บทที่ 2
งานวิจัยที่เกี่ยวข้อง

บทนี้กล่าวถึงงานวิจัยที่เกี่ยวกับการลดขนาดของโปรแกรม (Code-size reduction) ซึ่งมุ่งหาวิธีลดขนาดโปรแกรมในระบบฝังตัวให้มีขนาดเล็กที่สุดเพื่อจุดประสงค์ 3 ประการ ได้แก่

1. ต้องการลดต้นทุนในการผลิตของระบบฝังใน เนื่องจากต้นทุนการผลิตระบบฝังตัวขึ้นกับขนาดของวงจร ซึ่งมีส่วนประกอบหลักอยู่ที่หน่วยความจำภายในระบบนั้นๆ หรืออาจจะกล่าวได้ว่าต้นทุนการผลิตกับแปรผันตรงกับขนาดของหน่วยความจำนั้นเอง
2. ต้องการลดขนาดของโปรแกรมที่ต้องการใช้งานผ่านระบบเครือข่าย (Network) หรืออินเตอร์เน็ต (Internet) เพื่อลดเวลาในการส่งโปรแกรมให้น้อยที่สุด
3. การลดขนาดของโปรแกรมยังช่วยให้เกิดการเข้าถึงหน่วยความจำ (Memory Access) น้อยลง ทำให้ใช้พลังงานไฟฟ้าน้อยลง (Low Power Consumption)

ในบทความของ Beszedes และคณะ [7] ได้กล่าวถึงประเภทของการลดขนาดโปรแกรมไว้แล้วว่ามีด้วยกัน 2 ประเภทได้แก่ "การอัดแน่นโปรแกรม" (Code Compaction) และ"การบีบอัดโปรแกรม" (Code Compression) ซึ่งในวิทยานิพนธ์นี้สนใจเพียงหลักการการบีบอัดโปรแกรมและจะนำเสนองานวิจัยที่เกี่ยวกับการบีบอัดโปรแกรมเท่านั้น

การวัดประสิทธิภาพของการบีบอัดโปรแกรมจะวัดได้จาก "อัตราการบีบอัด (Compression Ratio)" ดังสมการที่ (2)

$$\text{อัตราการบีบอัด} = \frac{\text{ขนาดของโปรแกรมที่ผ่านการบีบอัด}}{\text{ขนาดของโปรแกรมก่อนการบีบอัด}} \qquad (2)$$

นอกจากอัตราการบีบอัดแล้วสิ่งที่จำเป็นต้องคำนึงถึงก็คือเวลาที่เสียไปในการคลายโปรแกรม ซึ่งจะมีผลต่อสมรรถนะรวมของระบบ และขนาดของวงจรที่ใช้ในการคลายโปรแกรม

ผลงานโดยรวมของการวิจัยในด้านนี้สามารถอ่านได้ในบทความของ Beszedes และคณะ [7] และวิทยานิพนธ์ของ Lefurgy [11] ซึ่งรวบรวมบทความเกี่ยวกับการลดขนาดโปรแกรมไว้โดยละเอียด ซึ่งในวิทยานิพนธ์จะขอกล่าวถึงงานวิจัยที่เกี่ยวกับการบีบอัดโปรแกรมเท่านั้น โดยแบ่งออกเป็น 3 ประเภทหลักๆ ด้วยกันได้แก่ การบีบอัดโดยการดัดแปลงการออกแบบชุดคำสั่งแบบริสก์ (Reduce Instruction Set Computer, RISC) การบีบอัดโปรแกรมในระดับชุดคำสั่ง (Instruction level compression) และการแปลคำสั่ง (Interpreter)

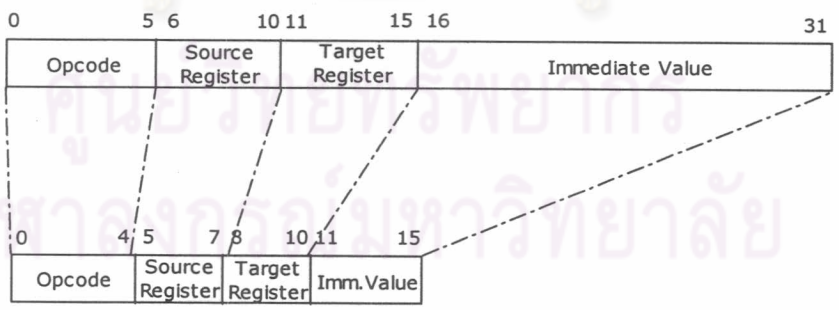
2.1 การดัดแปลงการออกแบบชุดคำสั่งของหน่วยประมวลผลแบบริสก์

ถึงแม้ว่าหน่วยประมวลผลแบบริสก์นั้นจะมีประสิทธิภาพในการทำงานสูง เนื่องจากมีการนำเอาการประมวลผลแบบสายท่อ (Pipeline) มาใช้ ซึ่งต้องบังคับให้ทุกคำสั่งในชุดคำสั่งมีขนาดเท่ากันทั้งหมด ทำให้บางคำสั่งต้องมีขนาดใหญ่เท่ากับคำสั่งอื่นๆ โดยไม่จำเป็น ทำให้ขนาดของโปรแกรมสำหรับหน่วยประมวลผลแบบนี้มีขนาดใหญ่

ดังนั้นจึงมีแนวความคิดที่จะดัดแปลงหน่วยประมวลผลให้สามารถทำงานได้กับชุดคำสั่งพิเศษที่มีการออกแบบให้มีขนาดที่เล็กลง โดยตัดความสามารถบางอย่างออกไปเช่น การตัดคำสั่งที่ไม่จำเป็นออก การลดจำนวนเรจิสเตอร์ที่อ้างถึงได้ในคำสั่ง หรือการลดขนาดของตัวถูกดำเนินการแบบทันที (Immediate operand)

ตัวอย่างของงานวิจัยด้านนี้ ได้แก่ Thumb [12] และ MIPS16 [13] โดยที่งานวิจัยทั้งสองชุดคำสั่งของหน่วยประมวลผล ARM [14] และ MIPS [15] ที่เดิมมีสถาปัตยกรรมชุดคำสั่งขนาด 32 บิตมาปรับปรุงให้เป็นชุดคำสั่งใหม่ที่มีขนาดลดลงเหลือ 16 บิต โดยออกแบบให้หน่วยประมวลกลางสามารถทำงานกับชุดคำสั่งเดิมขนาด 32 บิตและชุดคำสั่งใหม่ขนาด 16 บิตได้ทั้งสองแบบ

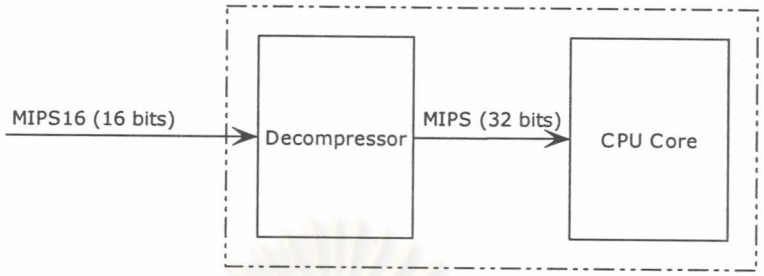
ในงานวิจัย MIPS16 [13] นำเสนอการออกแบบชุดคำสั่งใหม่ขนาด 16 บิตให้กับหน่วยประมวลผล MIPS ซึ่งเดิมคำสั่งมีขนาด 32 บิตทุกคำสั่ง โดยใช้การจำลองการทำงานกับโปรแกรมทดสอบแล้วเก็บสถิติต่างๆ เพื่อพิจารณาว่าคำสั่งใดที่มีการใช้น้อย การเข้าถึงเรจิสเตอร์โดยเฉลี่ยอยู่ที่เท่าไร และขนาดตัวถูกดำเนินการแบบทันทีมีขนาดเฉลี่ยเท่าใด ทำให้พิจารณาได้ว่าควรจะตัดคำสั่งไหน หรือตัวถูกดำเนินการส่วนไหนบ้าง โดยรูปแบบของคำสั่งที่ถูกออกแบบขึ้นใหม่แสดงได้ดังรูปที่ 2.1



รูปที่ 2.1 รูปแสดงโครงสร้างของคำสั่งใน MIPS16

จากรูปที่ 2.1 พบว่าเขตรหัสดำเนินการ (Operand field) ถูกลดขนาดจาก 6 บิตเหลือ 5 บิต แสดงว่าคำสั่งในชุดคำสั่ง MIPS16 สามารถมีคำสั่งได้มากที่สุดเพียง 32 คำสั่ง จากเดิม 64 คำสั่ง การอ้างถึงเรจิสเตอร์ภายในก็สามารถอ้างได้เพียง 8 ตัว และการใช้ค่าตัวถูกดำเนินการแบบทันทีก็ใช้ได้แค่เพียง 5 บิตเท่านั้น

ในงานวิจัยทั้งสองไม่มีการปรับแต่งการทำงานหลักของแกนหน่วยประมวลผล (CPU Core) มีเพียงการปรับปรุงโดยเพิ่มฮาร์ดแวร์พิเศษเข้าไปเพื่อทำหน้าที่แปลงคำสั่งขนาด 16 บิตให้เป็นคำสั่งเดิมขนาด 32 บิต ก่อนจะส่งให้หน่วยประมวลผลทำงานกับคำสั่งนั้น ดังรูปที่ 2.2



รูปที่ 2.2 กลไกการทำงานของหน่วยประมวลผลที่มีการดัดแปลงขนาดชุดคำสั่ง

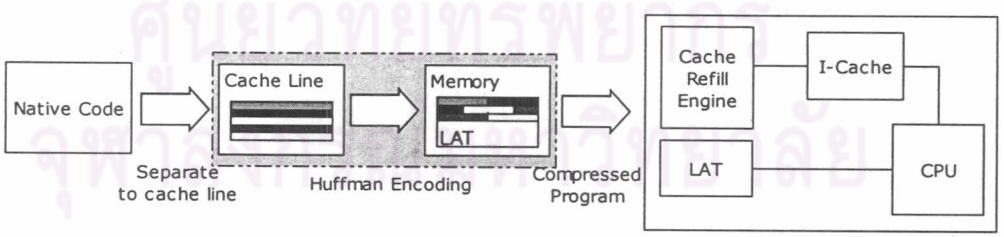
ผลการทดลองที่รายงานไว้ในงานวิจัยทั้งสอง พบว่าอัตราการบีบอัดของโปรแกรมของ MIPS16 และ Thumb อยู่ที่ 0.60 และ 0.70 ตามลำดับ

2.2 การบีบอัดโปรแกรมในระดับชุดคำสั่ง

การบีบอัดโปรแกรมในระดับชุดคำสั่งนั้นเป็นการนำเอาการเข้ารหัสข้อมูล (Data encoding) ต่างๆ มาประยุกต์กับการบีบอัดโปรแกรม ซึ่งในที่นี้จะขอกล่าวถึงงานวิจัย 3 ชิ้นดังนี้

2.2.1 การบีบอัดโปรแกรมสำหรับหน่วยประมวลผลแบบริสก์

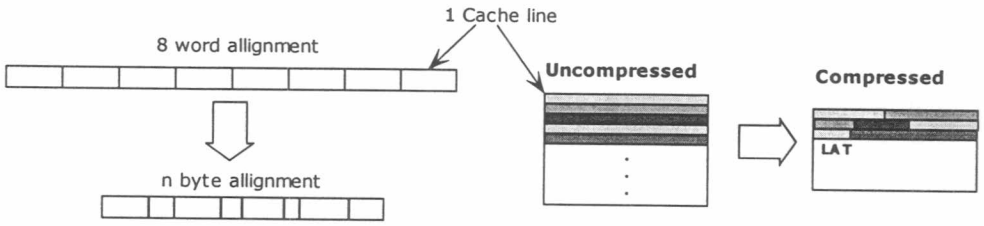
Kozuch และ Wolf [16] เสนอแนวคิดในการบีบอัดโปรแกรมที่มีชื่อว่า "การบีบอัดโปรแกรมสำหรับหน่วยประมวลผลแบบริสก์" (Code Compression RISC Processor, CCRP) ซึ่งใช้หลักการทำงานของหน่วยความจำแคช คำสั่งที่ถูกทำงานต่อไปในหน่วยประมวลผลจะถูกนำมาเก็บในหน่วยความจำแคชก่อนเสมอ และมีการคลายโปรแกรมที่บีบอัดก่อนการทำงาน



รูปที่ 2.3 กลไกการบีบอัดและทำงานของ CCRP

กลไกการบีบอัดและการทำงานแสดงไว้ดังรูปที่ 2.3 การบีบอัดจะเริ่มจากการแบ่งโปรแกรมทั้งหมดออกเป็นส่วนๆ ตามการถูกนำไปเก็บในหน่วยความจำแคช ที่เรียกว่าแคชไลน์ (Cache line) ซึ่งมีขนาด 16 คำสั่งต่อหนึ่งแคชไลน์ หลังจากนั้นเข้ารหัสโปรแกรมทีละส่วนโดยใช้

การเข้ารหัสของฮัฟฟ์แมน (Huffman encoding) ดังแสดงไว้ในรูปที่ 2.4 เมื่อเข้ารหัสจนครบทุกแคชไลน์แล้วจะได้โปรแกรมที่ผ่านการบีบอัดพร้อมนำไปใช้งาน



รูปที่ 2.4 การเข้ารหัสกับโปรแกรมทีละแคชไลน์

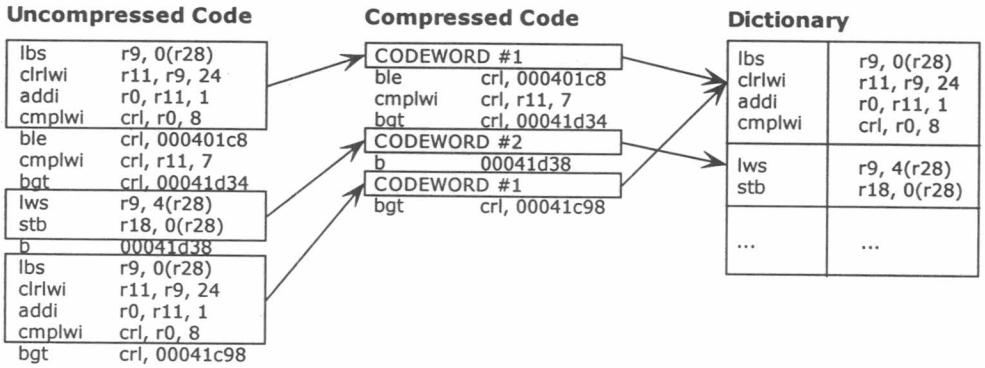
ในระบบปกติเวลาที่หน่วยประมวลผลทำงานจะเข้าไปอ่านคำสั่งจากหน่วยความจำแคช ถ้าคำสั่งดังกล่าวไม่ได้อยู่ในหน่วยความจำแคช หรือที่เรียกว่าเกิดแคชมิส (Cache miss) ตัวจัดการหน่วยความจำแคช (Cache refill engine) จะนำแคชไลน์ที่บรรจุคำสั่งที่หน่วยประมวลผลต้องการมาใส่ในแคชเพื่อให้หน่วยประมวลผลสามารถเข้าถึงคำสั่งได้ แต่ในระบบที่ทำงานกับโปรแกรมที่ถูกบีบอัดด้วยวิธีการนี้จำเป็นต้องพัฒนาตัวจัดการแคชให้สามารถคลายโปรแกรมทีละแคชไลน์ได้ โดยทุกครั้งที่เกิดแคชมิสตัวจัดการหน่วยความจำแคชต้องคลายโปรแกรมที่ถูกบีบอัดก่อนแล้วจึงค่อยใส่เข้าไปในหน่วยความจำแคช

ปัญหาในงานวิจัยนี้คือเลขที่อยู่ (Address) ของโปรแกรมหลังการเข้ารหัสจะไม่เหมือนกับโปรแกรมก่อนการเข้ารหัส ดังแสดงไว้ในรูปที่ 2.3 และรูปที่ 2.4 เห็นได้ว่าโปรแกรมในแคชไลน์มีขนาดลดลง ทำให้เลขที่อยู่ในโปรแกรมเปลี่ยนไป จึงใช้ตารางเก็บค่าเลขที่อยู่ (Line address table, LAT) ของแต่ละแคชไลน์ ชี้ว่าแต่ละแคชไลน์ที่ได้รับการเข้ารหัสแล้วอยู่ที่ใดในหน่วยความจำ

การทดลองทำบนชุดคำสั่งของหน่วยประมวลผล MIPS [15] ได้อัตราส่วนการบีบอัด 0.73

2.2.2 วิธีจัดเก็บในพจนานุกรม

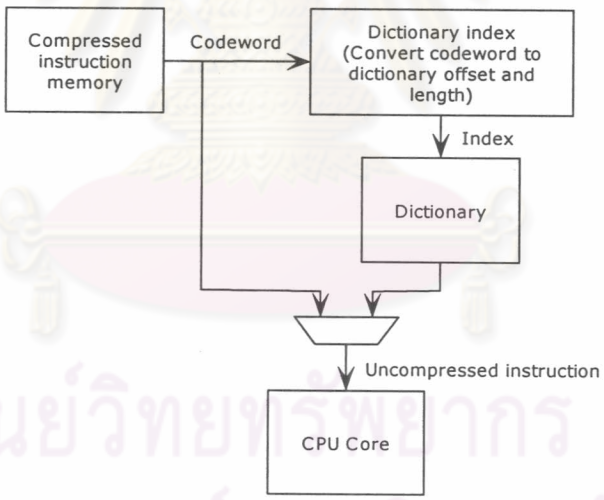
งานวิจัย CCRP เป็นการเข้ารหัสชุดคำสั่งในระดับบิต แต่ในงานวิจัยของ Lefurgy และคณะ [17] ตั้งข้อสังเกตถึงการแปลของโปรแกรมแปลภาษาว่าให้ลำดับของคำสั่งที่เหมือนๆ กันออกมา จึงนำเสนอการแทนลำดับของคำสั่ง (Instruction sequence) ที่เหมือนกันเหล่านั้นด้วยรหัสคำ (Codeword) ที่มีจำนวนบิตน้อยกว่าลำดับของคำสั่งที่นำรหัสเข้าไปแทนที่ และนำเอาลำดับของคำสั่งที่ซ้ำกันเหล่านั้นไปเก็บไว้ในพจนานุกรม ดังแสดงในรูปที่ 2.5



รูปที่ 2.5 รูปแสดงการแทนที่ของรหัสในโปรแกรม

ขั้นตอนการบีบอัดเริ่มจากการหาลำดับของคำสั่งที่ซ้ำกันด้วยอัลกอริทึมเชิงละโมบ (Greedy algorithm) สร้างเป็นพจนานุกรม หลังจากนั้นทำการเข้ารหัสรหัสคำ (Codeword encoding) และนำเอารหัสคำเหล่านั้นไปแทนที่ลำดับของคำสั่งภายในโปรแกรม

การถอดรหัสชุดคำสั่งมีโครงสร้างดังรูปที่ 2.6 โดยมีวงจรที่ทำหน้าที่นำเอาลำดับของคำสั่งภายในพจนานุกรมมาแทนที่รหัสคำที่พบในโปรแกรม ก่อนที่คำสั่งเหล่าจะถูกนำไปประมวลผลบนหน่วยประมวลผล



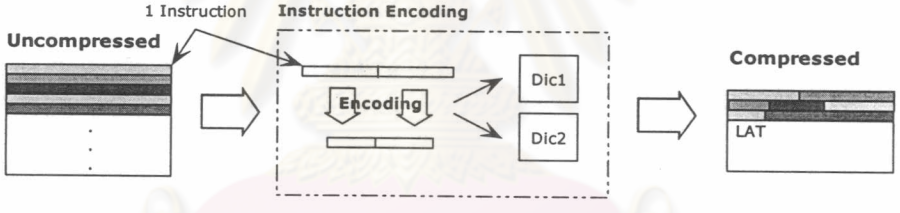
รูปที่ 2.6 โครงสร้างของการถอดรหัส

งานวิจัยนี้ทดลองบีบอัดโปรแกรมบนหน่วยประมวลผล 3 แบบด้วยกันคือ PowerPC [18], ARM [14] และ i386 [19] โดยได้ผลการทดลองได้ทำการวัดอัตราส่วนการบีบอัดของทั้ง 3 หน่วยประมวลได้ค่าเฉลี่ยเท่ากับ 0.61, 0.66 และ 0.74 ตามลำดับ นอกจากนั้นได้ทำการทดลองวัดค่าอัตราส่วนการบีบอัดที่ขนาดของพจนานุกรมและขนาดรหัสคำต่างๆ กันอีกด้วย

2.2.3 IBM CodePack

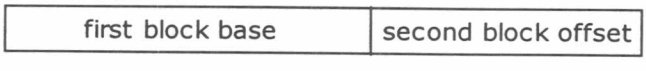
บริษัทไอบีเอ็ม [20] ได้คิดวิธีที่เรียกว่า "CodePack" เพื่อนำมาบีบอัดโปรแกรมที่ใช้บนสถาปัตยกรรมที่ใช้หน่วยประมวลผล PowerPC [18] โดยหลักการที่ไอบีเอ็มคิดขึ้นนี้ได้ประยุกต์ใช้การบีบอัดโปรแกรมโดยการเข้ารหัสโดยใช้พจนานุกรมของ Lefurgy และคณะ [17] และหลักการการคลายโปรแกรมของ Kozuch และ Wolfe [16] ที่ใช้การคลายโปรแกรมบนหน่วยความจำแคช

หลักการในการบีบอัดโปรแกรมของ CodePack นั้นจะเริ่มจากการสร้างพจนานุกรมสำหรับโปรแกรม และนำเอารหัสคำเข้าไปแทนที่ในโปรแกรมเหมือนกับแนวคิดของ Lefurgy และคณะแต่จะแตกต่างกันที่ CodePack จะเข้ารหัสทีละคำสั่ง ไม่ได้เข้ารหัสทีละส่วนเหมือนในแนวคิดของ Kozuch นอกจากนั้นยังแบ่งการเข้ารหัสคำสั่งออกเป็นสองส่วนและเข้ารหัสทีละส่วน เนื่องจากเมื่อลองพิจารณาดูแล้วจะพบว่าคำสั่งของ PowerPC ในส่วน 16 บิตแรกส่วนมากจะประกอบไปด้วยรหัสดำเนินการ (Opcode) และตัวถูกดำเนินการแบบเรจิสเตอร์ (Register operand) ส่วน 16 บิตถัดมาจะเป็นตัวถูกดำเนินการแบบทันที (Immediate operand) ซึ่งจะเห็นได้ว่าถ้าเข้ารหัสแยกสองส่วนออกจากกันน่าจะได้พจนานุกรมและรหัสคำที่มีขนาดเล็ก กว่าการเข้ารหัสทั้งคำสั่ง กลไกในการบีบอัดคำสั่งจะเป็นไปดังรูปที่ 2.7



รูปที่ 2.7 การบีบอัดคำสั่งของวิธี CodePack

การคลายโปรแกรมทำบนหน่วยความจำแคช และใช้ตารางเก็บค่าตัวชี้แคช (Cache Index Table) ซึ่งทำหน้าที่เก็บค่าเลขที่อยู่จริงของโปรแกรมที่ได้รับการบีบอัดแล้วภายในหน่วยความจำ คล้ายกับตารางเก็บค่าเลขที่อยู่ (LAT) ในงานวิจัยของ Lefurgy และคณะ แต่ต่างกันที่ในงานวิจัย CodePack จะนำเสนอวิธีการลดขนาดของตัวชี้แคชที่ชี้ไปยังตารางเก็บค่าเลขที่อยู่โดยจะให้ 1 ตัวชี้ชี้ไปยังแคชไลน์ 2 อัน โดยมีโครงสร้างของตัวชี้เป็นดังรูปที่ 2.8 ซึ่งจะเห็นได้ว่าส่วนแรกของตัวชี้จะชี้ไปยังแคชไลน์โดยตรง แต่ส่วนที่สองของตัวชี้จะเก็บค่าซึ่งอ้างอิงเทียบกับตัวชี้ในส่วนแรกเพื่อชี้ไปยังแคชไลน์ที่อยู่ติดกับแคชไลน์อันแรก



รูปที่ 2.8 โครงสร้างของตัวชี้แคช

กลไกในการคลายโปรแกรมเริ่มจากเมื่อหน่วยประมวลต้องการคำสั่งที่ไม่ได้ถูกบรรจุอยู่ในหน่วยความจำแคช (Cache miss) หน่วยประมวลผลจะส่งเลขที่อยู่ของแคชไลน์ที่ต้องการออกมา ดังในรูปที่ 2.9 -1 เนื่องจากเลขที่อยู่ที่หน่วยประมวลผลส่งมานี้เป็นเลขที่อยู่ของแคชไลน์ก่อนการบีบอัด ดังนั้นระบบจะนำเอาเลขที่อยู่ที่หน่วยประมวลผลส่งออกมาไปเทียบกับตารางตัวชี้แคช เพื่อหาเลขที่อยู่ของโปรแกรมที่ได้รับการบีบอัดแล้วในหน่วยความจำดังรูปที่ 2.9 -2

เมื่อได้แคชไลน์ที่ต้องการแล้ว ทำการคลายโปรแกรมที่ถูกบีบอัดทีละคำสั่งโดยการถอดรหัสคำในพจนานุกรม ดังรูปที่ 2.9-3 โปรแกรมที่ถูกคลายแล้วจะถูกนำไปใส่ในหน่วยความจำแคชเพื่อรอให้หน่วยประมวลผลนำไปประมวลผลต่อไป

บทความของ Kissell [20] รายงานผลการทดลองการวัดค่าอัตราการบีบอัดของโปรแกรมที่บีบอัดด้วยวิธีการนี้ได้ค่าเฉลี่ย 0.60 นอกจากนั้นในงานวิจัยของ Lefurgy และคณะ [21] ได้ทดลองวัดประสิทธิภาพของระบบที่ใช้ CodePack พบว่าระบบจะทำงานช้าลง 14% - 18% และยังมีผลการทดสอบเกี่ยวกับประสิทธิภาพด้านอื่นๆ อีกด้วย

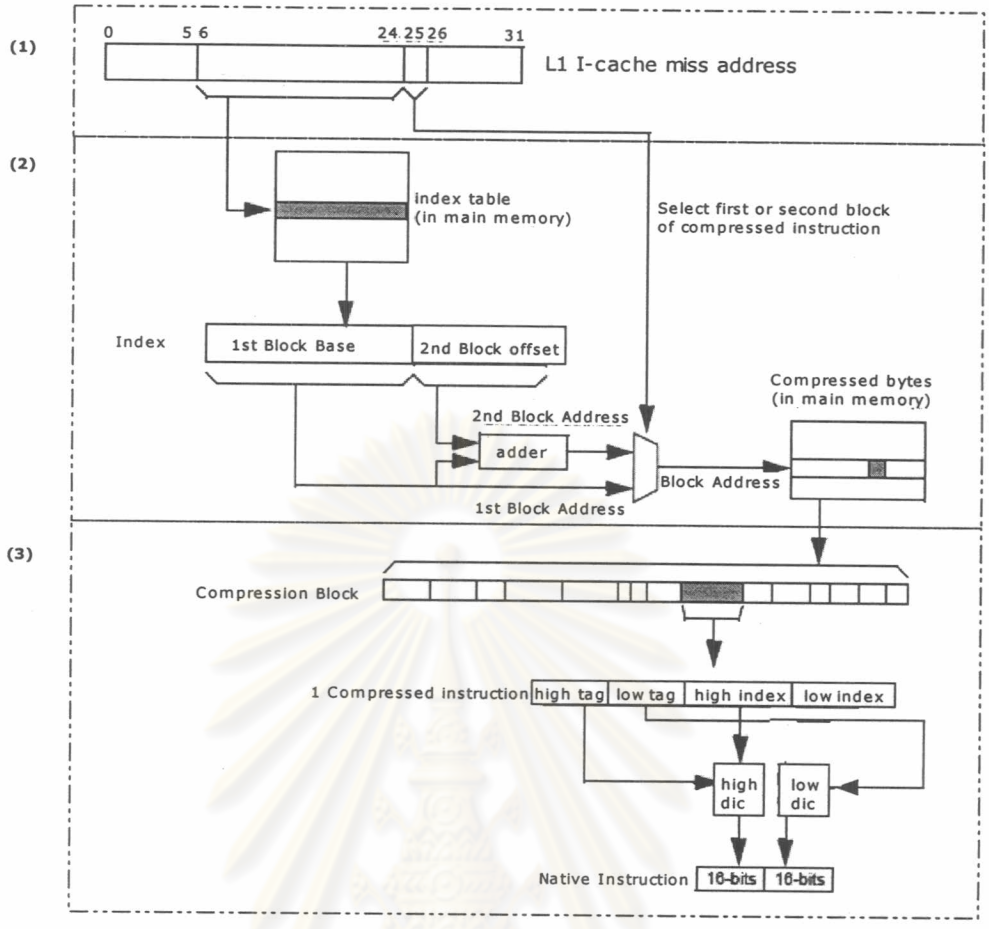
2.3 การแปลคำสั่ง

แนวคิดที่ใช้ในการบีบอัดข้อมูลที่น่าสนใจอีกวิธีหนึ่งคือการทำให้โปรแกรมฝังตัวถูกจัดเก็บอยู่ในรูปแบบของชุดคำสั่งกลาง (Intermediate code) ที่มีขนาดเล็กกว่าโปรแกรมที่อยู่ในรูปแบบของชุดคำสั่งภาษาเครื่อง (Machine code) โดยการใช้โปรแกรมแปลคำสั่งชุดคำสั่งกลางก่อนการทำงาน ในที่นี้จะขอนำเสนองานวิจัยสองชิ้นที่เกี่ยวกับใช้ชุดคำสั่งกลางกับการบีบอัดโปรแกรม

2.3.1 BRISC

Ernst และ Hauptstabe [9] นำเสนอการบีบอัดโปรแกรมที่น่าสนใจไว้สองแบบสำหรับการทำงานต่างสถานการณ์กัน แต่ในที่นี้จะขอกล่าวถึงเพียงแค่การบีบอัดโปรแกรมที่สามารถทำงานได้ทันทีบนโปรแกรมแปลคำสั่ง

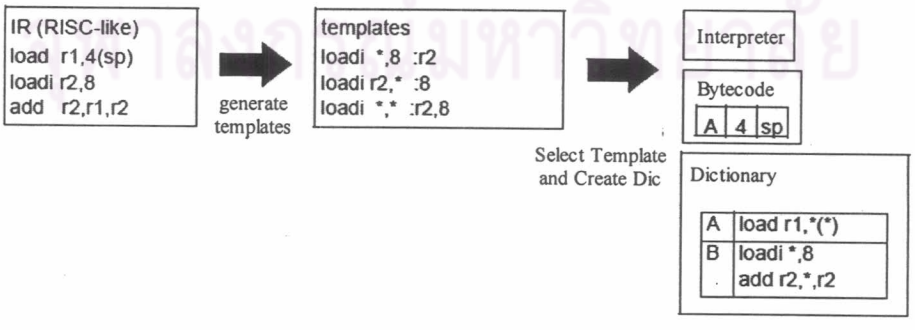
งานวิจัยนี้เน้นการสร้างโปรแกรมที่อยู่ในรูปแบบของชุดคำสั่งกลางที่มีขนาดเล็กมาก โดยในงานวิจัยให้ชื่อว่า BRISC โดยใช้หลักการ 2 ข้อได้แก่ “Operand specialization” และ “Opcode combination”



รูปที่ 2.9 แสดงการคลายโปรแกรมของ CodePack

หลักการการบีบอัดคือการค้นหารูปแบบของโปรแกรมที่ผ่านการจัดรูปตัวถูกดำเนินการแล้ว และสร้างเป็นแผ่นแบบ (Template) ดังแสดงไว้ในรูปที่ 2.10

ผลการทดลองพบว่าอัตราการบีบอัดของวิธีนี้อยู่ที่ 0.53 ถึง 0.69 แต่การทำงานของโปรแกรมแปลคำสั่งจะทำให้การทำงานช้าลงไป 9.6 – 15.4 เท่าของการทำงานกับโปรแกรมที่ไม่ได้บีบอัด



รูปที่ 2.10 การบีบอัดของ BRISC

2.3.2 Pipeline Interpreter

ในงานวิจัยของ Hoogerbrugge และคณะ [10] เป็นอีกงานวิจัยที่ใช้โปรแกรมแปลคำสั่งมีจุดประสงค์ที่ต้องการลดขนาดของโปรแกรม โดยที่ยังคงประสิทธิภาพการทำงานให้ไม่ช้าลงมากนัก

หลักการลดขนาดของโปรแกรมนั้นคือใช้ตัวแปลโปรแกรม (Compiler) สร้างโปรแกรมที่แบ่งออกเป็น 2 ส่วนได้แก่โปรแกรมส่วนวิกฤต (Critical code) และโปรแกรมส่วนไม่วิกฤต (Non-critical code) ส่วนที่เป็นโปรแกรมส่วนวิกฤตอยู่ในรูปภาษาเครื่องโดยตรง แต่ในส่วนที่เป็นโปรแกรมส่วนไม่วิกฤตอยู่ในรูปของชุดคำสั่งกลางที่มีขนาดเล็ก

โปรแกรมแปลคำสั่งใช้หลักการการทำงานแบบสายโยงใย (Thread) และการทำงานแบบสายท่อ (Pipeline interpreter) ซึ่งช่วยทำให้ทำงานได้เร็วมากกว่าโปรแกรมแปลคำสั่งแบบปกติ

ผลการทดลองพบว่าอัตราการบีบอัดอยู่ที่ 0.70 และการทำงานของโปรแกรมแปลคำสั่งทำให้การทำงานช้าลงประมาณ 8.59 เท่าของการทำงานปกติ

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย