บทที่ 4
การอัดคำสั่ง

ในบทที่แล้ว ได้นำเสนอขั้นตอนการออกแบบหน่วยประมวลผลที่ใช้ในงานวิจัยนี้ และเนื่องจากหน่วยประมวลผลที่ได้ทำการออกแบบในบทที่ 3 นั้น มีลักษณะการทำงานแบบ "อ่านคำสั่ง – กระทำคำสั่ง" ซึ่งการอ่านคำสั่งใช้เวลา 1 รอบนาฬิกา ในขณะที่การกระทำคำสั่งนั้นอาจใช้ 1 หรือ 2 รอบนาฬิกาขึ้นอยู่กับคำสั่งที่กระทำ จะเห็นได้ว่าเวลาที่ใช้ในการอ่านคำสั่งคิดเป็นร้อยละ 33 ถึง 50 ของเวลาที่หน่วยประมวลผลนี้ใช้ในการทำงาน

ดังที่ได้กล่าวมา หน่วยประมวลผลนี้ใช้เวลาในการทำงานเกือบครึ่งหนึ่ง ในการอ่านคำสั่ง ซึ่งถือเป็นอัตราส่วนที่สูงพอสมควร การลดเวลาที่ใช้ในการอ่านคำสั่งนี้ จึงสามารถช่วยเพิ่มประสิทธิภาพการทำงานของวงจรหน่วยประมวลผลได้อย่างมาก

ในบทนี้จะกล่าวถึงการเพิ่มสมรรถนะของหน่วยประมวลผลด้วยวิธีการอัดคำสั่ง (Instruction packing) การอัดคำสั่งนี้เป็นการบีบอัดโปรแกรม (Code compression) วิธีหนึ่ง เป็นการทำให้โปรแกรมนั้นมีขนาดเล็กลง และในขณะเดียวกันสามารถช่วยลดเวลาที่วงจรหน่วยประมวลผลใช้ในการอ่านคำสั่งได้ โดยการทำให้การอ่านคำสั่งแต่ละครั้งของหน่วยประมวลผลได้คำสั่งมากกว่าเดิม กล่าวคือ เดิมหน่วยประมวลผลนี้นั้นจะได้คำสั่งหนึ่งคำสั่งจากการอ่านคำสั่งครั้งหนึ่ง แต่เมื่อมีการอัดคำสั่ง การอ่านคำสั่งแต่ละครั้งของวงจรหน่วยประมวลผลอาจได้คำสั่งมากกว่าหนึ่งคำสั่ง โดยสำหรับวิธีการอัดคำสั่งที่จะได้นำเสนอในงานวิจัยนี้ การอ่านคำสั่งครั้งหนึ่งจะได้คำสั่ง 1 หรือ 2 คำสั่ง ซึ่งจะได้นำเสนอในส่วนต่อไป

4.1 วิธีการอัดคำสั่ง

การอัดคำสั่ง (Instruction packing) คือ การนำคำสั่งหลายๆ คำสั่งมารวมไว้ด้วยกัน ทำให้การอ่านหน่วยความจำครั้งหนึ่งได้คำสั่งมากกว่าหนึ่งคำสั่ง ซึ่งสามารถทำได้หลายวิธี วิธีหนึ่งที่ได้นำเสนอไปแล้วในบทที่ 2 งานวิจัยของฮีนส์ (S. Hines) และคณะ [2, 3, 4] ซึ่งทำการอัดคำสั่งโดยใช้รีจิสเตอร์คำสั่ง (Instruction register file) วิธีนี้เป็นวิธีที่ซับซ้อนและใช้ทรัพยากรค่อนข้างสูง จากการที่ต้องเพิ่มรีจิสเตอร์ไฟล์ขนาด 32 บิตจำนวน 32 ตัว สำหรับรีจิสเตอร์คำสั่ง และรีจิสเตอร์ไฟล์ขนาดเท่ากันอีกตัวหนึ่งสำหรับตารางตัวดำเนินการแบบทันที (Immediate table)

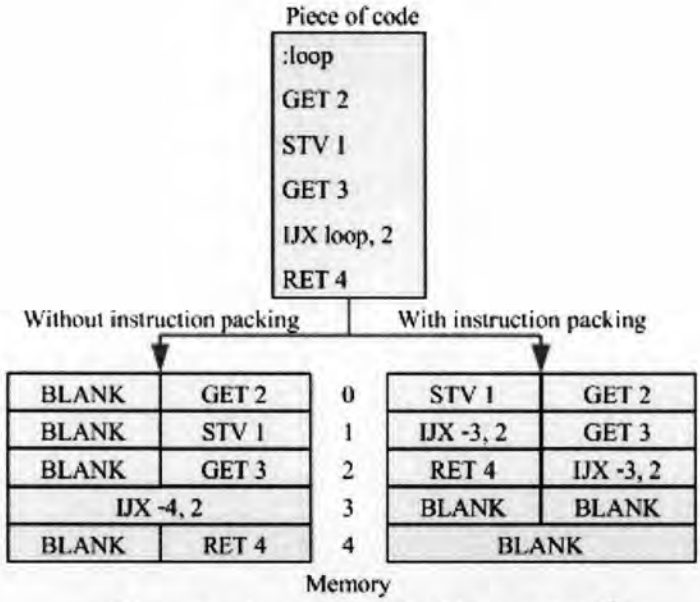
ในงานวิจัยนี้คำนึงถึงทรัพยากรที่ใช้ในการสร้างวงจรหน่วยประมวลผล การอัดคำสั่งในงานวิจัยนี้ใช้วิธีที่เรียบง่าย แต่สามารถเพิ่มประสิทธิภาพให้หน่วยประมวลผลได้ในระดับที่น่าพอใจ การอัดคำสั่งในงานวิจัยนี้ทำโดยการอนุญาตให้นำคำสั่งในรูปแบบสั้น (Short format) 2 คำสั่งมาใส่ไว้ในหน่วยความจำตำแหน่งเดียวกันได้ ดังแสดงในรูปที่ 4.1 โดยคำสั่งที่อยู่ในตำแหน่งบิตต่ำจะถูก

กระทำ (Execute) ก่อน จากนั้นจึงกระทำคำสั่งในบิตสูง แล้วจึงทำการอ่านคำสั่งในหน่วยความจำตำแหน่งถัดไป

เพื่อให้การอัดคำสั่งนี้มีประสิทธิภาพมากขึ้น สำหรับคำสั่งรูปแบบยาว (Long format) ซึ่งมีขนาด 32 บิต สามารถทำการตัดคำสั่งออกเป็น 2 ส่วนขนาด 16 บิตเท่ากัน เพื่อให้มีขนาดเท่ากับคำสั่งรูปแบบสั้น จึงสามารถทำการอัดคำสั่งได้เช่นเดียวกับที่กระทำกับคำสั่งรูปแบบสั้น โดยให้กระทำเหมือน 16 บิตต่ำ (Least significant bit) ของคำสั่งรูปแบบยาวนั้นเป็นคำสั่งรูปแบบสั้นคำสั่งหนึ่ง และ 16 บิตสูงของคำสั่งยาวเป็นคำสั่งรูปแบบสั้นคำสั่งถัดไป

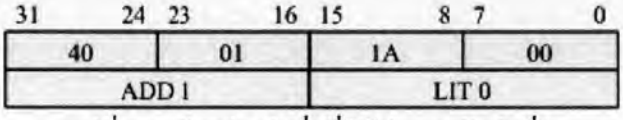
รูปที่ 4.1 ตัวอย่างโปรแกรมของหน่วยประมวลผลนี้ ทางด้านซ้ายของรูปเป็นการจัดเรียงคำสั่งลงในหน่วยความจำโดยไม่มีการอัดคำสั่ง จะเห็นว่าคำสั่งที่เป็นรูปแบบสั้น (Short format) ถูกใส่ลงในหน่วยความจำในตำแหน่งบิตต่ำ (Least significant bit) โดยทิ้งให้ตำแหน่งบิตสูง (Most significant bit) นั้นว่างไม่ได้ใช้ประโยชน์แต่อย่างใด

เปรียบกับทางด้านขวาของรูปที่ 4.1 เป็นการจัดเรียงคำสั่งลงในหน่วยความจำแบบใช้การอัดคำสั่ง จะเห็นว่าส่วนที่เคยเป็นช่องว่างนั้นสามารถใส่คำสั่งลงไปแทนได้ ทำให้ใช้พื้นที่หน่วยความจำที่ใช้ในการเก็บโปรแกรมนั้นเล็กลง



รูปที่ 4.1 ตัวอย่างการอัดคำสั่งที่ใช้ในงานวิจัยนี้

รูปที่ 4.2 แสดงตัวอย่างคำสั่งที่ได้รับการอัดคำสั่ง โดยในส่วนบิตต่ำเป็นคำสั่ง LIT 0 และในคำสั่งบิตสูงเป็นคำสั่ง ADD 1

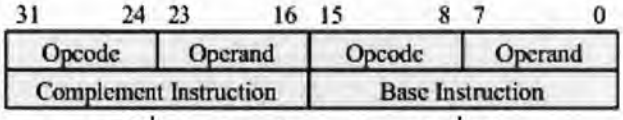


รูปที่ 4.2 ตัวอย่างคำสั่งที่ได้รับการอัดคำสั่ง

การอัดคำสั่งช่วยให้ใช้พื้นที่หน่วยความจำในการเก็บโปรแกรมน้อยลง จากการทำให้คำสั่งสั้นใช้พื้นที่ในการเก็บคำสั่งเพียงครึ่งหนึ่งของหน่วยความจำตำแหน่งหนึ่งๆ แทนที่จะเป็นหน่วยความจำทั้งตำแหน่ง อย่างไรก็ตามการอัดคำสั่งที่นำเสนอในงานวิจัยนี้ มีบางสถานการณ์ที่ไม่สามารถกระทำการอัดคำสั่งได้ซึ่งจะได้กล่าวถึงในส่วนถัดไป

ในรูปที่ 4.1 แสดงตัวอย่างการอัดคำสั่งที่ใช้ในงานวิจัยนี้ โดยได้แสดงให้เห็นความแตกต่างระหว่างโปรแกรมที่ได้ทำการอัดคำสั่ง (ทางด้านขวาของรูป) กับโปรแกรมปกติที่ไม่ได้ทำการอัดคำสั่ง (ทางด้านซ้ายของรูป) พบว่าโปรแกรมที่ได้รับการอัดคำสั่งใช้พื้นที่ในหน่วยความจำน้อยกว่าโปรแกรมปกติ นอกจากนี้การอัดคำสั่งช่วยลดเวลาที่ใช้ในการดึงคำสั่ง (Instruction fetch) จึงช่วยเพิ่มสมรรถนะให้กับหน่วยประมวลผลได้อีกด้วย

ในเนื้อหาที่จะกล่าวถึงในส่วนต่อไปนี้ ในหน่วยความจำที่เก็บโปรแกรม คำสั่งในตำแหน่งบิตต่ำ (บิต 15 ถึงบิต 0) ซึ่งเป็นคำสั่งที่ถูกกระทำก่อนจะเรียกว่า "คำสั่งฐาน" (Base instruction) และคำสั่งในตำแหน่งบิตสูง (บิต 31 ถึงบิต 16) ซึ่งเป็นคำสั่งที่ถูกกระทำทีหลังจะเรียกว่า "คำสั่งเติมเต็ม" (Complement instruction) ดังแสดงในรูปที่ 4.3



รูปที่ 4.3 ส่วนประกอบของคำสั่งอัด

สำหรับหน่วยความจำตำแหน่งที่ข้อมูลทั้ง 32 บิตนั้น ถูกใช้ในการเก็บคำสั่งทั้งหมด กล่าวคือ ไม่มีช่องว่างถูกเก็บอยู่ในตำแหน่งคำสั่งเติมเต็ม เรียกข้อมูล 32 บิตนี้ว่า "คำสั่งอัดที่ได้รับการอัดเต็ม" (Full-packed instruction) และคำสั่งอัดที่คำสั่งเติมเต็มเป็นคำสั่ง BLANK เนื่องจากไม่สามารถทำการอัดคำสั่งได้ เรียกข้อมูล 32 บิตนี้ว่า "คำสั่งอัดที่ได้รับการอัดครึ่ง" (Half-packed instruction)

4.2 การดัดแปลงทางเดินข้อมูลของหน่วยประมวลผลเพื่อให้รองรับการทำงานแบบอัดคำสั่ง

เพื่อให้หน่วยประมวลผลสามารถทำงานกับโค้ดที่ได้รับการอัดคำสั่งได้ การดัดแปลงที่ต้องกระทำแบ่งเป็น 2 ส่วนคือ การดัดแปลงวงจรส่วนทางเดินข้อมูล และการดัดแปลงขั้นตอนการทำงาน ซึ่งใช้ในการสร้างวงจรหน่วยควบคุม (Control unit)

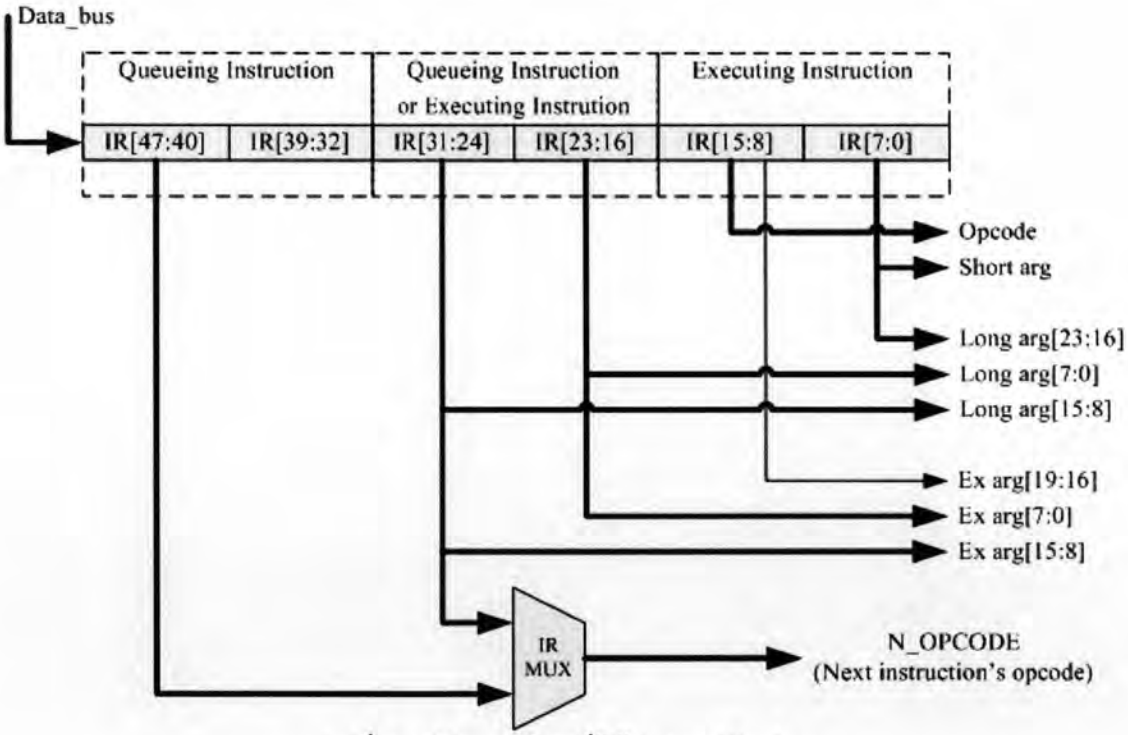
การดัดแปลงทางเดินข้อมูลนั้น กระทำกับรีจิสเตอร์ IR เพียงที่เดียว โดยรีจิสเตอร์ IR ตัวใหม่นี้เป็นรีจิสเตอร์ขนาด 48 บิตที่สามารถทำการเลื่อนขวา (Shift right) ได้ การทำงานคล้ายคิว (Queue) ที่ทำหน้าที่ลำเลียงคำสั่งที่อ่านมาจากหน่วยความจำ เข้ามากระทำในหน่วยประมวลผล โดยคำสั่งที่กำลังกระทำอยู่นั้น หากเป็นคำสั่งรูปแบบสั้นจะถูกเก็บอยู่ในตำแหน่งบิต 15 ถึงบิต 0 ของ

รีจิสเตอร์ สำหรับคำสั่งยาวจะถูกเก็บอยู่ในตำแหน่งบิต 31 ถึงบิต 0 ส่วน ส่วนที่เหลือของรีจิสเตอร์ถูกใช้ในการเก็บคำสั่งถัดไป ดังแสดงในรูปที่ 4.4

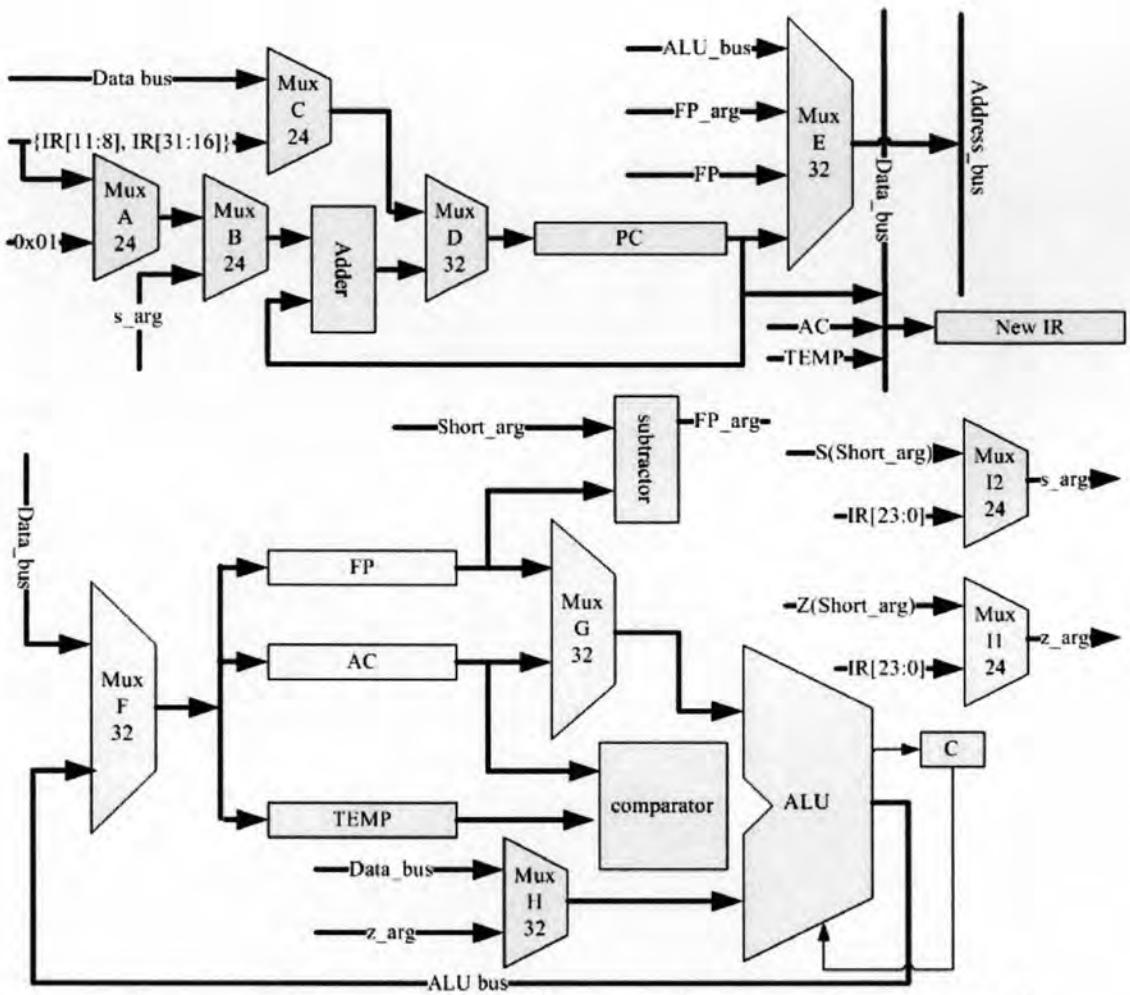
เหตุผลที่รีจิสเตอร์ IR ใหม่นี้มีขนาด 48 บิต เพื่อให้การกระทำคำสั่งรูปแบบยาวที่ถูกแบ่งเป็น 2 ส่วน สามารถกระทำได้อย่างมีประสิทธิภาพ เนื่องจากคำสั่งรูปแบบยาวนั้น อาจถูกแบ่งใส่ไว้ในหน่วยความจำคนละตำแหน่งกัน โดยในการทำงานเมื่อพบว่าส่วนบิตต่ำของคำสั่งรูปแบบยาว ถูกใส่ไว้ในตำแหน่งคำสั่งเติมเต็ม จะต้องทำการอ่านคำสั่งตำแหน่งถัดไป เพื่อให้ดึงเอาส่วนบิตสูงของคำสั่งรูปแบบยาวนั้นมาให้ครบเสียก่อน ซึ่งถ้าหากรีจิสเตอร์ IR มีขนาด 32 บิตเท่าเดิม จะทำให้การอ่านหน่วยความจำครั้งนี้ต้องทิ้งคำสั่งเติมเต็มออกไป เนื่องจากรีจิสเตอร์ IR นั้นเต็ม จากการเก็บคำสั่งรูปแบบยาวที่กำลังกระทำอยู่

ส่วนอื่นๆของทางเดินข้อมูลนั้น ยังคงเหมือนกับหน่วยประมวลผลเดิมที่นำเสนอในบทที่ 3 ทางเดินข้อมูลใหม่แสดงในรูปที่ 4.5

ข้อดีของการอัดคำสั่งด้วยวิธีนี้คือ การปรับปรุงหน่วยประมวลผลให้รองรับการอัดคำสั่งนั้นทำได้ง่ายและประหยัดทรัพยากร เห็นได้จากการดัดแปลงในส่วนของทางเดินข้อมูลนั้น เป็นการเปลี่ยนรีจิสเตอร์ IR เพียงตัวเดียว



รูปที่ 4.4 รีจิสเตอร์ IR ที่ได้รับการปรับปรุง



รูปที่ 4.5 ทางเดินข้อมูลของวงจรหน่วยประมวลผลที่รองรับการทำงานแบบอัดคำสั่ง

4.3 การทำงานโปรแกรมที่ได้รับการอัดคำสั่ง

จากเดิมที่การอ่านหน่วยความจำแต่ละครั้งจะได้คำสั่งเพียงคำสั่งเดียว แต่เมื่อมีการอัดคำสั่ง การอ่านหน่วยความจำแต่ละครั้ง อาจได้คำสั่งมามากกว่าหนึ่งคำสั่ง การทำงานโปรแกรมที่ได้รับการอัดคำสั่งจึงเปลี่ยนไป เพราะหากทำงานเหมือนเก่า เมื่อกระทำคำสั่งหนึ่งๆเสร็จ หน่วยประมวลผลจะทำการอ่านคำสั่งถัดไปทันที ทั้งที่คำสั่งถัดไปที่ถูกต้องนั้น คือ คำสั่งเติมเต็มที่อยู่ในรีจิสเตอร์ IR ในขณะนั้น

ดังที่ได้กล่าวไว้แล้ว รีจิสเตอร์ IR ตัวใหม่ที่สร้างมาเพื่อให้รองรับการทำงานกับโปรแกรมที่ได้รับการอัดคำสั่ง ทำงานคล้ายคิว (Queue) ที่ทำหน้าที่ลำเลียงคำสั่งเข้ามากระทำในหน่วยประมวลผล คำสั่งที่หน่วยประมวลผลกำลังกระทำอยู่คือ คำสั่งที่อยู่ในส่วนบิตต่ำ (Least significant bit) ของรีจิสเตอร์ IR ซึ่งอาจมีขนาด 16 บิตหรือ 32 บิต ขึ้นอยู่กับชนิดของคำสั่งที่กำลังกระทำอยู่

4.3.1 ขั้นตอนการทำงาน

ในการทำงานคำสั่งที่ได้รับการอัดคำสั่ง ระหว่างที่กำลังกระทำคำสั่งใดๆอยู่นั้น หน่วยประมวลผลจะต้องทำการถอดรหัสคำสั่งถัดไป ซึ่งเก็บอยู่ในตำแหน่งบิตสูง (Most significant bit) ของรีจิสเตอร์ IR ไปพร้อมๆกัน ซึ่งสัญญาณที่ส่งค่ารหัสดำเนินการ (Opcode) ของคำสั่งถัดไปให้วงจรหน่วยควบคุม (Control unit) ทำการถอดรหัสคือ สัญญาณ N_OPCODE ในรูปที่ 4.4 จะเห็นว่าสัญญาณนี้มีค่าเท่ากับบิต 31 ถึงบิต 24 หรือเป็นบิต 47 ถึงบิต 40 ของรีจิสเตอร์ IR ขึ้นกับคำสั่งที่กระทำอยู่ในขณะนั้นเป็นคำสั่งรูปแบบสั้นหรือคำสั่งรูปแบบยาว

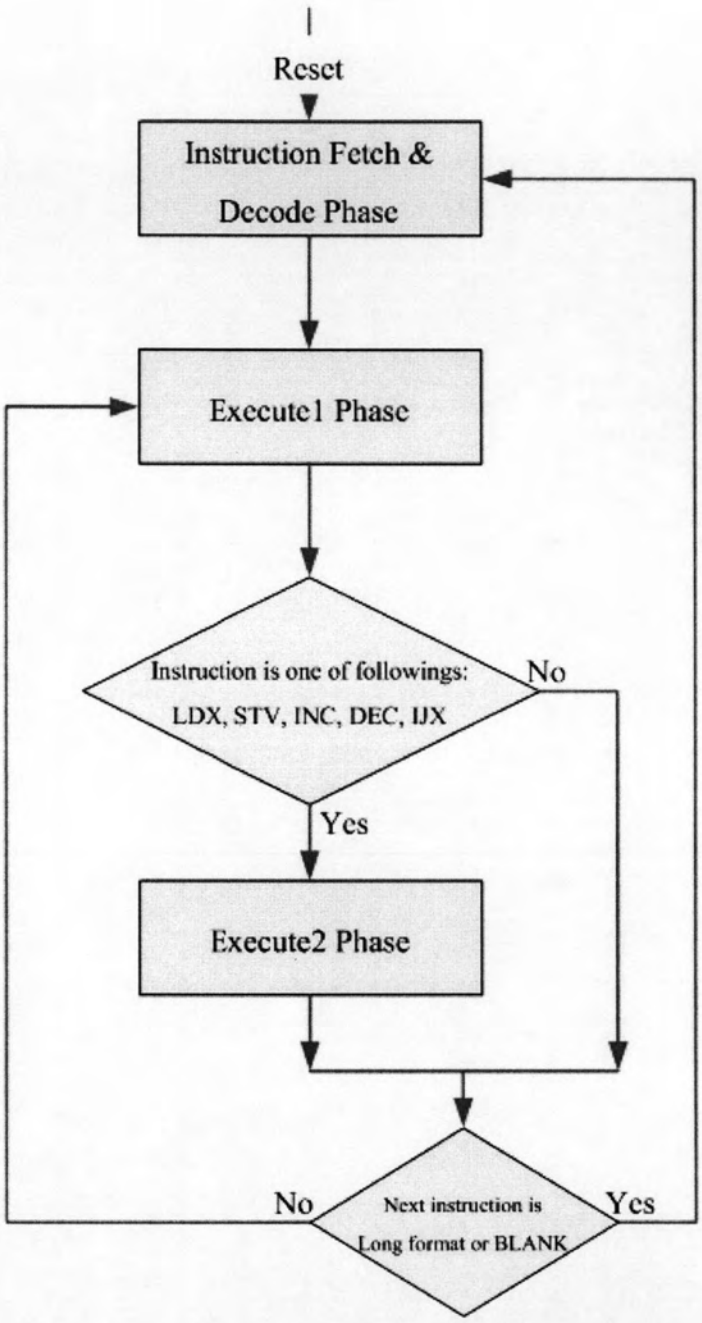
ขั้นตอนการทำงานโปรแกรมที่ได้รับการอัดคำสั่งนั้น ยังคงเหมือนขั้นตอนการทำงานของหน่วยประมวลผลเดิม คือ มี 3 ขั้นตอนได้แก่ ขั้นตอนการดึงคำสั่งและถอดรหัสคำสั่ง (Instruction fetch & decode phase) ขั้นตอนการกระทำที่ 1 (Execute1 phase) และขั้นตอนการกระทำที่ 2 (Execute2 phase) แต่การทำงานโปรแกรมที่ได้รับการอัดคำสั่งนั้น หน่วยประมวลผลจะทำงานในขั้นตอนการดึงคำสั่งและถอดรหัสคำสั่งน้อยกว่า เนื่องจากการอ่านคำสั่งแต่ละครั้งอาจได้คำสั่งมากกว่าหนึ่งคำสั่ง ดังแสดงในแผนภูมิขั้นตอนการทำงานในรูปที่ 4.6 เมื่อกระทำคำสั่งหนึ่งๆเสร็จหน่วยประมวลผลจะเข้าสู่ขั้นตอนการดึงคำสั่งและถอดรหัสคำสั่งก็ต่อเมื่อ รหัสดำเนินการของคำสั่งถัดไปนั้นบ่งบอกว่าคำสั่งถัดไปนั้นเป็น BLANK ซึ่งหมายความว่าคำสั่งในรีจิสเตอร์ IR ได้ถูกกระทำไปจนหมดแล้ว

อีกกรณีหนึ่งที่หน่วยประมวลผลจะเข้าสู่ขั้นตอนการดึงคำสั่งและถอดรหัสคำสั่งคือ เมื่อรหัสดำเนินการของคำสั่งถัดไปนั้นบ่งบอกว่าคำสั่งถัดไปนั้นเป็นคำสั่งรูปแบบยาว เนื่องจากรหัสดำเนินการ (Opcode) ของคำสั่งถัดไป เป็นรหัสดำเนินการของคำสั่งเติมเต็มเสมอ หากคำสั่งเติมเต็มเป็นคำสั่งรูปแบบยาว หมายความว่า อีกส่วนหนึ่งของคำสั่งรูปแบบยาวอยู่ในหน่วยความจำตำแหน่งถัดไป ซึ่งการที่จะกระทำคำสั่งนี้ได้จำเป็นต้องอ่านคำสั่งมาให้ครบเสียก่อน

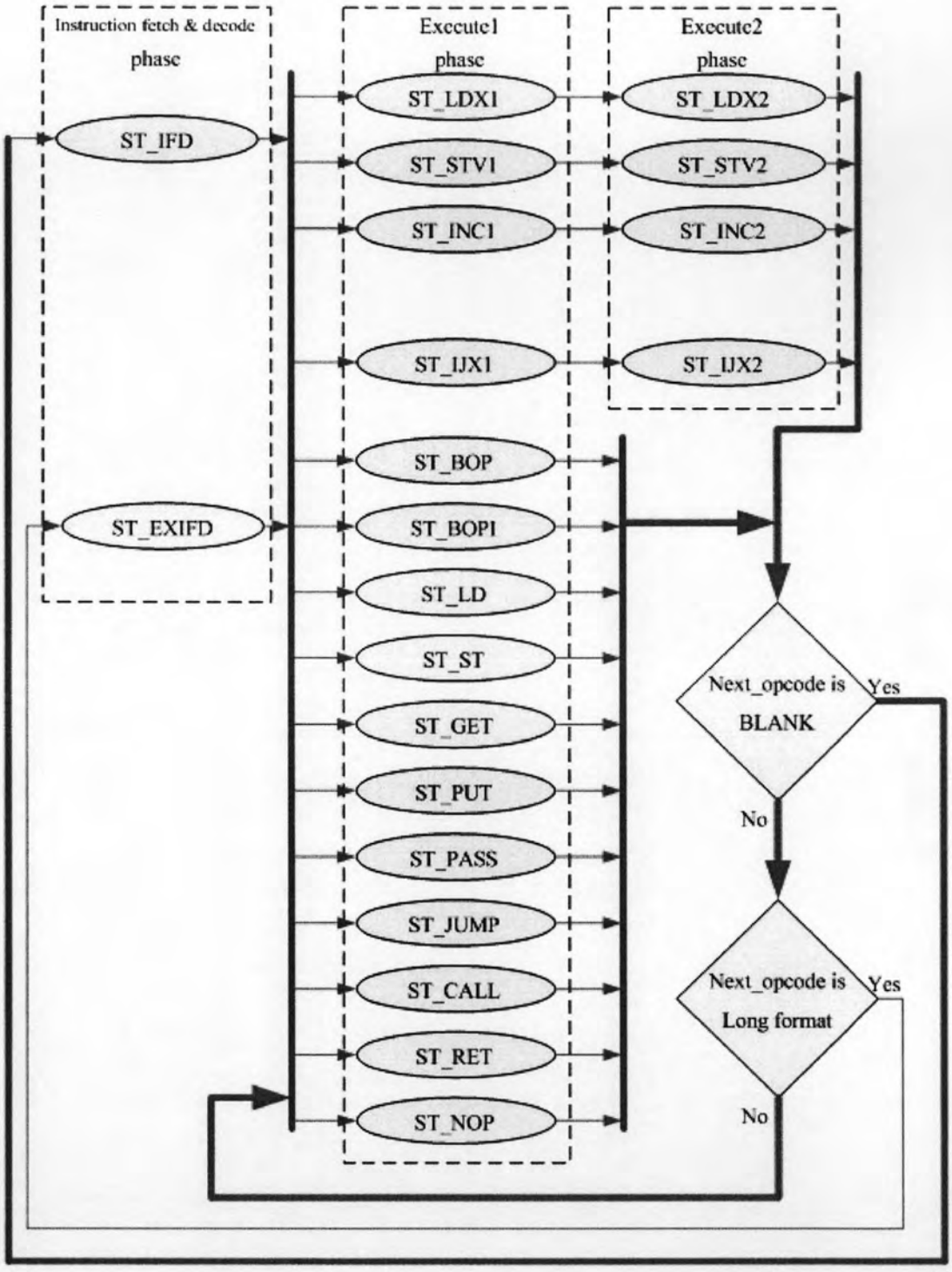
4.3.2 สถานะการทำงาน

มีเพียงสถานะดึงและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) ที่เพิ่มขึ้นมา สถานะการทำงานอื่นๆของหน่วยประมวลผลที่รองรับการอัดคำสั่งนี้ ใช้สถานะการทำงานของหน่วยประมวลผลเดิม การทำงานในแต่ละสถานะไม่ได้มีการเปลี่ยนแปลงแต่อย่างใด ดังแสดงในรูปที่ 4.7

สถานะดึงและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) เป็นสถานะที่ทำการอ่านคำสั่งเช่นเดียวกับสถานะดึงและถอดรหัสคำสั่ง (ST_IFD) แต่ทำในสถานการณ์ที่ต่างกัน สถานะดึงและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) ทำการอ่านคำสั่งในกรณีที่ยังมีคำสั่งตกค้างอยู่ในรีจิสเตอร์ IR การอ่านคำสั่งในสถานะนี้ คำสั่งที่อ่านได้จะถูกนำมาไว้ในรีจิสเตอร์ IR บิต 47 ถึงบิต 16



รูปที่ 4.6 แผนภูมิแสดงขั้นตอนการทำงานโค้ดที่ได้รับการอัดคำสั่ง



รูปที่ 4.7 แผนภูมิสถานะของหน่วยประมวลผลที่มีการอัดคำสั่ง

4.4 การดัดแปลงวงจรหน่วยควบคุมของหน่วยประมวลผลที่ให้รองรับการอัดคำสั่ง

หลังจากที่ได้กล่าวถึงการทำงานของหน่วยประมวลผล ที่ได้รับการปรับปรุงให้สามารถทำงานกับโปรแกรมที่ได้รับการอัดคำสั่งได้ ในส่วนนี้จะนำขั้นตอนการทำงานใหม่มาปรับปรุงวงจรหน่วยควบคุม (Control unit) เพื่อควบคุมให้หน่วยประมวลผลมีการทำงานตามที่ได้ออกแบบไว้

เนื่องจากการทำงานในแต่ละสถานะของหน่วยประมวลผลนั้นยังคงเหมือนเดิม ส่วนที่ต้องแก้ไขในวงจรหน่วยควบคุมจึงมีแค่การเพิ่มสถานะดึงและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) เข้าไป และปรับเลี่ยนสัญญาณที่ทำหน้าที่ควบคุมรีจิสเตอร์ IR

สำหรับสถานะดึงและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) นั้นมีการทำงานเหมือนสถานะดึงและถอดรหัสคำสั่ง (ST_IFD) ต่างกันเพียง 2 ประการคือ (1) คำสั่งที่อ่านได้ในสถานะดึงและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) ถูกใส่ในรีจิสเตอร์ IR บิต 47 ถึงบิต 16 และ (2) รหัสดำเนินการ (Opcode) ที่ใช้ในการถอดรหัสคำสั่งในสถานะนี้คือค่าของรีจิสเตอร์ IR บิต 15 ถึงบิต 8

ส่วนสัญญาณควบคุมรีจิสเตอร์ IR นั้น ในหน่วยประมวลผลเดิม มีหน้าที่เพียงควบคุมให้รีจิสเตอร์ทำการรับค่าจากภายนอกเข้ามาในสถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD) เท่านั้น แต่ในการทำงานโปรแกรมที่ได้รับการอัดคำสั่ง รีจิสเตอร์ IR ต้องมีการรับค่าในสถานะดึงคำสั่งและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) ดังแสดงในตารางที่ 4.2 สัญลักษณ์ที่ใช้อธิบายในตารางที่ 4.1

นอกจากนี้ สัญญาณควบคุมรีจิสเตอร์ IR ต้องทำหน้าที่ควบคุมการเลื่อนของรีจิสเตอร์ IR ด้วย เพื่อทำการเลื่อนคำสั่งที่กระทำเสร็จแล้วออกไปจากรีจิสเตอร์ และให้คำสั่งถัดไปเข้ามาอยู่ในตำแหน่งบิต 15 ถึงบิต 0 ของรีจิสเตอร์ IR ซึ่งเป็นตำแหน่งที่เก็บคำสั่งที่กำลังทำงานอยู่ โดยการเลื่อนข้อมูลในรีจิสเตอร์ IR มีพฤติกรรมดังแสดงในตารางที่ 4.3

ตารางที่ 4.1 สัญลักษณ์ที่ใช้ในการอธิบายพฤติกรรมการทำงานของรีจิสเตอร์ 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

ตารางที่ 4.2 การรับค่าเข้ารีจิสเตอร์ IR ในแต่ละสถานการณ์

สถานการณ์	ข้อมูลในรีจิสเตอร์ IR
สถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD)	IR ← { BLANK, Data_bus};
สถานะดึงคำสั่งและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD)	IR ← {Data_bus, IR[15:0]};

ตารางที่ 4.3 การเลื่อนข้อมูลในรีจิสเตอร์ IR ในแต่ละสถานการณ์

สถานการณ์	ข้อมูลในรีจิสเตอร์ IR
สถานะสุดท้ายของการกระทำคำสั่งรูปแบบสั้น	IR ← {BLANK, IR[47:16]};
สถานะสุดท้ายของการกระทำคำสั่งรูปแบบยาว	IR ← { BLANK, BLANK, IR[47:32]};

ตารางที่ 4.4 ได้แสดงการเปรียบเทียบขนาดของวงจรที่ต้องมีการปรับแต่งเพื่อให้รองรับการทำงานแบบอัดคำสั่ง โดยขนาดของวงจรในที่นี้พิจารณาจากจำนวนเกตสมมูล (Equivalent gate) ที่แสดงในผลการสังเคราะห์วงจรด้วยโปรแกรม Xilinx WebPack รุ่น 8.1i บนอุปกรณ์เอฟพีจีเอรุ่น Spartan 3 XC3S200 วงจรที่ได้รับการปรับแต่งประกอบด้วยรีจิสเตอร์ IR และวงจรหน่วยควบคุม (Control unit)

ตารางที่ 4.4 ทรัพยากรที่ใช้ในการปรับปรุงวงจรหน่วยประมวลผลเพื่อให้รองรับการอัดคำสั่ง

วงจร	จำนวนเกตสมมูล (Equivalent gate)		ขนาดใหญ่ขึ้น
	รองรับการอัดคำสั่ง	ไม่รองรับการอัดคำสั่ง	
รีจิสเตอร์ IR	1,049	259	305%
วงจรหน่วยควบคุม	1,885	1,837	2.61%
วงจรหน่วยประมวลผล	**13,060**	**12,677**	**3.02%**

จากตารางที่ 4.4 วงจรหน่วยประมวลผลหลังจากที่ได้ทำการปรับปรุงให้รองรับการอัดคำสั่ง พบว่าขนาดของหน่วยประมวลผล ซึ่งวัดจากจำนวนเกตสมมูลนั้นเปลี่ยนแปลงไปน้อยมาก โดยวงจรหน่วยประมวลผลที่ได้รับการปรับปรุงนั้นมีขนาดใหญ่ขึ้นเพียงร้อยละ 3 ของขนาดวงจรเดิมเท่านั้น จึงสามารถกล่าวได้ว่า วิธีอัดคำสั่งนี้ใช้ทรัพยากรน้อยตรงตามจุดประสงค์ที่ได้วางไว้ในเบื้องต้น

4.5 การถอดรหัสคำสั่งของหน่วยประมวลผลที่รองรับการอัดคำสั่ง

จากการทำงานเดิมที่ไม่มีการอัดคำสั่ง การอ่านหน่วยความจำหนึ่งครั้งจะมีการถอดรหัส (Decode) เพียงครั้งเดียว แต่เมื่อมีการอัดคำสั่ง จำนวนคำสั่งที่ได้จากการอ่านคำสั่งแต่ละครั้งอาจมี

มากกว่าหนึ่งคำสั่ง การถอดรหัสคำสั่งในหน่วยประมวลผลที่รองรับการอัดคำสั่งจึงเปลี่ยนไป การถอดรหัสคำสั่งในหน่วยประมวลผลที่รองรับการอัดคำสั่ง มีการทำ 3 รูปแบบตามสถานการณ์ในขณะนั้นดังนี้

4.5.1 การถอดรหัสคำสั่งในสถานะดึงคำสั่งและถอดรหัสคำสั่ง

การถอดรหัสคำสั่งในสถานะนี้ทำเหมือนการถอดรหัสคำสั่งในหน่วยประมวลผลเดิม โดยใช้ค่าจากสายสัญญาณ Data_bus บิต 15 ถึง 8 ซึ่งในขณะนั้นค่าดังกล่าวจะเป็นค่ารหัสดำเนินการ (Opcode) ของคำสั่งที่อ่านได้

4.5.2 การถอดรหัสคำสั่งถัดไปในระหว่างที่กระทำคำสั่งปัจจุบัน

ในหน่วยประมวลผลที่รองรับการอัดคำสั่ง ระหว่างที่กระทำคำสั่งหนึ่งๆอยู่ ต้องทำการถอดรหัสคำสั่งถัดไปไปพร้อมกัน ทั้งนี้เพื่อให้เมื่อกระทำคำสั่งปัจจุบันเสร็จ หน่วยประมวลผลจะเข้าสู่สถานะสำหรับกระทำคำสั่งถัดไปได้ทันที หรือหากคำสั่งถัดไปเป็นคำสั่งรูปแบบยาวหน่วยประมวลผลจะเข้าสู่สถานะดึงคำสั่งและถอดรหัสคำสั่งแบบพิเศษ (ST_EXIFD) เพื่ออ่านคำสั่งส่วนที่ยังขาดไปเข้ามาให้ครบเสียก่อน ในกรณีที่คำสั่งในรีจิสเตอร์ IR นั้นถูกกระทำไปจนหมดแล้ว หน่วยประมวลผลจะเข้าสู่สถานะดึงคำสั่งและถอดรหัสคำสั่ง (ST_IFD)

4.5.3 การถอดรหัสคำสั่งในสถานะดึงคำสั่งและถอดรหัสคำสั่งแบบพิเศษ

การถอดรหัสคำสั่งในสถานะนี้เป็นการถอดรหัสคำสั่งที่ตกค้างอยู่ในรีจิสเตอร์ IR ที่ยังไม่สามารถกระทำได้เนื่องจากคำสั่งที่อยู่ในรีจิสเตอร์ IR ยังไม่สมบูรณ์

4.6 ข้อจำกัดของการอัดคำสั่ง

ดังที่ได้กล่าวไว้ในช่วงแรกของบทนี้เกี่ยวกับสถานการณ์ที่บางสถานการณ์ ที่ไม่สามารถทำการอัดคำสั่งได้ ทำให้เกิดคำสั่งอัดที่ได้รับการอัดครึ่ง (Half-packed instruction) ในโปรแกรม เนื่องจากไม่สามารถนำคำสั่งถัดไปมาใส่ไว้ในตำแหน่งคำสั่งเติมเต็ม (Complement instruction) ได้

คำสั่งถัดไปจะไม่สามารถใส่ไว้ในตำแหน่งคำสั่งเติมเต็มได้ เมื่อมีลักษณะดังต่อไปนี้

4.6.1 คำสั่งถัดไปเป็นตำแหน่งปลายทางของการกระโดด

คำสั่งที่เป็นตำแหน่งปลายทางของการกระโดดหรือการเรียกโปรแกรมย่อย ไม่สามารถใส่ไว้ในตำแหน่งคำสั่งเติมเต็ม (Complement instruction) ได้ เนื่องจากโครงสร้างการทำงานของหน่วยประมวลผลนี้ไม่รองรับการกระโดดไปยังคำสั่งเติมเต็ม กล่าวคือ การที่จะกระทำคำสั่งเติมเต็มได้นั้นต้องกระทำคำสั่งฐานก่อนคำสั่งเติมเต็มทุกครั้ง ไม่สามารถกระทำคำสั่งเติมเต็มโดยไม่กระทำคำสั่งฐานได้ เพราะฉะนั้นคำสั่งที่เป็นปลายทางของการกระโดดจะต้องวางไว้ในตำแหน่งคำสั่งฐานเท่านั้น

4.6.2 คำสั่งถัดไปเป็นคำสั่งแรกของโปรแกรมย่อย

ในทำนองเดียวกันกับคำสั่งที่เป็นตำแหน่งปลายทางของการกระโดด คำสั่งที่เป็นคำสั่งแรกของโปรแกรมย่อยไม่สามารถอยู่ในตำแหน่งคำสั่งเติมเต็มได้ ในที่นี้คำสั่งที่เป็นคำสั่งแรกของโปรแกรมย่อยจะต้องถูกวางไว้ในตำแหน่งคำสั่งฐานเท่านั้น

4.6.3 คำสั่งถัดไปเป็นตำแหน่งปลายทางของการคืนค่าจากโปรแกรมย่อย

เนื่องจากหน่วยประมวลผลนี้ ไม่มีโครงสร้างสำหรับทำงานคำสั่งเติมเต็มโดยไม่กระทำคำสั่งฐานก่อน ดังนั้นคำสั่งแรกที่กระทำหลังจากคืนค่าจากโปรแกรมย่อยจึงใส่ไว้ในตำแหน่งคำสั่งเติมเต็มไม่ได้

ด้วยเหตุนี้ คำสั่ง CALL จึงเป็นคำสั่งเดียวที่ไม่อนุญาตให้ทำการอัดคำสั่งไม่ว่ากรณีใดๆ เนื่องจากแม้ว่าจะทำการอัดคำสั่ง CALL คำสั่งถัดจากคำสั่ง CALL ซึ่งเป็นคำสั่งแรกหลังจากคืนค่าจากโปรแกรมย่อยนั้น ต้องใส่ไว้ในตำแหน่งคำสั่งฐาน จึงไม่มีความจำเป็นใดๆ ที่จะต้องทำการอัดคำสั่งกับคำสั่ง CALL

4.7 วิเคราะห์ประสิทธิภาพการทำงานที่เพิ่มขึ้นจากคำสั่งอัด

โปรแกรมที่ได้รับการอัดคำสั่ง เมื่อจัดเรียงลงในหน่วยความจำจะใช้พื้นที่ในการเก็บน้อยกว่าโปรแกรมปกติ ซึ่งช่วยให้จำนวนครั้งที่ใช้ในการดึงคำสั่ง (Instruction fetch) นั้นลดลง ทำให้ทำงานโปรแกรมนั้นได้เร็วขึ้น ทั้งนี้ประสิทธิภาพที่เพิ่มขึ้นจากการอัดคำสั่งนั้น ขึ้นอยู่กับอัตราส่วนระหว่างคำสั่งรูปแบบสั้นและคำสั่งรูปแบบยาวในโปรแกรม หากโปรแกรมมีคำสั่งสั้นมากประสิทธิภาพก็จะเพิ่มขึ้นมากตามไปด้วย

อย่างไรก็ตามอีกปัจจัยหนึ่งที่มีผลกระทบอยู่ไม่น้อยก็คือโครงสร้างของโปรแกรม เนื่องจากการอัดคำสั่งที่ใช้นี้มีข้อจำกัด ทำให้มีช่องว่างเหลืออยู่ในโปรแกรมที่ได้ทำการอัดคำสั่งแล้ว ช่องว่างเหล่านี้ทำให้ประสิทธิภาพที่ได้จากการอัดคำสั่งไม่สูงเท่าที่ควร

4.8 แนวทางการปรับปรุงเพื่อกำจัดข้อจำกัดของการอัดคำสั่ง

เนื่องจากวิธีการอัดคำสั่งที่ได้นำเสนอไปแล้วนั้นมีข้อจำกัดหลายประการ ที่ทำให้ไม่สามารถทำการอัดคำสั่งได้อย่างเต็มที่ ทั้งนี้การที่จะกำจัดข้อจำกัดดังกล่าวออกไปได้นั้น สามารถทำได้โดยการกำหนดเลขที่อยู่ (Address) ให้กับคำสั่งที่อยู่ในตำแหน่งคำสั่งเติมเต็ม (Complement instruction) เพื่อให้หน่วยประมวลผลสามารถเข้าถึงคำสั่งในตำแหน่งดังกล่าวได้โดยตรง รายละเอียดของการปรับปรุงนี้จะกล่าวถึงในบทที่ 6