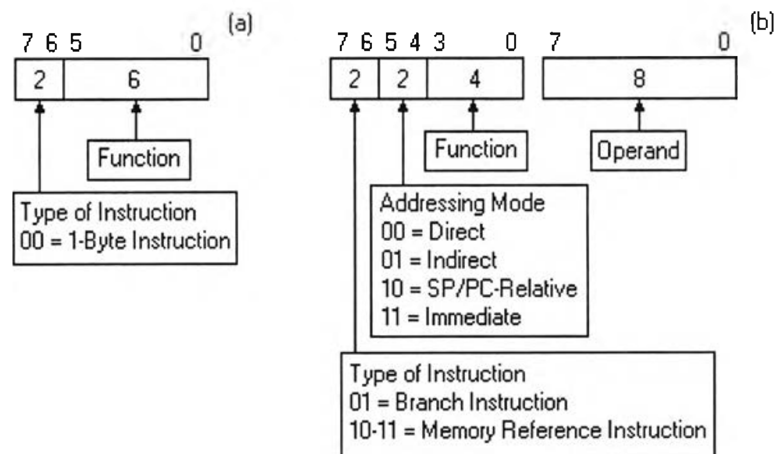
บทที่ 3

การออกแบบส่วนเส้นทางข้อมูล

ไมโครโพรเซสเซอร์โดยทั่วๆ ไปแล้วจะประกอบด้วย 2 ส่วนหลักๆ คือ ส่วนเส้นทางข้อมูลและส่วนควบคุม ในบทนี้จะกล่าวถึงการออกแบบในส่วนเส้นทางข้อมูลของไมโครโพรเซสเซอร์ ซึ่งเป็นส่วนที่ประกอบไปด้วยหน่วยคำนวณและตรรกะ (Arithmetic and Logic Unit – ALU) รีจิสเตอร์ (Register) ต่างๆ และบัส (Bus) โดยในส่วนนี้จะเป็นส่วนที่ทำหน้าที่ทุกอย่างตามคำสั่งที่ได้รับมาจากส่วนควบคุม ดังนั้นจึงต้องมีส่วนประกอบทุกส่วนอย่างครบถ้วนเพื่อให้สามารถทำงานตามชุดคำสั่งที่กำหนดไว้ในไมโครโพรเซสเซอร์ได้ ส่วนเส้นทางข้อมูลก็คือส่วนทำงานนั่นเอง

3.1 การออกแบบรูปแบบคำสั่ง

ไมโครโพรเซสเซอร์ที่ออกแบบนี้มีขนาด 8 บิต มีชุดคำสั่งต่างๆ ดังที่แสดงไว้ในภาคผนวก ก โดยที่คำสั่งในกลุ่ม Memory reference instructions และ Branch instructions นั้นเป็นคำสั่ง 2 ไบต์ โดยที่ไบต์แรกเป็นรหัสคำสั่ง (Opcode) และไบต์ที่สองเป็นโอเปอแรนด์ (Operand) ส่วนคำสั่งในกลุ่ม Miscellaneous instructions เป็นคำสั่งที่ใช้รหัสคำสั่งเพียงไบต์เดียวไม่มีโอเปอแรนด์ คำสั่งที่กระทำการเกี่ยวกับหน่วยความจำนั้นจะทำกับค่าที่อยู่ในรีจิสเตอร์ Acc และหลังจากนั้นก็เก็บผลลัพธ์ไว้ที่ รีจิสเตอร์ Acc สรุปการทำงานของคำสั่งทั้งหมดได้แสดงไว้ที่ภาคผนวก ข ส่วนรูปแบบของการกำหนดรหัสคำสั่งเป็นไปดังรูปที่ 3.1 รหัสคำสั่งทั้งหมดแสดงไว้ที่ภาคผนวก ค



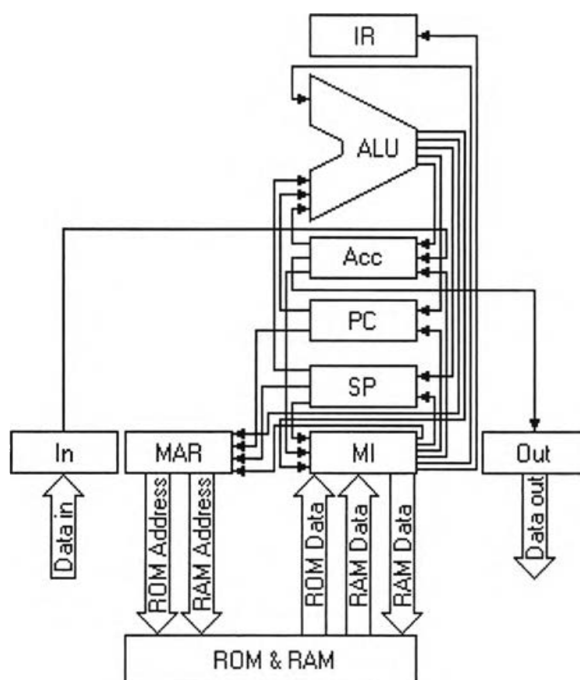
รูปที่ 3.1 รูปแบบของการกำหนดรหัสคำสั่ง (a) 1-Byte Instruction (b) 2-Byte Instruction

การทำงานจะใช้วิธีเก็บรหัสคำสั่งของโปรแกรมที่จะใช้งานอยู่ใน ROM (Read-Only Memory) แล้วอ่านค่ามาจาก ROM ทีละ 8 บิต หรือ 1 ไบต์เพื่อทำงาน และการทำงานใดๆ หากมีการเขียนค่าก็บันทึกลงสู่ RAM (Random Access Memory) โดยที่ทั้ง ROM และ RAM ต่างก็มีขนาดอ้างอิงได้ 256 ไบต์ หรือ 8 บิต พื้นที่ใน RAM ตำแหน่งที่ E0-FF (224-255) จะใช้งานเป็นสแต็ก (Stack)

บัสที่ใช้งานในไมโครโพรเซสเซอร์นี้จะไม่ใช้บัสร่วม (Common Bus) แต่จะใช้งานเป็นบัสแบบจุดต่อจุดแทน เพราะเหตุว่าการออกแบบด้วยโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือนจำต้องให้เป็นไปตามกฎ Isochronic Fork [11] ดังนั้นถ้าออกแบบด้วยการใช้บัสร่วมจะทำให้เป็นไปตามกฎ Isochronic Fork ทำได้ไม่สะดวกนัก การใช้บัสแบบจุดต่อจุดจึงเป็นทางออกที่ดีในการออกแบบ

3.2 โครงสร้างของส่วนเส้นทางข้อมูล

เนื่องจากไมโครโพรเซสเซอร์ตัวนี้อาศัยพื้นฐานการออกแบบมาจากโครงสร้างของไมโครโพรเซสเซอร์ไทแทกที่ได้แสดงไว้ที่รูป 1.1 ส่วนเส้นทางของมูลของไมโครโพรเซสเซอร์จึงอ้างอิงตามไมโครโพรเซสเซอร์ไทแทก ซึ่งแสดงไว้ดังรูปที่ 3.2 โครงสร้างส่วนใหญ่เกือบทั้งหมดยังคงเป็นแบบไทแทก แต่มีการเพิ่มเติมการใช้งาน ROM เข้ามา โดยใช้ ROM เก็บโปรแกรมที่จะใช้งาน เมื่อต้องการใช้งานโปรแกรมใดๆ ก็เพียงแต่เปลี่ยน ROM ที่บรรจุชุดคำสั่งให้เป็นโปรแกรมที่ต้องการ



รูปที่ 3.2 ส่วนเส้นทางข้อมูลของไมโครโพรเซสเซอร์

รูปที่ 3.2 แสดงส่วนเส้นทางข้อมูลของไมโครโพรเซสเซอร์ โดยที่สายสัญญาณแต่ละเส้นแทนเส้นทางข้อมูลขนาด 8 บิต ส่วนประกอบที่ทำหน้าที่ต่างๆ มีดังต่อไปนี้

1. รีจิสเตอร์ IR (Instruction Register) ทำหน้าที่เก็บรหัสคำสั่งหลังจากขั้นตอนการ Fetch กล่าวคือเมื่อ Fetch รหัสคำสั่งมาได้แล้วก็จะเก็บค่าไว้ที่นี่แล้วส่งต่อไปที่ส่วนควบคุมเพื่อเข้าสู่ขั้นตอนการ Decode และ Execute ต่อไป

2. หน่วยคำนวณและตรรกะ (ALU) ทำหน้าที่ทำงานตามคำสั่งเกี่ยวกับการคำนวณและตรรกะ ซึ่งอยู่ในกลุ่มคำสั่ง Memory reference instructions และใช้เพิ่มค่าให้กับรีจิสเตอร์ PC ด้วย ในหน่วยคำนวณและตรรกะจะมีรีจิสเตอร์ชั่วคราว (Temporary Register) อยู่ภายในตัวหนึ่งไว้เพื่อเก็บค่าผลลัพธ์ก่อนที่จะป้อนค่ากลับค่าผลลัพธ์จริงสู่รีจิสเตอร์ปลายทางต่อไป และมีแลตช์ (Latch) สำหรับเก็บค่า Flag ไว้ชั่วคราวด้วย เพื่อรอการป้อนค่ากลับเช่นเดียวกัน

3. รีจิสเตอร์ Acc (Accumulator) เป็นรีจิสเตอร์หลักในการทำงานส่วนใหญ่ และเป็นที่เก็บผลลัพธ์จากการทำงานด้วย กล่าวคือเมื่อทำงานเสร็จแล้วก็เก็บค่าไว้ที่นี่

4. รีจิสเตอร์ PC (Program Counter) เป็นรีจิสเตอร์ที่เป็นตัวชี้ตำแหน่งปัจจุบันของหน่วยความจำ ROM ของคำสั่งที่กำลังทำงานอยู่ โดยเมื่อเริ่มทำงานจะชี้ที่ตำแหน่งที่ 0 เสมอ การเปลี่ยนค่า PC จะใช้หน่วยคำนวณและตรรกะคำนวณ จากนั้นก็ส่งค่ากลับมาที่รีจิสเตอร์ PC ซึ่งการคำนวณค่า PC นี้จะไม่มีผลกระทบต่อ Flag ใดๆ

5. รีจิสเตอร์ SP (Stack Pointer) เป็นตัวชี้ตำแหน่งบนสุดของสแต็ก (Top-of-stack) ซึ่งสแต็กนั้นจะใช้เป็นพื้นที่ RAM ตำแหน่ง E0-FF (224-255) โดยตำแหน่งเริ่มต้นคือตำแหน่ง E0 เมื่อมีการเก็บค่าใดๆ ลงสแต็ก หลังจากเก็บค่าแล้วตัวชี้จะเลื่อนไปชี้ที่ตำแหน่งถัดไป เช่น ถ้าสั่งให้ตำแหน่ง E0 เก็บค่าใดๆ ไว้ ตัวรีจิสเตอร์ SP ก็จะเก็บค่าไว้ที่ตำแหน่ง E0 จากนั้นก็จะเลื่อนตัวชี้มาชี้อยู่ที่ตำแหน่ง E1

6. รีจิสเตอร์ MAR (Memory Address Register) เป็นรีจิสเตอร์ที่เป็นทำหน้าที่ติดต่อกับหน่วยความจำทั้ง ROM และ RAM โดยจะคอยส่งค่าตำแหน่งของหน่วยความจำไปที่ ROM เมื่อต้องการ Fetch รหัสคำสั่ง หรือส่งค่าตำแหน่งของหน่วยความจำไปที่ RAM เมื่อต้องการอ่านหรือเขียนข้อมูลกับ RAM

7. รีจิสเตอร์ MI (Memory Interface) เป็นรีจิสเตอร์ที่คอยรับส่งค่ากับหน่วยความจำ โดยที่จะรับข้อมูลจาก ROM เมื่อมีการ Fetch รหัสคำสั่งแล้วส่งต่อไปที่รีจิสเตอร์ IR หรือถ้ากรณีที่เป็นคำสั่ง 2 ไบต์ หลังจาก Fetch รหัสคำสั่งแล้ว ก็เป็นโอเปอแรนด์ (Operand) เมื่ออ่านข้อมูลโอเปอแรนด์จาก ROM แล้วก็ส่งต่อไปที่หน่วยคำนวณและตรรกะหรือรีจิสเตอร์ที่ต้องการ และรีจิสเตอร์ MI ยังทำหน้าที่รับส่งข้อมูลกับ RAM เมื่อมีการอ่านหรือเขียนข้อมูลกับ RAM อีกด้วย

8. รีจิสเตอร์ In (Input Register) เป็นรีจิสเตอร์ที่คอยรับค่าจากพอร์ต (Port) แล้วส่งค่าไปยังรีจิสเตอร์ Acc เมื่อมีสัญญาณ Request ส่งมาจากส่วนควบคุม

9. รีจิสเตอร์ Out (Output Register) เป็นรีจิสเตอร์ที่คอยส่งค่าจากรีจิสเตอร์ Acc ไปยังพอร์ตเมื่อมีสัญญาณ Request ส่งไปที่รีจิสเตอร์ Acc จากส่วนควบคุม

โครงสร้างตามรูปที่ 3.2 นั้นยังไม่ได้แสดงส่วนประกอบทั้งหมด ที่ได้แสดงไว้เช่นนั้นก็เพื่อต้องการเปรียบเทียบกับไทแทกเดิม ซึ่งนอกจากส่วนประกอบที่ได้แสดงไปในรูปที่ 3.2 แล้วนั้นส่วนประกอบอื่นๆ ที่เหลือที่ยังไม่ได้แสดงไว้ก็คือวงจร Shifter และ Flag

10. วงจร Shifter เป็นส่วนที่ติดต่อกับรีจิสเตอร์ Acc และ Flag สามารถทำงานตามคำสั่งประเภทเลื่อน (Shift) และหมุน (Rotate) บิต ได้ ในตัววงจร Shifter จำเป็นต้องมีรีจิสเตอร์ชั่วคราวอยู่ภายในเช่นเดียวกับหน่วยคำนวณและตรรกะเพื่อไว้เก็บค่าผลลัพธ์ก่อนที่จะป้อนค่ากลับ (Write-Back) ไปสู่เป้าหมาย

11. Flag ประกอบด้วย CF (Carry Flag) และ ZF (Zero Flag) เท่านั้น มีหน้าที่ในส่วนเส้นทางข้อมูลเพียงติดต่อกับหน่วยคำนวณและตรรกะและวงจร Shifter เมื่อมีการทำงานกับบางคำสั่งที่มีผลต่อ Flag จะส่งผลให้มีการเปลี่ยนค่าภายใน Flag ค่าอินพุตจะเป็น

ค่าที่มาจากหน่วยคำนวณและตรรกะ วงจร Shifter และจากส่วนควบคุม ในขณะที่ค่าเอาท์พุตจะส่งไปยังส่วนควบคุมอย่างเดียวเท่านั้น

เนื่องจากชุดคำสั่งของไทแทกเป็นแบบมีโอเปอแรนด์เดียว ทำให้เวลาทำงานใดๆ ก็ตามจำเป็นต้องเก็บค่าไว้ที่หน่วยความจำ RAM ก่อน แล้วก็นำค่านั้นมากระทำกับค่าที่อยู่ในรีจิสเตอร์ Acc แล้วเก็บผลลัพธ์ไว้ที่รีจิสเตอร์ Acc ตัวอย่างเช่น คำสั่ง ADD (30) เป็นคำสั่งบวกค่าใน Indirect Mode หมายถึงนำค่าที่อยู่ในหน่วยความจำตำแหน่งที่ชี้โดยหน่วยความจำตำแหน่งที่ 30 มาบวกกับค่าที่อยู่ในรีจิสเตอร์ Acc แล้วเก็บผลลัพธ์ไว้ที่รีจิสเตอร์ Acc ถ้าค่าในหน่วยความจำเป็นดังต่อไปนี้

Address	Data
28	17
29	5
30	28
31	96

ตารางที่ 3.1 ตัวอย่างค่าในหน่วยความจำ

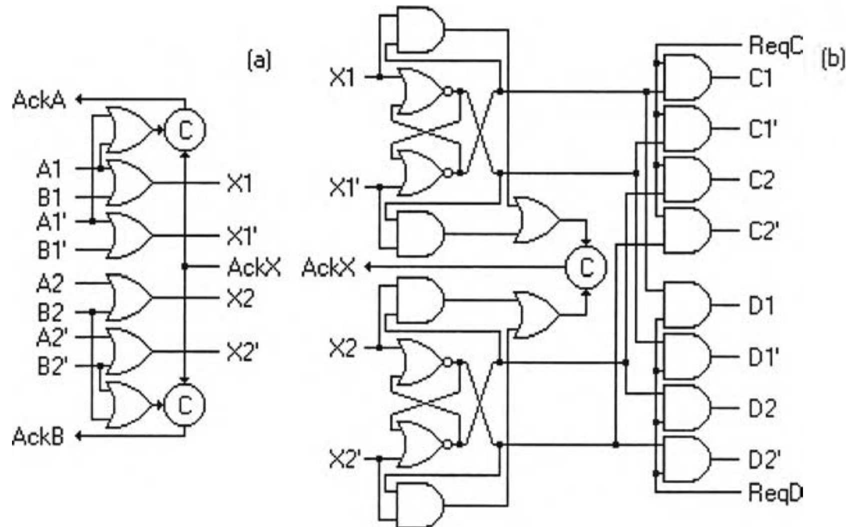
ถ้าค่าที่เก็บอยู่ในรีจิสเตอร์ Acc เท่ากับ 53 ก็จะได้ผลลัพธ์เป็น Acc = 53 + 17 = 70 ซึ่งค่า 70 นั้นก็จะถูกเก็บกลับไปไว้ที่รีจิสเตอร์ Acc

ในกรณีที่จะทำงานคำสั่งลบ จำเป็นต้องคำนึงถึงตัวตั้งและตัวลบด้วย ตัวตั้งจะเป็นค่าที่เก็บอยู่ในรีจิสเตอร์ Acc ส่วนตัวลบจะเป็นค่าที่ Fetch เข้ามาจากหน่วยความจำ ดังนั้นเวลาเก็บค่าก็จะต้องเก็บตัวลบไว้ที่ตำแหน่งใดตำแหน่งหนึ่งในหน่วยความจำ และเก็บตัวตั้งไว้ที่รีจิสเตอร์ Acc

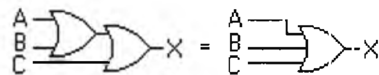
3.3 รีจิสเตอร์

การออกแบบรีจิสเตอร์โดยใช้บัสร่วม รีจิสเตอร์ที่ได้ก็จะมีอินพุต 1 ชุด และเอาท์พุตอีก 1 ชุดเท่านั้น แต่การออกแบบรีจิสเตอร์โดยใช้บัสแบบจุดต่อจุดนั้นจำนวนอินพุตและเอาท์พุตอาจจะมีมากกว่าหนึ่งตัวขึ้นอยู่กับว่าจะให้รีจิสเตอร์เชื่อมต่อกับจุดใดบ้าง ซึ่งวิธีการแบบนี้เรียกว่ารีจิสเตอร์แบบหลายพอร์ต (Multiport Register) ลักษณะดังกล่าวนี้จำเป็นต้องมีการตรวจสอบการมาถึงของ

สัญญาณอินพุต และตรวจสอบการเก็บค่าดังที่ได้แสดงตัวอย่างในรูปที่ 3.3 ถ้าอินพุตมี 3 อินพุตหรือมากกว่าก็ใช้วิธีต่อเกต OR มาต่อกันเป็นหลายๆ ระดับดังรูปที่ 3.4

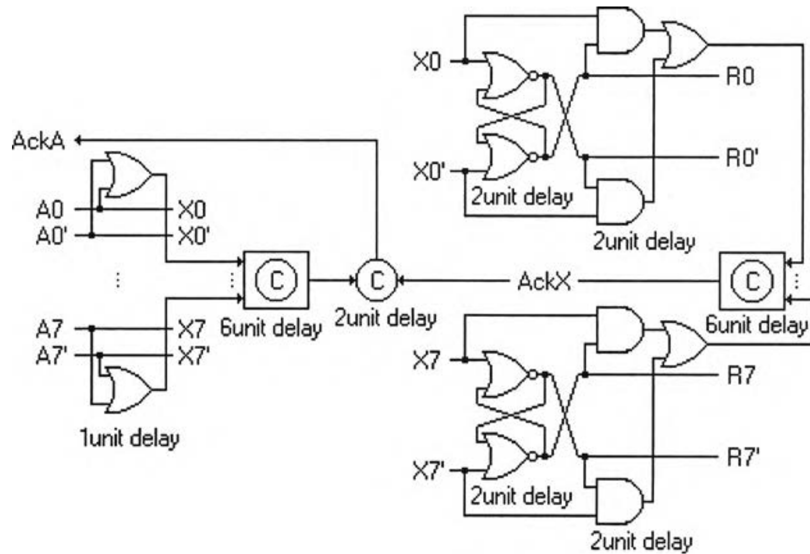


รูปที่ 3.3 แสดงตัวอย่างรีจิสเตอร์ 2 บิต (a) ส่วนอินพุต (b) ภายในรีจิสเตอร์และเอาท์พุต



รูปที่ 3.4 การต่อเกต OR เมื่อมีหลายอินพุต

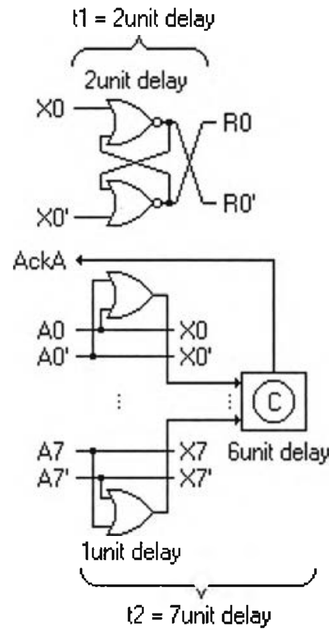
รูปที่ 3.3 (a) เป็นวงจรส่วนอินพุตมีการสร้างส่วนตอบรับตามชุดข้อมูลที่เข้ามา รูปที่ 3.3 (b) เป็นภายในตัวรีจิสเตอร์ มีการสร้างส่วนตอบรับไปส่งกลับไปทางอินพุตหลังจากทำงานเสร็จสิ้นแล้ว ที่เอาท์พุตจะส่งค่าออกไปก็ต่อเมื่อมีสัญญาณ Request เข้ามาเท่านั้น วงจรส่วนตอบรับมีการออกแบบเป็นทั้งโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือนและชนิดปรับมาตราส่วนได้ โดยให้ค่า K เป็น 2 วงจรที่ออกแบบจะมีกลุ่มของอุปกรณ์ชนิดซีที่ต่อกันเป็นระดับชั้นเช่นเดียวกันกับการต่อเกต OR ดังรูปที่ 3.4 โดยที่อุปกรณ์ชนิดซีแต่ละตัวประมาณความหน่วงเป็น 2 หน่วย และเกตอื่นๆ ทั่วไปเป็น 1 หน่วย วงจรส่วนตอบรับของรีจิสเตอร์ที่เป็นโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือนจะมีความหน่วงทั้งสิ้นตั้งแต่สัญญาณอินพุตเข้ามาจนให้สัญญาณตอบรับได้เป็น 12 หน่วย ตามรูปที่ 3.5



รูปที่ 3.5 วงจรส่วนตอบรับของรีจิสเตอร์ที่ออกแบบโดยโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือน

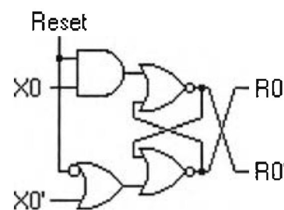
ส่วนวงจรส่วนตอบรับของรีจิสเตอร์ที่เป็นโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้มีการออกแบบที่แตกต่างออกไปดังรูปที่ 3.6 ซึ่งถ้าเปรียบเทียบกับรูปที่ 3.5 แล้วจะพบว่าเป็นเพียงการตัดเอาส่วนการตรวจสอบการเก็บค่ารีจิสเตอร์ออกไป แล้วใช้แค่ส่วนการตรวจสอบการมาถึงของอินพุตมาตรวจสอบเท่านั้น ซึ่งถ้าเทียบตามทฤษฎีแล้ว t1 มีค่าความหน่วง 2 หน่วย ส่วน t2 มีค่าความหน่วง 7 หน่วย กำหนดค่า K เป็น 2 จะเห็นว่า t2 มีความหน่วงมากกว่า t1 มากกว่า 2 เท่า ซึ่งสามารถรับประกันได้ว่าวงจรจะให้ค่า R ก่อนที่จะให้สัญญาณตอบรับแน่นอน เมื่อเทียบกันแล้วเห็นได้ชัดเจนว่าวงจรส่วนตอบรับที่ออกแบบด้วยโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้มีสมรรถนะสูงกว่าชนิดเสมือนอยู่พอสมควร เนื่องจากไม่ได้ตรวจสอบแบบระมัดระวังเกินความจำเป็นแต่ตรวจสอบเท่าที่จำเป็นเท่านั้น สมรรถนะที่ได้จึงดีกว่า

หอสมุดกลาง สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย



รูปที่ 3.6 วงจรส่วนตอบรับของรีจิสเตอร์ที่ออกแบบโดยโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้

รีจิสเตอร์แต่ละตัวจำเป็นต้องมีการรีเซ็ต (Reset) ค่าเมื่อเริ่มทำงาน เพราะเมื่อเริ่มต้นทำงานรีจิสเตอร์แต่ละตัวยังไม่มีค่าใดเก็บไว้ทำให้อาจทำงานผิดพลาดได้ ดังนั้นจึงควรให้ค่าใดๆ กับรีจิสเตอร์เก็บไว้ก่อนจะเริ่มทำงานใดต่อไป ในที่นี้ได้รีเซ็ตให้เป็นค่า (0,1) ซึ่งสามารถทำได้โดยการเพิ่มเกต AND NOT และ OR เข้าไปดังรูปที่ 3.7 เมื่อกดปุ่มรีเซ็ตค่า Reset จะเป็น 0 รีจิสเตอร์ก็จะเก็บค่า (0,1) ไว้จากนั้นเมื่อปล่อยจากปุ่มรีเซ็ตค่า Reset ก็จะเป็น 1 รีจิสเตอร์ก็จะสามารถทำงานได้ตามปกติ



รูปที่ 3.7 การรีเซ็ตค่าให้กับรีจิสเตอร์

3.4 หน่วยคำนวณและตรรกะ

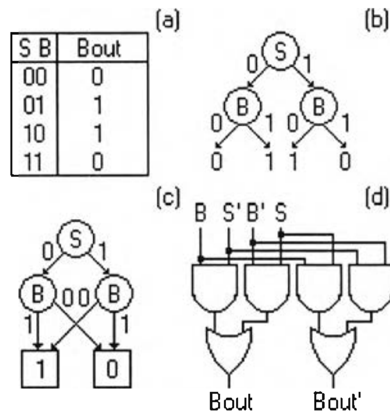
การออกแบบหน่วยคำนวณและตรรกะถือเป็นส่วนสำคัญส่วนหนึ่งในส่วนเส้นทางข้อมูล เพราะหน่วยคำนวณและตรรกะเป็นตัวที่ทำหน้าที่สำคัญมากมาย การออกแบบเริ่มจากการดูว่าฟังก์ชัน (Function) การทำงานมีอะไรบ้าง แล้วเขียนเป็นตารางค่าความจริง จากนั้นจึงเขียน BDD

จากตารางค่าความจริง แล้วก็ลดรูปที่ได้ลงเหลือเป็น ROBDD สุดท้ายก็ออกแบบวงจรตามแบบ ROBDD ที่ได้ ซึ่งจะนำเสนอการออกแบบในแต่ละส่วนต่อไป

วงจรในหน่วยคำนวณและตรรกะจะประกอบไปด้วยวงจรที่ทำหน้าที่ AND, OR, XOR, ADD, และ SUB โดยที่อินพุตจะเป็นข้อมูลขนาด 8 บิต 2 ค่า ในที่นี้จะแสดงด้วย A และ B ซึ่งมาจากรีจิสเตอร์ต่างๆ นั่นเอง โดยให้ B แทนอินพุตที่มาจากรีจิสเตอร์ MI ส่วน A แทนรีจิสเตอร์ตัวอื่นๆ นอกจากนี้ยังมีอินพุตอีกชุดหนึ่งที่ส่งมาจากหน่วยควบคุมที่ทำหน้าที่เป็นส่วนหนึ่งของการทำงานด้วย นั่นก็คือ (Cin,Cin’) แสดงค่า Carry In ที่เข้ามาจากส่วนควบคุม ค่า Cin จะใช้เพื่อทำ 2’s Complement ในวงจรลบ ดังนั้นค่าอินพุตจึงต้องมี (S,S’) ที่จะแสดงคำสั่งลบด้วย ค่านี้จะส่งมาจากส่วนควบคุมเช่นกัน โดยที่ถ้าค่า S ถูกเซ็ตค่าเป็น 1 แล้ว Cin ก็จะถูกเซ็ตด้วย และข้อมูล B ก็จะเปลี่ยนเป็น B’ และข้อมูล B’ จะเปลี่ยนเป็น B เพื่อเป็นการทำ 2’s Complement ให้วงจรบวกทำหน้าที่ทั้งบวกและลบข้อมูล นอกจากนี้เมื่อทำงานเสร็จสิ้นจนให้ค่าผลลัพธ์แล้ว ก็ยังมีรีจิสเตอร์ชั่วคราวอยู่ภายในอีกหนึ่งตัวเพื่อคอยเก็บค่าไว้รอการป้อนค่ากลับสู่เป้าหมายต่อไป และมี Latch ไว้คอยเก็บผลลัพธ์ของ CF ที่เป็นผลมาจากการทำงานในแต่ละครั้งด้วย โดยที่จะเก็บค่าไว้ก็ต่อเมื่อเป็นการทำงานที่มีผลต่อ Flag เท่านั้น ซึ่งก็จะรอการป้อนค่ากลับไปสู่ตัว Flag เช่นกัน ทั้งนี้สายสัญญาณที่เป็น Request เพื่อการป้อนค่ากลับที่ส่งไปยังรีจิสเตอร์ภายในหน่วยคำนวณและตรรกะกับ Latch ที่เก็บค่า Flag นั้นจะเป็นคนละเส้นกัน เพื่อประโยชน์ในการควบคุม เพราะการทำงานในบางคำสั่งมีผลเพียงแค่เปลี่ยนค่าของ Flag เท่านั้น แต่จะไม่มีการส่งผลลัพธ์ไปที่ใดทั้งสิ้น และการทำงานบางอย่างก็จะให้เพียงผลลัพธ์ที่จะไม่ส่งผลกระทบต่อ Flag เลย

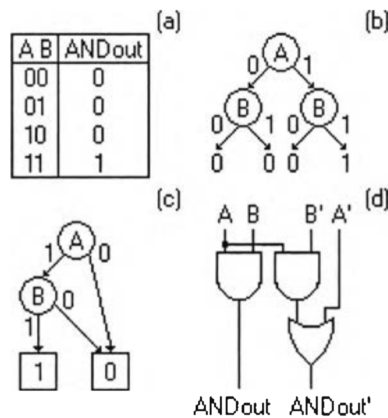
เนื่องจากไมโครโพรเซสเซอร์ตัวนี้ออกแบบอยู่บนพื้นฐานของไทแทกจึงไม่มีการใช้บัสร่วมแต่ใช้บัสแบบจุดต่อจุด ดังนั้นอินพุตและเอาท์พุตของหน่วยคำนวณและตรรกะจึงจำเป็นต้องออกแบบเป็นหลายพอร์ตเช่นเดียวกับรีจิสเตอร์

การออกแบบหน่วยคำนวณและตรรกะจะเริ่มจากวงจรสลับค่า B กับ B’ ก่อน เพราะการทำงานคำสั่งลบจำเป็นต้องใช้การสลับค่าของ B ในที่นี้จะให้ (Bout,Bout’) แทนค่า (B’,B) เมื่อมีสัญญาณ (S,S’) จากส่วนควบคุมมาเป็น (1,0) เป็นการแสดงถึงการให้ทำงานคำสั่งลบ วงจรนี้ก็จะสลับค่า B กับ B’ การออกแบบเป็นดังรูปที่ 3.8

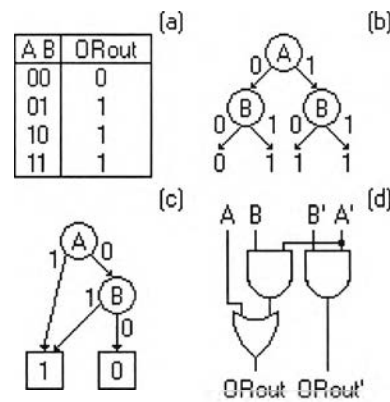


รูปที่ 3.8 (a) ตารางค่าความจริง (b) BDD (c) ROBDD (d) วงจรสลับค่า B เมื่อเป็นคำสั่งลบ

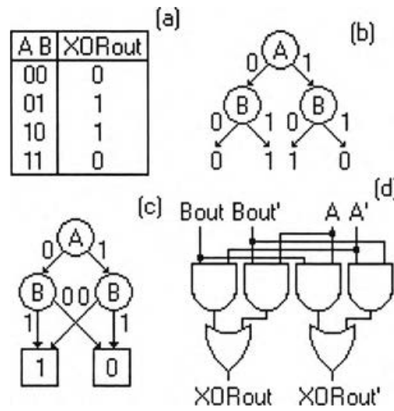
สังเกตดูจากตารางค่าความจริงจะพบว่านี่คือวงจร XOR นั่นเอง ดังนั้นการออกแบบลักษณะนี้นำไปใช้กับวงจร XOR ได้เลย ในส่วนต่อไปเป็นการออกแบบวงจร AND OR XOR และบวกลบ ซึ่งเป็นดังรูปที่ 3.9 3.10 3.11 และ 3.12 ตามลำดับ



รูปที่ 3.9 (a) ตารางค่าความจริง (b) BDD (c) ROBDD (d) วงจร AND

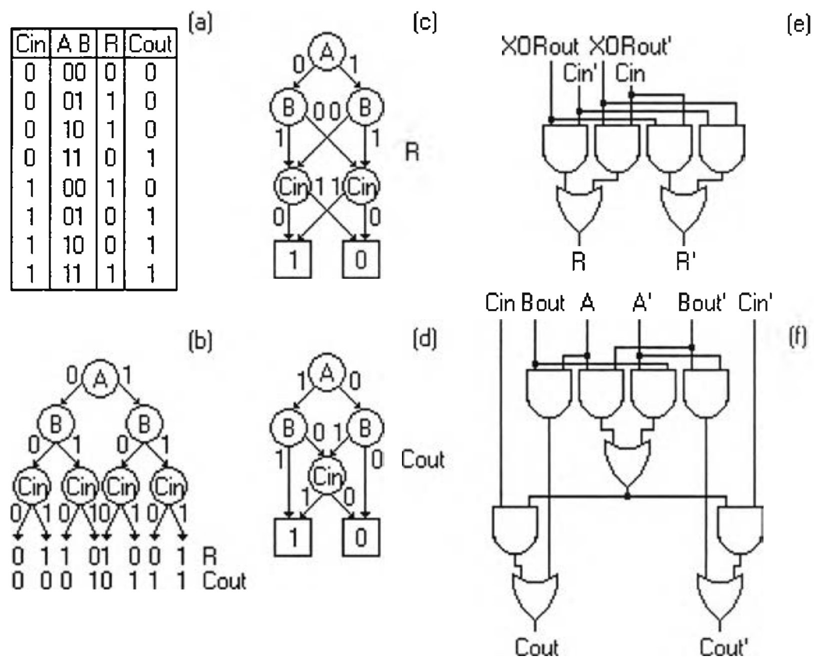


รูปที่ 3.10 (a) ตารางค่าความจริง (b) BDD (c) ROBDD (d) วงจร OR



A B	XORout
00	0
01	1
10	1
11	0

รูปที่ 3.11 (a) ตารางค่าความจริง (b) BDD (c) ROBDD (d) วงจร XOR ที่ใช้อินพุตเป็น (Bout,Bout')



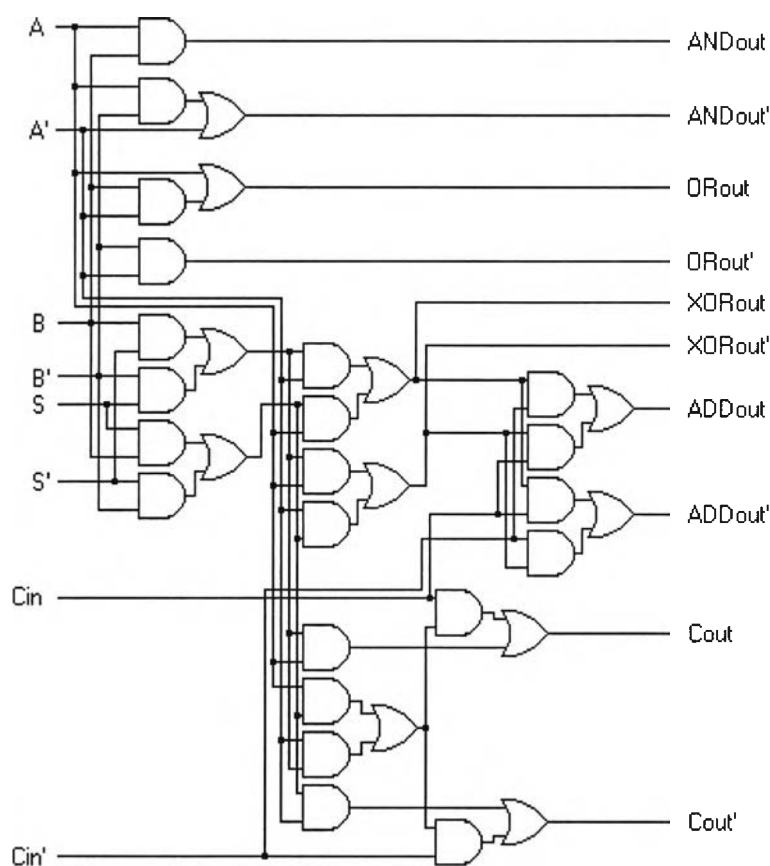
Cin	A B	R	Cout
0	00	0	0
0	01	1	0
0	10	1	0
0	11	0	1
1	00	1	0
1	01	0	1
1	10	0	1
1	11	1	1

รูปที่ 3.12 (a) ตารางค่าความจริง (b) BDD แสดง R และ Cout (c) ROBDD ของค่า R (d) ROBDD ของค่า Cout (e) วงจรบวกลบที่ให้ค่า R (f) วงจรบวกลบที่ให้ ค่า Cout

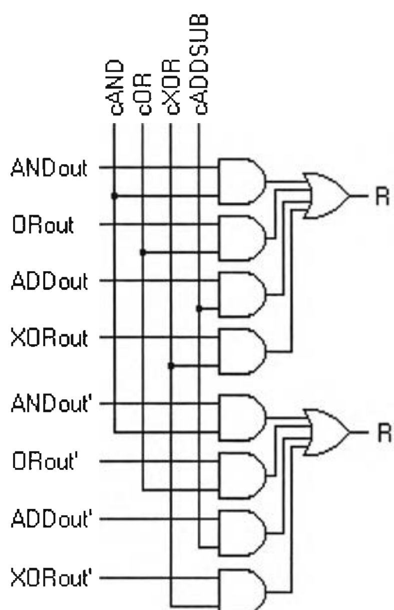
ในการออกแบบวงจร XOR จำเป็นต้องใช้อินพุตเป็น (Bout,Bout') แทนค่า (B,B') โดยมีค่า XORout และ XORout' เป็นเอาท์พุต เป็นเพราะว่าวงจร XOR นั้นจะมีการใช้งานร่วมกับวงจร ADD/SUB ด้วย การเปลี่ยนแปลงค่า B มีผลต่อค่าเอาท์พุต จึงทำให้ต้องใช้ค่า (Bout,Bout')

วงจรบวกลบซึ่งมีค่า (R,R') และ (Cout,Cout') เป็นเอาท์พุตนั้น สามารถนำสายสัญญาณมาจากผลลัพธ์ของวงจร XOR มาใช้ได้เลย จะไม่มีผลกระทบใดๆ เพราะถ้าสังเกตจากรูปที่ 3.11 (c) และรูปที่ 3.12 (c) แล้วจะพบว่า ROBDD ของรูปที่ 3.12 (c) นั้นเป็นเพียงการเพิ่มเติมขึ้นมาจากรูปที่ 3.11 (c) นั่นเอง

เมื่อออกแบบทุกส่วนแล้วก็นำมาประกอบเข้าด้วยกันเป็นหน่วยคำนวณและตรรกะทั้งตัวดังรูปที่ 3.13 และที่เอาท์พุตก็ต้องมีตัวเลือกผลลัพธ์ดังรูปที่ 3.14 เพื่อให้ได้ผลลัพธ์ที่ต้องการออกมา ซึ่งการเลือกผลลัพธ์จะถูกควบคุมโดยสายสัญญาณ cAND cOR cXOR และ cADDSUB ที่เข้ามาจากส่วนควบคุม ตัวเลือกผลลัพธ์นั้นจำเป็นต้องมีเฉพาะสำหรับค่า R เท่านั้น ค่า Cout ไม่จำเป็นต้องมีตัวเลือกผลลัพธ์ สามารถส่งผลลัพธ์ออกไปได้เลย และใช้สัญญาณ cADDSUB เป็นตัวกำหนดว่าจะให้เก็บค่า CF ไว้หรือไม่ ถ้าสัญญาณ cADDSUB เป็น 1 แสดงว่าเป็น คำสั่งบวกหรือลบ ก็ให้ค่าเก็บไว้แต่ว่าจะส่งออกไปเก็บยัง Flag จริงๆ หรือไม่นั้น ส่วนควบคุมจะเป็นตัวตัดสินใจ เพราะการบวกลบบางอย่างไม่มีผลต่อ Flag เช่น การเพิ่มค่าให้รีจิสเตอร์ PC เป็นต้น



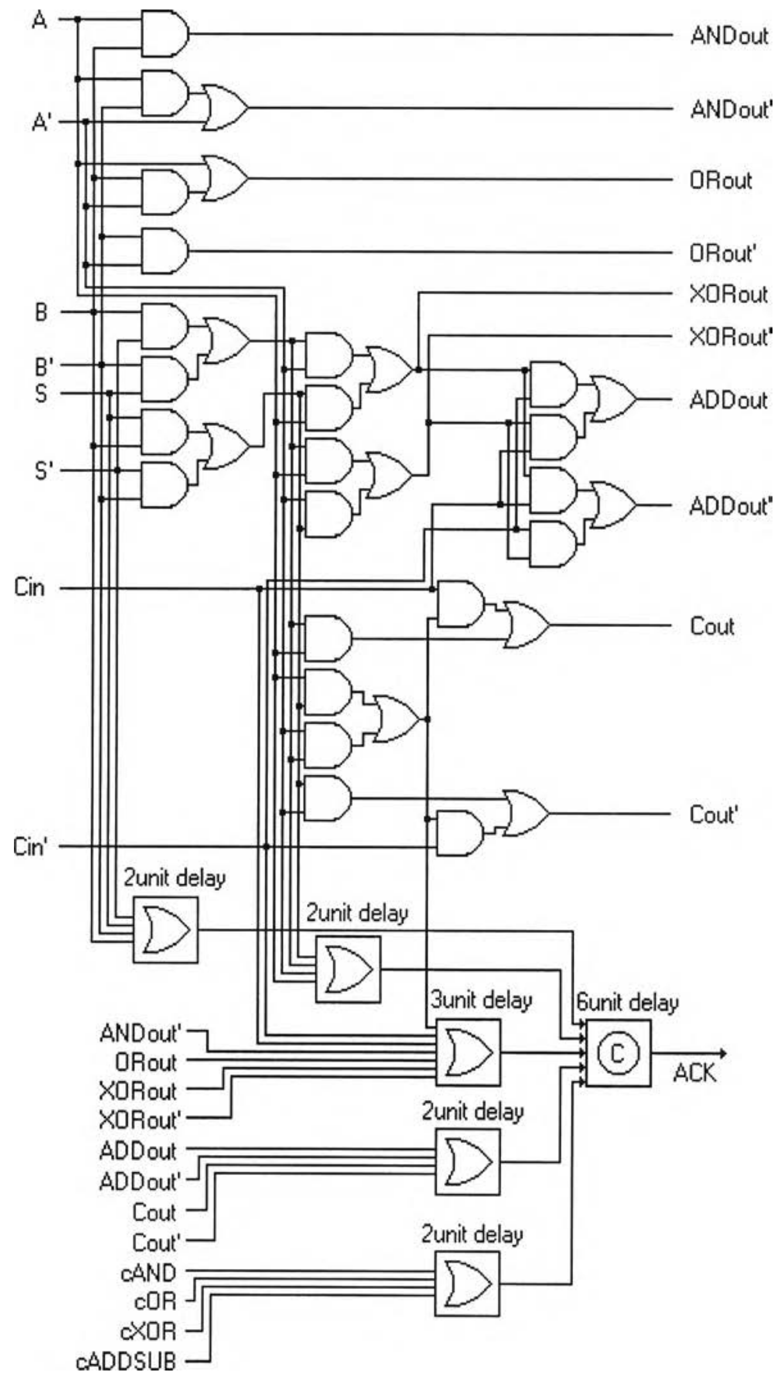
รูปที่ 3.13 หน่วยคำนวณและตรรกะ



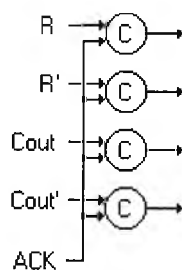
รูปที่ 3.14 ตัวเลือกผลลัพธ์ของหน่วยคำนวณและตรรกะ

หลังจากที่ออกแบบวงจรรางคู่ของหน่วยคำนวณและตรรกะเสร็จสิ้นแล้วก็สร้างวงจรตอบรับ ซึ่งวงจรตอบรับตามโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือนได้แสดงไว้ดังรูปที่ 3.15 เพราะเนื่องจากว่าวงจรรางคู่ได้ออกแบบโดยใช้ ROBDD ทำให้การสร้างวงจรตอบรับสามารถทำได้ด้วยวิธีการตรวจสอบการมาถึงของสัญญาณในแต่ละระดับ วงจรที่ออกแบบด้วย ROBDD นั้นในขั้นทำงานจะมีเพียงเส้นทางเดียวเท่านั้นที่มีค่าสัญญาณ 1 ดังนั้นเพียงแค่ตรวจสอบการไหลของสัญญาณในแต่ละระดับก็จะสามารถรับประกันความถูกต้องของการทำงานได้ จากนั้นก็นำสัญญาณ ACK ที่ได้จากวงจรตอบรับไปเข้าอุปกรณ์ชนิดซีร่วมกับผลลัพธ์ที่ได้เพื่อออกเป็นเอาท์พุตที่แท้จริงดังรูปที่ 3.16

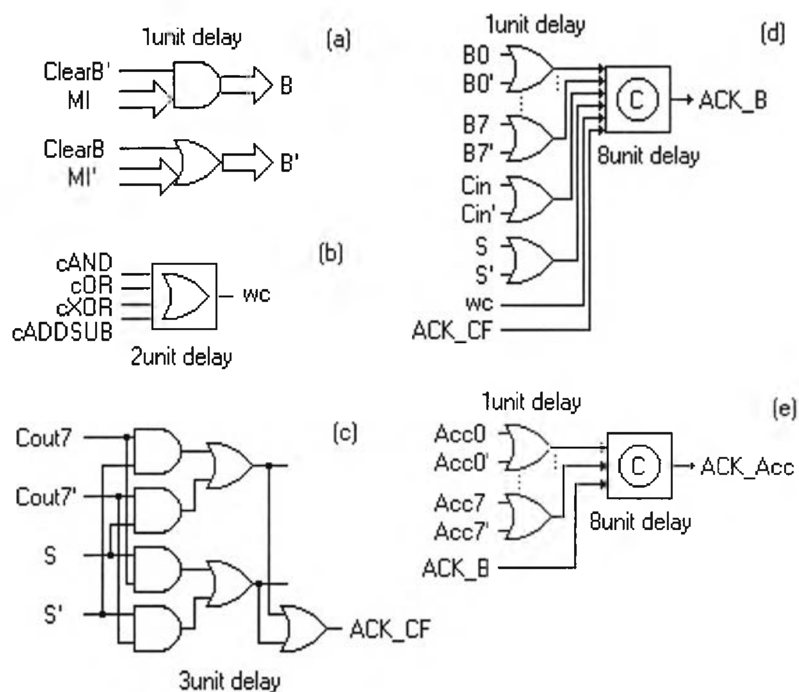
สังเกตว่าวงจรตอบรับตามโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือนดังรูปที่ 3.15 นั้นเป็นวงจรตอบรับสำหรับ 1 บิตเท่านั้น จะต้องสร้างวงจรแบบนั้นให้กับทุกๆ บิต ทำให้เป็นการสิ้นเปลืองมาก ดังนั้นถ้าใช้วิธีการออกแบบวงจรตอบรับด้วยโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ วงจรตอบรับก็จะเปลี่ยนไปไม่ต้องออกแบบวงจรเช่นนี้กับทุกๆ บิต แต่ใช้หลักการตามทฤษฎีซึ่งก็คือตรวจสอบการมาถึงของสัญญาณแล้วดูความหน่วงของการเปลี่ยนระดับสัญญาณให้มีค่าอย่างน้อยเป็น K เท่าของการเปลี่ยนระดับสัญญาณของวงจรรางคู่ ซึ่งวิธีนี้ใช้เกตน้อยกว่ามากเพราะเป็นการตรวจสอบทีเดียวทุกบิตเลย ไม่จำเป็นต้องมีวงจรในลักษณะเดียวกันซ้ำๆ ในแต่ละบิตสามารถตัดการตรวจสอบสัญญาณภายในแต่ละบิตออกไปได้เลย แล้วตรวจสอบทีเดียวทุกบิตแทน วงจรตอบรับที่ออกแบบด้วยโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้เป็นดังรูปที่ 3.17



รูปที่ 3.15 วงจรตอบรับของหน่วยคำนวณและตรรกะที่ออกแบบด้วยโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือน



รูปที่ 3.16 เอาท์พุตที่สมบูรณ์ของโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือน



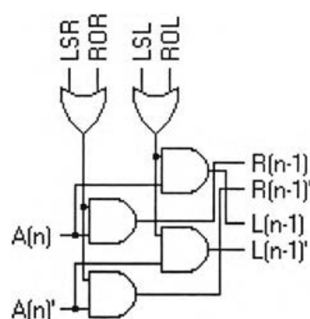
รูปที่ 3.17 วงจรตอบรับของหน่วยคำนวณและตรรกะที่ออกแบบด้วยโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้

วงจรตอบรับของโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้ออกแบบโดยกำหนดค่า K เป็น 2 จากรูปที่ 3.12 ถ้าประมาณความหน่วงของเกตเป็น 1 หน่วย และความหน่วงของอุปกรณ์ชนิดซีเป็น 2 หน่วย จะได้ว่าหน่วยคำนวณและตรรกะมีความหน่วงเป็น 6 หน่วย ดังนั้นตามทฤษฎีของโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้แล้ว วงจรส่วนตอบรับจะต้องมีความหน่วงไม่น้อยกว่า 12 หน่วยเพื่อรับประกันความถูกต้องโดยเผื่อความผิดพลาดไว้ K เท่า ในวงจรตามรูปที่ 3.17 วงจรตอบรับมีความหน่วงนับได้ 19 หน่วย ถือว่ารับประกันความถูกต้องได้

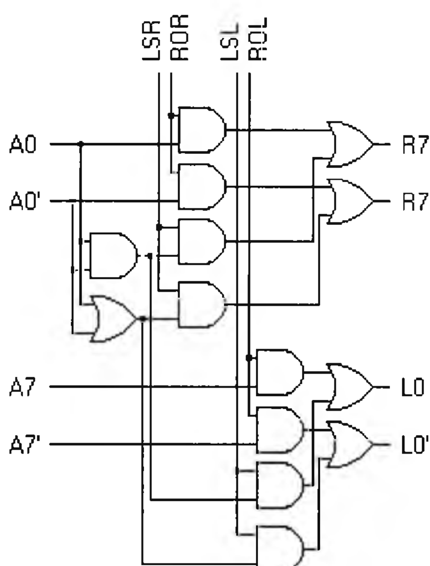
3.5 วงจร Shifter

ส่วนที่มีความสำคัญมากอีกส่วนหนึ่งนอกเหนือไปจากหน่วยคำนวณและตรรกะและรีจิสเตอร์ต่างๆ นั่นก็คือ วงจร Shifter เพราะอุปกรณ์ตัวนี้ใช้ทำหน้าที่เลื่อน (Shift) และหมุน (Rotate) บิต ตัววงจร Shifter นอกจากจะมีวงจรที่ทำงานได้แล้วยังจำเป็นต้องมีรีจิสเตอร์ชั่วคราวอยู่ภายในอีกตัวหนึ่งด้วยเพื่อทำหน้าที่เก็บค่าไว้รอการป้อนค่ากลับสู่เป้าหมายต่อไปหลังจากทำงานเสร็จ เมื่อวงจร Shifter ทำงานเสร็จสิ้นจะส่งผลให้ Flag เปลี่ยนแปลงด้วย โดย CF จะมีค่าเท่ากับค่าของบิตต่ำสุด (Least Significant Bit) ของผลลัพธ์

เนื่องจากวงจร Shifter ตัวนี้จะทำงานเพียงครั้งละบิตเดียวเท่านั้น คือเลื่อนบิตหรือหมุนบิตไปเพียงบิตเดียว การออกแบบวงจร Shifter จึงลดความยุ่งยากลงไปได้พอสมควร เริ่มจากสร้างตัวเลือกดังรูปที่ 3.18 เพื่อเลือกว่าจะเลื่อนไปยังบิตที่สูงกว่าหรือต่ำกว่าเดิม ถ้ากำหนดให้ทั้ง 8 บิตเป็นบิตที่ 0 ถึงบิตที่ 7 แล้ว ตัวเลือกนี้จะใช้งานได้ตั้งแต่อินพุตบิตที่ 1 ไปจนถึงบิตที่ 6 เพราะว่าคำสั่งเลื่อนหรือหมุนบิตนั้นจะมีผลให้บิตที่ 1 ถึงบิตที่ 6 ทำงานเหมือนกัน และเนื่องจากว่าสัญญาณที่จะมาจากส่วนควบคุมนั้นจะมีเพียงตัวเดียวจาก 4 ตัวเท่านั้น คือ 1 ตัวจาก LSR, LSL, ROR, หรือ ROL ดังนั้นการใช้เพียงเกต OR เพื่อควบคุมการเลื่อนบิตก็เป็นการเพียงพอแล้ว ส่วนบิตที่ 0 และบิตที่ 7 ซึ่งเป็นบิตต่ำสุดและบิตสูงสุดนั้นจะต้องออกแบบเฉพาะสำหรับแต่ละบิตดังรูปที่ 3.19



รูปที่ 3.18 ตัวเลือกเพื่อการเลื่อนบิตไปสูงกว่าหรือต่ำกว่าเดิม



รูปที่ 3.19 การออกแบบในส่วนบิตสูงสุดและบิตต่ำสุด

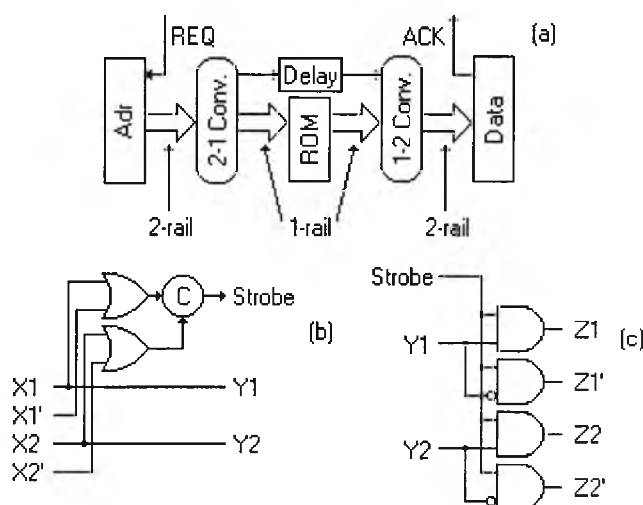
เมื่อนำส่วนต่างๆ มาประกอบกันก็จะได้วงจร Shifter ที่ทำงานได้อย่างถูกต้อง ผลลัพธ์ก็จะถูกเก็บไว้ในรีจิสเตอร์ชั่วคราวที่อยู่ภายในเพื่อรอการป้อนค่ากลับต่อไป และค่าของบิตต่ำสุดก็จะส่งไปที่ Flag เพื่อเซ็ตค่า CF ไม่มีความจำเป็นต้องใช้แลตช์เพื่อเก็บค่า Flag ไว้รอการป้อนค่ากลับเพราะ

ว่าค่านี้สามารถดึงเอามาจากบิตต่ำสุดได้เลย มีเพียงรีจิสเตอร์ชั่วคราวเพียงตัวเดียวก็พอแล้ว ส่วนวงจรตอบรับนั้นถ้าเป็นโมเดลที่ไม่ไวต่อความหน่วงชนิดเสมือนก็ใช้การตรวจสอบการมาถึงของสัญญาณร่วมกับการตรวจสอบการเก็บค่าในรีจิสเตอร์ชั่วคราวก็จะเพียงพอในการรับประกันความถูกต้องได้ ส่วนโมเดลที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้นั้นก็ตัดส่วนการตรวจสอบในรีจิสเตอร์ชั่วคราวออกไปเท่านั้น

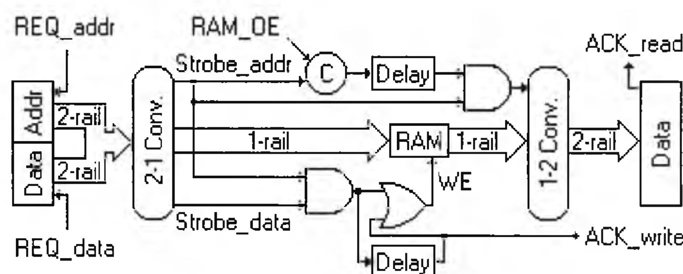
3.6 หน่วยความจำ (Memory Unit)

หน่วยความจำทั้ง ROM และ RAM ที่จะนำมาใช้งานในวงจรอสมวารนั้นเป็นหน่วยความจำแบบสมวารทั่วๆ ไป แต่มีการเพิ่มเติมโครงสร้างบางส่วนเข้าไปแล้วทำให้กลายเป็นหน่วยความจำแบบอสมวารได้

การออกแบบ ROM ทำได้ดังรูปที่ 3.20 เนื่องจาก ROM อ่านได้อย่างเดียว การออกแบบจึงไม่ซับซ้อนมาก ส่วน RAM มีความซับซ้อนมากขึ้นเนื่องจากทำได้ทั้งอ่านและเขียน สามารถออกแบบได้ดังรูปที่ 3.21



รูปที่ 3.20 (a) หน่วยความจำ ROM แบบอสมวาร (b) วงจร 2-1 Converter (c) วงจร 1-2 Converter



รูปที่ 3.21 หน่วยความจำ RAM แบบอสมวาร

ขั้นตอนการทำงานของหน่วยความจำ ROM เริ่มจากเมื่อรีจิสเตอร์ที่เก็บตำแหน่ง ซึ่งในที่นี้หมายถึงรีจิสเตอร์ MAR ที่มีบัสติดต่อกับทั้ง ROM และ RAM ส่งค่าตำแหน่งไปที่ ROM 2-1 Converter ซึ่งทำหน้าที่แปลงสัญญาณรางคู่ให้เป็นสัญญาณรางเดี่ยว (1-rail) ปกติแบบที่ใช้ในวงจรสมวารทั่วไป ค่าตำแหน่งนั้นก็จะส่งไปที่ตัว ROM แล้ว ROM ก็จะส่งค่าผลลัพธ์ออกมาผ่านตัว 1-2 Converter ที่ทำหน้าที่แปลงสัญญาณรางเดี่ยวให้เป็นสัญญาณรางคู่อีกทีหนึ่ง โดยก่อนที่จะให้ค่าผลลัพธ์ออกมานั้นจะมีส่วนที่คอยหน่วงสัญญาณไว้ไม่ให้ผลลัพธ์ออกมาทันที เพื่อที่ว่าจะให้แน่ใจว่าค่าตำแหน่งที่ส่งไปถึง ROM นั้นไปถึงแล้วจริงๆ เพราะ ROM นั้นจะมีค่าความหน่วงในการเข้าถึง (Access Time) ตัว ROM อยู่ด้วย การจะทำงานกับตัว ROM จะต้องรออย่างน้อยเท่ากับค่าความหน่วงในการเข้าถึงนี้จึงจะทำงานได้อย่างถูกต้องและมีเสถียรภาพ และเมื่อข้อมูลผลลัพธ์ที่ต้องการออกมาแล้วก็จะเก็บไว้ที่รีจิสเตอร์ปลายทาง ในที่นี้ก็คือรีจิสเตอร์ MI แล้วรีจิสเตอร์นี้ก็จะส่งสัญญาณตอบรับกลับไปที่ส่วนควบคุม บอกว่าได้รับข้อมูลมาแล้ว ส่วนอุปกรณ์ที่ทำหน้าที่เป็นตัวหน่วงสัญญาณในที่นี้ใช้ D Flipflop 2 ตัวต่อกัน เพื่อให้หน่วงไป 2 สัญญาณนาฬิกา

ส่วนในขั้นตอนการทำงานของหน่วยความจำ RAM นั้นแตกต่างออกไป เพราะจะต้องเริ่มจากการที่รีจิสเตอร์ MAR ส่งค่าตำแหน่งออกมา พร้อมกับรีจิสเตอร์ MI ก็ส่งค่าข้อมูลที่จะเขียนออกมาด้วย ซึ่งทั้ง 2 ค่านี้ก็จะส่งไปที่ RAM โดยที่จะผ่านตัว 2-1 Converter เพื่อแปลงสัญญาณรางคู่ให้เป็นสัญญาณรางเดี่ยวก่อน โดยที่ทั้ง 2 ค่าคือค่าตำแหน่งและข้อมูลที่ส่งมานั้นอาจมาถึงไม่พร้อมกันก็ได้ จึงต้องมีการตรวจสอบเพิ่มเติม คราวนี้สังเกตว่าต้องมีสัญญาณ RAM_OE (RAM Output Enable) คอยกำกับอยู่ด้วยเพื่อความถูกต้องในการทำงาน โดยสัญญาณดังกล่าวจะมาเมื่อเป็นการอ่านข้อมูลจาก RAM เท่านั้น และเนื่องจาก RAM ก็มีค่าความหน่วงในการเข้าถึงเช่นกัน ดังนั้นจึงต้องมีส่วนที่คอยหน่วงสัญญาณด้วย ซึ่งก็ใช้ D Flipflop 2 ตัวเช่นเดียวกับ ROM