การบรรจุคำสั่งหน่วยประมวลผลแบบแสตกขนาด 32 บิตที่ประหยัดทรัพยากร

Instruction Packing for a 32-bit Resource Efficient Processor

ภานุพันธ์ นันทนาวุฒิ และอลงกต บุรุษอาชาไนย, ประภาส จงสถิตย์วัฒนา

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

254 ถนนพญาไท เขตปทุมวัน กรุงเทพมหานคร 10330

โทรศัพท์ 0-2218-6985 โทรสาร 0-2218-6955

E-mail: prabhas@chula.ac.th, phanupan@cp.eng.chula.ac.th and g46abl@cp.eng.chula.ac.th

บทคัดย่อ

บทความนี้นำเสนอแนวคิดในการออกแบบหน่วยประมวลผลแบบแสตกขนาด 32 บิตที่ชื่อว่า “ไว” สำหรับใช้ในระบบฝังตัวที่มีข้อดีตรงที่มีขนาดเล็กเหมาะสำหรับระบบที่ต้องการทรัพยากรน้อย เนื่องจากเป็นหน่วยประมวลผลที่ไม่ซับซ้อน อีกทั้งโปรแกรมที่ใช้ยังมีขนาดเล็ก หน่วยประมวลผลไวใช้ชุดคำสั่งแบบแสตกที่เรียกว่าคำสั่งรหัสไบต์ จึงนำเสนอวิธีการบรรจุหลายคำสั่งลงในหนึ่งหน่วยคำสั่งขนาด 32 บิต ผลการทดลองพบว่าเมื่อเทียบกับหน่วยประมวลผลที่ใช้คำสั่งรหัสไบต์โดยตรงจะมีสมรรถนะเพิ่มขึ้นจากเดิมถึง 2.12 เท่า ในขณะที่ขนาดของโปรแกรมทดสอบยังมีขนาดคงที่ถึงแม้ว่าชุดคำสั่งของหน่วยประมวลผลไวจะมีขนาดใหญ่กว่าคำสั่งรหัสไบต์ธรรมดาก็ตาม

คำสำคัญ: หน่วยประมวลผล, การบรรจุคำสั่ง, คำสั่งรหัสไบต์

Abstract

This paper proposed the conceptual design of a 32-bit stack processor for embedded systems. Its advantage is resource efficiency. Because of the stack processing, the design is simple and the programs used in this processor is small. This processor uses a stack-based instruction set called bytecode, so this research has proposed a technique of instruction packing which packs several instructions into a 32-bit instruction unit. According to the result of this research, it showed that the execution of packing instruction on VY has improved its performance up to 2.12 times of executing on bytecode directly, while the size of benchmark programs is still small, though the size of VY’s bytecode was bigger than the normal bytecode.

Keywords: Processor, Instruction packing, Bytecode instruction

1. บทนำ

ปัจจุบันอุตสาหกรรมการผลิตอุปกรณ์อิเลคทรอนิกส์ก้าวเข้ามามีบทบาทสำคัญกับภาคอุตสาหกรรมของไทยอย่างมาก ดังจะเห็นได้จากปริมาณการส่งออกผลิตภัณฑ์เฉพาะวงจรรวม (IC) ในปี 2541 ที่มีมูลค่าถึง 9.8 หมื่นล้านบาท ทว่าสายการผลิตยังเป็นเพียงแค่การผลิตปลายน้ำ (Midstream และ Downstream) เท่านั้น กล่าวคือยังเป็นเพียงแค่การประกอบวงจรรวม ผลิตแผ่น PCB ประกอบฮาร์ดไดร์ฟ (Hard drive) ตลอดไปจนถึงการผลิตเครื่องใช้ไฟฟ้าขั้นสุดท้ายเท่านั้น ดังนั้นจึงหลีกเลี่ยงไม่ได้ที่ประเทศไทยต้องนำเข้าต้นทุนชิ้นส่วนวงจรอิเลคทรอนิกส์ไม่ว่าจะเป็นแผ่นแว่นผลึก (Wafer) และชิปวงจรรวมชนิดต่างๆ เป็นปริมาณมาก ซึ่งในปี 2541 ประเทศไทยนำเข้าแผ่นแว่นผลึกมีมูลค่าถึง 7 หมื่นล้านบาท [1]

การยกระดับให้ประเทศไทยเป็นผู้ผลิตต้นน้ำ (Upstream) จึงเป็นสิ่งที่จะทำให้อุตสาหกรรมการผลิตอุปกรณ์อิเลคทรอนิกส์ไทยมั่นคงและยั่งยืน ปัจจุบันรัฐบาลได้สนับสนุนโดยเปิดโรงงานผลิตวงจรรวมขนาด 5 ไมครอน [1, 2] และที่ผ่านมาก็มีชิปหน่วย

ประมวลผลที่คนไทยพัฒนาขึ้นมาเองตัวอย่างเช่นหน่วยประมวลผลที่ใช้ชุดคำสั่ง 8051 [2, 3] และ Thaitum-2 ของศูนย์เทคโนโลยีไมโครอิเลคทรอนิกส์ (TMEC) [3] แต่หน่วยประมวลผลทั้งสองตัวเป็นหน่วยประมวลผลขนาด 8 บิต ทำให้ไม่สามารถรองรับการประมวลผลโปรแกรมประยุกต์ที่หลากหลายได้ นอกจากนั้นยังมีหน่วยประมวลผลขนาด 16 บิตที่ประกอบด้วยตัวกรองเอฟไอาร์ที่ปรับความยาวได้ [4] และหน่วยประมวลผลขนาด 16 บิตสำหรับควบคุมเครื่องรับโทรทัศน์ [5]

ในบทความนี้จึงต้องการนำเสนอแนวคิดการออกแบบหน่วยประมวลผลแบบแสตกขนาด 32 บิต (32-bit stack-based processor) ที่ใช้ชื่อว่า “ไว” (VY Stack ยrocessor) สำหรับใช้ในอุปกรณ์ฝังตัว (Embedded systems) โดยมีจุดมุ่งหมายเพื่อนำมาผลิตและใช้ภายในประเทศเพื่อทดแทนการนำเข้าหน่วยประมวลผลจากต่างประเทศ อีกทั้งยังเป็นการส่งเสริมให้เกิดการออกแบบวงจรรวมภายในประเทศ

เนื้อหาจะแบ่งออกเป็น 6 ส่วนด้วยกัน เริ่มจากในหัวข้อที่ 2 จะกล่าวถึงแนวคิดที่ใช้ในการออกแบบหน่วยประมวลผลไว ส่วนในหัวข้อที่ 3 และ 4 จะกล่าวถึงการทดลอง ผลการทดลอง และการวิเคราะห์ผลการทดลองที่ได้ สุดท้ายในหัวข้อที่ 5 และ 6 จะเป็นการสรุปเนื้อหาในบทความนี้และกล่าวถึงสิ่งที่จะทำต่อไปในอนาคต

2. แนวคิดการออกแบบ

การออกแบบหน่วยประมวลผลไวได้แนวคิดมาจากหน่วยประมวลผลแบบแสตกขนาด 16 บิต ที่มีชุดคำสั่งแบบแสตกอย่างง่ายที่ครอบคลุมการทำงานในระบบเลขจำนวนเต็ม [6] โดยในบทความนี้จะเรียกหน่วยประมวลผลดังกล่าวว่า “หน่วยประมวลผล SMC” และเรียกชุดคำสั่งแบบแสตกของหน่วยประมวลนี้ว่า “คำสั่งรหัสไบต์ (Bytecode)” ข้อดีของหน่วยประมวลผล SMC คือ

1. เป็นหน่วยประมวลผลแบบแสตกทำให้การออกแบบไม่ซับซ้อน และใช้ทรัพยากรน้อย
2. ขนาดของโปรแกรมที่ใช้ในหน่วยประมวลผลแบบแสตกมีขนาดเล็กกว่าแบบเรจิสเตอร์เนื่องจากคำสั่งรหัสไบต์มีขนาดเล็กกว่าชุดคำสั่งแบบเรจิสเตอร์ (Register-based instruction set) ส่งผลให้ประหยัดพื้นที่หน่วยความจำโปรแกรม ซึ่งเป็นทรัพยากรที่สำคัญอย่างหนึ่งในระบบฝังตัว

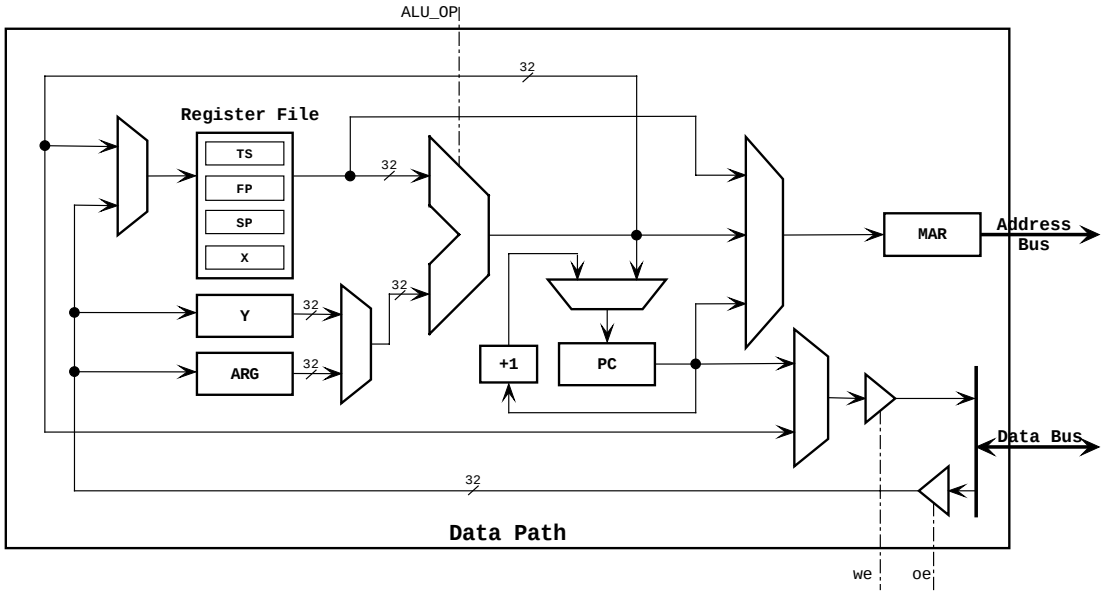
แต่หน่วยประมวลผล SMC มีจุดอ่อนที่สมรรถนะ (Performance) เนื่องจากเดิมนั้นส่วนทางเดินข้อมูล (Data path) ถูกออกแบบไว้อย่างจำกัด ทำให้ขั้นตอนการประมวลผลคำสั่งแต่ละคำสั่งมีมากเกินไป โดยเฉพาะในขั้นตอนการอ่านคำสั่งซึ่งออกแบบให้อ่านคำสั่งได้เพียงครั้งละหนึ่งไบต์ ในขณะที่คำสั่งมีขนาดยาวสุดถึงสามไบต์ ในบทความของอลงกตและคณะ [7] ได้รายงานไว้ว่า SMC เสียเวลากว่า 53% ของเวลาทั้งหมดในขั้นตอนการอ่านคำสั่ง

ดังนั้นในการออกแบบหน่วยประมวลผลไวนอกจากจะออกแบบให้รองรับการประมวลผลข้อมูลขนาด 32 บิตแล้ว ยังจะปรับปรุงส่วนทางเดินข้อมูลและขั้นตอนการทำงานโดยเฉพาะในขั้นตอนการอ่านคำสั่งเพื่อเพิ่มสมรรถนะการทำงานให้สูงขึ้น

อนึ่งการที่หน่วยประมวลผลไวรองรับการประมวลผลข้อมูลขนาด 32 บิต ทำให้ชุดคำสั่งมีขนาดใหญ่กว่าชุดคำสั่ง SMC ซึ่งจะส่งผลให้ขนาดของโปรแกรมที่ได้อาจใหญ่กว่าโปรแกรมที่ใช้ใน SMC ดังนั้นจึงได้นำเสนอการออกแบบการบรรจุคำสั่งรหัสไบต์หลายคำสั่งลงหน่วยคำสั่งขนาด 32 บิต พร้อมกับเพิ่มคำสั่งพิเศษในชุดคำสั่งเพื่อให้ขนาดของโปรแกรมยังคงมีขนาดเล็กเหมือนกับในหน่วยประมวลผล SMC

2.1 การปรับปรุงส่วนทางเดินข้อมูล และสัญญาณนาฬิกาแบบสองเฟส

ในการออกแบบหน่วยประมวลผล SMC เดิมนั้นได้ออกแบบในส่วนทางเดินข้อมูลไว้จำกัดมากเพื่อใช้ทรัพยากรให้น้อยที่สุด แต่กลับส่งผลให้ขั้นตอนการประมวลผลแต่ละคำสั่งมีมากเกินไป ทำให้ส่วนควบคุมมีขนาดใหญ่และทำงานช้าลงมาก ดังนั้นในการออกแบบหน่วยประมวลผลไวนี้จะปรับปรุงส่วนทางเดินข้อมูลเล็กน้อยเพื่อให้เกิดความคล่องตัวขึ้น และลดขั้นตอนการทำงานของแต่ละคำสั่งลง



รูปที่ 1 ส่วนทางเดินข้อมูลที่ได้รับการปรับปรุงแล้ว

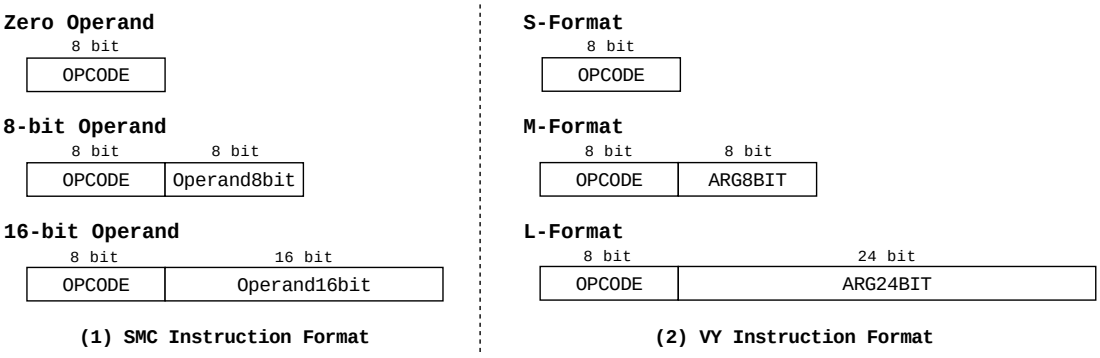
สิ่งที่ปรับปรุงคือการแยกเรจิสเตอร์ PC ออกมาจากส่วนแฟ้มเรจิสเตอร์ดังแสดงในรูปที่ 1 เนื่องจากเดิมนั้นเรจิสเตอร์ PC จะอยู่ในแฟ้มเรจิสเตอร์ เวลาอ่านคำสั่งจะไม่สามารถเพิ่มค่า PC พร้อมกับเก็บค่าลงเรจิสเตอร์ MAR ได้ ต้องรอให้ค่า PC เดิมถูกนำไปเก็บใน MAR ก่อนจึงเพิ่มค่า PC ได้ นอกจากนั้นการเพิ่มค่า PC ยังทำผ่าน ALU ทำให้การคำนวณต่างๆ ต้องรอจนกว่าจะคำนวณค่า PC เสร็จสิ้นก่อนจึงเริ่มคำนวณได้

นอกจากการแยกเรจิสเตอร์ PC ออกมาแล้วยังปรับให้มีการเก็บค่าลงเรจิสเตอร์โดยใช้หลักการของสัญญาณนาฬิกาแบบสองเฟส ซึ่งนำเสนอไว้ในบทความของอลงกตและประภาส [5] ซึ่งช่วยลดขั้นตอนทั้งการอ่านคำสั่งและข้อมูลจากหน่วยความจำลงไปได้มาก

2.2 การปรับปรุงชุดคำสั่ง

ชุดคำสั่งของหน่วยประมวลผลไวจะใช้ชุดคำสั่งของหน่วยประมวลผล SMC เป็นหลักเพียงแต่ต้องปรับให้หน่วยประมวลผลไวรองรับการประมวลผลข้อมูลขนาด 32 บิตซึ่งจำเป็นต้องดัดแปลงชุดคำสั่งจากหน่วยประมวลผล SMC ในรูปแบบตัวถูกดำเนินการ 16 บิตซึ่งเดิมมีตัวถูกดำเนินการขนาด 16 บิตให้เป็นตัวถูกดำเนินการขนาด 24 บิต ส่วนในรูปแบบอื่นจะคงไว้ดังเดิม

ในที่นี้จะอธิบายโดยแบ่งชุดคำสั่งของหน่วยประมวลผลไวแบ่งออกเป็น 3 รูปแบบได้แก่ คำสั่งขนาดเล็ก (Small instruction format: S-Format) คำสั่งขนาดกลาง (Medium instruction format: M-Format) และคำสั่งขนาดยาว (Long instruction format: L-Format) ดังรูปที่ 2(2)



รูปที่ 2 รูปแบบชุดคำสั่งของหน่วยประมวลผลไว

รูปแบบคำสั่งขนาดเล็กและขนาดกลางจะนำมาใช้กับคำสั่งที่ไม่มีตัวถูกดำเนินและมีตัวถูกดำเนินการขนาด 8 บิตของชุดคำสั่ง SMC เดิม (รูปที่ 2(1)) ส่วนรูปแบบคำสั่งขนาดยาวจะนำมาใช้กับคำสั่งที่มีตัวถูกดำเนินการขนาด 16 บิต โดยจะปรับให้ขนาดของตัวถูกดำเนินการจาก 16 บิตเป็น 24 บิต

ในส่วนชุดคำสั่งจะเพิ่มคำสั่งใหม่เข้าไปโดยมีวัตถุประสงค์เพื่อลดขนาดของโปรแกรมโดยรวมและเพิ่มประสิทธิภาพในการประมวลผล คำสั่งที่เพิ่มเข้ามาจะพิจารณาจากลำดับคำสั่งที่พบบ่อยในการเขียนโปรแกรม เช่นคำสั่งที่ใช้ในการเข้าถึงข้อมูลแบบอาร์เรย์ การลดขนาดคำสั่งการกระโดดไปในตำแหน่งที่ใกล้ และคำสั่งที่มีตัวถูกดำเนินการที่ใช้บ่อย ดังมีรายละเอียดดังนี้ (พิจารณาประกอบกับตารางที่ 1)

1. เพิ่มคำสั่งโหลดและสโตรแบบสัมบูรณ์ (Absolute load/store: LDA, STA) เพื่อให้สามารถเข้าถึงข้อมูลในหน่วยความจำได้ในคำสั่งเดียว
2. เพิ่มคำสั่งโหลดและสโตรแบบดรรชนี (Index load/store: LDX, STX) เพื่อใช้แทนลำดับของคำสั่งในการอ่านข้อมูลในอาร์เรย์
3. เพิ่มคำสั่งการกระโดดแบบสัมพัทธ์ (Relative Jump: JMPL, JTL, JFL) โดยลดขนาดของตัวถูกดำเนินการจากเดิม 24 บิต (L-Format) ลงเหลือ 8 บิต (M-Format) เนื่องจากการกระโดดในโปรแกรมส่วนมากจะไปในตำแหน่งที่ไม่ไกลมาก
4. เพิ่มคำสั่งบวก (ADDV) ค่าบนแสตกกับตัวแปรโลคอลตัวที่กำหนดในตัวถูกดำเนินการ
5. เพิ่มคำสั่ง LIT0, LIT1, GET1, GET2, GET3 และ GET4 ในรูปแบบคำสั่งขนาดเล็กแทนคำสั่ง LIT และ GET ในรูปแบบคำสั่งขนาดยาวและกลางตามลำดับ เนื่องจากคำสั่งเหล่านี้ใช้ตัวถูกดำเนินการ 0, 1, 2, 3 และ 4 บ่อยมาก ดังนั้นถ้ามีการใช้คำสั่งพิเศษเหล่านี้แทนคำสั่งเดิมจะทำให้ลดขนาดโปรแกรมโดยรวมลงได้มาก
6. เพิ่มคำสั่ง JLE (Jump of less than or equal) เพื่อใช้ในการวนลูป (For-loop)

ตารางที่ 1 การแปลงลำดับคำสั่งเป็นคำสั่งพิเศษ

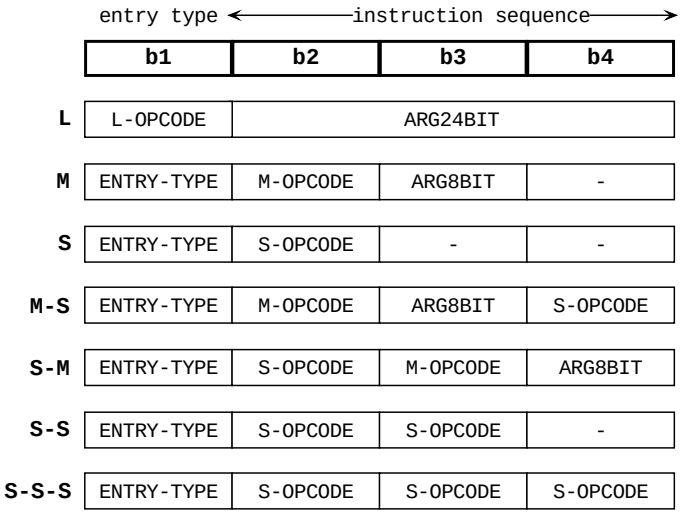
	Old sequence	New bytecode	Description	Size changed
1.	LIT abs-addr LD	LDA abs-addr	Load data from absolute address to top of stack	5 byte to 4 byte
	GET n-local LIT abs-addr ST	GET n-local STA abs-addr	Store top of stack to absolute address	
2.	GET n-local LIT base-addr GET n-local LD	GET n-local LIT base-addr LDX n-local	Load data from memory by using local variable index	3 byte to 2 byte
	GET n-local LIT base-addr GET n-local ST	GET n-local LIT base-addr LDX n-local	Store data to memory using local variable index	
3.	JMP rel-24bit	JMP rel-8bit	Unconditional jump	4 byte to 2 byte
	JT rel-24bit	JT rel-8bit	Jump if true	
	JF rel-24bit	JF rel-8bit	Jump if false	
4.	GET n-local GET n-local ADD	GET n-local ADDV n-local	Add 2 local variable together	3 byte to 2 byte
5.	LIT 0 LIT 1 GET 1 GET 2 GET 3 GET 4	LIT0 LIT1 GET1 GET2 GET3 GET4	Convert M-format to S-format	2 byte to 1 byte
6.	LE JT rel-8bit	JLE rel-8bit	Jump if less than or equal	5 byte to 2 byte

2.3 การบรรจุคำสั่งรหัสไบต์และขั้นตอนการอ่านคำสั่ง

การอ่านคำสั่งในหน่วยประมวลผล SMC เดิมนั้นอ่านคำสั่งทีละไบต์ทั้งที่คำสั่งมีตั้งแต่ขนาด 1 ถึง 3 ไบต์ ทำให้การทำงานช้าลงมาก โดยใช้เวลาในการอ่านคำสั่งทั้งสิ้น 53% ของเวลาทำงานทั้งหมด [6] และยิ่งในชุดคำสั่งของหน่วยประมวลผลไวถูกออกแบบให้ขนาดยาวที่สุด 32 บิต ถ้าจัดเก็บคำสั่งต่างๆ เรียงต่อกันและอ่านคำสั่งทีละไบต์เหมือนในหน่วยประมวลผล SMC จะทำให้หน่วยประมวลผลทำงานช้าลงมาก

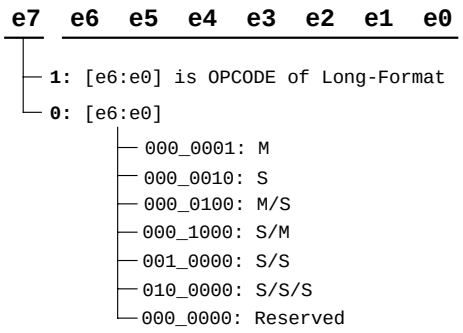
แต่ถ้าออกแบบให้หนึ่งคำสั่งมีขนาด 32 บิตทุกคำสั่งและอ่านครั้งละ 32 บิตเพื่อลดเวลาในการอ่านคำสั่ง จะทำให้สิ้นเปลืองเนื้อที่ในการจัดเก็บคำสั่งมาก เนื่องจากในคำสั่งรูปแบบขนาดเล็กและกลางนั้นมีขนาดเพียง 8 บิตและ 16 บิตทำให้เกิดที่ว่างเสียไปในหน่วยความจำ

ดังนั้นในการออกแบบนี้จึงออกแบบการบรรจุคำสั่งรหัสไบต์ (Bytecode packing) ให้สามารถอ่านคำสั่งครั้งละ 32 บิตโดยใช้พื้นที่ในหน่วยความจำให้ได้มากที่สุด โดยออกแบบการจัดคำสั่งออกเป็น 7 รูปแบบด้วยกันได้แก่ การจัดเรียงแบบ L, M, S, M-S, S-M, S-S และแบบ S-S-S ดังแสดงในรูปที่ 3



รูปที่ 3 การจัดเรียงคำสั่งแบบต่างๆ

โครงสร้างของการบรรจุจะบรรจุคำสั่งรหัสไบต์ทีละ 32 บิตเรียกว่า “หน่วยคำสั่ง (Instruction entry)” ไบต์แรกของแต่ละหน่วยคำสั่งเป็นตัวบอกชนิดการบรรจุ (Entry type) โดยถ้าบิตที่ 7 ของตัวบอกชนิดมีค่าเป็น 1 หน่วยคำสั่งนั้นจะเป็นคำสั่งในรูปแบบขนาดยาว และไบต์ชนิดการบรรจุนั้นจะถือว่าเป็นตัวดำเนินการ (Opcode) ของคำสั่งขนาดยาว ดังแสดงในรูปที่ 4 แต่ถ้าบิตแรกของตัวบอกชนิดเป็น 0 บิตที่เหลือจะเป็นตัวแยกประเภทของการบรรจุดังรูปที่ 4 และในไบต์ต่อๆ ไปของหน่วยคำสั่งก็จะเป็นคำสั่งรหัสไบต์เรียงไปตามชนิดของการจัดเรียงในรูปที่ 3



รูปที่ 4 การเข้ารหัสหน่วยคำสั่งในการจัดเรียงคำสั่ง

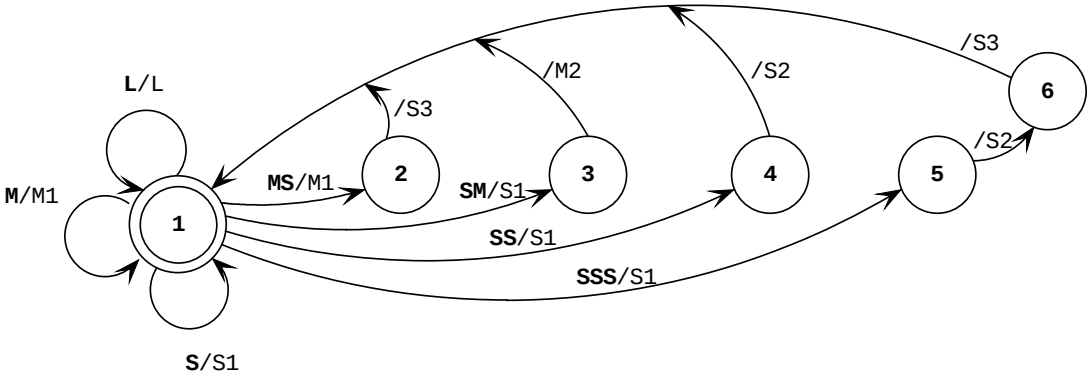
จากการปรับส่วนทางเดินข้อมูลและการออกแบบการบรรจุคำสั่งทำให้การอ่านคำสั่งเหลือเพียง 2 ขั้นตอนดังแสดงรูปที่ 5

```
1.   PC → \MAR, PC + 1 → \PC, memory read
2.   data bus (read) → /IR
```

รูปที่ 5 ขั้นตอนการอ่านคำสั่งของหน่วยประมวลผลไว

\MAR และ \PC คือการเขียนเรจิสเตอร์ MAR และ PC ตอนสัญญาณนาฬิกาลง, /IR คือการเขียนเรจิสเตอร์ IR ตอนสัญญาณนาฬิกาขึ้น

เมื่อหน่วยประมวลผลอ่านคำสั่งจะอ่านมาทั้งหน่วยคำสั่ง แล้วจึงถอดรหัสและประมวลผลคำสั่งทีละคำสั่ง โดยการถอดรหัสคำสั่งจะมีขั้นตอนดังแผนภูมิสถานะดังรูปที่ 6 โดยที่สถานะที่ 1 เป็นสถานะอ่านคำสั่ง เมื่ออ่านและถอดรหัสจนรู้ชนิดของหน่วยคำสั่งแล้ว จะแยกไปประมวลผลตามลำดับคำสั่งในรูปแบบการจัดเรียงนั้นๆ



รูปที่ 6 ขั้นตอนการถอดรหัสและประมวลผลคำสั่งในการบรรจุคำสั่งแบบต่างๆ

3. การทดลองและผลการทดลอง

เนื่องจากในบทความนี้นำเสนอเพียงแค่แนวคิดในการออกแบบเท่านั้น ดังนั้นการทดลองจึงเป็นเพียงการจำลองการทำงานของหน่วยประมวลผลด้วยโปรแกรมภาษาซี โดยปรับปรุงตัวแปลภาษา (Compiler) ของหน่วยประมวลผล SMC ให้สามารถรองรับการบรรจุคำสั่งเป็นหน่วยคำสั่งที่นำเสนอในหัวข้อที่ 2.3 สำหรับโปรแกรมทดสอบ (Benchmark) ใช้โปรแกรมทดสอบแบบจำนวนเต็มของสแตนฟอร์ด (Stanford's integer benchmark) ดังแสดงรายละเอียดในตารางที่ 2

ตารางที่ 2 รายละเอียดของโปรแกรมทดสอบจำนวนเต็มของสแตนฟอร์ด

Benchmark	Description
bubble	Sort 20 numbers using bubble sort algorithm
hanoi	Find a solution to move 6 disks in "Tower of Hanoi"
matmul	Multiply matrix 4×4
perm	Permute 4 digits of {0, 1, 2, 3}
8-queen	Find a number of solution in "8-queen" problem
quick	Sort 20 numbers using quick sort algorithm
sieve	Find all prime numbers less than 500

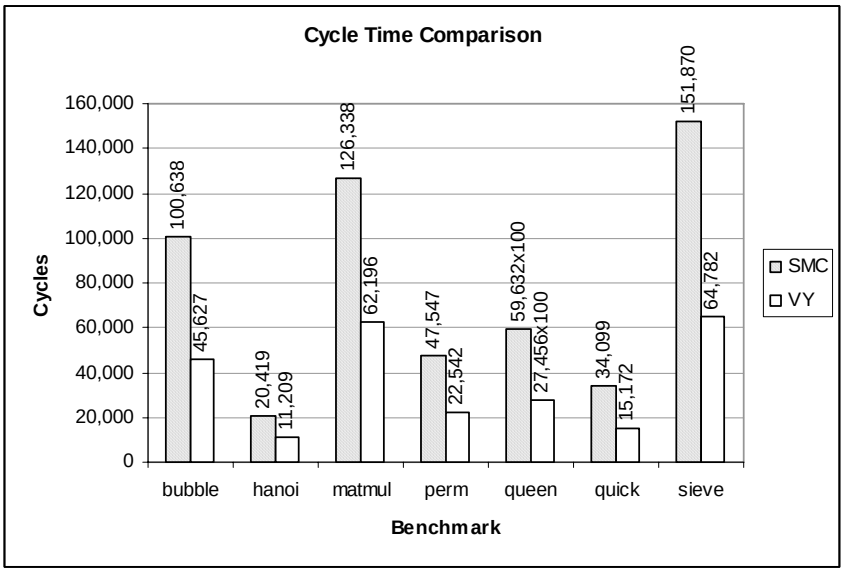
การทดลองจะวัดผลการทำงานของหน่วยประมวลผลไวเปรียบเทียบกับหน่วยประมวลผล SMC ที่ปรับปรุงให้สามารถทำงานกับสัญญาณนาฬิกาแบบสองเฟส [8] โดยสิ่งที่สนใจมีด้วยกันสองประการคือสมรรถนะการทำงานและขนาดของโปรแกรมทดสอบ

ผลการทดลองทั้งหมดแสดงดังรูปที่ 7 รูปที่ 8 รูปที่ 9 ตารางที่ 3 รูปที่ 10 และ ตารางที่ 4 โดยรูปที่ 7 และรูปที่ 8 เป็นกราฟเปรียบเทียบจำนวนสัญญาณนาฬิกาและจำนวนคำสั่งที่ถูกประมวลผลทั้งหมดที่ใช้ในการทำงานของแต่ละโปรแกรมทดสอบ พบว่าหน่วยประมวลผลไวทำงานเร็วกว่าหน่วยประมวลผล SMC ถึง 2.12 เท่า ในขณะที่จำนวนคำสั่งที่ประมวลผลในหน่วยประมวลผลไวลดลงคิดเป็น 81% ของหน่วยประมวลผล SMC และเมื่อคำนวณจำนวนสัญญาณนาฬิกาที่ใช้ในการประมวลผลหนึ่งคำสั่ง (Cycle per instruction, CPI) ของโปรแกรมทดสอบได้ค่าเฉลี่ยของหน่วยประมวลผลไวอยู่ที่ 4.99 สัญญาณนาฬิกาต่อหนึ่งคำสั่ง ในขณะที่หน่วยประมวลผล SMC ต้องใช้ 8.61 สัญญาณนาฬิกาต่อหนึ่งคำสั่ง

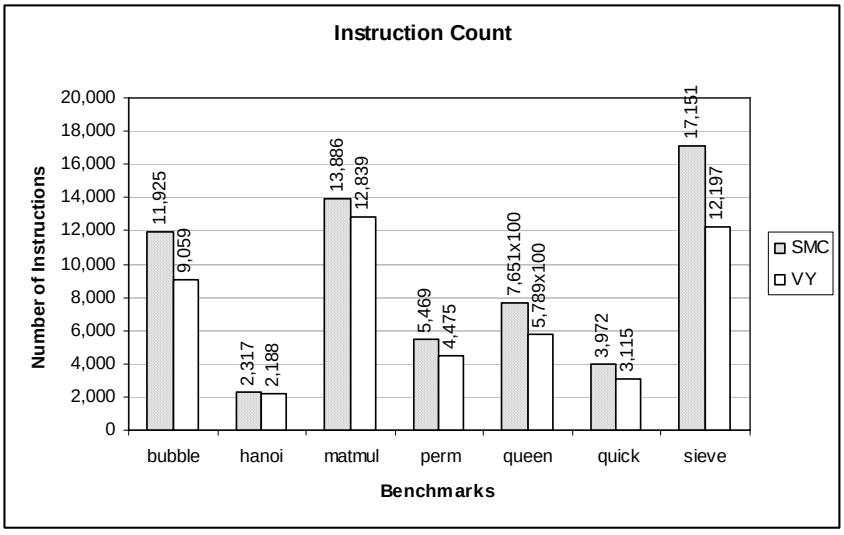
ในตารางที่ 3 แสดงสัดส่วนเวลาที่ใช้ในการอ่านคำสั่งของแต่ละโปรแกรมโดยเปรียบเทียบกับเวลาที่ใช้ในการทำงานทั้งหมดของโปรแกรม พบว่าหน่วยประมวลผลไวใช้เวลาอ่านคำสั่ง 32% ของเวลาทั้งหมด ในขณะที่ SMC ใช้เวลาในการอ่านคำสั่ง 55% ของเวลาทั้งหมด และในรูปที่ 9 เป็นกราฟเปรียบเทียบจำนวนครั้งในการเข้าถึงหน่วยความจำทั้งการเข้าถึงคำสั่งและข้อมูลในหน่วยความจำของหน่วยประมวลผลทั้งสอง พบว่าจำนวนการเข้าถึงหน่วยความจำของหน่วยประมวลผลไวเป็น 25% ของหน่วยประมวลผล SMC

สำหรับรูปที่ 10 เป็นกราฟเปรียบเทียบขนาดของโปรแกรมทดสอบที่ใช้สำหรับหน่วยประมวลผลทั้งสอง โดยแสดงขนาดเป็นไบต์ของหน่วยความจำที่ใช้ในการจัดเก็บโปรแกรมทั้งหมด พบว่าขนาดของโปรแกรมทั้งสองแบบมีขนาดใกล้เคียงกันมาก

สุดท้ายในตารางที่ 4 แสดงถึงจำนวนหน่วยความจำที่ว่างจากการบรรจุคำสั่งในโปรแกรมที่ใช้สำหรับหน่วยประมวลผลไว พบว่ามีพื้นที่ว่างที่ไม่ได้ใช้งานในหน่วยความจำ 12%



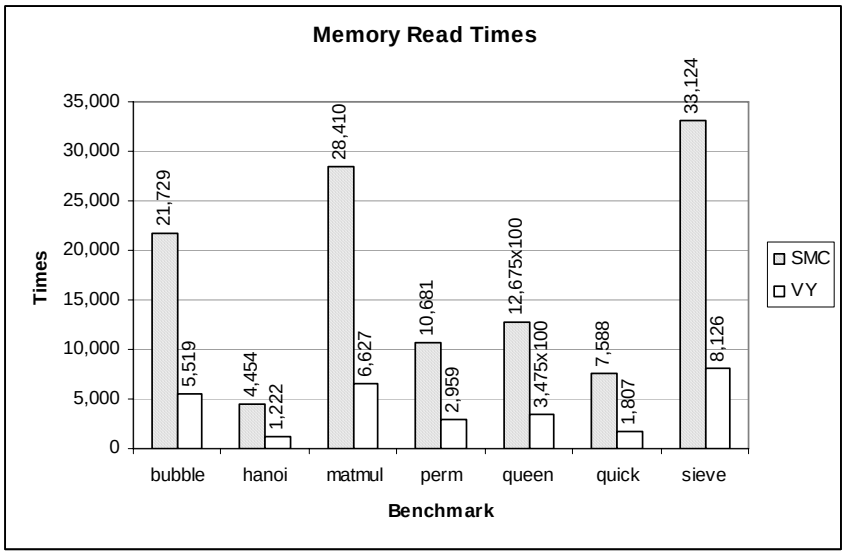
รูปที่ 7 กราฟเปรียบเทียบรอบสัญญาณนาฬิกาที่ใช้ในการประมวลผลแต่ละโปรแกรมของหน่วยประมวลผลทั้งสอง



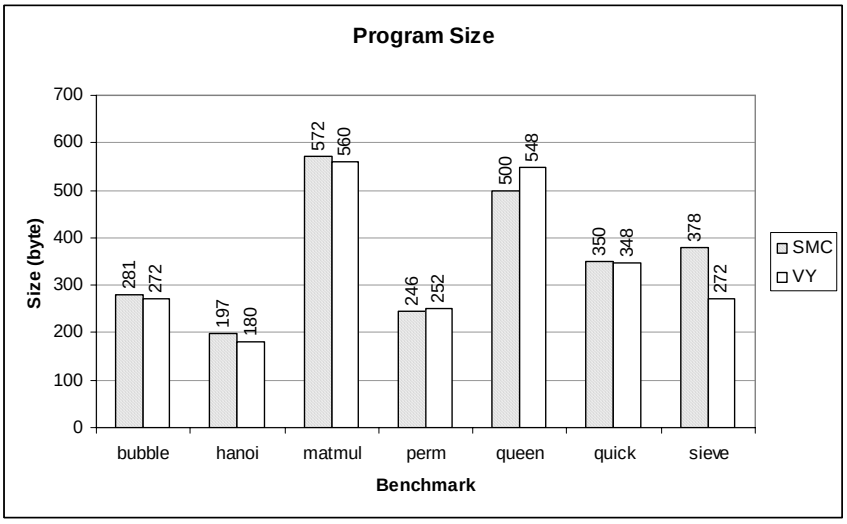
รูปที่ 8 กราฟเปรียบเทียบจำนวนคำสั่งที่ใช้ในการประมวลผลแต่ละโปรแกรมของหน่วยประมวลผลทั้งสอง

ตารางที่ 3 สัดส่วนเวลาที่ใช้ในการอ่านคำสั่งของแต่ละหน่วยประมวลผล

Benchmarks	SMC %fetch	VY %fetch
bubble	55%	32%
hanoi	54%	31%
matmul	56%	32%
perm	56%	33%
queen	55%	34%
quick	56%	33%
sieve	55%	32%
Average	**55%**	**32%**



รูปที่ 9 กราฟเปรียบเทียบจำนวนครั้งในการเข้าถึงหน่วยความจำทั้งคำสั่งและข้อมูล



รูปที่ 10 ขนาดของโปรแกรมทดสอบในรูปแบบชุดคำสั่ง SMC เปรียบเทียบกับ VY

ตารางที่ 4 จำนวนหน่วยความที่เสียไปจากการเข้ารหัสคำสั่ง

Benchmarks	VY (byte)	NOP (Byte)	%NOP
bubble	272	34	12.5%
hanoi	180	15	8.33%
matmul	560	61	10.89%
perm	252	32	12.70%
queen	548	75	13.69%
quick	348	40	11.49%
sieve	272	36	13.24%
NOP Byte Average			11.83%

4. วิเคราะห์ผลการทดลอง

เมื่อพิจารณาเวลาที่ใช้ในการประมวลผลโปรแกรมทดสอบเฉลี่ยพบว่าสมรรถนะของหน่อยประมวลผลไวสูงกว่าของหน่วยประมวลผล SMC ถึง 2.12 เท่า และหน่วยประมวลผลไวใช้สัญญาณนาฬิกาเฉลี่ย 5 สัญญาณในการประมวลผลคำสั่งหนึ่งคำสั่ง ในขณะที่หน่วยประมวลผล SMC มีค่าเฉลี่ยอยู่ที่ 8.61 สัญญาณต่อหนึ่งคำสั่ง ปัจจัยที่มีน่าจะผลต่อสมรรถนะของหน่วยประมวลผลคือ

1. การเพิ่มคำสั่งพิเศษเข้าไปทำให้โปรแกรมที่ทำงานบนหน่วยประมวลผลไวมีจำนวนคำสั่งที่ถูกประมวลผลลดลงถึง 20% ของหน่วยประมวลผล SMC
2. เวลาที่ใช้การอ่านคำสั่งและปริมาณการเข้าถึงหน่วยความจำของหน่วยประมวลผลทั้งสองลดลง ดังแสดงในตารางที่ 3 และรูปที่ 9 ซึ่งหน่วยประมวลผลไวใช้เวลาในการอ่านคำสั่ง 32% ของเวลาประมวลผลทั้งหมดในขณะที่หน่วยประมวลผล SMC ใช้เวลาถึง 55% อีกทั้งหน่วยประมวลผลไวเข้าถึงหน่วยความจำเป็น 25% ของหน่วยประมวลผล SMC

สำหรับขนาดของโปรแกรมที่ใช้ในหน่วยประมวลผลไวนั้น ในตอนแรกคาดว่าขนาดของโปรแกรมน่าจะมีขนาดที่ใหญ่กว่าโปรแกรมที่ใช้ในหน่วยประมวลผล SMC ไม่ต่ำกว่า 2 เท่า เนื่องจากหน่วยประมวลผลไวเป็นหน่วยประมวลผลขนาด 32 บิต ชุดคำสั่งของหน่วยประมวลผลไวจึงถูกออกแบบให้รองรับการทำงานกับข้อมูล 32 บิต ทำให้ขนาดของคำสั่งยาวขึ้นเป็น 32 บิต จึงน่าจะทำให้ขนาดของโปรแกรมโดยรวมใหญ่ขึ้น แต่จากกราฟในรูปที่ 10 พบว่าโปรแกรมทดสอบที่ถูกแปลให้อยู่ในรูปแบบของหน่วยประมวลผลไวมีขนาดใกล้เคียงกับหน่วยประมวลผล SMC สาเหตุที่ทำให้ขนาดโปรแกรมของหน่วยประมวลผลไวไม่เพิ่มขึ้นน่าจะเกิดจากกระบวนการบรรจุคำสั่งของโปรแกรมที่อธิบายไว้ในหัวข้อที่ 2.3 ที่ได้ประสิทธิภาพดังจะเห็นได้จากผลในตารางที่ 4 ว่าหน่วยความจำที่เสียไปจากการจัดเรียงคำสั่งเฉลี่ยมีเพียง 12% เท่านั้น นอกจากนั้นปัจจัยอีกอย่างน่าจะเกิดจากการเพิ่มคำสั่งเข้าไปในชุดคำสั่งดังที่อธิบายในหัวข้อ 2.2 ซึ่งทำให้ตัวแปลภาษาสามารถเลือกใช้คำสั่งที่มีขนาดและหน้าที่เหมาะสมกับการทำงานได้มากขึ้น

5. สรุป

บทความนี้นำเสนอแนวคิดในการออกแบบหน่วยประมวลผลไว ซึ่งออกแบบโดยยึดจากการออกแบบของหน่วยประมวลผล SMC ภายในการออกแบบหน่วยประมวลผลไวใช้หลักการในการบรรจุคำสั่งรหัสไบต์หลายคำสั่งลงในหน่วยคำสั่งขนาด 32 บิตเพื่อแก้ปัญหาสมรรถนะและขนาดของโปรแกรมที่ต้องใช้ชุดคำสั่งขนาด 32 บิต

จากการทดลองวัดผลเปรียบเทียบระหว่างหน่วยประมวลไวและหน่วยประมวลผล SMC พบว่าสมรรถนะการทำงานของหน่วยประมวลผลไวเพิ่มขึ้น 2.12 เท่าของหน่วยประมวลผล SMC ในขณะที่ขนาดของโปรแกรมที่ทำงานบนหน่วยประมวลผลไวไม่ได้ใหญ่ขึ้นกว่าของหน่วยประมวลผล SMC

6. สิ่งที่จะทำต่อไป

ในบทความกล่าวถึงเพียงแค่แนวคิดที่ใช้ในการออกแบบหน่วยประมวลผลไวเท่านั้น ยังไม่ได้พัฒนาหน่วยประมวลผลดังกล่าวให้สามารถทำงานได้จริง จึงยังไม่สามารถประกันได้ว่าการออกแบบทั้งหมดนั้นจะสามารถทำงานได้อย่างถูกต้อง อีกทั้งยังไม่สามารถพิจารณาได้ว่าหน่วยประมวลผลดังกล่าวนั้นใช้ทรัพยากรมากน้อยเพียงใด ดังนั้นสิ่งที่จะทำต่อไปคือการพัฒนาให้หน่วยประมวลผลดังกล่าวสามารถทำงานได้จริงบน FPGA และวัดผลการสังเคราะห์ว่าหน่วยประมวลผลใช้ทรัพยากรมากน้อยเพียงใด

นอกจากนั้นในการออกแบบหน่วยประมวลผลไวในขณะนี้นั้น ได้ออกแบบให้มีการทำแสตกแคชชิ่ง (Stack caching) ซึ่งจะเก็บข้อมูลบนแสตกไว้หนึ่งตัวในเรจิสเตอร์เพื่อเพิ่มสมรรถนะในการประมวลผล ต่อไปจะทดลองเพิ่มจำนวนแสตกแคชชิ่งขึ้นเพื่อเพิ่มสมรรถนะการทำงานของหน่วยประมวลผลขึ้นอีก

7. รายการอ้างอิง

[1] ดร. อิทธิ ฤทธาภรณ์. ไมโครชิพของไทยไปถึงไหนแล้ว ใช้กันสุดฤทธิ์ ผลิตได้บ้างหรือยัง [Microsoft Power Point]. ศูนย์เทคโนโลยีไมโครอิเลคทรอนิกส์ (TMEC). แหล่งที่มา: http://tmec.nectec.or.th [เข้าถึงวันที่ 1 มีนาคม 2548].

[2] เจนวิทย์ ศรีหารักษา, สุวิชา จิรายุเจริญศักดิ์, จันทิรา เจือกโว้น และชำนาญ ปัญญาใส. การพัฒนาชิพ 8051 High Speed Microcontroller. กรุงเทพมหานคร: ห้องปฏิบัติการไมโครอิเล็กทรอนิกส์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ. (ม.ป.ป.).

[3] Rittaporn, I., Punyasai, C. and Siamchai, P. (2000). “Progress Report on NECTEC's Microelectronics Project” NECTEC Technical Journal. Vol.2, No.7.

[4] รวิวร มะหะสิทธิ์ และ วันเฉลิม โปรา. (2546). การออกแบบวงจรรวมหน่วยประมวลผลสัญญาณดิจิทัลที่มีตัวกรองเอฟไออาร์. งานประชุมวิชาการวิศวกรรมไฟฟ้าครั้งที่ 26 (EECON' 26). สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ, 6 – 7 พฤศจิกายน 2546

[5] คณิตพงศ์ เพ็งวัน. (2545). การออกแบบวงจรรวมของไมโครคอนโทรลเลอร์ขนาด 16 บิตสำหรับเครื่องรับโทรทัศน์. วิทยานิพนธ์ปริญญามหาบัณฑิต. ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย.

[6] Burutarchanai, A., Nanthanavoot, P., Aporntewan, C., and Chongstitvatana, P. (2004). A Stack-Based Processor for Resource Efficient Embedded Systems. Proc. of IEEE TENCON 2004, 21-24 November 2004, Thailand.

[7] Burutarchanai, A., Kotrajaras, V. and Chongstitvatana, P. (2004). A Fast Instruction Fetch Unit for an Embedded Stack Processor. Proc. of Int. Conf. on Information and Communication Technologies (ICT 2004), 18-19 November, 2004. Thailand.

[8] Burutarchanai, A., and Chongstitvatana, P. (2004). Design of a Two-Phased Clocked Control Unit for Performance Enhancement of a Stack Processor. National Computer Science and Engineering Conference, Thailand, 21-22 Sept. 2004, pp.114-119.