การปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งแบบปรับปรุงการทำนายทางแยก

INSTRUCTION PIPELINE PERFORMANCE IMPROVEMENT USING ADAPTIVE BRANCH PREDICTION

พัชรินทร์ กลิ่นซ้อน

PATCHARIN KLINSORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2544

ISBN 974-648-399-4

การปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งแบบปรับปรุงการทำนายทางแยก

INSTRUCTION PIPELINE PERFORMANCE IMPROVEMENT USING ADAPTIVE BRANCH PREDICTION

พัชรินทร์ กลิ่นซ้อน

PATCHARIN KLINSORN

เลขหมู่......

เลขทะเบียน 40799

วัน, เดือน, ปี 26 พ.ย. 2544

.b......

.i......

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2544

ISBN 974-648-399-4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INSTRUCTION PIPELINE PERFORMANCE IMPROVEMENT USING ADAPTIVE BRANCH PREDICTION

PATCHARIN KLINSORN

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

2001

ISBN 974-648-399-4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

COPYRIGHT 2001

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งแบบปรับปรุงการทำนายทางแยก

INSTRUCTION PIPELINE PERFORMANCE IMPROVEMENT USING ADAPTIVE BRANCH PREDICTION

ชื่อนักศึกษา นางสาวพัชรินทร์ กลิ่นซ้อน

รหัสประจำตัว 39061051

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.บรรจง ปิยธำรง

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ผศ.ดร.เอื้อน	ปิ่นเงิน	
รศ.ดร.บุญธีร์	เครือตราชู	
ดร.ชุติเมษฎ์	ศรีนิลทา	
รศ.ประทีป	บัญญัติินพรัตน์	
รศ.บรรจง	ปิยธำรง	

วัน/เดือนปี ที่สอบ 24 กรกฎาคม 2544 เวลา 10.00-12.00 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-401)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่..........**เดือน**..........**พ.ศ.**..........

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อวิทยานิพนธ์	การปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งแบบปรับปรุงการทำนายทางแยก
นักศึกษา	นางสาว พัชรินทร์ กลิ่นซ้อน
รหัสประจำตัว	39061051
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2544
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ. บรรจง ปิยธำรง

บทคัดย่อ

ปัญหาการควบคุมทำให้ประสิทธิภาพของไปป์ไลน์คำสั่งลดลงเป็นอย่างมาก เมื่อคำสั่งทางแยกถูกนำเข้ามาประมวลผล ไมโครโปรเซสเซอร์จะต้องตรวจสอบเงื่อนไขของคำสั่งทางแยก เมื่อพบว่าคำสั่งทางแยกทำให้โปรแกรมเคาน์เตอร์เปลี่ยนแปลงไปยังตำแหน่งเป้าหมาย โปรเซสเซอร์จะต้องคำนวณตำแหน่งเป้าหมายนั้นและควบคุมให้การประมวลผลเกิดขึ้น ณ ตำแหน่งเป้าหมาย หลังจากนั้นโปรเซสเซอร์อาจจะมีการหน่วงเวลาหรือยกเลิกการทำงานของบางคำสั่งจนกว่าการประมวลผลของคำสั่งทางแยกจะเสร็จสิ้น ทำให้สูญเสียไซเคิลการทำงานมากขึ้น การทำนายผลลัพธ์ของคำสั่งทางแยกแบบไดนามิกซึ่งผลการทำนายเปลี่ยนแปลงได้ตลอดขณะโปรแกรมทำงานจึงช่วยให้การทำงานของโปรเซสเซอร์ประมวลผลคำสั่งทางแยกได้ดี วิทยานิพนธ์ฉบับนี้นำเสนอการปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งแบบการปรับปรุงการทำนายทางแยกด้วยวิธีการทำนายแบบ 2 บิต มีหลักการคือเก็บประวัติพฤติกรรมในอดีตของคำสั่งทางแยกไว้ในหน่วยความจำบีทีบีและบีเอชทีเพื่อใช้สำหรับคาดหมายผลลัพธ์ของคำสั่งทางแยกปัจจุบัน เมื่อได้ทดสอบการทำงานโปรเซสเซอร์แบบไปป์ไลน์คำสั่งภายใต้หลักการที่นำเสนอนี้กับโปรแกรมทดสอบที่มีลักษณะของลูปคำสั่งทางแยกซับซ้อน ผลการทดสอบสรุปได้ว่า การปรับปรุงการทำนายทางแยกที่นำเสนอดังกล่าว ทำให้ประสิทธิภาพของไปป์ไลน์คำสั่งเพิ่มขึ้น จำนวนไซเคิลการทำงานต่อคำสั่งมีค่าเข้าใกล้ 1 ความเร็วเพิ่มขึ้น การทำนายผลลัพธ์ของคำสั่งทางแยกผิดพลาดน้อยลงอย่างชัดเจน และจำนวนไซเคิลที่สูญเสียเนื่องจากปัญหาควบคุมลดลง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Thesis Title	Instruction Pipeline Performance Improvement using Adaptive Branch Prediction
Student	Miss Patcharin Klinsorn
Student ID.	39061051
Degree	Master of Engineering in Electrical Engineering
Programmme	Electrical Engineering
Year	2001
Thesis Advisor	Assoc.Prof.Bunjong Piyatamrong

ABSTRACT

Control hazards can cause a lot of performance of instruction pipeline microprocessor loss. When a branch is executed, the processor will test its condition. If a branch changes the program counter to its target address, the processor must calculate branch target address and execute that target address. To handle branches is to freeze or flush the pipeline, holding or deleting any instruction after the branch until the branch destination is known, which is waste time. With dynamic branch prediction scheme that branch outcome change all time as program running, reducing control hazards. This research presents the method to improve performance of instruction pipeline processor by using 2-bit scheme prediction. The algorithm is to keep the branch history in BTB (Branch Target Buffer) and BHT (Branch History Table) and use them to predict the outcome of the current branch instruction. From the simulation, with 2-bit prediction scheme the processor execution faster. The number of clock cycle per instruction is approaching to one, and improving the pipeline performance.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลงได้ด้วยคำแนะนำและคำปรึกษาจากผู้รู้หลายท่าน ขอขอบพระคุณ รศ. บรรจง ปิยธำรง อาจารย์ที่ปรึกษาเป็นอย่างสูง คุณจักรพันธุ์ วชิรภานนท์ ผู้ช่วยเหลือให้คำปรึกษาเรื่องงานวิจัย และครอบครัวที่ให้กำลังใจตลอดมา อุปสรรคหรือปัญหาใดๆ ที่เกิดขึ้น ผู้เขียนถือว่าเป็นแบบทดสอบที่มีคุณค่ายิ่ง

สุดท้ายนี้ขอขอบพระคุณ "**ครู**" ทุกท่านอีกครั้ง ความรู้และประสบการณ์ที่ได้รับจากการลงมือทำวิทยานิพนธ์นี้เป็นสิ่งที่มีความสำคัญมาก หากมีโอกาสอันสมควรผู้เขียนจะเผยแพร่ต่อไป

พัชรินทร์ กลิ่นซ้อน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

กิตติกรรมประกาศ III

สารบัญ IV

สารบัญตาราง VI

สารบัญรูป VII

บทที่ 1 บทนำ 1

1.1 ความเป็นมาและความสำคัญของปัญหา 1

1.2 วัตถุประสงค์ของงานวิจัย 2

1.3 ขอบเขตของงานวิจัย 2

1.4 รายละเอียดของวิทยานิพนธ์ 2

บทที่ 2 สถาปัตยกรรมของไปป์ไลน์คำสั่ง 4

2.1 สถาปัตยกรรมของไมโครโปรเซสเซอร์เดอลุกซ์ 4

2.2 ไมโครโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ 9

2.3 การวัดประสิทธิภาพ 15

บทที่ 3 ปัญหาของไปป์ไลน์คำสั่ง 16

3.1 ปัญหาโครงสร้างฮาร์ดแวร์ 16

3.2 ปัญหาข้อมูล 18

3.3 ปัญหาการควบคุม 23

3.3.1 การแก้ปัญหาการควบคุมโดยการปรับปรุงโครงสร้างดาต้าพาธของโปรเซสเซอร์เดอลุกซ์ 23

3.3.2 การแก้ปัญหาการควบคุมด้วยการทำนายผลลัพธ์ของคำสั่งทางแยกล่วงหน้า....................26

3.3.3 การแก้ปัญหาควบคุมด้วยหลักการดีเลย์บรานช์ 28

3.3.4 การแก้ปัญหาควบคุมด้วยการทำไปป์ไลน์สเคจดูลลิงและลูปอันโรลลิง....................29

3.3.5 การแก้ปัญหาการประมวลผลคำสั่งทางแยกในไมโครโปรเซสเซอร์อื่นๆ 33

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

หน้า

3.4 การวัดประสิทธิภาพ ...36

บทที่ 4 การแก้ปัญหาควบคุมในไปป์ไลน์คำสั่ง ...39

4.1 การทำนายผลลัพธ์ของคำสั่งทางแยกแบบเดิม ...40

4.2 การปรับปรุงการทำนายทางแยก ...47

บทที่ 5 การทดสอบการปรับปรุงการทำนายทางแยก ...57

5.1 ตัวแปรบอกประสิทธิภาพ ...57

5.2 โปรแกรมทดสอบ ...58

5.2.1 โปรแกรมทดสอบที่ 1 ...58

5.2.2 โปรแกรมทดสอบที่ 2 ...60

5.2.3 โปรแกรมทดสอบที่ 3 ...62

5.2.4 โปรแกรมทดสอบที่ 4 ...66

5.2.5 โปรแกรมทดสอบที่ 5 ...68

5.2.6 โปรแกรมทดสอบที่ 5 ที่มีพฤติกรรม Not taken สูง ...74

5.3 ผลการทดสอบ ...78

5.3.1 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 1 ...78

5.3.2 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 2 ...79

5.3.3 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 3 ...80

5.3.4 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 4 ...81

5.3.5 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 5 ...83

5.4.6 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 5 ที่มีพฤติกรรม Not taken สูง ...84

5.4 การเปรียบเทียบประสิทธิภาพ ...85

5.5 สรุปประสิทธิภาพของไมโครโปรเซสเซอร์ ...87

บทที่ 6 สรุปผลการวิจัยและบทวิจารณ์ ...90

6.1 สรุปผลการวิจัย ...90

6.2 บทวิจารณ์ ...92

เอกสารอ้างอิง ...95

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

หน้า

ผลงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์และได้รับการตีพิมพ์ 96
ประวัติผู้เขียน 97

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

ตารางที่ หน้า

4.1 จำนวนไซเคิลที่สูญเสียสำหรับการแก้ปัญหาควบคุมด้วยหลักการเดิม 40

4.2 จำนวนไซเคิลที่สูญเสียสำหรับคำสั่ง J และคำสั่ง JAL 55

4.3 จำนวนไซเคิลที่สูญเสียสำหรับคำสั่ง JR และคำสั่ง JALR 56

4.4 จำนวนไซเคิลที่สูญเสียสำหรับคำสั่ง BEQZ และคำสั่ง BNEZ 56

5.1 โค้ดแอดสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 1 60

5.2 โค้ดแอดสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 2 61

5.3 โค้ดแอดสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 3 64

5.4 โค้ดแอดสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 4 67

5.5 โค้ดแอดสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 5 70

5.6 โค้ดแอดสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 5 ที่มีพฤติกรรมของคำสั่งทางแยก Not taken สูง 74

5.7 สรุปประสิทธิภาพที่เพิ่มขึ้นของไมโครโปรเซสเซอร์เมื่อมีการปรับปรุงการทำนายทางแยก ...89

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

รูปที่		หน้า
2.1	ฟอร์แมทคำสั่งของไมโครโปรเซสเซอร์เดอลุกซ์	5
2.2	ตัวอย่างของคำสั่ง Load และ Store	6
2.3	ตัวอย่างคำสั่งการคำนวณในเอแอลยู	7
2.4	ตัวอย่างของคำสั่ง Jump และ Branch	7
2.5	คำสั่งทั้งหมดของไมโครโปรเซสเซอร์เดอลุกซ์	8
2.6	ดาต้าพาธไมโครโปรเซสเซอร์เดอลุกซ์ที่ไม่ได้ทำไปป์ไลน์คำสั่ง	13
2.7	การทำงานในแต่ละไปป์สเตจของไมโครโปรเซสเซอร์เดอลุกซ์	14
2.8	โครงสร้างดาต้าพาธของไมโครโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์	15
3.1	ปัญหาการขัดแย้งของทรัพยากรเมื่อ 2 คำสั่งต้องอ้างอิงข้อมูลจากหน่วยความจำเดียว	16
3.2	การหน่วงเวลาในไปป์ไลน์เมื่อมีปัญหาโครงสร้างฮาร์ดแวร์	17
3.3	พฤติกรรมของแต่ละไปป์สเตจเมื่อมีการหน่วงเวลาในไปป์ไลน์สำหรับปัญหาโครงสร้างฮาร์ดแวร์	18
3.4	ไซเคิลการทำงานของชุดคำสั่ง	19
3.5	ลำดับการทำงานของไปป์ไลน์เมื่อเกิดปัญหาการใช้ข้อมูลร่วมกัน	20
3.6	การแก้ปัญหาข้อมูลด้วยวิธีฟอร์เวิร์ดดิ้ง	22
3.7	การแก้ปัญหาข้อมูลด้วยการหน่วงเวลา	23
3.8	ดาต้าพาธของโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ที่มีการตรวจสอบคำสั่งทางแยก	24
3.9	พฤติกรรมในไปป์ไลน์เมื่อพบคำสั่งทางแยก	24
3.10	ดาต้าพาธของโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ที่ปรับปรุง	25
3.11	การหน่วงเวลา 1 ไซเคิลเมื่อต้องยกเลิกคำสั่งที่ตามหลังคำสั่งทางแยก	26
3.12	การทำงานของไปป์ไลน์เมื่อคำสั่งทางแยกมีพฤติกรรม taken	27
3.13	การทำงานของไปป์ไลน์เมื่อคำสั่งทางแยกมีพฤติกรรม not taken	27
3.14	การนำคำสั่งอิสระแทรกเข้าไปทำงานใน ดีเลย์สล็อท	28
3.15	ประวัติการพัฒนากลไกการทำนายผลลัพธ์คำสั่งทางแยกของโปรเซสเซอร์	33
3.16	การทำนายแบบปรับปรุง 2 ระดับ	33
3.17	หน่วยความจำบีทีบีของโปรเซสเซอร์ Intel P6	34
3.18	การทำนายผลลัพธ์ของคำสั่งทางแยกในโปรเซสเซอร์ DEC Alpha	35
3.19	โครงสร้างหน่วยความจำคำสั่งของโปรเซสเซอร์ DEC Alpha	35
3.20	การทำนายผลลัพธ์ของคำสั่งทางแยกในโปรเซสเซอร์ที่มีอยู่ในปัจจุบัน	36

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

รูปที่ หน้า

3.21 ประสิทธิภาพของการแก้ปัญหาของคำสั่งทางแยก 4 วิธี 38

4.1 ดาต้าพาธของการทำนายผลลัพธ์ทางแยกเดิม 41

4.2 โครงสร้างภายนอกของหน่วยการตรวจสอบ 42

4.3 ลำดับการทำงานของโค้ดโปรแกรมเมื่อทำงานกับคำสั่ง J 43

4.4 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BEQZ มีสถานะเป็น taken 45

4.5 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BEQZ มีสถานะเป็น not taken 45

4.6 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BNEZ มีสถานะเป็น taken 46

4.7 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BNEZ มีสถานะเป็น not taken 47

4.8 โครงสร้างของบีทีบี 48

4.9 โครงสร้างของบีทีบีในงานวิจัย 48

4.10 โครงสร้างของบีเอชที 49

4.11 สเตทแมชชีน 50

4.12 โครงสร้างของดาต้าพาธแบบปรับปรุงการทำนายทางแยกในงานวิจัย 52

4.13 หลักการทำงานของการปรับปรุงการทำนายทางแยกในงานวิจัย 53

5.1 เปอร์เซนต์ของคำสั่งทางแยกแบบมีเงื่อนไขและไม่มีเงื่อนไขของโปรแกรมทดสอบ 78

5.2 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 1 79

5.3 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 2 80

5.4 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 3 81

5.5 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 4 82

5.6 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 5 83

5.7 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 5 ที่มีพฤติกรรม Not taken สูง 84

5.8 การเปรียบเทียบ CPI ของโปรแกรมทดสอบทั้ง 5 โปรแกรมระหว่างหลักการเดิมและหลักการใหม่ 85

5.9 การเปรียบเทียบ Speedup ของโปรแกรมทดสอบทั้ง 5 โปรแกรมระหว่างหลักการเดิมและหลักการใหม่ 86

5.10 การเปรียบเทียบ %Mispredict ของโปรแกรมทดสอบทั้ง 5 โปรแกรมระหว่างหลักการเดิมและหลักการใหม่ 86

5.13 จำนวนไซเคิลที่สูญเสียเมื่อทดสอบกับโปรแกรมทดสอบที่แตกต่างกัน 88

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

รูปที่ หน้า

5.11 การเปรียบเทียบ Branch penalty ของโปรแกรมทดสอบทั้ง 5 โปรแกรมระหว่างหลักการเดิมและหลักการใหม่ 87
5.12 ประสิทธิภาพที่เพิ่มขึ้นของโปรเซสเซอร์เมื่อทดสอบกับโปรแกรมทดสอบที่แตกต่างกัน 87

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันความต้องการใช้ไมโครคอมพิวเตอร์ที่มีความสามารถในการประมวลผลมีมากขึ้น การพัฒนาไมโครโปรเซสเซอร์ให้ทำงานรองรับความต้องการดังกล่าวทำได้หลายวิธี แนวความคิดดั้งเดิมในการทำให้คอมพิวเตอร์ทำงานได้เร็วขึ้นนั้น จะใช้วิธีการเพิ่มขีดความสามารถของคำสั่ง โดยที่คำสั่งหนึ่งจะทำงานเพิ่มขึ้นและซับซ้อนขึ้น ทำให้สถาปัตยกรรมเฉพาะของซีพียูต้องรองรับหรือสนับสนุนคำสั่งใหม่ๆ เพิ่มขึ้น การทำงานของแต่ละคำสั่งใช้จำนวนไซเคิล (Clock cycle) ไม่เท่ากัน อีกแนวความคิดหนึ่งที่ต้องการให้คอมพิวเตอร์ทำงานได้เร็วขึ้นคือ ให้ซีพียูทำงานที่จำนวนไซเคิลแน่นอน โดยลดความซับซ้อนของคำสั่งลงให้เป็นคำสั่งพื้นฐานมากที่สุด แล้วใช้หลักการไปป์ไลน์คำสั่ง (Instruction pipeline) ได้แก่การนำคำสั่งพื้นฐานนั้นมาเรียงการทำงานให้มีลักษณะขนานเหลื่อมกัน หรือแบ่งการทำงานออกเป็นไปป์สเตจหรือสเตจ (Pipe stage) และคำสั่งเข้าทำงานแต่ละไปป์สเตจตามลำดับเรียงกันไป ทุกไปป์สเตจจะมีการทำงานตลอดเวลา ซึ่งเป็นหลักการของ RISC (Reduce Instruction Set Computers) โปรเซสเซอร์ที่มีการทำไปป์ไลน์คำสั่งอาจเรียกโปรเซสเซอร์นั้นว่าโปรเซสเซอร์ไปป์ไลน์ก็ได้

จากหลักการของไปป์ไลน์คำสั่ง ทำให้ซีพียูสามารถประมวลผลได้เร็วกว่าซีพียูที่ไม่ได้ทำไปป์ไลน์คำสั่งเป็นจำนวนเท่ากับจำนวนไปป์สเตจ แต่ในทางปฏิบัติมีปัญหาเกิดขึ้นหลายประการที่ทำให้ซีพียูไปป์ไลน์ไม่สามารถมีความเร็วได้เท่ากับในอุดมคติ ปัญหาของไปป์ไลน์คำสั่งได้แก่ ปัญหาโครงสร้างทางฮาร์ดแวร์ (Structure hazards) ปัญหาข้อมูล (Data hazards) และปัญหาการควบคุม (Control Hazards) โดยที่ปัญหาการควบคุมซึ่งเกี่ยวข้องกับคำสั่งทางแยก (Branch instruction) ถือว่าเป็นปัญหาสำคัญที่ทำให้ประสิทธิภาพของไปป์ไลน์คำสั่งลดลงมากที่สุด เนื่องจากในโปรแกรมส่วนใหญ่จะมีการประมวลผลคำสั่งทางแยกอยู่เสมอ และการประมวลผลคำสั่งทางแยกซีพียูจะต้องมีการตรวจสอบริจิสเตอร์ ด้วยความสำคัญของปัญหาของคำสั่งทางแยกดังกล่าว ถ้ามีกระบวนการใดๆที่รองรับการทำงานของคำสั่งทางแยก อาทิ กระบวนการที่สามารถตรวจสอบได้ล่วงหน้าว่าคำสั่งที่ถูกนำเข้ามาประมวลผลนั้นเป็นคำสั่งทางแยก หรือสามารถตรวจสอบได้ว่าคำสั่งทางแยกที่เข้ามาประมวลผลนั้นทำให้โปรแกรมเคาน์เตอร์ (Program Counter) เปลี่ยนแปลงไปอย่างไม่เป็นลำดับหรือชี้ไปยังแอดเดรสเป้าหมาย (Target address) แล้ว ปัญหาของคำสั่งทางแยกที่ทำให้ความเร็วของไปป์ไลน์คำสั่งลดลงก็จะน้อยลงไป จุดมุ่งหมายในการเพิ่มขีดความสามารถในการประมวลผลของคอมพิวเตอร์ก็จะสำเร็จลุล่วงมากขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1.2 วัตถุประสงค์ของงานวิจัย

- ศึกษาและเข้าใจลักษณะสถาปัตยกรรมของไปป์ไลน์คำสั่ง
- เข้าใจปัญหาที่เกิดขึ้นในไปป์ไลน์คำสั่งและวิธีแก้ไข
- เข้าใจปัญหาที่เกิดขึ้นเนื่องจากการประมวลผลคำสั่งทางแยกและวิธีแก้ปัญหาเบื้องต้น
- สามารถปรับปรุงวิธีการทำนายผลลัพธ์ของคำสั่งทางแยกในปัญหาการควบคุมได้
- มีความรู้เกี่ยวกับการออกแบบวงจรด้วยภาษาวีเอชดีแอล

1.3 ขอบเขตของงานวิจัย

ภายใต้งานวิจัยเกี่ยวกับการปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งแบบปรับปรุงการทำนายทางแยก ก่อนที่จะปรับปรุงประสิทธิภาพของไปป์ไลน์คำสั่งได้นั้นจะต้องมีโปรเซสเซอร์แบบไปป์ไลน์ที่มีการแก้ไขปัญหาคำสั่งทางแยกด้วยหลักการเดิม แล้วจึงทำการปรับปรุงการแก้ปัญหาจากหลักการเดิมนั้นด้วยหลักการใหม่ที่นำเสนอ หลังจากนั้นจึงทำการเปรียบเทียบประสิทธิภาพการทำงานของซีพียูระหว่างการแก้ปัญหาทั้งสองแบบ ซึ่งสามารถสรุปขั้นตอนการทำงานเป็นขอบข่ายของงานวิจัยได้ดังนี้

1. สร้างโปรเซสเซอร์ที่มีการประมวลผลคำสั่งแบบไปป์ไลน์ เพื่อใช้เป็นโปรเซสเซอร์ต้นแบบสำหรับการทดสอบในงานวิจัย
2. สร้างข้อมูลทดสอบ โดยเน้นให้ข้อมูลทดสอบนั้นมีผลทำให้เกิดปัญหาการควบคุม หรือปัญ-หาเกี่ยวกับคำสั่งทางแยก
3. ทดสอบการแก้ปัญหาของไปป์ไลน์คำสั่งเมื่อเกิดปัญหาเกี่ยวกับคำสั่งทางแยก ด้วยหลักการเดิมที่มีอยู่
4. ทดสอบการแก้ปัญหาของไปป์ไลน์คำสั่งเมื่อเกิดปัญหาเกี่ยวกับคำสั่งทางแยก ด้วยหลักการใหม่ที่นำเสนอ
5. เปรียบเทียบประสิทธิภาพระหว่างการแก้ปัญหาคำสั่งทางแยกทั้งสองแบบ

1.4 รายละเอียดของวิทยานิพนธ์

ภายใต้ขอบข่ายของงานวิจัย วิทยานิพนธ์ฉบับนี้ต้องการเรียงลำดับเนื้อหาอย่างต่อเนื่องเพื่อให้ครอบคลุมเนื้อหาทั้งหมดที่เกี่ยวข้องอย่างเหมาะสม อย่างไรก็ตามในบางเรื่องอาจจะไม่กล่าวถึงหรืออาจจะรวบรัดเนื่องจากมิใช่สาระสำคัญของงานวิจัยและเป็นส่วนความรู้พื้นฐานของผู้ที่อยู่ในวงการคอมพิวเตอร์มักคุ้นเคยเป็นอย่างดี อาทิ ความรู้เบื้องต้นเกี่ยวกับการเขียนโปรแกรมด้วยภาษาวีเอชดีแอล สำหรับรายละเอียดของวิทยานิพนธ์ประกอบด้วยเนื้อหาดังต่อไปนี้

- บทที่ 2 กล่าวถึงสถาปัตยกรรมของไปป์ไลน์คำสั่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- บทที่ 3 กล่าวถึงปัญหาของไปป์ไลน์คำสั่งและการแก้ปัญหาในโปรเซสเซอร์ทั่วไป
- บทที่ 4 กล่าวถึงการแก้ปัญหาการควบคุมในไปป์ไลน์คำสั่งด้วยหลักการที่นำเสนอ
- บทที่ 5 กล่าวถึงการทดสอบการปรับปรุงการทำนายทางแยกตามหลักการที่นำเสนอ
- บทที่ 6 สรุปผลการวิจัยและบทวิจารณ์

อนึ่ง รายละเอียดบางส่วนสามารถค้นคว้าเพิ่มเติมได้จากเอกสารอ้างอิง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
สถาปัตยกรรมไปป์ไลน์คำสั่ง

โปรเซสเซอร์ที่มีสถาปัตยกรรมแบบไปป์ไลน์คำสั่งหรือโปรเซสเซอร์ไปป์ไลน์ทั้งแบบเลขจำนวนเต็ม (Pipeline integer) และโฟลตติ้งพอยนต์ (Pipeline floating point) ที่ถือว่าเป็นต้นแบบของการศึกษาไมโครโปรเซสเซอร์ที่มีสถาปัตยกรรมแบบไปป์ไลน์คำสั่ง เนื่องจากมีโครงสร้างของสถาปัตยกรรมไม่ซับซ้อน ง่ายต่อการทำความเข้าใจ และสามารถสร้างขึ้นเองได้คือไมโครโปรเซสเซอร์เดอลุกซ์ (DLX) [1] งานวิจัยนี้จึงได้เลือกศึกษาสถาปัตยกรรมของไมโครโปรเซสเซอร์เดอลุกซ์และนำมาสร้างเป็นไมโครโปรเซสเซอร์ชนิดเลขจำนวนเต็ม เพื่อใช้เป็นต้นแบบทดสอบการแก้ปัญหาการควบคุม (Control hazards) การสร้างโปรเซสเซอร์ขึ้นดังกล่าวเป็นการจำลองการทำงานของโปรเซสเซอร์โดยใช้ภาษาบรรยายฮาร์ดแวร์วีเอชดีแอล

2.1 สถาปัตยกรรมของไมโครโปรเซสเซอร์เดอลุกซ์

สถาปัตยกรรมของไมโครโปรเซสเซอร์เดอลุกซ์ประกอบด้วยริจิสเตอร์ ชนิดข้อมูล การอ้างอิงแอดเดรส รูปแบบของชุดคำสั่ง โอเปอเรชัน และการวัดประสิทธิภาพของโปรเซสเซอร์ดังนี้

2.1.1 ริจิสเตอร์

ไมโครโปรเซสเซอร์เดอลุกซ์มีริจิสเตอร์ทั่วไป (General Purpose registers: GPRs) ขนาด 32 บิตจำนวน 32 ตัวแบบจำนวนเต็มคือ R0 ถึง R31 นอกเหนือจากริจิสเตอร์แบบโฟลตติ้งพอยนต์ (Floating-Point Regisrers: FPRs) โดยค่าในริจิสเตอร์ R0 เป็น 0 เสมอ นอกจากนี้ยังมีริจิสเตอร์พิเศษอีก 2-3 ตัวสำหรับการถ่ายโอนข้อมูลระหว่างริจิสเตอร์ด้วยกันเอง

2.1.2 ชนิดข้อมูล

ชนิดข้อมูลของไมโครโปรเซสเซอร์เดอลุกซ์มีขนาด 8 บิตไบต์ 16 บิตฮาร์ฟเวิร์ด (Half word) และ 32 บิตเวิร์ด (word) เมื่อข้อมูลแบบไบต์หรือฮาร์ฟเวิร์ดถูกนำเข้ามาไว้ในริจิสเตอร์จะมีการเพิ่มบิตเครื่องหมาย (sign-bit) หรือใส่เลขศูนย์ซ้ำจนกระทั่งริจิสเตอร์นั้นมีขนาดครบ 32 บิต

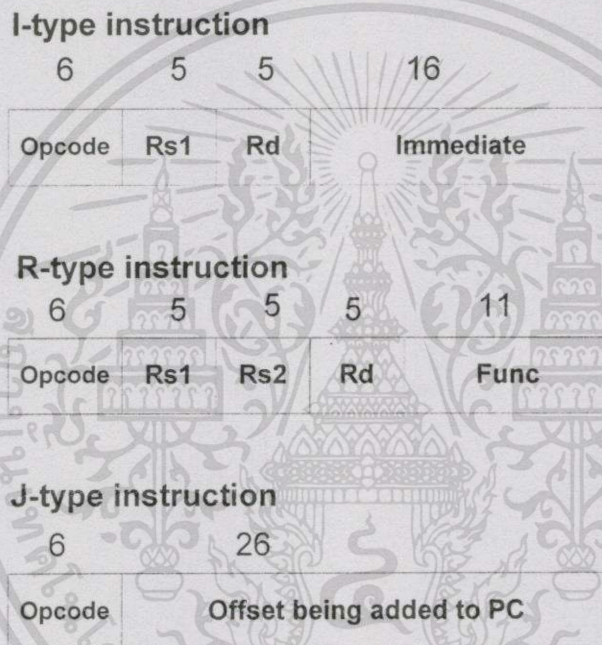
2.1.3 โหมดการอ้างอิงแอดเดรส

โหมดการอ้างอิงแอดเดรสของข้อมูลมี 2 โหมดคือ โหมดอิมมีเดียท (Immediate) และโหมดดิสเพลสเมนท์ (Displacement) ซึ่งมีขนาด 16 บิต การอ้างถึงหน่วยความจำแบบไบต์ในโหมดบิ๊กเอน-เดียน (Big Endian) ที่มีแอดเดรสขนาด 32 บิต ขณะที่สถาปัตยกรรมของโปรเซสเซอร์เกี่ยวข้องกับการนำข้อมูลเข้า (คำสั่ง Load) และการบันทึกข้อมูลกลับ (คำสั่ง Store) นั้นทุกการอ้างถึงหน่วยความจำจะเป็นการทำงานระหว่างหน่วยความจำกับริจิสเตอร์ทั่วไป

2.1.4 ฟอร์แมทของชุดคำสั่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เนื่องจากไมโครโปรเซสเซอร์เดอลุกซ์มีโหมดการอ้างอิงแอดเดรสอยู่เพียง 2 โหมด การเข้ารหัส (Encode) จึงกระทำกับออปโค้ด (Opcode) และเพื่อให้การสร้างโปรเซสเซอร์ไปป์ไลน์ง่ายขึ้น จึงกำหนดให้ 6 บิตแรกของคำสั่งขนาด 32 บิตนั้นเป็นออปโค้ด รูปที่ 2.1 แสดงฟอร์แมทคำสั่ง ลักษณะของฟอร์แมทคำสั่งนี้จะง่ายสำหรับการจัดเตรียมฟิลด์ขนาด 16 บิตเพื่อการอ้างอิงแอดเดรสแบบอิมมีเดียทและดิสเพลสเมนท์ รวมทั้งโปรแกรมเคาน์เตอร์สำหรับชี้แอดเดรสของทางแยก (branch) ด้วย



รูปที่ 2.1 ฟอร์แมทคำสั่งของไมโครโปรเซสเซอร์เดอลุกซ์ [2]

2.1.5 โอเปอเรชัน

โอเปอเรชันของไมโครโปรเซสเซอร์เดอลุกซ์มี 4 แบบตามลักษณะการทำงานของคำสั่งคือ โอเปอเรชันเกี่ยวกับการนำเข้า-การบันทึกกลับ (Load-Store) เอแอลยูโอเปอเรชัน (ALU operation) โอเปอเรชันเกี่ยวกับทางแยก (Branch-Jump operation) และโอเปอเรชันของโฟลตติ้งพอยนต์ ริจิสเตอร์ทั่วไปใดๆจะถูกนำข้อมูลเข้ามา-บันทึกกลับก็ได้ ยกเว้นริจิสเตอร์ R0 ซึ่งการนำเข้ามาไม่มีผลกระทบใดๆ ต่อริจิสเตอร์เนื่องจากเก็บค่าศูนย์เสมอ รูปที่ 2.2 แสดงตัวอย่างของคำสั่งการนำเข้ามา-บันทึกกลับ คำสั่งทั้งหมดของไมโครโปรเซสเซอร์เดอลุกซ์ แสดงดังรูปที่ 2.5 และเพื่อง่ายต่อการสื่อความหมาย สำหรับการเขียนคำสั่งแสดงการทำงานต่างๆจึงเขียนในรูปแบบของภาษาระดับสูงและกำหนดการใช้สัญลักษณ์ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ตัวห้อย (Subscript) ที่อยู่ข้างหลังเครื่องหมาย $\leftarrow$ ใช้แสดงการถ่ายโอนข้อมูลเมื่อความยาวของข้อมูลที่กำลังถ่ายโอนไม่ระบุชัดเจน และ $\leftarrow_n$ หมายถึงถ่ายโอนข้อมูลขนาด n บิต นอกจากนี้สัญลักษณ์ $x,y \leftarrow z$ หมายถึง z ถูกถ่ายโอนไปยัง x และ y
- ตัวห้อยใช้เพื่อให้ทราบถึงการเลือกบิตจากฟิลด์ ชื่อของบิตมาจากลำดับของบิตเริ่มจากบิต 0 ตัวห้อยอาจจะเป็นเลขตัวเดียวหรือเป็นช่วงก็ได้ เช่น $Regs[R4]_0$ หรือ $Regs[R3]_{24..31}$
- ตัวแปร Mem เป็นตัวแปรแบบอะเรย์ (Array) ใช้สำหรับหน่วยความจำหลักที่ถูกชี้โดยไบต์แอดเดรส
- ตัวยก (Superscript) ใช้เพื่อบอกว่าเป็นจำนวนซ้ำ เช่น 0^{24} หมายถึงเลขศูนย์จำนวน 24 บิต
- สัญลักษณ์ ## ใช้ในการเชื่อมต่อระหว่าง 2 ฟิลด์ และจะปรากฏอยู่ที่ใดก็ได้ ในการถ่ายโอนข้อมูล

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 2.2 ตัวอย่างของคำสั่ง Load และ Store [2]

ทุกคำสั่งในเอแอลยูเป็นคำสั่งที่ทำงานระหว่างริจิสเตอร์กับริจิสเตอร์ โอเปอแรนด์ของเอแอลยูมีทั้งการคำนวณอย่างง่าย คือ การบวก ลบ ตรรกะ และการเลื่อน นอกจากนี้ยังมีคำสั่งเปรียบเทียบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ซึ่งใช้เปรียบเทียบกันระหว่างริจิสเตอร์ 2 ตัว คือ เท่ากับ ไม่เท่ากับ น้อยกว่า มากกว่า น้อยกว่าหรือเท่ากับ และมากกว่าหรือเท่ากับ (=, ≠, <, >, ≤, ≥) ถ้าเงื่อนไขการเปรียบเทียบเป็นจริง คำสั่งจะเก็บค่า 1 ที่ริจิสเตอร์ปลายทาง สำหรับเงื่อนไขอื่นๆ จะเก็บค่า 0 ตัวอย่างของคำสั่งการคำนวณแสดงในรูปที่ 2.3

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 2.3 ตัวอย่างคำสั่งการคำนวณในเอแอลยู [3]

การควบคุมถูกจัดการผ่านชุดของคำสั่งทางแยก โดยที่คำสั่งทางแยกที่มีเงื่อนไขจะหมายถึงกลุ่มคำสั่ง Branch ส่วนกลุ่มคำสั่ง Jump จะเป็นคำสั่งทางแยกแบบไม่มีเงื่อนไข รูปที่ 2.4 แสดงตัวอย่างของชุดคำสั่ง jump และ branch

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

รูปที่ 2.4 ตัวอย่างของคำสั่ง Jump และ Branch [2]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Instruction type/ Opcode	Instruction meaning
Data transfer	Move data between registers and memory, or between the integer and FP or 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 (SP - single precision, DP - double precision)
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 integer or 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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). Both operands are in the FP registers.
__D, __F	DP and SP compares: "__" may be LT, GT, LE, GE, EQ, NE; set comparison bit in FP status register.

รูปที่ 2.5 คำสั่งทั้งหมดของไมโครโปรเซสเซอร์เดอลุกซ์ [2]

2.1.6 ประสิทธิภาพของไมโครโปรเซสเซอร์เดอลุกซ์ [1]

สมการที่ (2.1) เป็นสมการที่ใช้คำนวณหาเวลาที่โปรเซสเซอร์ใช้ในการประมวลผลคำสั่งทั้งหมดในโปรแกรมหนึ่งๆ โดยค่าที่คำนวณได้จะบอกถึงประสิทธิภาพการทำงานของไมโครโปรเซสเซอร์

$$\text{CPU time} = \text{Instructions count} \times \text{CPI} \times \text{Clock cycle time} \tag{2.1}$$

โดยที่ CPU time คือ เวลาทั้งหมดที่ซีพียูใช้ในการประมวลผลโปรแกรมหนึ่งๆ

Instructions count คือ จำนวนคำสั่งทั้งหมดในโปรแกรมที่ถูกประมวลผล

CPI คือ จำนวนไซเคิลที่ใช้ในการประมวลผล 1 คำสั่ง

Clock cycle time คือ ช่วงเวลาใน 1 ไซเคิล

2.2 ไมโครโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์

การทำไปป์ไลน์คำสั่งเป็นวิธีการจัดการคำสั่งหลายๆคำสั่งให้ทำงานพร้อมกันแบบเหลื่อมขนานกัน การทำไปป์ไลน์คำสั่งทำให้การประมวลผลคำสั่งเร็วขึ้นทำให้ประสิทธิภาพของซีพียูดีขึ้น หลักการของไปปไลน์คำสั่งเป็นเช่นเดียวกับโรงงานประกอบรถยนต์ที่ประกอบรถยนต์ได้หลายๆคันในเวลาไม่กี่นาที โดยโรงงานได้แบ่งขั้นตอนการประกอบชิ้นส่วนต่างๆของรถยนต์แยกกันเป็นสถานีงาน แต่ละสถานีงานทำงานในส่วนของตนเองพร้อมๆกับสถานีงานอื่นๆ ดังนั้นในเวลาเดียวกันทั้งโรงงานจะทำงานทุกสถานีงาน ผลที่ได้คือจำนวนของรถยนต์ที่ประกอบเสร็จเรียบร้อยใน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เวลาไม่นาน ถ้าเป็นกระบวนการของไปป์ไลน์คำสั่ง ก็จะแบ่งการทำงานของแต่ละคำสั่งออกเป็นขั้นตอนย่อยๆหลายขั้นตอน แต่ละขั้นตอนย่อยเรียกว่าไปป์สเตจหรือสเตจ แต่ละไปป์สเตจจะเชื่อมต่อกับไปป์สเตจอื่นและทำหน้าที่ประมวลผลในส่วนคำสั่งย่อยของตนพร้อมๆกับไปป์สเตจอื่นๆ การทำงานที่เหลื่อมขนานพร้อมกันของคำสั่งย่อยในเวลาเดียวกัน ทำให้ได้จำนวนคำสั่งที่ทำงานเสร็จสมบูรณ์ในหน่วยเวลา (Throughput) มากขึ้นเป็นจำนวนเท่ากับจำนวนของไปป์สเตจ เวลาที่ต้องการใช้สำหรับเคลื่อนย้ายคำสั่งย่อยในไปป์สเตจหนึ่งไปทำงานยังไปป์สเตจถัดไปคือ แมชชีนไซเคิล (machine cycle) เนื่องจากทุกไปป์สเตจทำงานพร้อมกัน ความยาวของแมชชีนไซเคิลจะเท่ากับเวลาของไปป์สเตจใดๆที่ทำงานช้าที่สุด ในระบบคอมพิวเตอร์แมชชีนไซเคิลนี้คือ 1 คล็อกไซเคิล (one clock cycle) เรียกสั้นๆว่า 1 ไซเคิล

จุดมุ่งหมายของนักออกแบบโปรเซสเซอร์ไปป์ไลน์คือ การทำให้ทุกไปป์สเตจมีความยาวของแมชชีนไซเคิลเท่ากัน (Balance) ถ้าทุกไปป์สเตจมีแมชชีนไซเคิลเท่ากันอย่างสมบูรณ์แล้ว เวลาที่ใช้ในการประมวลผลคำสั่ง 1 คำสั่งบนไปป์ไลน์โปรเซสเซอร์ซึ่งถือว่าเป็นเงื่อนไขอุดมคติจะเท่ากับอัตราส่วนระหว่างเวลาที่ใช้สำหรับประมวลผลหนึ่งคำสั่งบนโปรเซสเซอร์ที่ไม่ทำไปป์ไลน์ กับจำนวนของไปป์สเตจ

ภายใต้เงื่อนไขนี้ความเร็วที่เพิ่มขึ้น (Speedup) จากการทำไปป์ไลน์คำสั่งจะเท่ากับจำนวนของไปป์สเตจ อย่างไรก็ตามโดยปกติไปป์สเตจจะไม่เป็นสมดุลอย่างสมบูรณ์เนื่องจากปัญหาต่างๆที่เกิดขึ้นซึ่งจะได้กล่าวถึงในลำดับต่อไป

ในไมโครโปรเซสเซอร์เดอลุกซ์ที่ไม่ได้ทำไปป์ไลน์คำสั่งนั้น ได้แบ่งการทำงานออกเป็น 5 ไซเคิล [2] ดังนี้

1) ไซเคิลเฟ็ตช์คำสั่ง (Instruction fetch cycle: IF)

IR⟵Mem[PC]

NPC⟵PC+4

ส่งค่าโปรแกรมเคาน์เตอร์ปัจจุบันออกไปและเฟ็ตช์คำสั่งจากหน่วยความจำ ณ ตำแหน่งที่ถูกชี้โดยโปรแกรมเคาน์เตอร์เข้าไปไว้ในริจิสเตอร์คำสั่ง (IR) หลังจากนั้นทำการเพิ่มค่าของโปรแกรมเคาน์เตอร์โดยการบวก 4 เพื่อให้โปรแกรมเคาน์เตอร์ใหม่นี้ชี้ยังแอดเดรสของคำสั่งลำดับต่อไป ริจิสเตอร์ IR ใช้ในการคงค่าของคำสั่งที่อาจจะต้องใช้ในไซเคิลถัดไปเช่นเดียวกับริจิสเตอร์ NPC ที่คงค่าของโปรแกรมเคาน์เตอร์ลำดับถัดไป

2) ไซเคิลการถอดรหัสและเฟ็ตช์ริจิสเตอร์ (Instruction decode/register fetch cycle: ID)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$A \leftarrow Regs[IR_{6..10}];$

$B \leftarrow Regs[IR_{11..15}];$

$Imm \leftarrow ((IR_{16})^{16} \#\# IR_{16..31})$

ถอดรหัสคำสั่งและอ่านค่ารีจิสเตอร์จากรีจิสเตอร์ไฟล์นำไปเก็บไว้ในรีจิสเตอร์ชั่วคราว (A และ B) เพื่อใช้ไซเคิลต่อไป ส่วนค่า 16 บิตล่างของรีจิสเตอร์ IR เชื่อมกับบิตเครื่องหมายและเก็บไว้เป็นค่าอิมมิเดียท (Imm)

การถอดรหัสเกิดขึ้นพร้อมๆกับการอ่านค่าจากรีจิสเตอร์ ซึ่งสามารถกระทำได้เนื่องจากฟิลด์เหล่านี้อยู่ที่ตำแหน่งแน่นอนในฟอร์แมทคำสั่งของโปรเซสเซอร์เดอลุกซ์ การทำงานหลายงานสามารถกระทำได้เพราะการมีฟอร์แมทตำแหน่งฟิลด์ที่แน่นอน

3) ไซเคิลการทำงานและคำนวณแอดเดรส (Execution/effective address cycle: EX)

ในไซเคิลนี้เอแอลยูทำงานกับโอเปอแรนด์ที่ได้ถูกจัดเตรียมไว้แล้วจากไซเคิลก่อนหน้า การทำงานของเอแอลยูเกี่ยวข้องชนิดของคำสั่งต่อไปนี้

- การอ้างถึงหน่วยความจำ (Memory reference)

$ALUOutput \leftarrow A + Imm$

คำนวณ Effective address โดยเอแอลยูทำการบวกค่าจากโอเปอแรนด์ และเก็บผลลัพธ์ที่ได้ไว้ในรีจิสเตอร์ ALUOutput

- คำสั่งของเอแอลยูที่ประมวลผลระหว่างรีจิสเตอร์กับรีจิสเตอร์
  (Register-Register ALU Instruction)

$ALUOutput \leftarrow A \ func \ B$

เอแอลยูทำการประมวลผลตามที่ function code กำหนดโดยกระทำระหว่างรีจิสเตอร์ 2 ตัว เก็บผลลัพธ์ที่ได้ลงในรีจิสเตอร์ ALUOutput

- คำสั่งของเอแอลยูที่ประมวลผลระหว่างรีจิสเตอร์กับค่าอิมมีเดียท
  (Register-Immediate ALU Instruction)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ALUOutput$\leftarrow$A op Imm

เอแอลยูทำการประมวลผลตามที่ออปโค้ดกำหนดโดยกระทำระหว่างริจิสเตอร์ A กับริจิสเตอร์ Imm และเก็บผลลัพธ์ที่ได้ลงในริจิสเตอร์ ALUOutput

- คำสั่งทางแยก (Branch)

ALUOutput$\leftarrow$NPC + Imm

Cond$\leftarrow$(A op 0)

ทำการคำนวณหาแอดเดรสเป้าหมายของคำสั่งทางแยก โดยเอแอลยูทำการบวกค่าของ NPC กับค่าของอิมมิเดียท ริจิสเตอร์ A ซึ่งถูกอ่านค่าไว้แล้วจากไซเคิลก่อนหน้าถูกตรวจสอบว่าคำสั่งทางแยกนั้นต้องทำการเปลี่ยนลำดับของโปรแกรมเคาน์เตอร์หรือไม่

4) ไซเคิลการอ้างถึงหน่วยความจำและการประมวลผลคำสั่งทางแยกเสร็จสิ้น (Memory access/branch completion cycle: MEM)

สำหรับโปรเซสเซอร์เดอลุกซ์ เฉพาะคำสั่งการนำเข้ามา บันทึกกลับ และทางแยกเท่านั้นที่มีการทำงานในไซเคิลนี้

- การอ้างถึงหน่วยความจำ

LMD$\leftarrow$Mem[ALUOutput] or Mem[ALUOutput]$\leftarrow$B ;

มีการอ้างถึงหน่วยความจำ ถ้าเป็นคำสั่งการนำเข้า จะทำการดึงข้อมูลจากหน่วยความจำและเก็บค่าไว้ใน LMD ถ้าเป็นคำสั่งบันทึกกลับ ข้อมูลจากริจิสเตอร์ B จะถูกบันทึกลงในหน่วยความจำ ถ้าเป็นกรณีอื่นจะมีการคำนวนแอดเดรสในไซเคิลก่อนหน้าและเก็บค่าลงในริจิสเตอร์ ALUOutput

- คำสั่งทางแยก

if (cond) PC $\leftarrow$ALUOutput else PC$\leftarrow$NPC;

ถ้าเป็นคำสั่งทางแยก ค่าโปรแกรมเคาน์เตอร์ถูกแทนที่ด้วยแอดเดรสปลายทางของคำสั่งทางแยกจากริจิสเตอร์ ALUOutput ในกรณีอื่นค่าโปรแกรมเคาน์เตอร์จะเป็นค่าโปรแกรมเคาน์เตอร์ที่มีค่าเพิ่มขึ้นหรือริจิสเตอร์ NPC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5) ไซเคิลบันทึกค่ากลับ (Write-back cycle: WB)

- คำสั่งของเอแอลยูที่ประมวลผลระหว่างริจิสเตอร์กับริจิสเตอร์

$$Regs[IR_{16..20}] \leftarrow ALUOutput;$$

- คำสั่งของเอแอลยูที่ประมวลผลระหว่างริจิสเตอร์กับค่าอิมมีเดียท

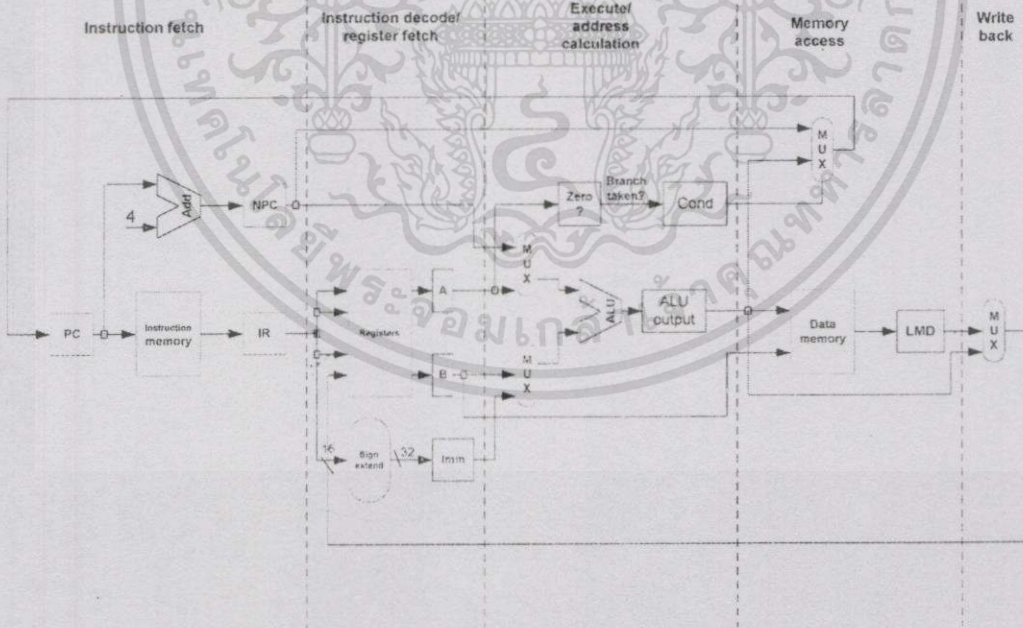
$$Regs[IR_{11..15}] \leftarrow ALUOutput;$$

- คำสั่งนำเข้ามา

$$Regs[IR_{11..15}] \leftarrow LMD ;$$

เป็นการบันทึกผลลัพธ์กลับลงไปในริจิสเตอร์ไฟล์ ไม่ว่าผลลัพธ์จะมาจากระบบหน่วยความจำหรือมาจากเอแอลยูก็ตาม ริจิสเตอร์ปลายทางขึ้นอยู่กับ Function code ที่กำหนด

รูปที่ 2.6 แสดงลำดับการทำงานในดาต้าพาธของไมโครโปรเซสเซอร์เดอลุกซ์ ที่ไม่ได้ทำไปป์ไลน์คำสั่ง โดยระหว่างไซเคิลจะมีริจิสเตอร์ชั่วคราวเก็บค่าต่างๆสำหรับใช้ในไซเคิลต่อไป



รูปที่ 2.6 ดาต้าพาธไมโครโปรเซสเซอร์เดอลุกซ์ที่ไม่ได้ทำไปป์ไลน์คำสั่ง [2]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูป 2.6 ซึ่งเป็นดาต้าพาธของไมโครโปรเซสเซอร์เดอลุกซ์ที่ยังไม่ได้ทำไปป์ไลน์คำสั่ง สามารถปรับปรุงดาต้าพาธดังกล่าวให้เป็นดาต้าพาธที่แสดงการทำไปป์ไลน์คำสั่งได้ โดยแต่ละไซเคิลจะทำหน้าที่เป็นการทำงานในแต่ละไปป์สเตจ การทำงานของแต่ละไปป์สเตจจะเป็นดังรูปที่ 2.7 แต่ละคำสั่งสามารถทำงานเสร็จสมบูรณ์ได้โดยใช้เวลา 5 ไซเคิล

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction i +1		IF	ID	EX	MEM	WB			
Instruction i +2			IF	ID	EX	MEM	WB		
Instruction i +3				IF	ID	EX	MEM	WB	
Instruction i +4					IF	ID	EX	MEM	WB

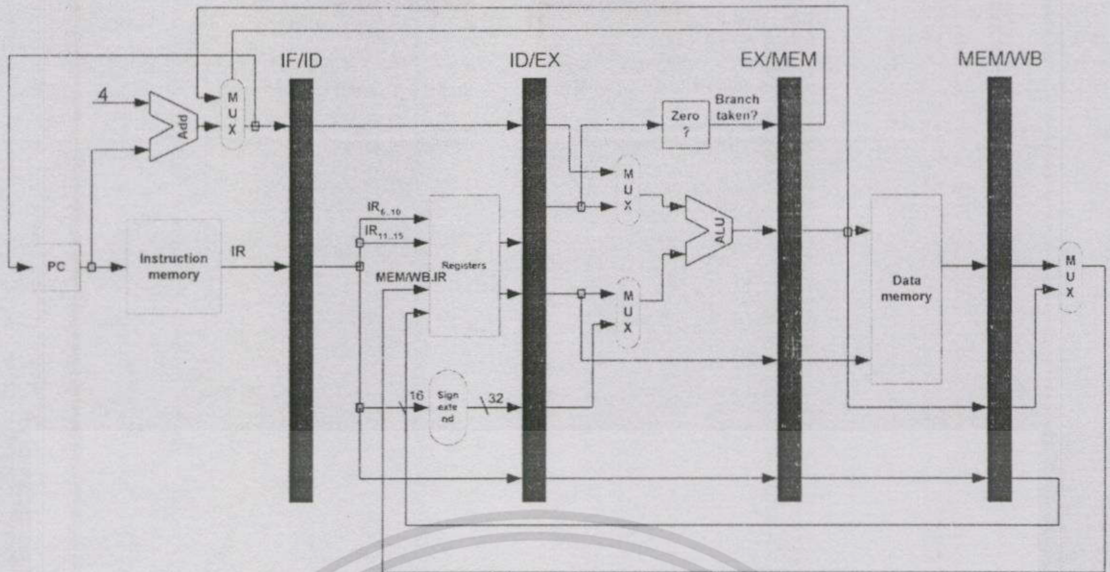
รูปที่ 2.7 การทำงานในแต่ละไปป์สเตจของไปป์ไลน์เดอลุกซ์ [2]

ระหว่างไปป์สเตจทั้ง 5 จะมีไปป์ไลน์รีจิสเตอร์คั่นระหว่างไป้สเตจจำนวน 4 ตัวทำหน้าที่เก็บค่ารีจิสเตอร์บางค่าไว้เพื่อใช้ในไปป์สเตจถัดไป คือ

1. Fetch/Decode register: IF/ID
   เป็นไปป์ไลน์รีจิสเตอร์ที่คั่นระหว่างสเตจ IF และ ID
2. Decode/Execute register: ID/EX
   เป็นไปป์ไลน์รีจิสเตอร์ที่คั่นระหว่างสเตจ ID และ EX
3. Execute/Memory register: EX/MEM
   เป็นไปป์ไลน์รีจิสเตอร์ที่คั่นระหว่างสเตจ EX และ MEM
4. Memory/Writeback register: MEM/WB
   เป็นไปป์ไลน์รีจิสเตอร์ที่คั่นระหว่างสเตจ MEM และ WB

โครงสร้างของ ดาต้าพาธ ของโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์แสดงดังรูปที่ 2.8

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.8 โครงสร้างดาต้าพาธของไมโครโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ [2]

2.3 การวัดประสิทธิภาพ

การทำไปป์ไลน์คำสั่งจะช่วยเพิ่มจำนวนของคำสั่งที่ได้ประมวลผลเสร็จสมบูรณ์แล้วต่อหน่วยเวลา แต่การทำไปป์ไลน์คำสั่งไม่ได้เป็นการลดเวลาในการประมวลผลในแต่ละส่วนของคำสั่งแต่อย่างใด ความเป็นจริงแล้วบางครั้งอาจจะทำให้เวลาในการประมวลผลแต่ละคำสั่งเพิ่มขึ้นเล็กน้อยด้วยเนื่องจากโอเวอร์เฮด (Overhead) ของไปป์ไลน์คอนโทรล (Pipeline control) การเพิ่มขึ้นของจำนวนคำสั่งที่ทำงานเสร็จสมบูรณ์ในหน่วยเวลาเป็นสิ่งที่แสดงให้เห็นว่าโปรแกรมสามารถทำงานได้เร็วขึ้นและส่งผลให้เวลาที่ใช้ในการประมวลผลทั้งหมดต่ำลง [3]

จากหลักการที่กล่าวว่าการทำไปป์ไลน์คำสั่งทำให้โปรเซสเซอร์มีความเร็วเพิ่มขึ้นมากกว่าโปรเซสเซอร์ที่ไม่มีการทำไปป์ไลน์คำสั่งเป็นจำนวนเท่าเท่ากับจำนวนไปป์สเตจ แต่ข้อจำกัดในทางปฏิบัติที่ทำให้ไม่สามารถสร้างให้ไปป์ไลน์คำสั่งมีจำนวนไปป์สเตจหรือความลึกของไปป์ไลน์ (Pipeline depth)ได้มากตามต้องการคือ

1. เวลาในการประมวลผล (Pipeline latency) เวลาในการประมวลผลแต่ละคำสั่งไม่ได้ลดลงเมื่อทำไปป์ไลน์คำสั่ง ทำให้การเพิ่มไปป์สเตจมีข้อจำกัด

2. ความไม่สมดุลของแต่ละไปป์สเตจ (Imbalance among pipeline stage) ไม่ว่าจำนวนไปป์ - สเตจจะเป็นเท่าใดก็ตาม แต่เวลาที่แต่ละไปป์สเตจใช้ทำงานไม่เท่ากัน (ไม่สมดุล) จะใช้เวลาของสเตจที่ทำงานช้าที่สุดเป็นตัวกำหนดการทำงานของสัญญาณนาฬิกา ดังนั้นการไม่สมดุลระหว่างไปป์สเตจจึงทำให้ประสิทธิภาพของไปป์ไลน์ลดลง

3. ค่าโอเวอร์เฮด เกิดจากผลรวมของไปป์ไลน์รีจิสเตอร์ดีเลย์ (setup time + propagation delay) และ clock skew

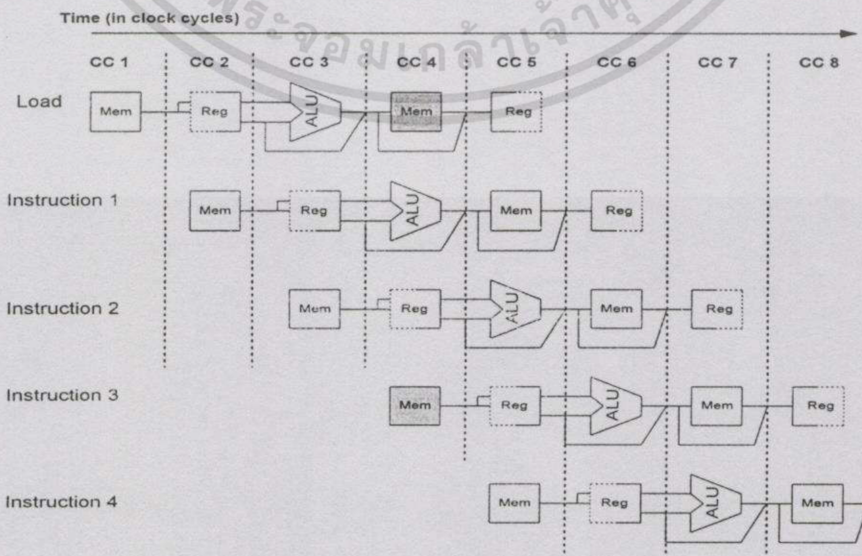
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
ปัญหาของไปป์ไลน์คำสั่ง

ปัญหาในไปป์ไลน์คำสั่งเกิดขึ้นจากคำสั่งไม่สามารถถูกนำเข้ามาประมวลผลได้ตามปกติ ไมโครโปรเซสเซอร์จะต้องมีระบบจัดการเพื่อให้การทำงานของไปป์ไลน์คำสั่งดำเนินต่อไป การจัดให้บางคำสั่งหยุดชะงักหรือหน่วงเวลาการทำงานของบางคำสั่งเพื่อให้คำสั่งอื่นทำงานเสร็จก่อนแล้วจึงนำคำสั่งที่ถูกหน่วงไว้มาทำงานใหม่นั้นเป็นการแก้ไขปัญหาในไปป์ไลน์ได้ระดับหนึ่ง แต่มีผลให้ประสิทธิภาพของไปป์ไลน์คำสั่งไม่เป็นไปตามทฤษฎี ปัญหาในไปป์ไลน์คำสั่งแบ่งได้เป็น 3 ประเภทคือ

3.1 ปัญหาโครงสร้างฮาร์ดแวร์ (Structural hazards)

ด้วยสถาปัตยกรรมของไปป์ไลน์คำสั่ง การทำงานที่เหลื่อมซ้อนกันของคำสั่งต้องการการสนับสนุนทางทรัพยากรฮาร์ดแวร์ด้วยโดยหน่วยทำงาน (Functional unit) ต้องมีจำนวนมากกว่าหนึ่ง (Duplication of resources) ถ้าฮาร์ดแวร์ไม่สามารถสนับสนุนการทำงานของคำสั่งที่เหลื่อมซ้อนกันนั้นได้ เช่น ทรัพยากรบางส่วนที่ต้องใช้งานร่วมกันมีจำนวนเดียว ทำให้การทำงานของคำสั่งไม่สามารถเหลื่อมซ้อนขนานกันได้ หรือหน่วยการทำงานบางหน่วยไม่เป็นไปป์ไลน์อย่างสมบูรณ์ เมื่อมีคำสั่งที่ถูกนำเข้ามาประมวลผลใช้หน่วยการทำงานที่ไม่เป็นไปป์ไลน์นั้น ทำให้คำสั่งหนึ่งคำสั่งไม่สามารถถูกประมวลผลเสร็จสิ้นได้ในไซเคิลเดียว ปัญหาเนื่องจากความเข้ากันไม่ได้ของทรัพยากรฮาร์ดแวร์ (Conflict resource) จะเกิดขึ้นทำให้ประสิทธิภาพของโปรเซสเซอร์ลดลง ดังนั้นเมื่อใดก็ตามที่ไปป์ไลน์มีปัญหาโครงสร้างฮาร์ดแวร์ กระบวนการในไปป์ไลน์จะหยุดชะงักการทำงาน ซึ่งมีผลให้จำนวนไซเคิลต่อคำสั่ง (CPI) เพิ่มขึ้น

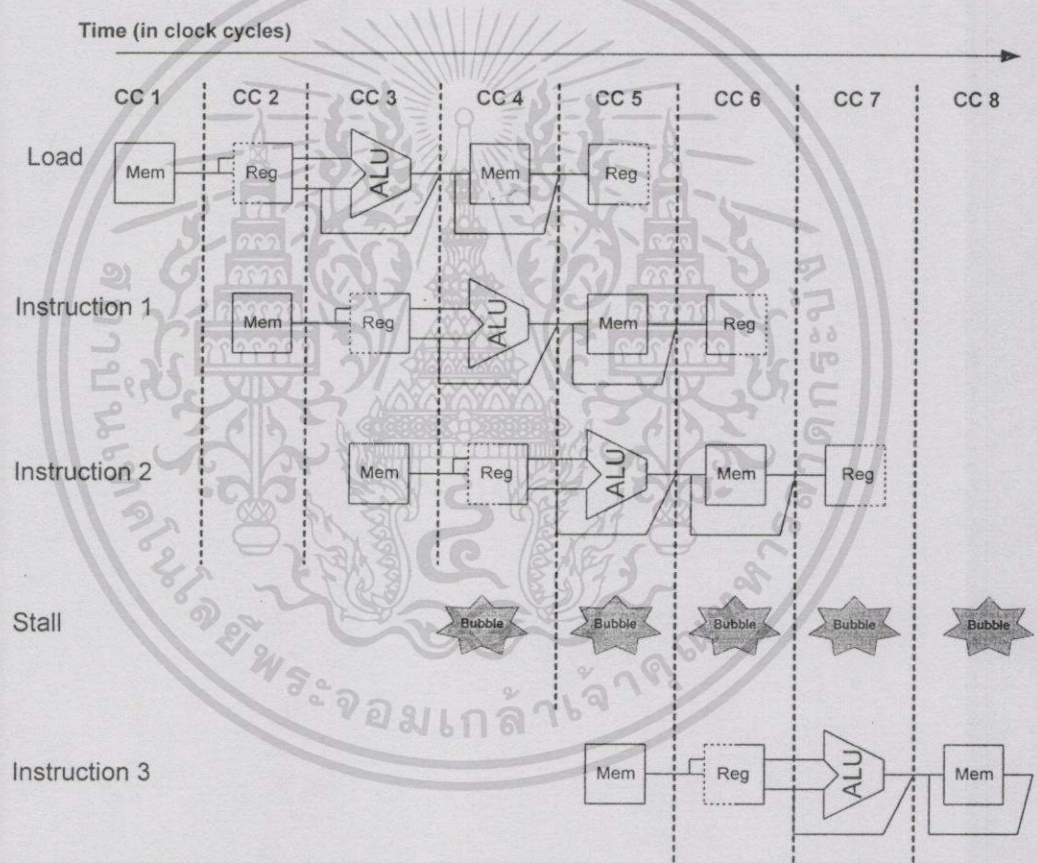


รูปที่ 3.1 ปัญหาการขัดแย้งของทรัพยากรเมื่อ 2 คำสั่งต้องอ้างอิงข้อมูลจากหน่วยความจำเดียว [3]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

รูปที่ 3.1 แสดงให้เห็นว่าสำหรับโปรเซสเซอร์ที่มีหน่วยความจำเดียว (Single memory) จะเกิดปัญหาการขัดแย้งขึ้นในไซเคิล 4 เมื่อคำสั่ง Load อ้างถึงข้อมูลจากหน่วยความจำพร้อมๆกับที่คำสั่ง instruction 3 ต้องการเฟ็ทช์คำสั่งจากหน่วยความจำเดียวกัน ไม่สามารถทำงานใดงานหนึ่งให้เสร็จเรียบร้อยได้ การแก้ปัญหาการขัดแย้งดังกล่าวสามารถแก้ไขโดยให้ไปป์ไลน์มีการหน่วงเวลาไป 1 ไซเคิล ช่วงเวลา 1 ไซเคิลนั้นให้คำสั่ง Load ได้ทำการเรียกใช้ข้อมูลจากหน่วยความจำจนเสร็จเรียบร้อย หลังจากนั้นคำสั่ง Instruction 3 จึงจะอ่านคำสั่งจากหน่วยความจำนั้น การกระทำดังกล่าวจะทำให้ไม่เกิดปัญหาโครงสร้างฮาร์ดแวร์ เมื่อมีการอ้างถึงข้อมูลในหน่วยความจำเดียว รูปที่ 3.2 ซึ่งแสดงการหน่วงเวลาที่เกิดขึ้นในไปป์ไลน์ ซึ่งการหน่วงเวลานั้นเสมือนกับการแทรกช่องว่าง (pipeline bubble หรือ bubble) เข้าไปในการทำงานของคำสั่ง



รูปที่ 3.2 การหน่วงเวลาในไปป์ไลน์เมื่อมีปัญหาโครงสร้างฮาร์ดแวร์ [3]

ไดอะแกรม (Diagram) ที่แสดงให้เห็นพฤติกรรมการทำงานในแต่ละไปป์สเตจเมื่อมีการหน่วงเวลาแสดงได้ดังรูปที่ 3.3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Clock cycle number										
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Load instruction	IF	ID	EX	MEM	WB					
Instruction 1		IF	ID	EX	MEM	WB				
Instruction 2			IF	ID	EX	MEM	WB			
				Stall	Stall	Stall	Stal!	Stall		
Instruction 3					IF	ID	EX	MEM	WB	
Instruction 4						IF	ID	EX	MEM	WB
Instruction 5							IF	ID	EX	MEM
Instruction 6								IF	ID	EX

รูปที่ 3.3 พฤติกรรมของแต่ละไปป์สเตจเมื่อมีการหน่วงเวลาในไปป์ไลน์สำหรับปัญหาโครงสร้างฮาร์ดแวร์

จากรูป 3.3 การหน่วงเวลาที่เกิดขึ้นทำให้คำสั่ง Instruction 3 ถูกเลื่อนการทำงานออกไป และทำงานเสร็จในไซเคิลที่ 9 สำหรับโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์จะไม่เกิดปัญหาในเรื่องโครงสร้างฮาร์ดแวร์เนื่องจากถูกออกแบบให้มีลักษณะดังต่อไปนี้

1. ริจิสเตอร์ไฟล์มีพอร์ตสำหรับอ่าน 2 พอร์ต และพอร์ตบันทึก 1 พอร์ต สเตจ ID ใช้พอร์ตสำหรับการอ่าน และสเตจ WB ใช้พอร์ตบันทึก จึงไม่เกิดการขัดแย้งกันของฮาร์ดแวร์

2. การอ่านข่าวสาร (ข้อมูลหรือคำสั่ง) โดยสเตจ IF และสเตจ MEM จากหน่วยความจำที่แยกจากกันอย่างชัดเจน สเตจ IF เฟ็ทช์คำสั่งจากหน่วยความจำคำสั่ง ส่วนสเตจ MEM อ่านและบันทึกข้อมูลกับหน่วยความจำข้อมูล

3. มีการกำหนดให้เอแอลยูที่อยู่ในสเตจ EX สามารถประมวลผลคำสั่งคณิตศาสตร์ใดๆ ที่เป็นจำนวนเต็มให้เสร็จได้ในไซเคิลเดียว

3.2 ปัญหาข้อมูล (Data hazards)

ปัญหาข้อมูลหรือเรียกอีกอย่างหนึ่งตามลักษณะการเกิดปัญหาในไปป์ไลน์คำสั่งว่าเป็นปัญหาของความต้องการการใช้ข้อมูลร่วมกัน คือ คำสั่งปัจจุบันต้องการใช้ผลลัพธ์จากคำสั่งก่อนหน้าขณะที่คำสั่งก่อนหน้ายังประมวลผลลัพธ์ไม่เสร็จ ทำให้คำสั่งปัจจุบันนำข้อมูลที่ผิดพลาดมาประมวลผล

ปัญหาการขึ้นต่อกันของข้อมูลสามารถแบ่งได้ 3 ประเภทตามลำดับของการอ่านและการบันทึกของคำสั่ง และเรียกชื่อปัญหาข้อมูลตามลำดับการทำงานในโปรแกรม ถ้ามีคำสั่ง 2 คำสั่งคือ คำสั่ง i และคำสั่ง j โดยที่คำสั่ง i ทำงานก่อนคำสั่ง j แล้ว ปัญหาข้อมูลที่เกิดขึ้นคือ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1. RAW (Read After Write)

คำสั่ง j พยายามอ่านค่าจากริจิสเตอร์ต้นทางก่อนที่คำสั่ง i จะบันทึกค่าริจิสเตอร์นั้นเสร็จ ดังนั้น คำสั่ง j จะได้รับค่าริจิสเตอร์เก่า

2.WAW (Write After Write)

คำสั่ง j พยายามบันทึกค่าลงริจิสเตอร์โอเปอแรนด์ก่อนที่จะมีการบันทึกโดยคำสั่ง i ซึ่งเป็นคำสั่งที่ต้องทำงานก่อน การบันทึกค่าโอเปอแรนด์ที่ผิดลำดับนี้ มักจะส่งผลให้มีโอกาสที่การละทิ้งการบันทึกค่าของคำสั่ง i เกิดขึ้นได้มากกว่าการละทิ้งการบันทึกค่าของคำสั่ง j ปัญหาข้อมูลประเภทนี้เกิดขึ้นในไปป์ไลน์ที่มีการบันทึกในไปป์สเตจมากกว่าหนึ่งไปป์สเตจพร้อมๆกัน หรือในไปป์ไลน์ที่ยินยอมให้คำสั่งทำงานต่อไปได้เมื่อคำสั่งก่อนหน้าถูกหน่วงเวลาอยู่ สำหรับโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ มีการบันทึกค่าริจิสเตอร์ได้เฉพาะในสเตจ WB

3. WAR (Write After Read)

คำสั่ง j พยายามบันทึกค่าลงริจิสเตอร์ปลายทางก่อนที่จะมีการอ่านค่านี้โดยคำสั่ง i ดังนั้น คำสั่ง i จะได้รับค่าใหม่ที่ไม่ถูกต้องถ้าคำสั่ง j ได้บันทึกค่าเรียบร้อยแล้ว สำหรับโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ปัญหาข้อมูลประเภทนี้จะไม่เกิดขึ้นเนื่องจากการอ่านค่าริจิสเตอร์เกิดขึ้นในตอนต้นของไปป์ไลน์คือสเตจ ID และบันทึกค่าของริจิสเตอร์ในสเตจ WB

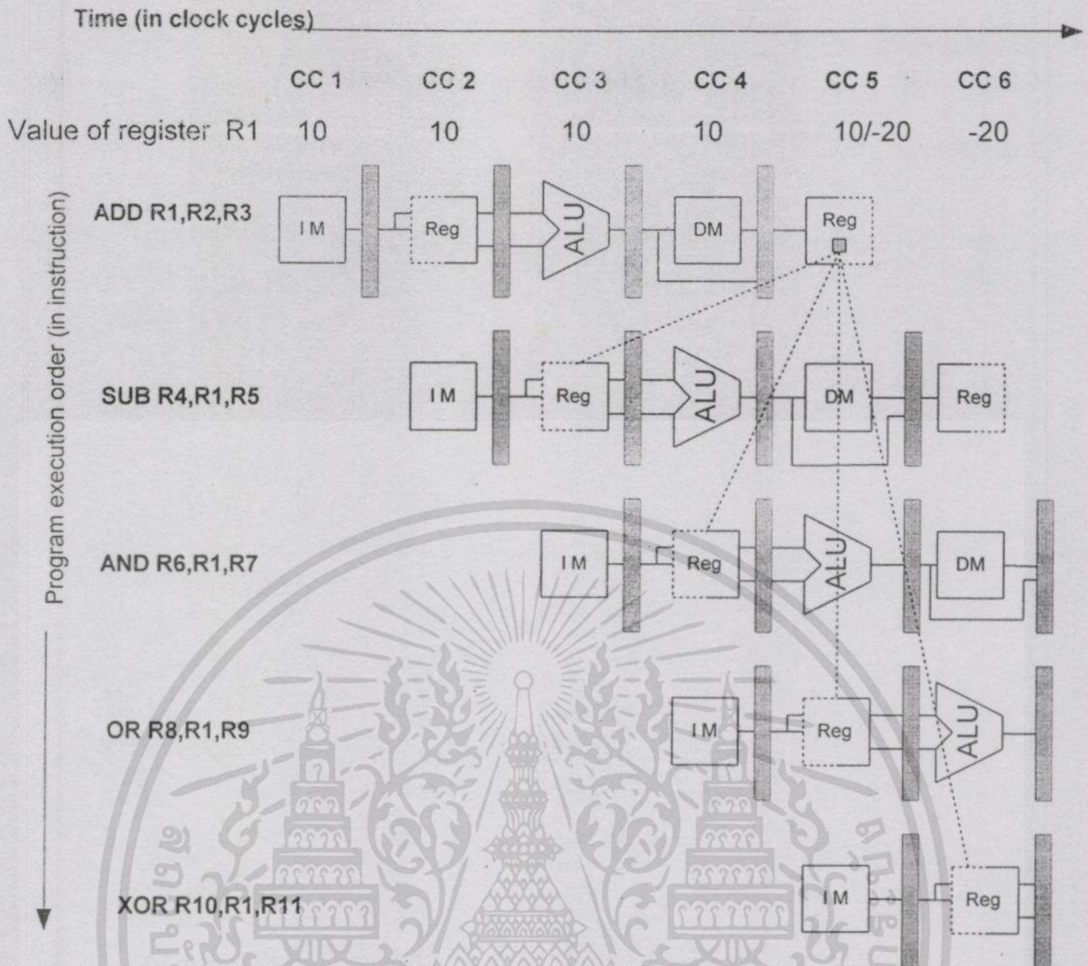
พิจารณาชุดคำสั่งในรูป 3.4 ดังนี้

คำสั่ง	**ไซเคิลการทำงาน**					
	1	2	3	4	5	6
ADD **R1**,R2,R3	IF	ID	EX	MEM	WB	
SUB R4,**R1**,R5		IF	ID	EX	MEM	WB
AND R6,**R1**,R7			IF	ID	EX	MEM
OR R8,**R1**,R9				IF	ID	EX
XOR R10,**R1**,R11					IF	ID

รูปที่ 3.4 ไซเคิลการทำงานของชุดคำสั่ง

จากรูปที่ 3.4 จะเห็นว่าทุกคำสั่งที่ตามหลังคำสั่ง ADD ต้องใช้ผลลัพธ์ของ R1 จากคำสั่ง ADD ถ้าสมมติว่าก่อนที่จะทำคำสั่ง ADD ในตอนเริ่มต้นค่าใน R1 มีค่าเป็น 10 และหลังจากที่ทำคำสั่ง ADD แล้วให้ R1 มีค่า -20 ดังนั้นค่า R1 ที่คำสั่งอื่นๆจะต้องนำไปใช้คือ -20 เมื่อนำชุดคำสั่งจากรูป 3.4 มาพิจารณาร่วมกับการเปลี่ยนแปลงค่าของข้อมูลตามไซเคิลการทำงานต่างๆ ดังแสดงไว้ในรูปที่ 3.5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.5 ลำดับการทำงานของไปป์ไลน์เมื่อเกิดปัญหาการใช้ข้อมูลร่วมกัน [3]

จากรูปที่ 3.5 คำสั่ง ADD เก็บค่ารีจิสเตอร์ผลลัพธ์ลงในขั้นตอน WB ในไซเคิลที่ 5 พิจารณาคำสั่งที่ตามมาทั้ง 4 คำสั่ง ดังนี้

1. คำสั่ง SUB จะอ่านค่ารีจิสเตอร์ R1 ในไซเคิลที่ 3 ซึ่งข้อมูล R1 ที่ได้มีค่าเป็น 10 (ได้รับข้อมูล R1 ผิดพลาด)

2. คำสั่ง AND จะอ่านค่ารีจิสเตอร์ R1 ในไซเคิลที่ 4 ซึ่งข้อมูล R1 ที่ได้มีค่าเป็น 10 (ได้รับข้อมูล R1 ผิดพลาด)

3. คำสั่ง OR อ่านค่ารีจิสเตอร์ R1 ในไซเคิลที่ 5 ซึ่งเป็นจังหวะเดียวกับที่คำสั่ง ADD เก็บผลลัพธ์ลงใน R1 พอดี ในกรณีนี้จะสามารถรับข้อมูล R1 ที่ถูกต้องได้

4. คำสั่ง XOR เนื่องจาก R1 ในคำสั่ง ADD ทำการเขียนข้อมูลลงรีจิสเตอร์ในไซเคิลการทำงานที่ 5 แต่คำสั่ง XOR นี้ทำการอ่านข้อมูลในขั้นตอน ID ในไซเคิลการทำงานที่ 6 จึงได้รับค่า R1 ที่ถูกต้อง

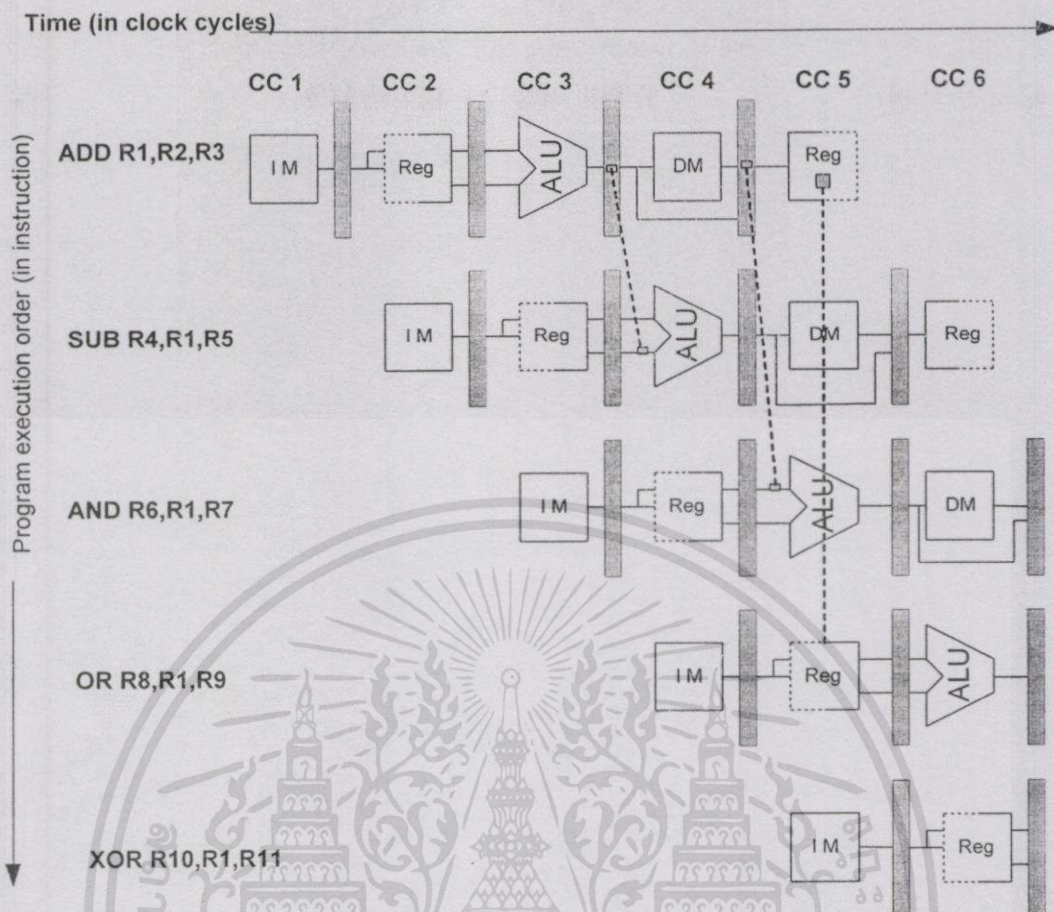
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

คำสั่งถัดจากคำสั่ง ADD คือ SUB และ AND จะเกิดปัญหาเนื่องจากข้อมูล R1 ไม่ถูกต้อง สามารถแก้ปัญหาแบบง่ายๆได้โดยใช้การหน่วงเวลาให้โปรเซสเซอร์ได้ทำการเขียนข้อมูล R1ของคำสั่ง ADD ที่ขั้นตอน WB ในไซเคิลที่ 5 ให้เสร็จก่อน หลังจากนั้นคำสั่ง SUB และ AND จึงอ่านข้อมูล R1 มาประมวลผล แต่วิธีนี้ไม่เหมาะสมในทางปฏิบัติ เพราะทำให้ประสิทธิภาพในการประมวลผลของโปรเซสเซอร์ลดลง สามารแก้ปัญหาการใช้ข้อมูลร่วมกันได้ดังนี้

1. การแก้ปัญหาด้วยวิธีการฟอร์เวิร์ดดิ้ง (Data forwarding)

จากปัญหาที่เกิดขึ้นดังรูปที่ 3.5 คือ คำสั่งตามหลังคำสั่ง ADD เพียง 2 คำสั่งเท่านั้นที่จะเกิดปัญหาการใช้ข้อมูลร่วมกัน ถ้าพิจารณาอย่างละเอียดจะพบว่าคำสั่ง ADD ได้ผลลัพธ์จากการประมวลผลเมื่อจบไซเคิลการทำงานที่ 3 โดยที่ผลลัพธ์เก็บอยู่ในไปป์ไลน์ริจิสเตอร์ EX/MEM ถ้าสามารถนำข้อมูลผลลัพธ์จากไปป์ไลน์ริจิสเตอร์ ส่งกลับไปยังเอแอลยูได้ก่อนที่คำสั่ง SUB จะเริ่มประมวลผลในไซเคิลการทำงานที่ 4 จะสามารถแก้ปัญหาระหว่างคำสั่ง SUB และคำสั่ง ADD ได้ แต่สำหรับคำสั่ง AND ต้องรอให้คำสั่ง ADD ทำงานจบไซเคิลการทำงานที่ 4 ซึ่งผลลัพธ์จะอยู่ในไปป์ไลน์ริจิสเตอร์ MEM/WB จึงจะสามารถนำข้อมูลผลลัพธ์ส่งกลับไปยังเอแอลยูได้ก่อนที่คำสั่ง AND จะเริ่มประมวลผลในไซเคิลการทำงานที่ 5 จากที่กล่าวมาข้างต้นสามารถสรุปหลักในการแก้ปัญหาข้อมูลด้วยวิธีฟอร์เวิร์ดดิ้ง คือ ต้องส่งผ่านข้อมูลจากไปป์ไลน์ริจิสเตอร์ EX/MEM หรือ MEM/WB ไปยังมัลติเพล็กเซอร์ (Multiplexer) ที่ขาเอแอลยู ณ สเตจ EX หรือมัลติเพล็กเซอร์ ณ สเตจ MEM ก่อนที่คำสั่งที่ตามมาจะเริ่มประมวลผล ดังนั้นงานขั้นแรกของการแก้ปัญหาคือ ต้องปรับโครงสร้างของดาต้าพาธโดยเพิ่มเส้นทางการไหลของข้อมูลผลลัพธ์จากไปป์ไลน์ริจิสเตอร์ EX/MEM หรือ MEM/WB กลับไปยังสเตจ EX และสเตจ MEM ขั้นตอนที่สองของการแก้ปัญหาคือต้องทำการวิเคราะห์ชุดคำสั่งอย่างรอบคอบว่ามีกรณีใดบ้างที่ทำให้เกิดปัญหาการใช้ข้อมูลร่วมกันและสามารถนำค่าที่ป้อนกลับจากไปป์ไลน์ริจิสเตอร์ EX/MEM หรือ MEM/WB มาใช้ได้ทันก่อนที่คำสั่งที่ตามมาจะเริ่มประมวลผล และขั้นตอนสุดท้ายคือนำเงื่อนไขในกรณีต่างๆที่ได้พิจารณาอย่างรอบคอบจากขั้นตอนที่สองนั้นมาสร้างเป็นหน่วยตรวจสอบการฟอร์เวิร์ด (Forwarding Unit) การทำงานของหน่วยนี้คล้ายวงจรเปรียบเทียบ (Comparator) อินพุทจะเป็นตำแหน่งของรีจิสเตอร์ซึ่งมาจากข้อมูลในไปป์ไลน์ริจิสเตอร์ ID/EX EX/MEM และ MEM/WB โดยการทำงานจะนำข้อมูลอินพุทมาเปรียบเทียบว่าคำสั่งก่อนหน้ากับคำสั่งปัจจุบันใช้ริจิสเตอร์ตัวเดียวกันและเป็นไปตามเงื่อนไขการฟอร์เวิร์ดหรือไม่ ถ้าเป็นไปตามเงื่อนไขหน่วยฮาร์ดแวร์ที่สร้างขึ้นจะส่งสัญญาณไปเลือกค่าอินพุทของมัลติเพล็กเซอร์ที่สเตจ EX และ สเตจ MEM การแก้ปัญหาด้วยวิธีฟอร์เวิร์ดดิ้งแสดงได้ดังรูป 3.6

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.6 การแก้ปัญหาข้อมูลด้วยวิธีฟอร์เวิร์ดดิ้ง [3]

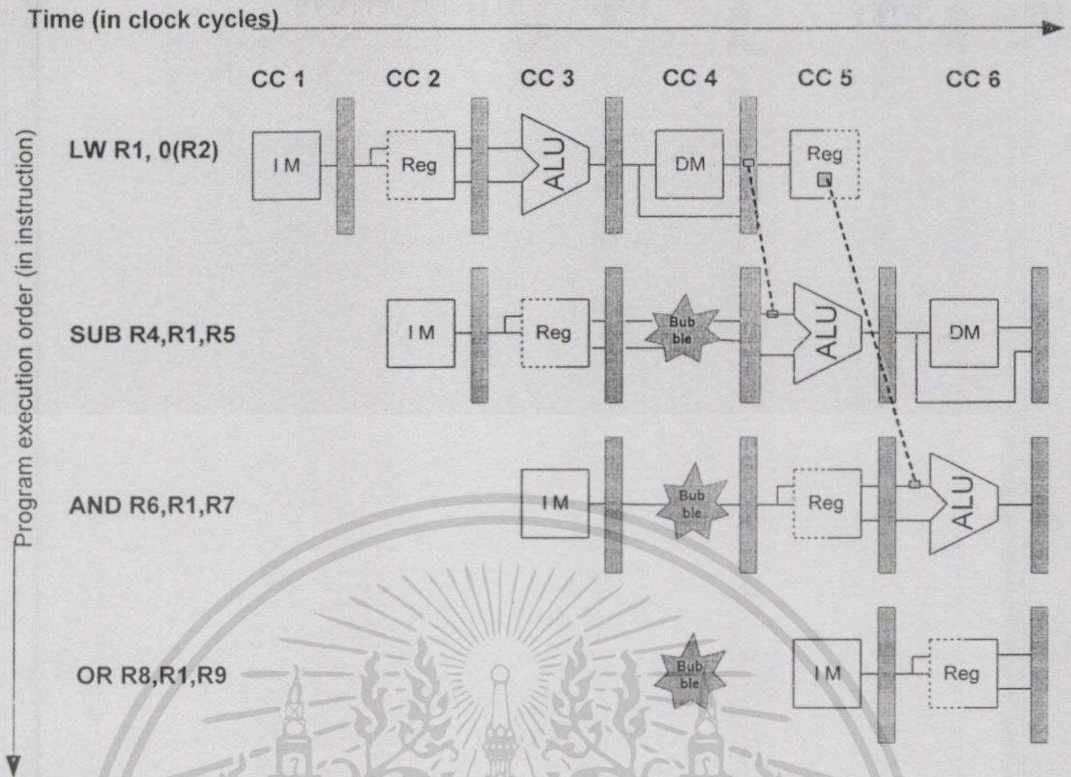
2. การแก้ปัญหาด้วยการหน่วงเวลา

   พิจารณาส่วนของโค้ดโปรแกรมดังนี้

LW	R1,0(R2)
SUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9

จากตัวอย่างของโค้ดโปรแกรมดังกล่าว จะเห็นว่าคำสั่งที่ตามหลังคำสั่ง LW ยกเว้นคำสั่ง Store ไม่สามารถแก้ปัญหาด้วยวิธีฟอร์เวิร์ดดิ้งได้เนื่องจากไม่สามารถฟอร์เวิร์ดข้อมูลกลับไปในอดีตได้ จะต้องฟอร์เวิร์ดไปในปัจจุบันและอนาคตเท่านั้น ส่วนคำสั่งอื่นๆจะทำการฟอร์เวิร์ดได้ ดังนั้น จะต้องเพิ่มฮาร์ดแวร์ในการตรวจสอบการหน่วงเวลาในสเตจ ID เพื่อให้คำสั่งใดๆที่ตามหลัง Load ยกเว้น Store ได้รับข้อมูลที่ถูกต้อง รูปที่ 3.7 แสดงการแก้ปัญหาข้อมูลด้วยการหน่วงเวลา

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.7 การแก้ปัญหาข้อมูลด้วยการหน่วงเวลา [3]

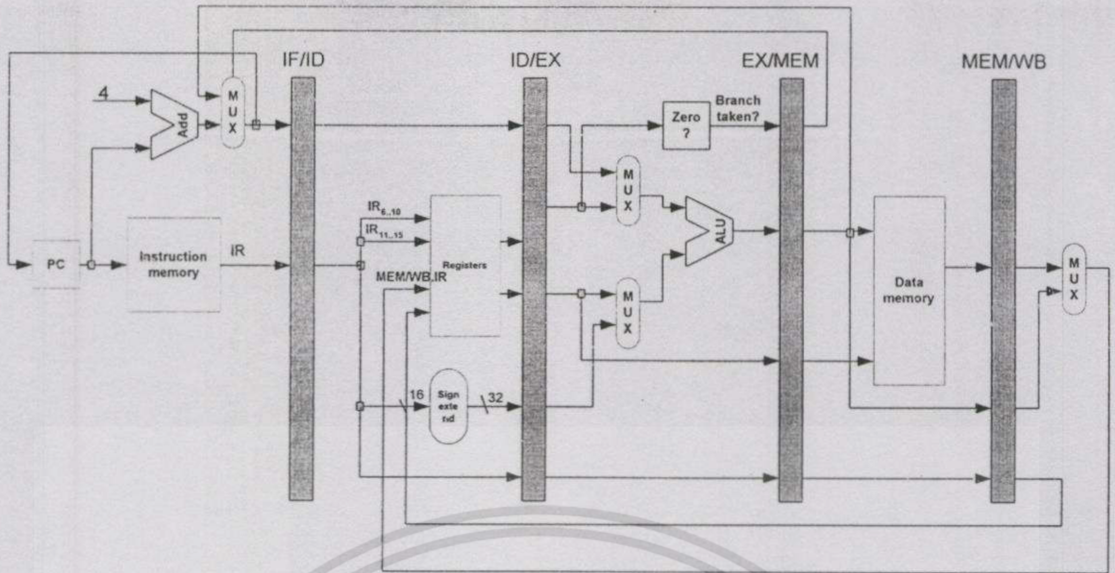
3.3 ปัญหาการควบคุม (Control hazards)

ปัญหาการควบคุมเป็นสาเหตุสำคัญที่ทำให้ประสิทธิภาพของไปป์ไลน์คำสั่งลดลง ในจำนวนคำสั่งที่ต้องแยกหรือกระโดดไปทำงานทั้งหมด (Branch, jump, call หรือ return) คำสั่งทางแยกแบบมีเงื่อนไข (Conditional branch) จะมีจำนวนมากและควบคุมได้ยากที่สุดเนื่องจากจะต้องมีการทดสอบค่าของริจิสเตอร์ เมื่อคำสั่งทางแยกถูกนำเข้ามาประมวลผล คำสั่งทางแยกนั้นอาจจะทำให้มีการเปลี่ยนแปลงลำดับของโปรแกรมเคาน์เตอร์ไปจากปกติ (PC+4) หรือไม่ก็ได้ ถ้าคำสั่งทางแยกนั้นทำให้โปรแกรมเคาน์เตอร์เปลี่ยนไปอยู่ที่ตำแหน่งเป้าหมาย (Target address) คำสั่งทางแยกนั้นมีพฤติกรรมเป็น Taken แต่ถ้าคำสั่งทางแยกนั้นไม่ทำให้โปรแกรมเคาน์เตอร์เปลี่ยนไปอยู่ที่ตำแหน่งเป้าหมายพฤติกรรมของคำสั่งทางแยกนั้นเป็น Not taken

3.3.1 การแก้ปัญกาการควบคุมโดยการปรับปรุงโครงสร้างดาต้าพาธของโปรเซสเซอร์เดอลุกซ์

ชุดคำสั่งของไมโครโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์มีคำสั่งทางแยก 2 คำสั่งซึ่งมีรูปแบบคำสั่งชนิด I-type และมีการทดสอบริจิสเตอร์ rs1 ว่าเป็นศูนย์หรือไม่เป็นศูนย์โดยใช้คำสั่ง BEQZ และคำสั่ง BNEZ ตามลำดับ ดาต้าพาธของไปป์ไลน์เดอลุกซ์ที่แสดงส่วนการทดสอบคำสั่งทางแยกเป็นดังรูปที่ 3.8

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.8 ดาต้าพาธของโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ที่มีการตรวจสอบคำสั่งทางแยก [2]

พิจารณารูป 3.8 พบว่าเมื่อไมโครโปรเซสเซอร์เฟ็ตช์คำสั่งทางแยกเข้ามา กว่าที่ไมโครโปรเซส-เซอร์จะคำนวณตำแหน่งเป้าหมายและเงื่อนไขของคำสั่งทางแยกเสร็จสิ้น คำสั่งอื่นๆที่ตามหลังคำสั่งทางแยกนั้นก็ถูกเฟ็ตช์เข้ามาในไปป์ไลน์คำสั่งจำนวนหนึ่งแล้ว ถ้าคำสั่งทางแยกที่ถูกนำเข้ามาประมวลผลนั้นมีพฤติกรรมเป็น Taken โปรแกรมเคาน์เตอร์จะต้องเปลี่ยนไปอยู่ที่ตำแหน่งเป้าหมายของคำสั่งทางแยก ทำให้ต้องยกเลิกคำสั่งหลังคำสั่งทางแยกนั้นทันที แล้วไมโครโปรเซสเซอร์ก็จะเฟ็ตช์คำสั่งจากตำแหน่งเป้าหมายเข้ามาประมวลผลแทน การยกเลิกคำสั่งหลังคำสั่งทางแยกและเฟ็ตช์คำสั่งใหม่ดังกล่าวทำให้ไมโครโปรเซสเซอร์เสียเวลาในการประมวลผลคำสั่งไปทั้งหมด 3 ไซเคิลสำหรับแต่ละคำสั่งทางแยก ไซเคิลการทำงานที่เสียไปแสดงในรูปที่ 3.9

Branch	IF	**ID**	EX	**MEM**	WB				
Branch successor		IF (Stall)	Stall	Stall	IF	ID	EX	MEM	WB
Branch successor + 1						IF	ID	EX	MEM	WB

รูปที่ 3.9 พฤติกรรมในไปป์ไลน์เมื่อพบคำสั่งทางแยก

จากรูปที่ 3.9 อธิบายได้ว่ากว่าที่ไปป์ไลน์จะพบคำสั่งทางแยกจะต้องมีการหน่วงเวลาถึง 3 ไซเคิล เนื่องจากจะรู้ว่าคำสั่งที่ถูกนำเข้ามาประมวลผลนั้นเป็นคำสั่งทางแยกเมื่อถึงสเตจ ID การ

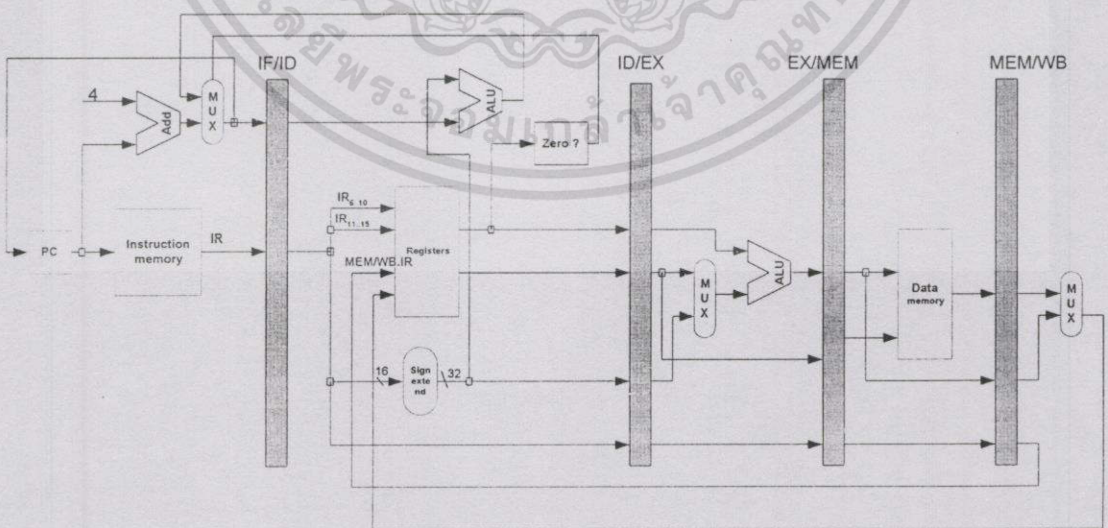
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

คำนวณตำแหน่งเป้าหมายมีขึ้นในสเตจ EX และจะรู้เงื่อนไขหรือผลลัพธ์พฤติกรรมของคำสั่งทางแยกเมื่อถึงสเตจ MEM เวลาทั้งหมด 3 ไซเคิลที่สูญเสียไปสำหรับทุกๆคำสั่งทางแยกเป็นการสูญเสียที่สำคัญมาก เนื่องจากถ้าความถี่ของการเป็นคำสั่งทางแยกในโปรแกรมเท่ากับ 30 % และค่าของจำนวนไซเคิลต่อคำสั่งทางอุดมคติมีค่าเป็น 1 มีผลให้โปรเซสเซอร์ที่สถาปัตยกรรมไปป์ไลน์มีประสิทธิภาพมากขึ้นจากโปรเซสเซอร์ที่ไม่มีสถาปัตยกรรมไปป์ไลน์เพียงครึ่งเดียวของประสิทธิภาพตามทฤษฎี ดังนั้นการลดการหน่วงเวลาที่เกิดขึ้นสำหรับปัญหาการควบคุมจึงเป็นเรื่องสำคัญอย่างยิ่ง จำนวนไซเคิลที่สูญเสียไปสำหรับการหน่วงเวลาคำสั่งทางแยกสามารถลดลงได้ภายใต้เงื่อนไข 2 ประการคือ

1. ต้องรู้ว่าคำสั่งทางแยกที่ถูกนำเข้ามาประมวลผลนั้นมีพฤติกรรม taken หรือ not taken ตั้งแต่ตอนต้นของกระบวนการไปป์ไลน์

2. เมื่อรู้ว่าคำสั่งทางแยกที่ถูกนำเข้ามาประมวลผลนั้น Taken ต้องทำการคำนวณค่าตำแหน่งเป้าหมายตั้งแต่ตอนต้นของกระบวนการไปป์ไลน์

สำหรับโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์นั้นคำสั่งทางแยกต้องทดสอบริจิสเตอร์ว่าเป็นศูนย์ เมื่อพิจารณาสเตจของไปป์ไลน์คำสั่ง ถ้าย้ายการทดสอบศูนย์ (Zero test) ไปทำในสเตจ ID แล้วจะทำให้รู้ว่าคำสั่งทางแยกนั้นมีพฤติกรรม Taken หรือ Not taken ตั้งแต่ในตอนท้ายของสเตจ ID การคำนวณตำแหน่งเป้าหมายของคำสั่งทางแยกก็จะสามารถกระทำได้ในสเตจ ID เช่นกัน โดยที่จะต้องมีหน่วยการคำนวณหรือแอดเดอร์ (Adder) เพิ่มขึ้นมาเนื่องจากเอแอลยูไม่สามารถทำงานได้ในสเตจนี้ เอแอลยูจะสามารถทำการคำนวณได้เมื่อสเตจนั้นเป็นสเตจ EX สำหรับการย้ายการทดสอบศูนย์มายังสเตจ ID ทำให้โครงสร้างดาต้าพาธของโปรเซสเซอร์เปลี่ยนไป สามารถแสดงได้ดังรูปที่ 3.10



รูปที่ 3.10 ดาต้าพาธของโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ที่ปรับปรุง [3]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

พิจารณาจากดาต้าพาธที่ได้ปรับปรุงแล้วจากรูปที่ 3.10 จะพบว่า การหน่วงเวลายังคงมีอยู่ และมีค่าเป็น 1 ไซเคิลเนื่องจากคำสั่งที่ตามหลังคำสั่งทางแยกได้ถูกเฟ็ตช์เข้ามาแล้วและต้องถูกยกเลิกไปเมื่อโปรเซสเซอร์จะต้องประมวลผลคำสั่งทางแยก ดังรูปที่ 3.11

Branch	IF	ID	EX	MEM	WB		
Branch successor		IF (Stall)	IF	ID	EX	MEM	WB

รูปที่ 3.11 การหน่วงเวลา 1 ไซเคิลเมื่อต้องยกเลิกคำสั่งที่ตามหลังคำสั่งทางแยก

3.3.2 การแก้ปัญหาควบคุมด้วยการทำนายผลลัพธ์ของคำสั่งทางแยกล่วงหน้า

การแก้ปัญหาเพื่อลดเวลาที่สูญเสียไปเนื่องจากการหน่วงเวลาโดยการทำนายผลลัพธ์ของคำสั่งทางแยกไว้ล่วงหน้า (Branch prediction) แบ่งเป็น 2 แบบ คือการทำนายทางแยกแบบสแตติก (Static branch prediction) และไดนามิก (Dynamic branch prediction) โดยการทำนายคำสั่งทางแยกแบบสแตติกจะกำหนดพฤติกรรมของคำสั่งทางแยกไว้ก่อนที่แน่นอนตายตัวตลอดการทำงานของโปรแกรม ในขณะที่การทำนายผลลัพธ์ของคำสั่งทางแยกแบบไดนามิกนั้นผลลัพธ์การทำนายของแต่ละคำสั่งทางแยกจะเปลี่ยนแปลงได้ขึ้นกับพฤติกรรมของคำสั่งทางแยกครั้งล่าสุดของตัวมันเอง หรือพฤติกรรมของคำสั่งทางแยกก่อนหน้าคำสั่งทางแยกนั้นขณะโปรแกรมนั้นๆกำลังทำงานทั้งนี้ขึ้นกับเทคนิคและวิธีการ

อย่างไรก็ตามสำหรับโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์เมื่อมีการปรับโครงสร้างดาต้าพาธดังรูปที่ 3.10 แล้ว ยังได้ใช้วิธีการทำนายผลลัพธ์ของคำสั่งทางแยกแบบสแตติกโดยทำนายผลลัพธ์ว่าคำสั่งทางแยกนั้น Not taken เสมอ เนื่องจากเมื่อพิจารณาจากดาต้าพาธ หลังจากการเฟ็ทช์คำสั่งทางแยกแล้ว สเตจ IF จะเฟ็ทช์คำสั่งที่ตามหลังคำสั่งทางแยกเข้ามา ถ้าคำสั่งทางแยกนั้นมีพฤติกรรม Taken ไปป์ไลน์ก็จะยกเลิกคำสั่งที่ตามหลังคำสั่งทางแยกนั้นทันที แต่ถ้าคำสั่งทางแยกนั้นมีพฤติกรรม not taken คำสั่งตามหลังคำสั่งทางแยกก็จะถูกประมวลผลต่อไปในจนครบทั้ง 5 ไปป์สเตจไม่มีการหน่วงเวลา ไม่มีการสูญเสีย 1 ไซเคิลดังกล่าว ดังนั้นถ้าคำสั่งทางแยกมีพฤติกรรม Taken คำสั่งที่ตามหลังคำสั่งทางแยกนั้นจะถูกแทนด้วยช่องว่าง การหน่วงเวลาที่เกิดขึ้นจะเกิดกับคำสั่งที่มีพฤติกรรม Taken เท่านั้น พิจารณาส่วนของโค้ดโปรแกรมต่อไปนี้

```
BNEZ  R1,Target
SW    R2,0(R3)
....  ......
....  ......
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Target : ADD R2,R1,R1

จากโค้ดโปรแกรมดังกล่าวสามารถเขียนไดอะแกรมเวลา (Time diagram) ได้ดังรูปที่ 3.12 ซึ่งแสดงการทำงานของไปป์ไลน์เมื่อคำสั่งทางแยกมีพฤติกรรม Taken และมีการหน่วงเวลาจำนวน 1 ไซเคิล

	IF	ID	EX	MEM	WB
TIME 1	BNEZ				
TIME 2	SW	BNEZ			
TIME 3	ADD	Stall	BNEZ		
TIME 4		ADD	Stall	BNEZ	
TIME 5			ADD	Stall	BNEZ
TIME 6				ADD	Stall
TIME 7					ADD

รูปที่ 3.12 การทำงานของไปป์ไลน์เมื่อคำสั่งทางแยกมีพฤติกรรม Taken

นอกจากนี้สำหรับโค้ดโปรแกรมข้างต้นเช่นเดียวกัน สามารถเขียนไดอะแกรมเวลาดังรูปที่ 3.13 ซึ่งแสดงการทำงานของไปป์ไลน์เมื่อคำสั่งทางแยกมีพฤติกรรม Not taken และไม่เกิดการหน่วงเวลาเลยเนื่องจากการทำนายผลลัพธ์ถูกต้อง

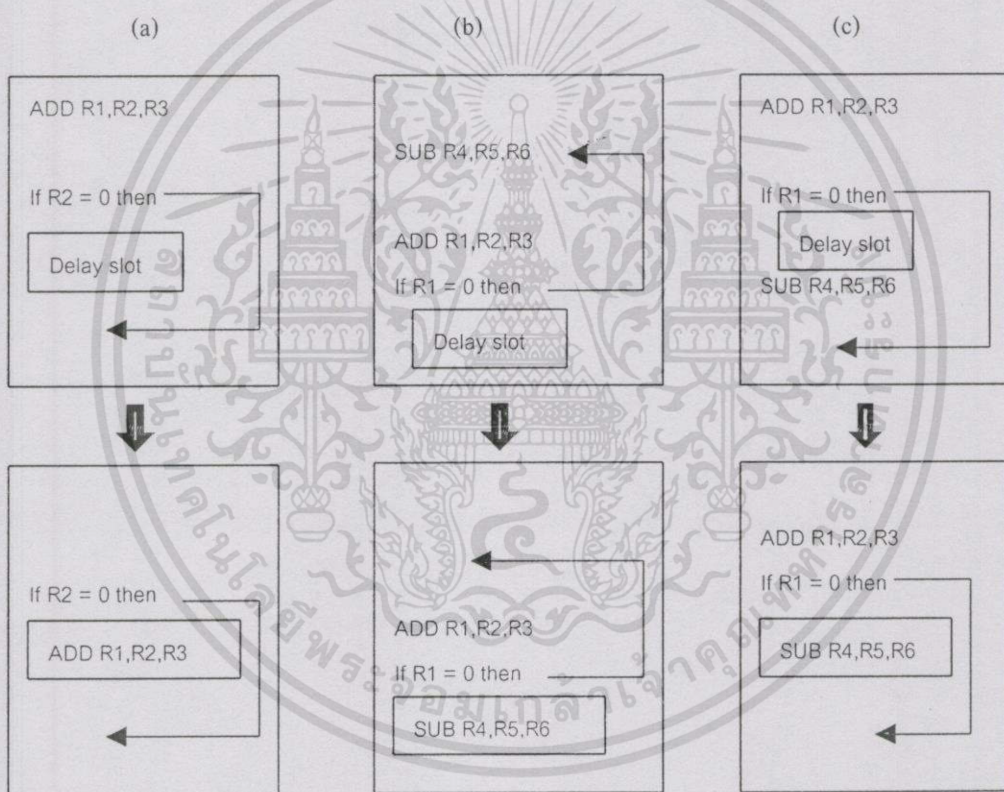
	IF	ID	EX	MEM	WB
TIME 1	BNEZ				
TIME 2	SW	BNEZ			
TIME 3		SW	BNEZ		
TIME 4			SW	BNEZ	
TIME 5				SW	BNEZ
TIME 6					SW

รูปที่ 3.13 การทำงานของไปป์ไลน์เมื่อคำสั่งทางแยกมีพฤติกรรม Not taken

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3.3 การแก้ปัญหาควบคุมด้วยหลักการดีเลย์บรานช์ (Delayed branch)

นอกจากการทำนายผลลัพธ์ของคำสั่งทางแยกแบบสแตติกที่โปรเซสเซอร์เดอลุกซ์ได้ใช้ในการแก้ปัญหาแล้ว ยังได้ใช้หลักการของดีเลย์บรานช์ (Delayed branch) [1] วิธีนี้เป็นวิธีทางซอฟต์แวร์โดยใช้คอมไพเลอร์จัดเรียงคำสั่ง มีหลักการคือ ให้คอมไพเลอร์ตรวจสอบว่าเมื่อมีคำสั่งทางแยก คอมไพเลอร์ต้องจัดการเรียงลำดับคำสั่งการทำงานทั้งหมดนั้นเสียใหม่ โดยระหว่างที่คำสั่งทางแยกกำลังตรวจสอบเงื่อนไขเพื่อไปทำงานตามตำแหน่งที่ถูกต้องนั้น คอมไพเลอร์จะต้องจัดการนำคำสั่งที่มีการประมวลผลสั้นๆ และเป็นคำสั่งอิสระ (Independent instruction) เข้ามาประมวลผลในช่างเวลาที่เกิดการหน่วงเวลาดังกล่าวเรียกว่าดีเลย์สล็อท (Branch delay slot) ทำให้กระบวนการของไปป์ไลน์ดำเนินต่อไปไม่หยุดชะงัก หลักการของดีเลย์บรานช์ แสดงดังรูปที่ 3.14



รูปที่ 3.14 การนำคำสั่งอิสระแทรกเข้าไปทำงานในดีเลย์สล็อท [1]

รูปที่ 3.14 (a) คอมไพเลอร์จัดการนำคำสั่งอิสระที่อยู่ก่อนคำสั่งทางแยกมาแทรกในดีเลย์สล็อท ส่วนรูปที่ 3.14 (b) คอมไพเลอร์จัดการนำคำสั่งอิสระที่เป็นเป้าหมายของคำสั่งทางแยกซึ่งต้องทำอยู่แล้วมาแทรกในดีเลย์สล็อท และรูปที่ 3.14 (c) คอมไพเลอร์จัดการนำคำสั่งอิสระที่อยู่นอกเงื่อนไขคำสั่งทางแยกมาแทรกในดีเลย์สล็อท

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ข้อจำกัดของดีเลย์บรานช์ คือความยากในการหาคำสั่งอิสระที่เหมาะสมมาประมวลผลในช่วงการหน่วงเวลา โดยเฉพาะอย่างยิ่งถ้าคำสั่งทางแยกนั้นจะต้องไปทำงานที่เป็นวงรอบซับซ้อนหรือมีการทำไปป์ไลน์ที่หลายชั้นขึ้น (Deeper pipeline) นอกจากนี้ยังต้องมีคอมไพเลอร์ที่มีความสามารถสูงในการช่วยจัดการเรียงลำดับคำสั่งอีกด้วย

3.3.4 การแก้ปัญหาควบคุมด้วยการทำไปป์ไลน์สเคจดูลลิงและลูปอันโรลลิง (Pipeline scheduling and loop unrolling) [2]

เป็นวิธีการที่ให้คอมไพเลอร์ช่วยจัดการ การประมวลผลคำสั่งในไปป์ไลน์โปรเซสเซอร์ ถ้าโปรแกรมที่นำมาประมวลผลมีคำสั่งทางแยกมากเท่าใดโอกาสสูญเสียไซเคิลการทำงานเนื่องจากคำสั่งทางแยกจะมากขึ้น การสูญเสียไซเคิลจำนวนมากหรือน้อยขึ้นกับประสิทธิภาพการทำนายผลลัพธ์ของคำสั่งทางแยก ถ้าความถูกต้องสูงการสูญเสียไซเคิลจะน้อย ไปป์ไลน์คำสั่งจะมีประสิทธิภาพสูง อย่างไรก็ตามเมื่อจำเป็นต้องประมวลผลคำสั่งทางแยกจำนวนมาก สามารถทำให้โปรแกรมที่ทำงานประมวลผลคำสั่งทางแยกลดลงได้ จากงานวิจัย [4] พบว่าในระหว่างลูปของคำสั่งทางแยกจะมีจำนวนคำสั่งอื่นๆที่ไม่ใช่คำสั่งทางแยกประมาณ 6-7 คำสั่ง นั่นคือทุกๆ 6-7 คำสั่งอาจจะมีการเปลี่ยนแปลงค่าโปรแกรมเคาน์เตอร์หรือมีการกระโดดไปทำงานของคำสั่งทางแยก ไปป์ไลน์คำสั่งจะมีประสิทธิภาพมากถ้าทุกไปป์สเตจมีการทำงานเต็มตลอดเวลา ดังนั้นต้องทำให้ไมโครโปรเซสเซอร์ประมวลผลคำสั่งอื่นๆที่ไม่ใช่คำสั่งทางแยกให้มากที่สุด ทางหนึ่งที่ทำได้ในการเพิ่มจำนวนคำสั่งที่อยู่ระหว่างลูปของคำสั่งทางแยกซึ่งเรียกว่าการทำลูปอันโรล (Loop unrolling) วิธีนี้กระทำโดยคอมไพเลอร์ โดยการเพิ่มคำสั่งที่สามารถประมวลผลระหว่างคำสั่งทางแยกในขั้นตอนการคอมไพล์โปรแกรม อย่างไรก็ตามเพื่อให้มีประสิทธิภาพสูงสุดคอมไพเลอร์จะต้องมีความสามารถในการจัดเรียงคำสั่งด้วยเพื่อลดปัญหาข้อมูลด้วยซึ่งเรียกว่าไปป์ไลน์สเคจดูลลิง (Pipeline scheduling)

ตัวอย่างส่วนของโปรแกรมต่อไปนี้เป็นลูปของคำสั่งที่ทำการบวกค่าคงที่เข้ากับค่าในหน่วยความจำ แล้วนำผลลัพธ์เก็บในหน่วยความจำ

```
for (I = 1; I <= 1000; I ++)
    x[I] = x[I] + s;
```

ถ้าแปลงโค้ดข้างต้นให้เป็นโค้ดภาษาแอสเซมบลีจะได้เป็น

```
Loop:  LD    F0,0(R1)
       ADD   F4,F0,F2
       SD    0(R1),F4
       SUBI  R1,R1,#8
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

BNEZ R1,LOOP

ก่อนถึงขั้นตอนการทำลูปอันโรลต้องต้องนำคำสั่งมาทำสเคจดูลก่อน โดยพิจารณาเปรียบเทียบจำนวนคล็อกที่ใช้ประมวลผลระหว่างการทำสเคจดูลและเมื่อไม่ทำสเคจดูล กำหนดเงื่อนไขว่าไม่เกิดปัญหาโครงสร้างฮาร์ดแวร์ขึ้นในไปป์ไลน์และกำหนดจำนวนไซเคิลที่สูญเสียสำหรับปัญหาข้อมูลดังนี้

Instruction producing Result	Instruction using Result	Latency in clock cycle
FP ALU OP	Another FP ALU OP	3
FP ALU OP	Store double	2
Load double	FP ALU OP	1
Load double	Store double	0
Integer ALU	Branch	1

1) กรณีไม่มีการทำสเคจดูล

พิจารณาชุดคำสั่งดังนี้

```
LOOP:  LD      F0, 0(R1)
       STALL
       ADDD    F4, F0, F2
       STALL
       STALL
       SD      0(R1), F4
       SUBI    R1, R1, #8
       STALL
       BNEZ    R1, LOOP
       STALL
```

ชุดคำสั่งใช้ไซเคิลจำนวน 10 ไซเคิลในการประมวลผล มีไซเคิลที่หน่วงเวลาคือ 1 ไซเคิลสำหรับคำสั่ง LD 2 ไซเคิลสำหรับคำสั่ง ADDD 1 ไซเคิลสำหรับคำสั่ง SUBI และอีก 1 ไซเคิลสำหรับดีเลย์บรานช์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2) กรณีมีการทำสเคจดูล

จากชุดคำสั่งกรณีไม่มีการทำสเคจดูลในข้อ 1) เมื่อมีการทำสเคจดูลจะได้เป็น

```
LOOP: LD      F0, 0(R1)
      SUBI    R1, R1, #8
      ADDD    F4, F0, F2
      STALL
      BNEZ    R1, LOOP
      SD      0(R1), F4
```

กรณีมีการทำสเคจดูลนี้คอมไพเลอร์ทำการจัดเรียงคำสั่งให้เหมาะสมโดยลดไซเคิลการหน่วงเวลาให้น้อยลงเหลือเพียงไซเคิลเดียว หลังการทำสเคจดูลแล้วชุดคำสั่งดังกล่าวใช้เวลาในการประมวลผล 6 ไซเคิลเท่านั้น

พิจารณาตัวอย่างดังกล่าวพบว่าใน 1 ลูปนั้นคำสั่งใช้เวลาทำงาน 6 ไซเคิล แต่งานจริงมีเพียง 3 โอเปอเรชันเท่านั้นคือ LOAD ADD และ STORE ซึ่งใช้เวลาเพียง 3 ไซเคิลเท่านั้น อีก 3 ไซเคิลจะเป็นลูปโอเวอร์เฮดคือคำสั่ง SUBI BNEZ และ STALL การกำจัด 3 ไซเคิลลูปโอเวอร์เฮดจะต้องเพิ่มจำนวนคำสั่งในลูปให้มากขึ้น วิธีการเพิ่มจำนวนคำสั่งในลูปคือการทำลูปอันโรล ทำได้โดยนำโค้ดโปรแกรมในแต่ละลูปมาเขียนเรียงลำดับกันไปทั้งหมดเพื่อกำจัดคำสั่งทางแยก หรือเรียงคำสั่งต่อกันไปจำนวนหนึ่งเพื่อลดจำนวนรอบในการวนลูป

การทำลูปอันโรลช่วยให้การทำสเคจดูลมีประสิทธิภาพมากขึ้นเพราะช่วยลดจำนวนคำสั่งทางแยก หรืออีกนัยหนึ่งเป็นการเพิ่มจำนวนคำสั่งที่ไม่ใช่คำสั่งทางแยกในลูปซึ่งมีผลให้สามารถนำคำสั่งเหล่านั้นมาทำสเคจดูลได้ เมื่อทำลูปอันโรลโดยการนำคำสั่งมาเรียงต่อกัน คำสั่งเดียวกันแต่อยู่ต่างลูปจะใช้ริจิสเตอร์ตัวเดียวกัน ทำให้มีจำนวนริจิสเตอร์ที่ต้องใช้งานมากขึ้น

ชุดคำสั่งต่อไปนี้ได้จากตัวอย่างข้างต้นนำมาทำลูปอันโรล พิจารณาในลูปจะมีคำสั่งแบ่งเป็น 4 ชุด โดยที่ริจิสเตอร์ R1 เป็นตำแหน่งของอะเรย์ตัวแรกและเพิ่มค่าครั้งละ 32 ที่ท้ายลูป ซึ่งจะทำให้ลดการประมวลผลคำสั่ง SUBI และ BNEZ ชุดละ 3 ไซเคิล ดังนี้

```
LOOP: LD      F0, 0(R1)
      ADDD    F4, F0, F2
      SD      0(R1), F4          ; drop SUBI& BNEZ
      LD      F6, -8(R1)
      ADDD    F8, F6, F2
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
SD      -8(R1), F8          ; drop SUBI& BNEZ
LD      F10, -16(R1)
ADDD    F12, F10, F2
SD      -16(R1), F12        ; drop SUBI& BNEZ
LD      F14, -24(R1)
ADDD    F16, F14, F2
SD      -24(R1), F16
SUBI    R1, R1, #32
BNEZ    R1, LOOP
```

หลังการทำลูปอันโรลแล้วถ้าไม่ทำสเคจดูลชุดคำสั่งข้างต้นจะใช้เวลาในการประมวลผล 28 ไซเคิล หรือคิดเป็น 7 ไซเคิลต่อลูป ซึ่งจะใช้เวลามากกว่าการไม่ทำลูปอันโรล ดังนั้นจะต้องมีการทำสเคจดูลเพื่อลดจำนวนไซเคิลต่อลูปลงดังนี้

```
LOOP: LD      F0, 0(R1)
      LD      F6, -8(R1)
      LD      F10, -16(R1)
      LD      F14, -24(R1)
      ADDD    F4, F0, F2
      ADDD    F8, F6, F2
      ADDD    F12, F10, F2
      ADDD    F16, F14, F2
      SD      0(R1), F4
      SD      -8(R1), F8
      SUBI    R1, R1, #32
      SD      -16(R1), F12
      BNEZ    R1, LOOP
      SD      8(R1), F16
```

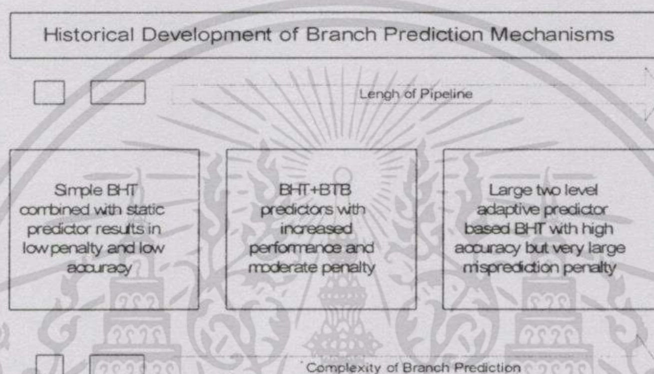
คำสั่งทั้งหมดมี 14 คำสั่งใช้เวลา 14 คล็อกไซเคิล แสดงว่าจำนวนไซเคิลต่อลูปเป็น 14/4 = 3.5 ซึ่งเป็นผลลัพธ์ที่ดีขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การทำลูปอันโรลเป็นการเพิ่มประสิทธิภาพในการประมวลผลคำสั่งในลูป โดยกำจัดลูปโอเวอร์เฮด แต่จะทำให้โค้ดโปรแกรมมีจำนวนเพิ่มขึ้นซึ่งอาจจะส่งผลให้เกิดปัญหาการขึ้นต่อกันของข้อมูลและเกิดการหน่วงเวลา การทำสเคจดูลควบคู่ไปด้วยจะทำให้ลดการหน่วงเวลานั้นลง

3.3.5 การแก้ปัญหาการประมวลผลคำสั่งทางแยกในไมโครโปรเซสเซอร์อื่นๆ

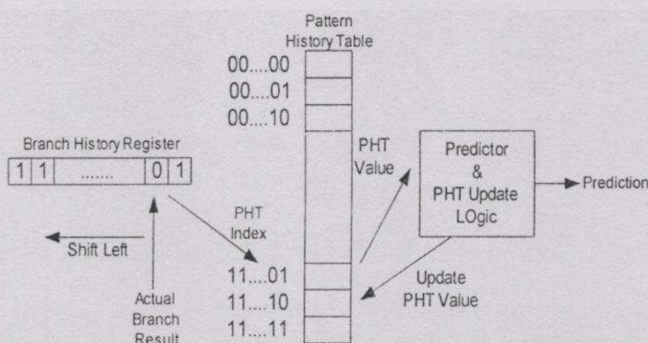
ไมโครโปรเซสเซอร์อื่นๆที่มีการใช้งานอยู่ในปัจจุบันได้ใช้หลักการทำนายผลลัพธ์ของคำสั่งทางแยกไว้ล่วงหน้าเพื่อแก้ปัญหาของไปป์ไลน์เมื่อต้องประมวลผลคำสั่งทางแยกเช่นเดียวกัน โดยมีการพัฒนากลไกการทำนายผลลัพธ์ให้แม่นยำมากขึ้นตามความยาวของไปป์ไลน์ซึ่งจะทำให้วิธีการทำนายซับซ้อนขึ้นดังรูป 3.15 ซึ่งเป็นประวัติการพัฒนาการทำนายผลลัพธ์คำสั่งทางแยก



รูปที่ 3.15 ประวัติการพัฒนากลไกการทำนายผลลัพธ์คำสั่งทางแยกของโปรเซสเซอร์

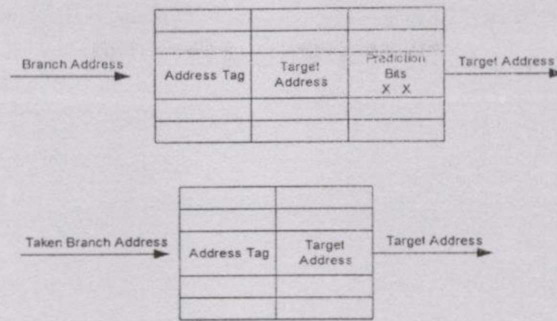
โปรเซสเซอร์ที่มีอยู่ในปัจจุบันได้ใช้การทำนายผลลัพธ์ของคำสั่งทางแยกเช่นกัน ดังนี้

Intel P6 ใช้การทำนายแบบการปรับปรุง 2 ระดับ (2-level adaptive) ร่วมกับบีทีบี (BTB: Branch Target Buffer) [5] การทำนายแบบการปรับปรุง 2 ระดับนอกจากจะใช้ประวัติของตัวมันเองในอดีตแล้วยังใช้ประวัติของคำสั่งทางแยกอื่นที่ประมวลผลก่อนคำสั่งทางแยกที่กำลังพิจารณาด้วย ซึ่งจะทำให้ความถูกต้องในการทำนายสูงกว่า 1 ระดับ โครงสร้างการทำนายแบบปรับปรุง 2 ระดับเป็นดังรูปที่ 3.16 และรูปที่ 3.17 เป็นหน่วยความจำบีทีบี



รูปที่ 3.16 การทำนายแบบปรับปรุง 2 ระดับ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.17 หน่วยความจำบีทีบีของโปรเซสเซอร์ Intel P6

Branch History Register ของ Intel P6 จะมีขนาด 4 บิทแต่ละบิทเป็นผลลัพธ์ของคำสั่งทางแยก โดยผลลัพธ์ของคำสั่งทางแยกจะนำเข้ามาใส่ทางขวาของ Branch History Register บิทที่เหลือจะถูกเลื่อนไปทางซ้าย (ผลลัพธ์ของคำสั่งทางแยก 1=Taken 0=Not taken) ค่าใน Branch History Register จะเป็นคีย์ในการเลือกบิทการทำนายจาก Pattern History Table ซึ่งแต่ละแถวประกอบด้วย 2 บิท ค่าของ Pattern History Table จะถูกนำไปประมวลผลด้วยสเตจแมชชีนที่ Predictor &PHT Update Logic เพื่อหาผลลัพธ์การทำนายและค่าที่ต้องการปรับปรุงลง Pattern History Table เมื่อได้ผลการทำนายแล้วจะนำค่านี้ไปใช้ควบคู่กับบีทีบี โดยบีทีบีมีขนาด 512 เอนทรี (Entry) เป็นหน่วยความจำแคชแบบ 4-way set associative ความถูกต้องในการทำนายนี้มีค่า 90% เมื่อทดสอบกับ specint 92 เมื่อทำนายผิดจะสูญเสียไซเคิลโดยเฉลี่ย 15 ไซเคิล โดยเมื่อไม่พบข้อมูลในบีทีบีจะสูญเสียไซเคิล 7 ไซเคิล

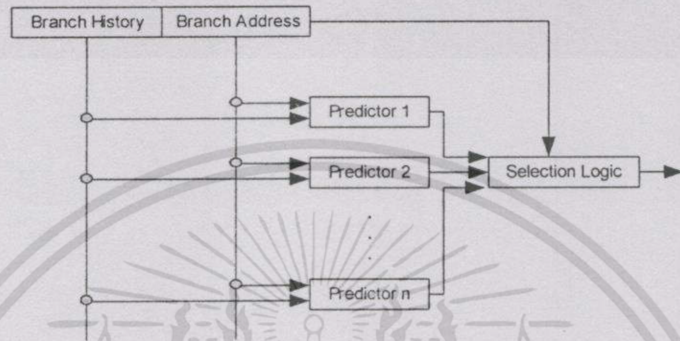
Motorola PowerPC 750 ใช้การทำนายผลลัพธ์แบบสแตติกโดยทำนายผลลัพธ์ล่วงหน้าว่า Not Taken แต่สามารถเลือกให้มีการทำนายแบบไดนามิกก็ได้ ในการทำนายแบบไดนามิกจะใช้บีทีไอซี (BTIC: Branch Target Instruction Cache) [6] ขนาด 64 เอนทรี (16 set, 4-way set associative) ซึ่งจะใช้ควบคู่กับหน่วยความจำแคชปกติ ดังนั้นถ้าพบข้อมูลในบีทีไอซี คำสั่งนั้นจะเป็นคำสั่งทางแยก สามารถนำตำแหน่งเป้าหมายไปใช้ได้ทันที การทำนายผลลัพธ์แบบไดนามิกของ Motorola PowerPC 750 ยังใช้บีทีไอซีร่วมกับบีเอชที (BHT: Branch History Table) [5] โดยบีเอชทีมีขนาด 512 เอนทรี แต่ละเอนทรีมีบิทการทำนายขนาด 2 บิท การทำนายผลลัพธ์สามารถทำนายได้ 2 คำสั่งทางแยกพร้อมๆ กัน

HP 8000 ใช้การทำนายผลลัพธ์แบบสแตติกโดยทำนายผลลัพธ์ล่วงหน้าว่า Taken แต่สามารถเลือกให้มีการทำนายแบบไดนามิกก็ได้ การทำนายผลลัพธ์แบบสแตติกสามารถแก้ปัญหาคำสั่งทางแยกที่มีเงื่อนไขและไม่มีเงื่อนไขได้ทุกกรณี ส่วนการทำนายผลลัพธ์แบบไดนามิกใช้บีทีเอซี (BTAC: Branch Target Address Cache) [6] ขนาด 256 เอนทรีซึ่งเป็นแคชแบบ Fully-associative บีทีเอซีประกอบด้วยตำแหน่งเป้าหมายของคำสั่งทางแยก ถ้าพบข้อมูลในบีทีเอซี แสดงว่าคำสั่งนั้นเป็นคำสั่งทางแยก สามารถนำค่าตำแหน่งเป้าหมายจากบีทีเอซีไปใช้ได้เลย ใช้บีเอชทีขนาด 256

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

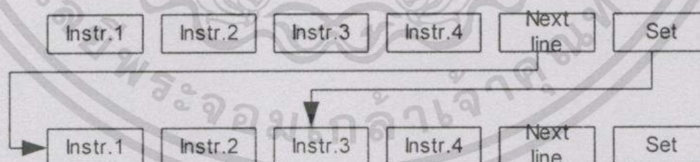
เอนทรีใช้บิทการทำนายขนาด 3 บิท การทำนายถูกต้องประมาณ 80 % และทำนายคำสั่งทางแยกได้ครั้งละ 1 คำสั่ง

DEC Alpha ใช้การทำนายแบบผสมระหว่างบีเอชทีและการปรับปรุงแบบ 2 ระดับ ประกอบด้วยหน่วยการทำนาย (Predictor) หลายหน่วยแต่จะถูกเรียกใช้ครั้งละ 1 หน่วยขึ้นกับตำแหน่งของคำสั่งทางแยกจากหน่วยความจำคำสั่ง (Instruction cache) การทำนายแบบผสมแสดงดังรูปที่ 3.18



รูปที่ 3.18 การทำนายผลลัพธ์ของคำสั่งทางแยกใน โปรเซสเซอร์ DEC Alpha

จากรูป Branch History จะเก็บประวัติของคำสั่งทางแยกก่อนหน้าคำสั่งทางแยกปัจจุบัน Branch History มีขนาด 3 บิท เมื่อนำประวัติและ Branch History เป็นคีย์สำหรับเลือกหน่วยการทำนายจำนวนหนึ่ง หลังจากนั้นหน่วยการทำนายจะถูกเลือกให้เหลือเพียง 1 หน่วย โครงสร้างหน่วยความจำคำสั่งของ DEC Alpha เป็นดังรูปที่ 3.19

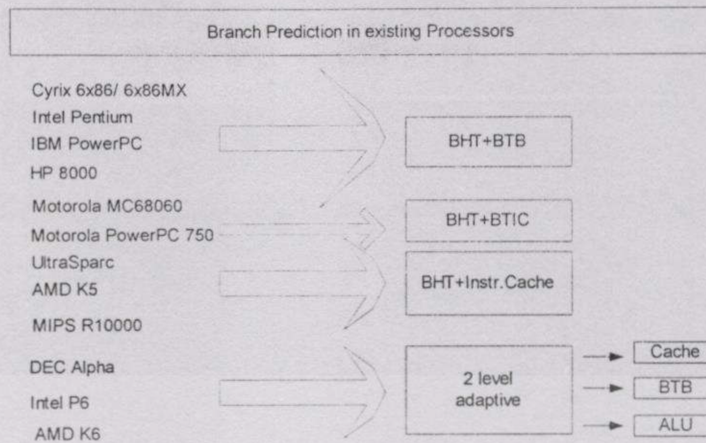


รูปที่ 3.19 โครงสร้างหน่วยความจำคำสั่งของโปรเซสเซอร์ DEC Alpha

AMD K6 ใช้การทำนายแบบปรับปรุง 2 ระดับร่วมกับบีทีซี (BTC: Branch Target Cache) [5] โดยการทำนายแบบปรับปรุง 2 ระดับถูกอิมพลีเมนท์แบบบีเอชทีขนาด 8192 เอนทรี บีทีซีจะเก็บตำแหน่งเป้าหมายของคำสั่งทางแยก ความถูกต้องของการทำนายมากกว่า 95 % เมื่อทดสอบกับ Specint 92

โปรเซสเซอร์ที่มีอยู่ในปัจจุบันได้ใช้การทำนายผลลัพธ์ของคำสั่งทางแยกสรุปได้ดังรูป 3.20

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.20 การทำนายผลลัพธ์ของคำสั่งทางแยกในโปรเซสเซอร์ที่มีอยู่ในปัจจุบัน

จากรูปที่ 3.20 พบว่าปัจจุบันโปรเซสเซอร์ส่วนใหญ่จะใช้การทำนายผลลัพธ์แบบไดนามิก ซึ่งจะต้องมีการจัดเก็บผลลัพธ์การทำนายไว้ในหน่วยความจำบีทีบี (BTB: Branch Target Buffer) หรือหน่วยความจำบีเอชที (BHT: Branch History Table) โดยข้อมูลที่ถูกจัดเก็บไว้จะถูกนำไปใช้ในการคาดหมายผลลัพธ์ของคำสั่งทางแยกปัจจุบันที่กำลังถูกประมวลผล ตลอดจนมีการปรับปรุงข้อมูลในหน่วยความจำเสมอเพื่อให้การนำไปใช้คาดหมายหรือทำนายในครั้งต่อไปถูกต้องแม่นยำยิ่งขึ้น รายละเอียดของเรื่องนี้จะกล่าวถึงอย่างละเอียดอีกครั้งในบทที่ 4 ซึ่งเป็นเนื้อหาของวิทยานิพนธ์

3.4 การวัดประสิทธิภาพของไปป์ไลน์คำสั่ง

เมื่อเกิดปัญหาในไปป์ไลน์คำสั่งแล้วไปป์ไลน์เกิดการหน่วงเวลาเพื่อให้การทำงานของคำสั่งใดๆ ทำงานได้เสร็จเรียบร้อยก่อนนั้น การหน่วงเวลาทำให้ประสิทธิภาพของไมโครโปรเซสเซอร์ลดลงไปจากอุดมคติ การวัดประสิทธิภาพของไปป์ไลน์สามรถคำนวณได้จากสมการต่างๆ [2] โดยมีการกำหนดความหมายของตัวแปรต่างๆ ในสมการ ดังนี้

ตัวแปรที่ใช้	ความหมาย
Speedup	ความเร็วที่เพิ่มขึ้น
Average instruction time pipelined	เวลาเฉลี่ยที่แต่ละคำสั่งใช้เมื่อเป็นไปป์ไลน์
Average instruction time unpipelined	เวลาเฉลี่ยที่แต่ละคำสั่งใช้เมื่อไม่เป็นไปป์ไลน์
$CPI_{pipelined}$	จำนวนไซเคิลที่ใช้ทำงานต่อหนึ่งคำสั่งในไปป์ไลน์
$CPI_{unpipelined}$	จำนวนไซเคิลที่ใช้ทำงานต่อหนึ่งคำสั่งเมื่อไม่เป็นไปป์ไลน์
Clock cycle $time_{pipelined}$	ช่วงเวลาของ 1 ไซเคิลที่ทำงานในไปป์ไลน์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Clock cycle $_{timeunpipelined}$	ช่วงเวลาของ 1 ไซเคิลที่ทำงานเมื่อไม่ปีนไปป์ไลน์
Ideal CPI	จำนวนไซเคิลที่ใช้ทำงานต่อหนึ่งคำสั่งตามอุดมคติ
Pipeline stall clock cycles per instruction	จำนวนไซเคิลที่เสียไปเมื่อมีการหน่วงเวลาในไปป์ไลน์
Pipeline depth	ความลึกของไปป์ไลน์

เมื่อเปรียบเทียบอัตราเร็วที่เพิ่มขึ้นระหว่างโปรเซสเซอร์ที่มีการทำไปป์ไลน์คำสั่งกับโปรเซสเซอร์ที่ไม่มีการทำไปป์ไลน์คำสั่งจะได้ดัง (3.1)

$$Speedup\ \ from\ \ pipelining = \frac{Average\ \ instruction\ \ time\ unpipelined}{Average\ instruction\ time\ \ pipelined} \qquad (3.1)$$

โดยที่ Average instruction time = CPI x Clock cycle time

ดังนั้น อัตราเร็วที่เพิ่มขึ้นเนื่องจากการทำไปป์ไลน์ สามารถหาได้จาก (3.2)

$$\text{Speedup from pipelining} = \frac{\text{CPI}_{\text{unpipelined}} \times \text{Clock cycle time}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}} \times \text{Clock cycle time}_{\text{pipelined}}} \qquad (3.2)$$

เนื่องจาก Ideal CPI ของไปป์ไลน์คำสั่งมีค่าเป็น 1 ดังนั้นถ้ามีปัญหาเกิดขึ้นในไปป์ไลน์คำสั่ง ไปป์ไลน์เกิดการหน่วงเวลา ค่าของ $\text{CPI}_{\text{pipelined}}$ หาได้จาก (3.3)

$$\begin{aligned}\text{CPI}_{\text{pipelined}} &= \text{Ideal CPI} + \text{Pipeline stall clock cycles per instruction} \\ &= 1 + \text{Pipeline stall clock cycles per instruction}\end{aligned} \qquad (3.3)$$

โดยไม่สนใจไซเคิลที่เป็นโอเวอร์เฮดของไปป์ไลน์และกำหนดว่าทุกไปป์สเตจสมดุลแล้ว สามารถเปรียบเทียบประสิทธิภาพระหว่างโปรเซสเซอร์ได้ดัง (3.4)

$$\text{Speedup} = \frac{\text{CPI}_{\text{unpipelined}}}{1 + \text{Pipeline stall clock cycles per instruction}} \qquad (3.4)$$

ถ้าทุกคำสั่งใช้จำนวนไซเคิลการทำงานเท่ากัน ซึ่งจะเท่ากับจำนวนของไปป์สเตจ (หรือเท่ากับ Pipeline depth) แล้ว CPI $_{unpipelined}$ จะเท่ากับ Pipeline depth ดังนั้นจะได้ดัง (3.5)

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall clock cycles per instruction}} \qquad (3.5)$$

ถ้าไม่มีการหน่วงเวลาของไปป์ไลน์ แสดงว่าการทำไปป์ไลน์คำสั่งทำให้ประสิทธิภาพของโปรเซสเซอร์เป็นไปตามอุดมคติคือเท่ากับความลึกของไปป์ไลน์หรือจำนวนไปป์สเตจ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากการแก้ปัญหาการหน่วงเวลาเนื่องจากคำสั่งทางแยก รูปที่ 3.21 แสดงให้เห็นประสิทธิภาพของวิธีการแก้ปัญหาแต่ละวิธี

Scheduling schemes	Branch penalty	Effective CPI	Pipeline speedup over Non-piped version	Pipelining speedup over stall pipe on branch schemes
Stall pipeline	3.0	1.42	3.5	1.00
Predict taken	1.0	1.14	4.4	1.26
Predict not taken	1.0	1.09	4.5	1.29
Delayed branch	0.5	1.07	4.6	1.31

รูปที่ 3.21 ประสิทธิภาพของการแก้ปัญหาของคำสั่งทางแยก 4 วิธี

จากรูปที่ 3.21 คำนวณจากประสิทธิภาพจาก (3.6)

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch Frequency} * \text{Branch Penalty}} \quad (3.6)$$

โดยที่ กำหนด CPI = 1 และ Stalls = Branch frequency * Branch Penalty

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4

การแก้ปัญหาการควบคุมในไปป์ไลน์คำสั่ง

สำหรับกลุ่มคำสั่งทางแยก ซึ่งประกอบด้วย 6 คำสั่งคือ คำสั่ง J JR JAL JALR BEQZ และ BNEZ ซึ่งได้กล่าวถึงในบทที่ 2 ตามรูปที่ 2.4 ดังนี้

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

ไมโครโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์มีวิธีการแก้ปัญหาควบคุมโดยการปรับปรุงดาต้าพาธด้วยการย้ายการตรวจสอบคำสั่งทางแยก (Branch instruction) มาไว้ที่สเตจ ID ปรากฏว่าการหน่วงเวลายังคงมีอยู่และมีการสูญเสียไซเคิลการทำงาน 1 ไซเคิลเมื่อต้องยกเลิกคำสั่งที่ตามหลังคำสั่งทางแยก นอกจากนี้ยังได้ใช้วิธีการทำนายผลลัพธ์ของคำสั่งทางแยกว่าเป็น not taken เสมอ ถ้าผลลัพธ์ของคำสั่งทางแยกนั้นมีพฤติกรรมเป็น not taken ก็จะไม่มีการสูญเสียไซเคิลการทำงานเลย แต่เมื่อใดก็ตามที่การทำนายผิดพลาด หรือผลลัพธ์ของคำสั่งทางแยกนั้นมีพฤติกรรมเป็น taken ก็ยังมีการสูญเสียไซเคิลการทำงานเช่นเดิม นอกจากนี้การแก้ปัญหาด้วยการปรับปรุงดาต้าพาธดังกล่าวแม้ว่าจะทำให้มีการสูญเสียไซเคิลการทำงานเพียงไซเคิลเดียวก็ตาม ผลเสียที่ตามมาคือเกิดปัญหาข้อมูลและไม่สามารถรองรับคำสั่งอื่นๆที่อยู่ในกลุ่มของชุดคำสั่งทางแยกได้แก่ คำสั่ง J JR JAL และ JALR ดังนั้น การทดลองของงานวิจัยนี้จึงได้นำหลักการที่มีการแก้ปัญหาควบคุมด้วยวิธีการเดิมคือการทำนายผลลัพธ์ของคำสั่งทางแยกว่า Not taken การตรวจสอบคำสั่งทางแยกอยู่สเตจ EX และได้ทำการปรับปรุงดาต้าพาธบางส่วนให้การแก้ปัญหาควบคุมไม่มีผลทำให้เกิดปัญหาข้อมูล นอกจากนี้ยังครอบคลุมทุกคำสั่งที่อยู่ในกลุ่มของชุดคำสั่งทางแยก ซึ่งถือเป็นหลักการเดิมที่ถูกนำเสนอไว้โดย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กลุ่มผู้พัฒนาไมโครโปรเซสเซอร์เดอลุกซ์ [2] และนำเสนอหลักการแก้ปัญหาควบคุมโดยวิธีการปรับปรุงการทำนายทางแยก ดังรายละเอียดต่อไปนี้

4.1 การทำนายผลลัพธ์ของคำสั่งทางแยกแบบเดิม

จากหลักการแก้ปัญหาคำสั่งทางแยกเดิมที่มีการตรวจสอบผลลัพธ์จริงของชุดคำสั่งทางแยกให้กระทำที่สเตจ EX ดาต้าพาธของไมโครโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์เป็นดังรูปที่ 4.1 และจากดาต้าพาธนี้ใช้หลักการทำนายทางแยกของคำสั่งทางแยกแบบสแตติกโดยการทำนายว่าทางแยกนั้นมีพฤติกรรม not taken เสมอ (prediction as not taken) การแก้ปัญหาควบคุมไม่มีผลกระทบทำให้เกิดปัญหาข้อมูลตามมาแต่อย่างใด นอกจากนี้ยังครอบคลุมทุกคำสั่งทั้งหมดที่อยู่ในกลุ่มของชุดคำสั่งทางแยก จำนวนไซเคิลที่สูญเสียสำหรับการแก้ปัญหาด้วยวิธีนี้สรุปได้ดังตารางที่ 4.1

ตารางที่ 4.1 จำนวนไซเคิลที่สูญเสียสำหรับการแก้ปัญหาควบคุมด้วยหลักการเดิม

คำสั่ง	จำนวนไซเคิลที่สูญเสีย
J	2
JR	2
JAL	2
JALR	2
BEQZ (กรณีผลลัพธ์ของคำสั่งทางแยกเป็น taken)	2
BEQZ (กรณีผลลัพธ์ของคำสั่งทางแยกเป็น not taken)	0
BNEZ (กรณีผลลัพธ์ของคำสั่งทางแยกเป็น taken)	2
BNEZ (กรณีผลลัพธ์ของคำสั่งทางแยกเป็น not taken)	0

จากดาต้าพาธในรูปที่ 4.1 การตรวจสอบผลลัพธ์จริงของกลุ่มคำสั่งทางแยกเกิดขึ้นในสเตจ EX โดยมีหน่วยตรวจสอบทางแยก (Branch Prediction Unit) ซึ่งมีหลักการทำงานโดยพิจารณารูปที่ 4.2 ดังนี้

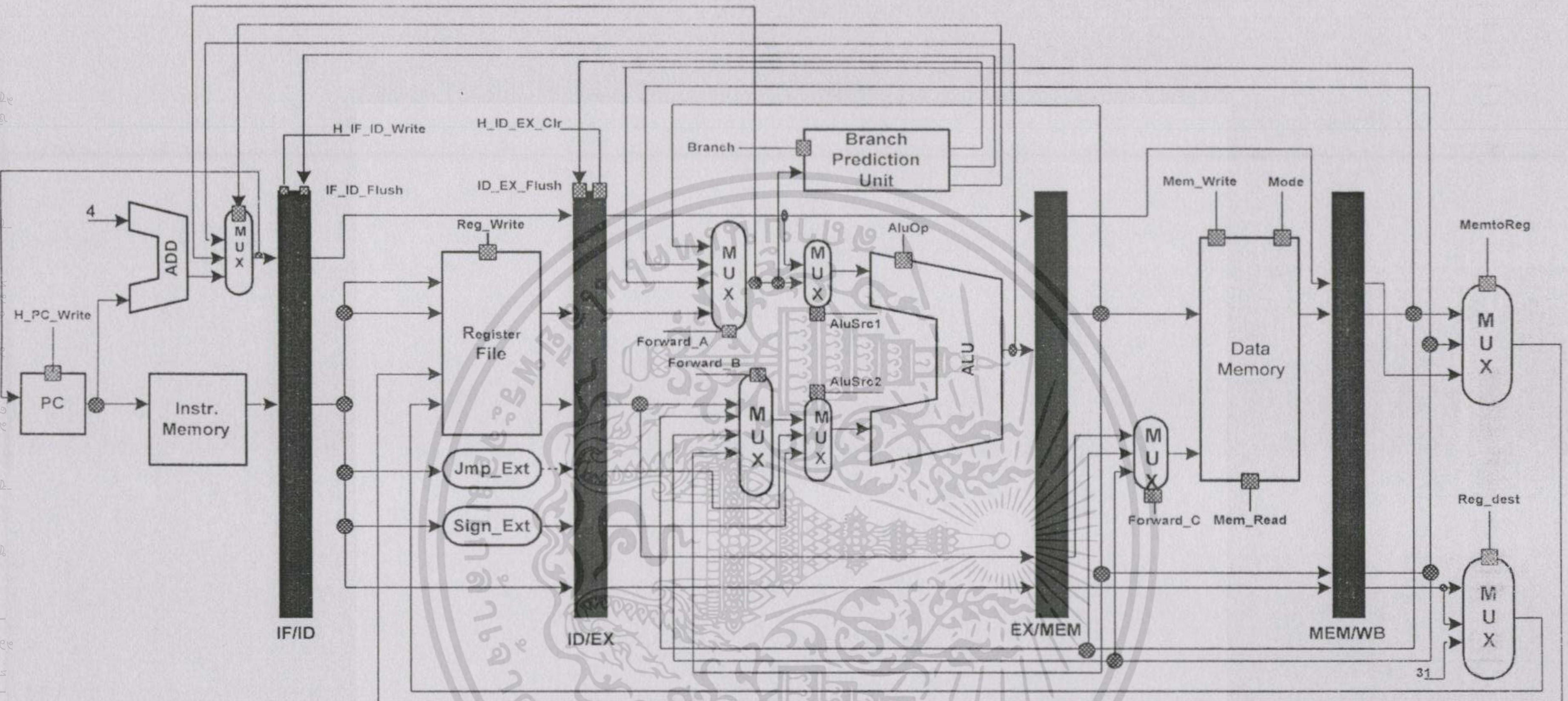
1. หน้าที่ของหน่วยตรวจสอบทางแยก

หน่วยตรวจสอบทางแยกเป็นหน่วยที่ทำหน้าที่ตรวจสอบการทำงานของคำสั่งในกลุ่มคำสั่งทางแยก (J, JAL, JR, JALR, BEQZ, BNEZ)

- คำสั่ง J, JR, JALR และ JAL เมื่อคำสั่งเหล่านี้อยู่ในสเตจ EX จะต้องยกเลิกข้อมูลในไปป์ไลน์รีจิสเตอร์ IF/ID และ ID/EX ทุกครั้ง

- คำสั่ง BEQZ และ BNEZ ข้อมูลในไปป์ไลน์รีจิสเตอร์ IF/ID และ ID/EX จะถูกยกเลิกหรือไม่ขึ้นกับการตรวจสอบค่าใน Register_input

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.1 ดาต้าพาธของการทำนายผลลัพธ์ทางแยกเดิม

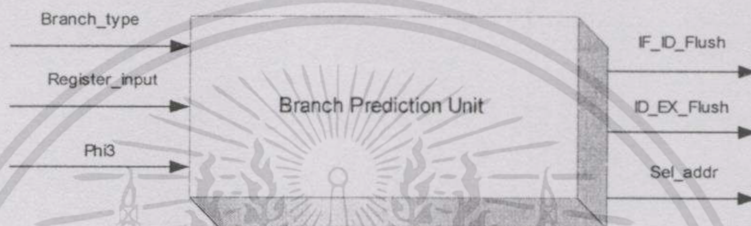
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- คำสั่ง BEQZ ถ้า Register_input เท่ากับศูนย์ จะต้องยกเลิกข้อมูลในไปป์ไลน์ริจิสเตอร์ IF/ID และ ID/EX

- คำสั่ง BNEZ ถ้า Register_input ไม่เท่ากับศูนย์ จะต้องยกเลิกข้อมูลในไปป์ไลน์ริจิสเตอร์ IF/ID และ ID/EX

2. โครงสร้างภายนอกของหน่วยตรวจสอบทางแยก

โครงสร้างภายนอกของหน่วยการตรวจสอบเป็นดังรูปที่ 4.2



รูปที่ 4.2 โครงสร้างภายนอกของหน่วยการตรวจสอบ

รายละเอียดของโครงสร้างภายนอกของหน่วยตรวจสอบทางแยกประกอบด้วย

1) อินพุท

- สัญญาณ Branch_type ขนาด 3 บิท เป็นสัญญาณเพื่อใช้แสดงชนิดของคำสั่งในกลุ่มคำสั่งทางแยก

- สัญญาณ Register_input ขนาด 32 บิท เป็นสัญญาณข้อมูลของริจิสเตอร์ เพื่อนำมาทดสอบค่าศูนย์

- สัญญาณ Phi3 ขนาด 1 บิท เป็นสัญญาณจากระบบสัญญาณนาฬิกา เพื่อจัดจังหวะการยกเลิกข้อมูลในไปป์ไลน์ริจิสเตอร์ IF/ID

2) เอาท์พุท

- สัญญาณ IF_ID_Flush ขนาด 1 บิท เป็นสัญญาณเพื่อใช้ยกเลิกข้อมูลในไปป์ไลน์ริจิสเตอร์ IF/ID

- สัญญาณ ID_EX_Flush ขนาด 1 บิท เป็นสัญญาณเพื่อใช้ยกเลิกข้อมูลในไปป์ไลน์ริจิสเตอร์ ID/EX

- สัญญาณ Sel_addr ขนาด 2 บิท เป็นสัญญาณเลือกของมัลติเพล็กเซอร์สำหรับเลือกแอดเดรสที่จะเฟ็ทช์คำสั่งถัดไป ณ สเตจ IF

"00" เลือกค่า PC+4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

"01" เลือกค่า Register สำหรับคำสั่ง (JR,JALR)

"10" เลือกค่า Branch Target Address ที่คำนวณจากเอแอลยู

"11" ไม่ใช้

พิจารณาชุดคำสั่งทางแยกแต่ละคำสั่งดังนี้

คำสั่ง J เป็นคำสั่งที่กระโดดไปทำงานยังตำแหน่งเป้าหมายโดยไม่มีเงื่อนไข (unconditional branch) โดยตำแหน่งเป้าหมายจะถูกคำนวณโดย (PC + 4) + อิมมีเดียทขนาด 26 บิทแบบคิดเครื่องหมาย คำสั่งนี้จะมีพฤติกรรม taken เสมอ ดังนั้นเมื่อคำสั่ง J ถูกเฟ็ทช์เข้ามาและทำงานจนถึงสเตจ EX หน่วยการตรวจสอบทางแยกจะต้องทำการยกเลิกคำสั่งที่อยู่ในสเตจ IF และ ID (เนื่องจากใช้หลักการทำนายว่าคำสั่งทางแยกมีพฤติกรรม not taken เสมอ) โดยการแทรกคำสั่ง nop ลงไปในไปป์ไลน์ริจิสเตอร์ IF/ID และ ID/EX และหน่วยตรวจสอบนี้จะต้องส่งสัญญาณเลือกของมัลติเพล็กเซอร์ที่อยู่ในสเตจ IF ให้เลือกตำแหน่งของเป้าหมายที่ถูกต้องโดยต้องเลือกแอดเดรสเป้าหมายที่มาจากเอแอลยู พิจารณาส่วนของโค้ดโปรแกรมต่อไปนี้

```
          J     LABEL1
          ADD   R1,R2,R3
          SUB   R4,R5,R6
          SLL   R7,R8,R9
          AND   R10,R11,R12
LABEL1 :  ORI   R17,R18,R19
```

ไดอะแกรมที่แสดงลำดับการทำงานของโค้ดโปรแกรมดังกล่าวแสดงดังรูปที่ 4.3

คำสั่ง	ไซเคิลการทำงาน							
	1	2	3	4	5	6	7	8
J LABEL1	IF	ID	EX	MEM	WB			
ADD R1,R2,R3		IF	ID	nop	nop	nop		
SUB R4,R5,R6			IF	nop	nop	nop	nop	
ORI R17,R18,R19				IF	ID	EX	MEM	WB

รูปที่ 4.3 ลำดับการทำงานของโค้ดโปรแกรมเมื่อทำงานกับคำสั่ง J

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สังเกตว่าในไซเคิลที่ 4 ไปป์ไลน์คำสั่งจะเกิดช่องว่างเพื่อไม่ให้โปรเซสเซอร์ประมวลผลคำสั่ง ADD R1,R2,R3 และคำสั่ง SUB R4,R5,R6 การทำงานของไปป์ไลน์ดำเนินต่อไปได้เมื่อเริ่มต้นไซเคิลที่ 4 และคำสั่งที่ถูกเฟ็ทช์เข้ามาประมวลผลจะเป็นคำสั่ง ORI R17,R18,R19 การทำงานของคำสั่งนี้เสีย 2 ไซเคิลเสมอ

คำสั่ง JAL เป็นคำสั่งที่กระโดดไปทำงานยังตำแหน่งเป้าหมายโดยไม่มีเงื่อนไข ตำแหน่งของเป้าหมายจะถูกคำนวณโดย (PC + 4) + อิมมีเดียทขนาด 26 บิท แต่คำสั่งนี้จะต้องเก็บค่า (PC + 4) ลงใน R31 คำสั่ง JAL ต้องทำงานคู่กับคำสั่ง JR เสมอในการเรียกฟังก์ชัน (Call function) มาทำงาน หน่วยการตรวจสอบทางแยกจะปฏิบัติต่อคำสั่ง JAL เช่นเดียวกับคำสั่ง J

คำสั่ง JR เป็นคำสั่งที่กระโดดไปทำงานยังตำแหน่งเป้าหมายโดยไม่มีเงื่อนไข โดยตำแหน่งของเป้าหมายจะอยู่ในค่ารีจิสเตอร์โอเปอแรนด์ของคำสั่ง JR คำสั่ง JR จะมีพฤติกรรม taken เสมอ ดังนั้นเมื่อคำสั่ง JR ถูกเฟ็ทช์เข้ามาและทำงานจนถึงสเตจ EX หน่วยตรวจสอบทางแยกจะต้องยกเลิกคำสั่งที่อยู่ในสเตจ IF และ ID โดยการแทรกคำสั่ง nop ลงไปในไปป์ไลน์รีจิสเตอร์ IF/ID และ ID/EX และหน่วยตรวจสอบนี้จะต้องส่งสัญญาณเลือกของมัลติเพล็กเซอร์ที่อยู่ในสเตจ IF ให้เลือกตำแหน่งของเป้าหมายที่ถูกต้อง โดยต้องเลือกแอดเดรสเป้าหมายที่มาจากค่ารีจิสเตอร์ในสเตจ EX การทำงานของคำสั่งนี้เสีย 2 ไซเคิลเสมอ ไดอะแกรมที่แสดงลำดับการทำงานจะเหมือนกับรูปที่ 4.3

คำสั่ง JALR เป็นคำสั่งที่กระโดดไปทำงานยังตำแหน่งเป้าหมายโดยไม่มีเงื่อนไข ตำแหน่งของเป้าหมายจะมาจากค่ารีจิสเตอร์โอเปอแรนด์ของคำสั่ง JALR แต่คำสั่งนี้จะต้องเก็บค่า (PC + 4) ลงใน R31 คำสั่ง JALR ต้องทำงานคู่กับคำสั่ง JR เสมอในการเรียกฟังก์ชัน (Call function) มาทำงาน หน่วยการตรวจสอบทางแยกจะปฏิบัติต่อคำสั่ง JALR เช่นเดียวกับคำสั่ง JR และการทำงานของคำสั่งนี้เสีย 2 ไซเคิลเสมอ

คำสั่ง BEQZ เป็นคำสั่งที่กระโดดไปทำงานยังตำแหน่งเป้าหมายโดยมีเงื่อนไข (conditional branch) ด้วยการทดสอบค่ารีจิสเตอร์โอเปอแรนด์ของคำสั่ง BEQZ ว่ามีค่าเท่ากับศูนย์หรือไม่ ดังนั้นเมื่อคำสั่ง BEQZ ถูกเฟ็ทช์เข้ามาและทำงานจนถึงสเตจ EX หน่วยตรวจสอบทางแยกจะต้องตรวจสอบผลลัพธ์จริงของคำสั่ง BEQZ ว่ามีพฤติกรรม Taken หรือ Not taken ถ้ามีพฤติกรรม Not taken (ค่ารีจิสเตอร์ไม่เท่ากับศูนย์) โปรเซสเซอร์ก็จะเฟ็ทช์คำสั่งต่อเนื่องไม่มีการกระโดดข้าม และไม่มีการสูญเสียไซเคิลการทำงาน แต่ถ้าพฤติกรรม Taken (ค่ารีจิสเตอร์เท่ากับศูนย์) โปรเซสเซอร์ก็จะเฟ็ทช์คำสั่งที่ตำแหน่งเป้าหมายแทน และยกเลิกคำสั่งที่ตามหลังคำสั่ง BEQZ การทำงานของคำสั่งเสีย 2 ไซเคิล ตำแหน่งเป้าหมายจะถูกคำนวณจาก (PC + 4) + อิมมีเดียทขนาด 16 บิทแบบคิดเครื่องหมาย การยกเลิกคำสั่งทำได้โดยการแทรกคำสั่ง nop ลงไปในไปป์ไลน์รีจิสเตอร์ IF/ID และ ID/EX และหน่วยนี้จะต้องส่งสัญญาณเลือกของมัลติเพล็กเซอร์ที่อยู่ในสเตจ IF ให้เลือกตำแหน่งเป้าหมายที่ถูกต้อง โดยต้องเลือกแอดเดรสเป้าหมายที่มาจากเอแอลยู พิจารณาส่วนของโค้ดโปรแกรมต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            BEQZ  R23,LABEL1
            ADD   R1,R2,R3
            SUB   R4,R5,R6
            SLL   R7,R8,R9
            AND   R10,R11,R12
LABEL1 :    ORI   R17,R18,R19
```

ไดอะแกรมที่แสดงลำดับการทำงานของโค้ดโปรแกรมดังกล่าวเมื่อคำสั่ง BEQZ กรณี taken แสดงดังรูปที่ 4.4

คำสั่ง	ไซเคิลการทำงาน							
	1	2	3	4	5	6	7	8
BEQZ R23,LABEL1	IF	ID	EX	MEM	WB			
ADD R1,R2,R3		IF	ID	nop	nop	nop		
SUB R4,R5,R6			IF	nop	nop	nop	nop	
ORI R17,R18,R19				IF	ID	EX	MEM	WB

รูปที่ 4.4 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BEQZ มีสถานะเป็น Taken

ไดอะแกรมที่แสดงลำดับการทำงานของโค้ดโปรแกรมดังกล่าวเมื่อคำสั่ง BEQZ กรณี Not taken แสดงดังรูปที่ 4.5

คำสั่ง	ไซเคิลการทำงาน							
	1	2	3	4	5	6	7	8
BEQZ R23,LABEL1	IF	ID	EX	MEM	WB			
ADD R1,R2,R3		IF	ID	EX	MEM	WB		
SUB R4,R5,R6			IF	ID	EX	MEM	WB	
SLL R7,R8,R9				IF	ID	EX	MEM	WB

รูปที่ 4.5 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BEQZ มีสถานะเป็น Not taken

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

คำสั่ง BNEZ เป็นคำสั่งที่กระโดดไปทำงานยังตำแหน่งเป้าหมายโดยมีเงื่อนไข (conditional branch) ด้วยการทดสอบค่ารีจิสเตอร์โอเปอแรนด์ของคำสั่ง BNEZ ว่ามีค่าไม่เท่ากับศูนย์หรือไม่ ดังนั้นเมื่อคำสั่ง BNEZ ถูกเฟ็ทช์เข้ามาและทำงานจนถึงสเตจ EX หน่วยตรวจสอบทางแยกจะต้องตรวจสอบผลลัพธ์จริงของคำสั่ง BNEZ ว่าพฤติกรรมเป็น Taken หรือ Not taken ถ้ามีพฤติกรรม Not taken (ค่ารีจิสเตอร์เท่ากับศูนย์) โปรเซสเซอร์ก็จะเฟ็ทช์คำสั่งต่อเนื่องไม่มีการกระโดดข้ามและไม่มีการสูญเสียไซเคิลการทำงาน แต่ถ้ามีพฤติกรรม Taken (ค่ารีจิสเตอร์ไม่เท่ากับศูนย์) โปรเซสเซอร์ก็จะเฟ็ทช์คำสั่ง ณ ตำแหน่งเป้าหมายแทน และยกเลิกคำสั่งที่ตามหลังคำสั่ง BNEZ จะสูญเสียไซเคิลการทำงาน 2 ไซเคิล ตำแหน่งเป้าหมายจะถูกคำนวณจาก (PC + 4) + อิมมีเดียทขนาด 16 บิทแบบคิดเครื่องหมาย การยกเลิกคำสั่งทำได้โดยการแทรกคำสั่ง nop ลงไปในไปป์ไลน์รีจิสเตอร์ IF/ID และ ID/EX และหน่วยตรวจสอบนี้จะต้องส่งสัญญาณเลือกของมัลติเพล็กเซอร์ที่อยู่ในสเตจ IF ให้เลือกตำแหน่งเป้าหมายที่ถูกต้อง โดยต้องเลือกแอดเดรสเป้าหมายที่มาจากเอแอลยู พิจารณาส่วนของโค้ดโปรแกรมต่อไปนี้

```
          BNEZ  R23,LABEL1
          ADD   R1,R2,R3
          SUB   R4,R5,R6
          SLL   R7,R8,R9
          AND   R10,R11,R12
LABEL1 :  ORI   R17,R18,R19
```

ไดอะแกรมที่แสดงลำดับการทำงานของโค้ดโปรแกรมดังกล่าวเมื่อคำสั่ง BNEZ กรณี Taken เป็นดังรูปที่ 4.6

คำสั่ง	ไซเคิลการทำงาน							
	1	2	3	4	5	6	7	8
BNEZ R23,LABEL1	IF	ID	EX	MEM	WB			
ADD R1,R2,R3		IF	ID	nop	nop	nop		
SUB R4,R5,R6			IF	nop	nop	nop	nop	
ORI R17,R18,R19				IF	ID	EX	MEM	WB

รูปที่ 4.6 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BNEZ มีสถานะเป็น Taken

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ไดอะแกรมที่แสดงลำดับการทำงานของโค้ดโปรแกรมดังกล่าวเมื่อคำสั่ง BNEZ กรณี Not taken เป็นดังรูปที่ 4.7

คำสั่ง	ไซเคิลการทำงาน							
	1	2	3	4	5	6	7	8
BNEZ R23,LABEL1	IF	ID	EX	MEM	WB			
ADD R1,R2,R3		IF	ID	EX	MEM	WB		
SUB R4,R5,R6			IF	ID	EX	MEM	WB	
SLL R7,R8,R9				IF	ID	EX	MEM	WB

รูปที่ 4.7 ลำดับการทำงานของโค้ดโปรแกรมเมื่อคำสั่ง BNEZ มีสถานะเป็น Not taken

4.2 การปรับปรุงการทำนายทางแยก

จากการแก้ปัญหาการควบคุมในโปรเซสเซอร์ไปป์ไลน์เดอลุกซ์ด้วยการปรับปรุงโครงสร้างของดาต้าพาธบางส่วนและวิธีการทำนายผลลัพธ์ของคำสั่งทางแยกเดิม ซึ่งสามารถแก้ปัญหาควบคุมได้โดยไม่เกิดปัญหาข้อมูล รวมทั้งรองรับชุดคำสั่งทางแยกได้ทุกคำสั่ง เมื่อการทำนายผลลัพธ์ถูกต้องจะไม่มีการสูญเสียไซเคิล (ขณะที่กรณีอื่นๆจะเสีย 2 ไซเคิล) หากได้มีการพิจารณาอย่างละเอียดพบว่าสามารถลดการสูญเสียจำนวนไซเคิลลงได้ให้เหลือน้อยกว่าที่สรุปไว้ในตารางที่ 4.1 โดยการปรับปรุงโครงสร้างดาต้าพาธและปรับปรุงวิธีการทำนายทางแยก ซึ่งงานวิจัยนี้ขอนำเสนอหลักการดังกล่าวตามลำดับดังนี้

4.2.1 หลักการของหน่วยความจำและการทำนาย

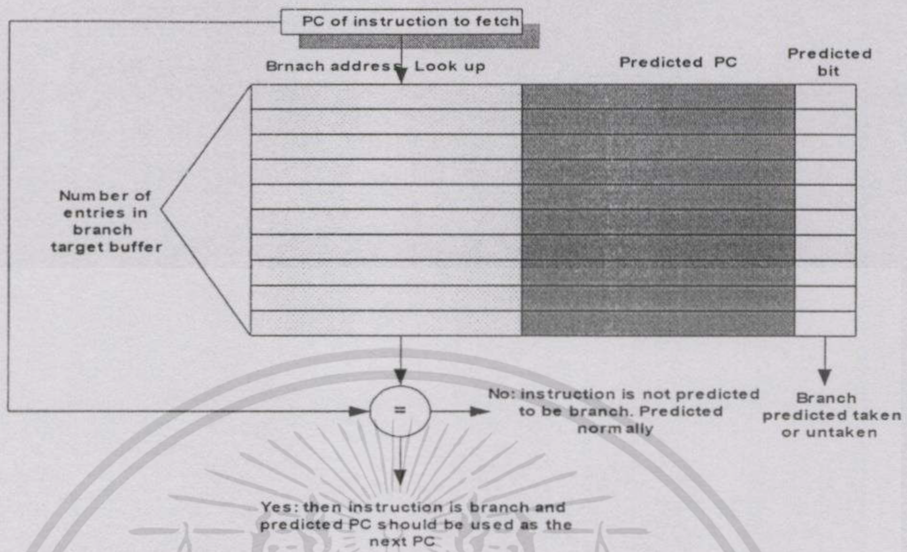
ทำการเก็บพฤติกรรมของคำสั่งทางแยกไว้ในหน่วยความจำที่เรียกว่าบีทีบี (Branch Target Buffer: บีทีบี) และบีเอชที (Branch History Table: BHT) และใช้วิธีการทำนายผลลัพธ์ของคำสั่งทางแยกแบบไดนามิก ซึ่งมีรายละเอียดของแต่ละส่วนดังนี้

1. บีทีบี

เป็นหน่วยความจำที่เก็บตำแหน่งของคำสั่งทางแยก (Branch address) ตำแหน่งปลายทางของคำสั่งทางแยกที่จะกระโดดไป (Branch target address) และบิทการทำนาย (Predicted bit) แต่ละแถวในบีทีบีเรียกว่าเอนทรี (Entry) จำนวนแถวทั้งหมดเรียกว่าขนาดบัฟเฟอร์ (Buffer size) ข้อมูล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

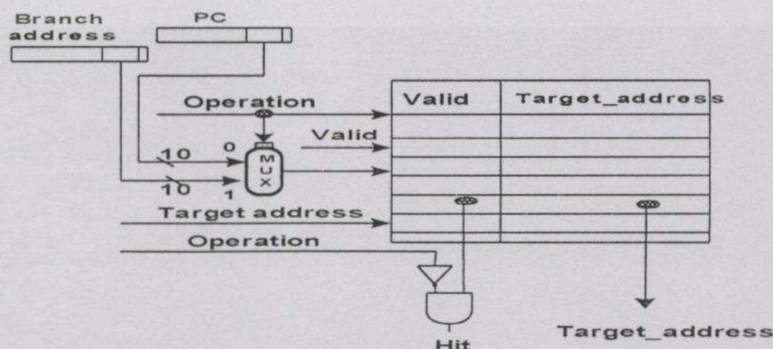
ของคำสั่งทางแยกจะถูกเก็บไว้ในบีทีบีเพื่อใช้ในการทำนายตำแหน่งปลายทางของคำสั่งทางแยก โครงสร้างของบีทีบีแสดงดังรูปที่ 4.8



รูปที่ 4.8 โครงสร้างของบีทีบี [2]

พิจารณารูปที่ 4.8 เมื่อไมโครโปรเซสเซอร์เฟ็ทช์คำสั่งทางแยกเข้ามา โปรเซสเซอร์จะนำตำแหน่งของคำสั่งทางแยกนั้นไปเปรียบเทียบกับค่าที่อยู่ในฟิลด์ Branch address ถ้าตรงกัน ก็จะนำข้อมูลตำแหน่งปลายทางจากฟิลด์ Predicted PC ในบีทีบีมาใช้เป็นตำแหน่งปลายทางที่จะต้องกระโดดไปทำงาน ดังนั้นถ้าผลลัพธ์จริงของคำสั่งทางแยกนั้นมีค่าตรงกับข้อมูลของบิทการทำนายจากฟิลด์ predicted bit ในบีทีบี โปรเซสเซอร์ก็จะทำงานเต็มประสิทธิภาพไม่มีการสูญเสียไซเคิล แต่ถ้าผลลัพธ์จริงของคำสั่งทางแยกนั้นไม่ตรงกับค่าบิทการทำนายที่ได้จากบีทีบี โปรเซสเซอร์ก็จะต้องยกเลิกข้อมูลที่เฟ็ทช์เข้ามาที่ตำแหน่ง Predicted PC ทำให้โปรเซสเซอร์ทำงานไม่เต็มประสิทธิภาพนั่นคือมีการสูญเสียไซเคิล

จากโครงสร้างของบีทีบีดังกล่าว จึงได้ปรับโครงสร้างของบีทีบีใหม่เพื่อให้เหมาะสมกับการแก้ปัญหาคำสั่งทางแยกในงานวิจัย โครงสร้างของบีทีบีที่ใช้ในงานวิจัยเป็นดังรูปที่ 4.9



รูปที่ 4.9 โครงสร้างของบีทีบีในงานวิจัย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูปที่ 4.9 การอ้างถึงหน่วยความจำเป็นแบบตรง (Direct map) ขนาด 1 K แต่ละฟิลด์ในบีทีบีมีความหมายดังนี้

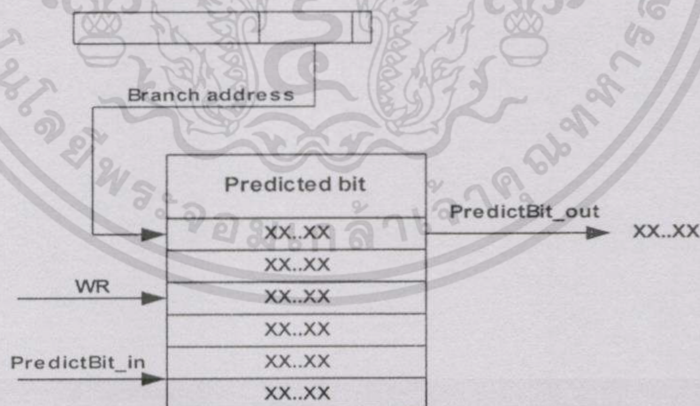
- ฟิลด์ Valid เป็นฟิลด์ที่ใช้ตรวจสอบข้อมูลที่อยู่ในเอนทรีว่าใช้งานได้หรือไม่ ถ้าค่าของ Valid = 1 แสดงว่าข้อมูลใช้ได้ ถ้าค่า Valid = 0 แสดงว่าเอนทรีนี้ไม่มีข้อมูลหรือข้อมูลถูกลบออกไปแล้ว

- ฟิลด์ Target address เป็นฟิลด์ที่เก็บตำแหน่งเป้าหมายของคำสั่งทางแยก

สำหรับบีทีบีที่ออกแบบจะเก็บข้อมูลของคำสั่งทางแยกแบบไม่มีเงื่อนไขทั้งหมด และคำสั่งทางแยกแบบมีเงื่อนไขที่มีพฤติกรรมเป็น Taken เท่านั้น

หลักการทำงานของบีทีบีเมื่อพิจารณาจากรูปที่ 4.9 คือ ประกอบด้วยโอเปอเรชันการทำงานสองโอเปอเรชันได้แก่ read (Operation = 0) กับ update (Operation = 1) โดยโอเปอเรชัน read ทำการค้นหาข้อมูลในหน่วยความจำบีทีบี ซึ่งเกิดขึ้นเมื่อคำสั่งนั้นอยู่ในสเตจ IF และใช้ค่า 10 บิทจากโปรแกรมเคาน์เตอร์เป็นตัวค้นหา ถ้าพบข้อมูลจะให้ค่า HIT = 1 แล้วนำค่าจากฟิลด์ Target Address ออกจากบีทีบี ส่วนโอเปอเรชัน update เกิดขึ้นเมื่อคำสั่งนั้นอยู่ในสเตจ EX ทำการเขียน แก้ไข หรือลบข้อมูลในบีทีบี ตำแหน่งที่ต้องการแก้ไขเก็บอยู่ใน Branch address ข้อมูลที่ต้องแก้ไขคือฟิลด์ Target address สำหรับโอเปอเรชัน update นี้ค่าของ HIT = 0 เสมอ

2. โครงสร้างของบีเอชที



รูปที่ 4.10 โครงสร้างของบีเอชที

บีเอชที [7] เป็นหน่วยความจำที่ใช้เก็บพฤติกรรมของคำสั่งทางแยกแต่ละคำสั่ง โดยมีโครงสร้างดังรูปที่ 4.10 ฟิลด์ Predicted bit จะมีขนาดเท่าใดก็ได้แล้วแต่วิธีการทำนาย สำหรับงานวิจัยนี้ได้กำหนดให้มีขนาด 2 บิท โดยกำหนดค่าเริ่มต้นของแต่ละคำสั่งให้มีค่าเป็น 01 (Weak not taken) บีเอชทีที่ใช้ในงานวิจัยนี้จะเก็บพฤติกรรมการทำนายเฉพาะคำสั่งแบบมีเงื่อนไขเท่านั้น และมีขนาด

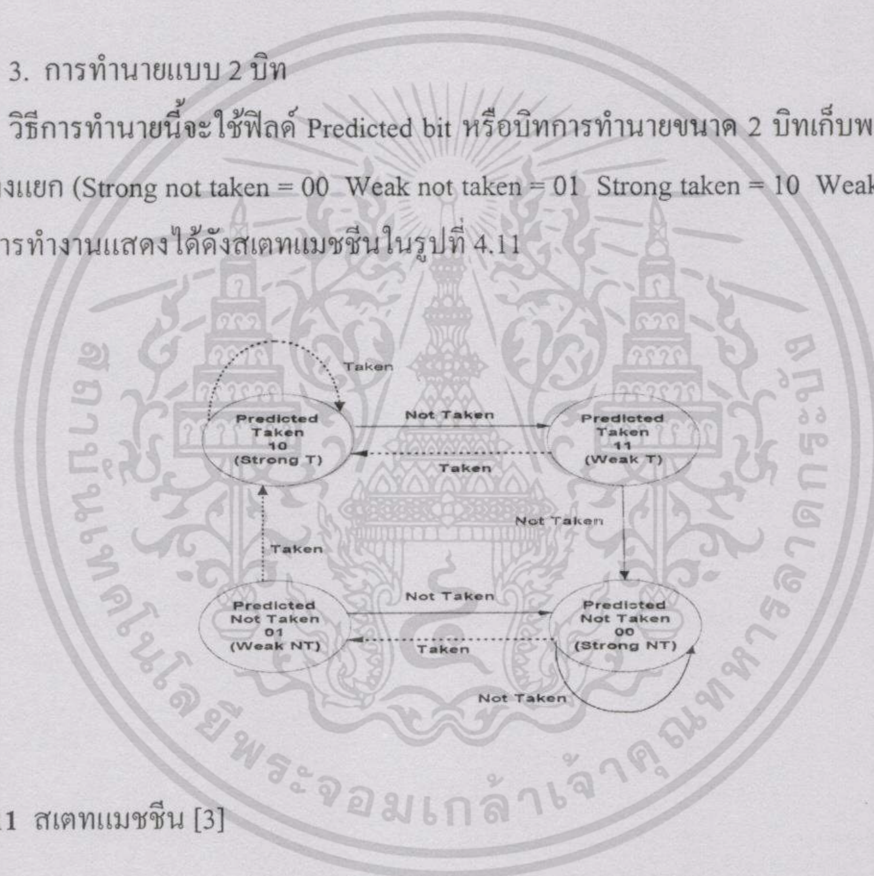
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1K โดยใช้ค่า 10 บิทของ Branch address เป็นคีย์ในการอ่าน (WR = 0) และแก้ไขข้อมูล (WR = 1) ค่า Predicted bit ในบีเอชทีจะถูกอ่านและแก้ไขในสเตจ ID ตามวิธีการทำนายแบบ 2 บิท แต่ละแถวในบีเอชทีเรียกว่าเอนทรี จำนวนแถวทั้งหมดเรียกว่าขนาดบัฟเฟอร์เช่นเดียวกับบีทีบี

หลักการทำงาน ถ้าคำสั่งในสเตจ ID เป็นคำสั่งทางแยกแบบมีเงื่อนไข ไมโครโปรเซสเซอร์จะหาผลลัพธ์จริงของคำสั่งทางแยกนั้น ในขณะเดียวกับที่อ่านค่า Predicted bit ของคำสั่งนั้นจากบีเอชทีด้วย เพื่อแก้ไขค่า Predicted bit แล้วบันทึกกลับลงไปในบีเอชทีตามหลักการทำนายแบบ 2 บิท ซึ่งผลลัพธ์ของ Predicted bit หลังจากแก้ไขแล้วจะเป็นตัวกำหนดว่าควรจะเพิ่มหรือลบคำสั่งทางแยกนั้นจากบีทีบี

3. การทำนายแบบ 2 บิท

วิธีการทำนายนี้จะใช้ฟิลด์ Predicted bit หรือบิทการทำนายขนาด 2 บิทเก็บพฤติกรรมของคำสั่งทางแยก (Strong not taken = 00 Weak not taken = 01 Strong taken = 10 Weak taken = 11) มีหลักการทำงานแสดงได้ดังสเตทแมชชีนในรูปที่ 4.11



รูปที่ 4.11 สเตทแมชชีน [3]

หลักการทำงานตามวิธีการทำนาย 2 บิทเป็นดังนี้ สมมติเริ่มต้นค่าในบิทการทำนายเป็น 00 คือทำนายว่ามีพฤติกรรมเป็น Not taken ชนิด Strong ถ้าค่าผลลัพธ์จริงของคำสั่งทางแยกเป็น Not taken ด้วยแสดงว่าทำนายถูกต้อง ค่าใน Predicted bit จะเป็นค่าเดิม แต่ถ้าผลลัพธ์จริงเป็น taken แสดงว่าทำนายผิด ค่าบิทการทำนายจะถูกแก้ไขเป็น 01 (Weak not taken) เมื่อเวลาผ่านไปไมโครโปรเซสเซอร์จะเฟ็ตช์คำสั่งทางแยกนี้เข้ามาอีกครั้ง ค่าในบิทการทำนายที่ใช้ในการทำนายเป็น 01 คือทำนายว่าเป็น Not taken ชนิด Weak และถ้าผลลัพธ์จริงเป็น Taken คือทำนายผิด ค่าบิทการทำนายจะถูกแก้เป็น 10 (Strong taken) แต่ถ้าผลลัพธ์จริงเป็น Not taken คือทำนายถูก บิทการทำนายจะถูกแก้ไขเป็น 00 (Strong not taken) เมื่อไมโครโปรเซสเซอร์เฟ็ตช์คำสั่งทางแยกนี้เข้ามาอีกครั้ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ค่าในบิทการทำนายก็จะมีการเปลี่ยนแปลงไปมาระหว่าง 4 ค่านี้ขึ้นกับผลการทำนายในบิทการทำนายว่าถูกต้องหรือไม่ ในงานวิจัยนี้คำสั่งทางแยกแบบมีเงื่อนไขทุกคำสั่งจะถูกเซ็ตให้อยู่ในสถานะ 01 เสมอ

4.2.2 การปรับปรุงโครงสร้างของดาต้าพาธ

ได้มีการปรับปรุงโครงสร้างของดาต้าพาธเพื่อให้สอดคล้องกับวิธีการทำนายทางแยกใหม่ที่นำเสนอสำหรับการแก้ปัญหาคำสั่งทางแยก โครงสร้างของดาต้าพาธที่ปรับปรุงและใช้ในงานวิจัยเป็นดังรูปที่ 4.12

จากโครงสร้างของดาต้าพาธ ได้ย้ายหน่วยตรวจสอบผลลัพธ์ของคำสั่งทางแยกและการคำนวณตำแหน่งเป้าหมายของคำสั่งทางแยกมาไว้ในสเตจ ID และนำบีทีบีมาเชื่อมต่อกับสเตจ IF และสเตจ EX และนำบีเอชทีเชื่อมต่อกับสเตจ ID หน่วยความจำบีทีบีถูกควบคุมโดย BTB controller หน่วยความจำบีเอชทีถูกควบคุมโดย Branch&BHT controller ทั้ง BTB controller และ Branch&BHT controller จะมีลอจิกควบคุมให้คำสั่งทางแยกที่เข้ามาในไปป์ไลน์คำสั่งมีการโฟลว (flow) ตามโฟลวชาร์ท (flowchart) ดังรูปที่ 4.13 หัวข้อ 4.2.3 การปรับปรุงการทำนายทางแยก

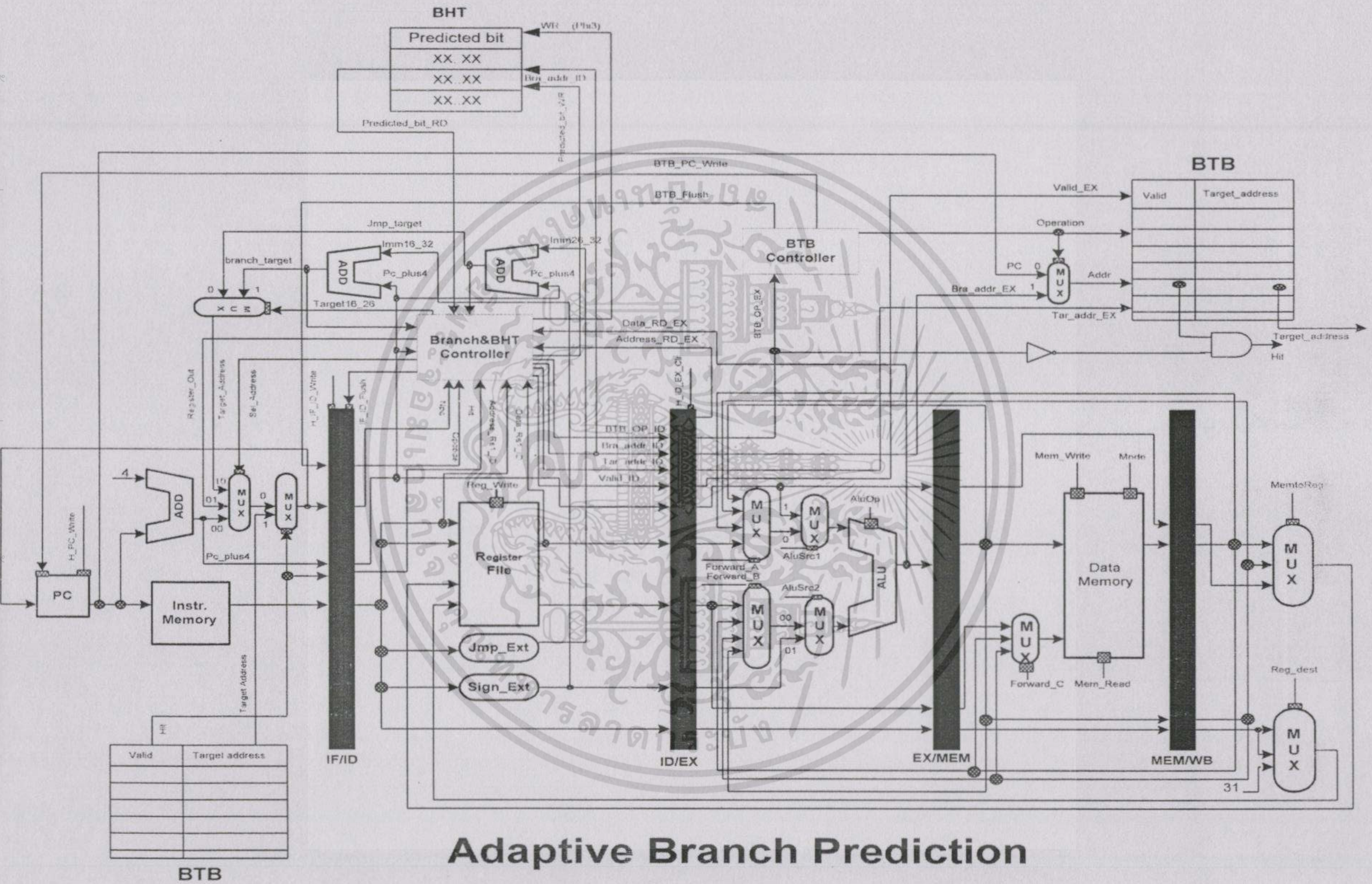
4.2.3 การปรับปรุงวิธีการทำนายทางแยก

การปรับปรุงการทำนายทางแยกใหม่นี้ได้ใช้หลักการของบีทีบีร่วมกับบีเอชที และใช้วิธีการทำนายแบบ 2 บิท โดยมี BTB controller และ Branch&BHT controller ควบคุมลอจิกของคำสั่งทางแยกในไปป์ไลน์คำสั่งให้เป็นไปตามโฟลว์ชาร์ท ซึ่งมีการทำงานเป็นไปตามรูปที่ 4.13 จากรูปจะแบ่งการพิจารณาเป็น 2 กลุ่มคำสั่งคือ กลุ่มคำสั่งทางแยกแบบไม่มีเงื่อนไข (J JAL JR JALR) และกลุ่มคำสั่งทางแยกแบบมีเงื่อนไข(BEQZ BNEZ)

4.2.3.1 กลุ่มคำสั่งทางแยกแบบไม่มีเงื่อนไข ในกลุ่มคำสั่งนี้เกี่ยวข้องกับหน่วยความจำบีทีบีเท่านั้น ไม่เกี่ยวข้องกับหน่วยความจำบีเอชที มีการทำงานดังนี้ เมื่อคำสั่งทางแยกอยู่ในสเตจ IF จะต้องนำค่าโปรแกรมเคาน์เตอร์เป็นคีย์สำหรับค้นหาข้อมูลของคำสั่งทางแยกนั้นในบีทีบี

1. ถ้าพบข้อมูลของคำสั่งทางแยกนั้นในบีทีบี แสดงว่าคำสั่งทางแยกนั้นได้เคยถูกประมวลผลมาก่อนแล้ว และให้นำค่าตำแหน่งเป้าหมายนั้นจากบีทีบีมาเป็นตำแหน่งของคำสั่งถัดไปที่จะถูกเฟ็ทช์เข้ามาในไมโครโปรเซสเซอร์ เมื่อคำสั่งทางแยกนั้นถูกย้ายการทำงานมาในสเตจ ID คำสั่งทางแยกนั้นจะต้องถูกตรวจสอบว่าเป็นคำสั่งทางแยกแบบไม่มีเงื่อนไขชนิดใด ถ้าเป็นชนิด J หรือ JAL ให้ไปป์ไลน์คำสั่งทำงานตามปกติ แต่ถ้าเป็นชนิด JR หรือ JALR ให้ตรวจสอบว่าตำแหน่งเป้าหมายของคำสั่งทางแยกนั้นว่าถูกต้องหรือไม่โดยตรวจสอบค่าในตำแหน่งเป้าหมายจากบีทีบีกับค่าจริงของริจิสเตอร์โอเปอแรนด์ขณะนั้น ถ้าถูกต้องคือมีค่าเท่ากันให้ไปป์ไลน์คำสั่งทำงานตาม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.12 โครงสร้างดาต้าพาธแบบปรับปรุงการทำนายทางแยกในงานวิจัย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

IF

Send PC to Instruction Memory and BTB

Found in BTB ?

No (12)

Yes (1) (hit)

ID

Send Target Address

JR,JALR, BNEQ,BNEZ ?

Yes (2)

No (11)

JR,JALR ?

Yes (8)

No (3)

JR,JALR and correct Target Address ?

No (9)

Yes (10)

Find Correct Target

Flush IF/ID

Fetch Instruction at New Target Address at next clock cycle

Find OutCome of BNEZ,BEQZ

Taken Branch ?

Yes (4)

No (5)

Flush IF/ID

Fetch Instruction at (PC+4) at next clock cycle

Updata BHT With Correct 2 bits Stage machine

2-bits state machine Taken ?

Yes (6)

No (7)

Update BHT with correct 2-bits state machine

BEQZ,BNEZ, J,JR,JALR,JAL ?

No (13)

Yes (14)

BEQZ,BNEZ ?

No (15)

Yes (16)

Find Target Address

Flush IF/ID

Fetch Instruction at Target Address at next clock cycle

Find OutCome of BNEZ,BEQZ and Target Address

Taken Branch ?

Yes (17)

No (20)

Flush IF/ID

Fetch Instruction at Target Address at next clock cycle

Update BHT with correct 2-bits state machine

Update BHT with correct 2-bits state machine

2-bits state machine Taken ?

No (18)

Yes (19)

EX

Normal Instruction Execute

Execute continue

1 Restart Fetch Instruction at Target Address at next clock cycle
2 Enter Target Address and set valid = 1 Into BTB
3 Flush IF/ID

1 Restart Fetch Instruction at Target Address at next clock cycle
2 Enter Target Address and set valid = 1 Into BTB
3 Flush IF/ID

Execute continue

Execute continue

Execute continue

1. Restart Fetch instruction at (PC+4) at next clock cycle
2. Delete Entry in BTB By Set Valid = '0'
3. Flush IF/ID

1. Restart Fetch instruction at New Target Address at next clock cycle
2. Update Target address in BTB
3. Flush IF/ID

Execute continue

Execute continue

รูปที่ 4.13 หลักการทำงานของการปรับปรุงการทำนายทางแยกในงานวิจัย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปกติ ถ้าไม่ถูกต้องคือมีค่าไม่เท่ากันให้ยกเลิกข้อมูลการเฟ็ทช์ในสเตจ IF และหาตำแหน่งเป้าหมายใหม่ให้ถูกต้องเพื่อนำตำแหน่งเป้าหมายนั้นไปเฟ็ทช์คำสั่งที่ไซเคิลถัดไป แล้วนำไปแก้ไขตำแหน่งเป้าหมายของคำสั่งทางแยกนั้นในบีทีบีให้ถูกต้องในสเตจ EX การแก้ไขข้อมูลในบีทีบีทำให้ต้องเสียไซเคิลการทำงานหนึ่งไซเคิลเนื่องจากไม่สามารถค้นหาข้อมูลและแก้ไขข้อมูลในบีทีบีได้ในเวลาเดียวกันจึงต้องหน่วงเวลาของคำสั่งเป้าหมายในสเตจ IF ไว้ก่อน เพื่อให้การแก้ไขข้อมูลในบีทีบีเสร็จเรียบร้อยแล้วจึงเฟ็ทช์คำสั่งเป้าหมายใหม่อีกครั้งที่ไซเคิลถัดไป ดังนั้นในกรณีนี้จึงสูญเสียไซเคิลการทำงาน 2 ไซเคิล

2. ถ้าไม่พบข้อมูลของคำสั่งทางแยกนั้นในบีทีบี แสดงว่าคำสั่งทางแยกถูกประมวลผลครั้งแรกเมื่อคำสั่งทางแยกนั้นถูกย้ายการทำงานมาในสเตจ ID จะต้องคำนวณตำแหน่งเป้าหมายของคำสั่งทางแยกนั้นและยกเลิกข้อมูลการเฟ็ทช์ของคำสั่งในสเตจ IF จากนั้นต้องเฟ็ทช์คำสั่งถัดไปที่ตำแหน่งเป้าหมายของคำสั่งทางแยกนั้นที่ไซเคิลถัดไป แล้วจึงนำข้อมูลตำแหน่งเป้าหมายของคำสั่งทางแยกนั้นบันทึกลงในบีทีบีที่สเตจ EX การบันทึกข้อมูลในบีทีบีทำให้ต้องเสียไซเคิลการทำงาน 1 ไซเคิลด้วยเหตุผลที่กล่าวมาแล้วในกรณีนี้จึงสูญเสียไซเคิลการทำงาน 2 ไซเคิล

4.2.3.2 กลุ่มคำสั่งทางแยกแบบมีเงื่อนไข กลุ่มคำสั่งนี้จะเกี่ยวข้องกับหน่วยความจำบีทีบีและบีเอชที โดยคำสั่งในกลุ่มนี้จะถูกบันทึกลงในบีทีบีก็ต่อเมื่อคำสั่งนั้นอยู่ในสถานะ Taken เมื่อพิจารณาจากสถานะของ 2 บิทสเตทแมชชีนซึ่งสถานะ Taken คือ 10 และ 11 และจะถูกลบออกจากบีทีบีเมื่อสถานะของ 2 บิทสเตทแมชชีนอยู่ในสถานะ Not taken คือ 00 และ 01 เมื่อคำสั่งทางแยกแบบมีเงื่อนไขอยู่ในสเตจ IF จะต้องนำค่าโปรแกรมเคาน์เตอร์มาเป็นคีย์ในการค้นหาข้อมูลของคำสั่งทางแยกนั้นในบีทีบี

1. ถ้าพบข้อมูลของคำสั่งทางแยกนั้นในบีทีบี แสดงว่าคำสั่งทางแยกนั้นได้เคยเก็บไว้ในบีทีบีมาก่อนแล้ว ให้นำค่าตำแหน่งเป้าหมายนั้นจากบีทีบีมาเป็นตำแหน่งของคำสั่งถัดไปที่จะถูกเฟ็ทช์เข้ามาในไมโครโปรเซสเซอร์ เมื่อคำสั่งทางแยกนั้นถูกย้ายการทำงานมาอยู่ในสเตจ ID จะต้องหาผลลัพธ์ของคำสั่งทางแยกนั้นว่าเป็น Taken หรือ Not Taken

1) ถ้าคำสั่งทางแยกเป็น Taken แสดงว่าทำนายผลลัพธ์ถูก ไม่ต้องแก้ไขข้อมูลในบีทีบี แต่จะต้องแก้ไขสถานะของ 2 บิทสเตทแมชชีนให้ถูกต้อง

2) ถ้าคำสั่งทางแยกเป็น Not Taken แสดงว่าทำนายผลลัพธ์ผิด ให้ยกเลิกข้อมูลการเฟ็ทช์ในสเตจ IF และเฟ็ทช์คำสั่งถัดไปที่ตำแหน่ง (PC+4) ของไซเคิลถัดไป นอกจากนี้ต้องแก้ไขสถานะของ 2 bit บิทสเตทแมชชีนของคำสั่งทางแยกนั้นในบีเอชทีให้ถูกต้อง เมื่อแก้ไขค่าให้ถูกต้องแล้วจะต้องตรวจสอบสถานะของ2 บิทสเตทแมชชีนว่าอยู่ในสถานะ Taken หรือ Not taken

- ถ้าสถานะ Taken ไปป์ไลน์คำสั่งจะทำงานต่อไปตามปกติ ดังนั้นในกรณีนี้จะเสียเพียงไซเคิลเดียวเท่านั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ถ้าสถานะ Not Taken จะต้องลบข้อมูลของคำสั่งทางแยกนั้นออกจากบีทีบี ซึ่งต้องเสียเวลาอีก 1 ไซเคิลด้วยเหตุผลดังที่กล่าวแล้ว ดังนั้นในกรณีนี้จะเสีย 2 ไซเคิล

2. ถ้าไม่พบข้อมูลของคำสั่งทางแยกนั้นในบีทีบี แสดงว่าคำสั่งทางแยกถูกประมวลครั้งแรกและเมื่อคำสั่งทางแยกนั้นย้ายการทำงานมาในสเตจ ID จะต้องคำนวณตำแหน่งเป้าหมายของคำสั่งทางแยกและหาผลลัพธ์ของคำสั่งทางแยกนั้นว่าเป็น Taken หรือ Not Taken

1) ถ้าคำสั่งทางแยกนั้นTaken แสดงว่าทำนายผิดให้ยกเลิกข้อมูลการเฟ็ทช์ในสเตจ IF และจะเฟ็ทช์คำสั่งถัดไปที่ตำแหน่งเป้าหมายที่คำนวณได้ที่ไซเคิลถัดไป และต้องแก้ไขสถานะของ 2 บิทสเตจแมชชีนของคำสั่งทางแยกนั้นในบีเอชทีให้ถูกต้อง เมื่อแก้ไขค่าถูกต้องแล้วจะต้องตรวจสอบสถานะของ2 บิทสเตจแมชชีนว่าอยู่ในสถานะ Taken หรือ Not Taken

- ถ้าสถานะ Taken ต้องนำค่าข้อมูลของของคำสั่งทางแยกนั้น ซึ่งเป็นตำแหน่งเป้าหมายที่คำนวณได้บันทึกลงในบีทีบีที่ Stage EX ซึ่งต้องเสียเวลาอีก 1ไซเคิล ดังนั้นในกรณีนี้จะเสีย 2 ไซเคิล

- ถ้าสถานะ Not Taken ไปป์ไลน์คำสั่งจะทำงานต่อไปตามปกติ ดังนั้น ในกรณีนี้จะเสียไซเคิลเดียวเท่านั้น

2) คำสั่งทางแยกนั้น Not Taken ต้องแก้ไขสถานะของ 2 บิทสเตจแมชชีนของคำสั่งทางแยกนั้นในบีเอชทีให้ถูกต้อง

4.2.4 ผลการปรับปรุงวิธีการทำนายทางแยก

เมื่อได้ปรับปรุงดาต้าพาธ รวมทั้งวิธีการทำนายผลลัพธ์ของคำสั่งทางแยกแบบ 2 บิทโดยใช้หลักการเก็บผลลัพธ์การทำนายไว้ในหน่วยความจำบีทีบีร่วมกับบีเอชที เพื่อแก้ปัญหาการควบคุมในไปป์ไลน์คำสั่ง ผลที่ได้รับคือการสูญเสียจำนวนไซเคิลเนื่องจากปัญหาการควบคุมลดลง สรุปเป็นแต่ละกรณีดังนี้

1. กลุ่มคำสั่งทางแยกที่ไม่มีเงื่อนไข

กลุ่มคำสั่งทางแยกที่ไม่มีเงื่อนไขแบ่งเป็น 2 กลุ่มคำสั่งคือ

- คำสั่ง J และคำสั่ง JAL จำนวนไซเคิลที่สูญเสียแสดงในตารางที่ 4.2

ตารางที่ 4.2 จำนวนไซเคิลที่สูญเสียสำหรับคำสั่ง J และคำสั่ง JAL

คำสั่ง	จำนวนไซเคิลที่สูญเสีย	
	กรณีที่พบข้อมูลในบีทีบี	กรณีที่ไม่พบข้อมูลในบีทีบี
J	0	2
JAL	0	2

- คำสั่ง JR และคำสั่ง JALR จำนวนไซเคิลที่สูญเสียแสดงในตารางที่ 4.3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 4.3 จำนวนไซเคิลที่สูญเสียสำหรับคำสั่ง JR และคำสั่ง JALR

คำสั่ง	จำนวนไซเคิลที่สูญเสีย		
	กรณีที่พบข้อมูลในบีทีบี		กรณีที่ไม่พบข้อมูลในบีทีบี
	ตำแหน่งปลายทางถูกต้อง	ตำแหน่งปลายทางผิด	
JR	0	2	2
JALR	0	2	2

2. กลุ่มคำสั่งทางแยกที่มีเงื่อนไข

- คำสั่ง Branch จำนวนไซเคิลที่สูญเสียแสดงในตารางที่ 4.4

ตารางที่ 4.4 จำนวนไซเคิลที่สูญเสียสำหรับคำสั่ง BEQZ และคำสั่ง BNEZ

คำสั่ง	จำนวนไซเคิลที่สูญเสีย					
	กรณีพบข้อมูลในบีทีบี			กรณีพบไม่ข้อมูลในบีทีบี		
	ผลลัพธ์จริง taken	ผลลัพธ์จริง not taken		ผลลัพธ์จริง taken		ผลลัพธ์จริง not taken
		2 bit status taken	2 bit status not taken	2 bit status taken	2 bit status not taken	
BEQZ	0	1	2	2	1	0
BNEZ	0	1	2	2	1	0

สำหรับการทดสอบหลักการปรับปรุงการทำนายทางแยก ได้มีการทดสอบกับโปรแกรมทดสอบ (Benchmark) ซึ่งเป็นโปรแกรมที่ประกอบด้วยลูปของคำสั่งทางแยกต่างๆ ดังที่จะได้กล่าวถึงอย่างละเอียดในเนื้อหาของบทต่อไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
การทดสอบการปรับปรุงการทำนายทางแยก

จากหลักการแก้ปัญหาคำสั่งทางแยกสำหรับโปรเซสเซอร์ไปป์ไลน์ด้วยการปรับปรุงการทำนายผลลัพธ์ของคำสั่งทางแยกที่นำเสนอในบทที่ 4 นั้น เพื่อแสดงว่าหลักการใหม่ดังกล่าวทำให้ประสิทธิภาพของโปรเซสเซอร์ดีขึ้น ในบทนี้จึงได้ทดสอบการทำงานของโปรเซสเซอร์ภายใต้หลักการแก้ปัญหาคำสั่งทางแยกด้วยวิธีการเดิมและวิธีการใหม่ โดยใช้โปรแกรมทดสอบ (Benchmark) ที่แตกต่างกันทั้งสิ้น 5 โปรแกรม และทำการวัดประสิทธิภาพของโปรเซสเซอร์ด้วยตัวแปรบอกประสิทธิภาพ 4 ตัวแปร นอกจากนี้ยังได้พิจารณาเพิ่มเติมว่าสำหรับโปรแกรมทดสอบเดียวกันแต่พฤติกรรมของคำสั่งทางแยกไม่เหมือนกัน แบบแรกพฤติกรรมของคำสั่งทางแยกในโปรแกรมทดสอบมีสัดส่วน Taken และNot taken ไม่แตกต่างกัน และแบบที่สองซึ่งคำสั่งทางแยกมีพฤติกรรม Not taken สูงนั้น เมื่อทำงานด้วยหลักการเดิมและหลักการใหม่ให้ประสิทธิภาพการทำงานของไมโครโปรเซสเซอร์แตกต่างกัน การทำงานของโปรเซสเซอร์ด้วยหลักการใหม่ยังคงให้ประสิทธิภาพดีกว่า

5.1 ตัวแปรบอกประสิทธิภาพ

กำหนดให้กลุ่มคำสั่งทางแยก หมายถึงคำสั่งทางแยกทั้งหมดซึ่งประกอบด้วยคำสั่ง BEQZ BNEZ J JR JAL และ JALR แบ่งเป็นคำสั่งทางแยกแบบมีเงื่อนไข (Conditional branch) ได้แก่คำสั่ง BEQZ และ BNEZ และคำสั่งทางแยกแบบไม่มีมีเงื่อนไข (Unconditional branch) ได้แก่คำสั่ง J JR JAL และ JALR ส่วนตัวแปรบอกประสิทธิภาพประกอบด้วย จำนวนไซเคิลต่อคำสั่ง (CPI) ความเร็วที่เพิ่มขึ้นของไมโครโปรเซสเซอร์ไปป์ไลน์คำสั่ง เมื่อเทียบกับไมโครโปรเซสเซอร์ที่ไม่เป็นไปป์ไลน์ (Speedup) เปอร์เซ็นต์ความผิดพลาดของการทำนายผลลัพธ์ของคำสั่งทางแยกแบบมีเงื่อนไข (% Mispredict) และจำนวนไซเคิลที่สูญเสียต่อหนึ่งคำสั่งทางแยก (Branch penalty)

1. จำนวนไซเคิลต่อคำสั่ง (CPI)

คำนวณจาก (5.1)

$$\text{CPI} = \frac{\text{Total clock}}{\text{Instructions count}} \tag{5.1}$$

โดยที่

Total clock หมายถึง จำนวนไซเคิลทั้งหมดที่ใช้ประมวลผลในแต่ละโปรแกรมทดสอบ

Instructions count หมายถึง คำสั่งทั้งหมดที่ถูกประมวลผลในแต่ละ โปรแกรมทดสอบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. ความเร็วที่เพิ่มขึ้นเมื่อเทียบกับไมโครโปรเซสเซอร์ที่ไม่เป็นไปป์ไลน์ (Speedup)

คำนวณจาก (5.2)

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \{(\text{Branch frequency x Branch penalty})\}} \tag{5.2}$$

โดยที่

$$\text{Branch frequency} = \frac{\text{Total Conditional Branch} + \text{Total Unconditional Branch}}{\text{Instruction Count}}$$

3. เปอร์เซ็นต์ความผิดพลาดในการทำนายคำสั่งทางแยกแบบมีเงื่อนไข (% Mispredict)

หลักการเดิมใช้สมการ (5.3)

$$\%\ \text{Mispredict} = \frac{\text{numberof conditional Branch Taken}}{\text{Total Conditional Branch}} * 100 \tag{5.3}$$

หลักการใหม่ใช้สมการ (5.4)

$$\%\ \text{MisPredict} = \frac{\text{Wrong T} + \text{Wrong NT}}{\text{Total Conditional Branch}} * 100 \tag{5.4}$$

โดยที่

number of conditional Branch Taken หมายถึง จำนวนครั้งที่ผลลัพธ์จริงของคำสั่งทางแยกแบบมีเงื่อนไขเป็น Taken

Wrong T หมายถึง จำนวนครั้งที่ทำนายว่า Taken แต่ผลลัพธ์จริงเป็น Not taken

และ Wrong NT หมายถึง จำนวนครั้งที่ทำนายว่า Not taken แต่ผลลัพธ์จริงเป็น Taken

4. จำนวนไซเคิลที่สูญเสียต่อ 1 คำสั่งทางแยก (Branch penalty)

คำนวณจาก (5.5)

$$\text{Branch penalty} = \frac{(\text{Total Clock} - \text{Instructions Count} - \text{Data and Structure hazards stall} - 4)}{\text{Total Branch}} \tag{5.5}$$

5.2 โปรแกรมทดสอบ

1. โปรแกรมทดสอบที่ 1

โปรแกรมการบวกเลขจำนวนเต็มตามวิธี Fibonacci Sequence

จากกลุ่มตัวเลขซึ่งประกอบด้วยตัวเลขเรียงกัน 9 ตัว ดังนี้

0 1 1 2 3 5 8 13 21

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวเลขตัวที่ 10 จะเป็นตัวเลข 34 หรือ 22_{16} เนื่องจากพิจารณาได้ว่าการเรียงลำดับของตัวเลข 9 ตัวนั้นเกิดจากการบวกกันของตัวเลขที่ติดกันตามหลักการของไฟโบนักชี (Fibonacci Sequence) ซึ่งสามารถเขียนโปรแกรมการหาผลบวกตามหลักการดังกล่าวด้วยภาษาระดับสูงได้ ดังนี้

```
if  n <= 1
        then fib := n
else
        begin
                lofib := 0 ;
                hifib := 1 ;
                for  i := 2  to n  do
                        begin
                                x := lofib ;
                                lofib := hifib ;
                                hifib := x + lofib ;
                        end
        end
fib := hifib;
end;
```

เมื่อแปลงโค้ดโปรแกรมของภาษาระดับสูงให้เป็นภาษาแอสแซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์โดยกำหนดริจิสเตอร์ให้สอดคล้องกับตัวแปรในภาษาระดับสูง ดังนี้

ริจิสเตอร์ R1 แทนตัวแปร fib
ริจิสเตอร์ R2 แทนตัวแปร lofib
ริจิสเตอร์ R3 แทนตัวแปร hifib
ริจิสเตอร์ R4 แทนตัวแปร n
ริจิสเตอร์ R5 แทนตัวแปร i
ริจิสเตอร์ R6 แทนตัวแปร x
ริจิสเตอร์ R8 เก็บผลลัพธ์สุดท้ายของการหาผลบวก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลังจากที่ได้แปลงจากภาษาระดับสูงเป็นโค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์แล้ว ต้องแปลงเป็นแมชชีนโค้ดฐานสิบหกของโปรเซสเซอร์เดอลุกซ์ สรุปได้ดังตารางที่ 5.1

ตารางที่ 5.1 โค้ดแอสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 1

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADD R1,R3,R0	00000028	0060820
J LOOPX	0000002c	08000014
LOOP3:ADD R6,R2,R0	00000030	00403020
ADD R2,R3,R0	00000034	00601020
ADD R3,R6,R2	00000038	00421820
ADDI R5,R5,R1	0000003c	20a50001
J LOOP2	00000040	0bffffdc
LOOPX:ADD R8,R1,R0	00000044	00204020

2. โปรแกรมทดสอบที่ 2

โปรแกรมการหาค่าแฟ็คตอเรียล (Factorial) ของเลขจำนวนเต็มใดๆ

ถ้าต้องการค่าแฟ็คตอเรียลของเลขจำนวนเต็มใดๆ (n!) โดยที่

$$n! = n(n-1)! \quad \text{และ} \quad 0! = 1$$

ดังนั้น $10! = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 3628800$ หรือ $375f00_{16}$

หลักการของการหาค่าแฟ็คตอเรียลของเลขจำนวนเต็มใดๆเขียนเป็นภาษาระดับสูงได้ดังนี้

```
x = n; prod = 1; sum = 0; fact = 0
while (x <> 0)
{
sum = 0
for (k = 1; k <= x; k++)
    { sum = sum + prod }
prod = sum
x =x -1
}
fact = prod
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เนื่องจากในไมโครโปรเซสเซอร์เดอลุกซ์ที่ออกแบบไม่มีการคูณ ในส่วนของโปรแกรมที่มีการคูณเลขจึงต้องเปลี่ยนเป็นการบวกกันหลายๆ ครั้งแทน เมื่อนำไปทดสอบการทำงานกับไมโครโปรเซสเซอร์เดอลุกซ์ต้องแปลงภาษาระดับสูงเป็นภาษาแอสเซมบลีของโปรเซสเซอร์เดอลุกซ์ก่อน กำหนดริจิสเตอร์ของแอสเซมบลีให้สอดคล้องกับตัวแปรของภาษาระดับสูงดังนี้

ริจิสเตอร์ R1 แทนตัวแปร fact

ริจิสเตอร์ R2 แทนตัวแปร x

ริจิสเตอร์ R3 แทนตัวแปร prod

ริจิสเตอร์ R4 แทนตัวแปร sum

ริจิสเตอร์ R5 แทนตัวแปร k

หลังจากแปลงจากภาษาระดับสูงเป็นโค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์แล้ว ต้องแปลงเป็นแมชชีนโค้ดฐานสิบหกของโปรเซสเซอร์เดอลุกซ์ ดังตารางที่ 5.2

ตารางที่ 5.2 โค้ดแอสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 2

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADDI R2,R0,#10	00000000	2002000a
ADDI R3,R0,#1	00000004	20030001
LOOP0: SNE R20,R2,R0	00000008	0040a029
BEQZ R20,LOOPX	0000000c	12800028
LOOP1: AND R4,R4,R0	00000010	00802024
ADDI R5,R0,#1	00000014	20050001
LOOP2: SLE R20,R5,R2	00000018	00a2a02c
BNEZ R20,LOOP3	0000001c	1680000c
ADD R3,R4,R0	00000020	00801820
SUBI R2,R2,#1	00000024	28420001
J LOOP0	00000028	0bffffdc
LOOP3:ADD R4,R4,R3	0000002c	00832020
ADDI R5,R5,#1	00000030	20a50001
J LOOP2	00000034	0bffffe0
LOOPX: ADD R1,R3,R0	00000038	00600820

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. โปรแกรมทดสอบที่ 3

โปรแกรมเรียงลำดับตัวเลขจากน้อยไปมากโดยใช้วิธีเรียงลำดับแบบบับเบิลซอร์ท (Bubble sort)

ถ้าต้องการเรียงกลุ่มตัวเลข 8 ตัวที่มีลำดับเป็น

25 57 48 37 12 92 86 33

ซึ่งกลุ่มตัวเลขนี้เมื่ออยู่ในรูปเลขฐานสิบหกจะเป็น

19_{16} 39_{16} 30_{16} 25_{16} c_{16} $5c_{16}$ 56_{16} 21_{16}

หลังจากเรียงลำดับตัวเลขจากน้อยไปมากแล้วกลุ่มตัวเลขดังกล่าวต้องเรียงลำดับใหม่เป็น

12 25 33 37 48 57 86 92

หรือเมื่ออยู่ในรูปเลขฐานสิบหกจะได้

c_{16} 19_{16} 21_{16} 25_{16} 30_{16} 39_{16} 56_{16} $5c_{16}$

ซึ่งหลักการของการเรียงลำดับแบบ Bubble sort เขียนเป็นภาษาระดับสูงได้ดังนี้

```
num1 = 25; num2 = 25; num3 = 57; num4 = 48;
num5 = 37; num6 = 12; num7 = 92; num8 = 86; num9 = 33;
temp=0
for (k = 1; k < 8; k++)
{
        if (num1 > num2)
          { temp = num1; num1 = num2; num2 = temp}
        if (num2 > num3)
          { temp = num2; num2 = num3; num3 = temp}
        if (num3 > num4)
          { temp = num3; num3 = num4; num4 = temp}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        if  (num4 > num5)
          { temp = num4; num4 = num5; num5 = temp}
        if  (num5 > num6)
          { temp = num5; num5 = num6; num6 = temp}
        if  (num6 > num7)
          { temp = num6; num6 = num7; num7 = temp}
        if  (num7 > num8)
          { temp = num7; num7 = num8; num8 = temp}
}
```

เมื่อนำไปทดสอบการทำงานกับไมโครโปรเซสเซอร์เดอลุกซ์ต้องแปลงภาษาระดับสูงเป็นภาษาแอสเซมบลีของโปรเซสเซอร์เดอลุกซ์ก่อน กำหนดริจิสเตอร์ของแอสเซมบลีให้สอดคล้องกับตัวแปรของภาษาระดับสูงดังนี้

ริจิสเตอร์ R1 แทนตัวแปร num1 ซึ่งมีค่าตัวเลขเท่ากับ 25
ริจิสเตอร์ R2 แทนตัวแปร num2 ซึ่งมีค่าตัวเลขเท่ากับ 57
ริจิสเตอร์ R3 แทนตัวแปร num3 ซึ่งมีค่าตัวเลขเท่ากับ 48
ริจิสเตอร์ R4 แทนตัวแปร num4 ซึ่งมีค่าตัวเลขเท่ากับ 37
ริจิสเตอร์ R5 แทนตัวแปร num5 ซึ่งมีค่าตัวเลขเท่ากับ 12
ริจิสเตอร์ R6 แทนตัวแปร num6 ซึ่งมีค่าตัวเลขเท่ากับ 92
ริจิสเตอร์ R7 แทนตัวแปร num7 ซึ่งมีค่าตัวเลขเท่ากับ 86
ริจิสเตอร์ R8 แทนตัวแปร num8 ซึ่งมีค่าตัวเลขเท่ากับ 33
ริจิสเตอร์ R11 แทนตัวแปร temp

หลังจากแปลงจากภาษาระดับสูงเป็นโค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์แล้วต้องแปลงเป็นแมชชีนโค้ดฐานสิบหกของโปรเซสเซอร์เดอลุกซ์ สรุปได้ดังตารางที่ 5.3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.3 โค้ดแอสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 3

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADDI R1,R0,#25	00000000	20010019
ADDI R2,R0,#57	00000004	20020039
ADDI R3,R0,#48	00000008	20030030
ADDI R4,R0,#37	0000000c	20040025
ADDI R5,R0,#12	00000010	2005000c
ADDI R6,R0,#92	00000014	2006005c
ADDI R7,R0,#86	00000018	20070056
ADDI R8,R0,#33	0000001c	20080021
ADDI R9,R0,#1	00000020	20090001
ADDI R10,R0,#8	00000024	200a0008
AND R11,R11,R0	00000028	01605824
LOOP0: SLT R20,R9,R10	0000002c	012aa02a
BEQZ R20,LOOPX	00000030	128000b0
LOOP1: SGT R20,R1,R2	00000034	0022a02b
BEQZ R20,LOOP2	00000038	1280000c
ADD R11,R1,R0	0000003c	00205820
ADD R1,R2,R0	00000040	00400820
ADD R2,R11,R0	00000044	01601020
LOOP2: AND R11,R11,R0	00000048	01605824
SGT R20,R2,R3	0000004c	0043a02b
BEQZ R20,LOOP3	00000050	1280000c
ADD R11,R2,R0	00000054	00405820
ADD R2,R3,R0	00000058	00601020
ADD R3,R11,R0	0000005c	01601820
LOOP3: AND R11,R11,R0	00000060	01605824
SGT R20,R3,R4	00000064	0064a02b
BEQZ R20,LOOP4	00000068	1280000c
ADD R11,R3,R0	0000006c	00605820
ADD R3,R4,R0	00000070	00801820

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.3 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADD R4,R11,R0	00000074	01602020
LOOP4: AND R11,R11,R0	00000078	01605824
SGT R20,R4,R5	0000007c	0085a02b
BEQZ R20,LOOP5	00000080	1280000c
ADD R11,R4,R0	00000084	00805820
ADD R4,R5,R0	00000088	00a02020
ADD R5,R11,R0	0000008c	01602820
LOOP5: AND R11,R11,R0	00000090	01605824
SGT R20,R5,R6	00000094	00a6a02b
BEQZ R20,LOOP6	00000098	1280000c
ADD R11,R5,R0	0000009c	00a05820
ADD R5,R6,R0	000000a0	00c02820
ADD R6,R11,R0	000000a4	01603020
LOOP6: AND R11,R11,R0	000000a8	01605824
SLT R20,R6,R7	000000ac	00c7a02a
BNEZ R20,LOOP7	000000b0	1680000c
ADD R11,R6,R0	000000b4	00c05820
ADD R6,R7,R0	000000b8	00e03020
ADD R7,R11,R0	000000bc	01603820
LOOP7: AND R11,R11,R0	000000c0	01605824
SLT R20,R7,R8	000000c4	00e8a02a
BNEZ R20,LOOP8	000000c8	1680000c
ADD R11,R7,R0	000000cc	00e05820
ADD R7,R8,R0	000000d0	01003820
ADD R8,R11,R0	000000d4	01604020
LOOP8: AND R11,R11,R0	000000d8	01605824
ADDI R9,R9,#1	000000dc	21290001
J LOOP0	000000e0	0bffff48
LOOPX: ADD R1,R1,R0	000000e4	00200820

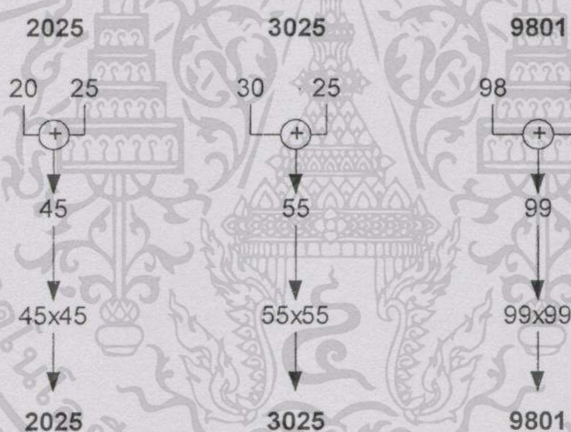
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.3 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADD R2,R2,R0	000000e8	00401020
ADD R3,R3,R0	000000ec	00601820
ADD R4,R4,R0	000000f0	00802020
ADD R5,R5,R0	000000f4	00a02820
ADD R6,R6,R0	000000f8	00c03020
ADD R7,R7,R0	000000fc	00e03820
ADD R8,R8,R0	00000100	01004020

4. โปรแกรมทดสอบที่ 4

เลขจำนวนเต็มบางตัวเช่น 2025 3025 หรือ 9801 มีคุณสมบัติพิเศษดังนี้



การหาตัวเลขที่มีคุณสมบัติดังกล่าวมีหลักการดังนี้

```
For n = 100 to 9999 do
        F ← int(n/100)
        L ← n - (100 x F)
        If (F+L)^2 <> n then
                Next n
        Else
                Output n, F, L
End
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ผลการทำงานของโปรแกรมทดสอบ n = 2025 F = 20 = 14_{16} และ L = 25 = 19_{16}

เมื่อนำไปทดสอบการทำงานกับไมโครโปรเซสเซอร์เดอลุกซ์ต้องแปลงภาษาระดับสูงเป็นภาษาแอสเซมบลีของโปรเซสเซอร์เดอลุกซ์ก่อน กำหนดริจิสเตอร์ของแอสเซมบลีให้สอดคล้องกับตัวแปรของภาษาระดับสูงดังนี้

ริจิสเตอร์ R1 แทนตัวแปร n
ริจิสเตอร์ R2 แทนตัวแปร F
ริจิสเตอร์ R3 แทนตัวแปร L
ริจิสเตอร์ R4, R6, R10, R11, R12 และ R13 แทนตัวแปรชั่วคราว
ริจิสเตอร์ R5 แทนตัวแปร เศษจากการหาร
ริจิสเตอร์ R20 แทนตัวแปรตรวจสอบลอจิก

หลังจากแปลงจากภาษาระดับสูงเป็นโค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์แล้วต้องแปลงเป็นแมชชีนโค้ดฐานสิบหกของโปรเซสเซอร์เดอลุกซ์ สรุปได้ดังตารางที่ 5.4

ตารางที่ 5.4 โค้ดแอสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 4

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADDI R1, R0, 2000	00000000	200107D0
SLEI R20, R1, 2025	00000004	703407E9
BNEZ R20, 8	00000008	16800008
ADD R0, R0, R0	0000000c	00000020
J 124	00000010	0800007C
ADD R5, R1, R0	00000014	00202820
AND R4, R4, R0	00000018	00802024
SGEI R20, R5, 100	0000001c	74B40064
BEQZ R20, 12	00000020	1280000C
SUBI R5, R5, 100	00000024	28A50064
ADDI R4, R4, 1	00000028	20840001
J -20	0000002c	0BFFFFEC
ADDI R10, R0, 1	00000030	200A0001

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.4 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADD R2, R4, R0	00000034	00801020
AND R4, R4, R0	00000038	00802024
AND R6, R6, R0	0000003c	00C03024
SLEI R20, R10, 100	00000040	71540064
BNEZ R20, 8	00000044	16800008
SUB R3, R1, R4	00000048	00241822
J 12	0000004c	0800000C
ADD R4, R4, R2	00000050	00822020
ADDI R10, R10, 1	00000054	214A0001
J -28	00000058	0BFFFFE4
AND R11, R11, R0	0000005c	01605824
ADD R11, R2, R3	00000060	00435820
AND R12, R12, R0	00000064	01806024
AND R13, R13, R0	00000068	01A06824
SLE R20, R12, R11	0000006c	018BA02C
BEQZ R20, 12	00000070	1280000C
ADD R13, R13, R11	00000074	01AB6820
ADDI R12, R12, 1	00000078	218C0001
J -20	0000007c	0BFFFFEC
SEQ R20, R13, R1	00000080	01A1A028
BNEZ R20, 8	00000084	16800008
ADDI R1, R1, 1	00000088	20210001
J -140	0000008c	0BFFFF74
ADD R1, R1, R0	00000090	00200820
ADD R2, R2, R0	00000094	00401020
ADD R3, R3, R0	00000098	00601820

5. โปรแกรมทดสอบที่ 5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Armstrong number เป็นตัวเลข n ใดๆ ที่มีคุณสมบัติคือ ตัวเลขแต่ละตัวของ n เมื่อยกกำลังสามแล้วนำมาบวกกันจะมีค่าเท่ากับตัวเลข n เดิม ดังนี้ n = 153 และ $1^3 + 5^3 + 3^3 = 153$ หลักการคำนวณหา Armstrong number เป็นดังนี้

```
For n = 135   to 155  do
        A ← int(n/100)
        B ← [ int(n/10) - 10 ] x A
        C ← n - [ (100 x A) + (10 x B) ]
        If  n <> A^3 + B^3 + C^3  then
                Next  n
        Else
                Output  n, A, B, C
End
```

ผลการทำงานของโปรแกรมทดสอบ n = 153 A = 1 B = 5 และ C = 3

เมื่อนำไปทดสอบการทำงานกับไมโครโปรเซสเซอร์เดอลุกซ์ต้องแปลงภาษาระดับสูงเป็นภาษาแอสเซมบลีของโปรเซสเซอร์เดอลุกซ์ก่อน กำหนดริจิสเตอร์ของแอสเซมบลีให้สอดคล้องกับตัวแปรของภาษาระดับสูงดังนี้

ริจิสเตอร์ R1 แทนตัวแปร n
ริจิสเตอร์ R2 แทนตัวแปร A
ริจิสเตอร์ R3 แทนตัวแปร B
ริจิสเตอร์ R4 แทนตัวแปร C
ริจิสเตอร์ R10, R11, R12, R13, R14 และ R15 แทนตัวแปรชั่วคราว
ริจิสเตอร์ R20 แทนตัวแปรตรวจสอบลอจิก

หลังจากแปลงจากภาษาระดับสูงเป็นโค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์แล้วต้องแปลงเป็นแมชชีนโค้ดฐานสิบหกของโปรเซสเซอร์เดอลุกซ์ สรุปได้ดังตารางที่ 5.5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.5 โค้ดแอสเซมบลี ตำแหน่ง และแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 5

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADDI R1, R0, 135	00000000	20010087
ADD R10, R1, R0	00000004	00205020
AND R11, R11, R0	00000008	01605824
SLEI R20, R1, 155	0000000c	7034009B
BNEZ R20, 8	00000010	16800008
ADD R1, R1, R0	00000014	00200820
J 412	00000018	0800019C
SGEI R20, R10, 100	0000001c	75540064
BNEZ R20, 8	00000020	16800008
ADD R2, R11, R0	00000024	01601020
J 12	00000028	0800000C
SUBI R10, R10, 100	0000002c	294A0064
ADDI R11, R11, 1	00000030	216B0001
J -28	00000040	0BFFFFE4
ADD R10, R1, R0	00000044	00205020
AND R11, R11, R0	00000048	01605824
AND R12, R12, R0	0000004c	01806024
SGEI R20, R10, 10	00000050	7554000A
BNEZ R20, 12	00000054	1680000C
ADD R11, R11, R0	00000058	01605820
SUBI R12, R11, 10	0000005c	296C000A
J 12	00000060	0800000C
SUBI R10, R10, 10	00000064	294A000A
ADDI R11, R11, 1	00000068	216B0001
J -32	0000006c	0BFFFFE0
AND R10, R10, R0	00000070	01405024
AND R11, R11, R0	00000074	01605824
SEQ R20, R10, R2	00000078	0142A028
BNEZ R20, 12	0000007c	1680000C

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.5 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADD R11, R11, R12	00000080	016C5820
ADDI R10, R10, 1	00000084	214A0001
J -20	00000088	0BFFFFEC
ADD R3, R11, R0	0000008c	01601820
AND R10, R10, R0	00000090	01405024
AND R11, R11, R0	00000094	01605824
AND R12, R12, R0	00000098	01806024
SEQ R20, R10, R2	0000009c	0142A028
BNEZ R20, 12	000000a0	1680000C
ADDI R11, R11, 100	000000a4	216B0064
ADDI R10, R10, 1	000000a8	214A0001
J -20	000000ac	0BFFFFEC
AND R10, R10, R0	000000b0	01405024
SEQ R20, R10, R3	000000b4	0143A028
BNEZ R20, 12	000000b8	1680000C
ADDI R12, R12, 10	000000bc	218C000A
ADDI R10, R10, 1	000000c0	214A0001
J -20	000000c4	0BFFFFEC
ADD R12, R12, R0	000000c8	01806020
ADD R11, R11, R12	000000cc	016C5820
SUB R4, R1, R11	000000d0	002B2022
ADD R10, R2, R0	000000d4	00405020
AND R11, R11, R0	000000d8	01605824
AND R12, R12, R0	000000dc	01806024
SEQ R20, R11, R10	000000e0	016AA028
BNEZ R20, 12	000000e4	1680000C
ADD R12, R12, R10	000000e8	018A6020
ADDI R11, R11, 1	000000ec	216B0001
J -20	000000f0	0BFFFFEC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.5 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADD R13, R12, R0	000000f4	01806820
AND R11, R11, R0	000000f8	01605824
AND R12, R12, R0	000000fc	01806024
SEQ R20, R11, R10	00000100	016AA028
BNEZ R20, 12	00000104	1680000C
ADD R12, R12, R13	00000108	018D6020
ADDI R11, R11, 1	0000010c	216B0001
J -20	00000110	0BFFFFEC
ADD R13, R12, R0	00000114	01806820
ADD R10, R3, R0	00000118	00605020
AND R11, R11, R0	0000011c	01605824
AND R12, R12, R0	00000120	01806024
SEQ R20, R11, R10	00000124	016AA028
BNEZ R20, 12	00000128	1680000C
ADD R12, R12, R10	0000012c	018A6020
ADDI R11, R11, 1	00000130	216B0001
J -20	00000134	0BFFFFEC
ADD R14, R12, R0	00000138	01807020
AND R11, R11, R0	0000013c	01605824
AND R12, R12, R0	00000140	01806024
SEQ R20, R11, R10	00000144	016AA028
BNEZ R20, 12	00000148	1680000C
ADD R12, R12, R14	0000014c	018E6020
ADDI R11, R11, 1	00000150	216B0001
J -20	00000154	0BFFFFEC
ADD R14, R12, R0	00000158	01807020
ADD R10, R4, R0	0000015c	00805020
AND R11, R11, R0	00000160	01605824
AND R12, R12, R0	00000164	01806024

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.5 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
SEQ R20, R11, R10	00000168	016AA028
BNEZ R20, 12	0000016c	1680000C
ADD R12, R12, R10	00000170	018A6020
ADDI R11, R11, 1	00000174	216B0001
J -20	00000178	0BFFFFEC
ADD R15, R12, R0	0000017c	01807820
AND R11, R11, R0	00000180	01605824
AND R12, R12, R0	00000184	01806024
SEQ R20, R11, R10	00000188	016AA028
BNEZ R20, 12	0000018c	1680000C
ADD R12, R12, R15	00000190	018F6020
ADDI R11, R11, 1	00000194	216B0001
J -20	00000198	0BFFFFEC
ADD R15, R12, R0	0000019c	01807820
AND R10, R10, R0	000001a0	01405024
AND R11, R11, R0	000001a4	01605824
AND R12, R12, R0	000001a8	01806024
ADD R10, R13, R14	000001ac	01AE5020
ADD R10, R10, R15	000001b0	014F5020
SEQ R20, R10, R1	000001b4	0141A028
BNEZ R20, 8	000001b8	16800008
ADDI R1, R1, 1	000001bc	20210001
J -436	000001c0	0BFFFE4C
ADD R1, R1, R0	000001c4	00200820
ADD R2, R2, R0	000001c8	00401020
ADD R3, R3, R0	000001cc	00601820
ADD R4, R4, R0	000001d0	00802020

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6. โปรแกรมทดสอบที่ 5 ที่มีพฤติกรรมเป็น Not taken สูง

เพื่อเปรียบเทียบการทำงานของโปรแกรมเดียวกันแต่มีพฤติกรรมการทำงานแตกต่างกัน จึงนำโปรแกรมทดสอบที่ 5 มาเขียนใหม่ให้มีลักษณะพฤติกรรมของคำสั่งทางแยก Not taken สูงหรือมีเงื่อนไขถูกต้องมากๆ แล้วทำการเปรียบเทียบค่าต่างๆของหลักการเดิมและหลักการใหม่ ตารางที่ 5.6 เป็นโค้ดโปรแกรมของโปรแกรมทดสอบที่ 5 ที่เขียนขึ้นใหม่

ตารางที่ 5.6 โค้ดแอสเซมบลี ตำแหน่งและแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์สำหรับโปรแกรมทดสอบที่ 5 ที่มีพฤติกรรมของคำสั่งทางแยก Not taken สูง

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
ADDI R10, R0, 134	00000000	200a0086
ADDI R10, R10, 1	00000004	214a0001
ADD R21, R0, R10	00000008	000aa820
SGTI R20, R10, 155	0000000c	6d54009b
BNEZ R20, 392	00000010	16800188
AND R12, R12, R0	00000014	01806024
SLTI R20, R21, 100	00000018	6ab40064
BNEZ R20, 12	0000001c	1680000c
SUBI R21, R21, 100	00000020	2ab50064
ADDI R22, R22, 1	00000024	22d60001
J -20	00000028	0bffffec
ADD R2, R0, R22	00000028	00161020
AND R22, R22, R0	00000030	02c0b024
ADD R21, R0, R10	00000034	000aa820
SLTI R20, R21, 10	00000038	6ab4000a
BNEZ R20, 12	0000003c	1680000c
SUBI R21, R21, 10	00000040	2ab5000a
ADDI R22, R22, 1	00000044	22d60001
J -20	00000048	0bffffec
ADD R11, R22, R0	0000004c	02c05820
AND R22, R22, R0	00000050	02c0b024
AND R21, R21, R0	00000054	02a0a824
SUBI R11, R11, 10	00000058	296b000a

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.6 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
SGE R20, R22, R2	0000005c	02c2a02d
BNEZ R20, 12	00000060	1680000c
ADD R21, R21, R11	00000064	02aba820
ADDI R22, R22, 1	00000068	22d60001
J -20	0000006c	0bffffec
ADD R3, R21, R0	00000070	02a01820
AND R21, R21, R0	00000074	02a0a824
AND R22, R22, R0	00000078	02c0b024
SGE R20, R22, R2	0000007c	02c2a02d
BNEZ R20, 12	00000080	1680000c
ADDI R21, R21, 100	00000084	22b50064
ADDI R22, R22, 1	00000088	22d60001
J -20	0000008c	0bffffec
ADD R12, R0, R21	00000090	00156020
AND R22, R22, R0	00000094	02c0b024
AND R21, R21, R0	00000098	02a0a824
SGE R20, R22, R3	0000009c	02c3a02d
BNEZ R20, 12	000000a0	1680000c
ADDI R21, R21, 10	000000a4	22b5000a
ADDI R22, R22, 1	000000a8	22d60001
J -20	000000ac	0bffffec
ADD R13, R0, R21	000000b0	00156820
AND R21, R21, R0	000000b4	02a0a824
AND R22, R22, R0	000000b8	02c0b024
ADD R21, R12, R13	000000bc	018da820
SUB R4, R10, R21	000000c0	01552022
AND R12, R12, R0	000000c4	01806024
AND R13, R13, R0	000000c8	01a06824
AND R21, R21, R0	000000cc	02a0a824

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.6 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
AND R22, R22, R0	000000d0	02c0b024
SGE R20, R22, R2	000000d4	02c2a02d
BNEZ R20, 12	000000d8	1680000c
ADD R21, R21, R2	000000dc	02a2a820
ADDI R22, R22, 1	000000e0	22d60001
J -20	000000e4	0bffffec
SGE R20, R12, R2	000000e8	0182a02d
BNEZ R20, 12	000000ec	1680000c
ADD R13, R13, R21	000000f0	01b56820
ADDI R12, R12, 1	000000f4	218c0001
J -20	000000f8	0bffffec
ADD R14, R0, R13	000000fc	000d7020
AND R12, R12, R0	00000100	01806024
AND R13, R13, R0	00000104	01a06824
AND R21, R21, R0	00000108	02a0a824
AND R22, R22, R0	0000010c	02c0b024
SGE R20, R22, R3	00000110	02c3a02d
BNEZ R20, 12	00000114	1680000c
ADD R21, R21, R3	00000118	02a3a820
ADDI R22, R22, 1	0000011c	22d60001
J -20	00000120	0bffffec
SGE R20, R12, R3	00000124	0183a02d
BNEZ R20, 12	00000128	1680000c
ADD R13, R13, R21	0000012c	01b56820
ADDI R12, R12, 1	00000130	218c0001
J -20	00000134	0bffffec
ADD R15, R0, R13	00000138	000d7820
AND R12, R12, R0	0000013c	01806024
AND R13, R13, R0	00000140	01a06824

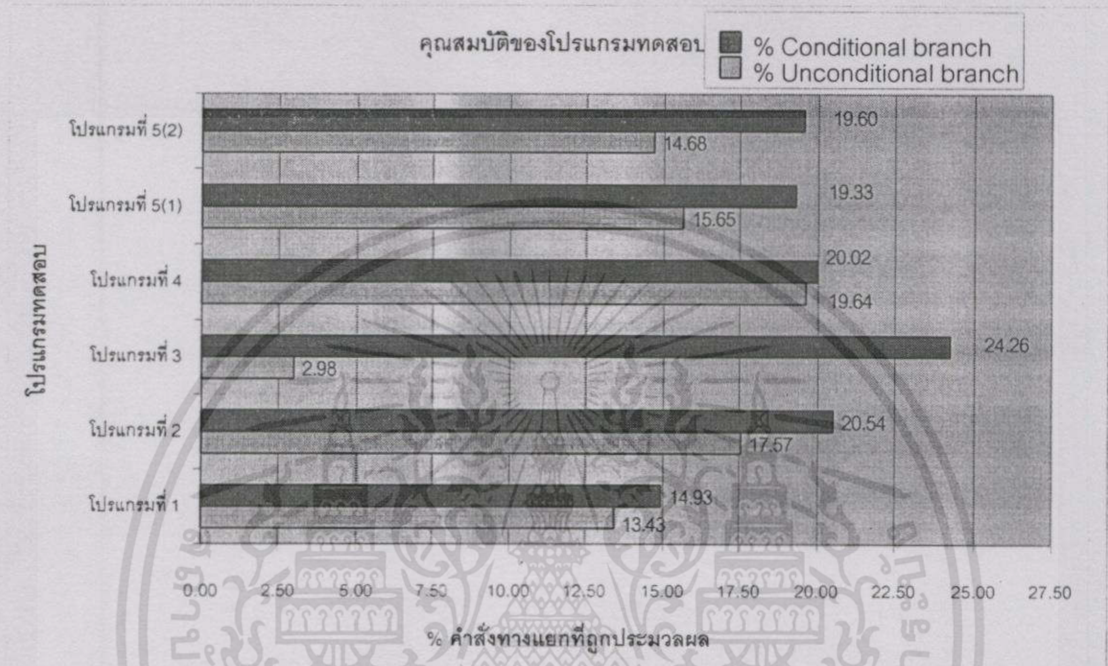
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.6 (ต่อ)

โค้ดแอสเซมบลีของไมโครโปรเซสเซอร์เดอลุกซ์	ตำแหน่ง	แมชชีนโค้ด
AND R21, R21, R0	00000144	02a0a824
AND R22, R22, R0	00000148	02c0b024
SGE R20, R22, R4	0000014c	02c4a02d
BNEZ R20, 12	00000150	1680000c
ADD R21, R21, R4	00000154	02a4a820
ADDI R22, R22, 1	00000158	22d60001
J -20	0000015c	0bffffec
SGE R20, R12, R4	00000160	0184a02d
BNEZ R20, 12	00000164	1680000c
ADD R13, R13, R21	00000168	01b56820
ADDI R12, R12, 1	0000016c	218c0001
J -20	00000170	0bffffec
ADD R16, R0, R13	00000174	000d8020
AND R12, R12, R0	00000178	01806024
AND R13, R13, R0	0000017c	01a06824
AND R21, R21, R0	00000180	02a0a824
AND R22, R22, R0	00000184	02c0b024
ADD R12, R14, R15	00000188	01cf6020
ADD R12, R12, R16	0000018c	01906020
SNE R20, R10, R12	00000190	014ca029
BNEZ R20, -404	00000194	169ffe6c
ADD R1, R0, R10	00000198	000a0820
ADD R1, R0, R1	0000019c	00010820
ADD R2, R0, R2	000001a0	00021020
ADD R3, R0, R3	000001a4	00031820
ADD R4, R0, R4	000001a8	00042020

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โปรแกรมทดสอบทั้ง 5 โปรแกรมมีเปอร์เซ็นต์ของการเป็นคำสั่งทางแยกแบบมีเงื่อนไขและไม่มีเงื่อนไขแตกต่างกัน โดยที่โปรแกรมที่ 5(1) หมายถึงโปรแกรมทดสอบที่ 5 ที่พฤติกรรมของคำสั่งทางแยกที่เป็น Taken และ Not taken ไม่แตกต่างกันมากนัก แต่โปรแกรมที่ 5(2) หมายถึงโปรแกรมทดสอบที่ 5 ที่พฤติกรรมของคำสั่งทางแยกเป็น Not taken สูงๆ สรุปได้ดังกราฟรูปที่ 5.1



รูปที่ 5.1 เปอร์เซ็นต์ของคำสั่งทางแยกแบบมีเงื่อนไขและไม่มีเงื่อนไขของโปรแกรมทดสอบ

5.3 ผลการทดสอบ

5.3.1 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 1

1) ทดสอบด้วยหลักการเดิม ผลการจำลองการทำงานได้ผลดังรูปที่ 5.2 (a)

จากรูปที่ 5.2 (a) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

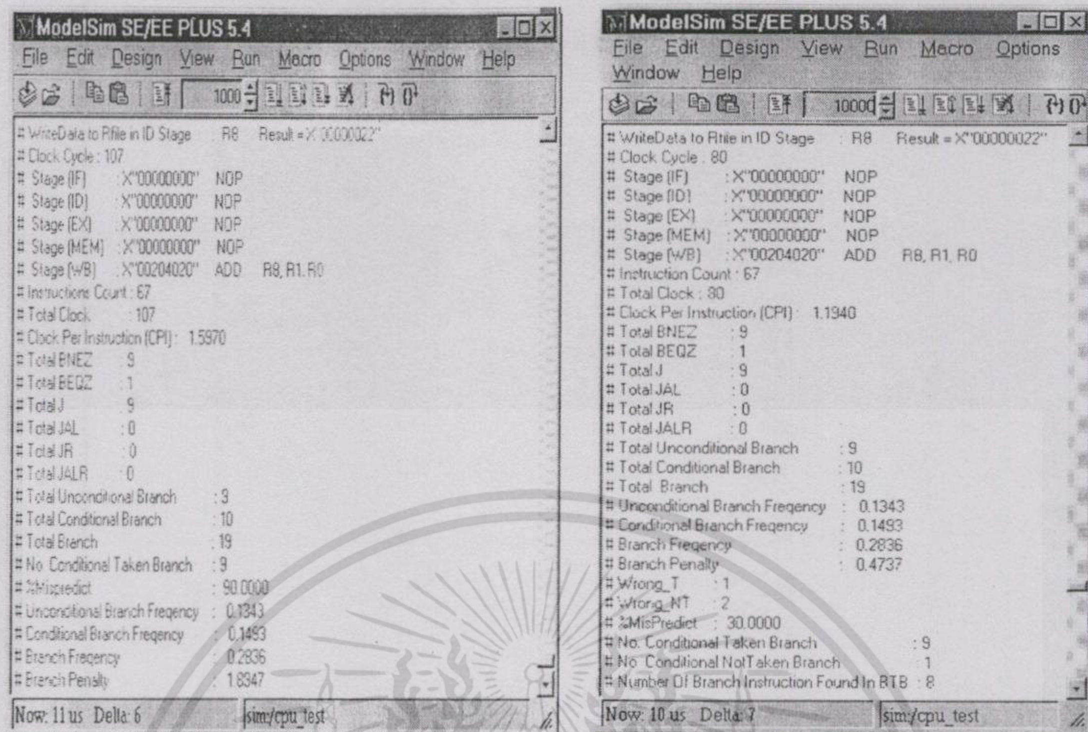
$$\text{CPI} = \frac{107}{67} = 1.5970$$

$$\text{Branch penalty} = (107-67-0-4) / 19 = 1.8947$$

$$\text{Speedup} = \frac{5}{1+\{(0.2836\text{x}1.8947)\}} = 3.2524$$

$$\text{\% Mispredict} = \frac{9}{10}\text{x}100 = 90\%$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.2 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 1

(a) หลักการเดิม (b) หลักการใหม่

2) ทดสอบด้วยหลักการใหม่ ผลการจำลองการทำงาน ได้ผลดังรูปที่ 5.2 (b) และจากรูปที่ 5.2 (b) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{80}{67} = 1.1940$$

$$\text{Branch penalty} = (80-67-0-4) / 19 = 0.4737$$

$$\text{Speedup} = \frac{5}{1+(0.2836\text{x}0.4737)} = 4.4097$$

$$\%\text{Mispredict} = \frac{1+2}{10}\text{x}100 = 30.0000\ \%$$

5.3.2 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 2

1) ทดสอบด้วยหลักการเดิม ผลการจำลองการทำงานได้ผลดังรูปที่ 5.3 (a)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
ModelSim SE/EE PLUS 5.4
File Edit Design View Run Macro Options Window Help
63000
# WriteData to Rfile in ID Stage   : R1   Result = X"00375F00"
# Clock Cycle : 616
# Stage (IF)    : X"00000000"   NOP
# Stage (ID)    : X"00000000"   NOP
# Stage (EX)    : X"00000000"   NOP
# Stage (MEM)   : X"00000000"   NOP
# Stage (WB)    : X"00600820"   ADD   R1, R3, R0
# Instructions Count : 370
# Total Clock      : 616
# Clock Per Instruction (CPI) : 1.6649
# Total BNEZ   : 65
# Total BEQZ   : 11
# Total J      : 65
# Total JAL    : 0
# Total JR     : 0
# Total JALR   : 0
# Total Unconditional Branch   : 65
# Total Conditional Branch     : 76
# Total Branch                 : 141
# No. Conditional Taken Branch : 56
# %Mispredict                  : 73.6842
# Unconditional Branch Freqency : 0.1757
# Conditional Branch Freqency   : 0.2054
# Branch Freqency               : 0.3811
# Branch Penalty                : 1.7163
Now: 63 us  Delta: 6    sim:/cpu_test
```

```
ModelSim SE/EE PLUS 5.4
File Edit Design View Run Macro Options
Window Help
10000
# WriteData to Rfile in ID Stage   : R1   Result = X"00375F00"
# Clock Cycle : 391
# Stage (IF)    : X"00000000"   NOP
# Stage (ID)    : X"00000000"   NOP
# Stage (EX)    : X"00000000"   NOP
# Stage (MEM)   : X"00000000"   NOP
# Stage (WB)    : X"00600820"   ADD   R1, R3, R0
# Instruction Count : 370
# Total Clock : 391
# Clock Per Instruction (CPI) : 1.0568
# Total BNEZ   : 65
# Total BEQZ   : 11
# Total J      : 65
# Total JAL    : 0
# Total JR     : 0
# Total JALR   : 0
# Total Unconditional Branch   : 65
# Total Conditional Branch     : 76
# Total Branch                 : 141
# Unconditional Branch Freqency : 0.1757
# Conditional Branch Freqency   : 0.2054
# Branch Freqency               : 0.3811
# Branch Penalty                : 0.1206
# Wrong_T    : 10
# Wrong_NT   : 2
# %MisPredict : 15.7895
# No. Conditional Taken Branch      : 56
# No. Conditional NotTaken Branch   : 20
# Number Of Branch Instruction Found In BTB : 64
Now: 40 us  Delta: 7    sim:/cpu_test
```

รูปที่ 5.3 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 2

(a) หลักการเดิม (b) หลักการใหม่

จากรูปที่ 5.3 (a) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{616}{370} = 1.6649$$

$$\text{Branch penalty} = (616-370-0-4) / 141 = 1.7163$$

$$\text{Speedup} = \frac{5}{1+(0.3811 \text{x} 1.7163)} = 3.0228$$

$$\text{\% Mispredict} = \frac{56}{76} \text{x} 100 = 73.6842 \text{ \%}$$

2) ทดสอบด้วยหลักการใหม่ผลการจำลองการทำงานได้ผลดังรูปที่ 5.3 (b) จากรูป 5.3 (b) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{391}{370} = 1.0568$$

$$\text{Branch penalty} = (391-370-0-4) / 141 = 0.1206$$

$$\text{Speedup} = \frac{5}{1+(0.3811 \text{x} 0.1206)} = 4.7803$$

$$\text{\%Mispredict} = \frac{10+2}{76} \text{x} 100 = 15.7895 \text{ \%}$$

5.3.3 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 3

1) ทดสอบด้วยหลักการเดิม ผลการจำลองการทำงานได้ผลดังรูปที่ 5.4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
ModelSim SE/EE PLUS 5.4
File Edit Design View Run Macro Options Window Help
# WriteData to Rfile in ID Stage    : R8   Result = X"0000005C"
# Clock Cycle : 327
# Stage (IF)    : X"00000000"   NOP
# Stage (ID)    : X"00000000"   NOP
# Stage (EX)    : X"00000000"   NOP
# Stage (MEM)   : X"00000000"   NOP
# Stage (WB)    : X"01004020"   ADD    R8, R8, R0
# Instructions Count : 235
# Total Clock    : 327
# Clock Per Instruction (CPI) : 1.3915
# Total BNEZ     : 14
# Total BEQZ     : 43
# Total J        : 7
# Total JAL      : 0
# Total JR       : 0
# Total JALR     : 0
# Total Unconditional Branch     : 7
# Total Conditional Branch       : 57
# Total Branch                   : 64
# No. Conditional Taken Branch   : 37
# %Mispredict                    : 64.9123
# Unconditional Branch Freqency  : 0.0298
# Conditional Branch Freqency    : 0.2426
# Branch Freqency                : 0.2723
# Branch Penalty                 : 1.3750
Now: 40 us  Delta: 6     sim:/cpu_test
```

```
ModelSim SE/EE PLUS 5.4
File Edit Design View Run Macro Options
Window Help
# WriteData to Rfile in ID Stage    : R8   Result = X"0000005C"
# Clock Cycle : 265
# Stage (IF)    : X"00000000"   NOP
# Stage (ID)    : X"00000000"   NOP
# Stage (EX)    : X"00000000"   NOP
# Stage (MEM)   : X"00000000"   NOP
# Stage (WB)    : X"01004020"   ADD    R8, R8, R0
# Instruction Count : 235
# Total Clock : 265
# Clock Per Instruction (CPI) : 1.1277
# Total BNEZ     : 14
# Total BEQZ     : 43
# Total J        : 7
# Total JAL      : 0
# Total JR       : 0
# Total JALR     : 0
# Total Unconditional Branch     : 7
# Total Conditional Branch       : 57
# Total Branch                   : 64
# Unconditional Branch Freqency  : 0.0298
# Conditional Branch Freqency    : 0.2426
# Branch Freqency                : 0.2723
# Branch Penalty                 : 0.4063
# Wrong_T      : 4
# Wrong_NT     : 13
# %MisPredict  : 29.8246
# No. Conditional Taken Branch       : 37
# No. Conditional NotTaken Branch    : 20
# Number Of Branch Instruction Found In BTB : 28
Now: 30 us  Delta: 7     sim:/cpu_test
```

รูปที่ 5.4 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 3

(a) หลักการเดิม (b) หลักการใหม่

จากรูปที่ 5.4 (a) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{327}{235} = 1.3915$$

$$\text{Branch penalty} = (327\text{-}235\text{-}0\text{-}4) / 64 = 1.3750$$

$$\text{Speedup} = \frac{5}{1+\{(0.2723\text{x}1.3750)\}} = 3.6379$$

$$\%\ \text{Mispredict} = \frac{37}{57}\text{x}100 = 64.9123\%$$

2) ทดสอบด้วยหลักการใหม่ ผลการจำลองการทำงานได้ผลดังรูปที่ 5.4 (b) จากรูปที่ 5.4 (b) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{265}{235} = 1.1277$$

$$\text{Branch penalty} = (265\text{-}235\text{-}0\text{-}4) / 64 = 0.4063$$

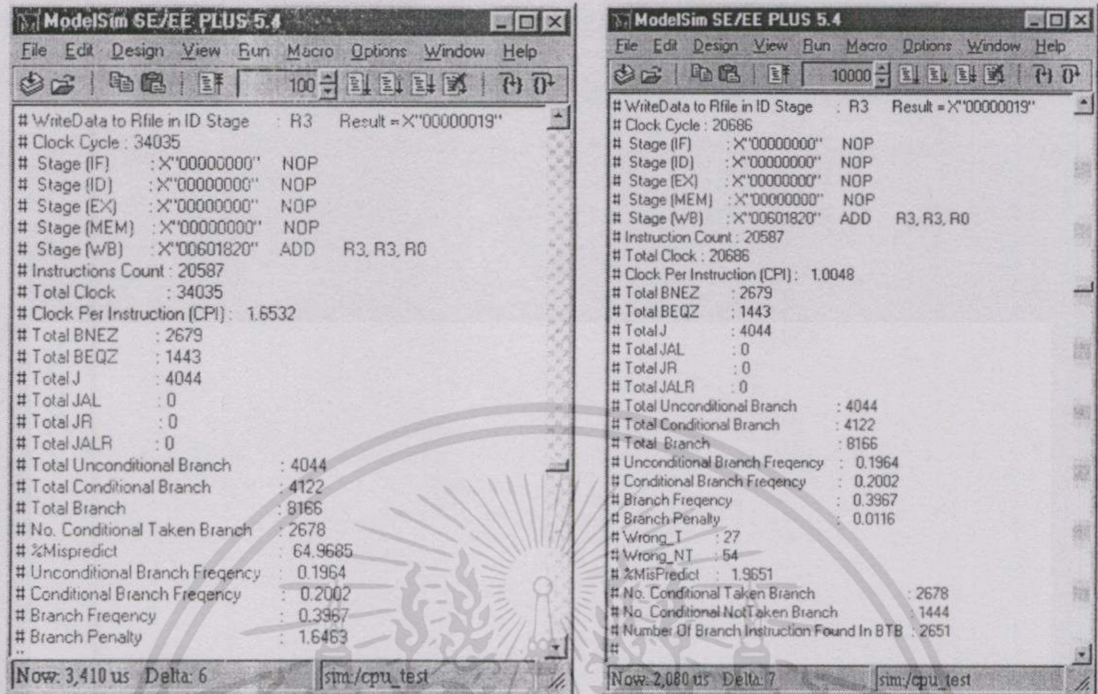
$$\text{Speedup} = \frac{5}{1+\{(0.2723\text{x}0.4063)\}} = 4.5019$$

$$\%\ \text{Mispredict} = \frac{4+13}{57}\text{x}100 = 29.8246\ \%$$

5.3.4 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1) ทดสอบด้วยหลักการเดิม ผลการจำลองการทำงานได้ผลดังรูปที่ 5.5 (a)



รูปที่ 5.5 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 4

(a) หลักการเดิม (b) หลักการใหม่

จากรูปที่ 5.5 (a) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์ มีค่าดังนี้

$$\text{CPI} = \frac{34035}{20587} = 1.6532$$

$$\text{Branch penalty} = (34035\text{-}20587\text{-}0\text{-}4) / 8166 = 1.6463$$

$$\text{Speedup} = \frac{5}{1+(0.3967\text{x}1.6463)} = 3.0246$$

$$\%\ \text{Mispredict} = \frac{2678}{4122}\text{x}100 = 64.9685\ \%$$

2) ทดสอบด้วยหลักการใหม่ ผลการจำลองการทำงานได้ผลดังรูปที่ 5.5 (b) จากรูปที่ 5.5 (b) ตัวแปรบอกประสิทธิภาพของโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{20686}{20587} = 1.0048$$

$$\text{Branch penalty} = (20686\text{-}20587\text{-}0\text{-}4) / 8166 = 0.0116$$

$$\text{Speedup} = \frac{5}{1+(0.3967\text{x}0.0116)} = 4.9771$$

$$\%\ \text{Mispredict} = \frac{27+54}{4122}\text{x}100 = 1.9651\ \%$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.3.5 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 5

1) ทดสอบด้วยหลักการเดิม ผลการจำลองการทำงานได้ผลดังรูปที่ 5.6 (a)

```
ModelSim SE/EE PLUS 5.4
File Edit Design View Run Macro Options Window Help
10000
# WriteData to Rfile in ID Stage    : R4    Result =X"00000003"
# Clock Cycle : 7774
#  Stage (IF)     : X"00000000"   NOP
#  Stage (ID)     : X"00000000"   NOP
#  Stage (EX)     : X"00000000"   NOP
#  Stage (MEM)    : X"00000000"   NOP
#  Stage (WB)     : X"00802020"   ADD     R4, R4, R0
# Instructions Count : 5194
# Total Clock        : 7774
# Clock Per Instruction (CPI) :  1.4967
# Total BNEZ      : 1004
# Total BEQZ      : 0
# Total J         : 813
# Total JAL       : 0
# Total JR        : 0
# Total JALR      : 0
# Total Unconditional Branch        : 813
# Total Conditional Branch          : 1004
# Total Branch                      : 1817
# No. Conditional Taken Branch      : 475
# %Mispredict                       : 47.3108
# Unconditional Branch Freqency     :  0.1565
# Conditional Branch Freqency       :  0.1933
# Branch Freqency                   :  0.3498
# Branch Penalty                    :  1.4177
Now: 780 us   Delta: 6        sim:/cpu_test
```

```
ModelSim SE/EE PLUS 5.4
File Edit Design View Run Macro Options
Window Help
100
# WriteData to Rfile in ID Stage    : R4    Result = X"00000003"
# Clock Cycle : 5458
#  Stage (IF)     : X"00000000"   NOP
#  Stage (ID)     : X"00000000"   NOP
#  Stage (EX)     : X"00000000"   NOP
#  Stage (MEM)    : X"00000000"   NOP
#  Stage (WB)     : X"00802020"   ADD     R4, R4, R0
# Instruction Count : 5194
# Total Clock : 5458
# Clock Per Instruction (CPI) :  1.0508
# Total BNEZ      : 1004
# Total BEQZ      : 0
# Total J         : 813
# Total JAL       : 0
# Total JR        : 0
# Total JALR      : 0
# Total Unconditional Branch        : 813
# Total Conditional Branch          : 1004
# Total Branch                      : 1817
# Unconditional Branch Freqency     :  0.1565
# Conditional Branch Freqency       :  0.1933
# Branch Freqency                   :  0.3498
# Branch Penalty                    :  0.1431
# Wrong_T     : 50
# Wrong_NT    : 171
# %MisPredict   : 22.0120
# No. Conditional Taken Branch          : 475
# No. Conditional NotTaken Branch       : 529
# Number Of Branch Instruction Found In BTB  : 354
Now: 546,100 ns   Delta: 7        sim:/cpu_test
```

รูปที่ 5.6 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 5

(a) หลักการเดิม (b) หลักการใหม่

จากรูปที่ 5.6 (a) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์ มีค่าดังนี้

$$\text{CPI} = \frac{7774}{5194} = 1.4967$$

$$\text{Branch penalty} = (7774-5194-0-4) / 1817 = 1.4177$$

$$\text{Speedup} = \frac{5}{1+(0.3498 \text{x} 1.4177)} = 3.3424$$

$$\text{\% Mispredict} = \frac{475}{1004} \text{x}100 = 47.3108\,\%$$

2) ทดสอบด้วยหลักการใหม่ ผลการจำลองการทำงานได้ผลดังรูปที่ 5.6 (b) จากรูปที่ 5.6 (b) ตัวแปรบอกประสิทธิภาพของโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{5458}{5194} = 1.0508$$

$$\text{Branch penalty} = (5458-5194-0-4) / 1817 = 0.1431$$

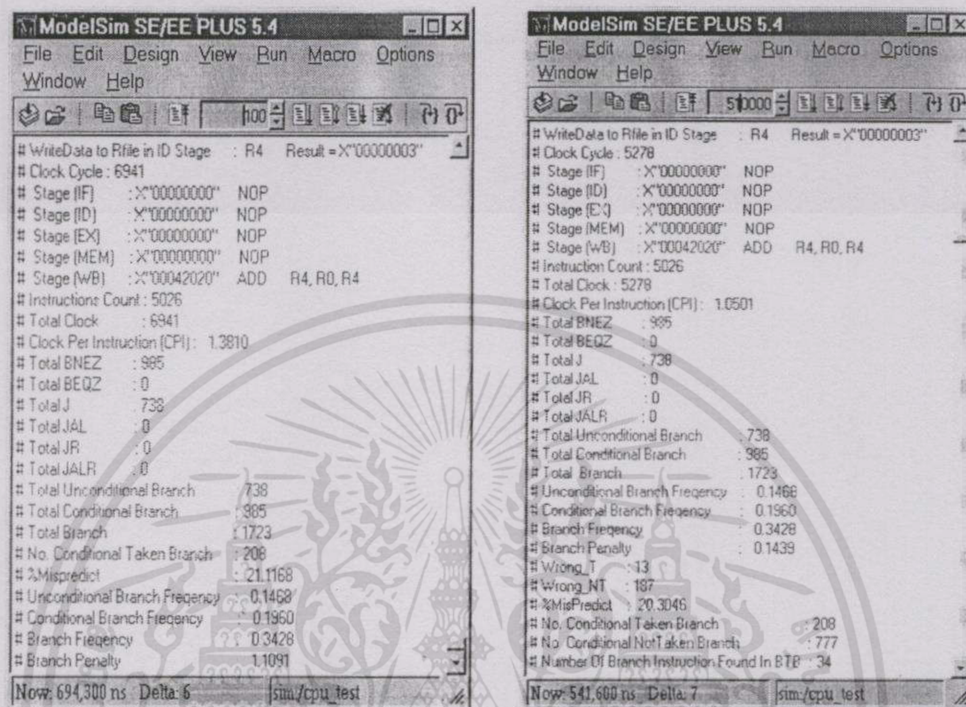
$$\text{Speedup} = \frac{5}{1+(0.3498 \text{x} 0.1431)} = 4.7616$$

$$\text{\% Mispredict} = \frac{50+171}{1004} \text{x}100 = 22.0120\,\%$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.3.6 ผลการทดสอบด้วยโปรแกรมทดสอบที่ 5 ที่มีพฤติกรรม Not taken สูง

1) ทดสอบด้วยหลักการเดิม ผลการจำลองการทำงานได้ผลดังรูปที่ 5.7 (a)



รูปที่ 5.7 ผลการจำลองการทำงานของโปรแกรมทดสอบที่ 5 ที่พฤติกรรม Not taken สูง

(a) หลักการเดิม (b) หลักการใหม่

จากรูปที่ 5.7 (a) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{6941}{5026} = 1.3810$$

$$\text{Branch penalty} = (6941-5026-0-4) / 1723 = 1.1091$$

$$\text{Speedup} = \frac{5}{1+(0.3428x0.1091)} = 3.6227$$

$$\text{\% Mispredict} = \frac{208}{985}x100 = 21.1168\,\%$$

2) ทดสอบด้วยหลักการใหม่ ผลการจำลองการทำงานได้ผลดังรูปที่ 5.7 (b)

จากรูปที่ 5.7 (b) ตัวแปรบอกประสิทธิภาพของไมโครโปรเซสเซอร์มีค่าดังนี้

$$\text{CPI} = \frac{5278}{5026} = 1.0501$$

$$\text{Branch penalty} = (5278-5026-0-4) / 1723 = 0.1439$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

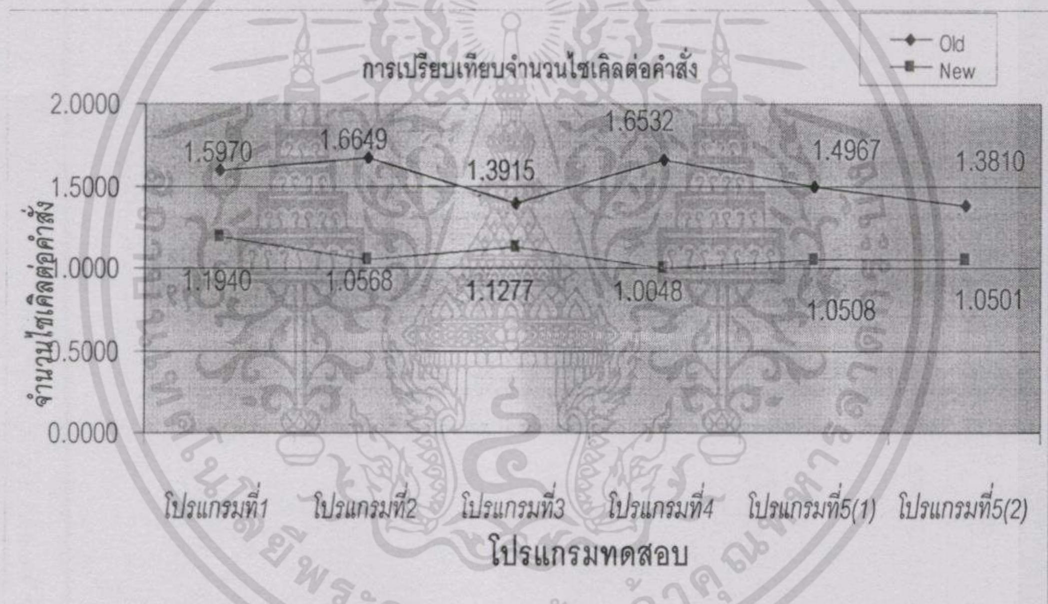
$$\text{Speedup} = \frac{5}{1+(0.3428x0.1439)} = 4.7650$$

$$\% \text{ Mispredict} = \frac{13+187}{985}x100 = 20.3046\ \%$$

5.4 เปรียบเทียบประสิทธิภาพ

การเปรียบเทียบประสิทธิภาพการทำงานของโปรเซสเซอร์ไปป์ไลน์เมื่อประมวลผลคำสั่งทางแยกด้วยหลักการเดิมและหลักการใหม่ แสดงได้ดังรูปที่ 5.8-5.11 ซึ่งเป็นกราฟความสัมพันธ์ระหว่างตัวแปรบอกประสิทธิภาพและโปรแกรมทดสอบภายใต้หลักการเดิมและหลักการใหม่

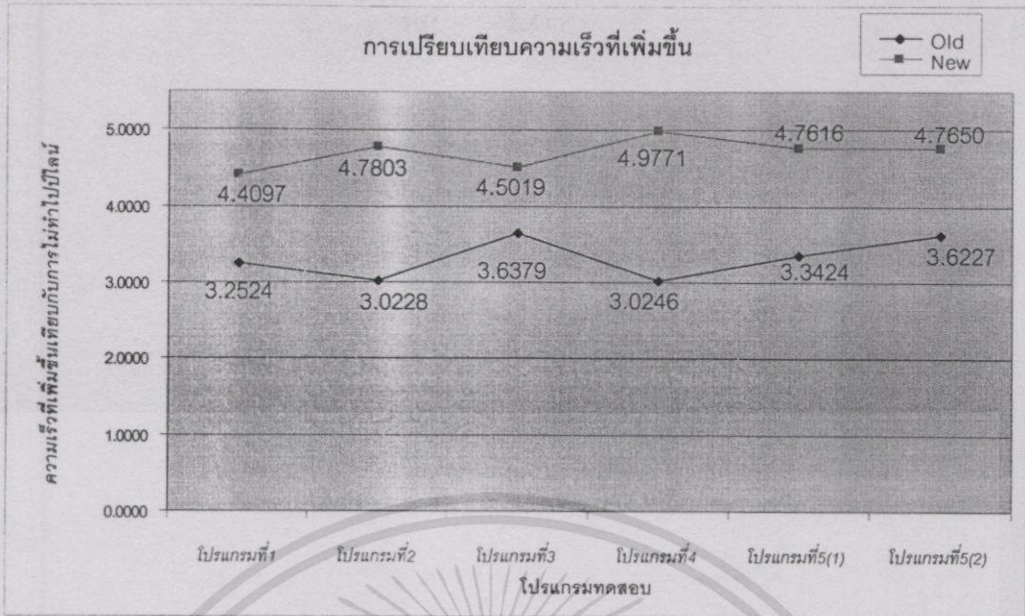
5.4.1 เปรียบเทียบ CPI



รูปที่ 5.8 การเปรียบเทียบ CPI ของโปรแกรมทดสอบทั้ง 5 โปรแกรมระหว่างหลักการเดิมและหลักการใหม่

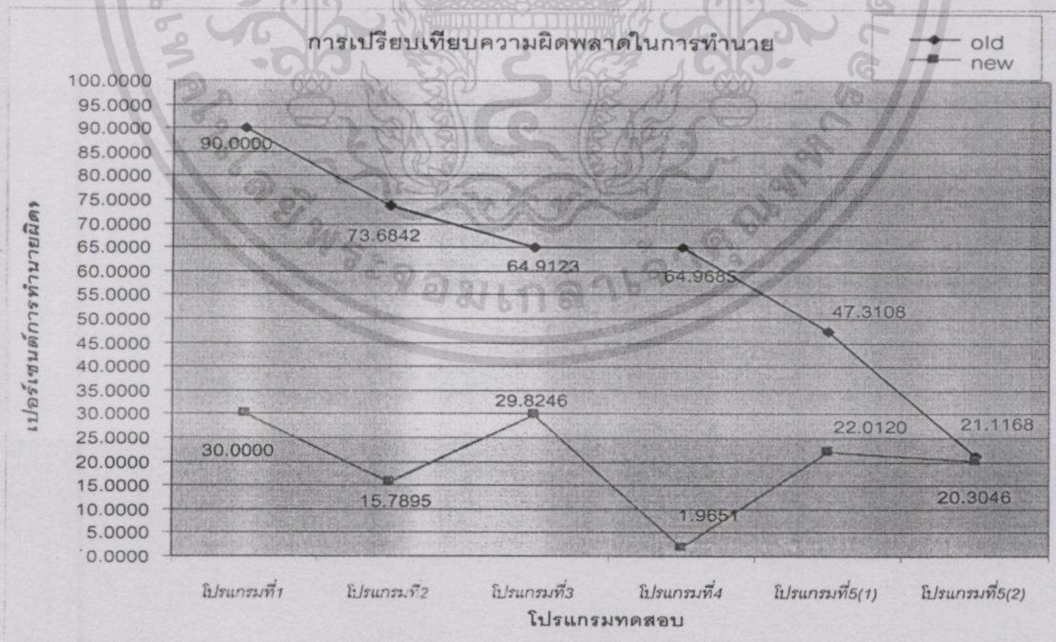
5.4.2 เปรียบเทียบ Speedup

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.9 การเปรียบเทียบ Speedup ของโปรแกรมทดสอบทั้ง 5 โปรแกรม ระหว่างหลักการเดิมและหลักการใหม่

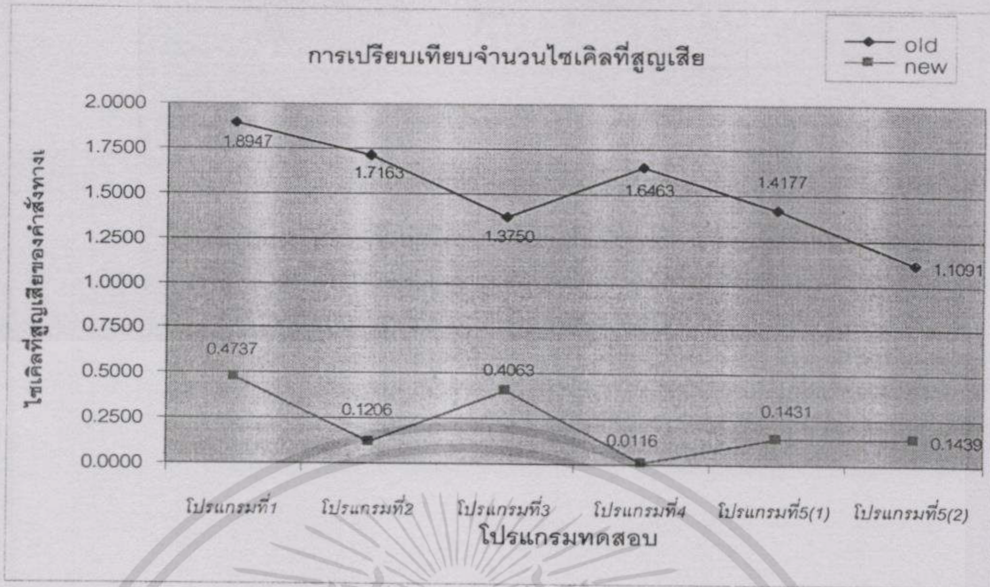
5.4.3 เปรียบเทียบ % Mispredict



รูปที่ 5.10 การเปรียบเทียบ %Mispredict ของโปรแกรมทดสอบทั้ง 5 โปรแกรมระหว่างหลักการเดิมและหลักการใหม่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

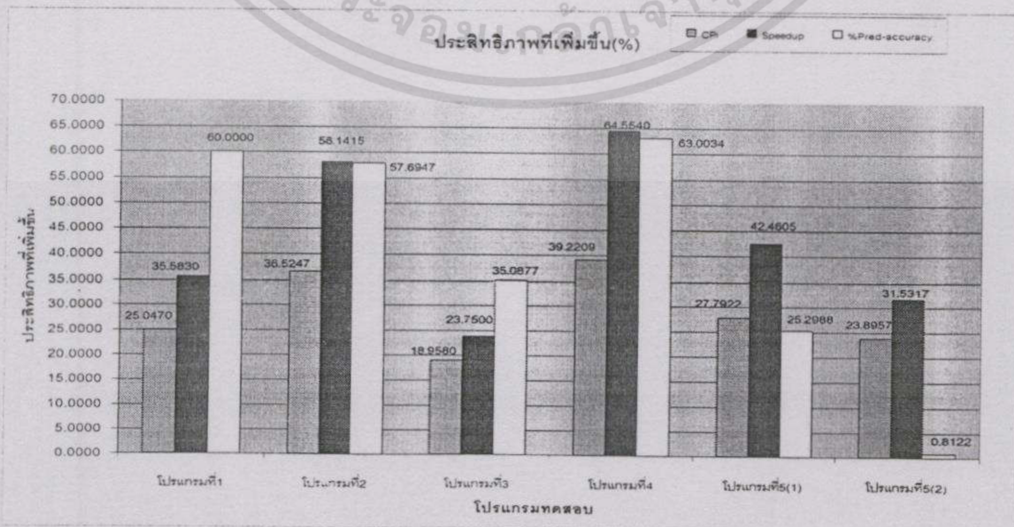
5.3.4 เปรียบเทียบ Branch penalty



รูปที่ 5.11 การเปรียบเทียบ Branch penalty ของโปรแกรมทดสอบทั้ง 5 โปรแกรมระหว่างหลักการเดิมและหลักการใหม่

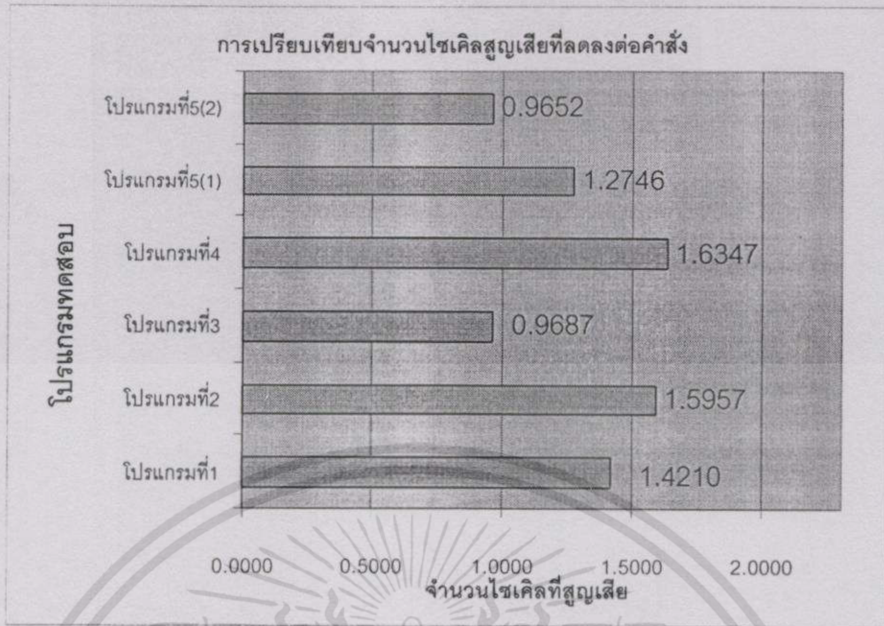
5.5 สรุปประสิทธิภาพของไมโครโปรเซสเซอร์

จากผลการทำงานของโปรเซสเซอร์ไปป์ไลน์เมื่อประมวลผลคำสั่งทางแยกภายใต้หลักการเดิมที่ทำนายผลลัพธ์ของคำสั่งทางแยกว่า Not taken เสมอกับหลักการใหม่ที่มีการปรับปรุงวิธีการทำนายผลลัพธ์ของคำสั่งทางแยกโดยใช้การวิธีการทำนายแบบ 2 บิท และมีการปรับปรุงโครงสร้างของดาต้าพาธ ผลการทดสอบการทำงานแสดงให้เห็นว่าประสิทธิภาพของโปรเซสเซอร์ดีขึ้น และจำนวนไซเคิลที่สูญเสียน้อยลง โดยรูปที่ 5.12-5.13 เป็นกราฟแสดงประสิทธิภาพของโปรเซสเซอร์



รูปที่ 5.12 ประสิทธิภาพที่เพิ่มขึ้นของโปรเซสเซอร์เมื่อทดสอบกับโปรแกรมทดสอบที่แตกต่างกัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.13 จำนวนไซเคิลที่สูญเสียเมื่อทดสอบกับโปรแกรมทดสอบที่แตกต่างกัน

นอกจากนี้เมื่อนำโปรแกรมทดสอบที่ 5 มาเขียนใหม่ให้โปรแกรมมีการประมวลผลคำสั่งทางแยกโดยพฤติกรรมของคำสั่งทางแยกเป็น Not taken สูงมากๆ เมื่อเทียบกับพฤติกรรม Taken จากนั้นนำผลที่ได้จากการทำงานมาเปรียบเทียบกับโปรแกรมทดสอบที่ 5 เดิมที่มีพฤติกรรมของคำสั่งทางแยกเป็น Not taken กับ Taken ไม่แตกต่างกันนัก ผลที่ได้ปรากฏว่าหลักการใหม่ที่นำเสนอยังคงทำให้ประสิทธิภาพในการประมวลผลคำสั่งทางแยกของไมโครโปรเซสเซอร์ดีกว่าหลักการเดิม ซึ่งสามารถแสดงให้เห็นชัดเจนยิ่งขึ้นดังกราฟที่นำเสนอข้างต้น จึงสรุปได้ว่าการประมวลผลคำสั่งทางแยกด้วยการทำนายผลลัพธ์ของคำสั่งทางแยกแบบไดนามิกวิธี 2 บิทสเตจแมชชีน ร่วมกับการจัดเก็บข้อมูลการทำนายไว้ในหน่วยความจำบีทีบีและบีเอชที ตลอดจนการปรับปรุงดาต้าพาธให้เหมาะสมทำให้ไมโครโปรเซสเซอร์มีประสิทธิภาพการประมวลผลคำสั่งทางแยกสูงขึ้นเมื่อเทียบกับหลักการเดิมซึ่งทำนายผลลัพธ์ของคำสั่งทางแยกว่า Not taken เสมอและมีการตรวจสอบคำสั่งทางแยกในสเตจ EX

ตารางที่ 5.7 เป็นตารางสรุปประสิทธิภาพที่เพิ่มขึ้นของไมโครโปรเซสเซอร์เมื่อทำการปรับปรุงการทำนายทางแยก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.7 สรุปประสิทธิภาพที่เพิ่มขึ้นของไมโครโปรเซสเซอร์เมื่อมีการปรับปรุงการทำนายทางแยก

โปรแกรมทดสอบ	วิธีการทดสอบ	CPI	CPI ที่ลดลง (%)	Speed up	ความเร็วที่เพิ่มขึ้น (%)	% Miss predict	% ความถูกต้องที่เพิ่มขึ้น	Branch penalty	จำนวนไซเคิลสูญเสียที่ลดลงต่อคำสั่ง
โปรแกรมที่ 1	วิธีเดิม	1.5970	**25.0470**	3.2524	**35.5830**	90.0000	**60.0000**	1.8947	**1.4210**
	วิธีใหม่	1.1940		4.4097		30.0000		0.4737	
โปรแกรมที่ 2	วิธีเดิม	1.6649	**36.5247**	3.0228	**58.1415**	73.6842	**57.8947**	1.7163	**1.5957**
	วิธีใหม่	1.0568		4.7803		15.7895		0.1206	
โปรแกรมที่ 3	วิธีเดิม	1.3915	**18.9580**	3.6379	**23.7500**	64.9123	**35.0877**	1.3750	**0.9687**
	วิธีใหม่	1.1277		4.5019		29.8246		0.4063	
โปรแกรมที่ 4	วิธีเดิม	1.6532	**39.2209**	3.0246	**64.5540**	64.9685	**63.0034**	1.6463	**1.6347**
	วิธีใหม่	1.0048		4.9771		1.9651		0.0116	
โปรแกรมที่ 5 (1)	วิธีเดิม	1.4967	**27.7922**	3.3424	**42.4605**	47.3108	**25.2987**	1.4177	**1.2746**
	วิธีใหม่	1.0508		4.7616		22.0120		0.1431	
โปรแกรมที่ 5 (2)	วิธีเดิม	1.3810	**23.8957**	3.6227	**31.5317**	21.1168	**0.8122**	1.1091	**0.9652**
	วิธีใหม่	1.0501		4.7650		20.3046		0.1439	

บทที่ 6
สรุปผลการวิจัยและบทวิจารณ์

6.1 สรุปผลการวิจัย

6.1.1 ผลการวิจัย

ปัญหาการควบคุมมีความสำคัญมากเพราะเป็นปัญหาสำคัญที่ทำให้ประสิทธิภาพของไปป์ไลน์คำสั่งลดลง ทุกครั้งที่คำสั่งทางแยกถูกนำเข้ามาประมวลผลไมโครโปรเซสเซอร์จะต้องตรวจสอบเงื่อนไขของคำสั่งทางแยก ถ้าคำสั่งทางแยกนั้นต้องกระโดดไปทำงานยังตำแหน่งเป้าหมาย ไมโครโปรเซสเซอร์ต้องคำนวณตำแหน่งเป้าหมายเพื่อให้การประมวลผลเกิดขึ้นที่ตำแหน่งเป้าหมายนั้น จากนั้นต้องยกเลิกคำสั่งที่ถูกเฟ็ทช์เข้ามาทำงานตามหลังคำสั่งทางแยก ทำให้มีการสูญเสียไซเคิลการทำงาน การลดจำนวนไซเคิลที่สูญเสียเนื่องจากคำสั่งทางแยกสามารถทำได้ 2 วิธีหลักๆคือ ใช้วิธีการทางคอมไพเลอร์หรือใช้วิธีการทำนายทางแยก ดังนี้

วิธีการทางคอมไพเลอร์ แบ่งเป็น

1) ดีเลย์บรานช์ ช่วยลดจำนวนไซเคิลที่สูญเสียเนื่องจากคำสั่งทางแยกโดยตรง ดีเลย์บรานช์เป็นวิธีการจัดเรียงคำสั่งให้เหมาะสมโดยนำคำสั่งที่ต้องถูกประมวลผลแน่นอนเมื่อคำสั่งทางแยกนั้นถูกประมวลผล โดยมีเงื่อนไขว่าคำสั่งนั้นจะต้องเป็นคำสั่งอิสระหรือไม่เป็นคำสั่งที่อาจก่อให้เกิดปัญหาการขึ้นต่อกันของข้อมูลได้ นำคำสั่งอิสระนั้นมาแทรกไว้ในดีเลย์สล็อทซึ่งจะถูกประมวลผลได้โดยไม่สูญเสียไซเคิล อย่างไรก็ตามวิธีนี้จะยุ่งยากขึ้นเมื่อความลึกของไปป์ไลน์มากขึ้นหรือไปป์ไลน์ซับซ้อนขึ้น ถ้ารู้ผลลัพธ์และตำแหน่งปลายทางของคำสั่งทางแยกได้เร็วเท่าใด การใช้วิธีดีเลย์บรานช์ก็จะมีความง่ายขึ้นเท่านั้น เนื่องจากการรู้ผลลัพธ์และตำแหน่งปลายทางของคำสั่งทางแยกได้ตั้งแต่ตอนต้นของกระบวนการไปป์ไลน์แสดงว่าจำนวนคำสั่งอิสระที่เลือกก็จะน้อยลงไปด้วย แต่ถ้ารู้ผลลัพธ์และตำแหน่งปลายทางของคำสั่งทางแยกได้ในสเตจท้ายๆ คำสั่งอิสระที่เลือกมาจะมีจำนวนมากขึ้น ซึ่งเป็นการยากในการหาคำสั่งอิสระจำนวนมากๆมาแทรกให้ทำงานจบในดีเลย์สล็อท

2) ลูปอันโรล ถ้าในโปรแกรมมีจำนวนคำสั่งทางแยกมาก โอกาสที่จะสูญเสียจำนวนไซเคิลก็จะมากขึ้น วิธีลูปอันโรลเป็นวิธีลดจำนวนคำสั่งทางแยกและโอเวอร์เฮดของลูปให้น้อยลง หลักการคือเมื่อพบคำสั่งทางแยกที่เป็นลูปจะสำเนา (Copy) ตัวลูปนั้นซ้ำๆกันไปเพื่อไม่ให้เกิดการวนลูป ถ้าสามารถสำเนาคำสั่งในตัวลูปได้เท่ากับจำนวนครั้งในการวนลูป ก็จะสามารถกำจัดคำสั่งทางแยกที่จะเกิดขึ้นในลูปได้ทั้งหมด ดังนั้นเมื่อกำจัดคำสั่งทางแยกซึ่งเป็นต้นเหตุได้ทั้งหมดการสูญเสียไซเคิลจึงไม่เกิดขึ้น แต่ในทางปฏิบัติไม่สามารถนำลูปใดๆมาทำลูปอันโรลเพื่อกำจัดคำสั่งทางแยกได้หมด เนื่องจากในช่วงการคอมไพล์โปรแกรมไม่สามารถรู้ค่าขอบเขตบน (Upper limit) ของลูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(จำนวนครั้งในการวนลูป) ได้ ฉนั้นคอมไพเลอร์จะต้องกำหนดจำนวนสำเนาของลูปที่เหมาะสมไว้ค่าหนึ่ง การทำลูปอันโรลจึงช่วยลดจำนวนคำสั่งทางแยกให้น้อยลง อย่างไรก็ตามการลดจำนวนคำสั่งทางแยกขณะประมวลผล (Runtime) จะมีผลให้จำนวนโค้ดของโปรแกรมที่เป็นภาษาแอสเซมบลียาวขึ้น จำนวนริจิสเตอร์ที่ต้องใช้งานมีมากขึ้นขึ้นกับจำนวนชุดของลูปที่สำเนา

3) สแตติกสเคจดูลลิ่ง เป็นวิธีการที่ช่วยในการจัดเรียงคำสั่งเพื่อลดปัญหาการขึ้นต่อกันของข้อมูล วิธีนี้ถึงแม้จะไม่ได้แก้ปัญหาเนื่องจากคำสั่งทางแยกโดยตรงแต่จะช่วยจัดเรียงคำสั่งหลังการทำลูปอันโรล การทำลูปอันโรลอาจจะทำให้มีปัญหาการขึ้นต่อกันของข้อมูลได้ การจัดเรียงคำสั่งให้เหมาะสมจึงเป็นเรื่องจำเป็น

วิธีการทำนายทางแยก แบ่งเป็น

1) วิธีการทำนายแบบสแตติก จะกำหนดพฤติกรรมการทำนายไว้ล่วงหน้าแน่นอนตายตัวว่าคำสั่งทางแยกมีพฤติกรรมเป็น Taken หรือ Not taken วิธีการทำนายแบบสแตติกจะใช้วิธีทางซอฟต์แวร์ร่วมกับวิธีทางฮาร์ดแวร์ โดยถ้าสมมติกำหนดการทำนายล่วงหน้าว่า Not taken คอมไพเลอร์จะแปลงลูปของคำสั่งให้มีพฤติกรรมแบบ Not taken มากที่สุดเท่าที่จะทำได้ เมื่อได้แมชชีนโค้ดแล้วนำไปประมวลผลบนฮาร์ดแวร์ ถ้าคำสั่งทางแยกนั้นมีพฤติกรรม Taken ฮาร์ดแวร์จะทำหน้าที่ยกเลิกคำสั่งหลังคำสั่งทางแยกทันที ทำให้มีการสูญเสียไซเคิลการทำงานในกรณีทำนายผิดได้ แต่จำนวนไซเคิลที่สูญเสียจะน้อยเนื่องจากโปรแกรมมีพฤติกรรม Not taken สูง

2) วิธีการทำนายแบบไดนามิก เป็นวิธีทางฮาร์ดแวร์อย่างเดียวโดยจะใช้พฤติกรรมในอดีตของตัวมันเองหรือพฤติกรรมของคำสั่งทางแยกอื่นมากำหนดพฤติกรรมของคำสั่งทางแยกปัจจุบัน ผลการทำนายจะปรับเปลี่ยนไปตลอดขณะโปรแกรมทำงาน ถ้าผลการทำนายถูกต้องจะไม่มีการสูญเสียจำนวนไซเคิล การทำนายผิดจะสูญเสียไซเคิล แต่จากงานวิจัยต่างๆรวมทั้งงานวิจัยนี้พบว่าโอกาสในการทำนายถูกต้องมีสูงกว่าการทำนายผิด ทำให้โอกาสการสูญเสียไซเคิลลดน้อยลง ซึ่งแปรผันตรงกับประสิทธิภาพในการทำนาย วิธีการทำนายแบบไดนามิกจะให้ประสิทธิภาพดีกว่าวิธีการทำนายแบบสแตติก

สำหรับการทำนายผลลัพธ์ของคำสั่งทางแยกทั้ง 2 วิธีนั้น วิธีการทำนายแบบสแตติกเน้นที่ความสามารถของคอมไพเลอร์ ค่าใช้จ่ายที่สูญเสียเนื่องจากวิธีการทำนายแบบสแตติกจะอยู่ที่การสร้างคอมไพเลอร์ให้มีประสิทธิภาพ แต่ฮาร์ดแวร์สร้างไม่ยากแค่เพียงยกเลิกคำสั่งหลังคำสั่งทางแยก แต่วิธีการทำนายแบบไดนามิกนั้นคอมไพเลอร์ไม่ต้องซับซ้อนมาก งานหนักทุกอย่างจะตกอยู่กับฮาร์ดแวร์ ค่าใช้จ่ายที่สูญเสียก็คือราคาของฮาร์ดแวร์เป็นหลัก

จากวิธีการลดจำนวนไซเคิลที่สูญเสียเนื่องจากคำสั่งทางแยกแต่ละวิธีที่กล่าวข้างต้นมีข้อดีข้อเสียต่างกัน ในทางปฏิบัติการแก้ปัญหาเนื่องจากการประมวลผลคำสั่งทางแยกของโปรเซสเซอร์จะต้องใช้วิธีทางคอมไพเลอร์และวิธีการทำนายทางแยกผสมผสานกันเพื่อให้โปรเซสเซอร์มีประสิทธิ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาพสูงสุดไม่สามารถใช้วิธีใดวิธีหนึ่งแก้ปัญหาได้ครบถ้วน เช่น เราสามารถแก้ปัญหาของคำสั่งทางแยกเบื้องต้นด้วยการทำลูปอันโรลเพื่อลดจำนวนคำสั่งทางแยก หลังจากนั้นทำสเคจดูลลิง เพื่อได้แมชชีนโค้ดไปประมวลผลบนฮาร์ดแวร์ เนื่องจากแมชชีนโค้ดเหล่านั้นอาจยังมีคำสั่งทางแยกอยู่ ทั้งนี้เพื่อให้เกิดประสิทธิภาพยิ่งขึ้นฮาร์ดแวร์ควรสร้างเป็นแบบการทำนายทางแยกแบบไดนามิก หรือเราอาจใช้วิธีทางการทำนายทางแยกแบบสแตติก แปลงโค้ดให้มีลักษณะเป็น Taken หรือ Not taken อย่างใดอย่างหนึ่งให้มาก หลังจากนั้นนำโค้ดมาทำลูปอันโรล จากนั้นทำสเคจดูลลิง สำหรับฮาร์ดแวร์ก็มีเฉพาะส่วนการยกเลิกข้อมูลของคำสั่งทางแยกเมื่อเกิดการทำนายผิดเท่านั้น

ดังนั้นในขบวนการออกแบบโปนเซสเซอร์เพื่อลดการสูญเสียไซเคิลเนื่องจากคำสั่งทางแยกเพื่อให้เกิดประสิทธิภาพมากที่สุดจะต้องวางแผนการออกแบบทางซอฟต์แวร์และฮาร์ดแวร์ควบคู่กันไป ซึ่งต้องคำนึงถึงปัจจัยต่างๆ เช่นค่าใช้จ่าย หรือความสามารถของวิศวกรรผู้ออกแบบด้วย

6.1.2 ประโยชน์ที่ได้รับ

ประโยชน์ที่ได้รับจากการทำงานวิจัยนี้ สรุปเป็นหัวข้อใหญ่ๆ ได้คือ

1. ได้รับความรู้และการแก้ปัญหาต่างๆ ที่เกี่ยวข้องกับงานวิจัย
2. สามรถนำความรู้ที่ได้ไปประยุกต์แก้ปัญหากับไมโครโปรเซสเซอร์อื่นๆ ได้

6.1.3 อุปสรรคและปัญหา

จากการทำงานวิจัยตั้งแต่เริ่มต้นจนกระทั่งถึงขั้นตอนสุดท้ายพบว่าต้องเผชิญกับปัญหาและอุปสรรคมากมาย แต่ละขั้นตอนต้องมีการแก้ปัญหาบางอย่างเสมอๆ บางครั้งการแก้ปัญหาที่ลุล่วงไปแล้วในขั้นตอนที่ผ่านมากลับเป็นปัญหาของขั้นตอนการทำงานต่อไป สามารถแบ่งปัญหาและอุปสรรคที่เกิดขึ้นออกเป็นแต่ละส่วน ดังนี้

1) อุปกรณ์และเครื่องมือที่ใช้ในงานวิจัย

- ไมโครโปรเซสเซอร์เดอลุกซ์ ซึ่งเป็นไมโครเซสเซอร์ที่ยกมาเป็นต้นแบบในงานวิจัย ไมโครโปรเซสเซอร์เดอลุกซ์ได้รับความนิยมมากเนื่องจากโครงสร้างไม่ซับซ้อน สามารถเข้าใจได้ง่ายจึงมีผู้นำไปเป็นต้นแบบและพัฒนาต่อมากมาย บางครั้งจึงไม่สามารถสรุปได้ว่าอันไหนคือปัจจุบันที่สุด แต่ไม่ว่าจะพัฒนาไปถึงระดับไหนก็ตาม สิ่งที่ผู้พัฒนาทุกคนคิดและปฏิบัติเหมือนกันก็คือบอกแต่เพียงหลักการที่ตนเองนำเสนอ แต่เทคนิควิธีการอิมพลีเมนท์จะไม่เปิดเผย ซึ่งเป็นอุปสรรคอย่างมากต่องานวิจัย เนื่องจากงานวิจัยนี้มีจุดมุ่งหมายในการแก้ปัญหาการควบคุมในไปป์ไลน์ ซึ่งหมายถึงว่าจะต้องมีโปรเซสเซอร์ที่ทำไปป์ไลน์คำสั่งรองรับเรียบร้อยแล้ว พร้อมที่จะนำโปรเซสเซอร์ไปแก้ปัญหาต่างๆตามวัตถุประสงค์ของการวิจัยได้ทุกเมื่อ อย่างไรก็ตามเพื่อให้งานวิจัยดำเนินต่อไปได้ ผู้วิจัยจึงต้องพยายามสร้างไมโครโปรเซสเซอร์เดอลุกซ์ต้นแบบขึ้นมาก่อนโดยยึดหลักการของผู้ที่ให้กำเนิดไมโครโปรเซสเซอร์เดอลุกซ์เดิม หลังจากนั้นจึงนำไมโครโปรเซสเซอร์เดอลุกซ์นั้นไปแก้ปัญหาไปป์ไลน์ตามจุดมุ่งหมายของงานวิจัยที่นำเสนอ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- โปรแกรมทดสอบ (Benchmark) การจะวัดประสิทธิภาพของไมโครโปรเซสเซอร์ที่ออกแบบให้ได้ผลเป็นที่ยอมรับและมีมาตรฐานนั้น จะต้องมีโปรแกรมทดสอบที่ยอมรับได้สำหรับทดสอบการทำงานของไมโครโปรเซสเซอร์ แต่ปัญหาที่พบคือ ไม่สามารถหาโปรแกรมทดสอบที่มีคุณสมบัติตามต้องการนั้นได้ไม่ว่าจากสื่อใดๆก็ตาม เนื่องจากการสงวนลิขสิทธิ์ นอกจากนี้โปรแกรมทดสอบยังต้องอยู่ในรูปของแมชชีนโค้ดของโปรเซสเซอร์เดอลุกซ์ อาทิ โปรแกรมทดสอบในกลุ่ม SPEC 95 ดังนั้นโปรแกรมทดสอบที่ใช้ในงานวิจัยจึงได้มาจากการนำหลักการคำนวณทางคณิตศาสตร์แบบต่างๆ ซึ่งเป็นที่รู้จักและคุ้นเคยมาเขียนขึ้นเองในรูปแบบของภาษาระดับสูง โดยเนื้อหาของโปรแกรมทดสอบจะเกี่ยวข้องกับส่วนที่ไมโครโปรเซสเซอร์ที่ออกแบบต้องการทดสอบ

- หนังสือสำหรับค้นคว้าเพิ่มเติม พบว่าหนังสือที่เกี่ยวข้องกับการออกแบบไมโครโปรเซสเซอร์ไปป์ไลน์ขั้นสูงมีน้อยมาก ที่มีอยู่จะเป็นการแนะนำหลักการเบื้องต้น แต่เมื่อต้องนำไปอิมพลีเมนท์จริงจะต้องแก้ปัญหาที่เกิดขึ้นมาก มีหนังสือบางเล่มที่บอกถึงเทคนิควิธีการอิมพลีเมนท์จริงซึ่งจะมีขายในต่างประเทศแต่มีราคาสูงมาก ฉนั้นในการทำวิจัยบางครั้งจึงต้องมีการลองผิดลองถูกจนกว่าจะได้วิธีการที่ถูกต้องและไม่เข้าใจผิด

- เครื่องคอมพิวเตอร์ส่วนบุคคล (PC) ที่ใช้ทำงานวิจัยมีสเป็กค่อนข้างไม่เหมาะสม บางครั้งทำงานได้ช้า

6.2 บทวิจารณ์

เนื่องจากผลที่ได้รับจากการทำงานวิจัยนี้ก่อให้เกิดประโยชน์มากมายต่อผู้วิจัย ทั้งในด้านความรู้และวิธีการแก้ปัญหา จึงไม่มีข้อวิจารณ์ใดๆ มีเพียงข้อสังเกตบางประการที่ต้องการเสนอแนะ ซึ่งอาจจะมีประโยชน์ต่อผู้ต้องการทำงานวิจัยต่อไปบ้าง ดังนี้

1. งานวิจัยที่เกี่ยวข้องการออกแบบไมโครโปรเซสเซอร์ ควรจะกระทำกันเป็นกลุ่ม ถ้ากลุ่มผู้วิจัยได้ร่วมมือกันออกแบบไมโครโปรเซสเซอร์ต้นแบบไว้แล้ว การนำไมโครโปรเซสเซอร์ต้นแบบนี้ไปพัฒนาต่อในเรื่องอื่นๆ ที่เกี่ยวข้องจะกระทำได้อย่างกว้างขวาง การมีโอกาสถกเถียงอภิปราย (Discus) ถึงปัญหาของงานวิจัยระหว่างผู้วิจัยในกลุ่มจะทำให้ปัญหาและอุปสรรคต่างๆที่ได้กล่าวข้างต้นลดน้อยลง และได้รับประโยชน์สูงสุดในการทำวิจัย

2. ควรมีทุนสำหรับงานวิจัยที่ชัดเจน งานวิจัยที่เกี่ยวข้องกับการออกแบบไมโครโปรเซสเซอร์จำเป็นต้องใช้เครื่องมือที่มีราคาสูง อาทิ ซอฟท์แวร์ที่ใช้ในการจำลองการทำงาน (Simulate) หรือเครื่องคอมพิวเตอร์ที่เหมาะสมกับงาน บางครั้งอุปสรรคของงานวิจัยส่วนหนึ่งมาจากการขาดทุนวิจัย

3. อย่างไรก็ตามผู้วิจัยยังเห็นว่า สิ่งที่สำคัญที่สุดของการทำงานวิจัยคือการมีความรู้พื้นฐานในเรื่องที่จะทำอย่างเพียงพอ การทำงานวิจัยใดๆที่มีจุดมุ่งหมายจะพัฒนาสิ่งที่มีอยู่เดิม โดยจะ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ถือว่าสิ่งที่ทำขึ้นนั้นเป็นของใหม่และดีกว่าของเดิมนั้นจะต้องถูกพิสูจน์ให้เห็นจริงว่าสิ่งนั้นมีคุณค่ากว่าสิ่งเดิม การทำสิ่งใหม่ใดๆ โดยที่ไม่ได้ประโยชน์จากสิ่งนั้นมากไปกว่าสิ่งเดิมจะยังไม่ถือว่าบรรลุจุดมุ่งหมายของการทำงานวิจัย เป็นเพียงการนำของเก่ามาขัดเกลาและเขียนขึ้นใหม่ให้ถ้อยคำต่างกันเท่านั้นแต่ได้ผลลัพธ์เหมือนเดิม ซึ่งไม่มีประโยชน์ใดๆเลย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารอ้างอิง

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

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

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

[4] Hilgendorf, R.B. et.al. 1999. "Evaluation of branch-prediction methods on traces from commercial applications." **IBM Journal Development.** 43(4) : 579-592.

[5] Buford, M.G. and Haggard, R.G. 1996. "High Performance Branch Prediction." **IEEE Trans. On Computers.** n. d.

[6] http://www.usafa.af.mil/

[7] Perlrberg, C.H. and Smith, A.J. 1993. "Branch Target Buffer Design and Optimization." **IEEE Trans. On Computers.** 42(4) : 396-412.

[8] Fagin, B. 1997. "Partial Resolution in Branch Target Buffer." **IEEE trans. On Computers.** 46(17) : 1142-1145.

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

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

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

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

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ผลงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์และได้รับการตีพิมพ์

พัชรินทร์ กลิ่นซ้อน และคณะ. 2544. "การเปรียบเทียบประสิทธิภาพการทำนายผลลัพธ์ของคำสั่งทางแยกแบบไดนามิก." หน้า 84-89. วิศวสารลาดกระบัง. 18(2).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประวัติผู้เขียน

นางสาว พัชรินทร์ กลิ่นซ้อน สำเร็จการศึกษาระดับอุดมศึกษาจากมหาวิทยาลัยบูรพา คณะวิทยาศาสตร์ สาขาฟิสิกส์ (Pure physics)

ประวัติการทำงาน

ปัจจุบันทำงานตำแหน่งอาจารย์ประจำสาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยหัวเฉียวเฉลิมพระเกียรติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้