บทที่ 9
การสนับสนุนระบบปฏิบัติการ

ในบทนี้เราจะเรียนถึงบทบาทของซีพียูต่อการทำงานของระบบปฏิบัติการ(OS : Operating System) ซึ่งเนื้อหาในบทนี้อาจซ้ำกับบางส่วนในวิชา OS อยู่บ้างเนื่องจากในหนังสือ OS แทบทุกเล่ม จะพูดถึงความต้องการสนับสนุนจากฮาร์ดแวร์ไว้อย่างคร่าวๆ อยู่แล้ว แต่อย่างไรก็ตาม ในวิชานี้จะเน้นเฉพาะเรื่องของความจำเป็นของการสนับสนุนการทำงานของโพรเซสเซอร์และระบบฮาร์ดแวร์ของโอเอสเป็นหลัก และจะมีการยกตัวอย่างโพรเซสเซอร์ใช้งานจริง เพื่อให้นักศึกษามองเห็นภาพ และเข้าใจได้มากยิ่งขึ้น

ระบบปฏิบัติการแทบทุกตัวในปัจจุบันจะสนับสนุนการทำมัลติทาสกิ้ง (Multitasking) ทั้งนี้เพราะจะทำให้เกิดการใช้ประโยชน์จากซีพียู(CPU Utilization) ได้มากที่สุด และยังช่วยอำนวยความสะดวกในการทำงานได้มาก ยกตัวอย่างเช่น ในขณะที่เรากำลังดาวน์โหลดข้อมูลจากอินเตอร์เนตอยู่ เราก็สามารถพิมพ์งานไปพร้อมๆ กันโดยไม่ต้องรอให้การดาวน์โหลดเสร็จสิ้นเสียก่อน หรือในขณะที่กำลังทำงาน เราอาจเปิดเพลงจากไฟล์ mp3 ไปพร้อมๆ กับการตกแต่งรูปภาพก็สามารถทำได้ เป็นต้น และเพื่อให้สามารถทำมัลติทาสกิ้งได้ สถาปัตยกรรมของคอมพิวเตอร์จะต้องประกอบไปด้วยองค์ประกอบสนับสนุน 4 อย่าง ดังต่อไปนี้

1. Dual-Mode Operation
2. program relocation
3. memory protection
4. timer interrupts

9.1.1 Dual-mode Operation

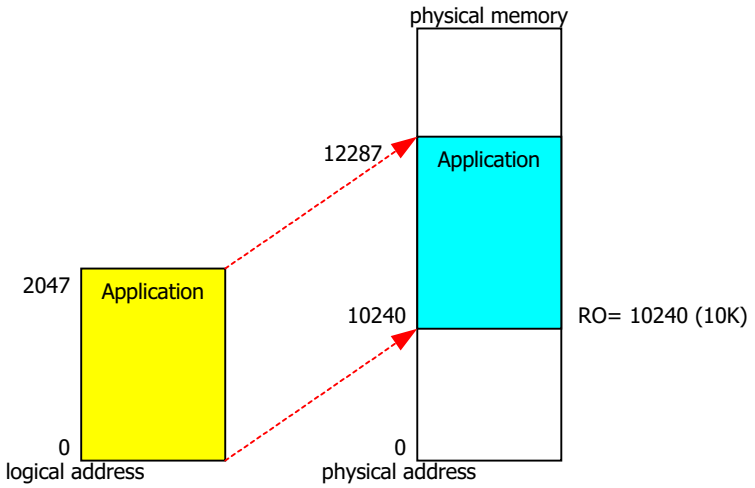
เพื่อที่จะให้โอเอสทำงานได้อย่างถูกต้อง จะต้องมีกลไกการป้องกันไม่ให้ตัวโอเอสถูกรบกวนโดยโปรแกรมอื่นได้ โดยซีพียูจะต้องมีโหมดการทำงานให้ใช้อย่างน้อยสองโหมดด้วยกันคือโหมดแรกเรียกว่า “monitor mode”(หรือบางครั้งอาจจะเรียกว่า system mode หรือ privileged mode ก็ได้) และโหมดที่สองเรียกว่า “user mode” ภายในซีพียูจะมีบิตสถานะไว้คอยบอกว่าในขณะนี้ซีพียูกำลังทำงานที่โหมดใด ซีพียูที่มีการทำงานแบบนี้จะมีคำสั่งอยู่กลุ่มหนึ่งซึ่งเรียกว่า privileged instructions ซึ่งเป็นกลุ่มของคำสั่งที่มีผลต่อเสถียรภาพของระบบ โดยคำสั่งพวกนี้สามารถที่จะรันได้ก็ต่อเมื่อซีพียูอยู่ใน monitor mode เท่านั้น หากมีการพยายามที่จะรันคำสั่งพวกนี้ในขณะที่ซีพียูกำลังอยู่ใน user mode แล้ว จะเกิด trap เพื่อบอกโอเอสว่าเกิดข้อผิดพลาดและมีการพยายามทำคำสั่งที่ไม่ได้รับอนุญาตเกิดขึ้น

เมื่อเริ่มเปิดเครื่องคอมพิวเตอร์ ซีพียูจะทำการโหลดระบบปฏิบัติการขึ้นมาในหน่วยความจำ และเซ็ตให้ระบบปฏิบัติการทำงานใน monitor mode ในโหมดนี้จะอนุญาตให้โพรเซสเซอร์ทำการเอกซีคิวต์คำสั่งได้ทุกคำสั่ง รวมทั้งสามารถรันคำสั่งประเภท privileged instructions ทำให้โอเอสมีสิทธิในการใช้ทรัพยากรทั้งหมดของระบบได้อย่างเต็มที่

เมื่อผู้ใช้รันโปรแกรม ก่อนที่โอเอสจะสวิทช์ซีพียูไปรันโปรแกรมมันจะเซ็ทให้ซีพียูอยู่ใน user mode ซึ่งจะจำกัดขอบเขตและสิทธิบางประการในการใช้ทรัพยากรของแอพพลิเคชั่นเอาไว้ แล้วปล่อยให้ซีพียูรันแอพพลิเคชั่นโปรแกรมใน user mode ซึ่งมีสิทธิในการใช้ทรัพยากรได้ตามที่โอเอสได้จัดสรรให้เท่านั้น นอกจากนี้หากแอพพลิเคชั่นพยายามที่จะใช้คำสั่งเพื่อสวิทช์ซีพียูให้อยู่ใน monitor mode (ซึ่งเป็นคำสั่งประเภท privileged instruction) ก็จะเกิด trap กลับมาที่โอเอสเพื่อบอกว่าเกิดข้อผิดพลาดและมีการพยายามทำคำสั่งที่ไม่ได้รับอนุญาต

9.1.2 Program relocation

อย่างที่นักศึกษาได้เรียนมาบ้างแล้วในวิชา Operating System ว่าโอเอสที่รองรับการทำงานแบบมัลติทาสกิ้งจะต้องมีความสามารถในการเคลื่อนย้ายแอพพลิเคชั่นโปรแกรมไปวางไว้ในส่วนใดของหน่วยความจำก็ได้ สมมุติว่าโปรแกรมตัวหนึ่งถูกเขียนให้ใช้ logical address ตั้งแต่ย่าน 0-2 Kbytes และโอเอสโหลดตัวโปรแกรมไปวางไว้ในหน่วยความจำตำแหน่งแอดเดรสที่ 10K-12K แล้ว เราจะถือว่าค่า relocation offset เท่ากับ 10K ระหว่างที่โปรแกรมกำลังถูกเอกซีคิวต์นั้นฮาร์ดแวร์ในตัวซีพียูจะต้องทำการแปลงค่าแอดเดรสที่โปรแกรมต้องการให้เป็นค่า physical address ด้วยการบวกค่า 10K เข้าไป ยกตัวอย่างเช่น หากตัวโปรแกรมเขียนให้อ่านหน่วยความจำตำแหน่งที่ 1024 แล้ว วงจรแปลงค่าแอดเดรสจะต้องนำค่านี้มาบวกกับค่า 10240 (10K) ซึ่งจะทำให้มีการอ่านค่า 11264 (10240+1024) ไปใช้งานแทน

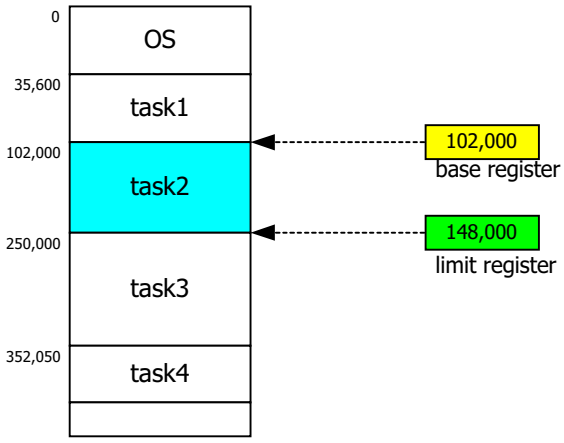


รูปที่ 9.1 ตัวอย่างการทำ program relocation และ relocation offset (RO)

9.1.3 Memory Protection

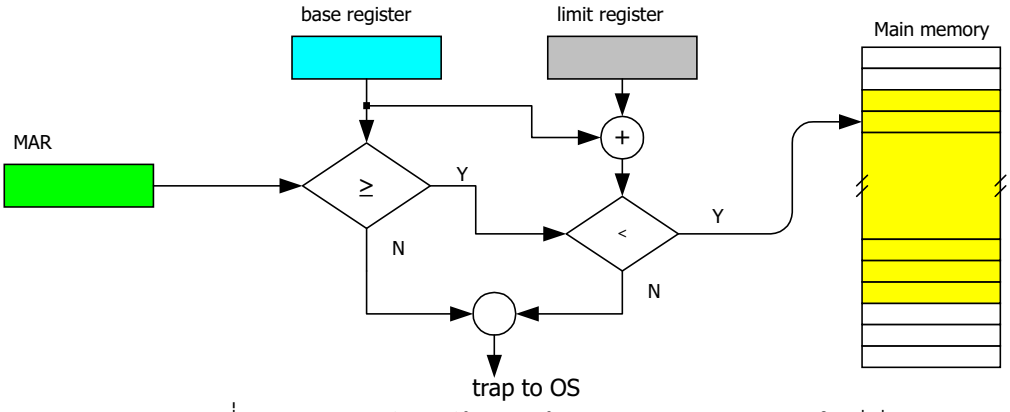
ในการทำมัลติทาสกิ้ง นั้น จะมีหลายๆ แอพพลิเคชั่น อยู่พร้อมๆ กันในหน่วยความจำ ตัวโอเอสจะเลือกว่าในช่วงเวลาใดเวลาหนึ่งนั้นให้ซีพียูเรียกแอพพลิเคชั่นตัวใดขึ้นมาทำงาน และเพื่อให้แอพพลิเคชั่นแต่ละตัวไม่ไปกวนการทำงานของแอพพลิเคชั่นตัวอื่น เช่น ตัวโปรแกรม Microsoft word จะต้องไม่สามารถไปแก้ไขหน่วยความจำตำแหน่งที่ถูกจัดสรรให้กับโปรแกรม Photoshop ซึ่งกำลังใช้ในการตกแต่งรูปอยู่ได้ และนอกจากนี้จะต้องมีกลไกป้องกันไม่ให้แอพพลิเคชั่นเข้าไปแก้ไขข้อมูลของโอเอส (เช่น Interrupt Service Routine) อีกด้วย

ดังนั้นซีพียูที่ออกแบบมาให้สามารถทำงานแบบมัลติทาสกิ้งได้จะต้องสนับสนุนกลไกของการปกป้องหน่วยความจำที่ถูกจัดสรรให้กับโปรแกรมหนึ่งจากการเข้าถึงของโปรแกรมตัวอื่นซึ่งทำได้โดยซีพียูจะมีรีจิสเตอร์พิเศษสำหรับงานนี้ 2 ตัว ได้แก่ base register และ limit register โดยจะใช้ base register ทำการเก็บค่าตำแหน่งเริ่มต้นของหน่วยความจำที่ถูกจัดสรรให้กับโปรแกรม ส่วน limit register จะเป็นตัวบอกว่าจำนวนหน่วยความจำที่ได้รับการจัดสรรให้แอพพลิเคชั่นนั้นได้ใช้มีจำนวนเท่าใด ยกตัวอย่างในรูป 9.2 จะเห็นว่า task2 ได้รับการจัดสรรหน่วยความจำให้จำนวน 148,000 ไบต์ โดยเก็บโปรแกรมไว้ในหน่วยความจำตำแหน่งตั้งแต่ 102,000-250,000



รูปที่ 9.2 การใช้ base register และ limit register

กลไกการทำงานของการทำ memory protection นั้นจะมีวงจรฮาร์ดแวร์พิเศษภายในซีพียูไว้ทำการตรวจเช็คว่ามีการใช้หน่วยความจำเกินกว่าย่านที่อนุญาตหรือไม่ โดยเปรียบเทียบว่าแอดเดรสที่ซีพียูต้องการอ่านโดยโปรแกรมมีค่ามากกว่าหรือเท่ากับค่าใน base register หรือไม่หากค่าแอดเดรสนั้นน้อยกว่าค่าใน base register ก็จะถือว่าเกิดข้อผิดพลาดเกิดขึ้น แต่ถ้ามีค่ามากกว่า ก็จะต้องตรวจอีกครั้งหนึ่งว่าแอดเดรสนั้นมีค่าน้อยกว่าผลรวมของค่าใน base register และค่าใน limit register หรือไม่หากมีค่าน้อยกว่าก็แสดงว่าการเข้าถึงหน่วยความจำครั้งนั้นอยู่ในย่านที่ถูกต้อง แต่ถ้าไม่ใช่ก็จะเกิด exception และจะมีการเรียกโอเอสขึ้นมาทำงานเพื่อจัดการกับปัญหาที่เกิดขึ้นดังตัวอย่างในรูปที่ 9.3

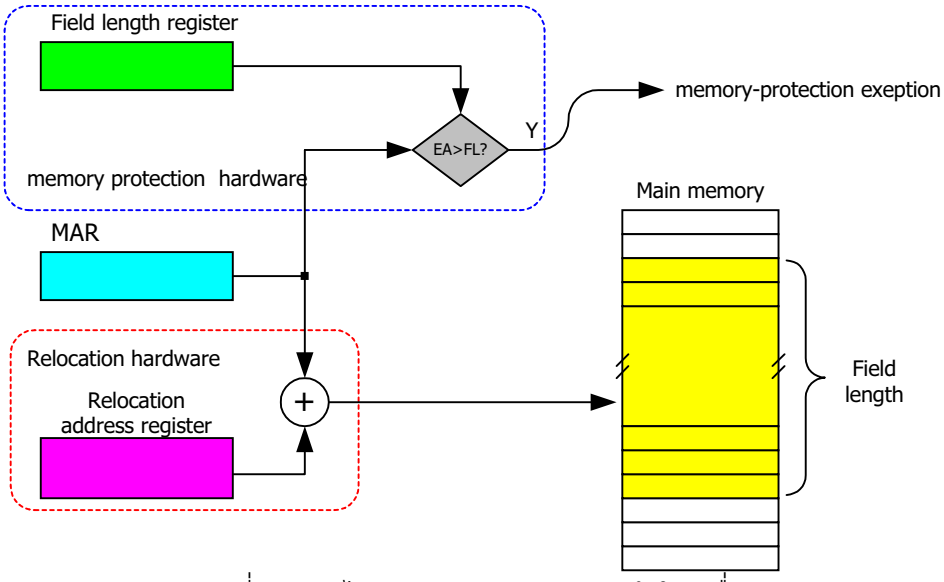


รูปที่ 9.3 วงจรฮาร์ดแวร์สำหรับทำ memory protection ในซีพียู

Case study : การทำ program relocation และ memory protection ในเครื่อง CDC 6600

เครื่องคอมพิวเตอร์ของ CDC 6600 จะมีรีจิสเตอร์พิเศษที่เรียกว่า relocation-address register(RA) ซึ่งใช้สำหรับการทำ program relocation โดยที่ตัวโอเอสเท่านั้นที่จะสามารถเข้าถึงรีจิสเตอร์ตัวนี้ได้(ด้วยการใช้คำสั่ง privileged instruction) ตัวโปรแกรมที่กำลังเอกซีคิวต์อยู่จะไม่รู้เลยว่ามีรีจิสเตอร์นี้อยู่ในระบบ ก่อนที่โอเอสจะโหลดโปรแกรมใส่ในหน่วยความจำ มันจะทำการหาบล๊อกของหน่วยความจำที่ไม่ถูกใช้งานเสียก่อน ซึ่งเรียกบล๊อกของหน่วยความจำนี้ว่า field โดยขนาดของหน่วยความจำที่ถูกจัดสรรให้โปรแกรมจะเรียกว่า field length (FL) ตัวโอเอสจะทำการนำค่าแอดเดรสของ field ไปใส่ไว้ในรีจิสเตอร์ RA จากนั้นจึงทำการโหลดตัวโปรแกรมใส่ในหน่วยความจำ field นั้น เมื่อโพรเซสเซอร์ต้องการเข้าถึงหน่วยความจำ ค่าแอดเดรสที่โพรเซสเซอร์อ้างถึงนี้จะถูกนำมาบวกเข้ากับค่าในรีจิสเตอร์ RA โดยอัตโนมัติด้วยแล้วจึงได้เป็นค่าแอดเดรสที่ส่งไปยังหน่วยความจำดังรูปที่ 9.4

นอกจากจะมีรีจิสเตอร์ RA ไว้สำหรับการทำ program relocation แล้ว เครื่อง CDC6600 ยังมีรีจิสเตอร์ Field-length register(FL) เพิ่มเติมไว้สำหรับการทำ memory protection อีกด้วย โดยรีจิสเตอร์นี้จะเก็บขนาดของหน่วยความจำที่ถูกจัดสรรให้โปรแกรมเอาไว้ ทุกครั้งที่ซีพียูเข้าถึงหน่วยความจำ วงจร memory protection จะนำค่าแอดเดรสนี้มาตรวจสอบก่อนว่ามีการใช้หน่วยความจำนอกเหนือไปจากย่านที่ได้รับการจัดสรรให้หรือไม่ หากยังอยู่ในย่านที่ระบุก็จะอนุญาตให้เข้าถึงหน่วยความจำตำแหน่งนั้นๆ ได้ แต่ถ้าไม่เป็นอย่างนั้นแล้ว วงจร memory protection จะทำการ trap บอกกับตัวโอเอสว่ามี memory protection exception เกิดขึ้น

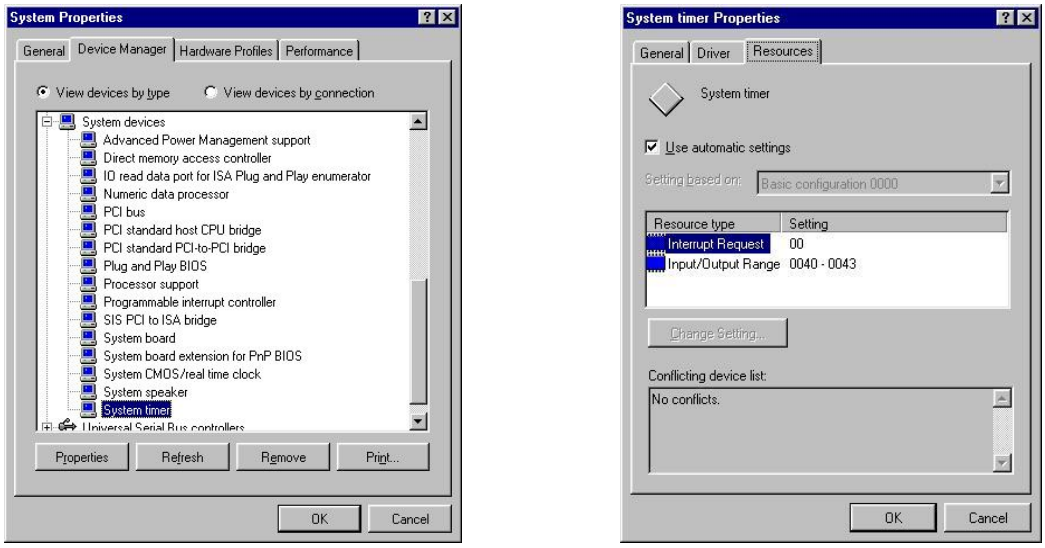


รูปที่ 9.4 กลไกการจัดการหน่วยความจำในเครื่อง CDC6600

9.1.4 Timer interrupts

ในการทำมัลติทาสกิ้งนั้น แม้ว่าในมุมมองของผู้ใช้คอมพิวเตอร์จะมองว่าหลายๆ โปรแกรมสามารถที่จะรันพร้อมๆ กันได้ แต่ความจริงแล้วเกิดจากการแบ่งเวลาของซีพียูให้กับแต่ละแอพพลิเคชั่นด้วยการสวิทช์ไปรันแอพพลิเคชั่นหนึ่งเป็นระยะเวลาที่กำหนดเอาไว้จากซีพียู(สมมุติว่าเป็นเวลา 100 mS) จากนั้นจึงสวิทช์ซีพียูไปรันแอพพลิเคชั่นถัดไปจนถึงแอพพลิเคชั่นสุดท้ายด้วยค่าเวลาเท่าๆ กัน แล้วจึงสวิทช์มารันแอพพลิเคชั่นแรกใหม่อีกครั้ง และเนื่องจากซีพียูมีความเร็วในการทำงานสูงมากจนทำให้ผู้ใช้เครื่องคอมพิวเตอร์มองว่าทุกๆ แอพพลิเคชั่นทำงานพร้อมๆ กัน

ก่อนที่โอเอสจะสวิทช์ซีพียูไปรันแอพพลิเคชั่นหนึ่งๆ มันจะไปเซ็ทค่าให้กับวงจรจับเวลา (Timer) ของระบบ หลังจากนั้นจึงสั่งให้ซีพียูสวิทช์ไปรันแอพพลิเคชั่นนั้น ในระหว่างที่แอพพลิเคชั่นถูกรันอยู่นั้น วงจรจับเวลาจะลดค่าใน counter ของมันเองไปเรื่อยๆ ทุกขอบขาขึ้นของสัญญาณนาฬิกา เมื่อวงจรจับเวลาพบว่าค่าใน counter ลดค่าถึงศูนย์มันจะทำการอินเตอร์รัพต์ซีพียู ส่งผลให้ซีพียูทำการหยุดการทำงานแอพพลิเคชั่นนั้นและโหลดโปรแกรมบริการอินเตอร์รัพต์ (Interrupt Service Routine : ISR) ซึ่งเป็นโปรแกรมส่วนหนึ่งของโอเอสขึ้นมาทำงาน จากนั้นโอเอสจะเซ็ทค่าในวงจรจับเวลาใหม่เพื่อเตรียมที่จะรันแอพพลิเคชั่นตัวอื่นต่อไป ดังนั้นจะเห็นว่าแต่ละแอพพลิเคชั่นจะสามารถรันในซีพียูได้แต่ละครั้งเป็นเวลาที่กำหนดใน Timer เท่านั้น และเนื่องจากคำสั่งที่ใช้ในการเซ็ทค่าใน Timer เป็นคำสั่งในกลุ่ม Privileged instruction ดังนั้นแอพพลิเคชั่นซึ่งรันใน user mode จึงไม่สามารถที่จะเซ็ทค่าใน Timer เองเพื่อที่จะยืดระยะเวลาการครอบครองซีพียูของตัวเองออกไปได้



รูปที่ 9.5 Timer ของเครื่อง PC

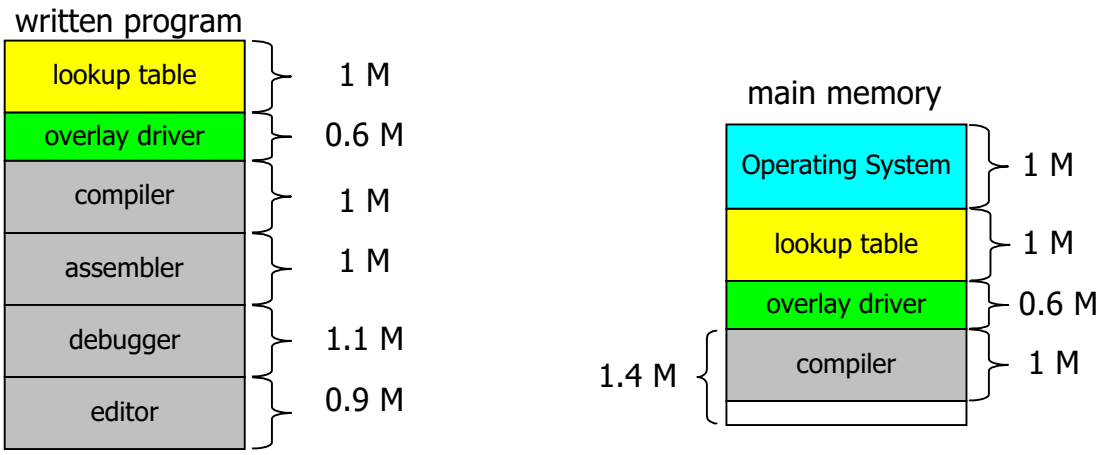
ในทางปฏิบัติแล้ว วงจร Timer นี้อาจสร้างขึ้นไว้ในซีพียูหรืออาจจะเป็นวงจรต่างหากบนเมนบอร์ดของระบบก็ได้ ทั้งนี้ขึ้นอยู่กับสถาปัตยกรรมของคอมพิวเตอร์แต่ละแบบ ซีพียูบางตัวอาจมีวงจร timer อยู่ในตัวเอง เช่น MCS-51 แต่บางซีพียูอาจไม่มีในตัวเองจึงต้องต่อวงจรภายนอก ยกตัวอย่างเช่น เครื่อง PC ที่ใช้ซีพียูตระกูล 80x86 จะมีไอซี 8253 (หรือเบอร์อื่นที่เทียบเท่า) อยู่บนเมนบอร์ดทำหน้าที่เป็นวงจร timer ของระบบ โดยไอซี 8253 จะมีวงจรจับเวลาภายในอยู่ 3 ตัวซึ่งต่ออยู่กับ I/O port หมายเลข 40-43 ดังรูปที่ 9.5

การใช้ Relocation register ในทำ program relocation นั้นเป็นเทคนิคที่ใช้ในเครื่องคอมพิวเตอร์ยุคแรกๆ เครื่องคอมพิวเตอร์ในปัจจุบันจะมีเทคนิคของหน่วยความจำเสมือน (virtual memory) เข้ามาจัดการหน่วยความจำ ซึ่งเทคนิคของหน่วยความจำเสมือนนี้จะสนับสนุนการทำ program relocation และ memory protection ในตัวของมันเองอีกด้วย

9.2 Virtual memory

ในการเขียนโปรแกรมในอดีต สิ่งที่โปรแกรมเมอร์จะต้องคำนึงถึงก็คือขนาดของหน่วยความจำที่โปรแกรมจะต้องไม่ใหญ่ไปกว่าขนาดของหน่วยความจำที่ติดตั้งลงไปในระบบ ยกตัวอย่างเช่น หากเมนบอร์ดของเครื่องคอมพิวเตอร์ที่ใช้อยู่ทำการติดตั้งแผงหน่วยความจำไว้ 4 เมกกะไบต์ ก็จะทำให้โปรแกรมที่เขียนขึ้นไม่สามารถมีขนาดใหญ่กว่า 4 เมกกะไบต์ได้ แต่เนื่องจากโปรแกรมบางตัวมีความซับซ้อนและมีฟังก์ชั่นการทำงานมากมาย ส่งผลให้ขนาดของฟังก์ชั่นต่างๆ ทั้งหมดอาจมีขนาดใหญ่กว่าขนาดของหน่วยความจำที่มีอยู่จริง เพื่อแก้ไขปัญหานี้ โปรแกรมเมอร์จะต้องแบ่งโปรแกรมออกเป็นส่วนๆ และเขียนโค๊ดให้อ่านทีละส่วนของโปรแกรมเข้ามาในหน่วยความจำเมื่อต้องการใช้ฟังก์ชั่นนั้นๆ เท่านั้น เทคนิคนี้เรียกว่าการเขียนโปรแกรมแบบ Overlay ซึ่ง

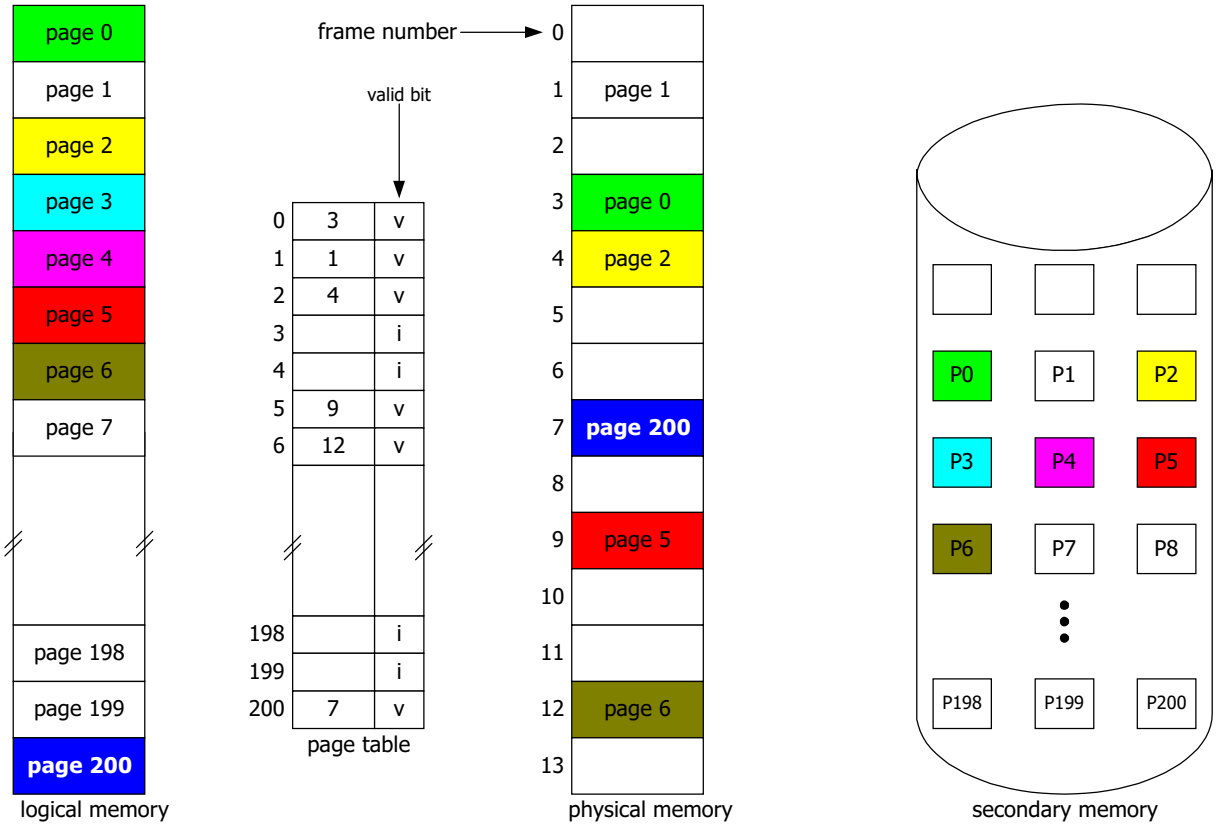
มีข้อยุ่งยากในการใช้งานมากพอสมควรด้วยกัน เพราะนอกจากโปรแกรมเมอร์จะต้องคิดอัลกอริทึมในการแก้ปัญหานั้นๆ ของโปรแกรมแล้ว ยังต้องมาพะวงเรื่องของขนาดของหน่วยความจำและการจัดแบ่งขนาดของ overlay อีกด้วย ในรูปที่ 9.6 แสดงให้เห็นถึงตัวอย่างของการแบ่งโปรแกรมออกเป็นส่วนๆ โดยสมมุติว่าหน่วยความจำหลักของระบบมีให้ใช้แค่ 4 เมกกะไบต์ และโปรแกรมที่เขียนคือตัวซอฟต์แวร์พัฒนาโปรแกรมด้วยภาษา C ซึ่งประกอบไปด้วย 5 ส่วน โดยสมมุติให้ส่วน lookup table เป็นส่วนที่ต้องใช้ตลอดเวลาในการทำงาน แต่ส่วนที่เหลือ 4 ส่วนได้แก่ compiler, assembler, debugger และ editor โดยแต่ละส่วนจะถูกรันไม่พร้อมกันในขณะทำงาน ดังนั้นโปรแกรมเมอร์จึงแบ่งโปรแกรมออกเป็นส่วนๆ อย่างชัดเจนตอนเขียนโปรแกรม และใช้ overlay driver ทำการโหลดส่วนของโปรแกรมเข้ามาในหน่วยความจำเมื่อต้องการใช้ส่วนนั้นๆ เท่านั้น



รูปที่ 9.6 การจัดการโปรแกรมแบบ Overlay

ในระบบหน่วยความจำเสมือน ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมได้อย่างอิสระโดยไม่จำเป็นต้องคำนึงถึงข้อจำกัดว่าหน่วยความจำที่มีอยู่จริงของระบบมีขนาดเท่าใด ในมุมมองของโปรแกรมเมอร์จะมองว่าหน่วยความจำที่ระบบมีให้มีขนาดใหญ่ไม่จำกัด

แอดเดรสของหน่วยความจำที่โปรแกรมอ้างถึงจะเรียกว่า Logical address โดยค่าแอดเดรสนี้จะแบ่งออกเป็นส่วนย่อยๆ เรียกว่า page โดยขนาดของ page จะมีขนาดตั้งแต่ 512-8,192 ไบต์แล้วแต่สถาปัตยกรรมของคอมพิวเตอร์แต่ละแบบ และหน่วยความจำหลักที่ระบบมีอยู่จริงจะแบ่งออกเป็นหน่วยๆ เรียกแต่ละหน่วยว่า frame โดยขนาดของเฟรมจะมีขนาดเท่ากับขนาดของ page ในรูปที่ 9.7 แสดงให้เห็นตัวอย่างการเขียนโปรแกรมบนระบบฮาร์ดแวร์ซึ่งสนับสนุนเทคนิคหน่วยความจำเสมือน โดยที่โปรแกรมเมอร์เขียนโปรแกรมโดยไม่สนใจว่าระบบมีหน่วยความจำใช้งานจริงขนาดเท่าใด แต่สมมุติว่าหลังคอมไพล์โปรแกรมแล้ว ได้ขนาดของโปรแกรมซึ่งต้องการเนื้อที่ของหน่วยความจำเป็นจำนวน 201 เพจ แต่เรานำโปรแกรมนี้ไปรันบนเครื่องคอม-

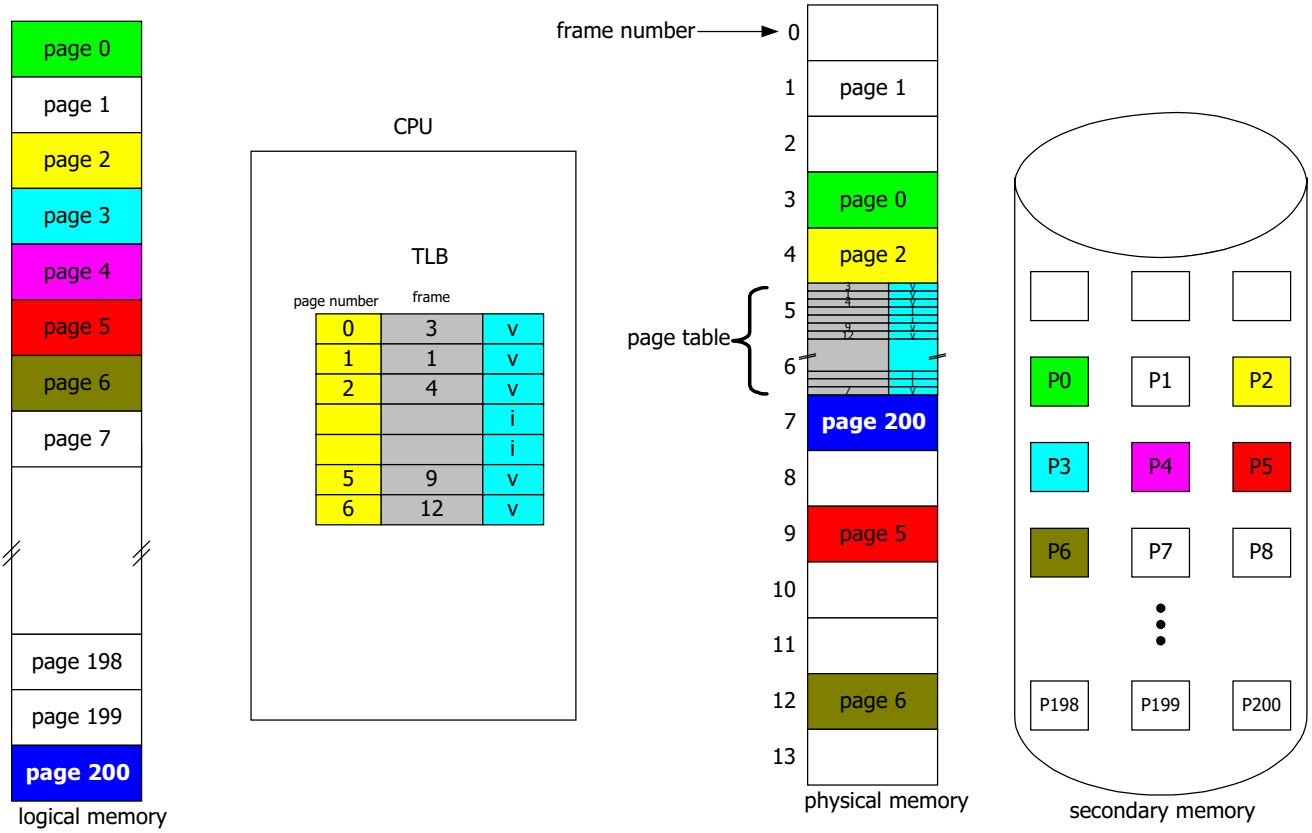


รูปที่ 9.7 ระบบหน่วยความจำเสมือน

-พิวเตอร์ที่มีขนาดหน่วยความจำบนเมนบอร์ดให้ใช้เพียง 14 เพจเท่านั้น ในการรันโปรแกรม ซีพียูจะอ่านโปรแกรมจากดิสก์เฉพาะ page ที่ต้องการใช้งานจริงเข้ามาใส่ใน frame ของหน่วยความจำ โดยการใส่ page ที่ต้องการใช้งานไว้ใน frame ไหนก็ได้ที่ว่างในหน่วยความจำหลัก (physical memory) ระบบจะมีส่วนของ page table เอาไว้สำหรับการระบุว่า page ใดของโปรแกรมอาศัยอยู่ในหน่วยความจำ frame ที่เท่าใด นอกจากนี้ใน page table จะมีส่วนของ valid bit เอาไว้บอกว่าขณะนั้น page ที่ระบุอาศัยอยู่ในหน่วยความจำหลักหรือไม่ โดยหากค่าบิตนี้มีสภาวะเป็น v (valid) จะหมายถึงว่ามี page ของโปรแกรมที่ต้องการอยู่ใน frame ที่ระบุ แต่ถ้าหาก valid bit มีค่าเป็น i(invalid) จะหมายถึงว่า page หมายเลขนั้นของโปรแกรมไม่ได้มีอยู่ใน frame ของหน่วยความจำแต่จะอาศัยอยู่ในดิสก์ซึ่งถือว่าเป็นอยู่ในหน่วยความจำประเภท secondary storage memory

เทคนิคหน่วยความจำเสมือนช่วยแก้ปัญหาเรื่อง program relocation ได้ในตัวของมันเอง ดังนั้นซีพียูที่สนับสนุนเทคนิคหน่วยความจำเสมือนจึงไม่มีรีจิสเตอร์ทำหน้าที่ relocation อีกเหมือนอย่างที่ใช้ในซีพียูสมัยก่อนๆ และเนื่องจาก page table ใช้ในการเก็บค่าว่า แต่ละ page ของโปรแกรมเก็บอยู่ใน frame ใดของหน่วยความจำหลัก และขนาดของ page table มักจะใหญ่เกินกว่าที่จะสามารถสร้างขึ้นได้ในซีพียู ดังนั้นในทางปฏิบัติจึงต้องเก็บตัว page table นี้ไว้ใน

หน่วยความจำ ดังนั้นจะเห็นว่าเทคนิค virtual memory นี้ในการอ่านค่าจากหน่วยความจำแต่ละครั้งของโปรแกรมจะต้องมีการเข้าถึงหน่วยความจำจริง 2 ครั้ง โดยครั้งแรกจะเป็น page table และครั้งที่สองจะเป็นการเข้าถึงหน่วยความจำ frame ที่ page table ระบุ ดังนั้นจะทำให้การทำงานของระบบช้าลงได้ เพื่อที่จะแก้ไขปัญหาดังกล่าว ผู้ออกแบบซีพียูส่วนใหญ่จะออกแบบให้มีตารางเก็บค่า page table ที่ถูกใช้บ่อยๆ เอาไว้ในซีพียูโดยเรียกตารางนี้ว่า Translation Lookaside Buffer(TLB) โดยที่ตาราง TLB นี้จะทำหน้าที่เป็นเสมือนแคช (cache) ของ page table ที่ใช้งานบ่อยๆ ของซีพียู ดังแสดงในรูปที่ 9.8



รูปที่ 9.8 TLB และ page table ในระบบหน่วยความจำเสมือน

References

1. Robert J, baron., *"Computer Architecture"*, Addison-Wesley publishing, 1992, pages 143-145, 179-186
2. Silberschatz, Abraham., *"Operating System Concepts"*, Addison-Wesley publishing, 4th edition, 1995, pages 45-50, 301-309, 267-275
3. Hamacher, Car., *"Computer Organization"*, McGraw-Hill publishing, 5th edition, 2002, pages 337-343