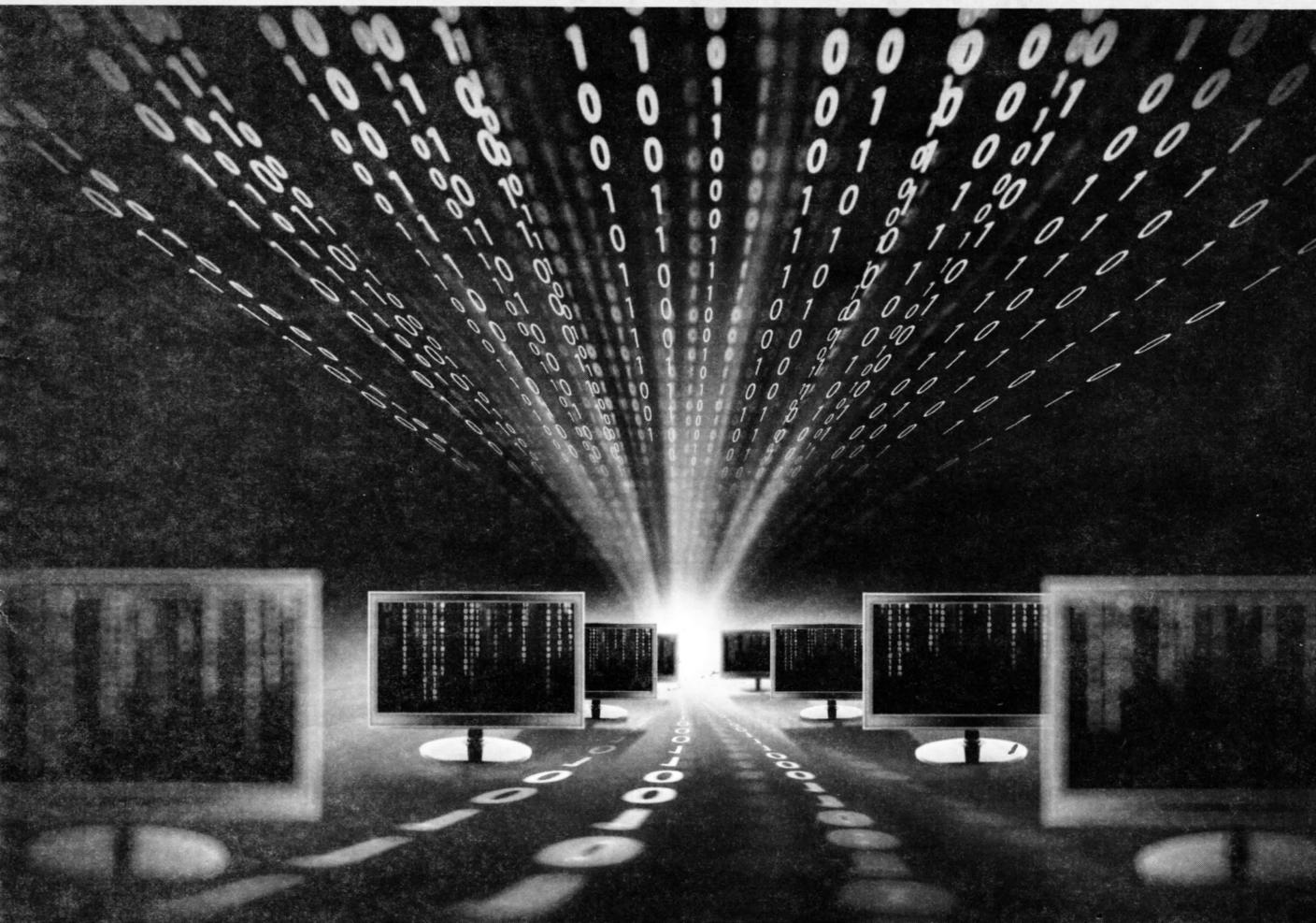


# 4

## การออกแบบหน่วยประมวลผล ทางคณิตศาสตร์และ Data Path



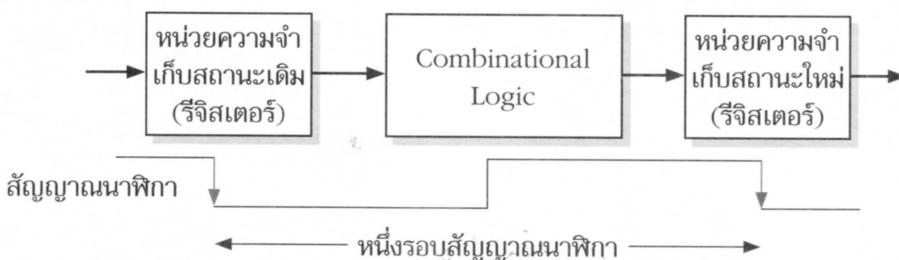
หน่วยประมวลผลประกอบด้วยส่วนที่สำคัญ 2 ส่วน ได้แก่ Data Processing และหน่วยควบคุม ในบทนี้จะกล่าวถึงหน่วยคำนวณต่างๆ ภายในหน่วยประมวลผล และการสร้างการเข้ามต่อระหว่างหน่วยคำนวณเหล่านั้นในการสร้างทางเดินของข้อมูลไปสู่หน่วยต่างๆ รวมทั้งจุดเข้ามต่อที่เราเรียกว่า Data Path หลังจากนั้นในบทต่อไปจะอธิบายถึงการออกแบบหน่วยควบคุมเพื่อควบคุมการส่งสัญญาณการไหลของข้อมูลไปยังหน่วยคำนวณให้ถูกต้องตามการทำงานของแต่ละคำสั่ง

ในการออกแบบ Data Path นั้น จะอ้างอิงการอธิบายจาก MIPS เป็นหลัก โดยจะเริ่มพิจารณาจากหน่วยที่สำคัญในแต่ละขั้นตอนของงาน และนำแต่ละหน่วยนั้นมาเข้ามกันเป็น Data Path จะเริ่มอธิบายจากการออกแบบแบบไฟเกลเดียว (Single Cycle) และหน่วยคำนวณต่างๆ ก่อน จากนั้นจะอธิบายรูปแบบแบบหลายไฟเกล (Multicycle) ซึ่งจะเป็นการสร้างพื้นฐานก่อนจะไปกล่าวถึงการสร้าง Data Path สำหรับไปป์ไลน์ในบทต่อไป

## ■ 4.1 การใช้หน่วยคำนวณต่างๆ ในแต่ละขั้นตอน

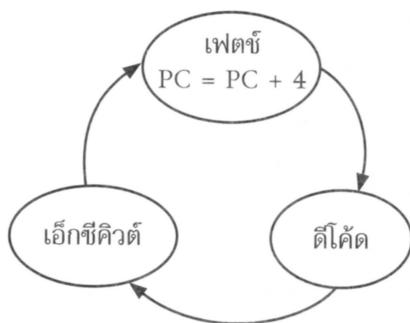
เครื่องคอมพิวเตอร์ทำงานโดยอาศัยสัญญาณนาฬิกา ความเร็วในการคำนวณสำหรับแต่ละคำสั่งจะขึ้นกับสัญญาณนาฬิกา ถ้าหากทำงานด้วยความถี่สูงก็จะคำนวณเร็วมากขึ้นตามไปด้วย

โดยปกติแล้วการคำนวณจะเริ่มต้นภายในรอบสัญญาณนาฬิกา เรียกว่าเป็นรูปแบบของการทำงานแบบหนึ่งไฟเกล (Single Cycle) นั่นเอง โดยตั้งแต่การอ่านข้อมูลจากเรจิสเตอร์ที่เก็บสถานะเดิมในรูปการทำงานคำสั่งด้วย Combinational Logic และเขียนผลลัพธ์จากการทำงานในเรจิสเตอร์ที่เก็บสถานะใหม่ ดังแสดงในรูปที่ 4.1 ในที่นี้พิจารณาการทำงานที่ขับเคลื่อนของสัญญาณนาฬิกาในหนึ่งรอบ



รูปที่ 4.1 การทำงานในหนึ่งรอบสัญญาณนาฬิกา

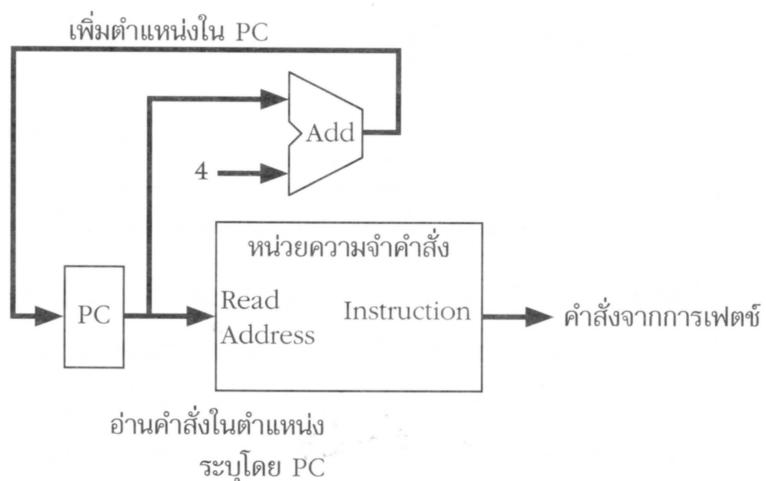
จากรูปแบบขั้นตอนการทำงานที่ได้กล่าวไว้แล้ว หนึ่งรอบสัญญาณจะทำทุกขั้นตอน ดังแต่การเฟตช์ไปจนถึงการເອົກຊີໂຄວົດ ดังแสดงในรูปที่ 4.2



รูปที่ 4.2 การทำงานหนึ่งรอบสัญญาณนาฬิกา

จากรูปข้างต้น ทำให้ใน Data Path ของ MIPS นั้นมีการแบ่งขั้นตอนการทำงานเป็นอย่างน้อย 3 ขั้น ได้แก่ การเฟตช์ (Fetch) ดีโคด (Decode) และເອັກຫີືຄົວຕໍ່ (Execute) ซึ่งมีรายละเอียดดังนี้

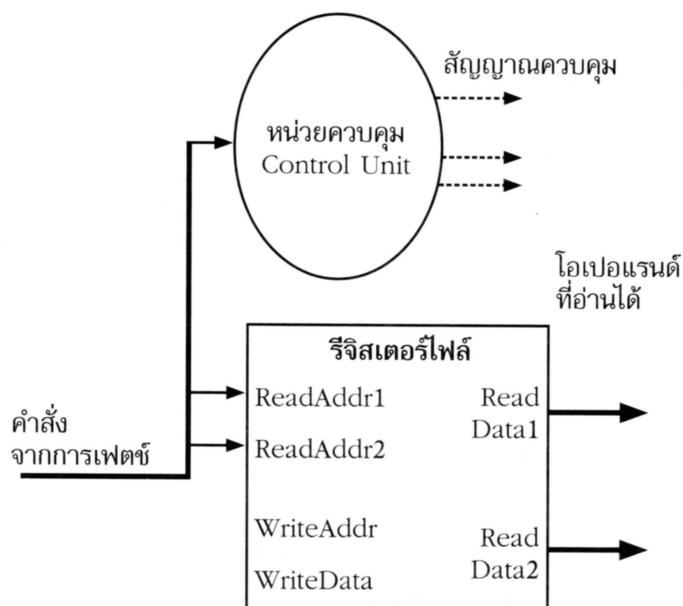
- เฟตช์ (Fetch)** หมายถึง การอ่านคำสั่งจากหน่วยความจำที่เก็บคำสั่งของโปรแกรม โดยตำแหน่งหน่วยความจำที่อ่านจะขึ้นกับค่าในรีจิสเตอร์ PC หลังจากอ่านคำสั่งเข้ามาแล้ว จะทำการเพิ่มค่าใน PC เพื่อไปอ่านคำสั่งในตำแหน่งถัดไป ในที่นี้คือตำแหน่ง  $PC + 4$  การอ่านคำสั่งและการปรับค่า PC นั้นทำทุกรอบ ดังนั้น จึงไม่ต้องการสัญญาณควบคุมแต่อย่างไร



รูปที่ 4.3 หน่วยความจำที่เกี่ยวกับการเฟตช์

รูปที่ 4.3 แสดงการไหลของข้อมูลสำหรับขั้นตอนการเพิ่ม ตัวบวก (Adder) จะนำค่าจาก PC กับค่าคงที่ 4 ค่าในรีจิสเตอร์ PC จะถูกใช้บวกกับตัวแหน่งของหน่วยความจำที่จะไปอ่านเพื่อนำคำสั่งที่จะทำงานออกมา

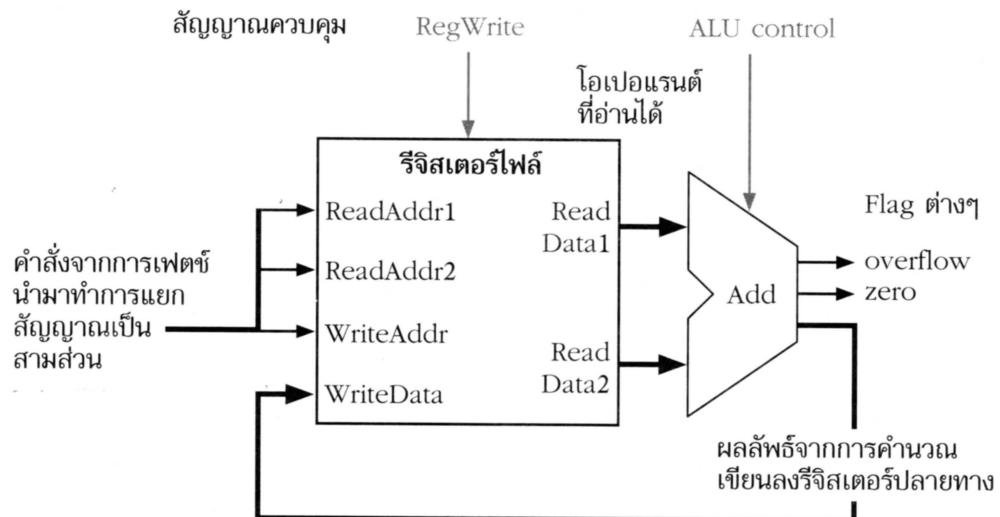
2. **ดีโคด (Decode)** การถอดรหัสโดยเปอแรนด์จะทำการแยกฟิล์ต่างๆ จากคำสั่งที่อ่านได้จากการเพิ่ม และแยกเป็นส่วนๆ ป้อนเข้าเป็นอินพุตของรีจิสเตอร์ไฟล์ เพื่อรับรู้ว่าจะอ่านค่าจากรีจิสเตอร์ตัวใดบ้างสำหรับโดยเปอแรนด์ทั้งสอง และไปทำการอ่านค่าโดยเปอแรนด์ออกมาจากรีจิสเตอร์ และแยกส่วนของ Opcode ส่งไปยังหน่วยควบคุม เพื่อให้หน่วยควบคุมส่งสัญญาณไปควบคุมหน่วยที่เกี่ยวข้องกับคำสั่งนั้นๆ



รูปที่ 4.4 หน่วยรีจิสเตอร์ที่ใช้งานสำหรับขั้นการดีโคด

3. **ເອັກຊີຄົວຕໍ່ (Execute)** เป็นการทำงานตามคำสั่ง โดยโดยเปอแรนด์ที่อ่านมาจากรีจิสเตอร์ในขั้นที่ 2 จะถูกป้อนให้กับ ALU เพื่อคำนวนตามคำสั่งนั้นๆ ระบุโดยสัญญาณ ALUControl หลังจากคำนวนเสร็จแล้ว จะทำการเขียนผลลัพธ์กลับไปยังรีจิสเตอร์ที่เก็บผลลัพธ์ปลายทาง (Destination Register) ต่อไป (โดยข้อมูลที่จะเขียนจะถูกส่งไปยังสัญญาณ WriteData) ในการทำงานขั้นตอนนี้จะต้องการสัญญาณควบคุม เช่น สัญญาณ RegWrite เพื่อควบคุมการเขียนรีจิสเตอร์ โดยให้ค่าเป็น 1 ถ้าต้องการเขียน

รีจิสเตอร์ คำสั่งบางคำสั่งไม่ต้องการการเขียนรีจิสเตอร์ เช่น คำสั่ง Store, Branch สัญญาณการเขียนระบุโดย WriteAddr สำหรับสัญญาณ ALUcontrol เพื่อเลือกฟังก์ชันของ ALU ให้ทำงานตามคำสั่ง เช่น คำสั่ง ADD, SUB, XOR เป็นต้น



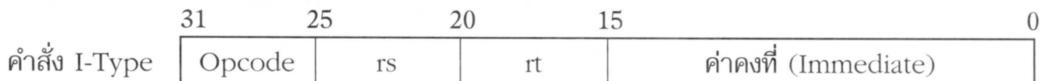
รูปที่ 4.5 หน่วยคำนวณที่เกี่ยวกับการอีกซีคิวต์

เนื่องจากคำสั่งที่ต้องการอีกซีคิวต์มี 3 รูปแบบใน MIPS จึงต้องพิจารณาว่าแต่ละรูปแบบทำงานอย่างไร ต้องวางแผน Data Path และต้องการสัญญาณควบคุมอะไรบ้าง พิจารณารูปแบบคำสั่งจากบทที่แล้ว

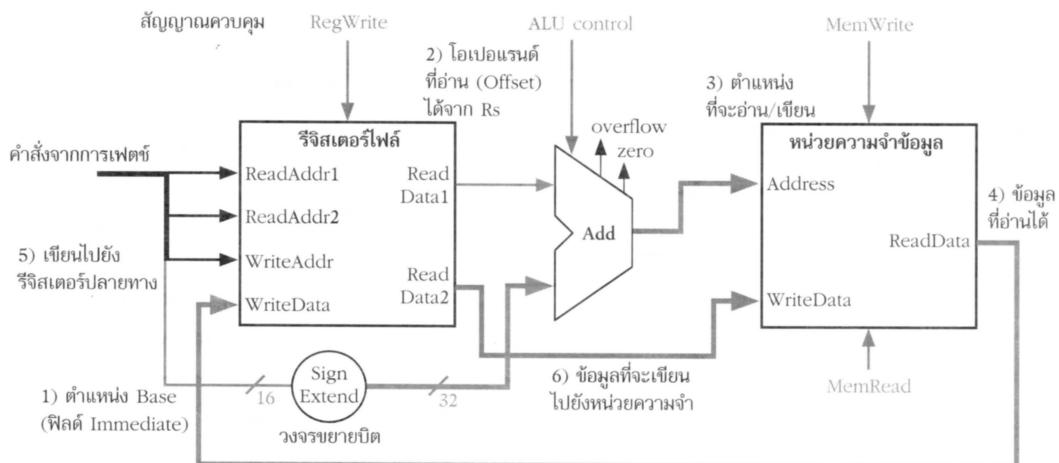
คำสั่ง R-Type	31	25	20	15	10	5	0
	Opcode	rs	rt	rd	sa	funct	

เช่น ถ้าคำสั่งเป็นแบบ R-Type จะมีการแยกตำแหน่งบิตต่างๆในคำสั่งรูปแบบนี้จะใช้กับคำสั่งที่ต้องการคำนวณโดย ALU ได้แก่ คำสั่ง add, sub, slt, and, or โดยที่ ReadAddr1, ReadAddr2 ถูกระบุใน rs, rt และ WriteAddr ถูกระบุโดย rd และ ALUcontrol จะถูกดูดรหัสมาจากการฟิลด์ Opcode ร่วมกับฟิลด์ funct

สำหรับรูปแบบคำสั่งประเภท I-Type มีการวางบิตดังรูป



คำสั่งแบบนี้ได้แก่ คำสั่งประเภท Load/Store ซึ่งจะมีการทำงานต่างออกไป ดังจะอธิบายดังต่อไปนี้

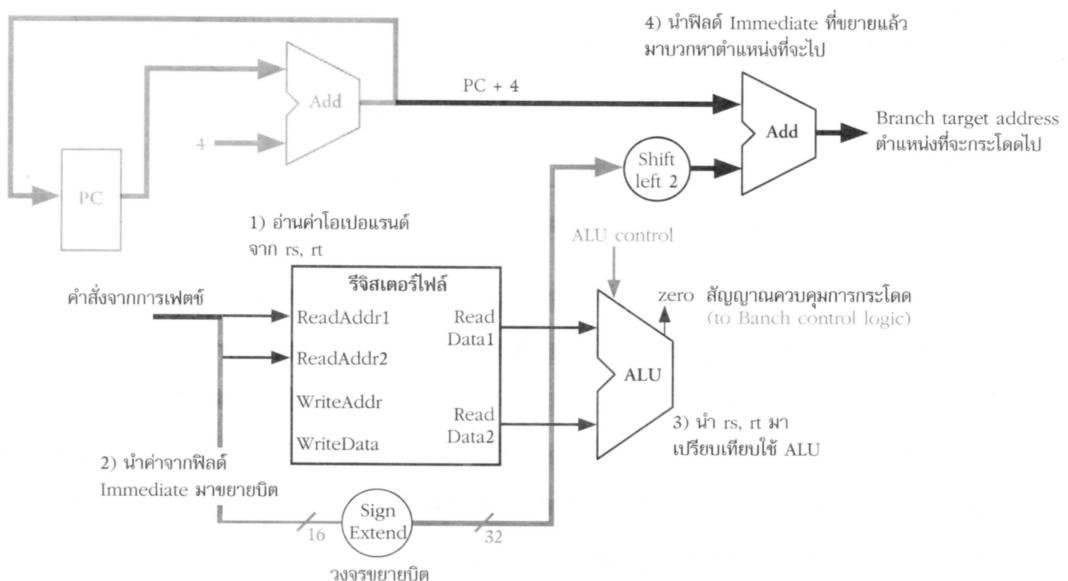


รูปที่ 4.6 หน่วยคำนวณที่เกี่ยวกับ Load/Store และ Data Path

สำหรับคำสั่ง Load จะทำการอ่านค่าจากหน่วยความจำข้อมูล (Data Memory) ระบุโดยตำแหน่ง (Address) เริ่มต้น 1) ค่าตำแหน่งจะได้มาจากการบวกระหว่าง Base ค่าคงที่ที่ปรากฏใน Immediate (ความยาว 16 บิต) และถูกขยายเป็น 32 บิตด้วยวิธีขยายบิต (Sign Extend) 2) จากนั้นนำ\_offset\_พิกัดที่ระบุใน rs หรือ ReadAddr1 3) ต่อไปจะนำผลลัพธ์ที่ได้ไประบุตำแหน่งที่จะอ่านค่าในหน่วยความจำข้อมูล และ 4) จะได้ข้อมูลที่ต้องการมาจากการสัญญาณ Read Data line 5) เพื่อมาเก็บในรีจิสเตอร์ที่ระบุโดยพิกัด rt ถูกป้อนไปยัง WriteAddr และตัวข้อมูลถูกป้อนไปยังสัญญาณ WriteData ในรีจิสเตอร์ไฟล์

สำหรับคำสั่ง Store จะต่างกัน เนื่องจากต้องการเก็บค่าในรีจิสเตอร์ไปยังหน่วยความจำข้อมูล ในส่วนของการคำนวณหาตำแหน่งหน่วยความจำจะเหมือนกับคำสั่ง Load คือจะใช้ ALU มาทำการหา Effective Address ในขั้นที่ 1) ถึงขั้นที่ 3) แต่จะนำค่าที่อ่านได้จากรีจิสเตอร์ rs ในขั้นที่ 2) มาใช้ในการจัดเก็บ โดยข้อมูลจะถูกป้อนเข้าไปในสาย WriteData ในขั้นที่ 6) เลย

สำหรับคำสั่ง Branch จะต่างกัน โดยในขั้นตอนจะเหมือนกัน ได้แก่ 1) ทำการอ่านโอเปอเรนด์ จาก rs, rt และ 2) ทำการอ่านค่า Offset จากพิล็อก Immediate จากนั้นจะต่างกันที่ขั้นตอนที่ 3) ทำการเปรียบโ/o เปอแรนด์ทั้งสองเพื่อเทียบเงื่อนไขว่าทั้งสองโ/o เปอแรนด์เท่ากันหรือไม่ ถ้ามีค่าเท่ากัน จะให้บิต Zero = 1 ถ้าไม่เท่ากัน จะมีบิต Zero = 0 เพื่อใช้ควบคุมการเปลี่ยนค่าใน PC ต่อไป และในขณะเดียวกัน 4) จะคำนวนหาตำแหน่งที่จะกระโดดไป ในการคำนวนหาตำแหน่งนี้ จะใช้ ALU ทำการบวกค่าตำแหน่งสัมพัทธ์ในพิล็อก Immediate เข้ากับผลที่ได้จาก PC + 4 การบวกนี้ใช้ Adder มาช่วย แต่ก่อนจะทำการบวกต้องการขยายบิตของ Immediate พิล็อกที่เป็น 16 บิตให้เป็น 32 บิตโดยใช้ Sign Extend เข่นกัน ดังแสดงในรูปที่ 4.7

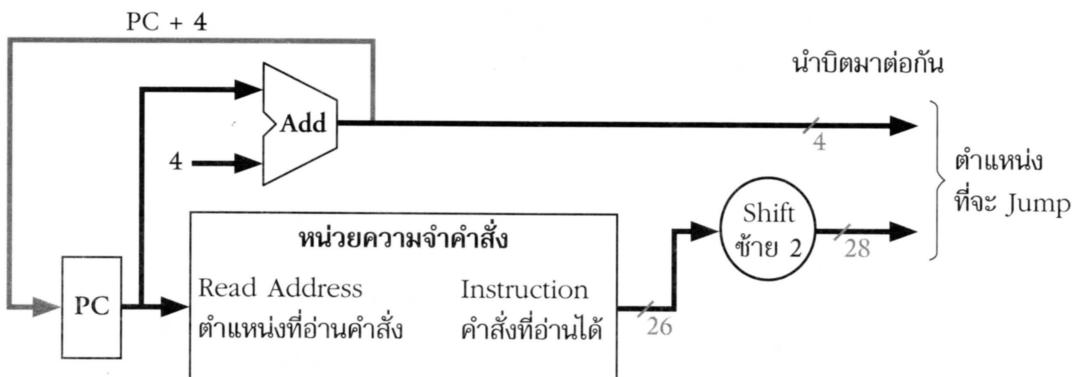


รูปที่ 4.7 หน่วยต่างๆ ที่เกี่ยวกับการ Branch และ Data Path

สำหรับคำสั่งกระโดดเป็นคำสั่งรูปแบบ J-Type มีลักษณะดังนี้

คำสั่ง J-Type	31	25	0
	Opcode	ตำแหน่งที่จะกระโดดไป (Jump Address)	

คำสั่งแบบนี้ได้แก่ Jump ซึ่งจะทำงานง่ายกว่า โดยจะนำค่าที่ได้จาก 4 บิตบนของ PC + 4 มาต่อ กับ ตำแหน่งขนาด 26 บิตซึ่งจะถูกขยายทำให้เป็น 28 บิตก่อน ในรูปค่า PC ใหม่ที่ได้จะมา จาก Jump Address ซึ่งจะได้มาจาก การนำ 26 บิตล่างมา Shift ซ้ายสองครั้ง และนำมาต่อ กับ 4 บิตบนของ PC + 4 ดังแสดงในรูปที่ 4.8



รูปที่ 4.8 หน่วยคำนวณที่เกี่ยวกับการ Jump และ Data Path

จะเห็นได้ว่าในการออกแบบรูปแบบคำสั่งของ MIPS ทั้ง 3 แบบนั้น จะต่างกันในส่วนของบิตด้าน LSB ทั้ง 3 รูปแบบจะมีลักษณะคล้ายๆ กันในส่วนของบิตบน เช่น Opcode อยู่ที่บิต 31-26 ส่วน rs อยู่ที่บิต 25-21 ส่วน rt อยู่ที่บิต 20-16 สำหรับคำสั่ง Load/Store และ Immediate อยู่ที่ 15-0 สำหรับกรณี Load/Store, Branch, ALU แบบ Immediate ส่วน rd อยู่ที่ 15-11 สำหรับ R-Type ทั้งนี้ การออกแบบให้คล้ายๆ กันนี้เพื่อให้ง่ายต่อการถอดรหัสพิลด์ และลดความซับซ้อนของชาร์ดแวร์ถอดรหัส

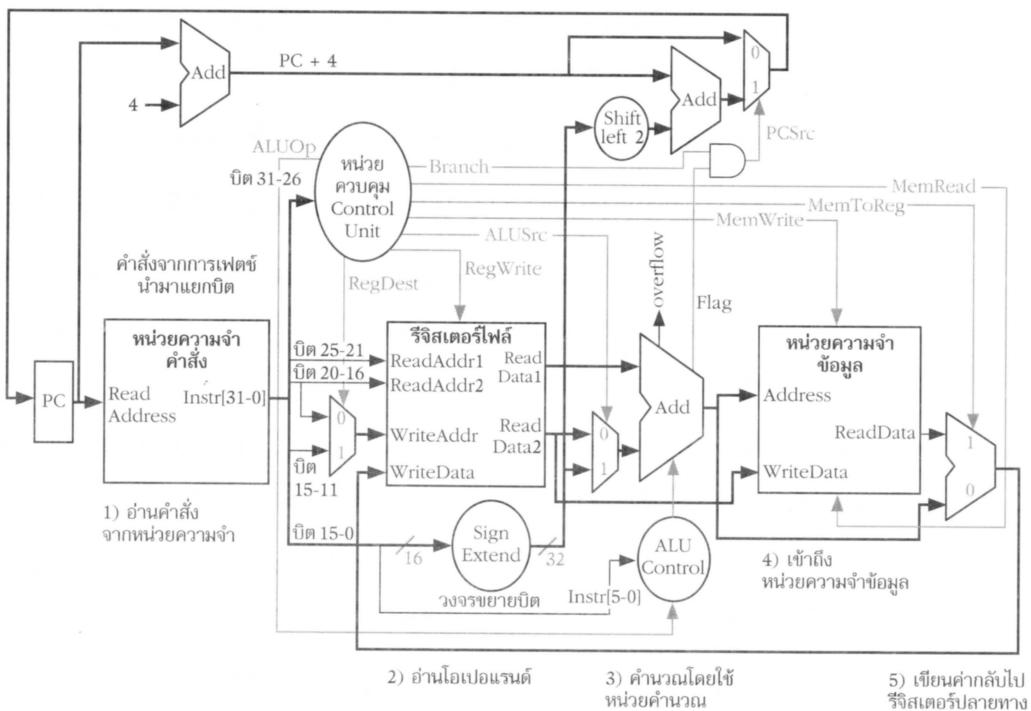
## ■ 4.2 การออกแบบเกลเดี้ยว

รูปที่ 4.9 เป็นการรวมแต่ละส่วนเข้าด้วยกันเป็น Data Path โดยรวมจากรูปที่ 4.3 ถึง รูปที่ 4.8 เข้าด้วยกัน ในที่นี้จะสร้างแบบใช้เกลเดี้ยว โดยให้การทำงานตั้งแต่การเฟตช์จนถึงการเอ็คซิคิวต์เสร็จลิ้นภายในหนึ่งรอบ โดยแบ่งเป็นขั้นตอนตามการใช้งานคำสั่งได้ดังนี้ เริ่มต้นจะทำการอ่านคำสั่งมาจากหน่วยความจำคำสั่งก่อน จากนั้นต่อมาคำสั่งจะถูกแยกสัญญาณและไปอ่าน รีจิสเตอร์ไฟล์ และรหัสคำสั่งส่งไปยังหน่วยควบคุมเพื่อถอดรหัสคำสั่ง และให้สัญญาณออกแบบควบคุม

หน่วยต่างๆ ขั้นที่สามจะเป็นขั้นตอนการใช้หน่วยคำนวณ ขั้นที่สี่เป็นขั้นการเข้าถึงหน่วยความจำข้อมูล และขั้นที่ห้าเป็นขั้นการเขียนค่ากลับตามรูป

หน่วยควบคุมจะส่งสัญญาณควบคุมหน่วยต่างๆ ตามลำดับขั้น กำหนดการให้ค่าดังนี้

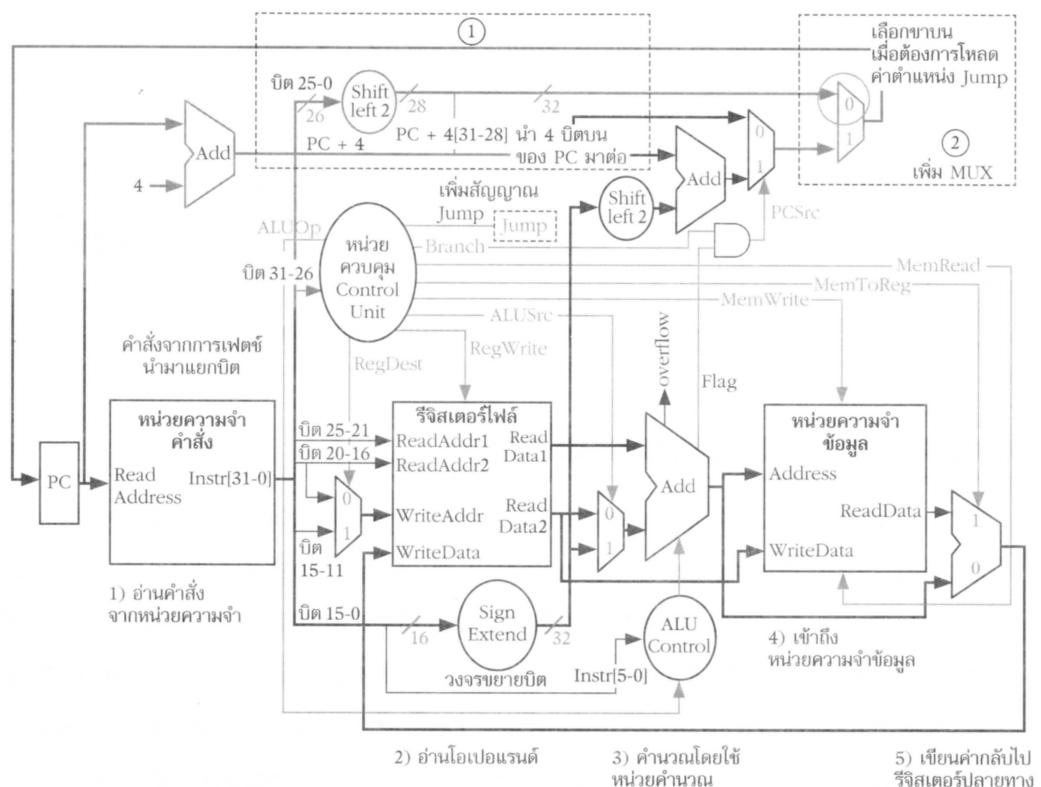
- ALUOp ควบคุมการทำงานของ ALU โดยจะถูกต่อครั้งตัวบวกหรือ ALUControl
- Branch จะให้ค่า 1 ในกรณีเป็นคำสั่ง Branch เพื่อควบคุมว่าจะกระโดดหรือไม่ โดยอาศัยการ AND และเป็นเงื่อนไขในปรับปรุงค่า PC ให้ใช้เป็น Branch Target ตัวที่ได้จากการเอา PC + Immediate
- ALUSrc ให้ค่า 0 เมื่อต้องการอ่านจาก Read Data 2 เข้าเป็นโอเปอเรนเดอร์ของ ALU และให้ค่าเป็น 1 ถ้าต้องการนำ Immediate เข้าไปยังโอเปอเรนเดอร์ข้างของ ALU
- RegDest ใช้เลือกช่องรีจิสเตอร์ที่จะเขียน ทั้งนี้ขึ้นกับประเภทคำสั่ง ถ้าเป็น R-Type อ่านจากคำสั่งบิต 15-11 ถ้าเป็น I-Type อ่านจากบิต 20-16
- RegWrite ใช้ควบคุมการเขียนรีจิสเตอร์ กรณีคำสั่ง R-Type หรือ Load กำหนดให้ RegWrite = 1
- MemWrite ใช้ควบคุมการเขียนหน่วยความจำ กรณีคำสั่ง Store ให้ค่า MemWrite = 1
- MemRead ใช้ควบคุมการอ่านหน่วยความจำ กรณีของคำสั่ง Load ให้ค่า MemRead = 1
- MemToReg ทำการเลือกข้อมูลที่จะเขียนไปยังรีจิสเตอร์ ถ้าเป็น MemToReg = 1 จะเลือกข้อมูลที่ได้จากหน่วยความจำข้อมูล ใช้สำหรับในกรณีของคำสั่ง Load และกรณีของ R-Type จะให้ค่า MemToReg = 0 โดยเลือกข้อมูลจาก ALUOutput มาเขียนไปยังรีจิสเตอร์ปลายทางระบุโดย Rd
- PCSrc เป็นสัญญาณที่ได้จากการ AND จากการตรวจสอบเงื่อนไขโดยดูจาก Zero Flag จากที่ได้จาก ALU ถ้าให้ค่า Zero = 1 แสดงว่าเงื่อนไขเป็นจริง และคำสั่งนั้นเป็นคำสั่ง Branch (สัญญาณ Branch = 1) ดังนั้น จะโหลดค่าตำแหน่งกระโดดปลายทางเข้าไปยัง PC และถ้า Zero = 0 จะโหลดค่า PC + 4 เข้าไปเป็นค่า PC ใหม่ ซึ่งจะไม่มีการกระโดดเกิดขึ้น



รูปที่ 4.9 Data Path แบบไนเกิลเดียร์ของ MIPS

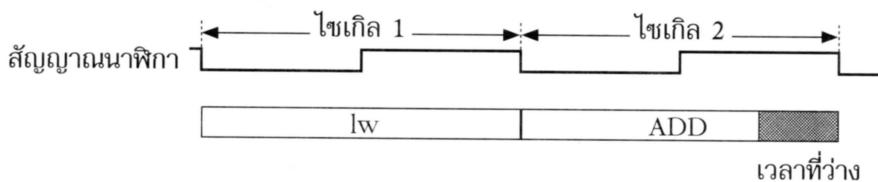
สัญญาณควบคุมเหล่านี้จะไปควบคุมจุดต่างๆ ในขั้นตอนการอ่านเข้ากีวิต ตัวอย่างเช่น การอ่านคำสั่ง Load หลังจากการเฟลด์ หน่วยควบคุมจะเลือกขาบนของ MUX โดยกำหนดให้ RegDest = 0 ซึ่งจะระบุรีจิสเตอร์ที่ต้องการอ่านไปใส่โดยบิตที่ 16-20 ของคำสั่ง และให้ RegWrite = 1 เพื่อบอกว่าต้องการเขียนรีจิสเตอร์ จากนั้น ALUSrc จะควบคุมโอเปอเรนด์ขาล่างของ ALU ให้อ่านมาจาก Immediate ในคำสั่งที่ได้จากบิต 0-15 ผ่านวงจรขยายบิต โดยจะมาจากการขาล่างของ MUX จากนั้นผลลัพธ์ที่ได้จะนำไปคำนวนด้วย ALU โดย ALUControl จะให้สัญญาณควบคุมเพื่อสั่งใน ALU ทำการบวกหั้ง 2 โอเปอเรนด์เข้าด้วยกัน เป็นการคำนวนผลดำเนินการที่จะอ่านข้อมูลในหน่วยความจำข้อมูลส่งไปยังสาย Address ของหน่วยความจำข้อมูล และข้อมูลที่อ่านได้จาก ReadData ถูกส่งไปยังรีจิสเตอร์ กำหนดโดยสัญญาณ MemtoReg = 1 สำหรับ MUX ตัวท้ายสุด สำหรับการเลือกแหล่งข้อมูลที่ส่งไปยังรีจิสเตอร์เพื่อเขียนสำหรับ MUX ตัวบนนั้น ให้สัญญาณ PCSrc = 0 เพื่อนำค่า PC + 4 มาเขียนใน PC ต่อไป

สำหรับคำสั่ง Jump ต้องเพิ่มองค์ประกอบตามกล่องดังแสดงในรูปที่ 4.10 เนื่องจากจะเพิ่มตัว MUX สำหรับเลือกตำแหน่งกระโดดปลายทางเพิ่มขึ้นมา สำหรับตำแหน่งกระโดดปลายทางนี้ จะหาได้จากพีล็อกบิต 25-0 และนำมาระบุ 2 บิตสุดท้าย และทำให้เป็น 32 บิต โดยการต่อ กับ 4 บิตบนของ PC + 4 ตัวในกล่องที่ 1) ในกล่องที่ 2) ตัว MUX จะเลือกงานเมื่อสัญญาณ Jump เท่ากับ 1 เพื่อให้ตำแหน่ง Jump ถูกโหลดไปในรีจิสเตอร์ PC



รูปที่ 4.10 Data Path แบบไชเกิลเดียวนีมีข้อเสียคือ สำหรับแต่ละคำสั่งจะใช้เวลาในการทำงานไม่เท่ากัน แต่เราต้องออกแบบ Cycle Time ให้ยาวเพียงพอที่จะรองรับคำสั่งที่ทำงานนานที่สุดได้ (เช่น คำสั่ง Load ในกรณีนี้) คำสั่ง Load จะใช้หน่วยคำนวนทุกตัวๆ ใน Data Path ตั้งแต่การอ่านหน่วยความจำคำสั่ง อ่านรีจิสเตอร์ คำนวน ALU อ่านหน่วยความจำข้อมูลและเขียนไปยังรีจิสเตอร์ แต่ว่าคำสั่งอื่นๆ เช่น R-Type (ADD) จะใช้หน่วยที่เกี่ยวข้องเพียง 4 หน่วย ได้แก่

หน่วยความจำคำสั่ง รีจิสเตอร์เพื่ออ่านค่าโอลูเมต์ ALU และรีจิสเตอร์เพื่อการเขียนค่าผลลัพธ์ และไม่ได้ใช้ส่วนของหน่วยความจำข้อมูล สำหรับคำสั่ง Jump จะใช้หน่วยสำคัญฯ เพียง 3 หน่วย เท่านั้น ดังนั้น การที่รอบไปเกลิที่ถูกกำหนดด้วยคำสั่งที่ยาวที่สุดนั้น ถ้าโปรแกรมทำงานคำสั่งอื่นที่ใช้เวลาอย่างกว่าระยะเวลาของรอบไปเกลิ การทำงานของคำสั่นจะมีเวลาที่เหลืออยู่ภายในรอบ ซึ่งเป็นเวลาที่ไม่สามารถทำงานอย่างอื่นได้ ดังนั้นการออกแบบที่ดีกว่ามีแนวคิดของการแบ่งเป็นไปเกลิสั้นๆ โดยใช้วิธีแบบหลายไปเกลินั่นเอง ตัวอย่างในรูปที่ 4.11 คำสั่ง lw ใช้เวลาเต็มไปเกลิ แต่การทำงานของคำสั่ง ADD ใช้เวลาอย่างกว่าประมาณ 1/4 ไปเกลิ

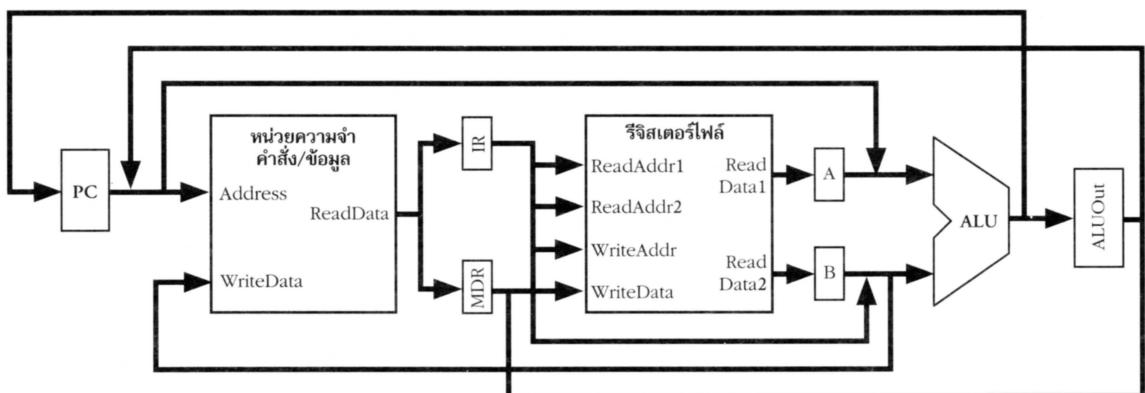


รูปที่ 4.11 เวลาในการอ่านข้อมูลที่ว่าง

### 4.3 การออกแบบแบบหลายไปเกลิ

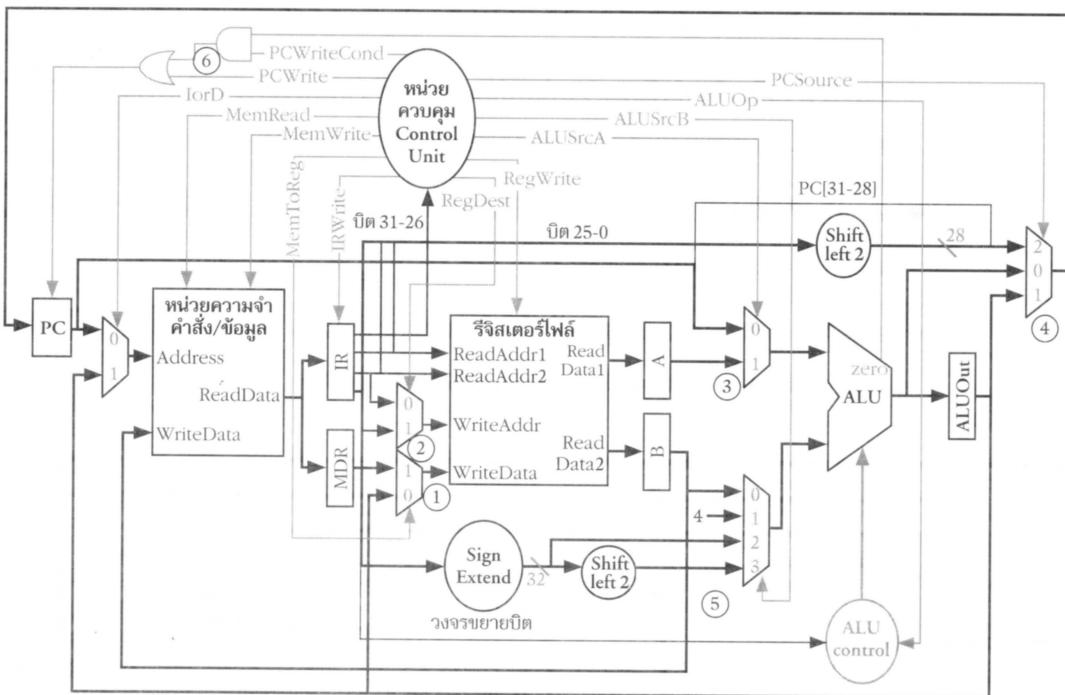
ในการออกแบบแบบหลายไปเกลิจะทำให้ Cycle Time สั้นลง แต่อย่างไรก็ต้องกำหนดให้ รายการเพียงพอที่จะให้หน่วยคำนวนหนึ่งๆ ทำงานเสร็จสิ้นได้ การออกแบบจึงมองที่แต่ละขั้นตอนของการทำงาน โดยพิจารณาว่าแต่ละขั้นนี้มีการใช้หน่วยคำนวนที่สำคัญอะไร การใช้แบบหลายไปเกลินี้จะทำให้แต่ละคำสั่งอาจจะใช้จำนวนไปเกลิในการทำงานไม่เท่ากัน นอกจากนี้ การทำงานแบบหลายไปเกลิจะทำให้เกิดการใช้ทรัพยากร่นอย่างต่างๆ ร่วมกันด้วย เช่น หน่วยความจำ อาจจะไม่จำเป็นต้องแยกเป็น 2 ส่วน เพราะใน 1 ไปเกลิจะทำการอ่านหรือเขียนหน่วยความจำอย่างเดียวเท่านั้น เป็นต้น แต่อย่างไรก็ต้องออกแบบแบบหลายไปเกลิต้องเพิ่มบันเฟอร์เพื่อเก็บค่าของข้อมูลชั่วคราว ที่เกิดขึ้นระหว่างไปเกลิเพื่อไปใช้ในไปเกลิถัดไป Data Path จึงต้องมีการเปลี่ยนแปลง และมีการเพิ่มรีจิสเตอร์ภายในสำหรับแต่ละส่วนของการทำงานให้อ่านค่าจากรีจิสเตอร์ภายใน ในตอนปลายของแต่ละไปเกลิจะมีการเขียนค่าลงไปรีจิสเตอร์ภายในตัว ดังแสดงในรูปที่ 4.12 จะเพิ่มรีจิสเตอร์ภายในได้แก่

- IR สำหรับเก็บค่าคำสั่งที่อ่านได้จากໄใชเกิลของการเฟตช์
- MDR สำหรับเก็บค่าข้อมูลจากໂອເປົອແຣນດ໌ທີ່ได้ຈາກໜ່ວຍຄວາມຈຳກຣົນຂອງคำสั่ง Load ແກ້ດັ່ງໃນໄໃເກີລຂອງການອ່ານໜ່ວຍຄວາມຈຳໃນການຮັນດຳສັ່ງ
- A, B สำหรับเก็บຄ່າໂອເປົອແຣນດ໌ທີ່ອ່ານໄດ້ຈາກຂັ້ນກາຣດີໂຄດ ເພື່ອໃຊ້ໃນກາເວັກຫີຕົວຕົ້ນຂອງ ALU ໃນໄໃເກີລຄັດໄປ
- ALUOutput สำหรับເກີບຜົລັບຜົວທີ່ເກີດຈາກການຄຳນວນ ALU ໄວເພື່ອຈະນຳໄປເຂົ້າໜ້າໂຮງ ສັ່ງໄປຢັງສ່ວນທີ່ເກີວ່າຂອງໃນການທຳການຂອງໄໃເກີລຄັດໄປ



รูปที่ 4.12 Data Path แบบหลายໄໃເກີລຂອງ MIPS

ในภาพรวม ສັງນູານຄວາມຄຸມຕ່າງກົນເປັນໄປ ມີກາເພີ່ມສັງນູານຄວາມຄຸມ ແລະ ເປັນແປງ Data Path ບາງສ່ວນໃຫ້ອ່ານຂໍ້ມູນມາຈາກສ່ວນຂອງຮີຈິສເຕົອຣ໌ຂ່າວຄວາມເຫຼັນເນື້ນເພື່ອທຳການ



รูปที่ 4.13 พิจารณาหน่วยควบคุม Data Path แบบหลายไบเกลชอง MIPS

มีการเปลี่ยนแปลงเทียบกับรูปที่ 4.10 ได้แก่ การใช้ MUX ขึ้นมา เช่น ในจุดที่ 1 จะเป็นการเลือกสัญญาณระบุการเขียนไปยังรีจิสเตอร์ เนื่องจากคำสั่ง Load (I-Type) และคำสั่ง ALU R-Type ทั้ง 2 คำสั่งเปลี่ยนไปยังรีจิสเตอร์ทั้งสิ้น แต่ถ้าเป็นคำสั่ง Load ข้อมูลที่เขียนจะมาจากรีจิสเตอร์ MDR ซึ่งอ่านจากหน่วยความจำในรอบที่แล้ว และคำสั่งแบบ R-Type ข้อมูลที่เขียนจะมาจาก ALUOutput ที่ได้จากการคำนวณของ ALU ในรอบที่แล้ว และในจุดที่ 2 เป็น MUX ตัวเดียว ดังแสดงในรูปที่ 4.11

ในจุดที่ 3 MUX ที่อยู่ที่โอเปอเรนเดอร์ของ ALU ใช้เลือกที่มาของโอเปอเรนเดอร์ตัวแรกของ ALU ว่ามาจาก A หรือมาจาก PC ในจุดที่ 4 จะเป็นตัว MUX เพื่อส่งสัญญาณออกไปเขียนทับไปในรีจิสเตอร์ PC ซึ่งอาจจะมาจากการนี้

- คำสั่ง Jump ให้ค่าสัญญาณควบคุมเป็น 2
- คำสั่ง Branch ให้ค่าสัญญาณควบคุมเป็น 1
- คำสั่งอื่นๆ ที่ต้องการเพียงค่าเดิม  $PC + 4$  ให้ค่าสัญญาณควบคุมเป็น 0 เป็นการเลื่อนไปทำคำสั่งถัดไป

ในจุดที่ 5 MUX ที่อยู่ขาล่างใช้เลือกโวเปอแรนด์ตัวที่สองของ ALU ว่ามาจากแหล่งใด มีทั้งหมด 4 กรณี

- จาก B สำหรับกรณีทำ  $A + B$  ให้ค่าสัญญาณควบคุมเป็น 0
- ค่าคงที่ 4 สำหรับกรณีทำ  $PC + 4$  ให้ค่าสัญญาณควบคุมเป็น 1
- เป็น Immediate สำหรับ  $A + Immediate$  สำหรับคำสั่งแบบ ALU ที่มีโวเปอแรนด์เป็น Immediate ให้ค่าสัญญาณควบคุมเป็น 2
- เป็น Immediate สำหรับจากคำสั่ง Branch ในกรณีของ I-Type ที่ต้องการเอา PC ปัจจุบัน + Immediate เพื่อคำนวนตำแหน่งที่จะกระโดด กรณีนี้ให้ค่าสัญญาณควบคุมเป็น 3

การออกแบบข้างต้นนี้จะทำให้เกิดกรณีต่างๆ ของโวเปอแรนด์ขาล่างดังนี้คือ

- $A + Immediate$  กรณีคำสั่งเกี่ยวกับ ALU กับ Immediate หรือ Load/Store
- $A + B$  กรณี R-Type
- $PC + 4$  เพิ่มค่า PC
- $PC + Immediate$  (Shift Left 2) กรณีคำสั่ง Branch

สำหรับจุดที่ 6 เป็นเงื่อนไขสำหรับควบคุมการเขียนค่าใน PC เนื่องจากการเปลี่ยนแปลงค่าใน PC จะไม่ได้ทำทุกรอบสัญญาณนาฬิกา จะทำสำหรับแต่ละคำสั่งเท่าไร ดังนั้น จะใช้สัญญาณควบคุมจาก PCWrite, PCWriteCond และ Zero Flag จาก ALU ร่วมกัน โดย PCWrite จะเป็นตัว Enable การเขียน PC แต่ PCWriteCond จะเป็น 1 สำหรับคำสั่งแบบ Branch และใช้ร่วมกับ Zero Flag (ซึ่งจะเป็น 1 เมื่อเปรียบเทียบกับโวเปอแรนด์แล้วได้ผลเท่ากัน) ซึ่งจะหมายถึงจะ Branch เมื่อค่าโวเปอแรนด์เท่ากันนั่นเอง

ถ้าเรามองจะแบ่งการทำงานออกเป็นขั้นตามลักษณะการใช้งานว่ายที่สำคัญ จะได้ดังนี้

- ขั้นเฟตช์ ใช้หน่วยความจำ และทำหน้าที่อ่านคำสั่งจากหน่วยความจำและเพิ่มค่า PC เป็น  $PC + 4$
- ขั้นตีโค้ด ให้รีจิสเตอร์ไฟล์ และทำหน้าที่ถอดรหัสคำสั่ง อ่านโวเปอแรนด์จากรีจิสเตอร์ และขยายขนาดโวเปอแรนด์

- ขั้นເອົກນີ້ຄົວຕໍ່ໃຊ້ ALU ແລະຈະทำการทำงานคำสັ່ງດ້າເປັນຮູບແບບ R-Type ແລະດ້າເປັນຮູບແບບ I-Type ຈະทำการคำນວณຕຳແໜ່ງປລາຍທາງທີ່ຈະກະໂດດ ຕຳແໜ່ງໜ່ວຍຄວາມຈຳທີ່ຈະອ່ານຫຼືເບີຍ ເບີຍນເທິຍບເ່ອນໄຟ ດຳສັ່ງ Branch ແລະດຳສັ່ງ Jump ຈະເສົ່າງສິນໃນໜັນນີ້
- ขັ້ນເຂົາຄຶງໜ່ວຍຄວາມຈຳ (Memory Access) ໃຊ້ໜ່ວຍຄວາມຈຳ ສໍາຫັບດຳສັ່ງ Load ຈະການອ່ານໜ່ວຍຄວາມຈຳ ກຣີນີ້ດຳສັ່ງ Store ເບີຍນໜ່ວຍຄວາມຈຳໃຫ້ເສົ່າງສິນ ແລະເບີຍຜລັກພົ້ມຢັ້ງຮົງສົດເຕົອຣີໄຟລ໌ສໍາຫັບກຣີນີ້ດຳສັ່ງ R-Type
- ขັ້ນເບີຍນຳກັບລັບ (Write Back) ໃຫ້ຮົງສົດເຕົອຣີ ໃນກຣີນີ້ດຳສັ່ງ Load ຈະເບີຍນຳກຳທີ່ອ່ານຈາກໜ່ວຍຄວາມຈຳໄປຢັ້ງຮົງສົດເຕົອຣີ ໃນກຣີນີ້ດຳສັ່ງ ALU R-Type ຈະເບີຍຜລັກພົ້ມຢັ້ງຮົງສົດເຕົອຣີທີ່ເກັບຜລັກພົ້ມ

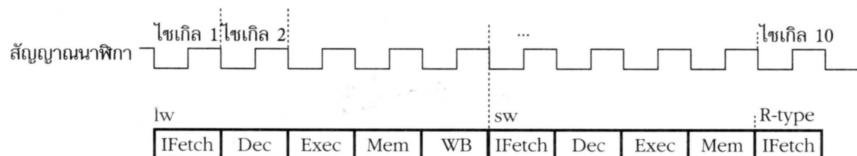
ตารางຕ່ອງໄປນີ້ສຸບປັນດອນການການທຳງານໆ ຊ່ອງແຮກແສດງວ່າສໍາຫັບດຳສັ່ງ ALU R-type ຕັ້ງທຳອະໄຣໃນແຕ່ລະໜັ້ນ ຂ່ອງຄົດໄປສໍາຫັບດຳສັ່ງ Load/Store ຂ່ອງທີ່ 3 ພາຍໃຕ້ ດຳສັ່ງ Branch ແລະ ຂ່ອງສຸດທ້າຍສໍາຫັບດຳສັ່ງ Jump

ໜັນດອນ	ດຳສັ່ງ ALU ແບບ R-Type	ດຳສັ່ງ Load/Store	ດຳສັ່ງ Branch	ດຳສັ່ງ Jump
ເພື່ອຄົດດຳສັ່ງ	ອ່ານດຳສັ່ງຮະບຸໂດຍ PC ໄສໃນ IR $PC = PC + 4$			
ສົດເຕົອຣີ	ອ່ານໂອເປອແຮນດັວງແຮກຈາກ IR ປີຕີ່ 25-21 ໃສ່ຮົງສົດເຕົອຣີ A ອ່ານໂອເປອແຮນດັວງທີ່ສອງຈາກ IR ປີຕີ່ 20-16 ໃສ່ຮົງສົດເຕົອຣີ B ການກາຍຍາຍ Immediate ໃນປີຕີ່ 15-0 ຂອງ IR ແລະ Shift ຫ້າຍໄປ 2 ຕຳແໜ່ງ ແລະນຳໄປບວກກັບຄ່າໃນ PC ເກັບຄ່າທີ່ໄດ້ໃສ ALUOut			
ເອົກນີ້ຄົວຕໍ່	ກະທຳ Operation ທີ່ ກຳຫັດກັນ A, B ເກັບ ຜລັກພົ້ມໄສ ALUOut	ການກາຍຍາຍບີຕີ່ Immediate ໃນ IR ປີຕີ່ 15-0 ແລ້ວນຳໄປ ບວກກັບ A ເກັບຜລັກພົ້ມ ໄວ້ທີ່ ALUOut	ເບີຍເທິຍບີຕີ່ A = B ດ້າໃໝ່ ກຳຫັດ PC = ALUOut	ນຳປີຕີ່ 25-0 ຂອງ IR ເລື່ອນໄປທາງ ຂວາ 2 ປີຕີ່ ມາຕ່ອ ກັບນຳເອາ 4 ປີຕີ່ ນຳອອງ PC

ขั้นตอน	คำสั่ง ALU แบบ R-Type	คำสั่ง Load/Store	คำสั่ง Branch	คำสั่ง Jump
เข้าถึง หน่วย ความจำ	นำค่าใน ALUOut ไป เก็บในรีจิสเตอร์ระบุ โดยบิต 15-11 ของ IR	อ่านค่าจากหน่วย ความจำตำแหน่ง ระบุโดย ALUOut ใส่ใน MDR  หรือนำค่าใน B ไป เก็บในหน่วยความ จำตำแหน่ง ระบุโดย ALUOut		
เพียน ค่ากลับ		นำค่าใน MDR ไป เพียนในรีจิสเตอร์ ระบุโดยบิต 20-16 ของ IR		

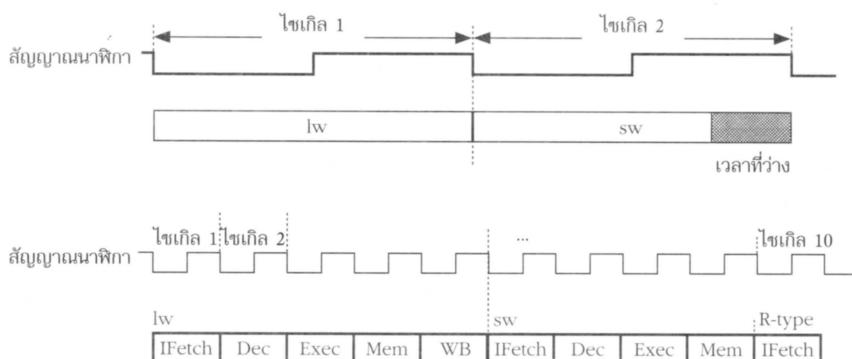
จะเห็นว่าทุกคำสั่งจะทำงานขั้นตอนเฟตช์และดีโคดเหมือนกัน จากนั้นในขั้นເອົກຫີ່ວິດ คำสั่ง ALU แบบ R-Type จะทำເອົກ 2 ขั้นตอน แต่คำสั่ง Load/Store จะทำທັງໝາດ 5 ขั้นตอน สำหรับ คำสั่ง Branch และ Jump จะทำเพียง 3 ขั้นตอนเท่านั้น ดังนั้น จำนวนໄຊເກີລຂອງແຕ່ລະປະເກດ คำສັ່ງຈຶ່ງໃຫ້ໄມ່ເທົກກັນ

ในการทำงานแบบหลายໄຊເກີລນີ້ จะทำໃຫ້เวลาในการรันແຕ່ລະคำສັ່ງຕ່າງກັນ ເພື່ອ คำสั่ง Load ຈະໃໝ່ 5 ໄຊເກີລ ແຕ່คำສັ່ງ Store ຈະໃໝ່ 4 ໄຊເກີລ ດຳເນີນ R-Type ຈະໃໝ່ 4 ໄຊເກີລ Jump/Branch ໃຊ້ 3 ໄຊເກີລ ເປັນຕົ້ນ ຮູບທີ 4.14 ແສດຕ້ວອຍໆຈໍານວນໄຊເກີລທີ່ໃໝ່ ໃນຮູບ IFetch ຍ່ອມາຈາກขັ້ນເຟັ້ນ Dec ຍ່ອມາຈາກขັ້ນດີໂຄດ Exec ຍ່ອມາຈາກขັ້ນເອົກຫີ່ວິດ Mem ຍ່ອມາຈາກขັ້ນເຂົ້າເຖິງໜ່ວຍຄວາມຈຳ ແລະ WB ຍ່ອມາຈາກขັ້ນເພີ້ນຄ່າກຳລັບ



ຮູບທີ 4.14 ເນື້ອດຳເນີນແຕ່ລະປະເກດ

ถ้าเทียบกับแบบไม้เกลเดี่ยวที่ใช้ 1 ไม้เกลต่อ 1 คำสั่ง จะพบว่าจะมีเวลาเหลือในแต่ละไม้เกล เมื่อรวมเวลาที่สูญเสียไป จุดนี้อาจจะเป็นเวลาที่มากและไม่สามารถใช้เวลาให้เป็นประโยชน์ได้ แต่ โดยปกติแล้ว Cycle Time แบบไม้เกลเดี่ยวจะไม่เป็น 5 เท่าของแบบหลายไม้เกลเสมอไป เนื่องจากในการออกแบบหลายไม้เกลนั้น จะมีเวลาเพิ่มเติมที่เป็น Overhead ระหว่างไม้เกลที่เกิดเนื่องจาก การจัดเก็บค่าและการอ่านค่าจากregisterภายในต่าง ๆ เพราะว่าใน Data Path มีการเปลี่ยนแปลง และมีการเพิ่มเติมรีจิสเตอร์เก็บค่าข้อมูลต่างๆ ดังแสดงในรูปที่ 4.15 เพื่อบ่งเวลาของห้องสองรูปแบบ



รูปที่ 4.15 เพียบสัญญาณนาฬิกาของแบบไม้เกลเดี่ยว กับแบบหลายไม้เกลของ MIPS รูปล่างแบบหลายไม้เกล

ในรูปที่ 4.15 จะเห็นว่าแบบไม้เกลเดี่ยวจะมีเวลาภายในไม้เกลที่ว่าง ในการนีของคำสั่ง Store เพราะไม่ได้ใช้ทุกหน่วย ในรูปล่างจะเห็นว่าคำสั่ง Load ใช้ 5 หน่วย ใช้เวลารวม 5 ไม้เกล และ คำสั่ง Store ใช้ 4 หน่วย ใช้เวลาเป็น 4 ไม้เกล แต่ว่าจะเห็นว่ากรณี 5 ไม้เกลรวมกันของรูปล่างจะ ยาวกว่า 1 รอบของรูปบน เนื่องจาก Overhead ต่างๆ จากบัฟเฟอร์ชั่วร้าวที่กล่าวมาข้างต้น ในการกำหนดให้ 1 รอบทำงานเพียง 1 คำสั่ง เราจะได้ว่า CPI = 1 ทุกคำสั่ง แต่ในกรณีของหลายไม้เกล ค่า CPI ของคำสั่งแต่ละประเภทจะไม่เท่ากัน ในตัวอย่างนี้ค่า CPI ของคำสั่ง lw เท่ากับ 5 แต่ของ คำสั่ง Store เท่ากับ 4 เป็นต้น ลองพิจารณาตัวอย่างต่อไปนี้

ถ้าพิจารณาคำสั่งของคำสั่ง Load, ADD, SUB, Store, Branch หากใช้แบบไม้เกลเดี่ยว จะใช้เวลา 5 ไม้เกล

แต่ถ้าใช้แบบหลายไม้เกล จะใช้  $5 + 4 + 4 + 4 + 3 = 20$  ไม้เกล พิจารณาระบบแบบไม้เกลเดี่ยว ทำให้มี Cycle Time เท่ากับ 10 นาโนวินาที แต่แบบหลายไม้เกลจะมี Cycle Time เป็น

2.2 นาโนวินาที ดังนั้น แบบปั๊กเกิลเดี่ยวจะใช้เวลา 50 นาโนวินาที และแบบหลายปั๊กเกิลใช้เวลา 44 นาโนวินาที ซึ่งจะให้ Speedup เท่ากับ 1.14

## 4.4 สรุป

ในการออกแบบ Data Path ภายใน ต้องคำนึงถึงหลายๆ ปัจจัย ตั้งแต่รูปแบบคำสั่ง หน่วยคำนวนที่ใช้ รีจิสเตอร์ เป็นต้น ในบทนี้กล่าวถึงหน่วยคำนวนที่จำเป็นในการวางแผนร่าง จากหน่วยคำนวนและรีจิสเตอร์นำมาเชื่อมตอกันเป็นทางเดินของข้อมูลสำหรับแต่ละขั้นตอน และแต่ละรูปแบบของคำสั่ง และเมื่อนำทุกรูปแบบมารวมกันทั้งหมดจึงได้ Data Path รวมออกแบบมา ในบทนี้ได้กล่าวถึงการกำหนดลักษณะของสัญญาณพิเศษ การออกแบบในลักษณะปั๊กเกิลเดี่ยวและลักษณะหลายปั๊กเกิล และผลต่อเวลาในการทำงานต่อหนึ่งคำสั่ง

## คำถ้ามก้าวยบท

- หน่วยคำนวนในบทนี้หมายถึงอะไร ได้แก่อะไรบ้าง
- พิจารณาการทำงานของ  $A = A + B$  งพิจารณาว่า Data Path ของโค้ดนี้ควรประกอบด้วย หน่วยคำนวนและการเชื่อมโยงอย่างไร
- รูปแบบการทำงานแบบไหนเกิดเดี่ยวต่างกันแบบหลายไชเกิลอย่างไร
- ถ้าเราเปลี่ยนให้ ALU สามารถทำการให้ Flag LT (น้อยกว่า), GT (มากกว่า) ได้ด้วย คำสั่งแบบใดจะได้ประโยชน์จาก Flag เหล่านี้บ้าง
- พิจารณาเราทำงานคำสั่ง MIPS ต่อไปนี้จำนวน 1,000 รอบ

Lw

Add

Sw

ถ้าทำงานในรูปแบบไชเกิลเดี่ยวจะรับด้วยความยาวสัญญาณนาฬิกา 2.0 นาโนวินาที แต่ถ้าทำงานด้วยรูปแบบหลายไชเกิล ความยาวสัญญาณนาฬิกาจะเป็น 0.5 นาโนวินาที อย่างทรายว่ารูปแบบไหนจะทำงานเร็วกว่ากัน และเร็วกว่ากี่เท่า

- คำสั่งใดใน MIPS ที่ใช้ CPI น้อยที่สุดในกรณีของหลายไชเกิล จงอธิบาย
- พิจารณาโครงสร้าง Data Path ของ MIPS หากต้องเพิ่ม ALU ที่ใช้การคำนวนเป็น 2 ตัว จงอธิบายและแสดงเค้าโครงกราฟเปลี่ยนแปลง เค้าโครงกราฟเปลี่ยนแปลงดังกล่าวจะมีผลผลกระทบต่อตัวแปรอะไรในสมการ CPU time บ้าง
- พิจารณาการเพิ่มคำสั่ง Add ที่ต้องการโอเบอร์เอนด์ด้านซ้ายเป็นเรจิสเตอร์ และด้านขวาเป็นหน่วยความจำ เช่น

Add rd, rs, 100(rt)

- 8.1 จงอธิบายการเปลี่ยนแปลง Data Path ในขั้นตอนต่างๆ เพื่อทำให้รองรับคำสั่งดังกล่าว
- 8.2 ถ้าการเปลี่ยนแปลงดังกล่าวทำให้รอบลัญญาณยาวขึ้น 1.3 เท่า พิจารณาตารางด้านล่าง

ประเภทคำสั่ง	ความถี่	CPI (ไซเกิล)
ประเภท A	30%	2
ประเภท B	25%	4
ประเภท C	30%	1
ประเภท D	15%	3

สมมติว่าคำสั่งดังกล่าวในประเภท A สามารถเปลี่ยนเป็นคำสั่งรูปแบบนี้จำนวนครึ่งหนึ่ง คำสั่งแบบนี้จะลดคำสั่ง ประเภท B ไปจำนวน เท่ากัน และคำสั่งแบบนี้ ใช้เวลา 3 ไซเกิล อย่างทราบว่าการเปลี่ยนแปลงนี้จะทำให้ตัดแบบใหม่นี้เร็วขึ้นหรือไม่ แบบไหนเร็วกว่าในปริมาณเท่าไร ถ้าไม่เร็วกว่าเดิมจะมีโอกาสที่จะทำให้เร็วกว่าเดิมหรือไม่ อธิบายในเบื้องตัวเลข

9. พิจารณาตัวอย่างของ MIPS ที่ประกอบด้วย Data Path 5 ขั้นตอน ได้แก่ Fetch, Decode, Execute, Memory Access และ Write Back นั้น พิจารณาโปรแกรมที่ประกอบด้วยคำสั่ง LW, LW, ADD, SUB, SW, BEQ

9.1 พิจารณาลักษณะการทำงานแบบไชเกิลเดียว ถ้าโปรแกรมนี้ทำงานจะใช้ทั้งหมดกี่ไชเกิล ถ้าทำงานบนเครื่องคอมพิวเตอร์ที่มีความถี่เท่ากับ 100 เมกะเฮิรตซ์ จะใช้เวลาเท่าไร

9.2 พิจารณาโปรแกรมดังกล่าว ถ้านำมาไปทำงานบนระบบแบบหลายไชเกิล ซึ่งจะแบบการทำงานในแต่ละขั้นตอนเป็น 1 ไชเกิล การใช้ Data Path แบบหลายไชเกิลนี้จะเพิ่ม Overhead ในแต่ละไชเกิลไป 25% ดังนั้นถ้าโปรแกรมนี้ทำงานจะใช้ทั้งหมดกี่ไชเกิล และถ้าเครื่องนี้ปรับปรุงมาจากเครื่องในข้อที่ 9.1 เครื่องนี้จะมีความถี่เท่าไร และใช้เวลาทั้งหมดเท่าไรในการทำงานสำหรับโปรแกรมดังกล่าว

9.3 ระบบแบบหลายไชเกิลและไชเกิลเดียวในตัวอย่างนี้ แบบใดดีกว่ากันสำหรับโปรแกรมนี้ อธิบาย

10. พิจารณาการออกแบบรูปแบบไฟเกลเดี่ยวเบรียบเทียบกับแบบหลายไฟเกล สมมติว่าในการออกแบบโดยใช้แบบไฟเกลเดี่ยว ประกอบด้วยขั้นตอนต่างๆ ได้แก่ การเฟตซ์ การตีโตัด และการเอ็กซ์คิวต์ โดยในการเฟตซ์จะทำการอ่านคำสั่งมาจากหน่วยความจำโปรแกรม 1 คำสั่ง และเพิ่มค่าในรีจิสเตอร์ PC ในการตีโตัดจะถอดรหัสคำสั่งโดยเข้าถึงรีจิสเตอร์ไฟล์ในการอ่านโอเปอเรนต์สำหรับการเอ็กซ์คิวต์ จะใช้ ALU ในการประมวลผลคำสั่งนั้น และเปลี่ยนผลลัพธ์ของการทำงานไปยังรีจิสเตอร์ตัวที่ระบุในคำสั่ง รวมไปถึงการเข้าถึงหน่วยความจำข้อมูลถ้าคำสั่งนั้นๆ ต้องการอ่านหรือเขียนหน่วยความจำข้อมูล

10.1 งบปรับปรุง Data Path ของ MIPS และวัด Data Path ใหม่สำหรับกรณีนี้

10.2 ถ้าให้การเข้าถึงหน่วยความจำข้อมูลใช้เวลา 3 นาโนวินาที การเข้าถึงรีจิสเตอร์ (ห้องเปลี่ยนและอ่าน) ใช้เวลา 1 นาโนวินาที การเข้าถึงหน่วยความจำข้อมูลใช้เวลา 3 นาโนวินาที การคำนวณใน ALU ใช้เวลา 2 นาโนวินาที อย่างทราบว่ารูปแบบไฟเกลเดี่ยวจะมีความถี่เท่าไร

10.3 จากข้อที่ 10.2 เปลี่ยนการทำงานแบบเป็นแบบหลายไฟเกล ให้วัด Data Path ใหม่

10.4 จากข้อที่ 10.2 ถ้าแต่ละไฟเกลมี Overhead เพิ่มขึ้น 20% ในแต่ละรอบสัญญาณนาฬิกา รูปแบบหลายไฟเกลนี้จะมีความยาวรอบสัญญาณนาฬิกาเป็นเท่าไร

11. พิจารณาการปรับปรุง MIPS Data Path ดังนี้

- เพิ่มหน่วย ALU เป็น 2 หน่วยเพื่อให้ทำงานพร้อมกันได้
- เพิ่มหน่วยความจำข้อมูล เป็น 2 หน่วยให้อ่านเขียนข้อมูลได้พร้อมกัน

จงวัดและอธิบายการเปลี่ยนแปลง Data Path ของทั้ง 2 กรณี โดยไม่ต้องคำนึงถึงสัญญาณควบคุมที่จะเพิ่มขึ้นมา

