Motorola Semiconductor Israel

Feig's 8x8 DCT algorithm on Motorola DSP56300

Dror Halahmi

September 1, 1998

1. Introduction **4**
2. Transform definition **5**
2.1 FDCT 5
2.2 IDCT 5
3. Test vector creation **7**
4. Reference code in Matlab **9**
5. Straight forward implementation **11**
5.1 Scaling selection 11
5.2 Sources 11
5.3 Tools 12
5.4 Verification 12
5.5 TEST Mode 12
5.6 Performance 13
5.7 Target 13
6. Fast Implementation **14**
6.1 Choosing the appropriate algorithm 14
6.2 Memory organization 15
6.2.1 In-place implementation 15
6.2.2 Data distribution 16
6.2.3 Add vs. multiply operations 16
6.2.4 Matrix rows and columns 16
6.2.5 Coefficients organization 18
6.3 Fixed point problems and scaling 19
6.4 Implementation 20
6.4.1 Flowchart optimization 20
6.4.2 Various block versions due to memory allocation 21
6.4.3 Using macros 21
6.5 Verification 21
6.5.1 permute_scale.m 21
6.5.2 Comparison in Matlab 21
6.6 Performance 22
7. References **23**

Appendix A:Tools **24**

A.1 shift.pl 24
A.2 cycles.awk 24

Appendix B:Straight Forward Implementation **25**

B.1 Sources 25
B.1.1 dct 25

B.1.2 dct.cmd 25
B.1.3 dct.asm 25
B.1.4 sincos.asm 27
B.1.5 dec_input.sf 27
B.2 Results 28
B.2.1 dec_output.sf 28

Appendix C:Fast Implementation 30

C.1 Sources 30
C.1.1 dct.asm 30
C.1.2 coefs.asm 33
C.1.3 R1rowsXY.asm 33
C.1.4 R1rowsYX.asm 35
C.1.5 R1columnsXY.asm 36
C.1.6 R1columnsYX.asm 37
C.1.7 M1XY.asm 38
C.1.8 M1YX.asm 39
C.1.9 M2XY.asm 39
C.1.10 M2YX.asm 40
C.1.11 M3YX.asm 41
C.1.12 R2rowsXY.asm 42
C.1.13 R2rowsYX.asm 43
C.1.14 R2columnsXY.asm 44
C.1.15 R2columnsYX.asm 45
C.2 Results 45
C.2.1 dec_output.fi 45

1. Introduction

The Discrete Cosine Transform (DCT) is implemented here. First, the transform and its inverse are defined. Then a test vector is created. Then a reference code is written in Matlab, and output reference is created. Then the code is implemented in DSP56300 asm in straight forward and then in fast methodology.

2. Transform definition

2.1 FDCT

The 2-D FDCT is defined as:

$$S_{v,u} = \frac{C_v}{2}\frac{C_u}{2}\sum_{y=0}^{7}\sum_{x=0}^{7} s_{y,x}\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}$$

Where:

$$C(\alpha) = \begin{cases} \frac{1}{\sqrt{2}} & \alpha = 0 \\ 1 & \alpha \neq 0 \end{cases}$$

In matrix presentation:

$$\begin{bmatrix} S_{0,0} & \cdots & S_{0,7} \\ \vdots & \cdots & \vdots \\ S_{7,0} & \cdots & S_{7,7} \end{bmatrix} = \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \cdots & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\cos\frac{\pi}{16} & \frac{1}{2}\cos\frac{3\pi}{16} & \cdots & \frac{1}{2}\cos\frac{15\pi}{16} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{1}{2}\cos\frac{7\pi}{16} & \frac{1}{2}\cos\frac{21\pi}{16} & \cdots & \frac{1}{2}\cos\frac{105\pi}{16} \end{bmatrix} \begin{bmatrix} s_{0,0} & \cdots & s_{0,7} \\ \vdots & \cdots & \vdots \\ s_{7,0} & \cdots & s_{7,7} \end{bmatrix} \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2}\cos\frac{\pi}{16} & \cdots & \frac{1}{2}\cos\frac{7\pi}{16} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2}\cos\frac{3\pi}{16} & \cdots & \frac{1}{2}\cos\frac{21\pi}{16} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{1}{2\sqrt{2}} & \frac{1}{2}\cos\frac{15\pi}{16} & \cdots & \frac{1}{2}\cos\frac{105\pi}{16} \end{bmatrix}$$

2.2 IDCT

The 2-D IDCT is defined as:

$$s_{y,x} = \sum_{v=0}^{7}\frac{C_v}{2}\sum_{u=0}^{7}\frac{C_u}{2}S_{v,u}\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}$$

Where:

$$C_\alpha = \begin{cases} \frac{1}{\sqrt{2}} & \alpha = 0 \\ 1 & \alpha \neq 0 \end{cases}$$

In matrix presentation:

$$\begin{bmatrix} s_{0,0} & \circ\circ\circ & s_{0,7} \\ \circ & \circ\circ\circ & \circ \\ \circ & \circ\circ\circ & \circ \\ \circ & \circ\circ\circ & \circ \\ s_{7,0} & \circ\circ\circ & s_{7,7} \end{bmatrix} = \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2}\cos\frac{\pi}{16} & \circ\circ\circ & \frac{1}{2}\cos\frac{7\pi}{16} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2}\cos\frac{3\pi}{16} & \circ\circ\circ & \frac{1}{2}\cos\frac{21\pi}{16} \\ \circ & \circ & \circ\circ\circ & \circ \\ \circ & \circ & \circ\circ\circ & \circ \\ \circ & \circ & \circ\circ\circ & \circ \\ \frac{1}{2\sqrt{2}} & \frac{1}{2}\cos\frac{15\pi}{16} & \circ\circ\circ & \frac{1}{2}\cos\frac{105\pi}{16} \end{bmatrix} \begin{bmatrix} S_{0,0} & \circ\circ\circ & S_{0,7} \\ \circ & \circ\circ\circ & \circ \\ \circ & \circ\circ\circ & \circ \\ \circ & \circ\circ\circ & \circ \\ S_{7,0} & \circ\circ\circ & S_{7,7} \end{bmatrix} \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \circ\circ\circ & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\cos\frac{\pi}{16} & \frac{1}{2}\cos\frac{3\pi}{16} & \circ\circ\circ & \frac{1}{2}\cos\frac{15\pi}{16} \\ \circ & \circ & \circ\circ\circ & \circ \\ \circ & \circ & \circ\circ\circ & \circ \\ \circ & \circ & \circ\circ\circ & \circ \\ \frac{1}{2}\cos\frac{7\pi}{16} & \frac{1}{2}\cos\frac{21\pi}{16} & \circ\circ\circ & \frac{1}{2}\cos\frac{105\pi}{16} \end{bmatrix}$$

3. Test vector creation

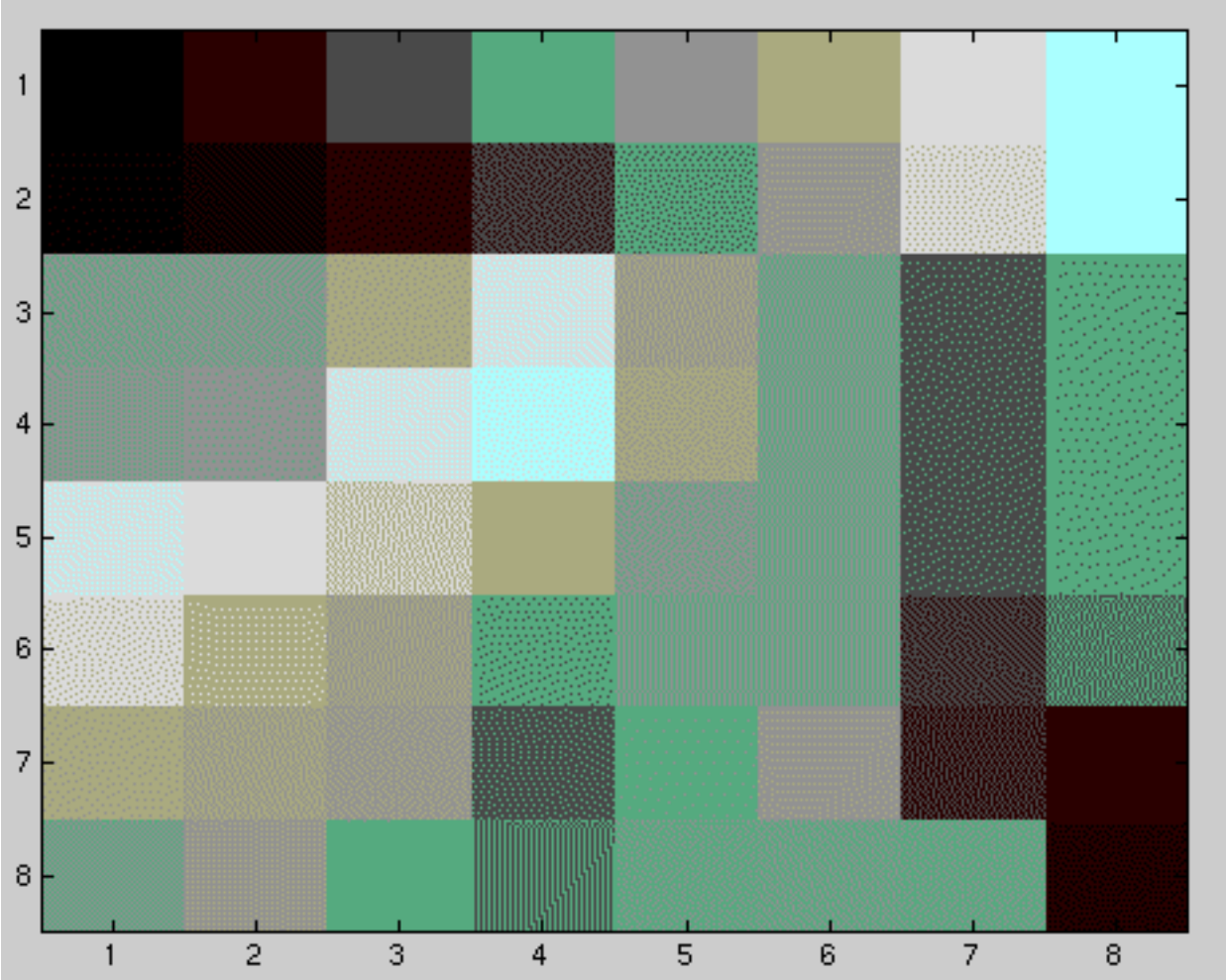
The input matrix is given in a 9 bit precision which can be presented in an integer presentation, i.e. any integer from -256 to 255. The output of such input matrix will be in the range [-2048, 2040] which can be presented in a 12 bit integer presentation, but with loss of accuracy. Therefore, in order to reduce the accuracy loss, the whole 24 bit precision of the DSP56300 will be used.

The following $s_{x,y}$ is used as input vector (matrix):

$$s = \begin{bmatrix} -256 & -183 & -110 & -37 & 36 & 109 & 182 & 255 \\ -250 & -230 & -190 & -130 & -50 & 50 & 170 & 255 \\ 10 & 15 & 99 & 200 & 70 & 0 & -100 & -40 \\ 20 & 30 & 200 & 240 & 88 & 0 & -100 & -40 \\ 200 & 180 & 150 & 109 & 17 & 0 & -100 & -40 \\ 171 & 117 & 73 & -44 & 0 & 0 & -133 & -71 \\ 100 & 81 & 60 & -96 & -33 & 50 & -150 & -180 \\ 0 & 56 & -35 & -73 & -14 & -8 & -15 & -203 \end{bmatrix}$$

This matrix represents the following 8x8 image:

Figure 3-1 Image presented by the matrix *s*



4. Reference code in Matlab

The following code was written in Matlab. The results appear, whenever interesting, within the program.

```
>> s=[[-256  -183  -110   -37    36   109   182   255
      -250  -230  -190  -130   -50    50   170   255
        10    15    99   200    70     0  -100   -40
        20    30   200   240    88     0  -100   -40
       200   180   150   109    17     0  -100   -40
       171   117    73   -44     0     0  -133   -71
       100    81    60   -96   -33    50  -150  -180
         0    56   -35   -73   -14    -8   -15  -203]];
>> for i=2:8
      for j=1:8
         c(i,j)=cos((i-1)*(2*j-1)*pi/16)/2;
      end
  end
>> for i=1:8
      c(1,i)=1/2/sqrt(2);
  end
>> S=c*s*c'

S =

  Columns 1 through 7

   60.2500   62.5031  -99.9727  -15.4651   -6.7500  -38.6295   78.7526
   30.6655 -633.8719  -17.8527 -179.1577  116.8557   50.0413  -18.5347
 -265.7115 -482.9829  120.5694  116.9538 -115.6955  -20.6970  -74.0768
   32.6844  -75.3972  173.7758   39.2529  -18.4916  -67.4746   49.2726
  104.2500   -3.0236  -75.1194  -52.9502   -6.7500   36.9318  -53.3111
   57.5726  122.7697 -155.7408  -30.5979    6.3663   -1.1566   66.8705
   64.0596  129.8241  -65.5768   -4.5419   38.1811    0.1737  -27.5694
   64.5107  154.1565  -15.0818   19.5622   -4.8628   -7.2301    5.4192

  Column 8

  -31.3820
  -23.2202
   41.9015
   13.8026
   -0.2193
   -6.5008
   -5.9591
  -33.7244

>> s_=ct*S*c

s_ =

  Columns 1 through 7

 -256.0000 -183.0000 -110.0000  -37.0000   36.0000  109.0000  182.0000
 -250.0000 -230.0000 -190.0000 -130.0000  -50.0000   50.0000  170.0000
   10.0000   15.0000   99.0000  200.0000   70.0000    0.0000 -100.0000
   20.0000   30.0000  200.0000  240.0000   88.0000    0.0000 -100.0000
  200.0000  180.0000  150.0000  109.0000   17.0000   -0.0000 -100.0000
  171.0000  117.0000   73.0000  -44.0000    0.0000   -0.0000 -133.0000
  100.0000   81.0000   60.0000  -96.0000  -33.0000   50.0000 -150.0000
    0.0000   56.0000  -35.0000  -73.0000  -14.0000   -8.0000  -15.0000

  Column 8

  255.0000
  255.0000
  -40.0000
  -40.0000
  -40.0000
  -71.0000
 -180.0000
 -203.0000

>> s
```

s =

```
  -256  -183  -110   -37    36   109   182   255
  -250  -230  -190  -130   -50    50   170   255
    10    15    99   200    70     0  -100   -40
    20    30   200   240    88     0  -100   -40
   200   180   150   109    17     0  -100   -40
   171   117    73   -44     0     0  -133   -71
   100    81    60   -96   -33    50  -150  -180
     0    56   -35   -73   -14    -8   -15  -203
```

It is seen that ***s_***, the reconstruction of ***s*** by applying the IDCT on ***S***, is pretty similar to the original matrix ***s***.

5. Straight forward implementation

5.1 Scaling selection

The DSP56300 is a fixed point DSP, i.e. all numbers must be in the range [-1,1). However, there is an exception for interim results in the **a** and **b** accumulators which can be in the range [-256,256). Since the inputs are given in integer presentation in the range of [-256,255] and the outputs can reach any value in the range [-2048, 2040], a scaling factor of 2^{-11} will be used on the inputs before using in the program and a scaling factor of 2^{11} will be used on the outputs after created by the program. These will be done using the tool *shift.pl*.

Table 5-1 I/O Files Description

File	Description
dec_input.sf[a]	s(y,x)
input.sf	2^{-11} s(y,x)
output.sf	2^{-11} S(y,x)
dec_output.sf	S(y,x)

a. *sf* stands for straight forward implementation

5.2 Sources

The following source files are used to create the output:

Table 5-2 Source Files Description

File	Description
dct	Batch file for the whole application
dct.cmd	Command file for the DSP56300 simulator

Table 5-2 Source Files Description

File	Description
dct.asm	DCT algorithm implementation of the DSP56300 assembly
sincos.asm	Macro that creates the coefficients for the DCT
dec_input.sf	64 integer words of the input s(y,x)

5.3 Tools

The following tools are used in the process of the simulation:

Table 5-3 Tools Description

Tool	Description
shift.pl	Scaling of a data file by shifting either left or right in a predefined number of bits
cycles.awk	Counts the cycles used in each of the stages: power up, init, kernel, term
asm56300	Assembler of the DSP56300
sim56300	Simulator of the DSP56300

5.4 Verification

The straight forward implementation is verified by comparing its output *dec_output.sf* (see Appendix B.2.1) to the **S** matrix of the simulation in Matlab, as appears in section 4.

After being convinced that both data files are similar, the *dec_output.sf* file becomes a reference file for the fast implementation such that the last will be verifiable.

5.5 TEST Mode

A special parameter *TEST* is used in the program. This parameter is used to select between two options:

1. TEST = 1: The implementation is tested and the results should be accurate.
2. TEST = 0: The performance is tested. There is no meaning for the results.

5.6 Performance

The following performance was measured on the straight forward implementation (TEST = 0):

Table 5-4 Performance of the straight forward implementation

Section	Cycle Count	Program Words
power up	0	0
init	13	13
kernel	21661	22
term	0	0

5.7 Target

After the straight forward implementation is done, a fast implementation will be used. The target is defined as *to implement the algorithm, i.e. reach similar results as the straight forward implementation, such that the minimum number of cycles in the kernel part will be achieved*.

6. Fast Implementation

In order to reach the target, the following steps were applied:

1. Choose the most efficient algorithm for DCT 8x8 implementation.
2. Implement it on the DSP56300 assembly.
3. Optimize the code.

Step 1 is the most difficult and with the highest impact on results. Therefore, most of the effort was invested on this step. When referring to the literature, there are plenty of algorithmic methods to calculate the DCT. Their efficiency is usually measured in minimum number of multiplications, which is not the best criterion for the DSP56300, which can do a *multiply* operation as "easy" as *add* operation. In fact it can do *multiply* and *add* together in the single *mac* operation, in the same single cycle which takes for *multiply* or *add*. Roughly speaking, every arithmetic operation takes one cycle. Therefore the criterion which was applied in finding the best algorithm was the algorithm that takes the minimum number of arithmetic operations.

The scaling factor was selected to 2^{-12} instead of 2^{-11} in the straight forward implementation in order to prevent overflow.

The same source files and input files, tools and source files were used, as in the straight forward implementation. The only adjustment to the Fast Implementation version was applied to the input and output files' extension. Instead of *sf* (Straight Forward) *fi* (Fast Implementation) was used.

6.1 Choosing the appropriate algorithm

Table 6-1 summarizes the various DCT algorithms that have been inspected. In [9], which is the most recent document, H. R. Wu and Z. Man claim that the implementation in [8] is the most efficient one. It has the same number of additions as Feig algorithm [7], but it achieves Fieg's lower bound on the multiplicative complexity of the 2-D 2n x 2n - point DCT. However Feig at [7] uses a trick to achieve the best performance so far. His trick is to take out of the DCT permutations and scaling operations which are practically done in the quantization stage after the DCT is done. Instead of multiplying each sample by a scaling number and then once more by a quantization factor, Feig suggests to do that together in a single operation which is excluded of the DCT. By that Feig could "beat" his own lower bound, a fact that shows that the comparison is not accurate.

All measurements and algorithms evaluation take into account the number of ALU operations i.e. multiplications, additions, shifts. However, the number of operations are much more than that. In order to implement an algorithm on a DSP, the operands must be there at the multiplier, at the adder and at the shifter inputs. Moreover, in order to manage such an amount of data, many move operations of data are required. An algorithm which takes this problem into account, e.g. all the operations are on the sequential order of the data, is much more preferable than a one with less ALU operations but more complex in data manipulation.

After trying to evaluate all the information that appears in Table 6-1 as well as evaluating the data manipulation complexity in each proposed algorithm Feig's scaled DCT algorithm [7] was chosen.

It was implemented on Motorola's DSP56300 and was verified by Matlab implementation. The permutations and scaling were not implemented on the DSP.

Algorithm	Multiplications	Additions	Shifts	Total arithmetic operations	Reference
1-D 2^n pts	$2^{n+1} - n - 2$ [a]				[2]
1-D 8 pts	11	29			[3]
1-D 16 pts	31	81			[3]
2-D 2^n x 2^n pts	$2^n (2^{n+1} - n - 2)$ [b]				[4]
2-D 8 x 8 pts	96	466			[5], [6]
2-D 8 x 8 pts	94	454			[7]
2-D 8 x 8 pts	88	454			[8]
2-D 8 x 8 pts	81	460			[1] (Vetterli)
2-D 8 x 8 pts	94	454		548	[7]
factorization of 2-D 8 x 8 pts	54	462	6		[1] (Feig)

Table 6-1 DCT algorithms' performance

a. According to [4] this is the theoretical lower bound on the multiplicative complexity of the one - dimensional (1-D) 2^n-points discrete cosine transform. In 2^n=8 this bound is 11.

b. According to [4] this is the theoretical lower bound on the multiplicative complexity of the 2-D 2^n x 2^n - point DCT. In 8 x 8, this bound is 88.

6.2 Memory organization

6.2.1 In-place implementation

Feig's algorithm is very convenient for in-place implementation method. The algorithm is divided

into stages where in each stage all or some of the data is replaced by the new results. After the last stage, the memory location that held the input values, now holds the DCT output.

6.2.2 Data distribution

The Motorola DSP56300 is capable of doing one arithmetic operation plus up to 2 data moves, one to/from X memory bank and the other to/from Y memory bank. In order to have the most efficient implementation the data must be balanced between X and Y memories such that the number of accesses to each bank will be approximately similar.

6.2.3 Add vs. multiply operations

The add operations are more convenient in memory efficient usage than the multiply ones. Adding two operands of *n* bits each can result in maximum *n+1* bits. However, multiplying them will result in *2n* bits. For the preadditions stage a very economical memory usage can be used, as in Figure 6-1. However, after the preadditions stage, the data must be separated in order to eliminate aliasing between two consecutive data words. A less economic options, but less complicate, appears in Figure 6-2. This one will take twice the amount of cycles than option 1, but eliminated the need to separate the data before the multiplication stage. Therefore, option 2 was chosen.

6.2.4 Matrix rows and columns

The algorithm works on a data base organized as 8x8 matrix. First preadditions stage works on rows then columns, the multiplication stage works on columns and the postadditions stage works on columns then rows. This matrix must be separated into one 64 length vector when allocated to memory. In order to have the minimum complexity, the input data was separated to columns, i.e. $s_{0,0}\, s_{1,0} \ldots s_{7,0}\, s_{0,1}$ etc. This way, most of the operations is done on a sequential data in the memory. First 4 elements of the column were located in one memory bank and the rest 4 in the other. In order to have a balance between X and Y memories also on the rows, in the first 4 rows the first 4 elements were located in X memory and in the rest 4 rows the first 4 elements were located in Y memory (see Figure 6-2). In order to reduce data pointers complexity, both X and Y base addresses are the same.

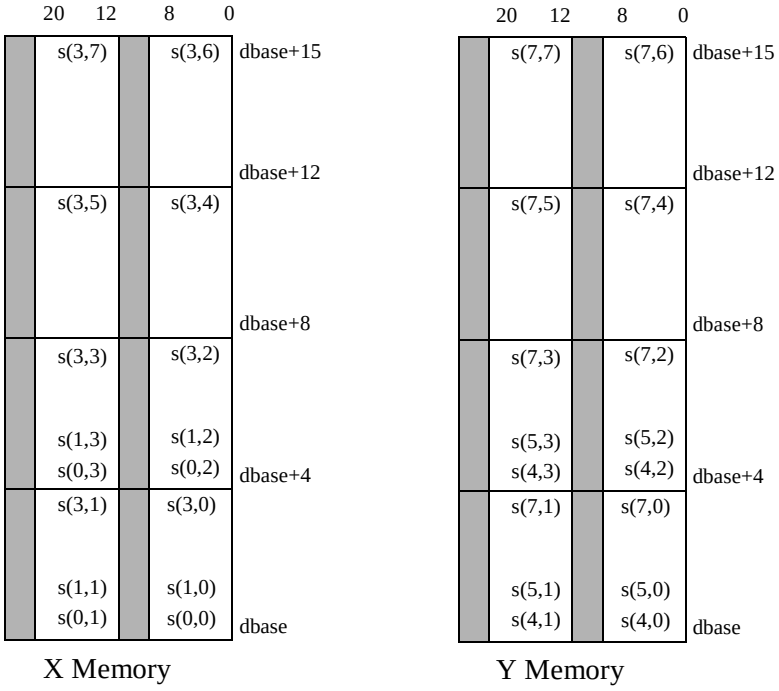


Figure 6-1 Data memory organization - Option 1

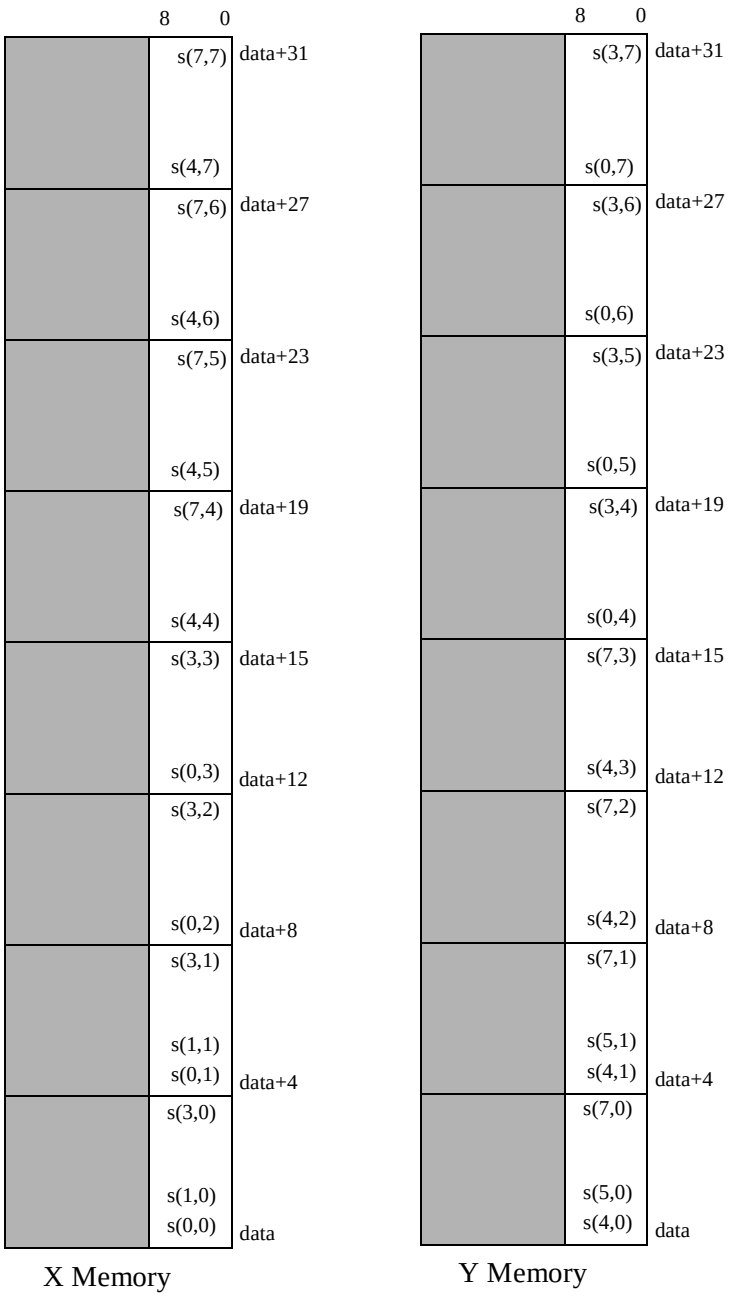


Figure 6-2 Data memory organization - Option 2

6.2.5 Coefficients organization

In order to be balanced in X and Y memory banks, both in the case where the data is in X memory and where the data is in Y memory, the coefficients were located both in X and Y memories. Moreover, in order to improve performance, most of the coefficients appear twice in each memory

bank (see Figure 6-3).

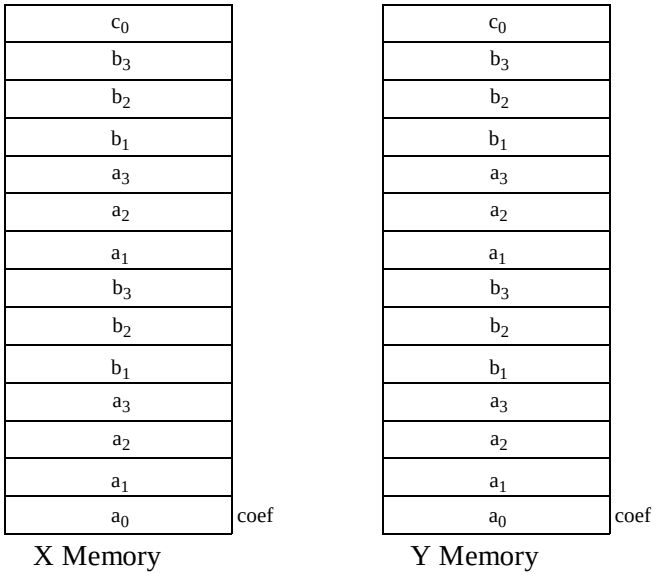


Figure 6-3 Coefficients organization

6.3 Fixed point problems and scaling

The Motorola DSP56300 is a fixed point processor, i.e. all the values it can handle must be in the range of [-1,1). In order to meet that constraint, all inputs must be scaled accordingly, as described in section 5.1. A more complicated problem exists in the multiplication stage where some of the operands are in power of 2, e.g. 1/2 or even 1. In order to eliminate the need to add operations for multiplying by 1, which is required whenever the coefficients are also scaled, the coefficients must not be scaled. However not all of them are in the range of [-1,1). There is one exception, a_3 with the value of 1.3066 which prevents the option not to scale the coefficients. The solution that was applied was to use the scaling mode of the DSP56300. This mode enables multiplying (scale up) or dividing (scale down) each or the arithmetic results by 2 automatically. In this way all the coefficients are divided by 2 in advance and the scaling mode multiplies the results by 2 such that the coefficients scaling becomes invisible. However, in some cases the arithmetic operations are more complex than a single operation. Each additional operation scales up the result in factor of 2. Therefore, in these cases, the coefficient (a_2 and b_2) must be divided by 4 in advance instead of by 2. The scaling mode is enabled before the start of M1 of the multiplication stage and disabled before Q3 of M3 of the multiplication stage, which does not use a_2.

6.4 Implementation

6.4.1 Flowchart optimization

The implementation followed Feig's flowchart [7] in general. However, few minor optimizations and adjustments were applied to the chart. That was especially for R1 preadditions stage (Figure 6-4)and Q3 of multiplication stage (Figure 6-5).

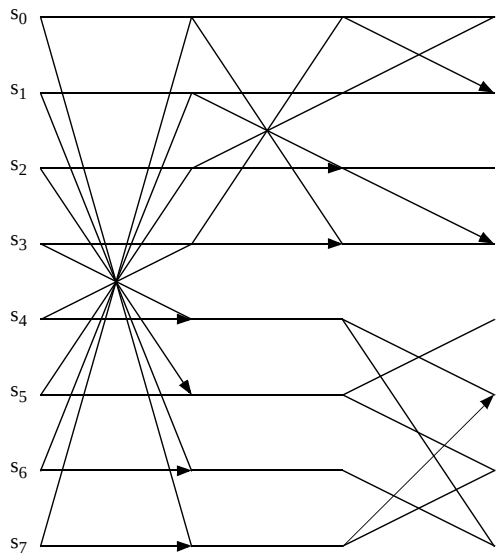


Figure 6-4 Circuit element R1 for preadditions stage

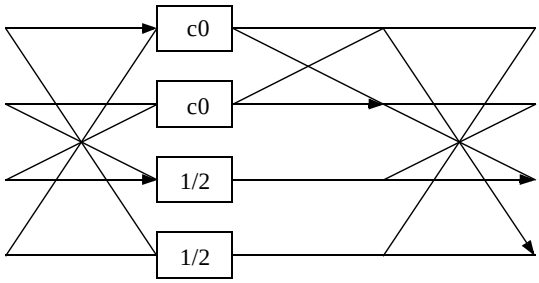


Figure 6-5 Q3 of mutiplication stage

6.4.2 Various block versions due to memory allocation

Since the data is organized in a very specific way (see section 6.2.4), the implementation is different when working on rows than on columns and when working on first 4 rows than on last 4 rows. Therefore 4 different versions were created to both R1 preadditions stage and R2 postadditions stage and 2 different versions were created to both M1 of multiplication stage and M2 of multiplication stage.

6.4.3 Using macros

In order to reach the best performance in cycle count, also in a penalty of program memory, macros were used instead of subroutines. This way the jump to subroutine and the return from subroutine operations are eliminated.

6.5 Verification

As described in section 6.1, this implementation does not include the permutation and scaling post stages. Therefore, in order to verify the results, the output of this implementation was driven to a Matlab program (see section 6.5.1) that implements these stages. The output of the Matlab program was compared to the reference (see *S* matrix in chapter 4.) and found to be similar (neglecting accuracy errors). The comparison program in Matlab and its result can be seen in section .

6.5.1 permute_scale.m

```
function y=permute_scale(fn)
fh=fopen(fn,'r');
x=fscanf(fh,'%f',64);
fclose(fh);
s1=reshape(x,8,8)';
p=[1,5,3,7,2,4,8,6];
for i=1:8
  for j=1:8
    s2(p(i),p(j))=s1(i,j);
  end
end
i1=[0,3,3,2,0,1,1,5];
i2=[0,6,5,7,0,2,7,6];
cf=[1,-4,4,-4,-1,-4,4,4];
for i=1:8
  for j=1:8
    rr(i,j)=cf(i)*cf(j)*gama(i1(i))*gama(i2(i))*gama(i1(j))*gama(i2(j))/8;
  end
end
y=s2.*rr;
```

6.5.2 Comparison in Matlab

```
>> permute_scale('dec_output.fi')

ans =

  Columns 1 through 7

   60.2500   62.5031  -99.9728  -15.4651   -6.7500  -38.6295   78.7525
   30.6655 -633.8721  -17.8529 -179.1578  116.8557   50.0413  -18.5347
 -265.7116 -482.9830  120.5693  116.9538 -115.6955  -20.6970  -74.0768
   32.6844  -75.3973  173.7758   39.2529  -18.4917  -67.4745   49.2726
  104.2500   -3.0236  -75.1193  -52.9502   -6.7500   36.9318  -53.3111
   57.5728  122.7696 -155.7407  -30.5980    6.3663   -1.1549   66.8705
   64.0596  129.8241  -65.5768   -4.5419   38.1811    0.1738  -27.5694
   64.5107  154.1565  -15.0816   19.5622   -4.8628   -7.2306    5.4192

  Column 8
```

```
 -31.3819
 -23.2202
  41.9017
  13.8025
  -0.2194
  -6.5010
  -5.9591
 -33.7245

>> permute_scale('dec_output.fi')-c*s*c'

ans =

  Columns 1 through 7

    0.0000   -0.0000   -0.0001    0.0000   -0.0000    0.0000   -0.0000
    0.0000   -0.0002   -0.0002   -0.0001   -0.0000   -0.0000   -0.0000
   -0.0000   -0.0001   -0.0002    0.0000   -0.0000    0.0000   -0.0000
    0.0000   -0.0001    0.0000   -0.0000   -0.0001    0.0000    0.0000
    0.0000   -0.0000    0.0001   -0.0000   -0.0000   -0.0000    0.0000
    0.0001   -0.0002    0.0001   -0.0001    0.0001    0.0017    0.0000
   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0002    0.0000
    0.0000    0.0001    0.0001   -0.0000   -0.0000   -0.0005    0.0000

  Column 8

    0.0000
   -0.0000
    0.0002
   -0.0001
   -0.0000
   -0.0003
   -0.0000
   -0.0001
```

6.6 Performance

On Table 6-2 the performance that was measured on the fast implementation (TEST = 0) is shown. For convenience, the old results of the straight forward implementation appear on the same table for comparison.

Section	Straight Forward		Fast	
	Cycle Count	**Program Words**	**Cycle Count**	**Program Words**
power up	0	0	0	0
init	13	13	6	6
kernel	21661	22	1245	1142
term	0	0	0	0

Table 6-2 Performance of the straight forward implementation

The fast implementation's cycle count is 17.4 times smaller than the straight forward one (94.3% reduction).

7. References

[1] W. B. Pennebaker and J. L. Mitchell, Jpeg: Still Image Data Compression Standard, Chapter 4 "The Discrete Cosine Transform", pp. 29-64.

[2] P. Dunhamel and H. H'Mida, "New 2^n DCT algorithms suitable for VLSI implementation," in *Proc. ICASSP-87*, pp. 1805-1808.

[3] C. Leoffler, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1D DCT algorithms with 11 mutiplications," in *Proc. IEEE ICASSP*, Feb. 1989, vol. 2, pp. 988-991.

[4] E. Feig and S. Winograd, "On the multiplicative complexity of Discrete cosine transforms," *IEEE Trans. Inform. Thoery*, vol. 38, pp. 1387-1391, July 1992.

[5] B. G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 1243-1245, Dec. 1984.

[6] H. S. Hou, "A fast recursive algorithm for computing the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 1455-1461, Oct. 1987.

[7] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," *IEEE Trans. Signal Processing*, vol. 40, pp. 2174-2193, Sep. 1992.

[8] H. R. Wu and D. Tan, "Implementation of Cho and Lee's 2D DCT algorithm using LLM 1D DCT algorithm," in *Proc. 1997 IEEE Int. Workshop on Intelligent Signal Processing and Communication Systems*, Nov. 1997, Sect. 24, pp. 2.1-2.5.

[9] H. R. Wu and Z. Man, "Comments on "Fast Algorithms and Implementation of 2-D Discrete Cosine Transform," *IEEE Trans. Circuits and Systems Video Tech.*, vol. 8, pp. 128-129, Apr. 1998.

Appendix A: Tools

Some of the tools, described in Table 5-3 were generated for this project, for convenience. These are presented here.

A.1. shift.pl

```
#!/usr/gnu/bin/perl
if (!$ARGV[2])
{
   print "\n=============================================\n";
   print "\nInteger to Fractional Presentation Convertor\n";
   print "\n=============================================\n";
   print "\nOperation:  int2frc.pl  <bits>  <data file>  <out file>\n";
   print "Output: each number will be divided by 2 in power of bits\n\n";
   print "$a\n";
}
else
{
   $bits = $ARGV[0];
   $data_file = $ARGV[1];
   $result = '>'.$ARGV[2];
   open(FHI,$data_file);
   open(FHO,$result);
   @input = <FHI>;
   close FHI;
   while(@input[$i])
   {
      chop $input[$i];
      $x=$input[$i]*(2**$bits);
      print FHO "$x\n";
      $i++;
   }
   close FHO;
}
```

A.2. cycles.awk

```
#!/usr/gnu/bin/gawk -f

BEGIN{
   y[1] = "power  : "
   y[2] = "init   : "
   y[3] = "kernel : "
   y[4] = "term   : "
   n = 0
   x = 0
}
/Break/{
   z = x
   x = substr($7,5,length($7)-4)
   if (n >= 1) { print y[n],x-z }
   n = n+1
}

END{}
```

Appendix B: Straight Forward Implementation

The sources, as well as the outputs, of the straight forward implementation are given here.

B.1. Sources

All the sources are given here as is, i.e. copying of the file into the document. Sometimes this format is less convenient for reading but it is more convenient to retry the simulation.

B.1.1 dct

```
#!/bin/tcsh -fbx
shift.pl -11 dec_input.sf input.sf
sim56300 dct.cmd
shift.pl 11 output.sf dec_output.sf
```

B.1.2 dct.cmd

```
load dct.cld
log s dct.log -o
input y:$ffc0 input.sf -rf
output y:$ffc1 output.sf -rf -o
change pc start
break p:power s
break p:init s
break p:kernel s
break p:term s
break p:end
go
quit
```

B.1.3 dct.asm

```
comment ~

------------------------------------------------------------
Discrete Cosine Transform (DCT) 8x8 on Motorola DSP56300
Straight Forward Implementation  of  the  DCT  algorithm
V1.0                                July 12, 1998
------------------------------------------------------------

This version is the straight, clear way to implement the algorithm.
There was no attemot to reach high performance in cycle count nor
program memory usage.

Operating Modes
---------------
There are two operating modes, controled by the parameter TEST.
TEST = 0: Measure the performance. No accurate results.
TEST = 1: Test the functionality. The results should be accurate.

Performance
-----------
            cycles    program words
            ------    -------------
power up:        0            0
init:           13           13
kernel:      21661           22
term:            0            0

Memory Map
----------
x: 0 .. 63   s(y=0,x=0) s(y=0,x=1) ... s(y=7,x=7)
x:64 ..127   S(v=0,u=0) S(v=1,u=0) ... S(v=7,u=7)
y: 0 .. 63   C(u=0,x=0) C(u=0,x=1) ... C(u=7,x=7)

comment end here ~

      include 'sincos'

TEST    equ     1
start   equ     $100
```

```
points  equ    8
data    equ    0
coef    equ    0
res     equ    64

        sincos  points,coef

        org     p:start
        nop
        nop
        nop
power
        if      TEST                    ; read in data
        move    #data,r0                ; to X memory
        move    r0,r2
        do      #points*points,readloop
        move    y:$ffc0,x0
        move    x0,x:(r0)+
readloop
        nop
        endif

init
        move    #-9,n0
        move    #-9,n1
        move    #-65,n4
        move    #coef,r0                ; r0 points to C(0,0)
        move    #coef,r1                ; r1 points to C(0,0)
        move    #data,r4                ; r4 pointes to s(0,0)
        move    #res,r5                 ; r5 points to S(0,0) (1st result)
kernel
        do      #8,_uloop
        do      #8,_vloop
        bsr     Suv                     ; calculate S(u,v)
        move    b,x:(r5)+               ; write result
_vloop
        lea     (r0+8),r0               ; u=u+1
        lea     (r1-64),r1              ; v=0
_uloop
term
        if      TEST
        move    #res,r0
        do      #points*points,writeloop        ; write output
        move    x:(r0)+,x0
        move    x0,y:$ffc1
writeloop
        nop
        endif
end:
done    jmp     done

; Subroutine: Suv
;
; Description: Calculates one DCT element
;
; INPUTS
; r0 points to .5*cos(u*pi/16) (or to .5/sqr(2) if u=0)
; r1 points to .5*cos(v*pi/16) (or to .5/sqr(2) if v=0)
; r4 points to s(0,0)
;
; OUTPUT
; b = S(v,u)
;
; RETURN VALUES
; r0 points to .5*cos(u*pi/16)
; r1 points to .5*cos((v+1)*pi/16)
; r4 points to s(0,0)

Suv
        clr     b               x:(r4)+,x1      y:(r1)+,y0      ; s(0,0) -> x1
        do      #8,_yloop                                       ; C(0,v) -> y0
        move    y:(r0)+,x0                                      ; C(x,u) -> x0
        do      #8,_xloop
        mpy     x0,y0,a         y:(r0)+,x0                      ; C(x,u)C(y,v) -> a
        move    a,y1                                            ; new C(x,u) -> x0
        mac     x1,y1,b         x:(r4)+,x1                      ; b=b+s(x,y)C(x,u)C(y,v)
_xloop
        move    (r0)+n0                                         ; r0 -> x=0
        move    y:(r1)+,y0                                      ; new C(y,v) -> y0
_yloop
        move    x:(r4)+n4,x0    y:(r1)-,y1                      ; dummy moves to update
```

```
        rts
```

B.1.4 sincos.asm

```
;---------------------------------------------------------------
;Discrete Cosine Transform (DCT) 8x8 on Motorola DSP56300
;Coefficients          Table          Generator
;V1.0                              July 12, 1998
;---------------------------------------------------------------
;
sincos  macro   points,coef
sincos  ident   1,2
;
;       sincos  -       macro to generate DCT coefficient
;                       lookup table
;
;       points  -       number of points (=8)
;       coef    -       base address of sine/cosine table
;                       negative cosine value in X memory
;                       negative sine value in Y memory
;

pi      equ     3.141592654

        org     y:coef
countx  set     0
countu  set     0
cu      set     0.70710678118654746172
        dup     points
        dup     points
        dc      cu*.5*@cos((2*countx+1)*pi*countu/16)
countx  set     countx+1
        endm
countx  set     0
countu  set     countu+1
cu      set     1
        endm

        endm    ;end of sincos macro
```

B.1.5 dec_input.sf

```
-256
-183
-110
-37
36
109
182
255
-250
-230
-190
-130
-50
50
170
255
10
15
99
200
70
0
-100
-40
20
30
200
240
88
0
-100
-40
200
180
150
109
```

17
0
-100
-40
171
117
73
-44
0
0
-133
-71
100
81
60
-96
-33
50
-150
-180
0
56
-35
-73
-14
-8
-15
-203

B.2. Results

B.2.1 dec_output.sf

60.2499072
30.6653184
-265.711616
32.684032
104.2499584
57.5725568
64.0595968
64.5105664
62.502912
-633.8719744
-482.9831168
-75.3975296
-3.0236672
122.7698176
129.8239488
154.1562368
-99.9729152
-17.8528256
120.5692416
173.7756672
-75.119616
-155.7409792
-65.57696
-15.0818816
-15.4652672
-179.158016
116.9537024
39.2527872
-52.9504256
-30.5979392
-4.5422592
19.5620864
-6.750208
116.8553984
-115.695616
-18.4918016
-6.750208
6.366208
38.180864
-4.862976
-38.6295808
50.0412416
-20.697088
-67.4746368
36.931584

-1.1567104
0.1736704
-7.2302592
78.7523584
-18.5350144
-74.0769792
49.2724224
-53.3112832
66.870272
-27.5695616
5.419008
-31.3821184
-23.2204288
41.9014656
13.802496
-0.2195456
-6.5009664
-5.9592704
-33.7246208

Appendix C: Fast Implementation

The sources, as well as the outputs, of the straight forward implementation are given here.

C.1. Sources

All the sources are given here as is, i.e. copying of the file into the document. Sometimes this format is less convenient for reading but it is more convenient to retry the simulation.

The scripts *dct* and *dct.cmd* are the same as in B.1.1 and B.1.2 respectively. One modification was applied to *dct* as follows: Instead of 11 there should be 12 (prescaling the data one more bit down in order to prevent overflow).

C.1.1 dct.asm

```
comment ~

------------------------------------------------------------
Discrete Cosine Transform (DCT) 8x8 on Motorola DSP56300
Feig  Scaled  2D  DCT  Algorithm  Implementation
Author: Dror Halahmi      V1.0      Agust 27, 1998
------------------------------------------------------------

This version is an implementation of Feig DCT 8x8 algorithm.
There was an attempt to hav a minimum cycle count in penalty
of program memory, for example.

Operating Modes
---------------
There are two operating modes, controled by the parameter TEST.
TEST = 0: Measure the performance. No accurate results.
TEST = 1: Test the functionality. The results should be accurate.

Performance
-----------
            cycles    program words
            ------    -------------
power up:        0            0
init:            6            6
kernel:       1245         1142
term:            0            0

Memory Map
----------
x:  0 .. 31   s(0,0) s(1,0) ... s(3,0) s(0,1) ... s(3,3) s(4,4) ... s(7,7)
y:  0 .. 31   s(4,0) s(5,0) ... s(7,0) s(4,1) ... s(7,3) s(0,4) ... s(3,7)
x:128 .. 141  a0 a1 a2 a3 b1 b2 b3 a1 a2 a3 b1 b2 b3 c0
y:128 .. 141  a0 a1 a2 a3 b1 b2 b3 a1 a2 a3 b1 b2 b3 c0

comment end here ~

TEST    equ     1
start   equ     $100
points  equ     8
data    equ     0
coef    equ     $80
res     equ     $100

        include 'coefs.asm'
        include 'R1rowsXY.asm'
        include 'R1rowsYX.asm'
        include 'R1columnsXY.asm'
        include 'R1columnsYX.asm'
        include 'M1XY.asm'
        include 'M1YX.asm'
        include 'M2XY.asm'
        include 'M2YX.asm'
        include 'M3YX.asm'
        include 'R2rowsXY.asm'
        include 'R2rowsYX.asm'
        include 'R2columnsXY.asm'
```

```
        include 'R2columnsYX.asm'


        org     p:start
        nop
        nop
        nop
power
        if      TEST                    ; read in data
        move    #data,r0                ; to X memory
        move    r0,r2
        do      #points*points/16,readloop1
        move    y:$ffc0,x0
        move    x0,x:(r0)+
        move    y:$ffc0,x0
        move    x0,x:(r0)+
        move    y:$ffc0,x0
        move    x0,x:(r0)+
        move    y:$ffc0,x0
        move    x0,x:(r0)+
        move    y:$ffc0,x0
        move    x0,y:(r2)+
        move    y:$ffc0,x0
        move    x0,y:(r2)+
        move    y:$ffc0,x0
        move    x0,y:(r2)+
        move    y:$ffc0,x0
        move    x0,y:(r2)+
readloop1
        do      #points*points/16,readloop2
        move    y:$ffc0,x0
        move    x0,y:(r2)+
        move    y:$ffc0,x0
        move    x0,y:(r2)+
        move    y:$ffc0,x0
        move    x0,y:(r2)+
        move    y:$ffc0,x0
        move    x0,y:(r2)+
        move    y:$ffc0,x0
        move    x0,x:(r0)+
        move    y:$ffc0,x0
        move    x0,x:(r0)+
        move    y:$ffc0,x0
        move    x0,x:(r0)+
        move    y:$ffc0,x0
        move    x0,x:(r0)+
readloop2
        nop
        endif

init
        move    #data,r0                ; r0->s(0,0)
        move    #data+$1c,r7            ; r7->s(0,7)
        move    #4,n0                   ; offset between s(i,j) and s(i,j+1)
        move    n0,n7
        move    n0,n1
        move    n0,n6
kernel
        R1columnsXY                     ; R1 preadditions stage on columns
        R1columnsXY                     ; R1 preadditions stage on columns
        R1columnsXY                     ; R1 preadditions stage on columns
        R1columnsXY                     ; R1 preadditions stage on columns

        move    (r0)-n0                 ; r0->s(4,0)
        move    (r7)-n7                 ; r7->s(4,7)

        R1columnsYX                     ; R1 preadditions stage on columns
        R1columnsYX                     ; R1 preadditions stage on columns
        R1columnsYX                     ; R1 preadditions stage on columns
        R1columnsYX                     ; R1 preadditions stage on columns

        move    #data+$3,r7             ; r7->s(7,0)
        move    (r0)-n0                 ; r0->s(0,0)
        move    #2,n0
        move    n0,n7
        move    n0,n1

        R1rowsXY                        ; R1 preadditions stage on rows
        R1rowsXY                        ; R1 preadditions stage on rows
        R1rowsXY                        ; R1 preadditions stage on rows
        R1rowsXY                        ; R1 preadditions stage on rows
```

```
        R1rowsYX                ; R1 preadditions stage on rows
        R1rowsYX                ; R1 preadditions stage on rows
        R1rowsYX                ; R1 preadditions stage on rows
        R1rowsYX                ; R1 preadditions stage on rows

        move    #data+3,r0      ; r0->s(3,0)
        move    #coef,r7        ; r7->a0
        move    #-2,n0
        ori     #8,mr           ; Scale up for M1 and M2

        M1XY                    ; M1 of mutiplication stage
        M1XY                    ; M1 of mutiplication stage
        M1XY                    ; M1 of mutiplication stage

        lua     (r0-3),r0       ; r0->s(0,3)
        move    r0,r4
        move    #coef+4,r1      ; r1->b1

        M2XY                    ; M2 of mutiplication stage

        lua     (r0+3),r0       ; r0->s(3,4)

        M1YX                    ; M1 of mutiplication stage

        lua     (r0-3),r0       ; r0->s(0,5)
        move    r0,r4

        M2YX                    ; M2 of mutiplication stage

        move    #2,n0
        move    (r7)+           ; r7->a1
        move    n0,n7
        move    n0,n1
        lua     (r0+4),r1       ; r1->s(0,7)

        M3YX

        move    #data+2,r0      ; r0->s(2,0)
        move    #data,r7        ; r7->s(4,0)

        R2rowsXY                ; R2 postadditions stage on rows
        R2rowsXY                ; R2 postadditions stage on rows
        R2rowsXY                ; R2 postadditions stage on rows
        R2rowsXY                ; R2 postadditions stage on rows

        R2rowsYX                ; R2 postadditions stage on rows
        R2rowsYX                ; R2 postadditions stage on rows
        R2rowsYX                ; R2 postadditions stage on rows
        R2rowsYX                ; R2 postadditions stage on rows

        move    #data+8,r0      ; r0->s(0,2)
        move    #data+16,r7     ; r7->s(0,4)
        move    #4,n0
        move    n0,n6
        move    n0,n7

        R2columnsXY             ; R2 postadditions stage on columns
        R2columnsXY             ; R2 postadditions stage on columns
        R2columnsXY             ; R2 postadditions stage on columns
        R2columnsXY             ; R2 postadditions stage on columns

        move    (r0)-n0         ; r0->s(4,2)
        move    (r7)-n7         ; r7->s(4,4)

        R2columnsYX             ; R2 postadditions stage on columns
        R2columnsYX             ; R2 postadditions stage on columns
        R2columnsYX             ; R2 postadditions stage on columns
        R2columnsYX             ; R2 postadditions stage on columns

        nop
term
        if      TEST
        move    #data,r0
        move    r0,r2
        do      #points*points/16,writeloop1    ; write output
        move    x:(r0)+,x0
        move    x0,y:$ffc1
        move    x:(r0)+,x0
        move    x0,y:$ffc1
        move    x:(r0)+,x0
```

```
        move    x0,y:$ffc1
        move    x:(r0)+,x0
        move    x0,y:$ffc1
        move    y:(r2)+,x0
        move    x0,y:$ffc1
        move    y:(r2)+,x0
        move    x0,y:$ffc1
        move    y:(r2)+,x0
        move    x0,y:$ffc1
        move    y:(r2)+,x0
        move    x0,y:$ffc1
writeloop1
        do      #points*points/16,writeloop2    ; write output
        move    y:(r2)+,x0
        move    x0,y:$ffc1
        move    y:(r2)+,x0
        move    x0,y:$ffc1
        move    y:(r2)+,x0
        move    x0,y:$ffc1
        move    y:(r2)+,x0
        move    x0,y:$ffc1
        move    x:(r0)+,x0
        move    x0,y:$ffc1
        move    x:(r0)+,x0
        move    x0,y:$ffc1
        move    x:(r0)+,x0
        move    x0,y:$ffc1
        move    x:(r0)+,x0
        move    x0,y:$ffc1
writeloop2
        nop
        endif
end:
done    jmp     done
```

C.1.2 coefs.asm

```
pi      equ     3.141592654

        org     x:coef

        dc      .5*@sin(pi/4)                   ; a0=sqr(2)/2
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))    ; a1=gama(6)-gama(2)
        dc      -.25*@cos(pi/8)                 ; a2=-gama(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))    ; a3=gama(6)+gama(2)
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))*@sin(pi/4) ; b1=a1/sqr(2)
        dc      -.25*@cos(pi/8)*@sin(pi/4)      ; b2=a2/sqr(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))*@sin(pi/4) ; b3=a3/sqr(2)
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))    ; a1=gama(6)-gama(2)
        dc      -.25*@cos(pi/8)                 ; a2=-gama(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))    ; a3=gama(6)+gama(2)
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))*@sin(pi/4) ; b1=a1/sqr(2)
        dc      -.25*@cos(pi/8)*@sin(pi/4)      ; b2=a2/sqr(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))*@sin(pi/4) ; b3=a3/sqr(2)
        dc      .5*@sin(pi/4)                   ; c0=sqr(2)/4

        org     y:coef

        dc      .5*@sin(pi/4)                   ; a0=sqr(2)/2
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))    ; a1=gama(6)-gama(2)
        dc      -.25*@cos(pi/8)                 ; a2=-gama(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))    ; a3=gama(6)+gama(2)
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))*@sin(pi/4) ; b1=a1/sqr(2)
        dc      -.25*@cos(pi/8)*@sin(pi/4)      ; b2=a2/sqr(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))*@sin(pi/4) ; b3=a3/sqr(2)
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))    ; a1=gama(6)-gama(2)
        dc      -.25*@cos(pi/8)                 ; a2=-gama(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))    ; a3=gama(6)+gama(2)
        dc      .5*(@cos(3*pi/8)-@cos(pi/8))*@sin(pi/4) ; b1=a1/sqr(2)
        dc      -.25*@cos(pi/8)*@sin(pi/4)      ; b2=a2/sqr(2) (1/2 of this due to scaling)
        dc      .5*(@cos(3*pi/8)+@cos(pi/8))*@sin(pi/4) ; b3=a3/sqr(2)
        dc      .5*@sin(pi/4)                   ; c0=sqr(2)/4
```

C.1.3 R1rowsXY.asm

```
;*************************************************************************
;*                                                                       *
;*  MODULE NAME -   R1rowsXY                                             *
;*                                                                       *
```

```
;* INPUT      -  s(0)..s(7) are as follows:                          *
;*              X:s(0) s(1) s(2) s(3)                              *
;*              Y:s(4) s(5) s(6) s(7)                              *
;*              r0 - pointer to s(0) in X memory                   *
;*              r7 - pointer to s(7) in Y memory                   *
;*              n0=2                                               *
;*              n1=2                                               *
;*              n7=2                                               *
;*                                                                 *
;* OUTPUT     -  s(0)..s(7) are replaced by R1 values              *
;*              r0 ponits to next row (r0+4)                       *
;*              r7 ponits to next row (r7+4)                       *
;*                                                                 *
;* SUBROUTINES                                                    *
;* CALLED     -  none                                             *
;*                                                                 *
;* MACROS USED -  none                                            *
;*                                                                 *
;* REGISTERS  -  r1,r6,x0,y0,y1,a,b                               *
;* CORRUPTED                                                      *
;*                                                                 *
;*******************************************************************************
;* CHANGE HISTORY                                                 *
;* dd/mm/yy  Code Ver  Description                    Author      *
;* --------  --------  -----------                    ------      *
;* 27/08/98  1.00      Initial version                Dror Halahmi *
;*******************************************************************************
;*                                                                 *
;* FUNCTION :  Calculates R1 preadditions stage                   *
;*                                                                 *
;*******************************************************************************

R1rowsXY        macro

      move              x:(r0)+,b       y:(r7),a        ; b=s0 a=s7

      sub     a,b       r7,r6                   ; b=s0-s7
      addl    b,a
      move              x:(r0)-,b       b,y:(r7)-       ; a=s0+s7 b=s1 t7=s0-s7 r0->s0 r7->s6
      move              a,x:(r0)+n0     y:(r7),a        ; t0=s0+s7 a=s6 r0->s2

      sub     a,b       r0,r1                   ; b=s1-s6
      addl    b,a
      move              x:(r0)-,b       b,y:(r7)-       ; a=s1+s6 b=s2 t6=s1-s6 r0->s1 r7->s5
      move              a,x:(r0)+n0     y:(r7)-,a       ; t1=s1+s6 a=s5 r0->s3 r7-s4

      sub     b,a       (r1)-n1                 ; a=s5-s2
      addl    a,b
      move              x:(r0)-,a       a,y1            ; b=s2+s5 a=s3 y1=s5-s2 r0->s2
      move              b,x:(r0)        y:(r7),b        ; t2=s2+s5 b=s4 r0->s2

      sub     b,a                               ; a=s3-s4
      addl    a,b       x:(r1)+,a       a,y:(r7)        ; b=s3+s4 a=t0 t4=s3-s4 r1->t1 r7->s4

      sub     b,a                               ; a=t0-t3
      addl    a,b                               ; b=t0+t3
      move              x:(r1)-,a       a,y0            ; a=t1 r1->t0
      move              b,x:(r1)                ; u0=t0+t3 r1->t0
      move              x:(r0)+,b               ; b=t2 r0->t3
      sub     b,a       y0,x:(r0)-              ; a=t1-t2 u3=t0-t3 r0->t2
      addl    a,b                               ; b=t1+t2
      move              x:(r1),a        a,y0            ; a=u0 y0=t1-t2 r1->u0
      sub     a,b       y0,x:(r0)-              ; b=u1-u0 u2=t1-t2 r0->u1
      addl    b,a                               ; a=u1+u0
      move              b,x:(r0)+n0             ; v1=u1-u0 r0->u3
      move              x:(r0),b                ; b=u3
      sub     y0,b      a,x:(r1)+n1     y:(r6)-,a       ; a=u3-u2 v0=u1+u0 a=u7 r1->u2 r6->u6
      move              b,x:(r0)+       y:(r7),b        ; v3=u3-u2 a=u4 r7->u4 r0->next
      sub     a,b                       y:(r6),y0       ; a=u4-u7 y0=u6
      add     y1,a                      y:(r7)+,x0      ; a=u5+u7 x0=u4 r7->u5
      move              y0,b            b,y:(r7)-       ; v5=u4-u7 r7->u4 a=u6
      add     x0,b                      y1,y:(r7)+n7    ; b=u4+u6 v4=u5 r7->u6
      move                              a,y:(r7)+       ; v6=u5+u7 r7->u7
      move                              b,y:(r7)+n7     ; v7=u4+u6 r7->next5
      move                              (r7)+n7         ; r7->next7

      endm
```

C.1.4 R1rowsYX.asm

```
;*****************************************************************************
;*                                                                           *
;* MODULE NAME -   R1rowsYX                                                  *
;*                                                                           *
;* INPUT       -   s(0)..s(7) are as follows:                                *
;*                 Y:s(0) s(1) s(2) s(3)                                     *
;*                 X:s(4) s(5) s(6) s(7)                                     *
;*                 r0 - pointer to s(0) in Y memory                          *
;*                 r7 - pointer to s(7) in X memory                          *
;*                 n0=2                                                      *
;*                 n1=2                                                      *
;*                 n7=2                                                      *
;*                                                                           *
;* OUTPUT      -   s(0)..s(7) are replaced by R1 values                      *
;*                 r0 ponits to next row (r0+4)                              *
;*                 r7 ponits to next row (r7+4)                              *
;*                                                                           *
;* SUBROUTINES                                                               *
;* CALLED      -   none                                                      *
;*                                                                           *
;* MACROS USED -   none                                                      *
;*                                                                           *
;* REGISTERS   -   r1,r6,y0,x0,x1,a,b                                        *
;* CORRUPTED                                                                 *
;*                                                                           *
;*****************************************************************************
;* CHANGE HISTORY                                                            *
;* dd/mm/yy  Code Ver  Description                         Author            *
;* --------  --------  -----------                         ------            *
;* 27/08/98   1.00     Initial version                     Dror Halahmi      *
;*****************************************************************************
;*                                                                           *
;* FUNCTION :  Calculates R1 preadditions stage                              *
;*                                                                           *
;*****************************************************************************

R1rowsYX        macro

        move            y:(r0)+,b       x:(r7),a        ; b=s0 a=s7

        sub     a,b     r7,r6                           ; b=s0-s7
        addl    b,a
        move            y:(r0)-,b       b,x:(r7)-       ; a=s0+s7 b=s1 t7=s0-s7 r0->s0 r7->s6
        move            a,y:(r0)+n0     x:(r7),a        ; t0=s0+s7 a=s6 r0->s2

        sub     a,b     r0,r1                           ; b=s1-s6
        addl    b,a
        move            y:(r0)-,b       b,x:(r7)-       ; a=s1+s6 b=s2 t6=s1-s6 r0->s1 r7->s5
        move            a,y:(r0)+n0     x:(r7)-,a       ; t1=s1+s6 a=s5 r0->s3 r7-s4

        sub     b,a     (r1)-n1                         ; a=s5-s2
        addl    a,b
        move            y:(r0)-,a       a,x1            ; b=s2+s5 a=s3 x1=s5-s2 r0->s2
        move            b,y:(r0)        x:(r7),b        ; t2=s2+s5 b=s4 r0->s2

        sub     b,a                                     ; a=s3-s4
        addl    a,b     y:(r1)+,a       a,x:(r7)        ; b=s3+s4 a=t0 t4=s3-s4 r1->t1 r7->s4

        sub     b,a                                     ; a=t0-t3
        addl    a,b                                     ; b=t0+t3
        move            y:(r1)-,a       a,x0            ; a=t1 r1->t0
        move            b,y:(r1)                        ; u0=t0+t3 r1->t0
        move            y:(r0)+,b                       ; b=t2 r0->t3
        sub     b,a     x0,y:(r0)-                      ; a=t1-t2 u3=t0-t3 r0->t2
        addl    a,b                                     ; b=t1+t2
        move            y:(r1),a        a,x0            ; a=u0 x0=t1-t2 r1->u0
        sub     a,b     x0,y:(r0)-                      ; b=u1-u0 u2=t1-t2 r0->u1
        addl    b,a                                     ; a=u1+u0
        move            b,y:(r0)+n0                     ; v1=u1-u0 r0->u3
        move            y:(r0),b                        ; b=u3
        sub     x0,b    a,y:(r1)+n1     x:(r6)-,a       ; a=u3-u2 v0=u1+u0 a=u7 r1->u2 r6->u6
        move            b,y:(r0)+       x:(r7),b        ; v3=u3-u2 a=u4 r7->u4 r0->next
        sub     a,b                     x:(r6),x0       ; a=u4-u7 x0=u6
        add     x1,a                    x:(r7)+,y0      ; a=u5+u7 y0=u4 r7->u5
        move            x0,b            b,x:(r7)-       ; v5=u4-u7 r7->u4 a=u6
        add     y0,b                    x1,x:(r7)+n7    ; b=u4+u6 v4=u5 r7->u6
        move                            a,x:(r7)+       ; v6=u5+u7 r7->u7
        move                            b,x:(r7)+n7     ; v7=u4+u6 r7->next5
```

```
    move                    (r7)+n7     ; r7->next7

    endm
```

C.1.5 R1columnsXY.asm

```
;*************************************************************************
;*                                                                       *
;* MODULE NAME -  R1columnsXY                                            *
;*                                                                       *
;* INPUT      -  s(0)..s(7) are as follows:                              *
;*               X:s(0)...s(1)...s(2)...s(3)                             *
;*               Y:s(4)...s(5)...s(6)...s(7)                             *
;*               r0 - pointer to s(0) in X memory                        *
;*               r7 - pointer to s(7) in Y memory                        *
;*               n0=4                                                    *
;*               n1=4                                                    *
;*               n6=4                                                    *
;*               n7=4                                                    *
;*                                                                       *
;* OUTPUT     -  s(0)..s(7) are replaced by R1 values                    *
;*               r0 ponits to next column (r0+1)                         *
;*               r7 ponits to next column (r7+1)                         *
;*                                                                       *
;* SUBROUTINES                                                           *
;* CALLED     -  none                                                    *
;*                                                                       *
;* MACROS USED -  none                                                   *
;*                                                                       *
;* REGISTERS  -  r1,r6,x0,y0,y1,a,b                                      *
;* CORRUPTED                                                             *
;*                                                                       *
;*************************************************************************
;* CHANGE HISTORY                                                        *
;* dd/mm/yy  Code Ver  Description                    Author             *
;* --------  --------  -----------                    ------             *
;* 27/08/98  1.00      Initial version                Dror Halahmi       *
;*************************************************************************
;*                                                                       *
;* FUNCTION :  Calculates preaddition R1 block                           *
;*                                                                       *
;*************************************************************************

R1columnsXY     macro

    move            x:(r0)+n0,a     y:(r7),b        ; a=s0 b=s7

    sub     b,a     r7,r6                   ; a=s0-s7
    addl    a,b     x:(r0)-n0,y0            ; b=s0+s7 y0=s1 r0->s0
    move            y0,a        a,y:(r7)-n7 ; a=s1 t7=s0-s7 r7->s6
    move            r0,r1
    move            b,x:(r0)+n0     y:(r7),b        ; t0=s0+s7 b=s6 r0->s2

    sub     b,a     (r0)+n0                 ; a=s1-s6
    addl    a,b     x:(r0)-n0,y0            ; b=s1+s6 y0=s2 r0->s1
    move            y0,a        a,y:(r7)-n7 ; a=s2 t6=s1-s6 r7->s5
    move            b,x:(r0)+n0     y:(r7),b        ; t1=s1+s6 b=s5 r0->s3

    sub     b,a     (r0)+n0                 ; a=s2-s5
    addl    a,b     x:(r0)-n0,y0            ; b=s2+s5 y0=s3 r0->s2
    move            y0,a        a,y:(r7)-n7 ; a=s3 t5=s2-s5 r7->s4
    move            b,x:(r0)+n0     y:(r7),b        ; t2=s2+s5 b=s4 r0->s3

    sub     b,a                             ; a=s3-s4
    addl    a,b     x:(r1),a        a,y:(r7)+n7     ; b=s3+s4 a=t0 t4=s3-s4 r7->s5

    sub     b,a                             ; a=t0-t3
    addl    a,b                             ; b=t0+t3
    move            a,x:(r0)-n0             ; u3=t0-t3 r0->t2
    move            b,x:(r1)+n1             ; u0=t0+t3 r1->t1
    move            x:(r1),a                ; a=t1
    move            x:(r0),b                ; b=t2
    sub     b,a                             ; a=t1-t2
    addl    a,b     a,x:(r0)-n0             ; b=t1+t2 u2=t1-t2 r0->u1
    move                    y:(r7)-n7,x0    ; x0=t5 r7->t4
    move            b,x:(r1)-n1     x0,b            ; u1=t1+t2 b=t5 r1->u0
    neg     b       x:(r0),a                ; b=-t5 a=u1
    move            x:(r1),b        b,y1            ; y1=-t5 b=u0
    sub     b,a                             ; a=u1-u0
    addl    a,b                             ; b=u1+u0
```

```
    move            a,x:(r0)-n0             ; v1=u1-u0 r0->v0
    move            b,x:(r1)+n1             ; v0=u1+u0 r1->u1
    move            (r1)+n1                 ; r1->u2
    move            x:(r1)+n1,b             ; b=u2 r1->u3
    move            x:(r1),a                ; a=u3
    sub   b,a                   y:(r6)-n6,b     ; a=u3-u2 b=u7 r6->u6
    move            a,x:(r1)        y:(r7)+n7,a     ; v3=u3-u2 a=u4 r7->u5
    sub   b,a       x:(r0)+,x0      y:(r6),y0       ; a=u4-u7 y0=u6 r0->next0
    move            y0,a            a,y:(r7)-n7     ; v5=u4-u7 r7->u4 a=u6
    add   y1,b                      y:(r7),x0       ; b=u5+u7 x0=u4
    add   x0,a                      y1,y:(r7)+n7    ; a=u4+u6 v4=u5 r7->u6
    move                            (r7)+n7
    move                            b,y:(r7)+n7     ; v6=u5+u7 r7->u7
    move                            a,y:(r7)+       ; v7=u4+u6 r7->next7

    endm
```

C.1.6 R1columnsYX.asm

```
;*************************************************************************
;*                                                       *
;* MODULE NAME -   R1columnsYX                                   *
;*                                                       *
;* INPUT       -   s(0)..s(7) are as follows:                    *
;*             Y:s(0)...s(1)...s(2)...s(3)                   *
;*             X:s(4)...s(5)...s(6)...s(7)                   *
;*             r0 - pointer to s(0) in Y memory              *
;*             r7 - pointer to s(7) in X memory              *
;*             n0=4                                          *
;*             n1=4                                          *
;*             n6=4                                          *
;*             n7=4                                          *
;*                                                       *
;* OUTPUT      -   s(0)..s(7) are replaced by R1 values          *
;*             r0 ponits to next column (r0+1)               *
;*             r7 ponits to next column (r7+1)               *
;*                                                       *
;* SUBROUTINES                                           *
;* CALLED      -   none                                  *
;*                                                       *
;* MACROS USED -   none                                      *
;*                                                       *
;* REGISTERS   -   r1,r6,y0,x0,x1,a,b                        *
;* CORRUPTED                                             *
;*                                                       *
;*************************************************************************
;* CHANGE HISTORY                                            *
;* dd/mm/yy  Code Ver  Description                   Author     *
;* --------  --------  -----------                   ------     *
;* 27/08/98   1.00     Initial version                 Dror Halahmi  *
;*************************************************************************
;*                                                       *
;* FUNCTION :  Calculates preaddition R1 block                   *
;*                                                       *
;*************************************************************************

R1columnsYX     macro

    move            y:(r0)+n0,a     x:(r7),b        ; a=s0 b=s7

    sub   b,a       r7,r6                   ; a=s0-s7
    addl  a,b       y:(r0)-n0,x0            ; b=s0+s7 x0=s1 r0->s0
    move            x0,a            a,x:(r7)-n7     ; a=s1 t7=s0-s7 r7->s6
    move            r0,r1
    move            b,y:(r0)+n0     x:(r7),b        ; t0=s0+s7 b=s6 r0->s2

    sub   b,a       (r0)+n0                 ; a=s1-s6
    addl  a,b       y:(r0)-n0,x0            ; b=s1+s6 x0=s2 r0->s1
    move            x0,a            a,x:(r7)-n7     ; a=s2 t6=s1-s6 r7->s5
    move            b,y:(r0)+n0     x:(r7),b        ; t1=s1+s6 b=s5 r0->s3

    sub   b,a       (r0)+n0                 ; a=s2-s5
    addl  a,b       y:(r0)-n0,x0            ; b=s2+s5 x0=s3 r0->s2
    move            x0,a            a,x:(r7)-n7     ; a=s3 t5=s2-s5 r7->s4
    move            b,y:(r0)+n0     x:(r7),b        ; t2=s2+s5 b=s4 r0->s3

    sub   b,a                               ; a=s3-s4
    addl  a,b       y:(r1),a        a,x:(r7)+n7     ; b=s3+s4 a=t0 t4=s3-s4 r7->s5

    sub   b,a                               ; a=t0-t3
```

```
    addl    a,b                             ; b=t0+t3
    move            a,y:(r0)-n0             ; u3=t0-t3 r0->t2
    move            b,y:(r1)+n1             ; u0=t0+t3 r1->t1
    move            y:(r1),a                ; a=t1
    move            y:(r0),b                ; b=t2
    sub     b,a                             ; a=t1-t2
    addl    a,b     a,y:(r0)-n0             ; b=t1+t2 u2=t1-t2 r0->u1
    move                    x:(r7)-n7,y0    ; y0=t5 r7->t4
    move            b,y:(r1)-n1     y0,b    ; u1=t1+t2 b=t5 r1->u0
    neg     b       y:(r0),a                ; b=-t5 a=u1
    move            y:(r1),b        b,x1    ; x1=-t5 b=u0
    sub     b,a                             ; a=u1-u0
    addl    a,b                             ; b=u1+u0
    move            a,y:(r0)-n0             ; v1=u1-u0 r0->v0
    move            b,y:(r1)+n1             ; v0=u1+u0 r1->u1
    move            (r1)+n1                 ; r1->u2
    move            y:(r1)+n1,b             ; b=u2 r1->u3
    move            y:(r1),a                ; a=u3
    sub     b,a             x:(r6)-n6,b     ; a=u3-u2 b=u7 r6->u6
    move            a,y:(r1)        x:(r7)+n7,a     ; v3=u3-u2 a=u4 r7->u5
    sub     b,a     y:(r0)+,y0      x:(r6),x0       ; a=u4-u7 x0=u6 r0->next0
    move            x0,a            a,x:(r7)-n7     ; v5=u4-u7 r7->u4 a=u6
    add     x1,b                    x:(r7),y0       ; b=u5+u7 y0=u4
    add     y0,a                    x1,x:(r7)+n7    ; a=u4+u6 v4=u5 r7->u6
    move                            (r7)+n7
    move                            b,x:(r7)+n7     ; v6=u5+u7 r7->u7
    move                            a,x:(r7)+       ; v7=u4+u6 r7->next7

    endm
```

C.1.7 M1XY.asm

```
;*************************************************************************
;*                                                                       *
;* MODULE NAME -   M1XY                                                  *
;*                                                                       *
;* INPUT       -   s(0)..s(7) are as follows:                            *
;*             X:s(0) s(1) s(2) s(3)                                     *
;*             Y:s(4) s(5) s(6) s(7)                                     *
;*             r0 - pointer to s(3) in X memory and s(7) in Y memory     *
;*             r7 - pointer to a0 both in X and Y memories               *
;*             n0=-2                                                     *
;*                                                                       *
;* OUTPUT      -   s(0)..s(7) are replaced by M1 values                  *
;*             r0 ponits to next row, s(3) (r0+4)                        *
;*             r7 reponits to a0 (r7)                                    *
;*                                                                       *
;* SUBROUTINES                                                           *
;* CALLED      -   none                                                  *
;*                                                                       *
;* MACROS USED -   none                                                  *
;*                                                                       *
;* REGISTERS   -   x0,x1,y0,y1,a,b                                       *
;* CORRUPTED                                                             *
;*                                                                       *
;*************************************************************************
;* CHANGE HISTORY                                                        *
;* dd/mm/yy  Code Ver  Description                       Author          *
;* --------  --------  -----------                       ------          *
;* 27/08/98   1.00     Initial version                   Dror Halahmi    *
;*************************************************************************
;*                                                                       *
;* FUNCTION :  Calculates M1 of multiplication stage                     *
;*                                                                       *
;*************************************************************************

M1XY    macro

    move            x:(r0)+n0,x0    y:(r7)+,y0      ; x0=s3 y0=a0 r0->s5 r7->a1
    mpy     x0,y0,a     x:(r7)+,x1      y:(r0)+,y1      ; a=a0*s3 x1=a1 y1=s5 r7->a2 r0->s6
    mpy     y0,y1,b                 y:(r0)+,y0      ; b=a0*s5 y0=s6 r0->s7
    mpy     x1,y0,a     a,x:(r0)+n0                 ; a=a1*s6 t3=a0*s3 r0->s5
    move                        b,y:(r0)-n0     ; t5=a0*s5 r0->s7
    move            x:(r7)+,x1      y:(r0),b        ; x1=a2 b=s7 r7->a3
    add     y0,b        x:(r7)-,x0      y:(r0)-,y0      ; b=s6+s7 x0=a3 y0=s7 r0->s6 r7->a2
    move            b,y1                    ; y1=s6+s7
    mpy     x1,y1,b     (r7)-                   ; b=a2*(s6+s7) r7->a1
    sub     b,a         (r7)-                   ; a=a1*s6-a2*(s6+s7) r7->a0
    mac     x0,y0,b                 a,y:(r0)+       ; b=a2*(s6+s7)+a3*s7 t6=a1*s6-a2*(s6+s7) r0->s7
    move                        b,y:(r0)-n0     ; t7=a2*(s6+s7)+a3*s7 r0->next1
```

```
    move                        (r0)-n0     ; r0->next3

    endm
```

C.1.8 M1YX.asm

```
;*************************************************************************
;*                                                                       *
;*  MODULE NAME -   M1YX                                                 *
;*                                                                       *
;*  INPUT       -   s(0)..s(7) are as follows:                           *
;*              Y:s(0) s(1) s(2) s(3)                                    *
;*              X:s(4) s(5) s(6) s(7)                                    *
;*              r0 - pointer to s(3) in Y memory and s(7) in X memory    *
;*              r7 - pointer to a0 both in X and Y memories              *
;*              n0=-2                                                    *
;*                                                                       *
;*  OUTPUT      -   s(0)..s(7) are replaced by M1 values                 *
;*              r0 ponits to next row, s(1) (r0+2)                       *
;*              r7 reponits to a0 (r7)                                   *
;*                                                                       *
;*  SUBROUTINES                                                          *
;*  CALLED      -   none                                                 *
;*                                                                       *
;*  MACROS USED -   none                                                 *
;*                                                                       *
;*  REGISTERS   -   y0,y1,x0,x1,a,b                                      *
;*  CORRUPTED                                                            *
;*                                                                       *
;*************************************************************************
;*  CHANGE HISTORY                                                       *
;*  dd/mm/yy  Code Ver  Description                 Author               *
;*  --------  --------  -----------                 ------               *
;*  27/08/98   1.00     Initial version             Dror Halahmi         *
;*************************************************************************
;*                                                                       *
;*  FUNCTION :  Calculates preaddition R1 block                          *
;*                                                                       *
;*************************************************************************

M1YX    macro

    move            y:(r0)+n0,y0    x:(r7)+,x0      ; y0=s3 x0=a0 r0->s5 r7->a1
    mpy     y0,x0,a     y:(r7)+,y1      x:(r0)+,x1      ; a=a0*s3 y1=a1 x1=s5 r7->a2 r0->s6
    mpy     x0,x1,b             x:(r0)+,x0      ; b=a0*s5 x0=s6 r0->s7
    mpy     y1,x0,a     a,y:(r0)+n0             ; a=a1*s6 t3=a0*s3 r0->s5
    move                    b,x:(r0)-n0     ; t5=a0*s5 r0->s7
    move            y:(r7)+,y1      x:(r0),b        ; y1=a2 b=s7 r7->a3
    add     x0,b        y:(r7)-,y0      x:(r0)-,x0      ; b=s6+s7 y0=a3 x0=s7 r0->s6 r7->a2
    move            b,x1                    ; x1=s6+s7
    mpy     y1,x1,b     (r7)-                   ; b=a2*(s6+s7) r7->a1
    sub     b,a         (r7)-                   ; a=a1*s6-a2*(s6+s7) r7->a0
    mac     y0,x0,b             a,x:(r0)+       ; b=a2*(s6+s7)+a3*s7 t6=a1*s6-a2*(s6+s7) r0->s7
    move                    b,x:(r0)-n0     ; t7=a2*(s6+s7)+a3*s7 r0->next1
    move                    (r0)-n0     ; r0->next3

    endm
```

C.1.9 M2XY.asm

```
;*************************************************************************
;*                                                                       *
;*  MODULE NAME -   M2XY                                                 *
;*                                                                       *
;*  INPUT       -   s(0)..s(7) are as follows:                           *
;*              X:s(0) s(1) s(2) s(3)                                    *
;*              Y:s(4) s(5) s(6) s(7)                                    *
;*              r0 - pointer to s(0) in X memory and s(4) in Y memory    *
;*              r1 - pointer to b1 both in X and Y memories              *
;*              r4 - pointer to s(0) in X memory and s(4) in Y memory    *
;*              r7 - pointer to a0 both in X and Y memories              *
;*              n0=-2                                                    *
;*                                                                       *
;*  OUTPUT      -   s(0)..s(7) are replaced by M2 values                 *
;*              r0 ponits to next row, s(0) (r0+4)                       *
;*              r1 reponits to b1 (r1)                                   *
;*              r4 ponits to next row, s(0) (r4+4)                       *
;*              r7 reponits to a0 (r7)                                   *
;*                                                                       *
```

```
;*  SUBROUTINES                                                 *
;*  CALLED     -  none                                          *
;*                                                              *
;*  MACROS USED -  none                                         *
;*                                                              *
;*  REGISTERS  -  x0,x1,y0,y1,a,b                               *
;*  CORRUPTED                                                   *
;*                                                              *
;****************************************************************
;*  CHANGE HISTORY                                              *
;*  dd/mm/yy  Code Ver  Description                 Author      *
;*  --------  --------  -----------                 ------      *
;*  27/08/98  1.00      Initial version             Dror Halahmi *
;****************************************************************
;*                                                              *
;*  FUNCTION :  Calculates M2 multiplication stage              *
;*                                                              *
;****************************************************************

M2XY    macro

      move              x:(r0)+,x0      y:(r7),y0       ; x0=s0 y0=a0 r0->s1
      mpy     x0,y0,a         x:(r0)+,x0      y:(r4),y1       ; a=a0*s0 x0=s1 y1=s4 r0->s2 r4->s0
      mpy     x0,y0,b         x:(r0)-,x0                      ; b=a0*s1 x0=s2 r0->s5
      mpy     x0,y0,a         a,x:(r4)+                       ; a=a0*s2 t0=a0*s0 r4->s1
      mpy     y1,y0,b         b,x:(r4)+       y:(r0)-,y0      ; b=a0*s4 t1=a0*s1 y0=s5 r4->s2 r0->s4
      tfr     y0,a            a,x:(r4)+                       ; a=s5 t2=a0*s2 r4->s3
      asr     a               x:(r4)-,b       b,y:(r0)+       ; b=s3 t4=a0*s4 r0->s5 r4->s6
      asr     a                                               ; a=.5*s5
      asr     b               x:(r1)+,x0      y:(r4)+,y0      ; x0=b1 y0=s6 r1->b2 r4->s7
      asr     b                                               ; b=.5*s3
      mpy     x0,y0,a                         a,y:(r0)-n0     ; a=b1*s6 t5=.5*s5 r0->s7
      move              b,x:(r4)+       y:(r0),b        ; t3=.5*s3 b=s7 r4->next0
      add     y0,b            x:(r1)+,x1                      ; b=s6+s7 x1=b2 r1->b3
      move              x:(r1)-,x0      b,y1            ; x0=b3 y1=s6+s7 r1->b2
      mpy     y1,x1,b                         y:(r0)-,y1      ; b=b2*(s6+s7) y1=s7 r0->s6
      sub     b,a             (r1)-                           ; a=b1*s6-b2*(s6+s7) r1->b1
      mac     x0,y1,b                                         ; b=b3*s7+b2*(s6+s7)
      move                                    a,y:(r0)+       ; t6=b1*s6-b2*(s6+s7) r0->s7
      move                                    b,y:(r0)+       ; t7=b3*s7+b2*(s6+s7) r0->next0

      endm
```

C.1.10 M2YX.asm

```
;****************************************************************
;*                                                              *
;*  MODULE NAME -  M2YX                                         *
;*                                                              *
;*  INPUT      -  s(0)..s(7) are as follows:                    *
;*              Y:s(0) s(1) s(2) s(3)                           *
;*              y:s(4) s(5) s(6) s(7)                           *
;*              r0 - pointer to s(0) in Y memory and s(4) in X memory   *
;*              r1 - pointer to b1 both in X and Y memories     *
;*              r4 - pointer to s(0) in Y memory and s(4) in X memory   *
;*              r7 - pointer to a0 both in X and Y memories     *
;*              n0=-2                                           *
;*                                                              *
;*  OUTPUT     -  s(0)..s(7) are replaced by M2 values          *
;*              r0 ponits to next row, s(0) (r0+4)              *
;*              r1 reponits to b1 (r1)                          *
;*              r4 ponits to next row, s(0) (r4+4)              *
;*              r7 reponits to a0 (r7)                          *
;*                                                              *
;*  SUBROUTINES                                                 *
;*  CALLED     -  none                                          *
;*                                                              *
;*  MACROS USED -  none                                         *
;*                                                              *
;*  REGISTERS  -  y0,y1,x0,x1,a,b                               *
;*  CORRUPTED                                                   *
;*                                                              *
;****************************************************************
;*  CHANGE HISTORY                                              *
;*  dd/mm/yy  Code Ver  Description                 Author      *
;*  --------  --------  -----------                 ------      *
;*  27/08/98  1.00      Initial version             Dror Halahmi *
;****************************************************************
;*                                                              *
;*  FUNCTION :  Calculates M2 multiplication stage              *
```

```
;*                                                                              *
;********************************************************************************

M2YX    macro

      move              y:(r0)+,y0      x:(r7),x0       ; y0=s0 x0=a0 r0->s1
      mpy     y0,x0,a       y:(r0)+,y0      x:(r4),x1       ; a=a0*s0 y0=s1 x1=s4 r0->s2 r4->s0
      mpy     y0,x0,b       y:(r0)-,y0                      ; b=a0*s1 y0=s2 r0->s5
      mpy     y0,x0,a       a,y:(r4)+                       ; a=a0*s2 t0=a0*s0 r4->s1
      mpy     x1,x0,b       b,y:(r4)+       x:(r0)-,x0      ; b=a0*s4 t1=a0*s1 x0=s5 r4->s2 r0->s4
      tfr     x0,a          a,y:(r4)+                       ; a=s5 t2=a0*s2 r4->s3
      asr     a             y:(r4)-,b       b,x:(r0)+       ; b=s3 t4=a0*s4 r0->s5 r4->s6
      asr     a                                             ; a=.5*s5
      asr     b             y:(r1)+,y0      x:(r4)+,x0      ; y0=b1 x0=s6 r1->b2 r4->s7
      asr     b                                             ; b=.5*s3
      mpy     y0,x0,a                       a,x:(r0)-n0     ; a=b1*s6 t5=.5*s5 r0->s7
      move                  b,y:(r4)+       x:(r0),b        ; t3=.5*s3 b=s7 r4->next0
      add     x0,b          y:(r1)+,y1                      ; b=s6+s7 y1=b2 r1->b3
      move                  y:(r1)-,y0      b,x1            ; y0=b3 x1=s6+s7 r1->b2
      mpy     x1,y1,b                       x:(r0)-,x1      ; b=b2*(s6+s7) x1=s7 r0->s6
      sub     b,a           (r1)-                           ; a=b1*s6-b2*(s6+s7) r1->b1
      mac     y0,x1,b                                       ; b=b3*s7+b2*(s6+s7)
      move                                  a,x:(r0)+       ; t6=b1*s6-b2*(s6+s7) r0->s7
      move                                  b,x:(r0)+       ; t7=b3*s7+b2*(s6+s7) r0->next0

      endm
```

C.1.11 M3YX.asm

```
;********************************************************************************
;*                                                                              *
;*  MODULE NAME -   M3YX                                                        *
;*                                                                              *
;*  INPUT       -   s(0)..s(7) S(0)..S(7) are as follows:                       *
;*                  Y:s(0) s(1) s(2) s(3) S(0) S(1) S(2) S(3)                   *
;*                  X:s(4) s(5) s(6) s(7) S(4) S(5) S(6) S(7)                   *
;*                  a1, a2, a3, b1, b2, b3, c0 coefficients are both in X, Y:   *
;*                  X:Y:a1 a2 a3 b1 b2 b3 a1 a2 a3 b1 b2 b3 c0                  *
;*                  r0 - pointer to s(0) in Y memory and s(4) in X memory       *
;*                  r1 - pointer to S(0) in Y memory and S(4) in X memory       *
;*                  r7 - pointer to a1 both in X and Y memories                 *
;*                  n0=n1=n7=2                                                  *
;*                                                                              *
;*  OUTPUT      -   s(0)..s(7) are replaced by M3 values                        *
;*                                                                              *
;*  SUBROUTINES                                                                 *
;*  CALLED      -   none                                                        *
;*                                                                              *
;*  MACROS USED -   none                                                        *
;*                                                                              *
;*  REGISTERS   -   x0,x1,y0,y1,a,b,r0,r1,r7                                    *
;*  CORRUPTED                                                                   *
;*                                                                              *
;********************************************************************************
;*  CHANGE HISTORY                                                              *
;*  dd/mm/yy  Code Ver  Description                         Author              *
;*  --------  --------  -----------                         ------              *
;*  27/08/98   1.00     Initial version                     Dror Halahmi        *
;********************************************************************************
;*                                                                              *
;*  FUNCTION :  Calculates M3 multiplication stage                              *
;*                                                                              *
;********************************************************************************

M3YX    macro

      move              x:(r7)+,x0      y:(r0)+,y0      ; x0=a1 y0=s0 r7->a2 r0->s1
      mpy     x0,y0,a       x:(r7)+,x0      y:(r1),b        ; a=a1*s0 x0=a2 b=S0 r7->a3 r1->S0
      add     y0,b          x:(r7)-,x1      y:(r1)+,y0      ; b=s0+S0 x1=a3 y0=S0 r7->a2 r1->S1
      move                                  b,y1            ; y1=s0+S0
      mpy     y1,x0,b       (r7)-                           ; b=a2*(s0+S0) r7->a1
      sub     b,a           x:(r7)+,x0                      ; a=a1*s0-a2*(s0+S0) x0=a1 r7->a2
      mac     x1,y0,b       x:(r7)+,x1      y:(r0)-,y0      ; b=a3*S0+a2*(s0+S0) x1=a2 y0=s1 r7->a3 r0->s0
      mpy     x0,y0,a                       a,y:(r0)+n0     ; a=a1*s1 t0=a1*s0-a2*(s0+S0) r0->s2
      move              b,x0            y:(r1),b        ; x0=a3*S0+a2*(s0+S0) b=S1
      add     y0,b                      y:(r1)-,y0      ; b=s1+S1 y0=S1 r1->S0
      move                                  x0,y:(r1)+n1    ; T0=a3*S0+a2*(s0+S0) r1->S2
      move                                  b,y1            ; y1=s1+S1
      mpy     y1,x1,b       x:(r7)-n7,x1                    ; b=a2*(s1+S1) x1=a3 r7->a1
      sub     b,a           x:(r7)+,x0                      ; a=a1*s1-a2*(s1+S1) x0=a1 r7->a2
```

```
    mac     x1,y0,b       x:(r7)+,x1      y:(r0)-,y0      ; b=a3*S1+a2*(s1+S1) x1=a2 y0=s2 r7->a3 r0->s1
    mpy     x0,y0,a                       a,y:(r0)+n0     ; a=a1*s2 t1=a1*s1-a2*(s1+S1) r0->s3
    move                  b,x0            y:(r1),b        ; x0=a3*S1+a2*(s1+S1) b=S2 r1->S2
    add     y0,b                          y:(r1)-,y0      ; b=s2+S2 y0=S2 r1->S1
    move                                  x0,y:(r1)+n1    ; T1=a3*S1+a2*(s1+S1) r1->S3
    move                                  b,y1            ; y1=s2+S2
    mpy     y1,x1,b       x:(r7)+,x1                      ; b=a2*(s2+S2) x1=a3 r7->b1
    sub     b,a           x:(r7)+,x0                      ; a=a1*s2-a2*(s2+S2) x0=b1 r7->b2
    mac     x1,y0,b       x:(r7)+,x1      y:(r0)-,y0      ; b=a3*S2+a2*(s2+S2) x1=b2 y0=s3 r7->b3 r0->s2
    mpy     x0,y0,a                       a,y:(r0)-n0     ; a=b1*s3 t2=a1*s2-a2*(s2+S2) r0->s4
    move                  b,x0            y:(r1),b        ; x0=a3*S2+a2*(s2+S2) b=S3 r1->S3
    add     y0,b                          y:(r1)-,y0      ; b=s3+S3 y0=S3 r1->S2
    move                                  x0,y:(r1)-n1    ; T2=a3*S2+a2*(s2+S2) r1->S4
    move                                  b,y1            ; y1=s3+S3
    mpy     y1,x1,b       x:(r7)+,x1                      ; b=b2*(s3+S3) x1=b3 r7->a1
    sub     b,a           x:(r7)+,x0                      ; a=b1*s3-b2*(s3+S3) x0=a1 r7->a2
    mac     x1,y0,b       x:(r0)+n0,x1    y:(r7)+,y1      ; b=b3*S3+b2*(s3+S3) y1=a2 x1=s4 r7->a3 r0->s2
    move                  (r0)+                           ; r0->s3
    mpy     x0,x1,a                       a,y:(r0)-n0     ; a=a1*s4 t3=b1*s3-b2*(s3+S3) r0->s5
    move                  x:(r1),b        b,y0            ; y0=b3*S3+b2*(s3+S3) b=S4 r1->S4
    add     x1,b          x:(r1)+n1,x0                    ; b=s4+S4 x0=S4 r1->S2
    move                  (r1)+                           ; r1->S3
    move                  b,x1            y0,y:(r1)-n1    ; x1=s4+S4 T3=b3*S3+b2*(s3+S3) r1->S5
    mpy     y1,x1,b       x:(r1),x1       y:(r7)+,y1      ; b=a2*(s4+S4) x1=dummy y1=a3 r1->S5 r7->b1
    sub     b,a                           y:(r7)+,y0      ; a=a1*s4-a2*(s4+S4) y0=b1 r7->b2
    mac     y1,x0,b       x:(r0)-,x1      y:(r7)+,y1      ; b=a3*S4+a2*(s4+S4) y1=b2 x1=s5 r7->b3 r0->s4
    mpy     y0,x1,a       a,x:(r0)+n0                     ; a=b1*s5 t4=a1*s4-a2*(s4+S4) r0->s6
    move                  x:(r1),b        b,y0            ; y0=a3*S4+a2*(s4+S4) b=S5 r1->S5
    add     x1,b          x:(r1)-,x0                      ; b=s5+S5 x0=S5 r1->S4
    move                  y0,x:(r1)+n1                    ; T4=a3*S4+a2*(s4+S4) r1->S6
    move                                  b,y0            ; y0=s5+S5
    mpy     y1,y0,b                       y:(r7)+,y1      ; b=b2*(s5+S5) y1=b3 r7->c0
    sub     b,a           x:(r1)-,y0                      ; a=b1*s5-b2*(s5+S5) y0=S6 r1->S5
    mac     y1,x0,b       x:(r0)-,x0      y:(r7),y1       ; b=b3*S5+b2*(s5+S5) x0=s6 y1=c0 r0->s5
    move                  a,x:(r0)+n0     x0,a            ; t5=b1*s5-b2*(s5+S5) a=s6 r0->s7
    move                  b,x:(r1)+n1                     ; T5=b3*S5+b2*(s5+S5) r1->S7
    move                  x:(r1),b                        ; b=S7
    andi    #7,mr                                         ; cancel scaling up mode before Q3
    sub     a,b                                           ; b=S7-s6
    addl    b,a                                           ; a=S7+s6
    move                  b,x0                            ; x0=S7-s6
    mpy     x0,y1,b       x:(r0)-,x0                      ; b=c0*(S7-s6) x0=s7 r0->s6
    asr     a                                             ; a=.5*(S7+s6)
    move                  b,x:(r0)        x0,b            ; t6=c0*(S7-s6) b=s7
    tfr     y0,a          a,x:(r1)-                       ; T7=.5*(S7+s6) a=S6 r1->S6
    sub     a,b                                           ; b=s7-S6
    addl    b,a           x:(r0),x0                       ; a=s7+S6 x0=c0*(S7-s6)
    asr     b                                             ; b=.5*(s7-S6)
    move                  a,y0                            ; y0=s7+S6
    mpy     y0,y1,a       b,x:(r1)+       x0,b            ; a=c0*(s7+S6) T6=.5*(s7-S6) b=c0*(S7-s6) r1->T7
    sub     a,b                                           ; b=c0*(S7-s6)-c0*(s7+S6)
    addl    b,a                                           ; a=c0*(S7-s6)+c0*(s7+S6)
    move                  x:(r1)-,b       b,y0            ; b=U7 y0=u7 r1->U6
    sub     a,b                                           ; b=U7-u6
    addl    b,a           x:(r1)+,y1                      ; a=U7+u6 y1=U6
    tfr     y0,b          b,x:(r1)-                       ; V7=U7-u6 b=u7 r1->U6
    tfr     y1,a          a,x:(r0)+                       ; v6=U7+u6 a=U6 r0->u7
    sub     a,b                                           ; b=u7-U6
    addl    b,a                                           ; a=u7+U6
    move                  b,x:(r1)                        ; V6=u7-U6
    move                  a,x:(r0)                        ; v7=u7+U6

    endm
```

C.1.12 R2rowsXY.asm

```
;*************************************************************************
;*                                                                       *
;*  MODULE NAME -   R2rowsXY                                             *
;*                                                                       *
;*  INPUT       -   s(0)..s(7) are as follows:                           *
;*                  X:s(0) s(1) s(2) s(3)                                *
;*                  Y:s(4) s(5) s(6) s(7)                                *
;*                  r0 - pointer to s(2) in X memory                     *
;*                  r7 - pointer to s(4) in Y memory                     *
;*                  n7=2                                                 *
;*                                                                       *
;*  OUTPUT      -   s(0)..s(7) are replaced by R2 values                 *
;*                  r0 ponits to next row (r0+4)                         *
;*                  r7 ponits to next row (r7+4)                         *
```

```
;*                                                                      *
;* SUBROUTINES                                                          *
;* CALLED     -  none                                                   *
;*                                                                      *
;* MACROS USED -  none                                                  *
;*                                                                      *
;* REGISTERS  -  r6,x0,y0,a,b                                           *
;* CORRUPTED                                                            *
;*                                                                      *
;************************************************************************
;* CHANGE HISTORY                                                       *
;* dd/mm/yy  Code Ver  Description                    Author            *
;* --------  --------  -----------                    ------            *
;* 27/08/98  1.00      Initial version                Dror Halahmi      *
;************************************************************************
;*                                                                      *
;* FUNCTION :  Calculates R2 for postadditions stage                    *
;*                                                                      *
;************************************************************************

R2rowsXY       macro

      move              x:(r0)+,b                ; b=s2 r0->s3
      move              x:(r0),a                 ; a=s3
      sub     b,a                                ; a=s3-s2
      addl    a,b                                ; b=s3+s2
      move              a,x:(r0)-      y:(r7)+,a      ; t3=s3-s2 a=s4 r0->s2 r7->s5
      move              b,x:(r0)+      y:(r7)+,b      ; t2=s3+s2 b=s5 r0->s3 r7->s6
      sub     a,b       r7,r6                    ; b=s5-s4 r6->s6
      addl    b,a                                ; a=s5+s4
      move              b,x0           y:(r7)+,b      ; x0=s5-s4 b=s6 r7->s7
      neg     b         x:(r0)+,x1     y:(r7)-,y0     ; b=-s6 x1=dummy y0=s7 r0->next0
      sub     a,b       (r0)+                    ; b=-s6-s5-s4 r0->next1
      addl    b,a       (r0)+                    ; a=s5+s4-s6 r0->next2
      move              y0,b           b,y:(r7)-n7    ; t6=-s6-s5-s4 b=s7 r7->s4
      tfr     x0,a                     a,y:(r7)+n7    ; a=s5-s4 t4=s5+s4-s6 r7->s6
      sub     a,b                      (r7)+          ; b=s7-s5+s4 r7->s7
      addl    b,a       (r6)-                    ; a=s7+s5-s4 r6->s5
      move                             b,y:(r7)+      ; t7=s7-s5+s4 r7->next4
      move                             a,y:(r6)       ; t5=s7+s5-s4

      endm
```

C.1.13 R2rowsYX.asm

```
;************************************************************************
;*                                                                      *
;* MODULE NAME -  R2rowsYX                                              *
;*                                                                      *
;* INPUT      -  s(0)..s(7) are as follows:                             *
;*               Y:s(0) s(1) s(2) s(3)                                  *
;*               X:s(4) s(5) s(6) s(7)                                  *
;*               r0 - pointer to s(2) in Y memory                       *
;*               r7 - pointer to s(4) in X memory                       *
;*               n7=2                                                   *
;*                                                                      *
;* OUTPUT     -  s(0)..s(7) are replaced by R2 values                   *
;*               r0 ponits to next row (r0+4)                           *
;*               r7 ponits to next row (r7+4)                           *
;*                                                                      *
;* SUBROUTINES                                                          *
;* CALLED     -  none                                                   *
;*                                                                      *
;* MACROS USED -  none                                                  *
;*                                                                      *
;* REGISTERS  -  r6,y0,x0,a,b                                           *
;* CORRUPTED                                                            *
;*                                                                      *
;************************************************************************
;* CHANGE HISTORY                                                       *
;* dd/mm/yy  Code Ver  Description                    Author            *
;* --------  --------  -----------                    ------            *
;* 27/08/98  1.00      Initial version                Dror Halahmi      *
;************************************************************************
;*                                                                      *
;* FUNCTION :  Calculates R2 for postadditions stage                    *
;*                                                                      *
;************************************************************************

R2rowsYX       macro
```

```
    move            y:(r0)+,b               ; b=s2 r0->s3
    move            y:(r0),a                ; a=s3
    sub   b,a                               ; a=s3-s2
    addl  a,b                               ; b=s3+s2
    move            a,y:(r0)-   x:(r7)+,a   ; t3=s3-s2 a=s4 r0->s2 r7->s5
    move            b,y:(r0)+   x:(r7)+,b   ; t2=s3+s2 b=s5 r0->s3 r7->s6
    sub   a,b       r7,r6                   ; b=s5-s4 r6->s6
    addl  b,a                               ; a=s5+s4
    move            b,y0        x:(r7)+,b   ; y0=s5-s4 b=s6 r7->s7
    neg   b         y:(r0)+,y1  x:(r7)-,x0  ; b=-s6 y1=dummy x0=s7 r0->next0
    sub   a,b       (r0)+                   ; b=-s6-s5-s4 r0->next1
    addl  b,a       (r0)+                   ; a=s5+s4-s6 r0->next2
    move            x0,b        b,x:(r7)-n7 ; t6=-s6-s5-s4 b=s7 r7->s4
    tfr   y0,a                  a,x:(r7)+n7 ; a=s5-s4 t4=s5+s4-s6 r7->s6
    sub   a,b                   (r7)+       ; b=s7-s5+s4 r7->s7
    addl  b,a       (r6)-                   ; a=s7+s5-s4 r6->s5
    move                        b,x:(r7)+   ; t7=s7-s5+s4 r7->next4
    move                        a,x:(r6)    ; t5=s7+s5-s4

    endm
```

C.1.14 R2columnsXY.asm

```
;*************************************************************************
;*                                                                       *
;*  MODULE NAME -   R2columnsXY                                          *
;*                                                                       *
;*  INPUT       -   s(0)..s(7) are as follows:                           *
;*                  X:s(0)...s(1)...s(2)...s(3)                          *
;*                  Y:s(4)...s(5)...s(6)...s(7)                          *
;*                  r0 - pointer to s(2) in X memory                     *
;*                  r7 - pointer to s(4) in Y memory                     *
;*                  n0=n6=n7=4                                           *
;*                                                                       *
;*  OUTPUT      -   s(0)..s(7) are replaced by R2 values                 *
;*                  r0 ponits to next row (r0+1)                         *
;*                  r7 ponits to next row (r7+1)                         *
;*                                                                       *
;*  SUBROUTINES                                                          *
;*  CALLED      -   none                                                 *
;*                                                                       *
;*  MACROS USED -   none                                                 *
;*                                                                       *
;*  REGISTERS   -   r6,x0,y0,a,b                                         *
;*  CORRUPTED                                                            *
;*                                                                       *
;*************************************************************************
;*  CHANGE HISTORY                                                       *
;*  dd/mm/yy  Code Ver  Description                         Author       *
;*  --------  --------  -----------                         ------       *
;*  27/08/98   1.00     Initial version                     Dror Halahmi *
;*************************************************************************
;*                                                                       *
;*  FUNCTION :  Calculates R2 for postadditions stage                    *
;*                                                                       *
;*************************************************************************

R2columnsXY     macro

    move            x:(r0)+n0,b             ; b=s2 r0->s3
    move            x:(r0),a                ; a=s3
    sub   b,a       r7,r6                   ; a=s3-s2 r6->s4
    addl  a,b                               ; b=s3+s2 r0->s3
    move            a,x:(r0)-n0             ; t3=s3-s2 r2->s2
    move                        y:(r6)+n6,a ; a=s4 r6->s5
    move            b,x:(r0)+   y:(r6)+n6,b ; t2=s3+s2 b=s5 r0->next2 r6->s6
    sub   a,b       r6,r7                   ; b=s5-s4 r7->s6
    addl  b,a                               ; a=s5+s4
    move            b,x0        y:(r6)+n6,b ; x0=s5-s4 b=s6 r6->s7
    neg   b                     y:(r6)-n6,y0 ; b=-s6 y0=s7 r6->s6
    sub   a,b                   (r7)-n7     ; b=-s6-s5-s4 r7->s5
    addl  b,a                   (r7)-n7     ; a=s5+s4-s6 r7->s4
    move            y0,b        b,y:(r6)+n6 ; t6=-s6-s5-s4 b=s7 r6->s7
    tfr   x0,a                  a,y:(r7)+n7 ; a=s5-s4 t4=s5+s4-s6 r7->s5
    sub   a,b                               ; b=s7-s5+s4
    addl  b,a                               ; a=s7+s5-s4
    move                        b,y:(r6)    ; t7=s7-s5+s4
    move                        a,y:(r7)-n7 ; t5=s7+s5-s4 r7->s4
    move                        (r7)+       ; r7->next4
```

```
    endm
```

C.1.15 R2columnsYX.asm

```
;*******************************************************************************
;*                                                                             *
;*  MODULE NAME -   R2columnsYX                                                *
;*                                                                             *
;*  INPUT       -   s(0)..s(7) are as follows:                                 *
;*                  Y:s(0)...s(1)...s(2)...s(3)                                *
;*                  X:s(4)...s(5)...s(6)...s(7)                                *
;*                  r0 - pointer to s(2) in Y memory                           *
;*                  r7 - pointer to s(4) in X memory                           *
;*                  n0=n6=n7=4                                                 *
;*                                                                             *
;*  OUTPUT      -   s(0)..s(7) are replaced by R2 values                       *
;*                  r0 ponits to next row (r0+1)                               *
;*                  r7 ponits to next row (r7+1)                               *
;*                                                                             *
;*  SUBROUTINES                                                                *
;*  CALLED      -   none                                                       *
;*                                                                             *
;*  MACROS USED -   none                                                       *
;*                                                                             *
;*  REGISTERS   -   r6,y0,x0,a,b                                               *
;*  CORRUPTED                                                                  *
;*                                                                             *
;*******************************************************************************
;*  CHANGE HISTORY                                                             *
;*  dd/mm/yy  Code Ver  Description                          Author            *
;*  --------  --------  -----------                          ------            *
;*  27/08/98   1.00     Initial version                      Dror Halahmi      *
;*******************************************************************************
;*                                                                             *
;*  FUNCTION :  Calculates R2 for postadditions stage                          *
;*                                                                             *
;*******************************************************************************

R2columnsYX     macro

        move            y:(r0)+n0,b             ; b=s2 r0->s3
        move            y:(r0),a                ; a=s3
        sub     b,a     r7,r6                   ; a=s3-s2 r6->s4
        addl    a,b                             ; b=s3+s2 r0->s3
        move            a,y:(r0)-n0             ; t3=s3-s2 r2->s2
        move                    x:(r6)+n6,a     ; a=s4 r6->s5
        move            b,y:(r0)+       x:(r6)+n6,b     ; t2=s3+s2 b=s5 r0->next2 r6->s6
        sub     a,b     r6,r7                   ; b=s5-s4 r7->s6
        addl    b,a                             ; a=s5+s4
        move            b,y0            x:(r6)+n6,b     ; y0=s5-s4 b=s6 r6->s7
        neg     b                       x:(r6)-n6,x0    ; b=-s6 x0=s7 r6->s6
        sub     a,b                     (r7)-n7         ; b=-s6-s5-s4 r7->s5
        addl    b,a                     (r7)-n7         ; a=s5+s4-s6 r7->s4
        move            x0,b            b,x:(r6)+n6     ; t6=-s6-s5-s4 b=s7 r6->s7
        tfr     y0,a                    a,x:(r7)+n7     ; a=s5-s4 t4=s5+s4-s6 r7->s5
        sub     a,b                             ; b=s7-s5+s4
        addl    b,a                             ; a=s7+s5-s4
        move                            b,x:(r6)        ; t7=s7-s5+s4
        move                            a,x:(r7)-n7     ; t5=s7+s5-s4 r7->s4
        move                            (r7)+           ; r7->next4

        endm
```

C.2. Results

C.2.1 dec_output.fi

```
482.0000768
54.0000256
-432.8390656
823.1612416
-392.8670208
171.6056064
-295.209779
85.2627456
-834.0000768
```

-54.0000256
325.2342784
557.234176
-19.0050304
-587.552768
2.0635648
81.5157248
-1150.4164864
500.9117184
282.51136
-419.0416896
1642.973184
-702.34112
213.3229568
24.7230464
669.58336
-399.0884352
-370.958336
-376.5112832
-1066.1781504
65.8481152
-73.242624
-0.5013504
-192.7499776
734.5033216
60.7305728
152.2159616
-3130.3979008
-1561.9555328
171.6215808
86.7807232
-362.6766336
-205.1895296
-1043.5731456
-714.3555072
-657.3375488
604.1423872
-180.0941568
-206.5711104
606.8527104
45.7445376
-76.7811584
66.6062848
-1139.3765376
-255.2455168
-373.0423808
18.7662336
-127.074304
14.0517376
186.036224
-192.843776
212.9047552
-93.6747008
16.8726528
-0.7032832