เครื่องควบคุมดีซีมอเตอร์

(Wireless Control Motor)

โดย

นายกิตติ อภิรัตนประภาส

นายเพียรเลิศ จันทร์เอี่ยม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตร์บัณฑิต

สาขาเทคโนโลยีอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปีการศึกษา 2542

เครื่องควบคุมดีซีมอเตอร์

(Wireless Control Motor)

โดย

นายกิตติ อภิรัตนประภาส

นายเพียรเลิศ จันทร์เอี่ยม

อาจารย์ที่ปรึกษา

อ.มนชนก ศรีเสือขาม

คำนำ

รายงานฉบับนี้ได้จัดทำขึ้นเพื่อช่วยอธิบายรายละเอียดที่เกี่ยวกับรีโมทคอนโทรล ซึ่งจะกล่าวถึงตั้งแต่วงจรเบื้องต้นจนไปถึงวงจรสำเร็จรูปที่สามารถนำไปใช้งานได้ทันที และหลักการออกแบบให้ได้วงจรตามที่ต้องการ ซึ่งผู้จัดทำได้นำหลักการที่ได้นี้มาประยุกต์และพัฒนาระบบฮาร์ดแวร์ประกอบในการทำชิ้นงาน ถ้าภายในเนื้อหาของรายงายฉบับนี้ขาดหายหรือมีข้อผิดพลาดประการใด ผู้จัดทำต้องขออภัยมา ณ ที่นี้ด้วย

ผู้จัดทำ

นายกิตติ อภิรัตนประภาส

นายเพียรเลิศ จันทร์เอี่ยม

สารบัญ

สารบัญภาพ

สารบัญตาราง

เครื่องควบคุมดีซีมอเตอร์แบบไร้สาย

นายกิตติ อภิรัตนประภาส
นายเพียรเลิศ จันทร์เอี่ยม
อาจารย์มนชนก ศรีเสือขาม (อาจารย์ที่ปรึกษา)

บทคัดย่อ

ภายในวิทยานิพนธ์ฉบับนี้ อธิบายถึงโครงสร้างเครื่องควบคุมไร้สายด้วยแสงอินฟราเรด และการเขียนโปรแกรมลงบนไอซี ให้ทำงานด้วยกัน โดยเครื่องควบคุมไร้สายจะประกอบด้วยภาคส่งและภาครับและจากภาครับจะส่งรหัสควบคุมที่ได้จากการกดสวิทช์ที่ภาคส่งมาที่ชุดควบคุมในแต่ละช่อง ต่อจากนั้นชุดควบคุมจะทำหน้าที่เป็นสวิทช์เพื่อเลือกการทำงานของ Stepper motor โดยที่มีไมโครคอนโทรลเลอร์ PIC16F84 เป็นตัวขับ Stepper motor ซึ่งโปรแกรมที่เขียนลงบนตัวไมโครคอนโทรลเลอร์ PIC16F84 จะสั่งให้ Stepper motor ทำงาน

Abstract

Inside this thesis present an infrared remote controller and write the program into IC . This infrared remote controller compose of transmitter and receiver . Receiver unit will send code to the channel of the control unit . Afterthat control unit will be switched function for select function of Stepper motor . The stepper motor can be drived by microcontroller PIC16F84 for the program is writed into microcontroler PIC16F84 will control stepper motor .

บทที่ 1

บทนำ

การส่งสัญญาณควบคุมในปัจจุบันมีอยู่ 2 รูปแบบ คือ แบบมีสาย และ แบบไร้สาย ซึ่งจะกล่าวในแบบไร้สาย โดยในสมัยก่อนจะมีการส่งสัญญาณที่ใช้ ทรานซิสเตอร์หลายตัวประกอบเข้าด้วยกันเป็นวงจร แต่ในปัจจุบันได้มีการผลิตไอซีที่มีหลายวงจรรวมอยู่ใน IC ทำให้สะดวกและราคาถูก ประหยัดเนื้อที่เนื่องจากมีขนาดเล็ก

ภาครับและภาคส่งจะมีการใช้สัญญาณนาฬิกาในการสร้างและรับสัญญาณ ซึ่งวงจรที่ผลิตนี้เรียกว่า " วงจรออสซิลเลเตอร์ " โดยที่ทางภาครับจะต้องสร้างความถี่มาให้ตรงกับทางภาคส่ง การปรับให้ความถี่ทั้งสองนี้เท่ากันนั้นทำได้ไม่ยาก

เมื่อเราได้สัญญาณจากภาครับจะสามารถนำไปควบคุมอุปกรณ์ไฟฟ้าต่างๆ ที่ต้องการได้ โดยการนำสัญญาณที่ได้จากภาครับนั้นมาต่อร่วมกับวงจรชุดควบคุม เพื่อนำไปตัดต่อโดยสวิตช์อัตโนมัติเป็นตัวเลือกทำงานในหน้าที่ต่างๆ กัน เช่น นำไปควบคุมการหมุนของมอเตอร์ไฟฟ้าที่ใช้ขับผ้าม่านให้เปิดและปิด , การเพิ่มความเร็วของมอเตอร์โดยรีโมตคอนโทรล เป็นต้น

วัตถุประสงค์

1. อธิบายถึงหลักการทำงานของรีโมตคอนโทรลได้
2. สามารถแก้ไขปัญหาที่เกิดจากการทำงานผิดพลาดของรีโมตคอนโทรลได้
3. ปรับปรุงและพัฒนารีโมตคอนโทรลให้มีประสิทธิสูงขึ้นและเท่าเทียมกับปัจจุบัน

บทที่ 2
ทฤษฎีและหลักการ

วงจรเข้ารหัส (Encode circuit)

ในโครงงานนี้เป็นเรื่องที่เกี่ยวกับ Logic ดังนั้นจึงเป็นการเปลี่ยนระดับของ Logic ของสวิทช์ทาง Input มาเป็นสัญญาณ Logic ตามรหัสที่เราต้องการออกมาทาง Output ซึ่งเราสามารถให้ Output ของวงจรเป็นรหัสอะไรก็ขึ้นอยู่กับความต้องการ

วงจรถอดรหัส (Decode circuit)

เป็นวงจรที่เปลี่ยนรหัสทาง Input เพื่อเป็นรหัสอื่นตามต้องการเช่น ถ้า Input เป็นรหัส BCD ก็จะเปลี่ยนเป็นแรงดันของเลขฐานสิบ

วงจร Multiplex

เป็นวงจรที่ส่งข้อมูลทาง Digital หลายข้อมูลออกไปบนสายส่งเดียวกัน

Shift Register

เป็นกลุ่มของ Flip Flop ที่ทำหน้าที่เลื่อนข้อมูลทาง Binary ไปทางซ้ายหรือทางขวาก็ได้ซึ่งอาจจะเลื่อนข้อมูลเป็นชุดในลักษณะขนานก็ได้เช่นกัน เราสามารถแบ่งรูปแบบการ Shift ออกได้เป็น

-Shift Left Register

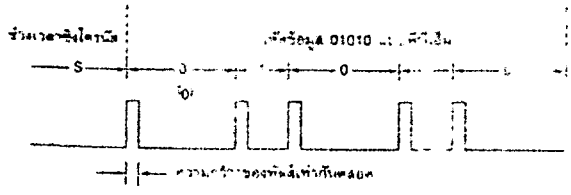
-Parallel Data Transfer

-Parallel in-Serial out Shift Register

-Parallel in-Parallel out Shift Register

สัญญาณพีพีเอ็ม (Pulse Position Modulation)

รูปแบบสัญญาณชนิด PPM เกิดจากการมอดูเลตสัญญาณในลักษณะของตำแหน่งพัลส์ กล่าวคือขนาดความกว้างของสัญญาณพัลส์จะมีค่าเท่ากันตลอดและไม่มีความสำคัญในการบอกชนิดของข้อมูลเลย แต่จะใช้คาบเวลาหรือพีเรียด (period) ของพัลส์แต่ละลูกเป็นตัวกำหนดชนิดของข้อมูล เช่น ข้อมูลที่เป็น "1" แทนด้วยพัลส์ที่มีคาบเวลาคงที่ค่าหนึ่ง ซึ่งแตกต่างจากคาบเวลา



ของพัลส์ที่แสดงข้อมูลที่เป็น "0" ดังแสดงในรูปด้านล่าง

โดยหลักการแล้วการมอดดูเลตสัญญาณแบบพีพีเอ็มจะใช้การแบ่งช่วงสัญญาณด้วยคาบเวลาที่เท่ากันแต่จุดเวลาที่แสดงสัญญาณพัลส์ต่างกัน เช่น หากสัญญาณเป็นศูนย์ สัญญาณพัลส์จะปรากฏ ณ ตำแหน่งกึ่งกลางของคาบเวลาที่กำหนดถ้าหากสัญญาณมีแอมปลิจูดเป็นบวก สัญญาณพัลส์จะปรากฏในตำแหน่งที่ล้ำไปทางขวาของจุดกึ่งกลางของคาบเวลา โดยมีระยะห่างขึ้นอยู่กับค่าของแอมปลิจูดในลักษณะเป็นสัดส่วนกัน

ถ้าหากสัญญาณแอมปลิจูดเป็นลบสัญญาณพัลส์จะปรากฏอยู่ในช่วงครึ่งแรกของคาบเวลา ดังนั้นการมอดดูเลตแบบพีพีเอ็มจึงสามารถใช้ได้ทั้งสัญญาณที่เป็นอะนาลอกและดิจิตอล

รีโมตอินฟราเรด 15 ช่อง

จากตารางที่ 1 แสดงการเปรียบเทียบเลขฐานสอง โดยให้เลขฐานสิบกับเลขฐานสอง โดยให้เลขฐานสองมีขนาด 4 หลัก หรือ 4 บิต จะได้เลข 0-15 ของบานสิบ ซึ่งเท่ากับ 16 ช่อง แต่ ช่องเลข 0 ไม่ใช้งาน เนื่องจากสภาวะ " 0 " ทั้ง 4 บิตจะไม่เกิดการเปลี่ยนแปลงอินพุตของวงจร ดังนั้นสัญญาณที่จะใช้ควบคุมจึงมีเพียง 15 ช่องเท่านั้น สัญญาณที่จะส่งออกไปแต่ละช่องอาศัยความแตกต่างของความกว้างของพัลส์แยกสภาวะ " 0 " กับ " 1 "

แนะนำไอซี

วงจรนี้ใช้ไอซี 3 ตัว ซึ่งถูกผลิตมาให้ใช้งานร่วมกันได้คือ SL490B สำหรับภาคส่ง , SL486 และ ML926 เป็นภาครับและขยายสัญญาณ

SL490B เป็นไอซีสำหรับเข้ารหัสเมื่อกดคีย์ โดยส่งออกไปเป็นลักษณะของพัลส์โพซิชั่น มอดูเลชั่น หรือ พีพีเอ็ม (Pulse Position Modulation , PPM) พีพีเอ็มมีลักษณะเปลี่ยนไปตามโค้ดที่ได้จากโค้ดรีจิสเตอร์ ถ้าเป็นการส่งย่านอัลตร้าโซนิกจะใช้อัลตร้าโซนิกเป็นตัวทรานสดิวเซอร์เป็นตัวปล่อยพัลส์ และใช้ LED แทนเมื่อส่งย่านอินฟราเรด โครงสร้างภายในของ SL490B แสดงในรูปที่ 1

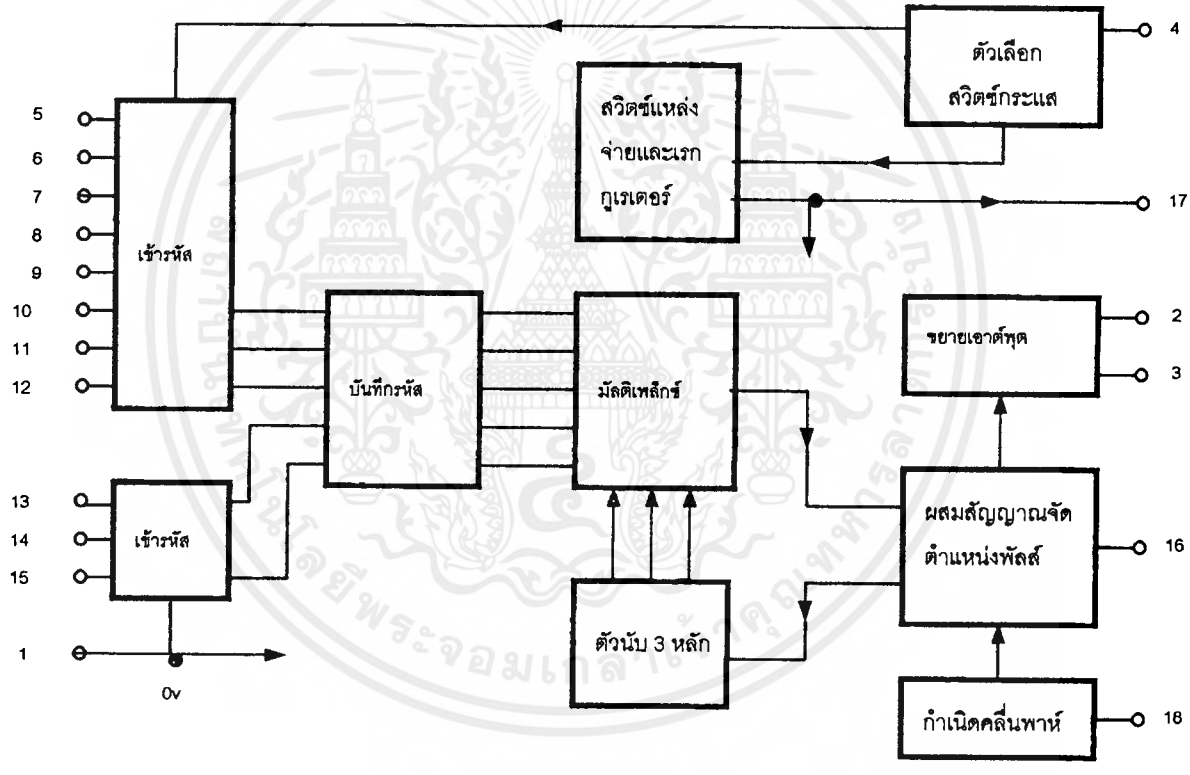
สัญญาณเอาท์พุตที่ขา 2 , 3 ได้มาจาก การผสมสัญญาณระหว่างความถี่ของคลื่นพาห์กับสัญญาณมัลติเพล็กซ์ได้เป็นสัญญาณพีพีเอ็มออกมาดังรูปที่ 2 โดย t_1 แสดงสภาวะ " 1 " ซึ่งแคบกว่า t_0 ที่แสดงสภาวะ " 0 " ส่วน t_g จะแบ่งแยก สัญญาณพีพีเอ็มแต่ะชุดออกจากกัน รูปนี้เป็นสัญญาณพีพีเอ็ม สัญญาณเลขฐาน 22 ของฐานสิบ เพราะไอซีตัวนี้สามารถผลิตสัญญาณเลขฐานสองได้ 5 บิต คือ 32 ช่องสัญญาณ รูปที่ 3 แสดงขาใช้งานแต่ละขา

SL486 เป็นภาครับสัญญาณเข้ามาโดยผ่านไดโอดรับคลื่น แล้วทำการขยายสัญญาณพีพีเอ็มที่รับเข้ามาส่งก่อนไป ML926

ML926 จะรับสัญญาณพีพีเอ็มเข้ามาแยกแล้วแปลงออกมาเป็นสัญญาณเลขฐานสองออกมาทางขา 5 , 6 , 7 และ 8 โครงสร้างของ ML926 ดูได้จากรูปที่ 4

หลักการทำงาน

รูปที่ 5 เป็นบล็อกไดอะแกรมของทั้งสองระบบ มีภาคส่งสัญญาณพีพีเอ็มออกมาแล้วภาครับจะรับสัญญาณมาขยายก่อนเปลี่ยนเป็นสัญญาณเลขฐานสองเพื่อนำไปใช้งานต่อไป รูปที่ 6 เป็นวงจรภาคเครื่องส่ง



รูปที่ 1 แสดงบล็อกไดอะแกรมของ SL490B

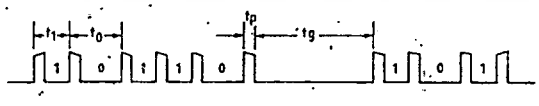
วงจรภาคส่งใช้ไฟ 9 โวลต์ ป้อนเข้าขา 4 ของ SL490B และเป็นไฟเลี้ยงให้ Q_1 , Q_2 เมื่อขา 2 ให้สัญญาณเอาท์พุตเป็นไบแอสสำหรับ Q_1 เพื่อขับ Q_2 ให้ทำงาน LED_1 , LED_2 จะส่งพัลส์อินฟราเรดออกมา โดยมี R_1 , R_2 , C_1 , C_2 เป็นตัวกำหนดความถี่คลื่นพาห์

วงจรภาครับในรูปที่ 7 ใช้แหล่งจ่ายไฟ 15 โวลต์ เลี้ยง SL486 , ML926 และ CD4515 มีไดโอด RS302 เป็นตัวรับแสงอินฟราเรด ซึ่งต่อเข้ากับ ขา 1 , 16 ของ SL486 ซึ่งจะดีดค้ดและขยายสัญญาณพีพีเอ็ม แล้วส่งเข้าอินพุต ขา 3 ของ ML926

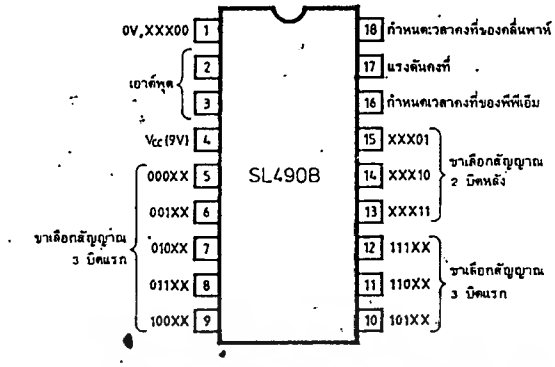
VR_1 100 kΩ ปรับค่าได้ใช้สำหรับปรับความถี่ออสซิลเลเตอร์ของ IC_2 ให้ตรงกับอินพุตที่รับเข้า ส่วนขา 5 , 6 , 7 ,8 ของ IC_2 เป็นเอาท์พุตสัญญาณเลขฐานสอง ซึ่งจะแสดงผลผ่าน LED_1 - LED_4 สำหรับ IC_3 CD4515 (หรือ CD4515) รับสัญญาณเข้ามาถอดรหัสให้ได้เอาท์พุต 15 ช่อง ในกรณีของ CD4515 เมื่อยังไม่มีการทำงานเอาท์พุตทั้งหมดจะเป็น " 0 " เวลากดสวิทช์ช่องใดช่องหนึ่ง ช่องนั้นจะมีเอาท์พุตเป็น " 1 " ส่วน CD4515 จะเป็นไปในทางตรงข้าม แต่ผลลัพธ์ในการควบคุมเหมือนกัน

ฐานสิบ	ฐานสอง	พีพีเอ็ม
0	0000	
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
10	1010	
11	1011	
12	1100	
13	1101	
14	1110	
15	1111	

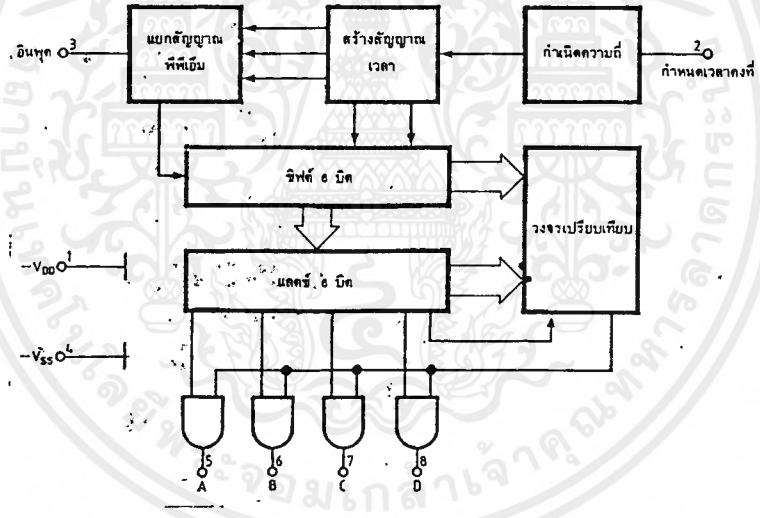
ตารางที่ 1 เปรียบเทียบหมายเลขช่องในฐาน 10 และฐาน 2 กับสัญญาณพีพี่เอ็ม



รูปที่ 2 สัญญาณ PPM ที่ส่งออกมา



รูปที่ 3 แสดงการใช้งานแต่ละขาของ SL490B



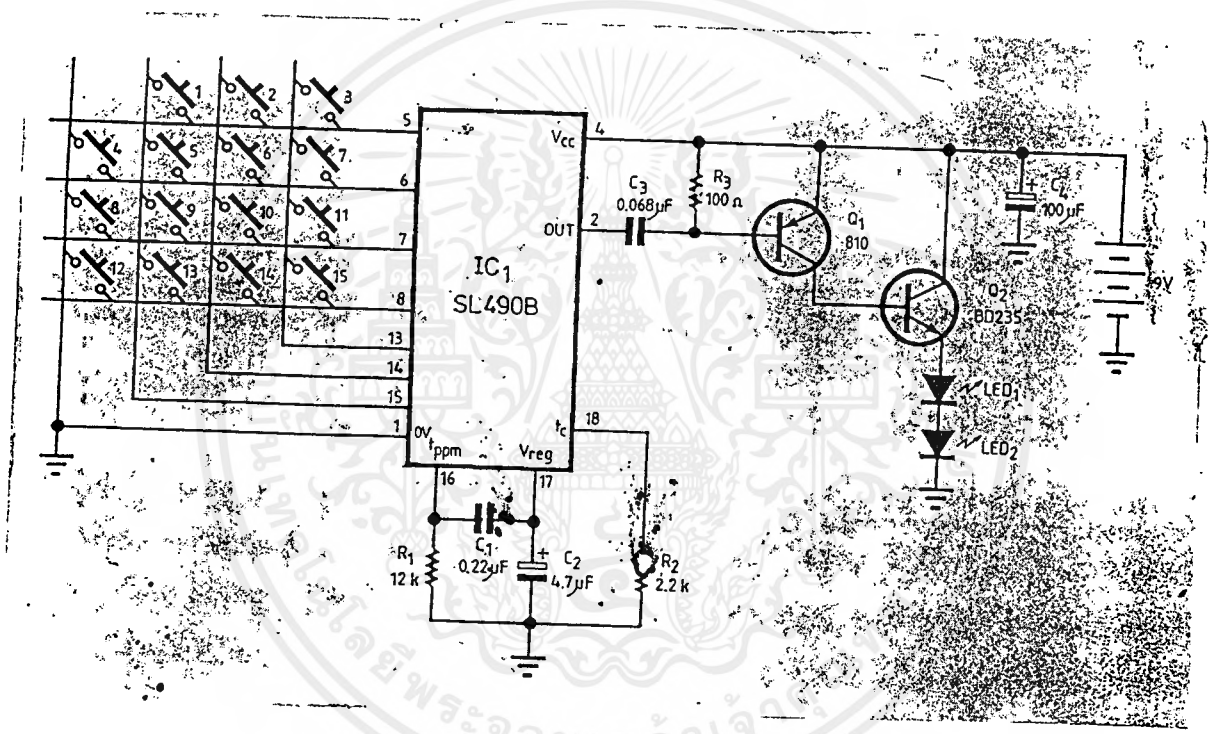
รูปที่ 4 แสดงบล็อกไดอะแกรมของ ML926

ตรงอินพุตของ IC_3 นี้ มีสภาวะเป็น " 0 " หมดทั้ง 4 อินพุต เมื่อยังไม่กดสวิทช์เลือกช่อง ถ้าสมมติว่าเรามีสวิทช์สำหรับเลือกช่องโดยให้สัญญาณเลขฐานสองเป็น " 0000 " จะมีสัญญาณพีพีเอ็มมาจากเครื่องส่ง ผ่านภาครับภาคขยาย พอออกจาก ML926 เอาท์พุตที่ได้จะเป็นสภาวะ " 0000 " ซึ่งเป็นสถาพปกติของอินพีตของ IC_3 อยู่แล้ว ดังนั้นจึงไม่มีอไรเกิดขึ้นทำให้เราไม่สามารถใช้งาน ช่อง 0 เป็นสัญญาณควบคุมได้นั่นเอง

เป็นวงจรภาคขับรีเลย์ใช้ CD4013 ซึ่งเป็นไอซีฟลิปฟลอป สามารถขับรีเลย์ได้ 2 ตัว ช่องสัญญาณ 15 ช่อง ใช้ CD4013 ทั้งหมด 8 ตัว เมื่อสัญญาณอินพุตช่อง 1 ถูกส่งเข้ามา (ขา 3 ของ

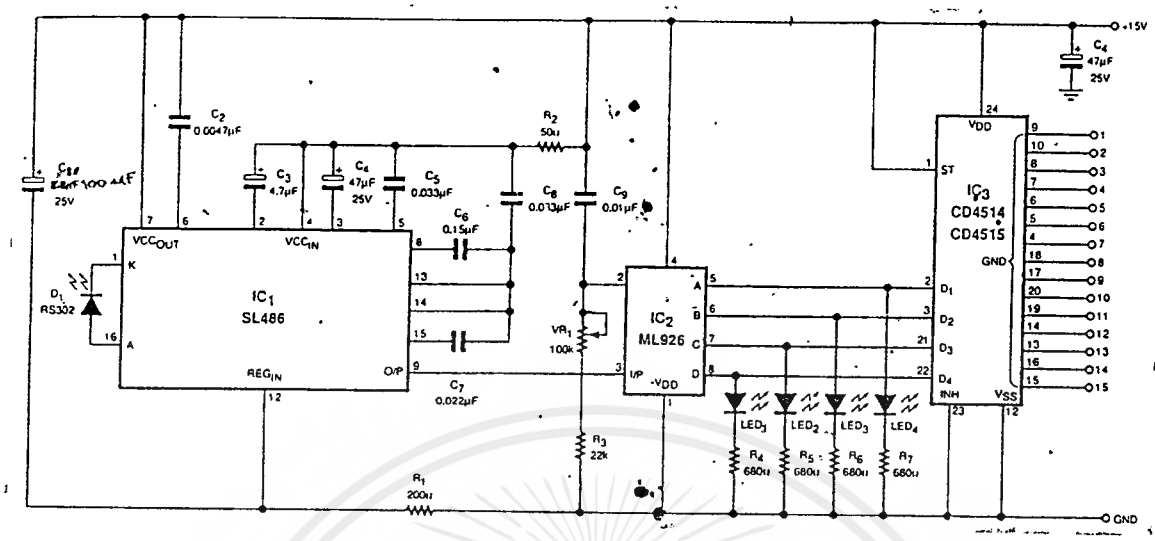
CD4013) สัญญาณเอาท์พุตของช่องที่ 1 ขา 1 ของ CD4013) ก็จะให้ไบแอสแก่ Q_1 เบอร์ 2SC930 ทำให้รีเลย์ 1 ทำงาน LED_1 จะสว่างเพื่อแสดงว่ารีเลย์ 1 ทำงานอยู่ช่องอื่นๆก็เช่นเดียวกัน C_1 , C_2 ใส่เพื่อป้องกันสัญญาณรบกวน

รูปที่ 5 แสดงหลักการทำงานเบื้องต้นของรีโมตคอนโทรล

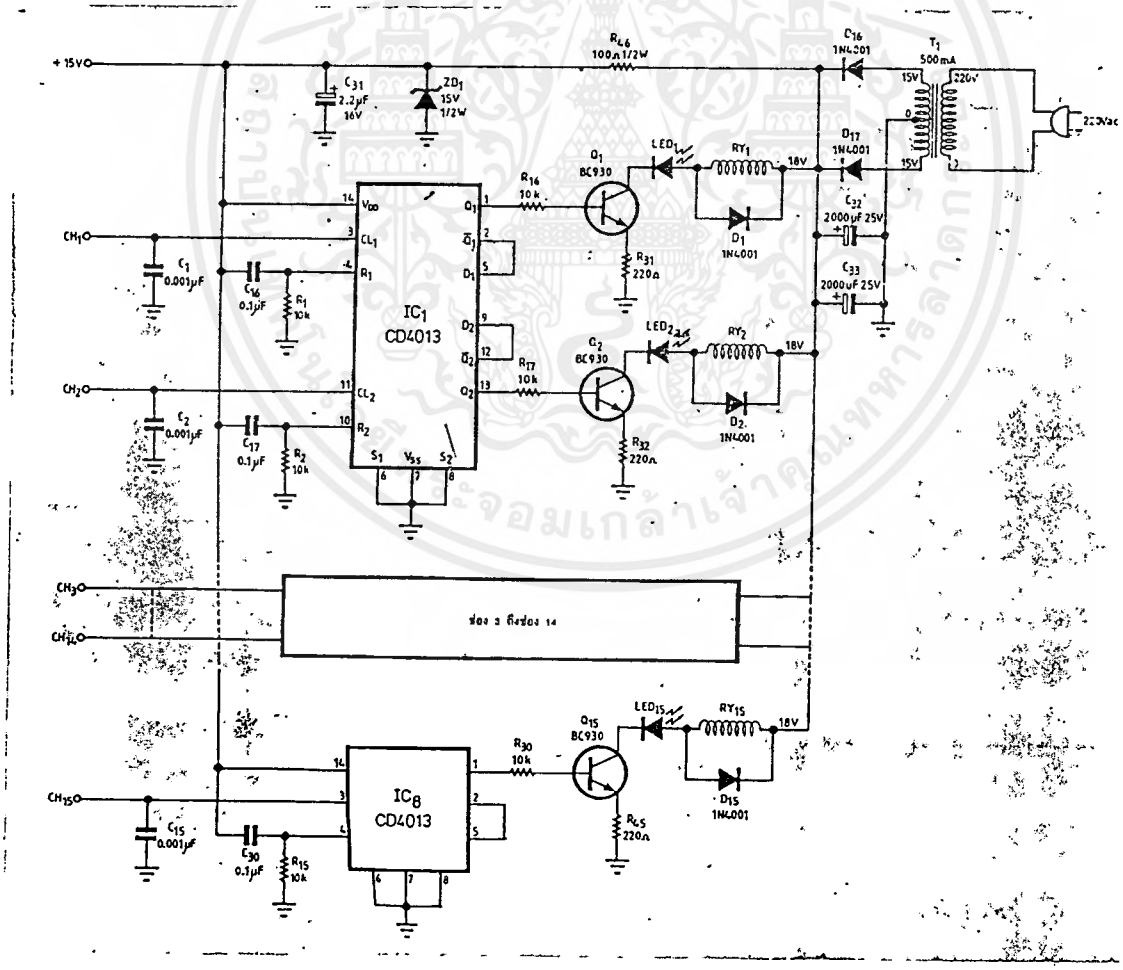


รูปที่ 6 วงจรภาคส่ง 15 ช่อง ระบบอินฟราเรด

ส่วน T_1 , D_{16} , C_{32}, C_{33} , C_{31}, R_{46} , ZD_1 เป็นส่วนของวงจรแหล่งจ่ายไฟตรงสำหรับภาคขับรีเลย์และภาครับสัญญาณอินฟราเรด



รูปที่ 7 แสดงภาครับ



รูปที่ 8 แสดงวงจรภาคขับรีเลย์จำนวน 15 ช่อง

การสร้าง

ขั้นแรกลงอุปกรณ์ของภาคส่งก่อน ลายวงจรที่ให้มานั้นจะติดตั้งอุปกรณ์และสวิทช์ ลงบนด้านเดียวกัน กรณีเช่นนี้เมื่อเราติดตั้งลงกล่องให้สวิตช์โผล่ออกมาไม่สามารถทำได้ แก้ไขโดยย้ายอุปกรณ์ทุกตัวไว้ด้านลายทองแดง ทำให้ด้านบนมีเพียงสวิทช์เท่านั้นที่โผล่ออกมานั้น อุปกรณ์ที่ติดตั้งด้านลายทองแดงก่อนติดต้องตรวจดูขั้วต่างๆให้ดีว่าลงถูกต้องหรือไม่และที่ต้องพิถีพิถันคือ ไอซี SL490B ให้ใช้วิธีพับขาขึ้นมาด้านตรงข้ามของเดิมพยายามดัดกลับมาให้เสร็จในครั้งเดียวอย่าดัดกลับไปมาหลายครั้ง เพราะขาไอซีอาจจะหักได้

กล่องที่ใช้ใส่ให้หาขนาดที่เหมาะมือ เจาะช่องตรงที่ LED อินฟราเรดติดตั้งอยู่ แล้วหาพลาสติกใสสีชามาปิดไว้เพื่อความสวยงาม ฝากล่องใช้แผ่นพลาสติกมาตัดเจาะเป็นช่องตามสวิตช์ให้พอดี ก่อนปิดฝาทับลงบนแผ่นวงจรพิมพ์ ซึ่งถูกยึดติดลงในกล่องก่อนแล้ว ฝาบนนี้เราตัดให้พอดีเบียดกับกล่องเล็กน้อยเวลาปิดจะได้คับพอดีไม่หลุด แต่อย่าลืมเผื่อร่องไว้เล็กน้อยสำหรับงัดฝาออกเวลาเปลี่ยนถ่าน

การปรับแต่ง

วงจรนี้ปรับแต่งที่ VR1 เพียงจุดเดียว โดยให้กดสวิตช์ของภาคส่งปุ่มใดก็ได้ แล้วปรับ VR1 จนกระทั่ง LED ติดสว่างอยู่นิ่งไม่กะพริบ

อีกปัญหาหนึ่งที่อาจเกิดขึ้นคือ กดสวิตช์แล้วรีเลย์ไม่ทำงาน แต่ LED ของรีเลย์นั้นติดสว่าง แสดงว่ากระแสไม่พอ ถ้าเป็นเช่นนี้ให้ลดค่าตัวต้านทาน 220 Ω ที่ต่อตรงขาอิมิตเตอร์ของ 2SC930 (R31 - R45) แต่ไม่ควรลดลงต่ำกว่า 120 Ω เพราะ LED จะเสียหาย

การใช้งาน

มาถึงตอนนี้เราก็เอาเจ้าเครื่องนี้ไปควบคุมสิ่งต่างๆ ได้ตามต้องการ อุปกรณ์ไฟฟ้าจะต่อผ่านหน้าสัมผัสของรีเลย์ เมื่อกดสวิตช์ควบคุมที่เครื่องส่งกดครั้งแรกรีเลย์จะต่ออุปกรณ์ไฟฟ้าให้ทำงาน กดครั้งที่สองจะปิด โดยมี LED ของแต่ละช่องแสดงผลการทำงาน

ความรู้เบื้องต้นเกี่ยวกับไมโครคอนโทรลเลอร์ PIC16F84

PIC16F84 เป็นไมโครคอนโทรลเลอร์ในตระกูล PIC (Peripheral Interface Controller) ของไมโครชิป เทคโนโลยี (Microchip Technology) ไมโครคอนโทลเลอร์ในตระกูล PIC มีด้วยกันหลายเบอร์ แต่ละเบอร์ก็จะมีขีดความสามารถแตกต่างกันออกไป สำหรับการเรียนรู้ในเบื้องต้นควรจะเริ่มศึกษาที่ PIC16F84 เนื่องจากภายใน
PIC16F84 มีหน่วยความจำโปรแกรม (Program memory) เป็นแบบแฟลช (Flash) ซึ่งเป็นหน่วยความจำที่สามารถเขียนและลบได้ด้วยสัญญาณไฟฟ้าเป็นพันครั้ง เมื่อผู้ใช้งานต้องการพัฒนาโปรแกรมก็สามารถทำได้โดยสะดวก

ไมโครคอนโทรลเลอร์ PIC16F84 เป็นไมโครคอนโทรลเลอร์สมัยใหม่ที่จัดอยู่ในกลุ่มของไมโครโปรเซสเซอร์แบบ RISC (Reduced Instruction Set Computer) กล่าวคือ ไมโครคอนโทรลเลอร์ตระกูลนี้จะมีคำสั่งน้อยมากเพียง 33 คำสั่งพื้นฐานเท่านั้นและทุกคำสั่งสามารถทำงานให้เสร็จสิ้นได้ด้วยการใช้สัญญาณนาฬิกาเพียงลูกเดียว ทั้งยังทำงานในลักษณะไปป์ไลน์ (Pipe Line) เหมือนกับไมโครโปรเซสเซอร์สมัยใหม่ ความเร็วในการทำงานจึงสูงมากเมื่อเทียบกับไมโครคอนโทรลเลอร์เบอร์อื่นที่มีความถี่ของสัญญาณนาฬิกาเท่ากัน

คุณสมบัติทางเทคนิคของ PIC16F84

สามารถแบ่งออกเป็น 3 ส่วนคือ **หน่วยประมวลผลกลาง** (Central Processing Unit : CPU) , **ส่วนของเพอริเฟอรัล** (Peripheral) **และคุณสมบัติพิเศษอื่นๆ**

คุณสมบัติทางเทคนิคของหน่วยประมวลผลกลางภายใน PIC16F84

-หน่วยประมวลผลกลางเป็นแบบ RISC

-มีคำสั่งเพียง 33 คำสั่ง ขนาด 14 บิต

-ทุกคำสั่งใช้เวลาในการประมวลผลเพียง 1 ไซเกิลของสัญญาณนาฬิกา หรือประมาณ 400 นาโนวินาทีที่ สัญญาณนาฬิกาความถี่ 10 MHz ยกเว้นชุดคำสั่งการกระโดดจะใช้เวลา 2 ไซเกิลของสัญญาณนาฬิกา

-ประมวลผลข้อมูลขนาด 8 บิต

-มีรีจีสเตอร์ฟังก์ชั่นพิเศษ 15 ตัว

-มีสแต็ก 8 ระดับ

-มีโหมดการอ้างอิงแอดเดรส 3 โหมดคือ แบบโดยตรง (direct),แบบโดยอ้อม (indirct), และแบบสัมพัทธ์ (relative)

-มีแหล่งกำเนิดอินเตอร์รัปต์ 4 แห่งได้แก่

1.รับสัญญาณจากภายนอก โดยป้อนสัญญาณอินเตอร์รัปต์เข้าที่ขาอินพุต RB0/INT

2.จาก TMR0 ไทม์เทอร์โอเวอร์โฟลว์

3.เมื่อเกิดการเปลี่ยนแปลงที่พอร์ต B

4.เมื่อการเขียนข้อมูลลงในหน่วยความจำอีอีพรอมเสร็จสมบูรณ์

-หน่วยความจำข้อมูล (data memory) เป็นแบบอีอีพรอมสามารถลบและเขียนได้ใหม่ประมาณล้านครั้งและเก็บข้อมูลได้นาน 40 ปี

-ขนาดหน่วยความจำโปรแกรมซึ่งเป็นแบบแฟลชมีขนาด 1 กิโลเวิร์ด (1 เวิร์ด ของ PIC16F84 มีขนาด 14 บิต),หน่วยคามจำอีอีพรอมภายใน 64 ไบต์ และหน่วยความจำแรม 68 ไบต์ซึ่งใช้เป็นรีจีสเตอร์

คุณสมบัติทางเทคนิคของเพอริเฟอรัลใน PIC16F84

-มีขาอินพุตเอาท์พุต 13 ขา สามารถกำหนดเป็นขาอินพุตเอาท์พุตได้อย่างอิสระ

-กระแสซิงก์/ซอร์สของแต่ละขาอินพุตเอาท์พุตสูงพอที่จะขับ LED ได้โดยตรง

-กระแสซิงก์สูงสุด 25 mA ต่อขา

-กระแสซอร์สสูงสุด 20 mA ต่อขา

-มีไทเมอร์/เคาน์เตอร์ขนาด 8 บิดคือ TMR0 พร้อมกับปรีสเกลเลอร์ขนาด 8 บิตที่สามารถโปรแกรมได้

คุณสมบัติอื่น ๆ

-มีเพาเวอร์ออนรีเซตในตัว (POR : Power-On Reset)

-มีเพาเวอร์อัปไทเมอร์ในตัว (PWRT: Power-up Timer)

-มีออสซิลเลเตอร์สตาร์ตอัปไทเมอร์ (OST: Oscillator Start-up Timer)

-มีวอตช์ด็อกไทเมอร์ (WDT: Watch Dog Timer) พร้อมกับวงจรออสซิลเลเตอร์ RC ภายในเพื่อช่วยให้การทำงานของไมโครคอนโทรลเลอร์มีความแน่นอนยิ่งขึ้น

-ป้องกันการคัดลอกข้อมูลในหน่วยความจำโปรแกรม

-มีโหมดประหยัดพลังงานหรือโหมดสลีป (Sleep mode)

-สามารถเลือกวงจรออสซิลเลเตอร์ที่ใช้กำหนดการทำงานได้

-การเขียนข้อมูลเข้าสู่หน่วยความจำภายในไมโครคอนโทรลเลอร์เป็นแบบอนุกรมผ่านขาใช้งานเพียง 2 ขา

-เป็นไมโครคอนโทรลเลอร์ที่ได้รับการพัฒนาภายใต้เทคโนโลยีซีมอสแฟลช/อีอีพรอม ความเร็วสูง พลังงานต่ำ

-ย่านไฟเลี้ยงต่ำ 2.0 – 6.0 V

-ปริมาณการใช้กระแสไฟฟ้า

2 mA ที่ไฟเลี้ยง + 5 V สัญญาณนาฬิกาความถี่ 4 MHz

15 μA ที่ไฟเลี้ยง + 2 V สัญญาณนาฬิกาความถี่ 32 kHz

1 μA ที่ไฟเลี้ยง + 2 V ขณะสแตนด์บาย

สถาปัตยกรรมของ PIC16F84

ไมโครคอนโทรลเลอร์ PIC16F84 ได้รับการบรรจุหน่วยประมวลผล,หน่วยความจำ,หน่วยคำนวณทางคณิตศาสตร์และอินพุตเอาท์พุต ทั้งยังมีไทเมอร์และวอตช์ด็อก รูปที่ 1-1 แสดงสถาปัตยกรรมของไมดครคอนโทรลเลอร์ PIC16F84

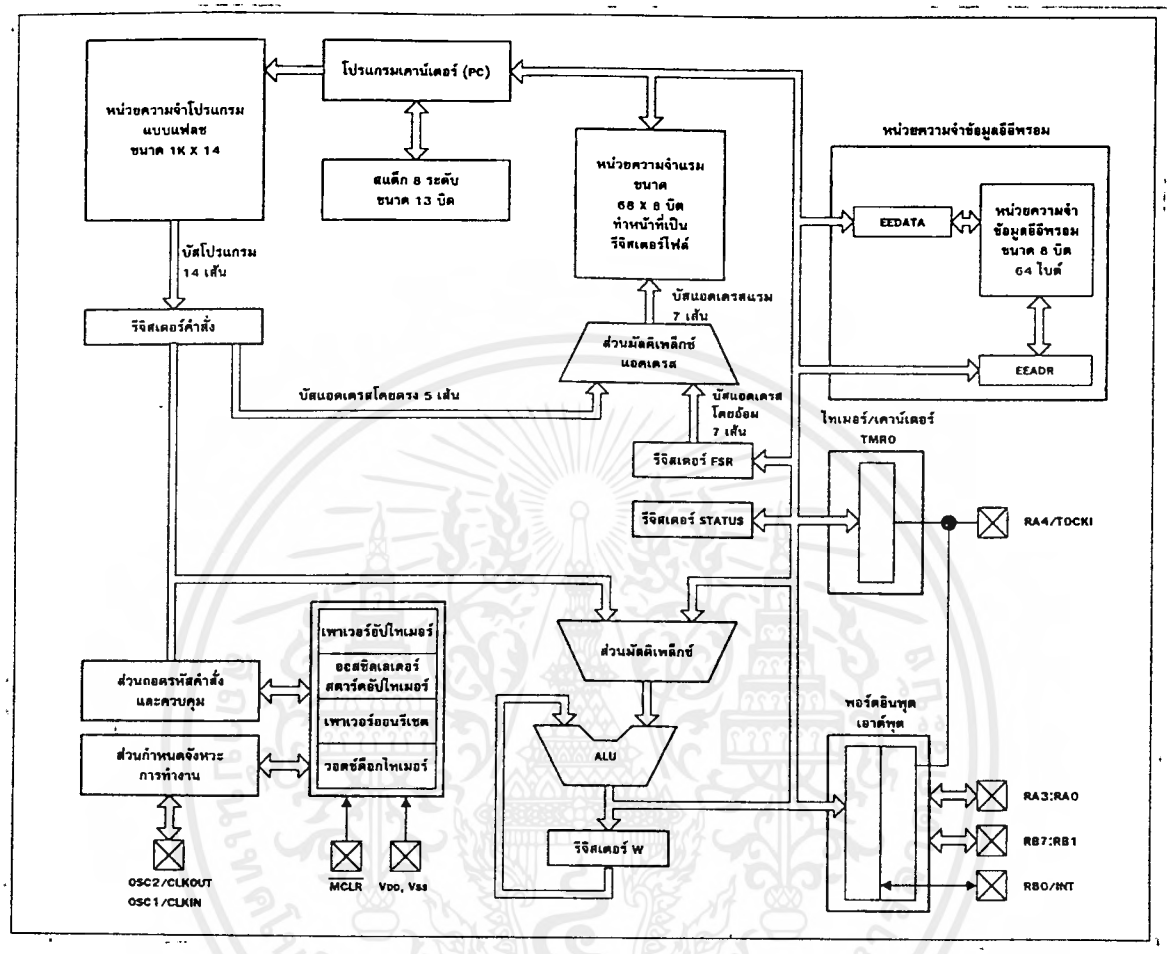
PIC16F84 มีการจัดสรรหน่วยความจำดังนี้

-หน่วยความจำโปรแกรมมีโครงสร้างเป็นหน่วยความจำแบบแฟลช มีขนาด 1 กิโลเวิร์ด

-หน่วยความจำข้อมูลเป็นหน่วยความจำแบบอีอีพรอมขนาด 64 ไบต์

-หน่วยความจำแรมได้รับการกำหนดให้ทำงานเป็นรีจีสเตอร์กำหนดแฟ้มข้อมูลหรือรีจีสเตอร์ไฟล์ขนาด 68 ไบต์

การเข้าถึงหน่วยความจำทั้งหมดของหน่วยประมวลผลกลางหรือซีพียูภายในไมโครคอนโทรลเลอร์นี้สามารถทำได้ทั้งในลักษณะโดยตรง,โดยอ้อมและแบบสัมพัทธ์ โดยมีรีจีสเตอร์ FSR (File Select Register) ทำหน้าที่ในการควบคุมการเข้าถึงหน่วยความจำ



รูปที่ 1 – 1 สถาปัตยกรรมของไมโครคอนโทรลเลอร์ PIC16F84

เมื่อไมโครคอนโทรลเลอร์ทำงานตามคำสั่งที่กำหนดให้ข้อมูลของชุดคำสั่งก็จะถูกนำไปเก็บไว้ในรีจีสเตอร์คำสั่ง (Instruction register) จากนั้นจะถูกส่งต่อไปยังวงจรถอดรหัส เพื่อทำการควบคุมไทเมอร์ทั้งหมดภายในตัวไมโครคอนโทรลเลอร์ นอกจากนั้นยังส่งไปควบคุมหน่วยคำนวณทางคณิตศาสตร์โดยผ่านวงจรมัลติเพล็กซ์

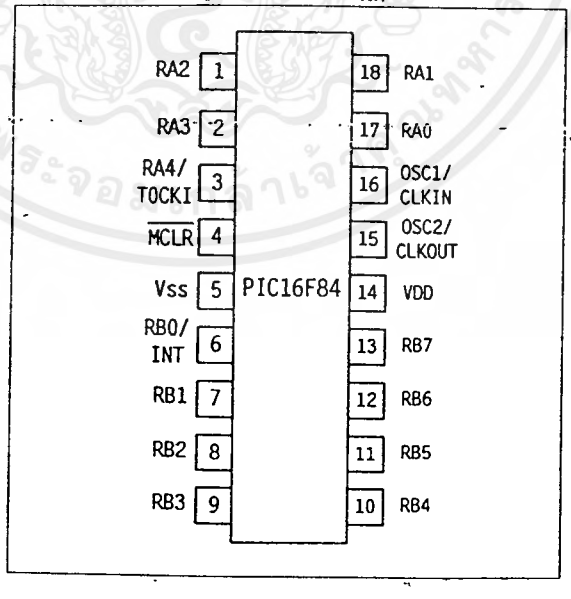
ใน PIC16F84 มีไทเมอร์/เคาน์เตอร์ขนาด 8 บิต 1 ตัวคือ TMR0 ภายในตัวไทเมอร์/เคาน์เตอร์ตัวนี้มีปรีสเกลเลอร์ขนาด 8 บิต ที่สามารถโปรแกรมได้ มีขาต่อใช้งาน 1 ขา คือขา RA4/TOCKI

หน่วยคำนวณทางตณิตศาสตร์ (Arithmatic Logic Unit :ALU) มีขนาด 8 บิต สามรถทำการบวก,ลบ,เลื่อนข้อมูลและประมวลผลทางลอจิก โดยใช้ฟังก์ชั่นบูลีน ในการทำงาน ALU จะต้องมีรีจีสเตอร์ W ช่วย ซึ่งซีพียูจะไม่สามรถเข้าถึงรีจีสเตอร์ W นี้ได้โดยตรงเมื่อ ALU ทำงานจะมีผลต่อบิตทด (carry bit),บิตหลักทด
(digit carry) และบิตศูนย์ (zero bit) ในรีจีสเตอร์ STATUS

ในส่วนของพอร์ตอินพุตใน PIC16F84 มีด้วยกัน2 พอร์ตคือ พอร์ต A และ B โดยในพอร์ต A มี 5 บิตคือ RA0-RA4 สำหรับขา RA4 ยังใช้เป็นขาอินพุตสำหรับรับสัญญาณนาฬิกาจากภายนอกให้แก่ TMR0 ด้วย ส่วนพอร์ต B มี 8 บิตคือ RB0-RB7 สำหรับขา RB0 ใช้เป็นขาอินพุตสำหรับรับสัญญาณอินเตอร์รัปต์

การกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์ PIC16F84 เป็นหน้าที่ของส่วนกำเนิดจังหวะการทำงาน (timing generation) ซึ่งต้องทำงานสัมพันธ์กับไทเมอร์ทั้ง 3 ตัวคือ เพาเวอรีอัปไทเมอร์,ออสซิลเลตอร์สตาร์ตอัปไทเมอร์และวอตช์ด็อกไทเมอร์ สำหรับ PIC16F84 สามารถใช้สัญญาณนาฬิกาจากภายนอกได้โดยต่อสัญญาณเข้าที่ขา OSC1 และ OSC2 หรือจะใช้สัญญาณนาฬิกาภายในไมโครคอนโทรลเลอร์ก็ได้

การจัดขาของ PIC16F84



รูปที่ 1 – 2 การจัดขาของไมโครคอนโทรลเลอร์ PIC16F84

ไมโครคอนโทรลเลอร์ PIC16F84 บรรจุอยู่ในตัวถัง 2 แบบคือ PDIP (Plastic Dual-In Line Package) ซึ่งมีลักษณะเดียวกับไอซีแบบตีนตะขาบและแบบ SOIC อันเป็นตัวถังแบบที่ใช้ติดตั้งบนผิวหน้าของแผ่นวงจรพิมพ์ ตัวถังทั้งสองแบบของ PIC16F84 มีขาต่อใช้งาน 18 ขา โดยแสดงในรูปที่ 1-2 ซึ่งสามารถจัดขาต่อใช้งานของ PIC16F84 เป็น 4 กลุ่มคือ

1.กลุ่มสัญญาณนาฬิกา มี 2 ขา คือ OCS1/CLKIN (ขา16) และ OSC2/CLKOUT (ขา15)

2.กลุ่มขาควบคุม มี 1 ขาคือ MCLR (ขา 4)

3.กลุ่มขาพอร์ตอินพุตเอาต์พุต มี13 ขา แบ่งเป็นขาพอร์ต A 5 ขา ได้แก่ RA0-RA4 (ขา 17,18,1,2 และ3) และขาพอร์ต B ได้แก่ขา RB0-RB7 (ขา6 ถึงขา 13)

4.กลุ่มขาไฟเลี้ยงมี 2 ขาคือ ขา Vss (ขา 5) หรือขาต่อกราวด์ และขา Vdd (ขา 14) หรือขาต่อไฟเลี้ยง ปกติใช้ +5 V

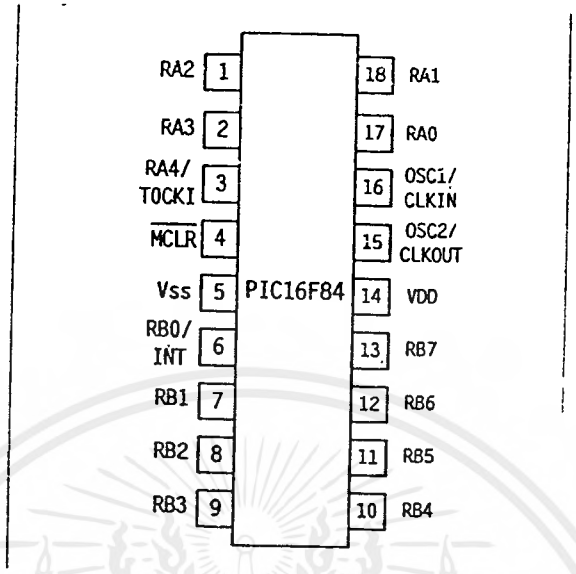
สำหรับรายละเอียดโดยสรุปของขาต่อใช้งานทั้งหมดแสดงในตารางที่ 1 – 1

ชื่อขา	ขาที่	ชนิดของขา	ชนิดของบัฟเฟอร์ที่ต่ออยู่	รายละเอียด
OSC1/CLKIN	16	อินพุต	ชมิตต์ทริกเกอร์/ซีมอส(3)	-เป็นขาสำหรับรับสัญญาณนาฬิกาจากคริสตอลหรือจากแหล่งกำเนิดสัญญาณนาฬิกาจากภายนอก
OSC2/CLKOUT	15	เอาต์พุต	-	-เป็นขาสำหรับส่งสัญญาณนาฬิกาออก -หากทำงานในโหมดคริสตอลให้ต่อกับขาหนึ่งของคริสตอลหรือเซรามิกเรโซเนเตอร์ -หากทำงานในโหมด RC ขานี้ปล่อยลอยไว้ -สัญญาณนาฬิกาที่ออกจากขานี้จะมีความถี่เท่ากับ 1/4 ของความถี่ที่ขา OSC1
$\overline{MCLR}$	4	อินพุต	ชมิตต์ทริกเกอร์	-เป็นขาสำหรับรับสัญญาณรีเซต โดยทำงานที่ลอจิก "0" -เป็นขารับแรงดันสำหรับโปรแกรมหรือเขียนข้อมูลลงในตัวไมโครคอนโทรลเลอร์ด้วย
ขาสัญญาณพอร์ต A				
RA0	17	อินพุต/เอาต์พุต	ทีทีแอล	-เป็นขาอินพุตเอาต์พุต 2 ทิศทางทุกขา
RA1	18	อินพุต/เอาต์พุต	ทีทีแอล	
RA2	1	อินพุต/เอาต์พุต	ทีทีแอล	
RA3	2	อินพุต/เอาต์พุต	ทีทีแอล	
RA4/TOCKI	3	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-เฉพาะขานี้ใช้เป็นขารับสัญญาณนาฬิกาให้แก่ TMR0 ด้วย
ขาสัญญาณพอร์ต B				
RB0/INT	6	อินพุต/เอาต์พุต	ทีทีแอล/ชมิตต์ทริกเกอร์(1)	-เป็นขาอินพุตเอาต์พุต 2 ทิศทางทุกขา -ขา RB0/INT ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์ด้วย
RB1	7	อินพุต/เอาต์พุต	ทีทีแอล	
RB2	8	อินพุต/เอาต์พุต	ทีทีแอล	
RB3	9	อินพุต/เอาต์พุต	ทีทีแอล	
RB4	10	อินพุต/เอาต์พุต	ทีทีแอล	-ขา RB4-RB7 ยังใช้เป็นขาที่ทำให้เกิดสัญญาณอินเตอร์รัปต์ได้ โดยการเปลี่ยนแปลงระดับลอจิกที่ขานี้
RB5	11	อินพุต/เอาต์พุต	ทีทีแอล	
RB6	12	อินพุต/เอาต์พุต	ทีทีแอล/ชมิตต์ทริกเกอร์(2)	-ขา RB6 ยังใช้เป็นขารับสัญญาณนาฬิกาของการโปรแกรมแบบอนุกรมด้วย ในขณะที่ขา RB7 ใช้เป็นขารับข้อมูลของการโปรแกรมแบบอนุกรม
RB7	13	อินพุต/เอาต์พุต	ทีทีแอล/ชมิตต์ทริกเกอร์(2)	
ขาไฟเลี้ยง				
Vss	5	ขาต่อไฟเลี้ยง	-	-ต่อกับกราวด์
VDD	14	ขาต่อไฟเลี้ยง	-	-ต่อกับไฟเลี้ยงบวก ตั้งแต่ 2-6 V

หมายเหตุ : (1) บัฟเฟอร์นี้จะมีอินพุตเป็นแบบชมิตต์ทริกเกอร์เมื่อมีการกำหนดให้เป็นขารับสัญญาณอินเตอร์รัปต์จากภายนอก

(2) บัฟเฟอร์นี้จะมีอินพุตเป็นแบบชมิตต์ทริกเกอร์เมื่อใช้ในการโหมดของการโปรแกรมแบบอนุกรม

(3) บัฟเฟอร์นี้จะมีอินพุตเป็นแบบชมิตต์ทริกเกอร์เมื่อกำหนดให้ออสซิลเลเตอร์ทำงานโหมด RC



รูปที่ 1 – 2 การจัดขาของไมโครคอนโทรลเลอร์PIC16F84

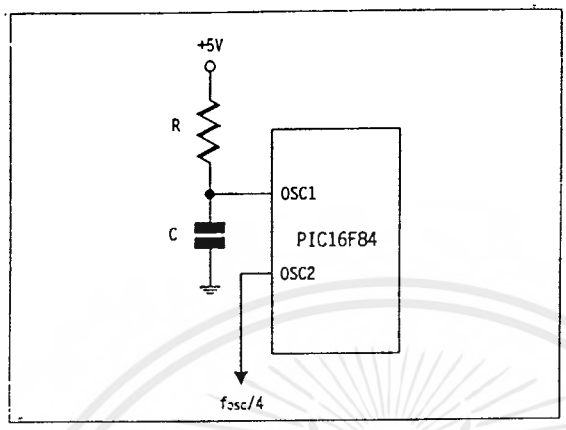
การป้อนสัญญาณนาฬิกาให้แก่ PIC16F84

สามารถเลือกได้ 4 วิธีดังนี้

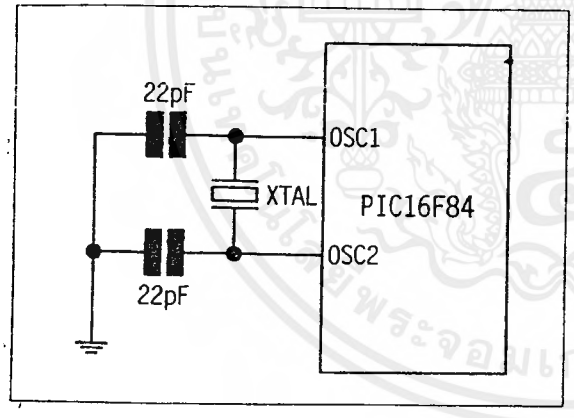
1. RC –เป็นการใช้ตัวต้านทานและตัวเก็บประจุร่วมกับแหล่งกำเนิดสัญญาณนาฬิกาภายในตัวไมโครคอนโทรลเลอร์เอง
2. XT-ใช้คริสตอลหรือเซรามิกเรโซเนเตอร์
3. HS-ใช้คริสตอลความเร็วสูง
4. LP-ใช้คริสตอลกำลังงานต่ำ

อาจสรุปได้ว่า การป้อนสัญญาณนาฬิกาให้แก่ PIC16F84ทำได้ 3 วิธีการ โดยใช้อุปกรณ์ 3 รูปแบบคือ

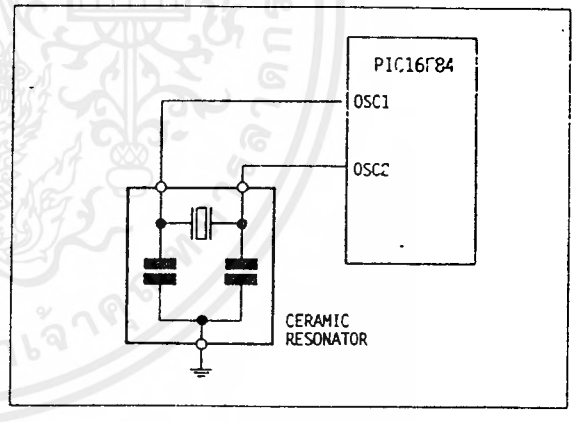
ตัวต้านทานร่วมกับตัวเก็บประจุ,เซรามิกเรโซเนเตอร์และคริสตอล โดยวงจรการต่อแสดงไว้ในรูปที่ 1 – 3, 1 – 4 และ 1 – 5



รูปที่ 1 –3 การป้อนสัญญาณนาฬิกาโดยใช้ตัวต้านทานและตัวเก็บประจุ (โหมด RC)



รูปที่ 1 – 4 การป้อนสัญญาณนาฬิกาโดยใช้คริสตอลเซรามิกเรโซเนเตอร์



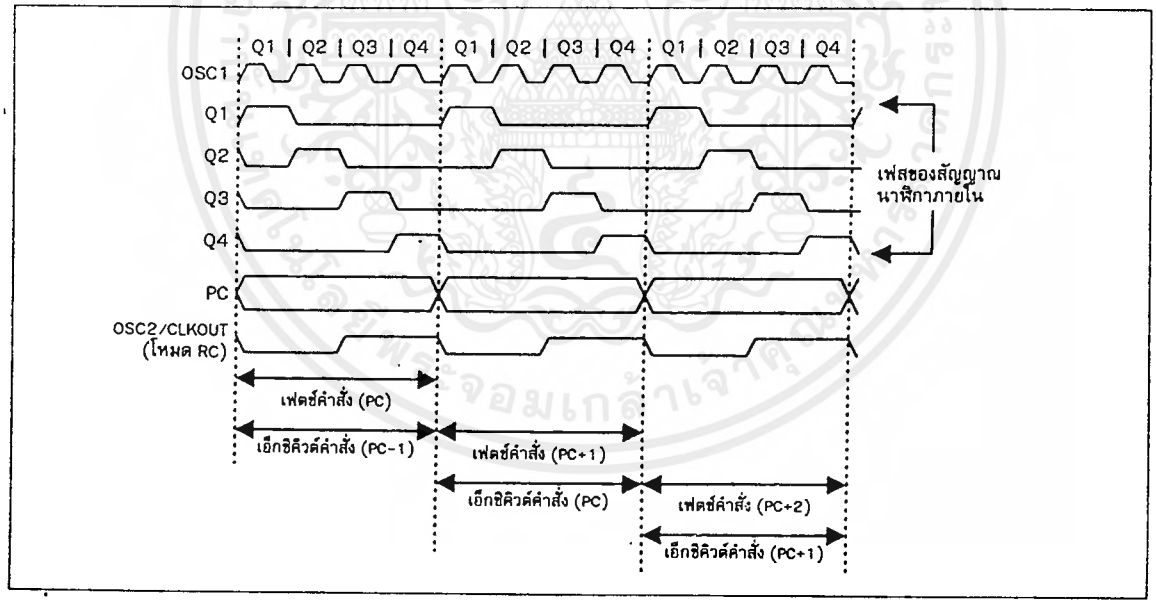
รูปที่ 1 – 5 การป้อนสัญญาณนาฬิกาโดยใช้

จังหวะสัญญาณนาฬิกาและไซเกิลการทำงานของ PIC16F84

สัญญาณนาฬิกาอินพุตของ OSC1 จะถูกหารด้วย 4แล้วแบ่งเป็น 4 ช่วง กำหนดเป็น Q1,Q2,Q3และQ4 โปรแกรมเคาน์เตอร์ (PC) ภายในซีพียูจะเพิ่มค่าขึ้นทุกๆครั้งที่ช่วงสัญญาณนาฬิกา Q1

เมื่อโปรแกรมเคาน์เตอร์เพิ่มค่าขึ้นคำสั่งจะถูกเฟตช์จากหน่วยความจำโปรแกรม (เฟตช์ :Fetch หมายถึงการเรียกหรือการเข้าถึงคำสั่ง แล้วทำการถอดรหัสเป็นภาษาเครื่อง แล้วส่งให้กับซีพียูเพื่อเตรียมประมวลผล) จากนั้นซีพียูทำการแลตช์ข้อมูลคำสั่งนั้นไว้ในรีจีสเตอร์คำสั่ง ที่ช่วงสัญญาณนาฬิกา Q4 คำสั่งจะถูกถอดรหัสและเอ็กซิคิวต์ (executed : การกระทำตามคำสั่งที่กำหนดให้) จนเสร็จสิ้นภายในช่วงสัญญาณนาฬิกา Q1-Q4 หรือภายใน 1 ไซเกิลของสัญญาณนาฬิกา

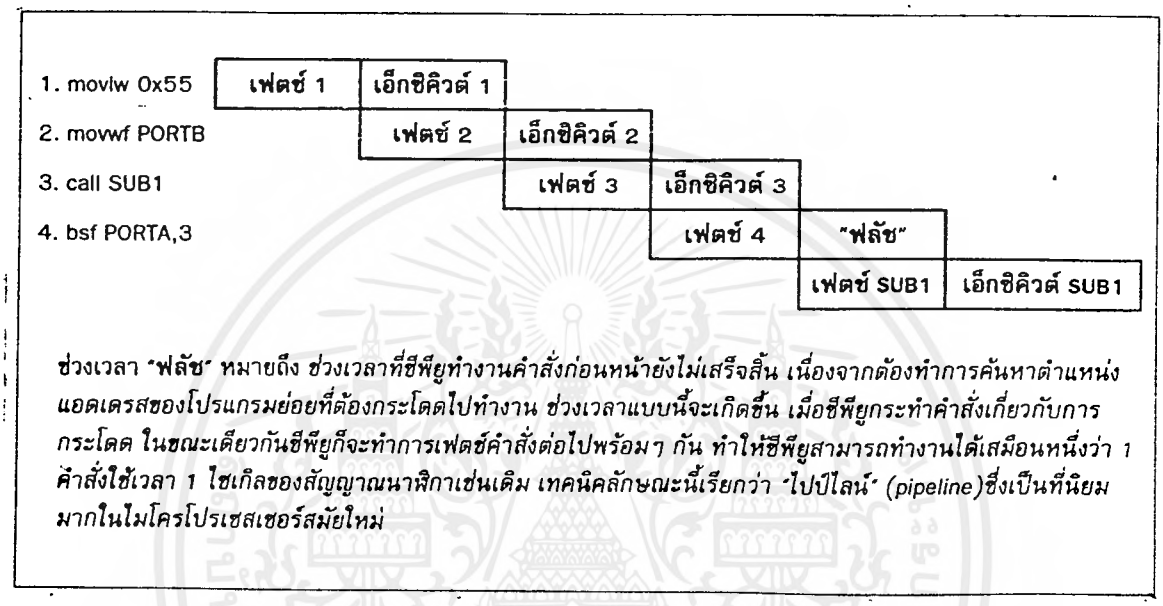
ในรูปที่ 1 – 6 เป็นไดอะแกรม แสดงความสัมพันธ์ระหว่างจังหวะของสัญญาณนาฬิกากับการประมวลผลคำสั่งของไมโครคอนโทรลเลอร์ PIC16F84 ถ้าพิจารณาให้ดีจะพบว่า ไมโครคอนโทรลเลอร์ PIC16F84 จะทำงานโดยใช้สัญญาณนาฬิกาเพียง 1 ลูกต่อหนึ่งคำสั่งเป็นอย่างน้อย แต่สัญญาณนาฬิกาเพียง 1 ลูก จะถูกแบ่งย่อยเป็น 4 ช่วงเพื่อกำหนดจังหวะการทำงานของซีพียูให้มีความชัดเจนมากขึ้น



รูปที่ 1 – 6 ไดอะแกรมเวลาแสดงจังหวะการทำงานของ PIC16F84

ไซเกิลการทำงาน (instruction cycle) ของ PIC16F84 แบ่งเป็น 2 ไซเกิลคือ เฟตช์และเอ็กซีคิวต์ ประกอบด้วย 4 คิวไซเกิล คือ Q1,Q2,Q3และQ4 การเฟตช์และการเอ็กซีคิวต์คำสั่งของ

ไมโครคอนโทรลเลอร์จะมีลักษณะเป็นแบบไปป์ไลน์ (pipeline การทำงานที่เหลื่อมกันในแต่ละขั้นตอนทางคอมพิวเตอร์ เพื่อที่จะทำให้คอมพิวเตอร์ทำงานได้เร็วขึ้น) กล่าวคือ เมื่อซีพียูทำคำสั่งหนึ่งโดยที่คำสั่งนั้นมี 2 ขั้นตอน เมื่อทำไปแล้ว 1 ขั้นตอน คือขั้นตอนเฟตช์คำสั่ง ซีพียูจะเริ่มเฟตช์คำสั่งต่อไปทันที พร้อมๆกับการเอ็กซีคิวต์คำสั่งก่อนหน้านี้ ดังแสดงกระบวนการไปป์ไลน์ในรูปที่ 1 –7



ช่วงเวลา "ฟลัช" หมายถึง ช่วงเวลาที่ซีพียูทำงานคำสั่งก่อนหน้ายังไม่เสร็จสิ้น เนื่องจากต้องทำการคันหาตำแหน่งแอดเดรสของโปรแกรมย่อยที่ต้องกระโดดไปทำงาน ช่วงเวลาแบบนี้จะเกิดขึ้น เมื่อซีพียูกระทำคำสั่งเกี่ยวกับการกระโดด ในขณะเดียวกันซีพียูก็จะทำการเฟตช์คำสั่งต่อไปพร้อมๆ กัน ทำให้ซีพียูสามารถทำงานได้เสมือนหนึ่งว่า 1 คำสั่งใช้เวลา 1 ไซเกิลของสัญญาณนาฬิกาเช่นเดิม เทคนิคลักษณะนี้เรียกว่า "ไปป์ไลน์" (pipeline)ซึ่งเป็นที่นิยมมากในไมโครโปรเซสเซอร์สมัยใหม่

รูปที่ 1 – 7 แสดงลักษณะการทำงานแบบไปป์ไลน์ที่ใช้ใน PIC16F84

จุดเด่นของไมโครคอนโทรลเลอร์ ตระกูล PIC นี้อยู่ตรงที่กระทำคำสั่งแบบไปป์ไลน์อันเป็นกระบวนการทำคำสั่งที่ทันสมัยและเป็นเทคโนโลยีที่ใช้ในไมโครโปรเซสเซอร์บนเครื่องคอมพิวเตอร์พีซี ตั้งแต่ระดับ 80286 เป็นต้นมา ทำให้การทำงานของไมโครคอนโทรลเลอร์ ตระกูล PIC นี้มีความเร็วสูงกว่า ไมโครคอนโทรลเลอร์เบอร์อื่นที่ความถี่ของสัญญาณนาฬิกาเท่ากัน

ไซเกิลการเฟตช์คำสั่งเริ่มต้นในทุกๆคิวไซเกิลที่ 1 พร้อมๆกับคำสั่ง ข้อมูลที่ได้รับการเฟตช์จะถูกนำเข้าไปเก็บไว้ในรีจีสเตอรีคำสั่งในคิวไซเกิลที่ 1 (Q1) จากนั้นคำสั่งดังกล่าวจะได้รับการถอดรหัส และนำไปดำเนินการจนเสร็จสิ้นภายในคิวไซเกิลที่ 2 – 4 (Q2 - Q4) โดยกำหนดให้ในคิวไซเกิลที่ 2 หน่วยความจำข้อมูลที่ใช้เก็บค่าโอเปอร์แรนด์จะถูกอ่าน และถูกเขียนข้อมูลใหม่เข้าไปแทนที่ในคิวไซเกิลที่ 4

กลับไปพิจารณาในรูปที่ 1 – 7 ไมโครคอนโทรลเลอร์ PIC16F84 จะกระทำทุกคำสั่งให้เสร็จสิ้นภายใน สัญญาณนาฬิกาภายใน 1 ลูก ยกเว้นกรณีคำสั่งที่เกิดการสั่งให้กระโดดไปกระทำ

คำสั่งในโปรแกรมย่อยอื่นๆ ดังเช่นที่ไซเกิลการทำงานที่ 4 จะต้องกระทำคำสั่ง bsf PORT A,3 ในไซเกิลการทำงานที่ 3 มีการสั่งให้ซีพียูกระโดดไปทำงานที่โปรแกรมย่อย SUB_1 ทำให้ซีพียูไม่สามารถกระทำคำสั่งในไซเกิลการทำงานที่ 4 ได้เสร็จสิ้นลง เพราะต้องกระโดดไปเฟตช์คำสั่งที่โปรแกรมย่อย SUB_1 ซีพียูจะกระทำคำสั่ง bsf PORT A,3 ค้างไว้จนกว่าจะเสร็จสิ้นการทำงานในโปรแกรมย่อย SUB_1 เมื่อเรียบร้อยก็จะกระโดดกลับมากระทำคำสั่งที่ค้างไว้จนเสร็จสิ้น ช่วงที่เกิดการทำงานไม่เสร็จสิ้น ช่วงที่เกิดการทำงานไม่เสร็จสิ้น อันเนื่องมาจากการกระโดดไปทำงานที่โปรแกรมย่อยเรียกว่า ฟลัช (flush) จะเกิดขึ้นเมื่อซีพียูกระทำคำสั่งที่เกี่ยวข้องกับการกระโดดเสมอ เนื่องจากคำสั่งนี้จะต้องใช้สัญญาณนาฬิกา 2 ลูก ซึ่งแตกกต่างจากคำสั่งอื่นที่สามารถดำเนินการเสร็จสิ้นได้ภายในสัญญาณนาฬิกาเพียงลูกเดียว

การจัดสรรหน่วยวามจำใน PIC16F84
รายละเอียดของรีจีสเตอร์ควบคุมและสแต็ก

การจัดสรรหน่วยความจำภายใน PIC16F84 สามารถแบ่งออกได้เป็น 2 ส่วนคือ หน่วยความจำโปรแกรมและหน่วยความจำข้อมูล

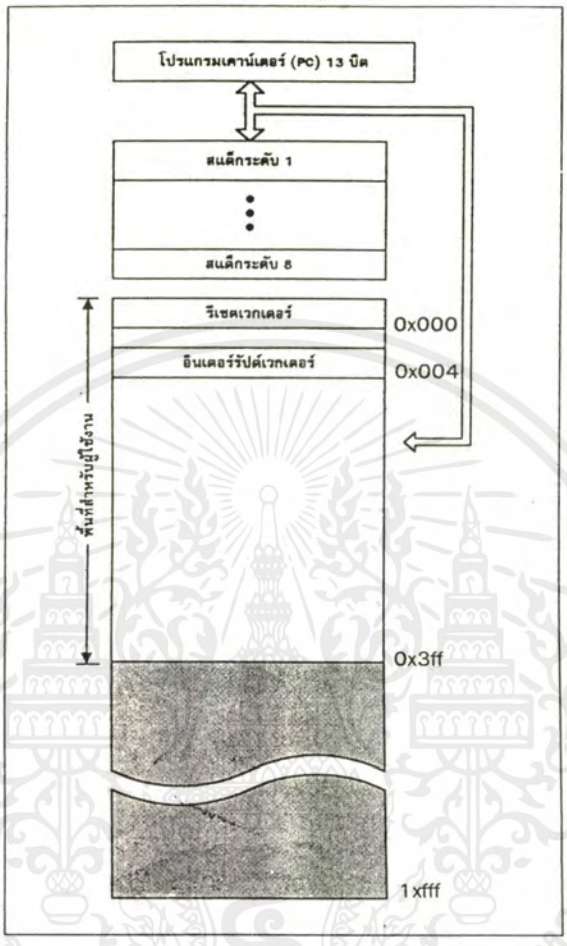
หน่วยความจำโปรแกรม

เป็นหน่วยความจำที่ใช้ในการโปรแกรมการควบคุมการทำงานของระบบหรือใช้เป็นที่เก็บโปรแกรมมอนิเตอร์นั่นเอง หน่วยความจำโปรแกรมภายใน PIC16F84 เป็นหน่วยความจำแบบแฟลช ในขณะที่ PIC16F84 ทำงานปกติหน่วยความจำนี้จะสามารถอ่านได้เพียงอย่างเดียวเท่านั้น และสามารถเขียนหรือแก้ไขได้ก็ต่อเมื่อ PIC16F84 อยู่ในโหมดของการโปรแกรมเท่านั้น

หน่วยความจโปรแกรมมีขนาด 1 กิโลเวิร์ด (จำนวนบิตต่อ 1 เวิร์ดของหน่วยความจำโปรแกรมของ PIC16F84 นี้เท่ากับ 14 บิต) ได้รับการจัดสรรอยู่ในตำแหน่ง 0000H – 03FFH สามารถเขียนข้อมูลเก็บลงในพื้นที่ของหน่วยความจำนี้ได้ทุกแอดเดรส เว้นแต่แอดเดรส 0000H ซึ่งถูกสงวนไว้สำหรับเก็บค่าเวกเตอร์ของการรีเซตหรือรีเซตเวกเตอร์ (Reset Vector) ในรูปที่ 2 – 1 แสดงการจัดสรรพื้นที่ของหน่วยความจำโปรแกรมในไมโครคอนโทรลเลอร์ PIC16F84

หน่วยความจำข้อมูล

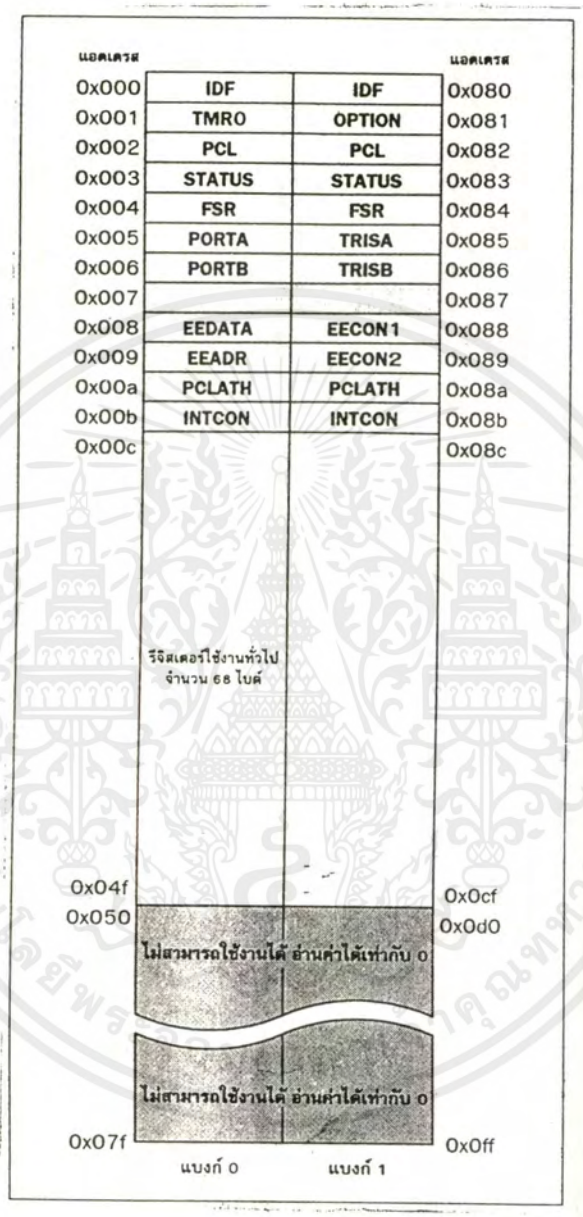
พื้นที่ของหน่วยความจำข้อมูลได้รับการจัดสรรเป้น 2 ส่วนคือ พื้นที่ของรีจีสเตอร์ฟังก์ชั่นพิเศษ (Special Function Registers : SFR) และพื้นที่ของรีจีสเตอร์ใช้งานทั่วไป (General Purpose Registers : GPR) การจัดสรรหน่วยความจำข้อมูลทั้งสองนั้น จะจัดแบ่งเป็นแบงก์ (Bang) ในแต่ละแบงก์ของ SFR จะเป็นรีจีสเตอร์ที่ทำหน้าที่แตกต่างกันออกไป การติดต่อกับหน่วยความจำข้อมูลในแต่ละแบงก์ จะต้องกำหนดค่าของบิตควบคุม RP0 และ RP1 ในรีจีสเตอร์ STATUS ในรูปที่ 2 – 2 เป็นการจัดสรรข้อมูลของหน่วยความจำของ PIC16F84



รูปที่ 2 – 1 การจัดสรรหน่วยความจำโปรแกรมใน PIC16F84

จะเห็นได้ว่าหน่วยความจำข้อมูลจะแบ่งออกเป็น 2 แบงก์ เพื่อเก็บค่าของ SFR และ GPR หน่วยความจำในแบงก์ 0 จะถูกเลือกเมื่อทำการเคลียร์บิต RP0 (บิต 5 ของรีจีสเตอร์ STATUS) และถ้าบิตนี้เซต (เป็น "1") จะเป็นการเลือกหน่วยความจำในแบงก์ 1 ในแต่ละแบงก์จะมีขนาด 128 ไบต์ (0x07f) และในทุก 12 ตำแหน่งแรกของแต่ละแบงก์จะถูกสำรองไว้สำหรับเก็บค่าของ SFR

การเข้าถึงหน่วยความจำข้อมูลสสามารถทำได้โดยตรง (direct) โดยกำหนดแฟ้มข้อมูลที่เก็บรีจีสเตอร์นั้นๆ หรือโดยอ้อม (indirect) โดยผ่านทางรีจีสเตอร์เลือกแฟ้มข้อมูล (File Select Register : FSR)



รูปที่ 2 – 2 การจัดสรรหน่วยความจำข้อมูลใน PIC16F84

รีจีสเตอร์ควบคุมของ PIC16F84

ใน PIC16F84 มีรีจีสเตอร์ควบคุมที่มีบทบาทสำคัญอยู่ 6 ตัวคือ STATUS , OPTION , INCON , PCL , PCLATH และ W ซึ่งจะได้อธิบายตามลำดับดังนี้

รีจีสเตอร์ STATUS

เป็นรีจีสเตอร์ที่ใช้แสดงสถานะทางคณิตศาสตร์ของ ALU, สถานะการทำงานของ PIC16F84 และใช้เป็นตัวกำหนดการเลือกแบงก์ของหน่วยความจำข้อมูล การเข้าถึงรีจีสเตอร์ STATUS เพื่ออ่านและเขียนข้อมูลสามารถกระทำได้ด้วยวิธีการเดียวกับการอ่านและเขียนรีจีสเตอร์ตัวอื่นๆและถ้าหากมีการกระทำคำสั่งเกี่ยวกับคณิตศาสตร์และลอจิก บิต Z,DC และ C จะเกิดการเปลี่ยนแปลงค่าตามผลการทำงานที่เกิดขึ้นโดยไม่คำนึงถึงค่าเดิมที่มีการเขียนหรือกำหนดมาก่อนหน้านี้

รีจีสเตอร์ STATUS มีแอดเดรสที่ตำแหน่ง 0x003 และ0x083 ในหน่วยความจำข้อมูล สำหรับความหมายและการกำหนดค่าในแต่ละบิตของรีจีสเตอร์ STATUS มีรายละเอียดตามรูปที่ 2 – 3 อย่างไรก็ตามในบางคำสั่งที่จัดการข้อมูลระดับบิตเช่น btfss หรือ btfsc จะไม่มีผลต่อค่ารีจีสเตอร์ STATUS bcf,bsf,swapf และ movwf ซึ่งเมื่อกระทำด้วยตัวดังกล่าวแล้ว บิตที่ใช้แสดงสถานจะไม่เกิดการเปลี่ยนแปลงแต่อย่างใด

รีจีสเตอร์ OPTION

เป็นรีจีสเตอร์ที่บรรจุบิตควบคุมการอินเตอร์รัปต์จากสัญญาณภายนอก,ปรีสเกลเลอร์ของ TMR0/WDT, การพูลอัปที่ขาของพอร์ต B และควบคุมการทำงานของ TMR0 มีแอดเดรสอยู่ที่ 0081H สำหรับรายละเอียดของรีจีสเตอร์ OPTION ในแต่ละบิตแสดงดังรูปที่ 2 – 4

รีจีสเตอร์ INTCON

เป็นรีจีสเตอรีที่เก็บค่าบิตของการอีนาเบิลสัญญาณอินเตอร์รัปต์ มีแอดเดรสอยู่ที่ 000BH มีบทบาทสำคัญมากในเรื่องการอินเตอร์รัปต์ สำหรับรายละเอียดของรีจีสเตอร์ INCONในแต่ละบิตแสดงในรูปที่ 2 – 5

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-X	R/W-X	R/W-X
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

R : เป็นบิตที่สามารถอ่านค่าได้
W : เป็นบิตที่สามารถเขียนค่าได้
U : เป็นบิตที่ไม่ใช้งาน อ่านค่าได้เท่ากับ "0"
-n : ค่าที่เกิดขึ้นหลังจาก เกิดเพาเวอร์ออนรีเซต

บิต 7 : **IRP บิตเลือกแบงก์ของรีจิสเตอร์** แต่ใน PIC16F84 ไม่ใช้บิตนี้ *ต้องกำหนดให้เป็น "0" เท่านั้น*

บิต 6-5 : **RP1 และ RP0 บิตเลือกแบงก์ของรีจิสเตอร์** สามารถเข้าถึงได้โดยการอ้างแอดเดรสโดยตรง (direct addressing) ใน PIC16F84 จะใช้เพียงบิต RP0 เท่านั้น ในขณะที่ RP1 ต้องกำหนดให้เป็น "0" การเลือกแบงก์ของรีจิสเตอร์ ทำได้โดยการกำหนดบิต RP1 และ RP0 ดังนี้

00 = แบงก์ 0 (แอดเดรส 0000H-007FH)

01 = แบงก์ 1 (แอดเดรส 0080H-00FFH)

บิต 4 : $\overline{TO}$ *(Time out bit)* **บิตขอบเซตเวลา** บิตนี้สามารถอ่านค่าได้อย่างเดียวเท่านั้น

เป็น "1" เมื่อมีการจ่ายไฟเลี้ยงให้แก่ PIC16F84 หรือเมื่อมีการกระทำคำสั่ง clrwdt หรือ sleep

เป็น "0" เมื่อวอตช์ด็อกไทเมอร์ (WDT) ภายใน PIC16F84 ทำงานครบเวลาที่กำหนดหรือเกิดไทม์เอาต์

บิต 3 : $\overline{PD}$ *(Power down bit)* **บิตเพาเวอร์ดาวน์** บิตนี้สามารถอ่านค่าได้อย่างเดียวเท่านั้น เป็นบิตแสดงสถานะการทำงานของ PIC16F84 ในโหมดสลีป (Sleep)

เป็น "1" เมื่อมีการจ่ายไฟเลี้ยงให้แก่ PIC16F84 หรือเมื่อกระทำคำสั่ง clrwdt

เป็น "0" เมื่อกระทำคำสั่ง sleep

บิต 2 : **Z** *(Zero bit)* **บิตศูนย์** เป็นบิตแสดงผลการกระทำคณิตสาสตร์

เป็น "1" เมื่อกระทำคำสั่งทางคณิตสาสตร์และลอจิกแล้ว เกิดค่าศูนย์ (0) ขึ้น

เป็น "0" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว ค่าไม่เป็นศูนย์

บิต 1 : **DC** *(Digit carry / $\overline{borrow\ bit}$)* **บิตทดหรือยืมระหว่างหลัก** เป็นบิตแสดงผลทางคณิตศาสตร์

ในกรณีที่กระทำคำสั่ง addwf และ addlw บิต DC จะเกิดผลดังนี้

เป็น "1" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว เกิดการทดจากบิตล่างที่4 ไปยังกลุ่มบิตบน (จากบิตที่ 4 ไปยังบิตที่ 5)

เป็น "0" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว ไม่เกิดการทดจากบิตล่างที่4 ไปยังกลุ่มบิตบน (จากบิตที่ 4 ไปยังบิตที่ 5)

ในกรณีที่กระทำคำสั่ง subwf และ sublw บิต DC จะเกิดผลดังนี้

เป็น "1" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว ไม่มีการยืมค่าจากบิตล่างที่4 โดยกลุ่มบิตบน (บิตที่ 5 ยืมค่าจากบิตที่ 4)

เป็น "0" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว เกิดการยืมค่าจากบิตล่างที่4 โดยกลุ่มบิตบน (บิตที่ 5 ยืมค่าจากบิตที่ 4)

บิต 0 : **C** *(Carry bit / $\overline{borrow\ bit}$)* **บิตทดหรือยืม** เป็นบิตที่ใช้แสดงผลการทดและยืมค่าทางคณิตศาสตร์ มีลักษณะงานคล้ายกับบิต DC แต่จะเกิดการเปลี่ยนแปลงค่าก็ต่อเมื่อเกิดการทดหรือยืมค่าจากบิตนัยสำคัญสูงสุดหรือบิต MS8

ในกรณีที่กระทำคำสั่ง addwf และ addlw บิต C จะเกิดผลดังนี้

เป็น "1" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว เกิดการทดจากบิต MSB

เป็น "0" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว ไม่เกิดการทด

ในกรณีที่กระทำคำสั่ง subwf และ sublw บิต C จะเกิดผลดังนี้

เป็น "1" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว ไม่มีการยืมค่าจากบิต MSB

เป็น "0" เมื่อกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้ว เกิดการยืมค่า

รูปที่ 2 – 3 รายละเอียดของบิตต่างในรีจีสเตอร์ STATUS

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

R : เป็นบิตที่สามารถอ่านค่าได้
W : เป็นบิตที่สามารถเขียนค่าได้
U : เป็นบิตที่ไม่ใช้งาน อ่านค่าได้เท่ากับ "0"
-n : ค่าที่เกิดขึ้นหลังจากเกิดเพาเวอร์ออนรีเซต

บิต 7 : **$\overline{\text{RBPU}}$** *(Port B pull-up enable bit)* **ใช้ในการอีนาเบิลการพูลอัปที่ขาพอร์ต B**
"1" - ดิสเอเบิลการพูลอัปที่พอร์ต B
"0" - อีนาเบิลการพูลอัปที่พอร์ต B

บิต 6 : **INTEDG** *(Interrupt edge select bit)* **บิตเลือกขอบขาของสัญญาณอินเตอร์รัปต์จากภายนอก** ใช้เลือกขอบขาของสัญญาณที่ทำให้เกิดการอินเตอร์รัปต์ที่ขา RB0/INT
"1" - เลือกขอบขาขึ้นของสัญญาณ
"0" - เลือกขอบขาลงของสัญญาณ

บิต 5 : **TOCS** *(TMR0 clock source select bit)* **บิตเลือกแหล่งจ่ายสัญญาณนาฬิกาให้แก่ไทเมอร์/เคาน์เตอร์**
"1" - เลือกจากการเปลี่ยนแปลงระดับสัญญาณที่ขา RA4/TOCKI
"0" - เลือกจากสัญญาณนาฬิกาที่ใช้กำหนดไซเกิลการทำงานภายใน PIC16F84

บิต 4 : **TOSE** *(TMR0 source edge select bit)* **บิตเลือกขอบขาของสัญญาณนาฬิกาที่ป้อนให้แก่ไทเมอร์/เคาน์เตอร์** ใช้เลือกขอบขาของสัญญาณนาฬิกาที่จ่ายเข้ามายังขา RA4/TOCKI ในกรณีที่เลือกแหล่งจ่ายสัญญาณนาฬิกาให้ไทม์เมอร์/เคาน์เตอร์ผ่านทางขา RA4/TOCKI โดยการกำหนดบิต TOCS ให้เป็น "1"
"1" - กำหนดให้ TMR0 เกิดการเพิ่มค่าเมื่อเกิดการเปลี่ยนแปลงระดับสัญญาณที่ขา RA4/TOCKI จากสูงมาต่ำ
"0" - กำหนดให้ TMR0 เกิดการเพิ่มค่าเมื่อเกิดการเปลี่ยนแปลงระดับสัญญาณที่ขา RA4/TOCKI จากต่ำไปสูง

บิต 3 : **PSA** *(Prescaler assignment bit)* **บิตกำหนดการทำงานของปรีสเกลเลอร์**
"1" - กำหนดให้ปรีสเกลเลอร์ทำงานร่วมกับวอตช์ด็อกไทเมอร์ เมื่อทำงานกับวอตช์ด็อกไทเมอร์จะเรียกปรีสเกลเลอร์ว่า โพสต์สเกลเลอร์
"0" - กำหนดให้ปรีสเกลเลอร์ทำงานร่วมกับ TMR0

บิต 2-0 : **PS2-PS0** *(Prescaler rate select bit)* **บิตเลือกอัตราส่วนของปรีสเกลเลอร์** เป็นบิตที่ใช้ในการกำหนดอัตราส่วนในการทำงานของปรีสเกลเลอร์ เมื่อทำงานกับ TMR0 และวอตช์ด็อกไทเมอร์ อัตราส่วนนี้จะมีค่าไม่เท่ากัน การกำหนดอัตราส่วนของปรีสเกลเลอร์เป็นไปดังนี้

PS2	*PS1*	*PS0*	*อัตราส่วนเมื่อทำงานกับ WDT*	*อัตราส่วนเมื่อทำงานกับ TMR0*
0	*0*	*0*	*1:1*	*1:2*
0	*0*	*1*	*1:2*	*1:4*
0	*1*	*0*	*1:4*	*1:8*
0	*1*	*1*	*1:8*	*1:16*
1	*0*	*0*	*1:16*	*1:32*
1	*0*	*1*	*1:32*	*1:64*
1	*1*	*0*	*1:64*	*1:128*
	1	*1*	*1:128*	*1:256*

รูปที่ 2 – 4 รายละเอียดของบิตต่างๆในรีจีสเตอร์ OPTION

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

R : เป็นบิตที่สามารถอ่านค่าได้
W : เป็นบิตที่สามารถเขียนค่าได้
U : เป็นบิตที่ไม่ใช้งาน อ่านค่าได้เท่ากับ "0"
-n : ค่าที่เกิดขึ้นหลังจาก เกิดเพาเวอร์ออนรีเซต

บิต 7 : **GIE** *(Global Interrupt enable bit)* **บิตอีนาเบิลการอินเตอร์รัปต์**
"1" – อีนาเบิลการอินเตอร์รัปต์ทั้งหมด
"0" – ดิสเอเบิลการอินเตอร์รัปต์ทั้งหมด

บิต 6 : **EEIE** *(EEPROM write complete Interrupt enable bit)* **บิตอีนาเบิลการอินเตอร์รัปต์เมื่อการเขียนข้อมูลลงในหน่วยความจำข้อมูลอีอีพรอมเสร็จสมบูรณ์**
"1" – อีนาเบิลการอินเตอร์รัปต์แบบนี้
"0" – ดิสเอเบิลการอินเตอร์รัปต์แบบนี้

บิต 5 : **TOIE** *(TMR0 overflow Interrupt enable bit)* **บิตอีนาเบิลการอินเตอร์รัปต์เมื่อ TMR0 เกิดการโอเวอร์โฟลว**
"1" – อีนาเบิลการอินเตอร์รัปต์แบบนี้
"0" – ดิสเอเบิลการอินเตอร์รัปต์แบบนี้

บิต 4 : **INTE** *(RB0/INT interrupt enable bit)* **บิตอีนาเบิลการอินเตอร์รัปต์จากสัญญาณภายนอกที่ขา RB0/INT**
"1" – อีนาเบิลการอินเตอร์รัปต์แบบนี้
"0" – ดิสเอเบิลการอินเตอร์รัปต์แบบนี้

บิต 3 : **RBIE** *(Port B change Interrupt enable bit)* **บิตอีนาเบิลการอินเตอร์รัปต์เนื่องจากเกิดการเปลี่ยนแปลงระดับสัญญาณที่ขาพอร์ต B บิตที่ 4-7**
"1" – อีนาเบิลการอินเตอร์รัปต์แบบนี้
"0" – ดิสเอเบิลการอินเตอร์รัปต์แบบนี้

บิต 2 : **TOIF** *(TMR0 overflow interrupt flag)* **บิตแจ้งการเกิดอินเตอร์รัปต์อันเนื่องมาจาก TMR0 เกิดโอเวอร์โฟล**
"1" – แจ้งให้ทราบว่าเกิดการอินเตอร์รัปต์อันเนื่องมาจาก TMR0 เกิดโอเวอร์โฟลว
"0" – ไม่เกิดการอินเตอร์รัปต์อันเนื่องมาจาก TMR0 เกิดโอเวอร์โฟลว

บิต 1 : **INTF** *(RB0/INT Interrupt flag)* **บิตแจ้งการเกิดอินเตอร์รัปต์เนื่องจากสัญญาณภายนอกที่ขา RB0/INT**
"1" – แจ้งให้ทราบว่าเกิดการอินเตอร์รัปต์อันเนื่องมาจากสัญญาณภายนอกที่ขา RB0/INT
"0" – ไม่เกิดการอินเตอร์รัปต์อันเนื่องมาจากสัญญาณภายนอกที่ขา RB0/INT

บิต 0 : **RBIF** *(Port B change Interrupt flag)* **บิตแจ้งการเกิดอินเตอร์รัปต์อันเนื่องมาจากเกิดการเปลี่ยนแปลงระดับสัญญาณที่ขาพอร์ต B บิตที่ 4-7**
"1" – แจ้งให้ทราบว่าเกิดการอินเตอร์รัปต์อันเนื่องมาจากเกิดการเปลี่ยนแปลงระดับสัญญาณที่ขาพอร์ต B บิตที่ 4-7
"0" – ไม่เกิดการอินเตอร์รัปต์อันเนื่องมาจากเกิดการเปลี่ยนแปลงระดับสัญญาณที่ขาพอร์ต B บิตที่ 4-7

รูปที่ 2 – 5 รายละเอียดของบิตต่างๆในรีจีสเตอร์ INCON

แอ็ดเดรส	ชื่อรีจิสเตอร์	บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0	ค่าที่เกิดขึ้นหากเกิดเพาเวอร์ออนรีเซ็ต	ค่าที่เกิดขึ้นหากเกิดการรีเซ็ตในแบบอื่น
แบงก์ 0											
00H	INDF	-ไม่ใช่รีจิสเตอร์หลัก ทำให้ไม่สามารถทำการอ่านข้อมูลได้โดยตรง -ใช้เก็บค่าของรีจิสเตอร์ FSR เพื่อเข้าถึงแอดเดรสของหน่วยความจำข้อมูล								---- ----	---- ----
01H	TMR0	รีลไทม์คลือกและเคาน์เตอร์ขนาด 8 บิต								ไม่ทราบค่า	ไม่เปลี่ยนแปลง
02H	PCL	รีจิสเตอร์เก็บค่า 8 บิตล่างของโปรแกรมเคาน์เตอร์ (PC)								0000 0000	0000 0000
03H	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04H	FSR	พอยน์เตอร์ 0 ที่ใช้ชี้แอดเดรสของหน่วยความจำข้อมูลโดยอ้อม								ไม่ทราบค่า	ไม่เปลี่ยนแปลง
05H	PORTA	-	-	-	RA4/ TOCKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06H	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	ไม่ทราบค่า	ไม่เปลี่ยนแปลง
07H		ไม่มีการใช้งาน อ่านค่าได้ 0								---- ----	---- ----
08H	EEDATA	รีจิสเตอร์สำหรับเก็บข้อมูลของหน่วยความจำอีอีพรอม								ไม่ทราบค่า	ไม่เปลี่ยนแปลง
09H	EEADR	รีจิสเตอร์สำหรับเก็บค่าแอดเดรสของหน่วยความจำอีอีพรอม								ไม่ทราบค่า	ไม่เปลี่ยนแปลง
0AH	PCLATH (1)	-	-	-	เป็นบัฟเฟอร์สำหรับเก็บค่า 5 บิตบนของ PC					---0 0000	---0 0000
0BH	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
แบงก์ 1											
80H	INDF	-ไม่ใช่รีจิสเตอร์หลัก ทำให้ไม่สามารถทำการอ่านข้อมูลได้โดยตรง -ใช้เก็บค่าของรีจิสเตอร์ FSR เพื่อเข้าถึงแอดเดรสของหน่วยความจำข้อมูล									---- ----	---- ----
81H	OPTION	$\overline{RBPU}$	$\overline{INTEDG}$	TOCS	TOSE	PS3	PS2	PS1	PS0	1111 1111	1111 1111
82H	PCL	-ไม่ใช่รีจิสเตอร์หลัก ทำให้ไม่สามารถทำการอ่านข้อมูลได้โดยตรง -ใช้เก็บค่าของรีจิสเตอร์ FSR เพื่อเข้าถึงแอดเดรสของหน่วยความจำข้อมูล									0000 0000	0000 0000
83H	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84H	FSR	พอยน์เตอร์ 0 ที่ใช้ชี้แอดเดรสของหน่วยความจำข้อมูลโดยอ้อม								ไม่ทราบค่า	ไม่เปลี่ยนแปลง
85H	TRISA	-	-	-	รีจิสเตอร์กำหนดทิศทางของพอร์ต A					---1 1111	---1 1111
86H	TRISB	รีจิสเตอร์กำหนดทิศทางการถ่ายทอดข้อมูลของพอร์ต B								1111 1111	1111 1111
87H		ไม่มีการใช้งาน อ่านค่าได้ 0								---- ----	---- ----
88H	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89H	EECON2	รีจิสเตอร์ควบคุมหน่วยความจำอีอีพรอม 2 (ไม่ใช่รีจิสเตอร์หลักไม่สามารถอ่านข้อมูลได้โดยตรง)								---- ----	---- ----
8AH	PCLATH	-	-	-	เป็นบัฟเฟอร์สำหรับเก็บค่า 5 บิตบนของ PC					---0 0000	---0 0000
88H	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u

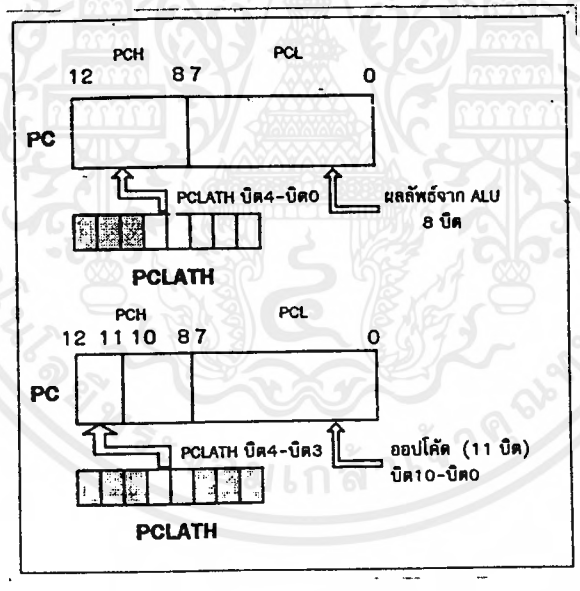
หมายเหตุ : x หมายถึง ไม่ทราบค่า, u หมายถึง ข้อมูลที่บิตนั้นไม่เปลี่ยนแปลง, q หมายถึง ค่าจะขึ้นอยู่กับเงื่อนไขของรีจิสเตอร์แต่ละตัว, – หมายถึง *ไม่สามารถเขียนได้*
(1) ค่าใน PCLATH ทั้ง 5 บิตสามารถถ่ายทอดให้ PC ได้ แต่ค่าของ PC 5 บิตบนไม่สามารถถ่ายทอดให้ PCLATH ได้
(2) บิต $\overline{TO}$ และ $\overline{PD}$ ไม่ได้รับผลกระทบในกรณีที่เกิดการรีเซตที่ขา $\overline{MCLR}$

ตารางที่ 2 – 1 รายละเอียดของรีจีสเตอร์ไฟล์ทั้งหมดของ PIC16F84

รีจีสเตอร์โปรแกรมเคาน์เตอร์ PCL และ PCLATH

โปรแกรมเคาน์เตอร์ (program counter : PC) เป็นรีจีสเตอร์ที่มีหน้าที่ชี้ตำแหน่งแอดเดรสต่อไปของหน่วยความจำโปรแกรมที่ซีพียูต้องไปทำงาน

โปรแกรมเคาน์เตอร์หรือ PC ใน PIC16F84 มีขนาด 13 บิต แบ่งเป็น 2 ส่วนคือ รีจีสเตอร์โปรแกรมเคาน์เตอร์ไบต์ต่ำหรือ PCL (program counter low byte) ซึ่งในส่วนนี้มีขนาด 8 บิต สามารถอ่านและเขียนค่าโดยตรง ในขณะที่อีกส่วนหนึ่งมีขนาด 5 บิต ไม่สามารถอ่านหรือเขียนข้อมูลได้โดยตรง ต้องอาศัยการเขียนและอ่านค่าผ่านรีจีสเตอร์ PCLATH โดยรีจีสเตอร์ PCLATH จะทำการเก็บค่า 5 บิตบนโปรแกรมเคาน์เตอร์ไว้และถ่ายทอดลงสู่ 5 บิตบนของโปรแกรมเคาน์เตอร์ไบต์สูง (PCH), ก็ต่อเมื่อโปรแกรมเคาน์เตอร์มีการโหลดค่าใหม่เข้ามาซึ่งจะเกิดขึ้นเมื่อกระทำคำสั่ง Call หรือ Goto ดังแสดงไดอะแกรมของการถ่ายทอดข้อมูลตามรูปที่ 2 – 6



รูปที่ 2 – 6 การถ่ายทอดข้อมูลภายในโปรแกรมเคาน์เตอร์

สแต็ก (Stack)

ในไมโครคอนโทรลเลอร์ PIC16F84 ได้จัดสรรสแต็กหรือพื้นที่ในหน่วยความจำ เพื่อใช้ในการเก็บค่าของโปรแกรมเคาน์เตอร์ชั่วคราวไว้ 8 ระดับ ในแต่ละระดับมีขนาด 13 บิต ถ้ามีการพุช (push) 1 ครั้งสแต็กจะทำการเก็บค่าของโปรแกรมเคาน์เตอร์ (PC) ไว้ 1 ค่า

สแต็กใน PIC16F84 จะรองรับได้ 8 ครั้งต่อเนื่องกันหากมีการพุชเป็นครั้งที่ 9 ซีพียูจะนำค่าของ PC ในครั้งที่ 9 นี้เก็บลงในสแต็กที่เก็บค่า PC ในครั้งที่ 1

การพุช (push) หมายถึ7การที่ซีพียูกระทำคำสั่งตามลำดับปกติแล้วไปพบคำสั่ง call หรือเกิดการอินเตอร์รัปต์ ก่อนที่ซีพียูจะทำการเก็บค่าของโปรแกรมเคาน์เตอร์ในขณะนั้นไว้ในหน่วยความจำชั่วคราว ซึ่งหน่วยความจำชั่วคราวดังกล่าวนั้นก็คือ สแต็ก เมื่อซีพียูกระทำคำสั่งในโปรแก่รมย่อยที่กระโดดไปเรียบร้อยแล้ว ซีพียูจะกลับไปอ่านค่าโปรแกรมเคาน์เตอร์ที่เก็บไว้ในสแต็ก แล้วกลับมากระทำคำสั่งในแอดเดรสต่อไป

รีจีสเตอร์ W

ไมโครคอนโทรลเลอร์ PIC16F84 มีรีจีสเตอร์ที่ใช้ในหารทำงานหลักอยู่ตัวหนึ่งก็คือ รีจีสเตอร์ W หากเปรียบเทียบกับไมโครคอนโทรลเลอร์เบอร์อื่นๆ รีจีสเตอร์ W เทียบได้กับแอกคิวมูเลเตอร์นั่นเอง เมื่อ PIC16F84 กระทำคำสั่งทางคณิตศาสตร์ รีจีสเตอร์W จะเป็นรีจีสเตอร์ที่ซีพียูติดต่อด้วยการโอนข้อมูลหรือการตรวจสอบข้อมูลจะกระทำที่รีจีสเตอร์ W นี้เป็นหลัก

รีจีสเตอร์ไฟล์ (File Register)

เนื่องจาก PIC F84 มีรีจีสเตอรีที่เกี่ยวข้องหลายตัวจะต้องใชการเข้าถึงแบบโดยอ้อม ดังนั้น PIC16F84 จึงมีการจัดรวบรวมรีจีสเตอร์ทั้งหมดที่ต้องใช้การเข้าถึงแบบโดยอ้อมไว้ในลักษณะแฟ้มข้อมูล ดังแสดงในตารางที่ 2 – 1

รีจีสเตอร์ได้รับการมารมมีชื่อว่า รีจีสเตอร์ไฟล์ (Register file) มีขนาด 8 บิต เมื่อพิจารณาตามตารางที่ 2-1จะพบว่าใน PIC16F84 มีรีจีสเตอร์ฟังก์ชั่นพิเศษ 12 ตัว ที่ถูกกำหนดหน้าที่และตำแหน่งไว้แล้ว และอีก 68 ตัว จะได้รับการกำหนดให้ใช้งานอย่างอิสระ

ชุดคำสั่งของ PIC16F84 และการเข้าถึงรีจิสเตอร์

ไมโครคอนโทรเลอร์ตระกูล PIC มีชุดคำสั่งหลักที่เหมือนกัน 35 คำสั่ง สำหรับ PIC16F84 จะมีเพิ่มเติมอีก 2 คำสั่งเป็น 37 คำสั่ง โดยแบ่งออกเป็น 6 กลุ่ม ตามลักษระการทำงานดังนี้

1) กลุ่มคำสั่งโอนย้ายหรือกำหนดค่าข้อมูล มี 3 คำสั่ง
2) กลุ่มคำสั่งเปลี่ยนแปลงค่าของรีจิสเตอร์ มี 10 คำสั่ง
3) กลุ่มคำสั่งการกระโดดและควบคุมลำดับการทำงานของโปรแกรม มี 9 คำสั่ง
4) กลุ่มคำสั่งควบคุมไมโครคอนโทรลเลอร์ มี 5 คำสั่ง
5) กลุ่มคำสั่งประมวลผลทางลอจิก มี 6 คำสั่ง
6) กลุ่มคำสั่งประมวลผลทางคณิตศาสตร์ มี 4 คำสั่ง

ในขณะที่ผู้ผลิตคือไมโครชิปได้แบ่งกลุ่มคำสั่งออกเป็น 3 กลุ่ม ตามขนาดของข้อมูลเพื่อให้เข้ากับรูปแบบของโปรแกรมแอสเซมเบลอร์ MPASM ดังนี้

1) กลุ่มคำสั่งจัดข้อมูลระดับไบต์ (Byte-oriented instructions)
2) กลุ่มคำสั่งจัดการข้อมูลระดับบิต (Bit-oriented instructions)
3) กลุ่มคำสั่งควบคุมการทำงาน (Literal and control operations)

ค่าคงที่ของฐานสิบหกที่เกี่ยวข้องในการเขียนโปรแกรมและอธิบายการทำงานของชุดคำสั่งของไมโครคอนโทรลเลอร์ตระกูล PIC จะมีการเขียนที่แตกต่างจากตระกูล MCS-51 โดยจะเขียนเป็น 0x00 2 หลักท้ายจะเป็นค่าของเขฐานสิบหก ส่วน x โดยปกติจะมีค่าเป็น " 0 " การเขียนในลักษณะนี้ได้รับการกำหนดมาจากโปรแกรมแอสแซมเบลอร์ MPASM ของไมโครชิบ จะเห็นได้ว่าไม่จำเป็นต้องมีอักษรตัว H แต่ถ้าหากค่าเป็นเลขฐานสิบ ใส่อักษร d ต่อท้ายด้วยเสมอ

ความหมายของตัวแปรที่ควรทราบ

ในแต่ละคำสั่งของ PIC16F84 จะมีตัวแปรที่เกี่ยวข้องอยู่หลายตัว โดยจะเขียนต่อจากคำสั่งเพื่อเป็นการกำหนดหรือระบุว่าคำสั่งนั้นๆกระทำ หรือมีผลต่อรีจิสเตอร์หรือข้อมูลใดบ้าง สามารถสรุปรายละเอียดความหมายของตัวแปรทั้งหมดได้ดังนี้

f หมายถึง รีจิสเตอร์

d หมายถึง ที่เก็บค่าปลายทาง ถ้าหากเป็น " 0 " จะเป็นการกำหนดให้เก็บค่าที่รีจิสเตอร์ W แต่ถ้าหากเป็น " 1 " จะเป็นการกำหนดให้เก็บค่าหลังจากการกระทำคำสั่งในรีจิสเตอร์ไฟล์

k หมายถึง ค่าคงที่ใดๆในกรณีที่ใช้กับคำสั่งประมวลผลของทางคณิตศาสตร์และลอจิก และยังหมายถึงตำแหน่งแอดเดรสที่ต้องการให้กระโดดไปในกรณีที่ใช้กับคำสั่งการกระโดด เช่น call หรือ goto

b หมายถึง ข้อมูลระดับบิตหรือบิตแอดเดรสภายในรีจิสเตอร์ไฟล์ขนาด 8 บิต

d หมายถึง ข้อมูลเลขฐานสิบ กรณีที่ใช้กับคำสั่งประมวลผลทางคณิตศาสตร์และลอจิก

W หมายถึง รีจิสเตอร์ W (แอกคิวมูเลเตอร์)

x หมายถึง ตำแหน่งที่ไม่สนใจ เมื่อใช้แอสแซมเบลอร์ MPASM ของไมโครชิบจะกำหนดค่า x นี้เท่ากับ " 0 " โดยอัตโนมัติ เว้นแต่จะมีการกำหนดเพจหรือหน้าของหน่วยความจำเป็นอย่างอื่น

label หมายถึง ตำแหน่งของแอสเดรสที่ต้องการให้กระโดดไปทำงาน

TOS หมายถึง ตำแหน่งบนสุดของสแต็ก

PC หมายถึง โปรแกรมเคาน์เตอร์

H หมายถึง ข้อมูลเลนฐานสิบหก

[ ] หมายถึง ออปชั่น หรือส่วนที่เพิ่มเติม

( ) หมายถึง ค่าของตัวแปร หรือข้อมูล

$\in$ หมายถึง เซตของข้อมูล หรือตัวแปร

กลุ่มคำสั่งการโอนย้าย หรือกำหนดค่าข้อมูล

มีทั้งสิ้น 3 คำสั่งคือ moviw , movf และ movwf มีรายละเอียดดังนี้

คำสั่ง movlw (move data to W register)

เป็นคำสั่งที่ใช้การกำหนดค่าคงที่ 8 บิตในรีจิสเตอร์ W ขอบเขตของค่าคงที่ที่ใช้ได้ระหว่าง 0-255 มีรูปแบบและรายละเอียดดังนี้

รูปแบบ : movlw k

โอเปอร์แรนด์ : $0 \leq k \leq 255$

การทำงาน : k - > (W)

จำนวนเวิร์ด : 1

จำนวนไซเกิล : 1

ตัวอย่าง : movlw 0 x 5A

หลังจากกระทำคำสั่ง

ค่าของ W = 0 x 5 A

คำสั่ง movf (move data from register to register)

เป็นคำสั่งที่ใช้ในการโอนย้ายข้อมูลของรีจิสเตอร์ที่กำหนดไปเก็บไว้ในรีจิสเตอร์ W หรือรีจิสเตอร์ W หรือ รีจิสเตอร์ไฟล์ที่กำหนด การเลือกตำแหน่งปลายทางกำหนดได้ด้วยข้อมูล " 0 " หรือ " 1 " ต่อหลังรีจิสเตอร์ไฟล์ต้นทาง หากกำหนดให้เป็น " 0 " จะหมายถึงให้โอนย้ายข้อมูลไปเก็บไว้ที่รีจิสเตอร์ W ถ้าหากกำหนดให้เป็น " 1 " ข้อมูลจะโอนย้ายมาที่ตำแหน่งเดิม ซึ่งก็คือไม่เกิดการโอนย้ายข้อมูล แต่จะเกดผลกับแฟลก โดยแฟลกศูนย์ (Z) จะเซต ดังนั่นคำสั่งนี้จึงสามารถนำไปใช้เซตแฟลกศูนย์ (Z) ได้เลย โดยไม่ต้องใช้คำสั่งทางคณิตศาสตร์แต่อย่างใด สำหรับรูปแบบและรายละเอียดของรายละเอียดของคำสั่งมีดังนี้

รูปแบบ : movf f, d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0, 1]$

การทำงาน : (f) - . (W) หรือ (f) - > (f), Z = 1

แฟลกที่การเปลี่ยนแปลง : แฟลกศูนย์ หรือ Z

จำนวนเวิร์ด : 1

จำนวนไซเกิล : 1

ตัวอย่าง : movf FSR , 0

หลังจากการกระทำคำสั่ง

ค่าของ W = ค่าของรีจิสเตอร์ FSR movf FRS , 1

หลังจากกระทำคำสั่ง

ค่าของ FSR จะเท่าเดิมแต่แฟลก z จะเซต

คำสั่ง movwf (move data from W register to file register)

เป็นคำสั่งที่ให้ผลตรงข้ามกับ movf นั่นคือ เป็นคำสั่งโอนย้ายข้อมูลจากรีจิสเตอร์ W มาเก็บไว้ในรีจิสเตอร์ไฟล์ที่กำหนดมีรายละเอียดและรูปแบบดังนี้

รูปแบบ : movwf f

โอเปอร์แรนด์ : $0 \leq f \leq 127$

การทำงาน : (W) - > (f)

ตัวอย่าง : movwf OPTION

ก่อนกระทำคำสั่ง

ค่าของ OPTION = 0xFF

ค่าของ W = 0x4F

หลังกระทำคำสั่ง

ค่าของ OPTION = 0x4F

ค่าของ W = 0x4F

กลุ่มคำสั่งเปลี่ยนแปลงค่าของรีจิสเตอร์

มีทั้งสิ้น 10 คำสั่ง คือ clrf , clrw , comf , decf , incf , bcf , bsf , rlf , rlf , rrf และ swapf

คำสั่ง clrf และ clrw (clear file register & clear W register)

clrf เป็นคำสั่งเคลียร์ค่าของรีจิสเตอร์ไฟล์ใดๆให้เป็นศูนย์ ส่วน clrw เป็นคำสั่งเคลียร์ค่าของรีจิสเตอร์ W ให้เป็นศูนย์ และเมื่อกระทำคำสั่งนี้แล้วแฟลกศูนย์ (Z) จะเซต สำหรับรายละเอียดและมีรูปแบบมีดังนี้

คำสั่ง clrf

รูปแบบ : clrf f

โอเปอร์แรนด์ : $0 \leq f \leq 127$

การทำงาน : 00H - > (F) , 1 - > Z

แฟลกที่มีการเปลี่ยนแปลง : แฟลกศูนย์ หรือ Z

ตัวอย่าง : clrf FLAG_REG

ก่อนกระทำคำสั่ง

FLAG_REG = 0x5A

หลังกระทำคำสั่ง

FLAG_REG = 0x00 , Z = 1

คำสั่ง clrw

รูปแบบ : clrw

การทำงาน : 00H - > (W) , 1 - > Z

แฟลกที่มีการเปลี่ยนแปลง : แฟลกศูนย์ หรือ Z

ตัวอย่าง : clrw

ก่อนกระทำคำสั่ง W = 0x5A

หลังกระทำคำสั่ง W = 0x00 , Z = 1

คำสั่ง comf (complement register)

เป็นคำสั่งใช้กลับข้อมูล " 0 " ให้เป็น " 1 " และกลับข้อมูล " 1 " ให้เป็น " 0 " ของรีจิสเตอร์ไฟล์ใดๆ หรือเรียกว่า **การทำคอมพลีเมนต์** หลังจากที่กระทำคำสั่งนี้แล้ว ค่าของรีจิสเตอร์ไฟล์ที่ถูกกำหนดจะมีค่าตรงกันข้าม เช่น จากเดิมมีค่า 1011 0011 หรือ 0XB3 จะกลายเป็น 0100 1100 หรือ 0x4C เป็นต้น สำหรับค่าที่ทำการคอมพลีเมนต์จะเลือกที่เก็บได้ 2 แห่ง คือ เก็บที่รีจิสเตอร์ W โดยการเขียนข้อมูล " 0 " ต่อท้ายรีจิสเตอร์ไฟล์ที่ต้องการทำคอมพลีเมนต์หรือเก็บที่รีจิสเตอร์ไฟล์ตัวเดิมที่ต้องการทำคอมพลีเมนต์โดยการเขียนข้อมูล " 1 " ต่อท้าย สำหรับรายละเอียดและรูปแบบมีดังนี้

รูปแบบ : comf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : f - > (W) หรือ f - > (f)

แฟลกที่มีการเปลี่ยนแปลง : Z

ตัวอย่าง : 1. Comf REG1 , 0

ก่อนกระทำคำสั่ง REG1 = 0x13

หลังกระทำคำสั่ง REG1 = 0x13

W = 0xEC

2. comf REG1 , 1

ก่อนกระทำคำสั่ง REG1 = 0x13

หลังกระทำคำสั่ง REG1 = 0xEC

คำสั่ง decf (decrement register)

เป็นคำสั่งลดค่าของรีจิสเตอร์ไฟล์ใดๆลงหนึ่งค่า สามารถเก็บค่าได้ 2 แห่งคือ ที่รีจิสเตอร์ W โดยการเขียนข้อมูล " 0 " ต่อกับรีจิสเตอร์ไฟล์ที่ต้องการลดค่า และที่รีจิสเตอร์ไฟล์ตัวเดิม โดยการเขียนข้อมูล " 1 " ต่อท้ายรีจิสเตอร์ไฟล์ที่ต้องการลดค่า สำหรับรูปแบบและรายละเอียดมีดังนี้

รูปแบบ : decf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : (f) - 1 - > (W) เมื่อ d = 0

(f) - 1 - > (f) เมื่อ d = 1

1 - > Z เมื่อผลลัพธ์ลดค่าเป็นศูนย์

แฟลกที่มีการเปลี่ยนแปลง : Z

ตัวอย่าง : 1. decf CNT , 0

ก่อนกระทำคำสั่ง CNT = 0X01 , Z = 0

หลังกระทำคำสั่ง CNT = 0x00 , W = 0x00 , Z = 1

2. decf CNT , 1

ก่อนกระทำคำสั่ง CNT = 0X01 , Z = 0

หลังกระทำคำสั่ง CNT = 0x00 , Z = 1

คำสั่ง incf (increment register)

เป็นคำสั่งที่ใช้เพิ่มค่าของรีจิสเตอร์ใดๆขึ้นหนึ่งค่า การเก็บผลลัพธ์เหมือนกับคำสั่ง decf ทุกประการ สำหรับรูปแบบและรายละเอียดมีดังนี้

รูปแบบ : incf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : (f) + 1 - > (W) **เมื่อ** d = 0

(f) + 1 - > (f) **เมื่อ** d = 1

1 - > Z **เมื่อผลลัพธ์ลดค่าเป็นศูนย์**

แฟลกที่มีการเปลี่ยนแปลง : Z

ตัวอย่าง : 1. Incf CNT , 1

ก่อนกระทำคำสั่ง CNT = 0XFF , Z = 0

หลังกระทำคำสั่ง CNT = 0x00 , Z = 1

คำสั่ง bcf (clear bit in register)

เป็นรายละเอียดคำสั่งเคลียร์บิตในรีจีสเตอร์ไฟล์ที่กำหนดให้เป็น " 0 " มีรูปแบบรายละเอียดดังนี้

รูปแบบ : bcf f, b

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$0 \leq b \leq 7$

การทำงาน : 0 - > (f < b >)

ตัวอย่าง : bcf FLAG_REG

ก่อนกระทำคำสั่ง

FLAG_REG = 0XC7 (11000111)

หลังกระทำคำสั่ง

FLAG_REG = 0x47 , (01000111)

บิตที่ 7 **ของ** FLAG_REG **ถูกเคลียร์เป็น " 0 "**

คำสั่ง bsf (set bit in register)

เป็นคำสั่งเซตบิตในรีจิสเตอร์ไฟล์ที่กำหนดให้เป็น " 1 " มีรูปแบบและรายละเอียดดังนี้

รูปแบบ : bsf f, b

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$0 \leq b \leq 7$

การทำงาน : 1 - > (f < b >)

ตัวอย่าง : bsf FLAG_REG

ก่อนกระทำคำสั่ง

FLAG_REG = 0X0A (11000111)

หลังกระทำคำสั่ง

FLAG_REG = 0x8A , (01000111)

บิตที่ 7 ของ FLAG_REG ถูกเซ็ตเป็น " 1 "

คำสั่ง rlf (rotate left register though carry)

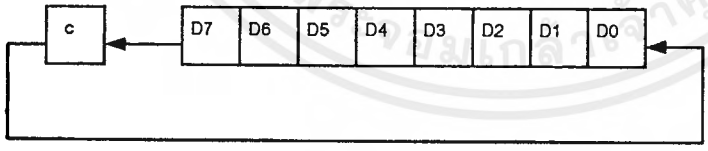
เป็นคำสั่งหมุนข้อมูลในรีจิสเตอร์ไปทางซ้ายเข้าสู่แฟลกทด (carry flag : C) มีรูปแบบรายละเอียดดังนี้

รูปแบบ : rlf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน :



แฟลกที่มีการเปลี่ยนแปลง : c

ตัวอย่าง : 1. rlf REG1 , 0

ก่อนกระทำคำสั่ง

REG1 = 1110 0110 , C = 0

หลังกระทำคำสั่ง

REG1 = 1110 0110

W = 1100 1100 , C = 0

2. rlf REG1 , 1

ก่อนกระทำคำสั่ง

REG1 = 1110 0110 , C = 0

หลังกระทำคำสั่ง

REG1 = 1100 1100 , C =1

คำสั่ง rrf (rotate right register through carry)

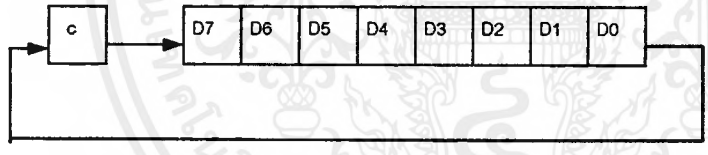
เป็นคำสั่งหมุนข้อมูลไปทางขวาผ่านเข้าไปในแฟลกทด มีการทำงานตรงข้าม rlf สำหรับรูปแบบและรายละเอียดดังนี้

รูปแบบ : rrf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน :



แฟลกที่มีการเปลี่ยนแปลง : C

ตัวอย่าง : 1. rrf REG1 , 0

ก่อนกระทำคำสั่ง

REG1 = 1110 0110 , C = 0

หลังกระทำคำสั่ง

REG1 = 1110 0110

W = 0111 0011 , C = 0

2. rrf REG1 , 1

ก่อนกระทำคำสั่ง

REG1 = 1110 0110 , C = 0

หลังกระทำคำสั่ง

REG1 = 0111 0011 , C =0

คำสั่ง swapf (swap register)

เป็นคำสั่งสลับข้อมูล 4 บิตบนหรือ 4 บิตล่างของรีจิสเตอร์ ผลของการสลับข้อมูลจะเก็บไว้ในรีจิสเตอร์ W หรือรีจิสเตอร์ตัวเดิมที่ทำการสลับค่าก็ได้ โดยการกำหนดข้อมูล " 0 " หรือ " 1 " ต่อเข้ากับรีจิสเตอร์ที่ต้องการสลับข้อมูล สำหรับข้อมูลและรายละเอียดดังนี้

รูปแบบ : swap f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : กรณี d = 0

(f<3:0>) -> (w<7:4>)

(f<7:4>) -> (w<3:0>)

กรณี d = 1

(f<3:0>) -> (f<7:4>)

(f<7:4>) -> (f<3:0>)

ตัวอย่าง : 1.swapf REG1,0

ก่อนกระทำคำสั่ง

REG1 = 0xA5

หลังกระทำคำสั่ง

REG1 = 0xA5

W = 0x5A

2. swapf REG1,1

ก่อนกระทำคำสั่ง REG1 = 0xA5

หลังกระทำคำสั่ง REG1 = 0x5A

กลุ่มคำสั่งการกระโดดและควบคุมลำดับการทำงานของโปรแกรม

มีด้วยกัน 9 คำสั่ง คือ goto , call , return , retlw , retfile , btfsc , btfss , decfsz และ incfsz

คำสั่ง goto (go to address)

เป็นคำสั่งที่กำหนดให้ซีพียูกระโดดไปทำงานยังแอดเดรสที่กำหนดให้ในหน่วยความจำโปรแกรมโดยไม่มีเงื่อนไขค่าที่กำหนดให้กระโดดไปนี้ต้องไม่เกิน 2047 รูปแบบและรายละเอียดมีดังนี้

รูปแบบ : goto k

โอเปอร์แรนด์ : $0 \leq f \leq 2047$

การทำงาน : k - > (PC < 10:0 >)

(PCLACTH < 4 :3 >) - > (PC < 12:11 >)

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 2

ตัวอย่าง : goto there

หลังกระทำคำสั่ง

PC = ค่าของแอดเดรส there

คำสั่ง call (subroutine call)

เป็นคำสั่งที่กำหนดให้ซีพียูกระโดดไปทำงานที่โปรแกรมย่อย โดยไม่มีเงื่อนไขจะกลับมายังโปรแกรมหลักก็ต่อเมื่อพบคำสั่ง return หรือ retlw เมื่อกระทำคำสั่งนี้ ซีพียูจะทำการเก็บค่า PC ไว้ในสแต็กโดยอัตโนมัติ หลังจากที่ซีพียูทำงานในโปรแกรมย่อยเสร็จ ก็จะกลับมาอ่าน PC จากสแต็กแล้วกลับไปทำงานต่ออย่างถูกต้อง

รูปแบบ : call k

โอเปอร์แรนด์ : $0 \leq f \leq 2047$

การทำงาน : (PC) + 1 - > TOS

k - > (PC < 10:0 >)

(PCLACTH < 4 :3 >) - > (PC < 12:11 >)

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 2

ตัวอย่าง : HERE call HERE

ก่อนกระทำคำสั่ง

PC = แอดเดรสของ HERE

หลังกระทำคำสั่ง

PC = แอดเดรสของ HERE

TOS = แอดเดรสของ HERE

คำสั่ง return (return from subroutine)

เป็นคำสั่งที่แจ้งให้ซีพียูทราบว่าสิ้นสุดการทำงานของโปรแกรมย่อยแล้ว เมื่อซีพียูกระทำคำสั่งแล้ว จะไปอ่านค่า PC จากสแต็กที่นำไปเก็บไว้ชั่วคราวก่อนหน้านี้ที่กระทำคำสั่ง call จากนั้นกระโดดไปยังแอดเดรสที่กำหนดโดย PC ในการเขียนย่อยทุกครั้งต้องมีคำสั่ง return และ retlw ต่อท้ายเสมอ

รูปแบบ : return

โอเปอร์แรนด์ : ไม่มี

การทำงาน : TOS - > PC

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 2

ตัวอย่าง : return

หลังจากกระทำคำสั่ง PC = TOS

คำสั่ง retlw (return from subroutine and literal to W register)

เป็นคำสั่งที่ใช้ในการออกจากโปรแกรมย่อยอีกแบบหนึ่งที่มีความแตกต่างจากคำสั่ง return ตรงที่หลังกระทำคำสั่งนี้แล้วซีพียูจะไปอ่าน PC ในสแต็ก เพื่อกระโดดกลับไปทำงานต่อที่แอดเดรสที่กำหนดไว้ใน PC อันเป็นแอดเดรสที่ต่อจากคำสั่งที่กำหนดให้ซีพียูกระโดดมาทำงานที่โปรแกรมย่อย พร้อมกันนั้นยังมีการโอนย้ายค่าคงที่ที่กำหนดไว้ต่อท้ายคำสั่ง retlw ไปเก็บไว้ใน register W ด้วย

ดังนั้นคำสั่งนี้จึงมีประโยชน์มากสำหรับโปรแกรมย่อยที่เกี่ยว ข้อง กับการเปิดตารางเพื่อค้นหาหรือแปลงข้อมูล หรือที่เรียกว่า look up table สำหรับรูปแบบและรายละเอียดมีดังนี้

รูปแบบ : retlw k

โอเปอร์แรนด์ : $0 \leq f \leq 255$

การทำงาน : k - > W

TOS - > PC

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 2

ตัวอย่าง :

```
        call TABLE ; รีจิสเตอร์ W เก็บค่าออฟเซ็ตของตาราง
        .
        .
TABLE   addwf PC
        retlw k1 ; เริ่มตาราง
        retlw k2
        .
        .
        .
        retlw kn ; สิ้นสุดตาราง
        ก่อนกระทำคำสั่ง W = 0x07
        หลังกระทำคำสั่ง W = ค่าของ kn
```

คำสั่ง retfile (return form interrupt service routine)

เป็นคำสั่งที่ใช้ในการออกจากโปรแกรมย่อยของการบริการอินเตอร์รัปต์ หลังจากกระทำคำสั่งนี้ซีพียูจะกลับไปทำงานยังแอดเดรสที่ถัดจากแอดเดรสสุดท้ายที่ทำงานก่อนเกิดการอินเตอร์รัปต์โดยค่าของแอดเดรสนี้จะเก็บไว้ในสแต็กโดยอัตโนมัติ หลังจากที่เกิดการอินเตอร์รัปต์ขึ้น พร้อมกันนั้นยังทำการเซต GIE ด้วย เพื่อทำการอนาเบิลการตอบสนองการอินเตอร์รัปต์ของซีพียูอีกครั้งหลังจากถูกดิสเอเบิลในขณะที่ทำการอินเตอร์รัปต์ สำหรับรูปแบบ และ รายละเอียดมีดังนี้

รูปแบบ : retfile

โอเปอร์แรนด์ : ไม่มี

การทำงาน : TOS - > (PC) , 1 - > GIE

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 2

ตัวอย่าง : retfile

หลังจากกระทำคำสั่ง

PC = TOS , GIE = 1

คำสั่ง btfsc (bit test register , skip if clear)

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขแบบหนึ่งที่จะทำการตรวจสอบบิตที่กำหนดในรีจิสเตอร์ก่อน หากตรงตามเงื่อนไขก็จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรส แล้วจึงทำงานต่อที่แอดเดรสถัดจากเงื่อนไขการตรวจสอบคือ บิตที่ตรวจสอบนั้นเป็น " 0 " หรือไม่ ถ้าใช่ก็จะกระโดดไม่ทำงานในแอดเดรสถัดไป แต่ถ้าไม่ใช่นั่นคือ ค่าเป็น " 1 " ซีพียูก็จะทำงานในแอดเดรสต่อไปตามปกติลักษณะของการทำงานในกรณีหลังนี้จะทำ ให้ผลการทำงานเหมือนกับคำสั่ง NOP

สำหรับจำนวนไซเกิลที่ใช้ หากตรวจสอบตรงตามเงื่อนไขจะใช้ 2 ไซเกิล แต่ถ้าไม่ตรงจะใช้เพียงไซเกิลเดียวสำหรับรูปแบบและรายละเอียดมีดังนี้

รูปแบบ : btfsc f, b

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$0 \leq b \leq 7$

การทำงาน : กระโดดข้ามแอดเดรส 1 ตำแหน่งถ้า (f < b >) = 0

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1 (2)

ตัวอย่าง : HERE btfsc FLAG, 1

FALSE goto CODE

TRUE

ก่อนกระทำคำสั่ง

PC = แอดเดรสของ HERE

หลังกระทำคำสั่ง

ถ้าบิต 1 ของ FLAG = 0 , PC = แอดเดรสของ TRUE

ถ้าบิต 1 ของ FLAG = 0 , PC = แอดเดรสของ FALSE

คำสั่ง btfss (bit test register , skip if set)

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขแบบหนึ่ง ที่จะทำการตรวจสอบบิตที่กำหนดในรีจิสเตอร์เสียก่อน หากตรงตามเงื่อนไขก็จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรส แล้วจึงทำงานต่อที่แอดเดรสถัดจากนั้น เงื่อนไขการตรวจสอบคือบิตที่ตรวจสอบนั้นเป็น " 1 " หรือไม่ ถ้าใช่ก็จะกระโดดไม่ทำงานในแอดเดรสถัดไป แต่ถ้าไม่ใช่นั่นคือ ค่าเป็น " 0 " ซีพียูก็จะทำงานในแอดเดรสต่อไปตามปกติลักษณะของการทำงานในกรณีหลังนี้จะทำ ให้ผลการทำงานเหมือนกับคำสั่ง NOP

สำหรับจำนวนไซเกิลที่ใช้ หากตรวจสอบตรงตามเงื่อนไขจะใช้ 2 ไซเกิล แต่ถ้าไม่ตรงจะใช้เพียงไซเกิลเดียว คำสั่ง btfss มีลักษณะการตรวจสอบเงื่อนไขของการกระโดดที่ตรงข้ามกับคำสั่ง btfsc สำหรับรูปแบบและรายละเอียดมีดังนี้

รูปแบบ : btfss f, b

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$0 \leq b \leq 7$

การทำงาน : กระโดดข้ามแอดเดรส 1 ตำแหน่งถ้า (f < b >) = 1

จำนวนไซเคิล : 1 (2)

ตัวอย่าง :

```
HERE  btfsc   FLAG, 1
FALSE goto    CODE
TRUE
```

ก่อนกระทำคำสั่ง

PC = แอดเดรสของ HERE

หลังกระทำคำสั่ง

ถ้าบิต 1 ของ FLAG = 1 , PC = แอดเดรสของ TRUE

ถ้าบิต 1 ของ FLAG = 0 , PC = แอดเดรสของ FALSE

คำสั่ง decfsz : decrement register , skip if zero

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขแบบหนึ่ง ที่จะทำการลดค่าในรีจิสเตอร์ลงหนึ่งค่า แล้วตรวจสอบว่าเป็น " 0 " หรือไม่ หากตรงตามเงื่อนไขก็จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรส แล้วจึงทำงานต่อที่แอดเดรสถัดจากนั้นค่าของรีจิสเตอร์ที่ทำการลดค่าลงแล้วจะถูกเก็บไว้ในรีจิสเตอร์ W หรือรีจิสเตอร์ตัวเดิมได้ ทั้งนี้ขึ้นอยู่กับการกำหนดข้อมูลที่ต่อจากรีจิสเตอร์

สำหรับจำนวนไซเกิลที่ใช้ หากตรวจสอบตรงตามเงื่อนไขจะใช้ 2 ไซเกิล แต่ถ้าไม่ตรงจะใช้เพียงไซเกิลเดียวสำหรับรูปแบบและรายละเอียดมีดังนี้

รูปแบบ : decfsz f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : กรณี d = 0

(f) - 1 - > (W) , กระโดดข้ามหนึ่งแอดเดรสถ้าผลของการลดค่าเป็น 0

กรณี d = 1

(f) - 1 - > (f) , กระโดดข้ามหนึ่งแอดเดรสถ้าผลของการลดค่าเป็น 0

จำนวนไซเคิล : 1 (2)

ตัวอย่าง :

```
HERE  decfsz  CNT, 1
      goto    LOOP
      CONT    .
              .
ก่อนกระทำคำสั่ง
PC = แอดเดรสของ LOOP
หลังกระทำคำสั่ง CNT = CNT - 1
ถ้า CNT = 0 , PC = แอดเดรสของ CONT
ถ้า CNT ≠ 0 , PC = แอดเดรสของ HERE + 1
```

คำสั่ง incfsz (increment register , skip if zero)

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขแบบหนึ่ง ที่จะทำการลดค่าในรีจิสเตอร์ลงหนึ่งค่า แล้วตรวจสอบว่าเป็น " 0 " หรือไม่ หากตรงตามเงื่อนไขก็จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรสแล้วจึงทำงานต่อที่แอดเดรสถัดจากนั้นค่าของรีจิสเตอร์ที่ทำการลดค่าลงแล้วจะถูกเก็บไว้ในรีจิสเตอร์ W หรือรีจิสเตอร์ตัวเดิมได้ ทั้งนี้ขึ้นอยู่กับการกำหนดข้อมูลที่ต่อจากรีจิสเตอร์

สำหรับจำนวนไซเกิลที่ใช้ หากตรวจสอบตรงตามเงื่อนไขจะใช้ 2 ไซเกิล แต่ถ้าไม่ตรงจะใช้เพียงไซเกิลเดียวสำหรับรูปแบบและรายละเอียดมีดังนี้

รูปแบบ : incfsz f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : กรณี d = 0

(f) + 1 - > (W) , กระโดดข้ามหนึ่งแอดเดรสถ้าผลของการลดค่าเป็น 0

กรณี d = 1

(f) + 1 - > (f) , กระโดดข้ามหนึ่งแอดเดรสถ้าผลของการลดค่าเป็น 0

จำนวนไซเคิล : 1 (2)

ตัวอย่าง :

```
HERE  incfsz   CNT, 1
      goto     LOOP
      CONT  .
            .
```

ก่อนกระทำคำสั่ง

PC = แอดเดรสของ LOOP

หลังกระทำคำสั่ง CNT = CNT + 1

ถ้า CNT = 0 , PC = แอดเดรสของ CONT

ถ้า CNT ≠ 0 , PC = แอดเดรสของ HERE + 1

กลุ่มคำสั่งควบคุมไมโครคอนโทรลเลอร์

มีทั้งสิ้น 5 คำสั่ง คือ clrwdt , option , sleep , tris และ nop แต่เหมาะที่จะใช้กับ PIC16F84 เพียง 3 คำสั่งคือ clrwdt , sleep และ nop แต่การอธิบายจะอธิบายทุกคำสั่ง ทั้งนี้เพราะคำสั่ง optionและ tris จะมีใช้ใน PIC16C5x

คำสั่ง clrwdt : clear Watchdog Timer

เป็นคำสั่งที่ใช้ในการเคลียร์ค่าของวอตช์ด็อกไทเมอร์และปรีสเกลเลอร์ภายในตัวไมโครคอนโทรลเลอร์ พร้อมกับเซตบิต $\overline{TO}$ และ $\overline{PD}$ สำหรับนูปแบบรายละเอียดมีดังนี้ คือ

รูปแบบ : clrwdt

โอเปอร์แรนด์ : ไม่มี

การทำงาน : 00H - > WDT

0 - > WDT ปรีสเกลเลอร์

1 - > $\overline{TO}$, 1 - > $\overline{PD}$

บิตสถานะที่มีการเปลี่ยนแปลง : $\overline{TO}$ และ $\overline{PD}$

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1

ตัวอย่าง : clrwdt

ก่อนกระทำคำสั่ง

WDT เคาน์เตอร์ = ?

หลังกระทำคำสั่ง

WDT เคาน์เตอร์ = 0x00

WDT เคาน์เตอร์ = 0

$\overline{TO}$ = 1 และ $\overline{PD}$ = 1

คำสั่ง sleep (goto standby mode)

เป็นคำสั่งที่กำหนดให้ซีพียูทำงานในโหมดสลีป เพื่อประหยัดพลังงาน หลังจากกระทำคำสั่งนี้ค่าการนับภายในวอตช์ด็อกไทเมอร์จะถูกเคลียร์ให้เป็นศูนย์ พร้อมๆกับค่าของปรีสเกลเลอร์ภายในวอตช์ด็อกไทเมอร์ด้วย ในขณะที่ $\overline{TO}$ จะถูกเซต และ $\overline{PD}$ จะถูกเคลียร์ให้เป็นศูนย์ เพื่อแจ้งให้ทราบว่า ขณะนี้ซีพียู อยู่ในโหมดสลีปแล้ว สำหรับรายละเอียดมีดังนี้

รูปแบบ : sleep

การทำงาน : 00H - > WDT

0 - > WDT ปรีสเกลเลอร์

1 - > $\overline{TO}$, 1 - > $\overline{PD}$

บิตสถานะที่มีการเปลี่ยนแปลง : $\overline{TO}$ และ $\overline{PD}$

จำนวนเวิร์ด : 1

กลุ่มคำสั่งประมวลผลทางลอจิก

มีด้วยกัน 6 คำสั่งคือ andlw , andwf , iorlw , xorlw และ xorwf

คำสั่ง andlw และ andwf (AND operations)

เป็นคำสั่งที่ใช้ในการ " แอนด์ " ค่าของข้อมูลโดย andlw จะเป็นการแอนด์ค่าคงที่กับค่าในรีจิสเตอร์ W แล้วผลของการแอนด์จะเก็บไว้ในรีจิสเตอร์ W ส่วน andwf จะเป็นการแอนด์ระหว่างค่ารีจิสเตอร์ใดๆกับรีจิสเตอร์ W ผลของการประมวลผลสามารถเลือกเก็บได้ว่าจะให้ไปเก็บที่รีจิสเตอร์ที่ทำการแอนด์หรือรีจิสเตอร์ W สำหรับรูปแบบ และรายละเอียดมีดังนี้

คำสั่ง andlw

รูปแบบ : andlw k

โอเปอร์แรนด์ : $0 \leq k \leq 255$

การทำงาน : (W) AND (k) - > (W)

บิตสถานะที่มีการเปลี่ยนแปลง : z

ตัวอย่าง : andlw 0x5F

ก่อนกระทำคำสั่ง W = 0xA3

หลังกระทำคำสั่ง W = 0x03

คำสั่ง andwf

รูปแบบ : andwf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : กรณี d = 0

(W) AND (f) - > (W)

กรณี d = 1

(W) AND (f) - > (f)

บิตสถานะที่มีการเปลี่ยนแปลง : z

ตัวอย่าง : 1. andwf FSR , 0

ก่อนกระทำคำสั่ง

W = 0x17 , FSR = 0xC2

หลังกระทำคำสั่ง

W = 0x02 , FSR = 0xc2

2. andwf FSR , 1

ก่อนกระทำคำสั่ง

W = 0x17 , FSR = 0xc2

หลังกระทำคำสั่ง

W = 0x02 , FSR = 0x02

คำสั่ง iorlw และ iorwf (OR operations)

เป็นคำสั่งที่ใช้ในการ " ออร์ " ค่าของข้อมูลโดย iorlw จะเป็นการออร์ค่าคงที่กับค่าในรีจิสเตอร์ W แล้วผลของการออร์จะเก็บไว้ในรีจิสเตอร์ W ส่วน iorwf จะเป็นการออร์ระหว่างค่ารีจิสเตอร์ใดๆกับรีจิสเตอร์ W ผลของการประมวลผลสามารถเลือกเก็บได้ว่าจะให้ไปเก็บ ที่รีจิสเตอร์ที่ทำการออร์หรือรีจิสเตอร์ W สำหรับรูปแบบ และรายละเอียดมีดังนี้

คำสั่ง iorlw

รูปแบบ : iorlw k

โอเปอร์แรนด์ : $0 \leq k \leq 255$

การทำงาน : (W) AND (k) - > (W)

บิตสถานะที่มีการเปลี่ยนแปลง : Z

ตัวอย่าง : iorlw 0x35

ก่อนกระทำคำสั่ง W = 0x9A

หลังกระทำคำสั่ง W = 0xBF

คำสั่ง iorwf

รูปแบบ : iorwf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : กรณี d = 0

(W) OR (f) - > (W)

กรณี d = 1

(W) OR (f) - > (f)

บิตสถานะที่มีการเปลี่ยนแปลง : z

ตัวอย่าง : 1. iorwf RESULT , 0

ก่อนกระทำคำสั่ง

W = 0x91 , RESULT = 0x13

หลังกระทำคำสั่ง

W = 0x93 , RESULT = 0x13

2. iorwf RESULT , 1

ก่อนกระทำคำสั่ง

W = 0x91 , RESULT = 0x13

หลังกระทำคำสั่ง

W = 0x91 , RESULT = 0x93

คำสั่ง xorlw และ worwf (XOR operations)

เป็นคำสั่งที่ใช้ในการ "เอ็กคลูซีพ-ออร์ " ค่าของข้อมูลโดย xorlw จะเป็นการเอ็กคลูซีพ-ออร์ค่าคงที่กับค่าในรีจิสเตอร์ W แล้วผลของการเอ็กคลูซีพ- ออร์จะเก็บไว้ในรีจิสเตอร์ W ส่วน xorwf จะเป็นการเอ็กคลูซีพ- ออร์ระหว่างค่า รีจิสเตอร์ใดๆกับรีจิสเตอร์ W ผลของการประมวลผลสามารถเลือกเก็บได้ว่าจะให้ไปเก็บ ที่รีจิสเตอร์ที่ทำการเอ็กคลูซีพ- ออร์หรือรีจิสเตอร์ W สำหรับรูปแบบ และรายละเอียดมีดังนี้

คำสั่ง xorlw

รูปแบบ : xorlw k

โอเปอร์แรนด์ : $0 \leq k \leq 255$

การทำงาน : (W) XOR (k) - > (W)

บิตสถานะที่มีการเปลี่ยนแปลง : z

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1

ตัวอย่าง : xorlw 0xAF

ก่อนกระทำคำสั่ง W = 0xB5

หลังกระทำคำสั่ง W = 0x1A

คำสั่ง xorwf

รูปแบบ : xorwf f , d

โอเปอร์แรนด์ : $0 \leq f \leq 127$

$d \in [0 , 1]$

การทำงาน : **กรณี** d = 0

(W) XOR (f) - > (W)

กรณี d = 1

(W) XOR (f) - > (f)

บิตสถานะที่มีการเปลี่ยนแปลง : Z

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1

ตัวอย่าง : 1. xorwf REG1 , 0

ก่อนกระทำคำสั่ง

W = 0xB5 , REG1 = 0xAF

หลังกระทำคำสั่ง

W = 0x1A , REG1 = 0xAF

2. xorwf Reg1 , 1

ก่อนกระทำคำสั่ง

W = 0xB5 , REG1 = 0xAF

หลังกระทำคำสั่ง

W = 0xB5 , REG1 = 0x1A

กลุ่มคำสั่งการประมวลผลทางคณิตศาสตร์

มีด้วยกัน 4 คำสั่ง แบ่งเป็นคำสั่งเกี่ยวกับการบวก 2 คำสั่ง คือ addlw และ addwf คำสั่งที่เกี่ยวกับการลบ 2 คำสั่ง คือ sublw และ subwf

คำสั่ง addlw และ addwf (ADD operations)

เป็นคำสั่งที่ใช้ทำการบวกค่าของข้อมูล โดย addlw จะเป็นการบวกค่าคงที่กับค่าใน รีจิสเตอร์ W แล้วผลของการบวกจะเก็บไว้ในรีจิสเตอร์ W ส่วน addwf จะเป็นการบวกระหว่างค่า รีจิสเตอร์ใดๆกับรีจิสเตอร์ W ผลของการประมวลผลสามารถเลือกเก็บได้ว่าจะให้ไปเก็บ ที่รีจิส เตอร์ที่ทำการบวกหรือรีจิสเตอร์ W สำหรับรูปแบบ และรายละเอียดมีดังนี้

คำสั่ง addlw

รูปแบบ : addlw k

โอเปอร์แรนด์ : $0 \leq k \leq 255$

การทำงาน : (W) + (k) - > (W)

บิตสถานะที่มีการเปลี่ยนแปลง : C , DC , z

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1

ตัวอย่าง : addwf 0x1D

ก่อนกระทำคำสั่ง

W = 0x00

หลังกระทำคำสั่ง

W = 0x1D

คำสั่ง addwf

รูปแบบ : addwf k

โอเปอร์แรนด์ : $0 \leq k \leq 127$

$d \in [0 , 1]$

การทำงาน : กรณี d = 0

(W) + (f) - > (W)

กรณี d = 1

(W) + (f) - > (f)

บิตสถานะที่มีการเปลี่ยนแปลง : C , DC , z

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1

ตัวอย่าง : 1. addwf REG1 , 0

ก่อนกระทำคำสั่ง

W = 0x17 , REG1 = 0xC2

หลังกระทำคำสั่ง

W = 0xD9 , REG1 = 0xC2

2. addwf REG1 , 1

ก่อนกระทำคำสั่ง

W = 0x17 , REG1 = 0xC2

หลังกระทำคำสั่ง

W = 0x17 , REG1 = 0xD9

คำสั่ง sublw **และ** subwf (Subtractor operations)

เป็นคำสั่งที่ใช้ทำการลบค่าของข้อมูล โดย sublw จะเป็นการลบค่าคงที่กับค่าใน รีจิสเตอร์ W แล้วผลของการลบจะเก็บไว้ในรีจิสเตอร์ W ส่วน subwf จะเป็นการลบระหว่างค่า รีจิสเตอร์ใดๆกับรีจิสเตอร์ W ผลของการประมวลผลสามารถเลือกเก็บได้ว่าจะให้ไปเก็บ ที่รีจิส เตอร์ที่ทำการลบหรือรีจิสเตอร์ W สำหรับรูปแบบ และรายละเอียดมีดังนี้

คำสั่ง sublw

รูปแบบ : sublw k

โอเปอร์แรนด์ : $0 \leq k \leq 255$

การทำงาน : (k) - (w) - > (W)

บิตสถานะที่มีการเปลี่ยนแปลง : C , DC , z

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1

ตัวอย่าง : sublw 0x02

กรณีที่ 1

ก่อนกระทำคำสั่ง

W = 0x01

หลังกระทำคำสั่ง

W = 0x01 , C = 1

ผลลัพธ์เป็นค่าบวก

กรณีที่ 2

ก่อนกระทำคำสั่ง

W = 0x02

หลังกระทำคำสั่ง

W = 0x00 , C = 1 , Z = 1

ผลลัพธ์มีค่าเท่ากับศูนย์

กรณีที่ 3

ก่อนกระทำคำสั่ง

W = 0x03

หลังกระทำคำสั่ง

W = 0xFF , C = 0

ผลลัพธ์เป็นค่าลบ

คำสั่ง subwf

รูปแบบ : subwf f,d

โอเปอร์แรนด์ : $0 \leq k \leq 255$

การทำงาน : กรณี d = 0 , (f) - (w) - > (W)

กรณี d = 1 , (f) - (w) - > (f)

บิตสถานะที่มีการเปลี่ยนแปลง : C , DC , z

จำนวนเวิร์ด : 1

จำนวนไซเคิล : 1

ตัวอย่าง : subwf REG1 , 0

กรณีที่ 1

ก่อนกระทำคำสั่ง

W = 0x02 , REG1 = 0x03

หลังกระทำคำสั่ง

W = 0x01 , C = 1

ผลลัพธ์เป็นค่าบวก

กรณีที่ 2

ก่อนกระทำคำสั่ง

W = 0x02 , REG1 = 0x02

หลังกระทำคำสั่ง

W = 0x00 , C = 1 , Z = 1

ผลลัพธ์มีค่าเท่ากับศูนย์

กรณีที่ 3

ก่อนกระทำคำสั่ง

W = 0x02 , REG1 = 0x01

หลังกระทำคำสั่ง

W = 0xFF , C = 0

ผลลัพธ์เป็นค่าลบ

การจัดกลุ่มคำสั่งตามที่กำหนดโดย MPASM

สำหรับการจัดกลุ่มคำสั่งตามข้อกำหนดของ MPASM นั้นได้ทำการสรุปไว้แล้วตามที่แสดงในตารางที่ 3-1 จะเห็นได้ว่า มีการแบ่งกลุ่มคำสั่งออกเป็น 3 กลุ่ม ดังนี้

1) กลุ่มคำสั่งจัดการข้อมูลระดับไบต์ มีทั้งสิ้น 18 คำสั่ง

2) กลุ่มคำสั่งจัดการข้อมูลระดับบิต มีทั้งสิ้น 4 คำสั่ง

3) กลุ่มคำสั่งจัดการข้อมุลค่าคงที่และควบคุมการทำงาน มีทั้งสิ้น 13 คำสั่ง

จะเห็นได้ว่าไม่มีคำสั่ง tris และ option เนื่องจาก 2 คำสั่งไม่เหมาะที่จะใช้กับ PIC16F84 ถ้าหากใช้คำสั่งนี้ในการเขียนโปรแกรม แล้วนำไปแอสเซมเบลอร์ด้วยโปรแกรม MPASM จะได้รับข้อความเตือนจากโปรแกรม MPASM แต่ก็จะยอมให้การแอสเซมเบลอร์ผ่านไปได้

ดังนั้นในการเขียนโปรแกรมจึงควรหลีกเลี่ยงคำสั่งทั้งสอง เพราะอาจส่งผลให้การทำงานของ PIC16F84 เกิดความไม่แน่นอนได้ แต่ถ้าหากพัฒนาระบบด้วย PIC16F84 คำสั่ง tris และ option มีความจำเป็นต้องใช้อย่างยิ่ง

การเข้าถึงรีจิสเตอร์และข้อมูลของ PIC16F84

ในไมคครคอนโทรลเลอร์ PIC16F84 มีกระบวนการเข้าถึงหน่วยความจำและรีจิสเตอร์อยู่ 4 แบบ คือ

1.แบบทันทีทันใด (Immediate Addressing Mode)

2. แบบโดยตรง (Direct Addressing Mode)

3. แบบโดยอ้อม (Indirect Addressing Mode)

4. แบบสัมพัทธ์ (Relative Addressing Mode)

การเข้าถึงแบบทันทีทันใด

จะเป็นการเข้าถึงค่าคงที่โดยตรง ด้วยการใช้ชุดคำสั่งที่เกี่ยวข้องกับค่าคงที่ (Literal) ร่วมกับรีจิสเตอร์ W หรือแอกคิวมูเลเตอร์ ตัวอย่างของคำสั่งที่แสดงให้เห็นถึงการเข้าถึงข้อมูลทันทีทันใดได้แก่ movlw k ที่ k คือค่าคงที่

การเข้าถึงข้อมูลแบบโดยตรง

เป็นการเข้าถึงข้อมูลหรือรีจิสเตอร์ด้วยการกำหนดแอดเดรสที่ต้องการเข้าถึงอย่างเจาะจง หรือระบุผานชื่อของรีจิสเตอร์ก็ได้ ดังตัวอย่าง

clrf TEMP เป็นคำสั่งที่ต้องการเคลียร์ค่าของรีจิสเตอร์

การเข้าถึงข้อมูลโดยอ้อม

มีรีจิสเตอร์หลายตัวในไมโครคอนโทรลเลอร์ PIC16F84 ไม่สามารถที่จะเขียนหรืออ่านข้อมูลได้โดยตรงจะต้องทำการเขียนหรืออ่านข้อมูลผ่านรีจิสเตอร์ FSR (File Select Register) หรืออาจจะกล่าวได้ว่าใช้รีจิสเตอร์ FSR เป็นตัวชี้แอดเดรสของรีจิสเตอร์หรือหน่วยความจำที่ต้องการเข้าถึง

การใช้รีจิสเตอร์ FSR จะมีลักษณะคล้ายๆกับการเข้าถึงข้อมูลแบบใช้ดัชนีหรืออินเด็กซ์ (index addressing) ของไมโครคอนโทรลเลอร์เบอร์อื่นๆ แต่เนื่องจาก PIC16F84 มีรีจิสเตอร์สำหรับใช้งานทั่วไป 68 ตัว ดังนั้นการใช้วิธีการเข้าถึงแบบโดยอ้อมอาจใช้ไม่บ่อยนัก

วิธีการเข้าถึงรีจิสเตอร์แบบนี้ จะเริ่มต้นด้วยการนำค่าของรีจิสเตอร์ที่ต้องการเข้าถึงการเก็บลงในรีจิสเตอร์ W จากนั้นนำค่าในรีจิสเตอร์ W ไปเก็บไว้ในรีจิสเตอร์ FSR เพื่อให้ FSR เป็นตัวชี้ข้อมูลที่ต้องการเข้าถึง เพื่อให้เห็นถึงวิธีการเข้าถึงข้อมูลหรือรีจิสเตอร์แบบโดยอ้อมอย่างชัด

เจนขึ้น พิจารณาโปรแกรมย่อยต่อไปนี้ ซึ่งเป็นโปรแกรมย่อยของการส่งข้อมูลตัวอักขระจากรีจิสเตอร์ไฟล์ 16 ตัว ตั้งแต่แอดเดรส 0x20-0x2F ไปแสดงที่ LCD โมดูล

```
movlw 0x20          ;กำหนดแอดเดรสที่ต้องการเข้าถึงลงในรีจิสเตอร์ W
movwf FSR           ;นำค่าจากรีจิสเตอร์ W ไปเก็บไว้ในรีจิสเตอร์ FSR ที่ชี้ตำแหน่ง
                    ;ของ INDF
movf INDF , W       ;นำค่าจากรีจิสเตอร์ INDF ซึ่งเป็นข้อมูลอักขระตัวแรกไปเก็บไว้
                    ;ในรีจิสเตอร์ W เพื่อเตรียมส่งออกไปแสดงผลที่จอ LCD
call SEND           ;กระโดดไปทำงานที่โปรแกรมย่อย send เพื่อส่งข้อมูลไปแสดง
                    ;ผล
incf FSR            ;ค่าตัวชี้ข้อมูล
btfss FSR , 4       ;ตรวจสอบว่าอ่านข้อมูลครบ 16 ตัวหรือไม่
goto NEXT           ;ถ้ายังไม่ครบกลับไปอ่านข้อมูลจากหน่วยความจำในตำแหน่ง ว
                    ;NEXT
```

จากโปรแกรมตัวอย่างจะเห็นได้ว่าการนำข้อมูลจากหน่วยความจำที่แอดเดรส 0x20-0x2F ไปแสดงผลนั้นจะผ่านทางรีจิสเตอร์ W แล้วส่งต่อไปยัง INDF โดยมีรีจิสเตอร์ FSR เป็นตัวชี้ตำแหน่งหน่วยความจำ

การเข้าถึงค่าข้อมูลแบบสัมพัทธ์

การเข้าถึงข้อมูลแบบนี้จะประกอบด้วยการคำนวณค่าสัมพัทธ์ของระยะห่างระหว่างแอดเดรสที่เริ่มต้นทำงานกับแอดเดรสที่ต้องการเข้าถึง โดยมีการใช้ค่าของโปรแกรมเคาน์เตอร์เข้ามาช่วยคำสั่งหนึ่งที่นำมาใช้ในการเข้าถึงข้อมูลสัมพัทธ์นี้คือ retlw

กระบวนการจะเริ่มต้นด้วยการกำหนดค่าออฟเซตลงในรีจิสเตอร์ W แล้วนำไปร่วมกับค่าของโปรแกรมเคาน์เตอร์ก็จะได้ค่าของแอดเดรสของหน่วยความจำที่ต้องการเข้าถึง การเข้าถึงแบบนี้มีข้อจำกัดคือ ระยะห่างของแอดเดรสเริ่มต้นกับแอดเดรสที่ต้องการเข้าถึงต้องไม่เกิน 256 ตำแหน่ง

ตัวอย่างการเข้าถึงข้อมูลแบบสัมพัทธ์คือ การเปิดตารางข้อมูลหรือ look up table ซึ่งมีตัวอย่างดังนี้

```
PC          equ   0x02          ;กำหนดแอดเดรสของ PC
```

```
INDEX     equ   0x0C          ;กำหนดแอดเดรสของ INDEX
                movlw0X02     ;กำหนดค่าเพื่อชี้ตำแหน่งลงใน W
                movwf INDEX   ;นำค่าของการชี้ตำแหน่งเก็บลงใน
                              ;INDEX
          movf  INDEX , 0     ;นำค่าของ INDEX กลับมาเก็บใน W
          call  SEGMENT       ;กระโดดไปทำงานที่ SEGMENT
SEGMENT   addwf PC , 1        ;บวกค่าของการชี้ตำแหน่งที่เก็บอยู่ใน
                              ; W เข้ากับ PC
          retlw 3F            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 06            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 53            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 4F            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 66            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 60            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 70            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 07            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 7F            ;ค่าของข้อมูลที่ต้องการเข้าถึง
          retlw 6F            ;ค่าของข้อมูลที่ต้องการเข้าถึง
```

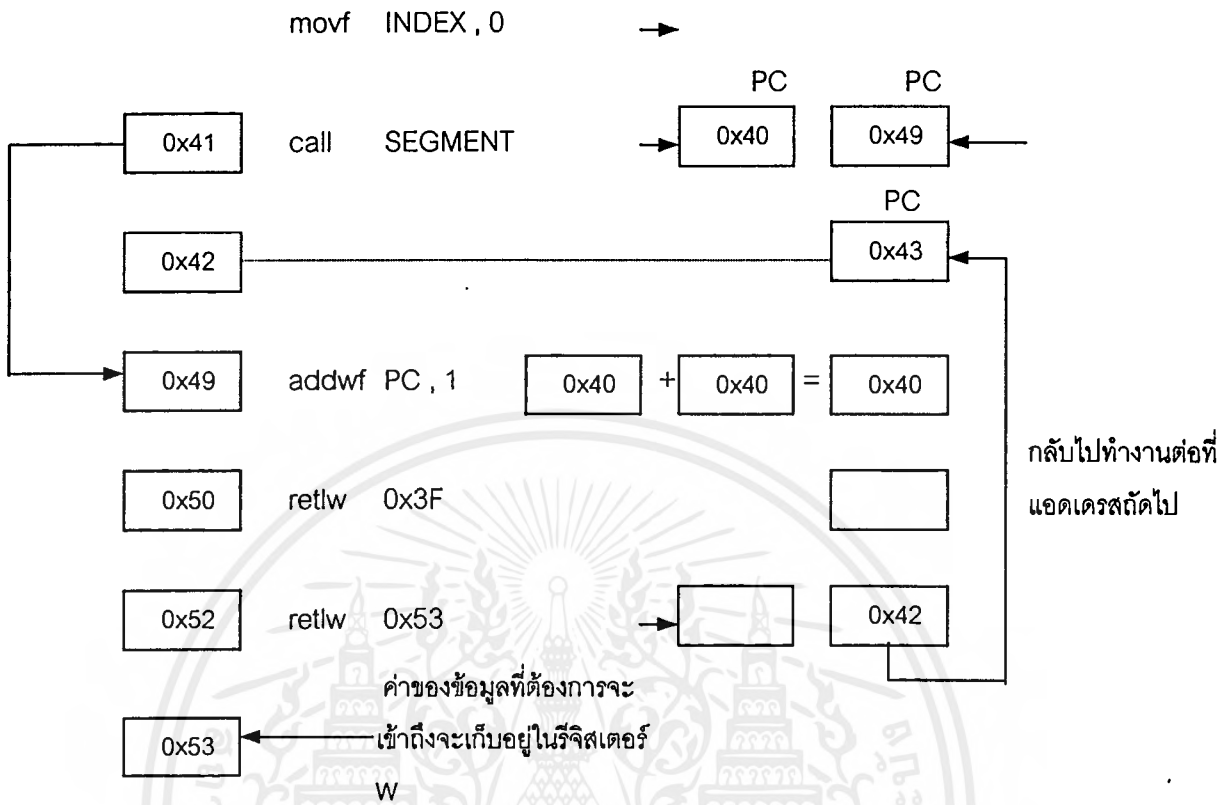
คำสั่งที่สำคัญ ในโปรแกรมย่อยคือ addwf PC , 1 คำสั่งนี้จะเป็นการกำหนดแอดเดรสที่ต้องการเข้าถึงโดยจะถูกชี้ด้วยค่าของผลรวมระหว่างโปรแกรมเคาน์เตอร์กับค่าของออฟเซ็ต หลังจากที่กระทำคำสั่งนี้แล้ว ค่าของ PC จะเท่ากับ ค่าของการชี้ตำแหน่ง (ในโปรแกรมนี้คือ 0x02 ซึ่งก็คือ 2 ในเลขฐานสิบหกนั่นเอง) รวมกับค่าของ PC ในขณะนั้น ดังนั้นค่าของ PC จะเท่ากับ 2+ (ค่าของ PC+1) จากนั้นโปรแกรมจะกระโดดไปทำงานยังตำแหน่งใหม่ที่ถูกชี้โดย PC ค่าใหม่ที่เกิดขึ้นหลังจากกระทำคำสั่ง addwf PC , 1 ซึ่งจะตรงกับแอดเดรสของข้อมูลที่ต้องการเข้าถึงที่ตำแหน่งใดตำแหน่งหนึ่งในโปรแกรมย่อย SEGMENT เพื่อให้ชัดเจนเข้าใจง่ายขึ้น พิจารณาไดอะแกรมการทำงานของโปรแกรมย่อยนี้ในรูปที่ 3-1

กลุ่มคำสั่งจัดการข้อมูลระดับไบต์																		
นีโมนิก	โอเปอร์แรนด์	รายละเอียด	ไซเกิล	ออปโค้ด 14 บิต													บิตสถานะที่เกี่ยวข้อง	หมายเหตุ
addwf	f,d	W+f	1	00	0111	d	f	f	f	f	f	f	f	f	f	f	C, DC, Z	1
andwf	f,d	W AND f	1	00	0101	d	f	f	f	f	f	f	f	f	f	f	Z	1
clrf	f	เคลียร์ค่าของ f	1	00	0001	d	f	f	f	f	f	f	f	f	f	f	Z	1
clrw	-	เคลียร์ค่าของ W	1	00	0001	0	0	0	0	0	0	1	1				Z	
comf	f,d	คอมพลีเมนต์ f	1	00	1001	d	f	f	f	f	f	f	f	f	f	f	Z	1
decf	f,d	ลดค่า f	1	00	0011	d	f	f	f	f	f	f	f	f	f	f	Z	1
decfsz	f,d	ลดค่า f, ข้ามหนึ่งเอดเดรส ถ้าค่าเป็น 0	1 (2)	00	1011	d	f	f	f	f	f	f	f	f	f	f	ไม่มี	1, 2
incf	f,d	เพิ่มค่า f	1	00	1010	d	f	f	f	f	f	f	f	f	f	f	Z	1
incfsz	f,d	เพิ่มค่า f, ข้ามหนึ่งเอดเดรส ถ้าค่าเป็น 0	1 (2)	00	1111	d	f	f	f	f	f	f	f	f	f	f	ไม่มี	1, 2
iorwf	f,d	W OR f	1	00	0100	d	f	f	f	f	f	f	f	f	f	f	Z	1
movf	f,d	โอนย้ายข้อมูลไปยัง f	1	00	1000	d	f	f	f	f	f	f	f	f	f	f	Z	1
movwf	f	โอนย้ายข้อมูลจาก W ไปยัง f	1	00	0000	1	f	f	f	f	f	f	f	f	f	f	ไม่มี	
nop	-	ไม่มีการทำงาน	1	00	0000	0	x	0	0	0	0	0	0	0	0	0	ไม่มี	
rlf	f,d	เลื่อนข้อมูลไปทางซ้าย 1 บิตผ่านบิต C	1	00	1101	d	f	f	f	f	f	f	f	f	f	f	C	1
rrf	f,d	เลื่อนข้อมูลไปทางขวา 1 บิตผ่านบิต C	1	00	1100	d	f	f	f	f	f	f	f	f	f	f	C	1
subwf	f,d	f-W	1	00	0010	d	f	f	f	f	f	f	f	f	f	f	C, DC, Z	1
swapf	f,d	สลับค่า 4 บิตบนกับ 4 บิตล่างของ f	1	00	1110	d	f	f	f	f	f	f	f	f	f	f	ไม่มี	1
xorwf	f,d	W XOR f	1	00	0110	d	f	f	f	f	f	f	f	f	f	f	Z	1
กลุ่มคำสั่งจัดการข้อมูลระดับบิต																		
นีโมนิก	โอเปอร์แรนด์	รายละเอียด	ไซเกิล	ออปโค้ด 14 บิต													บิตสถานะที่เกี่ยวข้อง	หมายเหตุ
bcf	f,b	เคลียร์บิตของ f	1	01	00	b	b	f	f	f	f	f	f	f	f	f	ไม่มี	1
bsf	f,b	เซตบิตของ f	1	01	01	b	b	f	f	f	f	f	f	f	f	f	ไม่มี	1
btfsc	f,b	ตรวจสอบบิต ข้ามหนึ่งแอดเดรสถ้าเป็น "0"	1 (2)	01	10	b	b	f	f	f	f	f	f	f	f	f	ไม่มี	2
btfss	f,b	ตรวจสอบบิต ข้ามหนึ่งแอดเดรสถ้าเป็น "1"	1 (2)	01	11	b	b	f	f	f	f	f	f	f	f	f	ไม่มี	2
กลุ่มคำสั่งจัดการกับข้อมูลค่าคงที่และควบคุมการทำงาน																		
นีโมนิก	โอเปอร์แรนด์	รายละเอียด	ไซเกิล	ออปโค้ด 14 บิต													บิตสถานะที่เกี่ยวข้อง	หมายเหตุ
addlw	k	ค่าคงที่ + W	1	11	111x	k	k	k	k	k	k	k	k				C, DC, Z	
andlw	k	ค่าคงที่ AND W	1	11	1001	k	k	k	k	k	k	k	k				Z	
call	k	เรียกโปรแกรมย่อย	2	10	0	k	k	k	k	k	k	k	k				ไม่มี	
clrwdt	-	เคลียร์ค่าของ WDT	1	00	0000	0	1	1	0	0	1	0	0				$\overline{TO}$, $\overline{PD}$	
goto	k	กระโดดไปยังแอดเดรสที่กำหนด	2	10	1	k	k	k	k	k	k	k	k				ไม่มี	
iorlw	k	ค่าคงที่ OR W	1	11	1000	k	k	k	k	k	k	k	k				Z	
movlw	k	นำค่าคงที่ไปเก็บไว้ใน W	1	11	00	x	x	k	k	k	k	k	k				ไม่มี	
retfie	-	ออกจากโปรแกรมย่อยบริการอินเตอร์รัปต์	2	00	0000	0	0	0	0	1	0	0	1				ไม่มี	
retlw	k	กลับสู่โปรแกรมด้วยค่าคงที่ใน W	2	11	01	x	x	k	k	k	k	k	k				ไม่มี	
return	-	ออกจากโปรแกรมย่อย	2	00	0000	0	0	0	0	1	0	0	0				ไม่มี	
sleep	-	เข้าสู่โหมดสลีป	1	00	0000	0	1	1	0	0	1	1					$\overline{TO}$, $\overline{PD}$	
sublw	k	ค่าคงที่ - W	1	11	110x	k	k	k	k	k	k	k	k				C, DC, Z	
xorlw	k	ค่าคงที่ XOR W	1	11	1010	k	k	k	k	k	k	k	k				Z	

หมายเหตุ :

1. ถ้ากระทำคำสั่งนี้กับรีจิสเตอร์ TMR0 ปรีสเกลเลอร์ภายในไทเมอร์เคาน์เตอร์จะถูกเคลียร์
2. ถ้ากระทำคำสั่งนี้แล้ว เงื่อนไขในการตรวจสอบถูกต้อง คำสั่งนี้จะใช้เวลา 2 ไซเกิล โดยในไซเกิลที่สอง จะเป็นกระทำคำสั่งในลักษณะ NOP ส่วนค่าของ PC จะเปลี่ยนแปลงเพิ่มขึ้น 2 ค่า

ตารางที่ 3-1 ตารางสรุปการแบ่งกลุ่มคำสั่งของ PIC16F84 ตามการกำหนดโดย MPASM



รูปที่ 3-1 แสดงการทำงานของโปรแกรมย่อยที่ใช้วิธีการเข้าถึงข้อมูลแบบสัมพัทธ์

การเขียนโปรแกรมภาษาแอสเซมบลีของ ไมโครคอนโทรลเลอร์ PIC16F84

เทคนิคการเขียนโปรแกรม

ความรู้เบื้องต้นก่อนการเขียนโปรแกรมคือไมโครคอนโทรลเลอร์ PIC16F84 และไมโครคอนโทรลเลอร์ส่วนใหญ่จะทำงานตามคำสั่งที่เก็บอยู่ในหน่วยความจำโปรแกรมเรียงลำดับกันไป และสำหรับผู้ออกแบบโปรแกรมและผู้เขียนโปรแกรมควรเริ่มต้นการเขียนโฟลว์ชาร์ตเพราะจะช่วยให้ไม่สับสน สามารถเขียนโปรแกรมได้อย่างเป็นระบบทำความเข้าใจและตรวจสอบง่าย

การเขียนโปรแกรมสำหรับข้อมูลที่เกิดขึ้นซ้ำๆกันนั้น สามรถเขียนโดยวิธีการต่อข้อมูลออกไปเรื่อยๆ แต่บางทีวิธีนี้ก็ไม่ถูกต้องนักเนื่องจากหน่วยความจำภายในที่ไมโครคอนโทรลเลอร์มีอยู่นั้นมีขนาดไม่มาก หากบรรจุข้อมูลของโปรแกรมที่ซ้ำๆ จะเป็นการสิ้นเปลือง

ดังนั้นการใช้วิธีวนลูปจึงเป็นเทคนิคที่มีความสำคัญและมักจะพบมากในระบบควบคุมที่ใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุมหลัก ทั้งนี้เนื่องจากในไมโครคอนโทรลเลอร์มีคำสั่งสำหรับการกระโดไปทำงานจุดต่างๆได้ การวนลูปให้ผลลัพธ์ดังนี้

-ถ้ามีการวนลูปโดยไม่มีการกำหนดเงื่อนไขให้หยุดโปรแกรมจะทำงานไปเรื่อยๆ จนกว่าจะหยุดจ่ายไฟให้กับไมโครคอนโทรลเลอร์

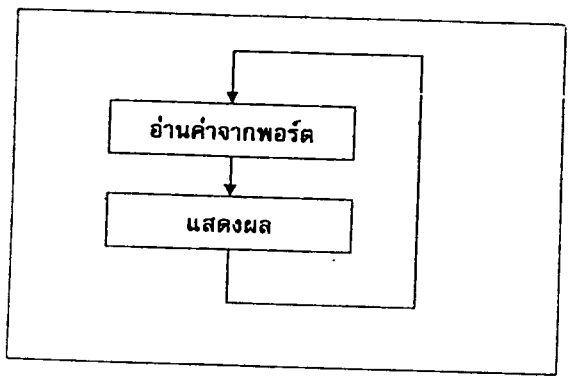
-ถ้ามีการวนลูปโดยใช้เคาน์เตอร์ เป็นตัวกำหนดรอบของการทำงานโปรแกรมจะทำงานเท่ากับจำนวนที่กำหนดไว้ในเคาน์เตอร์เท่านั้น เมื่อค่าในเคาน์เตอร์มีค่าเท่ากับค่าที่กำหนดโปรแกรมจะออกจากการวนลูป

-ถ้าการวนลูปมีการตรวจจับค่าอินพุต โปรแกรมจะออกจาการวนลูปเมื่อมีการป้อนค่าอินพุตเข้าสู่ระบบ

โฟลว์ชาร์ตง่ายๆของการวนลูปแสดงในรูป A โดยโปรแกรมจะเริ่มทำการอ่านค่า แล้วนำข้อมูลที่อ่านได้นี้ไปแสดงผลออกสูพอร์ตเอาต์พุต แล้วจึงวนกลับมาอ่านค่าอีกครั้ง

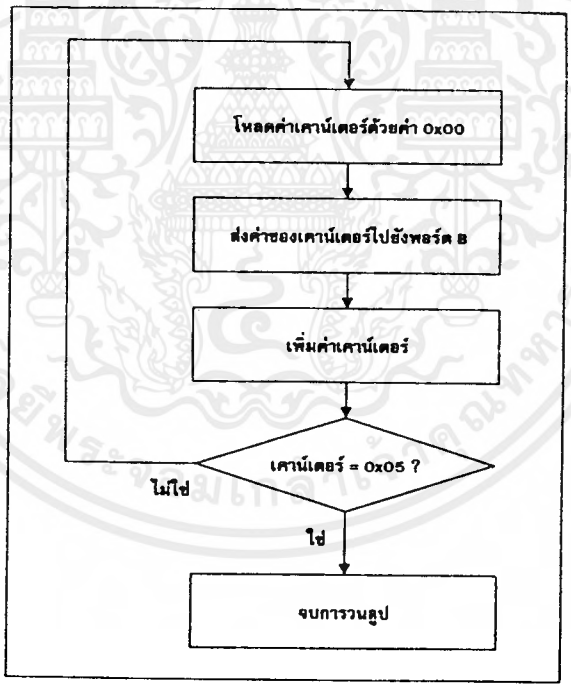
สำหรับโฟลว์ชาร์ตของโปรแกรมลูปที่ใช้เคาน์เตอร์ แสดงในรูป B ค่าของเวลาที่ใช้ในการวนลูปกำหนดได้ที่ตัวแปร N

เทคนิคอีอย่างหนึ่งที่ช่วยให้ขนาดของโปรแกรมเล็กลงก็คือ การใช้โปรแกรมย่อย หรือ Subroutine ในบางครั้งมีการกระทำกลุ่มคำสั่งใดๆ มากกว่าหนึ่งครั้ง ผู้เขียนโปรแกรมควรจะเขียนกลุ่มคำสั่งนั้นเพียงครั้งเดียว แล้วเก็บกลุ่มคำสั่งนั้นไว้ที่ใดที่หนึ่ง เมื่อต้องการใช้งานกลุ่มคำสั่งนี้ ก็ให้เรียกใช้ด้วยคำสั่ง CALL กลุ่มคำสั่งนี้ก็คือโปรแกรมย่อยนั่นเอง



รูป A แสดงโฟลว์ชาร์ตของโปรแกรมลูปอย่างง่าย

เมื่อซีพียูทำงานในโปรแกรมย่อยเรียบร้อยแล้วคำสั่ง RETURN หรือ RETLW จะถูกใช้ต่อท้ายโปรแกรมย่อยเพื่อกำหนดให้ซีพียูกระโดดกลับมาทำงานต่อยังโปรแกรมหลัก



รูป B แสด'โฟลว์ชาร์ตของโปรแกรมลูปที่ใช้เคาน์เตอร์

การแอสเซมเบลอร์

โปรแกรมภาษาแอสเซมบลีที่เขียนขึ้นมาจะขอเรียกว่า ไฟล์ซอร์สโค้ด เมื่อเขียนขึ้ด้วยเท็กซ์อิดิเตอร์ตัวใดตัวหนึ่งเรียบร้อยแล้วก็ต้องบันทึกลงในไฟล์ที่มีนามสกุลเป็น .ASM ทั้งนี้เพราะโปรแกรมแอสเซมเบลอร์จะแอสเซมเบลอร์ไฟล์ที่มีนามสกุล .ASM เท่านั้น

หลังจากที่ทำการแอสเซมเบลอร์โปรแกรมเรียบร้อยแล้วโปรแกรมแอสเซมเบลอร์จะทำการสร้างไฟล์ขึ้นมา 2 ไฟล์จากซอร์สโค้ดที่ผ่านการแอสเซมเบลอร์ โดยมีชื่อไฟล์เหมือนกับซอร์สโค้ดแต่จะแตกต่างกันที่นามสกุล ซึ่งจะเป็นไฟล์นามสกุล .LST และ .HEXสำหรับไฟล์นามสกุล . LST นั้นจะบรรจุรายละเอียดของโปรแกรมทั้งหมด ทั้งนีโมนิกและอ้อปโค้ดซึ่งเป็นเลขฐานสิบหกตลอดจนข้อความที่ใช้อธิบายการทำงานของโปรแกรม ดังแสดงในรูป C ส่วนไฟล์นามสกุล .HEX นั้นจะเป็นข้อมูลเลขฐานสิบหกที่ใช้สำหรับเขียนลงในหน่วยความจำโปรแกรมบนไมโครคอนโทรลเลอร์ PIC16F84

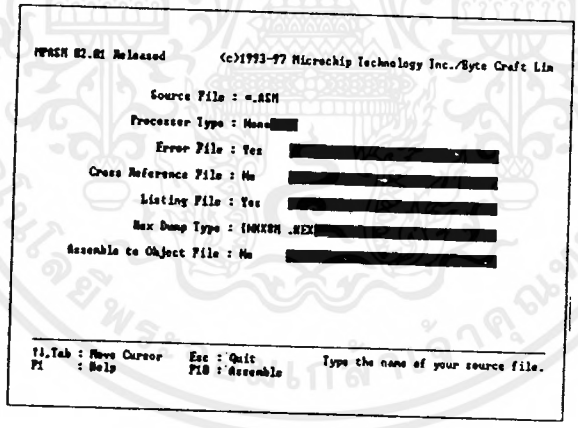
```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001     ;
                     00002     ; Sample MPASM Source Code (For Examples Only).
                     00003     ;
                     00004           list   p=16F84 ,r=HEX
0000                 00005           org    0x000    ; Program Start Here
0000     2801        00006           goto   Start    ; Goto Beginning
                     00007
0001     300A        00008     Start movlw  0x0a     ; Code For PIC16/17
0002     0086        00009           movwf  0x06     ; ...
0003     2801        00010           goto   Start    ; Do it Again...
                     00011           end

SYMBOL TABLE
  LABEL                             VALUE
Start                             00000001
__16F84                           00000001
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXX---------- ---------- ---------- ----------

All other memory blocks unused.

Program Memory Words Used:     4
Program Memory Words Free:  1020
Errors       :     0
Warnings :    0      reported,      0 suppressed
Messages :    0      reported,      0 suppressed
```

รูป C แสดงตัวอย่างไฟล์นามสกุล .LST

จากโปรแกรมตัวอย่าง A เมื่อทำการเขียนโปรแกรมเรียบร้อยแล้วให้ทำการบันทึกเป็นไฟล์ชื่อ EXAMPLE.ASM จากนั้นทำการแอสเซมเบลอร์เพื่อสร้างไฟล์นามสกุล .HEX

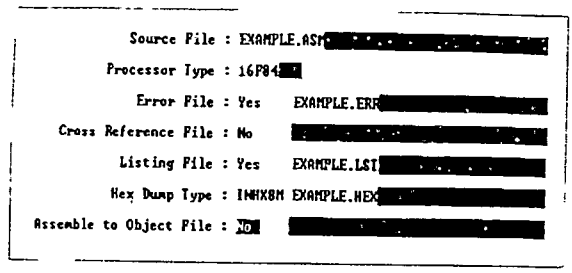
```
; Sample MPASM Source Code (For Examples Only).
        list    p=16F84,r=HEX
        org 0x000       ; Program Start Here
        goto    Start       ; Goto Beginning
Start   movlw   0x0a        ; Code For PIC16/17
        movwf   0x06        ; ...
        goto    Start       ; Do it Again...
        end
```

โปรแกรม A โปรแกรมภาษาแอสเซมบลีตัวอย่าง



รูป D หน้าตาของโปรแกรม MPASM

ขั้นตอนการทำแอสเซมเบลอร์เริ่มด้วยการเรียกโปรแกรมนี้อยู่ จากนั้นจะปรากฏ MPASM ในไดเรกตอรี่ย่อยที่เก็บโปรแกรมนี้ จากนั้นจะปรากฏภาพหน้าตามรูป D จากนั้นให้ป้อนชื่อซอร์สไฟล์ EXAMPLE1.ASM ลงไป กด ENTER ใช้คีย์ลูกศรเพื่อเลื่อนไปยังบรรทัดถัดไป ทำการเลือกโปรเซสเซอร์เป็นเบอร์ 16F84 แล้วเปลี่ยนค่าต่างๆตามรูป E จากนั้นกดคีย์ F10 เพื่อทำการแอสเซมเบลอร์



รูป E แสดงการกำหนดค่าต่างๆ ของไฟล์ Example.asm เพื่อทำการ Assembler File Example.asm โดยใช้ MPASM

หลังจากแอสเซมเบลอร์เรียบร้อยแล้ว โปรแกรมจะแสดงจำนวนคำสั่งที่ผิดพลาดที่หน้าจอ ถ้าเขียนซอร์สโค้ดถูกค่าที่ได้จะเป็น 0

หลังจากทำการแอสเซมเบลอร์เรียบร้อยแล้ว ให้ตรวจสอบดูว่า มีไฟล์เกิดขึ้นใหม่ 2 ไฟล์ที่มีชื่อเหมือนกับไฟล์ซอร์สโค้ด ซึ่งในที่นี้คือ EXAMPLE แต่มีนามสกุลต่างกัน นั่นคือไฟล์ EXAMPLE.LST และไฟล์ EXAMPLE.HEX หากโปรแกรมแอสเซมเบลอร์แจ้งว่ามีความผิดพลาดเกิดขึ้น ให้ใช้โปรแกรมเท็กซ์อิดิเตอร์เปิดไฟล์ EXAMPLE1.LST ขึ้นมาดูว่ามีจุดผิดพลาดที่ใด จากนั้นทำการแก้ไขจุดผิดพลาดในไฟล์นามสกุล .ASM เท่านั้น เมื่อแก้ไขแล้ว ทำการบันทึกซ้ำ แล้วเรียกโปรแกรม MPASM เพื่อแอสเซมเบลอร์ใหม่อีกครั้ง ทำเช่นนี้จนกว่าไฟล์ซอร์สโค้ดที่เขียนขึ้นไม่เกิดจุดผิดพลาดอีก

อย่างไรก็ตามการที่แอสเซมเบลอร์ไฟล์ซอร์สโค้ดถูกต้องก็ไม่ได้หมายความว่า โปรแกรมที่เขียนขึ้นนั้นจะทำงานได้อย่างถูกต้อง การแอสเซมเบลอร์ผ่านหมายถึงเพียงว่า โปรแกรมนั้นเขียนขึ้นถูกหลัก ใช้คำสั่งถูกต้อง ไม่ผิดรูปแบบเท่านั้น ขั้นตอนจากนี้ คือ การนำเอาข้อมูลของไฟล์นามสกุล .HEX ไปเขียนหรือโปรแกรมลงบนตัวไมโครคอนโทรลเลอร์ PIC16F84 เพื่อสังเกตการทำงานต่อไป

```
:0800000001280A3086000128E6
:00000001FF
```

รูป F แสดงรายละเอียดของไฟล์ Example.hex ที่ได้จากการ Assembler File

```
;-------------------------;
; MC3 Stepping Motor Control ;
;-------------------------;

        list    P=16F84
_config 0x3FF3 ; RC OSC : WDT OFF : CP OFF : PWT OFF

PCL             equ     0x02
STATUS          equ     0x03
PORTA                   equ     0x05
PORTB                   equ     0x06

TEMP            equ     0x0c
COUNT0          equ     0x0d
COUNT1          equ     0x0e
COUNT2          equ     0x0f
STEP            equ     0x11

F       equ     1
W       equ     0
C       equ     0
Z       equ     2
RP0     equ     5
        org     0x000
        goto    Start
;-------------;
; Intial Port ;
;-------------;
Start   bsf     STATUS,RP0;Set I/O
        movlw   b'11110000' ;RB4-RB7 as input
        movwf   PORTB  ;RB0-RB3 drive motor
```

```
        clrf    PORTA  ;PORTA as output
        bcf     STATUS,RP0
        movlw   0x01
        movwf   STEP
        movlw   0xf0  ;Clear output
        movwf   PORTB
        clrf    PORTA  ;Clear LED
;------------;
; SCAN SWITCH ;
;------------;
ch1     btfsc   PORTB,4       ;If RB4 press
        goto    ch2
        movlw   0x01    ;On RA0
        movwf   PORTA
        goto    testbit
ch2     btfsc   PORTB,5       ;If RB5 press
        goto    ch3
        movlw   0x02    ;On RA1
        movwf   PORTA
        goto    testbit

ch3     btfsc   PORTB,6       ;If RB6 press
        goto    ch4
        movlw   0x04    ;On RA2
        movwf   PORTA
        goto    testbit

ch4     btfsc   PORTB,7       ;If RA7 press
        goto    No_Key
        movlw   0x08    ;On RA3
        movwf   PORTA
```

```
        goto    testbit

No_Key      movlw  0x00    ;Test LED on
        xorwf   PORTA,w
        btfsc   STATUS,Z
        goto    ch1     ;All LED off goto loop

;-----------------------;
; CHECK BIT OF SWITCH ;
;-----------------------;
testbit movlw   0x10
        bcf     STATUS,C
        movwf   TEMP
        movlw   0x05
        movwf   COUNT0
loop1   decf    COUNT0,f  ;Chang PORTA data to function
        rrf     TEMP,f
        movf    TEMP,w
        xorwf   PORTA,w
        btfss   STATUS,Z
        goto    loop1
        movf    COUNT0,w
        addwf   PCL,f
        goto    ch1
        goto    Half_left
        goto    Half_right
        goto    Single_left
        goto    Single_right
;-----------------;
; FUNCTION A LEFT ;
```

```
;----------------;
 Half_left
        movf    STEP,w
        call    StepTable
        movwf   PORTB
        decfsz  STEP,f
        goto    H_ret
        movlw   0x08
        movwf   STEP
H_ret   call    Delay
        goto    ch1

;----------------;
; FUNCTION A RIGHT ;
;----------------;
 Half_right
        movf    STEP,w
        call    StepTable
        movwf   PORTB
        incf    STEP,f
        movf    STEP,w
        sublw   0x07
        btfsc   STATUS,C
        goto    H_ret
        movlw   0x01
        movwf   STEP
        goto    H_ret
;----------------;
; FUNCTION B LEFT ;
;----------------;
 Single_left
```

```
        movf    STEP,w
        call    StepTable
        movwf   PORTB
        decfsz  STEP,f
        goto    S_ret
        movlw   0x08
        movwf   STEP
S_ret   call    Delay
        clrf    PORTA
        goto    ch1


;------------------;
; FUNCTION B RIGHT ;
;------------------;
Single_right
        movf    STEP,w
        call    StepTable
        movwf   PORTB
        incf    STEP,f
        movf    STEP,w
        sublw   0x08
        btfsc   STATUS,C
        goto    S_ret
        movlw   0x01
        movwf   STEP
        goto    S_ret
```

```
;-------------------------;
; Table of stepping count ;
;-------------------------;
 StepTable
          addwf  PCL,F
          retlw   0x00
          retlw   0xfc
          retlw   0xf4
          retlw   0xf6
          retlw   0xf2
          retlw   0xf3
          retlw   0xf1
          retlw   0xf9
          retlw   0xf8
;-----------------;
; DELAY SUBRUTINE ;
;-----------------;
 Delay  movlw  0x40    ;Change value for setting stepper motor speed
        movwf  COUNT1
 Delay1         movlw  0x03
        movwf  COUNT2
        decfsz  COUNT2,F
        goto    $-1
        decfsz  COUNT1,F
        goto    Delay1
        return
        end
```

MICROCHIP

PIC16F8X

18-pin Flash/EEPROM 8-Bit Microcontrollers

Devices Included in this Data Sheet:

- PIC16F83
- PIC16F84
- PIC16CR83
- PIC16CR84
- Extended voltage range devices available (PIC16LF8X, PIC16LCR8X)

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single cycle except for program branches which are two-cycle
- Operating speed: DC - 10 MHz clock input
  DC - 400 ns instruction cycle

Device	Program Memory (words)	Data RAM (bytes)	Data EEPROM (bytes)	Max. Freq (MHz)
PIC16F83	512 Flash	36	64	10
PIC16F84	1 K Flash	68	64	10
PIC16CR83	512 ROM	36	64	10
PIC16CR84	1 K ROM	68	64	10

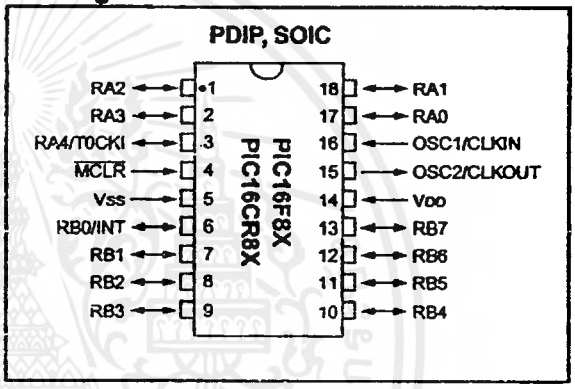
- 14-bit wide instructions
- 8-bit wide data path
- 15 special function hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt on change
  - Data EEPROM write complete
- 1000 erase/write cycles Flash program memory
- 10,000,000 erase/write cycles EEPROM data memory
- EEPROM Data Retention > 40 years

Peripheral Features:

- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 20 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Pin Diagrams

PDIP, SOIC



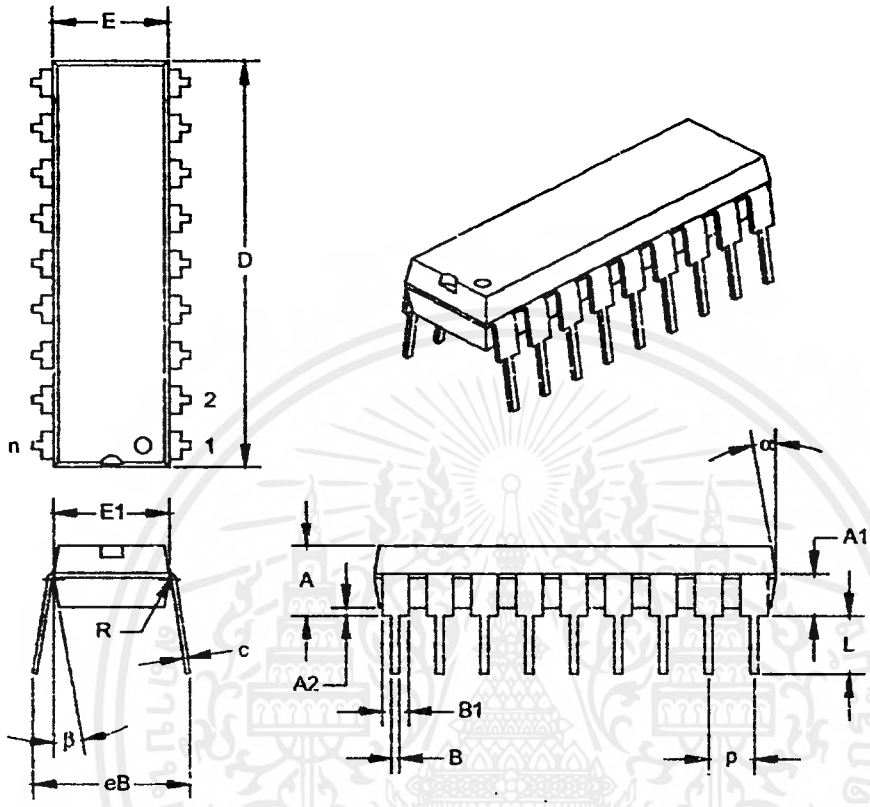
Special Microcontroller Features:

- In-Circuit Serial Programming (ICSP™) - via two pins (ROM devices support only Data EEPROM programming)
- Power-on Reset (POR)
- Power-up Timer (PWRT)
- Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Code-protection
- Power saving SLEEP mode
- Selectable oscillator options

CMOS Flash/EEPROM Technology:

- Low-power, high-speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 6.0V
  - Industrial: 2.0V to 6.0V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 1 µA typical standby current @ 2V

Package Type: K04-007 18-Lead Plastic Dual In-line (P) – 300 mil



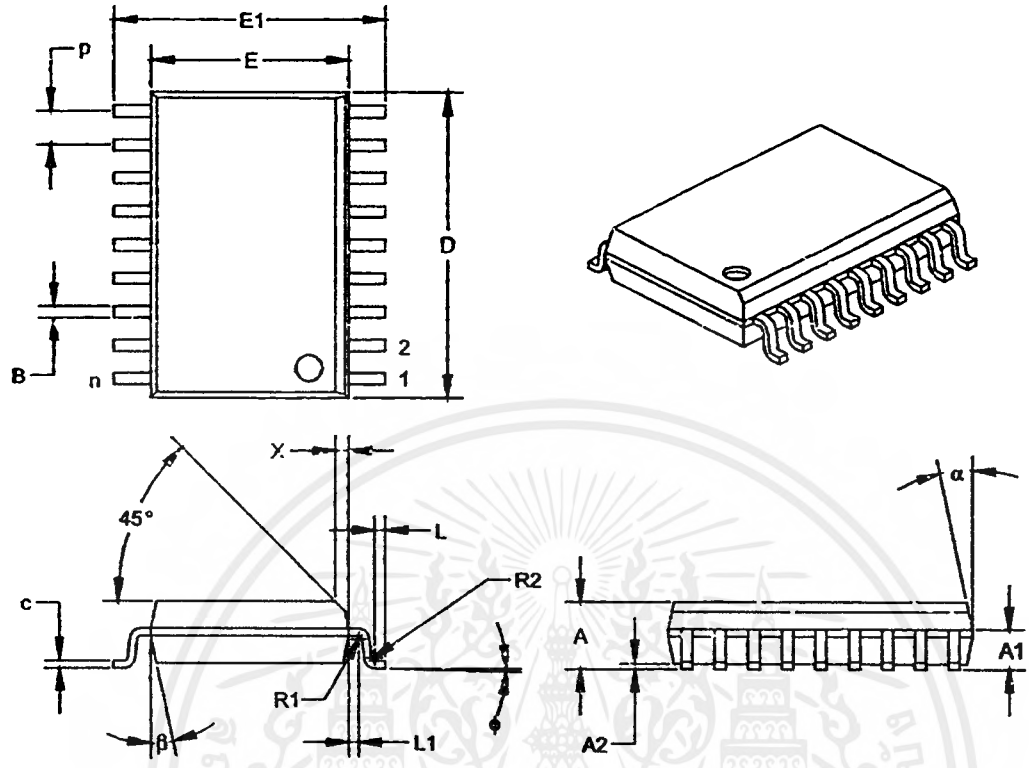
Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
PCB Row Spacing			0.300			7.62	
Number of Pins	n		18			18	
Pitch	p		0.100			2.54	
Lower Lead Width	B	0.013	0.018	0.023	0.33	0.46	0.58
Upper Lead Width	B1†	0.055	0.060	0.065	1.40	1.52	1.65
Shoulder Radius	R	0.000	0.005	0.010	0.00	0.13	0.25
Lead Thickness	c	0.005	0.010	0.015	0.13	0.25	0.38
Top to Seating Plane	A	0.110	0.155	0.155	2.79	3.94	3.94
Top of Lead to Seating Plane	A1	0.075	0.095	0.115	1.91	2.41	2.92
Base to Seating Plane	A2	0.000	0.020	0.020	0.00	0.51	0.51
Tip to Seating Plane	L	0.125	0.130	0.135	3.18	3.30	3.43
Package Length	D‡	0.890	0.895	0.900	22.61	22.73	22.86
Molded Package Width	E‡	0.245	0.255	0.265	6.22	6.48	6.73
Radius to Radius Width	E1	0.230	0.250	0.270	5.84	6.35	6.86
Overall Row Spacing	eB	0.310	0.349	0.387	7.87	8.85	9.83
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter.

† Dimension "B1" does not include dam-bar protrusions. Dam-bar protrusions shall not exceed 0.003" (0.076 mm) per side or 0.006" (0.152 mm) more than dimension "B1."

‡ Dimensions "D" and "E" do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010" (0.254 mm) per side or 0.020" (0.508 mm) more than dimensions "D" or "E."

Package Type: K04-051 18-Lead Plastic Small Outline (SO) – Wide, 300 mil



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Pitch	p		0.050			1.27	
Number of Pins	n		18			18	
Overall Pack. Height	A	0.093	0.099	0.104	2.36	2.50	2.64
Shoulder Height	A1	0.048	0.058	0.068	1.22	1.47	1.73
Standoff	A2	0.004	0.008	0.011	0.10	0.19	0.28
Molded Package Length	D‡	0.450	0.456	0.462	11.43	11.58	11.73
Molded Package Width	E‡	0.292	0.296	0.299	7.42	7.51	7.59
Outside Dimension	E1	0.394	0.407	0.419	10.01	10.33	10.64
Chamfer Distance	X	0.010	0.020	0.029	0.25	0.50	0.74
Shoulder Radius	R1	0.005	0.005	0.010	0.13	0.13	0.25
Gull Wing Radius	R2	0.005	0.005	0.010	0.13	0.13	0.25
Foot Length	L	0.011	0.016	0.021	0.28	0.41	0.53
Foot Angle	φ	0	4	8	0	4	8
Radius Centerline	L1	0.010	0.015	0.020	0.25	0.38	0.51
Lead Thickness	c	0.009	0.011	0.012	0.23	0.27	0.30
Lower Lead Width	B†	0.014	0.017	0.019	0.36	0.42	0.48
Mold Draft Angle Top	α	0	12	15	0	12	15
Mold Draft Angle Bottom	β	0	12	15	0	12	15

* Controlling Parameter.

† Dimension "B" does not include dam-bar protrusions. Dam-bar protrusions shall not exceed 0.003" (0.076 mm) per side or 0.006" (0.152 mm) more than dimension "B."

‡ Dimensions "D" and "E" do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010" (0.254 mm) per side or 0.020" (0.508 mm) more than dimensions "D" or "E."

PIC16F8X

TABLE 1-1 PIC16F8X FAMILY OF DEVICES

		PIC16F83	PIC16CR83	PIC16F84	PIC16CR84
Clock	Maximum Frequency of Operation (MHz)	10	10	10	10
Memory	Flash Program Memory	512	—	1K	—
	EEPROM Program Memory	—	—	—	—
	ROM Program Memory	—	512	—	1K
	Data Memory (bytes)	36	36	68	68
	Data EEPROM (bytes)	64	64	64	64
Peripherals	Timer Module(s)	TMR0	TMR0	TMR0	TMR0
	Interrupt Sources	4	4	4	4
	I/O Pins	13	13	13	13
Features	Voltage Range (Volts)	2.0-6.0	2.0-6.0	2.0-6.0	2.0-6.0
	Packages	18-pin DIP, SOIC	18-pin DIP, SOIC	18-pin DIP, SOIC	18-pin DIP, SOIC

All PICmicro™ Family devices have Power-on Reset, selectable Watchdog Timer, selectable code protect and high I/O current capability. All PIC16F8X Family devices use serial programming with clock pin RB6 and data pin RB7.

FIGURE 3-1: PIC16F8X BLOCK DIAGRAM