บทที่ 5

การออกแบบไมโครโพรเซสเซอร์แบบอสมวาร

ในบทนี้กล่าวถึงการออกแบบไมโครโพรเซสเซอร์แบบอสมวารเข้ารหัสหนึ่งในสี่ ซึ่งเป็นองค์ประกอบที่ได้ครอบครองการใช้งานบัสระบบ โดยใช้งานบัสระบบเพื่อตอบสนองงานตามคำสั่งที่ต้องการติดต่อรับส่งข้อมูลกับหน่วยความจำ รายละเอียดของการออกแบบไมโครโพรเซสเซอร์มีดังต่อไปนี้

5.1 คุณสมบัติของไมโครโพรเซสเซอร์

ไมโครโพรเซสเซอร์แบบอสมวารที่ออกแบบ มีคุณสมบัติโดยสรุปได้ดังนี้

- มีขนาด 8 บิต หรือ 16'1of4 บิต เช่นเดียวกับไมโครโพรเซสเซอร์เข้ารหัสรางคู่ [4,17]
- เข้ารหัสหนึ่งในสี่กับสัญญาณอาณัติแบบ 4 ขั้น แทนการเข้ารหัสรางคู่กับสัญญาณอาณัติแบบ 4 ขั้นของไมโครโพรเซสเซอร์เวอร์ชันเดิม [4,17] เพื่อลดการเปลี่ยนสถานะสัญญาณของไมโครโพรเซสเซอร์
- รองรับการบริการอินเตอร์รัพท์ และรองรับการทำงานร่วมกับบัสระบบและดีเอ็มเอ เช่นเดียวกับไมโครโพรเซสเซอร์เวอร์ชันเดิม [4] เพื่อให้การทำงานร่วมกับอุปกรณ์ต่อพ่วงมีความรวดเร็วยิ่งขึ้น
- ติดต่อกับอุปกรณ์อินพุท/เอาท์พุทแบบ I/O-mapped I/O หรือ Peripheral-mapped แทนการติดต่อกับอุปกรณ์อินพุท/เอาท์พุทแบบ Memory-mapped I/O ในไมโครโพรเซสเซอร์เวอร์ชันเดิม [4] เพื่อให้สามารถใช้หน่วยความจำได้ทุกตำแหน่งเลขที่อยู่
- มีจำนวนคำสั่ง 27 คำสั่ง ซึ่งเพิ่มจากไมโครโพรเซสเซอร์เวอร์ชันเดิม [17] ที่มีจำนวนคำสั่งเพียง 16 คำสั่ง

5.2 ชุดคำสั่งและรหัสดำเนินการ

ไมโครโปรเซสเซอร์ที่ออกแบบ ประกอบด้วยชุดคำสั่ง (Instruction Set) เดิมจากไมโครโพรเซสเซอร์ 8 บิตเข้ารหัสรางคู่ [17] และชุดคำสั่งซึ่งออกแบบเพิ่มเติมสำหรับเรียกใช้งานดีเอ็มเอ และการบริการอินเตอร์รัพท์ ชุดคำสั่งและรหัสดำเนินการทั้งหมดแสดงในภาคผนวก ก. โดยชุดคำสั่งทั้งหมดนั้นแบ่งออกเป็น 3 กลุ่ม มีจำนวนคำสั่งรวม 27 คำสั่ง รายละเอียดมีดังต่อไปนี้

1. กลุ่มคำสั่งเคลื่อนย้ายข้อมูล (Data Transfer Operation) ใช้สำหรับงานเคลื่อนย้ายข้อมูลระหว่างไมโครโพรเซสเซอร์ หน่วยความจำ และอุปกรณ์ต่อพ่วง ประกอบด้วยคำสั่งโหลด (Load: LD) คำสั่งสโตร์ (Store: ST) คำสั่งอิน(IN) และคำสั่งเอาท์ (OUT)

2. กลุ่มคำสั่งควบคุมการทำงาน (Program and Machine Control Operation) ใช้สำหรับควบคุมการทำงานตามเงื่อนไขที่กำหนดไว้ ประกอบด้วยคำสั่งกระโดดแบบไร้เงื่อนไข (Jump: JMP) คำสั่งกระโดดเมื่อค่าในรีจิสเตอร์ตัวสะสม (Accumulator Register: ACC, A) เท่ากับศูนย์ (Jump Zero: JZ) คำสั่งกระโดดเมื่อค่าในรีจิสเตอร์ตัวสะสมไม่เท่ากับศูนย์ (Jump not Zero: JNZ) คำสั่งกระโดดเมื่อค่าในรีจิสเตอร์ตัวสะสมมีตัวทด (Jump Carry: JC) คำสั่งกระโดดเมื่อค่าในรีจิสเตอร์ตัวสะสมไม่มีตัวทด (Jump not Carry: JNC) คำสั่งเรียกใช้งานโปรแกรมย่อย (CALL) คำสั่งกลับไปใช้งานโปรแกรมหลัก (Return: RET) และคำสั่งกลับจากงานบริการอินเตอร์รัพท์ (Return from Interrupt: RETI)

3. กลุ่มคำสั่งดำเนินการทางคณิตศาสตร์และตรรกะ (Arithmetic and Logic Operation) ประกอบด้วยคำสั่งสำหรับงานดำเนินการทางคณิตศาสตร์คือ คำสั่งบวก (Addition: ADD) คำสั่งลบ (Subtract: SUB) คำสั่งเพิ่มค่าหนึ่งค่า (Increment: INC) คำสั่งลดค่าหนึ่งค่า (Decrement: DEC) คำสั่งคูณ (Multiplicand: MUL) และประกอบด้วยคำสั่งสำหรับงานดำเนินการทางตรรกะ คือ คำสั่งเลื่อนทุกบิตไปทางซ้ายแบบไม่คิดเครื่องหมาย (Shift Logical Left: SLL) คำสั่งเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมาย (Shift Logical Right: SLR) คำสั่งเลื่อนทุกบิตไปทางซ้ายแบบคิดเครื่องหมาย (Shift Arithmetic Left: SAL) คำสั่งเลื่อนทุกบิตไปทางขวาแบบคิดเครื่องหมาย (Shift Arithmetic Right: SAR) คำสั่งกระทำลอจิกแอนด์ (Logical And: AND) คำสั่งกระทำลอจิกออร์ (Logical Or: OR) คำสั่งกระทำลอจิกเอ็กคลูซีฟออร์ (Logical Exclusive-or: XOR) คำสั่งกระทำบิตต่อบิตแอนด์ (Bitwise And: ANDB) คำสั่งกระทำบิตต่อบิตออร์ (Bitwise Or: ORB) และคำสั่งกระทำบิตต่อบิตเอ็กคลูซีฟออร์ (Bitwise Exclusive-or: XORB)

5.3 โครงสร้างของไมโครโพรเซสเซอร์

โครงสร้างของไมโครโพรเซสเซอร์เข้ารหัสหนึ่งในสี่ ปรับปรุงจากไมโครโพรเซสเซอร์เข้ารหัสรางคู่[17] ประกอบด้วย ส่วนอ่านคำสั่ง (Fetch Unit) ส่วนแปลความหมายของคำสั่ง (Decode Unit) ส่วนประมวลผล (Execute Unit) หน่วยคำนวณทางคณิตศาสตร์และตรรกะ (Arithmetic and Logic Units: ALU) ส่วนเขียนผลลัพธ์ (Write Result Unit) ส่วนบริการอินเตอร์รัพท์ (Interrupt Unit) และส่วนควบคุม (Control Unit) โดยมีรายละเอียดดังต่อไปนี้

5.3.1 ส่วนอ่านคำสั่งและส่วนแปลความหมายของคำสั่ง

ส่วนอ่านคำสั่ง ทำหน้าที่อ่านคำสั่งที่ไมโครโพรเซสเซอร์ได้รับมา และส่งต่อไปยังส่วนแปลความหมายของคำสั่ง เพื่อแปลความหมายคำสั่งออกเป็นขั้นตอนการทำงานย่อย จากนั้นส่งค่าที่ต้องใช้สำหรับการทำงานย่อยดังกล่าวไปยังส่วนประมวลผล เพื่อประมวลผลต่อไป ขั้นตอนการทำงานของส่วนอ่านคำสั่งและส่วนแปลความหมายของคำสั่ง เป็นดังรูปที่ 5.1 โดยมีขั้นตอนการทำงานดังนี้

ขั้นตอนที่ 1 เป็นดังรูป 5.1(ก) โดยค่าของรีจิสเตอร์ CS (Code Segment) และ PC (Program Counter) จะถูกเก็บไว้ที่รีจิสเตอร์ MAR (Memory Address Register) เพื่อชี้ตำแหน่งของคำสั่งที่ต้องการอ่าน จากนั้นอ่านรหัสดำเนินการของคำสั่ง 10'b บิตแรกเข้ามาเก็บไว้รีจิสเตอร์ MDR (Memory Data Register) และส่งต่อไปยังรีจิสเตอร์ IR1 (Instruction Register) เพื่อแปลความหมายของคำสั่งว่ามีความยาวของรหัสดำเนินการสูงสุดเท่าไหร่ และต้องอ่านข้อมูลจากหน่วยความจำหรือไม่ โดยแบ่งรูปแบบและการทำงานของคำสั่งออกเป็น

- คำสั่งที่มีความยาวของรหัสดำเนินการเท่ากับ 10'b บิต การทำงานเพียงขั้นตอนที่ 1 ก็สามารถอ่านรหัสดำเนินการได้ครบ จึงไม่ต้องดำเนินการขั้นตอนต่อไป
- คำสั่งที่มีความยาว่ของรหัสดำเนินการเท่ากับ 20'b บิต เมื่อเสร็จสิ้นขั้นตอนที่ 1 แล้ว จะทำขั้นตอนที่ 2 ต่อไป ดังรูปที่ 5.1(ข) คืออ่านรหัสดำเนินการอีก 10'b บิตถัดไปที่เหลือเข้ามาให้ครบ โดยเพิ่มค่าในรีจิสเตอร์ PC ขึ้นหนึ่งค่า และใช้ค่าในรีจิสเตอร์ CS และ PC เก็บไว้

ที่รีจิสเตอร์ MAR เพื่อชี้ไปยังตำแหน่งถัดไปของรหัสดำเนินการที่เหลือ จากนั้นอ่านรหัสดำเนินการที่เหลืออีก 10'b บิตเข้ามาเก็บไว้ในรีจิสเตอร์ MDR และส่งต่อไปยังรีจิสเตอร์ IR2

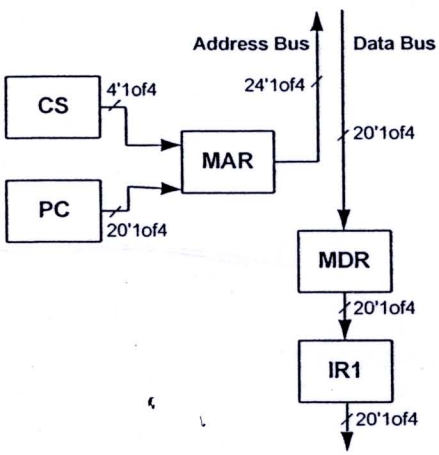
- คำสั่งที่มีความยาวของรหัสดำเนินการ 30'b บิต เมื่อเสร็จสิ้นขั้นตอนที่ 1 และขั้นตอนที่ 2 แล้ว จะทำขั้นตอนที่ 3 ต่อไป ดังรูปที่ 5.1(ค) คืออ่านรหัสดำเนินการอีก 10'b บิตถัดไปที่เหลือเข้ามาให้ครบ โดยเพิ่มค่าในรีจิสเตอร์ PC ขึ้นหนึ่งค่า และใช้ค่าในรีจิสเตอร์ CS และ PC เก็บไว้ที่รีจิสเตอร์ MAR เพื่อชี้ไปยังตำแหน่งถัดไปของรหัสดำเนินการที่เหลือ จากนั้นอ่านรหัสดำเนินการที่เหลืออีก 10'b บิตเข้ามาเก็บไว้ในรีจิสเตอร์ MDR และส่งต่อไปยังรีจิสเตอร์ IR3

- คำสั่งที่ต้องอ่านข้อมูลในหน่วยความจำ จากตำแหน่งซึ่งเป็นค่าคงที่ที่กำหนดขึ้นเองโดยไม่ผ่านดีเอ็มเอ เช่น LD A, @K เป็นต้น หลังจากทำขั้นตอนที่ 1 ถึง 3 ตามเงื่อนไขที่เหมาะสมแล้ว จะทำตามขั้นตอนที่ 4 ต่อไป ดังรูปที่ 5.1(ง) โดยใช้ค่าในรีจิสเตอร์ DS (Data Segment) และ IR2 เก็บไว้ที่รีจิสเตอร์ MAR เพื่อชี้ไปยังตำแหน่งของหน่วยความจำที่ต้องการอ่านข้อมูล จากนั้นอ่านข้อมูลจำนวน 8'b บิตจากหน่วยความจำ ผ่านบัสระบบเข้ามาเก็บไว้ในรีจิสเตอร์ MDR และส่งต่อไปยังรีจิสเตอร์ OP (Operand)

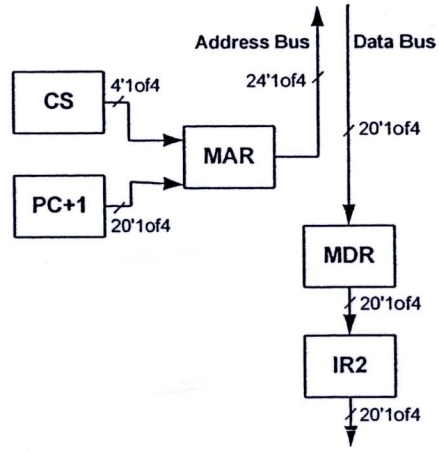
- คำสั่งที่ต้องอ่านข้อมูลในหน่วยความจำ จากตำแหน่งที่ค่าในรีจิสเตอร์ R0 หรือ R1 กำหนดโดยไม่ผ่านดีเอ็มเอ เช่น LD A, @Reg เป็นต้น หลังจากทำขั้นตอนที่ 1 ถึง 3 ตามเงื่อนไขที่เหมาะสมแล้ว จะทำตามขั้นตอนที่ 5 ต่อไป ดังรูปที่ 5.1(จ) โดยใช้ค่าในรีจิสเตอร์ DS และ R0 หรือ R1 เก็บไว้ที่รีจิสเตอร์ MAR เพื่อชี้ไปยังตำแหน่งของหน่วยความจำที่ต้องการอ่านข้อมูล จากนั้นอ่านข้อมูลจำนวน 8'b บิตจากหน่วยความจำ ผ่านบัสระบบเข้ามาเก็บไว้ในรีจิสเตอร์ MDR และส่งต่อไปยังรีจิสเตอร์ OP

- คำสั่งที่ต้องข้ามไปอ่านคำสั่งใหม่เข้ามา เช่น JMP CALL JZ JNZ เป็นต้น หลังจากทำขั้นตอนที่ 1 และ 2 แล้ว หากเป็นคำสั่งข้ามไปอ่านคำสั่งใหม่แบบไม่มีเงื่อนไข เช่น JMP CALL จะทำตามขั้นตอนที่ 6 ต่อไป แต่หากเป็นคำสั่งแบบมีเงื่อนไข เช่น JZ JNZ จะต้องรอผลลัพธ์การเปรียบเทียบเงื่อนไข จากค่าของแฟลกรีจิสเตอร์ ในส่วนประมวลผล โดยถ้าไม่เป็นไปตามเงื่อนไข จะสิ้นสุดการทำงานของคำสั่งนี้ แต่ถ้าเป็นไปตามเงื่อนไข จะทำตามขั้นตอนที่ 6 ต่อไปเช่นกัน ขั้นตอนที่ 6 เป็นดังรูปที่ 5.1(ฉ) คือส่วนอ่านคำสั่งจะอ่านคำสั่งใหม่ที่ตำแหน่งใหม่เข้ามาโดยใช้ค่าในรีจิสเตอร์ CS และ IR2 เก็บไว้ที่รีจิสเตอร์ MAR เพื่อชี้ตำแหน่งของคำสั่งใหม่ที่ต้องการอ่าน จากนั้นอ่านรหัสดำเนินการของคำสั่งใหม่ 10'b บิตแรกเข้ามาเก็บไว้รีจิสเตอร์ MDR และส่ง

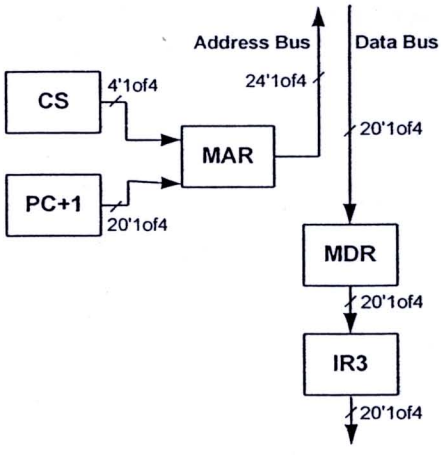
ต่อไปยังรีจิสเตอร์ IR1 เพื่อแปลความหมายของคำสั่งใหม่ว่ามีความยาวของรหัสดำเนินการสูงสุดเท่าไหร่ และต้องอ่านข้อมูลจากหน่วยความจำหรือไม่ จากนั้นทำงานตามความหมายของคำสั่งในขั้นตอนที่ 2 ถึง 6 ตามที่แปลได้



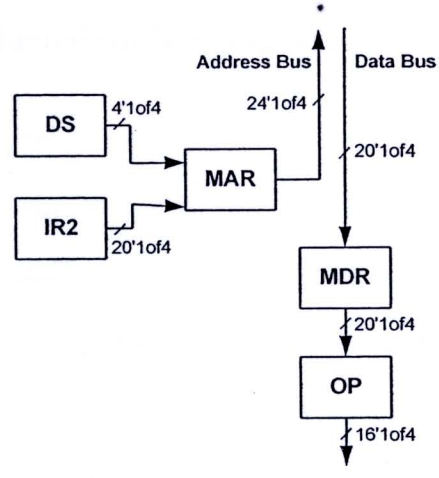
(ก) ขั้นตอนที่ 1



(ข) ขั้นตอนที่ 2

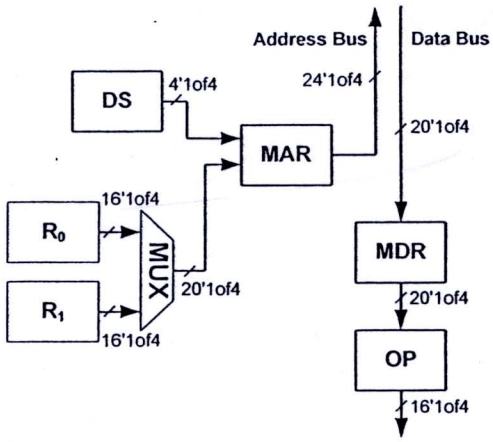


(ค) ขั้นตอนที่ 3

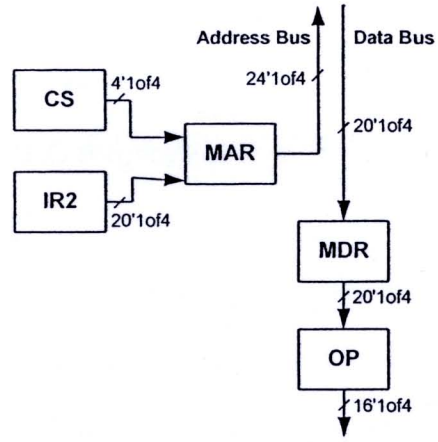


(ง) ขั้นตอนที่ 4

รูปที่ 5.1 ขั้นตอนการทำงานของส่วนอ่านคำสั่ง และส่วนแปลความหมายของคำสั่ง



(จ) ขั้นตอนที่ 5



(ฉ) ขั้นตอนที่ 6

รูปที่ 5.1 ขั้นตอนการทำงานของส่วนอ่านคำสั่ง และส่วนแปลความหมายของคำสั่ง (ต่อ)

หลังจากแปลความหมายของคำสั่งและทำงานในขั้นตอนที่ 1 ถึง 6 ตามเงื่อนไขของคำสั่งแล้ว จะสิ้นสุดการทำงานในส่วนนี้ และไปดำเนินการในส่วนประมวลผลต่อไป

5.3.2 ส่วนประมวลผลและหน่วยคำนวณทางคณิตศาสตร์และตรรกะ

ส่วนประมวลผล มีโครงสร้างวงจรดังรูปที่ 5.2 ทำหน้าที่ส่งค่ามาประมวลผลที่หน่วยคำนวณทางคณิตศาสตร์และตรรกะ หรือ ALU ในกลุ่มคำสั่งดำเนินการทางคณิตศาสตร์และตรรกะ แล้วเก็บผลลัพธ์ไว้ที่รีจิสเตอร์ตัวสะสม หรือ รีจิสเตอร์ ACC นอกจากนี้ ส่วนประมวลผลยังทำหน้าที่ส่งผ่านค่าในกลุ่มคำสั่งเคลื่อนย้ายข้อมูล สำหรับค่าในแฟลกรีจิสเตอร์คือ ZF (Zero Flag) CF (Carry Flag) OF (Overflow Flag) และ EI (Enable Interrupt Flag) จะถูกนำไปใช้พิจารณาเงื่อนไขในคำสั่งกลุ่มควบคุมการทำงาน โดยความหมายของค่าในแฟลกรีจิสเตอร์แต่ละแบบมีดังต่อไปนี้

ZF = 0 หมายความว่า ค่าในรีจิสเตอร์ ACC มีค่าไม่เท่ากับ 0

ZF = 1 หมายความว่า ค่าในรีจิสเตอร์ ACC มีค่าเท่ากับ 0

CF = 0 หมายความว่า ค่าในรีจิสเตอร์ ACC ไม่มีตัวทด

CF = 1 หมายความว่า ค่าในรีจิสเตอร์ ACC มีตัวทด

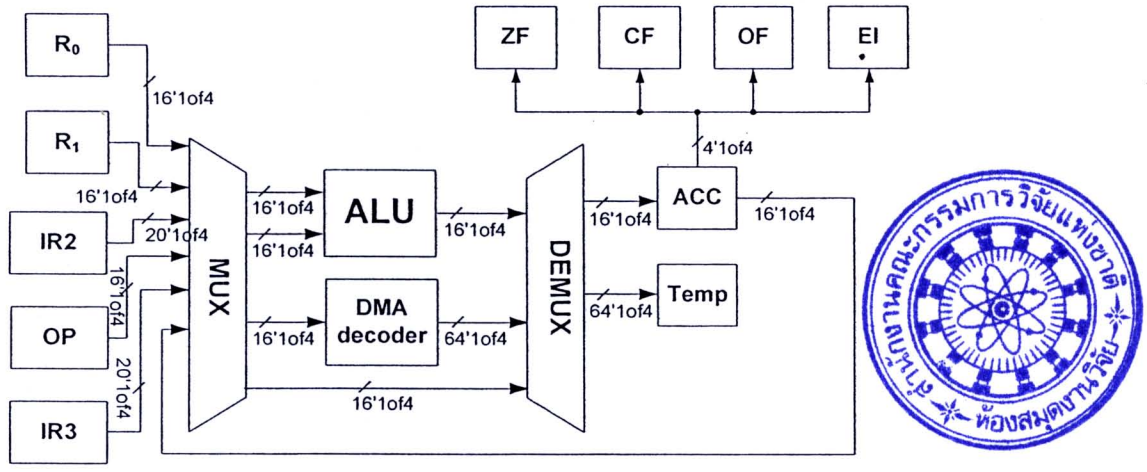
OF = 0 หมายความว่า ค่าในรีจิสเตอร์ ACC มีค่าไม่เกินจำนวนบิตสูงสุด

OF = 1 หมายความว่า ค่าในรีจิสเตอร์ ACC มีค่าเกินจำนวนบิตสูงสุด

EI = 0 หมายความว่า ค่าในรีจิสเตอร์ ACC ยังไม่สมบูรณ์และยังทำงานตามคำสั่งไม่เสร็จสิ้น ไม่อนุญาตให้ตอบสนองงานบริการอินเตอร์รัพท์

EI = 1 หมายความว่า ค่าในรีจิสเตอร์ ACC สมบูรณ์และเสร็จสิ้นการทำงานตามคำสั่งแล้ว อนุญาตให้ตอบสนองงานบริการอินเตอร์รัพท์ได้

จากรูปที่ 5.2 หากข้อมูลที่ใช้การประมวลผลคือค่าคงที่ที่กำหนดขึ้นเอง (#K) ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ IR2 มาประมวลผล หากข้อมูลที่ใช้ประมวลผลคือค่าที่อ่านมาจากรีจิสเตอร์ทั่วไป R0 หรือ R1 (Reg) ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ R0 หรือ R1 มาประมวลผล หากข้อมูลที่ใช้ประมวลผลคือค่าที่อ่านมาจากหน่วยความจำ (@K หรือ @Reg) ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ OP มาประมวลผล หากเป็นคำสั่งที่เรียกใช้ดีเอ็มเอ ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ IR2 และ IR3 มาประมวลผล ตัวอย่างการทำงานของส่วนประมวลผลในกลุ่มคำสั่งต่างๆมีดังนี้



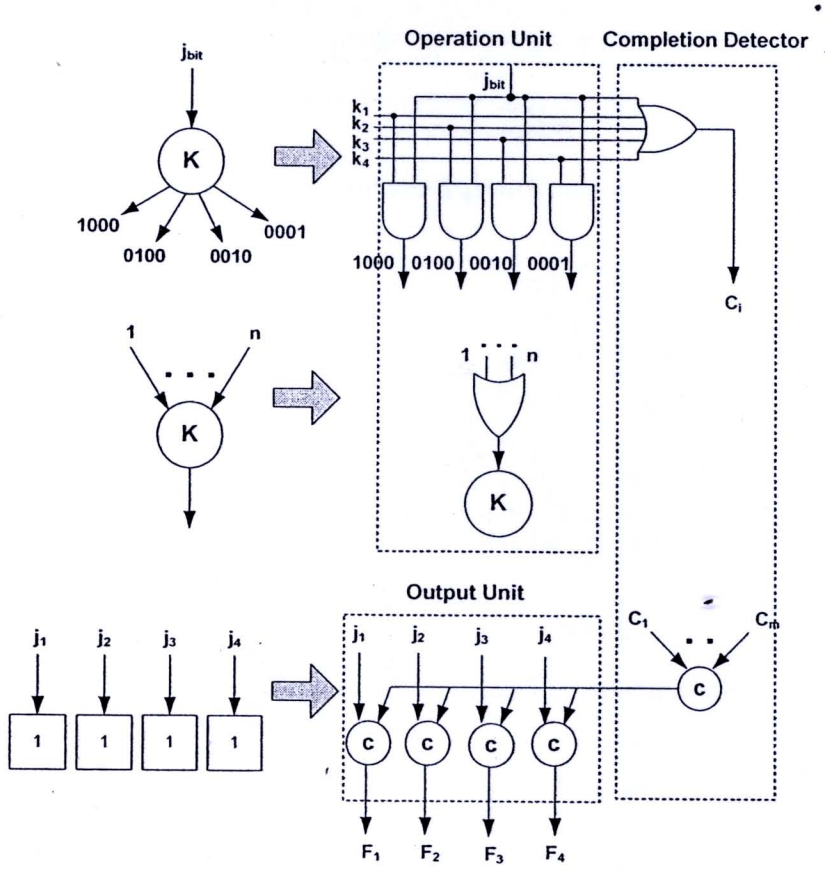
รูปที่ 5.2 ส่วนประมวลผล

- การทำงานในกลุ่มคำสั่งดำเนินการทางคณิตศาสตร์ เช่น การทำงานในคำสั่ง ADD A, @K มีขั้นตอนคือ ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ ACC และ รีจิสเตอร์ OP ไปประมวลผลที่วงจรบวกภายใน ALU และเก็บผลลัพธ์ที่ได้จากการประมวลผลไว้ในรีจิสเตอร์ ACC

- การทำงานในกลุ่มคำสั่งดำเนินการทางตรรกะ เช่น การทำงานในคำสั่ง SLL A มีขั้นตอนคือ ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ ACC ไปประมวลผลที่วงจรเลื่อนทุกบิตไปทางซ้ายแบบไม่คิดเครื่องหมายภายใน ALU และเก็บผลลัพธ์ที่ได้จากการประมวลผลไว้ในรีจิสเตอร์ ACC

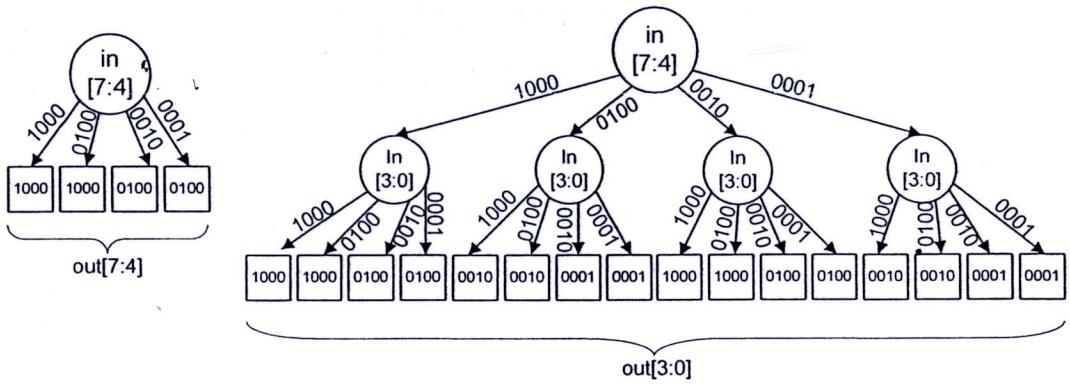
- การทำงานในกลุ่มคำสั่งเคลื่อนย้ายข้อมูล เช่น การทำงานในคำสั่ง LD A, #K มีขั้นตอนคือ ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ IR2 ส่งผ่านไปเก็บที่รีจิสเตอร์ ACC โดยไม่ผ่าน ALU สำหรับการทำงานในกลุ่มคำสั่งเคลื่อนย้ายข้อมูลซึ่งมีการเรียกใช้ดีเอ็มเอ เช่น การทำงานในคำสั่ง IN @K2,@K มีขั้นตอนคือ ส่วนประมวลผลจะนำค่าจากรีจิสเตอร์ IR2 และ IR3 ไปประมวลผลที่วงจรถอดรหัสคำสั่งดีเอ็มเอ (DMA Decoder) และเก็บผลลัพธ์ที่ได้จากการถอดรหัสไว้ในรีจิสเตอร์ Temp

- การทำงานในกลุ่มควบคุมการทำงาน เช่น การทำงานในคำสั่ง JZ Address มีขั้นตอนคือ ส่วนประมวลผลจะตรวจสอบค่าในแฟลกรีจิสเตอร์ ZF โดยหาก ZF มีค่าเท่ากับ 1 จะกลับไปทำงานในขั้นตอนที่ 6 ของส่วนอ่านคำสั่งและส่วนแปลของคำสั่ง แต่หาก ZF มีค่าเท่ากับ 0 จะสิ้นสุดการทำงานของคำสั่งนี้

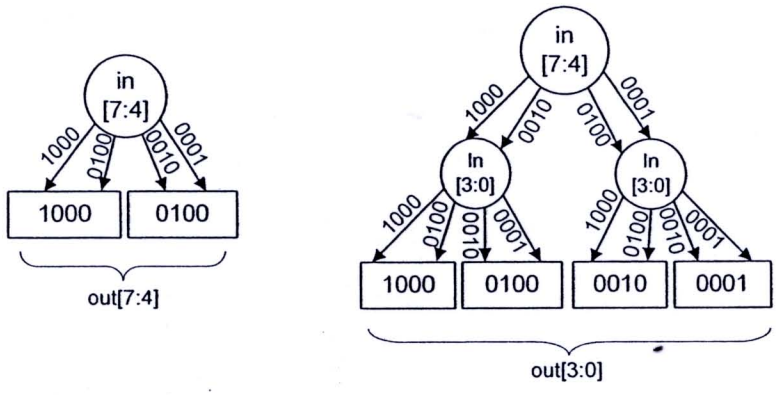


รูปที่ 5.3 การแปลงแผนภาพตัดสินใจแบบทวิภาคชนิดลดทอนอันดับเป็นวงจรหนึ่งในสี่

วงจรฟังก์ชันภายใน ALU ประกอบด้วย วงจรบวก วงจรลบ วงจรเพิ่มค่าหนึ่งค่า วงจรลดค่าหนึ่งค่า วงจรคูณ วงจรเลื่อนทุกบิตไปทางซ้ายแบบไม่คิดเครื่องหมาย วงจรเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมาย วงจรเลื่อนทุกบิตไปทางซ้ายแบบคิดเครื่องหมาย วงจรเลื่อนทุกบิตไปทางขวาแบบคิดเครื่องหมาย วงจรกระทำลอจิกแอนด์ วงจรกระทำลอจิกออร์ วงจรกระทำลอจิกเอ็กคลูซีฟออร์ วงจรกระทำบิตต่อบิตแอนด์ วงจรกระทำบิตต่อบิตออร์ และวงจรกระทำบิตต่อบิตเอ็กคลูซีฟออร์ วงจรดังกล่าวทั้งหมดเข้ารหัสหนึ่งในสี่ การออกแบบวงจรใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับหรือ ROBDD ในการออกแบบ โดยหลักการแปลงแผนภาพ ROBDD เป็นวงจรหนึ่งในสี่แสดงดังรูปที่ 5.3

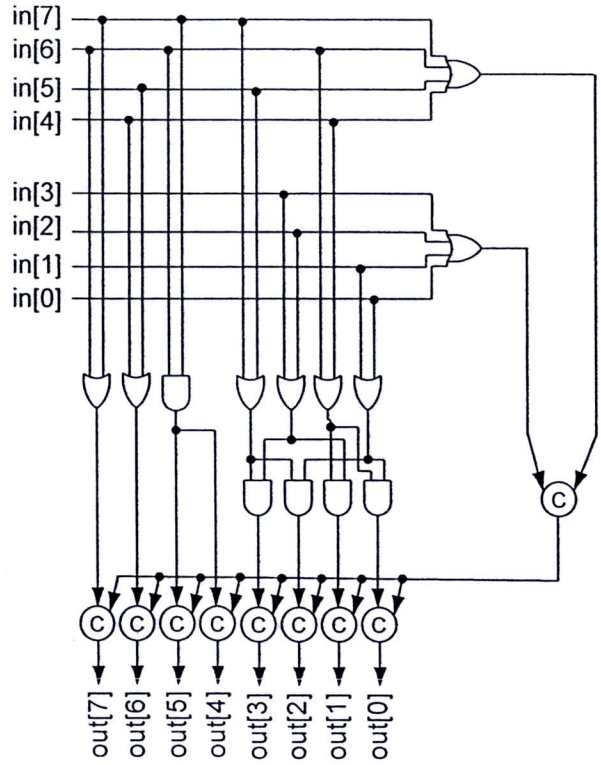


(ก) แผนภาพการตัดสินใจแบบทวิภาค



(ข) แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ

รูปที่ 5.4 การออกแบบวงจรหนึ่งในสี่ของฟังก์ชันเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมาย ขนาด 8 บิต โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ

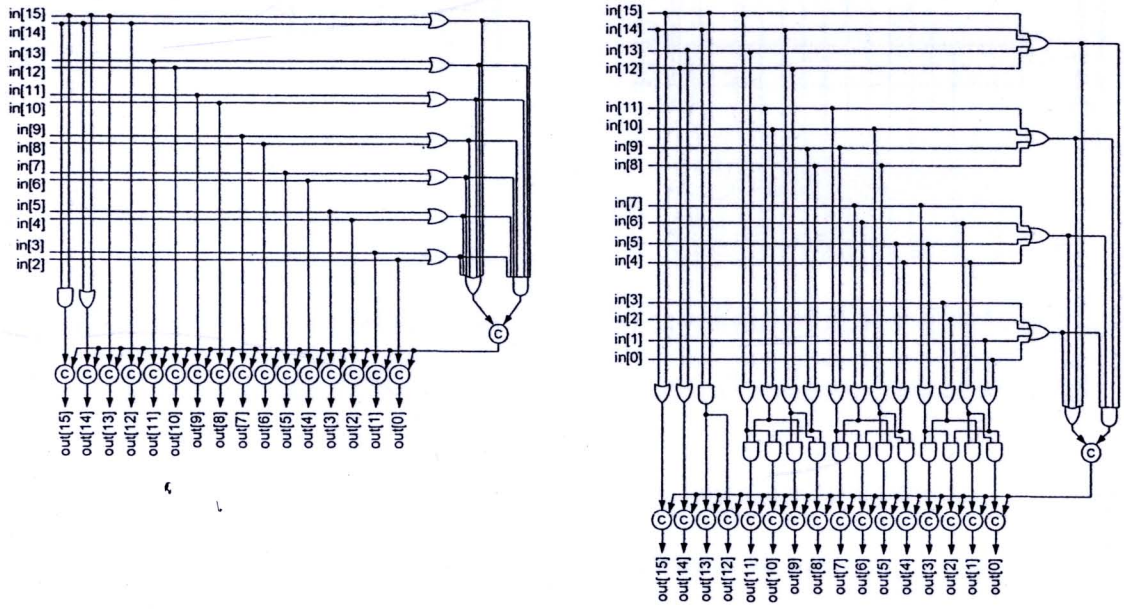


(ค) วงจรหนึ่งในสี่

รูปที่ 5.4 การออกแบบวงจรหนึ่งในสี่ของฟังก์ชันเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมาย ขนาด 8 บิต โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ (ต่อ)

รูปที่ 5.4 แสดงการออกแบบวงจรหนึ่งในสี่ของวงจรฟังก์ชันเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมายขนาด 8 บิต โดยใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ ซึ่งมีขั้นตอนตามขั้นตอนการออกแบบวงจรใช้แผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับในบทที่ 2 กล่าวคือ สร้างแผนภาพการตัดสินใจแบบทวิภาค โดยแทนค่าตัวแปรในฟังก์ชันเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมาย และเขียนแจกแจงเป็นโครงสร้างที่มีอันดับชั้น ได้ดังรูปที่ 5.4(ก) โดยแต่ละกิ่งเป็นการแทนค่าของลอจิกในฟังก์ชัน เช่น ในวงจรที่สร้างเอาท์พุท out บิตที่ 3 ถึง 0 กิ่งซ้ายสุดของแผนภาพวงจรดังกล่าวหมายถึง เมื่อค่าลอจิกของอินพุท in บิตที่ 7 ถึง 4 มีค่าเท่ากับ 1000 และ อินพุท in บิตที่ 3 ถึง 0 มีค่าเท่ากับ 1000 แล้วค่าลอจิกของเอาท์พุท out บิตที่ 3 ถึง 0 จะมีค่าเท่ากับ 1000 เป็นต้น จากนั้นลดทอนอันดับแผนภาพดังกล่าวจนได้ดังรูปที่ 5.4(ข) ในขั้นตอนสุดท้ายจะแปลงแผนภาพที่ลดทอนแล้วเป็นวงจรหนึ่งในสี่ของฟังก์ชัน

เลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมาย ตามหลักการในรูปที่ 5.3 จนได้เป็นวงจรเข้ารหัสหนึ่งในสี่ดังรูปที่ 5.4(ค) เป็นอันเสร็จสิ้นขั้นตอน

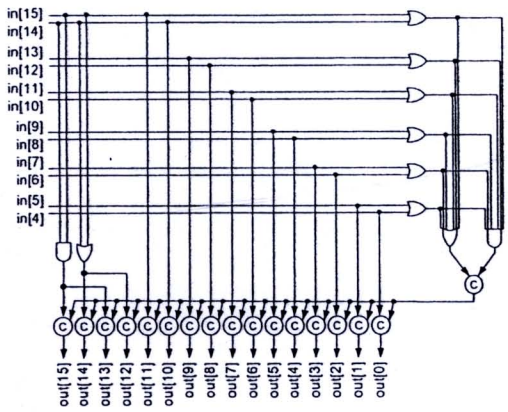


(ก) วงจรเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมายเข้ารหัสรางคู่

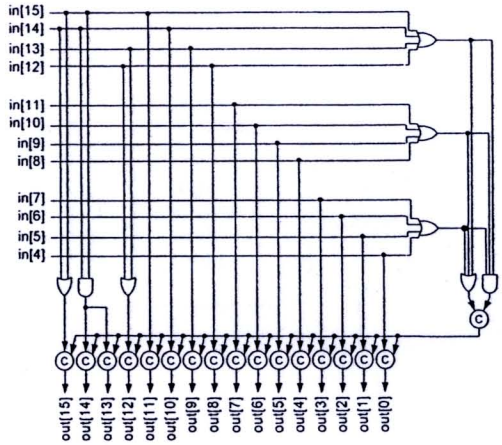
(ข) วงจรเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมายเข้ารหัสหนึ่งในสี่

รูปที่ 5.5 วงจรเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมายเข้ารหัสรางคู่และเข้ารหัสหนึ่งในสี่ขนาด 16 บิต

วงจรฟังก์ชันเข้ารหัสหนึ่งในสี่ที่สร้างจากแผนภาพตัดสินใจแบบทวิภาคชนิดมีการลดทอนอันดับ มีขนาดใหญ่กว่าวงจรฟังก์ชันเข้ารหัสรางคู่ซึ่งสร้างจากวิธีเดียวกัน ดังแสดงในรูปที่ 5.5 จะพบว่าวงจรเลื่อนทุกบิตไปทางขวาแบบไม่คิดเครื่องหมายแบบเข้ารหัสรางคู่ มีขนาดเล็กกว่าแบบเข้ารหัสหนึ่งในสี่ ทั้งนี้หากเป็นวงจรฟังก์ชันที่มีการคำนวณมากกว่าครั้งละ 1 หลัก เช่น วงจรเลื่อนทุกบิตไปทางขวาครั้งละ 2 บิตแบบไม่คิดเครื่องหมาย ซึ่งคำนวณครั้งละ 2 หลัก วงจรดังกล่าวแบบเข้ารหัสหนึ่งในสี่จะมีขนาดเล็กลง และมีขนาดใกล้เคียงกับวงจรแบบเข้ารหัสรางคู่ ดังแสดงในรูปที่ 5.6



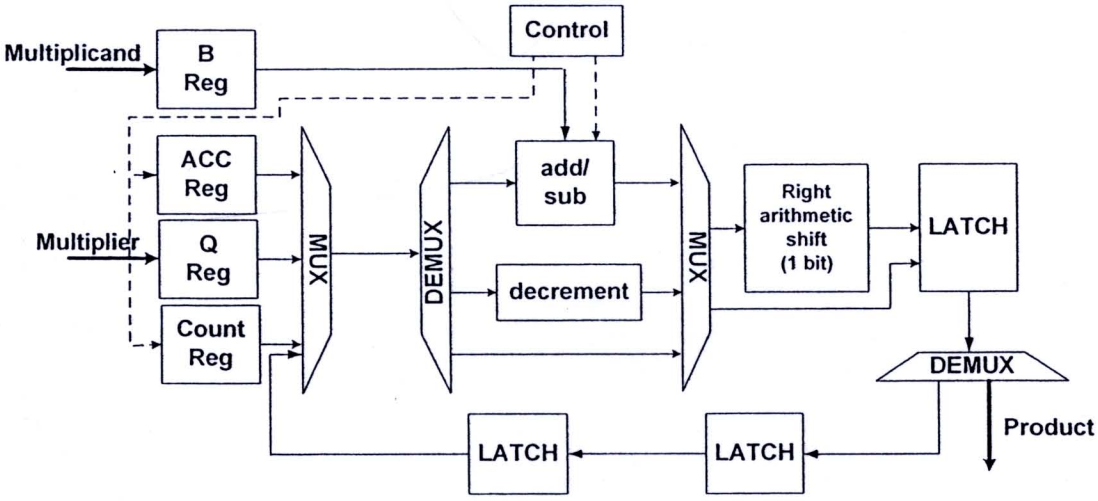
(ก) วงจรเลื่อนทุกบิตไปทางขวาครั้งละ 2 บิต แบบไม่คิดเครื่องหมายเข้ารหัสรางคู่



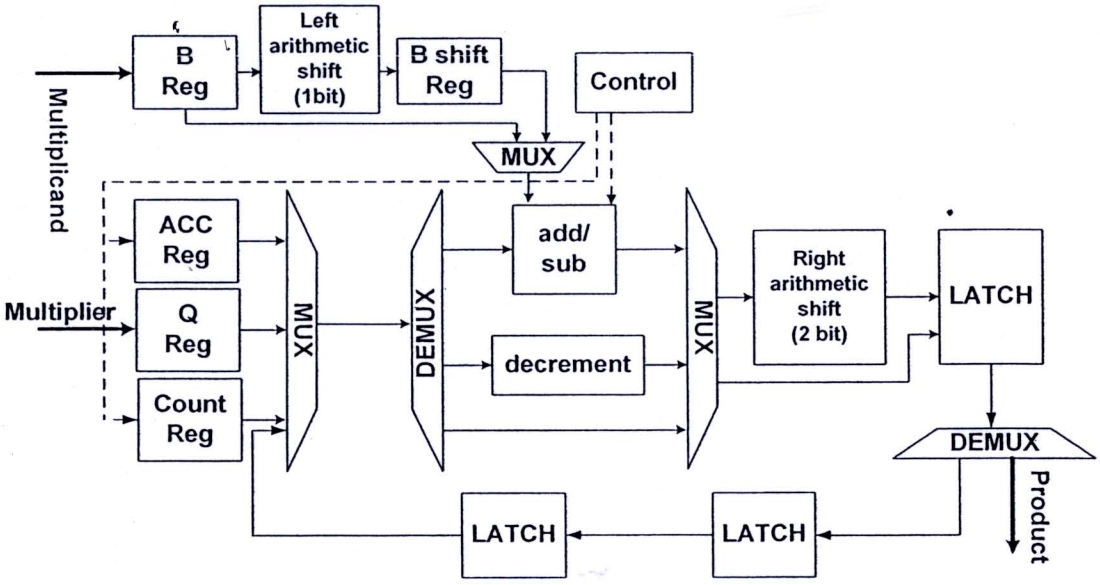
(ข) วงจรเลื่อนทุกบิตไปทางขวาครั้งละ 2 บิต แบบไม่คิดเครื่องหมายเข้ารหัสหนึ่งในสี่

รูปที่ 5.6 วงจรเลื่อนทุกบิตไปทางขวาครั้งละ 2 บิตแบบไม่คิดเครื่องหมาย เข้ารหัสรางคู่และเข้ารหัสหนึ่งในสี่ขนาด 16 บิต

ดังนั้นวงจรคูณเข้ารหัสหนึ่งในสี่ จะมีประสิทธิภาพดีขึ้นเมื่อเป็นวงจรคูณครั้งละ 2 หลักซึ่งประกอบด้วยวงจรเลื่อนทุกบิตไปทางขวาครั้งละ 2 บิตแบบไม่คิดเครื่องหมายเป็นส่วนประกอบ อีกทั้งหลักการทำงานของวงจรคูณครั้งละ 2 หลัก [25] จะทำงานจำนวน n/2 รอบเพื่อคูณตัวตั้ง (Multiplicand) และตัวคูณ(Multiplier) จำนวน n บิตออกมาเป็นผลคูณ (Product) ที่สมบูรณ์ แต่วงจรคูณครั้งละ 1 หลัก ต้องทำงานจำนวน n รอบเพื่อคูณตัวตั้งและตัวคูณจำนวน n บิตออกมาเป็นผลคูณที่สมบูรณ์ ในงานวิจัยนี้จึงออกแบบวงจรคูณบูทอัลกอลิทึม (Booth Algorithm) ครั้งละ 1 หลักเข้ารหัสหนึ่งในสี่ใช้ฟังก์ชันเข้ารหัสหนึ่งในสี่ ดังรูปที่ 5.7(ก) และพัฒนาเป็นวงจรคูณบูทอัลกอลิทึมครั้งละ 2 หลักเข้ารหัสหนึ่งในสี่ใช้ฟังก์ชันเข้ารหัสหนึ่งในสี่ (1-of-4 Radix-4 Booth Multiplier with 1-of-4 Function Unit) ดังรูปที่ 5.7(ข) เพื่อให้การคำนวณข้อมูลเข้ารหัสหนึ่งในสี่ที่ใช้ในงานวิจัยมีประสิทธิภาพดีขึ้น โดยจะทดลองประสิทธิภาพของวงจรคูณทั้งสองแบบ เพื่อหาข้อสรุปทางด้านประสิทธิภาพที่ชัดเจนต่อไปในบทที่ 6



(ก) วงจรคูณครั้งละ 1 หลักเข้ารหัสหนึ่งในสี่ใช้ฟังก์ชันเข้ารหัสหนึ่งในสี่



(ข) วงจรคูณครั้งละ 2 หลักเข้ารหัสหนึ่งในสี่ใช้ฟังก์ชันเข้ารหัสหนึ่งในสี่

รูปที่ 5.7 วงจรคูณครั้งละ 1 หลักและวงจรคูณครั้งละ 2 หลักเข้ารหัสหนึ่งในสี่

หลักการทำงานของวงจรวงจรคูณครั้งละ 2 หลักเข้ารหัสหนึ่งในสี่ที่ออกแบบ จะพิจารณาเงื่อนไขผ่านบิตที่เรียงต่อกันไปของตัวคูณจนครบทุกบิต ตามหลักการคูณของบูทอัลกอลิทึม โดยเมื่อเข้ารหัสหนึ่งในสี่แล้ว เงื่อนไขที่พิจารณาจะเป็นดังตารางที่ 5.1

ตารางที่ 5.1 เงื่อนไขการคูณครั้งละ 2 หลักของบูทอัลกอลิทึมเข้ารหัสหนึ่งในสี่

m_{i+1}	mi	m_{i-1}	เงื่อนไข	กระทำ
1000		01	เงื่อนไขที่ 5	ไม่ทำอะไร
1000		10	เงื่อนไขที่ 1	บวกด้วยตัวตั้ง
0100		01	เงื่อนไขที่ 1	บวกด้วยตัวตั้ง
0100		10	เงื่อนไขที่ 2	เลื่อนซ้ายตัวตั้ง 1 บิต, บวกด้วยตัวตั้ง
0010		01	เงื่อนไขที่ 4	เลื่อนซ้ายตัวตั้ง 1 บิต, ลบด้วยตัวตั้ง
0010		10	เงื่อนไขที่ 3	ลบด้วยตัวตั้ง
0001		01	เงื่อนไขที่ 3	ลบด้วยตัวตั้ง
0001		10	เงื่อนไขที่ 5	ไม่ทำอะไร

ตัวอย่างการคูณรหัสหนึ่งในสี่ 1000_1000_0100_0100 (5) ด้วยจำนวน 0010_0100_0001_0001 (-97) ตามเงื่อนไขการคูณของบูทอัลกอลิทึมเข้ารหัสหนึ่งในสี่ จากตารางที่ 5.1 เป็นดังนี้ โดยเมื่อเปรียบเทียบกับการคูณรหัสฐานสองแล้ว การคูณรหัสหนึ่งในสี่จำนวน 16 บิตจะเท่ากับคูณข้อมูลฐานสองขนาด 8 บิต (8'b บิต =16'1of4 บิต) จึงแบ่งการทำงานออกเป็น 4 รอบ (8/2=4)

เริ่มต้นจากเติมลอจิก 0 ที่บิตขวาสุดของตัวคูณ (บิตขวาสุดที่เติมมีเพียงบิตเดียว จึงเข้ารหัสบิตขวาสุดเป็นรหัสรางคู่ เนื่องจากการเข้ารหัสข้อมูลบิตเดียวด้วยรหัสหนึ่งในสี่จะทำให้วงจรที่ได้มีขนาดใหญ่ดังที่กล่าวไปแล้ว นอกเหนือจากนี้เป็นรหัสหนึ่งในสี่ทั้งหมด) จากนั้นพิจารณาบิตตัวคูณรวมทั้งบิตที่เติมเข้ามา และกระทำตามเงื่อนไข

รอบที่ 1 0010_0100_0001_**0001 | 01** (กระทำตามเงื่อนไขที่ 3)

1000_1000_1000_1000_1000_1000_1000_1000 ลบ

1000_1000_0100_0100 ได้

0001_0001_0001_0001_0001_0001_0010_0001

จากนั้นเลื่อนขวาบิตตัวคูณรวมบิตที่เติมเข้ามา 2 บิต เพื่อตรวจสอบเงื่อนไขในบิตถัดไป ได้เป็น 1000_0010_0100_0001 | 10

รอบที่ 2 1000_0010_0100_**0001 | 10** (กระทำตามเงื่อนไขที่ 5)

0001_0001_0001_0001_0001_0001_0010_0001

จากนั้นเลื่อนขวาบิตตัวคูณรวมบิตที่เติมเข้ามา 2 บิต เพื่อตรวจสอบเงื่อนไขในบิตถัดไป ได้เป็น 1000_1000_0010_0100 | 10

รอบที่ 3 1000_1000_0010_**0100 | 10** (กระทำตามเงื่อนไขที่ 2)

0001_0001_0001_0001_0001_0001_0010_0001 บวก

1000_1000_0010_0010 ได้

1000_1000_1000_1000_0010_0100_0010_0001

จากนั้นเลื่อนขวาบิตตัวคูณรวมบิตที่เติมเข้ามา 2 บิต เพื่อตรวจสอบเงื่อนไขในบิตถัดไป ได้เป็น 1000_1000_1000_0010 | 01

รอบที่ 4 1000_1000_1000_**0010 | 01** (กระทำตามเงื่อนไขที่ 4)

1000_1000_1000_1000_0010_0100_0010_0001 ลบ

1000_1000_0010_0010 ได้

0001_0001_0001_0010_1000_0100_0010_0001 (-485)

ดังนั้น คูณรหัสหนึ่งในสี่ 1000_1000_0100_0100 (5) ด้วยจำนวน

0010_0100_0001_0001 (-97) ได้ผลลัพธ์เท่ากับ

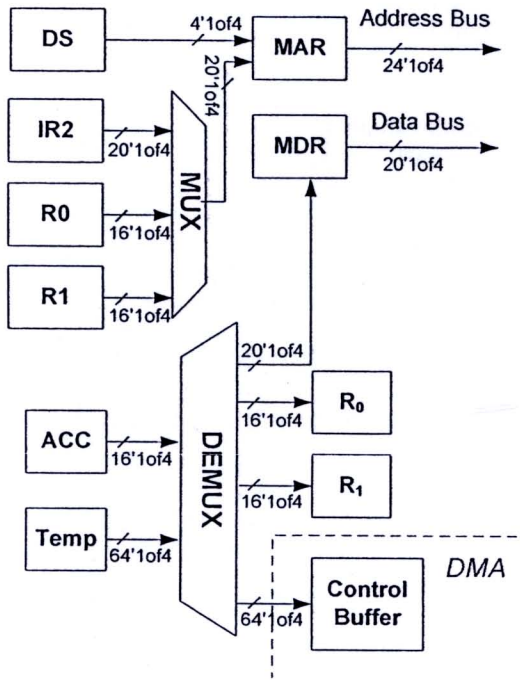
0001_0001_0001_0010_1000_0100_0010_0001 (-485)

5.3.3 **ส่วนเขียนผลลัพธ์**

ส่วนเขียนผลลัพธ์ ทำหน้าที่รับผลลัพธ์จากส่วนประมวลผลมาเขียนลงที่ รีจิสเตอร์หรือหน่วยความจำ ตามคำสั่ง โดยถ้าเป็นคำสั่งที่เรียกใช้งานดีเอ็มเอ ส่วนเขียนผลลัพธ์จะเขียนผลลัพธ์ลงที่บัฟเฟอร์ควบคุมของดีเอ็มเอซึ่งออกแบบไว้ในบทที่ 4 และดีเอ็มเอจะทำงานตามค่าในบัฟเฟอร์ควบคุมต่อไป โดยไม่ต้องเป็นภาระให้กับไมโครโพรเซสเซอร์ วงจรส่วนเขียนผลลัพธ์แสดงดังรูปที่ 5.8 ตัวอย่างการทำงานของส่วนเขียนผลลัพธ์ในคำสั่งต่างๆมีดังนี้

- คำสั่งที่ตัวรับผลลัพธ์เป็นรีจิสเตอร์ทั่วไป R0 หรือ R1 ส่วนเขียนผลลัพธ์จะนำค่าจากรีจิสเตอร์ ACC ไปเขียนผลลัพธ์ลงที่รีจิสเตอร์ทั่วไป R0 หรือ R1
- คำสั่งที่ตัวรับผลลัพธ์เป็นหน่วยความจำ และกำหนดให้เขียนผลลัพธ์ลงที่ตำแหน่งซึ่งเป็นค่าคงที่ที่กำหนดขึ้นเอง ส่วนเขียนผลลัพธ์จะใช้ค่าในรีจิสเตอร์ DS และ IR2 เก็บไว้

ที่รีจิสเตอร์ MAR เพื่อชี้ไปยังตำแหน่งที่ต้องการเขียนข้อมูลในหน่วยความจำ จากนั้นนำผลลัพธ์จากรีจิสเตอร์ ACC จำนวน 16'1of4 บิตเก็บไว้ที่ MDR และส่งต่อไปยังบัสระบบเพื่อเขียนผลลัพธ์ดังกล่าวไว้ในหน่วยความจำ



รูปที่ 5.8 ส่วนเขียนผลลัพธ์

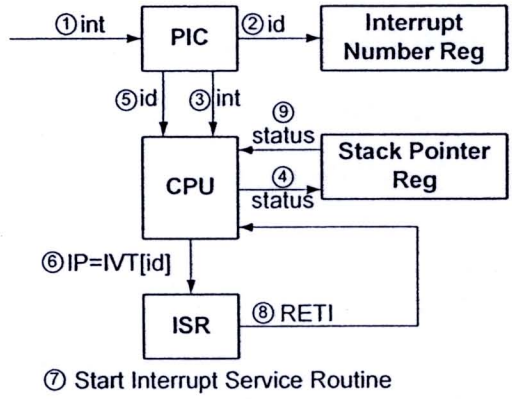
- คำสั่งที่ตัวรับผลลัพธ์เป็นหน่วยความจำ และกำหนดให้เขียนผลลัพธ์ลงที่ตำแหน่งที่ค่าในรีจิสเตอร์ R0 หรือ R1 กำหนด ส่วนเขียนผลลัพธ์จะใช้ค่าในรีจิสเตอร์ DS และ R0 หรือ R1 เก็บไว้ที่รีจิสเตอร์ MAR เพื่อชี้ไปยังตำแหน่งที่ต้องการเขียนข้อมูลในหน่วยความจำ จากนั้นนำผลลัพธ์จากรีจิสเตอร์ ACC จำนวน 16'1of4 บิตเก็บไว้ที่ MDR และส่งต่อไปยังบัสระบบเพื่อเขียนผลลัพธ์ดังกล่าวไว้ในหน่วยความจำ

- คำสั่งที่เรียกใช้งานดีเอ็มเอ ส่วนเขียนผลลัพธ์จะนำค่าจากรีจิสเตอร์ Temp ไปเขียนผลลัพธ์ลงที่บัฟเฟอร์ควบคุมของดีเอ็มเอ เพื่อใช้เป็นค่าเริ่มต้นการทำงานของดีเอ็มเอต่อไป

5.3.4 ส่วนบริการอินเตอร์รัพท์

ส่วนบริการอินเตอร์รัพท์ปรับปรุงจากส่วนบริการอินเตอร์รัพท์ในไมโครโพรเซสเซอร์เวอร์ชันเดิม [4] ทำหน้าที่รองรับการใช้งานอินเตอร์รัพท์ โดยมีหลักการคือ เมื่อมี

สัญญาณอินเตอร์รัพท์จากอุปกรณ์ต่อพ่วงเข้ามา CPU จะถูกขัดจังหวะการทำงานเดิม ให้ไปทำงานใหม่ตามที่กำหนดก่อน และเมื่อทำงานดังกล่าวเสร็จสิ้น CPU จะกลับไปทำงานเดิมต่อไป



รูปที่ 5.9 ขั้นตอนบริการอินเตอร์รัพท์

รูปที่ 5.9 แสดงขั้นตอนการทำงานของส่วนบริการอินเตอร์รัพท์ ซึ่งมีดังต่อไปนี้

1. เมื่อมีสัญญาณอินเตอร์รัพท์ (int) จากอุปกรณ์เข้ามา จะส่งสัญญาณอินเตอร์รัพท์ไปที่ PIC เพื่อตรวจสอบว่าเป็นสัญญาณอินเตอร์รัพท์จากอุปกรณ์หมายเลข (id) ใด
2. ส่งค่าหมายเลขของอุปกรณ์ ไปเก็บไว้ที่รีจิสเตอร์ Interrupt Number
3. ส่งค่าสัญญาณอินเตอร์รัพท์ไปที่ CPU
4. CPU เก็บสถานะ (Status) ของการทำงานปัจจุบันใส่ SP (Stack Pointer)
5. PIC ส่งค่าหมายเลขของอุปกรณ์มาที่ CPU
6. CPU คำนวณตำแหน่งของงาน (Instruction Pointer: IP) ใหม่ที่ต้องไปทำ ซึ่งเป็นงานของอินเตอร์รัพท์ (Interrupt Service Routine: ISR) โดยตรวจสอบตำแหน่งจากตารางตำแหน่งอินเตอร์รัพท์ (Interrupt Vector Table: IVT)
7. CPU เริ่มทำงานของอินเตอร์รัพท์
8. เมื่อสิ้นสุดงานอินเตอร์รัพท์ จะส่งสัญญาณ RETI (Return from Interrupt) ไปที่ CPU
9. CPU ดึงสถานะเก่าในข้อ 4. จาก SP คืนมา และทำงานต่อ

5.3.5 ส่วนควบคุม

ส่วนควบคุมไมโครโพรเซสเซอร์ทำหน้าที่ควบคุมส่วนต่างๆของไมโครโพรเซสเซอร์ให้ทำงานร่วมกันตามคำสั่งที่ได้รับได้อย่างถูกต้อง ส่วนควบคุมออกแบบโดยใช้อุปกรณ์ไปป์ไลน์ควบคุมสัญญาณร้องขอและสัญญาณตอบรับ เส้นทางการรับส่งข้อมูลภายในไมโครโพรเซสเซอร์ถูกควบคุมโดยวงจรควบคุมซึ่งสร้างจากกราฟบรรยายการเปลี่ยนสัญญาณ และใช้โปรแกรม Petrify สร้างวงจร เช่นเดียวกับการออกแบบตัวควบคุมบัสในบทที่ 3 ที่ได้กล่าวไปแล้ว

5.4 การทำงานของไมโครโพรเซสเซอร์

ส่วนอ่านคำสั่งของไมโครโพรเซสเซอร์จะอ่านคำสั่งจากแคชเก็บคำสั่ง (Instruction Cache) จากนั้นส่วนแปลความหมายของคำสั่งจะแปลความหมายของคำสั่งว่าต้องทำงานใดบ้าง แล้วส่งค่าไปประมวลผลที่ส่วนประมวลผลตามงานของคำสั่งที่แปลไว้ เมื่อประมวลผลเสร็จสิ้น ส่วนเขียนผลลัพธ์จะเขียนผลลัพธ์ที่ได้จากการประมวลผลไว้ที่รีจิสเตอร์ หรือหน่วยความจำ หรือบัฟเฟอร์ควบคุมของดีเอ็มเอ ซึ่งเป็นการสิ้นสุดการทำงานของคำสั่ง จากนั้นไมโครโพรเซสเซอร์จะเริ่มการทำงานของคำสั่งใหม่ โดยอ่านคำสั่งใหม่ที่ตำแหน่งถัดไปในแคชเก็บคำสั่ง และทำงานในคำสั่งถัดไปตามขั้นตอนที่ได้กล่าวมาอย่างต่อเนื่อง