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

TEXAS INSTRUMENTS

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

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 $x_0, x_1, \ldots, x_{N-1}$, the discrete cosine transform is given in [1] as

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

and its inverse is

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

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

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

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

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

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

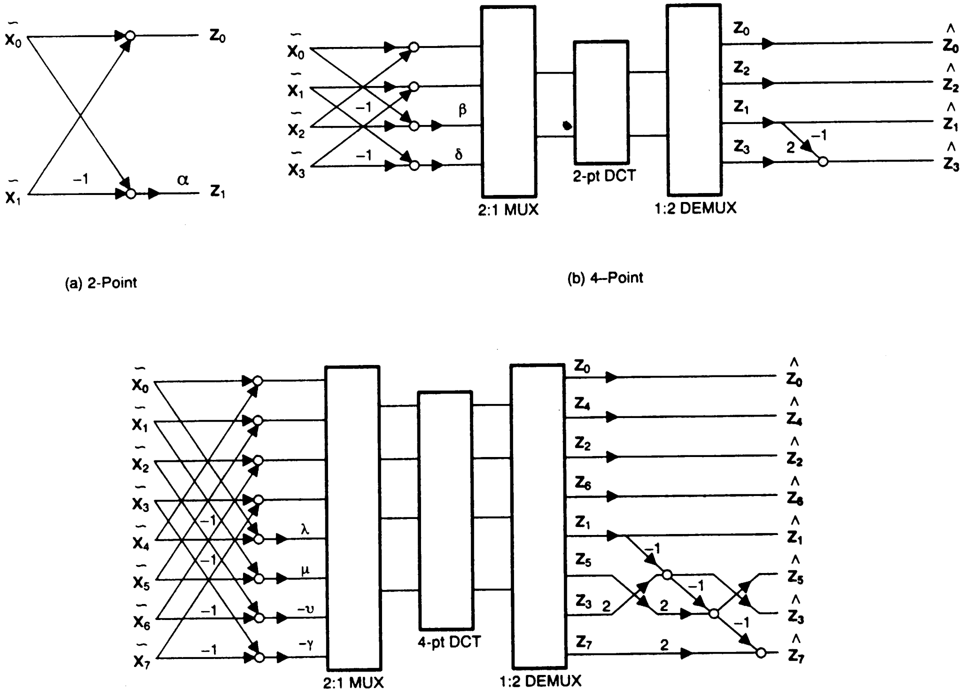
$$\begin{bmatrix} z_0 \\ z_4 \\ z_2 \\ z_6 \\ z_1 \\ z_5 \\ z_3 \\ z_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha & -\alpha & \alpha & -\alpha & \alpha & -\alpha & \alpha & -\alpha \\ \beta & -\delta & -\beta & \delta & \beta & -\delta & -\beta & \delta \\ \delta & \beta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

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

$$\tilde{x} = Px, \tag{5}$$

where

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



(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

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

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

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

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

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:

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

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 (μs)
	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 Tranform," *IEEE Transactions on ASSP,* vol. ASSP-33, pp. 1532-1539, December 1985.

Appendix A. DCT Algorithm for the TMS320C25

```
***************************************************************************
*
*    8 X 8 2D-DCT ALGORITHM FOR THE TMS320C25
*
*    THIS PROGRAM WILL PERFORM A TWO-DIMENSIONAL DCT ON EIGHT-BIT IMAGE DATA
*    AND NORMALIZE THE DATA TO MINIMIZE TRUNCATION AND ROUNDOFF.
*
*
***************************************************************************
*
         .title   '8x8 DCT'

*
*    RESET: BRANCH TO DCT, AND SET ARP TO 0
*
         .sect    "RESET"
         B        DCTINI,*,AR1
         .text
*
*    INITIALIZE MATRIX COEFFICIENTS AND ROUNDOFF VALUES INTO INTERNAL BLOCK B0
*
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,*+
*
*    HERE IS THE DCT FUNCTION
*
DCT      LARK     AR7,1     ; AR7: DIMENSION-1
         LARK     AR0,8     ; POINTER INCREMENT FOR DATA TRANSPOSITION
         CNFP               ; "MAC" NEEDS 1 OPERAND IN PROGRAM MEMORY
*
*    LOOP FOR DIMENSIONS
*
DIMS     .equu    $
*
*    FIRST SET OF COEFFICIENTS
*
         LARK     AR3,7     ; COUNT FOR 8 1-D DCTs
         LAR      AR1,SRC   ; SOURCE ADDRESS
         LAR      AR2,DST   ; DESTINATION ADDRESS (FIRST COLUMN)
         LT       *+        ; TREG = X0
         MPY      C_00      ; ACC = 0 , PREG= x0 * c00
T1       ZAC
         RPTK     6
         MAC      C01,*+    ; ACC = 0 ,PREG= X0 * C00
*
         LTA      *+,AR2    ; INCLUDE LAST PRODUCT AND LOAD PREG
         MPY      C_00
         ADD      RNDOFF
         SACH     *0+,AR3   ; STORE RESULT AND TRANSPOSE
         BANZ     T1,*-,AR1
*
*    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      C_10
T2       ZAC
         RPTK     6
         MAC      C11,*+
*
         LTAS     *+,AR1
         MPY      C_10
         ADD      RNDOFF
         SACH     *0+,AR3
         BANZ     T2,*-,AR2
*
*    THIRD SET OF COEFFICIENTS
*
         LAR      AR1,SRC   ; AR1 NOW SOURCE POINTER
         LAR      AR2,DST
         ADRK     2         ; THIRD COLUMN
         LARP     1         ; ACTIVATE AR1
         LARK     AR3,7
         LT       *+
         MPY      C_20
T3       ZAC
         RPTK     6
         MAC      C21,*+
         LTA      *+,AR2
         MPY      C_20
         ADD      RNDOFF
         SACH     *0+,AR3
         BANZ     T3,*-,AR1
*
*    FOURTH SET OF COEFFICIENTS
*
         LAR      AR1,DST
         ADRK     3
         LARP     2
         LAR      AR2,SRC
         LARK     AR3,7
         LT       *+
         MPY      C_30
T4       ZAC
```

```
        RPTK    6
        MAC     C31,*+
        LTA     *+,AR1
        MPY     C_30
        ADD     RNDOFF
        SACH    *0+,AR3
        BANZ    T4,*-,AR2
*
*   FIFTH SET OF COEFFICIENTS
*
        LAR     AR1,SRC
        LAR     AR2,DST
        ADRK    4
        LARP    1
        LARK    AR3,7
        LT      *+
        MPY     C_40
T5      ZAC
        RPTK    6
        MAC     C41,*+
        LTA     *+,AR2
        MPY     C_40
        ADD     RNDOFF
        SACH    *0+,AR3
        BANZ    T5,*-,AR1
*
*   SIXTH SET OF COEFFICIENTS
*
        LAR     AR1,DST
        ADRK    5
        LARP    2
        LAR     AR2,SRC
        LARK    AR3,7
        LT      *+
        MPY     C_50
T6      ZAC
        RPTK    6
        MAC     C51,*+
        LTA     *+,AR1
        MPY     C_50
        ADD     RNDOFF
        SACH    *0+,AR3
        BANZ    T6,*-,AR2
*
*   SEVENTH SET OF COEFFICIENTS
*
        LAR     AR1,SRC
        LAR     AR2,DST
        ADRK    6
        LARP    1
        LARK    AR3,7
        LT      *+
        MPY     C_60
T7      ZAC
        RPTK    6
        MAC     C61,*+
        LTA     *+,AR2
        MPY     C_60
        ADD     RNDOFF
        SACH    *0+,AR3
        BANZ    T7,*-,AR1
*
*   EIGHTH SET OF COEFFICIENTS
*
        LAR     AR1,DST
        ADRK    7
        LARP    2
        LAR     AR2,SRC
        LARK    AR3,7
        LT      *+
        MPY     C_70
T8      ZAC
        RPTK    6
        MAC     C71,*+
        LTA     *+,AR1
        MPY     C_70
        ADD     RNDOFF
        SACH    *0+,AR3
        BANZ    T8,*-,AR2
*
*   LOOP FOR NEXT DIMENSION
*
        LAC     DST             ; CHANGE SOURCE AND DESTINATION POINTERS,
        DMOV    SRC             ; SO RESULT OF FIRST PASS BECOMES OPERAND
        SACL    SRC             ; OF SECOND PASS. FINAL RESULT WILL BE IN
*                               ; PICT
        LARP    AR7             ; AR7 : DIMENSION COUNTER
        BANZ    DIMS,*-,AR1     ; LOOP FOR NEXT DIMENSION
*
STOP:   CNFD
        B       $               ; STOP HERE
        .page
*
*   DATAS - TABLES AND DECLARATIONS
*
        .asect  "RCOEF",OFF00h  ; THIS IS TO SET UP THE LABELS FOR A CNFP
        .label  IDATA           ; DCT COEFFICIENTS
C00     .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   8034            ; SECOND ROW OF COEFFICIENTS
C11     .word   6811
C12     .word   4551
```

```
C13     .word   1598
C14     .word   -1598       ; 1598 = (1/4) * SIN(PI/16) IN Q15 FORMAT
C15     .word   -4551       ; 4551 = (1/4) * SIN(3PI/16) IN Q15 FORMAT
C16     .word   -6811       ; 6811 = (1/4) * COS(3PI/16) IN Q15 FORMAT
C17     .word   -8034       ; 8034 = (1/4) * COS(PI/16) IN Q15 FORMAT
C20     .word   7568        ; third row of coefficients
C21     .word   3134        ; 3134 = (1/4) * SIN(PI/8) IN Q15 FORMAT
C22     .word   -3134       ; 7568 = (1/4) * COS(PI/8) IN Q15 FORMAT
C23     .word   -7568
C24     .word   -7568
C25     .word   -3134
C26     .word   3134
C27     .word   7568
C30     .word   6811        ; FOURTH ROW OF COEFFICIENTS
C31     .word   -1598
C32     .word   -8034
C33     .word   -4551
C34     .word   4551
C35     .word   8034
C36     .word   1598
C37     .word   -6811
C40     .word   5792        ; FIFTH ROW OF COEFFICIENTS
C41     .word   -5792
C42     .word   -5792
C43     .word   5792
C44     .word   5792
C45     .word   -5792
C46     .word   -5792
C47     .word   5792
C50     .word   4551        ; SIXTH ROW OF COEFFICIENTS
C51     .word   -8034
C52     .word   1598
C53     .word   6811
C54     .word   -6811
C55     .word   -1598
C56     .word   8034
C57     .word   -4551
C60     .word   3134        ; SEVENTH ROW OF COEFFICIENTS
C61     .word   -7568
C62     .word   7568
C63     .word   -3134
C64     .word   -3134
C65     .word   7568
C66     .word   -7568
C67     .word   3134
C70     .word   1598        ; EIGHTH ROW OF COEFFICIENTS
C71     .word   -4551
C72     .word   6811
C73     .word   -8034
C74     .word   8034
C75     .word   -6811
C76     .word   4551
C77     .word   -1598
        .label  EDATA       ; END OF COEFFICIENTS TABLE

        .word   12288       ; ROUNDOFF FACTOR
        .word   PICT        ; ADDRESS OF PICTURE
        .word   RESULT      ; ADDRESS OF RESULT
        .word   5792        ; C00 COEFFICIENT
        .word   8034        ; C10 COEFFICIENT
        .word   7568        ; C20 COEFFICIENT
        .word   6811        ; C30 COEFFICIENT
        .word   5792        ; C40 COEFFICIENT
        .word   4551        ; C50 COEFFICIENT
        .word   3134        ; C60 COEFFICIENT
        .word   1598        ; C70 COEFFICIENT
*
*   DATA DEFINITIONS
*
COEFF   .usect  "COEFFS",64 ; DCT COEFFICIENTS (GOES INTO B0)
        .BSS    PICT,64     ; PICTURE
        .BSS    RESULT,64   ; RESULT, AFTER DCT
        .BSS    RNDOFF,1    ; ROUNDOFF FACTOR
        .BSS    SRC,1       ; SOURCE ADDRESS FOR CURRENT DCT LOOP
        .BSS    DST,1       ; DESTINATION ADDRESS
        .BSS    C_00,1      ; C00 COEFFICIENT
        .BSS    C_10,1      ; C10 COEFFICIENT
        .BSS    C_20,1      ; C20 COEFFICIENT
        .BSS    C_30,1      ; C30 COEFFICIENT
        .BSS    C_40,1      ; C40 COEFFICIENT
        .BSS    C_50,1      ; C50 COEFFICIENT
        .BSS    C_60,1      ; C60 COEFFICIENT
        .BSS    C_70,1      ; C70 COEFFICIENT
*
        .end
```

Appendix B. DCT Algorithms for the TMS320C30

```
**********************************************************************
*
*    TITLE: 2-D DISCRETE COSINE TRANSFORM, (8x8) VERSION 1.0
*
*    AUTHOR: WILLIAM HOHL
*
*
*    THIS PROGRAM IS BASED ON A RECENT ALGORITHM PROPOSED BY H.S. HOU
*    (TRANSACTIONS ON ASSP, VOL. ASSP-35, NO. 10, OCTOBER 1987, PP. 1455-
*    1461).
*
*    INPUT MATRIX IS STORED IN RAM, AND THE RESULTS ARE STORED IN THE SAME
*    LOCATION.
*
**********************************************************************
*
        .BSS      OUT,64
        .BSS      INP,64
        .BSS      SCR,8            ; SCRATCHPAD MEMORY
        .global   COSTAB
        .global   START
        .data
*
_COS    .word     COSTAB
INPUT   .word     INP
OUTPUT  .word     OUT
SCRATCH .word     SCR
SCRLAST .word     SCR+7
RTN1    .word     TRANS1
RTN2    .word     TRANS2
        .text
*
START   LDI       7,RC
        LDI       2,IR0
        LDI       8,IR1
        LDI       8,BK             ; SET BUFFER LENGTH=8
        LDP       @SCRATCH
        LDI       @SCRATCH,AR4
        LDI       @OUTPUT,AR6      ; VARIABLE LOCATIONS
        LDI       @INPUT,AR5       ; HOLDS INPUT MATRIX
        LDF       0.25,R6          ; CONSTANT 0.25
        LDF       2.0,R7           ; CONSTANT 2.0
*
        LDI       @RTN1,R4         ; RETURN ADDRESS OF SUBROUTINE
        RPTB      BLK1
        BRD       DCT
        LDI       AR5,AR0          ; POINTS TO INPUT
        LDI       AR5,AR1
        ADDI      1,AR1
*
TRANS1: LDF       *AR4++(1)%,R1    ; TRANSPOSE THE ROWS
        STF       R1,*AR6++(IR1)   ; INTO COLUMNS
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR5++(IR1),R5
*
BLK1    SUBI      63,AR6
*
        LDI       @SCRATCH,AR4
        LDI       @OUTPUT,AR5      ; DO DCT ON COLUMN
        LDI       @INPUT,AR6       ; VECTORS
        LDI       7,RC
*
        LDI       @RTN2,R4         ; RETURN ADDRESS OF SUBROUTINE
        RPTB      BLK3
        BRD       DCT
        LDI       AR5,AR0          ; POINTS TO INPUT
        LDI       AR5,AR1
        ADDI      1,AR1
*
TRANS2: LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR5++(IR1),R5
*
BLK3    SUBI      63,AR6           ; INCREMENT POINTERS
*
END     BR        END              ; END
*
```

```
*    SHUFFLE THE DATA ACCORDING TO PERMUTATION MATRIX P
*
DCT     LDI     AR4,AR2         ; POINTS TO OUTPUT
        LDI     @SCRLAST,AR3
        LDI     @_COS,AR7       ; TABLE POINTER
*
        LDF     *AR0++(IR0),R0
||      LDF     *AR1++(IR0),R1
        STF     R0,*AR2++(1)    ; GOING DOWN
||      STF     R1,*AR3--(1)    ; GOING UP
        LDF     *AR0++(IR0),R0
||      LDF     *AR1++(IR0),R1
        STF     R0,*AR2++(1)
||      STF     R1,*AR3--(1)
        LDF     *AR0++(IR0),R0
||      LDF     *AR1++(IR0),R1
        STF     R0,*AR2++(1)
||      STF     R1,*AR3--(1)
        LDF     *AR0++(IR0),R0
||      LDF     *AR1++(IR0),R1
        STF     R0,*AR2++(1)
||      STF     R1,*AR3--(1)
*
*    MODIFIED FFT ALGORITHM
*
        LDI     AR4,AR0         ; POINT TO OUTPUT
        ADDI    1,AR0
        LDI     AR0,AR1
        ADDI    2,AR1           ; SET UP POINTERS
        LDI     AR1,AR2
        ADDI    2,AR2
        LDI     AR2,AR3
        ADDI    2,AR3
*
        LDF     *-AR2,R2        ; THESE SECTIONS PERFORM
||      LDF     *AR2,R3         ; TWO BUTTERFLIES AT ONCE
        SUBF3   *-AR2,*-AR0,R1
        SUBF3   *AR2,*AR0,R0    ; POINTERS ARE SET AS FOLLOWS:
        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
*
        LDF     *-AR1,R2        ; THIS IS THE SAME AS ABOVE EXCEPT THE
||      LDF     *AR1,R3         ;   POINTERS CHANGE
        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,R2
||      LDF     *AR3,R3
        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
```

```
        STF     R1,*AR2
||      STF     R2,*-AR2
        STF     R3,*-AR3
||      STF     R0,*AR3
*
*   CORRECT ORDER FROM BIT-REVERSED TO NATURAL
*
BITREV  LDF     *AR0,R0         ; ONLY TWO LOCATIONS ARE ACTUALLY SWITCHED
||      LDF     *-AR2,R1
        STF     R1,*AR0
||      STF     R0,*-AR2
        LDF     *AR1,R0
||      LDF     *-AR3,R1
        STF     R1,*AR1
||      STF     R0,*-AR3
*
*   CONTINUE WITH RECURSIVE ALGORITHM
*
RECURSE MPYF3   R7,*-AR3,R2
        MPYF3   R7,*AR3,R1
||      SUBF3   *-AR1,R2,R2     ; 2X(7)-X(3)
        SUBF3   *AR1,R1,R1      ; 2X(8)-X(4)
        STF     R1,*AR3
||      STF     R2,*-AR3
*
LASTLOOP MPYF3  R7,*AR1,R0      ; X(4)=2*X(4)
        MPYF3   R7,*AR2,R1      ; X(6)=2*X(6)
||      SUBF3   *AR0,R0,R2      ; R2=2X(4)-X(2)
        MPYF3   R7,*AR3,R3      ; R3=2*X(8)
||      STF     R2,*AR1
        SUBF3   *AR1,R1,R1      ; R1=2X(6)-X(4)
        SUBF3   R1,R3,R3        ; R3=2X(8)-X(6)
        STF     R1,*AR2
||      STF     R3,*AR3
*
*   SCALE FACTOR OF (2/N)=0.25
*
        MPYF3   R6,*AR3,R0
        STF     R0,*AR3--(1)
||      MPYF3   R6,*-AR3,R1
        STF     R1,*AR3--(1)
||      MPYF3   R6,*-AR3,R0
        STF     R0,*AR3--(1)
||      MPYF3   R6,*-AR3,R1
        STF     R1,*AR3--(1)
||      MPYF3   R6,*-AR3,R0
        STF     R0,*AR3--(1)    ; OK TO MOVE AR3
||      MPYF3   R6,*-AR3,R1
        STF     R1,*AR3--(1)
||      MPYF3   R6,*-AR3,R0
        STF     R0,*AR3--(1)
||      MPYF3   R6,*-AR3,R1
        STF     R1,*AR3
*
```

```
*   CORRECT X(0) IF NONZERO
*
EXIT    BUD     R4              ; RETURN
        LDF     *-AR0,R0
        MPYF3   *AR7,R0,R0      ; MULT BY 1/SQRT(2)
        STF     R0,*-AR0        ; STORE THE RESULT
        .end
*
COSTAB  .float  0.980785280403 ; LAMBDA
        .float  0.555570233019 ; MU
        .float  -0.195090322016 ; -NU
        .float  -0.831469612303 ; -GAMMA
        .float  0.923879532511 ; BETA
        .float  -0.382683432365 ; -DELTA
        .float  0.707106781188 ; ALPHA
        .end
```

```
************************************************************************
*
*    TITLE: 2-D INVERSE DISCRETE COSINE TRANSFORM, (8x8) VERSION 1.0
*
*    AUTHOR: WILLIAM HOHL
*
*
*    THIS PROGRAM IS BASED ON A RECENT ALGORITHM PROPOSED BY H.S. HOU
*    (TRANSACTIONS ON ASSP, VOL. ASSP-35, NO. 10, OCTOBER 1987, PP. 1455-
*    1461).
*
*    INPUT MATRIX IS STORED IN RAM, AND THE RESULTS ARE STORED IN THE SAME
*    LOCATION.
*
************************************************************************
*
        .BSS      OUT,64
        .BSS      INP,64
        .BSS      SCR,8
        .global   COS_TAB
        .global   START
        .data
*
_COS    .word     COS_TAB
INPUT   .word     INP
OUTPUT  .word     OUT
SCRATCH .word     SCR
RTN1    .word     TRANS1
RTN2    .word     TRANS2
        .text
*
START   LDI       7,RC
        LDI       2,IR0
        LDI       8,IR1
        LDF       2.0,R7          ; MULTIPLIER
        LDI       8,BK            ; SET BUFFER LENGTH=64
        LDP       @OUTPUT
        LDI       @OUTPUT,AR6     ; VARIABLE LOCATIONS
        LDI       @SCRATCH,AR4
        LDI       @INPUT,AR5      ; HOLDS INPUT MATRIX
*
        LDI       @RTN1,R4        ; RETURN ADDRESS OF SUBROUTINE
        RPTB      BLK1
        BRD       IDCT
        LDI       AR5,AR0         ; POINT TO INPUT
        LDI       @_COS,AR7       ; TABLE POINTER
        ADDI      1,AR0
*
TRANS1: LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR5++(IR1),R5
*
BLK1    SUBI      63,AR6
*
        LDI       @INPUT,AR6      ; REALIGN POINTERS
        LDI       @OUTPUT,AR5
        LDI       @SCRATCH,AR4
        LDI       7,RC
*
        LDI       @RTN2,R4        ; RETURN ADDRESS OF SUBROUTINE
        RPTB      BLK6
        BRD       IDCT
        LDI       AR5,AR0         ; POINT TO INPUT
        LDI       @_COS,AR7       ; TABLE POINTER
        ADDI      1,AR0
*
TRANS2: LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR4++(1)%,R1
        STF       R1,*AR6++(IR1)
||      LDF       *AR5++(IR1)R5
*
BLK6    SUBI      63,AR6
*
END     BR        END             ; END
*
*    CORRECT X(0) IF NONZERO
*
```

```
IDCT    LDI     AR0,AR1
        ADDI    2,AR1
        LDI     AR1,AR2
        ADDI    2,AR2
        LDI     AR2,AR3
        ADDI    2,AR3
*
        LDF     *-AR0,R0
        MPYF3   *AR7,R0,R0      ; MULT BY 1/SQRT(2)
        STF     R0,*-AR0        ; STORE THE RESULT
*
*   BEGIN WITH RECURSION
*
        SUBF3   *AR3,*AR2,R2    ; X(6)-X(8)
        SUBF3   R2,*AR1,R3      ; X(4)-X(6)
        MPYF3   *AR3,R7,R0      ; 2X(8)->R0
||      STF     R2,*AR2
        SUBF3   R3,*AR0,R2      ; X(2)-X(4)
        MPYF3   *AR2,R7,R1      ; 2*X(6)->R1
||      STF     R3,*AR1
        STF     R0,*AR3
||      STF     R1,*AR2
        MPYF3   *AR1,R7,R0
        STF     R0,*AR1
||      STF     R2,*AR0
SECLOOP SUBF3   *-AR3,*-AR1,R2 ; X(3)-X(7)
        SUBF3   *AR3,*AR1,R3    ; X(4)-X(8)
        MPYF3   R7,*-AR3,R0     ; 2*X(7)
||      STF     R2,*-AR1
        MPYF3   R7,*AR3,R1      ; 2*X(8)
||      STF     R3,*AR1
        STF     R0,*-AR3
||      STF     R1,*AR3
*
*   CORRECT ORDER FROM NATURAL TO BIT-REVERSED
*
BITREV  LDF     *AR0,R0         ; ONLY TWO LOCATIONS ARE ACTUALLY SWITCHED
||      LDF     *-AR2,R1
        STF     R1,*AR0
||      STF     R0,*-AR2
        LDF     *AR1,R0
||      LDF     *-AR3,R1
        STF     R1,*AR1
||      STF     R0,*-AR3
*
*   FIRST SET OF BUTTERFLIES
*
        LDF     *AR0,R0
||      LDF     *AR1,R1
        LDF     *AR2,R2
||      LDF     *AR3,R3
        MPYF3   *AR7,R1,R1      ; PERFORM THE ALPHA MULT'S
        MPYF3   *AR7,R0,R0
        MPYF3   *AR7,R2,R2
        MPYF3   *AR7++(1),R3,R3  ; SKIP TO NEXT COEFF
        STF     R1,*AR1
||      STF     R0,*AR0
        STF     R2,*AR2
||      STF     R3,*AR3
        LDF     *AR0,R2         ; THESE SECTIONS PERFORM
||      LDF     *AR1,R3         ; TWO BUTTERFLIES AT ONCE
        SUBF3   *AR0,*-AR0,R0
        SUBF3   *AR1,*-AR1,R1
        STF     R0,*AR0
        MPYF3   R1,*+AR7,R1     ; -DELTA
        ADDF3   R3,*-AR1,R0
        MPYF3   R0,*AR7,R0      ; BETA
||      ADDF3   R2,*-AR0,R2
        STF     R2,*-AR0
||      STF     R0,*-AR1
        STF     R1,*AR1
*
        LDF     *AR2,R2
||      LDF     *AR3,R3
        SUBF3   *AR2,*-AR2,R0
        SUBF3   *AR3,*-AR3,R1
        STF     R0,*AR2
        MPYF3   R1,*+AR7,R1     ; -DELTA ON NEXT GROUP
        ADDF3   R3,*-AR3,R0
        MPYF3   R0,*AR7++(IR0),R0  ; BETA ON NEXT GROUP
||      ADDF3   R2,*-AR2,R2
        STF     R2,*-AR2
||      STF     R0,*-AR3
        STF     R1,*AR3
*
*   SECOND GROUP OF BUTTERFLIES
*
        LDF     *-AR1,R2        ; THIS IS THE SAME AS ABOVE, EXCEPT THE
||      LDF     *AR1,R3         ; POINTERS CHANGE
        SUBF3   *-AR1,*-AR0,R1
        SUBF3   *AR1,*AR0,R0
        ADDF3   R3,*AR0,R3
        ADDF3   R2,*-AR0,R2
        STF     R1,*-AR1
||      STF     R2,*-AR0
        STF     R0,*AR1
||      STF     R3,*AR0
*
        LDF     *-AR3,R2
||      LDF     *AR3,R3
        SUBF3   *-AR3,*-AR2,R1
        SUBF3   *AR3,*AR2,R0
        MPYF3   *AR7++(1),R1,R1  ; -NU
||      ADDF3   R3,*AR2,R3
        MPYF3   *AR7++(1),R0,R0  ; -GAMMA
||      ADDF3   R2,*-AR2,R2
        MPYF3   R2,*AR7++(1),R2  ; LAMBDA
        MPYF3   R3,*AR7,R3      ; MU
```

```
        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,R1
        SUBF3   *AR2,*AR0,R0     ; POINTERS ARE SET AS FOLLOWS:
        ADDF3   R3,*AR0,R3       ; X(0)
        ADDF3   R2,*-AR0,R2      ; X(1)  AR0
        STF     R1,*-AR2         ; X(2)
||      STF     R2,*-AR0         ; X(3)  AR1
        STF     R0,*AR2          ; X(4)
||      STF     R3,*AR0          ; X(5)  AR2
        LDF     *-AR3,R2         ; X(6)
||      LDF     *AR3,R3          ; X(7)  AR3
        SUBF3   *-AR3,*-AR1,R1
        SUBF3   *AR3,*AR1,R0
        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          ; POINTS TO SCRATCH
        LDI     AR4,AR1
        ADDI    1,AR1
        LDI     AR5,AR2          ; POINTS TO INPUT
        LDI     7,AR3            ; VECTOR
        ADDI    AR2,AR3,AR3
        LDF     *AR2++(1),R0     ; GOING UP
||      LDF     *AR3--(1),R1     ; GOING DOWN
        STF     R0,*AR0++(IR0)
||      STF     R1,*AR1++(IR0)
        LDF     *AR2++(1),R0
||      LDF     *AR3--(1),R1
        STF     R0,*AR0++(IR0)
||      STF     R1,*AR1++(IR0)
        LDF     *AR2++(1),R0
||      LDF     *AR3--(1),R1
        BUD     R4               ; RETURN HOME
        STF     R0,*AR0++(IR0)
||      STF     R1,*AR1++(IR0)
        LDF     *AR2++(1),R0
||      LDF     *AR3--(1),R1
        STF     R0,*AR0++(IR0)
||      STF     R1,*AR1++(IR0)
        .end
```

```
        .global COS_TAB
        .data
COS_TAB .float   0.707106781188 ; ALPHA
        .float   0.923879532511 ; BETA
        .float  -0.382683432365 ; -DELTA
        .float  -0.195090322016 ; -NU
        .float  -0.831469612303 ; -GAMMA
        .float   0.980785280403 ; LAMBDA
        .float   0.555570233019 ; MU
        .end
```