204324 ปฏิบัติการระบบคอมพิวเตอร์

ปฏิบัติการที่ 4: สร้าง MIPS CPU แบบ pipeline บน Logisim (ส่วนที่ 2)

ณ จุดนี้สิ่งที่เรามีคือวงจร pipelined MIPS ที่ประมวลผลได้อย่างต่อเนื่องถ้าไม่มี data hazard

ในปฏิบัติการส่วนนี้เราจะทำ pipelined MIPS ให้สมบูรณ์โดยสร้าง forwarding network เพื่อจัดการกับ data hazard อ่านรายละเอียดเกี่ยวกับการสร้าง forwarding network ได้จากสไลด์ประกอบปฏิบัติการที่แนบมา (ไฟล์ forwarding_implementation.pdf)

อย่างไรก็ตามเราได้เห็นแล้วว่ามีบางกรณีของการประมวลผลคำสั่งที่ forwarding network ช่วยได้ไม่สมบูรณ์ กรณีนี้คือกรณี load-to-use ดังตัวอย่างต่อไปนี้

```
lw      $8, 12($10)
add     $12, $8, $9
```

ในกรณีนี้คำสั่ง add เป็นคำสั่งถัดจาก lw และใช้ผลลัพธ์จาก lw เมื่อ add อยู่ที่ EXE และต้องการค่าที่ถูกต้องของ $8 มาประมวลผล forwarding network ไม่สามารถจะส่งค่า $8 มาให้ได้ทันเนื่องจากค่าที่ถูกต้องของ $8 จะได้ก็ต่อเมื่อ lw ไปอยู่ที่ WB ดังนั้นในกรณีนี้เราจะต้องใส่ stall logic เพื่อหยุดการประมวลผล add ไว้ที่ EXE 1 cycle แล้วส่ง nop (bubble) เข้าไปที่ MEM ใน cycle ต่อมา

เราต้องออกแบบและสร้าง stall logic จัดการกับกรณี load-to-use ข้างต้นเพิ่มเข้าไปใน pipeline ด้วย

เมื่อสร้างวงจร pipelined MIPS สมบูรณ์แล้ว ให้ทดสอบกับกรณีทดสอบที่แนบมากับปฏิบัติการนี้คือ test1.asm โดย test1.asm คือ iterative hailstone ที่ใส่ nop เข้าไปใน delayed slot ก่อนจะทดลองรันจริง ให้นับจำนวน cycle ที่ใช้ในการประมวลผล test1.asm บน pipelined MIPS ที่สมบูรณ์ โดยเขียนเป็น comment ลงในแต่ละคำสั่ง ให้ว่าโปรแกรมเริ่มการทำงานที่ main: อย่าลืมว่ากรณี load-to-use จะต้องมีการ stall 1 cycle ด้วย nop ที่ใส่ไว้ท้ายคำสั่ง assembly ในไฟล์ test1.asm มีไว้เพื่อ “ล้าง” pipeline เพื่อให้มั่นใจว่าคำสั่ง assembly สุดท้ายที่มีความหมายที่จะต้องประมวลผล ถูกผลักจนไปอยู่ที่ write back ก่อนที่การประมวลผลจะสิ้นสุดเพราะไม่มีคำสั่งที่เหลือต้องประมวลผลใน MIPS ROM อีก

เมื่อวิเคราะห์เรียบร้อยแล้ว นำ test1.asm มาทดสอบกับวงจรที่สร้างขึ้นจริง เปรียบเทียบว่าจำนวน cycle ที่ใช้ประมวลผลเป็นไปตามที่วิเคราะห์ไว้หรือไม่

สร้าง test2.asm ที่เป็น iterative hailstone ที่เราหา**คำสั่งที่เป็นประโยชน์**ที่ไม่ใช่ nop มาใส่ใน delayed slot ให้นิสิตนับจำนวน cycle ที่ใช้ในการประมวลผล test2.asm บน MIPS pipeline ที่สมบูรณ์ แล้วเขียนเป็น comment ลงในแต่ละคำสั่งในลักษณะเดียวกับที่มีใน test1.asm เปรียบเทียบจำนวน cycle ที่ test1.asm ใช้กับจำนวน cycle ที่ test2.asm ใช้ เราได้ค่า speedup เพิ่มขึ้นมาเท่าใด

จากนั้นทำการทดสอบ test2.asm บนวงจร pipelined MIPS ที่สมบูรณ์แล้วดูว่าใช้ cycle ในการประมวลผลตรงตามที่คาดไว้จากการวิเคราะห์หรือไม่

ในการทดสอบวงจร pipeliined MIPS ที่สร้างขึ้นมากับกรณีทดสอบทั้ง 2 ให้บันทึกวิดีโอสั้นๆ แสดงถึงสภาวะการทำงานที่เปลี่ยนแปลงไปตั้งแต่เริ่มต้นประมวลผลจนเสร็จสิ้นการประมวลผล พร้อมบรรยายประกอบให้ผู้ฟังเข้าใจในสิ่งที่ต้องการนำเสนอ

สิ่งที่ต้องส่ง

- ไฟล์ mips_pipeline.circ ที่มี forwarding network และ stall logic ในการจัดการกับ data hazard อย่างสมบูรณ์แบบ
- ไฟล์ test1.asm ที่มีการนับ cycle ในการประมวลผลบน pipelined MIPS โดยใส่การนับ cycle ของแต่ละคำสั่งไว้ในส่วน comment
- ไฟล์ test2.asm ที่มีการนับ cycle ในการประมวลผลบน pipelined MIPS โดยใส่การนับ cycle ของแต่ละคำสั่งไว้ในส่วน comment
- ไฟล์วิดีโอหรือลิงค์ไปหาไฟล์วิดีโอที่แสดงการประมวลผล test1.asm และ test2.asm บนวงจร pipelined MIPS ที่ได้ถูกสร้างขึ้นมา

การส่งงาน:

- **นำงานที่ต้องส่งใส่ไว้ในโฟลเดอร์ชื่อ**

studentID1_firstname1_studentID2_firstname2_lab4_part_2

โดย studentID และ firstname คือเลขประจำตัวและชื่อแรกของสมาชิกที่ทำปฏิบัติการร่วมกัน จากนั้น zip โฟลเดอร์นี้แล้วส่ง zip ไฟล์มาที่ Google Classroom ของวิชาก่อนกำหนดส่ง

ถ้าถึงกำหนดส่งแล้วนิสิตยังทำปฏิบัติการไม่สมบูรณ์ ขอให้ชี้แจงอุปสรรค บั๊กต่างๆที่พบมาในไฟล์ README.pdf แล้วใส่ไฟล์นี้มาในโฟลเดอร์เดียวกับงานอื่นๆที่ต้องส่ง