บทที่ 1
บทนำ

ไมโครโพรเซสเซอร์ (Microprocessor) ถือเป็นวิวัฒนาการทางเทคโนโลยีที่เด่นที่สุดในรอบ 30 ปี หลังจากทรานซิสเตอร์ซึ่งปรากฏต่อชาวโลกเมื่อปี พ.ศ. 2491 เป็นต้นมา กล่าวกันว่าไมโครโพรเซสเซอร์ ไม่เพียงแต่จะทำให้เกิดการเปลี่ยนแปลงแนวความคิดครั้งยิ่งใหญ่ของวงจรอิเล็กทรอนิกส์เชิงเลข (Digital Electronics) เท่านั้น แต่จะมีผลกระทบกระเทือนต่อชีวิตความเป็นอยู่ของมนุษย์ชาติในปัจจุบันและอนาคต ดังที่ได้เห็นปัญหาในเรื่องของ Y2K ที่ทำให้ผู้ใช้คอมพิวเตอร์ทั่วโลกรู้สึกวิตกกังวลกันว่าเมื่อถึง ค.ศ. 2000 หรือ พ.ศ. 2543 จะเกิดอะไรขึ้นบ้าง

ไมโครโพรเซสเซอร์ตัวแรกถูกสร้างขึ้นในปี พ.ศ. 2514 โดยบริษัทอินเทล คอร์พอเรชั่น (Intel Corporation) ได้ออกแบบไมโครโพรเซสเซอร์ขนาด 4 บิต มีชื่อว่า อินเทล 4004 ซึ่งจุดประสงค์ในการสร้างในครั้งนั้น เพื่อประยุกต์ใช้ในเครื่องคำนวณอิเล็กทรอนิกส์ขนาดเล็ก ต่อมาได้มีบริษัทผู้ผลิตอุปกรณ์สารกึ่งตัวนำหลายบริษัททำการผลิตไมโครโพรเซสเซอร์ชนิดต่างๆ ขึ้นมาอีกเป็นจำนวนมาก

จนกระทั่งปี พ.ศ. 2523 บริษัท IBM โดย John Cocke ได้ออกแบบไมโครโพรเซสเซอร์รุ่น IBM 801 โดยใช้สถาปัตยกรรมแบบใหม่ที่มีชื่อว่า RISC (Reduced Instruction Set Computer) ซึ่งแตกต่างไปจากสถาปัตยกรรมแบบ CISC (Complex Instruction Set Computer) ตรงที่การใช้ชุดคำสั่งแบบง่ายๆ ทำให้จำนวนชุดคำสั่งที่ใช้ในการประมวลผลมีน้อยลงและกำหนดให้แต่ละคำสั่งมีความยาวเท่ากัน นอกจากนี้ยังเพิ่มจำนวนรีจิสเตอร์จาก 8 รีจิสเตอร์เป็น 32 รีจิสเตอร์ มีเฉพาะคำสั่ง Load และ Store เท่านั้นที่สามารถติดต่อกับหน่วยความจำได้ ผลก็คือขนาดของวงจรที่ได้จะเล็กลง และย่นระยะเวลาที่ใช้ในการออกแบบและปรับปรุงด้วย

ปัญหาที่สำคัญอันหนึ่งของการพัฒนาขีดความสามารถของเครื่องคอมพิวเตอร์ก็คือ ความล่าช้าในการติดต่อกับหน่วยความจำ ปัญหานี้ถูกบรรเทาโดยการแทรกแคช (cache) ขั้นไว้ระหว่างหน่วยประมวลผลกับหน่วยความจำหลัก (main memory) แคชเป็นหน่วยความจำที่มีความเร็วรองจากรีจิสเตอร์และมีความเร็วสูงกว่าหน่วยความจำหลักมาก กล่าวคือ การติดต่อกับหน่วยความจำหลักต้องใช้เวลาในการติดต่อนานถึง 4-20 เท่าของการติดต่อกับแคช ดังนั้นแคชจึงเป็นหน่วยความจำที่สำคัญสำหรับการเพิ่มความเร็วในการประมวลผลให้กับเครื่องคอมพิวเตอร์

1.1 ความสำคัญและความเป็นมาของปัญหา

คำสั่ง load และคำสั่ง store เป็นคำสั่งที่มีความสำคัญ ซึ่งในการประมวลผลโปรแกรมหนึ่งๆ จะมีการประมวลผลคำสั่ง load และคำสั่ง store ประมาณ 1 ใน 3 ของการประมวลผลทั้งหมด โดยคำสั่ง load เป็นคำสั่งที่ใช้อ่านค่าจากหน่วยความจำหลักมาเก็บไว้ในรีจิสเตอร์ และคำสั่ง store ใช้ในการเขียนค่าจากรีจิสเตอร์ลงใน

หน่วยความจำหลัก ซึ่งทั้งสองคำสั่งถูกประมวลผลในหน่วยโหลด/สโตร์ ถึงแม้ว่าจะมีการเพิ่มหน่วยความจำในส่วนของแคชเข้ามาแล้ว แต่การประมวลผลคำสั่งดังกล่าวก็ยังเกิดความล่าช้า อันเนื่องมาจากข้อมูลที่ต้องการอ้างถึงไม่ได้อยู่ในแคช ดังนั้นถ้าหากเราสามารถลดความล่าช้าในส่วนนี้ลงไปได้ ก็จะส่งผลให้การประมวลผลของไมโครโพรเซสเซอร์มีความเร็วมากขึ้น

งานวิจัยนี้จึงมุ่งเน้นในการพัฒนาประสิทธิภาพของหน่วยโหลด/สโตร์ โดยเฉพาะอย่างยิ่งในส่วนของการลดการเกิด stall pipeline อันเป็นสาเหตุของความล่าช้าในการติดต่อกับแคช (เกิดขึ้นได้ถ้าหากมีการติดต่อแคชมากกว่า 1 process) และหน่วยความจำหลัก

1.2 วัตถุประสงค์

ออกแบบ LSU (Load / Store Unit) ซึ่งมีหน้าที่ใช้ในการติดต่อและควบคุมการทำงานของ Data Cache และศึกษาสมรรถนะของการจัด Cache ในแบบ Direct-Mapped และ 2-way Set Associative ที่ขนาด 1K, 2K, 4K และ 8K bytes และนำไปตรวจสอบการเปลี่ยนแปลงค่า miss rate กับสภาพแวดล้อมต่างๆ รวมไปถึงการหาขนาดของบัฟเฟอร์ที่พอเหมาะกับการใช้งานภายใน LSU

1.3 ขอบเขตของการวิจัย

1) การวิจัยนี้จะทำการออกแบบวงจร LSU (Load / Store Unit) โดยใช้ภาษา Verilog เพื่อจำลองวงจรในระดับ gate และตรวจสอบการทำงาน
2) ภายใน LSU เป็น pipeline 4 stages ได้แก่ R, X, C และ W
3) ผู้ใช้สามารถเลือกได้ว่า ต้องการจัดรูปแบบของ cache เป็นแบบ Direct-mapped หรือ 2-way set associative ใช้การแทนที่เป็นแบบ Random
4) ผู้ใช้สามารถเลือกได้ว่า ขนาดของ cache เป็น 1 K, 2 K, 4 K, หรือ 8 Kbytes
5) ผู้ใช้สามารถเลือกได้ว่าขนาดของบัฟเฟอร์ในส่วนของ SHB เป็น 1 หรือ 2 บัฟเฟอร์
6) ผู้ใช้สามารถเลือกได้ว่าขนาดของ FIFO ในส่วนของ SMC เป็น 2, 4 หรือ 8 word / doubleword
7) Addressing เป็นแบบ little-endian
8) ผู้ใช้สามารถเลือก mode ได้ว่าเป็น write through หรือ write back

1.4 ขั้นตอนการทำวิจัย

1) ศึกษาการทำงานของระบบการควบคุมการทำงานของ LSU ทั้งหมด
2) ศึกษาการเขียนภาษา Verilog เพื่อการออกแบบวงจร digital
   - ศึกษาการเขียน Verilog Hardware Description Language
   - ศึกษาการใช้งาน Verilog-XL, Cwaves (simulators) และ tool อื่นๆ

3) วิเคราะห์และออกแบบ LSU โดยแบ่งการทำงานออกเป็นมอดูลย่อย ๆ เพื่อที่จะสามารถตรวจสอบความถูกต้อง ได้จากการ simulate ดูแต่ละมอดูล แล้วจึงนำมอดูลต่างๆ เหล่านั้นมารวมกันเป็นวงจรขนาดใหญ่
4) ใช้ภาษา C ในการสร้างสัญญาณอินพุต ได้แก่ คำสั่ง 32 บิตและสัญญาณควบคุมต่างๆ เพื่อใช้ในการทดสอบการทำงานของวงจร LSU ที่ออกแบบ โดยภาษา C ที่เขียนขึ้นมี random program สามารถสร้าง random input ให้ LSU เป็นล้านๆ cycle ได้ และนำผลการทดลองที่ได้เก็บลงไฟล์
5) ใช้ภาษา C ในการจำลองการทำงานของวงจร LSU เพื่อเอา output vector มาเปรียบเทียบกับ output vector ที่ได้จากการ simulate ของวงจรที่ออกแบบตามที่ทดสอบไว้ในข้อ 4 เพื่อเป็นการยืนยันว่าวงจรทำงานได้ถูกต้อง
6) วัดสมรรถนะของระบบ เมื่อมีการเปลี่ยนแปลง environment ต่างๆ เพื่อให้ทราบว่า miss rate จะมีการเปลี่ยนแปลงอย่างไร และวัดประสิทธิภาพในเรื่องความเร็วในการทำงานกับการเปลี่ยนแปลงขนาดของบัฟเฟอร์ภายใน LSU ซึ่งได้แก่ขนาดของ SHB (Store Hit Buffer) และ ขนาดของ FIFO (First-In-First-Out) ในส่วนของ SMC(Store Miss Control)
7) จัดทำเอกสาร และเขียนวิทยานิพนธ์ โดยมีเนื้อหาดังนี้
   - หลักการทำงานของ LSU
   - รายงานผลของระบบ เมื่อพารามิเตอร์เปลี่ยนแปลงไป
   - รายงานผลของระบบ เมื่อขนาดของบัฟเฟอร์เปลี่ยนแปลงไป

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) <u>ด้านวิชาการ</u> ข้อมูลที่ได้จากการปรับพารามิเตอร์ เช่น ขนาดของบัฟเฟอร์ จะทำให้ผู้ออกแบบตัวประมวลผลสามารถตัดสินใจเลือกจุดที่เหมาะสำหรับ ราคา-สมรรถนะ ของหน่วยโหลด/สโตร์ หนึ่งๆ ได้
2) <u>ด้านการศึกษา</u> สามารถใช้เป็นแบบอย่างในการออกแบบวงจรรวมขนาดใหญ่ได้ ซึ่งการออกแบบวงจรรวมขนาดใหญ่ในประเทศไทยนี้ ยังอยู่ในช่วงเริ่มต้น การผลิตผลงานในแนวนี้ออกมาย่อมช่วยให้มีผู้ศึกษาเพิ่มเติม ทำให้มีปริมาณคนในการพัฒนางานวิจัยทางด้านนี้ได้มีการพัฒนาอย่างรวดเร็วขึ้น และเป็นส่วนหนึ่งของการพัฒนาโครงงานทางด้านนี้ในระดับมหาวิทยาลัยอีกด้วย
3) <u>สำหรับตนเอง</u> ทำให้มีความรู้ความสามารถในการออกแบบวงจรรวมขนาดใหญ่ และประสบการณ์ในการใช้งานภาษาอธิบายลักษณะการทำงานของวงจรทางไฟฟ้า ในระบบดิจิทัล โดยในงานวิจัยนี้ใช้ภาษา Verilog

1.6 งานวิจัยที่เกี่ยวข้อง

K. I. Farkas, N. P. Jouppi, P. Chow (1995) ได้เสนอผลงานวิจัยในหัวข้อ *"How Useful Are Non-blocking Loads, Stream Buffers and Speculative Execution in Multiple Issue Processor?"* เป็นการวิจัยโดยการนำเทคนิคพิเศษ 3 อย่างซึ่งได้แก่ การทำ Non-blocking Loads, Stream Buffers และ Speculative

Execution มาใช้ร่วมกัน โดยเทคนิกการทำ Non-blocking Loads คือการยอมให้คำสั่งอื่นๆ สามารถทำงานต่อไปในขณะที่ประมวลผลคำสั่งที่เกิด Load Miss อยู่ ซึ่งคำสั่งที่จะประมวลผลได้ต้องไม่ใช้ Register ที่รอข้อมูลจากการ Load ทั้งนี้เพื่อลดการเกิด Stall Pipeline ในช่วงเวลาดังกล่าว ส่วนเทคนิกการทำ Stream Buffers เป็นการออกแบบ FIFO จำนวนหนึ่งที่แต่ละ FIFO Entry จะเก็บ Data และ Address เอาไว้ เมื่อมีการติดต่อกับ Cache ก็จะมีการตรวจสอบทุกๆ Entry นี้ในเวลาเดียวกัน ดังนั้นถ้าหากเกิด Cache Miss ขึ้นก็สามารถนำข้อมูลมาจาก FIFO นี้ได้ ผลปรากฏว่าการทำ Non-blocking Loads ช่วยลดเวลาที่ใช้ในการทำงานลงได้ 21% และการทำ Stream Buffers ช่วยลดเวลาในการทำงานลงได้ 26% และเมื่อนำสองเทคนิกทั้งสองมารวมกันจะช่วยลดเวลาลงได้ 47% และการทำ Speculative Execution เพียงอย่างเดียวช่วยเพิ่มสมรรถนะการทำงานได้ 20% และเมื่อรวมกันทั้งสามเทคนิกช่วยเพิ่มสมรรถนะการทำงานได้ถึง 40%

M.J. Charney และ T.R. Puzak(1997) ได้เสนอผลงานวิจัยในหัวข้อ *"Prefetching and memory system behavior of the SPEC95 benchmark suit"* โดยมีการวัดค่า Cache Miss Rate ทั้งบน Instruction Cache และ Data Cache สำหรับ SPEC95 Benchmark จำนวน 18 โปรแกรม โปรแกรมละ 500 ล้านคำสั่ง ซึ่งมีเพียงไม่กี่ Application เท่านั้นที่ส่งผลให้ค่า Miss Rate สูง และเป็นที่น่าสังเกตว่า ถ้าขนาดของ Instruction Cache เป็น 32KB ก็สามารถทำให้ Miss Rate ต่ำกว่า 0.1% (หรือเกิด Miss หนึ่งครั้งทุกๆ 1000 คำสั่ง) นอกจากนี้ยังได้วิเคราะห์ 2 Prefetching Algorithm ได้แก่ Next-Sequential Prefetching (NSP) และ Shadow-Directory Prefetching (SDP) ผลที่ได้แบ่งเป็น Instruction Prefetching และ Data Prefetching สำหรับ Instruction Prefetching นั้น วิธี SDP ให้ค่าเฉลี่ยของ Coverage สูงกว่าวิธี NSP 10% โดยค่า Coverage ของวิธี NSP และวิธี SDP เท่ากับ 60% และ 70% ตามลำดับ และค่า Accuracy ของทั้งสองวิธีเท่ากับ 90% ส่วน Data Prefetching ได้ค่า Coverage และ Accuracy ต่ำกว่า Instruction Prefetching โดยค่า Coverage ของวิธี NSP และวิธี SDP เท่ากับ 25% และ 40% ตามลำดับ และค่า Accuracy ของทั้งสองวิธีสูงกว่า 70%

1.7 ลำดับขั้นตอนในการเสนอผลงานวิจัย

ในบทที่ 2 จะเป็นการนำเสนอหลักการทำงานของ LSU เพื่อเป็นแนวทางในการออกแบบต่อไป ในบทที่ 3 กล่าวถึงไมโครโพรเซสเซอร์ที่ถูกออกแบบ เพื่อใช้เป็นองค์ประกอบเสริมในการวัดสมรรถนะของ LSU ที่ทำการพัฒนาขึ้น จากนั้น บทที่ 4 เสนอวิธีการออกแบบ LSU เพื่อช่วยลดการเกิดการ stall pipeline ในบทที่ 5 กล่าวถึงวิธีการที่ใช้ในการตรวจสอบความถูกต้องของวงจร และวิธีการที่ใช้สร้าง benchmark และการนำ benchmark มาใช้ ในบทที่ 6 กล่าวถึงผลการวัดสมรรถภาพของ LSU ที่ได้ออกแบบขึ้น และในบทที่ 7 เป็นบทสรุปของงานวิจัยนี้

1.8 ผลงานที่ตีพิมพ์จากงานวิจัย

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์เป็นบทความวิชาการ และนำเสนอในที่ประชุมวิชาการ The Second Annual National Symposium on Computational Science and Engineering (ANSCE'98) เมื่อวันที่ 26-27 มีนาคม พ.ศ. 2541 ในชื่อ A VLSI Design of a Load/Store Unit for a RISC Processor โดยผู้นำเสนอคือ Primas Taechashong และ Prabhas Chongstitvatana