528 c

A RASTER-SCAN VIDEO GRAPHICS DISPLAY SYSTEM

A RASTER-SCAN
VIDEO GRAPHICS DISPLAY SYSTEM
IMPLEMENTED ON
A KIM MICROCOMPUTER

By

MARK DOUGLAS DRUMMOND, B.SC., M.SC.

A Project

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

September 1980

MASTER OF SCIENCE (1980)
(Computation)

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE: A Raster-scan Video Graphics Display System
Implemented on a KIM Microcomputer

AUTHOR: Mark Douglas Drummond B.Sc. (Notre Dame University,
Nelson, B.C.)
M.Sc. (Theoretical Physics,
McMaster University)

SUPERVISOR : Dr. N.S. Solntseff

NUMBER OF PAGES : vi, 69

ABSTRACT

The "microelectronic revolution" and the accompanying decrease in the cost of semiconductor memory has increased the availability of raster-scan graphical displays, yet, as pointed out in a recent survey [BAE79], the implementation of graphics software for raster-scan systems has lagged behind that for random-scan systems. The aim of the work described in this report has been to apply random-scan display techniques to a system employing a relatively inexpensive raster-scan device. The system, incorporating a segmented display file processor, is implemented on a KIM™ microcomputer. The display device is composed of a Micro Technology Unlimited™ (MTU) 8K Visable Memory™ (VM) video board and a standard TV monitor.

ACKNOWLEDGMENTS

I would like to acknowledge financial support provided in part by Grant No. A8327 of the Natural Science and Engineering Council of Canada and in part by special funding to the Computation programme by Imperial Oil of Canada Limited.

I also wish to express my appreciation to the members of the McMaster University Unit for Computer Science for the many forms of help provided during my studies with them.

Finally, it is my pleasure to thank Dr. Nick Solntseff for suggesting this project and for his helpful direction and encouragement over the past year.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

1.1 The Role of Computer Graphics

The inability to communicate with the computer in a straightforward manner is a common source of error and frustration for many users of mankind's newest and most powerful tool. The advent of the "microprocessor revolution" and the accompanying increase in computer use by untrained personnel has emphasized the problem. In many cases, the man-machine communication can be significantly improved through the use of a well-designed video graphics system.

The part of a computer program that determines how the user and computer communicate is called the user interface and, as Newman and Sproull point out ([NEW79], p.443), its "...design has a particularly strong impact on program acceptability as a whole." The usual communication medium consisting of a typewriter-style keyboard and alphanumeric display, is not the most appropriate one for many applications. In typical keyboard dialogues "...the user must type many responses to achieve even the simplest results and must be willing to provide information in the exact order requested by the computer." ([NEW79], p.453.) Ideally, the interface allows the user to ignore the detailed workings of the computer and to concentrate on the task at hand, but "For design applications and other relatively creative tasks, the [keyboard] dialogue is unduly restrictive." Although natural language interactions, both spoken and written can be implemented, they

are, according to Newman and Sproull, too inefficient to be a practical alternative for most applications ([NEW79], p.455).

The principle argument in favour of the use of graphical computer displays lies in the long recognized fact that pictures provide effective communication enhancement. A video graphics display and a graphical input device such as a lightpen, joystick or digitizing tablet provide the user-interface designer with the means of overcoming most of the disadvantages inherent in keyboard-based interfaces. Typically, the input device is used to reposition an on-screen cursor or to select one element from a menu. By continually updating the display, the computer provides feedback that ensures the user that his commands are accurately received and fully understood by the program. The use of menu-selection prevents the user from choosing invalid items or erroneous commands and simplifies the user-computer interaction by eliminating typing, spelling and other syntax errors. This is especially important in applications in which untrained workers employ the computer in performing inherently difficult tasks. Applications which benefit from incorporation of graphics oriented interaction include game playing, simulation, computer aided design (CAD) [SAT77], musical composition [BUX78] and even program writing (See [NEW79], p.503 et seq. for an extensive bibliography.).

1.2 Types of Display Devices

Pictures may be composed in two distinct ways: The first is typified by the newspaper cartoon made up of an ordered set of lines; the second is exemplified by a hooked rug where the picture is composed

the second is exemplified by a hooked rug where the picture is composed of a collection of points. This distinction is also present in computer graphics. Random-scan (also called vector, or directed beam) display devices employ the first approach. TV-like, raster-scan devices employ the second. Both types use the cathode ray tube (CRT) as the basic display element. In a random-scan device, digital information from the computer is converted to an analog signal which is fed to the horizontal and vertical deflection plates of the CRT causing movement of the electron beam. If the beam is allowed to reach the front of the tube, a solid line will be traced on the screen. Alternately, the path traced by the electron beam in the CRT can be determined by external hardware located in the device itself as in a TV. In the latter case, a picture is drawn by switching on and off, or modulating, the electron beam as the predetermined path, or raster, is scanned. The computer controls the appearance of the display by storing numbers in a special region of memory called a frame buffer or bit map. This frame buffer may consist of a part of the computer memory or it may be a dedicated block of memory which is a part of the display device itself. By combining the contents of the frame buffer with the signal modulating the electron beam, the raster-scan device can produce the picture represented by the bits stored in the frame buffer. Each bit in the frame buffer appears on the display as a dot called a picture element or pixel. Some electronic hardware of course, is required to synchronize the scan of the frame buffer with that of the electron beam and to allow mutual access to the frame buffer by the computer and the display device.

Comparison of random-scan and raster-scan devices illustrates several important differences. The number of dots displayed by a raster-scan device, and the area occupied by the display on the screen determine the resolution of the display. High resolution displays require a large frame buffer which, in the past, has been prohibitively expensive. The "microelectronic revolution" and the accompanying decrease in the cost of semiconductor memory, however, has made high resolution raster-scan displays more feasible. On the other hand, random-scan display devices require only enough memory to store the beginning and ending points of the lines to be drawn. These lines though, must be continually redrawn or refreshed by the computer or by the display device if they are to remain visible. As a consequence, the number of lines that can be drawn is limited. When the number of lines to be drawn increases, the time required to redraw the display also increases until eventually, the display begins to flicker and becomes discontinuous. This problem can be overcome using a modified CRT called a direct-view storage tube (DVST) which employs a higher persistence phosphour coating on the inside face of the CRT and other modifications ([NEW79], p.39) to decrease the required refresh rate. Unfortunately, this means that the ability to selectively erase portions of the screen is lost, so that these terminals are usually restricted to the display of static pictures.

The need to continually refresh a random-scan display limits the complexity of the display, but it may also be used to advantage for dynamic displays in which some part of a picture is continually moved to

a new position on the screen. In such cases, no erasing is required and the moving portion of the picture need only be redrawn in the new position. In contrast, a raster-scan device must continually erase the moving portion of the picture by drawing it in the opposite or background "colour" before it can be redrawn in a new position. This may lead to the slowing down of dynamic displays and, more seriously, to the erasure of any static portions of the picture which happen to overlap the moving part. Finding and redrawing the intersections of two portions of a picture can be difficult and time-consuming. With a random-scan device, no such problem arises because the display is continually redrawn by the hardware.

Raster-scan devices have another, more fundamental, drawback: Pictures drawn may appear as a collection of dots rather than as a continuous image unless the resolution is high. Drawing one of the simplest pictures, a line, requires a special algorithm to ensure that it appears straight, terminates correctly, and has a density which is constant and independent of the length or angle. In addition, it must generate the same set of points independently of the direction in which the line is traversed to ensure that any subsequent erasure is complete and accurate. Since pictures are composed of many line segments, the algorithm must also be efficient. Such algorithms are well-known however ([NEW79], pp.20-27) and, in practice, the staircase effect in the appearance of diagonal lines is only a minor inconvenience if the resolution of the display is good.

In several respects, raster-scan devices are superior to those

employing a random-scan. A full-colour display can easily be achieved by adding a colour monitor and assigning additional memory bits to the representation of each pixel in the frame buffer. Another consideration in favour of raster-scan devices "...is their ability to display images containing solid areas along with lines and text. These solid areas may represent lines of various thicknesses, colored geometric shapes or facets of three-dimensional objects." ([NEW79], p.229.) The majority of random-scan displays on the other hand, are monochromatic and require numerous lines to shade even small areas of the display.

Another factor in favour of raster-scan displays is the ability to use the readily available, standard television as the display device with only minor modifications [LAN76]. As well, the highly developed existing video technology can be easily adapted to use in graphics systems [LIP80]. Included are devices such as video disks and tapes which permit storage and manipulation of raster-scan displays. Finally, since they employ analog circuitry, random-scan devices are subject to the problems of voltage drift caused by temperature fluctuations and other external influences. The design of a raster-scan device is simpler because temperature compensation and other such complexities are not major considerations. In the final analysis though, the choice between the two types of displays will depend upon the particular application and upon the materials available to the designer [CIA76].

1.3 Picture Structure

When a graphics display system is used to exhibit the image of a

picture, the representation of that image must be stored in the system in digital form. The exact nature of the representation chosen is an important design consideration; for several reasons, it is desirable that the representation reflect the structure that is clearly evident in most pictures. In an actual implementation however, the representation depends on the type of display device employed. In the case of a raster-scan device, the simplest level of representation is the frame buffer in which contiguous memory bits correspond to the discrete points in the original picture. As Figure 1.1 shows, there is no corresponding representation for random-scan displays; instead, the simplest possible representation is an encoded sequence of lines that must be drawn to produce a picture. This sequence, called a display file ([NEW79], p.46), is an integral feature of the display device and is employed primarily to refresh the displayed image. It is possible to simulate the natural random-scan encoded representation in a raster-scan system. This higher level of representation consists of a sequence of basic elements or primitives which correspond to the points, lines and text characters of the picture. On a raster-scan system, the encoded picture definition must be interpreted or scan converted by the equivalent of a display processor to produce the frame buffer for the picture. For this report, alternate terminology is used to distinquish random-scan processes, which are predominately hardware implemented, from software-implemented raster-scan processes. Following Giloi ([GIL78], p.12), the raster-scan structure corresponding to a random-scan display file is called the display processor program (DPP), and the software

equivalent of the display processor, is called the display program interpreter (DPI).

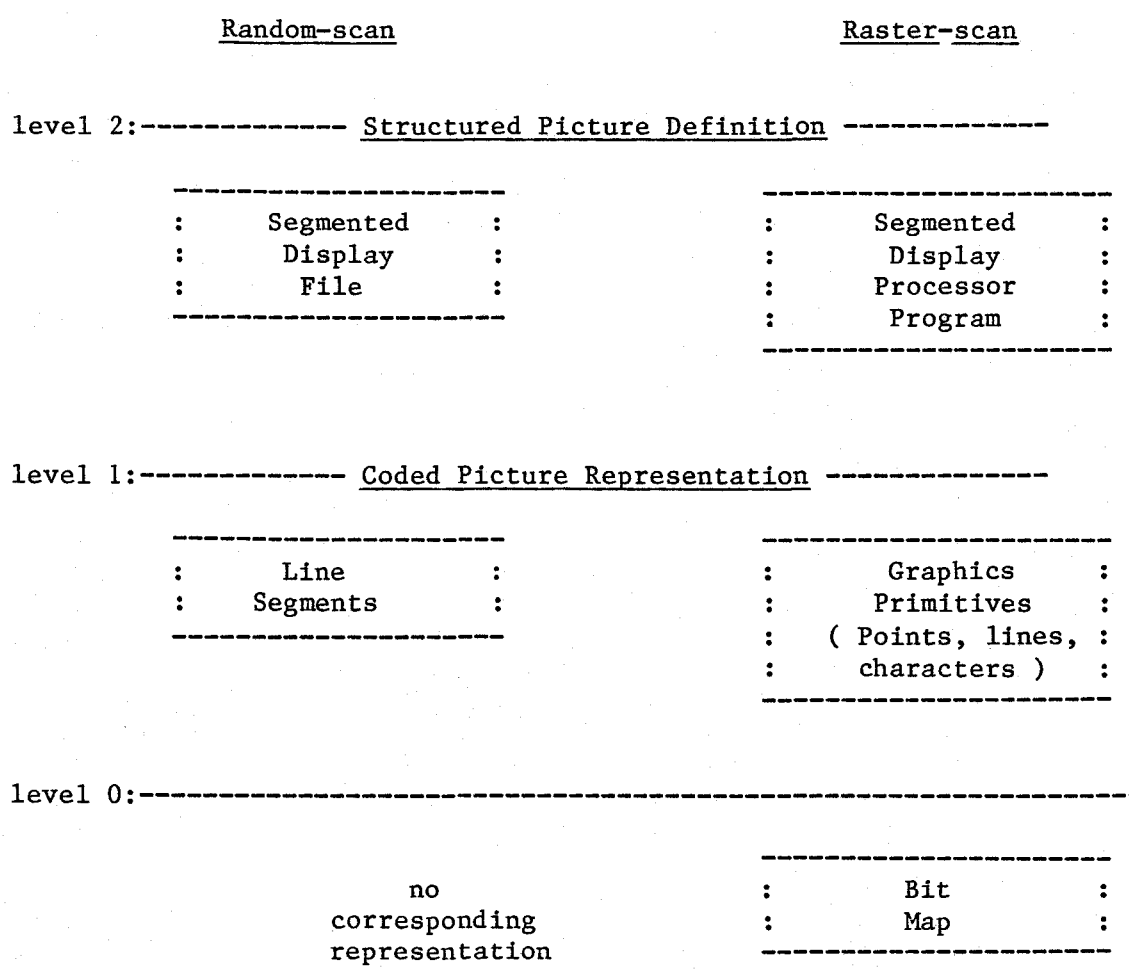

Figure 1.1 : The Picture Representation Hierarchy

At yet a higher level of representation, both the display file and the display processor program may consist of named groups of the

graphics primitives which together form a complete figure or other significant part of the picture. These groups, commonly called segments ([NEW79], p.93), may thus be considered as a hierarchically structured collection of instructions to be executed in order to produce a picture on the display. They serve to impose a structure on the digital representation that is a direct reflection of the structure possessed by the picture itself. Figure 1.1 shows the possible levels of representation for both raster and random-scan systems.

The utility of organizing picture descriptions into segments is well established by experience with random-scan devices [GIL75] [NEW79], where segments serve primarily as an organizational device which enable selective manipulation and dynamic modification of the display. For raster-scan devices however, the segmented display file may also serve to reduce regeneration of the display since an individual segment can be selectively redrawn when modifications to it are affected. This work saving effect is especially significant for dynamic displays. In the past, designs employing a coded picture definition have been implemented exclusively on random-scan display devices. However, as pointed out in a comprehensive survey by Baecker [BAE79], "There is an increasing body of opinion which holds that systems software for raster-scan devices can be constructed in the same manner as that for vector-scan graphics." Although this has been recognized for some time [SPR75], few systems have in fact been constructed. Thus, the present work represents an exploratory effort to assess the feasibility of using random-scan display techniques for computer graphics on raster-scan devices.

1.4 System Overview

Several distinct types of video display system architecture may be distinquished. The one described in this work is of the type called "systems with coded picture definition" by Baecker [BAE79]. Figure 1.2 shows the correspondence between the KIM design and that defined by his Figure 10. Three separate processes are present and, in general, each may consist of any hardware-software combination. The display system described here is implemented on a Commodore/MOS Technology KIM microcomputer which is based on the MCS6502 CPU chip. The system consists of the KIM CPU board, 32 Kbytes of MOS random access memory (RAM) and a Micro Technology Unlimited (MTU) Visable Memory (VM) video board. Processes P1 and P2 of Figure 1.2 consist of MCS6502 assembly-language programs. Display generation, represented by P3, is accomplished by the VM on-board hardware. Picture data generated in the host CPU are transmitted in coded format to the KIM where they are read by process P1 and stored in KIM memory as a display processor program. The encoded picture is then scan converted and stored in the frame buffer by process P2, the display program interpreter.

The host CPU is an Ohio Scientific Instruments (OSI) Challenger III microcomputer having 48 Kbytes of MOS RAM and an OSI CA-102 550 serial port communications board. The 550 board provides a standard RS232 serial communication link with the KIM.

Descriptions of the two processes P1 and P2 are given in Chapters 2 and 3 respectively. Chapter 4 gives a brief discussion of

the host software, implemented to control the KIM video display system. Chapter 5 discusses possible directions for expansion and use of the system.

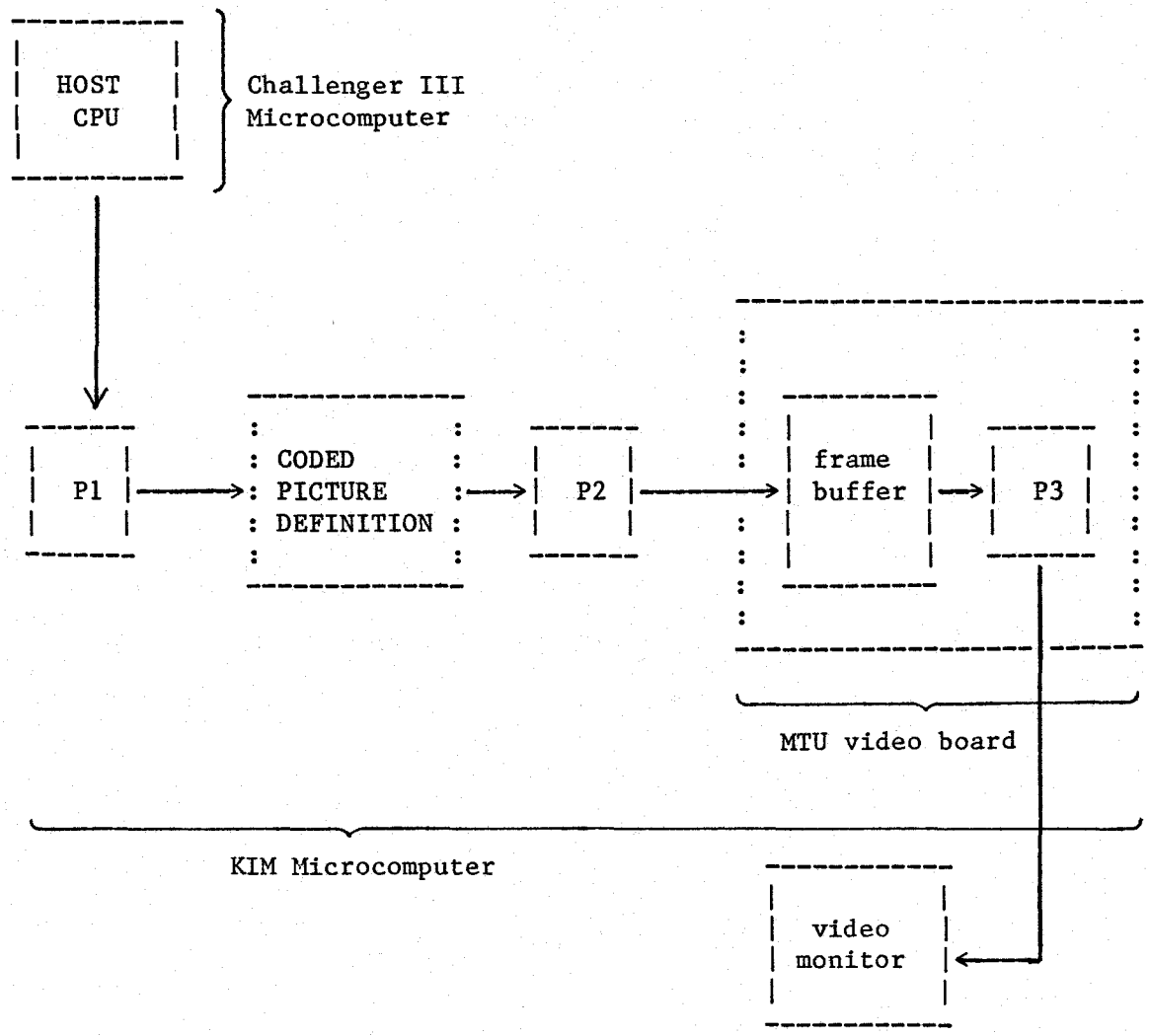

Figure 1.2 : The KIM display system

CHAPTER 2

THE COMMUNICATIONS PROCESS

The video display system described in this report is composed of two principle processes, represented by P1 and P2 in Figure 1.2. The description of the system commences below, with discussion of the function and operation of the communications process, P1.

The communications process is itself composed of four functionally distinct modules. Each of these modules performs a single task related to creation and maintenance of the display processor program. Figure 2.1 shows the relationship between the four processes which will be referred to as the Supervisor, Reader, Display Program Manager and Free-storage Manager. Each is described below.

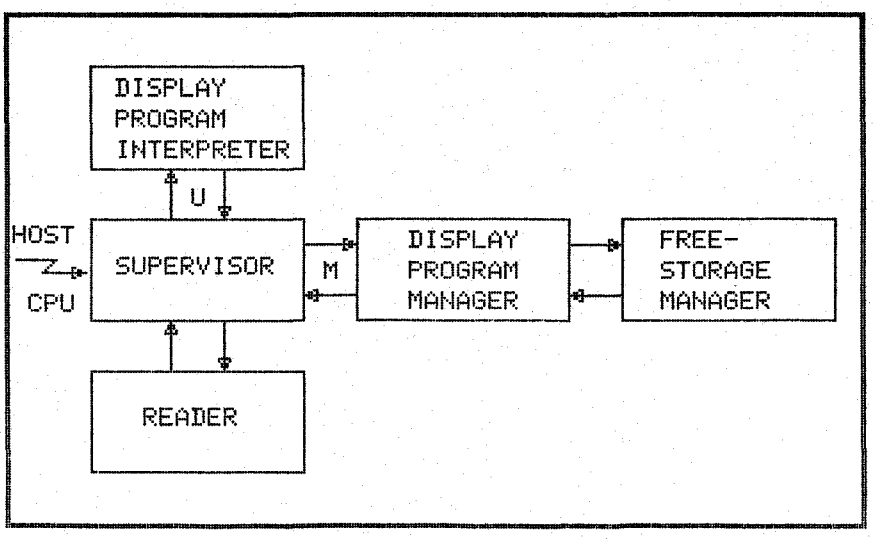

Figure 2.1 : The four modules which make up the communications process

2.1 The Supervisor

This routine controls the overall operation of the KIM display system. The three principle functions of the supervisor are

(a) to initialize the video display system for operation,

(b) to receive and initiate execution of commands from the host CPU, and

(c) to handle the various error conditions that arise in the system.

2.1.1 Initialization

The system initialization sequence is performed only upon startup of the system and consists of clearing the display and setting various system flags and parameters.

2.1.2 Command Receiving

The second, and most important, function of the Supervisor routine involves communication with the host CPU. This communication is initiated by the host using a control line connected to the interrupt request line (IRQ) of the KIM CPU ([MOS76], p.G-2.) The interrupt is recognized by the supervisor at a single point in the simple polling loop. Use of this interrupt "window" ensures that the interrupt service routine, which inputs a single-character command from the serial input line, is always executed at the same point in the supervisor routine. The command character is then decoded and control is transfered to the appropriate process. The valid command characters and their meanings

are summarized in Table 2.1.

Table 2.1 : The single-character commands recognized by the Supervisor

Command	Effect
C	CLEARs the display
M	Transfers control to the DP MANAGER
U	UPDATES the display by transfering control to the DPI
W	Sets the coordinates of the display rectangle

2.1.3 Error Handling

The third supervisory function is to process the two error conditions which are detected and reported by various system processes. The supervisory action initiated upon report of an error condition consists of activation of an appropriate error-message printing segment. Subsequent execution of the display program interpreter then causes the message to be displayed. Actual error recovery procedures however, are carried out at the point of detection in the relevant process. The error causes and recovery procedures are described in the appropriate sections of the report. The exact configuration of the error segments differs somewhat for the two possible cases. In the case of a "coordinate out of range" error, the message is simply printed out on the display but, for the more serious "display file full" error, the message is alternately printed and erased on the display so that a flashing effect is achieved. Both messages are deactivated and cleared

from the display when a subsequent interpretation of the display program is initiated by the host. An algorithm for the supervisor process is given below.

```
procedure Supervisor ;

     (* Process host-generated interrupts and system-generated errors *)

type Screen_Type = record
                      Xleft,Ybottom,
                      Xright,Ytop : integer
                   end ;

var Command   : char ;
    Screen    : Screen_Type ;
    Error     : integer ;
    Interrupt : boolean ;


begin   (* Supervisor *)
   Initialize ;
   Clear_Display ;
   while true do  (* poll until the host interrupts... *)
      begin
         if Interrupt  then
            begin     (* interrupt service routine *)
               Get_Command (Command) ;
               case Command of
                  C : Clear_Display ;
                  M : Display_Program_Manager ;
                  U : Update_Display ;
                  W : Set_Screen_Limits (Screen)
               end
            end

         if Error > 0  then
               case Error of
                  1 : Coordinate_out_of_range ;
                  2 : Display_File_Full
               end
      end
end ;   (* Supervisor *)
```

2.2 The Reader

The Reader module is responsible for the single task of receiving data blocks transmitted by the host. This module consists of a subroutine which is called by the Supervisor or by the DP Manager whenever they are required to receive data from the host. Synchronization of the data transmission between the host CPU and the KIM is controlled by the display system using a one-bit KIM I/O port to provide a Not-Clear-to-Send ($\overline{CTS}$) signal to the host serial-port Asynchronous Communications Interface Adaptor (ACIA). Data transmission by the host is disabled whenever the Motorola MC6850 ACIA detects a high logic level on the $\overline{CTS}$ line (see [OSB77] for complete details.). The Reader routine enables transmission before inputting an ASCII character using the MCS6530 PROM character receiving routine and disables transmission immediately after the character is obtained. Strict monitoring of the $\overline{CTS}$ signal by the corresponding transmission routine in the host ensures that the Reader is not overrun and eliminates any possiblity of deadlock (the dreaded situation in which both routines wait indefinitely for the other to perform some step in the communication sequence). Figure 2.2 shows the arrangement of the data transmission system.

The communication protocol employs three control characters to inform the Reader of the nature and number of data bytes to be sent. A blank (" ") is used to indicate that the two data bytes which follow, are to be interpreted as the initial storage address for the block of data being transmitted. The carriage return character (ASCII 0D)

precedes each data byte in the block; the data byte is received and stored in the location indicated by the current value of the storage address which is then incremented. The third control character, a period ("."), indicates to the Reader that the complete block has been sent. A two-byte checksum, transmitted following the ".", is used by the communications routines in the KIM and the host to verify that the data transmitted was properly received. If the checksum sent by the host equals the arithmetic sum of all the data bytes received by the Reader, transmission is complete and the Reader acknowledges successful reception by transmitting the ACK character (ASCII 06) to the host via the serial line. If, for some reason, the checksums do not match, the Reader transmits the NAK character (ASCII 15) to the host which then attempts retransmission of the data block.

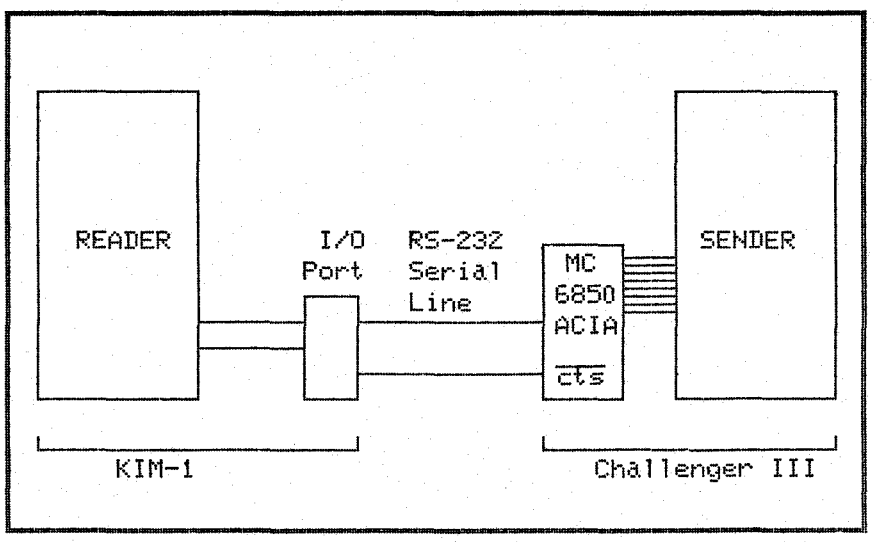

Figure 2.2 : The KIM/C-III communication system

2.3 The Display Program Manager

The display program manager is called by the Supervisor upon receiving the "M" command from the host, and is responsible for maintenance of the display processor program. The DP Manager may affect changes to the two principle data structures used in the video display system. The first of these is the DPP itself and the second is called the segment, or name, table ([NEW79], p.107). The segment table contains information relevant to each display program segment that is defined in the system. Typically, the information stored in the table describes the visibility of the segment and gives is location in the DPP. The following Pascal-like type definition gives the actual implementation structure:

```
type Segment_Record = record
                        STATUS_Byte,
                        Low_Address_Byte,
                        High_Address_Byte : integer
                      end ;

    Segment_Table_Type : array [0..EOT] of Segment_Record ;.
```

Use of the segment table allows both the DP Manager and the DP Interpreter to access a segment whose offset in the table has been given. Although one of the primary reasons for structuring the DPP into segments was to provide the facility for manipulating named portions of a picture, the KIM display system uses only a single integer value, the offset from the origin of the segment table, to designate segments. This is acceptable however and, as pointed out by Giloi [GIL75], it is, in fact, natural to associate the segment names with the application

program in the host since the names are actually meaningless to the display processor. Of course, some care must be taken to ensure that the application program always associates the correct number with a given segment name.

The functions of the DP Manager are:

(a) to change the information in the STATUS byte,

(b) to DELETE segments,

(c) to INSERT segments,

(d) to APPEND to segments, and

(e) to MOVE the location of segments on the display.

The function to be performed is controlled by the host through the use of a four-byte command which is received by the Reader upon entry to the DP Manager. The format of this command is:

byte 1	byte 2	byte 3	byte 4
No. of data bytes to be sent		Segment No.	FUNCTION

2.3.1 Status Changes

The first function that the DP Manager can perform allows the host to control the visibility of a segment on the display. For random-scan display devices, a segment can be rendered invisible simply by removing it from the display refresh cycle. This action, called UNPOSTing ([NEW79], p.95), does not destroy the segment and hence, rePOSTing can render the segment visible again. This ability is particularly useful in avoiding the need for repeatedly redefining

segments that are only temporarily removed from the display. Although the method used to remove segments from a raster-scan display is quite different, the terminology may still be used. Thus, in the KIM video display, the STATUS byte, stored in the segment table, is used to record whether a segment is POSTED or UNPOSTED. As described in Chapter 3, the display program interpreter uses this information to decide if a segment being interpreted should be drawn or erased. The host may therefore affect the visibility of a particular segment by transmitting a four-byte command with the FUNCTION field POST or UNPOST to the DP Manager. The DP Manager then stores this value in the appropriate STATUS byte in the segment table. It should be noted however, that UNPOSTing of a segment on the KIM display system does not automatically remove it from the display; the DPP must be re-interpreted to obtain the exact random-scan display effect.

2.3.2 Segment Deletion

The second function provided by the DP Manager allows the host to delete specified segments from the DPP. Upon receiving a command with the FUNCTION field DELETE, the DP Manager marks the relevant STATUS byte in the segment table with a special value used to indicate to the DPI that the segment should not be interpreted. In addition, the DP Manager calls the Free-storage Manager, described in the next section, to reclaim the space occupied by the deleted segment in the DPP storage area. The deleted segment then, is effectively removed from the system. Such a deleted segment, however, is not necessarily removed from the

display as are deleted segments on a random-scan display. This is due to the fact that segments are not automatically UNPOSTed before deletion takes place.

2.3.3 Segment Insertion

The DP Manager enables the host to add segments to the DPP so that a whole picture can be built up segment by segment. Segment addition is accomplished by transmission of a command giving the number of bytes in the segment, the new segment number and specifying the FUNCTION INSERT. Upon receiving such a command, the DP Manager calls the Free-storage Manager to obtain the starting address of an unused block of DPP storage area memory which is large enough to contain the new segment. The appropriate entry in the segment table is then marked with the STATUS UNPOSTED and starting address equal to that obtained from the Free-storage Manager. This starting address is passed to the Reader which is again called to receive the incoming segment data. This method of handling the incoming DPP data is advantageous because it does not require the use of a, possibly large, intermediate storage buffer to hold the new segment data.

2.3.4 Appending Segments

The three DP Manager functions described above constitute the basic requirements for providing access to the DPP. The following two functions are included primarily for convenience. The first of these, the APPEND function, is provided because "It is sometimes inconvenient

that a [DPP] segment, once it has been defined, cannot be incrementally modified but must always be completely rebuilt." ([NEW79], p.98.) The APPEND capability provided by the DP Manager allows the host to add DPP instructions to an existing segment. After receiving a command specifying the number of bytes to be added, the segment number and the FUNCTION APPEND, the Free-storage Manager is called to allocate a memory block large enough to store the existing segment plus the additional data to be sent by the host. The DP Manager then relocates the existing portion of the segment at the new address, calculates the address of the last byte of existing segment data and passes this to the Reader as the storage location of the incoming DPP data so that the new segment data is attached to the end of the original segment. Finally, the memory block occupied by the original segment data is released for reuse by the Free-storage Manager.

2.3.5 Segment Repositioning

The final capability provided by the DP Manager allows the host to specify the location of a segment on the display. Upon receiving a command specifying the segment number and the FUNCTION MOVESEG, the DP Manager modifies the segment so that the first instruction will cause the display cursor position to be set at the coordinates transmitted by the host. This can be accomplished easily if the first DPP instruction in the segment sets the location of the display cursor. The host-transmitted coordinates can then be placed directly into the DPP as they are received by the Reader. Alternately, the DP Manager must

relocate the segment so that an appropriate cursor positioning instruction can be inserted at the beginning of the segment. This is accomplished by a procedure similar to that used in appending a segment. The new cursor location is then placed in the instruction as it is received by the Reader. Subsequent host-initiated interpretation of the DPP will then cause the figure represented by the segment to appear at the new position.

It is appropriate at this point, to note that modifications to segments in a random-scan refresh display system are significantly more complicated than the corresponding operations on a system employing a raster-scan display. In the former case, the need to ensure uninterrupted display refresh during display file modifications, gives rise to the classic problem of synchronizing two concurrent processes which share some resource. The display processor and the graphics support system or application program, in such cases, share access to the display file which must constantly be maintained in a well-defined state. This gives rise to the need for double buffering and other special techniques ([NEW79], p.103) to avoid corruption of the display file. Such problems, however, do not arise in the case of raster-scan display systems because the display is not refreshed directly from the display processor program.

2.4 The Free-Storage Manager

2.4.1 Introduction

In many computer applications, it is necessary to store and retrieve data consisting of several logically related items. Usually this information is stored in contiguous computer memory locations called blocks, and in many situations, these blocks are continually shortened, lengthened, or deleted by some controlling process. Examples of these processes range from operating systems which need to store whole programs in a time-sharing environment, to a disk directory manager which keeps a record of the contents of a single floppy disk. Under these dynamic conditions it is necessary to have an efficient method for keeping track of the location and status (whether occupied or free for reuse) of the various data blocks. The dynamic Free-Storage Manager implemented for the KIM video display system provides the necessary control over the allocation and reclamation of space in the main DPP memory where the graphics segments are stored.

Before any algorithm for memory management can be implemented, it is necessary to decide upon the data structure to be used for partitioning the available space inside the computer. Typically, the available space is represented as a linked list of memory blocks and, because of the powerful indirect addressing modes of the MCS6502, this method is especially suitable. In such a representation, each free memory block contains a pointer to the beginning of the next free block and a value giving the number of memory words contained in the block.

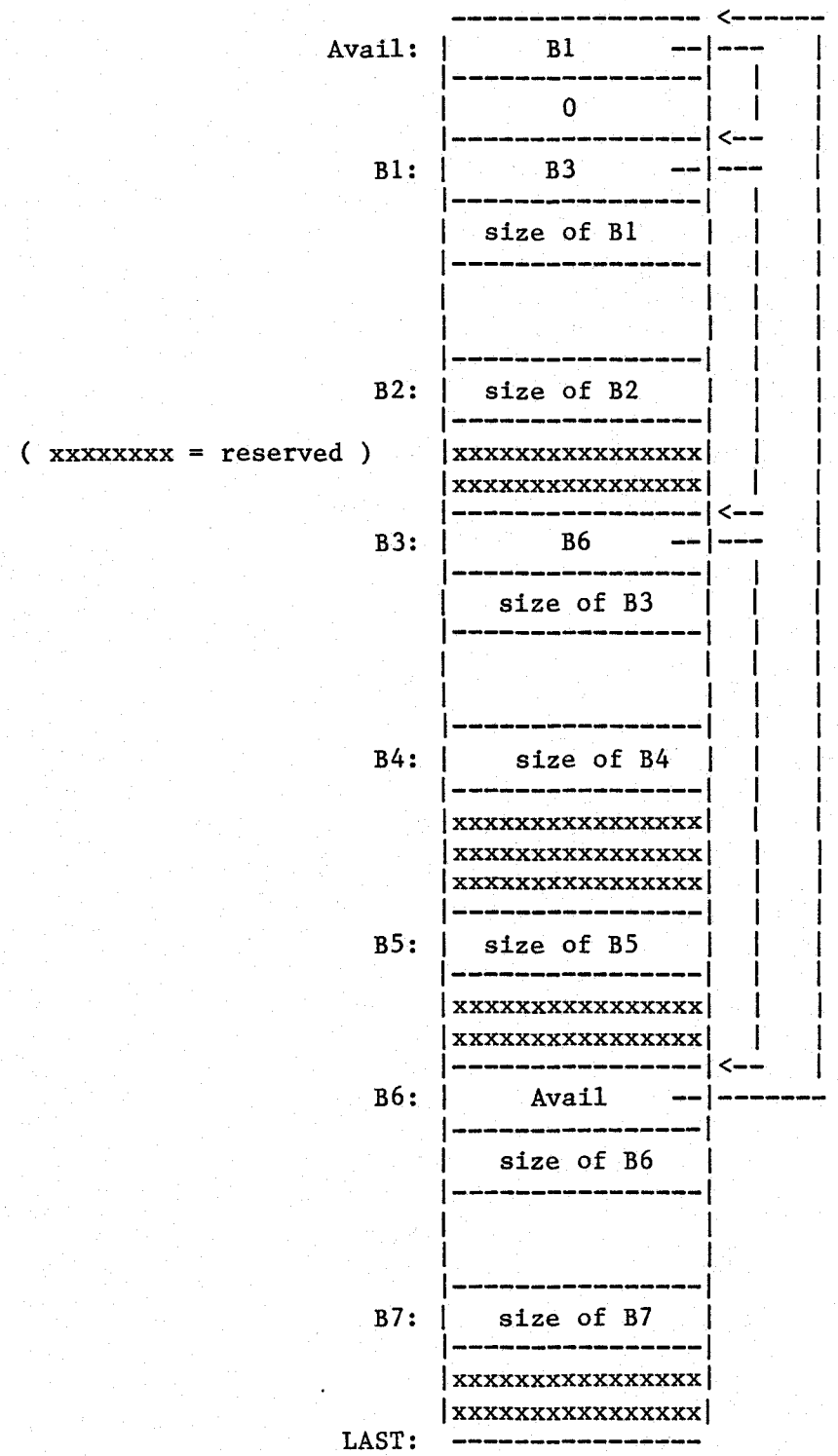

Figure 2.3 : A linked list of unreserved memory blocks

Figure 2.3 shows a situation where the free list contains three unreserved blocks.

2.4.2 Block Allocation

The literature contains many papers dealing with "dynamic storage allocation" algorithms for reserving and freeing variable-size memory blocks from a larger storage area and many such algorithms have been analysed both theoretically and practically (see [KNU68] p.460 et seq. for a bibliography and an excellent account of the topic). Two principal strategies for choosing a memory block from a list of unreserved ones have been described. The first, known as the "best-fit" method, searches the list of free blocks and reserves the smallest one which can store the new block; the second, called "first-fit", simply reserves the first block encountered which is large enough to contain the data. For various reasons, one method may be preferred over the other but, in general, the choice is application dependent. The algorithm incorporated in the video display system employs the latter method and is a modification of one presented by Knuth [KNU68].

A "first-fit" algorithm for reserving a memory block is given below. For the following program, the main memory is considered as a one-dimensional array; each element of the array corresponds to a single word of computer memory.

```
type Memory_Type : array [ Avail..LAST ] of integer ;
var  Memory      : Memory_Type ;
```

```
function Size ( Some_Block : integer ) : integer ;
                (* get the size of some block *)
begin
   Size := Memory [ Some_Block + 2 ]
end ;

function Link ( A_Block : integer ) : integer ;
                (* get the starting address of the next free block *)
begin
   Link := Memory [ A_Block ]
end ;


function Reserve ( N : integer ) : integer ;
           (* find the starting address of a free block of size >= N *)
var P,Eps,K,Start : integer ;

begin
   Start := Q ;                                    (* NOTE: Q is global *)
   P := Link (Q) ;
   while (Size (P) < N) and (P <> Start) do
      begin
         Q:= P ;
         P := Link (Q)
      end ;

   if Size (P) < N
     then  return              (* there is no block large enough *)
     else
        begin
           K := Size (P) - N ;      (* K words will be leftover... *)
           if K < Eps
              then               (* reserve the whole block anyway *)
                 begin
                    K := 0 ;
                    Memory [ Q ] := Link (P)
                 end
              else
                 Memory [ P + 2 ] := K ;          (* Size (P) := K *)

           Reserve := P + K ;
           Memory [ Reserve ] := N ;        (* Size (Reserve) := N *)
           Reserve := Reserve + 2
        end
end ;

(*  NOTE :  In the actual implementation both the Size  and  Link  of  a
            block occupy two words of Memory. *)
```

Figure 2.4 below, shows the initial configuration of the free list and the result of the first block reserve request.

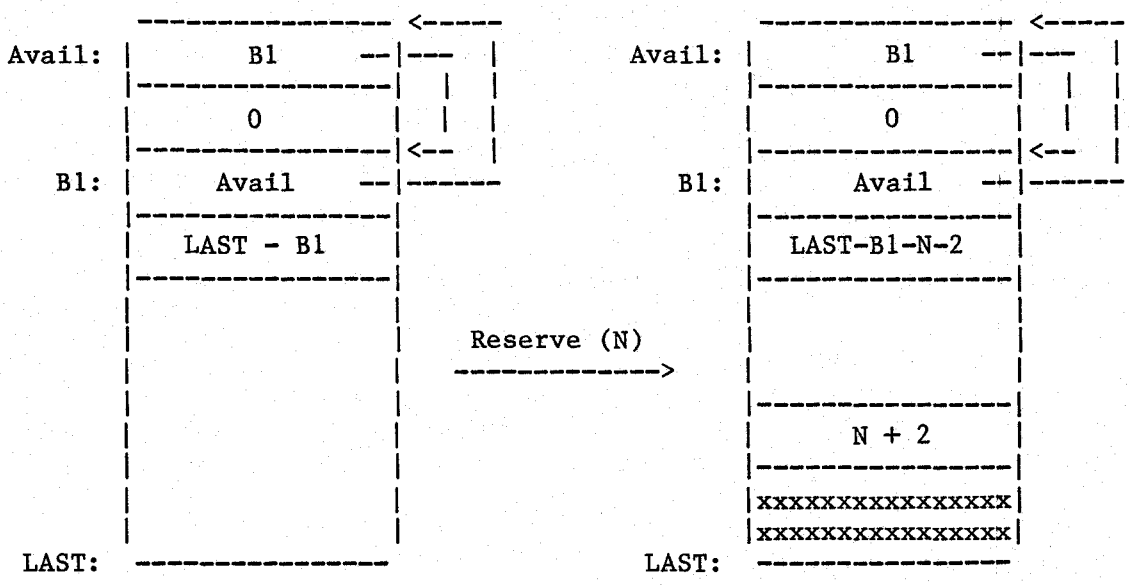

Figure 2.4 : The allocation of the first block

Two modifications to Knuth's basic algorithm have been incorporated in the program given above. Both are suggested in his discussion. The first modification is to implement the free list as a circular linked list. Instead of employing a special "end of list" pointer for the link field of the last free block, it is set to the address of the beginning of the storage area. The search for a large enough free block can then be initiated at any point in the list. (It is for this reason that the variable Q in the algorithm given above, is not initialized upon entry to the routine RESERVE.) This avoids the undesirable accumulation of small blocks at the beginning of the free list and hence decreases the amount of time required to find a suitable

free block. The search for a free block is terminated when a complete scan of the free list has been carried out; in such a case the block reserve request is not satisfied. The second modification is also included to avoid the accumulation of small Sized blocks. Usually, when a large enough free block has been located, the remaining words are added to the free list; this may result in the creation of uselessly small unreserved blocks. The solution is simply to reserve the whole block in such cases. The variable Eps in the program determines the smallest "leftovers" that will be added to the free list as separate blocks.

2.4.3 Block De-allocation

The second half of any dynamic free storage management package must provide the mechanism for de-allocating or liberating memory blocks which are no longer required. Numerous techniques have been devised for performing this vital task and, as in the case of the allocation schemes, all have their own associated advantages and disadvantages. One method, known as "compaction", periodically moves all the reserved blocks into consecutive locations leaving a single, large free block. This method however, becomes slow when the available memory is nearly full. A superior approach, again presented by Knuth, employs ongoing reclamation by amalgamating contiguous free blocks each time a block is released. In this method, free blocks which are adjacent to the one being reclaimed are merged with it into a single free block and the free list is modified accordingly. The liberation algorithm presented below

employs this method.

```
procedure Release ( P0 : integer ) ;
                  (* return the block starting at P0 to the free list *)
var N : integer ;


begin
   P0 := P0 - 2 ;              (* actually the block begins at P0 - 2 *)
   N  := Size (P0) ;

   if P0 < Q
      then                    (* reset Q to the head of the list *)
         Q := Avail ;

   while ( Link (Q) <= P0 ) and ( Link (Q) <> Avail ) do
     Q := Link (Q) ;

   if P0 + N = Link (Q)
      then                          (* the block above P0 is free *)
         begin
            Memory [ P0 ] := Link ( Link (Q) ) ;      (* set Link (P0) *)
            N := N + Size ( Link (Q) )
         end
      else
         Memory [ P0 ] := Link (Q) ;       (* Link (P0) := Link (Q) *)

   if P0 = Q + Size (Q)
      then                         (* the block below P0 is free *)
         begin
            Memory [ Q + 2 ] := Size (Q) + N ;        (* set Size (Q) *)
            Memory [ Q ] := Link (P0)                 (* set Link (Q) *)
         end
      else
         begin
            Memory [ Q ] := P0 ;                   (* Link (Q) := P0 *)
            Memory [ P0 + 2 ] := N                 (* Size (P0) := N *)
         end
end ;
```

A modification of Knuth's original algorithm which decreases the average time required to find the free block immediately preceeding the one being released is incorporated in the Release algorithm. By initiating the search at a block part way along the free list, a

significant portion of the list can be eliminated from the search by a single test. The first conditional statement in the procedure Release determines whether P0 is above or below the starting point Q; if below, the search must begin at the head of the free list; otherwise, it may proceed from the block indicated by the value of Q. Knuth ([KNU68] p.598) gives a short analysis of this modification. Figure 2.5 demonstrates the effect of the algorithm.

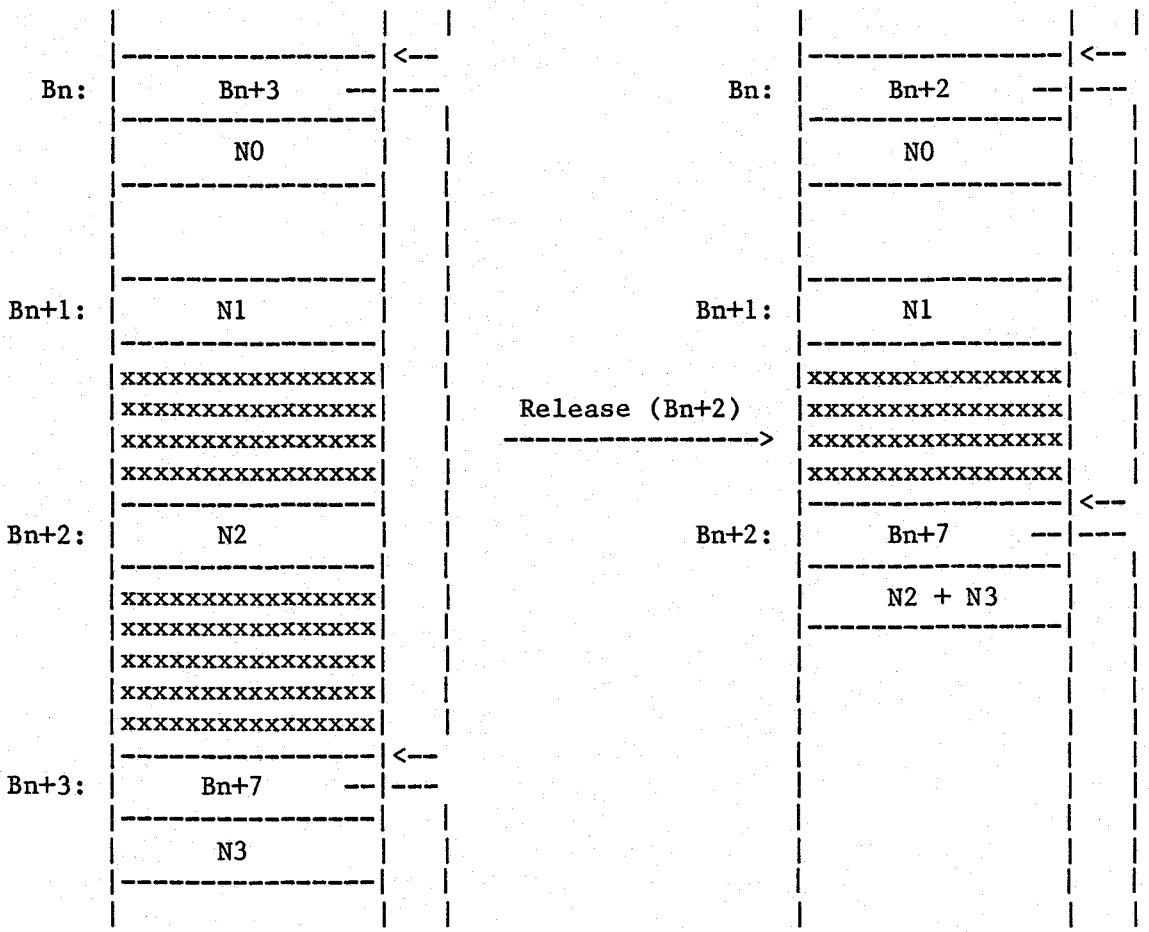

Figure 2.5 : Liberation of a block having a free neighbour above

CHAPTER 3

THE DISPLAY PROGRAM INTERPRETER

Process P2 of Figure 1.2, called the display program interpreter (DPI), scan converts the coded picture representation, otherwise known as the display processor program (DPP), to create the bit map of the picture to be displayed. Interpretation of the DPP is initiated by the Supervisor routine described in Section 2.1, in response to receiving a host-transmitted UPDATE command. Discussion of the DPI begins below, with a description of the instruction set used in the KIM video display system.

3.1 DPI Instructions

There are sixteen DPI instructions, each consisting of a one-byte opcode and a varying number of operands. The simplest instruction, a single-byte NOP having opcode 00, does nothing but increment the DPI Program Counter (PC). This counter is used to maintain the address of the next DPI instruction to be interpreted. Table 3.1 lists the remaining instructions, showing the format and effect of each instruction. These instructions may be divided into three categories which are:

(a) graphic primitive generation,

(b) display program sequence control, and

(c) dynamic segment control.

3.1.1 Primitive Generation

The first category of instructions constitutes a minimal set of graphic primitive generating instructions. Following Newman and Sproull ([NEW79], p.82), the three primitives chosen for the KIM system allow dots, lines or text characters to be drawn by single primitive generating instructions. Any pictures drawn on the display must be represented by sequences of these three instruction types.

Table 3.1 (a) The Primitive Generating Instructions

MNEMONIC	INSTRUCTION					EFFECT
	byte 1	byte 2	byte 3	byte 4	byte 5	
LSETPIX	91	XLOW	XHI	YLOW	YHI	Move cursor to (X,Y) and set pixel ON
LCLRPIX	11	XLOW	XHI	YLOW	YHI	Move cursor to (X,Y) and set pixel OFF
SSETPIX	81	DX	DY	unused		Move cursor by (DX,DY) and set pixel ON
SCLRPIX	01	DX	DY	unused		Move cursor by (DX,DY) and set cursor OFF
LINETO	92	XLOW	XHI	YLOW	YHI	Draw the best straight line to (X,Y)
ERASETO	12	XLOW	XHI	YLOW	YHI	Erase the best straight line to (X,Y)
LINE	82	DX	DY	0	0	Draw the line to (X+DX,Y+DY)
ERASE	02	DX	DY	0	0	Erase the line to (X+DX,Y+DY)
PRINT	84	An ASCII character string ending with EOT (04)				Print the character string

The actual scan conversion of the graphic primitives is accomplished by a set of MCS6502 assembly-language routines adapted from the graphics package supplied by MTU with the VM video board. These

routines allow the user to address the display as a 320x200 array of pixels as shown in Figure 3.1. The transformation for converting the X and Y coordinates into the address of the byte in the frame buffer that holds the pixel intensity value is given by:

BYTE ADDRESS = BASE ADDRESS + (199 - Y) * 40 + X DIV 8. (3-1)

Here, BASE ADDRESS represents the address of the first byte in the frame buffer. The bit number of a pixel is given by:

BIT NUMBER = X MOD 8. (3-2)

All multiplications and divisions for these calculations may be carried out using only sequences of MCS6502 shift and add instructions. This is important because it reduces the time required to execute the, frequently used, conversion routine.

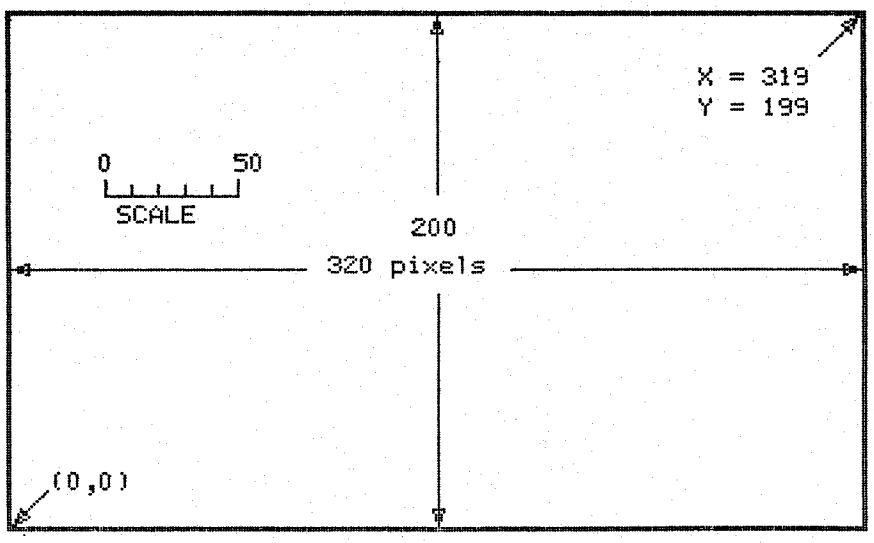

Figure 3.1 : The KIM display coordinate system

The simplest primitive-generating instruction enables the user to control the state of any picture element on the display and to set

the position of the "cursor". (The corresponding primitive for a random-scan display system, suggested by Newman and Sproull ([NEW79], p.82), only enables the user to set the current beam position and does not cause any visible change to the display.) The first form of the pixel setting instruction shown in Table 3.1 (a), specifies the pixel to be affected by giving its X and Y coordinates as a four-byte operand following the opcode. As in the case of the other primitives, one instruction is provided for setting pixels ON and another is provided for setting pixels OFF. An alternative to the absolute, or long, coordinate addressing mode is also provided. In the relative, or short, form of the instruction, pixels are addressed by specifying a two byte operand which gives the displacement of the pixel from the current cursor position. Since only a single byte is used to specify the displacement in each direction, both DX and DY must be in the range:

$-129 < DX \text{ or } DY < 128.$

The second instruction type listed in Table 3.1 (a) provides the basic line drawing facility. This instruction is used to draw a straight line from the current cursor position to a position which may be specified by either relative or absolute display coordinate addressing. Upon completion of the line drawing routine, the current cursor position is set to the endpoint of the line so that the effect is identical to the corresponding instruction executed on a random-scan display system. The short line drawing instruction allows multiple line segments to be drawn using only a single instance of the opcode. This may be done by including the X and Y components of the line segments as

a string of operands following the opcode. The string must be terminated by a zero-length segment. This feature is especially important for reducing the size of the DPP since most pictures will be composed of long sequences of short lines.

The final primitive-generating instruction allows the full ASCII character set to be drawn on the display. The characters specified by the string following the PRINT opcode are drawn on the display with the upper lefthand corner of the first character appearing at the current cursor position. After the text string is drawn, the cursor position is set at the upper lefthand corner of the last character drawn. The characters displayed by the MTU routine are stored as part of the graphics system in the form of a mask raster ([NEW79], p.221). The mask raster is a small (5x7) bit map of the character which is merged with the appropriate region of the frame buffer whenever the character is to be drawn on the display. These characters, however, can only be drawn exactly as they are depicted by the mask raster; no rotation or scaling of characters is possible. Erasure of characters is accomplished by replacing the characters in the string with blanks. This is required, for example, when a segment containing a PRINT instruction is to be UNPOSTED.

In the preceding discussion of the graphic primitive-generating instructions, it has been assumed that the display coordinates specified in the operands of the various instructions are always in the range of the VM display shown in Figure 3.1. This however, may not always be true; there is no reason to expect the user to ensure that only valid

pixels are addressed. The application program running in the host CPU can easily check that the operands of the absolute coordinate addressing instructions are always within the specified range. It is not generally feasible however, to detect in such a manner, out of range coordinates that arise as a result of the interpretation of an instruction which employs the relative coordinate addressing mode. It is therefore preferable that coordinate range checking be performed by the DPI. One method of ensuring that only valid pixels are addressed involves the inclusion of the checking routine at the level of the coordinate to pixel address transformation described by equations (3-1) and (3-2). This however, is not generally acceptable because it increases the time required to execute the conversion routine. A better approach is to include the range checking routines at the level of the primitive-generating instructions. Although the two approaches are equivalent for the simple single pixel setting instructions, a significant saving is achieved in the case of the line drawing instructions. Checking that the endpoints of the line to be drawn are in the required range, ensures that all the pixels making up the line are also in the required range. This method of coordinate checking is implemented in the KIM display system. Detection of a coordinate which exceeds the specified range causes the system error flag to be set to an appropriate value; the DPI then transfers control to the Supervisor process which, as described in section 2.1.3, generates an error message on the display.

The following portion of a display processor program

demonstrates the use of some of the DPI instructions listed in Table 3.1 (a). Interpretation of the program results in the display shown in Figure 3.2

```
LSETPIX (100,100)
LINE (90,0)
LINE (0,90)
LINE (-90,0)
LINE (0,-90)
LINE (0,0)
SCLRPIX (24,70)
PRINT "A Square"
SCLRPIX (-67,-15)
PRINT "with some text"
SCLRPIX (-72,-15)
PRINT "Characters"
SCLRPIX (-48,-15)
PRINT "in it."
LCLRPIX (50,50)
PRINT "A Simple Example of a KIM Display..."
```

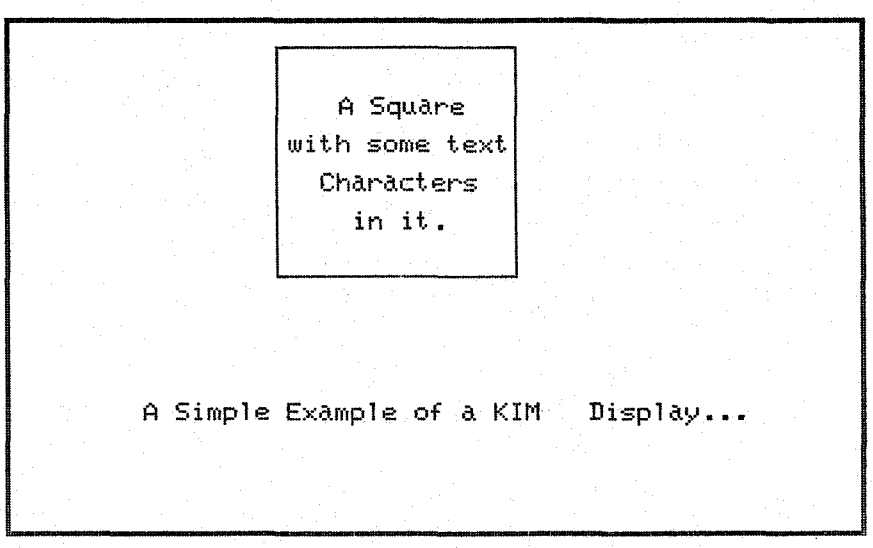

Figure 3.2 : The result of a simple display processor program

3.1.2 Sequence Control

The order in which the DPI instructions are interpreted may be controlled by the user. Primarily, this control is exercised at the segment level of the DPP through the use of the three JUMP instructions listed in Table 3.1 (b). The first of these, the absolute JUMPTO, specifies the number of a segment which is to be subsequently interpreted. Upon encountering this instruction, usually at the end of a segment, the DPI fetches the starting address of the given segment from the segment table. Interpretation of the DPI then continues from the new address.

Table 3.1 (b) : Sequence Control Instructions

MNEMONIC	INSTRUCTION			EFFECT
	byte 1	byte 2	byte 3	
JUMPTO	43	seg No.	unused	Start interpretation at the beginning of the given segment
JUMPSUB	23	seg No.	unused	Push the current segment No. and offset within the segment onto the STACK and begin interpretation of Segment seg#
RETURN	13	00	unused	Pull the segment No and offset and resume interpretation in the calling segment
EXIT	05	KIM	address	Exit DPI to the given address
TEST	07	DX	DY	Test the pixel at (X+DX,Y+DY) skip 2 bytes if pixel is ON

The second JUMP instruction may be considered as a control structure that makes it unnecessary to copy large numbers of identical DPI instructions that occur in more than one place in a DPP. It provides the subprogram structure common to almost all programming

languages ([PRA75], p.148). When the JUMPSUB instruction is encountered during interpretation of a segment, the DPI "pushes" the number of the segment currently being executed onto the software STACK. The offset, which is just the displacement of the JUMPSUB instruction from the beginning of the segment, is also "pushed" onto the STACK. Interpretation then continues at the beginning of the segment indicated in the operand of the JUMPSUB instruction.

The third type of JUMP instruction provides the method for RETURNing from a segment interpreted as a result of a JUMPSUB instruction. Upon encountering a RETURN instruction, the DPI "pulls" the offset and segment number off the STACK. The offset is then added to the address, obtained from the segment table, of the beginning of the segment whose number was retrieved from the STACK. This result is then incremented by two to obtain the address of the DPI instruction following the original JUMPSUB instruction in the calling segment. Interpretation is then resumed at the indicated instruction. The RETURN instruction may also be used as the default segment linking instruction; when no JUMPSUB call is pending, the DPI transfers control to the next valid segment listed in the segment table. It is then possible for the user to ignore the details of the segment linking by simply terminating each segment with a RETURN instruction. An additional advantage derived from this structure is that any segment may be interpreted as a subprogram using a JUMPSUB call. Figure 3.3 shows an example of the various segment linking methods.

The user may also control the point at which the DPI suspends

interpretation of a DPP using the fourth DPI instruction listed in Table 3.1 (b). When the EXIT instruction is encountered during interpretation of a DPP, the DPI transfers control directly to a machine-language routine located at the address given in the operand of the instruction. Typically, this is the address of a user-defined program or that of the Supervisor process. It is not necessary, however, to include such an instruction in every display processor program. Exit from the DPI normally occurs when all segments listed in the segment table have been interpreted. This takes place when one of the JUMP instructions encounters the End-of-Table marker in the STATUS field following the last entry in the table. The DPI then returns control to the point of call in the Supervisor process.

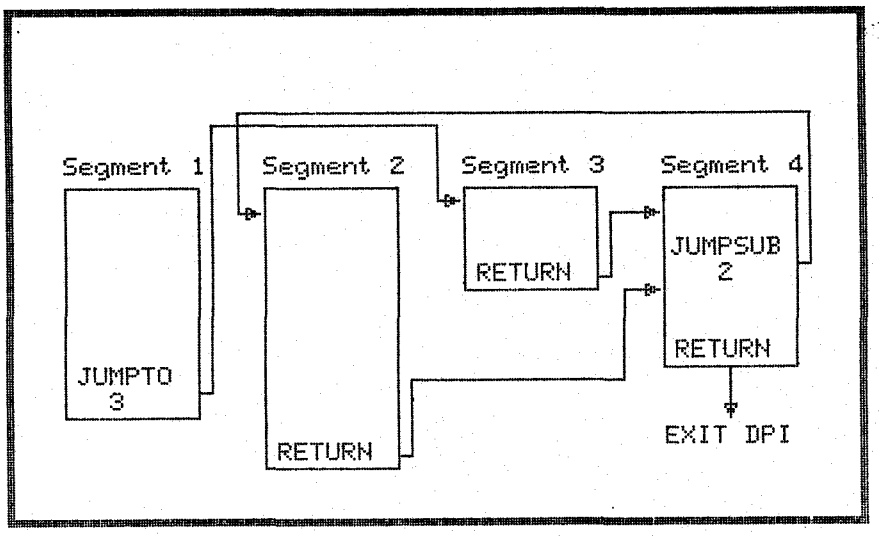

Figure 3.3 : A four segment display processor program

The four sequence control instructions discussed above provide only static control of the interpretation sequence; the order is determined strictly at the time the DPP is created. A small degree of

dynamic sequence control may be achieved through the use of the last DPI instruction listed in Table 3.1 (b). The TEST instruction allows the user to insert a conditional branch in the normal interpretation sequence. Depending on the state of the pixel indicated by the relative coordinate address given as the instruction operand, either zero or one instruction(s) will be skipped by the DPI. By following the TEST instruction with another sequence control instruction, an interpretation sequence that depends on the current state of the display can be constructed. This novel feature of the KIM display system is useful for example, in controlling dynamic displays without intervention by the host CPU.

3.1.3 Dynamic Segment Control

The last instruction type, listed in Table 3.1 (c), is included in an effort to enable the creation of simple animated displays. In other graphical display systems, animation must be controlled by an application program running in the host CPU. The use of the relative coordinate addressing mode and the SETDFL instruction however, allow the creation of segments which will automatically reposition themselves each time they are interpreted by the DPI. The SETDFL instruction, when included in a segment, causes the DYNAMIC system flag to be set. A set DYNAMIC flag is used to indicate to the Supervisor process that an automatic UPDATE of the display should be initiated. The result is that, as long as the dynamic segment remains POSTED, the DPI continually interprets the DPP and, because the initial cursor coordinates are

different for each interpretation, the segment will appear at a different position on the display each time.

Table 3.1 (c) The Dynamic Segment Control Instruction

MNEMONIC	INSTRUCTION	EFFECT
	byte 1	
SETDFL	06	Set the DYNAMIC system flag

3.2 The UPDATE Function

In section 2.1.2, the single-character commands transmitted to the KIM display system were described. One of these, the "U" character command, is included in order to reduce the time spent regenerating the display. For a raster-scan system to completely simulate the operation of a random-scan based system, the DPP must be re-interpreted each time a change is made to it. In most situations however, it is quite acceptable for the system to receive a batch of modifications to the DPP before regenerating the display and, since the user is best able to judge the acceptable number of modifications, a function to allow the user to request the system to UPDATE the display is included. An improvement, designed to further reduce the time required to regenerate the display, makes use of the fact that modifications to the display are usually minor and hence regeneration of the whole display is unnecessary. In many cases, only segments which have been modified since the previous UPDATE need to be re-interpreted. For this reason, an additional attribute, associated with each segment in the KIM system,

is used to record the relationship between a DPP segment and its appearance on the display. This single-bit flag, stored along with the POSTED/UNPOSTED flag in the STATUS byte of each entry in the segment table, allows the DPI to test whether each segment is PAINTED or UNPAINTED. Then, during interpretation, the following algorithm may be used to determine whether or not re-interpretation of a segment is required.

if (Segment is POSTED) **and** (Segment is PAINTED)
then No interpretation is required ; Segment is visible ;

if (Segment is UNPOSTED) **and** (Segment is UNPAINTED)
then No interpretation is required ; Segment is invisible ;

if (Segment is POSTED) **and** (Segment is UNPAINTED)
then Draw the segment , mark Segment PAINTED ;

if (Segment is UNPOSTED) **and** (Segment is PAINTED)
then Erase the segment , mark Segment UNPAINTED

The algorithm given above is incorporated in the JUMPTO instruction described in section 3.1.2 but not in the other JUMP instructions; the JUMPSUB instruction causes a segment to be interpreted whether it is PAINTED or UNPAINTED so that subprogram segments may be interpreted more than once during a single UPDATE. The DP Manager is responsible for making the appropriate changes to the segment STATUS whenever modifications to a segment are carried out; the relevant STATUS byte is marked UNPAINTED when a segment is inserted, appended or repositioned.

For the first two possibilities checked by the above algorithm, no interpretation of the segment is required and control is transferred to another segment. However, because of the variety of possible segment

linking mechanisms, it is not generally known which segment should be subsequently interpreted. A simple solution to this problem is to execute the DPI instruction which terminates the segment but, since the length of each segment is not known, a more easily located copy of the instruction is placed at the head of the segment when the DPP is created. This instruction must be one of the two-byte JUMP instructions listed in Table 3.1 (b). A segment may then be omitted from the interpretation simply by executing the DPI instruction occupying the first two bytes of the segment; if the segment is to be interpreted, this instruction is skipped. A similar segment linking structure is discussed by Newman and Sproull ([NEW79], p.107).

A second feature of the UPDATE function is included to solve the problems caused when a segment is erased. Whenever a segment is UNPOSTED, the subsequent interpretation of the DPI may cause the undesired erasure of other portions of the display. The simplest solution to this problem is to redraw any POSTED segments. During interpretation, the DPI records whether any segment has been erased so that, at the end of interpretation, any "holes" created can be redrawn. After the initial intepretation, all POSTED segments are marked UNPAINTED and the DPI is re-interpreted.

3.3 The DPI Algorithm

The algorithm embodied in the DPI can be expressed by the following Pascal-like program:

```
procedure Display_Program_Interpreter ;
type DPP_type = array [ First..Last ] of Memory ;
var PC,Opcode,Segment : integer ;
    DPP               : DPP_type ;
    Seg_Table         : Segment_Table_Type ;

procedure Decode ( No_of_Bytes : integer ) ;
begin  (* Decode *)
   PC := PC + No of_Bytes ;
   Opcode := DPP [ PC ] ;
   case Opcode of
      NOP    : Decode (1) ;
      POINT  : Setpoint ;   (* Graphic Primitive Generators *)
      VECTOR : Drawline ;
      PRINT  : Draw_Characters ;
      JUMP   : Next_Segment ;        (* Sequence Controlers *)
      EXIT   : goto DPP [ PC + 1 ] ;
      TEST   : Read_Pixel ;
      SETDFL : Set_Flag        (* Dynamic Segment Indicator *)
   end
end  (* Decode *) ;

begin  (* DPI *)
   Segment := 0 ;
   Next_Segment ;
   while Seg_Table [ Segment ].STATUS_Byte <> End_of_Table do ;
      Decode ( No_of_Bytes ) ;

   if ( any segment was erased )
      then                (* fill any "holes" in the display *)
         begin
            Segment := 1 ;
            while Seg_Table [ Segment ].STATUS_Byte <> End_of_Table do
               begin
                  if Seg_Table [ Segment ].STATUS_Byte = POSTED
                     then
                        Seg_Table [ Segment ].STATUS_Byte := UNPAINTED ;
                  Segment := Segment + 1
               end ;
            Display_Program_Interpreter
         end
end  (* DPI *)
```

3.4 Line Clipping

In section 3.1.1, the method used for handling display coordinate addressing overflow was described. Treating such overflow situations as errors is, in general, overly restrictive. It may be desirable for example, to display only a certain rectangular region of the complete picture represented by the DPP. This technique of viewing a picture requires that only pixels which lie inside the viewing rectangle be displayed. The rectangle itself may coincide with the actual display coordinate limits but, more generally, it may also be defined as some smaller sub-range of the total display coordinate system. In such a general case, the portions of the picture which lie outside the rectangle must not be displayed even though they are represented by valid display coordinates. The process of ensuring that only portions of the picture that lie inside the rectangle are displayed, is known as clipping. Numerous algorithms for clipping various graphic entities have been devised ([NEW79], pp.65-71 and [GIL78], pp.86-90), but most of these are concerned with the clipping of straight lines since other commonly used graphic entities, including text characters and curves, can be constructed from a series of straight line segments. The line clipping algorithm implemented on the KIM display system is a well-known variation of an algorithm devised by Cohen and Sutherland (see [NEW79], pp.65-68 for a detailed description of both algorithms.). It employs the method of mid-point subdivision, chosen because it uses no calculations requiring multiplication or division. A Pascal-like program for the algorithm is given below:

```
procedure Clip_Line ( var P0,P1 : Point ) ;
(*
   Clip the line segment P0-P1; Draw any portion of the line segment
   inside the rectilinearly oriented clipping rectangle defined by:
        ( Box.Xleft,Box.Ybottom ) and ( Box.Xright,Box.Ytop )
*)
type Point = record
                X,Y : integer
             end ;
     Edge = (Left,Right,Bottom,Top) ;
     C-S_Code_Type = set of Edge ;


procedure Code (A_Point : Point ; var C-S_Code : C-S_Code_Type) ;
(*
   Determine the Cohen-Sutherland code for the given point
   The value of the code for each region surrounding the screen is:

            [ Left,Top ]   |    [ Top ]    |  [ Right,Top ]
           ----------------|---------------|----------------
              [ Left ]     |      [ ]      |    [ Right ]
           ----------------|---------------|-----------------
           [ Left,Bottom ] |  [ Bottom ]   | [ Right,Bottom ]
*)
begin
   C-S_Code := [] ;
   if A_Point.X < Box.Xleft
      then
         C-S_Code := [ Left ]
      else
         if A_Point.X > Box.Xright then C-S_Code := [ Right ] ;

   if A_Point.Y < Box.Ybottom
      then
         C-S_Code := C-S_Code + [ Bottom ]
      else
         if A_Point.Y > Box.Ytop then C-S_Code := C-S_Code + [ Top ]
end ;

function Rejected ( Pa,Pb : Point ) : boolean ;

var C1,C2 : C-S_Code_Type ;

begin                              (* A line can be trivially rejected *)
   Code ( Pa,C1 ) ;                    (* if the logical intersection *)
   Code ( Pb,C2 ) ;                (* of the C-Scodes for the endpoints *)
   Rejected := ( C1 * C2 ) <> []                        (* is not empty *)
end ;
```

```
procedure Get_EndPoint ( From,To : Point ; var EndPoint : Point ) ;

(*
      Find the visible point on the line segment From-To
      which is farthest from the endpoint From.
      The method of repeated midpoint subdivision is used.
*)
var DX,DY : integer ;
    Done  : boolean ;
    Mid   : Point ;
    C-S_Code : C-S_Code_Type ;

begin
   DX := To.X - From.X ;
   DY := To.Y - From.Y ;
   Code ( To,C-S_Code ) ;
   Done := false ;
   if C-S_Code =  []
      then
         EndPoint := To
      else
         while not ( Rejected ( From,To ) or Done ) do
            begin          (* Divide the line in half *)
               DX := DX/2 ;   (* Just shift 1 bit to the right *)
               DY := DY/2 ;
               Mid.X := From.X + DX ;
               Mid.Y := From.Y + DY ;
               Done := (DX = 0) and (DY = 0) ;
               if not Done
                  then
                     if Rejected ( Mid,To )
                        then
                           To := Mid
                        else
                           From := Mid ;
               EndPoint := From
            end
end ;



begin   (* Clip_Line *)
   Get_EndPoint ( P0,P1,P1 ) ;
   Get_EndPoint ( P1,P0,P0 ) ;
   if not Rejected ( P0,P1 ) then Drawline
end ;   (* Clip_Line *)
```

In the KIM system, the clipping rectangle may be specified by the host CPU using the single-character "W" command described in section 2.1.2. Provision for disabling the clipping facility is also included. The effect of the line clipping algorithm can be seen from Figure 3.4 below.

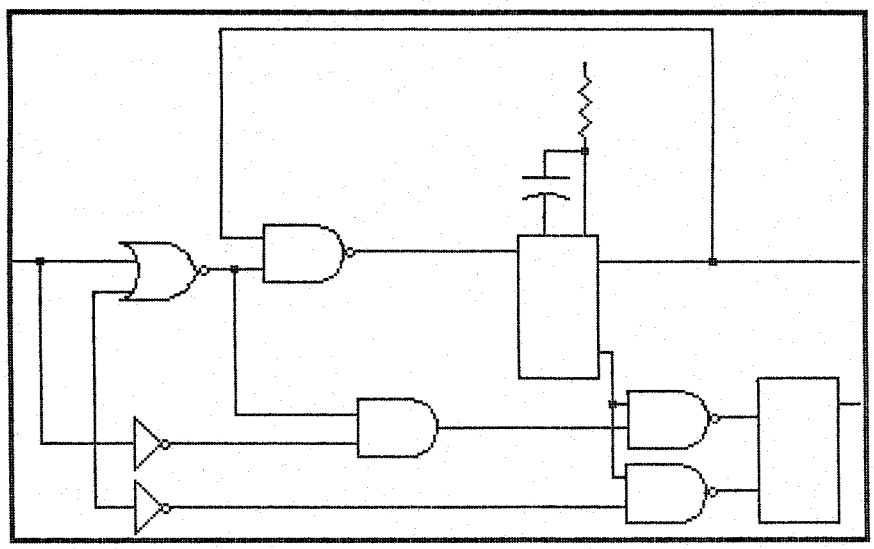

Figure 3.4 (a) : A display produced without clipping

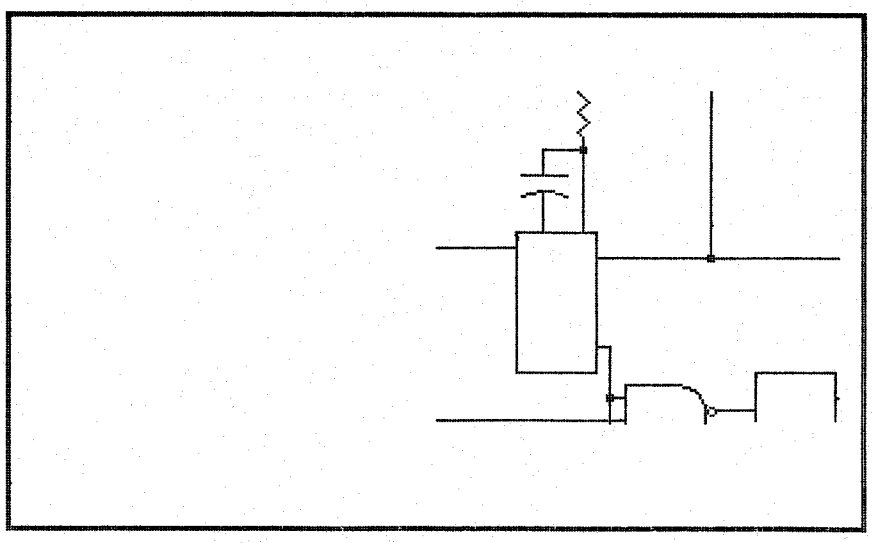

Figure 3.4 (b) : A display produced with clipping enabled

CHAPTER 4

TESTING

Testing may be defined as a set of procedures and activities intended to demonstrate the correct operation of a program within its intended environment [HET73] [YOU75]. Validation of the KIM display system software involved the development of a complete driver package for the OSI Challenger III (C-III) host microcomputer. Using the package, graphical displays may be produced on the KIM display interactively. Although primarily designed as a method of testing the graphics system, the C-III package proved to be usefull for creating simple diagrams such as the ones included in this report. Despite the fact that testing was carried out at each stage of development of the KIM system, it is well to remember that, as Dijkstra points out, testing shows the presence, not the absence of bugs [DIJ70].

4.1 The Host Driver Package

For the purposes of discussion, the package implemented on the host microcomputer can be divided into four parts which are responsible for the different aspects of display generation and manipulation. The first of these parts is used to decode the user input commands, a second process is dedicated to communication with the display system and the third, a multiple-byte arithmetic package adapted from one produced by N. Solntseff, is used to convert user input decimal display coordinates

to hexadecimal values for the display system. The fourth portion of the host system consists of a group of subroutines which execute the various commands. These commands may also be divided into four different categories. The first command type, listed in Table 4.1 (a), is concerned with generation of the DPI instruction codes.

Table 4.1 (a) Commands for Generating the DPI Codes

Command	Effect
MOVE (DX,DY)	Add SSETPIX, DX, DY to the open segment
MOVETO (X,Y)	Add LSETPIX,X,Y to the open segment
LINE (DX,DY)	Add LINE,DX,DY to the open segment
LINETO (X,Y)	Add LINETO,X,Y to the open segment
PRINT "String"	Add PRINT,"String" to the open segment
JUMPTO (Segname)	Add JUMPTO,Segnumber (Segname) to the segment
JMPSUB (Segname)	Add JUMPSUB,Segnumber (Segname) to the segment
RETURN	Add RETURN to the open segment
EXIT (hexadress)	Add EXIT,hexadress to the open segment
TEST (DX,DY)	Add TEST,DX,DY to the open segment
SETDFL (flag)	Add SETDFL,flag = 0 or 1 to the open segment

The second type of command recognized by the C-III system package includes those for segment manipulation. Segments, like disk files, must be OPENed before information can be added and must be CLOSEd when the last piece of data has been added. The OPEN function simply adds the new segment name to a list, assigns a new integer identifier to

the segment and initializes a transmission buffer used to store segment data which is to be sent to the display system when the segment is CLOSEd. Since only a single segment may be OPEN at a given time, it is not necessary to specify which segment is to be CLOSEd. The full set of commands in this category is listed, in Table 4.1 (b). All but the first of these commands transmit the command character "M" to the display system Supervisor process to indicate that the display program manager will be required.

Table 4.1 (b) Segment Manipulating Commands

Command	Effect
OPEN (Seg name)	Initialize a new segment; add Seg name to the name list
CLOSE	Transmit the segment data stored in the buffer to the KIM Display Program Manager
POST (Seg name)	Send POST,Segnumber (Seg name) to the DP Manager
UNPOST (Seg name)	Send UNPOST, Segnumber (Seg name) to KIM
DELETE (Seg name)	Send DELETE, Segnumber (Seg name) to KIM
APPEND (Seg name)	Send APPEND, Segnumber (Seg name) and reOPEN seg
PUTSEG (Seg name)	Read new coordinates; Send PUTSEG and data to KIM

The third command category consists of those commands which transmit single character directives to the KIM Supervisor process described in section 2.1.1. They are listed in Table 4.1 (c) below.

Table 4.1 (c) Commands to the KIM Supervisor

Command	Effect
CLEAR	Transmit "C" to have the display cleared
FRAME	Transmit "W"; Read and transmit new display limits
UPDATE	Transmit "U" to have DPP re-interpreted

The final type of command has no effect on the display system; it includes commands to the C-III system itself. These commands are listed in Table 4.1 (d).

Table 4.1 (d) C-III System Internal Commands

Command	Effect
WIPE	Initialize the C-III system
SCRAP	Scrap the segment being created; delete its name from the segment list; transmits no segment data
INTENS=	Set PEN ON or OFF; Sets drawing colour for subsequent DPI instructions which are added to a segment
QUIT	Exit from the C-III system to OS65D operating system

4.2 Using the Display System: A Sample User Session

Most of the figures shown in this report were generated interactively using the KIM video display system itself. To illustrate the methods used, a detailed example is included here.

4.2.1 System Loading and Startup

To use the display system and testing routines described in Section 4.1, it is first necessary to load the KIM microcomputer with the graphics system. This is accomplished by a program in the host computer which communicates with the KIM firmware via the RS232 line. A program similar to the Reader described in Section 2.2 is initially downloaded from the host to KIM. This KIM-resident routine is then used as a relatively high speed receiver for loading the complete graphics system. At the end of this process, control of the KIM system is transferred to the Supervisor described in section 2.1. In the C-III system, the testing routines are then loaded and the system is initialized. Execution of the entire startup sequence has been simplified using a "command file" which stores the complete list of required commands. This command file is stored on a single floppy disk along with the set of C-III and KIM MCS6502 assembly-language routines which make up the graphics system. The two OS 65D commands given below are used to initiate the startup sequence.

```
A*ca 4000=16,2
A*go 4000
(DO YOU WANT LOOPING?)
NO
(ENTER COMMANDS?)
NO
```

The system will then load the required programs from the floppy disk attached to the C-III host and transmit them to the KIM. At the end of the downloading sequence, the display shown in Figure 4.1 will appear on the KIM video monitor.

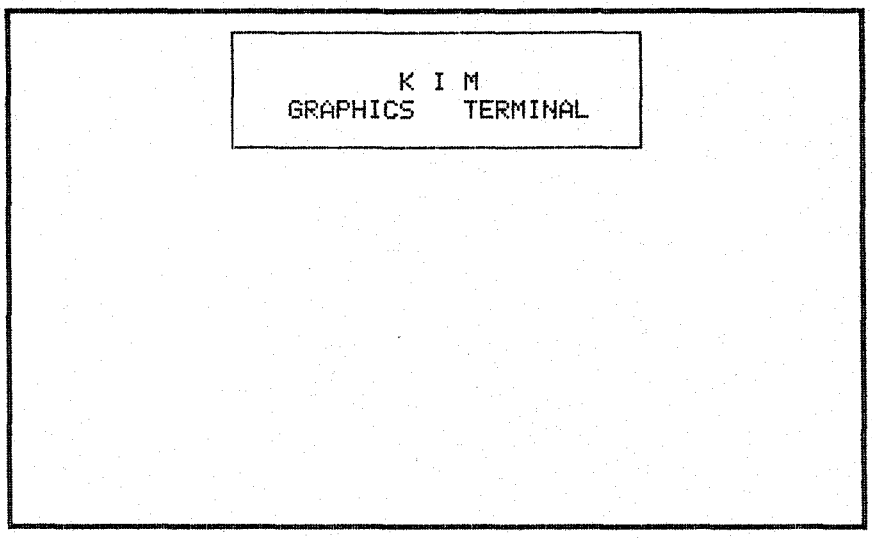

Figure 4.1 : The initial KIM system display

In the host system, control must then be transferred to the command reading/interpreting routine of the testing package by entering the OS 65D command:

A*go 68B1.

The system then prompts the user with the character "/", whenever it is ready to receive commands. The sequence

```
/WIPE
/INTENS=ON
/UPDATE
```

is then used to initialize the testing routines, set the drawing "colour" and to clear the video display. The user may then begin to define the first segment.

4.2.2 A Simple Segment

The first display generated for this example is a simple one,

requiring only a single segment. It is defined using the following sequence:

```
/OPEN (CUBE)
/MOVETO (85,60)
/LINE (70,0)
/LINE (0,70)
/LINE (-70,0)
/LINE (0,-70)
/LINE (35,35)
/LINE (70,0)
/LINE (0,70)
/LINE (-70,0)
/LINE (0,-70)
/CLOSE.
```

At this point, the block of segment data is transmitted to the KIM. However nothing will appear on the display until the segment is POSTed and the display file is interpreted:

```
/POST (CUBE)
/UPDATE.
```

The resulting display is shown below.

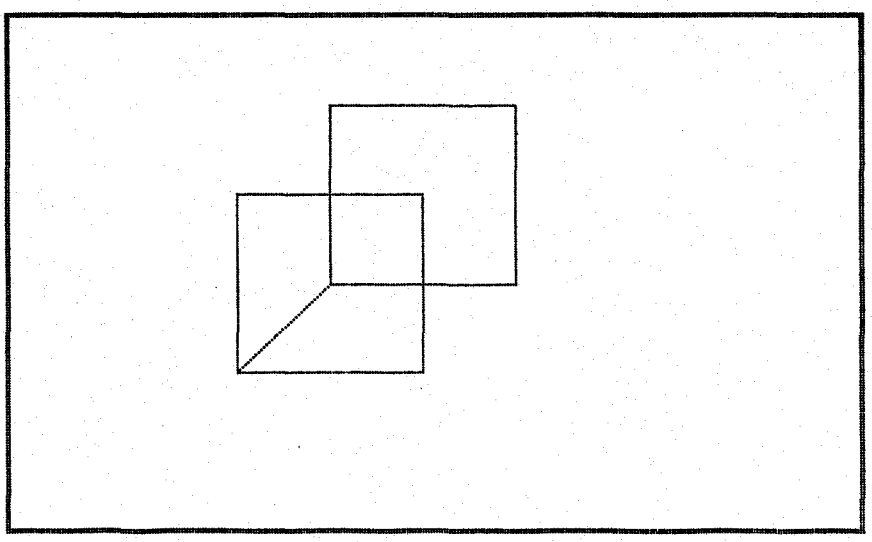

Figure 4.2 : The first segment

To complete the drawing of the cube, the APPEND function is used:

```
/APPEND (CUBE)
/MOVE (-35,35)
/LINE (35,35)
/MOVE (70,0)
/LINE (-35,-35)
/MOVE (0,-70)
/LINE (35,35)
/POST (CUBE)
/UPDATE.
```

The resulting display is reproduced below.

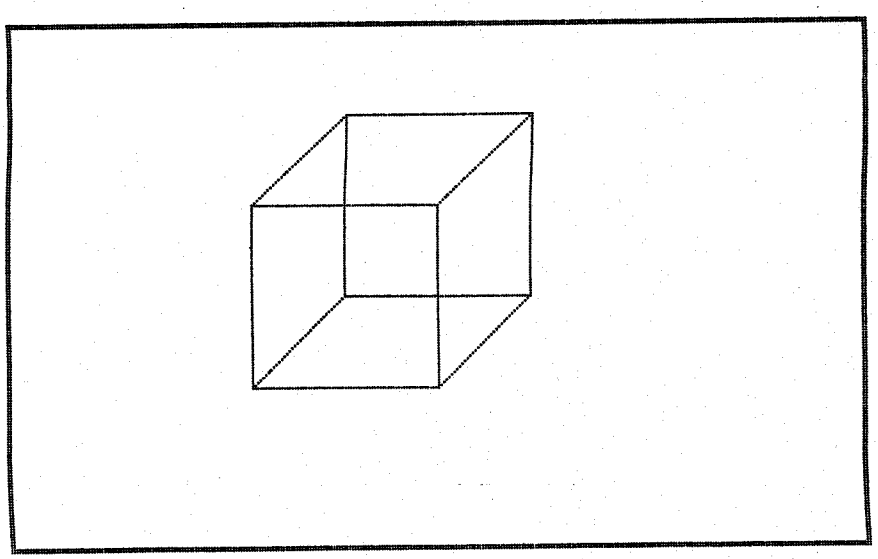

Figure 4.3 : The first segment after being APPENDed

It should be noted that POSTing a segment also caused the currently OPEN segment to be CLOSEd so that a possible source of user error is removed ([NEW79], p.79). The ability to incrementally build up segments using the APPEND function is important because it allows the user to monitor the progress of the segment definition. This is especially

useful during the creation of long, complicated segments. An additional point to note is that CUBE contains only a single DPI instruction which uses the absolute cursor addressing mode. This allows the segment repositioning instruction, described in Section 2.3.5, to be used. To reposition CUBE, the sequence:

```
/UNPOST (CUBE)
/UPDATE
```

is first used to erase the segment from the current position and

```
/PUTSEG (CUBE)
 where ?
- (245,100)
/POST (CUBE)
/UPDATE
```

causes the segment to be redrawn at the new position. Figure 4.4 shows the display which is produced.

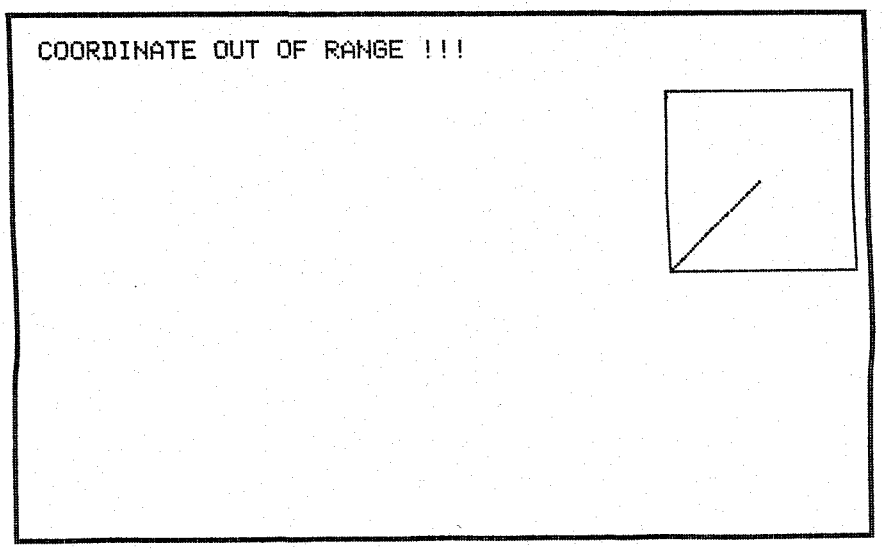

Figure 4.4 : An error message

Since drawing the sixth line segment would result in a horizontal

coordinate greater than 319, an error condition is reported. As described in Section 3.4, the command

```
/FRAME (1)
 NEW WINDOW ?
Y- (0,0)
 - (319,199)
```

may then be used to enable the line clipping mode and to set the clipping rectangle coordinates equal to the display limits. The command sequence:

```
/POST (CUBE)
/UPDATE
```

will then cause the error message to be erased and CUBE will be redrawn to produce the display shown in Figure 4.5. This display is a much better reflection of the user's probable intentions than the overly protective response demonstrated by Figure 4.4.

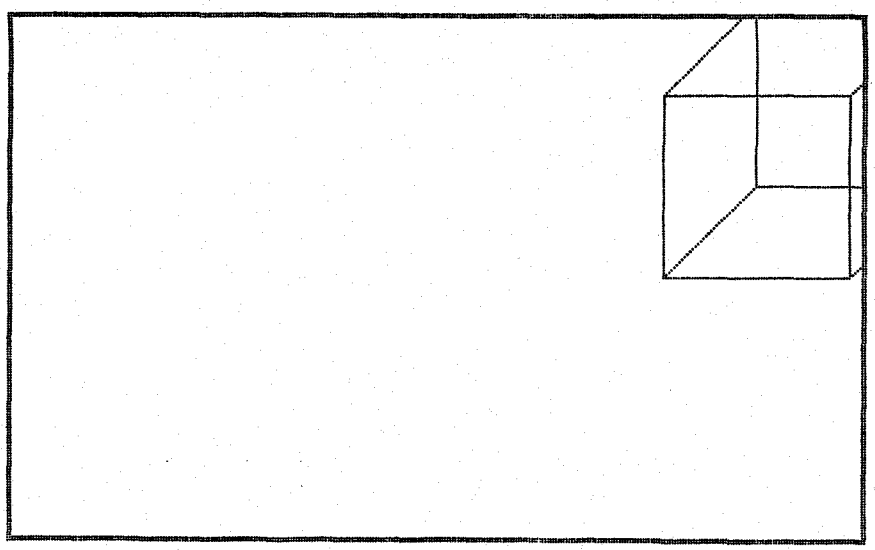

Figure 4.5 : A segment clipped to fit inside the display

4.2.3 The Use of Subroutine Segments

In preparation for the second example, the sequence:

```
/CLEAR
/DELETE (CUBE)
```

is used to erase the display and to remove the single segment from the system. The following example is chosen to demonstrate the use of subroutine segments for structuring the picture definition as well as to suggest a possible application of the system. It gives the details of one approach to the problem of producing diagrams of binary tree data structures. Four subroutine segments are defined initially:

```
/OPEN (NODE)
/LINE (10,0)
/LINE (4,4)
/LINE (0,5)
/LINE (-4,4)
/LINE (-10,0)
/LINE (-4,-4)
/LINE (0,-5)
/LINE (4,-4)
/CLOSE
```

```
/OPEN (LEAF)
/LINE (4,0)
/LINE (0,4)
/LINE (-4,0)
/LINE (0,-4)
/CLOSE.
```

An example of the use of these two subroutine segments is shown in Figure 4.6. These examples are drawn by another segment which, in addition to invoking the subroutine segment, prints the name and location of the subroutine segment.

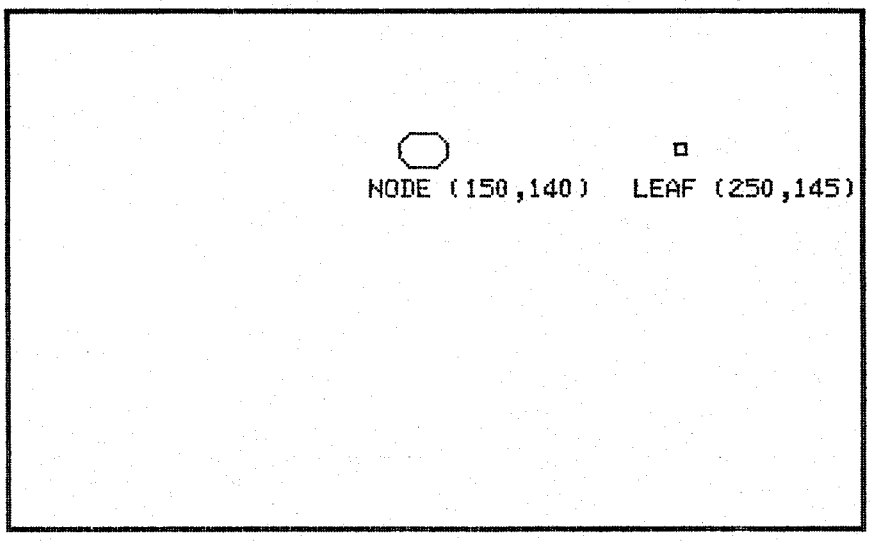

Figure 4.6 : Some simple subroutine segments

The remaining two subroutine segments are created by the sequence:

```
/OPEN (UNLEAF)
/INTENS=OFF
/LINE (4,0)
/LINE (0,4)
/LINE (-4,0)
/LINE (0,-4)
/CLOSE

/INTENS=ON
/OPEN (SON)
/JMPSUB (NODE)
/LINE (-8,-16)
/MOVE (-2,-5)
/JMPSUB (LEAF)
/MOVE (20,21)
/LINE (8,-16)
/MOVE (-2,-5)
/JMPSUB (LEAF)
/CLOSE.
```

It should be noted that the INTENS command can be used whether or not a segment is currently OPEN.

A segment for drawing the root node of a tree can easily be

created using the following sequence:

```
/OPEN (TREE)
/MOVETO (145,185)
/PRINT "47"
/MOVE (-12,-9)
/JMPSUB (SON)
/POST (TREE)
/UPDATE.
```

The resulting display is shown below.

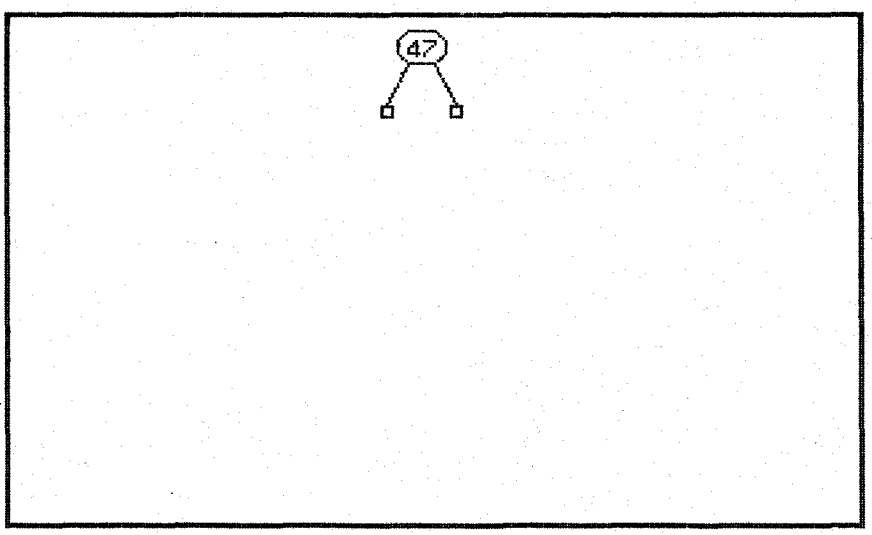

Figure 4.7 : The root node of a binary TREE

Descendants may then be added using the APPEND function:

```
/APPEND (TREE)
/MOVE (-26,0)
/JMPSUB (UNLEAF)
/MOVE (-5,0)
/PRINT "39"
/MOVE (-12,-9)
/JMPSUB (SON)
/CLOSE
/UPDATE.
```

Other levels are added in a similar manner. Addition of elements to the right hand side of TREE may be accomplished in a slightly different

manner:

```
/OPEN (RSON)
/MOVETO (180,140)
/JMPSUB (UNLEAF)
/PRINT "63"
/MOVE (-12,-9)
/JMPSUB (SON)
/JMPSUB (UNLEAF)
/PRINT "74"
/MOVE (-12,-9)
/JMPSUB (SON)
/JMPSUB (UNLEAF)
/PRINT "99"
/MOVE (-12,-9)
/JMPSUB (SON)
/MOVE (-26,0)
JMPSUB (UNLEAF)
/MOVE (-5,0)
/PRINT "13"
/MOVE (-12,-9)
/JMPSUB (SON)
/POST (RSON)
/UPDATE.
```

The display shown in Figure 4.8 is then produced.

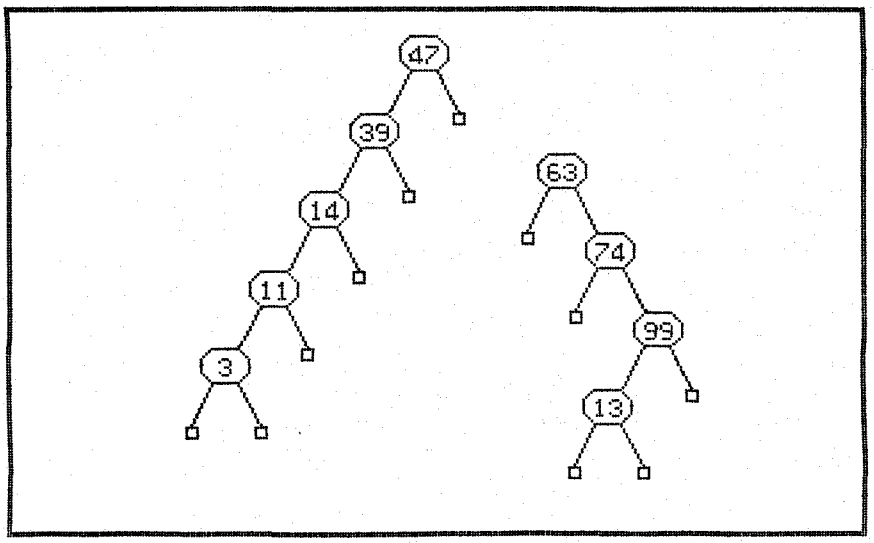

Figure 4.8 : A TREE and its right son

The RSON segment can be positioned correctly after several trials, using the PUTSEG command:

```
/PUTSEG (RSON)
 where ?
 - (166,156)
/CLEAR
/UPDATE.
```

The complete TREE diagram is shown in Figure 4.8.

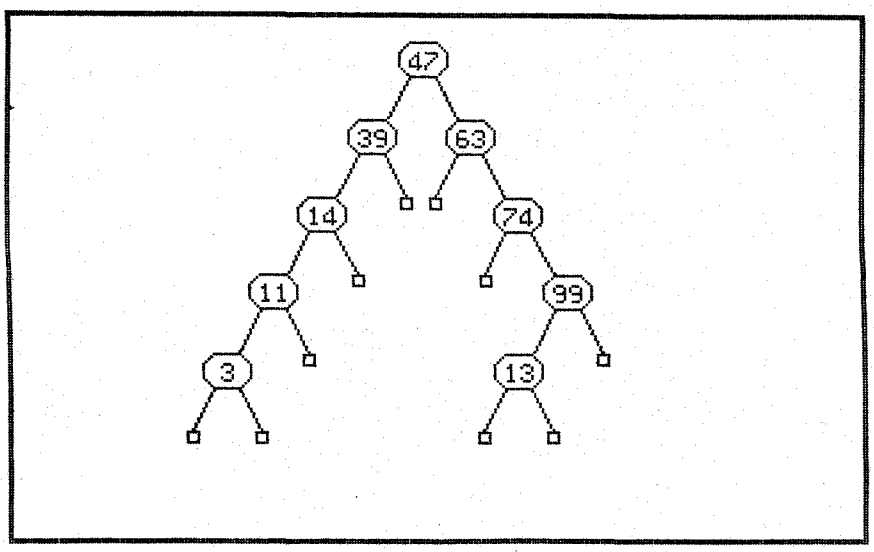

Figure 4.9 : A binary TREE

To terminate the session, the command

```
/QUIT
```

is used to return control of the C-III to the OS 65D operating system.

```
A*
```

CHAPTER 5

RECOMMENDATIONS AND CONCLUSIONS

The design and implementation of the system described in this report was suggested by the remark that "Experience with raster-scan graphics is simply too limited to justify offering an authoritative design." [NEW79] Baecker also points out that few systems for raster-scan graphics have been constructed on the same principle as random-scan systems [BAE79]. The present work is therefore intended as an exploratory project and represents only one of many possible approaches to the adaptation of random-scan techniques to raster-scan graphics systems. In particular, the segmented display processor program is an adaptation of a structure which evolved primarily for use on random-scan displays. The utility of organizing picture descriptions into segments is well established by experience with random-scan devices; it allows the user to structure the picture definition into logically related parts as well as to selectively manipulate portions of the picture. The KIM system described in this report demonstrates that a segmented display processor program can also be used to effectively organize, generate and manipulate a raster-scan based display system.

The KIM display system can serve as the test-bed for future exploration of the implementation of a graphics system on a relatively inexpensive microcomputer. For example, a future development currently planned includes the addition of a graphics input device such as a light

pen. This involves a simple, inexpensive addition to the MTU VM video board. Such a device is essential to any interactive graphics system. The software to interface the light pen and to incorporate graphical feedback techniques ([NEW79], pp. 160-181) can then be included in the system. In addition, adaptation of the host driver routines to permit their use by application programs is required. Such application programs may themslves be MCS6502 assembly-language routines or it may be desirable to employ the numerical capabilities of BASIC which is provided with the host operating system. The exact direction of any system expansion however, will depend on the particular application which incorporates the display system.

REFERENCES

[BAE79] Baecker R., "Digital Video Display Systems and Dynamic Graphics", Computer Graphics, Vol. 13, No. 2, (August 1979), pp. 48,56.

[BUX78] Buxton W., "Design Issues in the Foundation of a Computer-Based Tool for Music Composition", Technical Report CSRG-97, University of Toronto, (1978).

[CIA76] Ciarcia S., "Make Your Next Peripheral a Real Eye Opener", Byte, No. 15, (November 1976), p. 78.

[DIJ70] Randell B. and Buxton J.N., (eds.), "Software Engineering Techniques", NATO Scientific Affairs Division, Brussels 39, Belgium, (April 1970), p. 21.

[EAR77] Earnshaw R.A., "Line Generation For Incremental And Raster Devices", Computer Graphics, Vol. 11, No. 2, (August 77), p.199.

[GIL75] Giloi W.K., Encarnacao J. and Savitt S., "Interactive Graphics on Intelligent Terminals in a Time-Sharing Environment", Acta Informatica 5, (1975), pp. 257-271.

[GIL78] Giloi W.K., Interactive Computer Graphics, Data Structures, Algorithms, Languages, Prentice-Hall Inc., (1978).

[HET73] Hetzel W.C., Program Test Methods, Prentic-Hall,Inc., (1973)

[KNU68] Knuth D.E., The Art of Computer Programming, Vol. 1, Addison-Wesley Publishing Co., Reading Mass., (1968), pp. 435-453.

[LAN77] Lancaster D., The TV-Typewriter Cookbook, Howard W.Sams and Co. Inc., Indianapolis, Indiana, 46268, (1976).

[LIP80] Lippman A., "Movie-Maps: An Application of the Optical Videodisc to Computer Graphics", SIGGRAPH 80, to be published.

[MOS76] KIM-1 Microcomputer Module User Manual, MOS Technology Inc., Norristown, PA., (1976).

[MTU77] Micro Technology Unlimited, Documentation for the K-1008 Visable Memory Board, MTU, Manchester N.H., 03108, (1977).

[NEW79] Newman W.M. and Sproull R.F., Principles of Interactive Computer Graphics, 2nd Edition, McGraw-Hill (1979).

[OSB77] Osborn A., Jacobson S. and Kane J., An Introduction to Microcomputers Vol. 3 --Some Real Products, Adam Osborne and Associates, Berkeley California (June 1977), p. 8-63.

[PRA75] Pratt T.W., Programming Languages: Design and Implementation, Prentice-Hall Inc., Englewood Cliffs, N.J., (1975), p. 147.

[SAT77] Satterfield S.G., Rodriquez F. and Rogers D.F., "A Simple Approach to Computer Aided Milling With Interactive Graphics", Computer Graphics, Vol. 11, No. 2, (August 77), p.107.

[SPR75] Sproull R.F. and Newman W.M., "The Design of Gray-Scale Graphics Software", Procs. of the Conf. on Computer Graphics, Pattern Recognition, and Data Structure, IEEE Computer Society (May 14-15,1975), pp. 18-20.

[YOU75] Yourdon E., Techniques of Program Structure and Design, Prentice-Hall, Inc., Englewood Cliffs, N.J., (1975), p. 246.