

ストリーミング SIMD 拡張命令 2(SSE2)を使用した 逆離散コサイン変換

バージョン 2.0

2000 年 7 月

資料番号: 248670J-001

【輸出規制に関する告知と注意事項】

本資料に掲載されている製品のうち、外国為替および外国為替管理法に定める戦略物資等または役務に該当するものについては、輸出または再輸出する場合、同法に基づく日本政府の輸出許可が必要です。また、米国産品である当社製品は日本からの輸出または再輸出に際し、原則として米国政府の事前許可が必要です。

【資料内容に関する注意事項】

- ・本ドキュメントの内容を予告なしに変更することがあります。
 - ・インテルでは、この資料に掲載された内容について、市販製品に使用した場合の保証あるいは特別な目的に合うことの保証等は、いかなる場合についてもいたしかねます。また、このドキュメント内の誤りについても責任を負いかねる場合があります。
 - ・インテルでは、インテル製品の内部回路以外の使用にて責任を負いません。また、外部回路の特許についても関知いたしません。
 - ・本書の情報はインテル製品を使用できるようにする目的でのみ記載されています。
- インテルは、製品について「取引条件」で提示されている場合を除き、インテル製品の販売や使用に関して、いかなる特許または著作権の侵害をも含み、あらゆる責任を負わないものとします。
- ・いかなる形および方法によっても、インテルの文書による許可なく、この資料の一部またはすべてを複製することは禁じられています。

本資料の内容についてのお問い合わせは、下記までご連絡下さい。

インテル株式会社 資料センタ

〒305-8603 筑波学園郵便局 私書箱115号

Fax: 0120-47-8832

*一般にブランド名または商品名は各社の商標または登録商標です。

Copyright © Intel Corporation 1999, 2000

目次

1	はじめに	5
2	離散コサイン変換	5
2.1	DCT/IDCT のアプリケーション	8
2.2	IDCT アルゴリズムの実現	9
3	パフォーマンス	9
3.1	パフォーマンス向上	9
3.2	考察	10
4	結論	10
5	コード例の基礎	11
6	SSE2 を使用したアセンブリ・コード例	20
7	SSE2 IVEC を使用したコード例	30
付録 A	パフォーマンス・データ	A-1
	パフォーマンス・データの改訂履歴	A-1
	テスト・システム構成	A-3

改訂履歴

改訂	改訂履歴	Date
2.0	インテル® Pentium® 4 プロセッサに関する改訂	2000年7月
1.0	初版	1999年9月

参考資料

このアプリケーション・ノートでは次の資料を参考にした。次の資料には、ここで取り上げた事項を理解するための有用な情報が含まれている。

1. Pennebaker、Mitchell 著、『*JPEG: Still Image Data Compression Standard*』、Van Nostrand Reinhold、New York、1993年、29～64ページ
2. Rao、Yip 著、『*Discrete Cosine Transform Algorithms, Advantages, Applications*』、Academic Press, Inc.、Boston、1990年、付録 A.2
3. 『*A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX™ Instructions*』、インテル・アプリケーション・ノート、AP-922、Copyright 1999
4. 8×8の逆離散コサイン変換実現に関する IEEE 標準規格、IEEE 標準規格 1180-1990

1 はじめに

ストリーミング SIMD 拡張命令 2(SSE2、Streaming SIMD Extensions 2)では、SIMD(Single Instruction Multiple Data)倍精度浮動小数点命令、および SIMD 整数命令が IA-32 インテル® アーキテクチャに新しく導入された。倍精度 SIMD 命令による機能拡張の方法は、ストリーミング SIMD 拡張命令(SSE)で導入された単精度 SIMD 命令による機能拡張とよく似ている。128 ビットの整数 SIMD 拡張命令は、64 ビットの整数 SIMD 拡張命令の完全なスーパーセットで、より多くの整数データ型、整数と浮動小数点間のデータ型変換、キャッシュとシステム・メモリの効果的使用をサポートする命令が追加されている。これらの命令は、3D グラフィックス、リアルタイムの物理的な現象、空間的(3D)オーディオ、ビデオ・エンコーディング/デコーディング、暗号化、および科学計算アプリケーションによく使用される演算を高速化する。SSE2 テクノロジーで導入された 128 ビット整数 SIMD 拡張命令は、XMM レジスタを使用して 128 ビット・データを一度に処理できる。そのため、逆離散コサイン変換(IDCT)など重要なアルゴリズムを、SSE を使用するよりさらに高速化できる。このアプリケーション・ノートでは、SSE2 を使用して、IDCT を実行する方法を説明し、そのコード例を示す。

2 離散コサイン変換

このセクションでは、離散コサイン変換(DCT : Discrete cosine transform)を使用して、画像の空間データを周波数領域に変換する方法を説明する。セクション 2.1 では、DCT を使用して画像を圧縮する方法を説明する。セクション 2.2 では、このアプリケーション・ノートで使用した IDCT コード例について説明する。次の説明は、Pennebaker と Mitchell(参考資料[1]、29 ~ 64 ページ)から引用している。

1 次元 DCT 変換は、画像の 8 つのピクセル値を 8 つの係数に変換する。これらの係数は、8 つのリファレンス・コサイン波の振幅を表す。8 つのリファレンス・コサイン波とは、コサイン基底関数と呼ばれる直交波形である(図 1 を参照)。コサイン基底関数は、それぞれ異なる空間周波数を持つ。図 1(0)に示す一定基底関数にスケールをかける係数を、直流(DC)係数と呼び、他の 7 つの係数を交流(AC)係数と呼ぶ。

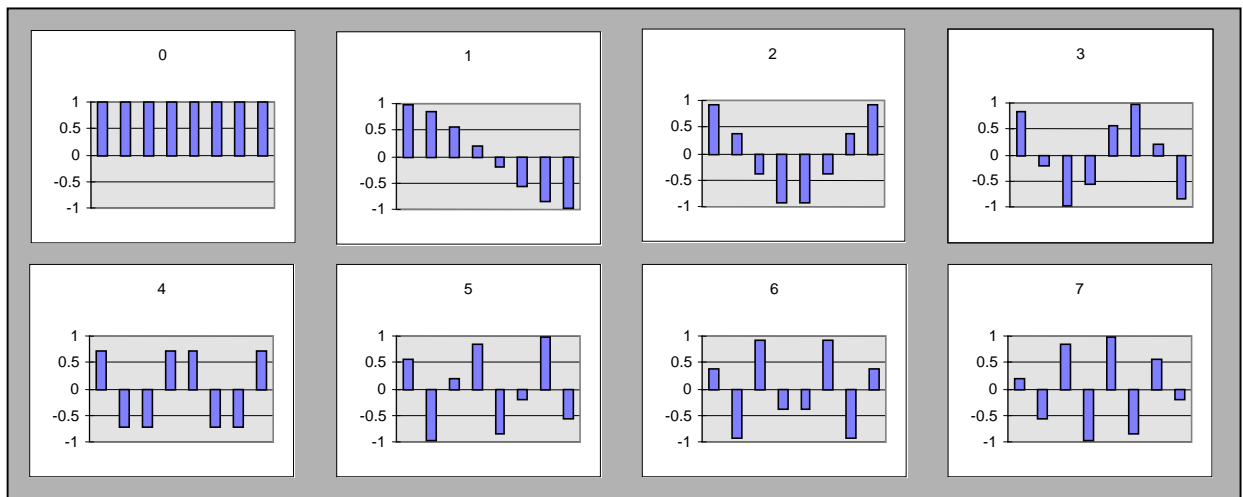


図 1: 1 次元 DCT で使用する 8 つのコサイン基底関数

2次元 DCT は、 8×8 の合計 64 ピクセル値を処理して 64 の係数を生成する。2次元 DCT では、水平方向の 8 つのコサイン基底関数と垂直方向の 8 つのコサイン基底関数を掛け合わせた 64 の基底関数を使用する。それらの基底関数を図 2 に示す。左上隅にある一定基底関数の係数を DC 係数と呼び、他の 63 の係数を AC 係数と呼ぶ。

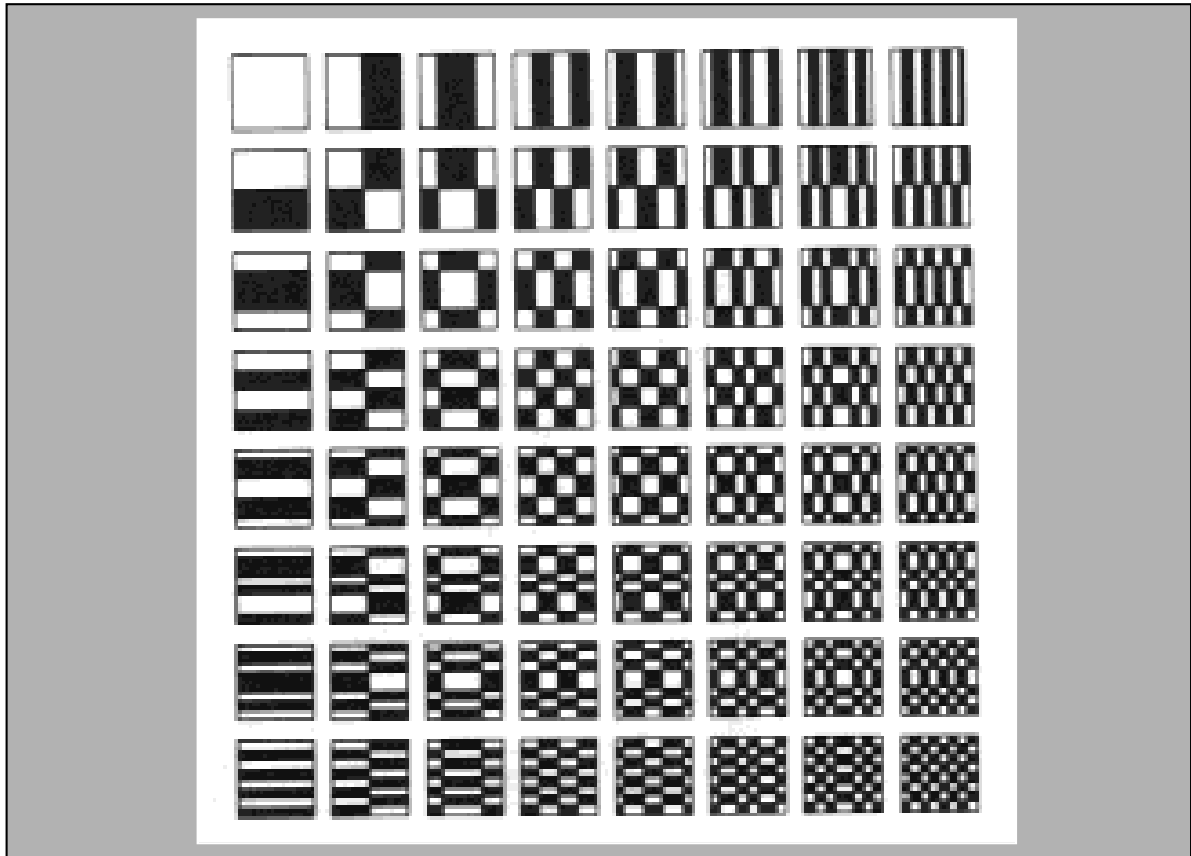


図 2: 2次元 DCT で使用する 64 の基底関数。グレーはゼロを表し、白は正の振幅、黒は負の振幅を表す。

前述したように、DCT を実行すると、画像のピクセル値が係数(コサイン基底関数の振幅)に変換される。生成された係数は、逆離散コサイン変換(IDCT)を使用して、元のピクセル値に変換できる。DCT と IDCT の計算式を次に示す。

1D DCT:

$$F(u) = \frac{1}{2} C(u) \left[\sum_{x=0}^7 f(x) * \cos \frac{(2x+1)u\pi}{16} \right] \quad (1)$$

2D DCT:

$$F(v, u) = \frac{1}{4} C(v) C(u) \left[\sum_{y=0}^7 \sum_{x=0}^7 f(y, x) * \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right] \quad (2)$$

1D IDCT:

$$f(x) = \sum_{u=0}^7 \frac{C(u)}{2} F(u) \cos \frac{(2x+1)u\pi}{16} \quad (3)$$

2D IDCT:

$$f(y, x) = \sum_{v=0}^7 \frac{C(v)}{2} \sum_{u=0}^7 \frac{C(u)}{2} F(v, u) \left[\cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right] \quad (4)$$

ここで、

$C(u)=1/\sqrt{2}$ ($u=0$ の場合)、	$C(v)=1/\sqrt{2}$ 、($v=0$ の場合)
$C(u)=1$ ($u>0$ の場合)、	$C(v)=1$ 、($v>0$ の場合)
$f(x)=1$ 次元サンプル値、	$f(y,x)=2$ 次元サンプル値
$F(u)=1$ 次元 DCT 係数、	$F(v,u)=2$ 次元 DCT 係数

上の式からわかるように、IDCTとDCTはどちらも演算回数が同じである。式(2)を使用して、2次元DCTを実行すると、1つの係数を求めるのに64回の乗算と63回の加算を行う。したがって、 8×8 のピクセル・ブロックを変換するには、4096回の乗算と4032回の加算を行うことになる。演算回数を削減するには、2次元DCTを16の1次元DCT(8行で8つの1次元DCTと、8列で8つの1次元DCT)に置き換える。式(1)を見ると、1次元DCTは64回の乗算と56回の加算で8つの係数を生成している。したがって、 8×8 のピクセル・ブロックを変換するのに、1024回の乗算と896回の加算で済む。この演算回数は上限である。より演算回数の少ないDCTとIDCTの実現方法が数多く紹介されている。DCTとIDCTの種々の実現方法については、Pennebaker-Mitchell(1993年、参考資料[1])と、Rao-Yip(1990年、参考資料[2])を参照のこと。セクション2.2では、このアプリケーション・ノートで使用する実現方法を説明する。

2.1 DCT/IDCT のアプリケーション

DCT は、MPEG または JPEG エンコーダで使用できる。すでに述べたように、DCT は画像の空間データを周波数領域の係数に変換する。係数は互いに独立しているため、どの係数も他の係数に影響することなく、個別に処理できる。人間の視覚システムは空間周波数に大きく依存しており、高周波数より低周波数の変化をより敏感に感じるため、この独立性は重要である。このため、高周波数基底関数の係数をゼロに設定しても、人間の目では知覚できない可能性がある。一般に、エンコーダは、圧縮率を高めるために、高周波数基底関数の係数をゼロにする。

図 3(a) に、可変長 JPEG エンコーダの実現例を示す。ここで、DCT を使用して画像を圧縮する方法を示す。DCT 自体は画像を圧縮するのではなく、画像から周波数領域への変換を行って圧縮しやすくする。図 3(a) に示すエンコーダでは、ラン・レンジ・エンコードおよび Huffman エンコードのステップで圧縮が行われる。JPEG 圧縮の詳細については、Pennebaker-Mitchell(1993 年、参考資料[1])を参照のこと。

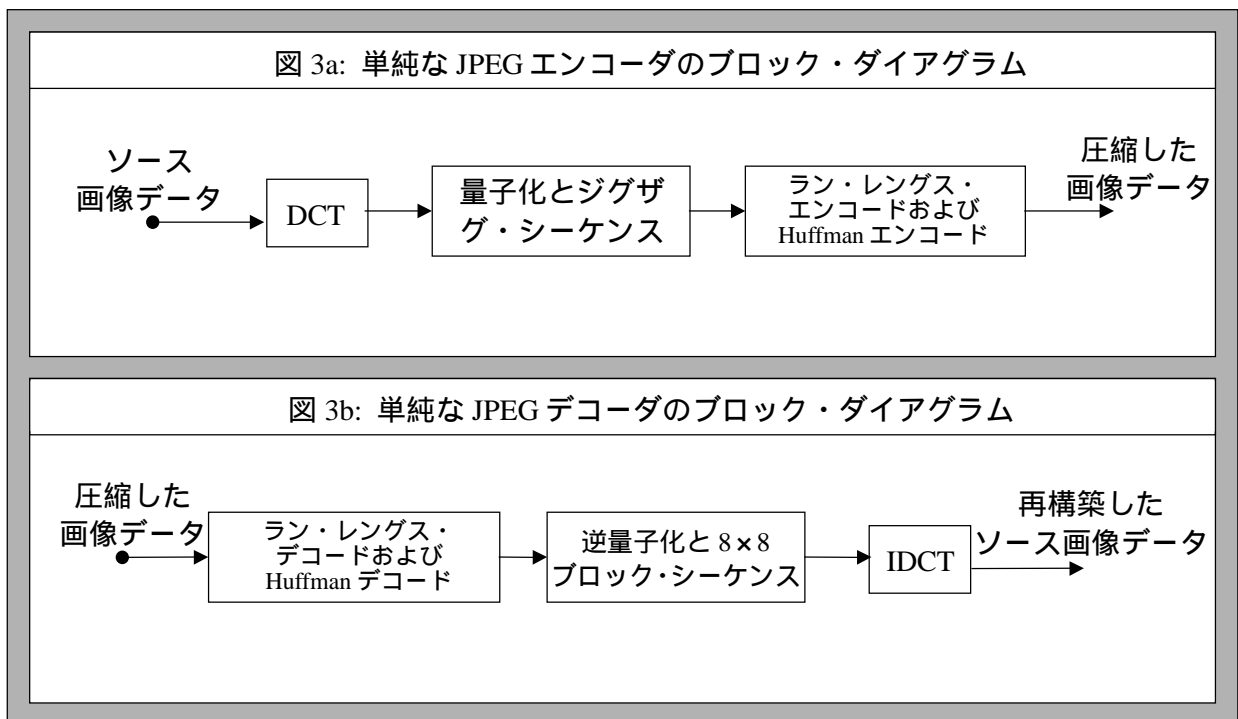


図 3: JPEG エンコーダ/デコーダの実現例

エンコーダで画像の圧縮に DCT を使用するように、デコーダでは画像の展開に IDCT を使用する(図 3(b)を参照)。アルゴリズムは異なるが、DCT/IDCT ステップはエンコーダ/デコーダの実行時間の約 20% を占める。

2.2 IDCT アルゴリズムの実現

すでに述べたように、1024回の乗算と896回の加算は上限である。さらに演算回数を減らすさまざまな方法が考え出されている。このアプリケーション・ノートでは、インテル・アプリケーション・ノート「*A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX™ Instructions*」、AP-922(参考資料[3])で紹介したIDCT実現方法を使用する。この方法の詳細は、アプリケーション・ノートAP-922を参照のこと。この方法では、320回の乗算、464回の加算、128回のシフトが必要である。この2次元IDCT実現方法を選択した理由を次に示す。

1. 行と列の両方の変換でSIMDを使用できる。SSE2を使用すると、2次元IDCTに必要な演算回数を、8回のSIMD乗算、78回のSIMD加算、32回のPMADD命令、88回のSIMD shift/shuffle命令に減らすことができる(SIMD幅は、8つのショート・データまたは8つの2バイト・データ)。
2. この実現方法では、列を処理する前に転置する必要がない。SIMDを使用する他の方法は転置を必要とするが、この方法では2つの異なる1次元IDCTを使用して転置を不要にしている。
3. このIDCT実現方法では、ポストスケール/プリスケール操作を行わない。すなわち、乗算がエンコーダ/デコーダの他のステップにまで広がらない。
4. この実現方法は、IEEE標準規格1180-1900の精度要求を満たしている。

3 パフォーマンス

このアプリケーション・ノートに示すIDCT実現方法は、以前にインテル® Pentium® IIIプロセッサでSSEを使用して最適化した(AP-922を参照)。次のセクションでは、SSE2で得られた、さらなる最適化について説明する。

3.1 パフォーマンス向上

SSE2を使用して、2次元IDCTの計算を最適化した。SSE2により、次の最適化が可能になった。

- **SIMD幅の拡張。** 整数SSE2は、64ビットのMMXテクノロジー・レジスタではなく、128ビットのXMMレジスタを使用する。SIMD幅が拡張されると、レジスタ制約が緩和し、1つの命令で同時に処理できるデータ量が2倍になった。
- **1次元IDCTにおける行の展開。** レジスタ制約が緩和されたため、行に対する1次元IDCTを展開し、2行同時に処理できるようになった。行を展開したのは、より多くの命令を並列処理するためである。特に、この展開により、pmaddwd命令をより早く他の命令と並列して処理できるようになった。
- **レジスタ制約の緩和。** レジスタ制約が緩和されたため、アセンブリ言語(ASM)のコードから8つの命令を削除できた。レジスタに値を保持しておけるので、ASMコードから2つのロード命令と2つのストア命令が削除できた。また、行に対するIDCTを展開して2行同時に処理されるので、行を適切に組み合わせ、4つの実効アドレス・ロード(lea)命令を削除できた。ASMコードから削除した8つの命令は、コメントにしてラベルを付けた。

3.2 考察

ASMコードの方が、クラス・ライブラリ(IVEC)を使用したコードよりわずかに高速だった。どちらのコードも SSE2を使用した。IVECを使用したコードでは、SIMD 演算のためのインテル® C++クラス・ライブラリ(インテル® C\C++コンパイラ・イントリンシックの C++ラッパ)を使用した点異なる。

イントリンシックとは、Cの関数呼び出し構文を使用して、通常は一對一にマッピングされた SSE2 を実行する機能である。イントリンシックまたは C++ SIMD クラスを使用する利点は、長々とアセンブリ言語でプログラミングせずに、SSE2を使用できることである。欠点は、アセンブリ言語でなければできない最適化が存在することである。例えば、ASMコードでは、2つのロード命令と2つのストア命令を削除できたが、IVECを使用したコードでは、コンパイラがレジスタの使用を制御するので、この最適化は不可能である。ASMコードの方が IVEC を使用したコードよりわずかに高速だったのは、このためである。

ただし、インライン ASMを使用すると、プロセス間およびプロセス内のコンパイラ最適化がオフになる可能性があるのに注意しなければならない。そのため、たいいていのアプリケーションでは、C++ SIMD クラスやイントリンシックを使用する方が、インライン ASMを使用するより高速になる。C++クラス・ライブラリとイントリンシックの詳細については、『*Intel C\C++ Class Libraries for SIMD Operations: With Support for the SSE2 Instructions*』資料番号 749100-001、および『*Intel C\C++ Compiler Intrinsics Reference Manual*』資料番号 748639-001を参照のこと。

4 結論

整数 SSE2を使用すると、SSEで最適化した2次元 IDCTをさらに高速化できた。SSE2は XMM レジスタを使用するため、SSE2の方が SIMD 幅が広い。SIMD 幅が拡張されてレジスタ制約が緩和されたため、ロード命令とストア命令を削減でき、行に対する1次元 IDCTを展開できた。行への IDCTを展開したことにより、Pentium 4 プロセッサのハードウェア・リソースをより有効に活用でき、より多くの命令を並列に実行できるようになった。

5 コード例の基礎

次のコードは、SSEを使用して2次元IDCTを実行する。2次元IDCTコードのまとめを以下に示す。

1. 8つの行に1次元逆DCTを実行する。 `DCT_8_INV_ROW` というマクロが、1つの行に対して1次元逆DCTを実行する。このマクロは、行の8つの値がすでに `mm0`, `mm1` の2つのMMX®テクノロジー・レジスタにロードされているものとする。また、`esi` レジスタは対応する定数乗数テーブルを指しているものとする。8行に対する1次元逆DCTを完了するには、このマクロを8回呼び出す必要がある。`DCT_8_INV_ROW` マクロの詳細は、下のコード例のコメントを参照のこと。
2. 8つの列に1次元逆DCTを実行する。 `DCT_8_INV_COL_4` というマクロが、4つの列に対して1次元逆DCTを実行する。このマクロは、第6行の4つの値が `mm0` MMXテクノロジー・レジスタにロードされているものとする。また、`edx` レジスタは処理対象の4つの列を指しているものとする。8列に対する1次元逆DCTを完了するには、このマクロを2回呼び出す必要がある。`DCT_8_INV_COL_4` マクロの詳細については、下のコード例のコメントを参照のこと。

```
#include "idct_kernel.h"

#define BITS_INV_ACC      4                      // 4 or 5 for IEEE
#define SHIFT_INV_ROW    16 - BITS_INV_ACC
#define SHIFT_INV_COL    1 + BITS_INV_ACC
const short RND_INV_ROW  = 1024 * (6 - BITS_INV_ACC); // 1 << (SHIFT_INV_ROW-1)
const short RND_INV_COL  = 16 * (BITS_INV_ACC - 3); // 1 << (SHIFT_INV_COL-1)
const short RND_INV_CORR = RND_INV_COL - 1; // correction -1.0 and round

static short one_corr[4] = {1, 1, 1, 1};
static short round_inv_row[4] = {RND_INV_ROW, 0, RND_INV_ROW, 0};
static short round_inv_col[4] = {RND_INV_COL, RND_INV_COL, RND_INV_COL, RND_INV_COL};
static short round_inv_corr[4] = {RND_INV_CORR, RND_INV_CORR, RND_INV_CORR, RND_INV_CORR};

static short tg_1_16[4] = {13036, 13036, 13036, 13036}; // tg * (2<<16) + 0.5
static short tg_2_16[4] = {27146, 27146, 27146, 27146}; // tg * (2<<16) + 0.5
static short tg_3_16[4] = {-21746, -21746, -21746, -21746}; // tg * (2<<16) + 0.5
static short cos_4_16[4] = {-19195, -19195, -19195, -19195}; // cos * (2<<16) + 0.5

//-----

// Table for rows 0,4 - constants are multiplied on cos_4_16

static short tab_i_04[] = {16384,21407,16384,8867, //movq -> w05 w04 w01 w00
                          16384, 8867, -16384, -21407, // w07 w06 w03 w02
                          16384, -8867, 16384, -21407, // w13 w12 w09 w08
                          -16384, 21407, 16384, -8867, // w15 w14 w11 w10
                          22725, 19266, 19266, -4520, // w21 w20 w17 w16
                          12873, 4520, -22725, -12873, // w23 w22 w19 w18
```

```

    12873,  -22725,  4520, -12873,    //          w29 w28 w25 w24
    4520, 19266,  19266, -22725};    //          w31 w30 w27 w26

// Table for rows 1,7 - constants are multiplied on cos_1_16

static short tab_i_17[] = {22725,  29692,  22725,  12299, //movq ->    w05 w04 w01 w00
    22725,  12299, -22725, -29692,    //          w07 w06 w03 w02
    22725,  -12299, 22725, -29692,    //          w13 w12 w09 w08
    -22725,  29692,  22725, -12299,   //          w15 w14 w11 w10
    31521,  26722,  26722,  -6270,    //          w21 w20 w17 w16
    17855,  6270,  -31521, -17855,    //          w23 w22 w19 w18
    17855,  -31521, 6270,  -17855,    //          w29 w28 w25 w24
    6270,  26722,  26722, -31521};    //          w31 w30 w27 w26

// Table for rows 2,6 - constants are multiplied on cos_2_16

static short tab_i_26[] = {21407,  27969,  21407,  11585, //movq ->    w05 w04 w01 w00
    21407,  11585, -21407, -27969,    //          w07 w06 w03 w02
    21407,  -11585, 21407, -27969,    //          w13 w12 w09 w08
    -21407, 27969,  21407, -11585,    //          w15 w14 w11 w10
    29692,  25172,  25172,  -5906,    //          w21 w20 w17 w16
    16819,  5906,  -29692, -16819,    //          w23 w22 w19 w18
    16819,  -29692, 5906,  -16819,    //          w29 w28 w25 w24
    5906,  25172,  25172, -29692};    //          w31 w30 w27 w26

// Table for rows 3,5 - constants are multiplied on cos_3_16

static short tab_i_35[] = {19266,  25172,  19266,  10426, //movq ->    w05 w04 w01 w00
    19266,  10426, -19266, -25172,    //          w07 w06 w03 w02
    19266,  -10426, 19266, -25172,    //          w13 w12 w09 w08
    -19266, 25172,  19266, -10426,    //          w15 w14 w11 w10
    26722,  22654,  22654,  -5315,    //          w21 w20 w17 w16
    15137,  5315,  -26722, -15137,    //          w23 w22 w19 w18
    15137,  -26722, 5315,  -15137,    //          w29 w28 w25 w24
    5315,  22654,  22654, -26722};    //          w31 w30 w27 w26

//-----

/*
;=====
;=====
;=====
;
; Inverse DCT
;
;-----
;
; This implementation calculates iDCT-2D by a row-column method.
; On the first stage the iDCT-1D is calculated for each row with use

```

```

; direct algorithm, on the second stage the calculation is executed
; at once for four columns with use of scaled iDCT-1D algorithm.
; Base R&Y algorithm for iDCT-1D is modified for second stage.
;
;=====
;-----
;
; The first stage - inverse DCTs of rows
;
;-----
; The 8-point inverse DCT direct algorithm
;-----
;
; static const short w[32] = {
;     FIX(cos_4_16),  FIX(cos_2_16),  FIX(cos_4_16),  FIX(cos_6_16),
;     FIX(cos_4_16),  FIX(cos_6_16), -FIX(cos_4_16), -FIX(cos_2_16),
;     FIX(cos_4_16), -FIX(cos_6_16), -FIX(cos_4_16),  FIX(cos_2_16),
;     FIX(cos_4_16), -FIX(cos_2_16),  FIX(cos_4_16), -FIX(cos_6_16),
;     FIX(cos_1_16),  FIX(cos_3_16),  FIX(cos_5_16),  FIX(cos_7_16),
;     FIX(cos_3_16), -FIX(cos_7_16), -FIX(cos_1_16), -FIX(cos_5_16),
;     FIX(cos_5_16), -FIX(cos_1_16),  FIX(cos_7_16),  FIX(cos_3_16),
;     FIX(cos_7_16), -FIX(cos_5_16),  FIX(cos_3_16), -FIX(cos_1_16) };
;
; #define DCT_8_INV_ROW(x, y)
; {
;     int a0, a1, a2, a3, b0, b1, b2, b3;
;
;     a0 = x[0] * w[ 0] + x[2] * w[ 1] + x[4] * w[ 2] + x[6] * w[ 3];
;     a1 = x[0] * w[ 4] + x[2] * w[ 5] + x[4] * w[ 6] + x[6] * w[ 7];
;     a2 = x[0] * w[ 8] + x[2] * w[ 9] + x[4] * w[10] + x[6] * w[11];
;     a3 = x[0] * w[12] + x[2] * w[13] + x[4] * w[14] + x[6] * w[15];
;     b0 = x[1] * w[16] + x[3] * w[17] + x[5] * w[18] + x[7] * w[19];
;     b1 = x[1] * w[20] + x[3] * w[21] + x[5] * w[22] + x[7] * w[23];
;     b2 = x[1] * w[24] + x[3] * w[25] + x[5] * w[26] + x[7] * w[27];
;     b3 = x[1] * w[28] + x[3] * w[29] + x[5] * w[30] + x[7] * w[31];
;
;     y[0] = SHIFT_ROUND ( a0 + b0 );
;     y[1] = SHIFT_ROUND ( a1 + b1 );
;     y[2] = SHIFT_ROUND ( a2 + b2 );
;     y[3] = SHIFT_ROUND ( a3 + b3 );
;     y[4] = SHIFT_ROUND ( a3 - b3 );
;     y[5] = SHIFT_ROUND ( a2 - b2 );
;     y[6] = SHIFT_ROUND ( a1 - b1 );
;     y[7] = SHIFT_ROUND ( a0 - b0 );
; }
;
;-----
;
; In this implementation the outputs of the iDCT-1D are multiplied

```

```

;   for rows 0,4 - on cos_4_16,
;   for rows 1,7 - on cos_1_16,
;   for rows 2,6 - on cos_2_16,
;   for rows 3,5 - on cos_3_16
; and are shifted to the left for rise of accuracy
;
; For used constants
;   FIX(float_const) = (short) (float_const * (1<<15) + 0.5)
;
;-----

;-----

;
; The second stage - inverse DCTs of columns
;
; The inputs are multiplied
;   for rows 0,4 - on cos_4_16,
;   for rows 1,7 - on cos_1_16,
;   for rows 2,6 - on cos_2_16,
;   for rows 3,5 - on cos_3_16
; and are shifted to the left for rise of accuracy
;
;-----

;
; The 8-point scaled inverse DCT algorithm (26a8m)
;
;-----

;
; #define DCT_8_INV_COL(x, y)
; {
;   short t0, t1, t2, t3, t4, t5, t6, t7;
;   short tp03, tm03, tp12, tm12, tp65, tm65;
;   short tp465, tm465, tp765, tm765;
;
;   tp765 = x[1]          + x[7] * tg_1_16;
;   tp465 = x[1] * tg_1_16 - x[7];
;   tm765 = x[5] * tg_3_16 + x[3];
;   tm465 = x[5]          - x[3] * tg_3_16;
;
;   t7   = tp765 + tm765;
;   tp65 = tp765 - tm765;
;   t4   = tp465 + tm465;
;   tm65 = tp465 - tm465;
;
;   t6   = ( tp65 + tm65 ) * cos_4_16;
;   t5   = ( tp65 - tm65 ) * cos_4_16;
;
;   tp03 = x[0] + x[4];
;   tp12 = x[0] - x[4];
;
;

```

```

;   tm03 = x[2]          + x[6] * tg_2_16;
;   tm12 = x[2] * tg_2_16 - x[6];
;
;   t0   = tp03 + tm03;
;   t3   = tp03 - tm03;
;   t1   = tp12 + tm12;
;   t2   = tp12 - tm12;
;
;   y[0] = SHIFT_ROUND ( t0 + t7 );
;   y[7] = SHIFT_ROUND ( t0 - t7 );
;   y[1] = SHIFT_ROUND ( t1 + t6 );
;   y[6] = SHIFT_ROUND ( t1 - t6 );
;   y[2] = SHIFT_ROUND ( t2 + t5 );
;   y[5] = SHIFT_ROUND ( t2 - t5 );
;   y[3] = SHIFT_ROUND ( t3 + t4 );
;   y[4] = SHIFT_ROUND ( t3 - t4 );
; }
;
;-----
*/

#define DCT_8_INV_ROW __asm{
__asm movq   mm2, mm0          /* 2      ; x3 x2 x1 x0*/
__asm movq   mm3, qword ptr [esi] /* 3      ; w05 w04 w01 w00*/
__asm pshufw mm0, mm0, 10001000b /*      ; x2 x0 x2 x0*/
__asm movq   mm4, qword ptr [esi+8] /* 4      ; w07 w06 w03 w02*/
__asm movq   mm5, mm1          /* 5      ; x7 x6 x5 x4*/
__asm pmaddwd mm3, mm0          /*      ; x2*w05+x0*w04 x2*w01+x0*w00*/
__asm movq   mm6, qword ptr [esi+32] /* 6      ; w21 w20 w17 w16*/
__asm pshufw mm1, mm1, 10001000b /*      ; x6 x4 x6 x4*/
__asm pmaddwd mm4, mm1          /*      ; x6*w07+x4*w06 x6*w03+x4*w02*/
__asm movq   mm7, qword ptr [esi+40] /* 7      ; w23 w22 w19 w18*/
__asm pshufw mm2, mm2, 11011101b /*      ; x3 x1 x3 x1*/
__asm pmaddwd mm6, mm2          /*      ; x3*w21+x1*w20 x3*w17+x1*w16*/
__asm pshufw mm5, mm5, 11011101b /*      ; x7 x5 x7 x5*/
__asm pmaddwd mm7, mm5          /*      ; x7*w23+x5*w22 x7*w19+x5*w18*/
__asm paddd  mm3, qword ptr round_inv_row /*      ; +rounder */
__asm pmaddwd mm0, qword ptr [esi+16] /*      ; x2*w13+x0*w12 x2*w09+x0*w08*/
__asm paddd  mm3, mm4          /* 4      ; a1=sum(even1) a0=sum(even0)*/
__asm pmaddwd mm1, qword ptr [esi+24] /*      ; x6*w15+x4*w14 x6*w11+x4*w10*/
__asm movq   mm4, mm3          /* 4      ; a1 a0 */
__asm pmaddwd mm2, qword ptr [esi+48] /*      ; x3*w29+x1*w28 x3*w25+x1*w24*/
__asm paddd  mm6, mm7          /* 7      ; b1=sum(odd1) b0=sum(odd0)*/
__asm pmaddwd mm5, qword ptr [esi+56] /*      ; x7*w31+x5*w30 x7*w27+x5*w26*/
__asm paddd  mm3, mm6          /*      ; a1+b1 a0+b0*/
__asm paddd  mm0, qword ptr round_inv_row /*      ; +rounder*/
__asm psrad  mm3, SHIFT_INV_ROW /*      ; y1=a1+b1 y0=a0+b0*/
__asm paddd  mm0, mm1          /* 1      ; a3=sum(even3) a2=sum(even2)*/
__asm psubd  mm4, mm6          /* 6      ; a1-b1 a0-b0 */
__asm movq   mm7, mm0          /* 7      ; a3 a2 */

```

```

__asm paddb   mm2, mm5          /* 5 ; b3=sum(odd3) b2=sum(odd2)*/ \
__asm paddb   mm0, mm2          /* a3+b3 a2+b2*/ \
__asm psrad   mm4, SHIFT_INV_ROW /* y6=a1-b1 y7=a0-b0*/ \
__asm psubd   mm7, mm2          /* 2 ; a3-b3 a2-b2*/ \
__asm psrad   mm0, SHIFT_INV_ROW /* y3=a3+b3 y2=a2+b2*/ \
__asm psrad   mm7, SHIFT_INV_ROW /* y4=a3-b3 y5=a2-b2*/ \
__asm packssdw mm3, mm0         /* 0 ; y3 y2 y1 y0*/ \
__asm packssdw mm7, mm4         /* 4 ; y6 y7 y4 y5*/ \
__asm pshufw  mm7, mm7, 10110001b /* y7 y6 y5 y4 */ \
}

#define DCT_8_INV_COL_4 __asm{ \
__asm movq    mm1, qword ptr tg_3_16 /* 1 ; tg_3_16 */ \
__asm movq    mm2, mm0              /* 2 ; x5 */ \
__asm movq    mm3, qword ptr [edx+3*16] /* 3 ; x3 */ \
__asm pmulhw  mm0, mm1              /* x5*tg_3_16 */ \
__asm movq    mm4, qword ptr [edx+7*16] /* 4 ; x7 */ \
__asm pmulhw  mm1, mm3              /* x3*tg_3_16 */ \
__asm movq    mm5, qword ptr tg_1_16 /* 5 ; tg_1_16 */ \
__asm movq    mm6, mm4              /*; 6 ; x7 */ \
__asm pmulhw  mm4, mm5              /* x7*tg_1_16 */ \
__asm paddsw  mm0, mm2              /* x5*tg_3_16 */ \
__asm pmulhw  mm5, [edx+1*16]       /* x1*tg_1_16 */ \
__asm paddsw  mm1, mm3              /* x3*tg_3_16 */ \
__asm movq    mm7, qword ptr [edx+6*16] /* 7 ; x6 */ \
__asm paddsw  mm0, mm3              /* 3 ; tm765 = x5*tg_3_16+x3 */ \
__asm movq    mm3, qword ptr tg_2_16 /* 3 ; tg_2_16 */ \
__asm psubsw  mm2, mm1              /* 1 ; tm465 = x5-x3*tg_3_16 */ \
__asm pmulhw  mm7, mm3              /* x6*tg_2_16 */ \
__asm movq    mm1, mm0              /* 1 ; tm765 */ \
__asm pmulhw  mm3, [edx+2*16]       /* x2*tg_2_16 */ \
__asm psubsw  mm5, mm6              /* 6 ; tp465 = x1*tg_1_16-x7 */ \
__asm paddsw  mm4, [edx+1*16]       /* tp765 = x1+x7*tg_1_16 */ \
__asm paddsw  mm0, mm4              /* t7 = tp765 + tm765 */ \
__asm paddsw  mm0, qword ptr one_corr /* correction t7 +1.0 */ \
__asm psubsw  mm4, mm1              /* 1 ; tp65 = tp765 - tm765 */ \
__asm paddsw  mm7, [edx+2*16]       /* tm03 = x2+x6*tg_2_16 */ \
__asm movq    mm6, mm5              /* 6 ; tp465 */ \
__asm psubsw  mm3, [edx+6*16]       /* tm12 = x2*tg_2_16-x6 */ \
__asm psubsw  mm5, mm2              /* tm65 = tp465 - tm465 */ \
__asm paddsw  mm5, qword ptr one_corr /* correction tm65 +1.0 */ \
__asm paddsw  mm6, mm2              /* 2 ; t4 = tp465 + tm465 */ \
__asm movq    [edx+7*16], mm0        /* 0 ; save t7 in y7 (tmp) */ \
__asm movq    mm1, mm4              /* 1 ; tp65 */ \
__asm movq    mm2, qword ptr cos_4_16 /* 2 ; cos_4_16 */ \
__asm paddsw  mm4, mm5              /* tp65 + tm65 */ \
__asm movq    mm0, qword ptr cos_4_16 /* 0 ; cos_4_16 */ \
__asm pmulhw  mm2, mm4              /* (tp65 + tm65)*cos_4_16 */ \
__asm movq    [edx+3*16], mm6        /* 6 ; save t4 in y3 (tmp) */ \
}

```

```

__asm psubsw mm1, mm5          /* 5          ; tp65 - tm65 */          \
__asm movq   mm6, [edx]        /* 6          ; x0 */                  \
__asm pmulhw mm0, mm1          /*          ; (tp65 - tm65)*cos_4_16 */ \
__asm movq   mm5, [edx+4*16]   /* 5          ; x4 */                  \
__asm paddsw mm4, mm2          /* 2 ; t6 = (tp65 + tm65)*cos_4_16 */ \
__asm por    mm4, qword ptr one_corr /* correction t6 +0.5 */ \
__asm paddsw mm5, mm6          /*          ; tp03 = x0 + x4 */      \
__asm psubsw mm6, [edx+4*16]   /*          ; tp12 = x0 - x4 */      \
__asm paddsw mm0, mm1          /* 1 ; t5 = (tp65 - tm65)*cos_4_16 */ \
__asm por    mm0, qword ptr one_corr /* correction t5 +0.5 */ \
__asm movq   mm2, mm5          /* 2          ; tp03 */                  \
__asm paddsw mm5, mm7          /*          ; t0 = tp03 + tm03 */      \
__asm movq   mm1, mm6          /* 1          ; tp12 */                  \
__asm paddsw mm5, qword ptr round_inv_col /* t0 + rounder */ \
__asm psubsw mm2, mm7          /* 7          ; t3 = tp03 - tm03 */      \
__asm movq   mm7, [edx+7*16]   /*          ; t7 */                  \
__asm paddsw mm6, mm3          /*          ; t1 = tp12 + tm12 */      \
__asm paddsw mm6, qword ptr round_inv_col /* t1 + rounder */ \
__asm paddsw mm7, mm5          /*          ; t0 + t7 */                  \
__asm psraw  mm7, SHIFT_INV_COL /* y0 = t0 + t7 */          \
__asm psubsw mm1, mm3          /* 3          ; t2 = tp12 - tm12 */      \
__asm paddsw mm2, qword ptr round_inv_corr /* correction t3 -1.0 +rounder */ \
__asm movq   mm3, mm6          /* 3          ; t1 */                  \
__asm paddsw mm1, qword ptr round_inv_corr /* correction t2 -1.0 +rounder */ \
__asm paddsw mm6, mm4          /*          ; t1 + t6 */                  \
__asm movq   [edx], mm7        /* 7          ; save y0 */                  \
__asm psraw  mm6, SHIFT_INV_COL /* y1 = t1 + t6 */          \
__asm movq   mm7, mm1          /* 7          ; t2 */                  \
__asm paddsw mm1, mm0          /*          ; t2 + t5 */                  \
__asm movq   [edx+1*16], mm6   /* 6          ; save y1 */                  \
__asm psraw  mm1, SHIFT_INV_COL /* y2 = t2 + t5 */          \
__asm movq   mm6, [edx+3*16]   /* 6          ; t4 */                  \
__asm psubsw mm7, mm0          /* 0          ; t2 - t5 */                  \
__asm paddsw mm6, mm2          /*          ; t3 + t4 */                  \
__asm psubsw mm2, [edx+3*16]   /*          ; t3 - t4 */                  \
__asm psraw  mm7, SHIFT_INV_COL /* y5 = t2 - t5 */          \
__asm movq   [edx+2*16], mm1   /* 1          ; save y2 */                  \
__asm psraw  mm6, SHIFT_INV_COL /* y3 = t3 + t4 */          \
__asm psubsw mm5, [edx+7*16]   /*          ; t0 - t7 */                  \
__asm psraw  mm2, SHIFT_INV_COL /* y4 = t3 - t4 */          \
__asm movq   [edx+3*16], mm6   /* 6          ; save y3 */                  \
__asm psubsw mm3, mm4          /* 4          ; t1 - t6 */                  \
__asm movq   [edx+4*16], mm2   /* 2          ; save y4 */                  \
__asm psraw  mm3, SHIFT_INV_COL /* y6 = t1 - t6 */          \
__asm movq   [edx+5*16], mm7   /* 7          ; save y5 */                  \
__asm psraw  mm5, SHIFT_INV_COL /* y7 = t0 - t7 */          \
__asm movq   [edx+6*16], mm3   /* 3          ; save y6 */                  \
__asm movq   [edx+7*16], mm5   /* 5          ; save y7 */                  \
}

```

```

int idct_P3ASM::TheCode()
{
    short* src = dctcoeffshort;
    short* dst = tst;
    __asm mov     ecx, src
    __asm mov     edx, dst
    __asm movq    mm0, [ecx]
    __asm movq    mm1, [ecx+8]
    __asm lea    esi, tab_i_04
DCT_8_INV_ROW; //Row 1, tab_i_04
    __asm movq    mm0, [ecx+16]
    __asm movq    qword ptr [edx], mm3 /* 3 /* save y3 y2 y1 y0 */
    __asm movq    mm1, [ecx+24]
    __asm movq    qword ptr [edx+8], mm7 /* 7 /* save y7 y6 y5 y4 */
    __asm lea    esi, tab_i_17
DCT_8_INV_ROW; //Row 2, tab_i_17
    __asm movq    mm0, [ecx+32]
    __asm movq    qword ptr [edx+16], mm3 /* 3 /* save y3 y2 y1 y0 */
    __asm movq    mm1, [ecx+40]
    __asm movq    qword ptr [edx+24], mm7 /* 7 /* save y7 y6 y5 y4 */
    __asm lea    esi, tab_i_26
DCT_8_INV_ROW; //Row 3, tab_i_26
    __asm movq    mm0, [ecx+48]
    __asm movq    qword ptr [edx+32], mm3 /* 3 /* save y3 y2 y1 y0 */
    __asm movq    mm1, [ecx+56]
    __asm movq    qword ptr [edx+40], mm7 /* 7 /* save y7 y6 y5 y4 */
    __asm lea    esi, tab_i_35
DCT_8_INV_ROW; //Row 4, tab_i_35
    __asm movq    mm0, [ecx+64]
    __asm movq    qword ptr [edx+48], mm3 /* 3 /* save y3 y2 y1 y0 */
    __asm movq    mm1, [ecx+72]
    __asm movq    qword ptr [edx+56], mm7 /* 7 /* save y7 y6 y5 y4 */
    __asm lea    esi, tab_i_04
DCT_8_INV_ROW; //Row 5, tab_i_04
    __asm movq    mm0, [ecx+80]
    __asm movq    qword ptr [edx+64], mm3 /* 3 /* save y3 y2 y1 y0 */
    __asm movq    mm1, [ecx+88]
    __asm movq    qword ptr [edx+72], mm7 /* 7 /* save y7 y6 y5 y4 */
    __asm lea    esi, tab_i_35
DCT_8_INV_ROW; //Row 6, tab_i_35
    __asm movq    mm0, [ecx+96]
    __asm movq    qword ptr [edx+80], mm3 /* 3 /* save y3 y2 y1 y0 */
    __asm movq    mm1, [ecx+104]
    __asm movq    qword ptr [edx+88], mm7 /* 7 /* save y7 y6 y5 y4 */
    __asm lea    esi, tab_i_26
DCT_8_INV_ROW; //Row 7, tab_i_26
    __asm movq    mm0, [ecx+112]
    __asm movq    qword ptr [edx+96], mm3 /* 3 /* save y3 y2 y1 y0 */
    __asm movq    mm1, [ecx+120]

```

```
__asm movq    qword ptr [edx+104],mm7 /* 7          /* save y7 y6 y5 y4 */
__asm lea    esi, tab_i_17
DCT_8_INV_ROW; //Row 8, tab_i_17
__asm movq    qword ptr [edx+112],mm3 /* 3          /* save y3 y2 y1 y0 */
__asm movq    mm0, qword ptr [edx+80] /* 0          /* x5 */
__asm movq    qword ptr [edx+120],mm7 /* 7          /* save y7 y6 y5 y4 */

DCT_8_INV_COL_4
__asm movq    mm0, qword ptr [edx+88] /* 0          /* x5 */
__asm add    edx, 8
DCT_8_INV_COL_4
//__asm emms
return(EXIT_SUCCESS);
}
```

6 SSE2 を使用したアセンブリ・コード例

このセクションとセクション7に示すコードは、SSE2 を使用して 2 次元 IDCT を実行する。この 2 次元 IDCT コードのまとめを次に示す。

1. 8 つの行に 1 次元逆 DCT を実行する。 `DCT_8_INV_ROW` というマクロを使用して、2 つの行に対して 1 次元逆 DCT を実行する。このマクロは、2 つの行がすでに `xmm0` および `xmm4` レジスタにロードされているものとする。また、`esi` および `ecx` レジスタが、対応する定数乗数テーブルを指しているものとする。8 行に対する 1 次元逆 DCT を完了するには、このマクロを 4 回呼び出す必要がある。`DCT_8_INV_ROW` マクロの詳細については、以下のコード例のコメントを参照のこと。
2. 8 つの列に 1 次元逆 DCT を実行する。 `DCT_8_INV_COL_8` というマクロを使用して、8 つの列に対して 1 次元逆 DCT を実行する。このマクロは、第 6 行の値が `xmm0` レジスタにロードされているものとする。また、`edx` レジスタは処理対象の 8 つの列を指しているものとする。8 行に対する 1 次元逆 DCT を完了するには、このマクロを 1 回呼び出す必要がある。`DCT_8_INV_COL_8` マクロの詳細については、以下のコード例のコメントを参照のこと。

```
#include <dvec.h>
#include "idct_kernel.h"

#define BITS_INV_ACC      4                      // 4 or 5 for IEEE
#define SHIFT_INV_ROW    16 - BITS_INV_ACC
#define SHIFT_INV_COL    1 + BITS_INV_ACC
const short RND_INV_ROW  = 1024 * (6 - BITS_INV_ACC); // 1 << (SHIFT_INV_ROW-1)
const short RND_INV_COL  = 16 * (BITS_INV_ACC - 3);  // 1 << (SHIFT_INV_COL-1)
const short RND_INV_CORR = RND_INV_COL - 1;         // correction -1.0 and round

__declspec(align(16)) short M128_one_corr[8] = {1,1,      1,1,  1,  1,  1,  1,
1};
__declspec(align(16)) short M128_round_inv_row[8] = {RND_INV_ROW, 0, RND_INV_ROW, 0,
RND_INV_ROW, 0, RND_INV_ROW, 0};

__declspec(align(16)) short M128_round_inv_col[8] = {RND_INV_COL, RND_INV_COL,
RND_INV_COL, RND_INV_COL, RND_INV_COL, RND_INV_COL, RND_INV_COL, RND_INV_COL};
__declspec(align(16)) short M128_round_inv_corr[8] = {RND_INV_CORR, RND_INV_CORR,
RND_INV_CORR, RND_INV_CORR, RND_INV_CORR, RND_INV_CORR, RND_INV_CORR, RND_INV_CORR};
__declspec(align(16)) short M128_tg_1_16[8] = {13036, 13036, 13036, 13036, 13036,
13036, 13036, 13036}; // tg * (2<<16) + 0.5
__declspec(align(16)) short M128_tg_2_16[8] = {27146, 27146, 27146, 27146, 27146,
27146, 27146, 27146}; // tg * (2<<16) + 0.5
__declspec(align(16)) short M128_tg_3_16[8] = {-21746, -21746, -21746, -21746,
-21746, -21746, -21746, -21746}; // tg * (2<<16) + 0.5
__declspec(align(16)) short M128_cos_4_16[8] = {-19195, -19195, -19195, -19195,
-19195, -19195, -19195, -19195}; // cos * (2<<16) + 0.5
```

```

//-----
// Table for rows 0,4 - constants are multiplied on cos_4_16

//movq -> w13 w12 w09 w08 w05 w04 w01 w00
//      w15 w14 w11 w10 w07 w06 w03 w02
//      w29 w28 w25 w24 w21 w20 w17 w16
//      w31 w30 w27 w26 w23 w22 w19 w18

//movq -> w05 w04 w01 w00
__declspec(align(16)) short M128_tab_i_04[] = {16384, 21407, 16384, 8867,
        16384, -8867, 16384, -21407, // w13 w12 w09 w08
        16384, 8867, -16384, -21407, // w07 w06 w03 w02
        -16384, 21407, 16384, -8867, // w15 w14 w11 w10
        22725, 19266, 19266, -4520, // w21 w20 w17 w16
        12873, -22725, 4520, -12873, // w29 w28 w25 w24
        12873, 4520, -22725, -12873, // w23 w22 w19 w18
        4520, 19266, 19266, -22725}; // w31 w30 w27 w26

// Table for rows 1,7 - constants are multiplied on cos_1_16

//movq -> w05 w04 w01 w00
__declspec(align(16)) short M128_tab_i_17[] = {22725, 29692, 22725, 12299,
        22725, -12299, 22725, -29692, // w13 w12 w09 w08
        22725, 12299, -22725, -29692, // w07 w06 w03 w02
        -22725, 29692, 22725, -12299, // w15 w14 w11 w10
        31521, 26722, 26722, -6270, // w21 w20 w17 w16
        17855, -31521, 6270, -17855, // w29 w28 w25 w24
        17855, 6270, -31521, -17855, // w23 w22 w19 w18
        6270, 26722, 26722, -31521}; // w31 w30 w27 w26

// Table for rows 2,6 - constants are multiplied on cos_2_16

//movq -> w05 w04 w01 w00
__declspec(align(16)) short M128_tab_i_26[] = {21407, 27969, 21407, 11585,
        21407, -11585, 21407, -27969, // w13 w12 w09 w08
        21407, 11585, -21407, -27969, // w07 w06 w03 w02
        -21407, 27969, 21407, -11585, // w15 w14 w11 w10
        29692, 25172, 25172, -5906, // w21 w20 w17 w16
        16819, -29692, 5906, -16819, // w29 w28 w25 w24
        16819, 5906, -29692, -16819, // w23 w22 w19 w18
        5906, 25172, 25172, -29692}; // w31 w30 w27 w26

// Table for rows 3,5 - constants are multiplied on cos_3_16

//movq -> w05 w04 w01 w00
__declspec(align(16)) short M128_tab_i_35[] = {19266, 25172, 19266, 10426,
        19266, -10426, 19266, -25172, // w13 w12 w09 w08
        19266, 10426, -19266, -25172, // w07 w06 w03 w02

```

```

        -19266, 25172, 19266, -10426, //          w15 w14 w11 w10
        26722, 22654, 22654, -5315, //          w21 w20 w17 w16
        15137, -26722, 5315, -15137, //         w29 w28 w25 w24
        15137, 5315, -26722, -15137, //         w23 w22 w19 w18
        5315, 22654, 22654, -26722}; //         w31 w30 w27 w26

//-----

/*
;=====
;=====
;=====
;
; Inverse DCT
;
;-----
;
; This implementation calculates iDCT-2D by a row-column method.
; On the first stage the iDCT-1D is calculated for each row with use
; direct algorithm, on the second stage the calculation is executed
; at once for four columns with use of scaled iDCT-1D algorithm.
; Base R&Y algorithm for iDCT-1D is modified for second stage.
;
;=====

;-----
;
; The first stage - inverse DCTs of rows
;
;-----
; The 8-point inverse DCT direct algorithm
;-----
;
; static const short w[32] = {
;     FIX(cos_4_16),  FIX(cos_2_16),  FIX(cos_4_16),  FIX(cos_6_16),
;     FIX(cos_4_16),  FIX(cos_6_16), -FIX(cos_4_16), -FIX(cos_2_16),
;     FIX(cos_4_16), -FIX(cos_6_16), -FIX(cos_4_16),  FIX(cos_2_16),
;     FIX(cos_4_16), -FIX(cos_2_16),  FIX(cos_4_16), -FIX(cos_6_16),
;     FIX(cos_1_16),  FIX(cos_3_16),  FIX(cos_5_16),  FIX(cos_7_16),
;     FIX(cos_3_16), -FIX(cos_7_16), -FIX(cos_1_16), -FIX(cos_5_16),
;     FIX(cos_5_16), -FIX(cos_1_16),  FIX(cos_7_16),  FIX(cos_3_16),
;     FIX(cos_7_16), -FIX(cos_5_16),  FIX(cos_3_16), -FIX(cos_1_16) };
;
; #define DCT_8_INV_ROW(x, y)
; {

```

```

;   int a0, a1, a2, a3, b0, b1, b2, b3;
;
;   a0  = x[0] * w[ 0] + x[2] * w[ 1] + x[4] * w[ 2] + x[6] * w[ 3];
;   a1  = x[0] * w[ 4] + x[2] * w[ 5] + x[4] * w[ 6] + x[6] * w[ 7];
;   a2  = x[0] * w[ 8] + x[2] * w[ 9] + x[4] * w[10] + x[6] * w[11];
;   a3  = x[0] * w[12] + x[2] * w[13] + x[4] * w[14] + x[6] * w[15];
;   b0  = x[1] * w[16] + x[3] * w[17] + x[5] * w[18] + x[7] * w[19];
;   b1  = x[1] * w[20] + x[3] * w[21] + x[5] * w[22] + x[7] * w[23];
;   b2  = x[1] * w[24] + x[3] * w[25] + x[5] * w[26] + x[7] * w[27];
;   b3  = x[1] * w[28] + x[3] * w[29] + x[5] * w[30] + x[7] * w[31];
;
;   y[0] = SHIFT_ROUND ( a0 + b0 );
;   y[1] = SHIFT_ROUND ( a1 + b1 );
;   y[2] = SHIFT_ROUND ( a2 + b2 );
;   y[3] = SHIFT_ROUND ( a3 + b3 );
;   y[4] = SHIFT_ROUND ( a3 - b3 );
;   y[5] = SHIFT_ROUND ( a2 - b2 );
;   y[6] = SHIFT_ROUND ( a1 - b1 );
;   y[7] = SHIFT_ROUND ( a0 - b0 );
; }
;
;-----
;
; In this implementation the outputs of the iDCT-1D are multiplied
;   for rows 0,4 - on cos_4_16,
;   for rows 1,7 - on cos_1_16,
;   for rows 2,6 - on cos_2_16,
;   for rows 3,5 - on cos_3_16
; and are shifted to the left for rise of accuracy
;
; For used constants
;   FIX(float_const) = (short) (float_const * (1<<15) + 0.5)
;
;-----
;-----
;
; The second stage - inverse DCTs of columns
;
; The inputs are multiplied
;   for rows 0,4 - on cos_4_16,
;   for rows 1,7 - on cos_1_16,
;   for rows 2,6 - on cos_2_16,
;   for rows 3,5 - on cos_3_16
; and are shifted to the left for rise of accuracy
;

```

```

;-----
;
; The 8-point scaled inverse DCT algorithm (26a8m)
;
;-----
;
; #define DCT_8_INV_COL(x, y)
; {
;   short t0, t1, t2, t3, t4, t5, t6, t7;
;   short tp03, tm03, tp12, tm12, tp65, tm65;
;   short tp465, tm465, tp765, tm765;
;
;   tp765 = x[1]          + x[7] * tg_1_16;
;   tp465 = x[1] * tg_1_16 - x[7];
;   tm765 = x[5] * tg_3_16 + x[3];
;   tm465 = x[5]          - x[3] * tg_3_16;
;
;   t7    = tp765 + tm765;
;   tp65   = tp765 - tm765;
;   t4    = tp465 + tm465;
;   tm65   = tp465 - tm465;
;
;   t6    = ( tp65 + tm65 ) * cos_4_16;
;   t5    = ( tp65 - tm65 ) * cos_4_16;
;
;   tp03  = x[0] + x[4];
;   tp12  = x[0] - x[4];
;
;   tm03  = x[2]          + x[6] * tg_2_16;
;   tm12  = x[2] * tg_2_16 - x[6];
;
;   t0    = tp03 + tm03;
;   t3    = tp03 - tm03;
;   t1    = tp12 + tm12;
;   t2    = tp12 - tm12;
;
;   y[0]  = SHIFT_ROUND ( t0 + t7 );
;   y[7]  = SHIFT_ROUND ( t0 - t7 );
;   y[1]  = SHIFT_ROUND ( t1 + t6 );
;   y[6]  = SHIFT_ROUND ( t1 - t6 );
;   y[2]  = SHIFT_ROUND ( t2 + t5 );
;   y[5]  = SHIFT_ROUND ( t2 - t5 );
;   y[3]  = SHIFT_ROUND ( t3 + t4 );
;   y[4]  = SHIFT_ROUND ( t3 - t4 );
; }
;

```

```

;-----
*/
//xmm7 = round_inv_row
#define DCT_8_INV_ROW__asm{
    __asmpshuflw    xmm0, xmm0, 0xD8    \
    __asmpshufd    xmm1, xmm0, 0       \
    __asmpmaddwd   xmm1, [esi]         \
    __asmpshufd    xmm3, xmm0, 0x55    \
    __asmpshufhw   xmm0, xmm0, 0xD8    \
    __asmpmaddwd   xmm3, [esi+32]      \
    __asmpshufd    xmm2, xmm0, 0xAA    \
    __asmpshufd    xmm0, xmm0, 0xFF    \
    __asmpmaddwd   xmm2, [esi+16]      \
    __asmpshufhw   xmm4, xmm4, 0xD8    \
    __asmpadd      xmm1, M128_round_inv_row \
    __asmpshuflw   xmm4, xmm4, 0xD8    \
    __asmpmaddwd   xmm0, [esi+48]      \
    __asmpshufd    xmm5, xmm4, 0       \
    __asmpshufd    xmm6, xmm4, 0xAA    \
    __asmpmaddwd   xmm5, [ecx]         \
    __asmpadd      xmm1, xmm2          \
    __asm movdqa   xmm2, xmm1          \
    __asmpshufd    xmm7, xmm4, 0x55    \
    __asmpmaddwd   xmm6, [ecx+16]      \
    __asmpadd      xmm0, xmm3          \
    __asmpshufd    xmm4, xmm4, 0xFF    \
    __asmpsubd     xmm2, xmm0          \
    __asmpmaddwd   xmm7, [ecx+32]      \
    __asmpadd      xmm0, xmm1          \
    __asmpsradd    xmm2, 12            \
    __asmpadd      xmm5, M128_round_inv_row \
    __asmpmaddwd   xmm4, [ecx+48]      \
    __asmpadd      xmm5, xmm6          \
    __asm movdqa   xmm6, xmm5          \
    __asmpsradd    xmm0, 12            \
    __asmpshufd    xmm2, xmm2, 0x1B    \
    __asmpackssdw  xmm0, xmm2          \
    __asmpadd      xmm4, xmm7          \
    __asmpsubd     xmm6, xmm4          \
    __asmpadd      xmm4, xmm5          \
    __asmpsradd    xmm6, 12            \
    __asmpsradd    xmm4, 12            \
    __asmpshufd    xmm6, xmm6, 0x1B    \
    __asmpackssdw  xmm4, xmm6          \
}

```

```

#define DCT_8_INV_COL_8 __asm{ \
    __asm movdqa    xmm1, XMMWORD PTR M128_tg_3_16 \
    __asm movdqa    xmm2, xmm0 \
    __asm movdqa    xmm3, XMMWORD PTR [edx+3*16] \
    __asm pmulhw    xmm0, xmm1 \
    __asm pmulhw    xmm1, xmm3 \
    __asm movdqa    xmm5, XMMWORD PTR M128_tg_1_16 \
    __asm movdqa    xmm6, xmm4 \
    __asm pmulhw    xmm4, xmm5 \
    __asm paddsw    xmm0, xmm2 \
    __asm pmulhw    xmm5, [edx+1*16] \
    __asm paddsw    xmm1, xmm3 \
    __asm movdqa    xmm7, XMMWORD PTR [edx+6*16] \
    __asm paddsw    xmm0, xmm3 \
    __asm movdqa    xmm3, XMMWORD PTR M128_tg_2_16 \
    __asm psubsw    xmm2, xmm1 \
    __asm pmulhw    xmm7, xmm3 \
    __asm movdqa    xmm1, xmm0 \
    __asm pmulhw    xmm3, [edx+2*16] \
    __asm psubsw    xmm5, xmm6 \
    __asm paddsw    xmm4, [edx+1*16] \
    __asm paddsw    xmm0, xmm4 \
    __asm paddsw    xmm0, XMMWORD PTR M128_one_corr \
    __asm psubsw    xmm4, xmm1 \
    __asm movdqa    xmm6, xmm5 \
    __asm psubsw    xmm5, xmm2 \
    __asm paddsw    xmm5, XMMWORD PTR M128_one_corr \
    __asm paddsw    xmm6, xmm2 \
    __asm movdqa    [edx+7*16], xmm0 \
    __asm movdqa    xmm1, xmm4 \
    __asm movdqa    xmm0, XMMWORD PTR M128_cos_4_16 \
    __asm paddsw    xmm4, xmm5 \
    __asm movdqa    xmm2, XMMWORD PTR M128_cos_4_16 \
    __asm pmulhw    xmm2, xmm4 \
    __asm movdqa    [edx+3*16], xmm6 \
    __asm psubsw    xmm1, xmm5 \
    __asm paddsw    xmm7, [edx+2*16] \
    __asm psubsw    xmm3, [edx+6*16] \
    __asm movdqa    xmm6, [edx] \
    __asm pmulhw    xmm0, xmm1 \
    __asm movdqa    xmm5, [edx+4*16] \
    __asm paddsw    xmm5, xmm6 \
    __asm psubsw    xmm6, [edx+4*16] \
    __asm paddsw    xmm4, xmm2 \
    __asm por      xmm4, XMMWORD PTR M128_one_corr \

```

```

__asm paddsw    xmm0, xmm1                \
__asm por      xmm0, XMMWORD PTR M128_one_corr \
__asm movdqa   xmm2, xmm5                \
__asm paddsw   xmm5, xmm7                \
__asm movdqa   xmm1, xmm6                \
__asm paddsw   xmm5, XMMWORD PTR M128_round_inv_col \
__asm psubsw   xmm2, xmm7                \
__asm movdqa   xmm7, [edx+7*16]          \
__asm paddsw   xmm6, xmm3                \
__asm paddsw   xmm6, XMMWORD PTR M128_round_inv_col \
__asm paddsw   xmm7, xmm5                \
__asm psraw    xmm7, SHIFT_INV_COL      \
__asm psubsw   xmm1, xmm3                \
__asm paddsw   xmm1, XMMWORD PTR M128_round_inv_corr \
__asm movdqa   xmm3, xmm6                \
__asm paddsw   xmm2, XMMWORD PTR M128_round_inv_corr \
__asm paddsw   xmm6, xmm4                \
__asm movdqa   [edx], xmm7               \
__asm psraw    xmm6, SHIFT_INV_COL      \
__asm movdqa   xmm7, xmm1                \
__asm paddsw   xmm1, xmm0                \
__asm movdqa   [edx+1*16], xmm6          \
__asm psraw    xmm1, SHIFT_INV_COL      \
__asm movdqa   xmm6, [edx+3*16]          \
__asm psubsw   xmm7, xmm0                \
__asm psraw    xmm7, SHIFT_INV_COL      \
__asm movdqa   [edx+2*16], xmm1          \
__asm psubsw   xmm5, [edx+7*16]          \
__asm psraw    xmm5, SHIFT_INV_COL      \
__asm movdqa   [edx+7*16], xmm5          \
__asm psubsw   xmm3, xmm4                \
__asm paddsw   xmm6, xmm2                \
__asm psubsw   xmm2, [edx+3*16]          \
__asm psraw    xmm6, SHIFT_INV_COL      \
__asm psraw    xmm2, SHIFT_INV_COL      \
__asm movdqa   [edx+3*16], xmm6          \
__asm psraw    xmm3, SHIFT_INV_COL      \
__asm movdqa   [edx+4*16], xmm2          \
__asm movdqa   [edx+5*16], xmm7          \
__asm movdqa   [edx+6*16], xmm3          \
}

```

```
//assumes src and destination are aligned on a 16-byte boundary
```

```

int idct_M128ASM::TheCode()
{
// assert(((src & 0xf) == 0)&&((dst & 0xf) == 0)) //aligned on 16-byte boundary
short* src = dctcoeffshort;
short* dst = tst;
__asm mov     eax, src
__asm mov     edx, dst
//.....//
__asm movdqa  xmm0, XMMWORD PTR[eax] //row 1
__asm lea    esi, M128_tab_i_04
__asm movdqa  xmm4, XMMWORD PTR[eax+16*2] //row 3

__asm lea    ecx, M128_tab_i_26
DCT_8_INV_ROW; //Row 1, tab_i_04 and Row 3, tab_i_26
__asm movdqa  XMMWORD PTR[edx],    xmm0
__asm movdqa  XMMWORD PTR[edx+16*2], xmm4
//.....//
__asm movdqa  xmm0, XMMWORD PTR[eax+16*4] //row 5
//__asm lea    esi, M128_tab_i_04
__asm movdqa  xmm4, XMMWORD PTR[eax+16*6] //row 7

//__asm lea    ecx, M128_tab_i_26
DCT_8_INV_ROW; //Row 5, tab_i_04 and Row 7, tab_i_26
__asm movdqa  XMMWORD PTR[edx+16*4], xmm0
__asm movdqa  XMMWORD PTR[edx+16*6], xmm4
//.....//
__asm movdqa  xmm0, XMMWORD PTR[eax+16*3] //row 4
__asm lea    esi, M128_tab_i_35
__asm movdqa  xmm4, XMMWORD PTR[eax+16*1] //row 2

__asm lea    ecx, M128_tab_i_17
DCT_8_INV_ROW; //Row 4, tab_i_35 and Row 2, tab_i_17
__asm movdqa  XMMWORD PTR[edx+16*3], xmm0
__asm movdqa  XMMWORD PTR[edx+16*1], xmm4
//.....//
__asm movdqa  xmm0, XMMWORD PTR[eax+16*5] //row 6
//__asm lea    esi, M128_tab_i_35
__asm movdqa  xmm4, XMMWORD PTR[eax+16*7] //row 8

//__asm lea    ecx, M128_tab_i_17
DCT_8_INV_ROW; //Row 6, tab_i_35 and Row 8, tab_i_17
//__asm movdqa  XMMWORD PTR[edx+80],    xmm0
//__asm movdqa  xmm0, XMMWORD PTR [edx+80] /* 0          /* x5 */
//__asm movdqa  XMMWORD PTR[edx+16*7],    xmm4
//__asm movdqa  xmm4, XMMWORD PTR [edx+7*16]/* 4          ; x7 */

DCT_8_INV_COL_8
// __asm emms

```

```
    return(EXIT_SUCCESS);  
  
}
```

7 SSE2 IVEC を使用したコード例

このセクションに示すコードは、SIMD クラスで実現した SSE2 を使用して 2 次元 IDCT を実行する。この 2 次元 IDCT コードのまとめを次に示す。

1. 8 つの行に 1 次元逆 DCT を実行する。 DCT_8_INV_ROW というマクロを使用して、2 つの行に対して 1 次元逆 DCT を実行する。このマクロは、2 つの行がすでに row および r2ow という Is16vec8 変数にロードされているものとする。また、table1 および table2 という Is16vec8 ポインタが、対応する定数乗数テーブルを指しているものとする。8 行に対する 1 次元逆 DCT を完了するには、このマクロを 4 回呼び出す必要がある。DCT_8_INV_ROW マクロの詳細については、下のコメントを参照のこと。
2. 8 つの列に 1 次元逆 DCT を実行する。 このコードでは、8 つの列に対して 1 次元逆 DCT を実行する DCT_8_INV_COL_8 というマクロを使用しない。かわりに 1 次元逆 DCT を実行するコードを関数にインクルードする。1 次元逆 DCT を実行するコードの詳細については、下のコメントを参照のこと。

```
#include <dvec.h>
#include "idct_kernel.h"

#define BITS_INV_ACC      4                      // 4 or 5 for IEEE
#define SHIFT_INV_ROW    16 - BITS_INV_ACC
#define SHIFT_INV_COL    1 + BITS_INV_ACC
const short RND_INV_ROW  = 1024 * (6 - BITS_INV_ACC);    //1 << (SHIFT_INV_ROW-1)
const short RND_INV_COL  = 16 * (BITS_INV_ACC - 3);      // 1 << (SHIFT_INV_COL-1)
const short RND_INV_CORR = RND_INV_COL - 1;             // correction -1.0 and round

Is16vec8 ivec_one_corr(1, 1, 1, 1, 1, 1, 1, 1);
Is16vec8 ivec_round_inv_row(0,RND_INV_ROW,0,RND_INV_ROW,0,RND_INV_ROW,0,
RND_INV_ROW);
Is16vec8 ivec_round_inv_col(RND_INV_COL, RND_INV_COL, RND_INV_COL, RND_INV_COL,
RND_INV_COL, RND_INV_COL, RND_INV_COL, RND_INV_COL);
Is16vec8 ivec_round_inv_corr(RND_INV_CORR, RND_INV_CORR, RND_INV_CORR, RND_INV_CORR,
RND_INV_CORR, RND_INV_CORR, RND_INV_CORR, RND_INV_CORR);

Is16vec8 ivec_tg_1_16(13036, 13036, 13036, 13036,13036, 13036, 13036, 13036);
Is16vec8 ivec_tg_2_16(27146, 27146, 27146, 27146,27146, 27146, 27146, 27146);
Is16vec8 ivec_tg_3_16(-21746, -21746, -21746, -21746, -21746, -21746,-21746,-21746);
Is16vec8 ivec_cos_4_16(-19195, -19195, -19195, -19195,-19195, -19195,-19195,-19195);

//-----

// Table for rows 0,4 - constants are multiplied on cos_4_16

//movq -> w13 w12 w09 w08 w05 w04 w01 w00
```

```

//      w15 w14 w11 w10 w07 w06 w03 w02
//      w29 w28 w25 w24 w21 w20 w17 w16
//      w31 w30 w27 w26 w23 w22 w19 w18
//movq ->      w05 w04 w01 w00
__declspec(align(16)) short M128tab_i_04[] = {16384,      21407, 16384, 8867,
      16384, -8867, 16384, -21407, //      w13 w12 w09 w08
      16384, 8867, -16384, -21407, //      w07 w06 w03 w02
      -16384, 21407, 16384, -8867, //      w15 w14 w11 w10
      22725, 19266, 19266, -4520, //      w21 w20 w17 w16
      12873, -22725, 4520, -12873, //      w29 w28 w25 w24
      12873, 4520, -22725, -12873, //      w23 w22 w19 w18
      4520, 19266, 19266, -22725}; //      w31 w30 w27 w26

// Table for rows 1,7 - constants are multiplied on cos_1_16
//movq ->      w05 w04 w01 w00
__declspec(align(16)) short M128tab_i_17[] = {22725, 29692, 22725, 12299,
      22725, -12299, 22725, -29692, //      w13 w12 w09 w08
      22725, 12299, -22725, -29692, //      w07 w06 w03 w02
      -22725, 29692, 22725, -12299, //      w15 w14 w11 w10
      31521, 26722, 26722, -6270, //      w21 w20 w17 w16
      17855, -31521, 6270, -17855, //      w29 w28 w25 w24
      17855, 6270, -31521, -17855, //      w23 w22 w19 w18
      6270, 26722, 26722, -31521}; //      w31 w30 w27 w26

// Table for rows 2,6 - constants are multiplied on cos_2_16
//movq ->      w05 w04 w01 w00
__declspec(align(16)) short M128tab_i_26[] = {21407, 27969, 21407, 11585,
      21407, -11585, 21407, -27969, //      w13 w12 w09 w08
      21407, 11585, -21407, -27969, //      w07 w06 w03 w02
      -21407, 27969, 21407, -11585, //      w15 w14 w11 w10
      29692, 25172, 25172, -5906, //      w21 w20 w17 w16
      16819, -29692, 5906, -16819, //      w29 w28 w25 w24
      16819, 5906, -29692, -16819, //      w23 w22 w19 w18
      5906, 25172, 25172, -29692}; //      w31 w30 w27 w26

// Table for rows 3,5 - constants are multiplied on cos_3_16
//movq ->      w05 w04 w01 w00
__declspec(align(16)) short M128tab_i_35[] = {19266, 25172, 19266, 10426,
      19266, -10426, 19266, -25172, //      w13 w12 w09 w08
      19266, 10426, -19266, -25172, //      w07 w06 w03 w02
      -19266, 25172, 19266, -10426, //      w15 w14 w11 w10
      26722, 22654, 22654, -5315, //      w21 w20 w17 w16
      15137, -26722, 5315, -15137, //      w29 w28 w25 w24
      15137, 5315, -26722, -15137, //      w23 w22 w19 w18
      5315, 22654, 22654, -26722}; //      w31 w30 w27 w26

```

```

Is16vec8 *ivec_tab_i_04 = (Is16vec8*)M128tab_i_04;
Is16vec8 *ivec_tab_i_17 = (Is16vec8*)M128tab_i_17;
Is16vec8 *ivec_tab_i_26 = (Is16vec8*)M128tab_i_26;
Is16vec8 *ivec_tab_i_35 = (Is16vec8*)M128tab_i_35;

//-----

/*
;=====
;=====
;=====
;
; Inverse DCT
;
;-----
;
; This implementation calculates iDCT-2D by a row-column method.
; On the first stage the iDCT-1D is calculated for each row with use
; direct algorithm, on the second stage the calculation is executed
; at once for four columns with use of scaled iDCT-1D algorithm.
; Base R&Y algorithm for iDCT-1D is modified for second stage.
;
;=====

;-----
;
; The first stage - inverse DCTs of rows
;
;-----
; The 8-point inverse DCT direct algorithm
;-----
;
; static const short w[32] = {
;     FIX(cos_4_16),  FIX(cos_2_16),  FIX(cos_4_16),  FIX(cos_6_16),
;     FIX(cos_4_16),  FIX(cos_6_16), -FIX(cos_4_16), -FIX(cos_2_16),
;     FIX(cos_4_16), -FIX(cos_6_16), -FIX(cos_4_16),  FIX(cos_2_16),
;     FIX(cos_4_16), -FIX(cos_2_16),  FIX(cos_4_16), -FIX(cos_6_16),
;     FIX(cos_1_16),  FIX(cos_3_16),  FIX(cos_5_16),  FIX(cos_7_16),
;     FIX(cos_3_16), -FIX(cos_7_16), -FIX(cos_1_16), -FIX(cos_5_16),
;     FIX(cos_5_16), -FIX(cos_1_16),  FIX(cos_7_16),  FIX(cos_3_16),
;     FIX(cos_7_16), -FIX(cos_5_16),  FIX(cos_3_16), -FIX(cos_1_16) };
;
; #define DCT_8_INV_ROW(x, y)
; {
;     int a0, a1, a2, a3, b0, b1, b2, b3;

```

```

;
;   a0   = x[0] * w[ 0] + x[2] * w[ 1] + x[4] * w[ 2] + x[6] * w[ 3];
;   a1   = x[0] * w[ 4] + x[2] * w[ 5] + x[4] * w[ 6] + x[6] * w[ 7];
;   a2   = x[0] * w[ 8] + x[2] * w[ 9] + x[4] * w[10] + x[6] * w[11];
;   a3   = x[0] * w[12] + x[2] * w[13] + x[4] * w[14] + x[6] * w[15];
;   b0   = x[1] * w[16] + x[3] * w[17] + x[5] * w[18] + x[7] * w[19];
;   b1   = x[1] * w[20] + x[3] * w[21] + x[5] * w[22] + x[7] * w[23];
;   b2   = x[1] * w[24] + x[3] * w[25] + x[5] * w[26] + x[7] * w[27];
;   b3   = x[1] * w[28] + x[3] * w[29] + x[5] * w[30] + x[7] * w[31];
;
;   y[0] = SHIFT_ROUND ( a0 + b0 );
;   y[1] = SHIFT_ROUND ( a1 + b1 );
;   y[2] = SHIFT_ROUND ( a2 + b2 );
;   y[3] = SHIFT_ROUND ( a3 + b3 );
;   y[4] = SHIFT_ROUND ( a3 - b3 );
;   y[5] = SHIFT_ROUND ( a2 - b2 );
;   y[6] = SHIFT_ROUND ( a1 - b1 );
;   y[7] = SHIFT_ROUND ( a0 - b0 );
; }
;
;-----
;
; In this implementation the outputs of the iDCT-1D are multiplied
;   for rows 0,4 - on cos_4_16,
;   for rows 1,7 - on cos_1_16,
;   for rows 2,6 - on cos_2_16,
;   for rows 3,5 - on cos_3_16
; and are shifted to the left for rise of accuracy
;
; For used constants
;   FIX(float_const) = (short) (float_const * (1<<15) + 0.5)
;
;-----
;-----
;
; The second stage - inverse DCTs of columns
;
; The inputs are multiplied
;   for rows 0,4 - on cos_4_16,
;   for rows 1,7 - on cos_1_16,
;   for rows 2,6 - on cos_2_16,
;   for rows 3,5 - on cos_3_16
; and are shifted to the left for rise of accuracy
;
;-----

```

```

;
; The 8-point scaled inverse DCT algorithm (26a8m)
;
;-----
; //Inverse 1-D DCT algorithm for the eight columns
;   short t0, t1, t2, t3, t4, t5, t6, t7;
;   short tp03, tm03, tp12, tm12, tp65, tm65;
;   short tp465, tm465, tp765, tm765;
;
;   tp765 = x[1]          + x[7] * tg_1_16;
;   tp465 = x[1] * tg_1_16 - x[7];
;   tm765 = x[5] * tg_3_16 + x[3];
;   tm465 = x[5]          - x[3] * tg_3_16;
;
;   t7   = tp765 + tm765;
;   tp65 = tp765 - tm765;
;   t4   = tp465 + tm465;
;   tm65 = tp465 - tm465;
;
;   t6   = ( tp65 + tm65 ) * cos_4_16;
;   t5   = ( tp65 - tm65 ) * cos_4_16;
;
;   tp03 = x[0] + x[4];
;   tp12 = x[0] - x[4];
;
;   tm03 = x[2]          + x[6] * tg_2_16;
;   tm12 = x[2] * tg_2_16 - x[6];
;
;   t0   = tp03 + tm03;
;   t3   = tp03 - tm03;
;   t1   = tp12 + tm12;
;   t2   = tp12 - tm12;
;
;   y[0] = SHIFT_ROUND ( t0 + t7 );
;   y[7] = SHIFT_ROUND ( t0 - t7 );
;   y[1] = SHIFT_ROUND ( t1 + t6 );
;   y[6] = SHIFT_ROUND ( t1 - t6 );
;   y[2] = SHIFT_ROUND ( t2 + t5 );
;   y[5] = SHIFT_ROUND ( t2 - t5 );
;   y[3] = SHIFT_ROUND ( t3 + t4 );
;   y[4] = SHIFT_ROUND ( t3 - t4 );
;
;-----
*/
//emm7 = round_inv_row
#define DCT_8_INV_ROWX2

```

```

row = (Is16vec8)_mm_shufflelo_epi16(row,0xd8); \
row20 = (Is16vec8)_mm_shuffle_epi32(row,0x00); \
row20 = (Is16vec8)_mm_madd_epi16(row20,table1[0]); \
row31 = (Is16vec8)_mm_shuffle_epi32(row,0x55); \
row = (Is16vec8)_mm_shufflehi_epi16(row,0xd8); \
row31 = (Is16vec8)_mm_madd_epi16(row31,table1[2]); \
/*-----*/ \
row64 = (Is16vec8)_mm_shuffle_epi32(row,0xaa); \
row75 = (Is16vec8)_mm_shuffle_epi32(row,0xff); \
/*-----*/ \
row64 = (Is16vec8)_mm_madd_epi16(row64,table1[1]); \
r2ow = (Is16vec8)_mm_shufflehi_epi16(r2ow,0xd8); \
row20 = (Is32vec4)row20 + (Is32vec4)ivec_round_inv_row; \
r2ow = (Is16vec8)_mm_shufflelo_epi16(r2ow,0xd8); \
row75 = (Is16vec8)_mm_madd_epi16(row75,table1[3]); \
r2ow20 = (Is16vec8)_mm_shuffle_epi32(r2ow,0x00); \
r2ow64 = (Is16vec8)_mm_shuffle_epi32(r2ow,0xaa); \
r2ow20 = (Is16vec8)_mm_madd_epi16(r2ow20,table2[0]); \
/*-----*/ \
a = (Is32vec4)row20 + (Is32vec4)row64; \
r2ow31 = (Is16vec8)_mm_shuffle_epi32(r2ow,0x55); \
r2ow64 = (Is16vec8)_mm_madd_epi16(r2ow64,table2[1]); \
b = (Is32vec4)row31 + (Is32vec4)row75; \
r2ow75 = (Is16vec8)_mm_shuffle_epi32(r2ow,0xff); \
Yhigh = ((Is32vec4)((Is32vec4)a-(Is32vec4)b)) >> SHIFT_INV_ROW; \
r2ow31 = (Is16vec8)_mm_madd_epi16(r2ow31,table2[2]); \
Ylow = ((Is32vec4)((Is32vec4)a+(Is32vec4)b)) >> SHIFT_INV_ROW; \
r2ow20 = (Is32vec4)r2ow20 + (Is32vec4)ivec_round_inv_row; \
r2ow75 = (Is16vec8)_mm_madd_epi16(r2ow75,table2[3]); \
a2 = (Is32vec4)r2ow20 + (Is32vec4)r2ow64; \
Yhigh = (Is16vec8)_mm_shuffle_epi32(Yhigh, 0x1b); \
b2 = (Is32vec4)r2ow31 + (Is32vec4)r2ow75; \
Y2high = ((Is32vec4)((Is32vec4)a2-(Is32vec4)b2)) >> SHIFT_INV_ROW; \
Y2low = ((Is32vec4)((Is32vec4)a2+(Is32vec4)b2)) >> SHIFT_INV_ROW; \
Y2high = (Is16vec8)_mm_shuffle_epi32(Y2high, 0x1b); \

Is16vec8 tm765;
Is16vec8 tm465;
Is16vec8 tp765;
Is16vec8 tp465;
Is16vec8 tm65;
Is16vec8 tp65;
Is16vec8 tmp;
Is16vec8 tm03;
Is16vec8 tp03;
Is16vec8 tm12;

```

```

Isl6vec8 tp12;
Isl6vec8 t0;
Isl6vec8 t1;
Isl6vec8 t2;
Isl6vec8 t3;
Isl6vec8 t4;
Isl6vec8 t5;
Isl6vec8 t6;
Isl6vec8 t7;

//assumes src and destination are aligned on a 16-byte boundary

int idct_M128IVEC::TheCode()
{
// assert(((src & 0xf) == 0)&&((dst & 0xf) == 0)) //aligned on 16-byte boundary
Isl6vec8* src = (Isl6vec8*)dctcoeffshort;
Isl6vec8* dst = (Isl6vec8*)tst;
Isl6vec8 row20, row64, row31, row75, r2ow20, r2ow64, r2ow31, r2ow75;
Isl6vec8 a, b, Ylow, Yhigh, a2, b2, Y2low, Y2high;

Isl6vec8 row = src[2], r2ow = src[3];
Isl6vec8 *table1 = ivec_tab_i_26, *table2 = ivec_tab_i_35;
//
DCT_8_INV_ROWX2; //Row 3, tab_i_26 //Row 4, tab_i_35
//
dst[2] = (Isl6vec8)_mm_packs_epi32(Ylow, Yhigh);
dst[3] = (Isl6vec8)_mm_packs_epi32(Y2low, Y2high);

row = src[4]; r2ow = src[5];
table1 = ivec_tab_i_04; //table2 = ivec_tab_i_35;
//
DCT_8_INV_ROWX2; //Row 5, tab_i_04 //Row 6, tab_i_35
//
dst[4] = (Isl6vec8)_mm_packs_epi32(Ylow, Yhigh);
dst[5] = (Isl6vec8)_mm_packs_epi32(Y2low, Y2high);

row = src[0]; r2ow = src[1];
/*table1 = ivec_tab_i_04;*/ table2 = ivec_tab_i_17;
//
DCT_8_INV_ROWX2; //Row 1, tab_i_04 //Row 2, tab_i_17
//
dst[0] = (Isl6vec8)_mm_packs_epi32(Ylow, Yhigh);

dst[1] = (Isl6vec8)_mm_packs_epi32(Y2low, Y2high);

row = src[6]; r2ow = src[7];
table1 = ivec_tab_i_26; //table2 = ivec_tab_i_17;

```

```

//
DCT_8_INV_ROWX2; //Row 7, tab_i_26 //Row 8, tab_i_17
//
dst[6] = (Is16vec8)_mm_packs_epi32(Ylow, Yhigh);
dst[7] = (Is16vec8)_mm_packs_epi32(Y2low, Y2high); //store in Y2high
//so compiler can
//keep value in
//SIMD register

//
tm765 = mul_high(ivec_tg_3_16,dst[5]) + dst[5] + dst[3];
tm465 = dst[5] - (mul_high(ivec_tg_3_16,dst[3]) + dst[3]);
tp765 = mul_high(dst[7],ivec_tg_1_16) + dst[1];
tp465 = mul_high(dst[1],ivec_tg_1_16) - dst[7];
/*-----*/
tm65 = tp465 - tm465 + ivec_one_corr;
tp65 = tp765 - tm765;
t4 = tp465 + tm465;
t7 = tp765 + tm765 + ivec_one_corr;
/*-----*/
tmp = tp65 + tm65;
t6 = mul_high(tmp,ivec_cos_4_16) + tmp;
t6 |= ivec_one_corr;
tmp = tp65 - tm65;
t5 = mul_high(tmp,ivec_cos_4_16) + tmp;
t5 |= ivec_one_corr;
/*-----*/
tm03 = mul_high(dst[6],ivec_tg_2_16) + dst[2];
tm12 = mul_high(dst[2],ivec_tg_2_16) - dst[6];
tp03 = dst[0] + dst[4];
tp12 = dst[0] - dst[4];
/*-----*/
t0 = tp03 + tm03 + ivec_round_inv_col;
t3 = tp03 - tm03 + ivec_round_inv_corr;
t1 = tp12 + tm12 + ivec_round_inv_col;
t2 = tp12 - tm12 + ivec_round_inv_corr;
/*-----*/
dst[0] = (t0+t7) >> SHIFT_INV_COL;
dst[1] = (t1+t6) >> SHIFT_INV_COL;
dst[2] = (t2+t5) >> SHIFT_INV_COL;
dst[3] = (t3+t4) >> SHIFT_INV_COL;
dst[4] = (t3-t4) >> SHIFT_INV_COL;
dst[5] = (t2-t5) >> SHIFT_INV_COL;
dst[6] = (t1-t6) >> SHIFT_INV_COL;
dst[7] = (t0-t7) >> SHIFT_INV_COL;

```

```
    return(EXIT_SUCCESS);  
}
```

付録 A - パフォーマンス・データ

パフォーマンス・データの改訂履歴

改訂	改訂履歴	日付
2.0	1.20 GHz Pentium 4 のパフォーマンス・データに関する改訂	2000 年 7 月
1.0	初版	1999 年 9 月

表 1: IDCT のパフォーマンス・データ

パフォーマンス・データ(単位はマイクロ秒)		
テスト・ケース	Pentium III プロセッサ (733 MHz)	Pentium 4 プロセッサ (1.20 GHz)
ストリーミング SIMD 拡張命令 (SSE) ASM	0.386	0.335
SSE2 ASM	-	0.255
SSE2 IVEC	-	0.277

表 2: 表 1 のパフォーマンス・データから求めた高速化

コーディング方法とプラットフォーム	高速化
Pentium 4 プロセッサ(SSE2 ASM vs. SSE ASM)	1.31
SSE ASM(Pentium 4 プロセッサ vs. Pentium III プロセッサ)	1.15
Pentium 4 プロセッサにおける SSE2 ASM vs. Pentium III プロセッサにおける SSE	1.51

Pentium III プロセッサにおけるパフォーマンスは 733 MHz Pentium III、Pentium 4 プロセッサにおけるパフォーマンスは 1.2 GHz Pentium 4 プロセッサで測定した。表 2 に示すように、Pentium 4 プロセッサで実行すると、SSE2 を使用したコードの方が SSE を使用したコードより 1.31 倍高速だった。これは、次の最適化の結果である。

- **SIMD 幅の拡張。** 整数 SSE2 は、64 ビットの MMX テクノロジ・レジスタではなく、128 ビットの XMM レジスタを使用する。SIMD 幅が拡張されると、レジスタ制約が緩和し、1 つの命令で同時に処理できるデータ量が 2 倍になった。
- **1次元 IDCT における行の展開。** レジスタ制約が緩和されたため、行に対する 1次元 IDCT を展開し、2行同時に処理できるようになった。行を展開したのは、より多くの命令を並列処理するためである。特に、この展開により、`pmaddwd` 命令をより早く他の命令と並列して処理できるようになった。

- **レジスタ制約の緩和。** レジスタ制約が緩和されたため、アセンブリ言語(ASM)のコードから 8 つの命令を削除できた。レジスタに値を保持できるので、ASM コードから 2 つのロード命令と 2 つのストア命令が削除可能になった。また、行に対する IDCT を展開して 2 行同時に処理するようにしたので、行を適切に組み合わせ、4 つの実効アドレス・ロード (lea) 命令を削除できた。ASM コードから削除した 8 つの命令は、コメントにしてラベルを付けた(セクション 6 を参照)。

ASM コードの方が、クラス・ライブラリ(IVEC)を使用したコードよりわずかに高速だった。どちらのコードも SSE2 を使用した。IVEC を使用したコードでは、SIMD 演算のためのインテル® C++ クラス・ライブラリ(インテル® C\C++ コンパイラ・イントリンシックの C++ ラッパ)を使用した点が異なる。

イントリンシックとは、C の関数呼び出し構文を使用して、通常は一對一にマッピングされた SSE2 を実行する機能である。イントリンシックまたは C++ SIMD クラスを使用する利点は、長々とアセンブリ言語でプログラミングすることなく、SSE2 を使用できることである。欠点は、アセンブリ言語でなければできない最適化の存在である。例えば、ASM コードでは、2 つのロード命令と 2 つのストア命令を削除できたが、IVEC を使用したコードでは、コンパイラがレジスタの使用を制御するので、この最適化は不可能である。ASM コードが IVEC を使用したコードよりわずかに高速だったのは、このためである。

ただし、インライン ASM を使用すると、プロセス間およびプロセス内のコンパイラ最適化がオフになる可能性があるのに注意しなければならない。そのため、たいていのアプリケーションでは、C++ SIMD クラスやイントリンシックを使用する方が、インライン ASM を使用するより高速になる。C++ クラス・ライブラリとイントリンシックの詳細については、『*Intel C\C++ Class Libraries for SIMD Operations: With Support for the SSE2 Instructions*』資料番号 749100-001、および『*Intel C\C++ Compiler Intrinsics Reference Manual*』資料番号 748639-001 を参照のこと。

テスト・システム構成

表 3: Pentium III プロセッサのシステム構成

プロセッサ	Pentium III プロセッサ(733 MHz)
システム	インテル® Desktop Board VC820
BIOS バージョン	VC82010A.86A.0028.P10
2次キャッシュ	256 KB
メモリ・サイズ	128 MB RDRAM PC800-45
Ultra ATA ストレージ・ドライバ	製品候補 6.00.012
ハード・ディスク	IBM DJNA-371800 ATA-66
ビデオ・コントローラ/ バス	Creative Labs 3D Blaster† Annihilator† Pro AGP nVidia GeForce256† DDR –32MB
ビデオ・ドライバの リビジョン	NVidia Reference Driver 5.22
オペレーティング・ システム	Windows† 2000 ビルド 2195

表 4: Pentium 4 プロセッサのシステム構成

プロセッサ	Pentium 4 プロセッサ(1.20 GHz)
システム	インテル Desktop Board D850GB
BIOS バージョン	GB85010A.86A.0014.D.0007201756
2次キャッシュ	256 KB
メモリ・サイズ	128 MB RDRAM PC800-45
Ultra ATA ストレージ・ ドライバ	製品候補 6.00.012
ハード・ディスク	IBM DJNA-371800 ATA-66
ビデオ・コントローラ/ バス	Creative Labs 3D Blaster Annihilator Pro AGP nVidia GeForce256 DDR –32MB
ビデオ・ドライバの リビジョン	NVidia Reference Driver 5.22
オペレーティング・ システム	Windows 2000 ビルド 2195