การปรับปรุงประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลสาร์โปรเซสเซอร์ด้วยวิธีการไดนามิกสเคจดูล

PERFORMANCE IMPROVEMENT OF DLX SUPERSCALAR PROCESSOR USING DYNAMIC SCHEDULING

จักรพันธุ์ วชิรภานนท์

JAKRAPUN WACHIRAPANON

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2545

ISBN 974-648-617-9

PERFORMANCE IMPROVEMENT OF DLX SUPERSCALAR PROCESSOR USING DYNAMIC SCHEDULING

JAKRAPUN WACHIRAPANON

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2002

ISBN 974-648-617-9

หัวข้อวิทยานิพนธ์	การปรับปรุงประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ด้วยวิธีการไดนามิกสเคจดูล
นักศึกษา	นาย จักรพันธุ์ วชิรภานนท์
รหัสประจำตัว	39061081
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2545
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ. บรรจง ปียธำรง

บทคัดย่อ

ซูเปอร์สเกลลาร์โปรเซสเซอร์เป็นโปรเซสเซอร์ที่มีความสามารถในการเฟ็ทช์ ถอดรหัส และประมวลผลคำสั่งได้หลายคำสั่งในหนึ่งไซเคิล โดยโปรเซสเซอร์จะต้องมีกรรมวิธีในการจัดหาหรือจัดเรียงคำสั่งใหม่ให้เหมาะสมเพื่อให้การประมวลผลมีประสิทธิภาพ และต้องมีกระบวนการแก้ปัญหาข้อมูลที่เกิดขึ้นระหว่างการประมวลผลเพื่อให้การประมวลผลข้อมูลถูกต้องและรวดเร็ว โปรเซสเซอร์แบบซูเปอร์สเกลลาร์ดำเนินกรรมวิธีดังกล่าวในช่วงขณะโปรแกรมกำลังทำงาน ซึ่งเรียกว่าวิธีการไดนามิกสเคจดูล ถ้ากระบวนการนี้ไม่มีประสิทธิภาพจะเกิดปัญหาคอขวดขึ้นเนื่องจากเฟ็ทช์คำสั่งเข้ามาจำนวนมากแต่คำสั่งที่ประมวลผลเสร็จมีจำนวนน้อยกว่า งานวิจัยนี้ได้ศึกษาและหาวิธีการทำไดนามิกสเคจดูลเพื่อนำมาปรับปรุงประสิทธิภาพด้านการประมวลผลของซูเปอร์สเกลลาร์โปรเซสเซอร์ โดยเลือกดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ซึ่งเดิมมีการทำไดนามิกสเคจดูลแบบสกอร์บอร์ดเป็นต้นแบบ ในงานวิจัยได้ดำเนินการปรับปรุงการทำไดนามิกสเคจดูล โดยออกแบบให้มีบัฟเฟอร์สำหรับเก็บคำสั่ง 1 ชุด คิววงกลม 1 ชุดในการเก็บลำดับของคำสั่งที่ประมวลผล อัลกอริทึมในการเลือกชุดคำสั่ง และแก้ปัญหาข้อมูลด้วยหลักการเปลี่ยนชื่อริจิสเตอร์ และเพิ่มประสิทธิภาพการทำงานของคำสั่งโหลดและสตอร์โดยการเพิ่มหน่วยทำงานสตอร์บัฟเฟอร์ จำลองการทำงานของโปรเซสเซอร์ทั้งสองด้วยภาษาบรรยายฮาร์ดแวร์วีเอชดีแอล ทดสอบการทำงานของซูเปอร์สเกลลาร์ดีแอลเอ็กซ์ด้วยโปรแกรมทดสอบที่เหมาะสม จากนั้นเปรียบเทียบประสิทธิภาพด้านการประมวลผลระหว่างโปรเซสเซอร์แบบเดิมกับแบบที่ปรับปรุง ผลการทดสอบปรากฏอย่างชัดเจนว่าหลักการที่นำเสนอสามารถทำให้ดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์มีประสิทธิภาพดีขึ้นโดยพิจารณาจากจำนวนคำสั่งต่อไซเคิลที่เพิ่มขึ้น

Thesis Title	Performance Improvement of DLX Superscalar Processor Using Dynamic Scheduling
Student	Mr. Jakrapun Wachirapanon
Student ID.	39061081
Degree	Master of Engineering in Electrical Engineering
Programme	Electrical Engineering
Year	2002
Thesis Advisor	Assoc.Prof.Bunjong Piyatamrong

ABSTRACT

Superscalar processor is the effective processor; it can fetch, decode and execute more instructions per clock cycle. It uses hardware for dynamic scheduling not only that to procure proper instructions or reorder the instructions before executing and it has solution for data dependency. If the dynamic scheduling is not work, it will be happen a bottleneck problem because more fetched instructions than completed executing. This research presents the dynamic scheduling improvement. DLX superscalar processor with dynamic scheduling called scorebord is the model for research. The buffer for keeping one set of instruction, one set of curricular queue for keeping the order of execution instruction, algorithm for searching proper instruction, resolve data dependency with register renaming and append the store buffer for load-store instruction are created to form dynamic scheduling improvement in the research. Using VHDL hardware language to simulate both scorebording and new propose. Test with appropriated programs and compare the result. It is clear that, instruction per cycle of dynamic scheduling improvement is more than scorebording.

กิตติกรรมประกาศ

ผู้เขียนขอขอบคุณผู้ที่มีส่วนในงานวิจัยนี้ อาทิ รศ. บรรจง ปิยธำรง อาจารย์ที่ปรึกษาผู้ให้คำแนะนำเกี่ยวกับวิทยานิพนธ์เป็นอย่างดี คุณพัชรินทร์ กลิ่นซ้อน ผู้ช่วยเหลือค้นคว้าข้อมูลและจัดทำเอกสารรูปเล่มวิทยานิพนธ์ คุณแม่ฐิติมา พญ. วรัฐฐา ทญ. หรรษมลล์ น้องสาวทั้งสองผู้ให้กำลังใจเสมอ และมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร (C&C) ที่ได้มอบทุนสนับสนุนการวิจัย

งานวิจัยนี้อาจมีข้อผิดพลาดอยู่บ้างซึ่งผู้เขียนถือว่าผู้อ่านที่พบข้อผิดพลาดดังกล่าวเป็นผู้ที่สนใจงานวิจัยนี้อยู่พอสมควรและอาจนำไปสู่การพัฒนางานวิจัยนี้ให้มีความสมบูรณ์ยิ่งขึ้น

จักรพันธุ์ วชิรภานนท์

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของการศึกษา

การพัฒนาโปรเซสเซอร์ให้มีความสามารถในการประมวลผลเพิ่มขึ้นมีหลายวิธี ช่วงเวลาหลายสิบปีที่ผ่านมาการเพิ่มขีดความสามารถของโปรเซสเซอร์ได้ใช้วิธีการเพิ่มขีดความสามารถของคำสั่งโดยคำสั่งจะมีการทำงานเพิ่มขึ้นและซับซ้อนขึ้น ทำให้การทำงานของแต่ละคำสั่งมีจำนวนไซเคิลไม่แน่นอน แนวความคิดนี้คือโปรเซสเซอร์ในกลุ่มของ CISC (Complex Instruction Set Computer) หลังจากนั้นในช่วงต้นปี 1980 ก็ได้มีอีกแนวความคิดหนึ่งที่ต้องการให้โปรเซสเซอร์ทำงานเร็วขึ้น โดยพยายามให้แต่ละคำสั่งทำงานด้วยไซเคิลที่แน่นอนโดยลดความซับซ้อนของคำสั่งลงให้เป็นคำสั่งพื้นฐานมากที่สุด แล้วใช้หลักการของไปป์ไลน์มาประยุกต์ใช้โดยการนำคำสั่งพื้นฐานเหล่านั้นมาแบ่งการทำงานออกเป็นไปป์สเตจหรือสเตจ(Pipe Stage) ย่อย จากนั้นคำสั่งแต่ละคำสั่งจะเข้าทำงานตามไปป์สเตจตามลำดับเรียงกันไปโดยที่ทุกไปป์สเตจจะมีการทำงานตลอดเวลาซึ่งแนวความคิดดังกล่าวคือโปรเซสเซอร์ในแบบของ RISC (Reduce Instruction Set Computer) ในปัจจุบัน RISCโปรเซสเซอร์สามารถประมวลผลได้ใกล้ประสิทธิภาพสูงสุดคือ 1 คำสั่งต่อไซเคิล

ถัดจากยุคของ RISC นักออกแบบโปรเซสเซอร์ได้หันมาสนใจการเพิ่มขีดความสามารถของ RISC โดยพยายามหาวิธีการประมวลผลให้มากกว่า 1 คำสั่งต่อไซเคิล ซึ่งทำได้โดยการเฟ็ทซ์ ถอดรหัส และประมวลผลคำสั่งหลายๆ คำสั่งพร้อมกันได้ในหนึ่งไซเคิลการทำงาน จากหลักการพัฒนาดังกล่าวทำให้เกิดสถาปัตยกรรมคอมพิวเตอร์ขึ้น 2 รูปแบบคือ VLIW (Very Long Instruction Word) และ ซูเปอร์สเกลลาร์ (Superscalar) [1] การจะประมวลผลคำสั่งหลายคำสั่งได้พร้อมกันนั้นต้องทำให้คำสั่งที่ส่งออกหลายคำสั่งนั้นมีความอิสระจากกันคือคำสั่งหนึ่งในหลายคำสั่งที่ส่งออกพร้อมกันนั้นจะต้องไม่รอผลลัพธ์จากคำสั่งหนึ่งคำสั่งใดในหลายคำสั่งนั้น ซึ่งการกระทำดังกล่าวทำให้เกิดความแตกต่างระหว่างสถาปัตยกรรมทั้งสอง ในสถาปัตยกรรมแบบ VLIW จะใช้คอมไพเลอร์ที่มีความสามารถสูงเพื่อหาคำสั่งที่จะส่งออกไปประมวลผลและแก้ปัญหาข้อมูลในช่วงเวลาคอมไพล์ เรียกว่าวิธีการสแตติกสเคจดูล (Static Scheduling) และเมื่อเลือกชุดคำสั่งได้แล้วจะรวมชุดคำสั่งนั้นให้เป็นคำสั่งขนาดใหญ่เพียงคำสั่งเดียว และส่งคำสั่งออกไปประมวลผล โดยฮาร์ดแวร์จะทำการถอดรหัสคำสั่งนั้นออกเป็นคำสั่งย่อยเพื่อให้แต่ละคำสั่งประมวลผลได้พร้อมกัน ข้อดีของวิธีนี้ทำให้การสร้างฮาร์ดแวร์ง่ายขึ้นแต่ข้อเสียคือคอมไพเลอร์ต้องมีความซับซ้อนมาก สำหรับอีกสถาปัตยกรรมหนึ่งคือซูเปอร์สเกลลาร์จะแตกต่างกับ VLIW คือกระบวนการเลือกชุดคำสั่งเพื่อส่งออกประมวลผลและกระบวนการแก้ปัญหาข้อมูลจะกระทำ

โดยฮาร์ดแวร์ทั้งหมดซึ่งเรียกว่าวิธีการไดนามิกสเคจดูล (Dynamic Scheduling) ซึ่งหลักการนี้จะเป็นหัวใจสำคัญของการออกแบบซูเปอร์สเกลลาร์โปรเซสเซอร์ ยิ่งขบวนการวิธีนี้มีประสิทธิภาพมากเท่าใด ความสามารถในการประมวลผลแบบซูเปอร์สเกลลาร์โปรเซสเซอร์จะมีมากขึ้นเท่านั้น ดังนั้นการออกแบบการทำไดนามิกสเคจดูลให้มีประสิทธิภาพจึงเป็นสิ่งที่สำคัญมากสิ่งหนึ่งสำหรับการออกแบบซูเปอร์สเกลลาร์โปรเซสเซอร์

1.2 วัตถุประสงค์ของงานวิจัย

- ศึกษาและเข้าใจลักษณะของสถาปัตยกรรมซูเปอร์สเกลลาร์โปรเซสเซอร์
- เข้าใจปัญหาที่เกิดขึ้นในซูเปอร์สเกลลาร์โปรเซสเซอร์
- ศึกษาหลักการของการทำไดนามิกสเคจดูลและการทำไดนามิกสเคจดูลด้วยวิธีสกอร์บอร์ด
- สามารถปรับปรุงวิธีการทำไดนามิกสเคจดูลให้มีประสิทธิภาพได้
- ศึกษาโครงสร้างของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์
- สามารถนำหลักการการทำไดนามิกสเคจดูลมาปรับปรุงประสิทธิภาพด้านการประมวลผลของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ได้

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้เป็นการปรับปรุงประสิทธิภาพด้านการประมวลผลของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โดยการปรับปรุงการทำงานในส่วนการทำไดนามิกสเคจดูลให้มีประสิทธิภาพยิ่งขึ้น ดังนั้นต้องสร้างโปรเซสเซอร์ต้นแบบของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ขึ้นมาก่อน แล้วจึงทำการปรับปรุงในส่วนของการทำไดนามิกของสเคจดูลเพื่อเพิ่มประสิทธิภาพด้านการประมวลผล จากนั้นจึงเปรียบเทียบประสิทธิภาพที่ได้ก่อนและหลังการปรับปรุงการทำไดนามิกสเคจดูล สรุปขั้นตอนการทำงานในขอบข่ายของงานวิจัยได้ดังนี้

1. กำหนดขอบเขตของโครงสร้างต้นแบบของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์
2. จำลองโครงสร้างต้นแบบของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ ซึ่งใช้การทำไดนามิกสเคจดูลตามวิธีสกอร์บอร์ดด้วยภาษาบรรยายฮาร์ดแวร์วีเอชดีแอล
3. ปรับปรุงโครงสร้างต้นแบบของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ในส่วนของการทำไดนามิกสเคจดูลที่นำเสนอ
4. ทดสอบประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ก่อนและหลังการปรับปรุงการทำไดนามิกสเคจดูลที่นำเสนอด้วยโปรแกรมทดสอบ

5. เปรียบเทียบประสิทธิภาพด้านการประมวลผล ทั้งก่อนและหลังการปรับปรุงการทำไดนามิกสเคจดูล

1.4 รายละเอียดของวิทยานิพนธ์

ภายใต้ขอบข่ายของงานวิจัย วิทยานิพนธ์นี้ต้องการเรียงลำดับเนื้อหาอย่างต่อเนื่องเพื่อให้ครอบคลุมเนื้อหาทั้งหมดที่เกี่ยวข้องอย่างเหมาะสม อย่างไรก็ตามในบางเรื่องอาจจะไม่กล่าวถึง หรืออาจรวบรัดเนื่องจากมิใช่สาระสำคัญของงานวิจัยและเป็นส่วนความรู้พื้นฐานของผู้ที่อยู่ในวงการคอมพิวเตอร์มักคุ้นเคยเป็นอย่างดี อาทิ ความรู้เบื้องต้นเกี่ยวกับการเขียนโปรแกรมด้วยภาษาวีเอชดีแอล สำหรับรายละเอียดของวิทยานิพนธ์ประกอบด้วยเนื้อหาดังต่อไปนี้

บทที่ 2 กล่าวถึงซูเปอร์สเกลลาร์โปรเซสเซอร์ หลักการพื้นฐานของซูเปอร์สเกลลาร์โปรเซสเซอร์ ข้อจำกัดด้านประสิทธิภาพ และหลักการของไดนามิกสเคจดูล

บทที่ 3 กล่าวถึงโครงสร้างของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ที่ใช้งานในงานวิจัย

บทที่ 4 กล่าวถึงการทำไดนามิกสเคจดูลตามวิธีสกอร์บอร์ด และตัวอย่างการประมวลผลคำสั่ง

บทที่ 5 กล่าวถึงวิธีไดนามิกสเคจดูลที่นำเสนอ ประกอบด้วยรายละเอียดตามหลักการทุกขั้นตอนและตัวอย่างการประมวลผลคำสั่ง

บทที่ 6 กล่าวถึงการทดสอบ เปรียบเทียบ และวิเคราะห์ประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ระหว่างวิธีเดิมและวิธีไดนามิกสเคจดูลที่นำเสนอ

บทที่ 7 เป็นบทสรุปของงานวิจัย ปัญหาและข้อเสนอแนะ

อนึ่ง รายละเอียดสามารถค้นคว้าเพิ่มเติมได้จากเอกสารอ้างอิง

บทที่ 2
ซูเปอร์สเกลลาร์กับการทำไดนามิกสเคจดูล

ความพยายามในการพัฒนาศักยภาพการทำงานของคอมพิวเตอร์ให้รองรับระบบการทำงานที่ซับซ้อนมากขึ้นในปัจจุบันทำให้ต้องปรับปรุงโปรเซสเซอร์ให้มีความสามารถมากขึ้น ซึ่งส่งผลให้โปรเซสเซอร์มีความซับซ้อนขึ้น เนื้อหาของบทนี้กล่าวถึงหลักการพื้นฐานของซูเปอร์สเกลลาร์โปรเซสเซอร์ ข้อจำกัดด้านประสิทธิภาพ และหลักการของไดนามิกสเคจดูล

2.1 พัฒนาการของซูเปอร์สเกลลาร์โปรเซสเซอร์

โปรเซสเซอร์แบบ RISC ซึ่งใช้หลักการไปป์ไลน์ได้พัฒนาการประมวลผลคำสั่งจนเกือบเทียบเท่าประสิทธิภาพสูงสุดของ RISCคือ 1 ไซเคิลต่อคำสั่ง [3] ปัจจุบันได้มีหลายวิธีการในการพยายามเพิ่มประสิทธิภาพ RISC เช่น ในซูเปอร์ไปป์ไลน์โปรเซสเซอร์ (Superpipeline processor) ทำการแบ่งสเตจการทำงานของไปป์ไลน์ให้มากขึ้น ทำให้แต่ละสเตจของไปป์ไลน์ทำงานน้อยลง คล็อกไซเคิลมีความกว้างน้อยลง จึงทำให้ความถี่คล็อกสำหรับการประมวลผลสูงขึ้น แต่ประสิทธิภาพยังคงเป็น 1 ไซเคิลต่อคำสั่งแต่ใช้เวลาน้อยลง [3] อีกวิธีหนึ่งคือพยายามทำให้โปรเซสเซอร์ประมวลผลได้หลายคำสั่งในไซเคิลเดียว ซึ่งมี 2 แนวทางคือ VLIW (Very-Long-Instruction-Word) และซูเปอร์สเกลลาร์ หลักการดังกล่าวคำสั่งจะถูกเฟ็ทช์ (Fetch) และส่งออก (Issue) ได้หลายคำสั่งต่อไซเคิล ซูเปอร์สเกลลาร์ใช้ฮาร์ดแวร์เลือกคำสั่งเพื่อส่งออกไปประมวลผล สามารถส่งออกได้ 1-8 คำสั่งต่อไซเคิล [1] สำหรับ VLIW มีคอมไพเลอร์จัดหาชุดคำสั่งดังกล่าว ซึ่งจะต้องมีหน่วยประมวลผลรองรับการทำงานอย่างเพียงพอ โดยหน่วยทำงานอาจเป็นไปป์ไลน์ซึ่งทำให้จำนวนหน่วยทำงานลดลง หรือหน่วยทำงานไม่เป็นไปป์ไลน์ซึ่งจำนวนหน่วยทำงานมากขึ้น

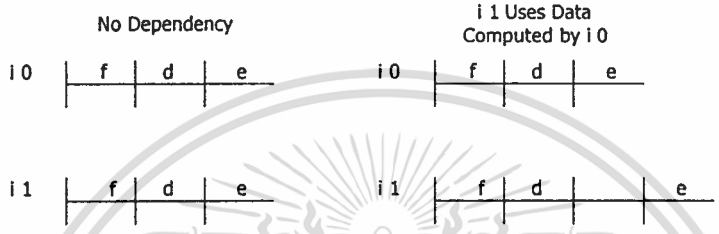
หลักการทำงานของซูเปอร์สเกลลาร์เปรียบเสมือนการนำไปป์ไลน์คำสั่งตั้งแต่ 2 ชุดขึ้นไปมาประมวลผลขนานกัน ประสิทธิภาพอยู่ที่ความสามารถทำให้ชุดคำสั่งนั้นมีความอิสระต่อกันและส่งเข้าไปป์ไลน์คำสั่งได้อย่างต่อเนื่อง ในทางปฏิบัติไม่สามารถทำได้เนื่องจากคุณสมบัติของชุดคำสั่งมีการขึ้นต่อกันโดยธรรมชาติ ฉะนั้นซูเปอร์สเกลลาร์ใดที่มีกระบวนการกลั่นกรองชุดคำสั่งที่ส่งออกให้ทำงานอย่างต่อเนื่อง ไม่มีปัญหาการขึ้นต่อกันของคำสั่ง คำสั่งประมวลได้ถูกต้องแล้ว ถือว่าโปรเซสเซอร์นั้นมีประสิทธิภาพสูงเข้าใกล้ประสิทธิภาพสูงสุดของซูเปอร์สเกลลาร์มากขึ้น กระบวนการดังกล่าวซูเปอร์สเกลลาร์ใช้ฮาร์ดแวร์ช่วยจัดการซึ่งรียกว่าวิธีการไดนามิกสเคจดูล (Dynamic schedule)

2.2 ข้อจำกัดประสิทธิภาพของซูเปอร์สเกลลาร์โปรเซสเซอร์

ลำดับคำสั่งในชุดคำสั่งอาจเกิดปัญหาการขึ้นต่อกันได้ซึ่งเป็นสาเหตุให้ประสิทธิภาพของโปรเซสเซอร์ลดลง [1] การขึ้นต่อกันของคำสั่งแบ่งเป็น 3 กรณีคือ

1) การขึ้นต่อกันของข้อมูล (Data dependency หรือ True data dependency) [3]

โดยทั่วไปผลลัพธ์ของคำสั่งหนึ่งๆ จะถูกนำไปเป็นอินพุทของคำสั่งอื่นๆ ดังนั้นคำสั่งใดที่รอผลลัพธ์จากคำสั่งก่อนหน้า คำสั่งนั้นจะเกิดปัญหาการขึ้นต่อกันของข้อมูลกับคำสั่งก่อนหน้า เมื่อซูเปอร์สเกลลาร์พยายามประมวลผล 2 คำสั่งที่มีปัญหานั้นพร้อมกัน โปรเซสเซอร์ต้องหน่วงเวลาคำสั่งที่รอคอยผลลัพธ์จนกระทั่งคำสั่งก่อนหน้าที่เป็นต้นกำเนิดผลลัพธ์ประมวลผลเสร็จ ดังรูปที่ 2.1



รูปที่ 2.1 การเกิดปัญหาข้อมูลขึ้นต่อกัน [3]

ตัวอย่างชุดคำสั่งที่มีปัญหาข้อมูลขึ้นต่อกัน

ADD R3, R1, R2 [R3←R1+R2]

↘

ADD R4, R3, R7 [R4←R3+R7]

โปรเซสเซอร์ต้องตรวจสอบปัญหาข้อมูลขึ้นต่อกันเพื่อหน่วงเวลา ถ้าไม่แก้ปัญหานี้ ปัญหาข้อมูลชนิด RAW (Read After Write) จะเกิดขึ้นนั่นคือคำสั่งที่มีปัญหาจะไม่รอผลลัพธ์จากคำสั่งก่อนหน้า ทำให้คำสั่งมีการใช้ค่าโอเปอแรนด์ที่ไม่ถูกต้อง

2) การขึ้นต่อกันของเนม (Name dependency)

ปัญหานี้เกิดขึ้นเมื่อ 2 คำสั่งใดๆใช้รีจิสเตอร์หรืออ้างถึงหน่วยความจำเดียวกัน สิ่งที่ถูกอ้างอิงถึงเรียกว่าเนม (Name) ปัญหานี้ไม่มีการส่งผ่านข้อมูลระหว่างคำสั่งที่ใช้เนมเดียวกัน

ถ้ามี 2 คำสั่งคือ j และ k โดยที่คำสั่ง j มาก่อน k ปัญหาการขึ้นต่อกันของริจิสเตอร์พิจารณาได้ 2 กรณีคือ

1. Antidependence เกิดขึ้นเมื่อคำสั่ง k ต้องการเขียนริจิสเตอร์หรือหน่วยความจำตำแหน่งเดียวกับที่คำสั่ง j อ่านข้อมูลไปใช้ กรณีนี้คำสั่ง k ต้องถูกหน่วงเวลาเพื่อให้คำสั่ง j อ่านข้อมูลไปใช้ก่อนที่คำสั่ง k จะเขียนค่าลงริจิสเตอร์ พิจารณาจากชุดคำสั่งต่อไปนี้

(j) ADD R3, R1, R2 [R3←R1+R2]

(k) ADD R1, R4, R5 [R1←R4+R5]

ถ้าไม่แก้ปัญหานี้จะทำให้เกิดปัญหาข้อมูลชนิด WAR (Write After Read)

2. Output dependence เกิดขึ้นเมื่อทั้งคำสั่ง j และ k พยายามเขียนข้อมูลลงริจิสเตอร์ปลายทางหรือหน่วยความจำตำแหน่งเดียวกัน ดังนั้นลำดับการประมวลผลเสร็จจึงสำคัญมากคือต้องให้คำสั่ง j ประมวลผลเสร็จก่อนคำสั่ง k พิจารณาได้จากชุดคำสั่งต่อไปนี้

(j) ADD R3, R4, R5 [R3←R4+R5]

(k) ADD R3, R1, R2 [R3←R1+R2]

ถ้าไม่แก้ปัญหานี้จะทำให้เกิดปัญหาข้อมูลชนิด WAW (Write After Write)

3) ปัญหาการควบคุม (Control dependency)

คำสั่งบางกลุ่มไม่สามารถเรียกมาประมวลผลได้ถ้าคำสั่งนั้นขึ้นอยู่กับคำสั่งทางแยก (Branch instruction) จึงต้องหาผลลัพธ์ของคำสั่งทางแยกก่อน แล้วจึงจะทราบว่าคำสั่งในกลุ่มนั้นจะถูกเรียกมาประมวลหรือไม่ พิจารณาจากส่วนของชุดคำสั่งต่อไปนี้

```
if (p1) then
   s1;
endif
if (p2) then
   s2;
endif
```

s1 จะเกิดปัญหาการควบคุมต่อ p1 และ s2 จะเกิดปัญหาการควบคุมต่อ p2 แต่ s2 จะไม่เกิดปัญหาการควบคุมต่อ p1

มีข้อกำหนด 2 ประการเกี่ยวกับปัญหาการควบคุมคือ

- คำสั่งที่เกิดปัญหาการควบคุมต่อคำสั่งทางแยก จะไม่สามารถย้ายไปอยู่ก่อนคำสั่งทางแยกได้ หรือไม่สามารถย้ายคำสั่งที่อยู่หลัง then ไปอยู่หน้า if ได้
- คำสั่งที่ไม่เกิดปัญหาการควบคุมต่อคำสั่งทางแยก จะไม่สามารถย้ายไปอยู่ภายใต้คำสั่งทางแยกได้ เช่น ไม่สามารถย้ายคำสั่งที่อยู่หน้า if มาไว้หลัง then ได้

ในระดับโครงสร้างไปป์ไลน์อุปสรรคที่ทำให้ไปป์ไลน์คำสั่งมีประสิทธิภาพลดลง [1] คือ

1. ปัญหาโครงสร้างฮาร์ดแวร์ (Structural Hazard) การทำงานที่เหลื่อมซ้อนกันหรือขนานกันของคำสั่ง ต้องการการสนับสนุนทางทรัพยากรฮาร์ดแวร์ โดยหน่วยทำงาน (Functional unit) จะต้องมีเพียงพอ หรือกล่าวอีกนัยหนึ่งคือมีงานหลายงานต้องการใช้ทรัพยากรฮาร์ดแวร์หรืออุปกรณ์ที่มีอยู่เพียงชิ้นเดียวพร้อมกันในเวลาเดียวกัน

2. ปัญหาข้อมูล (Data Hazard) เกิดขึ้นเนื่องจากคำสั่งปัจจุบันมีการขึ้นต่อกันของมูลกับคำสั่งก่อนหน้าที่ยังประมวลผลไม่เสร็จ ปัญหาข้อมูลแบ่งเป็น 3 ประเภทตามลำดับของการอ่านและการบันทึกของคำสั่ง และเรียกชื่อปัญหาข้อมูลตามลำดับการทำงานในโปรแกรม ถ้ามีคำสั่ง 2 คำสั่งคือ คำสั่ง i และคำสั่ง j โดยที่คำสั่ง i ทำงานก่อนคำสั่ง j แล้ว ปัญหาข้อมูลที่เกิดขึ้นคือ

1) RAW (Read After Write)

คำสั่ง j พยายามอ่านค่าจากริจิสเตอร์ต้นทางก่อนที่คำสั่ง i จะบันทึกค่าริจิสเตอร์นั้นเสร็จ ดังนั้น คำสั่ง j จะได้รับค่าริจิสเตอร์เก่า

2) WAW (Write After Write)

คำสั่ง j พยายามบันทึกค่าลงริจิสเตอร์โอเปอแรนด์ก่อนที่จะมีการบันทึกโดยคำสั่ง i ซึ่งเป็นคำสั่งที่ต้องทำงานก่อน เป็นการบันทึกค่าโอเปอแรนด์ที่ผิดลำดับ ปัญหาข้อมูลประเภทนี้เกิดขึ้นในไปป์ไลน์ที่มีการบันทึกในไปป์สเตจมากกว่าหนึ่งไปป์สเตจพร้อมๆกัน หรือในไปป์ไลน์ที่ยินยอมให้คำสั่งปัจจุบันทำงานต่อไปได้เมื่อคำสั่งก่อนหน้ากำลังถูกหน่วงเวลา

3) WAR (Write After Read)

คำสั่ง j พยายามบันทึกค่าลงริจิสเตอร์ปลายทางก่อนที่จะมีการอ่านค่านี้โดยคำสั่ง i ดังนั้น คำสั่ง i จะได้รับค่าใหม่ที่ไม่ถูกต้องถ้าคำสั่ง j ได้บันทึกค่าเรียบร้อยแล้ว

3. ปัญหาควบคุม (Control Hazard) ปัญหาการควบคุมทำให้ประสิทธิภาพของไปป์ไลน์คำสั่งลดลงเป็นอย่างมาก เมื่อคำสั่งทางแยกถูกนำเข้ามาประมวลผล โปรเซสเซอร์ต้องตรวจสอบเงื่อนไขของคำสั่งทางแยก เมื่อพบว่าคำสั่งทางแยกทำให้โปรแกรมเคาน์เตอร์เปลี่ยนแปลงไปยังตำแหน่งเป้าหมาย โปรเซสเซอร์จะต้องคำนวณตำแหน่งเป้าหมายนั้นและควบคุมให้การประมวลผลเกิดขึ้น ณ ตำแหน่งเป้าหมาย หลังจากนั้นโปรเซสเซอร์อาจจะมีการหน่วงเวลาหรือยกเลิกการทำงานของบางคำสั่งจนกว่าการประมวลผลของคำสั่งทางแยกจะเสร็จสิ้น ทำให้สูญเสียไซเคิลการทำงานมากขึ้น

2.3 รูปแบบการส่งออกคำสั่งและการประมวลผลของซูเปอร์สเกลลาร์

ซูเปอร์สเกลลาร์โปรเซสเซอร์มีเงื่อนไขการส่งออกคำสั่ง 3 แบบคือ

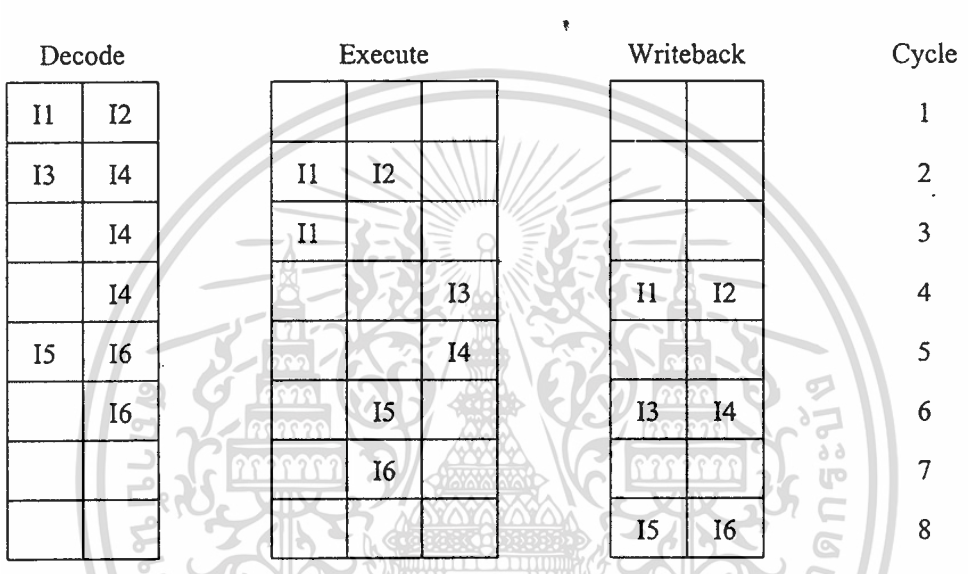
1. การส่งออกคำสั่งแบบเป็นลำดับและทำงานเสร็จแบบเป็นลำดับ

(In-Order-Issue with In-Order completion)

เป็นวิธีที่ง่ายและซับซ้อนน้อยที่สุด [3] โดยส่งออกคำสั่งไปประมวลผลที่หน่วยทำงานตามโปรแกรมต้นฉบับและประมวลผลเสร็จเป็นลำดับตามโปรแกรมต้นฉบับเช่นเดียวกัน รูปที่ 2.2 เป็น

ลำดับการประมวลผลคำสั่งตามเงื่อนไขวิธีนี้ ในรูปแสดงสเตจการทำงานในแนวนอนและไซเคิลการทำงานในแนวตั้ง สามารถถอดรหัสคำสั่งได้ครั้งละ 2 คำสั่ง มีหน่วยทำงาน 3 หน่วย และเขียนผลลัพธ์ได้ 2 คำสั่งต่อไซเคิล คำสั่ง I1-I6 มีข้อกำหนดดังนี้

- I1 ใช้เวลาประมวลผล 2 ไซเคิล
- I3 และ I4 ใช้หน่วยทำงานเดียวกัน
- I5 ใช้โอเปอแรนด์ที่เป็นผลลัพธ์จาก I4
- I5 และ I6 ใช้หน่วยทำงานเดียวกัน



รูปที่ 2.2 การส่งออกคำสั่งแบบเป็นลำดับและทำงานเสร็จแบบเป็นลำดับ [3]

จากรูป คำสั่งที่บันทึกกลับที่สเตจ WB จะมีลำดับเดียวกับคำสั่งที่เฟ็ทช์ คำสั่ง 2 คำสั่งใช้หน่วยทำงานเดียวกัน ทำให้เกิดปัญหาโครงสร้างฮาร์ดแวร์ มีการหน่วงเวลาให้คำสั่งที่มาก่อนได้ใช้หน่วยทำงานก่อน ไซเคิลที่ 4 กับ 5 และ 6 กับ 7 ต้องหน่วงเวลาคำสั่ง I4 กับ I6 ให้คำสั่ง I3 กับ I5 ทำงานก่อน นั่นคือคำสั่งที่ถอดรหัสพร้อมกันถ้าไม่เกิดปัญหาใดจะสามารถส่งออกได้พร้อมกัน ถ้าเกิดปัญหา คำสั่งแรกจะมีสิทธิ์ส่งออกได้ก่อน ถ้าคำสั่งแรกยังไม่ถูกส่งออก คำสั่งที่ 2 จะถูกหน่วงเวลาเช่นกัน อีกกรณีหนึ่งคือถ้าหน่วยทำงานที่ใช้เวลาประมวลผลมากกว่า 1 ไซเคิล เช่น คำสั่ง I1 ใช้เวลาประมวลผล 2 ไซเคิล คำสั่ง I3 จะส่งออกได้เมื่อคำสั่ง I1 ทำงานเสร็จแล้ว การทำงานทั้งหมดดังรูปใช้เวลา 8 ไซเคิล

2. การส่งออกคำสั่งแบบเป็นลำดับและประมวลผลเสร็จแบบไม่เป็นลำดับ

(In-Order-Issue with Out-of-Order completion)

วิธีนี้เป็นการส่งออกคำสั่งไปประมวลผลแบบเป็นลำดับตามโปรแกรมต้นฉบับ แต่ทำงานเสร็จแบบไม่เป็นลำดับ ดังรูปที่ 2.3 การประมวลผลเสร็จแบบไม่เป็นลำดับนี้ทำให้มีหลายๆคำสั่งประมวลผลได้พร้อมกันในหน่วยทำงาน จากรูปที่ 2.3 ขณะคำสั่ง I1 กำลังประมวลผลอยู่ โปรเซสเซอร์สามารถส่งออกคำสั่ง I3 ไปประมวลผลได้โดยไม่ต้องรอให้ I1 ทำงานเสร็จ ในสเตจ WB คำสั่ง I2 ประมวลผลเสร็จก่อน I1 การประมวลผลทั้งหมดจากรูปใช้เวลารวม 7 ไซเคิล โปรแกรมที่ประมวลผลเสร็จแบบไม่เป็นลำดับ การส่งออกจะถูกหน่วงเวลาเมื่อต้องใช้หน่วยทำงานเดียวกันหรือคำสั่งที่ส่งออกต้องการผลลัพธ์จากคำสั่งที่ยังประมวลผลไม่เสร็จ (ปัญหา RAW) และจะเกิดปัญหาข้อมูลชนิด WAR และ WAW ด้วย

Decode		Execute			Writeback		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

รูปที่ 2.3 การส่งออกคำสั่งแบบเป็นลำดับและทำงานเสร็จแบบไม่เป็นลำดับ [3]

3. การส่งออกคำสั่งแบบไม่เป็นลำดับและทำงานเสร็จแบบไม่เป็นลำดับ

(Out-of-Order-Issue with Out-of-Order completion)

การส่งออกคำสั่งและการประมวลผลเสร็จจะไม่เป็นไปตามลำดับของโปรแกรมต้นฉบับ การส่งออกแบบเป็นลำดับใน 2 วิธีข้างต้นนั้นโปรเซสเซอร์จะหยุดถอดรหัสคำสั่งเมื่อคำสั่งนั้นทำให้เกิดการขัดแย้งของฮาร์ดแวร์หน่วยทำงาน แต่โดยทั่วไปแม้ว่าคำสั่งที่มาก่อนอาจเกิดการขัดแย้งของหน่วยทำงานและเกิดปัญหาข้อมูลแต่คำสั่งลำดับหลังอาจพร้อมประมวลได้ ซึ่งเป็นหลักการของวิธีนี้นั่นเอง วิธีนี้จะต้องทำให้หน่วยถอดรหัสไม่เชื่อมต่อโดยตรงกับหน่วยทำงาน ซึ่งทำได้โดยเพิ่มบัฟเฟอร์ระหว่างหน่วยถอดรหัสและหน่วยทำงาน เรียกว่าวินโดว์คำสั่งหรือ IW (Instruction window) การเพิ่มบัฟเฟอร์วินโดว์คำสั่งทำให้สามารถถอดรหัสได้อย่างต่อเนื่องตราบเท่าที่ยังมีที่ว่างพอในบัฟเฟอร์ ขณะเดียวกันในบัฟเฟอร์ต้องมีกลไกในการเลือกคำสั่งที่พร้อมประมวลผล ฉะนั้นจึงสามารถส่งออกคำสั่งไปประมวลผลแบบไม่เป็นลำดับได้ รูปที่ 2.4 อธิบายการทำงานตามวิธีการส่งออกคำสั่งและประมวลผลเสร็จแบบไม่เป็นไปตามลำดับ จากรูปจะเห็นว่าคำสั่งถูกถอดรหัสและส่ง

เข้าบัฟเฟอร์อย่างต่อเนื่อง และบัฟเฟอร์จะส่งออกคำสั่งที่เหมาะสม เช่น ส่งออกคำสั่ง I6 ก่อน I5 เพราะคำสั่ง I5 ต้องรอผลลัพธ์จากคำสั่ง I4 ที่สเตจ WB นอกจากนี้คำสั่งถูกบันทึกแบบไม่เป็นลำดับ วิธีนี้ทำให้ประสิทธิภาพของโปรเซสเซอร์ดีที่สุด

Decode		Window	Execution			Writeback		Cycle
I1	I2							1
I3	I4	I1, I2	I1	I2				2
I5	I6	I3, I4	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

รูปที่ 2.4 การส่งออกคำสั่งแบบไม่เป็นลำดับและทำงานเสร็จแบบไม่เป็นลำดับ [3]

2.4 การเพิ่มประสิทธิภาพของซูเปอร์สเกลลาร์ด้วยการทำไดนามิกสเคจดูล

ในปี 1964 บริษัท CDC ได้นำโปรเซสเซอร์ CDC6600 ออกสู่ท้องตลาด โปรเซสเซอร์รุ่นนี้มีการทำไดนามิกสเคจดูลโดยใช้หลักการของสกอร์บอร์ด (Scoreboarding) [1] ต่อมามีการพัฒนาหลักการสกอร์บอร์ดและนำมาใช้ในเครื่องคอมพิวเตอร์ของ IBM รุ่น IBM 360/91 หลักการที่พัฒนาจากสกอร์บอร์ดเรียกว่าวิธีของ Tomasulo (Tomasulo's method) [1] จากงานวิจัย [6] แสดงให้เห็นว่าการเพิ่มประสิทธิภาพของโปรเซสเซอร์โดยการทำสแตติกสเคจดูลนั้นยังไม่เพียงพอที่จะทำให้ประสิทธิภาพของโปรเซสเซอร์สูงที่สุดได้ เนื่องจากขณะที่โปรแกรมทำงานอยู่นั้น จะไม่ทราบถึงพฤติกรรมของระบบหน่วยความจำและการประมวลผลคำสั่งทางแยกได้ แต่วิธีการไดนามิกสเคจดูลจะช่วยแก้ปัญหานี้ได้ หลักการไดนามิกสเคจดูลประกอบด้วย

2.4.1 การส่งออกคำสั่ง

นอกเหนือจากการเพิ่มจำนวนทรัพยากรฮาร์ดแวร์แล้ว การเฟ็ทช์ การถอดรหัส และการประมวลผลคำสั่งยังมีผลอย่างมากกับการทำให้ซูเปอร์สเกลลาร์โปรเซสเซอร์ทำงานแบบขนานได้ การส่งออกคำสั่งหมายถึงคำสั่งเริ่มต้นทำงานในหน่วยทำงานของโปรเซสเซอร์ การส่งออกคำสั่งมีผลต่อประสิทธิภาพของโปรเซสเซอร์

การส่งออกคำสั่งแบบเป็นลำดับและทำงานเสร็จตามลำดับคำสั่งเป็นการทำงานที่ง่ายที่สุด แต่เมื่อหน่วยทำงานเกิดการขัดแย้งหรือคำสั่งใช้เวลาทำงานหลายไซเคิล คำสั่งที่ส่งออกจะถูกหน่วงการส่งออกคำสั่งแบบนี้จึงมีข้อจำกัด

การทำงานเสร็จแบบไม่เป็นลำดับจะสนับสนุนการทำงานของคำสั่งที่ใช้เวลาหลายไซเคิล เมื่อคำสั่งที่ใช้เวลาหลายไซเคิลถูกส่งออก คำสั่งนั้นจะถูกนำไปจัดวางยังหน่วยทำงานที่เหมาะสม และทำงานต่อไป การหน่วงเวลาเกิดขึ้นได้เฉพาะเมื่อหน่วยทำงานเกิดการขัดแย้งหรือเมื่อคำสั่งไม่สามารถส่งออกได้เนื่องจากคำสั่งต้องรอผลลัพธ์ที่ยังไม่สมบูรณ์ หลักการทำงานเสร็จแบบไม่เป็นลำดับนี้คำสั่งจะถูกทำไปป์ไลน์ภายในหน่วยทำงานแต่ละหน่วย ผลลัพธ์ของคำสั่งจึงถูกบันทึกได้อย่างถูกต้องตามลำดับ นอกจากนี้การมีรีออเดอร์บัฟเฟอร์ (Reorder buffer) ยังช่วยทำให้การทำงานกับอินเตอร์รัพถูกต้องอีกด้วย เนื่องจากรีออเดอร์บัฟเฟอร์จะเก็บลำดับดั้งเดิมของคำสั่งขณะที่คำสั่งผ่านกระบวนการไปป์ไลน์ นอกจากนี้เพื่อให้การเรียกข้อมูลกลับคืนได้ถูกต้องเมื่อมีอินเตอร์รัพ (Exception) และการทำงานกับคำสั่งทางแยก ผลลัพธ์ของคำสั่งจะยังไม่ถูกบันทึกลงริจิสเตอร์ไฟล์จนกว่าคำสั่งก่อนจะทำงานเสร็จสมบูรณ์แล้ว

การส่งออกคำสั่งแบบไม่เป็นลำดับและทำงานเสร็จแบบไม่เป็นลำดับเป็นวิธีการที่ดีที่สุดโดยการสร้างบัฟเฟอร์ Lookahead window ระหว่างสเตจ Decode และ Execute หลังจากถูกถอดรหัสแล้วคำสั่งจะถูกวางใน Lookahead window ช่วงเวลาหนึ่ง คำสั่งถัดมาจะถูกถอดรหัสต่อไปแม้ว่าคำสั่งก่อนหน้าจะยังไม่ประมวลผลก็ตาม โปรเซสเซอร์มีความสามารถในการเลือกคำสั่งจาก Lookahead window มาประมวลผลได้คำสั่งจึงไม่จำเป็นต้องถูกส่งออกเป็นลำดับตามโปรแกรมต้นฉบับ ปัญหาข้อมูลขึ้นต่อกันได้รับการแก้ไขเนื่องจากโปรเซสเซอร์สามารถเลือกคำสั่งที่จะมาประมวลผลและส่งออกไปยังหน่วยทำงาน ข้อดีของการส่งออกคำสั่งแบบไม่เป็นลำดับและทำงานเสร็จแบบไม่เป็นลำดับคือเป็นการจัดหาคำสั่งจำนวนมากจากโปรเซสเซอร์(ซึ่งสามารถค้นหาคำสั่งอิสระมาทำงาน) ทำให้โปรเซสเซอร์สามารถประมวลผลแบบขนานได้สมบูรณ์

2.4.2 วินโดว์สำหรับการส่งออกคำสั่ง (Instruction issue window)

ในโปรเซสเซอร์ที่มีการส่งออกคำสั่งแบบไม่เป็นลำดับ หลังจากคำสั่งถอดรหัสแล้วจะถูกเก็บไว้ในบัฟเฟอร์ Instruction issue window จนกว่าจะถูกส่งออกไปยังหน่วยทำงานที่เหมาะสม บัฟเฟอร์สำหรับเก็บคำสั่งมี 2 แบบคือ Centralized issue window และ Reservation station ต่างกันคือ Centralized issue window จะเก็บคำสั่งทั้งหมดไว้ตำแหน่งเดียวกันส่วน Reservation station จะเก็บคำสั่งกระจายไปแต่ละหน่วยทำงาน หลักการของบัฟเฟอร์ทั้งสองแบบจะมีในโปรเซสเซอร์ปัจจุบัน ส่วนหลักการแบบผสมจะมีอยู่ในโปรเซสเซอร์บางรุ่นเท่านั้น

1. Reservation stations

หลักการของ Reservation station ถูกเสนอครั้งแรกโดย Tomasulo เพื่อช่วยในการเพิ่มประสิทธิภาพของโปรเซสเซอร์แบบโฟลทติ้งพอย์น IBM 360/91 หน่วยทำงานแต่ละหน่วยจะมี Reservation station ของตนเอง แต่ละเอนทรีของ Reservation station จะเก็บข้อมูลคำสั่ง โอเปอแรนด์ และตำแหน่งปลายทาง เมื่อโอเปอแรนด์มีค่าพร้อมใช้งาน ค่าของโอเปอแรนด์จะถูกเก็บลงในเอนทรีของ Reservation station แต่ถ้าโอเปอแรนด์ยังไม่พร้อมทำงาน ในเอนทรีของ Reservation

station จะมีแท็ก (Tag) ซึ่งสอดคล้องกับหน่วยทำงานบันทึกโอเปอแรนด์เพื่อให้โอเปอแรนด์พร้อมต่อไป เมื่อทุกโอเปอแรนด์สำหรับคำสั่งใน Reservation station พร้อม คำสั่งจะถูกส่งออกไปยังหน่วยทำงาน

Reservation station แบ่งคำสั่งออกเป็นส่วนๆ แบบไดนามิกโดยพิจารณาจากหน่วยทำงานซึ่งเป็นการกระจายคำสั่งที่จะส่งออกไป (Decentralized instruction issue) การกระจายคำสั่งช่วยลดความซับซ้อนของฮาร์ดแวร์บางส่วน ซึ่งจะตรงข้ามกับ Centralized window อย่างไรก็ตามทรัพยากรฮาร์ดแวร์ยังไม่ถูกใช้เต็มประสิทธิภาพเนื่องจากการกระจายคำสั่งเพื่อส่งออกนี้ Johnson [3] พบว่าเอนทรีส่วนใหญ่ของ Reservation station ยังถูกใช้ไม่เต็มที่เนื่องจากชุดคำสั่งมีความขนานไม่เพียงพอที่จะเก็บลงเอนทรีจนเต็ม นอกจากนั้นการควบคุมฮาร์ดแวร์สำหรับแต่ละ Reservation station ยังคล้ายคลึงกันทำให้ลอจิกการควบคุมซ้ำซ้อน

2. Centralized issue window

Centralized issue window มีความซับซ้อนเช่นกันแต่ได้ขจัดปัญหาความไร้ประสิทธิภาพและความซับซ้อนเหมือนที่มีใน Reservation station ออกไป เนื่องจากทุกคำสั่งเก็บอยู่ในบัฟเฟอร์เอนทรีในวินโดว์สามารถถูกใช้อย่างมีประสิทธิภาพกว่าเอนทรีใน Reservation station อย่างไรก็ตามแต่ละเอนทรีอาจต้องการข้อมูลมากขึ้นเพื่อสนับสนุนคำสั่งชนิดต่างๆ เช่น ถ้าชุดคำสั่งประกอบด้วยโอเปอแรนด์คำสั่ง 3 โอเปอแรนด์ แต่ละเอนทรีในวินโดว์จะต้องเตรียมที่เก็บโอเปอแรนด์ทั้ง 3 แม้ว่าบางคำสั่งจะมีโอเปอแรนด์เพียง 2 ตัวก็ตาม ปัญหานี้แก้ไขได้โดยใช้ Reservation station เพราะเอนทรีของ Reservation station จะใช้กับหน่วยทำงานที่เหมาะสม นอกจากนี้ Centralized issue window มีจำนวนเอนทรีมากกว่า Reservation station ซึ่งนำไปสู่การมีลอจิกซับซ้อน

3. Hybrid approach

เป็นการนำทั้ง Centralized issue window และ Reservation station มาประยุกต์ใช้งานร่วมกันเป็นลักษณะ Reservation station แบบง่ายโดยใช้เพียงหนึ่ง Reservation station เท่านั้นกับหน่วยทำงานแต่ละชนิด เช่น ถ้าโปรเซสเซอร์หนึ่งมีหน่วยทำงานจำนวนเต็ม 3 หน่วยก็จะรวมกันเป็น 1 Reservation station ที่มีขนาดใหญ่ขึ้น

2.4.3 กลไกการแก้ปัญหาข้อมูล

ไดนามิกสเคจดูลลิงเป็นวิธีการทางฮาร์ดแวร์ในการค้นหาวิธีการสเคจดูลเพื่อลดอินเตอร์ล็อก (Interlock) ทั้งสเคจดูลแบบสแตติกและไดนามิกทำงานเมื่อโอเปอแรนด์พร้อม ไดนามิกสเคจดูลลิงมีข้อได้เปรียบกว่าเนื่องจากใช้ฮาร์ดแวร์ จึงสามารถรู้ข้อมูลคำสั่งขณะโปรแกรมกำลังทำงานและใช้ข้อมูลนั้นมาทำสเคจดูลได้ ประสิทธิภาพของการทำสเคจดูลขึ้นอยู่กับความสามารถในการจัดเตรียมโอเปอแรนด์ให้พร้อม Johnson [3] ได้อธิบายหลักการสำคัญ 2 ประการเพื่อตรวจสอบการขึ้นต่อกันของข้อมูล (Data dependency) และการเปลี่ยนชื่อริจิสเตอร์ (Register renaming)

1. การขึ้นต่อกัน (Dependency mechanisms)

ฮาร์ดแวร์สเคจดูลลิงต้องสร้างข่าวสารของข้อมูลที่ขึ้นต่อกันขึ้นใหม่ขณะที่คำสั่งถูกถอดรหัสและวางไว้ในวินโดว์คำสั่ง (Issue window) ฮาร์ดแวร์อินเตอร์ล็อกทำให้มีการขึ้นต่อกันของคำสั่งชนิด RAW วิธีสกอร์บอร์ดช่วยแก้ปัญหาข้อมูลขึ้นต่อกัน โดยหลักการสกอร์บอร์ดยินยอมให้การถอดรหัสคำสั่งดำเนินต่อไปแม้ว่าจะมีปัญหา RAW เกิดขึ้นในชุดคำสั่ง

2. การเปลี่ยนชื่อริจิสเตอร์ (Register renaming)

การเปลี่ยนชื่อริจิสเตอร์จะช่วยแก้ปัญหาข้อมูลขึ้นต่อกันได้ โดยนำ Antidependence และ Output dependenceที่มีการขัดแย้งออก และสร้างริจิสเตอร์ตัวใหม่ขึ้นใช้งาน การหน่วงเวลาบางส่วนหมดไป

ซูเปอร์สเกลลาร์โปรเซสเซอร์ควรมีการทำ Speculative execution และ branch prediction เพื่อให้ประสิทธิภาพของโปรเซสเซอร์สูงขึ้น การทำนายทางแยกแบบไดนามิกจะให้ความแม่นยำกว่าแบบสแตติก [2] การทำ speculative execution ที่มีความลึกมากๆจะให้ผลดีกว่าวิธีทางคอมไพเลอร์

เมื่อมีฮาร์ดแวร์สนับสนุนการทำ Speculative execution ความต้องการความสนับสนุนด้านสถาปัตยกรรมเพื่อรองรับการทำนายทางแยกที่ผิดพลาดและการเรียกข้อมูลกลับคืนก็จำเป็นเช่นกัน การทำ speculative execution ทำให้ลำดับการทำงานในโปรแกรมเปลี่ยนไปโดยคำสั่งหนึ่งทำงานเสร็จสมบูรณ์ก่อนคำสั่งถัดไปจะเริ่มต้นทำงาน เพื่อสนับสนุนให้การทำงานกับอินเตอร์รัพถูกต้อง โปรเซสเซอร์ต้องเก็บสถานะที่สอดคล้องกับลำดับการทำงานของโปรแกรมเมื่อมีอินเตอร์รัพ ถ้าปราศจากฮาร์ดแวร์ที่เหมาะสมแล้ว การทำ speculative execution ก็ล้มเหลวในการทำให้โปรแกรมทำงานตามลำดับถูกต้อง

สิ่งที่น่าสนใจเกี่ยวกับ Speculative execution คือการมีรีออเดอร์บัฟเฟอร์ช่วยสนับสนุนการทำงานของฮาร์ดแวร์ ไม่เพียงแต่จัดการกับอินเตอร์รัพได้แล้วยังใช้กับการทำงานของคำสั่งแบบไม่เป็นลำดับด้วย นอกจากนี้เนื่องจากการเรียกข้อมูลกลับคืนจากการอินเตอร์รัพและการทำนายผลลัพธ์ของคำสั่งทางแยกผิดพลาดยังมีลักษณะคล้ายกัน รีออเดอร์บัฟเฟอร์จึงสามารถสนับสนุนการทำ Speculative execution การใช้รีออเดอร์บัฟเฟอร์จึงสามารถลดความซับซ้อนของฮาร์ดแวร์ของซูเปอร์สเกลลาร์โปรเซสเซอร์

2.5 ซูเปอร์สเกลลาร์โปรเซสเซอร์ในปัจจุบัน

ในเชิงการค้าซูเปอร์สเกลลาร์โปรเซสเซอร์ออกแบบให้มีการทำไดนามิกสเคจดูลลิงรวมทั้ง Speculative execution และ Branch Prediction ซึ่งได้ยกตัวอย่างพอสังเขป ดังนี้ [6]

1. DEC Alpha

โปรเซสเซอร์ Alpha ในยุคต่อไปจะมีการออกแบบเป็นซูเปอร์สเกลลาร์แบบ 4-way ซึ่งแตกต่างจากโปรเซสเซอร์ยุคก่อนๆ อย่างไรก็ตาม Alpha ก็มีฮาร์ดแวร์ของ Speculative execution ไม่ซับซ้อน คำสั่งถูกส่งออกอย่างเป็นลำดับด้วยหลักการของ Centralized issue window รูปแบบที่จำกัดของการทำงานคำสั่งไม่เป็นลำดับถูกช่วยเหลือโดยการใช้ lookup-free cache แคชระดับแรกเป็น non-blocking ซึ่งสนับสนุนการทำงานของคำสั่งอย่างต่อเนื่องแม้จะมี cache miss เกิดขึ้นก็ตาม เพื่อลดฮาร์ดแวร์ของการทำสเคจดูลลิงผู้ออกแบบโปรเซสเซอร์ Alpha ได้พยายามลดเวลาการประมวลผลคำสั่งลง โดยความถี่ของสัญญาณคล็อกจะสูงขึ้น

2. Sun AltraSPARC

บริษัทซันไมโครซิสเต็มได้เสนอไมโครโปรเซสเซอร์ยุคต่อไปออกสู่ท้องตลาด คือ UltraSPARC แต่ละไซเคิลการทำงานจะส่งออกคำสั่งได้ 4 คำสั่งสำหรับหน่วยทำงาน 6 หน่วย UltraSPARC ใช้ Decentralized in-order issue window โดยการแยกคิวส่งออกคำสั่งเป็นคิวจำนวนเต็มและคิวของโฟลทติ้งพอยน์ การออกแบบลักษณะนี้สนับสนุนการส่งออกคำสั่งแบบไม่เป็นลำดับระหว่างจำนวนเต็มและโฟลทติ้งพอยน์ ถึงแม้ว่า AltraSPARC จะมีฮาร์ดแวร์ซับซ้อนกว่า Alpha ก็ตามแต่การทำงานให้ได้ประสิทธิภาพสูงของ AltraSPARC ยังต้องพึ่งพาคอมไพเลอร์ประสิทธิภาพสูงอยู่มาก

3. PowerPC 604

ไมโครโปรเซสเซอร์ PowerPC 604 จะมีฮาร์ดแวร์ที่มีความสามารถและประสิทธิภาพสูง ซึ่งต่างจาก PowerPC ยุคแรกๆ ใน PowerPC 604 จะมี Speculative execution ใช้การทำนายทางแยกแบบไดนามิก มีการใช้หลักการเปลี่ยนชื่อริจิสเตอร์ และมี Reservation station ขนาด 2 เอนทรีสนับสนุนหน่วยทำงานแต่ละหน่วย การทำนายทางแยกแบบไดนามิกสนับสนุน Speculative execution ได้ครั้งละ 2 คำสั่งทางแยก นอกจากนี้ยังมีรีออเดอร์บัฟเฟอร์ขนาด 16 เอนทรีและหน่วยทำงาน 6 หน่วยสนับสนุนฮาร์ดแวร์ไดนามิกสเคจดูล

4. MIPS R10000

โปรเซสเซอร์ซูเปอร์สกเลลาร์แบบ 4 way ที่มีเทคโนโลยีของ MIPS คือ MIPS R10000 ที่ทำ Speculative ได้ถึง 4 คำสั่งทางแยกสนับสนุนหน่วยทำงาน 5 หน่วย MIPS R10000 ใช้แบบแผนผสมคือประกอบด้วย Instruction window ถึง 3 คิว โดยคิวจำนวนเต็มและโฟลทติ้งพอยน์ทำงานกับคำสั่งที่เหมาะสม ส่วนคิวแอดเดรสสนับสนุนหน่วย Load-store ในแต่ละคิวมีไดนามิกสเคจดูลลิงสนับสนุนการทำงานแบบไม่เป็นลำดับ ประโยชน์ของการใช้หลายๆคิวคือจะสนับสนุนคำสั่งจำนวนมากขึ้น

5. HP PA-RISC

โปรเซสเซอร์ยุคต่อไปของบริษัทฮิวเล็ทแพ็กการ์ดคือ PA-RISC ซึ่งมีสเคจดูลลิงฮาร์ดแวร์เช่นเดียวกับ PowerPC 604 นั่นคือสนับสนุนการทำงานของคำสั่งแบบไม่เป็นลำดับและ Speculative

execution ใช้ Centralized instruction issue window ขนาด 56 เอนทรีสำหรับการทำงานของหน่วยทำงาน 10 หน่วย

6. Intel P6

โปรเซสเซอร์ Intel P6 ค่อนข้างแตกต่างจากโปรเซสเซอร์อื่นตรงที่มีความเป็น RISC ไม่สมบูรณ์ถึงแม้ว่าจะเหมือนกับ RISC มากที่สุดกว่าโปรเซสเซอร์เพนเทียมอื่นๆก็ตาม คำสั่งใน Intel P6 จะถูกทำให้เป็นไมโครโอเปอเรชันซึ่งคล้ายกับคำสั่งของ RISC คำสั่งไมโครโอเปอเรชันเหล่านี้จะสนับสนุนโปรเซสเซอร์แบบไดนามิกสเคจดูลลิง มีการประมวลผลคำสั่งแบบไม่เป็นลำดับประกอบด้วย Reservation station ขนาด 20 เอนทรีสำหรับหน่วยทำงานแบบไม่เป็นลำดับ และยังมีรีออเดอร์บัฟเฟอร์ขนาด 40 เอนทรีสำหรับการประมวลผลแบบเป็นลำดับอีกด้วย

7. Motorola

โปรเซสเซอร์ Motorola 88110 เป็นโปรเซสเซอร์ที่เก่าที่สุด แตกต่างจากโปรเซสเซอร์อื่นตรงที่ส่งออกคำสั่ง 2 คำสั่งต่อไซเคิล สนับสนุนการทำงานเสร็จแบบไม่เป็นลำดับแต่จำกัดการส่งออกแบบไม่เป็นลำดับและ speculative execution มี Centralized instruction สนับสนุนหน่วยทำงาน 10 หน่วย

บทที่ 3
สถาปัตยกรรมดีแอลเอ็กซ์ซูเปอร์สเกลลาร์

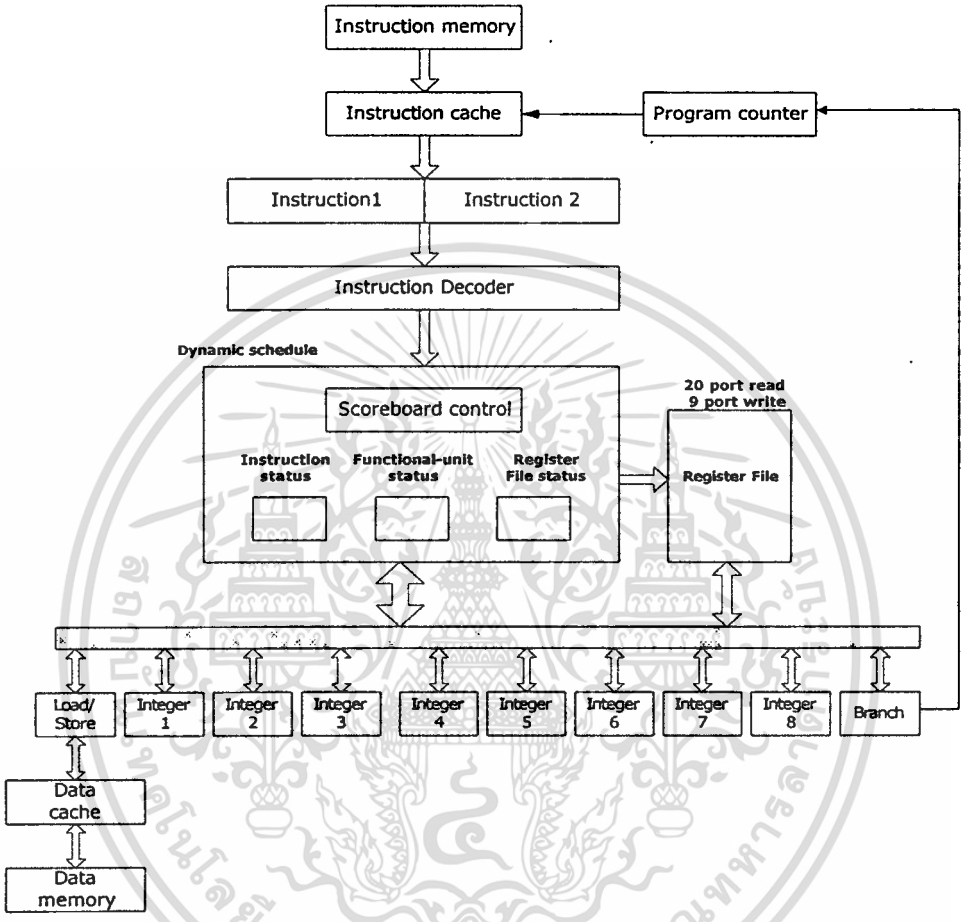
ไมโครโปรเซสเซอร์ดีแอลเอ็กซ์ขนาด 32 บิทมีสถาปัตยกรรมแบบ RISC [1] ถูกออกแบบโดย David Patterson และ John Hennessy ณ มหาวิทยาลัยแคลิฟอร์เนียเบิร์กเลย์โดยมีวัตถุประสงค์เพื่อให้นักศึกษาเข้าใจหลักของสถาปัตยกรรมคอมพิวเตอร์ การทำงานของไปป์ไลน์ และโครงสร้างแบบ RISC หลังจากนั้นได้มีกลุ่มผู้สนใจในมหาวิทยาลัยต่างๆ นำดีแอลเอ็กซ์ไปพัฒนาความรู้ที่เกี่ยวข้องกับการทำไปป์ไลน์คำสั่งและมีการออกแบบสร้างเครื่องมือต่างๆ อาทิ คอมไพเลอร์ แอสเซมเบลอร์ ฯลฯ การพัฒนาดีแอลเอ็กซ์ของกลุ่มผู้สนใจเหล่านั้นส่วนใหญ่ไม่ได้ส่งเผยแพร่ในวารสารระดับโลก เป็นแต่เพียงการศึกษาเฉพาะกลุ่มผู้สนใจเท่านั้น อย่างไรก็ตามมีเพียงโมเดลต้นแบบและแนวคิดสำหรับการแก้ปัญหาที่เกี่ยวข้องเบื้องต้นเท่านั้นที่ได้ถูกแสดงไว้ใน [1] เพื่อให้ผู้สนใจอื่นๆได้มีต้นแบบดั้งเดิมนำไปพัฒนาต่อไปอีกได้ ในงานวิจัยนี้ได้นำโครงสร้างของดีแอลเอ็กซ์เดิมมาเป็นต้นแบบเพื่อปรับปรุงเป็นซูเปอร์สเกลลาร์ และสนใจโครงสร้างในส่วนของการประมวลผลเลขจำนวนเต็ม

สถาปัตยกรรมของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ประกอบด้วยโครงสร้างสถาปัตยกรรม ชุดคำสั่งและตัวอย่างชุดคำสั่ง โครงสร้างของไปป์ไลน์ และประสิทธิภาพสูงสุดของโครงสร้าง

3.1 โครงสร้างสถาปัตยกรรม

งานวิจัยนี้ออกแบบให้ดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ สามารถเฟ็ทช์ (Fetch) ส่งออกคำสั่ง (Issue) และประมวลผล (Execute) เสร็จได้ 2 คำสั่งต่อไซเคิล มีกลไกการจัดหาคำสั่งเพื่อส่งออกไปประมวลผลและแก้ปัญหาข้อมูลด้วยการทำไดนามิกสเคจดูลตามหลักการของสกอร์บอร์ด ซึ่งมีเงื่อนไขการประมวลผลเป็นแบบส่งออกคำสั่งแบบเป็นลำดับ (In-Order Issue) และคำสั่งทำงานเสร็จแบบไม่เป็นลำดับ (Out-of-Order Completion) โครงสร้างสถาปัตยกรรมแสดงดังรูปที่ 3.1 ซึ่งอธิบายการทำงานได้ดังนี้ เมื่อคำสั่งถูกเฟ็ทช์จากหน่วยความจำแคชมาเก็บไว้ที่บัฟเฟอร์ Instruction1 และ Instruction 2 จากนั้นคำสั่งทั้ง 2 จะถูกถอดรหัสและจะถูกส่งเข้าสู่ส่วนการทำไดนามิกสเคจดูลซึ่งใช้หลักการสกอร์บอร์ด ซึ่งส่วนไดนามิกสเคจดูลจะส่งออกคำสั่งไปประมวลผลที่หน่วยทำงาน (Functional Unit) และดูแลตรวจสอบการทำงานเพื่อไม่ให้เกิดปัญหาข้อมูลระหว่างการประมวลผล คำสั่งทุกคำสั่งจะอ่านค่าโอเปอแรนด์จากริจิสเตอร์ไฟล์โดยตรง และเมื่อประมวลผลเสร็จก็จะเก็บผลลัพธ์ลงริจิสเตอร์ไฟล์โดยตรง

สถาปัตยกรรมประกอบด้วย รูปแบบของชุดคำสั่ง โครงสร้างของริจิสเตอร์ไฟล์ ชนิดข้อมูล การอ้างอิงแอดเดรส ตัวถอดรหัสคำสั่ง หน่วยทำงาน การทำไดนามิกสเคจดูล โครงสร้างของหน่วยความจำคำสั่ง โครงสร้างของหน่วยความจำข้อมูล โครงสร้างของแคชคำสั่งและข้อมูล หน่วยถอดรหัสคำสั่ง และโปรแกรมเคาน์เตอร์



รูปที่ 3.1 โครงสร้างดีแอลเอ็กซ์ซูเปอร์สเกลลาร์

3.1.1 รูปแบบของชุดคำสั่ง (Instruction Format)

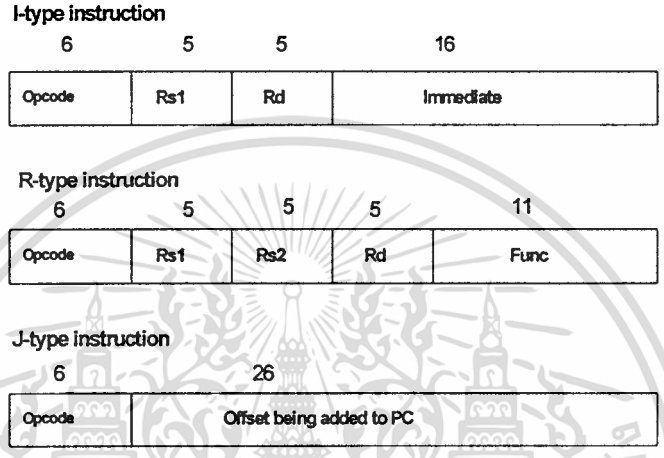
ไมโครโปรเซสเซอร์ดีแอลเอ็กซ์มีชุดคำสั่งขนาด 32 บิททั้งหมด 3 รูปแบบ และเพื่อให้การสร้างโปรเซสเซอร์ไปป์ไลน์ง่ายขึ้น จึงกำหนดให้ 6 บิทแรกของคำสั่งขนาด 32 บิทนั้นเป็นออปโค้ด (Opcode) รูปที่ 3.2 แสดงฟอร์แมท (Format) ของชุดคำสั่ง ลักษณะของฟอร์แมทคำสั่งนี้ง่ายสำหรับการจัดเตรียมฟิลด์ขนาด 16 บิทเพื่อการอ้างอิงแอดเดรสแบบอิมมีเดียท (Immediate) และดิสเพลสเมนท์ (Displacement) รวมทั้งโปรแกรมเคาน์เตอร์สำหรับชี้แอดเดรสของทางแยก (Branch) ด้วย

- คำสั่งชนิด I-Type คำสั่งในกลุ่มนี้คือ คำสั่งแบบจำนวนเต็มที่มีโอเปอแรนด์ตัวแรกเป็นริจิสเตอร์แต่มีโอเปอแรนด์ตัวที่ 2 เป็นค่าคงที่แบบ 16 บิทแบบคิดเครื่องหมาย คำสั่งอ่านข้อมูลจาก

หน่วยความจำ(Load) คำสั่งเก็บค่าข้อมูลลงหน่วยความจำ (Store) และคำสั่งทางแยกแบบที่มีระยะของตำแหน่งปลายทางจากจุดนั้นไม่เกิน 2^{15}

- คำสั่งชนิด R-Type คำสั่งในกลุ่มนี้คือ คำสั่งแบบจำนวนเต็มที่มีโอเปอแรนด์ตัวแรกและตัวที่ 2 เป็นค่ารีจิสเตอร์ทั้งคู่

- คำสั่งชนิด J-Type คำสั่งในกลุ่มนี้คือ คำสั่งทางแยกแบบที่มีระยะของตำแหน่งปลายทางจากจุดนั้นไม่เกิน 2^{25}



รูปที่ 3.2 ฟอร์แมทคำสั่งของดีแอลเอ็กซ์ [2]

3.1.2 โครงสร้างของริจิสเตอร์ไฟล์

ไมโครโปรเซสเซอร์ดีแอลเอ็กซ์มีริจิสเตอร์ทั่วไป (General Purpose registers: GPRs) ขนาด 32 บิทจำนวน 32 ตัวแบบจำนวนเต็มคือ R0 ถึง R31 เก็บอยู่ในริจิสเตอร์ไฟล์ โดยริจิสเตอร์ไฟล์สามารถอ่านค่าโอเปอแรนด์ต้นทางได้ 10 คำสั่งพร้อมกัน (โอเปอแรนด์ 2 ตัวต่อคำสั่ง) และเก็บค่าผลลัพธ์ลงริจิสเตอร์ไฟล์ได้พร้อมกัน 9 คำสั่ง โดยค่าในริจิสเตอร์ R0 เป็น 0 เสมอ นอกจากนี้ยังมีริจิสเตอร์พิเศษอีก 2-3 ตัวสำหรับการถ่ายโอนข้อมูลระหว่างริจิสเตอร์ด้วยกัน

3.1.3 ชนิดข้อมูล

ชนิดข้อมูลของไมโครโปรเซสเซอร์ดีแอลเอ็กซ์มีขนาด 8 บิทไบต์ 16 บิทฮาร์ฟเวิร์ด (Half word) และ 32 บิทเวิร์ด (word) เมื่อข้อมูลแบบไบต์หรือฮาร์ฟเวิร์ดถูกนำเข้ามาในริจิสเตอร์ จะมีการเพิ่มบิทเครื่องหมาย (sign-bit) หรือใส่เลขศูนย์ซ้ำจนกระทั่งริจิสเตอร์นั้นมีขนาดครบ 32 บิท

3.1.4 โหมดการอ้างอิงแอดเดรส

โหมดการอ้างอิงแอดเดรสของข้อมูลมี 2 โหมดคือ โหมดอิมมีเดียทและโหมดดิสเพลสเมนท์ ซึ่งมีขนาด 16 บิท การอ้างถึงหน่วยความจำแบบไบต์ในโหมดบิ๊กเอนเดียน (Big Endian) ที่มีแอด

เดรสขนาด 32 บิท ขณะที่สถาปัตยกรรมของโปรเซสเซอร์เกี่ยวข้องกับการนำข้อมูลเข้า (คำสั่ง Load) และการบันทึกข้อมูลกลับ (คำสั่ง Store) นั้นทุกการอ้างถึงหน่วยความจำจะเป็นการทำงานระหว่างหน่วยความจำกับริจิสเตอร์ทั่วไป

3.1.5 หน่วยทำงาน

โปรเซสเซอร์ดีแอลเอ็กซ์มีหน่วยทำงาน 10 หน่วย ประกอบด้วย หน่วยทำงานเลขจำนวนเต็ม (Integer Unit) 8 หน่วย หน่วยทำงานนำเข้า-การบันทึกกลับ (Load-Store unit) 1 หน่วย และหน่วยทำงานคำสั่งทางแยก (Branch unit) 1 หน่วย

1) หน่วยทำงานเลขจำนวนเต็ม ประกอบด้วยอินพุทขนาด 32 บิทจำนวน 2 ตัว และส่งผลลัพธ์กลับขนาด 32 บิทจำนวน 1 ตัว ใช้เวลาประมวลผล 1 ไซเคิล หน่วยทำงานนี้จะประมวลผลการบวก ลบ เปรียบเทียบ ตรรกพื้นฐาน และการเลื่อนบิท ในการเปรียบเทียบมีทั้งแบบคิดเครื่องหมายและแบบไม่คิดเครื่องหมาย การเลื่อนบิทสามารถเลื่อนได้ทั้งซ้ายและขวา และสามารถเลื่อนได้ครั้งละหลายบิท

2) หน่วยทำงานในการนำเข้า-การบันทึกกลับ

- หน่วยทำงานในการนำเข้า (Load unit) มีอินพุทเป็นตำแหน่งของหน่วยความจำขนาด 32 บิท และส่งผลลัพธ์จากหน่วยความจำขนาด 32 บิท ใช้เวลาประมวลผล 1 ไซเคิล

- หน่วยทำงานในการบันทึกกลับ (Store unit) มีอินพุทเป็นตำแหน่งของหน่วยความจำขนาด 32 บิท และข้อมูลที่จะบันทึกขนาด 32 บิท ใช้เวลาประมวลผล 1 ไซเคิล

หน่วยทำงานทั้ง 2 หน่วยถูกรวมอยู่ใน Load/store unit เพียงหน่วยเดียว

3) หน่วยทำงานคำสั่งทางแยก ใช้วิธีการหน่วงเวลาเพื่อให้คำสั่งทางแยกคำนวณหาตำแหน่งเป้าหมายที่จะกระโดดไป

3.1.6 การทำไดนามิกสเคจดูล

ดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ใช้หลักการสกอร์บอร์ดในการทำไดนามิกสเคจดูล โดยโครงสร้างของสกอร์บอร์ดประกอบด้วย 3 ส่วนคือ ส่วนเก็บสถานะคำสั่ง ส่วนเก็บสถานะของหน่วยทำงาน และส่วนเก็บสถานะการใช้งานรีจีสเตอร์ ซึ่งแต่ส่วนมีโครงสร้างดังนี้

1) สถานะคำสั่ง (Instruction status) มีโครงสร้างเป็นคิววงกลม สำหรับเก็บสถานะของคำสั่งขณะกำลังประมวลผลที่หน่วยทำงานต่างๆ มีโครงสร้างดังรูปที่ 3.3

Tag	Instruction	Status
xxxxxx	xxx……..xx	xxx
xxxxxx	xxx……..xx	xxx
xxxxxx	xxx……..xx	xxx

รูปที่ 3.3 โครงสร้างของสถานะคำสั่ง

จากรูปที่ 3.3 สถานะคำสั่งประกอบด้วย 3 ฟิลด์คือ

- ฟิลด์ Tag เป็นค่าที่สร้างโดยตัวควบคุมสกอร์บอร์ด (Scoreboard control) มีขนาด 6 บิท
- ฟิลด์ Instruction เก็บคำสั่งที่กำลังประมวลผล มีขนาด 32 บิท
- ฟิลด์ Status เก็บสถานะของคำสั่งที่กำลังประมวลผล โดยสถานะ "001" หมายถึงสถานะส่งออก "010" หมายถึงสถานะอ่านโอเปอแรนด์ (Read operand) "011" หมายถึงสถานะประมวลผล (Execute) และ "100" หมายถึงสถานะบันทึกกลับ (Write back)

แต่ละแถวของสถานะคำสั่งเรียกว่าเอนทรี (Entry) สำหรับงานวิจัยกำหนดให้เอนทรีมีขนาดมากเพียงพอที่จะไม่เกิดผลของการหน่วงเวลา

2) สถานะหน่วยทำงาน (Functional status) สำหรับเก็บรายละเอียดของคำสั่งที่ประมวลผลในแต่ละหน่วยทำงาน สถานะหน่วยทำงานมี 10 เอนทรีเท่ากับจำนวนหน่วยทำงาน แต่ละเอนทรีประกอบด้วยฟิลด์ดังตารางที่ 3.1

ตารางที่ 3.1 ฟิลด์ของเอนทรีในสถานะหน่วยทำงาน

ฟิลด์	ขนาด (บิท)	คำอธิบาย
Fu_tag	4	รหัสของหน่วยทำงาน
Busy	1	เป็นตัวระบุว่าเอนทรีกำลังถูกใช้งานอยู่หรือไม่
Opcode	6	โอเปอเรชัน
Opcode_ext	6	ส่วนขยายโอเปอเรชัน
Fi	5	ตำแหน่งริจิสเตอร์ปลายทาง
Fj	5	ตำแหน่งโอเปอเรชันตัวที่ 1
Fk	5	ตำแหน่งโอเปอเรชันตัวที่ 2
Qj	4	ถ้ามีการรอคอยโอเปอแรนด์ต้นทางตัวที่ 1 จากหน่วยทำงานอื่น ฟิลด์นี้จะระบุหน่วยทำงานที่กำลังรอคอย
Qk	4	ถ้ามีการรอคอยโอเปอแรนด์ต้นทางตัวที่ 2 จากหน่วยทำงานอื่น ฟิลด์นี้จะระบุหน่วยทำงานที่กำลังรอคอย
Rj	1	แฟล็กใช้บอกว่าโอเปอแรนด์ต้นทางตัวที่ 1 พร้อมใช้งานหรือไม่ ถ้าแฟล็กเป็น yes หมายถึงพร้อมใช้งาน
Rk	1	แฟล็กใช้บอกว่าโอเปอแรนด์ต้นทางตัวที่ 2 พร้อมใช้งานหรือไม่ ถ้าแฟล็กเป็น yes หมายถึงพร้อมใช้งาน

3) สถานะริจิสเตอร์ไฟล์ (Register file status) ใช้เก็บสถานะของริจิสเตอร์ปลายทางว่ากำลังถูกใช้จากหน่วยทำงานใด มีโครงสร้างดังรูปที่ 3.4

Register no.	R0	R1	R2	R3	R4	……..	R30	R31
Functional unit no.	xxxx	xxxx	xxxx	xxxx	xxxx		xxxx	xxxx

รูปที่ 3.4 โครงสร้างของสถานะริจิสเตอร์ไฟล์

จากรูปที่ 3.4 Register no. คือตำแหน่งของริจิสเตอร์ปลายทาง มีขนาด 5 บิท ส่วน Functional unit no. มีขนาด 4 บิท โดย "0001" หมายถึงหน่วยทำงาน Integer1 "0010" หมายถึงหน่วยทำงาน Integer2 "0011" หมายถึงหน่วยทำงาน Integer3 … "1000" หมายถึงหน่วยทำงาน Integer8 "1001" หมายถึงหน่วยทำงาน Load/store และ "1010" หมายถึงหน่วยทำงาน Branch

ซึ่งหลักการแก้ปัญหาโดยวิธีสกอร์บอร์ดนี้จะกล่าวถึงอย่างละเอียดอีกครั้งในบทที่ 4

3.1.7 โครงสร้างหน่วยความจำคำสั่งและหน่วยความจำข้อมูล

หน่วยความจำคำสั่งและหน่วยความจำเป็นหน่วยความจำขนาด 8 บิท จำนวน 2^{32} ตำแหน่ง

3.1.8 โครงสร้างหน่วยความจำแคชคำสั่งและแคชข้อมูล

บล็อกละ 2 เวิร์ดแบบไดเร็กแม็พ (Direct mapped)

3.1.9 หน่วยถอดรหัสคำสั่ง (Instruction decode)

สามารถถอดรหัสคำสั่งได้ 2 คำสั่งพร้อมกัน

3.2 ชุดคำสั่งและตัวอย่างของชุดคำสั่ง

ตัวอย่างของคำสั่งการนำเข้ามา-บันทึกกลับแสดงดังรูปที่ 3.5 คำสั่งทั้งหมดของไมโครโปรเซสเซอร์ดีแอลเอ็กซ์แสดงดังรูปที่ 3.8 และเพื่อง่ายต่อการสื่อความหมาย การเขียนคำสั่งแสดงการทำงานต่างๆ จึงเขียนในรูปแบบของภาษาระดับสูงและกำหนดการใช้สัญลักษณ์ดังนี้

- ตัวห้อย (Subscript) ที่อยู่ข้างหลังเครื่องหมาย ← ใช้แสดงการถ่ายโอนข้อมูลเมื่อความยาวของข้อมูลที่กำลังถ่ายโอนไม่ระบุชัดเจน และ $\leftarrow_n$ หมายถึงถ่ายโอนข้อมูลขนาด n บิท นอกจากนี้สัญลักษณ์ x,y ←z หมายถึง z ถูกถ่ายโอนไปยัง x และ y

- ตัวห้อยใช้เพื่อให้ทราบถึงการเลือกบิทจากฟิลด์ ชื่อของบิทมาจากลำดับของบิทเริ่มจากบิท 0 ตัวห้อยอาจจะเป็นเลขตัวเดียวหรือเป็นช่วงก็ได้ เช่น $Regs[R4]_0$ หรือ $Regs[R3]_{24..31}$

- ตัวแปร Mem เป็นตัวแปรแบบแถวลำดับ (Array) ใช้สำหรับหน่วยความจำหลักที่ถูกชี้โดยไบต์แอดเดรส

- ตัวยก (Superscript) ใช้เพื่อบอกว่าเป็นจำนวนซ้ำ เช่น 0^{24} หมายถึงเลขศูนย์จำนวน 24 บิท

- สัญลักษณ์ ## ใช้เชื่อมระหว่าง 2 ฟิลด์ และจะปรากฏอยู่ที่ใดก็ได้ในการถ่ายโอนข้อมูล

Example instruction	Instruction name	Meaning
LW R1,30 (R2)	Load word	Regs[R1] $\leftarrow_{32}$ Mem[30+Regs[R2]]
LW R1,1000 (R0)	Load word	Regs[R1] $\leftarrow_{32}$ Mem[1000+0]
LB R1,40 (R3)	Load byte	Regs[R1] $\leftarrow_{32}$ (Mem[40+Regs[R3]]$_0$)^{24}## Mem [40+Regs[R3]]
LBU R1,40 (R3)	Load byte unsigned	Regs[R1] $\leftarrow_{32}$ 0^{24} ## Mem[40+Regs[R3]]
LH R1,40 (R3)	Load half word	Regs[R1] $\leftarrow_{32}$ (Mem[40+Regs[R3]]$_0$)^{16} ## Mem [40+Regs[R3]] ## Mem[41+Regs[R3]]
LF F0,50 (R3)	Load float	Regs[F0] $\leftarrow_{32}$ Mem[50+Regs[R3]]
LD F0,50 (R2)	Load double	Regs[F0] ## Regs[F1] $\leftarrow_{64}$ Mem[50+Regs[R2]]
SW R3,500 (R4)	Store word	Mem[500+Regs[R4]] $\leftarrow_{32}$ Regs[R3]
SF F0,40 (R3)	Store float	Mem[40+Regs[R3]] $\leftarrow_{32}$ Regs[F0]
SD F0,40 (R3)	Store double	Mem[40+Regs[R3]] $\leftarrow_{32}$ Regs[F0]; Mem[44+Regs[R3]] $\leftarrow_{32}$ Regs[F1]
SH R3,502 (R2)	Store half	Mem[502+Regs[R2]] $\leftarrow_{16}$ Regs[R3]$_{16..31}$
SB R2,41 (R3)	Store byte	Mem[41+Regs[R3]] $\leftarrow_{8}$ Regs[R2]$_{24..31}$

รูปที่ 3.5 ตัวอย่างของคำสั่ง Load และ Store [1]

ทุกคำสั่งในเอแอลยูเป็นคำสั่งที่ทำงานระหว่างริจิสเตอร์กับริจิสเตอร์ โอเปอแรนด์ของเอแอลยูมีทั้งการคำนวณอย่างง่าย คือ บวก ลบ ตรรกะ และการเลื่อน นอกจากนี้ยังมีคำสั่งเปรียบเทียบซึ่งใช้เปรียบเทียบกันระหว่างริจิสเตอร์ 2 ตัว คือ เท่ากับ ไม่เท่ากับ น้อยกว่า มากกว่า น้อยกว่าหรือเท่ากับ และมากกว่าหรือเท่ากับ ($=, \neq, <, >, \leq, \geq$) ถ้าเงื่อนไขการเปรียบเทียบเป็นจริง คำสั่งจะเก็บค่า 1 ที่ริจิสเตอร์ปลายทาง สำหรับเงื่อนไขอื่นๆ จะเก็บค่า 0 ตัวอย่างของคำสั่งการคำนวณแสดงในรูปที่ 3.6

Example instruction	Instruction name	Meaning
ADD R1,R2,R3	Add	Regs[R1]←Regs[R2]+Regs[R3]
ADDI R1,R2,#3	Add immediate	Regs[R1]←Regs[R2]+3
LHI R1,#42	Load high immediate	Regs[R1]←42 ## 0^{16}
SLLI R1,R2,#5	Shift left logical immediate	Regs[R1]←Regs[R2] << 5
SLT R1,R2,R3	Set less than	If (Regs[R2] = Regs[R3]) Regs[R1]←1 else Regs[R1]←0

รูปที่ 3.6 ตัวอย่างคำสั่งการคำนวณในเอแอลยู [1]

การควบคุมถูกจัดการผ่านชุดของคำสั่งทางแยก โดยคำสั่งทางแยกแบบมีเงื่อนไขหมายถึงกลุ่มคำสั่ง Branch ส่วนกลุ่มคำสั่ง Jump จะเป็นคำสั่งทางแยกแบบไม่มีเงื่อนไข รูปที่ 3.7 แสดงตัวอย่างของชุดคำสั่ง jump และ branch

Example instruction	Instruction name	Meaning
J name	Jump	PC←name; ((PC+4)-2^{25})≤ name ≤ ((PC+4)+2^{25})
JAL name	Jump and link	Regs[R31]←PC+4; PC←name; ((PC+4)-2^{25})≤ name < ((PC+4)+2^{25})
JALR R2	Jump and link register	Regs[R31]←PC+4; PC←Regs[R2]
JR R3	Jump register	PC←Regs[R3]
BEQZ R4, name	Branch equal zero	If (regs[R4] = = 0) PC←name; ((PC+4) - 2^{15})≤ name < ((PC+4)+2^{15})
BNEZ R4, name	Branch not equal zero	If (regs[R4] ! = 0) PC←name; ((PC+4) - 2^{15})≤ name < ((PC+4)+2^{15})

รูปที่ 3.7 ตัวอย่างของคำสั่ง Jump และ Branch [1]

Instruction Type/Opcode	Instruction meaning
Data transfer	Move data between register and memory or between the integer and FPor special register; only memory address mode is 16-bit displacement+contents of a GPR
LB, LBU, SB	Load byte, load byte unsigned, store byte
LH, LHU, SH	Load halfword, load halfword unsigned, store halfword
LW, SW	Load word, store word (to/from integer registers)
LF, LD, SF, SD	Load SP float, load DP float, store SP float, store DP float
MOVI2S, MOVS2I	Move from/to GPR to/from a special register
MOVF, MOVD	Copy one floating-point register or a DP pair to another register or pair
MOVFP2I, MOVI2FP	Move 32 bits from/to FP tegister to/from integer registers
Arithmetic/Logical	Operations on integeror logical data in GPRs;signed arithmetics trap on overflow
ADD, ADDI, ADDU, ADDUI	Add, add immediate (all immediates are 16-bits); signed and unsigned
SUB, SUBI, SUBU, SUBUI	Subtract, subtract immediate; signed and unsigned
MULT, MULTU, DIV, DIVU	Multiply and divide, signed and unsigned; operands must be floating-point registers; all operations take and yield 32-bit values
AND, ANDI	And, and immediate
OR, ORI, XOP, XOPI	Or, or immediate, exclusive or, exclusive or immediate
LHI	Load high immediate - loads upper half of register with immediate
SLL, SRL, SRA, SLLI, SRLI, SRAI	Shifts: both immediate(S__I) and variable form(S__); shifts are shift left logical, right logical, right arithmetic
S__, S__I	Set conditional: "__"may be LT, GT, LE, GE, EQ, NE
Control	Conditional branches and jumps; PC-relative or through register
BEQZ, BNEZ	Branch GPR equal/not equal to zero; 16-bit offset from PC
BFPT, BFPF	Test comparison bit in the FP status register and branch; 16-bit offset from PC
J, JR	Jumps: 26-bit offset from PC(J) or target in register(JR)
JAL, JALR	Jump and link: save PC+4 to R31, target is PC-relative(JAL) to a register(JALR)
TRAP	Transfer to operating system at a vectored address
RFE	Return to user code from an exception; restore user code
Floating point	Floating-point operations on DP and SP formats
ADDD, ADDF	Add DP, SP numbers
SUBD, SUBF	Subtract DP, SP numbers
MULTD, MULTF	Multiply DP, SP floating point
DIVD, DIVF	Divide DP, SP floating point
CVTF2D,CVTF2I,CVTD2F, CVTD2I, CVTI2F, CVTI2D	Convert instructions: CVTx2y converts from type x to type y, where x and y are one of I(Integer), D(Double precision), or F(Single precision).
__D, __F	DP and SP compares: "__" may be LT, GT, LE, GE, EQ, NE

รูปที่ 3.8 คำสั่งทั้งหมดของไมโครโปรเซสเซอร์ดีแอลเอ็กซ์ [1]

3.3 โครงสร้างของไปป์ไลน์

โปรเซสเซอร์ดีแอลเอ็กซ์ที่มีการประมวลผลแบบซูเปอร์สเกลลาร์ประกอบด้วย 5 สเตจการทำงานดังนี้

1. สเตจ IF (Instruction Fetch Stage) เฟ็ทซ์คำสั่งจากหน่วยความจำโปรแกรม
2. สเตจ IS (Instruction Issue Stage) ถอดรหัสและส่งออกคำสั่งเพื่อประมวลผล
3. สเตจ RO (Read Operand Stage) อ่านค่าโอเปอแรนด์จากริจิสเตอร์ไฟล์
4. สเตจ EX (Execution Stage) ประมวลผลคำสั่ง
5. สเตจ WB (Write Back Stage) บันทึกค่าผลลัพธ์ลงริจิสเตอร์ไฟล์

โครงสร้างของไปป์ไลน์ดีแอลเอ็กซ์แสดงดังรูปที่ 3.9

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Instruction I	IF	IS	RO	EX	WB				
Instruction i +1	IF	IS	RO	EX	WB				
Instruction i +2		IF	IS	RO	EX	WB			
Instruction i +3		IF	IS	RO	EX	WB			
Instruction i +4			IF	IS	RO	EX	WB		
Instruction i +5			IF	IS	RO	EX	WB		
Instruction i +6				IF	IS	RO	EX	WB	
Instruction i +7				IF	IS	RO	EX	WB	
Instruction i +8					IF	IS	RO	EX	WB
Instruction i +9					IF	IS	RO	EX	WB

รูปที่ 3.9 การทำงานในแต่ละไปป์สเตจของไปป์ไลน์ดีแอลเอ็กซ์

3.4 ประสิทธิภาพสูงสุดของโครงสร้างดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์

จากโครงสร้างที่นำเสนอสามารถประมวลผลได้จำนวนไซเคิลต่อคำสั่งสูงสุดเท่ากับ 2 ไซเคิลต่อคำสั่งเมื่อไม่เกิดปัญหาของไปป์ไลน์

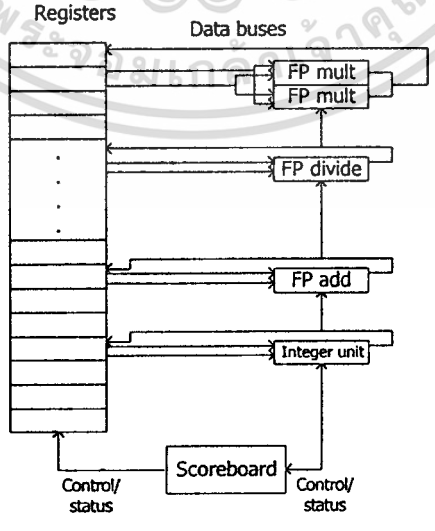
ต่อไปนี้เมื่อกล่าวถึงซูเปอร์สเกลลาร์ดีแอลเอ็กซ์แบบเดิม จะหมายถึงซูเปอร์สเกลลาร์ดีแอลเอ็กซ์ที่กล่าวถึงในบทนี้

บทที่ 4
หลักการของไดนามิกสเคจดูลด้วยวิธีสกอร์บอร์ด

เนื้อหาของบทนี้ประกอบด้วยการทำไดนามิกสเคจดูลในดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ตามวิธีสกอร์บอร์ดที่ถูกเสนอไว้ใน [1] และตัวอย่างการประมวลผลคำสั่งประกอบหลักการดังกล่าวอย่างละเอียด รายละเอียดที่เสนอนี้เป็นส่วนขยายการทำไดนามิกสเคจดูลในรูปที่ 3.1

4.1 หลักการของสกอร์บอร์ด

หลักการทำงานของสกอร์บอร์ดเป็นดังรูปที่ 4.1 หลังจากคำสั่งเฟ็ทช์เข้ามาแล้ว คำสั่งจะถูกส่งเข้าสกอร์บอร์ดเพื่อเก็บในบัฟเฟอร์คำสั่ง (Instruction status) ตัวควบคุมสกอร์บอร์ดจะทำหน้าที่พิจารณาการส่งออกคำสั่งที่อยู่ในบัฟเฟอร์คำสั่ง โดยการส่งออกคำสั่งของสกอร์บอร์ดจะเป็นแบบเรียงลำดับ (In-Order) เมื่อส่งออกคำสั่งไปประมวลผลที่หน่วยทำงาน (Functional unit) ตัวควบคุมสกอร์บอร์ดต้องตรวจสอบสถานะการทำงานของคำสั่งที่แต่ละหน่วยทำงานและบันทึกสถานะการทำงานไว้ในสกอร์บอร์ดเพื่อป้องกันปัญหาข้อมูล ถ้าตัวควบคุมสกอร์บอร์ดตรวจสอบพบว่ามีคำสั่งที่เกิดปัญหาข้อมูลขึ้น ตัวควบคุมสกอร์บอร์ดจะหน่วงเวลาคำสั่งนั้นเพื่อให้การประมวลผลข้อมูลถูกต้อง เมื่อหน่วยทำงานใดประมวลผลเสร็จ ตัวควบคุมสกอร์บอร์ดจะส่งสัญญาณเขียนกลับข้อมูลลงริจิสเตอร์ไฟล์ และบันทึกสถานะการทำงานของหน่วยทำงานว่าว่างจากการประมวลผล สกอร์บอร์ดอนุญาตให้คำสั่งส่งออกได้โดยไม่ต้องรอให้โอเปอแรนด์ต้นทางพร้อม ซึ่งทำให้การประมวลผลเสร็จแบบไม่เป็นลำดับ (Out-of-Order completion)



รูปที่ 4.1 แบบจำลองการทำงานของดีแอลเอ็กซ์ซูเปอสเกลลาร์เพื่ออธิบายหลักการสกอร์บอร์ด[1]

จากรูปที่ 4.1 แบบจำลองการทำงานของสกอร์บอร์ดมีหน่วยทำงาน 5 หน่วย ประกอบด้วย หน่วยคูณแบบโฟลทติ้งพอยน์ทั้งหมด 2 หน่วย หน่วยบวกแบบโฟลทติ้งพอยน์ทั้งหมด 1 หน่วย หน่วยหารแบบโฟลทติ้งพอยน์ทั้งหมด 1 หน่วย และหน่วยประมวลผลเลขจำนวนเต็ม 1 หน่วย ข้อมูลจะถูกส่งผ่านริจิสเตอร์ไฟล์ตามบัสแนวนอน สัญญาณควบคุมหน่วยทำงานและริจิสเตอร์ไฟล์เป็นสัญญาณในแนวตั้ง วิธีการของสกอร์บอร์ดสามารถรองรับปัญหาที่เกิดขึ้นกับคำสั่งเมื่อถูกส่งออกไปและทำงานที่หน่วยทำงาน การใช้ประโยชน์จากการทำงานแบบไม่เป็นลำดับนั้นต้องให้หลายๆคำสั่งอยู่ในหน่วยทำงานมากที่สุด ซึ่งทำได้เมื่อมีหน่วยทำงานจำนวนมาก

โครงสร้างของสกอร์บอร์ดประกอบด้วย 3 ส่วนคือ สถานะคำสั่ง (Instruction status) สถานะหน่วยทำงาน (Functional unit status) และสถานะผลลัพธ์ริจิสเตอร์ (Register result status) ซึ่งมีตัวควบคุมสกอร์บอร์ดควบคุมการทำงานทั้งหมด

1. ส่วนสถานะคำสั่ง เก็บคำสั่งที่ถอดรหัสและสถานะของแต่ละคำสั่งว่ากำลังถูกประมวลผลอยู่ในสเตจใด

2. ส่วนสถานะหน่วยทำงาน เก็บสถานะของหน่วยการทำงานแต่ละหน่วย มีฟิลด์ที่สำคัญคือ

- Busy ชี้ว่าหน่วยทำงานนั้นถูกใช้งานอยู่หรือไม่ (yes ถูกใช้งาน, no ไม่ถูกใช้งาน)
- Op ชี้ว่าหน่วยการทำงานนั้นกำลังประมวลผลด้วยโอเปอเรชันใด เช่น หน่วยการบวกอาจกำลังประมวลผลคำสั่งบวกหรือคำสั่งลบก็ได้
- F_i ตำแหน่งของริจิสเตอร์ปลายทางของคำสั่งในหน่วยทำงานนั้น
- F_j, F_k ตำแหน่งของโอเปอแรนด์ต้นทางตัวที่ 1และตัวที่ 2 ของคำสั่งในหน่วยทำงานนั้น
- Q_j, Q_k ถ้าโอเปอแรนด์ต้นทางตัวใดตัวหนึ่งหรือทั้ง 2 ตัวยังไม่พร้อมใช้งานฟิลด์นี้จะบอกว่าโอเปอแรนด์ต้นทางที่ต้องการถูกหน่วยทำงานใดใช้งานอยู่ แต่ถ้าโอเปอแรนด์พร้อมใช้งานทั้ง 2 ตัว ฟิลด์นี้จะไม่ระบุค่าใดๆ
- R_j, R_k แฟล็กที่ใช้บอกว่าโอเปอแรนด์ F_j, F_k พร้อมใช้งานหรือไม่ (yes ถ้าโอเปอแรนด์ที่ระบุนั้นๆพร้อมใช้งาน, no ถ้าโอเปอแรนด์ที่ระบุนั้นๆยังไม่พร้อมใช้งาน)

3. ส่วนสถานะผลลัพธ์ริจิสเตอร์ ใช้เก็บตำแหน่งริจิสเตอร์ปลายทางของคำสั่งทั้งหมดที่กำลังประมวลผลอยู่ในสกอร์บอร์ด

วิธีการของสกอร์บอร์ดเมื่อประยุกต์กับการทำงานของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์จะแบ่งการทำงานของไปป์ไลน์ได้ 5 สเตจการทำงาน แต่ละคำสั่งถูกเฟ็ทช์เก็บไว้ใน Instruction1 และ Instruction2 ซึ่งถือเป็นส่วนการเฟ็ทช์คำสั่ง (Instruction Fetch: IF) จากนั้นส่งผ่านอีก 4 สเตจซึ่งถูกควบคุมโดยสกอร์บอร์ด 4 สเตจดังกล่าวคือ IS (Instruction Issue stage) RO (Read Operand stage) EX (Execution stage) และ WB (Write Back stage) การทำงานของแต่ละสเตจเป็นดังนี้

1. IF เป็นสเตจการอ่านคำสั่งจากหน่วยความจำโปรแกรม โดยคำสั่งถูกอ่านมาเก็บในส่วนบัฟเฟอร์คำสั่งใน Instruction1 และInstruction2

2. IS เป็นสเตจการถอดรหัสคำสั่งเก็บลงในสถานะคำสั่ง และส่งออกคำสั่งจากส่วนสถานะคำสั่งไปประมวลผลที่หน่วยทำงาน โดยสกอร์บอร์ดส่งออกคำสั่งได้เมื่อ

- หน่วยทำงานที่จะประมวลผลคำสั่งนั้นว่าง โดยตรวจสอบจากค่าฟิลด์ Busy ของส่วนสถานะหน่วยทำงาน
- ต้องไม่มีคำสั่งใดที่ส่งออกก่อนหน้าคำสั่งนี้ (และคำสั่งยังอยู่ในสกอร์บอร์ด) ใช้ริจิสเตอร์ปลายทางตัวเดียวกับคำสั่งที่กำลังพิจารณาส่งออก โดยตรวจสอบการถูกใช้งานของริจิสเตอร์จากส่วนสถานะผลลัพธ์ริจิสเตอร์

เมื่อเป็นไปตามเงื่อนไขทั้ง 2 ข้อแล้วสกอร์บอร์ดสามารถส่งออกคำสั่งนั้นไปประมวลผลที่หน่วยทำงาน และต้องปรับปรุงค่าในส่วนสถานะหน่วยทำงาน โดยนำข้อมูลจากคำสั่งที่ส่งออกนั้นไปใส่ให้ครบ กำหนดค่า Busy ให้เป็น 1 ปรับปรุงค่าในส่วนสถานะผลลัพธ์ริจิสเตอร์ว่าริจิสเตอร์ปลายทางนั้นถูกใช้โดยหน่วยทำงานใด และปรับปรุงค่าในส่วนสถานะคำสั่งว่าคำสั่งนี้ได้ถูกส่งออกแล้ว เมื่อคำสั่งได้ส่งออกไปแล้วจะมั่นใจได้ว่าไม่เกิดปัญหา WAW

3. RO เป็นสเตจการอ่านค่าโอเปอแรนด์ เมื่อคำสั่งส่งออกไปที่หน่วยทำงาน คำสั่งอาจจะยังไม่ถูกประมวลผลเนื่องจากโอเปอแรนด์ไม่พร้อม สกอร์บอร์ดจะตรวจสอบความพร้อมของโอเปอแรนด์ต้นทางทุกไซเคิล โดยโอเปอแรนด์จะพร้อมใช้งานเมื่อ

- ไม่มีคำสั่งใดที่กำลังประมวลผลในสกอร์บอร์ดใช้ริจิสเตอร์โอเปอแรนด์นั้นเป็นริจิสเตอร์ปลายทาง โดยตรวจสอบการใช้ริจิสเตอร์ปลายทางได้จากส่วนสถานะผลลัพธ์ริจิสเตอร์

เมื่อโอเปอแรนด์พร้อมใช้งาน สกอร์บอร์ดจะส่งสัญญาณไปควบคุมหน่วยทำงานเพื่อให้หน่วยทำงานอ่านค่าโอเปอแรนด์จากริจิสเตอร์ไฟล์ การทำงานในสเตจนี้เป็นการแก้ปัญหาข้อมูลชนิด RAW จากลักษณะการทำงานนี้ทำให้คำสั่งประมวลผลแบบไม่เป็นลำดับ เมื่อมีการอ่านค่าโอเปอแรนด์แล้วสกอร์บอร์ดจะต้องปรับปรุงข้อมูลที่ส่วนสถานะคำสั่งว่าคำสั่งนั้นผ่านการอ่านค่าโอเปอแรนด์แล้ว และปรับปรุงข้อมูลในสถานะหน่วยทำงานว่าคำสั่งนั้นมีโอเปอแรนด์พร้อมใช้งานแล้ว โดยปรับปรุงค่าในฟิลด์ Qj Qk Rj และ Rk

4. EX เป็นสเตจการประมวลผลคำสั่งหลังจากได้รับค่าโอเปอแรนด์แล้วคำสั่งนั้นจะเริ่มต้นประมวลผลที่หน่วยทำงาน เมื่อหน่วยทำงานประมวลผลเสร็จจะส่งสัญญาณไปยังสกอร์บอร์ด ปรับปรุงข้อมูลที่ส่วนสถานะคำสั่งว่าคำสั่งนั้นผ่านการประมวลผลจนได้ผลลัพธ์แล้ว

5. WB เป็นสเตจที่เก็บผลลัพธ์จากการประมวลผลลงริจิสเตอร์ไฟล์ โดยจะเก็บผลลัพธ์ได้เมื่อสกอร์บอร์ดตรวจสอบแล้วว่าไม่มีปัญหา WAR ถ้ามีปัญหาสกอร์บอร์ดต้องหน่วงเวลาคำสั่งที่จะเข้าสู่สเตจ WB ไว้ก่อน เมื่อแก้ปัญหา WAR แล้วสามารถเก็บค่าลงริจิสเตอร์ไฟล์ได้ สำหรับสเตจนี้สกอร์บอร์ดต้องปรับปรุงข้อมูลทั้ง 3 ส่วนคือ

- ในสถานะคำสั่ง ต้องปรับปรุงว่าได้ผ่านการเก็บข้อมูลผลลัพธ์ลงริจิสเตอร์ไฟล์แล้ว และคำสั่งนั้นจะถูกนำออกจากสกอร์บอร์ด

- ในสถานะหน่วยทำงาน ต้องเคลียร์ค่าต่างๆของคำสั่งนั้น และกำหนดค่า Busy ให้เป็น 0 เพื่อระบุว่าหน่วยทำงานนั้นว่าง
- ในสถานะผลลัพธ์ริจิสเตอร์จะต้องเคลียร์ค่าริจิสเตอร์ปลายทางของคำสั่งนั้นให้ว่างลง

จากการประมวลผลทั้ง 5 สเตจจะเห็นว่า สกอร์บอร์ดอ่านค่าโอเปอแรนด์และเก็บค่าลงริจิสเตอร์ไฟล์โดยตรง สกอร์บอร์ดไม่ได้นำประโยชน์จากการฟอร์เวิร์ดข้อมูล (Data forwarding) มาใช้

รูปที่ 4.2 แสดงส่วนประกอบของสกอร์บอร์ดทั้ง 3 ส่วนซึ่งแสดงการประมวลผลชุดคำสั่งนี้

```
LD      F6, 34(R2)
LD      F2, 45(R3)
MULTD   F0, F2, F4
SUBD    F8, F6, F2
DIVD    F10, F0, F6
ADDD    F6, F8, F2
```

Instruction		Instruction status			
		Issue	Read operands	Execution complete	Write result
LD	F6, 34 (R2)	√	√	√	√
LD	F2, 45 (R3)	√	√	√	
MULTD	F0, F2, F4	√			
SUBD	F8, F6, F2	√			
DIVD	F10, F0, F6	√			
ADDD	F6, F8, F2				

	Functional unit status								
Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	R3				Yes	
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

	Register result status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Integer			Add	Divide			

รูปที่ 4.2 ส่วนประกอบของข้อมูลในสกอร์บอร์ด [1]

จากรูปที่ 4.2 คำสั่งทั้ง 6 คำสั่งถูกเฟ็ทช์ ถอดรหัส และนำไปเก็บในสถานะคำสั่ง บางคำสั่งผ่านการส่งออกแล้วและกำลังประมวลผลอยู่ในสเตจต่างๆ คำสั่งใดถูกส่งออก ข้อมูลของคำสั่งนั้นจะถูกบันทึกลงในสถานะหน่วยทำงาน และริจิสเตอร์ปลายทางของคำสั่งจะถูกบันทึกในสถานะผลลัพธ์ริจิสเตอร์ว่าได้ถูกใช้โดยหน่วยทำงานใด เช่น คำสั่ง SUB F8, F6, F2 ซึ่งอยู่ในสถานะคำสั่งได้ถูกส่งออกแล้วและใช้หน่วยทำงาน ADD (ดูที่สถานะหน่วยทำงาน) และกำลังรอคอยโอเปอแรนด์ F2 จากหน่วยทำงาน Integer ซึ่งคำสั่ง LD F2, 45(R3) กำลังใช้งานอยู่ ส่วนในสถานะผลลัพธ์ริจิสเตอร์จะระบุว่าหน่วยทำงาน ADD ใช้ริจิสเตอร์ F8 เป็นริจิสเตอร์ปลายทาง

ในสถานะคำสั่งระบุว่าคำสั่ง Load คำสั่งแรกประมวลผลเสร็จสิ้นและผลลัพธ์ถูกเขียนกลับลงริจิสเตอร์ไฟล์เรียบร้อยแล้ว คำสั่ง Load คำสั่งที่ 2 ประมวลผลเสร็จแต่ยังไม่ถูกบันทึกผลลัพธ์ลงริจิสเตอร์ไฟล์ และที่สถานะหน่วยทำงานบอกว่าหน่วยทำงาน Mult1 กำลังรอโอเปอแรนด์ซึ่งถูกใช้โดยหน่วยทำงาน integer หน่วยทำงาน ADD กำลังรอคอยโอเปอแรนด์ซึ่งถูกใช้งานโดยหน่วยทำงาน integer และหน่วยทำงาน Divide กำลังรอคอยโอเปอแรนด์ซึ่งถูกใช้งานโดยหน่วยทำงาน Mult1 คำสั่ง ADDD ถูกหน่วงเวลาการส่งออกเนื่องจากเกิดปัญหาโครงสร้างฮาร์ดแวร์ ซึ่งปัญหาจะถูกกำจัดออกไปได้เมื่อคำสั่ง SUBD ประมวลผลเสร็จและเขียนข้อมูลกลับลงริจิสเตอร์ไฟล์แล้ว เอนทรีใดในสถานะหน่วยทำงานที่ไม่ถูกใช้ จะเป็นช่องว่าง

รูปที่ 4.3 เป็นตัวอย่างการประมวลผลชุดคำสั่งโดยใช้ชุดคำสั่งเดิม ก่อนที่คำสั่ง MULTD จะเข้าสู่สเตจ WB คำสั่ง DIVD ยังไม่อ่านค่าโอเปอแรนด์เนื่องจากโอเปอแรนด์รอผลลัพธ์จากคำสั่ง MULTD คำสั่ง ADDD อ่านค่าโอเปอแรนด์เสร็จแล้วและคำสั่งอยู่ในสเตจ EX เข้าสู่สเตจ WB ไม่ได้เนื่องจากเกิดปัญหา WAR บนริจิสเตอร์ F6 ซึ่งถูกใช้โดยคำสั่ง DIVD

รูปที่ 4.4 แสดงตัวอย่างการประมวลผลชุดคำสั่งเดิม ก่อนที่คำสั่ง DIVD จะเข้าสู่สเตจ WB จากรูป คำสั่ง ADDD สามารถเข้าสู่สเตจ WB ได้ทันทีที่คำสั่ง DIVD ได้อ่านค่าโอเปอแรนด์แล้ว และจะเหลือคำสั่ง DIVD เป็นคำสั่งสุดท้ายที่อยู่ระหว่างการประมวลผล

Instruction		Instruction status			
		Issue	Read operands	Execution complete	Write result
LD	F6, 34 (R2)	√	√	√	√
LD	F2, 45 (R3)	√	√	√	√
MULTD	F0, F2, F4	√	√	√	
SUBD	F8, F6, F2	√	√	√	√
DIVD	F10, F0, F6	√			
ADDD	F6, F8, F2	√	√	√	

Name	Functional unit status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Add	F6	F8	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

	Register result status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Divide			

รูปที่ 4.3 สถานะของสกอร์บอร์ดเมื่อคำสั่ง MULTD กำลังจะเข้าสู่สเตจ WB [1]

Instruction		Instruction status			
		Issue	Read operands	Execution complete	Write result
LD	F6, 34 (R2)	√	√	√	√
LD	F2, 45 (R3)	√	√	√	√
MULTD	F0, F2, F4	√	√	√	√
SUBD	F8, F6, F2	√	√	√	√
DIVD	F10, F0, F6	√	√	√	
ADDD	F6, F8, F2	√	√	√	√

Name	Functional unit status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6			Yes	Yes

	Register result status								
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1					Divide			

รูปที่ 4.4 สถานะของสกอร์บอร์ดเมื่อคำสั่ง DIVD กำลังจะเข้าสู่สเตจ WB [1]

4.2 ตัวอย่างการประมวลผลคำสั่ง

จากชุดคำสั่งเดิม 6 คำสั่งนำมาแสดงการประมวลผลตามหลักการของสกอร์บอร์ดอย่างละเอียดทุกขั้นตอน โดยกำหนดให้หน่วยทำงาน ADD ใช้เวลาประมวลผล 2 ไซเคิล หน่วยทำงาน Mult1 และ Mult2 ใช้เวลาประมวลผล 10 ไซเคิล หน่วยทำงาน Integer ใช้เวลา 1 ไซเคิล และหน่วยทำงาน DIVIDE ใช้เวลาประมวลผล 40 ไซเคิล ดังรูปที่ 4.5-4.24 หมายเลขที่อยู่ในสถานะคำสั่งเป็นหมายเลขไซเคิลที่ประมวลผลถึงจุดนั้น

Cycle 0

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU									

รูปที่ 4.5 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 0

Cycle 1

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F6		R2				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Integer					

รูปที่ 4.6 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 1

Cycle 2

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F6		R2				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Integer					

รูปที่ 4.7 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 2

Cycle 3

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F6		R2				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Integer					

รูปที่ 4.8 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 3

Cycle 4

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F6		R2				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Integer					

รูปที่ 4.9 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 4

Cycle 5

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F2		R3				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU		Integer							

รูปที่ 4.10 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 5

Cycle 6

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F6		R3				Yes
Mult1	Yes	Mul	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	No								
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1	Integer							

รูปที่ 4.11 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 6

Cycle 7

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F2		R3				Yes
Mult1	Yes	Mul	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	No								

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1	Integer			Add				

รูปที่ 4.12 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 7

Cycle 8a

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	Yes	LD	F2		R3				Yes
Mult1	Yes	Mul	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1	Integer			Add	Divide			

รูปที่ 4.13 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 8a

Cycle 8b

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	Yes	Mul	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2			Yes	No
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1				Add	Divide			

รูปที่ 4.14 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 8b

Cycle 9

Instruction status

Instruction	i	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	Yes	Mul	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1				Add	Divide			

รูปที่ 4.15 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 9

Cycle 11

Instruction status

Instruction	i	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	Yes	Mul	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1				Add	Divide			

รูปที่ 4.16 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 11

Cycle 12

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	Yes	Mul	F0	F2	F4			Yes	Yes
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1					Divide			

รูปที่ 4.17 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 12

Cycle 13

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	Yes	Mul	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1			Add		Divide			

รูปที่ 4.18 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 13

Cycle 14

Instruction status

Instruction	i	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	Yes	Mul	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1			Add		Divide			

รูปที่ 4.19 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 14

Cycle 16

Instruction status

Instruction	i	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	Yes	Mul	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mul1		No	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mul1			Add		Divide			

รูปที่ 4.20 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 16

Cycle 20

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Add		Divide			

รูปที่ 4.21 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 20

Cycle 21

Instruction status

Instruction	I	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Add		Divide			

รูปที่ 4.22 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 21

Cycle 22

Instruction status

Instruction	i	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU						Divide			

รูปที่ 4.23 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 22

Cycle 61

Instruction status

Instruction	i	j	k	Instruction Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	
ADDD	F6	F8	F2	13	14	16	22

Functional Unit Status

Name	Busy	Op	Dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj ? Rj	Fk? Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register Result Status

Register	F0	F2	F4	F6	F8	F10	F12	...	F30
FU						Divide			

รูปที่ 4.24 การประมวลผลชุดคำสั่งด้วยวิธีสกอร์บอร์ดในไซเคิล 61

รูปที่ 4.25 เป็นรูปสรุปการบันทึกข้อมูลลงสเตจต่างๆ พร้อมเงื่อนไขการบันทึก

Instruction status	Wait unit	Bookkeeping
Issue	Not busy (FU) and not result (D)	Busy (FU) ←yes; Op (FU) ← op; Fi (FU) ←'D'; Fj(FU) ←'S1'; Fk(FU) ←'S2'; Qj←result ('S1'); Qk←result ('S2') Rj←not Qj; Rk ←not Qk; Result ('D') ← FU;
Read operands	Rj and Rk	Rj← yes; Rk ← yes; Qj ← clear; Qk ← clear
Execution complete	Functional unit done	
Write result	∀ f((Fj (f)≠Fi (FU) or Rj (f)=yes) & (Fk (f)≠Fi (FU) or Rk (f)=yes))	∀f (if Qj (f) = FU then Rj (f) ← yes) ; ∀ f (if Qk (f) = FU then Rk (f) ← yes) ; Result (Fi (FU)) ← clear; Busy (FU) ← No

รูปที่ 4.25 การบันทึกข้อมูลลงสเตจและเงื่อนไขการบันทึกตามหลักการของสกอร์บอร์ด [1]

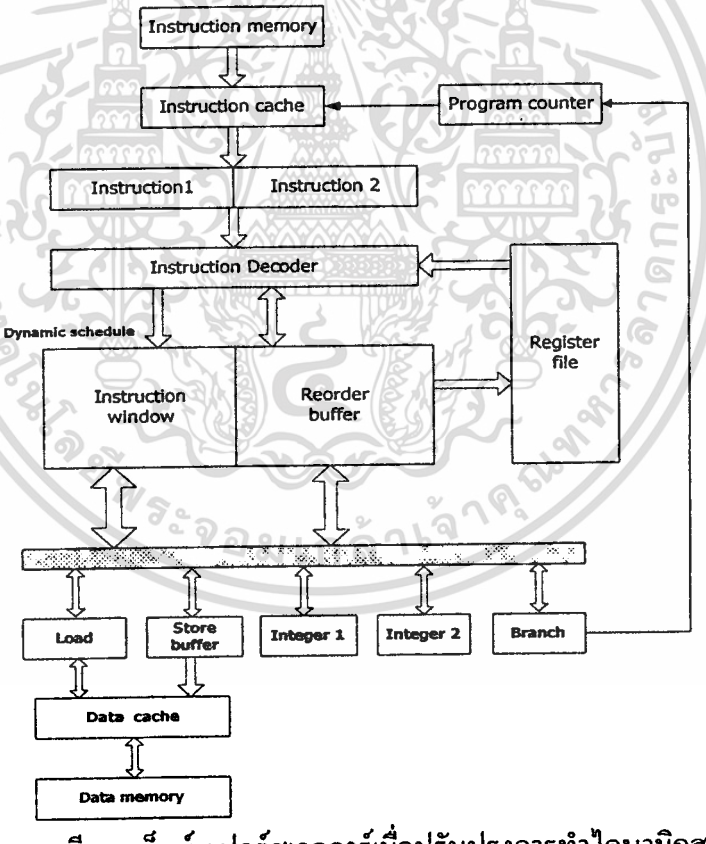
บทที่ 5

การปรับปรุงประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ด้วยการปรับปรุงการทำไดนามิกสเคจดูล

เนื้อหาในส่วนนี้ประกอบด้วยวิธีการไดนามิกสเคจดูลที่ออกแบบและรายละเอียดการทำงานอย่างละเอียดในแต่ละขั้นตอน โดยจะนำวิธีการไดนามิกสเคจดูลที่ออกแบบไปแทนที่การทำงานของไดนามิกสเคจดูลเดิม (สกอร์บอร์ด) และมีโครงสร้างสถาปัตยกรรมเป็นดังรูปที่ 5.1

รายละเอียดของโครงสร้างและเทคนิคที่นำมาปรับปรุงวิธีไดนามิกสเคจดูลมีดังนี้

5.1 โครงสร้างสถาปัตยกรรม



รูปที่ 5.1 สถาปัตยกรรมของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์เมื่อปรับปรุงการทำไดนามิกสเคจดูล

จากรูปที่ 5.1 ในงานวิจัยได้พยายามรักษาโครงสร้างโดยรวมของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ให้คงเดิมมากที่สุด สิ่งที่ปรับปรุงและเปลี่ยนแปลงจากรูปที่ 3.1 มีดังนี้

- แทนที่ส่วนการทำไดนามิกสเคจดูลของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์เดิม ด้วยหลักการไดนามิกสเคจดูลที่นำเสนอ
- เปลี่ยนโครงสร้างของริจิสเตอร์ไฟล์จากเดิมมีพอร์ตการอ่าน 20 พอร์ตให้เป็น 4 พอร์ตและพอร์ตการเขียนข้อมูล 9 พอร์ตเป็น 2 พอร์ต
- ลดหน่วยประมวลผลเลขจำนวนเต็ม (Integer unit) จาก 8 หน่วยทำงานเหลือ 2 หน่วยทำงาน
- เพิ่มหน่วยทำงาน Store buffer
- ดีแอลเอ็กซ์ซูเปอร์สเกลลาร์เดิมใช้หน่วย Load/Store ซึ่งทำงานได้ทั้งคำสั่ง Load และ Store โดยปรับปรุงหน่วยทำงานดังกล่าวให้ทำงานเฉพาะคำสั่ง Load เท่านั้นและเรียกว่าหน่วยทำงาน Load
- โครงสร้างของไปป์ไลน์ยังคงเป็น 5 สเตจการทำงานเท่าเดิม บางสเตจมีการเปลี่ยนชื่อให้เหมาะสมกับการทำงานในสเตจนั้น
- ประมวลผลคำสั่งทางแยกด้วยการทำนายทางแยกแบบสแตติก โดยคาดเดาว่า Not taken

เมื่อนำวิธีการไดนามิกสเคจดูลที่นำเสนอมาประยุกต์ใช้ ทำให้เงื่อนไขการประมวลผลเป็นแบบส่งออกคำสั่งไม่เป็นลำดับ (Out-of-Order issue) และคำสั่งทำงานเสร็จแบบไม่เป็นลำดับ (Out-of-Order completion) จากรูปที่ 5.1 หลังจากคำสั่งถูกเฟ็ทช์และถอดรหัสแล้วจะถูกส่งไปเก็บใน IW (Instruction Window) และ RB (Reorder Buffer) เพื่อให้กลไกของไดนามิกสเคจดูลทำการคัดเลือกชุดคำสั่งที่เหมาะสมก่อนส่งไปประมวลผลที่หน่วยทำงาน หลังจากหน่วยทำงานแต่ละหน่วยทำงานเสร็จจะส่งผลลัพธ์จากการประมวลมาเก็บไว้ใน RB ซึ่งเป็นขั้นตอนของการเขียนกลับ (Write back stage) และฟอร์เวิร์ดข้อมูล (Data forwarding) กลับไปยัง IW เพื่อให้คำสั่งอื่นๆที่รอคอยผลลัพธ์นั้นสามารถนำผลลัพธ์ไปใช้ต่อ เพื่อส่งออกและประมวลผลต่อไปได้ ผลลัพธ์ของคำสั่งที่ประมวลผลเสร็จสมบูรณ์เสร็จแล้วจะถูกจัดเก็บลงริจิสเตอร์ไฟล์เมื่อคำสั่งนั้นเคลื่อนที่มาอยู่ที่ส่วนหัวคิวของ RB ภายใน RB จะมีกลไกในการแก้ปัญหาข้อมูลด้วยการเปลี่ยนชื่อริจิสเตอร์ปลายทางให้เป็นตำแหน่งของคำสั่งที่อยู่ใน RB ซึ่งหลักการดังกล่าวจะขจัดปัญหาข้อมูลชนิด WAR และ WAW ได้หมด

5.2 โครงสร้างไปป์ไลน์

โครงสร้างของไปป์ไลน์ที่ปรับปรุงยังคงประกอบด้วย 5 สเตจการทำงานคือ

1. IF (Instruction fetch stage) เฟ็ทช์คำสั่งจากหน่วยความจำโปรแกรม
2. ID (Instruction decode stage) ถอดรหัสและส่งออกคำสั่ง
3. EX (Execution stage) ประมวลผลคำสั่งที่แต่ละหน่วยทำงาน

4. WB (Write back stage) เก็บผลลัพธ์จากการประมวลผลลงใน RB และ IW
5. RC (Result commit) เก็บผลลัพธ์ลงรีจิสเตอร์ไฟล์

5.3 โครงสร้างวินโดว์คำสั่ง (Instruction window)

วินโดว์คำสั่ง (Instruction window: IW) เป็นบัฟเฟอร์แบบคิววงกลม (Circular queue) สำหรับเก็บคำสั่งที่เตรียมส่งออกไปประมวลผล โดยรับคำสั่งที่ผ่านการถอดรหัสแล้วจากตัวถอดรหัสคำสั่ง (Instruction decoder) และส่งคำสั่งออกไปประมวลผลที่หน่วยทำงาน โครงสร้างของวินโดว์คำสั่งประกอบด้วยเอนทรี (Entry) สำหรับเก็บคำสั่ง ในงานวิจัยกำหนดให้จำนวนเอนทรีมีมากพอที่ไม่ทำให้เกิดการหน่วงเวลาเมื่อ IW เต็ม แต่ละเอนทรีประกอบด้วยข้อมูลแต่ละฟิลด์รวม 13 ฟิลด์ ดังตารางที่ 5.1

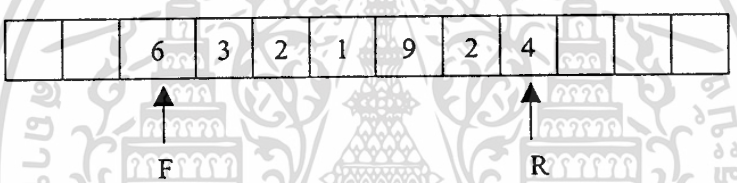
ตารางที่ 5.1 โครงสร้างข้อมูลในแต่ละเอนทรีของวินโดว์คำสั่ง

ฟิลด์	ขนาด (บิท)	คำอธิบาย
Rs1_data	32	ข้อมูลโอเปอแรนด์ต้นทางตัวที่ 1
Rs1_ready	1	แฟล็กใช้บอกว่าข้อมูลใน Rs1_data พร้อมที่จะประมวลผลหรือไม่
Rs1_tag	6	เก็บค่าแท็ก (Tag) ของโอเปอแรนด์ต้นทางตัวที่ 1 เพื่อใช้เปรียบเทียบกับแท็กผลลัพธ์ของคำสั่งที่ประมวลผลเสร็จ ซึ่ง Rs1_tag จะมีค่าในกรณีที่คำสั่งรอคอยโอเปอแรนด์ต้นทางตัวที่ 1
Rs2_data	32	ข้อมูลโอเปอแรนด์ต้นทางตัวที่ 2
Rs2_ready	1	แฟล็กใช้บอกว่าข้อมูลใน Rs2_data พร้อมที่จะประมวลผลหรือไม่
Rs2_tag	6	เก็บค่าแท็กของโอเปอแรนด์ต้นทางตัวที่ 2 เพื่อใช้เปรียบเทียบกับแท็กผลลัพธ์ของคำสั่งที่ประมวลผลเสร็จ ซึ่ง Rs2_tag จะมีค่าในกรณีที่คำสั่งรอคอยโอเปอแรนด์ต้นทางตัวที่ 2
Dest_tag	6	เก็บค่าแท็กของรีจิสเตอร์ปลายทาง
Opcode	6	โอเปอเรชันของคำสั่ง
Opcode_ext	6	ส่วนขยายโอเปอเรชันของคำสั่ง
Immed16	16	ค่าอิมมีเดียทขนาด 16 บิท
Immed26	26	ค่าอิมมีเดียทขนาด 26 บิท
Issued	1	แฟล็กใช้บอกว่าคำสั่งนั้นได้ถูกส่งออกไปแล้วหรือไม่
PC_addr	32	ตำแหน่งของคำสั่งในหน่วยความจำโปรแกรม

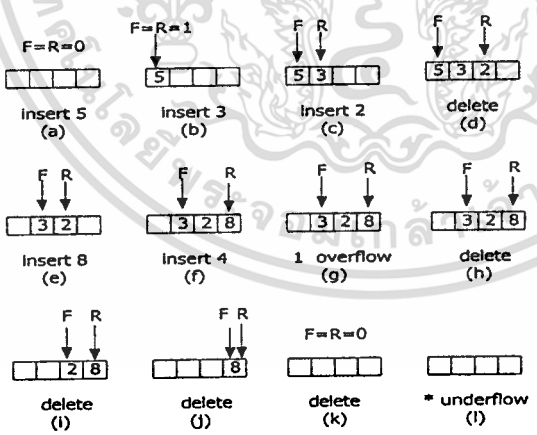
เนื่องจากในงานวิจัยได้อ้างอิงถึงคิวบ่อยครั้ง จึงขอกล่าวถึงโครงสร้างและการทำงานของคิวพอสังเขป

1. โครงสร้างแบบคิว (Queue)

จากลักษณะของการเข้าคิว สิ่งที่เข้าไปในคิวก่อนย่อมออกจากคิวก่อน คิวจึงมีโครงสร้างแบบ FIFO (First-In-First-Out) สามารถสร้างคิวได้โดยใช้แถวลำดับ (Array) และตัวชี้ตำแหน่ง (Pointer) ของหัวคิวและท้ายคิวชื่อ F (Front pointer) และ R (Rear pointer) ตามลำดับ โดยข้อมูลจะเข้าไปทางท้ายคิว (Rear) และออกจากคิวทางหัวคิว (Front) ตัวชี้ F จะชี้ไปยังหัวแถว (ตัวแรก) และตัวชี้ R จะชี้ไปยังหางแถว (ตัวสุดท้าย) ดังรูปที่ 5.2 ซึ่งเป็นโครงสร้างของคิวจำนวน 12 ช่อง โดยตอนแรกคิวยังว่างอยู่ F และ R มีค่า 0 ทั้งคู่ การกระทำกับคิวได้แก่ การใส่ข้อมูลในคิวและการลบข้อมูลออกจากคิว ลำดับการใส่ข้อมูลเข้าสู่คิว และการลบข้อมูลออกจากคิวขนาด 4 ช่องเป็นดังรูปที่ 5.3 โดยเรียงลำดับเหตุการณ์จาก a ถึง l



รูปที่ 5.2 โครงสร้างของคิวจำนวน 12 ช่อง



รูปที่ 5.3 ลำดับเหตุการณ์การใส่ข้อมูลเข้าสู่คิวและการลบข้อมูลออกจากคิว

ข้อห้ามการนำข้อมูลเข้าสู่คิวคือไม่สามารถนำเข้าข้อมูลเข้าคิวในขณะที่คิวเต็มหรือไม่มีที่ว่างด้าน R ถ้าพยายามทำจะเกิดข้อผิดพลาด Overflow ข้อห้ามในการนำข้อมูลออกจากคิวคือไม่สามารถนำอะไร

ออกจากคิวที่ว่างเปล่า ถ้าพยายามทำจะเกิดข้อผิดพลาด Underflow รูปที่ 5.4 แสดงอัลกอริทึมการใส่ข้อมูลลงในคิว และลบข้อมูลออกจากคิว โดยคิวมีขนาด N

การใส่ข้อมูลลงในคิว

Qinsert: Procedure(Y)
If (R=N) then
  Message "Queue full";
  Return;
Endif
R=R+1
Q(R) = Y
If (F = 0) then
  F=1; R=1
Endif
END Qinsert

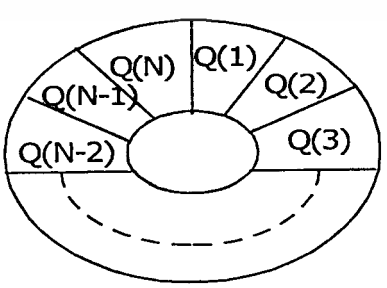
การลบข้อมูลออกจากคิว

Qdelete: Procedure(Y)
If (F=0) then
  Message "Queue empty";
  Return;
Endif
If (F = R) then
  F=0; R=0
Else
  F=F+1
Endif
END Qdelete

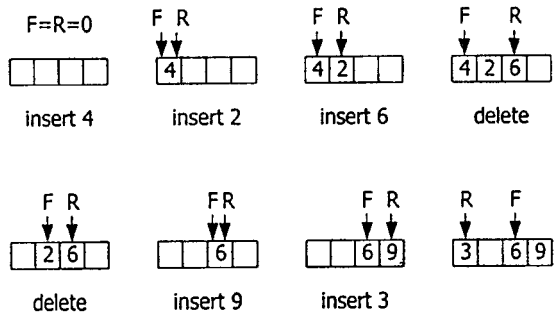
รูปที่ 5.4 อัลกอริทึมในการใส่ข้อมูลลงในคิวและลบข้อมูลออกจากคิว

2. คิววงกลม (Circular queue)

การจัดให้แถวลำดับทำงานเป็นคิวมีข้อจำกัดที่ว่าบางครั้งคิวที่ว่างอยู่ด้าน F แต่ด้าน R เต็ม ทำให้ไม่สามารถนำข้อมูลเข้าสู่คิวได้ การจะนำที่ว่างในส่วนหน้ามาใช้ได้อีกทำได้โดย ให้ส่วนท้ายของคิวที่ตำแหน่ง Q(N) ไปต่อกับส่วนหน้าที่ตำแหน่ง Q(1) โครงสร้างนี้เรียกว่าคิววงกลมดังรูปที่ 5.5 ในคิวแบบธรรมดา เมื่อ R=N หมายถึงคิวเต็ม แต่คิวแบบวงกลมเมื่อ R=N จะปรับให้ R=1 ลำดับเหตุการณ์การใส่ข้อมูลลงคิวและลบข้อมูลออกจากคิววงกลมแสดงดังรูปที่ 5.6



รูปที่ 5.5 โครงสร้างของคิววงกลมขนาด N ช่อง



รูปที่ 5.6 ลำดับเหตุการณ์การใส่ข้อมูลลงคิวและลบข้อมูลออกจากคิววงกลม

การกระทำกับคิววงกลมคือการใส่ข้อมูลในคิวและการลบข้อมูลออกจากคิวมีอัลกอริทึมดังรูปที่ 5.7

การใส่ข้อมูลลงในคิววงกลม

```
Cqinsert: Procedure(Y)
If (R=N) then
  R=1
Else
  R=R+1;
Endif
If (R=F) then
  Message "Queue overflow";
  Return;
Endif
Q(R) = Y
If (F=0) then
  F=1; R=1;
  Endif;
END CQinsert
```

การลบข้อมูลออกจากคิววงกลม

```
Qdelete: Procedure(Y)
If (F=0) then
  Message "Queue empty";
  Return;
Endif
Y = Q(F);
If (F = R) then
  F=0; R=0;
  Return;
Endif
If (F=N) then
  F=1
Else
  F=F+1;
Endif
END CQdelete
```

รูปที่ 5.7 อัลกอริทึมการใส่ข้อมูลลงในคิววงกลมและการลบข้อมูลออกจากคิววงกลม

5.4 โครงสร้างรีออเดอร์บัฟเฟอร์ (Reorder buffer: RB)

เป็นบัฟเฟอร์ที่มีโครงสร้างแบบคิววงกลม สำหรับเก็บรักษาลำดับคำสั่งและผลลัพธ์ของคำสั่งที่เข้ามาประมวลผล ข้อมูลของคำสั่งใน RB ถูกส่งมาจากตัวถอดรหัสเช่นเดียวกับ IW ผลลัพธ์จากการประมวลผลจะถูกนำมาเก็บใน RB คำสั่งที่ประมวลผลเสร็จแล้วและยังคงอยู่ใน RB และจะถูกนำไปเก็บที่รีจิสเตอร์ไฟล์อย่างเป็นลำดับตามคำสั่งที่เข้าสู่โปรเซสเซอร์ โดยการเก็บผลลัพธ์ลงรีจิสเตอร์ไฟล์จะกระทำกับคำสั่งที่อยู่หัวคิวของ RB โครงสร้างของ RB ที่ออกแบบแต่ละเอนทรีประกอบด้วย 4 ฟิลด์ ดังตารางที่ 5.2

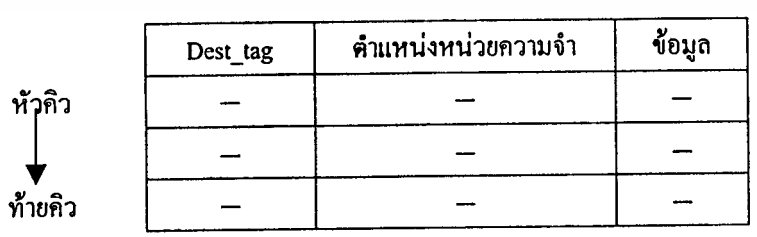
ตารางที่ 5.2 โครงสร้างข้อมูลของแต่ละเอนทรีใน RB

ฟิลด์	ขนาด (บิท)	ความหมาย
Ready_dest	1	แฟล็กใช้ระบุว่าคำสั่งนั้นประมวลผลเสร็จแล้วหรือไม่
Dest_tag	6	แท็กของรีจิสเตอร์ปลายทาง
Dest_addr	5	ตำแหน่งของรีจิสเตอร์ปลายทาง
Dest_data	32	ผลลัพธ์จากการประมวลผล

แต่ละเอนทรีใน RB ถูกระบุด้วยค่า Dest_tag ในแต่ละเอนทรีจะมีค่า Dest_tag ไม่ซ้ำกัน ค่า Dest_tag ถูกสร้างจากตัวถอดรหัสคำสั่งและถูกใช้เป็นคีย์ในการบันทึกผลลัพธ์จากหน่วยประมวลผลลง RB และเป็นคีย์ในการฟอร์เวิร์ดข้อมูลไปยัง IW และใช้ขจัดปัญหาข้อมูลประเภท WAR และ WAW

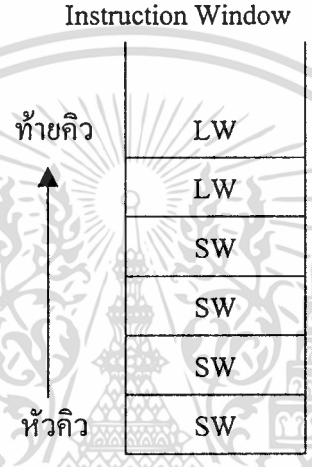
5.5 โครงสร้างของ Store buffer

Store buffer เป็นบัฟเฟอร์ที่มีโครงสร้างแบบคิววงกลมสำหรับเก็บคำสั่ง Store ที่ส่งมาจาก IW โดยคำสั่ง Store ถูกส่งออกมาอย่างเป็นลำดับเมื่อเทียบกับคำสั่ง Load และ Store ด้วยกัน โครงสร้างของ Store buffer ที่ออกแบบเป็นดังรูปที่ 5.8



รูปที่ 5.8 โครงสร้างของ Store buffer

Store buffer ช่วยเพิ่มประสิทธิภาพการทำงานให้กับคำสั่งในกลุ่ม Load และ Store ซึ่งส่งผลถึงประสิทธิภาพโดยรวมของโปรเซสเซอร์ ความสำคัญของการมี Store buffer คือปกติการอ้างอิงหน่วยความจำข้อมูลทำได้ไซเคิลละคำสั่งเท่านั้น การออกแบบซูเปอร์สเกลลาร์ให้มีหน่วยทำงาน Load/Store มากกว่า 1 หน่วยจะเป็นการสิ้นเปลือง ถ้าออกแบบให้มีหน่วยทำงานเดียว ประสิทธิภาพการทำงานอาจจะไม่ดีเท่าที่ควร ทั้งนี้ขึ้นอยู่กับโปรแกรมและแอพลิเคชันที่นำมาทดสอบประสิทธิภาพ พิจารณาตัวอย่างชุดคำสั่งที่อยู่ใน IW ดังรูป 5.9



รูปที่ 5.9 คำสั่งใน IW เพื่อใช้อธิบายประโยชน์ของ Store buffer

ดีแอลเอ็กซ์ซูเปอร์สเกลลาร์สามารถส่งออกคำสั่งได้ครั้งละ 2 คำสั่ง และจากโครงสร้างในบทที่ 3 รูปที่ 3.1 ซึ่งมีหน่วย Load/store 1 หน่วย IW ส่งออกคำสั่ง Store ได้ครั้งละคำสั่ง จึงต้องใช้เวลา 4 ไซเคิล จากนั้นคำสั่ง Load จึงจะเริ่มส่งออกได้ และการที่คำสั่ง Load จะโหลดค่าจากหน่วยความจำข้อมูลเพื่อเป็นอินพุทให้กับคำสั่งประเภทเอแอลยู การส่งออกคำสั่ง Load ช้าทำให้มีการรอการประมวลผล ถ้าออกแบบให้มีหน่วยทำงาน Load (ทำงานเฉพาะคำสั่งในกลุ่ม Load) ทำงานคู่กับหน่วยทำงาน Store buffer (ทำงานเฉพาะคำสั่งในกลุ่ม Store) จะทำให้ประสิทธิภาพการทำงานดังกล่าวเพิ่มขึ้นโดยสามารถส่งออกคำสั่ง Store ทั้ง 4 คำสั่งภายใน 2 ไซเคิล และคำสั่ง Load จะเริ่มส่งออกได้ ทำให้ลดการรอการประมวลผลของคำสั่งประเภทเอแอลยู

หลังจากเพิ่ม Store buffer ทำให้ส่งออกคำสั่ง Store 2 คำสั่งได้พร้อมกัน ข้อมูลใน Store buffer จะเก็บลงในหน่วยความจำข้อมูลได้เมื่อคำสั่ง Store นั้นเคลื่อนที่มาถึงหัวคิวใน RB และเมื่อคำสั่งเข้าสู่หรือออกจาก Store buffer ต้องมีการปรับเปลี่ยนตำแหน่งหัวคิวและท้ายคิวของ Store buffer ให้ถูกต้องด้วย

5.6 การถอดรหัสคำสั่ง

ตัวถอดรหัสคำสั่งจะอ่านคำสั่งจากบัฟเฟอร์ Instruction1 และ Instruction2 มาถอดรหัสหาตำแหน่งริจิสเตอร์ต้นทาง ริจิสเตอร์ปลายทาง และค่าอื่นๆ เพื่อให้ได้ข้อมูลครบตามโครงสร้างของ IW และ RB เมื่อถอดรหัสคำสั่งเรียบร้อยแล้ว ตัวถอดรหัสจะส่งข้อมูลทั้งหมดเก็บลงใน IW และ RB ถ้ามีเอนทรีว่าง ตัวถอดรหัสจะต้องสร้างแท็กให้กับแต่ละคำสั่ง เรียกว่า Dest_tag (Destination tag) ข้อมูลของ IW และ RB รวมทั้งหมด 17 ฟิลด์ โดย IW มี 13 ฟิลด์ และ RB มี 4 ฟิลด์ตัวถอดรหัสต้องหาข้อมูลมาใส่ให้ครบทุกฟิลด์

พิจารณาข้อมูลในฟิลด์ต่างๆ ของ IW ค่าในฟิลด์ที่ 1-6 จากตารางที่ 5.1 ใน IW จะเกี่ยวกับโอเปอแรนด์ต้นทางทั้ง 2 ตัว ซึ่งมีวิธีการหาค่าดังนี้

หลังถอดรหัสคำสั่งแล้วจะทราบโอเปอแรนด์ต้นทางทั้ง 2 ตัวของคำสั่งนั้น โอเปอแรนด์จะถูกอ่านจาก RB หรือริจิสเตอร์ไฟล์อย่างใดอย่างหนึ่งเท่านั้น ในงานวิจัยนี้โอเปอแรนด์ต้นทางได้มาจาก 3 กรณีดังนี้

1) เมื่อนำตำแหน่งของโอเปอแรนด์ต้นทางที่ได้จากการถอดรหัสคำสั่งไปค้นหาและเปรียบเทียบกับตำแหน่งของโอเปอแรนด์ปลายทางในฟิลด์ Dest_addr ของ RB ถ้าเท่ากันและฟิลด์ Ready_dest ของ RB = 1 แสดงว่าคำสั่งนั้นได้ประมวลผลเสร็จสิ้นแล้ว โอเปอแรนด์ต้นทางจะถูกอ่านจากฟิลด์ Dest_data ใน RB แต่ถ้าพบข้อมูลใน RB มากกว่า 1 เอนทรีต้องเลือกค่าเอนทรีของคำสั่งล่าสุดเท่านั้น โดยส่วนท้ายคิวจะมีลำดับความสำคัญสูงสุด กรณีนี้จะได้ค่าทั้ง 6 ฟิลด์ ดังนี้

Rs1_data และหรือ Rs2_data จะถูกกำหนดให้มีค่าเท่ากับ Dest_data

Rs1_ready และหรือ Rs2_ready เท่ากับ 1

Rs1_tag และหรือ Rs2_tag ไม่สนใจ

2) เมื่อนำตำแหน่งของโอเปอแรนด์ต้นทางที่ได้จากการถอดรหัสคำสั่งไปค้นหาและเปรียบเทียบกับตำแหน่งของโอเปอแรนด์ปลายทางในฟิลด์ Dest_addr ของ RB ถ้าเท่ากันแต่ฟิลด์ Ready_dest ของ RB = 0 แสดงว่าคำสั่งนั้นยังอยู่ในระหว่างการประมวลผล กรณีนี้จะได้ค่าทั้ง 6 ฟิลด์ ดังนี้

Rs1_data และหรือ Rs2_data ไม่สนใจ

Rs1_ready และหรือ Rs2_ready เท่ากับ 0

Rs1_tag และหรือ Rs2_tag จะถูกกำหนดให้มีค่าเท่ากับ Dest_tag

3) เมื่อนำตำแหน่งของโอเปอแรนด์ต้นทางที่ได้จากการถอดรหัสคำสั่ง ไปเปรียบเทียบกับตำแหน่งของโอเปอแรนด์ปลายทางในฟิลด์ Dest_addr ของ RB ถ้าไม่พบข้อมูลใน RB ให้อ่านค่าโอเปอแรนด์ของคำสั่งนั้นจากริจิสเตอร์ไฟล์โดยตรง กรณีนี้จะได้ค่าทั้ง 6 ฟิลด์ ดังนี้

Rs1_data และหรือ Rs2_data เป็นค่าที่ถูกอ่านจากริจิสเตอร์ไฟล์

Rs1_ready และหรือ Rs2_ready เท่ากับ 1

Rs1_tag และหรือ Rs2_tag ไม่สนใจ

7. Dest_tag ได้จากการสร้างแท็กของตัวถอดรหัส
8. Opcode ได้จาก Instruction1 หรือ Instruction2 (ดูฟอร์แมทคำสั่งในบทที่ 3)
9. Opcode_ext ได้จาก Instruction1 หรือ Instruction2 (ดูฟอร์แมทคำสั่งในบทที่ 3)
10. Immed16 ได้จาก Instruction1 หรือ Instruction2 (ดูฟอร์แมทคำสั่งในบทที่ 3)
11. Immed26 ได้จาก Instruction1 หรือ Instruction2 (ดูฟอร์แมทคำสั่งในบทที่ 3)
12 Issued กำหนดค่าให้เป็น "0"
13. PC_addr ได้จากตำแหน่งของคำสั่ง

พิจารณาข้อมูลในฟิลด์ต่างๆ ของ RB

1. Ready_dest กำหนดให้มีค่าเท่ากับ 0 เนื่องจากยังไม่มีผลลัพธ์จากการประมวลผล
2. Dest_tag ได้จากการสร้างแท็กของตัวถอดรหัส
3. Dest_addr ตำแหน่งของริจิสเตอร์ปลายทางได้จาก Instruction 1 หรือ Instruction 2
4. Dest_data ไม่สนใจค่านี้เนื่องจากคำสั่งยังไม่ประมวลผล

5.7 วิธีการส่งออกคำสั่ง (Issue)

การส่งออกคำสั่งพิจารณาจากส่วนล่างของ IW (หัวคิว) ซึ่งมีลำดับความสำคัญสูงสุด นั่นคือในแต่ละรอบไซเคิลการส่งออกถ้ามีหลายคำสั่งที่พร้อมจะส่งออกได้ คำสั่งที่อยู่ใกล้หัวคิวจะถูกพิจารณาให้ส่งออกก่อนเมื่อคำสั่งนั้นยังไม่ถูกส่งออก (ค่าในฟิลด์ Issue=0) และโอเปอแรนด์พร้อม (เนื่องจากคำสั่งลำดับหลังมักจะรอผลลัพธ์จากคำสั่งลำดับต้นที่มาก่อน จึงควรประมวลผลคำสั่งลำดับต้นให้เสร็จก่อน) จากนั้นคำสั่งในลำดับถัดไปจึงจะถูกพิจารณา ในแต่ละไซเคิลจะส่งออกคำสั่งได้สูงสุดไม่เกิน 2 คำสั่ง การพิจารณาคำสั่งแต่ละประเภทมีดังนี้

5.7.1 คำสั่งเอแอลยูประเภท R-type

คำสั่งพร้อมส่งออกได้เมื่อโอเปอแรนด์ต้นทางตัวที่ 1 และ 2 พร้อม นั่นคือโอเปอแรนด์ทั้งคู่มีค่าพร้อมใช้งานและถูกเก็บอยู่ใน IW คำสั่งชนิดนี้ถูกส่งออกไปประมวลผลที่หน่วยทำงาน integer1 หรือ integer2

5.7.2 คำสั่งเอแอลยูประเภท I-type

คำสั่งพร้อมส่งออกเมื่อโอเปอแรนด์ต้นทางตัวที่ 1 พร้อมเพียงตัวเดียวเท่านั้น คำสั่งชนิดนี้ถูกส่งออกไปประมวลผลที่หน่วยทำงาน integer1 หรือ integer2

5.7.3 คำสั่ง Load และ Store คำสั่งชนิดนี้จัดอยู่ในประเภท I-type

5.7.3.1 คำสั่ง Load (LW, LB, LH, LBU, LHU) พร้อมส่งออกเมื่อโอเปอแรนด์ต้นทางตัวที่ 1 พร้อมเพียงตัวเดียวเท่านั้น และคำสั่ง Store ที่อยู่ก่อนหน้าคำสั่ง Load ที่พิจารณาต้องถูกส่งออกแล้ว ตัวอย่างการพิจารณาการส่งออกคำสั่ง Load แสดงดังรูปที่ 5.10

Instruction Window : IW

		:	:	ลำดับโปรแกรม
ท้ายคิว	5	ADD R9, R7, R8	Not issue	ADD R4, R5, R6
	4	LW 1(R2), R6	Not issue	SW 0(R1), R4
	3	SUB R7, R2, R1	Issued	SUB R7, R2, R1
	2	SW 0(R1), R4	Not issue	LW 1(R2), R6
หัวคิว	1	ADD R4, R5, R6	Issued	ADD R9, R7, R8

รูปที่ 5.10 ตัวอย่างการพิจารณาการส่งออกคำสั่ง Load

รูปที่ 5.10 เป็นการพิจารณาคำสั่ง LW ในแถวที่ 4 ว่าจะส่งออกได้หรือไม่ แม้ว่าโอเปอแรนด์ต้นทางตัวที่ 1 ของคำสั่ง LW(R2) พร้อมใช้งานก็ตาม แต่ยังมีคำสั่ง SW ในแถวที่ 2 ยังไม่ถูกส่งออก ดังนั้นคำสั่ง LW ในแถวที่ 4 จึงไม่สามารถส่งออกได้ คำสั่ง ADD R9, R7, R8 ในแถวที่ 5 จะถูกพิจารณาในลำดับถัดมา คำสั่งในกลุ่ม Load จะถูกส่งไปประมวลผลที่ Load unit และสามารถถูกส่งออกแบบไม่เป็นลำดับได้โดยคำสั่ง Store และคำสั่งทางแยกที่อยู่ก่อนหน้า Load ต้องถูกส่งออกหมดแล้ว

5.7.3.2 คำสั่ง Store (SW, SH, SB) คำสั่งเหล่านี้พร้อมส่งออกเมื่อโอเปอแรนด์ต้นทางตัวที่ 1 และ 2 พร้อมใช้งาน คำสั่ง Load และ Store อื่นๆที่อยู่ก่อนหน้ามันถูกส่งออกแล้ว และหน้าคำสั่ง Store ที่พิจารณาต้องไม่มีคำสั่งทางแยกที่ยังไม่ส่งออก

จากรูปที่ 5.11 สมมติกำลังพิจารณาการส่งออกของคำสั่ง Store ในแถวที่ 4 ว่าส่งออกได้หรือไม่ ถึงแม้ว่าโอเปอแรนด์ตัวที่ 1 และ 2 ของคำสั่งในแถวที่ 4 (R1, R8) พร้อมใช้งาน คำสั่งนี้ก็จะไม่สามารถส่งออกได้เพราะมีคำสั่ง Store ในแถวที่ 2 ยังไม่ถูกส่งออก

Instruction Window : IW

		:	:
ท้ายคิว	4	SW 2(R1), R8	Not issue
	3	LW 2(R2), R5	Issued
	2	SW 0(R4), R6	Not issue
หัวคิว	1	SLL R4, R2, R1	Issued

รูปที่ 5.11 ตัวอย่างการพิจารณาการส่งออกของคำสั่ง Store

คำสั่งในกลุ่ม Store ถ้าส่งออกได้จะถูกส่งไปเก็บใน Store buffer คำสั่งในกลุ่ม Store ด้วยกันเองจะถูกส่งออกแบบเป็นลำดับเท่านั้น

5.7.4 คำสั่งทางแยก

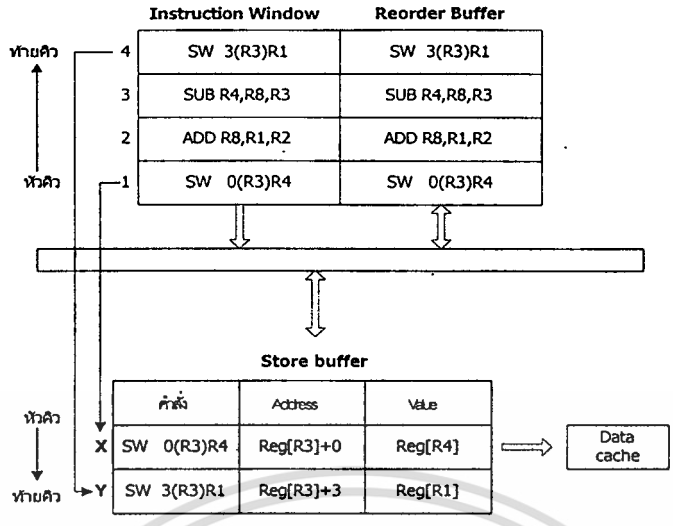
คำสั่งทางแยกแบบมีเงื่อนไขถูกส่งออกได้เมื่อโอเปอแรนด์ตัวที่ 1 พร้อมใช้งาน ส่วนคำสั่งทางแยกแบบไม่มีเงื่อนไขบางคำสั่งที่ไม่ต้องการโอเปอแรนด์สามารถส่งออกได้ทันทีเมื่อพิจารณาการส่งออกคำสั่งนั้น คำสั่งทางแยกถูกส่งออกไปประมวลผลที่ Branch unit

5.8 วิธีการประมวลผลคำสั่ง (Execute)

5.8.1 คำสั่งเอแอลยูทั้งชนิด R-type และ I-type การส่งออกไปประมวลผลที่หน่วยทำงาน Integer จะใช้เวลา 1 ไซเคิล

5.8.2 คำสั่งกลุ่ม Store เมื่อส่งออกไปประมวลผลจะถูกเก็บลงใน Store buffer เนื่องจากคำสั่งในกลุ่มนี้ประมวลผลแบบเป็นลำดับเมื่อเทียบกับคำสั่ง Store ด้วยกัน คำสั่ง Store ใน Store buffer จะเก็บคำสั่งลงหน่วยความจำข้อมูลเมื่อคำสั่ง Store นั้นเคลื่อนที่มาอยู่ส่วนหัวคิว ดังรูปที่ 5.12

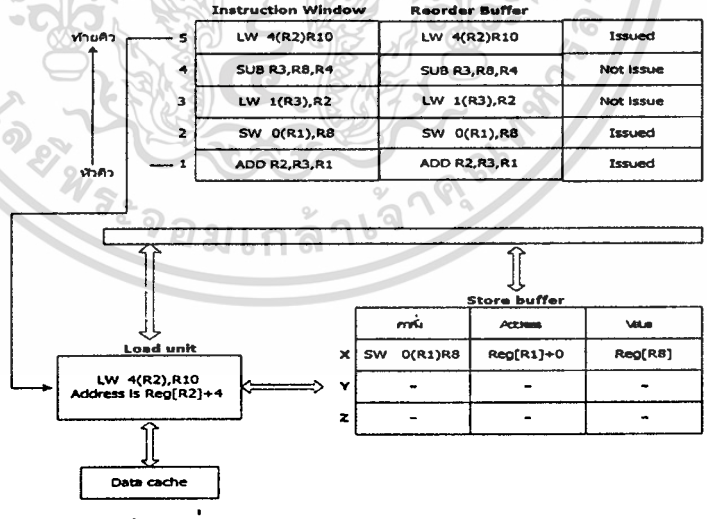
จากรูป 5.12 พบว่าคำสั่ง Store ใน IW ถูกส่งไปเก็บไว้ใน Store buffer ตามลำดับก่อนหลังใน IW คำสั่ง SW ที่อยู่ในแถว X ของ Store buffer สามารถเก็บลงในหน่วยความจำข้อมูลได้เนื่องจากคำสั่งดังกล่าวที่อยู่ใน RB ได้เคลื่อนที่มาถึงหัวคิวแล้ว สำหรับคำสั่ง SW ที่อยู่ในแถว Y ของ Store buffer จะยังไม่ถูกเก็บลงในหน่วยความจำข้อมูลจนกว่าคำสั่ง SW ที่ตรงกับแถวที่ 4 ของ RB ได้เคลื่อนที่มาถึงหัวคิว



รูปที่ 5.12 อธิบายการประมวลผลคำสั่งกลุ่ม Store

5.8.3 กลุ่มคำสั่ง Load เมื่อคำสั่งนี้ถูกส่งออกไปประมวลผลที่ Load unit คำสั่ง Load ใน Load unit ต้องตรวจสอบใน Store buffer ว่ามีคำสั่ง Store คำสั่งใดที่มีตำแหน่งอ้างถึงหน่วยความจำข้อมูลตำแหน่งเดียวกับตัวมันหรือไม่

- ถ้ามี คำสั่ง Load ใน Load unit จะนำค่าข้อมูลของคำสั่ง Store ไปใช้ได้ทันที
- ถ้าไม่มี คำสั่งที่อยู่ใน Load unit จะอ่านค่าข้อมูลที่ต้องการจากหน่วยความจำข้อมูลตำแหน่งที่อ้างอิง พิจารณาดังรูปที่ 5.13



รูปที่ 5.13 อธิบายการประมวลผลของกลุ่มคำสั่ง Load

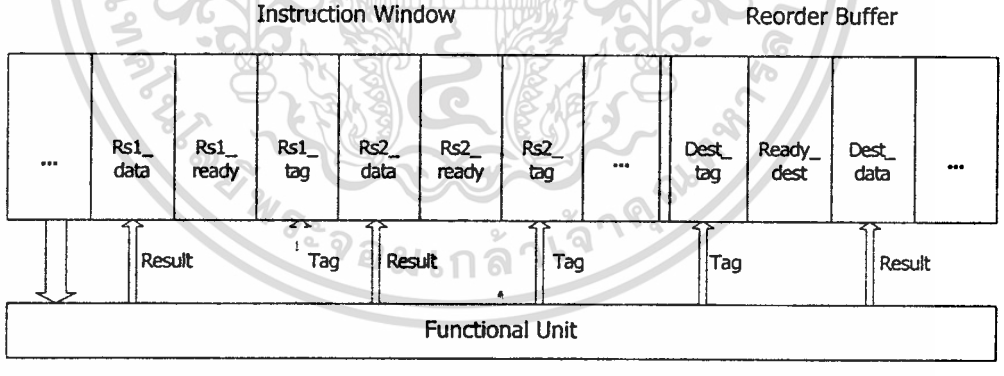
จากรูป 5.13 เมื่อคำสั่ง Load ในแถวที่ 5 ของ IW ถูกส่งออกมายัง Load unit คำสั่ง Load ใน Load unit จะนำตำแหน่งของหน่วยความจำคือ (Reg[R2]+4) ค้นหาในฟิลด์ Address ของ Store buffer ซึ่งขณะนี้มีเพียงเอนทรีเดียว จากนั้นเปรียบเทียบค่าระหว่าง (Reg[R2]+4) และ (Reg[R1]+0) ถ้ามีค่าตรง

กันคำสั่ง Load สามารถนำค่าในฟิลด์ Value ไปใช้ได้ทันที แต่ถ้าไม่มีค่าตรงกัน คำสั่ง Load จะนำค่านั้นมาจากหน่วยความจำข้อมูลแทน

5.8.4 คำสั่งทางแยก การประมวลผลคำสั่งทางแยกใช้วิธีการทำนายทางแยกแบบสแตติกและคาดเดาพฤติกรรมของคำสั่งทางแยกว่า Not taken เมื่อคำสั่งทางแยกถูกส่งออกไปที่ Branch unit ซึ่ง Branch unit ประมวลผลโดยพิจารณาว่าคำสั่งทางแยกนั้นมีพฤติกรรม Taken หรือ Not taken (คำสั่งทางแยกแบบไม่มีเงื่อนไขมีพฤติกรรม Taken เสมอ) และคำนวณหาตำแหน่งของโปรแกรมเคาน์เตอร์ที่จะกระโดดไป ถ้าคำสั่งมีพฤติกรรม Taken ก็จะโหลดตำแหน่งของคำสั่งถัดไปเข้าสู่โปรแกรมเคาน์เตอร์เพื่อเฟ็ทช์คำสั่งใหม่ในไซเคิลถัดไป และเคลียร์ค่าเอนทรีใน IW และ RB ที่ตามหลังคำสั่งทางแยกทิ้งไป แต่ถ้ามีพฤติกรรม Not taken แสดงว่าการทำนายถูกต้อง ไม่ต้องดำเนินการใดๆทั้งสิ้น

5.9 วิธีบันทึกผลลัพธ์ของคำสั่งลง RB และ IW (Write back)

โปรเซสเซอร์จะบันทึกผลลัพธ์จากการประมวลผลคำสั่งลง RB พร้อมกับการฟอร์เวิร์ดข้อมูลไปยังคำสั่งที่รอคอยโอเปอแรนด์ใน IW โครงสร้างบางส่วนของ IW และ RB เพื่ออธิบายการบันทึกผลลัพธ์คำสั่งเป็นดังรูปที่ 5.14



รูปที่ 5.14 อธิบายวิธีการบันทึกผลลัพธ์คำสั่ง

- การบันทึกผลลัพธ์ลง RB

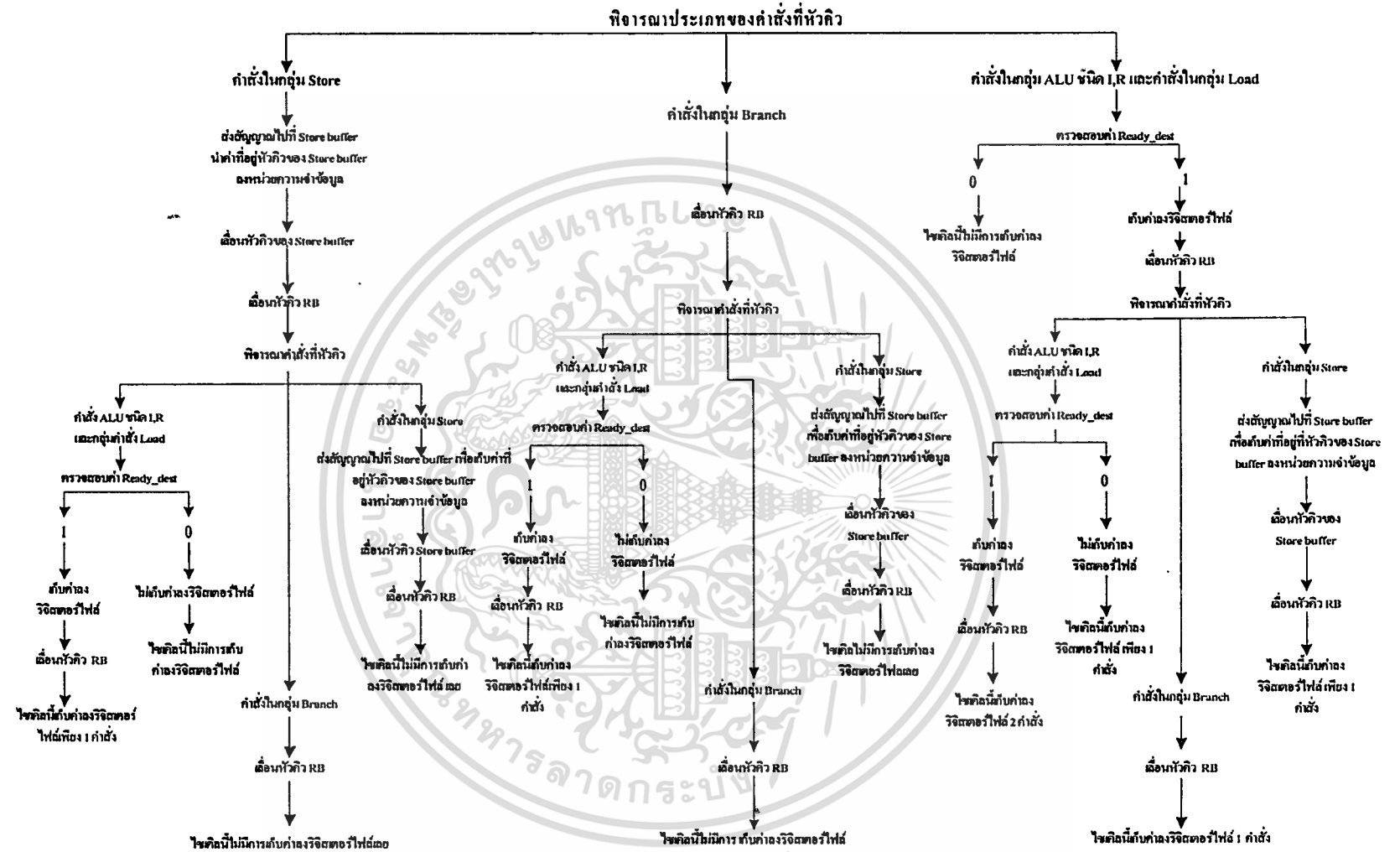
เมื่อหน่วยทำงานประมวลผลเสร็จแล้วจะบันทึกผลลัพธ์ลง RB หน่วยทำงานจะส่งค่าแท็กของคำสั่งและผลลัพธ์ไปยัง RB ซึ่ง RB จะนำค่าแท็กไปเปรียบเทียบกับ Dest_tag ใน RB ถ้าตรงกับเอนทรีใดก็จะนำผลลัพธ์ใส่ในฟิลด์ Dest_data ของเอนทรีนั้น และกำหนดค่า Ready_dest ให้เป็น 1

- การฟอร์เวิร์ดข้อมูล (Data forwarding)

เมื่อหน่วยทำงานประมวลผลเสร็จจะนำค่าแท็กและผลลัพธ์ของคำสั่งส่งไปยัง IW จากนั้น IW นำค่าแท็กไปเปรียบเทียบกับฟิลด์ Rs1_tag และ Rs2_tag ของทุกเอนทรีใน IW ที่ยังไม่ถูกส่งออกและโอเปอแรนด์ยังไม่พร้อม ถ้าตรงกันก็จะนำผลลัพธ์จากการประมวลผลใส่ใน Rs1_data และ Rs2_data จากนั้นกำหนดค่า Rs1_ready และ Rs2_ready ให้เป็น 1

5.10 วิธีการเก็บผลลัพธ์คำสั่งจาก RB ลงรีจิสเตอร์ไฟล์ (Result commit)

คำสั่งใน RB ที่ได้รับผลลัพธ์จากการประมวลผลจากหน่วยทำงานแล้ว (ซึ่งมีค่าในฟิลด์ Ready_dest = '1') RB จะเก็บผลลัพธ์เหล่านั้นลงริจิสเตอร์ไฟล์เมื่อคำสั่งเหล่านั้นเคลื่อนที่มาอยู่ที่หัวคิว โปรเซสเซอร์จะเก็บผลลัพธ์ลงริจิสเตอร์ไฟล์ไซเคิลละไม่เกิน 2 คำสั่ง โดยมีเงื่อนไขตามกระบวนการดังรูปที่ 5.15

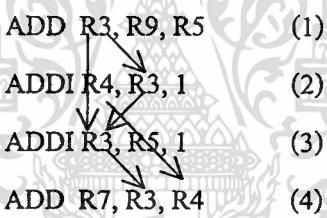


รูปที่ 5.15 ลำดับการเก็บผลลัพธ์คำสั่งจาก RB ลงรีจิสเตอร์ไฟล์

5.11 วิธีแก้ปัญหาข้อมูล

5.11.1 ปัญหาข้อมูลชนิด RAW คำสั่งที่เกิดปัญหา RAW คือคำสั่งที่ต้องรอผลลัพธ์จากคำสั่งก่อนหน้าซึ่งยังประมวลผลไม่เสร็จ ทำให้คำสั่งนั้นไม่สามารถประมวลผลได้เนื่องจากโอเปอแรนด์ไม่พร้อม จึงเกิดการหน่วงเวลาของคำสั่งลำดับหลังๆได้ หลักการแก้ปัญหา RAW คือ ต้องแยกคำสั่งที่เกิดปัญหาให้ห่างออกจากกัน (เพื่อไม่ให้รอคอยโอเปอแรนด์) แล้วพยายามหาคำสั่งอื่นมาแทรกระหว่างคำสั่งที่เกิดปัญหา ในงานวิจัยได้ใช้กลไกการส่งออกคำสั่งแบบส่งออกไม่เป็นลำดับและประมวลผลแบบไม่เป็นลำดับ ฉนั้นคำสั่งที่มีโอเปอแรนด์พร้อมจึงสามารถส่งออกคำสั่งไปประมวลผลได้ ส่วนคำสั่งที่ไม่พร้อมต้องรอคอยโอเปอแรนด์ใน IW หลักการดังกล่าวทำให้ลำดับโปรแกรมที่ประมวลผลที่หน่วยทำงานต่างไปจากลำดับของโปรแกรมต้นฉบับ เป็นการแยกคำสั่งที่เกิดปัญหา RAW ออกจากกันแล้วหาคำสั่งอื่นใส่ในช่องว่างนั้นแทน จึงไม่ทำให้เกิดการสูญเสียไซเคิล

5.11.2 ปัญหาข้อมูลชนิด WAR และ WAW ปัญหานี้เกิดขึ้นเมื่อกลไกในการประมวลเสร็จเป็นแบบไม่เรียงลำดับ (Out-of-Order completion) พิจารณาชุดคำสั่งดังรูปที่ 5.16



รูปที่ 5.16 ชุดคำสั่งที่เกิดปัญหาข้อมูลชนิด WAR และ WAW

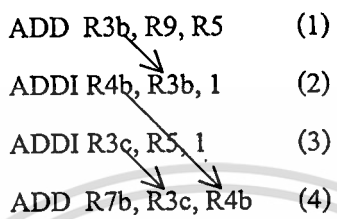
ชุดคำสั่งในรูป 5.16 เมื่อพิจารณาเฉพาะริจิสเตอร์ R3 เกิดปัญหาดังนี้

- คำสั่งที่ 2 รอคอยโอเปอแรนด์ R3 ของคำสั่งที่ 1 (RAW)
- คำสั่งที่ 1 และ 3 ใช้ริจิสเตอร์ปลายทางตัวเดียวกันคือ R3 (WAW)
- คำสั่งที่ 2 ใช้โอเปอแรนด์ต้นทางตัวที่ 1 คือ R3 ซึ่งเป็นตัวเดียวกับริจิสเตอร์ปลายทางของคำสั่ง 3
- คำสั่งที่ 4 รอคอยโอเปอแรนด์ R3 จากคำสั่งที่ 3 (RAW)

เพื่อขจัดปัญหา WAR และ WAW ต้องทำดังนี้

- บังคับให้คำสั่งที่ 1 ประมวลผลเสร็จก่อนคำสั่งที่ 3
- บังคับให้คำสั่งที่ 2 อ่านค่าโอเปอแรนด์ R3 ก่อนที่คำสั่งที่ 3 จะประมวลผลเสร็จ

การแก้ปัญหาเหล่านี้ด้วยการหน่วงเวลาตามวิธีสกอร์บอร์ด จะทำให้สูญเสียไซเคิลจำนวนมาก ในงานวิจัยได้ใช้การเปลี่ยนชื่อริจิสเตอร์ (Register renaming) แก้ปัญหา โดยการเปลี่ยนชื่อริจิสเตอร์ปลายทางของทุกคำสั่งที่เข้ามาประมวลผล จากชุดคำสั่งในรูปที่ 5.16 เมื่อเปลี่ยนชื่อริจิสเตอร์แล้วจะได้ชุดคำสั่งใหม่ดังรูปที่ 5.17

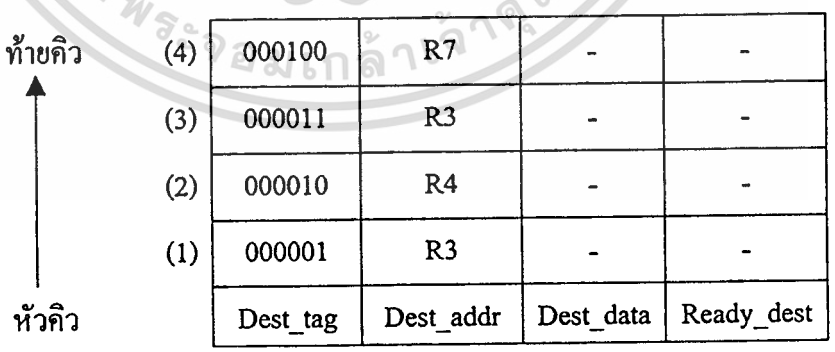


รูปที่ 5.17 ชุดคำสั่งเมื่อเปลี่ยนชื่อริจิสเตอร์เพื่อแก้ปัญหาข้อมูลชนิด WAR และ WAW

ชุดคำสั่งเมื่อเปลี่ยนชื่อริจิสเตอร์แล้วตามรูปที่ 5.17 ปัญหา WAR และ WAW จะถูกขจัดไปหมด ในงานวิจัย หลักการเปลี่ยนชื่อริจิสเตอร์ประกอบด้วย

- Reorder buffer
- Dest_tag

ค่า Dest_tag ถูกสร้างจากตัวถอดรหัสคำสั่ง เมื่อนำคำสั่งที่ถอดรหัสแล้วมาเก็บใน RB ทำให้แต่ละเอนทรีใน RB มีค่า Dest_tag ไม่ซ้ำกัน เปรียบเสมือนว่าริจิสเตอร์ปลายทางของแต่ละคำสั่งถูกเปลี่ยนให้เป็นค่า Dest_tag แทน ดังรูปที่ 5.18



(4)	000100	R7	-	-
(3)	000011	R3	-	-
(2)	000010	R4	-	-
(1)	000001	R3	-	-
	Dest_tag	Dest_addr	Dest_data	Ready_dest

รูปที่ 5.18 เอนทรีใน RB แต่ละเอนทรีมีค่า Dest_tag ไม่ซ้ำกัน

เมื่อแต่ละคำสั่งประมวลผลเสร็จแล้วจะนำผลลัพธ์ไปเก็บตาม Dest_tag ที่ระบุ คำสั่งที่ประมวลผลเสร็จก่อนจะเก็บผลลัพธ์ได้ทันที ไม่มีการหน่วงเวลาเกิดขึ้น ผลลัพธ์ของคำสั่งจะถูกเก็บลงริจิสเตอร์ไฟล์อย่างเป็นลำดับ

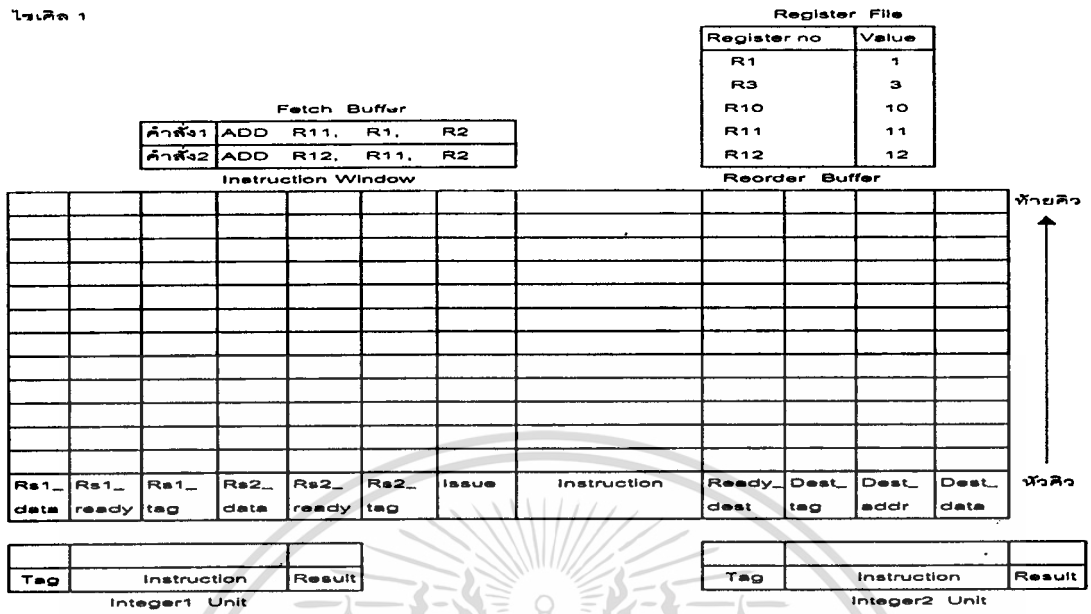
5.12 ตัวอย่างการประมวลผลคำสั่ง

ในส่วนนี้จะแสดงขั้นตอนการประมวลผลคำสั่งประเภท R-type ตามหลักการไดนามิกที่นำเสนอ คำสั่งชนิดนี้ต้องการโอเปอแรนด์ต้นทาง 2 ตัวในการประมวลผล พิจารณาชุดคำสั่งตามรูปที่ 5.19

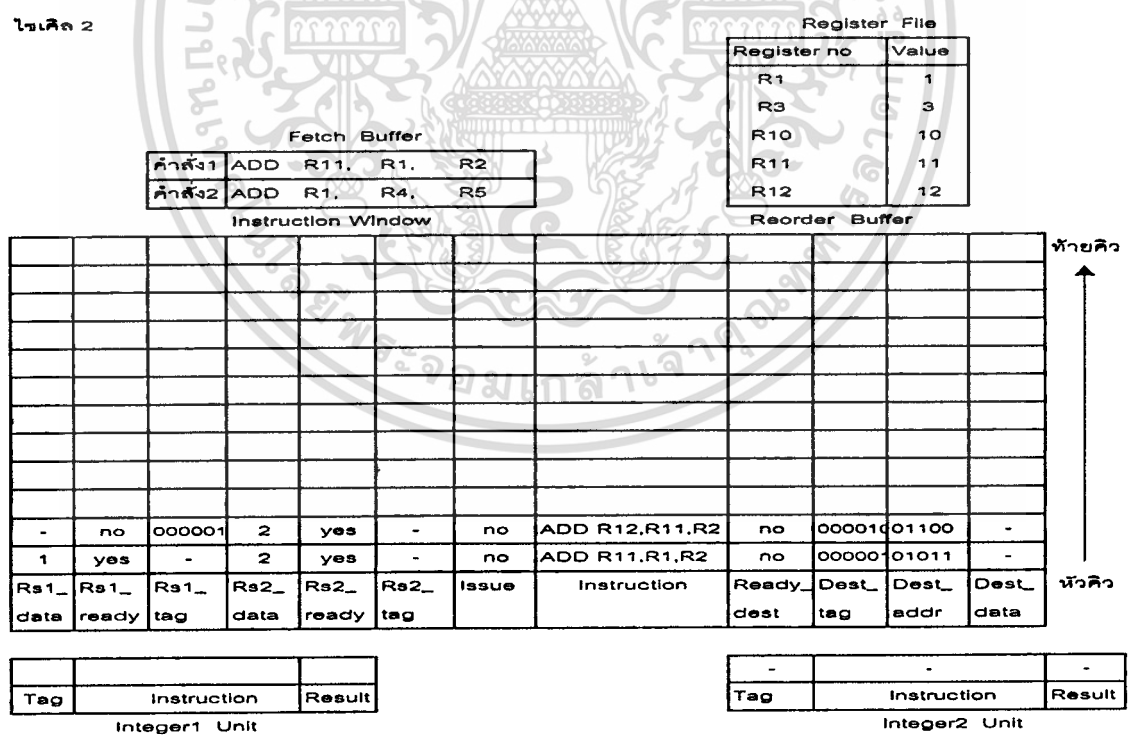
ลำดับ	คำสั่ง	ผลลัพธ์
1	ADD R11, R1, R2	R11=3
2	ADD R12, R11, R2	R12=5
3	ADD R11, R1, R12	R11=6
4	ADD R1, R4, R5	R1=9
5	ADD R1, R2, R3	R1=5
6	ADD R10, R1, R3	R10=8
7	ADD R10, R10, R3	R10=11
8	ADD R3, R1, R4	R3=9
9	ADD R12, R3, R10	R12=20
10	ADD R10, R12, R1	R10=25

รูปที่ 5.19 ชุดคำสั่งประเภท R-type สำหรับแสดงขั้นตอนการทำงานตามหลักการไดนามิกสเคจดูล

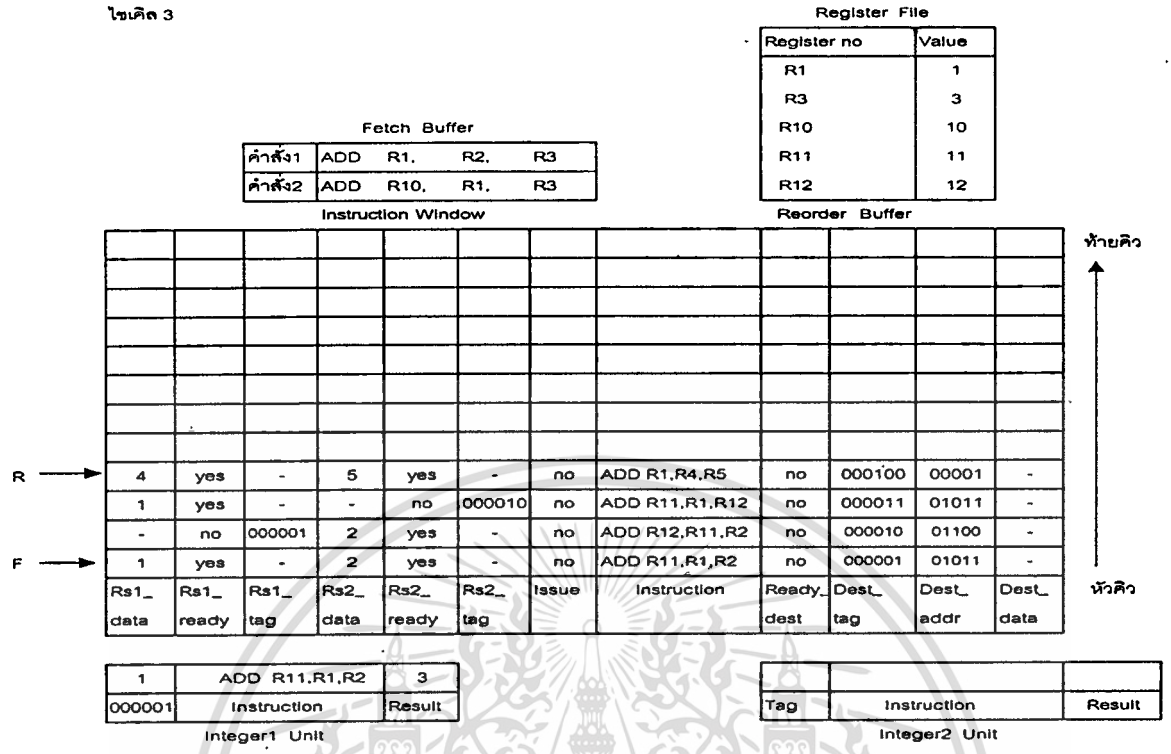
จากรูป 5.19 กำหนดค่าเริ่มต้นในริจิสเตอร์ไฟล์ R1=1 R2=2 R3=3 ... R31=31 หลังจากประมวลผลเสร็จแล้วทุกคำสั่ง จะได้ค่าในริจิสเตอร์ไฟล์ R1=5 R3=9 R10=25 R11=6 R12=20 ส่วนตัวอื่นๆ ค่าเหมือนเดิม และใช้เวลาทั้งหมด 15 ไซเคิล ซึ่งแสดงให้เห็นการทำงานแต่ละไซเคิลดังรูปที่ 5.20-5.34



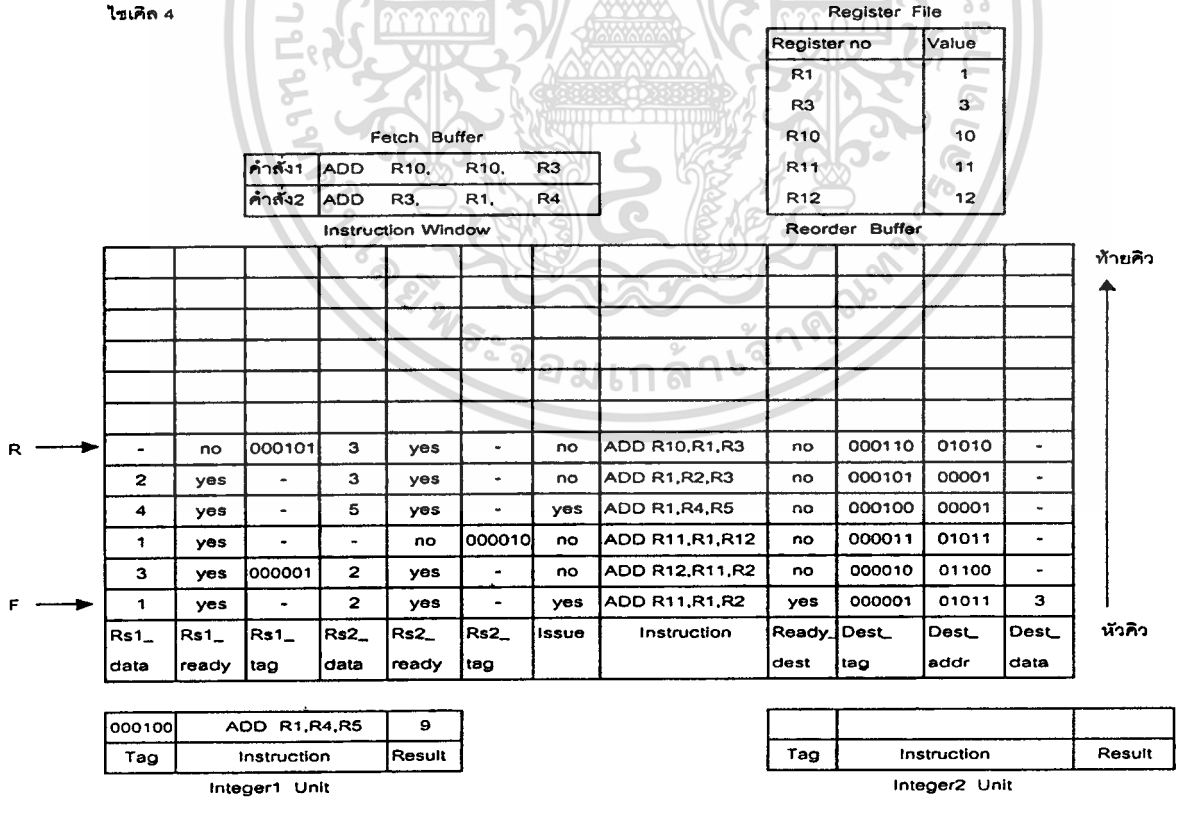
รูปที่ 5.20 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 1



รูปที่ 5.21 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 2



รูปที่ 5.22 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 3



รูปที่ 5.23 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 4

ไซเคิล 5

Register File .

Register no	Value
R1	1
R3	3
R10	10
R11	3
R12	12

Fetch Buffer

คำสั่ง1	ADD	R10,	R10,	R3
คำสั่ง2	ADD	R3,	R1,	R4

Instruction Window

Reorder Buffer

											ท้ายคิว
R →	-	no	000101	4	yes	-	no	ADD R3,R1,R4	no	001000	00011	-
	-	no	000110	3	yes	-	no	ADD R10,R10,R3	no	000111	01010	-
	-	no	000101	3	yes	-	no	ADD R10,R1,R3	no	000110	01010	-
	2	yes	-	3	yes	-	yes	ADD R1,R2,R3	no	000101	00001	-
	4	yes	-	5	yes	-	yes	ADD R1,R4,R5	yes	000100	00001	9
	1	yes	-	-	no	000010	no	ADD R11,R1,R12	no	000011	01011	-
F →	3	yes	000001	2	yes	-	yes	ADD R12,R11,R2	no	000010	01100	-
	Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data	หัวคิว

000010	ADD R12,R11,R2	5
Tag	Instruction	Result

Integer1 Unit

000101	ADD R1,R2,R3	5
Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.24 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 5

ไซเคิล 6

Register File

Register no	Value
R1	1
R3	3
R10	10
R11	3
R12	12

Fetch Buffer

คำสั่ง1	
คำสั่ง2	

Instruction Window

Reorder Buffer

											ท้ายคิว
R →	-	no	001001	-	no	000101	no	ADD R10,R12,R1	no	001010	01010	-
	-	no	001000	-	no	000111	no	ADD R12,R3,R10	no	001001	01100	-
	5	yes	000101	4	yes	-	no	ADD R3,R1,R4	no	001000	00011	-
	-	no	000110	3	yes	-	no	ADD R10,R10,R3	no	000111	01010	-
	5	yes	000101	3	yes	-	no	ADD R10,R1,R3	no	000110	01010	-
	2	yes	-	3	yes	-	yes	ADD R1,R2,R3	yes	000101	00001	-
	4	yes	-	5	yes	-	yes	ADD R1,R4,R5	yes	000100	00001	9
	1	yes	-	5	yes	000010	yes	ADD R11,R1,R12	no	000011	01011	-
F →	3	yes	000001	2	yes	-	yes	ADD R12,R11,R2	yes	000010	01100	-
	Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data	หัวคิว

Tag	Instruction	Result

Integer1 Unit

Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.25 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 6

ไซเคิล 7

Register File

Register no	Value
R1	1
R3	3
R10	10
R11	3
R12	5

Fetch Buffer

คำสั่ง1	
คำสั่ง2	

Instruction Window

Reorder Buffer

											ท้ายคิว
R → -	no	001001	5	no	000101	no	ADD R10,R12,R1	no	001010	01010	-
-	no	001000	-	no	000111	no	ADD R12,R3,R10	no	001001	01100	-
5	yes	000101	4	yes	-	no	ADD R3,R1,R4	no	001000	00011	-
-	no	000110	3	yes	-	no	ADD R10,R10,R3	no	000111	01010	-
5	yes	000101	3	yes	-	yes	ADD R10,R1,R3	no	000110	01010	-
2	yes	-	3	yes	-	yes	ADD R1,R2,R3	yes	000101	00001	5
4	yes	-	5	yes	-	yes	ADD R1,R4,R5	yes	000100	00001	9
F → 1	yes	-	5	yes	000010	yes	ADD R11,R1,R12	no	000011	01011	-
Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data	หัวคิว

000011	ADD R11,R1,R12	6
Tag	Instruction	Result

Integer1 Unit

000110	ADD R10,R1,R3	8
Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.26 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 7

ไซเคิล 8

Register File

Register no	Value
R1	1
R3	3
R10	10
R11	3
R12	5

Fetch Buffer

คำสั่ง1	
คำสั่ง2	

Instruction Window

Reorder Buffer

											ท้ายคิว
R → -	no	001001	5	no	000101	no	ADD R10,R12,R1	no	001010	01010	-
-	no	001000	-	no	000111	no	ADD R12,R3,R10	no	001001	01100	-
5	yes	000101	4	yes	-	yes	ADD R3,R1,R4	no	001000	00011	-
8	yes	000110	3	yes	-	no	ADD R10,R10,R3	no	000111	01010	-
5	yes	000101	3	yes	-	yes	ADD R10,R1,R3	yes	000110	01010	8
2	yes	-	3	yes	-	yes	ADD R1,R2,R3	yes	000101	00001	5
4	yes	-	5	yes	-	yes	ADD R1,R4,R5	yes	000100	00001	9
F → 1	yes	-	5	yes	000010	yes	ADD R11,R1,R12	yes	000011	01011	6
Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data	หัวคิว

001000	ADD R3,R1,R4	9
Tag	Instruction	Result

Integer1 Unit

Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.27 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 8

ไซเคิล 9

Register File

Register no	Value
R1	9
R3	3
R10	10
R11	6
R12	5

Fetch Buffer

คำสั่ง1	
คำสั่ง2	

Instruction Window

Reorder Buffer

											ท้ายคิว
R → -	no	001001	5	yes	000101	no	ADD R10,R12,R1	no	001010	01010	-
9	yes	001000	-	no	000111	no	ADD R12,R3,R10	no	001001	01100	-
5	yes	000101	4	yes	-	yes	ADD R3,R1,R4	yes	001000	00011	9
8	yes	000110	3	yes	-	no	ADD R10,R10,R3	no	000111	01010	-
5	yes	000101	3	yes	-	yes	ADD R10,R1,R3	yes	000110	01010	8
F → 2	yes	-	3	yes	-	yes	ADD R1,R2,R3	yes	000101	00001	5
Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data	หัวคิว

000111	ADD R10,R10,R3	11
Tag	Instruction	Result

Integer1 Unit

Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.28 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 9

ไซเคิล 10

Register File

Register no	Value
R1	5
R3	3
R10	8
R11	6
R12	5

Fetch Buffer

คำสั่ง1	
คำสั่ง2	

Instruction Window

Reorder Buffer

											ท้ายคิว
R → -	no	001001	5	yes	000101	no	ADD R10,R12,R1	no	001010	01010	-
9	yes	001000	11	yes	000111	no	ADD R12,R3,R10	no	001001	01100	-
5	yes	000101	4	yes	-	yes	ADD R3,R1,R4	yes	001000	00011	9
F → 8	yes	000110	3	yes	-	yes	ADD R10,R10,R3	yes	000111	01010	11
Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data	หัวคิว

Tag	Instruction	Result

Integer1 Unit

Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.29 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 10

ไซเคิล 11

Register File

Register no	Value
R1	5
R3	9
R10	11
R11	6
R12	5

Fetch Buffer

คำสั่ง1	
คำสั่ง2	

Instruction Window

Reorder Buffer

-	no	001001	5	yes	000101	no	ADD R10,R12,R1	no	001010	01010	-
9	yes	001000	11	yes	000111	yes	ADD R12,R3,R10	no	001001	01100	-
Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data

ท้ายคิว

หัวคิว

001001	ADD R12,R3,R10	20
Tag	Instruction	Result

Integer1 Unit

Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.30 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 11

ไซเคิล 12

Register File

Register no	Value
R1	5
R3	9
R10	11
R11	6
R12	5

Fetch Buffer

คำสั่ง1	
คำสั่ง2	

Instruction Window

Reorder Buffer

R →	20	yes	001001	5	yes	000101	no	ADD R10,R12,R1	no	001010	01010	-
F →	9	yes	001000	11	yes	000111	yes	ADD R12,R3,R10	yes	001001	01100	20
Rs1_ data	Rs1_ ready	Rs1_ tag	Rs2_ data	Rs2_ ready	Rs2_ tag	Issue	Instruction	Ready_ dest	Dest_ tag	Dest_ addr	Dest_ data

ท้ายคิว

หัวคิว

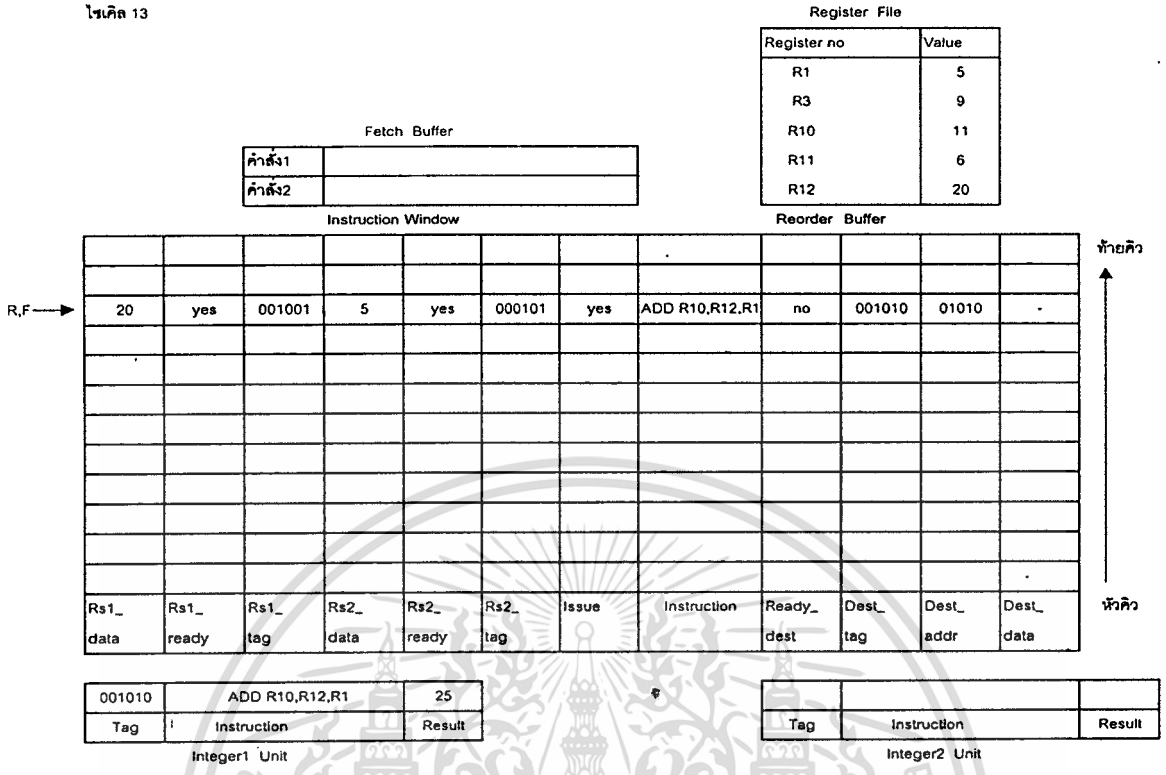
Tag	Instruction	Result

Integer1 Unit

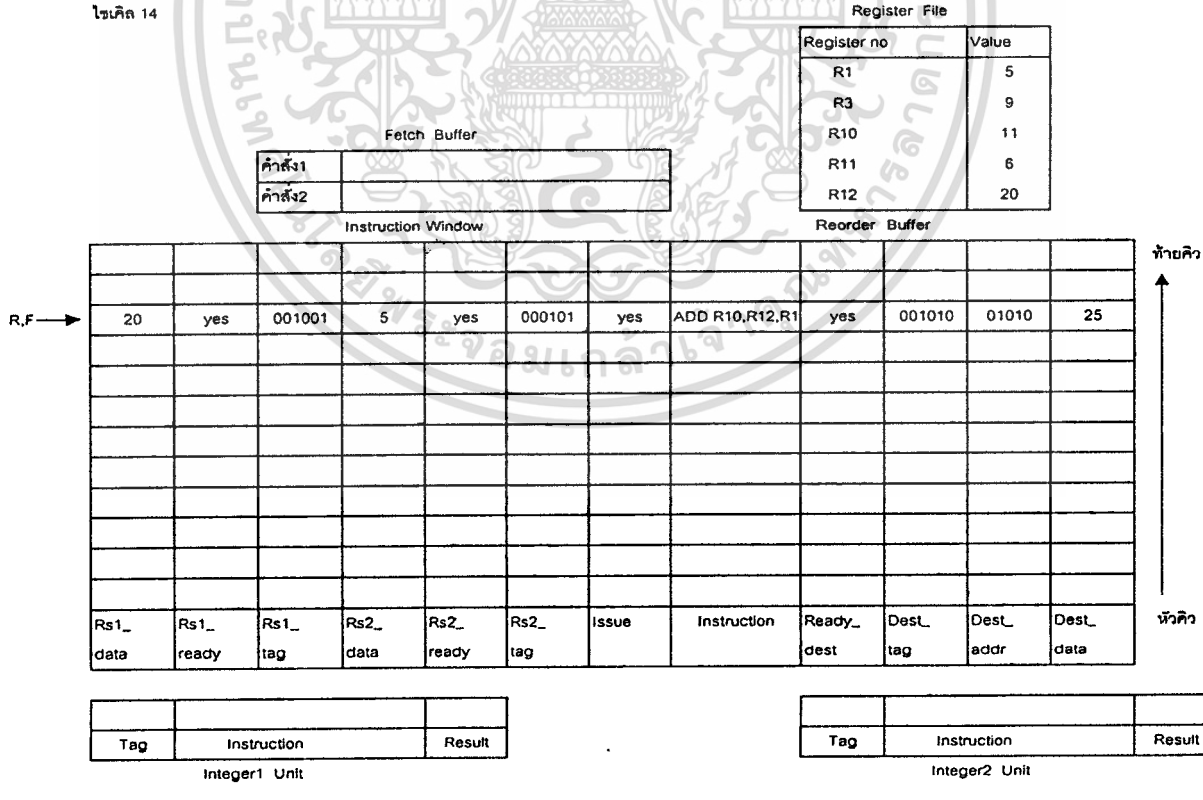
Tag	Instruction	Result

Integer2 Unit

รูปที่ 5.31 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 12

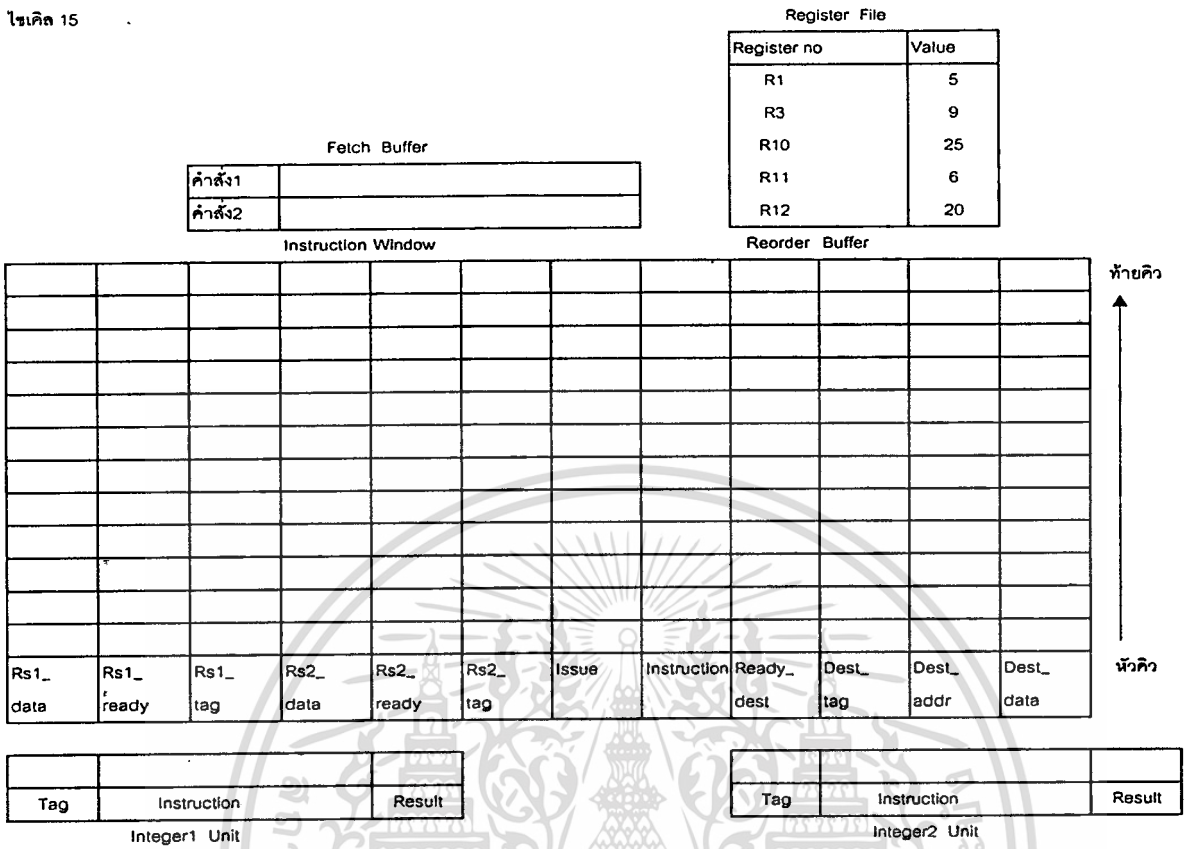


รูปที่ 5.32 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 13



รูปที่ 5.33 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 14

ไซเคิล 15



รูปที่ 5.34 การประมวลผลคำสั่งตามวิธีสเคจดูลที่นำเสนอในไซเคิลที่ 15

บทที่ 6
การทดสอบการปรับปรุงประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์

เนื้อหาในบทนี้เป็นการทดสอบประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ทั้งก่อนและหลังการปรับปรุงวิธีการไดนามิกสเคจดูล การทดสอบเริ่มด้วยการจำลองการทำงานของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ตามหลักการสกอร์บอร์ด และจำลองการทำงานภายใต้หลักการไดนามิกสเคจดูลที่นำเสนอโดยใช้ภาษาบรรยายฮาร์ดแวร์วีเอชดีแอล จากนั้นนำดีแอลเอ็กซ์ซูเปอร์สเกลลาร์มาทดสอบด้วยโปรแกรมทดสอบ 50 โปรแกรม วัดประสิทธิภาพการทำงานโดยพิจารณาจากจำนวนคำสั่งที่ประมวลผลได้ต่อไซเคิล 40 โปรแกรมแรกของโปรแกรมทดสอบประกอบด้วยการขึ้นต่อกันของคำสั่งทั้ง 3 ชนิดคือ True data dependency Antidependency และ Output dependency ไม่มีคำสั่งทางแยกเลยเพื่อใช้ทดสอบประสิทธิภาพการทำไดนามิกสเคจดูลที่มุ่งเน้นการแก้ปัญหาข้อมูลโดยเฉพาะซึ่งเป็นส่วนสำคัญของงานวิจัย ส่วนโปรแกรมทดสอบที่ 41-45 จะเป็นการทดสอบกับคำสั่งทางแยกที่เป็นลูปซับซ้อนซึ่งเป็นส่วนที่เพิ่มเติมขึ้นมาของงานวิจัย เนื่องจากความสำคัญของคำสั่งทางแยกที่มีผลต่อประสิทธิภาพของซูเปอร์สเกลลาร์ โปรแกรมที่ 46-50 ได้จากการนำโปรแกรมที่ 41-45 มาทำลูปอันโรล ผลที่ได้คือจะไม่มีคำสั่งทางแยกในชุดนี้ จุดประสงค์เพื่อให้ลักษณะการขึ้นต่อกันของคำสั่งใกล้เคียงกับสภาวะการทำงานจริงมากที่สุด และเพื่อให้มั่นใจได้ว่าผลลัพธ์การประมวลผลถูกต้อง จึงได้นำโปรแกรมทดสอบทั้งหมดไปทดสอบกับไปป์ไลน์ดีแอลเอ็กซ์จาก [13] เพื่อยืนยันผลลัพธ์การประมวลผล

6.1 ตัวแปรบอกประสิทธิภาพ

ประสิทธิภาพพิจารณาจากจำนวนคำสั่งต่อไซเคิล (Instruction Per Clock: IPC) จาก (6.1)

$$IPC = \frac{Total\ instructions}{Total\ clock\ cycles} \tag{6.1}$$

โดยที่ Total instructions คือจำนวนคำสั่งทั้งหมดในโปรแกรม และ
Total clock cycles คือ จำนวนไซเคิลที่ใช้ประมวลผลโปรแกรม

6.2 คุณสมบัติของโปรแกรมทดสอบและผลการทดสอบประสิทธิภาพ

โปรแกรมทดสอบทั้งหมดมี 50 โปรแกรมแบ่งเป็น 3 ส่วน ส่วนแรกเป็นโปรแกรมทดสอบ 40 โปรแกรมที่ผู้วิจัยได้สร้างขึ้นเอง ส่วนที่สองนำมาจาก [13] ส่วนที่สามเกิดจากนำส่วนที่สองมาทำ

ลูปอันโรล โดยในส่วนโปรแกรมทดสอบที่สร้างขึ้นเองนั้นเนื้อหาของโปรแกรมเน้นการขึ้นต่อกันทั้ง 3 ชนิด โดยการกำหนดตัวแปรเพื่ออธิบายลักษณะการขึ้นต่อกันของคำสั่งในแต่ละโปรแกรม และเพื่อให้ตัวแปรดังกล่าวแสดงถึงความแตกต่างกันของแต่ละโปรแกรมทดสอบ ในงานวิจัยจึงกำหนดให้ทุกโปรแกรมมีจำนวนคำสั่งเท่ากันจำนวน 600 คำสั่ง ตัวแปรที่กำหนดขึ้นเพื่ออธิบายลักษณะของโปรแกรมทดสอบมี 3 ตัวคือ

1. จำนวน True data dependency เป็นตัวแปรที่บอกจำนวนครั้งของการขึ้นต่อกันของคำสั่งประเภท True data dependency มีวิธีการหาคือ สำหรับคำสั่งปัจจุบันที่กำลังพิจารณากับอีก 6 คำสั่งถัดมานั้น ถ้ารีจิสเตอร์ปลายทางของคำสั่งปัจจุบันตรงกับรีจิสเตอร์ต้นทางของอีก 6 คำสั่งให้นับจำนวนที่ตรงกันนั้นเป็นจำนวน True data dependency แต่มีข้อยกเว้นคือ ถ้าคำสั่งใดภายใน 6 คำสั่งนั้นมีรีจิสเตอร์ปลายทางตรงกับคำสั่งปัจจุบัน การนับจำนวน True data dependency จะจำกัดเฉพาะจากคำสั่งปัจจุบันถึงคำสั่งก่อนหน้าที่รีจิสเตอร์ปลายทางตรงกับคำสั่งปัจจุบันเท่านั้น จากนั้นจึงเปลี่ยนมาพิจารณาคำสั่งต่อไปทุกๆคำสั่งให้เป็นคำสั่งปัจจุบันคู่กับอีก 6 คำสั่งถัดไปเรื่อยๆตามเงื่อนไขข้างต้นจนจบโปรแกรม และการนับจำนวน True data dependency ให้นับสะสม

2. จำนวน Antidependency เป็นตัวแปรที่บอกจำนวนครั้งของการขึ้นต่อกันของคำสั่งประเภท Antidependency มีวิธีการหาคือ สำหรับคำสั่งปัจจุบันที่กำลังพิจารณากับอีก 6 คำสั่งที่อยู่ก่อนหน้าถ้ารีจิสเตอร์ปลายทางของคำสั่งปัจจุบันตรงกับรีจิสเตอร์ต้นทางของอีก 6 คำสั่งให้นับจำนวนที่ตรงกันนั้นเป็นจำนวน Antidependency แต่มีข้อยกเว้นคือ ถ้าคำสั่งใดภายใน 6 คำสั่งนั้นมีรีจิสเตอร์ปลายทางตรงกับคำสั่งปัจจุบัน การนับจำนวน Antidependency จะจำกัดเฉพาะจากคำสั่งปัจจุบันขึ้นไปก่อนถึงคำสั่งนั้น (ซึ่งรีจิสเตอร์ปลายทางตรงกับคำสั่งปัจจุบัน) เท่านั้น จากนั้นจึงเปลี่ยนมาพิจารณาคำสั่งต่อไปทุกๆคำสั่งให้เป็นคำสั่งปัจจุบันคู่กับอีก 6 คำสั่งที่อยู่ก่อนหน้าไปเรื่อยๆตามเงื่อนไขข้างต้นจนจบโปรแกรม และการนับจำนวน Antidependency ให้นับสะสม

3. จำนวน Output dependency เป็นตัวแปรที่บอกถึงจำนวนครั้งของคำสั่งประเภท Output dependency มีวิธีการพิจารณาคือ ให้พิจารณาที่ทุกคำสั่ง จากรีจิสเตอร์ปลายทางของคำสั่งปัจจุบันที่กำลังพิจารณา ให้มองหาคำสั่งถัดมาอีก 6 คำสั่ง ถ้าใน 6 คำสั่งนั้นมีคำสั่งที่ใช้รีจิสเตอร์ปลายทางตัวเดียวกับคำสั่งปัจจุบันอย่างน้อย 1 ตัว ให้นับเพิ่มจำนวนครั้งของการเกิด Output dependency

ตัวอย่างการหาจำนวน True data dependency Antidependency และ Output dependency พิจารณาการประมวลผลจากชุดคำสั่งต่อไปนี้

(1) ADD R3, R1, R2

(2) ADD R4, R3, R7

(3) ADD R7, R3, R4

(4) ADD R3, R4, R9

(5) ADD R4, R0, R7

(6) ADD R2, R3, R4

(7) ADD R7, R2, R10

(8) ADD R10, R7, R4

(9) ADD R9, R2, R3

(10) ADD R4, R10, R11

ผลการพิจารณาแสดงดังตารางที่ 6.1-6.3 ซึ่งจะได้จำนวน True data dependency เท่ากับ 13 จำนวน Antidependency เท่ากับ 10 และจำนวน Output dependency เท่ากับ 4

ตารางที่ 6.1 ผลการหาจำนวน True data dependency

คำสั่งที่พิจารณา	คำสั่งที่ 1		คำสั่งที่ 2		คำสั่งที่ 3		คำสั่งที่ 4		คำสั่งที่ 5		คำสั่งที่ 6		คำสั่งที่ 7		คำสั่งที่ 8		คำสั่งที่ 9		คำสั่งที่ 10		จำนวน
	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	
1				/	/																2
2						/	/														2
3									/												1
4											/						/				2
5												/				/					2
6													/								2
7															/						1
8																			/		1
9																					-
10																					-
จำนวน True data dependency																					13

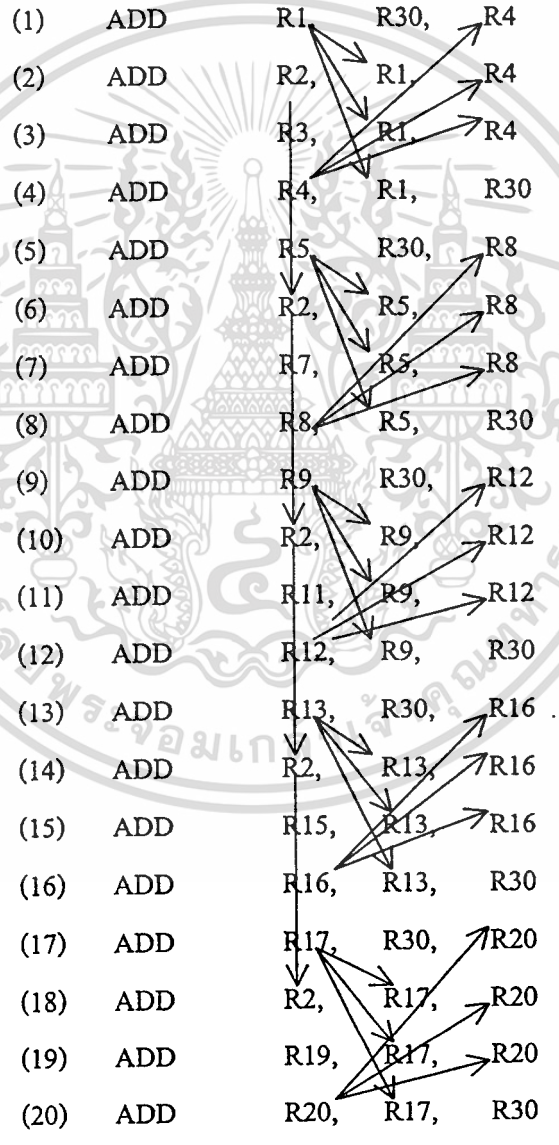
ตารางที่ 6.2 ผลการหาจำนวน Antidependency

คำสั่งที่พิจารณา	คำสั่งที่ 1		คำสั่งที่ 2		คำสั่งที่ 3		คำสั่งที่ 4		คำสั่งที่ 5		คำสั่งที่ 6		คำสั่งที่ 7		คำสั่งที่ 8		คำสั่งที่ 9		คำสั่งที่ 10		จำนวน
	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	Op1	Op2	
1																					-
2																					-
3				/																	1
4			/		/																2
5						/	/														2
6		/																			1
7									/												1
8													/								1
9								/													1
10																/					1
จำนวน Antidependency																					10

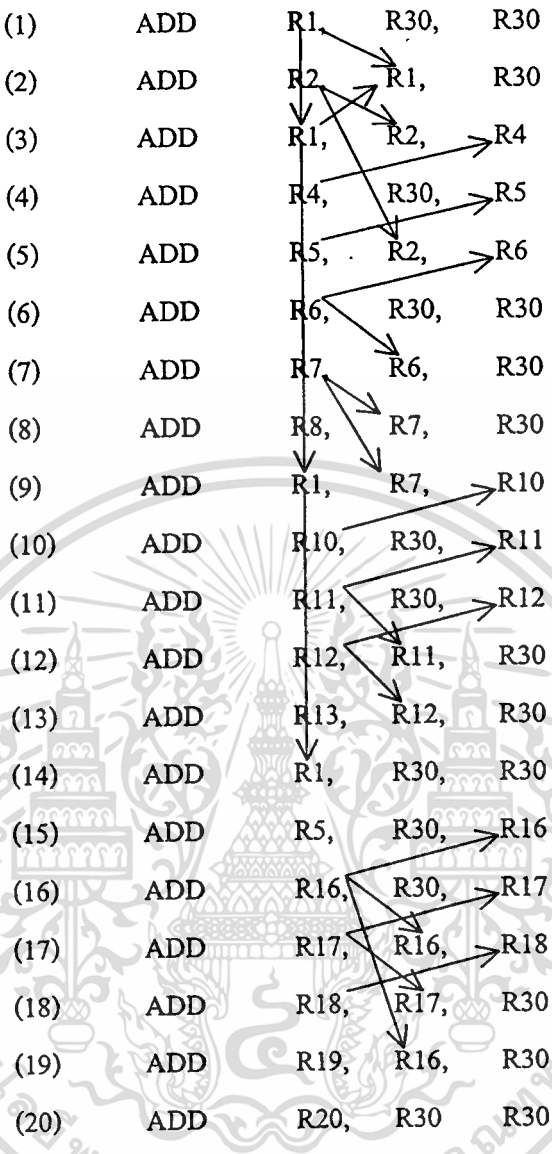
ตารางที่ 6.3 ผลการหาจำนวน Output dependency

คำสั่งที่พิจารณา	คำสั่งที่ 1	คำสั่งที่ 2	คำสั่งที่ 3	คำสั่งที่ 4	คำสั่งที่ 5	คำสั่งที่ 6	คำสั่งที่ 7	คำสั่งที่ 8	คำสั่งที่ 9	คำสั่งที่ 10	ปริมาณ
1				/							1
2					/						1
3							/				1
4											-
5										/	1
6											-
7											-
8											-
9											-
10											-
จำนวน Output dependency											4

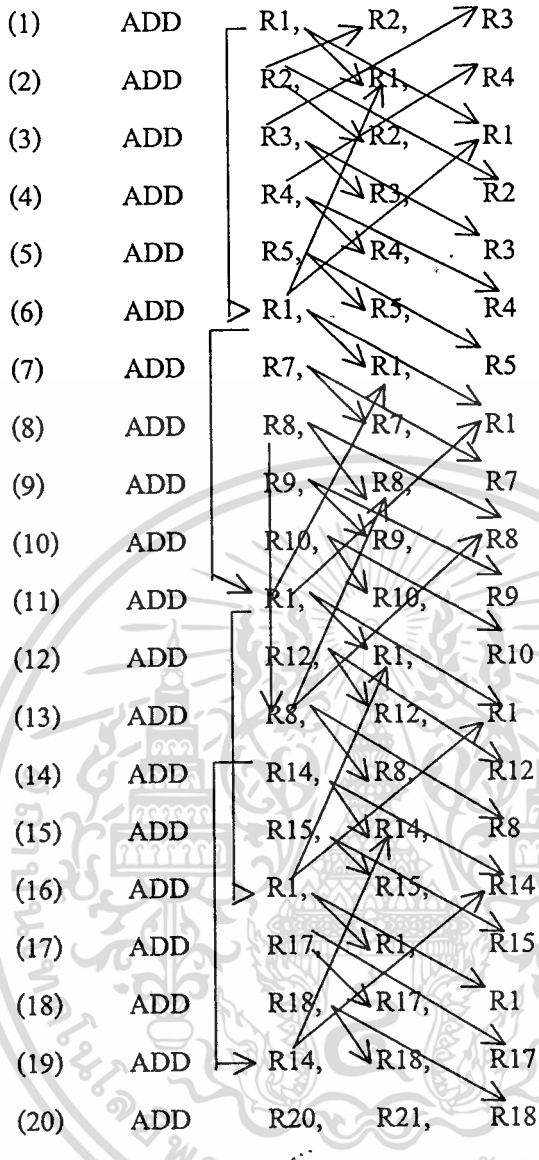
โปรแกรมทดสอบที่สร้างขึ้นเอง 40 โปรแกรมแบ่งออกเป็น 2 กลุ่ม กลุ่มแรกมี 4 ชุด ในแต่ละชุดมี 8 โปรแกรม แต่ละชุดมีรูปแบบการขึ้นต่อกันของข้อมูลแตกต่างกันโดยสิ้นเชิง โดย 8 โปรแกรมที่อยู่ในชุดเดียวกันจะมีลักษณะการขึ้นต่อกันของข้อมูลที่คล้ายคลึงกันโดยมีต้นแบบของแต่ละชุดดังรูปที่ 6.1-6.4 ทุกๆ 20 คำสั่งใน 600 คำสั่งมีรูปแบบตายตัว เช่น โปรแกรมชุดที่ 1 ซึ่งมีต้นแบบดังรูปที่ 6.1 โปรแกรมแรกในชุดนี้ทุกๆ 20 คำสั่งจะมีรูปแบบตามรูปที่ 6.1 อีก 7 โปรแกรมที่เหลือได้จากการเปลี่ยนแปลงจำนวนการขึ้นต่อกันของข้อมูลใน 20 คำสั่งของโปรแกรมต้นแบบ ซึ่งทำได้โดยลดการเชื่อมโยงระหว่างคำสั่งลง ทุกๆ 20 คำสั่งของแต่ละโปรแกรมจะมีรูปแบบตายตัว กลุ่มที่สองเป็นโปรแกรมที่ไม่เจาะจงรูปแบบการขึ้นต่อกันของข้อมูลมีทั้งหมด 8 โปรแกรม



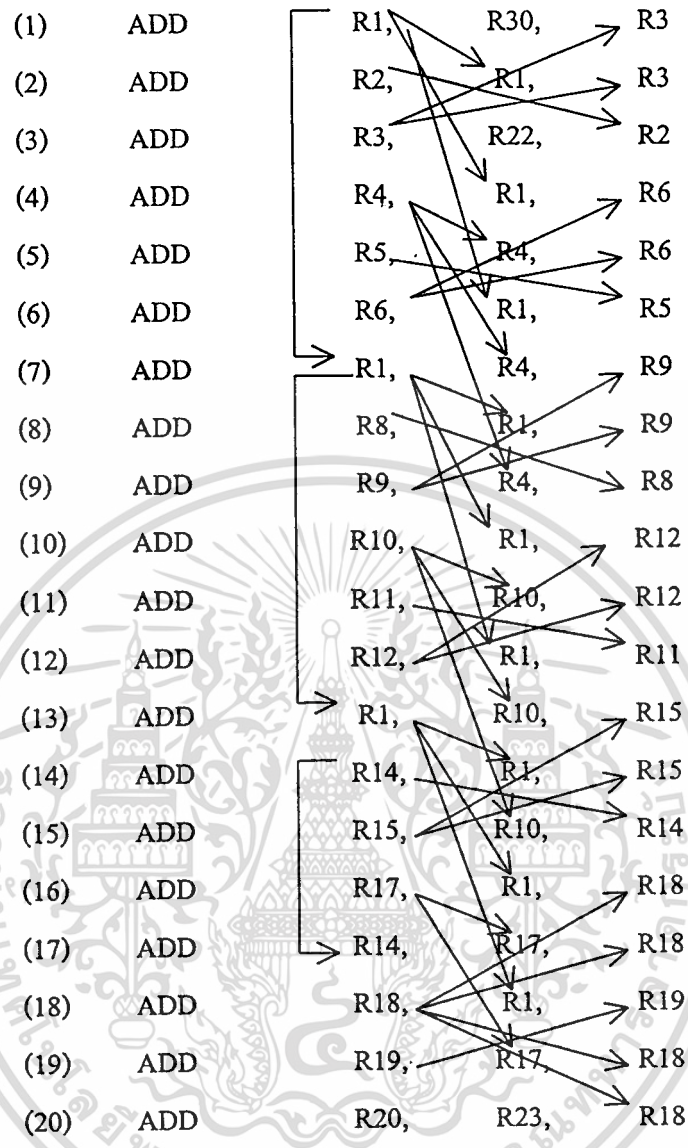
รูปที่ 6.1 รูปแบบการขึ้นต่อกันของข้อมูลของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 1



รูปที่ 6.2 รูปแบบการขึ้นต่อกันของข้อมูลของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 2



รูปที่ 6.3 รูปแบบการขึ้นต่อกันของข้อมูลของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 3



รูปที่ 6.4 รูปแบบการขึ้นต่อกันของข้อมูลของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 4

คุณสมบัติของโปรแกรมทดสอบและผลการทดสอบประสิทธิภาพของโปรแกรมทดสอบทั้ง 40 โปรแกรมแสดงดังตารางที่ 6.4-6.8

ตารางที่ 6.4 คุณสมบัติของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 1 และผลการทดสอบประสิทธิภาพ

คุณสมบัติ	โปรแกรมทดสอบที่							
	1	2	3	4	5	6	7	8
จำนวน True data dependency (RAW)	450	180	450	180	450	180	450	180
จำนวน Antidependency (WAR)	450	450	180	180	450	450	180	180
จำนวน Output dependency (WAW)	149	149	149	149	60	60	60	60
IPC วิธีของสกอร์บอร์ด	0.992	0.997	0.992	0.997	1.235	1.412	1.235	1.242
IPC ตามการปรับปรุงไดนามิกสเคจดูล	1.899	1.917	1.905	1.923	1.911	1.923	1.923	1.935

ตารางที่ 6.5 คุณสมบัติของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 2 และผลการทดสอบประสิทธิภาพ

คุณสมบัติ	โปรแกรมทดสอบที่							
	9	10	11	12	13	14	15	16
จำนวน True data dependency (RAW)	330	180	330	180	330	180	330	180
จำนวน Antidependency (WAR)	300	300	150	150	300	300	150	150
จำนวน Output dependency (WAW)	90	90	90	90	30	30	30	30
IPC วิธีของสกอร์บอร์ด	1.038	1.045	1.038	1.045	1.230	1.319	1.232	1.333
IPC ตามการปรับปรุงไดนามิกสเคจดูล	1.899	1.917	1.905	1.923	1.917	1.923	1.917	1.942

ตารางที่ 6.6 คุณสมบัติของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 3 และผลการทดสอบประสิทธิภาพ

คุณสมบัติ	โปรแกรมทดสอบที่							
	17	18	19	20	21	22	23	24
จำนวน True data dependency (RAW)	1080	720	960	660	450	360	150	1020
จำนวน Antidependency (WAR)	448	208	118	148	149	88	90	0
จำนวน Output dependency (WAW)	179	89	29	0	119	90	60	0
IPC วิธีของสกอร์บอร์ด	0.398	0.682	0.484	0.789	0.765	1.307	1.408	0.634
IPC ตามการปรับปรุงไดนามิกสเคจดูล	1.796	1.857	1.791	1.835	1.929	1.929	1.948	1.796

ตารางที่ 6.7 คุณสมบัติของโปรแกรมทดสอบกลุ่มที่ 1 ชุดที่ 4 และผลการทดสอบประสิทธิภาพ

คุณสมบัติ	โปรแกรมทดสอบที่							
	25	26	27	28	29	30	31	32
จำนวน True data dependency (RAW)	660	420	300	450	218	517	316	218
จำนวน Antidependency (WAR)	658	329	509	507	304	414	193	711
จำนวน Output dependency (WAW)	90	0	30	90	30	90	0	30
IPC วิธีของสกอร์บอร์ด	0.794	1.319	1.101	1.104	1.314	0.925	1.419	1.211
IPC ตามการปรับปรุงไดนามิกสเคจดูล	1.875	1.911	1.917	1.917	1.923	1.893	1.967	1.923

ตารางที่ 6.8 คุณสมบัติของโปรแกรมทดสอบกลุ่มที่ 2 และผลการทดสอบประสิทธิภาพ

คุณสมบัติ	โปรแกรมทดสอบที่							
	33	34	35	36	37	38	39	40
จำนวน True data dependency (RAW)	480	459	592	547	169	242	725	628
จำนวน Antidependency (WAR)	398	376	522	610	60	241	495	0
จำนวน Output dependency (WAW)	82	118	166	190	34	59	50	0
IPC วิธีของสกอร์บอร์ด	1.143	0.951	0.664	0.704	1.500	1.227	0.521	0.332
IPC ตามการปรับปรุงไดนามิกสเคจดูล	1.813	1.863	1.775	1.744	1.929	1.929	1.622	1.667

สำหรับโปรแกรมทดสอบที่ 41-45 นำมาจาก [13] ซึ่งมีคุณสมบัติและได้ผลการทดสอบดังตารางที่ 6.9

ตารางที่ 6.9 คุณสมบัติของโปรแกรมทดสอบแบบคำสั่งทางแยกและผลการทดสอบประสิทธิภาพ

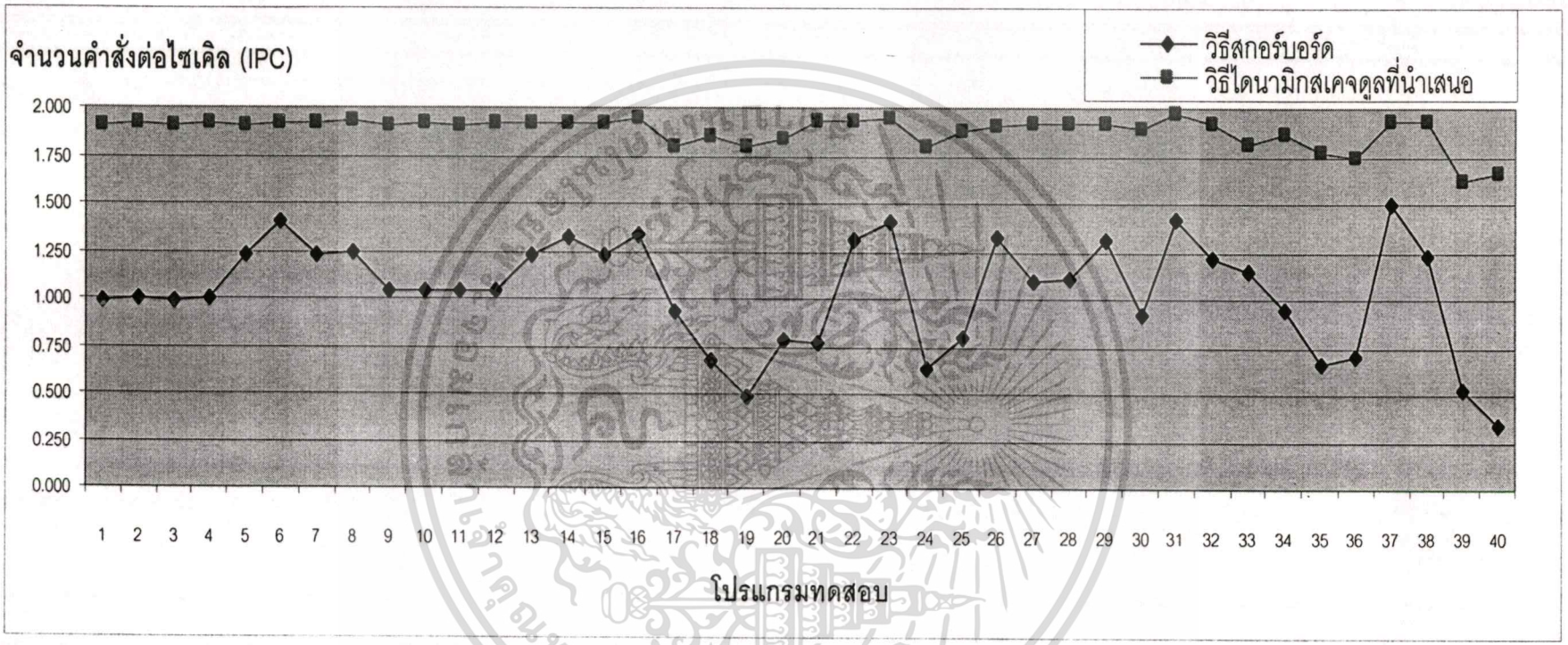
โปรแกรมทดสอบ	จำนวนคำสั่งทั้งหมดที่ถูกประมวลผล	%Conditional Branch	%Unconditional Branch	IPC
Fibonacci	67	14.93	13.43	1.108
Factorial	370	20.54	17.57	1.080
Bubble sort	235	24.26	2.98	1.200
Special number	20587	20.02	19.64	1.094
Armstrong number	5194	19.33	15.65	1.162

ตารางที่ 6.10 คุณสมบัติของโปรแกรมทดสอบที่ 46-50 และผลการทดสอบประสิทธิภาพ

คุณสมบัติ	โปรแกรมทดสอบที่				
	46 factorail	47 bubble sort	48 Fibonacci	49 Armstrong number	50 Special number
จำนวน True data dependency (RAW)	107	38	45	1434	4333
จำนวน Antidependency (WAR)	112	73	38	1492	4044
จำนวน Output dependency (WAW)	203	122	40	2974	12156
IPC วิธีของสกอร์บอร์ด	0.744	0.600	0.750	0.768	0.747
IPC ตามการปรับปรุง	1.762	1.600	1.200	1.955	1.989

6.3 การเปรียบเทียบประสิทธิภาพ

ผลการทดสอบโปรแกรมที่ 1-40 สามารถนำมาเปรียบเทียบความแตกต่างของประสิทธิภาพระหว่างวิธีสกอร์บอร์ดและวิธีไดนามิกสเคจดูลที่นำเสนอได้ดังรูปที่ 6.5



รูปที่ 6.5 กราฟเปรียบเทียบค่า IPC ระหว่างวิธีสกอร์บอร์ดและวิธีไดนามิกสเคจดูลที่นำเสนอ

6.4 สรุปผลการทดสอบประสิทธิภาพ

จากผลการทดสอบ 40 โปรแกรมแรกพบว่าประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์หลังจากปรับปรุงไดนามิกสเคจดูลให้ประสิทธิภาพการประมวลผลดีกว่าสกอร์บอร์ดมาก โดยสังเกตได้จาก IPC ที่แตกต่างกันอย่างเห็นได้ชัด ค่า IPC ของสกอร์บอร์ดอยู่ในช่วง 0.332-1.500 ส่วน IPC ของวิธีการไดนามิกสเคจดูลที่นำเสนออยู่ในช่วง 1.622-1.967

ประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ที่ปรับปรุงไดนามิกสเคจดูลลิงจะลดลงเล็กน้อยเมื่อชุดคำสั่งมีการขึ้นต่อกันแบบ True data dependency มากๆ แต่ประสิทธิภาพโดยรวมเมื่อการขึ้นต่อกันของข้อมูลมากขึ้นไม่ลดลงมากนักหรือเกือบคงที่ ซึ่งต่างจากวิธีสกอร์บอร์ดเมื่อการขึ้นต่อกันของข้อมูลเพิ่มขึ้นประสิทธิภาพจะลดลงอย่างมาก ซึ่งแสดงให้เห็นว่าวิธีใหม่ที่นำเสนอมีความเสถียรภาพในการประมวลผล และสามารถเห็นได้ชัดเจนจากกราฟดังรูป 6.5

โปรแกรมที่ 41-45 เป็นโปรแกรมที่เกี่ยวข้องกับคำสั่งทางแยกลูปซับซ้อนซึ่งเป็นส่วนที่เพิ่มเติมของงานวิจัย และโปรเซสเซอร์ที่ออกแบบใช้การทำนายทางแยกแบบสแตติกโดยทำนายพฤติกรรมของคำสั่งทางแยกว่า Not taken จากการทดสอบพบว่าประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ลดลงอย่างเห็นได้ชัด โดย IPC อยู่ระหว่าง 1.080-1.200 ประสิทธิภาพที่ลดลงมากเนื่องจากการสูญเสียไซเคิลจากการประมวลผลคำสั่งทางแยก ซึ่งแสดงให้เห็นว่าคำสั่งทางแยกมีผลต่อประสิทธิภาพของซูเปอร์สเกลลาร์โปรเซสเซอร์มาก สามารถเพิ่มประสิทธิภาพการประมวลผลของโปรเซสเซอร์สำหรับคำสั่งทางแยกโดยใช้หลักการทำนายทางแยกแบบไดนามิก (Dynamic branch prediction)

โปรแกรมที่ 46-50 ผลการทดสอบได้ผลเช่นเดียวกับโปรแกรมที่ 1-40

บทที่ 7
สรุปผลการวิจัยและข้อเสนอแนะ

7.1 สรุปผลการวิจัย

7.1 สรุปงานวิจัย

การทำไดนามิกสเคจดูลเป็นส่วนสำคัญมากส่วนหนึ่งจากหลายๆส่วนในการเพิ่มประสิทธิภาพของซูเปอร์สเกลลาร์โปรเซสเซอร์ ไดนามิกสเคจดูลเป็นกลไกทางฮาร์ดแวร์ในการหาคำสั่งที่เหมาะสมเพื่อส่งออกไปประมวลผลที่หน่วยทำงาน โดยมีจุดมุ่งหมายเพื่อให้การประมวลผลข้อมูลมีความถูกต้องและรวดเร็ว และต้องมีวิธีแก้ปัญหาข้อมูลที่เกิดขึ้นด้วย ขั้นตอนของการวิจัยประกอบด้วยการทำงาน 2 ส่วนหลักคือ ส่วนแรกศึกษาหลักการไดนามิกสเคจดูลของสกอร์บอร์ด จากนั้นทำการออกแบบเพื่อนำหลักการดังกล่าวประยุกต์ใช้กับดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ แล้วจึงจำลองการทำงานด้วยภาษาบรรยายฮาร์ดแวร์วีเอชดีแอลเพื่อคำนวณหาประสิทธิภาพการสเคจดูลคำสั่งของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ การทำงานส่วนที่สองเริ่มจากออกแบบการทำไดนามิกสเคจดูลใหม่เพื่อปรับปรุงประสิทธิภาพของดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ และทำการทดสอบประสิทธิภาพ การทดสอบประสิทธิภาพการทำไดนามิสเคจดูลของงานวิจัยจะเน้นที่ประสิทธิภาพการทำสเคจดูลข้อมูลให้มีความถูกต้องและรวดเร็ว โดยไม่ได้คำนึงถึงผลของคำสั่งทางแยกมากนัก หรือเน้นทดสอบกับคำสั่งที่ไม่มีการเปลี่ยนแปลงโปรแกรมเคาน์เตอร์ จากผลการทำงานทั้ง 2 ส่วนและผลการทดสอบพบว่าวิธีไดนามิกสเคจดูลที่นำเสนอให้ประสิทธิภาพการประมวลผลสูงกว่าและมีเสถียรภาพกว่าหลักการของสกอร์บอร์ดอย่างชัดเจน และประหยัดฮาร์ดแวร์กว่าด้วยการลดจำนวนหน่วยทำงานเลขจำนวนเต็มจาก 8 หน่วยเหลือเพียง 2 หน่วย ลดพอร์ตการอ่านเขียนริจิสเตอร์ไฟล์ สามารถรองรับอินเตอร์รัพรวมทั้งการทำนายทางแยกได้ดี และลดการหน่วงเวลาเนื่องจากปัญหาข้อมูลขึ้นต่อกัน อย่างไรก็ตามในดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ที่ปรับปรุงใหม่นอกเหนือจากการทำไดนามิกสเคจดูลแล้วยังได้เพิ่มส่วนการทำนายทางแยกแบบสแตติกอีกด้วย โดยคาดหมายพฤติกรรมของคำสั่งทางแยกล่วงหน้าว่า Not taken สำหรับโปรแกรมทดสอบส่วนหนึ่งนำมาจาก [13] เพื่อทดสอบผลของคำสั่งทางแยกที่มีผลต่อประสิทธิภาพของโปรเซสเซอร์ ผลการทดสอบด้านการประมวลผลคำสั่งทางแยกปรากฏว่าดีแอลเอ็กซ์ซูเปอร์สเกลลาร์ที่ปรับปรุงมีประสิทธิภาพลดลงแต่ยังมีประสิทธิภาพสูงกว่าโปรเซสเซอร์ที่แสดงไว้ในงานวิจัย [13] ซึ่งใช้หลักการทำนายทางแยกแบบเดียวกัน ผลการทดสอบดังกล่าวแสดงให้เห็นถึงความสำคัญของประสิทธิภาพการทำนายทางแยกที่มีผลต่อซูเปอร์สเกลลาร์โปรเซสเซอร์

ประสิทธิภาพของซูเปอร์สเกลลาร์จะดีมากเพียงใดขึ้นกับองค์ประกอบหลายส่วน นอกเหนือจากการทำไดนามิกสเคจดูลแล้ว ควรมีการทำนายทางแยกแบบไดนามิกและการทำสเป็กคูเลทีฟ (Speculative) ด้วย ประสิทธิภาพที่ได้จะสูงขึ้นมาก อย่างไรก็ตามในการออกแบบโปรเซสเซอร์ควรเน้นการออกแบบทั้งฮาร์ดแวร์และซอฟแวร์ควบคู่กันไป เพื่อไม่ให้ภาระการทำงานตกอยู่กับฮาร์ดแวร์มากเกินไปซึ่งอาจทำให้ฮาร์ดแวร์มีลอจิกการทำงานซับซ้อน ส่งผลให้มีดีเลย์มากขึ้น เช่น การทำสเคจดูลนั้นส่วนหนึ่งควรให้คอมไพเลอร์ทำการสเคจดูลขั้นแรกก่อน (Static schedule) เช่น อาจมีการทำลูปอันโรล (Loop unrolling) และจัดเรียงคำสั่งให้เหมาะสมบางส่วน แล้วจึงส่งต่อมายังฮาร์ดแวร์ (Dynamic schedule) ในสิ่งที่คอมไพเลอร์ทำไม่ได้ เช่น ตำแหน่งการอ้างอิงหน่วยความจำ ผลลัพธ์ของคำสั่งทางแยก และจัดเรียงคำสั่งใหม่ให้เหมาะสม

จากงานวิจัยสรุปความแตกต่างระหว่างวิธีสกอร์บอร์ดและวิธีที่นำเสนอได้ดังตารางที่ 7.1

ตารางที่ 7.1 ความแตกต่างระหว่างวิธีสกอร์บอร์ดและวิธีปรับปรุงการทำไดนามิกสเคจูล

ข้อเปรียบเทียบ	สกอร์บอร์ด	วิธีที่นำเสนอ
เงื่อนไขการส่งออกคำสั่งไปประมวลผล	ส่งออกเป็นลำดับและประมวลผลเสร็จแบบไม่เป็นลำดับ	ส่งออกและประมวลผลเสร็จแบบไม่เป็นลำดับ
ริจิสเตอร์ไฟล์	20 พอร์ตสำหรับการอ่านและ 9 พอร์ตสำหรับเขียน	4 พอร์ตสำหรับการอ่านและ 2 พอร์ตสำหรับเขียน
หน่วยทำงานจำนวนเต็ม	8 หน่วยทำงาน	2 หน่วยทำงาน
การทำนายทางแยก	โครงสร้างไม่สนับสนุนการทำนายทางแยก	โครงสร้างสนับสนุนการทำนายทางแยก
การแก้ปัญหาข้อมูล	หน่วงเวลา	การเปลี่ยนชื่อริจิสเตอร์
เสถียรภาพการประมวลผล	ไม่แน่นอน	ดีและค่อนข้างคงที่
รองรับการเกิดอินเตอร์รัพ	ไม่รองรับ	รองรับ

7.1.2 ประโยชน์ที่ได้รับ

ประโยชน์ที่ได้รับจากการทำงานวิจัยนี้ สรุปเป็นหัวข้อใหญ่ๆ ได้คือ

1. เข้าใจการทำงานของโปรเซสเซอร์ในระดับสูงดีขึ้น
2. สามารถประมวลความรู้หลายด้านมาใช้ในการทำงานวิจัยได้อย่างเหมาะสม เป็นขั้นตอน
3. เพิ่มพูนความรู้และทักษะการแก้ปัญหาต่างๆ ในงานวิจัยและเรื่องที่เกี่ยวข้อง

4. สามารถนำความรู้ที่ได้ไปประยุกต์แก้ปัญหากับซูเปอร์สเกลลาร์ไมโครโปรเซสเซอร์อื่นๆ ได้เป็นอย่างดี

7.1.3 อุปสรรคและปัญหา

จากการทำงานวิจัยตั้งแต่เริ่มต้นจนกระทั่งถึงขั้นตอนสุดท้ายพบว่าต้องเผชิญกับปัญหาและอุปสรรคมากมาย แบ่งปัญหาและอุปสรรคที่เกิดขึ้นออกเป็นแต่ละส่วน ดังนี้

1) อุปกรณ์และเครื่องมือที่ใช้ในงานวิจัย

- ไมโครโปรเซสเซอร์ดีแอลเอ็กซ์ ซึ่งเป็นไมโครเซสเซอร์ที่ยกมาเป็นต้นแบบในงานวิจัยนั้น ได้รับความนิยมมากเนื่องจากโครงสร้างไม่ซับซ้อน สามารถเข้าใจได้ง่าย จึงมีผู้นำไปเป็นต้นแบบและพัฒนาต่อมากมาย จึงไม่สามารถสรุปได้ว่าอันไหนคือปัจจุบันที่สุด และจากเอกสารหรือหนังสือที่เผยแพร่นั้นจะมีเฉพาะหลักการที่นำเสนอ วิธีการอิมพลีเมนท์จะไม่เปิดเผย ซึ่งเป็นอุปสรรคพอสมควรต่องานวิจัย เนื่องจากงานวิจัยนี้มีจุดมุ่งหมายในการนำดีแอลเอ็กซ์ซูเปอร์สเกลลาร์มาปรับปรุงประสิทธิภาพโดยการทำไดนามิกสเคจดูล ซึ่งหมายถึงว่าจะต้องมีดีแอลเอ็กซ์โปรเซสเซอร์ที่เป็นซูเปอร์สเกลลาร์อยู่ในห้องวิจัยแล้ว พร้อมใช้งาน แต่เมื่อไม่มีและเพื่อให้งานวิจัยดำเนินต่อไปได้ ผู้วิจัยจึงต้องสร้างดีแอลเอ็กซ์ซูเปอร์สเกลลาร์โปรเซสเซอร์ต้นแบบขึ้นมาก่อนโดยยึดหลักการของผู้ที่ให้กำเนิดไมโครโปรเซสเซอร์ดีแอลเอ็กซ์เดิมตาม [1] จากนั้นจึงนำดีแอลเอ็กซ์โปรเซสเซอร์นั้นไปทำงานตามจุดมุ่งหมายของงานวิจัยที่นำเสนอ

- โปรแกรมทดสอบ (Benchmark) การจะวัดประสิทธิภาพของไมโครโปรเซสเซอร์ที่ออกแบบให้ได้ผลเป็นที่ยอมรับและมีมาตรฐานนั้น จะต้องมีโปรแกรมทดสอบที่ใช้งานจริงตามท้องตลาดหรือโปรแกรมทดสอบที่ยอมรับได้สำหรับทดสอบการทำงานของซูเปอร์สเกลลาร์โปรเซสเซอร์ แต่ปัญหาคือผู้วิจัยไม่สามารถหาโปรแกรมทดสอบที่มีคุณสมบัติตามต้องการนั้นได้ไม่ว่าจากห้องวิจัยหรือจากสื่อใดๆ เนื่องจากปัญหาลิขสิทธิ์และอื่นๆ นอกจากนี้โปรแกรมทดสอบยังต้องอยู่ในรูปของแมชชีนโค้ดของโปรเซสเซอร์ดีแอลเอ็กซ์และข้อจำกัดของภาษาบรรยายฮาร์ดแวร์วีเอชดีแอลที่ใช้จำลองการทำงานไม่สนับสนุนงานบางอย่าง ดังนั้นโปรแกรมทดสอบที่ใช้ในงานวิจัย ผู้วิจัยจึงนำหลักการคำนวณทางคณิตศาสตร์แบบต่างๆซึ่งเป็นที่รู้จักและคุ้นเคยกันดีมาเขียนขึ้นเองในรูปแบบของภาษาระดับสูงแล้วแปลงเป็นแมชชีนโค้ดของโปรเซสเซอร์ดีแอลเอ็กซ์ โดยเนื้อหาของโปรแกรมทดสอบจะเกี่ยวข้องกับส่วนที่โปรเซสเซอร์ที่ออกแบบต้องการทดสอบ

- หนังสือสำหรับค้นคว้าเพิ่มเติม พบว่าหนังสือที่เกี่ยวข้องกับการออกแบบไมโครโปรเซสเซอร์ไปปีไลน์ขั้นสูงมีน้อยมาก ที่มีอยู่จะเป็นการแนะนำหลักการเบื้องต้น แต่เมื่อต้องนำไปอิมพลีเมนท์จริงจะต้องแก้ปัญหาที่เกิดขึ้นมาก มีหนังสือบางเล่มที่บอกถึงเทคนิควิธีการอิมพลีเมนท์จริงซึ่งจะมีขาย

ในต่างประเทศแต่มีราคาสูงมาก ฉนั้นในการทำวิจัยบางครั้งจึงต้องมีการลองผิดลองถูกจนกว่าจะได้วิธีการที่ถูกต้องและไม่เข้าใจผิด

- เครื่องคอมพิวเตอร์ส่วนบุคคล ที่ใช้ทำงานวิจัยมีสเป็กค่อนข้างไม่เหมาะสม ทำงานช้า และซอฟท์แวร์ที่ใช้ก็ไม่สนับสนุนการทำงานบางอย่าง

7.2 ข้อเสนอแนะ

จากหลักการไดนามิกสเคจดูลที่นำเสนอสามารถนำไปประยุกต์ใช้เพื่อเพิ่มประสิทธิภาพให้กับซูเปอร์สเกลลาร์โปรเซสเซอร์ แต่ต้องปรับปรุงให้เหมาะสมกับโปรเซสเซอร์นั้นๆ แนวทางในการปรับปรุงผู้วิจัยขอเสนอว่า

1. เพิ่มการเฟ็ทช์คำสั่งจากไซเคิลละ 2 คำสั่งเป็น 4 คำสั่ง
2. ปรับปรุงการทำนายทางแยกให้เป็นแบบไดนามิก แบบอื่นๆ หรือที่เสนอไว้ใน [13]
3. นำหลักการที่นำเสนอไปอิมพลีเมนท์บน FPGA
4. ถ้าเป็นไปได้ ควรมีโปรแกรมทดสอบที่ใช้งานจริงในท้องตลาด ผลการทดสอบที่ได้จะถูกยอมรับมากขึ้น

ประโยชน์ของงานวิจัยนี้นอกเหนือจากความรู้ที่ได้รับแล้ว ผู้วิจัยยังต้องการเสนอแนะหลักการและแนวทางใหม่ๆ เพื่อให้นักวิจัยรุ่นต่อไปได้เห็นความสำคัญของการออกแบบโปรเซสเซอร์ เนื่องจากปัจจุบันเป็นยุคของเทคโนโลยี การประมวลผลข้อมูล การรับรู้ข่าวสารได้รวดเร็วและมีประสิทธิภาพเป็นเรื่องได้เปรียบ

เอกสารอ้างอิง

[1] Patterson, D.A. and Hennessy, J.L. 1995. **Computer Architecture: A Quantitative Approach.** 2nd.ed. California : Morgan Kaufmann Publisher.

[2] Patterson, D.A. and Hennessy, J.L. 1990. **Computer Architecture: A Quantitative Approach.** California : Morgan Kaufmann Publisher.

[3] Johnson, M. 1991. **Superscalar Microprocessor design.** Prentice Hall.

[4] Patterson, D.A. and Hennessy, J.L. 1998. **Computer Organization & Design : The Hardware/Software Interface.** 2nd.ed. California : Morgan Kaufmann Publisher.

[5] Bolychevsky, A. et.al. 1996. "Dynamic Scheduling in RISC architecture." **IEEE Proc.-Comput.digit,Tech.** 143(5) : 309-317.

[6] Lo, J. 1995. "Comparing Static and Dynamic Scheduling on Superscalar Processors."

[7] Arons, T. et.al. 1999. "Verifying Tomasulo's Algorithm by Refinement." : 306-309.

[8] Lee, J.K. and Smith, A.J. 1984. "Branch Prediction Strategies and Branch Target Buffer Design." **IEEE Trans. On Computers.** 33(17) : 7-22.

[9] Mano, M.M. 1993. **Computer System Architecture.** Los Angeles : Prentice-Hall.

[10] Mano, M.M. and Kime, C.R. 2001. **Logic and Computer Design Fundamentals.** 2nd , rev.ed. Los Angeles : Prentice-hall.

[11] Perleberg, C.H. 1989. "Branch Target Buffer Design." Master Thesis of University of California, Berkley.

[12] Stallings, W. 1998. **Computer Organization and Architecture : Designing for Performance.** 5th.ed. New Jersey : Prentice-Hall.

[13] พัชรินทร์ กลิ่นซ้อน. "การปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งแบบปรับปรุงการทำนายทางแยก." วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต. 2544.

ผลงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์และได้รับการตีพิมพ์

[1] จักรพันธุ์ วชิรภานนท์ และคณะ. 2544. "การเปรียบเทียบประสิทธิภาพการทำนายผลลัพธ์ของคำสั่งทางแยกแบบไดนามิก." **วิศวสารลาดกระบัง.** 18(2) : 84-89.

[2] จักรพันธุ์ วชิรภานนท์ และคณะ. 2544. "การเพิ่มประสิทธิภาพของไปป์ไลน์คำสั่งด้วยการปรับปรุงประสิทธิภาพการทำนาย." **วิศวสารลาดกระบัง.** 18(3) : 49-54.

[3] จักรพันธุ์ วชิรภานนท์ และคณะ. 2544. "การเพิ่มประสิทธิภาพของไปป์ไลน์คำสั่งด้วยการปรับปรุงการทำนายทางแยก." หน้า 1374-1379. ใน **การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 24.** กรุงเทพฯ : คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.

ประวัติผู้เขียน

นายจักรพันธุ์ วชิรภานนท์ สำเร็จการศึกษาระดับอุดมศึกษา สาขาวิศวกรรมไฟฟ้า (สื่อสาร) คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์