บทที่ 4

การออกแบบและพัฒนาโปรแกรมควบคุมเมาส์สำหรับโปรแกรมเลียนแบบเทอร์มินอล

โปรแกรมควบคุมเมาส์ สำหรับโปรแกรมเลียนแบบเทอร์มินอล ทำงานโดยการรับข้อมูลจากอุปกรณ์เมาส์และนำข้อมูลที่ได้มาประมวลผลส่งค่าให้กับคีร์บอร์ดบัฟเฟอร์ ซึ่งการใช้งาน ผู้ใช้สามารถกำหนดความต้องการของเมาส์ให้สามารถทำงานตามที่ผู้ใช้ต้องการได้ โดยกำหนดในไฟล์ หรือกำหนดขณะที่โปรแกรมกำลังทำงาน และการเรียกใช้งานจะเป็นลักษณะโปรแกรม Pop-up เมนู แบบฝังตัว

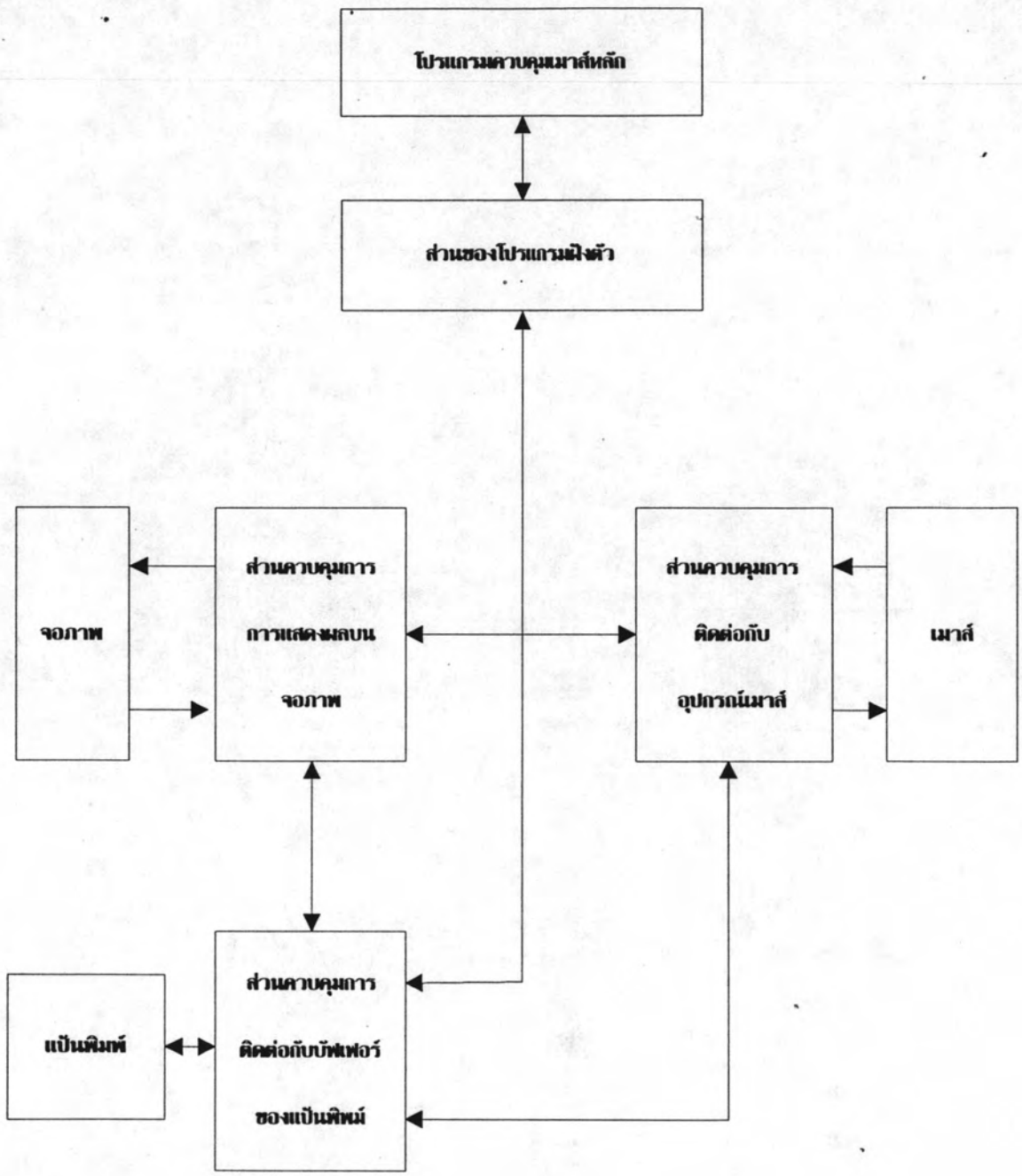
การพัฒนาโปรแกรมควบคุมเมาส์ พัฒนาขึ้นด้วยภาษาซี โดยออกแบบและพัฒนาสำหรับเครื่องคอมพิวเตอร์ตระกูลไอบีเอ็มพีซี ซึ่งการพัฒนาสามารถแยกตามหน้าที่การทำงาน ออกเป็นส่วนสำคัญๆ ได้ดังต่อไปนี้

1. โปรแกรมหลักและส่วนควบคุมการทำงานแบบฝังตัว

2. ส่วนควบคุมการติดต่อกับเมาส์

3. ส่วนควบคุมการติดต่อกับบัฟเฟอร์ของแป้นพิมพ์

4. ส่วนควบคุมการแสดงผลบนจอภาพ



ภาพที่ 4.1 แสดงผังการทำงานของส่วนควบคุมต่าง ๆ ของโปรแกรมควบคุมมาส์สำหรับ

โปรแกรมเลียนแบบเทอร์มินอล

4.1 โปรแกรมหลักและส่วนควบคุมการทำงานแบบฝังตัว

การสร้างโปรแกรมควบคุมเมาส์ที่จัดทำ ได้อาศัยการเขียนโปรแกรมลักษณะฝังตัว (Terminate and Stay Resident หรือย่อว่า TSR) และสำหรับการเขียนโปรแกรมประเภทฝังตัว (TSR) นั้น จะขึ้นอยู่กับฮาร์ดแวร์และระบบปฏิบัติการ (Operating System) ดังนั้นในโปรแกรมที่จะกล่าวถึงต่อไปนี้ จะทำงานได้เฉพาะบนเครื่องไอบีเอ็มพีซี คอมแพทติเบิล และใช้ดอสเป็นระบบปฏิบัติการ ซึ่งฟังก์ชันต่างๆจะเป็นของเทอร์โบซีเวอร์ชัน 1.0 ขึ้นไป และในการเขียนโปรแกรมแบบฝังตัวนั้น ต้องมีการดัดแปลงตารางอินเทอร์รัพต์เวกเตอร์ (Interrupt Vector Table) ประกอบกับมีความยากในการจัดการ ดังนั้นอาจเกิดข้อผิดพลาดที่เกิดจากการทำงานของโปรแกรมได้ ที่เรียกว่า System Crash ซึ่งอาจทำให้ข้อมูลในฮาร์ดดิสก์เสีย ดังนั้นในระหว่างการพัฒนาจึงต้องมีการทำสำเนาข้อมูล (Backup) บนฮาร์ดดิสก์ไว้ เพื่อความปลอดภัย

4.1.1 พื้นฐานเกี่ยวกับโปรแกรมแบบฝังตัว

โปรแกรมแบบฝังตัวเป็นโปรแกรมที่ เมื่อโหลดเข้ามาทำงานแล้วจะคืนการทำงานให้กับดอส กล่าวคือมีการเรียกใช้ฟังก์ชันของดอสหมายเลข 49 แล้วตัวโปรแกรมจะยังค้างอยู่ในหน่วยความจำ ซึ่งโปรแกรมจะไม่ได้ถูกควบคุมโดยดอสแล้ว แต่สามารถทำงานได้โดยไม่ต้องทำการโหลดอีกครั้ง ตัวอย่างของโปรแกรมชนิดฝังตัว (TSR) เช่น โปรแกรม Sidekick ของ Borland International เป็นต้น

โปรแกรมแบบฝังตัวจะถูกเรียกขึ้นมาทำงานได้ โดยการใช้อินเทอร์รัพต์จากสัญญาณนาฬิกา, คีบอร์ด, เมาส์ หรืออุปกรณ์อื่น ๆ ซึ่งสิ่งที่จะปลุกโปรแกรมแบบฝังตัวเหล่านี้จะใช้ชื่อว่า Hot Key

4.1.2 ข้อมูลอินเทอร์รัพต์ในไมโครโปรเซสเซอร์ ตระกูล 8086

ในไมโครโปรเซสเซอร์ตระกูล 8086 จะมีอินเทอร์รัพต์เวกเตอร์ (Interrupt Vector) อยู่ทั้งหมด 256 ตำแหน่ง ซึ่งในแต่ละตำแหน่งจะเป็นค่าของแอดเดรสเริ่มต้นของโปรแกรม ที่มีชื่อเรียกรวม ๆ ว่า "รูทีนจัดการอินเทอร์รัพต์" (Interrupt Service Routine หรือย่อว่า ISR) และรูทีนเหล่านี้จะถูกรวบรวมไว้ในหน่วยความจำส่วนหนึ่งที่เรียกว่า "ตารางอินเทอร์รัพต์" (Interrupt Vector table)

หมายเลขเวกเตอร์	การทำงาน (Action)
00h	Divide By Zero
01h	Single Step
02h	Nonmasable Interrupt
03h	Break Point
04h	Overflow
05h	Print Screen
06h	Unused
07h	Unused
08h	Hardware IRQ 0 (Timer Tick)
09h	Keyboard Input Interrupt
0Ah	Reserved
0Bh	Asynchronous Port Controller 1 (Com2)

ตารางที่ 4.1 แสดงหน้าที่ของอินเทอร์รัพต์เวกเตอร์หมายเลขต่าง ๆ

หมายเลขเวกเตอร์	การทำงาน (Action)
0Ch	Asynchronous Port Controller 0 (Com1)
0Dh	Fixed Disk Controller
0Eh	Floppy Disk Controller
0Fh	Printer Controller
10h	Video Driver
11h	Equipment Configuration Check
12h	Memory Size Check
13h	Floppy Disk/Fixed Disk (PC/XT)
14h	Comm Port Driver
15h	Cassette/Network Service
16h	Keyboard Driver
17h	Printer Driver
18h	ROM BASIC
19h	Restart System
1Ah	Set/Read Real Time Clock
1Bh	Ctrl-Break Handler
1Ch	Timer Tick (User Defined)
1Dh	Video Parameter Table
1Eh	Disk Parameter Table
1Fh	Graphic Character Table (Character 80-FFh)
20h	Program Terminate

ตารางที่ 4.1 แสดงหน้าที่ของอินเทอร์รัพต์เวกเตอร์หมายเลขต่าง ๆ (ต่อ)

หมายเลขเวกเตอร์	การทำงาน (Action)
21h	DOS Function Dispatcher
22h	Terminate Vector
23h	Ctrl-C Vector
24h	Critical-Error Vector
25h	Absolute Disk Read
26h	Absolute Disk Write
27h	Terminate And Stay Resident
28h	DOS OK Interrupt
2Fh	Multiplex Interrupt

ตารางที่ 4.1 แสดงหน้าที่ของอินเทอร์รัพต์เวกเตอร์หมายเลขต่าง ๆ (ต่อ)

ตารางอินเทอร์รัพต์เวกเตอร์นี้ มีขนาด 1024 ไบต์ โดยเริ่มต้นที่แอดเดรส 0000:0000 และเนื่องจาก ISR นั้นอยู่ที่ตำแหน่งใดก็ได้ในหน่วยความจำดังนั้นจึงต้องใช้แอดเดรส ที่ระบุตำแหน่งของ ISR ขนาด 32 บิด เพื่อที่จะสามารถอ้างหน่วยความจำได้ทั้งหมด ทำให้แต่ละตำแหน่งของอินเทอร์รัพท์เวกเตอร์ ต้องการเนื้อที่ 4 ไบต์ จึงทำให้การหาค่าของอินเทอร์รัพต์เวกเตอร์ ตำแหน่งต่าง ๆ เป็นดังนี้ เช่น ถ้าต้องการหาแอดเดรสเริ่มต้นของ ISR ของอินเทอร์รัพต์เวกเตอร์หมายเลข 0 ต้องอ่านค่าในหน่วยความจำที่ตำแหน่ง 0000:0000 ถึง 0000:0003 และ อินเทอร์รัพต์เวกเตอร์หมายเลข 1 อ่านตำแหน่งที่ 0000:0004 ถึง 0000:0007 เป็นต้น

4.1.3 ปัญหาอินเทอร์รัพต์ของดอสและไบออสที่มีผลต่อโปรแกรมฝังตัว

โดยทั่วไปแล้วคำสั่งบนดอสไม่สามารถเกิด Re-entrance ได้หรือมีความหมายว่า ขณะที่กำลังเรียกใช้โปรแกรมหนึ่งบนดอส จะไม่สามารถเรียกใช้อีกโปรแกรมหนึ่งไปพร้อมกันได้ ซึ่งด้วยเหตุดังกล่าว จึงไม่สามารถเรียกใช้โปรแกรมในลักษณะ Multitasking ภายใต้ดอสได้ ดังนั้นการเขียน ISR จึงไม่สามารถใช้ Function Call ของดอสได้ มิฉะนั้นแล้วจะเกิด System Crash โปรแกรมที่ทำหน้าที่เป็น ISR จึงต้องใช้คำสั่งที่ไม่ได้ประกอบไปด้วย Function Call ของดอส เช่น การใช้วิธีติดต่อกับ VIDEO RAM โดยตรงในการเขียนอักษรลงบนจอภาพ เป็นต้น

ส่วน Function Call ของBios นั้นสามารถเกิดRe-entrance ได้บ้าง เช่น การใช้อินเทอร์รัพต์เวกเตอร์หมายเลข 16 ที่ทำหน้าที่รับอินพุตจากการกดคีย์บอร์ด ซึ่งสามารถทำงานได้โดยไม่เกิดผลข้างเคียง โดยเฉพาะฟังก์ชัน 0 ส่วนอินเทอร์รัพต์ของ Bios ตัวอื่น ๆ นั้น อาจไม่ปลอดภัยถ้านำไปใช้ และสำหรับการที่จะรู้ว่าอินเทอร์รัพต์หมายเลขใด สามารถใช้ได้หรือไม่ ทำได้โดยทดลองเขียน ISR โดยใช้ Function Call หมายเลขนั้น และให้ ISR นั้นทำงาน ถ้าใช้ไม่ได้จะเกิด System Crash เกิดขึ้น สำหรับฟังก์ชันมาตรฐานในภาษาซีนั้นประกอบด้วย Function Call ของดอส และของ BIOS ข้อควรจำก็คือ ฟังก์ชันมาตรฐานที่ทำงานเกี่ยวกับอินพุต และเอาต์พุต ถ้าไม่ใช้ Function Call ของดอสก็เป็นของ BIOS เช่น ฟังก์ชัน Malloc() จะใช้ Function Call ของดอส เพื่อหาว่าขณะนั้นมีหน่วยความจำว่างอยู่เท่าไร เป็นต้น

4.1.4 โครงสร้างของโปรแกรมฝังตัว

โปรแกรมควบคุมเมาส์ชนิดฝังตัวที่ได้จัดทำ สามารถแบ่งโครงสร้างได้เป็น 2 ส่วน โดยส่วนแรกคือ ส่วนของโปรแกรมหลักหรือส่วนติดตั้ง มีหน้าที่ตั้งค่าเริ่มต้นต่างๆ ของโปรแกรม และกลับสู่ดอสโดยใช้ Function Call ของดอสที่ทำให้โปรแกรม

เป็นโปรแกรมฝังตัว โปรแกรมส่วนนี้ จะไม่ทำงานอีก ยกเว้นเมื่อโหลดโปรแกรมมาทำงานอีกครั้ง การทำงานของส่วนนี้คือ ตั้งค่าแอดเดรสเริ่มต้นของโปรแกรม ส่วนที่เป็น ISR ลงในตารางอินเทอร์รัพต์เวกเตอร์

สำหรับส่วนที่สองเป็นส่วนที่ทำหน้าที่แสดงเมนูแบบ Pop-up ซึ่งเป็นส่วนเกี่ยวกับการแสดงวินโดว์ และการแสดงผลในลักษณะวินโดว์นี้เมื่อทำงานเสร็จ ตัวอักษรที่อยู่บนจอภาพของคอมพิวเตอร์ ต้องไม่เปลี่ยนแปลง เมื่อเทียบกับหน้าจอก่อนที่โปรแกรมส่วนนี้ทำงาน

4.1.5 การตรวจสอบว่าเมื่อไร DOS จึงอยู่ในสถานะปลอดภัยที่จะเรียกใช้อินเทอร์รัพต์

โดยทั่ว ๆ ไปแล้วจะไม่มีการอธิบายถึง ความสามารถของดอสที่ให้โปรแกรมฝังตัวใช้แหล่งข้อมูลหลาย ๆ ตัว ในช่วงระยะเวลาหนึ่งได้ ซึ่งปกติถ้าโปรแกรมฝังตัวใช้แหล่งข้อมูลพวก I/O Console ก็จะไม่เกิดปัญหาอะไร แต่ปัญหาจะเกิดขึ้นเมื่อมีการใช้แหล่งข้อมูลพวกไฟล์ในดิสก์ หรือ พวกอุปกรณ์สื่อสาร (I/O Port) ซึ่งจุดนี้ในหนังสือ DOS Technical Reference ก็ไม่มีการกล่าวถึง ซึ่งเมื่อดอสอยู่ในสถานะปลอดภัย (Safe) จะเรียกใช้อินเทอร์รัพต์ หมายเลข 28 H และจากที่ทราบอยู่ทั่วไปที่มีการทำงานในระบบปฏิบัติการ ส่วนหนึ่งเรียกว่า Critical System ซึ่งไม่สามารถจะเกิดอินเทอร์รัพต์ หมายเลข 28H เมื่อทำงานอยู่ในส่วนนี้ได้ จากจุดนี้เองเราสามารถนำคุณสมบัตินี้มาใช้กับโปรแกรมฝังตัวได้

สำหรับการใช้อินเทอร์รัพต์ หมายเลข 28H เป็นตัวเริ่มทำงานของโปรแกรมฝังตัวมีข้อควรระวังคือ เมื่อเกิดอินเทอร์รัพต์นี้ขึ้น หลังจากนี้โปรแกรมฝังตัวจะไม่สามารถทำงานโดยวิธีตรวจสอบปุ่มที่ถูกกดได้ ทางแก้ปัญหาก็คือ โปรแกรมส่วนที่ตรวจสอบการกดปุ่ม จะตั้งค่าของตัวบ่งชี้ (Flag) ที่ทำหน้าที่บอกสถานะหรือ DOS -

Active Flag ถ้ามีการกดคีย์พิเศษก่อนที่โปรแกรมฝังตัวจะทำงาน และจะมีการสร้างอินเทอร์รัพต์ หมายเลข 28H ขึ้นมาตรวจสอบว่า DOS - Active Flag ถูกตั้งค่าไว้หรือไม่ ถ้ามีการตั้งค่าไว้จะทำการรีเซตค่าใน DOS - Active Flag แล้ว โปรแกรมฝังตัวจะเริ่มทำงาน การใช้วิธีนี้ต้องมีการเปลี่ยนค่าในตารางอินเทอร์รัพต์ ของอินเทอร์รัพต์ 28H เป็นค่าแอดเดรสเริ่มต้นของส่วน ISR ของโปรแกรมฝังตัว และค่าของตำแหน่งของ ISR เดิมจะถูกย้ายไปที่อินเทอร์รัพต์ที่ไม่มีการใช้งาน นั่นคือตั้งแต่หมายเลข 60 ขึ้นไป และจะถูกเรียกใช้งานโดยรูทีนของโปรแกรมฝังตัว เมื่อมีการเรียกใช้อินเทอร์รัพต์ 28H ซึ่งการใช้วิธีนี้จะสามารถป้องกันอันตรายจากการเรียกใช้ Function Call ของดอส ในโปรแกรมฝังตัวได้

4.1.6 การใช้งานอินเทอร์รัพต์เวลา (Timer Interrupt)

อินเทอร์รัพต์ที่จะใช้ในการปลุกโปรแกรม TSR ที่จะพัฒนาขึ้นนี่จะจะใช้ Timer Interrupt เป็นตัวช่วย ซึ่งโดยปกติแล้วระบบนาฬิกาของเครื่องจะผลิตอินเทอร์รัพต์หมายเลข 8H ทุก ๆ 18.2 วินาที ซึ่งเราจะใช้อินเทอร์รัพต์นี้ไปปลุกโปรแกรมที่ทำหน้าที่ตรวจสอบ DOS - Active Flag และถ้า DOS - Active Flag อยู่ในสภาวะปลอดภัย ประกอบกับเมาส์มีการใช้งาน (Hot Key) โปรแกรมควบคุมเมาส์ที่ฝังตัวอยู่ที่จะถูกปลุกขึ้นมาทำงาน

4.1.7 คำสั่งใช้งานสำหรับโปรแกรมฝังตัวของเทอร์โบซี

ฟังก์ชันของเทอร์โบซีหลัก ๆ ที่ใช้สำหรับโปรแกรมฝังตัวที่จัดทำขึ้นนี้ ได้แก่

ฟังก์ชัน Int86() หรือ Int86x()

ใช้ในการผลิตซอฟแวร์อินเทอร์รัพต์

ฟังก์ชัน Getvect()

ใช้ในการอ่านค่าแอดเดรสในตารางอินเทอร์รัพต์เวกเตอร์ของหมายเลขอินเทอร์รัพต์ ที่กำหนดให้

ฟังก์ชัน setvect()

ใช้ในการ ตั้งค่าแอดเดรสในตารางอินเทอร์รัพต์เวกเตอร์ของหมายเลขอินเทอร์รัพต์ ที่กำหนดให้

ฟังก์ชัน Keep()

ใช้คำสั่งนี้ เพื่อออกจากโปรแกรมแล้วกลับสู่การทำงานของดอส แต่ยังคงทิ้งโปรแกรมที่ฝังตัวอยู่ในหน่วยความจำ เพื่อที่จะถูกปลุกขึ้นมาทำงานได้

ฟังก์ชัน Freemem()

ใช้คำสั่งนี้ เพื่อที่จะปล่อยหน่วยความจำที่เคยจองไว้สำหรับโปรแกรมฝังตัว และใช้เมื่อเลิกใช้โปรแกรมฝังตัว

4.2 ส่วนควบคุมการติดต่อกับเมาส์

จากการเรียกใช้อินเทอร์รัพต์หมายเลข 33H ที่ควบคุมและอ่านค่าสถานะของเมาส์ สามารถนำมาสร้างฟังก์ชันเพื่อใช้พัฒนาโปรแกรมควบคุมเมาส์นี้ได้ ซึ่งทำให้การเขียนโปรแกรมควบคุมเมาส์สะดวกขึ้น ฟังก์ชันที่สร้างขึ้นมาใช้งานมีดังต่อไปนี้

ฟังก์ชันรีเซตเมาส์ (Mouse_reset)

ทำหน้าที่ตรวจสอบว่ามีการติดตั้งฮาร์ดแวร์และซอฟต์แวร์ ของเมาส์เรียบร้อยแล้วหรือไม่ และยังตรวจสอบว่าเมาส์ที่ติดตั้งเป็นแบบ 2 ปุ่มหรือ 3 ปุ่มด้วย

ฟังกชันเกี่ยวกับการแสดงเคอร์เซอร์ของเมาส์ (Mouse_on, Mouse_off)

ทำหน้าที่กำหนดการแสดงเคอร์เซอร์ของเมาส์ ให้ปรากฏบนจอภาพหรือลบเคอร์เซอร์ของเมาส์ไม่ให้ปรากฏบนจอภาพ ซึ่งขึ้นอยู่กับการเลือกใช้งาน

ฟังก์ชันตรวจสอบการกดปุ่ม (Button_press)

ทำหน้าที่ตรวจสอบการกดปุ่มเมาส์ว่าเป็นการกดปุ่มทางซ้าย หรือทางขวา หรือไม่มีการกดปุ่มใด ๆ

ฟังก์ชันตรวจสอบการปล่อยปุ่ม (Button_release)

ทำหน้าที่ตรวจสอบการปล่อยปุ่มเมาส์ว่า เป็นการกดปล่อยปุ่มทางซ้ายหรือทางขวา นับจากที่มีการเรียกอินเทอร์รัพต์นี้ครั้งสุดท้าย

ฟังก์ชันตรวจสอบการเคลื่อนที่ (Move_mouse)

ทำหน้าที่ตรวจสอบระยะทางที่เปลี่ยนไป จากตำแหน่งสุดท้ายที่มีการ

เรียกอินเทอร์รัพต์ โดยจะส่งค่าระยะทางของตำแหน่งที่เปลี่ยนแปลงในแนวนอนและ แนวตั้งกลับมา

ฟังก์ชันการอ่านและกำหนดตำแหน่งของเคอร์เซอร์ (Set_mose_posn และ Mouse_txt_posn)

ทำหน้าที่กำหนดตำแหน่งของเคอร์เซอร์ของเมาส์ และอ่านตำแหน่งของเคอร์เซอร์ของเมาส์ฯลฯ ตามลำดับ

ฟังก์ชันตรวจสอบตำแหน่งของเคอร์เซอร์เทียบกับขอบเขตที่กำหนด (Mouse_in_box)

ทำหน้าที่ตรวจสอบตำแหน่งของเคอร์เซอร์ของเมาส์ โดยเรียกใช้ฟังก์ชัน mouse_txt_posn แล้วนำค่าตำแหน่งที่ได้ ไปตรวจสอบว่าอยู่ในกรอบที่กำหนดหรือไม่

4.3 ส่วนควบคุมการติดต่อกับบัฟเฟอร์ของแป้นพิมพ์

เป็นส่วนที่ควบคุมการนำเอา Scan Code และ Ascii Code ที่ผู้ใช้ได้สั่งงานจากการกดปุ่มเมาส์ หรือเลื่อนเมาส์ ไปใส่ไว้ในบัฟเฟอร์ของคีย์บอร์ด โดยบัฟเฟอร์ที่ทำหน้าที่เก็บรหัสของปุ่มที่ถูกกดของดอสนั้น สามารถเก็บรหัสไว้ได้ 16 ตัว การทำงานจะเป็นลักษณะดังนี้คือ ทุกครั้งที่มีการเรียกใช้เมาส์ จะมีการเรียกใช้อินเทอร์รัพต์ 33H ขึ้น โปรแกรม ISR ที่ทำหน้าที่นี้ จะอ่านรหัสที่แปรความหมายจากการถูกกดปุ่มเมาส์ มาเก็บในบัฟเฟอร์ สำหรับตำแหน่งของบัฟเฟอร์นี้อยู่ที่แอดเดรส 0000:041E หรือ 1054 เมื่อเป็นเลขฐาน 10 ซึ่งในการจัดเก็บรหัส Scan Code ของคีย์บอร์ดบัฟเฟอร์ต้องสอดคล้องกับการนำค่าจากการกดปุ่มคีย์บอร์ดมาจัดเก็บ และการกดปุ่มแต่ละครั้งจะมีการสร้าง Scan Code ขนาด 16 บิต ดังนั้นบัฟเฟอร์จึงต้องมีขนาด 30 ไบต์ สำหรับรหัส 15 ตัว และอีก

2 ไบต์ที่ตำแหน่งสุดท้าย จะเก็บรหัสของ <Return> ไว้โดยอัตโนมัติ การทำงานของบัฟเฟอร์จะมีลักษณะเป็น Circular Queue การติดต่อกับบัฟเฟอร์สามารถติดต่อโดยอาศัยตัวชี้ตำแหน่งต่าง ๆ คือ

- ตัวชี้ตำแหน่งส่วนหัว (Head Pointer) จะชี้ไปที่รหัส ที่ถูกกดตัวสุดท้าย โดยค่าของตำแหน่งของตัวชี้ตำแหน่งส่วนหัวนี้ จะเก็บไว้ที่หน่วยความจำตำแหน่ง 0000:041A (เลขฐาน 10 คือ 1050)

- ตัวชี้ตำแหน่งส่วนหาง (Tail Pointer) จะชี้ไปที่รหัสตัวต่อไปที่จะถูกส่งไปให้ Function Call ของดอสหรือของ Bios ที่ต้องการข้อมูล โดยค่าของตำแหน่งของตัวชี้ตำแหน่งส่วนหางนี้ จะเก็บไว้ที่หน่วยความจำตำแหน่ง 0000:041C (เลขฐาน 10 คือ 1052)

4.4 ส่วนควบคุมการแสดงผลบนจอภาพ

สำหรับการออกแบบส่วนควบคุมนี้ มีความต้องการให้โปรแกรมสามารถทำงานได้ทั้งการ์ดเฮอร์คิวลิส อีจีเอ และ ในการนำไปใช้งานผู้ใช้ไม่ต้องคอยตรวจสอบว่าการ์ดที่ใช้เป็นชนิดใหน โดยตัวโปรแกรมจะตรวจสอบให้ และทำงานตามคุณสมบัติเฉพาะของการ์ดเหล่านั้น

การเขียนโปรแกรมเพื่อให้ได้คุณสมบัติดังที่กล่าวมาแล้ว ต้องมีความเข้าใจแผงวงจรหรือวงจรส่วนที่ทำหน้าที่เกี่ยวกับการแสดงผลไปยังจอภาพ(Adapter)แบบต่าง ๆ ก่อน ซึ่งในปัจจุบันมี Adapter การแสดงผลอยู่ 4 แบบคือ โมโนโครม, ซีจีเอ, อีจีเอ และ วีจีเอ โดย ซีจีเอ ผู้ผลิตได้เลิกผลิตแล้ว ส่วน 2 แบบหลัง สามารถแสดงผลได้หลายโหมดและแสดงสีได้หลายสี ดังแสดงในตารางที่ 4.2 ซึ่งในการใช้งานจะใช้โหมด 7 สำหรับจอโมโนโครม และโหมด 2 หรือ 3 สำหรับจอ อีจีเอ หรือ วีจีเอ

โหมด	แบบ	ขนาด	อะแดปเตอร์
0	Text, B/W	40x25	CGA, EGA
1	Text, 16 Colors	40x25	CGA, EGA
2	Text, B/W	80x25	CGA, EGA
3	Text, 16 Colors	80x25	CGA, EGA
4	Graphics, 4 Colors	320x200	CGA, EGA
5	Graphics, 4 Grey Tones	320x200	CGA, EGA
6	Graphics, B/W	640x200	CGA, EGA
7	Text, B/W	80x25	Monochrome
8	Graphics, 16 Colors	160x200	PCjr
9	Graphics, 16 Colors	320x200	PCjr
10	Graphics, 4 Colors ,16 Colors	640x200	PCjr, EGA
13	Graphics, 16 Colors	320x200	EGA
14	Graphics, 16 Colors	640x200	EGA
15	Graphics, 4 Colors	640x350	EGA

ตารางที่ 4.2 แสดงลักษณะการทำงานของแต่ละโหมดของอะแดปเตอร์แต่ละชนิด

โดยปกติหน่วยความจำ (Memory) แบบ RAM ของเครื่องคอมพิวเตอร์จะถูกสงวนไว้สำหรับเก็บ ลักษณะของตัวอักษรที่แสดงบนจอภาพส่วนหนึ่ง ซึ่งหน่วยความจำส่วนนี้ในจอ โมโนโครม จะเริ่มที่แอดเดรส B0000000H ส่วนจอภาพแบบอื่น ๆ จะเริ่มที่แอดเดรส B8000000H และการแสดงผลของตัวอักษรในโหมด 80x25 นั้น อักษรแต่ละตัวจะใช้หน่วยความจำ ไบต์ที่ 2 เป็นตัวเก็บลักษณะการแสดงสี (Attribute) ของตัวอักษร

หน่วยความจำที่สงวนไว้ สำหรับการแสดงผล โดยปกติจะมีไว้ใช้งานนอกเหนืออื่น ๆ ซึ่งมากกว่าที่จะใช้ในการแสดงผลแบบ 80x25 (Text Mode) และงานนอกเหนืออื่น ๆ นั้นได้แก่

1. เพื่อไว้ใช้ในการแสดงผลโหมดกราฟิก
2. เพื่อสามารถ เก็บการแสดงผลหลายหน้าจอไว้ในหน่วยความจำ และสามารถเลือกว่าจะนำหน้าจอไหนมาแสดง ซึ่งหน่วยความจำของแต่ละหน้าจอนี้เรียกว่า Video Page และโดยทั่วไปดอสใช้ Video Page 0

และการแสดงผลหรือการติดต่อกับส่วนแสดงผลนั้นสามารถทำได้ 3 วิธี คือ

1. การติดต่อผ่าน Function Call ของดอส ซึ่งการใช้วิธีนี้ ถ้านำมาเขียนโปรแกรมประเภทวินโดว์จะช้ามาก
2. การติดต่อผ่าน Function Call ของ Bios การทำงานมีความเร็วพอสมควร
3. การติดต่อโดยเขียน หรืออ่านข้อมูลบนหน่วยความจำสำหรับการแสดงผลโดยตรง ซึ่งวิธีนี้โปรแกรมจะทำงานได้เร็วที่สุด แต่การเขียนโปรแกรมจะยุ่งยากขึ้น และพัฒนาได้ล่าช้า

ซึ่งในการวิจัยในครั้งนี้ จะใช้วิธีการติดต่อกับส่วนแสดงผล โดยการติดต่อผ่าน Function Call ของ Bios เหตุผลเนื่องจาก โปรแกรมที่จัดทำต้องอยู่ในลักษณะฝังตัว จึงมีขนาดเล็กอยู่แล้ว และความสามารถในการติดต่อกับส่วนแสดงผลไม่จัดว่าช้า นอกจากนี้ ยังสามารถพัฒนาได้อย่างรวดเร็ว โดยลดข้อผิดพลาดที่อาจเกิดผลกระทบต่อโปรแกรมฝังตัวได้ด้วย