บทที่ 5
วิธีการตรวจสอบและทดลอง

ในบทนี้จะกล่าวถึงวิธีการที่ใช้ในการตรวจสอบการทำงานของ LSU ที่ได้ถูกออกแบบ จากนั้นจะกล่าวถึงวิธีการที่ใช้ในทดลองวัดสมรรถนะการทำงาน โดยจะกล่าวถึงวิธีการที่ใช้สร้าง Trace (ลำดับของคำสั่งที่ใช้สำหรับวัดสมรรถนะไมโครโพรเซสเซอร์) และความพยายามในการลดขนาดของ Trace เพื่อเพิ่มความเร็วในการทำงาน

5.1 วิธีตรวจสอบการทำงานของวงจร LSU

การตรวจสอบความถูกต้องในการทำงานของวงจร LSU มีขั้นตอนดังต่อไปนี้

1) จำลองการทำงานของวงจร LSU ด้วยโปรแกรมภาษาซี
2) กำหนดหน่วยความจำเริ่มต้นที่ใช้ในการทดสอบ
3) ผลการทดสอบความถูกต้องของการทำงานของวงจร

5.1.1 จำลองการทำงานของวงจร LSU ด้วยโปรแกรมภาษาซี

ผู้วิจัยได้จำลองการทำงานของวงจร LSU ด้วยโปรแกรมภาษาซีเพื่อตรวจสอบความถูกต้องของวงจรที่ออกแบบ โดยการนำผลการทำงานที่ได้จากการจำลองการทำงาน LSU ด้วยภาษาซีไปเปรียบเทียบกับผลการทำงานกับวงจร LSU ที่เขียนด้วยภาษา Verilog ว่าตรงกันหรือไม่ แต่เพื่อเพิ่มความมั่นใจในการทำงาน ผู้วิจัยจึงได้ออกแบบให้วงจร LSU ที่เขียนด้วยภาษาซีนั้นทำงานเป็น Sequential ซึ่งต่างกับการทำงานของวงจร LSU ที่เขียนด้วยภาษา Verilog ที่ทำงานเป็น Pipeline

วงจร LSU ที่เขียนด้วยภาษาซี มีการทำงานดังแสดงใน Flowchart รูปที่ 5.1 การประมวลผลจะทำทีละ 1 คำสั่ง การทำงานเริ่มต้นที่ Fetch Instruction จากนั้นตรวจสอบว่าคำสั่งที่ได้รับมานั้นมีค่าเท่ากับ 0x0A ถ้าใช่ก็ให้หยุด แต่ถ้าไม่ใช่ก็จะตรวจสอบว่าเป็นคำสั่งที่สามารถประมวลผลได้ใน LSU หรือไม่ ถ้าหากว่าไม่สามารถประมวลผลที่ LSU ได้ ก็จะทำการ fetch Instruction ใหม่ แต่ถ้าหากประมวลผลที่ LSU ได้ก็จะตรวจสอบต่อไปว่า เป็นคำสั่งใดในชุดคำสั่ง LOAD, STORE หรือคำสั่ง ADD โดยแต่ละคำสั่งมีการเก็บ Output Vectore เมื่อมีการเขียนข้อมูลลงในรีจีสเตอร์ (GPR), D-Cache Tag, D-Cache Data และหน่วยความจำ (MEM) โดยแต่ละคำสั่งที่ประมวลผลจะมีการเก็บค่า Output Vector ดังนี้

1. คำสั่ง ADD เก็บผลว่ามีการเขียนค่าลงในรีจีสเตอร์ที่ตำแหน่งใด และข้อมูลที่เขียนเป็นเท่าไร
2. ชุดคำสั่ง LOAD จะมีการสร้าง Output Vector ใน 2 ลักษณะขึ้นอยู่กับว่า Hit หรือ Miss
   - ถ้า LOAD HIT ก็จะเก็บผลของรีจีสเตอร์และข้อมูลที่เขียน
   - ถ้า LOAD MISS โปรแกรมจำเป็นต้องตรวจสอบค่า WB bit ในบรรทัดของแคชที่จะถูกแทนที่ก่อน โดยถ้าค่า WB bit = '1' จะมี Output Vector ของการเขียนข้อมูลจากแคชลงในหน่วย

ความจำ แต่ถ้าค่า WB bit = '0' จะไม่มีการเก็บค่า Output Vector ในส่วนนี้ จากนั้นจะมีการเขียนข้อมูลลงใน D-Cache Tag / Data และนำข้อมูลที่ต้องการโหลดเขียนลงในรีจีสเตอร์

3. ชุดคำสั่ง STORE จะมีการสร้าง Output Vector ใน 2 ลักษณะขึ้นอยู่กับว่า Hit หรือ Miss
   - ถ้าเกิด STORE HIT โปรแกรมจะเก็บผลของการเขียนข้อมูลลง D-Cache Data และถ้าหาก WB bit ใน D-Cache Tag มีค่าเป็น '0' ก่อนการประมวลผลคำสั่ง STORE ต้องตั้งค่าให้เป็น 1' ด้วย และสำหรับโหมดการเขียนที่เป็น Write Through จะเพิ่มในส่วนของการเขียนข้อมูลลงในหน่วยความจำหลักด้วย
   - ถ้าเกิด STORE MISS โปรแกรมเก็บผลของการเขียนข้อมูลลงในหน่วยความจำ

เมื่อเสร็จสิ้นการทำงานในชุดคำสั่ง LOAD, STORE หรือคำสั่ง ADD แล้วก็จะทำ Fetch Instruction ใหม่เข้ามาประมวลผลต่อ ทำจนกระทั่งสิ้นสุดการทำงาน (พบคำสั่ง 0x0A)

จากการทำงานของโปรแกรมภาษาซี เราจะได้ Output Vector ของการเขียนข้อมูลลง GPR, D-Cache Tag, D-Cache Data และหน่วยความจำ (MEM) โดยจะมีการเก็บข้อมูลในลักษณะดังต่อไปนี้

1. write GPR มีการเขียนข้อมูลลงใน GPR ตัวอย่างของ Output Vector ดังกล่าวคือ

   "Wr GPR[0d]=0a030a03"

   หมายถึง มีการเขียนข้อมูล 0a030a03 ลงในรีจีสเตอร์ตัวที่ 13 เป็นต้น

2. write DCT มีการเขียนข้อมูลลงใน D-Cache Tag 2 ลักษณะ ตัวอย่างของ Output Vector ดังกล่าวคือ

   "Wr TagDCache1[80] = 000052" และ

   "WB TagDCache1[80] = 000053"

   ตัวอย่างในบรรทัดแรกหมายถึงเขียนข้อมูล 000052 ลงใน D-Cache Tag Set 1 ที่ตำแหน่ง 80 โดยข้อมูล Tag ในที่นี้คือ 000014 (ขนาด 22 บิต) ,มีค่า WB bit = '0' และมีค่า Valid bit = '1' และตัวอย่างในบรรทัดต่อมาเป็นการตั้งค่า WB bit ที่ D-Cache Tag Set 1 ที่ตำแหน่ง 80 จึงมีข้อมูลเหมือนเดิมใน Tag และ Valid bit เหมือนเดิม แต่ค่า WB bit = '1'

3. write DCD มีการเขียนข้อมูลลงใน D-Cache Data ตัวอย่างของ Output Vector ดังกล่าวคือ

   "Wr DCD1Write=ff, DCDData=0a030a030a030a03, DCDAddr=203"

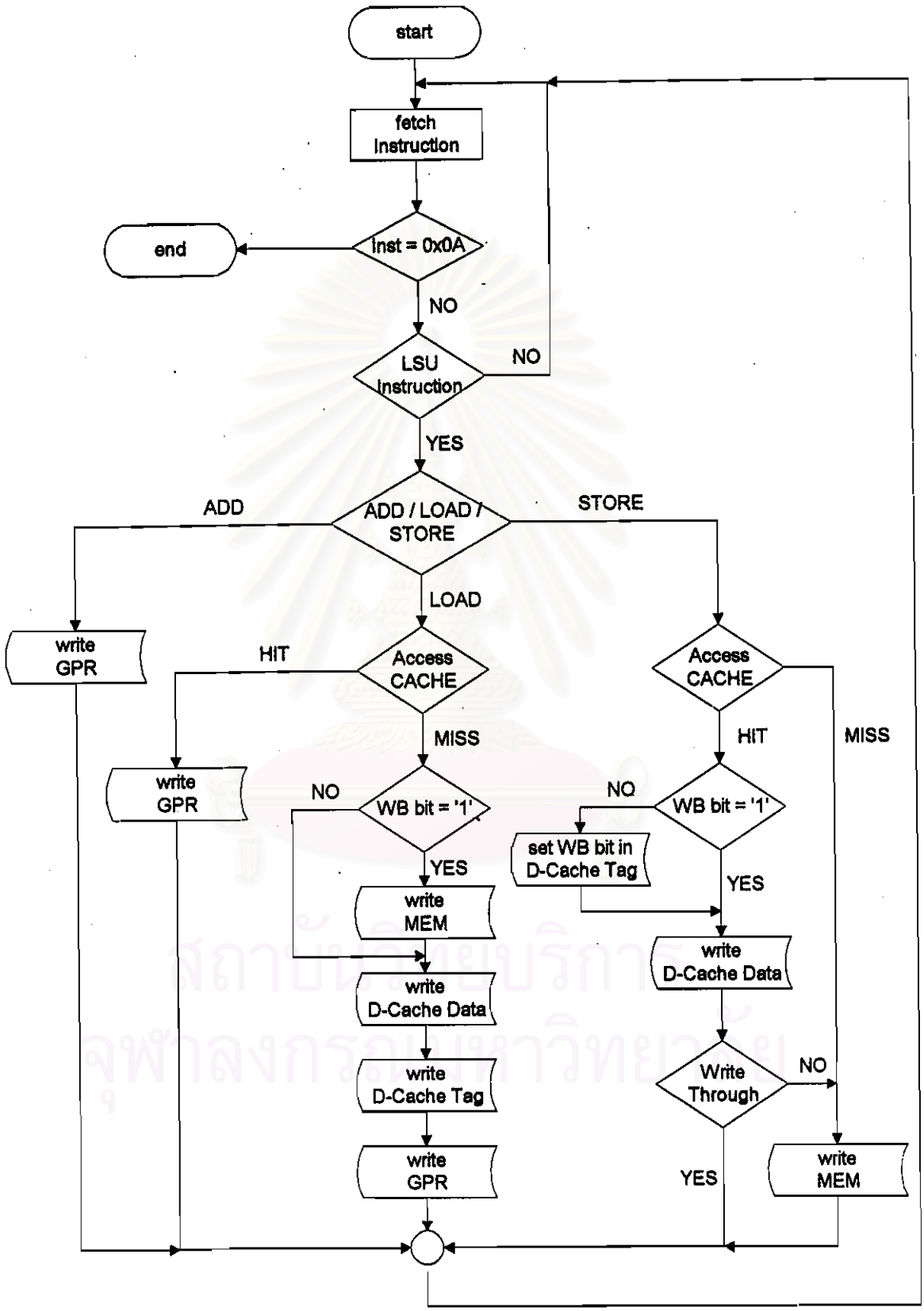
   หมายถึง เขียนข้อมูล 0a030a030a030a03 ลงใน D-Cache Data Set 1 ที่ตำแหน่ง 203 โดยสั่งเขียนข้อมูลเป็น ff ซึ่งหมายถึงเขียนข้อมูลลงในทุกๆ ไบต์

4. write MEM มีการเขียนข้อมูลลงใน MEM 2 ลักษณะ ตัวอย่างของ Output Vector ดังกล่าวคือ

   "Wr memory[0202] = 0000000f02020202" และ

   "Wr dword memory[0e00] = 0000000b0e000e00"

   ตัวอย่างในบรรทัดแรก 0000000f02020202 หมายถึงผลการเขียนข้อมูลลงหน่วยความจำในตำแหน่ง 0202 ซึ่งอาจขอเขียนเป็น byte, halfword, 3 bytes หรือ word ก็ได้ และในตัวอย่างบรรทัดต่อมาหมายถึงการเขียนข้อมูล 0000000b0e000e00 ลงในหน่วยความจำตำแหน่ง 0e00 โดยมีร้องการเขียนเป็นแบบ doubleword



รูปที่ 5.1 Flowchart การทำงานของวงจร LSU ในภาษาซี

ส่วน Output Vector ที่ได้จากวงจร LSU ที่ออกแบบด้วยภาษา Verilog นั้น ได้ออกมาในขณะที่รันโปรแกรมภาษา Verilog โดยเราต้องกำหนดให้โปรแกรมภาษา Verilog มีการแสดงผลเมื่อมีการเขียนข้อมูลลงในรีจีสเตอร์ (GPR), D-Cache Tag, D-Cache Data และหน่วยความจำ (MEM) โดยใช้คำสั่ง $display ซึ่งผลการแสดงค่าหลังจากรันโปรแกรมเสร็จจะเก็บไว้ที่แฟ้ม verilog.log ทั้งหมด โดยเรียงตามลำดับเหตุการณ์ว่าเกิดการเขียนข้อมูลที่ใดขึ้นก่อน

การเปรียบเทียบ Output Vector ที่ได้จากโปรแกรมภาษา Verilog และจากโปรแกรมภาษาซีไม่สามารถทำได้โดยตรงเนื่องจากลำดับเหตุการณ์การเขียนข้อมูลลงในหน่วยความจำที่เกิดขึ้นไม่ตรงกัน เพราะการทำงานของวงจร LSU ในโปรแกรมภาษา Verilog มี WBB (Write Back Buffer), SHB (Store Hit Buffer) และ FIFO พักข้อมูลที่ต้องการเขียนลง D-Cache Tag, D-Cache Data และหน่วยความจำไว้ แล้วค่อยเขียนในภายหลัง อีกทั้งการเกิด load miss ของโปรแกรมภาษา Verilog ไม่มีการรอจนกระทั่งได้ข้อมูลที่โหลดมาเขียนลงรีจีสเตอร์เหมือนกับโปรแกรมภาษาซีอีกด้วย

แต่เราก็สามารถเปรียบเทียบ Output Vector ที่ได้จากโปรแกรมในภาษาทั้งสองโดยการแยกแฟ้มตามการเขียนข้อมูลลงหน่วยจำ ซึ่งได้แก่ รีจีสเตอร์ (GPR), D-Cache Tag, D-Cache Data และหน่วยความจำ (MEM) แล้วนำแฟ้มที่ได้จากทั้งสองโปรแกรมมาเปรียบเทียบกัน แต่ก็ยังคงมีปัญหาในเรื่องลำดับในการเขียนข้อมูลอยู่บ้าง ผู้วิจัยจึงได้ทำการเปรียบเทียบโดยการดูว่ามีข้อมูลเหมือนกัน โดยไม่สนใจลำดับเหตุการณ์มากนัก ซึ่งผลของ Output Vector คงจะมีการสลับบรรทัดกันไปบ้าง นอกจากปัญหาในเรื่องลำดับข้อมูลแล้ว ยังมีปัญหาที่การตรวจสอบการเขียนข้อมูลลงใน MEM อีกเล็กน้อย เนื่องจากการทำงานของวงจร LSU ในภาษา Verilog มีการจอง FIFO เป็นชนิด doubleword ทำให้การ Store Miss ของ 2 คำสั่งที่ตำแหน่ง doubleword เดียวกันมีการเขียนข้อมูลลงใน MEM เป็น doubleword แต่โปรแกรมจำลองวงจร LSU ในภาษาซีจะเขียนลงหน่วยความจำ 2 ครั้ง ผู้วิจัยจึงได้แก้ปัญหาโดยการตรวจดู Output Vector ที่ได้จากการเขียนภาษาซีว่าต้องมีการเขียนข้อมูลลง MEM 2 ครั้งต่อการเกิดเหตุการณ์ STORE MISS ที่ตำแหน่ง doubleword เดียวกัน

5.1.2 กำหนดหน่วยความจำเริ่มต้นที่ใช้ในการทดสอบ

ส่วนของคำสั่ง (Code Segment) ที่ใช้ในการทดสอบความถูกต้องในการทำงานของวงจร LSU ถูกสร้างด้วยวิธีการสุ่มคำสั่งขึ้นมาจำนวนหนึ่ง พร้อมกันนั้นได้กำหนดข้อมูลในส่วน Data Segment ขึ้นมาด้วย และกำหนดให้ Register Base อยู่ในช่วง R0 ถึง R3 (ไม่อนุญาตให้ Register Target ใช้รีจีสเตอร์ 4 ตัวนี้) โดยวิธีการที่ใช้ในการสุ่ม เป็นดังนี้

1) วิธีที่ 1 ทำการสุ่มมาจากชุดคำสั่งที่มีคำสั่งทั้งหมด 41 คำสั่ง โดยเป็นคำสั่งที่สามารถทำงานได้จริงในวงจร LSU ที่ออกแบบเพียง 13 คำสั่ง ทั้งนี้เพื่อจำลองสถานการณ์ให้เหมือนกับการประมวลผลโปรแกรมจริง กล่าวคือมีคำสั่งที่สามารถทำงานได้บน LSU อยู่ประมาณ 1 ใน 3 ของทั้งหมด และมี Register Target อยู่ในช่วง R4 ถึง R31
2) วิธีที่ 2 ทำการสุ่มมาเฉพาะชุดคำสั่งที่สามารถทำงานได้บน LSU เท่านั้น และมี Register Target อยู่ในช่วง R4 ถึง R31

3) วิธีที่ 3 ทำการสุ่มมาจากชุดคำสั่งชุดคำสั่งที่สามารถทำงานได้บน LSU เท่านั้น โดยกำหนดพื้นที่ในการอ้างอิงแคชให้เล็กลงเหลือเพียง 32 ไบต์ (ด้วยวิธีที่ 1 และวิธีที่ 2 มีการอ้างอิงตำแหน่งของแคชเต็มพื้นที่) และ Register Target อยู่ในช่วง R10 ถึง R17
4) วิธีที่ 4 ทำการสุ่มมาจากชุดคำสั่ง LOAD ทั้งหมด 7 คำสั่ง โดยกำหนดพื้นที่ในการอ้างอิงแคชให้เล็กลงเหลือเพียง 32 ไบต์ และ Register Target อยู่ในช่วง R10 ถึง R17
5) วิธีที่ 5 ทำการสุ่มมาจากชุดคำสั่ง STORE ทั้งหมด 5 คำสั่ง โดยกำหนดพื้นที่ในการอ้างอิงแคช 32 ไบต์ และ Register Target อยู่ในช่วง R10 ถึง R17

สรุปวิธีการสุ่มทั้ง 5 วิธี ดูได้จากตารางที่ 5.1 จะเห็นได้ว่าวิธีที่ 1 สร้างขึ้นมาเพื่อจำลองสภาพการณ์จริง แต่ในวิธีที่ 2 และ 3 เป็นการสุ่มคำสั่งที่ทำให้มีโอกาสข้อผิดพลาดที่อาจมีภายในวงจรที่ออกแบบมากขึ้น ทำให้ผู้วิจัยสามารถตรวจสอบและแก้จุดบกพร่องได้ง่ายขึ้น ส่วนในวิธีที่ 4 ทำเพื่อเน้นคำสั่ง Load และวิธีที่ 5 ทำเพื่อเน้นคำสั่ง Store

	วิธีที่ 1	วิธีที่ 2	วิธีที่ 3	วิธีที่ 4	วิธีที่ 5
จำนวน Instruction ที่ RANDOM	41	13	13	7	5
Base Register	R0-R3	R0-R3	R0-R3	R0-R3	R0-R3
Target Register	R4-R31	R4-R31	R10-R17	R10-R17	R10-R17
Block Address ref.	4096	4096	32	32	32

ตารางที่ 5.1 วิธี RANDOM คำสั่งที่ใช้ในการทดสอบความถูกต้องของการทำงาน

วิธีการสุ่มที่เปลี่ยนแปลงไปเพื่อให้การทำงานของวงจร LSU ที่ออกแบบด้วยภาษา Verilog มีโอกาสเกิดการผิดพลาดมากขึ้น เพื่อให้เราได้สามารถตรวจสอบหาจุดผิดพลาดได้เร็วขึ้น โดยเฉพาะการกำหนดพื้นที่การอ้างอิงตำแหน่งแคชให้แคบลง ทำให้ตรวจสอบการทำงานของ LSU ที่ออกแบบได้เร็วขึ้นอย่างมาก

หลังจากที่ได้ Code Segment และ Data Segment จากการสุ่มแล้ว ผู้วิจัยได้เอาข้อมูลทั้งหมดนี้เก็บลงในตัวแปรภายในหน่วย MEM และกำหนดให้เริ่มการประมวลผลที่ตำแหน่ง PC = 0000 0000

5.1.3 ผลการทดสอบความถูกต้องของการทำงานของวงจร

จากการตรวจสอบด้วยวิธีการดังกล่าว ได้ผลออกมาว่าวงจรที่ออกแบบยังไม่สามารถทำงานได้ถูกต้อง 100% ความผิดพลาดบางส่วนเกิดจาก bug ของโปรแกรมซึ่งต้องใช้เวลาในการค้นหานานมาก เช่นเกิดความผิดพลาดขึ้นเมื่อทดสอบกับชุดคำสั่งจำนวน 1,000 คำสั่ง ผู้วิจัยไม่สามารถตรวจจับสัญญาณด้วย Timming Diagram ได้เพราะว่าต้องใช้เวลาในการหานานมาก อีกทั้งความผิดพลาดอาจเกิดขึ้นเนื่องมาจากการที่วงจรที่ออกแบบทำงานถูกต้องอยู่แล้ว แต่ Output Vector จากภาษาซีนั้นให้ผลไม่ตรงกัน เช่น กรณีที่คำสั่งที่ตามกันมาเกิด Store Hit ในแอดเดรสตำแหน่งเดียวกัน Output Vector ที่ได้จากภาษา Verilog จะทำการเขียนข้อมูลลงแคชเพียงครั้งเดียว แต่ Output Vector จากภาษาซีที่ทำงานเป็น Sequential นั้นจะเขียน 2 ครั้ง ทำให้ผลที่ได้

ออกมาไม่เหมือนกัน ผลดังกล่าวก็จะเห็นเป็นข้อผิดพลาด ซึ่งผู้วิจัยไม่สามารถแยกแยะได้ว่าข้อผิดพลาดที่เกิดขึ้นนั้นมาจากสาเหตุใด

และเมื่อนำวงจรที่ได้ไปใช้งานกับ Trace จริง ผู้วิจัยมีความจำเป็นต้องตัดการทำงานส่วนที่มีการเขียนข้อมูลลงหน่วยความจำออกไป เพื่อป้องกันการเขียนข้อมูลทับในส่วนของ Code Segment และการทำงานจะไม่มีการแสดงค่า Output Vector เพื่อให้ประมวลผลได้เร็วขึ้น ผู้วิจัยจึงดูความถูกต้องของการทำงานจากจำนวนครั้งที่เกิดจากการ Load และ Store ซึ่งผลที่ได้เป็นที่น่าพอใจ กล่าวคือมีจำนวนคำสั่ง Load และ Store บางส่วนที่หายไป โดยการประมวลผลคำสั่ง Load ที่หายไปเท่ากับ 0.01% และการประมวลผลคำสั่ง Store ครบตามที่ต้องการ

5.2 วิธีการที่ใช้ในการสร้าง Trace

ผู้วิจัยได้เขียนโปรแกรมภาษาซีเพื่อสร้าง Trace โดยได้กล่าวถึงวิธีการเขียนโปรแกรมดังกล่าวในภาคผนวก ข. Trace ที่ใช้ได้จากการรันโปรแกรมที่ใช้งานทั่วไป ดังต่อไปนี้

1. gzip.asm

   Trace นี้ได้จากการทำงานตามคำสั่ง "gzip opcode4.dat" โดยผลลัพธ์ของการบีบอัดแฟ้มที่ได้เปลี่ยนจากขนาด 1,187 ไบต์ไปเป็น 291 ไบต์

2. gunzip.asm

   Trace นี้ได้จากการทำงานตามคำสั่ง "gzip -d opcode4.dat.gz" โดยผลลัพธ์ที่ได้จากการขยายแฟ้มเปลี่ยนจาก 291 ไบต์ไปเป็น 1,187 ไบต์

3. diff.asm

   Trace นี้ได้จากการทำงานตามคำสั่ง "diff opcode4.dat opcode5.dat" โดยขนาดของแฟ้ม opcode5.dat มีขนาด 302 ไบต์ และข้อมูลในแฟ้ม opcode5.dat นั้นเป็น subset ของข้อมูลในแฟ้ม opcode4.dat และขนาดของความแตกต่างที่ได้เท่ากับ 992 ไบต์

4. compress.asm

   Trace นี้ได้จากการทำงานตามคำสั่ง "compress gzip.asm" ซึ่ง gzip.asm เป็น Trace ที่ได้จากการทำ gzip ในคำสั่งที่ 1 โดย gzip.asm นี้มีขนาด 2,091,792 ไบต์ และผลจากการ compress แฟ้ม gzip.asm นี้ได้แฟ้ม gzip.asm.Z ขนาดออกมาเท่ากับ 314,837 ไบต์

5. uncompress.asm

   Trace นี้ได้จากการทำงานตามคำสั่ง "uncompress gzip.asm.Z" ซึ่งผลการทำ uncompress ได้แฟ้มขนาดย้อนกลับกับการทำงานตามคำสั่ง compress

6. cc.asm

   Trace นี้ได้จากการทำงานตามคำสั่ง "cc -g -o mcr11 mcr11.cpp" ซึ่งเป็นการ compile โปรแกรมภาษาซี โดยในการสร้างแฟ้ม cc.asm นี้กินเวลาในการสร้างและการรันเพื่อเก็บข้อมูลในการวัดสมรรถนะนานมาก ซึ่งระยะเวลาไม่เพียงพอในการทำวิจัยนี้ ดังนั้นผู้วิจัยจึงได้จำกัดจำนวนคำสั่งให้แฟ้ม cc.asm ทำงานเพียงประมาณ 1 ล้านคำสั่ง

5.3 การลดขนาด Trace

Trace คือ ลำดับของแอดเดรส (Sequence of Address) ที่มีการติดต่อกับหน่วยความจำ ซึ่งก็คือค่าแอดเดรสที่มาจากคำสั่ง Load และคำสั่ง Store (จาก Trace ที่เราสร้างนั้นมีจำนวนเฉลี่ยของคำสั่ง Load และคำสั่ง Store อยู่ประมาณ 37 เปอร์เซนต์) โดยมีค่าประมาณ 1 ใน 3 ของโปรแกรมทั้งหมด

เนื่องจากในการวิเคราะห์การทำงานของแคชสำหรับไมโครโพรเซสเซอร์นั้นกินเวลาในการทดสอบนานมาก จึงมีผู้พยายามที่จะลดเวลาในส่วนนี้ลง จนกระทั่ง Harold S. Stone (ค.ศ. 1985) ได้เสนอวิธีการลดขนาดของ Trace เอาไว้ 2 แบบคือ

1. Trace Filtering ทำได้โดย ในครั้งแรกให้ทดลองกับขนาดของแคชที่เล็กที่สุด แล้วทำการเก็บ trace ที่ทำให้เกิดแคช Miss เอาไว้ ส่วน Trace ที่เกิดแคช Hit นั้นตัดทิ้งไปได้ ซึ่งถ้าได้ Miss Ratio ที่ขนาดของแคชเล็กที่สุดเป็น 10 เปอร์เซนต์ ก็สามารถลด Trace ไปได้ถึง 90 เปอร์เซนต์ จากนั้นก็นำ Trace ที่เหลือขนาดเพียง 10 เปอร์เซนต์นั้นไปรันกับแคชที่ขนาดใหญ่ขึ้น ก็จะได้ค่า Miss Ratio เปลี่ยนแปลงตามขนาดของแคชที่เราต้องการ

   นอกจากนี้ยังดัดแปลงให้ช่วยลดขนาดของ Trace เมื่อเปลี่ยนแปลง Degree of Set Associative ได้อีกด้วย โดยในครั้งแรกให้ทดลองกับการจัดแคชแบบ direct-map เพื่อเก็บ trace ที่เกิดแคช Miss และในครั้งต่อไปก็นำไปทดลองกับ N-way Set Associative ที่ N มีค่าเท่ากับ 2, 4, 8 เป็นต้น

   ด้วยวิธีการลด Trace แบบนี้จะทำให้ผลจากการวัดค่า Miss Ratio ที่ได้ มีค่าเท่ากับค่า Miss Ratio ที่ไม่มีการตัด Trace เลย

2. Set Sampling เทคนิคนี้กล่าวไว้ว่าบรรทัดหนึ่งของแคชมีพฤติกรรมเหมือนกับบรรทัดอื่นๆ ดังนั้น การวัดสมรรถนะของแคชสามารถประมาณได้จากแคชเพียงบรรทัดเดียว แต่ในความเป็นจริงวิธีการนี้ให้ผลที่ไม่ถูกต้องนัก แต่ถ้าหากใช้จำนวนบรรทัดที่เป็นตัวแทนในการตรวจสอบมากขึ้น ก็จะได้ความมั่นใจกับผลของการวัดสมรรถนะที่ได้มากขึ้น เช่นการใช้แคช 6 บรรทัดเป็นตัวแทนการวัดแคชขนาด 64 บรรทัด จะได้ผลการวัดที่ถูกต้องเพียงพอ กล่าวคือใช้ตัวแทนประมาณ 10% จะให้ผลการวัดเป็นที่น่าพอใจ

เมื่อลดขนาดขนาด trace ด้วยทั้งสองเทคนิคนี้แล้ว จะทำให้ trace ลดขนาดลงเหลือเพียง 1% ของ Trace ที่มีอยู่ในตอนแรก (ด้วยวิธี Trace Filtering สามารถลดขนาดของ Trace ไปได้ 10% และวิธี Set Sampling สามารถลด trace ไปได้อีก 10%) ทำให้เวลาที่ใช้ในการวิเคราะห์พฤติกรรมของแคชเพื่อหาค่า Miss Ratio เหลือเพียง 1% จากที่เคยใช้ Trace ขนาดใหญ่ในตอนแรก

ผู้วิจัยเห็นว่าไม่เหมาะสมที่จะนำเทคนิคทั้งสองมาใช้กับงานวิจัยนี้ เนื่องจากว่าไม่มีเหตุการณ์ Store Hit เกิดขึ้น ถ้าเราใช้เทคนิคดังกล่าวทำให้ไม่สามารถวัดสมรรถนะของ SHC (Store Hit Control) Unit จากการปรับขนาดของบัฟเฟอร์ได้ นอกจากนี้ ถ้าหากเราใช้ Trace เฉพาะคำสั่งที่มีการติดต่อกับหน่วยความจำเพียงอย่างเดียว (คำสั่ง Load และคำสั่ง Store) จะทำให้เกิดแคช Miss ขึ้นบ่อยมาก อันมีผลต่อการทำงานของ LMC (Load Miss Control) Unit และ SMC (Store Miss Control) Unit กล่าวคือ ทำให้เกิดการ "STALL" Pipeline อันเนื่องมาจาก LSU เป็นประจำ ทำให้เราไม่สามารถวัดสมรรถนะที่ดีขึ้นของการมี LMC, SHC และ SMC ได้ กล่าวคือ การลดขนาด Trace ด้วยวิธีการดังกล่าวใช้หาค่า Miss Ratio ได้เท่านั้น ไม่สามารถนำมาใช้วัดผลอื่นๆ

ที่มีความจำเป็นในงานวิจัยนี้ได้เลย ดังนั้น Trace ที่ใช้ทดสอบสมรรถนะในงานวิจัยนี้ จึงใช้ Trace ที่ได้มาทั้งหมดทดลอง

5.4 วิธีการเอา Trace ไปใช้

Trace ที่ได้เป็นแฟ้ม Assembly ของ CPU MIPS R4000 บนเครื่อง Siligon Graphic ผู้วิจัยได้เขียนโปรแกรมแปลงภาษา assembly ให้เป็น machine code โดยโปรแกรมที่เขียนขึ้นจะทำเฉพาะชุดคำสั่ง LOAD, STORE และคำสั่ง ADD (ตามที่กำหนดไว้ในภาคผนวก ก.) คำสั่งนอกเหนือจากนั้นได้กำหนด machine code ให้เป็น 0x05 (เป็นค่าที่ผู้วิจัยกำหนดขึ้นเอง) และจบโปรแกรมด้วยค่า 0x0A เพื่อเป็นคำสั่งที่บอกให้วงจรรู้ว่าสิ้นสุดการทำงานเพียงเท่านี้

ในการทดสอบช่วงแรกของการทำวิจัยนี้ ผู้วิจัยได้นำคำสั่งจาก Trace ที่ได้เขียนลงในตัวแปรภายในหน่วย MEM ซึ่งมีผลเหมือนกับการกำหนดค่าภายในตัวแปร array ของภาษาซี การทำแบบนี้ส่งผลให้แฟ้ม MEM ที่ใช้มีขนาดใหญ่พอๆ กับแฟ้มที่เก็บคำสั่งที่ใช้ในการทดสอบ ซึ่งในช่วงของการตรวจสอบความถูกต้องในการทำงานของวงจร แฟ้มดังกล่าวเกิดจากการ RANDOM คำสั่งมาไม่เกินหนึ่งพันคำสั่ง จึงไม่มีผลกระทบอะไร แต่เมื่อต้องการวัดสมรรถนะของวงจร แฟ้มที่ใช้เป็น Trace ขนาดใหญ่ที่สุดคือสองล้านกว่าคำสั่ง ส่งผลให้เนื้อที่บนจานบรรทึกแบบแข็ง (Hard disk) ของเครื่องคอมพิวเตอร์ที่ใช้ไม่เพียงพอต่อการทำงาน อีกทั้งยังมีปัญหาในเรื่องของหน่วยความจำขณะที่ทำงาน (ขนาดของ RAM) อีกด้วย

วิธีแก้ไขคือการใช้คำสั่ง $readmemb("filename",<memname>); โดยคำสั่ง $readmemb ใช้ในการขออ่านแฟ้มเป็น binary , filename คือชื่อแฟ้มที่ต้องการอ่าน และ memname คือตัวแปร array ที่ต้องการอ่านข้อมูลมาเก็บ อันเป็นการลดขนาดของแฟ้ม MEM ลงได้อย่างมาก แต่วิธีการนี้ไม่สมควรใช้งานในกรณีที่ต้องการทดสอบความถูกต้องของการทำงานของวงจร เนื่องจากการ RANDOM เราไม่ได้สร้างส่วน data segment ขึ้นมาทั้งหมด ดังนั้นแฟ้มที่จะให้ตัวแปรอ่านค่ามาเก็บนั้นจะต้องมีขนาดใหญ่มากจึงจะครอบคลุมพื้นที่ของหน่วยความจำที่ใช้ทั้งหมด

5.5 สรุป

บทนี้ได้กล่าวถึงวิธีการที่ใช้ในการตรวจสอบความถูกต้องของการทำงานของวงจร โดยวิธีที่ใช้คือ เขียนโปรแกรมภาษาซีเพื่อจำลองการทำงานของวงจร LSU สุ่มคำสั่งขึ้นมาประมวลผลกับวงจรดังกล่าว และเอา Output Vector ที่ได้มาเปรียบเทียบกับผลการทำงานของวงจร LSU ที่เขียนด้วยโปรแกรมภาษา Verilog โดยผลการตรวจสอบความถูกต้องที่ได้เกือบสมบูรณ์ 100% และได้กล่าวถึงวิธีการที่ใช้ในการสร้าง Trace และวิธีการลดขนาด Trace ลงเพื่อให้ทำงานเร็วขึ้น แต่วิธีการลดขนาด Trace ดังกล่าว ไม่สามารถนำมาใช้กับงานวิจัยนี้ได้ เพราะจะทำให้ไม่สามารถวัดสมรรถนะการทำงานของหน่วย LMC, SHC และ SMC ซึ่งเป็นหน่วยสำคัญ นอกจากนี้แล้วยังได้กล่าวถึงวิธีการนำเอา Trace ที่ได้ไปใช้ ซึ่งทำโดยการเก็บคำสั่งลงในตัวแปรที่จองไว้ใน MEM