บทที่ 3

ภาวะการทำงานของวินโดวส์

วินโดวส์ 3.1 มีภาวะการทำงานอยู่ 2 ภาวะคือ ภาวะมาตรฐาน (Standard Mode) และภาวะเสริม (Enhance Mode) โดยวินโดวส์จะเลือกภาวะการทำงานที่ให้ประสิทธิภาพสูงสุดโดยอัตโนมัติ ด้วยการตรวจสอบชนิดของหน่วยประมวลผลกลางและทรัพยากรอื่นๆของระบบ

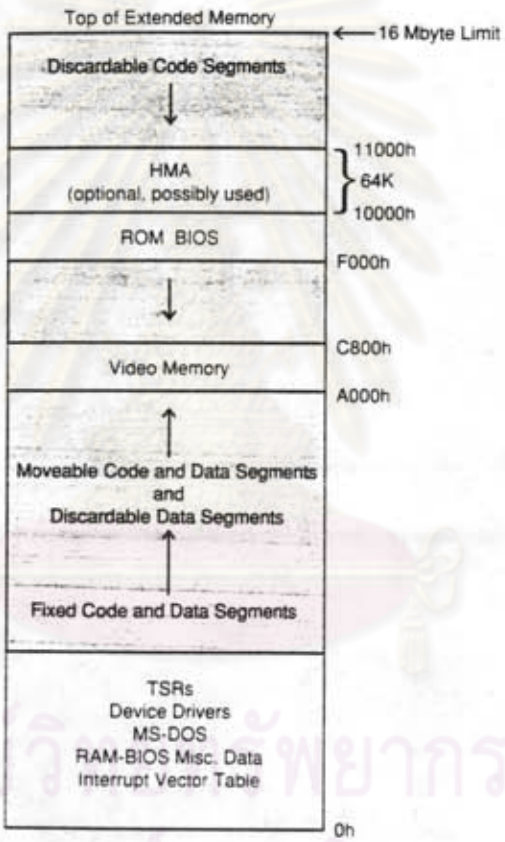
ทั้งภาวะมาตรฐานและภาวะเสริมทำงานโดยที่หน่วยประมวลผลกลางอยู่ในภาวะป้องกัน (Protected Mode) ทั้งนี้วินโดวส์ใช้การทำงานในภาวะป้องกันเพื่อทำให้อ้างแอดเดรสได้มากขึ้น อีกทั้งยังได้ประโยชน์จากระบบป้องกันต่างๆ ในภาวะป้องกันทำให้เสถียรภาพ และประสิทธิภาพของระบบดีขึ้น โดยวินโดวส์และโปรแกรมประยุกต์ส่วนใหญ่จะทำงานภายใต้ภาวะป้องกันแบบ 16 บิต ทั้งนี้โดยเหตุผลที่ต้องการให้โปรแกรมประยุกต์จำนวนมากที่ออกแบบมาในสมัยหน่วยประมวลผลกลางรุ่น 80286 ยังคงทำงานร่วมกับวินโดวส์ได้ แต่บางส่วนที่สำคัญที่ต้องทำงานด้วยประสิทธิภาพสูงเช่น หน่วยจัดการความจำเสมือน(Virtual Memory Manager : VMM) และภาคขับอุปกรณ์เสมือน (Virtual Device Driver : VxD) จะถูกออกแบบและทำงานในแบบ 32 บิตภาวะป้องกันซึ่งจะมีเฉพาะในหน่วยประมวลผลกลางรุ่น 80386 ขึ้นไปเท่านั้น

การทำงานของวินโดวส์ในภาวะมาตรฐาน

ภาวะมาตรฐานของวินโดวส์ ทำงานในลักษณะอ้างอิงสถาปัตยกรรมของหน่วยประมวลผลกลางรุ่น 80286 เป็นหลัก โดย DLL, ภาคขับอุปกรณ์ และโปรแกรมประยุกต์ของวินโดวส์ จะถูกประมวลผลในภาวะป้องกัน ขนาดของหน่วยความจำสูงสุดที่อ้างได้คือ 16 MB. (ลิเนียร์ แอดเดรส) ไม่สนับสนุนการทำฮาร์ดแวร์เสมือนและหน่วยความจำเสมือน ซึ่งทำให้การติดต่อกับฮาร์ดแวร์เป็นไปโดยตรงระหว่างโปรแกรมประยุกต์กับฮาร์ดแวร์นั้นๆ สำหรับการประมวลผลโปรแกรมประยุกต์ของดอส (DOS Application) หรือติดต่อกับภาคขับอุปกรณ์ของดอส และบริการต่างๆ ที่ถูกโหลดก่อนที่วินโดวส์จะทำงาน จะทำโดยการเปลี่ยนภาวะการทำงานไปเป็นภาวะ

จริงซึ่งมีผลทำให้วินโดวส์หยุดการทำงานชั่วคราว ในสภาวะเช่นนี้การทำงานทุกอย่างจะอาศัยระบบปฏิบัติการดอส.

โดยเหตุที่ว่าภาวะมาตรฐานไม่สนับสนุนการทำฮาร์ดแวร์เสมือน ดังนั้นเมื่อวินโดวส์ประมวลผลโปรแกรมประยุกต์ของดอส จึงประมวลผลได้เพียงโปรแกรมเดียวและต้องประมวลผลในภาวะเต็มจอเท่านั้น การจัดสรรหน่วยความจำ เมื่อวินโดวส์ทำงานในภาวะมาตรฐาน แสดงได้ดังรูปที่ 3.1

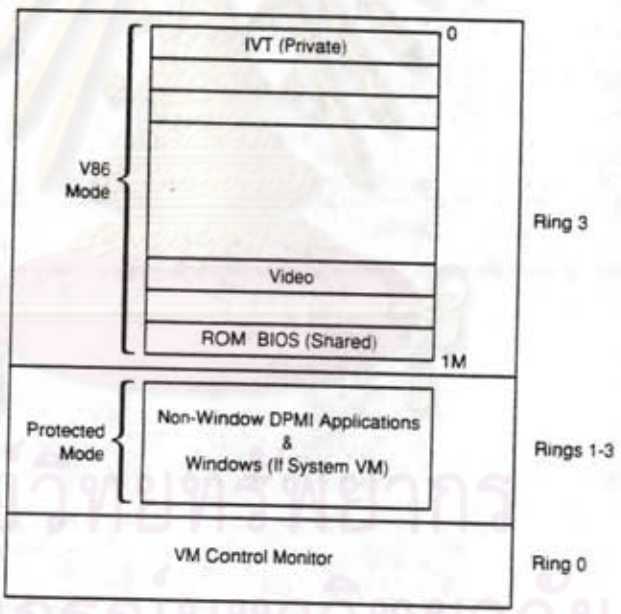


รูปที่ 3.1 แสดงการจัดสรรหน่วยความจำของวินโดวส์ในภาวะมาตรฐาน

ที่มา Klein, M. **Programmer's guide to DLLs and memory management** (Indiana:Sams publishing, 1992), p.66

การทำงานของวินโดวส์ในภาวะเสริม

วินโดวส์ในภาวะเสริมมีการดำเนินการที่ต่างๆ ออกไปโดยจะมีการสร้างเครื่องจักรเสมือน(Virtual Machine :VM) ให้กับโปรแกรมประยุกต์แต่ละกลุ่ม โครงสร้างของเครื่องจักรเสมือนแต่ละชุดจะประกอบด้วย ส่วนที่เป็นภาวะป้องกันและภาวะจริง (ภาวะV86) โดยที่ ภาวะ V86 จะทำหน้าที่จำลองการทำงานให้เสมือนเป็นพีซีคอมพิวเตอร์รุ่นเอ็กซ์ทีเครื่องหนึ่งซึ่งภายในประกอบไปด้วย หน่วยประมวลผลกลางรุ่น 808x อ้างแอดเดรสได้ 1 MB.ในรูปแบบเซกเมนต์: ออฟเซต และมีฮาร์ดแวร์เสมือนต่างๆ ของตัวเอง ข้อแตกต่างระหว่างภาวะจริง แท้ๆกับ ภาวะV86 ก็คือการติดต่อกับฮาร์ดแวร์ในภาวะV86 จะกระทำกับฮาร์ดแวร์เสมือนเท่านั้น (ฮาร์ดแวร์เสมือนเหล่านี้ได้แก่ พอร์ตสื่อสารทั้งขนานและอนุกรม รวมทั้งหน่วยความจำของการ์ดแสดงผล) ทั้งนี้เนื่องโปรแกรมใน ภาวะV86 ไม่มีระดับเอกสิทธิ์ (Privilege Level) สูงพอที่จะติดต่อกับอุปกรณ์ภายนอกได้โดยตรง โครงสร้างของเครื่องจักรเสมือนแต่ละชุดแสดงได้ดังรูปที่3.2



รูปที่ 3.2 แสดงผังโครงสร้างของเครื่องจักรเสมือนในวินโดวส์ในภาวะเสริม

ที่มา Klein, M. **Programmer's guide to DLLs and memory management** (Indiana:Sams publishing, 1992), p.64

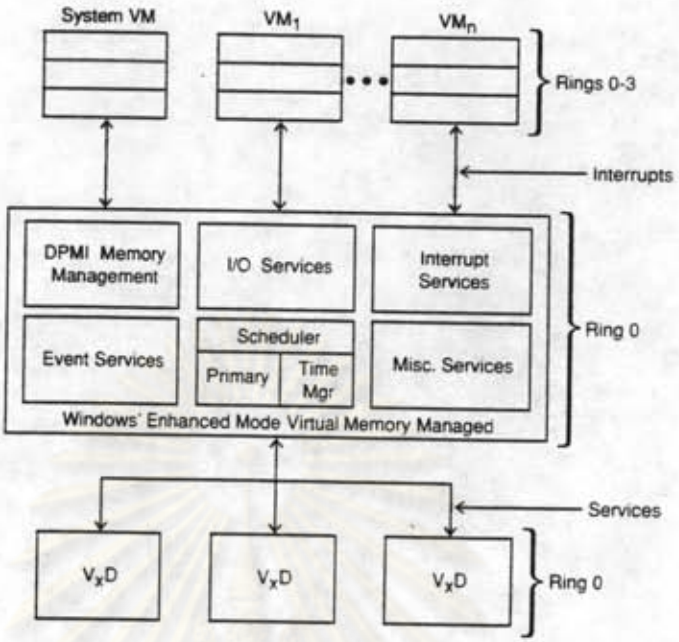
จะเห็นได้ว่าในเครื่องจักรเสมือนแต่ละชุด จะมีส่วนควบคุม (VM Control Monitor) ซึ่งทำงานที่เอกสิทธิ์ระดับ 0 คอยดูแลประสานงานระหว่างเครื่องจักรเสมือนด้วยกัน

สำหรับขั้นตอนการสร้างเครื่องจักรเสมือนนั้น วินโดวส์จะเริ่มต้นโดยการสร้างเครื่องจักรเสมือนระบบ (System VM) ขึ้นมาก่อนโดยในส่วนของ ภาวะV86 ของเครื่องจักรเสมือนระบบจะใช้สำหรับเก็บต้นฉบับของระบบปฏิบัติการดอสและสภาพแวดล้อมต่างๆก่อนที่วินโดวส์จะเริ่มทำงาน ข้อมูลในส่วนนี้นอกจากจะใช้ให้บริการแก่โปรแกรมประยุกต์ของวินโดวส์แล้ว ยังใช้ในการคัดลอกลงไปใน ภาวะV86 ของเครื่องจักรเสมือนที่จะสร้างขึ้นใหม่เพื่อใช้ประมวลผลโปรแกรมประยุกต์ของดอสทั้งนี้เพื่อให้ เครื่องจักรเสมือนของดอส ทุกชุดมีส่วนของระบบปฏิบัติการเป็นของตนเองแยกอิสระไม่รบกวนซึ่งกันและกัน ในส่วนภาวะป้องกันของเครื่องจักรเสมือนระบบ วินโดวส์ใช้สำหรับประมวลผลโปรแกรมประยุกต์ของวินโดวส์ และ DLL ที่เกี่ยวข้อง

โดยอาศัยหลักการของเครื่องจักรเสมือนทำให้วินโดวส์ในภาวะเสริมสามารถประมวลผลโปรแกรมประยุกต์ของดอสได้หลายโปรแกรมพร้อมกันอีกทั้งโปรแกรมประยุกต์ของดอสเหล่านี้ยังสามารถทำงานในลักษณะหลายภาระกิจอย่างแท้จริง (True Multitasking) ทั้งนี้เนื่องวินโดวส์จะใช้หลักการตัดตอน (Preemtive) ในการจัดสรรเวลาให้แก่เครื่องจักรเสมือนแต่ละชุด

ลักษณะที่น่าสนใจมากในวินโดวส์ในภาวะเสริมคือเครื่องจักรเสมือนแต่ละชุดจะมีตารางอินเตอร์รัพท์เวคเตอร์ (Interrupt Vector Table : IVT) และ ตารางอินเตอร์รัพท์เดสคริปเตอร์ (Interrupt Descriptor Table : IDT) เป็นของตัวเอง โปรแกรมในเครื่องจักรเสมือนแต่ละชุดสามารถทำการเปลี่ยนแปลงตารางเหล่านี้ของตนเองได้โดยไม่มีผลกระทบต่อเครื่องจักรเสมือนอื่นๆ อีกสิ่งหนึ่งที่วินโดวส์จัดการในภาวะเสริมคือการสร้างภาคขับอุปกรณ์เสมือน (VxD) ขึ้นมาทำหน้าที่จำลองฮาร์ดแวร์ให้กับเครื่องจักรเสมือนแต่ละชุด ซึ่งหมายความว่าการติดต่อกับอุปกรณ์ภายนอกของโปรแกรมประยุกต์เมื่อวินโดวส์ทำงานในภาวะเสริม ไม่ว่าจะเป็นโปรแกรมประยุกต์ของดอสหรือโปรแกรมประยุกต์ของวินโดวส์ จะเป็นการติดต่อกับ VxD ทั้งสิ้น โดย VxD จะเป็นผู้ตัดสินใจในการติดต่อฮาร์ดแวร์จริงอีกทอดหนึ่ง

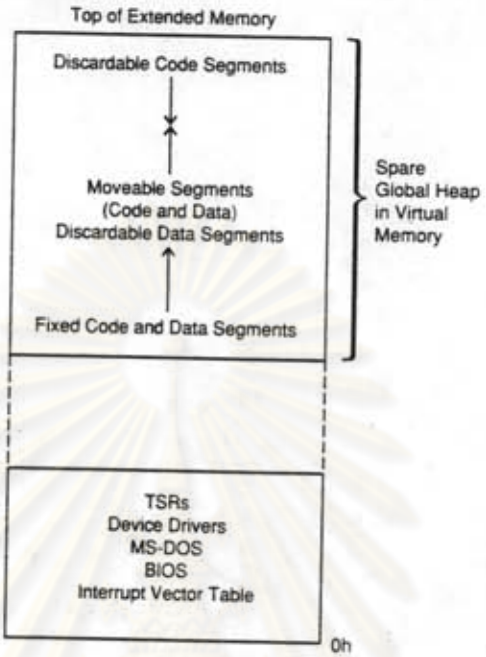
โดยใช้ VxD เราจึงสามารถประมวลผลโปรแกรมประยุกต์ของดอสได้ ทั้งในแบบแบ่งเป็นช่องหน้าต่าง (Window) หรือแบบเต็มจอ (Full Screen) อีกทั้งในการประมวลผลโปรแกรมประยุกต์ของดอสที่มีการใช้ฮาร์ดแวร์พร้อมกันหลายชุดก็สามารถทำได้เหมือนเรากำลังทำงานอยู่บนเครื่อง พีซีเอกซ์ที พร้อมกันหลายๆ เครื่อง ผังแสดงระบบโดยรวมเมื่อวินโดวส์ทำงานในภาวะเสริมแสดงในรูปที่ 3.3



รูปที่ 3.3 แสดงผังระบบโดยรวมในวินโดวส์ในภาวะเสริม

ที่มา Klein, M. **Programmer's guide to DLLs and memory management** (Indiana:Sams publishing, 1992), p.90

ในส่วนการจัดการบริหารหน่วยความจำของวินโดวส์ในภาวะเสริม แสดงได้ในรูปที่ 3.4



รูปที่ 3.4 แสดงการจัดสรรหน่วยความจำเมื่อวินโดวส์ทำงานในภาวะเสริม

ที่มา Klein, M. **Programmer's guide to DLLs and memory management** (Indiana:Sams publishing, 1992), p.67

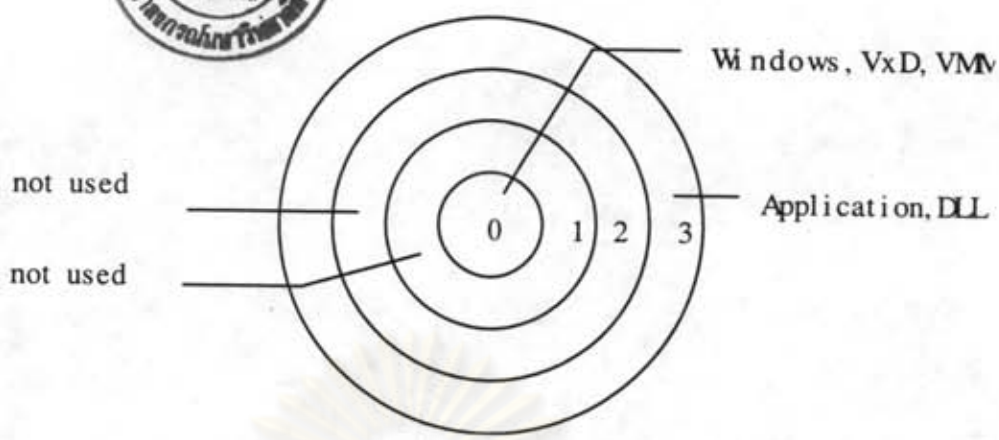
กลไกการป้องกันระบบของวินโดวส์

เนื่องจากวินโดวส์เป็นระบบจัดการแบบหลายภาระกิจ (Multitasking) จึงไม่สามารถยอมให้โปรแกรมหนึ่งโปรแกรมใดในระบบยึดครองทรัพยากรหรือรบกวนการทำงานของระบบปฏิบัติการได้วินโดวส์จึงต้องจัดวางระบบป้องกันเพื่อสร้างเสถียรภาพให้เกิดขึ้นในระบบ

กลไกป้องกันของหน่วยประมวลผลกลางในตระกูล 80x86 สร้างขึ้นโดยใช้กลไกดังต่อไปนี้

1. การจัดแบ่งหน่วยความจำ (Segmentation)
2. การเพจจิ้ง (Paging)
3. การกำหนดคำสั่งต้องห้าม (Restriction Op-Code)

กลไกทั้ง 3 ถูกใช้ในวินโดวส์ในลักษณะที่ต่างกัน ซึ่งรูปแบบของระดับการป้องกันที่ใช้ในวินโดวส์ 3.1 แสดงได้ดังรูปที่ 3.5



รูปที่ 3.5 แสดงการใช้วงแหวนระดับการป้องกันในระบบวินโดวส์

ที่มา Klein, M. **Programmer's guide to DLLs and memory management** (Indiana:Sams publishing, 1992), p.87

จากรูปหมายเลขน้อยแสดงระดับเอกสิทธิ์ที่สูงกว่าหมายเลขมาก,วงแหวนในสุด ระดับ 0 ใช้เป็นระดับการทำงานของระบบส่วนกลางและองค์ประกอบอื่นๆ ที่สำคัญ เช่น ภาคขับอุปกรณ์เสมือนและหน่วยจัดการหน่วยความจำเสมือน,ระดับ 1 และ ระดับ 2 ในวินโดวส์ 3.1 ไม่มีการใช้งานส่วน ระดับ 3 ซึ่งเป็นระดับเอกสิทธิ์ต่ำสุด วินโดวส์ใช้ในการประมวลผลโปรแกรมประยุกต์ต่างๆ (ทั้งวินโดวส์และดอส) รวมทั้งDLL ทุกชนิด

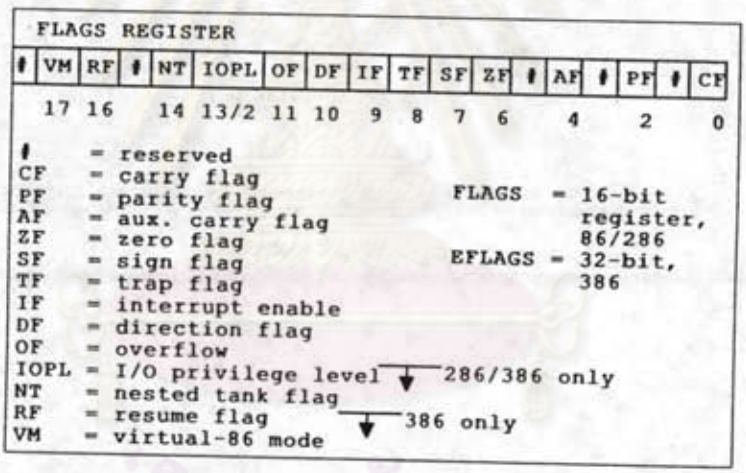
ในการบังคับใช้ระบบป้องกันโดยกลไกต่างๆ นั้นหน่วยป้องกันในหน่วยประมวลผลกลาง จะเป็นผู้ดำเนินการ โดยจะทำการตรวจสอบไปพร้อมกับการถอดรหัสคำสั่งและแปลงค่าแอดเดรส โดยวิธีนี้จะไม่ทำให้เกิดผลกระทบกับสมรรถนะของหน่วยประมวลผลกลางเลยแม้แต่น้อย หน่วยป้องกันจะคอยตรวจสอบและควบคุมพฤติกรรมของโปรแกรมประยุกต์ โดยจะตรวจสอบสิ่งต่อไปนี้

1. ตรวจสอบและควบคุมการใช้หน่วยความจำผิดประเภท (Type Check And Control) : เช่น อ่านอย่างเดียว (R/O) , เขียน-อ่าน (R/W) หรือ เอ็กซีคิวท์อย่างเดียว (X/O)
2. ตรวจสอบการใช้หน่วยความจำเกินขอบเขต (Limit Check)
3. ตรวจสอบหน่วยความจำที่ไม่มีสิทธิ์ใช้ (Restriction Of Addressable Domain) โดยดูจากระดับเอกสิทธิ์ ณ ขณะนั้น (Current Privilege Level : CPL) เทียบกับระดับเอกสิทธิ์ของ เดสคริปเตอร์ (Descriptor Privilege Level : DPL) และระดับเอกสิทธิ์ ของผู้ขอใช้ (Requestor Privilege Level : RPL)

4. ตรวจสอบการใช้ชุดคำสั่งต้องห้าม (Restriction Of Instruction Set)
5. ตรวจสอบการใช้บริการของระบบปฏิบัติการที่มีการสงวนสิทธิ์พิเศษ (Restriction Of Procedure Entry Point)

ในการป้องกันระบบอีกระดับหนึ่งคือการไม่ให้โปรแกรมประยุกต์ที่ไม่ใช่ส่วนของระบบปฏิบัติการมีสิทธิ์ในการติดต่อกับอุปกรณ์ภายนอกหรือทำการเปลี่ยนแปลงแฟลกที่สำคัญของระบบโดยตรง ซึ่งในการดำเนินการนี้ทำได้โดยการกำหนดระดับเอกสิทธิ์ในการใช้คำสั่งพิเศษ (IOPL Sensitive Instruction) และสงวนคำสั่งต้องห้าม (Privileged Instruction) ไว้ใช้เฉพาะโปรแกรมประยุกต์ที่มีระดับเอกสิทธิ์สูงสุด

ในการกำหนดระดับเอกสิทธิ์การใช้คำสั่งพิเศษ ทำโดยการกำหนดฟิลด์ IOPL ในแฟลกรีจิสเตอร์ให้มีค่าเท่ากับระดับเอกสิทธิ์ต่ำสุดที่จะยอมให้ใช้คำสั่งเหล่านี้ดังแสดงในรูปที่ 3.6



รูปที่ 3.6 แสดงแฟลกรีจิสเตอร์และตำแหน่ง IOPL ฟิลด์

ที่มา Kualer, B.**Windows assembly language and system programming** (New York:Prentice-Hall, 1993), p.174

วินโดวส์กำหนดระดับเอกสิทธิ์ของการใช้คำสั่งพิเศษไว้ที่ 0 (สูงสุด) ทำให้ไม่มีโปรแกรมประยุกต์ใดๆ ที่ทำงานในเอกสิทธิ์ระดับ 3 มีสิทธิ์ใช้คำสั่งพิเศษดังต่อไปนี้ได้โดยตรง

คำสั่ง	การกระทำ
CLI	ดิสเอเบิลอินเตอร์รัพท์
STI	อีนาเบิลอินเตอร์รัพท์
IN, INS	อ่านข้อมูลจากพอร์ต
OUT, OUTS	เขียนข้อมูลไปที่พอร์ต

ตารางที่ 3.1 แสดงชุดคำสั่งพิเศษ

ที่มา Kualer, B.**Windows assembly language and system programming** (New York:Prentice-Hall, 1993), p.175

นอกจากนี้ ยังมีคำสั่ง POPF และ POPFD ที่ถือว่าเป็นคำสั่งเฉพาะกิจ เพราะอาจมีผลทำให้แฟลกรีจิสเตอร์เปลี่ยนแปลงได้ หน่วยประมวลผลกลางจะไม่ทำงานตามที่ควรจะเป็นเมื่อพบคำสั่งนี้ในส่วนของโปรแกรมประยุกต์ที่ไม่ได้มีระดับเอกสิทธิ์เป็นศูนย์ (แฟลกรีจิสเตอร์จะไม่เปลี่ยนแปลง)

ในวินโดวส์เมื่อโปรแกรมประยุกต์ใดใช้คำสั่งในตารางที่3.1 จะทำให้เกิดการขัดจังหวะแบบพิเศษ (Exception) เพื่อกระตุ้นในส่วนของโปรแกรมป้องกันระบบ (Exception Handler) เริ่มทำงานแต่โปรแกรมนี้จะยอมให้คำสั่งเหล่านี้ประมวลผลต่อไปได้โดยมีข้อแม้ว่าต้องไม่มีการรบกวนกันระหว่างโปรแกรมประยุกต์ นั่นหมายความว่าวินโดวส์ยอมให้โปรแกรมประยุกต์ติดต่อกับอุปกรณ์ภายนอกได้ ภายใต้การดูแลของวินโดวส์ (กระบวนการนี้โปรแกรมประยุกต์จะไม่ทราบเลย) แต่คำสั่ง POPF และ POPFD จะมีผลดังเช่นที่กล่าวมาแล้ว ดังนั้นโปรแกรมประยุกต์ในวินโดวส์จึงไม่ควรใช้คำสั่งนี้ เพราะว่ามันจะไม่ทำงานอย่างที่ต้องการ ในกรณีต้องการเปลี่ยนแปลงค่าแฟลกรีจิสเตอร์ให้ดำเนินการโดยใช้วิธีการอื่นแทน

โดยอาศัยกลไกป้องกันชนิดต่างๆ เหล่านี้วินโดวส์จึงสามารถสร้างระบบฮาร์ดแวร์เสมือนเพื่อรองรับการทำงานแบบหลายภาระกิจได้

การจัดการอินเตอร์รัพท์ในวินโดวส์

วิธีจัดการอินเตอร์รัพท์ในวินโดวส์นั้นแตกต่างกับดอสอย่างมาก ทั้งนี้เนื่องจากวินโดวส์นั้นทำงานในภาวะป้องกัน อีกทั้งยังแยกภาวะการทำงานออกเป็น 2 ภาวะ ดังที่กล่าวมาในตอนต้น ซึ่งจะได้อธิบายดังนี้

1) การจัดการอินเตอร์รัพท์ในภาวะจริง

ก่อนที่จะอธิบายถึงกระบวนการจัดการอินเตอร์รัพท์ในวินโดวส์นั้น การทบทวนถึงกลไกการอินเตอร์รัพท์ทางซอฟต์แวร์ซึ่งเราคุ้นเคยกันดีในระบบดอส จะช่วยทำให้เราเข้าใจกระบวนการเหล่านั้นดีขึ้นโดยในระบบปฏิบัติการดอสจัดให้บริการต่างๆของระบบปฏิบัติการและไบออส เช่น บริการเกี่ยวกับการติดต่อกับอุปกรณ์อินพุต-เอาท์พุตของระบบปฏิบัติการ, บริการจัดการไฟล์, บริการจัดการหน่วยความจำ, บริการจัดการจอภาพและการ์ดแสดงผล และบริการอื่นๆ อีกหลายอย่าง มีการเชื่อมต่อกับผู้ใช้โดยผ่านทางคำสั่ง INT ซึ่งมีรูปแบบดังนี้

INT n

โดยที่ "n" เป็นจำนวนเต็มที่มีค่าตั้งแต่ 0-FFh โดยการผ่านพารามิเตอร์ จะทำโดยใช้รีจิสเตอร์ที่บริการนั้นๆ เป็นผู้กำหนด ซึ่งหลายๆ บริการก็ได้ทำการแบ่งส่วนออกเป็นบริการย่อย ซึ่งโดยมากระบุให้รีจิสเตอร์ AH เป็นหมายเลขบริการย่อยนั้นๆ

บริการของระบบปฏิบัติการที่สำคัญและถือเป็นศูนย์รวมของบริการทั้งหมดได้แก่ บริการ INT 21h ซึ่งภายในแบ่งออกเป็นบริการย่อยจำนวนมาก เพื่อให้ผู้ใช้ได้เรียกใช้ตามต้องการ (สำหรับผู้ใช้วินโดวส์ในการขอใช้บริการต่างๆ ของระบบจะทำโดยใช้คำสั่ง CALL ตามด้วยชื่อของฟังก์ชันบริการนั้นๆ รายละเอียดหาดูได้จากคู่มือ Application Programming Interface : API)

เมื่อมีการใช้คำสั่ง INT จะมีผลให้ หน่วยประมวลผลกลาง ตอบสนองดังนี้

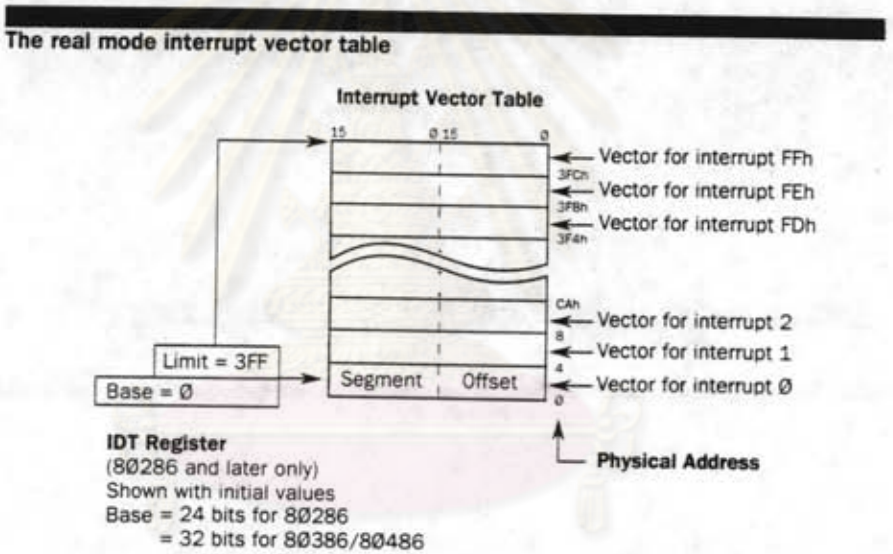
1. หน่วยประมวลผลกลาง จะเก็บค่ารีจิสเตอร์ตัวชี้คำสั่ง (Instruction Pointer : IP) และโค๊ดเซกเมนต์รีจีสเตอร์ (Code Segement: CS) พร้อมทั้งแฟลก รีจีสเตอร์ (Flag) ลงในสแตก
2. หน่วยประมวลผลกลาง จะใช้ค่า "n" เป็นตัวชี้ไปยังตารางอินเตอร์รัพท์เวคเตอร์ (Interrupt Vector Table : IVT) เพื่อนำค่าแอดเดรสของโปรแกรมบริการออกมา
3. หน่วยประมวลผลกลาง นำค่าแอดเดรสที่ได้มาใส่ลงในรีจิสเตอร์ CS และ IP แล้วเริ่มต้นประมวลผล ณ ตำแหน่งนั้น
4. ในโปรแกรมบริการจะจบด้วยคำสั่ง IRET ซึ่งมีผลทำให้ค่าของแสตก 3 รายการบนสุดถูกดึงขึ้น และนำไปใส่ยังรีจิสเตอร์ CS, IP และแฟลก ซึ่งทำให้หน่วยประมวลผล

กลาง ประมวลผลต่อไปจากจุดที่เคยถูกอินเตอร์รัพท์ได้, เป็นการจบการอินเตอร์รัพท์ครั้งนี้

2) การจัดการอินเตอร์รัพท์ในภาวะป้องกัน

ในภาวะป้องกันจะใช้ตารางอินเตอร์รัพท์ที่เดสคริปเตอร์ (Interrupt Descriptor Table : IDT) แทนตาราง IVT ซึ่งใช้ในภาวะจริง (สำหรับในวินโดวส์นั้นตาราง IVT จะยังคงถูกเก็บรักษาไว้ทั้งนี้เนื่องจากวินโดวส์ยังคงใช้ตารางนี้อยู่แต่โดยนัยอื่น)

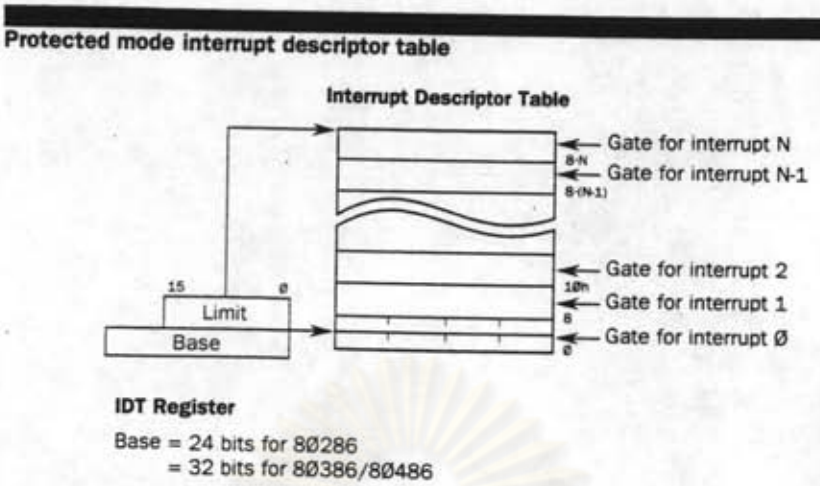
ตางราง IVT ถูกกำหนดให้อยู่ที่แอดเดรส 0000 : 0000 และกินพื้นที่ขนาด 1024 ไบต์ต่อเนื่องกัน ดังแสดงในรูปที่ 3.7



รูปที่ 3.7 แสดงโครงสร้างของตาราง IVT

ที่มา Hummel, R.L. **PC magazine technical reference. the processor and coprocessor** (California:Osborn McGraw-Hill,1988), p. 171

ค่าเริ่มต้นของ IVT จะถูกกำหนดโดยไบออส และดอสในระหว่างที่ระบบเริ่มทำงาน จะเห็นได้ว่า ภายในตารางบรรจุค่า เซกเมนต์ : ออฟเซต ของโปรแกรมบริการไว้ซึ่งเป็นสิ่งที่ใช้ไม่ได้ในภาวะป้องกัน (ภาวะV86ใช้ได้) ดังนั้นจึงมีการใช้ตาราง IDT แทน ดังแสดงในรูปที่ 3.8

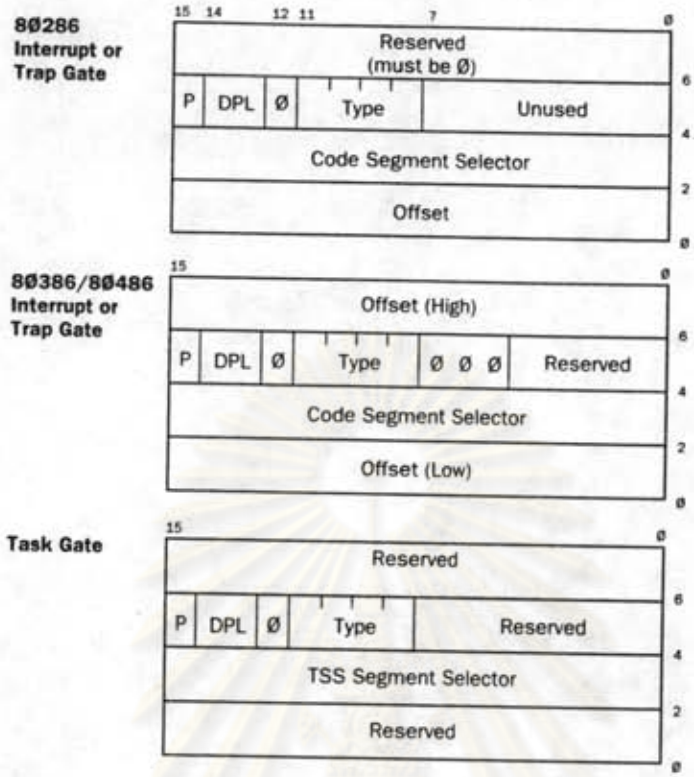


รูปที่ 3.8 แสดงโครงสร้างของตาราง IDT

ที่มา Hummel, R.L. **PC magazine technical reference. the processor and coprocessor** (California:Osborn McGraw-Hill,1988), p. 172

จากรูปแต่ละรายการของ IDT เรียกว่าเกต (GATE) ซึ่งมีจำนวนสูงสุด 256 รายการ (00-Ffh) เกตแต่ละชุดมีขนาด 8 ไบต์ใช้สำหรับบรรจุค่าลิเนียร์แอดเดรสของโปรแกรมบริการ ตำแหน่งที่อยู่ของ IDT สามารถกำหนดได้โดยใช้รีจิสเตอร์ IDTR, รูปแบบของเกตแสดงได้ดังรูปที่ 3.9

Interrupt descriptor formats



Field	Description
P	Present
DPL	Descriptor privilege level
Type	Value of the Type field determines the gate type as follows:

Value	Gate type
Ø1Ø1b	Task gate
Ø11Øb	16-bit (8Ø286) interrupt gate
Ø111b	16-bit (8Ø286) trap gate
111Øb	16-bit (8Ø386/8Ø486) interrupt gate
1111b	16-bit (8Ø386/8Ø486) trap gate

รูปที่ 3.9 แสดงรูปแบบของเกต

ที่มา Hummel, R.L. **PC magazine technical reference. the processor and coprocessor** (California:Osborn McGraw-Hill,1988), p. 174

เมื่อมีการอินเตอร์รัพท์เกิดขึ้นหน่วยประมวลผลกลาง จะทำการตอบสนองเช่นเดียวกับในภาวะจริงแต่ต่างกันที่ใช้ตาราง IDT แทน

3) การจัดการซอฟต์แวร์อินเตอร์รัพท์ในวินโดวส์

ถึงแม้ว่าวินโดวส์จะทำงานในภาวะป้องกัน แต่การใช้งานซอฟต์แวร์อินเตอร์รัพท์เพื่อติดต่อกับดอส หรือไบออส ส่วนใหญ่ยังคงทำงานได้เหมือนเดิม (แต่มีข้อต้องระวังอยู่บ้าง) นอกจากนั้นในส่วนของวินโดวส์เองก็ยังให้บริการพิเศษ ที่เรียกว่า DPMI (Dos Protected Mode Interface) รุ่น 0.9 ซึ่งเรียกใช้โดยผ่าน INT 2Fh และ INT 31h (Kualer, 1993)

จากเอกสารถึงผู้พัฒนาโปรแกรม (Developer's Notes) ของบริษัทไมโครซอฟต์ที่ได้เพิ่มเติมรายละเอียดสำคัญที่ว่า วินโดวส์ไม่รองรับการใช้บริการของดอสดังต่อไปนี้

INT 20h		Terminate program
INT 25h		Absolute disk read
INT 26h		Absolute disk write
INT 27h		Terminate and stay resident
INT 21h	AH	=
	00h	Terminate process
	0fh	Open file with FCB.
	10h	Close file with FCB.
	14h	Sequential read.
	15h	Sequential write.
	16h	Create file with FCB.
	21h	Random read.
	22h	Random write.
	23h	Get file size.
	24h	Set relative record.
	27h	Random block read.
	28h	Random block write.

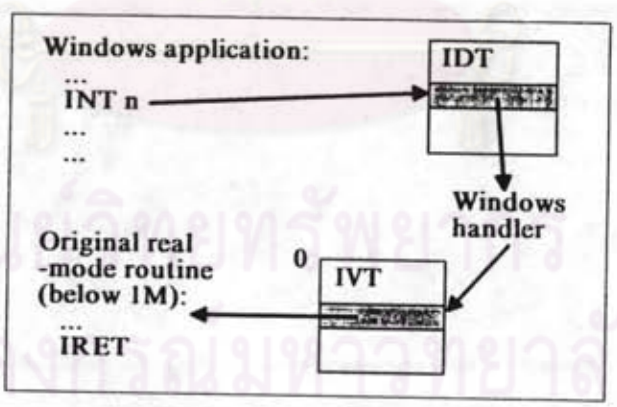
และบริการ INT 21h ต่อไปนี้ได้ถูกแก้ไขใหม่ให้ทำงานในภาวะป้องกัน ซึ่งจะทำให้ผลการทำงานไม่เหมือนกับในภาวะจริงดอส

AH = 25h / 35h	Get / Set interrupt vector.
AH = 38h	Get country data.
AH = 44h	Subfunction 02h, 03h, 04h and 05h.
AH = 65h	Get extended country information.

เอกสารฉบับดังกล่าวยังเพิ่มเติมอีกว่านอกจากบริการที่ระบุมานี้ โปรแกรมประยุกต์ที่ประมวลผลในวินโดวส์สามารถใช้บริการของดอสและไบออสได้ตามปกติ โดยวินโดวส์จะใช้วิธีเปลี่ยนภาวะการทำงานกับไปสู่ภาวะจริง (หรือ ภาวะV86 ในกรณีวินโดวส์ในภาวะเสริม) ทั้งนี้โดยการกำหนดค่าในตาราง IDT ให้ชี้ไปยังส่วนของโปรแกรมบริการพิเศษ (Special Handler) ที่จะทำการเปลี่ยนภาวะการทำงานของหน่วยประมวลผลกลาง ซึ่งเราเรียกการทำเช่นนี้ว่า การสะท้อนอินเตอร์รัพท์ (Interrupt Reflection)

4) การสะท้อนอินเตอร์รัพท์

เนื่องจากวินโดวส์มิได้จัดให้มีการบริการของดอส และไบออส ในภาวะป้องกันทุกๆ บริการ ดังนั้นการให้บริการที่นอกเหนือไปจากที่กล่าวมาแล้วนั้น จะทำโดยดอส หรือไบออสเดิม (ภาวะจริง) วินโดวส์จึงจึงต้องทำการเปลี่ยนภาวะการทำงานดังแสดงได้ในรูปที่ 3.10

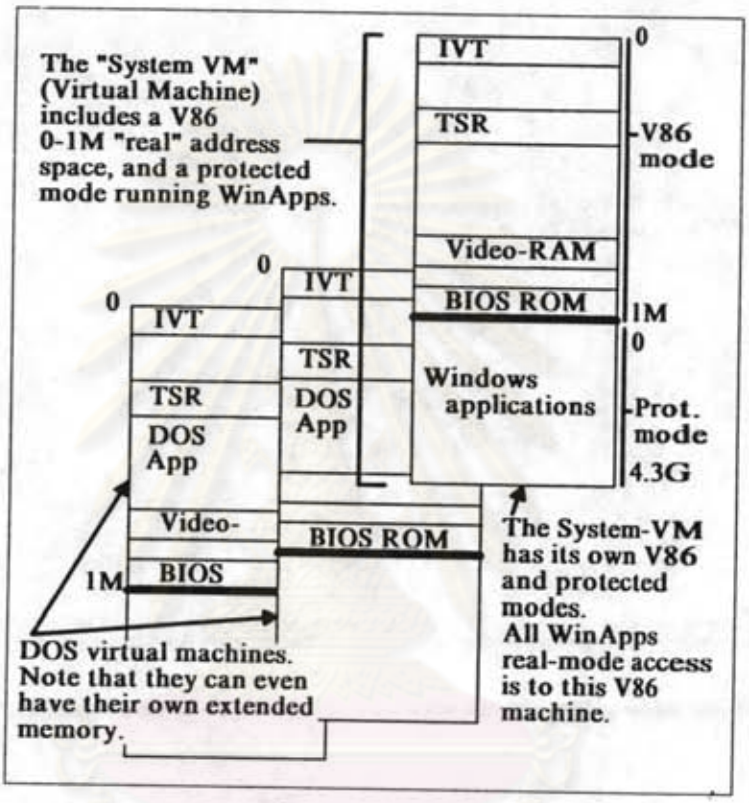


รูปที่ 3.10 การสะท้อนอินเตอร์รัพท์เพื่อเปลี่ยนภาวะการทำงาน

ที่มา Kualer, B.**Windows assembly language and system programming** (New York:Prentice-Hall, 1993), p.195

สำหรับในวินโดวส์ภาวะมาตรฐานผลของการทำเช่นนี้จะทำให้หน่วยประมวลผลกลางกลับไปทำงานในภาวะจริงและวินโดวส์หยุดการทำงานชั่วคราว แต่ในวินโดวส์ภาวะเสริมการ

เปลี่ยนภาวะจะเป็นการเปลี่ยนจากภาวะป้องกันกลับไปสู่ ภาวะV86 ทั้งนี้เนื่องจากในวินโดวส์ในภาวะเสริม มีขีดความสามารถในการสร้างในการสร้างเครื่องจักรเสมือนของแต่ละระบบ ซึ่งเครื่องจักรเสมือนแต่ละตัวเปรียบเสมือนเป็นเครื่องพีซีคอมพิวเตอร์ แต่ละเครื่องที่สามารถมีหน่วยความจำ, ตาราง IVT , ตาราง IDT และฮาร์ดแวร์ต่างๆ ของตนเอง ซึ่งแสดงได้ในรูปที่ 3.11



รูปที่ 3.11 แสดงโครงสร้างเครื่องจักรเสมือน และการจำลอง IVT พร้อมหน่วยความจำ

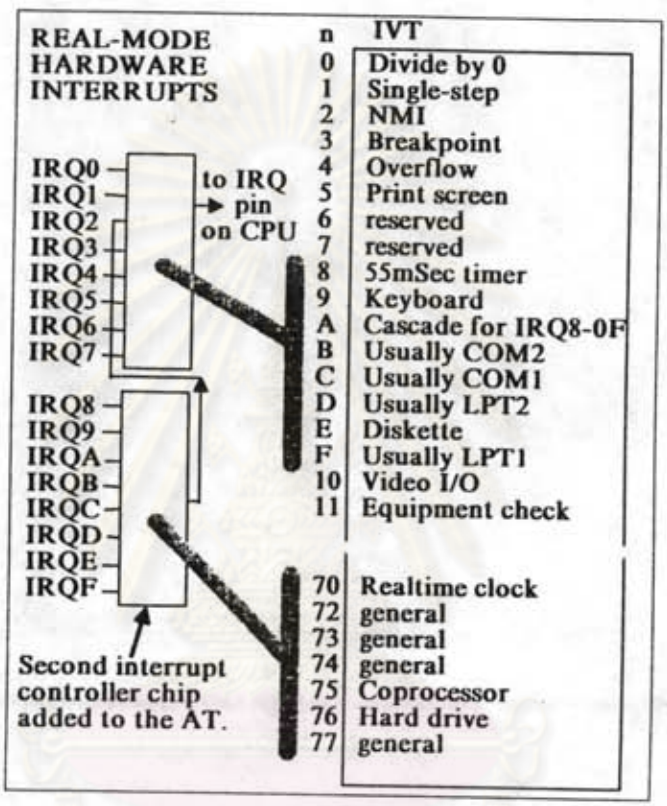
ที่มา Kualer, B.**Windows assembly language and system programming**

(New York:Prentice-Hall, 1993), p.201

จากรูปจะเห็นว่าเครื่องจักรเสมือนทุกตัวมีส่วนที่เป็น ภาวะV86 และภาวะป้องกันของตนเอง โดยที่เครื่องจักรเสมือนของระบบ จะเป็นเครื่องจักรเสมือนตัวแรกที่วินโดวส์สร้างขึ้นเพื่อใช้ประมวลผลระบบปฏิบัติการดอสในส่วน ภาวะV86 และโปรแกรมประยุกต์ของวินโดวส์ในส่วนของภาวะป้องกัน ส่วนโปรแกรมประยุกต์ของดอสจะประมวลผลในเครื่องจักรเสมือนของตนเองซึ่งจะถูกสร้างขึ้นใหม่

5) การจัดการฮาร์ดแวร์อินเตอร์รัพท์ในวินโดวส์

การกำหนด การใช้หมายเลข อินเตอร์รัพท์ของ อุปกรณ์แต่ละชนิดในคอมพิวเตอร์ พีซี แสดงไว้ในรูปที่ 3.12



รูปที่ 3.12 แสดงการใช้หมายเลขอินเตอร์รัพท์ของอุปกรณ์แต่ละชนิดในคอมพิวเตอร์ พีซี

ที่มา Kualer, B.**Windows assembly language and system programming**

(New York:Prentice-Hall, 1993), p.185

นอกจากนั้นหน่วยประมวลผลกลางตั้งแต่รุ่น 80286 เป็นต้นมา กำหนดให้หมายเลขอินเตอร์รัพท์ ตั้งแต่ 00-1Fh เป็นหมายเลขสงวนสำหรับหน่วยประมวลผลกลางใช้ ดังแสดงในตารางที่ 3.2 ซึ่งทำให้เกิดมีการใช้หมายเลขอินเตอร์รัพท์ซ้ำซ้อนกันขึ้น วินโดวส์จึงต้องทำการโปรแกรม PIC เสียใหม่ เพื่อให้ใช้หมายเลขอินเตอร์อื่นแทนหมายเลขเดิมโดยที่ภายใต้วินโดวส์ IRQ0-IRQF จะใช้หมายเลขอินเตอร์รัพท์ 50h-5Fh (เดิมใช้ 08-0Fh และ 70-77h) แต่เนื่องจากหมายเลขอินเตอร์รัพท์ของฮาร์ดแวร์ที่ใช้อยู่เดิมนั้นเป็นมาตรฐานที่โปรแกรมประยุกต์มากมายใช้อ้างอิงอยู่ การย้ายหมายเลขอินเตอร์รัพท์ในลักษณะนี้จะมีผลทำให้การออกแบบโปรแกรมประยุกต์ มีความยุ่งยาก

ขึ้น อีกทั้งจะมีปัญหาความไม่เข้ากัน (Incompatibilty) ของโปรแกรมประยุกต์อีก ดังนั้นวินโดวส์จึงยังคงต้องทำให้ผู้ใช้และผู้พัฒนาโปรแกรมประยุกต์ใช้อินเตอร์รัพท์หมายเลขเดิมอยู่แล้วโดยใช้วิธีการ สะท้อนฮาร์ดแวร์อินเตอร์รัพท์จากหมายเลข 50h - 5Fh กลับไปที่อินเตอร์รัพท์หมายเลข 08-0 Fh และ 70-77h ดังเดิม แต่กลไกที่ใช้ในการทำจะแตกต่างกันไปในแต่ละภาวะการทำงาน ผลที่ได้ทำให้ในภาวะมาตรฐาน การอ้างอิงหมายเลขฮาร์ดแวร์อินเตอร์รัพท์สามารถใช้ได้ทั้ง 2 ชุด (ทั้งเก่าและใหม่) แต่ในภาวะเสริมจะต้องใช้หมายเลขอินเตอร์รัพท์เดิมเท่านั้น (08-0Fh ,70-77h)

Processor-Defined Exceptions

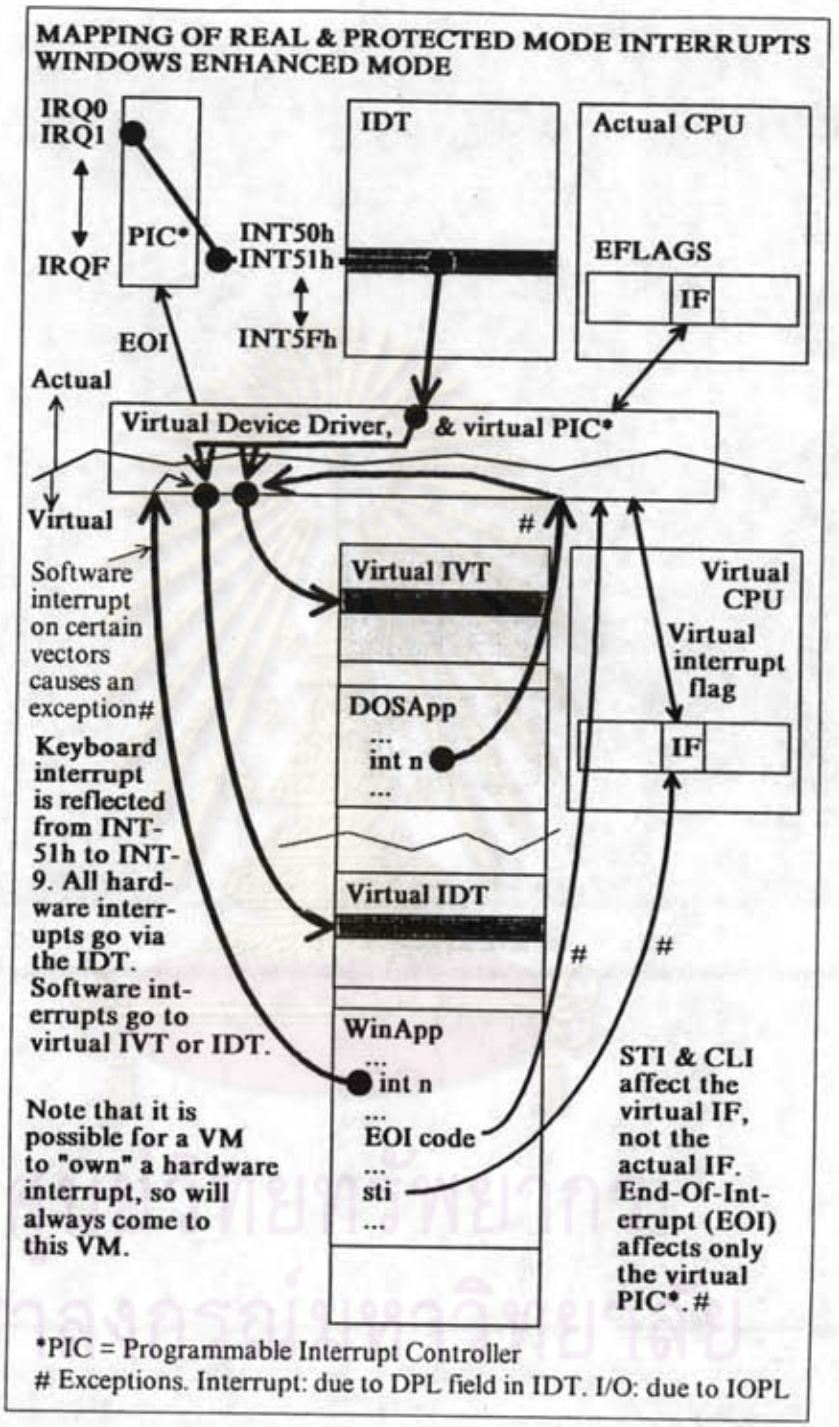
Interrupt Number	Description	Type	CS:IP Points to Instruction That Caused Exception	Pushes Error Code on Stack	Processor 8088/ 8086	80286	80386	80486
0	Divide error	F [1]	Yes [1]	No	•	•	•	•
1	Single-step	T	No	No	•	•	•	•
	Debug exceptions	T,F	[2]	No			•	•
2	Nonmaskable interrupt (NMI)	T	No [3]	No	•	•	•	•
3	Breakpoint interrupt	S,T	No [4]	No	•	•	•	•
4	INTO Overflow	S,T	No [5]	No	•	•	•	•
5	BOUND range exceeded	S,F	Yes	No		•	•	•
6	Invalid opcode	F	Yes	No		•	•	•
7	Coprocessor not available	F	Yes	No		•	•	•
8	Double-fault	A	Yes	Yes [6]		•	•	•
9	Coprocessor segment overrun	A	No	No		•	•	[7]
Ah	Invalid task segment	F	Yes	Yes			•	•
Bh	Segment not present	F	Yes	Yes			•	•
Ch	Stack fault	F	Yes	Yes			•	•
Dh	General protection	F,A	Yes	Yes		•	•	•
Eh	Page fault	F	Yes	Yes			•	•
Fh	Reserved by Intel	–	–	–	–	–	–	–
10h	Coprocessor error	F	Yes [8]	No		•	•	•
11h	Alignment check	F	Yes	Yes				•
12h-1Fh	Reserved by Intel	–	–	–	–	–	–	–

F Fault
T Trap
A Abort
S Software generated

[1] On the 8086 and 8088 only, interrupt 0 is a trap, not a fault. The CS:IP saved on the stack points to the instruction that follows the trapped instruction.
[2] Some debug exceptions are traps and others are faults. Refer to Chapter 9 for more details.
[3] The NMI is generated by an external signal.
[4] The saved CS:IP points to the byte after the breakpoint instruction.
[5] The INTO instruction may be executed at any time, regardless of when the overflow occurred.
[6] The error code pushed is always 0.
[7] This exception is not used by the 80486. Interrupt Dh occurs instead.
[8] The saved CS:IP value points to the instruction that caused the exception to be detected, not the instruction that caused the error.

ตารางที่ 3.2 แสดงหมายเลขอินเตอร์รัพท์ที่สงวนไว้สำหรับหน่วยประมวลผลกลาง

ที่มา Hummel, R.L. **PC magazine technical reference. the processor and coprocessor** (California:Osborn McGraw-Hill,1988), p. 160



รูปที่ 3.13 แสดงไดอะแกรมการทำงานเมื่อเกิดฮาร์ดแวร์อินเตอร์รัพท์ในภาวะเสริม

ที่มา Kualer, B.**Windows assembly language and system programming**

(New York:Prentice-Hall, 1993), p.174

เมื่อเกิดอินเตอร์รัพท์ขึ้น หน่วยประมวลผลกลางจะใช้ตาราง IDT ในการหาตำแหน่งโปรแกรมบริการ ในภาวะมาตรฐานเนื่องจากไม่มีการจำลอง IDT จึงเป็นการใช้ IDT ของระบบซึ่งทำให้เราสามารถใช้หมายเลขเวกเตอร์ได้ทั้งเก่าและใหม่ แต่สำหรับภาวะเสริมซึ่งมีการใช้ภาคขับอุปกรณ์เสมือน ในการทำงานจึงมีการจำลอง IDT จากรูปส่วนที่เรียกว่า Virtual Device Driver จะทำหน้าที่สะท้อนการอินเตอร์รัพท์กลับมายัง IDT หรือ IVT ของ เครื่องจักรเสมือน ที่กำลังทำงานอยู่ ทั้งนี้ขึ้นอยู่กับว่าในขณะนี้หน่วยประมวลผลกลางทำงานในภาวะป้องกัน หรือ ภาวะV86

โปรแกรมประยุกต์ของดอสหรือโปรแกรมประยุกต์ของวินโดวส์ จะมองเห็นเฉพาะ IVT หรือ IDT ที่จำลองขึ้นใน เครื่องจักรเสมือน ของตนเท่านั้น และนี่เป็นเหตุผลว่าทำไมในภาวะเสริมจึงต้องใช้หมายเลขอินเตอร์รัพท์เดิมเท่านั้น

ในรูปที่ 3.13 จะเห็นว่าแม้แต่ IF แฟลก และ PIC ในภาวะเสริมก็ถูกจำลองขึ้นด้วยเช่นกัน นั่นหมายความว่า คำสั่ง CLI หรือ STI ไม่ได้กระทำโดยตรงกับแฟลกรีจีสเตอร์ แต่การใช้คำสั่งดังกล่าวทำให้เกิดเอคเซปชันขึ้นและส่วนของโปรแกรมบริการนั้นทำหน้าที่จำลอง IF แฟลกขึ้นมา ทั้งนี้เพื่อให้เครื่องจักรเสมือนแต่ละตัวมีฮาร์ดแวร์ส่วนนี้เป็นอิสระไม่ขึ้นแก่กัน

สำหรับ PIC ก็เช่นกัน ได้ถูกจำลองขึ้นโดยใช้ VPIC กล่าวคือ การติดต่อระหว่างโปรแกรมประยุกต์กับ PIC จะเป็นการติดต่อระหว่างโปรแกรมประยุกต์กับ VPIC และ VPIC จะทำหน้าที่ควบคุม PIC ที่แท้จริงอีกทอดหนึ่งดังแสดงในรูปที่ 3.13

อินเตอร์รัพท์เสมือนกับวินโดวส์ในภาวะเสริม

ในระบบพีซี การจัดการฮาร์ดแวร์อินเตอร์รัพท์ ทำโดยใช้อุปกรณ์ควบคุมอินเตอร์รัพท์ (Programable Interrupt Controler : PIC) ซึ่งทำหน้าที่ส่งสัญญาณอินเตอร์รัพท์ และให้ค่าหมายเลขอินเตอร์รัพท์ที่ต้องการแก่หน่วยประมวลผลกลาง แต่สำหรับวินโดวส์ในภาวะเสริมได้ดำเนินการโดยจัดสร้างภาคขับอุปกรณ์เพื่อจำลองตัวควบคุมอินเตอร์รัพท์เสมือน (Virtual PIC Driver : VPICD) สำหรับใช้จัดการอินเตอร์รัพท์ให้แก่โปรแกรมประยุกต์แทนซึ่งโดยเหตุนี้โปรแกรมประยุกต์จึงไม่ได้ติดต่อกับ PIC โดยตรงแต่ใช้ VPICD แทนในการจัดการอินเตอร์รัพท์

VPICD มีหน้าที่จัดการดูแลสัญญาณร้องขออินเตอร์รัพท์ (Interrupt Request : IRQ) ทุกเส้นของระบบโดยการควบคุมที่ PIC โดยตรง โดยปกติสัญญาณเหล่านี้ VPICD แยกออกเป็น 2

ส่วนขึ้นอยู่กับสถานะ เมื่อ VPICD เริ่มกำหนดค่าเริ่มต้น โดยที่ถ้า IRQ ใดเปิด (Enable) อยู่จะจัดอยู่ในกลุ่ม IRQร่วม (Global IRQ) ถ้า IRQ ใดปิด (Disable) อยู่ ถือว่าเป็น IRQท้องถิ่น (Local IRQ)

เมื่อเกิดอินเตอร์รัพท์จากสัญญาณในกลุ่ม IRQร่วม VPICD จะถ่ายทอดการอินเตอร์รัพท์นั้นไปยังเครื่องจักรเสมือนตัวปัจจุบันทันที, ทั้งนี้โดยถือว่าโปรแกรมบริการนี้จะต้องมีอยู่ในเครื่องจักรเสมือนทุกตัว กล่าวคือ โปรแกรมบริการอินเตอร์รัพท์จะอยู่ในหน่วยความจำที่เครื่องจักรเสมือนทุกตัวใช้ร่วมกัน (Global ISR)

สำหรับ IRQท้องถิ่น ตามปกติจะปิดอยู่ ถ้าโปรแกรมประยุกต์ในเครื่องจักรเสมือนใดทำการเปิดสัญญาณในกลุ่มนี้ VPICD จะถือว่า เครื่องจักรเสมือนนั้นเป็นเจ้าของ IRQ นี้ และจะทำการถ่ายทอดการอินเตอร์รัพท์ไปให้กับ เครื่องจักรเสมือน นั้นเป็นเจ้าของ IRQ นี้ และจะทำการถ่ายทอดการอินเตอร์รัพท์ไปให้กับ เครื่องจักรเสมือน นั้นๆ ในกรณีที่โปรแกรมประยุกต์ในเครื่องจักรเสมือนอื่น พยายามจะขอใช้ IRQ เดียวกันนี้ VPICD จะยกเลิกการทำงานของ เครื่องจักรเสมือนตัวที่ 2 แล้วแสดงข้อความแจ้งผู้ใช้

การติดตั้งโปรแกรมบริการอินเตอร์รัพท์ ภายใต้วินโดวส์

ไม่มี API ของวินโดวส์สำหรับใช้ติดตั้งโปรแกรมบริการอินเตอร์รัพท์ ดังนั้นการติดตั้งจะต้องใช้บริการ INT 21h หมายเลขย่อยที่ 25h / 35h ซึ่งมีรูปแบบการใช้งานเป็นดังนี้

1) การอ่านค่าในตาราง IDT : (Get Interrupt Vector)

ค่ารีจีสเตอร์ที่ใช้	AL	=	หมายเลขเวคเตอร์
	AH	=	35h
ค่าผลลัพธ์	CS : BX	=	ซีเลคเตอร์ : ออฟเซต ของโปรแกรมบริการ

2) การเปลี่ยนแปลงค่าในตาราง IDT : (Set Interrupt Vector)

ค่ารีจีสเตอร์ที่ใช้	AL	=	หมายเลขเวคเตอร์
	AH	=	25h
	DS : DX	=	ซีเลคเตอร์ : ออฟเซต ของโปรแกรมบริการ
ค่าผลลัพธ์	ไม่มี		

ข้อสังเกตและข้อจำกัดของการติดตั้งโปรแกรมบริการอินเตอร์รัพท์โดยใช้ INT 21h

1. โปรแกรมบริการที่ติดตั้งโดยวิธีนี้จะทำงานเฉพาะในภาวะป้องกันของเครื่องจักรเสมือนที่เป็นเจ้าของ IRQ นั้นเท่านั้น กล่าวคือหน่วยประมวลผลกลางต้องไม่อยู่ในภาวะจริง หรือ ภาวะV86 หรือขณะเกิดอินเตอร์รัพท์ เครื่องจักรเสมือนตัวอื่นกำลังทำงานอยู่
2. โปรแกรมบริการจะทำงานในระดับเอกสิทธิ์ 3 ซึ่งเท่ากับโปรแกรมประยุกต์ธรรมดา นั่นหมายถึงไม่มีสิทธิ์ติดต่อกับอุปกรณ์ภายนอกโดยตรง จะต้องผ่านการจำลองของ VxD เสมอ
3. โปรแกรมบริการสำหรับการอินเตอร์รัพท์ในลักษณะนี้จะมีการหน่วงเวลาค่อนข้างมาก เนื่องจากการจำลองสภาวะต่างๆ ในภาวะเสริม โปรแกรมประยุกต์ที่ต้องการการตอบสนองแบบทันทีจึงไม่ควรใช้โปรแกรมบริการในลักษณะนี้
4. ห้ามมิให้ใช้ API ของวินโดวส์ในโปรแกรมบริการเนื่องจาก API ส่วนใหญ่ของวินโดวส์เป็นลักษณะเรียกใช้ซ้ำไม่ได้ (Non-Reentrance) API ที่วินโดวส์สร้างขึ้นเป็นพิเศษ เพื่อให้ใช้ได้มีเพียง PostMessage, PostAppMessage, OutDebugStr, DriverCallback, timeGetSystemTime, timeGetTime, timeSetEvent, timeKillEvent, midiOutShortMsg และ midiOutLongMsg.