บทที่ 4

ซอฟท์แวร์ของเครื่องควบคุมชนิดโปรแกรมได้

โปรแกรมซอฟท์แวร์ควบคุมระบบของเครื่อง PC ในการวิจัยนี้เป็นภาษาแอสเซมบลี้ (Assembly language) เพราะเป็นภาษาที่มีความเร็วในการทำงานสูง และสามารถเขียนโปรแกรมควบคุมในรายละเอียดส่วนต่าง ๆ ของวงจรได้ดี การเขียนโปรแกรมจะเป็นลักษณะโมดูล แล้วนำมาเชื่อมต่อกัน (Link) เป็นโปรแกรมรวมอีกครั้ง การพัฒนาหรือแก้ไขเพิ่มเติมในอนาคตสามารถทำได้สะดวก โปรแกรมควบคุมการทำงานของเครื่องจะแบ่งออกได้เป็น 2 ส่วนที่สำคัญคือ โปรแกรมควบคุมในโหมดโปรแกรม และโปรแกรมควบคุมในโหมดทำงาน สำหรับเนื้อหาของบทนี้นอกจากจะกล่าวถึงโปรแกรมควบคุมการทำงานหลัก 2 ส่วนข้างต้นแล้วยังจะกล่าวถึงส่วนอื่นๆ อีกเช่น การอินเตอร์รัพท์ การแสดงผล การจัดเก็บข้อมูล และคำสั่ง การทำงานของคำสั่งฟังก์ชั่นต่าง ๆ

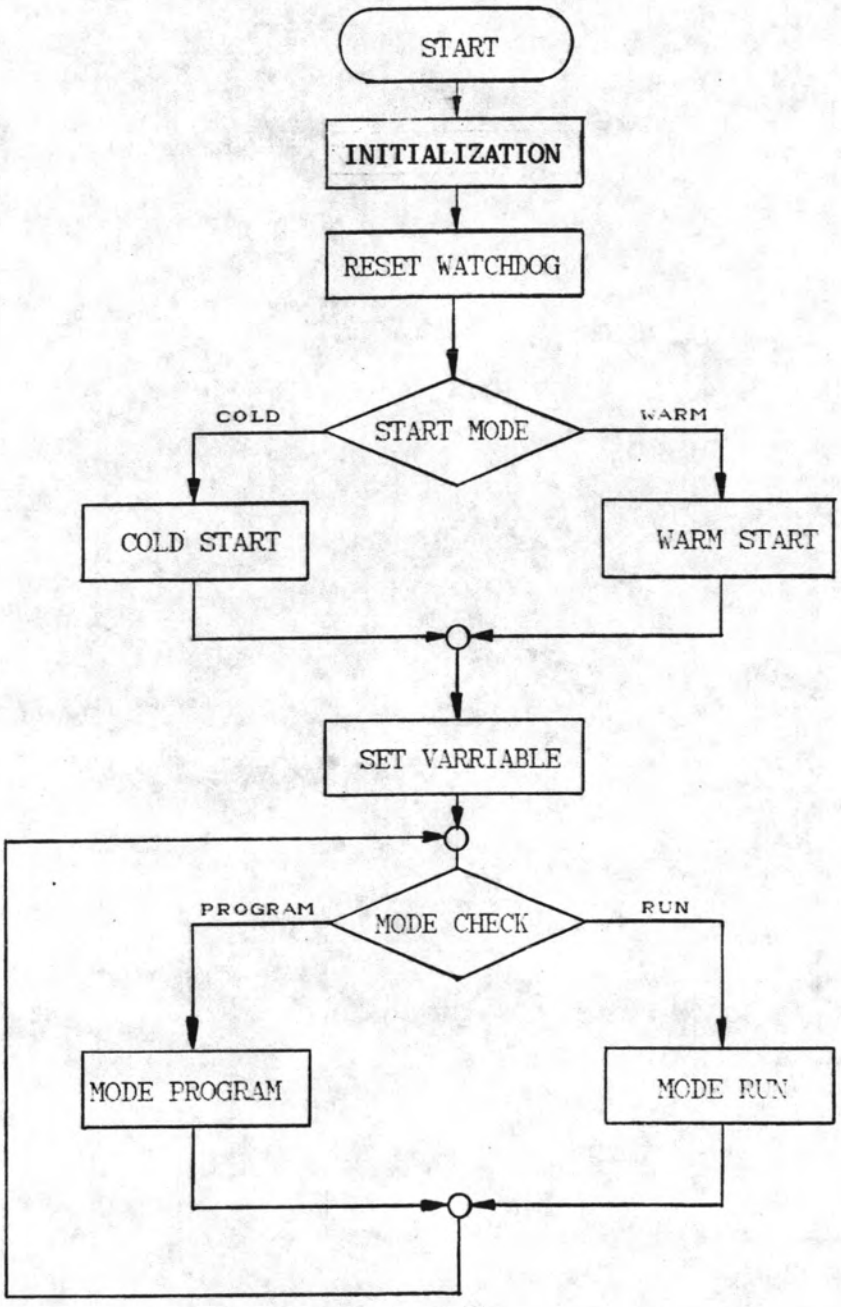
4.1 การทำงานของโปรแกรมควบคุม

ลักษณะการทำงานของโปรแกรมควบคุมเครื่องสามารถแสดงด้วยรูปที่ 4.1 โดยเมื่อเริ่มต้นทำงานจะมีการกำหนดค่าเริ่มต้นต่างๆ ของตัวแปร และตั้งโปรแกรมของอุปกรณ์ต่างๆ มีการรีเซ็ทวงจรวอชท์ดอก แล้วจึงตรวจสอบว่าการเริ่มต้นทำงานเป็น Cold start* หรือ Warm start* ลำดับต่อมาจะเป็นการตรวจสอบว่าเป็นการทำงานในโหมดโปรแกรม (Program mode) หรือโหมดการทำงาน (Run mode) ในขณะที่มีการทำงานในโหมดต่าง ๆ จบหนึ่งครั้งก็จะมีการตรวจสอบดูว่ามีการเปลี่ยนโหมดการทำงานหรือไม่ สำหรับรายละเอียดในการทำงานของโปรแกรมส่วนต่าง ๆ ที่สำคัญ จะอธิบายเป็นส่วน ๆ ต่อไป

4.1.1 การเริ่มต้นทำงาน

การเริ่มต้นทำงานของโปรแกรมควบคุมเครื่องจะเริ่มที่ตำแหน่งแอดเดรส 0000 โดย

* จะอธิบายในหัวข้อ 4.1.1



รูปที่ 4.1 แสดงโฟล์วชาร์ทหลักในการทำงานของเครื่อง

มีการเริ่มต้นทำงานได้ 2 วิธีคือ

1. การเริ่มต้นทำงานด้วยการเปิดเครื่อง หรือเรียกว่า Cold start

2. การเริ่มต้นทำงานจากคำสั่งหรือสัญญาณภายในเครื่อง หรือเรียกว่า Warm start

การแยกการเริ่มต้นทำงานของโปรแกรมออกเป็น 2 แบบนี้ เพื่อต้องการให้เครื่อง PC สามารถทำงานต่อเนื่องโดยอัตโนมัติต่อไปได้ เมื่อมีสัญญาณรบกวนจากภายนอกมาทำให้เครื่อง PC ทำงานผิดพลาดจนถูกรีเซ็ทโดยวงจรวอชท์ดอกไทเมอร์

เนื่องจากการรีเซ็ทโดยวงจร Power on reset หรือโดยวงจร Watchdog timer และการกระโดดมาทำงานที่แอดเดรส 0000 นั้น ซีพียูไม่สามารถรู้ได้ว่าเป็นการเริ่มต้นทำงานโดยวิธีใด ดังนั้นการตรวจสอบว่าเป็น Cold start หรือ Warm start จะใช้วิธีการอ่านค่าพารามิเตอร์ของอุปกรณ์บางตัวในวงจร ถ้าเป็น Cold start หรือเริ่มเปิดเครื่องค่าพารามิเตอร์นี้จะมีค่าไม่แน่นอนเพราะยังไม่มีการโปรแกรมลงไป แต่ถ้า Warm start จะมีค่าตรงกับที่โปรแกรมไว้ ซึ่งทำให้สามารถแยกสถานะทั้งสองได้

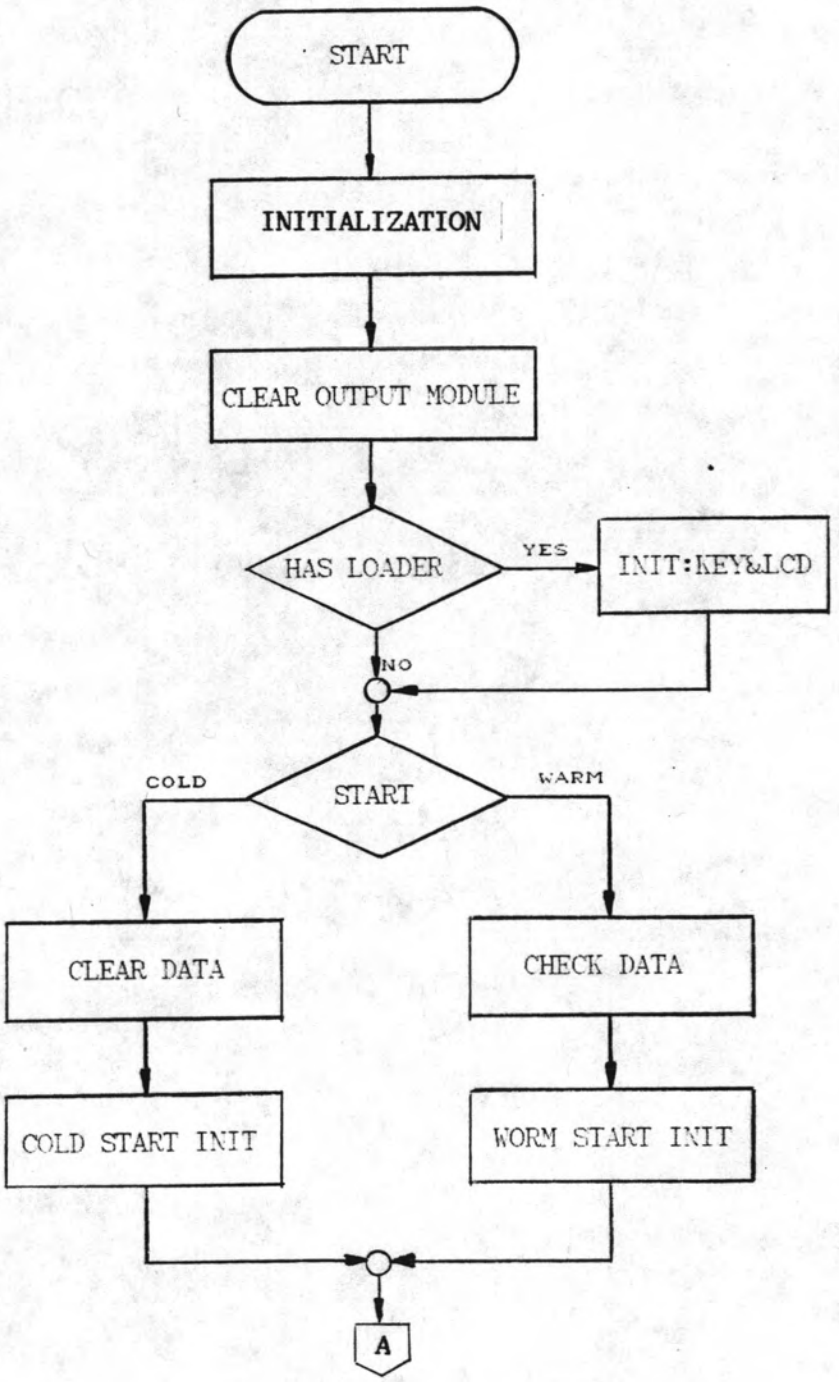
การทำงานของ Warm start นี้จะช่วยให้เสถียรภาพของเครื่องดีขึ้นคือ เมื่อเครื่อง PC กำลังทำงานในโหมดทำงานอยู่ และมีสัญญาณรบกวนจากภายนอกเข้ามาทำให้ซีพียูทำงานผิดพลาด และไม่อยู่ในโปรแกรมที่ให้ทำงาน ซึ่งเมื่อเวลาผ่านไปวงจรวอชท์ดอกไทเมอร์จะส่งสัญญาณรีเซ็ทไปยังซีพียู ซีพียูจะเริ่มต้นทำงานใหม่และตรวจสอบพบว่าเป็น Warm start ก็จะไม่ลบข้อมูลต่าง ๆ ของเดิมทิ้งโดยจะทำงานต่อเนื่องไป ทำให้กระบวนการที่เครื่อง PC ควบคุมอยู่สามารถทำงานต่อเนื่องไปได้โดยไม่หยุดชะงัก

การทำงานของโปรแกรมควบคุมตอนเริ่มต้น ดังแสดงในรูปที่ 4.2 ประกอบด้วยบล๊อกต่าง ๆ ซึ่งมีหน้าที่ดังนี้คือ

INITIALIZATION — เป็นโปรแกรมกำหนดค่าเริ่มต้น ในการทำงานของเครื่อง PC คือ

- เซ็ทอินเตอร์รัพท์โหมด และสถานะการอินเตอร์รัพท์
- เซ็ทค่าสแตก
- กำหนดค่าเริ่มต้นให้ตัวแปรต่าง ๆ ในการทำงาน
- ตรวจสอบโมดูลต่าง ๆ ที่ต่อในระบบ และสร้างตารางเก็บสถานะของโมดูลต่าง ๆ ไว้

Clear output module — เนื่องจากเมื่อเริ่มเปิดเครื่องสถานะของเอาท์พุทพอร์ทของโมดูลต่าง ๆ จะมีค่าไม่แน่นอน จำเป็นต้องเคลียร์สถานะให้เป็น OFF ทั้งหมด



รูปที่ 4.2 แสดงโฟล์วชาร์ทการทำงานของเครื่อง PC

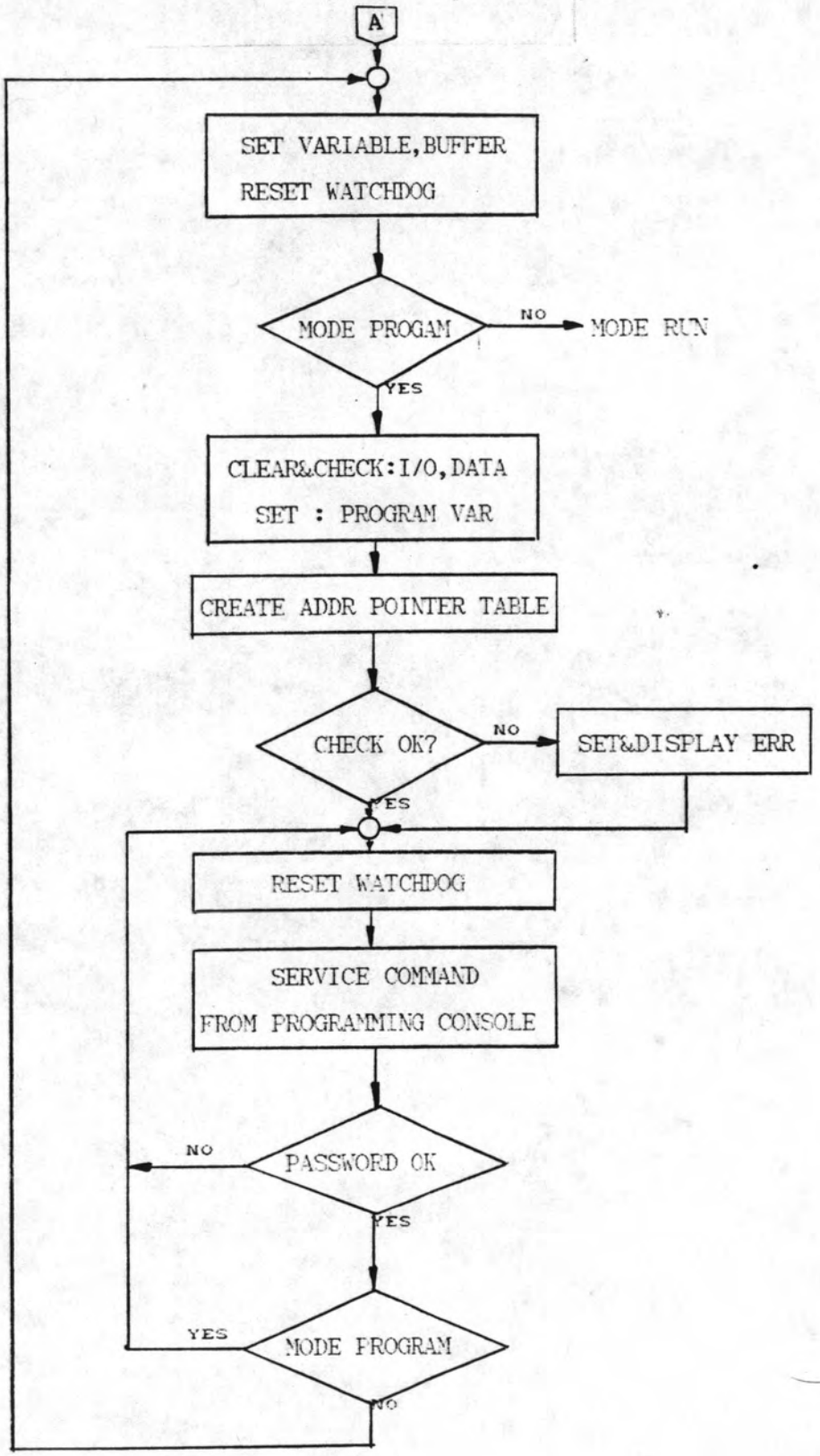
Keyboard & LCD Initialization	เป็นการโปรแกรมการทำงานของวงจรคีย์บอร์ด (ไอซี 8279) และการเซ็ทโหมดการทำงานต่างๆ ของวงจร LCD ที่ใช้แสดงผล กรณีที่ตัวป้อนโปรแกรมแบบมือไม่ได้ต่ออยู่กับเครื่อง PC จะมีการเซ็ทค่าตัวแปรเพื่อป้องกันโปรแกรมควบคุมการทำงาน ไม่ให้เสียเวลาในการอ่านคีย์บอร์ด และแสดงผล
Clear data	เป็นโปรแกรมเคลียร์ข้อมูลอินพุท/เอาท์พุทต่าง ๆ ในหน่วยความจำ และเคลียร์ข้อมูลของตัวตั้งเวลา (Timer, ข้อมูลสำหรับคำสั่งจัดการข้อมูล)
Check data	เป็นโปรแกรมจะตรวจสอบข้อมูลต่างๆ ในหน่วยความว่าถูกต้องหรือไม่ ถ้ามีค่าไม่ถูกต้องก็จะเคลียร์ข้อมูลที่ตำแหน่งนั้น ถ้าข้อมูลมีรูปแบบถูกต้องก็ยังคงสถานะเดิมไว้
Cold start initialization	เป็นการโปรแกรมฟังก์ชั่นการทำงานของอุปกรณ์ต่างๆ เช่น Z80-CTC โมดูล LCD
Warm start initialization	เป็นการกำหนดค่าเริ่มต้นให้ตัวแปรต่างๆ

4.1.2 <u>การทำงานโหมดโปรแกรม</u> (Program mode)

โหมดโปรแกรมออกแบบมาให้ใช้งานเกี่ยวกับ การป้อนโปรแกรมขั้นบันไดลงในหน่วยความจำ การแก้ไขคำสั่งหรือโอเปอร์แรนด์ (Operand) ของคำสั่งต่าง ๆ การลบคำสั่ง (Delete) การสอดแทรกคำสั่ง (Insert) การค้นหาตำแหน่งของคำสั่ง (Search) การดูสถานะหรือค่าของข้อมูลต่างๆ (Monitor) และการเซ็ทหรือรีเซ็ทสถานะของข้อมูลรีเลย์ และการทำงานของคำสั่งช่วยของเครื่อง เช่น การลบโปรแกรม การลบข้อมูล การป้อนรหัส Password

การทำงานของโปรแกรมควบคุมในโหมดโปรแกรมแสดงเป็นโฟล์วชาร์ทในรูปที่ 4.3 ซึ่งมีขั้นตอนการทำงานดังนี้คือ

1. รีเซ็ทวงจรวอทช์ดอกไทเมอร์ และกำหนดค่าเริ่มต้นให้ตัวแปรที่ใช้ร่วมในโหมดโปรแกรมและโหมดทำงาน



รูปที่ 4.3 แสดงโฟล์วชาร์ทการทำงานโหมดโปรแกรม

2. ตรวจสอบว่าเป็นการทำงานโหมดใด ถ้าเป็นโหมดทำงานก็กระโดดไปที่โหมดทำงาน ถ้าเป็นโหมดโปรแกรมก็ทำงานในขั้นต่อไป

3. ลบค่าของข้อมูลรีเลย์ และข้อมูลแบบ Non Retentive ในเครื่อง PC ทั้งหมด ตรวจสอบข้อมูลแบบ Retentive ถ้ามีรูปแบบที่ผิดก็ลบข้อมูลตัวนั้นทิ้ง และกำหนดค่าให้ตัวแปรที่ต้องใช้งานในโหมดโปรแกรม

4. เป็นการสร้างตารางชี้ตำแหน่งโปรแกรมขั้นบันไดของผู้ใช้ เพื่อใช้ในการเลื่อนไปยังตำแหน่งต่าง ๆ ของโปรแกรมขั้นบันได เพราะโปรแกรมขั้นบันไดแต่ละคำสั่งมีความยาวไปเท่ากัน

5. ตรวจสอบโปรแกรมขั้นบันไดที่เก็บในหน่วยความจำว่ามีที่ผิดพลาดหรือเปล่า ถ้ามีก็จะเซ็ทค่าตัวแปรไว้ และแสดงค่าที่ผิดพลาดบอกให้ผู้ใช้ทราบ

6. รีเซ็ทวงจรวอทช์ดอกไทเมอร์

7. เป็นการตรวจสอบการกดคีย์บอร์ดของผู้ใช้ ถ้ามีการกดคีย์บอร์ดก็จะไปทำงานที่โปรแกรมจัดการคีย์บอร์ด ซึ่งใช้อัลกอริธึมแบบ Keyboard Parser โปรแกรมการทำงานส่วนนี้จะคล้ายกับโปรแกรมในส่วน Service peripheral device ในโหมดทำงาน

8. หลังจากทำงานตามการกดคีย์แล้วจะตรวจสอบสถานะของ Password ถ้ายังไม่ป้อน Password จะไม่ยอมเปลี่ยนโหมด โดยจะกลับไปทำงานในขั้นตอนที่ 6 ซ้ำใหม่

9. ตรวจสอบว่ามีการเปลี่ยนโหมดหรือไม่ ถ้าไม่เปลี่ยนจะไปทำงานในขั้นตอนที่ 6 ซ้ำใหม่ ถ้ามีการเปลี่ยนโหมดเป็นโหมดทำงาน ก็จะข้ามไปยังขั้นตอนที่ 1 อีกครั้ง

4.1.3 <u>โหมดทำงาน</u> (Run mode)

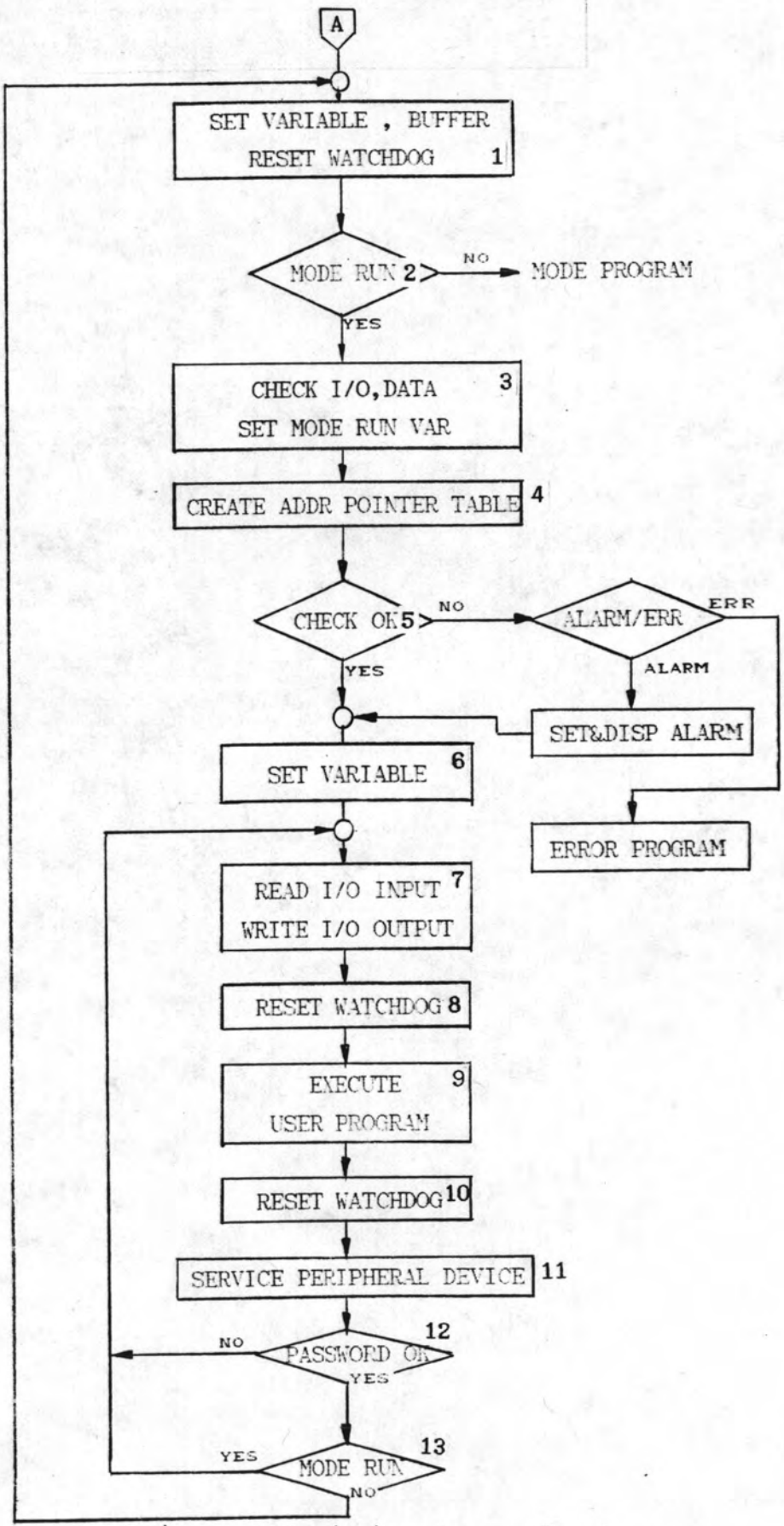
การทำงานของโหมดทำงานที่สำคัญคือ การทำงานตามโปรแกรมขั้นบันไดของผู้ใช้ที่เก็บไว้ในหน่วยความจำ ซึ่งต้องการให้ทำงานเร็วที่สุด สำหรับการแปลงโปรแกรมขั้นบันไดของเครื่องนี้จะใช้วิธี Compiler[8] ร่วมกับวิธี Call[8] ซึ่งได้กล่าวมาแล้วในบทที่ 2 สำหรับการอ่าน I/O อินพุท และการเขียน I/O เอาท์พุท ในโหมดทำงานนี้จะกล่าวถึงอีกครั้งในหัวข้อถัดไป การทำงานในโหมดทำงานมีขั้นตอนดังนี้

1. รีเซ็ทวงจรวอทช์ดอกไทเมอร์ และกำหนดค่าให้กับตัวแปรต่าง ๆ ที่ใช้ร่วมกันทั้งโหมดโปรแกรมและโหมดทำงาน

2. ตรวจสอบโหมดการทำงาน และไปทำงานตามโหมดนั้น ๆ

3. ตรวจสอบข้อมูลของ I/O รีเลย์ต่าง ๆ ถ้ามีรูปแบบที่ผิดพลาดจะเคลียร์ข้อมูลตำแหน่งนั้น และเซ็ทค่าเริ่มต้นสำหรับตัวแปรของฟังก์ชั่น TIM, TIMH, SFT, CNT, DIFU, DIFD

4. สร้างตารางชี้ตำแหน่งโปรแกรมขั้นบันไดที่เก็บในหน่วยความจำ เพื่อใช้ในการ



รูปที่ 4.4 แสดงโฟลว์ชาร์ทการทำงานของโหมดทำงาน

เลื่อนไปยังตำแหน่งต่างๆ ของคำสั่ง ในการเรียกดูโปรแกรมและการค้นหาโปรแกรมของผู้ใช้

5. ตรวจสอบความผิดพลาด ซึ่งแบ่งออกได้เป็น 2 แบบคือ ความผิดพลาดเล็กน้อยสามารถทำงานได้เรียกว่า ALARM เช่น สถานะข้อมูลที่เก็บไว้ผิดพลาด เป็นต้น และความผิดพลาดที่ระบบไม่สามารถทำงานได้เรียกว่า ERROR เช่น ผู้ใช้ลืมโปรแกรมคำสั่งจบโปรแกรม (END) ในโปรแกรมชั้นบันไดที่เขียน เป็นต้น

กรณีที่เกิด ERROR ขึ้นระบบจะไม่ทำงานตามโปรแกรมชั้นบันได ผู้ใช้ต้องเปลี่ยนการทำงานไปโหมดโปรแกรมแล้วแก้ไขโปรแกรมให้ถูกต้องเสียก่อน

6. กำหนดค่าเริ่มต้นให้กับตัวแปรหรือพารามิเตอร์ต่าง ๆ ที่ใช้ในโหมดทำงาน เช่น ข้อมูลเริ่มต้น ตำแหน่งเริ่มทำงาน

7. อ่านข้อมูลอินพุทพอร์ทของโมดูลต่าง ๆ มาเก็บในหน่วยความจำ และเขียนข้อมูลจากหน่วยความจำไปยังพอร์ทเอาท์พุทของโมดูลต่าง ๆ

8. รีเซ็ทวงจรวอทช์ดอกไทเมอร์

9. ทำงานตามโปรแกรมชั้นบันไดที่เก็บในหน่วยความจำ โดยเริ่มทำงานจากคำสั่งแรกจนถึงคำสั่งจบโปรแกรม (END)

10. รีเซ็ทวงจรวอทช์ดอกไทเมอร์

11. ตรวจสอบสถานะของอุปกรณ์ที่มาต่อว่าจะต้องทำงานอะไรให้บ้างหรือไม่ เช่น ถ้ามีการกดคีย์บอร์ด ก็จะทำงานตามโปรแกรมควบคุมคีย์บอร์ด ซึ่งมีลักษณะเหมือนการทำงานในโหมดโปรแกรม

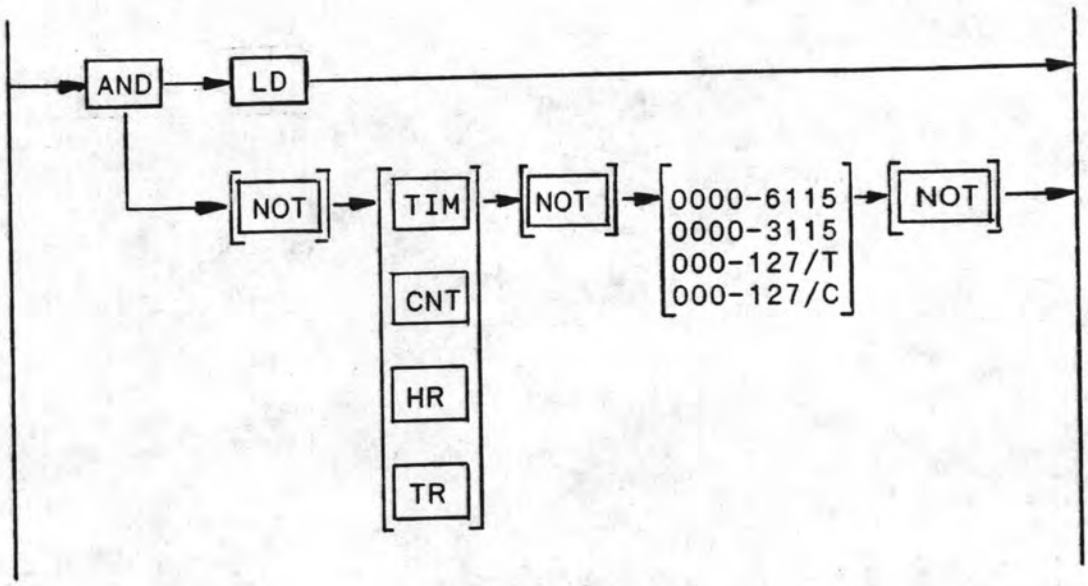
12. ตรวจสอบว่ามีการป้อน (Password) หรือยัง ถ้ายังไม่ป้อนจะกลับไปทำงานซ้ำขั้นตอนที่ 7 ใหม่

13. ตรวจสอบการเปลี่ยนโหมดการทำงาน ถ้าไม่มีการเปลี่ยนจะกลับไปทำงานขั้นตอนที่ 7 ถ้ามีการเปลี่ยนเป็นโหมดโปรแกรมจะไปทำงานขั้นตอนที่ 1

4.1.4 โปรแกรมจัดการคีย์บอร์ด

ลำดับขั้นการโต้ตอบการกดคีย์บอร์ดของผู้ใช้ในเครื่อง PC มีลักษณะที่ยุ่งยากซับซ้อน เนื่องจากมีปุ่มคีย์บอร์ด 42 คีย์ และมีคำสั่งชั้นบันไดต่าง ๆ ที่มีลักษณะแตกต่างกันเป็นจำนวนมาก การเขียนโปรแกรมควบคุมการทำงานในส่วนนี้จะใช้อัลกอริธึมที่เรียกว่า Keyboard Parser [9] ซึ่งมีลักษณะการทำงานแบบลำดับ โดยมีการเก็บสถานะลำดับปัจจุบันไว้

ตัวอย่างแสดงลักษณะการกดคีย์ "AND" เพื่อป้อนโปรแกรมซึ่งสามารถกดคีย์บอร์ดลำดับต่อไปหลายแบบ ดังแสดงในรูปที่ 4.5



รูปที่ 4.5 แสดงลำดับขั้นการป้อนคำสั่ง AND

เพื่อความสะดวกในการทำงานของโปรแกรมควบคุมการกดคีย์บอร์ด เราจะมีการแปลงรหัสของคีย์บอร์ดที่อ่านได้จากการกดคีย์ โดยคีย์บอร์ดที่กดมา 1 ครั้งจะแทนด้วยพารามิเตอร์ 2 ตัวคือ FNKY และ NUMB

FNKY หมายถึง รหัสฟังก์ชั่นในการทำงานของคีย์นั้น

NUMB หมายถึง ค่าข้อมูลของรหัส (FNKY) คีย์นั้น ๆ

ลักษณะที่เห็นได้ชัดคือ คีย์ตัวเลข 0, 1, 2, 3,..., 9 จะแทนด้วย FNKY ค่าเดียวกัน แต่มี NUMB ต่างกัน เพราะหน้าที่การทำงานของคีย์ตัวเลข 0-9 จะเหมือนกัน แต่มีค่าข้อมูลต่างกันค่า FNKY และ NUMB ของคีย์ต่าง ๆ แสดงในตารางที่ 4.1

หน้าที่ของโปรแกรมควบคุมการกดคีย์บอร์ดที่สำคัญ แบ่งออกเป็น 3 ส่วนคือ

1. อ่านค่าคีย์บอร์ดที่มีการกด และแปลงเป็นรหัส FNKY และ NUMB
2. ตรวจสอบค่า FNKY ของคีย์ที่กด เทียบกับสถานะ (State) เดิมว่าควรเปลี่ยนสถานะเดิมเป็นสถานะใหม่อะไร
3. ทำโปรแกรมตอบสนองการกดคีย์นั้น

การทำงานของโปรแกรมควบคุมการกดคีย์บอร์ด มีตัวแปรที่เกี่ยวข้องในการทำงานดังนี้

FNKY เป็นตัวแปรที่เก็บรหัสการกดคีย์บอร์ด

NUMB เป็นตัวแปรเก็บค่าตัวเลขของการกดคีย์บอร์ด

PREST เป็นตัวแปรที่ใช้เก็บสถานะปัจจุบัน (Present state) ซึ่งจะใช้อ้างอิงในการเปรียบเทียบเพื่อหาสถานะต่อไป

PARSER STATE TABLE หมายถึงตารางที่ใช้ในการเปรียบเทียบเพื่อหาสถานะต่อไป ตารางนี้ประกอบด้วย FNKYT, NEXST และ ARNO

FNKYT เป็นรหัสของคีย์ที่เก็บไว้ในตาราง PARSER STATE ซึ่งจะเก็บไว้สำหรับเปรียบเทียบกับตัวแปร FNKY เพื่อหาค่า NEXST และ ARNO

NEXST เป็นค่าของสถานะที่เก็บไว้ในตาราง PARSER STATE เพื่อที่จะนำไปเปลี่ยนค่าของสถานะปัจจุบัน

ARNO ACTION ROUNTINE NUMBER เก็บไว้ใน PARSER STATE TABLE ใช้เป็นตัวชี้ตำแหน่งโปรแกรมตอบสนองการกดคีย์

โฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมคีย์บอร์ดแสดงในรูปที่ 4.6 ซึ่งมีลักษณะการทำงานดังนี้

1. ตรวจสอบว่ามีการกดคีย์บอร์ด ถ้ามีจะนำรหัสคีย์บอร์ดไปแปลงเป็นค่า FNKY และ NUMB กรณีที่ FNKY ไม่เท่ากับ OFFH ก็จะทำขั้นต่อไป

2. ใช้ค่าสถานะปัจจุบัน (PREST) เป็นตัวชี้หาตำแหน่งเริ่มต้นของตาราง PARSER STATE TABLE แล้วนำค่า FNKY ไปเปรียบเทียบกับค่า FNKYT ในตารางจนกว่าจะพบค่าที่เท่ากัน หรือจนกว่าจะพบเครื่องหมาย "*"

3. นำค่า NEXST ที่ได้จากตารางไปแทนค่า PREST

4. ตรวจสอบดูค่าของ PREST ถ้าเท่ากับ Initial state ก็กลับไปทำขั้นตอนที่ 2 ใหม่

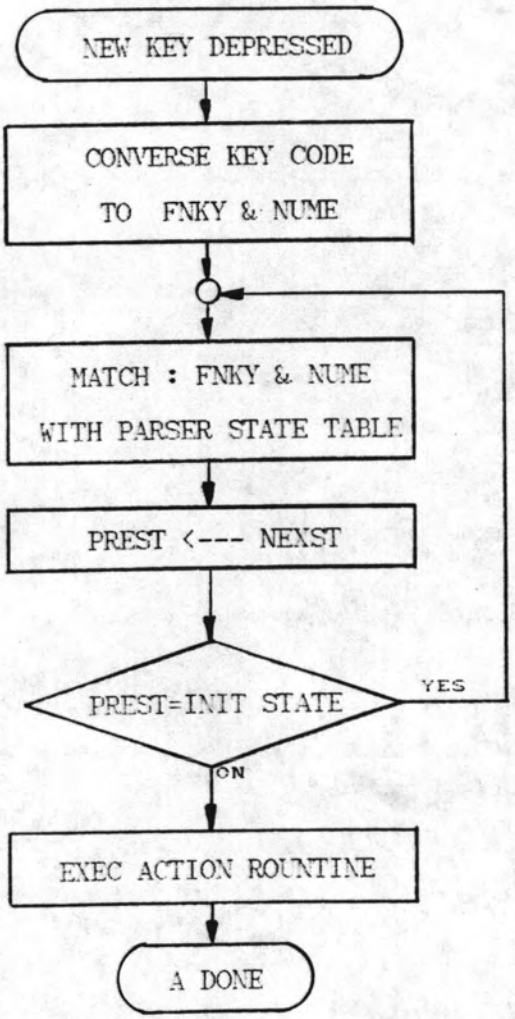
5. ทำโปรแกรมตอบสนองการกดคีย์บอร์ดตามค่า ARNO ที่ได้จากตาราง

KEY LABEL	KEY CODE	FNKY	NUMB
0	06	01	0
1	05	01	1
2	0D	01	2
3	15	01	3
4	04	01	4
5	0C	01	5
6	14	01	6
7	03	01	7
8	0B	01	8
9	13	01	9
LD	02	02	0
AND	01	03	0
OR	09	04	0
OUT	0A	05	0
TIM	12	06	0
CNT	11	07	0
NOT	10	08	0
HR	29	09	0
TR	19	0A	0
DM	1A	0B	0
*	22	0C	0
#	2A	0D	0
SHIFT	28	0E	0
FUN	00	0F	0
CLR	16	10	0
WRITE	26	11	0

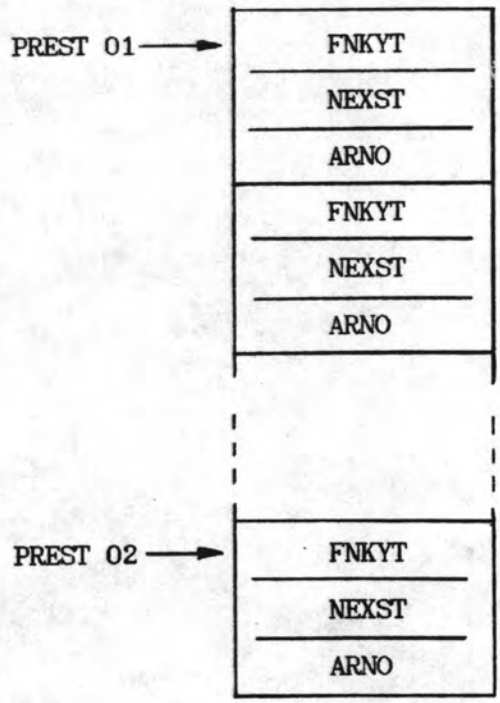
ตารางที่ 4.1 แสดงการแปลงรหัสของคีย์บอร์ด

KEY LABEL	KEY CODE	FNKY	NUMB
DEL	24	12	0
INS	25	13	0
MONTR	2C	14	0
SRCH	2B	15	0
CHG	23	16	0
SET	1C	17	0
RESET	1D	18	0
⇧	2D	19	0
⇩	2E	1A	0
SFT	08	1B	0
A	06	1C	0A
B	05	1C	0B
C	0D	1C	0C
D	15	1C	0D
E	04	1C	0E
F	0C	1C	0F
CH	2A	1D	0
CONT	22	1E	0
---	---	FF	0

ตารางที่ 4.1 (ต่อ) แสดงการแปลงรหัสของคีย์บอร์ด



รูปที่ 4.6 แสดงโฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมคีย์บอร์ด



รูปที่ 4.7 แสดงตำแหน่งของการจัดเก็บของ PARSER STATE TABLE

4.1.5 การอินเตอร์รัพท์ (Interrupts)

การอินเตอร์รัพท์ของเครื่อง PC นี้จะใช้สัญญาณ INT จากไอซี Z80-CTC ซึ่งโปรแกรมไว้ให้สร้างฐานเวลา 0.005 วินาที โหมดของการอินเตอร์รัพท์ที่ใช้เป็นโหมด 1 เมื่อซีพียูได้รับสัญญาณ INT จะกระโดดไปทำงานที่แอดเดรส 0038H

หน้าที่ที่สำคัญของโปรแกรมบริการอินเตอร์รัพท์คือ

1. เซ็ทพารามิเตอร์สำหรับฐานเวลา 0.01 และ 0.1 วินาที เพื่อใช้ในคำสั่ง TIMH และ TIM
2. สร้างสัญญาณพัสช์สำหรับรีเลย์ช่วยต่อไปนี้

รีเลย์เบอร์	6300	สัญญาณ 0.1 วินาที
	6301	สัญญาณ 0.2 วินาที
	6302	สัญญาณ 1 วินาที

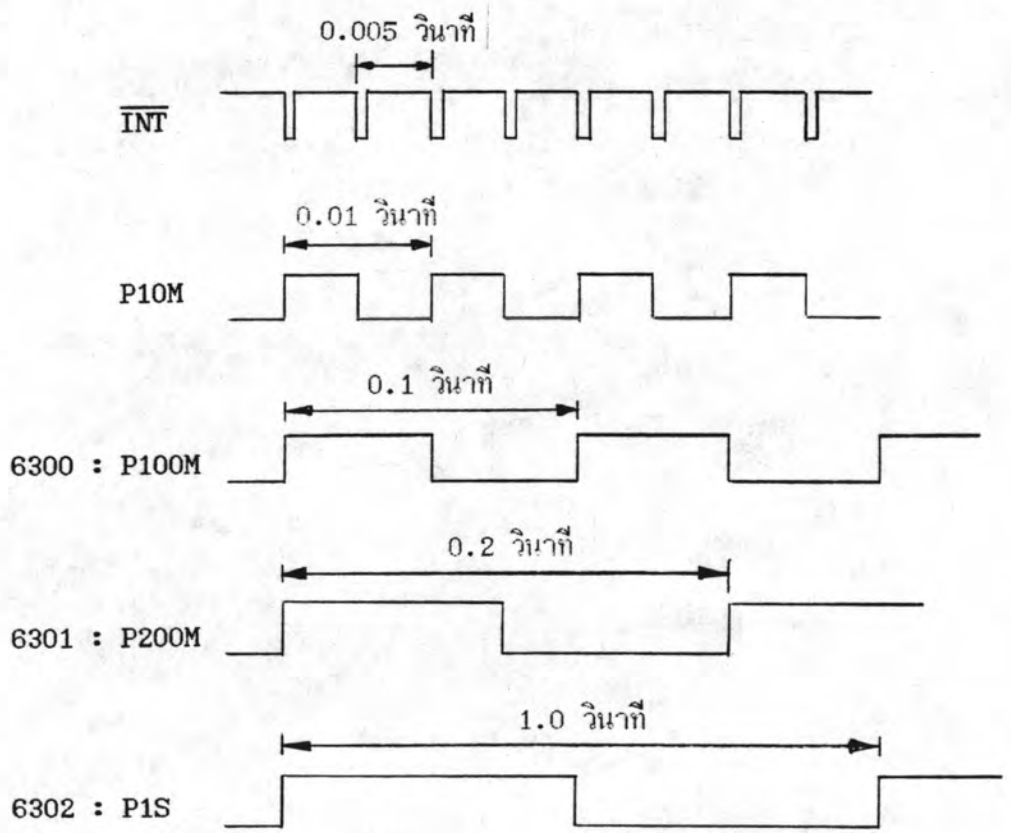
3. แสดงผลของข้อมูลที่ LCD
4. แสดงผลของข้อมูล สำหรับการแสดงสถานะ (MONITOR)

การอินเตอร์รัพท์จะเกิดขึ้นทุกๆ 0.005 วินาที ดังนั้นโปรแกรมการอินเตอร์รัพท์ต้องออกแบบให้กระทัดรัด เพื่อให้การทำงานรวดเร็ว ซึ่งรวมถึงตำแหน่งของตัวแปรที่ใช้ในโปรแกรม

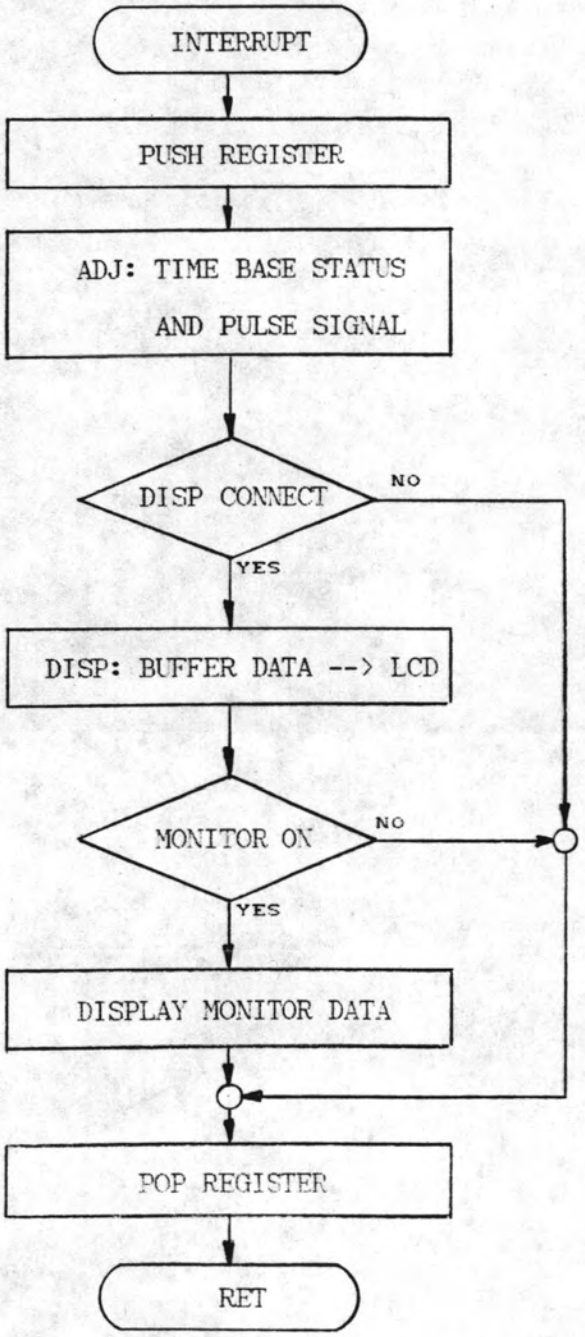
ควรมีการวางตำแหน่งที่เหมาะสมเพื่อให้โปรแกรมทำงานได้เร็ว

การทำงานของฐานเวลา 0.01 วินาทีและ 0.1 วินาที สำหรับคำสั่ง TIMH (High speed timer) และคำสั่ง TIM (Timer) จะทำงานโดยโปรแกรมอินเตอร์รัพท์จะเซ็ทสถานะ F10M ให้เป็น 01 ทุก ๆ 0.01 วินาทีและเซ็ทสถานะ F100M ให้เป็น 01 ทุก ๆ 0.1 วินาที เมื่อซีพียูทำงานตามโปรแกรมขั้นบันได(Ladder) และพบคำสั่ง TIMH หรือ TIM ก็จะตรวจสอบสถานะ F10M/F100M ถ้ามีค่าสถานะเป็น 01 ก็จะทำตามคำสั่งนั้นคือ ลดค่าจำนวนเวลาลง 1 และทำงานตามโปรแกรมขั้นบันไดต่อไปเรื่อยๆ เมื่อทำงานครบหนึ่งรอบการทำงาน (Scantime) เครื่องก็จะเซ็ทสถานะของ F10M และ F100M ให้เป็น 00

การแสดงผลข้อมูลที่ LCD จะทำทุก ๆ 0.01 วินาที โดยจะแสดงผลเพียง 1 ตัวอักษร ดังนั้นถ้าแสดงผลข้อมูลหลาย ๆ ตัว จะต้องคอยสัญญาณอินเตอร์รัพท์หลายครั้งจึงแสดงผลหมดทุกตัวอักษร สำหรับการทำงานในการแสดงข้อมูลของการดูสถานะ (Monitor) จะมีลักษณะเหมือนที่กล่าวมาเพียงแต่จะแสดงผลทุก ๆ 0.05 วินาที



รูปที่ 4.8 แสดงลักษณะของสัญญาณฐานเวลาต่าง ๆ

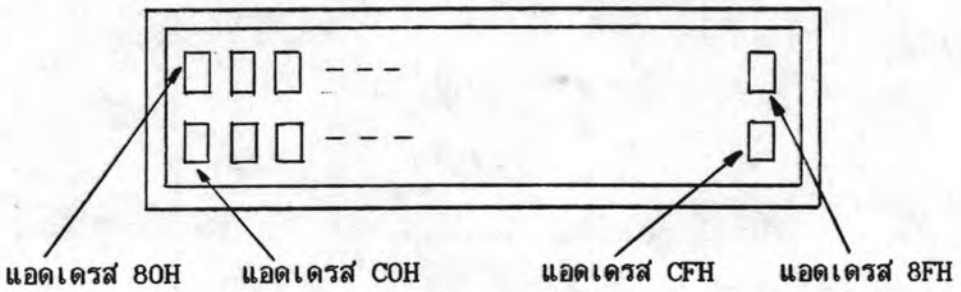


รูปที่ 4.9 แสดงโฟลว์ชาร์ทการทำงานของโปรแกรมอินเตอร์รัปท์

4.1.6 การแสดงผลของ LCD

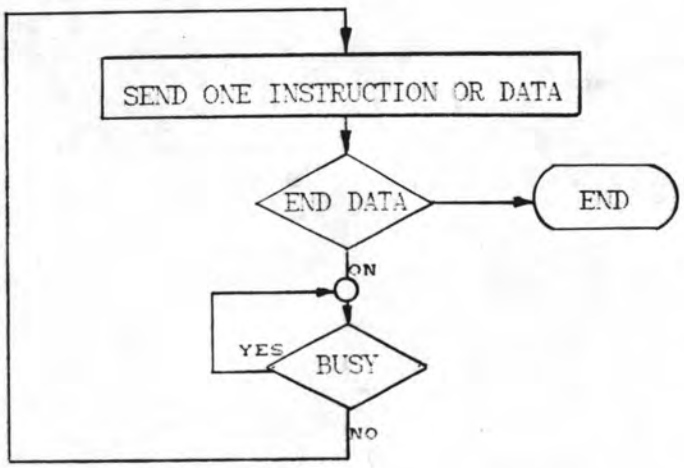
การแสดงผลของเครื่อง PC ที่ออกแบบนี้ใช้จอ LCD ขนาด 16 ตัวอักษร 2 แถว โดยตัวอักษรที่แสดงเป็นด็อทซ์เมทริกขนาด 5 x 7 รหัสในการแสดงข้อมูลจะคล้ายรหัสแอสกี้ แต่มีรหัสคำสั่งควบคุมต่างหาก สำหรับการเซ็ทโหมดการทำงานของ LCD เราเซ็ทไว้แล้วในส่วนเริ่มต้นการทำงานของโปรแกรมควบคุม (Initialization) คำสั่งควบคุม LCD ที่ใช้ในการแสดงผลที่สำคัญได้แก่ คำสั่งในการเคลียร์การแสดงผลทั้งหมด และคำสั่งในการกำหนดตำแหน่งสำหรับการแสดงผล การส่งสัญญาณไปยัง LCD จะแบ่งออกเป็น 2 ชนิดคือ

1. การส่งสัญญาณควบคุมการแสดงผล ใช้พอร์ทหมายเลข 80H
2. การส่งสัญญาณรหัสตัวอักษรที่แสดงผล ใช้พอร์ทหมายเลข 81H



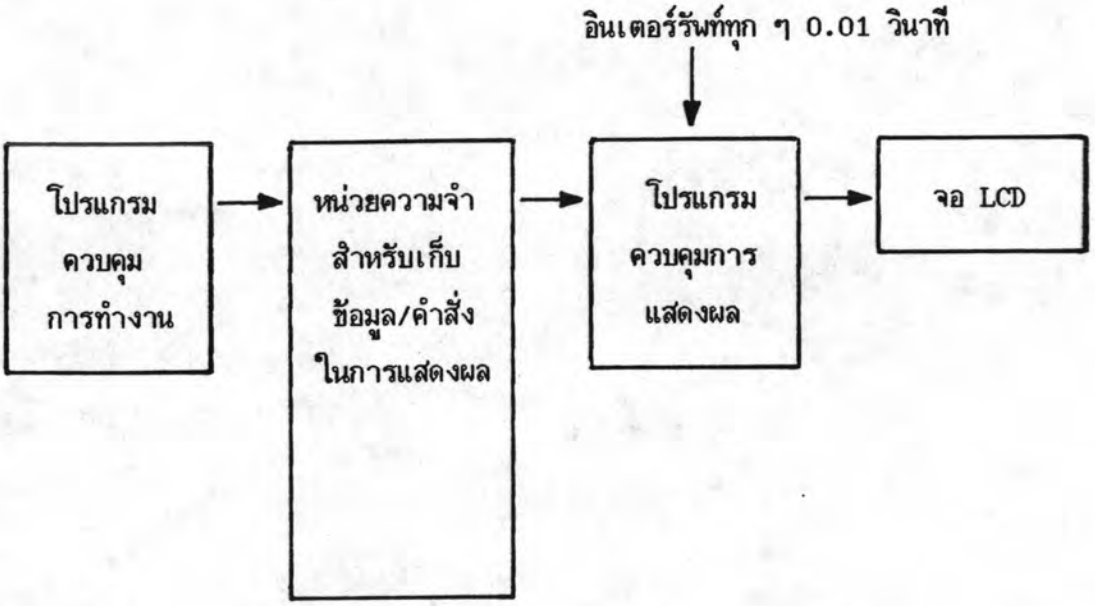
รูปที่ 4.10 แสดงตำแหน่งในการแสดงผลของ LCD

เนื่องจากตัวแสดงผล LCD ไม่สามารถรับข้อมูลในการแสดงผล หรือคำสั่งติดต่อกันได้หลายตัวในการส่งข้อมูลไปให้ตัวแสดงผล LCD 1 ครั้ง ซีพียูต้องคอยตรวจสอบว่าตัว LCD พร้อมที่จะรับข้อมูลตัวต่อไปหรือยัง ซึ่งข้อมูลและคำสั่งแต่ละชนิดมีเวลาในการทำงาน (Delay time) ไม่เท่ากัน เช่น คำสั่งเคลียร์จอ LCD ใช้เวลาในการทำงานประมาณ 1.64 mSec การแสดงข้อมูลตัวอักษรใช้เวลาในการทำงานประมาณ 40 µSec



รูปที่ 4.11 แสดงโฟลว์ชาร์ทการแสดงผลทางจอ LCD แบบปกติ

ถ้าใช้การแสดงผลแบบปกตินี้คือ ให้ซีพียูคอยตรวจสอบสัญญาณ BUSY ของ LCD นี้ จะทำให้ซีพียูเสียเวลาในการคอยมาก ซึ่งจะทำให้เครื่อง PC ทำงานได้ช้าลง วิธีการแสดงผลแบบนี้จึงไม่เหมาะสม ดังนั้นในการแสดงผลของเครื่อง PC นี้ จึงใช้สัญญาณอินเตอร์รัพท์เป็นตัวช่วยในการแสดงผล โดยออกแบบให้มีการส่งคำสั่งหรือข้อมูลไปยังตัว LCD ทุก ๆ 0.01 วินาที เมื่อมีการแสดงผล



รูปที่ 4.12 แสดงลักษณะการทำงานของโปรแกรมควบคุมการแสดงผล

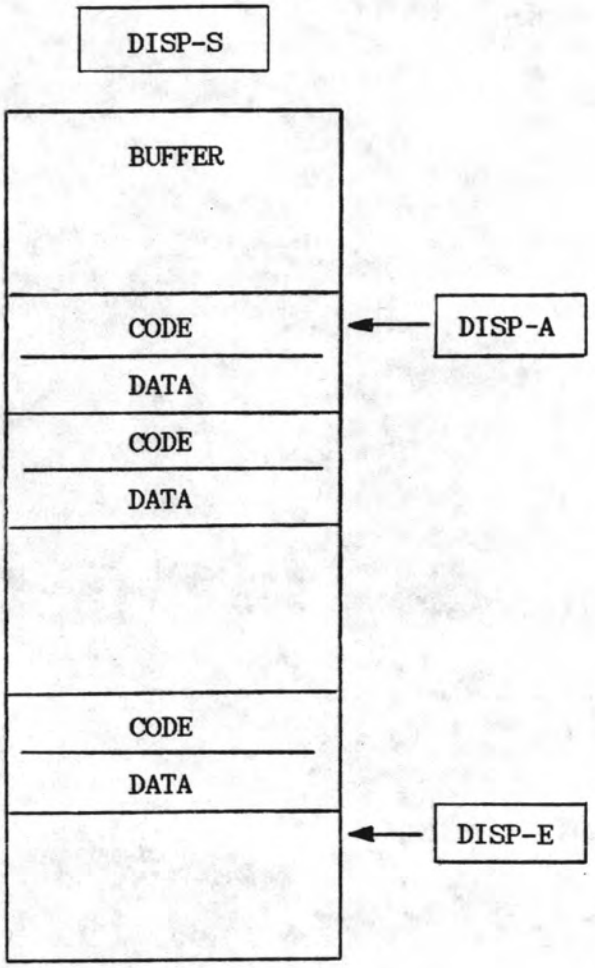
การทำงานของการแสดงผลนี้จะมีการสร้างพื้นที่สำหรับเก็บข้อมูลในการแสดงผล (Display data buffer) และตัวแปรอีก 3 ตัวอีก

DISP-S เป็นตัวเก็บสถานะว่ามีข้อมูลในการแสดงผล หรือไม่
รหัส 01 แทนยังมีข้อมูลในการแสดงผล
รหัส 00 แทนไม่มีข้อมูลในการแสดงผล

DISP-A เก็บตำแหน่งของข้อมูลที่ต้องแสดงผล ตัวแปรนี้จะชี้ตำแหน่งของข้อมูลในบัฟเฟอร์ตัวต่อไปที่ต้องแสดงผล ตัวแปรต้องปรับค่า (Update) ทุกครั้งที่มีการแสดงผล

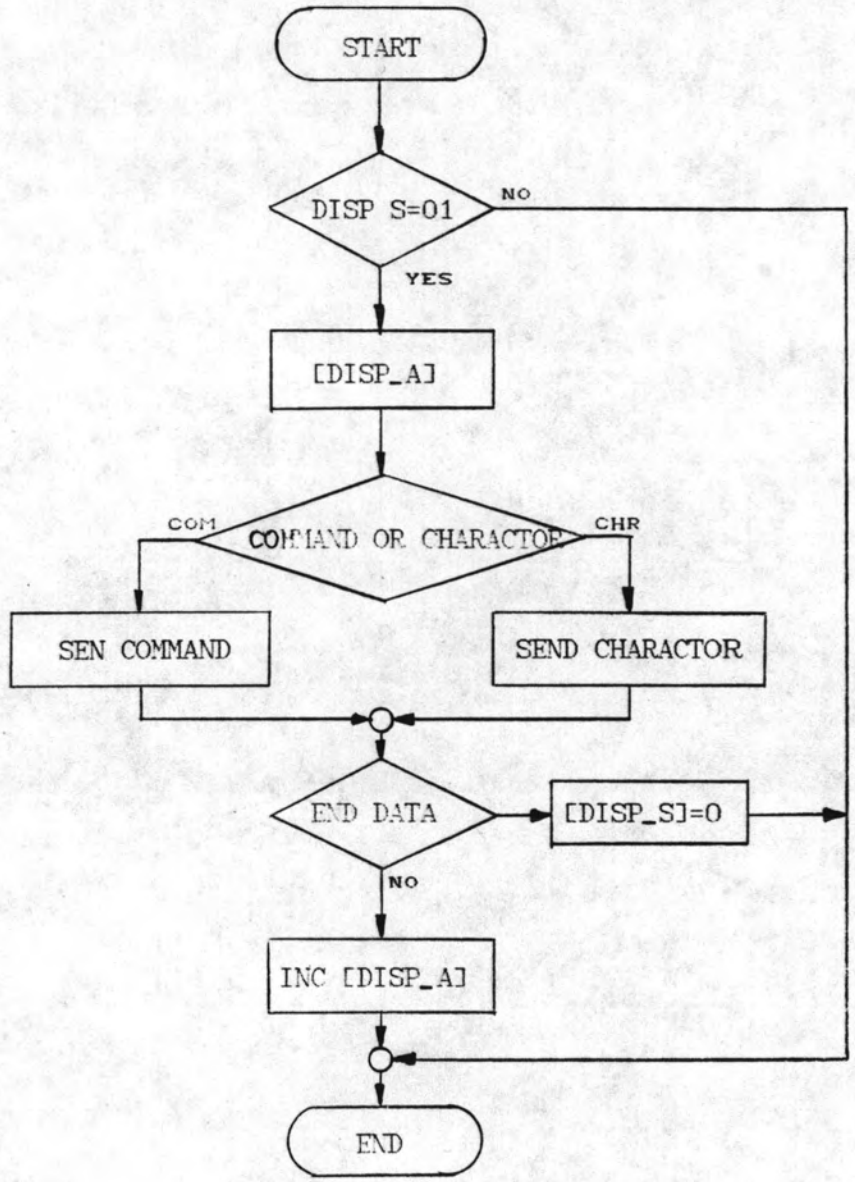
DISP-E เก็บตำแหน่งสุดท้ายของข้อมูลในบัฟเฟอร์ เพื่อใช้ในการเพิ่มข้อมูลที่ต้องแสดงผลลงไป

ข้อมูลที่จะเก็บลงในบัฟเฟอร์ 1 ตัว ประกอบด้วย 2 ส่วนคือ รหัสเพื่อบอกว่าเป็นคำสั่งหรือตัวอักษร และส่วนที่สองคือ ข้อมูล



รูปที่ 4.13 แสดงการจัดเก็บข้อมูลในการแสดงผล

การทำงานของโปรแกรมควบคุมการแสดงผลจะตรวจสอบสถานะ DISP-S ถ้าเท่ากับ 01 ก็จะไปอ่านข้อมูลที่ชี้ตำแหน่งโดย DISP-A แล้วตรวจสอบว่าข้อมูลดังกล่าวเป็นคำสั่งหรือตัวอักษร แล้วส่งข้อมูลไปยัง LCD และตรวจดูว่าหมดข้อมูลหรือยัง ถ้ายังก็เพิ่มค่าของ DISP-A เป็นตำแหน่งตัวถัดไป ถ้าข้อมูลหมดแล้วก็เซ็ทค่า DISP-S เป็น 00



รูปที่ 4.14 แสดงโฟลว์ชาร์ทการทำงานของโปรแกรมควบคุมการแสดงผล

4.1.7 การอ่าน และเขียนอินพุท/เอาท์พุท

การอ่าน และเขียนอินพุท/เอาท์พุทของเครื่อง PC สามารถแบ่งออกได้ 2 วิธี [3] คือวิธี Continuous updating และวิธี Mass input/output copy ทั้ง 2 วิธีมีลักษณะที่แตกต่างกันคือ

1. วิธี Continuous updating

ซีพียูจะอ่าน และเขียนอินพุท/เอาท์พุท เมื่อกำลังทำงานคำสั่งนั้นในโปรแกรมขั้นบันไดโดยตรง ซึ่งถ้าเป็นอินพุทก็จะได้ค่าจริงของข้อมูลที่มี delay time ตามอินพุทพอร์ทนั้น ถ้าเป็นเอาท์พุทซีพียูก็จะเขียนเอาท์พุทออกที่เอาท์พุทพอร์ทโดยตรงการทำงานของวิธีนี้มีลักษณะดังต่อไปนี้

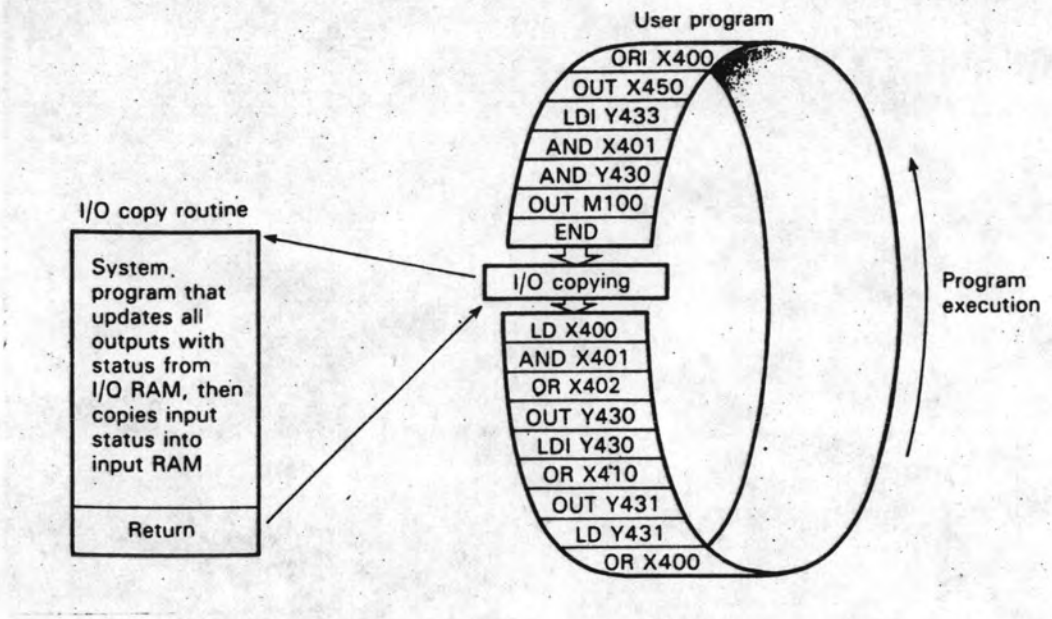
EXECUTE FIRST INSTRUCTION	SCAN I/O	NEXT INSTRUCTION	SCAN I/O	NEXT INSTRUCTION	SCAN I/O

2. วิธี Mass input/output copy

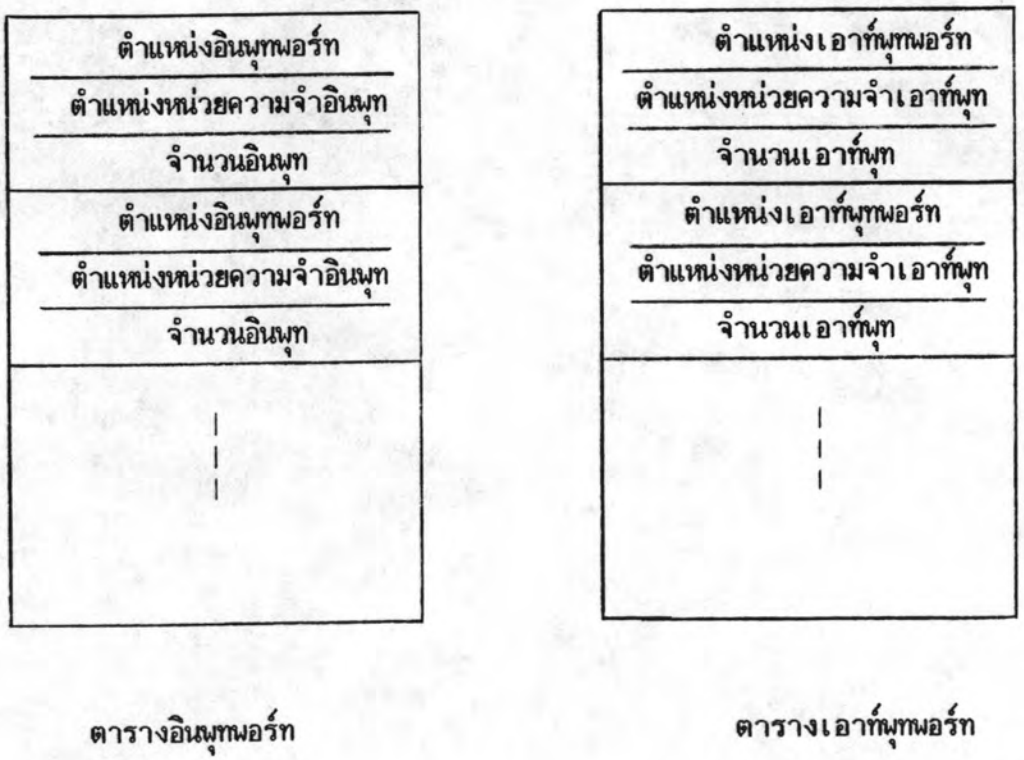
ในเครื่อง PC ขนาดใหญ่ที่มีจำนวนอินพุท/เอาท์พุทมาก และมีโปรแกรมขั้นบันไดยาว สถานะของอินพุทขณะที่ทำงานต้นโปรแกรม กับสถานะอินพุทเมื่อทำงานปลายโปรแกรม อาจมีค่าแตกต่างกัน ซึ่งอาจมีผลทำให้การทำงานของโปรแกรมควบคุมไม่ถูกต้องได้

การอ่าน และเขียนอินพุท/เอาท์พุทวิธีนี้ ดูรูปที่4.15 จะสร้างตารางในการเก็บสถานะของอินพุท/เอาท์พุทในหน่วยความจำขึ้นมา และในการทำงานของซีพียูตามคำสั่งโปรแกรมขั้นบันไดก็จะติดต่อกับตารางหน่วยความจำที่เก็บข้อมูลแทน เมื่อทำงานครบหนึ่งรอบทำงาน (Scantime) ก็จะอ่านข้อมูลจากอินพุทพอร์ทต่าง ๆ มาเก็บในตาราง และนำค่าเอาท์พุทที่เก็บในตารางส่งไปยังเอาท์พุทพอร์ท ซึ่งจะมีการคงสถานะ (Latch) ของข้อมูลไว้จนกว่าจะมีการเขียนครั้งใหม่ วิธีนี้จะทำให้ค่าของอินพุท/เอาท์พุทมีค่าเหมือนกันตลอดการทำงานหนึ่งรอบการทำงาน

ในเครื่อง PC ที่ออกแบบจะใช้วิธีอ่านและเขียนอินพุทแบบ Mass input/output copy การสร้างตารางในการเก็บตำแหน่งต่าง ๆ ของอินพุท/เอาท์พุท จะแยกเป็น 2 ตาราง คือ ตารางสำหรับอินพุทพอร์ท และตารางสำหรับเอาท์พุทพอร์ท ตารางเหล่านี้จะถูกสร้างตอนเริ่มต้นขณะ Initialization



รูปที่ 4.15 แสดงการอ่าน และเขียนแบบ Mass input/output copy



รูปที่ 4.16 แสดงตารางในการเก็บอินพุท/เอาท์พุทพอร์ท

4.2 โครงสร้าง และการจัดเก็บข้อมูล

ข้อมูลที่ใช้ในเครื่อง PC มีหลายอย่างเช่น ข้อมูลอินพุท/เอาท์พุท ข้อมูลของตัวตั้งเวลาหรือตัวนับ และข้อมูลที่ใช้ในการจัดการข้อมูล การจัดแบ่งข้อมูลในเครื่อง PC นี้แบ่งออกได้หลายชนิดคือ

1. I/O Relay
2. Auxilary Relay
3. Holding Relay
4. Data memory
5. Timer/Counter Data
6. Temporary Relay
7. Timer/Counter Contact

I/O Relay หมายถึง หน่วยความจำที่เก็บสถานะของอินพุทและเอาท์พุทในการทำงาน ใน 1 รอบการทำงานของเครื่อง PC จะมีการอ่านค่าจากโมดูลอินพุทไปเก็บลงใน I/O Relay ตามหมายเลขที่กำหนด และจะมีการอ่านค่าของ I/O Relay ส่งไปยังโมดูลเอาท์พุท จำนวนของ I/O Relay ในเครื่องที่ออกแบบมี 512 ตำแหน่ง ในกรณีที่ I/O Relay ใช้งานไม่ทั้งหมด ส่วนที่เหลือสามารถใช้งานเป็น Auxilary relay ได้

Auxilary relay หมายถึง รีเลย์ช่วยในการทำงานต่างกับ I/O relay ตรงที่ไม่สามารถติดต่อกับอุปกรณ์ภายนอกได้โดยตรง ถ้าต้องการติดต่อกับอุปกรณ์ภายนอกต้องติดต่อผ่าน I/O Relay จำนวนของ I/O Relay ในเครื่องมีทั้งหมด 512 ตำแหน่ง แต่มีบางส่วนประมาณ 48 ตำแหน่งที่นำไปใช้เป็นรีเลย์ช่วยพิเศษ (Special auxilary relay) ถ้า I/O relay มีจำนวนไม่เพียงพอ เครื่อง PC ที่ออกแบบนี้ผู้ใช้สามารถนำ Auxilary relay ไปใช้แทนได้ การใช้งานต่าง ๆ ของ Auxilary relay จะเหมือนกับ I/O relay โดยปกติเมื่อมีการเปิดเครื่องหรือเปลี่ยนจากโหมดโปรแกรมไปเป็นโหมดทำงาน ข้อมูลต่าง ๆ ใน Auxilary relay จะถูกเคลียร์ก่อนเสมอ

Holding relay หมายถึง Auxilary relay แบบ Rententive การทำงานจะเหมือน Auxilary relay ทุกอย่างแต่ข้อมูลของ Holding Relay จะไม่ถูกเคลียร์เมื่อเปิดเครื่องหรือเปลี่ยนโหมดจากโหมดโปรแกรมเป็นโหมดการทำงาน Holding relay ในเครื่องถูกออกแบบให้มีทั้งหมด 512 ตำแหน่ง และในการใช้งานจะเรียกชื่อย่อว่า HR

Data memory เป็นหน่วยความจำสำหรับเก็บข้อมูลที่ใช้ในคำสั่งเกี่ยวกับการจัดการข้อมูล หน่วยความจำนี้มีขนาด 16 บิทแบ่งออกเป็นแบบ Rententive และแบบ Non-rententive

อย่างละ 256 ตำแหน่ง

Timer/counter data หมายถึงหน่วยความจำที่ใช้เก็บข้อมูลของคำสั่ง TIM หรือ คำสั่ง CNT การเก็บข้อมูลจะมีขนาด 16 บิท ต่อหนึ่งตำแหน่ง ในเครื่อง PC นี้มี Timer/ counter data ทั้งหมด 128 ตำแหน่ง

Temporary relay หมายถึงรีเลย์ที่ไว้เก็บค่าในการประมวลผลชั่วคราว ซึ่งจะใช้ได้เฉพาะคำสั่ง LD และ OUT เท่านั้น ในเครื่องที่ออกแบบมี Temperary relay ทั้งหมด 8 ตำแหน่ง ในการใช้งานจะเรียกชื่อย่อว่า TR

Timer/counter contact เป็นรีเลย์ที่เก็บผลลัพธ์ของคำสั่งเกี่ยวกับตัวตั้งเวลาและตัวนับ ซึ่งจะมีค่าเป็น ON หรือ OFF ในเครื่องมี TIM/CNT รวมกัน 128 ตัว แต่การเก็บข้อมูลเอาท์พุท จะแยกเป็นของ Timer 128 ตัว ของ Counter 128 ตัว เพื่อความสะดวกในการตรวจสอบของโปรแกรมควบคุม ซึ่งตำแหน่งที่เก็บข้อมูลจริงนั้นรวมกันทั้ง Timer และ Counter ไม่เกิน 128 ตำแหน่ง

ลักษณะของข้อมูลที่เก็บในเครื่องสามารถแบ่งออกได้เป็น 2 ชนิดคือ

1. ข้อมูลที่เป็นสถานะของอินพุท/เอาท์พุท

ข้อมูลชนิดนี้ได้แก่ I/O relay, Auxilary relay, Holding relay ข้อมูลเหล่านี้จะมีเพียง 2 สถานะคือ สถานะ ON หรือ OFF ในการจัดเก็บจะใช้หน่วยความจำ 1 ไบท์ ในการเก็บข้อมูลหนึ่งตำแหน่ง โดยให้ข้อมูล 00H แทนสถานะ OFF และข้อมูล 80H แทนสถานะ ON

1	0	0	0	0	0	0	0	สถานะ ON

0	0	0	0	0	0	0	0	สถานะ OFF

เพื่อความเหมาะสมในการเคลื่อนย้าย และในการจัดการข้อมูลต่างๆ ข้อมูลชนิดนี้ถูกจัดแบ่งออกเป็นกลุ่ม กลุ่มละ 16 ตำแหน่ง และเรียกข้อมูลแต่ละกลุ่มว่าแชลแนล (Channel) และในการอ้างอิงตำแหน่งของข้อมูลชนิดนี้จะมี 2 แบบคือ

1. อ้างอิงตำแหน่งข้อมูล จะใช้ตัวเลขฐานสิบขนาด 4 หลักในการอ้างอิงตำแหน่งของข้อมูลนั้น โดยใช้ตัวเลข 2 หลักแรกแทนหมายเลขแชลแนล และตัวเลข 2 ตำแหน่งหลังแทนตำแหน่งของข้อมูล ซึ่งมีค่าระหว่าง 00 ถึง 15

ตำแหน่งข้อมูล : XX XX

หมายเลขแชลแนล — หมายเลขตำแหน่ง (00-15)

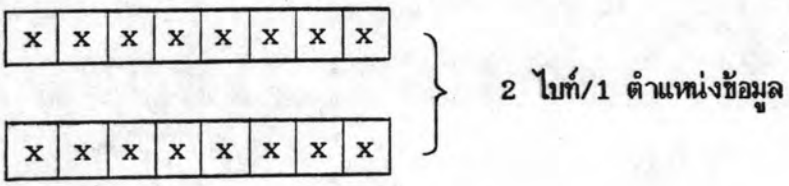
2. อ้างอิงตำแหน่งของกลุ่มข้อมูลจะใช้ตัวเลขฐานสิบขนาด 2 หลักในการอ้างอิงกลุ่มของข้อมูล โดยจะเรียกว่าหมายเลขแชลแนล ซึ่งหมายถึงกลุ่มข้อมูล 16 ตำแหน่ง

ตำแหน่งกลุ่มข้อมูล : xx
└── หมายเลขแชลแนล

สำหรับการบอกชนิดต่าง ๆ ของข้อมูลทั้ง 2 แบบ จะระบุชนิดของข้อมูลลงไว้ด้านหน้าของตำแหน่งข้อมูล และด้านหน้าของตำแหน่งกลุ่มข้อมูล

2. ข้อมูลที่เก็บค่าตัวเลข

ข้อมูลชนิดนี้ได้แก่ Data memory และ Timer/counter data ข้อมูลชนิดนี้มีขนาด 16 บิท ซึ่งข้อมูลหนึ่งตำแหน่งจะใช้หน่วยความจำในการเก็บ 2 ไบท์



ค่าตัวเลขของข้อมูลชนิดนี้อาจเป็นเลข BCD ขนาด 16 บิท หรือเลข Binary ขนาด 16 บิทก็ได้ โดยข้อมูลบิท 0 จะเก็บที่บิทต่ำของไบท์แอดเดรสต่ำ และบิท 15 จะเก็บที่บิทสูงของไบท์แอดเดรสสูง

การอ้างอิงตำแหน่งข้อมูลจะใช้ตัวเลขฐานสิบขนาด 3 หลักในการอ้างอิง โดยมีชนิดของข้อมูลระบุไว้ การอ้างอิงข้อมูลที่บิทต่าง ๆ ไม่สามารถทำได้โดยตรง แต่อาจใช้วิธีย้ายข้อมูลไปยังรีเลย์ช่วย (Auxilary relay) แล้วอ้างอิงตำแหน่งของรีเลย์ช่วยแทน

DM xxx
└── หมายเลขตำแหน่ง

การอ้างอิงตำแหน่งของข้อมูลชนิดต่างๆ ที่กล่าวมาแล้วข้างต้นสามารถสรุปเป็นตารางได้ดังนี้

	จำนวน	อ้างอิงตำแหน่ง	อ้างอิงแชลแนล
I/O Relay	512	0000 - 3115	00 - 31
Auxilary relay	512	3200 - 6315	32 - 63
Holding relay	512	HR 0000 - HR 3115	HR 00 - HR 31
Data memory			
- Retentive	256	-	DM 000 - DM 255
- Non retentive	256	-	DM 256 - DM 512
Timer/counter	128	-	TIM/CNT 000 - 127
Temporary	8	TR0 - TR7	-

ตารางที่ 4.2 แสดงชนิดของข้อมูลต่าง ๆ

Name	NO.	Relay number															
		0000 - 3115															
		00CH	01CH	02CH	03CH	04CH	05CH	06CH	07CH	08CH	09CH	10CH	11CH	12CH	13CH	14CH	15CH
		00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
		01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
		02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
		03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
		04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04
		05	05	05	05	05	05	05	05	05	05	05	05	05	05	05	05
		06	06	06	06	06	06	06	06	06	06	06	06	06	06	06	06
		07	07	07	07	07	07	07	07	07	07	07	07	07	07	07	07
		08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
		09	09	09	09	09	09	09	09	09	09	09	09	09	09	09	09
		10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
		11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
		13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
Input/		14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
Output	512	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
relay		16CH	17CH	18CH	19CH	20CH	21CH	22CH	23CH	24CH	25CH	26CH	27CH	28CH	29CH	30CH	31CH
		00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
		01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
		02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
		03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
		04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04
		05	05	05	05	05	05	05	05	05	05	05	05	05	05	05	05
		06	06	06	06	06	06	06	06	06	06	06	06	06	06	06	06
		07	07	07	07	07	07	07	07	07	07	07	07	07	07	07	07
		08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
		09	09	09	09	09	09	09	09	09	09	09	09	09	09	09	09
		10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
		11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
		13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
		14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
		15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

ตารางที่ 4.3 แสดงการเก็บข้อมูลของ I/O Relay

Name	NO.	Relay number														
		3200 - 6015														
		32CH	33CH	34CH	35CH	36CH	37CH	38CH	39CH	40CH	41CH	42CH	43CH	44CH	45CH	46CH	47CH
		00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
		01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
		02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
		03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
		04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04
		05	05	05	05	05	05	05	05	05	05	05	05	05	05	05	05
		06	06	06	06	06	06	06	06	06	06	06	06	06	06	06	06
		07	07	07	07	07	07	07	07	07	07	07	07	07	07	07	07
		08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
		09	09	09	09	09	09	09	09	09	09	09	09	09	09	09	09
		10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
		11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
		13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
		14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
Auxiliary	464	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
relay		48CH	49CH	50CH	51CH	52CH	53CH	54CH	55CH	56CH	57CH	58CH	59CH	60CH			
		00	00	00	00	00	00	00	00	00	00	00	00	00			
		01	01	01	01	01	01	01	01	01	01	01	01	01			
		02	02	02	02	02	02	02	02	02	02	02	02	02			
		03	03	03	03	03	03	03	03	03	03	03	03	03			
		04	04	04	04	04	04	04	04	04	04	04	04	04			
		05	05	05	05	05	05	05	05	05	05	05	05	05			
		06	06	06	06	06	06	06	06	06	06	06	06	06			
		07	07	07	07	07	07	07	07	07	07	07	07	07			
		08	08	08	08	08	08	08	08	08	08	08	08	08			
		09	09	09	09	09	09	09	09	09	09	09	09	09			
		10	10	10	10	10	10	10	10	10	10	10	10	10			
		11	11	11	11	11	11	11	11	11	11	11	11	11			
		12	12	12	12	12	12	12	12	12	12	12	12	12			
		13	13	13	13	13	13	13	13	13	13	13	13	13			
		14	14	14	14	14	14	14	14	14	14	14	14	14			
		15	15	15	15	15	15	15	15	15	15	15	15	15			

ตารางที่ 4.4 แสดงการเก็บข้อมูลของ Auxiliary relay

CH NO.	Relay No.	หน้าที่การทำงาน
61	6100	ถ้า ON เอาท์พุททั้งหมดจะปิด (OFF)
	6101	ถ้า ON ข้อมูลของ I/O และ Auxiliary relay จะไม่ถูกลบ
62	6200	ON เมื่อเกิด Alarm
	6201	ON เมื่อเกิด Error
	6202	ON เมื่อแบตตอรี่ในเครื่องไม่ปกติ
	6203	จะ ON หนึ่งรอบการทำงานแรกเท่านั้น
	6204	ON ตลอด
	6205	OFF ตลอด
	6206	-
	6207	-
	6208	รหัสแสดงความผิดพลาด
	6209	
	6210	
	6211	
	6212	
	6213	
	6214	
	6215	
63	6300	ใช้เป็นสัญญาณนาฬิกา 0.1 วินาที
	6301	ใช้เป็นสัญญาณนาฬิกา 0.2 วินาที
	6302	ใช้เป็นสัญญาณนาฬิกา 1.0 วินาที
	6303	ON เมื่อข้อมูล BCD มีข้อผิดพลาด
	6304	ON เมื่อเกิดตัวทดจากการคำนวณ (CY)
	6305	ON เมื่อการเปรียบเทียบได้ผลเป็นมากกว่า (>)
	6306	ON เมื่อการเปรียบเทียบได้ผลเป็นเท่ากับ (=)
	6307	ON เมื่อการเปรียบเทียบได้ผลเป็นน้อยกว่า (<)
Tempo rary relay (TR)	0	เป็นรีเลย์ชั่วคราวตัวที่ 0
	1	เป็นรีเลย์ชั่วคราวตัวที่ 1
	2	เป็นรีเลย์ชั่วคราวตัวที่ 2
	3	เป็นรีเลย์ชั่วคราวตัวที่ 3
	4	เป็นรีเลย์ชั่วคราวตัวที่ 4
	5	เป็นรีเลย์ชั่วคราวตัวที่ 5
	6	เป็นรีเลย์ชั่วคราวตัวที่ 6
	7	เป็นรีเลย์ชั่วคราวตัวที่ 7

ตาราง 4.5 แสดงรีเลย์พิเศษ

Name	NO.	Relay number															
		HR 0000 - 3115															
		00CH	01CH	02CH	03CH	04CH	05CH	06CH	07CH	08CH	09CH	10CH	11CH	12CH	13CH	14CH	15CH
		00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
		01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
		02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
		03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
		04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04
		05	05	05	05	05	05	05	05	05	05	05	05	05	05	05	05
		06	06	06	06	06	06	06	06	06	06	06	06	06	06	06	06
		07	07	07	07	07	07	07	07	07	07	07	07	07	07	07	07
		08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
		09	09	09	09	09	09	09	09	09	09	09	09	09	09	09	09
		10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
		11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
		13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
		14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
Holding	512	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
relay		16CH	17CH	18CH	19CH	20CH	21CH	22CH	23CH	24CH	25CH	26CH	27CH	28CH	29CH	30CH	31CH
		00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
		01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
		02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
		03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
		04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04
		05	05	05	05	05	05	05	05	05	05	05	05	05	05	05	05
		06	06	06	06	06	06	06	06	06	06	06	06	06	06	06	06
		07	07	07	07	07	07	07	07	07	07	07	07	07	07	07	07
		08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
		09	09	09	09	09	09	09	09	09	09	09	09	09	09	09	09
		10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
		11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
		13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
		14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
		15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

ตารางที่ 4.6 แสดงการเก็บข้อมูลของ Holding relay

Name	NO.	Data memory number															
		DM 000 - 511															
Data memory	512 X 16 bit	000	010	020	030	040	050	060	070	080	090	100	110	120	130	140	150
		001	011	021	031	041	051	061	071	081	091	101	111	121	131	141	151
		002	012	022	032	042	052	062	072	082	092	102	112	122	132	142	152
		003	013	023	033	043	053	063	073	083	093	103	113	123	133	143	153
		004	014	024	034	044	054	064	074	084	094	104	114	124	134	144	154
		005	015	025	035	045	055	065	075	085	095	105	115	125	135	145	155
		006	016	026	036	046	056	066	076	086	096	106	116	126	136	146	156
		007	017	027	037	047	057	067	077	087	097	107	117	127	137	147	157
		008	018	028	038	048	058	068	078	088	098	108	118	128	138	148	158
		009	019	029	039	049	059	069	079	089	099	109	119	129	139	149	159
		160	170	180	190	200	210	220	230	240	250	260	270	280	290	300	310
		161	171	181	191	201	211	221	231	241	251	261	271	281	291	301	311
		162	172	182	192	202	212	222	232	242	252	262	272	282	292	302	312
		163	173	183	193	203	213	223	233	243	253	263	273	283	293	303	313
		164	174	184	194	204	214	224	234	244	254	264	274	284	294	304	314
		165	175	185	195	205	215	225	235	245	255	265	275	285	295	305	315
		166	176	186	196	206	216	226	236	246	256	266	276	286	296	306	316
		167	177	187	197	207	217	227	237	247	257	267	277	287	297	307	317
		168	178	188	198	208	218	228	238	248	258	268	278	288	298	308	318
		169	179	189	199	209	219	229	239	249	259	269	279	289	299	309	319
		320	330	340	350	360	370	380	390	400	410	420	430	440	450	460	470
		321	331	341	351	361	371	381	391	401	411	421	431	441	451	461	471
		322	332	342	352	362	372	382	392	402	412	422	432	442	452	462	472
		323	333	343	353	363	373	383	393	403	413	423	433	443	453	463	473
		324	334	344	354	364	374	384	394	404	414	424	434	444	454	464	474
		325	335	345	355	365	375	385	395	405	415	425	435	445	455	465	475
		326	336	346	356	366	376	386	396	406	416	426	436	446	456	466	476
		327	337	347	357	367	377	387	397	407	417	427	437	447	457	467	477
		328	338	348	358	368	378	388	398	408	418	428	438	448	458	468	478
		329	339	349	359	369	379	389	399	409	419	429	439	449	459	469	479
		480	490	500	510												
		481	491	501	511												
		482	492	502													
		483	493	503													
		484	494	504													
		485	495	505													
		486	496	506													
		487	497	507													
		488	498	508													
		489	499	509													

ตารางที่ 4.7 แสดงการเก็บข้อมูลของ Data memory

Name	NO.	Timer/Counter number												
		TIM/CNT 000 - 127												
Timer/ Counter	128	000	010	020	030	040	050	060	070	080	090	100	110	120
		001	011	021	031	041	051	061	071	081	091	101	111	121
		002	012	022	032	042	052	062	072	082	092	102	112	122
		003	013	023	033	043	053	063	073	083	093	103	113	123
		004	014	024	034	044	054	064	074	084	094	104	114	124
		005	015	025	035	045	055	065	075	085	095	105	115	125
		006	016	026	036	046	056	066	076	086	096	106	116	126
		007	017	027	037	047	057	067	077	087	097	107	117	127
		008	018	028	038	048	058	068	078	088	098	108	118	
		009	019	029	039	049	059	069	079	089	099	109	119	

ตารางที่ 4.8 แสดงการเก็บข้อมูลของ Timer/counter

การจัดตำแหน่งของหน่วยความจำเพื่อเก็บข้อมูลชนิดต่างๆ ต้องพิจารณาถึงความเหมาะสมและความสะดวกในการเขียนโปรแกรมควบคุมด้วย เครื่องที่ออกแบบนี้ถูกจัดตำแหน่งข้อมูลให้ลงตัวเป็นหน้า (Page) โดยมีขนาดหน้าเท่ากับ 128 ไบท์ ซึ่งมีตำแหน่งในการเก็บข้อมูลต่าง ๆ ดังนี้

ชนิดข้อมูล	ขนาด	แอดเดรสที่เก็บ
I/O Relay	512 : 0000-3115	C000-C1FF
Auxiliary relay	512 : 3200-6315	C200-C3FF
Holding relay	512 : HR0000-3115	C400-C5FF
TIM/CNT relay	127 : TIM000-127	C600-C6FF
Data memory	512 : DM 000-511	C800-CFFF

ตารางที่ 4.9 แสดงตำแหน่งหน่วยความจำที่เก็บข้อมูลต่าง ๆ

ในการทำงานของคำสั่งขั้นบันไดบางคำสั่งจำเป็นต้องใช้หน่วยความจำในการเก็บสถานะในการทำงานบางอย่าง สถานะเหล่านี้ผู้ใช้ไม่จำเป็นต้องรู้ แต่จะใช้เฉพาะในการทำงานของคำสั่งนั้น คำสั่งที่ต้องใช้หน่วยความจำในการเก็บสถานะบางอย่างได้แก่ คำสั่ง SFT,DIFU, DIFD และคำสั่งเกี่ยวกับ Timer หรือ Counter การเก็บของสถานะเหล่านี้จะใช้ลักษณะเป็นหน้า (Page) เช่นเดียวกันเพื่อสะดวกในการอ้างอิง

TYPE	MEMORY ADDRESS																NO.
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
I/O 0000-3115	1	1	0	0	0	0	0	x	x	x	x	x	x	x	x	x	512
AUX 3200-6315	1	1	0	0	0	0	1	x	x	x	x	x	x	x	x	x	512
HR 0000-3115	1	1	0	0	0	1	0	x	x	x	x	x	x	x	x	x	512
TIM 000-127	1	1	0	0	0	1	1	0	x	x	x	x	x	x	x	0	128
CNT 000-127	1	1	0	0	0	1	1	0	x	x	x	x	x	x	x	1	128
DIFU/DIFD	1	1	0	0	0	1	1	1	0	x	x	x	x	x	x	x	128
SFT	1	1	0	0	0	1	1	1	1	x	x	x	x	x	x	x	128
C/T, DSET	1	1	0	0	1	0	0	0	x	x	x	x	x	x	x	0	128
PUL-ST	1	1	0	0	1	0	0	1	x	x	x	x	x	x	x	0	128
COUNT VALUE	1	1	0	0	1	0	1	0	x	x	x	x	x	x	x	0	128
PRESET VALUE	1	1	0	0	1	0	1	1	x	x	x	x	x	x	x	0	128
DM 000-511	1	1	0	0	1	1	x	x	x	x	x	x	x	x	x	x	512
SYSTEM RESERVE	1	1	0	1	x	x	x	x	x	x	x	x	x	x	x	x	4096

ตารางที่ 4.10 แสดงแอดเดรสของหน่วยความจำในการเก็บข้อมูล

4.3 การจัดเก็บคำสั่ง Ladder

การเขียนโปรแกรมในเครื่อง PC ของใช้เขียนจะเป็นภาษาขั้นบันได (Ladder)และฟังก์ชั่นบล๊อก (Function block) โดยการป้อนโปรแกรมจะแบ่งออกเป็นสองส่วนคือ นิวโมนิค และโอเปอร์แรนด์ บางคำสั่งอาจมีโอเปอร์แรนด์หลายตัว โปรแกรมที่ผู้ใช้เขียนจะถูกแปลงให้เป็นคำสั่งภาษาเครื่องที่ซีพียูสามารถทำงานได้ และเก็บลงในหน่วยความจำสำหรับเก็บโปรแกรม (User program area) สำหรับการแปลคำสั่งขั้นบันไดมาเป็นภาษาเครื่องนี้ จะเป็นการทำงานของโปรแกรมควบคุมการกดคีย์บอร์ด เมื่อมีการกดคำสั่งเขียน (write) ก็จะมีการตรวจสอบความถูกต้องของคำสั่งขั้นบันได และแปลงเป็นภาษาเครื่องเก็บลงหน่วยความจำ

คำสั่งที่ใช้ในการเขียนโปรแกรมขั้นบันไดของเครื่อง PC นี้ จะจำแนกออกได้เป็น 2 ชนิดคือ

1. คำสั่งเบื้องต้น เป็นคำสั่งเกี่ยวกับการอ่าน/เขียน และทำลอจิกต่าง ๆ ของรีเลย์ ซึ่งมีทั้งหมด 10 คำสั่งคือ

LD	LD-NOT	OR
OR-NOT	AND	AND-NOT
AND-LD	OR-LD	OUT
OUT-NOT		

2. คำสั่งฟังก์ชั่นเป็นคำสั่งในการจัดการข้อมูลต่าง ๆ รวมทั้งคำสั่งในการควบคุมการทำงานของโปรแกรมขั้นบันได ประกอบด้วยคำสั่งทั้งหมด 54 คำสั่งคือ

NOP (FUN00)	END (FUN01)	IL (FUN02)
ILC (FUN03)	JMP (FUN04)	JME (FUN05)
F06 (FUN06)	F07 (FUN07)	TIM (FUN08)
CNT (FUN09)	SFT (FUN10)	KEEP(FUN11)
CNTR(FUN12)	DIFU(FUN13)	DIFD(FUN14)
TIMH(FUN15)	WSFT(FUN16)	MOVL(FUN17)
MOVH(FUN18)	F19 (FUN19)	CMP (FUN20)
MOV (FUN21)	MVN (FUN22)	BIN (FUN23)
BCD (FUN24)	ASL (FUN25)	ASR (FUN26)
ROL (FUN27	ROR (FUN28)	COM (FUN29)
ADD (FUN30)	SUB (FUN31)	MUL (FUN32)
DIV (FUN33)	ANDW(FUN34)	ORW (FUN35)

XORW(FUN36)	XNRW(FUN37)	INC (FUN38)
DEC (FUN39)	STC (FUN40)	CLC (FUN41)
BKM (FUN42)	BKS (FUN43)	DAEX(FUN44)
ODSL(FUN45)	ODSR(FUN46)	DECO(FUN47)
ENCO(FUN48)	7SEG(FUN49)	F50 (FUN50)
F51 (FUN51)	F52 (FUN52)	F53 (FUN53)

การจัดเก็บคำสั่งเบื้องต้นจะมีลักษณะเป็นเอ็กซีคิ้วโปรแกรมคือ เป็นคำสั่งภาษาเครื่องที่ซีพียูสามารถทำงานได้เลย โดยค่าโอเปอร์แรนด์ของคำสั่งขั้นบันไดจะถูกแปลง และเก็บไว้ในเอ็กซีคิ้วโปรแกรมแล้ว เช่น

คำสั่ง LD 0001

จะมีการเก็บเป็นภาษาเครื่องคือ

SRL	:	CB 3F
LD HL, C001	:	21 01 C0
OR (HL)	:	B6

คำสั่ง LD 0001 = CB 3F 21 01 C0 B6

จะเห็นว่าคำสั่งขั้นบันได LD 0001 ถูกแปลงเป็นภาษาเครื่องมีความยาว 6 ไบท์ และโอเปอร์แรนด์ 0001 จะถูกแปลงเป็นแอดเดรสของหน่วยความจำที่เก็บข้อมูลตำแหน่ง 0001 ซึ่งมีค่าเท่ากับ 0C001H

สำหรับคำสั่งฟังก์ชั่นต่าง ๆ นั้น แบ่งออกได้ 2 อย่างคือ คำสั่งที่มีโอเปอร์แรนด์ และคำสั่งที่ไม่มีโอเปอร์แรนด์ การจัดเก็บคำสั่งฟังก์ชั่นนี้จะเก็บในลักษณะเป็นคำสั่งการเรียกไปยังตารางของสับรูทีน(Subroutine) ตามด้วยชนิดของโอเปอร์แรนด์และโอเปอร์แรนด์ที่ 1, 2 และ 3 ตามลำดับ การดูว่าคำสั่งที่เก็บนี้เป็นฟังก์ชั่นอะไร สามารถหาได้จากค่าของตำแหน่งในตารางที่สับรูทีน (Subroutine) ต้องไปทำงาน โดยตารางนี้จะเก็บไว้ที่ตำแหน่ง 0300H ถึง 0400H

ตัวอย่าง คำสั่ง MOV (21) -

\- 02

\- HR 05

จะมีการเก็บคำสั่งเป็น

CALL 033F : CD 03 3F

Data type : 00

S (Source) : 20 C0

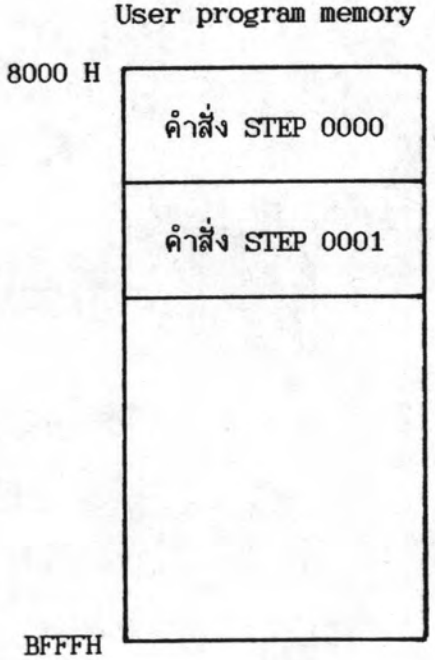
D (Destination) : 50 C4

คำสั่ง : MOV (21) - จะเก็บในหน่วยความจำคือ CD 03 3F 00 20 C0 50 C4

\- 02

\- HR05

เครื่อง PC ที่ออกแบบนี้สามารถเขียนโปรแกรมได้ประมาณ 4000 คำสั่ง ซึ่งจำนวนที่แน่นอนขึ้นอยู่กับความยาวของแต่ละคำสั่งที่เขียนโปรแกรม โปรแกรมที่ผู้ใช้เขียนนี้จะเก็บในหน่วยความจำขนาด 16K bytes ตำแหน่ง 8000H ถึง BFFFH การเก็บจะมีลักษณะเป็นการนำคำสั่งต่าง ๆ ที่แปลงเป็นภาษาเครื่องแล้ว มาเรียงต่อกันโดยเริ่มจากสเต็ป 0000 เก็บที่ตำแหน่ง 8000H แล้วเก็บคำสั่งสเต็ปอื่น ๆ เรียงต่อไปตามลำดับ



FUN NO	INSTRUCTION	1	2	3	4	5	6	7	8	9	10	REMARK
	LD XXXX	SRL		LD HL,XXXX			OR HL					
		CB	3F	21	XXXX		B6					
	LD NOT XXXX	SRL		LD HL,XXXX			OR HL	XOR C				
		CB	3F	21	XXXX		B6	A9				
	OR XXXX	LD HL,XXXX			OR HL							
		21	XXXX		B6							
	OR NOT XXXX	LD B,A	LD A,(XXXX)			XOR C	OR B					
		47	3A	XXXX		A9	B0					
	AND XXXX	LD B,A	LD A,(XXXX)			OR 7FH		AND B				
		47	3A	XXXX		F6	F7	A0				
	AND NOT XXXX	LD B,A	LD A,(XXXX)			XOR C	OR 7FH		AND B			
		47	3A	XXXX		A9	F6	7F	A0			
	AND LD	ADD A,A	JR C,02		AND 7FH							
		87	38	02	E6	7F						
	OR LD	ADD A,A	JR NC,01		OR C							
		87	30	01	B1							
	OUT XXXX	AND C	LD (XXXX),A									
		A1	32	XXXX								
	OUT NOT XXXX	AND C	XOR C	LD (XXXX),A			XOR C					
		A1	A9	32	XXXX		A9					

ตารางที่ 4.11 แสดงการเก็บคำสั่งเบื้องต้น

FUN NO	INSTRUCTION	1	2	3	4	5	6	7	8	9	10	REMARK
00	NOP	NOP	NOP	NOP	NOP							
		00	00	00	00							
01	END	CALL 0304										
		CD	04	03								
02	IL	CALL 0308										
		CD	08	03								
03	ILC	CALL 030C										
		CD	0C	03								
04	JMP	CALL 0310										
		CD	10	03								
05	JME	CALL 0314										
		CD	14	03								
06	F06	CALL 0318										
		CD	18	03								
07	F07	CALL 031C										
		CD	1C	03								
08	TIM	CALL 0320			NO.	DT	D1					
	D1	CD	20	03	XX	- 00**	XXXX					
09	CNT	CALL 0324			NO.	DT	D1					
	D1	CD	24	03	XX	- 00**	XXXX					

ตารางที่ 4.12 แสดงการเก็บคำสั่งฟังก์ชั่น

FUN NO	INSTRUCTION	1	2	3	4	5	6	7	8	9	10	REMARK
10	SFT	CALL 0328			DT	D1		D2				
	D1,D2	CD	28	03	- ****	XXXX		XXXX				
11	KEEP XXXX	CALL 032C			OPERAND I							
		CD	2C	03	XXXX							
12	CNTR XXX	CALL 0330			NO.	DT	D1					
	D1	CD	30	03	XX	- 00**	XXXX					
13	DIFU XXXX	CALL 0334			OPERAND I							
		CD	34	03	XXXX							
14	DIFD XXXX	CALL 0338			OPERAND I							
		CD	38	03	XXXX							
15	TIMH XXX	CALL 033C			NO.	DT	D1					
	D1	CD	3C	03	XX	- 00**	XXXX					
16	WSFT	CALL 0340			DT	D1		D2				
	D1,D2	CD	40	03	- ****	XXXX		XXXX				
17	MOVL	CALL 0344			DT	S1		D1				
	S1,D1	CD	44	03	- ****	XXXX		XXXX				
18	MOVH	CALL 0348			DT	S1		D1				
	S1,D1	CD	48	03	- ****	XXXX		XXXX				
19	F19	CALL 034C										
		CD	4C	03								

ตารางที่ 4.13 แสดงการเก็บคำสั่งฟังก์ชั่น (ต่อ)

FUN NO	INSTRUCTION	1	2	3	4	5	6	7	8	9	10	REMARK
20	CMP	CALL 0350			DT	S1		S2				
	S1,S2	CD	50	03	- ****	XXXX		XXXX				
21	MOV	CALL 0354			DT	S1		D1				
	S1,D1	CD	54	03	- ****	XXXX		XXXX				
22	MVN	CALL 0358			DT	S1		D1				
	S1,D1	CD	58	03	- ****	XXXX		XXXX				
23	BIN	CALL 035C			DT	S1		D1				
	S1,D1	CD	5C	03	- ****	XXXX		XXXX				
24	BCD	CALL 0360			DT	S1		D1				
	S1,D1	CD	60	03	- ****	XXXX		XXXX				
25	ASL	CALL 0364			DT	D						
	D	CD	64	03	- 00**	XXXX						
26	ASR	CALL 0368			DT	D						
	D	CD	68	03	- 00**	XXXX						
27	ROL	CALL 036C			DT	D						
	D	CD	6C	03	- 00**	XXXX						
28	ROR	CALL 0370			DT	D						
	D	CD	70	03	- 00**	XXXX						
29	COM	CALL 0374			DT	D						
	D	CD	74	03	- 00**	XXXX						

ตารางที่ 4.14 แสดงการเก็บคำสั่งฟังก์ชั่น (ต่อ)

FUN NO	INSTRUCTION	1	2	3	4	5	6	7	8	9	10	REMARK
30	ADD	CALL 0378			DT	S1		S2		D		
	S1,S2,D	CD	78	03	-******	XXXX		XXXX		XXXX		
31	SUB	CALL 037C			DT	S1		S2		D		
	S1,S2,D	CD	7C	03	-******	XXXX		XXXX		XXXX		
32	MUL	CALL 0380			DT	S1		S2		D		
	S1,S2,D	CD	80	03	-******	XXXX		XXXX		XXXX		
33	DIV	CALL 0384			DT	S1		S2		D		
	S1,S2,D	CD	84	03	-******	XXXX		XXXX		XXXX		
34	ANDW	CALL 0388			DT	S1		S2		D		
	S1,S2,D	CD	88	03	-******	XXXX		XXXX		XXXX		
35	ORW	CALL .038C			DT	S1		S2		D		
	S1,S2,D	CD	8C	03	-******	XXXX		XXXX		XXXX		
36	XORW	CALL 0390			DT	S1		S2		D		
	S1,S2,D	CD	90	03	-******	XXXX		XXXX		XXXX		
37	XNRW	CALL 0394			DT	S1		S2		D		
	S1,S2,D	CD	94	03	-******	XXXX		XXXX		XXXX		
38	INC	CALL 0398			DT	D						
	D	CD	98	03	- 00**	XXXX						
39	DEC	CALL 039C			DT	D						
	D	CD	9C	03	- 00**	XXXX						

ตารางที่ 4.15 แสดงการเก็บคำสั่งฟังก์ชั่น (ต่อ)

FUN NO	INSTRUCTION	1	2	3	4	5	6	7	8	9	10	REMARK
40	STC	CALL 03A0										
		CD	A0	03								
41	CLC	CALL 03A4										
		CD	A4	03								
42	BKM	CALL 03A8			DT	W		S		D		
	W,S,D	CD	A8	03	-******	XXXX		XXXX		XXXX		
43	BKS	CALL 03AC			DT	S		D1		D2		
	S,D1,D2	CD	AC	03	-******	XXXX		XXXX		XXXX		
44	DAEX	CALL 03B0			DT	D1		D2		-		
	D1,D2,-	CD	B0	03	-******	XXXX		XXXX		-		
45	ODSL	CALL 03B4			DT	D1		D2		-		
	D1,D2,-	CD	B4	03	-******	XXXX		XXXX		-		
46	ODSR	CALL 03B8			DT	D1		D2		-		
	D1,D2,-	CD	B8	03	-******	XXXX		XXXX		-		
47	DECO	CALL 03BC			DT	S		K		D		
	S,K,D	CD	BC	03	-******	XXXX		0-3		XXXX		
48	ENCO	CALL 03C0			DT	S		K		D		
	S,K,D	CD	C0	03	-******	XXXX		0-3		XXXX		
49	7SEG	CALL 03C4			DT	S		K		D		
	S,K,D	CD	C4	03	-******	XXXX		0-3		XXXX		

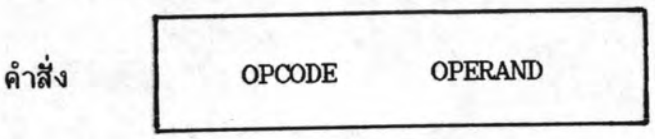
ตารางที่ 4.16 แสดงการเก็บคำสั่งฟังก์ชั่น (ต่อ)

FUN NO	INSTRUCTION	1	2	3	4	5	6	7	8	9	10	REMARK
50	F50	CALL 03C8										
		CD	C8	03								
51	F51	CALL 03CC										
		CD	CC	03								
52	F52	CALL 03CF										
		CD	CF	03								
53	F53	CALL 03D4										
		CD	D4	03								

ตารางที่ 4.17 แสดงการเก็บคำสั่งฟังก์ชั่น (ต่อ)

4.4 การทำงานของคำสั่งเบื้องต้น

คำสั่งเบื้องต้นเป็นคำสั่งเกี่ยวกับการอ่านค่าอินพุท การทำตรรกศาสตร์และการส่งผลลัพธ์ไปยังเอาท์พุท ซึ่งได้แก่คำสั่ง LD, LD NOT, OR, OR NOT, AND, AND NOT, AND LD, OR LD, OUT และ OUT NOT รูปแบบของคำสั่งจะประกอบด้วยออ๊อฟโค๊ด (Opcode) และโอเปอร์แรนด์ (Operand)



โอเปอร์แรนด์จะประกอบด้วยชนิดของข้อมูล และตำแหน่งของข้อมูล คำสั่งแต่ละคำสั่งจะสามารถใช้กับโอเปอร์แรนด์ได้แตกต่างกันดังนี้

Content of Operand	LD	LD NOT OR OR NOT AND AND NOT	OUT	OUT NOT	AND LD OR LD
I/O and AUX relay	0000-6307		0000-6015		-
Holding relay	HR 0000-3115				-
Timer, counter	TIM/CNT 000-127		-		-
Temporary relay	TRO-7	-	TRO-7	-	-

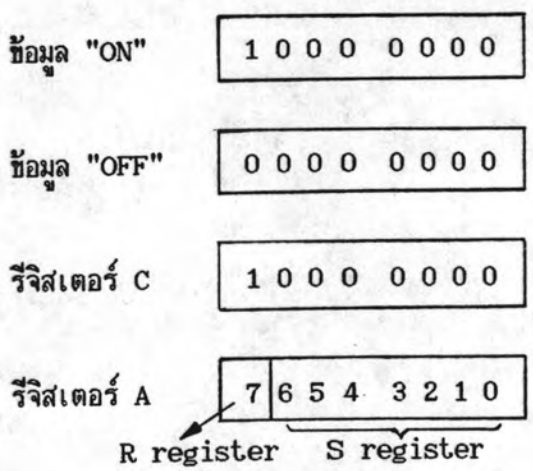
ตารางที่ 4.18 แสดงการใช้โอเปอร์แรนด์ของคำสั่งเบื้องต้น

การทำงานของคำสั่งเบื้องต้นเหล่านี้จะเกี่ยวข้องกับข้อมูล 2 สถานะคือ สถานะ ON และสถานะ OFF โดยสถานะ ON จะมีข้อมูลเป็น 80 H และสถานะ OFF จะมีข้อมูลเป็น 00H ในการใช้งานของผู้ใช้จะมองเสมือนว่ามีการทำงานของคำสั่งจะเกี่ยวข้องกับรีจิสเตอร์ 2 ชนิด คือ Result register และ Stack register

Result register (R) หมายถึงที่เก็บข้อมูลของผลลัพท์ ในการทำงานของคำสั่งต่าง ๆ

Stack register (S) หมายถึงที่พักข้อมูลชั่วคราว สำหรับไว้ใช้ในคำสั่งถัดไป

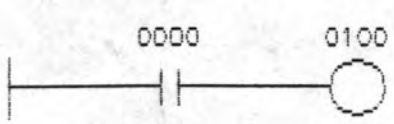
ในเครื่อง PC ที่ออกแบบนี้จะใช้แอคคิวมูเลเตอร์รีจิสเตอร์ของซีพียู (A register) เป็นที่เก็บข้อมูลของ R และ S โดย R จะถูกเก็บที่บิท 7 และ S จะเก็บที่บิท 6 จนถึงบิท 1 ซึ่งทำให้มี S ถึง 7 ตัว ทำให้สามารถโปรแกรมขั้นบันไดที่ต้องเก็บสถานะข้อมูลไว้ได้ถึง 7 ขั้น การทำงานคำสั่งเหล่านี้ของซีพียูจะต้องให้รีจิสเตอร์ C มีค่าเป็น 80H ด้วย ซึ่งจะอธิบายรายละเอียดในการทำงานของแต่ละคำสั่งต่อไป



รูปที่ 4.17 แสดงการเก็บข้อมูลของรีจิสเตอร์ภายในขณะทำงาน

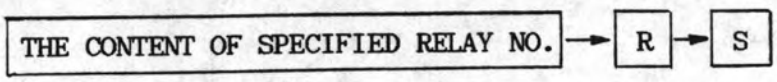
4.4.1 คำสั่ง LOAD (LD)

จะใช้คำสั่ง LD เป็นคำสั่งเริ่มต้นสำหรับรีเลย์อินพุทตัวแรกของแต่ละแถว ที่มีหน้าสัมผัสเป็นแบบปกติเปิด (Normally open)



CODING

STEP	PECODE	OPERAND
0100	LD	0000
0101	OUT	0100



รูปที่ 4.18 แสดงการใช้งาน และการทำงานของคำสั่ง LD

การทำงานของคำสั่ง LD จะอ่านข้อมูลอินพุทตามหมายเลขที่ระบุในโอเปอร์แรนด์ของคำสั่งมาเก็บไว้ในรีจิสเตอร์ R และนำค่าสถานะเดิมของรีจิสเตอร์ R ไปเก็บไว้ที่รีจิสเตอร์ S

การทำงานภาษาเครื่องของคำสั่ง LD

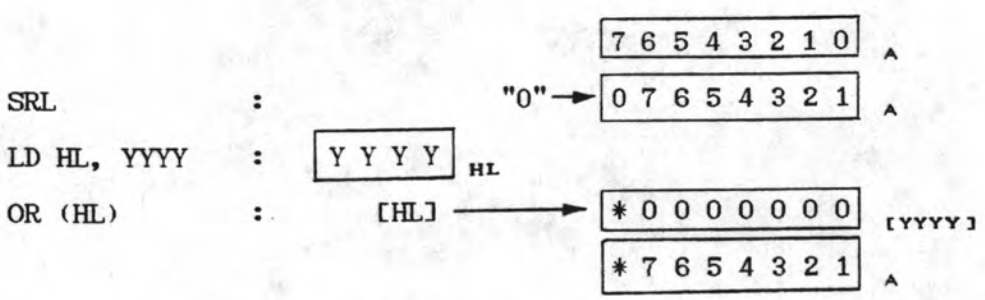
คำสั่ง LD เมื่อแปลงเป็นภาษาเครื่องมีคำสั่งดังนี้

```
LD xxxx  :  SRL
            LD  HL,YYYY
            OR  (HL)
```

คำสั่ง SRL เป็นการเลื่อนค่า S ไปหนึ่งบิท และย้ายข้อมูลจาก R ไปยัง S

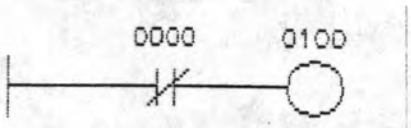
คำสั่ง LD HL, YYYY เป็นการกำหนดตัวชี้ตำแหน่งข้อมูล

คำสั่ง OR (HL) อ่านข้อมูลสถานะของอินพุทไปเก็บไว้ที่ R



4.4.2 คำสั่ง LOAD-NOT (LD-NOT)

จะใช้คำสั่ง LD-NOT เป็นคำสั่งเริ่มต้นสำหรับรีเลย์อินพุทตัวแรกของแต่ละแถวที่มีหน้าสัมผัสเป็นแบบปกติปิด (Normally close)



CODING

STEP	OPCODE	OPERAND
0102	LD NOT	0000
0103	OUT	0100

รูปที่ 4.19 แสดงการใช้งาน และการทำงานของคำสั่ง LD-NOT

การทำงานของคำสั่ง LD-NOT จะอ่านข้อมูลสถานะอินพุทตามหมายเลขที่ระบุในโอเปอร์แรนด์ของคำสั่ง แล้ว Inverse ข้อมูลและนำไปเก็บในรีจิสเตอร์ R นำค่าสถานะเดิมของรีจิสเตอร์ R ไปเก็บไว้ในรีจิสเตอร์ S

การทำงานภาษาเครื่องของคำสั่ง LD-NOT

คำสั่ง LD-NOT เมื่อแปลงเป็นภาษาเครื่องจะมีคำสั่งดังนี้

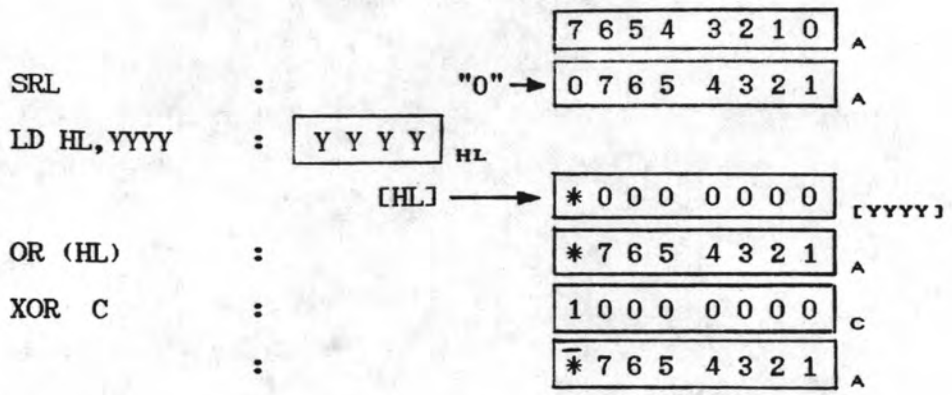
```
LD-NOT xxxx     :     SRL
                      LD  HL, YYYY
                      OR  (HL)
                      XOR C
```

คำสั่ง SRL เป็นการเลื่อนค่าของ S และย้ายค่ารีจิสเตอร์ R ไปยังรีจิสเตอร์ S

คำสั่ง LD HL, YYYY เป็นการกำหนดตัวชี้ตำแหน่งข้อมูล

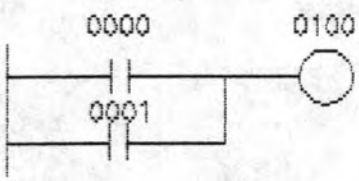
คำสั่ง OR (HL) เป็นการอ่านข้อมูลสถานะอินพุทไปเก็บไว้ที่รีจิสเตอร์ R

คำสั่ง XOR C เป็นการ Inverse ค่าของรีจิสเตอร์ R



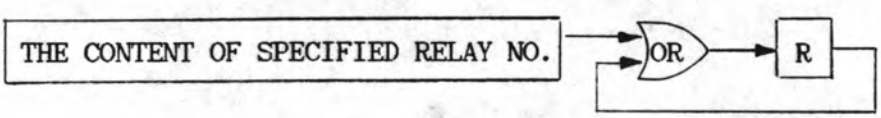
4.4.3 คำสั่ง OR

คำสั่ง OR จะใช้แทนรีเลย์อินพุทแบบปกติเปิด (NO) ที่ต่อขนานอยู่



CODING

STEP	OPCODE	OPERAND
0000	LD	0000
0001	OR	0001
0002	OUT	0100



รูปที่ 4.20 แสดงการใช้งาน และการทำงานของคำสั่ง OR

การทำงานของคำสั่ง OR จะเป็นการอ่านสถานะข้อมูลที่ถูกระบุตำแหน่งโดยโอเปอร์แรนด์มาทำลอจิก OR กับค่าของรีจิสเตอร์ R แล้วนำค่าไปเก็บไว้ที่รีจิสเตอร์ R

การทำงานภาษาเครื่องของคำสั่ง OR

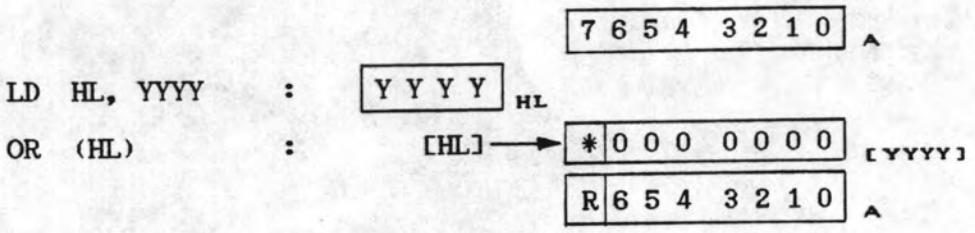
คำสั่ง OR เมื่อแปลงเป็นภาษาเครื่องมีคำสั่งดังนี้

OR xxxx : LD HL, xxxx

OR (HL)

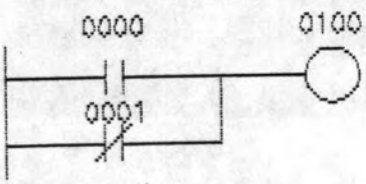
คำสั่ง LD HL, xxxx เป็นการกำหนดตัวชี้ตำแหน่งของข้อมูล

คำสั่ง OR (HL) เป็นการ OR ข้อมูลอินพุทกับ R และเก็บไว้ที่ R



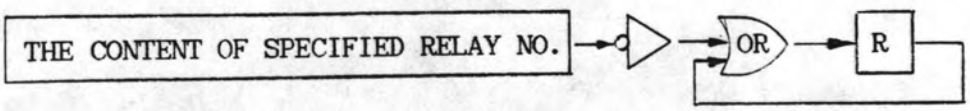
4.4.4 คำสั่ง OR-NOT

คำสั่ง OR-NOT จะใช้แทนรีเลย์อินพุทแบบปกติปิด (NC) ที่ต่อขนานอยู่



CODING

STEP	OPCODE	OPERAND
0000	LD	0000
0001	OR NOT	0001
0002	OUT	0100



รูปที่ 4.21 แสดงการใช้งาน และการทำงานของคำสั่ง OR-NOT

การทำงานของคำสั่ง OR-NOT จะเป็นการอ่านสถานะข้อมูลที่ตำแหน่ง ซึ่งถูกระบุโดยโอเปอร์แรนด์ในคำสั่ง และ Inverse ข้อมูล แล้วทำลอจิก OR กับรีจิสเตอร์ R นำค่าผลลัพธ์ไปเก็บไว้ในรีจิสเตอร์ R

การทำงานภาษาเครื่องของคำสั่ง OR-NOT

คำสั่ง OR NOT xxxx เมื่อแปลงเป็นภาษาเครื่องมีคำสั่งดังนี้

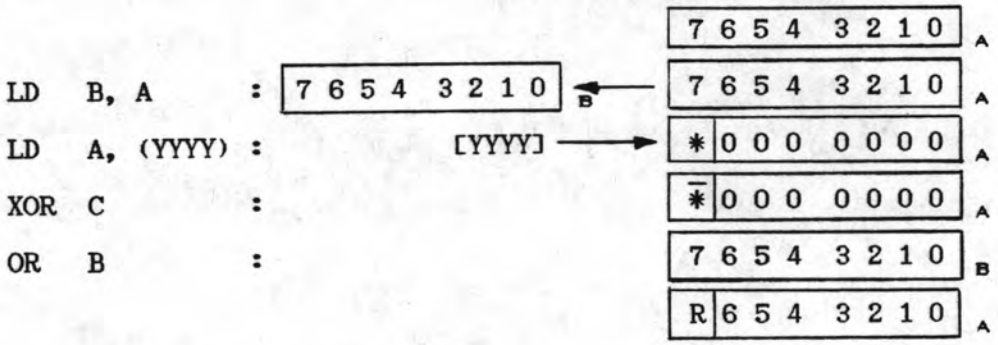
```
OR xxxx  :  LD   B, A
            LD   A, (YYYY)
            XOR  C
            OR   B
```

คำสั่ง LD B,A เก็บค่าของ R และ S ไว้ที่รีจิสเตอร์ B

คำสั่ง LD A, (YYYY) อ่านคำข้อมูลสถานะของอินพุท

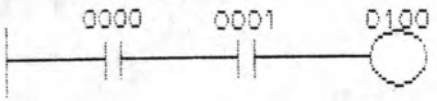
คำสั่ง XOR C Inverse ค่าข้อมูลสถานะของอินพุท

คำสั่ง OR B นำค่า Inverse ของข้อมูลไปทำลอจิก OR กับค่า R และเก็บไว้ที่ R



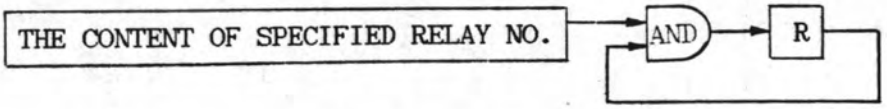
4.4.5 คำสั่ง AND

คำสั่ง AND จะใช้แทนรีเลย์อินพุทแบบปกติเปิด (NO) ที่ต่ออนุกรมอยู่



CODING

STEP	OPCODE	OPERAND
0000	LD	0000
0001	AND	0001
0002	OUT	0100



รูปที่ 4.22 แสดงการใช้งาน และการทำงานของคำสั่ง AND

การทำงานของคำสั่ง AND จะอ่านสถานะข้อมูลที่ตำแหน่งซึ่งระบุโดยโอเปอร์แรนด์ในคำสั่ง นำมาทำลอจิก AND กับค่าในรีจิสเตอร์ R แล้วเก็บผลลัพท์ไว้ในรีจิสเตอร์ R

การทำงานภาษาเครื่องของคำสั่ง AND

คำสั่ง AND xxxx แปลงเป็นภาษาเครื่องมีคำสั่งดังนี้

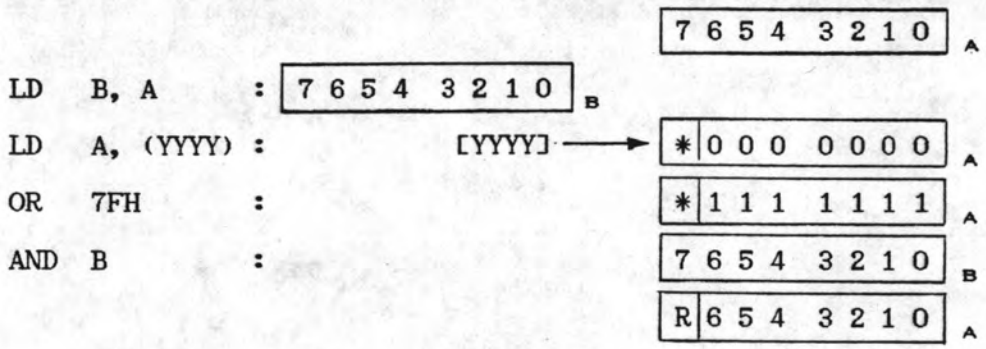
```
AND xxxx  :  LD   B, A
             LD   A, (YYYY)
             OR   7FH
             AND  B
```

คำสั่ง LD B, A ย้ายค่าของ R และ S ไปเก็บที่รีจิสเตอร์ B

คำสั่ง LD A, (YYYY) อ่านค่าอินพุทไปเก็บที่รีจิสเตอร์ A

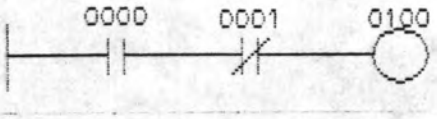
คำสั่ง OR 7FH จัดค่าข้อมูลอินพุทให้เหมาะสม

คำสั่ง AND B นำค่าของข้อมูลอินพุทไปทำลอจิก AND กับค่า R และเก็บไว้ที่ R



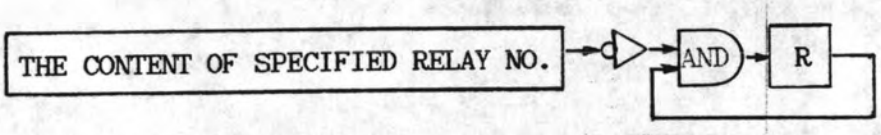
4.4.6 คำสั่ง AND NOT

คำสั่ง AND NOT ใช้แทนรีเลย์อินพุทแบบปกติปิด (NC) ที่ต่ออนุกรมอยู่



CODING

STEP	OPCODE	OPERAND
0000	LD	0000
0001	AND NOT	0001
0002	OUT	0100



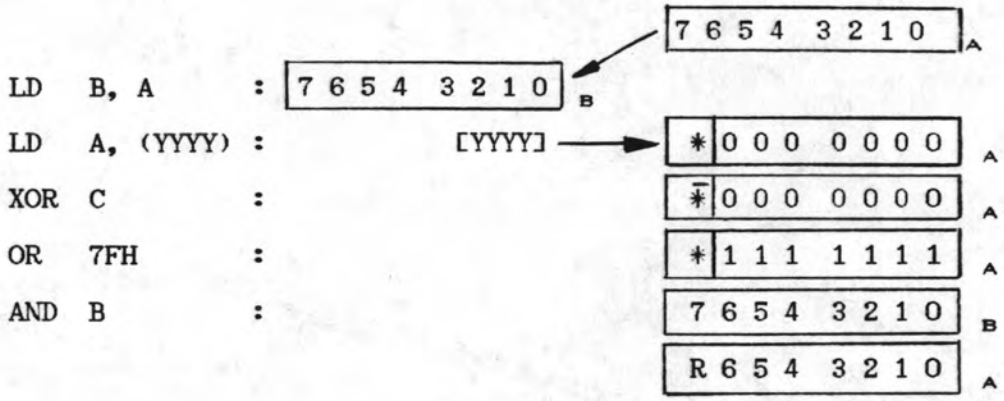
รูปที่ 4.23 แสดงการใช้งาน และการทำงานของคำสั่ง AND NOT

การทำงานของคำสั่ง AND-NOT จะอ่านสถานะข้อมูลที่ตำแหน่ง ซึ่งระบุโดยโอเปอร์แรนด์ของคำสั่ง นำไป Inverse และทำลอจิก AND กับรีจิสเตอร์ R นำผลลัพธ์ไปเก็บไว้ที่รีจิสเตอร์ R

การทำงานภาษาเครื่องของคำสั่ง AND-NOT

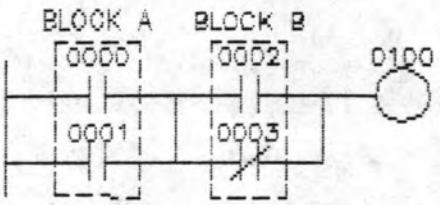
```
คำสั่ง AND NOT xxxx  :  LD   B, A
                        LD   A, (YYYY)
                        XOR  C
                        OR   7FH
                        AND  B
```

คำสั่ง LD	B, A	นำค่าของ R และ S ไปเก็บที่รีจิสเตอร์ B
คำสั่ง LD	A, (YYYY)	อ่านค่าข้อมูลสถานะของอินพุท
คำสั่ง XOR	C	Inverse ค่าข้อมูลสถานะของอินพุท
คำสั่ง OR	7FH	จัดค่าข้อมูลอินพุทให้เหมาะสม
คำสั่ง AND	B	นำค่าข้อมูลอินพุทไปทำลอจิก AND กับค่า R แล้วนำไปเก็บที่ R



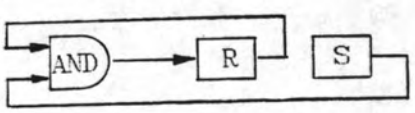
4.4.7 คำสั่ง AND LOAD (AND LD)

คำสั่ง AND LD จะใช้ในการ AND กันของบล๊อกภายในตั้งแต่ 2 บล๊อก หรือมากกว่า ดังแสดงในรูปที่ 4.24



CODING

STEP	OPCODE	OPERAND
0300	LD	0000
0301	OR	0001
0302	LD	0002
0303	OR NOT	0003
0304	AND LD	-
0305	OUT	0100



รูปที่ 4.24 แสดงการใช้ และการทำงานของคำสั่ง AND LD

การทำงานของคำสั่ง

1. จากคำสั่ง LD 0000 และ OR 0001 จะได้ผลลัพธ์ของลอจิก OR ในบล๊อก A และจะเก็บผลลัพธ์ไว้ในรีจิสเตอร์ R
2. จากคำสั่ง LD 0002 จะทำให้ค่าในรีจิสเตอร์ R ถูกย้ายไปเก็บในรีจิสเตอร์ S และผลลัพธ์ของการทำลอจิกของคำสั่ง LD 0002 และคำสั่ง OR-NOT 0003 จะถูกเก็บในรีจิสเตอร์ R
3. คำสั่ง AND-LD จะเป็นการทำลอจิก AND ระหว่างรีจิสเตอร์ R และรีจิสเตอร์ S ผลลัพธ์ที่ได้จะเก็บในรีจิสเตอร์ R

การทำงานภาษาเครื่องของคำสั่ง AND-LD

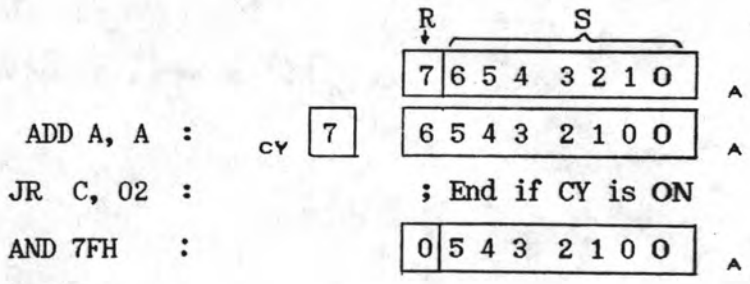
คำสั่ง AND-LD เมื่อแปลงเป็นภาษาเครื่องจะมีคำสั่งดังนี้

```
AND-LD  :  ADD  A, A
           JR   C, 02
           AND  7FH
```

คำสั่ง ADD A, A เลื่อนค่า S ไปเก็บที่ R และตรวจสอบค่า R เดิม

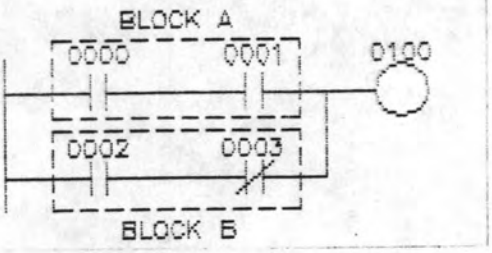
คำสั่ง JR C, 02 ถ้าค่า R เดิมมีสถานะ ON ค่าผลลัพธ์จะเท่ากับค่าของ S ที่เก็บอยู่ที่ R ในปัจจุบัน

คำสั่ง AND 7FH ถ้าค่า R เดิมมีสถานะ OFF ผลลัพธ์ที่ R ปัจจุบันต้องเซ็ทเป็น OFF



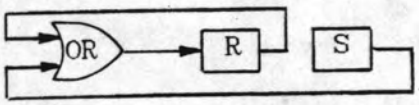
4.4.8 คำสั่ง OR-LOAD (OR-LD)

คำสั่ง OR-LD จะใช้ในการ OR กันของบล๊อกภายในตั้งแต่ 2 บล๊อก หรือมากกว่าดังแสดงในรูปที่ 4.25



CODING

STEP	OPCODE	OPERAND
0300	LD	0000
0301	AND	0001
0302	LD	0002
0303	AND-NOT	0003
0304	OR-LD	-
0305	OUT	0100



รูปที่ 4.25 แสดงการใช้ และการทำงานของคำสั่ง OR LD

การทำงานของคำสั่ง

1. จากคำสั่ง LD 0000 และ AND 0001 จะได้ผลลัพธ์ของลอจิก OR ในบล๊อก A เก็บผลลัพธ์ในรีจิสเตอร์ R
2. จากคำสั่ง LD 0002 จะทำให้ค่าในรีจิสเตอร์ R ถูกย้ายไปเก็บในรีจิสเตอร์ S และผลลัพธ์การทำลอจิกของคำสั่ง LD 0002 และคำสั่ง AND-NOT 0003 ในบล๊อก B จะเก็บในรีจิสเตอร์ R
3. คำสั่ง OR-LD จะเป็นการทำลอจิก OR ระหว่างค่าในรีจิสเตอร์ R และค่าในรีจิสเตอร์ S ผลลัพธ์ที่ได้เก็บไว้ในรีจิสเตอร์ R

การทำงานภาษาเครื่องของคำสั่ง OR-LD

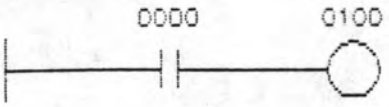
คำสั่ง OR-LD เมื่อแปลงเป็นภาษาเครื่องจะมีคำสั่งดังนี้

```
OR-LD   :   ADD   A, A
            JR    NC, 01
            OR    C
```

คำสั่ง ADD A,A จะเลื่อนค่าของ S ไปเก็บที่ R และตรวจสอบค่าของ R เดิม
คำสั่ง JR NC, 01 ถ้าค่า R เดิมมีสถานะ OFF ก็จบ
คำสั่ง OR C ถ้าค่า R เดิมมีสถานะ ON ค่าผลลัพธ์ที่ R ปัจจุบันต้องเซ็ทให้ ON

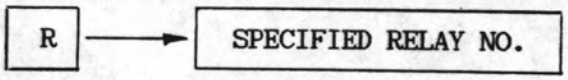
4.4.9 คำสั่ง OUT

คำสั่ง OUT ใช้สำหรับแทนเอาท์พุทรีเลย์



CODING

STEP	OPCODE	OPERAND
0200	LD	0000
0201	OUT	0100



รูปที่ 4.26 แสดงการใช้ และการทำงานของคำสั่ง OUT

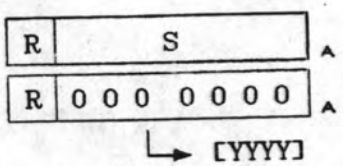
การทำงานของคำสั่ง OUT จะเป็นการนำค่าในรีจิสเตอร์ R ส่งไปที่เอาท์พุทของรีเลย์ ซึ่งถูกระบุโดยโอเปอร์แรนด์ของคำสั่ง การทำคำสั่งนี้ค่าสถานะของรีจิสเตอร์ R ยังคงเดิม แต่ค่าของรีจิสเตอร์ S จะถูกลบไป

การทำงานภาษาเครื่องของคำสั่ง OUT

คำสั่ง OUT เมื่อแปลงเป็นภาษาเครื่องจะมีคำสั่งดังนี้

```
OUT xxxx  :  AND  C
             LD   [YYYY], A
```

คำสั่ง AND C เป็นการจัดข้อมูลสำหรับเอาท์พุทให้เหมาะสม
คำสั่ง LD (YYYY), A เป็นการนำค่าของรีจิสเตอร์ R ไปเก็บในตำแหน่งเอาท์พุทที่กำหนด

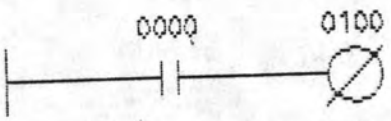


AND C :

LD (YYYY), A

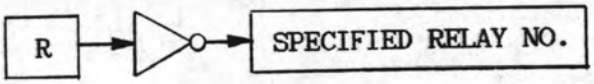
4.4.10 คำสั่ง OUT-NOT

คำสั่ง OUT-NOT ใช้แทนเอาท์พุทรีเลย์ที่เป็น Inverse



CODING

STEP	OPCODE	OPERAND
0200	LD	0001
0201	OUT NOT	0100



รูปที่ 4.27 แสดงการใช้ และการทำงานของคำสั่ง OUT-NOT

การทำงานของคำสั่ง OUT-NOT จะเป็นการนำค่าของรีจิสเตอร์ R มาทำ Inverse แล้วส่งไปที่เอาท์พุทของรีเลย์ ซึ่งถูกระบุโดยโอเปอร์แรนด์ของคำสั่ง การทำคำสั่งนี้ค่าของรีจิสเตอร์ R ยังคงเดิมแต่ค่าของรีจิสเตอร์ S จะถูกลบไป

การทำงานภาษาเครื่องของคำสั่ง OUT-NOT

คำสั่ง OUT-NOT เมื่อแปลงเป็นภาษาเครื่องจะมีคำสั่งดังนี้

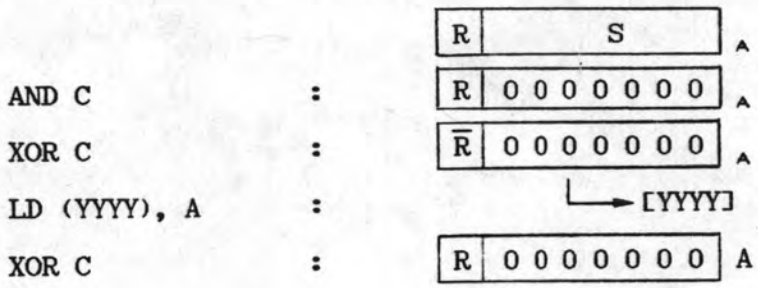
OUT-NOT xxxx :
- AND C
- XOR C
- LD [YYYY], A
- XOR C

คำสั่ง AND C เป็นจัดข้อมูลสำหรับเอาท์พุทให้เหมาะสม

คำสั่ง XOR C เป็นการ Inverse ค่าของรีจิสเตอร์ R

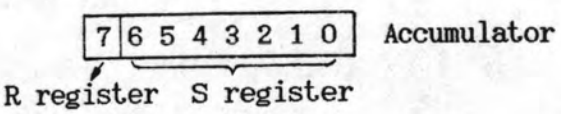
คำสั่ง LD (YYYY), A เป็นการนำค่าข้อมูลไปเก็บในตำแหน่งเอาท์พุทที่กำหนด

คำสั่ง XOR C เป็นการจัดข้อมูลของรีจิสเตอร์ R ให้ถูกต้อง



4.5 การทำงานของคำสั่งฟังก์ชั่น

คำสั่งฟังก์ชั่นมีทั้งหมด 54 คำสั่ง มีหมายเลขฟังก์ชั่นตั้งแต่ 00 ถึง 53 เงื่อนไขการทำงานของฟังก์ชั่นส่วนใหญ่ขึ้นอยู่กับค่าสถานะของรีจิสเตอร์ R ถ้ามีสถานะ ON จะทำงานตามฟังก์ชั่นนั้น ถ้ามีสถานะ OFF จะไม่ทำงานตามฟังก์ชั่นนั้น แต่มีฟังก์ชั่นบางคำสั่งที่ใช้ค่ารีจิสเตอร์ R และค่าในรีจิสเตอร์ S เป็นเงื่อนไขในการทำงาน ค่าของรีจิสเตอร์ R และ S นี้จะเป็นผลลัพธ์ของการทำคำสั่งเบื้องต้นต่าง ๆ ซึ่งจะเก็บอยู่ในแอคคิวมูเลเตอร์ของซีพียู

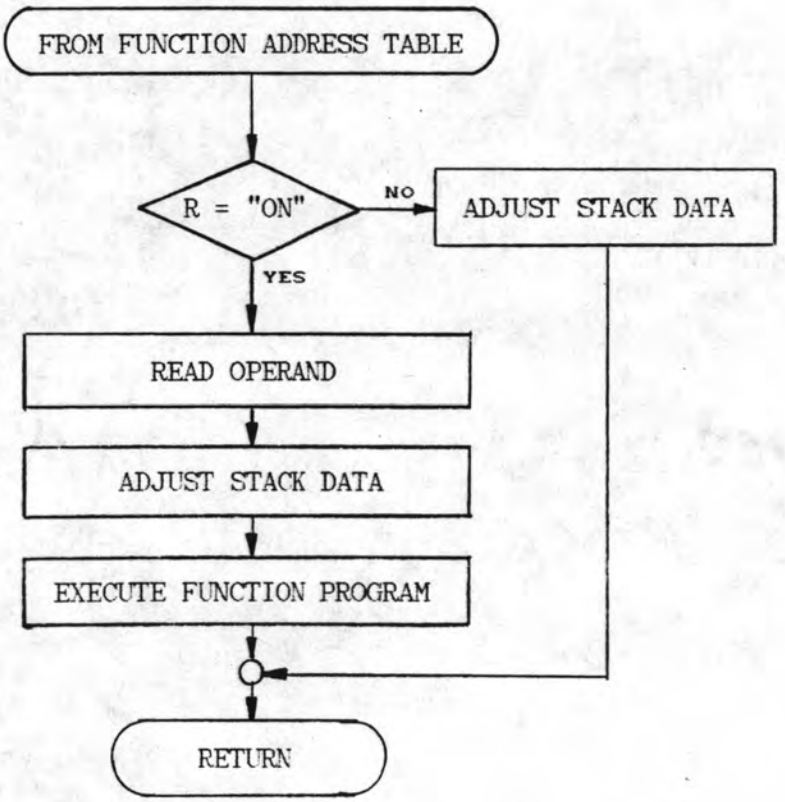


รหัสคำสั่งภาษาเครื่องของฟังก์ชั่นต่างๆ จะเป็นคำสั่งให้ซีพียูกระโดดไปทำงานที่สับรูทีนต่าง ๆ ซึ่งกำหนดไว้ในตาราง สำหรับฟังก์ชั่นที่มีโอเปอร์แรนด์ ก็จะเก็บโอเปอร์แรนด์ไว้ต่อท้ายคำสั่งกระโดดไปทำงานสับรูทีนด้วย

CALL 0354		DT	S1	D1
CD	54 03	0000 $*_2$ $*_1$	XXXX	XXXX

รูปที่ 4.28 แสดงภาษาเครื่องของคำสั่ง MOV(21)

ลักษณะการทำงานของคำสั่งฟังก์ชั่น แสดงในโฟล์วชาร์ทรูปที่ 4.29 โดยเริ่มทำงานจากคำสั่งกระโดดไปทำงานสับรูทีน (CALL) ซึ่งจะไปทำงานยังตารางที่สร้างไว้ตารางนี้จะเป็นตัวชี้ให้ไปทำงานตามโปรแกรมสำหรับฟังก์ชั่นต่างๆ ขณะนั้นค่าในสแต็กจะเป็นตำแหน่งแอดเดรสถัดจากคำสั่ง CALL ซึ่งจะเป็นตัวชี้ตำแหน่งโอเปอร์แรนด์ให้กับฟังก์ชั่นนั้น เมื่อโปรแกรมของฟังก์ชั่นทำงานเสร็จแล้ว ก็จะปรับค่าในสแต็กให้ชี้ไปยังตำแหน่งของคำสั่งชั้นบันไดตัวต่อไป แล้วทำคำสั่ง RET เพื่อไปทำงานตามคำสั่งชั้นบันไดคำสั่งต่อไป สำหรับฟังก์ชั่นที่ไม่มีโอเปอร์แรนด์ก็ไม่ต้องปรับค่าข้อมูลในสแต็ก

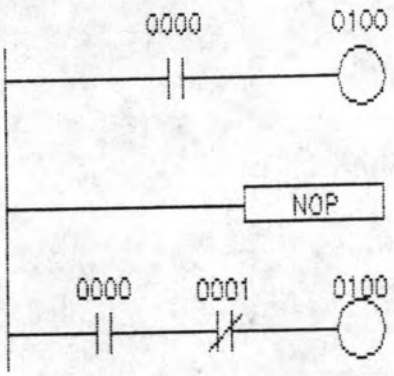


รูปที่ 4.29 แสดงโฟลว์ชาร์ทการทำงานของฟังก์ชั่นทั่วไป

ฟังก์ชั่นต่าง ๆ เหล่านี้สามารถแบ่งออกตามหน้าที่การทำงานได้หลายกลุ่ม เช่น คำสั่งเกี่ยวกับการควบคุม คำสั่งคำนวณ คำสั่งการเปรียบเทียบ คำสั่งลอจิก และคำสั่งในการแปลงรหัสต่าง ๆ สำหรับการอธิบายการทำงานและการใช้ฟังก์ชั่นต่าง ๆ ในลำดับต่อไปจะเรียงตามลำดับหมายเลขฟังก์ชั่นเป็นหลัก โดยไม่จำแนกเป็นกลุ่มต่าง ๆ

4.5.1 คำสั่ง NOP (FUN00)

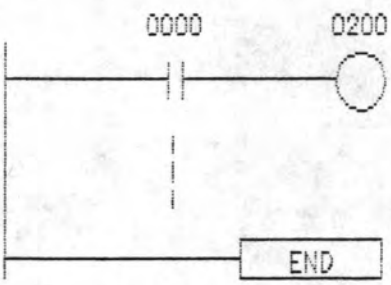
คำสั่งนี้เหมือนกับไม่มีการทำงานจะไม่ผลต่อข้อมูลต่าง ๆ ของเครื่อง คำสั่งนี้สามารถใส่เพื่อเป็นช่องว่างระหว่างคำสั่งอื่น ๆ ได้ ทุกครั้งที่มีการลบโปรแกรมทั้งหมดของผู้ใช้คำสั่ง NOP จะถูกใส่ลงหน่วยความจำทั้งหมด



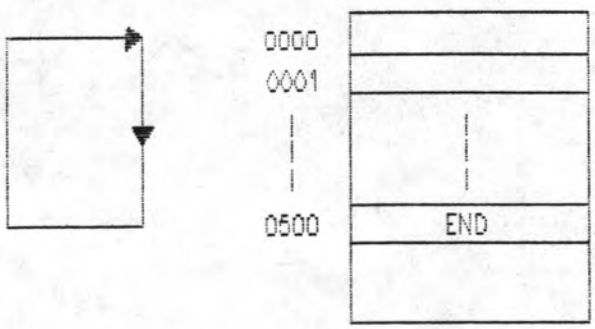
STEP	OPERAND	OPCODE
0000	LD	0000
0001	OUT	0100
0002	NOP (00)	
0003	LD	0001
0004	AND NOT	0002
0005	OUT	0101

4.5.2 คำสั่ง END (FUN01)

คำสั่ง END จะใส่เป็นคำสั่งสุดท้ายในการเขียนโปรแกรม เพื่อให้เครื่องรู้ว่าจบโปรแกรมและกลับมาเริ่มทำงานคำสั่งแรกใหม่



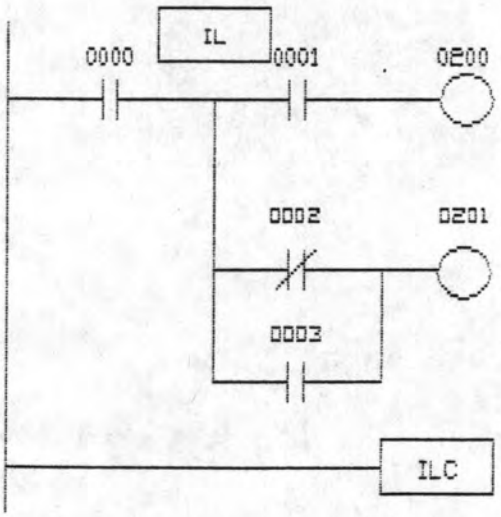
STEP	OPCODE	OPERAAND
0000	LD	0000
0500	END (01)	



ถ้าไม่มีคำสั่ง END ในโปรแกรมเครื่องจะไม่ยอมทำงาน และแสดง Error มีข้อความที่จอ LCD ว่า "NO END INSTR"

4.5.3 คำสั่ง INTERLOCK [IL (FUN02)] และ INTERLOCK CLEAR [FUN03)]

คำสั่ง IL และ ILC จะใช้ร่วมกันเป็นคู่ โดยใช้ IL เป็นจุดเริ่มต้นและ ILC เป็นจุดสุดท้ายในช่วงโปรแกรมที่ควบคุม การทำงานจะเหมือนว่าเอาท์พุททั้งหมดในช่วงคำสั่ง IL ถึง LC จะถูกควบคุมด้วยผลลัพธ์ของรีจิสเตอร์ R ขณะพบคำสั่ง IL



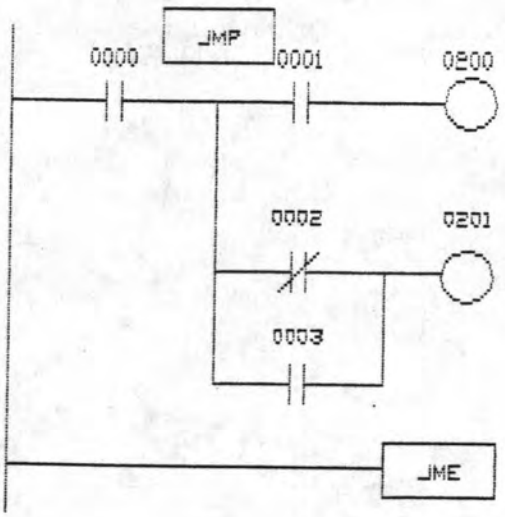
STEP	OPCODE	OPERAND
0100	LD	0000
0101	IL(02)	
0102	LD	0001
0103	OUT	0200
0104	LD NOT	0002
0105	OR	0003
0106	OUT	0201
0107	ILC(03)	

ถ้าสถานะของคำสั่ง IL เป็น ON การทำงานของโปรแกรมจะเหมือนไม่มีคำสั่ง IL และ ILC ในโปรแกรม แต่ถ้าสถานะของคำสั่ง IL เป็น OFF จะมีผลทำให้เอาท์พุท และข้อมูลต่าง ๆ ที่อยู่ระหว่างคำสั่ง IL และ ILC เป็นดังนี้

รีเลย์เอาท์พุท, รีเลย์ช่วย	OFF
ตัวตั้งเวลา	รีเซ็ท
ตัวนับ, ชิปรีจิสเตอร์	คงสถานะเดิม

4.5.4 คำสั่ง JUMP [JMP(FUN04)] และ JUMP END [JME(FUN05)]

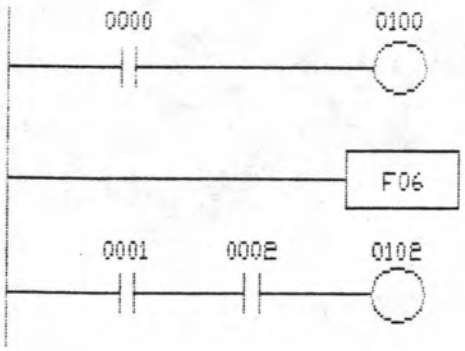
การใช้งานคำสั่ง JMP จะร่วมกับคำสั่ง JME ซึ่งมีการทำงานคือ ถ้าสถานะของคำสั่ง JMP เป็น ON โปรแกรมจะทำงานเหมือนปกติ (เหมือนไม่มีคำสั่ง JMP และ JME) แต่ถ้าสถานะของคำสั่ง JMP เป็น OFF โปรแกรมขั้นบันไดที่อยู่ระหว่างคำสั่ง JMP และคำสั่ง JME จะถูกข้ามไป



STEP	OPCODE	OPERAND
0200	LD	0000
0201	JMP(04)	
0202	LD	0001
0203	OUT	0200
0204	LD NOT	0002
0205	OR	0003
0206	OUT	0201
0207	JME(05)	

4.5.6 คำสั่ง F06 (FUN06), F07 (FUN07), F19(FUN19), F50(FUN50), F51(FUN51), F52(FUN52) และ F53(FUN52)

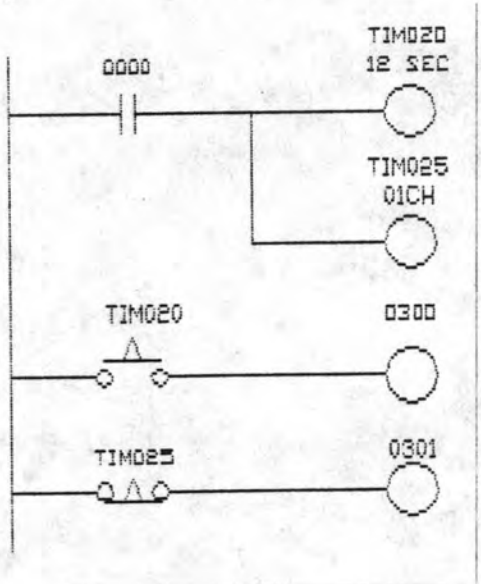
คำสั่งเหล่านี้เป็นคำสั่งว่างไม่มีผลต่อการทำงานของโปรแกรมขั้นบันได จะมีหน้าที่คล้ายคำสั่ง NOP คำสั่งเหล่านี้สามารถนำมาทำเพิ่มเป็นคำสั่งอื่น ๆ ในอนาคตได้



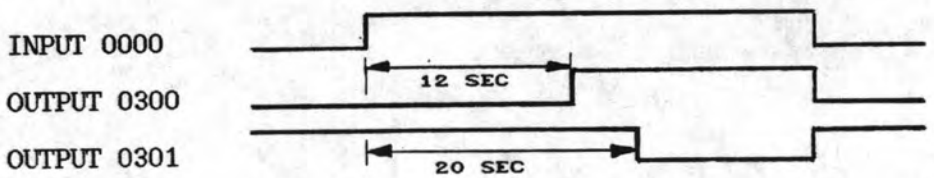
STEP	OPCODE	OPERAND
0100	LD	0000
0101	OUT	0100
0102	F06	
0103	LD	0001
0104	AND	0002
0105	OUT	0102

4.5.7 คำสั่ง TIMER [TIM(08)]

เป็นคำสั่งตัวตั้งเวลา ซึ่งเป็นแบบ ON-delay timer การทำงานสามารถแสดงได้ด้วยรูป 4.30



STEP	OPCODE	OPERAND
0200	LD	0000
0201	TIM	020
		# 0120
0202	TIM	025
		01
0203	LD	TIM 020
0204	OUT	0300
0205	LD NOT TIM	025
0206	OUT	0301



รูปที่ 4.30 แสดงการทำงานของคำสั่ง TIM

หมายเลขของ TIM อยู่ระหว่าง 000-127 ซึ่งจะต้องใช้ร่วมกับคำสั่ง Timer และ Counter อื่น ๆ ดังนั้นการใช้งานต้องระวังหมายเลขซ้ำกันด้วย

ข้อมูลที่ใช้เป็นเวลาของคำสั่ง TIM มีความละเอียด 0.1 วินาที หมายถึงค่าข้อมูลหลัง TIM จะต้องหารด้วย 10 จึงมีหน่วยเป็นวินาที

ข้อมูลที่จะใช้เป็นเวลาของคำสั่ง TIM มีหลายชนิดคือ

I/O, AUX Channel : 00-63

HR relay : HR00-31

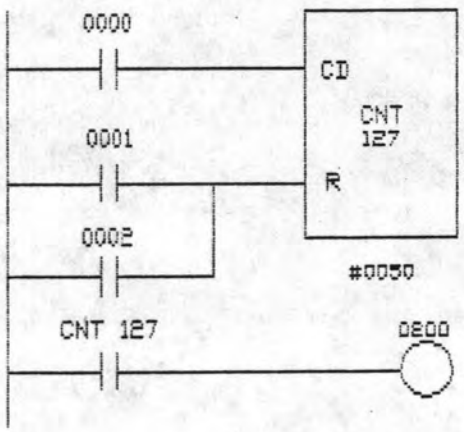
Constant : 0000-9999

เมื่อไฟดับหรือปิดเครื่องตัวตั้งเวลาจะถูกรีเซ็ทไปเริ่มต้นใหม่ ถ้าต้องการให้ยังคงสถานะข้อมูลเวลาเดิมไว้ทำงานต่อ จะต้องดัดแปลงใช้ตัวนับ (Counter) มาทำงานแทนโดยนำสัญญาณ 0.1 วินาที (I/O relay เบอร์ 6300) มาเป็นอินพุทของตัวนับ

4.5.8 คำสั่ง COUNTER [CNT(FUN09)]

คำสั่ง CNT เป็นตัวตั้งเวลาแบบนับลง (Count down) หมายถึงเมื่อมีอินพุทเข้ามาข้อมูลภายในจะถูกนับลงทีละหนึ่งจนเป็นศูนย์ หน้าสัมผัส NO ของตัวนับจะทำงานสัญญาณเข้าของตัวนับจะมี 2 สัญญาณคือ CP เป็นสัญญาณอินพุทสำหรับการนับและ R เป็นสัญญาณรีเซ็ทให้ตัวนับเริ่มทำงานใหม่

เมื่อ Counter นับลงจนข้อมูลเป็นศูนย์แล้วรีเลย์เอาท์พุทของตัวนับจะ ON ต่อเนื่องตลอดเวลาถึงแม้จะมีสัญญาณ CP เพิ่มเข้ามา จนกว่าจะมีสัญญาณรีเซ็ทเข้ามาตัวนับจึงเริ่มเซ็ทค่าจำนวนนับใหม่ และรีเลย์เอาท์พุทจะ OFF



STEP	OPCODE	OPERAND
0100	LD	0000
0101	LD	0001
0102	OR	0002
0103	CNT	127
		# 0050
0104	LD	CNT 127
0105	OUT	0200

หมายเลขของ COUNTER จะมีค่าระหว่าง 000-127 ซึ่งหมายเลขนี้ใช้ร่วมกับคำสั่ง TIMER และ COUNTER อื่น ๆ ผู้ใช้ต้องระวังหมายเลขซ้ำกัน

ข้อมูลจำนวนที่นับ และสถานะเอาท์พุทรีเลย์ของตัวนับจะยังคงค่าเดิมอยู่เมื่อไฟดับ หรือเริ่มทำงานใหม่

ข้อมูลที่จะใช้เป็นจำนวนนับของคำสั่ง CNT มีหลายชนิดคือ

I/O, AUX Channel : 00-63

HR relay : HR00-31

Constant : 0000-9999

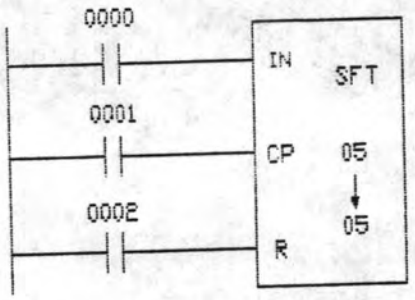
4.5.9 คำสั่ง SHIFT REGISTER [SFT (FUN10)]

เป็นคำสั่งในการเลื่อนข้อมูลครั้งละ 1 ตำแหน่ง โดยเลื่อนจากตำแหน่งต่ำไปยังตำแหน่งสูง ขนาดข้อมูลที่เลื่อนกำหนดเป็นแชลแนล ซึ่งสามารถเลื่อนมากกว่าครั้งละหนึ่งแชลแนลได้ สัญญาณของคำสั่ง SFT มี 3 สัญญาณคือ

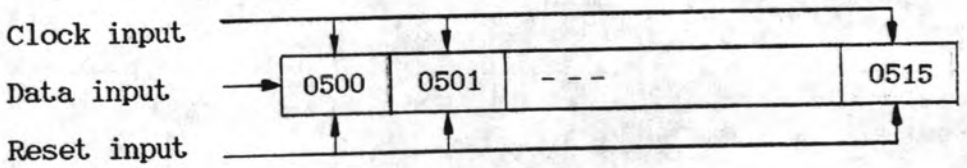
Data input (IN) เป็นข้อมูลที่จะเลื่อนเข้า

Clock input (CP) เป็นสัญญาณควบคุมการเลื่อนข้อมูล โดยใช้ขอบขาขึ้น (Leading edge)

Reset input (R) เป็นสัญญาณลบข้อมูลทั้งหมด



STEP	OPCODE	OPERAND
0300	LD	0000
0301	LD	0001
0302	LD	0002
0303	SFT	05
	-	05



รูปแบบคำสั่งของ SFT ประกอบด้วย

SFT D1 (Start channel number)

\- D2 (End channel number)

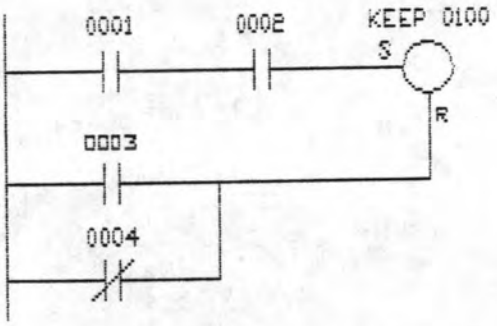
ชนิดของข้อมูลโอเปอร์แรนด์ประกอบด้วย

I/O, AUX relay : 00-60

Holding relay : HR 00-31

4.5.10 คำสั่ง KEEP (FUN11)

เป็นคำสั่งที่ใช้แทน Latchang relay คำสั่งนี้มีสัญญาณอินพุท 2 สัญญาณคือ สัญญาณเซ็ท (S) และสัญญาณรีเซ็ท (R) การทำงานของคำสั่งนี้รีเลย์เอาท์พุทจะ ON เมื่อมีสัญญาณ S เข้ามา และรีเลย์เอาท์พุท OFF เมื่อมีสัญญาณ R เข้ามา ถ้าสัญญาณ S และ R เข้ามาพร้อมกัน รีเลย์เอาท์พุทจะ OFF



STEP	OPCODE	OPERAND
0500	LD	0001
0501	AND	0002
0502	LD	0003
0503	OR NOT	0004
0506	KEEP (11)	0100

รูปแบบคำสั่ง KEEP

KEEP Operand

โอเปอร์แรนด์ของคำสั่งประกอบด้วย

I/O relay : 0000-3115

Aux relay : 3200-6015

Holding relay : HR 0000-3115

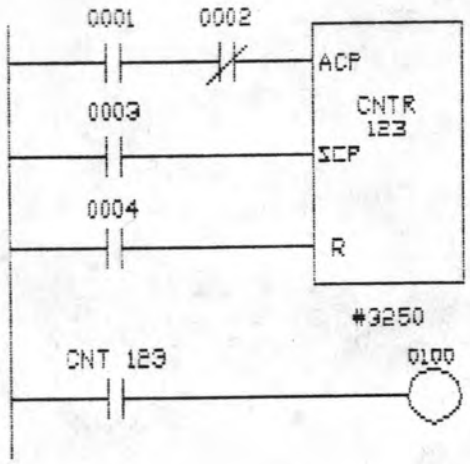
4.5.11 คำสั่ง UP-DOWN COUNTER [CNTR(FUN 12)]

เป็นคำสั่งสร้างตัวนับชนิดนับขึ้นลงได้ มีลักษณะคล้ายตัวนับแบบ CNT แต่เพิ่มสัญญาณอินพุทเข้ามาอีกหนึ่งสัญญาณ สัญญาณอินพุทจะแบ่งออกเป็น

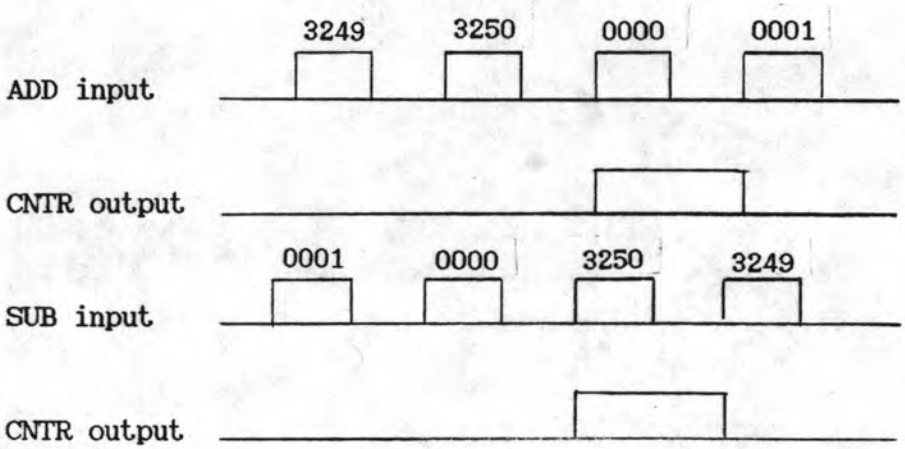
สัญญาณ ADD input (ACP) เป็นสัญญาณให้เพิ่มจำนวนนับทีละหนึ่ง

สัญญาณ SUBTRACT input (SCP) เป็นสัญญาณให้ลดจำนวนนับลงทีละหนึ่ง

สัญญาณ Reset input (R) เป็นการรีเซ็ทให้จำนวนนับเป็นศูนย์



STEP	OPCODE	OPERAND
0500	LD	0001
0501	AND-NOT	0002
0502	LD	0003
0503	LD	0004
0504	CNTR(12)	123
		# 3250
0505	LD	CNT 123
0506	OUT	0100



สัญญาณ ADD input และ SUB input เป็นแบบ Leading edge

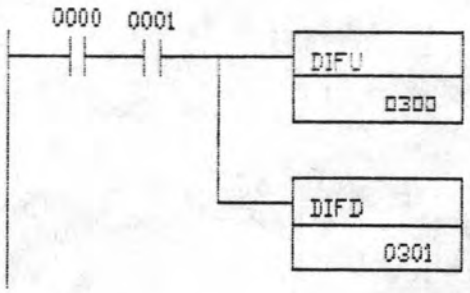
หมายเลขของตัวนับมีค่าระหว่าง 000-127 ซึ่งใช้ร่วมกับคำสั่ง TIMER และ Counter อื่น ๆ ผู้ใช้ต้องระวังไม่ให้ซ้ำกัน

ข้อมูลที่ใช้เป็นจำนวนนับของคำสั่ง CNTR ประกอบด้วย

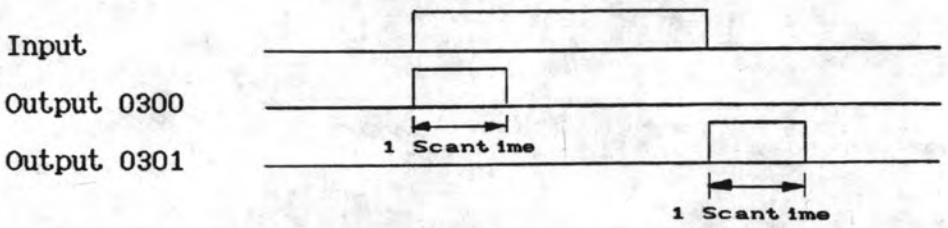
I/O, AUX channel	:	00-63
HR channel	:	HR 00-31
Constant	:	0000-9999

4.5.12 คำสั่ง DIFFERENTIATION UP [DIFU(FUN13)] และคำสั่ง DIFFRENTIATION DOWN [DIFD(FUN14)]

คำสั่ง DIFU และ DIFD เป็นคำสั่งที่ให้เอาท์พุทเพียง 1 รอบการทำงาน (Scan time) โดยคำสั่ง DIFU จะให้เอาท์พุทที่ขอบขาขึ้น (Leading edge) ของอินพุท และคำสั่ง DIFD จะให้เอาท์พุทที่ขอบขาลง (Failing edge) ของอินพุท



STEP	OPCODE	OPERAND
0020	LD	0000
0021	AND	0001
0022	DIFU(13)	0300
0023	DIFD(14)	0301



โอเปอร์แรนด์ของคำสั่งประกอบด้วย

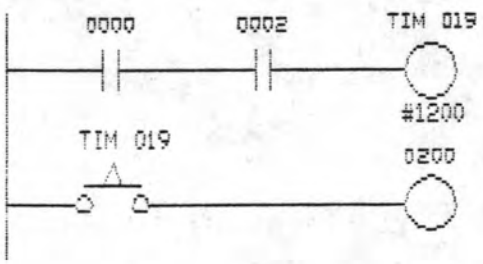
I/O, AUX relay : 0000-6015

Holding relay : HR 0000-3115

คำสั่ง DIFU และ DIFD ทั้งหมดภายในโปรแกรมมีได้ไม่เกิน 128 คำสั่ง

4.5.13 คำสั่ง HIGH-SPEED TIMER [TIMH(FUN15)]

คำสั่ง TIMH จะมีรูปแบบคำสั่ง และลักษณะการทำงานเหมือนคำสั่ง TIM ทุกอย่าง แต่คำสั่ง TIMH มีความละเอียดของเวลามากกว่าคือ มีหน่วยเป็น 0.01 วินาที

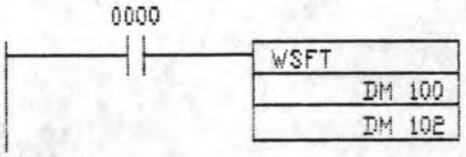


STEP	OPCODE	OPERAND
0500	LD	0000
0501	AND	0002
0502	TIM	019
		# 1200
0503	LD	TIM 019
0504	OUT	0200

หมายเลขของ TIMH มีค่าระหว่าง 000-127 ซึ่งใช้ร่วมกับคำสั่ง Timer และ Counter อื่น ๆ ผู้ใช้ต้องระวังไม่ให้ซ้ำกัน

4.5.14 คำสั่ง WORD SHIFT [WSFT(FUN16)]

คำสั่ง WSFT ใช้เลื่อนข้อมูลทีละแชลแนล เมื่อทำคำสั่งนี้แล้วค่าของแชลแนลเริ่มต้นจะเป็นศูนย์



STEP	OPCODE	OPERAND
0500	LD	0000
0501	WSFT(16)	-
		DM 100
		DM 102

เมื่อค่าของรีจิสเตอร์ R เป็น 1 (สถานะ ON) คำสั่ง WSFT จะทำงานทุก ๆ รอบการทำงาน ถ้าต้องการให้ทำงานหนึ่งรอบการทำงานต้องใช้คำสั่ง DIFU และ DIFD ช่วย

รูปแบบของคำสั่ง WSFT

```
WSFT  -
 -    D1 : Start channel
 -    D2 : End channel
```

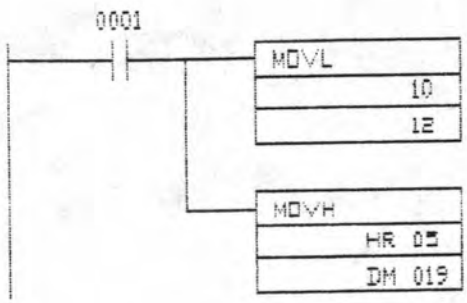
ชนิดของข้อมูลโอเปอร์แรนด์ประกอบด้วย

I/O, AUX channel	:	00-63
Holding relay	:	HR 00-31
Data memory	:	DM 000-511

4.5.15 คำสั่ง MOVE TO LOW BYTE [MOVL(FUN17)] และคำสั่ง MOVE TO HIGH BYTE [MOVH(FUN18)]

คำสั่ง MOVL เป็นการคัดลอกข้อมูลไบท์สูง (บิท 8-15) ไปยังไบท์ต่ำ (บิท 0-7) ตามตำแหน่งที่ระบุในโอเปอร์แรนด์

คำสั่ง MOVH เป็นการคัดลอกข้อมูลไบท์ต่ำ (บิท 0-7) ไปยังไบท์สูง (บิท 8-15) ตามตำแหน่งที่ระบุในโอเปอร์แรนด์



STEP	OPCODE	OPERAND
0500	LD	0001
0501	MOVL(17)	-
		00
		12
0502	MOVH(18)	-
		HR05
		DM019

รูปแบบของคำสั่ง MOVL/MOVH -

\- S1 : Transfer data

D1 : Tranfer destinating CH NO.

ชนิดข้อมูลของโอเปอร์แรนด์

	S1	D1
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Data memory	: DM 000-511	: DM 000-511

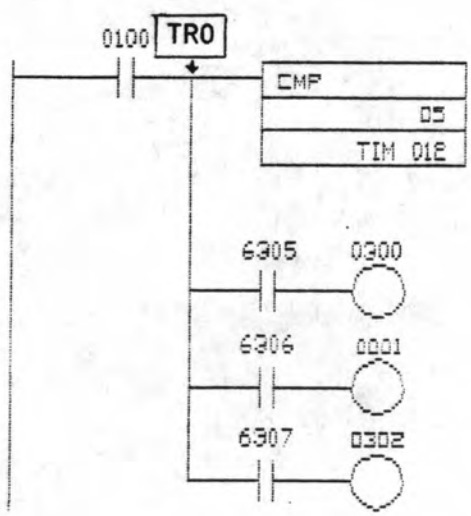
4.5.16 คำสั่ง COMPARE [CMP(FUN20)]

คำสั่ง CMP ใช้สำหรับเปรียบเทียบข้อมูล 2 แชลแนล ในลักษณะเลขไบนารี 16 บิท ซึ่งผลลัพธ์ของการเปรียบเทียบคือ

รีเลย์ 6305 จะมีสถานะ ON ถ้า S1 มีค่ามากกว่า S2

รีเลย์ 6306 จะมีสถานะ ON ถ้า S1 มีค่าเท่ากับ S2

รีเลย์ 6307 จะมีสถานะ ON ถ้า S1 มีค่าน้อยกว่า S2



STEP	OPCODE	OPERAND
0500	LD	0001
0501	OUT	TR0
0502	CMP(20)	-
		05
		TIM02
0503	LD	TR0
0504	AND	6305
0505	OUT	0300
0506	LD	TR0
0507	AND	6306
0508	OUT	0301
0509	LD	TR0
0510	AND	6307
0511	OUT	0302

รูปแบบคำสั่ง CMP -

S1 : Compare data 1

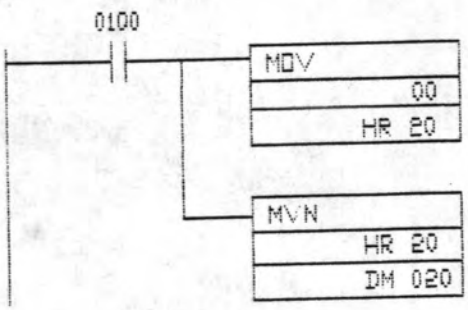
S2 : Compare data 2

ชนิดของข้อมูลโอเปอร์แรนด์ S1 และ S2

I/O, AUX relay	:	00-63
Holding relay	:	HR 00-31
Timer/counter	:	TIM/CNT 000-127
Data memory	:	DM 000-511
Constant	:	0000-FFFF

4.5.17 คำสั่ง MOVE[FUN21)] และ MOVE NOT [MVN(FUN21)]

คำสั่ง MOV เป็นการคัดลอกข้อมูลขนาด 16 บิทจากแชลแนลต้นทางไปยังแชลแนลปลายทาง โดยมีตำแหน่งระบุไว้ในโอเปอร์แรนด์ คำสั่ง MVN ทำงานเหมือนคำสั่ง MOV แต่จะ Inverse ข้อมูลก่อน



STEP	OPCODE	OPERAND
0200	LD	0100
0201	MOV	-
		00
		HR 20
0202	MVN	-
		HR 20
		DM 020

รูปแบบคำสั่ง MOV/MVN -

S : Transfer data

D : Destination channel

ชนิดของข้อมูลของโอเปอร์แรนด์

	S	D
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Timer/counter	: TIM/CNT 000-127	: -
Data memory	: DM 000-511	: DM 000-511
Constant	: 0000-FFFF	: -

4.5.18 คำสั่ง BCD-TO-BIN CONVERTION [BIN(FUN23)]

คำสั่ง BIN ใช้ในการแปลงข้อมูลฐานสิบขนาด 4 หลักไปเป็นเลขไบนารีขนาด 16 บิท ตำแหน่งต่าง ๆ ของข้อมูลระบุที่โอเปอร์แรนด์

รูปแบบคำสั่ง BIN -
- S : Conversion data
- D : Destination Channel

ชนิดข้อมูลของโอเปอร์แรนด์

	S	D1
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Timer/Counter	: TIM/CNT 000-127	: -
Data memory	: DM 000-511	: DM 000-511

ถ้าผลลัพธ์ของคำสั่ง BIN เป็นศูนย์สถานะของรีเลย์ 6306 จะเป็น ON
ถ้าข้อมูลอินพุทไม่ใช่เลข BCD คำสั่งนี้จะไม่ทำงาน และรีเลย์ 6303 จะมีสถานะ ON

4.5.19 คำสั่ง BIN-TO-BCD CONVERSION [BCD(FUN24)]

คำสั่ง BCD ใช้แปลงข้อมูลไบนารีขนาด 16 บิท ไปเป็นเลข BCD ขนาด 4 หลัก ตำแหน่งของข้อมูลต่าง ๆ ระบุที่โอเปอร์แรนด์

รูปแบบคำสั่ง BCD -

- S : Conversion data
- D : Destination channel

ชนิดข้อมูลของโอเปอร์แรนด์

	S	D1
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Data memory	: DM 000-511	: DM 000-511

ถ้าผลลัพธ์ของคำสั่งเป็นศูนย์รีเลย์ 6306 จะมีสถานะ ON

ถ้าผลลัพธ์ของคำสั่งมีค่ามากกว่า 9999 รีเลย์ 6303 จะมีสถานะ ON

4.5.20 คำสั่ง ASL (FUN25), ASR(FUN26), ROL(FUN27), ROR(FUN28) และ COM(FUN29)

รูปแบบคำสั่ง Function code -

- D : Data channel

ชนิดข้อมูลของโอเปอร์แรนด์

I/O, AUX relay	: 00-60
Holding relay	: HR 00-31
Data memory	: DM 000-511

การทำงานของคำสั่ง

ASL : Arithmetic shift left เป็นคำสั่งเลื่อนข้อมูลจากซ้ายไปขวาหนึ่งตำแหน่ง (บิท) เมื่อทำคำสั่งนี้แล้วข้อมูลบิท 0 จะมีค่าเป็น "0" และข้อมูลเดิมของบิท 15 จะไปเก็บที่รีเลย์ตัวทด (6304)

ASR : Arithmetic shift right เป็นคำสั่งเลื่อนข้อมูลจากขวาไปซ้ายหนึ่งตำแหน่ง (บิท) เมื่อทำคำสั่งนี้แล้วข้อมูลบิท 15 จะมีค่าเป็น "0" และข้อมูลเดิมของบิท 0 จะนำไปเก็บที่รีเลย์ตัวทด (Carry)

ROL : Rotate left เป็นคำสั่งหมุนข้อมูลผ่านรีเลย์ตัวทด (6304) ในทิศทางซ้ายมือ ซึ่งจะทำให้ข้อมูลบิท 15 เลื่อนไปอยู่ในรีเลย์ตัวทด (6304) และข้อมูลรีเลย์ตัวทดจะเลื่อนไปยังบิท 0

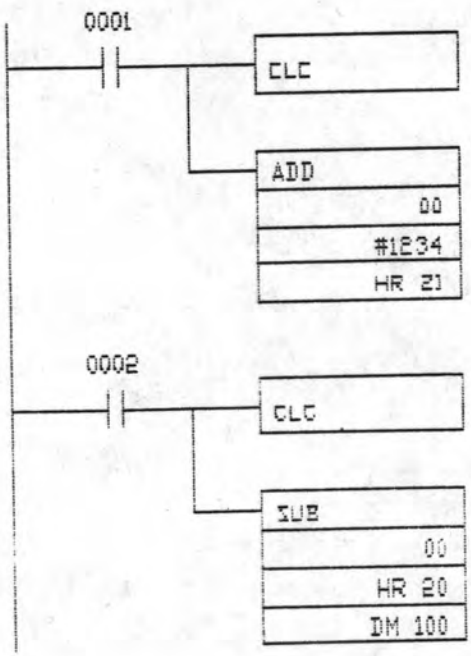
ROR : Rotate right เป็นคำสั่งหมุนข้อมูลทางขวาผ่านรีเลย์ตัวทด (6304) การทำงานจะทำให้ข้อมูลตัวทด (6304) เลื่อนไปยังบิท 15 และข้อมูลบิท 0 เลื่อนไปยังรีเลย์ตัวทด

COM : Complement เป็นคำสั่งทำ Inverse ข้อมูล โดยเปลี่ยนค่าทุกบิทของข้อมูลจาก 0 เป็น 1 และจาก 1 เป็น 0

4.5.21 คำสั่ง ADD (FUN30) และ SUBTRACT [SUB(FUN31)]

คำสั่ง ADD เป็นการบวกเลข BCD 4 หลัก 2 จำนวน โดยจำนำค่า S1 มาบวกกับ S2 และบวกรีเลย์ตัวทด (6304) นำผลลัพธ์ไปเก็บที่ D

คำสั่ง SUB เป็นการลบเลข BCD 4 หลัก 2 จำนวน โดยนำค่า S1 มาลบด้วย S2 และลบกับรีเลย์ตัวทด (6304) นำผลลัพธ์ไปเก็บที่ D



STEP	OPCODE	OPERAND
0500	LD	0001
0501	CLC	-
0502	ADD	-
		00
		# 1234
		HR 21
0503	LD	0002
0504	CLC	-
0505	SUB	-
		00
		HR 20
		DM 100

รูปแบบคำสั่ง ADD/SUB -

\- S1 : Data 1

\- S2 : Data 2

\- D : Result channel

ชนิดข้อมูลของโอเปอร์แรนด์

	S1, S2	D
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Timer/counter	: T/C 000-127	: -
Data memory	: DM 000-511	: DM 000-511
Constant	: 0000-9999	: -

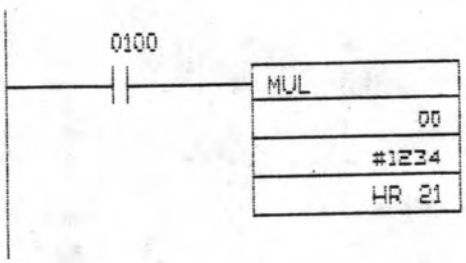
ก่อนทำคำสั่งซีพียูจะตรวจสอบข้อมูลถ้าไม่ใช่เลข BCD รีเลย์ 6303 จะมีสถานะ ON และคำสั่งนี้จะไม่ทำงาน

ถ้าผลลัพธ์ของการทำคำสั่ง ADD หรือ SUB เป็นศูนย์รีเลย์ 6306 จะมีสถานะ ON

ถ้าผลลัพธ์ของคำสั่ง ADD มากกว่า 9999 หรือผลลัพธ์ของคำสั่ง SUB มีค่าน้อยกว่า 0000 รีเลย์ 6304 (Carry) จะมีสถานะ ON

4.5.22 คำสั่ง MULTIPLY [MUL(FUN 32)]

MUL เป็นการคูณเลข BCD 4 หลักที่เก็บใน S1 และ S2 ผลลัพธ์ที่ได้เป็น BCD ขนาด 8 หลัก (2 Channel) ซึ่งถูกเก็บไว้ที่ D และแซลแนลถัดไป



STEP	OPCODE	OPERAND
0000	LD	0100
0001	MUL	-
		00
		# 1234
		HR 21

รูปแบบคำสั่ง MUL -

- S1 : Data 1
- S2 : Data 2
- D : Result channel

ชนิดข้อมูลของโอเปอร์แรนด์

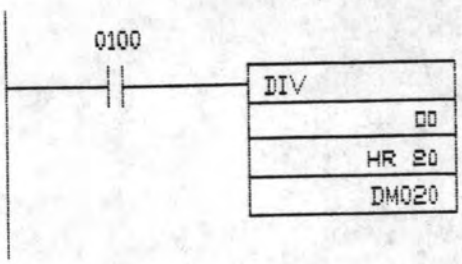
	S1, S2	D
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Timer/counter	: T/C 000-127	: -
Data memory	: DM 000-511	: DM 000-511
Constant	: 0000-9999	: -

ถ้าผลลัพธ์ของการคำนวณเป็น 0 รีเลย์ 6306 จะมีสถานะ ON

ถ้าข้อมูลที่นำมาคูณไม่ใช่เลข BCD คำสั่ง MUL จะไม่ทำงาน และรีเลย์ 6303 จะ ON

4.5.23 คำสั่ง DIVIDE [DIV (FUN 33)]

คำสั่ง DIV จะนำค่า S1 หารด้วยค่าของ S2 นำผลลัพธ์ไปเก็บที่ D และนำเศษของการหารไปเก็บที่ตำแหน่งถัดไป



STEP	OPCODE	OPERAND
0000	LD	0100
0001	DIV	-
		00
		HR 20
		DM 020

รูปแบบคำสั่ง DIV -

\- S1 : Data 1

\- S2 : Divide data

\- D : Result channel

ชนิดข้อมูลของโอเปอร์แรนด์

	S1, S2	D
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Timer/counter	: T/C 000-127	: -
Data memory	: DM 000-511	: DM 000-511
Constant	: 0000-9999	: -

ซีพียูจะตรวจสอบข้อมูล S1 และ S2 ถ้ามีความผิดพลาดจะเซ็ทรีเลย์ 6303 ให้ ON และไม่ทำคำสั่ง DIV นี้

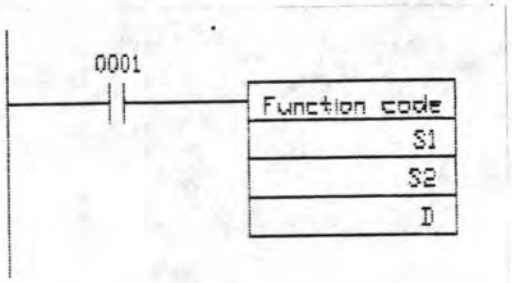
4.5.24 คำสั่ง ANDW (FUN 34), ORW (FUN 35), XORW (FUN 36), XNRW (FUN 37)

คำสั่งเหล่านี้เป็นคำสั่งเกี่ยวกับการทำลอจิกของข้อมูล

รูปแบบของคำสั่ง Function code -
- S1 : Data 1
- S2 : Data 2
- D : Result channel

ชนิดข้อมูลของโอเปอร์แรนด์

	S1, S2	D
I/O, AUX relay	: 00-63	: 00-60
Holding relay	: HR 00-31	: HR 00-31
Timer/counter	: T/C 000-127	: -
Data memory	: DM 000-511	: DM 000-511
Constant	: 0000-FFFF	: -



ANDW : And word เป็นการทำลอจิก AND ระหว่าง S1 และ S2 ผลลัพธ์ไปเก็บที่ D

ORW : Or word เป็นการทำลอจิก OR ระหว่าง S1 และ S2 ผลลัพธ์ไปเก็บที่ D

XORW : Exclusive word เป็นการทำลอจิก XOR ระหว่าง S1 และ S2 ผลลัพธ์ไปเก็บที่ D

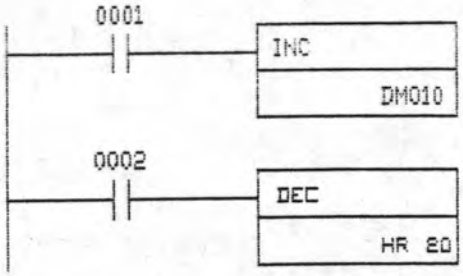
XNRW : Exclusive or not word เป็นการทำลอจิก XNRW ระหว่าง S1 และ S2

ถ้าผลลัพธ์ของการทำคำสั่งเหล่านี้เป็น 0 รีเลย์ 6306 จะเซ็ทเป็น ON

4.5.25 คำสั่ง INCREMENT [INC(FUN 38)] และคำสั่ง DECREMENT [DEC (FUN 39)]

คำสั่ง INC จะเพิ่มค่าของข้อมูลอีกหนึ่ง คำสั่ง DEC เป็นการลดค่าของข้อมูลลงหนึ่ง ถ้าผลลัพธ์การทำงานของคำสั่ง INC หรือ DEC เป็นศูนย์รีเลย์ 6306 จะถูกเซ็ทเป็น ON

ในการทำงานซีพียูจะตรวจสอบค่าข้อมูล ถ้าไม่ใช่รหัส BCD จะเซ็ทรีเลย์ 6303 เป็น ON และจะไม่ทำคำสั่งนี้ด้วย



STEP	OPCODE	OPERAND
0000	LD	0001
0001	INC	-
		DM 010
0002	DEC	-
		HR 20

รูปแบบคำสั่ง INC/DEC -

\- D : Data channel

ชนิดข้อมูลของโอเปอร์แรนด์

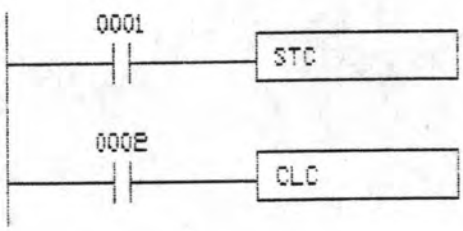
I/O, AUX relay : 00-60

Holding relay : HR 00-31

Data memory : DM 000-511

4.5.26 คำสั่ง SET CARRY [STC(FUN 40)] และคำสั่ง CLEAR CARRY [CLC (FUN 41)]

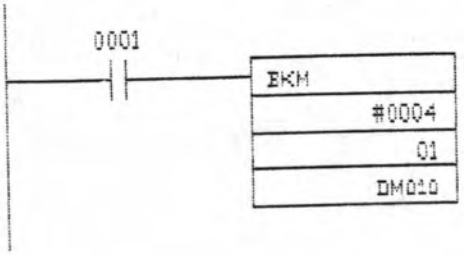
ใช้คำสั่ง STC สำหรับเซ็ทรีเลย์ตัวทด (6304) ให้เป็น "1" และคำสั่ง CLC สำหรับรีเซ็ทรีเลย์ตัวทด (6304) ให้เป็น "0" คำสั่งเหล่านี้จะใช้ร่วมกับคำสั่งการคำนวณ



STEP	OPCODE	OPERAND
0100	LD	0001
0101	STC	-
0102	LD	0002
0103	CLC	-

4.5.27 คำสั่ง BLOCK MOVE [BKM(FUN 42)]

เป็นคำสั่งย้ายข้อมูลครั้งละหลาย ๆ แชลแนล



STEP	OPCODE	OPERAND
0000	LD	0001
0001	BKM (42)	-
		# 0004
		01
		DM 010

รูปแบบคำสั่ง BKM -

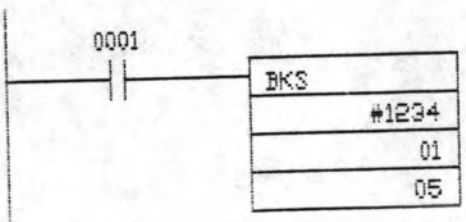
- W : No. of channel
- S : Start channel
- D : Destination channel

ชนิดข้อมูลของโอเปอร์แรนด์

	W	S	D
I/O, AUX relay	-	00-63	00-60
Holding relay	-	HR 00-31	HR 00-31
Timer/counter	-	T/C 000-127	T/C 000-127
Data memory	-	DM 000-511	DM 000-511
Constant	0000-0511	-	-

4.5.28 คำสั่ง BLOCK SET [BKS(FUN 43)]

เป็นคำสั่งกำหนดค่าของข้อมูลครั้งละหลาย ๆ แชลแนล



STEP	OPCODE	OPERAND
0000	LD	0001
0001	BKS	-
		# 1234
		01
		05

รูปแบบคำสั่ง BKS -

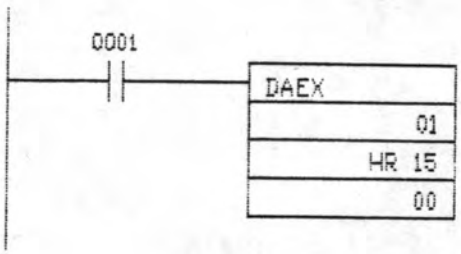
- S : Setting data
- D1 : Start channel
- D2 : End channel

ชนิดข้อมูลของโอเปอร์แรนด์

	S	D1, D2
I/O, AUX relay	00-63	00-60
Holding relay	HR 00-31	HR 00-31
Timer/counter	T/C 000-127	T/C 000-127
Data memory	DM 000-511	DM 000-511
Constant	0000-FFFF	-

4.5.29 คำสั่ง DATA EXCHANGE [DAEX(FUN 44)]

คำสั่ง DAEX เป็นสลับข้อมูลของแชลแนลหนึ่งกับแชลแนลอื่น



STEP	OPCODE	OPERAND
0000	LD	0001
0001	DAEX	-
		01
		HR 15
		00

รูปแบบคำสั่ง DAEX -

\- D1 : Data 1

\- D2 : Data 2

\- -

ชนิดข้อมูลของโอเปอร์แรนด์

I/O, AUX relay : 00-60

Holding relay : HR 00-31

Data memory : DM 000-511

4.5.30 คำสั่ง ONE DIGIT SHIFT LEFT [ODSL(FUN 45)] และ คำสั่ง ONE DIGIT SHIFT RIGHT [ODSR(FUN 46)]

รูปแบบคำสั่ง ODSL/ODSR -

\- D1 : Start channel

\- D2 : End channel

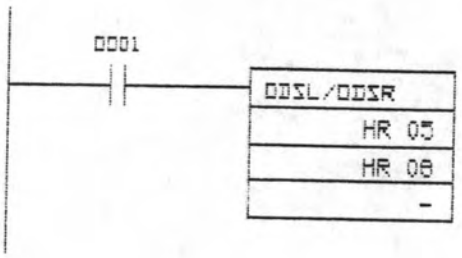
\- -

ชนิดข้อมูลของโอเปอร์แรนด์ D1, D2

I/O, AUX relay : 00-60

Holding relay : HR 00-31

Data memory : DM 000-511



คำสั่ง ODSL จะเป็นการเลื่อนข้อมูลระหว่าง D1 ถึง D2 ไปทางซ้ายมือครั้งละ 4 บิท และนำค่า 0000 ใส่ที่บิท 0-3 ของ D1

คำสั่ง ODSR เป็นการเลื่อนข้อมูลระหว่าง D1 ถึง D2 ไปทางขวามือครั้งละ 4 บิท และนำค่า 0000 ใส่ที่บิท 15-12 ของ D2

4.5.31 คำสั่ง 4-TO-16 DECODER [DECO(FUN 47)], คำสั่ง 16-TO-4 ENCODER [ENCO(FUN 48)], คำสั่ง 7-SEGMENT DECODER [7 SEG(FUN 49)]

รูปแบบคำสั่ง Function code -

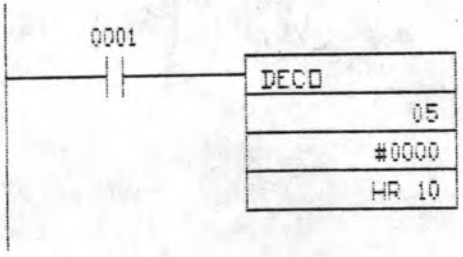
S : Conversion data

K : Digit No.

D : Result channel

ชนิดข้อมูลของโอเปอร์แรนด์

	S	K	D
I/O, AUX relay	00-63	-	00-60
Holding relay	HR 00-31	-	HR 00-31
Timer/counter	T/C 000-127	-	T/C 000-127
Data memory	DM 000-511	-	DM 000-511
Constant	-	0000-0003	-



คำสั่ง DECO เป็นการดีโค้ดค่าไบนารีของข้อมูล S ณ หลัก (Digit) ที่กำหนดโดย K และเก็บผลลัพธ์ไว้ที่ D

คำสั่ง ENCO เป็นการหาค่าตำแหน่งบิทสูงสุดของข้อมูล "1" ของ S แล้วแปลงค่าเป็นเลขไบนารีเก็บลงที่ D ณ หลัก (Digit) ที่กำหนดโดย K

คำสั่ง 7SEG เป็นการแปลงค่าไบนารีของข้อมูล S ณ หลัก (Digit) ที่กำหนดโดย K ไปเป็นรหัส 7 Segment เก็บไว้ที่ไบท์ต่ำของ D