บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

เนื่องจากการอัดคำสั่ง (Instruction packing) ที่ทำในงานวิจัยนี้มีขั้นตอนวิธีคล้ายคลึงกับการลดขนาดโปรแกรม (Code size reduction) ในบทนี้จะกล่าวถึงทฤษฎีพื้นฐานเกี่ยวกับการลดขนาดโปรแกรม และข้อดีข้อเสียของวิธีการลดขนาดโปรแกรมวิธีต่างๆ เพื่อเป็นพื้นฐานในการทำความเข้าใจเนื้อหาที่นำเสนอในบทถัดๆไป พร้อมทั้งนำเสนองานวิจัยที่เกี่ยวข้องกับการลดขนาดโปรแกรม และการนำวิธีลดขนาดโปรแกรมมาประยุกต์ใช้จริง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 ต้นทุนการผลิตชิปของระบบฝังตัว

ต้นทุนการผลิตชิปในระบบฝังตัวนั้นแปรผันตรงกับจำนวนทรานซิสเตอร์ในวงจร จากงานวิจัยของ มาลี่ (Maly) [5] ได้เสนอแบบจำลองคำนวณต้นทุนการผลิตชิปในอุตสาหกรรมระบบฝังตัวดังสมการที่ 1 ดังนั้นการสร้างชิปให้มีขนาดเล็กที่สุดจึงเป็นเป้าหมายหลักของการออกแบบชิปในระบบฝังตัวมาโดยตลอด

$$C_{tr} = \frac{C_w}{N_{ch} N_{tr} Y} \qquad (1)$$

โดย C_{tr} คือ ต้นทุนของทรานซิสเตอร์

C_w คือ ต้นทุนการผลิตแว่นผลึก

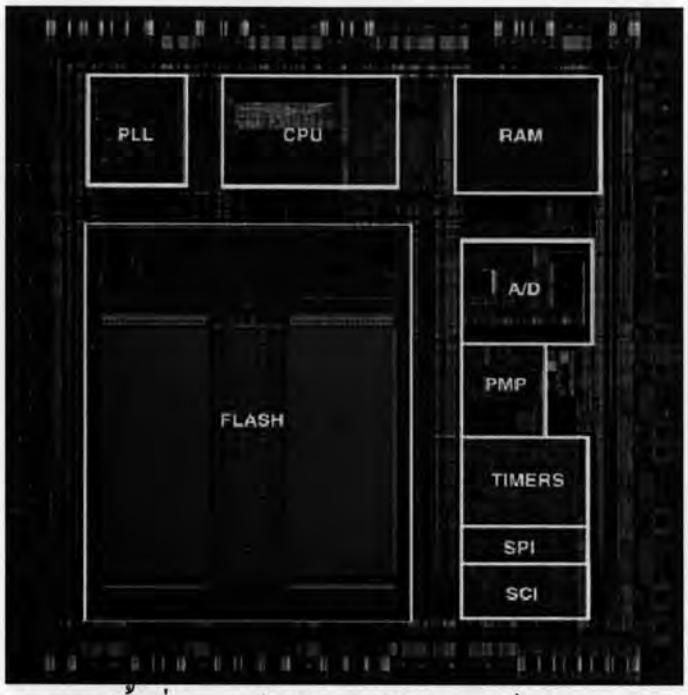
N_{ch} คือ จำนวนดายของวงจรรวมต่อหนึ่งแว่นผลึก

N_{tr} คือ จำนวนทรานซิสเตอร์ต่อหนึ่งดาย

Y คือ จุดคราก (Yield) การผลิต

ในวงจรหน่วยประมวลผลพบว่า จำนวนทรานซิสเตอร์ส่วนใหญ่ถูกใช้ไปในส่วนของหน่วยความจำโปรแกรม (Program Memory) เห็นได้จากขนาดพื้นที่ดายของไมโครคอนโทรลเลอร์ MC68HC908GP20 [6] ของบริษัท Motorola ในรูปที่ 2.1 ที่พื้นที่ส่วนใหญ่เป็นพื้นที่ของหน่วยความจำแฟลช (Flash Memory) สำหรับเก็บโปรแกรม

การลดขนาดของโปรแกรมจึงมีผลทางอ้อมในการช่วยลดพื้นที่ที่ใช้ในการเก็บโปรแกรม และช่วยลดต้นทุนในการผลิตชิปได้ จึงได้มีงานวิจัยมากมายที่เกี่ยวกับการลดขนาดของโปรแกรม (Code Size Reduction) ในระบบฝังตัว



รูปที่ 2.1 แสดงพื้นที่ดายของไมโครคอนโทรลเลอร์ MC68HC908GP20

2.1.2 การลดขนาดโปรแกรมในระบบฝังตัว

มีวิธีการมากมายในการลดขนาดโปรแกรมในระบบฝังตัว ซึ่งแต่ละวิธีมีข้อดีข้อเสียแตกต่างกันไป ซึ่งผู้ออกแบบจำเป็นต้องรู้ข้อดีข้อเสียเหล่านี้ เพื่อให้สามารถเลือกใช้วิธีที่เหมาะสมกับจุดประสงค์การใช้งานของหน่วยประมวลผลนั้นๆ

วิธีหนึ่งที่ช่วยในการลดขนาดโปรแกรม คือ การออกแบบชุดคำสั่งแบบซีไอเอสซี (Complex Instruction Set Computer : CISC) ชุดคำสั่งประเภทนี้คำสั่งมีการทำงานซับซ้อนกว่าคำสั่งในชุดคำสั่งแบบอาร์ไอเอสซี (RISC) ทำให้โปรแกรมของหน่วยประมวลผลแบบซีไอเอสซี มีขนาดเล็กกว่าโปรแกรมของหน่วยประมวลผลแบบอาร์ไอเอสซี [7] แต่เนื่องจากหน่วยประมวลผลแบบอาร์ไอเอสซีมีประสิทธิภาพในการทำงานสูงกว่าหน่วยประมวลผลแบบซีไอเอสซี จากการใช้เทคนิกการประมวลผลแบบสายท่อ (Pipeline) ทำให้หน่วยประมวลผลแบบซีไอเอสซีจึงไม่เป็นที่นิยมในปัจจุบัน

เนื่องจากหน่วยประมวลผลแบบซีไอเอสซี (CISC) มีประสิทธิภาพการทำงานต่ำกว่าหน่วยประมวลผลแบบอาร์ไอเอสซี (RISC) งานวิจัยส่วนใหญ่จึงมุ่งเน้นในการพัฒนาประสิทธิภาพของหน่วยประมวลผลแบบอาร์ไอเอสซี ในการลดขนาดของโปรแกรมในหน่วยประมวลผลแบบอาร์-ไอเอสซี ในงานวิจัยของ เบซเซเดซ (Beszedez) [8] การลดขนาดโปรแกรมสามารถแบ่งออกได้เป็น 2 ประเภทคือ การอัดแน่นโปรแกรม (Code Compaction) และการบีบอัดโปรแกรม (Code Compression) วิธีการอัดแน่นโปรแกรมทำโดยการตัดคำสั่งที่ไม่จำเป็นในโปรแกรมออกไป การทำงานส่วนใหญ่เป็นหน้าที่ของคอมไพล์เลอร์ (Compiler) ส่วนการบีบอัดโปรแกรมเป็นการนำ

โปรแกรมมาทำการบีบอัดให้มีขนาดเล็กลง โปรแกรมที่ทำการบีบอัดต้องได้รับการคลายโปรแกรม (Decompression) โดยฮาร์ดแวร์หรือซอฟต์แวร์ก่อนจึงจะสามารถทำงานได้

การออกแบบชุดคำสั่งเพื่อให้ขนาดของโปรแกรมมีขนาดเล็ก เป็นการบีบอัดโปรแกรมวิธีหนึ่ง ตัวอย่างชุดคำสั่งที่ทำให้โปรแกรมมีขนาดเล็ก เช่น จาวาไบต์โค้ด (Java Bytecode) ซึ่งเป็นรหัสภาษาเครื่อง (Machine Code) ของเครื่องเสมือนจาวา (Java Virtual Machine : JVM) [9] ที่ออกแบบมาให้โปรแกรมมีขนาดเล็กเพื่อประสิทธิภาพในการส่งโปรแกรมในระบบเครือข่าย (Network)

โปรแกรมของจาวาไบต์โค้ดมีขนาดเล็ก เนื่องจากจาวาไบต์โค้ดเป็นคำสั่งที่มีทำงานแบบแสตก (Stack) คำสั่งส่วนใหญ่ดำเนินการบนแสตก ทำให้ในคำสั่งไม่ต้องมีบิตสำหรับกำหนดเลขที่อยู่ (Addressing) ของตัวถูกดำเนินการ (Operand) ในหน่วยความจำ คำสั่งส่วนใหญ่ของไบต์โค้ดจึงสั้นกว่าชุดคำสั่งทั่วไป เพราะประกอบด้วยรหัสดำเนินการ (Opcode) เพียงอย่างเดียว

2.2 งานวิจัยที่เกี่ยวข้อง

บทนี้จะกล่าวถึงงานวิจัยที่เกี่ยวข้องกับการลดขนาดของโปรแกรม และการนำวิธีการลดขนาดของโปรแกรมนั้นมาประยุกต์ใช้งานจริง การลดขนาดของโปรแกรมนั้นในปัจจุบันมีจุดประสงค์ 3 ประการ ได้แก่

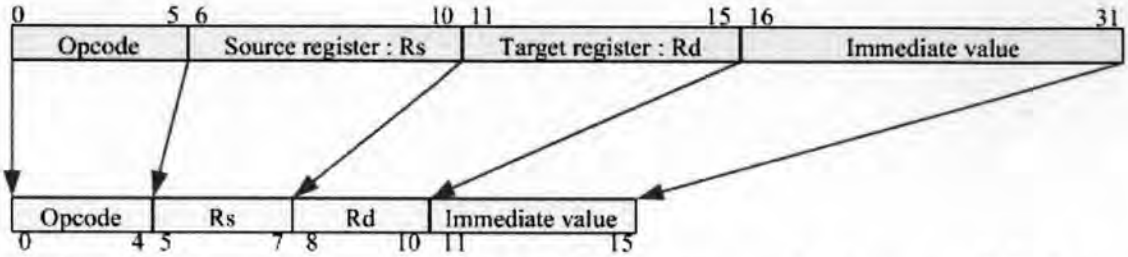
1. เพื่อลดต้นทุนในการผลิตชิป เนื่องจากต้นทุนการผลิตชิปแปรผันตรงกับขนาดของพื้นที่ดาย ซึ่งส่วนประกอบสำคัญของวงจรหน่วยประมวลผลคือ หน่วยความจำที่ทำหน้าที่เก็บโปรแกรม การลดขนาดของโปแกรมจึงสามารถช่วยลดขนาดของหน่วยความจำดังกล่าวและลดต้นทุนการผลิตชิปได้
2. เพื่อลดขนาดของโปรแกรมที่ต้องการส่งผ่านระบบเครือข่าย
3. เพื่อลดการใช้พลังงาน โดยการลดการเข้าถึงหน่วยความจำ

2.2.1 การดัดแปลงการออกแบบชุดคำสั่งของหน่วยประมวลผลแบบอาร์ไอเอสซี

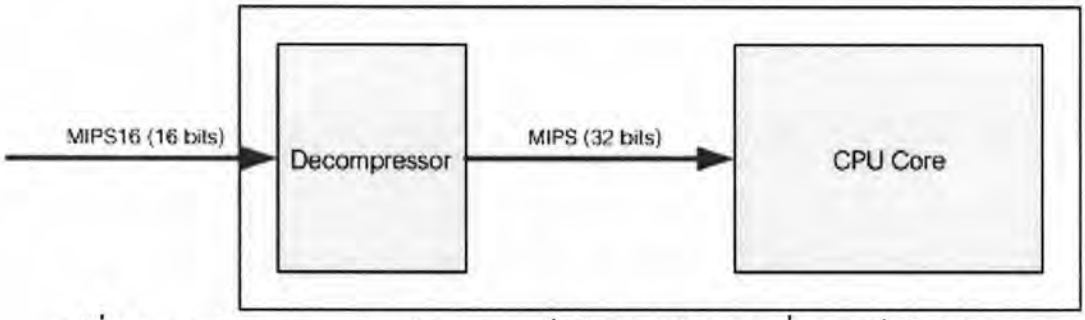
การดัดแปลงการออกแบบชุดคำสั่งของหน่วยประมวลผลแบบอาร์ไอเอสซี (RISC) เป็นการบีบอัดโปรแกรม (Code compression) วิธีหนึ่ง ทำโดยการออกแบบชุดคำสั่งพิเศษ ซึ่งคำสั่งในชุดคำสั่งนี้มีขนาดเล็กกว่าคำสั่งในชุดคำสั่งปกติ ดังแสดงในรูปที่ 2.2 ช่วยให้โปรแกรมที่ใช้ชุดคำสั่งพิเศษมีขนาดเล็กกว่าโปรแกรมที่เขียนด้วยชุดคำสั่งปกติ ตัวอย่างการใช้วิธีการนี้คือ ธัมบ์ (Thumb) [10] ของหน่วยประมวลผลตระกูลอาร์ม (ARM) และเอ็มไอพีเอส16 (MIPS16) [11] ของหน่วยประมวลผลตระกูลเอ็มไอพีเอส (MIPS)

ชุดคำสั่งพิเศษที่มีขนาดเล็กลงนี้ ได้มาจากการลดจำนวนบิตของคำสั่งในชุดคำสั่งปกติ การลดจำนวนบิตดังกล่าวพิจารณาจากการทำงานโปรแกรมทดสอบ เพื่อหาว่าบิตใดที่มีการใช้งานน้อยและสามารถตัดออกไปได้โดยไม่ส่งผลกระทบมากนัก ในขณะเดียวกันก็ทำการตัดคำสั่งในชุดคำสั่ง

ปกติที่มีการใช้งานน้อยออกไปด้วย เพื่อลดจำนวนบิตของรหัสดำเนินการ (Opcode) หน่วยประมวลผลที่ทำงานกับโปรแกรมที่บีบอัดนี้จะมีประสิทธิภาพการทำงานที่ต่ำลง เนื่องจากคำสั่งที่มีในการทำงานปกติบางคำสั่งไม่สามารถใช้งานได้ รีจิสเตอร์ที่เรียกใช้ได้มีน้อยลง อีกทั้งในการทำงานคำสั่งที่ถูกบีบอัดจะต้องถูกแปลงเป็นคำสั่งปกติก่อนที่จะถูกทำงานเหมือนคำสั่งปกติต่อไป ดังรูปที่ 2.3



รูปที่ 2.2 ตัวอย่างการลดจำนวนบิตในชุดคำสั่งเอ็มไอพีเอส16ของหน่วยประมวลผลเอ็มไอพีเอส



รูปที่ 2.3 การทำงานของหน่วยประมวลผลเอ็มไอพีเอสกับชุดคำสั่งแบบเอ็มไอพีเอส16

จากที่ได้กล่าวมาข้างต้น จะเห็นได้ว่าเมื่อหน่วยประมวลผลทำงานกับชุดคำสั่งพิเศษประสิทธิภาพการทำงานของหน่วยประมวลผลนั้นจะลดด้วยไปด้วย ซึ่งเป็นข้อเสียของการบีบอัดโปรแกรม (Code compression) ด้วยวิธีการนี้ อย่างไรก็ตามวิธีการดัดแปลงการออกแบบชุดคำสั่งของหน่วยประมวลผลแบบอาร์ไอเอสซี (RISC) นี้ สามารถลดขนาดของโปรแกรมได้ถึงประมาณร้อยละ 30 ของขนาดโปรแกรมปกติ

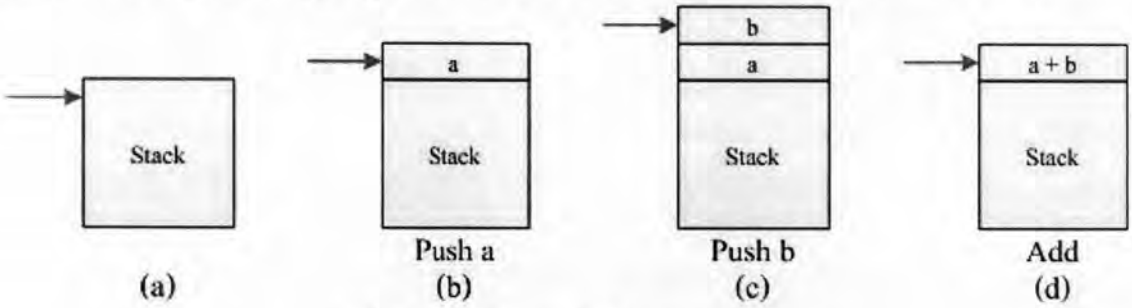
งานวิจัยของ กฤษณสวามี (Krishnaswamy) [12] ได้นำเสนอการผสมใช้ชุดคำสั่งอาร์มปกติและชุดคำสั่งธัมบ์เนื่องจากการใช้ชุดคำสั่งธัมบ์แม้ว่าจะช่วยให้ได้โปรแกรมที่มีขนาดเล็ก แต่ในขณะเดียวกันก็ทำให้ประสิทธิภาพการทำงานของหน่วยประมวลผลต่ำลงเช่นกัน งานวิจัยของกฤษณสวามี ได้พัฒนาอัลกอริทึม (Algorithm) ในการเลือกใช้ชุดคำสั่งทั้งสอง เพื่อให้ได้โค้ดที่มีขนาดเล็กและประสิทธิภาพที่ดีขึ้น

ในปี ค.ศ. 2003 หน่วยประมวลผลตระกูลอาร์มได้เปิดตัวชุดคำสั่ง ธัมบ์-2 (Thumb-2) ซึ่งเป็นชุดคำสั่งของหน่วยประมวลผลอาร์ม11 (ARM11) ชุดคำสั่ง ธัมบ์-2 นี้เป็นชุดคำสั่งที่ออกแบบมาเพื่อให้โปรแกรมมีขนาดเล็กเช่นเดียวกับชุดคำสั่งธัมบ์เดิม แต่ได้มีการปรับปรุงเพื่อขจัดข้อเสีย

ด้านประสิทธิภาพที่ลดลง ธัมบ์-2 นี้เป็นชุดคำสั่งขนาด 16 บิต และมีคำสั่งพิเศษขนาด 32 บิต หน่วยประมวลผลสามารถทำงานกับชุดคำสั่งนี้ได้ทันที โดยไม่จำเป็นต้องทำการคลายโปรแกรม (Decompression) เหมือนชุดคำสั่งธัมบ์ ชุดคำสั่งธัมบ์-2นี้ประสบความสำเร็จในการเพิ่มประสิทธิภาพการทำงานจนใกล้เคียงชุดคำสั่งอาร์มปกติ โดยที่ขนาดของโปรแกรมนั้นมีขนาดใกล้เคียงโปรแกรมที่เขียนด้วยชุดคำสั่งธัมบ์

2.2.2 การออกแบบวงจรหน่วยประมวลผลแบบแสตก

จากการที่การดำเนินการบนแสตก (Stack operation) สามารถนำมาเขียนเป็นคำสั่งได้กะทัดรัด เนื่องจากการดำเนินการบนแสตกนั้น ตัวถูกดำเนินการ (Operand) คือข้อมูลที่อยู่บนสุดของแสตก (Top of stack) ดังแสดงในรูปที่ 2.4 ทำให้คำสั่งที่มีการดำเนินการบนแสตกไม่จำเป็นต้องมีการระบุตัวถูกดำเนินการ ดังตัวอย่างแสดงในรูปที่ 2.4 เป็นการบวกค่าตัวแปร a และ b บนแสตก ขั้นตอนการทำงาน เริ่มต้นด้วยการดัน (Push) ค่า a และ b ลงบนแสตก จากนั้นคำสั่งบวกจะทำการบวกค่าของข้อมูล 2 ตัวที่อยู่บนสุดของแสตกจากนั้นดันค่าผลลัพธ์ที่ได้กลับลงแสตกเช่นเดิม



รูปที่ 2.4 ตัวอย่างการดำเนินการบนแสตก

จากรูปที่ 2.4 จะเห็นว่าคำสั่งบวก (Add) นั้นไม่จำเป็นต้องระบุตัวถูกดำเนินการ ซึ่งทำให้คำสั่งมีความกะทัดรัด และเล็กกว่าคำสั่งปกติซึ่งต้องมีการระบุตัวถูกดำเนินการด้วย จากข้อได้เปรียบนี้ ทำให้ในวงการวิจัยมีการออกแบบพัฒนาชุดคำสั่งที่มีโครงสร้างการทำงานบนแสตกออกมามากมาย เนื่องจากชุดคำสั่งประเภทนี้มีศักยภาพที่จะช่วยให้ได้โปรแกรมที่มีขนาดเล็ก ซึ่งมีประโยชน์ในหลายด้าน เช่น ลดปริมาณการส่งโปรแกรมผ่านระบบเครือข่าย (Network) และลดขนาดของหน่วยความจำที่ใช้ในการเก็บโปรแกรม เป็นต้น

การนำชุดคำสั่งแบบแสตกมาประยุกต์ใช้ที่เป็นที่รู้จักกันมากที่สุดคือ จาวาไบต์โค้ด (Java Bytecode) เป็นรหัสภาษาเครื่อง (Machine Code) ของเครื่องเสมือนจาวา (Java Virtual Machine : JVM) ซึ่งเป็นซอฟต์แวร์ (Software) ทำหน้าที่กระทำจาวาไบต์โค้ดบนระบบปฏิบัติการ (Operating system) หรือ ฮาร์ดแวร์ (Hardware) ที่มันฝังตัวอยู่

การทำงานของจาวาไบต์โค้ดนั้นเนื่องจากเป็นการทำงานบนเครื่องเสมือน ซึ่งต้องมีการแปลงจาวาไบต์โค้ดที่จะทำงานให้เป็นรหัสเครื่องของฮาร์ดแวร์ท้องถิ่น (Local hardware) การแปลง

รหัสนี้ทำให้เครื่องเสมือนจาวาไม่เหมาะสมที่จะนำมาใช้งานในระบบฝังตัว (Embedded system) เนื่องจากต้องเสียเวลาในการแปลงจาวาไบต์โค้ดมาเป็นรหัสภาษาเครื่องเสียก่อน จึงจะเริ่มทำงานได้ ทำให้ต้องเสียเวลาในการทำงานมากกว่าปกติ อีกทั้งระบบฝังตัวนั้นจะต้องเสียพื้นที่หน่วยความจำเพื่อเก็บโปรแกรมที่ทำหน้าที่เป็นเครื่องเสมือนจาวาอีกด้วย

จากข้อด้อยของการใช้เครื่องเสมือน (Virtual machine) ในการทำงานจาวาไบต์โค้ดที่ได้กล่าวไป งานวิจัยของ โอคอนเนอร์ และ เทรมบ์เลย์ [13] จึงได้วิจัยพัฒนาฮาร์ดแวร์ที่มีรหัสภาษาเครื่องเป็นจาวาไบต์โค้ด กล่าวคือ เป็นหน่วยประมวลผลที่มีชุดคำสั่งเป็นจาวาไบต์โค้ดนั่นเอง เพื่อกำจัดค่าใช้จ่ายในการแปลงจาวาไบต์โค้ดเป็นรหัสภาษาเครื่องก่อนการทำงาน ซึ่งช่วยเพิ่มศักยภาพของจาวาไบต์โค้ดในอุตสาหกรรมระบบฝังตัว

อีกงานวิจัยหนึ่งซึ่งเป็นการประยุกต์ชุดคำสั่งแบบแสตก มาช่วยในการลดขนาดโปรแกรม คืองานวิจัยของ นายภานุพันธ์ นันทนาวุฒิ และคณะ [14] งานวิจัยนี้ได้เริ่มมาจากการนำชุดคำสั่งแบบแสตก เรียกว่า "ไบต์โค้ด" (Byte code) มาเป็นโค้ดภาษาระดับกลาง (Intermediate code) ในการทำงานไบต์โค้ดนี้ ไบต์โค้ดจะถูกแปลงเป็นรหัสภาษาเครื่องด้วยวงจร ซึ่งต่างจากจาวาไบต์โค้ดที่ใช้ซอฟต์แวร์ในการแปลงรหัส การใช้วงจรในการแปลงรหัสช่วยให้หน่วยประมวลผลไม่ต้องเสียพื้นที่หน่วยความจำในการเก็บเครื่องเสมือน อีกทั้งการแปลงรหัสด้วยวงจรนั้นทำได้เร็วกว่าการแปลงด้วยซอฟต์แวร์

งานวิจัยของ นายอลงกต บุรุษอาชาไนย และคณะ [15] ได้นำเอาชุดคำสั่งไบต์โค้ดจากงานวิจัย [14] มาพัฒนาสร้างเป็นชุดคำสั่งของหน่วยประมวลผลแบบแสตก พัฒนาประสิทธิภาพการทำงานของไบต์โค้ด จากการที่ไม่มีการเสียเวลาในการแปลงรหัสไบต์โค้ดเป็นรหัสภาษาเครื่อง

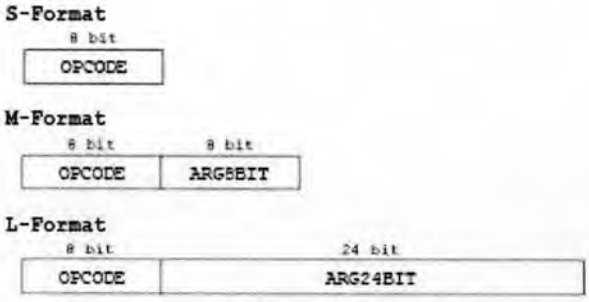
ในงานวิจัยถัดมา [16] พบว่า 53% ของเวลาที่ใช้ในการทำงานของหน่วยประมวลผลแบบแสตก [15] ถูกใช้ไปกับการดึงคำสั่ง (Instruction fetch) ในงานวิจัยนี้ได้ปรับปรุงวิธีการดึงคำสั่งของหน่วยประมวลผล เพื่อให้การดึงคำสั่งใช้เวลาน้อยลง ซึ่งวิธีนี้สามารถเพิ่มประสิทธิภาพในการทำงานของหน่วยประมวลผลแบบแสตกได้ถึง 32% และ 37% เมื่อใช้การบีบอัดโปรแกรมร่วมด้วย

2.2.3 การอัดคำสั่งของหน่วยประมวลผลแบบแสตก

งานวิจัยการอัดคำสั่งของหน่วยประมวลผลแบบแสตก [17] ได้นำเสนอการประยุกต์การบรรจุคำสั่ง หรือการอัดคำสั่ง (Instruction packing) ของหน่วยประมวลผลแบบแสตก [15] เพื่อเพิ่มประสิทธิภาพการทำงานให้แก่หน่วยประมวลผลแบบแสตกนี้ พร้อมทั้งปรับทางเดินข้อมูล (Data path) ของวงจรหน่วยประมวลผลเพิ่มเป็น 32 บิต โดยเรียกหน่วยประมวลผลแบบแสตกตัวใหม่นี้ว่า "หน่วยประมวลผลไว"

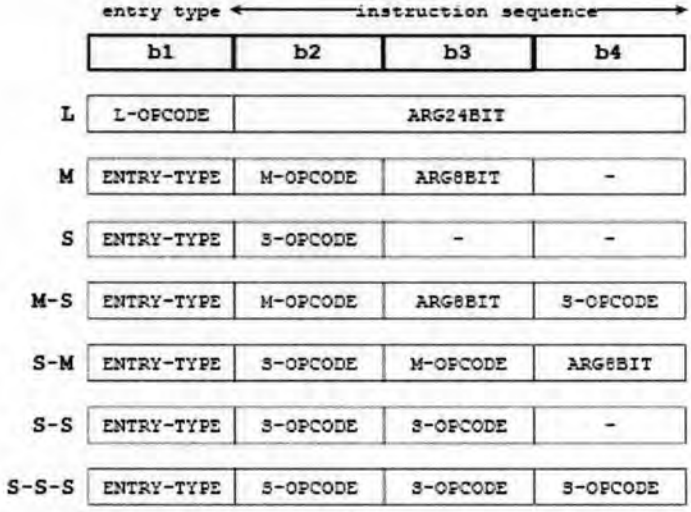
การอัดคำสั่งในหน่วยประมวลผลไวทำโดยการนำคำสั่งหลายๆคำสั่งมาบรรจุเข้าไว้ด้วยกัน ชุดคำสั่งของหน่วยประมวลผลไวนั้นมี 3 รูปแบบคือ คำสั่งขนาดเล็ก (Short format: S-Format)

คำสั่งขนาดกลาง (Medium format: M-Format) และคำสั่งขนาดยาว (Long format: L-Format) ดังแสดงในรูปที่ 2.5



รูปที่ 2.5 รูปแบบคำสั่งของหน่วยประมวลผลไว

คำสั่งของหน่วยประมวลผลไวเหล่านี้ ถูกรวมกันสร้างขึ้นเป็นคำสั่งอัดขนาด 32 บิต เพื่อให้ใช้พื้นที่ในหน่วยความจำให้ได้มากที่สุด การจัดเรียงคำสั่งมี 7 รูปแบบดังแสดงในรูปที่ 2.6 โดยไบต์แรก (b1) ทำหน้าที่บอกรูปแบบของการอัดคำสั่ง เพื่อให้หน่วยประมวลผลกระทำคำสั่งที่ถูกอัดได้อย่างถูกต้อง



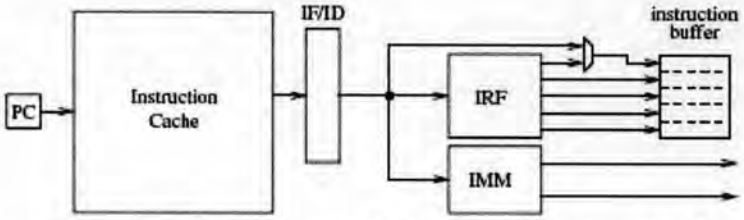
รูปที่ 2.6 รูปแบบการอัดคำสั่งของหน่วยประมวลผลไว

จากการอัดคำสั่งที่ใช้ในหน่วยประมวลผลไว ทำให้หน่วยประมวลผลนี้มีความเร็วเพิ่มขึ้นจากหน่วยประมวลผลแบบแสตก [15] เดิมถึง 2.12 เท่า และลดเวลาที่ใช้ในการอ่านคำสั่งลงเหลือเพียงร้อยละ 32 ของเวลาที่ในการทำงานทั้งหมด จากเดิมที่ใช้ถึงร้อยละ 55

2.2.4 การอัดคำสั่งลงในรีจิสเตอร์

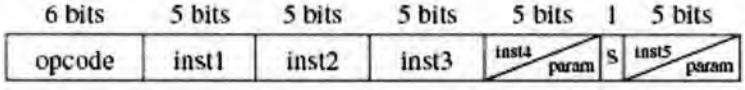
งานวิจัยของ ฮีนส์ (S. Hines) และคณะ [2, 3, 4] ได้ทำงานวิจัยการอัดคำสั่ง (Instruction packing) โดยใช้รีจิสเตอร์คำสั่ง (Instruction register file : IRF) การอัดคำสั่งในงานวิจัยนี้มีจุดประสงค์หลักเพื่อลดการใช้พลังงานในการดึงคำสั่ง (Instruction fetch)

การอัดคำสั่งของงานวิจัยนี้มีหลักการเหมือนการใช้รีจิสเตอร์ข้อมูล (Data register file) ของหน่วยประมวลผลทั่วไป นั้นคือคำสั่งที่ถูกใช้งานบ่อยในโปรแกรม 31 คำสั่งจะถูกเลือกโดยคอมไพล์เลอร์ (Compiler) ใส่ลงในรีจิสเตอร์คำสั่ง (IRF) ซึ่งเก็บคำสั่งได้ 32 คำสั่ง ดังแสดงในรูปที่ 2.7 นอกจากคำสั่งแล้วค่าของตัวถูกดำเนินการแบบทันที (Immediate operand) ที่มีการใช้งานบ่อยที่สุดก็ถูกใส่ไว้ในตารางเก็บตัวถูกดำเนินการแบบทันที (Immediate table : IMM) ซึ่งเก็บค่าได้ 32 ค่าเช่นกัน



รูปที่ 2.7 โครงสร้างการอัดคำสั่งด้วยวิธีการใช้รีจิสเตอร์คำสั่ง

การเรียกคำสั่งที่มีอยู่ในรีจิสเตอร์คำสั่งสามารถทำได้โดยการใช้คำสั่งพิเศษอ้างอิงรีจิสเตอร์คำสั่งที่คำสั่งนั้นถูกเก็บอยู่ คำสั่งพิเศษนี้มีรูปแบบดังแสดงในรูป 2.8 คำสั่งพิเศษขนาด 32 บิต สามารถอ้างอิงคำสั่งในรีจิสเตอร์คำสั่งได้สูงสุด 5 คำสั่ง หรืออาจใช้ในการอ้างอิงค่าของตัวถูกดำเนินการแบบทันที (Immediate operand) ที่ถูกเก็บอยู่ในตารางเก็บตัวถูกดำเนินการแบบทันที (Immediate table : IMM) รหัสดำเนินการ (Opcode) ขนาด 6 บิต และบิต S ทำหน้าที่กำหนดวิธีการนำส่วนประกอบต่างๆของคำสั่งไปใช้



รูปที่ 2.8 รูปแบบของคำสั่งที่ทำการอัดคำสั่งด้วยรีจิสเตอร์

การอัดคำสั่งลงในรีจิสเตอร์ช่วยลดขนาดของโปรแกรมได้ จากการที่คำสั่งขนาด 32 บิต สามารถอ้างอิงคำสั่งในรีจิสเตอร์คำสั่ง (IRF) ได้ถึง 5 คำสั่ง ช่วยลดพลังงานที่ใช้ในการดึงคำสั่ง (Instruction fetch) จากแคชคำสั่ง (Instruction cache) นอกจากนี้การใช้รีจิสเตอร์คำสั่งช่วยเพิ่มประสิทธิภาพการทำงานของแคชคำสั่ง เนื่องจากจำนวนคำสั่งที่เก็บในแคชคำสั่งมีมากขึ้นจากการที่คำสั่งหนึ่งคำสั่งในแคชคำสั่งสามารถอ้างอิงคำสั่งในรีจิสเตอร์คำสั่งได้หลายคำสั่ง