Volume 2, Number 11 November 1984

IBM Personal Computer Seminar Proceedings

The Publication for Independent Developers
of Products
for IBM Personal Computers

Published by International Business Machines Corporation
Entry Systems Division

IBM®

Changes are made periodically to the information herein; any such changes will be reported in subsequent Proceedings.

It is possible that this material may contain reference to, or information about IBM products (machines and programs), programming or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM believes the statements contained herein are accurate as of the date of publication of this document. However, IBM makes no warranty of any kind with respect to the accuracy or adequacy of the contents hereof.

This publication could contain technical inaccuracies or typographical errors. Also, illustrations contained herein may show prototype equipment. Your system configuration may differ slightly. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

All specifications are subject to change without notice.

Printed in the
United States
of America

Contents

List of Illustrations

Introduction and Welcome

These are the Proceedings of the IBM Personal Computer Seminar, designed for independent developers of products for IBM Personal Computers. The purpose of these Proceedings is to aid you in your development efforts by providing relevant information about new product announcements and enhancements to existing products. This issue is prepared in conjunction with this seminar. The Proceedings of future siminars for the IBM Personal Computers also will be published and will cover topics presented at those seminars.

Throughout these Proceedings, the term Personal Computer and the term family of IBM Personal Computers address the IBM Personal Computer, the IBM Personal Computer XT, the IBM PC*jr*, the IBM Portable Personal Computer, and the IBM Personal Computer AT.

Purpose

What is our purpose in putting out a publication such as this? It is quite simple.

The IBM Personal Computer family is a resounding success. We've had a lot of help in achieving this success, and much of it came from the independent developers.

As you proceed with your development, do you at times wish for some bit of information or direction which would make the job easier? Information which IBM can provide? This is the type of information we want to make available to you.

Since we want to be assured of giving you the information you need, we ask you to complete the questionnaire which appears at the end of these Proceedings. Your response to this questionnaire will be taken into account in preparing the content of future issues, as well as the content of seminars we will present at microcomputer industry trade shows.

Topics

The following list gives a general indication of the topics we plan to cover in future seminars and include in the IBM Personal Computer Seminar proceedings:

- Information exchange forum — letters to the editor format
- Development tools — languages, database offerings
- Compatibility issues
- New devices — capacities and speeds
- System capacities — disk and memory
- Enhancements in maintenance releases
- Tips and techniques
- New system software
- Hardware design parameters
- Tips on organizing and writing documents for clear and easy reading
- Changes to terms and conditions

IBM Enhanced Graphics Adapter

The IBM Enhanced Graphics Adapter (EGA) is a new graphics adapter for the IBM Personal Computer family with the exception of the IBM PC*jr*. The design is based on a bit-plane architecture and involves concepts and techniques substantially more complex than its predecessors. It has more function and flexibility than the IBM Color Graphics/Monitor Adapter and IBM Monochrome Display and Printer Adapter. As a result of its enhanced capabilities, the EGA is a complex subject to understand. The Options and Adapters Technical Reference manual provides a complete hardware description and Basic Input Output System (BIOS) listing for the EGA.

This document aids in understanding the internal operation and programming of the EGA and it describes how the new BIOS calls can be used to support its enhanced capabilities. Also included are many programming examples and sample code listings.

Hardware Configurations

System Board Switches

On the IBM Personal Computer's System Board, switches 5 and 6 of Switch Block 1 must be on when the IBM Enhanced Graphics Adapter is installed, regardless of the attached display type. This is true even if a second display adapter is installed in the system. Likewise, on the IBM Personal Computer XT's System Board, switches 5 and 6 of Switch Block 1 must be on when the IBM Enhanced Graphics Adapter is installed.

There is only one switch on the system board of the IBM Personal Computer AT. When the only adapter you have is an IBM Color Graphics/Monitor Adapter, push the switch toward the front of your system unit. For an IBM Color Graphics/Monitor Adapter with another adapter, or for any other adapter or combination of adapters, push the switch toward the back of your system unit. You can then run the "Setup" program on the diagnostic diskette to tell your system what options you have installed.

Jumper Descriptions

There are three jumpers designated P1, P2 and P3, on the IBM Enhanced Graphics Adapter. When placed on pins 2 and 3, jumper P1 changes the function of pin 2 of the direct drive interface to ground (see Figure 1 on page 3). This selection supports displays that require four color input signals such as the IBM Color Display.

When jumper P1 is placed on pins 1 and 2, the least significant red output is placed on pin 2 of the direct drive interface (see Figure 1 on page 3). This selection supports the IBM Enhanced Color Display which requires six color input signals.

Jumper P2 is for a light pen attachment.

Jumper P3 changes the I/O address port of the IBM Enhanced Graphics Adapter within the system. In its normal position (on pins 1 and 2), all Enhanced Graphics Adapter addresses are in the range 3xxh. Moving jumper P3 to pins 2 and 3 changes the addresses to 2xxh. Operation of the adapter in the 2xxh mode is not supported in BIOS.

9-PIN DIRECT DRIVE INTERFACE SIGNALS

	Signal Name - Description	Pin	
	Ground	1	
	Secondary Red	2	
IBM Enhanced	Primary Red	3	Enhanced
Color	Primary Green	4	Graphics Adapter
Display	Primary Blue	5	(Hi-Res modes)
(model 5154)	Secondary Green/Intensity	6	
	Secondary Blue/Mono Video	7	
	Horizontal Drive	8	
	Vertical Drive	9	

	Signal Name - Description	Pin	
	Ground	1	
	Ground	2	
IBM Color	Red	3	Color/Graphics
Display	Green	4	Adapter or
(model 5153)	Blue	5	(EGA emulation
	Intensity	6	modes)
	Reserved	7	
	Horizontal Drive	8	
	Vertical Drive	9	

Figure 1. 9-Pin Direct Drive Interface Signals

Adapter Configurations

Only ONE IBM Enhanced Graphics Adapter can be in a system at a time. Since the cards are mapped to the same memory space, two EGAs in the system simultaneously will cause a conflict when accessing display memory, I/O ports and the Basic Input Output System (BIOS) stored in Read Only Memory (ROM).

Since there is only one 9-pin D-shell connector on the EGA card, only ONE monitor may be connected at a time.

An IBM Enhanced Graphics Adapter and an IBM Monochrome Display and Printer Adapter (MDPA) (see Figure 2 on page 4) can coexist in the IBM Personal Computer, IBM Personal Computer XT or IBM Personal Computer AT systems. In this configuration the MDPA drives the IBM Monochrome Display (model 5151) as usual and the EGA drives either the IBM Color Display (model 5153) or the IBM Enhanced Color Display (model 5154).

The only situation when you cannot have an EGA and an MDPA in the same system is when you use the EGA configured to drive the Monochrome Display. This causes an I/O and memory conflict when accessing I/O ports at 3Bxh and display memory at segment B000h. If the Power On Self Test (POST) finds an IBM Enhanced Graphics Adapter configured to drive the Monochrome Display, it initializes the IBM Enhanced Graphics Adapter but ignores the IBM Monochrome Display and Printer Adapter leaving it uninitialized.

When the IBM Enhanced Graphics Adapter is configured to drive the Monochrome Display, an IBM Color Graphics/Monitor Adapter (CGMA) or IBM Professional Graphics Controller (PGC) may be used to drive their companion color displays.

The IBM Enhanced Graphics Adapter and the IBM Professional Graphics Controller (PGC) can coexist with some exceptions. The following chart should clarify these configurations. In the chart, CGMA = IBM Color Graphics/Monitor Adapter, and MDPA = IBM Monochrome Display and Printer Adapter.

PGC	EGA	CGMA	MDPA	Exceptions
			1	
		1		
		1	1	
	1			
	1		1	EGA in color mode only
	1	1		EGA in mono mode only
1				
1			1	
1		1		PGC not in CGMA emulation mode
1		1	1	PGC not in CGMA emulation mode
1	1			PGC and EGA cannot both be in CGMA emulation mode
1	1		1	EGA in color mode, PGC not in emulation mode
1	1	1		EGA in mono and PGC not in CGMA emulation mode

Figure 2. Adapter Configuration Exceptions

A PGC cannot be installed in the system unit of an IBM Personal Computer. However it can be put in an IBM PC Expansion Unit, an IBM Personal Computer XT, or an IBM Personal Computer AT.

	EGA	MDPA	CGMA
sw4 on sw3 off sw2 off sw1 on	Primary Color Display (40x25)	Secondary	-
sw4 on sw3 off sw2 off sw1 off	Primary Color Display (80x25)	Secondary	-
sw4 off sw3 on sw2 on sw1 on	Primary Enhanced Display Emulation Mode	Secondary	-
sw4 off sw3 on sw2 on sw1 off	Primary Enhanced Display High Resolution Mode (640x350)	Secondary	-
sw4 off sw3 on sw2 off sw1 on	Primary Monochrome	-	Secondary (40x25)
sw4 off sw3 on sw2 off sw1 off	Primary Monochrome	-	Secondary (80x25)

Figure 3. EGA Primary Switch Settings

Note: The "primary" adapter is the adapter that the system defaults to at power-on.

Enhanced Display Emulation Mode means that the 5x7 character in an 8x8 box on the IBM Color Graphics/Monitor Adapter modes will be replaced with an enhanced 7x9 character in an 8x14 box on the IBM Enhanced Graphics Adapter.

	EGA	MDPA	CGMA
sw4 on sw3 on sw2 on sw1 on	Secondary Color Display (40x25)	Primary	-
sw4 on sw3 on sw2 on sw1 off	Secondary Color Display (80x25)	Primary	-
sw4 on sw3 on sw2 off sw1 on	Secondary Enhanced Display Emulation Mode	Primary	-
sw4 on sw3 on sw2 off sw1 off	Secondary Enhanced Display High Resolution Mode (640x350)	Primary	-
sw4 on sw3 off sw2 on sw1 on	Secondary Monochrome	-	Primary (40x25)
sw4 on sw3 off sw2 on sw1 off	Secondary Monochrome	-	Primary (80x25)

Figure 4. EGA Secondary Switch Settings

EGA Memory Options

The IBM Enhanced Graphics Adapter contains 64K bytes of storage configured as four 16K byte bit planes. The Graphics Memory Expansion Card plugs into the memory expansion connector on the adapter and adds one bank of 16K bytes to each of the four bit planes, thus increasing the graphics memory to 128K bytes. The expansion card also provides Dual In-line Package (DIP) sockets for further memory expansion. Populating the DIP sockets with the Graphics Memory Module Kit adds two additional 16K banks to each bit plane, bringing the graphics memory to 256K bytes.

The IBM Enhanced Graphics Adapter supports 640x350 graphics on the IBM Monochrome Display and the IBM Enhanced Color Display. Four color capability is supported on the EGA without the Graphics Memory Expansion Card (base 64K bytes), and sixteen colors are supported when the Graphics Memory Expansion Card is added to the EGA (128K bytes or above).

Feature Connector and Video Jacks

Signals are provided for custom applications on the IBM Enhanced Graphics Adapter Feature Connector. Pin 4 is connected to the auxiliary jack 1 on the card bracket of the adapter. Pin 5 is connected to the auxiliary jack 2 on the card bracket of the adapter. For example, one of these connectors could be used to receive external sync signals.

Note: The auxiliary jacks are not directly connected to any logic on the IBM Enhanced Graphics Adapter but may be connected through use of the Feature Connector.

Overview of Architecture

The basic blocks of the IBM Enhanced Graphics Adapter are as follows:

- ROM BIOS
- CRT Controller (CRTC)
- Sequencer
- Graphics Controller
- Attribute Controller
- Display Buffer
- Support Logic

The overall architecture is based on bit-mapped graphics configured as four bit planes. Each bit plane contains from 16KB to 64KB of Random Access Memory (RAM). There is a bit in the Map Mask Register to permit or deny access to each of the bit-planes. This allows the user to specify the specific bit plane to be accessed and/or modified. The base adapter contains 64K bytes of storage, and the memory expansion options allow up to 256K bytes of storage.

ROM BIOS

The IBM Enhanced Graphics Adapter contains a BIOS module that has extended video BIOS functions as listed in the figure below.

- Set Palette Registers
- RAM Loadable Character Generator
- Alternate Select
- TTY Write String

Figure 5. EGA extended BIOS Functions

Also within the BIOS are extended and enhanced mode selections to increase the resolution, increase the number of colors, and provide enhanced alphanumeric capabilities.

The BIOS provides support for both alphanumeric (A/N) modes and all-points-addressable (APA) graphic modes. This includes all modes supported by the IBM Monochrome Display and Printer Adapter and by the IBM Color Graphics/Monitor Adapter.

One of the EGA's advanced functions is a RAM Loadable Character Generator. Two character sets are stored within the EGA's BIOS ROM, one of which is automatically loaded into bit plane 2 when an alphanumeric mode is selected. This BIOS call (AH=1xh) should be used to change the character generator if desired. (This will be discussed in detail later.)

CRT Controller

This Large Scale Integrated (LSI) device generates the horizontal and vertical synchronizing timings, addressing for the regenerative video buffer, cursor and underline timings, and refresh addressing for the EGA's Dynamic Random Access Memories (DRAM's).

The Cathode Ray Tube Controller (CRTC) contains numerous Central Processing Unit (CPU) I/O programmable registers that allow the user to program the horizontal and vertical synchronization timings and cursor type and position. This device also contains I/O registers to program the maximum number of scan lines per character row, to read the memory location of the light pen, and a preset row scan register for smooth scrolling.

A line compare register can aid in setting up a status area that is immune to scrolling (i.e. a split screen).

Sequencer

This LSI device generates memory timings for the dynamic RAM's and the character clock for controlling regenerative memory fetches. It allows the CPU to access memory during active display intervals by inserting dedicated CPU memory cycles periodically between the display memory cycles. Map mask registers are available to protect entire memory maps from being changed. A clocking mode register is used to select the dot clock to be generated. The character map register is used to aid the BIOS in selecting which character generator fonts are to be used.

Graphics Controller

The Graphics Controller manipulates the data from the video memory to the Attribute Controller and the CPU.

In graphics modes, memory data is sent in serialized form to the Attribute Controller. In alpha modes the memory data is sent in parallel form, bypassing the Graphics Controller.

The Graphics Controller formats the data for use in compatible modes and provides color comparators for use in color painting modes. Hardware assists are provided to allow the CPU to write 32 bits (8 bits per plane) in a single memory cycle for fast color presetting of the display areas.

Several read and write modes allow ease in memory map manipulations. Additional logic allows the CPU to write data to the display on non-byte boundaries.

Hardware is provided to manipulate data in logical forms such as AND, OR, XOR, Rotate, and Color Writes. The Graphics Controller provides an I/O register to select the memory mapped location of the display buffer segment (hex A000, B000, or B800).

Attribute Controller

The Attribute Controller provides a palette of 16 colors selectable from a possible 64, each of which may be specified separately. Six color outputs are available for driving a display (R',G'/I,B'/V,R,G,B).

Note: I is intensity for IBM Color Graphics/Monitor Adapter and IBM Monochrome Display and Printer Adapter compatible modes; V is video data for monochrome modes.

Blinking and underlining are controlled by this device. The Attribute Controller takes data from the display memory and formats it for display on the CRT screen. Additional hardware has been provided to assist in horizontal pel panning in both alphanumeric and all-points-addressable modes.

Support Logic

The remaining logic on the EGA supports the LSI modules and creates latched buses for the CRTC, the CPU and the character generator. Two clock sources (14MHz and 16MHz) provide the desired dot rate. The clock is multiplexed under CPU I/O control. There are four other I/O registers on the adapter that are not part of the LSI devices. These registers are the Miscellaneous Output Register, the Feature Control Register, Input Status Register Zero, and Input Status Register One.

EGA Memory Organization

The following discussion is an overview of the EGA memory organization for the new 16 color graphics modes 0Dh, 0Eh and 10h. The EGA memory organizations for modes 0 - 7 are not discussed here since they are identical to their respective modes on the IBM Color Graphics/Monitor Adapter and IBM Monochrome Display and Printer Adapter.

In graphics mode, the memory on the IBM Enhanced Graphics Adapter is arranged as four planes sharing a common PC address space. The graphics Very Large Scale Integration (VLSI) chips contain several read/write assist registers to manipulate this memory. Each graphics chip supports two planes, therefore the EGA card has two graphics chips. In the following discussion (unless noted otherwise) this distinction is ignored and both chips are treated as a single entity.

In graphics mode each pel is mapped to one bit in the address space. The color or attribute of a given pel is determined by the four bits on the four memory planes behind a given bit address.

The IBM Enhanced Graphics Adapter card contains a latch register for each bit plane (32-bits total). Each memory read will cause the contents of the accessed location in each plane to be stored in the latches. Memory write operations may combine the source data with the latch contents before storing into the accessed memory location.

Pixel addressing is achieved by specifying a bit mask to apply with write operations. The bit masks specifies whether the stored memory location receives a given bit from the CPU's source data or from the latch registers.

Note: In this case writing bits from the latch registers is intended to preserve the current contents of the stored location. This will happen only if the latches have been "primed" by reading the location to be stored.

The Bit Mask Register selects which pel(s) in the addressed byte are affected. The bits enabled by the bit mask will reflect the changes, while the bits not enabled are unchanged. Thus the bit mask register provides pel addressability.

The Sequencer Map Mask Register (SMM) selects which planes are enabled/disabled for CPU writes. For example, the Sequencer Map Mask register set to 0011b disables all writes to maps 2 and 3.

Note: A logical 1 enables the bit plane.

The Data Rotate/Function Select (FS) register selects a read-modify-write function. The EGA card supports 4 logical read-modify-write functions (see Figure 6 on page 9) that take place between the CPU data and the data in the latches during a CPU write.

- 00 replace
- 01 and
- 10 or
- 11 xor

Figure 6. EGA read-modify-write functions

There are three write modes and two read modes between the CPU and the memory. This requires two write bits and one read bit (wwr); see Figure 7 on page 10. The write modes govern how the CPU data is modified by a CPU write. Because any byte on the EGA addressed by the CPU contains 32 bits, two read modes are provided that reduce 32 bits to the 8 bits returned on the data bus.

- Write mode 0

  In write mode 0, all enabled memory planes respond identically to write operations. This mode responds like normal PC memory, except that the data is replicated across enabled planes. This is the BIOS default write mode.

- Write mode 1

  Write mode 1 provides an on-card 32-bit move assist. During a CPU write, the current contents of the latches (which were loaded by a previous CPU read) are written directly to memory; the data bus is effectively disconnected in this mode.

- Write mode 2

  Write mode 2 is a write color interface. The CPU data is a color written to the four planes for the pels enabled by the bit mask register. CPU data bit 0 is written to map 0, data bit 1 is written to map 1, and so on.

  Note: Up to 8 pels are written depending on the bit mask register.

- Read mode 0

  In read mode 0, the 8 bits in the addressed byte on the map indicated by the read map select register are returned. This mode will resemble normal PC memory, and is the BIOS default.

- Read mode 1

  Read mode 1 performs a color compare operation on the byte addressed. The color compare value is loaded into the color equivalence register (Graphics Chip Register "02). Responding to a mode 1 CPU read, the EGA card will return a 1 in every bit position that matched the color compare value.

Figure 7 on page 10 diagrams the major elements for a write operation. The latches are first loaded by a CPU read from the byte (four maps) that is to be modified. The Bit Mask and SMM registers are masks that enable/disable changes to each bit in the memory. The enabled bits are modified with the CPU data according to the current write mode and Function Select (FS) i.e read-modify-write mode. The byte (all four maps) addressed by the address bus is then replaced with the latch data logically combined with the source data (as specified by FS).

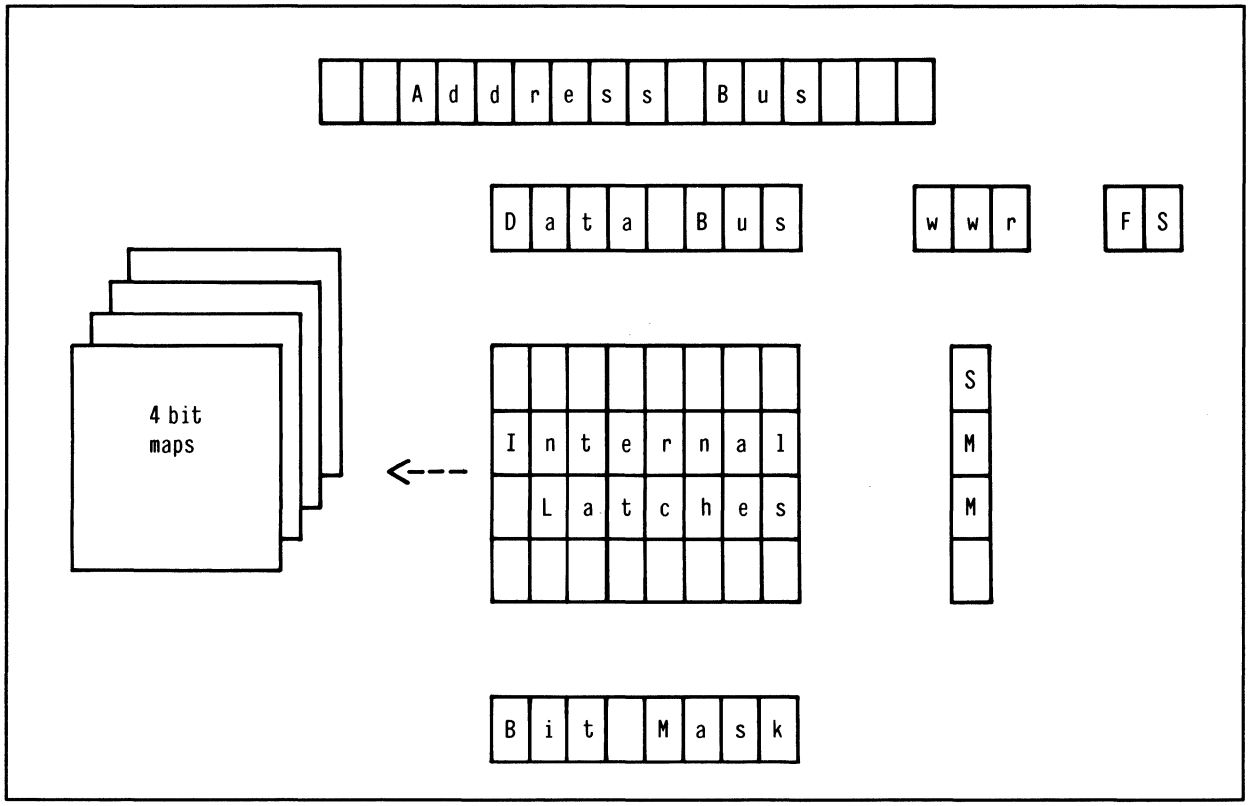

Figure 7. Outline of EGA Architecture

The following example of code writes the byte pointed to by ES:SI (8 pels) in color 5 in write mode 0. It is assumed that the graphics chips are already in write mode 0. This is the write mode used by the BIOS.

```
; --- first clear current contents
        mov     al,0            ; this clears 8
        mov     es:[si],al      ; pels to color 0

; --- set map mask to color
        mov     dx,03C4h
        mov     ah,02h          ; bit 1
        mov     al,05h          ; color 5
        call    out_dx

; --- write 8 magenta pels
        mov     al,0FFh
        mov     es:[si],al

; --- subroutine to output word to indexed port
out_dx  proc    near            ; ah=index al=data dx=port
        xchg    al,ah           ; index
        out     dx,al           ; set index register
        xchg    al,ah           ; data
        inc     dx              ; data register
        out     dx,al           ; set data register
        dec     dx              ; restore index register
        ret
out_dx  endp
```

The next code section writes one pel in the byte addressed by ES:SI in color 5 in write mode 0. Note that this code latches the location with a CPU read in order to apply the bit mask.

```
; --- select 1 bit to change
        mov     dx,03CEh
        mov     al,02h          ; bit 1
        mov     ah,08h          ; bit mask register
        call    out_dx

; --- clear current contents
        mov     al,es:[si]      ; latch current contents
        mov     al,0
        mov     es:[si],al

; --- set map mask to color
        mov     dx,03C4h
        mov     ah,02h          ; bit 1
        mov     al,05h          ; color 5
        call    out_dx

; --- write 1 magenta pel
        mov     al,02h
        mov     es:[si],al
```

An alternative when writing less than 8 pels is to make use of write mode 2. This code writes color 4 to pel 1 of the byte addressed by ES:SI in write mode 2. Once again it is assumed that the card is already in write mode 2.

```
; --- select 1 bit to change
        mov     dx,03CEh
        mov     al,02h          ; bit 1
        mov     ah,08h          ; bit mask register
        call    out_dx

; --- write 1 red pel
        mov     ah,es:[si]      ; latch
        mov     al,04h          ; red
        mov     es:[si],al
```

Reading the color of a particular pixel requires a separate read from each bit plane in read mode 0. In this example, ES:SI points to the desired byte address and CL is the bit number (0-7) of the pel to read. The color value is returned in BL.

```
        mov     dx,3CEh         ; graphics chip
        mov     al,04h          ; read map select
        mov     ah,3            ; 4 bit maps (3210)
        mov     ch,80h          ; 1 bit
        shr     ch,cl           ; align pel mask
        xor     bl,bl           ; color accumulator
rdl:    out     dx,ax           ; select read map
        mov     bh,es:[si]      ; read 8 bits on map
        and     bh,ch           ; isolate pel (1 bit)
        shr     bh,cl           ; right justify it
        shl     bl,1            ; make room
        or      bl,bh           ; set this pel
        dec     ah              ; next map (3210)
        jge     rdl             ; repeat
```

Modes of Operation

The new modes supported by the IBM Enhanced Graphics Adapter are as follows:

IBM Color Display

Hex Mode	Type	Colors	Alpha Format	Buffer Start	Box Size	Max. Pages	Memory Required(k)	Res.
D	APA	16	40x25	A0000	8x8	2/4/8	64/128/256	320x200
E	APA	16	80x25	A0000	8x8	1/2/4	64/128/256	640x200

IBM Monochrome Display

Hex Mode	Type	Atr.	Alpha Format	Buffer Start	Box Size	Max. Pages	Memory Required (k)	Res.
F	APA	4	80x25	A0000	8x14	1/2	64/128	640x350

IBM Enhanced Color Display

Hex Mode	Type	Colors	Alpha Format	Buffer Start	Box Size	Max. Pages	Memory Required (k)	Res.
0	A/N	16/64	40x25	B8000	8x14	8	64	320x350
1	A/N	16/64	40x25	B8000	8x14	8	64	320x350
2	A/N	16/64	80x25	B8000	8x14	8	64	640x350
3	A/N	16/64	80x25	B8000	8x14	8	64	640x350
10	APA	4/64	80x25	A0000	8x14	1	64	640x350
10	APA	16/64	80x25	A0000	8x14	1/2	128/256	640x350

Figure 8. New EGA Modes

Note: Modes 0, 1, 2, and 3 are also supported when using the IBM Color Display. However, BIOS provides enhanced support for these modes when the IBM Enhanced Color Display is attached.

New BIOS Calls

The IBM Enhanced Graphics Adapter has an on-board ROM which contains extensions to video BIOS to support its enhanced capabilities. A discussion of the new BIOS calls, including examples, follows.

Set Palette Registers

The BIOS call AH=10h provides a software interface to the IBM Enhanced Graphics Adapter's color palette registers. Also included is a BIOS interface to the Intensify/Blinking attribute for alphanumeric text modes. The following is a listing of the BIOS interface for this call as it is published in the Options and Adapters Technical Reference manual.

```
AH = 10h  SET PALETTE REGISTERS

       AL = 0         SET INDIVIDUAL PALETTE REGISTER
        BL  = PALETTE REGISTER TO BE SET
        BH  = VALUE TO SET

       AL = 1         SET OVERSCAN REGISTER
        BH  = VALUE TO SET

       AL = 2         SET ALL PALETTE REGISTERS AND OVERSCAN
        ES:DX POINTS TO A 17-BYTE TABLE
        BYTES 0 - 15 ARE THE PALETTE VALUES, RESPECTIVELY
        BYTE 16 IS THE OVERSCAN VALUE

       AL = 3         TOGGLE INTENSIFY/BLINKING BIT
        BL  = 0       ENABLE INTENSIFY
        BL  = 1       ENABLE BLINKING
```

Figure 9. Set Palette Register BIOS Interface AH = 10h

Set Single Palette Register

There are 16 palette registers on the EGA. Each palette register can take on 1 of 64 values. For 16/64 color high-resolution mode on the IBM Enhanced Color Display the register value should have this format:

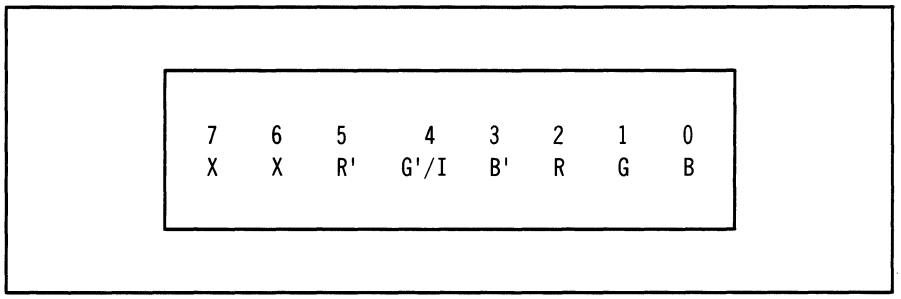

Figure 10. Palette Data Definition

Note: For IBM Color Graphics/Monitor Adapter compatibility modes, G' (bit 4) maps to the intensity bit (I).

The default palette for 16-color high-resolution mode set up by BIOS is:

Palette Register	Color (in hex)	Color
0	0	black
1	1	blue
2	2	green
3	3	cyan
4	4	red
5	5	magenta
6	14	brown
7	7	white
8	38	dark gray
9	39	light blue
10	3A	light green
11	3B	light cyan
12	3C	light red
13	3D	light magenta
14	3E	yellow
15	3F	intensified white

Figure 11. IBM Default Color Palette

For example, to change palette register 1 to light red we would use the following code:

```
mov     ah,10h
mov     al,0
mov     bl,1            ;palette register 1
mov     bh,0Ch          ;light red
int     10h
```

Set Overscan Register

To set the overscan (border) register use the following code:

```
mov     ah,10h
mov     al,1
mov     bh,1            ;blue
int     10h
```

Note: The IBM Enhanced Color Display supports overscan in 200 line modes only.

Set Palette and Overscan

All of the palette registers and the overscan register may be set in just one BIOS call if a table of 17 values is provided. The following example demonstrates the loading of all 17 palette registers.

```
code        segment public 'code'
            assume  cs:code,ds:code
            org     100h
start:      jmp     init
pal_table   label   byte
            db      10h             ; table for...
            db      11h             ;
            db      12h             ;
            db      13h             ;
            db      14h             ;
            db      15h             ; the 16...
            db      16h             ;
            db      17h             ;
            db      18h             ;
            db      19h             ;
            db      1Ah             ; colors of...
            db      1Bh             ;
            db      1Ch             ;
            db      1Dh             ;
            db      1Eh             ; the palette.
            db      1Fh             ;
            db      20h             ; overscan value

init:       mov     ah,10h
            mov     al,2
            push    ds              ; put the segment of
            pop     es              ; pal_table into es
            mov     dx,offset pal_table
            int     10h
            int     20h             ; exit to DOS
code        ends
            end     start
```

Toggle Intensify/Blinking Bit

The data format for alphanumeric text modes on the IBM Enhanced Graphics Adapter is the same as its predecessors the IBM Monochrome Display and Printer Adapter and IBM Color Graphics/Monitor Adapter, i.e. a byte of character data followed by a byte of attribute data.

The attribute data for color A/N modes is as follows:

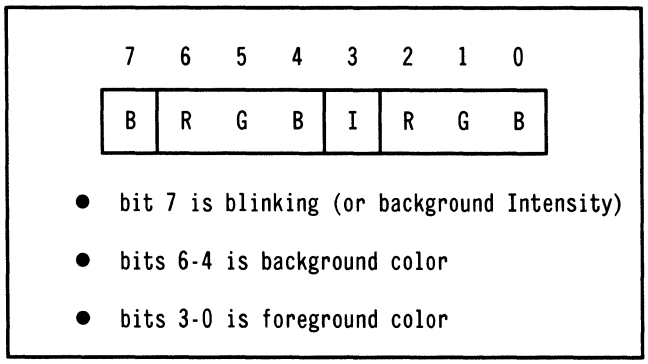

Figure 12. Attribute Data Definition

The interpretation of bit 7 of the attribute byte can be changed from Foreground Blink to Background Intensity. This is accomplished on the IBM Color Graphics/Monitor Adapter and IBM Monochrome Display and Printer Adapter by changing bit 5 of the Mode Select Register (3D8h and 3B8h respectively). When bit 5 is set to 1, attribute bit 7 set to 1 enables the foreground blink attribute. If bit 5 of the Mode Select Register is set to 0, attribute bit 7 set to 1 enables background Intensity.

The IBM Enhanced Graphics Adapter provides a BIOS interface for this attribute mapping, call AH=10h and AL=3. This BIOS call with BL=0 sets attribute bit 7 to control background intensity, for BL=1, attribute bit 7 controls foreground Blinking.

The following program fills the first ten lines of the screen with character 'E' and an attribute of blink, white foreground, and blue background. It fills the next ten lines with character 'G' and an attribute of no blink, white foreground, and blue background. When a key is pressed, the BIOS call toggles attribute bit 7 and waits for another key press. Each subsequent key press will switch between Blink and Intensify. To return to DOS, press the Enter Key.

Note: Only the top part of the screen toggles, because its attribute was the only one that had attribute bit 7 set to a 1.

```
code            segment public 'code'
                assume  cs:code,ds:code
                org     100h
start:          jmp     initial

char1           db      45h     ; 'E'
atr1            db      97h     ; blink, white foreground, blue background
char2           db      47h     ; 'G'
atr2            db      17h     ; white foreground and blue background
msg             db     'Type any key to toggle or <Enter> to return to DOS.'

check           proc    near
                cmp     al,13   ; check for <Enter>
                je      exit    ; if <Enter> then return to DOS
                ret
check           endp

pmsg            proc    near
                mov     ah,13h  ; BIOS call to write string
                push    ds      ; load ES with the
                pop     es      ;   segment of the string
                mov     bp,offset msg
                mov     cx,48   ; number of characters in the string
                mov     dx,1400h        ; set cursor position  (20,0)
                mov     bh,0    ; page number
                mov     al,0
                mov     bl,20h  ; attribute, black on green
                int     10h
                ret
pmsg            endp

initial:        mov     ah,2    ; EGA BIOS call to set cursor position
                mov     dx,0    ; store row(0) in DH and column(0) in DL
                mov     bh,0    ; store page in BH
                int     10h     ;
                mov     ah,9    ; EGA BIOS call to write char and attribute
                mov     bh,0    ; store page in BH
                mov     cx,800  ; fill in 10 lines
                mov     al,char1        ; load AL with char to write
                mov     bl,atr1         ; load BL with attribute
```

```
                int     10h             ;
                mov     ah,2            ; set cursor position
                mov     dx,0a00h        ; row 10, column 0
                mov     bh,0            ; page number
                int     10h             ;
                mov     ah,9            ; write char and attribute
                mov     bh,0            ; page number
                mov     cx,800          ; write to 10 lines
                mov     al,char2        ; char to write
                mov     bl,atr2         ; attribute
                int     10h             ;
                mov     ah,2            ; set cursor position
                mov     dx,1400h        ; row 20, column 0
                mov     bh,0            ; page number
                int     10h             ;

loop1:          call    pmsg            ; print message
                mov     ah,0            ; wait for key press
                int     16h             ;  using BIOS call 16
                call    check           ; check for <Enter>
                mov     ax,1003h        ; AH=10h, AL=3h
                mov     bl,0            ; BL=0 for intensify
                int     10h             ;
                call    pmsg            ; print message
                mov     ah,0            ; wait for key press
                int     16h
                call    check           ; check for <Enter>
                mov     ax,1003h        ; AH=10h, AL=3h
                mov     bl,1            ; BL=1 for blinking
                int     10h
                jmp     loop1
exit:           int     20h             ; return to DOS

code            ends
                end     start
```

Character Generator Routines

The new BIOS call AH=11h provides a software interface to the IBM Enhanced Graphics Adapter's ram-loadable character generator. Through this call either a graphics or an alpha character set may be loaded. The source of these character sets may be either the EGA's on-board ROM or a user specified file.

The following is a listing of this BIOS interface as it is published in the Options and Adapters Technical Reference manual.

```
(AH) = 11  CHARACTER GENERATOR ROUTINE
       NOTE : THIS CALL WILL INITIATE A MODE SET, COMPLETELY
              RESETTING THE VIDEO ENVIRONMENT BUT MAINTAINING
              THE REGEN BUFFER.

       AL = 00   USER ALPHA LOAD
           ES:BP - POINTER TO USER TABLE
               CX    - COUNT TO STORE
               DX    - CHARACTER OFFSET INTO TABLE
               BL    - BLOCK TO LOAD
               BH    - NUMBER OF BYTES PER CHARACTER
       AL = 01   ROM MONOCHROME SET
               BL    - BLOCK TO LOAD
       AL = 02   ROM 8X8 DOUBLE DOT
               BL    - BLOCK TO LOAD
       AL = 03   SET BLOCK SPECIFIER
               BL    - CHAR GEN BLOCK SPECIFIER
                       D3-D2  ATTR BIT 3 ONE,  CHAR GEN 0-3
                       D1-D0  ATTR BIT 3 ZERO, CHAR GEN 0-3
               NOTE : WHEN USING AL = 03 A FUNCTION CALL
                       AX = 1000H
                       BX = 0712H
                       IS RECOMMENDED TO SET THE COLOR PLANES
                       RESULTING IN 512 CHARACTERS AND EIGHT
                       CONSISTENT COLORS.
```

Figure 13. Character Generator BIOS Interface AH = 11h

```
NOTE : THE FOLLOWING INTERFACE (AL=1X) IS SIMILAR IN FUNCTION
       TO (AL=0X) EXCEPT THAT :
              - PAGE ZERO MUST BE ACTIVE
              - POINTS (BYTES/CHAR) WILL BE RECALCULATED
              - ROWS WILL BE CALCULATED FROM THE FOLLOWING:
                     INT[(200 OR 350) / POINTS] - 1
              - CRT_LEN WILL BE CALCULATED FROM :
                     (ROWS + 1) * CRT_COLS * 2

              - THE CRTC WILL BE REPROGRAMMED AS FOLLOWS:

                     R09H = POINTS - 1       MAX SCAN LINE
                            R09H done only in mode 7
                     R0AH = POINTS - 2       CURSOR START
                     R0BH = 0                CURSOR END
                     R12H =                  VERT DISP END
                            [(ROWS + 1) * POINTS] - 1
                     R14H = POINTS           UNDERLINE LOC

       THE ABOVE REGISTER CALCULATIONS MUST BE CLOSE TO THE
       ORIGINAL TABLE VALUES OR UNDETERMINED RESULTS WILL
       OCCUR.

       NOTE : THE FOLLOWING INTERFACE IS DESIGNED TO BE
              CALLED ONLY IMMEDIATELY AFTER A MODE SET HAS
              BEEN ISSUED. FAILURE TO ADHERE TO THIS PRACTICE
              MAY CAUSE UNDETERMINED RESULTS.

       AL = 10   USER ALPHA LOAD
           ES:BP - POINTER TO USER TABLE
               CX    - COUNT TO STORE
               DX    - CHARACTER OFFSET INTO TABLE
               BL    - BLOCK TO LOAD
               BH    - NUMBER OF BYTES PER CHARACTER
       AL = 11   ROM MONOCHROME SET
               BL    - BLOCK TO LOAD
       AL = 12   ROM 8X8 DOUBLE DOT
               BL    - BLOCK TO LOAD
```

Figure 14. Character Generator BIOS Interface (Continued)

```
NOTE : THE FOLLOWING INTERFACE IS DESIGNED TO BE
       CALLED ONLY IMMEDIATELY AFTER A MODE SET HAS
       BEEN ISSUED. FAILURE TO ADHERE TO THIS PRACTICE
       MAY CAUSE UNDETERMINED RESULTS.

AL = 20   USER GRAPHICS CHARS  INT 01FH (8X8)
    ES:BP - POINTER TO USER TABLE
AL = 21   USER GRAPHICS CHARS
    ES:BP - POINTER TO USER TABLE
        CX    - POINTS (BYTES PER CHARACTER)
        BL    - ROW SPECIFIER

                BL = 0  USER
                        DL - ROWS
                BL = 1  14 (0EH)
                BL = 2  25 (19H)
                BL = 3  43 (2BH)

AL = 22   ROM 8 X 14 SET
        BL    - ROW SPECIFIER
AL = 23   ROM 8 X 8 DOUBLE DOT
        BL    - ROW SPECIFIER


AL = 30   INFORMATION
                CX    - POINTS
                DL    - ROWS
        BH    - 0       RETURN CURRENT INT 1FH PTR
            ES:BP - PTR TO TABLE
        BH    - 1       RETURN CURRENT INT 44H PTR
            ES:BP - PTR TO TABLE
        BH    - 2       RETURN ROM 8 X 14 PTR
            ES:BP - PTR TO TABLE
        BH    - 3       RETURN ROM DOUBLE DOT PTR
            ES:BP - PTR TO TABLE
        BH    - 4       RETURN ROM DOUBLE DOT PTR (TOP)
            ES:BP - PTR TO TABLE
        BH    - 5       RETURN ROM ALPHA ALTERNATE 9X14
            ES:BP - PTR TO TABLE
```

Figure 15. Character Generator BIOS Interface (Continued)

User Alpha Character Loads

The calls for AL='0x' and AL='1x' both load character sets for the IBM Enhanced Graphics Adapter's alphanumeric modes and are quite similar in function. The main difference is that a mode set is automatically initiated for AL='0x', resetting the video environment and maintaining the contents of the video buffer.

The AL='1x' call, designed to immediately follow a mode set. POINTS, ROWS, CRT_LEN and the CRT cursor definition are updated.

When AL=03, up to four character sets of 256 characters may be loaded into the EGA's ram. Once loaded any two of these may be linked together to form a character set of 512 characters.

Note: Only one block of 256 characters may be loaded without memory expansion for the EGA. Each increment of 64K bytes of memory adds memory space for 1 more 256 character block; up to 4 blocks with the EGA's memory expanded to 256K bytes.

Note: When using a 512 character set, bit 3 of the attribute byte selects between the upper and lower 256 character blocks. The redefinition of this bit causes the loss of 8 foreground colors and the blink attribute.

User Graphics Character Loads

This call is similar to AL = '0x' and AL = '1x' except it is used to load character sets for the IBM Enhanced Graphics Adapter's graphics modes. A COUNT for this mode may not be specified and a full set of 256 characters must be supplied defining the entire character set.

Information

This call, AL = 30h, returns information about the current state of the EGA's character set.

- CX returns POINTS = Bytes/Character.
- DL returns ROWS =
  INT[(200 OR 350) / POINTS] - 1.

  Note: 200 for 200 line modes. 350 for 350 line modes. The INT function takes the integer value.

By setting BH, pointers to the current and ROM character tables may be returned.

Examples

Load ROM Character Set

The following code will temporarily load the ROM 8X8 Double Dot character set into block zero for Mode 2:

```
        MOV     AX,0002h    ; set up call for Mode 2
        INT     10h         ; make call
        MOV     AX,1112h    ; specify 8X8 Character Set load
        MOV     BL,00       ;    into block zero
        INT     10h         ; make call
```

This code may be imbedded into an application or loaded with debug. The result will be an 80 by 43 character screen in high-res (350 line) modes or an 80 by 25 character screen in medium-res (200 line) modes.

Load User Character Set

The following code will temporarily load an 8X8 Double Dot Character set from a disk file called UFONT1.I into block zero for Mode 2:

Note: This character set definition will be maintained until the next mode set.

```
        title   set custom screen and font
code    segment para 'code'
        org     100h
        assume  cs:code,ds:code
begin   proc    near
        jmp     start
;
; - Main Routine ------------------------------------
;--- point to font table
start:  mov     bp,offset font1 ; 8x8 font
        mov     cx,256          ; 256 characters
        mov     dx,0            ; 0-FF
        mov     bx,0800h        ; 8 bytes/char- block 0
        mov     ax,1110h        ; user alpha load
        int     10h
        ret
begin   endp
; --- 8x8 font specification
font1   label   byte
        include ufont1.i
limit   equ     $
code    ends
        end     begin
```

The font file, UFONT1.I, contains 2048 define bytes i.e. an 8 byte definition for each of 256 characters. In general a character set definition file requires the product of POINTS (bytes/character) and number of characters to be loaded.

Note: Any number of characters (up to 256 per block) may be loaded for alpha modes. Graphics modes require all 256 characters to be loaded.

The following is an example of a character definition for an 8X8 capital letter A as it would appear in an assembly language file.

```
db      030h,078h,0CCh,0CCh,0FCh,0CCh,0CCh,000h ; A D_41
```

Figure 16. Character Data Example 1

A complete definition file for 256 8X8 characters would have an entry for each character, including non-printable control characters which have an entry of all zeros. Below is an example of how the above data is transformed into a character.

Note: The numbers in this example are in Hexadecimal notation.

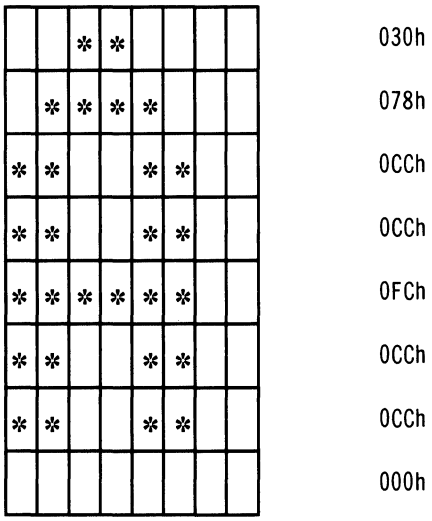

Figure 17. Character Data Example 2

The following is an example of character set definition file abbreviated from the ROM 8X8 double dot font. The complete listing of this file may be found in the EGA BIOS listing in the Options and Adapters Technical Reference manual.

```
                ;--------------------------------------;
                ; Control characters                   ;
                ;--------------------------------------;
                                                ; Double Dot
db      000h,000h,000h,000h,000h,000h,000h,000h ; D_00
db      07Eh,081h,0A5h,081h,0BDh,099h,081h,07Eh ; D_01
db      07Eh,0FFh,0dbh,0FFh,0C3h,0E7h,0FFh,07Eh ; D_02
db      06Ch,0FEh,0FEh,0FEh,07Ch,038h,010h,000h ; D_03
db      010h,038h,07Ch,0FEh,07Ch,038h,010h,000h ; D_04
db      038h,07Ch,038h,0FEh,0FEh,07Ch,038h,07Ch ; D_05
db      010h,010h,038h,07Ch,0FEh,07Ch,038h,07Ch ; D_06
db      000h,000h,018h,03Ch,03Ch,018h,000h,000h ; D_07
db      0FFh,0FFh,0E7h,0C3h,0C3h,0E7h,0FFh,0FFh ; D_08
db      000h,03Ch,066h,042h,042h,066h,03Ch,000h ; D_09
db      0FFh,0C3h,099h,0BDh,0BDh,099h,0C3h,0FFh ; D_0A
db      00Fh,007h,00Fh,07Dh,0CCh,0CCh,0CCh,078h ; D_0B
db      03Ch,066h,066h,066h,03Ch,018h,07Eh,018h ; D_0C
        "   "   "   "   "   "   "   "   "
        "   "   "   "   "   "   "   "   "
  ***** Section of table omitted for brevity *****
        "   "   "   "   "   "   "   "   "
        "   "   "   "   "   "   "   "   "
        ;--------------------------------------;
        ; Start printable ASCII characters     ;
        ;--------------------------------------;
                                                ;
db      000h,000h,000h,000h,000h,000h,000h,000h ; SP D_20
db      030h,078h,078h,030h,030h,000h,030h,000h ; ! D_21
```

(continued on next page)

```
db      06Ch,06Ch,06Ch,000h,000h,000h,000h,000h ; " D_22
db      06Ch,06Ch,0FEh,06Ch,0FEh,06Ch,06Ch,000h ; # D_23
db      030h,07Ch,0C0h,078h,00Ch,0F8h,030h,000h ; $ D_24
db      000h,0C6h,0CCh,018h,030h,066h,0C6h,000h ; % D_25
db      038h,06Ch,038h,076h,0DCh,0CCh,076h,000h ; & D_26
db      060h,060h,0C0h,000h,000h,000h,000h,000h ; " D_27
db      018h,030h,060h,060h,060h,030h,018h,000h ; ( D_28
db      060h,030h,018h,018h,018h,030h,060h,000h ; ) D_29
db      000h,066h,03Ch,0FFh,03Ch,066h,000h,000h ; * D_2A
db      000h,030h,030h,0FCh,030h,030h,000h,000h ; + D_2B
db      000h,000h,000h,000h,000h,030h,030h,060h ; , D_2C
db      000h,000h,000h,0FCh,000h,000h,000h,000h ; - D_2D
db      000h,000h,000h,000h,000h,030h,030h,000h ; . D_2E
db      006h,00Ch,018h,030h,060h,0C0h,080h,000h ; / D_2F

db      07Ch,0C6h,0CEh,0DEh,0F6h,0E6h,07Ch,000h ; 0 D_30
db      030h,070h,030h,030h,030h,030h,0FCh,000h ; 1 D_31
db      078h,0CCh,00Ch,038h,060h,0CCh,0FCh,000h ; 2 D_32
db      078h,0CCh,00Ch,038h,00Ch,0CCh,078h,000h ; 3 D_33
db      01Ch,03Ch,06Ch,0CCh,0FEh,00Ch,01Eh,000h ; 4 D_34
db      0FCh,0C0h,0F8h,00Ch,00Ch,0CCh,078h,000h ; 5 D_35
db      038h,060h,0C0h,0F8h,0CCh,0CCh,078h,000h ; 6 D_36
db      0FCh,0CCh,00Ch,018h,030h,030h,030h,000h ; 7 D_37
db      078h,0CCh,0CCh,078h,0CCh,0CCh,078h,000h ; 8 D_38
db      078h,0CCh,0CCh,07Ch,00Ch,018h,070h,000h ; 9 D_39
db      000h,030h,030h,000h,000h,030h,030h,000h ; : D_3A
db      000h,030h,030h,000h,000h,030h,030h,060h ; ; D_3B
db      018h,030h,060h,0C0h,060h,030h,018h,000h ; < D_3C
db      000h,000h,0FCh,000h,000h,0FCh,000h,000h ; = D_3D
db      060h,030h,018h,00Ch,018h,030h,060h,000h ; > D_3E
db      078h,0CCh,00Ch,018h,030h,000h,030h,000h ; ? D_3F

db      07Ch,0C6h,0DEh,0DEh,0DEh,0C0h,078h,000h ; @ D_40
db      030h,078h,0CCh,0CCh,0FCh,0CCh,0CCh,000h ; A D_41
db      0FCh,066h,066h,07Ch,066h,066h,0FCh,000h ; B D_42
db      03Ch,066h,0C0h,0C0h,0C0h,066h,03Ch,000h ; C D_43
db      0F8h,06Ch,066h,066h,066h,06Ch,0F8h,000h ; D D_44
db      0FEh,062h,068h,078h,068h,062h,0FEh,000h ; E D_45
db      0FEh,062h,068h,078h,068h,060h,0F0h,000h ; F D_46
db      03Ch,066h,0C0h,0C0h,0CEh,066h,03Eh,000h ; G D_47
        "   "   "   "   "   "   "   "   "
        "   "   "   "   "   "   "   "   "
 ***** Section of table omitted for brevity *****
        "   "   "   "   "   "   "   "   "
        "   "   "   "   "   "   "   "   "
        ;--------------------------------------;
        ; Last entries of character set        ;
        ;--------------------------------------;
                                               ;
db      000h,076h,0DCh,000h,076h,0DCh,000h,000h ;  D_F7
db      038h,06Ch,06Ch,038h,000h,000h,000h,000h ;  D_F8
db      000h,000h,000h,018h,018h,000h,000h,000h ;  D_F9
db      000h,000h,000h,000h,018h,000h,000h,000h ;  D_FA
db      00Fh,00Ch,00Ch,00Ch,0ECh,06Ch,03Ch,01Ch ;  D_FB
db      078h,06Ch,06Ch,06Ch,06Ch,000h,000h,000h ;  D_FC
db      070h,018h,030h,060h,078h,000h,000h,000h ;  D_FD
db      000h,000h,03Ch,03Ch,03Ch,03Ch,000h,000h ;  D_FE
db      000h,000h,000h,000h,000h,000h,000h,000h ;  D_FF
```

Figure 18. Character Set File Example

Alternate Select

The new BIOS call AH=12h provides a software interface status information particular to the IBM Enhanced Graphics Adapter. An alternate print screen routine may also be selected through this call. The following is a listing of this BIOS interface as it is published in the Technical Reference.

```
(AH) = 12   ALTERNATE SELECT

        BL = 10  RETURN EGA INFORMATION
                BH = 0 - COLOR MODE IN EFFECT <3><D><X>
                     1 - MONOC MODE IN EFFECT <3><B><X>
                BL = MEMORY VALUE
                        0 0 - 064K      0 1 - 128K
                        1 0 - 192K      1 1 - 256K
                CH = FEATURE BITS
                CL = SWITCH SETTING

        BL = 20  SELECT ALTERNATE PRINT SCREEN ROUTINE
```

Figure 19. Alternate Select BIOS Interface AH = 12h

This program has AH=12h and BL=10h. It returns information on the EGA.

- BH tells whether the EGA is in color or monochrome mode.
- BL returns the amount of memory on the EGA.
- CL returns the switch setting of the EGA.
- CH returns the feature bits value.

CH is set up with bits 7-4 unused, bits 3 and 2 are reserved, and bits 1 and 0 are the feature control bits. The output of bit 1 goes to FEAT1 (pin 17) and the output of bit 0 goes to FEAT0 (pin 19).

```
code            segment public 'code'
                assume  CS:code,DS:code,ES:code
                org     100h
begin:          jmp     start

mem             db      ?
swset           db      ?
fbits           db      ?
MEM0            db     '64k of memory on ega'
MEM1            db     '128k of memory on ega'
MEM2            db     '192k of memory on ega'
MEM3            db     '256k of memory on ega'
MODEC           db     'color mode in effect'
MODEM           db     'monochrome mode in effect'
SWITCH          db      48,49,50,51,52,53,54,55,56,57,97,98,99,100,101,102
swmsg           db     'Switch setting = '
strg1           db     '7   6   5   4   3   2   1   0'
fmsg            db     'The feature bits are:'

; --- write string
write           proc    near
                mov     ah,13h
                mov     bh,0
                mov     bl,07h
                mov     al,1
```

```
                int     10h
                inc     dh
                ret
write           endp

; --- print feature bits
print           proc    near
                mov     ah,0Eh
                int     10h
                inc     dl
                mov     cx,03h
print1:         mov     al,' '
                mov     ah,0eh
                int     10h
                inc     dl
                loop    print1
                ret
print           endp

clr_screen      proc    near
                mov     ax,0F00h
                int     10h
                xor     ah,ah
                int     10h
                ret
clr_screen      endp

start:          call    clr_screen
                mov     dx,0000h        ;set cursor to 0,0
                mov     ah,12h          ;EGA BIOS call to
                mov     bl,10h          ;   return EGA information
                int     10h
                mov     mem,bl          ;save memory value in mem
                mov     swset,cl        ;save switch setting in swset
                mov     fbits,ch        ;save feature bits in fbits

; --- check mode
                cmp     bh,0
                je      color           ;if bh=0 then color mode
                mov     bp,offset modem ;if bh=1 then mono mode
                mov     cx,25           ;save offset and length of string
                call    write           ;write message
                jmp     label1
color:          mov     bp,offset modec ;offset of string
                mov     cx,20           ;length of string
                call    write           ;write message
; --- check memory amount
label1:         cmp     mem,1
                je      mem11           ;if mem=1 then 128k
                jl      mem10           ;if mem=0 then 64k
                cmp     mem,2
                je      mem12           ;if mem=2 then 192k
                mov     bp,offset MEM3  ;256k
                mov     cx,21
                call    write
                jmp     label2
mem10:          mov     bp,offset MEM0  ;64k
                mov     cx,20
                call    write
                jmp     label2
mem11:          mov     bp,offset MEM1  ;128k
```

```
                mov     cx,21
                call    write
                jmp     label2
meml2:          mov     bp,offset MEM2  ;192k
                mov     cx,21
                call    write

; --- check switch setting
label2:         mov     ah,2            ;set cursor at 2,0
                mov     dx,200h
                mov     bh,0
                int     10h
                mov     bp,offset swmsg ;write
                mov     cx,17           ;  switch setting
                call    write           ;      message
                mov     si,word ptr swset       ;move switch setting to SI
                and     si,0fh          ;keep the lower 4 bits
                mov     ah,0eh          ;write
                mov     al,switch[si]   ;  the setting using
                int     10h           ;        teletype

; --- check feature bits
label3:         mov     bp,offset fmsg  ;write
                mov     cx,21           ;  feature bits
                call    write           ;     message
                mov     bp,offset strg1
                mov     cx,29
                mov     ah,13h
                mov     bh,0
                mov     bl,9h
                mov     al,1
                int     10h
                mov     ah,2            ;set cursor to 5,0
                mov     dx,500h
                mov     bx,0
                int     10h
                mov     cx,8            ;set loop counter
loop1:          push    cx              ;store loop counter
                mov     al,fbits
                shl     al,1            ;shift left through carry once
                mov     fbits,al
                jc      print1          ;if carry print out a 1
                mov     al,30h          ;if no carry print out a 0
                call    print
                pop     cx
                loop    loop1           ;loop if not zero
                jmp     exit            ;end to DOS
print1:         mov     al,31h
                call    print
                pop     cx
                loop    loop1           ;loop if not zero
exit:           INT     20h             ; return to DOS

CODE            ENDS
                END     begin
```

This program has AH=12h and BL=20h which selects an alternate print screen routine. This program sets up 43 lines of text and then sets the print screen routine to print all 43 instead of just 25. This BIOS call will set up the print screen routine so that all the lines of text on the screen will be printed.

Note: This call should be invoked any time ROWS is changed.

```
        title   AH=12, BL=20
;
print   macro   addr
        ifdif   <addr>,<dx>
         mov     dx,offset addr
        endif
        mov     ah,09
        int     21h
        endm
;
        .radix  16
code    segment para 'code'
        org     100h
        assume  cs:code,ds:code
begin   proc    near
        jmp     Start

; --- Parameters
msg8    db      ' DOS 2.0+ is required.',0Dh,0Ah,'$'
msg7    db      ' Error encountered.','$'
msg1    db      'BIOS Call AH=12, BL=20',0Dh,0Ah,'$'
msg2    db       0Dh,0Ah,'$'
        db      '       '
msgn    db      ' ','$'
msg10   db      '01234567890123456789012345678901234567890123456789'
        db      '01234567890123456789012345678901234567890123456789',
                0Dh,0Ah
        db      'LINE #2',0Dh,0Ah
        db      'LINE #3',0Dh,0Ah
        db      'LINE #4',0Dh,0Ah
        db      'LINE #5',0Dh,0Ah
        db      'LINE #6',0Dh,0Ah
        db      'LINE #7',0Dh,0Ah
        db      'LINE #8',0Dh,0Ah
        db      'LINE #9',0Dh,0Ah
        db      'LINE #10',0Dh,0Ah
        db      'LINE #11',0Dh,0Ah
        db      'LINE #12',0Dh,0Ah
        db      'LINE #13',0Dh,0Ah
        db      'LINE #14',0Dh,0Ah
        db      'LINE #15',0Dh,0Ah
        db      'LINE #16',0Dh,0Ah
        db      'LINE #17',0Dh,0Ah
        db      'LINE #18',0Dh,0Ah
        db      'LINE #19',0Dh,0Ah
        db      'LINE #20',0Dh,0Ah
        db      'LINE #20',0Dh,0Ah
        db      'LINE #21',0Dh,0Ah
        db      'LINE #22',0Dh,0Ah
        db      'LINE #23',0Dh,0Ah
        db      'LINE #24',0Dh,0Ah
        db      'LINE #25',0Dh,0Ah
        db      'LINE #26',0Dh,0Ah
        db      'LINE #27',0Dh,0Ah
        db      'LINE #28',0Dh,0Ah
        db      'LINE #29',0Dh,0Ah
        db      'LINE #30',0Dh,0Ah
        db      'LINE #31',0Dh,0Ah
        db      'LINE #32',0Dh,0Ah,'$'
msg21   db      'BIOS TEST for mode AH = 12',0Dh,0Ah
        db      '                    BL = 20',0Dh,0Ah,'$'
msg20   db      '  Alt. print screen routine installed',
                0Dh,0Ah,'$'
msg3    db      ' ','$'
msg22   db      '  Press <SHIFT> PrtSc to test',0Dh,0Ah
        db      '  Press any other key to continue........',
                '$'
```

```
help    db      'BIOSTEST  AH=12, BL=20',0Dh,0Ah,'$'
;       db      ' syntax: A>ah12a ',0Dh,0Ah
feature db      0
switch  db      0
cmode   db      0
cpage   db      0
ncol    db      0
        db      1Ah

;
; --- clear screen
cscreen proc    near
        mov     ax,0F00h
        int     10h
        xor     ah,ah
        int     10h
        ret
cscreen endp
;
;---Main Routine----------------------------------------------------------
; --- Check DOS version for 2.0+
Start:  nop
        mov     ah,30h          ; DOS version
        int     21h
        cmp     al,02h          ; chk pre 2.0
        jge     bt0             ; ge: ok, else
        print   msg8            ; apologize
        ret


; --- Make appropriate BIOS call for the test
bt0:    call    cscreen
        print   msg1            ; say hello
        print   msg2
;
; --- Save current cmode
        mov     ax,0F00h
        int     10h
        mov     cmode,al
;
; --- Set up 43 line mode print screen now needs modification
        mov     ax,1112h
        mov     bl,0h
        int     10h
;
; --- Print a screen full of data
        print msg10
;
; --- BIOS call to set alternate print screen routine to track 43 lines
        print   msg2
        print   msg21
        print   msg2
        mov     ax,1200h
        mov     bl,20h
        int     10h
        print msg20
        print msg22
;
; --- DOS keyboard check
malb:   mov     ah,0Bh          ; check input status
        int     21h
```

```
        cmp     al,0FFh
        jne     malb            ; no character so loop
        mov     ah,07h          ; character found so
        int     21h             ;   take character to clear buffer
;
; --- Restore previous mode
        mov     ah,0h
        mov     al,cmode
        int     10h
        ret
;
begin   endp
;
code    ends
        end     begin
```

Write String

The new BIOS call AH=13h provides a software interface to the new TTY Write String function for the IBM Enhanced Graphics Adapter. The following is a listing of this BIOS interface as it is published in the Options and Adapters Technical Reference manual.

```
(AH) = 13   WRITE STRING
                ES:BP - POINTER TO STRING TO BE WRITTEN
                CX    - CHARACTER ONLY COUNT
                DX    - POSITION TO BEGIN STRING, IN CURSOR
                        TERMS
                BH    - PAGE NUMBER

        AL = 0
                BL    - ATTRIBUTE
                STRING - (CHAR, CHAR, CHAR, ...)
                CURSOR NOT MOVED
        AL = 1
                BL    - ATTRIBUTE
                STRING - (CHAR, CHAR, CHAR, ...)
                CURSOR  IS MOVED
        AL = 2
                STRING - (CHAR, ATTR, CHAR, ATTR, ...)
                CURSOR NOT MOVED
        AL = 3
                STRING - (CHAR, ATTR, CHAR, ATTR, ...)
                CURSOR  IS MOVED

        NOTE : CHAR RET, LINE FEED, BACKSPACE, AND BELL ARE
               TREATED AS COMMANDS RATHER THAN PRINTABLE
               CHARACTERS.
```

Figure 20. Write String BIOS Interface AH = 13h

The following programs show an example of the EGA BIOS call with AH = 13h. For this call:

- ES should contain the segment of the string to be written;
- BP should contain the offset of the string to be written;
- CX should contain the count of characters in the string;

- DX should contain the cursor position to start the string (DH contains the row and DL contains the column);
- BH should contain the page number.

The following example is of AL=0. This BIOS call writes the string without moving the cursor. The cursor will be at the position of the first character written. BL contains the attribute of the string to be written.

```
        title   WRITE STRING  AH=13h  AL=0
code    segment para 'code'
        org     100h
        assume  cs:code,ds:code,es:code
begin   proc    near
        jmp     start

mml             db      strg1_end-string1-1
string1         label   byte
                db      '1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ'

strg1_end       equ     $

current_page    db      00h

clr_screen      proc    near
        mov     ax,0F00h
        int     10h
        xor     ah,ah
        int     10h
        ret
clr_screen      endp
start:
        call    clr_screen
        mov     ax,ds
        mov     es,ax                   ; segment of string
        mov     bp,offset string1       ; pointer to string
        mov     bx,offset mml           ; strg1_end
        mov     si,0
        xor     ch,ch
        mov     cl,[bx]                 ; character count
        inc     cl
        mov     dx,0                    ; cursor position
        mov     bh,0                    ; current_page
        mov     al,0
        mov     bl,17h                  ; attribute
        mov     ah,13h                  ; write string
        int     10h

RETURN_DOS:
        int     20h                     ; return to DOS
begin   endp
code    ends
        end     begin
```

When AL is set to a 1, the same thing occurs as when AL=0 except the cursor moves. The cursor will be at the position after the last character written. BL contains the attribute of the string.

When AL = 2 or 3, the difference between that and when AL=0 or 1 is that the string is set up with alternating character and attribute bytes. BL is not needed since each character is given its own attribute. For AL=2, the cursor will be at the position of the first character written. For AL=3, the cursor will be at the position after the last character written. CX contains the count of characters only, not the total length of the string.

The string for AL=2 and AL=3 is set up as follows:

```
string2     label   byte
            db      '1',01h,'2',02h,'3',03h,'4',04h,'5',05h
            db      "6',06h,'7',07h,'8',08h,'9',09h,'0',010h
            db      'A',01Ah,'B',0Bh,'C',0Ch,'D',01Dh,'E',0Eh
            db      'F',0Fh,'G',010h,'H',021h,'I',032h,'J',043h
            db      'K',054h,'L',065h,'M',076h,'N',087h,'O',098h
            db      'P',0A9h,'Q',0BAh,'R',0CBh,'S',0DCh,'T',0EDh
            db      'U',0FEh,'V',00Fh,'W',020h,'X',030h,'Y',040h
            db      'Z',050h
```

This will print the same string, '1-Z', with each character having a different attribute.

EGA Programming Techniques

Pel Scrolling and Panning

The EGA card supports smooth scrolling (pel scrolling or panning) in both the horizontal and vertical directions. This function is provided by the CRT Controller and the attribute VLSI chip and operates in both alphanumeric and graphic modes.

The following registers on the EGA card are used in panning:

CRTC	0C/0D	start address register
	13	offset register
	08	preset row scan register
ATTR	13	horizontal pel panning register

In general, panning is a two-step process. First the CRTC is configured for a larger memory size than is displayable at one time. This is accomplished with the offset register, which specifies the logical screen width (normally set greater than or equal to the physical width). The start address register is loaded with the address of the upper left corner of the display screen (see Figure 23 on page 50).

To actually pan or scroll the screen, the upper left corner position is updated in sync with the vertical retrace. Panning repositions the physical display window on the logical screen; moving the display window to the right appears to move the picture to the left.

- Moving the display horizontally

  The start address register specifies the byte (8 pels) of the upper left corner of the display screen; the pel panning value tunes the position to an individual pel. The horizontal pel panning register is used to move it a fraction of a byte (0-7 pels).

- Moving the display vertically

  Vertical scrolling is dependent on whether the EGA is in an alphanumeric or graphic mode.

  In graphic modes the start address register points to the start of the row of pixels that make up the top of the display screen.

  In alphanumeric modes, the start address register points to the first character of the text line at the top of the display, and the preset row scan register specifies the starting row scan (how many scan lines into the character cell) at which to start the actual display.

  The preset row scan register should be set to zero for graphics applications.

  Because the CRTC supports character cells up to 32 scan lines, the preset row scan register typically varies from 0 to the current character height (see maximum scan line register CRTC register #09).

The CRTC latches the start address register at the start of vertical retrace. The preset row scan register is also referenced at this time. These values should be loaded during active video time. The attribute chip uses the current horizontal pel panning value; this register should be loaded during vertical retrace to avoid an unpredictable display.

As an example, consider a logical screen size of 800 pels horizontally and 600 lines vertically. This requires 480,000 pels, which at 4 bits/pel translates to 235K (the EGA card supports up to 256K) or 60,000 bytes per map. If the EGA card is configured for 320 by 200 16-color graphics mode, there are three full screens vertically and 2 1/2 screens horizontally. A width of 800 pels is an offset of 100 bytes. The offset register would be loaded with the value 50.

The logical screen size is limited by a 64K segment (the 800 by 600 example uses 59K). Because the memory requirements of alphanumeric modes are much less than graphics modes, much larger logical screens can be used in alphanumeric modes.

The following assembly language subroutine pans the screen to a specified pel coordinate. The coordinate system used assumes the origin is the upper left corner of the display. The x and y coordinates are passed in CX and AX respectively.

Note: The variable SCREEN_WIDTH contains the logical width of the buffer in bytes, not pels.

```
screen_width dw 100             ; 800 pels wide

pan     proc    near
; --- convert coordinate to byte address, pel value
        mul     screen_width    ; offset to scan line
        mov     bx,ax
        mov     ax,cx
        shr     ax,1
        shr     ax,1
        shr     ax,1            ; offset into scan line
        add     bx,ax           ; bx is start address
        and     cl,07h          ; cl is pel panning value

; --- get in sync w/ vertical retrace
        mov     dx,03DAh
        cli                     ; disable interrupts
pa1:    in      al,dx           ; video status
        test    al,00001000b    ; look for retrace
        jz      pa1             ; off: try again
pa2:    in      al,dx           ; video status
        test    al,00000001b    ; look for active
        jnz     pa2             ; retrace: try again

; --- during active video- set start register
        mov     dx,03DAh
        mov     al,0Ch          ; start address high
        mov     ah,bh
        out     dx,ax
        mov     al,0Dh          ; start address low
        mov     ah,bl
        out     dx,ax

; --- wait for vertical retrace
        mov     dx,03DAh
pa3:    in      al,dx           ; video status
        test    al,00001000b    ; look for retrace
        jz      pa3             ; off: try again

; --- in vertical retrace- set pel pan
        mov     dx,03C0h
        mov     al,33h          ; pel pan reg (w/ video enable)
        out     dx,al
        mov     al,cl           ; pel pan count
        out     dx,al
        sti                     ; enable interrupts
        ret
pan     endp
```

Figure 21 on page 32 shows a logical view of the upper left corner of the display on a pel boundary. The line indicates the exclusive boundary of the displayable screen. In this example the horizontal pel panning register would be set to 4 and the preset row scan register would be 3. The start address register would point to the byte which contains the character code for "A".

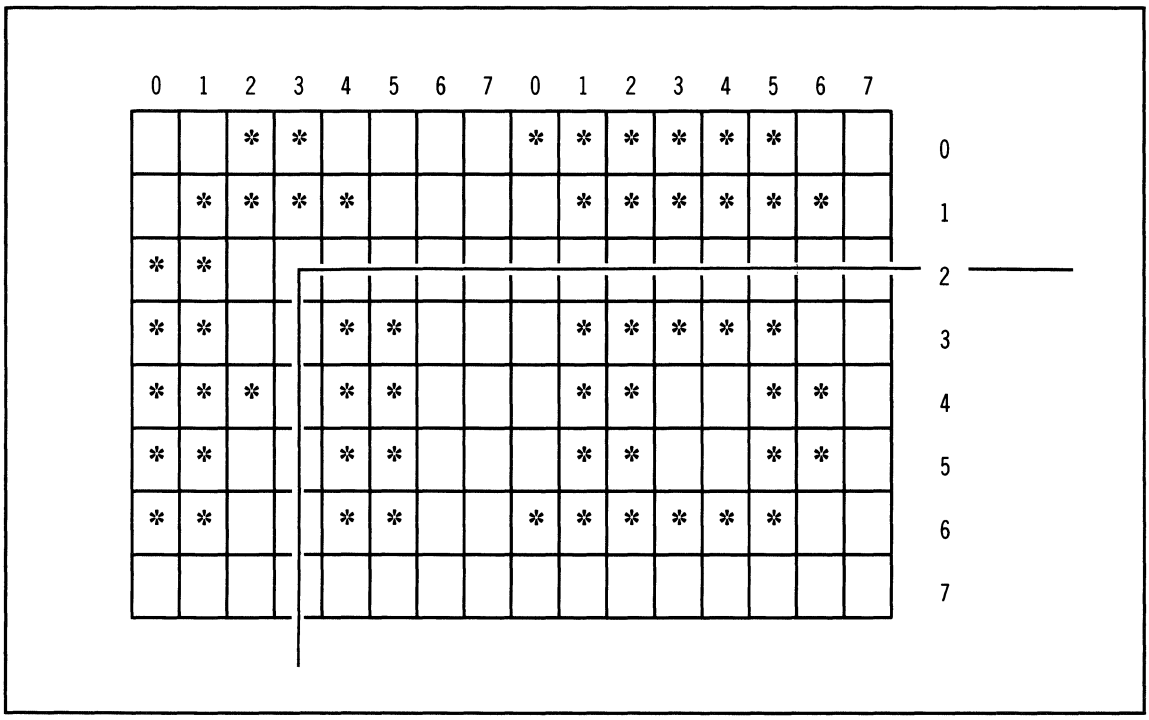

Figure 21. Horizontal and Vertical Panning

Alternate Parameter Tables

SAVE_PTR is a double word pointer at low memory address 04ABh. It points to a structure of 7 double words (28 bytes) that BIOS uses to maintain various tables and save areas. The following is a listing of the saveptr definition as it is published in the Options and Adapters Technical Reference manual.

```
     org    04A8h
save_ptr    label   dword


save_ptr is a pointer to a table as described as follows:

dword_1     video parameter table pointer
dword_2     dynamic save area pointer
dword_3     alpha mode auxiliary char gen pointer
dword_4     graphics mode auxiliary char gen pointer
dword_5     reserved
dword_6     reserved
dword_7     reserved

dword_1     Parameter Table Pointer
            Initialized to BIOS EGA parameter table.
            This value MUST exist.

(continued on next page)
```

Figure 22. SAV_PTR Format Listing

```
                        (continued from previous page)

dword_2     Parameter Save area pointer
            Initialized to 0000:0000, this value is optional.
            When non-zero, this pointer will be used as pointer
            to a RAM area where certain dynamic values are to
            be saved. When in EGA operation this RAM area will
            hold the 16 EGA palette register values plus
            the overscan value in bytes 0-16d respectively.
            At least 256 bytes must be allocated for this area.

dword_3     Alpha Mode Auxiliary pointer
            Initialized to 0000:0000, this value is optional.
            When non-zero, this pointer is used as a pointer
            to a tables described as follows:

            byte    bytes/character
            byte    block to load, should be zero for normal
                    operation
            word    count to store, should be 256d for normal
                    operation
            word    character offset, should be zero for normal
                    operation
            dword   pointer to a font table
            byte    displayable rows
                    if 'FF' the maximum calculated value will
                    be used, else this value will be used
            byte    consecutive bytes of mode values for which
                    this font description is to be used.
                    The end of this stream is indicated by a
                    byte code of 'FF'

dword_4     Graphics Mode Auxiliary pointer
            Initialized to 0000:0000, this value is optional.
            When non-zero, this pointer is used as a pointer
            to a table described as follows:

            byte    displayable rows
            word    bytes per character
            dword   pointer to a font table
            byte    consecutive bytes of mode values for which
                    this font description is to be used.
                    The end of this stream is indicated by a
                    byte code of 'FF'

dword_5 thru dword_7
            Reserved and set to 0000:0000.
```

Figure 22. SAVPTR Format Listing (continued from previous page)

By changing SAVE_PTR to point to a different table, other sets of parameters may be made resident in BIOS. For example, the 8X8 character set that gives a 43-line screen could be made resident.

The following program shows how to make this 43-line screen resident for mode 2.

Note: Modification of video parameters may cause unpredictable results such as loss of video synchronization.

```
        title   set 43 line screen and 8X8 font
code    segment para 'code'
        org     100h
        assume  cs:code,ds:code
begin   proc    near
        jmp     Start


; --- save pointer tables
mark    db      'set43',1Fh,'resident',0,0
table   dw      14 dup (0)
old     dw      14 dup (0)

; --- alpha mode auxiliary pointer
alpha   db      8               ; byte/character
        db      0               ; load block 0
        dw      256             ; 256 characters
        dw      0               ; 0-FF
        dw      font1,0         ; font pointer
        db      43              ; 43 rows on screen
mode    db      2               ; this is for mode 2
        db      -1              ; end of mode list


; - Main Routine ------------------------------------
;--- copy current save pointer table
Start:  xor     ax,ax           ; video BIOS
        mov     ds,ax           ;  data segment
        mov     bx,4A8h         ; save_ptr
        lds     si,[bx]         ; current table
        push    si              ; current offset
        mov     di,offset old   ; old table copy
        mov     cx,14d          ; 14 words
   rep  movsw                   ; copy
        pop     si              ; current offset
        mov     di,offset table ; local table
        mov     cx,14d          ; 14 words
   rep  movsw                   ; copy
        push    cs              ; re-establish
        pop     ds              ; local segment

; --- change alpha block pointer
        mov     bx,offset table
        mov     word ptr [bx+8],offset alpha
        mov     word ptr [bx+0Ah],cs
        mov     bx,offset alpha         ; alpha table
        mov     word ptr [bx+8],cs      ; font segment

; --- change video parameters pointer
        mov     bx,offset table
        mov     word ptr [bx+00h],offset parms
        mov     word ptr [bx+02h],cs
```

```
; --- update pointer to new table
        xor     ax,ax           ; BIOS video
        mov     ds,ax           ;  data segment
        mov     bx,4A8h         ; save_ptr
        cli                     ; disable interrupts
        mov     word ptr [bx],offset table
        mov     word ptr [bx+2],cs
        sti                     ; enable interrupts

; --- disable cursor emulation
        mov     bx,487h         ; info byte
        or      byte ptr bx,1 ; turn off emulation

; --- mark resident label string
        push    cs              ; re-establish
        pop     ds              ; local segment
        mov     bx,offset mark  ; label string
        mov     [bx+14],0F5F3h  ; 35 ebcdic

; --- enable changes
        mov     al,mode         ; mode
        mov     ah,00h          ; set mode
        int     10h

; --- terminate & remain resident
        mov     dx,offset limit ; resident size
        int     27h
begin   endp

; --- 8x8 font specification
font1   label   byte
        include ufont1.i
parms   label   byte
        include vparms.i
limit   equ     $

code    ends
        end     begin
```

Note: The file VPARMS.I is a file that contains the listing for vparms as published in the BIOS listing in the Options and Adapters Technical Reference manual. The file UFONT1.I is the same as in the example in the section on Mode 11h.

Mode Switching

Switching between modes is slightly more complicated with the IBM Enhanced Graphics Adapter since there are several new modes. In order to maintain compatibility with programs designed for the IBM Color Graphics/Monitor Adapter and the IBM Monochrome Display and Printer Adapter, bits 5 and 6 of the equipment flag (low memory 0410h) must be set properly for the respective mode.

Upon installation of the EGA card, the display switches (5 and 6 of the system configuration DIP switch SW1 on the system board of the IBM Personal Computer PC and IBM Personal Computer XT) are both set to ON. The Power On Self Test tests the system to determine what display adapters are installed. If an EGA is found, the POST reads the DIP switches on the EGA to determine what kind of display is attached and whether or not the EGA is the primary display. If a color mode is primary, bits 5 and 6 of the equipment flag are set to ON OFF for 80-column color and OFF ON for 40-column color. If a monochrome mode is primary, these bits are set to OFF OFF.

In order to switch modes between adapters reliably, the equipment flag must be set for the destination adapter before BIOS is called.

Examples

The following code is an example of switching between Color mode 3 and Monochrome mode 7.

```
        title   goto monochrome mode 7
code    segment para 'code'
        org     100h
        assume  cs:code,ds:code
begin   proc    near
        jmp     Start
;
; - Main Routine ------------------------------------
;--- set equipment flag
Start:  push    es              ; save es
        xor     ax,ax           ; clear ax
        mov     es,ax           ; point to low memory
        mov     bl,030h         ; set flag data
        mov     bh,es:410h      ; read equipment flag
        and     bh,not 30h      ; clear these 2 bits
        or      bh,bl           ; set these 2 bits
        mov     es:410h,bh      ; change bits
;
;--- perform BIOS mode set
        mov     ax,07h          ; set up for mode 7
        int     10h             ; make BIOS call
        pop     es              ; restore es
        ret
begin   endp
;
code    ends
        end     begin
```

The following code is an example of switching between Monochrome mode 7 and Color mode 3.

```
        title   goto color mode 3
code    segment para 'code'
        org     100h
        assume  cs:code,ds:code
begin   proc    near
        jmp     Start
;
; - Main Routine ------------------------------------
;--- set equipment flag
Start:  push    es              ; save es
        xor     ax,ax           ; clear ax
        mov     es,ax           ; point to low memory
        mov     bl,020h         ; set flag data
        mov     bh,es:410h      ; read equipment flag
        and     bh,not 30h      ; clear these 2 bits
        or      bh,bl           ; set these 2 bits
        mov     es:410h,bh      ; change bits
```

```
;--- perform BIOS mode set
        mov     ax,03h          ; set up for mode 3
        int     10h             ; make BIOS call
        pop     es              ; restore es
        ret
begin   endp
;
code    ends
        end     begin
```

Presence Test

The following procedure is only one of many ways to determine the presence of IBM video adapters. It should be used as a guideline in writing the appropriate presence test for a specific application.

General Procedure:

1. Determine if EGA is present.

   There are many ways to do this. One way is to read the Equipment Flag (40:10) in low System RAM. Then test bits 5 and 6 for '00'. If '00', then either the EGA is installed or no display adapter is attached.

2. Making the assumption that bits 5 and 6 are '00' and that the EGA is present, read the low RAM byte called 'INFO' (40:87) to determine which mode the EGA is in. This is done by testing bit 1. If bit 1 = 0, the EGA has a color display attached; if bit 1 = 1, the EGA has a monochrome display attached.

3. After determining which type of display is attached, look for the opposite type adapter. Example: If the EGA has a monochrome display attached, look for a color display adapter.

4. If the EGA is not present, look for other IBM display adapters. An easy way to determine if a non-EGA adapter is present is to write and read the cursor registers on the card. If the data read is equal to the data written, the card is present. Care should be used to store the present value before the test is performed. The registers used in the sample program are 3D4h = 0Fh for Color and 3B4h = 0Fh for monochrome.

5. Which adapter is active?

   Using EGA video BIOS call (AH) = 0Fh, the current state of the adapter is returned. The value is returned in AL and is always the current (active) mode.

   To determine if the EGA is the active display, read the low system RAM byte called 'INFO' (40:87), and test bit 3. If bit 3 = 0, the EGA is the active monitor; if bit 3 = 1, the EGA is not active.

6. Which adapter is primary and/or secondary? To determine if the EGA is the primary display, use the video BIOS call AH = 12h (Alternate Select) with BL = 10h. The return information is in CL, which contains the EGA switch settings. If the value in the CL register is 5 or less, the EGA is the secondary adapter. If the value in the CL register is more than 5, the EGA is the primary adapter.

Note: When an EGA is not present in the system and two adapters are present, it is always assumed that the IBM Monochrome Display and Printer Adapter is primary.

```
        title   PRESTEST
;
print   macro   addr
        ifdif   <addr>,<dx>
         mov     dx,offset addr
        endif
        mov     ah,09h
        int     21h
        endm
;
        .radix  16
code    segment para 'code'
        org     100h
        assume  cs:code,ds:code
begin   proc    near
        jmp     start


; --- Parameters
msg8    db      ' PRESTEST requires DOS 2.0+',0Dh,0Ah,'$'
msg7    db      ' Error encountered.','$'
msg1    db      ' PRESTEST ',0Dh,0Ah,'$'
msg2    db      0Dh,0Ah,'$'
        db      '       '
msgn    db      ' ','$'
cmsgp   db      ' Color/Graphics Monitor Adapter is Present',0Dh,0Ah,'$'
crlf    db      0DH,0Ah,'$'
mmsgp   db      ' Monochrome Display Adapter is Present',0Dh,0Ah,'$'
pgcp    db      ' Professional Graphics Controller is Present'
        db      0Dh,0Ah,'$'
msgwait db      ' Press any key to continue.....','$'
egap    db      ' Enhanced Graphics Adapter is Present','$'
prim_msg     db     ' is the Primary Display',0Dh,0Ah,'$'
secn_msg     db     ' is the Secondary Display',0Dh,0Ah,'$'
active_msg      db     ' and is Active',0Dh,0Ah,0Dh,0Ah,'$'
not_active_msg  db     ' and is Not Active',0Dh,0Ah,0Dh,0Ah,'$'
help    db      ' PRESTEST ',0Dh,0Ah,'$'
;       db      ' syntax: A>PRESTEST ',0Dh,0Ah
        db      ' feature..'
feature db      0
        db      ' switch..'
switch  db      0
        db      ' corm..'
corm    db      0
        db      ' mem..'
mem     db      0
cmode   db      0
cpage   db      0
ncols   db      0
ecount  db      0

        db      1Ah

base    dw      10              ; decimal
```

```
;  Subroutines
;
;
; --- clear screen
cscreen proc    near
        mov     ax,0F00h
        int     10h
        xor     ah,ah
        int     10h
        ret
cscreen endp



;----CK_CURSOR_REG-----------------------------------
;
;  This routine tests the cursor to verify the presence
;  of the adapter.
;       INPUT:  DX = port address of card (3?4h, where
;                    B= monochrome and D=color)
;       OUTPUT: AL = 1 if present
;
;-------------------------------------------------------
ck_cursor_reg   proc    near
        mov     al,0Fh          ; set crtc addr to cursor reg
        out     dx,al
        inc     dx
        in      al,dx           ; save original value
        mov     bh,al
        mov     al,5Ah          ; set test value
        out     dx,al
        jmp     $+2
        jmp     $+2
        jmp     $+2
        in      al,dx
        cmp     al,5Ah
        mov     al,bh           ; restore original value
        out     dx,al
        jne     ck_cursor_reg_not_present
        mov     ax,1
        ret
ck_cursor_reg_not_present:
        xor     ax,ax
        ret
ck_cursor_reg   endp

get_equip_value proc    near
        push    es
        xor     ax,ax
        mov     es,ax           ; point into low memory
        mov     si,0410h        ; offset to equipment flag
        mov     al,es:[si]      ; get data
        and     al,030h         ; mask off display bits
        pop     es
        ret
get_equip_value endp
```

```
;--------------------------------------------------------
;
;  Presence test for the professional graphics controller
;  procedure:
;       memory write to C600:03D4h
;       I/O read from port 3D4
;       compare if identical then PGC is emulating CGA
;
;--------------------------------------------------------
port_3D4        equ     03D4h   ; CGA/PGC common port
pgcl_seg        equ     0C600h  ; PGC ram segment
test_pattern    equ     028h    ; test data for PGC test

pres_test_PGC_emulator        proc    near
        push    es
        push    ax
        push    dx
        push    bx
        mov     dx,pgcl_seg
        mov     es,dx           ; set PGC segment pointer
        mov     bx,port_3D4     ; point to PGC memory location
        nop                     ; save the current contents
        mov     ah,byte ptr es:[bx]
        nop                     ; write test data
        mov     byte ptr es:[bx],test_pattern
        mov     dx,port_3D4     ; point to i/o mem wrap
        in      al,dx           ; get current contents
        mov     es:[bx],ah      ; restore PGC memory location
        pop     bx
        pop     dx
        cmp     al,test_pattern ; compare written to read
        pop     ax
        pop     es
        ret
pres_test_PGC_emulator        endp
;----------------

get_info        proc    near
        push    es
        xor     ax,ax
        mov     es,ax           ; point into low memory
        mov     si,0487h        ; offset to EGA Info byte
        mov     al,es:[si]      ; get data
        pop     es
        ret
get_info        endp
```

```
;--------------------------------------------------------
;   Presence test for the professional graphics controller
;   procedure:
;        memory write to C600:03DBh
;        memory read from port 3D4h
;        compare if identical then PGC is present
;--------------------------------------------------------
pgc_pres_test   proc    near
        push    ax
        push    es
        mov     ax,0c600h
        mov     es,ax
        mov     si,3dbh
        mov     bh,byte ptr es:[si]     ; save original value
        mov     byte ptr es:[si],5ah    ; set test value
        mov     al,byte ptr es:[si]
        cmp     al,5ah
        mov     byte ptr es:[si],bh     ; restore original value
        pop     es
        pop     ax
        jne     pgc_not_present
        mov     bx,1            ; return a 1 in bx if PGC is present
        ret
pgc_not_present:
        xor     bx,bx           ; return a 0 in bx if PGC is not present
        ret
pgc_pres_test   endp

;---Main Routine-----------------------------------------
;
;   This test is designed to determine the hardware
;   configuration and state of video display hardware in the
;   IBM Personal Computer, IBM Personal Computer XT, and
;   IBM Personal Computer AT.  Valid devices for this test
;   include the IBM Monochrome Display and Printer Adapter,
;   IBM Color/Graphics Monitor Adapter, IBM Enhanced Graphics
;   Adapter, and IBM Professional Graphics Controller. Other
;   configurations of non-IBM display adapters are not
;   supported and may cause this program to return erroneous
;   results. This should not be used as a test for
;   compatibility for other display devices.
;
;--------------------------------------------------------
;
; --- Check DOS version for 2.0+
start:  nop
        mov     ah,30h    ; DOS version
        int     21h
        cmp     al,2      ; chk pre 2.0
        jge     pt0       ; ge: ok, else
        print   msg8      ; apologize
        ret
```

```
; --- Make appropriate BIOS call for the test

pt0:    call    cscreen   ; clear screen
        print   msg1      ; Say hello
        print   msg2
;
; --- Save current cmode
        mov     ax,0F00h  ; get present video mode
        int     10h       ; video interrupt
        mov     cmode,al  ; save mode
        mov     ncols,ah  ; save number of columns
        mov     cpage,bh  ; save active page
;
; --- Test for enhanced BIOS Functions
;   This assumes that only ax will return modified if EGA BIOS is present
;
        mov     ax,1200h  ; Set ax for BIOS call (Alternate Select)
        mov     bl,10h    ; Set bl for Return EGA Information
        mov     bh,0FFh   ; Load BH with invalid info for test
        mov     cl,0Fh    ; Load CL with reserved switch setting
        int     10h       ; Call video BIOS
;
; --- Store returned values
;
        mov     corm,bh   ; Save color or mono bit
        mov     mem,bl    ; Save memory size bits
        mov     switch,cl ; Save EGA switch setting
;
; --- Test for impossible values
;
        cmp     cl,0Ch    ; test reserved switch settings
        jl      pt2
        inc     ecount    ; increment error count
        mov     al,switch
pt2:    cmp     bh,01h    ; check info range 0-1
        jle     pt3
        inc     ecount    ; increment error count
        mov     al,corm
pt3:    cmp     bl,03h    ; check range 0-3
        jle     pt4
        inc     ecount    ; increment error count
        mov     al,mem
pt4:    nop
        mov     al,ecount ; check error total
;
        cmp     ecount,0h ; If no errors, we have found an EGA
        jne     NEGA      ; no EGA present
        jmp     YEGA      ; EGA present
```

```
;
;---NOEGA-----------------------------------------------
;
;   No EGA was found in the system
;   Check for other display adapters
;
;-------------------------------------------------------

NEGA:   nop

        mov     dx,3B4h         ; is a mono adapter in
        call    ck_cursor_reg   ; check presence using cursor register
        cmp     ax,0            ; was it active
        jz      lk_for_color    ; no, look for color type adapter
        print   mmsgp           ; yes, print monochrome present
        call    get_equip_value ; get equipment flag from low RAM
        cmp     al,030h         ; was it set for monochrome
        jne     nega_prim_m     ; no,continue elsewhere
        print   prim_msg        ; yes, print active message
nega_prim_m:
        mov     ah,0Fh          ; get current video state
        int     10h             ; video interrupt
        cmp     al,07h          ; is it a monochrome mode
        jne     MONO_NOT_ACTIVE ; no, continue elsewhere
        print   active_msg      ; yes,
        jmp     LK_FOR_COLOR    ; go look for color type adapter
MONO_NOT_ACTIVE:
        print   NOT_ACTIVE_MSG  ; print monochrome adapter not active
lk_for_color:
        mov     dx,3D4h         ; is a color type adapter in the system
        call    ck_cursor_reg   ; check using cursor register
        call    pgc_pres_test   ; check to see if PGC is present
        cmp     ax,0
        jz      no_color        ; no color adapter is found
        call    pres_test_PGC_emulator  ; if active verify adapter is
                                        ; not a professional graphics
        je      pgc_present     ; if PGC is present print message
        print   cmsgp           ; print color adapter present
        mov     ah,0Fh          ; get current video mode
        int     10h             ; video interrupt
        cmp     al,07h          ;
        je      nega_nactive
        print   active_msg      ; print color active
        cmp     bx,0            ; see if PGC is present
        jz      ex              ; exit if not present
        print   pgcp            ; PGC is present
        print   pgc_graphic     ; and in graphics mode
        jmp     exit
nega_nactive:
        print   not_active_msg  ; print color not active
        jmp     chk_pgc
ex:
        jmp     exit
pgc_present:
        print   pgcp            ; PGC present
        print   pgc_emulate     ; and in emulate mode
        jmp     exit
```

```
no_color:
        cmp     bx,0            ; check to see if PGC is present
        jz      ex              ; if not, then exit
        print   pgcp            ; if yes, then in graphics mode
        print   pgc_graphic
        jmp     ex
;
;---YESEGA-------------------------------------------
;
;   An EGA was found in the system
;   Check for other video adapters
;
;-----------------------------------------------------
;
YEGA:   nop
        print   egap            ; print EGA present
        cmp     switch,5        ; check EGA switches for primary
        jbe     ega_secondary_adapter   ; EGA is secondary display
        print   prim_msg        ; print EGA is primary display
        call    get_info        ; test low RAM byte Info for active state
        and     al,08h          ; Mask off display bits
        cmp     al,0h           ; if ega active monitor bit 4 = (0)
        jne     yega_ex         ; bit 4 = 1
        print   active_msg      ; bit 4 = 0 EGA active
        jmp     yega_cont1      ; continue elsewhere
yega_ex:
        print   not_active_msg  ; EGA not active
        jmp     yega_cont1      ; continue elsewhere
ega_secondary_adapter:
        print   secn_msg        ; print EGA secondary adapter
        call    get_info        ; test low RAM Info for active state
        and     al,08h          ; Mask off display bits
        cmp     al,0h           ; if ega active monitor bit 4 = (0)
        jne     yega_cont       ; bit 4 = 1 ; EGA not active
        print   active_msg      ; bit 4 = 0 print EGA active
        jmp     yega_CONT1      ; continue elsewhere
yega_cont:
        print   not_active_msg  ; print EGA not active
yega_cont1:
                                ; test color or mono bit
        cmp     corm,0h         ; is EGA set for color
        jnz     look_for_color  ; no,look for color type card
        mov     dx,3B4h         ; yes, look for monochrome adapter
        call    ck_cursor_reg   ; use cursor register for presence
        cmp     ax,0h           ;
        jz      chk_pgc         ; check for a PGC
        print   mmsgp           ; print monochrome adapter present also
yega_next:
        mov     ah,0Fh          ; get current video mode
        int     10h             ; video interrupt
        cmp     al,07h          ; is monochrome mode active
        jne     look_no_more    ; no , print not active
        print   active_msg      ; yes, print monochrome is active
chk_pgc:
        call    pgc_pres_test
        cmp     bx,0
        jz      ex1             ; exit if no PGC
        print   pgcp            ; a PGC is present and in graphics mode
        print   pgc_graphic
        jmp     exit
```

```
look_no_more:
        PRINT   NOT_ACTIVE_MSG
        jmp     chk_pgc
look_for_color:
        mov     dx,3D4h         ; color I/O address
        call    ck_cursor_reg   ; check cursor register for presence
        call    pgc_pres_test   ; check for a PGC
        cmp     ax,0
        jz      no_color1       ; no color adapter is found
        call    pres_test_pgc_emulator  ; verify PGC not in color emulation
        je      yega_pgc_present
        print   cmsgp           ; print Color/Graphics adapter present
        call    get_equip_value ; get low RAM equipment flag
        cmp     al,010h         ; test for color 80x25
        jne     yega_here
        print   active_msg      ; print color active

        cmp     bx,0            ; see if PGC is present
        jz      ex1             ; exit if not present

        print   pgcp            ; PGC is present
        print   pgc_graphic     ; and in graphics mode

ex1:
        JMP     EXIT
yega_here:
        CALL    GET_EQUIP_VALUE ; get low RAM equipment flag
        cmp     al,01h          ; test for color 40x25
        jne     yega_next_1
        print   active_msg      ; print color active
yega_next_1:
        mov     ah,0Fh          ; get current video mode
        int     10h
        cmp     al,07h          ;
        je      yega_ex_1
        print   active_msg
        jmp     chk_pgc
yega_ex_1:
        PRINT   NOT_ACTIVE_MSG
        jmp     chk_pgc
yega_pgc_present:
        print   pgcp            ; PGC present
        print   pgc_emulate     ; and in emulate mode
        jmp     exit

no_color1:
        cmp     bx,0            ; check for a PGC
        jz      exit            ; if no, then exit
        print   pgcp            ; if yes, then in graphics mode
        print   pgc_graphic
        jmp     exit
```

```
; --- DOS keyboard check
;
Exit:   print   msgwait
kbwait: mov     ah,0Bh          ; check input status
        int     21h             ; return back to DOS
        cmp     al,0FFh
        jne     kbwait          ; no character so loop
        mov     ah,07h          ; character found so
        int     21h             ; take character to clear buffer
;
; --- Restore previous mode
        mov     ah,0h           ; do a video mode set
        mov     al,cmode        ; set to previous mode
        int     10h             ; video interrupt
        ret
;
begin   endp
;
code    ends
        end     begin
```

Palette Programming

The following program is an example of palette loading. It uses mode 10h and requires 128k of memory on the IBM Enhanced Graphics Adapter and a IBM Enhanced Color Display. The screen is cleared and 16 boxes of different colors are drawn using write mode 2. The palette registers are updated during vertical retrace. The palette is continually being changed, which gives the impression of the boxes moving. Any key press will return control to DOS.

Note: It is assumed that the EGA is initialized to mode 10h. The boxes are only drawn once.

```
code    segment public 'code'
        assume  cs:code,ds:code
        org  100h
start:  jmp  begin

bar_pos label   word
; column + row*80
        dw   0+140*80
        dw   5+140*80
        dw   10+140*80
        dw   15+140*80
        dw   20+140*80
        dw   25+140*80
        dw   30+140*80
        dw   35+140*80
        dw   40+140*80
        dw   45+140*80
        dw   50+140*80
        dw   55+140*80
        dw   60+140*80
        dw   65+140*80
        dw   70+140*80
        dw   75+140*80

bar_size     label   word
; width in columns + height in rows*256
        dw   5 +30*256

dpal_table   label   byte                    ; default palette
        db   0,1,2,3,4,5,14h,7,38h,39h,3Ah,3Bh,3Ch,3Dh,3Eh,3Fh,0
pal_table    label   byte
        db   0,36,52,54,22,18,3,9,1,8,13,5,36,52,54,22,0     ; palette 1
        db   0,52,54,22,18,3,9,1,8,13,5,36,52,54,22,18,0     ; palette 2
        db   0,54,22,18,3,9,1,8,13,5,36,52,54,22,18,3,0      ; palette 3
        db   0,22,18,3,9,1,8,13,5,36,52,54,22,18,3,9,0       ; palette 4
        db   0,18,3,9,1,8,13,5,36,52,54,22,18,3,9,1,0        ; palette 5
        db   0,3,9,1,8,13,5,36,52,54,22,18,3,9,1,8,0         ; palette 6
        db   0,9,1,8,13,5,36,52,54,22,18,3,9,1,8,13,0        ; palette 7
        db   0,1,8,13,5,36,52,54,22,18,3,9,1,8,13,5,0        ; palette 8
        db   0,8,13,5,36,52,54,22,18,3,9,1,8,13,5,36,0       ; palette 9
        db   0,13,5,36,52,54,22,18,3,9,1,8,13,5,36,52,0      ; palette 10
        db   0,5,36,52,54,22,18,3,9,1,8,13,5,36,52,54,0      ; palette 11

clr_screen   proc    near
        mov  ax,0F00h
        int  10h
        xor  ah,ah
        int  10h
        ret
clr_screen   endp
```

```
graphics_colorbar   proc    near
        push di             ; save DI
        push si             ; save SI
        mov  al,05h         ;
        mov  dx,03CEh       ; graphics write mode register
        out  dx,al          ;
        mov  al,02H         ; change to write mode 2
        mov  dx,03CFh       ;
        out  dx,al          ;
        xor  si,si          ; initialize SI
        mov  al,0h          ;
bar_loop:
        mov  bx,bar_size    ; move bar size to be displayed
        push si
        push ax
        mov  ax,02h         ; move multiplicand into AX
        mul  si             ; multiply SI for table indexing
        mov  si,ax          ; move index value to SI
        mov  bp,bar_pos[si] ; get bar position using index
        pop  ax             ; recover value
        pop  si             ;
        mov  cx,0A000h      ; load regen area into CX
        mov  es,cx          ; set seg reg to video buffer area

        xor  dx,dx          ; clear DX
        mov  dl,bl          ; load row
bar_fill:
        mov  cx,dx          ; use as counter
        mov  di,bp          ;
        rep  stosb
        mov  di,bp
        add  bp,80d
        dec  bh
        jnz  bar_fill
        inc  si
        inc  al
        cmp  si,16d
        jc   bar_loop

        pop  si
        pop  di
        ret
graphics_colorbar       endp
begin:  call clr_screen
        call graphics_colorbar
        mov  bx,0           ; initialize pointer to palette
vert:
        mov  dx,3DAh        ; check
        in   al,dx          ;    for
        and  al,08h         ;       vertical retrace
        jz   vert           ; not in retrace, check again
vert2:  mov  dx,3DAh        ; wait for
        in   al,dx          ;
        and  al,08h         ;       next occurrence
        cmp  al,08h         ;
        je   vert2          ;           of not in retrace
```

```
        mov  ah,10h
        mov  al,02h
        push ds
        pop  es
        mov  dx,offset pal_table
        add  dx,bx
        int  10h            ; set palette
        add  bx,17          ; point to next palette
        cmp  bx,170         ; see if at last palette
        jle  keychk         ; if no, go to keychk
        mov  bx,0           ; if yes, point to first palette

; --- DOS keyboard check
keychk: mov  ah,0Bh         ; check input status
        int  21h
        cmp  al,0FFh
        jne  vert           ; no character so load palette
        mov  ah,07h         ; character found so
        int  21h            ;   exit

exit:   mov  ah,10h
        mov  al,02h
        push ds
        pop  es
        mov  dx,offset dpal_table
        int  10h            ; restore default palette

        call clr_screen

        int  20h            ; return to DOS
code    ends
        end  start
```

Vertical Retrace Interrupt

The EGA card supports a Vertical Retrace Interrupt. Applications that require synchronization with vertical retrace (for example blinking) can make use of the Vertical Retrace Interrupt to handle those tasks asynchronously. The Vertical Retrace Interrupt is on IRQ2 (INT 0Ah).

The CRTC must be programmed to generate an interrupt at Vertical Retrace. Bit 5 of the Vertical Sync end register (11h) enables/disables the vertical interrupt. Bit 4 of the same register clears the interrupt at the CRTC. This bit is set to zero to clear the Vertical Retrace Interrupt at the CRTC. It must be programmed to a one to permit interrupts to occur. Bits 3-0 of the Vertical Sync end register are mode dependent and must have the correct values.

The following assembly language program skeleton shows the steps involved in a vertical retrace interrupt handler.

Note: The value 04h used to clear the vertical interrupt is the value for 200 line alphanumeric modes.

```
; --- field vertical retrace interrupt
        cli                 ; interrupts off
        push    ax
        push    dx

; --- perform function in vertical retrace


; --- clear vertical interrupt at CRTC
        mov     dx,3D4h
        mov     al,11h
        mov     ah,04h      ; clear vertical
        out     dx,ax
        or      ah,10h      ; re-enable
        out     dx,ax

; --- service 8259 priority interrupt controller
        mov     al,20h      ; non specific EOI
        out     20h,al

; --- restore registers
        pop     dx
        pop     ax
        iret
```

Note: Interrupts were disabled at the start of the interrupt handler. This assures that any operations that are to take place during the Vertical Retrace are not interrupted and delayed past the Vertical Retrace interval.

In some hardware environments more than one device may make use of IRQ2. The EGA card does not support hardware chaining (more than one device sharing IRQ2) of interrupts. When more than one device can issue an IRQ2 and Vertical Interrupts are enabled, the Vertical Retrace bit of the input status register (3DAh) will return the IRQ2 (from the system bus) state rather than the Vertical Retrace state of the EGA.

The Vertical Retrace feature of the EGA must be used with care in these situations.

Vertical Split Screens

The IBM Enhanced Graphics Adapter card supports split screening in both alphanumeric and graphic modes. Split screening dynamically merges two areas of the regen buffer on the display; the regen buffer is unchanged by this operation.

The following registers on the EGA card are used in split screening:

CRTC	18	line compare register
	07	overflow register

During normal operation (no split screen) the line compare value should be set to the maximum value (1FFh).

At the start of Vertical Retrace, the CRTC latches the start address register value as the address of the first line of the display. When an internal horizontal scan counter equals the line compare value, the CRT memory address is cleared to 0000h. Thus the CRTC always splits the start of the regen buffer (address 0000h) onto the screen buffer at the line indicated by the line compare register. Operations that take place on either of the buffers (for example scrolling) proceed independently. Thus the lower (or upper) "window" will appear immune to any changes in the upper (or lower) window.

The line compare register contains the low 8 bits of the line compare value; the overflow register, bit 4, contains the ninth bit. The line compare and overflow registers should be written during Vertical Retrace. The split screen window can be smoothly scrolled onto the display by continuously updating these registers in sync with Vertical Retrace.

In Figure 23 on page 50 a sample split screen is shown. The video page at offset 8000h into segment B800h is the current BIOS display page. At line 150 the CRTC splits the video page at offset 0000h onto the screen.

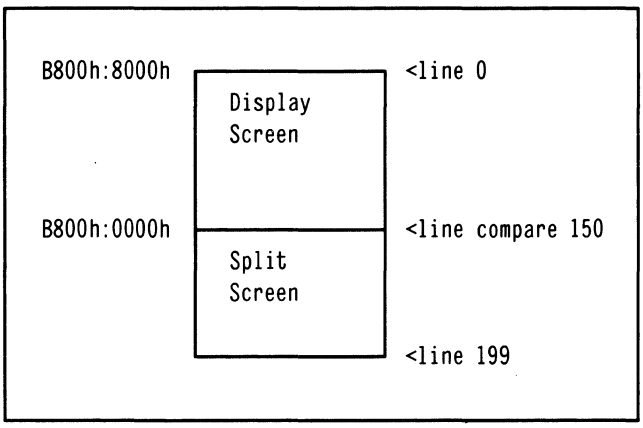

Figure 23. Vertical Split Screen

The following assembly language subroutine loads the line compare and overflow registers during a Vertical Retrace. The number of lines to split onto the screen is passed in CX, and the video mode is assumed to be a 200-line mode. Note that the overflow register is used by several other CRTC registers. Whatever values are programmed into the other overflow bits for the current mode must be preserved by the split screen routine.

```
; --- convert # lines to CRTC parameters
split   proc    near          neg     ax
        sub     ax,200          ; 9 bit line compare value
        mov     cl,al
        mov     ch,01h          ; overflow <256
        test    ah,01h
        jz      Spl
        or      ch,10h          ; set if >255

; --- get in sync w/ vertical retrace
spl:    mov     dx,03DAh
        cli                     ; disable interrupts
sp2:    in      al,dx           ; video status
        test    al,00000001b    ; look for active
        jnz     sp2             ; retrace on: try again
sp3:    in      al,dx           ; video status
        test    al,00001000b    ; look for retrace
        jz      sp3             ; off: try again

; --- in vertical retrace, set CRTC
        mov     dx,03DAh
        mov     al,18h          ; line compare
        mov     ah,cl
        out     dx,ax
        mov     al,07h          ; overflow
        mov     ah,ch
        out     dx,ax
        sti                     ; enable interrupts
        ret
split   endp
```

The following code will smoothly scroll a 50 line window onto the display:

```
maxsplit dw     50
temp    dw      0

        mov     temp,1
sp4:    mov     ax,temp
        inc     temp
        cmp     ax,maxsplit
        jge     sp5
        call    split
        jmp     sp4
sp5:                    ; continue
```

IBM Corporation
Editor, IBM Personal Computer Seminar Proceedings
4629
Post Office Box 1328
Boca Raton FL 33432