Formation and transmission of a dynamic graphics display

Eshragh, Nadereh

Formation and Transmission of a Dynamic Graphics Display

by

Nadereh Eshragh, B.Sc.

A thesis submitted in accordance with the regulation for the degree of Master of Science in the Department of Applied Physics and Electronics at the University of Durham.

December 1985

ABSTRACT

The NEC 7220 / GDC is a high resolution colour graphics display controller. It is programmable, and can generate lines, arcs and rectangles at high speeds with little intervention from the host computer. The GDC has interesting capabilities such as scrolling, DMA transfers and read and write of its display memory through the FIFO buffer.

This thesis describes the GDC and its relation to the other components of a graphics terminal. Software programs are developed and implemented to show how the GDC's capabilities can be used to generate a dynamic graphics picture on the CRT screen. The programs are written in both Pascal and 8086 assembler.

Two methods are presented for the transfer of a graphical display from one NEC/APC to another one. The first technique sends the display memory's pixels and the second one transfers the picture codes for the reconstruction of the image. For each of them software programs are developed and tested thoroughly and found to perform as stipulated.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my supervisor Professor C.T. Spracklen for his help, supervision and encouragement throughout both the work and preparation of this thesis. I would also like to thank Dr. J. Wood for the pleasure and privilege of working in the Department of Applied Physics and Electronics, and Dr. C. Smythe for his help and guidance during the preparation of this thesis.

To Mehdi

CONTENTS

Glossary of Terms

ACK	Acknowledgment
APC	Advanced Personal Computer
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAL	Computer Aided Learning
CCHAR	Cursor § Character Characteristics
CPU	Central Processing Unit
CRT	Cathode Ray Tube
CURD	Cursor Address Read
CURS	Cursor Position Specify
CX	Accumulator C
daD	Dot Address
DIR	Direction
DMA	Direct Memory Access
DMAR	DMA Read
DMAW	DMA Write
DS	Data Segment
DVST	Direct View Storage Tube
DX	Accumulator D
Ead	Execute Address
FDL	Figure Drawing Logic
FIFO	First-In-First-Out
FIGD	Figure Draw
FIGS	Figure Specify
GCHRD	Graphics Character Draw
GDC	Graphics Display Controller
GDOS	Graphics Device Operating System
GIOS	Graphics Input/Output System
GKS	Graphics Kernel System
HBPORCH	Horizontal Back Porch
Hex	Hexadecimal
HFPORCH	Horizontal Front Porch
I/O	Input/Output
LAN	Local Area Network
LBA	Line Base Address

LEN	Length
LPRD	Light Pen Address Read
LSB	Least Significant Bit
LSI	Large Scale Integrated Circuit
MOD	Modify
MSB	Most Significant Bit
NAK	Negative Acknowledgment
NDC	Normalized Device Coordinates
PB	Parameter List
PRAM	Parameter RAM
PSAD	Packet starting address
RAM	Random Access Memory
RDAT	Read Data
RMW	Read Modify Write
SAD	Starting Address
VBPORCH	Vertical Back Porch
VDI	Virtual Device Interface
VSYNC	Vertical Sync
WDAT	Write Data

Chapter One
An Introduction to Computer Graphics

1.1 - Introduction

"Graphics" is defined in the Oxford English Dictionary as "of drawing, painting, engraving, etching, etc; vividly descriptive, lifelike; of diagrams and symbolic curves". This definition includes much of what computer graphics can already do and what it will be able to achieve, in the future. The key phrase from that definition is 'vividly descriptive'. For a long time computers have been used to produce diagrams and symbolic curves but they are now capable of painting lifelike pictures, or creating animated films of imaginary landscapes and the creatures to fill them; a good example is the Disney World Film "TRON".

The earliest use of computer graphics was simply to output data from high-speed computers. Many early computers (e.g MIT's Whirlwind computer) had cathode-ray tubes (CRTs) on which data points could be displayed more rapidly and easily than they could be plotted on any other output devices then available. Plottings on hard copy devices such as teletypes and line printers dates from the early days of computing. MIT's 1950 Whirlwind computer had computer driven CRT displays in the control room, while the SAGE Air Defence System in the middle of the 1950s was the first to use command and control CRT display consoles on which operators identified targets by pointing at them with light pens.

Modern concepts in computer graphics began in 1963, with the work of Sutherland [1] on the Sketchpad drawing system. The objective of this work was communication by interaction, which visualized a person sitting in front of a screen and dynamically interacting with the displayed graphics by means of a light pen and symbol menu. Also, around 1963, a historically significant graphics program was started independently at General Motors [2]. DAC/1 (Design Augmented by Computer) evolved into a major computer aided design effort, which has become a key element in the design of GM cars and trucks. This was one of the earliest computer aided design (CAD) implementations using graphics.

The display devices developed in the mid-sixties, and still in use today, were vector refresh CRTs. These were cathode ray tubes in which an electron beam drew a picture on a phosphor coated screen as a collection of straight lines (or vectors). Since the light output of the phosphor decays in a fraction of a second, the picture had to be continuously redrawn at least 30 times per second to avoid flicker. The disadvantage of refresh displays was that they were complicated, expensive and had severe limitations on the number of lines they could display and update because of the large memory requirements involved. Every line that went into a picture had to be stored inside the display so that the picture could be repeatedly drawn. This problem was resolved in the late sixties with the introduction of the direct view storage tube (DVST). In a DVST, the screen is coated with long persistence phosphors such that once an electron beam has traced a line on the screen it remains (until erased) and requires no further refreshing. This was a step forward in making more complex diagrams than were possible with the existing refresh systems. DVST's are still popular today for applications that demand large numbers (tens of thousands) of high-precision lines and characters but do not need dynamic picture manipulation.

The next major hardware advance was to relieve the central computer of the heavy demands of the refresh display device (especially user-interaction handling and picture updating) by attaching the display to a minicomputer. At the same time the hardware of the display processor itself was becoming more sophisticated, taking over many of the routine but time-consuming tasks for the graphics software.

The most significant contribution to the development of computer graphics during the mid-seventies was that of cheap raster graphics based on television technology. In raster displays the display primitives such as lines, characters and solid areas (typically polygon) are stored in a refresh buffer in terms of their component points, called picture elements or pixels. The image is formed from the raster which is a set of horizontal scanning lines each made up of individual pixels. Therefore the raster is simply a matrix of pixels covering the entire screen area. The size of this matrix is known as the resolution of the terminal. A common display resolution is 480 * 640 which provides 307200 individual dots (pixels). The terminal must have sufficient memory to describe at least two values (for a one colour system) for every single pixel.

The development that made raster graphics possible was that of inexpensive solid state memory that could provide refresh buffers considerably larger than those of a decade ago. Low-cost memory and low-cost microprocessors have had a significant impact on the cost trend of computer graphics. Companies like NEC Corporation, Tokyo, have been able to use recent memory technology such as 64k memory integrated circuits to create inexpensive high-resolution and highly interactive raster graphics display systems.

New LSI display processors played an important role in the recent growth of computer graphics. A good example is the NEC 7220 graphics Display Controller (GDC) which is capable of handling a display memory as large as 256k 16-bit words and of drawing lines, arcs, circles, rectangles at a rate of less than 800 nanosecond per pixel [3].

The development of graphics software was not as rapid as that for the corresponding hardware. From the early 1960s hardware manufacturers provided graphics software to drive their own products. This software normally consisted of a set of routines that would allow the user to draw dots, lines and curves - but only specific to that manufacturer's hardware. It was left to the user to write software to accomplish anything more useful.

The most recent development in computer graphics is the arrival of an international graphics language: The Graphics Kernel System (GKS) [4]. GKS is a graphics system which defines a standard interface to all graphics devices. It is provided as a software package which consists of two sections: the device independent section which handles the user required graphics computations and the device dependent section which translates these results into codes that can be understood by the user specific terminal. The standard provides facilities for line-drawing, area fill, segmentation (holding independent or related drawings within the one system), text attributes, colour indexing and many other "intelligent" functions.

The applications of computer graphics from the 60's and well into the present time include computer aided design in the aircraft and textile industries, management information systems, simulations, pattern recognition, graphics art and computer generated movies [2]. Some of the today's application of computer graphics are: Computer Aided Engineering (CAE), Computer Aided Learning (CAL), art and animation, medicine, robotics, video games, picture processing and communications.

The work presented here introduces the NEC 7220 graphics Display Controller (GDC) and shows how dynamic graphics can be provided using its capabilities. The GDC is in the graphics controller board inside NEC's personal computer, the APC. The graphics controller board provides a complete high resolution 3-colour plane graphics video controller. It generates the raster scan display and manages the video display memory. The general purpose CPU in the NEC/APC is the 16-bit 8086 microprocessor.

*

1.2 - Raster Scan Colour Displays

The output device for a raster display is a television monitor that is basically the same as a home television receiver. In raster scan an electron beam continuously traces from left to right and from top to bottom across the display screen, figure (1.1). Changes in the beam intensity determine what actually appears on the screen. The picture is stored in a refresh memory. As a scan line is swept by the monitor, a controller retrieves words from the refresh memory, converts them to an analogue voltage and applies the voltage to the beam intensity amplifier. Each bit of a word in the refresh memory corresponds to a pixel on the display screen. Fig (1.2) shows a picture of a house as stored in the memory and as displayed on the screen.

The front of a black-and-white display tube is coated with a phosphor that glows white when "struck" by electrons. A colour display uses three phosphors that glow, red, green and blue respectively when electrons strike them. This is generally performed by covering the face of the tube with a network of tiny dots or lines of colour. The phosphor dots are excited by using three electron beams in the tube and causing each to strike its respective phosphor colour. The refresh memory is divided into three fields (planes), one for each primary colour. A three colour plane system requires three bits per pixel and provides a total of 8 colour combinations, figure (1.3). Many systems today have up to 24 bits (or planes) per pixel providing more than 16 million colour combinations (e.g. AED 512, Genisco GCT-3000).

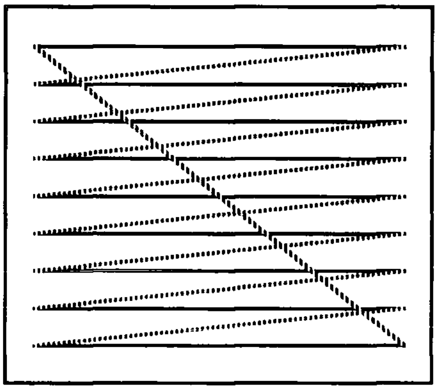


Figure 1.1 Raster Scan

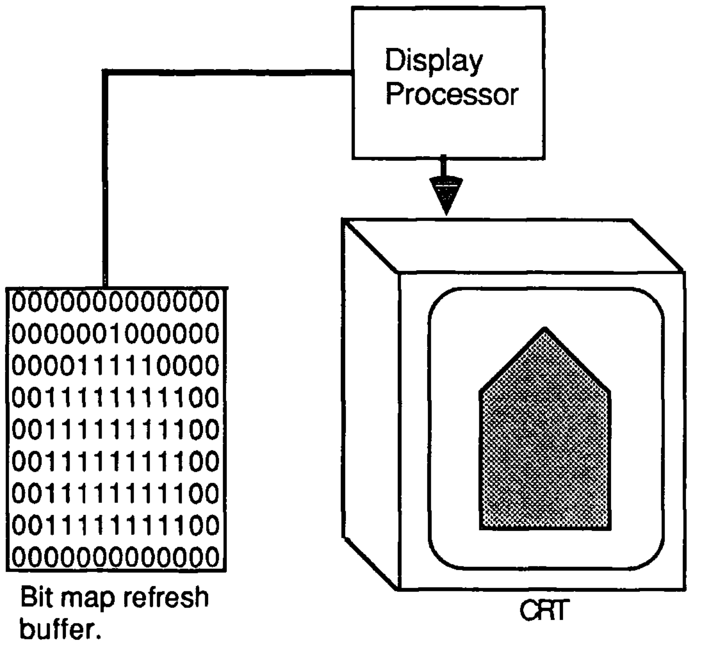


Figure 1.2 Typical raster graphics display showing a house.

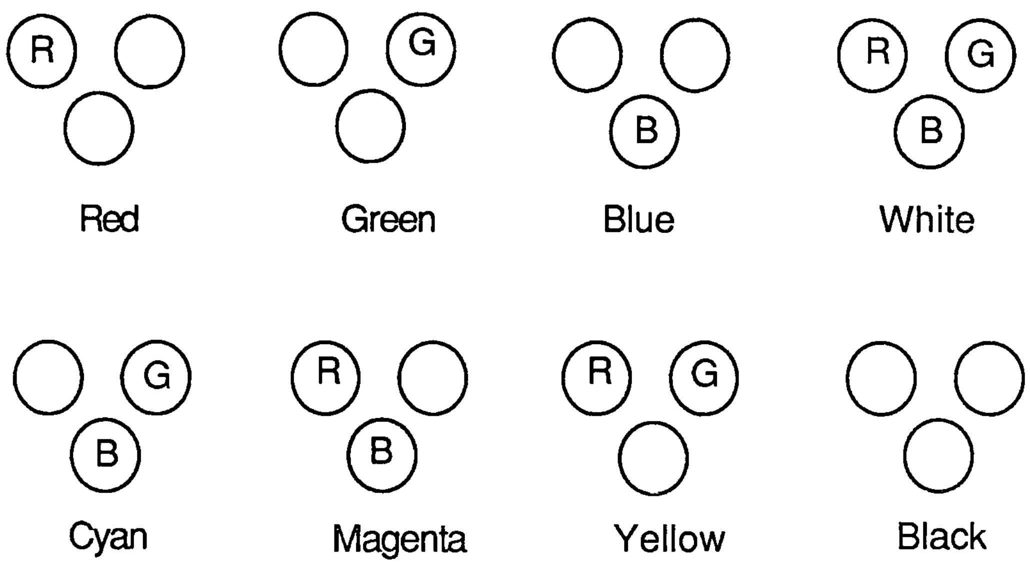


Figure 1.3 Colour Combinations for a three Colour Plane System.

1.3 - The Scope of the Work Described

The work described in this thesis covers the following subjects:

- Programing the Graphics Display Controller (GDC) both directly and using software provided with the NEC Personal Computer (APC).
- Development of additional programs for the GDC to provide functions such as window generation and scrolling.
- Generation of dynamic pictures by manipulating windows.
- Transmission of graphics pictures between two NEC computers.

In addition to these subjects, the text includes detailed descriptions of the NEC graphics device and how it is programmed. This provides significantly improved documentation for future users.

The content of each of the following chapters is now explained.
Chapter two explains the graphics display controller board (GDC) inside the NEC personal computer (APC) and the relation of the board to other units of the computer (eg host CPU and DMA). The host CPU is the 8086 processor. The chapter explains some technical points which are given in the manuals but are not at all clearly described there, together with some points that are not covered. In particular, the documentation describing the GDC chip is difficult to follow and several users have reported problems with understanding some of the points mentioned. There have been revisions of the GDC hardware (mask changes) and many errors, omissions and inaccuracies have been discovered. The aims of this chapter are to detail the author's interpretation of the GDC documentation, to correct the errors discovered and to provide the reader with examples of good GDC programming practice to show how to make the best use of the facilities provided.

Chapter three deals with programming techniques for the GDC. It can be programmed from a high level program using GKS procedures which are provided as a software package (GSX) with the NEC personal computer (APC). This

package only supports the GKS functions for figure drawing. In order to implement other functions such as window generation and scrolling, new software must be developed by directly programming the GDC. Examples of how the GDC can be programmed directly are also given in this chapter.
The GDC has been programmed to demonstrate certain facilities provided with the NEC at hardware level which are not directly supported by the available GSX software. The NEC documents don't describe these hardware facilities clearly and give no examples. The documents don't even describe how to use the available software and give no examples for this either. The first part of the chapter is actually a complementary description to the provided software and examples are given in Appendix B1. The second part of chapter three describes how the GDC can be directly programmed. Examples are given in Appendix B2.

Chapter four explains how scrolling is performed and how the windows are manipulated to generate a dynamic graphics picture. The main aim of this chapter is to show the GDC capabilities (for example scrolling or DMA transfers) with examples of how they can be implemented. A dynamic picture is generated to show how display memory blocks can be accessed and moved around the screen using the GDC's capabilities. Another aim is to show how double buffering can be implemented to create a dynamic picture on the screen.
The information given in this chapter about the GDC's display memory or programming the chip are either not given or not fully described in the NEC manuals.
There are two limitations for creating a dynamic picture as described in this chapter. First, the GDC can transfer a rectangular block of memory which in the case of moving more than one object, if these objects overlap, can create boundary problems. Second, the device operates very fast when moving one

object (the motion has to be slowed down in order for it to be seen on the screen) but the speed will be reduced if more objects are to move at the same time.

Chapter five is concerned with the transfer of graphics pictures between two NEC/APCs. Two methods of transmission are described: pixel transmission and picture code transmission. The original aim of the pixel transmission system was to deliberately create a busy transmission medium to be used as a test environment for a Spread Spectrum local area network [10]. However both methods are valid ways of transfering graphics between computers. A demonstration program is given in appendix B6.

Chapter six contains discussion of the work and conclusions.

Chapter Two
The Graphics Display Controller (GDC)

2.1 - Introduction

This chapter describes the GDC and explains how it can be used to form the basis of a colour graphics terminal. Figure (2.1) shows a typical graphics system with the GDC as the display controller. The host microprocessor passes the drawing instructions to the GDC. The GDC translates these instructions into digital signals and stores the digital picture in the display memory. The video generator circuitry continuously scans the display memory and converts its contents into the TV signals.

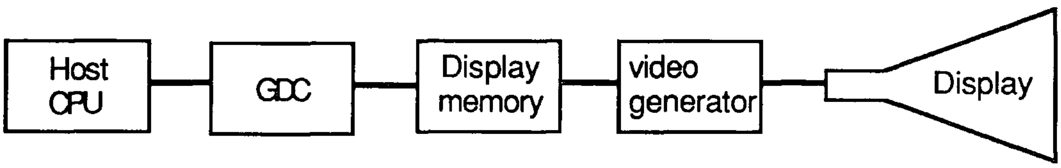


Figure 2.1 Block diagram of a typical graphics system.

The GDC can draw lines, arcs, circles and rectangles at a rate of less than 800 nanoseconds per pixel. It isolates the display memory from the system memory so that the main CPU can calculate the drawing parameters for the next figure or it can communicate with the terminal user, while the current figure is being drawn. The display memory can be as large as 256k words of 16 bit each. For bit-map graphics this can be organized as 2048 pixels by 2048 lines or 1024 by 1024 with 4 bits (colour planes) per pixel. The display memory is often larger than the display area, so as to make possible double buffered display frames or multiple-frame "movies".

In bit-mapped graphics mode each word of the display memory contains 16 horizontal adjacent pixels while in character mode each word contains a character code and its attributes. If each character occupies a 7*10 dot window, a 80 character by 40 row display can be generated with a resolution of 560 by 400.

The display memory may be horizontally split into four character-display or two graphics display areas. Each area can be independently scrolled either in the horizontal or in the vertical directions.

The GDC's two most important properties are its read-modify-write (RMW) cycle and figure drawing capabilities. The GDC handles both data and address information in the display memory. This is used to provide the RMW cycle capability and makes high speed, hardware figure drawing practical.

Fig (2.2) shows a system block diagram of a graphics terminal with the GDC as a graphics subsystem. The GDC provides an interface to the host CPU through data and address buses as well as control lines. The external DMA controller also interfaces to the GDC via a pair of handshake lines. The display memory is driven and controlled by the GDC for both display raster scanning and RMW cycles. The time division multiplexed address and data bus between the GDC and the display memory also passes video data to the video output circuitry. The video circuitry then generates all the necessary video signals for the CRT display unit.

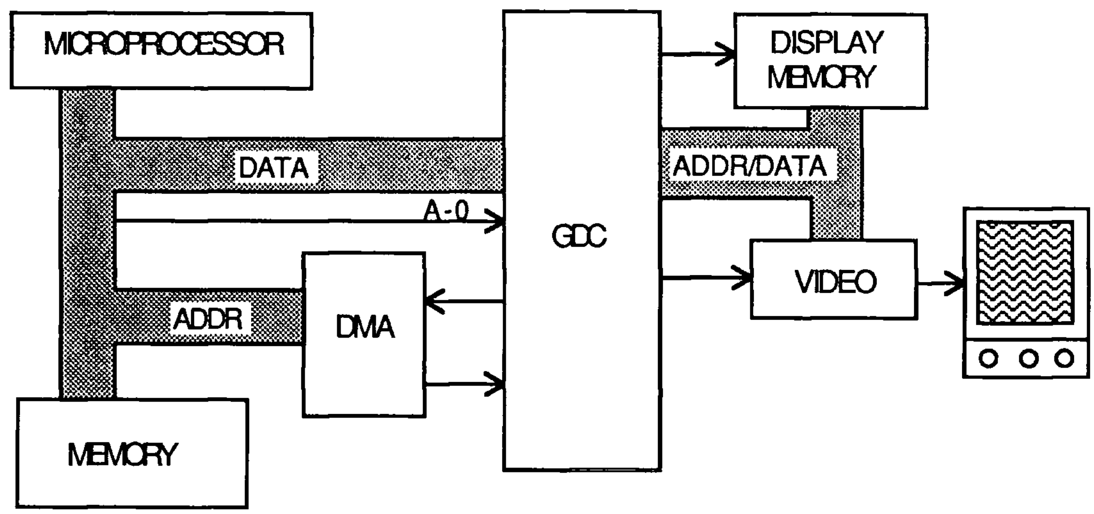


Figure 2.2 Block Diagram of a Graphics Terminal.

2.2 - GDC components

Within the GDC the functions are represented by individual subsystems, Fig (2.3). Starting with the interface between the GDC and the host CPU there is an 8 bit bidirectional data bus and three control lines that are generally driven from the host's address output. The host CPU outputs commands and parameters to the GDC through a First-In-First-Out (FIFO) buffer whose contents are interpreted by the command processor. The command processor decodes the command bytes and distributes the succeeding parameters to their proper destinations within the GDC.

The DMA control circuitry in the GDC is interfaced to an external DMA controller via a pair of handshake lines. The external DMA controller uses the microprocessor interface bus to transfer the display memory data to the system memory and vice versa.

The 16 byte Parameter RAM stores parameters that are used repetitively during the display cycle and the drawing process. In bit-mapped graphics mode it holds two sets of partitioned display areas and the drawing pattern.

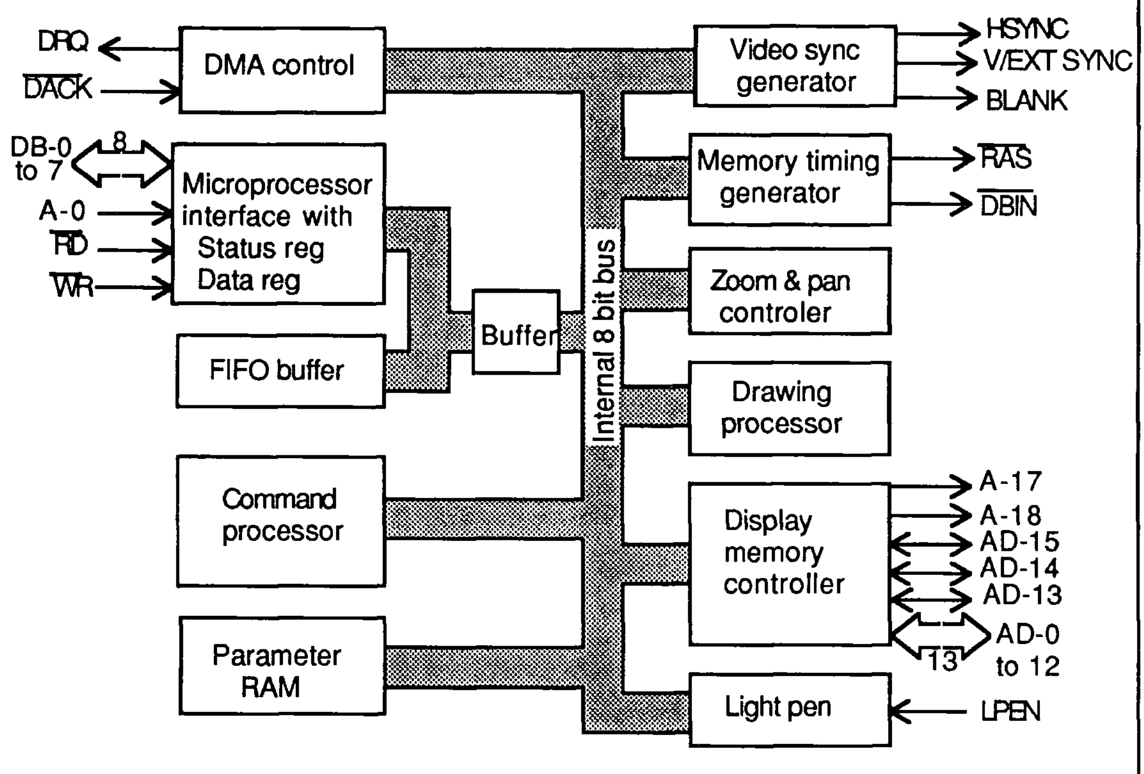


Figure 2.3 GDC Block Diagram.

The drawing processor includes figure drawing hardware to draw lines, arcs, circles, rectangles and 8*8 pixel graphics characters. Given a starting point and the appropriate drawing parameters the drawing controller needs no further assistance to complete the drawing of the figures.

The display memory controller provides a time multiplexed address and data bus to control the memory. It also includes hardware circuitry for RMW operations and figure drawing.

The zoom and pan controller needs additional external circuitry to perform display zoom magnification and smooth horizontal panning. To drive the CRT display unit a video sync timing generator provides the necessary signals for raster-scanning, partitioned display areas, zoomed display, panning and scrolling. A light pen interface capability is also included.

2.3 - GDC command summary

Commands to the GDC can be categorised into five groups. (a) Video Control, (b) Display Control, (c) Drawing Control, (d) Data Read and (e) DMA control. The first two groups allow the video timing and display formats to be specified to the GDC. The figure drawing hardware has its own group of commands. There are commands for reading the display memory, the cursor address and the light pen address. DMA transfers can be initiated with the DMA control commands so that any rectangular area of the display memory can be accessed. The command summary given below further illustrates these commands. Appendix (A1) contains the GDC's data sheet which gives more information about commands and parameter bytes.

Video control commands

RESET : resets the GDC to its idle state and sets the video format.
VSYNC : selects master or slave video synchronization when multiple GDC's are used.
CCHAR : specifies the cursor and character row heights.

Display control commands

START : starts the display scanning process
ZOOM : set zoom factors
CURS : set cursor position
PRAM : load the parameter RAM
PITCH : set the width of display memory

Drawing control commands

WDAT : write data into the display memory
MASK : set the mask value

FIGS : specify the figure to be drawn
FIGD : start figure drawing
GCHRD : start graphics character drawing

Data read commands
RDAT : read data from display memory
CURD : cursor position read
LPRD : read the light pen address

DMA control commands
DMAR : request a DMA read operation
DMAW : request a DMA write operation

2.4 - The Display Memory Architecture

The display memory is organized like a standard computer program memory. From the first location in memory to the last there are no discontinuities or missing memory locations. The GDC scans this linear one-dimensional memory to generate an X,Y two-dimensional display on the CRT without any need for actual two-dimensional addressing (line and pixel number) in the display memory.

As an example imagine a display memory consisting of 20 words and a 3*3 CRT screen. Figure (2.4) shows how the display memory words are displayed on the screen. Pitch (p) shown in figure (2.4) is the width of the display memory which can be different from the CRT display width. As the display memory size is larger than the display area, some words do not appear on the screen.

The display memory in figure (2.4) can be represented as a two dimensional array of words as shown in figure (2.5). The order in which the display memory words are output to the CRT screen locates the origin (0,0) in the upper left hand corner. This is similar to the fourth quadrant of the cartesian plane. Horizontal moves are accomplished with simple increments or decrements while vertical moves require the addition or subtruction of p. Diagonal moves require a combination of both of these operations.

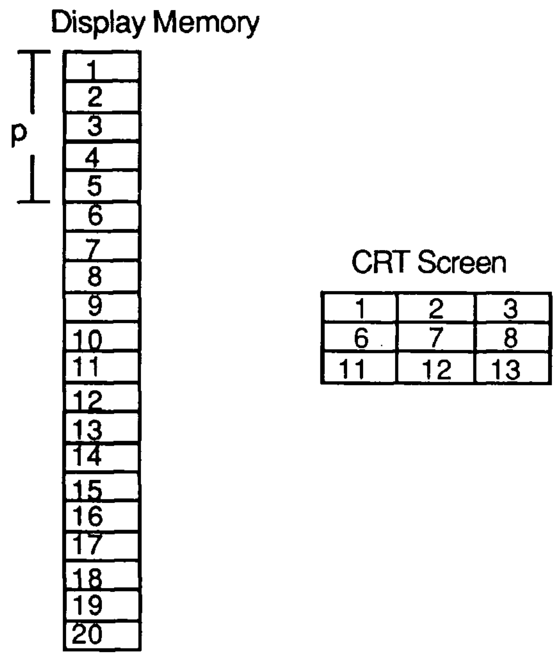


Figure 2.4 The display memory words on the display screen.

p

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Figure 2.5 The display memory as a two dimensional array.

2.4.1 - Display Memory Contents

In the graphics mode, the 16 bits of each word in the display memory are used for 16 horizontally adjacent pixels which are serially output to the CRT. The GDC assumes that the least significant bit, 0, is sent out first. Figure (2.6) shows a word in the display memory and figure (2.7) shows how it appears on the CRT.

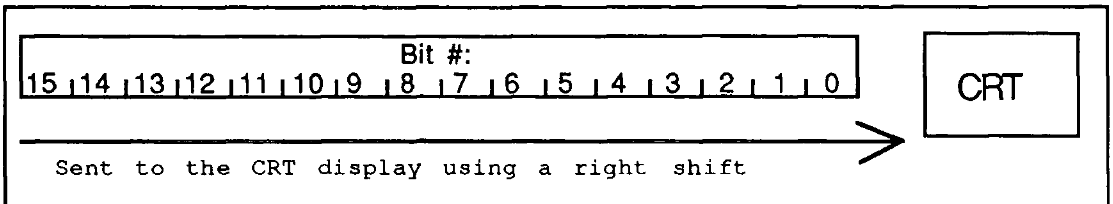


Fig 2.6 word in display memory.

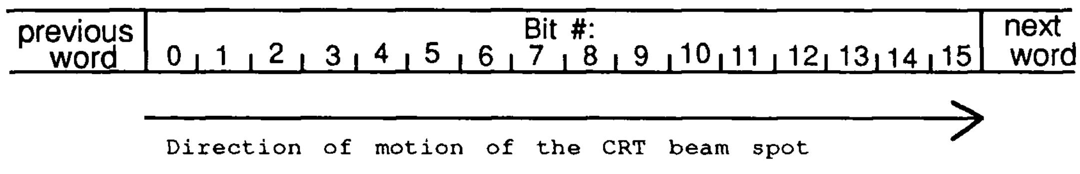


Fig 2.7 Word appearance on CRT.

2.4.2 - Specifying a pixel address in the display memory

There can be up to 2**22 pixels in the display memory, organized into 2**18, (256k) 16-bit words. The address of one of these pixels is specified to the GDC in two parts. First, an 18-bit address selects the display memory word, which contains the pixel. Second a 4-bit value points to the individual pixel within the word. The word address is called the Execute Address, or Ead, and effectively acts like a cursor in the GDC. The pixel address is called the daD or Dot address. The relationship between dots and words is given in a magnified view of the upper left-corner of the CRT screen in figure (2.8). Figure (2.9) shows the arrangment of word addresses on the display memory. p (pitch), the width of the display memory can be defined as: the number of 16-bit words across one line of the display memory.

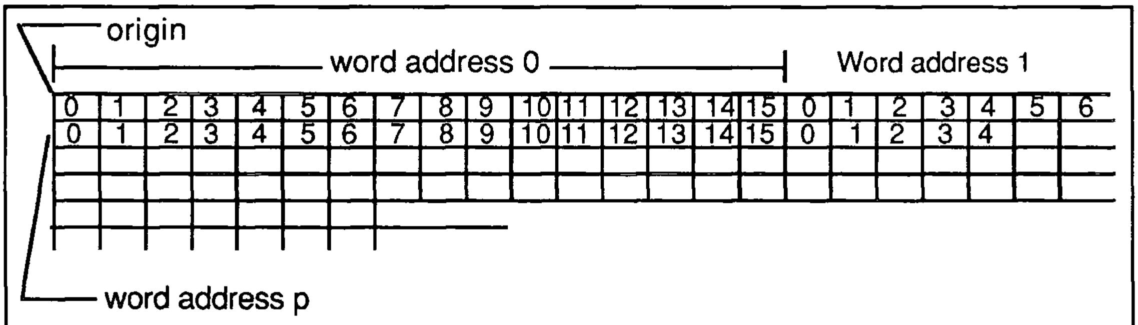


Fig 2.8 Address of dots within their respective words.

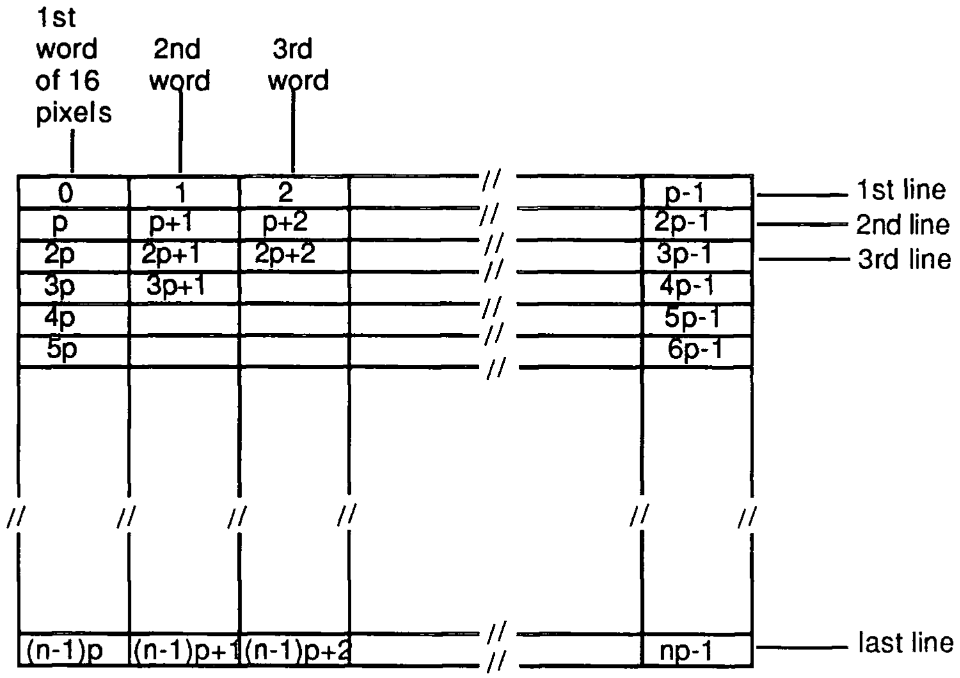


Figure 2.9 word addresses on the display memory.

Given a pixel's x and y coordinates on the display memory its word and dot addresses can be easily calculated . Figure (2.10) shows a pixel with coordinates (X,Y) on the display memory. For the sake of simplicity the origin (0,0) is assumed to be in the upper left-corner which is the same point as the start of the CRT raster-scan. The address of the line in which the pixel lies is called the Line Base Address (LBA):

LBA = p * Y

To find Ead the LBA must be added to the number of words along the line due to the value of the x-coordinate :

Ead= LBA + INTEGER (X/16) = p * Y + INTEGER (X/16)   Equation (2.1).

The above division yields the integer part of the X/16 division, trancating off the remainder. The dot address within the word is the remainder of that division treated as an integer value :

daD = REMAINDER (X/16) * 16 = RESIDUE (X/16)

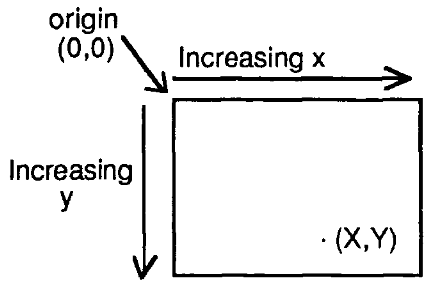


Figure 2.10 a pixel shown on the screen.

An example using actual values is given below.

Example : Let the display memory be configured as 1024 pixels by 1024 lines (Xmax = Ymax = 1024) and the point be (x,y) = (357, 438) :

p = (Xmax + 1)/16 = 1024/16 = 64 words/line

Ead = p * y + INTEGER (x/16)

Ead = 64 * 438 + INTEGER (357/16) = 6D97 Hex

daD = RESIDUE (x/16)

daD = RESIDUE (357/16) = 7

These two numbers, converted to base 2, can be sent to the GDC to specify the particular pixel of interest.

2.4.3 - Multiplane systems

For colour graphics systems the most common way of implementing colour is by building multiple planes of the display memory, each plane representing one of the main primary colours. By sending the video data from all the planes together, a large number of colours may be generated. For example, a three colour plane system provides a total of eight colour combinations, figure (1.3).

The NEC/APC's display memory is organized as three 1024 by 1024 colour planes (red, green and blue), figure (2.11). The Ead word address two most significant bits (bits 16 and 17) select one of the display memory's colour planes.The address of the word within that plane is specified by the Ead's bits 0 through 15 as calculated in the previous section.

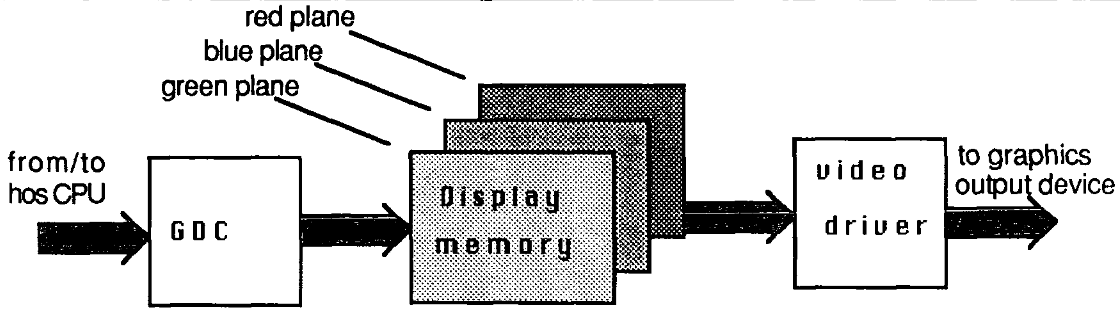


Figure 2.11 A three colour plane system.

2.5 - Read-Modify-Write

Data transfers between the GDC and the display memory are accomplished using a RMW memory cycle, fig (2.12). The four clock period timing of the RMW cycle is used to :

1 - output the address
2 - read data from the display memory
3 - modify the data
4 - write the modified data back into the initially selected memory address.

Figure drawing, DMA transfers, write and read data operations all use RMW cycles.

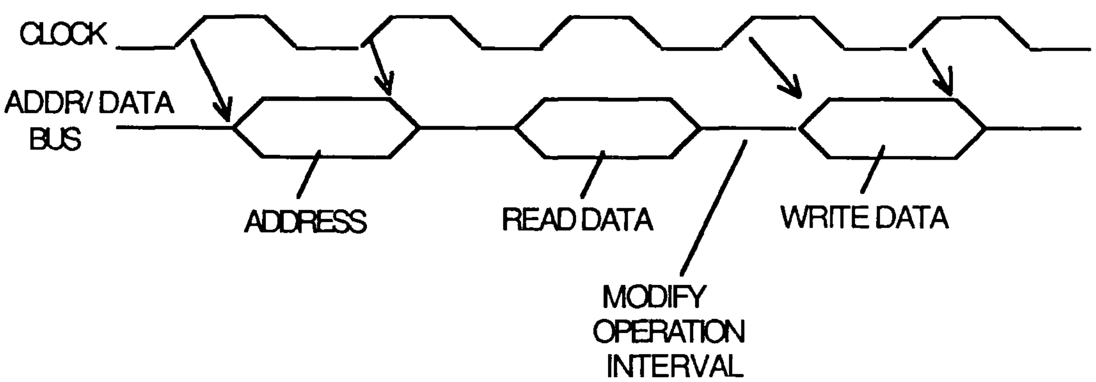


Figure 2.12 Read Modify Write Cycle.

During a figure drawing process, the GDC must modify a number of bytes in the display memory. In this case each RMW cycle modifies only one pixel. Since the GDC can access only 16-bit words, each RMW cycle consists of :

1 - Read 16-bits of data pointed to by Ead

2 - Modify the bit pointed to by daD

3 - Write 16 bits back into the display memory.

During the other operations (e.g. DMA transfers, Write data into the dispay memory) any or all of the bits of a 16-bit display memory word can be modified during one RMW cycle.

2.5.1 - RMW Hardware

Figure (2.13) shows the block diagram of the RMW hardware. The Figure Drawing Logic (FDL) is responsible for calculating the addresses of the RMW cycle, figure (2.14). Before drawing is started the Ead register is loaded with the address of the first word in the display memory using the Cursor Specify Command (CURS). The mask register is loaded with the address of the first pixel to be modified within that word. This address comes from the daD field of the CURS command and is decoded into a 16-bit value with only one bit set.

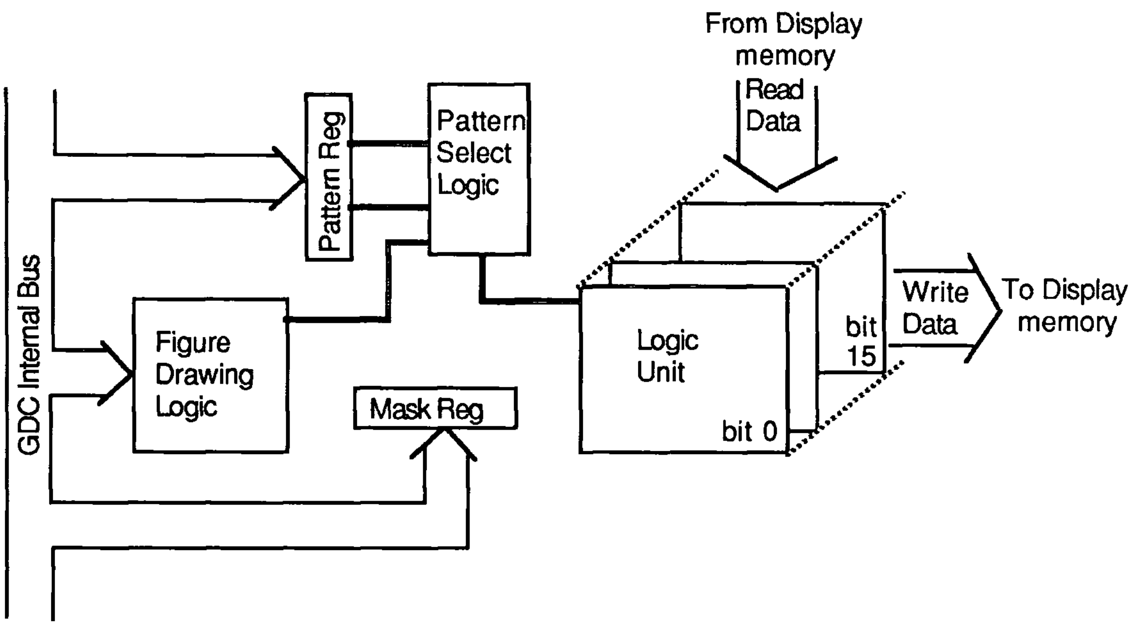


Figure 2.13 RMW hardware block diagram.

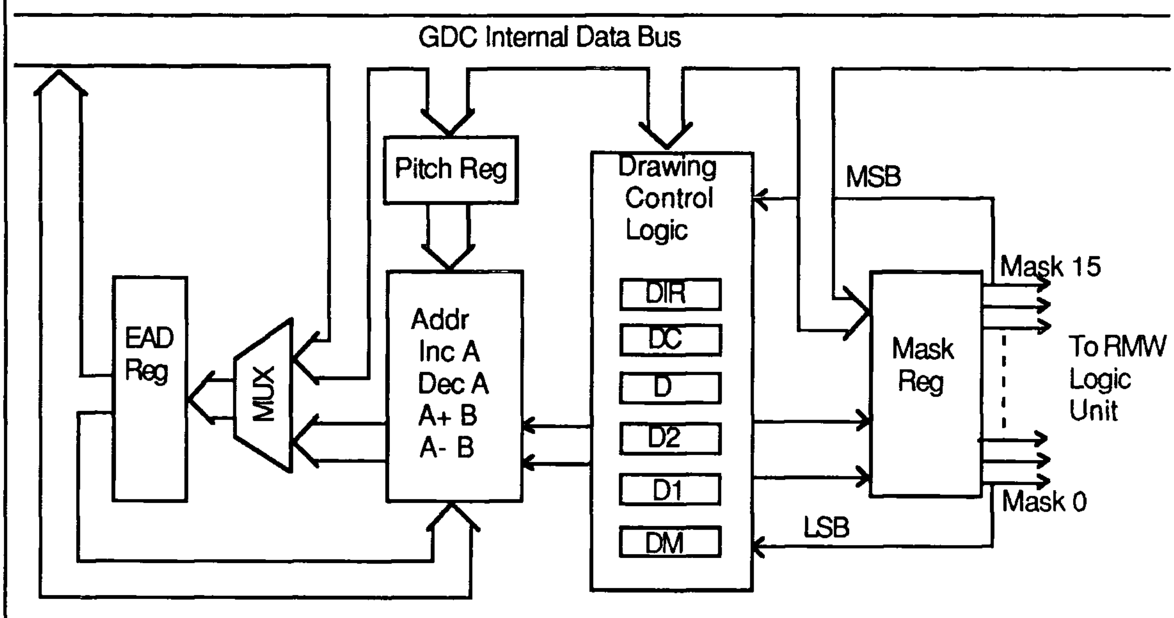


Figure 2.14 GDC Figure Drawing Logic.

The six registers DIR, DC, D, D2, D1, DM (fig 2.14) are loaded with the appropriate values to specify the details of the figure, using the Figure Specify Command (FIGS). After the Figure Draw Command (FIGD) is given to the GDC, the first address is output to the display memory. The content of that address goes into the GDC, is modified by the logic unit and is written back into the display memory.

During this RMW cycle, the FDL calculates the next address. It manipulates the values in the D, D1, D2 and DM registers, looks at the DIR register and examines the MSB and the LSB bits of the MASK register. This generates the next address. At this point, the FDL may increment or decrement the Ead register by adding or subtracting from the Ead the value stored in the PITCH register. It may also shift the contents of the MASK register right or left. The position of the next pixel to be written determines these decisions. The results are new values in some of the drawing control logic registers and a new value in the MASK register.

For operations such as DMA transfers and graphics character drawings the MASK register is loaded with the MASK command. In this case all 16 bits can be set to any desired value.

The Pattern register and Pattern select logic select the bits on which the logic operation will be performed. During the figure drawing process, the pattern select moves a pointer from the LSB to the MSB of the pattern register; if the bit is one, the logic operation takes place, if a zero the bit is not modified. For figure drawing the pattern register is loaded using the PRAM command.

The Logic Unit does the actual RMW data modification. It offers four logic functions : 1) Replace, 2) Complement, 3) Clear and 4) Set, as shown in figure (2.15). These are selectable via the MOD fields of the WDAt and DMAW commands. Figure (2.16) shows the Logic Unit. The pattern register holds the 16-bit data pattern, the mask register points to the pixel to be modified and the logic operation select determines which logic function is to be performed. Table 2.1 further illustrates function of the Logic Unit.

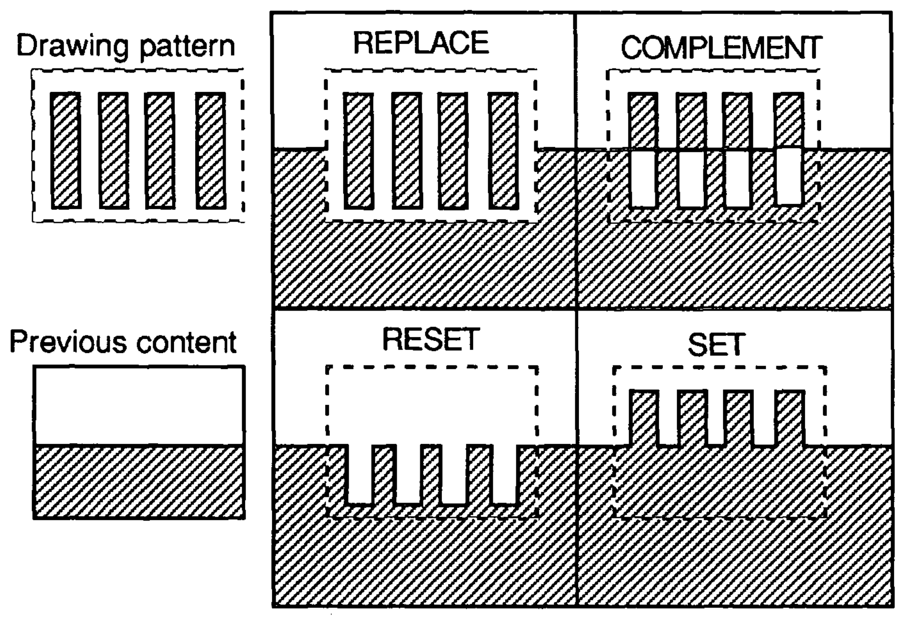


Figure 2.15 RMW operations.

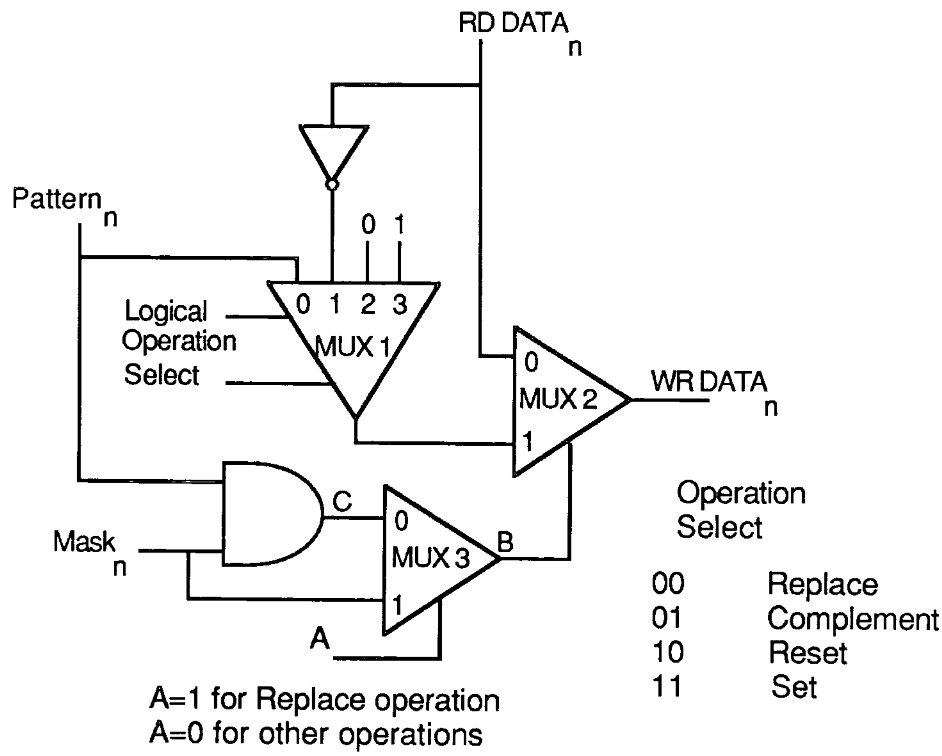


Figure 2.16 Logic Unit Circuitry.

Table 2.1

$Pattern_n$	$Mask_n$	C	A	B	$WR\ DATA_n$	Operation
0	1	0	1	1	$Pattern_n$	Replace
1	1	1	1	1	$Pattern_n$	
0	1	0	0	0	RD DATA	Complement
1	1	1	0	1	$\overline{\text{RD DATA}}$	
0	1	0	0	0	RD DATA	Reset
1	1	1	0	1	0	
0	1	0	0	0	RD DATA	Set
1	1	1	0	1	1	

2.6 - Figure Drawing

The GDC can draw a number of graphics figures into the display memory automatically, under simple commands from the host CPU. These figures include lines, arcs, circles, rectangles and graphics characters. The linear figures can be drawn as solid, dotted and dashed lines, according to the pattern word which is loaded into the GDC by the host CPU. The necessary parameters that describe a figure, together with the figure starting address, must be loaded into the GDC proior to any figure drawing. After this is accomplished the GDC needs no further assistance from the system microprocessor to draw the specified figure.

2.6.1 - Drawing directions

As figure drawing proceeds the next pixel to be modified can be any one of the eight nearest neighbours of the current figure pixel. The GDC assigns each of these 8 directions a number as shown in figure (2.17).

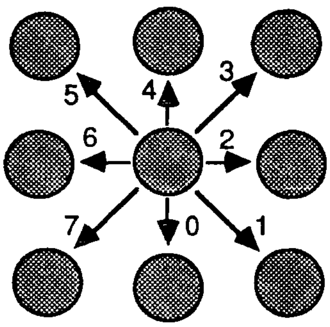


Figure 2.17 Drawing Directions.

To move to the pixel below or above the current pixel, the width of the display memory, pitch, should be added or subtracted from, the word address Ead and the MASK register remains unchanged. In the horizontal direction the MASK register is rotated and its extreme left or right bit is examined. If this is a zero then the next right or left pixel within the current word is modified. If the extreme bit is a one then the word address Ead is incremented or decremented to move to the right or left word. For diagonal directions the add or subtract of the pitch operation must be combined with a MASK register rotation.

2.6.2 - Preparing the GDC for figure drawing

To prepare the GDC for a drawing operation the details of the desired figure must be loaded to the GDC through the FIFO buffer. The type of the RMW cycle can be selected using the WDAT command. The cursor can be positioned to the word and dot addresses, Ead and daD, of the starting pixel of the figure, using the CURS command. For continuous, dotted, dashed, etc figure lines, the PRAM locations 8 and 9 must be loaded with the drawing pattern using the PRAM command. For graphics character drawing and patterned area filling, PRAM locations 8 to 15 can be loaded with the desired pattern or character.

For each figure drawing operation the FIGS command is used to specify the details of the figure; its first parameter byte determines the type (e.g. line, arc etc.) and the direction. The rest of the parameter bytes load the five figure drawing registers, DC, D, D1, D2 and DM to provide the necessary information about the figure.

After the FIGS command and its parameters are sent to the GDC the host CPU must initiate the RMW operation with one of the following commands :

FIGD = Figure Draw Start
GCHRD = Area Filling and graphics Character Drawing Start
WDAT = Write Data into the display memory
DMAW = DMA Write sequence initiate
DMAR = DMA Read sequence initiate.

2.7 - The FIFO Buffer

The main pathway for information flow between the host CPU and the GDC is the First-In-First-out (FIFO) buffer internal to the GDC. Commands and parameters are loaded into the buffer by the host and removed at the other end by the GDC's command processor. The command processor then handles them when it finishes execution of the previous command. The FIFO is also used to buffer data for the host as it is read from the display memory or internal registers.

As is true with all FIFOs, the length of the GDC's FIFO is limited, and if data is output when the FIFO is full, the oldest data in the FIFO will be overwritten and lost. When the host is performing reads from the FIFO, the data is moved from the FIFO into a temporary data register to allow fast access time onto the system data bus.

Three bits relating to the FIFO can be read in the GDC's Status register : FIFO-EMPTY, FIFO-FULL and DATA-READY. The names of these bits describe their ONE state condition. For example, the FIFO-FULL bit is a zero when the FIFO is not full. The two FIFO status bits are meaningful whether the data is flowing from the host into the GDC or the reverse. The DATA-READY bit is used only for data reads out of the GDC. None of these bits are meaningful before the first RESET command opcode is sent to the GDC after power-up.

When commands and parameter bytes are being written into the GDC the FIFO is in Data Write mode. After one of the commands which requests data from the GDC is executed, the FIFO is turned around into Data Read mode. Bytes of data are then read from the Data register, which is in turn filled from the FIFO. The host CPU must check the DATA-READY status bit before each read operation. During the read mode if a command byte is output to the GDC, the FIFO will automatically change direction into the Data Write mode. Any read data in the FIFO at the time of the turn-around will be lost. Turn-arounds of the FIFO to either mode will completely empty the FIFO of any contents.

During outputs to the GDC, the FIFO must not be overflowed. There are two approaches for preventing this. The first is to check the FIFO-FULL status bit for a zero before outputting each command and parameter byte. The second is to wait for the FIFO to become empty and then send 16 bytes or less, in sequence to the GDC.

2.8 - Parameter RAM Contents

The Parameter RAM is used to store two types of information. First it specifies the details of the display area partitions, in blocks of four bytes. The four parameters stored in each block include the starting address in the display memory of each display area and its length. Also there are two mode bits for each area which specify whether that area is a bit-mapped graphics area or a coded character area, and whether a 16-bit or 32-bit wide display cycle is to be used for that area. The other use for the PRAM contents is to supply the pattern for figure drawing when in bit-mapped graphics mode.

In character mode the PRAM can hold up to four sets of display area partitions starting addresses (SAD) and lengths (LEN). In bit-mapped graphics and mixed graphics and character mode, the PRAM locations 0 through 7 hold two sets of display partiton parameters and locations 8-15 supply the pattern for figure drawing. For area filling and graphics character drawing the PRAM locations 8-15 contain the desired character or pattern to be displayed. For line, arc and rectangle drawing (linear figures) locations 8 and 9 are loaded into the pattern register to allow the GDC to draw dotted, dashed, etc lines; for example if an all "1's" pattern is loaded, continuous figures will be drawn.

The parameters stored in the PRAM, are available for the GDC to refer to repeatedly during figure drawing and raster-scanning. In each mode of operation the values in the PRAM are interpreted by the GDC in a predetermined fashion. The host microprocessor must load the appropriate parameters into the proper PRAM locations. The PRAM loading command allows the host to write into any location of the PRAM and transfer as many bytes as desired. In this manner any stored parameter byte or bytes may be changed without influencing the other bytes.

2.9 - Summary

The GDC forms the basis of a colour graphics termianl. It translates the graphics commands into digital signals and stores the digital picture in its display memory which is isolated from the system memory. The GDC's internal structure is shown in figure 2.3. The host CPU outputs commands and parameters to the GDC through the FIFO buffer. The DMA control circuitry in the GDC interfaces to the external DMA controller which uses the microprocessor's interface bus to transfer the display memory data to the system memory and vice versa. In bit-mapped graphics mode the PRAM holds two sets of partitioned display areas and the drawing pattern. The FIFO buffer and PRAM were explained in greater detail, later in the chapter, due to their importance to the rest of the work.

The GDC's display memory is like a standard computer program memory. The GDC scans this linear one-dimensional memory to generate an X,Y two dimensional display on the screen. In the graphics mode the 16 bits of each word in the display memory are used for 16 horizontally adjacent pixels. The address of one of these pixels is specified to the GDC in two parts. First the address of the display memory word which contains the pixel or Ead and second the address of the individual pixel within the word or daD.

Data transfers between the GDC and the display memory are accomplished using a RMW memory cycle. Each RMW cycle uses four clock periods to output the address, read data from the display memory, modify data and write the modified data back into the initially selected memory address. The figure drawing hardware is responsible for calculating the addresses of the RMW cycle.

Chapter Three
Programming the GDC

3.1 - Introduction

The graphics Display Controller (GDC) can be programmed either by using a GSX-86 standard interface or by directly programming the chip. GSX-86 allows the application program to be written in a high level language (e.g. Pascal, Fortran) according to the GKS (Graphics Kernel System) procedures. GSX-86 does not use all of the GDC's capabilities (e.g. DMA transfers, Scrolling) and in order to use these capabilities the GDC must be programmed directly.

The first part of this chapter introduces GSX-86 and describes how it can be driven from an application program. The second part discusses the direct programming techniques for the GDC.

3.2 - Programming the GDC using GSX-86

This section briefly describes GSX-86, the graphics System Extension of the CPM-86 operating system [5] and shows how it can be driven by a Pascal program. An example is given for programming the GDC using GSX-86, but it should be noted that other graphics devices (e.g. plotter, printer) can be programmed by a similar technique. The definitions given in this chapter are as specified in the references [5], [6] and are restated here for the sake of clarity.

GSX-86 defines a standard interface between graphics devices and applications programs. It is an integral part of the CPM-86 operating system and consists of two components :
* Graphics Device Operating System (GDOS).
* Graphics Input /Output System (GIOS).
GDOS provides the interface to the graphics devices and is responsible for loading the desired device driver into the system memory. GDOS also performs coordinate scaling. Applications programs use Normalized Device Coordinates (NDC) which range from 0 to 32767 along each axis . The full scale NDC space is mapped to the full dimensions of the graphics devices in each axis, e.g. the full scale values for the APC are :

X = 640 pixels , Y = 474 pixels

GIOS contains a set of available device drivers that directly program the graphics devices. These can be invoked by GSX-86 through a standard interfacing method which is called the virtual device interface (VDI). To implement the VDI the application program calls GDOS via an interrupt with a function code (interrupt #224 with function code 0473H in register CX as shown in table (3.1)). Registers DS and DX contain the segment base and offset respectively, of the parameter list (PB), to be explained later. Table (3.1) contains a listing of the procedure written in assembly language of the 8086 processor [7] to link an application program to GSX-86 by the VDI method. The application programs written in the Pascal MT+86 language [8] should be linked to this assembly language module. Appendix (B1) contains a demonstration which shows how the GDC can be programmed to draw different markers, using the GSX-86 standard interface. *

The parameter list (PB) consists of five double-word addresses which are the addresses of five integer arrays as follows:

PB	address of input control array
PB + 4	address of input parameter array
PB + 8	address of input point coordinate array
PB + 12	address of output parameter array
PB + 16	address of input point coordinate array

The parameter arrays contain the following values when GDOS is called:

Input Control Array

control(1)	Opcode
control(2)	Number of vertices in input coordinate point array (ptsin)
control(4)	Length of input parameter array
control(6-n)	Opcode-dependent (intin)

Input Parameter Array (intin)

intin	Array of input parameters. Length of array is opcode-dependent and specified in control(4).

Input Point Coordinata Array (ptsin)

ptsin	Array of input coordinates. Each point is specified by an X,Y coordinate pair given in Normalized Device Coordinates (0-32767 with length

control(2) * 2).

Output Control Array

control(3)	Number of vertices in output point array (ptsout)
control(5)	Length of input parameter array
cpntrol(6-n)	Opcode-dependent

Output Parameter Array (intout)

intout	Array of output parameters. Length of array is opcode-dependent.

Output Point Coordinate Array (ptsout)

ptsout	Array of output coordinates.

Table 3.1

```
        public   GSX
        name     gsx-module
        assume   cs : code , ds : data

data    segment  public
data    ends

code    segment  public

gsx     proc     near
        push     ds             ;save registers
        push     es

        mov      ax , ss        ;get segment address
                                ;of the parameter list
        mov      ds , ax
        mov      dx , sp        ;get offset address
                                ;of the parameter list
        add      dx , 6         ;advance the pointer
                                ;past the stored data
        mov      cx , 473H      ;GSX-86 function code
        int      224            ;call GDOS

        pop      es             ;restore registers
        pop      ds
        ret      20             ;return
gsx     endp
code    ends
        end
```

3.3 - Direct programming of the GDC

The host microprocessor can program the GDC for graphics operations by sending the appropriate commands and parameters to it. Commands to the GDC consist of a command byte followed by a set of parameter bytes to specify the details of the command. The command and its parameters are written into the GDC's FIFO. The GDC's command processor decodes the command and loads the parameters into the appropriate internal registers and initiates the required operation. Appendix (A1) contains an explanation of the GDC's command and parameter bytes. Four of the GDC's registers can be accessed by the 8086 processor : Status register, First-In-First-Out (FIFO), Command register and Data register. The input output addresses, functions and bit maps are summarized in Appendix (A2).

3.3.1 - Initializing the GDC

To configure the GDC for the desired mode of operation it must be initialized by a series of commands and parameters. The host microprocessor normally outputs them to the GDC. This command sequence needs to be done only once after power up. The necessary commands and parameters for the GDC initialization are listed below and further details about their definitions can be found in Appendix (A1).

1 - RESET Opcode.

2 - SYNC Opcode.
  P1 = mode bits
  P2 = active words per line
  P3 = VSYNC and HSYNC widths
  P4 = HFPORCH and VSYNC widths
  P5 = HBPORCH width
  P6 = VFPORCH width
  P7 = active lines per video field
  P8 = VBPORCH width and active lines count

3 - VSYNC Opcode + Master / Slave bit

4 - PITCH Opcode
  P1 - Display memory width

5 - PRAM Opcode + PRAM starting address of zero
  P1 = display window starting word address, low byte
  P2 = display window starting address, high byte
  P3 = window length low bits + starting top bits
  P4 = mode bits + window length top bits

6 - CCHAR Opcode
P1 = sweep lines per character row - 1 (P1 = 0 for graphics mode)

7 - ZOOM Opcode
P1 = display + writing zoom magnification factors

8 - START Opcode.

Except for the RESET command which must come first, the order in which the rest of the commands are given is not important, since they do not interact with each other. After outputing the RESET command the GDC will be in the idle mode until a START command is issued. During the idle mode the display will be blanked and the video timing is synchronized. The SYNC parameters may follow the RESET command and the SYNC command need not be given.

Appendix (B2) contains a demonstration program which shows how the GDC is programmed by sending commands and parameters to it. At the beginning of the program the GDC is initialized for graphics mode.

3.3.2 - Programming the GDC to draw markers

Marker drawing is performed by transferring the contents of the PRAM locations 8 through 15 to the display memory, starting at the pixel position specified by the CURS command. The MASK register must be loaded with all ones to insure proper incrementing of the Ead word address and the Area Fill / graphics Chracter mode should be selected via the FIGS command.

The marker type is set by loading the PRAM locations 8-15 with the pattern to be displayed. For example to draw a ' * ', PRAM locations 8-15 are loaded with the following set of data :
92H , 54H , 38H , FEH , 38H , 54H , 92H , 00H
Figure (3.1) shows the PRAM after being loaded with the above set of data.

	PRAM	data
RA-8	1 0 0 1 0 0 1 0	92H
9	0 1 0 1 0 1 0 0	54H
10	0 0 1 1 1 0 0 0	38H
11	1 1 1 1 1 1 1 0	FEH
12	0 0 1 1 1 0 0 0	38H
13	0 1 0 1 0 1 0 0	54H
14	1 0 0 1 0 0 1 0	92H
15	0 0 0 0 0 0 0 0	00H

Figure 3.1 PRAM locations 8 to 15 loaded with the data of a star.

Each marker occupies an area of 8 by 8 pixels when displayed on the screen with a zoom factor of zero. The pattern can be drawn in any of the eight orientations specified by DIR bits of the FIGS command. If direction 2 is selected the FIGS parameters for the marker (graphics character) drawing are as follows :

P1 = type + direction = 00010 + 010 = 00010010

P2 = DC low byte

P3 = DC high byte

P4 = D low byte

P5 = D high byte

where :

DC = (Number of pixels in perpendicular direction) - 1

DC = 8 - 1 = 7

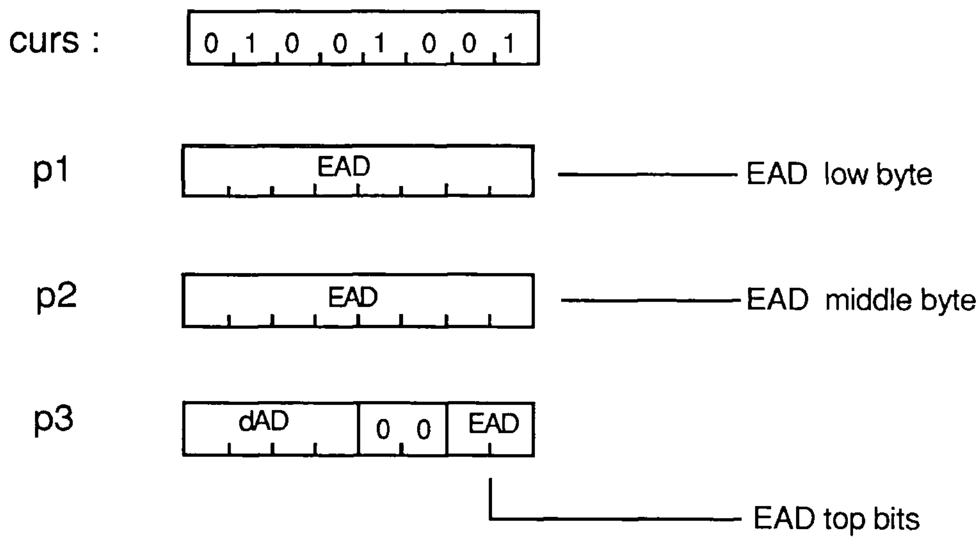
D = Number of pixels in the initial direction.

D = 8

The starting pixel position of the marker is specified by the CURS command. The CURS command also specifies the colour plane address'. The Ead word address two most significant bits (bits 16 and 17) which select one of the three display memory's colour planes, are as follows:

Ead top bits	plane
00	Green
01	Blue
10	Red

These two bits are positioned in the third parameter byte of the CURS command as shown below :



With three main colours, red, green and blue, it is possible to have 8 different colour combinations as follows:

Black	no colour
Red	Red
Green	Green
Blue	Blue
Cyan	Blue + Green
Yellow	Red + Green
Magenta	Red + Blue
White	Red + Green + Blue

For example to draw a yellow marker, it must be drawn once in the red plane and once in the green plane in set or replace writing mode (writing mode is given via the WDAT command). It should also be drawn in the blue plane in reset writing mode to clear any blue shade from previously drawn figures.

The size of the marker is determined by the ZOOM command. Zoom magnification factors of 1 to 16 can be given via the four LSB of the zoom parameter byte.

The necessary commands and parameters for marker drawing are listed below:

1 - ZOOM Opcode

P1 = display + writing zoom magnification factor

2 - MASK Opcode
P1 = FF (Hex)
P2 = FF (Hex)
3 - WDAT Opcode + transfer type + writing mode

4 - CURS Opcode
P1 = word address Ead, low byte
P2 = word address Ead, middle byte
P3 = Dot address daD (bits 0 to 3) + 00 + Ead top bits

5 - PRAM Opcode + PRAM starting address of 8
P1 = Pattern byte 8 (Last drawn)
P2 = Pattern byte 9
P3 = Pattern byte 10
P4 = Pattern byte 11
P5 = Pattern byte 12
P6 = Pattern byte 13
P7 = Pattern byte 14
P8 = Pattern byte 15 (Drawing starts with Bit-0)

6 - FIGS Opcode
P1 = Type (00010) + DIR
P2 = DC low byte
P3 = DC high byte
P4 = D low byte
P5 = D high byte

7 - GCHRD Opcode to initiate the drawing operation.

The above command sequence should be given three times once for each colour plane. Appendix (B2) contains a demonstration program which implements the above techniques to draw markers of different sizes, types and colours. *

3.4 - Summary

Two methods for programming the GDC have been presented: a high level language which implements GKS standard procedures and a low level or direct programming of the chip itself. The former uses the software package provided with the NEC/APC which contains the GSX-86 (the Graphic System Extension of the CPM-86 operating system). GSX-86 allows the application programs to be written in a high level language according to the GKS procedures. The application programs are linked to GSX-86 by a Virtual Device Interface (VDI) method. The interface procedure written in 8086 assembler is listed in table (3.1).

The GDC can be directly programmed by sending the necessary commands and parameters to it; the command series for the GDC initialization and marker drawing were listed. Demonstration programs written in both Pascal and assembly language are given in appendices B1 and B2.

Chapter Four
Dynamic Pictures

4.1 - Introduction

This chapter will demonstrate three different GDC capabilities by which a graphics picture on the screen may be moved. These are scrolling, DMA transfers and read and write through the FIFO buffer.

The first section explains how the screen can be divided into two independently scrollable areas using the PRAM. The second part discusses the DMA capability of the GDC to generate and move multiple graphics windows on the screen. The last section is similar to the second section with the system microprocessor substituting for the external DMA controller. Each section contains a demonstration which consists of a Pascal program and its assembly language module. The assembly language module contains the interface procedure GSX (explained in chapter three), which is called from the Pascal program to draw graphics figures. It also includes new developed procedures for the performance of the above mentioned capabilities.

The coordinates passed to the assembly language module must be in Normalized Device Coordinates (NDC) for GSX and in actual device units for the new procedures. The NDC is in the range from 0 to 32767 along each axis on the CRT screen. The actual device units are:

XM = 640 pixels maximum x on the CRT screen
YM = 474 lines (pixels) maximum y on the CRT screen
The display memory is larger than the display area (CRT screen) and its dimensions are:
XM = 1024 pixels
YM = 1024 lines (pixels)
The origin (0,0) is in the lower left hand corner when NDC is implemented and in the upper left hand corner when working with the actual device units. Figure (4.1) shows the display area dimensions in both NDC space and actual device units.

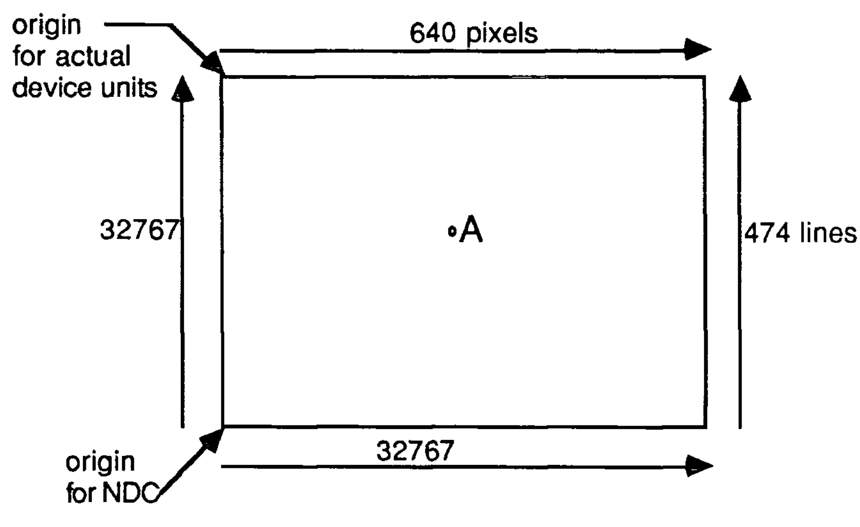


Figure 4.1 The actual device units versus NDC.

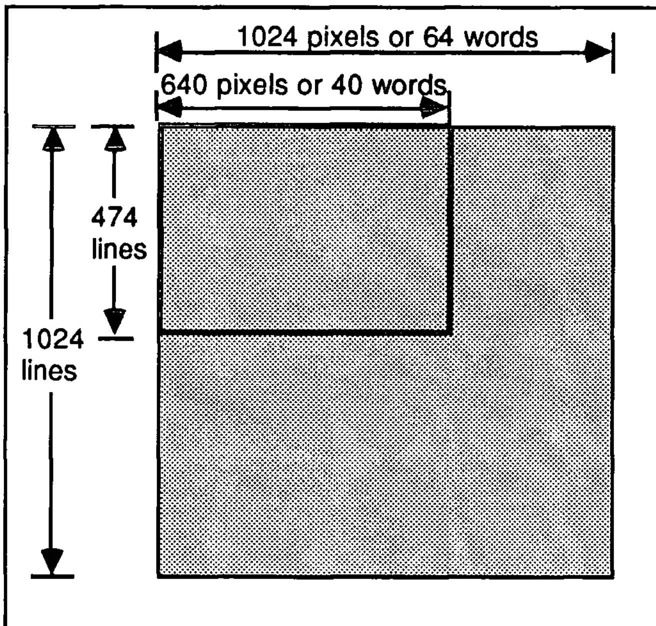


Figure 4.2 The display area and the display memory.

4.2 - Scrolling

The screen can be considered as a window on the display memory because the display memory is often larger than the display area. Figure (4.2) shows a 640 by 474 CRT display as a part of a 1024 by 1024 display memory.

It is possible to move the display window around and see other areas of the display memory through it, figure (4.3). This movement is accomplished by changing the window starting address (SAD), stored in the PRAM. To move the window to the right or left by one word, the SAD should be incremented or decremented respectively. To move the window up or down by one line the SAD should be increased or decreased by the pitch (width of the display memory). Note that when the window is moved, the picture on the screen seems to have moved in the opposite direction. For example if the window is moved by one word to the right, it seems that the picture moves one word to the left.

In bit-mapped graphics mode in the GDC, the display can be divided into two independently scrollable areas by loading PRAM locations 0 through 7 with the display partitions starting addresses and lengths. Figure (4.4) shows the screen memory divided into two equal areas.

Equation (2.1) is implemented to find each area's starting address :

Ead = Y * pitch + INTEGER (X/16) Equation (2.1)

If the starting point has coordinates xs and ys then:

SAD = ys * pitch + INTEGER (xs/16)

The starting address of area one with xs=0 and ys=0 (fig 4.4) can be calculated as follows:

SAD1 = 0 * 64 + INTEGER (0/16) = 0 area one starting address

The starting address of area two with xs=0 and ys=237 (fig 4.4) can also be calculated:

SAD2 = 237 * 64 + INTEGER (0/16)

SAD2= $3B40_{16}$ area two starting address

Both areas have the same length, 237 lines therefore :

LEN1 = 237 = ED_{16}

LEN2 = 237 = ED_{16}

After being loaded by the above set of data, PRAM locations 0-7 are shown below:

location	content	
RA-0	00 H	SAD1L
RA-1	00 H	SAD1H
RA-2	D0 H	LEN1L (4 through 7)
RA-3	0E H	LEN1H (0 through 5)
RA-4	40 H	SAD2 L
RA-5	3B H	SDA2 H

RA-6	D0 H	LEN2 L (4 through 7)
RA-7	0E H	LEN2 H (0 through 5)

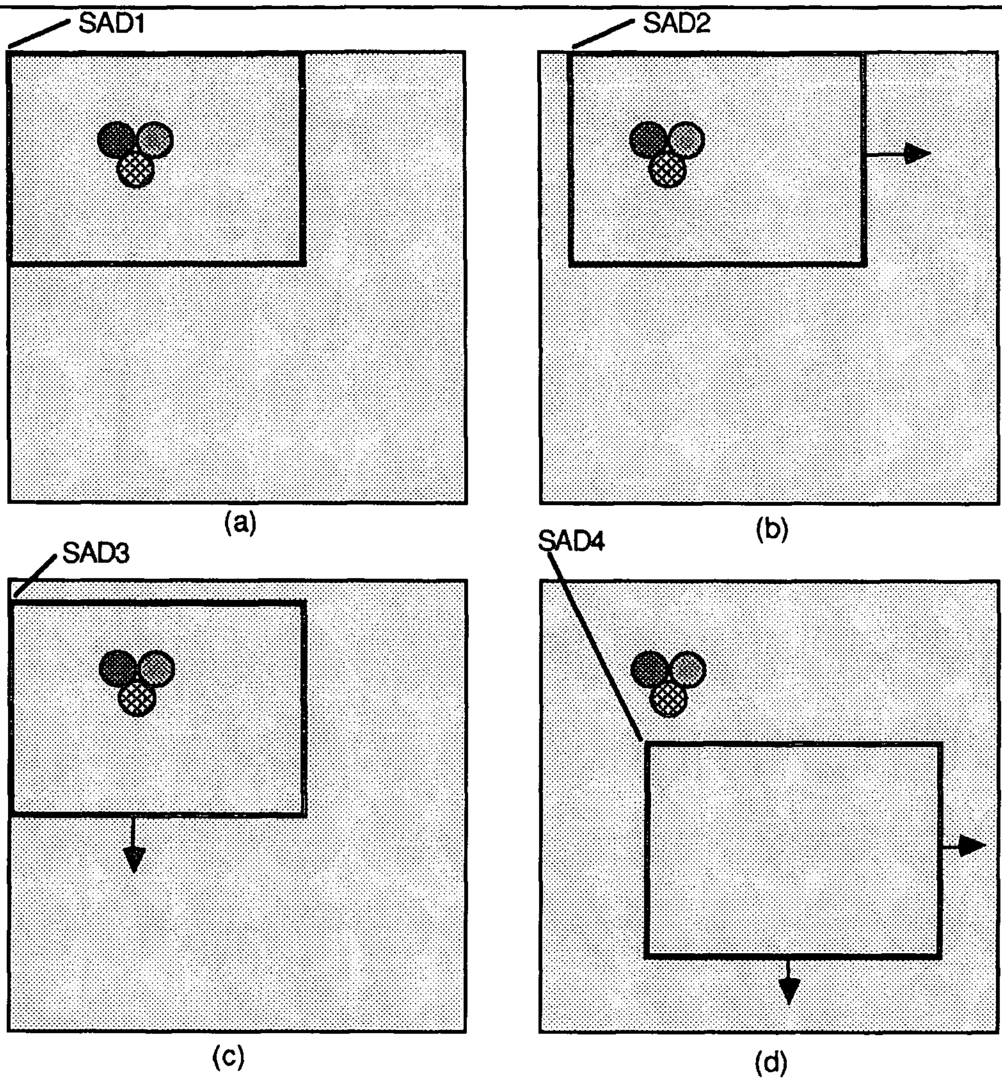


Figure 4.3 The display area in four different situations : (a) in normal position (b) moving to the right direction (c) moving downward and (d) moving in both directions.

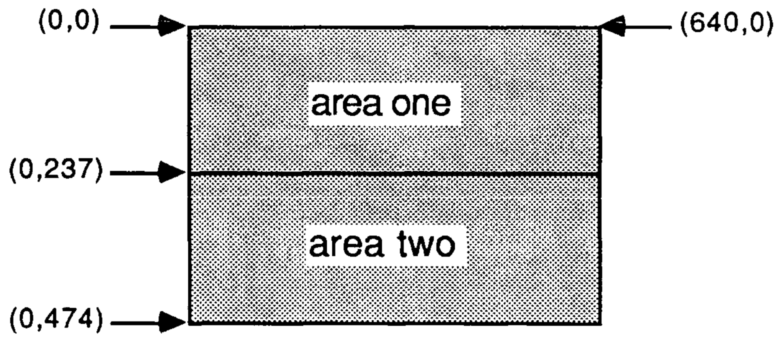


Figure 4.4 The screen memory divided into two equal areas.

To scroll each area, the location of that area's starting address in the PRAM should be given via the PRAM command's SA field. This should be followed by the area's new starting address. For example to move area two one word to the left, if the previous start address is $3B40_{16}$, the following bytes should be sent to the GDC :

1 - 74 (Hex)	PRAM opcode + PRAM starting address of 4
2 - 41 (Hex)	(previous address + 1) Low byte
3 - 3B (Hex)	(previous address + 1) High byte

The contents of the rest of the locations will remain unchanged.

Appendix (B3) cntains a demonstration program which divides the screen into two areas and allows the user to move each area either horizontally or vertically using the four movement keys on the NECAPC keyboard. In the demonstration program the graphical display is drawn by implementing the GSX interface procedure. To divide the screen into two areas procedure "grscrol" is called. To scroll each area the proper PRAM command + PRAM starting address, followed by that area's new starting address are given to the GDC by calling the procedure "scrol".

4.3 - DMA Transfers

The display memory data can be written into the system memory once the exernal DMA controller and the GDC have been set up for the transfer. The GDC and the external DMA controller each provide a memory address in their respective domains. The DMA controller supplies successive addresses in the system memory, while the GDC provides addresses of a two-dimentional block in the system memory. The DMA capability is useful to move data around the display memory. Multiple windows of the display memory can be generated, stored in the system memory and written back into any location of the display memory.

4.3.1 - Preparing for a DMA transfer

The external DMA controller, 8237-5, has four channels from which channel 2 is reserved for graphics operations. Appendix (A3) contains a list of instructions and I/O addresses together with bit maps of registers. The external DMA controller should be programmed for channel 2 and memory to I/O transfers. If data is to be read from the display memory and written into the system memory, the DMA controller should be programmed for a write operation. If data is to be read from the system memory and written into the display memory the DMA controller should be programmed for a read operation. The total number of bytes to be transfered and the address of the first system memory byte to be accessed must be given through the appropriate registers.

To program the GDC for the transfer, the cursor must be pointed to the first display memory word address to be accessed. The mask register should be set to all ones, to make sure of incrementing the Ead word address properly. The FIGS command is implemented to set the TYPE, DIR, DC and D values. For DMA data writing the following command sequence should be given to the GDC:

1 - CURS opcode
 P1 = word address Ead (0 through 7)
 P2 = word address Ead (8 through 15)
 P3 = Dot address daD (0 through 3) + Ead (16 through 17)

2 - MASK opcode
 P1 = FF H
 P2 = FF H

3 - FIGS opcode
 P1 = TYPE (00000) + DIR
 P2 = DC low byte
 P3 = DC high byte

P4 = D low byte

P5 = D high byte

4 - DMAW opcode + transfer type + RMW operation.

The FIGS parameters DC and D are defined as follows :

DC = (Number of word addresses in the direction at right angles to the initially specified DIR, direction) - 1.

D = (Number of bytes to be transfered in the initially specified direction) - 1.

The following command sequence is used for DMA data read :

1 - CURS opcode

P1 = Ead low byte

P2 = Ead high byte

P3 = daD + Ead high bits

2 - MASK opcode

P1 = FF H

P2 = FF H

3 - FIGS opcode

P1 = TYPE (00000) + DIR

P2 = DC low byte

P3 = DC high byte

P4 = D low byte

P5 = D high byte

P6 = D2 low byte

P7 = D2 high byte

4 - DMAR opcode + transfer type + RMW operation.

The FIGS parameters DC, D and D2 are defined as follows :

DC = (Number of word addresses in the direction at right angles to the initially specified DIR, direction) - 1.

D = (Number of bytes to be transfered in the initially specified direction) - 2.

D2 = D/2.

4.3.2 - Dynamic picture generation

The DMA transfer capability of the GDC, makes possible moving pictures across the CRT screen. Complex movements are accomplished by storing a window of the picture in the system memory, replacing the picture on the screen with the background and then inserting the stored window into any desired location on the display in place of the previous one. If two display buffers are available the process of replacement can be made invisible; while the first display is seen through the screen, the necessary changes are performed on the second one. The screen window is then switched from the first to the second display buffer. This process can be continued to generate a continuous motion on the CRT screen.

Example : Appendix (B4) cotains a demonstration program which shows a graphical duck moving across a pond. The implemented technique is described below :

At the beginning, the desired background is drawn on the display area. Using DMA transfers, the display area is then read out of the display memory, stored in the host memory and written back into another area of the display memory. Figure (4.5) shows the display area starting from point (0,0) and its image starting from point (0,474) on the display memory. The display area is called area1 and the image is called area2.

The screen may be considered as a window which can switch to either area one or two. When locations 0 and 1 of the PRAM contain the starting word address of one of the areas, that area is seen through the screen.

The duck is drawn on area1 at point A, figure (4.6). It is desired to move the duck to point B. A window containing the duck is generated and stored in the system memory. This window is called window one. While area1 is seen on the screen, window one is drawn at position B' on area two (B' is the image of B), Figure (4.7). After window one is drawn at B', the display is switched from area one to area two. In this manner the new display shows the duck in a different position and the duck seems to moved from point A to B.

To move the duck further from B' to C', a window from the background behind the duck is generated and stored in the system memory. This is called window2 and has the same size as window one. Now while area2 is seen on the screen, the duck at point A on area1 is overwritten by window2, to replace the duck with the background. Then window1 is written into area1's point C which is the image of C' on area2. After this is accomplished, the display is switched from area2 to area1 and the duck seems to moved from B' (B) to C' (C).

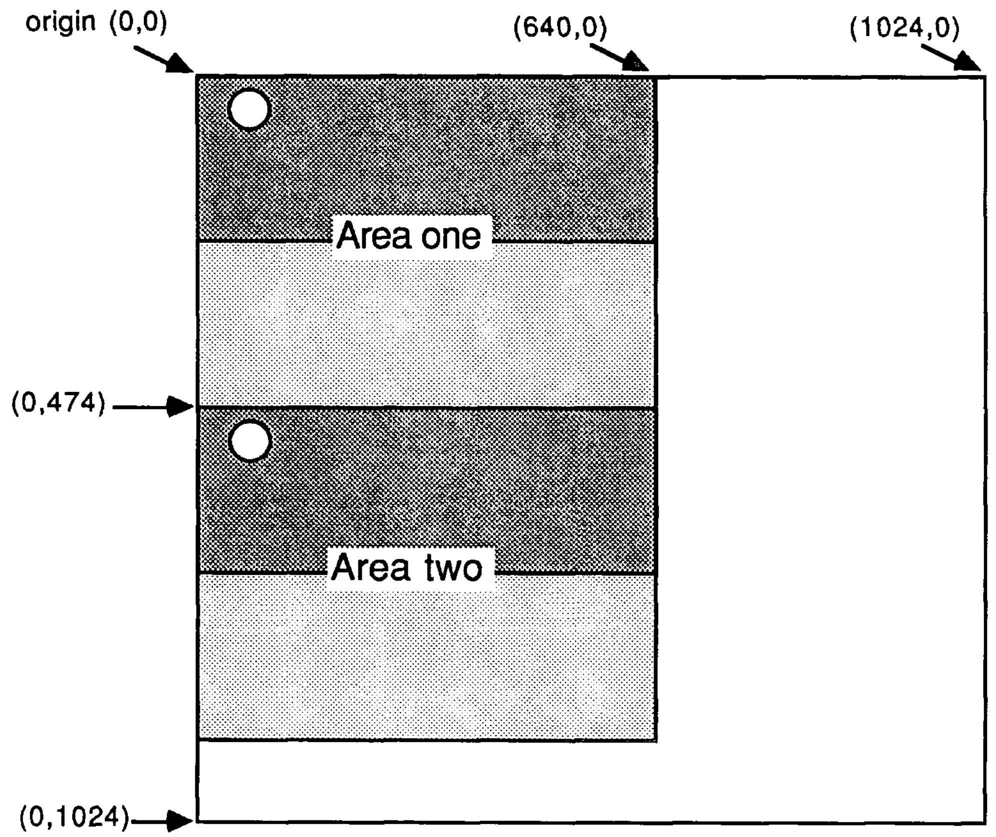


Figure 4.5 The display area and its image on the display memory.

*

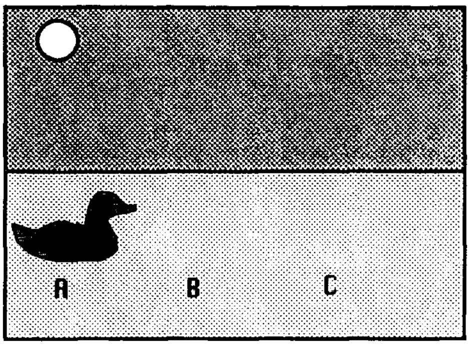


Figure 4.6 Area one.

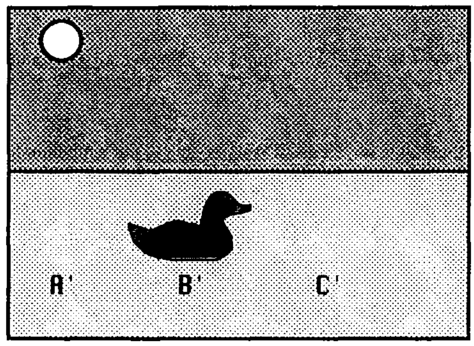


Figure 4.7 Area two.

4.4 - Data Read and Write through the FIFO

The host CPU can access the display memory through the FIFO buffer. Data can be read from the display memory, stored in the system memory and written back into any desired location of the display memory. A dynamic picture may be generated with the techniques explained in the previous chapter.

4.4.1 - Reading the display memory data

For this purpose, the cursor should point to the first word to be read using CURS command. The MASK register should be set to all ones except for directions zero and four, for proper incrementing of the Ead word address. The FIGS parameters require Type, direction and DC values. D1 and D2 parameters are not needed since the GDC reads data through a one dimensional array of words. This is unlike the DMA operation in which the GDC could read a two-dimensional block of data. Given the direction and the DC value, the GDC reads the display memory in that direction until the DC is decremented to zero. The host CPU can program the GDC to read a two-dimensional block by properly advancing the cursor, each time the DC value has reached a zero. After the RDAT command has been issued the GDC loads data into the FIFO. The GDC's data register is in turn loaded with a byte of data and the DATA READY flag in the status register is set. After data has been fetched from the data register by the host CPU, the next data byte is loaded from the FIFO into the data register. The host microprocessor should check the DATA READY status bit before each data read from the data register.

After outputting the RDAT command, the FIFO will turn around from write into read mode. Any command or parameter in the FIFO will be lost at the time of turn around. The process of data read can be aborted by outputting a command byte to the FIFO. The FIFO Empty bit of the status register must not be tested after the termination of the read operation, because the FIFO might still contain some data bytes, and the system will hang. This problem is solved by outputting a dummy command to the GDC after the read operation to turn the FIFO from the read operation to the write mode and flush out any data.

The following command sequence is used to program the GDC to read a block of data from the display memory in direction zero or four:

REPEAT

1 - CURS Opcode
- P1 = Ead high
- P2 = Ead low
- P3 = dAD + plane address

2 - Mask opcode
- P1 = FF (Hex)
- P2 = FF (Hex)

3 - FIGS Opcode
- P1 = Type and direction = 00000 + 100 = 00000100
- P2 = GD Bit + DC low byte
- P3 = DC high byte

REPEAT

RDAT Opcode + Transfer type + RMW operation

UNTIL end of column

increment Ead

UNTIL end of row

This command sequence should be issued three times, one time for each colour plane.

4.4.2 - Writing data into the display memory

In Graphics mode by using the mask register to hold the pattern data, any pattern of bits may be written into a display memory word in one RMW cycle. First the cursor is set with the CURS command, the mask register is then loaded with the data, which can be any arbitrary pattern. The WDAT command should be followed by two dummy bytes with a one in bit zero of the low byte. The reason is that in bit-mapped graphics mode, only the LSB of the first parameter byte following the WDAT command is used to set the pattern register. This allows the pattern register to be loaded with all ones or zeroes. If the pattern register is loaded by all ones, the contents of the mask register is written in each RMW cycle. Since the mask register also controls the Ead address incrementing, an arbitrary mask pattern will not always advance Ead properly. This does not apply to directions four and zero in which the Ead will be incremented after each RMW cycle regardless of the contents of the mask register. If a direction other than four and zero is selected the cursor must be set for each word to be written. For directions four and zero, Ead is incremented linearly in that direction. To write into a block of the display memory, the cursor can be set to the beginning of a column. After writing that column of data, the host CPU should advance the cursor to the beginning of the next column and so on. The following command sequence is used to write data into the display memory in direction 4:

1 - FIGS Opcode
P1 = Type + direction = 00000 + 100 = 00000100

REPEAT

2 - CURS Opcode

P1 = Word address Ead low byte
P2 = Word address Ead high byte
P3 = Dot address daD + plane address

REPEAT

3 - MASK Opcode
P1 = Pattern low byte
P2 = Pattern high byte

4 - WDAT Opcode + Transfer type + RMW operation
P1 = FF (Hex), Dummy pattern low byte
P2 = FF (Hex), Dummy pattern high byte

UNTIL end of column
increment Ead

UNTIL end of row

Both the read and write operations work on a word basis meaning that they start from the first pixel of the word and dAD dot address does not play any role. A block of data can be read from the display memory and written back into another location provided that the direction selected is the same for both read and write operations and data is written in the same order as it was read. A block of data should be read three times, each time in one colour plane and written in the same order as it was read. For example if for the read operation, data is read first in the red plane, second in the green plane and third in the blue plane, it must be written first in the red, second in the green and third in the blue plane.

4.4.3 - Dynamic Picture generation

A dynamic picture may be generated with the same technique as explained in section 4.3.2, with only one difference : A window must be cleared prior to being replaced with the background. The reason is that the mask register holds the data to be written and the bit pattern in the mask register is used only as masking bits, not as new data. To perform the replace operation the word would have to be first cleared in the display memory.

Appendix (B5) contains a demonstration program which implements Read and Write through the FIFO to move a graphical duck across the CRT screen. *

4.5 - Summary

Locations 0 and 1 of the Parameter RAM hold the display area's start address. The display area was moved on the display memory to show the scrolling capability by changing its start address via the PRAM command. The display area was then divided into two equal areas by loading PRAM locations 0-7 with the display partitions starting addresses and lengths and each area was scrolled independently. A demonstration program is listed in appendix B3 to show how the scrolling is performed.

The DMA capability allows access to any rectangular block of data (graphical window) in the display memory. The GDC and the external DMA controller cooperate with each other to read a window from the display memory and store it in the system memory. The stored window can then be read from the system memory and written back into any location of the display memory. In this manner multiple windows of the display memory can be maintained and moved to demonstrate the dynamic picture generation. The access to the display memory can be performed without the external DMA controller by substituting the host CPU in its role. In this situation the host CPU can read data from or write data into the display memory through the FIFO buffer. This was implemented to generate similar dynamic picture as with the DMA. Software programs were developed and thoroughly tested for each of the above mentioned capabilities and found to perform as desired. The programs use GKS procedures to draw the graphical display (to reduce the programming effort) and use the direct programming technique to demonstrate scrolling or dynamic picture generation. Appendices B4 and B5 contain demonstration programs for DMA transfers and read and write through the FIFO buffer respectively.

Chapter Five
Graphics Picture Transmission

5.1 - Introduction

graphics pictures may be transferred by sending either their GKS information or their pixels in the display memory, from one computer to another one. This chapter deals with the transfer of graphics figures between two NECAPCs. Section two introduces the transmission protocol and error detection method which is implemented in this chapter. The third section discusses the transfer of a graphics display by sending the display memory's pixels. Section four explains how a graphics operation is transferred by sending its GSX-86 opcode and parameters.

5.2 - The Transmission Protocol and Error Detection Method

The transmission protocol described in this section is a type of packetized acknowledge based protocol chosen to send large amounts of data more efficiently. This protocol is used by two NECAPCs to communicate with each other. The communication is point to point and bidirectional.

The communication operation requires the transmitter and receiver to:

1 - Establish connection - the transmitter must be sure that the receiver is ready.

2 - Transfer the data reliably - no errors must occur nor must data be lost.

3 - Terminate the transfer so that the receiver knows there is no more data.

To establish the connection, the receiver sends the first ACK message and waits to receive data. If within a period data has not arrived, the receiver sends another ACK and waits again. If after sending a number of ACK messages the computer does not receive data it assumes that the transmitter is not on line. Similarly if the transmitter does not receive ACK within a certain time it assumes that the receiver is not ready. Having received the first ACK, the transmitter knows that the receiver is ready and starts sending data.

The data is sent in packets. The first byte in each packet contains its sequence number, the second byte specifies the length of the packet in bytes, followed by one or more data bytes and the last two bytes are check bytes which can be the sum of all the data bytes in the packet. The packet format is shown in figure (5.1).

sequence number	packet length	data	checksum
one byte	one byte	one or more bytes	two bytes

Figure 5.1 Packet format

The sequence number is the number of the next packet to be transmitted by the sender and the number of the next packet expected by the receiver. The sequence number is initially set to one when the connection is established; this gives a common start value for both ends.

Having received a packet the receiver generates its own local checksum and compares it with the received checksum. If the two checksums are equivalent then the packet is accepted and an ACK message is returned to the data source. The receiver then waits to receive the next packet. If the two checksums are not equivalent the receiver detects that an error has occurred and discards the entire packet. The receiver then sends a negative acknowledgement, NAK, to the data source. A NAK is also sent if either the received packet sequence number is not the same as the expected one, or any of the packet bytes are lost. After sending a NAK the receiver waits to receive the next packet.

Having sent a packet the transmitter waits for the response of the receiver. If the sender receives ACK, it knows that its last packet has been transferred correctly and starts sending the next one. If the transmitter gets a NAK message, it transmits the previous packet again.

To terminate the transfer the transmitter sends a termination message to the receiver. After receiving this message the receiver knows that there are no more packets.

5.3 - Transfer of Display Memory Data

The graphics picture can be transferred by sending the display memory data bytes. The transfer of the entire display area is implemented by the reading and the storing of the display area using the techniques explained in the previous chapter. The display area must be read three times, once for each of the three colour planes, Fig (5.2). Each colour plane consists of 474 lines by 40 words, so the total number of bytes to be transferred for each plane is 37920 bytes. This large amount of data is sent in packets. To determine a packet size, the display area can be divided into forty vertical strips, each with width = one word and length = 474 lines, shown in figure (5.3). If each of these strips represent a packet, then each packet contains 948 bytes. The transmitter adds a byte containing the packet sequence number to the beginning and two checksum bytes to the end of each packet (the checksum is the sum of all data bytes in the packet). As the packet length is fixed it is not necessary to transmit this information.

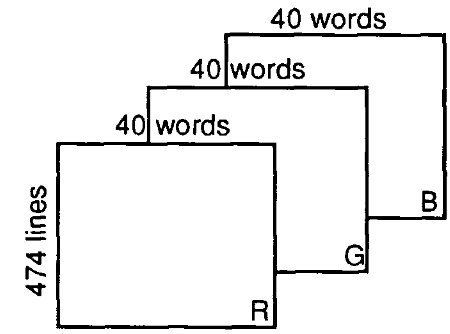


Figure 5.2 The display area.

Having received an error-free packet, the receiver displays the packet data from the same display memory address and in the same direction as was used by the transmitter. The receiver must be programmed to display data in the same sequence of colour planes as they were read in the transmitter.

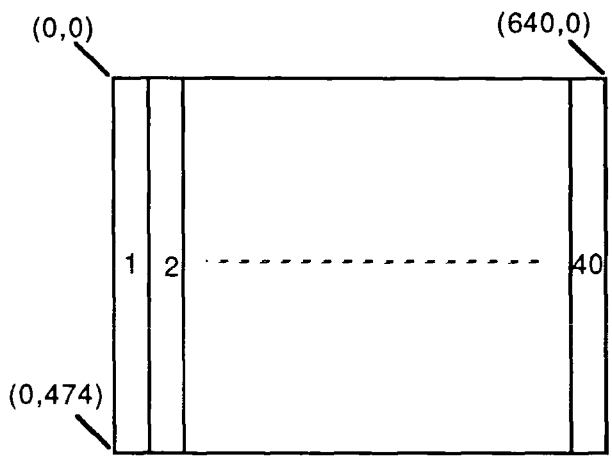


Figure 5.3 Packets on the display area.

The first packet sent by the transmitter starts from point (0,474), the lower left hand corner on the screen. After receiving this packet the computer displays the packet from (0,474) on its screen. After successfully transferring a packet, the packet start address, PSAD, is incremented by one word in both the transmitter and the receiver; in this manner the packet number is synchronized at both ends. After receiving the entire display area in three sets of 40 packets, the receiver knows that the transfer is complete.

It is possible to send any window of the display memory. However the receiver must be informed of the change of the packet size, total number of packets and the start address of each packet within the display memory. The transmitter can be programmed to send a packet containing the necessary information to the receiver before the start of any window data transfer.

In the demonstration programs "PIXELTX" and "PIXELRX" the entire display area is transferred between two NECAPCs using the above techniques. These two programs are listed in Appendix (B6). *

5.4 - Transfer of Picture Codes

A GSX-86 operation is determined by an opcode and a series of parameters which specify the details of that opcode. It is possible to transfer these values between two computers, by implementing the protocol described in section 5.2.

The transmitter sends a packet whose contents include the packet number, length, data, and checksum. The first word of the packet data contains the GSX-86 opcode and the rest are the necessary parameter values. For example if packet number 10 carries the information of the line (0,0) to (1280,1280) on the CRT screen, its contents include:

packet number	0A H
packet length	06 H
opcode	0006 H
number of points	0002 H
first X	0000 H
first Y	0000 H
second X	0500 H
second Y	0500 H

checksum 0012 H sum of all data bytes

Having received an error-free packet, the receiver inspects the first word of the packet data (which contains the opcode) and loads the rest of the data into the proper GSX variables relevant to the opcode and initiates the appropriate graphics operation. The transfer of non-GSX operations (e.g. Pan, Scroll) is possible if a separate opcode is provided for each of them.

The receiver and transmitter algorithms must be complementary. For example the transmitter must not send the circle opcode unless the receiver has the necessary procedures for circle drawing. If a new procedure is added to the transmitter, a similar one must also be added to the receiver.

Appendix (B7) contains the transmission and the reception programs which demonstrate the transfer of graphics commands between two NECAPCs. *

5.5 - Summary

A packetized acknowledge type transmission protocol has been introduced which utilizes a simple checksum for error correction. This protocol was employed for the transfer of the entire display area between two APCs. This was achieved by transmitting the display memory's pixels and required the transfer of 37920 bytes per each of the three colour planes (when the APCs resolution of 474 by 640 pixels is considered). This is a slow process and consequently the build up of the picture can be seen on the screen. Appendix B6 contains the Software programs which were developed to demonstrate pixel transfer between two NEC/APCs.

As an alternative the graphical information was transfered by transmitting the relevant GKS codes. In this manner each graphics operation required the transfer of only a few bytes of information, thereby considerably reducing the transmission time. Demonstration programs were developed to show the transfer of GKS codes between two NEC/APCs and are listed in appendix B7.

Chapter Six
Conclusions

The aim of this work was to drive the Graphic Display Controller (GDC) inside the NEC's Advanced Personal Computer (the APC) and to take advantage of its capabilities to transfer a dynamic graphic picture between two APCs.

The work was initially concerned with a study of the GDC and its display memory. The GDC's internal structure and its relation to the other parts of the graphics terminal were discussed; the display memory was explained in detail due to its importance to the rest of the work. The Read Modify Write (RMW) and figure drawing capabilities were also explained to illustrate how the GDC handles data and addresses within the display memory.

Two methods were presented for GDC programming: a high level language which implements Graphics Kernel System (GKS) standard procedures and a low level or direct programming of the chip itself. The GKS software package provided with the NEC/APC did not use all of the GDC's capabilities, such as scrolling, DMA transfers and data read/write through the FIFO buffer and so the direct programming method was specifically implemented to show these capabilities. Demonstration programs which were written in both Pascal and assembly language (for the 8086 host CPU) were developed and implemented for the two methods.

The technique by which the display area can be moved on the display memory was discussed to show the scrolling capability. The display area was also divided into two equal areas with each area being scrolled independently. The scrolling capability was used later in dynamic picture generation in which the display was double buffered, to provide switching between the two displays.

Access to the display memory by either the system microprocessor or the external DMA controller, enabled the generation and movement of multiple graphical windows. Software programs were developed to move an object against a constant background on the CRT screen. The developed software uses GKS procedures to draw the graphical display (to reduce the programing effort) and the direct programming technique to generate and move the graphical windows. The programs were implemented and thoroughly tested on the NEC/APC.

The entire display area was transferred between two APCs by transmitting the display memory pixels. This required the transfer of 37920 bytes per each of the three colour planes (when the APCs resolution of 474 by 640 pixels is considered), which gives a total of 113760 bytes or the equivalent of 910080 pixels. As expected this is a slow process and consequently the build up of the picture on the screen can be seen. Calculation of the theoretical transmission time is determined by dividing the total number of bits on the screen by the speed of the communication line (baud rate). The 4800 baud rate was selected experimentally for the best performance of the system. The transmission time is calculated as follows:

910080 pixels / 4800 bits per second
= 189.6 seconds or 3.16 minutes

The actual recorded time is longer than the calculated one. This is due to software overheads and the occurrence of errors coupled with the time required to correct them. Software programs were developed to demonstrate pixel transfer between two NEC/APCs. The implemented transmission protocol is a packetized acknowledge type which utilizes a simple checksum for the error correction.

The graphical information can be transferred by transmitting the relevant GKS codes. In this manner each graphics operation requires the transfer of only a few bytes thus reducing the transmission time considerably. This method is preferred to pixel transfer if high speed transmission is required. Demonstration programs show the transfer of GKS codes between two NEC/APCs. The transmission protocol is the same as for the pixel transfer method but each packet contains the GKS information relevant for the reconstruction of the image, instead of carrying pixels.

A dynamic picture may be transferred by implementing either of the transmission techniques explained above. If the pixel transfer method is implemented, the entire display is transmitted each time a movement occurs on the screen. The movement is slowed in both the transmitter and the receiver due to the transmission time required to transfer the display area. Transfer of picture codes can solve the speed problem. Since the provided GKS package can not generate dynamic pictures, the necessary software for DMA transfers, scrolling etc must be developed with the corresponding introduction of new codes. In this manner both the GKS codes and the new operations codes can be transferred between the two computers. It should be mentioned that all developed software programs were implemented and thoroughly tested and found to perform as desired.

Two areas of computer graphics were discussed and demonstrated in this work: Dynamic graphics and Communication graphics. The applications of dynamic graphics are in video games, production of cartoons, T.V commercials, computer-aided instruction, computer aided learning, computer simulation modelling etc. Whereas communication graphics is applied in teleconferencing, transmission of satellite and radar images , videotext and teletext etc.

The developed software for pixel transfer can be implemented as a test system for applications which require a continuous flow of data in the transmission medium. An example of this is the "Spread Spectrum" project [10] currently being investigated in the digital electronics laboratory at Durham University. In this instance the pixel transmission produces a heavy traffic environment for the new Spread Spectrum Local Area Network (LAN). This is necessary to investigate the capability of the new network to handle a large amount of simultaneously generated information - it is this simultaneous transmission aspect which is unique in LANs.

REFERENCES

[1] I.E. Sutherland, SKETCHPAD:" A Man-Machine Graphical communication System", PHD Thesis MIT (1962), MIT Lincoln Laboratory Technical Report No. 296, May 1965, Abridged version in Spring Joint Computer Conference, pp 329, Spartan Books, 1963.

[2] John C. Beatty and Kellogg S. Booth, Tutorial: Computer Graphics, IEEE Computer Society.

[3] Jeffrey L. Wise and Henryk Szejnwald (NEC Microcomputer Inc., Wellesley, Mass), "Display controlle simplifies design of sophisticated graphics terminals", Electronics / April 7, 1981, pp 153-157.

[4] Introduction to the Graphical Kernel System (GKS), by F.R.A.Hopgood, D.A.Duce, J.R.Gallop and D.C.Sutcliffe.

[5] NEC Information System, Inc., CP/M-86 SYSTEM REFERENCE GUIDE Advanced Personal Computer (APC).

[6] GSX-86 Graphics Extension Programmer's Guide, Revision 00, November 1, 1983, Manual P/N 7100-0082.

[7] THE 8086 BOOK, by Russell Rector - George Alexy.

[8] Pascal/MT+86 Language Reference Manual, Copyright 1982, Digital Research P.O. Box 579, 160 Central Avenue, Pacific Grove, CA 93950 (408) 649-3896 TWX 910 360 5001.

[9] Jeffrey Wise and Henryk Szejnwald (NEC Electronics Inc., Natick, Massachusetts)," A high speed graphics display controller, Electronic Product Design", February 1982, pp 43-47.

[10] Smythe, C., "Direct Sequence Spread Spectrum Techniques in Local Area Network", PHD Thesis, Durham University, 1985.

[11] Wright, J., "A picture is worth millions of words", Eng. Comps. vol.2, no.3, May 1983, p16(21).

[12] Novac, M. ,Pinkam, R., "Inside Graphics System From Top to Bottom", Electronics. Des. vol. 31, no. 15, 21/7/83, p.183(5).

[13] Coit, S., "Raster tech advances score high marks", Data Mngmt. vol.21 , no.5, May 1983, p.14(3).

[14] Mnuel, T.,"Computer Graphics", Electronics. vol.57, no.13, June 1984, p.113(124).

[15] Mac Donald, P., "Computer graphics as present and future communicator", Irish Comp. vol.4, no.2, April 1980, p.18-20.

[16] Ellis, R.L, "Telecommunicating graphics", Comput.Graphics World (USA). vol.8, no.2, Feb 1985, p.10-12.

[17] Holland, G.L, "NAPLPS standard defines graphics and text communication", EDN(USA). vol.30, no.1, p.179-192 (10 Jan 1985).

[18] Ellis, B., "Graphics for all seasons", Computer FX 84. Computer Animation and Digital Effects. Proceeding of the Conference, London, England, 9-11 Oct 1984, p.157-71.

[19] Yonezawa, H., Maejima, H., Minorikawa, K., "CRT chip controls bit-mapped graphics and alphanumerics", Electronic Design. vol.32, no.12, p.247-59 (14 June 1984).

[20] "Computer Graphics and Applications" by D. Harris.

[21] "Computer Graphics Programming GKS - The Graphics Standard" by G. Enderle, K. Kansy, G. Pfaff.

Appendices

Appendix A1
7220/GDC

GRAPHICS DISPLAY CONTROLLER

Description

The μPD7220 Graphics Display Controller (GDC) is an intelligent microprocessor peripheral designed to be the heart of a high-performance raster-scan computer graphics and character display system. Positioned between the video display memory and the microprocessor bus, the GDC performs the tasks needed to generate the raster display and manage the display memory. Processor software overhead is minimized by the GDC's sophisticated instruction set, graphics figure drawing, and DMA transfer capabilities. The display memory supported by the GDC can be configured in any number of formats and sizes up to 256K 16-bit words. The display can be zoomed and panned, while partitioned screen areas can be independently scrolled. With its light pen input and multiple controller capability, the GDC is ideal for advanced computer graphics applications.

Features

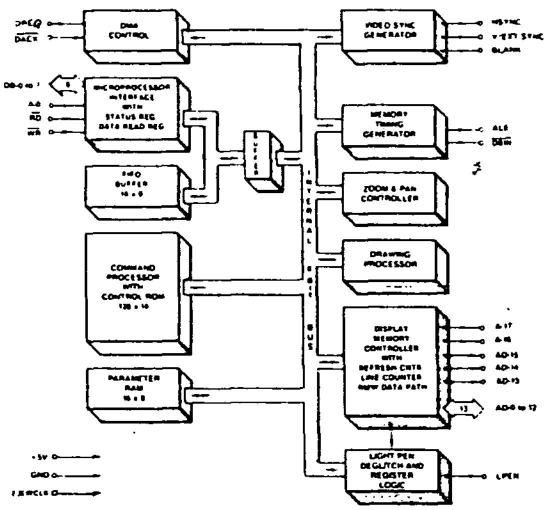
- ☐ Microprocessor Interface
  - DMA transfers with 8257- or 8237-type controllers
  - FIFO Command Buffering
- ☐ Display Memory Interface
  - Up to 256K words of 16 bits
  - Read-Modify-Write (RMW) Display Memory cycles in under 800ns
  - Dynamic RAM reresh cycles for nonaccessed memory
- ☐ Light Pen Input
- ☐ External video synchronization mode
- ☐ Graphics Mode:
  - Four megabit, bit-mapped display memory
- ☐ Character Mode:
  - 8K character code and attributes display memory
- ☐ Mixed Graphics and Characters Mode
  - 64K if all characters
  - 1 megapixel if all graphics
- ☐ Graphics Capabilities:
  - Figure drawing of lines, arc/circles, rectangles, and graphics character in 800ns per pixel
  - Display 1024-by-1024 pixels with 4 planes of color or grayscale.
  - Two independently scrollable areas
- ☐ Character Capabilities:
  - Auto cursor advance
  - Four independently scrollable areas
  - Programmable cursor height
  - Characters per row: up to 256
  - Character rows per screen: up to 100
- ☐ Video Display Format
  - Zoom magnification factors of 1 to 16
  - Panning
  - Command-settable video raster parameters
- ☐ Technology
  - Single +5 volt, NMOS, 40-pin DIP
- ☐ DMA Capability:
  - Bytes or word transfers
  - 4 clock periods per byte transferred

System Considerations

The GDC is designed to work with a general purpose microprocessor to implement a high-performance computer graphics system. Through the division of labor established by the GDC's design, each of the system components is used to the maximum extent through six-level hierarchy of simultaneous tasks. At the lowest level, the GDC generates the basic video raster timing, including sync and blanking signals. Partitioned areas on the screen and zooming are also accomplished at this level. At the next level, video display memory is modified during the figure drawing operations and data moves. Third, display memory addresses are calculated pixel by pixel as drawing progresses. Outside the GDC at the next level, preliminary calculations are done to prepare drawing parameters. At the fifth level, the picture must be represented as a list of graphics figures drawable by the GDC. Finally, this representation must be manipulated, stored, and communicated. By handling the first three levels, the GDC takes care of the high-speed and repetitive tasks required to implement a graphics system.

GDC Components

The GDC block diagram illustrates how these tasks are accomplished.



Microprocessor Bus Interface

Control of the GDC by the system microprocessor is achieved through an 8-bit bidirectional interface. The status register is readable at any time. Access to the FIFO buffer is coordinated through flags in the status register and operates independently of the various internal GDC operations, due to the separate data bus connecting the interface and the FIFO buffer.

Command Processor

The contents of the FIFO are interpreted by the command proçessor. The command bytes are decoded, and the succeeding parameters are distributed to their proper destina-

Reprinted through courtesy of NEC Electronics, U.S.A., Inc.

NOTE: These manufacturer's specifications are provided for reference. The APC may not use some of the functions described here.

tions within the GDC. The command processor yields to the bus interface when both access the FIFO simultaneously.

DMA Control

The DMA control circuitry in the GDC coordinates transfers over the microprocessor interface when using an external DMA controller. The DMA Request and Acknowledge handshake lines directly interface with a μPD8257 or μPD8237 DMA controller, so that display data can be moved between the microprocessor memory and the display memory.

Parameter RAM

The 16-byte RAM stores parameters that are used repetitively during the display and drawing processes. In character mode, this RAM holds four sets of partitioned display area parameters; in graphics mode, the drawing pattern and graphics character take the place of two of the sets of parameters.

Video Sync Generator

Based on the clock input, the sync logic generates the raster timing signals for almost any interlaced, non-interlaced, or "repeat field" interlaced video format. The generator is programmed during the idle period following a reset. In video sync slave mode, it coordinates timing between multiple GDCs.

Memory Timing Generator

The memory timing circuitry provides two memory cycle types: a two-clock period refresh cycle and the read-modify-write (RMW) cycle which takes four clock periods. The memory control signals needed to drive the display memory devices are easily generated from the GDC's ALE and $\overline{DBIN}$ outputs.

Zoom & Pan Controller

Based on the programmable zoom display factor and the display area entries in the parameter RAM, the zoom and pan controller determines when to advance to the next memory address for display refresh and when to go on to the next display area. A horizontal zoom is produced by slowing down the display refresh rate while maintaining the video sync rates. Vertical zoom is accomplished by repeatedly accessing each line a number of times equal to the horizontal repeat. Once the line count for a display area is exhausted, the controller accesses the starting address and line count of the next display area from the parameter RAM. The system microprocessor, by modifying a display area starting address, can pan in any direction, independent of the other display areas.

Drawing Processor

The drawing processor contains the logic necessary to calculate the addresses and positions of the pixels of the various graphics figures. Given a starting point and the appropriate drawing parameters, the drawing processor needs no further assistance to complete the figure drawing.

Display Memory Controller

The display memory controller's tasks are numerous. Its primary purpose is to multiplex the address and data information in and out of the display memory. It also contains the 16-bit logic unit used to modify the display memory contents during RMW cycles, the character mode line counter, and the refresh counter for dynamic RAMs. The memory controller apportions the video field time between the various types of cycles.

Light Pen Deglitcher

Only if two rising edges on the light pen input occur at the same point during successive video fields are the pulses accepted as a valid light pen detection. A status bit indicates to the system microprocessor that the light pen register contains a valid address.

Programmer's View of GDC

The GDC occupies two addresses on the system microprocessor bus through which the GDC's status register and FIFO are accessed. Commands and parameters are written into the GDC's FIFO and are differentiated based on address bit A0. The status register or the FIFO can be read as selected by the address line.

A0	READ	WRITE
0	STATUS REGISTER	PARAMETER INTO FIFO
1	FIFO READ	COMMAND INTO FIFO

GDC Microprocessor Bus Interface Registers

Commands to the GDC take the form of a command byte followed by a series of parameter bytes as needed for specifying the details of the command. The command processor decodes the commands, unpacts the parameters, loads them into the appropriate registers within the GDC, and initiates the required operations.

The commands available in the GDC can be organized into five categories as described in the following section.

GDC Command Summary

Video Control Commands

1. RESET: Resets the GDC to its idle state.
2. SYNC: Specifies the video display format.
3. VSYNC: Selects master or slave video synchronization mode.
4. CCHAR: Specifies the cursor and character row heights.

Display Control Commands

1. START: Ends Idle mode and unblanks the display.
2. BCTRL: Controls the blanking and unblanking of the display.
3. ZOOM: Specifies zoom factors for the display and graphics characters writing.
4. CURS: Sets the position of the cursor in display memory.
5. PRAM: Defines starting addresses and lengths of the display areas and specifies the eight bytes for the graphics character.
6. PITCH: Specifies the width of the X dimension of display memory.

Drawing Control Commands

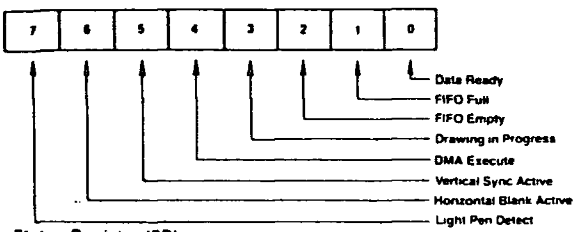
1. WDAT: Writes data words or bytes into display memory.
2. MASK: Sets the mask register contents.
3. FIGS: Specifies the parameters for the drawing processor.
4. FIGD: Draws the figure as specified above.
5. GCHRD: Draws the graphics character into display memory.

Data Read Commands

1. RDAT: Reads data words or bytes from display memory.
2. CURD: Reads the cursor position.
3. LPRD: Reads the light pen address.

DMA Control Commands

1. DMAR: Requests a DMA read transfer.
2. DMAW: Requests a DMA write transfer.



Status Register (SR)

Status Register Flags

SR-7: Light Pen Detect

When this bit is set to 1, the light pen address (LAD) register contains a deglitched value that the system microprocessor may read. This flag is reset after the 3-byte LAD is moved into the FIFO in response to the light pen read command.

SR-6: Horizontal Blanking Active

A 1 value for this flag signifies that horizontal retrace blanking is currently underway.

SR-5: Vertical Sync

Vertical retrace sync occurs while this flag is a 1. The vertical sync flag coordinates display format modifying commands to the blanked interval surrounding vertical sync. This eliminates display disturbances.

SR-4: DMA Execute

This bit is a 1 during DMA data transfers.

SR-3: Drawing in Progress

While the GDC is drawing a graphics figure, this status bit is a 1.

SR-2: FIFO Empty

This bit and the FIFO Full flag coordinate system microprocessor accesses with the GDC FIFO. When it is 1, the Empty flag ensures that all the commands and parameters previously sent to the GDC have been processed.

SR-1: FIFO Full

A 1 at this flag indicates a full FIFO in the GDC. A 0 ensures that there is room for at least one byte. This flag needs to be checked before each write into the GDC.

SR-0: Data Ready

When this flag is a 1, it indicates that a byte is available to be read by the system microprocessor. This bit must be tested before each read operation. It drops to a 0 while the data is transferred from the FIFO into the microprocessor interface data register.

FIFO Operation & Command Protocol

The first-in, first-out buffer (FIFO) in the GDC handles the command dialogue with the system microprocessor. This flow of information uses a half-duplex technique, in which the single 16-location FIFO is used for both directions of data movement, one direction at a time. The FIFO's direction is controlled by the system microprocessor through the GDC's command set. The microprocessor coordinates these transfers by checking the appropriate status register bits.

The command protocol used by the GDC requires the differentiation of the first byte of a command sequence from the succeeding bytes. This first byte contains the operation code and the remaining bytes carry parameters. Writing into the GDC causes the FIFO to store a flag value alongside the data byte to signify whether the byte was written into the command or the parameter address. The command processor in the GDC tests this bit as it interprets the entries in the FIFO.

The receipt of a command byte by the command processor marks the end of any previous operation. The number of parameter bytes supplied with a command is cut short by the receipt of the next command byte. A read operation from the GDC to the microprocessor can be terminated at any time by the next command.

The FIFO changes direction under the control of the system microprocessor. Commands written into the GDC always put the FIFO into write mode if it wasn't in it already. If it was in read mode, any read data in the FIFO at the time of the turnaround is lost. Commands which require a GDC response, such as RDAT, CURD and LPRD, put the FIFO into read mode after the command is interpreted by the GDC's command processor. Any commands and parameters behind the read-evoking command are discarded when the FIFO direction is reversed.

Read-Modify-Write Cycle

Data transfers between the GDC and the display memory are accomplished using a read-modify-write (RMW) memory cycle. The four clock period timing of the RMW cycle is used to: 1) output the address, 2) read data from the memory, 3) modify the data, and 4) write the modified data back into the initially selected memory address. This type of memory cycle is used for all interactions with display memory including DMA transfers, except for the two clock period display and RAM refresh cycles.

The operations performed during the modify portion of the RMW cycle merit additional explanation. The circuitry in the GDC uses three main elements: the Pattern register, the Mask register, and the 16-bit Logic Unit. The Pattern register holds the data pattern to be moved into memory. It is loaded by the WDAT command or, during drawing, from the parameter RAM. The Mask register contents determine which bits of the read data will be modified. Based on the contents of these registers, the Logic Unit performs the selected operations of REPLACE, COMPLEMENT, SET, or CLEAR on the data read from display memory.

The Pattern register contents are ANDed with the Mask register contents to enable the actual modification of the memory read data, on a bit-by-bit basis. For graphics drawing, one bit at a time from the Pattern register is combined with the Mask. When ANDed with the bit set to a 1 in the Mask register, the proper single pixel is modified by the Logic Unit. For the next pixel in the figure, the next bit in the Pattern register is selected and the Mask register bit is moved to identify the pixel's location within the word. The Execution word address pointer register, EAD, is also adjusted as required to address the word containing the next pixel.

In character mode, all of the bits in the Pattern register are used in parallel to form the respective bits of the modify data word. Since the bits of the character code word are used in parallel, unlike the one-bit-at-a-time graphics drawing process, this facility allows any or all of the bits in a memory word to be modified in one RMW memory cycle. The Mask register must be loaded with 1s in the positions where modification is to be permitted.

The Mask register can be loaded in either of two ways. In graphics mode, the CURS command contains a four-bit dAD field to specify the dot address. The command processor converts this parameter into the one-of-16 format used in the Mask register for figure drawing. A full 16 bits can be loaded into the Mask register using the MASK command. In addition to the character mode use mentioned above, the 16-bit MASK load is convenient in graphics mode when all of the pixels of a word are to be set to the same value.

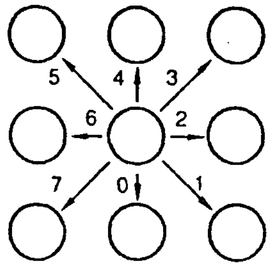
The Logic Unit combines the data read from display memory, the Pattern Register, and the Mask register to generate the data to be written back into display memory. Any one of four operations can be selected: REPLACE, COMPLEMENT, CLEAR or SET. In each case, if the respective Mask bit is 0, that particular bit of the read data is returned to memory unmodified. If the Mask bit is 1, the modification is enabled. With the REPLACE operation, the modify data simply takes the place of the read data for modification enabled bits. For the other three operations, a 0 in the modify data allows the read data bit to be returned to memory. A 1 value causes the specified operation to be performed in the bit positions with set Mask bits.

Figure Drawing

The GDC draws graphics figures at the rate of one pixel per read-modify-write (RMW) display memory cycle. These cycles take four clock periods to complete. At a clock frequency of 5MHz, this is equal to 800ns. During the RMW cycle the GDC simultaneously calculates the address and position of the next pixel to be drawn.

The graphics figure drawing process depends on the display memory addressing structure. Groups of 16 horizontally adjacent pixels form the 16-bit words which are handled by the GDC. Display memory is organized as a linearly addressed space of these words. Addressing of individual pixels is handled by the GDC's internal RMW logic.

During the drawing process, the GDC finds the next pixel of the figure which is one of the eight nearest neighbors of the last pixel drawn. The GDC assigns each of these eight directions a number from 0 to 7, starting with straight down and proceeding counterclockwise.



Drawing Directions

Figure drawing requires the proper manipulation of the address and the pixel bit position according to the drawing direction to determine the next pixel of the figure. To move to the word above or below the current one, it is necessary to subtract or add the number of words per line in display memory. This parameter is called the pitch. To move to the word to either side, the Execute word address cursor, EAD, must be incremented or decremented as the dot address pointer bit reaches the LSB or the MSB of the Mask register. To move to a pixel within the same word, it is necessary to rotate the dot address pointer register to the right or left.

The table below summarizes these operations for each direction.

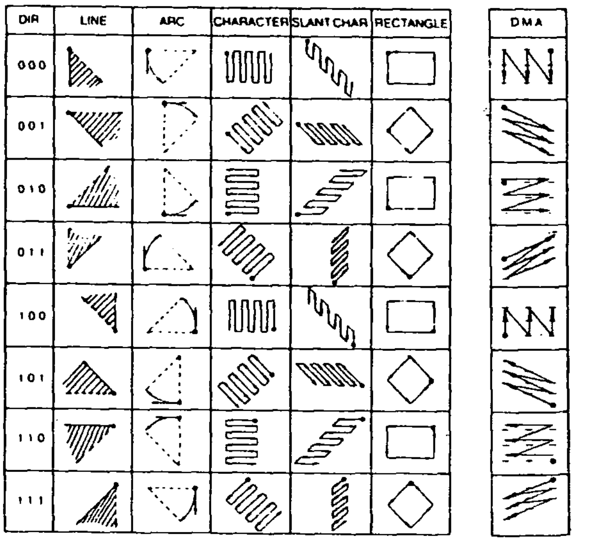
Whole word drawing is useful for filling areas in memory with a single value. By setting the Mask register to all 1s with the MASK command, both the LSB and MSB of the

DIR	OPERATIONS TO ADDRESS THE NEXT PIXEL
000	EAD · P → EAD
001	EAD · P → EAD dAD (MSB) 1 EAD · 1 → EAD dAD → LR
010	dAD (MSB) 1 EAD · 1 → EAD dAD → LR
011	EAD P → EAD dAD (MSB) 1 EAD · 1 → EAD dAD → LR
100	EAD P → EAD
101	EAD P → EAD dAD (LSB) 1 EAD 1 → EAD dAD → RR
110	dAD (LSB) 1 EAD · 1 → EAD dAD → RR
111	EAD · P → EAD dAD (LSB) - 1 · EAD - 1 → EAD dAD → RR

Where P : Pitch, LR : Left Rotate, RR : Right Rotate
EAD : Execute Word Address
dAD : Dot Address stored in the Mask Register

dAD will always be 1, so that the EAD value will be incremented or decremented for each cycle regardless of direction. One RMW cycle will be able to effect all 16 bits of the word for any drawing type. One bit in the Pattern register is used per RMW cycle to write all the bits of the word to the same value. The next Pattern bit is used for the word, etc.

For the various figures, the effect of the initial direction upon the resulting drawing is shown below:



Note that during line drawing, the angle of the line may be anywhere within the shaded octant defined by the DIR value. Arc drawing starts in the direction initially specified by the DIR value and veers into an arc as drawing proceeds. An arc may be up to 45 degrees in length. DMA transfers are done on word boundaries only, and follow the arrows indicated in the table to find successive word addresses. The slanted paths for DMA transfers indicate the GDC changing both the X and Y components of the word address when moving to the next word. It does not follow a 45 degree diagonal path by pixels.

Drawing Parameters

In preparation for graphics figure drawing, the GDC's Drawing Processor needs the figure type, direction and drawing parameters, the starting pixel address, and the pattern from the microprocessor. Once these are in place within the GDC, the Figure Draw command, FIGD, initiates the drawing operation. From that point on, the system microprocessor is not involved in the drawing process. The GDC Drawing Processor coordinates the RMW circuitry and address registers to draw the specified figure pixel by pixel.

The algorithms used by the processor for figure drawing are designed to optimize its drawing speed. To this end, the specific details about the figure to be drawn are reduced by the microprocessor to a form conducive to high-speed address calculations within the GDC. In this way the repetitive, pixel-by-pixel calculations can be done quickly, thereby minimizing the overall figure drawing time. The table below summarizes the parameters.

DRAWING TYPE	DC	D	D2	D1	DM
Initial Value*	0	8	8	-1	-1
Line	$\lvert\Delta I\rvert$	$2\lvert\Delta D\rvert - \lvert\Delta I\rvert$	$2(\lvert\Delta D\rvert - \lvert\Delta I\rvert)$	$2\lvert\Delta D\rvert$	—
Arc**	$\lceil r \sin \phi \rceil$	$r-1$	$2(r-1)$	-1	$\lfloor r \sin \theta \rfloor$
Rectangle	3	A-1	B-1	-1	A-1
Area Fill	B-1	A	A	—	—
Graphic Character***	B-1	A	A	—	—
Read & Write Data	W-1	—	—	—	—
DMAW	D-1	C-1	—	—	—
DMAR	D-1	C-1	(C-1)/2 ♦	—	—

* Initial values for the various parameters are loaded during the handling of the FIGS op code byte.

** Circles are drawn with 8 arcs, each of which span 45°, so that $\sin \phi = 1/\sqrt{2}$ and $\sin \theta = 0$.

*** Graphic characters are a special case of bit-map area filling in which B and A ≤ 8. If A = 8 there is no need to load D and D2.

Where:

-1 = all ONES value.

All numbers are shown in base 10 for convenience. The GDC accepts base 2 numbers (2s complement notation where appropriate)

- — = No parameter bytes sent to GDC for this parameter.
- ΔI = The larger of Δx or Δy.
- ΔD = The smaller of Δx or Δy.
- r = Radius of curvature, in pixels.
- φ = Angle from major axis to end of the arc. φ ≤ 45
- θ = Angle from major axis to start of the arc. θ ≤ 45
- ⌈ ⌉ = Round up to the next higher integer.
- ⌊ ⌋ = Round down to the next lower integer.
- A = Number of pixels in the initially specified direction.
- B = Number of pixels in the direction at right angles to the initially specified direction.
- W = Number of words to be accessed.
- C = Number of bytes to be transferred in the initially specified direction. (Two bytes per word if word transfer mode is selected).
- D = Number of words to be accessed in the direction at right angles to the initially specified direction.
- DC = Drawing count parameter which is one less than the number of RMW cycles to be executed.
- DM = Dots masked from drawing during arc drawing.
- ♦ = Needed only for word reads.

Graphics Character Drawing

Graphics characters can be drawn into display memory pixel-by-pixel. The up to 8-by-8 character is loaded into the GDC's parameter RAM by the system microprocessor. Consequently, there are no limitations on the character set used. By varying the drawing parameters and drawing direction, numerous drawing options are available. In area fill applications, a character can be written into display memory as many times as desired without reloading the parameter RAM.

Once the parameter RAM has been loaded with up to eight graphics character bytes by the appropriate PRAM command, the GCHRD command can be used to draw the bytes into display memory starting at the cursor. The zoom magnification factor for writing, set by the zoom command, controls the size of the character written into the display memory in integer multiples of 1 through 16. The bit values in the PRAM are repeated horizontally and vertically the number of times specified by the zoom factor.

The movement of these PRAM bytes to the display memory is controlled by the parameters of the FIGS command. Based on the specified height and width of the area to be drawn, the parameter RAM is scanned to fill the required area.

For an 8-by-8 graphics character, the first pixel drawn uses the LSB of RA-15, the second pixel uses bit 1 of RA-15, and so on, until the MSB of RA-15 is reached. The GDC jumps to the corresponding bit in RA-14 to continue the drawing. The progression then advances toward the LSB of RA-14. This snaking sequence is continued for the other 6 PRAM bytes. This progression matches the sequence of display memory addresses calculated by the drawing processor as shown above. If the area is narrower than 8 pixels wide, the snaking will advance to the next PRAM byte before the MSB is reached. If the area is less than 8 lines high, fewer bytes in the parameter RAM will be scanned. If the area is larger than 8 by 8, the GDC will repeat the contents of the parameter RAM in two dimensions, as required to fill the area with the 8-by-8 mozaic. (Fractions of the 8-by-8 pattern will be used to fill areas which are not multiples of 8 by 8.)

Parameter RAM Contents: RAM Address RA 0 to 15

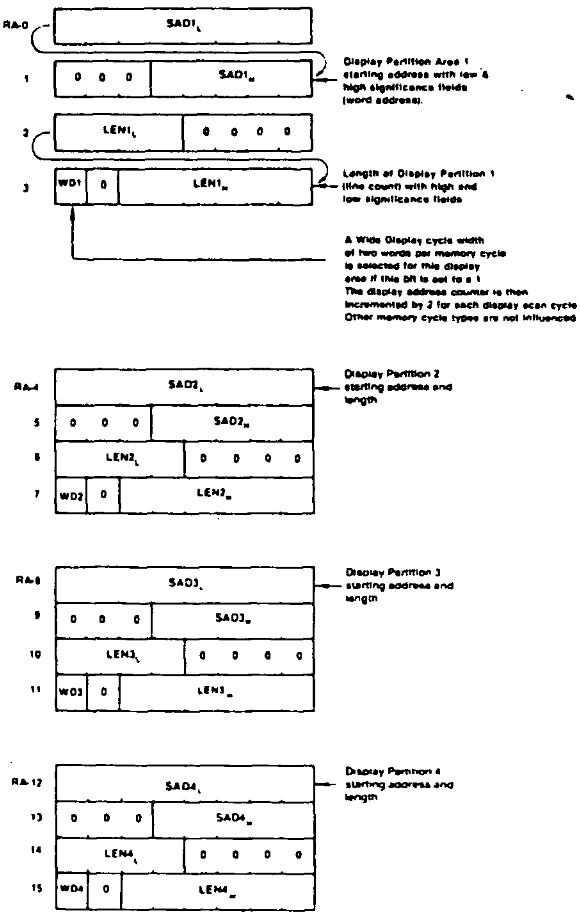
The parameters stored in the parameter RAM, PRAM, are available for the GDC to refer to repeatedly during figure drawing and raster-scanning. In each mode of operation the values in the PRAM are interpreted by the GDC in a predetermined fashion. The host microprocessor must load the appropriate parameters into the proper PRAM locations. PRAM loading command allows the host to write into any location of the PRAM and transfer as many bytes as desired. In this way any stored parameter byte or bytes may be changed without influencing the other bytes.

The PRAM stores two types of information. For specifying the details of the display area partitions, blocks of four bytes are used. The four parameters stored in each block include the starting address in display memory of each display area, and its length. In addition, there are two mode bits for each area which specify whether the area is a bit-mapped graphics area or a coded character area, and whether a 16-bit or a 32-bit wide display cycle is to be used for that area.

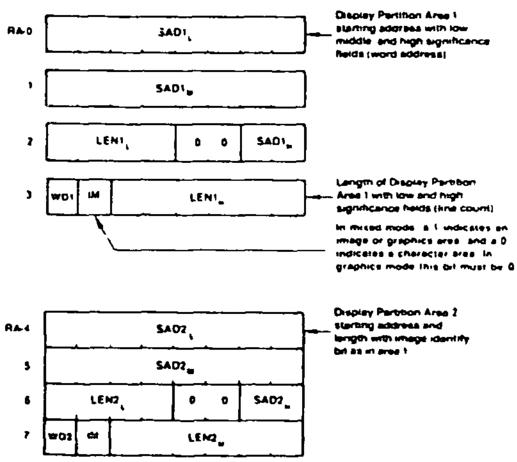
The other use for the PRAM contents is to supply the pattern for figure drawing when in a bit-mapped graphics area or mode. In these situations, PRAM bytes 8 through 16 are reserved for this patterning information. For line, arc, and rectangle drawing (linear figures) locations 8 and 9 are loaded into the Pattern Register to allow the GDC to draw dotted, dashed, etc. lines. For area filling and graphics bit-mapped character drawing locations 8 through 15 are referenced for the pattern or character to be drawn.

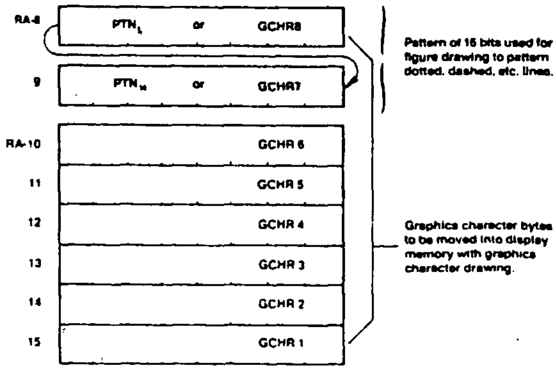
Details of the bit assignments are shown on the following pages for the various modes of operation.

Character Mode



Graphics and Mixed Graphics and Character Modes



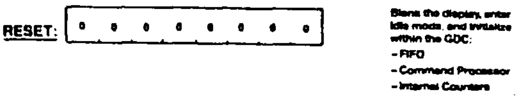


Command Bytes Summary

RESET:	0	0	0	0	0	0	0	0
SYNC:	0	0	0	0	1	1	1	DE
VSYNC:	0	1	1	0	1	1	1	M
CCHAR:	0	1	0	0	1	0	1	1
START:	0	1	1	0	1	0	1	1
BCTRL:	0	0	0	0	1	1	0	DE
ZOOM:	0	1	0	0	0	1	1	0
CURS:	0	1	0	0	1	0	0	1
PRAM:	0	1	1	1	SA			
PITCH:	0	1	0	0	0	1	1	1
WDAT:	0	0	1	TYPE		0	MOD	
MASK:	0	1	0	0	1	0	1	0
FIGS:	0	1	0	0	1	1	0	0
FIGD:	0	1	1	0	1	1	0	0
GCHRD:	0	1	1	0	1	0	0	0
RDAT:	1	0	1	TYPE		0	MOD	
CURD:	1	1	1	0	0	0	0	0
LPRD:	1	1	0	0	0	0	0	0
DMAR:	1	0	1	TYPE		1	MOD	
DMAW:	0	0	1	TYPE		1	MOD	

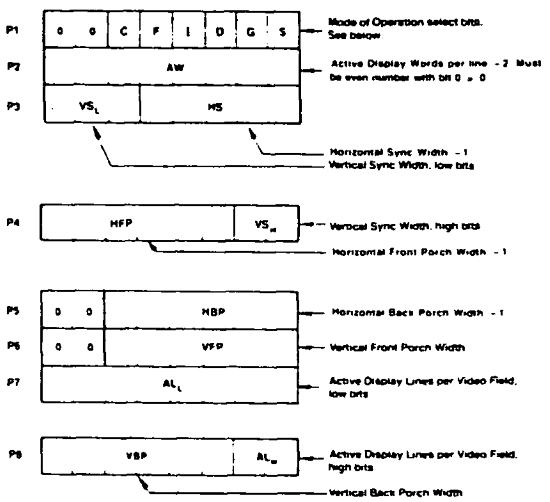
Video Control Commands

Reset



This command can be executed at any time and does not modify any of the parameters already loaded into the GDC.

If followed by parameter bytes, this command also sets the sync generator parameters as described below.
Idle mode is exited with the START command.



In graphics mode, a word is a group of 16 pixels. In character mode, a word is one character code and its attributes, if any.
The number of active words per line must be an even number from 2 to 256.
An all-zero parameter value selects a count equal to 2^n where n = number of bits in the parameter field for vertical parameters.
All horizontal widths are counted in display words.
All vertical intervals are counted in lines.

SYNC Generator Period Constraints

Horizontal Back Porch Constraints

1. In general:
   HBP $\geq$ 3 Display Word Cycles (6 clock cycles).
2. If the IMAGE or WD modes change within one video field:
   HBP $\geq$ 5 Display Word Cycles (10 clock cycles).

Horizontal Front Porch Constraints

1. If the display ZOOM function is used at other than 1X:
   HFP $\geq$ 2 Display Word Cycles (4 clock cycles).
2. If the GDC is used in the video sync Slave mode:
   HFP $\geq$ 4 Display Word Cycles (8 clock cycles).
3. If the Light Pen is used:
   HFP $\geq$ 6 Display Word Cycles (12 clock cycles).

Horizontal SYNC Constraints

1. If Interlaced display mode is used:
   HS $\geq$ 3 Display Word Cycles (6 clock cycles).

Modes of Operation Bits

C	G	Display Mode
0	0	Mixed Graphics & Character
0	1	Graphics Mode
1	0	Character Mode
1	1	Invalid

I	S	Video Framing
0	0	Noninterlaced
0	1	Invalid
1	0	Interlaced Repeat Field for Character Displays
1	1	Interlaced

Repeat Field Framing: 2 Field Sequence with ½ line offset between otherwise identical fields.
Interlaced Framing: 2 Field Sequence with ½ line offset. Each field displays alternate lines.
Noninterlaced Framing: 1 field brings all of the information to the screen.

Total scanned lines in interlace mode is odd. The sum of VFP + VS + VBP + AL should equal one less than the desired odd number of lines.

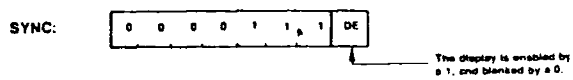
D	Dynamic RAM Refresh Cycles Enable
0	No Refresh — STATIC RAM
1	Refresh — Dynamic RAM

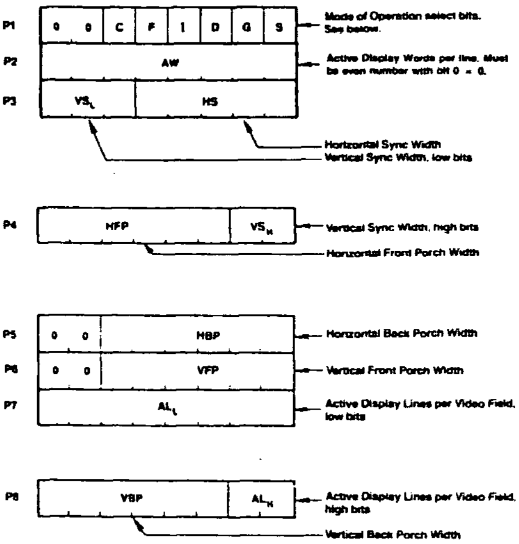
Dynamic RAM refresh is important when high display zoom factors or DMA are used in such a way that not all of the rows in the RAMs are regularly accessed during display raster generation and for otherwise inactive display memory.

F	Drawing Time Window
0	Drawing during active display time and retrace blanking
1	Drawing only during retrace blanking

Access to display memory can be limited to retrace blanking intervals only, so that no disruptions of the image are seen on the screen.

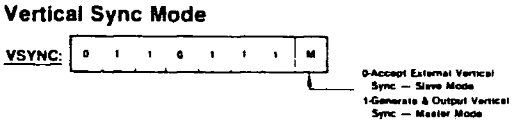
SYNC Format Specify





This command also loads parameters into the sync generator. The various parameter fields and bits are identical to those at the RESET command. The GDC is not reset nor does it enter idle mode.

Vertical Sync Mode



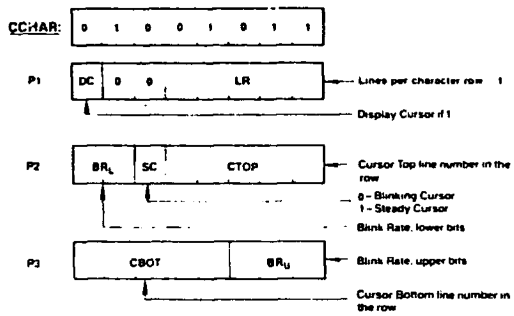
When using two or more GDCs to contribute to one image, one GDC is defined as the master sync generator, and the others operate as its slaves. The VSYNC pins of all GDCs are connected together.

Slave Mode Operation

A few considerations should be observed when synchronizing two or more GDCs to generate overlayed video via the VSYNC INPUT/OUTPUT pin. As mentioned above, the Horizontal Front Porch (HFP) must be 4 or more display cycles wide. This is equivalent to eight or more clock cycles. This gives the slave GDCs time to initialize their internal video sync generators to the proper point in the video field to match the incoming vertical sync pulse (VSYNC). This resetting of the generator occurs just after the end of the incoming VSYNC pulse, during the HFP interval. Enough time during HFP is required to allow the slave GDC to complete the operation before the start of the HSYNC interval.

Once the GDCs are initialized and set up as Master and Slaves, they must be given time to synchronize. It is a good idea to watch the VSYNC status bit of the Master GDC and wait until after one or more VSYNC pulses have been generated before the display process is started. The START command will begin the active display of data and will end the video synchronization process, so be sure there has been at least one VSYNC pulse generated for the Slaves to synchronize to.

Cursor & Character Characteristics



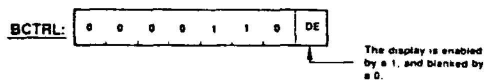
In graphics mode, LR should be set to 0.

The blink rate parameter controls both the cursor and attribute blink rates. The cursor blink-on time = blink-off time = 2 x BR (video frames). The attribute blink rate is always ½ the cursor rate but with a ¾ on-¼ off duty cycle.

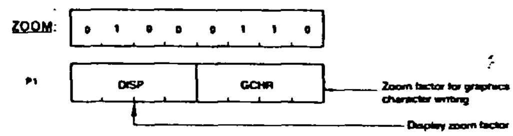
Display Control Commands

Start Display & End Idle Mode

Display Blanking Control

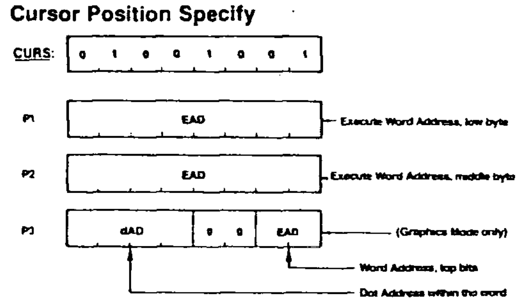


Zoom Factors Specify



Zoom magnification factors of 1 through 16 are available using codes 0 through 15, respectively.

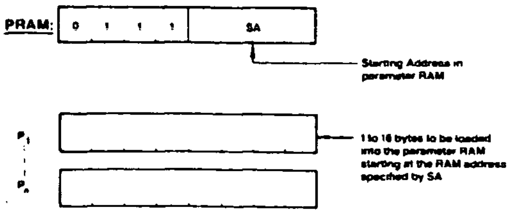
Cursor Position Specify



In character mode, the third parameter byte is not needed. The cursor is displayed for the word time in which the display scan address (DAD) equals the cursor address.

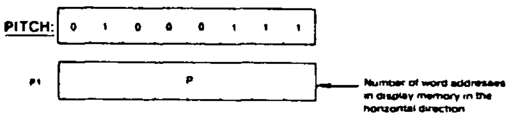
In graphics mode, the cursor word address specifies the word containing the starting pixel of the drawing; the dot address value specifies the pixel within that word.

Parameter RAM Load



From the starting address, SA, any number of bytes may be loaded into the parameter RAM at incrementing addresses, up to location 15. The sequence of parameter bytes is terminated by the next command byte entered into the FIFO. The parameter RAM stores 16 bytes of information in predefined locations which differ for graphics and character modes. See the parameter RAM discussion for bit assignments.

Pitch Specification

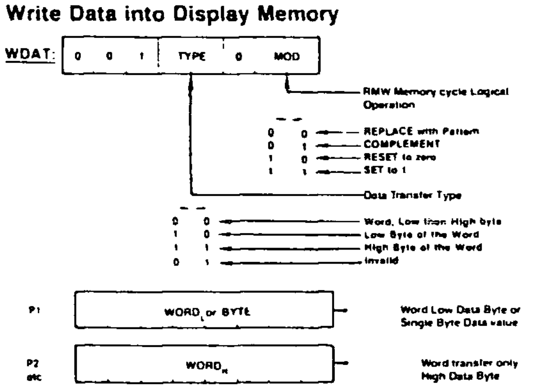


This value is used during drawing by the drawing processor to find the word directly above or below the current word, and during display to find the start of the next line.

The Pitch parameter (width of display memory) is set by two different commands. In addition to the PITCH command, the RESET (or SYNC) command also sets the pitch value. The "active words per line" parameter, which specifies the width of the raster-scan display, also sets the Pitch of the display memory. In situations in which these two values are equal there is no need to execute a PITCH command.

Drawing Control Commands

Write Data into Display Memory



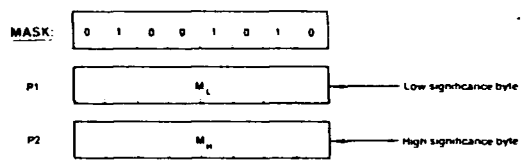
Upon receiving a set of parameters (two bytes for a word transfer, one for a byte transfer), one RMW cycle into Video Memory is done at the address pointed to by the cursor EAD. The EAD pointer is advanced to the next word, according to the previously specified direction. More parameters can then be accepted.

For byte writes, the unspecified byte is treated as all zeros during the RMW memory cycle.

In graphics bit-map situations, only the LSB of the WDAT parameter bytes is used as the pattern in the RMW operations. Therefore it is possible to have only an all ones or all zeros pattern. In coded character applications all the bits of the WDAT parameters are used to establish the drawing pattern.

The WDAT command operates differently from the other commands which initiate RMW cycle activity. It requires parameters to set up the Pattern register while the other commands use the stored values in the parameter RAM. Like all of these commands, the WDAT command must be preceeded by a FIGS command and its parameters. Only the first three parameters need be given following the FIGS opcode, to set up the type of drawing, the DIR direction, and the DC value. The DC parameter + 1 will be the number of RMW cycles done by the GDC with the first set of WDAT parameters. Additional sets of WDAT parameters will see a DC value of 0 which will cause only one RMW cycle to be executed.

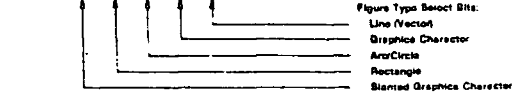
Mask Register Load

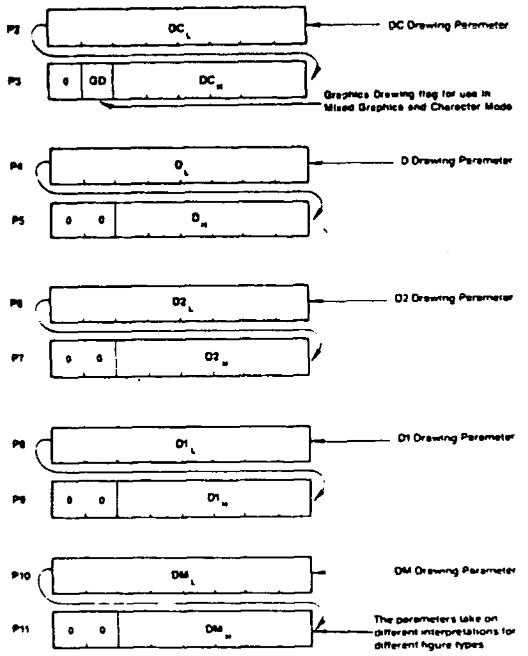


This command sets the value of the 16-bit Mask register of the figure drawing processor. The Mask register controls which bits can be modified in the display memory during a read-modify-write cycle.

The Mask register is loaded both by the MASK command and the third parameter byte of the CURS command. The MASK command accepts two parameter bytes to load a 16-bit value into the Mask register. All 16 bits can be individually one or zero, under program control. The CURS command on the other hand, puts a "1 of 16" pattern into the Mask register based on the value of the Dot Address value, dAD. If normal single-pixel-at-a-time graphics figure drawing is desired, there is no need to do a MASK command at all since the CURS command will set up the proper pattern to address the proper pixels as drawing progresses. For coded character DMA, and screen setting and clearing operations using the WDAT command, the MASK command should be used after the CURS command if its third parameter byte has been output.

Figure Drawing Parameters Specify





Valid Figure Type Select Combinations

SL	R	A	GC	L	Operation
0	0	0	0	0	Character Display Mode Drawing, Individual Dot Drawing, DMA, WDAT, and RDAT
0	0	0	0	1	Straight Line Drawing
0	0	0	1	0	Graphics Character Drawing and Area filling with graphics character pattern
0	0	1	0	0	Arc and Circle Drawing
0	1	0	0	0	Rectangle Drawing
1	0	0	1	0	Slanted graphics character drawing and slanted area filling

Only these bit combinations assure correct drawing operation.

Figure Draw Start

FIGD: 0 1 1 0 1 1 0 0

On execution of this instruction, the GDC loads the parameters from the parameter RAM into the drawing processor and starts the drawing process at the pixel pointed to by the cursor, EAD, and the dot address, dAD.

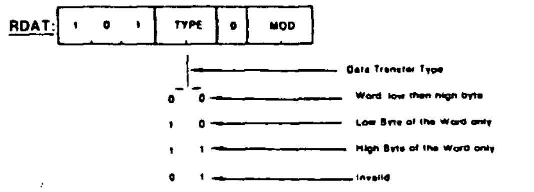
Graphics Character Draw and Area Filling Start

GCHRD: 0 1 1 0 1 0 0 0

Based on parameters loaded with the FIGS command, this command initiates the drawing of the graphics character or area filling pattern stored in Parameter RAM. Drawing begins at the address in display memory pointed to by the EAD and dAD values.

Data Read Commands

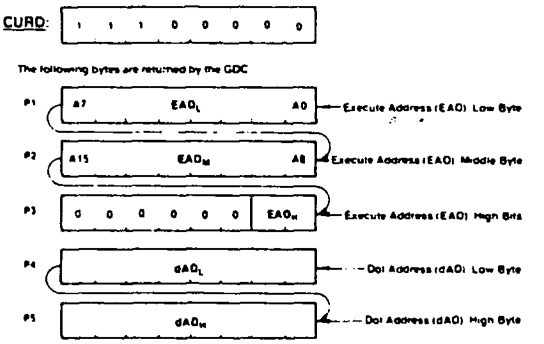
Read Data from Display Memory



Using the DIR and DC parameters of the FIGS command to establish direction and transfer count, multiple RMW cycles can be executed without specification of the cursor address after the initial load (DC number of words or bytes)

As this instruction begins to execute, the FIFO buffer direction is reversed so that the data read from display memory can pass to the microprocessor. Any commands or parameters in the FIFO at this time will be lost. A command byte sent to the GDC will immediately reverse the buffer direction back to write mode, and all RDAT information not yet read from the FIFO will be lost. MOD should be set to 00 if no modification to video buffer is desired.

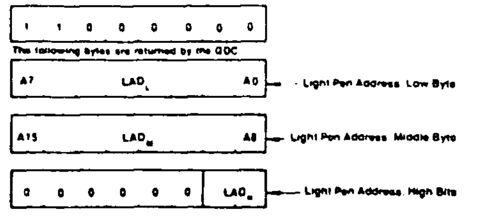
Cursor Address Read



The Execute Address, EAD, points to the display memory word containing the pixel to be addressed.

The Dot Address, dAD, within the word is represented as a 1-of-16 code for graphics drawing operations.

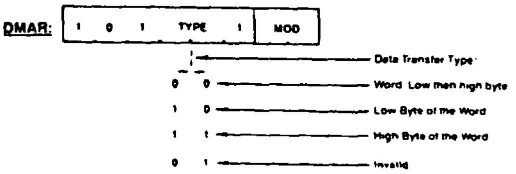
Light Pen Address Read



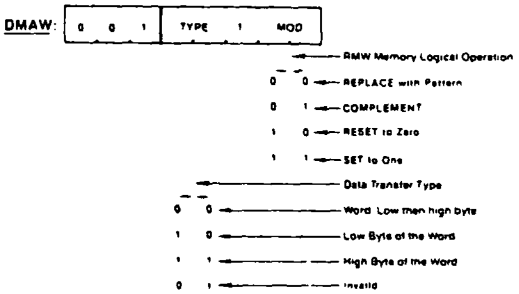
The light pen address, LAD, corresponds to the display word address, DAD, at which the light pen input signal is detected and deglitched.

The light pen may be used in graphics, character, or mixed modes but only indicates the word address of light pen position.

DMA Read Request



DMA Write Request



Absolute Maximum Ratings* (Tentative)

Ambient Temperature under Bias	0 °C to 70 °C
Storage Temperature	−65 °C to 150 °C
Voltage on any Pin with respect to Ground	−0.5V to +7V
Power Dissipation	1.5 Watt

*COMMENT: Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

t_a = 0 °C to 70 °C; V_{CC} = 5V ± 10%; GND = 0V

Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
Input Low Voltage	V_{IL}	−0.5	0.8	V	
Input High Voltage	V_{IH}	2.0	V_{CC} + 0.5	V	
Output Low Voltage	V_{OL}		0.45	V	I_{OL} = 2.2 mA
Output High Voltage	V_{OH}	2.4		V	I_{OH} = −400 µA
Input Low Leak Current	I_{IL}		−10	µA	V_I = 0V
Input High Leak Current	I_{IH}		+10	µA	V_I = V_{CC}
Output Low Leak Current	I_{OL}		−10	µA	V_O = 0V
Output High Leak Current	I_{OH}		+10	µA	V_O = V_{CC}
Clock Input Low Voltage	V_{CL}	−0.5	0.6	V	
Clock Input High Voltage	V_{CH}	3.9	V_{CC} + 1.0	V	
V_{CC} Supply Current	I_{CC}		270		

Capacitance

t_a = 25 °C; V_{CC} = GND = 0V

Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
Input Capacitance	C_{IN}		10	pF	fc = 1 MHz V_1 (unmeasured) = 0V
I/O Capacitance	$C_{I/O}$		20	pF	
Output Capacitance	C_{OUT}		20	pF	
Clock Input Capacitance	C_{ϕ}		20	pF	

AC Characteristics

t_A = 0 °C to 70 °C; V_{CC} = 5.0V ± 10%; GND = 0V

Read Cycle (GDC ↔ CPU)

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{AR}	Address Setup to $\overline{RD}$↓	0		ns	
t_{RA}	Address Hold from $\overline{RD}$↑	0		ns	
t_{RR1}	$\overline{RD}$ Pulse Width	t_{RD1} + 20	80	ns	
t_{RD1}	Data Delay from $\overline{RD}$↓		80	ns	C_L = 50 pF
t_{DF}	Data Floating from $\overline{RD}$↑	0	100	ns	
t_{RCY}	$\overline{RD}$ Pulse Cycle	4 t_{CLK}		ns	

Write Cycle (GDC ↔ CPU)

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{AW}	Address Setup to $\overline{WR}$↓	0		ns	
t_{WA}	Address Hold from $\overline{WR}$↑	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	100		ns	
t_{DW}	Data Setup to $\overline{WR}$↑	80		ns	
t_{WD}	Data Hold from $\overline{WR}$↑	0		ns	
t_{WCY}	$\overline{WR}$ Pulse Cycle	4 t_{CLK}		ns	

DMA Read Cycle (GDC ↔ CPU)

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{KR}	$\overline{DACK}$ Setup to $\overline{RD}$↓	0		ns	
t_{RK}	$\overline{DACK}$ Hold from $\overline{RD}$↑	0		ns	
t_{RR2}	$\overline{RD}$ Pulse Width	t_{RD2} + 20		ns	
t_{RD2}	Data Delay from $\overline{RD}$↓		1.5 t_{CLK} + 80	ns	C_L = 50 pF
t_{REQ}	DREQ Delay from 2XCCLK↑		120	ns	C_L = 50 pF
t_{QK}	DREQ Setup to $\overline{DACK}$↓	0		ns	
t_{DK}	$\overline{DACK}$ High Level Width	t_{CLK}		ns	
t_E	$\overline{DACK}$ Pulse Cycle	4 t_{CLK}		ns	
$t_{KQ(R)}$	DREQ↓ Delay from $\overline{DACK}$↓		2 t_{CLK} + 120	ns	C_L = 50 pF

DMA Write Cycle (GDC ↔ CPU)

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{KW}	$\overline{DACK}$ Setup to $\overline{WR}$↓	0		ns	
t_{WK}	$\overline{DACK}$ Hold from $\overline{WR}$↑	0		ns	
$t_{KQ(R)}$	DREQ↓ Delay from $\overline{DACK}$↓		t_{CLK} + 120	ns	C_L = 50 pF

R/M/W Cycle (GDC ↔ Display Memory)

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{AD}	Address/Data Delay from 2XCCLK↑		130	ns	C_L = 50 pF
t_{OFF}	Address/Data Floating from 2XCCLK↑	10	130	ns	C_L = 50 pF
t_{DIS}	Input Data Setup to 2XCCLK↓	40		ns	
t_{DIH}	Input Data Hold from 2XCCLK↓	0		ns	
t_{DBI}	DBIN Delay from 2XCCLK↓		90	ns	C_L = 50 pF
t_{RR}	ALE↑ Delay from 2XCCLK↑	30	110	ns	C_L = 50pF
t_{RF}	ALE↓ Delay from 2XCCLK↓	20	90	ns	C_L = 50 pF
t_{RW}	ALE Width	1/3 t_{CLK}		ns	C_L = 50 pF

Display Cycle **(GDC → Display Memory)**

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{VD}	Video Signal Delay from 2XCCLK↑		120	ns	C_L = 50 pF

Input Cycle **(GDC → Display Memory)**

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{PS}	Input Signal Setup to 2XCCLK↑	20		ns	
t_{PW}	Input Signal Width	t_{CLK}		ns	

Clock

Symbol	Parameter	Limits Min	Limits Max	Unit	Test Conditions
t_{CR}	Clock Rise Time		10	ns	
t_{CF}	Clock Fall Time		10	ns	
t_{CH}	Clock High Pulse Width	95		ns	
t_{CL}	Clock Low Pulse Width	95		ns	
t_{CLK}	Clock Cycle	200	2000	ns	

Appendix A2
I/O Port Adresses and Instructions for the GDC

I/O Port Addresses and Instructions for the Graphics Display Controller

INSTRUCTION	READ/ WRITE	I/O ADDRESS	DATA BUS							
Read Status	R	70	LP	HB	VS	DMA	DW	FE	FF	DR
Write Parameter	W	70	P7	P6	P5	P4	P3	P2	P1	P0
Read Data	R	72	D7	D6	D5	D4	D3	D2	D1	D0
Write Command	W	72	C7	C6	C5	C4	C3	C2	C1	C0
Graph Enable	W	76	_	_	_	_	_	_	_	1/0

Appendix A3
8237-5 Programmable DMA Controller

2.3 DIRECT MEMORY ACCESS

Because it bypasses processor intervention, DMA provides a much faster way of moving data between I/O devices and memory. Supported by the NEC LSI 8237-5 DMA Controller, DMA employs 16 address lines and 4 bits of page addressing, thus enabling it to address one megabyte of memory. Although the DMA is a synchronous device, it can interface with low-speed memory or I/O devices by using the external Ready line.

The four DMA channels are assigned as follows:

Channel 0	CRT
Channel 1	FDD
Channel 2	Reserved for graphic operations option
Channel 3	Future.

See Table 2-2 for a list of instructions and I/O addresses. Figures 2-10, 2-11, and 2-12 show the DMA registers.

Table 2-2 DMA Instructions

INSTRUCTION	READ/ WRITE	I/O ADDRESS	DATA BUS 7	6	5	4	3	2	1	0
Write Command	W	09	KS	DS	WS	PR	TM	CE	AH	MM
Write Mode	W	1B	MS1	MS0	ID	AT	TR1	TR0	CS1	CS0

Table 2-2 DMA Instructions (cont'd)

INSTRUCTION	READ/ WRITE	I/O ADDRESS	DATA BUS 7	6	5	4	3	2	1	0
Write RQ Register	W	19	—	—	—	—	—	RB	CS1	CS0
Write Single Mask	W	0B	—	—	—	—	—	MK	CS1	CS0
Write All Mask	W	1F	—	—	—	—	MB3	MB2	MB1	MB0
Read Status	R	09	RQ3	RQ2	RQ1	RQ0	TC3	TC2	TC1	TC0
CH0 DMA Address	R/W	01	A7 A15	A6 A14	A5 A13	A4 A12	A3 A11	A2 A10	A1 A9	A0 A8
CH0 DMA Count	R/W	11	W7 W15	W6 W14	W5 W13	W4 W12	W3 W11	W2 W10	W1 W9	W0 W8
CH1 DMA Address	R/W	03	A7 A15	A6 A14	A5 A13	A4 A12	A3 A11	A2 A10	A1 A9	A0 A8
CH1 DMA Count	R/W	13	W7 W15	W6 W14	W5 W13	W4 W12	W3 W11	W2 W10	W1 W9	W0 W8
CH2 DMA Address	R/W	05	A7 A15	A6 A14	A5 A13	A4 A12	A3 A11	A2 A10	A1 A9	A0 A8
CH2 DMA Count	R/W	15	W7 W15	W6 W14	W5 W13	W4 W12	W3 W11	W2 W10	W1 W9	W0 W8
CH3 DMA Address	R/W	07	A7 A15	A6 A14	A5 A13	A4 A12	A3 A11	A2 A10	A1 A9	A0 A8
CH3 DMA Count	R/W	17	W7 W15	W6 W14	W5 W13	W4 W12	W3 W11	W2 W10	W1 W9	W0 W8

Table 2-2 DMA Instructions (cont'd)

INSTRUCTION	READ/ WRITE	I/O ADDRESS	DATA BUS 7	6	5	4	3	2	1	0
CH0 Page Register	W	38	0	0	0	0	A 19	A 18	A 17	A 16
CH1 Page Register	W	3A	0	0	0	0	A 19	A 18	A 17	A 16
CH2 Page Register	W	3C	0	0	0	0	A 19	A 18	A 17	A 16
CH3 Page Register	W	3E	0	0	0	0	A 19	A 18	A 17	A 16
Read Temp Register	R	1D	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Master Clear	W	1D	—	—	—	—	—	—	—	—

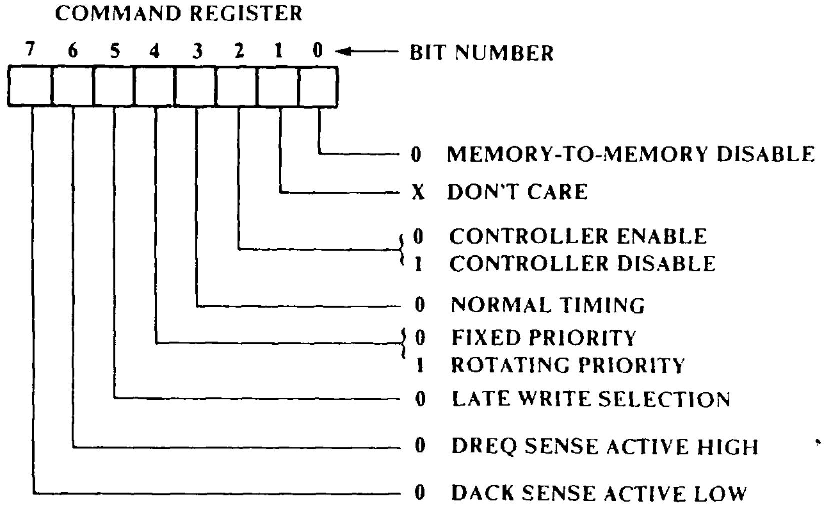


Figure 2-10 DMA Command and Mode Registers

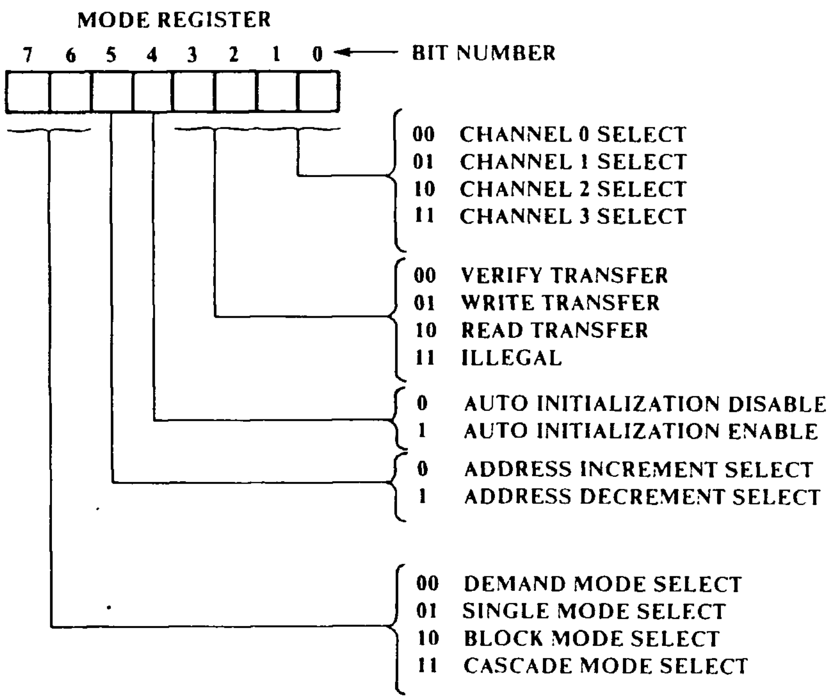
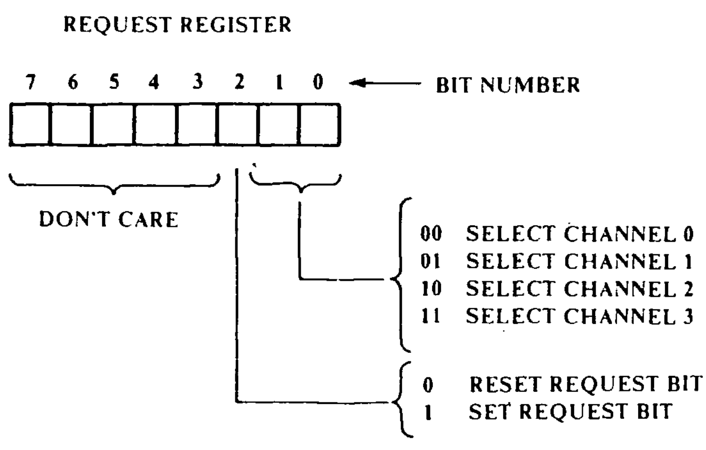
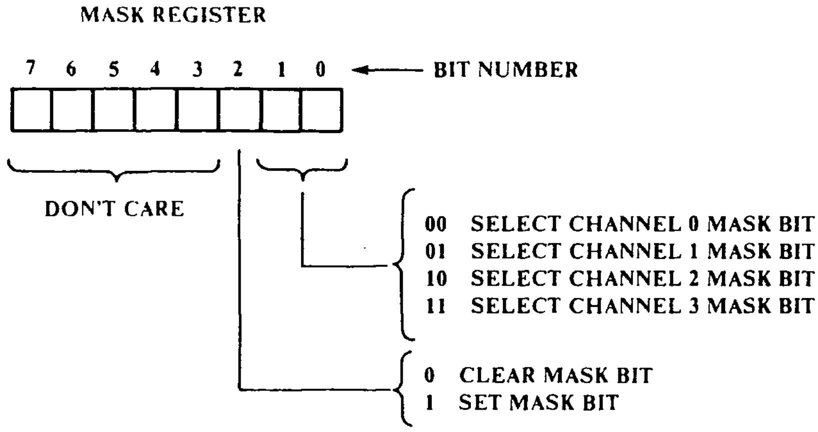


Figure 2-10 DMA Command and Mode Registers (cont'd)

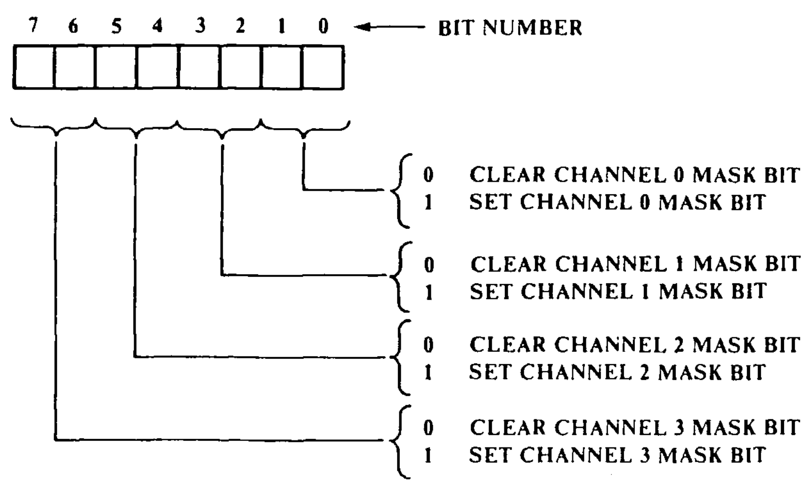


SOFTWARE REQUESTS WILL BE SERVICED ONLY IF THE CHANNEL IS IN BLOCK MODE.

Figure 2-11 DMA Request and Mask Register

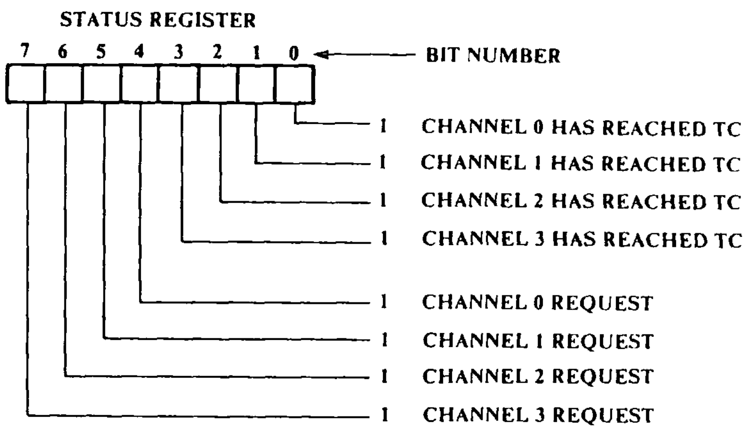


THE INSTRUCTION, WHICH SEPARATELY SETS OR CLEARS THE MASK BITS, IS SIMILAR IN FORM TO THAT USED WITH THE REQUEST REGISTER.



ALL FOUR BITS OF THE MASK REGISTER MAY ALSO BE WRITTEN WITH A SINGLE COMMAND.

Figure 2-11 DMA Request and Mask Register (cont'd)



THIS INFORMATION INCLUDES WHICH CHANNELS HAVE REACHED A TERMINAL COUNT AND WHICH CHANNELS HAVE A PENDING DMA REQUEST. BITS 0 THROUGH 3 ARE SET EVERY TIME A TC IS REACHED BY THAT CHANNEL OR AN EXTERNAL $\overline{EOP}$ IS APPLIED. THESE BITS ARE CLEARED UPON RESET AND ON EACH STATUS READ.

Figure 2-12 DMA Status Register

Appendix B1

Demonstration of a GSX-86 implementation

```
(**************************************************************************)
(*This program gives an example of GDC programming by using GSX_86 standard *)
(*interface.                                                               *)
(*It has an assembly language module named GSXPAS.I86.                     *)
(**************************************************************************)
(* a:                                                                      *)
(* mt+86 b:gsxdemo                                                         *)
(* asmt86 b:gsxpas                                                         *)
(* linkmt b:gsxdemo,b:gsxpas,fpreals,trancend,paslib/s                     *)
(* graphics                                                                *)
(* b:                                                                      *)
(* gsxdemo                                                                 *)
(**************************************************************************)
program GSX_DEMO;
CONST
    OPEN_CMD      = 1 ;
    CLOSE_CMD     = 2 ;
    CLEAR_CMD     = 3 ;

    PMARK_CMD     = 7 ;

    screensize = 32767 ;
TYPE
    cntrl_array = array [ 1..10 ] of integer ;
    intin_array = array [1..80 ]  of integer;
    intout_array = array [1..45 ] of integer;
    ptsin_array = array [ 1..100 ] of integer;
    ptsout_array = array [1..100 ] of integer;
 VAR
    contrl   :  cntrl_array;
    intin    :  intin_array;
    intout   :  intout_array;
    ptsin    :  ptsin_array;
    ptsout   :  ptsout_array;
    ch       :  char;
external procedure GSX( var ptsout:ptsout_array;
                        var intout:intout_array;
                        var ptsin :ptsin_array;
                        var intin :intin_array;
                        var contrl:cntrl_array);


(**************************************************************************)
(*Initialize the graphic device.                                           *)
(**************************************************************************)
 procedure open_wk( dev_no :integer);
 var
    i : integer;
```

```
begin
    contrl[ 1 ] := OPEN_CMD;
    contrl[ 2 ] :=0;
    contrl[ 4 ] :=10;               (*length of intin *)
    intin[ 1 ] :=dev_no;            (*workstation identifier*)
    for i :=1 to 10 do
                intin[ i ]:=1;      (*initialization parameters*)
    GSX( ptsout,intout,ptsin,intin,contrl);
end;


(***************************************************************************)
(*Erase the CRT screen.                                                   *)
(***************************************************************************)
procedure clear_it;
begin
    contrl[ 1 ] := CLEAR_CMD;
    contrl[ 2 ] :=0;
    GSX( ptsout,intout,ptsin,intin,contrl);
end;


(***************************************************************************)
(*Terminate the graphic device operation.                                 *)
(***************************************************************************)
procedure exit_gsx;
begin
    contrl[ 1 ] := CLOSE_CMD;
    contrl[ 2 ] :=0;
    GSX( ptsout,intout,ptsin,intin,contrl);
end;


(***************************************************************************)
(*Set marker colour or type.                                              *)
(***************************************************************************)
procedure set_attrib(cmd,attrib:integer);
begin
    contrl[1]:=cmd;
    contrl[2]:=0;
    intin[1]:=attrib;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;


(***************************************************************************)
(*This procedure draws a marker :                                         *)
(* x,y = coordinates of the center point                                  *)
(* scale = size of the marker                                             *)
(* color = marker colour                                                  *)
(* ptype = marker type                                                    *)
(***************************************************************************)
procedure draw_marker(x,y,scale,color,ptype:integer);
```

```
begin
    polym_scale(scale);
    set_attrib(20,color);
    set_attrib(18,ptype);
    contrl[1]:= PMARK_CMD;
    contrl[2]:=1;                    (*number of markers*)
    ptsin[1]:=x;
    ptsin[2]:=y;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;


(***************************************************************************)
(*This procedure sets marker size.                                        *)
(***************************************************************************)
procedure polym_scale(scale:integer);
begin
    contrl[1]:=19;
    contrl[2]:=1;
    ptsin[1]:=0;
    ptsin[2]:=scale;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;



(******************************** Main Program *****************************)
begin
    open_wk(1);
    clear_it;
    draw_marker(2000,2000,500,1,5);
    draw_marker(6000,6000,1000,2,4);
    draw_marker(10000,10000,2000,3,3);
    draw_marker(16000,16000,3000,4,2);
    draw_marker(20000,20000,5000,5,1);
    draw_marker(24000,24000,2000,1,5);
    draw_marker(28000,28000,1000,4,4);
    draw_marker(7500,22500,5000,1,3);
    draw_marker(22500,7500,5000,5,2);
    read(ch);
    clear_it;
    exit_gsx;
end.
```

```
;*****************************************************************************
;The assembly language module of the Pascal program GSX_DEMO.PAS.
;*****************************************************************************
        public    GSX
        name      pasgsx
        assume    cs:code, ds:data
data    segment   public
data    ends
code    segment   public
GDOS    EQU       0E0H
;
;*****************************************************************************
;GSX_86 standard interface procedure.
;*****************************************************************************
GSX     proc    near
        push    ds          ;save registers
        push    es
;
        mov     ax,ss       ;get segment address of the parameter list
        mov     ds,ax
        mov     dx,sp       ;get offset address of the parameter list
        add     dx,6        ;advance pointer past the stored data
        mov     cx,473H     ;GSX_86 function code
        int     GDOS        ;call GDOS
;
        pop     es          ;restore registers
        pop     ds
        ret     20          ;return
GSX     endp
;
code    ends
        end
```

Appendix B2

Direct Programming of the GDC

```
(**************************************************************************)
(*          DIRECT  PROGRAMMING OF GDC                                    *)
(**************************************************************************)
(*This program demonstrates the direct programming method of the GDC.     *)
(*It has an assembly language module named GDCASM.I86                     *)
(**************************************************************************)
(* a:                                                                     *)
(* mt+86 b:gdcdemo                                                        *)
(* asmt86 b:gdcasm                                                        *)
(* linkmt b:gdcdemo,b:gdcasm,fpreals,trancend,paslib/s                    *)
(* b:                                                                     *)
(* gdcdemo                                                                *)
(**************************************************************************)
program marker;

const
     mode_rep     = $20  ;        (*replace all bits with the pattern*)
     mode_com     = $21  ;        (*xor all bits with pattern*)
     mode_res     = $22  ;        (*reset 1 bits to zero*)
     mode_set     = $23  ;        (*set 1 bits to zero*)

     gplane       = 0    ;        (*green plane address*)
     bplane       = 1    ;        (*blue  plane address*)
     rplane       = 2    ;        (*red  plane address*)

     red_com      = 1    ;        (*red  component*)
     blue_com     = 4    ;        (*blue component*)
     green_com    = 2    ;        (*green component*)

    var
    p1,p2,p4     :    integer;    (*FIGS parameters*)
    zoomf        :    integer;    (*zoom factor*)
    pcolor       :    integer;    (*polymarker colour*)
    pmtype       :    integer;    (*polymarker type*)
    neccol       :    byte   ;    (*device colour*)
    l            :    byte   ;    (*dummy variable*)
    gcol         :    integer;    (*plane address*)
    gattrib      :    integer;    (*writing attribute*)
    xad,yad      :    integer;    (*x,y coordinates in actual device units*)
    ch           :    char   ;    (*dummy variable*)
    esc,s        :    string ;    (*dummy variables*)


external procedure xycv;
external procedure gchw;
external procedure ginit;
external procedure gclear;
external procedure grmaskw;
```

```
external procedure grzoomw;



(**************************************************************************)
(*This procedure draws a marker of type 'ptype' and color 'pcolor with zoom  *)
(*factor 'pzoom' at x=px,y=py on the CRT screen.                             *)
(*Xmax=640         pixels in actual device units                             *)
(*Ymax=474         pixels in actual device units                             *)
(*16>=pzoom<=0                                                               *)
(*color index      color                                                    *)
(*     -----       ------
(*       0         black                                                     *)
(*       1         red                                                       *)
(*       2         green                                                     *)
(*       3         blue                                                      *)
(*       4         cyan                                                      *)
(*       5         yellow                                                    *)
(*       6         magenta                                                   *)
(*       7         white                                                     *)
(*       8-n       white                                                     *)
(**************************************************************************)
(*marker types:  1- .   2- +   3- *   4- o  5- x                              *)
(**************************************************************************)
procedure polym(px,py,pcolor,pzoom,ptype:integer);
begin
    p1:=$12;                    (*direction and type of the figure*)
    p2:=7;                      (*DC,pixels in horizontal direction*)
    p4:=8;                      (*D1,pixels in vertical direction*)
    xad:=px;                    (*set x and y addresses in the display memory*)
    yad:=py;
    xycv;                       (*convert xad,yad to display memory addresses*)
    if ptype>5 then
            ptype:=5;
    pmtype:=(ptype-1)*8;        (*Find the start of the marker bits in the*)
                                (*assembly module*)
    if pzoom>15 then
            pzoom:=15;
    zoomf:=pzoom;
    grzoomw;                    (*set the size*)
    grmaskw;                    (*set the mask values*)
    cvcolor(pcolor);            (*get the colour components*)
    dis_char;                   (*display the character*)
end;



(**************************************************************************)
(*This procedure converts the color index to another number containing the  *)
(*proper color components in it.                                             *)
(**************************************************************************)
procedure cvcolor(color:integer);
```

```
begin
    case color of
          0 : neccol:=0;                                   (*black*)
          1 : neccol:=red_com;                             (*red*)
          2 : neccol:=green_com;                           (*green*)
          3 : neccol:=blue_com;                            (*blue*)
          4 : neccol:=blue_com + green_com;                (*cyan*)
          5 : neccol:=red_com + green_com;                 (*yellow*)
          6 : neccol:=red_com + blue_com;                  (*magenta*)
          7 : neccol:=red_com + green_com + blue_com;      (*white*)
    else
          neccol:=red_com + green_com + blue_com;          (*white*)
    end;
end;



(**************************************************************************)
procedure dis_char;
begin
    gcol:=gplane;               (*select the green plane*)
    l:=neccol & green_com;      (*check for  the green component in neccol*)
    if l<>0 then                (*if there is green then*)
           gattrib:=mode_rep    (*set writing mode to replace*)
    else
           gattrib:=mode_res;   (*if no green then set writing mode to reset*)
    gchw;                       (*draw the marker in the green plane*)
    gcol:=bplane;               (*select the blue plane*)
    l:=neccol&blue_com;         (*check for the blue component in neccol*)
    if l<>0 then                (*if there is blue then*)
           gattrib:=mode_rep    (*set writing mode to replace*)
    else
           gattrib:=mode_res;   (*if no blue then set writing mode to reset*)
    gchw;                       (*draw the marker in the blue plane*)
    gcol:=rplane;               (*select the red plane*)
    l:=neccol&bplane;           (*check for the red component in neccol*)
    if l<>0 then                (*if there is red then*)
           gattrib:=mode_rep    (*set writing mode to replace*)
    else
           gattrib:=mode_res;   (*if no red then set writing mode to reset*)
    gchw;                       (*draw the marker in the red plane*)
end;



(************************************************************************)
(*This procedure erases the entire screen.                              *)
(************************************************************************)
procedure sc_clear;
begin
    gclear;                       (*erase the graphic screen*)
    ch_clear;                     (*erase the alpha screen*)
end;
```

```
(***************************************************************************)
procedure ch_clear;
begin
    esc:=chr(27);
    s:=concat(esc,'[','2','J');  (*escape function to erase the alpha screen*)
    writeln(s);                  (*erase the alpha screen*)
end;



(**************************** MAIN PROGRAM *********************************)
begin
    ginit;                   (*initialize the 7220 chip*)
    ch_clear;               (*erase the alpha screen*)
    polym(100,100,1,12,1);
    polym(200,200,2,6,2);
    polym(300,300,5,5,3);
    polym(400,400,7,4,4);
    polym(500,250,6,3,5);
    polym(100,350,4,3,3);
    read(ch);
    sc_clear;                (*clear the screen*)
end.



;****************************************************************************
;   The assembly language module of the Pascal program:GDCDEMO.PAS
;****************************************************************************
; All the procedures in this modules can be accessed by the Pascal program if
; they are declared as external there. The  external declarations in the data
; segment of this module are all Pascal program global variables.
;****************************************************************************
        public      ginit , gscrol , gsync , gclear
        public      xycv  , wdat , gchw ,pat
        public      grmaskw , cursw , grfigs , grzoomw
        public      gdccl , gdcc
        public      damout
        name        pasgsx
        assume      cs:code, ds:data

data    segment     public
ead     dw   ?
dad     db   ?
        extrn       gcol:byte
        extrn       gattrib:byte
        extrn       zoomf:byte
        extrn       pmtype:word
```

```
        extrn      p1:byte,p2:word,p4:word
        extrn      xad:word,yad:word
data    ends
code    segment    public



;***************************************************************************
;Initialize 7220 chip
;***************************************************************************
ginit  proc near
        call   gsync          ;set sync generator parameters
        call   damout         ;send the following commands and parameters out
        db     8              ;total number of commands and parameters to be sent
;high byte=I/O address    low byte=command or parameter
;comments describe low bytes
        dw     726eh          ;Vsync
        dw     7601h          ;Graph enable
        dw     7247h          ;Pitch command
        dw     7040h          ;pitch
        dw     7246h          ;Zoom command
        dw     7000h          ;set zoom factor to zero
        dw     724bh          ;CCHAR command
        dw     7000h          ;CCHAR parameter
        call   gscrol         ;send scroll command and parameters out
        call   gclear         ;clear the display area
        mov    al,6bh         ;get the START command
        out    72h,al         ;send it out
        ret
ginit   endp



;***************************************************************************
;Mode of operation and sync generator set
;***************************************************************************
gsync   proc near
        call   damout         ;multicommand output routine
        db     9              ;Total number of commands and parameters
;high byte=I/O address  low byte=command or parameter
;comments describe low bytes
        dw     7200h          ;RESET command
        dw     7016h          ;mode of operation select bits
        dw     7026h          ;active display words per line
        dw     7046h          ;horizontal sync and vertical sync
        dw     700eh          ;horizontal front porch and vertical sync
        dw     7003h          ;horizontal back porch width
        dw     7013h          ;vertical front porch width
        dw     70dbh          ;active display lines per video field
        dw     7091h          ;vertical back porch width and active
                              ;display words per line high bits
        ret
```

```
gsync   endp



;************************************************************************
;Scroll command and parameters output
;************************************************************************
gscrol  proc near
        call  gdccl         ;GDC status check
        call  gdcc          ;GDC status check
        call  damout        ;send the following command and parameters
        db    9             ;total number of commands and parameters
;high byte=I/O address     low byte=command or parameter
;comments describe low bytes
        dw    7270h         ;PRAM command+PRAM start address of zero
        dw    7000h         ;area one starting address low bits
        dw    7000h         ;area one starting address high bits
        dw    70b0h         ;first four bits of this parameter=LEN1l
                            ;LEN1=length of area one
        dw    701dh         ;LEN1h
        dw    7000h         ;area two starting address high bits
        dw    7000h         ;area two starting address low bits
        dw    7000h         ;LEN2l
        dw    7000h         ;LEN2h
        ret
gscrol  endp



;************************************************************************
;This procedure uses the area filling techniques to clear the display area.
;A pattern of all ones is written into the display memory in reset writing
;mode.
;************************************************************************
gclear   proc near
        mov   zoomf,0       ;set the zoom factor to zero
        call  grzoomw       ;write zoom
        call  grmaskw       ;write a mask of all ones
        mov   ead,0         ;starting word address in each colour plane
        mov   dad,0         ;starting dot address in each colour plane
        mov   gattrib,22h   ;set writing mode to reset
        mov   pmtype,40     ;get the start address of the pattern bits for
                            ;the area filling
;In the following three lines p1,p2 and p4 are FIGS parameters

        mov   p1,10h        ;set the type and direction of the figure
        mov   p2,639        ;width of the screen memory in pixels
        mov   p4,475        ;hight of the screen memory in lines

        mov   cx,3          ;three planes to be cleared
gclrl:
```

```
        push  cx            ;save the loop counter
        dec   cx            ;address of the plane to be cleared
        mov   gcol,cl       ;set the colour plane address
        call  gchw          ;clear the plane
        pop   cx            ;restore the loop counter
        loop  gclrl
        ret
gclear  endp
```

```
;************************************************************************
; This procedure converts (xad,yad) coordinates on the CRT display
; to word and dot addresses ,ead and dad,in the display memory
;************************************************************************
xycv   proc near
        mov   cx,xad        ;get the x address
        and   cx,000fh      ;separate the first four bits of xad
        push  cx            ;save it
        mov   cl,4          ;get a shift factor
        mov   ax,xad        ;get the x address
        sar   ax,cl         ;shift 4 bits to the right to form Intg(x/16)
        push  ax            ;save it
        mov   ax,40h        ;load ax with the pitch value
        mov   cx,yad        ;get the y address
        mul   cx            ;multiply yad by pitch
        pop   cx            ;restore the calculated Intg(x/16)
         add  ax,cx         ;add yad*pitch to Intg(xad/16)
        mov   ead,ax        ;drop it into ead
        mov   cl,4          ;get a shift factor
        pop   ax            ;restore the first four bits of xad
        shl   ax,cl         ;shift it left
        mov   dad,al        ;save the dot address
        ret
xycv    endp
```

```
;************************************************************************
;                          Draw a graphic character
;************************************************************************
gchw    proc near
        call  wdat          ;set writing mode
        call   cursw        ;set start address of the figure
        call  pat           ;set pattern
        call  grfigs        ;specify figure parameters
        mov   al,68h        ;GCHRD command
        out   72h,al        ;send GCHRD command out to start drawing
        ret
gchw    endp
```

```
;***************************************************************************
;                        Cursor Positioning
;***************************************************************************
cursw   proc   near
        call  gdcc          ;GDC status check
        mov   al,49h        ;CURS command
        out   72h,al
        mov   ax,ead        ;word address
        out   70h,al        ;low byte first
        mov   al,ah
        out   70h,al
        mov   al,dad        ;dot address
        or    al,gcol       ;put the cursor in the proper colour plane
        out   70h,al
        ret
cursw   endp
```

```
;****************************************************************************
;          Send FIGS command and parameters out
;****************************************************************************
grfigs   proc   near
        call  gdcc          ;GDC status check
        mov   al,4ch        ;FIGS command
        out   72h,al        ;send it out
        mov   al,p1         ;type and direction of the figure
        out   70h,al        ;send first parameter out
        mov   ax,p2         ;number of pixels prependicular to
                            ;initial direction
        out   70h,al        ;send second parameter low byte
        mov   al,ah
        out   70h,al        ;send second parameter high byte
        mov   ax,p4         ;number of pixels in initial direction
        out   70h,al
        mov   al,ah
        out   70h,al
        ret
grfigs   endp
```

```
;****************************************************************************
;                    Write a mask of all ones
;****************************************************************************
grmaskw   proc  near
        call  gdcc          ;GDC status check
        mov   al,4ah        ;MASK command
        out   72h,al        ;send it out
        mov   al,0ffh       ;get the first parameter
```

```
        out    70h,al          ;send it out
        mov    al,0ffh         ;get the second parameter
        out    70h,al          ;send it out
        ret
grmaskw   endp




;***************************************************************************
;                              Zoom set
;***************************************************************************
grzoomw  proc near
        call   gdcc            ;GDC status check
        mov    al,46h          ;ZOOM command
        out    72h,al
        mov    al,zoomf        ;zoom factor
        out    70h,al
        ret
grzoomw   endp




;***************************************************************************
;                       Set writing attribute
;***************************************************************************
wdat    proc   near
        mov    al,gattrib   ;writing mode
        out    72h,al
        ret
wdat    endp




;***************************************************************************
;Load the PRAM with the graphic character or area filling pattern
;***************************************************************************
pat     proc near
        call   gdcc
        mov    al,78h          ;PRAM command and PRAM start address of 8
        out    72h,al
        mov    cx,8            ;number of parameters
        mov    bx,(offset dapat)      ;get the markers start address
        add    bx,pmtype              ;add address of the desired marker
patloop:
        mov    al,cs:byte ptr[bx]     ;get a byte
        out    70h,al                 ;load it into the PRAM
        inc    bx                     ;point to the next byte
        loop   patloop
        ret
pat     endp
```

```
;***************************************************************************
;This procedure tests three bits of the STATUS Register:'Drawing in process',
;'FIFO empty' and 'VSYNC active'.
;***************************************************************************
gdccl    proc  near
gdcc10:
        in    al,70h        ;read STATUS register
        test  al,08h
        jnz   gdcc10        ;jump if drawing in process
        not   al
        test  al,24h        ;check FIFO empty and VSYNC
        jnz   gdcc10
        ret
gdccl   endp



;***************************************************************************
; This routine checks the FIFO empty bit of the STATUS Register.
;***************************************************************************
gdcc    proc  near
gdcc00:
       in     al,70h        ;read STATUS register
       test   al,04h        ;test FIFO empty bit
       jz     gdcc00        ;jump if FIFO is not empty
       ret
gdcc    endp



;***************************************************************************
;This procedure sends the command and parameters to the I/O addresses.It
;expects a series of data bytes starting from its return address.The first
;byte must be the total number of commands and parameters.For the rest of
;the data the high byte of each word is the I/O address and the low byte is
;the command or parameter.
;***************************************************************************
damout   proc  near
        pop   bx            ;restore the return address
        mov   cl,cs:byte ptr[bx]    ;load cl with the number of
                                    ;commands and parameters
        inc   bx            ;point to the first word
        mov   ch,0          ;set high byte of cx to zero
        mov   dh,0          ;set high byte of dx to zero
damout00:
        mov   ax,cs:word ptr[bx]    ;get the word
        mov   dl,ah         ;high byte is I/O address
        out   dx,al         ;send low byte to the I/O address
        add   bx,2          ;point to the next word
        loop  damout00
        jmp   bx            ;jump to the return
```

```
damout   endp



;*************************************************************************
dapat:
      db  00h,00h,00h,10h,00h,00h,00h,00h          ;pattern data for (.)
      db  10h,10h,10h,0feh,10h,10h,10h,00h         ;pattern data for (+)
      db  92h,54h,38h,0feh,38h,54h,92h,00h         ;pattern data for (*)
      db  0feh,82h,82h,82h,82h,82h,0feh,00h        ;pattern data for (o)
      db  82h,44h,28h,10h,28h,44h,82h,00h          ;pattern
      db  0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh  ;pattern of all ones
code   ends
        end
```

Appendix B3

Demonstration for Scrolling

```
(*************************************************************************)
(*This program divides the CRT screen into two independently scrolable areas.*)
(*The function keys PF1 and PF2 select area one and area two respectively.  *)
(*Each area can be scrolled either horizontally or vertically using the four *)
(*movement keys on the keyboard.                                            *)
(*This program has an assembly language module named scrolasm.i86.         *)
(*************************************************************************)
(* a:                                                                       *)
(* mt+86 b:scrol                                                            *)
(* asmt86 b:scrolasm                                                        *)
(* linkmt b:scrol,b:scrolasm,fpreals,trancend,paslib/s                      *)
(* graphics                                                                 *)
(* b:                                                                       *)
(* scrol                                                                    *)
(*************************************************************************)
program demop ;
const
     OPEN_CMD          = 1 ;
     CLOSE_CMD         = 2 ;
     CLEAR_CMD         = 3 ;

     FILAREA_CMD       = 9 ;
     GDP_CMD           = 11 ;


     FILL_STYL_CMD     = 23 ;
     FILL_INDX_CMD     = 24 ;

     MAX_CNTL_VALS     = 10 ;
     MAX_INTIN_VALS    = 80 ;
     MAX_INTOUT_VALS   = 45 ;
     MAX_PTS_VALS      = 100 ;

     SCREENSIZE        = 32767;       (*normalized device units*)

     XMAX              = 1024;        (*display memory width in pixels*)
     YMAX              = 1024;        (*display memory length in pixels*)

TYPE
     cntrl_array  = array  [ 1..MAX_CNTL_VALS ] of integer ;
     intin_array  = array  [ 1..MAX_INTIN_VALS ] of integer ;
     intout_array = array  [ 1..MAX_INTOUT_VALS ] of integer ;
     ptsin_array  = array  [ 1..MAX_PTS_VALS ] of integer;
     ptsout_array = array  [ 1..MAX_PTS_VALS ] of integer ;
VAR
     contrl    :    cntrl_array ;       (*input control array*)
     intin     :    intin_array ;       (*input parameter array*)
     intout    :    intout_arrray ;     (*output parameter array*)
```

```
     ptsin       :     ptsin_array ;        (*input point coordinate array*)
     ptsout      :     ptsout_array ;       (*output point coordinate array*)
     S1,S        :     STRING;
     XSAD        :     INTEGER;             (*x_coordinate of the starting
address*)
     YSAD        :     INTEGER;             (*y_coordinate of the starting
address*)
     SEAD        :     INTEGER;             (*starting address*)
     SCROCMD     :     INTEGER;             (*PRAM command+PRAM start address*)
     ESC         :     CHAR;
     ch          :     char ;
     STOP1       :     BOOLEAN;
     STOP2       :     BOOLEAN;
     START       :     BOOLEAN;

external procedure GSX( var ptsout :   ptsout_array ;
                        var intout :   intout_array ;
                        var ptsin  :   ptsin_array ;
                        var intin  :   intin_array ;
                        var contrl :   cntrl_array ) ;
EXTERNAL PROCEDURE SCROL;
EXTERNAL PROCEDURE GRSCROL;



(****************************************************************************)
PROCEDURE MENU1;
VAR
   S1 : STRING;
BEGIN
    S1:=CONCAT(ESC,'[','5',';','17','m');         (*Blink and red character*)
    WRITELN(S1);
    S1:=CONCAT(ESC,'[','7',';','20','f');         (*position the cursor*)
    WRITELN(S1,'ENTER  ESC  TO STOP ');
    S1:=CONCAT(ESC,'[','23','m');                 (*white character*)
    WRITELN(S1);
    S1:=CONCAT(ESC,'[','3','B');                  (*cursor down *)
    WRITELN(S1);
    S1:=CONCAT(ESC,'[','15','C');                 (*cursor forward *)
    WRITELN(S1,'The Screen is divided into two windows.You can roam');
    WRITELN(S1,'either of them around video memory.');
    WRITELN;
    S1:=CONCAT(ESC,'[','20','C');                 (*cursor forward*)
    WRITELN(S1,'Press PF1 for window one');
    WRITELN(S1,'Press PF2 for window two');
    WRITE(S1);
END;
```

```
(******************************************************************************)
PROCEDURE MENU2;
VAR
   S : STRING;
BEGIN
    S:=CONCAT(ESC,'[','21','m');                        (*yellow color*)
    WRITELN(S);
    S:=CONCAT(ESC,'[','7','B');                         (*cursor down*)
    WRITELN(S);
    S:=CONCAT(ESC,'[','25','C');                        (*cursor forward*)
    WRITELN(S,'UP       ',CHR(167));
    WRITELN(S,'DOWN     ',CHR(169));
    WRITELN(S,'FORWARD  ',CHR(171));
    WRITELN(S,'BACKWARD ',CHR(170));
    WRITELN;
    WRITELN;
    WRITE(S,'PRESS ANY KEY TO CONTINUE');
END;


(*****************************************************************************)
PROCEDURE MENU3;
BEGIN
    S1:=CONCAT(ESC,'[','19','M');                       (*purple color*)
    WRITELN(S1);
    S1:=CONCAT(ESC,'[','8',';','20','H');               (*cursor position*)
    WRITELN(S1,'DO YOU WANT ANOTHER GO?');
    S1:=CONCAT(ESC,'[','20','C');                       (*cursor forward*)
    WRITE(S1,'(y=yes any other key=no)');
END;


(*****************************************************************************)
(*This procedure sets different attributes for subsequent operations.       *)
(*****************************************************************************)
procedure set_attrib( cmd, attribute : integer ) ;
begin
     contrl[1]:= cmd;                                   (*opcode*)
     contrl[2]:= 0;
     intin[ 1 ]  := attribute ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end;


(**************************  CLOSE WORKSTATION  ***************************)
procedure exit_gsx ;
begin
```

```
     contrl[ 1 ] := CLOSE_CMD ;
     contrl[ 2 ] := 0 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;


(************************** CLEAR THE SCREEN ********************************)
procedure clear_it ;
begin
     contrl[ 1 ] := CLEAR_CMD ;
     contrl[ 2 ] := 0 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;


(*************************** OPEN WORKSTATION ******************************)
procedure open_wk( dev_no : integer );
var
     i : integer ;
begin
     contrl[ 1 ] := OPEN_CMD ;
     contrl[ 2 ] := 0         ;
     contrl[ 4 ] := 10 ;                (*length of input parameter array*)
     intin[ 1 ] := dev_no ;             (*logical device number*)
     for i := 2 to 10 do
     intin[ i ] := 1 ;                  (*input parameter array*)
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
     intout[1]:=1024;
end ;


(**************************************************************************)
(*This procedure draws a bar :                                             *)
(* xl = x coordinate of lower left hand corner of bar                      *)
(* yl = y coordinate of lower left hand corner of bar                      *)
(* xu = x coordinate of upper right hand corner of bar                     *)
(* yu = y coordinate of upper right hand corner of bar                     *)
(**************************************************************************)
procedure draw_bar(xl,yl,xu,yu,color:integer);
begin
     set_attrib(23,1);                  (*solid fill interior style*)
     set_attrib(25,color);              (*set colour*)
     contrl[1]:=GDP_CMD;
     contrl[2]:=2;                      (*bar*)
     contrl[6]:=1;
     ptsin[1]:=xl;
     ptsin[2]:=yl;
     ptsin[3]:=xu;
     ptsin[4]:=yu;
     GSX(ptsout,intout,ptsin,intin,contrl);
end;
```

```
(***************************************************************************)
procedure draw_circle(color,x,y,r:integer);
begin
     set_attrib(25,color);                  (*set colour*)
     contrl[1]:=GDP_CMD;
     contrl[2]:=3;                          (*circle*)
     contrl[6]:=4;
     ptsin[1]:=x;                           (*x_coordinate of center*)
     ptsin[2]:=y;                           (*y_coordinate of center*)
     ptsin[3]:=0;
     ptsin[4]:=0;
     ptsin[5]:=r;                           (*radius*)
     ptsin[6]:=0;
     GSX(ptsout,intout,ptsin,intin,contrl);
end;


(***************************************************************************)
(*This procedure selects the area to be scrolled.                         *)
(***************************************************************************)
PROCEDURE AREA_SELECT;
BEGIN
    READ(CH);                              (*read choice*)
      IF CH='' THEN                       (*if area one*)
              BEGIN
                 SCROCMD:=$70;             (*PRAM command+PRAM start address of 0*)
                 XSAD:=0;                  (*starting x_coordinate of area one*)
                 YSAD:=0;                  (*starting y_coordinate of area one*)
                 START:=TRUE;
              END;
      IF CH='' THEN                       (*if area two*)
              BEGIN
                 SCROCMD:=$74;             (*PRAM command+PRAM start address of 4*)
                 XSAD:=0;                  (*starting x_coordinate of area two*)
                 YSAD:=237;                (*starting y_coordinate of area two*)
                 START:=TRUE;
              END;
      IF CH=CHR(27) THEN
                 STOP2:=TRUE;
END;


(***************************************************************************)
PROCEDURE PICTURE;
begin
    draw_bar(0,0,32767,16382,1);
    draw_bar(0,16382,32767,32767,5);
    draw_circle(1,16383,24575,5000);
    draw_circle(5,16383,8191,5000);
end;
```

```
(***********************************************************************)
(*This procedure moves the previously selected area in one of the four     *)
(*directions using the four movement keys on the keyboard.                 *)
(***********************************************************************)
PROCEDURE MOVE_AREA;
BEGIN
    REPEAT
      READ(CH);                           (*read choice*)
      IF CH = CHR(12) THEN RIGHT;
      IF CH = CHR(29) THEN LEFT;
      IF CH = CHR(11) THEN UP;
      IF CH = CHR(10) THEN DOWN;
      SEAD := YSAD*64+XSAD DIV 16;        (*calculate the starting address*)
      SCROL;                              (*move the area*)
    UNTIL CH = CHR(27);                   (*repeat until ESC is pressed*)
END;


(***********************************************************************)
(*The following four procedures adjust the x and y coordinates of the area's*)
(*starting address to move the area to the left,right,up or down.          *)
(***********************************************************************)
PROCEDURE LEFT;
 BEGIN
    XSAD:=XSAD+16;                        (*add one word to the current XSAD*)
    IF XSAD=XMAX THEN
    XSAD:=0;
 END;
(***********************************************************************)
PROCEDURE RIGHT;
 BEGIN
    XSAD:=XSAD-16;                        (*subtract one word*)
    IF XSAD=-XMAX THEN
    XSAD:=0;
 END;
(***********************************************************************)
PROCEDURE UP;
BEGIN
    YSAD:=YSAD+1;                         (*add one line to the current YSAD*)
    IF YSAD=YMAX THEN
    YSAD:=0;
END;
(***********************************************************************)
PROCEDURE DOWN;
BEGIN
    YSAD:=YSAD-1;                         (*subtract one line*)
    IF YSAD=-YMAX THEN
    YSAD:=0;
END;
```

```
(**************************** MAIN PROGRAM *********************************)
begin
    ESC:=CHR(27);
    STOP1:=FALSE;
    START:=FALSE;
    open_wk(1);
    clear_it;
       REPEAT
         MENU1;
         AREA_SELECT;                       (*select one area*)
         clear_it;                          (*clear the screen*)
         IF NOT STOP2 AND START THEN
                        BEGIN
                           MENU2;          (*show the keys to be used*)
                           read(ch);
                           clear_it;
                           PICTURE;         (*draw picture*)
                           GRSCROL;         (*divide the screen*)
                           MOVE_AREA;       (*move the previously selected area*)
                           clear_it;
                        END;
         START:=FALSE;
         STOP2:=FALSE;
         MENU3;
         READ(CH);
         clear_it;
         IF CH='y' THEN STOP1:=FALSE
         ELSE STOP1:=TRUE;
       UNTIL STOP1;
    exit_gsx;
    S:=CONCAT(ESC,'[','20','M');            (*back to defult color,green*)
    WRITELN(S);
end.
```

```
;*****************************************************************************
;The assembly launguage module of the Pascal program SCROL.PAS
;*****************************************************************************
     public    GSX
     public    GRGDCC
     public    GRSCROL
     public    SCROL
     public    DELAY
     name      pasgsx
     assume    cs:code, ds:data
data    segment  public
     EXTRN     SEAD     :WORD         ;starting address
     EXTRN     SCROCMD  :BYTE         ;PRAM command and PRAM starting address
data    ends
```

```
code segment    public
GDOS            EQU     0E0H
GRCMD           EQU     72H         ;I/O port address to send commands
GRPARA          EQU     70H         ;I/O port address to send parameters
GRSTATUS        EQU     70H         ;I/O port address to read status register



;********************* GSX-86 interface procedure **************************
GSX     proc    near
        push    ds
        push    es
        mov     ax,ss
        mov     ds,ax
        mov     dx,sp
        add     dx,6
        mov     cx,473h
        int     GDOS
        pop     es
        pop     ds
        ret     20
GSX     endp



;************************* GDC's STATUS CHECK ******************************
GRGDCC    PROC NEAR
        IN      AL,GRSTATUS     ;read status
        TEST    AL,08H          ;test if drawing in process
        JNZ     GRGDCC
        NOT     AL              ;VERTICAL SYNC(DB5)
        TEST    AL,24H          ;FIFO EMPTY(DB3)
        JNZ     GRGDCC          ;
        RET                     ;return if GDC is ready
GRGDCC  ENDP



;**************************************************************************
;This procedure divides the screen into two equal areas by loading their
;starting addresses and lengths into the PRAM.
;**************************************************************************
GRSCROL   PROC  NEAR
        CALL    GRGDCC          ;check GDC status
        MOV     AL,70H          ;PRAM command+PRAM starting address of zero
        OUT     GRCMD,AL        ;send it out
        MOV     CX,8            ;number of parameters
        mov     bx,(offset dascrol)       ;load bx with the address of the
                                          ;first byte of data
GRSCRO:
        mov     AL,cs:byte ptr[bx]        ;get the data
        OUT     GRPARA,AL                 ;send it out
```

```
        inc     bx                          ;address increment
        LOOP    GRSCRO
        RET
GRSCROL ENDP
;


;***************************************************************************
;This procedure scrols an area by changing its starting address.
;***************************************************************************
SCROL   PROC   NEAR
        CALL    GRGDCC          ;GDC status check
        MOV     AL,SCROCMD      ;PRAM command and starting address
        OUT     GRCMD,AL        ;
        MOV     AX,SEAD         ;load ax with the area's starting address
        OUT     GRPARA,AL       ;send low byte first
        MOV     AL,AH           ;then high byte
        OUT     GRPARA,AL       ;
        CALL    DELAY           ;slow the movement
        RET
SCROL   ENDP



;***************************************************************************
DELAY   PROC   NEAR
        PUSH    BX
        MOV     BX,0FFFH
DEL10:
        DEC     BX
        JNZ     DEL10
        POP     BX
        RET
DELAY   ENDP
dascrol:
        DB      00H             ;SAD1l
        DB      00H             ;SAD1h
        DB      0D0H            ;LEN1l
        DB      0EH             ;LEN1h
        DB      40H             ;SAD2l
        DB      3BH             ;SAD2h
        DB      0D0H            ;LEN2l
        DB      0EH             ;LEN2h
CODE    ENDS
;
END
```

Appendix B4

Demonstration for DMA Transfers

```
(*************************************************************************)
(*This program demonstrates the GDC's DMA capability.It shows how graphics  *)
(*windows may be generated,stored and moved.                                *)
(*It has an assembly language module named DMAASM.I86.                      *)
(*************************************************************************)
(* a:                                                                        *)
(* mt+86 b:dmademo                                                           *)
(* asmt86 b:dmaasm                                                           *)
(* linkmt b:dmademo,b:dmaasm,fpreals,trancend,paslib/s                       *)
(* graphics                                                                  *)
(* b:                                                                        *)
(* dmademo                                                                   *)
(*************************************************************************)
program MOV_DEMO;
const
   OPEN_CMD        = 1 ;
   CLOSE_CMD       = 2 ;
   CLEAR_CMD       = 3 ;

   PLINE_CMD       = 6 ;
   FILAREA_CMD     = 9 ;
   GDP_CMD         = 11;

   dev_no          = 1 ;
   screensize      = 32767;
type
   cntrl_array = array [ 1..10 ] of integer ;
   intin_array = array [1..80 ]  of integer;
   intout_array = array [1..45 ] of integer;
   ptsin_array = array [ 1..100 ] of integer;
   ptsout_array = array [1..100 ] of integer;
   addr_array = array [1..10 ] of integer;
   data_array = array [1..150 ] of integer;

VAR
   contrl         :   cntrl_array;
   intin          :   intin_array;
   intout         :   intout_array;
   ptsin          :   ptsin_array;
   ptsout         :   ptsout_array;
   addr           :   addr_array;
   duck           :   data_array;
   xypts          :   data_array;

   f              :   text;
   xw1 , yw1      :   integer;     (*x and y of window one start point*)
   xw2 , yw2      :   integer;     (*x and y of window two start point*)
   xd  , yd       :   integer;     (*x and y displacements in pixel*)
   ead            :   integer;     (*word address in the display memory*)
```

```
    plane           :   integer;    (*colour plane address*)
    DIR,DC,D1,D2    :   integer;    (*FIGS parameters*)
    Hbyte           :   integer;    (*number of horizontal bytes in window*)
    Vbyte           :   integer;    (*number of vertical bytes in window*)
    dmaaddr         :   integer;    (*window start address in the system memory*)
    dmacount        :   integer;    (*total number of bytes to be transferred*)
    x1,y1           :   integer;    (*window start address in area one*)
    x2,y2           :   integer;    (*window start address in area two*)
    i,j,k,n,m       :   integer;
    ch,ESC          :   char;
     s              :   string;


external procedure GSX( var ptsout:ptsout_array;
                        var intout:intout_array;
                        var ptsin :ptsin_array;
                        var intin :intin_array;
                        var contrl:cntrl_array);

external procedure grdmar;
external procedure grdmaw;
external procedure scrol;
external procedure delay;



(*********************** open the  workstation ****************************)
procedure open_wk( dev_no :integer);
var
   i : integer;
begin
    contrl[ 1 ] :=OPEN_CMD;
    contrl[ 2 ] :=0;
    contrl[ 4 ] :=10;
    intin[ 1 ] :=dev_no;
    for i :=1 to 10 do
           intin[ i ]:=1;
    GSX( ptsout,intout,ptsin,intin,contrl);
end;


(*********************** clear the screen **********************************)
procedure clear_it;
begin
    contrl[ 1 ] :=CLEAR_CMD;
    contrl[ 2 ] :=0;
    GSX( ptsout,intout,ptsin,intin,contrl);
end;
```

```
(*********************** close the workstation *****************************)
procedure exit_gsx;
begin
    contrl[ 1 ] :=CLOSE_CMD;
    contrl[ 2 ] :=0;
    GSX( ptsout,intout,ptsin,intin,contrl);
end;


(**************************************************************************)
procedure set_attrib(cmd,attrib:integer);
begin
    contrl[1]:=cmd;
    contrl[2]:=0;
    intin[1]:=attrib;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;


(**************************************************************************)
(*This procedure draws a circle:                                          *)
(*x,y = coordinates of center point                                       *)
(* r  = radius                                                            *)
(**************************************************************************)
procedure circle(x,y,r,col:integer);
begin
    set_attrib(23,1);
    set_attrib(25,col);
    contrl[1]:=GDP_CMD;
    contrl[2]:=3;                   (*circle*)
    contrl[6]:=4;
    ptsin[1]:=x;
    ptsin[2]:=y;
    ptsin[3]:=0;
    ptsin[4]:=0;
    ptsin[5]:=r;
    ptsin[6]:=0;
    GSX( ptsout,intout,ptsin,intin,contrl);
end;


(**************************************************************************)
(*This procedure draws a bar:                                             *)
(*left   = x coordinate of lower left hand corner of bar                  *)
(*bottom = y coordinate of lower left hand corner of bar                  *)
(*right  = x coordinate of upper right hand corner of bar                 *)
(*up     = y coordinate of upper right hand corner of bar                 *)
(**************************************************************************)
procedure bar(left,bottom,right,up,color:integer);
begin
    set_attrib(23,1);
```

```
    set_attrib(25,color);
    contrl[1]:=GDP_CMD;
    contrl[2]:=2;            (*bar*)
    contrl[6]:=1;
    ptsin[1]:=left;
    ptsin[2]:=bottom;
    ptsin[3]:=right;
    ptsin[4]:=up;
    GSX( ptsout,intout,ptsin,intin,contrl);
end;



(****************************************************************************)
(*This procedure fills a polygon specified by xypts array                   *)
(*xypts = array of x and y coordinates                                      *)
(* n    = number of points                                                  *)
(* c    = colour                                                            *)
(****************************************************************************)
procedure fill_area(n,c:integer;var xypts:data_array);
var
  i : integer;
begin
    set_attrib(23,1);
    set_attrib(25,c);
    contrl[1]:=FILAREA_CMD;
    contrl[2]:=n;
    k:=2*n;
    for i:=1 to k do
        ptsin[i]:=xypts[i];
GSX(ptsout,intout,ptsin,intin,contrl);
end;



(****************************************************************************)
(*This procedure draws a polyline connecting the points given in xypts array*)
(*xypts = array of x and y coordinates                                      *)
(* n    = number of points                                                  *)
(* c    = colour                                                            *)
(****************************************************************************)
procedure polyline(n,c:integer;xypts:data_array);
var
   i : integer;
begin
    set_attrib(17,c);
    contrl[1]:=PLINE_CMD;
    contrl[2]:=n;
    k:=2*n;
    for i:=1 to k do
            ptsin[i]:=xypts[i];
    GSX(ptsout,intout,ptsin,intin,contrl);
end;
```

```
(***************************************************************************)
(*This procedure draws a background on the screen                         *)
(***************************************************************************)
procedure background;
begin
    circle(5000,30000,1000,5);
    bar(0,70,32767,20000,3);
    draw_duck(1000,2000,15,7,1);
    draw_duck(3000,15000,15,2,6);
    draw_duck(20000,14000,15,5,1);
    draw_duck(25000,6000,15,1,7);
    grass(70);
    grass(20000);
end;



(***************************************************************************)
procedure menu;
begin
    ESC:=chr(27);
    s:=concat(ESC,'[','19',';','m');
    writeln(s);
    s:=concat(ESC,'[','50','C');
    writeln(s,'Enter ESC to continue,');
    writeln(s,'any other key to stop.');
    s:=concat(ESC,'[','20',';','m');
    writeln(s);
    s:=concat(ESC,'[','1',';','1','H');
    write(s);
end;



(***************************************************************************)
(*This procedure sets a window size                                        *)
(*width = number of words to be read horizontally                          *)
(*length = number of lines to be read vertically                           *)
(***************************************************************************)
procedure set_w_size(width,length:integer);
begin
    DIR:=4;                      (*select the direction*)
    dmacount:=2*length*width;    (*total number of bytes to be transferred*)
    Hbyte:=width;               (*number of bytes horizontally*)
    Vbyte:=2*length;             (*number of bytes vertically*)
    addr[1]:=28672;              (*starting address of a free block in the host*)
                                 (*memory*)
end;
```

```
(***************************************************************************)
(*This procedure reads a window from the display memory out to the host     *)
(*system memory.The window size must have been specified using procedure    *)
(*set_w_size.                                                                *)
(*n     =     window number                                                  *)
(*x,y   =     window start point in the display memory                       *)
(***************************************************************************)
procedure set_window(n,x,y:integer);
var
   i  :  integer;
begin
    dmaaddr:=addr[n];           (*window start address in the system memory*)
    ead:=y*64+x div 16;         (*window start address in the display memory*)
    (* DC, D and D2 are FIGS parameters *)
    DC:=Hbyte-1;                (*set DC value*)
    D1:=Vbyte-2;                (*set D value*)
    D2:=D1 div 2;               (*set D2*)
    for i:=1 to 3 do
          begin
            plane:=i-1;                 (*set the colour plane address*)
                                        (*green=0 ,blue=1, red=2*)
            grdmar;                     (*read the window in the plane i-1 and*)
                                        (*store it in the host memory*)
            dmaaddr:=dmaaddr+dmacount;
          end;
    addr[n+1]:=dmaaddr;         (*System memory's starting address for the*)
                                (*next window storage*)
end;
```

```
(***************************************************************************)
(*This procedure reads a block of data out from the host memory into the    *)
(*display memory.                                                            *)
(*n     =     window number                                                  *)
(*x,y   =     window start address in the display memory                     *)
(***************************************************************************)
procedure dis_window(n,x,y:integer);
var
   i : integer;
begin
    dmaaddr:=addr[n];           (*window start address in the system memory*)
    ead:=y*64+x div 16;         (*window start address in the display memory*)
    DC:=Hbyte-1;
    D1:=Vbyte-1;
    for i:=1 to 3 do
          begin
            plane:=i-1;                      (*set the color plane*)
            grdmaw;                          (*write the stored window into the *)
                                             (*display memory*)
            dmaaddr:=dmaaddr+dmacount;       (*host memory address for the next *)
                                             (*colour component*)
          end;
end;
```

```
(*************************************************************************)
(*This procedure reads the display area and writes it back into a second  *)
(*area of the display memory staring from x=0 and y=474.                  *)
(*************************************************************************)
procedure rwscreen;
begin
    set_w_size(40,237);         (*set window size to half the screen hight*)
    set_window(1,0,474);        (*read the lower half of the screen*)
    dis_window(1,0,948);        (*write it into the lower half of the second *)
                                (*area*)
    set_window(1,0,237);        (*read the upper half of the screen*)
    dis_window(1,0,711);        (*write it into the upper half of the second *)
end;                            (*area*)



(*************************************************************************)
(*This procedure moves a window                                           *)
(* xd    = displacement in x direction in one step                        *)
(* yd    = displacement in y direction in one step                        *)
(*steps  = number of steps                                                *)
(*************************************************************************)
procedure mov_duck(xd,yd,steps:integer);
var
   i:integer;
begin
    for i:= 1 to steps do
          begin
             x2:=x1+xd;                 (*adjust x2 and y2 for area two*)
             y2:=y1+474+yd;
             dis_window(1,x2,y2);       (*draw duck in area two*)
             ead:=474*64;               (*area two starting address*)
             scrol;                     (*switch to area two*)
             delay;                     (*slow the movement*)
             dis_window(2,x1,y1);       (*draw water on duck in area one*)
             x1:=x2+xd;                 (*adjust x1 and y1 for area one*)
             y1:=y2-474+yd;
             dis_window(1,x1,y1);       (*draw duck in area one*)
             ead:=0;                    (*area one starting address*)
             scrol;                     (*switch the display to area one*)
             delay;                     (*slow the movement*)
             dis_window(2,x2,y2);       (*display water on duck in area
two*)
          end;
end;
```

```
(*************************************************************************)
procedure grass(y:integer);
var
  i,j,k,h:integer;
begin
    set_attrib(25,2);
    k:=273;
    h:=1000;
    ptsin[5]:=0;
    for i:=1 to 60 do
          begin
            ptsin[1]:=ptsin[5];
            ptsin[2]:=y;
            ptsin[3]:=ptsin[1]+k;
            ptsin[4]:=y+h;
            ptsin[5]:=ptsin[3]+k;
            ptsin[6]:=y;
            h:=h+1000;
            if h=4000 then h:=1000;
            contrl[1]:=9;
            contrl[2]:=3;
            GSX(ptsout,intout,ptsin,intin,contrl);
          end;
end;


(*************************************************************************)
procedure readduck;
begin
    assign(f,'text.tst');
    reset(f);
    if ioresult=255 then
       writeln('error opening')
    else
       i:=1;
    while not eof(f) do
            begin
               read(f,m);
               duck[i]:=m;
               i:=i+1;
             end;
end;


(*************************************************************************)
(*This procedure draws a graphical duck on the screen:                   *)
(*x,y   = start point of the duck                                        *)
(*mag   = magnification factor                                           *)
(*cduck = duck colour                                                    *)
(*cbeak = beak colour                                                    *)
(*************************************************************************)
```

```
  procedure draw_duck(x,y,mag,cduck,cbeak:integer);
var
    n,k,j:integer;
begin
    j:=1;
    for k:=1 to 43 do
          begin
            xypts[j]:=duck[j]*mag+x;      (*calculate x-coordinates*)
            j:=j+1;
            xypts[j]:=duck[j]*mag+y;      (*calculate y-coordinates*)
            j:=j+1;
          end;
    fill_area(43,cduck,xypts);            (*draw duck*)
    xypts[87]:=xypts[1];                  (*make last x equal first x*)
    xypts[88]:=xypts[2];                  (*make last y equal first y*)
    polyline(44,0,xypts);                 (*draw line arround the duck*)
    j:=1;
    for k:=1 to 8 do
        begin
          xypts[j]:=duck[j+86]*mag+x;   (*calculate x coordinates of the beak*)
          j:=j+1;
          xypts[j]:=duck[j+86]*mag+y;   (*calculate y coordinates of the beak*)
          j:=j+1;
        end;
    fill_area(8,cbeak,xypts);                (*draw beak*)
    xypts[17]:=xypts[1];                     (*make last x equal first x*)
    xypts[18]:=xypts[2];                     (*make last y equal first y*)
    polyline(9,0,xypts);                     (*draw a line around the beak*)
    j:=1;
    for k:=1 to 10 do
          begin
            xypts[j]:=duck[j+102]*mag+x;     (*x coordinates of the wing*)
            j:=j+1;
            xypts[j]:=duck[j+102]*mag+y;     (*y coordinates of the wing*)
            j:=j+1;
          end;
    polyline(10,0,xypts);                    (*draw the wing*)
    j:=duck[123]*mag+x;
    k:=duck[124]*mag+y;
    n:=duck[125];
    circle(j,k,n,0);                         (*draw the eye*)
end;



(****************************  MAIN PROGRAM  ********************************)
begin
    readduck;
    open_wk(1);
    clear_it;
    background;
    rwscreen;
    draw_duck(8500,10500,15,7,1);
```

```
    (* xw1 = (8500/32767) * 640 = 166 pixels *)
    (* yw1 = (474 - (10500/32767) * 474 = 322 pixels *)
    xw1:=166;                           (*window one start point in actual*)
    yw1:=322;                           (*device units*)
    (* The start point of a window of water is (256,322) *)
    xw2:=256;                           (*window two start point in actual*)
    yw2:=322;                           (*device units*)
    (* The size of a window containing the duck is 6 words by 38 lines *)
    set_w_size(6,38);                   (*specify window size for duck*)
    set_window(1,xw1,yw1);              (*generate window one for duck*)
    set_window(2,xw2,yw2);              (*generate window two for water*)
    x1:=xw1;                            (*x1,y1 are in area one*)
    y1:=yw1;
    menu;
    REPEAT
        mov_duck(16,0,6);                   (*move the duck to the right*)
        mov_duck(0,3,12);                   (*down*)
        mov_duck(-16,0,6);                  (*left*)
        mov_duck(0,-3,12);                  (*up*)
        read(ch);
        if ch=chr(27) then
               begin
                   x1:=xw1;
                   y1:=yw1;
               end;
    UNTIL ch <> chr(27);
    clear_it;
    exit_gsx;
end.
```

```
;*******************************************************************************
;The assembly language module of the Pascal program DMADEMO.PAS
;*******************************************************************************
      public    GSX
      public    extdmar , extdmaw
      public    grdmar , grdmaw
      public    curw
      public    grmask
      public    wdat
      public    gdcc , gdcc2
      public    scrol
      public    damout
      public    delay
      name      pasasm
      assume    cs : code , ds : data
data  segment   public
eadl  dw  ?
      extrn     plane : byte
      extrn     ead : word
      extrn     dmaaddr : word
```

```
        extrn     dmacount : word
        extrn     DIR : byte
        extrn     DC : word
        extrn     D1 : word
        extrn     D2 : word
data    ends
code    segment   public
GDOS    EQU       0e0h


;****************** GKS standard interface procedure *********************
GSX       proc   near
          push   ds
          push   es
;
          mov    ax,ss
          mov    ds,ax
          mov    dx,sp
          add    dx,6
          mov    cx,473h
          int    GDOS
;
          pop    es
          pop    ds
          ret    20
GSX       endp


;*************************************************************************
;This procedure programs the external DMA controller to write data into the
;system memory.
;dmaaddr = address of the first byte of data in system memory.
;dmacount = total number of bytes to transferred.
;dmaaddr and dmacount must be determined in the Pascal program.
;*************************************************************************
extdmaw  proc near
          mov    ax,dmaaddr           ;DMA address in system memory
          mov    addrl,al
          mov    addrh,ah
          call   damout               ;multicommand output
          db     7                    ;total number of commands and parameters

;in the following data words high byte=I/O address,low byte=command or
parameter

          dw     0900h                ;write command register
          dw     1b16h                ;write mode register
          dw     1906h                ;write request register
          dw     0b02h                ;write mask register
          dw     3c07h                ;write channel2 page register
addrl     db     00                   ;channel2 DMA address low byte
          db     05h                  ;I/O address for CH2 DMA address
```

```
addrh   db      00                      ;CH2 DMA address high byte
        db      05h
        mov     ax,dmacount             ;CH2 DMA count
        out     15h,al
        mov     al,ah
        out     15h,al
        ret
extdmaw  endp



;***************************************************************************
;This procedure reads a block of data from the display memory out to the
;system memory.First it calls 'extdmaw' to program the external DMA controller,
;then it programs the GDC for the transfer.
;***************************************************************************
grdmar  proc near
        call    extdmaw
        call    curw                    ;write cursor
        call    grmask                  ;write mask
        mov     al,4ch                  ;FIGS command
        out     72h,al                  ;send it out
        mov     al,DIR                  ;type and direction
        out     70h,al                  ;send the parameter out
        mov     ax,DC                   ;DC drawing parameter
        out     70h,al
        mov     al,ah
        out     70h,al
        mov     ax,D1                   ;D1 drawing parameter
        out     70h,al
        mov     al,ah
        out     70h,al
        mov     ax,D2                   ;D2 drawing parameter
        out     70h,al
        mov     al,ah
        out     70h,al
        push    ds                      ;register save
        push    es
        push    di
        push    si
        mov     al,0a4h                 ;DMAR command
        out     72h,al
        call    gdcc                    ;GDC status check
        call    gdcc2                   ;GDC status check
        pop     si                      ;restore registers
        pop     di
        pop     es
        pop     ds
        ret
grdmar   endp
```

```
;*****************************************************************************
;This procedure programs the external DMA controller to read data from the
;system memory.
;dmaaddr = address of the first byte of data in the system memory.
;dmacount = total number of bytes to be transferred.
;dmaaddr and dmacount must be determined in the Pascal program.
;*****************************************************************************
extdmar proc  near
        call  damout                ;multicommand output
        db    5                     ;total number of bytes to be sent
        dw    0900h                 ;write command register
        dw    1b1ah                 ;write DMA MODE register
        dw    1906h                 ;write request register
        dw    0b02h                 ;write DMA MASK register
        dw    3c07h                 ;write page register
        mov   ax,dmaaddr            ;DMA address in system memory
        out   05h,al
        mov   al,ah
        out   05h,al
        mov   ax,dmacount           ;total number of bytes to be transferred
        out   15h,al                ;send to the I/O port
        mov   al,ah
        out   15h,al
        ret
extdmar  endp


;*****************************************************************************
;This procedure transfers data from system memory into the display memory.
;*****************************************************************************
grdmaw   proc  near
        call  extdmar
        call  wdat;                 ;set writing mode
        call  curw                  ;write cursor
        call  grmask                ;write mask
        mov   al,4ch                ;FIGS command
        out   72h,al
        mov   al,DIR                ;figure type and direction
        out   70h,al
        mov   ax,DC                 ;DC drawing parameter
        out   70h,al
        mov   al,ah
        out   70h,al
        mov   ax,D1                 ;D1 drawing parameter
        out   70h,al
        mov   al,ah
        out   70h,al
        push  ds                    ;save registers
        push  es
        push  di
        push  si
        mov   al,24h                ;DMAW command
        out   72h,al
```

```
        call   gdcc                  ;GDC status check
        call   gdcc2                 ;GDC status check
        pop    si                    ;restore registers
        pop    di
        pop    es
        pop    ds
        ret
grdmaw    endp


;***************************************************************************
;This procedure puts the cursor in the previously specified plane and word
;addresses in the Pascal section.
;***************************************************************************
curw    proc   near
        mov    al,49h                ;CURS CMD
        out    72h,al
        mov    ax,ead                ;word address
        out    70h,al                ;send low byte first
        mov    al,ah                 ;then high byte
        out    70h,al
        mov    al,plane              ;third parameter is plane address because
                                     ;dad is zero for DMA transfers
        out    70h,al
        ret
curw    endp


;*********************** Write a mask of all ones *************************
grmask  proc   near
        call   damout
        db     3                     ;three bytes to be sent
        dw     724ah                 ;MASK CMD
        dw     70ffh                 ;load mask register with all ones
        dw     70ffh
        ret
grmask    endp


;**************************************************************************
;This procedure expects a series of data bytes starting from its return
;address.The first byte must be total number of commands and parameters.
;For the rest of the data the high byte of each word is I/O address and
;the low byte is command or parameter to be sent.
;********************** Multicommand output routine **********************
damout    proc near
        pop    bx                    ;restore return address
        mov    cl,cs:byte ptr[bx]    ;get the number of bytes to be sent
        inc    bx                    ;point to the next byte
```

```
        mov     ch,0                    ;set high byte of cx register to zero
        mov     dh,0                    ;set high byte of dx register to zero
mout10:
        mov     ax,cs:word ptr[bx]      ;get the word
        mov     dl,ah                   ;high byte is I/O address
        out     dx,al                   ;send low byte of the word out
        add     bx,2                    ;point to the next word
        loop    mout10
        jmp     bx                      ;return
damout  endp


;***************************************************************************
;This procedure tests FIFO Empty bit of the GDC's status register.
;***************************************************************************
gdcc    proc near
gdcc10:
        in      al,70h                  ;read status register
        test    al,04h                  ;test FIFO Empty bit
        jz      gdcc10                  ;jump if FIFO is not empty
        ret                             ;return if FIFO is empty
gdcc    endp


;***************************************************************************
;This procedure test DMA Execute bit of the GDC's status register.
;***************************************************************************
gdcc2    proc  near
ann:    in      al,70h                  ;read status register
        test    al,10h                  ;test DMD Execute bit
        jnz     ann                     ;jump if DMA is busy
        ret                             ;return if DMA transfer is finished
gdcc2   endp


;***************************************************************************
;This procedure sets writing attributes.
;***************************************************************************
wdat    proc near
        mov     al,23h                  ;WDAT CMD
        out     72h,al                  ;send it out
        ret
wdat    endp
```

```
;*************************************************************************
;This procedure changes starting address of the display area.
;*************************************************************************
scrol  proc  near
        call  gdcc              ;GDC status check
        mov   al,70h            ;PRAM command+PRAM start address of zero
        out   72h,al
        mov   ax,ead            ;word address
        out   70h,al            ;send low byte first
        mov   al,ah             ;then high byte
        out   70h,al
        ret
scrol  endp


;*************************************************************************
delay   proc  near
        mov   bx,0afffh
del00:  dec   bx
        jnz   del00
        ret
delay   endp
code   ends
        end
```

Appendix B5

Demonstration for Read/Write through the FIFO buffer

```
(**************************************************************************)
(*This program demonstrates how the host microprocessor can read data from *)
(*or write data into the GDC's display memory,through the FIFO buffer.It   *)
(*shows how windows can be generated,stored and moved.*)
(**************************************************************************)
(*This program has an assembly language module named FIFOASM.I86.         *)
(**************************************************************************)
(* a:                                                                      *)
(* mt+86 b:fiforw                                                          *)
(* asmt86 b:fifoasm                                                        *)
(* linkmt b:fiforw,b:fifoasm,fpreals,trancend,paslib/s                     *)
(* graphicss                                                               *)
(* b:                                                                      *)
(* fiforw                                                                  *)
(**************************************************************************)
program FIFO ;
const
     OPEN_CMD          = 1 ;
     CLOSE_CMD         = 2 ;
     CLEAR_CMD         = 3 ;

     PLINE_CMD         = 6 ;
     TEXT_CMD          = 8 ;
     FILAREA_CMD       = 9 ;
     GDP_CMD           = 11 ;

     TEXT_HGT_CMD      = 12 ;

     LINE_STYL_CMD     = 15 ;
     FILL_STYL_CMD     = 23 ;
     FILL_INDX_CMD     = 24 ;

     MAX_CNTL_VALS     = 10 ;
     MAX_INTIN_VALS    = 80 ;
     MAX_INTOUT_VALS   = 45 ;
     MAX_PTS_VALS      = 100 ;

     SCREENSIZE        = 32767 ;
type
     cntrl_array   = array  [ 1..MAX_CNTL_VALS ]   of integer ;
     intin_array   = array  [ 1..MAX_INTIN_VALS ]  of integer ;
     intout_array  = array  [ 1..MAX_INTOUT_VALS ] of integer ;
     ptsin_array   = array  [ 1..MAX_PTS_VALS ]    of integer ;
     ptsout_array  = array  [ 1..MAX_PTS_VALS ]    of integer ;
     data_array    = array  [ 1..200 ]             of integer ;
    var
     contrl          :     cntrl_array ;
     intin           :     intin_array ;
```

```
     intout              :      intout_arrray ;
     ptsin               :      ptsin_array ;
     ptsout              :      ptsout_array ;
     duck                :      data_array ;
     xypts               :      data_array ;

     x1,y1               :      integer ;  (*window start point in area one*)
     x2,y2               :      integer ;  (*window start point in area two*)
     sad                 :      integer ;  (*start address in the display
memory*)
     xad,yad             :      integer ;  (*x,y addresses in the display
memory*)
     addr                :      integer ;  (*address in the system memory*)
     count               :      integer ;  (*window size in bytes*)
     plane               :      integer ;  (*colour plane address*)
     len,wid             :      integer ;  (*length and width of the window*)
     i,j,k,m             :      integer ;  (*dummy variables*)
     ch,ESC              :      char    ;
     s                   :      string  ;
     stop                :      boolean ;
     f                   :      text    ;

external procedure GSX( var ptsout :   ptsout_array ;
                        var intout :   intout_array ;
                        var ptsin  :   ptsin_array ;
                        var intin  :   intin_array ;
                        var contrl :   cntrl_array ) ;
external procedure xycv;
external procedure rdat;
external procedure display;
external procedure scrol;
external procedure gchw;



(**************************************************************************)
procedure set_attrib(cmd,attrib:integer);
begin
    contrl[1]:=cmd;
    contrl[2]:=0;
    intin[1]:=attrib;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;


(************************ Close the workstation *************************)
procedure exit_gsx ;
begin
    contrl[ 1 ] := CLOSE_CMD ;
    contrl[ 2 ] := 0 ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;
```

```
(**************************** Clear the screen ****************************)
procedure clear_it ;
begin
    contrl[ 1 ] := CLEAR_CMD ;
    contrl[ 2 ] := 0 ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;



(************************* Open the workstation *************************)
procedure open_wk( dev_no : integer );
var
    i : integer ;
begin
    contrl[ 1 ] := OPEN_CMD ;
    contrl[ 2 ] := 0          ;
    contrl[ 4 ] := 10 ;
    intin[ 1 ] := dev_no ;
    for i := 2 to 10 do
             intin[ i ] := 1 ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;



(************************************************************************)
(*This procedure draws a bar:                                           *)
(*left   = x coordinate of lower left hand corner of bar                *)
(*bottom = y coordinate of lower left hand corner of bar                *)
(*right  = x coordinate of upper right hand corner of bar               *)
(* up    = y coordinate of upper right hand corner of bar               *)
(************************************************************************)
procedure draw_bar(left,bottom,right,up,color:integer);
begin
    set_attrib(23,1) ;
    set_attrib(25,color) ;
    contrl[1] := GDP_CMD ;
    contrl[2] := 2 ;          (*bar*)
    contrl[6] := 1;
    ptsin[1] :=left ;
    ptsin[2] :=bottom ;
    ptsin[3] :=right ;
    ptsin[4] :=up ;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;
```

```
(**************************************************************************)
(*This procedure draws a circle:                                         *)
(*x,y = coordinates of center point                                      *)
(* r  = radius                                                           *)
(**************************************************************************)
procedure circle(x,y,r,color:integer);
begin
    set_attrib(25,color);
    contrl[1] := GDP_CMD ;
    contrl[2] := 3 ;          (*circle*)
    contrl[6] := 4 ;
    ptsin[1] := x ;
    ptsin[2] := y ;
    ptsin[3] := 0 ;
    ptsin[4] := 0 ;
    ptsin[5] := r ;
    ptsin[6] := 0 ;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;


(**************************************************************************)
(*This procedure fills a polygon specified by xypts array                *)
(*xypts = array of x and y coordinates                                   *)
(* n    = number of points                                               *)
(* c    = colour                                                         *)
(**************************************************************************)
procedure fill_area(n,c:integer;var xypts:data_array);
var
    i : integer ;
begin
    set_attrib(23,1);
    set_attrib(25,c);
    contrl[1] := FILAREA_CMD ;
    contrl[2] := n ;
    k := 2*n;
    for i:=1 to k do
           ptsin[i] := xypts[i] ;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;


(**************************************************************************)
(*This procedure draws a polyline which connects the points given in xypts *)
(*array.                                                                   *)
(*xypts = array of x and y coordinates                                     *)
(* n    = number of points                                                 *)
(* c    = colour                                                           *)
(**************************************************************************)
procedure polyline(n,c:integer;var xypts:data_array);
```

```
var
   i : integer ;
begin
    set_attrib(17,c);
    contrl[1] := PLINE_CMD ;
    contrl[2] := n ;
    k := 2*n;
    for i:=1 to k do
           ptsin[i] := xypts[i];
    GSX(ptsout,intout,ptsin,intin,contrl);
end;



(******************************************************************************)
procedure menu;
    begin
        ESC:=chr(27);
        s:=concat(ESC,'[','21','m');
        writeln(s);
        s:=concat(ESC,'[','7',';','50','H');
        writeln(s,'Press ESC to continue,');
        s:=concat(ESC,'[','8',';','50','H');
        writeln(s,'any other key to stop.');
        s:=concat(ESC,'[','20','m');
        writeln(s);
        s:=concat(ESC,'[','0',';','0','H');
        writeln(s);
    end;



(******************************************************************************)
(*This procedure reads data for drawing a duck, from the file text.tst.     *)
(******************************************************************************)
procedure readdata;
  var
    k:integer;
begin
    assign(f,'text.tst');
    reset(f);
    if ioresult=255 then
           writeln('error opening')
    else
           i:=1;
    while not eof(f) do
               begin
                    read(f,m);
                    duck[i]:=m;
                    i:=i+1;
               end;
end;
```

```
(**************************************************************************)
(*This procedure draws a background on the screen.                       *)
(**************************************************************************)
procedure background;
begin
    draw_bar(0,100,12500,7000,5);
    draw_bar(19500,100,32767,7000,4);
    draw_bar(0,13000,12500,20000,6);
    draw_bar(19500,13000,32767,20000,3);
    draw_bar(0,26000,12500,32700,1);
    draw_bar(19500,26000,32767,32700,7);
end;



(************************************************************************)
(*This procedure draws a graphical duck on the screen:                  *)
(*x,y   = start point of the duck                                       *)
(*mag   = magnification factor                                          *)
(*cduck = duck colour                                                   *)
(*cbeak = beak colour                                                   *)
(************************************************************************)
procedure draw_duck(x,y,mag,cduck,cbeak:integer);
var
    n,k,j : integer ;
begin
    j:=1;
    for k:=1 to 43 do
            begin
                 xypts[j]:=duck[j]*mag+x;      (*calculate x-coordinates*)
                 j:=j+1;
                 xypts[j]:=duck[j]*mag+y;      (*calculate y-coordinates*)
                 j:=j+1;
            end;
    fill_area(43,cduck,xypts);                 (*draw duck*)
    xypts[87]:=xypts[1];                       (*make last x equal first x*)
    xypts[88]:=xypts[2];                       (*make last y equal first y*)
    polyline(44,0,xypts);                      (*draw line arround the duck*)
    j:=1;
    for k:=1 to 8 do
            begin
                xypts[j]:=duck[j+86]*mag+x;    (*calculate x of beak*)
                j:=j+1;
                xypts[j]:=duck[j+86]*mag+y;    (*calculate y of beak*)
                j:=j+1;
            end;
    fill_area(8,cbeak,xypts);                  (*draw beak*)
    xypts[17]:=xypts[1];                       (*last x equal first x*)
    xypts[18]:=xypts[2];                       (*last y equal first y*)
    polyline(9,0,xypts);                       (*draw line arround beak*)
    j:=1;
    for k:=1 to 10 do
```

```
            begin
                 xypts[j]:=duck[j+102]*mag+x;      (*x-coordinates of wing*)
                 j:=j+1;
                 xypts[j]:=duck[j+102]*mag+y;      (*y-coordinates of wing*)
                 j:=j+1;
            end;
    polyline(10,0,xypts);                          (*draw wing*)
    j:=duck[123]*mag+x;
    k:=duck[124]*mag+y;
    n:=duck[125];
    circle(j,k,n,0);                               (*draw eye*)
end;



(**************************************************************************
)
(*This procedure reads the display area and writes it back into a second
area*)
(*of the display memory starting from x=0 and y=474 .
*)
(**************************************************************************
)
procedure draw_area2;
var
  i : integer;
begin
    len:=474;                    (*display area's length in lines*)
    wid:=40;                     (*display area's width in words*)
    addr:=0;                     (*start address in system memory*)
    for i:=1 to 3 do
            begin
                 plane:=i-1;     (*set the colour plane*)
                 xad:=0;         (*xad,yad:first area's starting point*)
                 yad:=474;
                 xycv;           (*convert xad,yad to word and dot
addresses*)
                 rdat;           (*read and store the first area*)
                 xad:=0;         (*xad,yad:second area's starting point*)
                 yad:=948;
                 xycv;           (*convert xad,yad to word and dot
addresses*)
                 display;        (*draw the second area*)
            end;
end;



(**************************************************************************)
(*This procedure reads a window from the display memory out to the host     *)
(*system memory.                                                            *)
(*xw,yw  =  window start point on the display memory in actual device units *)
```

```
(*Vlines =  number of lines to be read in vertical direction            *)
(*Hwords =  number of words to be read in horizontal direction          *)
(***********************************************************************)
procedure read_window(xw,yw,Vlines,Hwords:integer);
var
   i  :  integer;
begin
    len:=Vlines;
    wid:=Hwords;
    count:=2*len*wid;           (*total number of bytes to be transferred*)
                                (*for each plane*)
    addr:=0;                    (*window start address in the system memory*)
    for i:=1 to 3 do
            begin
                plane:=i-1;     (*set the colour plane*)
                xad:=xw;        (*xad,yad:window start point in the*)
                yad:=yw;        (*display memory*)
                xycv;           (*convert xad,yad to screen memory addresses*)
                rdat;           (*read window and store it*)
                addr:=addr+count;       (*adjust addr for the next colour*)
                                        (*plane*)
            end;
end;



(***********************************************************************)
(*This procedure reads a block of data from the host system memory out to   *)
(*the display memory .                                                 *)
(* addr   = window start address in the system memory                  *)
(* xw,yw  = window start point on the display memory in actual device units *)
(* Vlines = number of window lines in vertical direction               *)
(* Hwords = number of window words in horizontal direction             *)
(***********************************************************************)
procedure draw_window(xw,yw,Vlines,Hwords:integer);
var
   i  :  integer;
begin
    len:=Vlines;
    wid:=Hwords;
    addr:=0;
    count:=2*len*wid;           (*total number of bytes to be transferred*)
                                (*for each plane*)
    for i:=1 to 3 do
            begin
                plane:=i-1;     (*set colour plane address*)
                xad:=xw;
                yad:=yw;
                xycv;           (*convert xad,yad to display memory
addresses*)
                display;        (*draw the window*)
                addr:=addr+count;       (*adjust addr for the next colour*)
```

```
                                        (*plane*)
            end;
end;


(***************************************************************************)
(*This procedure clears a window on the display memory                     *)
(*xw,yw  = window start point in actual device units                       *)
(*Vlines = lines vertically                                                *)
(*Hwords = words horizontally                                              *)
(***************************************************************************)
procedure clear_window(xw,yw,Vlines,Hwords:integer);
var
   i  :  integer;
begin
    len:=Vlines;
    wid:=16*Hwords;
    for i:=1 to 3 do
            begin
                plane:=i-1;      (*set colour plane address*)
                xad:=xw;
                yad:=yw;
                xycv;            (*convert xad,yad to display memory address*)
                gchw;            (*clear the window*)
            end;
end;


(***************************************************************************)
(*This procedure moves a window                                            *)
(*  xd  = pixel displacement in x direction in one step                    *)
(*  yd  = line displacement in y direction in one step                     *)
(*steps = number of steps                                                  *)
(***************************************************************************)
procedure move_duck(xd,yd,steps:integer);
var
  j  :  integer;
begin
    for j:=1 to steps do
         begin
             clear_window(x2,y2,50,7);      (*clear duck in second area*)
             x2:=x1+xd;                     (*adjust x2,y2 for the second
area*)
             y2:=y1+474+yd;
             draw_window(x2,y2,50,7);       (*draw duck in the second area*)
             sad:=474*64;                   (*second area's starting address*)
             scrol;                         (*display second area on the
screen*)
             clear_window(x1,y1,50,7);      (*clear duck in first area*)
             x1:=x2+xd;                     (*adjust x1,y1 for the first area*)
             y1:=y2-474+yd;
```

```
            draw_window(x1,y1,50,7);       (*draw duck in the first area*)
            sad:=0;                        (*first area's start address*)
            scrol;                         (*display first area on the
screen*)
        end;
end;



(********************** MAIN PROGRAM ************************************)
    begin
        open_wk(1);
        clear_it;
        readdata;
        background;
        draw_area2;
        menu;
        repeat
            draw_duck(0,21000,19,7,1);
            (* x1:=(0/32767)*474 = 0  *)
            (* y1:=474-(21000/32767)*474 = 170  *)
            x1:=0;                     (*start point of the window of the
duck*)
            y1:=170;                   (*in first area,in actual device units*)
            x2:=x1;                    (*adjust x2,y2 for the second area*)
            y2:=y1+474;
            read_window(x1,y1,50,7);   (*read a window of duck*)
            move_duck(16,0,8);         (*move the duck to the right*)
            move_duck(0,3,30);         (*move the duck down*)
            move_duck(16,0,8);         (*move the duck to the right*)
            read(ch);
            if ch=chr(27) then
                begin
                    stop:=false;
                    clear_window(x1,y1,50,7);
                    clear_window(x2,y2,50,7);
                end
            else
                stop:=true;
        until stop;
        clear_it;
        exit_gsx;
    end.
```

```
;***************************************************************************
;The assembly language module of the Pascal program FIFORW.PAS.
;***************************************************************************
        public      GSX
        public      rdat,display,gchw
        public      xycv,grzoom,grmask
        public      gdcc,gdccl
        public      pat,grfigs,grwrite
        public      scrol,delay
        public      grmout,curw
        name        pasgsx
        assume      cs:code , ds:data
data    segment     public
        extrn       xad : word , yad : word
        extrn       len : word , wid : word
        extrn       plane : byte
        extrn       addr : word
        extrn       sad : word
data    ends
code    segment     public
GDOS    EQU         0e0h


;****************** GSX_86 standard interface procedure *******************
GSX     proc    near
        push    ds
        push    es
        mov     ax,ss
        mov     ds,ax
        mov     dx,sp
        add     dx,6
        mov     cx,473h         ;GSX_86 function code
        int     GDOS
        pop     es
        pop     ds
        ret     20
GSX     endp


;***************************************************************************
;This procedure converts (xad,yad) coordinates on the CRT screen to word and
;dot addresses,ead and dad,in the display memory.
;***************************************************************************
xycv    proc    near
        mov     cx,xad
        and     cx,000fh        ;separate the first four bits of xad
        push    cx              ;save it
        mov     cl,4            ;get a shift factor
        mov     ax,xad
        sar     ax,cl          ;shift four bits to the right to form Intg(xad/16)
```

```
        push    ax              ;save it
        mov     ax,40h          ;load ax with pitch
        mov     cx,yad
        mul     cx              ;multiply yad by pitch
        pop     cx              ;restore Intg(xad/16)
        add     ax,cx           ;add yad*pitch to Intg(xad/16)
        mov     ead,ax          ;drop it into ead
        pop     ax              ;restore the first four bits of xad
        mov     cl,4            ;get a shift factor
        shl     ax,cl           ;shift four bits to the left
        mov     dad,al          ;save it
        ret
xycv    endp


;****************************************************************************
;In this procedure the host microprocessor reads a block of the display memory
;data through the FIFO bufer.The width and length of the block are:
;wid = number of words horizontally
;len = number of lines vertically
;****************************************************************************
rdat    proc    near
        mov     di,addr          ;get the offset address
        push    es               ;save es register
        mov     ax,7000h         ;get the address of a free block of the host
                                 ;memory
        mov     es,ax            ;load it into es
        mov     cx,wid           ;get the width
rdat01:
        push    cx               ;save the first loop counter
        call    curw             ;put the cursor in the proper plane
        mov     maskl,0ffh       ;set the low byte of the mask
        mov     maskh,0ffh       ;set the high byte of the mask
        call    grmask           ;set the mask value
        mov     p1,04h           ;set type and direction
        mov     ax,len           ;get the length of the block
        mov     p2,ax            ;put it into second parameter of FIGS command
        mov     cx,3             ;three bytes of data to be sent
        call    grfigs           ;send the parameters
        mov     al,0a0h          ;RDAT CMD
        out     72h,al           ;send it out
        mov     cx,2             ;get a multiplication factor
        mov     ax,len           ;load ax with block length
        mul     cx               ;2*length
        mov     cx,ax            ;load it into cx
rdatloop:
        call    gdccl
        in      al,72h                  ;read display memory's data
        mov     es:byte ptr[di],al      ;write it into the system memory
        inc     di                      ;point to the next byte
        loop    rdatloop
        mov     al,6bh                  ;dummy command
```

```
        out     72h,al
        pop     cx                      ;restore the first loop counter
        add     ead,1                   ;point to the next display memory word
        loop    rdat01
        pop     es                      ;restore the es register
        ret
rdat    endp



;***************************************************************************
;This procedure reads data from the system memory out into the display memory,
;through the FIFO buffer.
;***************************************************************************
display proc    near
        mov     di,addr         ;get the start of data in the system memory
        push    es              ;save es register
        mov     ax,7000h        ;get the segment address
        mov     es,ax           ;load it into es register
        mov     cx,wid          ;get the width of the block
disp01:
        push    cx              ;save first loop count
        mov     p1,04h          ;select type and direction of the figure
        mov     cx,1            ;one byte to be sent
        call    grfigs          ;send p1
        call    curw            ;put cursor in the proper plane
        mov     cx,len          ;get the length of the block
disploop:
        push    cx                      ;save the second loop count
        mov     al,es:byte ptr[di]      ;get byte from the system memory
        mov     maskl,al                ;load it into the low mask byte
        inc     di                      ;point to the next byte
        mov     al,es:byte ptr[di]      ;get byte from the system memory
        mov     maskh,al                ;load it into the high mask byte
        inc     di                      ;point to the next byte
        call    grmask                  ;set the mask value
        mov     wattrib,23h             ;set writing attribute
        mov     wpara,0ffffh            ;set writing parameter to all ones
        mov     cx,2                    ;two parameter bytes to be sent
        call    grwrite                 ;set writing mode and parameters
        pop     cx                      ;recover second loop counter
        loop    disploop
        pop     cx                      ;recover first loop count
        add     ead,1                   ;point to the next display memory word
        loop    disp01
        pop     es                      ;restore es register
        ret
display endp
```

```
;*************************************************************************
;This procedure tests FIFO Empty bit of the GDC's status register.
;*************************************************************************
gdcc    proc    near
aa:     in      al,70h          ;read status register
        test    al,04h          ;test FIFO Empty bit
        jz      aa              ;jump if FIFO is not empty
        ret                     ;return if FIFO is empty
gdcc    endp


;***************************************************************************
;This procedure tests the DATA READY bit of the GDC's status register.
;***************************************************************************
gdcc1   proc    near
gdcc10:
        in      al,70h          ;read status register
        test     al,01h         ;test DATA READY bit
        jz       gdcc10         ;jump if data is not ready
        ret                     ;return if data is ready
gdcc1   endp



;***************************************************************************
;Position the cursor
;***************************************************************************
curw    proc    near
        mov     bx,(offset dacurw)      ;get the address
        mov     cx,2                    ;two parameter bytes to be sent
        call    grmout                  ;send them out
        mov     al,dad                  ;get dot address
        or      al,plane                ;set plane address
        out     70h,al                  ;send it out
        ret
curw    endp



;***************************************************************************
;Write mask value
;***************************************************************************
grmask  proc    near
        mov     bx,(offset damask)      ;get the address
        mov     cx,2                    ;two parameter byte to be sent
        call    grmout                  ;send them out
        ret
grmask  endp
```

```
;*****************************************************************************
;Send FIGS command and its parameters
;*****************************************************************************
grfigs  proc   near
        mov    bx,(offset dafigs)      ;get the address
        call   grmout                  ;send command and its parameters out
        ret
grfigs  endp



;*****************************************************************************
;set zoom factor
;*****************************************************************************
grzoom  proc   near
        mov    bx,(offset dazoom)      ;get the address
        mov    cx,1                    ;one parameter byte to be sent
        call   grmout                  ;send them out
        ret
grzoom  endp



;*****************************************************************************
;Set Writing mode and parameters
;*****************************************************************************
grwrite proc   near
        mov    bx,(offset dawrite)     ;get the address
        call   grmout                  ;send staff out
        ret
grwrite endp



;*****************************************************************************
;This procedure fills a rectangular area of the display memory.The start point
;of the rectangle is specified by the CURS command,the direction and size by
;the FIGS command,and the pattern is loaded into the PRAM locations 8-15.
;*****************************************************************************
gchw    proc   near
        call   grzoom          ;set zoom
        mov    maskl,0ffh      ;low byte of the mask
        mov    maskh,0ffh      ;high byte of the mask
        call   grmask          ;write mask
        mov    wattrib,22h     ;get writing mode
        mov    cx,0            ;no parameter follows the WDAT command
        call   grwrite         ;set writing mode
        call   curw            ;put cursor in the proper plane
        call   pat             ;set pattern
        mov    p1,12h          ;get type and direction
        mov    ax,len          ;get length of the rectangle
```

```
        mov     p2,ax               ;put it into proper parameter
        mov     ax,wid              ;get width of the rectangle
        mov     p4,ax               ;put it into the proper parameter
        mov     cx,5                ;five parameter bytes to be sent
        call    grfigs              ;send FIGS command and its parameters
        mov     al,68h              ;GCHRD command
        out     72h,al              ;start area filling
        ret
gchw    endp


;***************************************************************************
;Set the pattern for area filling
;***************************************************************************
pat     proc    near
        mov     bx,(offset dapat)       ;get the address
        mov     cx,8                    ;8 parameters to be sent
        call    grmout                  ;send them out
        ret
pat     endp


;****************************************************************************
;Upon entry to this procedure bx contains the offset address of the start of
;command and parameter bytes,and cx contains the total number of parameters
;following the command byte.
;****************************************************************************

grmout  proc    near
        call    gdcc                    ;GDS's status check
        mov     al,cs:byte ptr[bx]      ;get the command byte
        out     72h,al                  ;send it out
        cmp     cx,0                    ;check number of parameters
        jz      grmoutret               ;if no parameter then return
grmout0:
        inc     bx                      ;point to the next byte
        mov     al,cs:byte ptr[bx]      ;get the byte
        out     70h,al                  ;send it out
        loop    grmout0
grmoutret:
        ret
grmout  endp


;***************************************************************************
;This procedure scrolls the display window by changing its starting address
;***************************************************************************
scrol   proc    near
```

```
        mov     ax,sad                  ;get the start address
        mov     grsad,ax                ;put it into the data structure
        mov     cx,2                    ;two parameter bytes to be sent
        mov     bx,(offset dascrol)     ;get the address
        call    grmout                  ;send them out
        ret
scrol   endp


;*****************************************************************************
delay   proc    near
        mov     bx,00fffh
del00:
        dec     bx
        jnz     del00
        ret
delay   endp


;*****************************************************************************

dafigs:
        db      4ch                     ;FIGS command
p1      db      00h                     ;Type + DIR
p2      dw      00h                     ;DC
p4      dw      00h                     ;D


dapat:
        db      78h                     ;PRAM opcode + PRAM start address of 8
        dw      0ffffh                  ;locations 8 and 9
        dw      0ffffh                  ;locations 10 and 11
        dw      0ffffh                  ;locations 12 and 13
        dw      0ffffh                  ;locations 14 and 15


damask:
        db      4ah                     ;MASK command
maskl   db      00h                     ;mask low byte
maskh   db      00h                     ;mask high byte


dacurw:
        db      49h                     ;CURS command
ead     dw      0000h                   ;word address
dad     db      00h                     ;dot address
```

```
dazoom:
        db      46h                     ;ZOOM command
zoomf   db      00h                     ;zoom factor


dascrol:
        db      70h                     ;PRAM command + PRAM start address of 0
grsad   dw      0000h                   ;starting address of the display area


dawrite:
wattrib   db    00h                     ;writing attribute
wpara     dw    0000h                   ;writing parameter
code    ends
        end
```

Appendix B6
Demonstration for Pixel Transfers

```
(****************************************************************************)
(*This program demonstrates picture transmission by sending pixels of the   *)
(*display memory.                                                           *)
(*The data is sent using packetized acknowledge based protocol.             *)
(*This program has an assembly language module named TXASM.I86.            *)
(****************************************************************************)
(* a:                                                                       *)
(* mt+86 b:pixeltx                                                          *)
(* asmt86 b:txasm                                                           *)
(* linkmt b:pixeltx,b:txasm,fpreals,trancend,paslib/s                       *)
(* graphics                                                                 *)
(* b:                                                                       *)
(* pixeltx                                                                  *)
(****************************************************************************)
program transmit;
const
     OPEN_CMD          = 1 ;
     CLOSE_CMD         = 2 ;
     CLEAR_CMD         = 3 ;

     PLINE_CMD         = 6 ;
     TEXT_CMD          = 8 ;
     FILAREA_CMD       = 9 ;
     GDP_CMD           = 11 ;

     TEXT_HGT_CMD      = 12 ;

     LINE_STYL_CMD     = 15 ;
     FILL_STYL_CMD     = 23 ;
     FILL_INDX_CMD     = 24 ;

     MAX_CNTL_VALS     = 10 ;
     MAX_INTIN_VALS    = 80 ;
     MAX_INTOUT_VALS   = 45 ;
     MAX_PTS_VALS      = 100 ;

     SCREENSIZE        = 32767;

type
     cntrl_array  = array  [ 1..MAX_CNTL_VALS ] of integer ;
     intin_array  = array  [ 1..MAX_INTIN_VALS ] of integer ;
     intout_array = array  [ 1..MAX_INTOUT_VALS ] of integer ;
     ptsin_array  = array  [ 1..MAX_PTS_VALS ] of integer;
     ptsout_array = array  [ 1..MAX_PTS_VALS ] of integer ;
     data_array = array  [ 1..20 ] of integer;


    var
     contrl    :    cntrl_array ;
```

```
     intin      :    intin_array ;
     intout     :    intout_arrray ;
     ptsin      :    ptsin_array ;
     ptsout     :    ptsout_array ;
     cur_dev    :    integer ;
     px ,py     :    data_array;
     f          :    text;
     rx         :    boolean;    (*receiver ready flag*)
     addr       :    integer;    (*address in system memory*)
     psad       :    integer;    (*packets start address in the display memory*)
     pnum       :    integer;    (*packet number*)
     retx       :    boolean;    (*retransmission*)
     plane      :    integer;    (*colour plane address*)
     psize      :    integer;    (*total number of bytes in each packet*)
     pcount     :    integer;    (*total number of packets*)
     ch         :    char;       (*dummy variable*)
     i,j,k,m    :    integer;    (*dummy variables*)


external procedure GSX( var ptsout :   ptsout_array ;
                        var intout :   intout_array ;
                        var ptsin  :   ptsin_array ;
                        var intin  :   intin_array ;
                        var contrl :   cntrl_array ) ;
external procedure display;
external procedure datout;
external procedure rdat;
external procedure acknack;




(***************************** Open Workstation *****************************)
procedure open_wk( dev_no : integer );
var
     i : integer ;
begin
     contrl[ 1 ] := OPEN_CMD ;
     contrl[ 2 ] := 0         ;
     contrl[ 4 ] := 10 ;
     intin[ 1 ] := dev_no ;
     for i := 2 to 10 do
          intin[ i ] := 1 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;
```

```
(**************************** Clear the Screen ****************************)
procedure clear_it ;
begin
     contrl[ 1 ] := CLEAR_CMD ;
     contrl[ 2 ] := 0 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;



(**************************** Close Workstation ****************************)
procedure exit_gsx ;
begin
     contrl[ 1 ] := CLOSE_CMD ;
     contrl[ 2 ] := 0 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;



(****************************************************************************)
(*This procedure sets different attributes.                                *)
(****************************************************************************)
procedure set_attrib( cmd, attribute : integer ) ;
begin
     contrl[1]:= cmd;
     contrl[2]:= 0;
     intin[ 1 ]  := attribute ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end;



(****************************************************************************)
(*This procedure draws a bar.                                              *)
(****************************************************************************)
procedure draw_bar(left,bottom,right,up,pat,col:integer);
begin
     set_attrib(25,col);
     set_attrib(23,2);
     set_attrib(24,pat);
     contrl[1]:=11;
     contrl[2]:=2;
     contrl[6]:=1;
     ptsin[1]:=left;
     ptsin[2]:=bottom;
     ptsin[3]:=right;
     ptsin[4]:=up;
```

```
     GSX(ptsout,intout,ptsin,intin,contrl);
end;




(*****************************************************************************)
(*This procedure draws a pattern on the screen.                               *)
(*****************************************************************************)
procedure polyline;
var
     i,j,col:integer;
begin
     read_points;
     for i:=1 to 16 do
            begin
                for j:=1 to 16 do
                       begin
                           contrl[1]:=6;
                           contrl[2]:=2;
                           ptsin[1]:=px[i];
                           ptsin[2]:=py[i];
                           ptsin[3]:=px[j];
                           ptsin[4]:=py[j];
                           GSX(ptsout,intout,ptsin,intin,contrl);
                       end;
                col:=col+1;
                if col>7 then col:=1;
                set_attrib(17,col);
            end;
end;




(*****************************************************************************)
(*This procedure reads data from the file XYPTS.TST.                          *)
(*****************************************************************************)
procedure read_points;
var
     k : integer;
begin
     assign(f,'xypts.tst');
     reset(f);
     if ioresult=255 then
             writeln('error opening')
     else
             for k:=1 to 16 do
                     begin
                         read(f,m);
                         px[k]:=m;
```

```
                        read(f,m);
                        py[k]:=m;
                    end;
end;
```

```
(***************************************************************************)
(*This procedure draws different patterns on the screen                   *)
(***************************************************************************)
procedure draw_pat;
begin
     draw_bar(0,0,10000,15000,3,1);
     draw_bar(0,0,10000,15000,4,1);
     draw_bar(11000,0,21000,15000,1,2);
     draw_bar(11000,0,21000,15000,4,2);
     draw_bar(22000,0,32000,15000,1,3);
     draw_bar(22000,0,32000,15000,2,3);
     draw_bar(0,16000,10000,31000,1,5);
     draw_bar(0,16000,10000,31000,2,5);
     draw_bar(11000,16000,21000,31000,3,6);
     draw_bar(11000,16000,21000,31000,4,6);
     draw_bar(22000,16000,32000,31000,2,7);
     draw_bar(22000,16000,32000,31000,4,7);
end;
```

```
(***************************************************************************)
(*This procedure checks for the receiver.If receiver is ready it calls a   *)
(*procedure to send data,if not it writes a message on the screen.         *)
(***************************************************************************)
procedure transmit;
begin
    acknack;                (*waite for receiver ack*)
    if rx=true then         (*if receiver sends ack*)
           send_data        (*go and send packets of data*)
    else                    (*if receiver dosen't send ack*)
           writeln('Receiver is not ready');
end;
```

```
(*****************************************************************************)
(*This procedure transmits the display area packet by packet.A packet can    *)
(*be retransmitted if the receiver requires.If the receiver is not ready     *)
(*a message is written on the screen and the transmission is stoped.         *)
(*****************************************************************************)
procedure send_data;
var
    i,j : integer;
begin
    plane:=0;                       (*set the first plane address*)
    repeat
         psad:=474*64;              (*set packets sad in the display memory*)
         psize:=474*2;              (*set packet size*)
         pcount:=40;                (*set number of packets*)
         addr:=$7000;               (*set offset address in the system memory*)
         rdat;                      (*read screen*)
         pnum:=1;                   (*first packet number*)
         repeat
             writeln(pnum:9);
             datout;                               (*send a packet*)
             acknack;                              (*get the receiver response*)
             if not retx and rx then               (*if no error*)
                   begin
                       addr:=addr+948;             (*point to the next packet*)
                        pnum:=pnum+1;              (*get next packet number*)
                   end;
             if retx and rx then                   (*if error detected*)
                    writeln('Retransmission',pnum:4);
             if not rx then                        (*if receiver is not ready*)
                   begin
                        writeln('Receiver is not ready');
                        pnum:=pcount+1;            (*set these values to go*)
                        plane:=2;                  (*out of the loops*)
                   end;
         until pnum=pcount+1;                      (*repeat until all the
packets*)
                                                   (*are sent*)
         plane:=plane+1;                           (*set the next plane address*)
    until plane=3;                                 (*three times for three
planes*)
end;                                               (*end of procedure*)




(**************************** Main Program ******************************)
begin
     open_wk(1);
     clear_it;
    draw_pat;
    transmit;
```

```
    clear_it;
     if rx then
        begin
           polyline;
           transmit;
         end;
     read(ch);
     clear_it;
     exit_gsx;
end.
```

```
;***************************************************************************
;The assembly language module of the  Pascal program PIXELTX.PAS
;***************************************************************************
      public    GSX
      public    curw,gdcc,gdccl
      public    acknack,datout
      public    rdat
      public    grmask,grfigs
      public    stest,rssn00
      public    rtest,rsrv00

      name      pasgsx
      assume    cs:code, ds:data
data  segment   public
sum   dw        ?
      extrn     rx:byte
      extrn     addr:word
      extrn     psad:word
      extrn     pnum:byte
      extrn     retx:byte
      extrn     plane:byte
      extrn     pcount:word,psize:word
data  ends
code segment    public
GDOS equ        0e0h
;

;********************** GSX_86 interface procedure ************************
GSX     proc       near
        push    ds
        push    es
;
        mov     ax,ss
        mov     ds,ax
        mov     dx,sp
        add     dx,6
        mov     cx,473h
```

```
        int     GDOS
;
        pop     es
        pop     ds
        ret     20
GSX     endp
;
```

```
;**************************************************************************
;This procedure returns two flags  rx and retx
;rx=0    Time out and receiver is not ready
;rx=1    Receiver is ready
;retx=0  Receiver sent ack
;retx=1  Receiver sent nack
;**************************************************************************
acknack  proc   near
        push    cx          ;register save
        push    bx
        mov     rx,1        ;set rx flag
        mov     bx,02ffh   ;get a time out loop counter
rec00:
        mov     cx,0        ;init time out counter of 8251
rec10:
        in      al,32h      ;read 8251 status register
        test    al,02h      ;check rxrdy bit
        jnz     rec20       ;jump if a byte is ready to receive
        loop    rec10
        jmp     rec40       ;jump if time is out
rec20:
        call    rtest       ;go get the byte
        cmp     al,0aah     ;is it low byte of ack?
        jz      rec30       ;yes then jump
        cmp     al,0ffh     ;is it low byte of nack?
        jz      rec60       ;yes then jump
        jmp     rec40       ;jump to continue the loop
rec30:
        call    rtest       ;get the next byte
        cmp     al,55h      ;is it high byte of ack?
        jz      ack00       ;yes then jump
        jmp     rec40       ;jump to continue the loop
rec60:
        call    rtest       ;go get the next byte
        cmp     al,00       ;is it high byte of nack?
        jz      rec70       ;yes then jump
rec40:
        dec     bx          ;decrement loop counter
        jnz     rec00       ;jump if loop counter is not zero
        mov     rx,0        ;time is out and receiver is not ready
        jmp     rec50       ;jump to return
rec70:
```

```
        mov     retx,1          ;nack is received,retransmission is asked
        jmp     rec50           ;jump to return
ack00:
        mov     retx,0          ;ack is received
rec50:
        pop     bx              ;restore registers
        pop     cx
        ret
acknack  endp
```

```
;***************************************************************************
;This procedure sends packet number,the packet and sum of all bytes in the
;packet.
;****************************************************************************

datout  proc    near
        mov     sum,0           ;initialize the sum
        push    es              ;register save
        push    di
        mov     al,pnum         ;get the packet number
        call    stest           ;send it
        mov     ax,7000h        ;get the segment address
        mov     es,ax           ;put it into es
        mov     ax,addr         ;get the offset address
        mov     di,ax           ;put it into di
        mov     bx,psize        ;get the packet size
send20:
        mov     al,es:byte ptr[di]      ;get the byte from system memory
        mov     ah,0                    ;make high byte of ax zero
        add     sum,ax                  ;add data together
        call    stest                   ;send data byte
        inc     di                      ;point to the next byte
        dec     bx                      ;decrement loop counter
        jnz     send20                  ;repeat if loop counter is not zero
        mov     ax,sum                  ;get the sum
        call    stest                   ;send low byte first
        mov     al,ah                   ;then high byte
        call    stest
datoutret:
        pop     di                      ;restore registers
        pop     es
        ret
datout  endp
```

```
;************************* Cursor Positioning ****************************
curw    proc   near
        call   gdcc
        mov    al,49h       ;CURS command
        out    72h,al
        mov    ax,psad      ;word address
        out    70h,al
        mov    al,ah
        out    70h,al
        mov    al,plane     ;colour plane address
        out    70h,al
        ret
curw    endp



;**************************** Mask Write *********************************
grmask  proc   near
        call   gdcc
        mov    al,4ah       ;MASK command
        out    72h,al
        mov    ax,0ffffh    ;mask parameter of all ones
        out    70h,al
        mov    al,ah
        out    70h,al
        ret
grmask  endp



;*****************************************************************************
;Send FIGS command and its parameters
;*****************************************************************************
grfigs  proc   near
        call   gdcc         ;GDC status check
        mov    al,4ch       ;FIGS command
        out    72h,al
        mov    al,04h       ;type and direction
        out    70h,al
        mov    ax,psize     ;number of bytes in the packet
        shr    ax           ;divide by two to get DC
        out    70h,al       ;send DC low byte
        mov    al,ah        ;send DC high byte
        out    70h,al
        ret
grfigs  endp
```

```
;*************************************************************************
;This procedure tests FIFO empty bit of the GDC's status register.
;*************************************************************************
gdcc    proc    near
gdc00:
        in      al,70h          ;read status register
        test    al,04h          ;test FIFO empty bit
        jz      gdc00           ;jump if FIFO is not empty
        ret                     ;return if FIFO is empty
gdcc    endp
```

```
;*************************************************************************
;This procedure tests DATA READY bit of the GDC's status register.
;*************************************************************************
gdccl   proc    near
gdc10:
        in      al,70h          ;read status register
        test    al,01h          ;test DATA READY bit
        jz      gdc10           ;jump if data is not ready
        ret                     ;return if data is ready
gdc10   endp
```

```
;*************************************************************************
;This procedure reads the display area packet by packet.Each packet consists
;of a strip of pixels on the screen.The width of each strip is one word and
;the length is psize/2 words.
;pcount = total number of packets
;psize = total number of bytes in each packet
;*************************************************************************
rdat   proc  near
        push    es              ;save segment register
        mov     ax,7000h        ;get segment address
        mov     es,ax           ;put it into segment register
        mov     bx,addr         ;get offset address
        mov     cx,pcount       ;first loop counter is number of packets
rdat01:
        push    cx              ;save the first loop counter
        call    curw            ;put the cursor in the proper plane
        call    grmask          ;write mask value
        call    grfigs          ;send FIGS command and parameters
        mov     al,0a0h         ;RDAT command
        out     72h,al          ;send it out
        mov     cx,psize        ;second loop counter is number of packet bytes
rdatloop:
        call    gdccl           ;test if data is ready
        in      al,72h          ;get the byte
        mov     es:byte ptr[bx],al     ;store it in the system memory
```

```
        inc     bx              ;point to the next location
        loop    rdatloop        ;repeat until all bytes are read
        mov     al,6bh          ;get a dummy command
        out     72h,al          ;send it out
        pop     cx              ;recover the first loop counter
        add     psad,1          ;point to the next display memory word
        loop    rdat01          ;repeat until all packets are read
        pop     es              ;restore the segment register
        ret
rdat    endp



;*****************************************************************************
;This procedure receives a data byte.
;*****************************************************************************
rtest   proc    near
        push    dx
        push    bx
        mov     bx,0003H  ;get a loop counter
reader00:
        call    rsrv00    ;RS232 routine
        mov     dl,al     ;save the received byte
        test    ah,80h    ;test time out bit
        jz      reader10  ;jump if data is received
        dec     bx
        jnz     reader00
reader10:
        mov     al,dl     ;load al with the received byte
        pop     bx
        pop     dx
        ret
rtest   endp



;*****************************************************************************
;This procedure programs the serial I/O communication controller,8251A,to
;receive data from an RS232 modem.It is assumed that the chip has been
;initialized when the computer has been powered up.
;*****************************************************************************
rsrv00  proc    near
        push    bx
        push    cx
        push    dx
        mov     al,07h   ;modem ER on
        out     32h,al   ;command register
        sub     cx,cx    ;initialize time out counter
rsrv10:
        in      al,32h   ;read status
```

```
        test    al,80h   ;test DR
        jnz     rsrv20   ;jump if DR is on
        loop    rsrv10   ;waite DR
        jmp     rsrv30
rsrv20:
        test    al,02h   ;test receiver ready bit
        jnz     rsrv40
        loop    rsrv10
rsrv30:
        or      al,80h   ;set time out
        mov     ah,al
        sub     al,al
        jmp     rsret
rsrv40:
        and     al,7fh   ;clr time out
        mov     ah,al
        in      al,30h   ;receive character
rsret:
        pop     dx
        pop     cx
        pop     bx
        ret
rsrv00  endp



;******************************************************************************
;This procedure transmits a character
;                    input   al=send character

;******************************************************************************
stest   proc    near
        push    ax
punch00:
        call    rssn00      ;RS232 routin
        test    ah,80h      ;check time out
        jz      punch10     ;if not time out go out of the loop
        jmp     punch00
punch10:
        pop    ax
        ret
stest   endp



;******************************************************************************
;This procedure programs the serial I/O communication controller 8251 to send
;data to an RS232 modem.It is assumed that the chip has been initialized when
;the computer has been powered up.
;
```

```
;                 input  al=send character
;**************************************************************************
rssn00  proc    near
        push    cx
        push    dx
        push    ax
        mov     al,27h          ;modem RS,ER on
        out     32h,al          ;set the command register
        sub     cx,cx           ;init time out counter
rssn10:
        in      al,32h          ;read status
        test    al,80h          ;check DSR bit
        jnz     rssn20          ;jump if DSR is on
        loop    rssn10
        jmp     rssn30          ;jump if time is out
rssn20:
        in      al,34h          ;read signal
        test    al,04h          ;test CS bit
        jnz     rssn25          ;jump if CS is on
        loop    rssn20          ;continue the loop
        jmp     rssn30          ;jump if time is out
rssn25:
        in      al,32h          ;read status register
        test    al,05h          ;if TXE and TXRDY are on
        jnz     rssn40          ;jump
        loop    rssn10
rssn30:
        or      al,80h          ;set time out
        mov     dl,al           ;status save
        pop     ax              ;recover ax
        jmp     rssn50          ;jump
rssn40:
        and     al,7fh          ;clr time out
        mov     dl,al           ;status restore
        pop     ax              ;recover the send character
        out     30h,al          ;send it out
rssn50:
        mov     ah,dl           ;status restore
        pop     dx              ;restore registers
        pop     cx
        ret
rssn00  endp
code  ends
        end
```

```
(*************************************************************************)
(*This program receives packets of data containing the display memory pixels *)
(*and writes them into the display memory.The packet format, total number of *)
(*packets to be received and the start addresses of the packets in the      *)
(*dsplay memory must be the same as in the transmitter program.              *)
(*This program has an assembly language module named RXASM.I86              *)
(*************************************************************************)
(* a:                                                                    *)
(* mt+86 b:pixelrx                                                       *)
(* asmt86 b:rxasm                                                        *)
(* linkmt b:pixelrx,b:rxasm,fpreals,trancend,paslib/s                    *)
(* graphics                                                              *)
(* b:                                                                    *)
(* pixelrx                                                               *)
(*************************************************************************)
program receiver;
const
     OPEN_CMD             = 1 ;
     CLOSE_CMD            = 2 ;
     CLEAR_CMD            = 3 ;

     PLINE_CMD            = 6 ;
     TEXT_CMD             = 8 ;
     FILAREA_CMD          = 9 ;
     GDP_CMD              = 11 ;

     TEXT_HGT_CMD         = 12 ;

     LINE_STYL_CMD        = 15 ;
     FILL_STYL_CMD        = 23 ;
     FILL_INDX_CMD        = 24 ;

     MAX_CNTL_VALS        = 10 ;
     MAX_INTIN_VALS       = 80 ;
     MAX_INTOUT_VALS      = 45 ;
     MAX_PTS_VALS         = 100 ;

     SCREENSIZE           = 32767;

type
     cntrl_array  = array  [ 1..MAX_CNTL_VALS ] of integer ;
     intin_array  = array  [ 1..MAX_INTIN_VALS ] of integer ;
     intout_array = array  [ 1..MAX_INTOUT_VALS ] of integer ;
     ptsin_array  = array  [ 1..MAX_PTS_VALS ] of integer;
     ptsout_array = array  [ 1..MAX_PTS_VALS ] of integer ;
    var
     contrl     :    cntrl_array ;
     intin      :    intin_array ;
     intout     :    intout_arrray ;
     ptsin      :    ptsin_array ;
     ptsout     :    ptsout_array ;
     cur_dev    :    integer ;
     ch         :    char ;
```

```
     i,k,m,j    :    integer;
     tx         :    boolean;        (*transmitter ready flag*)
     psad       :    integer;        (*packets sad in the display memory*)
     flag       :    integer;        (*error flag*)
     txpnum     :    integer;        (*packet number sent by the transmitter*)
     rxpnum     :    integer;        (*packet number expected by the receiver*)
     plane      :    integer;        (*colour plane address*)
     stop       :    boolean;
     psize      :    integer;        (*number of bytes in a packet*)
     plength    :    integer;        (*number of words in a packet*)
     pcount     :    integer;        (*total number of packets*)

external procedure GSX( var ptsout :   ptsout_array ;
                        var intout :   intout_array ;
                        var ptsin  :   ptsin_array ;
                        var intin  :   intin_array ;
                        var contrl :   cntrl_array ) ;
external procedure datin;
external procedure display;
external procedure handsk;
external procedure acknack;


(********************** Open Workstation *********************************)
procedure open_wk( dev_no : integer );
var
     i : integer ;
begin
     contrl[ 1 ] := OPEN_CMD ;
     contrl[ 2 ] := 0          ;
     contrl[ 4 ] := 10 ;
     intin[ 1 ] := dev_no ;
     for i := 2 to 10 do
          intin[ i ] := 1 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;



(********************** Clear Workstation *********************************)
procedure clear_it ;
begin
     contrl[ 1 ] := CLEAR_CMD ;
     contrl[ 2 ] := 0 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;
```

```
(********************** Close Workstation ********************************)
procedure exit_gsx ;
begin
     contrl[ 1 ] := CLOSE_CMD ;
     contrl[ 2 ] := 0 ;
     GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;




(*************************************************************************)
(*This procedure receives packets containing pixel information and displays *)
(*them on the screen.                                                     *)
(*************************************************************************)
procedure receive;
var
i,j:integer;
begin
    psize:=474*2;              (*total number of bytes in a packet*)
    plength:=474;              (*length of a packet*)
    pcount:=40;                (*total number of packets*)
    plane:=0;
    repeat
       psad:=474*64;           (*packets sad in the display memory*)
       rxpnum:=0;              (*initialize the packet number*)
       flag:=1;                (*assume no errors*)
       repeat
           rxpnum:=rxpnum+1;   (*get the packet number*)
           handsk;             (*send message to the transmitter*)
           if tx=true then
                 begin
                    datin;                        (*receive a packet*)
                    if flag=0 then                (*if there is an error*)
                        begin
                            writeln('retransmission',rxpnum:3);
                            rxpnum:=rxpnum-1;     (*get the previous pnum*)
                        end
                    else                     (*if the packet is error free*)
                        display;             (*display it on the screen*)
                 end;
           if tx=false then
                 begin
                    writeln('Transmitter is not ready');
                    plane:=2;
                    rxpnum:=pcount;
                 end;
       until rxpnum=pcount;    (*repeat until all packets are received*)
       plane:=plane+1;
   until plane=3;
    acknack;
  end;
```

```
(*********************** Main Program ***********************************)
begin
    open_wk(1);
    clear_it;
    receive;
    clear_it;
    if tx then
         receive;
    read(ch);
    clear_it;
    exit_gsx;
end.
```

```
;***************************************************************************
;The assembly language module of the Pascal program PIXELRX.PAS.
;***************************************************************************
        public    GSX
        public    handsk
        public    acknack
        public    display
        public    gdcc , curw , grfigs
        public   grwrite
        public    putc
        public    datin
        public    rtest
        public    rsrv00
        name      pasgsx
        assume    cs:code, ds:data
data    segment   public
rlc     dw    ?
suml    db    ?
sumh    db    ?
sum     dw    ?
    extrn       tx:byte
    extrn       psad:word
    extrn       flag:byte
    extrn       txpnum:byte
    extrn       rxpnum:byte
    extrn       plane:byte
    extrn       psize:word
    extrn       plength:word
    extrn       pcount:word
data    ends
code segment   public
GDOS equ       0e0h
```

```
;*************************************************************************
;                      GSX_86 standard interface procedure
;*************************************************************************
GSX     proc      near
        push    ds
        push    es
;
        mov     ax,ss
        mov     ds,ax
        mov     dx,sp
        add     dx,6
        mov     cx,473h
        int     GDOS
;
        pop     es
        pop     ds
        ret     20
GSX     endp


;*************************************************************************
;This procedure sends an ACK or NAK to the transmitter and waits to receive
;a packet.The first byte of data which is the packet number is received here.
;The rest of the data is received in the procedure datin.
;If the transmitter doesn't send data during a time out loop, the 'tx' flag
;will go to zero.
;*************************************************************************
handsk   proc   near
        mov     rlc,10          ;set the loop counter for the proc rtest
        mov     cx,000fh        ;time out loop counter
hand20:
        call    acknack         ;send ACK or NAK
        call    rtest           ;waite for data
        test    ah,80h          ;test if data is ready
        jz      hand30          ;if yes jump out of the loop
        loop    hand20          ;go back to the loop
        mov     tx,0            ;time is out then set tx to zero
        jmp     hand40          ;jump to return
hand30:
        mov     tx,1            ;set tx flag
        mov     txpnum,al       ;the first received byte is the packet number
hand40:
        ret
handsk  endp


;*************************************************************************
;This procedure sends an ACK if flag=1 and a NAK if flag=0.
;*************************************************************************
acknack  proc   near
```

```
        cmp     flag,1          ;is flag equal one?
        jz      ack00           ;yes then jump to send ACK
        mov     ax,00ffh        ;get NAK word
        jmp     ack10           ;jump to send NAK
ack00:
        mov     ax,55aah        ;get ACK word
ack10:
        call    putc            ;send low byte first
        mov     al,ah           ;then high byte
        call    putc
        ret
acknack   endp



;*****************************************************************************
;This procedure receives a packet and its checksum.It calculates sum of all
;bytes in the received packet and compares it with the received checksum,if
;they are not equal then the error flag will be reset.The error flag will
;also be reset if one or more data bytes are missing.
;*****************************************************************************
datin   proc    near
        mov     sum,0           ;initialize the sum
        push    es              ;save segment register
        push    di              ;save offset register
        mov     rlc,5           ;wait loop counter for proc rtest
        mov     ax,7000h        ;start address of a free block in system memory
        mov     es,ax           ;load es with the address
        mov     di,ax           ;load di with the offset address
        mov     cx,psize        ;get number of bytes in a packet
dat20:
        call    rtest           ;go to get data byte
        test    ah,80h          ;is data received?
        jnz     error           ;if not then jump to error
        mov     es:byte ptr[di],al      ;store the data
        inc     di              ;point to the next system memory location
        mov     ah,0            ;make high byte of ax zero
        add     sum,ax          ;add received bytes together
        loop    dat20           ;continue until all the data is received
        call    rtest           ;get data byte
        test    ah,80h          ;is data received?
        jnz     error           ;no then jump out of the loop
        mov     suml,al         ;save the received byte
        call    rtest           ;go get data
        test    ah,80h
        jnz     error
        mov     sumh,al         ;save the received byte
        mov     al,suml         ;get checksum low byte
        mov     ah,sumh         ;get checksum high byte
        cmp     ax,sum          ;compare the received sum with the expected one
        jnz     error           ;if not equal then indicate an error
        mov     al,txpnum       ;get the received packet number
        cmp     al,rxpnum       ;comare it with the expected packet number
```

```
        jnz     error           ;if they are not equal then jump to error
        mov     flag,1          ;set the flag to one
        jmp     datinret        ;jump to return
error:
        mov     flag,0          ;make the flag zero
datinret:
        pop     di              ;restore registers
        pop     es
        ret
datin   endp




;****************************************************************************
;This procedure reads data from the system memory and writes it into the
;display memory.
;****************************************************************************
display  proc   near
        push    es              ;register save
        push    di
        mov     ax,7000h        ;get the address of the first byte of data
        mov     es,ax           ;set the segment address
        mov     di,ax           ;set the offset address
        mov     cx,1            ;load cx with the width of the data block
dis10:
        push    cx              ;save the first loop counter
        call    grfigs
        call    curw
        mov     cx,plength      ;get the length of the data block
dis20:
        mov     al,4ah          ;get the MASK command
        out     72h,al          ;send it to the GDC
        mov     al,es:byte ptr[di]      ;get byte of data
        out     70h,al                  ;send it as low byte of the mask
        inc     di                      ;point to the next byte
        mov     al,es:byte ptr[di]      ;get the data byte
        out     70h,al                  ;send it as high byte of the mask
        call    grwrite
        inc     di                      ;point to the next byte
        loop    dis20                   ;
        pop     cx                      ;recover the first loop counter
        add     psad,1                  ;point to the next display memory word
        loop    dis10
        pop     di
        pop     es
display  endp


;************************** position the cursor *****************************
curw    proc    near
        call    gdcc
```

```
        mov     al,49h          ;get the CURS command byte
        out     72h,al          ;send it out
        mov     ax,psad         ;get the packet start address in the display
memory
        out     70h,al          ;send low byte first
        mov     al,ah           ;then high byte
        out     70h,al
        mov     al,plane        ;get the colour plane address
        out     70h,al          ;send it to the GDC
        ret
curw    endp


;******************** Set FIGS command and parameters ***********************
grfigs   proc   near
        call    gdcc
        mov     al,4ch          ;get the FIGS command
        out     72h,al          ;send it out
        mov     al,04h          ;get Type and DIR
        out     70h,al          ;send it out
        ret
grfigs   endp



;****************************************************************************
;Set Write Data command and its parameters.
;****************************************************************************
grwrite    proc   near
        call    gdcc                    ;check GDC's status
        mov     al,20h                  ;get the WDAT command
        out     72h,al                  ;send it to the GDC
        mov     al,0ffh                 ;first writing parameter
        out     70h,al                  ;send it out
        mov     al,0ffh                 ;second writing parameter
        out     70h,al                  ;send it out
        ret
grwrite  endp



;****************************************************************************
;This procedure tests FIFO Empty bit of the GDC status register.
;****************************************************************************
gdcc    proc    near
gdcc10:
        in      al,70h                  ;read status register
        test    al,04h                  ;test FIFO Empty bit
        jz      gdcc10
        ret
gdcc    endp
```

```
;***************************************************************************
;This procedure uses BDOS function 4 to transmit a byte of data via the serial
;I/O communication controller,8251A.
;                          input    al = send data
;***************************************************************************
putc    proc    near
        push    ax                  ;save registers
        push    es
        push    di
        push    si
        push    dx
        push    cx
        push    bx
        mov     dl,al
        mov     cl,4                ;function code
        int     224                 ;BDOS interrupt number
        pop     bx                  ;restore registers
        pop     cx
        pop     dx
        pop     si
        pop     di
        pop     es
        pop     ax
        ret
putc    endp
```

```
;***************************************************************************
;This procedure returns a test value in high byte of ax register to indicate
;whether the data is received or missed.The received data is returned in the
;low byte of the ax register.
;***************************************************************************
rtest   proc    near
        push    dx
        push    bx
        mov     bx,rlc          ;wait loop
reader00:
        call    rsrv00          ;RS232 routine
        mov     dl,al           ;save the received byte
        test    ah,80h          ;test time out bit
        jz      reader10
        dec     bx
        jnz     reader00        ;continue if time is not out
reader10:
        pop     bx
        mov     al,dl           ;put the byte in al
        pop     dx              ;restore dx
        ret
```

```
rtest   endp


;*****************************************************************************
;This procedure programs the serial I/O communication controller,8251A,to
;receive data from an RS232 modem.It is assumed that the chip has been
;initialized when the computer has been powered up.
;*****************************************************************************
rsrv00  proc    near
        push    bx
        push    cx
        push    dx
        mov     al,07h          ;modem ER on
        out     32h,al          ;command register
        sub     cx,cx           ;initialize time out counter
rsrv10:
        in      al,32h          ;read status
        test    al,80h          ;test DR
        jnz     rsrv20          ;if DR is on
        loop    rsrv10          ;waite DR
        jmp     rsrv30          ;jump if time out
rsrv20:
        test    al,02h          ;test receiver ready bit
        jnz     rsrv40
        loop    rsrv10          ;waite receiver
rsrv30:
        or      al,80h          ;set time out
        mov     ah,al
        sub     al,al
        jmp     rsret
rsrv40:
        and     al,7fh          ;clr time out
        mov     ah,al
        in      al,30h          ;receive character
rsret:
        pop     dx
        pop     cx
        pop     bx
        ret
rsrv00  endp
code    ends
        end
```

Appendix B7
Demonstration for Code Transfers

```
(***************************************************************************)
(*This program demonstrates how the graphic pictures are transmitted by   *)
(*sending their GKS information.                                           *)
(*It has an assembly language module named CMDASM.I86.                     *)
(***************************************************************************)
(* a:                                                                      *)
(* mt+86 b:cmdtx                                                           *)
(* asmt86 b:cmdasm                                                         *)
(* linkmt b:cmdtx,b:cmdasm,fpreals,trancend,paslib/s                       *)
(* graphics                                                                *)
(* b:                                                                      *)
(* cmdtx                                                                   *)
(***************************************************************************)
program transmitter;
CONST

     OPEN_CMD            = 1 ;
     CLOSE_CMD           = 2 ;
     CLEAR_CMD           = 3 ;

     PLINE_CMD           = 6 ;
     FILAREA_CMD         = 9 ;
     GDP_CMD             = 11;


     MAX_CNTL_VALS       = 10 ;
     MAX_INTIN_VALS      = 80 ;
     MAX_INTOUT_VALS     = 45 ;
     MAX_PTS_VALS        = 100 ;

     SCREENSIZE          = 32767;

TYPE
     cntrl_array  = array  [ 1..MAX_CNTL_VALS ] of integer ;
     intin_array  = array  [ 1..MAX_INTIN_VALS ] of integer ;
     intout_array = array  [ 1..MAX_INTOUT_VALS ] of integer ;
     ptsin_array  = array  [ 1..MAX_PTS_VALS ] of integer;
     ptsout_array = array  [ 1..MAX_PTS_VALS ] of integer ;
     pdat_array   = array  [ 1..256 ] of integer;
VAR
     contrl     :    cntrl_array ;
     intin      :    intin_array ;
     intout     :    intout_arrray ;
     ptsin      :    ptsin_array ;
     ptsout     :    ptsout_array ;
     pdat       :    pdat_array;
     pnum       :    integer;      (*packet number*)
     plen       :    integer;      (*packet length*)
     rx         :    boolean;      (*receiver ready flag*)
```

```
     retx         :    boolean;      (*retransmission flag*)
     xsad,ysad    :    integer;      (*x,y addresses in the display memory*)
     sead         :    integer;      (*word address in the display memory*)
     x,y,r        :    integer;
     esc          :    char ;
     s,sm         :    string;
     ch           :    char ;
     i,j,k,m      :    integer;
     cindex       :    integer;      (*colour index*)
     fill         :    integer;      (*fill interrior index*)
     vtc          :    integer;      (*number of vertices*)


external procedure GSX( var ptsout :   ptsout_array ;
                        var intout :   intout_array ;
                        var ptsin  :   ptsin_array ;
                        var intin  :   intin_array ;
                        var contrl :   cntrl_array ) ;
external procedure datout(var pdat:ptsout_array);
external procedure grscrol;
external procedure acknack;




(**************************************************************************)
(*This procedure checks if the receiver is ready.                        *)
(**************************************************************************)
procedure handshake;
begin
    acknack;                          (*get the first ACK from the receiver*)
    if rx=false then                  (*if no ACK then receiver is not ready*)
          writeln('Receiver is not ready');
end;




(****************************************************************************)
(*This procedure displays the main menu on the screen,asks for an option and *)
(*initiates the proper graphic operation corresponding to the entered option.*)
(****************************************************************************)
procedure menu;
begin
    if rx=true then                                  (*if receiver is ready*)
         begin
              pos_cur('1','1');                      (*position the cursor at*)
                                                     (*first line and column*)
              sm:=CONCAT(esc,'[','60','C');          (*cursor forward 60
column*)
              w_chr('17');                           (*set red char attribute*)
              writeln(sm,'GRAPH MENU');
              w_chr('21');
```

```
            writeln(sm,'1   polyline');
            writeln(sm,'2   polygon');
            writeln(sm,'3   circle');
            writeln(sm,'4   pan');
            writeln(sm,'5   scrol');
            writeln(sm,'6   screen clear');
            writeln(sm,'7   End');
            w_chr('23');
            pos_cur('24','20');
            write('Enter Option  ');
            read(ch);                               (*read choise*)
            clear_chr;                              (*clear screen*)
            case ch of
                      '1' : polyline;
                      '2' : polygon;
                      '3' : circle;
                      '4' : pan;
                      '5' : scrol;
                      '6' : clear_it;
                      '7' : exit_gsx;
            else menu;
            end;                                    (*end of case*)
      end;                                          (*end of if*)
end;                                                (*end of procedure*)



(******************************************************************************)
procedure col_menu;
begin
    writeln('Enter color index <return>');
    writeln('0=Black  1=Red  2=Green  3=Blue  4=Cyan');
    writeln('5=Yellow  6=Magenta  7=White');
end;



(******************************************************************************)
procedure fil_menu;
begin
    writeln('Enter fill enterior style <return>');
    writeln('0=Hollow  1=Solid  3=Hatch');
end;



(*****************************************************************************)
(*This procedure erases the entire alpha screen.                             *)
(*****************************************************************************)
procedure clear_chr;
```

```
begin
    s:=CONCAT(esc,'[','2','J');
    writeln(s);
end;


(****************************************************************************)
(*This procedure moves the alpha cursor to the specified row and column     *)
(*address :                                                                 *)
(* s1 = row number                                                          *)
(* s2 = column number                                                       *)
(****************************************************************************)
procedure pos_cur(s1,s2:string);
begin
     s:=concat(esc,'[',s1,';',s2,'f');
     write(s);
end;


(****************************************************************************)
(*Set character colour                                                      *)
(* s1 = colour code                                                         *)
(****************************************************************************)
procedure w_chr(s1:string);
begin
    s:=concat(esc,'[',s1,';','m');
    write(s);
end;


(****************************************************************************)
(*This procedure draws a polyline and then transmits the relevant GSX      *)
(*information of that polyline.                                            *)
(****************************************************************************)
procedure polyline;
var
    i : integer;
begin
    col_menu;                    (*write colour choices*)
    readln(cindex);               (*read colour index*)
    set_attrib(17,cindex);
    writeln('Enter number of vertices(X/Y pairs) in polyline <return>');
    readln(vtc);
    k:=2*vtc;                     (*total number of x and y coordinates*)
    contrl[1]:=PLINE_CMD;         (*polyline opcode*)
    contrl[2]:=vtc;               (*number of vertices*)
    get_ptsin(k);                 (*get ptsin array*)
    GSX( ptsout, intout, ptsin, intin, contrl ) ;       (*draw polyline*)
    plen:=k+2;                    (*number of data words in the packet*)
    pdat[1]:=contrl[1];           (*set the packet data*)
    pdat[2]:=contrl[2];
```

```
    for i:=1 to k doo
           pdat[i+2]:=ptsin[i];
    send_data;                        (*send the packet*)
end;



(*************************************************************************)
(*This procedure sets the ptsin_array by reading x and y coordinates values.*)
(* nxys = number of xy pairs                                              *)
(*************************************************************************)
procedure get_ptsin(nxys:integer);
begin
    i:=1;
    m:=1;
    repeat
        writeln('Enter X',m:3,' <return>');
        readln(x);
        ptsin[i]:=x;
        i:=i+1;
        writeln('Enter Y',m:3,' <return>');
        readln(y);
        ptsin[i]:=y;
        i:=i+1;
        m:=m+1;
    until i=nxys+1;
    clear_chr;
end;



(************************************************************************)
(*This procedure draws a circle and then transmits the relevant GSX      *)
(*information for that circle.                                           *)
(************************************************************************)
procedure circle;
begin
    col_menu;
    readln(cindex);
    fil_menu;
    readln(fill);
    set_attrib(25,cindex);
    set_attrib(23,fill);
    writeln('Enter X-coordinate of center <return>');
    readln(x);
    writeln('Enter Y-coordinate of center <return>');
    readln(y);
    writeln('Enter Radius <return>');
    readln(r);
    clear_chr;
    contrl[1]:=GDP_CMD;               (*GDP opcode*)
    contrl[2]:=3;                     (*circle*)
```

```
    contrl[6]:=4;
    ptsin[1]:=x;
    ptsin[2]:=y;
    ptsin[5]:=r;
    GSX(ptsout,intout,ptsin,intin,contrl);    (*draw the circle*)
    plen:=6;                             (*packet length in words*)
    pdat[1]:=contrl[1];                  (*set the packet data*)
    pdat[2]:=contrl[2];
    pdat[3]:=contrl[6];
    pdat[4]:=ptsin[1];
    pdat[5]:=ptsin[2];
    pdat[6]:=ptsin[5];
    send_data;                           (*send the packet*)
 end;



(******************************************************************************)
(*This procedure sets the requested attribute and then transmits the relevant*)
(*GSX information for that attribute.                                        *)
(******************************************************************************)
procedure set_attrib( cmd, attribute : integer ) ;
begin
    contrl[1]:= cmd;
    contrl[2]:= 0;
    intin[ 1 ]  := attribute ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
    plen:=2;
    pdat[1]:=contrl[1];
    pdat[2]:=intin[1];
    send_data;
end;



(*********************** Close Workstation ********************************)
procedure exit_gsx ;
begin
    contrl[1]:=2;
    contrl[ 2 ] := 0 ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
    if rx=true then
          begin
             plen:=1;
             pdat[1]:=2;              (*close opcode*)
             send_data;
          end;
    rx:=false;
end ;
```

```
(*********************** Clear the screen ******************************)
procedure clear_it ;
begin
    contrl[ 1 ] :=3;
    contrl[ 2 ] := 0 ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
    if rx=true then
          begin
             plen:=1;
             pdat[1]:=3;          (*clear opcode*)
             send_data;
          end;
end ;


(*********************** Open Workstation *****************************)
procedure open_wk( dev_no : integer );
var
    i : integer ;
begin
    contrl[ 1 ] := 1 ;
    contrl[ 2 ] := 0;
    contrl[ 4 ] := 10 ;
    intin[ 1 ] := dev_no ;
    for i := 2 to 10 do
            intin[ i ] := 1 ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ;



(**********************************************************************)
(*This procedure draws a polygon and then transmits the polygon's relevant  *)
(*GSX information to the receiver.                                          *)
(**********************************************************************)
procedure polygon;
var
    i : integer;
begin
    col_menu;
    read(cindex);
    fil_menu;
    read(fill);
    set_attrib(25,cindex);
    set_attrib(23,fill);
    if fill=3 then
           set_attrib(24,3);
    writeln('Enter number of sides of polygon <return>');
    readln(vtc);
    k:=2*vtc;
    contrl[1]:=FILAREA_CMD;
```

```
    contrl[2]:=vtc;                         (*set number of vertices*)
    get_ptsin(k);                           (*go read x,y coordinates*)
    GSX(ptsout,intout,ptsin,intin,contrl);
    plen:=k+2;
    pdat[1]:=contrl[1];
    pdat[2]:=contrl[2];
    for i:=1 to k do
            pdat[i+2]:=ptsin[i];
    send_data;
end;


(*************************************************************************)
(*This procedure moves the display area horizontally from x=0 to x=1024 on  *)
(*the display memory,and in steps of 16 pixels.After the operation is       *)
(*completed a packet containing a code number for pan procedure is          *)
(*transmitted to the other end.                                             *)
(*************************************************************************)
procedure pan;
begin
    xsad:=0;                (*initialize x and y of the start point*)
    ysad:=0;
    repeat
        xsad:=xsad+16;      (*add 16 pixels to starting x address*)
        sead:=sead+1;       (*increment starting word address by one*)
        grscrol;            (*write the new start address*)
    until xsad=1024;        (*repeat until x=Xmax on the display memory*)
    sead:=0;                (*set display area's start address to normal*)
    grscrol;                (*write the start address*)
    plen:=1;                (*packet length in words*)
    pdat[1]:=40;            (*pan opcode*)
    send_data;              (*send the packet*)
end;


(*************************************************************************)
(*This procedure moves the display vertically from y=0 to y=1024 on the    *)
(*display memory,in steps of one pixel.After the operation is completed a  *)
(*code number specifying scrol procedure is sent to the other end.         *)
(*************************************************************************)
procedure scrol;
begin
    xsad:=0;                (*initialize x and y of the start point*)
    ysad:=0;
    repeat
        ysad:=ysad+1;       (*add one line to starting y address*)
        sead:=sead+64;      (*add pitch to the starting word address*)
        grscrol;            (*write the new start address*)
    until ysad=1024;        (*until y=maximum y on the display memory*)
```

```
    sead:=0;                 (*get the normal start address of the display*)
    grscrol;                 (*write the start address*)
    plen:=1;                 (*one data word to be sent*)
    pdat[1]:=41;             (*scrol opcode*)
    send_data;               (*send the packet*)
end;
```

```
(***************************************************************************)
(*This procedure sends a packet.A packet can be retransmitted if the receiver*)
(*requires.If the receiver is not ready a message is written on the screen   *)
(*and the transmission is stoped.                                            *)
(***************************************************************************)
procedure send_data;
begin
    datout(pdat);                 (*send the packet*)
    acknack;                      (*get the receiver's response*)
    if not retx and rx then       (*if retransmission not requested*)
            pnum:=pnum+1;         (*increment the packet number*)
    if retx and rx then           (*if retransmission requested*)
        begin
            writeln('Retransmission');
            send_data;            (*send the packet again*)
        end;
    if not rx then                (*if receiver is not ready*)
            writeln('Receiver is not ready');
end;
```

```
(****************************  Main Program  ******************************)
begin
    esc:=chr(27);
    rx:=false;
    pnum:=1;            (*initialize the packet number*)
    open_wk(1);
    clear_it;
    handshake;          (*check if receiver is ready*)
    repeat
        menu;
    until rx=false;
    w_chr('20');
end.
```

```
;**************************************************************************
;The assembly language module of the Pascal program CMDTX.PAS.
;**************************************************************************
        public    GSX
        public    grscrol
        public    datout
        public    putc
        public    getc
        public    acknack
        public    delay


      name        pasgsx
      assume      cs:code, ds:data
data  segment     public
        extrn     pnum : byte
        extrn     xsad : word,ysad : word
        extrn     sead : word
        extrn     rx : byte,retx : byte
        extrn     plen : byte
sum       dw      ?
retaddr   dw      ?
data    ends
code    segment   public
GDOS    equ       0e0h


;********************** GSX_86 interface procedure ************************
GSX     proc   near
        push   ds
        push   es
;
        mov    ax,ss
        mov    ds,ax
        mov    dx,sp
        add    dx,6
        mov    cx,473h
        int    GDOS
;
        pop    es
        pop    ds
        ret    20
GSX     endp


;**************************************************************************
;This procedure returns two flags rx and retx
;rx=0     Time out and receiver is not ready
;rx=1     Receiver is ready
;retx=0   Receiver sent ACK
;retx=1   Receiver sent NAK
```

```
;****************************************************************************
acknack  proc   near
        push    cx              ;register save
        push    bx
        mov     rx,1            ;assume receiver is ready
        mov     bx,00ffh        ;get a time out loop counter
rec00:
        mov     cx,0            ;init time out counter for 8251
rec10:
        in      al,32h          ;read 8251 status register
        test    al,02h          ;check rxrdy bit
        jnz     rec20           ;jump if a byte is ready to receive
        loop    rec10           ;repeat if time is not out
        jmp     rec40           ;jump to continue the main loop
rec20:
        call    getc            ;go receive the byte
        cmp     al,0aah         ;is it low byte of ack?
        jz      rec30           ;yes then jump
        cmp     al,0ffh         ;iis it low byte of nak?
        jz      rec60           ;yes then jump
        jmp     rec40           ;jump to continue the loop
rec30:
        call    getc            ;get the next byte
        cmp     al,55h          ;is it high byte of ack?
        jz      ack00           ;yes then jump
        jmp     rec40           ;jump to continue the loop
rec60:
        call    getc            ;get the next byte
        cmp     al,00           ;is it high byte of nak?
        jz      rec70           ;if yes jump
rec40:
        dec     bx              ;decrement the loop counter
        jnz     rec00           ;if bx not zero,then continue the loop
        mov     rx,0            ;time is out and receiver is not ready
        jmp     rec50           ;jump to return
rec70:
        mov     retx,1          ;nak is received,retransmission is asked
        jmp     rec50           ;jump to return
ack00:
        mov     retx,0          ;ack is received
rec50:
        pop     bx              ;restore registers
        pop     cx
        ret
acknack  endp


;****************************************************************************
;This procedure sends a packet by transmitting the packet number,packet length
;packet data and packet checksum.
;****************************************************************************
datout  proc    near
```

```
        pop     bx              ;get the return address
        mov     retaddr,bx      ;store it
        pop     ax              ;get offset address of the packet data
        pop     bx              ;get segment address of the packet data
        push    di              ;save the offset register
        push    es              ;save the segment register
        mov     di,ax           ;get the offset
        mov     es,bx           ;get the segment
        mov     al,pnum         ;get the packet number
        call    putc            ;transmit the packet number
        mov     al,plen         ;get the packet length
        call    putc            ;transmit
        mov     ah,0            ;set high byte of ax to zero
        mov     cx,ax           ;load cx with the packet length
        mov     sum,0           ;initialize the sum
        add     cx,cx           ;packet length in bytes
dat10:
        mov     al,es:byte ptr[di]      ;get the packet data byte
        mov     ah,0                    ;set high byte to zero
        add     sum,ax                  ;add data together
        call    putc                    ;send the packet data out
        inc     di                      ;point to the next data byte
        loop    dat10                   ;repeat until all bytes are sent
        mov     ax,sum                  ;get the checksum
        call    putc                    ;send low byte of the checksum
        mov     al,ah                   ;get the high byte
        call    putc                    ;send high byte of the checksum
        pop     es                      ;recover segment register
        pop     di                      ;recover offset register
        mov     bx,retaddr              ;get the return address
        jmp     bx                      ;return
datout  endp


;*****************************************************************************
;This procedure uses BDOS function 3 to receive a byte from the serial I/O
;communication controller.
;                 output :  al = received byte
;*****************************************************************************
getc    proc    near
        push    es              ;save registers
        push    di
        push    si
        push    dx
        push    cx
        push    bx
        mov     cl,3            ;load cl with the function code
        int     224             ;call BDOS
        pop     bx              ;restore registers
        pop     cx
        pop     dx
        pop     si
```

```
        pop     di
        pop     es
        ret
getc    endp



;**************************************************************************
;This procedure uses BDOS function 4 to transmit a byte via serial I/O
;communication controller.
;                    input  al = sent byte
;**************************************************************************
putc    proc    near
        push    ax              ;register save
        push    es
        push    di
        push    si
        push    dx
        push    cx
        push    bx
        mov     dl,al           ;load dl with the byte to be sent
        mov     cl,4            ;load cl with the function code
        int     224             ;call BDOS
        pop     bx              ;recover register
        pop     cx
        pop     dx
        pop     si
        pop     di
        pop     es
        pop     ax
        ret
putc   endp



;**************************************************************************
;This procedure moves the display by  changing the display starting address.
;The new start address is given to the GDC via the PRAM command.
;**************************************************************************
grscrol proc    near
        mov     al,70h          ;PRAM command + PRAM start address of zero
        out     72h,al          ;send it out
        mov     ax,sead         ;get the new start address
        out     70h,al          ;send low byte first
        mov     al,ah           ;get the high byte
        out     70h,al          ;send high byte of start address
        call    delay           ;slow the movement
        ret
grscrol   endp
```

```
;*****************************************************************************
delay   proc    near
        mov     bx,0fffh
del00:
        dec     bx
        jnz     del00
        ret
delay   endp
code    ends
        end
```

```
(****************************************************************************)
(*This program demonstrates how graphic operations may be performed by      *)
(*receiving their GKS information.                                          *)
(*It has an assembly language module named RCMDASM.I86.                     *)
(****************************************************************************)
(* a:                                                                       *)
(* mt+86 b:cmdrx                                                            *)
(* asmt86 b:rcmdasm                                                         *)
(* linkmt b:cmdrx,b:rcmdasm,fpreals,trancend,paslib/s                       *)
(* graphics                                                                 *)
(* b:                                                                       *)
(* cmdrx                                                                    *)
(****************************************************************************)
program receiver;
const

    MAX_CNTL_VALS       = 10 ;
    MAX_INTIN_VALS      = 80 ;
    MAX_INTOUT_VALS     = 45 ;
    MAX_PTS_VALS        = 100 ;

    SCREENSIZE          = 32767;

TYPE
    cntrl_array  = array  [ 1..MAX_CNTL_VALS ] of integer ;
    intin_array  = array  [ 1..MAX_INTIN_VALS ] of integer ;
    intout_array = array  [ 1..MAX_INTOUT_VALS ] of integer ;
    ptsin_array  = array  [ 1..MAX_PTS_VALS ] of integer;
    ptsout_array = array  [ 1..MAX_PTS_VALS ] of integer ;
    pdat_array   = array  [ 1..256 ] of integer ;
VAR
    contrl       :      cntrl_array;
    intin        :      intin_array;
```

```
    intout          :      intout_arrray;
    ptsin           :      ptsin_array;
    ptsout          :      ptsout_array;
    pdat            :      pdat_array;
    rxpnum          :      integer;          (*expected packet number*)
    flag            :      boolean;          (*error flag*)
    tx              :      boolean;          (*transmitter ready flag*)
    xsad,ysad       :      integer;          (*x,y addresses in the display memory*)
    sead            :      integer;          (*word address in the display memory*)
    cmd             :      integer;          (*opcode*)
    i,k,m           :      integer;          (*dummy variables*)
    esc             :      char;
    ch              :      char;


external procedure GSX( var ptsout :   ptsout_array ;
                        var intout :   intout_array ;
                        var ptsin  :   ptsin_array ;
                        var intin  :   intin_array ;
                        var contrl :   cntrl_array ) ;

external procedure getdat(var pdat : ptsout_array);
external procedure handsk;
external procedure grscrol;
external procedure datin;
external procedure acknack;



(*****************************************************************************)
(* This procedure receives a packet of data.                                *)
(*****************************************************************************)
procedure receive;
begin
    flag:=true;
    repeat
       handsk;                              (*waite to receive a packet*)
       if tx=false then                     (*if transmitter is not ready*)
             begin
                writeln('Transmitter is not ready');
                flag:=true;
             end;
       if tx=true then                      (*if transmitter is ready*)
             begin
                datin;                      (*receive a packet*)
                if flag=true then           (*if no errors*)
                    begin
                      getdat(pdat);         (*go get the received data from*)
                                            (*assembly module*)
                      fig_draw;             (*draw the figure*)
                      rxpnum:=rxpnum+1;     (*next packet number*)
```

```
                    end
                else                      (*if error occurred,write*)
                                          (*a message*)
                    writeln('Retransmission');
             end;
    until flag=true;
end;
```

```
(*****************************************************************************)
(*This procedure performs a graphical operation according to the received   *)
(*data.                                                                      *)
(*****************************************************************************)
procedure fig_draw;
begin
    cmd:=pdat[1];                         (*get the opcode*)
    case cmd of
           2  :  exit_gsx;
           3  :  clear_it;
           6  :  mlafig;
           9  :  mlafig;
           11 :  circle;
           17 :  set_attrib;
           23 :  set_attrib;
           24 :  set_attrib;
           25 :  set_attrib;
           40 :  pan;
           41 :  scrol;
    end;                                  (*case*)
end;                                      (*procedure*)
```

```
(*****************************************************************************)
(*This procedure draws a circle.                                             *)
(*****************************************************************************)
procedure circle;
var
    x,y,r,fill,cindex:integer;
begin
    contrl[1]:=11;
    contrl[2]:=pdat[2];
    contrl[6]:=pdat[3];
    ptsin[1]:=pdat[4];
    ptsin[2]:=pdat[5];
    ptsin[3]:=0;
    ptsin[4]:=0;
    ptsin[5]:=pdat[6];
    ptsin[6]:=0;
    GSX(ptsout,intout,ptsin,intin,contrl);
```

```
 end;



(**************************************************************************)
procedure set_attrib;
begin
    contrl[1]:= pdat[1];
    contrl[2]:= 0;
    intin[ 1 ]  := pdat[2];
    GSX( ptsout, intout, ptsin, intin, contrl );
end;



(***************************************************************************)
(*This procedure draws a polymarker,a polyline or fills a polygon.        *)
(**************************************************************************)
procedure mlafig;
var
    j,m,i,cindex:integer;
begin
    contrl[1]:=pdat[1];                    (*set opcode*)
    contrl[2]:=pdat[2];                    (*set number of vertices*)
    k:=2*pdat[2];
    for i:=1 to k do
            ptsin[i]:=pdat[i+2];           (*get the x,y coordinates*)
    GSX(ptsout,intout,ptsin,intin,contrl);
end;



(************************** Close Workstation *****************************)
procedure exit_gsx ;
begin
    contrl[1]:=2;
    contrl[ 2 ] :=0;
    GSX(ptsout,intout,ptsin,intin,contrl);
    flag:=true;
    acknack;
    tx:=false;
end ;



(************************* Clear Workstation ******************************)
procedure clear_it ;
begin
```

```
    contrl[ 1 ] :=3;
    contrl[ 2 ] := 0;
    GSX(ptsout,intout,ptsin,intin,contrl);
end;


(************************** Open Workstation ********************************)
procedure open_wk( dev_no : integer );
var
     i : integer ;
begin
    contrl[ 1 ] := 1;
    contrl[ 2 ] := 0;
    contrl[ 4 ] := 10;
    intin[ 1 ] := dev_no;
    for i := 2 to 10 do
             intin[ i ] := 1;
    GSX(ptsout,intout,ptsin,intin,contrl);
end ;


(***************************************************************************)
(*This procedure moves the screen window horizontally on the display memory *)
(***************************************************************************)
procedure pan;
begin
    xsad:=0;             (*display area starting x and y addresses*)
    ysad:=0;
    sead:=0;             (*display area starting word address*)
    repeat
        xsad:=xsad+16;   (*adjust next x address*)
        sead:=sead+1;    (*starting word address increments by one word*)
        grscrol;         (*move the display*)
    until xsad=1024;
    sead:=0;             (*normal display starting address*)
    grscrol;             (*move the display to normal position*)
end;


(***************************************************************************)
(*This procedure moves the screen window vertically on the display memory.  *)
(***************************************************************************)
procedure scrol;
var
    n : integer;
begin
    xsad:=0;             (*display area starting x and y addresses*)
    ysad:=0;
    sead:=0;             (*display area starting word address*)
```

```
    repeat
        ysad:=ysad+1;       (*adjust next y address*)
        sead:=sead+64;      (*starting word address increases by 'pitch'*)
        grscrol;            (*move the display*)
    until ysad=1024;
    sead:=0;
    grscrol;
end;
```

```
(***************************  Main Program  ********************************)
begin
    open_wk(1);
    clear_it;
    rxpnum:=1;              (*packet numbers start from one*)
    repeat
          receive;          (*receive the data*)
    until tx=false;
    read(ch);
    clear_it;
    exit_gsx;
end.
```

```
;***************************************************************************
;The assembly language module of the pascal program CMDRX.PAS.
;***************************************************************************
      public    GSX
      public    handsk
        public    acknack
        public    datin
        public    putc
        public    getdat
        public    grscrol
        public    rtest,rsrv00
        public    delay

        name      pasgsx
        assume    cs:code, ds:data
data    segment   public
retaddr dw     ?
rlc     dw     ?
suml    db     ?
sumh    db     ?
sum     dw     ?
txpnum  db     ?
plen    db     ?
```

```
        extrn     sead : word
        extrn     tx : byte
        extrn     flag : byte
        extrn     rxpnum : byte
data    ends
code    segment   public
GDOS    equ       0e0h


;***************************************************************************
;                   GSX_86 standard interface procedure
;***************************************************************************
GSX     proc      near
        push      ds
        push      es
;
        mov       ax,ss
        mov       ds,ax
        mov       dx,sp
        add       dx,6
        mov       cx,473h
        int       GDOS
;
        pop       es
        pop       ds
        ret       20
GSX     endp




;***************************************************************************
;This procedure sends an ACK or a NAK to the transmitter and waits to receive
;a packet. The first byte of data which is the packet number is received here.
;The rest of the data is received in the procedure datin.
;If the transmitter doesn't send data during a time out loop, the TX flag will
;go to zero.
;***************************************************************************
handsk  proc  near
        mov   rlc,10           ;set the loop counter for the proc rtest
        mov   cx,003fh         ;time out loop counter
hand20:
        call  acknack          ;send ACK or NAK
        call  rtest            ;wait for data
        test  ah,80h           ;test if data is ready
        jz    hand30           ;if yes jump out of the loop
        loop  hand20           ;go back to the loop
        mov   tx,0             ;time is out then set tx to zero
        jmp   hand40           ;jump to return
hand30:
        mov   tx,1             ;set the tx fflag to one
```

```
        mov     txpnum,al           ;the first received byte is the packet number
hand40:
        ret
handsk  endp
```

```
;*****************************************************************************
;This procedure sends an ACK if flag=1, and a NAK if flag=0.
;*****************************************************************************
acknack  proc near
        cmp     flag,1              ;is flag equal one?
        jz      ack00               ;yes then jump to send ACK
        mov     ax,00ffh            ;get NAK word
        jmp     ack10               ;jump to send NAK
ack00:
        mov     ax,55aah            ;get ACK word
ack10:
        call    putc                ;send low byte first
        mov     al,ah               ;then high byte
        call    putc
        ret
acknack  endp
```

```
;*****************************************************************************
;This procedure receives a packet and its checksum.It calculates the sum of
;all bytes in the received packet and compares it with the received checksum,
;if they are not equal then the error flag will be reset.The error flag will
;also be reset if one or more data bytes are missing.
;*****************************************************************************
datin   proc    near
        mov     sum,0               ;initialize the sum
        mov     rlc,5               ;set loop counter for the proc rtest
        mov     bx,(offset tine)        ;get the address of the storage buffer
        call    rtest               ;go and get the first data byte
        test    ah,80h              ;is data received?
        jnz     error               ;if not then jump to error
        mov     plen,al             ;save the received byte
        mov     ah,0                ;set high byte to zero
        mov     cx,ax               ;get the packet length
        add     cx,cx               ;loop counter is number of bytes to be received
daloop:
        call    rtest               ;get the next byte
        test    ah,80h              ;is data received?
        jnz     error               ;if not then jump to error
```

```
        mov   ah,0              ;set high byte to zero
        add   sum,ax            ;add received bytes together
        mov   cs:byte ptr[bx],al      ;save the received byte
        inc   bx                ;point to the next location
        loop  daloop
        call  rtest             ;go to get the sumcheck
        test  ah,80h
        jnz   error
        mov   suml,al           ;save it
        call  rtest             ;get data
        test  ah,80h
        jnz   error
        mov   sumh,al           ;store it
        mov   al,suml           ;get the low byte of the sumcheck
        mov   ah,sumh           ;get the high byte
        cmp   ax,sum            ;compare the received sum with the expected one
        jnz   error             ;if not equal then error occurred
        mov   al,txpnum         ;get the transmitter packet number
        cmp   al,rxpnum         ;compare it with the expected one
        jnz   error             ;if not equal then error occurred
        mov   flag,1            ;set flag to no error
        jmp   datinret          ;jump to return
error:
        mov   flag,0            ;indicate an error
datinret:
        ret
datin   endp



;*****************************************************************************
;This procedure passes the received data to pdat array in the Pascal program.
;The array start address is passed from Pascal to this procedure.
;*****************************************************************************
getdat  proc  near
        pop   bx                ;get the return address
        mov   retaddr,bx        ;save it
        pop   bx                ;restore pdat offset address
        pop   ax                ;restore pdat segment address
        push  di                ;save offset register
        push  es                ;save segment register
        mov   di,bx             ;set the offset address
        mov   es,ax             ;set the segment address
        mov   bx,(offset tine)       ;get the address of the storage buffer
        mov   al,plen                ;get the number of words in the packet
        mov   ah,0                   ;set high byte to zero
        mov   cx,ax                  ;set the loop counter
getdat10:
        mov   ax,cs:word ptr[bx]     ;get word from the storage buffer
        add   bx,2                   ;point to the next word of buffer
        mov   es:word ptr[di],ax     ;put the word in the pdat array
        add   di,2                   ;point to the next word of the array
        loop  getdat10
```

```
        pop    es                    ;restore the segment register
        pop    di                    ;restore the offset register
        mov    bx,retaddr            ;get the return address
        jmp    bx                    ;return
getdat  endp



;*****************************************************************************
;This procedure uses BDOS function 4 to transmit a data byte.
;                 input     al = send byte
;*****************************************************************************
putc    proc   near
        push   ax                    ;save registers
        push   es
        push   di
        push   si
        push   dx
        push   cx
        push   bx
        mov    dl,al                 ;get the byte to be sent
        mov    cl,4                  ;function code
        int    224                   ;BDOS interrupt number
        pop    bx
        pop    cx
        pop    dx
        pop    si
        pop    di
        pop    es
        pop    ax
        ret
putc   endp



;*****************************************************************************
;This procedure returns a test value in high byte of ax register to indicate
;whether data is received or missed.The received data is returned in the low
;byte of ax register.
;*****************************************************************************
rtest   proc   near
        push   dx                    ;register save
        push   bx
        mov    bx,rlc                ;get the loop counter
reader00:
        call   rsrv00                ;go to receive data
        mov    dl,al                 ;save the received byte
        test   ah,80h                ;test if data is arrived
```

```
        jz      reader10
        dec     bx                      ;decrement loop counter
        jnz     reader00                ;continue if time is not out
reader10:
        pop     bx
        mov     al,dl                   ;put the byte in al
        pop     dx
        ret
rtest   endp




;*****************************************************************************
;This procedure programs the serial I/O communication controller,8251A,to
;receive data from an RS232 modem.It is assummed that the chip has been
;initialized when the computer has been powered up.
;*****************************************************************************
rsrv00  proc    near
        push    bx
        push    cx
        push    dx
        mov     al,07h                  ;modem ER on
        out     32h,al                  ;command register
        sub     cx,cx                   ;initialize time out counter
rsrv10:
        in      al,32h                  ;read status
        test    al,80h                  ;test DR
        jnz     rsrv20                  ;jump if DR is on
        loop    rsrv10                  ;wait DR
        jmp     rsrv30                  ;jump if time is out
rsrv20:
        test    al,02h                  ;test receiver ready bit
        jnz     rsrv40
        loop    rsrv10                  ;wait receiver
rsrv30:
        or      al,80h                  ;set time out
        mov     ah,al
        sub     al,al
        jmp     rsret
rsrv40:
        and     al,7fh                  ;clear time out bit
        mov     ah,al
        in      al,30h                  ;receive the byte
rsret:
        pop     dx
        pop     cx
        pop     bx
        ret
rsrv00  endp
```

```
;***************************************************************************
;This procedure moves the screen window by changing the window starting
;address in the PRAM.
;***************************************************************************
grscrol  proc near
        mov   al,70h                 ;PRAM command + PRAM starting address of 0
        out   72h,al                 ;send it to the GDC
        mov   ax,sead                ;get the new start address
        out   70h,al                 ;send low byte first
        mov   al,ah                  ;then high byte
        out   70h,al
        call  delay                  ;slow the motion
        ret
grscrol   endp




;***************************************************************************
delay   proc   near
        mov    bx,0fffh
del00:
        dec    bx
        jnz    del00
        ret
delay   endp




;************************* Storage buffer *********************************
tine:
        dw  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        dw  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        dw  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        dw  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        dw  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        dw  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
code    ends
        end
```