ปีการศึกษา 2537

โปรแกรมนำเสนอภาพบนไมโครคอมพิวเตอร์

IMAGES PRESENTATION PROGRAM ON MICROCOMPUTER

โดย

นายสุริศักดิ์	ช่องราไพ	35.102124
นายพรชัย	ศิษย์สิงห์ไพโรจน์	35.103280
นายสมศักดิ์	วีรประเสริฐสกุล	35.103296

อาจารย์ที่ปรึกษา

อ.ทรงชัย วีระทวีมาศ

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมนำเสนอภาพบนไมโครคอมพิวเตอร์

ผู้จัดทำ

1. นายสุริศักดิ์ ช่องราไพ
2. นายพรชัย ศิษย์สิงห์ไพโรจน์
3. นายสมศักดิ์ วีรประเสริฐสกุล

..... อาจารย์ที่ปรึกษา
(..... อ.ทรงชัย วีระทวีมาศ .....)

..... อาจารย์ที่ปรึกษา
(.....)

..... อาจารย์ที่ปรึกษา
(.....)

โปรแกรมนำเสนอภาพบนไมโครคอมพิวเตอร์

นายสุริศักดิ์ ช่องรำไพ
นายพรชัย ศิษย์สิงห์ไพโรจน์
นายสมศักดิ์ วีรประเสริฐสกุล
อ.ทรงชัย วีระทวีมาศ อาจารย์ที่ปรึกษา
ปีการศึกษา 2537

บทคัดย่อ

ในโครงงานนี้เป็นโครงงานที่เกี่ยวกับการเขียนโปรแกรมนำเสนอบนไมโครคอมพิวเตอร์ ซึ่งภาพที่ต้องการจะแสดงนั้นจะถูกสแกนแล้วเก็บข้อมูลภาพเหล่านั้นในฟอร์แมตต่างๆ ซึ่งฟอร์แมตที่จะสามารถนำมาใช้กับโปรแกรมนี้ได้คือ PCX FILES, BMP FILES, WPG FILES,LBM FILES, และ TGA FILES ความละเอียดของภาพจะต้องมีความละเอียด 320x200 จุด สีของภาพที่โปรแกรมนี้สามารถแสดงได้สูงสุดคือ 256 สี ส่วนรูปแบบของการแสดงบนหน้าจอ และรูปแบบการหายไปของภาพมีหลายรูปแบบขึ้นอยู่กับเทคนิคการเขียนโปรแกรม จุดสำคัญของการเขียนโปะแกรมคือต้องสามารถเขียนโปรแกรมถอดรหัสข้อมูลของภาพแล้วนำมาเสนอบนจอภาพ การเข้ารหัสของแต่ละฟอร์แมตไม่เหมือนกันดังนั้น การเขียนโปรแกรมถอดรหัสก็จะต่างกันโดยการป้อนชื่อภาพตามลำดับที่ต้องการแสดง ป้อนรูปแบบการแสดงกำหนดเวลาที่ต้องการหน่วงและป้อนรูปแบบการหายไปของภาพประโยชน์ของโปรแกรมนี้ คือ สามารถนำโปรแกรม มาช่วยเป็นสื่อการเรียนการสอนให้กับนักเรียน การอบรมเรื่องงานให้กับพนักงานในบริษัทและอื่นๆ อีกมากมาย โปรแกรมยังสามารถพัฒนาต่อไปได้อีก เช่น เสนอภาพพร้อมด้วยเสียงบรรยายในภาพนั้น

IMAGES PRESENTATION PROGRAM ON MICROCOMPUTER

Surisak Chongrampai
Pornchai Sitsingpairote
Somsak Veeraprasertsakul
Songchai Veerataveemas Avisor
1994

Abstract

This project is the project which deal about image presentation programing. The image which we would like to present was scaned and save as an image database in different format. The format that we use with this program consist of PCX files,BMP files,WPG files,LBM files and TGA files. The image must have resolution 320*200 pixels and this program can show 256 color maximum.

About the type of appearing and disappearing of the image. We have many type depend on programing techics. But the topic of this program is must can be decode data of the image and present on the monitor. Because of encoding of the different of other kinds of format,so the way to program to decode the other kinds of format must use different way to program too.

The ability of this program can present many images in the continuous mode by input the image name in order, type of appearing, delay time and type of disappearing of the images. The helpful of this program is can be use as the device to help in the classroom, the company's orientation and many kind of usage.

สารบัญ

บทที่ 1
บทนำ

เราคงเคยฟังการบรรยายสรุปเรื่องราวต่างๆมาบ้างแล้ว จะเห็นได้ว่าการบรรยายสรุปที่ดีสามารถดึงดูดความสนใจของผู้ฟังได้นั้น จะต้องมีการเตรียมการอย่างดี ให้ผู้ฟังใช้ประสาทสัมผัสให้มากที่สุดเพื่อซึมซับคำบรรยายได้แม่นยำเช่น นอกจากจะให้เห็นด้วยสายตาแล้วยังให้ได้ยินด้วยหู หรือนอกจากแสดงภาพนิ่งแล้วอาจจะให้เห็นภาพเคลื่อนไหวพร้อมทั้งคำบรรยาย โปรแกรมนำเสนอภาพมีความสามารถในการทำบรรยายสรุป ให้เป็นที่น่าสนใจแก่ผู้ชมและผู้ฟัง ได้โดยใช้คอมพิวเตอร์กับอุปกรณ์ประกอบในลักษณะเดียวกับ สไลด์มัลติวิชั่น โดยคำบรรยายที่ได้ เตรียมการไว้จะนำไปเสนอต่อผู้ชม โดยเรียงภาพต่าง ๆ ไว้ให้แสดงผลตามลำดับเป็นเวลาเท่า ๆ กันในแต่ละจอภาพ หรือจะให้ทันสมัยยิ่งขึ้น ด้วยการเสนอเป็นภาพเคลื่อนไหว พร้อมทั้งเอฟเฟกต์พิเศษต่างๆ ที่จะช่วยให้การเสนอคำบรรยายมีรสชาติชวนสนใจและไม่เบื่อหน่ายเป็นอย่างดี ปัจจุบันนี้ตามท้องตลาดก็มีโปรแกรมสำเร็จรูปในการนำเสนอภาพอยู่บ้างแต่ส่วนใหญ่ จะแสดงสีได้เพียง 16 ระดับสี ในการจะแสดงสีให้ได้ถึง 256 ระดับสีนั้น มีเทคนิคการเขียนโปรแกรมในการนำเสนอภาพแบบต่างๆ มากยิ่งขึ้น

ในปัจจุบันโปรแกรมคอมพิวเตอร์ส่วนใหญ่ มักจะอยู่ในรูปแบบของตัวโปรแกรมที่ค่อนข้างสวยงาม บางโปรแกรมอาจจะถึงกับวิจิตรพิศดารก็มี ในจำนวนโปรแกรมเหล่านั้นมีวิธีใช้ในการตกแต่งหน้าตาโปรแกรมมีมากมายหลายอย่าง เช่น ใช้เงา, ใช้สีฉูดฉาด, ภาพเคลื่อนไหวได้หรือมีภาพประกอบ เป็นต้น และในที่นี้เราจะทำการยกตัวอย่างของการตกแต่งจอด้วยการแสดงภาพในรูปแบบกราฟิก BITIMAGE ในปัจจุบันพบว่ามี (Format) มากมายที่ใช้เก็บภาพกราฟิกทั้ง PCX, BMP, WPG, TGA และLBM เป็นต้น

ภาพกราฟิกดูจะเป็นเรื่องที่สามารถบ่งบอกถึงความสามารถของโปรแกรมและประสิทธิภาพของเครื่องคอมพิวเตอร์ ไม่ว่าจะเป็นความละเอียดของภาพ จำนวนสี ความเร็วในการแสดงผลทั้ง โปรแกรมบนดอสและวินโดว์ แต่สิ่งหนึ่งที่ภาพกราฟิกจะต้องมีก็คือ ลักษณะภาพ

(Format) ซึ่งไฟล์ภาพ PCX หรือไฟล์ภาพทั่วไป โดยปกติแล้วจะประกอบไปด้วยเฮดเดอร์ (Header) ข้อมูลรูปภาพและ แพเลตต์ (Palette)

Header จะเป็นตัวชี้ลักษณะของรูปภาพนั้นว่ามีขนาดกว้างยาวเท่าใดเป็นฟอแมตร์ชนิดใด มีจำนวนสีเท่าไหร่เป็นต้น ส่วนข้อมูลรูปภาพจะเป็นข้อมูลแบบเข้ารหัสหรือบีบข้อมูล ในการบีบข้อมูลมีความแตกต่างกันในแต่ละฟอร์แมตมีทั้งการบีบข้อมูลแบบนับซ้ำ(Run length limit) แบบนับความถี่เชิงบิต (แบบ LZW = Lempel and Ziv (Welch 1984)) บางฟอร์แมตก็จะมีการจัดเก็บรูปภาพโดยตรง ไม่มีการบีบข้อมูลเลย

Palette จะเป็นการกำหนดค่าสีพอจะอธิบายสั้นๆ ได้คือในการเก็บภาพไว้แต่ละภาพไม่สามารถเก็บเป็นค่าสีมาตรฐานได้ เช่น ภาพนกอินทรีทอง ต้องการให้สีที่ปากเป็นสีเหลืองธรรมชาติจริงๆ แต่ค่าสีเหลืองที่มีอยู่ไม่ตรงตามความต้องการ เราจึงสามารถเซตแพเลตต์ได้ค่าของสีแต่ละแม่สีมีอยู่ 64(0-63) ระดับเท่านั้น จะเป็นได้ว่าสีเหลืองที่เราต้องการสามารถกำหนดได้จากสีแดงผสมกับสีเขียวถึง 63x63 ระดับทีเดียว (ค่า 0 ไม่ได้ทำให้เกิดสีผสมเราจึงไม่นำมาคิดด้วย)ก็เพียงพอกับสายตาและความต้องการของมนุษย์ขณะนี้แล้ว (ในอนาคตค่อยดูกันอีกที)

วัตถุประสงค์

1 สามารถนำโปรแกรมนี้ไปใช้เป็นสื่อการเรียนการสอนได้

2 สามารถนำโปรแกรมนี้ไปใช้ในการอบรมหรือบรรยายพิเศษในวาระต่างๆ

3 สามารถนำคอมพิวเตอร์มาประยุกต์ใช้งานให้เกิดประโยชน์มากขึ้นนอกเหนือจากประโยชน์ในด้านอื่นๆของคอมพิวเตอร์

บทที่ 2

ทฤษฎีและหลักการ

2.1 คอมพิวเตอร์กราฟิกกับการนำไปใช้งาน

คอมพิวเตอร์กราฟิกที่ใช้กันในคอมพิวเตอร์มีหลายหลากชนิด แต่ก็พอจะแบ่งประเภทออกได้เป็น 7 ประเภทใหญ่ ๆ คือ

1. โปรแกรมสำหรับวาดภาพ (Paint Programs)

โปรแกรมวาดภาพเป็นการสเก็ตภาพอย่างหยาบๆ โดยการใช้มือหรือการนำภาพที่ได้จากเครื่องรับภาพเข้าสู่คอมพิวเตอร์ (เช่น scanner) มาเปลี่ยนแปลงแก้ไขสำหรับงานพิมพ์ภาพที่ได้จากโปรแกรมนี้จะอยู่ในรูปของบิตแม็พ ซึ่งจะมีคุณภาพด้วยลงไปเมื่อถูกนำมาขยาย และคุณภาพของภาพนี้ขึ้นอยู่กับฮาร์ดแวร์ที่ใช้ โปรแกรมพวกนี้จะใช้รูปแบบไฟล์เก็บแฟ้มที่มีนามสกุล .PCX ตัวอย่าง เช่น PC Paintbrush,Publisher,Window paint หรือ Drhalo เป็นต้น

2. โปรแกรมสร้างภาพประกอบและการออกแบบ (Illustration/Design Software)

โปรแกรมประเภทนี้สร้างภาพในลักษณะเวกเตอร์ ใช้การลากเส้นตรงและเส้นโค้งเป็นหลัก มีรูปแบบการสร้างสีสันที่ค่อนข้างจะยุ่งยาก ความนิยมของโปรแกรมประเภทนี้พอ ๆ กับโปรแกรมวาดภาพ เนื่องจากคุณภาพที่ออกมาไม่ได้ผูกติดกับฮาร์ดแวร์ตัวที่สร้างภาพนั้นคืออุปกรณ์เขียนภาพที่มีความละเอียดสูงก็จะได้ภาพมีความละเอียดสูง ภาพที่สร้างโดยโปรแกรมประเภทนี้ส่วนมากอยู่ในรูปของ CGM (Computer Graphic Matafile) ซึ่งเป็นรูปแบบที่เตรียมตัวจะเป็นแบบมาตรฐานของแฟ้ม สำหรับภาพที่อยู่ในรูปเวกเตอร์ ตัวอย่างของโปรแกรมประเภทนี้ก็เช่น Adobe Illustrator,Corel Draw,GEM Artline, MASS-11 Dra และ Micrografix Designer

3. โปรแกรมกราฟิกเพื่อการนำเสนอ (Presentation Graphic Program)

โปรแกรมกราฟิกสำหรับการนำเสนอช่วยทำตัวเลข หรือคำบรรยายให้อยู่ในรูปของกราฟิกที่ดูแล้วเข้าใจได้ง่าย เช่นกราฟชนิดต่าง ๆ หรือการสร้างแผนผังการจัดองค์กร โปรแกรมประเภทนี้ส่วนมากใช้ในงานธุรกิจซึ่งบางทีก็เรียกว่า Business graphic อาจจะใช้อุปกรณ์สำหรับนำภาพออกได้หลายชนิด เช่น ลงแผ่นฟิล์ม, เครื่องลงจุด, เครื่องพิมพ์เลเซอร์ หรือเครื่องพิมพ์สี เป็นต้น ตัวอย่างของโปรแกรมสำเร็จรูป เช่น FreeLance Plus, Graph Plus, Harvard Graphics เป็นต้น

4. โปรแกรมสำหรับการทำภาพจำลองและเคลื่อนไหว (Animation Software)

โปรแกรมประเภทนี้จะเชื่อมโยงและเรียงลำดับภาพเพื่อการสร้างวิดิทัศน์ มีการเปลี่ยนแปลงภาพหนึ่งไปอีกภาพหนึ่งโดยวิธีการต่าง ๆ กันเช่น ค่อยๆ เลือนไปหรือลบหายไปทั้งภาพ ส่วนมากรวมเอาภาษาสำหรับการเขียนบท (script) ของภาพเอาไว้ด้วย ซึ่งคุณสามารถที่จะสร้างการเคลื่อนไหวต่าง ๆ และรวมถึงเครื่องมือสำหรับการสร้างภาพกราฟิกอย่างง่าย ๆ การดึงเอาภาพจากโปรแกรมอื่นมาใช้งานก็เป็นขีดความสามารถที่รวมในโปรแกรมประเภทนี้ เนื่องจากขีดความสามารถของซีพียูเอง การทำภาพ Animation โดย IBM PC จึงยังไม่อาจจะนับได้ว่าเป็น Animation อย่างแท้จริง ส่วนมากจะนิยมใช้สำหรับการสาธิตหรือใช้เป็นติวเตอร์การใช้งานของโปรแกรมต่าง ๆ ตัวอย่างของโปรแกรม เช่น PC Storyboard Plus,Show Partner F/X

5.โปรแกรมสำหรับช่วยงานออกแบบ (CADD Program)

โปรแกรมประเภทนี้เป็นลักษณะของโปรแกรมช่วยในการออกแบบต่าง ๆ (Computer Aided Design and Drafting) สำหรับงานทางด้านสถาปัตยกรรม วิศวกรรม อุตสาหกรรมและการจัดการทางด้านฟาซิลิตี้ต่าง ๆ ขีดความสามารถของโปรแกรมประเภทนี้จัดอยู่ใน

ชั้นหัวแถว เช่น การใส่มิติ การสร้างภาพ 3 มิติ การแรเงา การทำโมเดลของวัตถุ เป็นต้น ตัวอย่างของโปรแกรม เช่น Auto CAD

6. โปรแกรมสำหรับงานพิมพ์ (DeskTop Publishing)

โปรแกรมประเภทนี้นำเอาภาพจากโปรแกรมอื่นที่สร้างเอาไว้มาประกอบกับข้อความหรือบทความ เพื่อสร้างการพิมพ์โดยเฉพาะความสามารถอื่น ๆ ก็เช่นการเปลี่ยนแปลงรูปแบบของอักขระที่จะจัดพิมพ์ การจัดอาร์ตเวอร์กของหน้ากระดาษที่จะพิมพ์ เป็นต้น ตัวอย่างโปรแกรม เช่น IBM Interleaf Publisher

7. เครื่องมือสำหรับการทำกราฟิกอื่น ๆ

โปรแกรมช่วยจัดการทางด้านกราฟิกอื่น ๆ ช่วยทำให้การทำกราฟิกง่ายเข้า เช่น การดึงเอาภาพที่ปรากฏบนจอมาเก็บไว้ในแฟ้ม เพื่อนำไปแก้ไขหรือการถ่ายโอนแฟ้มของภาพระหว่างโปรแกรมต่าง ๆ ห้องสมุดสำหรับเก็บภาพต่าง ๆ เพื่อดึงมาใช้ได้ทันที เป็นต้น การใช้งานในส่วนการทำภาพเคลื่อนไหวที่เรียกว่า Animation นั้น IBM PC ยังไม่มีขีดความสามารถเพียงพอทั้งความสามารถทางด้านกราฟิกฮาร์ดแวร์และ Architecture ของตัวซีพียูเอง เราอาจจะเห็นการ์ตูนบนจอทีวีบ้านเราที่ใช้คอมพิวเตอร์มาช่วยในการเขียน แต่บน Amiga ของ Commadore ไม่ใช่ IBM XT หรือ AT คงทำได้ดีเฉพาะการนำมาทำภาพศิลปะและการทำภาพ slide สวย ๆ เพื่อการนำเสนอเท่านั้น สำหรับงานด้านการพิมพ์เข้ามามีบทบาทในการใช้ไมโครคอมพิวเตอร์เสมือนหนึ่งโรงพิมพ์บนโต๊ะทำงานของท่าน และกำลังได้รับความสนใจอย่างมาก ท่านอาจจะเคยได้ยินคำว่าเดสค์ท็อปพับลิชชิง เมื่อมีโปรแกรมอย่าง Ventura,Page Maker โปรแกรมพวกนี้จัดได้ว่าอยู่ในข่ายที่เรียกว่า WYSIWYG (What you see is what you get) หรือคุณเห็นอย่างไรคุณก็ได้อย่างนั้น นั่นคือเห็นบนจออย่างไรก็ได้อย่างนั้นพิมพ์ออกมา ไม่ว่าจะเป็นการนำเอาภาพมาประกอบในหน้าพิมพ์ การเปลี่ยนแปลงรูปแบบของตัวอักษรที่จะพิมพ์ โปรแกรมภาษาไทยหลายบริษัทก็พยายามพัฒนาให้ใกล้ WYSIWYG มากขึ้น นับว่าเป็นความพยายามที่ดี แม้ว่าจะยังห่างไกลมากนักเพราะภาษาไทย

ก็มีเรื่องยุ่ง ๆ ที่จะต้องนำมาคิดให้ปวดหัวพอควรอยู่แล้ว ต้องไม่ลืมว่าสิ่งที่ปรากฏบนจอจะปรากฏบนกระดาษหรือรูปแบบอื่นจะสมจริงสมจังได้จะต้องมีอุปกรณ์พิมพ์ที่มีความสามารถพอ

2.2 เทคนิคการแสดงกราฟิกบนจอแบบ Raster Scan

เพื่อให้ทำความเข้าใจเรื่องกราฟิกที่แสดงบนจอให้ดียิ่งขึ้น ผมขอกล่าวอย่างย่อ ๆ ถึงวิธีการนำแสดงผลกราฟิกบนหน้าจอพอเป็นสังเขป เทคนิคการแสดงกราฟิก อาจจะแบ่งได้เป็น 2 วิธีใหญ่ ๆ คือ

1. Bit Mapped
2. Vector

วิธีการของบิตแม็พก็คือทุกจุดบนหน้าจอ ซึ่งเรียกกันว่าพิกเซลนั้นมีความสัมพันธ์โดยตรงกับหน่วยความจำแบบหนึ่งต่อหนึ่ง แต่ก็มิได้หมายความว่า 1 พิกเซล จะเป็น 1 บิต ในหน่วยความจำเสมอไป อาจจะเป็น 4 บิตหรือ 8 บิต แล้วแต่ว่าจะให้ 1 พิกเซลสามารถแสดงสีหรือระดับความเข้มขาวดำมากน้อยแค่ไหน ข้อสำคัญก็คือจากตำแหน่งของพิกเซลบนหน้าจอเราสามารถมองหาค่าของพิกเซลในหน่วยความจำได้โดยตรง หรือการเปลี่ยนแปลงหน่วยความจำมีผลโดยตรงต่อการแสดงผลบนหน้าจอเป็นที่เรียกกันติดปากว่าหน่วยความจำ ที่ใช้ในการแสดงผลคือ วีดีโอแรม

ดังนั้นเทคนิคการแสดงกราฟิกโดยวิธีบิตแม็ต ก็คือการแก้ไขเปลี่ยนหน่วยความจำของการแสดงผลโดยตรง ข้อมูลภาพกราฟิกก็คือข้อมูลที่นำไปเขียนบนหน่วยความจำแสดงผล วิธีการแสดงกราฟิกแบบเวกเตอร์นั้นแตกต่างไปจากวิธีการของบิตแม็ต คือถ้าข้อมูลของภาพไม่ได้อยู่ในรูปที่จะนำไปแสดงบนจอโดยตรง แต่จะอยู่ในรูปของคำสั่งของการลากเส้น เขียนวงกลม เขียนรูปหลายเหลี่ยม และการระบายสีแทน ภาพที่ได้จากการสร้างโดยวิธีของเวกเตอร์จึงเรียกว่า Object-based Image คือบ่งบอกลักษณะของภาพมากกว่าที่จะเป็นตัวภาพเอง

จอภาพของการแสดงผลที่ใช้ในไมโครคอมพิวเตอร์ในปัจจุบันจะเป็น Raster Scan

คือแสดงในลักษณะกวาดแสดงไปบนจอติดต่อกันจากซ้ายไปขวา โดยใช้หลักการเดียวกับทีวี จำนวนเส้นที่กวาดแสดงผลนั้นเป็นตัวกำหนดขีดความสามารถของวิดีโอการ์ด ซึ่งจะมีตั้งแต่ 200,350,480 จนถึง 800 เส้น แต่เดิมนั้นจอภาพที่แสดงสี CGA สามารถรับสัญญานภาพให้แสดงผลได้ 200 เส้นใน 1 ภาพหรือที่เรียกว่า Frame Monochrome หรือ Hercules Graphic Adapter ก็ 350 เส้น จะเห็นว่าจำนวนเส้นของการแสดงจะใกล้เคียงกับระบบทีวีที่มีอยู่มาก คือระบบทีวีที่อยู่ในบ้านเราใช้ 625 เส้นใน 2 frame ทุกจุดที่ปรากฏบนจอจะต้องมีหน่วยความจำเก็บค่าของพิกเซลเอาไว้ นั่นก็คือ การมีหน่วยความจำสำหรับเก็บภาพแบบบิตแม็ตนั่นเอง กราฟิกโดยวิธีการบิตแม็ตจึงสามารถนำแสดงบนหน้าจอได้ทันที ส่วนกราฟิกแบบเวกเตอร์นั้น จะต้องให้ซีพียูแปลงออกมาเป็นบิตแม็ตในหน่วยความจำของการแสดงผลเสียก่อน จึงจะเห็นภาพปรากฏบนหน้าจอได้

ถ้าเปรียบเทียบให้เห็นชัดถึงความแตกต่างระหว่างบิตแม็ตและเวกเตอร์ ก็คงจะพอประมาณได้กับงานศิลปะ 2 วิธีคือ การวาดภาพ จิตรกรรม และงานปฏิมากรรม อย่างแรกเป็นการสร้างภาพขึ้นมาเปรียบได้กับกราฟิกชนิดบิตแม็ต อย่างที่สองเป็นสร้างงานศิลปะขึ้นมาสามารถที่จะมองในแง่มุนที่แตกต่างกันได้ ซึ่งเปรียบได้กับกราฟิกแบบเวกเตอร์ อย่างที่สามเป็นการสร้างงานศิลปะขึ้นมา สามารถที่จะมองในแง่มุมที่แตกต่างกันได้ ซึ่งเปรียบได้กับกราฟิกแบบเวกเตอร์

งานกราฟิกที่ใช้ทางด้านสถาปนิก และวิศวกรรมจะต้องเป็นชนิดเวกเตอร์คือจะต้องมีทั้ง 3 มิติ และสามารถใช้ขีดความสามารถของซีพียูแปลงเวกเตอร์ของกราฟิกให้ปรากฏออกมาบนจอได้ในแง่มุมที่แตกต่างกัน โปรแกรม AutoCAD เป็นตัวอย่างหนึ่งของการสร้างภาพแบบเวกเตอร์ ซึ่งสามารถกำหนดแง่มุมของการมองได้ไม่เพียงแต่เท่านั้นกราฟิกในรูปแบบเวกเตอร์สามารถที่จะนำมาแสดงในลักษณะย่อหรือขยายในอัตราส่วนเท่าไหร่ก็ได้ โดยที่ภาพที่แสดงออกยังคงรูปร่างที่เหมือนกับความเป็นจริง

เพื่อให้ผู้อ่านได้เห็นภาพได้ชัดเจน ระหว่างบิตแม็ตและเวกเตอร์ลองมาดูการแสดงตัวหนังสือบนจอกราฟิก โดยมากการแสดงอักขระบนหน้าจอจะเป็นแบบบิตแม็ต อักษรแต่ละตัว

ถูกกำหนดให้มีหน้าตาต่างกัน และเข้าไปครอบครองเนื้อที่ในหน่วยความนำเท่า ๆ กัน การนำอักขระเหล่านี้มาแสดงจึงไม่น่าจะผิดที่จะเรียกว่า การพิมพ์ตัวอักษรบนจอ ถ้าเราจะขยายการพิมพ์ให้ใหญ่ขึ้นก็จะต้องขยายให้เป็นอัตราส่วนที่เป็นเลขจำนวนเต็มเท่านั้น ภาพของอักษรที่ปรากฏบนหน้าจอไม่ได้มีความละเอียดเพิ่มขึ้น แต่กลับจะลดถอยลงไปมองเห็นรอยยัก ๆ บนตัวอักษรได้ชัดเจนยิ่งขึ้น

ในทางตรงข้ามการแสดงอักขระบนจอแบบเวกเตอร์นั้น เป็นการบอกวิธีการเขียนตัวอักษรบนจอ เช่น จากจุดเริ่มต้นลากเส้นขึ้นข้างบน 4 หน่วย ลากไปทางขวา 3 หน่วย ลากลง 4 หน่วย ดังกล่าวเป็นต้น ลักษณะเช่นนี้จึงไม่ผิดที่จะเรียกว่าเป็นการสอนคอมพิวเตอร์ให้รู้จักเขียนหนังสือ

ข้อดีของการเขียนแบบเวกเตอร์ก็คือ สามารถขยายโดยอัตราส่วนเท่าใดก็ได้ โดยไม่เสียความละเอียดของแง่มุม สามารถจะเขียนให้ทิศทางใด ๆ ก็ได้ไม่จำเป็นต้องเขียนจากซ้ายไปขวา อาจจะเขียนให้เอียงเป็นมุมเท่าใดก็ได้

การใช้งานของวิธีการสร้างภาพกราฟิกบนจออาจจะนำมาใช้ผสมกันทั้ง สองแบบ คือ บิตแม็พและเวกเตอร์แต่เวลาเก็บผลงานคือภาพที่ได้ออกมาเอาไปเก็บแบบบิตแม็พ วิธีการเช่นนี้ใช้ในโปรแกรมสำเร็จรูปในการสร้างกราฟิกเพื่อการนำเสนอ อย่างเช่น PC Paint

2.3 อุปกรณ์ที่เกี่ยวข้องกับคอมพิวเตอร์กราฟิก

ถ้าจะแบ่งอุปกรณ์ที่เกี่ยวข้องกับคอมพิวเตอร์กราฟิกก็เห็นจะแบ่งประเภท ดังนี้

1. ตัวไมโครคอมพิวเตอร์ ซึ่งบางทีก็ใช้คำว่า Graphic Engine
2. อุปกรณ์แสดงภาพ อันได้แก่ การ์ดแสดงผลและตัวจอด้วย
3. อุปกรณ์พิมพ์รูปกราฟิก
4. อุปกรณ์นำภาพจากภายนอกเข้าไปยังคอมพิวเตอร์
5. อุปกรณ์ช่วยในการเปลี่ยนแปลงแก้ไขภาพ

2.4 VGA ฮาร์ดแวร์

เทคโนโลยีการแสดงผลบนเครื่องคอมพิวเตอร์ส่วนบุคคลนับว่ามีการพัฒนาขึ้นมามากทีเดียว จากเดิมซึ่งเป็นการแสดงผลบนจอโมโนโครมที่ใช้การ์ดโมโนโครมอะแดปเตอร์ หรือการ์ดเฮอร์คิวลิสแล้วพัฒนาต่อมาเป็นการแสดงผลบนจอสีที่เรียกว่า Color Graphic Adapter (CGA) และ Enhanced Graphic Adapter (EGA) ซึ่งล้วนแล้วแต่เป็นระบบแสดงผลที่เป็นสัญญาณดิจิตอล

นั่นก็คือสัญญาณที่ออกจากการ์ดแสดงผลที่จะไปควบคุมจอภาพนั้นเป็นสัญญาณดิจิตอล ในระบบดิจิตอลนี้การ์ด EGA จะมีความละเอียดสูงสุดคือจำนวนจุด (Pixel) เท่ากับ 640x350 จุด และสามารถแสดงสีได้ 16 สีพร้อมกันจากจำนวนสีทั้งหมด 64 สี ต่อมาไอบีเอ็มได้พัฒนาระบบแสดงผลแบบใหม่ขึ้นมาอีกเรียกว่า Video Graphic Array (VGA) ซึ่งใช้สัญญาณอะนาลอกควบคุมจอแสดงผล ทำให้แสดงจำนวนสีได้มากขึ้นเพราะว่าคอมบิเนชั่นของสัญญาณอะนาลอกมีมากกว่าสัญญาณดิจิตอล ระบบ VGA จึงสามารถแสดงภาพสีได้ถึง 256 สีพร้อมกันจากจำนวนสีที่เป็นได้ทั้งหมด 256x1024 สี และยังมีความละเอียดสูงขึ้นถึง 640x480 จุด ทำให้สามารถแสดงภาพเสมือนจริงได้ ซึ่งมีประโยชน์อย่างมากในการใช้คอมพิวเตอร์กับงานด้านต่าง ๆ อาทิเช่น งานออกแบบทางด้านอุตสาหกรรม (CAD/CAM) งานทางการแพทย์ งานทางด้านการประมวลผลของรูปภาพ ซึ่งสามารถประยุกต์ใช้กับงานต่าง ๆ ได้อย่างมากมาย เนื่องจากข้อดีของระบบแสดงผลแบบ VGA นี้เอง ไอบีเอ็มจึงรวมการ์ดแสดงผลระบบ VGA ลงบนเมนบอร์ดของเครื่อง IBM PS/2 เช่น PS/2 model 50,60,70 จึงถือได้ว่าเป็นมาตรฐานการแสดงผลของเครื่อง IBM PS/2

ในปัจจุบันมีผู้ผลิตรายอื่นที่ทำการ์ด VGA ให้มีความละเอียดสูงกว่า VGA ของไอบีเอ็ม โดยเรียกว่า super VGA ที่มีความละเอียดถึง 800x600 จุด หรือ 1024x768 จุด ซึ่งต้องใช้กับจอแสดงผลแบบมัลติซิงค์หรือจอ VGA ความละเอียดสูง แนวโน้มของการ์ดแสดงผล VGA นับวันจะถูกลงเรื่อย ๆ ทำให้มีผู้ห้นมานิยมใช้กันมากขึ้น

2.5 โครงสร้างของการ์ด VGA

ระบบแสดงผลแบบ VGA หรือที่เรารู้จักกรดีว่าการ์ด VGA นั้นประกอบด้วยส่วนสำคัญอยู่ 3 ประการด้วยกันคือ

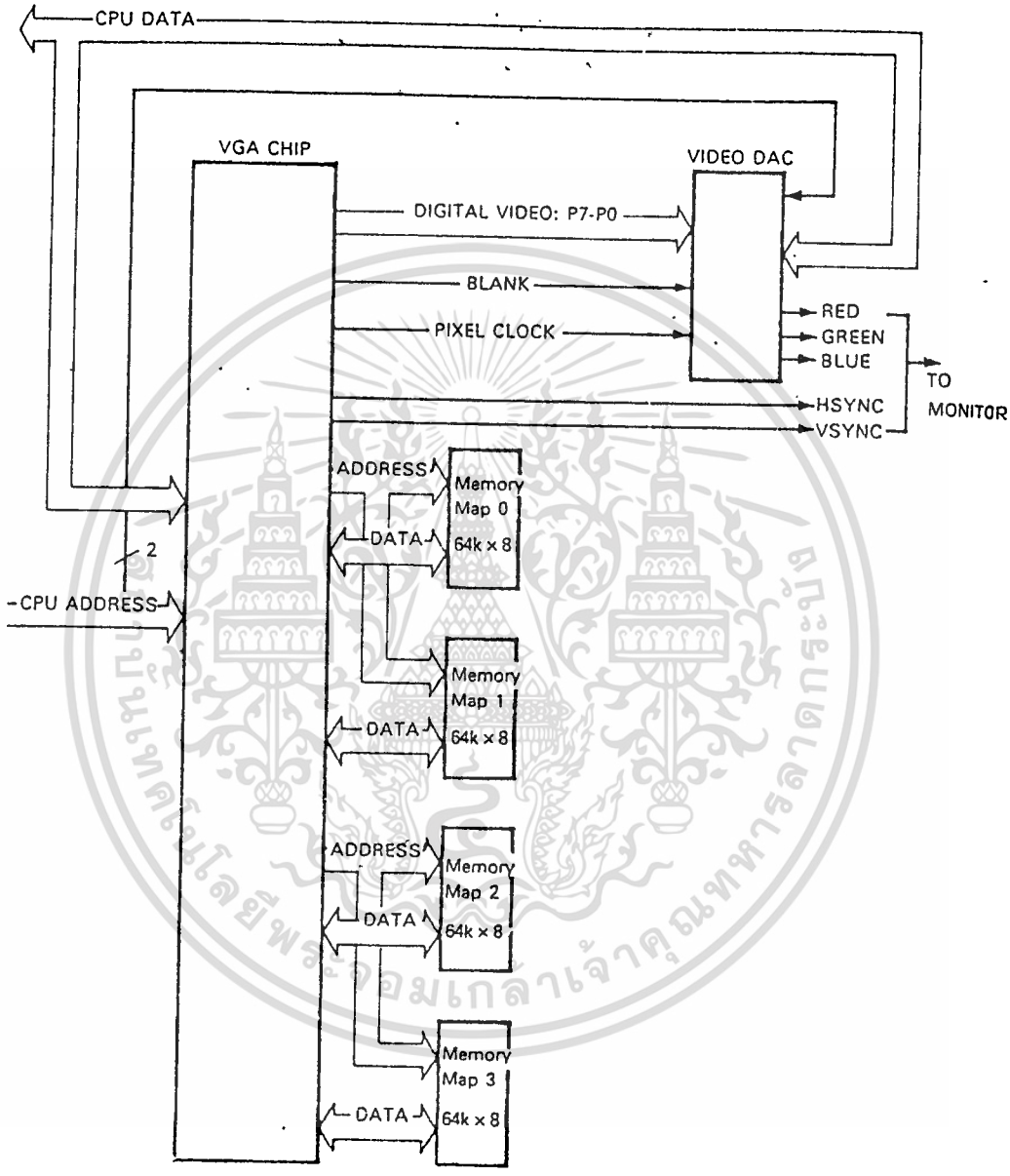
1. หน่วยความจำของส่วนแสดงผล (video memory)
2. ชิพ VGA
3. ตัวเปลี่ยนสัญญาณดิจิตอลให้เป็นสัญญาณอะนาลอก (Video DAC)

2.5.1 หน่วยความจำแสดงผล

หน่วยความจำของการ์ด VGA ตามมาตรฐานของไอบีเอ็มจะมีจำนวน 256 กิโลไบต์ เป็นไดนามิกแรมที่ถูกแบ่งออกเป็น 4 plane ข้อมูลที่ถูกใช้แสดงผลจะถูกเก็บอยู่ในหน่วยความจำนี้ ซึ่งรูปแบบของการเก็บจะแตกต่างกันไปขึ้นกับโหมดของการแสดงผล ตัวอย่างเช่น การแสดงผลในเท็กซ์โหมด (Text Mode) เพลน 0 ซึ่งเป็นหน่วยความจำแอดเดรสคู่จะเก็บข้อมูลที่เป็นรหัสแอสกีหน่วยความจำแอดเดรสคี่ ซึ่งอยู่ในเพลน 1 จะเก็บค่าแอตทริบิวต์ ส่วนหน่วนความจำเพลน 2 จะสำรองไว้ใช้สำหรับเก็บค่าคาแรกเตอร์เยนเนอเรเตอร์ หน่วนความจำเพลน 3 จะไม่ถูกใช้ ส่วนในโหมดกราฟิกก็จะแตกต่างกันไป ในปัจจุบันการ์ด VGA ที่ถูกผลิตขึ้น มักจะมีความละเอียดในการแสดงผลสูงกว่ามาตรฐานของไอบีเอ็ม ซึ่งจำเป็นจะต้องใช้หน่วยความจำมากขึ้นด้วย เพื่อเก็บข้อมูลภาพที่มากขึ้นนั่นเอง

หน่วยความจำในเท็กซ์โหมด

การแสดงผลในเท็กซ์โหมด (Text Mode) มีความซับซ้อนน้อยกว่าการแสดงผลในโหมดกราฟิกมาก เพราะว่าเป็นการจัดการกับรหัสแอสกี มิใช่จัดการกับจุดใดจุดหนึ่งบนจอภาพ มาตรฐานของเท็กซ์โหมดแบ่งเป็น 25 บรรทัด 40 คอลัมน์ หรือ 80 คอลัมน์ต่อบรรทัด ในกรณีที่เป็น 80 คอลัมน์ต่อบรรทัด ใน 1 จอภาพสามารถแสดงตัวอักขระได้ทั้งสิ้น 2,000 ตัว แต่การแสดงของตัวอักขระ 1 ตัว ต้องใช้หน่วยความจำ 2 ไบต์ ดังนั้นใน 1 จอภาพจะ



รูปที่ 2.1 ส่วนประกอบหลักของการ์ด VGA

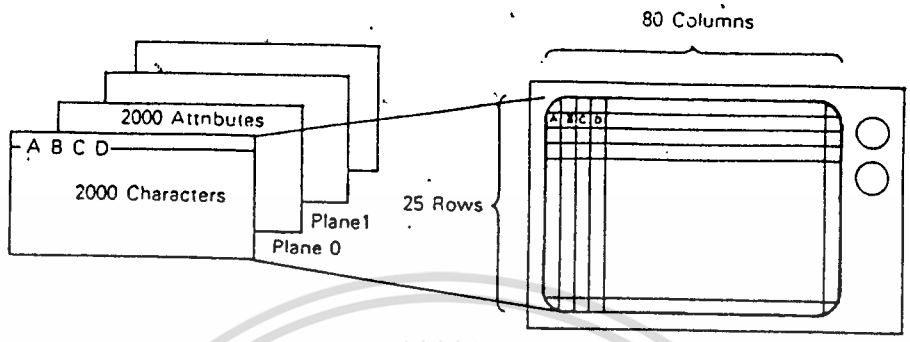
ต้องใช้หน่วยความจำทั้งสิ้น 4,000 ไบต์ แต่หน่วยความจำแสดงผลแบ่งออกเป็นเพจ ๆ ละ 4096 ไบต์ ซึ่งจะเหลือที่ว่าง 96 ไบต์ที่ไม่ถูกใช้

ในการเปลี่ยนรหัสแอสกีไปเป็นจุดที่เรียงกันเป็นตัวอักขระบนจอภาพนั้น จะต้องใช้ตารางการแปลงที่เรียกว่าคาแรกเตอร์เยนเนอเรเตอร์ในการแสดงผลระบบเดิม เช่น การ์ดโมโนโครมตารางการแปลงนี้จะเก็บอยู่ในหน่วยความจำที่เป็นชนิด ROM (อ่านได้อย่างเดียว) จึงไม่สามารถแก้ไขรูปแบบของอักขระได้โดยง่าย แต่ใน VGA หรือ EGA ตารางการแปลงนี้จะถูกโหลดลงในเพลนที่ 2 ของหน่วยความจำ (ซึ่งเป็นไดนามิคแรม) ทำให้สามารถแก้ไขรูปแบบตัว อักขระได้ง่าย ใน EGA จะมีตารางแปลงนี้ได้ถึง 4 ชุด ส่วนใน VGA มีได้ถึง 8 ชุด แต่ละชุดเก็บได้ถึง 256 ตัว

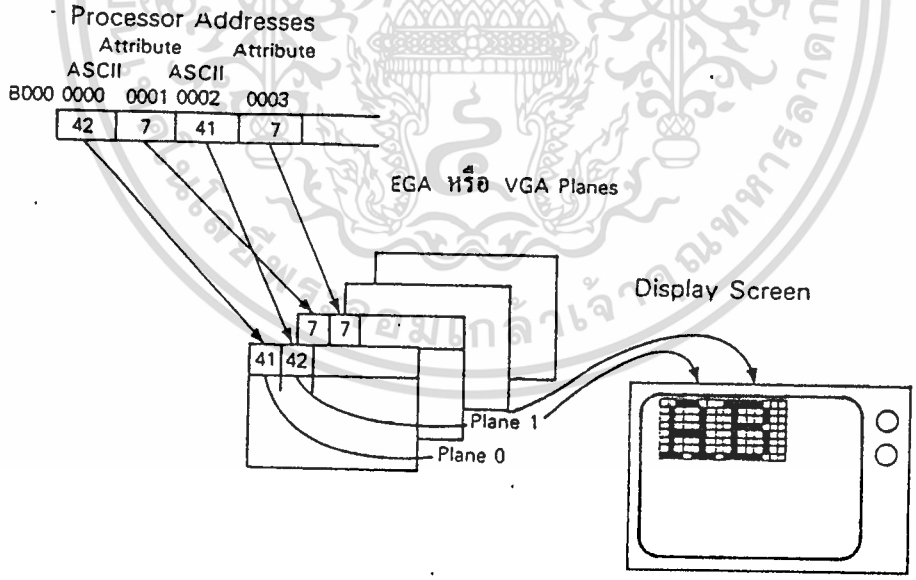
มาตรฐานของตัวอักขระในตารางแปลงนี้ จะมีจำนวนจุด (pixel) ต่างกันไปขึ้นอยู่กับโหมดของการแสดงผล เช่นในโหมด CGA (การ์ด VGA สามารถทำงานในโหมดที่มีความละเอียดต่ำกว่าได้) ตัวอักขระจะมีขนาด 8x8 (กว้างxยาว) จุดใน EGA จะเป็น 8x14 จุด และ 8x16 จุด สำหรับ VGA ค่าในตารางการแปลงนี้จะถูกโหลดลงใหม่ทุกครั้งโดยไบออสบนการ์ด เมื่อมีการเปลี่ยนโหมดของการทำงาน

หน่วยความจำในกราฟิกโหมด

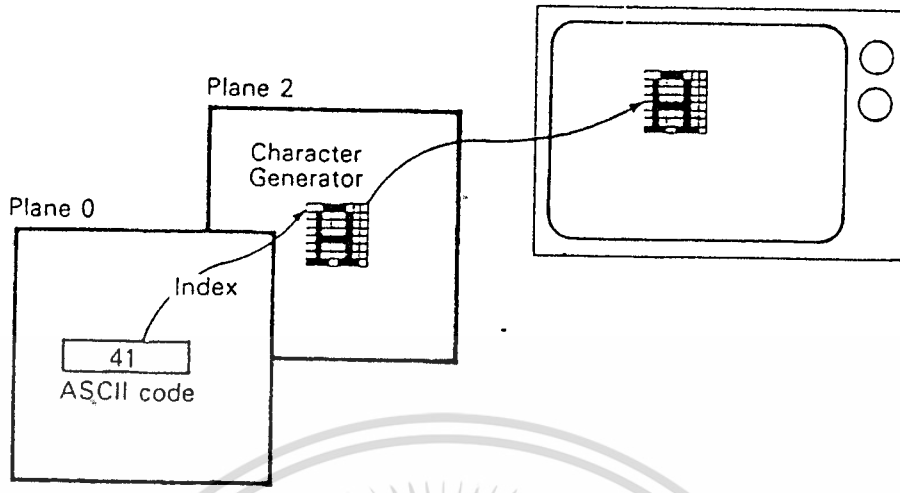
ในโหมดกราฟิกจุดๆ หนึ่งบนจอจะแทนด้วยข้อมูลที่มีจำนวนบิตต่างๆ กัน เช่นในโหมด CGA 2 สี (mode 6) จะใช้ข้อมูล 1 บิต ในการแสดงจุดหนึ่งจุด (8 จุดต่อไบต์) ในที่นี้จะขอกล่าวถึงโหมด Enhanced Color Graphics (mode 10H) เท่านั้น ในโหมด 10H นี้เป็นที่นิยมสำหรับการใช้กับงานต่าง ๆ เพราะว่ามีความละเอียดสูงถึง 640x350 จุด และแสดงสีได้ 16 สีพร้อมกันในการแทนจุด ๆ หนึ่งบนจอภาพ จะใช้ข้อมูล 4 บิต โดยที่แต่ละบิตมาจากแต่ละเพลนของหน่วยความจำ



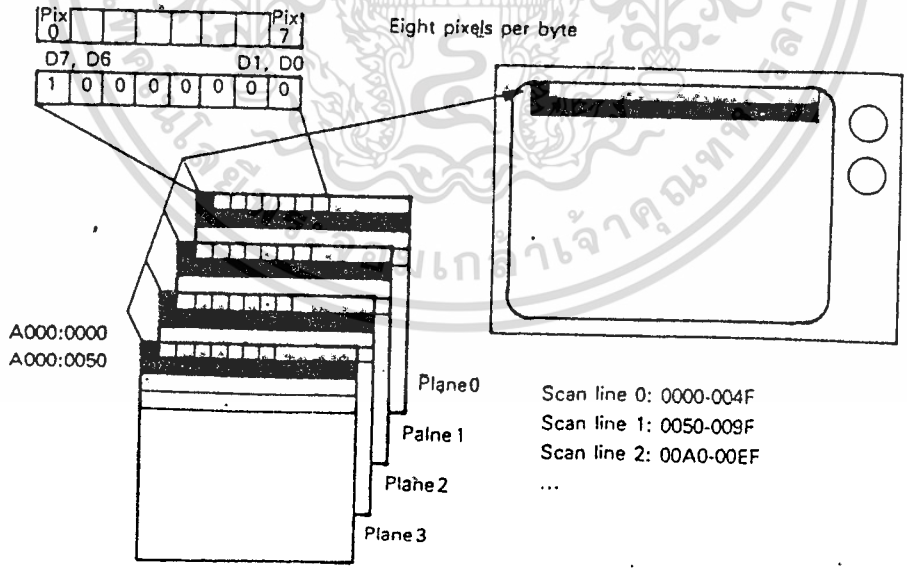
รูปที่ 2.2 (ก) รูปแบบการจัดหน่วยความจำในเท็กซ์โหมด



รูปที่ 2.2 (ข) ความสัมพันธ์ระหว่างแอดเดรสกับเพลนของหน่วยความจำ



รูปที่ 2.3 ขั้นตอนการทำให้เกิดตัวอักขระบนจอภาพ



รูปที่ 2.4 รูปแบบการเก็บข้อมูลในหน่วยความจำสำหรับโหมดกราฟิก

2.5.2 ชิพ VGA

ระบบแสดงผลแบบ VGA มาตรฐานของไอบีเอ็มนั้นใช้ชิพ 82706 Video Graphic Array ซึ่งคอมแพตติเบิ้ลระดับไบออสกับระบบเดิมคือ EGA CGA และ MDA

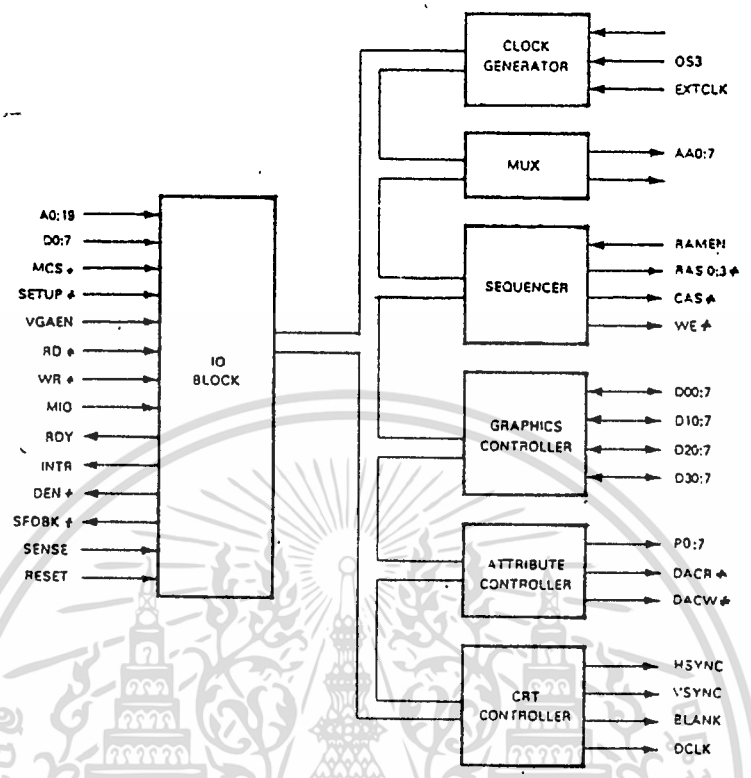
82706 VGA controller เป็นตัวอินเตอร์เฟสระหว่างซีพียู ซึ่งในที่นี้อาจจะเป็น 8088,80286 หรือ 80386 กับหน่วยความจำของการแสดงผล (video memory)และเป็นตัวส่งข้อมูลของภาพไปยัง videoDAC (Digital to Analog Converter) ทำการแปลงสัญญาณดิจิตอลไปเป็นสัญญาณอะนาลอก เพื่อส่งให้จอแสดงผลต่อไป

การแอ็กเซสกับหน่วยความจำของการแสดงผลจะต้องผ่านตัว 82706 เสมอ ซึ่งมีข้อดีก็คือ ซีพียูสามารถแอ็กเซสกับหน่วยความจำขณะที่ทำการรีเฟรชหน่วยความจำ ซึ่งทำให้ซีพียูเขียน หรืออ่านจากหน่วยความจำได้โดยไม่จำเป็นต้องรอให้ถึงช่วงเวลาของการรีเทรช (retrace) ของจอภาพ

82706 ใช้กับหน่วยความจำได้ 256 กิโลไบต์ โดยสามารถโปรแกรมแอดเดรสเริ่มต้นได้ 3 ตำแหน่ง ซึ่งโดยทั่วไปมักจะขึ้นกับโหมดการแสดงผลจาก รูปที่ 2.5 จะเห็นว่าภายในชิพ82706 จะยังมีตัวควบคุมการทำงานหลักอยู่ 4 ตัว คือ

1. CRT controller

ทำหน้าที่กำเนิดสัญญาณที่ใช้ควบคุมการทำงานของจอภาพ เช่น สัญญาณซิงก์ตามแนวนอน สัญญาณซิงก์ตามแนวตั้ง สัญญาณแบล็งกิ้ง และแอดเดรส สำหรับการรีเฟรชหน่วยความจำควบ CRT controller มีรีจิสเตอร์ 25 ตัว ซึ่งมีบางตัวที่คอมแพตติเบิ้ลกับ 6845 CRT controller ที่มีอยู่บนการ์ดแสดงผลแบบโมโนโครม



รูปที่ 2.5 บล็อกไดอะแกรมของชิพ VGA

2. Graphic Controller

เป็นตัวกลางเชื่อมทางเดินข้อมูลระหว่างหน่วยความจำของการแสดงผล กับตัวโปรเซสเซอร์หลักกับแอดทริบิวต์คอนโทรลเลอร์ในสภาวะปกติข้อมูลจากโปรเซสเซอร์จะสามารถถูกส่งไปยังหน่วยความจำโดยทะลุกราฟิกคอนโทรลเลอร์ได้โดยตรง แต่ในกรณีอื่นกราฟิกคอนโทรลเลอร์มีฟังก์ชั่นช่วยในการวาดรูปกราฟิก โดยนำข้อมูลที่ผ่านตัวมันมากระทำฟังก์ชันทางลอจิก ก่อนที่จะเขียนลงไปยังหน่วยความจำ ภายในกราฟิกคอนโทรลเลอร์มีรีจิสเตอร์อยู่ 9 รีจิสเตอร์ ดังแสดงในตารางรูปที่ 2.7 ซึ่งสามารถสรุปการทำงานได้ดังรูปที่ 2.8 จากรูปจะเห็นว่ามีข้อมูลที่เรียกว่า Latched Data อยู่ 4 ไบต์ (แต่ละไบต์มาจากแต่ละเพลน) ข้อมูลเหล่านี้จะถูกเก็บอยู่ในที่ที่หนึ่ง ซึ่งจะเกิดขึ้นทุกครั้งที่มีการอ่านข้อมูลจากโปรเซสเซอร์

(Processcer Data) จะผ่านการ ROTATE ซึ่งอาจจะเป็นการ ROTATE ตั้งแต่ 0 บิต (ไม่มีการ ROTATE) ถึง 7 บิต ทั้งนี้ขึ้นกับค่าในรีจิสเตอร์ Data Rotate (บิตที่ 0-2) จากนั้นจะถูกนำมากระทำฟังก์ชันใดนั้นขึ้นกับค่าในรีจิสเตอร์ Data Rotate (บิตที่ 3-4) ฟังก์ชันทางลอจิกแสดงดังตารางในรูปที่ 2.9

Register Name	R/W	Index	Read Port	Write Port
CRT Controller Address	R/W		03?4	03?4
Horizontal Total	R/W	00	03?5	03?5
Horizontal Display Enable	R/W	01	03?5	03?5
Start Horizontal Blanking	R/W	02	03?5	03?5
End Horizontal Blanking	R/W	03	03?5	03?5
Start Horizontal Retrace Pulse	R/W	04	03?5	03?5
End Horizontal Retrace	R/W	05	03?5	03?5
Vertical Total	R/W	06	03?5	03?5
Overflow	R/W	07	03?5	03?5
Preset Row Scan	R/W	08	03?5	03?5
Maximum Scan Line	R/W	09	03?5	03?5
Cursor Start	R/W	0A	03?5	03?5
Cursor End	R/W	0B	03?5	03?5
Start Address High	R/W	0C	03?5	03?5
Start Address Low	R/W	0D	03?5	03?5
Cursor Location High	R/W	0E	03?5	03?5
Cursor Location Low	R/W	0F	03?5	03?5
Vertical Retrace Start	R/W	10	03?5	03?5
Vertical Retrace End	R/W	11	03?5	03?5
Vertical Display Enable End	R/W	12	03?5	03?5
Offset	R/W	13	03?5	03?5
Underline Location	R/W	14	03?5	03?5
Start Vertical Blank	R/W	15	03?5	03?5
End Vertical Blank	R/W	16	03?5	03?5
CRTC Mode Control	R/W	17	03?5	03?5
Line Compare	R/W	18	03?5	03?5

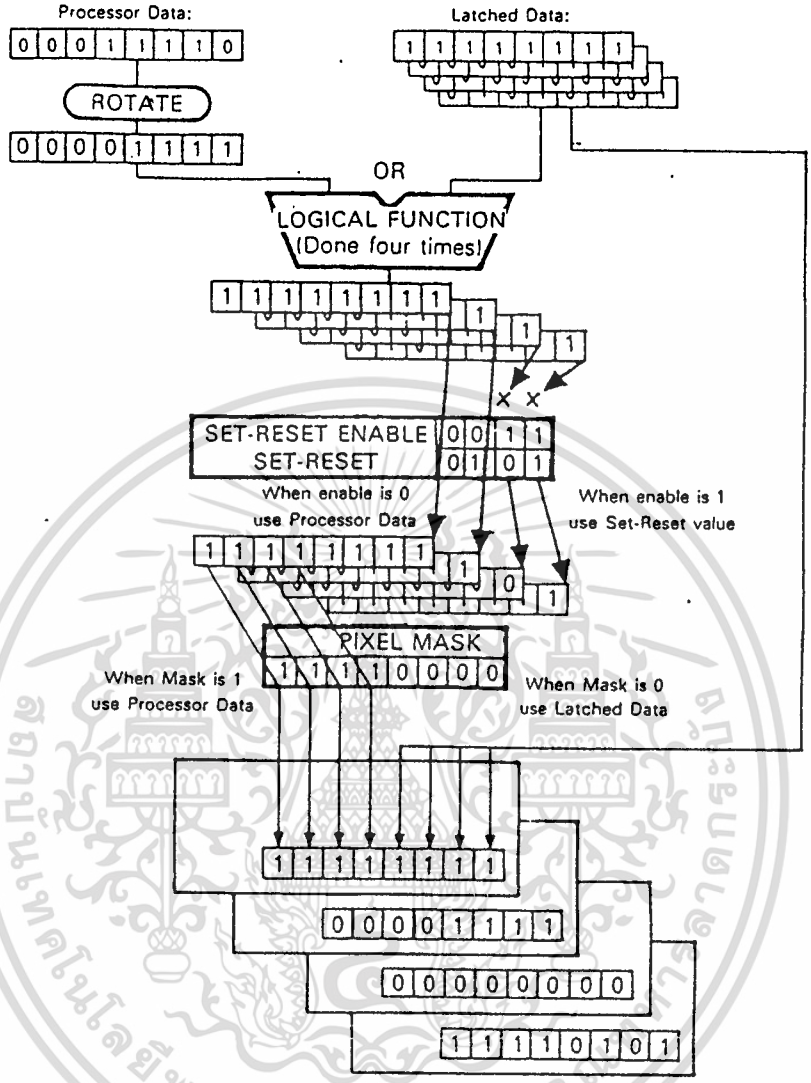
NOTES:
? = B in Monochrome Emulation Modes
? = D in Color Emulation Modes
All addresses are given in Hex

รูปที่ 2.6 ตารางแสดงรีจิสเตอร์ต่าง ๆ ใน CRT Controller

Register Name	R/W	Index	Read Port	Write Port
Graphics Address	R/W		03CE	03CE
Set/Reset	R/W	00	03CE	03CE
Enable Set/Reset	R/W	01	03CE	03CE
Color Compare	R/W	02	03CE	03CE
Data Rotate	R/W	03	03CE	03CE
Read Map Select	R/W	04	03CE	03CE
Graphics Mode	R/W	05	03CE	03CE
Miscellaneous	R/W	06	03CE	03CE
Color Don't Care	R/W	07	03CE	03CE
Bit Mask	R/W	08	03CE	03CE

รูปที่ 2.7 รีจิสเตอร์ในกราฟิกคอนโทรลเลอร์

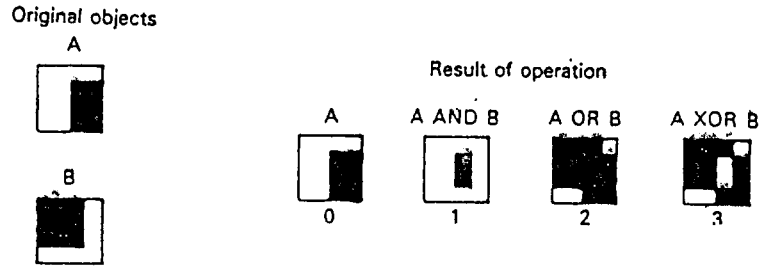
จากนั้นข้อมูลจะผ่านรีจิสเตอร์ 2 ตัวคือ SET/RESET และ SET/RESET ENABLE แต่ละรีจิสเตอร์จะใช้เพียง 4 บิต แต่ละบิตแทนแต่ละเพลน จากรูปบิตที่ 2 และที่ 3 ของรีจิสเตอร์ SET/RESET ENABLE เป็น 0 ดังนั้น ข้อมูลเพลนที่ 2 และ 3 จะผ่านไปได้โดยตรงส่วนบิตที่ 0 และ 1 มีค่าเป็น 1 ต้องพิจารณาในรีจิสเตอร์ SET/RESET เช่นบิตที่ 0 มีค่าเป็น 1 ดังนั้นเพลนที่ 0 จึงมีค่าเป็น 1 ทั้ง 8 บิต จากนั้นข้อมูลจะผ่านรีจิสเตอร์ Bit Mask ถ้าค่าในรีจิสเตอร์เป็น 1 ก็จะไม่มีการเปลี่ยนแปลงค่าของข้อมูล แต่ถ้าค่าในรีจิสเตอร์เป็น 0 ข้อมูลในบิตนั้น ๆ จะถูกนำมาจาก Latched Data แทนที่จะเป็นข้อมูลเดิม รีจิสเตอร์ Bit Mask เป็นรีจิสเตอร์สุดท้ายของกราฟิกคอนโทรลเลอร์ที่ทำให้ข้อมูลมีการเปลี่ยนแปลง หลังจากนี้แล้วข้อมูลนี้ยังมิได้ถูกเขียนลงไปยังหน่วยความจำโดยตรง จะต้องผ่านตัวควบคุมอีกตัวหนึ่งเรียกว่า Sequencer ซึ่งจะได้กล่าวต่อไปตัวอย่างของการกระทำฟังก์ชันทางลอจิกของกราฟิกคอนโทรลเลอร์แสดงดังรูปที่ 2.10 จากรูปจะเห็นว่าเดิมเรามีรูปร่างของ A และ B อยู่ เราสามารถทำให้เกิดรูปร่างอื่น ๆ ขึ้นได้ ซึ่งมีประโยชน์ในการทำให้เกิดภาพต่าง ๆ ในโหมดกราฟิก



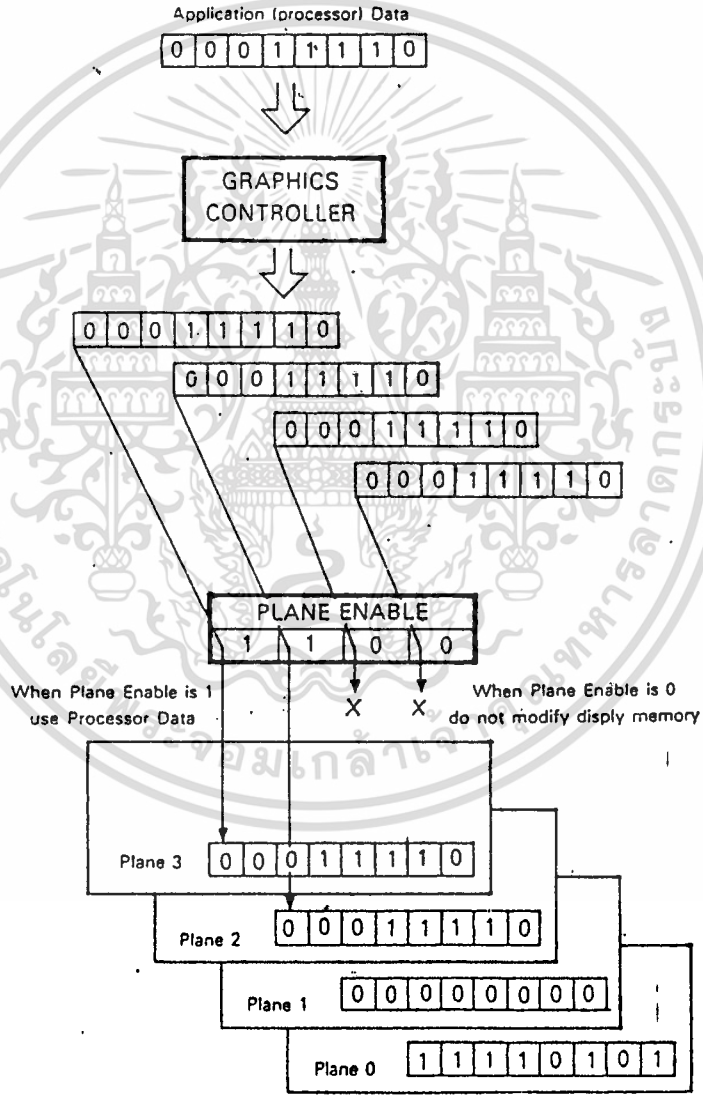
รูปที่ 2.8 บล็อกไดอะแกรมของกราฟิกคอนโทรลเล่อร์

D_4 D_3	ฟังก์ชัน
0 0	ข้อมูลจากโปรเซสเซอร์ไม่เปลี่ยนแปลง
0 1	ข้อมูลจากโปรเซสเซอร์ AND กับ Latched Data
1 0	ข้อมูลจากโปรเซสเซอร์ OR กับ Latched Data
1 1	ข้อมูลจากโปรเซสเซอร์ XOR กับ Latched Data

รูปที่ 2.9 ตารางแสดงฟังก์ชั่นลอจิก



รูปที่ 2.10 การกระทำฟังก์ชั่นทางลอจิกระหว่างข้อมูล 2 ข้อมูล



รูปที่ 2.11 การทำงานของรีจิสเตอร์ Map mask (Plane enable)

Register Name	R/W	Index	Read Port	Write Port
Sequencer Address	R/W		03C4	03C4
Reset	R/W	00	03C5	03C5
Clocking Mode	R/W	01	03C5	03C5
Map Mask	R/W	02	03C5	03C5
Character Map Select	R/W	03	03C5	03C5
Memory Mode	R/W	04	03C5	03C5

รูปที่ 2.12 ตารางแสดงรีจิสเตอร์ใน Sequencer

3. Sequencer

ทำหน้าที่กำเนิดสัญญาณนาฬิกาที่ควบคุมการรีเฟรชหน่วยความจำควบคุมช่วงเวลา ในการเขียนและอ่านกับหน่วยความจำ และยังมีวงจรลอจิกควบคุมการยอม หรือไม่ยอมให้โปรเซสเซอร์กระทำกับหน่วยความจำเพลนใดเพลนหนึ่ง ซึ่งแสดงดังรูปที่ 2.11 และตารางในรูปที่ 2.12 แสดงรีจิสเตอร์ต่าง ๆ ของ Sequencer จากรูปจะเห็นว่า ถ้าค่าในรีจิสเตอร์ Map mask (plane enable) เป็น 0 (D0 แทนเพลน 0)เพลนนั้นจะไม่สามารถถูกเปลี่ยนข้อมูลได้

4. Attribute Controller

ทำหน้าที่ควบคุมแอตทริบิวต์ของการแสดงผลเช่นแอตทริบิวต์ที่เป็นสีต่างๆ แอตทริบิวต์ที่แสดงการกระพริบ (blinking) หรือการขีดเส้นใต้ (underline) แอตทริบิวต์คอนโทรลเลอร์ประกอบด้วยรีจิสเตอร์ 20 รีจิสเตอร์ดังแสดงในตารางในรูปที่ 2.13 ในการกระทำ (เขียนหรืออ่าน) กับรีจิสเตอร์ของ VGA นั้นจะใช้คำสั่งที่กระทำกับพอร์ตคือ คำสั่ง In และ Out แต่ละกลุ่มของรีจิสเตอร์จะมีแอดเดรสพอร์ตประจำแต่ละกลุ่มเช่น กราฟิกคอนโทรลเลอร์จะมีแอดเดรสพอร์ตที่ 3CE เป็นอินเด็กซ์รีจิสเตอร์และ 3CF เป็นดาต้ารีจิสเตอร์ ตัวอย่างเช่น ต้องการ Out ค่า 3EH (ฐานสิบหก) ไปที่รีจิสเตอร์ Data Rotate ก็ทำได้โดย Out

ค่า 03 ไปที่พอร์ต 3CE (03 เป็นอินเด็กซ์ของ Data Rotate) จากนั้นจึง Out ค่า 3CH ไปที่พอร์ต 3CF สำหรับรีจิสเตอร์อื่นก็ทำนองเดียวกัน แต่สำหรับแอตทริบิวต์คอนโทรลเลอร์ ซึ่งมีพอร์ตสำหรับการเขียนอยู่แอดเดรสเดียวกัน ไซเกิลของการเขียนจะทำให้มีการเปลี่ยนไปมาระหว่างอินเด็กซ์รีจิสเตอร์กับดาต้ารีจิสเตอร์ ซึ่งจะสามารถเริ่มต้นได้ด้วยคำสั่ง In ที่พอร์ต 3DA หลังจากนั้นคำสั่งแรกที่ Out ไปที่พอร์ต 3C0 จะถูกส่งไปที่อินเด็กซ์รีจิสเตอร์ และคำสั่ง Out คำสั่งต่อไปจะถูกส่งไปที่ดาต้ารีจิสเตอร์

ส่วนสำคัญของแอตทริบิวต์คอนโทรลเลอร์อยู่ที่ตารางค้นหาสี (Color look-up table) ซึ่งจะทำการแปลงข้อมูลขนาด 4 บิตที่เก็บในหน่วยความจำของการแสดงผลไปเป็น

Register Name	R/W	Index	Read Port	Write Port
Address	R/W		03C0	03C0
Palette Registers	R/W	00-0F	03C1	03C0
Attribute Mode Control	R/W	10	03C1	03C0
Overscan Color	R/W	11	03C1	03C0
Color Plane Enble	R/W	12	03C1	03C0
Horizontal PEL Panning	R/W	13	03C1	03C0
Color Select	R/W	14	03C1	03C0

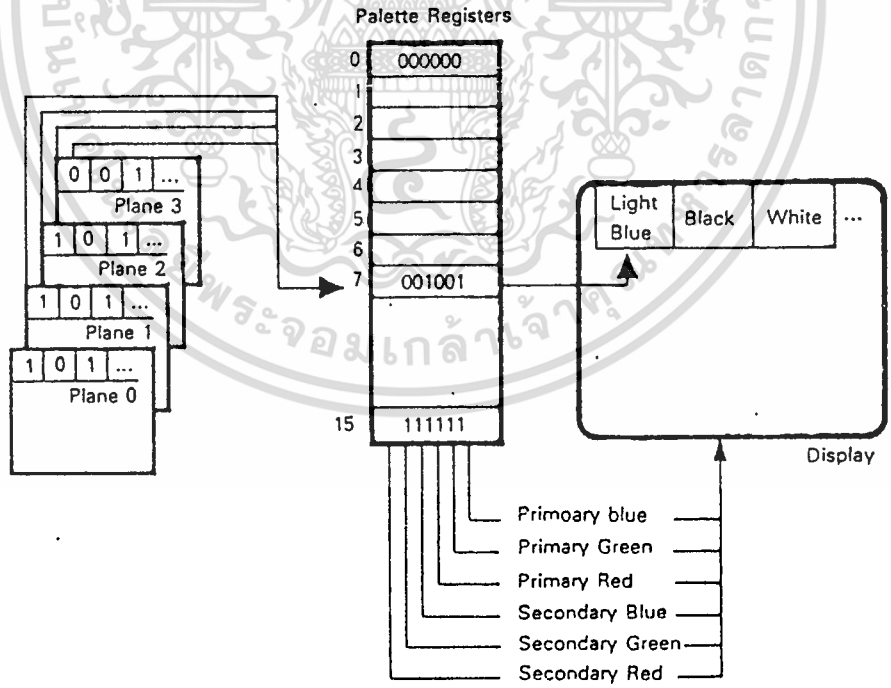
รูปที่ 2.13 ตารางแสดงรีจิสเตอร์ในแอตทริบิวต์คอลโทรลเลอร์

ข้อมูลของสีขนาด 6 บิต ซึ่งจะรวมกับข้อมูลจากรีจิสเตอร์ Color Select อีก 2 บิต รวมเป็น 8 บิต ที่จะส่งไปให้กับ Video DAC ต่อไป แต่สำหรับ EGA ข้อมูลสี 6 บิต จะถูกส่งไปยังจอแสดงผลโดยตรง ซึ่งมีข้อแตกต่างระหว่างโหมดตัวอักษร (TEXT MODE) กับโหมดกราฟิก รูปที่ 2.14 แสดงตารางค้นหาสีของแอตทริบิวต์คอนโทรลเลอร์ในรูป (ก) นั้นเป็นการแสดงในโหมดกราฟิก ค่าของสีจุด ๆ หนึ่ง (pixel) มีค่าเป็น 0111 (เท่ากับ 7) ค่าของสีนี้จะถูกใช้เป็นแอดเดรสชี้ไปรีจิสเตอร์ที่ 7 ของตารางค้นหาสี ซึ่งอยู่ภายในมีค่าเป็น 001001 บิตที่ 0 และบิตที่ 3 แทนสีฟ้าที่มีความเข้มแตกต่างกัน ดังนั้นจุดของภาพ (pixel) จุดแรกจึงมีสีฟ้า ส่วนจุดต่อไปมีค่าเป็น 0000 ซึ่งชี้ไปที่รีจิสเตอร์ 0 และค่าภายในเป็น 0 ดังนั้นจุด (pixel) นี้จึงเป็นสีดำในรูปที่ 14 (ข)เป็นการแสดงในโหมดตัวอักขระจะเห็นว่า เพลน 0 เก็บค่า 41 ซึ่งเป็นรหัสแอสกีของตัวอักษร A เพลน 1 เก็บค่า 07 ซึ่งเป็น

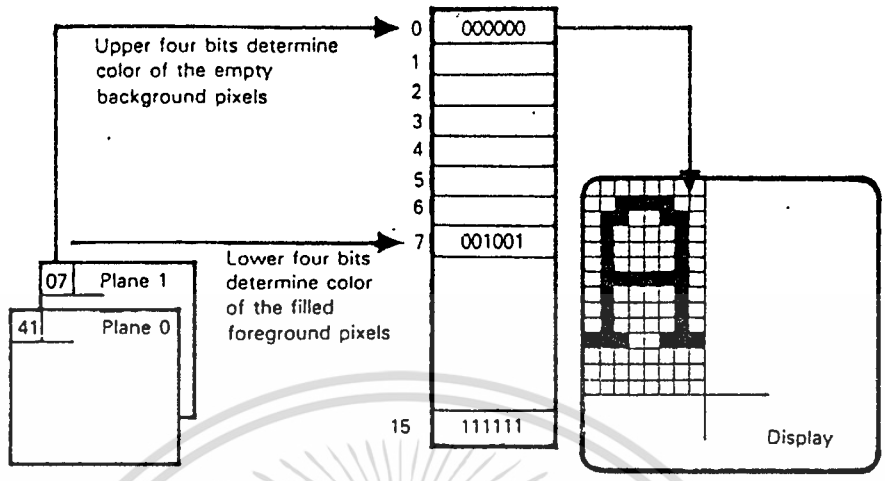
แอตทริบิวต์ของตัวอักษร A 4 บิต บนของแอตทริบิวต์เป็นตัวกำหนดสีของแบคกราวนด์และ 4 บิตล่างเป็นตัวกำหนดสีของโฟร์กราวนด์

2.5.3 ตัวเปลี่ยนสัญญาณดิจิตอลให้เป็นสัญญาณอะนาลอก (VIDEO DAC)

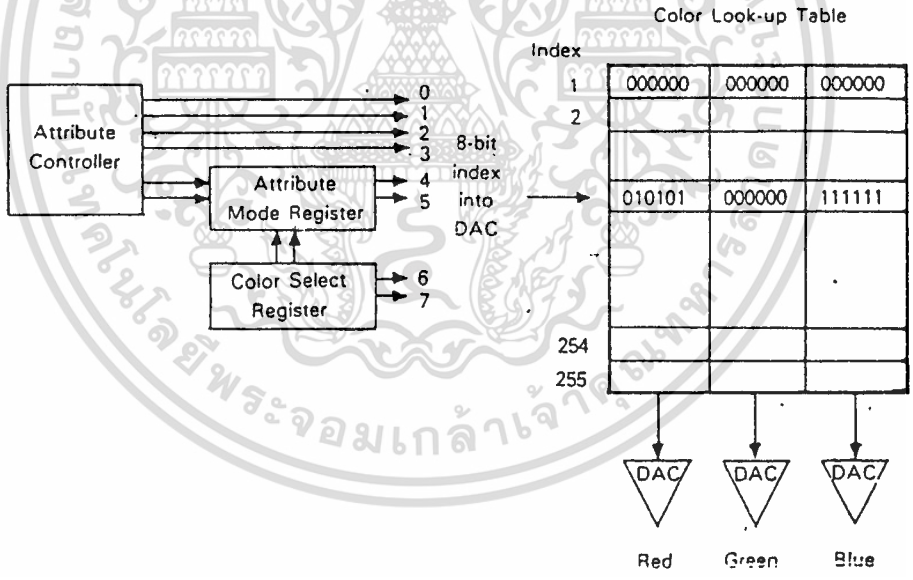
ส่วนประกอบหลักตัวสุดท้ายของระบบแสดงผล VGA นี้ก็คือ VIDEO DAC (Digital to Analog Converter) ตัวแปลงข้อมูลดิจิตอลให้เป็นสัญญาณอะนาลอกเพื่อขับจอแสดงผล ไอบีเอ็มใช้ VIDEO DAC ของบริษัท Immos เบอร์ IMSG-171 ภายในประกอบด้วยตัวแปลงสัญญาณ 3 ชุด สำหรับ 3 สีคือ แดง เขียว น้ำเงิน และตารางค้นหาสี (Color Look-up Table) ที่รับข้อมูลขนาด 8 บิตจากแอตทริบิวต์คอนโทรลเลอร์มาเป็นตัวชี้รีจิสเตอร์ขนาด 18 บิต 1 ใน 256 รีจิสเตอร์ เพื่อเลือกข้อมูลที่จะนำไปแปลงเป็นสัญญาณอะนาลอกดังแสดงในรูปที่ 2.15



รูปที่ 2.14 (ก) แสดงการแปลงข้อมูลจากหน่วยความจำไปเป็นรูป (จุด) ในโหมดกราฟิก



รูปที่ 2.14 (ข) การแปลงจากรหัสไปเป็นตัวอักขระพร้อมทั้งแอตทริบิวต์



รูปที่ 2.15 แสดงตารางการแปลงข้อมูลสีของ DAC

จากรีจิสเตอร์จำนวน 256 รีจิสเตอร์ ทำให้สามารถแสดงสีได้พร้อม 256 สีแต่ภายใน 1 รีจิสเตอร์ ซึ่งมีขนาด 18 บิตจะมีสีแตกต่างกันได้ทั้งสิ้น 2^{18} = 256K สี และนี่คือเหตุผลที่ว่าระบบ VGA สามารถแสดงสีได้ 256 สีพร้อมกัน จากจำนวนสีทั้งสิ้น 256 K สี

ที่กล่าวมาทั้งหมดคือโครงสร้างทางฮาร์แวร์ของการ์ด VGA จะเห็นว่าภายใน VGA มีรีจิสเตอร์ต่าง ๆ มากมาย ซึ่งแต่ละตัวก็มีหน้าที่แตกต่างกันไปการที่จะแก้ไขค่าในรีจิสเตอร์นั้นควรมีการศึกษาดูก่อนว่ารีจิสเตอร์ตัวนั้นถ้าแก้ไขแล้วจะทำให้เกิดความเสียหายหรือไม่อย่างเช่น รีจิสเตอร์บางตัวของ CRT Controller ถ้ามีค่าไม่เหมาะสมจะทำให้จอภาพเสียหายได้ การใช้ฟังก์ชันต่าง ๆ ของการ์ด VGA นอกจากการกระทำกับรีจิสเตอร์โดยตรงแล้วสามารถทำได้โดยเรียกใช้บริการของรอมไบออสบนการ์ด ฟังก์ชันเหล่านี้ได้แก่ เซตโหมดของการแสดงผล อ่านโหมดของการแสดงผล เซตพาร์เล็ตรีจิสเตอร์ โหลดค่าลงในตารางแปลงอักขระ เป็นต้น

2.6 การโปรแกรมใช้การ์ด VGA

หลังจากที่ทราบโครงสร้างพื้นฐานของการ์ด VGA แล้ว การโปรแกรมเพื่อที่จะใช้งานการ์ด VGA เป็นที่สำคัญที่ควรจะทราบในลำดับถัดไป เนื่องจากการ์ด VGA มีความสลับซับซ้อนมากการใช้งานก็สามารถทำได้หลายรูปแบบดังนั้น ในบทความนี้จะกล่าวถึงวิธีการโปรแกรมใช้งานการ์ด VGA เบื้องต้น

ก่อนอื่นใดเราควรจะทำความรู้จักกับโหมดการแสดงที่เป็นมาตรฐาน ซึ่งการ์ด VGA สามารถแสดงได้ในตารางรูปที่ 2.16 จะแสดงให้เห็นโหมดต่าง ๆ ที่ระบบ VGA สนับสนุนโหมดการแสดงผลของการ์ดแสดงผล MDA และ CGA ในระบบเก่าด้วย

ต่อไปจะขออธิบายรายละเอียดของแต่ละโหมดการแสดงผลดังต่อไปนี้

2.6.1 การแสดงผลโหมด 0 และ 1 (สำหรับข้อความมีสี)

สำหรับในโหมดการแสดงโหมดนี้นั้นเป็นของการ์ด CGA บนการ์ด VGA ฟังก์ชันการทำงานทุกอย่างเหมือนกับการ์ด CGA ทุกประการ เพียงแต่ไม่มีขั้วต่อสำหรับสัญญาณวีดีโอคอนโพสิต จอภาพที่ใช้ในโหมดนี้จะเป็นจอ Color Display (CD), Enhanced Color Display (ECD), จอ VGA หรือจอหลายความถี่ (multifrequency) ที่มีความละเอียด

40 ตัวอักษร 25 บรรทัด ฟอนต์ตัวอักษรในโหมดนี้มีขนาด 8x8 จุดโหมดนี้เป็นโหมดข้อความที่มีรายละเอียดต่ำสุด

สำหรับซอฟต์แวร์ที่ทำงานบนระบบ CGA เดิม จะสามารถทำงานบนการ์ด VGA ได้ แต่จะต้องเป็นซอฟต์แวร์ที่ใช้ฟังก์ชันบริการของไบออสซอฟต์แวร์ที่กระทำโดยตรงต่อหน่วนความจำแสดงผลหรือติดต่อโดยตรงกับรีจิสเตอร์อินพุต/เอาต์พุตของการ์ดแสดงผล อาจจะมีปัญหาเกิดขึ้นในการทำงาน

การ์ด VGA ในโหมด 0 และ 1 นี้มี 8 เพจ การเลือกเพจสำหรับแสดงผลอาจจะควบคุมผ่านไปไบออสหรือผ่านทาง Start Address register ใน CRT Controller สำหรับเรื่องสีก็เป็นไปตามมาตรฐานคือ จะประกอบด้วย 8 สีตามปกติและอีก 8 สี เมื่อเพิ่มความเข้ม (intensity) ดังตารางรูปที่ 4 สำหรับจอภาพ VGA ที่มีความระเอียดสูง เมื่อแสดงผลในโหมดนี้ VGA จะใช้เทคนิคพิเศษในการจัดให้อัตราส่วนของภาพในแนวตั้งใกล้เคียวกับจอ CGA ซึ่งจอ CGA มีความละเอียดต่ำคือ 200 เส้นสแกน แต่ถ้าเป็นจอ VGA ใน 200 เส้น จะถูกสแกน 2 ครั้ง เป็น 400 เส้นสแกน เรียกเทคนิคนี้ว่า "Double Scanning"เทคนิคนี้จะใช้ในโหมด 0,1,2,3,4,5,6,D และ E

2.6.2 การแสดงผลโหมด 0* และ 1* (โหมดข้อความสี)

โหมด 0* และ 1* เป็นโหมดที่เหนือขึ้นมาจากโหมด 0 และ 1 การแสดงผลยังเป็นขนาด 40 ตัวอักษร 25 บรรทัด แต่ขนาดของตัวอักษรซึ่งในโหมดที่ 0 และ 1 มีขนาด 8x8 จุดก็จะเป็นขนาด 8x14 จุดแทน ดังนั้นในโหมดนี้จอภาพ CD จึงใช้ไม่ได้ โหมดนี้จะใช้ได้กับจอ ECD,VGA หรือจอหลายความถี่ (Multifrequency) เนื่องจากความละเอียดของตัวอักษรที่เพิ่มขึ้นจะทำให้อ่านง่ายขึ้นในเรื่องความคอมแพตติเบิ้ลกับระบบ CGA เดิมก็จะลดลงโดยเฉพาะในเรื่องเกี่ยวกับตัวอักษร เช่น การปรับขนาดของเคอร์เซอร์และการขีดเส้นใต้ของตัวอักษรเช่นเดียวกับโหมด 0 และ 1 โหมด 0* และ 1* ก็จะมี 8 เพจ สำหรับการแสดงผลเช่นกัน โดยแต่ละเพจก็จะมีตำแหน่งหน่วยความจำดังรูปที่ 2.20

Mode	Type	Colors	Resolution	Compatible Displays
0, 1	Color text	16	40×25 8×8 char cell	CD, ED, VGA Multifrequency
0*, 1*	Color text	16	40×25 8×14 char cell	ED, VGA Multifrequency
0+, 1+	Color text	16	40×25 9×16 char cell	VGA Multifrequency
2, 3 ➡	Color text	16	80×25 8×8 char cell	CD, ED, VGA Multifrequency
2*, 3* ➡	Color text	16	80×25 8×14 char cell	ED, VGA Multifrequency
2+, 3+	Color text	16	80×25 9×16 char cell	VGA Multifrequency
4, 5	Color graphics	4	320×200	CD, ED, VGA Multifrequency
6	Color graphics	2	640×200	CD, ED, VGA Multifrequency
7 ➡	Monochrome text	2	80×25 8×14 char cell	Monochrome VGA
7	Monochrome text		80×25 9×16 char cell	VGA only
8, 9, A	PC jr only			
D ➡	Color graphics	16	320×200	CD, ED, VGA Multifrequency
E ➡	Color graphics	16	640×200	CD, ED, VGA Multifrequency
F ➡	Mono graphics		640×350	Monochrome VGA
10 ➡	Color graphics	16	640×350	ED, VGA Multifrequency
11	Color graphics	2	640×480	VGA Multifrequency
12	Color graphics	16	640×480	VGA Multifrequency
13	Color graphics	256	320×200	VGA Multifrequency

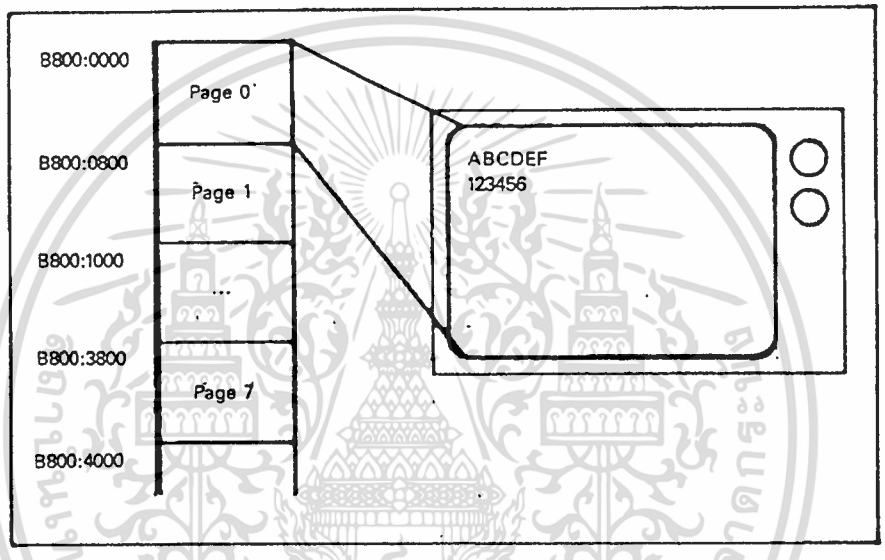
Most multifrequency displays are VGA compatible.
The original NEC Multisync is not.

CD = Color Display.
ED = Enhanced Color Display.

รูปที่ 2.16 ตารางแสดงโหมดแสดงผลมาตรฐานของระบบวีดีโอ IBM

เพจ 0 ตำแหน่ง B800:0000	เพจ 4 ตำแหน่ง B800:2000
เพจ 1 ตำแหน่ง B800:0800	เพจ 5 ตำแหน่ง B800:2800
เพจ 2 ตำแหน่ง B800:1000	เพจ 6 ตำแหน่ง B800:3000
เพจ 3 ตำแหน่ง B800:1800	เพจ 7 ตำแหน่ง B800:3800

รูปที่ 2.17



รูปที่ 2.18 แสดงตำแหน่งหน่วยความจำแต่ละเพจในโหมด 0 และ 1

Artribute	Standard Color	Intensified Color
000	Black	Gray
001	Blue	Light Blue
010	Green	Light Green
011	Cyan	Light Cyan
100	Red	Light Red
101	Magenta	Light Magenta
110	Brown	Yellow
111	Gray	White

รูปที่ 2.19 ตารางแสดงสีของอักษรในโหมดข้อความที่มีสี

เพจ 0 ตำแหน่ง B800:0000	เพจ 4 ตำแหน่ง B800:2000
เพจ 1 ตำแหน่ง B800:0800	เพจ 5 ตำแหน่ง B800:2800
เพจ 2 ตำแหน่ง B800:1000	เพจ 6 ตำแหน่ง B800:3000
เพจ 3 ตำแหน่ง B800:1800	เพจ 7 ตำแหน่ง B800:3800

รูปที่ 2.20

2.6.3 การแสดงผลโหมด 0+ และ 1+ (โหมดข้อความ สี)

สำหรับโหมดนี้ก็จะเป็นโหมดที่มีเพิ่มขึ้นมาจากการแสดงผลในโหมด 0 และ 1 ของ CGA ซึ่งมีความละเอียดของจอจะเป็น 40 ตัวอักษรต่อคอลัมน์และ 25 บรรทัด แต่ชุดอักษรที่ใช้ในโหมดของ CGA ที่เคยมีขนาด 8x8 จุดก็จะใช้ชุดตัวอักษรของ VGA แทน ซึ่งมีขนาดกว้าง 9 จุด สูง 16 จุด

2.6.4 การแสดงผลโหมด 2 และ 3 (โหมดข้อความสี)

เช่นเดียวกันในโหมด 0 กับโหมด 1 คือทั้งโหมด 2 กับโหมด 3 การทำงานทั้งสองโหมดเหมือนกันทุกประการในโหมดนี้จะมีความละเอียด 80 คอลัมน์ 25 บรรทัด และสำหรับจอภาพที่ใช้ในโหมดนี้คือ CD,ECD,VGA หรือจอภาพหลายความถี่ ตัวอักษรที่ปรากฏจะมีขนาด 8x8 จุดต่อตัวอักษรในโหมดนี้ทั้ง EGA และ VGA สามารถแสดงผลได้ 8 เพจ (ยกเว้นกรณีที่การ์ด EGA ของไอบีเอ็มที่มีหน่วยความจำเพียง 64 กิโลไบต์ จะได้การแสดงผลเพียงแค่ 4 เพจเท่านั้น) การเลือกเพจแสดงผลนั้นสามารถทำได้โดยผ่านทางไบออสหรือการควบคุมผ่านทาง Start Address register ใน CRT Controller การแสดงผลแต่ละเพจจะมีหน่วยความจำต่างกันดังรูปที่ 2.21

สีที่ใช้ในการแสดงผลก็เช่นเดียวกับในโหมด 0 และ 1 ในตารางในรูปที่ 2.21

เพจ 0 ตำแหน่ง B800:0000	เพจ 4 ตำแหน่ง B800:4000
เพจ 1 ตำแหน่ง B800:1000	เพจ 5 ตำแหน่ง B800:5000
เพจ 2 ตำแหน่ง B800:2000	เพจ 6 ตำแหน่ง B800:6000
เพจ 3 ตำแหน่ง B800:3000	เพจ 7 ตำแหน่ง B800:7000

รูปที่ 2.21

2.6.5 การแสดงผลโหมด 2* และ 3* (โหมดข้อความสี)

โหมด 2* และ 3* ต่างจากโหมด 2 และ 3 คือ ขนาดของตัวอักษรที่ใช้แสดงจะเป็น 8x14 จุดต่อตัวอักษร ซึ่งจะทำให้ไม่สามารถใช้กับจอภาพ เช่น CD ได้ แต่ใช้จอภาพแบบ EDC,VGA และจอภาพหลายความถี่และเนื่องจากความละเอียดของตัวอักษรมากกว่าในโหมด 2 และ 3 ดังนั้นฟังก์ชันเกี่ยวกับการเปลี่ยนขนาดของเคอร์เซอร์จึงแตกต่างกันจำนวนเพจสำหรับแสดงผลก็มี 8 เพจเช่นเดียวกับโหมด 2 และ 3 ตำแหน่งของหน่วยความจำแต่ละเพจก็ตรงกับโหมด 2 และ 3

2.6.6 การแสดงผลโหมด 2+ และ 3+ (โหมดข้อความสี)

โหมดนี้ก็เช่นเดียวกันเป็นโหมดการแสดงผลเพิ่มขึ้นมาจากโหมด 2 และ 3 ของมาตรฐานระบบ CGA คือมี 80 ตัวอักษรคอลัมน์และ 25 บรรทัด แต่ชุดตัวอักษรจะเป็นชุดของ VGA แทน ซึ่งตัวอักษร 1 ตัวมีขนาดกว้าง 9 จุดและสูง 16 จุด

2.6.7 การแสดงผลโหมด 4 และ 5 (โหมดกราฟิก 4 สี ความละเอียด 320x200 จุด)

ในโหมด 4 และ 5 เป็นโหมดแสดงกราฟิกของระบบ CGA ที่นิยมมากที่สุด ดังนั้นจึงไม่ต้องสงสัยเลยว่าในระบบ EGA หรือ VGA จะมีการแสดงผลโหมดนี้ด้วยความละเอียดของระบบนี้คือ 320 จุดในแนวนอน และ 200 จุดในแนวตั้ง จอภาพที่ใช้กับโหมดนี้คือจอ CD, ECD,VGA และจอภาพหลายความถี่

ในโหมด 4 และ 5 สีที่ใช้แสดงมีอยู่ด้วยกัน 4 สี ซึ่งอาจจะมาจากกลุ่มสี 2 กลุ่ม ดังในตารางในรูปที่ 2.22 สำหรับซอฟต์แวร์บนระบบ CGA เดิมที่มีการสั่งงานหรือควบคุมจอภาพผ่านทางรีจิสเตอร์อินพุต/เอาต์พุทของการ์ด CGA เมื่อนำมาใช้งานในโหมดนี้บนการ์ดแสดงผล EGA และ VGA อาจจะทำงานได้ไม่ถูกต้อง แต่เป็นซอฟต์แวร์ที่ควบคุมภาพทางไบออสจะทำงานได้ทั้งนี้เนื่องจากการ์ดแสดงผล EGA และ VGA ไม่คอมแพตติเบิ้ลกับจอ CGA ทั้งหมด

ในโหมดนี้จะมีเพจสำหรับแสดงผลเพียงอย่างเดียว โดยเริ่มที่ตำแหน่งหน่วยความจำ B800:0000 ข้อมูลของจุดบนจอจะเหมือนในระบบ CGA เดิมทุกประการคือหนึ่งจุดจะแทนด้วย 2 บิต การ์ด CGA รุ่นเก่าซึ่งใช้ชิพของโมโตโรล่า 6845 การสแกนในแนวตั้งจะถูก

Standard Color	Alternate Color
Black	Black
Light Cyan	Green
Light Magenta	Red
White	Brown

รูปที่ 2.22 ตารางแสดงสีมาตรฐานในโหมด 4 และ 5

จำกัดอยู่ที่ 128 เส้นสแกน ดังนั้นเพื่อให้ได้ความละเอียดในแนวตั้งสูงถึง 200 เส้น จึงต้องโปรแกรมให้ CRI Controller ทำงานในโหมดข้อความโดยมี 100 แถว แต่จำนวนจุดในความสูงของตัวอักษรจะมีเพียง 2 จุดเท่านั้น ดังนั้นสิ่งที่แสดงบนจอภาพจึงได้มาจากรูปร่างของตัวอักษร ซึ่งทำให้ตำแหน่งหน่วยความจำไม่ต่อเนื่องกันเหมือนตำแหน่งของจุดบนจอภาพ ต้องใช้การคำนวณ

2.6.8 การแสดงผลโหมด 6 (โหมดกราฟิก 2 สี ความละเอียด 640x200 จุด)

โหมด 6 เป็นการแสดงผลที่มีความละเอียดสูงสุดในโหมดกราฟิกของระบบ CGA โดยมีความละเอียด 640 จุดในแนวนอน และ 200 จุด ในแนวตั้งจอภาพที่ใช้ในโหมดนี้คือจอภาพ CD,ECD,VGA และจอภาพหลายความถี่ เช่นเดียวกันกับโหมดกราฟิกก่อนหน้านี้คือ

ซอฟต์แวร์ที่ใช้งานบน CGA อาจจะทำงานได้ไม่ถูกต้องถ้ามาทำงานบนการ์ด EGA หรือ VGA ในกรณีที่ซอฟต์แวร์ตัวนั้นไม่ได้ควบคุมจอภาพผ่านทางไบออส

ต้งที่อธิบายในโหมด 4 และ 5 แล้วตำแหน่งของหน่วยความจำจะไม่เรียงต่อกันและในโหมดนี้จะส่งแสดงผลเพียงเพจเดียวคือที่ตำแหน่ง B800:0000

2.6.9 การแสดงผลในโหมด 7 (โหมดข้อความของโมโนโครม)

ในโหมด 7 เป็นโหมดของข้อความซึ่งมีอยู่ในระบบ Momochrome Display Adapter (MDA) จอภาพที่ต้องใช้กับโหมดนี้คือจอโมโนโครม หรือจอ VGA ที่มีความละเอียดของตัวอักษร 80 ตัว อักษรต่อคอลัมน์ 25 แถว หนึ่งตัวอักษรประกอบด้วยจุด 8x14 จุดแต่เมื่อแสดงบนจอภาพจะถูกขยายเป็น 9x14 จุด (เพื่อให้ได้จำนวนจุดในแนวนอนเป็น 720 จุด) โดยที่บิตที่ 8 หรือบิตสุดท้ายของข้อมูลจะถูกขยายออกไปเป็นบิตที่เก้า ซึ่งช่วยให้การเขียนตัวอักษรที่ใช้สำหรับทำกรอบสี่เหลี่ยม (ตั้งแต่รหัสแอสกี C0 hex ถึง DF hex) มีรอยต่อเชื่อมสนิท อย่างไรก็ตามการขยายบิตที่เก้าสามารถจะควบคุมได้

โหมดนี้มีเพจสำหรับแสดงผล 8 เพจ (ยกเว้นกรณีของการ์ด EGA ของไอบีเอ็ม ซึ่งมีหน่วยความจำ 64 กิโลไบต์จะมี 4 เพจ) โดยแต่ละเพจมีตำแหน่งหน่วยความจำดัง รูปที่ 2.23 ส่วนแอตทริบิวต์ของตัวอักษรก็ยังคงเป็นเช่นเดิมโมโนโครม ซึ่งประกอบด้วย กะพริบ เข้มขีดเส้นใต้และรีเวิร์สวิดีโอ

เพจ 0 ตำแหน่ง B000:0000	เพจ 4 ตำแหน่ง B000:4000
เพจ 1 ตำแหน่ง B000:1000	เพจ 5 ตำแหน่ง B000:5000
เพจ 2 ตำแหน่ง B000:2000	เพจ 6 ตำแหน่ง B000:6000
เพจ 3 ตำแหน่ง B000:3000	เพจ 7 ตำแหน่ง B000:7000

รูปที่ 2.23

2.6.10 การแสดงผลโหมด 7+ (โหมดข้อความโมโนโครม)

โหมดนี้เป็นโหมดเพิ่มขึ้นมาจากโหมดข้อความของ MDA มาตรฐานโดยชุดตัวอักษรของ VGA ขนาด 9x16 จุด

2.6.11 การแสดงผลโหมด D (โหมดกราฟิก 16 สี 320x200 จุด)

ในโหมด D นี้ไม่เหมือนการแสดงผลโหมดต่าง ๆ ที่ผ่านมา เพราะว่าไม่ใช่ทำเพื่อความคอมแพตติเบิ้ลกับระบบแสดงผลเก่า ถึงแม้ความละเอียดจะเท่ากับโหมด 4 ของ CGA แต่ในโหมด D สามารถแสดงสีได้ถึง 16 สีและเพราะความละเอียดของจอภาพที่จำกัดไว้ที่ 320x200 จุด จึงทำให้มีซอฟต์แวร์จำนวนน้อยมากที่ทำงานบนโหมดนี้คือจอ CD,ECD,VGA และจอภาพหลายความถี่บางชนิด สำหรับในการ์ด EGA ที่มีหน่วยความจำ 256 กิโลไบต์ จะมีเพจสำหรับแสดงผลได้ 8 เพจ และสำหรับการ์ด EGA ที่มีหน่วยความจำขนาด 128 กิโลไบต์และ 64 กิโลไบต์ จะมีเพจแสดงเพียง 4 เพจและ 2 เพจตามลำดับ แต่ละเพจอยู่ที่ตำแหน่งหน่วยความจำดังรูปที่ 2.24

เพจ 0 ตำแหน่ง A000:0000	เพจ 4 ตำแหน่ง A000:4000
เพจ 1 ตำแหน่ง A000:1000	เพจ 5 ตำแหน่ง A000:5000
เพจ 2 ตำแหน่ง A000:2000	เพจ 6 ตำแหน่ง A000:6000
เพจ 3 ตำแหน่ง A000:3000	เพจ 7 ตำแหน่ง A000:7000

รูปที่ 2.24

ในโหมดนี้นั้นตำแหน่งของจุดกับตำแหน่งความจำจะเรียงต่อกันเรื่อยไป และมีสีใช้งาน 16 สีอยู่ในตารางรูปที่ 2.26.

2.6.12 การแสดงผลโหมด E (กราฟิก 16 สี ขนาด 640x200 จุด)

โหมด E กับโหมด 6 จะเป็นกราฟิกความละเอียดเท่ากันต่างกันที่โหมด E สามารถแสดงสี 16 สีและเช่นเดียวกันโหมด D เนื่องจากความละเอียดยังไม่ดีพอ ซอฟต์แวร์สมัยใหม่จึงไม่นิยมทำงานในโหมด E นี้ จอที่ใช้สำหรับโหมดนี้คือ CD,ECD,VGA และจอความถี่หลายความถี่ สำหรับการ์ด EGA ที่มีหน่วยความจำเต็มที่ถึง 256 กิโลไบต์ สามารถจะแสดงผลในโหมดนี้ได้ถึง 4 เพจ ถ้ามีหน่วยความจำ 128 กิโลไบต์จะแสดงได้ 2 เพจ และถ้ามีหน่วยความจำเพียง 64 กิโลไบต์ ก็จะแสดงผลได้เพียงเพจเดียว ตำแหน่งหน่วยความจำแต่ละเพจเป็นดังรูปที่ 2.25 ในโหมดนี้นั้นตำแหน่งของจุดกับตำแหน่งหน่วยความจำจะเรียงต่อกันและสีที่ใช้งานจะอยู่ในตารางที่ 2.26

เพจ 0 ตำแหน่ง A000:0000
เพจ 1 ตำแหน่ง A000:4000
เพจ 2 ตำแหน่ง A000:8000
เพจ 3 ตำแหน่ง A000:C000

รูปที่ 2.25

2.6.13 การแสดงผลโหมด F (โหมดกราฟิก ขาวดำ 640x350)

เฉพาะการ์ด EGA และ VGA เท่านั้น ที่สามารถแสดงผลในโหมดนี้เพราะว่าไม่ใช่โหมดการแสดงผลที่สร้างขึ้นมาเพื่อให้คอมแพตทิเบิ้ลกับการ์ดใด ๆ จอภาพที่ใช้กับโหมดนี้คือจอ Monochrome VGA ซึ่งต้องมีความละเอียดในแนวนอน 640 จุด และความละเอียดในแนวตั้ง 350 จุด (จะพบว่าความละเอียดน้อยกว่าโหมดกราฟิกของเฮอร์คิวลิส ที่มีความละเอียด 720x380 จุด) ในโหมด F ตำแหน่งของจุดบนจุดถึงตำแหน่งของหน่วยความจำจะเรียงต่อกัน ไม่เหมือนกับโหมดกราฟิกของเฮอร์คิวลิส การแสดงผลจะมีได้ 2 เพจ ยกเว้นการ์ด EGA ของไอบีเอ็มที่มีหน่วยความจำเพียง 64 กิโลไบต์ จะมีการแสดงผลได้เพียง 1 เพจ ตำแหน่งหน่วยความจำของแต่ละเพจเป็นดังรูปที่ 2.27

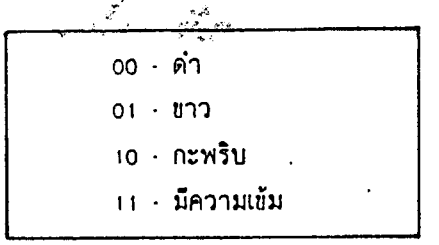
Plane	Full (128K+)	Partial (64 KB)
3 2 1 0	Colors	Colors
0 0 0 0	Black	Black
0 0 0 1	Blue	Blue
0 0 1 0	Green	Black
0 0 1 1	Cyan	Blue
0 1 0 0	Red	Red
0 1 0 1	Magenta	White
0 1 1 0	Brown	Red
0 1 1 1	White	White
1 0 0 0	Dark Gray	Black
1 0 0 1	Light Blue	Blue
1 0 1 0	Light Green	Black
1 0 1 1	Light Cyan	Blue
1 1 0 0	Light Red	Red
1 1 0 1	Light Magenta	White
1 1 1 0	Yellow	Red
1 1 1 1	Intens. White	White

รูปที่ 2.26 ตารางแสดงสีมาตราฐานในกราฟิก 16 สี

เพจ 0 ตำแหน่ง A000:0000
เพจ 1 ตำแหน่ง A000:8000

รูปที่ 2.27

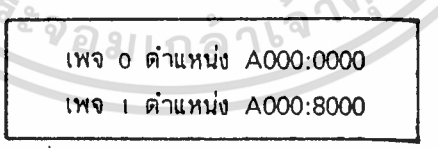
ส่วนเรื่องของสีนั้นจะเกิดมาจากหน่วนความจำเพลนสี 2 เพลน ซึ่งจะทำให้จุดแต่ละจุดมีแอตทริบิวต์ได้ดังรูปที่ 2.28



รูปที่ 2.28

2.6.14 การแสดงผลโหมด 10H (โหมดกราฟิกสีแบบ Enchanced ขนาด 640x350 จุด)

ในโหมด 10H เป็นโหมดที่เพิ่มขึ้นใน EGA และ VGA โหมดนี้เป็นโหมดที่นิยมใช้กับสำหรับโปรแกรมกราฟิกทั่วไป ความละเอียดของโหมดจะมีถึง 640 จุดในแนวนอน และ 350 จุดในแนวตั้ง โหมดนี้จะใช้กับจอภาพชนิด CD ไม่ได้ ต้องใช้กับจอภาพ ECD,VGA หรือจอหลายความถี่ การควบคุมสีใช้เพลนสีถึง 4 เพลน ดังนั้นสามารถแสดงผลได้ 16 สี ในเวลาเดียวกัน ยกเว้นการ์ด EGA ของไอบีเอ็มที่มีหน่วยความจำเพียง 64 กิโลไบต์ ซึ่งจะใช้เพลนสีได้แค่ 2 เพลน ในโหมด 10H จะแสดงผลได้ 2 เพจ (สำหรับ EGA ที่มี 64 กิโลไบต์จะมีได้ 1 เพจ) โดยหน่วยความจำแต่ละเพจดังรูปที่ 2.29



รูปที่ 2.29

สีที่ใช้แสดงในโหมด 10H อยู่ในตารางรูปที่ 2.26 โหมดการแสดงต่อไปนี้จะเป็นโหมดการแสดงที่มีเพิ่มขึ้นเฉพาะในระบบ VGA เท่านั้น

2.6.15 การแสดงผลโหมด 11H (โหมดกราฟิก 2 สี 640x480 จุด)

เป็นโหมดการแสดงผลกราฟิกที่มีความละเอียดสูงสุดแต่แสดงสีได้เพียง 2 สี ในโหมดนี้สามารถจะใช้แสดงข้อความได้ถึง 30 แถว แถวละ 80 คอลัมน์ หน่วยความจำแสดงผลเริ่มต้นที่ A000:0000

2.6.16 การแสดงผลโหมด 12H (โหมดกราฟิก 16 สี 640x480 จุด)

เป็นโหมดการแสดงผลกราฟิกที่มีความละเอียดและแสดงสีได้ถึง 16 สีพร้อมกันดังในตารางรูปที่ 22 หน่วยความจำแสดงผลเริ่มที่ตำแหน่ง A000:0000

2.6.17 การแสดงผลโหมด 13 (โหมดกราฟิก 256 สี 320x200 จุด)

ในกราฟิกโหมดนี้สามารถแสดงสีได้พร้อมกันถึง 256 สี แต่ความละเอียดจะลดลงเหลือเพียง 320x200 จุด หน่วยความจำแสดงผลเริ่มที่ตำแหน่ง A000:0000

2.7 หน่วยความจำแสดงผลแต่ละโหมด

2.7.1 หน่วยความจำแสดงผลของโหมด 4 และ 5 (กราฟิกของ CGA 4 สี)

ในโหมดกราฟิกนี้หนึ่งจุดจะประกอบด้วยข้อมูล 2 จุดต่อไบต์และข้อมูลก็จะถูกเก็บไว้ในหน่วยความจำเพลนที่ 0 การเรียงลำดับจุดกับข้อมูลจะเรียงจากบิตสูงสุดไปบิตต่ำเช่น ข้อมูลในบิตที่ 7 และ 6 ของไบต์แรกในเพลน 0 จะแทนจุดบนซ้ายสุดของจอภาพในรูปที่ 2.30 แสดงการจัดคู่กันระหว่างจุดบนจอภาพกับข้อมูลในหน่วยความจำ ซึ่งข้อมูลในหน่วยความจำกับจอภาพจะไม่ต่อเนื่องกัน ครึ่งบนของหน่วยความจำแสดงผลจะเก็บข้อมูลของจุดบนเส้นสแกนเส้นคู่ ส่วนหน่วยความจำแสดงผลครึ่งล่างก็จะเก็บข้อมูลของจุดบนเส้นสแกนเส้นคี่

การอ้างอิงข้อมูลในหน่วยความจำแสดงผลจากจุด X,Y ใดๆ บนจอภาพก็จะสามารถทำได้โดยอาศัยสูตรดังนี้ (อ้างอิงจุดบนจอภาพโดยจุดมุมบนซ้ายมือให้เป็นจุด 0,0 และจุดล่างขวามือสุดให้เป็นจุด 319,199)

ถ้า Y เป็นเลขคู่

ตำแหน่งของไบต์ = 80*(Y/2) + (X/4)

ถ้า Y เป็นเลขคี่

ตำแหน่งของไบต์ = 4096 + 80*((Y-1)/2) + (X/4)

ตำแหน่งของบิตในไบต์ = (X modulo 2)*2

2.7.2 หน่วยความจำแสดงผลของโหมด 6 (กราฟิกของ CGA 2 สี)

โหมดนี้มีความละเอียด 640x200 หนึ่งจุดแทนด้วยหนึ่งบิต หรือ 8 จุดต่อไบต์ ข้อมูลบิตที่เป็นศูนย์คือจุดที่เป็นสีดำ ข้อมูลจะถูกเก็บในหน่วยความจำแสดงผลเพลน 0 เช่นเดียวกับโหมด 4 และ 5 คือข้อมูลในหน่วยความจำแสดงผลไม่เรียงต่อเนื่องกันต้องอาศัยการคำนวณดังนี้ (อ้างอิงจากจุดบนซ้ายมือสุดคือจุด 0,0 และจุดล่างขวามือคือจุด 639,199) ดังรูปที่ 2.31

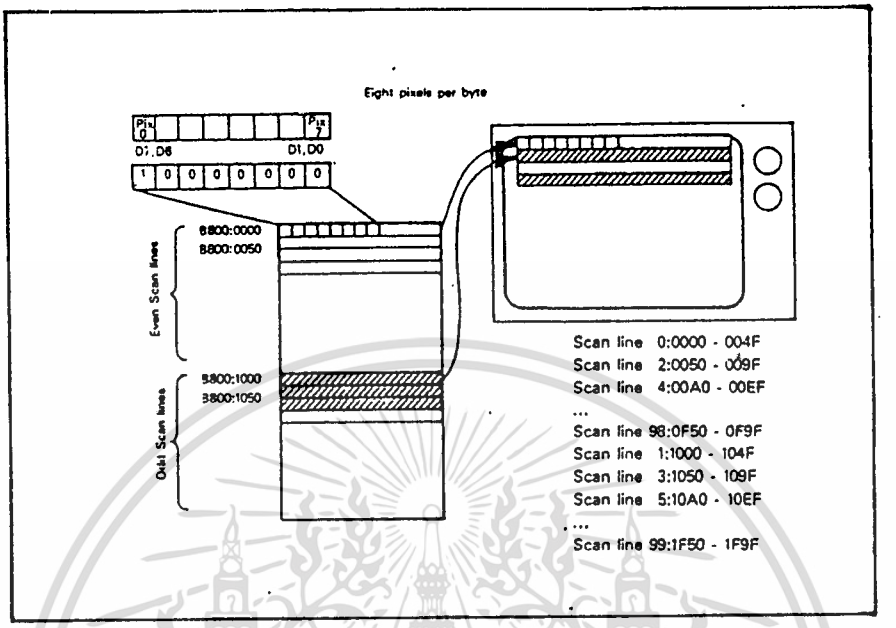
ในกรณี Y เป็นเลขคู่

ตำแหน่งของไบต์ = 80*(Y/2) + (X/8)

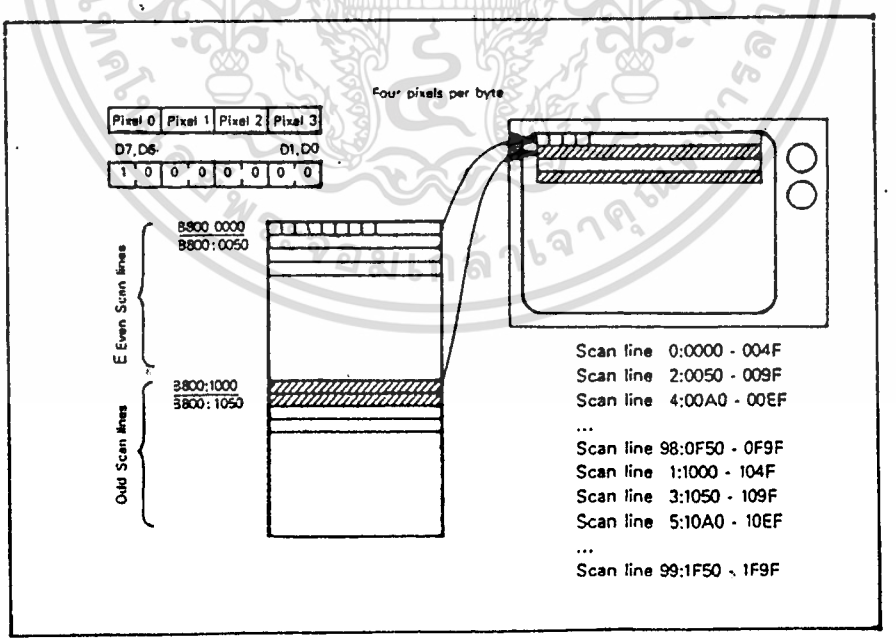
กรณี Y เป็นเลขคี่

ตำแหน่งของไบต์ = 4096 + 80*((Y-1)/2) + (X/8)

ตำแหน่งบิตในไบต์ = 7-(X modulo 8)



รูปที่ 2.30 แสดงการจัดเรียงหน่วยความจำในโหมดกราฟิก 4 และ 5



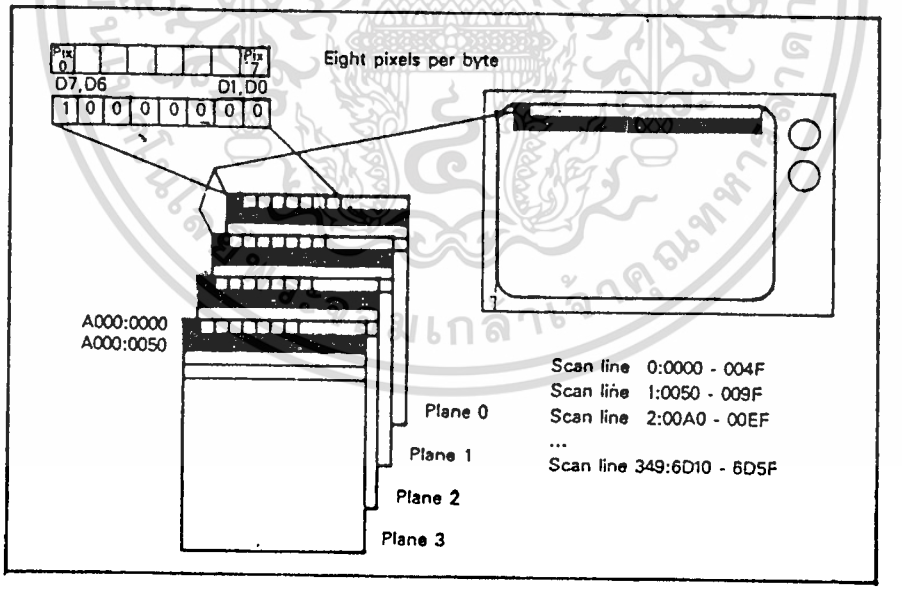
รูปที่ 2.31 แสดงการจัดเรียงหน่วยความจำในโหมดกราฟิก 6

2.7.3 หน่วยความจำแสดงผลในโหมด F (กราฟิกของโมโนโครม)

ในโหมดนี้การจัดเรียงหน่วยความจำแสดงผลจะต่อเนื่องกันความละเอียด 640x350 จุด หนึ่งจุดบนจอจะถูกแทนด้วยหนึ่งบิตบนเพลนสี (Color plane) คือเพลน 0 และเพลน 1 ดังนั้นข้อมูลในหนึ่งเพลน 1 ไบต์จะแทนได้ 8 จุด สีจะถูกควบคุมโดยบิตเพลน 0 และเพลน 1 ซึ่งจะได้ 4 สีคือ ดำ สว่าง สว่างเข้มและกะพริบ ทั้งสองเพลนสามารถควบคุมได้ทาง Color Write Enable register ของ Sequencer ในรูปที่ 2.32 แสดงหน่วยความจำที่สัมพันธ์กับจุดบนจอภาพและการหาตำแหน่งหน่วยความจำจะหาได้จากสูตร (อ้างอิงจากจุดบนซ้ายมือ 0,0 และจุดล่างขวามือ 639,349)

ตำแหน่งไบต์ = (Y*80) + (X/8)

ตำแหน่งบิต = 7 - (X modulo 8)



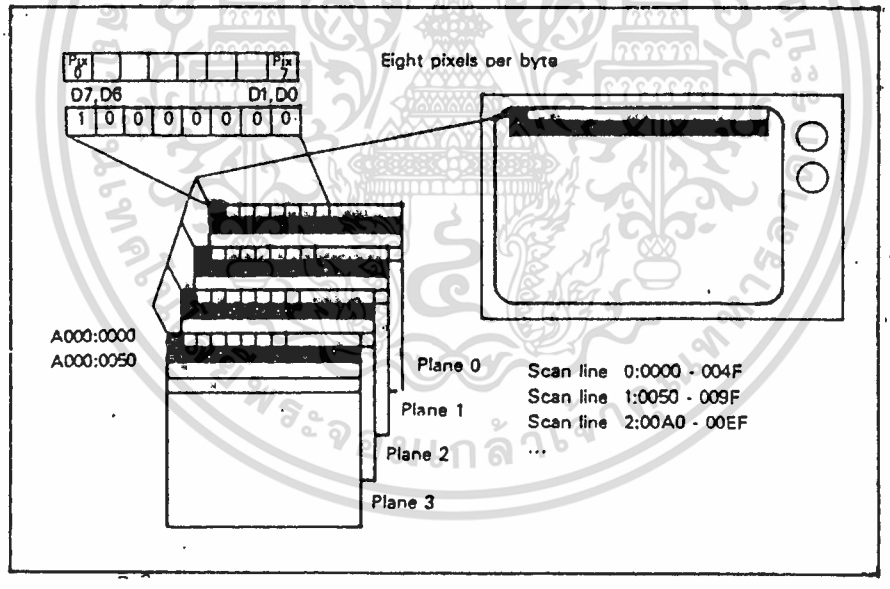
รูปที่ 2.32 การจัดเรียงหน่วยความจำในโหมดกราฟิก F

2.7.4 หน่วยความจำแสดงผลโหมด 10H (กราฟิกสี Enchanced)

มีซอฟต์แวร์จำนวนมากที่ทำงานบนโหมดความละเอียดของจอภาพมีขนาด 640x350 จุด ข้อมูลที่แทนจุดบนจอภาพอยู่เพลนสีทั้ง 4 เพลนหนึ่งบิตในแต่ละเพลนแทนจุดหนึ่ง ซึ่งสีของจุดจะถูกควบคุมโหยข้อมูลแต่ละบิตทั้ง 4 เพลน สีที่ตรงกัน ดังนั้นจะสามารถแสดงสีได้ 16 สี ในรูปที่ 2.33 แสดงหน่วยความจำบนเพลนสีทั้งสี่ที่สัมพันธ์กับจุดบนจอภาพ การหาตำแหน่งของหน่วยความจำจากจุดบนจอภาพหาได้จาก

ตำแหน่งไบต์ = Y*80 + X/8

ตำแหน่งบิต = 7 - (X modulo 8)



รูปที่ 2.33 การจัดเรียงหน่วยความจำโหมดกราฟิก D,E,10H,12H

2.7.5 หน่วยความจำแสดงผลโหมด D และ E (กราฟิก 16 สี)

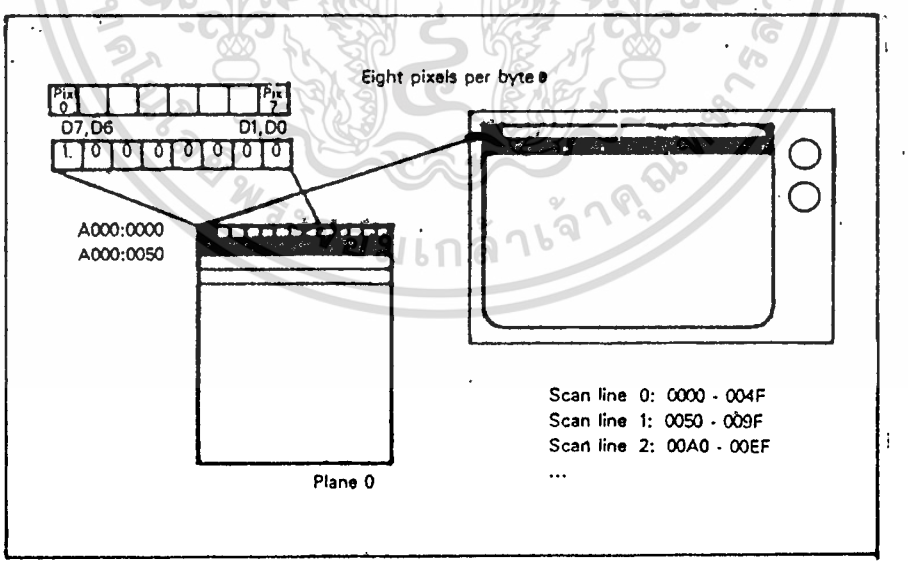
โหมด D และ E จะคล้ายกับโหมด 10 ต่างกันที่จำนวนจุดหรือความละเอียดของจอภาพ คือโหมด D มีความละเอียด 320x200 จุด ส่วนโหมด E มีความละเอียด 640x200 ทั้งสองโหมดมีซอฟต์แวร์ที่ทำงานบนทั้งโหมดน้อยทั้งนี้เนื่องจากความละเอียดน้อยกว่าโหมด 10

2.7.6 หน่วยความจำแสดงผลโหมด 11 (กราฟิก 2 สี)

โหมดนี้มีอยู่บน VGA เท่านั้นโดยมีความละเอียดเป็น 640x480 จุด มีความสามารถแสดงสีได้เพียง 2 สีดังกล่าวมาแล้ว ข้อมูลบนหน่วยความจำ 1 บิตจะแทน 1 จุดบนจอภาพและถูกเก็บอยู่ในเพลน 0 เท่านั้น ข้อมูลบนหน่วยความจำแทนจุดบนจอภาพอย่างต่อเนื่อง การหาตำแหน่งหน่วยความจำได้จากสูตร

ตำแหน่งไบต์ = (Y*80) + (X/8)

ตำแหน่งบิต = 7 - (X modulo 8)



รูปที่ 2.34 การจัดเรียงหน่วยความจำโหมดกราฟิก 11H

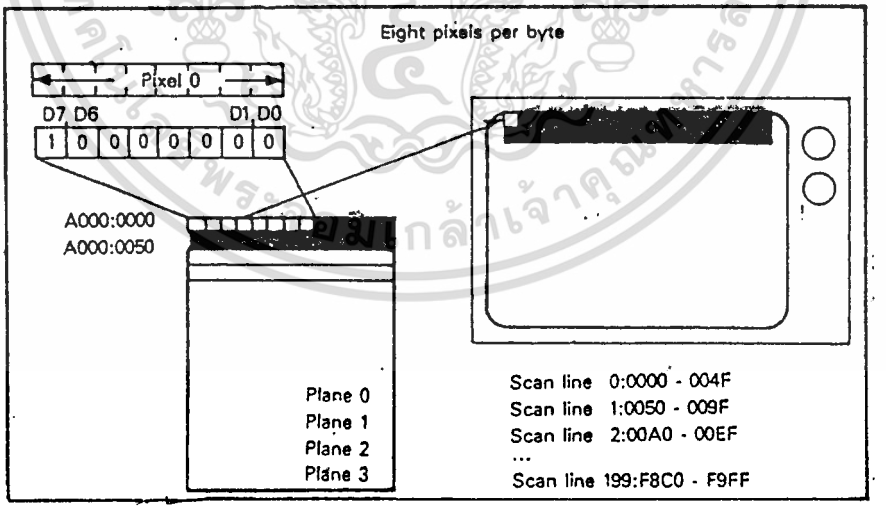
2.7.8 หน่วยความจำแสดงผลโหมด 12H (กราฟิก 16 สี)

โหมด 12H มีลักษณะคล้ายโหมด 10H ต่างกันที่ความละเอียดในแนวตั้งจะเป็น 480 จุดแทน 350 จุด ข้อมูลจะเก็บลงทั้ง 4 เพลน เหมือนโหมด 10H แสดงสีได้ 16 สีพร้อมกัน การจัดเรียงหน่วยความจำจะคล้ายรูปที่ 2.33

2.7.9 หน่วยความจำแสดงผลโหมด 13H (กราฟิก 256 สี)

ในโหมด 13H นี้สามารถแสดงสีได้ถึง 256 สีพร้อมกัน แต่ความละเอียดจะต่ำคือ 320x200 จุด การจัดเรียงหน่วยความจำอยู่ใน รูปที่ 2.35 หนึ่งจุดบนจอภาพจะแทนด้วยข้อมูลขนาด 1 ไบต์ ซึ่งจัดเก็บอยู่บนทั้ง 4 เพลน การหาตำแหน่งของหน่วยความจำจากจุดบนจอภาพหาได้จากสูตร

ตำแหน่งไบต์ = (Y*320) + X



รูปที่ 2.35 การจัดเรียงหน่วยความจำโหมดกราฟิก 13H

บทที่ 3
โครงสร้างไฟล์

3.1 โครงสร้างของ PCX ไฟล์

ไฟล์ PCX เริ่มต้นด้วยข้อมูล 128 ไบต์ ซึ่งจะทำการเก็บข้อมูลต่างๆเพื่อนำไปใช้ในการ Restore ภาพ ดังรูปที่ 3.1

ข้อมูล Color Map ถูกใช้ในการแทนค่าเป็น Palette Register 16 ตัว โดยที่ Palette Register 1 ตัว ของระบบแสดงผลอีจีเอจะบรรจุข้อมูล 6 บิตเท่านั้นโดยแต่ละ 2 บิตจะแทนสีหลักคือ แดง,น้ำเงิน,เขียว ดังรูปที่ 3.2 ตัวอักษรตัวใหญ่จะแทนความเข้มของสี 75% ตัวอักษรเล็กแทน 25% แต่ละสีหลักจะมี 4 ระดับของการแสดงสีคือ 0%, 25%, 75%, 100% (ค่าบิตของตัวอักษรตัวเล็กและตัวอักษรตัวใหญ่เป็น '1' หมด) ดังรูปที่ 3.3 เช่น

- ค่ารีจิสเตอร์เป็น "00100001" หมายถึง แสดงสีแดงที่มีความเข้ม 75% และแสดงสีน้ำเงินที่มีความเข้ม 25%
- ค่ารีจิสเตอร์เป็น "00100101" หมายถึง แสดงสีแดงที่มีความเข้ม 100% และแสดงสีน้ำเงินที่มีความเข้ม 25%
- ค่ารีจิสเตอร์เป็น "00000000" หมายถึง ไม่แสดงสีใดเลย เป็นต้น

ข้อมูล Color Map ในตอนต้นไฟล์ (Header File) ประกอบด้วยข้อมูล 16 ชุดๆ ละ 3 ไบต์ ไบต์ที่ 1 ของแต่ละชุด จำนวน 3 ไบต์ย่อยนั้นๆ เป็นค่าสำหรับแทนสีแดงโดยถูกนำมาสร้างเป็นเลข 4 จำนวนตั้งแต่ 0-3 (00,01,10,11) และตัวเลขนี้จะต้องคูณกับ 85 เพื่อจะทำการเก็บลงไฟล์ในส่วนเฮดเดอร์ไฟล์ ในกรณีเดียวกันสีเขียวคือไบต์ที่ 2 และสีน้ำเงินคือไบต์ที่ 3 วิธีการนี้จะถูกทำซ้ำๆ กัน 16 รอบ เพื่อจะได้ค่าของสีแดง น้ำเงิน เขียว จนครบ 16 ชุด

ไบต์ที่	ขนาด (Byte)	ชื่อข้อมูล	รายละเอียด
0	1	Password	ค่า '0aH' = ไฟล์ .PCX
1	1	Version	เก็บค่า Version (รุ่น) ของ PC Paintbrush ดังนี้ 0=Version 2.5 2=Version 2.8 with Palette 3=Version 2.8 without Palette 5=Version 3.0
2	1	Encoding	ค่า = 1
3	1	Bits per Pixel	จำนวนของบิตที่ใช้แสดง 1 Pixel จาก 1 Plane (ระนาบสี) 1 = EGA, VGA, or HERC 4 = CGA
4	8	Windows-Dimemsions	ค่า integer 4 ค่า (ค่าละ 2 Byte) ให้ตำแหน่งมุมบนซ้ายและมุมล่างขวาของภาพ กำหนดให้รูปแบบ x1, y1, x2, y2
12	2	Horizontal-Resolution	ความละเอียดของการแสดงภาพในแนวนอน 640 = EGA, VGA 320 = CGA 720 = HERCULES
14	2	Vertical-Resolution	ความละเอียดของการแสดงภาพในแนวตั้ง 480 = VGA 350 = EGA 200 = CGA 348 = HERC
16	48	Color Map	ข้อมูล Color Palette ดูรูป 2
64	1	Reserved	
65	1	No. of Plane	จำนวนของ Planes ที่ใช้แสดงภาพ
66	1	Byte per Line	จำนวนไบต์ต่อการสแกน 1 บรรทัด
68	2	Palette Info	How to interpret the palette
70	8	Maximum x value	Special for fractal files
78	8	Minimum x value	Special for fractal files
86	8	Maximum y value	Special for fractal files
94	8	Minimum y value	Special for fractal files
102	8	P Value	Special for fractal files
110	8	Q Value	Special for fractal files
118	10	Not used	

หมายเหตุ ข้อมูลไบต์ที่ 70-127 ใช้เฉพาะ Fractal Programming

รูปที่ 3.1 ตารางโครงสร้าง PCXHEADER

ไบต์ที่	Palette	ไบต์ที่	Palette
16,17,18	0	40,41,42	8
19,20,21	1	43,44,45	9
22,23,24	2	46,47,48	10
25,26,27	3	49,50,51	11
28,29,30	4	52,53,54	12
31,32,33	5	55,56,57	13
34,35,36	6	58,59,60	14
37,38,39	7	61,62,63	15

รูปที่ 3.2 แสดงตำแหน่งในการเก็บสี จำนวน 3 ไบต์ คือ แดง,เขียว,ฟ้า

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
– x	– x	R 75 %	G 75 %	B 75 %	r 25 %	g 25 %	b 25 %

รูปที่ 3.3

การแทนความหมาย Palette

ถ้า Palette Register มีค่า xx100001 แล้ว (ใน Palette ใดๆ (0-15))

บิตที่ 5,2 -> แดง

บิตที่ 4,1 -> เขียว

บิตที่ 3,0 -> น้ำเงิน

แดง คือ '10' = 2 , เขียว '00' = 0, น้ำเงิน '01' = 1

ดังนั้นค่าที่จะเก็บใน Color Map ของ Palette ใดๆ (0-15) คือ (x85)

Red	Green	Blue
170	0	85

หรือถ้า Palette Register มีค่า 'xx111111' แล้ว ค่าที่จะเก็บคือ

Red	Green	Blue
255	255	255

ข้อแตกต่างพิเศษในจอวีจีเอ

จอภาพวีจีเอจะมีการควบคุมสีที่แตกต่างกับอีจีเอคือ ในแต่ละ Palette Register จะเก็บลำดับที่ของ Color Register ไว้ 1 ตัว และตัวของ Color Register จะใช้เนื้อที่จำนวน 6 บิต ดังนั้นค่าสีที่เป็นไปได้จะเป็น 64 ค่า (หมายถึง Shades ของสีนั้นเอง) ของแต่ละสี ในวีจีเอสามารถอ่านค่า (ค่า 6 บิตของ สีแดง น้ำเงิน เขียว) ข้อมูลใน Palette Register ใน Color Register เพื่อสร้าง Color Map ขึ้นมา โดยอ่านค่าแต่ละ Palette Register และนำไปแอดเดรส Color Register เพื่อทำการอ่าน Color Register อีกที, ทำการคูณสีแดง, น้ำเงิน, เขียว ด้วย 4 และเก็บผลลงในกลุ่ม 3 ไบต์ของ Palette นั้นๆ

ในวีจีเอจะมีสถานะสี 256 สีที่แตกต่างกัน รูปแบบการใช้งานเหมือนกับการแสดง 16 สี ข้อมูลใน Palette จะยาวกว่า 16 สี ข้อมูลนี้จะอยู่ที่ส่วนท้ายของไฟล์ .PCX การเข้าถึงข้อมูลเป็นดังนี้ต้องตรวจไบต์ตรงตำแหน่งเวอร์ชันของไฟล์ว่าเป็น 5 (เวอร์ชัน 3.0) หรือไม่ และข้อมูล Palette จะต้องนับถอยหลังจากท้ายไฟล์ไป 769 ไบต์ ในไบต์นั้นจะต้องเป็น 0cH (12DECIMAL) จากนั้นข้อมูลที่ท้ายไฟล์จะเป็นข้อมูล Palette 256 สี

พื้นฐานในการอ่านข้อมูลเก็บลงไฟล์

ภาพที่จะอ่านจะถูกอ่านตามแนวของจอภาพในแนวนอนจากซ้ายไปขวา เริ่มที่ Pixel ที่ตำแหน่งบน, ซ้ายไปทางขวาจนสุดภาพแล้วจึงอ่านในแถวถัดไปโดยในวีจีเอและอีจีเอ ซึ่งมีหน่วยความจำหลายๆ plane ดังนั้นใน 1 แถว (scanline) จะอ่านข้อมูลในทุก plane เริ่มที่ plane 0,1,2,3 ตามลำดับจนครบแล้วจึงอ่านแถวถัดไปจนหมดภาพที่ต้องการเก็บ

การเข้ารหัสไฟล์ .PCX

ไฟล์ .PCX มีวิธีการเข้ารหัสที่เรียกว่า Run-Length Encoding (RLE) ซึ่งเป็นวิธีการหนึ่งในการลดขนาดไฟล์ดังรายละเอียดดังนี้

ถ้าไบต์ใดๆ มีค่าไม่เหมือนกับไบต์อื่นๆ และถ้าค่า 2 บิตบนไม่เท่ากับ "11" ไบต์นั้นจะถูกเก็บลงไฟล์เลยนอกนั้นจะมีตัวนับ (Counter) อยู่คอยนับว่าเหมือนกันกี่ไบต์ แต่ต้องไม่เกิน 63 ถ้าเกินให้นำค่าที่ได้ OR กับ C0h แล้วเก็บค่าตัวนับนั้นลงในไฟล์ แล้วตามด้วยค่าของไบต์นั้นๆ จากนั้นจึงเริ่มนับใหม่เป็น 1 ต่อไป ถ้าในกรณีของไบต์เดียวที่มีค่าบิตบนเป็น "11" ให้เขียนตัวนับ = 1 (OR with C0h ='C1h') ลงไปในไฟล์และตามด้วยค่าไบต์นั้น ดังแสดงเป็นในรูปที่ 3.4

เงื่อนไข	การกระทำ
ค่าที่เหมือนไบต์อื่น	นับ Counter + 1
ค่า < 'C0h' ไม่เหมือนไบต์อื่น	เก็บลงไฟล์เลย
ค่า > 'C0h', ไม่เหมือนไบต์อื่น	เก็บ 'C1h' ลงไฟล์ตามด้วยค่าไบต์นั้น
Counter > 63	Counter=1, เก็บค่า 'FFh' และค่าไบต์ลงไฟล์

รูปที่ 3.4

การถอดรหัสไฟล์ .PCX

การถอดรหัสใช้วิธีการตรงกันข้ามกับการเข้ารหัส พอสรุปได้ดังนี้

อ่านข้อมูลมา 1 ไบต์ แล้วตรวจดูว่ามากกว่า 'C0h' หรือไม่ (ค่า 2 บิตบนเป็น 11) ถ้าใช่ค่านี้จะเป็นค่าตัวนับทันที (XOR ด้วย C0h จะได้ค่าจริงที่น้อยกว่า 63 ออกมา) ซึ่งไบต์ต่อไปก็จะเป็นค่าของไบต์ (value) แล้วทำการขยายข้อมูลออกมาตามจำนวนตัวนับนั้นๆ แล้วเก็บในหน่วยความจำ ถ้าค่าไบต์ที่อ่านน้อยกว่า C0h จะทำการเก็บข้อมูลไบต์นั้นตัว

เดียว ลงหน่วยความจำที่มีอยู่ได้เลยโดยไม่ต้องขยายข้อมูล ทำเช่นนี้จนจบไฟล์ก็จะได้ข้อมูลของไฟล์ ทั้งหมด จากนั้นนำข้อมูลในหน่วยความจำที่ได้มาแสดงบนจอภาพ โดยใช้ฟังก์ชัน POKEB() ในการนำเอาข้อมูลไปไว้ในหน่วยความจำแสดงผล (display memory) หรืออาจจะใช้ฟังก์ชั่น PUTPIXEL() แสดงภาพออกมาก็ได้

```
0A050101  00000000  7F023200  64006400  00000000  00FF00FF
0000FFFF  FF0000FF  00FFFFFF  00FFFFFF  00000000  00FF00FF
0000FFFF  FF0000FF  00FFFFFF  00FFFFFF  00045000  00000000
00000000  00000000  00000000  00000000  00000000  00000000
00000000  00000000  00000000  00000000  00000000  00000000
00000000  00000000  D2FF7FFF  FFFFFFFF  FFE0FFFF  00D100D2
FF7FFFFF  FFFFFFFF  E0FFFF00  D100D2FF  7FFFFFFF  FFFFFFE0
FFFF00D1  00D2FF7F  FFFFFFFF  FFFFE0FF  FF00D100  CFFFC1ED
C2FF7FFF  FFFFFFFF  FFE0FFFF  00D100C2  FFC1DDCF  FF7FFFFF
CCFFC1FD  C1D6FFFF  FFFFD2FF  FF00D100  C1FF8EC1  F76FCEFF
7FFFFF88  FFFFD0FF  B5BFC8FF  C1FDFFFF  C1FFFF00  D100C1FF
C1F7A0C1  EAADBFCC  FF7FFEFF  C1EFC1E5  3FFFFFCE  FFC1DDC1
C4C1E86F  FFFFCCFF  FF00D100  C1FF7E88  36125BC1  EFC5FFC1
F7ADC1DF  BDC2FF7F  FEFF8E94  C1C1158F  FFFFCCFF  C1FE536C
949FC7FF  C1FEC1FF  C1F7FFFF  C1FFFF00  D100C1FF......more.
```

รูปที่ 3.5 ตัวอย่าง PCX ไฟล์

ตัวอย่างข้อมูลในไฟล์ .PCX

ตัวอย่างไฟล์ .PCX บางส่วนแสดงดังรูปที่ 3.5 โดยแสดงออกมาเป็นเลขฐาน 16 เป็นเฮดเดอร์ 128 บิตเทียบดูได้จากรูปที่ 3.1 จะสังเกตว่าค่าของแต่ละไบต์ส่วนใหญ่จะมีค่ามากกว่า C0h แล้วจะตามด้วยค่าที่มากกว่า C0h อีกตัวหนึ่ง ซึ่งจะต่อกันเป็นคู่ ๆ แต่ถ้าค่าที่น้อยกว่า C0h จะพบว่าจะอยู่เดี่ยวๆ เพราะจะถูกเขียนลงไฟล์เป็นไบต์เดี่ยว ค่าที่มากกว่า C0h เมื่อถูก XOR ด้วย C0h แล้วจะมีค่าไม่เกิน 63 หรือ 3Fh

3.2 โครงสร้างของ Word Perfect Graphics (WPG) ไฟล์

ไฟล์แบบนี้จะเป็นแบบ Bitmap หรือ Vector Graphics ถ้าเป็น Bitmap จะเป็นชนิด 1 บิต,4 บิตหรือ 8 บิต บาง Version ของ Word Perfect จะแสดงได้แค่ 1 บิต และ 4 บิต ไฟล์ชนิดนี้แบ่งเป็น 4 ส่วนโดยแต่ละส่วนนั้นมีรายละเอียดซึ่งจะเสนอดังต่อไปนี้

3.2.1 โครงสร้างของส่วน WPGHEAD

สามารถกำหนดเป็นตัวแปรโครงสร้างด้วย Turbo C ดังนี้

```
typedef  struct {
                  char id[4]
                  long start;
                  char product;
                  char filetype;
                  char majorversion;
                  char minorversion;
                  unsigned int encrypt;
                  unsigned int reserved;
                } WPGHEAD
```

id จะเก็บข้อความ "\337WPC"

start เก็บตำแหน่งไบต์เริ่มต้นของ STARTRECORD ปกติมีค่าเท่ากับ 16L

product จะเป็น 1

filetype จะเป็น 16H

majorversion จะเป็น 1

minorversion จะเป็น 0

encrypt จะเป็น 0

reserved จะถูก set เป็น 0

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	4	id	เก็บข้อความ "\337WPC"
4	4	start	เก็บตำแหน่งของไบต์เริ่มต้นของ STARTRECORD มีค่าเท่ากับ 16L
8	1	product	มีค่าเท่ากับ 1
9	1	filetype	มีค่าเท่ากับ 16H
10	1	majorversion	มีค่าเท่ากับ 1
11	1	minorversion	มีค่าเท่ากับ 0
12	2	encrypt	มีค่าเท่ากับ 0
14	2	reserved	เซ็ตให้เท่ากับ 0

รูปที่ 3.6 ตารางโครงสร้าง WPGHEAD

3.2.2 โครงสร้างของส่วน Start WPG Data record

สามารถกำหนดได้ในรูปของ Turbo C ดังนี้

```
typedef struct {
                    char version;
                    char flags;
                    int screen width;
                    int screen depth;
                  } STARTRECORD
```

version จะเป็น 1

flags ถ้า AND กับ 01H แล้วเป็นจริงจะมี Postscript เพิ่มต่อจาก WPG ไฟล์เวลาอ่านข้อมูล

screen width ความละเอียดของจอภาพทางแนวนอนเป็น Pixels

screen depth ความละเอียดของจอภาพทางแนวตั้งเป็น Pixels

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	1	version	มีค่าเท่ากับ 1
1	1	flags	ถ้า AND กับ 01H แล้วเป็นจริงจะมี Postcript เพิ่มต่อจาก WPG ไฟล์เวลาอ่านข้อมูล
2	2	screenwidth	ความละเอียดของจอภาพทางแนวนอนเป็น Pixels
4	2	screendepth	ความละเอียดของจอภาพทางแนวตั้งเป็น Pixels

รูปที่ 3.7 ตารางโครงสร้าง STARTRECORD

3.2.3 โครงสร้างของส่วน Color Map record

สามารถกำหนดได้ในรูปแบบของ Turbo C ดังนี้

```
typedef struct {
                int startindex;
                int palettesize;
              } COLORMAP
```

startindex จะเป็นตัวชี้ตำแหน่งสีเริ่มต้นของข้อมูลภาพ

palettesize จะเก็บจำนวนสีที่มีอยู่ใน color map

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	2	start index	ชี้ตำแหน่งของสีเริ่มต้นของข้อมูลภาพ
2	2	palettesize	เก็บจำนวนของสีที่มีอยู่ใน Color map

รูปที่ 3.8 ตารางโครงสร้าง COLORMAP

3.2.4 โครงสร้างของส่วน Bitmap record

สามารถกำหนดได้ในรูปแบบของ Turbo C ดังนี้

```
typedef struct {
                int width;
                int depth;
                int bits;
                int xresolution;
                int yresolution;
               } BITMAP
```

width เก็บขนาดความละเอียดทางแนวนอนเป็น Pixels

depth เก็บขนาดความละเอียดทางแนวตั้งเป็น Pixels

bits เก็บจำนวนบิต ที่ใช้ในการแสดง 1 pixel

xresolution เก็บขนาดของความละเอียดตามแนวนอนต่อ 1 นิ้ว

yresolution เก็บขนาดของความละเอียดตามแนวตั้งต่อ 1 นิ้ว

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	2	width	เก็บขนาดความละเอียดทางแนวนอน เป็น Pixels
2	2	depth	เก็บขนาดความละเอียดทางแนวตั้ง เป็น Pixels
4	2	bits	เก็บจำนวนบิตที่ใช้ในการแสดง 1 Pixels
6	2	xresolution	เก็บขนาดความละเอียดทางแนวนอน เป็น Pixels ต่อ 1 นิ้ว
8	2	yresolution	เก็บขนาดความละเอียดทางแนวตั้ง เป็น Pixels ต่อ 1 นิ้ว

รูปที่ 3.9 ตารางโครงสร้าง BITMAP

3.3 โครงสร้างของ IFF/ILBM ไฟล์

ไฟล์ชนิดนี้เป็นแบบ Bitmap กราฟิก โครงสร้างของ Header ไฟล์ชนิดนี้ค่อนข้างจะมีขนาดเล็ก โครงสร้างของ Header ไฟล์ชนิดนี้สามารถกำหนดเป็นตัวแปรโครงสร้างด้วย Turbo C ได้ดังนี้

```
typedef struct {
                    char type[4];
                    unsigned long size;
                    char subtype[4];
                  } IFFHEADER
```

type จะเก็บข้อความ " FORM","LIST" หรือ "CAT" โดยปกติจะเก็บข้อความ "FORM" เป็นส่วนใหญ่

size จะเก็บขนาดของไฟล์เป็นไบต์

subtype ถ้าเป็น IFF ไฟล์จริงๆจะเก็บข้อความ "ILBM" แต่ถ้าเป็น Deluxe Paint LBM ไฟล์จะเก็บข้อความ "PBM"

ส่วนสำคัญของ IFF ไฟล์ มีอยู่ 3 ส่วนด้วยกัน

- BMHD CHUNK จะเป็นส่วน dimension ของไฟล์
- CMAP CHUNK จะเป็นส่วน Color map ของไฟล์
- BODY CHUNK จะเป็นส่วนข้อมูลของไฟล์

BMHD CHUNK

ส่วนนี้จะมีความสำคัญในการถอดรหัสข้อมูลในส่วนของ BODY CHUNK เพราะว่าส่วนนี้จะเก็บรายละเอียดต่าง ๆ เกี่ยวกับไฟล์ โครงสร้างส่วนนี้สามารถกำหนดเป็นตัวแปรโครงสร้างด้วย Turbo C ได้ดังนี้

```
typedef struct {
                    char id[4];
                    unsigned long size;
                    unsigned int w,h;
                    int x,y;
                    char nplanes;
                    char masking;
                    char compression;
                    char padl
                    unsigned int transparent color;
                    char xAspect,yAspect;
```

```
    int pageW,pageH;
} BMHD;
```

id เก็บข้อความ "BMHD"

size เก็บขนาดของ CHUNK DATA มีค่าเท่ากับ 20 ไบต์

w,h จะเป็นความละเอียดทางแนวนอนและแนวตั้งมีหน่วยเป็น Pixels

x,y จะเป็นจุด Coordinates ที่มุมบนซ้ายของภาพ

nPlanes จะเป็นจำนวนเพลนที่ต้องการเพื่อใช้ในการแสดงภาพ

masking จะเป็น Masking type

Compression จะเป็นชนิดของการเก็บข้อมูลภาพ ถ้าเป็น 0 ส่วนของ BODY CHUNK จะเป็นข้อมูลของภาพจริงๆ ที่ยังไม่ได้เข้ารหัส แต่ถ้าเป็น 1 ส่วนของ BODY CHUNK จะเป็นข้อมูลของภาพที่เข้ารหัสแล้ว

padl จะเป็น 0

transparent Color จะกำหนดตำแหน่งของ Color Index และจะใช้ควบคู่กับ masking ฟิลด์

xAspect,yAspect อัตราส่วน Pixel ระหว่างภาพกับจอ ในแนวนอนและแนวตั้ง

pageW,pageH ขนาดความกว้างและความสูงของจอภาพเป็น Pixel

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	4	id	เก็บข้อความ "BMHD"
4	4	size	ขนาดของ CHUNK data มีค่าเท่ากับ 20 ไบต์
8	2	w	เก็บความละเอียดของภาพทางแนวนอน เป็น Pixels

รูปที่ 3.10 ตารางโครงสร้าง BMHD bitmap header

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
10	2	h	เก็บความละเอียดของภาพทางแนวตั้ง เป็น Pixels
12	2	x	เก็บตำแหน่งเริ่มต้นตามแนวนอนที่มุมบนด้านซ้าย
14	2	y	เก็บตำแหน่งเริ่มต้นตามแนวตั้งที่มุมบนด้านซ้าย
16	1	nPlanes	จำนวนเพลนที่ต้องการเพื่อใช้ในการแสดงภาพ
17	1	masking	เก็บ Masking type
18	1	compression	ถ้าเท่ากับ 0 ส่วน Body chunk เป็นข้อมูลที่ยังไม่ได้เข้ารหัส แต่ถ้าเท่ากับ 1 ส่วน Body chunk จะเป็นข้อมูลที่เข้ารหัสแล้ว
19	1	pad1	เท่ากับ 0
20	2	transparentcolor	กำหนดตำแหน่ง color index ของภาพ
22	1	xAspect	อัตราส่วนความละเอียดทางแนวนอนของภาพกับจอ
23	1	yAspect	อัตราส่วนความละเอียดทางแนวตั้งของภาพกับจอ
24	2	pageW	ความละเอียดทางแนวนอนของจอ
26	2	pageH	ความละเอียดทางแนวตั้งของจอ

รูปที่ 3.10 ตารางโครงสร้าง BMHD bitmap header (ต่อ)

CMAP CHUNK

จะเป็นพื้นที่ในหน่วยความจำที่ใช้เก็บค่าของสี RGB ในส่วนนี้จะมีขนาด 768 ไบต์ แสดงสีได้ 256 สีใน 1 สีจะใช้หน่วยความจำ 3 Bytes แต่ละไบต์จะบอกเปอร์เซ็นต์ของสีแต่ละสี ซึ่งได้แก่ สีแดง,สีเขียว,และสีน้ำเงิน

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	4	id	เก็บข้อความ "CMAP"
4	4	size	เก็บขนาดของ CHUNK data มีค่าเท่ากับ 3 เท่าของจำนวนสีที่ใช้
8	3*N	cmap	เก็บสีของภาพ

รูปที่ 3.11 ตารางโครงสร้าง CMAP CHUNK

BODY CHUNK จะเป็นส่วนข้อมูลของภาพจริง ที่ต้องการอ่านข้อมูลแล้วนำมาแสดง

3.4 โครงสร้างของ BMP ไฟล์

โครงสร้างของ header ไฟล์แบ่งเป็น 2 ส่วนคือ ส่วนของไฟล์ headerและ Image header โครงสร้างของ header ไฟล์ชนิดนี้สามารถกำหนดตัวแปรโครงสร้างด้วย Turbo C ได้ดังนี้

```
typedef struct  {
                    char id[2];
                    long filesize;
                    int  reserved1[2];
```

```
                int  reserved2[2];
                long headersize;
                long infosize;
                long width;
                long depth;
                int  biPlanes;
                int  bits;
                long biCompression;
                long bisizeImage;
                long biXPelsPerMeter;
                long biYPelsPerMeter;
                long biClrUsed;
                long biClrImportant;
            } BMPHEAD;
```

id จะเก็บข้อความ "BM"

filesize จะเก็บขนาดของไฟล์เป็นไบต์

reserved1 สำรองไว้จำนวน 2 ไบต์

reserved2 สำรองไว้จำนวน 2 ไบต์

headersize จำนวนไบต์ของ header ไฟล์

infosize จำนวนไบต์ของ infoheader

width ความละเอียดทางด้านแนวนอนของภาพเป็น Pixels

depth ความละเอียดทางด้านแนวตั้งของภาพเป็น Pixels

biPlanes จำนวนเพลนที่ใช้ในการแสดงภาพ ต้องเป็น 1 เสมอ

bits จำนวนบิตที่ใช้ในการแสดงภาพ 1 Pixel

biCompressions โดยมากจะมีค่าเป็น 0L ซึ่งหมายถึงไม่มีการบีบข้อมูลเวลาเข้ารหัส

bisizeImage ขนาดของภาพที่เข้ารหัสแล้วเป็นไบต์

biXPelsPerMeter ความละเอียดทางแนวนอนเป็น Pixels ต่อเมตร

biYPelsPerMeter ความละเอียดทางแนวตั้งเป็น Pixels ต่อเมตร

biClrUsed จำนวนสีที่ใช้อย่างต่ำ

biClrImportant จำนวนของสีที่สำคัญ

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	2	id	เก็บข้อความ "BM"
2	4	filesize	เก็บขนาดของไฟล์เป็นไบต์
6	2	reserved1	สำรอง
8	2	reserved2	สำรอง
10	4	headersize	จำนวนไบต์ของ header ไฟล์

รูปที่ 3.12 ตารางโครงสร้าง BITMAPFILEHEADER file header

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
14	4	infosize	เก็บจำนวนไบต์ของ infoheader
18	4	width	ความละเอียดตามแนวนอนของภาพ
22	4	depth	ความละเอียดตามแนวตั้งของภาพ
26	2	biPlanes	จำนวนเพลนที่ใช้แสดงภาพ
28	2	bits	จำนวนบิตที่ใช้ในการแสดงภาพ 1 Pixel

รูปที่ 3.13 ตารางโครงสร้าง BITMAPINFOHEADER image header

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
30	4	bicompression	มีค่าเท่ากับ 0L
34	4	bisizeimage	ขนาดของภาพที่เข้ารหัสแล้วมีหน่วยเป็นไบต์
38	4	biXPelsPerMeter	ความละเอียดตามแนวนอนของภาพเป็น Pixels ต่อ 1 เมตร
42	4	biYPelsPerMeter	ความละเอียดตามแนวตั้งของภาพเป็น Pixels ต่อ 1 เมตร
46	4	biClrUsed	จำนวนของสีที่ใช้อย่างต่ำ
50	4	biClrImportant	จำนวนของสีสำคัญที่ใช้
54	4*N	bmiColors	Color map

รูปที่ 3.13 ตารางโครงสร้าง BITMAPINFOHEADER image header (ต่อ)

3.5 โครงสร้างของ TARGA (TGA) ไฟล์

โดยทั่วๆไป TGA ไฟล์ประกอบด้วยส่วนต่างๆคือ ส่วนของ HEADER ส่วนของ Palette และส่วนที่เก็บข้อมูลภาพจริงๆ โครงสร้างของไฟล์ชนิดนี้สามารถกำหนดเป็นตัวแปรโครงสร้างด้วย Turbo C ได้ดังนี้

```
typedef struct {
                char identsize;
                char colormaptype;
                char imagetype;
                unsigned int colormapstart;
```

```
        unsigned int colormaplength;
        char colormapbits;
        unsigned int xstart;
        unsigned int ystart;
        unsigned int width;
        unsigned int depth;
        char bits;
        char descriptor;
    }   TGAHEAD;
```

identsize จะเก็บขนาดของไฟล์ header เป็นไบต์

colormaptype จะกำหนด color map ที่ใช้ในไฟล์ ถ้าค่าของ colormaptype เป็น 1 แสดงว่าไฟล์นี้ใช้ color map แต่ถ้าค่าเป็น 0 แสดงว่าไฟล์ไม่ได้ใช้ color map

imagetype จะกำหนดชนิดและการเข้ารหัสข้อมูลของภาพ

ถ้าเป็น 1 แสดงว่าเป็น Uncompressed palette-driven image

ถ้าเป็น 2 แสดงว่าเป็น Uncompressed RGB image

ถ้าเป็น 3 แสดงว่าเป็น Uncompressed monchrome image

ถ้าเป็น 9 แสดงว่าเป็น Run-length encoded palette driver image

ถ้าเป็น 10 แสดงว่าเป็น Run-length encoded RGB image

ถ้าเป็น 11 แสดงว่าเป็น Run-length monochrome image

colormapstart จะชี้ตำแหน่งของสีเริ่มต้น

colormaplength จะเก็บจำนวนของสีทั้งหมดที่ใช้ในภาพ

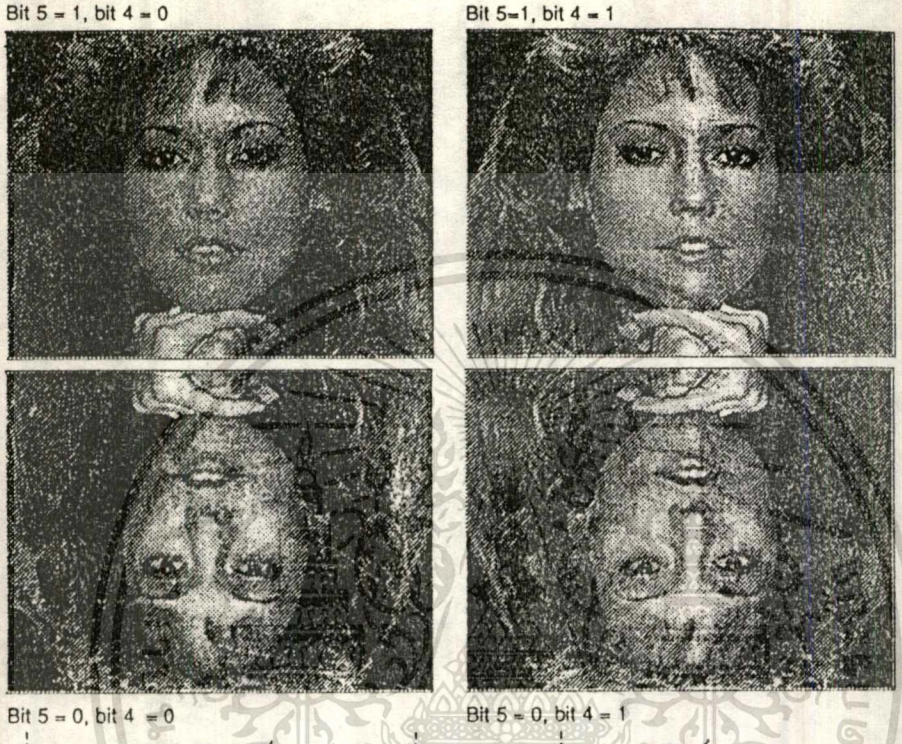
colormapbits จำนวนบิตที่ใช้ในการแสดงสี 1 สี

xstart,ystart จะเป็นตำแหน่งที่มุมบนด้านซ้ายของจอ

width,depth จะแสดงความละเอียดตามแนวนอนและแนวตั้งเป็น Pixels

bits แสดงจำนวนบิตที่ใช้ในการแสดงภาพ 1 Pixel

descriptor จะเป็นตัวกำหนดให้ภาพแสดงในลักษณะอย่างไร ซึ่งในฟิลด์นี้จะสนใจบิตที่ 5 และบิตที่ 4 เท่านั้น



รูปที่ 3.14 ความสัมพันธ์ระหว่างบิตที่ 4 และบิตที่ 5 ของฟิลด์ descriptor กับลักษณะของการแสดงภาพ

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
0	1	identsize	เก็บขนาดของไฟล์ header เป็นไบต์
1	1	colormaptype	เท่ากับ 1 แสดงว่าใช้ color map เท่ากับ 0 แสดงว่าไม่ใช้ color map
2	1	imagetype	ชนิดและการเข้ารหัสข้อมูลของภาพ
3	2	colormapstart	ชี้ตำแหน่งของสีเริ่มต้น

รูปที่ 3.15 ตารางโครงสร้าง TGAHEADER

ไบต์ที่	ขนาด	ชื่อข้อมูล	รายละเอียด
5	2	colormaplength	เก็บจำนวนของสีทั้งหมดที่ใช้ในภาพ
7	1	colormapbits	เก็บจำนวนของบิตที่ใช้ในการแสดงสี 1 สี
8	2	xstart	ตำแหน่งทางแนวนอนที่มุมบนด้านซ้าย
10	2	ystart	ตำแหน่งทางแนวตั้งที่มุมบนด้านซ้าย
12	2	width	ความละเอียดทางแนวนอนของภาพ
14	2	depth	ความละเอียดทางแนวตั้งของภาพ
16	2	bits	จำนวนบิตที่ใช้ในการแสดง 1 Pixel
18	1	descriptor	กำหนดลักษณะการแสดงของภาพ

รูปที่ 3.15 ตารางโครงสร้าง TGAHEADER (ต่อ)

การเก็บข้อมูลลงในไฟล์ .POT

จะขอยกตัวอย่างในการเก็บข้อมูลในไฟล์ .POT มาประกอบเพื่อจะช่วยให้เข้าใจในเรื่องการเก็บข้อมูลมากยิ่งขึ้น สมมุติว่าในไฟล์ TEST.POT มีข้อมูลเก็บไว้ดังนี้

‹ TEST.POT ›

FILES	FORMAT_IN	TIMES	FORMAT_OUT
tong.pcx	light_dot_in	10	light_dot_out
.....	.....	..	.....
.....	.....	..	.....

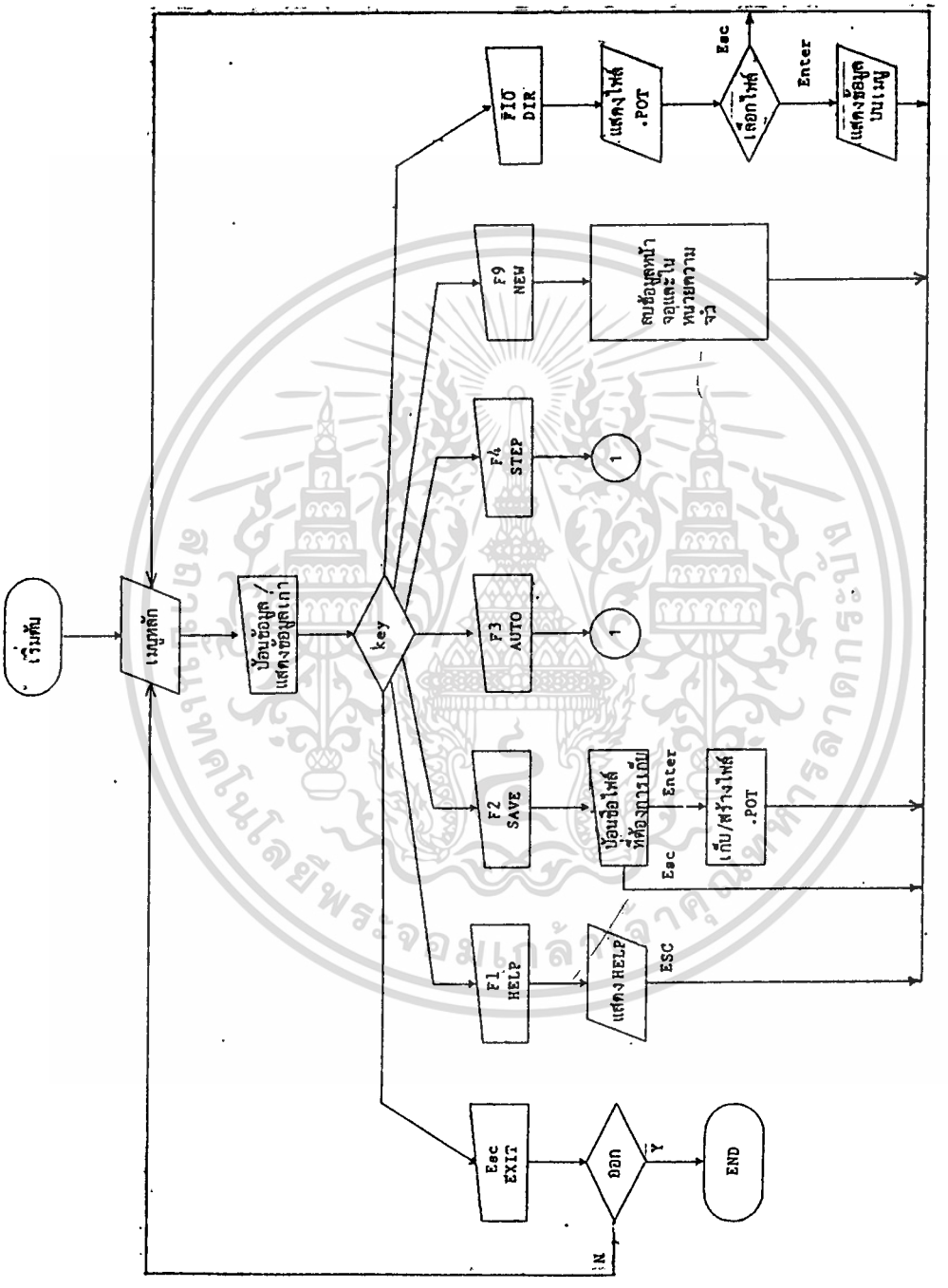
ลักษณะการเก็บข้อมูลเมื่อแปลงเป็นแอสกี้โค้ด จะได้ดังนี้

74 6F 6E 67 2E 70 63 78 20 20 20 20 20 20 20 20 20 20 20 20 20
20 6C 69 67 68 74 5F 64 6F 74 5F 69 6E 20 20 20 20 20 20 20 20
31 30 20 20 20 20 20 20 20 20 20 6C 69 67 68 74 5F 64 6F 74 5F 6F
75 74 00

ทุกครั้งที่มีการจัดเก็บข้อมูล (SAVE) เราจะกำหนดให้ตรงตำแหน่งสุดท้ายของข้อมูลเป็น 0 เสมอ และเมื่อต้องการเรียกข้อมูลที่เก็บอยู่ในไฟล์ .POT มาแสดง เราก็ทำการเปิดไฟล์.POT จากนั้นให้อ่านข้อมูลที่เก็บอยู่ในไฟล์.POT นั้นมาครั้งละ 1 ไบต์(1 ตัวอักษร) และทำการตรวจสอบว่าข้อมูลนั้นๆ ว่ามีค่าเป็น 0 หรือไม่ ถ้าไม่เป็น 0 ให้กลับไปอ่านข้อมูลถัดไป ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะตรวจสอบพบข้อมูลที่เป็น 0 ก็ให้ออกจากการวนลูป แล้วนำข้อมูลมาแสดงบนหน้าจอ ในกรณีที่มีการป้อนข้อมูลมากๆ ข้อมูลเหล่านี้ก็จะจัดเรียงต่อกันเป็นลำดับ

บทที่ 4

ผังงานของการทำงานในส่วนต่างๆ

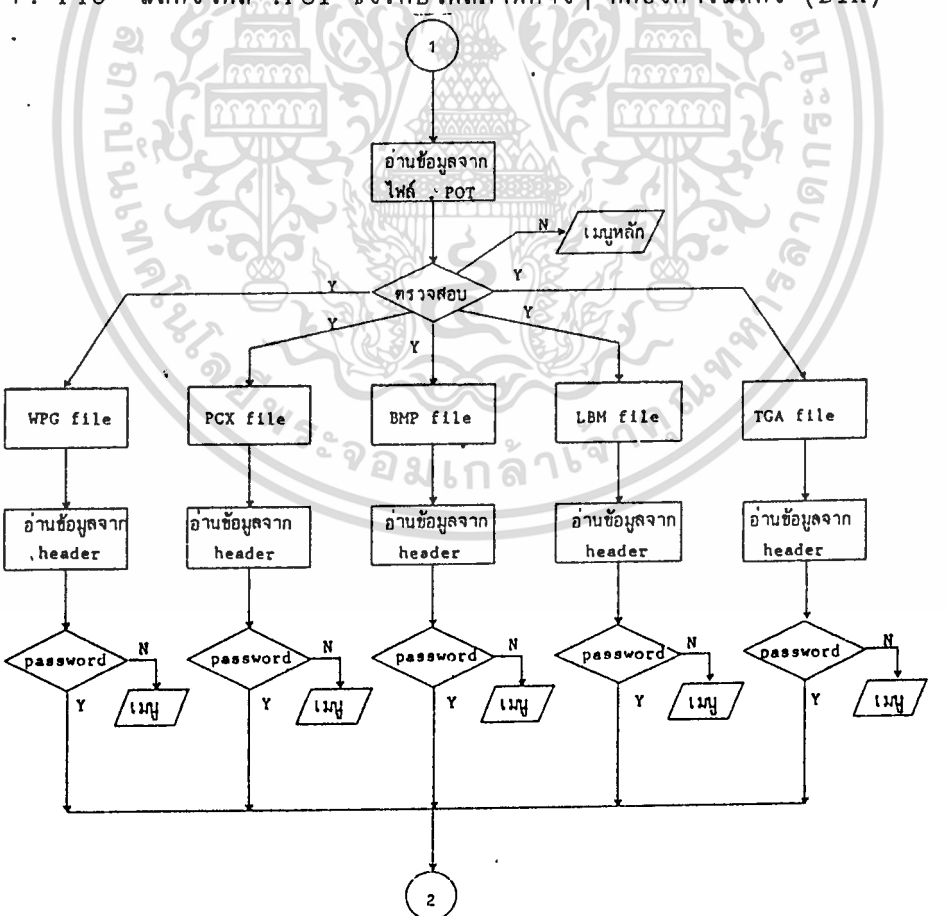


รูปที่ 4.1 ผังงานการทำงานในส่วนของเมนูหลัก

คำอธิบายผังงานรูปที่ 4.1

เมื่อโปรแกรมนี้ถูกเรียกใช้ โปรแกรมจะแสดงเมนูหลัก หลังจากนั้นผู้ใช้ก็ต้องป้อนชื่อไฟล์ภาพที่ต้องการแสดงหรือเรียกไฟล์เก่าที่มีไฟล์ภาพอยู่แล้วออกมา ชื่อไฟล์ที่เรียกออกมาก็จะแสดงให้เห็นบนหน้าจอ ต่อไปก็คือ โปรแกรมจะรอรับ key โดยจะมี key ไว้สำหรับเลือกใช้ดังต่อไปนี้

1. Esc ออกจากโปรแกรม (QUIT)
2. F1 แสดงวิธีการใช้งานโปรแกรม (HELP)
3. F2 การเก็บและสร้างชื่อไฟล์ .POT (SAVE)
4. F3 การแสดงไฟล์ภาพที่อยู่ในไฟล์ .POT อย่างต่อเนื่อง (AUTO)
5. F4 การแสดงไฟล์ภาพแบบทีละไฟล์ (STEP)
6. F9 ลบชื่อไฟล์บนจอเมนูและลบข้อมูลใน buffer ด้วย(NEW)
7. F10 แสดงไฟล์ .POT ซึ่งเก็บไฟล์ภาพต่างๆ ที่ต้องการแสดง (DIR)



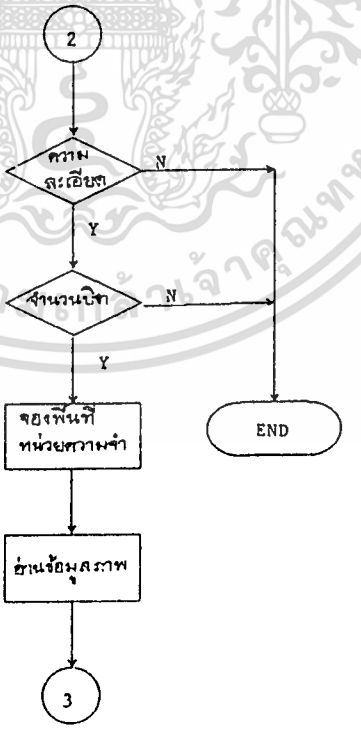
รูปที่ 4.2 ผังงานการตรวจสอบไฟล์และ password

คำอธิบายผังงานรูปที่ 4.2

อ่านไฟล์ภาพไฟล์แรกจากไฟล์ .POI ว่าเป็นนามสกุลใด ถ้าตรงกับนามสกุลไหน ก็ให้โปรแกรมไปทำงานยังส่วนของนามสกุลนั้น ถ้าไม่ตรงกับนามสกุลไหนเลย ก็จะกลับไปยังเมนูหลักเหมือนเดิม แต่ถ้าตรงนามสกุลที่มีอยู่โปรแกรมก็จะเช็ค password ของนามสกุลนั้นว่าถูกต้องหรือไม่ โดยที่ความสัมพันธ์ระหว่างแต่ละนามสกุลกับ password เป็นดังนี้

นามสกุล	password
.WPG	4 ไบต์แรกเก็บข้อความ "\337WPC"
.PCX	ไบต์แรกเก็บข้อความ "0A"
.LBM	4 ไบต์แรกเก็บข้อความ "FORM","LIST" หรือ "CAT"
.BMP	2 ไบต์แรกเก็บข้อความ "BM"
.TGA	ไบต์ที่ 2 เก็บข้อความ "01","02","03","09","0A" หรือ "0B"

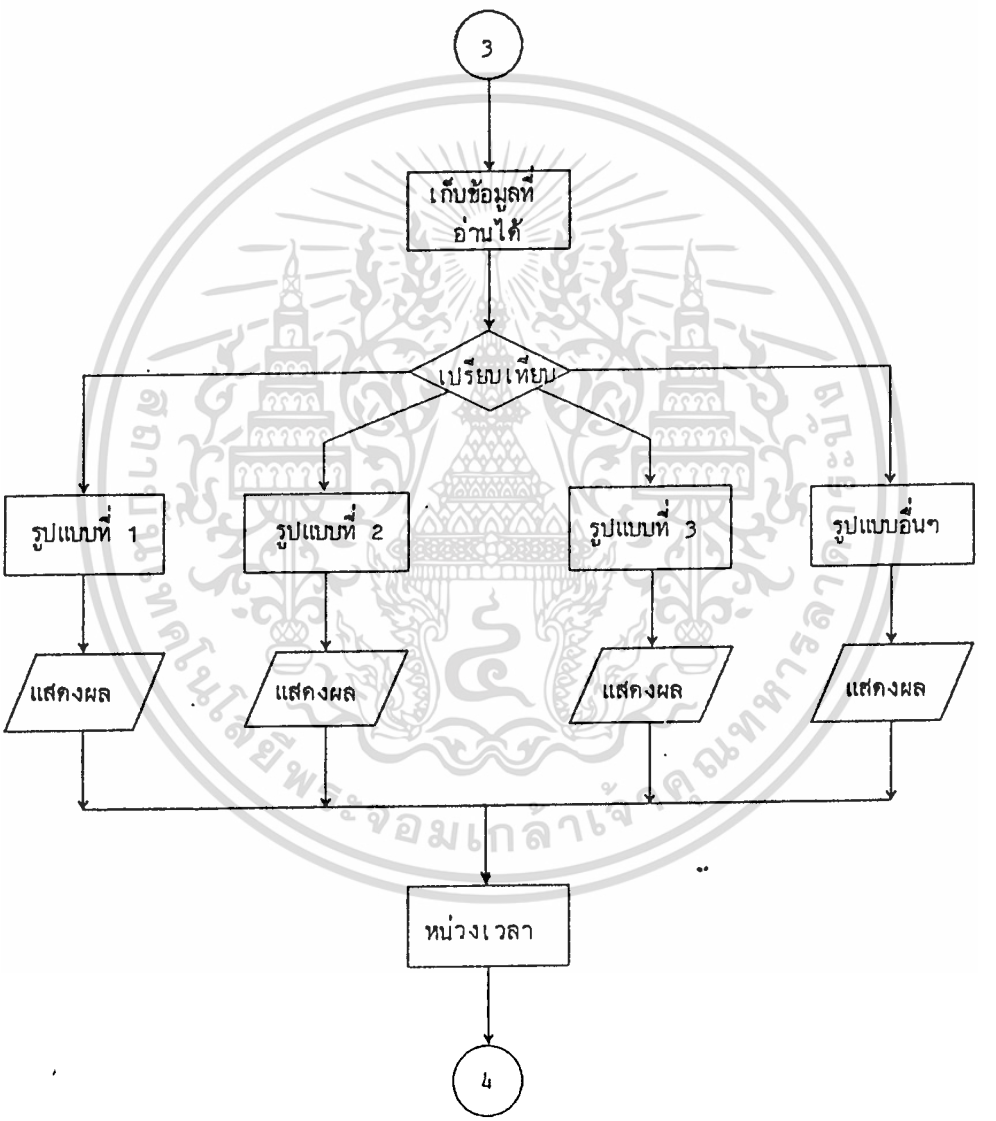
ถ้า password ไม่ถูกต้อง โปรแกรมจะกลับสู่เมนูหลักเหมือนเดิม



รูปที่ 4.3 ผังงานการตรวจสอบความละเอียดและจำนวนบิต

<u>คำอธิบายผังงานรูปที่ 4.3</u>

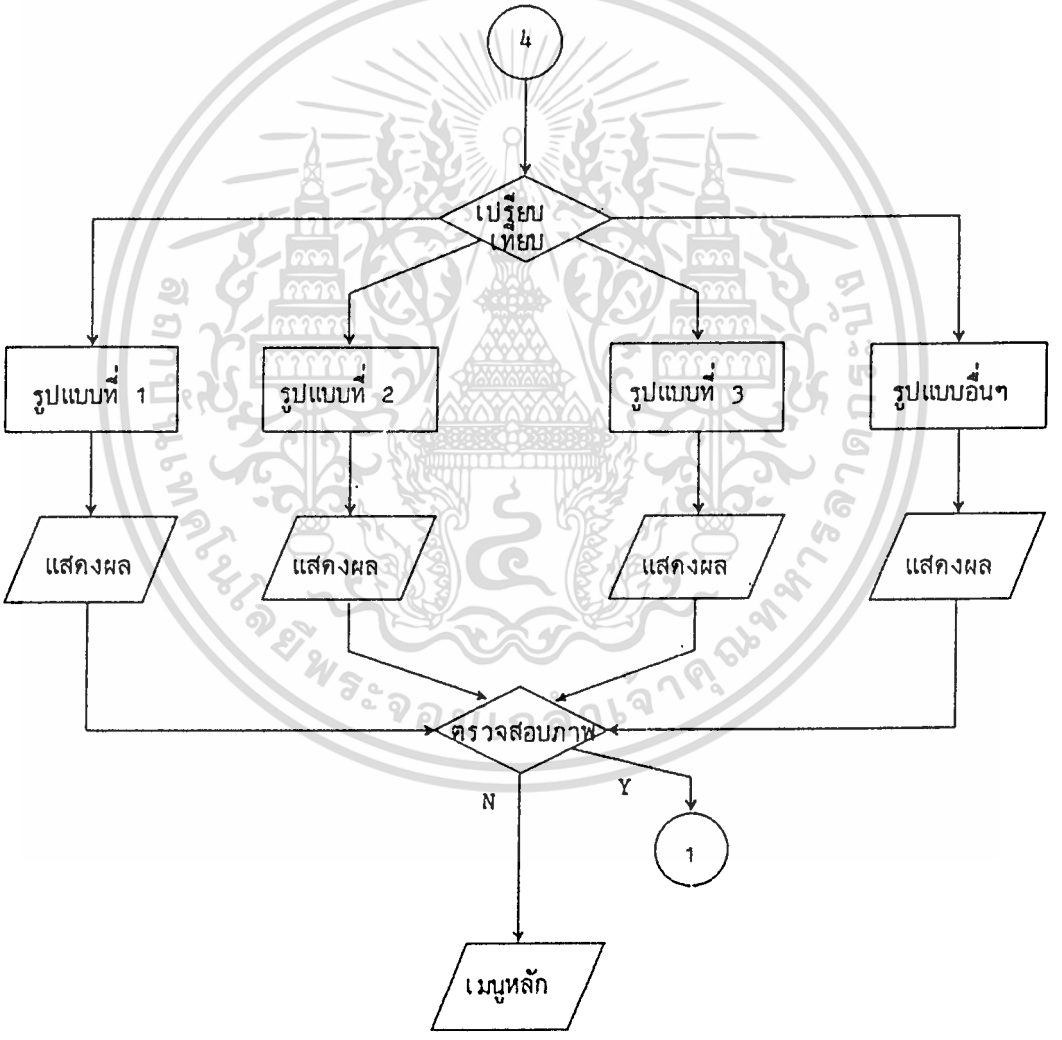
เช็คความละเอียดของภาพแต่ละไฟล์ภาพว่าเท่ากับ 320*200 จุดหรือไม่ ถ้าไม่ใช่ให้ออกจากโปรแกรมถ้าใช่ ก็ให้เช็คจำนวนบิตที่ใช้ในแสดงภาพ 1 Pixel ว่าเท่ากับ 8 บิต์หรือไม่ ถ้าไม่ใช่ให้ออกจากโปรแกรม ถ้าใช่ก็จะให้จองพื้นที่หน่วยความจำ (buffer) และอ่านข้อมูลไฟล์ภาพ



รูปที่ 4.4 ผังงานของรูปแบบการแสดงผล (format in)

คำอธิบายผังงานรูปที่ 4.4

เก็บข้อมูลที่อ่านได้ลงในหน่วยความจำ (buffer) เปรียบเทียบคำสั่งของรูปแบบการแสดงภาพ (format in) ว่าตรงกับรูปแบบการแสดงภาพแบบใด โปรแกรมก็จะให้ไปทำงาน ในส่วนของการแสดงภาพในรูปแบบนั้น ในการแสดงผลเป็นการนำข้อมูลในหน่วยความจำ (buffer) ไปยังหน่วยความจำแสดงผล (video memory) ซึ่งวิธีการนำเสนอข้อมูลจะมีลักษณะที่ต่างกันไปขึ้นอยู่กับรูปแบบการแสดงนั้นๆ



รูปที่ 4.5 ผังงานของรูปแบบการหายไปของภาพ (format out)

คำอธิบายผังงานรูปที่ 4.5

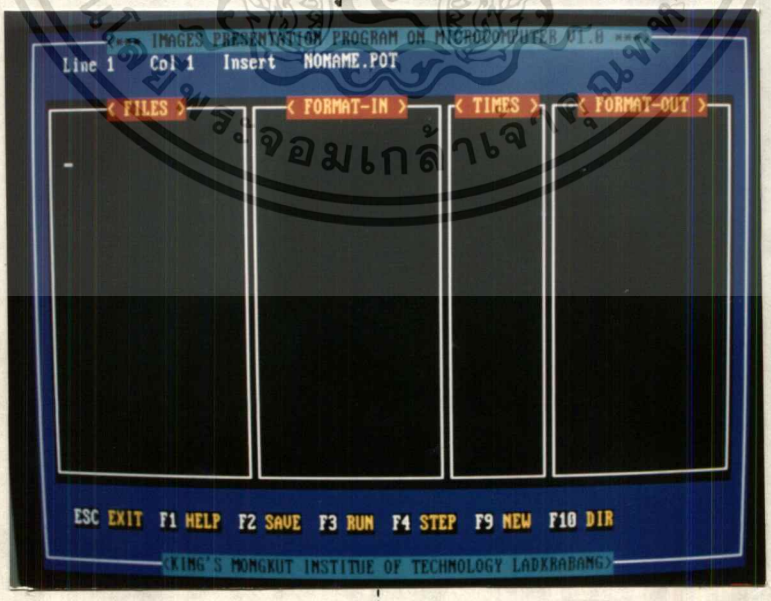
เปรียบเทียบคำสั่งของรูปแบบการหายไปของภาพ (format out) ว่าตรงกับแบบการหายไปแบบใด โปรแกรมก็จะให้ไปทำงานในส่วนของการหายไปแบบนั้น หลังจากนั้นตรวจสอบดูว่ายังมีไฟล์ภาพต่อไปที่ต้องการแสดงหรือไม่ ถ้ามีก็โปรแกรมก็จะกลับไปอ่านไฟล์ภาพนั้นจากไฟล์ .POI ใหม่อีกครั้ง ถ้าไม่มีไฟล์ภาพที่ต้องการแสดงอีกโปรแกรมก็จะกลับไปยังเมนูหลัก

บทที่ 5

วิธีการใช้งานของโปรแกรม

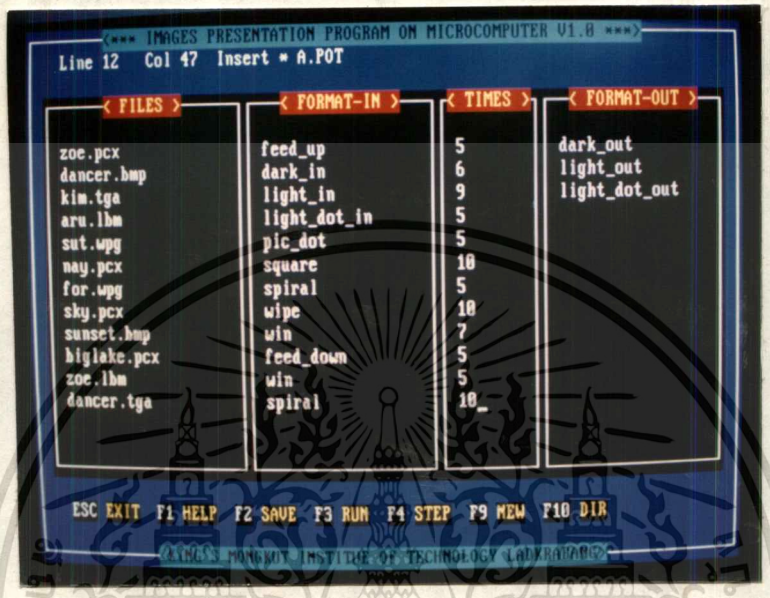
เมื่อโปรแกรมนี้ถูกเรียกใช้จะปรากฏภาพที่หน้าจอดังรูปที่ 5.1 พร้อมด้วยเสียงเพลง หลังจากนั้นให้กด key ใดๆ เพื่อเข้าสู่เมนูหลัก เมื่อเรากด key ใดๆแล้วที่บนหน้าจอของคอมพิวเตอร์จะแสดงเมนูหลักดังรูปที่ 5.2

รูปที่ 5.1



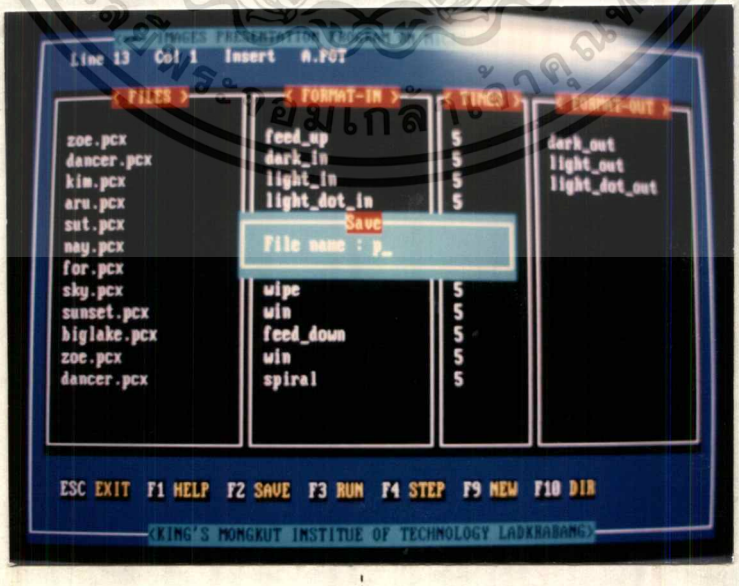
รูปที่ 5.2

ต่อไปก็คือถ้าต้องการแสดงภาพก็ให้ป้อนชื่อไฟล์ภาพพร้อมด้วยนามสกุล รูปแบบการแสดง (format in) ระยะเวลาที่ต้องการหน่วง (delay time) โดยจะมีหน่วยเป็นวินาที และสุดท้ายก็คือ การป้อนรูปแบบการหายไป (format out) ของภาพ ดังรูปที่ 5.3



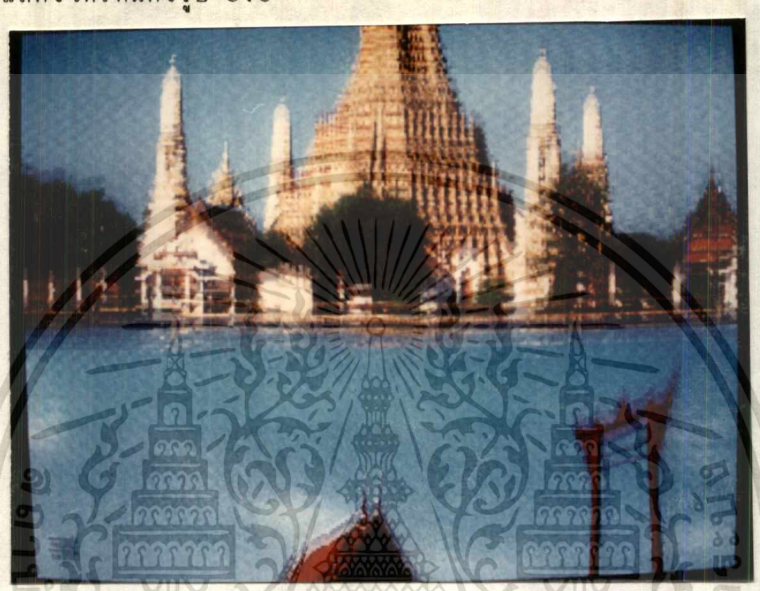
รูปที่ 5.3

เมื่อต้องการแสดงภาพตามลำดับที่ป้อนก่อนจะทำการรันโปรแกรมจะต้องกด key F2 ก่อนเสมอ ในขณะนี้หน้าจอของคอมพิวเตอร์จะปรากฏดังรูป 5.4 ซึ่งเป็นการ SAVE ไฟล์ภาพที่ป้อนทั้งหมดให้ไปเก็บไว้ในไฟล์ .POI โดยผู้ใช้เป็นผู้ตั้งชื่อไฟล์เอง



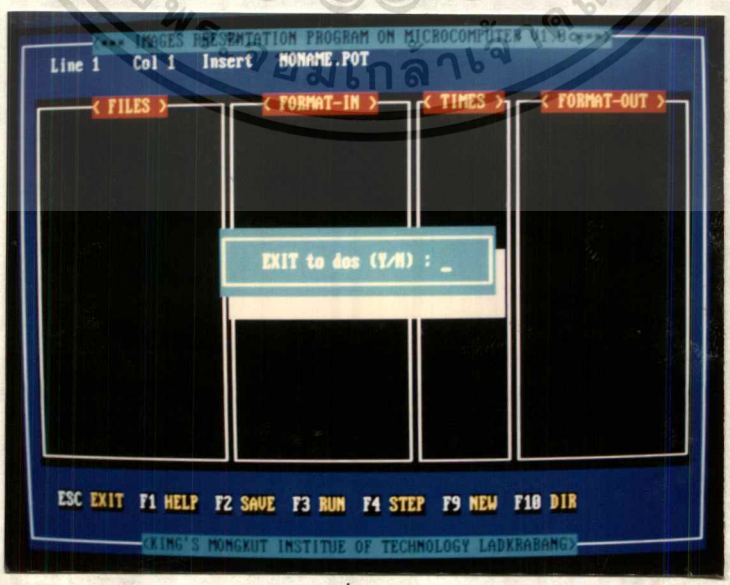
รูปที่ 5.4

หลังจากนั้นก็ทำการเลือกว่าต้องการแสดงภาพแบบอัตโนมัติ (AUTO) ก็ให้กด F3 ภาพจะแสดงตามลำดับแบบต่อเนื่อง โดยจะมีการหน่วงเวลาเข้ามาเกี่ยวข้องเมื่อจะแสดงภาพถัดไป หรือต้องการแสดงภาพทีละภาพ (STEP) ให้กด F4 ภาพจะแสดงออกมาและยังแสดงภาพนั้นอยู่จนกว่าจะกด key ใดๆ ภาพก็จะหายไปแล้วแสดงภาพถัดไปออกมา ลักษณะการแสดงภาพแสดงให้เห็นดังรูป 5.5



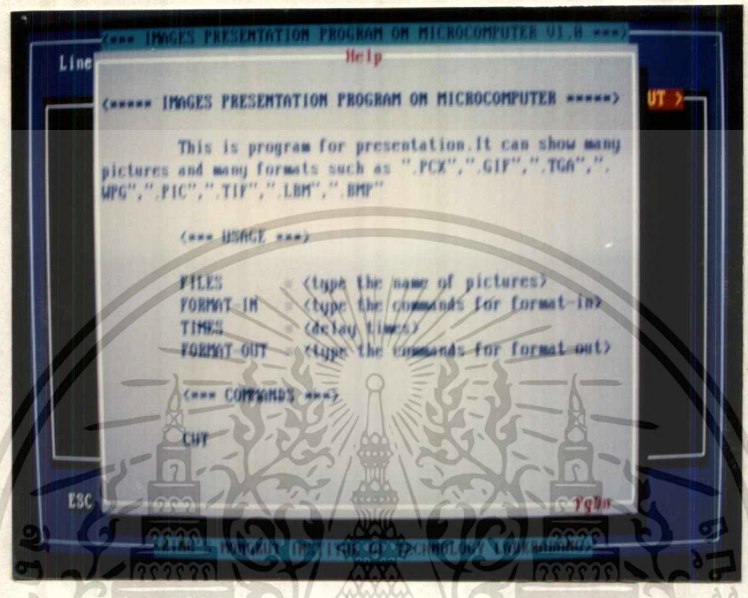
รูปที่ 5.5

เมื่อต้องการออกจากโปรแกรมให้กด key Esc โดยจะปรากฏภาพบนหน้าจอดังรูป 5.6 ถ้าต้องการออกจากโปรแกรมให้กดอักษร Y หรือถ้าไม่ต้องการออกจากโปรแกรมให้กดอักษร N หรือ key ใดๆ



รูปที่ 5.6

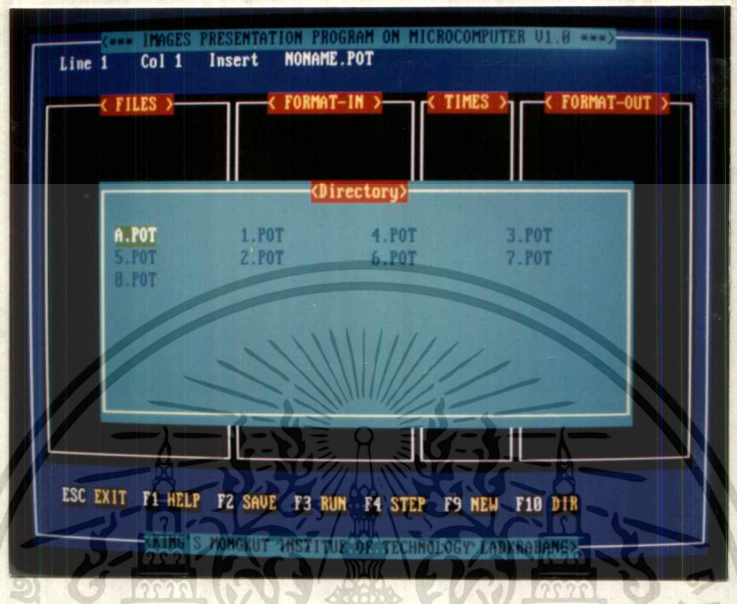
การกด key F1 บนหน้าจอเมนูหลักจะเป็นการอธิบายเกี่ยวกับการใช้คำสั่งในการแสดงภาพแบบต่างๆ และขีดจำกัดความสามารถของโปรแกรม ดังรูป 5.7



รูปที่ 5.7

การกด key F9 เป็นการลบข้อมูลทั้งหมดที่พิมพ์อยู่บนหน้าจอของเมนูหลักและข้อมูลทั้งหมดที่อยู่ใน buffer และเซตค่าต่างๆให้เป็นค่าเริ่มต้นทั้งหมด

การกด key F10 เป็นการแสดงไฟล์ .POT ดังรูป 5.8 ซึ่งไฟล์ .POT นี้จะเก็บไฟล์ภาพต่างๆไว้ ถ้าต้องการนำไฟล์ภาพชุดต่างๆมาแสดงอีกครั้งหนึ่งก็ให้เลื่อนแถบสีไปยังไฟล์ .POT ที่เก็บไฟล์ภาพชุดเหล่านั้นไว้ออกมาแสดงบนเมนูหลัก ในขณะนี้เราสามารถรันโปรแกรมได้เลย หรือทำการแก้ไขและเพิ่มเติมไฟล์ภาพ เมื่อมีการแก้ไขหรือเพิ่มเติมไฟล์ภาพก่อนจะทำการรันโปรแกรมต้อง SAVE ก่อนเสมอไม่เช่นนั้นโปรแกรมจะแสดงผลตามไฟล์ภาพชุดเดิมก่อนทำการแก้ไข แต่ถ้าไม่มีการแก้ไขหรือเพิ่มเติมไฟล์ภาพก็สามารถรันโปรแกรมได้เลย



รูปที่ 5.8

ส่วนบนของเมนูหลัก

Line 1 แสดงบรรทัดที่เคอร์เซอร์อยู่ ในที่นี้อยู่บรรทัดที่ 1

Col 1 แสดงคอลัมน์ที่เคอร์เซอร์อยู่ ในที่นี้อยู่คอลัมน์ที่ 1

Insert เมื่อมีข้อความนี้อยู่ในสภาวะ Insert on การพิมพ์ในสภาวะนี้จะเป็นการพิมพ์แทรกตัวอักษรที่ตำแหน่งเคอร์เซอร์อยู่ โดยตัวอักษรที่ถูกแทรกจะเลื่อนไปทางขวาของเคอร์เซอร์ เมื่อกด key Ins ข้อความ Insert จะหายไป สภาวะนี้เป็นสภาวะ Insert off การพิมพ์ในสภาวะนี้จะเป็นการพิมพ์ทับตัวอักษรที่ตำแหน่งเคอร์เซอร์อยู่เมื่อกด key Ins อีกครั้งจะเป็นสภาวะ Insert on อีกครั้งหนึ่ง

NONAME.POT เป็นชื่อไฟล์ที่อยู่ในหน่วยความจำ เมื่อเข้าสู่เมนูหลักโดยไม่ได้โหลดมาจากไฟล์ .POT เดิม โปรแกรมจะตั้งชื่อนี้ให้ชั่วคราว

key ต่างๆ ที่ควรทราบ

	เลื่อนเคอร์เซอร์ไปทางขวา 1 คอลัมน์
	เลื่อนเคอร์เซอร์ไปทางซ้าย 1 คอลัมน์
	เลื่อนเคอร์เซอร์ขึ้นไปบรรทัดบน 1 บรรทัด
	เลื่อนเคอร์เซอร์ลงไปบรรทัดล่าง 1 บรรทัด
PgUp	key นี้จะใช้เฉพาะเมื่อมีการเรียกใช้ HELP ซึ่งเป็นการเรียกข้อความที่อธิบายอยู่ในหน้าก่อนมาแสดง
PgDn	key นี้จะใช้เฉพาะเมื่อมีการเรียกใช้ HELP ซึ่งเป็นการเรียกข้อความที่อธิบายอยู่ในหน้าถัดไปมาแสดง
Backspace	ลบตัวอักษรที่อยู่ทางซ้ายของเคอร์เซอร์

คำสั่งต่างๆ ที่ใช้กับโปรแกรมนี้

คำสั่งของรูปแบบการแสดงผลบนจอ (format in command)

1.	CUT	เป็นการแสดงภาพปกติโดยนำภาพถัดไปมาทับภาพแรก
2.	PIC_DOT	เป็นการแสดงภาพทับภาพเดิมโดยจะค่อยๆ แสดงทับภาพเดิมทีละจุดแบบ random
3.	FEED_UP	เป็นการแสดงภาพโดยภาพจะค่อยๆ เลื่อนขึ้น
4.	FEED_DOWN	เป็นการแสดงภาพโดยภาพจะค่อยๆ เลื่อนลง
5.	DARK_IN	เป็นการแสดงภาพโดยภาพจะค่อยๆ สว่างจากจอที่มืดอยู่จนเห็นเป็นภาพที่ชัดเจน
6.	LIGHT_IN	เป็นการแสดงภาพโดยภาพจะค่อยๆ มืดลงจากจอที่สว่างอยู่จนเห็นเป็นภาพที่ชัดเจน
7.	DARK_DOT_IN	เป็นการแสดงภาพโดยภาพจะค่อยๆ แสดงภาพทีละจุดบนจอที่มืดอยู่
8.	LIGHT_DOT_IN	เป็นการแสดงภาพโดยภาพจะค่อยๆ แสดงภาพทีละจุดบนจอที่สว่างอยู่

9. SPIRAL เป็นการแสดงภาพโดยภาพจะม้วนเป็นแบบก้นหอย
10. WIPE เป็นการแสดงภาพโดยภาพที่แสดงจะเหมือนกับเปิดผ้าม่าน
11. WIN เป็นการแสดงภาพโดยภาพที่แสดงจะเหมือนกับเปิดบานเกร็ด
12. SNAKE เป็นการแสดงภาพโดยภาพที่แสดงจะเหมือนกับงูเลื้อย
13. SQUARE เป็นการแสดงภาพโดยภาพที่แสดงจะถูกแสดงเป็นสี่เหลี่ยมเล็กๆ อย่าง random จนกว่าจะเห็นภาพหมดทุกส่วน

คำสั่งของรูปแบบการหายไปของภาพบนจอ (format out command)

1. CUT_DARK เป็นการหายไปของภาพ โดยภาพจะมืดทันที
2. DARK_OUT ภาพจะค่อยๆ มืดลงจนจอมืดสนิท
3. LIGHT_OUT ภาพจะค่อยๆ สว่างขึ้นจนจอสว่างเป็นสีขาว
4. DARK_DOT_OUT ภาพจะค่อยๆ หายไปทีละจุด จนกว่าจอจะมืดทั้งหมด
5. LIGHT_DOT_OUT ภาพจะค่อยๆ หายไปทีละจุด จนกว่าจอจะสว่างทั้งหมด

ตัวอย่างการป้อนไฟล์ภาพที่ต้องการแสดง

FILE	FORMAT-IN	TIME	FORMAT-OUT
ZOE.PCX	CUT	5	DARK_OUT
KIM.TGA	DARK_IN	10	LIGHT_OUT
SKY.LBM	LIGHT_IN	5	LIGHT_DOT_OUT
SUNSET.BMP	LIGTH_DOT_IN	10	
BIGLAKE.WPG	FEED_UP	9	CUT_DARK

จากตัวอย่างการป้อนไฟล์ภาพในส่วนของ FILE จะป้อนชื่อไฟล์ภาพที่ต้องการแสดงไว้ในส่วนนี้ คำสั่งของรูปแบบการแสดงป้อนไว้ในส่วนของ FORMAT_IN เวลาในการหน่วงของแต่ละภาพไว้ในส่วนของ TIME และคำสั่งของรูปแบบการหายไปของภาพไว้ในส่วนของ FORMAT_OUT ในกรณีที่ต้องการแสดงภาพทับภาพเดิมไม่จำเป็นต้องมีคำสั่งของรูปแบบการหายไปก็ได้ แต่คำสั่งของรูปแบบการแสดงต้องมีทุกครั้ง

บทที่ 6

บทวิจารณ์ สรุป และแนวทางในการพัฒนา

6.1 บทสรุป

โปรแกรมนำเสนอภาพบนไมโครคอมพิวเตอร์นี้ ทางผู้จัดทำได้วางขอบเขตความสามารถของโปรแกรมไว้สังเขป ดังนี้

- ต้องสามารถแสดงภาพได้หลายๆ ฟอร์แมต
- สามารถเลือกการแสดงภาพบนหน้าจอว่าต้องการให้แสดงแบบต่อเนื่อง หรือแบบทีละภาพโดยรอรับคีย์ใดๆ เพื่อเปลี่ยนเป็นภาพถัดไป
- สามารถเก็บข้อมูลที่อยู่บนหน้าจอไว้ในแฟ้มข้อมูลที่เราสร้างขึ้น ซึ่งจะอยู่ในไฟล์ .POT
- สามารถโหลดข้อมูลจากแฟ้มข้อมูลขึ้นมาแสดง หรือทำการแก้ไขข้อมูลเก่าได้
- มีคำอธิบายเพื่อแนะนำวิธีการใช้โปรแกรมให้กับผู้ใช้

จากการที่ได้ทดลองโปรแกรมนี้แล้ว ปรากฏผลที่ได้ เป็นที่น่าพอใจระดับหนึ่ง ข้อจำกัดของโปรแกรมนี้ก็คือ ภาพที่นำมาแสดงจะต้องมีความละเอียด 320*200 จุด แต่ข้อดีก็คือสามารถแสดงสีได้สูงสุด 256 สี ซึ่งในปัจจุบันนี้โปรแกรมนำเสนอภาพ ตามท้องตลาดก็มีอยู่บ้างแต่ที่สามารถแสดงสีได้ถึง 256 สีนั้นยังมีอยู่น้อย โปรแกรมนี้ยังสามารถพัฒนาต่อไปได้อีกมากมาย

6.2 แนวทางในการพัฒนา

ทางผู้จัดทำพยายามที่จะจัดทำโปรแกรมนี้ให้เป็นโปรแกรมนำเสนอภาพ ที่มีประสิทธิภาพมากที่สุดเท่าที่จะทำได้ในระยะเวลาที่จำกัด และนอกเหนือจากนี้เรายังสามารถที่จะพัฒนาโปรแกรมให้ดีขึ้น เช่น

- สามารถเสนอภาพพร้อมด้วยเสียงบรรยายประกอบการแสดงภาพ ซึ่งจะคล้ายกับสไลด์มัลติวิชั่น

- สามารถแสดงภาพออกทางเครื่องพิมพ์
- สามารถแสดงภาพทุกๆ ฟอร์แมตและทุกๆ ความละเอียด นอกเหนือจากความสามารถของโปรแกรมนี้

ภาคผนวก

โปรแกรมการทำงาน

```
**********

JECT-1.C

**********

#include <conio.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <mem.h>
#include <string.h>
#include <math.h>
#include <bios.h>
#include <process.h>
#include <graphics.h>
#include <dir.h>
#include <stdarg.h>
#include <ctype.h>
#include "info.h"

extern sunmenu1();
extern void keyin(int length);
extern void ins(int length);
extern void v_line();
extern void save();
extern void select();
```

```
extern void shadow();
extern void dir();
extern void load();
extern auto_com();

void run();

char *farPtr(char *p,long l);
int putline(char *p,unsigned int n);
char *getline(unsigned int n);

char modevdo=3,flag_alloc=0;
int delayout=50;
char *taget,*buff,*taget1,*taget2,*taget3,*taget4,*buff_org;
char *pic_name;
char palet_tab[768];          /* palette of screen image */
unsigned length_fname;
unsigned ptr_fname;  /* pointer on current image (taget) */
unsigned cnt_color_scr[256],cnt_color_buf[256];
unsigned point[256]; /* index pointer for transform palette */
unsigned long cnt_buff;
unsigned cnt_width;
unsigned scr_width;
unsigned scr_depth;
char current_vga_bank=0;

FILE *fp;
```

```
char *sel;
char *pic_old;
FILEINFO fi;
unsigned char readin,subreadin;
unsigned char str[500][70];
char name[20];
struct ffblk blk;
int aa,a,b,x,y,xy,xx,yy,ly;
/*****************************************************/
void main()
{
 clrscr();
 submenu();
 submenu1();
 keyin(73);
 closegraph();
}
/*****************************************************/

void run()
{
if(sel!=NULL){
taget= (char *) malloc((unsigned)65535);
if(taget != NULL)
{
read_name(sel);
gphmode(19);
```

```
auto_com();
txtmode();
free(pic_name);
free(taget);
}
submenu1();
shadow();
}
}
/*******************************************************/
 read_name(char name[])
{
 FILE *stream;
 unsigned long cnt,cnt1;
 int n=0;

 if((stream=fopen(name,"r"))!=NULL)
 {
  length_fname=fread(taget,1,64000,stream);
  fclose(stream);
 /*--------allocate OBJ----------*/
  pic_name = (char *) malloc((unsigned)length_fname);
  if (pic_name == NULL)
  terminate("Good bye... \n");

  strupr(taget);
  cnt1=0,cnt=0;
```

```
  while(cnt<length_fname)
  {
   switch(taget[cnt])
   {
    case ' ' :
    case '\t':
    case '\n':while(cnt<length_fname&&(taget[cnt]==' '||
        taget[cnt]=='\t'||
        taget[cnt]=='\n')){
      pic_name[cnt1]=0;cnt++;}
      n++;
      break;
    default  :
      if(n!=0){
      cnt1++;n=0;
      }
      pic_name[cnt1]=taget[cnt];
      cnt1++;cnt++;
      break;
   }
  }
  length_fname=cnt1;
 }
 else
  terminate("COMMAND file can't open...");
}
/*****************************************/
```

```
unpack(char name[])
{
 char *type_img[]={"BMP","TGA","WPG","LBM","PCX","TIF","GIF"};
 unsigned cnt=0,cnt1;
 char flag;

 while(name[cnt]!='.')      /* seek to ext file name */
  cnt++;
 cnt++;
 for(cnt1=0,flag=0xff;cnt1<7&&flag!=0;cnt1++)
  flag=strcmp(type_img[cnt1],name+cnt);
 switch(cnt1-1)
 {
  case 0:
unpack_bmp(name);
break;
  case 1:
unpack_tga(name);
break;
  case 2:
unpack_wpg(name);
break;
  case 3:
unpack_iff(name);
break;
  case 4:
unpack_pcx(name);
```

```
break;
  case 5:
unpack_tif(name);
break;
  case 6:
/*unpack_gif(name);*/
break;
 }
}
/*****************************************/

terminate(char disp[])
{
 txtmode();
 printf("%s",disp);
 exit(0);
}
/*****************************************************/
cut_out()
{
 dark_screen();
}
/*****************************************/
fad_out_1(char t)
{
 int cnt;
 for(cnt=0;cnt<=55;cnt++)
```

```
 {
  delay(t);
  colorpalet(cnt);
 }
  dark_screen();
}
/*****************************************/
fad_out_lw(char t)
{
 int cnt;
 for(cnt=0;cnt<=63;cnt++)
 {
  delay(t);
  colorpalet_w(cnt);
 }
}
/*****************************************/
fad_out_150(char t)
{
 int cnt;
 for(cnt=0;cnt<=30;cnt++)
 {
  delay(t);
  colorpalet(cnt);
 }
}
/*****************************************/
```

```
fad_out_d()
{
 unsigned counter=0,data;
 char *note;
 unsigned bank_mem;

 if(flag_alloc==1)
  bank_mem=65536/4;
 else
  bank_mem=16000;

 /*--------allocate OBJ----------*/
 note= (char *) malloc((unsigned)bank_mem);
 if (note== NULL)
    terminate("Allocate failed note\n");
 palet_tab[0]=0;
 palet_tab[256]=0;              /* backgroud black */
 palet_tab[256*2]=0;
 colorpalet(0);

 memset(note,0,bank_mem);
 do
 {
  data=random(bank_mem);
  if(*(note+data)!=0xff)
  {
   if(flag_alloc==1)
```

```
    fad_out_random640(data);
   else
    fad_out_random(data,bank_mem);
   *(note+data)=0xff;
   counter++;
  }
 }
while(counter<15950);
for(data=0;data<bank_mem;data++)
 if(*(note+data)!=0xff)
 {
  if(flag_alloc==1)
   fad_out_random640(data);
  else
   fad_out_random(data,bank_mem);
 }
free(note);
}
/*****************************************/
fad_out_random640(unsigned data)
{
 fad_out_random(data,65536/4);
 bank_sel(1);
 fad_out_random(data,65536/4);
 bank_sel(2);
 fad_out_random(data,65536/4);
 bank_sel(3);
```

```
 fad_out_random(data,65536/4);
 bank_sel(4);
 fad_out_random(data,45060/4);
 bank_sel(0);
}
/*****************************************/
fad_out_random(unsigned data,unsigned num_bank)
{
 pokeb(0xa000,data,0);
 pokeb(0xa000,data+num_bank,0);
 pokeb(0xa000,data+num_bank*2,0);
 pokeb(0xa000,data+num_bank*3,0);
}
/*****************************************/
fad_out_dw()
{
 unsigned counter=0,data;
 char *note;
 unsigned bank_mem;

 if(flag_alloc==1)
  bank_mem=65536/4;
 else
  bank_mem=16000;
 /*--------allocate OBJ----------*/
 note= (char *) malloc((unsigned)bank_mem);
 if (note== NULL)
```

```
   terminate("Allocate failed note\n");
palet_tab[0]=63;
palet_tab[256]=63;              /* backgroud white */
palet_tab[256*2]=63;
colorpalet(0);

memset(note,0,bank_mem);
do
{
 data=random(bank_mem);
 if(*(note+data)!=0xff)
 {
  if(flag_alloc==1)
   fad_out_random640(data);
  else
   fad_out_random(data,bank_mem);
  *(note+data)=0xff;
  counter++;
 }
}
while(counter<15950);
for(data=0;data<bank_mem;data++)
 if(*(note+data)!=0xff)
 {
  if(flag_alloc==1)
   fad_out_random640(data);
  else
```

```
    fad_out_random(data,bank_mem);
   }
 free(note);
}
/****************************************/
feed_up_out()
{
 int cnt1,cnt;
 cnt1=0;

   do
   {
    for(cnt=0;cnt<=scr_depth-cnt1;cnt++)
     memmove(MK_FP(0xa000,cnt*scr_width),MK_FP(0xa000,cnt*scr_width+scr_width),
    cnt1++;
   }
   while(cnt1<scr_depth);
}
/****************************************/
/*        input image to sreen */
/****************************************/
feed_up_in_out(char step)
{
 int cnt1,cnt;



 color128scr();
```

```
 color128buf();
 exchange_palette();
 colorpalet(0);
 cnt1=0;
 do
 {
  memmove(MK_FP(0xa000,scr_depth*scr_width-scr_width*step),(taget+cnt1*scr_widt
  for(cnt=0;(cnt+step)<scr_depth;cnt=cnt+step)
   memmove(MK_FP(0xa000,cnt*scr_width),MK_FP(0xa000,cnt*scr_width+scr_width*ste
  cnt1=cnt1+step;
 }
 while((cnt1+step)<scr_depth&&bioskey(1)==0);
 memmove(MK_FP(0xa000,scr_depth*scr_width-scr_width*step),(taget+cnt1*scr_width

 if(bioskey(1)!=0) /* check key press */
  terminate("terminate by user.");
 unpack(pic_name+ptr_fname);
 tranto256scr();
 cut_in();
}
/*****************************************/
feed_down_in_out(char step)
{
 int cnt1,cnt;
 color128scr();
 color128buf();
 exchange_palette();
```

```
 colorpalet(0);
 cnt1=scr_depth-step;
 do
 {
  movmem(MK_FP(0xa000,0),MK_FP(0xa000,scr_width*step),scr_width*(scr_depth-step
  movmem(taget+cnt1*scr_width,MK_FP(0xa000,0),scr_width*step);
  cnt1=cnt1-step;
 }
 while(cnt1>0&&bioskey(1)==0);
 movmem(MK_FP(0xa000,0),MK_FP(0xa000,scr_width*step),scr_width*(scr_depth-step)
 movmem(taget,MK_FP(0xa000,0),scr_width*step);

 if(bioskey(1)!=0)/* check key press */
  terminate("terminate by user.");
 unpack(pic_name+ptr_fname);
 tranto256scr();
 cut_in();
}
/*****************************************/
cut_in()
{
 unsigned cnt;

 colorpalet(0);
 if (flag_alloc==0)
  movmem(taget,MK_FP(0xa000,0),64000);
 else
```

```
 {
  movmem(taget,MK_FP(0xa000,0),65535);
  bank_sel(1);
  movmem(taget1,MK_FP(0xa000,0),65535);
  bank_sel(2);
  movmem(taget2,MK_FP(0xa000,0),65535);
  bank_sel(3);
  movmem(taget3,MK_FP(0xa000,0),65535);
  bank_sel(4);
  movmem(taget4,MK_FP(0xa000,0),45060);
  bank_sel(0);
 }
}
/*****************************************/

fad_in_1(char t)
{
 int cnt;

 dark_screen();
 if (flag_alloc==0)
  movmem(taget,MK_FP(0xa000,0),64000);
 else
 {
  movmem(taget,MK_FP(0xa000,0),65535);
  bank_sel(1);
  movmem(taget1,MK_FP(0xa000,0),65535);
```

```
  bank_sel(2);
  movmem(taget2,MK_FP(0xa000,0),65535);
  bank_sel(3);
  movmem(taget3,MK_FP(0xa000,0),65535);
  bank_sel(4);
  movmem(taget4,MK_FP(0xa000,0),45060);
  bank_sel(0);
 }
 for(cnt=55;cnt>=0;cnt--)
 {
  delay(t);
  colorpalet(cnt);
 }
}
/*****************************************/
fad_in_lw(char t)
{
 int cnt;

 if (flag_alloc==0)
  movmem(taget,MK_FP(0xa000,0),64000);
 else
 {
  movmem(taget,MK_FP(0xa000,0),65535);
  bank_sel(1);
  movmem(taget1,MK_FP(0xa000,0),65535);
  bank_sel(2);
```

```
  movmem(taget2,MK_FP(0xa000,0),65535);
  bank_sel(3);
  movmem(taget3,MK_FP(0xa000,0),65535);
  bank_sel(4);
  movmem(taget4,MK_FP(0xa000,0),45060);
  bank_sel(0);
 }
 for(cnt=55;cnt>=0;cnt--)
 {
  delay(t);
  colorpalet_w(cnt);
 }
}
/*****************************************/
win()
{
int off,i;
color128scr();
color128buf();
exchange_palette();
colorpalet(0);
for(off=40;off>=0;off--)
{
  for(i=0;i<320;i++)
 {
    pokeb(0xa000,(off*320+i),taget[off*320+i]);
    pokeb(0xa000,((off+40)*320+i),taget[(off+40)*320+i]);
```

```
    pokeb(0xa000,((off+80)*320+i),taget[(off+80)*320+i]);
    pokeb(0xa000,((off+120)*320+i),taget[(off+120)*320+i]);
    pokeb(0xa000,((off+160)*320+i),taget[(off+160)*320+i]);
}
}
unpack(pic_name+ptr_fname);
tranto256scr();
cut_in();
}
/*************************************************************/
wipe()
{
int i,j,n;
color128scr();
color128buf();
exchange_palette();
colorpalet(0);
for(j=0;j<162;j++)
{
    for(i=0;i<200;i++)
  {
    n=159+(320*i);
    pokeb(0xa000,n-j,taget[n-j]);
    pokeb(0xa000,n+j,taget[n+j]);
  }
}
unpack(pic_name+ptr_fname);
```

```
tranto256scr();
cut_in();
}
/****************************************/
spiral()
{
int i,x,y,x0,x1,y0,y1;
color128scr();
color128buf();
exchange_palette();
colorpalet(0);
for(i=1;i<=20;i++)
{
switch(i)
{
case 1: x0=0;x1=300;y0=0;y1=20;a1(x0,x1,y0,y1);break;
case 2: x0=300;x1=320;y0=0;y1=180;a2(x0,x1,y0,y1);break;
case 3: x0=319;x1=20;y0=180;y1=200;a3(x0,x1,y0,y1);break;
case 4: x0=0;x1=20;y0=200;y1=20;a4(x0,x1,y0,y1);break;
case 5: x0=20;x1=280;y0=20;y1=40;a1(x0,x1,y0,y1);break;
case 6: x0=280;x1=300;y0=20;y1=160;a2(x0,x1,y0,y1);break;
case 7: x0=299;x1=40;y0=160;y1=180;a3(x0,x1,y0,y1);break;
case 8: x0=20;x1=40;y0=180;y1=40;a4(x0,x1,y0,y1);break;
case 9: x0=40;x1=260;y0=40;y1=60;a1(x0,x1,y0,y1);break;
case 10: x0=260;x1=280;y0=40;y1=140;a2(x0,x1,y0,y1);break;
case 11: x0=279;x1=60;y0=140;y1=160;a3(x0,x1,y0,y1);break;
case 12: x0=40;x1=60;y0=160;y1=60;a4(x0,x1,y0,y1);break;
```

```
case 13:  x0=60;x1=240;y0=60;y1=80;a1(x0,x1,y0,y1);break;
case 14:  x0=240;x1=260;y0=60;y1=120;a2(x0,x1,y0,y1);break;
case 15:  x0=259;x1=80;y0=120;y1=140;a3(x0,x1,y0,y1);break;
case 16:  x0=60;x1=80;y0=140;y1=80;a4(x0,x1,y0,y1);break;
case 17:  x0=80;x1=220;y0=80;y1=100;a1(x0,x1,y0,y1);break;
case 18:  x0=220;x1=240;y0=80;y1=100;a2(x0,x1,y0,y1);break;
case 19:  x0=239;x1=100;y0=100;y1=120;a3(x0,x1,y0,y1);break;
case 20:  x0=80;x1=100;y0=120;y1=100;a4(x0,x1,y0,y1);break;
case 21:  x0=100;x1=200;y0=100;y1=120;a4(x0,x1,y0,y1);break;
 }
}
unpack(pic_name+ptr_fname);
tranto256scr();
cut_in();
}
a1(int x0,int x1,int y0,int y1)
{
int x,y;
for(x=x0;x<x1;x++)
for(y=y0;y<y1;y++)
pokeb(0xa000,(320*y)+x,taget[(320*y)+x]);
}
a2(int x0,int x1,int y0,int y1)
{
int x,y;
for(y=y0;y<y1;y++)
for(x=x0;x<x1;x++)
```

```
pokeb(0xa000,(320*y)+x,taget[(320*y)+x]);
}
a3(int x0,int x1,int y0,int y1)
{
int x,y;
for(x=x0;x>=x1;x--)
for(y=y0;y<y1;y++)
pokeb(0xa000,(320*y)+x,taget[(320*y)+x]);
}
a4(int x0,int x1,int y0,int y1)
{
int x,y;
for(y=y0;y>=y1;y--)
for(x=x0;x<x1;x++)
pokeb(0xa000,(320*y)+x,taget[(320*y)+x]);
}
/*****************************************************/
square()
{
 int p[40];
 int i;
 int x1,x2,y1,y2;
 unsigned counter=0,data;
 color128scr();
 color128buf();
 exchange_palette();
 colorpalet(0);
```

```
do
{
 data=random(40);
   switch(data)
   {
   case 0: x1=0;x2=40;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 1: x1=40;x2=80;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 2: x1=80;x2=120;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 3: x1=120;x2=160;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 4: x1=160;x2=200;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 5: x1=200;x2=240;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 6: x1=240;x2=280;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 7: x1=280;x2=320;y1=0;y2=40;ran1(x1,x2,y1,y2);break;
   case 8: x1=0;x2=40;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 9:  x1=40;x2=80;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 10: x1=80;x2=120;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 11: x1=120;x2=160;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 12: x1=160;x2=200;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 13: x1=200;x2=240;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 14: x1=240;x2=280;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 15: x1=280;x2=320;y1=40;y2=80;ran1(x1,x2,y1,y2);break;
   case 16: x1=0;x2=40;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
   case 17: x1=40;x2=80;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
   case 18: x1=80;x2=120;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
   case 19: x1=120;x2=160;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
   case 20: x1=160;x2=200;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
   case 21:  x1=200;x2=240;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
```

```
    case 22:  x1=240;x2=280;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
    case 23:  x1=280;x2=320;y1=80;y2=120;ran1(x1,x2,y1,y2);break;
    case 24:  x1=0;x2=40;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 25:  x1=40;x2=80;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 26: x1=80;x2=120;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 27: x1=120;x2=160;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 28: x1=160;x2=200;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 29: x1=200;x2=240;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 30: x1=240;x2=280;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 31: x1=280;x2=320;y1=120;y2=160;ran1(x1,x2,y1,y2);break;
    case 32:  x1=0;x2=40;y1=120;y2=200;ran1(x1,x2,y1,y2);break;
    case 33:  x1=40;x2=80;y1=160;y2=200;ran1(x1,x2,y1,y2);break;
    case 34: x1=80;x2=120;y1=160;y2=200;ran1(x1,x2,y1,y2);break;
    case 35: x1=120;x2=160;y1=160;y2=200;ran1(x1,x2,y1,y2);break;
    case 36: x1=160;x2=200;y1=160;y2=200;ran1(x1,x2,y1,y2);break;
    case 37: x1=200;x2=240;y1=160;y2=200;ran1(x1,x2,y1,y2);break;
    case 38: x1=240;x2=280;y1=160;y2=200;ran1(x1,x2,y1,y2);break;
    case 39: x1=280;x2=320;y1=160;y2=200;ran1(x1,x2,y1,y2);break;
    }
   counter++;
 }
 while(counter<300);
 unpack(pic_name+ptr_fname);
 tranto256scr();
 cut_in();
}
ran1(x1,x2,y1,y2)
```

```
int x1,x2,y1,y2;
{
int x,y;
for(y=y1;y<y2;y++)
for(x=x1;x<x2;x++)
pokeb(0xa000,(320*y)+x,taget[(320*y)+x]);
}
/*******************************************************/
snake()
{
int i;
int x1,x2,x3,x4,x5,x6,x7,x8;
color128scr();
color128buf();
exchange_palette();
colorpalet(0);

for(i=0;i<200;i++)
{delay(1);
for(x1=0;x1<40;x1++)
 pokeb(0xa000,320*i+x1,taget[320*i+x1]);
}
for(i=200;i>=0;i--)
{delay(1);
for(x2=40;x2<80;x2++)
 pokeb(0xa000,320*i+x2,taget[320*i+x2]);
}
```

```
for(i=0;i<200;i++)
{delay(1);
for(x3=80;x3<120;x3++)
 pokeb(0xa000,320*i+x3,taget[320*i+x3]);
}
for(i=200;i>=0;i--)
{delay(1);
for(x4=120;x4<160;x4++)
 pokeb(0xa000,320*i+x4,taget[320*i+x4]);
}
for(i=0;i<200;i++)
{delay(1);
for(x5=160;x5<200;x5++)
 pokeb(0xa000,320*i+x5,taget[320*i+x5]);
}
for(i=200;i>=0;i--)
{delay(1);
for(x6=200;x6<240;x6++)
 pokeb(0xa000,320*i+x6,taget[320*i+x6]);
}
for(i=0;i<200;i++)
{delay(1);
for(x7=240;x7<280;x7++)
 pokeb(0xa000,320*i+x7,taget[320*i+x7]);
}
for(i=200;i>=0;i--)
{delay(1);
```

```
for(x8=280;x8<320;x8++)
 pokeb(0xa000,320*i+x8,taget[320*i+x8]);
}
unpack(pic_name+ptr_fname);
tranto256scr();
cut_in();
}
/*****************************************/
fad_in_150(char t)
{
 int cnt;

 colorpalet(30);
 if (flag_alloc==0)
  movmem(taget,MK_FP(0xa000,0),64000);
 else
 {
  movmem(taget,MK_FP(0xa000,0),65535);
  bank_sel(1);
  movmem(taget1,MK_FP(0xa000,0),65535);
  bank_sel(2);
  movmem(taget2,MK_FP(0xa000,0),65535);
  bank_sel(3);
  movmem(taget3,MK_FP(0xa000,0),65535);
  bank_sel(4);
  movmem(taget4,MK_FP(0xa000,0),45060);
  bank_sel(0);
```

```
 }
 for(cnt=30;cnt>=0;cnt--)
 {
  delay(t);
  colorpalet(cnt);
 }
}
/*****************************************/
fad_in_d()
{
 unsigned counter=0,data;
 char *note;
 unsigned bank_mem;

 if(flag_alloc==1)
  bank_mem=65536/4;
 else
  bank_mem=16000;
 /*--------allocate OBJ----------*/
 note= (char *) malloc((unsigned)bank_mem);
 if (note== NULL)
    terminate("Allocate failed note\n");
 palet_tab[0]=0;
 palet_tab[256]=0;          /* backgroud black */
 palet_tab[256*2]=0;
 colorpalet(0);
 memset(note,0,bank_mem);
```

```
do
{
 data=random(bank_mem);
 if(*(note+data)!=0xff)
 {
  if(flag_alloc==1)
   fad_in_random640(data);
  else
   fad_in_random(taget,data,bank_mem);
  *(note+data)=0xff;
  counter++;
 }
}
while(counter<15950);

for(data=0;data<bank_mem;data++)
 if(*(note+data)!=0xff)
 {
  if(flag_alloc==1)
   fad_in_random640(data);
  else
   fad_in_random(taget,data,bank_mem);      /* fill remain after random */
 }
free(note);
}
/*****************************************/
fad_in_random640(unsigned data)
```

```
{
 fad_in_random(taget,data,65536/4);
 bank_sel(1);
 fad_in_random(taget1,data,65536/4);
 bank_sel(2);
 fad_in_random(taget2,data,65536/4);
 bank_sel(3);
 fad_in_random(taget3,data,65536/4);
 bank_sel(4);
 fad_in_random(taget4,data,45060/4);
 bank_sel(0);
}
/*****************************************/
fad_in_random(char *tag,unsigned data,unsigned num_bank)
{
 pokeb(0xa000,data,*(tag+data));
 pokeb(0xa000,data+num_bank,*(tag+data+num_bank));
 pokeb(0xa000,data+num_bank*2,*(tag+data+num_bank*2));
 pokeb(0xa000,data+num_bank*3,*(tag+data+num_bank*3));
}
/*****************************************/
fad_in_dw()
{
 unsigned counter=0,data;
 char *note;
 unsigned bank_mem;
```

```
if(flag_alloc==1)
 bank_mem=65536/4;
else
 bank_mem=16000;
/*--------allocate OBJ----------*/
note= (char *) malloc((unsigned)bank_mem);
if (note== NULL)
   terminate("Allocate failed note\n");
palet_tab[0]=63;
palet_tab[256]=63;      /* backgroud white */
palet_tab[256*2]=63;
colorpalet(0);

memset(note,0,bank_mem);
do
{
 data=random(bank_mem);
 if(*(note+data)!=0xff)
 {
  if(flag_alloc==1)
   fad_in_random640(data);
  else
   fad_in_random(taget,data,bank_mem);
  *(note+data)=0xff;
  counter++;
 }
}
```

```
 while(counter<15000);

 for(data=0;data<bank_mem;data++)
  if(*(note+data)!=0xff)
  {
   if(flag_alloc==1)
    fad_in_random640(data);
   else
    fad_in_random(taget,data,bank_mem);     /* fill remain after random */
   }
 palet_tab[0]=0;
 palet_tab[256]=0;       /* backgroud black */
 palet_tab[256*2]=0;
 colorpalet(0);
 free(note);
}
/*****************************************/
fad_out_d_over()
{
 unsigned counter=0,data;
 char *note;
 unsigned bank_mem;

 if(flag_alloc==1)
  bank_mem=65536/4;
 else
  bank_mem=16000;
```

```
color128scr();
color128buf();
exchange_palette();
colorpalet(0);
/*--------allocate note----------*/
note= (char *) malloc((unsigned)bank_mem);
if (note== NULL)
   terminate("Allocate failed note\n");
memset(note,0,bank_mem);

do
{
 data=random(bank_mem);
 if(*(note+data)!=0xff)
 {
  if(flag_alloc==1)
   fad_in_random640(data);
  else
   fad_in_random(taget,data,bank_mem);
  *(note+data)=0xff;
  counter++;
 }
}
while(counter<15950);
for(data=0;data<bank_mem;data++)
 if(*(note+data)!=0xff)
 {
```

```
    if(flag_alloc==1)
     fad_in_random640(data);
    else
     fad_in_random(taget,data,bank_mem);
   }
  free(note);
  unpack(pic_name+ptr_fname);
  tranto256scr();
  cut_in();
 }

 char *farPtr(p,l)
 char *p;
 long l;
 {
  unsigned int seg,off;
  seg=FP_SEG(p);
  off=FP_OFF(p);
  seg+=(off/16);
  off&=0x000f;
  off+=(unsigned int)(l&0x000fL);
  seg+=(l/16L);
  p=MK_FP(seg,off);
  return(p);
 }
 putline(char *p,unsigned int n)
 {
```

```
if(n>=0 && n<fi.depth)
 memcpy(farPtr(taget,(long)n*(long)fi.width),p,fi.width);
}
char *getline(unsigned int n)
{
return(farPtr(taget,(long)n*(long)fi.width));
}
strmfe(char *new,char *old,char *ext)
{
while(*old != 0 && *old != '.') *new++=*old++;
*new++='.';
while(*ext) *new++=*ext++;
*new=0;
}

fgetword(FILE *fp)
{
return((fgetc(fp) & 0xff) + ((fgetc(fp) & 0xff) << 8));
}
```

```
**********************

PCX.C

**********************

#include <conio.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <mem.h>
#include <string.h>
#include <math.h>
#include <bios.h>
#include <process.h>
#include "info.h"

extern FILE *fp;
extern char *taget,*buff;
extern unsigned long cnt_buff;
extern unsigned scr_width;
extern unsigned scr_depth;
extern unsigned cnt_width;
extern FILEINFO fi;
/*****************************************/
unpack_pcx(char name[])
{
```

```
FILE *stream;
unsigned row,col;
int cnt;
char tmpstr[50];

if((stream=fopen(name,"rb"))==NULL)
{
 sprintf(tmpstr,"PCX file (%s) can't open...",name);
 terminate(tmpstr);
}
else
{
 if(fgetc(stream)!=0xa)          /* Zsoft PCX file */
  terminate("file PCX formate Error.");
 else
 {
  fseek(stream,3,SEEK_SET);
  fi.bits=fgetc(stream);          /* get bit per bytes */
  fseek(stream,65,SEEK_SET);
  fi.plane=fgetc(stream);         /* No. of plane */
  fi.bytes=fgetword(stream);        /* get byte per line */
  fseek(stream,4,SEEK_SET);
  row=fgetword(stream);
  col=fgetword(stream);
  fi.width=fgetword(stream)+1;
  fi.depth=fgetword(stream)+1;
  if(fi.width>scr_width)
```

```
{
 sprintf(tmpstr,"%s size over than screen. SCREEN x=%d,y=%d,
   FILE x=%d,y=%d\n",name,scr_width,scr_depth,fi.width,fi.depth);
 terminate(tmpstr);
}
fseek(stream,128,SEEK_SET);
run_lengPCX(stream);
switch(fi.bits)
{
 case 1:/* 1 bits */
if(fi.plane==1)          /* 2 color image */
{
 fi.palette[1]=63;
 fi.palette[1+256]=63;
 fi.palette[1+256*2]=63;
}
else
{
 if(fi.plane==4)     /* 4 color image */
 {
  fseek(stream,16,SEEK_SET);
  for(row=0;row<16;row++)
  {
   fi.palette[row]=fgetc(stream);
   fi.palette[row+256]=fgetc(stream);
   fi.palette[row+256*2]=fgetc(stream);
  }
```

```
   }
   else
    terminate("PCX file plane don't support");
  free(buff);
  break;
    case 8:/* 8 bits */
  fseek(stream,-769,SEEK_END);
  if(fgetc(stream)!=0xc)
   terminate("file PCX format Error.");
  for(row=0;row<256;row++)
   for(col=0;col<3;col++)
    fi.palette[row+col*256]=getc(stream)>>2;
  break;
    }
   }
  }
  fclose(stream);
 }
}
/*****************************************/
run_lengPCX(FILE *stream)
{
 unsigned i,j,counter;
 char data;

 cnt_buff=0;
 cnt_width=0;
```

```
 while((unsigned long)cnt_buff<=((unsigned long)fi.depth*scr_width))
 {
  if((data=fgetc(stream))>0xc0)
  {
   counter=(data^0xc0);          /* XOR < 63  */
   data=fgetc(stream);
   for(i=counter;i>0;i--)
    Sav_Byte_Mem(data);
  }
  else
    Sav_Byte_Mem(data);
 }
}
/*****************************************/
```

```
**********************

        WPG.C

**********************

#include <stdio.h>
#include <alloc.h>
#include <dos.h>
#include "head.h"
#include "info.h"

typedef struct {
char id[4];
long start;
char product;
char filetype;
char majorversion;
char minorversion;
unsigned int encrypt;
unsigned int reserved;
} WPGHEAD;

extern FILE *fp;
extern FILEINFO fi;
/*********************************/
unpack_wpg(char name[])
{
char path[81];
```

```
strmfe(path,name,"WPG");
strupr(path);
if((fp = fopen(path,"rb")) != NULL) {
unpackwpg(fp,&fi);
fclose(fp);
} else printf("Error opening %s",path);
}

/* unpack a WPG file */
unpackwpg(fp,fi)
FILE *fp;
FILEINFO *fi;
{
WPGHEAD wpg;
char *p,*pr;
unsigned long offset=0L;
int i;

/* set the dimensions to illegal values */
fi->width=fi->depth=fi->bytes=fi->bits=0;

/* get the header */
if(fread((char *)&wpg,1,sizeof(WPGHEAD),fp)==sizeof(WPGHEAD)) {

/* check the header */
if(!memcmp(wpg.id,"\377WPC",4)) {
```

```
/* find the start of the records */
fseek(fp,wpg.start,SEEK_SET);

/* read all the records */
do {
i=readrecord(fi,fp,&offset);
} while(i != 16 && i != EOF);

/* see if there was a valid bitmap */
if(i==16 &&
   fi->width != 0 &&
   fi->depth != 0 &&
   offset != 0L) {

/* allocate a line buffer */
if((p=malloc(fi->bytes)) != NULL) {

/* find the start of the image data */
fseek(fp,offset,SEEK_SET);

/* read all the lines */
for(i=0;i<fi->depth;++i) {
if(readwpgline(p,fp,fi->bytes)
    != fi->bytes) {
free(p);
return(BAD_READ);
}
```

```
/* translate the line types into VGA */
switch(fi->bits) {
case 8:
putline(p,i);
break;
}
}
free(p);
return(GOOD_READ);
} else return(MEMORY_ERROR);
} else return(BAD_FILE);
} else return(BAD_FILE);
} else return(BAD_READ);
}

/* read one record of a wpg file */
readrecord(fl,fp,offset)
FILEINFO *fl;
FILE *fp;
unsigned long *offset;
{
unsigned long l,t;
unsigned int i,j,type,fc;
int row;

type=fgetc(fp);
t=ftell(fp);
```

```
i=fgetc(fp) & 0x00ff;
if(i == 0xff) {
i=fgetword(fp);
if(i & 0x8000) {
l = (unsigned long)(i & 0x7fff) << 16;
i = fgetword(fp);
l += (((unsigned long)l)+4L);
} else l=(((unsigned int)i)+2L);
} else l=(unsigned long)i;

switch(type) {
case 11:      /* bitmap */
fi->width=fgetword(fp);
fi->depth=fgetword(fp);
fi->bits=fgetword(fp);
fgetword(fp);
fgetword(fp);
*offset=ftell(fp);
if(fi->bits == 8) fi->bytes=fi->width;
else fi->bytes=pixels2bytes(fi->width)*fi->bits;
break;
case 14:/* palette */
fgetword(fp);
fgetword(fp);
  for(row=0;row<256;++row)
 {
 fi->palette[row+0*256]=getc(fp)>>2;
```

```
 fi->palette[row+1*256]=getc(fp)>>2;
 fi->palette[row+2*256]=getc(fp)>>2;
 }
break;
}
fseek(fp,t+1+1L,SEEK_SET);
return(type);
}

/* uncompress one line of a wpg image */
readwpgline(p,fp,bytes)
char *p;
FILE *fp;
int bytes;
{
static int repeat;
int c,d,i,n=0;

if(repeat) {
--repeat;
n=bytes;
}
else {
do {
c=fgetc(fp);
if((c & 0x0080) && (c & 0x007f)) {
d=fgetc(fp) & 0xff;
```

```
for(i=0;i<(c & 0x7f);++i) p[n++]=d;
}
else if((c & 0x0080) && !(c & 0x007f)) {
d=fgetc(fp) & 0xff;
for(i=0;i<d;++i) p[n++]=0xff;
}
else if(!(c & 0x0080) && (c & 0x007f)) {
for(i=0;i<(c & 0x7f);++i) p[n++]=fgetc(fp);
}
else {
repeat=fgetc(fp);
n=bytes;
}
} while(n < bytes);
}
return(n);
}
/************************************/
```

****

BMP.C

****

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include "head.h"
#include "info.h"

typedef struct{
   char id[2];
   long filesize;
   int reserved[2];
   long headersize;
   long infosize;
   long width;
   long depth;
   int biPlanes;
   int bits;
   long biCompression;
   long biSizeImage;
   long biXPelsPerMeter;
   long biYPelsPerMeter;
   long biClrUsed;
   long biClrImportant;
   }BMPHEAD;
```

```
extern FILE *fp;
extern FILEINFO fi;
/*****************************************/
unpack_bmp(char name[])
{
if((fp = fopen(name,"rb")) != NULL) {
unpackbmp(fp,&fi);
fclose(fp);
} else printf("Error opening %s",name);
}

/* unpack a BMP file */
unpackbmp(fp,fi)
FILE *fp;
FILEINFO *fi;
{
BMPHEAD bmp;
char *p,*pr;
int i,n;
int row,col;

 /* get the header */
 if(fread((char *)&bmp,1,sizeof(BMPHEAD),fp)==sizeof(BMPHEAD)) {

 /* check the header */
 if(!memcmp(bmp.id,"BM",2)) {
```

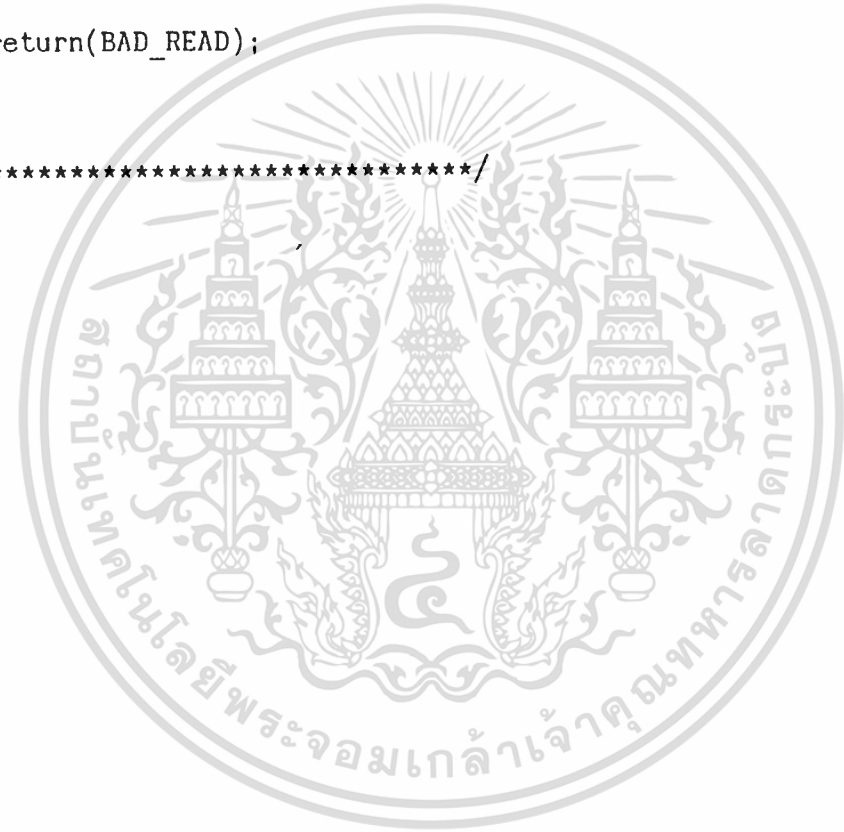
```
/* set the details */
fi->width=(int)bmp.width;
fi->depth=(int)bmp.depth;
fi->bits=bmp.bits;
fi->plane=bmp.biPlanes;
/* work out the line width */
if(fi->bits==8) fi->bytes=fi->width;

 for(row=0;row<256;++row)
 {
 fi->palette[row+2*256]=getc(fp)>>2;
 fi->palette[row+1*256]=getc(fp)>>2;
 fi->palette[row+0*256]=getc(fp)>>2;
 getc(fp);
 }
/* allocate a line buffer */
if((p=malloc(fi->bytes)) != NULL) {

/* find the start of the image data */
fseek(fp,bmp.headersize,SEEK_SET);

/* read all the lines */
for(i=0;i<fi->depth;++i) {
if(fread(p,1,fi->bytes,fp) != fi->bytes) {
free(p);
return(BAD_READ);
}
```

```
/* translate the line types into VGA */
putline(p,fi->depth-1-i);
}
free(p);
return(GOOD_READ);
} else return(MEMORY_ERROR);
} else return(BAD_FILE);
} else return(BAD_READ);
}
/*************************************/
```



```
**********

LBM.C

**********

#include <stdio.h>
#include <alloc.h>
#include <dos.h>
#include "head.h"
#include "info.h"

typedef struct {
unsigned int w,h;
int x,y;
char nPlanes;
char masking;
char compression;
char pad1;
unsigned int transparentColor;
char xAspect,yAspect;
int pageW,pageH;
} BMHD;

long motr2intl(long l);
char *planes2byte(char *line,FILEINFO *fl);

char masktable[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
char bittable[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
```

```
extern FILE *fp;
extern FILEINFO fi;
/************************************/
unpack_iff(char name[])
{
if((fp = fopen(name,"rb")) != NULL) {
unpackiff(fp,&fi);
fclose(fp);
} else printf("Error opening %s",name);
}

/* unpack an IFF/LBM file */
unpackiff(fp,fi)
FILE *fp;
FILEINFO *fi;
{
BMHD bmhd;
unsigned long l;
char *p,*pr,b[4];
int i,n;
int row,col;

/* get the type */
fread(b,1,4,fp);
if(!memcmp(b,"FORM",4) ||
   !memcmp(b,"LIST",4) ||
   !memcmp(b,"CAT ",4)) {
```

```
/* ignore the size */
fread((char *)&l,1,4,fp);

/* get the subtype */
fread(fi->subtype,1,4,fp);

/* read all the chunks */
do {
fread(b,1,4,fp);
fread((char *)&l,1,4,fp);
l=motr2intl(l);
if(l & 1L) ++l;

/* check for a bitmap header */
if(!memcmp(b,"BMHD",4)) {
if(fread((char *)&bmhd,1,sizeof(BMHD),fp)
    != sizeof(BMHD)) return(BAD_FILE);
fi->width=motr2intl(bmhd.w);
fi->depth=motr2intl(bmhd.h);
fi->bits=bmhd.nPlanes;
if(!memcmp(fi->subtype,"ILBM",4))
    fi->bytes=pixels2bytes(fi->width)*fi->bits;
else
    fi->bytes=fi->width;
}

/* check for a palette */
```

```
else if(!memcmp(b,"CMAP",4)) {
   for(row=0;row<256;row++)
   for(col=0;col<3;col++)
    fi->palette[row+col*256]=getc(fp)>>2;
       }
/* check for an image */
else if(!memcmp(b,"BODY",4)) {

if((p=malloc(fi->width)) != NULL) {
for(i=0;i<fi->depth;++i) {
if(bmhd.compression)
    n=readline(p,fp,fi->bytes);
else
    n=fread(p,1,fi->bytes,fp);

if(n != fi->bytes) {
free(p);
return(BAD_READ);
}

if(!memcmp(fi->subtype,"ILBM",4) ||
  (!memcmp(fi->subtype,"PBM ",4) &&
   fi->bits < 8)) {
if((pr=planes2byte(p,fi)) == NULL) {
free(p);
return(MEMORY_ERROR);
}
```

```
putline(pr,i);
free(pr);
} else
putline(p,i);
}
free(p);
} else return(MEMORY_ERROR);
}

/* skip an unknown chunk */
else fseek(fp,l,SEEK_CUR);

} while(!ferror(fp) && memcmp(b,"BODY",4));
return(GOOD_READ);

} else return(BAD_FILE);
}

/* convert a planar line to a VGA line */
char *planes2byte(line,fi)
char *line;
FILEINFO *fi;
{
char *p,*pr;
int i,j,n;

/* allocate a place to put the line */
```

```
if((p=malloc(fi->width)) != NULL) {

/* get the width of one plane */
n=pixels2bytes(fi->width);

/* sscan through the pixels */
for(i=0;i<fi->width;++i) {
pr=line;
p[i]=0;

/* fetch each planar pixel */
for(j=0;j<fi->bits;++j) {
if(pr[i>>3] & masktable[i & 0x0007])
    p[i] |= bittable[j];
pr+=n;
}
}
return(p);
}
return(NULL);
}

/* read a compressed PackBits line */
readline(p,fp,bytes)
char *p;
FILE *fp;
int bytes;
```

```
{
int c,i,n=0;

do {
c=fgetc(fp) & 0xff;
if(c & 0x80) {
if(c != 0x80) {
i = ((~c) & 0xff)+2;
c=fgetc(fp);
while(i--) p[n++] = c;
}
}
else {
i=(c & 0xff)+1;
while(i--) p[n++] = fgetc(fp);
}
} while(n < bytes);
return(n);
}

long motr2intl(l)
long l;
{
return(((l & 0xff000000L) >> 24) +
       ((l & 0x00ff0000L) >> 8) +
       ((l & 0x0000ff00L) << 8) +
       ((l & 0x000000ffL) << 24));
```

```
}
motr2intl(n)
int n;
{
return(((n & 0xff00) >> 8) | ((n & 0x00ff) << 8));
}
/*************************************/
```

```
**********

PSCR.C

**********

#include <conio.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <mem.h>
#include <string.h>
#include <math.h>
#include <bios.h>
#include <process.h>
#include <graphics.h>

/*****************************************************/

Put_Byte_Scr(unsigned long count,char value)
{
 char *buffer;
 unsigned long temp;

 temp=count;
 if((unsigned long)count<65536)          bank_sel(0);
 else
```

```
 if((unsigned long)count<131072) { bank_sel(1); temp=temp-65536; }
 else
  if((unsigned long)count<196608) { bank_sel(2); temp=temp-65536*2; }
  else
   if((unsigned long)count<262144) { bank_sel(3); temp=temp-65536*3;}
   else
    { bank_sel(4); temp=temp-65536*4; }

 pokeb(0xa000,temp,value);
}
/*****************************************/
char Pull_Byte_Scr(unsigned long count)
{
 char *buffer;
 unsigned long temp;

 temp=count;
 if((unsigned long)count<65536)          bank_sel(0);
 else
  if((unsigned long)count<131072)  { bank_sel(1); temp=temp-65536; }
  else
   if((unsigned long)count<196608) { bank_sel(2); temp=temp-65536*2; }
   else
    if((unsigned long)count<262144) { bank_sel(3); temp=temp-65536*3;}
    else
     { bank_sel(4); temp=temp-65536*4; }
```

```
  return(peekb(0xa000,temp));
}
/*****************************************/
```

```
**********

SET.C

**********

#include <conio.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <mem.h>
#include <string.h>
#include <math.h>
#include <bios.h>
#include <process.h>
#include <graphics.h>
#include "info.h"
extern FILEINFO fi;
extern char modevdo,flag_alloc;
extern char *taget,*buff,*tageti,*taget2,*taget3,*taget4,*buff_org;
extern char palet_tab[768];          /* palette of screen image */
extern unsigned cnt_color_scr[256],cnt_color_buf[256];
extern unsigned point[256]; /* index pointer for transform palette */
extern unsigned long cnt_buff;
extern unsigned scr_width;
extern unsigned scr_depth;
extern char current_vga_bank;
```

```
/*****************************************************/
txtmode()
{
 _AX=0x0003;
 geninterrupt(0x10);
}
/*****************************************/
gphmode(char modegph)
{
 switch(modegph)
 {
  case 0x13:  /* (mode 19) 256 color 320*200 */
    if(flag_alloc==1)
    {
     free(taget1);free(taget2);free(taget3);free(taget4);
     flag_alloc=0;
    }
    scr_width=320;
    scr_depth=200;
    break;
  case 0x10:  /* 16 color 640*350 */
  case 0x12:  /* 16 color 640*480 */
  case 0x5c:  /* 256 color 640*400 */
  case 0x5d:  /*(mode 93) 256 color 640*480 */
    scr_width=640;
    scr_depth=480;
    flag_alloc=1;
```

```
    alloc_buffer();
    break;
  default:terminate("Graphic mode can't support.");
 }
 _AH=0;
 _AL=modegph;
 geninterrupt(0x10);
 dark_screen();
 modevdo=modegph;
}
/*****************************************/
alloc_buffer()
{
    flag_alloc=1;
    /*--------allocate OBJ----------*/
    taget1= (char *) malloc((unsigned)65535);
    if (taget1 == NULL)
     terminate("Allocate failed taget1 \n");
    /*--------allocate OBJ----------*/
    taget2= (char *) malloc((unsigned)65535);
    if (taget2 == NULL)
     terminate("Allocate failed taget2 \n");
    /*--------allocate OBJ----------*/
    taget3= (char *) malloc((unsigned)65535);
    if (taget3 == NULL)
     terminate("Allocate failed taget3 \n");
    /*--------allocate OBJ----------*/
```

```
    taget4= (char *) malloc((unsigned)45060);
    if (taget4 == NULL)
     terminate("Allocate failed taget4 \n");
}
/****************************************/

bank_sel(char vga_bank)
{
 unsigned dat;

 if(current_vga_bank==vga_bank) return;

 outportb(0x3ce,6);
 dat=inportb(0x3cf);
 dat=dat|4;
 dat=dat<<8;
 dat=dat|6;
 outport(0x3ce,dat);

 outportb(0xc4,0xb);
 dat=inport(0xc5);

 dat=vga_bank;
 dat=(dat<<8)^0x200;
 dat=dat|0xe;
 outport(0x3c4,dat);
```

```
 current_vga_bank=vga_bank;
}
/*****************************************/
/* cut palette form 256 to 128 color     */
/* by cut color at pixel lessness and    */
/* replace with near pixel color         */
/* modify : taget[] data                 */
/*          index at cnt_color_buf[]     */
/*  if cnt_color_buf[] == 0 then FREE    */
/*****************************************/
color128buf()
{
 find128Lbuf();
 fill128buf();
}
/*****************************************/
find128Lbuf()
{
 unsigned row,col;
 unsigned cnt,temp;
 char data;

 for(col=0;col<256;col++)          /* clear buffer color */
  cnt_color_buf[col]=0;
 for(row=0;row<scr_depth;row++)          /* accumulate data -> cnt_color_buf */
  for(col=0;col<scr_width;col++)
   cnt_color_buf[taget[row*scr_width+col]]++;
```

```
 for(data=0,col=0;col<256;col++)                /* count data */
  if(cnt_color_buf[col]==0) data++;

 for(row=0;row<256&&data<128;row++)    /* find less value 128 byte */
  if(cnt_color_buf[row]!=0)
  {
   temp=row;
   for(col=row+1;col<256;col++)
    if(cnt_color_buf[col]!=0)
     if(cnt_color_buf[temp]>cnt_color_buf[col])
      temp=col;
   cnt_color_buf[temp]=0;
   data++;

  }
}
/*****************************************/
fill128buf()
{
 unsigned row,col,temp;
 char data;

 /*--------allocate Buff----------*/
 buff= (char *) malloc((unsigned)64000);
 if (buff== NULL)
    terminate("Allocate failed buff\n");
 do
```

```
{
 memmove(buff,taget,64000);
 temp=0;
 for(row=0;row<scr_depth;row++)              /* near pixel to fill color */
  for(col=0;col<scr_width;col++)
   if(cnt_color_buf[buff[row*scr_width+col]]==0)
   {
    if(cnt_color_buf[data=buff[row*scr_width+col-1]]!=0)
     taget[row*scr_width+col]=data;
    else
    {
     if(cnt_color_buf[data=buff[row*scr_width+col+1]]!=0)
      taget[row*scr_width+col]=data;
     else
     {
      if(cnt_color_buf[data=buff[(row+1)*scr_width+col]]!=0)
   taget[row*scr_width+col]=data;
      else
      {
   if(cnt_color_buf[data=buff[(row-1)*scr_width+col]]!=0)
    taget[row*scr_width+col]=data;
else
{
     if(cnt_color_buf[data=buff[(row+1)*scr_width+col-1]]!=0)
      taget[row*scr_width+col]=data;
 else
 {
```

```
        if(cnt_color_buf[data=buff[(row+1)*scr_width+col+1]]!=0)
         taget[row*scr_width+col]=data;
    else
    {
         if(cnt_color_buf[data=buff[(row-1)*scr_width+col+1]]!=0)
taget[row*scr_width+col]=data;
     else
     {
          if(cnt_color_buf[data=buff[(row-1)*scr_width+col-1]]!=0)
 taget[row*scr_width+col]=data;
      else
       temp=1;
     }
    }
   }
}
          }
         }
        }
       }
     }
  while(temp==1);
  free(buff);
 }
```

```
/*****************************************/
/* cut palette form 256 to 128 color     */
/* by cut color at pixel lessness and    */
/* replace with near pixel color         */
/* modify : SCREEN display               */
/*          index at cnt_color_scr[]     */
/*  if cnt_color_scr[] == 0 then FREE    */
/*****************************************/
color128scr()
{
 find128Lscr();
 fill128scr();
}
/*****************************************/
find128Lscr()
{
 unsigned row,col;
 unsigned cnt,temp;
 char data;

 for(col=0;col<256;col++)          /* clear buffer color */
  cnt_color_scr[col]=0;
 for(row=0;row<scr_depth;row++) /* accumulate data -> cnt_color_scr */
  for(col=0;col<scr_width;col++)
   cnt_color_scr[peekb(0xa000,row*scr_width+col)]++;

 for(data=0,col=1;col<256;col++)                /* count data */
```

```
 if(cnt_color_scr[col]==0) data++;

for(row=0;row<256&&data<128;row++)    /* find less value 128 byte */
 if(cnt_color_scr[row]!=0)
 {
  temp=row;
  for(col=row+1;col<256;col++)
   if(cnt_color_scr[col]!=0)
    if(cnt_color_scr[temp]>cnt_color_scr[col])
     temp=col;
  cnt_color_scr[temp]=0;
  data++;
 }
}
/*****************************************/
fill128scr()
{
 unsigned row,col,temp;
 char data;

 /*--------allocate OBJ----------*/
 buff= (char *) malloc((unsigned)64000);
 if (buff== NULL)
    terminate("Allocate failed buff\n");
 do
 {
  memmove(buff,MK_FP(0xa000,0),64000);
```

```
  temp=0;
  for(row=0;row<scr_depth;row++)        /* near pixel to fill color */
   for(col=0;col<scr_width;col++)
    if(cnt_color_scr[buff[row*scr_width+col]]==0)
    {
     if(cnt_color_scr[data=buff[row*scr_width+col-1]]!=0)
      pokeb(0xa000,row*scr_width+col,data);
     else
     {
      if(cnt_color_scr[data=buff[row*scr_width+col+1]]!=0)
       pokeb(0xa000,row*scr_width+col,data);
      else
      {
       if(cnt_color_scr[data=buff[(row+1)*scr_width+col]]!=0)
pokeb(0xa000,row*scr_width+col,data);
       else
       {
if(cnt_color_scr[data=buff[(row-1)*scr_width+col]]!=0)
 pokeb(0xa000,row*scr_width+col,data);
else
{
 if(cnt_color_scr[data=buff[(row+1)*scr_width+col-1]]!=0)
  pokeb(0xa000,row*scr_width+col,data);
 else
 {
  if(cnt_color_scr[data=buff[(row+1)*scr_width+col+1]]!=0)
   pokeb(0xa000,row*scr_width+col,data);
```

```
 else
  {
   if(cnt_color_scr[data=buff[(row-1)*scr_width+col+1]]!=0)
    pokeb(0xa000,row*scr_width+col,data);
   else
   {
    if(cnt_color_scr[data=buff[(row-1)*scr_width+col-1]]!=0)
     pokeb(0xa000,row*scr_width+col,data);
    else
     temp=1;
   }
  }
 }
}
       }
      }
     }
    }
  }
 while(temp==1);
 free(buff); }
/****************************************/
/* transfered palette by compose        */
/* (palet_tab and fi.palette) and       */
/* modify image (taget)                 */
/* follow by cnt_color_scr space ZERO   */
/****************************************/
```

```
exchange_palette()
{
 char Pull_Byte_Mem();
 unsigned idx_buf,idx_scr,row,col,tmp;

 for(col=0;col<256;col++) /* clear point */
  point[col]=0;
 col=0;
 for(idx_buf=0,idx_scr=1;idx_scr<256&&idx_buf<256;)
 {
  while(idx_scr<256&&cnt_color_scr[idx_scr]!=0) /* find 0 */
   idx_scr++;
  while(idx_buf<256&&cnt_color_buf[idx_buf]==0) /* find !=0  */
   idx_buf++;
  if(idx_scr<256&&idx_buf<256)
  {
   palet_tab[idx_scr]=fi.palette[idx_buf];
   palet_tab[idx_scr+256]=fi.palette[idx_buf+256];
   palet_tab[idx_scr+256*2]=fi.palette[idx_buf+256*2];

   point[idx_buf]=idx_scr;
   idx_scr++;
   idx_buf++;
   col++;
   tmp=idx_scr;
  }
 }
```

```
 for(idx_scr=tmp,idx_buf=1;col<127&&idx_buf<256;idx_buf++)
  if(point[idx_buf]==0)
  {
   while(idx_scr<256&&cnt_color_scr[idx_scr]!=0) /* find 0 */
    idx_scr++;
   point[idx_buf]=idx_scr; /* fill rest full 128 byte with 0xff */
   col++;
   idx_scr++;
  }

 cnt_buff=0;buff_org=taget;
 for(row=0;row<scr_depth;row++)
  for(col=0;col<scr_width;col++)
   if(point[Pull_Byte_Mem()]!=0)
    Put_Byte_Mem(point[Pull_Byte_Mem()]);
   else
    cnt_buff++;
}
/****************************************/
/* transfered palette to 256 color on   */
/* on buffer -> screen (same picture)   */
/* modify : taget[] data                */
/*          and pallette                */
/*          index at cnt_color_buf[]    */
/*  if cnt_color_scr[] == 0 then FREE   */
/****************************************/
```

```
tranto256scr()
{
 char Pull_Byte_Mem();
 unsigned idx_buf,idx_scr,row,col;

 col=0;
 for(idx_buf=1,idx_scr=0;idx_scr<256&&idx_buf<256;)
 {
  while(idx_scr<256&&cnt_color_scr[idx_scr]==0) /* find !=0 */
    idx_scr++;
  while(idx_buf<256&&point[idx_buf]!=0) /* find 0 */
    idx_buf++;
  if(idx_scr<256&&idx_buf<256)
  {
   palet_tab[idx_scr]=fi.palette[idx_buf];
   palet_tab[idx_scr+256]=fi.palette[idx_buf+256];
   palet_tab[idx_scr+256*2]=fi.palette[idx_buf+256*2];

   point[idx_buf]=idx_scr;
   idx_scr++;
   idx_buf++;
   col++;
  }
 }
 for(row=2,idx_scr=point[1];row<255;row++) /* find MAX value */
  if(point[row]>idx_scr)
     idx_scr=point[row];
```

```
 idx_scr++;

 for(;col<128&&idx_buf<256;idx_buf++)
  if(point[idx_buf]==0)
  {
   palet_tab[idx_scr]=fi.palette[idx_buf];
   palet_tab[idx_scr+256]=fi.palette[idx_buf+256];
   palet_tab[idx_scr+256*2]=fi.palette[idx_buf+256*2];

   point[idx_buf]=idx_scr;   /* fill rest full 128 byte with 0xff */
   col++;
   idx_scr++;
  }

 cnt_buff=0;buff_org=taget;
 for(row=0;row<scr_depth;row++)
  for(col=0;col<scr_width;col++)
    Put_Byte_Mem(point[Pull_Byte_Mem()]);
 memmove(fi.palette,palet_tab,256*3);
}
/****************************************/
/****************************************/
/*      set color palette               */
/****************************************/
/* set the VGA palette and background */
colorpalet(signed char light)
 {
```

```
  int i,j;
  union REGS r;

  outp(0x3c6,0xff);
  for(i=0;i<256;++i)
  {
   outp(0x3c8,i);
   for(j=0;j<3;j++)
    if (palet_tab[j*256+i] > light)
     outp(0x3c9,palet_tab[j*256+i] - light);
    else
     outp(0x3c9,0);
  }
  r.x.ax=0x1001;
  r.h.bh=0;       /* page */
  int86(0x10,&r,&r);
}
/*****************************************/
/* set the VGA palette and background BLACK -> WHITE */
colorpalet_w(signed char light)
 {
  int i,j;
  union REGS r;

  outp(0x3c6,0xff);
  for(i=0;i<256;++i)
  {
```

```
    outp(0x3c8,i);
    for(j=0;j<3;j++)
     if ((palet_tab[j*256+i] + light)<64)
      outp(0x3c9,palet_tab[j*256+i] + light);
     else
      outp(0x3c9,63);
   }
   r.x.ax=0x1001;
   r.h.bh=0;       /* page */
   int86(0x10,&r,&r);
}
/*****************************************/
dark_screen()
{
 colorpalet(100);
 if (flag_alloc==0)
  memset(MK_FP(0xa000,0),0,64000);
 else
 {
  memset(MK_FP(0xa000,0),0,65535);
  bank_sel(1);
  memset(MK_FP(0xa000,0),0,65535);
  bank_sel(2);
  memset(MK_FP(0xa000,0),0,65535);
  bank_sel(3);
  memset(MK_FP(0xa000,0),0,65535);
  bank_sel(4);
```

```
  memset(MK_FP(0xa000,0),0,45060);
  bank_sel(0);
 }
}
/*****************************************/
```

```
**********************

        STEP_C.C

**********************


#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <graphics.h>
#include "info.h"

extern char *pic_name;
extern char palet_tab[768];          /* palette of screen image */
extern unsigned length_fname;
extern unsigned ptr_fname;  /* pointer on current image (taget) */
extern FILEINFO fi;
/*****************************************************/
step_com()
{
 unsigned cnt,cntptr;
 int n=0;
 char flag;

 char *cmd[] = {
"CUT",          /*1 Display image                          */
"DARK_DOT_IN",  /*2 Random black dot to dark screen        */
```

```
"LIGHT_DOT_IN", /*3 Random white dot to light screen        */
"PIC_DOT",      /*4 Random dot with another image          */
"DARK_IN",      /*5 Become dark screen                     */
"LIGHT_IN",     /*6 Become light screen                    */
"FEED_UP",      /*7 Feed up and another image come in      */
"FEED_DOWN",    /*8 Feed down and another image come in    */
"CUT_DARK",     /*9 Cut image to dark screen               */
"DARK_DOT_OUT", /*10 Random black dot image form dark screen */
"LIGHT_DOT_OUT",/*11 Random white dot image form light screen*/
"DARK_OUT",     /*12 Screen become Image form dark screen   */
"LIGHT_OUT",    /*13 Screen become Image form light screen  */
"WIN",
"WIPE",
"SNAKE",
"SPIRAL",
"SQUARE"
      };
 cntptr=0;
 while(cntptr<length_fname)
 {
  flag=1;
  for(cnt=0;cnt<19&&flag!=0;cnt++)
   flag=strcmp(cmd[cnt],pic_name+cntptr);
  if(flag!=0)
   {      if(n==0){
          ptr_fname=cntptr;
  unpack(pic_name+cntptr);
```

```
cntptr=cntptr+strlen(pic_name+cntptr);
}
else if(n==1) {
getch();
cntptr=cntptr+strlen(pic_name+cntptr);
cntptr++;
flag=1;
for(cnt=0;cnt<19&&flag!=0;cnt++)
 flag=strcmp(cmd[cnt],pic_name+cntptr);
if(flag!=0)
 cntptr--;
n=0;
}
  }

switch(cnt)
{
 case 1:memmove(palet_tab,fi.palette,256*3); /* CUT */
cut_in();n++;
        cntptr=cntptr+strlen(pic_name+cntptr);
break;                    /*   cut   */
 case 2:memmove(palet_tab,fi.palette,256*3);   /* DARK_DOT_IN */
fad_in_d();n++;
        cntptr=cntptr+strlen(pic_name+cntptr);
break;                    /* fad(dot black)*/
 case 3:memmove(palet_tab,fi.palette,256*3);   /* LIGHT_DOT_IN */
fad_in_dw();n++;
```

```
        cntptr=cntptr+strlen(pic_name+cntptr);
break;                  /* fad(dot white)*/
 case 4:fad_out_d_over();n++;     /* PIC_DOT */
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
 case 5:memmove(palet_tab,fi.palette,256*3);          /* DARK_IN */
 fad_in_l(atoi(pic_name+cntptr));n++;
 cntptr=cntptr+strlen(pic_name+cntptr);
 break;                 /* fad(to black)*/
 case 6:memmove(palet_tab,fi.palette,256*3);          /* LIGHT_IN */
 fad_in_lw(atoi(pic_name+cntptr));n++;
 cntptr=cntptr+strlen(pic_name+cntptr);
    break;              /* fad (to white) */
 case 7:feed_up_in_out(5);n++;       /* FEED_UP */
 cntptr=cntptr+strlen(pic_name+cntptr);
    break;
 case 8:feed_down_in_out(5);n++;     /* FEED_DOWN */
 cntptr=cntptr+strlen(pic_name+cntptr);
    break;
 case 9:cut_out();n++;
 cntptr=cntptr+strlen(pic_name+cntptr);
 n=0;
 break;    /*CUT_DARK*/                          /* CUT_DARK */
 case 10:fad_out_d();n++;
 cntptr=cntptr+strlen(pic_name+cntptr);
 n=0;
 break; /*DARK_DOT_OUT*/                         /* DARK_DOT_OUT */
```

```
case 11:fad_out_dw();n++;
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
break;                                          /* LIGHT_DOT_OUT */
case 12:fad_out_1(5);                           /* DARK_OUT */
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
   break;
case 13:fad_out_lw(5);                          /* LIGHT_OUT */
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
   break;
case 14:win();n++;                              /*WINDOW*/
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
case 15:wipe();n++;                             /*WIPE*/
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
case 16:snake();n++;                            /*SNAKE*/
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
case 17:spiral();n++;                           /*SPIRAL*/
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
case 18:square();n++;                           /*SQUARE*/
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
```

```
  }
  cntptr++;
 }
}
/*****************************************/
```

```
**********

TGA.C

**********

#include <stdio.h>
#include <alloc.h>
#include <dos.h>
#include "head.h"
#include "info.h"

typedef struct {
char identsize;
char colourmaptype;
char imagetype;
unsigned int colourmapstart;
unsigned int colourmaplength;
char colourmapbits;
unsigned int xstart;
unsigned int ystart;
unsigned int width,depth;
char bits;
char descriptor;
} TGAHEAD;

extern FILE *fp;
extern FILEINFO fi;
/*****************************************/
```

```
unpack_tga(char name[])
{
char path[81];
int r;

strmfe(path,name,"TGA");
strupr(path);
if((fp = fopen(path,"rb")) != NULL) {
unpacktga(fp,&fi);
fclose(fp);
} else printf("Error opening %s",path);
}

/* unpack a TGA file */
unpacktga(fp,fi)
FILE *fp;
FILEINFO *fi;
{
TGAHEAD tga;
char *p,*pr;
int i,n,startline,endline,incline;
int row;

/* set a default monochrome palette */
memcpy(fi->palette,"\000\000\000\377\377\377",6);

/* get the header */
```

```
if(fread((char *)&tga,1,sizeof(TGAHEAD),fp)==sizeof(TGAHEAD)) {

if(tga.imagetype==0x01 ||
   tga.imagetype==0x02 ||
   tga.imagetype==0x03 ||
   tga.imagetype==0x09 ||
   tga.imagetype==0x0a ||
   tga.imagetype==0x0b) {

/* set the details */
fi->width=tga.width;
fi->depth=tga.depth;
fi->bits=tga.bits;

/* work out the line width */
if(fi->bits==1) fi->bytes=pixels2bytes(fi->width);
else if(fi->bits==8) fi->bytes=fi->width;
else fi->bytes=fi->width*3;

fseek(fp,(long)tga.identsize,SEEK_CUR);

/* get the palette */
if(tga.colourmaptype) {
switch(tga.colourmapbits) {
case 24:

      for(row=0;row<256;++row)
```

```
{
fi->palette[row+2*256]=getc(fp)>>2;
fi->palette[row+1*256]=getc(fp)>>2;
fi->palette[row+0*256]=getc(fp)>>2;
}

break;
case 16:
for(i=tga.colourmapstart;i<tga.colourmaplength;++i) {
if(i >= 768) break;
n=fgetword(fp);
fi->palette[i*RGB_SIZE+RGB_BLUE]=((n >> 10) & 0x1f) << 3;
fi->palette[i*RGB_SIZE+RGB_GREEN]=((n >> 5) & 0x1f) << 3;
fi->palette[i*RGB_SIZE+RGB_RED]=(n & 0x1f) << 3;
}
break;
}
}

if(fi->bits > 8) {
for(i=0;i<256;++i)
    memset(fi->palette+(i*RGB_SIZE),i,RGB_SIZE);
}
else if(fi->bits==1) memcpy(fi->palette,"\000\000\000\377\377\377",6);

/* allocate a line buffer */
if((p=malloc(fi->bytes)) != NULL) {
```

```
if(!(tga.descriptor & 0x20)) {
startline=fi->depth-1;
endline=-1;
incline=-1;
}
else {
startline=0;
endline=fi->depth;
incline=1;
}

/* read all the lines */
for(i=startline;i != endline;i+=incline) {
if(readtgaline(p,fp,&tga)) {
free(p);
return(BAD_READ);
}

/* translate the line types into VGA */
switch(fi->bits) {
case 8:
reverse(p,&tga);
putline(p,i);
break;
}
}
free(p);
```

```
return(GOOD_READ);
} else return(MEMORY_ERROR);
} else return(BAD_FILE);
} else return(BAD_READ);
}

/* reverse one line left for right needs be */
reverse(p,tga)
char *p;
TGAHEAD *tga;
{
char *pr;
int i;

if(!(tga->descriptor & 0x10)) return;
if((pr=malloc(tga->width)) != NULL) {
for(i=0;i<tga->width;++i) pr[i]=p[tga->width-1-i];
memcpy(p,pr,tga->width);
free(pr);
}
}

/* read one line in the appropriate mode */
readtgaline(p,fp,tga)
char *p;
FILE *fp;
TGAHEAD *tga;
```

```
{
int c,i,n=0,size;
int r,g,b,linesize;

if(tga->bits==1) linesize=pixels2bytes(tga->width);
else linesize=tga->width;

/* handle uncompressed lines */
if(tga->imagetype==0x01 ||
   tga->imagetype==0x02 ||
   tga->imagetype==0x03) {
switch(tga->bits) {
case 1:
fread(p,1,pixels2bytes(tga->width),fp);
break;
case 8:
fread(p,1,tga->width,fp);
break;
case 16:
for(i=0;i<tga->width;++i) {
c=fgetword(fp);
r=((c >> 10) & 0x1f) << 3;
g=((c >> 5) & 0x1f) << 3;
b=(c & 0x1f) << 3;
*p++=r;
*p++=g;
*p++=b;
```

```
}
break;
case 24:
for(i=0;i<tga->width;++i) {
b=fgetc(fp);
g=fgetc(fp);
r=fgetc(fp);
*p++=r;
*p++=g;
*p++=b;
}
break;
case 32:
for(i=0;i<tga->width;++i) {
b=fgetc(fp);
g=fgetc(fp);
r=fgetc(fp);
fgetc(fp);
*p++=r;
*p++=g;
*p++=b;
}
break;
}
}

/* handle compressed lines */
```

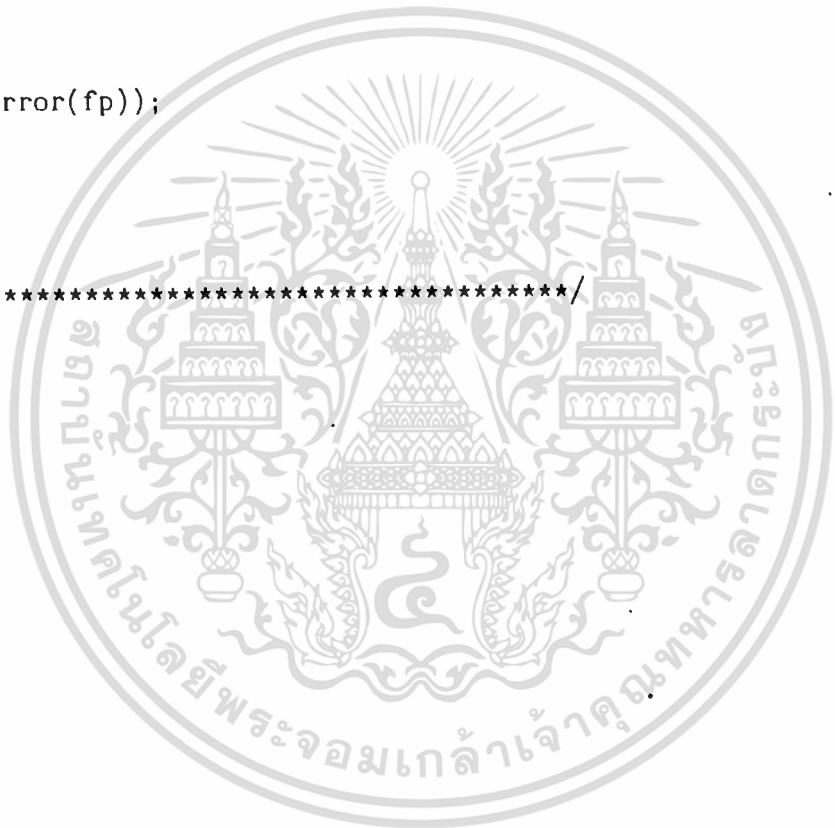
```
else {
do {
c=fgetc(fp);
size=(c & 0x7f)+1;
n+=size;
if(c & 0x80) {
switch(tga->bits) {
case 1:
case 8:
c=fgetc(fp);
for(i=0;i<size;++i) *p++=c;
break;
case 16:
c=fgetword(fp);
r=((c >> 10) & 0x1f) << 3;
g=((c >> 5) & 0x1f) << 3;
b=(c & 0x1f) << 3;
for(i=0;i<size;++i) {
*p++=r;
*p++=g;
*p++=b;
}
break;
case 24:
b=fgetc(fp);
g=fgetc(fp);
r=fgetc(fp);
```

```
c=fgetword(fp);
r=((c >> 10) & 0x1f) << 3;
g=((c >> 5) & 0x1f) << 3;
b=(c & 0x1f) << 3;
*p++=r;
*p++=g;
*p++=b;
}
break;
case 24:
for(i=0;i<size;++i) {
b=fgetc(fp);
g=fgetc(fp);
r=fgetc(fp);
*p++=r;
*p++=g;
*p++=b;
}
break;
case 32:
for(i=0;i<size;++i) {
b=fgetc(fp);
g=fgetc(fp);
r=fgetc(fp);
fgetc(fp);
*p++=r;
*p++=g;
```

```
*p++=b;
}
break;
}
}
} while(n < linesize);
}

return(ferror(fp));
}

/*****************************************/
```



```
*********************

      SAVE.C

*********************

#include <conio.h>
#include <stdio.h>
#include <alloc.h>
#include <dir.h>
#include <graphics.h>

/*****************************************************/
void save()
{
extern char *taget,*pic_name;
extern FILE *fp;
extern char *pic_old;
extern char *sel;
extern unsigned char readin,subreadin;
extern unsigned char str[500][70];
extern char name[20];
extern int aa,a,b,xx,x,xy,y,yy,xx2;
extern submenu1(); extern shadow();
int i,x1;
char k,k1;
if(str[0][0]!=0x20){
TextWindow(25,10,55,12,14,3);
textbackground(4);
```

```
gotoxy(37,10);
cprintf("Save");
textbackground(3);
textcolor(14);
gotoxy(28,11);
printf("File name : ");
i=0;x1=40;
do{
k=getch();
if(k==0) k1=getch();
if(k>0x20){
 name[i]=k;
 gotoxy(x1,11);
 printf("%c",k);
 x1++;
 i++;
 if(i>=20){
  x1=59;
  i=20;
 }
}
else{
switch(k){
 case 0:switch(k1){
  case 0x4b:
   printf("%c %c",8,8);
   x1--;
```

```
    i--;
    name[i]=0x20;
    if(i<=0){
     i=0;x1=40;
    }
    gotoxy(x1,11);
   break;
   default:break;
   }
  break;
  case 0x08:
   printf("%c %c",8,8);
   i--;x1--;
   name[i]=0x20;
   if(i<=0){
    i=0;x1=40;
   }
   gotoxy(x1,11);
   break;
  case 0x1b:
   submenu1();
   shadow();
   break;
  case 0x0d:
    name[i]=0;
    sel=malloc(i);
    sel=&name;
```

```
 strcat(sel,".pot");
 if((fp=fopen(sel,"w"))!=NULL){
 if(xx==0 || xx>=44){
 if(str[xy+1][0]==0x20)
 if(str[xy][xx]<=0x20)
 str[xy][xx]=0x00;
 }
 for(a=0;a<=yy;a++)
 for(b=0;b<70;b++){
 readin=str[a][b];
 fputc(readin,fp);
 } }
 else {
 gotoxy(28,11);
 printf("Cannot run file");
 }
 fclose(fp);
 submenu1();
 shadow();
break;
default:break;
 }/* end of _switch */
}
}while(k!=0x1b&&k!=0x0d);
]
}
/*******************************************************/
```

****

INFO.H

****

```
#define GOOD_READ 0/* return codes */
typedef struct
{
 int background;
 int flags;
 char subtype[4];
 unsigned width,depth;
 unsigned bytes;
 char bits;
 char plane;
 unsigned int rowsperstrip;
 unsigned int count;
 unsigned int samples;
 unsigned int planarconfig;
 unsigned int compression;
 unsigned int bitspersample;
 unsigned long bytecount;
 unsigned long offset;
 char palette[768];
} FILEINFO;
/*FILEINFO fi;*/
```

```
**********

HEAD.H

**********

#define GOOD_READ 0/* return codes */
#define BAD_FILE 1
#define BAD_READ 2
#define MEMORY_ERROR 3
#define BAD_CODE 4
#define BAD_FIRSTCODE 5
#define BAD_ALLOC 6
#define BAD_SYMBOLSIZE  7
#define SCREENWIDE 320     /* mode 13 screen dimensions */
#define SCREENDEEP 200
#define STEP 32      /* size of a step when panning */
#define HOME 0x4700/* cursor control codes */
#define CURSOR_UP 0x4800
#define CURSOR_LEFT 0x4b00
#define CURSOR_RIGHT 0x4d00
#define END 0x4f00
#define CURSOR_DOWN 0x5000
#define RGB_RED 0
#define RGB_GREEN 1
#define RGB_BLUE 2
#define RGB_SIZE 3
#define pixels2bytes (n)((n+7)/8)
#define greyvalue(r,g,b) (((r*30)/100) + ((g*59)/100) + ((b*11)/100))
```

****

SAV.C

****

```
#include <conio.h>
#include <stdio.h>
#include <dir.h>
extern char *sel;
extern char readin;
extern unsigned char str[500][70];
extern char name[20];
extern int aa,a,b,xx,x,xy,y,yy,xx1,xx2,page,ly;
extern struct ffblk blk;
/*****************************************************/
void load()
{
extern submenu1();
FILE *fin;
   for(a=0;a<=yy;a++)
    for(b=0;b<70;b++)
     str[a][b]=0x20;
   fin=fopen(sel,"r");
   if(fin==NULL){
    textcolor(14);
    gotoxy(28,11);
    printf("File not found");
    fclose(fin);
```

```
 getch();
 submenu1();
}
else{
 a=0;b=0;
 readin=fgetc(fin);
 if(readin!=0x20){
 str[a][b]=readin;
 b=1;
 do{
 readin=fgetc(fin);
 str[a][b]=readin;
 b++;
 if(b==70){
 a++;b=0;
 }
 }while(readin!=0x00);
xx=b-1;yy=a;xy=yy;
if(xy>12) xy=12;
}
fclose(fin);
submenu1();
textcolor(15);
if(yy<13){
for(a=0;a<=yy;a++)
 for(b=0;b<70;b++){
    gotoxy(x+b,y+a);
```

```
      printf("%c",str[a][b]);
    }
   ly=yy;
   }
   else{
   for(a=0;a<=xy;a++)
    for(b=0;b<70;b++){
      gotoxy(x+b,y+a);
      printf("%c",str[a][b]);
    }
   ly=xy;
   }
   v_line();
}
}
/*****************************************************/
void select()
{
extern submenu1();
int round,i,ch,ch1,x1,y1;
x1=12;y1=10;round=0;
textbackground(2);
textcolor(15);
findfirst("*.pot",&blk,0);
gotoxy(x1,y1);
cprintf("%s",blk.ff_name);
sel=blk.ff_name;
```

```
gotoxy(x+xx,y+xy);
do{
ch=getch();
if(ch==0) ch1=getch();
switch(ch){
 case 0:switch(ch1){
  case 0x4d:
   if(a>2){
   textbackground(3);
   textcolor(1);
   gotoxy(x1,y1);
   cprintf("%s",sel);
   x1+=15;
   if(x1>60){
    y1++;x1=12;
    if(y1>=17) y1=16;
   }
   findnext(&blk);
   textbackground(2);
   textcolor(15);
   gotoxy(x1,y1);
   cprintf("%s",blk.ff_name);
   round++;
   a--;
   }
  break;
  case 0x4b:
```

```
 textbackground(3);
 textcolor(1);
 gotoxy(x1,y1);
 cprintf("%s",sel);
 findfirst("*.pot",&blk,0);

 for(i=1;i<round;i++)
  findnext(&blk);
 x1-=15;
 if(x1<12){
  y1--;x1=57;
  if(y1<10){
   y1=10;x1=12;round++;a--;
   }
 }
  textbackground(2);
  textcolor(15);
  gotoxy(x1,y1);
  cprintf("%s",blk.ff_name);
  round--;a++;
 break;
}
default:break;
}
textcolor(3);
gotoxy(x+xx,y+xy);
}while(ch1=0x1b&&ch1=0x0d);
```

```
 if(ch==0x0d)
  load();
 else{
  submenu1();
  shadow();
 }
}
/*******************************************************/
void dir()
{
int ch,c,done,x1,y1;
x1=12;y1=10;a=1;
TextWindow(10,8,70,18,14,3);
textbackground(4);
gotoxy(35,8);
cprintf("<Directory>");
done=findfirst("*.pot",&blk,0);
textbackground(3);
textcolor(1);
while(!done){
 gotoxy(x1,y1);
 cprintf("%s",blk.ff_name);
 x1+=15;
 if(x1>60){
  y1++;
  x1=12;
  if(y1>=17){
```

```
 yl=16;
  ch=getch();
  if(ch==0) c=getch();
  switch(ch){
   case 0:switch(c){
    case 0x50:
     movetext(10,10,70,17,10,9);
     break;
    case 0x48:
     break;
    default:break;
   }
  break;
  default:break;
  }
 }
 done=findnext(&blk);
 a++;
}
textcolor(3);
gotoxy(x+xx,y+xy);
}
/*******************************************************/
shadow()
{
if(yy<13)
```

```
 xx2=yy;
if(ly>yy)
 xx2=12+(page-(ly-yy));
if(ly>=0 && ly<=yy)
 xx2=12+page;
textcolor(15);
for(a=page;a<=xx2;a++)
 for(b=0;b<70;b++){
  readin=str[a][b];
  gotoxy(x+b,y+a-page);
  cprintf("%c",readin);
 }
 v_line();
}
/*******************************************************/
v_line()
{
vline(25,5,20);
vline(26,5,20);
vline(46,5,20);
vline(47,5,20);
vline(57,5,20);
vline(58,5,20);
}
/*******************************************************/
```

```
**********

AUTO_C.C

**********

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <graphics.h>
#include "info.h"

extern char *pic_name;
extern char palet_tab[768];          /* palette of screen image */
extern unsigned length_fname;
extern unsigned ptr_fname;  /* pointer on current image (taget) */
extern FILEINFO fi;
/*****************************************************/
auto_com()
{
 unsigned cnt,cntptr;
 int time_cnt,time_base,time_tmp;
 int n=0;
 char flag;

 char *cmd[] = {
"CUT",          /*1 Display image                          */
```

```
"DARK_DOT_IN",  /*2 Random black dot to dark screen          */
"LIGHT_DOT_IN", /*3 Random white dot to light screen        */
"PIC_DOT",      /*4 Random dot with another image           */
"DARK_IN",      /*5 Become dark screen                      */
"LIGHT_IN",     /*6 Become light screen                     */
"FEED_UP",      /*7 Feed up and another image come in       */
"FEED_DOWN",    /*8 Feed down and another image come in     */
"CUT_DARK",     /*9 Cut image to dark screen                */
"DARK_DOT_OUT", /*10 Random black dot image form dark screen */
"LIGHT_DOT_OUT",/*11 Random white dot image form light screen*/
"DARK_OUT",     /*12 Screen become Image form dark screen   */
"LIGHT_OUT",     /*13 Screen become Image form light screen  */
"WIN",
"WIPE",
"SNAKE",
"SPIRAL",
"SQUARE"
      };
 struct time basetime;
 gettime(&basetime);
 time_base=basetime.ti_hour*60+basetime.ti_min*60+basetime.ti_sec;
 time_cnt=0;
 cntptr=0;
 while(cntptr<length_fname)
 {
  flag=1;
  for(cnt=0;cnt<19&&flag!=0;cnt++)
```

```
flag=strcmp(cmd[cnt],pic_name+cntptr);
if(flag!=0)
 {      if(n==0){
        ptr_fname=cntptr;
unpack(pic_name+cntptr);
cntptr=cntptr+strlen(pic_name+cntptr);
}
else if(n==1) {
gettime(&basetime);
time_tmp=basetime.ti_hour*60+basetime.ti_min*60+basetime.ti_sec;
time_cnt=time_cnt+atoi(pic_name+cntptr);
time_tmp=time_tmp-time_base;
if((time_cnt-time_tmp) > 0)
 delay((time_cnt-time_tmp)*1000);
cntptr=cntptr+strlen(pic_name+cntptr);
cntptr++;
flag=1;
for(cnt=0;cnt<19&&flag!=0;cnt++)
 flag=strcmp(cmd[cnt],pic_name+cntptr);
if(flag!=0)
 cntptr--;
n=0;
 }
   }

 switch(cnt)
 {
```

```
 case 1:memmove(palet_tab,fi.palette,256*3); /* CUT */
cut_in();n++;
        cntptr=cntptr+strlen(pic_name+cntptr);
break;                    /*   cut   */
 case 2:memmove(palet_tab,fi.palette,256*3);   /* DARK_DOT_IN */
fad_in_d();n++;
        cntptr=cntptr+strlen(pic_name+cntptr);
break;                    /* fad(dot black)*/
 case 3:memmove(palet_tab,fi.palette,256*3);   /* LIGHT_DOT_IN */
fad_in_dw();n++;
        cntptr=cntptr+strlen(pic_name+cntptr);
break;                    /* fad(dot white)*/
 case 4:fad_out_d_over();n++;     /* PIC_DOT */
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
 case 5:memmove(palet_tab,fi.palette,256*3);        /* DARK_IN */
 fad_in_l(atoi(pic_name+cntptr));n++;
 cntptr=cntptr+strlen(pic_name+cntptr);
 break;                   /* fad(to black)*/
 case 6:memmove(palet_tab,fi.palette,256*3);        /* LIGHT_IN */
 fad_in_lw(atoi(pic_name+cntptr));n++;
 cntptr=cntptr+strlen(pic_name+cntptr);
    break;                /* fad (to white) */
 case 7:feed_up_in_out(5);n++;       /* FEED_UP */
 cntptr=cntptr+strlen(pic_name+cntptr);
    break;
 case 8:feed_down_in_out(5);n++;     /* FEED_DOWN */
```

```
cntptr=cntptr+strlen(pic_name+cntptr);
   break;
case 9:cut_out();n++;
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
break;    /*CUT_DARK*/                          /* CUT_DARK */
case 10:fad_out_d();n++;
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
break; /*DARK_DOT_OUT*/                        /* DARK_DOT_OUT */
case 11:fad_out_dw();n++;
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
break;                                        /* LIGHT_DOT_OUT */
case 12:fad_out_l(5);                          /* DARK_OUT */
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
   break;
case 13:fad_out_lw(5);                        /* LIGHT_OUT */
cntptr=cntptr+strlen(pic_name+cntptr);
n=0;
   break;
case 14:win();n++;                                  /*WINDOW*/
        cntptr=cntptr+strlen(pic_name+cntptr);
break;
case 15:wipe();n++;                                  /*WIPE*/
        cntptr=cntptr+strlen(pic_name+cntptr);
```

```
   break;
   case 16:snake();n++;                                    /*SNAKE*/
           cntptr=cntptr+strlen(pic_name+cntptr);
   break;
   case 17:spiral();n++;                                   /*SPIRAL*/
           cntptr=cntptr+strlen(pic_name+cntptr);
   break;
   case 18:square();n++;                                   /*SQUARE*/
           cntptr=cntptr+strlen(pic_name+cntptr);
   break;
  }
  cntptr++;
 }
}
/*********************************************************/
```

```
**********

KEYI.C

**********

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <graphics.h>
#include <dir.h>

extern char *taget;
extern char *pic_name;
extern FILE *fp;
extern char *sel;
int INS=1;
extern unsigned char readin,subreadin;
extern unsigned char str[500][70];
extern char name[20];
extern struct ffblk blk;
extern int aa,a,b,x,y,xy,xx,yy,ly;
int xx1,page,xx2;
/*****************************************************/
void keyin(int length)
{
extern submenu1();
extern void ins(int length);
```

```
extern void v_line();
extern void save();
extern void select();
extern void shadow();
extern void dir();
extern void load();
extern void run();
 int k,slong,g;

 x=5;y=6;xy=0;xx=0;yy=0;g=0;
 page=1;xx1=0;xx2=13;
 for(a=0;a<500;a++)
  for(b=0;b<70;b++)
   str[a][b]=0x20;
 gotoxy(32,2);
 printf("NONAME.POT");
 do{
  textcolor(14);
  gotoxy(5,2);
  printf("Line ");
  printf("     ");
  gotoxy(10,2);
  printf("%d",ly);
  gotoxy(15,2);
  printf("Col ");
  printf("    ");
  gotoxy(19,2);
```

```
 printf("%d",xx);
 if(INS==1){
 gotoxy(23,2);
 printf("Insert");
 }
 else{
 gotoxy(23,2);
 printf("      ");
 }
 textcolor(15);
 gotoxy(x+xx,y+xy);
 readin=getch();
 if(readin<=0x20)
 {
 if(!readin) subreadin=getch();
  switch(readin)
  {
   case 0x00:switch(subreadin)
     {
     case 0x4b:     /* ok */
if(xx>0 && xx<19)
{
 xx--;
 if(xx<=0) xx=0;
}
else if(xx>22 && xx<41)
{
```

```
 xx--;
 if(xx==22) xx=18;
}
else if(xx>43 && xx<52)
{
 xx--;
 if(xx==43) xx=39;
}
else if(xx>54 && xx<71)
{
 xx--;
 if(xx==54) xx=50;
}
break;
      case 0x4d:          /* ok */
xx++;
if(xx==71){
xx=0;xy++;yy++;
}
if(xx==19) xx=23;
if(xx==40) xx=44;
if(xx==51) xx=55;
break;
      case 0x48:if(xy>0)    /* ok */
xy--;
ly--;
if(ly<=0) ly=0;
```

```
break;
      case 0x50:if(xy<12)     /* ok */
xy++;
ly++;
     if(xy>=13)
     {
 movetext(5,6,23,19,5,5);
 movetext(28,6,44,19,28,5);
 movetext(49,6,55,19,49,5);
 movetext(60,6,76,19,60,5);
 xy=12;
     }
     if(ly==13*page){
      page++;
      xx1=xx2;
      xx2=xx2*page;
     }
 break;
       case 0x52:
        INS=!INS;
        break;
       case 0x53:          /* ok */
 if(str[xy][xx]<=0x00){
  str[xy][xx]=0x20;
  gotoxy(30,2);
  printf("*");
 }
```

```
if(xx>=0 && xx<19){
 movetext(x+xx+1,y+xy,x+19,y+xy,x+xx,y+xy);
 for(a=xy;a<=xy;a++)
  for(b=xx;b<19;b++)
  str[a][b]=str[a][b+1];
}
if(xx>22 && xx<41){
 movetext(x+xx+1,y+xy,x+40,y+xy,x+xx,y+xy);
 for(a=xy;a<=xy;a++)
  for(b=xx;b<41;b++)
  str[a][b]=str[a][b+1];
}
if(xx>43 && xx<52){
 movetext(x+xx+1,y+xy,x+50,y+xy,x+xx,y+xy);
 for(a=xy;a<=xy;a++)
  for(b=xx;b<52;b++)
  str[a][b]=str[a][b+1];
}
if(xx>54 && xx<71){
 movetext(x+xx+1,y+xy,x+70,y+xy,x+xx,y+xy);
 for(a=xy;a<=xy;a++)
  for(b=xx;b<69;b++)
  str[a][b]=str[a][b+1];
}
v_line();
break;
      case 0x3b:          /* ok */
```

```
help();
submenu1();
if(g!=0){
 gotoxy(30,2);
 printf("*");
 }
 else{
                          gotoxy(30,2);
  printf(" ");
 }
 gotoxy(32,2);
 printf("%s",sel);
 textcolor(15);
 shadow();
 break;
       case 0x3c:          /* ok */
 xx2=yy;
 save();g=1;
 gotoxy(30,2);
 printf(" ");
 gotoxy(32,2);
 printf("%s",sel);
 break;
        case 0x3d:
 run();
 gotoxy(32,2);
 printf("%s",sel);
```

```
break;
      case 0x3e:
if(sel!=NULL){
taget= (char *) malloc((unsigned)65535);
if(taget != NULL)
{
read_name(sel);
gphmode(19);
step_com();
txtmode();
free(pic_name);
free(taget);
}
submenu1();
gotoxy(32,2);
printf("%s",sel);
shadow();
}
break;
      case 0x43:
sel=0;
submenu1();
gotoxy(32,2);
printf("NONAME.POT");
for(a=0;a<=yy;a++)
 for(b=0;b<70;b++)
  str[a][b]=0x20;
```

```
xx=0;xy=0;yy=0;
page=1;xx1=0;xx2=13;
break;
      case 0x44:
dir();
select();
gotoxy(32,2);
printf("%s",sel);
break;
      default:break;
      }
    break;
    case 0x1b:                    /* ok */
 TextWindow(26,11,56,13,7,7);
 TextWindow(25,10,55,12,14,3);
 textcolor(14);
 gotoxy(30,11);
 printf("EXIT to dos (Y/N) : ");
 k=getch();
 if(toupper(k)=='Y') {
  fclose(fp);
  terminate("Good bye... \n");
 }
 else{
 submenu1();
 if(g!=0){
  gotoxy(30,2);
```

```
 printf("*");
}
else{
 gotoxy(30,2);
 printf(" ");
 }
 if(sel==0){
      gotoxy(32,2);
 printf("NONAME.POT");
 }
 else{
 gotoxy(32,2);
 printf("%s",sel);
 }
 shadow();
 }
 break;
     case 0x08:                    /* ok */
 if(str[xy][xx]<=0x00){
  str[xy][xx]=0x20;
  gotoxy(30,2);
  printf("*");g=0;
  }
  if(xx>0)
  {
   printf("%c %c",8,8);
   if(xx>0 && xx<19)
```

```
{
 movetext(x+xx,y+xy,x+19,y+xy,x+xx-1,y+xy);
 xx--;
 if(xx<=0) xx=0;
 for(a=xy;a<=xy;a++)
  for(b=xx;b<19;b++)
   str[a][b]=str[a][b+1];
}
else if(xx>22 && xx<41)
{
 movetext(x+xx,y+xy,x+40,y+xy,x+xx-1,y+xy);
 xx--;
 if(xx==22) xx=18;
 for(a=xy;a<=xy;a++)
  for(b=xx;b<41;b++)
   str[a][b]=str[a][b+1];
 }
 else if(xx>43 && xx<52)
 {
  movetext(x+xx,y+xy,x+50,y+xy,x+xx-1,y+xy);
  xx--;
  if(xx==43) xx=39;
  for(a=xy;a<=xy;a++)
   for(b=xx;b<52;b++)
    str[a][b]=str[a][b+1];
  }
  else if(xx>54 && xx<71)
```

```
{
 movetext(x+xx,y+xy,x+70,y+xy,x+xx-1,y+xy);
 xx--;
 if(xx==54) xx=50;
 for(a=xy;a<=xy;a++)
  for(b=xx;b<69;b++)
   str[a][b]=str[a][b+1];
 }
 v_line();
 }
 break;
    case 0x20:                  /* ok */
 if(str[xy][xx]<=0x00){
  str[xy][xx]=0x20;
  gotoxy(30,2);
  printf("*");g=0;
 }
 if(xx>=0 && xx<19){
 movetext(x+xx,y+xy,x+19,y+xy,x+xx+1,y+xy);
 xx++;
 if(xx==19) xx=23;
 gotoxy(x+xx,y+xy);
 printf("%c %c",8,8);
 for(a=xy;a<=xy;a++)
 for(b=18;b>=(xx-1);b--){
  str[a][b+1]=str[a][b];
  if(b==(xx-1)) str[a][b]=0x20;
```

```
}
for(a=xy;a<=yy;a++)
 for(b=0;b<xx;b++)
  if(str[a][b]<0x20) str[a][b]=0x20;
}
if(xx>22 && xx<41){
movetext(x+xx,y+xy,x+40,y+xy,x+xx+1,y+xy);
xx++;
if(xx==40) xx=44;
        gotoxy(x+xx,y+xy);
printf("%c %c",8,8);
for(a=xy;a<=xy;a++)
for(b=40;b>=(xx-1);b--){
 str[a][b+1]=str[a][b];
 if(b==(xx-1)) str[a][b]=0x20;
}
for(a=xy;a<=yy;a++)
 for(b=0;b<xx;b++)
  if(str[a][b]<0x20) str[a][b]=0x20;
}
if(xx>43 && xx<52){
movetext(x+xx,y+xy,x+50,y+xy,x+xx+1,y+xy);
xx++;
if(xx==51) xx=55;
        gotoxy(x+xx,y+xy);
printf("%c %c",8,8);
for(a=xy;a<=xy;a++)
```

```
for(b=51;b>=(xx-1);b--){
 str[a][b+1]=str[a][b];
 if(b==(xx-1)) str[a][b]=0x20;
}
for(a=xy;a<=yy;a++)
 for(b=0;b<xx;b++)
  if(str[a][b]<0x20) str[a][b]=0x20;
}
if(xx>54 && xx<71){
movetext(x+xx,y+xy,x+70,y+xy,x+xx+1,y+xy);
xx++;
if(xx==71){
xx=0;xy++;yy++;}
        gotoxy(x+xx,y+xy);
printf("%c %c",8,8);
for(a=xy;a<=xy;a++)
for(b=68;b>=(xx-1);b--){
 str[a][b+1]=str[a][b];
 if(b==(xx-1)) str[a][b]=0x20;
}
for(a=xy;a<=yy;a++)
 for(b=0;b<xx;b++)
  if(str[a][b]<0x20) str[a][b]=0x20;
}
v_line();
break;
   case 0x0d:                /* ok */
```

```
    if(str[xy][xx]<=0x00){
    str[xy][xx]=0x20;
    gotoxy(30,2);
    printf("*");g=0;
    }
    if(xx>=0 && xx<19) xx=23;
    else if(xx>22 && xx<41) xx=44;
    else if(xx>43 && xx<52) xx=55;
    else if(xx>54 && xx<71)
    {
    xx=0;xy++;yy++;
    if(xy>=13)
    {
movetext(5,6,23,19,5,5);
movetext(28,6,44,19,28,5);
movetext(49,6,55,19,49,5);
movetext(60,6,76,19,60,5);
xy=12;
     }
     if(yy==13*page){
      page++;
      xx1=xx2;
      xx2=xx2*page;
     }
     }
     break;
    default:break;
```

```
    }
   }
   else{
   gotoxy(30,2);
   printf("*");g=0;
   gotoxy(x+xx,y+xy);
    if(xx==18) xx=18;
    else if(xx==39) xx=39;
    else if(xx==50) xx=50;
    else if(xx==70) xx=70;
    else ins(length);
   }
 }while(1);
}
/*********************************************************/
```

****

INS.C

****

```
#include <conio.h>
#include <stdio.h>
#include <alloc.h>
#include <dir.h>
#include <graphics.h>

/*****************************************************/
void ins(int length)
{
extern int INS;
extern char readin;
extern unsigned char str[500][70];
extern int aa,a,b,xx,x,xy,y,yy,ly;
if(INS==1){
      str[xy][xx]=readin;
      cprintf("%c",readin);
       if(xx<(length-3))
        xx++;
      }
     else{
       if(xx>=0 && xx<19){
       for(a=xy;a<=xy;a++)
        for(b=18;b>=xx;b--){
```

```
str[a][b+1]=str[a][b];
if(b==xx) str[a][b]=0x20;
      }
     for(a=xy;a<=yy;a++)
      for(b=0;b<xx;b++)
if(str[a][b]<0x20) str[a][b]=0x20;
     str[xy][xx]=readin;
      movetext(x+xx,y+xy,x+19,y+xy,x+xx+1,y+xy);
      xx++;
      if(xx==19) xx=23;
     }
     if(xx>22 && xx<41){
     for(a=xy;a<=xy;a++)
      for(b=40;b>=xx;b--){
str[a][b+1]=str[a][b];
if(b==xx) str[a][b]=0x20;
      }
     for(a=xy;a<=yy;a++)
      for(b=0;b<xx;b++)
if(str[a][b]<0x20) str[a][b]=0x20;
     str[xy][xx]=readin;
      movetext(x+xx,y+xy,x+40,y+xy,x+xx+1,y+xy);
      xx++;
      if(xx==40) xx=44;
     }
     if(xx>43 && xx<52){
     for(a=xy;a<=xy;a++)
```

```
        for(b=51;b>=xx;b--){
str[a][b+1]=str[a][b];
if(b==xx) str[a][b]=0x20;
        }
       for(a=xy;a<=yy;a++)
        for(b=0;b<xx;b++)
if(str[a][b]<0x20) str[a][b]=0x20;
       str[xy][xx]=readin;
        movetext(x+xx,y+xy,x+50,y+xy,x+xx+1,y+xy);
        xx++;
        if(xx==51) xx=55;
       }
       if(xx>54 && xx<71){
       for(a=xy;a<=xy;a++)
        for(b=69;b>=xx;b--){
str[a][b+1]=str[a][b];
if(b==xx) str[a][b]=0x20;
        }
       for(a=xy;a<=yy;a++)
        for(b=0;b<xx;b++)
if(str[a][b]<0x20) str[a][b]=0x20;
       str[xy][xx]=readin;
        movetext(x+xx,y+xy,x+70,y+xy,x+xx+1,y+xy);
        xx++;
        if(xx==71){
xx=0;xy++;yy++;
        }
```

```
        }
        gotoxy(x+xx,y+xy);
        printf("%c %c",8,8);
        xx--;
        v_line();

        gotoxy(x+xx,y+xy);
        cprintf("%c",readin);
        if(xx<(length-1))
          xx++;
       }
}
/************************************/
```

****

PMEM.C

****

```
#include <conio.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <mem.h>
#include <string.h>
#include <math.h>
#include <bios.h>
#include <process.h>
#include <graphics.h>
#include "info.h"
extern FILEINFO fi;
extern char *taget,*buff,*taget1,*taget2,*taget3,*taget4,*buff_org;
extern unsigned long cnt_buff;
extern unsigned cnt_width;
extern unsigned scr_width;
extern unsigned scr_depth;
extern char current_vga_bank;
/*****************************************************/

char Pull_Byte_Mem()
```

```
{
 char *buffer;

 if((unsigned long)cnt_buff<65536)
  buffer=taget;
 else
  if((unsigned long)cnt_buff<131072)
   buffer=taget1;
  else
   if((unsigned long)cnt_buff<196608)
    buffer=taget2;
   else
    if((unsigned long)cnt_buff<262144)
     buffer=taget3;
    else
     buffer=taget4;

  return(buffer[cnt_buff]);
}
/*****************************************/
Put_Byte_Mem(char data)
{
 if((unsigned long)cnt_buff<65536)
  buff_org=taget;
 else
  if((unsigned long)cnt_buff<131072)
   buff_org=taget1;
```

```
  else
   if((unsigned long)cnt_buff<196608)
    buff_org=taget2;
   else
    if((unsigned long)cnt_buff<262144)
     buff_org=taget3;
    else
     buff_org=taget4;

  buff_org[cnt_buff]=data;          /* */
  cnt_buff++;
}
/*****************************************/
Sav_Byte_Mem(char data)
{
 unsigned cnt;

 if(cnt_width==fi.width)
 {
  for(cnt=cnt_width+1;cnt<scr_width;cnt++)
   Put_Byte_Mem(0);
  cnt_width=0;
 }
 else
  cnt_width++;
 switch(fi.bits)
 {
```

```
 case 1:       /* 2 or 16 color */
switch(fi.plane)
{
 case 1:for(cnt=0;cnt<8;cnt++)
 if((data>>(7-cnt)&1)==0)
  Put_Byte_Mem(1);
 else
  cnt_buff++;
if((cnt_buff%scr_width)>=fi.bytes*8)
{
 cnt_buff=cnt_buff+(scr_width-fi.bytes*8);
 cnt_width=0;
}
break;
 case 4:/*if(cnt_buff%scr_width<160)
{
 for(cnt=0;cnt<8;cnt++,cnt_buff++)
  if((data>>(7-cnt)&1)==0)
   buffer[(unsigned)(cnt_buff/scr_width)*
              scr_width+(cnt_buff%scr_width)]=1;
}
else
 if(cnt_buff%scr_width<320)
 {
  for(cnt=0;cnt<8;cnt++,cnt_buff++)
   if((data>>(7-cnt)&1)==0)
    buffer[(unsigned)(cnt_buff/scr_width)*scr_width+
```

```
    (cnt_buff%scr_width)-160]=buffer[(unsigned)(cnt_buff/scr_width)*
    scr_width+(cnt_buff%scr_width)-160]|2;
  }
   else
   if(cnt_buff%scr_width<480)
   {
    for(cnt=0;cnt<8;cnt++,cnt_buff++)
     if((data>>(7-cnt)&1)==0)
      buffer[(unsigned)(cnt_buff/scr_width)*scr_width+
      (cnt_buff%scr_width)-320]=buffer[(unsigned)(cnt_buff/scr_width)*
      scr_width+(cnt_buff%scr_width)-320]|4;
   }
   else
    for(cnt=0;cnt<8;cnt++,cnt_buff++)
     if((data>>(7-cnt)&1)==0)
      buffer[(unsigned)(cnt_buff/scr_width)*scr_width+
      (cnt_buff%scr_width)-480]=buffer[(unsigned)(cnt_buff/scr_width)*
       scr_width+(cnt_buff%scr_width)-480]|8;
 break;
   */
/* for(cnt=0;cnt<2;cnt++,cnt_buff++)
  buffer[cnt_buff]=(data>>(4*cnt))&0xf;
*//* if((cnt_buff%scr_width)>=(fi.bytes*fi.plane*2))
  cnt_buff=cnt_buff+(scr_width-fi.bytes*fi.plane*2);*/
 break;
  }
 break;
```

```
  case 8:Put_Byte_Mem(data);          /* 256 color */
 break;
 }
}
/**********************************************/
```

กิตติกรรมประกาศ

ขอขอบพระคุณ อาจารย์ ทรงชัย วีระทวีมาศ เป็นอย่างสูงที่ได้ให้คำแนะนำและให้คำปรึกษาที่เป็นประโยชน์ต่อปริญญานิพนธ์ฉบับนี้ จนทำให้ปริญญานิพนธ์นี้เสร็จสมบูรณ์ตามวัตถุประสงค์ที่คณะผู้จัดทำได้ตั้งเป้าหมายไว้

ขอขอบพระคุณ คุณพ่อ-คุณแม่ ของคณะผู้จัดทำทุกคนที่ได้ให้ทุนทรัพย์และให้กำลังใจมาโดยตลอด และเพื่อนทุกคนที่ให้ความช่วยเหลือเป็นอย่างดีจนทำให้โครงงานนี้สำเร็จลุล่วงไปได้ด้วยดี

คณะผู้จัดทำ

หนังสืออ้างอิง

1. ดร.ไพศาล สงวนหมู่,"กราฟิกบนไมโครคอมพิวเตอร์",วารสารไมโครคอมพิวเตอร์, ฉบับที่ 55,2533,หน้า 269-283.

2. ถวัลย์ ตั้งลิตานนท์,"เปิดโฉม VGA ฮาร์ดแวร์",วารสารไมโครคอมพิวเตอร์, ฉบับที่ 55,2533,หน้า 284-294.

3. จเร เลิศสุดวิชัย,"การเขียนโปรแกรมใช้งานการ์ด VGA ตอน 1",วารสารไมโครคอมพิวเตอร์,ฉบับที่ 55,2533,หน้า 295-305.

4. ชัยยุทธ ต้นประทุมวงษ์,"แกะรอย PCX ด้วยเทอร์โบซี",วารสารไมโครคอมพิวเตอร์, ฉบับที่ 78,2535,หน้า 256-265.

5. สมพันธ์ ชาญศิลย์,"ภาษาซี",ภาควิชาไฟฟ้า,คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น,2535.

6. David C. Kay and John R. Levine,"Graphics File Formats", Mc GRAW-HILL,1992.

7. Steve Rimmer,"Super Charged Bitmapped Graphics",Mc GRAW-HILL, 1992.