0; snum<s_Leng; snum++)  // signal length
    {
        for (filtnum=0; filtnum<3; filtnum++)  // 3 filters
        {
            //-------- cancel echo ---------
            for (y=0, hnum=h_Leng-1; hnum>=0; hnum--)
            {
                y+=dI[hnum+snum]*(long)hIH[hnum+filtnum*h_Leng]
                -dQ[hnum+snum]*(long)hQH[hnum+filtnum*h_Leng];
            }
            s[filtnum+3*snum]=s[filtnum+3*snum]-(short)(y>>14);}
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

; 
; //----- canceler adaption -------
; for (hnum=0; hnum<h_Leng; hnum++)
; {
;     adapt=((long)hIH[hnum+filtnum*h_Leng]<<16)
;         |((long)hIL[hnum+filtnum*h_Leng])&0x0000ffff);
;     adapt=adapt+((s[filtnum+3*snum]*(long)dI[hnum+snum])>>MU);
;     hIH[hnum+filtnum*h_Leng]=adapt>>16;
;     hIL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
;     adapt=((long)hQH[hnum+filtnum*h_Leng]<<16)
;         |((long)hQL[hnum+filtnum*h_Leng])&0x0000ffff);
;     adapt=adapt-((s[filtnum+3*snum]*(long)dQ[hnum+snum])>>MU);
;     hQH[hnum+filtnum*h_Leng]=adapt>>16;
;     hQL[hnum+filtnum*h_Leng]=adapt&0x0000ffff;
; }
; }

where:
ArrayCopy(oldX, dX)
{
    for (qcount = h_Leng + s_Leng -1; qcount >= 0 ; qcount--)
    {
        dX[qcount*4] = oldX[qcount];
        dX[(qcount-1)*4+1] = oldX[qcount];
        dX[(qcount-2)*4+2] = oldX[qcount];
        dX[(qcount-3)*4+3] = oldX[qcount];
    }
}
title PsEchoCancelerMMX
;****************************************************************************************
; Constants:
MU = 3 ;; adaptation step size
DATASIZE = 2 ;; in bytes. Revisit beginning of snum loop if this number
             ;; changes (hardcoded there).
TOTFILTS = 3 ;; number of filters
SLMAX = 88 ;; upper limit of: s_Leng + h_Leng
;****************************************************************************************
.model flat, c
;****************************************************************************************

.data
emptyspace1 WORD 16 DUP (02222H) ;; because the 8 words just before dQ_ array
    ; linker should align this segment to a 32 byte boundary
ALIGN 16
    ; to speed up the CancelEcho loop
dQalgn_mem WORD SLMAX*4 DUP (0AAAAH) ;; aligned version of dQ_ for all accesses
emptyspace2 WORD 16 DUP (01111H) ;; because the 8 words just before dI_ array
ALIGN 16
    ; to speed up the CancelEcho & ComputArray loops
dIalgn_mem WORD SLMAX*4 DUP (01111H) ;; aligned version of dI_ for all accesses
ALIGN 8
curs_mem QWORD (0EEEEEEEEH) ; repository for 16 bit value of current
s[filtnum+3*snum] ; duplicated for MMX code computation in canceler adaption loop. 64 bit value is:
; 0 | s | 0 | s must be quad aligned for fast access
sptr_mem DWORD (011111111H) ; s_[3 * snum] - must be quad aligned
snum_mem DWORD (022222222H) ; snum loop count value
filtnum_mem DWORD (011111111H) ; filtnum loop count value
s_LengQ_mem DWORD (022222222H); s Leng * DATASIZE * 4 to speed up loop iterations
h_LengDS_mem DWORD (0111111111H); h Leng * DATASIZE for hIH...hQL updates
h_LengLQ_mem DWORD (0111111111H); pointer to last quad:
; (h_Leng-4) * DATASIZE to speed up loop iterations
h_LengL8_mem DWORD (033333333H); pointer to last eight elements:
; (h_Leng-8) * DATASIZE to speed up loop iterations
dI_mem DWORD (022222222H) ; needed because of register conflict between
dQ_mem DWORD (011111111H) ; dI_ and dQ_ in canceler adaption
hIL_mem DWORD (022222222H) ; storage for hIL[filtnum*h_Leng]
hQL_mem DWORD (0111111111H) ; storage for hQL[filtnum*h_Leng]

PsEchoCancelerMMX PROC C uses ebx ecx edx esi edi,
dI_base:PTR WORD, dQ_base:PTR WORD, s_base:PTR WORD,
hIH_base:PTR WORD, hQH_base:PTR WORD,
hIL_base:PTR WORD, hQL_base:PTR WORD,
h_Leng:DWORD, s_Leng:DWORD

;=================== snum loop ===========================
; Locals:
snum TEXTEQU <edi> ; lives at entrance and exit of snum loop
itmp TEXTEQU <esi> ; lives when filtnum does not
itmp2 TEXTEQU <edx> ; before oldI is loaded and after CancelEcho
dI_ TEXTEQU <eax> ; lives until middle of CancelerAdaption
dQ_ TEXTEQU <ebx> ; lives until beginning of CancelerAdaption
qcount TEXTEQU <ecx> ; loop counter for ArrayCopy

;----------- LOOP SETUP -------------------------------
mov itmp, s_base ; initialize sptr_mem
mov qcount, h_Leng
mov sptr_mem, itmp
add qcount, qcount ; DATASIZE = 2 is hardcoded here
; additional loop setup instructions are merged
; with ArrayCopy loop setup below.

;============= ArrayCopy loop =======================

; creation of aligned versions of dI and dQ

; Locals:
oldI TEXTEQU <edx> ; overrides itmp2
oldQ TEXTEQU <edi> ; overrides snum
dI1 TEXTEQU <mm0>
dQ1 TEXTEQU <mm1>
dI0 TEXTEQU <mm2>
dQ0 TEXTEQU <mm3>
dI4 TEXTEQU <mm4>
dQ4 TEXTEQU <mm5>

;----------- LOOP SETUP -------------------------------
mov h_LengDS_mem, qcount ; h_LengDS_mem = h_Leng * DATASIZE
mov oldQ, dQ_base
sub qcount, 2 * DATASIZE
lea dI_, dIalign_mem
mov itmp2, qcount

; empty slot
sub itmp2, 6 * DATASIZE
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

```assembly
mov     itmp, s_Leng
mov     h_LengL8_mem, itmp2 ; h_LengL8_mem = (h_Leng-8) * DATASIZE
sub     qcount, 2* DATASIZE
add     itmp, itmp ; DATASIZE = 2 is hardcoded here
mov     h_LengLQ_mem, qcount ; h_LengLQ_mem = (h_Leng-4) * DATASIZE
add     qcount, itmp ; (h_Leng+s_Leng-4) * DATASIZE
lea     dQ_, dQalgn_mem
add     itmp, itmp
pxor    dI1, dI1
add     itmp, itmp

; empty slot
mov     oldI, dI_base
pxor    dQ1, dQ1-
movq    dI0, [oldI+qcount] ;ia03;
mov     s_LengQ_mem, itmp ; s_LengQ_mem = s_Leng * DATASIZE * 4
mov     itmp, [dI+_qcount*4] ;ia01;
        ; dI1 = dI7 | dI6 | dI5 | dI4
movdf   [dI+_qcount*4+10*DATASIZE], dI1
        ;ia02;W dI5 | dI4 | - | -
movq    dI4, dI1 ;ia05;
movq    [dI+_qcount*4], dI0 ;ia04;W dI3 | dI2 | dI1 | dI0
psllq   dI4, 48 ;ia06;
dI4      48 ;ia06; dI4 | 0 | 0 | 0

; ArrayCopy Preconditions:
; Name          value                register
; qcount        current counter value          ecx
; dI_           pointer to new, aligned dI       eax
; dQ_           pointer to new, aligned, dQ      ebx
; itmp          temporary variable              esi
; oldI          pointer to old dI               edx
; oldQ          pointer to old dQ               edi    maps over snum
; dI0           undefined                       mm2
; dQ0           undefined                       mm3
; dI1           content of quad 1 of dI values   mm0
; dQ1           content of quad 1 of dQ values   mm1
; dI4           undefined                       mm4
; dQ4           undefined                       mm5

; Notation for loop instruction numbering:
; qa03
;     ||V____ instruction number: from 01 to last instruction of loop
;     ||_____ iteration number: a for 1st iteration, b for 2d iteration
;     |______ data set: i for dI, q for dQ
;
; Additional preconditions to ArrayCopy loop:
; instructions ia01 through ia06 already executed
;
; The following comments show how instructions of psecopt.asm were reordered
; before instructions from the I-set and Q-set were merged together.
;
; I-set code                numbering in place in
; psecopt.asm   psecopt.asm
; moved to next loop iteration: (if different)
; mov     itmp, [dI+_qcount*4-16*DATASIZE] ;ib01
; movq    dI0, [oldI+_qcount*4*DATASIZE] ;ib02 ;i03
; movdf   [dI+_qcount*4-6*DATASIZE], dI1 ;ib03 ;i02
; movq    [dI+_qcount*4-16*DATASIZE], dI0 ;ib04
; movq    dI4, dI1 ;ib05
; psllq   dI4, 48 ;ib06
```
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

; I-set code in current loop
;
psrlq dI0, 16 ;ia07
por dI4, dI0 ;ia08
psrlq dI0, 16 ;ia09 ;i10
psllq dI1, 16 ;ia10 ;i13
movdf [dI_+qcount*4+8*DATASIZE], dI0 ;ia11
movq [dI_+qcount*4+4*DATASIZE], dI4 ;ia12 ;i09
psrlq dI0, 16 ;ia13 ;i12
por dI1, dI0 ;ia14
movq [dI_+qcount*4+12*DATASIZE], dI1 ;ia15
movq dI1, [oldI+qcount] ;ia16
;
;
Q-set code
;
Q-set computations in current set:
;
mov itmp, [dQ_+qcount*4] ;qa01
movdf [dQ_+qcount*4+10*DATASIZE], dQ1 ;qa02
movq dQ0, [oldQ+qcount] ;qa03
movq dQ4, dQ1 ;qa04 ;q05
movq [dQ_+qcount*4+4*DATASIZE], dQ0 ;qa05 ;q04
psllq dQ4, 48 ;qa06
psrlq dQ0, 16 ;qa07
por dQ4, dQ0 ;qa08
psrlq dQ0, 16 ;qa09 ;q10
movq [dQ_+qcount*4+4+8*DATASIZE], dQ4 ;qa10 ;q09
psllq dQ1, 16 ;qa11 ;q13
movdf [dQ_+qcount*4+8*DATASIZE], dQ0 ;qa12 ;q11
psrlq dQ0, 16 ;qa13 ;q12
por dQ1, dQ0 ;qa14
movq [dQ_+qcount*4+12*DATASIZE], dQ1 ;qa15
movq dQ1, [oldQ+qcount] ;qa16

ArrayCopy: ;---------- LOOP START -------------------------
mov itmp, [dQ_+qcount*4] ;qa01; load line in cache
psrlq dI0, 16 ;ia07; 0 | dI3 | dI2 | dI1
movdf [dQ_+qcount*4+10*DATASIZE], dQ1 ;qa02; W dQ5 | dQ4 | - | -
por dI4, dI0 ;ia08; dI4 | dI3 | dI2 | dI1
movq dQ0, [oldQ+qcount] ;qa03;
por dQ1, dI0 ;ia09;
mov itmp, [dI_+qcount*4+4-16*DATASIZE] ;ib01; load line in cache
psllq dI1, 16 ;ia10; dI6 | dI5 | dI4 | 0
movdf [dI_+qcount*4+4+8*DATASIZE], dI0 ;ia11; W - | - | dI3 | dI2
movq dQ4, dQ1 ;qa04;
movq [dI_+qcount*4+4+4*DATASIZE], dI4 ;ia12; W dI4 | dI3 | dI2 | dI1
psrlq dI0, 16 ;ia13; 0 | 0 | 0 | 0 | dI3
movq [dQ_+qcount*4], dQ0 ;qa05; W dQ3 | dQ2 | dQ1 | dQ0
por dI1, dI0 ;ia14;
psllq dQ4, 48 ;qa06; dQ4 | 0 | 0 | 0

; empty slot
movq [dI_+qcount*4+12*DATASIZE], dI1 ;ia15; W dI6 | dI5 | dI4 | dI3
psrlq dQ0, 16 ;ia16; for next iteration
por dQ4, dQ0 ;qa07; 0 | dQ3 | dQ2 | dQ1
movq dI1, [oldI+qcount] ;ia16; for next iteration
por dQ4, dQ0 ;qa08; dQ4 | dQ3 | dQ2 | dQ1
movq dI0, [oldI+qcount-4*DATASIZE] ;ib02; OK to prefetch input dI (in assumptions)
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

psrlq dQ0, 16 ;qa09;
movq [dQ_+qcount*4+4*DATASIZE], dQ4 ;qa10;W dQ4 | dQ3 | dQ2 | dQ1
psllq dQ1, 16 ;qa11; dQ6 | dQ5 | dQ4 | 0
movdf [dQ_+qcount*4+8*DATASIZE], dQ0 ;qa12;W - | - | dQ3 | dQ2
psrlq dQ0, 16 ;qa13; 0 | 0 | 0 | dQ3
; dI1 = dI7 | dI6 | dI5 | dI4
movdf [dI_+qcount*4-6*DATASIZE], dI1 ;ib03;W dI5 | dI4 | - | -
por dQ1, dQ0 ;qa14;
movq [dI_+qcount*4-16*DATASIZE], dI0 ;ib04;W dI3 | dI2 | dI1 | dI0
movq [dQ_+qcount*4+12*DATASIZE], dQ1 ;qa15;W dQ6 | dQ5 | dQ4 | dQ3
movq dI4, dI1 ;ib05;
movq dQ1, [oldQ+qcount] ;qa16; for next iteration
psllq dI4, 48 ;ib06; dI4 | 0 | 0 | 0
sub qcount, 4 * DATASIZE
jge ArrayCopy

;ArrayCopy ---------- LOOP END ------------------------------
; Postconditions:
; Name value memory location
; dIalgn_mem 4 copies of the original dI dIalgn_mem
; dQalgn_mem 4 copies of the original dQ dQalgn_mem
; initialize loop counter
mov snum, 0
SnumLoop: ;------ LOOP START ------------------------------

; Preconditions:
; Name value register
; snum new value of loop counter edi
; dI_ undefined eax
; dQ_ undefined ebx
;
; ; compute actual snum
mov itmp, hIL_base ; create hIL[filtnum*h_Leng]
lea dI_, dIalgn_mem
mov hIL_mem, itmp
lea dQ_, dQalgn_mem
mov itmp, hQL_base ; create hQL[filtnum*h_Leng]
add dI_, snum ; create dI_[snun] pointer
mov hQL_mem, itmp
add dQ_, snum ; create dQ_[snun] pointer
mov dI_mem, dI_ ; store dI_ and dQ_ for canceler
mov dQ_mem, dQ_ ; adaption: they use the same physical
; register there

;=============== filtnum loop =========================

; Locals:
filtnum TEXTEQU <esi> ; replaces itmp in this loop
hIH TEXTEQU <ecx>
hQH TEXTEQU <edx>

;---------- LOOP SETUP ------------------------------

mov snum_mem, snum ; store the counter, now can modify the register
mov hIH, hIH_base
mov hQH, hQH_base
mov filtnum, 0 ; initialize loop counter
mov filtnum_mem, filtnum ; store variable

FiltnumLoop: ;- LOOP START ------------------------------

; Preconditions:
; Name value register memory
; hIH hIH[filtnum * h_Leng] ecx
; hQH hQH[filtnum * h_Leng] edx

39
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

; filtnum loop count value - filtnum_mem
; dI_ dI_[snum] eax
; dQ_ dQ_[snum] - dQ_mem
;
; =============== cancel echo loop =================
; Locals:
  hnum TEXTEQU <edi> ;; loop counter and index
  y TEXTEQU <mm0>
  dIhIHa TEXTEQU <mm1>
  dIhIHb TEXTEQU <mm2>
  dQhQHa TEXTEQU <mm3>
  dQhQHb TEXTEQU <mm4>

;---------------- LOOP SETUP ----------------
; initialize hnum loop counter and index variable
  mov hnum, h_LengL8_mem ; point to last quad: h_LengLQ_mem = (h_Leng-4) * DATASIZE
  mov dQ_, dQ_mem ; destroyed by reg conflict with dI_ in adaptation loop

; 1 cycle penalty because of hnum usage below
  movq dIhIHa, [dI_+hnum*4+16*DATASIZE] ;1a; dI3 | dI2 | dI1 | dI0
  + dI.hIH[0]
  movq dQhQHa, [dQ_+hnum*4+16*DATASIZE] ;3a; dQ3 | dQ2 | dQ1 |
  dQ0
  pmaddwd dQhQHa, [hQH+hnum+4*DATASIZE] ;4a
  pxor y,y ; y = 0
CancelEchoLoop: |-- LOOP START --------------------
; Preconditions:
; Name value register
; hIH hIH[filtnum * h_Leng] ecx
; hQH hQH[filtnum * h_Leng] edx
; dI_ dI_[snum] eax
; dQ_ dQ_[snum] ebx
; filtnum undefined esi
; hnum current loop count value edi
; y cumulative result so far mm0
; dIhIHa computed mm1
; dQhQHa computed mm3

; Notation for loop instruction numbering:
; 8a
; |___ iteration number: a for 1st iteration, b for 2d, c for 3d
; |___ instruction number: from 1 to last instruction of loop
;
; Additional precondition to ArrayCopy loop:
; instructions 1a through 4a already executed
;
; the loop is size bound by the number of memory accesses needed to
; perform the computation.
; loop enrolling used to hide the latency of pmadd.
  movq dIhIHb, [dI_+hnum*4] ;1b; dI3 | dI2 | dI1 | dI0
  + dI.hIH[0]
  psubd dQhQHa,dIhIHa ;5a;
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

movq    dQhQHb, [dQ_ + hnum*4]          ;3b;        dQ3 |     dQ2 |     dQ1 | dQ0
padd    y, dQhQHa                       ;6a; S2;3|dQ.hQH-dI.hIH |
S0;1[dQ.hQH-di.hIH]
; vacant slot
movq    dIhIHa, [dI_+hnum*4-16*DATASIZE] ;1c;        dI3 |     dI2 |     dI1 |
; vacant slot
; vacant slot
movq    dQhQHa, [dQ_+hnum*4-16*DATASIZE];3c;        dQ3 |     dQ2 |     dQ1 |
psubd   dQhQHb,dIhIHb                   ;5b;
pmaddwd dQhQHa, [hIH+hnum-4*DATASIZE]   ;4c;
padd    y, dQhQHb                       ;6b; S2;3|dQ.hQH-dI.hIH |
S0;1[dQ.hQH-di.hIH]
sub     hnum, 8 * DATASIZE          ; decrement pointer
jge     CancelEchoLoop
; CancelEchoLoop -- LOOP END -----------------------------
; Postconditions:
; Name         value                        register
; y         result of computation             mm0
; dQ_     TEXTEQU <eax>         ; conflicts with dI_; runtime swaps w/ mem
hnum    TEXTEQU <edi>         ; loop counter and index
sdIl    TEXTEQU <mm3>
sdQl    TEXTEQU <mm1>
sdIh    TEXTEQU <mm2>
sdQh    TEXTEQU <mm0>
hXHl    TEXTEQU <mm4>
hXHh    TEXTEQU <mm5>
adaptl  TEXTEQU <mm6>
adapth  TEXTEQU <mm7>
; Locals:
sptr    TEXTEQU <eax>         ; replaces dI_
itmp   TEXTEQU <ebx>         ; replaces dQ_
itmpshort TEXTEQU <bx>          ; lower 16 bits of itmp
tmp    TEXTEQU <mm6>         ; for local computations
; compute final y, compute s
movq    tmp, y
mov     hnum, h_LengLQ_mem ; point to last quad: (h_Leng-4) * DATASIZE
mov     filtnum, filtnum_mem ; get variable
movq    sdIl, [dI_+ hnum*4] ;ia03;     di3 |     dI2 |     dI1 | dI0
psrlq  tmp, 32
mov     sptr, [sptr_mem]      ; destroys dI_
padd   y, tmp                  ; ......     S[dQ.hQH-dI.hIH]
psrld  y, 14                   ; ......     y >> 14
movq    sdIh, sdIl           ;ia04
movdf  itmp, y
punpcklwd sdIh, sdIl          ;ia07     di1 |     dI1 |     dI0 | dI0
; 2 cycle penalty due to 16 bit access below!
; add   itmpshort, [sptr + filtnum] ; ......     y + s
; empty slot
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

```
; Register mapping in this loop:
; Name     value          register
; hIH      hIH[filtnum * h_Leng]     ecx
; hQH      hQH[filtnum * h_Leng]     edx
; dI_      dI_[snum]               eax  | 1st part of loop
; dQ_      dQ_[snum]               eax  | 2d part of loop
; filtnum  filtnum                esi  | before loop
; hIL      hIL[filtnum * h_Leng]    esi  | in loop
; hQL      hQL[filtnum * h_Leng]    ebx
; hnum     hnum                    edi

; Locals:
; hIL     TEXTEQU <esi>
; hQL     TEXTEQU <ebx>         ; 2d part of loop

;----------------- LOOP SETUP -----------------
and     itmp, 0FFFFh           ;                       0   |   s
mov     hIL, hIL_mem    ; load hIL[filtnum * h_Leng] pointer
mov     DWORD PTR curs_mem, itmp
mov     DWORD PTR curs_mem[4], itmp   ;  0  |   s   |   0   |   s
; WARNING: hQL and itmp map to the same register. The lines above
; and below cannot be swapped!
pmaddwd sdIl, DWORD PTR curs_mem;ia08;          s.dI1    |   s.dI0
mov     hQL, hQL_mem    ; load hQL[filtnum * h_Leng] pointer
; instruction to disable the effect of the first q25b instruction:
movdt   adapth, [hQH+hnum+6*DATASIZE]

; empty slot
; empty cycle: sdIl computation

psrad   sdIl, MU                ;ia09;     s.dI1 >> MU   |  s.dI0 >> MU

CancelerAdaptionLoop: ;-- LOOP START --------------------------
; Preconditions:
; Name     value          register
; hIH      hIH[filtnum * h_Leng]     ecx
; hQH      hQH[filtnum * h_Leng]     edx
; dI_      dI_[snum]               eax
; dQ_      undefined              (eax)
; filtnum  undefined             (esi)
; hIL      hIL[filtnum * h_Leng]    esi
; hQL      hQL[filtnum * h_Leng]    ebx
; hnum     new hnum value         edi

; instructions ia03, ia04, ia07, ia08 & ia09 were already executed

; pairing difficulties due to too many instructions using the shifter
; original, non reordered code in file psecalin.asm, macro ComputAdapt
movdf   [hQH+hnum+6*DATASIZE], adapth ;qb26; access beyond hQH first time

; empty slot
movq    adaptl, [hIL + hnum]      ;ia01;     hIL3 | hIL2 | hIL1 | hIL0
punpckhwd sdIl, sdIh       ;ia10;     dI3 | dI3 | dI2 | dI2
pmaddwd sdIl, DWORD PTR curs_mem[4];ia11;     s.dI3 |  s.dI2
movq    adaptl, adaptl      ;ia02;
punpcklwd adaptl, [hIH + hnum] ;ia05;     hIH1 | hIL1 | hIH0 | hIL0

; empty slot
```
punpckhwd adaptl, [hIH + hnum]; ia06; hIH3 | hIL3 | hIH2 | hIL2
padd adapt1, sdIl; ia13; hIH1 | hIL1 | hIH0 | hIL0

RESULT

movq sdIl, [dI_ + hnum*4-16*DATASIZE]; ib03; dI3 | dI2 | dI1 | dI0
psrad sdIh, MU; ia12; s.dI3 >> MU | s.dI2 >> MU
movq hXHl, adapt1; ia15;
mov dQ_, dQ_mem; ia27; Now dQ_ is in and dI_ is out
padd adaptl, sdIl; ia14; hIH3 | hIL3 | hIH2 | hIL2

RESULT

psrlq hXHl, 32; ia16; ... | ... | hIH1 | hIL1
punpcklwd adaptl, hXHl; ia17; hIH1 | hIL1 | hIH0 | hIL0
movq hXHh, adaptl; ia21;
movq sdQl, [dQ_ + hnum*4]; qa03; dQ3 | dQ2 | dQ1 | dQ0
movq sdIh, sdIl; ib04;
movdf [hIL + hnum], adaptl; ia18;
psrlq hXHh, 32; ia22; ... | ... | hIH3 | hIL3
movq sdQh, sdQl; qa04;
psrlq adaptl, 32; ia19; ... | ... | hIH1 | hIL0
punpcklwd adaptl, hXHh; ia23; hIH3 | hIH2 | hIL3 | hIL2
mov dI_, dI_mem; qa27; Now dI_ is in and dQ_ is out
movdf [hIH + hnum], adaptl; ia20;
punpcklwd sdQl, sdQl; qa07; dQ1 | dQ1 | dQ0 | dQ0
movdf [hIH+hnum+2*DATASIZE], adaptl; ia24;
punpcklwd sdIl, sdIl; ib07; dI1 | dI1 | dI0 | dI0
pmaddwd sdQl, DWORD PTR curs_mem; qa08; s.dQ1 | s.dQ0
psrlq adaptl, 32; ia25; ... | ... | hIH3 | hIH2
punpcklwd adaptl, [hQH + hnum]; qa05; hQH1 | hQL1 | hQH0 | hQL0
punpckhwd adaptl, [hQH + hnum]; qa06; hQH3 | hQL3 | hQH2 | hQL2
psubd adaptl, sdQl; qa13; hQH1 | hQL1 | hQH0 | hQL0

RESULT

psrad sdQh, MU; qa12; s.dQ3 >> MU | s.dQ2 >> MU
punpcklwd adaptl, hXHl; qa17; hQH1 | hQH0 | hQL1 | hQL0
movq hXHh, adaptl; qa21;
pmaddwd sdQl, DWORD PTR curs_mem; ib08; s.dI1 | s.dI0
psrlq hXHh, 32; qa22; ... | ... | hQH3 | hQL3
movdf [hQH + hnum], adaptl; qa16;
punpcklwd adaptl, hXHh; qa23; hQH3 | hQH2 | hQL3 | hQL2
psrlq adaptl, 32; qa19; ... | ... | hQH1 | hQH0

RESULT

psrad sdQh, MU; qa12; s.dQ3 >> MU | s.dQ2 >> MU
punpcklwd adaptl, hXHl; qa17; hQH1 | hQH0 | hQL1 | hQL0
movq hXHh, adaptl; qa21;
pmaddwd sdQl, DWORD PTR curs_mem; ib08; s.dI1 | s.dI0
psrlq hXHh, 32; qa22; ... | ... | hQH3 | hQL3
movdf [hQH + hnum], adaptl; qa16;
punpcklwd adaptl, hXHh; qa23; hQH3 | hQH2 | hQL3 | hQL2
psrlq adaptl, 32; qa19; ... | ... | hQH1 | hQH0

; empty slot

movdf [hQH+hnum+2*DATASIZE], adaptl; qa24;
psrad sdIl, MU; ib09; s.dI1 >> MU | s.dI0 >> MU
movdf [hQH + hnum], adaptl; qa20;
psrlq adaptl, 32; qa25; ... | ... | hQH3 | hQH2
; CancelerAdaptionLoop termination
Using MMX™ Instructions to Implement a Modem Passband Canceler

March 1996

```
sub     hnum, 4 * DATASIZE      ; decrement pointer
jge     CancelerAdaptionLoop
; CancelerAdaptionLoop -- LOOP END -----------------------------
; Postconditions:
; Name   value                         memory location
; hIH_base new filter coefficients    hIH_base
; hQH_base new filter coefficients    hQH_base
; hIL_base new filter coefficients    hIL_base
; hQL_base new filter coefficients    hQL_base
;
filtNum TEXTEQU <edi>                 ; to avoid register conflict for pairing
; FiltnumLoop termination
    add     hIH, h_LengDS_mem           ; update hIH[filtNum*h_Leng] pointer
; empty slot
    movdf   [hQH+hnum+6*DATASIZE], adapth  
    add     hQH, h_LengDS_mem           ; update hQH[filtNum*h_Leng] pointer
    add     hIL, h_LengDS_mem           ; update hIL[filtNum*h_Leng] pointer
    add     hIL, h_LengDS_mem           ; update hIL[filtNum*h_Leng] pointer
    add     hQL, h_LengDS_mem           ; update hQL[filtNum*h_Leng] pointer
    cmp     filtnum, TOTFILTS * DATASIZE
    mov     hIL_mem, hIL                ; store hIL, hQL values for next
    mov     filtnum_mem, filtnum        ; store variable
    mov     hQL_mem, hQL                ; iteration
    jl      FiltnumLoop
; FiltnumLoop ----- LOOP END ------------------------------
    mov     snum, snum_mem              ; access loop counter in memory
    mov     itmp, sptr_mem
    add     snum, 4 * DATASIZE          ; move to next quad
    add     itmp, 3 * DATASIZE
    mov     sptr_mem, itmp              ; = s_[3*snum/4]
    cmp     snum, s_LengQ_mem
    ; SnumLoop termination
    jl      SnumLoop
; SnumLoop --------- LOOP END -------------------------------
    emms
    ret
PsEchoCancelerMMx EndP
END
```