



Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

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. Introduction
2. Overview
3. Functional Description
4. Color Converter Interface
5. Choosing Algorithm for Color Conversion To RGB24 Without Zooming
6. YUV12 to RGB24 Conversion Using Lookup Tables (first method)
 - 6.1 Extracting Y,U and V Impacts From Lookup Tables
 - 6.2 Aspect Ratio Calculation
 - 6.3 Size of Lookup Tables
7. YUV12 to RGB24 Zoom by Two
8. YUV12 to RGB16 Conversion Using Lookup Tables
9. YUV12 to RGB8 (CLUT8) Conversion Using Lookup Tables
 - 9.1 Algorithm Description
 - 9.2 Calculating UV Impact
 - 9.3 Calculating Y impact
10. Converting to RGB8, Zoom by 2
 - 10.1 Implementation Notes
11. Assumptions
12. Appendix 1. Definition of palette (used for color space conversion to RGB8)
13. Appendix 2. Color conversion to RGB8
14. Appendix 3. Color conversion to RGB8 zoom by 2
15. Appendix 4. Color conversion to RGB24
16. Appendix 5. Color Conversion to RGB24 Zoom by 2
17. Appendix 6. Color Conversion to RGB16
18. Appendix 7. Color Conversion to RGB16 Zoom by 2
19. References

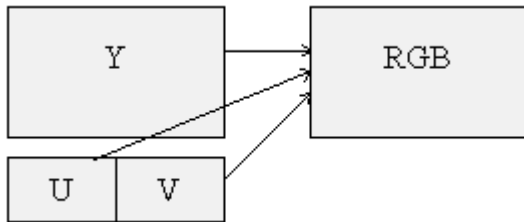
1. Introduction

This application note describes the usage of the new Intel MMX™ instruction set to implement Color Conversion Kernels (CCK) from YUV12 to RGB color space. The MMX™ instructions are Intel's implementation of Single Instruction Multiple Data (SIMD) instructions.

2. Overview

YUV12 color space is the native output for many video decoders including MPEG and H26x. This color space must be converted to RGB color space (the native color space of common PC graphics cards) to be displayed properly. Graphics cards support all or a subset of RGB8, RGB16, RGB24 or RGB32 color depths.

U and V are subsampled 2:1 in both vertical and horizontal directions. As a result, every U and V values are used for 4 Y values and generate 4 RGB pixels. The diagram shows that the number of bytes in the RGB buffer is the same as for the Y buffer. This is only true for RGB8. For RGB16, the number of bytes is twice as much, and for RGB24 it is 3 times as much.



3. Functional Description

For each 2x2 block of *RGB* pixels, 4 *Y* bytes 1 *U* and 1 *V* byte are needed as shown in Figure 1.

The input and output signals for *Y*, *U*, *V* fall within this range:

16 *Y* 235

16 *u,v* 240

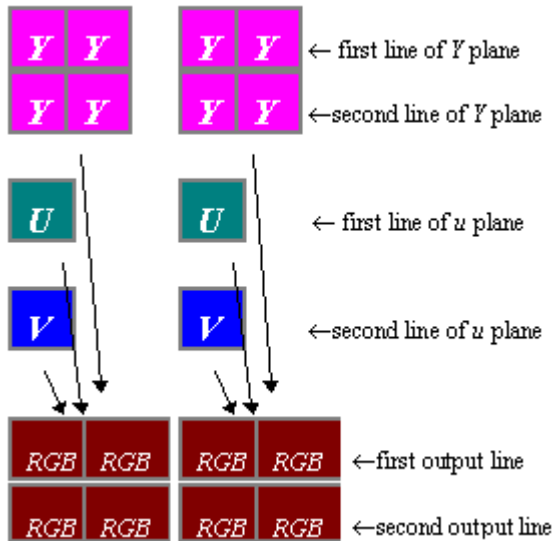


Figure 1- color conversion scheme

Conversion is performed according to the following:

$$G = 1.164 (Y-16) - 0.391(u-128) - 0.813(v-128)$$

$$R = 1.164 (Y-16) + 1.596(v-128)$$

$$B = 1.164 (Y-16) + 2.018(u-128)$$

The ranges of *R,G,B* values can be obtained by substituting the *Y,U,V* limits into the above equations, as follows:

$$-179 = 0 - 179 < R < 255 + 179 = 433$$

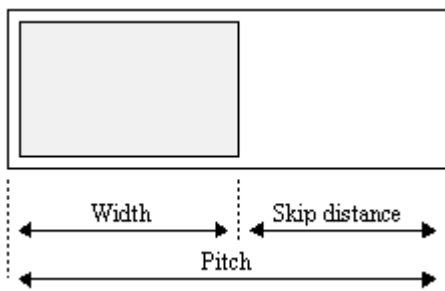
$$-135 = 0 - 135 < G < 255 + 135 = 390$$

$$-227 = 0 - 227 < B < 255 + 227 = 365$$

Once the *R,G,B* values are calculated, result should be translated to their final range. For example, in the case of *RGB24* format, each output pixel is represented by 24 bits; each color component is represented by one byte. Therefore, each of the *R,G,B* values must be clamped to within 0..255. The ranges for *RGB* above shows signed values, which means that the all calculations should use signed arithmetic. On the other hand, the final legal ranges of *RGB* is 0..255, which requires that the saturation uses unsigned arithmetics. For *RGB16*, the output range is further reduced to fit the *RGB* values in 16bits. This is done by dropping some of the least significant bits of each color.

4. Color Converter Interface

Each Color Converter Kernel (*CCK*) receives as input three planes: *Y*, *U*, *V*, a *Y* pitch, and *UV* pitch (*U*, *V* pitches are always the same). It also receives a pointer to the output buffer and its pitch (*CCOPitch*). In addition, it receives an *aspect ratio* adjustment count, which enables adjustment of the destination height to fit a specific aspect ratio of the display device.



5. Choosing Algorithm for Color Conversion To RGB24 Without Zooming

Three different implementations of the *YUV12* to *RGB24* algorithm using the MMX™ technology will be discussed in this section.

The first implementation of the algorithm utilizes the maximum parallelism offered by MMX™ Technology. It performs *byte* operations on 8 pixels at a time. This method uses pre-calculated tables and should yield the best throughput of the methods described here. However, since the temporary results during calculations may be larger than 8 bits, the *YUV* impact data is scaled down before calculations are made. This results in loss of precession of the final *RGB* data. However, this loss of data is not recognized by the naked eye and is very well acceptable.

The second method also uses lookup tables. It obtains precise final results by using MMX™ Technology to operate on *words*. This method has its own drawbacks, since only 4 pixels can be calculated at a time (compared to 8 in the first method). Moreover, the final *word* values have to be packed to *byte* format before storing it to the output buffer. Finally, the lookup tables doubles in size yielding worse cache locality.

The third approach uses direct calculations instead of lookup tables. This approach could be a good alternative to the first because it does not use lookup tables and thus has better cache behavior. Another advantage is realized because memory writes to the graphics card are uncached and slow which gives the CPU enough time to perform the required calculations. On the other hand, this method requires *word* arithmetic which reduces the amount of parallelism in half, and requires repacking the final results to *byte format*. Nonetheless, measurements show that this method can be as fast as the first approach.

6. YUV12 to RGB24 Conversion Using Lookup Tables(first method)

The YUV12 to RGB color conversion formulas could be represented as follows:

$$R = Y_impact[Y] + VR_impact[v]$$

$$G = Y_impact[Y] + UG_impact[u] + VG_impact[v]$$

$$B = Y_impact[Y] + UB_impact[u]$$

where (values from section 3):

$$Y_impact(Y) = 1.164(Y-16),$$

$$VR_impact(V) = 1.596(V-128)$$

$$VG_impact(V) = -0.813(V-128)$$

$$UG_impact(U) = -0.391(U-128)$$

$$UB_impact(U) = 2.018(U-128)$$

As mentioned above, for *byte* calculations, the *Y,U,V* impact data have to be scaled down so that the results do not exceed the data range. Using the scale factor $1/4$, ranges of *U,V* impact can be reduced to $-64..64$, and *Y* impact can be reduced to $0..64$. Adding the impacts together gives an *R,G,B* values between $-64..128$.

To clamp negative *R,G,B* values to 0, a constant 64 could be included in the *Y* impact tables which puts yields a range between $0..196$. As a result, all calculation could be *unsigned byte* operations, which is a perfect fit for the MMX™ technology.

Figure 2. illustrates a block diagram of this algorithm.

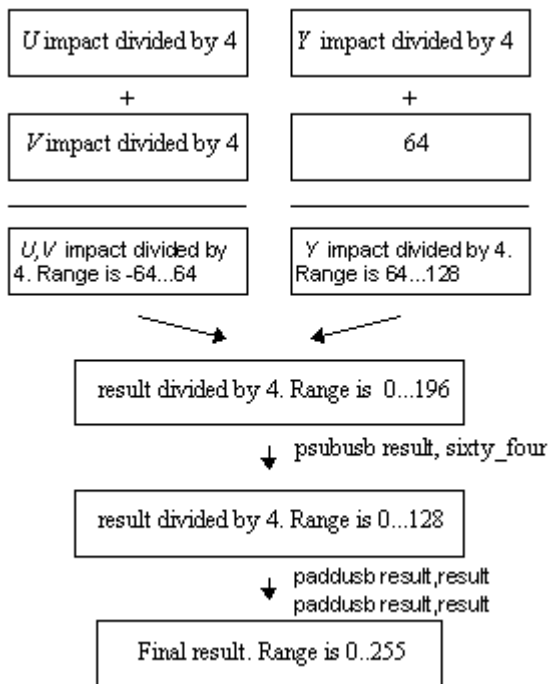


Figure 2- Conversion scheme YUV12 RGB24 using look up tables.

6.1 Extracting Y,U and V Impacts From Lookup Tables

The inner loop of the algorithm generates a 2x4 block of RGB pixels. It processes two lines at a time, since the impact of the *U* and *V* components is the same for two consecutive lines. Twelve bytes are generated for four RGB24 pixels. Thus three *dwords* are written to the output buffer.

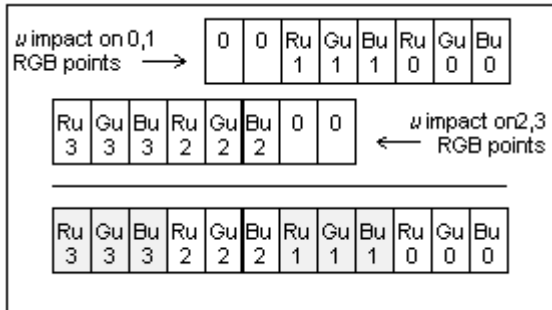


Figure 3- obtaining *u* impact on four RGB points.

As shown in Figure 3, the first *U* input byte is used to reference the *U_impact* table for the first 2 RGB pixels. The second *U* input byte is used to reference the *U_impact* table for the next 2 RGB pixels. The *UV_impact* will be used for two consecutive lines.

R	G	B	R	G	B	R	G	B	R	G	B
uv3	uv3	uv3	uv2	uv2	uv2	uv1	uv1	uv1	uv0	uv0	uv0

The *Y_impact* is calculated for each line. To get *Y_impact* on even-numbered lines (*Ye..*) four *Y_impact* values are combined together as follows:

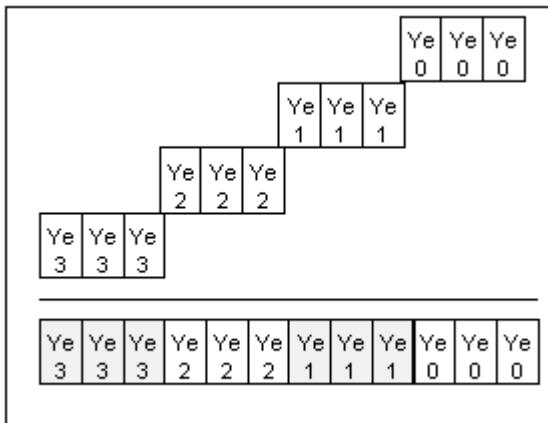


Figure 4- obtaining *Y* impact.

The *Y_impact* for odd-numbered lines is calculated in the same manner.

Yo	Yo	Yo	Yo	Yo	Yo	Yo	Yo	Yo	Yo	Yo	Yo
3	3	3	2	2	2	1	1	1	0	0	0

Adding the *Y* lines to the *U,V*-impact, and continuing to perform operations as illustrated in Figure 2, the final *R,G,B* results are generated as follows:

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

Re	Ge	Be	Re	Ge	Be	Re	Ge	Be	Re	Ge	Be
3	3	3	2	2	2	1	1	1	0	0	0

Ro	Go	Bo	Ro	Go	Bo	Ro	Go	Bo	Ro	Go	Bo
3	3	3	2	2	2	1	1	1	0	0	0

Optimized implementation of this algorithm is found in Appendix 4.

6.2 Aspect Ratio Calculation

The *Aspect ratio* parameter allows for adjustment of picture aspect ratio (width/height). The algorithm only allows for reduction in height of picture by dropping certain lines when generating the output. For example if the aspect ratio is 12, each 12th line is be dropped. Two solutions were considered. In the first one, each output line is processed separately and if the line number is a multiple of the *aspect ratio*, the line is dropped. The drawback of this solution is that the *UV* impacts, which are common for two consecutive lines, are either calculated twice, or stored in a temporary buffer. Both of them increase the amount of accessing required when no line is dropped, which is most of the cases.

The second solution always processes two lines at time. A line is skipped by writing the second calculated line over the first line. Thus, the amount of work is the same as if no lines are dropped at all. Therefore, the benefit of this method comes from the fact that *U, V* calculation is only done once.

6.3 Size of Lookup Tables

All tables contain 256 elements. The *Y* table contains *dword* entries, which yields 1K tables size. Each *U, V* table has *qword* entries, which yields 2K table each. Therefore, the total *Y, U, V* table size is 5K.

In the *U, V* tables, the *RGB* values in locations 0,1,2 are the same as the values in locations 3,4,5 respectively. This is due to the fact that *U, V* impacts two consecutive pixels. The *U, V* table sizes could be reduced by half eliminating the duplication. This could be done using shifts at run time to generate the proper format. However, this costs more CPU cycles.

To position the *Y* impacts in the right places, a *shift* instruction can be used. It is possible to use four tables for *Y*, and store shifted value in them. However, such tables will consume more memory, which could add additional pressure on the data cache.

7. YUV12 to RGB24 Zoom by Two

In this algorithm each output point is enlarged into 22 block. So now U and V values impact a 4x4 block, and Y values impact a 2x2 block. This algorithm was implemented using direct calculations of RGB values, and it uses the same ideas like RGB16 zoom by two.

Implementation of this algorithm can be found in Appendix 5.

8. YUV12 to RGB16 Conversion Using Lookup Tables

In the *RGB16* color format, every pixel is represented by 16-bit color components. Different graphics cards assign different number of bits for each of the *R*, *G* and *B* components, as follows:

x555 [ignore high order bit, then R,G,B where B is low]

655 [R=6(high), G=5, B=5(low)]

565 [R=5(high), G=6, B=5(low)]

664 [R=6(high), G=6, B=4(low)]

For example in x555 allocation, 5 bits are used to encode each color.

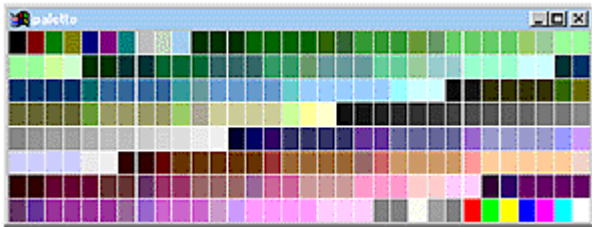
The first stage of *YUV12* to *RGB16* conversion is identical to *YUV12* to *RGB24* conversion. There is an additional step which decimates the *RGB24* color components and packs them into the appropriate 16 bit format.

Implementation of this algorithm can be found in Appendix 5.

9. YUV12 to RGB8 (CLUT8) Conversion Using Lookup Tables

9.1 Algorithm Description

RGB8 format represents each color in 8 bits, yielding a total of 256 colors. The contents of the 8 bits is an index into a *Color Lookup Table* known as a color palette. Graphics adapters are programmed with this palette either by the operating system or by the application. The operating system reserves the first 10 and last 10 entries of the palette for system usage. The rest of the entries are used by the active application.



(In 256 color mode this picture may look wrong. Use 16 or 24 bit color mode to see this picture properly)

The palette used for this implementation of RGB8 color is divided into 9 zones each with 26 gradients of the same color. *U* and *V* impacts are used to determine which color zone they represent, and the *Y* impact determines the intensity of the color in that zone. Definition of the palette may can be found in Appendix 1.

The *Y,U,V* impacts are calculated according to the following equations:

<i>V</i> impact=	0,	$U < 64h$
	1ah,	$64h \leq U < 84h$
	34h,	$U \geq 84h$
<i>U</i> impact=	0,	$V < 64h$
	4eh,	$64h \leq V < 84h$
	9ch,	$V \geq 84h$
<i>Y</i> impact=	0,	$Y < 10h$
	78,	$10h \leq Y < e6h$
	19h,	$Y \geq e6h$

Table 1 - Color Conversion Rules for RGB8 CCK

In addition, a noise pattern is added to the input *Y,U,V* values to give the picture a smooth look. The noise pattern is shown in Table 2. This extra processing consumes more precious cycles of the CPU, especially since that *U* and *V* impacts are different on different lines and thus must be calculated separately.

V-noise:

Line 1	10h	8	18h	0
Line 2	18h	0	10h	8
Line 3	8	10h	0	18h
Line 4	0	18h	8	10h

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

U-noise:

Line 1	8	10h	0	18h
Line 2	0	18h	8	10h
Line 3	10h	8	18h	0
Line 4	18h	0	10h	8

Y-noise:

Line 1	4	2	6	0
Line 2	6	0	4	2
Line 3	2	4	0	6
Line 4	0	6	2	4

Table 2 - Noise Matrixes for RGB8 CCK.

Since the noise values are added to the input Y, U, V data, the color conversion rules are different for every pixel in the 44 matrix. For example, consider the first pixel in *Line 1*. With the noise values added to it, a new color conversion table is derived as follows:

V impact=	0,	$U < 64h + 10h$
	1ah,	$10h + 64h \leq U < 84h + 10h$
	34h,	$U \geq 84h + 10h$
U impact=	0,	$V < 64h + 8h$
	4eh,	$8h + 64h \leq V < 84h + 8h$
	9ch,	$V \geq 84h + 8h$
Y impact=	0,	$Y < 16h + 4$
	78,	$4 + 16h \leq Y < ebh + 4$
	19h,	$Y \geq ebh + 4$

Table 3 - Color Conversion Rules for RGB8 CCK.

9.2 Calculating UV Impact

This implementation performs color conversion of 8 consecutive pixels at a time, as shown in Figure 5. To calculate the U impact, the algorithm loads 4 U bytes and duplicates them across the 8 bytes (since every U value impacts 2 neighboring pixels). The result is compared against the pre-calculated constants, U_{low_b} & U_{high_b} . Note that IA MMX™ Technology instructions compare only signed numbers; therefore, arguments should be converted to sign range. U_{low_b} & U_{high_b} are pre-calculated, such that the only needed conversion is needed at run time, for all of the 8 U bytes. The instruction `psubb mm0, convert_to_sign` does this conversion.

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

$151b1719151b1719 = 1b1b1b1b1b1b1b1b - 0600040206000402$

$1b1519171b151917 = 1b1b1b1b1b1b1b1b - 0006020400060204$

Adding *saturate_to_Y_high* constant, converts all values above *19h* to *FFh*, which puts it in the range *E6h..FFh*. Subtracting *return_from_Y_high* constant, brings all values to the range *Ah..23h*, which is *Y* range *0..19h* plus *Ah*. The constant *Ah* is added to the result; this is the first 10 reserved colors by the operating system.

$saturate_to_Y_high = e6e6e6e6e6e6e6 ; e6 = ff - 19$

$return_from_Y_high = dcdcdcdcdcdcdc ; ff - 19 - a$

Implementation of this algorithm can be found in Appendix 2.

10. Converting to *RGB8*, Zoom by 2

In this algorithm each output point is duplicated into a 2x2 block. Therefore, each *U* and *V* value impacts a 4x4 block, and each *Y* value impacts a 2x2 block. Before starting to calculate *Y,U,V* impacts from Table 1, the noise values are added (using matrixes from Table 2). To calculate *U* (and *V*) impact on the first line of the 4x4 block, use the technique shown in Figure 5 (one more *punpack* for duplicating *u* points should be added). *U*, *V* impacts are added together, giving a *UV* impact for 4 pixels in the block. Since the noise values are the same, but in different byte locations in the 4x4 matrix, the rest of the *UV* impacts for the following three lines could be calculated by shuffling these values accordingly. For example if *UV* impact on first line is:

UV3 UV2 UV1 UV0

then the rest of lines are:

UV1 UV0 UV3 UV2 - second line

UV2 UV3 UV0 UV1 - third line

UV0 UV1 UV2 UV3 - fourth line

The rest of the algorithm is similar to the non-zoomed algorithm.

10.1 Implementation Notes

Two algorithms were implemented for *YUV12* to *RGB8* color conversion..

The First algorithm has two sequential loops. The first loop calculates the common *UV impacts* on four lines. The results are stored in a temporary buffer. The second loop calculates the *Y impact* and combines them with the pre-calculated *UV impacts* to calculate the *RGB* pixel values. Each iteration of the second loop yields a 4x16 block of *RGB* pixels. This algorithm was found to be slow compared to the second algorithm (below), because of the nature of its calculations. The algorithm performs calculations of *RGB pixels*, and then writes them out to the graphics card. Due to the slow bandwidth of the graphics card compared to the CPU, the CPU write buffers were almost always full, causing a slow down in performance.

The second algorithm is based on interleaving the writes to the graphics card with *RGB* calculations. This algorithm is composed of one loop that calculates the *Y,U,V impacts* and combines them to generate the *RGB* values. As a result of the extra calculations of *U,V impacts* inside the loop, the size of the loop is increased, thus spreading the writes to the graphics card between calculations. The change in code structure resulted in a 1.3x speedup.

Implementation of the second algorithm can be found in Appendix 3.

11. Assumptions

For optimal performance, the algorithms assume that the output buffer is aligned on *qword* (8 byte) boundary. If it is aligned on 4 byte boundary, 4 bytes from the previous iteration and 4 bytes from the current iteration should be packed into *qword*. Then, write the 8 bytes to a *qword* aligned address. *Qword* writes are almost twice as fast as *dword* writes.

The code sample found in Appendix 5 are optimized for the Pentium® processor. The code samples for YUV to RGB24 converter with lookup tables is also optimized to avoid partial stalls on the Pentium Pro® processor.

12. Appendix 1. Definition of palette (used for color space conversion to RGB8).

As mentioned before, the first and last 10 colors are reserved by the operating system. Therefore, the first entry in the table corresponds to the 10th entry in the palette table. There are three values for each entry, corresponding to *blue*, *green* and *red* consecutively.

```
unsigned char
PalTable[26*9*3] = {
    0, 39+ 15, 0,
    0, 39+ 24, 0,
    0, 39+ 33, 0,
    0, 39+ 42, 0,
-44+ 51, 39+ 51, 0,
-44+ 60, 39+ 60, -55+ 60,
-44+ 69, 39+ 69, -55+ 69,
-44+ 78, 39+ 78, -55+ 78,
-44+ 87, 39+ 87, -55+ 87,
-44+ 96, 39+ 96, -55+ 96,
-44+105, 39+105, -55+105,
-44+114, 39+114, -55+114,
-44+123, 39+123, -55+123,
-44+132, 39+132, -55+132,
-44+141, 39+141, -55+141,
-44+150, 39+150, -55+150,
-44+159, 39+159, -55+159,
-44+168, 39+168, -55+168,
-44+177, 39+177, -55+177,
-44+186, 39+186, -55+186,
-44+195, 39+195, -55+195,
-44+204, 39+204, -55+204,
-44+213, 39+213, -55+213,
-44+222, 255, -55+222,
-44+231, 255, -55+231,
-44+240, 255, -55+240,
    0, 26+ 15, 0+ 15,
    0, 26+ 24, 0+ 24,
    0, 26+ 33, 0+ 33,
    0, 26+ 42, 0+ 42,
-44+ 51, 26+ 51, 0+ 51,
-44+ 60, 26+ 60, 0+ 60,
-44+ 69, 26+ 69, 0+ 69,
-44+ 78, 26+ 78, 0+ 78,
-44+ 87, 26+ 87, 0+ 87,
-44+ 96, 26+ 96, 0+ 96,
-44+105, 26+105, 0+105,
-44+114, 26+114, 0+114,
-44+123, 26+123, 0+123,
-44+132, 26+132, 0+132,
-44+141, 26+141, 0+141,
-44+150, 26+150, 0+150,
-44+159, 26+159, 0+159,
-44+168, 26+168, 0+168,
-44+177, 26+177, 0+177,
-44+186, 26+186, 0+186,
-44+195, 26+195, 0+195,
-44+204, 26+204, 0+204,
-44+213, 26+213, 0+213,
-44+222, 26+222, 0+222,
-44+231, 255, 0+231,
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
-44+240,    255,    0+240,
    0,    14+ 15,    55+ 15,
    0,    14+ 24,    55+ 24,
    0,    14+ 33,    55+ 33,
    0,    14+ 42,    55+ 42,
-44+ 51,    14+ 51,    55+ 51,
-44+ 60,    14+ 60,    55+ 60,
-44+ 69,    14+ 69,    55+ 69,
-44+ 78,    14+ 78,    55+ 78,
-44+ 87,    14+ 87,    55+ 87,
-44+ 96,    14+ 96,    55+ 96,
-44+105,    14+105,    55+105,
-44+114,    14+114,    55+114,
-44+123,    14+123,    55+123,
-44+132,    14+132,    55+132,
-44+141,    14+141,    55+141,
-44+150,    14+150,    55+150,
-44+159,    14+159,    55+159,
-44+168,    14+168,    55+168,
-44+177,    14+177,    55+177,
-44+186,    14+186,    55+186,
-44+195,    14+195,    55+195,
-44+204,    14+204,    255,
-44+213,    14+213,    255,
-44+222,    255,    255,
-44+231,    255,    255,
-44+240,    255,    255,
    0+ 15,    13+ 15,    0,
    0+ 24,    13+ 24,    0,
    0+ 33,    13+ 33,    0,
    0+ 42,    13+ 42,    0,
    0+ 51,    13+ 51,    0,
    0+ 60,    13+ 60,   -55+ 60,
    0+ 69,    13+ 69,   -55+ 69,
    0+ 78,    13+ 78,   -55+ 78,
    0+ 87,    13+ 87,   -55+ 87,
    0+ 96,    13+ 96,   -55+ 96,
    0+105,    13+105,   -55+105,
    0+114,    13+114,   -55+114,
    0+123,    13+123,   -55+123,
    0+132,    13+132,   -55+132,
    0+141,    13+141,   -55+141,
    0+150,    13+150,   -55+150,
    0+159,    13+159,   -55+159,
    0+168,    13+168,   -55+168,
    0+177,    13+177,   -55+177,
    0+186,    13+186,   -55+186,
    0+195,    13+195,   -55+195,
    0+204,    13+204,   -55+204,
    0+213,    13+213,   -55+213,
    0+222,    13+222,   -55+222,
    0+231,    13+231,   -55+231,
    0+240,    13+242,   -55+240,
    0+ 15,    0+ 15,    0+ 15,
    0+ 24,    0+ 24,    0+ 24,
    0+ 33,    0+ 33,    0+ 33,
    0+ 42,    0+ 42,    0+ 42,
    0+ 51,    0+ 51,    0+ 51,
    0+ 60,    0+ 60,    0+ 60,
    0+ 69,    0+ 69,    0+ 69,
    0+ 78,    0+ 78,    0+ 78,
    0+ 87,    0+ 87,    0+ 87,
    0+ 96,    0+ 96,    0+ 96,
    0+105,    0+105,    0+105,
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
0+114, 0+114, 0+114,
0+123, 0+123, 0+123,
0+132, 0+132, 0+132,
0+141, 0+141, 0+141,
0+150, 0+150, 0+150,
0+159, 0+159, 0+159,
0+168, 0+168, 0+168,
0+177, 0+177, 0+177,
0+186, 0+186, 0+186,
0+195, 0+195, 0+195,
0+204, 0+204, 0+204,
0+213, 0+213, 0+213,
0+222, 0+222, 0+222,
0+231, 0+231, 0+231,
0+240, 0+240, 0+240,
0+ 15, -13+ 15, 55+ 15,
0+ 24, -13+ 24, 55+ 24,
0+ 33, -13+ 33, 55+ 33,
0+ 42, -13+ 42, 55+ 42,
0+ 51, -13+ 51, 55+ 51,
0+ 60, -13+ 60, 55+ 60,
0+ 69, -13+ 69, 55+ 69,
0+ 78, -13+ 78, 55+ 78,
0+ 87, -13+ 87, 55+ 87,
0+ 96, -13+ 96, 55+ 96,
0+105, -13+105, 55+105,
0+114, -13+114, 55+114,
0+123, -13+123, 55+123,
0+132, -13+132, 55+132,
0+141, -13+141, 55+141,
0+150, -13+150, 55+150,
0+159, -13+159, 55+159,
0+168, -13+168, 55+168,
0+177, -13+177, 55+177,
0+186, -13+186, 55+186,
0+195, -13+195, 55+195,
0+204, -13+204, 255,
0+213, -13+213, 255,
0+222, -13+222, 255,
0+231, -13+231, 255,
0+240, -13+240, 255,
44+ 15, -14+ 15, 0,
44+ 24, -14+ 24, 0,
44+ 33, -14+ 33, 0,
44+ 42, -14+ 42, 0,
44+ 51, -14+ 51, 0,
44+ 60, -14+ 60, -55+ 60,
44+ 69, -14+ 69, -55+ 69,
44+ 78, -14+ 78, -55+ 78,
44+ 87, -14+ 87, -55+ 87,
44+ 96, -14+ 96, -55+ 96,
44+105, -14+105, -55+105,
44+114, -14+114, -55+114,
44+123, -14+123, -55+123,
44+132, -14+132, -55+132,
44+141, -14+141, -55+141,
44+150, -14+150, -55+150,
44+159, -14+159, -55+159,
44+168, -14+168, -55+168,
44+177, -14+177, -55+177,
44+186, -14+186, -55+186,
44+195, -14+195, -55+195,
44+204, -14+204, -55+204,
255, -14+213, -55+213,
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    255, -14+222, -55+222,
    255, -14+231, -55+231,
    255, -14+242, -55+240,
44+ 15,      0,      0+ 15,
44+ 24,      0,      0+ 24,
44+ 33, -26+ 33,    0+ 33,
44+ 42, -26+ 42,    0+ 42,
44+ 51, -26+ 51,    0+ 51,
44+ 60, -26+ 60,    0+ 60,
44+ 69, -26+ 69,    0+ 69,
44+ 78, -26+ 78,    0+ 78,
44+ 87, -26+ 87,    0+ 87,
44+ 96, -26+ 96,    0+ 96,
44+105, -26+105,    0+105,
44+114, -26+114,    0+114,
44+123, -26+123,    0+123,
44+132, -26+132,    0+132,
44+141, -26+141,    0+141,
44+150, -26+150,    0+150,
44+159, -26+159,    0+159,
44+168, -26+168,    0+168,
44+177, -26+177,    0+177,
44+186, -26+186,    0+186,
44+195, -26+195,    0+195,
44+204, -26+204,    0+204,
    255, -26+213,    0+213,
    255, -26+222,    0+222,
    255, -26+231,    0+231,
    255, -26+240,    0+240,
44+ 15,      0,      55+ 15,
44+ 24,      0,      55+ 24,
44+ 33,      0,      55+ 33,
44+ 42, -39+ 42,    55+ 42,
44+ 51, -39+ 51,    55+ 51,
44+ 60, -39+ 60,    55+ 60,
44+ 69, -39+ 69,    55+ 69,
44+ 78, -39+ 78,    55+ 78,
44+ 87, -39+ 87,    55+ 87,
44+ 96, -39+ 96,    55+ 96,
44+105, -39+105,    55+105,
44+114, -39+114,    55+114,
44+123, -39+123,    55+123,
44+132, -39+132,    55+132,
44+141, -39+141,    55+141,
44+150, -39+150,    55+150,
44+159, -39+159,    55+159,
44+168, -39+168,    55+168,
44+177, -39+177,    55+177,
44+186, -39+186,    55+186,
44+195, -39+195,    55+195,
44+204, -39+204,    255,
    255, -39+213,    255,
    255, -39+222,    255,
    255, -39+231,    255,
    255, -39+240,    255,
};
```

13. Appendix 2. Color conversion to RGB8.

The noise matrix is 4x4 in size. Therefore, even-numbered and odd-numbered lines have different noise values. However, since every loop processes 2 lines at a time, the noise values for the two lines must be calculated before entering the loop and stored in the appropriate variables.

tmpV3_U1low_bound[esp] - constants for odd line
tmpV3_U1high_bound[esp]
tmpU3_V1low_bound[esp]
tmpU3_V1high_bound[esp]

tmpV2_U0low_bound[esp] - constants for even line
tmpV2_U0high_bound[esp]
tmpU2_V0low_bound[esp]
tmpU2_V0high_bound[esp]

tmpY0_low[esp] - Constants for Y values
tmpY1_low[esp]

```
-----  
;  
;  
; cxm1281 -- This function performs YUV12 to CLUT8 color conversion for H26x.  
;          It dithers among 9 chroma points and 26 luma points, mapping the  
;          8 bit luma pels into the 26 luma points by clamping the ends and  
;          stepping the luma by 8.  
;  
;          Color convertor is not destructive.  
; Requirement:  
;          U and V plane SHOULD be followed by 4 bytes (for read only)  
;          Y plane SHOULD be followed by 8 bytes (for read only)  
.586P  
include iammx.inc  
ASSUME ds:FLAT, cs:FLAT, ss:FLAT  
-----  
PQ      equ PD  
PD      equ DWORD PTR  
-----  
;=====
```

```
_DATA SEGMENT PARA PUBLIC USE32 'DATA'  
    align 8  
PUBLIC Y0_low  
PUBLIC Y1_low  
PUBLIC U_low_value  
PUBLIC V_low_value  
PUBLIC U2_V0high_bound  
PUBLIC U2_V0low_bound  
PUBLIC U3_V1high_bound  
PUBLIC U3_V1low_bound  
PUBLIC V2_U0high_bound  
PUBLIC V2_U0low_bound  
PUBLIC V3_U1high_bound  
PUBLIC V3_U1low_bound  
PUBLIC return_from_Y_high  
PUBLIC saturate_to_Y_high  
PUBLIC clean_MSB_mask  
PUBLIC convert_to_sign  
if 0 ;old_constants  
V2_U0low_bound      dq 0f3ebfbe3f3ebfbe3h ; 746c7c64746c7c64 - 8080808080808080  
    U2_V0low_bound   dq 0ebf3e3fbefb3e3fbh ; 6c74647c6c74647c - 8080808080808080
```


Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
tmpV3_U1low_bound      = 24      ; qw
tmpV2_U0high_bound     = 32      ; qw
tmpU2_V0high_bound     = 40      ; qw
tmpU3_Vlhigh_bound     = 48      ; qw
tmpV3_Ulhigh_bound     = 56      ; qw
tmpY0_low              = 64      ; qw
tmpY1_low              = 72      ; qw
tmpBlockParity         = 80
AspectCount            = 84
tmpYCursorEven         = 88
tmpYCursorOdd          = 92
tmpCCOPitch            = 96
Old_esp                = 100
End_of_locals          = 104
LCL EQU <esp+>
;=====
; extern void "C" MMX_YUV12ToCLUT8 (
;
;         U8* YPlane,
;         U8* UPlane,
;         U8* VPlane,
;         UN  FrameWidth,
;         UN  FrameHeight,
;         UN  YPitch,
;         UN  VPitch,
;         UN  AspectAdjustmentCount,
;         U8* ColorConvertedFrame,
;         U32 DCIOffset,
;         U32 CCOffsetToLine0,
;         int  CCOPitch,
;         int  CCType)
;
; The local variables are on the stack.
; The tables are in the one and only data segment.
;
; CCOffsetToLine0 is relative to ColorConvertedFrame.
;
PUBLIC C MMX_YUV12ToCLUT8
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
MMX_YUV12ToCLUT8:
    push esi
    push edi
    push ebp
    push ebx
    mov  ebp,esp
    sub  esp,LocalFrameSize
    and  esp,0fffffff8h
    mov  [esp+Old_esp],ebp
    mov  ecx,[ebp+arg_YPitch]
    mov  ebx,[ebp+arg_FrameWidth]
    mov  eax,[ebp+arg_YPlane]
    add  eax,ebx          ; Points to end of Y even line
    mov  tmpYCursorEven[esp],eax
    add  eax,ecx          ; add YPitch
    mov  tmpYCursorOdd[esp],eax
    lea  edx,[edx+2*ebx]  ; final value of Y-odd-pointer
    mov  esi,PD [ebp+arg_VPlane]
    mov  edx,PD [ebp+arg_UPlane]
    mov  eax,PD [ebp+arg_ColorConvertedFrame]
    add  eax,PD [ebp+arg_DCIOffset]
    add  eax,PD [ebp+arg_CCOffsetToLine0]
    sar  ebx,1
    add  esi,ebx
    add  edx,ebx
    lea  edi,[eax+2*ebx]  ; CCOCursor
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    mov     ecx,[ebp+arg_AspectAdjustmentCount]
    mov     AspectCount[esp],ecx
    test   ecx,ecx      ; if AspectCount=0 we should not drop any lines
    jnz    non_zero_AspectCount
    dec    ecx
non_zero_AspectCount:
    mov     AspectCount[esp],ecx
    cmp    ecx,1
    jbe    finish
    neg    ebx
    mov     [ebp+arg_FrameWidth],ebx
    movq   mm6,PQ U_low_value      ; store some frequently used values in registers
    movq   mm7,PQ V_low_value
    xor    eax,eax
    mov    tmpBlockParity[esp],eax

; Register Usage:
;
; esi -- points to the end of V Line
; edx -- points to the end of U Line.
; edi -- points to the end of even line of output.
; ebp -- points to the end of odd line of output.
;
; ecx -- points to the end of even/odd Y Line
; eax -- 8*(line&2) == 0, on line%4=0,1
;           == 8, on line%4=2,3
;           in the loop, eax points to the end of even Y line
; ebx -- Number of points, we havn't done yet. (multiplied by -0.5)
;
;
;-----
; Noise matrix is of size 4x4 , so we have different noise values in even pair of lines,
; and in odd pair of lines. But in our loop we are doing 2 lines. So here we are preparing
; constants for next two lines.
; This code is done each time we are starting to convert next pair of lines.
PrepareNext2Lines:
    mov    eax,tmpBlockParity[esp]
;constants for odd line
    movq   mm0,PQ V3_U1low_bound[eax]
    movq   mm1,PQ V3_U1high_bound[eax]
    movq   mm2,PQ U3_V1low_bound[eax]
    movq   mm3,PQ U3_V1high_bound[eax]
    movq   PQ tmpV3_U1low_bound[esp],mm0
    movq   PQ tmpV3_U1high_bound[esp],mm1
    movq   PQ tmpU3_V1low_bound[esp],mm2
    movq   PQ tmpU3_V1high_bound[esp],mm3
;constants for even line
    movq   mm0,PQ V2_U0low_bound[eax]
    movq   mm1,PQ V2_U0high_bound[eax]
    movq   mm2,PQ U2_V0low_bound[eax]
    movq   mm3,PQ U2_V0high_bound[eax]
    movq   PQ tmpV2_U0low_bound[esp],mm0
    movq   PQ tmpV2_U0high_bound[esp],mm1
    movq   PQ tmpU2_V0low_bound[esp],mm2
    movq   PQ tmpU2_V0high_bound[esp],mm3
; Constants for Y values
    movq   mm4,PQ Y0_low[eax]
    movq   mm5,PQ Y1_low[eax]
    xor    eax,8
    mov    tmpBlockParity[esp],eax
    movq   PQ tmpY0_low[esp],mm4
    movq   PQ tmpY1_low[esp],mm5
; if AspectCount<2 we should skip a line. In this case we are steel doing two
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
; lines, but output pointers are the same, so we just overwriting line which we should skip
mov     eax,[ebp+arg_CCOPitch]
mov     ebx, AspectCount[esp]
xor     ecx,ecx
sub     ebx,2
mov     tmpCCOPitch[esp],eax
ja     continue
mov     eax,[ebp+arg_AspectAdjustmentCount]
mov     tmpCCOPitch[esp],ecx ; 0
lea     ebx,[ebx+eax] ; calculate new AspectCount
jnz    continue ; skipping even line
;skip_odd_line
mov     eax,tmpYCursorEven[esp]
; set odd constants to be equal to even_constants
; Odd line will be performed as even
movq    PQ tmpV3_U1low_bound[esp],mm0
movq    PQ tmpV3_U1high_bound[esp],mm1
movq    PQ tmpU3_V1low_bound[esp],mm2
movq    PQ tmpU3_V1high_bound[esp],mm3
movq    PQ tmpY1_low[esp],mm4
mov     tmpYCursorOdd[esp],eax
; when we got here, we already did all preparations.
; we are entering a main loop which is starts at do_next_8x2_block label
continue:
mov     AspectCount[esp],ebx

mov     ebx,[ebp+arg_FrameWidth]
mov     ebp,edi
add     ebp,tmpCCOPitch[esp] ; ebp points to the end of odd line of output
mov     eax,tmpYCursorEven[esp]
mov     ecx,tmpYCursorOdd[esp]
movdtd  mm0,[edx+ebx] ; read 4 U points
movdtd  mm2,[esi+ebx] ; read 4 V points
punpcklbw mm0,mm0 ; u3:u3:u2:u2|u1:u1:u0:u0
psubb   mm0,PQ convert_to_sign
punpcklbw mm2,mm2 ; v3:v3:v2:v2|v1:v1:v0:v0
movq    mm4,[eax+2*ebx] ; read 8 Y points from even line
movq    mm1,mm0 ; u3:u3:u2:u2|u1:u1:u0:u0
do_next_8x2_block:
psubb   mm2,PQ convert_to_sign ; convert to sign range (for comparison)
movq    mm5,mm1 ; u3:u3:u2:u2|u1:u1:u0:u0
pcmpgtb mm0,PQ tmpV2_U0low_bound[esp]
movq    mm3,mm2

pcmpgtb mm1,PQ tmpV2_U0high_bound[esp]
pand    mm0,U_low
psubusb mm4,PQ tmpY0_low[esp]
pand    mm1,U_high
pcmpgtb mm2,PQ tmpU2_V0low_bound[esp]
psrlq   mm4,3
pand    mm4,PQ clean_MSB_mask
pand    mm2,V_low

paddusb mm4,PQ saturate_to_Y_high
paddb   mm0,mm1 ; U03:U03:U02:U02|U01:U01:U00:U00
psubusb mm4,PQ return_from_Y_high
movq    mm1,mm5
pcmpgtb mm5,PQ tmpV3_U1low_bound[esp]
padd    mm0,mm2
pcmpgtb mm1,PQ tmpV3_U1high_bound[esp]
pand    mm5,U_low
padd    mm0,mm4
movq    mm2,mm3
pcmpgtb mm3,PQ tmpU2_V0high_bound[esp]
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    pand    mm1,U_high
    movq   mm4,[ecx+2*ebx]      ; read next 8 Y points from odd line
    paddb  mm5,mm1             ; u impact on odd line
    psubusb mm4,PQ tmpY1_low[esp]
    movq   mm1,mm2
    pcmptgb mm2,PQ tmpU3_V1low_bound[esp]
    psrlq  mm4,3
    pand   mm4,PQ clean_MSB_mask
    pand   mm2,V_low
    paddusb mm4,PQ saturate_to_Y_high
    padd   mm5,mm2
    psubusb mm4,PQ return_from_Y_high
    pand   mm3,V_high
    pcmptgb mm1,PQ tmpU3_V1high_bound[esp]
    paddb  mm3,mm0
    movdt  mm0,[edx+ebx+4]      ; read next 4 U points
    pand   mm1,V_high
    movdt  mm2,[esi+ebx+4]      ; read next 4 V points
    padd   mm5,mm4
    movq   mm4,[eax+2*ebx+8]    ; read next 8 Y points from even line
    paddb  mm5,mm1
    psubb  mm0,PQ convert_to_sign
    punpcklbw mm2,mm2          ; v3:v3:v2:v2|v1:v1:v0:v0
    movq   [edi+2*ebx],mm3      ; write even line
    punpcklbw mm0,mm0          ; u3:u3:u2:u2|u1:u1:u0:u0
    movq   [ebp+2*ebx],mm5      ; write odd line
    movq   mm1,mm0             ; u3:u3:u2:u2|u1:u1:u0:u0
    add    ebx,4
    jl     do_next_8x2_block

; update pointers to input and output buffers, to point to the next lines
    mov    ebp,[esp+Old_esp]
    mov    eax,tmpYCursorEven[esp]
    mov    ecx,[ebp+arg_YPitch]
    add    edi,[ebp+arg_CCOPitch]      ; go to the end of next line
    add    edi,tmpCCOPitch[esp]        ; skip odd line
    lea   eax,[eax+2*ecx]
    mov    tmpYCursorEven[esp],eax
    add    eax,[ebp+arg_YPitch]
    mov    tmpYCursorOdd[esp],eax
    add    esi,[ebp+arg_ChromaPitch]
    add    edx,[ebp+arg_ChromaPitch]
    sub    PD [ebp+arg_FrameHeight],2
    ja     PrepareNext2Lines

;-----
finish:

    emms

    mov    esp,[esp+Old_esp]
    pop    ebx
    pop    ebp
    pop    edi
    pop    esi
    ret
_TEXT ENDS
END
```


Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
LocalFrameSize      = 24
RegisterStorageSize = 16
; Arguments:
YPlane              = LocalFrameSize + RegisterStorageSize + 4
UPlane              = LocalFrameSize + RegisterStorageSize + 8
VPlane              = LocalFrameSize + RegisterStorageSize + 12
FrameWidth          = LocalFrameSize + RegisterStorageSize + 16
FrameHeight         = LocalFrameSize + RegisterStorageSize + 20
YPitch              = LocalFrameSize + RegisterStorageSize + 24
ChromaPitch         = LocalFrameSize + RegisterStorageSize + 28
AspectAdjustmentCount = LocalFrameSize + RegisterStorageSize + 32
ColorConvertedFrame = LocalFrameSize + RegisterStorageSize + 36
DCIOffset           = LocalFrameSize + RegisterStorageSize + 40
CCOffsetToLine0     = LocalFrameSize + RegisterStorageSize + 44
CCOPitch            = LocalFrameSize + RegisterStorageSize + 48
EndOfArgList        = LocalFrameSize + RegisterStorageSize + 56
; Locals (on local stack frame)
CCOCursor           = 0
DistanceFromVToU    = 4
AspectCount         = 8
CCOLine1            = 12
CCOLine2            = 16
CCOLine3            = 20
LCL EQU <esp+>
;=====
MMXCODE1 SEGMENT PARA USE32 PUBLIC 'CODE'
; extern void "C" MMX_YUV12ToCLUT8ZoomBy2 (
;
;             U8* YPlane,
;             U8* UPlane,
;             U8* VPlane,
;             UN  FrameWidth,
;             UN  FrameHeight,
;             UN  YPitch,
;             UN  VPitch,
;             UN  AspectAdjustmentCount,
;             U8* ColorConvertedFrame,
;             U32 DCIOffset,
;             U32 CCOffsetToLine0,
;             int CCOPitch,
;             int CCType)
;
; The local variables are on the stack.
; The tables are in the one and only data segment.
;
; CCOffsetToLine0 is relative to ColorConvertedFrame.
;
PUBLIC C MMX_YUV12ToCLUT8ZoomBy2
MMX_YUV12ToCLUT8ZoomBy2:
    push esi
    push edi
    push ebp
    push ebx
    sub  esp,LocalFrameSize
    mov  ebx,PD [esp+VPlane]
    mov  ecx,PD [esp+UPlane]
    sub  ecx,ebx
    mov  PD [esp+DistanceFromVToU],ecx
    mov  eax,PD [esp+ColorConvertedFrame]
    add  eax,PD [esp+DCIOffset]
    add  eax,PD [esp+CCOffsetToLine0]
    mov  PD [esp+CCOCursor],eax
;    Ledx  FrameHeight
;    Lecx  YPitch
;    imul edx,ecx
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    Ledi CCOPitch
    Lesi YPlane                ; Fetch cursor over luma plane.
    Seax CCOCursor
;   add  edx,esi
;   Sedx YLimit
    Ledx AspectAdjustmentCount
    Sedx AspectCount
    mov   edi,esi
    Lebx  FrameWidth
    Leax  CCOCursor
    sar   ebx,1
    sub   ebx,4                ; counter starts from maxvalue-4, and in last iteration it
equals 0
    mov   ecx,eax
    ADDedi YPitch              ; edi = odd Y line cursor
    ADDecx CCOPitch
    Sebx  FrameWidth
    Secx  CCOLine1
    Lebx  CCOPitch
; in each outer loop iteration, 4 lines of output are done.
; in each inner loop iteration block 4x16 of output is done.
; main task of outer loop is to prepare pointers for inner loop
NextFourLines:
; prepare output pointers
; ebx=CCOPitch
; eax=CCOLine0
; ecx=CCOLine1
    Lebp  AspectCount
    sub   ebp,2
    ja    continue1           ; jump if it still>0
    ADDebp AspectAdjustmentCount
    mov   ecx,eax             ; Output1 will overwrite Output0 line
    Secx  CCOLine1
continue1:
    lea   edx,[ecx+ebx]
    sub   ebp,2
    Sedx  CCOLine2
    ja    continue2           ; jump if it still>0
    ADDebp AspectAdjustmentCount
    xor   ebx,ebx             ; Output1 will overwrite Output0 line
continue2:
    Sebp  AspectCount
    lea   ebp,[edx+ebx]
    Sebp  CCOLine3
; output pointers are done
; Inner loop does 4x16 block of output points
; Register Usage
;
; esi -- Cursor over even Y line
; edi -- Cursor over odd Y line
; edx -- Cursor over V line
; ebp -- Cursor over U line.
; eax -- cursor over Output
; ecx -- cursor over Output1,2,3
; ebx -- counter
    Lebp  VPlane
    Lebx  FrameWidth
    mov   edx,ebp
    ADDebp DistanceFromVToU    ; Cursor over U line.
    movdt mm3,[ebp+ebx]        ; read 4 U points
    movdt mm2,[edx+ebx]        ; read 4 V points
    punpcklbw mm3,mm3          ; u3:u3:u2:u2|u1:u1:u0:u0
prepare_next4x8:
    psubb mm3,PQ convert_to_sign
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    punpcklwb mm2,mm2          ; v3:v3:v2:v2|v1:v1:v0:v0
    psubb    mm2,PQ convert_to_sign
    movq    mm4,mm3
    movdt   mm7,[esi+2*ebx]    ; read even Y line
    punpcklwd mm3,mm3        ; u1:u1:u1:u1|u0:u0:u0:u0
    Lecx    CCOLine1
    movq    mm1,mm3
    pcmpgtb mm3,PQ V2_U0low_bound
    punpcklwb mm7,mm7        ; y3:y3:y2:y2|y1:y1:y0:y0
    pand    mm3,PQ U_low_value
    movq    mm5,mm7
    psubusb mm7,PQ Y0_correct
    movq    mm6,mm2
    pcmpgtb mm1,PQ V2_U0high_bound
    punpcklwd mm2,mm2        ; v1:v1:v1:v1|v0:v0:v0:v0
    pand    mm1,PQ U_high_value
    psrlq   mm7,3
    pand    mm7,PQ clean_MSB_mask
    movq    mm0,mm2
    pcmpgtb mm2,PQ U2_V0low_bound
; empty slot !!!!
    pcmpgtb mm0,PQ U2_V0high_bound
    paddb   mm3,mm1
    pand    mm2,PQ V_low_value
    pand    mm0,PQ V_high_value
; two empty slots !!!!
    paddusb mm7,PQ saturate_to_Y_high
    paddb   mm3,mm2
    psubusb mm7,PQ return_from_Y_high ; Y impact on line0
    padd    mm3,mm0                ; common U,V impact on line 0
    psubusb mm5,PQ Y1_correct
    paddb   mm7,mm3                ; final value of line 0
    movq    mm0,mm3                ; u31:u21:u11:u01|u30:u20:u10:u00
    psrlq   mm5,3
    pand    mm5,PQ clean_MSB_mask
    psrld   mm0,16                  ;      : :u31:u21|      : :u30:u20
    paddusb mm5,PQ saturate_to_Y_high
    pslld   mm3,16                  ; u11:u01:      : |u10:u00:      :
    psubusb mm5,PQ return_from_Y_high ; Y impact on line0
    por     mm0,mm3                ; u11:u01:u31:u21|u10:u00:u30:u20
    movdt   mm3,[edi+2*ebx]        ; odd Y line
    paddb   mm5,mm0                ; final value of line 0
    punpcklwb mm3,mm3              ; y3:y3:y2:y2|y1:y1:y0:y0
    movq    mm2,mm0                ; u11:u01:u31:u21|u10:u00:u30:u20
    movq    [ecx+4*ebx],mm5        ; write Output1 line
    movq    mm1,mm3
    movq    [eax+4*ebx],mm7        ; write Output0 line
    psrlw   mm0,8                  ;      :u11:      :u31|      :u10:      :u30
    psubusb mm1,PQ Y3_correct
    psllw   mm2,8                  ; u01:      :u21:      |u00:      :u20:
    psubusb mm3,PQ Y2_correct
    psrlq   mm1,3
    pand    mm1,PQ clean_MSB_mask
    por     mm0,mm2                ; u01:u11:u21:u31|u00:u10:u20:u30
    paddusb mm1,PQ saturate_to_Y_high
    psrlq   mm3,3
    psubusb mm1,PQ return_from_Y_high
    movq    mm5,mm0                ; u01:u11:u21:u31|u00:u10:u20:u30
    pand    mm3,PQ clean_MSB_mask
    paddb   mm1,mm0
    paddusb mm3,PQ saturate_to_Y_high
    psrld   mm5,16
    psubusb mm3,PQ return_from_Y_high
    pslld   mm0,16
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
Lecx    CCOLine3
por     mm5,mm0                ; u21:u31:u01:u11|u20:u30:u00:u10
movdt  mm2,[esi+2*ebx+4]      ; read next even Y line
paddb  mm5,mm3
movq   [ecx+4*ebx],mm1        ; write Output3 line
punpckhwd mm4,mm4            ; u3:u3:u3:u3|u2:u2:u2:u2
; start next 4x8 block of output
; SECOND uv-QWORD
; mm6, mm4 are live
Lecx    CCOLine2
movq   mm3,mm4
pcmpgtb mm4,PQ V2_U0low_bound
punpckhwd mm6,mm6
movq   [ecx+4*ebx],mm5        ; write Output2 line
movq   mm7,mm6
pand   mm4,PQ U_low_value
punpcklbw mm2,mm2            ; y3:y3:y2:y2|y1:y1:y0:y0
pcmpgtb mm3,PQ V2_U0high_bound
movq   mm5,mm2
pand   mm3,PQ U_high_value
pcmpgtb mm6,PQ U2_V0low_bound
paddb  mm4,mm3
pand   mm6,PQ V_low_value
pcmpgtb mm7,PQ U2_V0high_bound
paddb  mm4,mm6
pand   mm7,PQ V_high_value
psubusb mm2,PQ Y0_correct
padd   mm4,mm7
psubusb mm5,PQ Y1_correct
psrlq  mm2,3
pand   mm2,PQ clean_MSB_mask
movq   mm3,mm4                ; u31:u21:u11:u01|u30:u20:u10:u00
paddusb mm2,PQ saturate_to_Y_high
pslld  mm3,16                 ; u11:u01:  : |u10:u00:  :
psubusb mm2,PQ return_from_Y_high
psrlq  mm5,3
pand   mm5,PQ clean_MSB_mask
paddb  mm2,mm4                ; MM4=u31:u21:u11:u01|u30:u20:u10:u00, WHERE U STANDS FOR UNATED U
AND V IMPACTS
paddusb mm5,PQ saturate_to_Y_high
psrld  mm4,16                 ;  :  :u31:u21|  :  :u30:u20
psubusb mm5,PQ return_from_Y_high
por    mm4,mm3                ; u11:u01:u31:u21|u10:u00:u30:u20
paddb  mm5,mm4
Lecx    CCOLine1
movdt  mm0,[edi+2*ebx+4]      ; read odd Y line
movq   mm7,mm4                ; u11:u01:u31:u21|u10:u00:u30:u20
movq   [ecx+4*ebx+8],mm5        ; write Output1 line
punpcklbw mm0,mm0            ; y3:y3:y2:y2|y1:y1:y0:y0
movq   [eax+4*ebx+8],mm2        ; write Output0 line
movq   mm1,mm0
psubusb mm1,PQ Y2_correct
psrlw  mm4,8                  ;  :u11:  :u31|  :u10:  :u30
psubusb mm0,PQ Y3_correct
psrlq  mm1,3
pand   mm1,PQ clean_MSB_mask
psllw  mm7,8                  ; u01:  :u21:  |u00:  :u20:
paddusb mm1,PQ saturate_to_Y_high
por    mm4,mm7                ; u01:u11:u21:u31|u00:u10:u20:u30
psubusb mm1,PQ return_from_Y_high
psrlq  mm0,3
pand   mm0,PQ clean_MSB_mask
movq   mm5,mm4                ; u01:u11:u21:u31|u00:u10:u20:u30
paddusb mm0,PQ saturate_to_Y_high
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    psrld    mm5,16
    psubusb mm0,PQ return_from_Y_high
    paddb   mm0,mm4
    Lecx    CCOLine3
    movdt   mm3,[ebp+ebx-4]           ; read next 4 U points
    pslld   mm4,16
    movq    [ecx+4*ebx+8],mm0         ; write Output3 line
    por     mm5,mm4                   ; u21:u31:u01:u11|u20:u30:u00:u10
    paddb   mm5,mm1
    Lecx    CCOLine2
    movdt   mm2,[edx+ebx-4]         ; read next 4 V points
    punpcklbw mm3,mm3                ; u3:u3:u2:u2|u1:u1:u0:u0
    movq    [ecx+4*ebx+8],mm5         ; write Output2 line
    sub     ebx,4
    jae     prepare_next4x8
    Lebx    CCOPitch
    Lecx    CCOLine3
    Lebpb   YPitch
    Ledx    VPlane
    lea    eax,[ecx+ebx]             ; next Output0 = old Output3 + CCOPitch
    lea    ecx,[ecx+2*ebx]           ; next Output1 = old Output3 + 2* CCOPitch
    ADDedx  ChromaPitch
    Secx    CCOLine1
    lea    esi,[esi+2*ebp]           ; even Y line cursor goes to next line
    lea    edi,[edi+2*ebp]           ; odd Y line cursor goes to next line
    Sedx    VPlane                   ; edx will point to V plane
    sub    PD FrameHeight[esp],2
    ja     NextFourLines
emms
add    esp,LocalFrameSize
pop    ebx
pop    ebp
pop    edi
pop    esi
retn
MMXCODE1 ENDS
END
```

15. Appendix 4. Color conversion to *RGB24*.

This code sample is an optimized version of color conversion from *YUV12* to *RGB24* format.

```
-----  
; cx512241 -- This function performs YUV12-to-RGB24 color conversion for H26x.  
; It is tuned for best performance on the Pentium(r) Microprocessor.  
; It handles the format in which the low order byte is B, the  
; second byte is G, and the high order byte is R.  
;  
; The YUV12 input is planar, 8 bits per pel. The Y plane may have  
; a pitch of up to 768. It may have a width less than or equal  
; to the pitch. It must be DWORD aligned, and preferably QWORD  
; aligned. Pitch and Width must be a multiple of four. For best  
; performance, Pitch should not be 4 more than a multiple of 32.  
; Height may be any amount, but must be a multiple of two. The U  
; and V planes may have a different pitch than the Y plane, subject  
; to the same limitations.  
;  
; The color convertor is destructive; the input Y, U, and V  
; planes will be clobbered. The Y plane MUST be preceded by  
; 3104 bytes of space for scratch work.  
OPTION PROLOGUE:None  
OPTION EPILOGUE:ReturnAndRelieveEpilogueMacro  
include iammx.inc  
include locals.inc  
.xlist  
.list  
.DATA  
; any data would go here  
ALIGN 8  
sixty_four dd 40404040h, 40404040h  
include small_ta.asm  
.CODE  
ASSUME ds:FLAT, cs:FLAT, ss:FLAT  
; void FAR ASM_CALLTYPE MMX_YUV12ToRGB24 (  
; U8* Yplane,  
; U8* Uplane,  
; U8* Vplane,  
; UN FrameWidth,  
; UN FrameHeight,  
; UN YPitch,  
; UN VPitch,  
; UN AspectAdjustmentCount,  
; U8* ColorConvertedFrame,  
; U32 DCIOffset,  
; U32 CCOffsetToLine0,  
; IN CCOPitch,  
; IN CCType)  
;  
; The local variables are on the stack.  
; The tables are in the one and only data segment.  
;  
; CCOffsetToLine0 is relative to ColorConvertedFrame.  
;  
PUBLIC C YUV12ToRGB24  
; due to the need for the ebp reg, these parameter declarations aren't used,  
; they are here so the assembler knows how many bytes to relieve from the stack  
LocalFrameSize = 40  
RegisterStorageSize = 16  
; Arguments:  
; Arguments:
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
YPlane          = LocalFrameSize + RegisterStorageSize + 4
UPlane          = LocalFrameSize + RegisterStorageSize + 8
VPlane          = LocalFrameSize + RegisterStorageSize + 12
FrameWidth      = LocalFrameSize + RegisterStorageSize + 16
FrameHeight     = LocalFrameSize + RegisterStorageSize + 20
YPitch          = LocalFrameSize + RegisterStorageSize + 24
ChromaPitch     = LocalFrameSize + RegisterStorageSize + 28
AspectAdjustmentCount = LocalFrameSize + RegisterStorageSize + 32
ColorConvertedFrame = LocalFrameSize + RegisterStorageSize + 36
DCIOffset       = LocalFrameSize + RegisterStorageSize + 40
CCOffsetToLine0 = LocalFrameSize + RegisterStorageSize + 44
CCOPitch        = LocalFrameSize + RegisterStorageSize + 48
EndOfArgList    = LocalFrameSize + RegisterStorageSize + 52
; Locals (on local stack frame)
CCOCursor       = 0
CCOSkipDistance = 4
ChromaLineLen   = 8
YSkipDistance   = 12
YCursor         = 16
DistanceFromVToU = 20
tmpYCursorEven  = 24
tmpYCursorOdd   = 28
tmpCCOPitch     = 32
AspectCount     = 36
LCL EQU <esp+>
YUV12ToRGB24:
    push esi
    push edi
    push ebp
    push ebx

    sub esp,LocalFrameSize
    mov ebx,PD [esp+VPlane]
    mov ecx,PD [esp+UPlane]
    sub ecx,ebx
    mov PD [esp+DistanceFromVToU],ecx
    mov eax,PD [esp+ColorConvertedFrame]
    add eax,PD [esp+DCIOffset]
    add eax,PD [esp+CCOffsetToLine0]
    mov PD [esp+CCOCursor],eax
; Ledx FrameHeight
    Lecx YPitch
; imul edx,ecx ; FrameHeight*YPitch
Lebx FrameWidth
    Leax CCOPitch
    sub eax,ebx ; CCOPitch-FrameWidth
    sub ecx,ebx ; YPitch-FrameWidth
    sub eax,ebx ; CCOPitch-2*FrameWidth
    Secx YSkipDistance
    sub eax,ebx ; CCOPitch-3*FrameWidth
    Lesi YPlane ; Fetch cursor over luma plane.
    sar ebx,1 ; FrameWidth/2
    Seax CCOSkipDistance ; CCOPitch-3*FrameWidth
    add edx,esi ; YPlane+Size_of_Y_array
    Sebxc ChromaLineLen ; FrameWidth/2
; Sedx YLimit
    Sesi YCursor
    Ledx AspectAdjustmentCount
    Lesi VPlane
    test edx,edx ; if AspectCount=0 we should not drop any lines
    jnz non_zero_AspectCount
    dec edx
non_zero_AspectCount:
    Sedx AspectAdjustmentCount
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    xor    eax,eax
Lebp    DistanceFromVToU
Ledi    YCursor                ; Fetch Y Pitch.
    lebx  FrameWidth

    add    edi,ebx
    Sedi  tmpYCursorEven
    Leax  YPitch
    add    edi,eax
    Sedi  tmpYCursorOdd
    sar   ebx,1
    add    esi,ebx
    add    ebp,esi
    neg   ebx
    Sebx  FrameWidth
Ledi    CCOCursor

; Register Usage:
;
; edx -- Y Line cursor.  Chroma contribs go in lines above current Y line.
; esi -- V Line cursor.
; ebp -- U Line cursor
; edi -- Cursor over the color converted output image.
; ebx -- Number of points, we havn't done yet.
;
;
; ecx -- V contribution to RGB; sum of U and V contributions.
; eax -- Alternately a U and a V pel.
;-----
    sub    edi,12
    movq   mm7,sixty_four
    Leax  AspectAdjustmentCount
    Seax  AspectCount
    cmp   eax,1
    jbe   finish
PrepareChromaLine:
    lebx  FrameWidth
    Leax  AspectCount
    Ledx  CCOPitch
    xor   ecx,ecx
    sub   eax,2
    Sedx  tmpCCOPitch
    ja    continue
    Leax  AspectAdjustmentCount
    Secx  tmpCCOPitch    ; 0
    jnz   skip_even_line
skip_odd_line:
    Ledx  tmpYCursorEven
    Seax  AspectCount
    Sedx  tmpYCursorOdd
    jmp   do_next_4x2_block
skip_even_line:
    dec   eax

continue:
    Seax  AspectCount
    Ledx  tmpYCursorEven

    xor   eax,eax
    mov   cl,[edx+2*ebx]    ; Ye0
    mov   al,[edx+2*ebx+1] ; Ye1
    movdt mm1,PD Y0[eax*4] ; 0: 0: 0: 0| 0:Ye1: Ye1: Ye1
do_next_4x2_block:
    movdt mm3,PD Y0[ecx*4] ; 0: 0: 0: 0| 0:Ye0: Ye0: Ye0
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```

psllq mm1,24 ; 0: 0: Ye1: Ye1| Ye1: 0: 0: 0
xor ecx,ecx
mov al,[edx+2*ebx+2] ; Ye2

mov cl,[edx+2*ebx+3] ; Ye3
xor edx,edx
mov dl,[esi+ebx+1] ; v1
add edi,12 ; output
movdt mm4,PD Y0[eax*4] ; 0: 0: 0: 0| 0: 0:Ye2 : Ye2
por mm3,mm1 ; 0: 0: Ye1: Ye1| Ye1:Ye0: Ye0: Ye0

movdt mm5,PD Y0[ecx*4] ; 0: 0: Ye3: Ye3: Ye3
psllq mm4,48 ;Ye2 : Ye2: 0: 0| 0: 0: 0: 0
mov cl,[ebp+ebx] ; u0
mov al,[esi+ebx] ; v0
movq mm2,PD v0[edx*8] ; 0: 0: Rv3: Gv3| Bv3: Rv2: Gv2: Bv2 u,v impact on RGB[0]
and RGB[1] is equal
por mm3,mm4 ;Ye2 : Ye2: Ye1: Ye1| Ye1: Ye0: Ye0: Ye0
movq mm0,PD u0[ecx*8] ; 0: 0: Ru1: Gu1| Bu1: Ru0: Gu0: Bu0 u,v impact on RGB[0]
and RGB[1] is equal
psllq mm5,8 ; 0: 0: Ye3: Ye3: Ye3: 0
mov cl,[ebp+ebx+1] ; u1
Ledx tmpYCursorOdd
paddb mm0,PD v0[ecx*8] ; 0: 0:Ruv1:Guv1|Buv1:Ruv0:Guv0:Bu0
psrlq mm4,56 ; 0: 0: 0: 0| 0: 0: 0: Ye2

paddb mm2,PD u0[ecx*8] ; 0: 0:Ruv3:Guv3|Buv3:Ruv2:Guv2:Bu2
por mm4,mm5 ; 0: 0: Ye3: Ye3: Ye3: Ye2
movq mm1,mm2
psllq mm2,48 ;Guv2:Bu2: 0: 0| 0: 0: 0: 0

psrlq mm1,16 ; 0: 0: 0: 0|Ruv3:Guv3:Bu3:Ruv2
por mm0,mm2 ;Guv2:Bu2:Ruv1:Guv1|Buv1:Ruv0:Guv0:Bu0
paddb mm3,mm0 ; r0:g0:b0:r1|g1:b1:r2:g2
mov cl,[edx+2*ebx+1] ; Yo1
mov al,[edx+2*ebx] ; Yo0
psubusb mm3,mm7 ; mm7=sixty_four
movdt mm6,PD Y0[ecx*4] ; 0: 0: Ye1: Ye1| Ye1:Ye0: Ye0: Ye0
paddb mm4,mm1 ; x: x: 0: 0| b2: r3: g3: b3
movdt mm5,PD Y0[eax*4] ; 0: 0: 0: 0| 0:Ye0: Ye0: Ye0
psubusb mm4,mm7 ; mm7=sixty_four
psllq mm6,24 ; 0: 0: 0: 0| 0:Ye0: Ye0: Ye0
mov al,[edx+2*ebx+2] ; Yo2
paddusb mm3,mm3
por mm5,mm6
movdt mm6,PD Y0[eax*4] ; 0: 0: 0: 0| 0: 0: Ye2: Ye2
paddusb mm3,mm3
paddusb mm4,mm4
psllq mm6,48 ;Ye2: Ye2: 0: 0| 0: 0: 0: 0
movdf [edi],mm3
paddusb mm4,mm4
mov cl,[edx+2*ebx+3] ; Yo3
psrlq mm3,32
movdf [edi+8],mm4
por mm5,mm6 ;Ye2: Ye2: Ye1: Ye1| Ye1: Ye0: Ye0: Ye0
movdt mm2,PD Y0[ecx*4] ; 0: 0: Ye3: Ye3: Ye3: Ye2
paddb mm5,mm0 ; r0:g0:b0:r1|g1:b1:r2:g2
psllq mm2,8
Ledx tmpYCursorEven
psubusb mm5,mm7 ; mm7=sixty_four
psrlq mm6,56 ; 0: 0: 0: 0| 0: 0: 0: Ye2
por mm6,mm2 ; 0: 0: Ye3: Ye3: Ye3: Ye2
paddusb mm5,mm5
Leax tmpCCOPitch

```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
paddusb mm5,mm5
paddb mm6,mm1 ; x: x: 0: 0| b2: r3: g3: b3
mov cl,[edx+2*ebx+1+4] ; Ye1
movdf [edi+eax],mm5
psrlq mm5,32
movdf [edi+4],mm3
psubusb mm6,mm7 ; mm7=sixty_four
movdf [edi+eax+4],mm5
paddusb mm6,mm6
movdt mm1,PD Y0[ecx*4] ; 0: 0: 0: 0| 0:Ye1: Ye1: Ye1
paddusb mm6,mm6
mov cl,[edx+2*ebx+4] ; Ye0
add ebx,2
movdf [edi+eax+8],mm6
mov eax,zero
jl do_next_4x2_block
Leax YPitch
ADDEDI CCOSkipDistance ; go to begin of next line
ADDEDI tmpCCOPitch ; skip odd line
Ledx tmpYCursorEven
lea edx,[edx+2*eax]
Sedx tmpYCursorEven
ADDEDX YPitch
Sedx tmpYCursorOdd
ADDESI ChromaPitch
ADDEBP ChromaPitch
sub PD FrameHeight[esp],2 ; Done with last line?
ja PrepareChromaLine
;-----
finish:
add esp,LocalFrameSize
emms
pop ebx
pop ebp
pop edi
pop esi
return
END
```

16. Appendix 5. Color Conversion to RGB24 Zoom by 2.

```
-----  
OPTION PROLOGUE:None  
OPTION EPILOGUE:ReturnAndRelieveEpilogueMacro  
include iammx.inc  
include locals.inc  
.586  
.xlist  
.list  
  ASSUME ds:FLAT, cs:FLAT, ss:FLAT  
MMXCODE1 SEGMENT PARA USE32 PUBLIC 'CODE'  
MMXCODE1 ENDS  
MMXDATA1 SEGMENT PARA USE32 PUBLIC 'DATA'  
MMXDATA1 ENDS  
MMXDATA1 SEGMENT  
; any data would go here  
  ALIGN 8  
;constants for direct RGB calculation: 4x10.6 values  
Minusg          dd  00800080h,00800080h  
VtR             dd  00660066h,00660066h ;01990199h,01990199h  
UtB            dd  00810081h,00810081h ;02050205h,02050205h  
Ymul           dd  004a004ah,004a004ah ;012a012ah,012a012ah  
Yadd           dd  10101010h,10101010h  
UVtG           dd  00340019h,00340019h ;00d00064h,00d00064h  
MASK_036       dd  0ff0000ffh,00ff0000h  
MASK_147       dd  0000ff00h,0ff0000ffh  
tmpYCursorEven dd  0  
tmpYCursorOdd  dd  0  
tmpBuffer      db  48 dup (?) ; aligned on 8 byte boundary scratch buffer  
MMXDATA1 ENDS  
LocalFrameSize = 20  
RegisterStorageSize = 16  
; Arguments:  
YPlane          = LocalFrameSize + RegisterStorageSize + 4  
UPlane          = LocalFrameSize + RegisterStorageSize + 8  
VPlane          = LocalFrameSize + RegisterStorageSize + 12  
FrameWidth      = LocalFrameSize + RegisterStorageSize + 16  
FrameHeight     = LocalFrameSize + RegisterStorageSize + 20  
YPitch          = LocalFrameSize + RegisterStorageSize + 24  
ChromaPitch     = LocalFrameSize + RegisterStorageSize + 28  
AspectAdjustmentCount = LocalFrameSize + RegisterStorageSize + 32  
ColorConvertedFrame = LocalFrameSize + RegisterStorageSize + 36  
DCIOffset       = LocalFrameSize + RegisterStorageSize + 40  
CCOffsetToLine0 = LocalFrameSize + RegisterStorageSize + 44  
CCOPitch        = LocalFrameSize + RegisterStorageSize + 48  
CCType          = LocalFrameSize + RegisterStorageSize + 52  
EndOfArgList    = LocalFrameSize + RegisterStorageSize + 56  
; Locals (on local stack frame)  
CCOCursor       = 0  
CCOSkipDistance = 4  
ChromaLineLen  = 8  
R3G3B3R2       = 12  
G2B2R1G1       = 16  
AspectCount     = 20  
LCL EQU <esp+>  
MMXCODE1 SEGMENT  
; extern void "C" MMX_YUV12ToRGB24ZoomBy2 (U8 * YPlane,  
;                                       U8 * UPlane,  
;                                       U8 * VPlane,  
;                                       UN  FrameWidth,
```


Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
;
; ebp -- Y Line cursor.  Chroma contribs go in lines above current Y line.
; esi -- V
; edx -- U
; edi -- Cursor over the color converted output image.
; ebx -- Number of points, we havn't done yet.
;
;
; ecx -- 3*CCOPitch
; eax -- CCOPitch.
;-----
PrepareChromaLine:
    Lebp    AspectCount
    Leax   CCOPitch
    sub    ebp,4
    Lebx   FrameWidth
    lea   ecx,[eax+2*eax] ; pointer to fourth output line
    ja    continue
    lea   ecx,[2*eax]
    ADDebp AspectAdjustmentCount
continue:
    Sebp   AspectCount
align 16
do_next_8x2_block:
; ; ; ; ; ; trsansformation U, V
    movdt   mm1, [edx+ebx]      ; 4 u values
    pxor    mm0,mm0           ; mm0=0
    movdt   mm2, [esi+ebx]      ; 4 v values
    punpcklbw mm1,mm0         ; get 4 unsign u
    psubw   mm1,Minusg        ; get 4 unsign u-128
    punpcklbw mm2,mm0         ; get unsign v
    psubw   mm2,Minusg        ; get unsign v-128
    movq    mm3,mm1           ; save the u unsign
    mov    ebp,tmpYCursorEven
    punpcklwd mm1,mm2         ; get 2 low u,v unsign pairs
    pmaddwd mm1,UVtG
    movq    mm5,mm3           ; save u-128
    movq    mm6,[ebp+2*ebx]    ; mm6 has 8 y pixels
    punpckhwd mm3,mm2         ; create high 2 unsign uv pairs
    pmaddwd mm3,UVtG
    psubusb mm6,Yadd          ; mm6 has 8 y-16 pixels
    packssdw mm1,mm3         ; packed the results to signed words
    movq    mm7,mm6          ; save the 8 y-16 pixels
    punpcklbw mm6,mm0         ; mm6 has 4 low y-16 unsign
    pmullw  mm6,Ymul
    punpckhbw mm7,mm0         ; mm7 has 4 high y-16 unsign
    pmullw  mm7,Ymul
    movq    mm4,mm1
    movq    PD [tmpBuffer],mm1 ; save 4 chroma G values
    punpcklwd mm1,mm1         ; chroma G replicate low 2
    movq    mm0,mm6          ; low y
    movq    mm3,mm7          ; high y
    punpckhwd mm4,mm4         ; chroma G replicate high 2
    psubw   mm6,mm1          ; 4 low G
    movq    mm1, mm5         ; 4 u values
    psraw  mm6,6             ; low G

    psubw   mm7,mm4          ; 4 high G values in signed 16 bit
    punpcklwd mm1,mm1         ; replicate the 2 low u pixels
    pmullw  mm1,UtB
    punpckhwd mm5,mm5
    pmullw  mm5,UtB
    psraw  mm7,6             ; high G
    movq    PD [tmpBuffer+8],mm1 ; low chroma B
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
packuswb mm6,mm7          ; mm6: G7 G6 G5 G4 G3 G2 G1 G0
movq      PD [tmpBuffer+16],mm5 ; high chroma B
paddw    mm5,mm3          ; 4 high B values in signed 16 bit
paddw    mm1,mm0          ; 4 low B values in signed 16 bit
psraw    mm5,6           ; high B
movq     mm7, mm2
punpcklwd mm2,mm2        ; replicate the 2 low v pixels
psraw    mm1,6           ; low B
pmullw   mm2,VtR
punpckhwd mm7,mm7
pmullw   mm7,VtR
packuswb mm1,mm5          ; mm1: B7 B6 B5 B4 B3 B2 B1 B0
movq     PD [tmpBuffer+24],mm2 ; low chroma R
paddw    mm2,mm0          ; 4 low R values in signed 16 bit
psraw    mm2,6           ; low R
movq     PD [tmpBuffer+32],mm7 ; high chroma R
paddw    mm7,mm3          ; 4 high R values in signed 16 bit
psraw    mm7,6           ; high R
movq     PD [tmpBuffer+40],mm1 ; save B in memory
packuswb mm2,mm7          ; mm2: R7 R6 R5 R4 R3 R2 R1 R0
movq     mm3,mm6          ; save G in mm3
punpcklbw mm1,mm1        ; mm1: B3 B3 B2 B2 B1 B1 B0 B0
movq     mm0,mm1
punpcklwd mm1,mm1        ; mm1: B1 B1 B1 B1 B0 B0 B0 B0
pand     mm1,MASK_036    ; mm1: 0 B1 0 0 B0 0 0 B0
punpcklbw mm6,mm6        ; mm6: G3 G3 G2 G2 G1 G1 G0 G0
movq     mm5,mm6
punpcklwd mm6,mm6        ; mm6: G1 G1 G1 G1 G0 G0 G0 G0
movq     mm4,mm2          ; save R in mm4
punpcklbw mm2,mm2        ; mm2: R3 R3 R2 R2 R1 R1 R0 R0
pand     mm6,MASK_036    ; mm6: 0 G1 0 0 G0 0 0 G0
movq     mm7,mm2
punpcklwd mm2,mm2        ; mm2: R1 R1 R1 R1 R0 R0 R0 R0
pand     mm2,MASK_036    ; mm2: 0 R1 0 0 R0 0 0 R0
psllq   mm6,8            ; mm6: G1 0 0 G0 0 0 G0 0
psllq   mm2,16           ; mm2: 0 0 R0 0 0 R0 0 0
por     mm1,mm6
por     mm2,mm1          ; mm2: G1 B1 R0 G0 B0 R0 G0 B0
movq     mm1,mm0        ; mm1: B3 B3 B2 B2 B1 B1 B0 B0
movq     PD [edi],mm2   ; store result
psrlq   mm1,24          ; mm1: 0 0 0 B3 B3 B2 B2 B1
movq     PD [edi+eax],mm2 ; store result
punpcklwd mm1,mm1        ; mm1: B3 B2 B3 B2 B2 B1 B2 B1
;; 2nd phase
pand     mm1,MASK_036    ; mm1: 0 B2 0 0 B2 0 0 B1
movq     mm6,mm5        ; mm6: G3 G3 G2 G2 G1 G1 G0 G0
psllq   mm1,8           ; mm1: B2 0 0 B2 0 0 B1 0
psrlq   mm6,16          ; mm6: 0 0 G3 G3 G2 G2 G1 G1
movq     mm2,mm7
pand     mm6,MASK_036    ; mm6: 0 G2 0 0 G2 0 0 G1
psrlq   mm2,16          ; mm2: 0 0 R3 R3 R2 R2 R1 R1
psllq   mm6,16          ; mm6: 0 0 G2 0 0 G1 0 0
punpcklwd mm2,mm2        ; mm2: R2 R2 R2 R2 R1 R1 R1 R1
por     mm1,mm6
pand     mm2,MASK_036    ; mm2: 0 R2 0 0 R1 0 0 R1
movq     mm6,mm5        ; mm6: G3 G3 G2 G2 G1 G1 G0 G0
por     mm2,mm1          ; mm2: B2 R2 G2 B2 R1 G1 B1 R1
movq     mm1,mm0        ; mm1: B3 B3 B2 B2 B1 B1 B0 B0
movq     PD [edi+8],mm2 ; store result
psrlq   mm1,48          ; mm1: 0 0 0 0 0 0 B3 B3
movq     PD [edi+eax+8],mm2 ; store result
punpcklwd mm1,mm1        ; mm1: 0 0 0 0 B3 B3 B3 B3
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```

;; 3rd phase
    pand                mm1, MASK_036      ; mm1:  0  0  0  0 B3  0  0 B3
    psrlq               mm6, 40           ; mm6:  0  0  0  0  0 G3 G3 G2
    punpcklwd           mm6, mm6          ; mm6:  0 G3  0 G3 G3 G2 G3 G2
    movq                mm2, mm7
    pand                mm6, MASK_036      ; mm6:  0 G3  0  0 G3  0  0 G2
    psllq               mm1, 16           ; mm1:  0  0 B3  0  0 B3  0  0
    punpckhwd           mm2, mm2          ; mm2:  R3 R3 R3 R3 R2  0 R2  0
    por                 mm1, mm6
    pand                mm2, MASK_147      ; mm2:  R3  0  0 R3  0  0 R2  0
    movq                mm6, mm3          ; restore mm6 with G
    por                 mm2, mm1          ; mm2:  R3 G3 B3 R3 G3 B3 R2 G2
    movq                mm1, PD [tmpBuffer+40] ; restore mm1 with B
    movq                PD [edi+16], mm2   ; store result
    punpckhbw           mm1, mm1          ; mm1:  B7 B7 B6 B6 B5 B5 B4 B4

    movq                PD [edi+eax+16], mm2 ; store result
    movq                mm2, mm4          ; restore mm2 with R

;; 4th phase
    movq                mm0, mm1
    punpckhbw           mm6, mm6          ; mm6:  G7 G7 G6 G6 G5 G5 G4 G4
    punpcklwd           mm1, mm1          ; mm1:  B5 B5 B5 B5 B4 B4 B4 B4
    movq                mm5, mm6
    pand                mm1, MASK_036      ; mm1:  0 B5  0  0 B4  0  0 B4
    punpcklwd           mm6, mm6          ; mm6:  G5 G5 G5 G5 G4 G4 G4 G4
    pand                mm6, MASK_036      ; mm6:  0 G5  0  0 G4  0  0 G4
    punpckhbw           mm2, mm2          ; mm2:  R7 R7 R6 R6 R5 R5 R4 R4
    psllq               mm6, 8            ; mm6:  G5  0  0 G4  0  0 G4  0
    movq                mm7, mm2

    punpcklwd           mm2, mm2          ; mm2:  R5 R5 R5 R5 R4 R4 R4 R4
    pand                mm2, MASK_036      ; mm2:  0 R5  0  0 R4  0  0 R4
    psllq               mm2, 16           ; mm2:  0  0 R4  0  0 R4  0  0
    por                 mm1, mm6
    por                 mm2, mm1          ; mm2:  G5 B5 R4 G4 B4 R4 G4 B4
    movq                mm1, mm0          ; mm1:  B7 B7 B6 B6 B5 B5 B4 B4
    movq                PD [edi+24], mm2   ; store result
    psrlq               mm1, 24           ; mm1:  0  0  0 B7 B7 B6 B6 B5
    movq                PD [edi+eax+24], mm2 ; store result
    punpcklwd           mm1, mm1          ; mm1:  B7 B6 B7 B6 B6 B5 B6 B5

;; 5th phase
    pand                mm1, MASK_036      ; mm1:  0 B6  0  0 B6  0  0 B5
    movq                mm6, mm5
    psllq               mm1, 8            ; mm6:  G7 G7 G6 G6 G5 G5 G4 G4
    movq                mm2, mm7          ; mm1:  B6  0  0 B6  0  0 B5  0
    psrlq               mm6, 24           ; mm6:  0  0  0 G7 G7 G6 G6 G5
    psrlq               mm2, 16           ; mm2:  0  0 R7 R7 R6 R6 R5 R5
    punpcklwd           mm6, mm6          ; mm6:  G7 G6 G7 G6 G6 G5 G6 G5
    pand                mm6, MASK_036      ; mm6:  0 G6  0  0 G6  0  0 G5
    punpcklwd           mm2, mm2          ; mm2:  R6 R6 R6 R6 R5 R5 R5 R5
    pand                mm2, MASK_036      ; mm2:  0 R6  0  0 R5  0  0 R5
    psllq               mm6, 16           ; mm6:  0  0 G6  0  0 G5  0  0

;>>>>
    por                 mm2, mm6
    por                 mm2, mm1          ; mm2:  B6 R6 G6 B6 R5 G5 B5 R5
    movq                mm1, mm0          ; mm1:  B7 B7 B6 B6 B5 B5 B4 B4
    psrlq               mm1, 48           ; mm1:  0  0  0  0  0  0 B7 B7
    movq                mm6, mm5          ; mm6:  G7 G7 G6 G6 G5 G5 G4 G4
    movq                PD [edi+32], mm2   ; store result
    punpcklwd           mm1, mm1          ; mm1:  0  0  0  0 B7 B7 B7 B7
    movq                PD [edi+eax+32], mm2 ; store result
    psrlq               mm6, 40           ; mm6:  0  0  0  0  0 G7 G7 G6

;; 6th phase
    pand                mm1, MASK_036      ; mm1:  0  0  0  0 B7  0  0 B7

```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```

        punpcklwd    mm6,mm6                ; mm6:  0 G7  0 G7  G7 G6  G7 G6
        pand        mm6,MASK_036          ; mm6:  0 G7  0  0 G7  0  0 G6
        psllq       mm1,16                ; mm1:  0  0 B7  0  0 B7  0  0
        movq        mm2,mm7
        por                    mm1,mm6
mov    ebp,tmpYCursorOdd
        punpckhwd    mm2,mm2                ; mm2:  0 R7  0 R7  R7 R6  R7 R6
        pand        mm2,MASK_147          ; mm2:  0 R7  0  0 R7  0  0 R6
;
        lea        ecx,[eax+2*eax]
        por                    mm2,mm1          ; mm2:  R7 G7  B7 R7  G7 B7  R6 G6
;- start odd line
        movq        mm1,[ebp+2*ebx]          ; mm1 has 8 y pixels
        pxor        mm0, mm0
        psubusb     mm1,Yadd                ; mm1 has 8 pixels y-16
        movq        mm5,mm1
        punpcklbw   mm1,mm0                ; get 4 low y-16 unsign pixels word
        pmullw      mm1,Ymul                ; low 4 luminance contribution
        punpckhbw   mm5,mm0                ; 4 high y-16
        pmullw      mm5,Ymul                ; high 4 luminance contribution
        movq        PD [edi+40],mm2          ; store result
        movq        PD [edi+eax+40],mm2      ; store result
        movq        mm2,mm1
        paddw       mm2,PD [tmpBuffer+24]    ; low 4 R
        movq        mm6,mm5
        paddw       mm5,PD [tmpBuffer+32]    ; high 4 R
        psraw       mm2,6
        psraw       mm5,6
        packuswb   mm2,mm5                ; mm0:  R7 R6  R5 R4  R3 R2  R1 R0
        movq        mm0,mm1
        paddw       mm0,PD [tmpBuffer+8]     ; low 4 B
        movq        mm5,mm6
        paddw       mm5,PD [tmpBuffer+16]    ; high 4 B
        psraw       mm0,6
        movq        mm3,PD [tmpBuffer]      ; chroma G low 4
        psraw       mm5,6
        packuswb   mm0,mm5                ; mm2:  B7 B6  B5 B4  B3 B2  B1 B0
        movq        mm4,mm3
        punpcklwd   mm3,mm3                ; replicate low 2
        punpckhwd   mm4,mm4                ; replicate high 2
        psubw       mm1,mm3                ; 4 low G
        psubw       mm6,mm4                ; 4 high G values in signed 16 bit
        psraw       mm1,6                ; low G
        movq        PD [tmpBuffer+40],mm0    ; save B in memory
        psraw       mm6,6                ; high G
        packuswb   mm1,mm6                ; mm1:  G7 G6  G5 G4  G3 G2  G1 G0
        movq        mm4,mm2                ; save R in mm4
        movq        mm6,mm1
        movq        mm1,mm0
        movq        mm3,mm6                ; save G in mm3
        punpcklbw   mm1,mm1                ; mm1:  B3 B3  B2 B2  B1 B1  B0 B0
        movq        mm0,mm1
        punpcklwd   mm1,mm1                ; mm1:  B1 B1  B1 B1  B0 B0  B0 B0
        pand        mm1,MASK_036          ; mm1:  0 B1  0  0 B0  0  0 B0
        punpcklbw   mm6,mm6                ; mm6:  G3 G3  G2 G2  G1 G1  G0 G0
        movq        mm5,mm6
        punpcklwd   mm6,mm6                ; mm6:  G1 G1  G1 G1  G0 G0  G0 G0
        pand        mm6,MASK_036          ; mm6:  0 G1  0  0 G0  0  0 G0
        punpcklbw   mm2,mm2                ; mm2:  R3 R3  R2 R2  R1 R1  R0 R0
        psllq       mm6,8                ; mm6:  G1  0  0 G0  0  0 G0  0
        movq        mm7,mm2
        punpcklwd   mm2,mm2                ; mm2:  R1 R1  R1 R1  R0 R0  R0 R0
        por                    mm1,mm6
        pand        mm2,MASK_036          ; mm2:  0 R1  0  0 R0  0  0 R0
        movq        mm6,mm5                ; mm6:  G3 G3  G2 G2  G1 G1  G0 G0
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```

        psllq          mm2,16          ; mm2:  0  0 R0  0  0 R0  0  0
        por           mm2,mm1         ; mm2:  G1 B1 R0 G0 B0 R0 G0 B0
        psrlq          mm6,24         ; mm6:  0  0  0 G3 G3 G2 G2 G1
movq     PD [edi+ecx],mm2          ; store result
        movq          mm1,mm0         ; mm1:  B3 B3 B2 B2 B1 B1 B0 B0
        movq          PD [edi+2*eax],mm2 ; store result
        psrlq          mm1,24         ; mm1:  0  0  0 B3 B3 B2 B2 B1
;; 2nd phase
        punpcklwd     mm1,mm1         ; mm1:  B3 B2 B3 B2 B2 B1 B2 B1
        movq          mm2,mm7
        pand          mm1,MASK_036    ; mm1:  0 B2  0  0 B2  0  0 B1
        punpcklwd     mm6,mm6         ; mm6:  G3 G2 G3 G2 G2 G1 G2 G1
        pand          mm6,MASK_036    ; mm6:  0 G2  0  0 G2  0  0 G1
        psllq          mm1,8          ; mm1:  B2  0  0 B2  0  0 B1  0
        psllq          mm6,16         ; mm6:  0  0 G2  0  0 G1  0  0
        psrlq          mm2,16         ; mm2:  0  0 R3 R3 R2 R2 R1 R1
        por           mm1,mm6
        punpcklwd     mm2,mm2         ; mm2:  R2 R2 R2 R2 R1 R1 R1 R1
        movq          mm6,mm5         ; mm6:  G3 G3 G2 G2 G1 G1 G0 G0
        pand          mm2,MASK_036    ; mm2:  0 R2  0  0 R1  0  0 R1
        psrlq          mm6,40         ; mm6:  0  0  0  0  0  0 G3 G3 G2
        por           mm2,mm1         ; mm2:  B2 R2 G2 B2 R1 G1 B1 R1
        movq          mm1,mm0         ; mm1:  B3 B3 B2 B2 B1 B1 B0 B0
movq     PD [edi+ecx+8],mm2          ; store result
        punpckhwd     mm1,mm1         ; mm1:  B3 B3 B3 B3 0  0  0  0
        movq          PD [edi+2*eax+8],mm2 ; store result
        punpcklwd     mm6,mm6         ; mm6:  0 G3  0 G3 G3 G2 G3 G2
;; 3rd phase
        pand          mm1,MASK_147    ; mm1:  0  0  0  0 B3  0  0 B3
        movq          mm2,mm7
        psrlq          mm1,16         ; mm1:  0  0 B3  0  0 B3  0  0
        pand          mm6,MASK_036    ; mm6:  0 G3  0  0 G3  0  0 G2
        psrlq          mm2,40         ; mm2:  0  0  0  0  0 R3 R3 R2
        punpcklwd     mm2,mm2         ; mm2:  0 R3  0 R3 R3 R2 R3 R2
        por           mm1,mm6
        pand          mm2,MASK_036    ; mm2:  0 R3  0  0 R3  0  0 R2
        movq          mm6,mm3         ; restore mm6 with G
        psllq          mm2,8          ; mm2:  R3  0  0 R3  0  0 R2  0
        por           mm2,mm1         ; mm2:  R3 G3 B3 R3 G3 B3 R2 G2
movq     PD [edi+ecx+16],mm2          ; restore mm1 with B
        movq          mm1,PD [tmpBuffer+40] ; store result
        psrlq          mm1,32         ; 0  0  0  0 B7 B6 B5 B4
        movq          PD [edi+2*eax+16],mm2 ; store result
        psrlq          mm6,32         ; 0  0  0  0 G7 G6 G5 G4
        movq          mm2,mm4         ; restore mm2 with R
        punpcklbw     mm1,mm1         ; mm1:  B7 B7 B6 B6 B5 B5 B4 B4
;; 4th phase
        psrlq          mm2,32         ; 0  0  0  0 R7 R6 R5 R4
        movq          mm0,mm1
        punpcklwd     mm1,mm1         ; mm1:  B5 B5 B5 B5 B4 B4 B4 B4
        pand          mm1,MASK_036    ; mm1:  0 B5  0  0 B4  0  0 B4
        punpcklbw     mm6,mm6         ; mm6:  G7 G7 G6 G6 G5 G5 G4 G4
        punpcklbw     mm2,mm2         ; mm2:  R7 R7 R6 R6 R5 R5 R4 R4
        movq          mm5,mm6
        punpcklwd     mm6,mm6         ; mm6:  G5 G5 G5 G5 G4 G4 G4 G4
        movq          mm7,mm2
        pand          mm6,MASK_036    ; mm6:  0 G5  0  0 G4  0  0 G4
        punpcklwd     mm2,mm2         ; mm2:  R5 R5 R5 R5 R4 R4 R4 R4
        pand          mm2,MASK_036    ; mm2:  0 R5  0  0 R4  0  0 R4
        psllq          mm6,8          ; mm6:  G5  0  0 G4  0  0 G4  0
        psllq          mm2,16         ; mm2:  0  0 R4  0  0 R4  0  0
        por           mm1,mm6
    
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```

        por                mm2,mm1                ; mm2: G5 B5 R4 G4 B4 R4 G4 B4
        movq               mm1,mm0                ; mm1: B7 B7 B6 B6 B5 B5 B4 B4
        psrlq              mm1,24                ; mm1: 0 0 0 B7 B7 B6 B6 B5
        movq               mm6,mm5                ; mm6: G7 G7 G6 G6 G5 G5 G4 G4
        movq               PD [edi+ecx+24],mm2    ; store result
        punpcklwd          mm1,mm1                ; mm1: B7 B6 B7 B6 B6 B5 B6 B5

        movq               PD [edi+2*eax+24],mm2    ; store result
        psrlq              mm6,24                ; mm6: 0 0 0 G7 G7 G6 G6 G5
;; 5th phase
        pand               mm1,MASK_036          ; mm1: 0 B6 0 0 B6 0 0 B5
        punpcklwd          mm6,mm6                ; mm6: G7 G6 G7 G6 G6 G5 G6 G5
        pand               mm6,MASK_036          ; mm6: 0 G6 0 0 G6 0 0 G5
        psllq              mm1,8                 ; mm1: B6 0 0 B6 0 0 B5 0
        psllq              mm6,16                ; mm6: 0 0 G6 0 0 G5 0 0
        movq               mm2,mm7
        psrlq              mm2,16                ; mm2: 0 0 R7 R7 R6 R6 R5 R5
        por                mm1,mm6
        punpcklwd          mm2,mm2                ; mm2: R6 R6 R6 R6 R5 R5 R5 R5
        movq               mm6,mm5                ; mm6: G7 G7 G6 G6 G5 G5 G4 G4
        pand               mm2,MASK_036          ; mm2: 0 R6 0 0 R5 0 0 R5
        psrlq              mm6,40                ; mm6: 0 0 0 0 0 0 G7 G7 G6
        por                mm2,mm1                ; mm2: B6 R6 G6 B6 R5 G5 B5 R5
        punpcklwd          mm6,mm6                ; mm6: 0 G7 0 G7 G7 G6 G7 G6
        pand               mm6,MASK_036          ; mm6: 0 G7 0 0 G7 0 0 G6
        movq               mm1,mm0                ; mm1: B7 B7 B6 B6 B5 B5 B4 B4
        movq               PD [edi+ecx+32],mm2    ; store result
        punpckhwd          mm1,mm1                ; mm1: B7 B7 B7 B7 0 0 0 0
        movq               PD [edi+2*eax+32],mm2    ; store result
        movq               mm2,mm7
;; 6th phase
        pand               mm1,MASK_147          ; mm1: B7 0 0 B7 0 0 0 0
        psrlq              mm2,40                ; mm2: 0 0 0 0 0 R7 R7 R6
        punpcklwd          mm2,mm2                ; mm2: 0 R7 0 R7 R7 R6 R7 R6
        pand               mm2,MASK_036          ; mm2: 0 R7 0 0 R7 0 0 R6
        psrlq              mm1,16                ; mm1: 0 0 B7 0 0 B7 0 0
        psllq              mm2,8                 ; mm2: R7 0 0 R7 0 0 R6 0
        por                mm1,mm6
        por                mm2,mm1                ; mm2: R7 G7 B7 R7 G7 B7 R6 G6
        movq               PD [edi+ecx+40],mm2    ; store result
        movq               PD [edi+2*eax+40],mm2    ; store result
add     edi,48 ; ih take 48 instead of 12 output
add     ebx,4 ; ? to take 4 pixels together instead of 2
jl      do_next_8x2_block ; ? update the loop for 8 y pixels at once
ADDdedi CCOSkipDistance
add     edi,ecx ; set output pointer after fourth line
Leax   YPitch
mov     ebp,tmpYCursorOdd
lea    ebp,[ebp+2*eax] ; skip two lines
mov     tmpYCursorOdd,ebp
mov     ebp,tmpYCursorEven
lea    ebp,[ebp+2*eax]
mov     tmpYCursorEven,ebp
ADDdesi ChromaPitch
ADDdedx ChromaPitch
sub     PD FrameHeight[esp],4
ja      PrepareChromaLine
;-----
finish:
add     esp,LocalFrameSize
emms
pop     ebx
pop     ebp
pop     edi

```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    pop    esi
    ret
MMXCODE1 ENDS
END
```

17. Appendix 6. Color Conversion to RGB16.

```
-----  
; cxml2161 -- This function performs YUV12-to-RGB16 color conversion for H26x.  
; It handles any format in which there are three fields, the low  
; order field being B and fully contained in the low order byte, the  
; second field being G and being somewhere in bits 4 through 11,  
; and the high order field being R and fully contained in the high  
; order byte.  
;  
; The YUV12 input is planar, 8 bits per pel. The Y plane may have  
; a pitch of up to 768. It may have a width less than or equal  
; to the pitch. It must be DWORD aligned, and preferably QWORD  
; aligned. Pitch and Width must be a multiple of four. For best  
; performance, Pitch should not be 4 more than a multiple of 32.  
; Height may be any amount, but must be a multiple of two. The U  
; and V planes may have a different pitch than the Y plane, subject  
; to the same limitations.  
;  
include iammx.inc  
include locals.inc  
.586  
.xlist  
.list  
  ASSUME ds:FLAT, cs:FLAT, ss:FLAT  
MMXCODE1 SEGMENT PARA USE32 PUBLIC 'CODE'  
MMXCODE1 ENDS  
MMXDATA1 SEGMENT PARA USE32 PUBLIC 'DATA'  
MMXDATA1 ENDS  
MMXDATA1 SEGMENT  
ALIGN 8  
RGB_formats:  
  dd RGB565  
  dd RGB555  
  dd RGB664  
  dd RGB655  
Minusg          dd  00800080h, 00800080h  
Yadd            dd  10101010h, 10101010h  
VtR            dd  00660066h, 00660066h ;01990199h,01990199h  
VtG            dd  00340034h, 00340034h ;00d000d0h,00d000d0h  
UtG            dd  00190019h, 00190019h ;00640064h,00640064h  
UtB            dd  00810081h, 00810081h ;02050205h,02050205h  
Ymul           dd  004a004ah, 004a004ah ;012a012ah,012a012ah  
UVtG          dd  00340019h, 00340019h ;00d00064h,00d00064h  
VtRtB         dd  01990205h, 01990205h  
fourbitu      dd  0f0f0f0f0h, 0f0f0f0f0h  
fivebitu      dd  0e0e0e0e0h, 0e0e0e0e0h  
sixbitu       dd  0c0c0c0c0h, 0c0c0c0c0h  
MMXDATA1 ENDS  
LocalFrameSize = 156  
RegisterStorageSize = 16  
; Arguments:  
YPlane        = LocalFrameSize + RegisterStorageSize + 4  
UPlane        = LocalFrameSize + RegisterStorageSize + 8  
VPlane        = LocalFrameSize + RegisterStorageSize + 12  
FrameWidth    = LocalFrameSize + RegisterStorageSize + 16  
FrameHeight   = LocalFrameSize + RegisterStorageSize + 20  
YPitch        = LocalFrameSize + RegisterStorageSize + 24  
ChromaPitch   = LocalFrameSize + RegisterStorageSize + 28  
AspectAdjustmentCount = LocalFrameSize + RegisterStorageSize + 32  
ColorConvertedFrame = LocalFrameSize + RegisterStorageSize + 36
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
DCIOffset          = LocalFrameSize + RegisterStorageSize + 40
CCOffsetToLine0    = LocalFrameSize + RegisterStorageSize + 44
CCOPitch           = LocalFrameSize + RegisterStorageSize + 48
CCType             = LocalFrameSize + RegisterStorageSize + 52
EndOfArgList       = LocalFrameSize + RegisterStorageSize + 56
; Locals (on local stack frame)
CCOCursor          = 0
CCOSkipDistance    = 4
ChromaLineLen     = 8
YCursor           = 12
DistanceFromVToU  = 16
EndOfChromaLine   = 20
AspectCount       = 24
AspectBaseCount   = 28
tmpYCursorEven    = 32
tmpYCursorOdd     = 36
tmpCCOPitch       = 40
temp_mmx          = 44 ; note it is 48 bytes
RLeftShift        = 92
GLeftShift        = 100
RRightShift       = 108
GRightShift       = 116
BRightShift       = 124
RUpperLimit       = 132
GUpperLimit       = 140
BUpperLimit       = 148
MMXCODE1 SEGMENT
; extern void "C" MMX_YUV12ToRGB16 (
;
;           U8* YPlane,
;           U8* UPlane,
;           U8* VPlane,
;           UN  FrameWidth,
;           UN  FrameHeight,
;           UN  YPitch,
;           UN  VPitch,
;           UN  AspectAdjustmentCount,
;           U8* ColorConvertedFrame,
;           U32 DCIOffset,
;           U32 CCOffsetToLine0,
;           IN  CCOPitch,
;           IN  CCType)
;
; The local variables are on the stack,
; The tables are in the one and only data segment.
;
; CCOffsetToLine0 is relative to ColorConvertedFrame.
; CCType used by RGB color convertors to determine the exact conversion type.
;   RGB565 = 0
;   RGB555 = 1
;   RGB664 = 2
;   RGB655 = 3
PUBLIC C MMX_YUV12ToRGB16
MMX_YUV12ToRGB16:
    push    esi
    push    edi
    push    ebp
    push    ebx
    sub     esp, LocalFrameSize
    mov     eax, [esp+CCType]
    cmp     eax, 4
    jae    finish
    jmp     RGB_formats[eax*4]
RGB555:
    xor     eax, eax
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
mov     ebx, 2    ; 10-8 for byte shift
mov     [esp+RLeftShift], ebx
mov     [esp+RLeftShift+4], eax
mov     ebx, 5
mov     [esp+GLeftShift], ebx
mov     [esp+GLeftShift+4], eax
mov     ebx, 9
mov     [esp+RRightShift], ebx
mov     [esp+RRightShift+4], eax
mov     [esp+GRightShift], ebx
mov     [esp+GRightShift+4], eax
mov     [esp+BRightShift], ebx
mov     [esp+BRightShift+4], eax
movq    mm0, fivebitu
movq    [esp+RUpperLimit], mm0
movq    [esp+GUpperLimit], mm0
movq    [esp+BUpperLimit], mm0
jmp     RGBEND
RGB664:
xor     eax, eax
mov     ebx, 2    ; 8-6
mov     [esp+RLeftShift], ebx
mov     [esp+RLeftShift+4], eax
mov     ebx, 4
mov     [esp+GLeftShift], ebx
mov     [esp+GLeftShift+4], eax
mov     ebx, 8
mov     [esp+RRightShift], ebx
mov     [esp+RRightShift+4], eax
mov     [esp+GRightShift], ebx
mov     [esp+GRightShift+4], eax
mov     ebx, 10
mov     [esp+BRightShift], ebx
mov     [esp+BRightShift+4], eax
movq    mm0, sixbitu
movq    [esp+RUpperLimit], mm0
movq    [esp+GUpperLimit], mm0
movq    mm0, fourbitu
movq    [esp+BUpperLimit], mm0
jmp     RGBEND
RGB655:
xor     eax, eax
mov     ebx, 2    ; 8-6
mov     [esp+RLeftShift], ebx
mov     [esp+RLeftShift+4], eax
mov     ebx, 5
mov     [esp+GLeftShift], ebx
mov     [esp+GLeftShift+4], eax
mov     ebx, 8
mov     [esp+RRightShift], ebx
mov     [esp+RRightShift+4], eax
mov     ebx, 9
mov     [esp+GRightShift], ebx
mov     [esp+GRightShift+4], eax
mov     [esp+BRightShift], ebx
mov     [esp+BRightShift+4], eax
movq    mm0, sixbitu
movq    [esp+RUpperLimit], mm0
movq    mm0, fivebitu
movq    [esp+GUpperLimit], mm0
movq    [esp+BUpperLimit], mm0
jmp     RGBEND
RGB565:
xor     eax, eax
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
mov     ebx, 3      ; 8-5
mov     [esp+RLeftShift], ebx
mov     [esp+RLeftShift+4], eax
mov     ebx, 5
mov     [esp+GLeftShift], ebx
mov     [esp+GLeftShift+4], eax
mov     ebx, 9
mov     [esp+RRightShift], ebx
mov     [esp+RRightShift+4], eax
mov     [esp+BRightShift], ebx
mov     [esp+BRightShift+4], eax
mov     ebx, 8
mov     [esp+GRightShift], ebx
mov     [esp+GRightShift+4], eax
movq    mm0, fivebitu
movq    [esp+RUpperLimit], mm0
movq    [esp+BUpperLimit], mm0
movq    mm0, sixbitu
movq    [esp+GUpperLimit], mm0
; jmp   RGBEND
RGBEND:
mov     ebx, [esp+VPlane]
mov     ecx, [esp+UPlane]
sub     ecx, ebx
mov     [esp+DistanceFromVToU], ecx
mov     eax, [esp+ColorConvertedFrame]
add     eax, [esp+DCIOffset]
add     eax, [esp+CCOffsetToLine0]
mov     [esp+CCOCursor], eax
Lecx   YPitch
Lebx   FrameWidth
Leax   CCOPitch
sub     eax, ebx      ; CCOPitch-FrameWidth
sub     eax, ebx      ; CCOPitch-2*FrameWidth
sar     ebx, 1        ; FrameWidth/2
Lesi   YPlane        ; Fetch cursor over luma plane.
Sebx   ChromaLineLen ; FrameWidth/2
Seax   CCOSkipDistance ; CCOPitch-3*FrameWidth
Sesi   YCursor
Ledx   AspectAdjustmentCount
Lesi   VPlane
cmp     edx, 1
je     finish
Sedx   AspectCount
Sedx   AspectBaseCount
xor     eax, eax
Ledi   ChromaLineLen
Sedi   EndOfChromaLine
Ledi   CCOCursor
Ledx   DistanceFromVToU
Lebp   YCursor        ; Fetch Y Pitch.
Lebx   FrameWidth
add     ebp, ebx
Sebp   tmpYCursorEven
Leax   YPitch
add     ebp, eax
Sebp   tmpYCursorOdd
sar     ebx, 1
add     esi, ebx
add     edx, esi
neg     ebx
Sebx   FrameWidth
; Register Usage:
;
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
-----  
PrepareChromaLine:  
  Lebp  AspectCount  
  Lebx  FrameWidth  
  sub   ebp,2  
  Leax  CCOPitch  
  Seax  tmpCCOPitch  
  ja    continue  
  xor   eax,eax  
  ADDebp AspectAdjustmentCount  
  Seax  tmpCCOPitch  
continue:  
  Sebp  AspectCount  
do_next_8x2_block:  
  Lebp  tmpYCursorEven  
; here is even line  
movdtd mm1, [edx+ebx]      ; 4 u values  
  pxor  mm0, mm0          ; mm0=0  
movdtd mm2, [esi+ebx]      ; 4 v values  
  punpcklbw mm1, mm0      ; get 4 unsign u  
  psubw   mm1, Minusg      ; get 4 unsign u-128  
  punpcklbw mm2, mm0      ; get unsign v  
  psubw   mm2, Minusg      ; get unsign v-128  
  movq    mm3, mm1         ; save the u-128 unsign  
  movq    mm5, mm1         ; save u-128 unsign  
  punpcklwd mm1, mm2      ; get 2 low u, v unsign pairs  
  pmaddwd mm1, UVtG  
  punpckhwd mm3, mm2      ; create high 2 unsign uv pairs  
  pmaddwd mm3, UVtG  
  movq    temp_mmx[esp], mm2 ; save v-128  
  movq    mm6, [ebp+2*ebx]   ; mm6 has 8 y pixels  
  psubusb mm6, Yadd         ; mm6 has 8 y-16 pixels  
  packssdw mm1, mm3         ; packed the results to signed words  
  movq    mm7, mm6         ; save the 8 y-16 pixels  
  punpcklbw mm6, mm0      ; mm6 has 4 low y-16 unsign  
  pmullw   mm6, Ymul  
  punpckhbw mm7, mm0      ; mm7 has 4 high y-16 unsign  
  pmullw   mm7, Ymul  
  movq    mm4, mm1  
  movq    temp_mmx[esp+8], mm1 ; save 4 chroma G values  
  punpcklwd mm1, mm1      ; chroma G replicate low 2  
  movq    mm0, mm6         ; low y  
  punpckhwd mm4, mm4      ; chroma G replicate high 2  
  movq    mm3, mm7         ; high y  
  psubw   mm6, mm1         ; 4 low G  
  psraw   mm6, [esp+GRightShift] ; 4 high G values in signed 16 bit  
  psubw   mm7, mm4  
  movq    mm2, mm5  
  punpcklwd mm5, mm5      ; replicate the 2 low u pixels  
  pmullw   mm5, UtB  
  punpckhwd mm2, mm2  
  psraw   mm7, [esp+GRightShift]  
  pmullw   mm2, UtB  
  packuswb mm6, mm7      ; mm6: G7 G6 G5 G4 G3 G2 G1 G0  
  movq    temp_mmx[esp+16], mm5 ; low chroma B  
  paddw   mm5, mm0        ; 4 low B values in signed 16 bit  
  movq    temp_mmx[esp+40], mm2 ; high chroma B  
  paddw   mm2, mm3        ; 4 high B values in signed 16 bit  
  psraw   mm5, [esp+BRightShift] ; low B scaled down by 6+(8-5)  
  psraw   mm2, [esp+BRightShift] ; high B scaled down by 6+(8-5)  
  packuswb mm5, mm2      ; mm5: B7 B6 B5 B4 B3 B2 B1 B0  
  movq    mm2, temp_mmx[esp] ; 4 v values  
  movq    mm1, mm5        ; save B  
  movq    mm7, mm2
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    punpcklwd  mm2, mm2          ; replicate the 2 low v pixels
    pmullw    mm2, VtR
    punpckhwd mm7, mm7
    pmullw    mm7, VtR
    paddusb   mm1, [esp+BUpperLimit] ; mm1: saturate B+0FF-15
    movq      temp_mmx[esp+24], mm2 ; low chroma R
    paddw     mm2, mm0           ; 4 low R values in signed 16 bit
    psraw     mm2, [esp+RRightShift] ; low R scaled down by 6+(8-5)
    pxor      mm4, mm4          ; mm4=0 for 8->16 conversion
    movq      temp_mmx[esp+32], mm7 ; high chroma R
    paddw     mm7, mm3          ; 4 high R values in signed 16 bit
    psraw     mm7, [esp+RRightShift] ; high R scaled down by 6+(8-5)
    psubusb   mm1, [esp+BUpperLimit]
    packuswb  mm2, mm7          ; mm2: R7 R6 R5 R4 R3 R2 R1 R0
    paddusb   mm6, [esp+GUpperLimit] ; G fast patch ih
    psubusb   mm6, [esp+GupperLimit] ; fast patch ih
    paddusb   mm2, [esp+RUpperLimit] ; R
    psubusb   mm2, [esp+RUpperLimit]
; here we are packing from RGB24 to RGB16
; input:
    ; mm6: G7 G6 G5 G4 G3 G2 G1 G0
    ; mm1: B7 B6 B5 B4 B3 B2 B1 B0
    ; mm2: R7 R6 R5 R4 R3 R2 R1 R0
; assuming 8 original pixels in 0-H representation on mm6, mm5, mm2
; when H=2**xBITS-1 (x is for R G B)
; output:
;     mm1- result: 4 low RGB16
;     mm7- result: 4 high RGB16
; using: mm0- zero register
;     mm3- temporary results
; algorithm:
;   for (i=0; i<8; i++) {
;     RGB[i]=256*(R[i]<<(8-5))+(G[i]<<5)+B[i];
;   }
    psllq     mm2, [esp+RLeftShift] ; position R in the most significant part of the byte
    movq      mm7, mm1           ; mm1: Save B
; note: no need for shift to place B on the least significant part of the byte
;   R in left position, B in the right position so they can be combined
    punpcklbw mm1, mm2          ; mm1: 4 low 16 bit RB
    pxor      mm0, mm0          ; mm0: 0
    punpckhbw mm7, mm2          ; mm5: 4 high 16 bit RB
    movq      mm3, mm6          ; mm3: G
    punpcklbw mm6, mm0          ; mm6: low 4 G 16 bit
    psllw     mm6, [esp+GLeftShift] ; shift low G 5 positions
    punpckhbw mm3, mm0          ; mm3: high 4 G 16 bit
    por       mm1, mm6          ; mm1: low RBG16
    psllw     mm3, [esp+GLeftShift] ; shift high G 5 positions
    por       mm7, mm3          ; mm5: high RBG16
    lebp     tmpYCursorOdd      ; moved to here to save cycles before odd line
    movq      [edi], mm1        ; !! aligned
;- start odd line
    movq      mm1, [ebp+2*ebx]    ; mm1 has 8 y pixels
    pxor      mm2, mm2
    psubusb   mm1, Yadd          ; mm1 has 8 pixels y-16
    movq      mm5, mm1
    punpcklbw mm1, mm2          ; get 4 low y-16 unsign pixels word
    pmullw    mm1, Ymul          ; low 4 luminance contribution
    punpckhbw mm5, mm2          ; 4 high y-16
    pmullw    mm5, Ymul          ; high 4 luminance contribution
    movq      [edi+8], mm7      ; !! aligned
    movq      mm0, mm1
    paddw     mm0, temp_mmx[esp+24] ; low 4 R
    movq      mm6, mm5
    psraw     mm0, [esp+RRightShift] ; low R scaled down by 6+(8-5)
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
paddw    mm5, temp_mmx[esp+32] ; high 4 R
movq     mm2, mm1
psraw   mm5, [esp+RRightShift] ; high R scaled down by 6+(8-5)
paddw   mm2, temp_mmx[esp+16] ; low 4 B
packuswb mm0, mm5 ; mm0: R7 R6 R5 R4 R3 R2 R1 R0
psraw   mm2, [esp+BRightShift] ; low B scaled down by 6+(8-5)
movq    mm5, mm6
paddw   mm6, temp_mmx[esp+40] ; high 4 B
psraw   mm6, [esp+BRightShift] ; high B scaled down by 6+(8-5)
movq    mm3, temp_mmx[esp+8] ; chroma G low 4
packuswb mm2, mm6 ; mm2: B7 B6 B5 B4 B3 B2 B1 B0
movq    mm4, mm3
punpcklwd mm3, mm3 ; replicate low 2
punpckhwd mm4, mm4 ; replicate high 2
psubw   mm1, mm3 ; 4 low G
psraw   mm1, [esp+GRightShift] ; low G scaled down by 6+(8-5)
psubw   mm5, mm4 ; 4 high G values in signed 16 bit
psraw   mm5, [esp+GRightShift] ; high G scaled down by 6+(8-5)
paddusb mm2, [esp+BUpperLimit] ; mml: saturate B+0FF-15
packuswb mm1, mm5 ; mml: G7 G6 G5 G4 G3 G2 G1 G0
psubusb mm2, [esp+BupperLimit]
paddusb mm1, [esp+GUpperLimit] ; G
psubusb mm1, [esp+GUpperLimit]
paddusb mm0, [esp+RUpperLimit] ; R
Leax    tmpCCOPitch
psubusb mm0, [esp+RUpperLimit]
; here we are packing from RGB24 to RGB16
; mm1: G7 G6 G5 G4 G3 G2 G1 G0
; mm2: B7 B6 B5 B4 B3 B2 B1 B0
; mm0: R7 R6 R5 R4 R3 R2 R1 R0
; output:
; mm2- result: 4 low RGB16
; mm7- result: 4 high RGB16
; using: mm4- zero register
; mm3- temporary results
psllq   mm0, [esp+RLeftShift] ; position R in the most significant part of the byte
movq    mm7, mm2 ; mm7: Save B
; note: no need for shift to place B on the least significant part of the byte
; R in left position, B in the right position so they can be combined
punpcklbw mm2, mm0 ; mml: 4 low 16 bit RB
pxor     mm4, mm4 ; mm4: 0
movq    mm3, mm1 ; mm3: G
punpckhbw mm7, mm0 ; mm7: 4 high 16 bit RB
punpcklbw mm1, mm4 ; mml: low 4 G 16 bit
punpckhbw mm3, mm4 ; mm3: high 4 G 16 bit
psllw   mm1, [esp+GLeftShift] ; shift low G 5 positions
por     mm2, mm1 ; mm2: low RBG16
psllw   mm3, [esp+GLeftShift] ; shift high G 5 positions
por     mm7, mm3 ; mm7: high RBG16
movq    [edi+eax], mm2
movq    [edi+eax+8], mm7 ; aligned
add     edi, 16 ; ih take 16 bytes (8 pixels-16 bit)
add     ebx, 4 ; ? to take 4 pixels together instead of 2
jl     do_next_8x2_block ; ? update the loop for 8 y pixels at once
ADDedi CCOSkipDistance ; go to begin of next line
ADDedi tmpCCOPitch ; skip odd line (if it is needed)
; Leax AspectCount
; Lebp CCOPitch ; skip odd line
; sub  eax, 2
; jg   @f
; Addeax AspectBaseCount
; xor  ebp, ebp
;@@:
; Seax AspectCount
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
; add      edi, ebp
; leax    YPitch
; lebp    tmpYCursorOdd
; add     ebp, eax      ; skip one line
; lea    ebp, [ebp+2*eax] ; skip two lines
; sebp    tmpYCursorEven
; sebp    tmpYCursorOdd
; add     ebp, eax      ; skip one line
; sebp    tmpYCursorOdd
; lebp    tmpYCursorEven
; lea    ebp, [ebp+2*eax]
; sebp    tmpYCursorEven
; ADDesi  ChromaPitch
; ADDedx  ChromaPitch
; leax    YLimit      ; Done with last line?
; cmp     ebp, eax
; jbe     PrepareChromaLine
; sub     PD FrameHeight[esp],2
; ja     PrepareChromaLine
;-----
finish:
; emms
; add     esp, LocalFrameSize
; pop     ebx
; pop     ebp
; pop     edi
; pop     esi
; retn
MMXCODE1 ENDS
END
```

18. Appendix 7. Color Conversion to RGB16 Zoom by 2

```
-----  
; cx512162 -- This function performs zoom-by-2 YUV12-to-RGB16 color conversion  
; for H26x. It handles 555, 655, 565, and 664 formats.  
;  
; The YUV12 input is planar, 8 bits per pel. The Y plane may have  
; a pitch of up to 768. It may have a width less than or equal  
; to the pitch. It must be DWORD aligned, and preferably QWORD  
; aligned. Pitch and Width must be a multiple of eight.  
; Height must be a multiple of two. The U and V planes may have  
; a different pitch than the Y plane, subject to the same limitations.  
;  
; The color convertor is non destructive.  
-----  
include iammx.inc  
include locals.inc  
.586  
.xlist  
.list  
ASSUME ds:FLAT, cs:FLAT, ss:FLAT  
RTIME16=1  
DITHER=1  
MMXDATA1 SEGMENT PARA USE32 PUBLIC 'DATA'  
ALIGN 8  
RGB_formats:  
    dd RGB565  
    dd RGB555  
    dd RGB664  
    dd RGB655  
Minusg                dd  00800080h,  00800080h  
VtR                   dd  00660066h,  00660066h ;01990199h,01990199h  
VtG                   dd  00340034h,  00340034h ;00d000d0h,00d000d0h  
UtG                   dd  00190019h,  00190019h ;00640064h,00640064h  
UtB                   dd  00810081h,  00810081h ;02050205h,02050205h  
Ymul                  dd  004a004ah,  004a004ah ;012a012ah,012a012ah  
Yadd                  dd  10101010h,  10101010h  
UVtG                  dd  00340019h,  00340019h ;00d00064h,00d00064h  
VtRtB                 dd  01990205h,  01990205h  
fourbitu              dd  0f0f0f0f0h,  0f0f0f0f0h  
fivebitu              dd  0e0e0e0e0h,  0e0e0e0e0h  
sixbitu               dd  0c0c0c0c0h,  0c0c0c0c0h  
shiftwo               dd  02020202h,  02020202h  
shifttwo              dd  04040404h,  04040404h  
shifthree             dd  08080808h,  08080808h  
MMXDATA1 ENDS  
LocalFrameSize       = 174  
RegisterStorageSize  = 16  
; Arguments:  
YPlane                = LocalFrameSize + RegisterStorageSize + 4  
UPlane                = LocalFrameSize + RegisterStorageSize + 8  
VPlane                = LocalFrameSize + RegisterStorageSize + 12  
FrameWidth            = LocalFrameSize + RegisterStorageSize + 16  
FrameHeight           = LocalFrameSize + RegisterStorageSize + 20  
YPitch                = LocalFrameSize + RegisterStorageSize + 24  
ChromaPitch           = LocalFrameSize + RegisterStorageSize + 28  
AspectAdjustmentCount = LocalFrameSize + RegisterStorageSize + 32  
ColorConvertedFrame   = LocalFrameSize + RegisterStorageSize + 36  
DCIOffset             = LocalFrameSize + RegisterStorageSize + 40  
CCOffsetToLine0       = LocalFrameSize + RegisterStorageSize + 44  
CCOPitch              = LocalFrameSize + RegisterStorageSize + 48
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
CCType                = LocalFrameSize + RegisterStorageSize + 52
EndOfArgList          = LocalFrameSize + RegisterStorageSize + 56
; Locals (on local stack frame)
CCOCursor             = 0
CCOSkipDistance       = 4
ChromaLineLen         = 8
YCursor               = 12
DistanceFromVToU     = 16
EndOfChromaLine      = 20
AspectCount           = 24
tmpYCursorEven        = 28
tmpYCursorOdd         = 32
temp_mmx              = 36 ; 48 bytes
RLeftShift            = 84
GLeftShift            = 92
RRightShift           = 100
GRightShift           = 108
BRightShift           = 116
RUpperLimit           = 124
GUpperLimit           = 132
BUpperLimit           = 140
RDither               = 148
GDither               = 156
BDither               = 164
; Switches used by RGB color convertors to determine the exact conversion type.
LCL EQU <esp+>
MMXCODE1 SEGMENT PARA USE32 PUBLIC 'CODE'
; void FAR ASM_CALLTYPE YUV12ToRGB16ZoomBy2 (
;                                     U8* YPlane,
;                                     U8* UPlane,
;                                     U8* VPlane,
;                                     UN  FrameWidth,
;                                     UN  FrameHeight,
;                                     UN  YPitch,
;                                     UN  UVPitch,
;                                     UN  AspectAdjustmentCount,
;                                     U8* ColorConvertedFrame,
;                                     U32 DCIOffset,
;                                     U32 CCOffsetToLine0,
;                                     int  CCOPitch,
;                                     int  CCType)
;
; The local variables are on the stack,
; The tables are in the one and only data segment.
;
; CCOffsetToLine0 is relative to ColorConvertedFrame.
;
PUBLIC C MMX_YUV12ToRGB16ZoomBy2
MMX_YUV12ToRGB16ZoomBy2:
    push esi
    push edi
    push ebp
    push ebx
    sub     esp, LocalFrameSize
    mov     eax, [esp+CCType]
    cmp     eax, 4
    jae     finish
    jmp     RGB_formats[eax*4]
RGB555:
    xor     eax, eax
    mov     ebx, 2 ; 10-8 for byte shift
    mov     [esp+RLeftShift], ebx
    mov     [esp+RLeftShift+4], eax
    mov     ebx, 5
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
mov     [esp+GLeftShift], ebx
mov     [esp+GLeftShift+4], eax
mov     ebx, 9
mov     [esp+RRightShift], ebx
mov     [esp+RRightShift+4], eax
mov     [esp+GRightShift], ebx
mov     [esp+GRightShift+4], eax
mov     [esp+BRightShift], ebx
mov     [esp+BRightShift+4], eax
movq   mm0, fivebitu
movq   [esp+RUpperLimit], mm0
movq   [esp+GUpperLimit], mm0
movq   [esp+BUpperLimit], mm0
movq   mm0, shifttwo      ; 1<<(7-5) for dither
movq   [esp+RDither], mm0
movq   [esp+GDither], mm0
movq   [esp+BDither], mm0
jmp    RGBEND
RGB664:
xor     eax, eax
mov     ebx, 2      ; 8-6
mov     [esp+RLeftShift], ebx
mov     [esp+RLeftShift+4], eax
mov     ebx, 4
mov     [esp+GLeftShift], ebx
mov     [esp+GLeftShift+4], eax
mov     ebx, 8
mov     [esp+RRightShift], ebx
mov     [esp+RRightShift+4], eax
mov     [esp+GRightShift], ebx
mov     [esp+GRightShift+4], eax
movq   mm0, sixbitu
movq   [esp+RUpperLimit], mm0
movq   [esp+GUpperLimit], mm0
mov     ebx, 10
mov     [esp+BRightShift], ebx
mov     [esp+BRightShift+4], eax
movq   mm0, fourbitu
movq   [esp+BUpperLimit], mm0
movq   mm0, shiftone      ; 1<<(7-6) for dither
movq   [esp+RDither], mm0
movq   [esp+GDither], mm0
movq   mm0, shiftthree    ; 1<<(7-4) for dither
movq   [esp+BDither], mm0
jmp    RGBEND
RGB655:
xor     eax, eax
mov     ebx, 2      ; 8-6
mov     [esp+RLeftShift], ebx
mov     [esp+RLeftShift+4], eax
mov     ebx, 5
mov     [esp+GLeftShift], ebx
mov     [esp+GLeftShift+4], eax
mov     ebx, 9
mov     [esp+GRightShift], ebx
mov     [esp+GRightShift+4], eax
mov     [esp+BRightShift], ebx
mov     [esp+BRightShift+4], eax
mov     ebx, 8
mov     [esp+RRightShift], ebx
mov     [esp+RRightShift+4], eax
movq   mm0, fivebitu
movq   [esp+GUpperLimit], mm0
movq   [esp+BUpperLimit], mm0
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
movq mm0,sixbitu
movq [esp+RUpperLimit], mm0
movq mm0,shiftwo ; 1<<(7-5) for dither
movq [esp+GDither],mm0
movq [esp+BDither],mm0
movq mm0,shiftone ; 1<<(7-6) for dither
movq [esp+RDither],mm0
jmp RGBEND
RGB565:
xor eax, eax
mov ebx, 3 ; 8-5
mov [esp+RLeftShift], ebx
mov [esp+RLeftShift+4], eax
mov ebx, 5
mov [esp+GLeftShift], ebx
mov [esp+GLeftShift+4], eax
mov ebx, 9
mov [esp+RRightShift], ebx
mov [esp+RRightShift+4], eax
mov [esp+BRightShift], ebx
mov [esp+BRightShift+4], eax
movq mm0, fivebitu
movq [esp+RUpperLimit], mm0
movq [esp+BUpperLimit], mm0
mov ebx, 8
mov [esp+GRightShift], ebx
mov [esp+GRightShift+4], eax
movq mm0, sixbitu
movq [esp+GUpperLimit], mm0
movq mm0,shiftwo ; 1<<(7-5) for dither
movq [esp+RDither],mm0
movq [esp+BDither],mm0
movq mm0,shiftone ; 1<<(7-6) for dither
movq [esp+GDither],mm0
; jmp RGBEND
RGBEND:
mov ebx, [esp+VPlane]
mov ecx, [esp+UPlane]
sub ecx, ebx
mov [esp+DistanceFromVToU], ecx
mov eax, [esp+ColorConvertedFrame]
add eax, [esp+DCIOffset]
add eax, [esp+CCOffsetToLine0]
mov [esp+CCOCursor], eax
Lebx FrameWidth
Leax CCOPitch
Lesi YPlane ; Fetch cursor over luma plane.
shl ebx, 2 ; FrameWidth*2
sub eax, ebx ; CCOPitch-2*FrameWidth
shr ebx, 3 ; FrameWidth*3
Sesi YCursor
Sebx ChromaLineLen ; FrameWidth*3
Seax CCOSkipDistance ; CCOPitch-3*FrameWidth
Leax AspectAdjustmentCount
Lesi VPlane
Seax AspectCount
xor eax, eax
Ledi ChromaLineLen
Sedi EndOfChromaLine
Ledi CCOCursor
Ledx DistanceFromVToU
Lebp YCursor ; Fetch Y Pitch.
Lebx FrameWidth
add ebp, ebx
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
Sebp    tmpYCursorEven
Leax    YPitch
add     ebp, eax
Sebp    tmpYCursorOdd
sar     ebx, 1
add     esi, ebx
add     edx, esi
neg     ebx
Sebx    FrameWidth
; Register Usage:
;
; ebp -- Y Line cursor. Chroma contribs go in lines above current Y line.
; esi -- Chroma Line cursor.
; edx -- Distance from V pel to U pel.
; edi -- Cursor over the color converted output image.
; ebx -- Number of points taken together.
;
;
; ecx -- Point to Far line (2 lines away)
; eax -- Line Pitch
;-----
PrepareChromaLine:
    Lebx    FrameWidth
    Leax    CCOPitch
do_next_8x2_block:
    Lebp    tmpYCursorEven
    movdt   mm1, [edx+ebx]           ; 4 u values
    pxor    mm0, mm0               ; mm0=0
    movdt   mm2, [esi+ebx]         ; 4 v values
    punpcklbw mm1, mm0             ; get 4 unsign u
    psubw   mm1, Minusg            ; get 4 unsign u-128
    punpcklbw mm2, mm0             ; get unsign v
    psubw   mm2, Minusg            ; get unsign v-128
    movq    mm3, mm1               ; save the u-128 unsign
    movq    mm5, mm1               ; save u-128 unsign
    punpcklwd mm1, mm2             ; get 2 low u, v unsign pairs
    pmaddwd mm1, UVtG              ; create high 2 unsign uv pairs
    punpckhwd mm3, mm2
    pmaddwd mm3, UVtG
    movq    temp_mmx[esp], mm2     ; save v-128
    movq    mm6, [ebp+2*ebx]       ; mm6 has 8 y pixels
    psubusb mm6, Yadd              ; mm6 has 8 y-16 pixels
    packssdw mm1, mm3              ; packed the results to signed words
    movq    mm7, mm6               ; save the 8 y-16 pixels
    punpcklbw mm6, mm0             ; mm6 has 4 low y-16 unsign
    pmullw  mm6, Ymul
    punpckhbw mm7, mm0             ; mm7 has 4 high y-16 unsign
    pmullw  mm7, Ymul
    movq    mm4, mm1
    movq    temp_mmx[esp+8], mm1   ; save 4 chroma G values
    punpcklwd mm1, mm1             ; chroma G replicate low 2
    movq    mm0, mm6               ; low y
    punpckhwd mm4, mm4             ; chroma G replicate high 2
    movq    mm3, mm7
    psubw   mm6, mm1               ; 4 low G
    ; movq    mm1, mm5             ; 4 u values
    psraw  mm6, [esp+GRightShift]
    psubw   mm7, mm4               ; 4 high G values in signed 16 bit
    movq    mm2, mm5
    punpcklwd mm5, mm5             ; replicate the 2 low u pixels
    pmullw  mm5, UtB
    punpckhwd mm2, mm2
    pmullw  mm2, UtB
    psraw  mm7, [esp+GRightShift]
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
packuswb mm6, mm7 ; mm6: G7 G6 G5 G4 G3 G2 G1 G0
movq temp_mmx[esp+16], mm5 ; low chroma B
paddw mm5, mm0 ; 4 low B values in signed 16 bit
movq temp_mmx[esp+40], mm2 ; high chroma B
paddw mm2, mm3 ; 4 high B values in signed 16 bit
psraw mm5, [esp+BRightShift] ; low B scaled down by 6+(8-5)
psraw mm2, [esp+BRightShift] ; high B scaled down by 6+(8-5)
packuswb mm5, mm2 ; mm1: B7 B6 B5 B4 B3 B2 B1 B0
movq mm2, temp_mmx[esp] ; 4 v values
movq mm1, mm5 ; save B
movq mm7, mm2
punpcklwd mm2, mm2 ; replicate the 2 low v pixels
pmullw mm2, VtR
punpckhwd mm7, mm7
pmullw mm7, VtR
paddusb mm1, [esp+BUpperLimit] ; mm1: saturate B+0FF-15
movq temp_mmx[esp+24], mm2 ; low chroma R
paddw mm2, mm0 ; 4 low R values in signed 16 bit
psraw mm2, [esp+RRightShift] ; low R scaled down by 6+(8-5)
pxor mm4, mm4 ; mm4=0 for 8->16 conversion
movq temp_mmx[esp+32], mm7 ; high chroma R
paddw mm7, mm3 ; 4 high R values in signed 16 bit
psraw mm7, [esp+RRightShift] ; high R scaled down by 6+(8-5)
psubusb mm1, [esp+BUpperLimit]
packuswb mm2, mm7 ; mm2: R7 R6 R5 R4 R3 R2 R1 R0
paddusb mm6, [esp+GUpperLimit] ; G
psubusb mm6, [esp+GUpperLimit]
paddusb mm2, [esp+RUpperLimit] ; R
psubusb mm2, [esp+RUpperLimit]
psllq mm2, [esp+RLeftShift] ; position R in the most significant
part of the byte
movq mm7, mm1 ; mm1: Save B
; note: no need for shift to place B on the least significant part of the byte
; R in left position, B in the right position so they can be combined
punpcklbw mm1, mm2 ; mm1: 4 low 16 bit RB
pxor mm0, mm0 ; mm0: 0
punpckhbw mm7, mm2 ; mm7: 4 high 16 bit RB
movq mm3, mm6 ; mm3: G
punpcklbw mm6, mm0 ; mm6: low 4 G 16 bit
psllw mm6, [esp+GLeftShift] ; shift low G 5 positions
punpckhbw mm3, mm0 ; mm3: high 4 G 16 bit
psllw mm3, [esp+GLeftShift] ; shift high G 5 positions
por mm1, mm6 ; mm1: low RBG16
movq mm2, mm1
por mm7, mm3 ; mm7: high RBG16
punpcklwd mm1, mm1
movq [edi], mm1 ; !! aligned
punpckhwd mm2, mm2
movq [edi+eax], mm1 ; !! patch
movq [edi+8], mm2 ; !! patch
movq [edi+eax+8], mm2 ; !! patch
movq mm6, mm7
punpcklwd mm7, mm7 ; get 4 low y-16 unsign pixels word
movq [edi+16], mm7 ; !! aligned
punpckhwd mm6, mm6 ; get 4 low y-16 unsign pixels word
movq [edi+eax+16], mm7 ; !! aligned
movq [edi+24], mm6 ; !! aligned
movq [edi+eax+24], mm6 ; !! aligned
;- start odd line
Lebp tmpYCursorOdd ; moved here to save cycles before odd line
movq mm1, [ebp+2*ebx] ; mm1 has 8 y pixels
pxor mm2, mm2
psubusb mm1, Yadd ; mm1 has 8 pixels y-16
movq mm5, mm1
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```

punpcklbw mm1, mm2                ; get 4 low y-16 unsign pixels word
pmullw mm1, Ymul                  ; low 4 luminance contribution
punpckhbw mm5, mm2                ; 4 high y-16
pmullw mm5, Ymul                  ; high 4 luminance contribution
movq mm0, mm1
paddw mm0, temp_mmx[esp+24]        ; low 4 R
movq mm6, mm5
psraw mm0, [esp+RRightShift]      ; low R scaled down by 6+(8-5)
paddw mm5, temp_mmx[esp+32]        ; high 4 R
movq mm2, mm1
psraw mm5, [esp+RRightShift]      ; high R scaled down by 6+(8-5)
paddw mm2, temp_mmx[esp+16]        ; low 4 B
packuswb mm0, mm5                 ; mm0: R7 R6 R5 R4 R3 R2 R1 R0
psraw mm2, [esp+BRightShift]      ; low B scaled down by 6+(8-5)
movq mm5, mm6
paddw mm6, temp_mmx[esp+40]        ; high 4 B
psraw mm6, [esp+BRightShift]      ; high B scaled down by 6+(8-5)
movq mm3, temp_mmx[esp+8]         ; chroma G low 4
packuswb mm2, mm6                 ; mm2: B7 B6 B5 B4 B3 B2 B1 B0
movq mm4, mm3
punpcklwd mm3, mm3                ; replicate low 2
punpckhwd mm4, mm4                ; replicate high 2
psubw mm1, mm3                    ; 4 low G
psraw mm1, [esp+GRightShift]      ; low G scaled down by 6+(8-5)
psubw mm5, mm4                    ; 4 high G values in signed 16 bit
psraw mm5, [esp+GRightShift]      ; high G scaled down by 6+(8-5)
pxor mm3, mm3                     ; B
paddusb mm2, [esp+BUpperLimit]    ; mm1: saturate B+0FF-15
packuswb mm1, mm5                 ; mm1: G7 G6 G5 G4 G3 G2 G1 G0
psubusb mm2, [esp+BupperLimit]
paddusb mm1, [esp+GUpperLimit]    ; G
psubusb mm1, [esp+GUpperLimit]
paddusb mm0, [esp+RUpperLimit]    ; R
psubusb mm0, [esp+RUpperLimit]
lea ecx, [eax+2*eax]              ; ecx - point to next 3 line
psllq mm0, [esp+RLeftShift]       ; position R in the most significant
part of the byte
movq mm7, mm2                     ; mm7: Save B
; note: no need for shift to place B on the least significant part of the byte
; R in left position, B in the right position so they can be combined
punpcklbw mm2, mm0                ; mm1: 4 low 16 bit RB
pxor mm4, mm4                     ; mm4: 0
movq mm3, mm1                     ; mm3: G
punpckhbw mm7, mm0                ; mm7: 4 high 16 bit RB
punpcklbw mm1, mm4                 ; mm1: low 4 G 16 bit
punpckhbw mm3, mm4                 ; mm3: high 4 G 16 bit
psllw mm1, [esp+GLeftShift]       ; shift low G 5 positions
por mm2, mm1                       ; mm2: low RGB16
psllw mm3, [esp+GLeftShift]       ; shift high G 5 positions
movq mm4, mm2                     ; mm4: save low RGB16
por mm7, mm3                       ; mm7: high RGB16
punpcklwd mm2, mm2                 ; replicate low low RGB16
movq [edi+2*eax], mm2
punpckhwd mm4, mm4                 ; replicate high low RGB16
movq [edi+2*eax+8], mm4            ; patch
movq mm5, mm7                     ; save high RGB16
movq [edi+ecx], mm2
punpcklwd mm7, mm7
movq [edi+ecx+8], mm4              ; patch
punpckhwd mm5, mm5
movq [edi+ecx+16], mm7             ; aligned
movq [edi+2*eax+16], mm7          ; aligned
movq [edi+ecx+24], mm5            ; aligned
movq [edi+2*eax+24], mm5          ; aligned
```

Color Conversion from YUV12 to RGB Using Intel MMX™ Technology

March 1996

```
    add     edi, 32                ; ih take 16 bytes (8 pixels-16 bit)
    add     ebx, 4                 ; ? to take 4 pixels together instead of 2
    jl     do_next_8x2_block      ; ? update the loop for 8 y pixels at once
    ADDedi  CCOSkipDistance       ; go to begin of next line
    ADDedi  CCOPitch              ; skip odd line
    ADDedi  CCOPitch              ; skip odd line
    ADDedi  CCOPitch              ; skip odd line
    Leax   CCOPitch
    Leax   YPitch
    Leb    tmpYCursorOdd
    lea    ebp, [ebp+2*eax]       ; skip two lines
    Sebp   tmpYCursorOdd
    Leb    tmpYCursorEven
    lea    ebp, [ebp+2*eax]
    Sebp   tmpYCursorEven
    ADDesi  ChromaPitch
    ADDEDx  ChromaPitch
    sub     PD FrameHeight[esp],2
    ja     PrepareChromaLine
;-----
finish:
    emms
    add     esp, LocalFrameSize
    pop    ebx
    pop    ebp
    pop    edi
    pop    esi
    retn
MMXCODE1 ENDS
END
```

19. References

[1] Recommendation and Reports. Recommendation 601-1. Encoding Parameters of Digital Television For Studios