

An 8X8 Discrete Cosine Transform Implementation on the TMS320C25 or the TMS320C30

APPLICATION REPORT: SPRA115

*William Hohl
Digital Signal Processor Products
Semiconductor Group
Texas Instruments*

Digital Signal Processing Solutions



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

An 8X8 Discrete Cosine Transform Implementation on the TMS320C25 or the TMS320C30

Abstract

The Discrete Cosine Transform (DCT) stands apart from other orthogonal transforms because of its favorable comparison to the Karhunen-Loeve Transform (KLT). However, there is no fast algorithm to compute the KLT, which makes the DCT an attractive alternative. This book presents two 8X8 DCT routines and is divided into the following pieces:

- ❑ The DCT algorithm
- ❑ Implementation in the TMS320C25 and TMS320C30 processors
- ❑ TMS320C25 code for a roundoff routine
- ❑ Signal flow graphs for 2-2-point, 4-point, and 8-point DCTs
- ❑ TMS320C30 code for bit reversal
- ❑ Execution times and memory requirements

The appendices at the end of the book contain code for the DCT algorithms for both the TMS320C25 and TMS320C30 processors.



Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to dsph@ti.com. Questions receive prompt attention and are usually answered within one business day.

Introduction

In the general class of orthogonal transforms, there exists one in particular, the discrete cosine transform (DCT), that has recently gained wide popularity in signal processing. The DCT has found applications in such areas as data compression, pattern recognition, and Weiner filtering, primarily because of its close comparison to the Karhunen-Loeve Transform (KLT) with respect to rate distortion criteria [1]. Although the KLT is considered to be optimal, there is no fast algorithm to compute it. Since there is no fast KLT algorithm, the DCT is an attractive alternative.

For image coding, the DCT works well because of the high correlation among adjacent data samples (pixel values). Because of this correlation, the DCT provides near optimal reduction while retaining high image quality. In a comparative study [2], the DCT was shown to outperform the Fourier, Hartley, and cas-cas transforms for image compression, providing even more motivation for finding fast implementations.

A number of algorithms have been developed, most notably those of Hou [3] and Lee [4], which generate higher-order DCTs from lower-order ones. This paper presents two 8×8 DCT routines, one for the TMS320C25 and another for the TMS320C30, based upon the routine in [3].

The DCT Algorithm

For a given real data sequence x_0, x_1, \dots, x_{N-1} , the discrete cosine transform is given in [1] as

$$z_k = \sqrt{\frac{2}{N}} \alpha(k) \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad k = 0, 1, \dots, N-1 \quad (1a)$$

and its inverse is

$$x_n = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} \alpha(k) z_k \cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad k = 0, 1, \dots, N-1 \quad (1b)$$

where $\alpha(k) = \frac{1}{\sqrt{2}}$ for $k = 0$; otherwise, the transform is unitary. If z_0 is scaled up by 2, the DCT can also be written in matrix form as

$$\mathbf{z} = \sqrt{\frac{2}{N}} T(N) \mathbf{x}, \quad (2)$$

where \mathbf{x} and \mathbf{z} are column vectors denoting the input and output data sequences, and $T(N)$ is the DCT matrix of order N . Actually, expanding the matrix (neglecting the factor of $\sqrt{\frac{2}{N}}$ for the moment), a 4-point DCT appears as

$$\begin{bmatrix} z_0 \\ z_2 \\ z_1 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \alpha & -\alpha & \alpha & -\alpha \\ \beta & -\delta & -\beta & \delta \\ \delta & \beta & -\delta & -\beta \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_3 \\ x_1 \end{bmatrix}, \quad (3)$$

where $\alpha = \frac{1}{\sqrt{2}}$, $\beta = \cos\left(\frac{\pi}{8}\right)$, and $\delta = \sin\left(\frac{\pi}{8}\right)$. Similarly, the 8-pt DCT can be expressed as

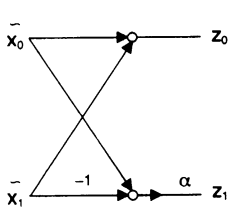
$$\begin{bmatrix} z_0 \\ z_4 \\ z_2 \\ z_6 \\ z_1 \\ z_5 \\ z_3 \\ z_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha & -\alpha & \alpha & -\alpha & \alpha & -\alpha & \alpha & -\alpha \\ \beta & -\delta & -\beta & \delta & \beta & -\delta & -\beta & \delta \\ \delta & \beta & -\delta & -\beta & \delta & \beta & -\delta & -\beta \\ \lambda & \mu & -\nu & -\gamma & -\lambda & -\mu & \nu & \gamma \\ \mu & \nu & -\gamma & \lambda & -\mu & -\nu & \gamma & -\lambda \\ \gamma & -\lambda & \mu & \nu & -\gamma & \lambda & -\mu & -\nu \\ \nu & \gamma & \lambda & \mu & -\nu & -\gamma & -\lambda & -\mu \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \\ x_7 \\ x_5 \\ x_3 \\ x_1 \end{bmatrix}, \quad (4)$$

where $\lambda = \cos\left(\frac{\pi}{16}\right)$, $\gamma = \cos\left(\frac{3\pi}{16}\right)$, $\mu = \sin\left(\frac{3\pi}{16}\right)$, and $\nu = \sin\left(\frac{\pi}{16}\right)$. Note that the input is no longer in natural order but has been rearranged according to the permutation matrix P and the relation

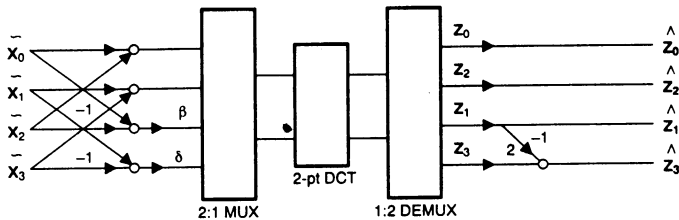
$$\bar{x} = Px, \quad (5)$$

where

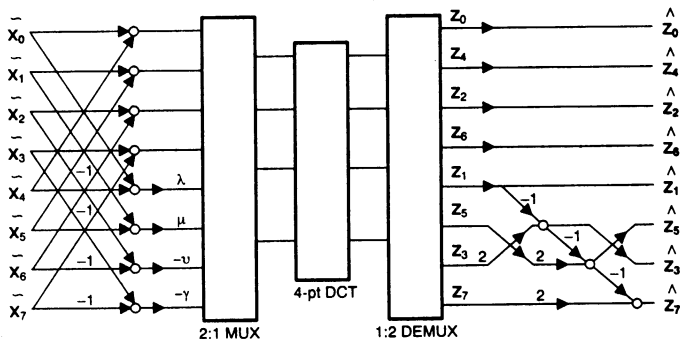
$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



(a) 2-Point



(b) 4-Point



(c) 8-Point

Figure 1. Signal Flow Graphs for 2-Point, 4-Point, and 8-Point DCTs

The structure of the algorithm looks very much like that of a Fast Fourier Transform (FFT), since the most fundamental computation is a 2-point butterfly. This routine is actually a generalized case of the Cooley-Tukey FFT algorithm with the addition of the recursion at the end. If the equations for the signal flow graph are written explicitly, the recursive nature of the DCT becomes clear; for a 4-point DCT, we have

$$\begin{aligned} \hat{z}_0 &= z_0, \\ \hat{z}_2 &= z_2, \\ \hat{z}_1 &= z_1, \\ \hat{z}_3 &= 2z_3 - \hat{z}_1, \end{aligned}$$

and for the 8-point DCT,

$$\begin{aligned}
 \hat{z}_0 &= z_0, \\
 \hat{z}_4 &= z_4, \\
 \hat{z}_2 &= z_2, \\
 \hat{z}_6 &= z_6, \\
 \hat{z}_1 &= z_1, \\
 \hat{z}_3 &= 2z_3 - \hat{z}_1, \\
 \hat{z}_5 &= 2z_5 - \hat{z}_3, \\
 \hat{z}_7 &= 2z_7 - \hat{z}_5.
 \end{aligned}$$

To create a unitary transform, each element in the vector should be multiplied by the scaling factor $\sqrt{\frac{2}{N}}$ for both the forward and inverse transforms. The inverse transform is obtained by completely reversing the direction of the signal flow graph; i.e., performing the bit-reversal first, then the recursions and the butterflies, and finally, the data permutation.

For the two-dimensional case of interest, the DCT can be described in the form

$$z(k,l) = \frac{2}{N} \alpha(k) \alpha(l) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m,n) \cos\left(\frac{\pi(2m+1)k}{2N}\right) \cos\left(\frac{\pi(2n+1)l}{2N}\right) \quad (8a)$$

$$x(m,n) = \frac{2}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \alpha(k) \alpha(l) z(k,l) \cos\left(\frac{\pi(2m+1)k}{2N}\right) \cos\left(\frac{\pi(2n+1)l}{2N}\right) \quad (8b)$$

where $\alpha(k) = \frac{1}{\sqrt{2}}$ for $k = 0$, unity otherwise. Like the FFT, the DCT kernel is separable, allowing the transform to be performed in two steps, first along the rows and then the columns.

Implementation on the TMS320C25

The DCT algorithm may be carried out in one of two ways, either using

1. A matrix formulation, where the DCT coefficients are simply multiplied by the data, or
2. The signal flow graph.

This routine uses a matrix formulation, which requires the sixty-four cosine coefficients to be stored in an array in memory. The matrix formulation is based on the following equation:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \lambda & \gamma & \mu & \nu & -\nu & -\mu & -\gamma & -\lambda \\ \beta & \delta & -\delta & -\beta & -\beta & -\delta & \delta & \beta \\ \gamma & -\nu & -\lambda & -\mu & \mu & \lambda & \nu & -\gamma \\ \alpha & -\alpha & -\alpha & \alpha & \alpha & -\alpha & -\alpha & \alpha \\ \mu & -\lambda & \nu & \gamma & -\gamma & -\nu & \lambda & -\mu \\ \delta & -\beta & \beta & -\delta & -\delta & \beta & -\beta & \delta \\ \nu & -\mu & \gamma & -\lambda & \lambda & -\gamma & \mu & -\nu \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \quad (7)$$

where $\lambda = \cos\left(\frac{\pi}{16}\right)$, $\gamma = \cos\left(\frac{3\pi}{16}\right)$, $\mu = \sin\left(\frac{3\pi}{16}\right)$, and $\nu = \sin\left(\frac{\pi}{16}\right)$.

The algorithm described above has been shown to be numerically stable for fixed-point processors; however, to prevent serious data errors, truncation and roundoff must be accounted for. A roundoff technique similar to the one in [6], is used to prescale the matrix coefficients by $(2^{15} - 1)$. This product is then loaded into the accumulator with a one-bit left shift, effectively dividing it by 2^{15} . After a multiplication is performed, the 32-bit value in the accumulator must be rounded to sixteen bits, where bits 13,14, and 15 are used to determine the value of the sixteenth bit. The TMS320C25 performs this operation in a single instruction by adding 3000h to the accumulator product with a one-bit left shift, as outlined in the code shown in Figure 2.

```

*
*   INITIALIZE MATRIX COEFFICIENTS AND ROUNDOFF VALUES INTO
*   INTERNAL BLOCK 0
*
DCTINI   LDPK      RNDOFF
         RSXM      ; SIGN-EXTENSION MODE
         SPM       1 ; LEFT SHIFT 1 BIT
         LRLK     AR1,COEFF ; COEFFICIENTS
         RPTK     EDATA-IDATA
         BLKP     IDATA,*+
         LRLK     AR1,RNDOFF ; VARIABLES
         RPTK     10
         BLKP     EDATA,*+
         .
         .
         .
*
*   SECOND SET OF COEFFICIENTS
*
         LAR      AR1,DST ; AR1 IS NOW DESTINATION
                           POINTER
         MAR      *+,AR2 ; WORK ON SECOND COLUMN
         LAR      AR2,SRC
         LARK     AR3,7
         LT       *+,AR2
         MPY     C10
T2       ZAC
         RPTK     6
         MAC     C11,*+
*
         LTA     *+,AR1
         MPY     C10
         ADD     RNDOFF
         SACH    *0+,AR3
         BANZ    t2,*-,AR2

```

Figure 2. TMS320C25 Code for Roundoff Routine

After the multiplications are computed, the results are stored in another array area in transposed order; thus, a separate routine for transposing the matrix is not needed. Once the rows are transformed, the pointers for the input and output matrices are exchanged. When the procedure is repeated, the output is stored as rows, completing the transform. Appendix A contains a complete program listing for the forward transform on the TMS320C25. To perform an inverse DCT, the table of cosine coefficients should be replaced with those used for an inverse transform.

Implementation on the TMS320C30

The TMS320C30's increased speed and flexible addressing modes can reduce execution time substantially. In using the FFT-like structure, extraneous multiplications are removed, and because of the TMS320C30's ability to perform parallel multiplication/additions, two butterflies can be computed at once. After an initial subtraction is done, the coefficient multiplication can be executed in parallel with the addition of the data. The TMS320C30's floating-point capability eliminates not only the problems of roundoff error associated with fixed point processors but also the need for any truncation routines.

Because the DCT size is fixed to eight points, there are only four locations that need exchanging; this allows for a fast bit-reversal of the data. When using the TMS320C30's extended-precision registers for temporary storage, the transfers can be done in-place. These data transfers are also done in parallel, since two load or store operations can be performed simultaneously. The code for performing the bit reversal is shown in Figure 3 below.

```

*      CORRECT ORDER FROM BIT REVERSED TO NATURAL
*
BITREV  LDF      *AR0,R0      ;      ONLY FOUR LOCATIONS ARE
||      LDF      *-AR2,R1     ;      ACTUALLY SWITCHED
        STF      R1,*AR0
||      STF      R0,*-AR2
        LDF      *AR1,R0
||      LDF      *-AR3,R1
        STF      R1,*AR1
||      STF      R0,*-AR3

```

Figure 3. TMS320C30 Code for Bit Reversal

Because of the amount of data shuffling that occurs, an eight-word scratch-pad vector has been created with four permanent pointers set up at every other memory location. This allows access to each element in the vector (by predecrement or preincrement addressing) without requiring constant alteration of one or two pointer locations. Although there is no overhead for looping on the TMS320C30, straight-line coding is used as much as possible to increase performance.

You can transpose the DCT matrix in the same way as in the TMS320C25 implementation: namely, store the transformed row vector as a column vector in another matrix and interchange the input and output pointers.

The complete routines for the forward and inverse transforms are given in Appendix B.

Results

The execution times and memory requirements for the two routines are given in Table 1. For the TMS320C30 implementation, the forward transform contains the scale factor of $\frac{2}{N}$, so the transform is not unitary. When the signal flow is reversed, instructions accumulate and the time required to perform the inverse transform actually increases (see Table 1). This increase occurs because certain multiplications cannot be performed in parallel with another instruction. The two times are identical on a TMS320C25 because it uses a matrix routine to compute the transform.

Table 1. Execution Times and Memory Requirements[†]

Device	Memory Required		Time Required (μ s)
	Program	Data	
TMS320C25-50 (matrix)	232 words * 232 words	203 words * 203 words	205.8 (forward) 205.8 (inverse)
TMS320C30-40 (signal-flow)	125 words ** 112 words	138 words ** 137 words	33.6 (forward) 31.9 (inverse)
TMS320C30-40 (matrix)	115 word ** 115 words	128 words ** 128 words	65.8 (forward) 65.8 (inverse)

[†]Improvements have been made and are shown in this table. You may obtain the latest code from the BBS, (713) 274-2323.

* TMS320C25 wordlengths are 16 bits.

**TMS320C30 wordlengths are 32 bits.

Summary

Two routines for a two-dimensional Discrete Cosine Transform are presented: one for the TMS320C25 and one for the TMS320C30, with a development of the algorithm given for clarification. This report also discussed the similarities of the DCT to the Cooley-Tukey FFT algorithm and arithmetic shortcuts which can reduce the DCT's execution time. Although these implementations use the most recent formulation, there is still room for investigation into more efficient methods. Another approach that might prove fruitful is to deal with the entire 8×8 array all at once, as suggested by Haque [7], rather than transforming the array by rows and columns. However, both routines given in the appendices provide fast, numerically stable solutions for applications requiring the DCT.

Acknowledgements

The author thanks Steve Ford for supplying the original code for the TMS320C25 implementation. Francois Charlot helped in modifying the code for the TMS320C25, as well as in preparing this manuscript. Daniel Chen improved the performance of the code for both the TMS320C25 and the TMS320C30.

References

- [1] Ahmed, N., Natarajan, T., and Rao, K.R. "Discrete Cosine Transform," *IEEE Transactions on Computing*, vol. C-23, pp. 90-93, January 1974.
- [2] Perkins, M. "A Comparison of the Hartley, Cas-Cas, Fourier, and Discrete Cosine Transforms for Image Coding," *IEEE Transactions on Computing*, vol. 36, pp. 758-760, June 1988.
- [3] Hou, H.S. "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform," *IEEE Transactions on ASSP*, vol. ASSP-35, No. 10, pp. 1455-1461, October 1987.
- [4] Lee, B.G. "FCT - A Fast Cosine Transform," *Proceedings of 1984 Conference on ASSP*, pp. 28.A.3.1-28.A.3.4, March 1984.
- [5] Jayant, N.S., and Noll, P. *Digital Coding of Waveforms*, New York, Prentice-Hall, 1984.
- [6] Srinivasan, S., Jain, A.K., and Chin, T.M. "Cosine Transform Block Codec for Images Using the TMS32010," *Proceedings of IEEE ISCAS '86*, Cat. No. 86CH2255-8, vol. 1, pp. 299-302.
- [7] Haque, M.A. "A Two-Dimensional Fast Cosine Transform," *IEEE Transactions on ASSP*, vol. ASSP-33, pp. 1532-1539, December 1985.


```

RPTK 6
MNC C41,++
LTA ++,AR1
PPY C.30
AND RNDOFF
SACH ++,AR3
BANZ T4,+-,AR2

```

```

* FIFTH SET OF COEFFICIENTS
*

```

```

LAR AR1,SRC
LAR AR2,IST
ADRK AR2,IST 4
LARP 1
LARK AR3,7
LT ++
PPY C.40
ZAC
RPTK 6
MNC C41,++
LTA ++,AR2
PPY C.40
AND RNDOFF
SACH ++,AR3
BANZ T5,+-,AR1

```

```

* SIXTH SET OF COEFFICIENTS
*

```

```

LAR AR1,IST
ADRK 5
LARP 2
LAR AR2,SRC
LARK AR3,7
LT ++
PPY C.50
ZAC
RPTK 6
MNC C51,++
LTA ++,AR1
PPY C.50
AND RNDOFF
SACH ++,AR2
BANZ T6,+-,AR2

```

```

16

```

```

* SEVENTH SET OF COEFFICIENTS
*

```

```

LAR AR1,SRC
LAR AR2,IST
ADRK 6
LARP 1
LARK AR3,7
LT ++
PPY C.60
ZAC

```

```

17

```

```

RPTK 6
MNC C41,++
LTA ++,AR2
PPY C.60
AND RNDOFF
SACH ++,AR3
BANZ T7,+-,AR1

```

```

* EIGHTH SET OF COEFFICIENTS
*

```

```

LAR AR1,IST
ADRK 7
LARP 2
LAR AR2,SRC
LARK AR3,7
LT ++
PPY C.70
ZAC
RPTK 6
MNC C71,++
LTA ++,AR1
PPY C.70
AND RNDOFF
SACH ++,AR3
BANZ T8,+-,AR2

```

```

* LOOP FOR NEXT DIMENSION
*

```

```

LAC IST
DRIV SRC
SACL SRC
LARP AR7
BANZ DIMS,+-,AR1
STOP: CNFD
      B $
      .page

```

```

* DATAS - TABLES AND DECLARATIONS
*

```

```

.asect "MODEP",OFFWIN ; THIS IS TO SET UP THE LABELS FOR A CNFP
.label IDNITA ; ICT COEFFICIENTS
D00 .word 5792 ; FIRST ROW OF COEFFICIENTS
C01 .word 5792 ; 5792 = (1/4) * 2**(-1/2) IN Q15 FORMAT
C02 .word 5792
C03 .word 5792
C04 .word 5792
C05 .word 5792
C06 .word 5792
C07 .word 5792
C10 .word 8694
C11 .word 6811
C12 .word 4851

```

```

; CHANGE SOURCE AND DESTINATION POINTERS,
; SO RESULT OF FIRST PASS BECOMES OPERAND
; OF SECOND PASS. FINAL RESULT WILL BE IN
; P1CT
; LARP : DIMENSION COUNTER
; LOOP FOR NEXT DIMENSION
; STOP HERE

```

```

C13 .word 1598
C14 .word -1598
C15 .word -4551
C16 .word -6811
C17 .word -8034
C18 .word 7568
C19 .word 3134
C20 .word -3134
C21 .word -7568
C22 .word -3134
C23 .word -7568
C24 .word -3134
C25 .word 3134
C26 .word 7568
C27 .word 6811
C28 .word -1598
C29 .word -8034
C30 .word -4551
C31 .word 8034
C32 .word 1598
C33 .word -6811
C34 .word 5792
C35 .word -5792
C36 .word -5792
C37 .word 5792
C38 .word -5792
C39 .word 5792
C40 .word -5792
C41 .word -5792
C42 .word 5792
C43 .word -5792
C44 .word 5792
C45 .word -5792
C46 .word -5792
C47 .word 5792
C48 .word 4551
C49 .word -8034
C50 .word 1598
C51 .word 6811
C52 .word -8034
C53 .word -6811
C54 .word -1598
C55 .word 8034
C56 .word -4551
C57 .word 3134
C58 .word -7568
C59 .word 3134
C60 .word -7568
C61 .word -3134
C62 .word 7568
C63 .word -3134
C64 .word 7568
C65 .word -7568
C66 .word 3134
C67 .word 1598
C68 .word -4551
C69 .word 6811
C70 .word -8034
C71 .word -6811
C72 .word -8034
C73 .word -6811
C74 .word 8034
C75 .word -6811
C76 .word 4551
C77 .word -1598
      .table

; FIFTH ROW OF COEFFICIENTS
; 3134 = (1/4) * SIN(PI/8) IN Q15 FORMAT
; 7568 = (1/4) * COS(PI/8) IN Q15 FORMAT
; 1598 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 4551 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 6811 = (1/4) * SIN(3*PI/16) IN Q15 FORMAT
; 8034 = (1/4) * COS(3*PI/16) IN Q15 FORMAT
; 7568 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 3134 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 1598 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 4551 = (1/4) * COS(PI/16) IN Q15 FORMAT

; FOURTH ROW OF COEFFICIENTS
; 3134 = (1/4) * SIN(3*PI/16) IN Q15 FORMAT
; 7568 = (1/4) * COS(3*PI/16) IN Q15 FORMAT
; 1598 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 4551 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 6811 = (1/4) * SIN(3*PI/16) IN Q15 FORMAT
; 8034 = (1/4) * COS(3*PI/16) IN Q15 FORMAT
; 7568 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 3134 = (1/4) * COS(PI/16) IN Q15 FORMAT

; DATA DEFINITIONS
;
; COEFF
; .asect "COEFFS", 64
; .bss PICT_64
; .bss RESULT_64
; .bss ROUND_1
; .bss SRC_1
; .bss DST_1
; .bss C_00_1
; .bss C_10_1
; .bss C_20_1
; .bss C_30_1
; .bss C_40_1
; .bss C_50_1
; .bss C_60_1
; .bss C_70_1
; .end

; END OF COEFFICIENTS TABLE

; SEVENTH ROW OF COEFFICIENTS
; 3134 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 7568 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 1598 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 4551 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 6811 = (1/4) * SIN(3*PI/16) IN Q15 FORMAT
; 8034 = (1/4) * COS(3*PI/16) IN Q15 FORMAT
; 7568 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 3134 = (1/4) * COS(PI/16) IN Q15 FORMAT

; EIGHTH ROW OF COEFFICIENTS
; 3134 = (1/4) * SIN(3*PI/16) IN Q15 FORMAT
; 7568 = (1/4) * COS(3*PI/16) IN Q15 FORMAT
; 1598 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 4551 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 6811 = (1/4) * SIN(3*PI/16) IN Q15 FORMAT
; 8034 = (1/4) * COS(3*PI/16) IN Q15 FORMAT
; 7568 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 3134 = (1/4) * COS(PI/16) IN Q15 FORMAT

; SIXTH ROW OF COEFFICIENTS
; 3134 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 7568 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 1598 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 4551 = (1/4) * COS(PI/16) IN Q15 FORMAT
; 6811 = (1/4) * SIN(3*PI/16) IN Q15 FORMAT
; 8034 = (1/4) * COS(3*PI/16) IN Q15 FORMAT
; 7568 = (1/4) * SIN(PI/16) IN Q15 FORMAT
; 3134 = (1/4) * COS(PI/16) IN Q15 FORMAT

; ROUND OFF FACTOR
; ADDRESS OF PICTURE
; ADDRESS OF RESULT
; C00 COEFFICIENT
; C10 COEFFICIENT
; C20 COEFFICIENT
; C30 COEFFICIENT
; C40 COEFFICIENT
; C50 COEFFICIENT
; C60 COEFFICIENT
; C70 COEFFICIENT

; DCT COEFFICIENTS (GOES INTO 60)
; PICTURE
; RESULT, AFTER DCT
; ROUND OFF FACTOR
; SOURCE ADDRESS FOR CURRENT DCT LOOP
; DESTINATION ADDRESS
; C00 COEFFICIENT
; C10 COEFFICIENT
; C20 COEFFICIENT
; C30 COEFFICIENT
; C40 COEFFICIENT
; C50 COEFFICIENT
; C60 COEFFICIENT
; C70 COEFFICIENT

```


* SHAFFLE THE DATA ACCORDING TO PERMUTATION MATRIX P

```

* DCT
  LDI  A01,AR2      ; POINTS TO OUTPUT
  LDI  &COLAST,AR3
  LDI  &L0S,AR7    ; TABLE POINTER
*
  LDF  *AR0+*(I0),R0
  LDF  *AR1+*(I0),R1
  STF  *R0,AR2+*(1) ; GOING DOWN
  STF  *R1,AR3-(1)  ; GOING UP
  LDF  *AR0+*(I0),R0
  LDF  *AR1+*(I0),R1
  STF  *R0,AR2+*(1)
  STF  *R1,AR3-(1)
  LDF  *AR0+*(I0),R0
  LDF  *AR1+*(I0),R1
  STF  *R0,AR2+*(1)
  STF  *R1,AR3-(1)

```

* MODIFIED FFT ALGORITHM

```

*
  LDI  A01,AR0      ; POINT TO OUTPUT
  LDI  A00,AR1
  ANDI 2,AR1
  LDI  A01,AR2
  ANDI 2,AR2
  LDI  A02,AR3
  ANDI 2,AR3
*
  LDF  *AR2,R2      ; THESE SECTIONS PERFORM
  LDF  *AR2,R3      ; TWO BUTTERFLIES AT ONCE
  SUBF3 *AR2,*AR0,R1
  SUBF3 *AR2,*AR0,R0
  MPYF3 R1,*AR7+*(1),R1 ;
  ADDF3 R3,*AR0,R3 ; X(0)
  MPYF3 R0,*AR7+*(1),R0 ; X(1) AR0
  ADDF3 R2,*AR0,R2 ; X(2)
  STF  R1,*AR2      ; X(3) AR1
  STF  R2,*AR0      ; X(4)
  STF  R0,*AR2      ; X(5) AR2
  STF  R3,*AR0      ; X(6)
  LDF  *AR3,R2      ; X(7) AR3
  LDF  *AR3,R3
  SUBF3 *AR3,*AR1,R1
  SUBF3 *AR3,*AR1,R0
  MPYF3 R1,*AR7+*(1),R1
  ADDF3 R3,*AR1,R3
  MPYF3 R0,*AR7+*(1),R0
  ADDF3 R2,*AR1,R2

```

```

  STF  R1,*AR3
  STF  R2,*AR1
  STF  R0,*AR3
  STF  R3,*AR1
*
  * SECOND GROUP OF BUTTERFLIES
  *
  * THIS IS THE SAME AS ABOVE EXCEPT THE
  * POINTERS CHANGE
  *
  LDF  *AR1,R2
  LDF  *AR1,R3
  SUBF3 *AR1,*AR0,R1
  SUBF3 *AR1,*AR0,R0
  MPYF3 R1,*AR7+*(1),R1
  ADDF3 R3,*AR0,R3
  MPYF3 R0,*AR7-(1),R0
  ADDF3 R2,*AR0,R2
  STF  R1,*AR1
  STF  R2,*AR0
  STF  R0,*AR1
  STF  R3,*AR0
  LDF  *AR3,R3
  LDF  *AR3,R2
  SUBF3 *AR3,*AR2,R1
  SUBF3 *AR3,*AR2,R0
  MPYF3 R1,*AR7+*(1),R1
  ADDF3 R3,*AR2,R3
  MPYF3 R0,*AR7+*(1),R0
  ADDF3 R2,*AR2,R2
  STF  R1,*AR3
  STF  R2,*AR2
  STF  R0,*AR3
  STF  R3,*AR2
*
  * LAST SET OF BUTTERFLIES
  *
  LDF  *AR0,R2
  LDF  *AR1,R3
  SUBF3 *AR0,*AR0,R1
  SUBF3 *AR1,*AR1,R0
  MPYF3 R1,*AR7,R1
  ADDF3 R3,*AR1,R3
  MPYF3 R0,*AR7,R0
  ADDF3 R2,*AR0,R2
  STF  R1,*AR0
  STF  R2,*AR0
  STF  R3,*AR1
  STF  R0,*AR1
  LDF  *AR2,R2
  LDF  *AR3,R3
  SUBF3 *AR2,*AR2,R1
  SUBF3 *AR3,*AR3,R0
  MPYF3 R1,*AR7,R1
  ADDF3 R3,*AR3,R3
  MPYF3 R0,*AR7,R0
  ADDF3 R2,*AR2,R2

```

```

:: R1, #R2
STF R2, #R2
STF R3, #R3
STF R0, #R3
* CORRECT ORDER FROM BIT-REVERSED TO NATURAL
* BITREV LDF #R0, R0 ; ONLY TWO LOCATIONS ARE ACTUALLY SWITCHED
LDF #R2, R1
LDF R1, #R0
STF R0, #R2
LDF #R1, R0
LDF #R3, R1
LDF R1, #R1
STF R0, #R3
* CONTINUE WITH RECURSIVE ALGORITHM
* RECURSE MPYF3 R7, #R3, R2
MPYF3 R7, #R3, R1
SUBF3 #R1, R2, R2 ; 2X(7)-X(3)
SUBF3 #R1, R1, R1 ; 2X(8)-X(4)
STF R1, #R3
STF R2, #R3
* LASTLOOP MPYF3 R7, #R1, R0 ; X(4)=2X(4)
MPYF3 R7, #R2, R1 ; X(6)=2X(6)
SUBF3 #R0, R0, R2 ; R2=2X(4)-X(2)
MPYF3 R7, #R3, R3 ; R3=2X(8)
STF R2, #R1
SUBF3 #R1, R1, R1 ; R1=2X(6)-X(4)
SUBF3 R1, R3, R3 ; R3=2X(8)-X(6)
STF R1, #R2
STF R3, #R3
* SCALE FACTOR OF (2/N)=0.25
MPYF3 R6, #R3, R0
STF R0, #R3-(1)
MPYF3 R6, #R3, R1
STF R1, #R3-(1)
MPYF3 R6, #R3, R0
STF R0, #R3-(1)
MPYF3 R6, #R3, R1
STF R1, #R3-(1)
MPYF3 R6, #R3, R0
STF R0, #R3-(1)
MPYF3 R6, #R3, R1
STF R1, #R3
* OK TO MOVE ARG

```

```

* CORRECT X(0) IF NONZERO
*
* EXIT
LDF #R4, R4 ; RETURN
LDF #R7, R0, R0 ; MULT BY 1/SQRT(2)
STF R0, #R0 ; STORE THE RESULT
*
* COSTAB
float 0.980783280403 ; LAMBDA
float 0.535570233019 ; MU
float -0.195090322016 ; NU
float -0.831469412303 ; -GAMMA
float 0.922879532511 ; BETA
float -0.362660422585 ; -DELTA
float 0.707106781186 ; ALPHA
*
*
*

```



```

::
STF R1,=AR3
STF R2,=AR2
STF R0,AR3
STF R3,AR2
*
* LAST SET OF BUTTERFLIES
*
LDF =AR2,R2
LDF =AR2,R3
SUBF3 =AR2,AR0,R0
ADDF3 R3,AR0,R3
STF R1,=AR2
STF R2,=AR0
STF R0,AR2
STF R3,AR0
LDF =AR3,R2
LDF =AR3,R3
SUBF3 =AR3,=AR1,R1
ADDF3 R3,AR1,R3
ADDF3 R2,=AR1,R2
STF R1,=AR3
STF R2,=AR1
STF R0,AR3
STF R3,AR1
*
* SHUFFLE THE DATA ACCORDING TO PERMUTATION MATRIX P
*
LDI AR4,AR0
LDI ARN,AR1
ADDI 1,AR1
LDI AR5,AR2
LDI 7,AR3
ADDI AR2,AR3,AR3
LDF =AR2++(1),R0
LDF =AR3--(1),R1
STF R0,AR0++(1),R0
STF R1,AR1++(1),R0
LDF =AR2++(1),R0
LDF =AR3--(1),R1
STF R0,AR0++(1),R0
STF R1,AR1++(1),R0
LDF =AR2++(1),R0
LDF =AR3--(1),R1
BLD RN
STF R0,AR0++(1),R0
STF R1,AR1++(1),R0
LDF =AR2++(1),R0
LDF =AR3--(1),R1
STF R0,AR0++(1),R0
STF R1,AR1++(1),R0
*end

```

POINTERS ARE SET AS FOLLOWS:

```

.global COS_TAN
.data
float 0.707106781188 ; ALPHA
float -0.923879532511 ; BETA
float -0.36606842245 ; -DEL TA
float -0.195090322016 ; -NU
float -0.831469612303 ; -GAMMA
float 0.980785280403 ; LAMBDA
float 0.333570230019 ; MU
.end

```

```

.global COS_TAN
.data
float 0.707106781188 ; ALPHA
float -0.923879532511 ; BETA
float -0.36606842245 ; -DEL TA
float -0.195090322016 ; -NU
float -0.831469612303 ; -GAMMA
float 0.980785280403 ; LAMBDA
float 0.333570230019 ; MU
.end

```