บทที่ 6
การอัดคำสั่งแบบปรับปรุง

ในบทนี้นำเสนอวิธีการอัดคำสั่งแบบปรับปรุงสำหรับหน่วยประมวลผล การอัดคำสั่งแบบปรับปรุงนี้สามารถอัดคำสั่งให้ได้โปรแกรมที่มีขนาดเล็กกว่า โปรแกรมที่ได้จากการอัดคำสั่งที่ได้นำเสนอในบทที่ 4 แต่เนื่องจากการอัดคำสั่งด้วยวิธีปรับปรุงนี้ ต้องการตัวสร้างโค้ด (Code generator) ที่มีคุณภาพ เพื่อให้ได้โปรแกรมที่มีประสิทธิภาพสูงกว่าโปรแกรมที่ได้จากการอัดคำสั่งในบทที่ 4 ในที่นี้จึงนำเสนอแนวคิดและวิธีการอัดคำสั่งแบบปรับปรุงนี้ โดยไม่ได้ทำการทดสอบด้านประสิทธิภาพการทำงานของโปรแกรมที่ได้จากการอัดคำสั่งแบบปรับปรุงนี้

เพื่อความกะทัดรัดของเนื้อหาในบทนี้ การอัดคำสั่งที่ได้นำเสนอในบทที่ 4 จะเรียกว่า "การอัดคำสั่งแบบธรรมดา" เพื่อไม่ให้เกิดความสับสนกับ "การอัดคำสั่งแบบปรับปรุง" ในบทนี้

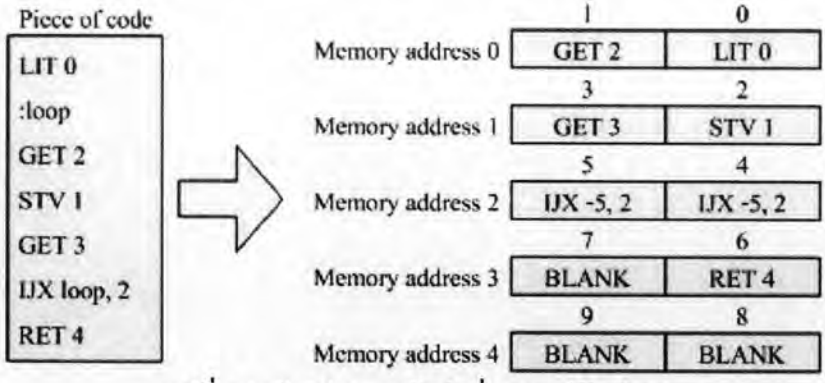
วิธีการอัดคำสั่งที่ได้นำเสนอไปในบทที่ 4 มีข้อจำกัดสามประการ ที่ทำให้ไม่สามารถอัดคำสั่งได้ เนื่องจากบางคำสั่งในโปรแกรมไม่อนุญาตให้ใส่ไว้ในคำสั่งเติมเต็ม คือ (1) คำสั่งนั้นเป็นปลายทางของการกระโดด (2) คำสั่งนั้นเป็นคำสั่งแรกของโปรแกรมย่อย กล่าวคือเป็นปลายทางของการเรียกฟังก์ชันนั้นเอง (3) คำสั่งนั้นเป็นคำสั่งแรกหลังจากการคืนค่าจากโปรแกรมย่อย ข้อจำกัดทั้งสามนี้เกิดขึ้นเนื่องจากหน่วยประมวลผลนี้ ไม่มีโครงสร้างสำหรับการอ้างอิงถึงคำสั่งที่อยู่ในคำสั่งเติมเต็มโดยตรง ทำให้ไม่สามารถทำการกระโดดไปกระทำคำสั่งในคำสั่งเติมเต็มได้

การอัดคำสั่งแบบปรับปรุงที่นำเสนอในบทนี้ จำเป็นต้องทำการเพิ่มโครงสร้างสำหรับอ้างอิงถึงคำสั่งเติมเต็ม ซึ่งทำได้โดยการกำหนดเลขที่อยู่ให้กับคำสั่งเติมเต็ม สำหรับใช้ในการอ้างอิงถึงคำสั่งเติมเต็มในการกระโดดและเรียกโปรแกรมย่อยได้ ซึ่งจะกำจัดข้อจำกัดทั้ง 3 ประการของการอัดคำสั่งแบบธรรมดาออกไปได้

6.1 การกำหนดเลขที่อยู่ของคำสั่งใหม่

เลขที่อยู่ที่กำหนดให้กับคำสั่งเติมเต็มนี้เป็นเลขที่อยู่เทียม กล่าวคือ เลขที่อยู่ดังกล่าวมิใช่เลขที่อยู่ของหน่วยความจำ แต่เป็นเพียงเลขที่อยู่ที่กำหนดขึ้นมาเพื่อใช้ในการระบุถึงคำสั่งที่อยู่ในตำแหน่งคำสั่งเติมเต็มเท่านั้น ดังแสดงในรูปที่ 6.1 ตัวเลขที่กำกับอยู่เหนือคำสั่งคือ เลขที่อยู่ชุดใหม่ที่มีการกำหนดเลขที่อยู่ให้กับคำสั่งเติมเต็มด้วย เลขที่อยู่ 0 และ 1 อยู่ในหน่วยความจำตำแหน่งเดียวกันคือ หน่วยความจำตำแหน่งที่ 0 โดยเลขที่อยู่ 0 อยู่ในตำแหน่งบิตต่ำ ซึ่งเป็นตำแหน่งของคำสั่งฐาน และเลขที่อยู่ 1 อยู่ในตำแหน่งบิตบน ซึ่งเป็นตำแหน่งของคำสั่งเติมเต็ม

รูปที่ 6.1 เป็นตัวอย่างการจัดเรียงคำสั่งลงในหน่วยความจำโดยใช้วิธีการอัดคำสั่งแบบปรับปรุง จะเห็นว่าด้วยการกำหนดเลขที่อยู่แบบใหม่ ทำให้คำสั่ง IJX ในโปรแกรมนี้สามารถกระโดดไปยังคำสั่ง GET 2 ซึ่งอยู่ในตำแหน่งคำสั่งเติมเต็มได้



รูปที่ 6.1 ตัวอย่างการอัดคำสั่งแบบปรับปรุง

การกำหนดเลขที่อยู่แบบใหม่นี้ ใช้ในการเข้าถึงหน่วยความจำส่วนโปรแกรม (Code segment) เท่านั้น ทั้งนี้เพื่อไม่ให้เกิดผลกระทบต่อการทำงานในส่วนอื่นๆ เช่น หากหน่วยความจำส่วนแสตก (Stack segment) ใช้การกำหนดเลขที่อยู่แบบใหม่นี้ จะทำให้จำนวนตัวแปรเฉพาะที่ (Local variable) ที่แต่ละโปรแกรมย่อยจะมีได้ลดลงเหลือครึ่งหนึ่ง คือ 128 ตัวแปร เนื่องจากเลขที่อยู่แบบใหม่ต้องใช้ที่อยู่ 2 ที่ในการเก็บตัวแปรหนึ่งตัว

การกำหนดเลขที่อยู่แบบใหม่นี้มีข้อเสียคือ ทำให้ระยะกระโดดของคำสั่งกระโดดนั้นสั้นลง จากการที่เลขที่อยู่หนึ่งที่นั้นมีคำสั่งน้อยลง จากเดิมเลขที่อยู่ที่หนึ่งอาจมีคำสั่งถึง 2 คำสั่ง หรือมีเพียงคำสั่งเดียวเป็นอย่างน้อย แต่ในการกำหนดเลขที่อยู่แบบใหม่ เลขที่อยู่ที่หนึ่งมีคำสั่งได้เพียงหนึ่งคำสั่ง หรืออาจเป็นเพียงครึ่งคำสั่งในกรณีที่คำสั่งดังกล่าวเป็นคำสั่งรูปแบบยาว

อย่างไรก็ตาม ข้อเสียนี้ไม่ส่งผลกระทบมากนัก เนื่องจากแม้ว่าจะทำการกระโดดได้สั้นลง แต่ระยะกระโดดที่สั้นลงนั้น ยังเพียงพอสำหรับคำสั่งกระโดดส่วนใหญ่ในโปรแกรม กล่าวคือ คำสั่งกระโดดที่มีระยะทางไกลนั้น เป็นส่วนน้อยของคำสั่งกระโดดทั้งหมดในโปรแกรม และหาได้ยาก เหตุการณ์ที่คำสั่งกระโดดมีระยะทางไกลจนต้องใช้คำสั่งกระโดดในรูปแบบยาวนั้นจึงไม่เกิดขึ้นบ่อยนัก

6.2 การปรับเปลี่ยนวงจรหน่วยประมวลผลเพื่อรองรับการอัดคำสั่งแบบปรับปรุง

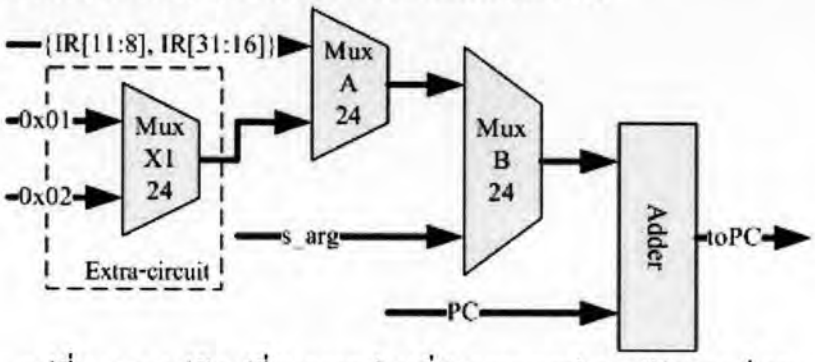
การกำหนดเลขที่อยู่แบบใหม่ที่ทำในการอัดคำสั่งแบบปรับปรุง แม้ว่าจะกระทำกับหน่วยความจำส่วนโปรแกรม (Code segment) เพียงส่วนเดียว แต่ก็ส่งผลกระทบกับวงจรหน่วยประมวลผลในหลายๆ จุด โดยการเปลี่ยนแปลงส่วนใหญ่จะอยู่ในทางเดินข้อมูลส่วนที่เกี่ยวข้องกับคำสั่ง เช่น วงจรที่เกี่ยวข้องกับรีจิสเตอร์ PC และวงจรหน่วยควบคุม

6.2.1 วงจรส่วนที่ทำการบวกค่าของรีจิสเตอร์ PC

ส่วนแรกที่จะกล่าวถึงคือวงจรส่วนที่ทำการบวกค่าของรีจิสเตอร์ PC เพื่อให้ค่านั้นชี้ไปที่คำสั่งถัดไป จากเดิมที่การอ่านคำสั่งทุกครั้งค่าของรีจิสเตอร์ PC จะเพิ่มขึ้น 1 ค่า เนื่องจากคำสั่งที่กระทำอยู่ได้รับการกำหนดเลขที่อยู่ให้หนึ่งค่าเสมอ แต่ในการกำหนดเลขที่อยู่แบบใหม่ คำสั่ง

รูปแบบยาวจะได้รับการกำหนดเลขที่อยู่ให้สองค่า ในขณะที่คำสั่งรูปแบบสั้นได้รับเลขที่อยู่หนึ่งค่า ทำให้การบวกค่าของรีจิสเตอร์ PC ต้องคำนึงถึงรูปแบบคำสั่งที่อ่านได้ด้วย เพื่อให้ค่าของรีจิสเตอร์ PC ชี้ไปที่คำสั่งถัดไปอย่างถูกต้อง

ในรูปที่ 6.2 แสดงวงจรส่วนที่ทำการบวกค่าของรีจิสเตอร์ PC สำหรับการกำหนดเลขที่อยู่แบบใหม่ ค่าสำหรับบวกเพิ่มให้รีจิสเตอร์ PC เป็นได้ทั้ง 1 หรือ 2 จึงต้องเพิ่มวงจรรวมส่งสัญญาณ (MUX) หนึ่งตัว เพื่อใช้เลือกว่าค่าของรีจิสเตอร์ PC จะเพิ่มขึ้นเท่าใด

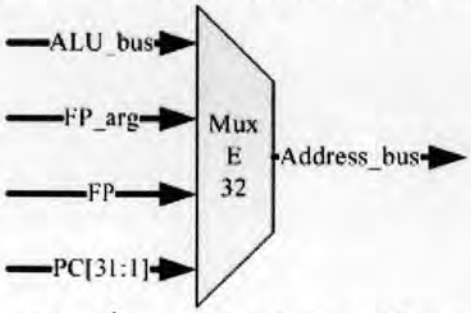


รูปที่ 6.2 การปรับเปลี่ยนวงจรส่วนที่ทำการบวกค่าของรีจิสเตอร์ PC

6.2.2 วงจรส่วนนำค่าของรีจิสเตอร์ PCไปใช้

ค่าของรีจิสเตอร์ PC ใช้ในการระบุเลขที่อยู่ของคำสั่งถัดไป การกำหนดเลขที่อยู่แบบใหม่นี้ทำให้การอ่านคำสั่งเปลี่ยนไปจากเดิม วิธีการใช้ค่าของรีจิสเตอร์ PC จึงเปลี่ยนไปด้วย จากเดิมที่นำค่าของรีจิสเตอร์ PC ระบุเลขที่อยู่ของหน่วยความจำที่ต้องการอ่านโดยตรง เมื่อกำหนดเลขที่อยู่แบบใหม่ค่าที่ใช้ระบุเลขที่อยู่ของหน่วยความจำจะเป็นค่าของรีจิสเตอร์ PC บิต 31 ถึงบิต 1 แทน เนื่องจากค่าของรีจิสเตอร์ PC บิต 0 นั้นใช้ในการระบุว่ากระทำคำสั่งที่อยู่ในตำแหน่งคำสั่งฐานหรือคำสั่งเติมเต็ม ไม่ได้เป็นตัวระบุเลขที่อยู่ของคำสั่งในหน่วยความจำ

ค่าของรีจิสเตอร์ที่จะนำออกสู่สายสัญญาณ Address_bus เพื่อระบุเลขที่อยู่สำหรับการอ่านหน่วยความจำ ผ่านวงจรรวมส่งสัญญาณ MUXE ดังแสดงในรูปที่ 6.3 จะเห็นว่ามีเฉพาะค่าของรีจิสเตอร์ PC เท่านั้น ที่ใช้บิต 31 ถึงบิต 1 ในการอ่านหน่วยความจำ ค่าอื่นๆ ยังคงเหมือนเดิม เนื่องจากการกำหนดเลขที่อยู่แบบใหม่นี้กระทำกับหน่วยความจำส่วนโปรแกรมเท่านั้น



รูปที่ 6.3 การปรับเปลี่ยนวงจรส่วนนำค่าของรีจิสเตอร์ PC ไปใช้

6.2.3 วงจรหน่วยควบคุม

เนื่องจากการทำงานโปรแกรมที่ได้รับการอัดคำสั่งแบบปรับปรุงนั้น แตกต่างจากการทำงานโปรแกรมที่ได้รับการอัดคำสั่งแบบธรรมดาค่อนข้างมาก การปรับเปลี่ยนวงจรหน่วยควบคุม (Control unit) จึงเป็นส่วนที่สำคัญที่สุด

การทำงานโปรแกรมที่ได้รับการอัดคำสั่งแบบปรับปรุงยังคงมี 3 ขั้นตอนเช่นเดิม และมีสถานะการทำงาน เหมือนสถานะการทำงานของหน่วยประมวลผลที่รองรับการอัดคำสั่งแบบธรรมดา ซึ่งแสดงในแผนภูมิสถานะในรูปที่ 4.7 การทำงานเดิมที่มีในแต่ละสถานะยังคงเหมือนเดิม แต่ในจะมีการทำงานบางส่วนถูกเปลี่ยนแปลงหรือเพิ่มขึ้นมาเพื่อรองรับการกำหนดเลขที่อยู่แบบใหม่คือ การเพิ่มค่าของรีจิสเตอร์ PC และการรับค่าของรีจิสเตอร์ IR

การปรับเปลี่ยนวงจรหน่วยควบคุม จึงเป็นการปรับเปลี่ยนวงจรส่วนที่ทำหน้าที่สร้างสัญญาณควบคุมรีจิสเตอร์ PC และรีจิสเตอร์ IR โดยสัญญาณควบคุมรีจิสเตอร์ทั้งสอง เดิมทำหน้าที่ควบคุมการรับค่าจากภายนอกของรีจิสเตอร์ทั้งสองตัว

การรับค่าจากภายนอกของรีจิสเตอร์ PC ทำเพื่อรับค่าเลขที่อยู่ของคำสั่งถัดไปเข้ามาเก็บในรีจิสเตอร์ ในการทำงานแบบปกติของหน่วยประมวลผล รีจิสเตอร์ PC จะมีการรับค่าในสถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD) และสถานะกระทำคำสั่งกระโดด (ST_JUMP) กรณีที่ทำการกระโดดเท่านั้น การรับค่าในสถานะดึงคำสั่งและถอดรหัสคำสั่ง กระทำเพื่อให้รีจิสเตอร์ PC ชี้ไปยังคำสั่งถัดไป หลังจากที่ได้อ่านคำสั่งมาจากหน่วยความจำแล้ว ส่วนในสถานะกระทำคำสั่งกระโดด กระทำเพื่อรับค่าเลขที่อยู่ของคำสั่งที่เป็นปลายทางของการกระโดด เพื่อเตรียมอ่านคำสั่งในตำแหน่งนั้นต่อไป

การกำหนดเลขที่อยู่แบบใหม่ ทำให้การเพิ่มค่าของรีจิสเตอร์ PC ต้องกระทำบ่อยขึ้น เนื่องจากคำสั่งในตำแหน่งคำสั่งฐานและคำสั่งในตำแหน่งคำสั่งเติมเต็ม มีเลขที่อยู่ต่างกัน โดยปกติพฤติกรรมการทำงานของหน่วยประมวลผล ค่าของรีจิสเตอร์ PC ในขณะที่กระทำคำสั่งใดๆ นั้นจะต้องเป็นเลขที่อยู่ของคำสั่งถัดไปเสมอ เพื่อให้กระทำคำสั่งกระโดดได้อย่างถูกต้อง ดังนั้นเมื่อกระทำคำสั่งใดๆเสร็จ รีจิสเตอร์ PC จะต้องทำการเพิ่มค่าทันที ค่าที่เพิ่มให้กับรีจิสเตอร์ PC ในแต่ละครั้งนั้น ขึ้นอยู่กับปัจจัยอื่นๆ ดังแสดงในตารางที่ 6.1

ในตอนที่กระทำคำสั่งรูปแบบสั้นที่อยู่ในตำแหน่งคำสั่งเติมเต็มเสร็จนั้น ไม่จำเป็นต้องเพิ่ม เนื่องจากหน่วยประมวลผลจะเข้าสู่สถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD) ที่มีการเพิ่มค่าให้อยู่แล้ว เช่นเดียวกับการกระทำคำสั่งรูปแบบยาวที่อยู่ในตำแหน่งคำสั่งฐาน ที่เมื่อกระทำเสร็จจะเข้าสู่สถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD) เช่นกัน

ตารางที่ 6.1 ค่าที่เพิ่มให้กับรีจิสเตอร์ PC ในแต่ละสถานการณ์

สถานการณ์	ค่าที่เพิ่ม
สถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD)	
- คำสั่งที่อ่านได้เป็นคำสั่งรูปแบบสั้น	1
- คำสั่งที่อ่านได้เป็นคำสั่งรูปแบบยาว	2
สถานะสุดท้ายของการกระทำคำสั่งรูปแบบสั้น	
- คำสั่งปัจจุบันอยู่ในตำแหน่งคำสั่งฐาน (ค่าของรีจิสเตอร์ PC บิต 0 เป็น 1)	1
- คำสั่งปัจจุบันอยู่ในตำแหน่งคำสั่งเติมเต็ม (ค่าของรีจิสเตอร์ PC บิต 0 เป็น 0)	0
สถานะสุดท้ายของการกระทำคำสั่งรูปแบบยาว	
- คำสั่งปัจจุบันอยู่ในตำแหน่งคำสั่งฐาน (ค่าของรีจิสเตอร์ PC บิต 0 เป็น 0)	0
- คำสั่งปัจจุบันอยู่ในตำแหน่งคำสั่งเติมเต็ม (ค่าของรีจิสเตอร์ PC บิต 0 เป็น 1)	1
สถานะดึงคำสั่งและถอดรหัสคำสั่งพิเศษ (ST_EXIFD)	1

ส่วนสัญญาณควบคุมรีจิสเตอร์ IR นั้น เนื่องจากการอัดคำสั่งแบบปรับปรุงอนุญาตให้ทำการกระโดดไปยังคำสั่งเติมเต็มได้ การรับค่าของรีจิสเตอร์ IR จึงเปลี่ยนไปเมื่อทำการกระโดดไปยังคำสั่งเติมเต็ม โดยเปลี่ยนไปเฉพาะการรับค่าในสถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD) เท่านั้น เนื่องจากหลังจากกระโดดหน่วยประมวลผลจะเข้าสู่สถานะนี้ การรับค่าในสถานะดึงคำสั่งและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) ยังคงเหมือนเดิม ดังแสดงในตารางที่ 6.2 สัญลักษณ์ที่ใช้อธิบายในตารางที่ 6.3 ส่วนการเลื่อนค่าของรีจิสเตอร์ IR นั้นยังคงเหมือนที่แสดงในบทที่ 4 ตารางที่ 4.3

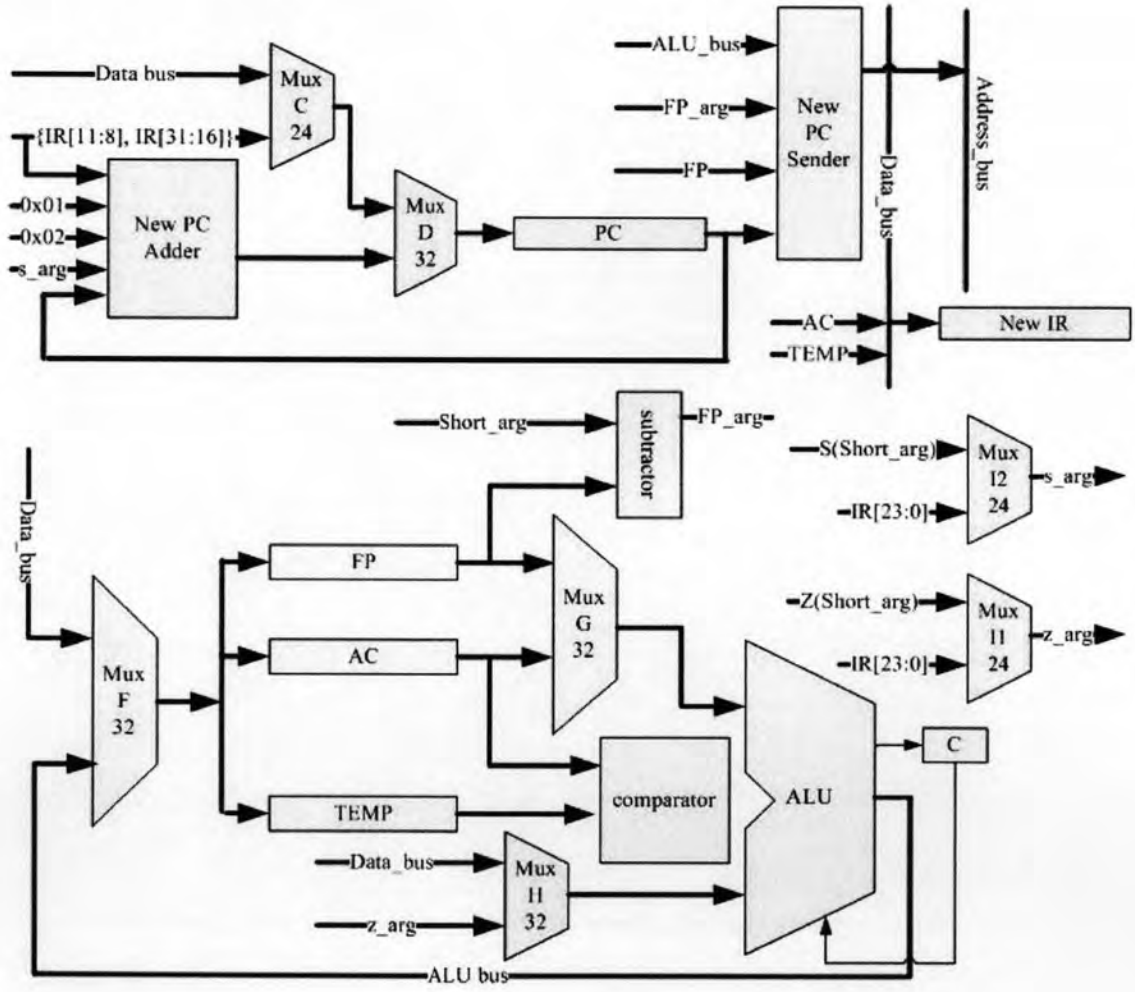
ในส่วนสุดท้ายจึงทำการสรุปการปรับเปลี่ยนทั้งหมด รูปทางเดินข้อมูล (Data path) ของหน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุง แสดงในรูปที่ 6.4

ตารางที่ 6.2 การรับค่าเข้ารีจิสเตอร์ IR ในแต่ละสถานการณ์

สถานการณ์	ข้อมูลในรีจิสเตอร์ IR
สถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD)	
- ค่าของรีจิสเตอร์ PC บิต 0 เป็น 0	IR ← {BLANK, Data_bus};
- ค่าของรีจิสเตอร์ PC บิต 0 เป็น 1	IR ← {BLANK, BLANK, Data_bus[31:16]};
สถานะดึงคำสั่งและถอดรหัสคำสั่งพิเศษ (ST_EXIFD)	IR ← {Data_bus, IR[15:0]};

ตารางที่ 6.3 สัญลักษณ์ที่ใช้ในการอธิบายการทำงานของรีจิสเตอร์ IR

สัญลักษณ์	คำอธิบาย
IR	ค่าของรีจิสเตอร์ IR
Data_bus	ค่าของสายสัญญาณ Data_bus ในทางเดินข้อมูล
BLANK	คำสั่งสงวนขนาด 16 บิตซึ่งมีรหัสเท่ากับ 0x3F00 เป็นค่าที่ระบุว่าข้อมูลในตำแหน่งดังกล่าวของรีจิสเตอร์ IR นั้นว่างเปล่า
X[Y:Z]	ค่าของสัญญาณ X บิต Y ถึงบิต Z
{X, Y}	สัญญาณที่ได้จากการนำสัญญาณ X และ Y มารวมกัน โดยสัญญาณ X อยู่ในตำแหน่งบิตสูง และสัญญาณ Y อยู่ในตำแหน่งบิตต่ำ
X ← Y;	ค่าของรีจิสเตอร์ X เท่ากับสัญญาณ Y



รูปที่ 6.4 ทางเดินข้อมูลของหน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุง

6.3 วิเคราะห์ประสิทธิภาพของการอัดคำสั่งแบบปรับปรุง

ผู้เขียนได้ทำการเปรียบเทียบคุณสมบัติของวิธีการอัดคำสั่งแบบปรับปรุง กับวิธีการอัดคำสั่งที่ได้นำเสนอในบทที่ 4 ของวิทยานิพนธ์เล่มนี้ โดยใช้โปรแกรมทดสอบ 7 โปรแกรมดังแสดงในตารางที่ 6.4 โปรแกรมที่ได้รับการอัดคำสั่งด้วยวิธีปรับปรุงนี้ ได้สร้างโดยตัวสร้างโค้ด (Code generator) ที่ทำหน้าที่สร้างโค้ดให้มีขนาดเล็กที่สุด กล่าวคือ ไม่มีการปรับแต่งโค้ดที่สร้างให้มีประสิทธิภาพสูงขึ้นแต่อย่างใด

เมื่อทำการเปรียบเทียบด้านขนาดของโปรแกรมที่ได้จากวิธีการอัดคำสั่งทั้ง 2 วิธี โปรแกรมที่ได้จากวิธีการอัดคำสั่งแบบปรับปรุงมีขนาดเล็กกว่า เนื่องจากสามารถทำการอัดคำสั่งได้กับคำสั่งทั้งหมด จากโปรแกรมทดสอบทั้ง 7 โปรแกรม แสดงในตารางที่ 6.4 พบว่าการอัดคำสั่งแบบปรับปรุงนี้สามารถลดขนาดของโปรแกรมได้มากกว่าวิธีการอัดคำสั่งที่ได้นำเสนอในบทที่ 4 ประมาณร้อยละ 3.7 ดังแสดงในตารางที่ 6.5

ตารางที่ 6.4 โปรแกรมที่ใช้ในการทดสอบ

ชื่อโปรแกรม	คำอธิบาย
Bubble	Bubble sort ข้อมูลจำนวน 100 ตัว ข้อมูลเริ่มต้นเรียงจากมากไปน้อย
Merge	Merge sort ข้อมูลจำนวน 100 ตัว ข้อมูลเริ่มต้นเรียงจากมากไปน้อย
Quick	Quick sort ข้อมูลจำนวน 100 ตัว ข้อมูลเริ่มต้นเรียงจากมากไปน้อย
Fibo	หาค่าของลำดับที่ 10 ของลำดับ Fibonacci
Hanoi	แก้ปัญหาฮานอย 6 จาน (6 disks Hanoi problem)
Sieve	หาจำนวนเฉพาะทุกตัวที่มีค่าน้อยกว่า 100
AES	เข้ารหัสเออีเอส (Advance Encryption Standard : AES) ข้อมูลขนาด 128 บิต ด้วยคีย์ขนาด 128 บิต

ด้านความถี่สูงสุดของสัญญาณนาฬิกาที่หน่วยประมวลผลสามารถทำงานได้ หน่วยประมวลผลที่รองรับการอัดคำสั่งแบบธรรมดาที่นำเสนอในบทที่ 4 สามารถทำงานได้ที่ความถี่สูงกว่าหน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุง ดังแสดงในตารางที่ 6.6 ทั้งนี้เนื่องจากวงจรหน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุงนั้นมีความซับซ้อนมากกว่า และสิ่งนี้ก็เป็นเหตุให้วงจรหน่วยประมวลผลนี้ มีขนาดใหญ่กว่าวงจรหน่วยประมวลผลที่รองรับการอัดคำสั่งในบทที่ 4 ดังแสดงในตารางที่ 6.7

ตารางที่ 6.5 การเปรียบเทียบขนาดของโปรแกรมที่ได้รับการอัดคำสั่งแบบปรับปรุงกับโปรแกรมที่ได้รับการอัดคำสั่งแบบธรรมดา

โปรแกรมทดสอบ	ขนาดของโปรแกรม (ไบต์)			ขนาดที่ลดลง (%)	
	ไม่ได้รับการอัดคำสั่ง	อัดคำสั่งแบบธรรมดา	อัดคำสั่งแบบปรับปรุง	อัดคำสั่งแบบปกติ	อัดคำสั่งแบบปรับปรุง
Bubble	100	68	62	32.0	38.0
Merge	380	212	198	44.2	47.9
Quick	252	168	154	33.3	38.9
Fibo	56	36	32	35.7	42.9
Hanoi	180	108	106	40.0	41.1
Sieve	152	84	84	44.7	44.7
AES	1,336	860	828	35.6	38.0
			เฉลี่ย	**37.9**	**41.6**

ตารางที่ 6.6 ความถี่สูงสุดของสัญญาณนาฬิกาที่หน่วยประมวลผลสามารถทำงานได้

ค่าความถี่สูงสุด (เมกะเฮิร์ทซ์)	
หน่วยประมวลผลที่มีการอัดคำสั่งแบบธรรมดาที่นำเสนอในบทที่ 4	**หน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุง**
63	52

ตารางที่ 6.7 ทรัพยากรที่ใช้ในการสร้างหน่วยประมวลผล

จำนวนเกตสมมูล (Equivalent gate)	
หน่วยประมวลผลที่มีการอัดคำสั่งแบบธรรมดาที่นำเสนอในบทที่ 4	**หน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุง**
13,060	14,340

6.4 สรุปประสิทธิภาพของการอัดคำสั่งแบบปรับปรุง

จากที่ได้กล่าวถึงการปรับเปลี่ยนหน่วยประมวลผลเพื่อให้รับรองการอัดคำสั่งแบบปรับปรุง จะเห็นว่าหน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุง มีการทำงานที่ซับซ้อนกว่าหน่วยประมวลผลที่รองรับการอัดคำสั่งที่ได้นำเสนอในบทที่ 4 ทรัพยากรที่ใช้ในการปรับปรุงหน่วย

ประมวลผลเดิมให้รองรับการอัดคำสั่งแบบปรับปรุงนี้จึงมากขึ้นตามไปด้วย ดังแสดงในตารางที่ 6.7 หน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุง ใช้จำนวนเกตสมมูลมากกว่าหน่วยประมวลผลที่รองรับการอัดคำสั่งในบทที่ 4 ถึงร้อยละ 9.8 และหากเปรียบเทียบกับทรัพยากรที่ใช้ในการสร้างหน่วยประมวลผลเดิมที่ไม่มีการอัดคำสั่ง หน่วยประมวลผลนี้ใช้ทรัพยากรมากกว่าถึงร้อยละ 13.1

เมื่อพิจารณาถึงวงจรหน่วยควบคุม (Control unit) ของหน่วยประมวลผลที่รองรับการอัดคำสั่งแบบปรับปรุงแล้ว วงจรหน่วยควบคุมนี้มีการทำงานที่ซับซ้อนกว่าจรหน่วยควบคุมของหน่วยประมวลผลที่รองรับการอัดคำสั่งที่นำเสนอในบทที่ 4 มาก เนื่องจากต้องรองรับการทำงานที่ซับซ้อนจากการอัดคำสั่งที่ไม่มีข้อจำกัดใดๆ ความซับซ้อนในการทำงานนี้ทำให้เวลาประวิงการแพร่กระจาย (Propagation delay) เพิ่มขึ้น ซึ่งจะส่งผลให้ค่าความถี่สูงสุดของสัญญาณนาฬิกาที่วงจรสามารถทำงานได้นั้นลดลง

ในด้านขนาดของโปรแกรมและประสิทธิภาพการทำงาน เนื่องจากวิธีการอัดคำสั่งแบบปรับปรุงสามารถอัดคำสั่งได้ดีกว่าวิธีการอัดคำสั่งธรรมดา ขนาดของโปรแกรมที่ได้รับการอัดคำสั่งแบบปรับปรุงจึงเล็กกว่าขนาดของโปรแกรมที่ได้รับการอัดคำสั่งแบบธรรมดาที่ได้นำเสนอในบทที่ 4 ในกรณีที่แย่ที่สุดขนาดของโปรแกรมที่ได้รับการอัดคำสั่งทั้งสองแบบจะมีขนาดเท่ากัน เกิดขึ้นเมื่อโปรแกรมที่ได้รับการอัดคำสั่งแบบธรรมดา ไม่มีคำสั่งอัดครึ่ง (Half–packed instruction) เลย

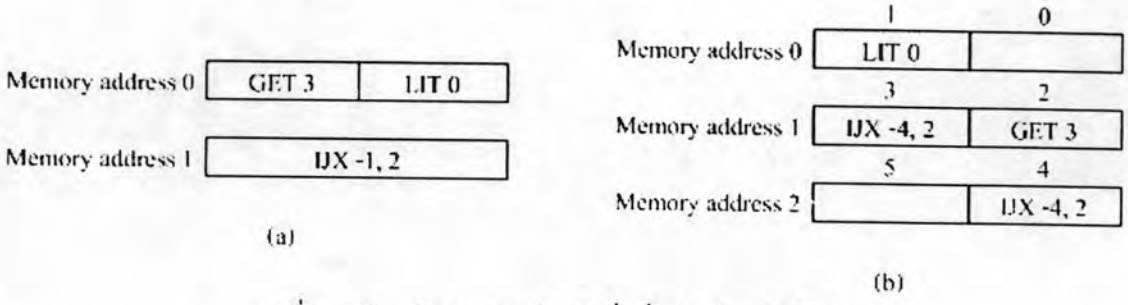
ในด้านประสิทธิภาพการทำงาน แม้ว่าการอัดคำสั่งแบบปรับปรุงนี้ ทำให้ได้โปรแกรมที่มีขนาดเล็กกว่าโปรแกรมที่ได้จากการอัดคำสั่งแบบธรรมดา แต่ในด้านประสิทธิภาพการทำงานนั้นไม่ได้สูงกว่าเสมอไป ทั้งนี้เนื่องจากการจัดวางคำสั่งในโปรแกรมที่ได้รับการอัดคำสั่งแบบปรับปรุงนั้น ไม่มีกฎเกณฑ์ข้อบังคับใดๆ ทำให้อาจเกิดวงวน (Loop) ที่จัดเรียงคำสั่งที่ไม่มีประสิทธิดังตัวอย่างในรูปที่ 6.5 ได้

วงวนที่จัดเรียงคำสั่งไม่มีประสิทธิภาพจะเกิดขึ้นกับ วงวนที่มีขนาดวัดจากจำนวนเลขที่อยู่เป็นเลขคู่ เมื่อทำการจัดเรียงคำสั่งในวงวนดังกล่าวลงในหน่วยความจำ การอ่านคำสั่งจากหน่วยความจำ N ตำแหน่งนั้น เมื่อ N เป็นเลขคู่ ใช้เวลาน้อยที่สุดเท่ากับ N/2 รอบนาฬิกา แต่หากการจัดเรียงทำได้ไม่ดีดังรูปที่ 6.5 จะทำให้การอ่านคำสั่งดังกล่าวต้องใช้เวลาเท่ากับ (N/2)+1 รอบนาฬิกา ข้อเสียนี้เห็นผลได้ชัดเมื่อเวลาส่วนใหญ่ใช้ในการทำงานในวงวนนั้นๆ

การอัดคำสั่งที่ได้นำเสนอในบทที่ 4 นั้นไม่เกิดกรณีเช่นนี้เนื่องจากคำสั่งที่เป็นปลายทางของการกระโดดหรือการเรียกโปรแกรมย่อยนั้น จะถูกบังคับให้ใส่ไว้เป็นคำสั่งแรกของหน่วยความจำตำแหน่งนั้นๆ เสมอ

รูปที่ 6.5 แสดงการเปรียบเทียบการจัดเรียงโปรแกรมของการอัดคำสั่งแบบธรรมดาที่ได้นำเสนอในบทที่ 4 แสดงในรูปที่ 6.5(a) และการจัดเรียงโปรแกรมของการอัดคำสั่งแบบปรับปรุงที่ได้นำเสนอในบทนี้ แสดงในรูปที่ 6.5(b) โดยแสดงการจัดเรียงคำสั่งลงในหน่วยความจำ โปรแกรม

มีการทำงานเป็นวงวน จะเห็นว่าหากจัดเรียงคำสั่งดังแสดงในรูปที่ 6.5 นี้ โปรแกรมที่ได้รับการอัดคำสั่งแบบธรรมดาจะทำงานวนหนึ่งรอบใช้เวลา 6 รอบนาฬิกา ซึ่งเร็วกว่าโปรแกรมที่ได้รับการอัดคำสั่งแบบปรับปรุงที่ใช้ถึง 7 รอบนาฬิกา เนื่องจากต้องทำการอ่านคำสั่งถึง 3 ครั้ง



รูปที่ 6.5 ตัวอย่างการจัดเรียงคำสั่งที่ไม่มีประสิทธิภาพ

จากที่ได้ยกตัวอย่าง พบว่าการอัดคำสั่งแบบปรับปรุงไม่มีคุณสมบัติที่จะรับประกันได้ว่าโปรแกรมที่ได้รับการอัดคำสั่งด้วยวิธีนี้ จะมีประสิทธิภาพการทำงานที่ดีกว่าโปรแกรมที่ได้รับการอัดคำสั่งแบบธรรมดาในบทที่ 4 ทั้งนี้ประสิทธิภาพการทำงานของโปรแกรมจะขึ้นอยู่กับคุณภาพของการจัดวางคำสั่งลงในหน่วยความจำ ซึ่งเป็นหน้าที่ของตัวสร้างโค้ด (Code generator)

จากที่ได้กล่าวมาทั้งหมดในบทนี้ จะเห็นได้ว่าแม้การอัดคำสั่งแบบปรับปรุงนี้จะสามารถสร้างโปรแกรมที่มีขนาดเล็กกว่าโปรแกรมที่ได้จากการอัดคำสั่งที่ได้นำเสนอไป แต่ในด้านอื่นๆ ไม่ว่าจะเป็นด้านประสิทธิภาพการทำงาน และการประหยัดทรัพยากร การอัดคำสั่งแบบปรับปรุงนี้ไม่ได้มีคุณสมบัติที่ดีกว่าการอัดคำสั่งที่ได้นำเสนอไปในบทที่ 4 วิทยานิพนธ์เล่มนี้จึงเลือกทำการอัดคำสั่งด้วยวิธีวิธีที่ได้นำเสนอในบทที่ 4 ซึ่งมีคุณประโยชน์ที่ไม่แพ้กัน แต่สามารถสร้างได้ด้วยทรัพยากรที่ต่ำกว่า และสามารถพัฒนาโปรแกรมได้ง่ายกว่า