บทที่ 3

การเขียนโปรแกรมเพื่อเข้าถึงข้อมูลบนวีดีโอบลาสเตอร์ด้วยสิ่งแวดล้อม การทำงานของไมโครซอฟต์วินโดวส์

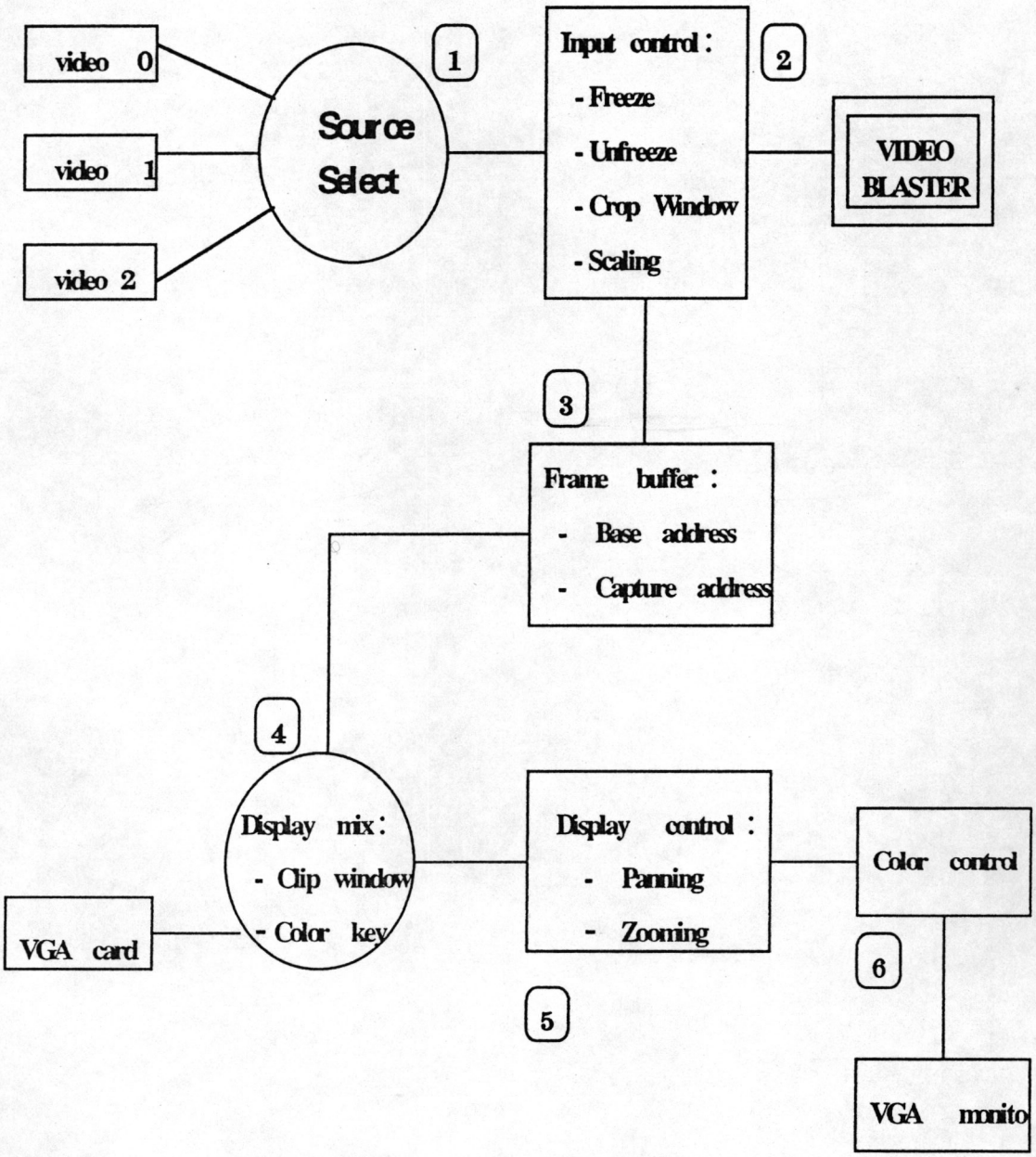
แผงวงจรวีดีโอบลาสเตอร์ (VIDEO BLASTER CARD)

วีดีโอบลาสเตอร์เป็นแผงวงจร [Video Blaster User Reference Manual,1992;Video Blaster Developer Kit,1992]สำหรับนำสัญญาณภาพจากเครื่องเล่นเทปมาซ้อนทับ (overlay) กับภาพกราฟิคที่ได้จากแผงวงจรวีจีเอเพื่อแสดงผลบนจอภาพวีจีเอ การทำงานของแผงวงจรนี้ถูกควบคุมโดยชิพ (chip) PCVIDEO ซึ่งผลิตโดย Chip and Technology

คุณสมบัติของแผงวงจรวีดีโอบลาสเตอร์ มีดังนี้

- รับสัญญาณวีดีโอได้ 3 สัญญาณ แต่แสดงผลได้ครั้งละหนึ่งสัญญาณ โดยสามารถสลับไปมา ระหว่าง 3 สัญญาณขณะแสดงผลได้
- สัญญาณวีดีโอ ที่ใช้ได้กับแผงวงจรมี 2 แบบคือ NTSC และ PAL
- นำสัญญาณภาพจากวีดีโอมาซ้อนทับกับภาพกราฟฟิคที่ได้จากแผงวงจรวีจีเอ
- ทำการย่อหรือขยายสัญญาณภาพ
- ทำการตัดหรือเลือกเฉพาะบางส่วนของสัญญาณภาพ
- แสดงผลของสัญญาณภาพที่ถูกย่อหรือขยาย
- แสดงผลของสัญญาณภาพที่ถูกตัดหรือเลือกเฉพาะบางส่วน
- สามารถควบคุมหรือปรับปรุงค่าของสี(hue) ค่าความอิ่มตัวของแสง (saturation or purity) ค่าความสว่าง (brightness or luminance) และ ค่าคอนทราสท์ (contrast)
- แฟ้มข้อมูลที่แผงวงจรสนับสนุน ต้องมีนามสกุลของแฟ้มเป็น PCX TIF BMP MMP GIF และ TGA
- ในการติดตั้งแผงวงจร มีซอฟต์แวร์ให้ผู้ใช้สามารถเลือก port address และ IRQ
- หน่วยความจำของแผงวงจรจะถูกทำให้เสมือนเป็นหน่วยความจำของระบบโดยมีตำแหน่งของหน่วยความจำอยู่เหนือหน่วยความจำของระบบ

การทำงานของแผนผังต่อไปนี้ แสดงการไหลของข้อมูลภาพจากเครื่องเล่นเทป บันทึกภาพผ่านขั้นตอน ขบวนการต่างๆ และแสดงผลบนจอภาพวีจีเอ



รูปที่ 3.1 แสดงการไหลของข้อมูลภาพ

ขั้นตอนต่างๆ ที่แสดงในรูปที่ 3.1 อธิบายได้ดังนี้

1. แผงวงจรวีดีโอบลาสเตอร์จะรับข้อมูลในที่นี้คือสัญญาณวีดีโอ ได้จาก 3 แหล่งข้อมูล โดยสามารถเลือกได้ว่า จะรับสัญญาณวีดีโอจากแหล่งข้อมูลใด เช่น video0 video1 หรือ video2

2. สัญญาณวีดีโอจากแหล่งข้อมูลที่เลือกจะถูกส่งผ่านฟังก์ชันต่างๆ เช่น การทำให้ภาพหยุดนิ่งหรือเคลื่อนไหว การย่อหรือขยายสัญญาณภาพ การเลือกฟังก์ชันเพื่อจัดการกับสัญญาณภาพขึ้นอยู่กับลักษณะของงานนั้นๆ

3. ขั้นตอนนี้ เป็นการบันทึกข้อมูลลงบนหน่วยความจำ ของแผงวงจรวีดีโอบลาสเตอร์และสามารถกำหนดได้ว่าจะให้เริ่มบันทึกลงบนตำแหน่งใดของหน่วยความจำ โดยข้อมูลภาพจะถูกบันทึกในรูปแบบ YUV [Video Blaster Developer Kit,1992]

4. ข้อมูลจากหน่วยความจำของแผงวงจรวีดีโอบลาสเตอร์และแผงวงจรวีจีเอ จะถูกนำมารวมกันที่แผงวงจรวีดีโอบลาสเตอร์ก่อนแล้วจึงส่งข้อมูลภาพนี้ไปแสดงบนจอภาพวีจีเอ โดยจะมีฟังก์ชันต่าง ๆ ทำหน้าที่จัดการว่าจะแสดงภาพที่มีลักษณะสีอย่างไร บนบริเวณใด ของจอภาพ

5. การแสดงภาพบนบริเวณใดๆของจอภาพวีจีเอนั้น ภาพที่แสดงอาจเป็นภาพที่ถูกย่อ หรือขยายแล้วก็ได้

6. ข้อมูลภาพที่แสดงบนจอภาพวีจีเอ จะผ่านขบวนการเกี่ยวกับสี (color processing) ก่อน เช่น การปรับปรุงค่าความอิ่มตัวของแสง ค่าความสว่าง เป็นต้น

จากลักษณะการไหลของข้อมูลภาพดังกล่าว ผู้ที่ต้องการทำงานกับแผงวงจรวีดีโอบลาสเตอร์จริงๆ ต้องทดลองเขียนโปรแกรมง่ายๆขึ้นมาเพื่อทดสอบการทำงานของแผงวงจร จะทำให้เข้าใจ ลักษณะการทำงานของแผงวงจรยิ่งขึ้นและที่สำคัญสามารถเลือกใช้ฟังก์ชันต่างๆ เพื่อจัดการกับสัญญาณภาพ ได้อย่างเหมาะสมกับลักษณะของแต่ละงาน

รูปแบบข้อมูลภาพบนหน่วยความจำ (Framebuffer Format)

แผงวงจรวีดีโอบลาสเตอร์ มีหน่วยความจำ 768 กิโลไบต์ สำหรับบันทึกสัญญาณวีดีโอหรือ ข้อมูลภาพหนึ่งภาพ หน่วยความจำของแผงวงจรถูกเรียกว่าเฟรมบัฟเฟอร์ (frame buffer) และเรียกข้อ-มูลภาพ 1 ภาพว่าวีดีโอเฟรม (video frame) หน่วยความจำ 768 กิโลไบต์ จะถูกทำให้เสมือนว่าเป็นหน่วยความจำของระบบ มีขนาดเท่ากับ 1 เมกะไบต์ ซึ่งซีพียู (CPU) ของเครื่องคอมพิวเตอร์สามารถอ้างถึงตำแหน่งต่างๆ เพื่อทำการประมวลผล เช่น อ่านหรือเขียนข้อมูลบนหน่วยความจำ 1 เมกะไบต์ นี้ได้ โดยทั้งนี้เพื่อไม่ให้เกิดการขัดแย้งหรือสับสนกันว่าซีพียูต้องการทำงานบนหน่วยความจำของแผงวงจรวีดีโอบลาสเตอร์หรือทำงานบนหน่วยความจำของระบบ การติดตั้งและใช้งานจึงจำเป็นต้องกำหนดให้หน่วยความจำของแผงวงจรวีดีโอบลาสเตอร์อยู่เหนือหน่วยความจำของระบบ เพราะถ้าเกิดการสับสนแล้วการทำงานของระบบอาจหยุดชงักไม่สามารถทำงานต่อไปได้

หน่วยความจำของแผงวงจรวีดีโอบลาสเตอร์ จะบันทึกข้อมูลภาพในรูปแบบ YUV โดยกำหนดให้ Y แทนค่าลิวมิแนนส์ (luminance) ขณะที่ UV แทนค่าโครมิแนนส์ (chrominance) และรูปแบบ YUV สามารถเปลี่ยนให้เป็นรูปแบบ RGB ได้ (ดูรูปที่ 2.6) ข้อมูลภาพรูปแบบ YUV จะมีจำนวนบิต (bit) ที่ใช้แทนแต่ละค่าของ YUV มีขนาดเป็น 7 บิต [Y7, U7, V7] แสดงว่าแต่ละพิกเซลของข้อมูลภาพมี 21 บิต ซึ่งสามารถใช้แสดงค่าสีต่างๆ ได้ถึง 2 ล้านสี อย่างไรก็ตามรูปแบบ YUV ได้ถูกกำหนดให้มีอัตราส่วนของจำนวนบิตที่ใช้แทนแต่ละค่าของ YUV เป็น 4:1:1 นั่นคือ Y มีได้ 4 ค่า ขณะที่ U และ V มีได้เพียงค่าเดียว โดยที่จำนวนบิตที่ใช้แทนค่า UV ค่าเดียวกัน จะกระจายอยู่ใน 4 พิกเซลของข้อมูลภาพ ทั้งนี้เพราะการสร้างภาพจำลองให้มีลักษณะสีเหมือนจริงนั้น สีที่แสดงในภาพจำลองมีไม่กี่สี แต่มีความเข้มสีที่แตกต่างกันหลายค่าสำหรับแต่ละสีและดวงตาของมนุษย์จะตอบสนองต่อการเปลี่ยนแปลงของความเข้มสีได้ดีกว่าการเปลี่ยนแปลงของค่าสี ดังนั้นจำนวนบิตที่ใช้แทนค่า Y จึงมีมากกว่าค่า UV

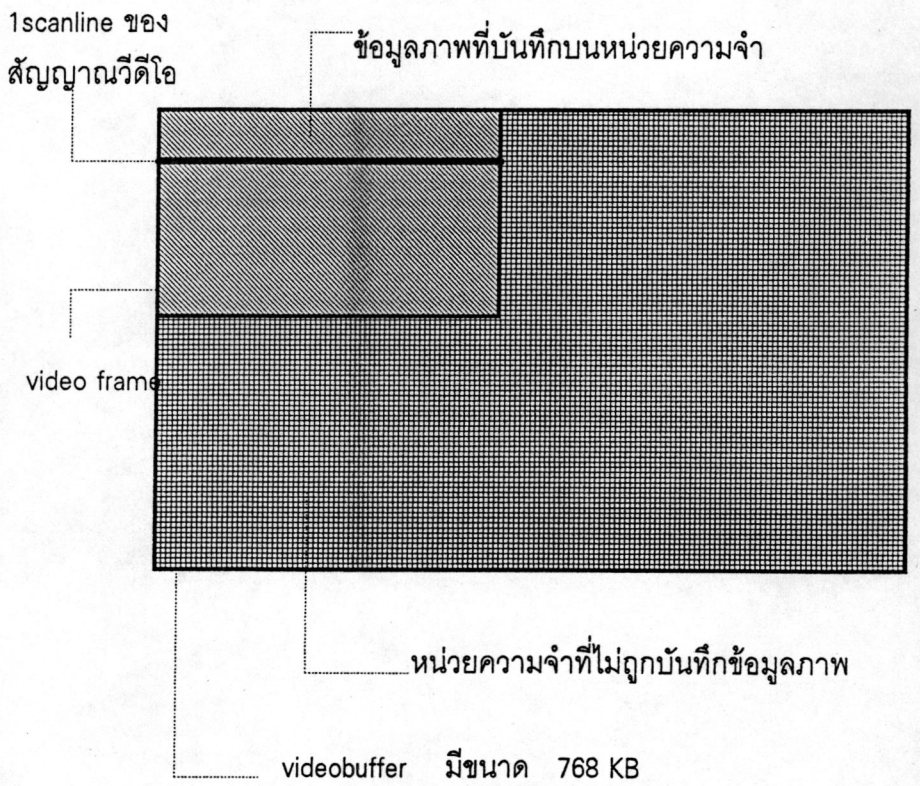
เพื่อความสะดวกและเหมาะสมในการนำข้อมูลภาพ ที่มีรูปแบบ YUV จากหน่วยความจำของแผงวงจรไปใช้งาน แผงวงจรวีดีโอบลาสเตอร์จะทำการบันทึกข้อมูลภาพรูปแบบ YUV บนหน่วยความจำของแผงวงจรวีดีโอบลาสเตอร์ โดยกำหนดให้แต่ละพิกเซลของข้อมูลภาพมี

ขนาด 16 บิต โดยในการใช้งานจริงๆ จะใช้เพียง 11 บิตเท่านั้น บิต 0 และ บิตที่ 8 ถึงบิตที่ 11 จะไม่ถูกใช้งาน ดังตาราง 3.1

Pixel #	Byte Addr	D15							D8	D7		D0
0	8n+0	U6	U5	V6	V5	X	X	X	X	Y0 : 6	.....	Y0 : 0X
1	8n+2	U4	U3	V4	V3	X	X	X	X	Y1 : 6	.....	Y1 : 0X
2	8n+4	U2	U1	V2	V1	X	X	X	X	Y2 : 6	.....	Y2 : 0X
3	8n+6	U0	X	V0	X	X	X	X	X	Y3 : 6	.....	Y3 : 0X

ตาราง 3.1 แสดงบิตที่ใช้แทนค่า UV ที่กระจายอยู่ใน 4 พิกเซล

การบันทึกข้อมูลภาพลงบนวีดีโอบัฟเฟอร์ จะทำการบันทึกในลักษณะสแคนไลน์-ไบ-สแคนไลน์ (scanline by scanline) โดย 1 สแคนไลน์ มีขนาดเท่ากับ 1024 พิกเซล (เท่ากับหน่วยความจำ 2 กิโลไบต์) ดังนั้นข้อมูลภาพ 1 สแคนไลน์ จึงมีขนาดเป็น 2 กิโลไบต์ รูปที่ 3.2 แสดงลักษณะของหน่วยความจำของแผงวงจรเมื่อถูกบันทึกข้อมูลภาพ



รูปที่ 3.2 แสดงลักษณะของหน่วยความจำของแผงวงจรเมื่อถูกบันทึกข้อมูลภาพ

การเข้าถึงข้อมูลภาพบนหน่วยความจำ (Accessing Video Buffer Memory)

ไมโครโปรเซสเซอร์ตระกูล 80x86 จะมีลักษณะการทำงาน 2 โหมดด้วยกัน คือ เรียลโหมด (real mode) และโพรเทคโหมด (protect mode) [Feraro,1988;Video Blaster Developer Kit,1992]

- เรียลโหมด การทำงานในโหมดนี้ส่วนใหญ่จะเป็นการพัฒนาโปรแกรมเพื่อทำงานภายใต้เอ็มเอส-ดอส (MS-DOS) และการแทนค่าตำแหน่งของหน่วยความจำเชิงตรรก (Logical) ประกอบด้วยค่าเซ็กเม็นท์ (segment) และค่าออฟเซ็ต (offset) แต่ละค่าแทนด้วยจำนวนบิท 16 บิต และถ้าให้ค่าเซ็กเม็นท์คูณด้วย 16 ก่อนที่จะนำไปบวกกับค่าออฟเซ็ทผลลัพธ์ที่ได้มีขนาด 20 บิต ใช้แทนค่าตำแหน่งของหน่วยความจำแบบเชิงเส้น (linear) ซึ่งในภาวะการทำงานแบบเรียลโหมดนี้ ตำแหน่งของหน่วยความจำแบบเชิงเส้น เป็นตำแหน่งเดียวกันกับหน่วยความจำแบบกายภาพ (physical) ซึ่งมีขนาดของหน่วยความจำใหญ่สุดเท่ากับ 1024 กิโลไบต์ (1 เมกะไบต์) หน่วยความจำขนาด 1 เมกะไบต์ นี้ ซีพียูสามารถกำหนดตำแหน่งต่างๆได้ใหญ่สุดขนาด 64 กิโลไบต์ ดังนั้นหน่วยความจำขนาด 1 เมกะไบต์ จึงเสมือนถูกแบ่งเป็น 16 ส่วน ๆ ละ 64 กิโลไบต์นั่นเอง

- โพรเทคโหมด การคำนวณหาค่าตำแหน่งของหน่วยความจำแบบเชิงเส้น ไม่สามารถทำได้ง่ายๆ เหมือนกับภาวะการทำงานแบบเรียลโหมด ทั้งนี้เพราะในภาวะการทำงานแบบโพรเทคโหมด ค่าเซ็กเม็นท์จะถูกกำหนดให้มีตำแหน่งเริ่มต้นอยู่ตำแหน่งใดก็ได้บนหน่วยความจำของระบบ ซึ่งมีขนาดสูงสุด 16 เมกะไบต์ และค่าเซ็กเม็นท์นี้กำหนดตำแหน่งหน่วยความจำได้ตั้งแต่ขนาด 1 ไบต์ ถึง 64 กิโลไบต์ ดังนั้น การหาตำแหน่งของหน่วยความจำแบบเชิงเส้น จึงต่างจากภาวะการทำงานแบบเรียลโหมด โดยที่ค่าเซ็กเม็นท์ถูกกำหนดให้เป็นตัวชี้ (pointer) ชี้ไปยังค่าๆ หนึ่งเรียกค่านี้ว่าตำแหน่งฐาน (base address) และค่าตำแหน่งฐานจะถูกนำมาบวกกับค่าออฟเซ็ส ส่วนค่าเซ็กเม็นท์หรือพอยน์เตอร์เรียกว่า ซิเลคเตอร์ (selector) โดยจำนวนบิตที่ใช้แทนค่าซิเลคเตอร์และออฟเซ็ตมี 16 บิต เท่ากัน ขณะที่ค่าตำแหน่งฐานใช้ 24 บิต ดังนั้นจำนวนบิตที่ใช้แทนตำแหน่งของหน่วยความจำแบบเชิงเส้น จึงมีขนาด 24 บิต

แผงวงจรวีดีโอบลาสเตอร์จะกำหนดให้เฟรมบัฟเฟอร์มีตำแหน่งอยู่เหนือหน่วยความจำของระบบและซีพียูสามารถกำหนดตำแหน่งของเฟรมบัฟเฟอร์ได้ในช่วง 1 ถึง 15 เมกะไบต์เท่านั้น หมายความว่าถ้าระบบมีหน่วยความจำ 4 เมกะไบต์ ตำแหน่งของเฟรมบัฟเฟอร์จะถูกกำหนดให้มีค่าอยู่ในช่วง 5 ถึง 15 เมกะไบต์ ซึ่งเรียกว่าตำแหน่งฐาน (Base Address) และตำแหน่งของหน่วยความจำที่อยู่ต่อจากตำแหน่งฐานขนาด 1 เมกะไบต์ คือหน่วยความจำที่เก็บข้อมูลภาพ (video buffer memory) ดังนั้นถ้าต้องการอ่านข้อมูลภาพต้องกำหนดตำแหน่งของหน่วยความจำที่เก็บข้อมูลภาพนี้ให้ได้ก่อน

เทคนิคการกำหนดตำแหน่งของหน่วยความจำที่เก็บข้อมูลภาพสามารถทำได้โดยถ้าเป็นการทำงานภายใต้เอ็มเอส-ดอสให้เรียกใช้ไบออสอินเตอร์รัพท์ (BIOS Interrupt) หมายเลข 15 H และฟังก์ชันหมายเลข 87 H แต่ถ้าเป็นการทำงานในสิ่งแวดล้อมของไมโครซอฟต์วินโดวส์ต้องทำการกำหนดซิเลคเตอร์ขึ้นมา ซึ่งมีขั้นตอนและวิธีการดังนี้

1. หาตำแหน่งฐาน โดยเรียกใช้ฟังก์ชัน vbcGetVideoAddress( ) จะให้ค่าหมายเลขของหน่วยความจำที่เป็นตำแหน่งเริ่มต้นของเฟรมบัฟเฟอร์

2. เปลี่ยนค่าหมายเลขของตำแหน่งฐานให้เป็นค่าของตำแหน่งของหน่วยความจำแบบ ฟิซซิคัลและเปลี่ยนตำแหน่งของหน่วยความจำแบบฟิซซิคัล ให้เป็นแบบล็อจจิคัล ซึ่งสามารถทำได้ โดยใช้ ไบออสอินเตอร์รัพท์ หมายเลข 31 H และ ฟังก์ชันหมายเลข 800 H

3. นำค่าตำแหน่งของหน่วยความจำแบบ logical มากำหนดช่วงของหน่วยความจำที่ซิเลคเตอร์สามารถกำหนดตำแหน่งได้ โดยเรียกใช้ฟังก์ชัน SetSelectorBase (#Selector, logical add) และฟังก์ชัน SetSelectorLimit (#Selector ,logical add)

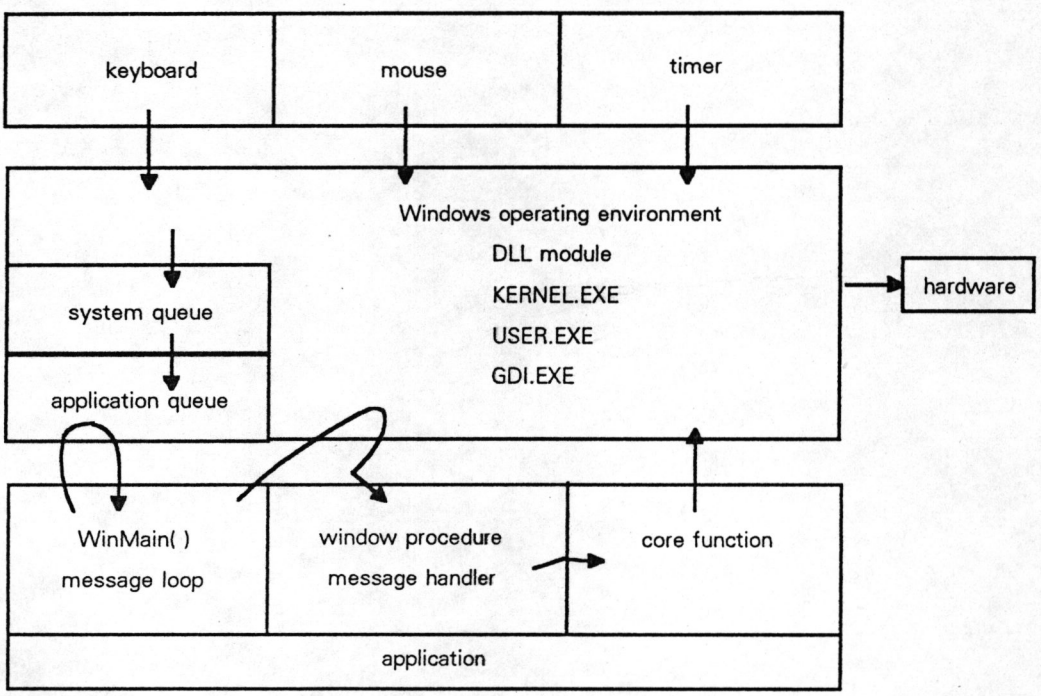
เนื่องจากเฟรมบัฟเฟอร์ของแผงวงจรวีดีโอบลาสเตอร์มีขนาด 768 กิโลไบต์ และซิเลคเตอร์ หนึ่งค่าสามารถกำหนดตำแหน่งของหน่วยความจำได้ขนาด 64 กิโลไบต์ ดังนั้นเพื่อความรวดเร็วในการอ่านข้อมูล จากเฟรมบัฟเฟอร์ จึงต้องใช้ ซิเลคเตอร์ทั้งหมด 12 ค่า (768 kB / 64kB = 12) แสดงการกำหนด ซิเลคเตอร์ 12 ค่า ได้ดังตัวอย่างโปรแกรมต่อไปนี้

```
/*-------------------------------------------------------------------------------------------------*/
        MACRO to create far pointer of specified selector & offset
        Define MK_FP(s,o) ((WORD far *)(((unsigned long) (s) << 16) | (0)))
        An undocument function about Selector
            - UINT SetSelectorBase(UINT sel, DWORD base);
            - void FAR PASCAL SetSelectorLimit(WORD sel, DWORD limit);
            - DWORD FAR PASCAL GetSelectorBase(WORD sel);
            - DWORD FAR PASCAL GetSelectorLimit(WORD sel);
        Function to create 12 selector that specified memory of
            VIDEOBLASTER buffer
        768 KB, return value is 0 if allocate successful.
/*-------------------------------------------------------------------------------------------------*/
#include <windows.h>
#include <dos.h>
#include "pcvideo.h"
#define NOSEL 12

int InitSelector(WORD *wSel)
{
        int  i;
        DWORD    dwBase, dwLogBase;
        WORD     sel, lBase, hBase;

        dwBase = vbcGetVideoAddress();
        dwBase = dwBase * 1024L * 1024L;
        _asm mov sel, ds

        /* in 386-Enhance mode physical != logical base address
           therefore must convert physical to logical before allocate selector
           use int 31h, function 800h
           NOTE: in Standard mode this function will return
           logical address as the same physical address */
```

```
    for (i=0;i<=NOSEL-1;i++)
    {
        lBase = LOWORD(dwBase);
        hBase = HIWORD(dwBase);
        _asm mov ax, 0x0800
        _asm mov cx, lBase
        _asm mov bx, hBase
        _asm mov di, 0x0000
        _asm mov si, 0x0001
        _asm int 31h
        _asm mov lBase, cx
        _asm mov hBase, bx


        /* find logical address of specify base physical memory */
        dwLogBase = MAKELONG(lBase, hBase);

        /* allocate selector for this 64K */
        if ((*(wSel+i) = AllocSelector(sel)) == 0)
            return i+1;

        /* set base address and limit of this selector */
        SetSelectorBase (*(wSel+i), dwLogBase);
        SetSelectorLimit(*(wSel+i), (unsigned long)64*1024-1);

        /* find next 64K of physical address */
        dwBase = dwBase + 65536;
    }
    return 0;
}
```

การทำงานของไมโครซอฟต์วินโดวส์

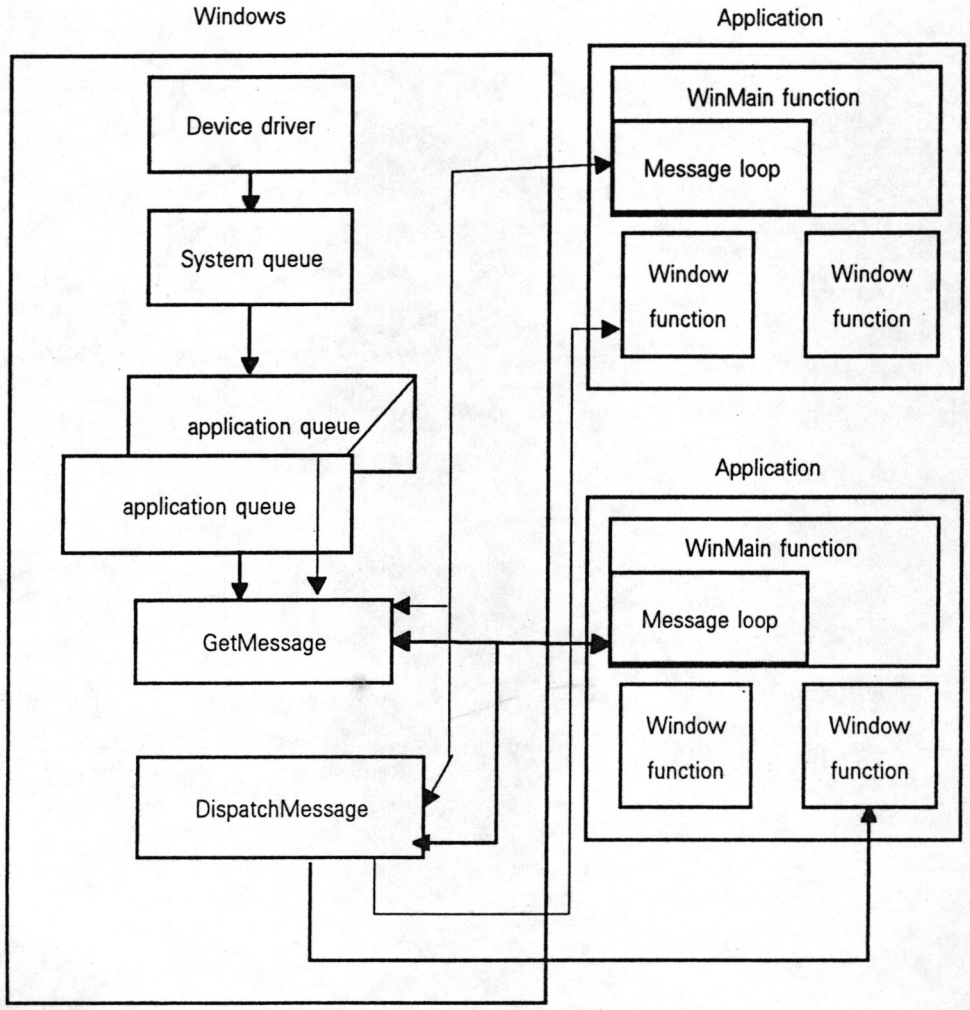
ซอฟต์แวร์ไมโครซอฟต์วินโดวส์เป็นซอฟต์แวร์ที่ช่วยให้การใช้งานเครื่องคอมพิวเตอร์เป็นไปอย่างมีประสิทธิภาพโดยไมโครซอฟต์วินโดวส์จะเป็นตัวเชื่อมต่อระหว่างโปรแกรมประยุกต์กับอุปกรณ์ฮาร์ดแวร์ โดยจะมีฟังก์ชันการติดต่อให้โปรแกรมประยุกต์ต่าง ๆ เรียกใช้ (Application Programming Interface) ดังรูปที่ 3.3



รูปที่ 3.3 แสดงการทำงานของไมโครซอฟต์วินโดวส์และโปรแกรมประยุกต์

ไมโครซอฟต์วินโดวส์จะเป็นตัวคอยจับสัญญาน จากอุปกรณ์นำเข้าข้อมูล (Input Hardware) อาทิเช่น แผงแป้นอักขระ เมาส์ นาฬิกา เมื่ออุปกรณ์เหล่านี้ให้กำเนิดสัญญานมา วินโดวส์ก็จะสร้างข้อความ (message) ที่เหมาะสมกับสัญญานนั้นๆเก็บไว้ในแถวคอยของระบบ (system queue) ตามหลักเข้าก่อนออกก่อน (first-in-first-out : FIFO) จากนั้นวินโดวส์จะพิจารณาว่าข้อความต่างๆ นั้นเป็นของโปรแกรมประยุกต์ใดแล้วจึงส่งข้อความนั้นไปยังแถวคอยของโปรแกรมประยุกต์นั้นๆ โดยที่แต่ละโปรแกรมประยุกต์ก็จะมีแถวคอยของโปรแกรมประยุกต์

(application queue) ของตนเองคอยรับข้อความที่ส่งมา จากนั้นโปรแกรมประยุกต์จึงนำข้อความที่ได้รับไปประมวลผลอีกทีหนึ่ง



รูปที่ 3.5 แสดงการไหลของข้อความ

จากรูปที่ 3.5 เมื่อเกิดข้อความเช่นผู้ใช้กดแผงแป้นอักขระ วินโดวส์จะเก็บข้อความไว้ที่แถวคอยของระบบและพิจารณาว่าเป็นข้อความของโปรแกรมประยุกต์ใด แล้วจึงส่งไปเก็บไว้ที่แถวคอยของโปรแกรมประยุกต์นั้น จากนั้นโปรแกรมประยุกต์จะรับข้อความมาตรวจสอบว่าเป็นข้อความการหยุดโปรแกรมหรือไม่ ถ้าใช่ก็จะหยุดการทำงานของโปรแกรมนั้น แต่ถ้าไม่ใช่ก็จะจัดส่งกลับไปยังวินโดวส์เพื่อให้วินโดวส์จัดส่งไปยังวินโดวส์ฟังก์ชันของโปรแกรมประยุกต์ที่เหมาะสมเพื่อประมวลผลข้อความนั้นตามต้องการต่อไป

ลักษณะการทำงานของวินโดวส์โดยใช้ข้อความนี้เอง ทำให้วินโดวส์สามารถทำงานได้หลาย งานพร้อมๆ กันโดยการสลับการทำงานของแต่ละโปรแกรมประยุกต์จะขึ้นอยู่กับข้อความ กล่าวคือถ้าโปรแกรมประยุกต์ที่กำลังทำงานอยู่ไม่มีข้อความในแถวคอย วินโดวส์ก็จะสลับการทำงานไปยังโปรแกรมประยุกต์หลังแทน ซึ่งการสลับการทำงานแบบนี้เรียกว่า jumpy multitasking หรือ nonpreemptive multitasking (Petzold, 1990)

ลักษณะโปรแกรมที่ทำงานบนไมโครซอฟต์วินโดวส์

โปรแกรมที่เขียนขึ้นสำหรับการทำงานบนไมโครซอฟต์วินโดวส์นั้น มี 2 ลักษณะดังนี้

1. Executable Program (.EXE files)

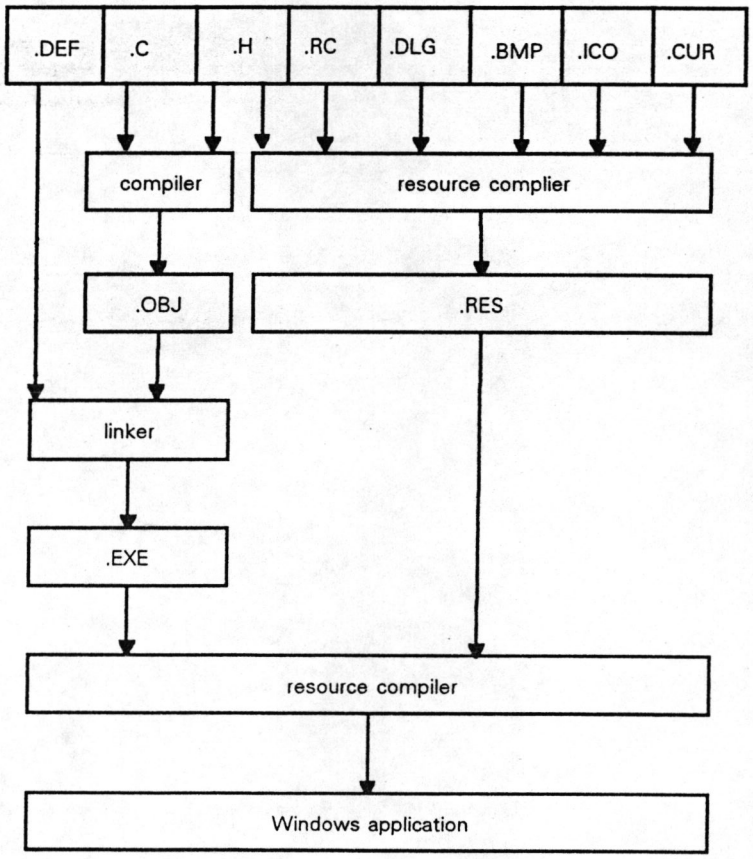
เป็นโปรแกรมที่สามารถทำงานได้ด้วยตนเองเช่นเดียวกับโปรแกรมต่างๆ ที่เขียนขึ้นบน ระบบปฏิบัติการดอสทั่วๆไป

2. Dynamic Link Libraries (.DLL files)

เป็นโปรแกรมชนิดหนึ่งที่ไม่สามารถทำงานได้ด้วยตนเอง กล่าวคือโปรแกรมนี้จะเป็นการรวบรวมฟังก์ชันการทำงานต่างๆไว้ ให้โปรแกรมประเภทแรกเรียกใช้ ความแตกต่างที่สำคัญของโปรแกรมประเภทนี้ก็คือ หน่วยความจำที่ใช้ เนื่องจากโปรแกรมที่ทำงานบนไมโครซอฟต์วินโดวส์สามารถทำงานได้พร้อมๆกัน วินโดวส์จะจัดสรรหน่วยความจำให้ทุกครั้งที่มีการเรียกใช้โปรแกรมชนิดแรก นั่นคือแต่ละโปรแกรมจะมีหน่วยความจำของตนเอง ในขณะที่โปรแกรมประเภท DLL นี้ วินโดวส์จะจัดสรรหน่วยความจำให้เพียงชุดเดียวและถูกเรียกใช้จากโปรแกรมหลายๆโปรแกรมได้ ทำให้สามารถประหยัดหน่วยความจำและขนาดของโปรแกรม .EXE นั้นก็จะมีขนาดเล็กลงอีกทั้งการแก้ไขฟังก์ชันการทำงานต่างๆ ก็ทำกับโปรแกรมประเภท .DLL นี้เพียงอย่างเดียว ไม่ต้องไปแก้ไโปรแกรมแบบ .EXE อีก

การเขียนโปรแกรมเพื่อทำงานบนไมโครซอฟต์วินโดวส์

หลักการเขียนโปรแกรมเพื่อทำงานบนไมโครซอฟต์วินโดวส์อยู่บนพื้นฐานของการเขียนโปรแกรมแบบบมัลติโมดูล (mutimodule programimng) กล่าวคือจะประกอบด้วยข้อมูลประเภทต่างๆ ที่ถูกแปลและนำมาเชื่อมต่อกันเป็นโปรแกรมประเภทที่สามารถทำงานได้ โดยแฟ้มข้อมูลต่างๆ มีความสัมพันธ์กัน (ดังรูปที่ 3.6) ดังนี้



รูปที่ 3.6 แสดงส่วนประกอบและขั้นตอนการสร้างโปรแกรม

1. แฟ้มข้อมูลต้นฉบับ (source .C file)

เป็นแฟ้มข้อมูลที่เขียนขึ้นโดยใช้ภาษาคอมพิวเตอร์ เช่น ซี ประกอบด้วยฟังก์ชันต่างๆ ตามขั้นตอนการทำงานที่กำหนดโดยโครงสร้างของแฟ้มข้อมูลจะต้องประกอบด้วยส่วนสำคัญดังนี้

1.1 การ include แฟ้มข้อมูล windows.h ซึ่งเป็นแฟ้มข้อมูลส่วนหัวที่ขาดไม่ได้ เนื่องจากเป็นแฟ้มข้อมูลที่เก็บการประกาศฟังก์ชันต่างๆ ที่วินโดวส์เตรียมไว้ให้ใช้ ประเภทข้อมูล ค่าคงที่และโครงสร้างข้อมูลต่างๆ

1.2 ฟังก์ชัน WinMain เป็นฟังก์ชันที่เป็นจุดเริ่มต้นของการทำงานคล้ายกับฟังก์ชัน main ของโปรแกรมภาษาซีทั่วๆไป โดยจะต้องใช้ชื่อนี้เท่านั้นและการทำงานในฟังก์ชันนี้จะต้องประกอบด้วย

1.2.1 การลงทะเบียนวินโดว์คลาส (register window class) เป็นการกำหนดค่าให้กับตัวแปรโครงสร้างชนิด WINDCLASS เพื่อกำหนดรายละเอียดลักษณะพื้นฐานของวินโดว์ที่ต้องการสร้างในโปรแกรม เช่น ชื่อของ window procedure สัญรูปและตัวชี้ตำแหน่งของวินโดว์นั้น โดยการลงทะเบียนนี้จะกระทำเพียงครั้งแรกที่โปรแกรมถูกเรียกขึ้นมาทำงาน โดยถ้าก่อนหน้ามีโปรแกรมเดียวกันทำงานอยู่ก่อนแล้ววินโดว์ก็จะไม่ทำการลงทะเบียนวินโดว์คลาสอีก ซึ่งเป็นผลดีทำให้ไม่เปลืองเนื้อที่หน่วยความจำ

1.2.2 การสร้างวินโดวส์ (creating window) ในขั้นตอนนี้จะเป็นการสร้างวินโดว์หลักของโปรแกรมโดยจะมีการกำหนดชนิดของวินโดว์ที่ต้องการสร้าง ขนาดและตำแหน่งเริ่มต้นของวินโดว์ รายการเลือก การสร้างวินโดว์นี้ จะเป็นการสร้างขึ้นภายในตัวไมโครซอฟต์วินโดวส์เท่านั้นยังไม่ถูกแสดงผลบนจอภาพ จะต้องเรียกฟังก์ชัน Show Window และ Update Window โดยฟังก์ชัน Show window จะทำหน้าที่สร้างวินโดว์ตามลักษณะที่ต้องการบนหน้าจอและฟังก์ชัน Update Window จะทำหน้าที่ส่งข้อความ WM_PAINT ไปให้กับ window procedure เพื่อให้พื้นที่ภายในวินโดว์ถูกวาดตามที่โปรแกรมกำหนด

1.2.3 วังวนข้อความ (message loop) เป็นส่วนที่โปรแกรมจะเชื่อมต่อกับวินโดวส์ โดยจะเป็นการทำงานในลักษณะวนไปเรื่อยๆ จนกว่าต้องการจะจบการทำงานของโปรแกรม ภายในวังวนจะมีฟังก์ชันการทำงานสองฟังก์ชันได้แก่ Translate Message ที่ทำหน้าที่แปลงข้อความเกี่ยวกับ แผงแป้นอักขระให้มีค่าที่สามารถนำไปใช้งานได้และ Dispatch Message ที่ทำหน้าที่ส่งข้อความกลับไปยังวินโดวส์เพื่อให้วินโดวส์จัดการส่งไปยัง window function ต่อไป

1.2.4 Window procedure ในส่วนนี้ถือว่าเป็นส่วนสำคัญของการเขียนโปรแกรมเพื่อทำงานบนไมโครซอฟต์วินโดวส์ เพราะว่าเป็นส่วนที่ทำหน้าที่ประมวลผลข้อความต่างๆที่ถูกส่งมา โดยชื่อของฟังก์ชันจะต้องกำหนดไว้ในขั้นตอนการลงทะเบียนวินโดว์คลาส เพื่อให้วินโดวส์รู้ว่าต้องส่งข้อความไปประมวลผลที่ฟังก์ชันใด โดยทั่วไปรูปแบบภายในของฟังก์ชันจะอยู่ในรูปของการใช้คำสั่ง switch - case statement เพื่อเลือกเอาข้อความไปประมวลผลเฉพาะ

ตามต้องการ ส่วนข้อความที่ไม่ต้องการจะต้องถูกส่งไปที่ DefWindowProc ซึ่งเป็นฟังก์ชันภายในวินโดวส์ที่ถูกกำหนดให้ทำงานกับข้อความต่างๆ ที่ไม่ได้ถูกประมวลผล ซึ่งค่าที่ส่งกลับจาก DefWindowProc นี้จะถูกส่งกลับคืนสู่วินโดวส์อีกทีหนึ่ง

ตัวอย่างของแฟ้มข้อมูลแสดงได้ดังนี้

```
/* Example.C file */
# include <windows.h>

long FAR PASCAL WndProc(HWND, WORD, WORD, WORD);
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstancd,
                   LPSTR lpszCmdParam, int nCmdShow)
{
HWND        hWnd;
MSG         msg;
WNDCLASS    wnclass;

if (!hPrevInstance)
        {
        wndclass.style          = CS_HREDRAW | CS_VREDRAW;
        wndclass.lpfnWndProc    = WndProc;
        wndclass.cbClsExtra     = 0;
        wndclass.cbWndExtra     = 0;
        wndclass.hInstance      = hInstance;
        wndclass.hIcon          = LoadIcon(NULL, IDI_APPLICATION);
        wndclas.,hCursor        = LoadCursor(NULL,IDC_ARROW);
        wndclass.hbrBackgroung  = GetStockObject(WHITE_BRUSH);
        wndclass.lpszMenuName   = NULL;
        wndclass.lpszclassName  = "Example";

        RegisterClass(&wndclass);
        }
```

```
hWnd = CreateWindow ("Example", "Example source file",
            WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,CW_USEDEFAULT,
            CW_USEDEFAULT, CW_USEDEFAULT,
            NULL, NULL, hInstance, NULL);

ShowWindow (hWnd, nCmdShow);
UpdateWindow (hWnd);
while (GetMessage(&msg,NULL,0,0))
            {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
            }
return msg.wParam;
}


long FAR PASCAL WndProc(HWND hWnd,WORD message,WORD wParam,LONG
lparam)
{
HDC             hDC;
PAINTSTRUCT ps;

switch (message)
            {
            case WM_PAINT;
            hDC = BeginPaint(hWnd, &ps);
            Endpaint(hWnd, &ps);
            return 0;
            }
return DefWindowProc(hWnd, message, wParam,lParam);
}
```

2. Module definition file (.DEF file) เป็นแฟ้มข้อมูลที่ช่วยในขั้นตอนการเชื่อมต่อโปรแกรมโดยประกอบด้วยข้อมูลทางเทคนิคของโปรแกรมที่สร้างขึ้น เช่น ขนาดของสแตก ขนาดของหน่วยความจำ ขณะเริ่มต้นโปรแกรม ชื่อของโปรแกรมที่จะถูกเรียกให้ทำงาน คุณสมบัติของ code segment และ data segment รวมทั้งชื่อของ window procedure ทั้งหมดตัวอย่างของ module definition file แสดงได้ดังนี้

```
/* example of .DEF file */

NAME        EXAMPLE
DESCRIPTION 'example of module definition file'
EXETYPE     WINDOWS
STUB        'WINSTUB.EXE'
CODE        PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE MULTIPLE
HEAPSIZE    1024
STACKSIZE   8192
EXPORTS     WndProc
```

3. แฟ้มข้อมูลส่วนหัว (.H file) เป็นแฟ้มข้อมูลส่วนหัวเช่นเดียวกับการเขียนโปรแกรมภาษาซีทั่วไป

4. Resource script file (.RC file) เป็นแฟ้มข้อมูลที่ใช้กำหนดการใช้ทรัพยากรต่างๆ ของ โปรแกรมประกอบด้วย การบรรยายการสร้างรายการเลือกและแป้นลัด ข้อความ ชื่อและชนิดของแฟ้มข้อมูลแบบแผนที่บิต แฟ้มข้อมูลสัญรูป แฟ้มข้อมูลตัวชี้ตำแหน่ง รวมทั้งแฟ้มข้อมูลกล่องคำโต้ ตอบ(dialog box) ต่างๆ ตัวอย่างของแฟ้มข้อมูลแสดงได้ดังนี้

```
/* example of .RC file */
# include <windows.h>
cur1        CURSOR          cursor.cur
ico1        ICON            icon.ICO
```

```
bmp1        BITMAP                  bmp.BMP
Example MENU
{
POPUP "&File"
                        {
                        MENUITEM "&New",   IDM_NEW
                        MENUITEM "&Save",IDM_SAVE
                        MENUITEM "&Quit",   IDM_QUIT
                        }
}
```

5. แฟ้มข้อมูลกล่องคำโต้ตอบ (.DLG file) เป็นแฟ้มข้อมูลที่ใช้ในการบรรยายการสร้างกล่องคำโต้ตอบต่างๆ

6. แฟ้มข้อมูลแผนที่บิต (.BMP file) เป็นแฟ้มข้อมูลรูปภาพแบบแผนที่บิตที่ใช้ในโปรแกรม

7. แฟ้มข้อมูลสัญรูป (.ICO file) เป็นแฟ้มข้อมูลรูปภาพสัญรูปที่ใช้ในโปรแกรม

8. แฟ้มข้อมูลตัวชี้ตำแหน่ง (,.CUR file) เป็นแฟ้มข้อมูลรูปภาพตัวชี้ตำแหน่งที่ใช้ในโปรแกรม

แฟ้มข้อมูลทั้งหมดนี้จะถูกแปลและนำมาเชื่อมต่อกัน จนกลายเป็นโปรแกรมที่สามารถทำงาน บนไมโครซอฟต์วินโดวส์ได้ สำหรับรายละเอียดปลีกย่อยอื่นๆ ตลอดจนเทคนิคขั้นสูง สามารถศึกษาเพิ่มเติมได้จากคู่มือสำหรับการเขียนโปรแกรมเพื่อใช้งานบนไมโครซอฟต์วินโดวส์ (Adam,1992; Petzold,1990;MicrosoftStaff,1990b, 1990c,1990d; Norton and Yao, 1990; Rector, 1992; Richter,1991)