# PRODUCT DESCRIPTION
# μPD 7281

# IMAGE PIPELINED PROCESSOR APPLICATION LIBRARY

NEC ELECTRONICS (EUROPE) GMBH

# PRODUCT DESCRIPTION

# µPD7281

# IMAGE PIPELINED PROCESSOR

# APPLICATION LIBRARY

5/85 V1.0

VOLUME I

# BINARY IMAGE PROCESSING

Table of Contents

# Introduction

The uPD7281 incorporates a configuration which is quite different from those of other microcomputers, and most readers may be unfamiliar with its program development facilities. The uPD7281 Application Library is a collection of programs that provide specific examples of programming on the uPD7281. The document is intended as a guide to learning uPD7281 programming methods and techniques.

This Application Library (Volume I) addresses "Binary Image Processing," which is one of the application areas of the uPD7281. The applications presented in the following pages include:

* Block Transfer
  - Word Boundary Transfer
* Logical Operations
  - NOT (single-operand operations)
  - AND/OR/Exclusive OR (double-operand operations)
* Enlargement/Shrinking
  - Simple 1/2 Shrinking
  - 4-Point OR 1/2 Shrinking (Logical Addition)
  - Neighboring 16-Point Addition 1/4 Shrinking (the Majority Rule)
  - Simple Double Enlargement
  - Simple Quadruple Enlargement
* Affine Transformation
* Profiling
  - Horizontal
  - Vertical
* 3 x 3 Masking
  - Smoothing
  - Thinning
  - Edge Detection

Each example is discussed in terms of the following items:

(1) Processing
(2) Algorithm
(3) Parameters and their applicable ranges
(4) Flow graphs
(5) Tips on preparing flow graphs
(6) Assembler source listing

This Application Library explains uPD7281 programming methods. The reader is advised to consult the following publications for information on the uPD7281 itself and the assembler:

(1) uPD7281 User's Manual
(2) uPD7281 Software Package Operating Manual

In addition, the following document may be helpful:

(3) uPD9305 (uPD7281 Peripheral LSI) User's Manual

Note:
This document is intended solely as an explanation of uPD7281 programming methods. The programing examples provided have not been tested on an actual system, and no claim is made for their validity.

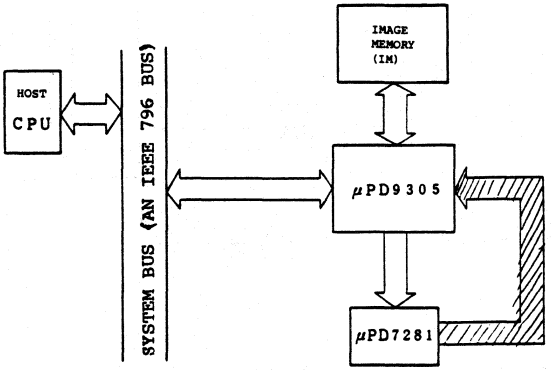
# Chapter 1

## System Configuration

The programs included in this Application Library have been written with the assumption that they will be executed on a system configured as shown in Figure 1-1. A characteristic of this system is the use of a uPD7281 peripheral LSI, the uPD9305. This means that a uPD9305-defined token is used in accessing the image memory (IM). The programs presented in this document are not directly applicable to those systems in which a uPD9305 is not employed.

Although the programs shown in this document are written for a single uPD7281, the uPD9305 permits the use of several uPD7281s. Therefore, it is possible to increase the processing speed by partitioning the memory area among several uPD7281s, each running the same program. In such a case, the input token into the uPD7281s should be set up in accordance with the memory area to which it is assigned.

Figure 1-1
System Configuration

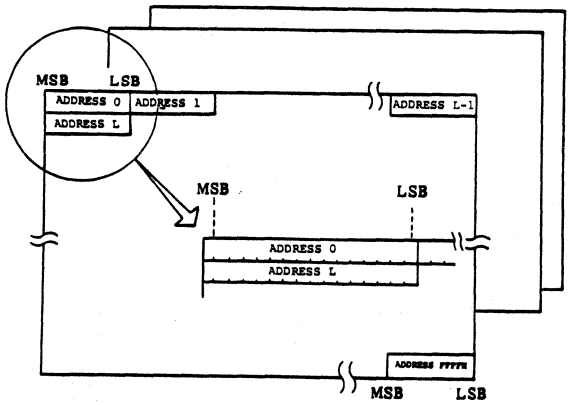


The following is a description of image memory configuration.

Shown in Figure 1-2 is an image memory configuration of the system addressed in this document. Viewing the memory in terms of screen display, the upper left corner of the screen corresponds to the most significant bit (MSB); this has the address of 0. The lower right corner the least significant bit (LSB); this has the address FFFFH*. Since there are 16 bits in a word, if there are L words in a horizontal line, the number of picture elements (pixels) present in that line is 16 x L dots.

In this system the addresses 0 through FFFFH comprise a screen image (one bank). However, since the 8 high order bits of the 24 image memory address bits of the uPD9305 are not set (i.e., neither the bank-defining read high address or write high address is set), it is necessary to set up a high address register in the uPD9305 to use multiple banks. For the same reason, the addressable memory space of these programs is addresses 0 through FFFFH, both for source and destination images.

Figure 1-2
Image Memory Configuration




* : This image memory configuration is different from that of the graphic display controllers uPD7220 and uPD7220A offered by NEC.

# Chapter 2

# Block Transfer

In this chapter we consider a program for transferring a specified rectangular area to another one. Although such a transfer can be accomplished with the same results by using a special case of the affine transformation (horizontal move), the method described below employs a different algorithm to reduce processing time. A discussion of word boundary transfer follows.

## 2.1 Word Boundary Transfer

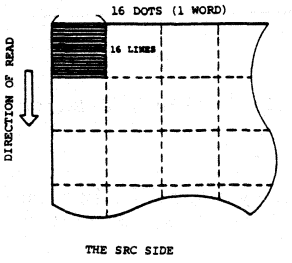
### 2.1.1 Processing Explained

A word boundary transfer is a block-oriented transfer in which a rectangular area defined by 16 horizontal dots (comprising a word specified by the image memory address) and 16 vertical dots are transferred as a block of data. Although the minimum size of the transfer block (16 x 16 dots) cannot be varied, the program permits variations in horizontal screen size L (number of words), and the number of horizontal blocks H and vertical blocks V to be transferred.

### 2.1.2 Algorithm

(1) First, the program and the constants are downloaded to the uPD7281. This is followed by the input from the host computer of two start-up tokens: the starting address (STARTS) of a source image area (SRC), and the starting address (STARTD) of the destination image area (DST).

Figure



(2) One block of data (SRC data) is read vertically, starting from the address indicated by STARTS.

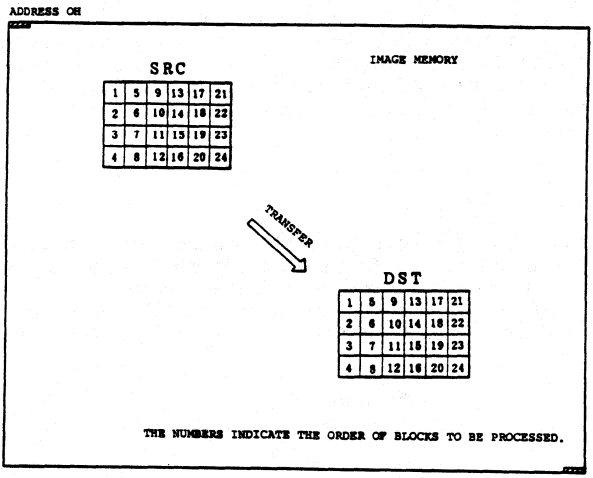
(3) Simultaneously, one block of memory addresses needed for the storage of SRC data is generated, starting at the address indicated by STARTD.

(4) The SRC data is then written into the DST of the image memory in accordance with the corresponding DST addresses.

(5) When all of V blocks have been transferred by repeating this operation vertically, the process is repeated in the horizontal direction by incrementing the horizontal address by 1.

(6) The transfer operation is terminated when step (5) is completed for all H blocks in the horizontal direction.

Figure 2-1
An Example of Data Transfer (for H=6, V=4)




### 2.1.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area words in horizontal direction

V ... Number of source image area blocks in vertical direction

<Start-up token-defined parameters>

STARTS: Starting address of the source image area
STARTD: Starting address of the destination image area

The allowable values of these parameters are indicated in the table below:

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535 | (64) |
| H | 1 - 256 | (32) |
| V | 1 - 256 | (32) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (32)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

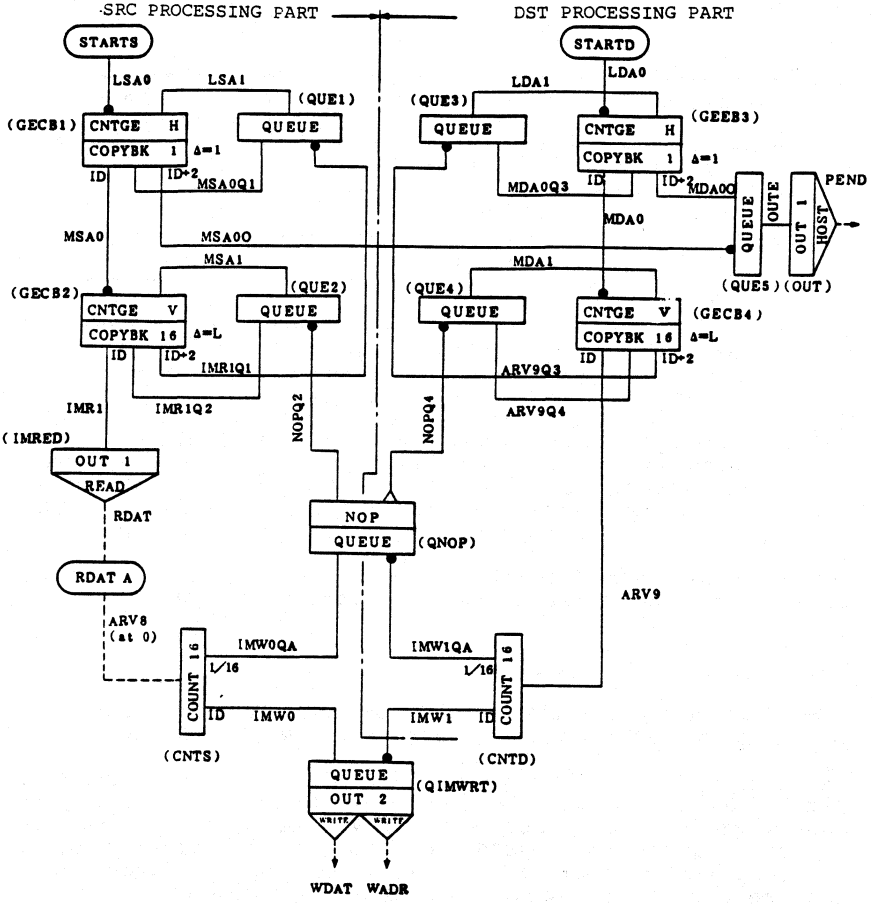
Furthermore, H and V should be defined with care, since this program does not accommodate the switching of banks.

### 2.1.4 Flow Graph Explained

A word boundary transfer is divided into the SRC processing part and the DST processing part, as shown in Figure 2-2.

In Figure 2-2, GECB2 generates 16 (1 block) vertical direction addresses for each of the V blocks. Upon completion of the processing of V blocks of data in the vertical direction, GECB1 increments the address by 1 (corresponding to one block in the horizontal direction), and executes GECB2 again. The GECB2 thus executed repeats the vertical direction processing in the same manner. This combination of vertical and horizontal processing creates a set of addresses comprising a rectangular area of H x V blocks.

Figure 2-2
Word Boundary Transfer Flow Graph



The image memory data (SRC data) indicated by these addresses are read on the SRC side and written to the output address generated on the DST side. Futher, since data are read on SRC and written to DST, SRC and DST are made to synchronize their actions so that the level of QIMWRT will not exceed 16.

<Explanation of Nodes>

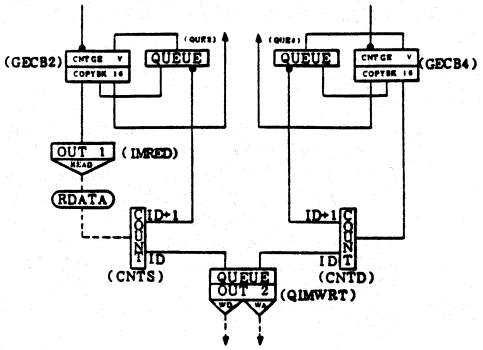
| | |
|---|---|
| GECB1/GECB3 | : Generate the starting address of V vertical blocks of the SRC/DST. |
| GECB2/GECB4 | : Generate the addresses within the V vertical blocks of the SRC/DST. |
| IMRED | : Reads SRC data. |
| QIMWRT | : Writes data to DST. |
| OUT | : Indicates completion of data transfer to the host. |
| CNTS, CNTD, QNOP | : Synchronize processing between the SRC and DST. |
| QUE1/QUE3 | : Synchronize the actions of GECB1 and GECB3 so that addresses are generated for V blocks at a time. |
| QUE2/QUE4 | : Synchronize the actions of GEB2 and GEB4 so that addresses are generated for one block at a time. |
| QUE5 | : Synchronizes SRC and DST to verify completion of their processing tasks. |

### 2.1.5 Tips on Writing Flow Graphs

A program for word boundary data transfer is made up of an address generation part to read from the SRC side and an address generation part to write to the DST side. These parts are identical except for the portion that concerns the reading of SRC data.

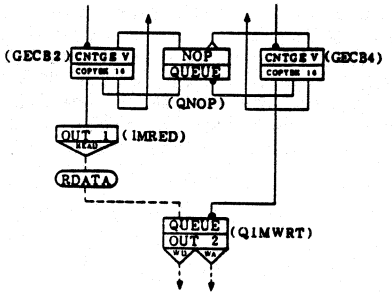
The SRC processing part generates addresses and reads SRC data from the image memory, whereas the DST processing part is involved only in the preparation of addresses. For this reason, the SRC part works slower than the DST part does. If a synchronizing node (QNOP) was not provided, as in the case of Figure 2-3(a), the DST addresses would be created one after another asynchronously with the SRC address generation and tokens sent by DST would cause an overflow in the QIMWRT node. To prevent this overflow, a node, QNOP, in Figure 2-2, is required to synchronize the SRC and DST parts.

Figure 2-3 (a)
Overflow Caused by Difference in Processing Speeds between SRC and DST




To synchronize these two processing parts, the scheme indicated in Figure 2-3 (b) might suffice and would also increase the processing speed. However, there are additional considerations: the amount of time required by the token issued to read the image memory and to make a return trip to the uPD7281, and situations where there may be some systems where the overflow occurs depending on the number of uPD7281s involved.

Figure 2-3 (b)
A Flow Graph in Which an Overflow May Occur Depending on the Time From When the Token is Issued to Read the Image Memory Until the Token Returns to the uPD7281



## 2.1.6 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;       W O R D   B O U N D A R Y
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  H       =       512/16          ;
10: EQUATE  V       =       512/16          ;
11: EQUATE  L       =       1024/16         ;
12: ;
13: EQUATE  HOST    =       0               ;
14: EQUATE  READ    =       4               ;
15: EQUATE  WRITE   =       5               ;
16: ;
17: EQUATE  STARTS  =       0               ;
18: EQUATE  STARTD  =       32              ;
19: ;
20: ;****************************************
21: ;
22: ;       INPUT-OUTPUT
23: ;
24: ;----------------------------------------
25: ;
26: INPUT   LSA0,   LDA0,   ARV8 AT 0       ;
27: ;
28: OUTPUT  RDAT,   WDAT,   WADR,   PEND    ;
29: ;
30: ;****************************************
31: ;
32: ;       LINK TABLE
33: ;
34: ;----------------------------------------
35: ;
36: LINK    MSA0,   MSA0Q1, MSA0O   =       GECB1   (LSA1,  LSA0    )       ;
37: LINK    LSA1                    =       QUE1    (MSA0Q1,IMR1Q1  )       ;
38: LINK    IMR1,   IMR1Q2, IMR1Q1  =       GECB2   (MSA1,  MSA0    )       ;
39: LINK    MSA1                    =       QUE2    (IMR1Q2,NOPQ2   )       ;
40: LINK    RDAT                    =       IMRED   (IMR1           )       ;
41: LINK    IMW0,   IMW0QA          =       CNTS    (ARV8           )       ;
42: LINK    MDA0,   MDA0Q3, MDA0O   =       GECB3   (LDA1,  LDA0    )       ;
43: LINK    LDA1                    =       QUE3    (MDA0Q3,ARV9Q3  )       ;
44: LINK    ARV9,   ARV9Q4, ARV9Q3  =       GECB4   (MDA1,  MDA0    )       ;
45: LINK    MDA1                    =       QUE4    (ARV9Q4,NOPQ4   )       ;
46: LINK    IMW1,   IMW1QA          =       CNTD    (ARV9           )       ;
47: LINK    NOPQ2,  NOPQ4           =       QNOP    (IMW0QA,IMW1QA  )       ;
48: LINK    WDAT,   WADR            =       QIMWRT  (IMW0,  IMW1    )       ;
49: LINK    OUTE                    =       QUE5    (MDA0O, MSA0O   )       ;
50: LINK    PEND                    =       OUT     (OUTE           )       ;
51: ;
```

```
52: ;****************************************
53: ;
54: ;       FUNCTION TABLE
55: ;
56: ;----------------------------------------
57: ;
58: FUNCTION   IMRED   =   OUT1    (READ.  0)                                  ;
59: FUNCTION   QIMWRT  =   OUT2    (WRITE. 20H. 0).QUEUE   (Q6.    16)       ;
60: FUNCTION   OUT     =   OUT1    (HOST.  0)                                  ;
61: FUNCTION   GECB1   =   COPYBK  (1.     1),     CNTGE   (H      )         ;
62: FUNCTION   GECB2   =   COPYBK  (16.    L),     CNTGE   (V      )         ;
63: FUNCTION   GECB3   =   COPYBK  (1.     1),     CNTGE   (H      )         ;
64: FUNCTION   GECB4   =   COPYBK  (16.    L),     CNTGE   (V      )         ;
65: FUNCTION   QUE1    =   QUEUE   (Q1.    1)                                ;
66: FUNCTION   QUE2    =   QUEUE   (Q2.    1)                                ;
67: FUNCTION   QUE3    =   QUEUE   (Q3.    1)                                ;
68: FUNCTION   QUE4    =   QUEUE   (Q4.    1)                                ;
69: FUNCTION   QUE5    =   QUEUE   (Q5.    1)                                ;
70: FUNCTION   CNTS    =   COUNT   (16     )                                 ;
71: FUNCTION   CNTD    =   COUNT   (16     )                                 ;
72: FUNCTION   QNOP    =   NOP     (XY     ),      QUEUE   (QA.    1)        ;
73: ;
74: ;****************************************
75: ;
76: ;       DATA MEMORY
77: ;
78: ;----------------------------------------
79: ;
80: MEMORY  Q1      =       AREA    (1      )       ;
81: MEMORY  Q2      =       AREA    (1      )       ;
82: MEMORY  Q3      =       AREA    (1      )       ;
83: MEMORY  Q4      =       AREA    (1      )       ;
84: MEMORY  Q5      =       AREA    (1      )       ;
85: MEMORY  Q6      =       AREA    (16     )       ;
86: MEMORY  QA      =       AREA    (1      )       ;
87: ;
88: ;****************************************
89: ;
90: ;       START
91: ;
92: ;----------------------------------------
93: ;
94: START                                           ;
95: ;
96: DATA    EXEC    (IPP.   LSA0.   STARTS  )       ;
97: DATA    EXEC    (IPP.   LDA0.   STARTD  )       ;
98: ;
99: END                                             ;
```

# Chapter 3

# Logical Operations

## 3.1 NOT (Single-Operand Operation)

### 3.1.1 Processing Explained

The NOT operation is used in writing data, read from the image memory source area (SRC), to the destination area (DST) by performing bit inversions.

### 3.1.2 Algorithm

First the SRC starting address (STARTS) and the DST starting address (STARTD) are received from the host computer as input parameters, as occurs in a word boundary transfer. Then, based on STARTS, data is read from SRC, block by block, in the vertical direction. The data is then NOT-operated, and the inverted data is written to the DST after being combined with its corresponding DST addresses (as in the case of a word boundary transfer).

Details are provided in Section 2.1, "Word Boundary Transfer."

### 3.1.3 Parameters and Their Applicable Ranges

<Assembler-Coded Parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area words in horizontal direction
V ... Number of destination image area blocks in vertical direction

<Start-up token-defined parameters>

STARTS ... Source image area (SRC) starting address
STARTD ... Destination image area (DST) starting address

The values that can be assigned to these parameters are as follows:

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535 | (64) |
| H | 1 - 256 | (32) |
| V | 1 - 256 | (32) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (32)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since this program has no provision for bank switching, care should be exercised in setting the values for the parameters H and L.

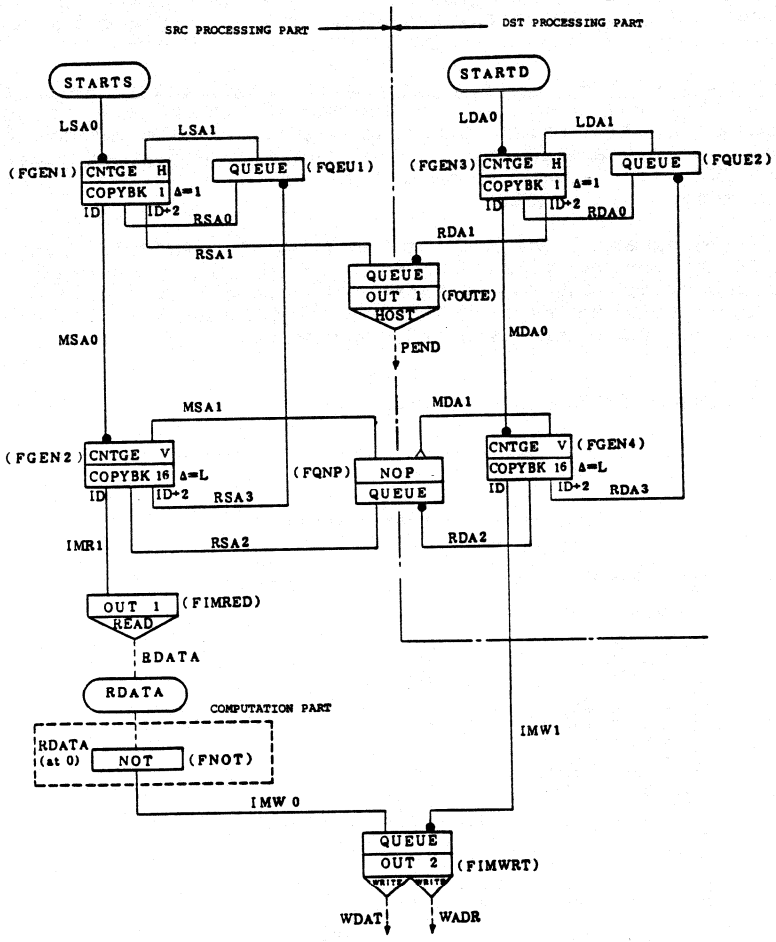
<Initial Values>

Initial values used in this program are the same as those used in the word boundary transfer program. (See 2.1.3).

### 3.1.4 Flow Graph Explained

The flow graph for the NOT operation, shown in Figure 3-1, represents the addition of SRC data NOT processing to the word boundary transfer flow graph given in Figure 2-1, except that the node for the COUNT instruction to ensure synchronization during a word boundary transfer has been removed. The method employed in the NOT operation for synchronizing SRC and DST processing is not an ideal one because there is a risk of an overflow occuring, depending on the reading time of SRC data. However, for the typical system configuration used in this example and for those cases where image memory access time is short, the synchronization method illustrated in the flow graph should be adequate and should yield improvements in processing speed.

Figure 3.1
A NOT (Single-Operand Operation) Flow Graph



### 3.1.5 Tips on Writing Flow Graphs

The SRC and DST processing must be synchronized for logical operations and transfer. In the absence of synchronization, some programs can run into a situation where an excess of data for either SRC processing or DST processing is created, resulting in a uPD7281 QUEUE overflow. The methods used in ensuring synchronization can vary, depending on the particular objective pursued and the way the flow graph is written. Any method used, however, must meet the following conditions:

- the work of the SRC token (the first input token) needs to be controlled; i.e., if the second of two tokens is not input after the first token is input, it should not be allowed to process SRC side operations independent of another side (DST, for example) resulting in an overflow on the image memory write QUEUE.

- when an external token is issued from the uPD7281, such as the image memory access token, you should take care to preclude errors due to the amount of access time required in the external circuits or due to the maximum number of tokens flowing around in the external components.

### 3.1.6 Assembler Source Listing

```
 1: ;*************************************
 2: ;
 3: ;       NOT OPERATION
 4: ;
 5: ;-------------------------------------
 6: ;
 7: MODULE  IPP     =       8                       ;
 8: ;
 9: EQUATE  L       =       64                      ;
10: EQUATE  H       =       32                      ;
11: EQUATE  V       =       32                      ;
12: ;
13: EQUATE  HOST    =       0                       ;
14: EQUATE  READ    =       4                       ;
15: EQUATE  WRITE   =       5                       ;
16: ;
17: EQUATE  START3  =       0                       ;
18: EQUATE  STARTD  =       32                      ;
19: ;
20: ;*************************************
21: ;
22: ;       INPUT-OUTPUT
23: ;
24: ;-------------------------------------
25: INPUT   LSA0.   LDA0.   RDATA AT 0              ;
26: ;
27: OUTPUT  RDAT.   WDAT.   WADR.   PEND            ;
28: ;
```

```
29: ;*************************************
30: ;
31: ;       LINK TABLE
32: ;
33: ;-------------------------------------
34: ;
35: LINK    MSA0.   RSA0.   RSA1    =       FGEN1   (LSA1.   LSA0    )       ;
36: LINK    IMR1.   RSA2.   RSA3    =       FGEN2   (MSA1.   MSA0    )       ;
37: LINK    LSA1                    =       FQUE1   (RSA0.   RSA3    )       ;
38: LINK    PEND                    =       FOUTE   (RSA1.   RDA1    )       ;
39: LINK    RDAT                    =       FIMRED  (IMR1            )       ;
40: LINK    MSA1.   MDA1            =       FQNP    (RSA2.   RDA2    )       ;
41: LINK    IMW0                    =       FNOT    (RDATA           )       ;
42: LINK    MDA0.   RDA0.   RDA1    =       FGEN3   (LDA1.   LDA0    )       ;
43: LINK    IMW1.   RDA2.   RDA3    =       FGEN4   (MDA1.   MDA0    )       ;
44: LINK    LDA1                    =       FQUE2   (RDA0.   RDA3    )       ;
45: LINK    WDAT.   WADR            =       FIMWRT  (IMW0.   IMW1    )       ;
46: ;
47: ;*************************************
48: ;
49: ;       FUNCTION TABLE
50: ;
51: ;-------------------------------------
52: ;
53: FUNCTION    FIMRED  =   OUT1    (READ.  0)                                  ;
54: FUNCTION    FIMWRT  =   OUT2    (WRTIE. 20H. 0).QUEUE   (QUEW. 16)          ;
55: FUNCTION    FOUTE   =   OUT1    (HOST.  0).     QUEUE   (QUEE.  1)          ;
56: FUNCTION    FGEN1   =   COPYBK  (1.     1).     CNTGE   (H      )           ;
57: FUNCTION    FGEN2   =   COPYBK  (16.    L).     CNTGE   (V      )           ;
58: FUNCTION    FGEN3   =   COPYBK  (1.     1).     CNTGE   (H      )           ;
59: FUNCTION    FGEN4   =   COPYBK  (16.    L).     CNTGE   (V      )           ;
60: FUNCTION    FQNP    =   NOP     (XY     ).  _   QUEUE   (QUEN.  1)          ;
61: FUNCTION    FNOT    =   NOT     (X      )
62: FUNCTION    FQUE1   =   QUEUE   (QUE1.  1)
63: FUNCTION    FQUE2   =   QUEUE   (QUE2.  1)
64: ;
65: ;*************************************
66: ;
67: ;       DATA MEMORY
68: ;
69: ;-------------------------------------
70: ;
71: MEMORY  QUE1    =       AREA    (1      )       ;
72: MEMORY  QUE2    =       AREA    (1      )       ;
73: MEMORY  QUEN    =       AREA    (1      )       ;
74: MEMORY  QUEW    =       AREA    (16     )       ;
75: MEMORY  QUEE    =       AREA    (1      )       ;
76: ;
77: ;*************************************
78: ;
79: ;       START
80: ;
81: ;-------------------------------------
82: ;
83: START                                           ;
84: ;
85: DATA    EXEC    (IPP.   LSA0.   STARTS  )       ;
86: DATA    EXEC    (IPP.   LDA0.   STARTD  )       ;
87: ;
88: END                                             ;
```

## 3.2 AND, OR, Exclusive OR (Double-Operand Operations)

### 3.2.1 Processing Explained

Given SRC1 and SRC2 as source image areas, these operations perform the specified operation (AND, OR, or Exclusive OR) between the data in these areas and outputs the resultant data to the destination image area. As in the case of the word boundary data transfer (discussed in 2.1), each of the areas, SRC1, SRC2, and DST, in this program are divided into word-aligned horizontal 16-dot (1 word) and vertical 16-dot (16 lines) rectangular areas (blocks) as units of data transfer. For the order of processing, refer to Section 2.1, "Word Boundary Transfer."

### 3.2.2 Algorithm

The starting read addresses S1ADR and S2ADR of source image areas SRC1 and SRC2, respectively, and the starting address DSTADR of destination image area DST are input from the host computer into the uPD7281 as part of a token. Then, one block of addresses are generated for each S1ADR and S2ADR, and the contents of the image memory at those addresses are read. Data in the two blocks just read are operated upon (AND, OR, or Exclusive OR) in conjunction with their corresponding data, resulting in the creation of DST data. The DST data are written out to the DST addresses generated on the basis of DSTADR.

### 3.2.3 Parameters and Their Applicable Ranges

Assembler-coded parameters:
L ... Number of image memory words in horizontal direction
H ... Number of source image area words in horizontal direction
V ... Number of source image area blocks in vertical direction

Start-up token-defined parameters:
S1ADR ... First operand source image area (SRC1) starting address
S2ADR ... Second operand source image area (SRC2) starting address
DSTADR .. Destination image area (DST) starting address

The allowable values of these parameters are as follows:

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535 | (64) |
| H | 1 - 256 | (16) |
| V | 1 - 256 | (16) |
| S1ADR | 0 - 65535 | ( 0)* |
| S2ADR | 0 - 65535 | (4000H)* |
| DSTADR | 0 - 65535 | ( 0)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting values for parameters H and V.

<Initial Values>

Most of the initial values used in this program are the same as those used in the word boundary transfer program, except that initial values need to be assigned to the two SRC addresses.

### 3.2.4 Flow Graph Explained

This program illustrates the use of the OR operation. To use AND or Exclusive OR, change "OR" to "AND" or "XOR" (XOR is the instruction code for the Exclusive OR operation) in the FQOR node in Figure 3-2.

<Explanation of Nodes>

FGEN1/FGEN3 : Creates H blocks of horizontal starting addresses on the basis of the starting addresses S1ADR/S2ADR for SRC1/SRC2.

FGEN2/FGEN4 : Creates V blocks of in-block addresses on the basis of addresses sent from FGEN1/FGEN3.

FIMRD1/FIMRD2 : Reads the contents of SRC1/SRC2 on the basis of addresses created in FGEN2/FGEN4.

FQOR : Creates DST data by performing the OR

between the data read in FIMRD1 and FIMRD2.

FGEN5 : Creates H blocks of DST horizontal starting addresses on the basis of the DST starting address DSTADR

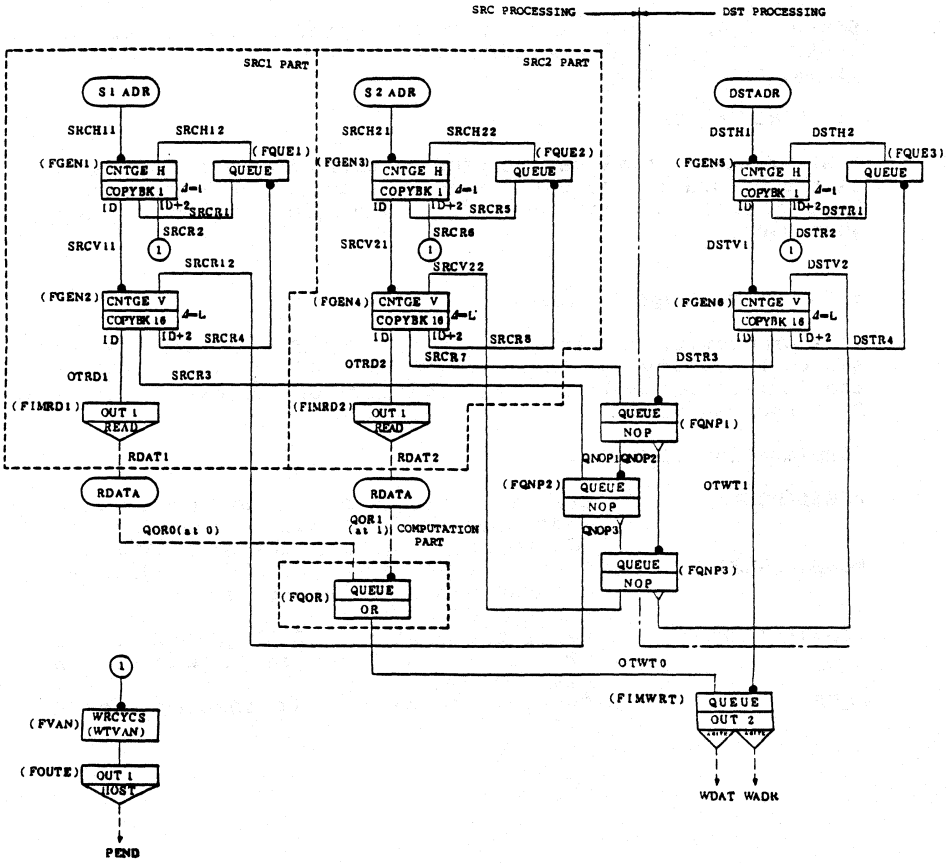
FGEN6 : Creates V blocks of in-block DST addresses on the basis of addresses generated in FGEN5.

FIMWRT : Writes DST data created in FQOR to the DST addresses generated in FGEN6.

FQUE1/FQUE2/FQUE3 : Synchronizes the actions of FGEN1/FGEN3/FGEN5 so that they make V blocks worth of addresses at a time.

FQNP1/FQNP2/FQNP3 : Synchronizes the actions of SRC and DST processing whenever one block of addresses are created in FGEN2, FGEN4, or FGEN6.

Figure 3-2
A Double-Operand Operation Flow Graph (OR)



### 3.2.5 Tips on Writing Flow Graphs

This program is basically the same as the word boundary transfer program in 2.1, except for minor differences that exist in the synchronization method because of the need to access three areas.

### 3.2.6 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;       LOGICAL OPERATION
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  L       =       64              ;
10: EQUATE  H       =       16              ;
11: EQUATE  V       =       16              ;
12: ;
13: EQUATE  HOST    =       0               ;
14: EQUATE  READ    =       4               ;
15: EQUATE  WRITE   =       5               ;
16: ;
17: EQUATE  S1ADR   =       0               ;
18: EQUATE  S2ADR   =       256*L           ;
19: EQUATE  DSTADR  =       0               ;
20: ;
21: ;
22: ;****************************************
23: ;
24: ;       INPUT-OUTPUT
25: ;
26: ;----------------------------------------
27: ;
28: INPUT   SRCH11, SRCH21, DSTH1                    ;
29: INPUT   QOR0 AT 0                               ;
30: INPUT*  QOR1 AT 1                               ;
31: ;
32: OUTPUT  RDAT1,  RDAT2,  WDAT,   WADR,   PEND    ;
33: ;
34: ;****************************************
35: ;
36: ;       LINK TABLE
37: ;
38: ;----------------------------------------
39: ;
40: LINK    SRCV11, SRCR1,  SRCR2   =       FGEN1   (SRCH12,SRCH11  )       ;
41: LINK    OTRD1,  SRCR3,  SRCR4   =       FGEN2   (SRCV12,SRCV11  )       ;
42: LINK    SRCH12                  =       FQUE1   (SRCR1, SRCR4   )       ;
43: LINK    OUTH                    =       FVAN    (       ,SRCR2  )       ;
44: LINK    RDAT1                   =       FIMRD1  (OTRD1          )       ;
45: LINK    SRCV12, QNOP3           =       FQNP2   (SRCR3, QNOP1   )       ;
46: LINK    SRCV21, SRCR5,  SRCR6   =       FGEN3   (SRCH22,SRCH21  )       ;
47: LINK    OTRD2,  SRCR7,  SRCR8   =       FGEN4   (SRCV22,SRCV21  )       ;
48: LINK    SRCH22                  =       FQUE2   (SRCR5, SRCR8   )       ;
49: LINK    OUTH                    =       FVAN    (       ,SRCR6  )       ;
50: LINK    RDAT2                   =       FIMRD2  (OTRD2          )       ;
51: LINK    QNOP1,  QNOP2           =       FQNP1   (SRCR7, DSTR3   )       ;
52: LINK    OTWT0                   =       FQOR    (QOR0,  QOR1    )       ;
53: LINK    SRCV22, DSTV2           =       FQNP3   (QNOP3, QNOP2   )       ;
54: LINK    DSTV1,  DSTR1,  DSTR2   =       FGEN5   (DSTH2, DSTH1   )       ;
55: LINK    OTWT1,  DSTR3,  DSTR4   =       FGEN6   (DSTV2, DSTV1   )       ;
56: LINK    DSTH2                   =       FQUE3   (DSTR1, DSTR4   )       ;
57: LINK    OUTH                    =       FVAN    (       ,DSTR2  )       ;
58: LINK    WDAT,   WADR            =       FIMWRT  (OTWT0, OTWT1   )       ;
59: LINK    PEND                    =       FOUTE   (OUTH           )       ;
60: ;
```

```
  61: ;****************************************
  62: ;
  63: ;       FUNCTION TABLE
  64: ;
  65: ;----------------------------------------
  66: ;
  67: FUNCTION    FIMRD1  =   OUT1    (READ.    0)                                  ;
  68: FUNCTION    FIMRD2  =   OUT1    (READ.    1)                                  ;
  69: FUNCTION    FIMWRT  =   OUT2    (WRTIE. 20H. 0).QUEUE   (QUEW.   16)      ;
  70: FUNCTION    FOUTE   =   OUT1    (HOST.    0)                                  ;
  71: FUNCTION    FGEN1   =   COPYBK  (1.       1).   CNTGE   (H        )      ;
  72: FUNCTION    FGEN2   =   COPYBK  (16.      L).   CNTGE   (V        )      ;
  73: FUNCTION    FGEN3   =   COPYBK  (1.       1).   CNTGE   (H        )      ;
  74: FUNCTION    FGEN4   =   COPYBK  (16.      L).   CNTGE   (V        )      ;
  75: FUNCTION    FGEN5   =   COPYBK  (1.       1).   CNTGE   (H        )      ;
  76: FUNCTION    FGEN6   =   COPYBK  (16.      L).   CNTGE   (V        )      ;
  77: FUNCTION    FQOR    =   OR      (X         ).   QUEUE   (QUEOR. 16)      ;
  78: FUNCTION    FQNP1   =   NOP     (XY        ).   QUEUE   (QUEN1. 1)       ;
  79: FUNCTION    FQNP2   =   NOP     (XY        ).   QUEUE   (QUEN2. 1)       ;
  80: FUNCTION    FQNP3   =   NOP     (XY        ).   QUEUE   (QUEN3. 1)       ;
  81: FUNCTION    FQUE1   =   QUEUE   (QUE1.    1)                                  ;
  82: FUNCTION    FQUE2   =   QUEUE   (QUE2.    1)                                  ;
  83: FUNCTION    FQUE3   =   QUEUE   (QUE3.    1)                                  ;
  84: FUNCTION    FVAN    =   WRCYCS  (WTVAN.   3)                                  ;
  85: ;
  86: ;****************************************
  87: ;
  88: ;       DATA MEMORY
  89: ;
  90: ;----------------------------------------
  91: ;
  92: MEMORY  QUE1    =       AREA    (1      )       ;
  93: MEMORY  QUE2    =       AREA    (1      )       ;
  94: MEMORY  QUE3    =       AREA    (1      )       ;
  95: MEMORY  QUEN1   =       AREA    (1      )       ;
  96: MEMORY  QUEN2   =       AREA    (1      )       ;
  97: MEMORY  QUEN3   =       AREA    (1      )       ;
  98: MEMORY  QUEOR   =       AREA    (16     )       ;
  99: MEMORY  QUEW    =       AREA    (16     )       ;
 100: MEMORY  WTVAN   =       AREA    (3      )       ;
 101: ;
 102: ;****************************************
 103: ;
 104: ;       START
 105: ;
 106: ;----------------------------------------
 107: ;
 108: START                                           ;
 109: ;
 110: DATA    EXEC    (IPP.   SRCH11. S1ADR   )       ;
 111: DATA    EXEC    (IPP.   SRCH21. S2ADR   )       ;
 112: DATA    EXEC    (IPP.   DSTH1.  DSTADR  )       ;
 113: ;
 114: END                                             ;
```

# Chapter 4

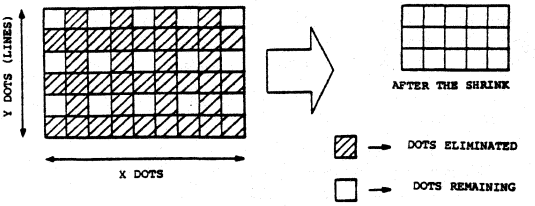
# Enlargement and Shrinking

## 4.1 Simple One-Half Shrinking

### 4.1.1 Processing Explained

Simple one-half shrinking means a shrinking by simple elimination of source image area (SRC) data, as shown in the figure below.

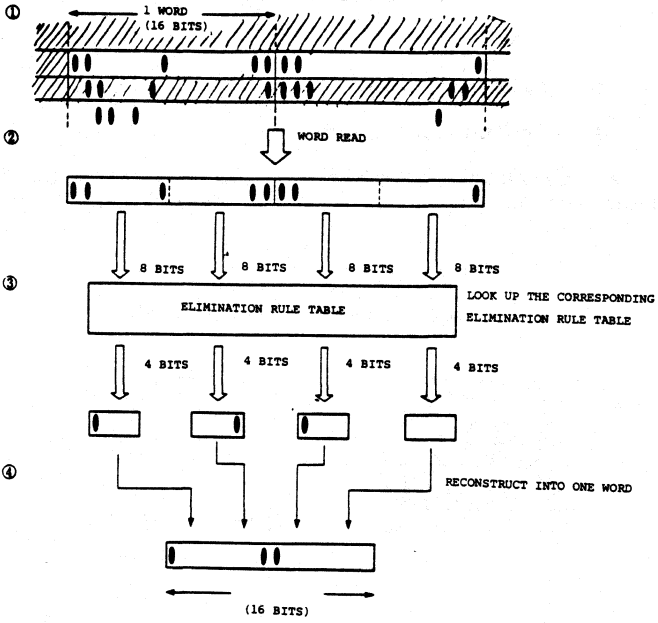
Figure 4-1
Simple One-Half Shrinking




### 4.1.2 Algorithm

Since all that is involved is a simple elimination, data are read vertically by skipping every other line. Horizontally, 16-bit data that are read are subjected to elimination by looking up an elimination rule table. To avoid the problem of handling a large 16-bit table, the data are divided into high-order and low-order 8-bit segments. A 256-word (8-bit address) elimination rule table is provided for each of these segments. The use of the 256-word table results in 4-bit data. Four pieces of such 4-bit data are written to the destination image area (DST) as one word (See Figure 4-2).

Figure 4-2
Algorithm for Simple One-Half Shrinking




### 4.1.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of destination image area (DST) words in horizontal direction
M ... Number of destination image area (DST) lines in vertical direction

<Start-up token-defined parameters>

STARTS ... Source image area (SRC) starting address
STARTD ... Destination image area (DST) starting address

The allowable values of these parameters are indicated in the table below.

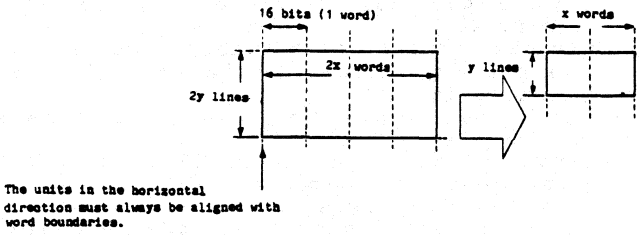
| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 32767 | (64) |
| H | 1 - 16 | (16) |
| M | 1 - 256 | (256) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (32)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting the values of parameters H and M. The maximum allowable values of these parameters in this program are H=16 and M=256; in other words, the maximum size of a destination image that can be reduced is 16 words horizontally and 256 lines vertically.

<Initial Values>

In setting initial values, the horizontal size of a SRC to be shrunk is defined in an even number of words, and the vertical size is defined in an even number of lines. Since the process involved is a simple one-half shrinking, once the size on the SRC side is determined, the size of the reduced image on the DST side is determined automatically.



Notice that, because DSTs are written word by word, the horizontal and vertical SRC definition numbers must always be multiples of 2.

### 4.1.4 Flow Graph Explained

This program consists of a SRC processing part (which reads every other line of the SRC data and creates shrunken data), and a DST part (which creates addresses necessary for writing the shrunk data). Shrunk data is generated on the basis of words of data at two contiguous addresses on the SRC side (the first word corresponds with the high-order 8 bits, and the second word with the low-order 8 bits). The SRC part receives from the host computer an SRC address which indicates the starting address of the shrinking processing, reads the SRC data horizontally every other line starting with the given SRC address, and separates them into high-order 8-bit and low-order 8-bit segments.

Each of the high-order 8-bit and low-order 8-bit segments is used as an address for a look-up table to generate high-order 4-bit and low-order 4-bit data generations. Four pieces of the 4-bit data thus created by two SRC data are placed in their respective bit positions to constitute DST (shrunk) data. That is, one DST data is created by two SRC data.

The DST processing part receives the starting address for writing the shrunk data from the host computer and creates write addresses on the basis of this address.

<Explanation of Nodes>

| | |
|---|---|
| FNOP0 | : Synchronizes the actions of start-up tokens received from the host computer and indicating SRC (STARTS) and DST (STARTD) addresses, and sends them to the SRC and DST processing parts. |
| FGEN1 | : Creates line starting addresses for every other line in order to read SRC data. (Creates vertical addresses for every other line on the basis of the SRC address sent from FNOP0.) |
| FGEN2 | : Creates one line of SRC data addresses in two passes, on the basis of addresses generated in FGEN1. |
| FOUTR | : Reads the contents of SRC addresses generated in FGEN2. |
| FARV1 | : Sends SRC data read in FOUTR alternately (high-order and low-order 8 bits) to the DST data generation node. |
| FCOP1/FCOP2 | : In the generation of high-order and low-order 8-bit data from the 16-bit data separated by FARV1, makes two copies of the data so that the high-order 4 bits of DST |

data can be created from the high-order 8 bits of 16-bit SRC data, and the low-order 4 bits of DST data from the low-order 8 bits of the SRC data.

FRED1/FRED3 : Performs a right 8-bit shift to enable FTRN1/FTRN3 to look up the elimination rule data table on the basis of the high-order 8 bits of 16-bit SRC data.

FTRN1/FTRN3 : Looks up the elimination rule data table on the basis of the high-order 8 bits of 16-bit SRC data, and creates high-order 4-bit data for the high-order and low-order 8-bit DST data.

FTRN2/FTRN4 : Looks up the elimination rule table on the basis of the low-order 8 bits of a second copy of SRC data made in FCOP1/FCOP2, creates 4-bit elimination data for the low-order 8 bits, and assigns them as low-order 4-bit portions of the high-order and low-order 8-bit DST data. (4-bit data created in FTRN4 becomes the low-order 4-bit portion of the low-order 8-bit portion of the DST data.)

FRED12/FRED08 : Performs a 12/8 left shift on the 4-bit data created in FTRN1/FTRN2 and makes them into the high-order and low-order 4 bits of the high-order 8-bit DST data.

FRED04 : Performs a 4-bit left shift on the 4-bit data created in FTRN3 and makes it into high-order 4 bits of the low-order 8 bits of DST data.

FADD1/FADD2 : Adds the data created in FRED12/FRED04 and FRED08/FTRN4 together and creates high-order and low-order 8 bits of DST data.

FADD3 : Adds the data created in FADD1 and FADD2 together and makes them into DST data.

FARV2 : To enable FGEN2 to create SRC readout addresses each time 8 DST data are generated, notifies FQUE1 that the eighth data has been copied and that 8 DST data have been generated.

FGEN3 : Creates M lines of addresses for H horizontal words starting from the DST starting address STARTD sent from FNOP0.

FOUTW : Writes the reduced data generated by the SRC processing into the DST address created in FGEN3.

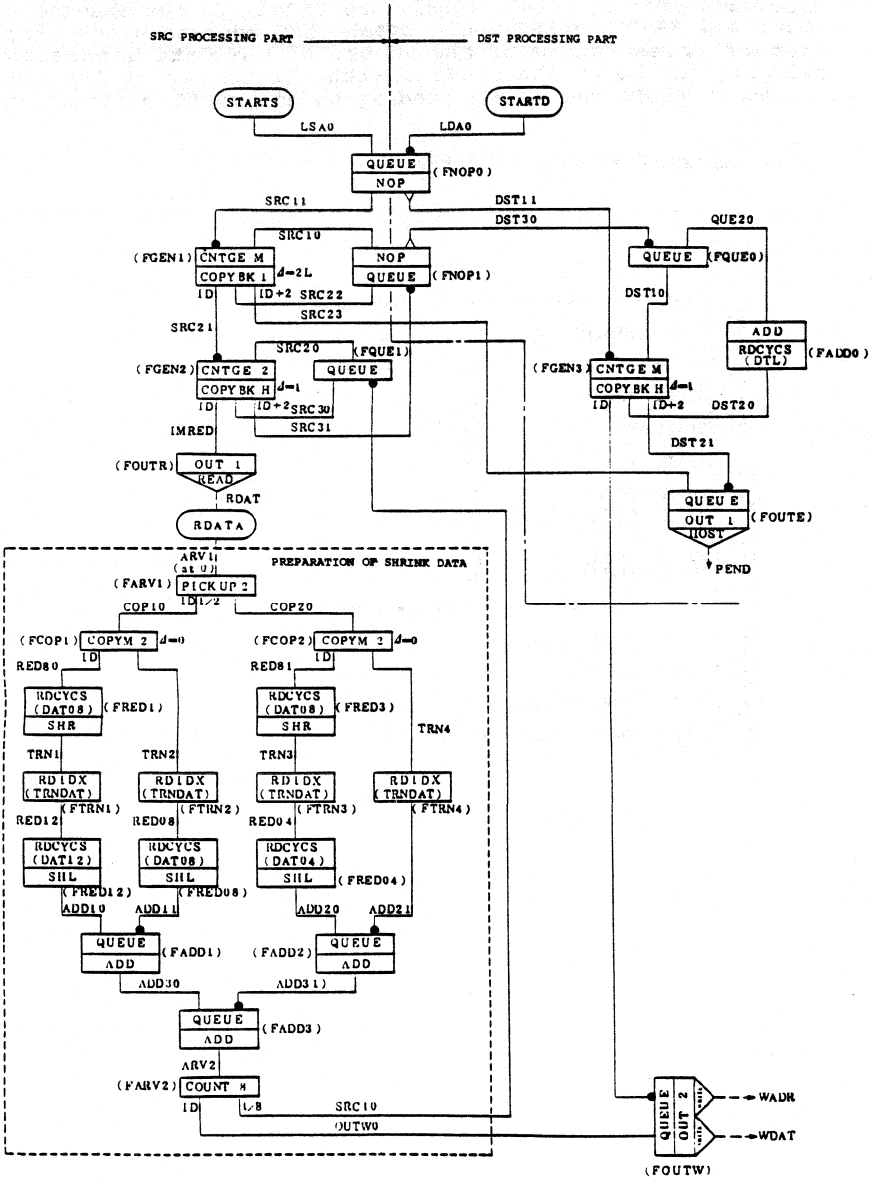
FADD0 : Upon completion of the generation of H words of horizontal addresses by FGEN3, modifies the addresses sent by FGEN3 in order to create addresses for the next line.

FNOP1 : Synchronizes the actions of FGEN1 and FGEN2 so that FGEN1 can create the starting address of a line each time FGEN2 generates addresses for one line of SRC data.

FQUE1 : Synchronizes the actions of FGEN2 and FARV2 so that SRC addresses will be created each time 8 DST data are generated.

FQUE2 : Synchronizes the actions of the SRC and DST processing parts so that the DST processing can create one line of addresses each time the SRC processing part generates one line of DST data.

Figure 4-3
A Flow Graph for Simple One-Half Shrinking



### 4.1.5 Tips on Writing Flow Graphs

In this program 16 SRC addresses are created for each 8 DST data generated by FGEN2, FQUE1, and FARV2. In the absence of FQUE1 and FARV2, FGEN2 would create SRC addresses one after another irrespective of the number of DST data generated, filling the GQ of the uPD7281 with the tokens of SRC data copied in FROP1 and FCOP2, leading to an overflow situation.

### 4.1.6 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;       SHRINK 1/2
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  L       =       64              ;
10: EQUATE  M       =       256             ;
11: EQUATE  H       =       16              ;
12: ;
13: EQUATE  HOST    =       0               ;
14: EQUATE  READ    =       4               ;
15: EQUATE  WRITE   =       5               ;
16: ;
17: EQUATE  STARTS  =       0               ;
18: EQUATE  STARTD  =       32              ;
19: ;
20: ;****************************************
21: ;
22: ;       INPUT-OUTPUT
23: ;
24: ;----------------------------------------
25: ;
26: INPUT   LSA0,   LDA0,   ARV1 AT 0       ;
27: ;
28: OUTPUT  RDAT,   WDAT,   WADR,   PEND    ;
29: ;
```

```
30: ;****************************************
31: ;
32: ;       LINK TABLE
33: ;
34: ;----------------------------------------
35: ;
36: LINK    SRC11,  DST11           =       FNOP0   (LSA0,  LDA0    )       ;
37: LINK    SRC21,  SRC22,  SRC23   =       FGEN1   (SRC10, SRC11   )       ;
38: LINK    OUTW1,  DST20,  DST21   =       FGEN3   (DST10, DST11   )       ;
39: LINK    IMRED,  SRC30,  SRC31   =       FGEN2   (SRC20, SRC21   )       ;
40: LINK    SRC10,  DST30           =       FNOP1   (SRC22, SRC31   )       ;
41: LINK    PEND                    =       FOUTE   (SRC23, DST21   )       ;
42: LINK    RDAT                    =       FOUTR   (IMRED          )       ;
43: LINK    SRC20                   =       FQUE1   (SRC30, SRC40   )       ;
44: LINK    COP10,  COP20           =       FARV1   (ARV1           )       ;
45: LINK    RED80,  TRN2            =       FCOP1   (COP10          )       ;
46: LINK    TRN1                    =       FRED1   (RED80          )       ;
47: LINK    RED12                   =       FTRN1   (TRN1           )       ;
48: LINK    ADD10                   =       FRED12  (RED12          )       ;
49: LINK    RED08                   =       FTRN2   (TRN2           )       ;
50: LINK    ADD11                   =       FRED08  (RED08          )       ;
51: LINK    ADD30                   =       FADD1   (ADD10, ADD11   )       ;
52: LINK    RED81,  TRN4            =       FCOP2   (COP20          )       ;
53: LINK    TRN3                    =       FRED3   (RED81          )       ;
54: LINK    RED04                   =       FTRN3   (TRN3           )       ;
55: LINK    ADD20                   =       FRED04  (RED04          )       ;
56: LINK    ADD21                   =       FTRN4   (TRN4           )       ;
57: LINK    ADD31                   =       FADD2   (ADD20, ADD21   )       ;
58: LINK    ARV2                    =       FADD3   (ADD30, ADD31   )       ;
59: LINK    OUTW0,  SRC40           =       FARV2   (ARV2           )       ;
60: LINK    WDAT,   WADR            =       FOUTW   (OUTW0, OUTW1   )       ;
61: LINK    QUE20                   =       FADD0   (DST20          )       ;
62: LINK    DST10                   =       FQUE0   (QUE20, DST30   )       ;
63: ;
64: ;****************************************
65: ;
66: ;       FUNCTION TABLE
67: ;
68: ;----------------------------------------
69: ;
70: FUNCTION    FOUTR   =   OUT1    (READ,  0)                              ;
71: FUNCTION    FOUTW   =   OUT2    (WRTIE, 20H, 0),QUEUE   (QUEW, 16)      ;
72: FUNCTION    FOUTE   =   OUT1    (HOST,  0),     QUEUE   (QUEE,  1)      ;
73: FUNCTION    FGEN1   =   COPYBK  (1,   2*L),     CNTGE   (M      )       ;
74: FUNCTION    FGEN2   =   COPYBK  (H,     1),     CNTGE   (2      )       ;
75: FUNCTION    FGEN3   =   COPYBK  (H,     1),     CNTGE   (M      )       ;
76: FUNCTION    FARV1   =   PICKUP  (2      )                               ;
77: FUNCTION    FARV2   =   COUNT   (8      )                               ;
78: FUNCTION    FCOP1   =   COPYM   (2,     0)                              ;
79: FUNCTION    FCOP2   =   COPYM   (2,     0)                              ;
80: FUNCTION    FRED1   =   SHR     (X      ),      RDCYCS  (DAT08, 1)      ;
81: FUNCTION    FRED3   =   SHR     (X      ),      RDCYCS  (DAT08, 1)      ;
82: FUNCTION    FRED12  =   SHL     (X      ),      RDCYCS  (DAT12, 1)      ;
83: FUNCTION    FRED08  =   SHL     (X      ),      RDCYCS  (DAT08, 1)      ;
84: FUNCTION    FRED04  =   SHL     (X      ),      RDCYCS  (DAT04, 1)      ;
85: FUNCTION    FTRN1   =   RDIDX   (TRNDAT )                               ;
86: FUNCTION    FTRN2   =   RDIDX   (TRNDAT )                               ;
87: FUNCTION    FTRN3   =   RDIDX   (TRNDAT )                               ;
88: FUNCTION    FTRN4   =   RDIDX   (TRNDAT )                               ;
89: FUNCTION    FADD0   =   ADD     (X      ),      RDCYCS  (DTL,   1)      ;
90: FUNCTION    FADD1   =   ADD     (X      ),      QUEUE   (QUE4,  8)      ;
91: FUNCTION    FADD2   =   ADD     (X      ),      QUEUE   (QUE5,  8)      ;
92: FUNCTION    FADD3   =   ADD     (X      ),      QUEUE   (QUE6,  8)      ;
93: FUNCTION    FNOP0   =   NOP     (XY     ),      QUEUE   (QUE2,  1)      ;
94: FUNCTION    FNOP1   =   NOP     (XY     ),      QUEUE   (QUE3,  1)      ;
95: FUNCTION    FQUE0   =   QUEUE   (QUE0,  1)                              ;
96: FUNCTION    FQUE1   =   QUEUE   (QUE1,  1)                              ;
97: ;
```

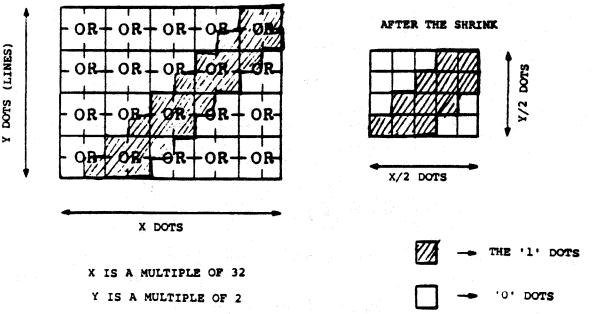
```
 98: ;*************************************
 99: ;
100: ;       DATA MEMORY
101: ;
102: ;-------------------------------------
103: ;
104: MEMORY  QUE0    =       AREA    (1      )       ;
105: MEMORY  QUE1    =       AREA    (1      )       ;
106: MEMORY  QUE2    =       AREA    (1      )       ;
107: MEMORY  QUE3    =       AREA    (1      )       ;
108: MEMORY  QUE4    =       AREA    (8      )       ;
109: MEMORY  QUE5    =       AREA    (8      )       ;
110: MEMORY  QUE6    =       AREA    (8      )       ;
111: MEMORY  QUEW    =       AREA    (16     )       ;
112: MEMORY  QUEE    =       AREA    (1      )       ;
113: MEMORY  DAT04   =       4                       ;
114: MEMORY  DAT08   =       8                       ;
115: MEMORY  DAT12   =       12                      ;
116: MEMORY  DTL     =       L-16                    ;
117: MEMORY  TRNDAT  =        0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
118:                          4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
119:                          0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
120:                          4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
121:                          8, 9, 8, 9,10,11,10,11, 8, 9, 8, 9,10,11,10,11,
122:                         12,13,12,13,14,15,14,15,12,13,12,13,14,15,14,15,
123:                          8, 9, 8, 9,10,11,10,11, 8, 9, 8, 9,10,11,10,11,
124:                         12,13,12,13,14,15,14,15,12,13,12,13,14,15,14,15,
125:                          0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
126:                          4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
127:                          0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3,
128:                          4, 5, 4, 5, 6, 7, 6, 7, 4, 5, 4, 5, 6, 7, 6, 7,
129:                          8, 9, 8, 9,10,11,10,11, 8, 9, 8, 9,10,11,10,11,
130:                         12,13,12,13,14,15,14,15,12,13,12,13,14,15,14,15,
131:                          8, 9, 8, 9,10,11,10,11, 8, 9, 8, 9,10,11,10,11,
132:                         12,13,12,13,14,15,14,15,12,13,12,13,14,15,14,15 ;
133: ;
134: ;*************************************
135: ;
136: ;       START
137: ;
138: ;-------------------------------------
139: ;
140: START                                           ;
141: ;
142: DATA    EXEC    (IPP,   LSA0,   STARTS  )       ;
143: DATA    EXEC    (IPP,   LDA0,   STARTD  )       ;
144: ;
145: END                                             ;
```

## 4.2 Four-Point OR One-Half Shrinking

### 4.2.1 Processing Explained

In the four-point OR one-half shrinking method, the OR operation is performed on four adjacent points in a given source image area (SRC), and the resulting single dot is made into destination image (DST) data, as shown in Figure 4-4.

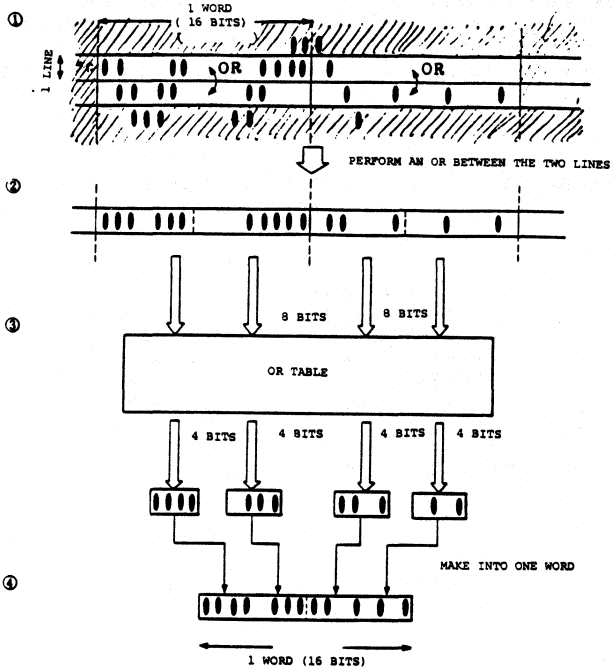
Figure 4-4
An Example of a Four-Point OR One-Half Shrinking
(45o straight line)




### 4.2.2 Algorithm

Since the four-point OR one-half shrinking method involves performing the OR operation on four adjacent points, the data that would not have been useful under the Simple One-Half Reduction method of Section 4.1 is required here. As shown in Figure 4-5, in this program a given line and the line following it are read one after another from the SRC in word units, the two lines are ORed, and the results are used to look up the OR table.

Figure 4.5
Algorithm for the Four-Point OR One-Half Shrinking Method




### 4.2.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of destination image area (DST) words in horizontal direction
M ... Number of destination image area (DST) lines in vertical direction

<Start-up token-defined parameters>

STARTS ... Source image area (SRC) starting address
STARTD ... Destination image area (DST) starting address

The allowable values of these parameters are indicated in the table below.

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 32767 | (64) |
| H | 1 - 16 | (16) |
| M | 1 - 256 | (256) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (32)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting the values of the parameters H and M. Maximum values that can be assigned to these parameters are H=16 and M=256; therefore, the maximum size destination image that can be shrunk by this program is 16 horizontal words and 256 vertical lines.

<Initial Values>

The initial values used in this program are basically the same as those employed in the Simple One-Half Shrinking Method described in Section 4.1, except that the lines that would be skipped in a simple one-half shrinking need to be read out from the SRC and initial values must be provided for the processing involved in the creation of these read addresses.

### 4.2.4 Flow Graph Explained

The four-point OR one-half shrinking method differs from the simple one-half shrinking method of Section 4.1 in the following respects:

(1) With an SRC starting address (STARTS) entered, the processing branches into two parts: A, which handles the creation of SRC addresses for every other line, as done during the simple one-half shrinking; and A', which creates those addresses which are not created by A above.
(2) These two processing steps are performed in parallel, each reading the SRC data.
(3) The two types of SRC data read in step (2) undergo an

OR operation, followed by a table lookup and the generation of shrunk data. The table used in this step is different from the one employed in the simple one-half shrinking method.

As indicated in Figure 4-6, this program consists of an SRC processing part (which reads two lines each (word unit) of SRC data and creates shrunk data) and an DST processing part (which is concerned with the generation of output addresses for the shrunk data). Each word of the shrunk data is made from four horizontally and vertically adjacent words.

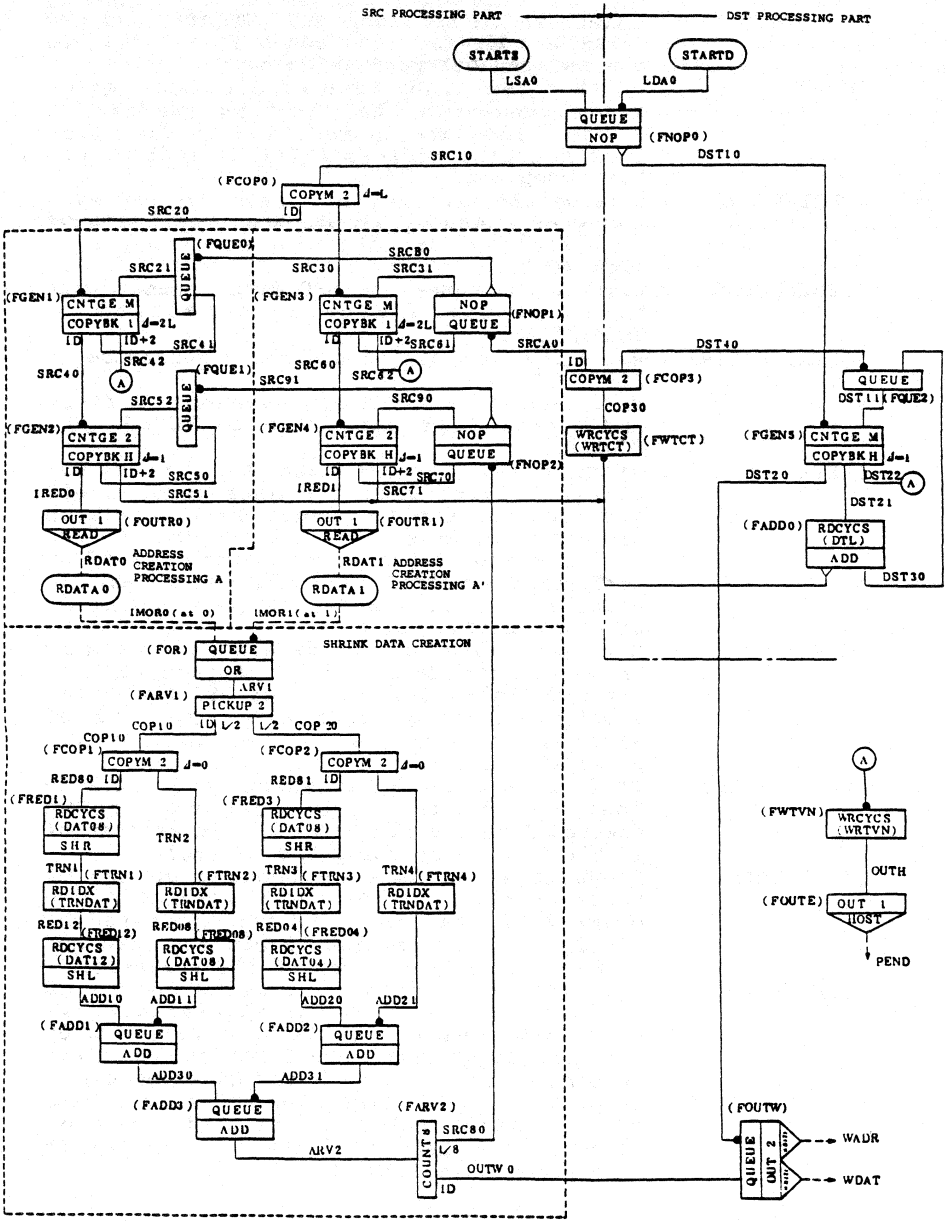
The SRC processing part receives an SRC address from the host computer that serves as the starting point of the shrinking processing. Starting with that address, the SRC processing part reads two lines each of word-unit data. An OR operation is performed between the first and second lines of data that have been read, and the resulting data is alternately distributed to the high-order 8-bit and the low-order 8-bit DST data creation processing tasks.

The high-order 8-bit and the low-order 8-bit creation processing tasks look up the OR table* on the basis of the high-order 8-bit and low-order 8-bit values received, respectively. Each then generates high-order 4-bit and low-order 4-bit data. This means 8-bit shrunk data is created by two DST words. The two 8-bit data thus generated are placed in their respective positions within the 16-bit word from which the shrunk data is generated.

The DST processing part receives a starting address from the host computer for writing the shrunk data. It generates the output addresses based on this address.

* : This is a 4-bit data table which is made by dividing 8-bit data into 2-bit segments and by performing an OR on each of the segments.

Figure 4-6
A Four-Point OR One-Half Shrinking Flow Graph



<Explanation of Nodes>

| | |
|---|---|
| FNOP0 | : Synchronizes the actions of the start-up tokens which indicate SRC (STARTS) and DST (STARTD) addresses and sends these token to the SRC and DST processing parts. |
| FCOP0 | : So that two lines each of SRC data can be read, generates the starting address of second line data in addition to the first line starting address (STARTS) sent by the host computer. |
| FGEN1/FGEN3 | : Creates the starting addresses for odd/even numbered lines of SRC data on the basis of first/second line starting addresses sent from FCOP0. |
| FGEN2/FGEN4 | : Creates addresses for SRC data lines on the basis of the addresses generated in FGEN1/FGEN3. |
| FOUTR0/FOUTR1 | : Reads the contents of addresses generated in FGEN2/FGEN4. |
| FOR | : Performs an OR operation between the first and second line data read in FOUTR0 and FOUTR1. |
| FARV1 | : Distributes the OR data sent from FOR alternately between the high-order 8-bit creation and the low-order 8-bit creation processing tasks so that a horizontal OR can be performed on two 16-bit data created by performing a vertical OR (between lines), in order to generate DST data (four-point OR one-half shrunk data). |
| FCOP1/FCOP2 | : Make two copies of the data distributed by FARV1 to provide for the fact that each of the DST high-order 8-bit and DST low-order 8-bit processing steps are comprised of high-order 4-bit and low-order 4-bit processing tasks. |
| FRED1, FRED3 | : Performs a right 8-bit shift on OR data so that a high-order 4-bit segment can be created from the high-order 8-bit segment of 16-bit OR data by using the OR table. |
| FTRN1/FTRN2 | : Prepares 4-bit data by looking up the OR table on the basis of the high-order/low-order 8-bit values of OR data (created by dividing eight bits into 2-bit segments and performing an OR on each of the segments). |
| FTRN3/FTRN4 | : Prepares 4-bit data by looking up the OR table on the basis of the high-order/low-order 8-bit values of OR data (created by dividing eight bits into 2-bit segments and performing an OR on each of the segments). |
| FRED12/FRED08 | : Prepares the high-order/low-order four bits of the high-order 8-bit DST data by performing a left 12/8 shift on the 4-bit data created in FTRN1/FTRN2. |

FRED04 : Creates the high-order four bits of the low-order 8-bit DST data by performing a left 4-bit shift on the 4-bit data created in FTRN3.

FADD1 : Creates the high-order eight bits of DST data by adding together the data generated by FRED12 and FRED08.

FADD2 : Creates the low-order eight bits of DST data by adding together the data generated by FRED04 and FTRN4.

FADD3 : Creates 16-bit DST data by adding together the data generated by FADD1 and FADD2.

FARV2 : For each 8 DST data sent from FADD3, makes two copies of the data and sends one copy to FQUE1 to notify completion of the generation of the eight DST data.

FGEN5 : Creates M lines of addresses for H horizontal words, starting from the DST address (STARTD) sent from FNOP0.

FOUTW : Writes the DST data generated by the SRC processing part into the DST address created in FGEN3.

FADD0 : Modifies the address sent from FGEN5 in order to address the starting address of the next line upon completion of address generation for H horizontal words by FGEN5.

FNOP1/FNOP0, FQUE0/FQUE1 : Regulates and synchronizes the address generation by FGEN1/FGEN2, and FGEN3/FGEN4.

FWTCT : Synchronizes the actions of the SRC and DST processing parts for each completion of SRC and DST addresses (i.e., first and second line addresses) for one line of DST data.

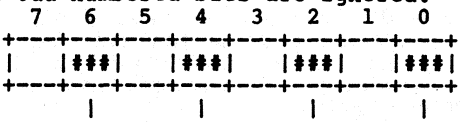
FCOP3 : Notifies the SRC and DST processing parts of the completion of addresses for one line of DST data by the SRC and DST processsing parts.

FWTVN : Synchronizes completion of all addresses by the SRC and DST processing parts.

PEND : Notifies the host computer of completion of generation for all addresses for a reduction processing task.

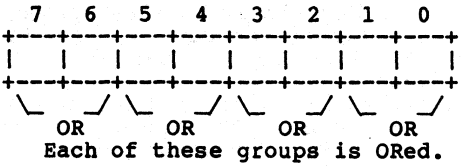
| SRC Data | Simple One-Half Shrinking Table (Note 1) | Four-Point OR One-Half Shrinking Table (Note 2) |
|---|---|---|
| Bits 7 6 5 4 3 2 1 0 | Table Data | Table Data |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 | 0 0 0 0 |
| 0 0 0 0 0 0 0 1 | 0 0 0 1 | 0 0 0 1 |
| 0 0 0 0 0 0 1 0 | 0 0 0 0 | 0 0 0 1 |
| 0 0 0 0 0 0 1 1 | 0 0 0 1 | 0 0 0 1 |
| 0 0 0 0 0 1 0 0 | 0 0 1 0 | 0 0 1 0 |
| : | : | : |
| : | : | : |
| 1 0 1 1 1 0 0 1 | 0 1 0 1 | 1 1 1 1 |
| 1 0 1 1 1 0 1 0 | 0 1 0 0 | 1 1 1 1 |
| 1 0 1 1 1 0 1 1 | 0 1 0 1 | 1 1 1 1 |
| : | : | : |
| 1 1 1 1 1 1 1 1 | 1 1 1 1 | 1 1 1 1 |

Note 1: The odd-numbered bits are ignored.




Only these bits are eliminated.

Note 2: An OR is performed between the odd-numbered and even-numbered bits.

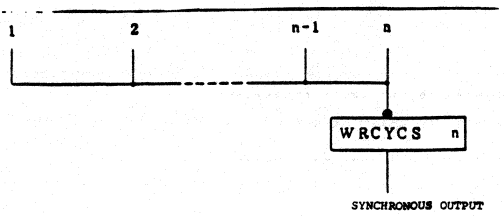



Each of these groups is ORed.

### 4.2.5 Tips on Writing Flow Graphs

Although the QUEUE instruction is normally employed in synchronizing various activities, in some cases the WRCYCS and WRCYCL instructions are used (as in the case of the FWTCT and FWTVN nodes in this program).

When BS (buffer size) = WC (write counter), the FTRC = 1 token does not vanish in either the WRCYCS or WRCYCL instruction. When there are many data senders and you wish to synchronize various actions including those of the senders, the flow graph should be constructed in the manner indicated in the figure below.




Notice that in this case n words in the DM (Data Memory) must be assigned as dummy data.

## 4.2.6 Assembler Source Listing

```
  1: ;****************************************
  2: ;
  3: ;       SHRINK 1/2  (4 POINT OR )
  4: ;
  5: ;----------------------------------------
  6: ;
  7: MODULE  IPP     =       8               ;
  8: ;
  9: EQUATE  L       =       64              ;
 10: EQUATE  M       =       256             ;
 11: EQUATE  H       =       16              ;
 12: ;
 13: EQUATE  HOST    =       0               ;
 14: EQUATE  READ    =       4               ;
 15: EQUATE  WRITE   =       5               ;
 16: ;
 17: EQUATE  STARTS  =       0               ;
 18: EQUATE  STARTD  =       32              ;
 19: ;
 20: ;****************************************
 21: ;
 22: ;       INPUT-OUTPUT
 23: ;
 24: ;----------------------------------------
 25: ;
 26: INPUT   LSA0,   LDA0,   IMOR0 AT 0,     IMOR1 AT 1      ;
 27: ;
 28: OUTPUT  RDAT0,  RDAT1,  WDAT,   WADR,   PEND            ;
 29: ;
 30: ;****************************************
 31: ;
 32: ;       LINK TABLE
 33: ;
 34: ;----------------------------------------
 35: ;
 36: LINK    SRC10,  DST10           =       FNOP0   (LSA0,  LDA0    )       ;
 37: LINK    SRC20,  SRC30           =       FCOP0   (SRC10          )       ;
 38: LINK    DST20,  DST21,  DST22   =       FGEN5   (DST11, DST10   )       ;
 39: LINK    SRC40,  SRC41,  SRC42   =       FGEN1   (SRC21, SRC20   )       ;
 40: LINK    SRC60,  SRC61,  SRC62   =       FGEN3   (SRC31, SRC30   )       ;
 41: LINK    IRED0,  SRC50,  SRC51   =       FGEN2   (SRC52, SRC40   )       ;
 42: LINK    SRC21                   =       FQUE0   (SRC41, SRC80   )       ;
 43: LINK    OUTH                    =       FWTVN   (       ,SRC42  )       ;
 44: LINK    RDAT0                   =       FOUTR0  (IRED0          )       ;
 45: LINK    SRC52                   =       FQUE1   (SRC50, SRC91   )       ;
 46: LINK    COP30                   =       FWTCT   (       ,SRC51  )       ;
 47: LINK    IRED1,  SRC70,  SRC71   =       FGEN4   (SRC90, SRC60   )       ;
 48: LINK    SRC31,  SRC80           =       FNOP1   (SRC61, SRCA0   )       ;
 49: LINK    OUTH                    =       FWTVN   (       ,SRC62  )       ;
 50: LINK    RDAT1                   =       FOUTR1  (IRED1          )       ;
 51: LINK    SRC90,  SRC91           =       FNOP2   (SRC70, SRC80   )       ;
 52: LINK    COP30                   =       FWTCT   (       ,SRC71  )       ;
 53: LINK    ARV1                    =       FOR     (IMOR0, IMOR1   )       ;
 54: LINK    COP10,  COP20           =       FARV1   (ARV1           )       ;
 55: LINK    RED00,  TRN2            =       FCOP1   (COP10          )       ;
 56: LINK    TRN1                    =       FRED1   (RED00          )       ;
 57: LINK    RED12                   =       FTRN1   (TRN1           )       ;
 58: LINK    ADD10                   =       FRED12  (RED12          )       ;
 59: LINK    RED08                   =       FTRN2   (TRN2           )       ;
 60: LINK    ADD11                   =       FRED08  (RED08          )       ;
```

```
LINK     ADD30                    =      FADD1   (ADD10. ADD11    )  ;
LINK     RED81.  TRN4             =      FCOP2   (COP20           )  ;
LINK     TRN3                     =      FRED3   (RED81           )  ;
LINK     RED04                    =      FTRN3   (TRN3            )  ;
LINK     ADD20                    =      FRED04  (RED04           )  ;
LINK     ADD21                    =      FTRN4   (TRN4            )  ;
LINK     ADD31                    =      FADD2   (ADD20. ADD21    )  ;
LINK     ARV2                     =      FADD3   (ADD30. ADD31    )  ;
LINK     OUTW0.  SRC80            =      FARV2   (ARV2            )  ;
LINK     WDAT.   WADR             =      FOUTW   (OUTW0. DST20    )  ;
LINK     DST30.  DST31            =      FADD0   (DST21           )  ;
LINK     OUTH                     =      FWTVN   (       ,DST22   )  ;
LINK     DST11                    =      FQUE2   (DST30. DST40    )  ;
LINK     COP30                    =      FWTCT   (       ,DST31   )  ;
LINK     SRCA0.  DST40            =      FCOP3   (COP30           )  ;
LINK     PEND                     =      FOUTE   (OUTH            )  ;
:
:****************************************
;
;       FUNCTION TABLE
;
;-------------------------------------
;
FUNCTION    FOUTR0  =   OUT1    (READ.  0)                                  ;
FUNCTION    FOUTR1  =   OUT1    (READ.  1)                                  ;
FUNCTION    FOUTW   =   OUT2    (WRTIE. 20H. 0),QUEUE   (QUEW.  16)         ;
FUNCTION    FOUTE   =   OUT1    (HOST.  0)                                  ;
FUNCTION    FGEN1   =   COPYBK  (1.   2*L),     CNTGE   (M          )       ;
FUNCTION    FGEN2   =   COPYBK  (M .    1),     CNTGE   (2          )       ;
FUNCTION    FGEN3   =   COPYBK  (1.   2*L),     CNTGE   (M          )       ;
FUNCTION    FGEN4   =   COPYBK  (M.     1),     CNTGE   (2          )       ;
FUNCTION    FGEN5   =   COPYBK  (M.     1),     CNTGE   (M          )       ;
FUNCTION    FCOP0   =   COPYM   (2.     L)                                  ;
FUNCTION    FCOP1   =   COPYM   (2.     0)                                  ;
FUNCTION    FCOP2   =   COPYM   (2.     0)                                  ;
FUNCTION    FCOP3   =   COPYM   (2.     0)                                  ;
FUNCTION    FARV1.  =   PICKUP  (2       )                                  ;
FUNCTION    FARV2   =   COUNT   (8       )                                  ;
FUNCTION    FRED1   =   SHR     (X       ),     RDCYCS  (DAT08. 1)          ;
FUNCTION    FRED3   =   SHR     (X       ),     RDCYCS  (DAT08. 1)          ;
FUNCTION    FRED12  =   SHL     (X       ),     RDCYCS  (DAT12. 1)          ;
FUNCTION    FRED08  =   SHL     (X       ),     RDCYCS  (DAT08. 1)          ;
FUNCTION    FRED04  =   SHL     (X       ),     RDCYCS  (DAT04. 1)          ;
FUNCTION    FTRN1   =   RDIDX   (TRNDAT  )                                  ;
FUNCTION    FTRN2   =   RDIDX   (TRNDAT  )                                  ;
FUNCTION    FTRN3   =   RDIDX   (TRNDAT  )                                  ;
FUNCTION    FTRN4   =   RDIDX   (TRNDAT  )                                  ;
FUNCTION    FADD0   =   ADD     (XY      ),     RDCYCS  (DTL.   1)          ;
FUNCTION    FADD1   =   ADD     (X       ),     QUEUE   (QUE6.  8)          ;
FUNCTION    FADD2   =   ADD     (X       ),     QUEUE   (QUE7.  8)          ;
FUNCTION    FADD3   =   ADD     (X       ),     QUEUE   (QUE8.  8)          ;
FUNCTION    FNOP0   =   NOP     (XY      ),     QUEUE   (QUE3.  1)          ;
FUNCTION    FNOP1   =   NOP     (XY      ),     QUEUE   (QUE4.  1)          ;
FUNCTION    FNOP2   =   NOP     (XY      ),     QUEUE   (QUE5.  1)          ;
FUNCTION    FQUE0   =   QUEUE   (QUE0.  1)                                  ;
FUNCTION    FQUE1   =   QUEUE   (QUE1.  1)                                  ;
FUNCTION    FQUE2   =   QUEUE   (QUE2.  1)                                  ;
FUNCTION    FWTCT   =   WRCYCS  (WRTCT. 3)                                  ;
FUNCTION    FWTVN   =   WRCYCS  (WRTVN. 3)                                  ;
FUNCTION    FOR     =   OR      (X       ),     QUEUE   (QUE9.  16)         ;
```

```
121: ;
122: ;****************************************
123: ;
124: ;       DATA MEMORY
125: ;
126: ;----------------------------------------
127: ;
128: MEMORY  QUE0    =       AREA    (1      )       ;
129: MEMORY  QUE1    =       AREA    (1      )       ;
130: MEMORY  QUE2    =       AREA    (1      )       ;
131: MEMORY  QUE3    =       AREA    (1      )       ;
132: MEMORY  QUE4    =       AREA    (1      )       ;
133: MEMORY  QUE5    =       AREA    (1      )       ;
134: MEMORY  QUE6    =       AREA    (8      )       ;
135: MEMORY  QUE7    =       AREA    (8      )       ;
136: MEMORY  QUE8    =       AREA    (8      )       ;
137: MEMORY  QUE9    =       AREA    (16     )       ;
138: MEMORY  QUEW    =       AREA    (16     )       ;
139: MEMORY  DAT04   =       4                       ;
140: MEMORY  DAT08   =       8                       ;
141: MEMORY  DAT12   =       12                      ;
142: MEMORY  DTL     =       L-H                     ;
143: MEMORY  WRTCT   =       AREA    (3      )       ;
144: MEMORY  WRTVN   =       AREA    (3      )       ;
145: MEMORY  TRNDAT  =        0, 1, 1, 1, 2, 3, 3, 3, 2, 3, 3, 3, 2, 3, 3, 3,
146:                          4, 5, 5, 5, 6, 7, 7, 7, 6, 7, 7, 7, 6, 7, 7, 7,
147:                          4, 5, 5, 5, 6, 7, 7, 7, 6, 7, 7, 7, 6, 7, 7, 7,
148:                          4, 5, 5, 5, 6, 7, 7, 7, 6, 7, 7, 7, 6, 7, 7, 7,
149:                          8, 9, 9, 9,10,11,11,11,10,11,11,11,10,11,11,11,
150:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
151:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
152:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
153:                          8, 9, 9, 9,10,11,11,11,10,11,11,11,10,11,11,11,
154:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
155:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
156:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
157:                          8, 9, 9, 9,10,11,11,11,10,11,11,11,10,11,11,11,
158:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
159:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15,
160:                         12,13,13,13,14,15,15,15,14,15,15,15,14,15,15,15 ;
161: ;
162: ;****************************************
163: ;
164: ;       START
165: ;
166: ;----------------------------------------
167: START                                       ;
168: ;
169: DATA    EXEC    (IPP,   LSA0,   STARTS  )       ;
170: DATA    EXEC    (IPP,   LDA0,   STARTD  )       ;
171: ;
172: END                                         ;
```

## 4.3 Neighboring 16-Point Addition One-Quarter Shrinking

### 4.3.1 Processing Explained

A one-quarter shrinking involves shrinking 4 horizontal and 4 vertical bits, for a total of 16 bits, into one bit. In this neighboring 16-point addition reduction, the number of "1"s in the 16-bit data are counted and shrunk data is set to "1" if the sum is greater than a given value and "0" otherwise.

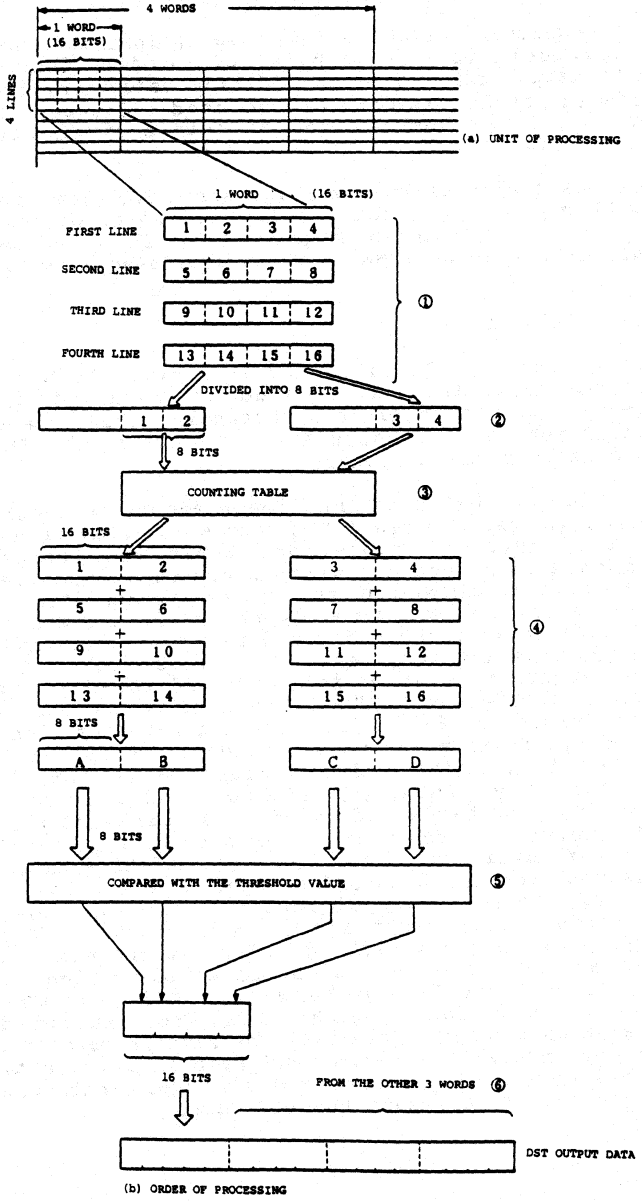
The basic size of source image area (SRC) that can be shrunk by this program is four horizontal words by four vertical words, as shown in Figure 4-7. This is because the destination image area (DST) is addressed in word units.

### 4.3.2 Algorithm

The number of "1"s contained in a given 16-bit data, four bits by four lines, is counted. If the number is number is greater than or equal to a threshold value, the shrunk data bit is set to "1"; otherwise, it is set to "0". The counting is done by a table lookup.

To write the DST data into image memory in 1-word (16-bit) units, (1) each of the 4 SRC data words read from the SRC is divided into (2) high-order 8-bit and low-order 8-bit segments to reduce the size of the table required, and (3) the table is looked up for each segment. The contents of this table are determined by the following method: each 8-bit segment is further divided into high-order and low-order 4-bit subsegments, and the number of "1"s in these subsegments is written in a word addressed by the value of each 8-bit segment. (The table used in the example program is 256-words x 16-bit data divided into the high-order and low-order 8-bit parts whose contents are the number of "1"s in the high-order and low-order 4-bits of the segment respectively.) Then, (4) data for four vertical lines are read from the table and added, thus tallying the number of "1"s included in the area encompassed by four vertical lines and four horizontal bits. Then two additive data, AB and CD (created from the high-order eight bits and low-order eight bits of SRC data), are divided into (5) high-order (A,C) and low-order (B,D) eight bits. Each of these four data is compared with the threshold value. If the number of "1"s is greater than or equal to the threshold value, the shrunk data is set to "1"; otherwise, it is set to "0". As a result, the four 1-bit data are extracted as the constituents of DST data. (6) The same processing is done on the continuous three words and four lines, yielding 12-bit shrunk data. This is combined with the 4-bit data from the previous step to create the DST data.

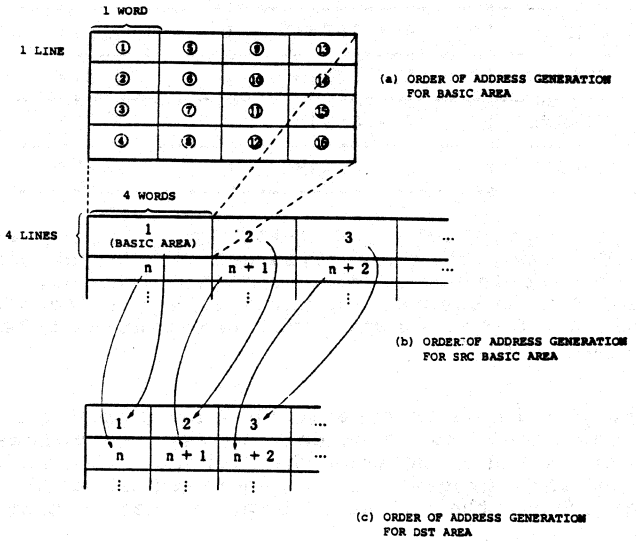
Figure 4-7
Algorithm for the Neighboring 16-Point Addition
One-Quarter Shrinking Method



The generation of SRC area addresses is based on the generation of addresses for an area defined by four horizontal words and four vertical lines. This constitutes the minimum unit of a shrinking processing. This basic area is shifted horizontally (Figure 4-8 (b)). The addresses in the basic area are generated four vertical lines at a time, as indicated in Figure 4-8 (a).

For the generation of addresses for a DST area, one address is created horizontally for each 16 addresses of the SRC basic area, as shown in Figure 4-8 (c).

Figure 4-8
Order of Address Generation




### 4.3.3 Parameters and Their Applicable Range

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area (SRC) words in horizontal direction
M ... Number of source image area (SRC) lines in vertical direction

<Start-up token-defined parameters>

THRDAT ... Threshold value
STARTS ... Source image area (SRC) starting address
STARTD ... Destination image area (DST) starting address

The allowable values of these parameters are indicated in the table below.

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 16383 | (64) |
| H | 4 - 1024** | (32) |
| M | 4 - 1024** | (512) |
| THRDAT | 0 - 65535 | ( 8)* |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (48)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.
** : A multiple of 4

Since no provision is made in this program for switching banks, you should exercise care in setting values of the parameters H and M. Because this minimum area that can be shrunk by this program is four horizontal words by four vertical lines, the values of H and M must be multiples of four.

### 4.3.4 Flow Graph Explained

Since, in this program the image memory is accessed in units of 16 bits, data are written to DST in word units. This also simplifies processing. Therefore, the SRC area is treated by using a 4-word horizontal by 4-line vertical area as the minimum unit.

The following explains the flow graph in Figure 4-9.

FGEC1 creates the addresses for every other four vertical lines of the SRC area, starting from the SRC starting

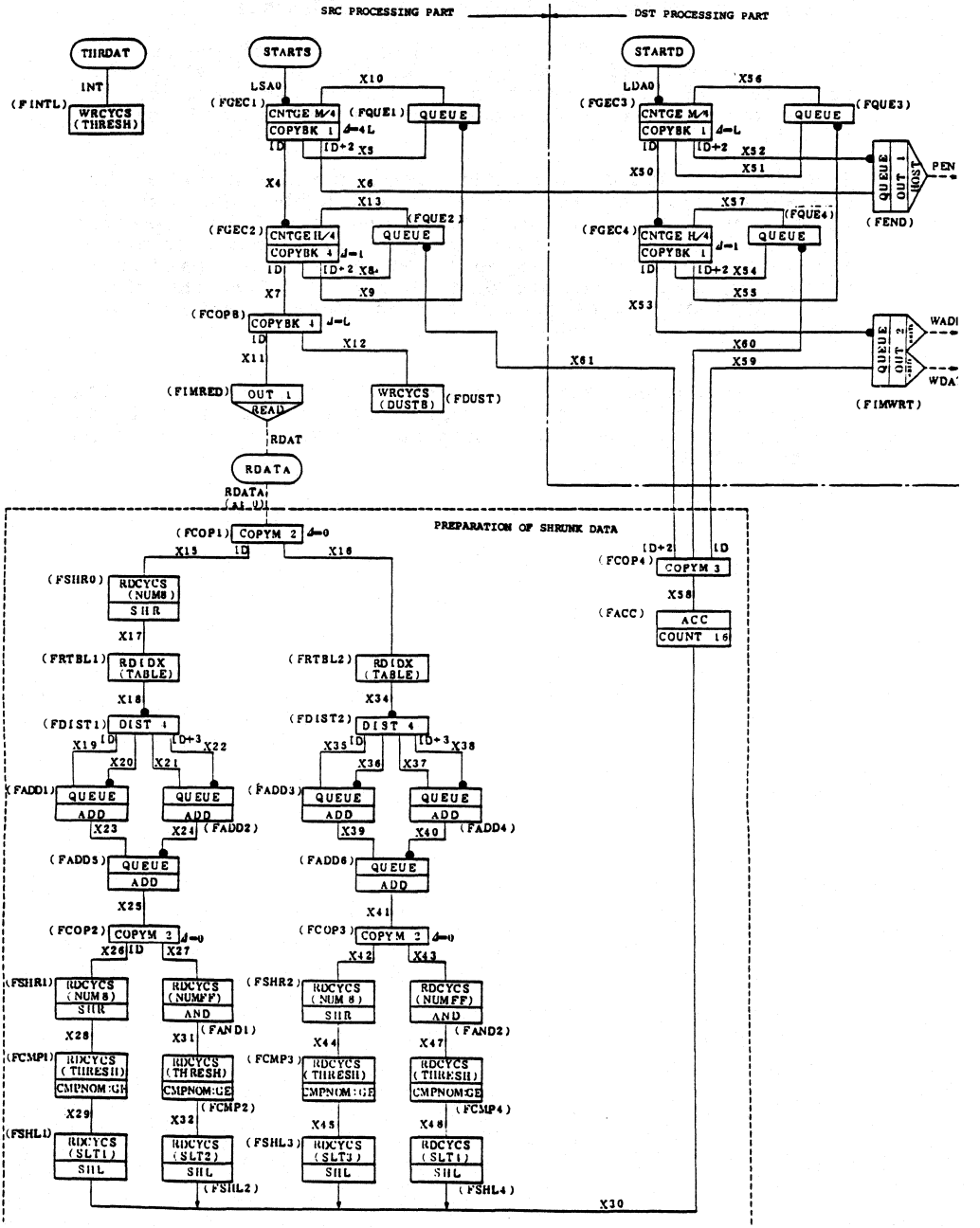
address (STARTS) given by the host computer. Then, FGEC2 creates four horizontal addresses from the given addresses. Upon receipt of these addresses, FCOPB creates addresses for four lines, making them into SRC read addresses.

FCOP1 divides the SRC data into high-order and low-order 8-bit segments (making two copies of the read data) so that when the word is segmented into four bits the number of "1"s contained in the 4-bit segments can be counted using a table*.

Specifically, FRTBL1 and FTTBL2 determine the number of "1"s contained in four 4-bit segments, the high-order and low-order 4-bit segments of the high-order 8-bit segment (FRTBL1), and the high-order and low-order 4-bit segments of the low-order 8-bit segment (FTTBL2); the results of processing four lines are distributed by FDIST1 and FDST2; and additions are performed for the four lines by FADD5, FADD3, FADD4, and FADD6.

* : The table represents, in eight bits, the number of "1"s contained in the high-order 4-bit and low-order 4-bit segments of the value that is an address to reference the table.

Figure 4-9
A Flow Graph of the Neighboring 16-Point Addition
One-Quarter Shrinking Method



The high-order 8-bit and low-order 8-bit segments of the data resulting from the addition operation indicate the number of "1"s in the four bit by four line area and the number of "1"s in the next area, respectively; high-order eight bits and low-order eight bits are divided into two words. If the value of this word is less than the value specified in the program, 1-bit data "0" is created and shifted by FSHL1 through FSHL4 so as to create the constituent of DST data. Otherwise, 1-bit data "1" is created and shifted in the same manner.

One DST address is created in the horizontal direction for each 16 addresses (4 horizontal words by 4 vertical lines) of the SRC area, using the DST starting address (STARTD) sent from the host computer as the starting point.

When all 16 bits of DST data have been generated, they are written to the given DST address.

### 4.3.5 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;       SHRINK 1/4
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  L       =       64              ;
10: EQUATE  H       =       32              ;
11: EQUATE  M       =       512             ;
12: ;
13: EQUATE  THRDAT  =       8               ;
14: EQUATE  STARTS  =       0               ;
15: EQUATE  STARTD  =       48              ;
16: ;
17: EQUATE  HOST    =       0               ;
18: EQUATE  READ    =       4               ;
19: EQUATE  WRITE   =       5               ;
20: ;
21: ;****************************************
22: ;
23: ;       INPUT-OUTPUT
24: ;
25: ;----------------------------------------
26: ;
27: INPUT   INT.    LSA0.   LDA0.   RDATA AT 0      ;
28: ;
29: OUTPUT  RDAT.   WDAT.   WADR.   PEND            ;
30: ;
31: ;****************************************
32: ;
33: ;       LINK TABLE
34: ;
35: ;----------------------------------------
36: ;
37: LINK                            =       FINTL   (INT            )       ;
38: LINK    X4.     X5.     X6      =       FGEC1   (X10.   LSA0    )       ;
39: LINK    X7.     X8.     X9      =       FGEC2   (X13.   X4      )       ;
40: LINK    X10                     =       FQUE1   (X5.    X9      )       ;
41: LINK    PEND                    =       FEND    (X6.    X52     )       ;
42: LINK    X11.    X12             =       FCOP8   (X7             )       ;
43: LINK    X13                     =       FQUE2   (X8.    X61     )       ;
44: LINK    RDAT                    =       FINRED  (X11            )       ;
45: LINK                            =       FDUST   (X12            )       ;
46: LINK    X15.    X16             =       FCOP1   (RDATA          )       ;
47: LINK    X17                     =       FSHR0   (X15            )       ;
48: LINK    X18                     =       FRTBL1  (X17            )       ;
49: LINK    X19.  X20.  X21. X22    =       FDIST1  (       .X18    )       ;
50: LINK    X23                     =       FADD1   (X19.   X20     )       ;
51: LINK    X24             51      =       FADD2   (X21.   X22     )       ;
52: LINK    X25                     =       FADD5   (X23.   X24     )       ;
53: LINK    X26.    X27             =       FCOP2   (X25            )       ;
54: LINK    X28                     =       FSHR1   (X26            )       ;
```

```
 55: LINK    X29                          =       FCMP1   (X28             )  ;
 56: LINK    X30                          =       FSHL1   (X29             )  ;
 57: LINK    X31                          =       FAND1   (X27             )  ;
 58: LINK    X32                          =       FCMP2   (X31             )  ;
 59: LINK    X30                          =       FSHL2   (X32             )  ;
 60: LINK    X34                          =       FRTBL2  (X16             )  ;
 61: LINK    X35,  X36,  X37,  X38        =       FDIST2  (       ,X34     )  ;
 62: LINK    X39                          =       FADD3   (X35,   X36      )  ;
 63: LINK    X40                          =       FADD4   (X37,   X38      )  ;
 64: LINK    X41                          =       FADD6   (X39,   X40      )  ;
 65: LINK    X42,    X43                  =       FCOP3   (X41             )  ;
 66: LINK    X44                          =       FSHR2   (X42             )  ;
 67: LINK    X45                          =       FCMP3   (X44             )  ;
 68: LINK    X30                          =       FSHL3   (X45             )  ;
 69: LINK    X47                          =       FAND2   (X43             )  ;
 70: LINK    X48                          =       FCMP4   (X47             )  ;
 71: LINK    X30                          =       FSHL4   (X48             )  ;
 72: LINK    X58                          =       FACC    (X30             )  ;
 73: LINK    X59,    X60,    X61          =       FCOP4   (X58             )  ;
 74: LINK    X50,    X51,    X52          =       FGEC3   (X56,   LDA0     )  ;
 75: LINK    X53,    X54,    X55          =       FGEC4   (X57,   X50      )  ;
 76: LINK    X56                          =       FQUE3   (X51,   X55      )  ;
 77: LINK    WDAT,   WADR                 =       FIWRT   (X59,   X53      )  ;
 78: LINK    X57                          =       FQUE4   (X54,   X60      )  ;
 79: ;
 80: ;**************************************
 81: ;
 82: ;       FUNCTION TABLE
 83: ;
 84: ;--------------------------------------
 85: ;
 86: FUNCTION   FINTL   =   WRCYCS  (THRESH,1)                                  ;
 87: FUNCTION   FINRED  =   OUT1    (READ,  0)                                  ;
 88: FUNCTION   FINWRT  =   OUT2    (WRTIE, 20H, 0),QUEUE   (QUEW, 1)           ;
 89: FUNCTION   FEND    =   OUT1    (HOST,  0),     QUEUE   (QUEE, 1)           ;
 90: FUNCTION   FGEC1   =   COPYBK  (1,   4*L),     CNTGE   (M/4     )          ;
 91: FUNCTION   FGEC2   =   COPYBK  (4,     1),     CNTGE   (M/4     )          ;
 92: FUNCTION   FGEC3   =   COPYBK  (1,     L),     CNTGE   (M/4     )          ;
 93: FUNCTION   FGEC4   =   COPYBK  (1,     1),     CNTGE   (M/4     )          ;
 94: FUNCTION   FQUE1   =   QUEUE   (QUE1,  1)                                  ;
 95: FUNCTION   FQUE2   =   QUEUE   (QUE2,  1)                                  ;
 96: FUNCTION   FQUE3   =   QUEUE   (QUE3,  1)                                  ;
 97: FUNCTION   FQUE4   =   QUEUE   (QUE4,  1)                                  ;
 98: FUNCTION   FCOPB   =   COPYBK  (4,     L)                                  ;
 99: FUNCTION   FCOP1   =   COPYM   (2,     0)                                  ;
100: FUNCTION   FCOP2   =   COPYM   (2,     0)                                  ;
101: FUNCTION   FCOP3   =   COPYM   (2,     0)                                  ;
102: FUNCTION   FCOP4   =   COPYM   (3,     0)                                  ;
103: FUNCTION   FSHR0   =   SHR     (X       ),     RDCYCS  (NUM8,  1)          ;
104: FUNCTION   FSHR1   =   SHR     (X       ),     RDCYCS  (NUM8,  1)          ;
105: FUNCTION   FSHR2   =   SHR     (X       ),     RDCYCS  (NUM8,  1)          ;
106: FUNCTION   FSHL1   =   SHL     (X       ),     RDCYCS  (SLT1,  4)          ;
107: FUNCTION   FSHL2   =   SHL     (X       ),     RDCYCS  (SLT2,  4)          ;
108: FUNCTION   FSHL3   =   SHL     (X       ),     RDCYCS  (SLT3,  4)          ;
109: FUNCTION   FSHL4   =   SHL     (X       ),     RDCYCS  (SLT4,  4)          ;
110: FUNCTION   FAND1   =   AND     (X       ),     RDCYCS  (NUMFF, 1)          ;
111: FUNCTION   FAND2   =   AND     (X       ),     RDCYCS  (NUMFF, 1)          ;
112: FUNCTION   FCMP1   =   CMPNOM  (X,    GE),     RDCYCS  (THRESH,1)          ;
113: FUNCTION   FCMP2   =   CMPNOM  (X,    GE),     RDCYCS  (THRESH,1)          ;
114: FUNCTION   FCMP3   =   CMPNOM  (X,    GE),     RDCYCS  (THRESH,1)          ;
115: FUNCTION   FCMP4   =   CMPNOM  (X,    GE),     RDCYCS  (THRESH,1)          ;
116: FUNCTION   FRTBL1  =   RDIDX   (TABLE   )                                  ;
117: FUNCTION   FRTBL2  =   RDIDX   (TABLE   )                                  ;
118: FUNCTION   FADD1   =   ADD     (X       ),     QUEUE   (QUEAD1,1)          ;
119: FUNCTION   FADD2   =   ADD     (X       ),     QUEUE   (QUEAD2,1)          ;
120: FUNCTION   FADD3   =   ADD     (X       ),     QUEUE   (QUEAD3,1)          ;
121: FUNCTION   FADD4   =   ADD     (X       ),     QUEUE   (QUEAD4,1)          ;
122: FUNCTION   FADD5   =   ADD     (X       ),     QUEUE   (QUEAD5,1)          ;
123: FUNCTION   FADD6   =   ADD     (X       ),     QUEUE   (QUEAD6,1)          ;
124: FUNCTION   FDIST1  =   DIST    (4       )                                  ;
125: FUNCTION   FDIST2  =   DIST    (4       )                                  ;
126: FUNCTION   FACC    =   ACC     (X       ),     COUNT   (16      )          ;
127: FUNCTION   FDUST   =   WRCYCS  (DUSTB, 1)                                  ;
128: ;
```

```
129: ;****************************************
130: ;
131: ;       DATA MEMORY
132: ;
133: ;----------------------------------------
134: ;
135: MEMORY  NUM8    =       8                                       ;
136: MEMORY  NUMFF    =       0FFH                                    ;
137: MEMORY  SLT1    =       15.     11.     7.      3               ;
138: MEMORY  SLT2    =       14.     10.     6.      2               ;
139: MEMORY  SLT3    =       13.     9.      5.      1               ;
140: MEMORY  SLT4    =       12.     8.      4.      0               ;
141: MEMORY  THRESH  =       AREA    (1      )                       ;
142: MEMORY  DUSTB   =       AREA    (1      )                       ;
143: MEMORY  QUEW    =       AREA    (1      )                       ;
144: MEMORY  QUEE    =       AREA    (1      )                       ;
145: MEMORY  QUE1    =       AREA    (1      )                       ;
146: MEMORY  QUE2    =       AREA    (1      )                       ;
147: MEMORY  QUE3    =       AREA    (1      )                       ;
148: MEMORY  QUE4    =       AREA    (1      )                       ;
149: MEMORY  QUEAD1  =       AREA    (1      )                       ;
150: MEMORY  QUEAD2  =       AREA    (1      )                       ;
151: MEMORY  QUEAD3  =       AREA    (1      )                       ;
152: MEMORY  QUEAD4  =       AREA    (1      )                       ;
153: MEMORY  QUEAD5  =       AREA    (1      )                       ;
154: MEMORY  QUEAD6  =       AREA    (1      )                       ;
155: MEMORY  TABLE   =       0000H.  0001H.  0001H.  0002H.
156:                         0001H.  0002H.  0002H.  0003H.
157:                         0001H.  0002H.  0002H.  0003H.
158:                         0002H.  0003H.  0003H.  0004H.
159:                         0100H.  0101H.  0101H.  0102H.
160:                         0101H.  0102H.  0102H.  0103H.
161:                         0101H.  0102H.  0102H.  0103H.
162:                         0102H.  0103H.  0103H.  0104H.
163:                         0100H.  0101H.  0101H.  0102H.
164:                         0101H.  0102H.  0102H.  0103H.
165:                         0101H.  0102H.  0102H.  0103H.
166:                         0102H.  0103H.  0103H.  0104H.
167:                         0200H.  0201H.  0201H.  0202H.
168:                         0201H.  0202H.  0202H.  0203H.
169:                         0201H.  0202H.  0202H.  0203H.
170:                         0202H.  0203H.  0203H.  0204H.
171:                         0100H.  0101H.  0101H.  0102H.
172:                         0101H.  0102H.  0102H.  0103H.
173:                         0101H.  0102H.  0102H.  0103H.
174:                         0102H.  0103H.  0103H.  0104H.
175:                         0200H.  0201H.  0201H.  0202H.
176:                         0201H.  0202H.  0202H.  0203H.
177:                         0201H.  0202H.  0202H.  0203H.
178:                         0202H.  0203H.  0203H.  0204H.
179:                         0200H.  0201H.  0201H.  0202H.
180:                         0201H.  0202H.  0202H.  0203H.
```

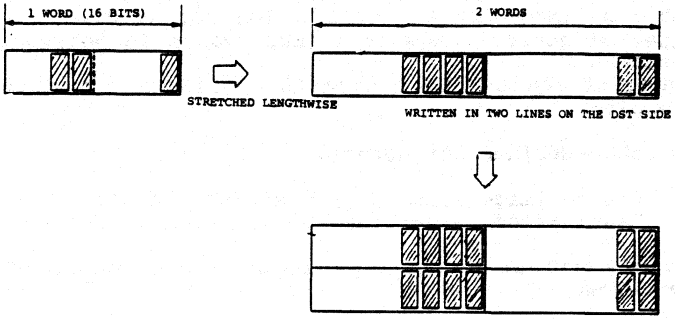
```
                                0201H.  0202H.  0202H.  0203H.
                                0202H.  0203H.  0203H.  0204H.
                                0300H.  0301H.  0301H.  0302H.
                                0301H.  0302H.  0302H.  0303H.
                                0301H.  0302H.  0302H.  0303H.
                                0302H.  0303H.  0303H.  0304H.
                                0100H.  0101H.  0101H.  0102H.
                                0101H.  0102H.  0102H.  0103H.
                                0101H.  0102H.  0102H.  0103H.
                                0102H.  0103H.  0103H.  0104H.
                                0200H.  0201H.  0201H.  0202H.
                                0201H.  0202H.  0202H.  0203H.
                                0201H.  0202H.  0202H.  0203H.
                                0202H.  0203H.  0203H.  0204H.
                                0200H.  0201H.  0201H.  0202H.
                                0201H.  0202H.  0202H.  0203H.
                                0201H.  0202H.  0202H.  0203H.
                                0202H.  0203H.  0203H.  0204H.
                                0300H.  0301H.  0301H.  0302H.
                                0301H.  0302H.  0302H.  0303H.
                                0301H.  0302H.  0302H.  0303H.
                                0302H.  0303H.  0303H.  0304H.
                                0200H.  0201H.  0201H.  0202H.
                                0201H.  0202H.  0202H.  0203H.
                                0201H.  0202H.  0202H.  0203H.
                                0202H.  0203H.  0203H.  0204H.
                                0300H.  0301H.  0301H.  0302H.
                                0301H.  0302H.  0302H.  0303H.
                                0301H.  0302H.  0302H.  0303H.
                                0302H.  0303H.  0303H.  0304H.
                                0300H.  0301H.  0301H.  0302H.
                                0301H.  0302H.  0302H.  0303H.
                                0301H.  0302H.  0302H.  0303H.
                                0302H.  0303H.  0303H.  0304H.
                                0400H.  0401H.  0401H.  0402H.
                                0401H.  0402H.  0402H.  0403H.
                                0401H.  0402H.  0402H.  0403H.
                                0402H.  0403H.  0403H.  0404H   ;
;
;*****************************************
;
;       START
;
;-----------------------------------------
START                                                           ;
;
DATA    EXEC    (IPP,   INT,    THRDAT  )                       ;
DATA    EXEC    (IPP,   LSA0,   STARTS  )                       ;
DATA    EXEC    (IPP,   LDA0,   STARTD  )                       ;
;
END                                                             ;
```

## 4.4 Simple Double Enlargement

### 4.4.1 Processing Explained

In simple double enlargement, data is read from the the source image area (SRC) and each bit is duplicated, as shown in Figure 4-10, to double the data in horizontal direction. The data are also written in two consecutive lines of the destination image area (DST) to double them vertically.

Figure 4-10
Simple Double Enlargement

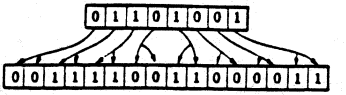



### 4.4.2 Algorithm

As shown in Figure 4-10, (1) one word is read from the SRC, then (2) the word is separated into high-order 8-bit and low-order 8-bit segments*. The SRC data divided into two 8-bit segments (3) are used in looking up the double enlargement table, in terms of the high-order and low-order segment values, and become two words of 16-bit data (see Figure 4-11). Each of two words of the 16-bit data (4) is written to the contiguous addresses and also in the contiguous lines of the DST. (5) This processing is performed on all SRC data.

* : This procedure is intended to double the data by means of a table lookup. However, if one word (16 bits) were used directly as an address for table lookup, 65,536 locations would be required, exceeding the number of the uPD7281 DM words (512 locations). Therefore, it is necessary to segment the data read from the SRC into 8-bit data. In this case, only 256 locations are required for a lookup table.

Figure 4-11
An Example of Double Horizontal Enlargement




### 4.4.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area (SRC) words in horizontal direction
M ... Number of source image area (SRC) lines in vertical direction

<Start-up token-defined parameters>

STARTS ... Source image area (SRC) starting address
STARTD ... Destination image area (DST) starting address

The allowable values of these parameters are indicated in the table below.

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535 | (64) |
| H | 1 - 256 | ( 8) |
| M | 1 - 256 | (128) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (32)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

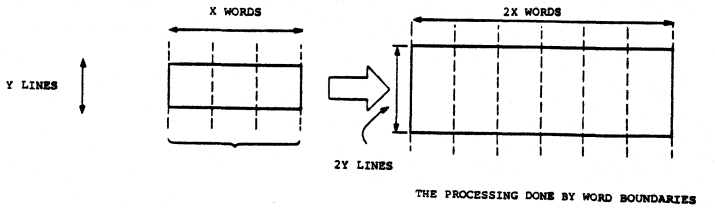
Since no provision is made in this program for switching banks, you should exercise care in setting the values of parameters H and M.

The maximum size of source image area allowed is 256 horizontal words and 256 vertical lines.

<Initial values>

To set initial values, determine the size of the SRC to be expanded horizontally in word units (word boundary), and the size in vertical line units.

Since this is a double enlargement, the size of the DST after enlargement is determined automatically once the size of the SRC is determined.




### 4.4.4 Flow Graph Explained

A flow graph for this program is shown in Figure 4-12.

Start-up tokens indicating the starting addresses, STARTS and STARTD, of the SRC and DST areas required in performing the enlargement processing are received from the host computer and sent to the SRC and DST processing parts, respectively.

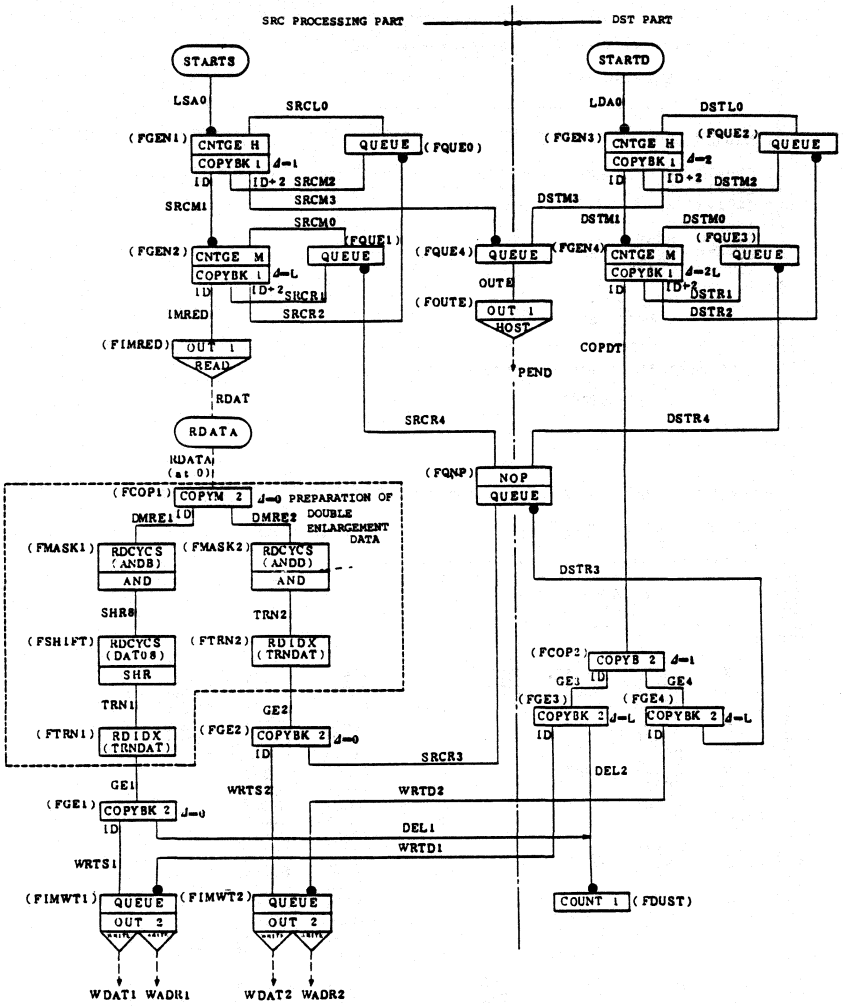
The SRC processing part creates SRC addresses on the basis of the SRC address (STARTS) and reads the contents of these addresses. The SRC addresses are created in FGEN1 and FGEN2 contiguously in the vertical direction starting from STARTS. FIMRED reads the contents of the addresses thus created. Two copies of this SRC data are made for reasons explained above, and the values of high-order eight bits and low-order eight bits are used in looking up the double enlargement table, with the result being DST data. Then, FGE1 and FGE2 make two copies of the DST data to create output data for two lines.

The DST processing part creates DST addresses on the basis of the DST starting address (STARTD).

Four DST addresses, consisting of two contiguous words by two lines, must be created to write one SRC data. First, FGEN3 and FGEN4 create addresses for every other word horizontally and every other line vertically, starting from STARTD. Upon receipt of these addresses, FCOP2 creates addresses for two contiguous words. Further, FGE3 and FGE4 create two lines of addresses for two contiguous words as the addresses for writing the DST data generated in the SRC processing part.

The writing of the DST data is done in FIMWT1 and FIMWT2.

**Figure 4.12**
**A flow Chart of a Simple Double Enlargement**



FIMWT1 writes the DST (enlargement) data representing the high-order eight bits of the data read from SRC, and FIMWT2 writes the low-order eight bits .

### 4.4.5 Assembler Source Listing

```
  1: ;**************************************
  2: ;
  3: ;       ENLARGE X2
  4: ;
  5: ;--------------------------------------
  6: ;
  7: MODULE  IPP     =       8       ;
  8: ;
  9: EQUATE  L       =       64      ;
 10: EQUATE  H       =       8       ;
 11: EQUATE  M       =       128     ;
 12: ;
 13: EQUATE  HOST    =       0       ;
 14: EQUATE  READ    =       4       ;
 15: EQUATE  WRITE   =       5       ;
 16: ;
 17: EQUATE  STARTS  =       0       ;
 18: EQUATE  STARTD  =       32      ;
 19: ;
 20: ;
 21: ;**************************************
 22: ;
 23: ;       INPUT-OUTPUT
 24: ;
 25: ;--------------------------------------
 26: ;
 27: ;
 28: ;
 29: INPUT   LSA0,   LDA0,   RDATA AT 0                      ;
 30: ;
 31: OUTPUT  RDAT,   WDAT1,  WADR1,  WDAT2,  WADR2,  PEND    ;
 32: ;
 33: ;**************************************
 34: ;
 35: ;       LINK TABLE
 36: ;
 37: ;--------------------------------------
 38: ;
 39: ;
 40: LINK    SRCM1,  SRCM2,  SRCM3   =       FGEN1   (SRCL0, LSA0    )      ;
 41: LINK    IMRED,  SRCR1,  SRCR2   =       FGEN2   (SRCM0, SRCM1   )      ;
 42: LINK    SRCL0                   =       FQUE0   (SRCM2, SRCR2   )      ;
 43: LINK    OUTE                    =       FQUE4   (DSTM3, SRCM3   )      ;
 44: LINK    SRCM0                   =       FQUE1   (SRCR1, SRCR4   )      ;
 45: LINK    RDAT                    =       FIMRED  (IMRED          )      ;
 46: LINK    DMRE1,  DMRE2           =       FCOP1   (RDATA          )      ;
 47: LINK    SHR8                    =       FMASK1  (DMRE1          )      ;
 48: LINK    TRN2                    =       FMASK2  (DMRE2          )      ;
 49: LINK    TRN1                    =       FSHIFT  (SHR8           )      ;
 50: LINK    GE1                     =       FTRN1   (TRN1           )      ;
 51: LINK    WRTS1,  DEL1            =       FGE1    (GE1            )      ;
 52: LINK    WDAT1,  WADR1           =       FIMWT1  (WRTS1, WRTD1   )      ;
 53: LINK                            =       FDUST   (       ,DEL1   )      ;
 54: LINK    GE2                     =       FTRN2   (TRN2           )      ;
 55: LINK    WRTS2,  SRCR3           =       FGE2    (GE2            )      ;
 56: LINK    WDAT2,  WADR2           =       FIMWT2  (WRTS2, WRTD2   )      ;
 57: LINK    SRCR4,  DSTR4           =       FQNP    (SRCR3, DSTR3   )      ;
 58: ;
 59: LINK    DSTM1,  DSTM2,  DSTM3   =       FGEN3   (DSTL0, LDA0    )      ;
 60: LINK    COPDT,  DSTR1,  DSTR2   =       FGEN4   (DSTM0, DSTM1   )      ;
```

```
LINK    DSTL0                   =       FQUE2   (DSTM2, DSTR2   )
LINK    DSTM0                   =       FQUE3   (DSTR1, DSTR4   )
LINK    GE3,    GE4             =       FCOP2   (COPDT          )
LINK    WRTD1,  DEL2            =       FGE3    (GE3            )
LINK    WRTD2,  DSTR3           =       FGE4    (GE4            )
LINK                            =       FDUST   (       ,DEL2   )
LINK    PEND                    =       FQUTE   (QUTE           )
;
;
;****************************************
;
;       FUNCTION TABLE
;
;----------------------------------------
;
FUNCTION    FIMRED  =   OUT1    (READ,  0)
FUNCTION    FIMWT1  =   OUT2    (WRITE, 20H, 0),QUEUE   (QUEW1, 2)
FUNCTION    FIMWT2  =   OUT2    (WRTIE, 20H, 0),QUEUE   (QUEW2, 2)
FUNCTION    FQUTE   =   OUT1    (HOST,  0)
FUNCTION    FGEN1   =   COPYBK  (1,     1),     CNTGE   (H      )
FUNCTION    FGEN2   =   COPYBK  (1,     L),     CNTGE   (M      )
FUNCTION    FGEN3   =   COPYBK  (1,     2),     CNTGE   (H      )
FUNCTION    FGEN4   =   COPYBK  (1,   2*L),     CNTGE   (M      )
FUNCTION    FMASK1  =   AND     (X      ),      RDCYCS  (ANDB,  1)
FUNCTION    FMASK2  =   AND     (X      ),      RDCYCS  (ANDD,  1)
FUNCTION    FSHIFT  =   SHR     (X      ),      RDCYCS  (DAT08, 1)
FUNCTION    FTRN1   =   RDIDX   (TRNDAT )
FUNCTION    FTRN2   =   RDIDX   (TRNDAT )
FUNCTION    FCOP1   =   COPYM   (2,     0)
FUNCTION    FCOP2   =   COPYM   (2,     1)
FUNCTION    FGE1    =   COPYBK  (2,     0)
FUNCTION    FGE2    =   COPYBK  (2,     0)
FUNCTION    FGE3    =   COPYBK  (2,     L)
FUNCTION    FGE4    =   COPYBK  (2,     L)
FUNCTION    FQNP    =   NOP     (XX     ),      QUEUE   (QUEN,  1)
FUNCTION    FQUE0   =   QUEUE   (QUE0,  1)
FUNCTION    FQUE1   =   QUEUE   (QUE1,  1)
FUNCTION    FQUE2   =   QUEUE   (QUE2,  1)
FUNCTION    FQUE3   =   QUEUE   (QUE3,  1)
FUNCTION    FQUE4   =   QUEUE   (QUE4,  1)
FUNCTION    FDUST   =   COUNT   (1      )
;
;
;****************************************
;
;       DATA MEMORY
;
;----------------------------------------
;
MEMORY  QUE0    =       AREA    (1      )               ;
MEMORY  QUE1    =       AREA    (1      )               ;
MEMORY  QUE2    =       AREA    (1      )               ;
MEMORY  QUE3    =       AREA    (1      )               ;
MEMORY  QUE4    =       AREA    (1      )               ;
MEMORY  QUEN    =       AREA    (1      )               ;
MEMORY  QUEW1   =       AREA    (2      )               ;
MEMORY  QUEW2   =       AREA    (2      )               ;
MEMORY  DAT08   =       8                               ;
MEMORY  ANDB    =       0FF00H                          ;
MEMORY  ANDD    =       000FFH                          ;
```

```
121: MEMORY  TRNDAT  =        00000H,00003H,0000CH,0000FH,
122:                          00030H,00033H,0003CH,0003FH,
123:                          000C0H,000C3H,000CCH,000CFH,
124:                          000F0H,000F3H,000FCH,000FFH,
125:                          00300H,00303H,0030CH,0030FH,
126:                          00330H,00333H,0033CH,0033FH,
127:                          003C0H,003C3H,003CCH,003CFH,
128:                          003F0H,003F3H,003FCH,003FFH,
129:                          00C00H,00C03H,00C0CH,00C0FH,
130:                          00C30H,00C33H,00C3CH,00C3FH,
131:                          00CC0H,00CC3H,00CCCH,00CCFH,
132:                          00CF0H,00CF3H,00CFCH,00CFFH,
133:                          00F00H,00F03H,00F0CH,00F0FH,
134:                          00F30H,00F33H,00F3CH,00F3FH,
135:                          00FC0H,00FC3H,00FCCH,00FCFH,
136:                          00FF0H,00FF3H,00FFCH,00FFFH,
137:                          03000H,03003H,0300CH,0300FH,
138:                          03030H,03033H,0303CH,0303FH,
139:                          030C0H,030C3H,030CCH,030CFH,
140:                          030F0H,030F3H,030FCH,030FFH,
141:                          03300H,03303H,0330CH,0330FH,
142:                          03330H,03333H,0333CH,0333FH,
143:                          033C0H,033C3H,033CCH,033CFH,
144:                          033F0H,033F3H,033FCH,033FFH,
145:                          03C00H,03C03H,03C0CH,03C0FH,
146:                          03C30H,03C33H,03C3CH,03C3FH,
147:                          03CC0H,03CC3H,03CCCH,03CCFH,
148:                          03CF0H,03CF3H,03CFCH,03CFFH,
149:                          03F00H,03F03H,03F0CH,03F0FH,
150:                          03F30H,03F33H,03F3CH,03F3FH,
151:                          03FC0H,03FC3H,03FCCH,03FCFH,
152:                          03FF0H,03FF3H,03FFCH,03FFFH,
153:                          0C000H,0C003H,0C00CH,0C00FH,
154:                          0C030H,0C033H,0C03CH,0C03FH,
155:                          0C0C0H,0C0C3H,0C0CCH,0C0CFH,
156:                          0C0F0H,0C0F3H,0C0FCH,0C0FFH,
157:                          0C300H,0C303H,0C30CH,0C30FH,
158:                          0C330H,0C333H,0C33CH,0C33FH,
159:                          0C3C0H,0C3C3H,0C3CCH,0C3CFH,
160:                          0C3F0H,0C3F3H,0C3FCH,0C3FFH,
161:                          0CC00H,0CC03H,0CC0CH,0CC0FH,
162:                          0CC30H,0CC33H,0CC3CH,0CC3FH,
163:                          0CCC0H,0CCC3H,0CCCCH,0CCCFH,
164:                          0CCF0H,0CCF3H,0CCFCH,0CCFFH,
165:                          0CF00H,0CF03H,0CF0CH,0CF0FH,
166:                          0CF30H,0CF33H,0CF3CH,0CF3FH,
167:                          0CFC0H,0CFC3H,0CFCCH,0CFCFH,
168:                          0CFF0H,0CFF3H,0CFFCH,0CFFFH,
169:                          0F000H,0F003H,0F00CH,0F00FH,
170:                          0F030H,0F033H,0F03CH,0F03FH,
171:                          0F0C0H,0F0C3H,0F0CCH,0F0CFH,
172:                          0F0F0H,0F0F3H,0F0FCH,0F0FFH,
173:                          0F300H,0F303H,0F30CH,0F30FH,
174:                          0F330H,0F333H,0F33CH,0F33FH,
175:                          0F3C0H,0F3C3H,0F3CCH,0F3CFH,
176:                          0F3F0H,0F3F3H,0F3FCH,0F3FFH,
177:                          0FC00H,0FC03H,0FC0CH,0FC0FH,
178:                          0FC30H,0FC33H,0FC3CH,0FC3FH,
179:                          0FCC0H,0FCC3H,0FCCCH,0FCCFH,
180:                          0FCF0H,0FCF3H,0FCFCH,0FCFFH,
```

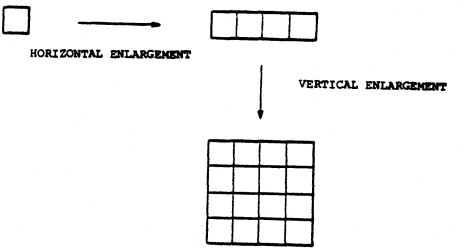
```
181:                                0FF00H,0FF03H,0FF0CH,0FF0FH,
182:                                0FF30H,0FF33H,0FF3CH,0FF3FH,
183:                                0FFC0H,0FFC3H,0FFCCH,0FFCFH,
184:                                0FFF0H,0FFF3H,0FFFCH,0FFFFH
185: ;
186: ;
187: ;****************************************
188: ;
189: ;       START
190: ;
191: ;----------------------------------------
192: ;
193: START
194: ;
195: DATA    EXEC    (IPP,   LSA0,   STARTS  )
196: DATA    EXEC    (IPP,   LDA0,   STARTD  )
197: ;
198: END
```

## 4.5 Simple Quadruple Enlargement

### 4.5.1 Processing Explained

The simple quadruple enlargement method, shown in Figure 4-13, simply expands source image area (SRC) data fourfold horizontally and vertically. Similar to the case of the simple double enlargement method discussed in Section 4.3, in this method four output bits are assigned to each input bit, a fourfold enlargement is performed horizontally, followed by the writing of the data in four contiguous lines.

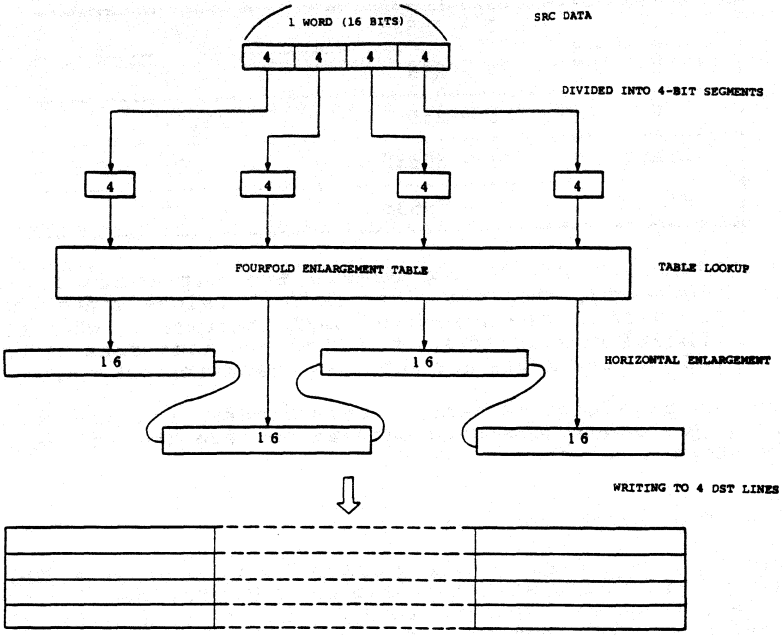
Figure 4-13
Simple Quadruple Enlargement




### 4.5.2 Algorithm

The algorithm employed in this method is similar to that used in the simple double enlargement method, as shown in Figure 4-14. It differs in the fact that, to achieve the fourfold enlargement, each SRC word (16 bits) is segmented into 4 bits, and a fourfold enlargement table is used for each of these 4-bit segments to obtain 4 DST words. First, for a given SRC data word four DST data words representing a horizontal enlargement, are obtained. These four words are then copied to four lines in the vertical direction. This completes processing of an SRC data word.

Figure 4-14
Algorithm for Simple Quadruple Enlargement




### 4.5.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area (SRC) words in horizontal direction
V ... Number of source image area (SRC) lines in vertical direction

<Start-up token-defined parameters>

STARTS ... Source image area (SRC) starting address
STARTD ... Destination image area (DST) starting address

The allowable values of these parameters are indicated in the table below.

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535 | (64) |
| H | 1 - 256 | ( 8) |
| V | 1 - 256 | (128) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (32)* |

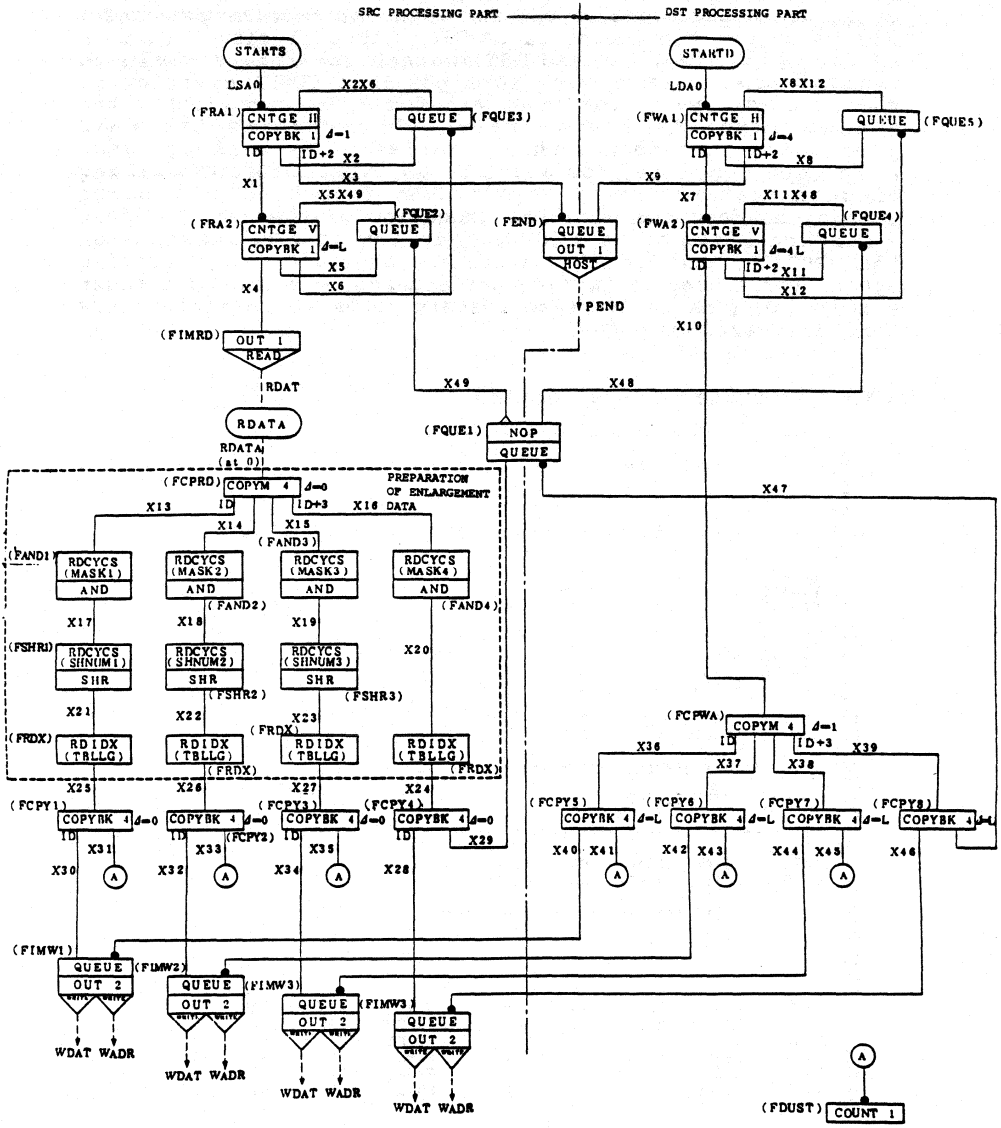
* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting the values of parameters H and M.

### 4.5.4 Flow Graph Explained

Shown in Figure 4-15 is a flow graph for this program. This flow graph is basically the same as that provided for the double enlargement program. However, to the extent that the division of words into 4-bit segments is required to produce fourfold enlargement, this flow graph is more complicated.

**Figure 4.15**
**A Flow Graph of the Simple Quadruple Enlargement Method**



### 4.5.5 Tips on Preparing Flow Graphs

This program uses an FQUE1 node (in the center of Figure 4.15) to synchronize the generation of SRC addresses and DST basic addresses (i.e., the addresses that form the basis of creating 16 addresses in DST for each SRC address generated) and to adjust the amount of uPD7281 internal data. If this was not provided, SRC and DST address generation would be uncoordinated, giving rise to a possible QUEUE overflow in the write-out part. The input arc for FQUE1 constitutes the output from FCPY4 and FCPY8; this permits the next processing to commence when a series of processing tasks have been completed (or about to be completed). The timing of such restarts has a significant impact on processing speeds, as does the parallel execution of the job. Too fast a restart could cause a QUEUE overflow or a GQ overflow on the uPD7281, and too slow a restart degrades the processing speed. Therefore, to improve performance, one should conduct simulation runs to increase restart rates while making sure that no overflows occur.

### 4.5.6 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;        ENLARGE  X4
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8       ;
 8: ;
 9: ;
10: EQUATE  H       =       8       ;
11: EQUATE  V       =       128     ;
12: EQUATE  L       =       64      ;
13: ;
14: ;
15: EQUATE  HOST    =       0       ;
16: EQUATE  READ    =       4       ;
17: EQUATE  WRITE   =       5       ;
18: ;
19: EQUATE  STARTS  =       0       ;
20: EQUATE  STARTD  =       20H     ;
21: ;
22: ;****************************************
23: ;
24: ;       INPUT-OUTPUT
25: ;
26: ;----------------------------------------
27: ;
28: INPUT   LSA0.   LDA0.   RDATA AT 0              ;
29: OUTPUT  RDAT.   WDAT.   WADR.   PEND            ;
30: ;
31: ;****************************************
32: ;
33: ;       LINK TABLE
34: ;
35: ;----------------------------------------
36: ;
37: LINK    X1.     X2.     X3      =       FRA1    (X2X6.  LSA0    )       ;
38: LINK    X4.     X5.     X6      =       FRA2    (X5X49. X1      )       ;
39: LINK    X2X6                    =       FQUE3   (X2.    X6      )       ;
40: LINK    PEND                    =       FEND    (X9.    X3      )       ;
41: LINK    RDAT                    =       FIMRD   (X4             )       ;
42: LINK    X5X49                   =       FQUE2   (X5.    X49     )       ;
43: ;
```

```
44: LINK    X13.  X14.  X15.  X16   =      FCPRD   (RDATA          )  ;
45: LINK    X17                     =      FAND1   (X13            )  ;
46: LINK    X18                     =      FAND2   (X14            )  ;
47: LINK    X19                     =      FAND3   (X15            )  ;
48: LINK    X20                     =      FAND4   (X16            )  ;
49: LINK    X21                     =      FSHR1   (X17            )  ;
50: LINK    X22                     =      FSHR2   (X18            )  ;
51: LINK    X23                     =      FSHR3   (X19            )  ;
52: LINK    X24                     =      FRDX    (X20            )  ;
53: LINK    X25                     =      FRDX    (X21            )  ;
54: LINK    X26                     =      FRDX    (X22            )  ;
55: LINK    X27                     =      FRDX    (X23            )  ;
56: LINK    X28.    X29             =      FCPY4   (X24            )  ;
57: LINK    X30.    X31             =      FCPY1   (X25            )  ;
58: LINK    X32.    X33             =      FCPY2   (X26            )  ;
59: LINK    X34.    X35             =      FCPY3   (X27            )  ;
60: LINK    WDAT.   WADR            =      FIMW4   (X28.   X46     )  ;
61: LINK    WDAT.   WADR            =      FIMW1   (X30.   X40     )  ;
62: LINK    WDAT.   WADR            =      FIMW2   (X32.   X42     )  ;
63: LINK    WDAT.   WADR            =      FIMW3   (X34.   X44     )  ;
64: LINK                            =      FDUST   (       ,X31    )  ;
65: LINK                            =      FDUST   (       ,X33    )  ;
66: LINK                            =      FDUST   (       ,X35    )  ;
67: ;
68: LINK    X7.     X8.     X9      =      FWA1    (X8X12, LDA0    )  ;
69: LINK    X10.    X11.    X12     =      FWA2    (X11X48, X7     )  ;
70: LINK    X8X12                   =      FQUE5   (X8.    X12     )  ;
71: LINK    X11X48                  =      FQUE4   (X11.   X48     )  ;
72: LINK    X36.  X37.  X38.  X39   =      FCPWA   (X10            )  ;
73: LINK    X40.    X41             =      FCPY5   (X36            )  ;
74: LINK    X42.    X43             =      FCPY6   (X37            )  ;
75: LINK    X44.    X45             =      FCPY7   (X38            )  ;
76: LINK    X46.    X47             =      FCPY8   (X39            )  ;
77: LINK    X48.    X49             =      FQUE1   (X29    .X47    )  ;
78: LINK                            =      FDUST   (       .X41    )  ;
79: LINK                            =      FDUST   (       .X43    )  ;
80: LINK                            =      FDUST   (       .X45    )  ;
81: ;
82: ;**************************************
83: ;
84: ;       FUNCTION TABLE
85: ;
86: ;--------------------------------------
87: ;
88: FUNCTION    FRA1    =   COPYBK (1.     1),     CNTGE   (H          )  ;
89: FUNCTION    FQUE3   =   QUEUE  (DQUE3. 1)                              ;
90: FUNCTION    FRA2    =   COPYBK (1.     L),     CNTGE   (V          )  ;
91: FUNCTION    FQUE2   =   QUEUE  (DQUE2. 1)                              ;
92: FUNCTION    FIMRD   =   OUT1   (READ.  0)                              ;
93: FUNCTION    FEND    =   OUT1   (HOST.  0),     QUEUE   (ENDQUE. 1)     ;
94: ;
95: FUNCTION    FCPRD   =   COPYM  (4.     0)                              ;
96: FUNCTION    FAND1   =   AND.                   RDCYCS  (MASK1.  1)     ;
97: FUNCTION    FAND2   =   AND.                   RDCYCS  (MASK2.  1)     ;
98: FUNCTION    FAND3   =   AND.                   RDCYCS  (MASK3.  1)     ;
99: FUNCTION    FAND4   =   AND.                   RDCYCS  (MASK4.  1)     ;
100: FUNCTION   FSHR1   =   SHR.                   RDCYCS  (SHNUM1. 1)     ;
101: FUNCTION   FSHR2   =   SHR.                   RDCYCS  (SHNUM2. 1)     ;
102: FUNCTION   FSHR3   =   SHR.                   RDCYCS  (SHNUM3. 1)     ;
103: FUNCTION   FRDX    =   RDIDX  (TBLLG   )                              ;
104: FUNCTION   FCPY1   =   COPYBK (4.     0)                              ;
105: FUNCTION   FCPY2   =   COPYBK (4.     0)                              ;
106: FUNCTION   FCPY3   =   COPYBK (4.     0)                              ;
107: FUNCTION   FCPY4   =   COPYBK (4.     0)                              ;
108: FUNCTION   FIMW1   =   OUT2 (WRITE. 20H. 0),  QUEUE   (QUEWR1. 4)     ;
109: FUNCTION   FIMW2   =   OUT2 (WRITE. 20H. 0),  QUEUE   (QUEWR2. 4)     ;
110: FUNCTION   FIMW3   =   OUT2 (WRITE. 20H. 0),  QUEUE   (QUEWR3. 4)     ;
```

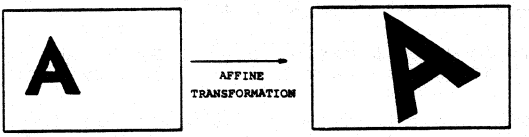
```
111: FUNCTION    FIMW4   =    OUT2 (WRITE, 20H, 0),   QUEUE  (QUEWR4, 4)      ;
112: FUNCTION    FDUST   =    COUNT  (1        )                              ;
113: ;
114: FUNCTION    FWA1    =    COPYBK (1,     4),      CNTGE  (H         )     ;
115: FUNCTION    FWA2    =    COPYBK (1,   4*L),      CNTGE  (V         )     ;
116: FUNCTION    FQUE5   =    QUEUE  (DQUE5, 1)                               ;
117: FUNCTION    FQUE4   =    QUEUE  (DQUE4, 1)                               ;
118: FUNCTION    FQUE1   =    NOP    (XY       ),     QUEUE  (DQUE1, 1)      ;
119: ;
120: FUNCTION    FCPWA   =    COPYM  (4,     1)                               ;
121: FUNCTION    FCPY5   =    COPYBK (4,     L)                               ;
122: FUNCTION    FCPY6   =    COPYBK (4,     L)                               ;
123: FUNCTION    FCPY7   =    COPYBK (4,     L)                               ;
124: FUNCTION    FCPY8   =    COPYBK (4,     L)                               ;
125: ;
126: ;*****************************************
127: ;
128: ;       DATA MEMORY
129: ;
130: ;-----------------------------------------
131: ;
132: MEMORY  DQUE1   =       AREA    (1       )       ;
133: MEMORY  DQUE2   =       AREA    (1       )       ;
134: MEMORY  DQUE3   =       AREA    (1       )       ;
135: MEMORY  DQUE4   =       AREA    (1       )       ;
136: MEMORY  DQUE5   =       AREA    (1       )       ;
137: MEMORY  ENDQUE  =       AREA    (1       )       ;
138: MEMORY  MASK1   =       0F000H                   ;
139: MEMORY  MASK2   =       0F00H                    ;
140: MEMORY  MASK3   =       00F0H                    ;
141: MEMORY  MASK4   =       000FH                    ;
142: MEMORY  SHNUM1  =       12                       ;
143: MEMORY  SHNUM2  =       8                        ;
144: MEMORY  SHNUM3  =       4                        ;
145: MEMORY  TBLLG.  =       00000H,0000FH,000F0H,000FFH,00F00H,00F0FH,
146:                         00FF0H,00FFFH,0F000H,0F00FH,0F0F0H,0F0FFH,
147:                         0FF00H,0FF0FH,0FFF0H,0FFFFH      ;
148: MEMORY  QUEWR1  =       AREA    (4       )       ;
149: MEMORY  QUEWR2  =       AREA    (4       )       ;
150: MEMORY  QUEWR3  =       AREA    (4       )       ;
151: MEMORY  QUEWR4  =       AREA    (4       )       ;
152: ;
153: ;*****************************************
154: ;
155: ;       START
156: ;
157: ;-----------------------------------------
158: ;
159: START                                            ;
160: ;
161: ;
162: DATA    EXEC    (IPP,   LSA0,   STARTS  )       ;
163: DATA    EXEC    (IPP,   LDA0,   STARTD  )       ;
164: ;
165: END                                              ;
```

Chapter 5

# Affine Transformation

## 5.1 Processing Explained

The affine transformation provides a versatile means of performing image transformations. It can be used for screen enlargement, reduction, rotation, and displacement simultaneously.

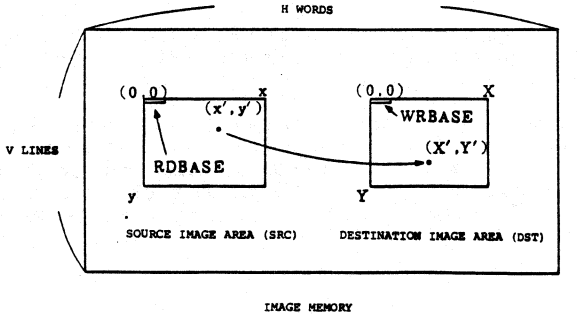



However, this technique takes longer to execute than a program utilizing double enlargement, quadruple enlargement, 90 degree rotation, or some other specific algorithms. Therefore, the affine transformation should be used only when required by the application.

## 5.2 Algorithm

<Logical coordinates and physical addresses>

Figure 5-1
Logical Coordinates and Physical Addresses



An image memory may be thought of as being comprised of H words horizontally and V lines vertically (i.e., 16 x H horizontal dots and V vertical dots). In this image memory are two image areas (source image area "SRC" and destination image area "DST). These graphic image areas have logical coordinates (x,y) and (X,Y), respectively, as shown in Figure 5-1. Further, the origin (0,0) of the logical coordinates (x,y) corresponds with the MSB of physical address RD BASE, and the origin (0,0) of the logical coordinates (X,Y) corresponds with the MSB of physical address WR BASE*. In this case the coordinates (x',y') of an arbitrary point on SRC are physically:

$$\begin{aligned} &\text{Physical address} \quad RDBASE + y' \cdot H + [\, x' / 16 \,] \\ &\text{Bit position} \quad 15 - R(\, x' / 16 \,) \end{aligned} \tag{1}$$

where [a/b] means the division of a by b, and
R(a/b) means the remainder of the division of a by b.

* : The MSB correspondence differs for the NEC graphic display controllers uPD7220 and uPD7220A.

Similarly, the coordinates (X',Y') correspond with:

$$\begin{aligned} &\text{Physical address} \quad WRBASE + Y' \cdot H + [\, X' / 16 \,] \\ &\text{Bit position} \quad 15 - R(\, X' / 16 \,) \end{aligned} \tag{2}$$

In the affine transformation used in this program, the DST logical coordinates (X',Y') and the SRC logical coordinates (x',y') are related to each other by the following formulas:

$$\begin{aligned} x' &= aX' + bY' + c \\ y' &= dX' + eY' + f \end{aligned} \tag{3}$$

where a, b, c, d, e, and f are parameters that determine whether a given transformation is an enlargement, reduction, rotation, or displacement.

Note: The affine transformation formulas used in this program specify the coordinates before transformation in terms of coordinates after transformation.

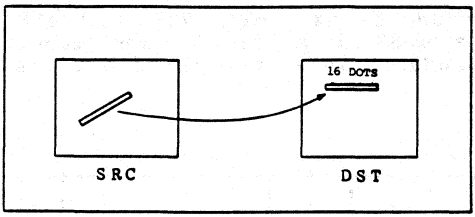
In this program the logical coordinates (X',Y') are allowed to vary from the point (0,0) to the point (max.(X),max.(Y)). The logical coordinates (x',y') corresponding to each of the coordinates are calculated, and the bits for these coordinates are read from SRC and written to DST. Information is written to DST in word units (16 bits).

<Process Flow>

This program assumes that a DST is a 256 x 256 dots word boundary area.

First, (X',Y') is set to (0,0). Then the coordinates (x',y') corresponding to the variation of (X',Y') from the origin to (256,0) are calculated. However, because all 256 values cannot be calculated at once, 256 is divided into 16 segments and 16 dots are calculated at a time. Since Y does not vary during this process, the amount of computation required is cut down by calculating bY' + c and eY' + f in Equation (3) only once for the processing of the first line and by adding these values to aX' and dX'.

Actual physical addresses and bit positions are calculated for each 16 sets of (x',y') values obtained by calculations on one word (16 bits) of DST. SRC data are read using these physical addresses, and the desired bits are extracted using the values for bit positions.

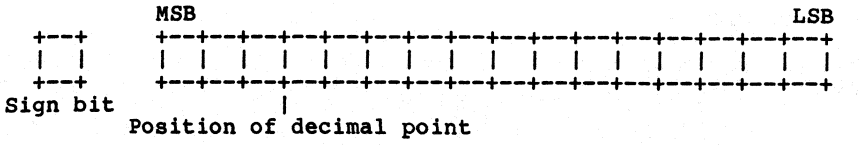



The 16-bit data obtained in this manner are packed into a word and written in DST. Since one DST line has 256 dots, writing 16 words completes the processing of one line. This process is repeated line by line for as many lines as there are lines in DST to complete the processing of one screen image.

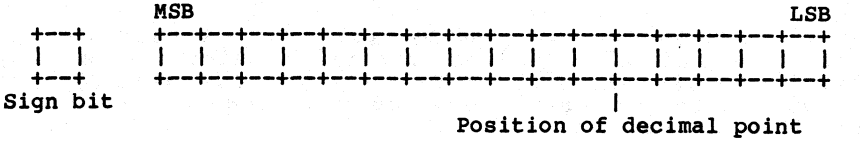
<Precision of Computation>

In multiplying 16 bits by 16 bits in the uPD7281, it is possible to select the high-order 16 bits and/or low-order 16 bits from the results. This program selects the high-order 16 bits only. For this reason the following scheme is employed in calculating the values of parameters a, b, c, d, e, and f, and coordinates (X',Y') and (x',y'):

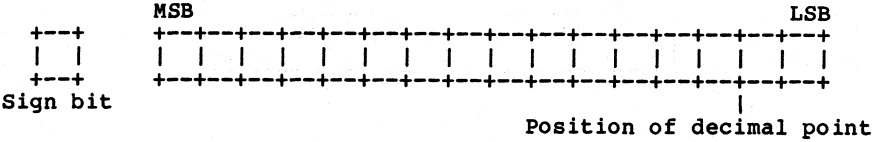
(a) The parameters a, b, d, and e have a 16-bit precision (17 bits including the sign bit) with the decimal point occurring in the 14th bit position.



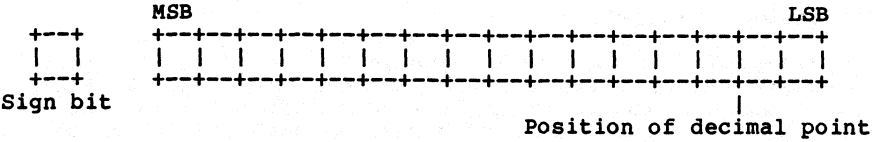

(b) Logically, the coordinates (X',Y') are allowed to increment by one at a time over DST, but the placement of the decimal point for X' and Y' at the sixth bit position from the LSB actually makes them increment by 32 at a time.




(c) Since the products aX', bY', dX', and eY' represent the high-order 16 bits of 32-bit multiplication products, the decimal point in these products is at the third bit position from the LSB.




(d) Consequently, to be consistent with the precision of their preceding terms, the parameters c and f assume values with the decimal point set at the third bit position from the LSB.




(e) As a result, the values of (x',y') have the decimal point set at the third bit position from the LSB. Therefore, a given value is reduced to an integer by performing a 2-bit right shift (note).

Note: (x',y') values have a precision of two bits below the decimal point. Therefore, if a given value was 0.75 or 1.75, retaining a value of 1 or 2 instead of rounding off the bits below the decimal point through bit shifting would result in a more accurate screen image. For this reason, 0.5 is added to the values of x' and y' in this program. After this, a 2-bit right shift is performed. The addition of 0.5 to these parameters is done by adding 0.5 to parameters c and f.

## 5.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of destination image area (DST) words in horizontal direction
V ... Number of destination image area (DST) lines in vertical direction

<Start-up token-defined parameters>

SETA ..... Affine transformation parameter a
SETB ..... Affine transformation parameter b
SETC ..... Affine transformation parameter c
SETD ..... Affine transformation parameter d
SETE ..... Affine transformation parameter e
SETF ..... Affine transformation parameter f
RDBASE ... Source image area (SRC) starting address
WRBASE ... Destination image area (DST) starting address

The allowable values of these parameters are indicated in the table below.

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535 | (64) |
| H | 1 - 256 | (16) |
| V | 1 - 256 | (256) |
| SETA | -7.999 - +7.999 | ( 0)* |
| SETB | -7.999 - +7.999 | ( 0)* |
| SETC | -2047.9 - +2047.9 | ( 0)* |
| SETD | -7.999 - +7.999 | ( 0)* |
| SETE | -7.999 - +7.999 | ( 0)* |
| SETF | -2047.9 - +2047.9 | ( 0)* |
| RDBASE | 0 - 65535 | ( 0)* |
| WRBASE | 0 - 65535 | (32)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting the values for parameters H and V.

<Input Tokens>

This program provides ten input tokens. The first and last of these tokens require special explanations.

The first token is a command reset. It clears the ACC register of the PU in the uPD7281 to 0. Bear in mind that, when this token is input into the uPD7281, all programs in execution at the time are erased. Make sure, when inputting this token, that no program is running within the uPD7281.

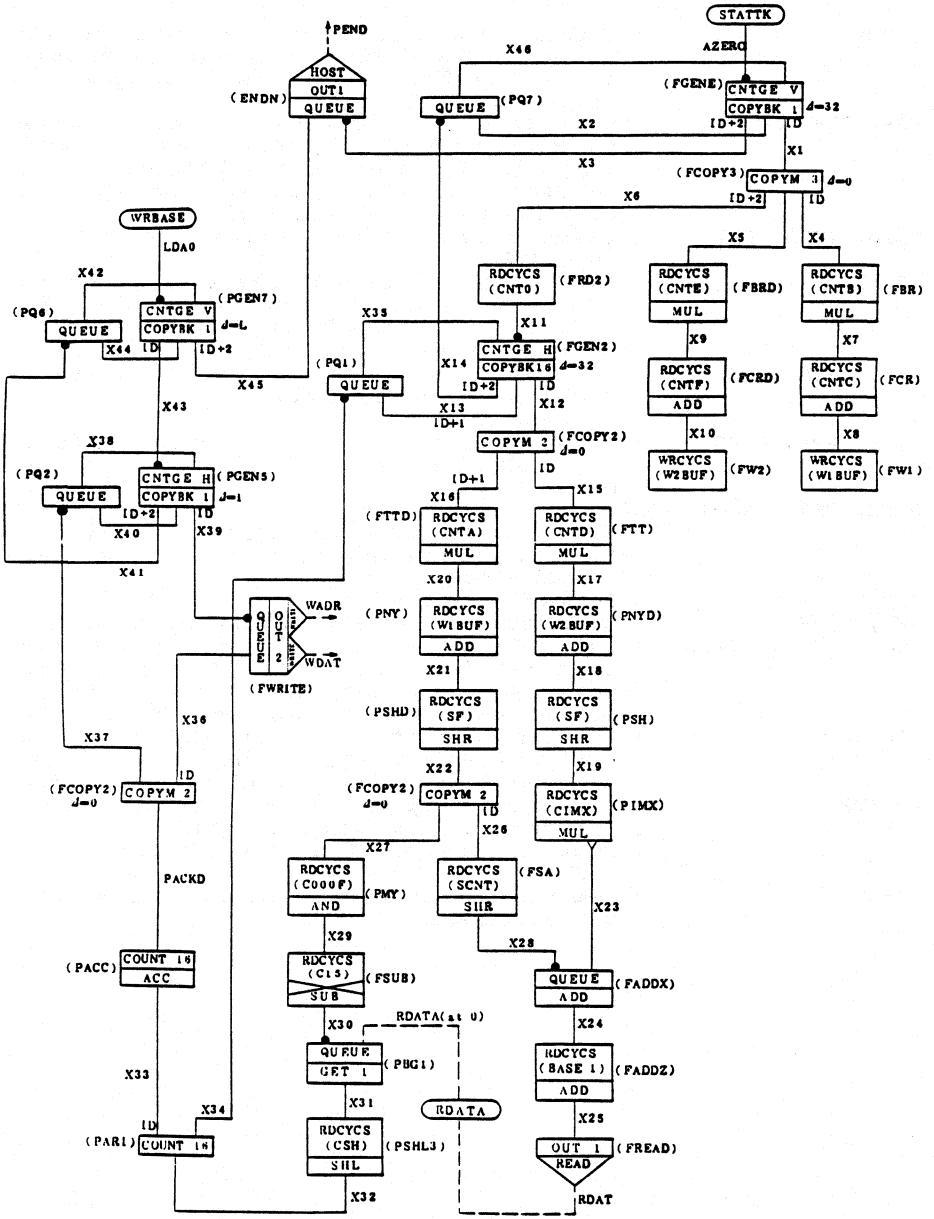
The last token is a program start-up token. When this token is input, the uPD7281 starts operation. Since the data value for this token is also used as the line counter for the number of DST lines it should always be input as 0.

## 5.4 Flow Graph Explained

<Explanation of Main Nodes>

| | |
|---|---|
| FGENE | : Controls the counting of the number of DST lines. If the CNTGE instruction parameter is set to 256, 256 lines of images are processed. |
| FBR, FCR, FW1 | : Calculates bY' + c and writes the results to W1BUF. This value needs to be calculated only once for each value of Y', and it can be read from W1BUF at any time when an addition to aX' is performed. |
| FBRD, FCRD, FW2 | : Calculates eY' + f and writes the results to W2BUF. |
| FGEN2 | : Increments X' from 0 to max.(X) and outputs the results as a token. Since not all values for 0 through max.(X) can be output at once, 16 values are output at a time. |
| FTT, PNYD, PSH | : Calculates dX' + eY' + f and performs a right shift to round off values below the decimal point. |
| FTTD, PNY, PSHD | : Calculates aX' + by' + c and performs a right shift to round off values below the decimal point. |
| PIMX, FSA, FADDX, FADDZ | : From a computed value (x',y'), calculates the physical address of the given coordinates. |
| FREAD | : Reads SRC data. |
| PMY, FSUB | : Calculates bit positions from x' values. |
| PBG1, PSHL3, PACC | : Extracts bits from the bit positions calculated by PMY and FSUB from data that were read, and sets these bits in their proper positions in a word of DST data. |
| PAR1 | : Detects completion of the setting of one word of DST data (the arrival of 16 tokens) and starts the processing of the next word. |
| FWRITE | : Writes DST data to the DST address. |
| PGEN7 | : Creates starting addresses for DST lines, starting from the DST starting address WR BASE. |
| PGEN5 | : Creates addresses for one line among the DST addresses. |

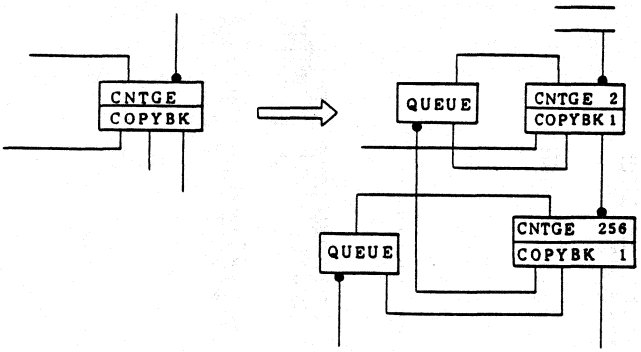
Figure 5.2
An Affine Transformation Flow Graph



## 5.5 Tips on Writing Flow Graphs

This program assumes that a DST is 256 dots by 256 dots. If a 512-dot by 512-dot DST is desired, you must use loop-forming nodes (as shown in Figure 5-3) in addition to the nodes FGENE and PGEN7 and change the GNTGE instruction parameter in PGEN5 to 512/16=32.

Figure 5-2
A Method for Enlarging a Processing Area




The CNTGE instruction parameter can not be allowed to exceed 256. To accommodate more than 256 values the node addition described above is required.

## 5.6 Assembler Source Listing

```
 1: ;*****************************************
 2: ;
 3: ;       AFFINE TRANSFORMATION
 4: ;
 5: ;-----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  H       =       16              ;
10: EQUATE  L       =       64              ;
11: EQUATE  V       =       256             ;
12: ;
13: EQUATE  HOST    =       0               ;
14: EQUATE  READ    =       4               ;
15: EQUATE  WRITE   =       5               ;
16: ;
17: EQUATE  RDBASE  =       0               ;
18: EQUATE  WRBASE  =       32              ;
19: ;
```

```
20: ;****************************************
21: ;
22: ;       INPUT-OUTPUT
23: ;
24: ;----------------------------------------
25: ;
26: INPUT   LSA0.   LDA0.   RDATA AT 0                                    ;
27: ;
28: INPUT   SETA.   SETB.   SETC.   SETD.   SETE.   SETF.   AZERO        ;
29: ;
30: OUTPUT  RDAT.   WDAT.   WADR.   PEND                                 ;
31: ;
32: ;****************************************
33: ;
34: ;       LINK TABLE
35: ;
36: ;----------------------------------------
37: ;
38: LINK                              =       FSETA   (SETA           )   ;
39: LINK                              =       FSETB   (SETB           )   ;
40: LINK                              =       FSETC   (SETC           )   ;
41: LINK                              =       FSETD   (SETD           )   ;
42: LINK                              =       FSETE   (SETE           )   ;
43: LINK                              =       FSETF   (SETF           )   ;
44: LINK                              =       FSETRB  (LSA0           )   ;
45: ;
46: LINK    X1.     X2.     X3        =       FGENE   (X46    .AZERO  )   ;
47: LINK    X4.     X5.     X6        =       FCOPY3  (X1             )   ;
48: LINK    X7                        =       FBR     (X4             )   ;
49: LINK    X8                        =       FCR     (X7             )   ;
50: LINK                              =       FW1     (X8             )   ;
51: LINK    X9                        =       FBRD    (X5             )   ;
52: LINK    X10                       =       FCRD    (X9             )   ;
53: LINK                              =       FW2     (X10            )   ;
54: LINK    X11                       =       FRD2    (X6             )   ;
55: LINK    X12.    X13.    X14       =       FGEN2   (X35    .X11    )   ;
56: LINK    X15.    X16               =       FCOPY2  (X12            )   ;
57: LINK    X17                       =       FTT     (X15            )   ;
58: LINK    X18                       =       PNYD    (X17            )   ;
59: LINK    X19                       =       PSH     (X18            )   ;
60: LINK    X20                       =       FTTD    (X16            )   ;
61: LINK    X21                       =       PNY     (X20            )   ;
62: LINK    X22                       =       PSHD    (X21            )   ;
63: LINK    X23                       =       PINX    (X19            )   ;
64: LINK    X24                       =       FADDX   (X23    .X28    )   ;
65: LINK    X25                       =       FADDZ   (X24            )   ;
66: LINK    RDAT                      =       FREAD   (X25            )   ;
67: LINK    X26.    X27               =       FCOPY2  (X22            )   ;
68: LINK    X28                       =       FSA     (X26            )   ;
69: LINK    X29                       =       PMY     (X27            )   ;
70: LINK    X30                       =       FSUB    (X29            )   ;
71: LINK    X31                       =       PBG1    (RDATA  .X30    )   ;
72: LINK    X32                       =       PSHL3   (X31            )   ;
73: LINK    X33.    X34               =       PARI    (X32            )   ;
74: LINK    X35                       =       PQ1     (X13    .X34    )   ;
75: LINK    PACKD                     =       PACC    (X33            )   ;
76: LINK    X36.    X37               =       FCOPY2  (PACKD          )   ;
77: LINK    WDAT.   WADR              =       FWRITE  (X36    .X39    )   ;
78: LINK    X38                       =       PQ2     (X40    .X37    )   ;
79: LINK    X39.    X40.    X41       =       PGEN5   (X38    .X43    )   ;
80: LINK    X42                       =       PQ6     (X44    .X41    )   ;
81: LINK    X43.    X44.    X45       =       PGEN7   (X42    .LDA0   )   ;
82: LINK    PEND                      =       ENDN    (X45    .X3     )   ;
83: LINK    X46                       =       PQ7     (X2     .X14    )   ;
84: ;
85: ;****************************************
86: ;
87: ;       FUNCTION TABLE
88: ;
89: ;----------------------------------------
90: ;
```

```
 91: FUNCTION    FGENE   =   COPYBK  (1,    32),     CNTGE   (V       )  ;
 92: FUNCTION    FCOPY3  =   COPYM   (3,     0)                          ;
 93: FUNCTION    FSR     =   MUL.                     RDCYCS  (CNTB.  1)  ;
 94: FUNCTION    FCR     =   ADD.                     RDCYCS  (CNTC.  1)  ;
 95: FUNCTION    FW2     =   WRCYCS  (W2BUF, 1)                           ;
 96: FUNCTION    FBRD    =   MUL.                     RDCYCS  (CNTE.  1)  ;
 97: FUNCTION    FRD2    =   RDCYCS  (CNT0 , 1)                           ;
 98: FUNCTION    FCRD    =   ADD.                     RDCYCS  (CNTF.  1)  ;
 99: FUNCTION    FW1     =   WRCYCS  (W1BUF, 1)                           ;
100: FUNCTION    FCOPY2  =   COPYM   (2,     0)                          ;
101: FUNCTION    FGEN2   =   COPYBK  (16,   32),      CNTGE   (H       )  ;
102: FUNCTION    FTT     =   MUL.                     RDCYCS  (CNTD.  1)  ;
103: FUNCTION    PNY     =   ADD.                     RDCYCS  (W1BUF. 1)  ;
104: FUNCTION    PSH     =   SHR.                     RDCYCS  (SF.    1)  ;
105: FUNCTION    FTTD    =   MUL.                     RDCYCS  (CNTA.  1)  ;
106: FUNCTION    PNYD    =   ADD.                     RDCYCS  (W2BUF. 1)  ;
107: FUNCTION    PSHD    =   SHR.                     RDCYCS  (SF.    1)  ;
108: FUNCTION    PIMX    =   MUL     (Y       ),      RDCYCS  (CIMX.  1)  ;
109: FUNCTION    FADDX   =   ADD.                     QUEUE   (QUE1. 16)  ;
110: FUNCTION    FADDZ   =   ADD.                     RDCYCS  (BASE1, 1)  ;
111: FUNCTION    FREAD   =   OUT1    (READ.  0)                           ;
112: FUNCTION    FSA     =   SHR.                     RDCYCS  (SCNT.  1)  ;
113: FUNCTION    PMY     =   AND.                     RDCYCS  (C000F. 1)  ;
114: FUNCTION    FSUB    =   SUB     (XCH     ),      RDCYCS  (C15.   1)  ;
115: FUNCTION    PSG1    =   GET1.                    QUEUE   (BSF.  16)  ;
116: FUNCTION    PSHL3   =   SHL.                     RDCYCS  (CSH.  16)  ;
117: FUNCTION    PAR1    =   COUNT   (16      )                           ;
118: FUNCTION    PQ1     =   QUEUE   (QUE3,  1)                           ;
119: FUNCTION    PACC    =   ACC.                     COUNT   (16      )  ;
120: FUNCTION    FWRITE  =   OUT2    (WRITE, 20H, 0), QUEUE   (QUE4. 16)  ;
121: FUNCTION    PQ2     =   QUEUE   (QUE5,  1)                           ;
122: FUNCTION    PGEN5   =   COPYBK  (1,     1),      CNTGE   (H       )  ;
123: FUNCTION    PQ6     =   QUEUE   (QUE8,  1)                           ;
124: FUNCTION    PGEN7   =   COPYBK  (1,     L),      CNTGE   (V       )  ;
125: FUNCTION    ENDN    =   OUT1    (HOST,  0),      QUEUE   (QUE9,  1)  ;
126: FUNCTION    PQ7     =   QUEUE   (QUE11, 1)                           ;
127: FUNCTION    FSETA   =   WRCYCS  (CNTA,  1)                           ;
128: FUNCTION    FSETB   =   WRCYCS  (CNTB,  1)                           ;
129: FUNCTION    FSETC   =   WRCYCS  (CNTC,  1)                           ;
130: FUNCTION    FSETD   =   WRCYCS  (CNTD,  1)                           ;
131: FUNCTION    FSETE   =   WRCYCS  (CNTE,  1)                           ;
132: FUNCTION    FSETF   =   WRCYCS  (CNTF,  1)                           ;
133: FUNCTION    FSETRB  =   WRCYCS  (BASE1, 1)                           ;
134: ;
135: ;****************************************
136: ;
137: ;       DATA MEMORY
138: ;
139: ;----------------------------------------
140: ;
141: MEMORY  CNTA    =       0                            ;
142: MEMORY  CNTB    =       2000H                        ;
143: MEMORY  CNTC    =       0                            ;
144: MEMORY  CNTD    =       2000H                        ;
145: MEMORY  CNTE    =       0                            ;
146: MEMORY  CNTF    =       0                            ;
147: MEMORY  W1BUF   =       AREA    (1       )           ;
148: MEMORY  W2BUF   =       AREA    (1       )           ;
149: MEMORY  CNT0    =       0                            ;
150: MEMORY  SF      =       2                            ;
151: MEMORY  CIMX    =       1                            ;
152: MEMORY  BASE1   =       0                            ;
153: MEMORY  SCNT    =       4                            ;
154: MEMORY  C000F   =       000FH                        ;
155: MEMORY  C15     =       15                           ;
156: MEMORY  BSF     =       AREA    (16      )           ;
157: MEMORY  CSH     =       0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15   ;
158: MEMORY  QUE1    =       AREA    (16      )           ;
159: MEMORY  QUE3    =       1                            ;
160: MEMORY  QUE4    =       AREA    (16      )           ;
161: MEMORY  QUE5    =       1                            ;
162: MEMORY  QUE8    =       AREA    (1       )           ;
163: MEMORY  QUE9    =       AREA    (1       )           ;
164: MEMORY  QUE11   =       AREA    (1       )           ;
165: ;
```

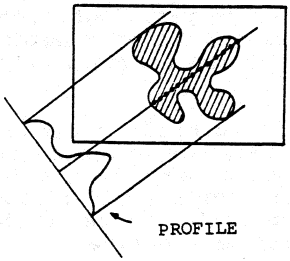
```
166: ;****************************************
167: ;
168: ;       START
169: ;
170: ;----------------------------------------
171: ;
172: START                                                    ;
173: ;
174: DATA    CRESET  (IPP                          )          ;
175: DATA    EXEC    (IPP,   SETA,   2000H        )          ;
176: DATA    EXEC    (IPP,   SETB,   0000H        )          ;
177: DATA    EXEC    (IPP,   SETC,   0000H        )          ;
178: DATA    EXEC    (IPP,   SETD,   0000H        )          ;
179: DATA    EXEC    (IPP,   SETE,   2000H        )          ;
180: DATA    EXEC    (IPP,   SETF,   0000H        )          ;
181: DATA    EXEC    (IPP,   LSA0,   RDBASE       )          ;
182: DATA    EXEC    (IPP,   LDA0,   WRBASE       )          ;
183: DATA    EXEC    (IPP,   AZERO,  0            )          ;
184: ;
185: END                                                      ;
```

# Chapter 6

# Profiling

Profiling is the process of projecting a binary image in a certain direction, counting the number of pixel having the value of "1", and forming a histogram of the results. Profiling is illustrated in Figure 6-1.

Figure 6-1
Profiling

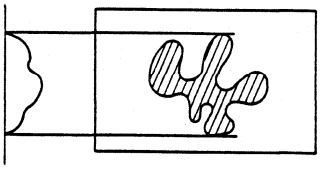



## 6.1 Horizontal Profiling

### 6.1.1 Processing Explained

A horizontal profile is a histogram that is obtained by projecting a binary image in the horizontal direction, as shown in Figure 6-2.

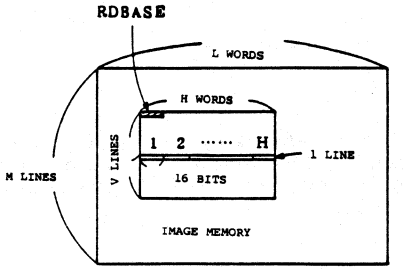
Figure 6-2
Horizontal Profiling



### 6.1.2 Algorithm

Suppose that an image memory is comprised of L words horizontally and M lines vertically, as shown in Figure 6-3.

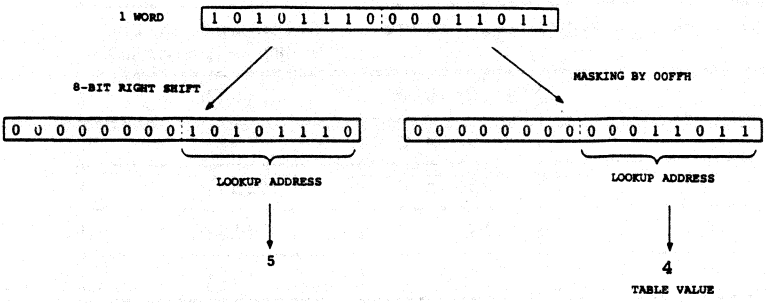
Figure 6-3
Definition of a Source Image Area




The following explains the method of creating a horizontal profile for a source image area of H words horizontally and V lines vertically, with the origin at the physical address RD BASE.

Consider the horizontal profile for one line. Since the line consists of H words, the program reads these H words one by one from the source image area and counts the number of "1" bits existing in each word. The counts are added for all H words. Let the resulting value be the histogram value (the horizontal profile) of the line.

The "1" bits in each word are counted by means of a table lookup within the DM of the uPD7281. The uPD7281 RDIDX instruction in the AG&FC is a table lookup instruction. This instruction is capable of specifying an index address in eight bits. Consequently, the word (16 bits) read from the image memory is divided into 2 parts of 8 bits each, and each of these values is used as an index to the lookup table. If the numbers of "1" bits in the corresponding addresses are stored in the table beforehand, the number of "1" bits in a word can be counted in an efficient manner by cumulative addition of the values obtained from the table lookup process.




To accumulate the looked-up values, the ACC instruction in the PU of the uPD7281 is used. This requires that the ACC register be set to 0 by the reset command before start of the program. Also, because each word is divided into two segments, the number of additions performed for n words will be 2n.

Since reading information from the image memory one word at a time would be an inefficient use of the uPD7281's internal pipelined ring, the program reads Q words of data continuously. Therefore, "H words" used in this program must a multiple of Q. Further, the cumulative values obtained through the addition operations are written in the image memory contiguously from a specified address.

### 6.1.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area (SRC) words in horizontal direction
V ... Number of source image area (SRC) lines in vertical direction
Q ... Number of continuous reads from the image memory (horizontal direction)
R ... Number of segments read from the image memory (vertical direction)

<Start-up token-defined parameters>

STARTS ... Source image area (SRC) starting address
STARTD ... Starting address of a profile storage

The allowable values of these parameters are indicated in the table below.

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535/R | (64) |
| H | Q - 256 x Q | (32) |
| V | R - 256 x R | (256) |
| Q | 1 - 16 | (16) |
| R | 1 - 256 | (32) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (1FFH)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting values of parameters H and V.

### 6.1.4 Flow Graph Explained

<Explanation of main nodes>

FGEN1, FGEN2 : Generates the starting addresses of lines on which profiling computations are performed based on the SRC starting address STARTS. Since only up to 256 can be counted by one CNTGE instruction, CNTGE instruction are used in a double loop in this program to permit a count exceeding 256.

FGEN5 : Generates data read addresses for a line in the horizontal direction. For improved internal processing of the uPD7281, this node generates addresses for Q contiguous words.

FOUTR : Reads SRC data.

FCOP1, FAND1, FSHR1: Makes two copies of the SRC data read and extracts the high-order (FSHR1) and low-order (FAND1) eight bits.

FTAB1 : Reads the uPD7281's DM by using the 8-bit data extracted in FAND1 and FSHR1 as index addresses. In other words,

this node obtains the number of "1"s in the 8-bit data by using a table lookup.

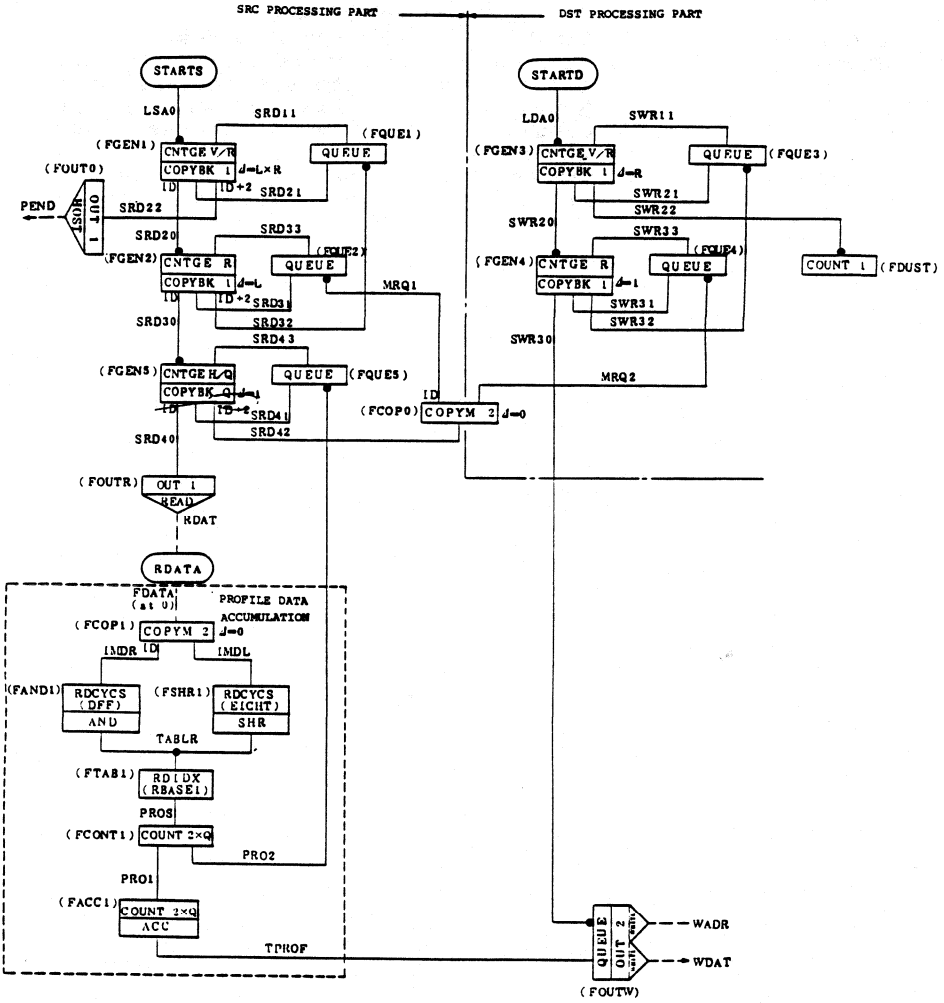
FCONT1 : Detects completion of the operation of counting Q words and starts up FGEN5. When 2 x Q tokens arrive in this node, FCONT1 creates one token for starting the next processing step.

FACC1 : Cumulatively adds count values.

FOUTW : Matches the cumulative count value for a line and the storage address of a horizontal profile and writes the cumulative count value in the image memory.

FGEN3, FGEN4 : Creates storage addresses based on the profile storage starting address STARTD.

## Figure 6.4
## A Flow Graph of a Horizontal Profile



## 6.1.5 Assembler Source Listing

```
1: ;****************************************
2: ;
3: ;       HORIZONTAL PROFILE
4: ;
5: ;----------------------------------------
6: ;
7: MODULE  IPP     =       8               ;
8: ;
9: EQUATE  L       =       64              ;
10: EQUATE  H       =       32              ;
11: EQUATE  Q       =       16              ;
12: EQUATE  R       =       32              ;
13: EQUATE  V       =       256             ;
14: ;
15: EQUATE  HOST    =       0               ;
16: EQUATE  READ    =       4               ;
17: EQUATE  WRITE   =       5               ;
18: ;
19: EQUATE  STARTS  =       0               ;
20: EQUATE  STARTD  =       1FFH            ;
21: ;
22: ;****************************************
23: ;
24: ;       INPUT-OUTPUT
25: ;
26: ;----------------------------------------
27: INPUT   LSA0.   LDA0.   RDATA AT 0      ;
28: ;
29: OUTPUT  PEND.   WDAT.   WADR.   RDAT    ;
30: ;
31: ;****************************************
32: ;
33: ;       LINK TABLE
34: ;
35: ;----------------------------------------
36: ;
37: LINK    SRD20.  SRD21.  SRD22   =       FGEN1   (SRD11. LSA0    )       ;
38: LINK    SRD30.  SRD31.  SRD32   =       FGEN2   (SRD33. SRD20   )       ;
39: LINK    SRD40.  SRD41.  SRD42   =       FGEN5   (SRD43. SRD30   )       ;
40: LINK    SRD11                   =       FQUE1   (SRD21. SRD32   )       ;
41: LINK    SRD33                   =       FQUE2   (SRD31. MRQ1    )       ;
42: LINK    SRD43                   =       FQUE5   (SRD41. PRO2    )       ;
43: LINK    PEND                    =       FOUT0   (SRD22          )       ;
44: LINK    RDAT                    =       FOUTR   (SRD40          )       ;
45: LINK    MRQ1.   MRQ2            =       FCOP0   (SRD42          )       ;
46: ;
47: LINK    SWR20.  SWR21.  SWR22   =       FGEN3   (SWR11. LDA0    )       ;
48: LINK    SWR30.  SWR31.  SWR32   =       FGEN4   (SWR33. SWR20   )       ;
49: LINK    SWR11                   =       FQUE3   (SWR21. SWR32   )       ;
50: LINK    SWR33                   =       FQUE4   (SWR31. MRQ2    )       ;
51: LINK                            =       FDUST   (       .SWR22  )       ;
52: ;
53: LINK    IMDR.   IMDL            =       FCOP1   (RDATA          )       ;
54: LINK    TABLR                   =       FAND1   (IMDR           )       ;
55: LINK    TABLR                   =       FSHR1   (IMDL           )       ;
56: LINK    PROS                    =       FTAB1   (TABLR          )       ;
57: LINK    PRO1.   PRO2            =       FCONT1  (PROS           )       ;
58: LINK    TPROF                   =       FACC1   (PRO1           )       ;
59: LINK    WDAT.   WADR            =       FOUTW   (TPROF. SWR30   ).      ;
60: ;
```

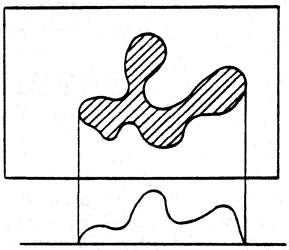
```
  61: ;****************************************
  62: ;
  63: ;       FUNCTION TABLE
  64: ;
  65: ;----------------------------------------
  66: ;
  67: FUNCTION    FGEN1   =   COPYBK  (1.   L*R),     CNTGE   (V/R    )       ;
  68: FUNCTION    FGEN2   =   COPYBK  (1.     L),     CNTGE   (R      )       ;
  69: FUNCTION    FGEN5   =   COPYBK  (Q.     1),     CNTGE   (H/Q    )       ;
  70: FUNCTION    FQUE1   =   QUEUE   (QBASE1.1)                               ;
  71: FUNCTION    FQUE2   =   QUEUE   (QBASE2.1)                               ;
  72: FUNCTION    FQUE5   =   QUEUE   (QBASE5.1)                               ;
  73: FUNCTION    FOUT0   =   OUT1    (HOST.  0)                               ;
  74: FUNCTION    FOUTR   =   OUT1    (READ.  0)                               ;
  75: FUNCTION    FCOP0   =   COPYM   (2.     0)                               ;
  76: FUNCTION    FGEN3   =   COPYBK  (1.     R),     CNTGE   (V/R    )       ;
  77: FUNCTION    FGEN4   =   COPYBK  (1.     1),     CNTGE   (R      )       ;
  78: FUNCTION    FQUE3   =   QUEUE   (QBASE3.1)                               ;
  79: FUNCTION    FQUE4   =   QUEUE   (QBASE4.1)                               ;
  80: FUNCTION    FDUST   =   COUNT   (1      )                                ;
  81: FUNCTION    FCOP1   =   COPYM   (2.     0)                               ;
  82: FUNCTION    FAND1   =   AND     (X      ),      RDCYC3  (DFF.   1)       ;
  83: FUNCTION    FSHR1   =   SHR     (X      ),      RDCYC5  (EIGHT. 1)       ;
  84: FUNCTION    FTAB1   =   RDIDX   (RBASE1 )                                ;
  85: FUNCTION    FCONT1  =   COUNT   (2*Q    )                                ;
  86: FUNCTION    FACC1   =   ACC     (X      ),      COUNT   (2*Q    )       ;
  87: FUNCTION    FOUTW   =   OUT2    (WRITE. 20H. 0).QUEUE   (QBASE6.1)       ;
  88: ;
  89: ;****************************************
  90: ;
  91: ;       DATA MEMORY
  92: ;
  93: ;----------------------------------------
  94: ;
  95: MEMORY  QBASE1  =       AREA    (1      )       ;
  96: MEMORY  QBASE2  =       AREA    (1      )       ;
  97: MEMORY  QBASE3  =       AREA    (1      )       ;
  98: MEMORY  QBASE4  =       AREA    (1      )       ;
  99: MEMORY  QBASE5  =       AREA    (1      )       ;
 100: MEMORY  QBASE6  =       AREA    (1      )       ;
 101: MEMORY  DFF     =       00FFH                   ;
 102: MEMORY  EIGHT   =       8                       ;
 103: MEMORY  RBASE1  =       0.1.1.2.1.2.2.3.1.2.2.3.2.3.3.4.
 104:                         1.2.2.3.2.3.3.4.2.3.3.4.3.4.4.5.
 105:                         1.2.2.3.2.3.3.4.2.3.3.4.3.4.4.5.
 106:                         2.3.3.4.3.4.4.5.3.4.4.5.4.5.5.6.
 107:                         1.2.2.3.2.3.3.4.2.3.3.4.3.4.4.5.
 108:                         2.3.3.4.3.4.4.5.3.4.4.5.4.5.5.6.
 109:                         2.3.3.4.3.4.4.5.3.4.4.5.4.5.5.6.
 110:                         3.4.4.5.4.5.5.6.4.5.5.6.5.6.6.7.
 111:                         1.2.2.3.2.3.3.4.2.3.3.4.3.4.4.5.
 112:                         2.3.3.4.3.4.4.5.3.4.4.5.4.5.5.6.
 113:                         2.3.3.4.3.4.4.5.3.4.4.5.4.5.5.6.
 114:                         3.4.4.5.4.5.5.6.4.5.5.6.5.6.6.7.
 115:                         2.3.3.4.3.4.4.5.3.4.4.5.4.5.5.6.
 116:                         3.4.4.5.4.5.5.6.4.5.5.6.5.6.6.7.
 117:                         3.4.4.5.4.5.5.6.4.5.5.6.5.6.6.7.
 118:                         4.5.5.6.5.6.6.7.5.6.6.7.6.7.7.8 ;
 119: ;
 120: ;****************************************
 121: ;
 122: ;       START .
 123: ;
 124: ;----------------------------------------
 125: ;
 126: START                                               ;
 127: ;
 128: DATA    CRESET  (IPP                        )       ;
 129: DATA    EXEC    (IPP.   LSA0.   STARTS      )       ;
 130: DATA    EXEC    (IPP.   LDA0.   STARTD      )       ;
 131: ;
 132: END                                                 ;
```

## 6.2 Vertical Profiling

### 6.2.1 Processing Explained

A vertical profile is a histogram obtained by projecting a binary image in the vertical direction, as shown in Figure 6-5.

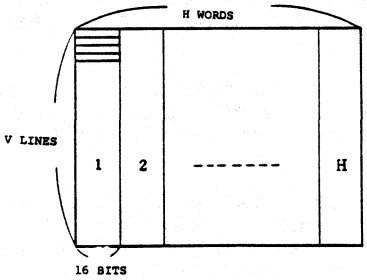
Figure 6-5
A Vertical Profile



### 6.2.2 Algorithm

An object screen is comprised of H words horizontally and V lines vertically.

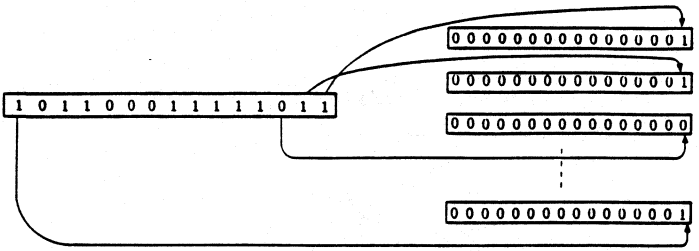
Figure 6-6
Definition of the Source Image Area




In a vertical profiling, the screen is divided horizontally into H blocks of one word each. In this program a vertical block is sought for each block. Since there are 16 bits in a word, processing a single block produces 16 cumulative values.

The processing of a block starts with one word which comprises the block copying it to 16 parts. The PU

instruction GET1 is used to shift all bits of the data to the LSBs of its 16 copies. This processing can be carried on efficiently by the folowing procedure: have the data 0-15 stored in the DM, and execute the GET1 instruction while reading this data cyclically by the use of the AG & FC instructsion RDCYCS.

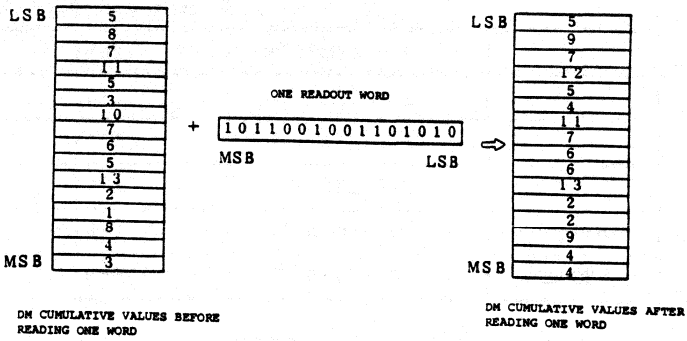



Sixteen contiguous areas are allocated in the DM, and the corresponding cumulative values are stored in these areas (Figure 6-7). The cyclic address generation function of the RDCYCS instruction in the AG&FC is used in reading from and writing to these 16 DM areas. These 16 DM areas are cleared to 0 before the processing of a block begins. In this program these processing steps and the clearing operation are performed in parallel in order to minimize the loss of efficiency through the generation of read addresses and access to external memories. (Although the clearing operation can be performed much more quickly in this program than the writing of cumulative values, the sequence of execution of PU and GE instructions cannot be predicted in general. You should investigate such a sequence using the uPD7281 simulator).

Upon completion of the processing of a block, these areas hold the cumulative values of the corresponding bits. These values are read and written in specified image memory addresses.

As in the case of horizontal profiling, control of the H blocks is with the CNTGE instruction. Be aware that the order in which data are read from the image memory is different with horizontal profiling.

Figure 6-7
Updating the DM




### 6.2.3 Parameters and Their Applicable Ranges

<Assembler-coded parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area (SRC) words in horizontal direction (number of blocks)
V ... Number of source image area (SRC) lines in vertical direction
R ... Number of segments read from the image memory (vertical direction)

<Start-up token-defined parameters>

STARTS ... Source image area (SRC) starting address
STARTD ... Starting address of a profile storage

The allowable values of these parameters are indicated in the table below.

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 65535/R | (64) |
| H | 1 - 256 | (16) |
| V | R - 256 x R | (512) |
| R | 1 - 256 | (16) |
| STARTS | 0 - 65535 | ( 0)* |
| STARTD | 0 - 65535 | (1FFH)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting values of parameters H and V. It is assumed that source areas are given in word boundaries.

### 6.2.4 Flow Graph Explained

<Explanation of main nodes>

FBLOK : Creates the starting addresses of blocks based on the SRC starting address STARTS.

FRDZ, FGEZ, FWRZ : Clears the save area (BASE1) to 0 for 16 count values.

FRDAD1, FRDAD2 : Creates the read addresses within a block using the GNTGE instruction in double loops.

FOUTR : Reads SRC data.

FGEDA : Makes 16 copies of one-word SRC data that was read.

FGET : Outputs the value of a given bit position in the read data in the LSB bit. Information specifying the bit positions is contained in the DM, written contiguously. FGET reads this information and passes it as a GET1 instruction token.

DM data
15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0

For example, the first token arriving

in this node reads 15 from the DM and executes the GET1 instruction. Therefore, the fifteenth bit of the arriving data is shifted to the LSB. The next token arriving in the node reads 14 from the DM, causing the fourteenth bit to be shifted to the LSB. By using the DM's cyclic read function and the GET1 instruction of the PU in this manner it is possible to output any desired bit with one node.

FADD, FWRP : Reads the contents of the DM count value save area (BASE1) successively, adds to them the bits that were output at node FGET, and writes the results back to the same area. The cyclic read and cyclic write are also used in this case.

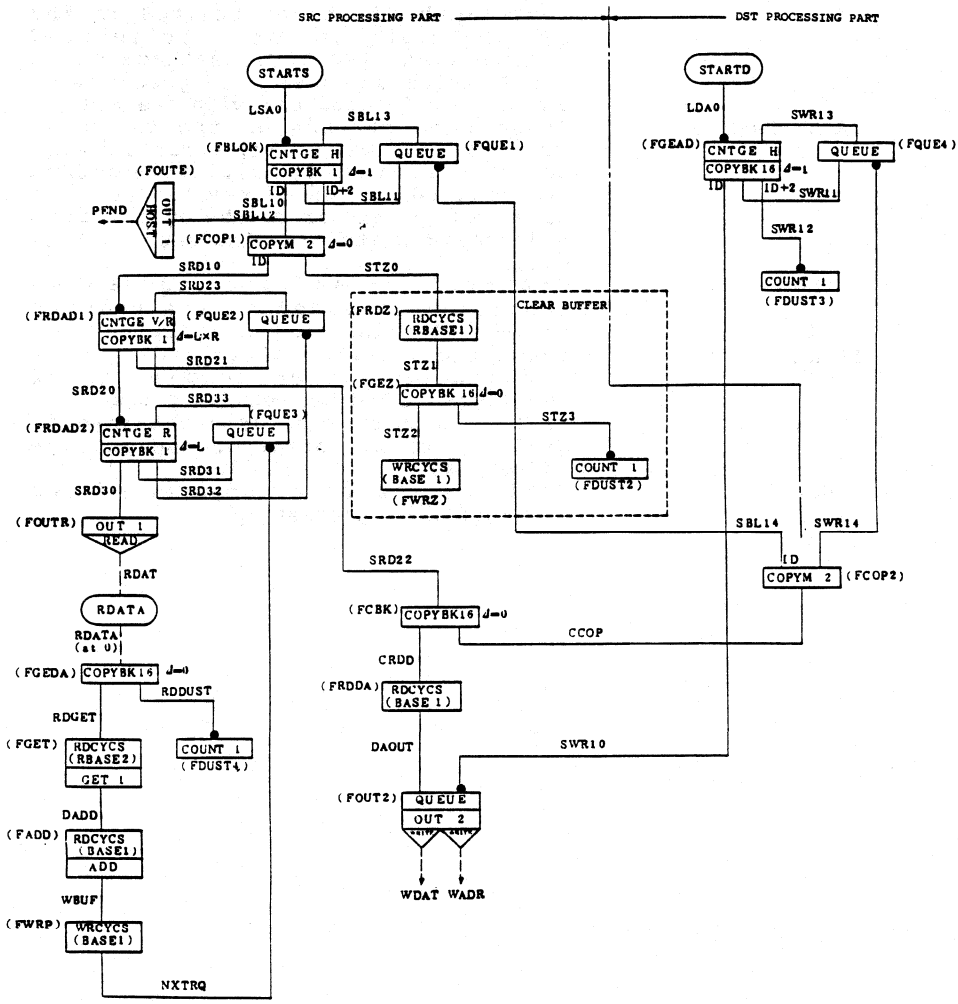
FGEAD : Creates 16 storage addresses continuously based on the profile storage starting address STARTD.

FRDDA : Reads the 16 data values from the DM continuously upon completion of cumulative addition for one block.

FOUT2 : Writes 16 values of data obtained in FRDDA to their corresponding storage addresses.

FOUTE : Notifies the host computer of the completion of processing.

Figure 6.8
A Flow Graph of Vertical Profiling



## 6.2.5 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;       VERTICAL PROFILE
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  H       =       16              ;
10: EQUATE  V       =       512             ;
11: EQUATE  R       =       16              ;
12: EQUATE  L       =       64              ;
13: ;
14: EQUATE  HOST    =       0               ;
15: EQUATE  READ    =       4               ;
16: EQUATE  WRITE   =       5               ;
17: ;
18: EQUATE  STARTS  =       0               ;
19: EQUATE  STARTD  =       1FFH            ;
20: ;
21: ;****************************************
22: ;
23: ;       INPUT-OUTPUT
24: ;
25: ;----------------------------------------
26: ;
27: INPUT   LSA0.   LDA0.   RDATA AT 0       ;
28: ;
29: OUTPUT  PEND.   RDAT.   WDAT.   WADR     ;
30: ;
31: ;****************************************
32: ;
33: ;       LINK TABLE
34: ;
35: ;----------------------------------------
36: ;
37: LINK    SBL10.  SBL11.  SBL12   =       FBLOK   (SBL13. LSA0    )
38: LINK    SBL13                   =       FQUE1   (SBL11. SBL14   )
39: LINK    SRD10.  STZ0            =       FCOP1   (SBL10          )
40: LINK    PEND                    =       FOUTE   (SBL12          )
41: LINK    SBL14.  SWR14           =       FCOP2   (CCOP           )
42: LINK    STZ1                    =       FRDZ    (STZ0           )
43: LINK    STZ2.   STZ3            =       FGEZ    (STZ1           )
44: LINK                            =       FWRZ    (STZ2           )
45: LINK                            =       FDUST2  (       ,STZ3   )
46: ;
47: LINK    SRD20.  SRD21.  SRD22   =       FRDAD1  (SRD23  ,SRD10  )
48: LINK    SRD23                   =       FQUE2   (SRD21  ,SRD32  )
49: LINK    SRD30.  SRD31.  SRD32   =       FRDAD2  (SRD33  ,SRD20  )
50: LINK    SRD33                   =       FQUE3   (SRD31  ,NXTRQ  )
51: LINK    RDAT                    =       FOUTR   (SRD30          )
52: ;
53: LINK    RDGET.  RDDUST          =       FGEDA   (RDATA          )
54: LINK                            =       FDUST4  (       ,RDDUST )
55: LINK    DADD                    =       FGET    (RDGET          )
56: LINK    WBUF                    =       FADD    (DADD           )
57: LINK    NXTRQ                   =       FWRP    (       ,WBUF   )
58: ;
59: LINK    SWR10.  SWR11.  SWR12   =       FGEAD   (SWR13  ,LDA0   )
60: LINK    SWR13                   =       FQUE4   (SWR11  ,SWR14  )
61: LINK                            =       FDUST3  (       ,SWR12  )       ;
62: ;
63: LINK    CRDD.   CCOP            =       FCBK    (SRD22          )       ;
64: LINK    DAOUT                   =       FRDDA   (CRDD           )       ;
65: LINK    WDAT.   WADR            =       FOUT2   (DAOUT  ,SWR10  )       ;
66: ;
```

```
67: ;****************************************
68: ;
69: ;       FUNCTION TABLE
70: ;
71: ;----------------------------------------
72: ;
73: FUNCTION    FBLOK   =   COPYBK  (1,     1),     CNTGE   (H       )
74: FUNCTION    FQUE1   =   QUEUE   (QBASE1,1)
75: FUNCTION    FCOP1   =   COPYM   (2,     0)
76: FUNCTION    FOUTE   =   OUT1    (HOST,  0)
77: FUNCTION    FCOP2   =   COPYM   (2,     0)
78: FUNCTION    FRDZ    =   RDCYCS  (RBASE1,1)
79: FUNCTION    FGEZ    =   COPYBK  (16,    0)
80: FUNCTION    FWRZ    =   WRCYCS  (BASE1,16)
81: FUNCTION    FDUST2  =   COUNT   (1       )
82: ;
83: FUNCTION    FRDAD1  =   COPYBK  (1,   L*R),     CNTGE   (V/R     )
84: FUNCTION    FQUE2   =   QUEUE   (QBASE2,1)
85: FUNCTION    FRDAD2  =   COPYBK  (1,     L),     CNTGE   (R       )
86: FUNCTION    FQUE3   =   QUEUE   (QBASE3,1)
87: FUNCTION    FOUR    =   OUT1    (READ,  0)
88: ;
89: FUNCTION    FGEDA   =   COPYBK  (16,    0)
90: FUNCTION    FDUST4  =   COUNT   (1       )
91: FUNCTION    FGET    =   GET1    (X       ),     RDCYCS  (RBASE2,16)
92: FUNCTION    FADD    =   ADD     (X       ),     RDCYCS  (BASE1, 16)
93: FUNCTION    FWRP    =   WRCYCS  (BASE1,16)
94: ;
95: FUNCTION    FGEAD   =   COPYBK  (16,    1),     CNTGE   (H       )
96: FUNCTION    FQUE4   =   QUEUE   (QBASE4,16)
97: FUNCTION    FDUST3  =   COUNT   (       1)
98: ;
99: FUNCTION    FCBK    =   COPYBK  (16,    0)
100: FUNCTION   FRDDA   =   RDCYCS  (BASE1,16)
101: FUNCTION   FOUT2   =   OUT2   (WRITE, 20H, 0), QUEUE   (QBASE5,16)
102: ;
103: ;****************************************
104: ;
105: ;       DATA MEMORY
106: ;
107: ;----------------------------------------
108: ;
109: MEMORY  QBASE1  =       AREA    (1       )                      ;
110: MEMORY  QBASE2  =       AREA    (1       )                      ;
111: MEMORY  QBASE3  =       AREA    (1       )                      ;
112: MEMORY  QBASE4  =       AREA    (16      )                      ;
113: MEMORY  QBASE5  =       AREA    (16      )                      ;
114: MEMORY  RBASE1  =       0                                       ;
115: MEMORY  BASE1   =       AREA    (16      )                      ;
116: MEMORY  RBASE2  =       15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0  ;
117: ;
118: ;****************************************
119: ;
120: ;       START
121: ;
122: ;----------------------------------------
123: ;
124: START                                                   ;
125: ;
126: DATA    EXEC    (IPP,   LSA0,   STARTS  )              ;
127: DATA    EXEC    (IPP,   LDA0,   STARTD  )              ;
128: ;
129: END                                                     ;
```

# Chapter 7

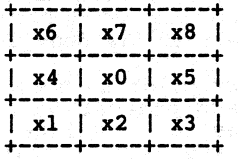
# Mask Processing

The mask processing discussed in this chapter performs the following conversion of nine points, including the object point x0:

x0'=F(x0,x1,x2,x3,x4,x5,x6,x7,x8)

The value x0' of the object point x0 is determined by the eight adjoining points and x0.

Figure 7-1
Mask Processing




Mask processing can be roughly divided structurally into two processing parts: the image memory address generation part and the computation part.

This Application Library contains three examples of mask processing: smoothing, thinning, and edge detection. Image memory address generation, which is common to all these examples, is explained as a separate item.

## 7.1 Common Processing (Image Memory Address Generation)

### 7.1.1 Processing Explained

As the name implies, image memory address generation concerns the generation of addresses for image memory read/write. These addresses are required in the computational processing discussed later in this chapter. Image memory address generation consists of the following three parts:
(1) a read address generation part which generates read addresses and reads the data;
(2) a write address generation part which generates write addresses and writes the data processed in the computation part; and
(3) a final processing part which writes final data and initializes some of the FTT fields for the AG & FC

instruction.

The address generation schemes employed in the read address generation part and the write address generation part share the same flow graph organization.

The final processing part reinitializes the counters in the FTT that underwent changes in value as a result of the actions of the read address generation part, write address generation part, and the computation part (discussed later). This ensures that these counters will be ready to operate correctly when the next startup token is input without downloading the same program to the uPD7281 again.

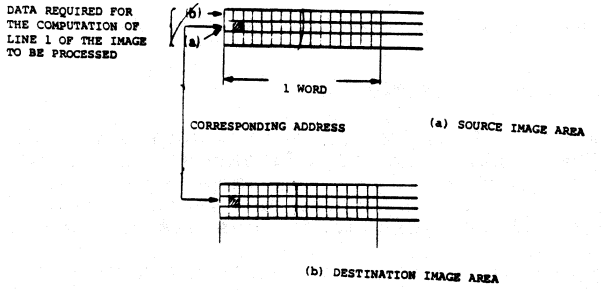
### 7.1.2 Algorithm

All mask processing programs described in this chapter require data occupying an area three dots horizontally by three dots vertically, as shown in Figure 7-1. Consequently, the reading of image memory source area is done in units of a total of three words: one word horizontally and three lines vertically. The writing is done in 1-word units.

Further, since an object point x0 needs the adjoining eight points, the read address generation part of this program generates read addresses for the line containing the point x0 and the lines above and below simultaneously. Therefore, the processing of H words horizontally involves the generation of 3 x H readout addresses.

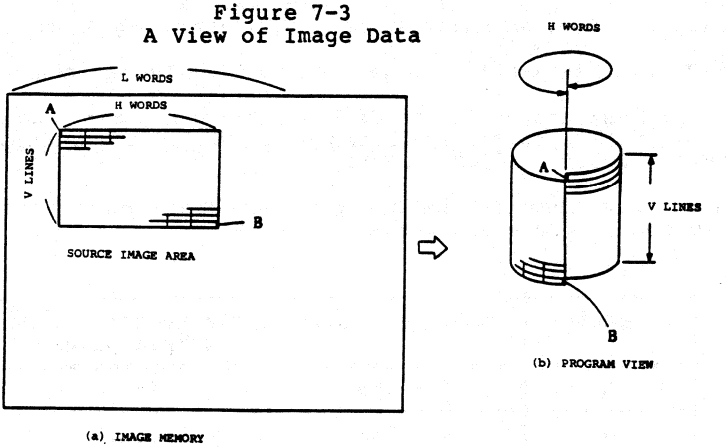
As a result, the address for the source image area in this program is specified as (b), rather than (a) in Figure 7-2.

Figure 7-2
Address Correspondence



Further, when one word of write data is generated from three words of read data, the LSB of the output data is indeterminate, as will be discussed later (see Section 7.2, "Computational Processing"). To get around these problems, data obtained in this program is first stored in the Data Memory (DM) of the uPD7281, and the writing of data from the next computational pass is done by reading it from the DM and adding one bit to it. This scheme, however, requires that each source area have preceding data and each data at the right margin of screen have successive data. In response to this requirement image data is treated as a loop in this program, as shown in Figure 7-3(b), on the assumption that, in general, data at the right or left edge of a screen is not significant. This means that only the points A (starting point) and B (end point) in the figure need to be considered. The data read from the DM at the starting point A is invalidated (through processing done in the computation part), and the data value from bit 2 at the end point (next to the LSB) is written to the LSB, which was indeterminate (final processing part).

Figure 7-3
A View of Image Data




In a binary image, an image in memory can be a 0-background image or a 1-background image, depending on the type of background data utilized. Mask processing produces different results depending on the image's background. For instance, if a 0-background thinning program was used on a 1-background graphic image, the line would get wider instead of narrower. However, the same program can be used for both 0-background and 1-background graphic image processing tasks if the computation parts of a program are written for the 0 background and data bits are reversed from 0 to 1 or from 1 to 0, as appropriate.

Therefore, the C bit of the STARTD token (which indicates the starting address of a destination image area) is used in this program to determine whether the destination image has a 0-background or a 1-background. If it has a 1-background the data is reversed between the input and output parts of the computation part. This requires that the C bit of image data is always 0 in this program.

### 7.1.3 Parameters and Their Applicable Ranges

<Assembler-Coded Parameters>

L ... Number of image memory words in horizontal direction
H ... Number of source image area (SRC) words in horizontal direction
V ... Number of source image area (SRC) lines in vertical direction
R ... Number of segments read from the image memory (vertical direction)

<Startup Token-Defined Parameters>

STARTS ... Source image area (SRC) starting address - L (Note 1)
STARTD ... Destination image area (DST) starting address

Note 1: Mask processing needs the line above for processing. (i.e., if SRC starting address = 0000H and L = 0080H, then STARTS = 0000H - 0080H = FF80H)

The values that can be assigned to these parameters are indicated in the table below:

| Parameter | Applicable range | (Value set in the example program) |
|---|---|---|
| L | 0 - 255 | (64) |
| H | 1 - 256 | (32) |
| V | R - 256 x R | (512) |
| R | 1 - 256 | ( 2) |
| STARTS | 0 - 65535 | (FF80H)* |
| STARTD | 0 - 65535 | (20H)* |

* : Although STARTS and STARTD are variables (i.e., addresses), they are given default values since they

are used in the assembler DATA statement. When the uPD7281 is started up, the starting addresses of the SRC and DST areas are input as execution tokens.

Since no provision is made in this program for switching banks, you should exercise care in setting values for parameters H and V.

### 7.1.4 Flow Graph Explained

A flow graph of the common processing part (image memory address generation) is shown in Figure 7-4. A combination of this flow graph and one of the computational part described below makes up a flow graph of a mask processing program.

<Explanation of Main Nodes>

FT1 and FT2 extract the information that determine whether the image has a 0-background or a 1-background from the C bit of the STARTD token, and store the information in the Data Memory (DM). This information is used in the computation part.

FT3, FT4, and FT5 generate destination image write addresses based on STARTD. Two steps, FT3 and FT4, are required because there can be more than 257 vertical lines.

FT6, FT7, and FT8 serve to synchronize the generation of addresses. FT8 synchronizes the activity of the read address generation part as well. FT10 through FT12 and FT15 through FT17 generate source image read addresses upon receipt of the STARTS token (the source image area starting address minus L) In this process FT17 works in synchronization with the computation part.

FT13 is a node for creating addresses for three contiguous vertical words.

FT27 and FT28 are nodes for creating read tokens and write tokens, respectively.

In this process the arc X46 is a write data token generated in the computation part.

The word at the end is processed in the computation part. The results of the computation are stored in the Data Memory (DM) of the uPD7281, and the token returns to FT17. This causes FT12, FT16, FT11, FT15, and FT10 to be driven, in this order, and the token is transferred to the final processing part.

The final processing part takes the results of processing the end word in FT18, FT59, FT73, FT21, FT58, and FT23 and

sets the LSB of this word equal to the value of the second bit. Then the destination image data is complete.

In addition to this data generation, FT59 updates the read counter located in the FTT part so that the values in this read counter will indicate that all data have been read from the storage area. As a result, FT59 keeps itself in synchronization with the computation part.

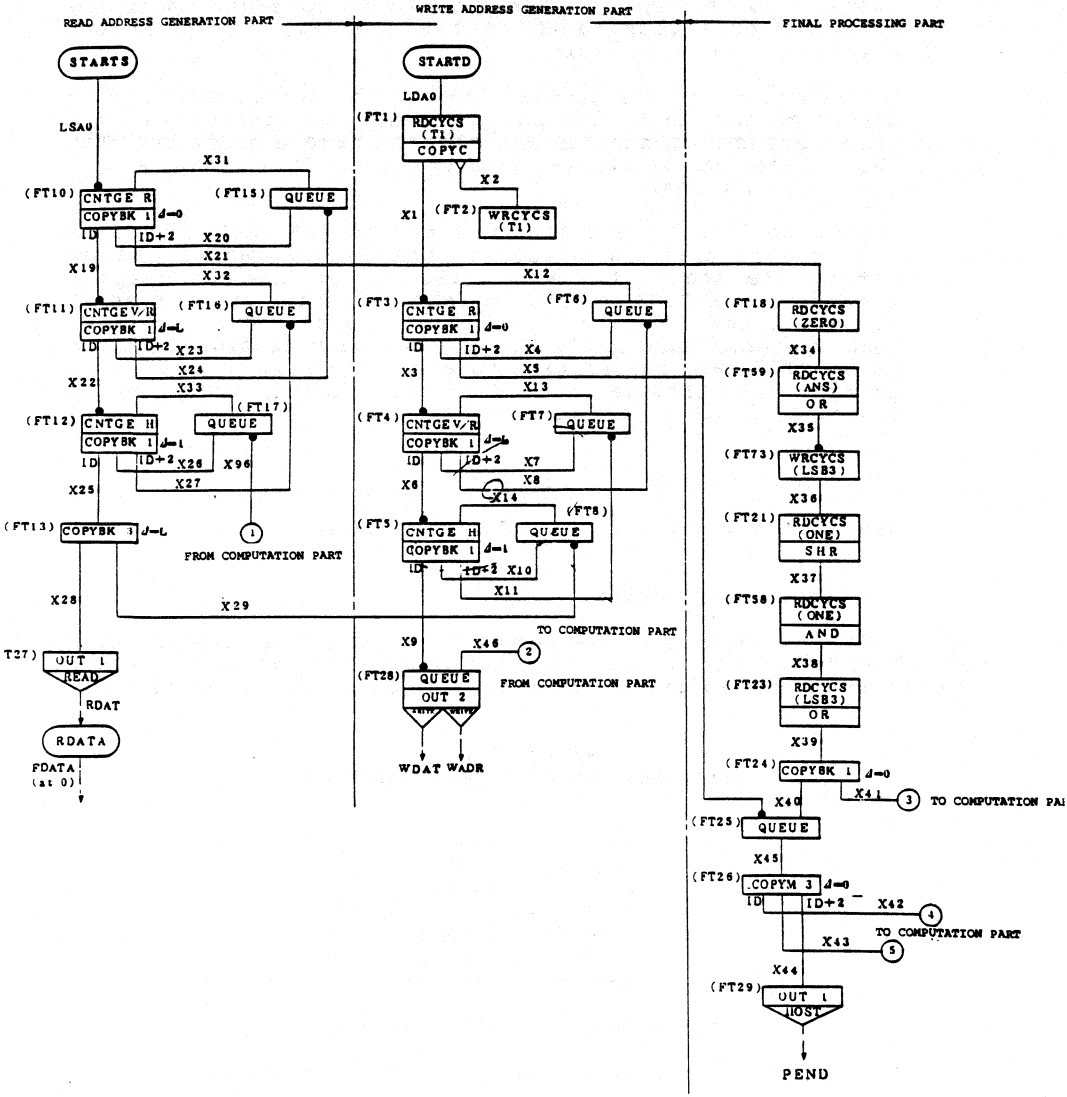
The process of performing an OR operation in FT59 requires another operand. Data for this operand is generated in FT18.

FT73 performs copying by storing one copy in the Data Memory (DM). Setting FTRC = 1 keeps the input data at the node from being erased.

FT21 and FT58 extract bit 2 from this data and set it to the value of the LSB. FT73 generates processing result data; this is stored in the Data Memory (DM).

The processing result data copied in FT24 is sent to the computation part and written to the destination image area. The other copy data waits for completion of the generation of destination area write addresses (FT25). It then initializes the uPD7281 counters (arc X42, X43) and issues a completion notification token (FT29).

Figure 7-4
A Flow Graph of a Mask Processing Image Memory
Address Generation Part



### 7.1.5 Tips on Writing Flow Graphs

To ensure the proper execution of this program, the counters in the FTT field must be initialized. (This is due to the way the computational part works, as will be discussed later.) You must be careful when initializing the ACC instruction and the FTT for the AG & FC instruction if the program, once downloaded to the uPD7281, is to be run multiple times with startup tokens.

For example, the CUT instruction, which cuts as many FTRC=0 tokens as specified will not perform a cut operation if the instruction's counter is not cleared before a new run. The CUT counter can be cleared by sending an FTRC=1 token at the tail end of a run.

The Write Counter (WC) field of the WRCYCS instruction is set to 1 so that one piece of data is written to the DM before the start of a run. This causes the counterpart instruction RDCYCS in FT59 to read data that appears as if it were written during the preceding pass. Thus, the data that was processed in the computation part is stored in the DM by FT57, and the writing to the image memory is done by getting data that was stored in the preceding pass and by appending 1 bit to the LSB.

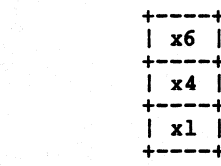
### 7.1.6 Assembler Source Listing

See Sections 7.2.1.4, 7.2.2.5., and 7.2.3.4.

## 7.2 Computation Processing

During data generation in the computation part, the determination of "0" or "1" on the object point x0' is made by performing the F operation on the adjoining eight points, as shown in Figure 7-5.

Figure 7-5
Determination of Object Point x0'




$$x0' = F(x0, x1, x2, \ldots, x8)$$

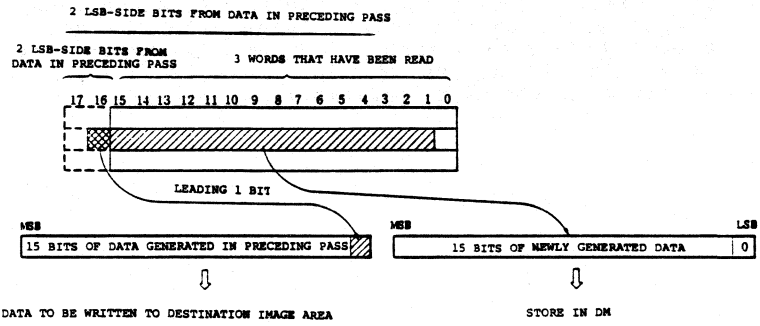
where x0' represents x0 after transformation.

For this purpose, one word of destination image data is generated by reading a total of three words from the source image area lines vertically (as explained in Section 7.1). To be more specific, the one word of written data is derived from a total of 18 x 3 bits of data: the 3 words of data read from the source image area and bits 1 and 0 of the 3-word data that was read in the preceding pass (see Figure 7-6).

The low-order 15 bits of the 16-bit data thus generated are stored in the DM of the uPD7281, along with an appended LSB of 0. The leading bit is combined with 15 bits that were stored in the DM in the preceding pass to form one word of destination image data which is written to the target image area. This is because not all eight bits in the positions adjoining the LSB of the word are available during a pass, preventing the execution of the F operation. The determination of value of the LSB, therefore, has to be deferred until the next pass.

To facilitate subsequent computations, the LSB is set to 0.

Figure 7-6
Creation of 1-Word Data to be Written




### 7.2.1 Smoothing

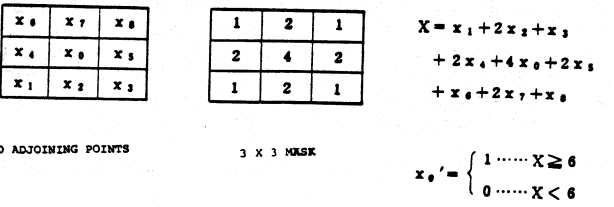
#### 7.2.1.1 Processing Explained

Smoothing is used to remove minor noise data from images and to smooth out their edges. The processing involves masking using object point x0 and picture element data in the adjoining points (8-point neighborhood).

### 7.2.1.2 Algorithm

Point x0' is determined by the use of a three by three mask shown in Figure 7-7. The point x0' is a mapping of x0.

Figure 7-7
Smoothing Algorithm

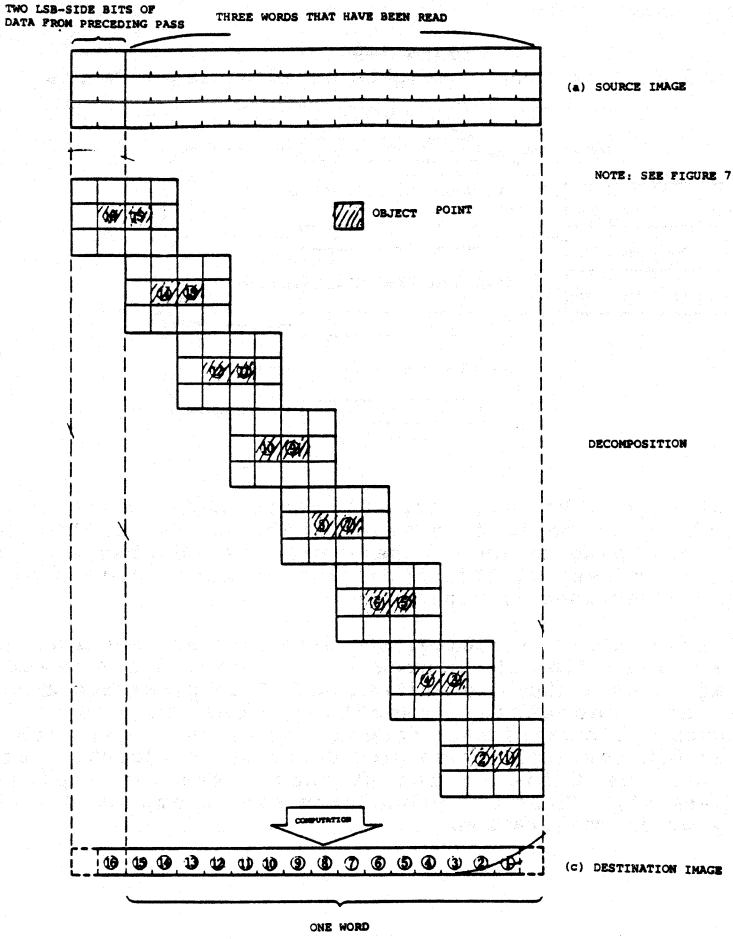



X0 AND ADJOINING POINTS

3 X 3 MASK

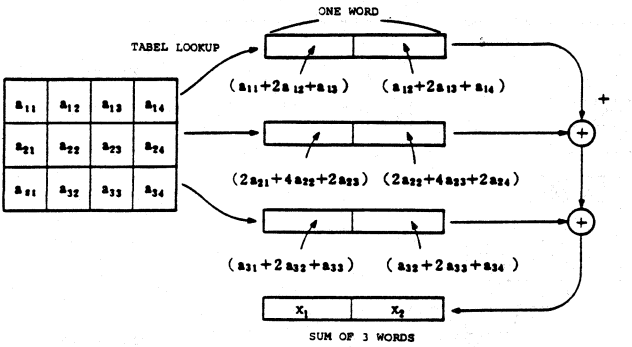
To increase processing speed in actual situations, the three words read from the source image area divided into eight areas of four bits each so that two points can be obtained at a time by the use of a DM table lookup (the resultant values of 3 x 3 mask operation).

## Figure 7-8
## Data Decomposition and Assembly



Although this processing mainly involves computations on x0 and the adjoining points, a table lookup is performed in this program on the basis of 4-bit decomposition data to obtain the sum of the weights of the lines in terms of an high-order byte (x1) and a low-order byte (x2). In other words, calculations on nine adjoining points centered on each of a22 and a23 are carried on simultaneously. This processing is performed in parallel on the eight 4-bit decomposition data.

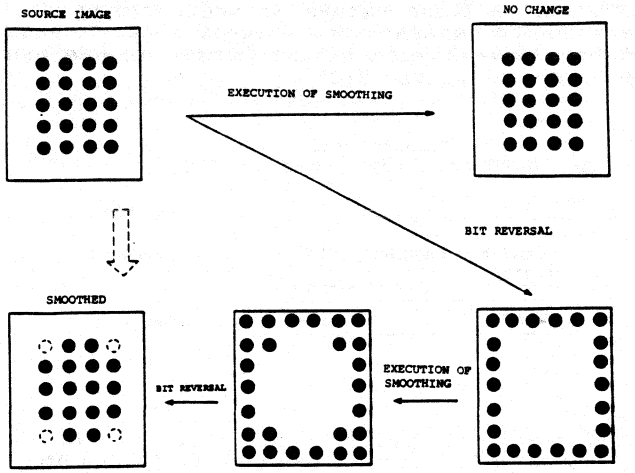
Figure 7-9
A Method of Obtaining Weights of Adjoining 9 Points




By this algorithm, however, x0' becomes 1 if x0, x1, x2, x4 are all equal to 1. This means that, no matter how many times smoothing is done, there will be no change in the results. Therefore, this type of smoothing would have no effect on rectangular figures.

To overcome this difficulty, the data bits are reversed when they are read from the source image area and are reversed back again when they are written out. This procedure ensures x0'=0 and successful smoothing (see Figure 7-10). Consequently, smoothing is done in one of two ways: with and without bit reversal. These procedures are differentiated by means of the C bit in the startup token (as explained previously). These two procedures can be employed either singly or in combination.

Figure 7-10
Smoothing of a Rectangular Figure




7.2.1.3 Flow Graph Explained

Figure 7-12 shows a flow graph (computation part) for a smoothing operation. The smoothing program is made up of this flow graph and a flow graph which is discussed in Section 7.1.

<Explanation of Main Nodes>

FT30 is the node for reversing image data bits in cases where the background is 1. Whether a given background is 0 or 1 is determined on the basis of bit C of the startup token STARTD (Figure 7-12).

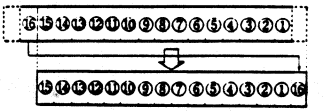
FT31, FT32, and FT35 through FT48 produce a computed value X from source image data.

FT31 distributes the read data according to lines in the vertical direction. FT32 and FT35 through FT43 create 4-bit data. FT44 and FT45 determine line values in order to obtain a computed value X by table lookup. FT47 and FT48 obtain the computed value X by adding together the line values determined in FT44 and FT45.

In addition to creating 4-bit data, FT32 copies data so that the lowest two bits of the data can be stored in the DM (LSB1, LSB2, and LSB3 areas). This is necessary to perform computations on the next data.

FT49 through FT54 evaluate the computed value X. FT49 decomposes the binary values within a word obtained in FT48, and FT50 and FT51 determine whether these values are 0 or 1. FT52 and FT53 place these values in their proper positions within the word and generate one word of data. It should be noted that bit 16 in Figure 7-8 is placed on the LSB side within the word (see Figure 7-11).

Figure 7-11
One-Word Data Generated in FT54




FT56 and FT57 change the value of bit 16 of the 16-bit data obtained in FT54 to 0 and store the result in the DM (in the ANS area) for use in the computation of the subsequent step.

FT58, FT59, and FT60 add bit 16 of the data obtained in FT54 to the 15-bit data that was stored in the preceding computation pass. They pass the result to the address generation part.

In the above process the leading edge word in the source image area has invalid data in the ANS area of the DM. Node FT60 erases this invalid data.

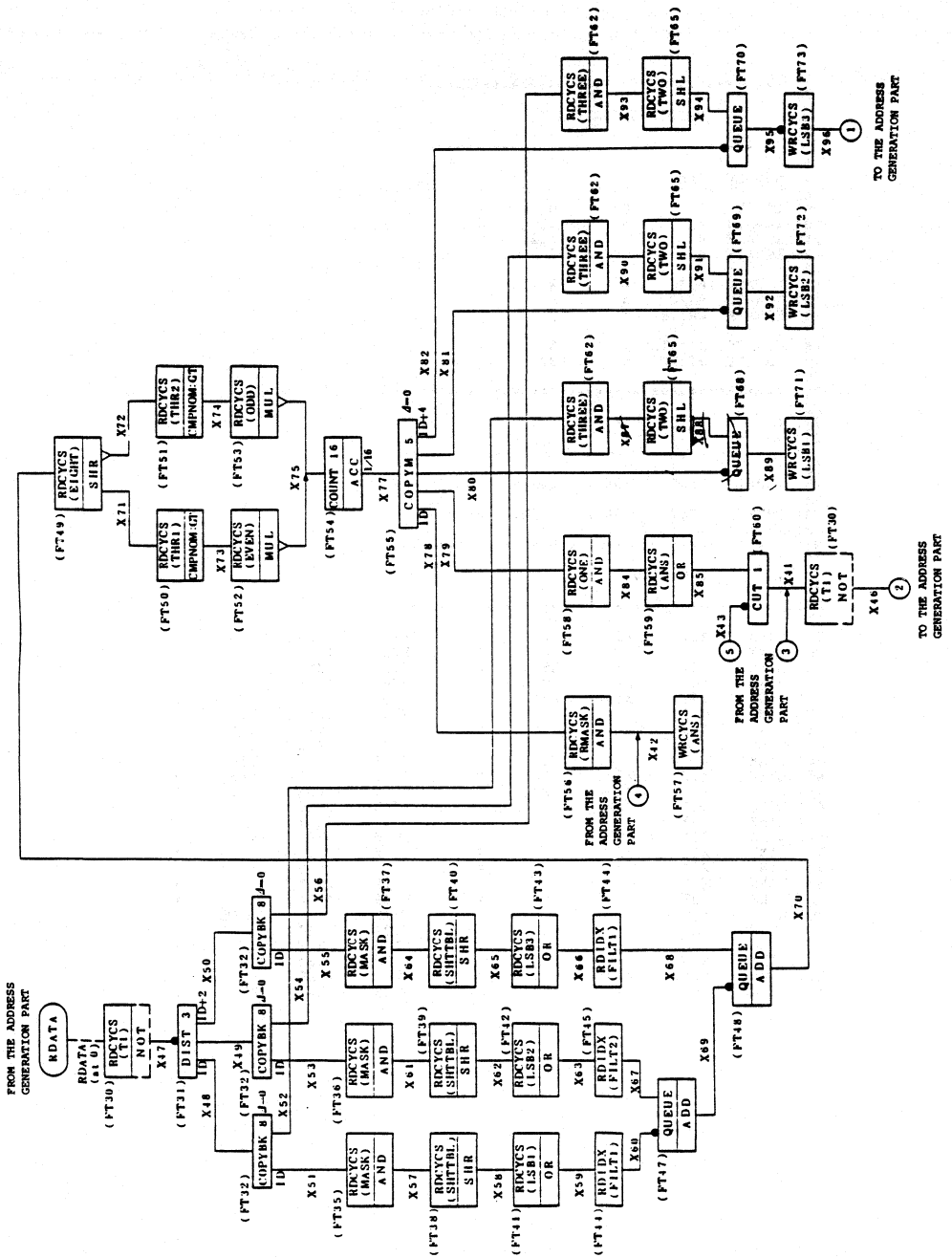
The tail end word left in the ANS area of the DM with its LSB still undetermined is sent to the final processing part of the address generation part, where it undergoes final processing.

The LSB data of this write data is undetermined (and is set to 0); in the program the value of the second lowest bit is used as an LSB value. This is done on the assumption that such a substitution made on the right hand edge of the area to be processed will have no appreciable impact.

FT62, FT65, and FT68-FT73 store the low-order two bits in the LSB1 - LSB3 area of the DM for use in subsequent computation. Nodes FT68 - FT70 ensure synchronization with the computation operations.

FT73 signals completion of the processing of a line to the address generation part. FTRC is set to 1 so that the data will not be erased.

Figure 7-12
A Flow Graph of Smoothing Processing
(Computation Part)



### 7.2.1.4 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;       SMOOTH OPERATION
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  L       =       64              ;
10: EQUATE  H       =       32              ;
11: EQUATE  V       =       512             ;
12: EQUATE  R       =       2               ;
13: ;
14: EQUATE  HOST    =       0               ;
15: EQUATE  READ    =       4               ;
16: EQUATE  WRITE   =       5               ;
17: ;
18: EQUATE  STARTS  =       0               ;
19: EQUATE  STARTD  =       32              ;
20: ;
21: ;****************************************
22: ;
23: ;       INPUT-OUTPUT
24: ;
25: ;----------------------------------------
26: ;
27: INPUT   LSA0,   LDA0,   RDATA AT 0       ;
28: ;
29: OUTPUT  RDAT,   WDAT,   WADR,   PEND     ;
30: ;
31: ;****************************************
32: ;
33: ;       LINK TABLE
34: ;
35: ;----------------------------------------
36: ;
37: LINK    X1,     X2              =       FT1     (LDA0           )       ;
38: LINK                            =       FT2     (X2             )       ;
39: LINK    X3,     X4,     X5      =       FT3     (X12,   X1      )       ;
40: LINK    X6,     X7,     X8      =       FT4     (X13,   X3      )       ;
41: LINK    X9,     X10,    X11     =       FT5     (X14,   X6      )       ;
42: LINK    X12                     =       FT6     (X4,    X8      )       ;
43: LINK    X13                     =       FT7     (X7,    X11     )       ;
44: LINK    X14                     =       FT8     (X10,   X29     )       ;
45: LINK    X19,    X20,    X21     =       FT10    (X31,   LSA0    )       ;
46: LINK    X22,    X23,    X24     =       FT11    (X32,   X19     )       ;
47: LINK    X25,    X26,    X27     =       FT12    (X33,   X22     )
48: LINK    X28,    X29             =       FT13    (X25            )
49: LINK    X30                     =       FT14    (X17,   X21     )
50: LINK    X31                     =       FT15    (X20,   X24     )
51: LINK    X32                     =       FT16    (X23,   X27     )
52: LINK    X33                     =       FT17    (X26,   X96     )
53: LINK    X34                     =       FT18    (X18            )
54: LINK    X35                     =       FT59    (X34            )
55: LINK    X36                     =       FT73    (       ,X35    )
56: LINK    X37                     =       FT21    (X36            )       ;
57: LINK    X38                     =       FT58    (X37            )       ;
58: LINK    X39                     =       FT23    (X38            )       ;
59: LINK    X40,    X41             =       FT24    (X39            )       ;
60: LINK    X45                     =       FT25    (X40,   X5      )       ;
```

```
 61: LINK    X42.    X43.    X44     =     FT26   (X45          )   ;
 62: LINK    RDAT                    =     FT27   (X28          )   ;
 63: LINK    WDAT.   WADR            =     FT28   (X46.   X9    )   ;
 64: LINK    PEND                    =     FT29   (X44          )   ;
 65: ;
 66: LINK    X47                     =     FT30   (RDATA        )   ;
 67: LINK    X48.    X49.    X50     =     FT31   (       .X47  )   ;
 68: LINK    X51.    X52             =     FT32   (X48          )   ;
 69: LINK    X53.    X54             =     FT32   (X49          )   ;
 70: LINK    X55.    X56             =     FT32   (X50          )   ;
 71: LINK    X57                     =     FT35   (X51          )   ;
 72: LINK    X58                     =     FT38   (X57          )   ;
 73: LINK    X59                     =     FT41   (X58          )   ;
 74: LINK    X60                     =     FT44   (X59          )   ;
 75: LINK    X61                     =     FT36   (X53          )   ;
 76: LINK    X62                     =     FT39   (X61          )   ;
 77: LINK    X63                     =     FT42   (X62          )   ;
 78: LINK    X67                     =     FT45   (X63          )   ;
 79: LINK    X69                     =     FT47   (X67.   X60   )   ;
 80: LINK    X64                     =     FT37   (X55          )   ;
 81: LINK    X65                     =     FT40   (X64          )   ;
 82: LINK    X66                     =     FT43   (X65          )   ;
 83: LINK    X68                     =     FT44   (X66          )   ;
 84: LINK    X70                     =     FT48   (X68.   X69   )   ;
 85: LINK    X71.    X72             =     FT49   (X70          )   ;
 86: LINK    X73                     =     FT50   (X71          )   ;
 87: LINK    X74                     =     FT51   (X72          )   ;
 88: LINK    X75                     =     FT52   (X73          )   ;
 89: LINK    X75                     =     FT53   (X74          )   ;
 90: LINK    X77                     =     FT54   (X75          )   ;
 91: LINK    X78. X79. X80. X81. X82 =     FT55   (X77          )   ;
 92: LINK    X42                     =     FT56   (X78          )   ;
 93: LINK                            =     FT57   (X42          )   ;
 94: LINK    X84                     =     FT58   (X79          )   ;
 95: LINK    X85                     =     FT59   (X84          )   ;
 96: LINK    X41                     =     FT60   (X85.   X43   )   ;
 97: LINK    X46                     =     FT30   (X41          )   ;
 98: LINK    X87                     =     FT62   (X52          )   ;
 99: LINK    X88                     =     FT65   (X87          )   ;
100: LINK    X89                     =     FT68   (X88.   X80   )   ;
101: LINK    X90                     =     FT62   (X54          )   ;
102: LINK    X91                     =     FT65   (X90          )   ;
103: LINK    X92                     =     FT69   (X91.   X81   )   ;
104: LINK    X93                     =     FT62   (X56          )   ;
105: LINK    X94                     =     FT65   (X93          )   ;
106: LINK    X95                     =     FT70   (X94.   X82   )   ;
107: LINK                            =     FT71   (X89          )   ;
108: LINK                            =     FT72   (X92          )   ;
109: LINK    X96                     =     FT73   (       .X95  )   ;
110: ;
```

```
;****************************************
;
;       FUNCTION TABLE
;
;----------------------------------------
;
FUNCTION    FT1    =   COPYC    (XY       ),     RDCYCS  (T1,     1)
FUNCTION    FT2    =   WRCYCS   (T1,     1)
FUNCTION    FT3    =   COPYBK   (1,      0),     CNTGE   (R       )
FUNCTION    FT4    =   COPYBK   (1,      L),     CNTGE   (V/R     )
FUNCTION    FT5    =   COPYBK   (1,      1),     CNTGE   (H       )
FUNCTION    FT6    =   QUEUE    (QUE1,   1)
FUNCTION    FT7    =   QUEUE    (QUE2,   1)
FUNCTION    FT8    =   QUEUE    (QUE3,   1)
FUNCTION    FT10   =   COPYBK   (1,      0),     CNTGE   (R       )
FUNCTION    FT11   =   COPYBK   (1,      L),     CNTGE   (V/R     )
FUNCTION    FT12   =   COPYBK   (1,      1),     CNTGE   (H       )
FUNCTION    FT13   =   COPYBK   (3,      L)
FUNCTION    FT14   =   QUEUE    (QUE4,   1)
FUNCTION    FT15   =   QUEUE    (QUE5,   1)
FUNCTION    FT16   =   QUEUE    (QUE6,   1)
FUNCTION    FT17   =   QUEUE    (QUE7,   1)
FUNCTION    FT18   =   RDCYCS   (ZERO,   1)
FUNCTION    FT21   =   SHR,                      RDCYCS  (ONE,    1)
FUNCTION    FT23   =   OR,                       RDCYCS  (LSB3,   1)
FUNCTION    FT24   =   COPYBK   (1,      0)
FUNCTION    FT25   =   QUEUE    (QUE14,  1)
FUNCTION    FT26   =   COPYM    (3,      0)
FUNCTION    FT27   =   OUT1     (READ,   0)
FUNCTION    FT28   =   OUT2     (WRITE, 20H, 0), QUEUE   (QUE8,  16)
FUNCTION    FT29   =   OUT1     (HOST,   0)
;
FUNCTION    FT30   =   NOT      (CNOP     ),     RDCYCS  (T1,     1)
FUNCTION    FT31   =   DIST     (3        )
FUNCTION    FT32   =   COPYBK   (8,      0)
FUNCTION    FT35   =   AND,                      RDCYCS  (MASK,   8)
FUNCTION    FT36   =   AND,                      RDCYCS  (MASK,   8)
FUNCTION    FT37   =   AND,                      RDCYCS  (MASK,   8)
FUNCTION    FT38   =   SHR,                      RDCYCS  (SHTTBL,8)
FUNCTION    FT39   =   SHR,                      RDCYCS  (SHTTBL,8)
FUNCTION    FT40   =   SHR,                      RDCYCS  (SHTTBL,8)
FUNCTION    FT41   =   OR,                       RDCYCS  (LSB1.   8)
FUNCTION    FT42   =   OR,                       RDCYCS  (LSB2,   8)
FUNCTION    FT43   =   OR,                       RDCYCS  (LSB3,   8)
FUNCTION    FT44   =   RDIDX    (FILT1    )
FUNCTION    FT45   =   RDIDX    (FILT2    )
FUNCTION    FT47   =   ADD,                      QUEUE   (QUE9,   8)
FUNCTION    FT48   =   ADD,                      QUEUE   (QUE10,  8)
FUNCTION    FT49   =   SHR      (XY       ),     RDCYCS  (EIGHT,  1)
FUNCTION    FT50   =   CMPNOM   (GT       ),     RDCYCS  (THR1,   1)
FUNCTION    FT51   =   CMPNOM   (GT       ),     RDCYCS  (THR2.   1)
FUNCTION    FT52   =   MUL      (Y        ),     RDCYCS  (EVEN,   8)
FUNCTION    FT53   =   MUL      (Y        ),     RDCYCS  (ODD,    8)
FUNCTION    FT54   =   ACC,                      COUNT   (16       )
FUNCTION    FT55   =   COPYM    (5,      0)
FUNCTION    FT56   =   AND,                      RDCYCS  (RMASK,  1)
FUNCTION    FT57   =   WRCYCS   (ANS,    2),     1
FUNCTION    FT58   =   AND,                      RDCYCS  (ONE,    1)
FUNCTION    FT59   =   OR,                       RDCYCS  (ANS,    2)
FUNCTION    FT60   =   CUT      (1        )
FUNCTION    FT62   =   AND,                      RDCYCS  (THREE,  1)
FUNCTION    FT65   =   SHL,                      RDCYCS  (TWO,    1)
FUNCTION    FT68   =   QUEUE    (QUE11,  1)
FUNCTION    FT69   =   QUEUE    (QUE12,  1)
FUNCTION    FT70   =   QUEUE    (QUE13,  1)
FUNCTION    FT71   =   WRCYCS   (LSB1,   1)
FUNCTION    FT72   =   WRCYCS   (LSB2,   1)
FUNCTION    FT73   =   WRCYCS   (LSB3,   1)
;
```

```
;****************************************
;
;       DATA MEMORY
;
;----------------------------------------
;
MEMORY  T1      =       0                                       ;
MEMORY  MASK    =       0C000H, 0F000H, 3C00H,  0F00H,
                        03C0H,  00F0H,  003CH,  000FH   ;
MEMORY  SHTTBL  =       14, 12, 10, 8,  6,  4,  2,  0   ;
MEMORY  LSB1    =       0,  0,  0,  0,  0,  0,  0,  0   ;
MEMORY  LSB2    =       0,  0,  0,  0,  0,  0,  0,  0   ;
MEMORY  LSB3    =       0,  0,  0,  0,  0,  0,  0,  0   ;
MEMORY  FILT1   =       0H,     1H,     102H,   103H,
                        201H,   202H,   303H,   304H,
                        100H,   101H,   202H,   203H,
                        301H,   302H,   403H,   404H    ;
MEMORY  FILT2   =       0H,     2H,     204H,   206H,
                        402H,   404H,   606H,   608H,
                        200H,   202H,   404H,   406H,
                        602H,   604H,   806H,   808H    ;
MEMORY  ZERO    =       0                                       ;
MEMORY  ONE     =       1                                       ;
MEMORY  TWO     =       2                                       ;
MEMORY  THREE   =       3                                       ;
MEMORY  EIGHT   =       8                                       ;
MEMORY  THR1    =       5H                                      ;
MEMORY  THR2    =       500H                                    ;
MEMORY  ODD     =       0001H,  4000H,  1000H,  0400H,
                        0100H,  0040H,  0010H,  0004H   ;
MEMORY  EVEN    =       8000H,  2000H,  0800H,  0200H,
                        0080H,  0020H,  0008H,  0002H   ;
MEMORY  RMASK   =       0FFFEH                                  ;
MEMORY  ANS     =       AREA    (2      )                       ;
MEMORY  QUE1    =       AREA    (1      )                       ;
MEMORY  QUE2    =       AREA    (1      )                       ;
MEMORY  QUE3    =       AREA    (1      )                       ;
MEMORY  QUE4    =       AREA    (1      )                       ;
MEMORY  QUE5    =       AREA    (1      )                       ;
MEMORY  QUE6    =       AREA    (1      )                       ;
MEMORY  QUE7    =       AREA    (1      )                       ;
MEMORY  QUE8    =       AREA    (16     )                       ;
MEMORY  QUE9    =       AREA    (8      )                       ;
MEMORY  QUE10   =       AREA    (8      )                       ;
MEMORY  QUE11   =       AREA    (1      )                       ;
MEMORY  QUE12   =       AREA    (1      )                       ;
MEMORY  QUE13   =       AREA    (1      )                       ;
MEMORY  QUE14   =       AREA    (1      )                       ;
;
;****************************************
;
;       START
;
;----------------------------------------
;
START                                                   ;
;
DATA    EXEC    (IPP,   LDA0,   STARTD  )               ;
DATA    EXEC    (IPP,   LSA0,   STARTS  )               ;
;
END                                                     ;
```

### 7.2.2 Thinning

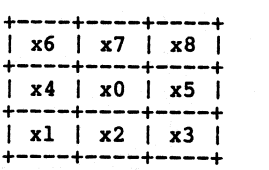
#### 7.2.2.1 Processing Explained

The purpose of this procedure is to erase the contours of a binary image line by line, using the adjoining point data. By repeatedly executing this program it is possible to determine the center of a linear graph.

#### 7.2.2.2 Algorithm

The point x0' after transformation of point x0 is determined by performing the operation F on the adjoining eight points, as shown in Figure 7-13.

Figure 7-13
Determination of Object Point x0'

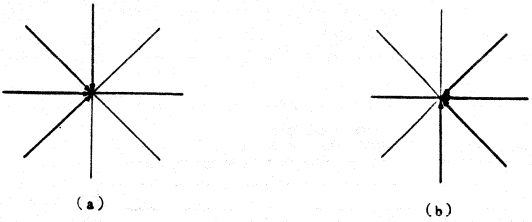



$$x0' = F(x0, x1, x2, x3, x4, x5, x6, x7, x8)$$
$$= x0 \cdot G(x1, x2, x3, x4, x5, x6, x7, x8)$$

where G(.) is a function yielding 1 or 0, depending on the particular pattern of the adjoining eight points. Hence, to make object point x0' = 0, you need only plug in 0 for G. To make x0' = x0, plug in 1 for G.

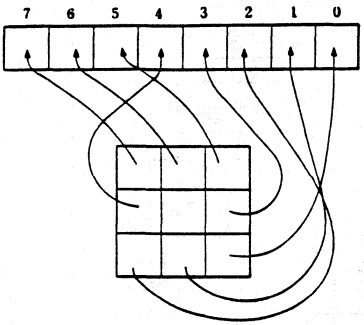
A thinning program working with a 0 background assigns 0 or 1 to each of the 256 patterns generated by 8 points (points x1 through x8), and executes the processing by means of a table lookup. However, instead of erasing points in all eight directions, the program performs the erasure from four directions in two passes. This is because, in parallel processing as opposed to serial processing, it would be difficult to perform a thinning, such as reducing a 2-dot line into a 1-dot line, if the points were erased from eight directions.

Figure 7-14
Direction of Erasure




For table lookup, the adjoining eight points are ordered as shown in Figure 7-15.

Figure 7-15
Ordering of Adjoining Eight Points

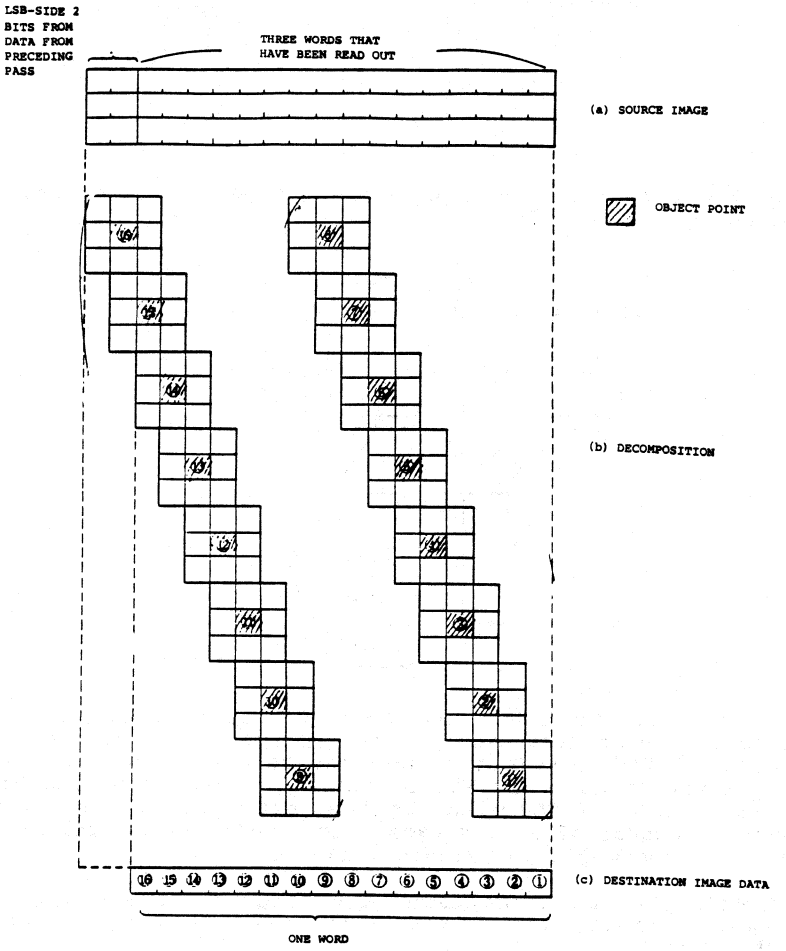



Unlike the smoothing and edge detection program, this program cannot determine several points at the same time. A total of 256 tables are required to determine one point. Since the DM has only 512 locations, 18 bits (16 bits for a word plus the 2 LSB bits obtained previously) are divided into 16 groups of 3 bits each, and a point is determined for each of these groups (Figure 7-16).

In this program 15 patterns are used to erase a point in Figure 7-14(a) and Figure 7-14(b), for a total of 30 patterns.

The 30 patterns used in this program are not necessarily the best possible patterns for producing good quality images. In some cases it would be necessary to generate and try various patterns to arrive at an appropriate pattern for the particular application.

## Figure 7-16
## Data Decomposition and Assembly



Values produced by table lookups can consist of values for Figure 7-14(a) and those for Figure 7-14(b). The appropriate values should be used.

This program uses a CHGDIR token, which has a value of 0 or 8, to control the number of right shifts.

#### 7.2.2.3 Flow Graph Explained

Shown in Figure 7-17 is a thinning flow graph (computation part) that, together with the image memory address generation part flow graph discussed in Section 7.1, makes up the thinning program.

FT30 complements image data bits from 1 to 0 and 0 to 1 in cases where the background is 1.

FT21, FT31, FT32 and FT35 through FT51 determine transformed object point data from source image data.

FT31 distributes source image data. FT32 and FT35 through FT43 create data (Figure 7-16(b)) to serve as units of computation.

This thinning requires the generation of data for the table lookup as shown in Figure 7-15. For this purpose, data are directed to bit positions 7, 6, and 5 in FT38 (Figure 7-15) and to the lowest three bit positions in FT39 and FT40.

FT41 through FT43 treat data as units of computation by adding to each datum either a 1 bit or the 2 LSB-side bits in the data from the preceding pass (Note 1).

FT44, FT45, FT46, and FT48 through FT50 are used to determine the values of G, shown in Figure 7-13. The bits 7-5 and bits 2-0 in Figure 7-15 are generated in FT41 and FT43. FT45 makes two copies of data (Note 2) and generates bits 4 and 3 by performing a table lookup (MTB1 in the DM).

After the generation of lookup table data, the values of G are determined through a table lookup in FT49.

However, since the values so obtained include both values for Figure 7-14(a) and Figure 7-14(b), FT50 chooses between these sets of values.

FT47 and FT21 extract data on point x0 from the other copy of data made in FT45, and FT51 determines the computed value of F from the extracted data.

FT52 and FT53 reconstitute the object point x0' after the transformation of a word which is organized as indicated in Figure 7-16(c).

FT54 copies the destination image data sought and furnishes this data for processing in the subsequent steps.

FT55 and FT56 add 0 to the LSBs of the low-order 15 bits (bits 15-1 in Figure 7-16(c)) of the destination image data obtained in FT53, writes the results to the DM (the ANS area), and uses them in the subsequent computation.

FT57, FT59, and FT60 extract bit 16 (the MSB) from the destination image data, make it into LSB data, add to it the high-order 15 bits of data which was stored in the DM in the previous pass, and pass the results onto the address generation part.

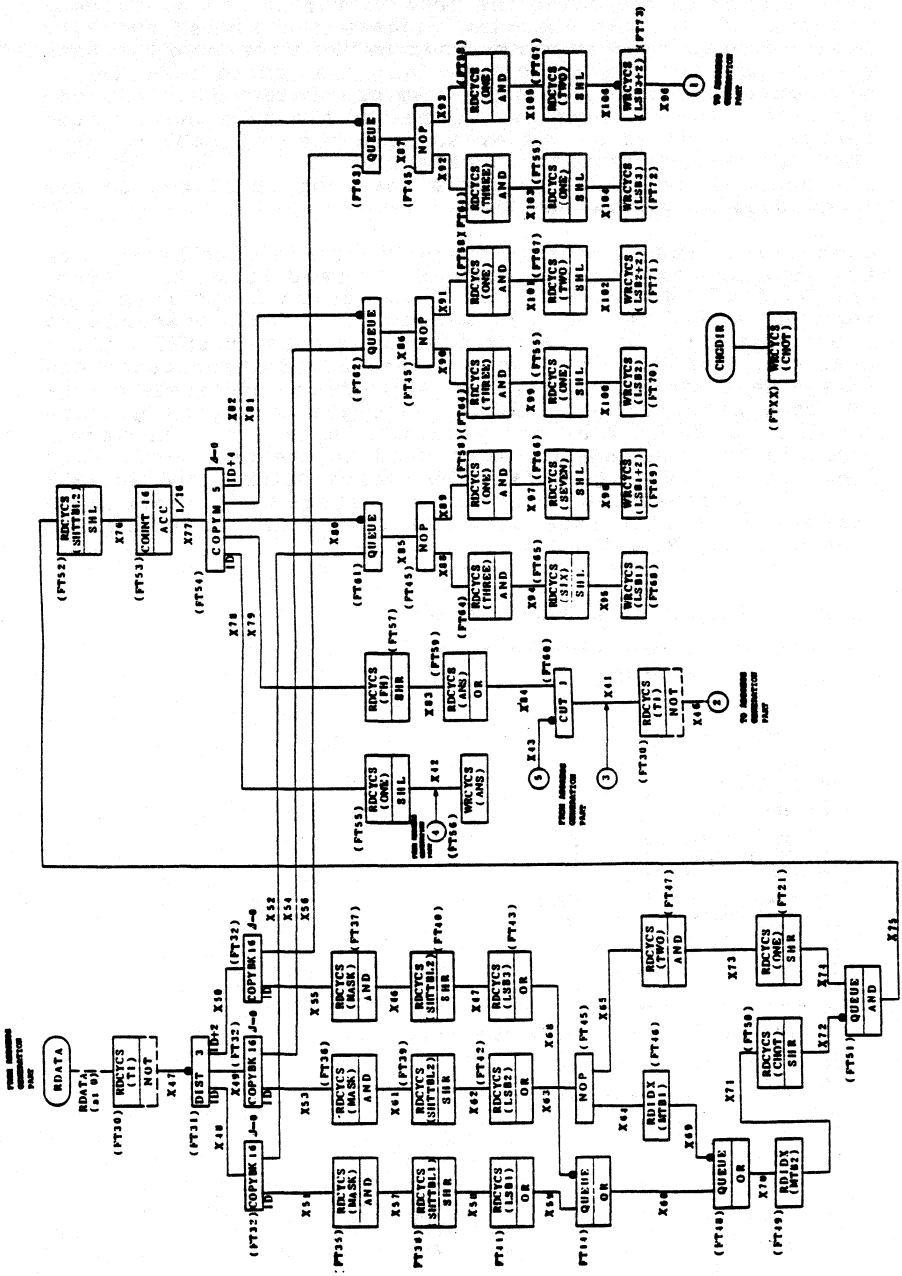
The procedure employed in this program for handling the leading and ending edges is that same as that used in the smoothing program. Refer to Section 7.2.1 for further details.

FT61 - FT73 are nodes used for storing the lowest two bits and the LSB bit of the read data copied in FT32 in the LSB1 - LSB3 of the DM, for use in subsequent computations (Note 1). FT61 - FT63 are concerned with keeping these operations in synchronization with the computational steps.

FT45 perform copying operations by using NOP (Note 2).

Note 1: See Section 7.2.2.4.
Note 2: Generally COPYBK is used to copy data. This program uses the XX output of the NOP instruction to preclude overflows on the Generate Queue of the uPD7281.

Figure 7-17
A Thinning Flow Graph
(Computation Part)



### 7.2.2.4 Tips on Writing Flow Graphs

In the smoothing program two LSB bits of source image data were stored in the DM of the uPD7281 for use in subsequent processing. In this thinning program the lowest two bits from a preceding pass can be used in two ways: use two bits to generate 3-bit data, or use only one bit to generate 3-bit data. Therefore, this program provides separate 2-bit and 1-bit storage locations in the DM. Although these spaces can be considered as one area, comprised of LSB1 - LSB3, they can be differentiated by means of a table which will be discussed later. This requires that copies of the source image data be generated.

When 3-bit data is made in conjunction with the 1-bit data from the preceding pass, the data is read from LSB1, LSB2, and LSB3 of the DM, and the OR operation is performed on them. It would be a simple matter if it were possible to place the LSB1 bit from the preceding pass at LSB1 + 1 (or LSB2 + 1, or LSB3 + 1). However, the WRCYCS instruction can assign base addresses only in even-numbered addresses of the DM. To provide for this fact, the space for storing 1-bit data is moved back by one position to reverse the order. Accordingly, the MASK table is used in creating 3-bit data from the source image data. The tables SHTTBL1 and SHTTBL2 used in resequencing the data are ordered differently to reflect this fact (see Line 193 in the Assembler Source listing: ...LSB1 + 2 ...).

### 7.2.2.5 Assembler Source Listing

```
  1: ;*************************************
  2: ;
  3: ;       THIN OPERATION
  4: ;
  5: ;-------------------------------------
  6: ;
  7: MODULE  IPP     =       8       ;
  8: ;
  9: EQUATE  L       =       64      ;
 10: EQUATE  H       =       32      ;
 11: EQUATE  V       =       512     ;
 12: EQUATE  R       =       2       ;
 13: ;
 14: EQUATE  HOST    =       0       ;
 15: EQUATE  READ    =       4       ;
 16: EQUATE  WRITE   =       5       ;
 17: ;
 18: EQUATE  STARTS  =       0FF80H  ;
 19: EQUATE  STARTD  =       20H     ;
 20: ;
 21: ;*************************************
 22: ;
 23: ;       INPUT-OUTPUT
 24: ;
 25: ;-------------------------------------
 26: ;
 27: INPUT   LSA0.   LDA0.   CHGDIR. RDATA AT 0      ;
 28: ;
 29: OUTPUT  RDAT.   WDAT.   WADR.   PEND            ;
 30: ;
 31: ;*************************************
 32: ;
 33: ;       LINK TABLE
 34: ;
 35: ;-------------------------------------
 36: ;
 37: LINK    X1.     X2              =       FT1     (LDA0           )       ;
 38: LINK                            =       FT2     (X2             )       ;
 39: LINK    X3.     X4.     X5      =       FT3     (X12.   X1      )       ;
 40: LINK    X6.     X7.     X8      =       FT4     (X13.   X3      )       ;
 41: LINK    X9.     X10.    X11     =       FT5     (X14.   X6      )       ;
```

```
42: LINK    X12                          =    FT6   (X4.   X8    )
43: LINK    X13                          =    FT7   (X7.   X11   )
44: LINK    X14                          =    FT8   (X10.  X29   )
45: LINK    X19.   X20.   X21            =    FT10  (X31.  LSA0  )
46: LINK    X22.   X23.   X24            =    FT11  (X32.  X19   )
47: LINK    X25.   X26.   X27            =    FT12  (X33.  X22   )
48: LINK    X28.   X29                   =    FT13  (X25         )
49: LINK    X31                          =    FT15  (X20.  X24   )
50: LINK    X32                          =    FT16  (X23.  X27   )
51: LINK    X33                          =    FT17  (X26.  X96   )
52: LINK    X34                          =    FT18  (X21         )
53: LINK    X35                          =    FT59  (X34         )
54: LINK    X36                          =    FT73  (      .X35  )
55: LINK    X37                          =    FT21  (X36         )
56: LINK    X38                          =    FT58  (X37         )
57: LINK    X39                          =    FT23  (X38         )
58: LINK    X40.   X41                   =    FT24  (X39         )
59: LINK    X45                          =    FT25  (X40.  X5    )
60: LINK    X42.   X43.   X44            =    FT26  (X45         )
61: LINK    RDAT                         =    FT27  (X28         )
62: LINK    WDAT.  WADR                  =    FT28  (X46.  X9    )
63: LINK    PEND                         =    FT29  (X44         )
64: ;
65: LINK    X47                          =    FT30  (RDATA       )
66: LINK    X48.   X49.   X50            =    FT31  (      .X47  )
67: LINK    X51.   X52                   =    FT32  (X48         )
68: LINK    X53.   X54                   =    FT32  (X49         )
69: LINK    X55.   X56                   =    FT32  (X50         )
70: LINK    X57                          =    FT35  (X51         )
71: LINK    X58                          =    FT38  (X57         )
72: LINK    X59                          =    FT41  (X58         )
73: LINK    X60                          =    FT44  (X59.  X68   )
74: LINK    X61                          =    FT36  (X53         )
75: LINK    X62                          =    FT39  (X61         )
76: LINK    X63                          =    FT42  (X62         )
77: LINK    X64.   X65                   =    FT45  (X63         )
78: LINK    X66                          =    FT37  (X55         )
79: LINK    X67                          =    FT40  (X66         )
80: LINK    X68                          =    FT43  (X67         )
81: LINK    X69                          =    FT46  (X64         )
82: LINK    X70                          =    FT48  (X60.  X69   )
83: LINK    X71                          =    FT49  (X70         )
84: LINK    X72                          =    FT50  (X71         )
85: LINK    X73                          =    FT47  (X65         )
86: LINK    X74                          =    FT21  (X73         )
87: LINK    X75                          =    FT51  (X74.  X72   )
88: LINK    X76                          =    FT52  (X75         )
89: LINK    X77                          =    FT53  (X76         )
90: LINK    X78.   X79.   X80.   X81.  X82 =  FT54  (X77         )
91: LINK    X42                          =    FT55  (X78         )
92: LINK    X83                          =    FT57  (X79         )
93: LINK    X84                          =    FT59  (X83         )
94: LINK    X85                          =    FT61  (X52.  X80   )
95: LINK    X86                          =    FT62  (X54.  X81   )
96: LINK    X87                          =    FT63  (X56.  X82   )
97: LINK    X88.   X89                   =    FT45  (X85         )
98: LINK    X90.   X91                   =    FT45  (X86         )
99: LINK    X92.   X93                   =    FT45  (X87         )
100: LINK   X94                          =    FT64  (X88         )
101: LINK   X95                          =    FT65  (X94         )
102: LINK   X96                          =    FT73  (      .X106 )
103: LINK   X97                          =    FT58  (X89         )
104: LINK   X98                          =    FT66  (X97         )
105: LINK   X99                          =    FT64  (X90         )
106: LINK   X100                         =    FT55  (X99         )
107: LINK   X101                         =    FT58  (X91         )
108: LINK   X102                         =    FT67  (X101        )
109: LINK   X103                         =    FT64  (X92         )
110: LINK   X104                         =    FT55  (X103        )
111: LINK   X105                         =    FT58  (X93         )
112: LINK   X106                         =    FT67  (X105        )
113: LINK                                =    FT68  (X95         )
114: LINK                                =    FT69  (X98         )
115: LINK                                =    FT70  (X100        )
116: LINK                                =    FT71  (X102        )
117: LINK                                =    FT72  (X104        )
118: LINK   X42                          =    FT55  (X78         )
119: LINK                                =    FT56  (X42         )
120: LINK   X41                          =    FT60  (X84.  X43   )
```

```
121: LINK    X46                    =       FT30    (X41            )       ;
122: ;
123: LINK                           =       FTXX    (CHGDIR         )       ;
124: ;
125: ;**************************************
126: ;
127: ;       FUNCTION TABLE
128: ;
129: ;--------------------------------------
130: ;
131: FUNCTION    FT1     =   COPYC   (XY      ),      RDCYCS  (T1.    1)      ;
132: FUNCTION    FT2     =   WRCYCS  (T1.    1)      ;
133: FUNCTION    FT3     =   COPYBK  (1.     0),     CNTGE   (R      )       ;
134: FUNCTION    FT4     =   COPYBK  (1.     L),     CNTGE   (V/R    )       ;
135: FUNCTION    FT5     =   COPYBK  (1.     1),     CNTGE   (H      )       ;
136: FUNCTION    FT6     =   QUEUE   (QUE1.  1)      ;
137: FUNCTION    FT7     =   QUEUE   (QUE2.  1)      ;
138: FUNCTION    FT8     =   QUEUE   (QUE3.  1)      ;
139: FUNCTION    FT10    =   COPYBK  (1.     0),     CNTGE   (R      )       ;
140: FUNCTION    FT11    =   COPYBK  (1.     L),     CNTGE   (V/R    )       ;
141: FUNCTION    FT12    =   COPYBK  (1.     1),     CNTGE   (H      )       ;
142: FUNCTION    FT13    =   COPYBK  (3.     L)      ;
143: FUNCTION    FT15    =   QUEUE   (QUE5.  1)      ;
144: FUNCTION    FT16    =   QUEUE   (QUE6.  1)      ;
145: FUNCTION    FT17    =   QUEUE   (QUE7.  1)      ;
146: FUNCTION    FT18    =   RDCYCS  (ZERO.  1)      ;
147: FUNCTION    FT21    =   SHR.                    RDCYCS  (ONE.   1)      ;
148: FUNCTION    FT23    =   OR.                     RDCYCS  (LSB3.  1)      ;
149: FUNCTION    FT24    =   COPYBK  (1.     0)      ;
150: FUNCTION    FT25    =   QUEUE   (QUE4.  1)      ;
151: FUNCTION    FT26    =   COPYM   (3.     0)      ;
152: FUNCTION    FT27    =   OUT1    (READ.  0)      ;
153: FUNCTION    FT28    =   OUT2    (WRITE. 2000. 0), QUEUE   (QUE8. 16)      ;
154: FUNCTION    FT29    =   OUT1    (HOST.  0)      ;
155: ;
156: FUNCTION    FT30    =   NOT     (CNOP   ),      RDCYCS  (T1.    1)      ;
157: FUNCTION    FT31    =   DIST    (3      )       ;
158: FUNCTION    FT32    =   COPYBK  (16.    0)      ;
159: FUNCTION    FT35    =   AND.                    RDCYCS  (MASK. 16)      ;
160: FUNCTION    FT36    =   AND.                    RDCYCS  (MASK. 16)      ;
161: FUNCTION    FT37    =   AND.                    RDCYCS  (MASK. 16)      ;
162: FUNCTION    FT38    =   SHR.                    RDCYCS  (SHTTBL1. 16)   ;
163: FUNCTION    FT39    =   SHR.                    RDCYCS  (SHTTBL2. 16)   ;
164: FUNCTION    FT40    =   SHR.                    RDCYCS  (SHTTBL2. 16)   ;
165: FUNCTION    FT41    =   OR.                     RDCYCS  (LSB1. 16)      ;
166: FUNCTION    FT42    =   OR.                     RDCYCS  (LSB2. 16)      ;
167: FUNCTION    FT43    =   OR.                     RDCYCS  (LSB3. 16)      ;
168: FUNCTION    FT44    =   OR.                     QUEUE   (QUE9. 16)      ;
169: FUNCTION    FT45    =   NOP     (XX      )      ;
170: FUNCTION    FT46    =   RDIDX   (MTB1    )      ;
171: FUNCTION    FT47    =   AND.                    RDCYCS  (TWO.   1)      ;
172: FUNCTION    FT48    =   OR.                     QUEUE   (QUE10.16)      ;
173: FUNCTION    FT49    =   RDIDX   (MTB2    )      ;
174: FUNCTION    FT50    =   SHR.                    RDCYCS  (CHOT.  1)      ;
175: FUNCTION    FT51    =   AND.                    QUEUE   (QUE11.16)      ;
176: FUNCTION    FT52    =   SHL.                    RDCYCS  (SHTTBL2. 16)   ;
177: FUNCTION    FT53    =   ACC.                    COUNT   (16     )       ;
178: FUNCTION    FT54    =   COPYM   (5.     0)      ;
179: FUNCTION    FT55    =   SHL.                    RDCYCS  (ONE.   1)      ;
180: FUNCTION    FT56    =   WRCYCS  (ANS.   2),     1                       ;
181: FUNCTION    FT57    =   SHR.                    RDCYCS  (FH.    1)      ;
182: FUNCTION    FT58    =   AND.                    RDCYCS  (ONE.   1)      ;
183: FUNCTION    FT59    =   OR.                     RDCYCS  (ANS.   2)      ;
184: FUNCTION    FT60    =   CUT     (1       )      ;
185: FUNCTION    FT61    =   QUEUE   (QUE12. 1)      ;
186: FUNCTION    FT62    =   QUEUE   (QUE13. 1)      ;
187: FUNCTION    FT63    =   QUEUE   (QUE14. 1)      ;
188: FUNCTION    FT64    =   AND.                    RDCYCS  (THREE. 1)      ;
189: FUNCTION    FT65    =   SHL.                    RDCYCS  (SIX.   1)      ;
190: FUNCTION    FT66    =   SHL.                    RDCYCS  (SEVEN. 1)      ;
191: FUNCTION    FT67    =   SHL.                    RDCYCS  (TWO.   1)      ;
192: FUNCTION    FT68    =   WRCYCS  (LSB1.  1)      ;
193: FUNCTION    FT69    =   WRCYCS  (LSB1+2.1)      ;
194: FUNCTION    FT70    =   WRCYCS  (LSB2.  1)      ;
195: FUNCTION    FT71    =   WRCYCS  (LSB2+2.1)      ;
196: FUNCTION    FT72    =   WRCYCS  (LSB3.  1)      ;
197: FUNCTION    FT73    =   WRCYCS  (LSB3+2.1)      ;
198: ;
199: FUNCTION    FTXX    =   WRCYCS  (CHOT.  1)      ;
200: ;
```

```
;****************************************
;
;       DATA MEMORY
;
;----------------------------------------
;
MEMORY  T1      =       0                                       ;
MEMORY  CHOT    =       0                                       ;
MEMORY  MASK    =       08000H, 0E000H, 0C000H, 07000H,
                        03800H, 01C00H, 00E00H, 00700H,
                        00380H, 001C0H, 000E0H, 00070H,
                        00038H, 0001CH, 00008H, 00007H  ;
MEMORY  SHTTBL1 =       10, 8, 9, 7, 6, 5, 4, 3,
                         2, 1, 0,-1,-2,-3,-4,-5         ;
MEMORY  SHTTBL2 =       15,13,14,12,11,10, 9, 8,
                         7, 6, 5, 4, 3, 2, 1, 0         ;
MEMORY  LSB1    =       0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0          ;
MEMORY  LSB2    =       0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0          ;
MEMORY  LSB3    =       0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0          ;
MEMORY  ZERO    =       0                                       ;
MEMORY  ONE     =       1                                       ;
MEMORY  TWO     =       2                                       ;
MEMORY  THREE   =       3                                       ;
MEMORY  SIX     =       6                                       ;
MEMORY  SEVEN   =       7                                       ;
MEMORY  FH      =       15                                      ;
MEMORY  MTB1    =       00H, 00H, 00H, 00H, 10H, 10H, 10H, 10H  ;
MEMORY  MTB2    =       101H,101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,100H,101H,101H,101H,100H,
                        101H,101H,101H,101H,101H,101H,001H,001H,
                        101H,101H,101H,101H,101H,101H,101H,100H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,100H,101H,101H,101H,100H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,100H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        100H,100H,101H,100H,101H,101H,101H,100H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,100H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        001H,101H,101H,101H,001H,101H,001H,001H,
                        101H,101H,101H,101H,101H,101H,101H,101H,
                        100H,100H,101H,100H,101H,101H,101H,100H,
                        001H,101H,101H,101H,001H,101H,001H,001H,
                        001H,101H,101H,101H,001H,001H,101H,101H;
MEMORY  ANS     =       AREA    (2      )       ;
MEMORY  QUE1    =       AREA    (1      )       ;
MEMORY  QUE2    =       AREA    (1      )       ;
MEMORY  QUE3    =       AREA    (1      )       ;
MEMORY  QUE4    =       AREA    (1      )       ;
MEMORY  QUE5    =       AREA    (1      )       ;
MEMORY  QUE6    =       AREA    (1      )       ;
MEMORY  QUE7    =       AREA    (1      )       ;
MEMORY  QUE8    =       AREA    (16     )       ;
MEMORY  QUE9    =       AREA    (16     )       ;
MEMORY  QUE10   =       AREA    (16     )       ;
MEMORY  QUE11   =       AREA    (16     )       ;
MEMORY  QUE12   =       AREA    (1      )       ;
MEMORY  QUE13   =       AREA    (1      )       ;
MEMORY  QUE14   =       AREA    (1      )       ;
;
;****************************************
```

```
280: ;
281: ;       START
282: ;
283: ;-------------------------------------
284: ;
285: START                                                    ;
286: ;
287: DATA    EXEC    (IPP.   CHGDIR. 0       )       ;
288: DATA    EXEC    (IPP.   LDA0.   STARTD  )       ;
289: DATA    EXEC    (IPP.   LSA0.   STARTS  )       ;
290: ;
291: END     ;
292:
```

### 7.2.3 Edge Detection

#### 7.2.3.1 Processing Explained

The purpose of this procedure is to extract contour lines in which binary image data changes from 0 to 1 and from 1 to 0. Contour lines produced by this program form an 8-way connected figure* (a line figure consisting of points connected in eight directions).

* : An 8-way connected figure is one in which any point of those comprising the figure has an adjoining point that is comprised of line segments in 8 directions, vertical, horizontal, and diagonal. In contrast, a figure whose points are comprised of vertical and horizontal lines only is called a 4-way connected figure.

#### 7.2.3.2 Algorithm

Figure 7-18

| | | |
|---|---|---|
| x6 | x7 | x8 |
| x4 | x0 | x5 |
| x1 | x2 | x3 |

In this program transformed object point x0' is determined by:

$$x0' = F(x0, x1, x2, x3, x4, x5, x6, x7, x8)$$

$$= x0 \cdot \overline{x2 \cdot x5 \cdot x7 \cdot x4}$$

In actual processing the program processes four object points at a time to optimize processing speed. This requires a three by six area, shown in Figure 7-19. (1) Specifically, 4-bit data are obtained for each line from 6-bit data of each line (Figure 7-20). Then, (2) an OR is performed on the three 4-bit data thus obtained to yield a mask pattern. (3) The object point data after the transformation are obtained by applying the AND operation between the four object points and the mask pattern.

Patterns for the lines are obtained in the program principally by means of table lookups. Lines 1 and 3 contain points which correspond to x2 and x7 of Figure 7-18 and line 2 contains points which correspond with x4 and x5. The value of these points enter into operation F. If any of these points are 0, the transformed object point x0' should be the same as x0 of the source image.

Figure 7-19
Data Decomposition and Assembly

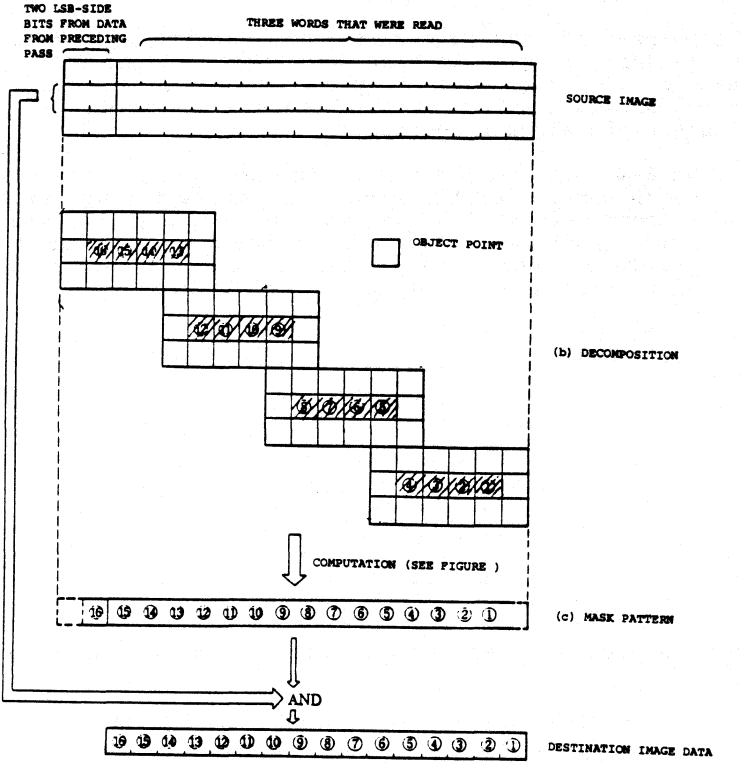
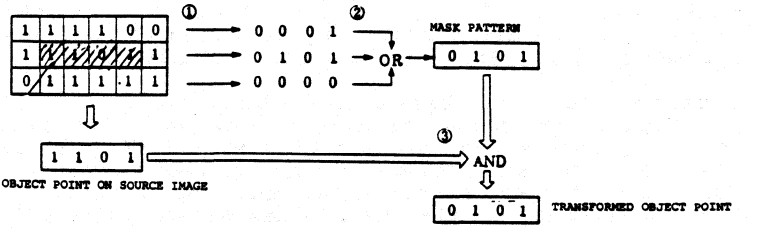



Figure 7-20



7.2.3.3 Flow Graph Explained

Shown in Figure 7-21 is an edge detection flow graph (computation part) which, together with the image memory address generation part flow graph discussed in Section 7.1, makes up the edge detection program.

<Explanation of Main Nodes>

FT30 reverses image data bits for cases where the background is 1 (see Section 7.1.2).

FT31, FT32, FT35 through FT48, FT21, and FT24 determine transformed object points from the source image data. FT31 distributes the data that were read to vertical lines. FT32 and FT35 through FT43 create 6-bit data. FT44 and FT46 perform the table lookup on the basis of the 6-bit data to obtain 4-bit data that form the basis for mask patterns.

Since the creation of transformed object data requires source image object points, FT24 copies the data that are 1-bit right shifted in FT21 for position alignment.

FT45 and FT47 are concerned with mask pattern creation. The patterns created in these nodes are AND-operated with the source image object points generated in FT21, resulting in transformed object points.

In addition to creating 6-bit data, FT32 copies the data so that its lowest two bits can be stored in the DM (the LSB1-LSB3 area) for use in subsequent computations.

FT49 and FT50 reconstitute the transformation object point data obtained in FT48 into one word. This word is organized as shown in Figure 7-19(d).

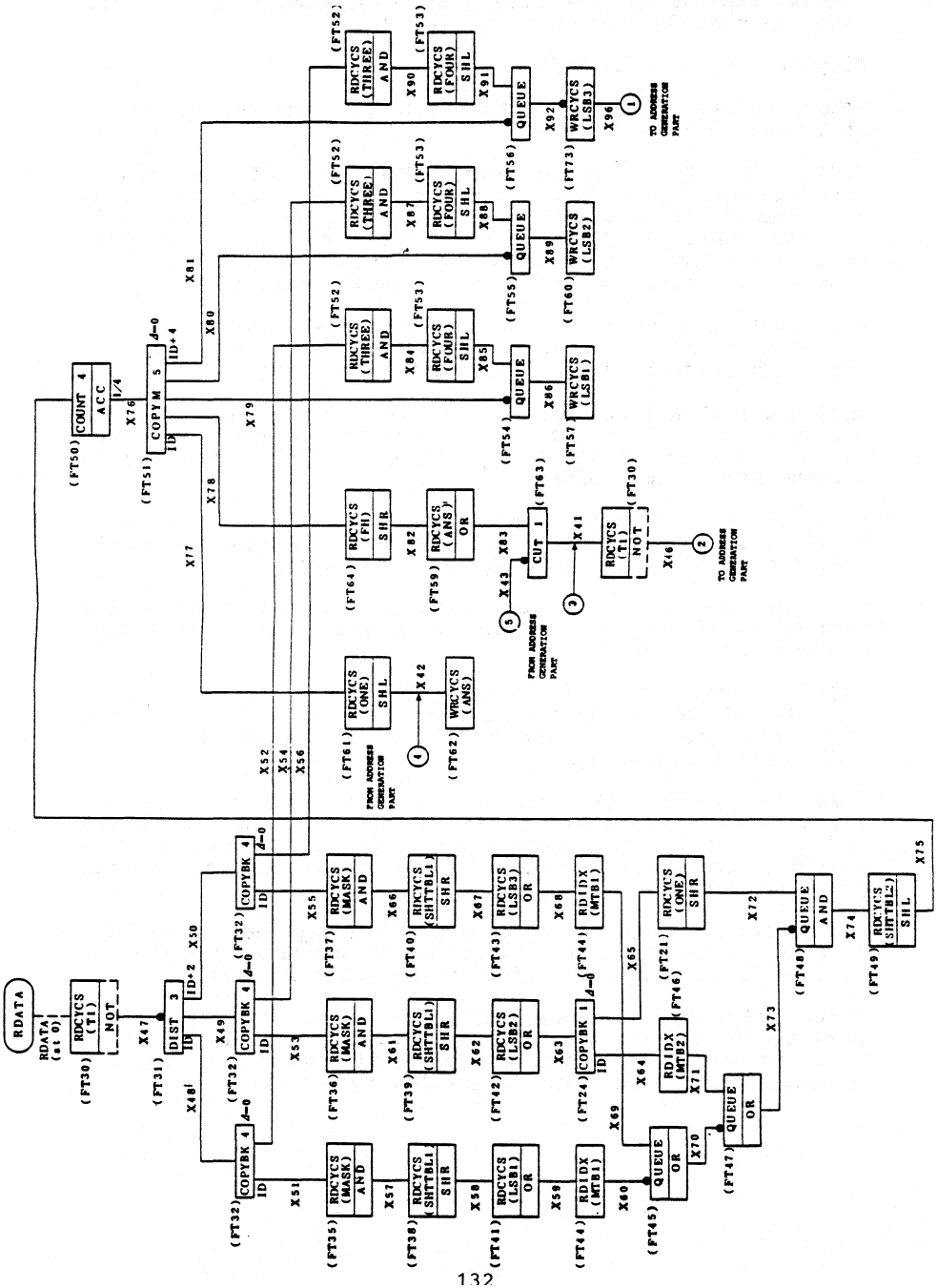
FT61 and FT62 add 0 to the LSBs of the low-order 15 bits, (15)-(1), of the destination image data obtained in FT50 and write the results in the ANS area of the DM for use in subsequent computations.

FT64, FT59, and FT63 extract the MSB (bit 16) from the destination image data to use as LSB data, add the high-order 15 bits that were stored in the DM during the preceding computation, and pass the results to the address generation part.

The procedure employed in handling leading and ending edge data is the same as that used in other programs (see Section 7.2.1.3, "Flow Graph Explained," Smoothing Program).

FT52, FT53, FT54 through FT57, FT60, and FT73 store the lowest two bits of read data, copied in FT32, in the LSB1-LSB3 area of the DM. FT54 through FT56 ensure synchronization with the computation operations.

Figure 7-21
An Edge Detection Flow Graph
(Computation Part)



## 7.2.3.4 Assembler Source Listing

```
 1: ;****************************************
 2: ;
 3: ;       EDGE DETECTION
 4: ;
 5: ;----------------------------------------
 6: ;
 7: MODULE  IPP     =       8               ;
 8: ;
 9: EQUATE  L       =       64              ;
10: EQUATE  H       =       32              ;
11: EQUATE  V       =       512             ;
12: EQUATE  R       =       2               ;
13: ;
14: EQUATE  HOST    =       0               ;
15: EQUATE  READ    =       4               ;
16: EQUATE  WRITE   =       5               ;
17: ;
18: EQUATE  STARTS  =       0FF80H          ;
19: EQUATE  STARTD  =       32              ;
20: ;
21: ;****************************************
22: ;
23: ;       INPUT-OUTPUT
24: ;
25: ;----------------------------------------
26: ;
27: INPUT   LDA0,   LSA0,   RDATA AT 0      ;
28: ;
29: OUTPUT  RDAT,   WDAT,   WADR,   PEND    ;
30: ;
31: ;****************************************
32: ;
33: ;       LINK TABLE
34: ;
35: ;----------------------------------------
36: ;
37: LINK    X1,     X2              =       FT1     (LDA0           )       ;
38: LINK                            =       FT2     (X2             )       ;
39: LINK    X3,     X4,     X5      =       FT3     (X12,   X1      )       ;
40: LINK    X6,     X7,     X8      =       FT4     (X13,   X3      )       ;
41: LINK    X9,     X10,    X11     =       FT5     (X14,   X6      )       ;
42: LINK    X12                     =       FT6     (X4,    X8      )       ;
43: LINK    X13                     =       FT7     (X7,    X11     )       ;
44: LINK    X14                     =       FT8     (X10,   X29     )       ;
45: LINK    X19,    X20,    X21     =       FT10    (X31,   LSA0    )       ;
46: LINK    X22,    X23,    X24     =       FT11    (X32,   X19     )       ;
47: LINK    X25,    X26,    X27     =       FT12    (X33,   X22     )       ;
48: LINK    X28,    X29             =       FT13    (X25            )       ;
49: LINK    X31                     =       FT15    (X20,   X24     )       ;
50: LINK    X32                     =       FT16    (X23,   X27     )       ;
51: LINK    X33                     =       FT17    (X26,   X96     )       ;
52: LINK    X34                     =       FT18    (X21            )       ;
53: LINK    X35                     =       FT59    (X34            )       ;
54: LINK    X36                     =       FT73    (       ,X35    )       ;
55: LINK    X37                     =       FT21    (X36            )       ;
56: LINK    X38                     =       FT58    (X37            )       ;
57: LINK    X39                     =       FT23    (X38            )       ;
58: LINK    X40,    X41             =       FT24    (X39            )       ;
59: LINK    X45     .               =       FT25    (X40,   X5      )       ;
60: LINK    X42,    X43,    X44     =       FT26    (X45            )       ;
```

```
 61: LINK    RDAT                          =      FT27    (X28              )      ;
 62: LINK    WDAT.   WADR                  =      FT28    (X46.   X9        )      ;
 63: LINK    PEND                          =      FT29    (X44              )      ;
 64: ;
 65: LINK    X47                           =      FT30    (RDATA            )      ;
 66: LINK    X48.    X49.    X50           =      FT31    (       .X47      )      ;
 67: LINK    X51.    X52                   =      FT32    (X48              )      ;
 68: LINK    X53.    X54                   =      FT32    (X49              )      ;
 69: LINK    X55.    X56                   =      FT32    (X50              )      ;
 70: LINK    X57                           =      FT35    (X51              )      ;
 71: LINK    X58                           =      FT38    (X57              )      ;
 72: LINK    X59                           =      FT41    (X58              )      ;
 73: LINK    X60                           =      FT44    (X59              )      ;
 74: LINK    X61                           =      FT36    (X53              )      ;
 75: LINK    X62                           =      FT39    (X61              )      ;
 76: LINK    X63                           =      FT42    (X62              )      ;
 77: LINK    X64.    X65                   =      FT24    (X63              )      ;
 78: LINK    X66                           =      FT37    (X55              )      ;
 79: LINK    X67                           =      FT40    (X66              )      ;
 80: LINK    X68                           =      FT43    (X67              )      ;
 81: LINK    X69                           =      FT44    (X68              )      ;
 82: LINK    X70                           =      FT45    (X69.   X60       )      ;
 83: LINK    X71                           =      FT46    (X64              )      ;
 84: LINK    X72                           =      FT21    (X65              )      ;
 85: LINK    X73                           =      FT47    (X71.   X70       )      ;
 86: LINK    X74                           =      FT48    (X72.   X73       )      ;
 87: LINK    X75                           =      FT49    (X74              )      ;
 88: LINK    X76                           =      FT50    (X75              )      ;
 89: LINK    X77. X78. X79. X80. X81       =      FT51    (X76              )      ;
 90: LINK    X82                           =      FT64    (X78              )      ;
 91: LINK    X83                           =      FT59    (X82              )      ;
 92: LINK    X84                           =      FT52    (X52              )      ;
 93: LINK    X85                           =      FT53    (X84              )      ;
 94: LINK    X86                           =      FT54    (X85.   X79       )      ;
 95: LINK    X87                           =      FT52    (X54              )      ;
 96: LINK    X88                           =      FT53    (X87              )      ;
 97: LINK    X89                           =      FT55    (X88.   X80       )      ;
 98: LINK    X90                           =      FT52    (X56              )      ;
 99: LINK    X91                           =      FT53    (X90              )      ;
100: LINK    X92                           =      FT56    (X91.   X81       )      ;
101: LINK    X96                           =      FT73    (       .X92      )      ;
102: LINK    X41                           =      FT63    (X83.   X43       )      ;
103: LINK    X46                           =      FT30    (X41              )      ;
104: LINK                                  =      FT61    (X77              )      ;
105: LINK                                  =      FT62    (X42              )      ;
106: LINK                                  =      FT57    (X86              )      ;
107: LINK                                  =      FT60    (X89              )      ;
108: ;
```

```
109: ;****************************************
110: ;
111: ;       FUNCTION TABLE
112: ;
113: ;----------------------------------------
114: ;
115: FUNCTION    FT1     =   COPYC   (XY      ),     RDCYCS  (T1,    1)      ;
116: FUNCTION    FT2     =   WRCYCS  (T1,    1)                              ;
117: FUNCTION    FT3     =   COPYBK  (1,     0),    CNTGE   (R      )       ;
118: FUNCTION    FT4     =   COPYBK  (1,     L),    CNTGE   (V/R    )       ;
119: FUNCTION    FT5     =   COPYBK  (1,     1),    CNTGE   (H      )       ;
120: FUNCTION    FT6     =   QUEUE   (QUE1,  1)                              ;
121: FUNCTION    FT7     =   QUEUE   (QUE2,  1)                              ;
122: FUNCTION    FT8     =   QUEUE   (QUE3,  1)                              ;
123: FUNCTION    FT10    =   COPYBK  (1,     0),    CNTGE   (R      )       ;
124: FUNCTION    FT11    =   COPYBK  (1,     L),    CNTGE   (V/R    )       ;
125: FUNCTION    FT12    =   COPYBK  (1,     1),    CNTGE   (H      )       ;
126: FUNCTION    FT13    =   COPYBK  (3,     L)                              ;
127: FUNCTION    FT15    =   QUEUE   (QUE5,  1)                              ;
128: FUNCTION    FT16    =   QUEUE   (QUE6,  1)                              ;
129: FUNCTION    FT17    =   QUEUE   (QUE7,  1)                              ;
130: FUNCTION    FT18    =   RDCYCS  (ZERO,  1)                              ;
131: FUNCTION    FT21    =   SHR,                   RDCYCS  (ONE,   1)       ;
132: FUNCTION    FT23    =   OR,                    RDCYCS  (LSB3,  1)       ;
133: FUNCTION    FT24    =   COPYBK  (1,     0)                              ;
134: FUNCTION    FT25    =   QUEUE   (QUE4,  1)                              ;
135: FUNCTION    FT26    =   COPYM   (3,     0)                              ;
136: FUNCTION    FT27    =   OUT1    (READ,  0)                              ;
137: FUNCTION    FT28    =   OUT2    (WRITE, 20H, 0), QUEUE  (QUE8, 16)      ;
138: FUNCTION    FT29    =   OUT1    (HOST,  0)                              ;
139: ;
140: FUNCTION    FT30    =   NOT     (CNOP   ),     RDCYCS  (T1,    1)       ;
141: FUNCTION    FT31    =   DIST    (3      )                               ;
142: FUNCTION    FT32    =   COPYBK  (4,     0)                              ;
143: FUNCTION    FT35    =   AND,                   RDCYCS  (MASK,  4)       ;
144: FUNCTION    FT36    =   AND,                   RDCYCS  (MASK,  4)       ;
145: FUNCTION    FT37    =   AND,                   RDCYCS  (MASK,  4)       ;
146: FUNCTION    FT38    =   SHR,                   RDCYCS  (SHTTBL1,4)      ;
147: FUNCTION    FT39    =   SHR,                   RDCYCS  (SHTTBL1,4)      ;
148: FUNCTION    FT40    =   SHR,                   RDCYCS  (SHTTBL1,4)      ;
149: FUNCTION    FT41    =   OR,                    RDCYCS  (LSB1,  4)       ;
150: FUNCTION    FT42    =   OR,                    RDCYCS  (LSB2,  4)       ;
151: FUNCTION    FT43    =   OR,                    RDCYCS  (LSB3,  4)       ;
152: FUNCTION    FT44    =   RDIDX   (MTB1   )                               ;
153: FUNCTION    FT45    =   OR,                    QUEUE   (QUE9,  4)       ;
154: FUNCTION    FT46    =   RDIDX   (MTB2   )                               ;
155: FUNCTION    FT47    =   OR,                    QUEUE   (QUE10, 4)       ;
156: FUNCTION    FT48    =   AND,                   QUEUE   (QUE11, 4)       ;
157: FUNCTION    FT49    =   SHL,                   RDCYCS  (SHTTBL2,4)      ;
158: FUNCTION    FT50    =   ACC,                   COUNT   (4      )        ;
159: FUNCTION    FT51    =   COPYM   (5,     0)                              ;
160: FUNCTION    FT52    =   AND,                   RDCYCS  (THREE, 1)       ;
161: FUNCTION    FT53    =   SHL,                   RDCYCS  (FOUR,  1)       ;
162: FUNCTION    FT54    =   QUEUE   (QUE12, 1)                              ;
163: FUNCTION    FT55    =   QUEUE   (QUE13, 1)                              ;
164: FUNCTION    FT56    =   QUEUE   (QUE14, 1)                              ;
165: FUNCTION    FT57    =   WRCYCS  (LSB1,  1)                              ;
166: FUNCTION    FT60    =   WRCYCS  (LSB2,  1)                              ;
167: FUNCTION    FT73    =   WRCYCS  (LSB3,  1)                              ;
168: FUNCTION    FT61    =   SHL,                   RDCYCS  (ONE,   1)       ;
169: FUNCTION    FT62    =   WRCYCS  (ANS,   2),    1                        ;
170: FUNCTION    FT64    =   SHR,                   RDCYCS  (FH,    1)       ;
171: FUNCTION    FT58    =   AND,                   RDCYCS  (ONE,   1)       ;
172: FUNCTION    FT59    =   OR,                    RDCYCS  (ANS,   2)       ;
173: FUNCTION    FT63    =   CUT     (1      )                               ;
174: ;
```

```
175: ;**************************************
176: ;
177: ;       DATA MEMORY
178: ;
179: ;--------------------------------------
180: ;
181: MEMORY  T1      =       0                                               ;
182: MEMORY  MASK    =       0F000H, 03F00H, 003F0H, 0003FH                  ;
183: MEMORY  SHTTBL1 =       12.     8.      4.      0                       ;
184: MEMORY  SHTTBL2 =       12.     8.      4.      0                       ;
185: MEMORY  LSB1    =       0.      0.      0.      0                       ;
186: MEMORY  LSB2    =       0.      0.      0.      0                       ;
187: MEMORY  LSB3    =       0.      0.      0.      0                       ;
188: MEMORY  ZERO    =       0                                               ;
189: MEMORY  ONE     =       1                                               ;
190: MEMORY  TWO     =       2                                               ;
191: MEMORY  THREE   =       3                                               ;
192: MEMORY  FOUR    =       4                                               ;
193: MEMORY  FH      =       15                                              ;
194: MEMORY  MTB1    =       0FH, 0FH, 0EH, 0EH, 0DH, 0DH, 0CH, 0CH,
195:                         0BH, 0BH, 0AH, 0AH, 09H, 09H, 08H, 08H,
196:                         07H, 07H, 06H, 06H, 05H, 05H, 04H, 04H,
197:                         03H, 03H, 02H, 02H, 01H, 01H, 00H, 00H,
198:                         0FH, 0FH, 0EH, 0EH, 0DH, 0DH, 0CH, 0CH,
199:                         0BH, 0BH, 0AH, 0AH, 09H, 09H, 08H, 08H,
200:                         07H, 07H, 06H, 06H, 05H, 05H, 04H, 04H,
201:                         03H, 03H, 02H, 02H, 01H, 01H, 00H, 00H;
202: MEMORY  MTB2    =       0FH, 0FH, 0FH, 0FH, 0FH, 0EH, 0FH, 0EH,
203:                         0FH, 0FH, 0DH, 0DH, 0FH, 0EH, 0DH, 0CH,
204:                         0FH, 0FH, 0FH, 0FH, 0BH, 0AH, 0BH, 0AH,
205:                         0FH, 0FH, 0DH, 0DH, 0BH, 0AH, 09H, 08H,
206:                         0FH, 0FH, 0FH, 0FH, 0FH, 0EH, 0FH, 0EH,
207:                         07H, 07H, 05H, 05H, 07H, 06H, 05H, 04H,
208:                         0FH, 0FH, 0FH, 0FH, 0BH, 0AH, 0BH, 0AH,
209:                         07H, 07H, 05H, 05H, 03H, 03H, 01H, 00H  ;
210: ;
211: MEMORY  ANS     =       AREA    (2      )       ;
212: MEMORY  QUE1    =       AREA    (1      )       ;
213: MEMORY  QUE2    =       AREA    (1      )       ;
214: MEMORY  QUE3    =       AREA    (1      )       ;
215: MEMORY  QUE4    =       AREA    (1      )       ;
216: MEMORY  QUE5    =       AREA    (1      )       ;
217: MEMORY  QUE6    =       AREA    (1      )       ;
218: MEMORY  QUE7    =       AREA    (1      )       ;
219: MEMORY  QUE8    =       AREA    (16     )       ;
220: MEMORY  QUE9    =       AREA    (4      )       ;
221: MEMORY  QUE10   =       AREA    (4      )       ;
222: MEMORY  QUE11   =       AREA    (4      )       ;
223: MEMORY  QUE12   =       AREA    (1      )       ;
224: MEMORY  QUE13   =       AREA    (1      )       ;
225: MEMORY  QUE14   =       AREA    (1      )       ;
226: MEMORY  QUE15   =       AREA    (1      )       ;
227: ;
228: ;**************************************
229: ;
230: ;       START
231: ;
232: ;--------------------------------------
233: ;
234: START                                                       ;
235: ;
236: DATA    EXEC    (IPP,   LDA0,   STARTD  )       ;
237: DATA    EXEC    (IPP,   LSA0,   STARTS  )       ;
238: ;
239: END                                                         ;
```

# Appendix A

## Image Memory Read/Write

The system addressed in this document performs image memory access through the uPD9305. The contents of access tokens used are listed in Table A-1. For further details consult the "uPD9305 Users' Manual".

Table A-1
MN Values and Token Types

| Data Category | MN | ID | Function | Abbr |
|---|---|---|---|---|
| (1) | 0 0 0 0 | * * * * * * * | uPD7281 output data to the host computer | CPU |
| (2) | 0 0 0 1 | MN' ID' | Image memory read #1 (RHAR1-selected) | IMR |
| | | 1 1 1 * * * * | RHAR1 assignment (Note 2) | |
| | 0 0 1 0 | MN' ID' | Image memory read #2 (RHAR2-selected) | |
| | | 1 1 1 * * * * | RHAR2 assignment (Note 2) | |
| | 0 0 1 1 | MN' ID' | Image memory read #3 (RHAR3-selected) | |
| | | 1 1 1 * * * * | RHAR3 assignment | |
| | 0 1 0 0 | MN' ID' | Image memory read #4 (RHAR4-selected) | |
| | | 1 1 1 * * * * | RHAR4 assignment (Note 2) | |
| | 0 1 0 1 | 0 0 0 0 0 DIR | Image memory write | IMW |
| | | 0 0 1 * * DIR | High address assignment for write (selection register: DIR=1) | IMWHA |
| | | 0 1 0 * * DIR | Data assignment for write (selection register: DIR=1) | IMWD |
| | | 0 1 1 * * DIR | High address assignment for read (selection register: DIR=1) | IMRHA |
| | | 1 0 0 Mask DIR | Read-modify-write 1 | RMW1 |
| | | 1 0 1 * * DIR | Read-modify-write 2 (Mask is selected with CS bit of image memory write data) | RMW2 |
| (3) | | 1 1 0 * * * * | DMA1 (Host -> uPD7281) | DMA1 |
| | | 1 1 1 * * * * | DMA2 (uPD7281 -> host) | DMA2 |
| (2) | 0 1 1 0 | 0 0 * * * DIR | Self Object Load 1 | SOL1 |
| | | 0 1 * * * DIR | Self Object Load 2 (MN-interchanged) | SOL2 |
| | | 1 * * * * * * | MN assignment for Self Object Load | SOLN |
| (4) | 0 1 1 1 | | Module number for uPD7281 (Valid when RHASEL=1) | PASS |
| | 1 0 0 0 | | Module numbers for uPD7281 | |
| | 1 0 0 1 | | | |
| | 1 0 1 0 | | | |
| | 1 0 1 1 | | | |
| | 1 1 0 0 | | | |
| | 1 1 0 1 | | | |
| | 1 1 1 0 | | | |
| (5) | 1 1 1 1 | | Deletion | VANISH |

Note 1: MN' indicates returned MN (MN' ≠ 111).
ID' indicates returned ID.

Note 2: Becomes an image memory read token when RHASEL of the Mode Register is 1.

**NEC OFFICES**

NEC Electronics (Europe) GmbH, Oberrather Str. 4, D-4000 Düsseldorf 30, W.-Germany, Tel. (0211) 650301, Telex 858996-0

NEC Electronics (Germany) GmbH, Oberrather Str. 4, D-4000 Düsseldorf 30, Tel. (0211) 650302, Telex 858996-0
- Hindenburgstr. 28/29, D-3000 Hannover 1, Tel. (0511) 881013–16, Telex 9230109
- Arabellastr. 17, D-8000 München 81, Tel. (089) 4160020, Telex 522971
- Heilbronner Str. 314, D-7000 Stuttgart 30, Tel. (0711) 890910, Telex 7252220

NEC Electronics (Benelux), Boschdijk 187a, NL-5612 HB Eindhoven, Tel. (040) 445845, Telex 51923

NEC Electronics (Scandinavia) – Box 4039, S-18304 Täby, Tel. (08) 756 7245, Telex 13839

NEC Electronics (France) S.A., Tour Chenonceaux, 204, Rond Point du Pont de Sèvres, F-92516 Boulogne Billancourt, Tel. (01) 609 90 04, Telex 203544

NEC Electronics Italiana s.r.l., Via Cardano 3, I-20124 Milano, Tel. (02) 6709108, Telex 315355

NEC Electronics (UK) Ltd., Block 3 Carfin Industrial Estate, Motherwell ML1 4UL, Scotland, Tel. (0698) 732221, Telex 777565