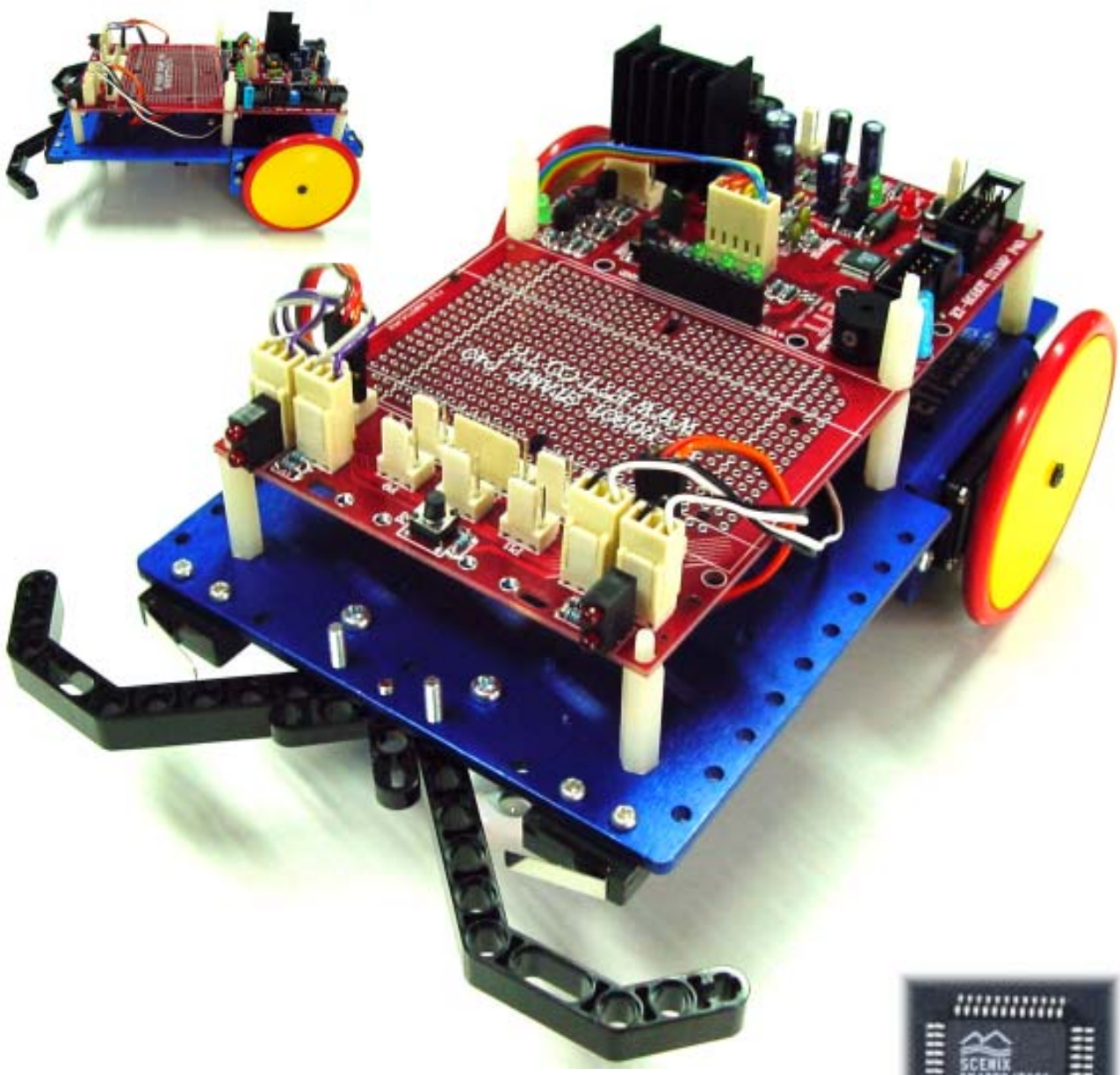
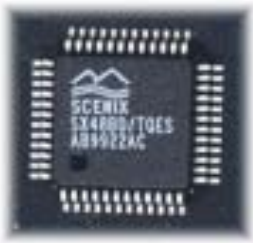
ET-ROBOTICS SYSTEM 2

คู่มือเรียนรู้และใช้งาน

ET-ROBOT STAMP P40

และชุดคำสั่ง BASIC STAMP





ETT
www.ett.co.th

บริษัท อีทีที จำกัด ETT CO., LTD.

1112/96-98 ถนนสุขุมวิท แขวงพระโขนง เขตคลองเตย กรุงเทพฯ 10110 http://www.etteam.com
1112/96-98 Sukhumvit Rd., Phrakanong Klongtoey Bangkok 10110 http://www.ett.co.th
Tel : 02-7121120 Fax : 02-3917216 email : sale@etteam.com

ชื่อหนังสือ “เรียนรู้และใช้งาน ET ROBOT STAMP P40 ”

ISBN 974-91440-9-0

ผู้เขียน นายวัชรินทร์ เคารพ , นายเรวัฒน์ ชินพัฒนวานิช

พิมพ์ครั้งที่ 1 จำนวน 1000 เล่ม

1 สิงหาคม 2546

จำนวน 112 หน้า

ราคา 120 บาท

จัดพิมพ์โดย
บริษัท อีทีที จำกัด
1112/96-98 ถนนสุขุมวิท แขวงพระโขนง
เขตคลองเตย กรุงเทพ 10110
โทร. 02-712-1120 - 1 FAX 02- 391-7216.
E-Mail : sale@etteam.com

ETT
www.ett.co.th

คำนำ

หลายคนคงเคยมีความคิดที่อยากจะสร้างหุ่นยนต์ของตัวเองขึ้นมาสักตัวหนึ่ง แต่ก็ไม่รู้ว่าจะเริ่มอย่างไร จะลงมือทำเองตั้งแต่ต้นก็เสียเวลา และ อาจมองเป็นเรื่องยาก ทำให้หมดกำลังใจที่จะทำ จากจุดนี้เองเราจึงได้สร้างหุ่นยนต์สำหรับเรียนรู้ทดลองชื่อ ET-ROBOT STAMP P40 เพื่อที่จะตอบสนองความต้องการของท่าน ซึ่งประกอบไปด้วยองค์ประกอบพื้นฐานของหุ่นยนต์ที่ครบถ้วน เช่น โครงหุ่นยนต์ ,มอเตอร์ ,ชุดเซนเซอร์ และ บอร์ดควบคุมที่สามารถโปรแกรมได้ ทำให้ท่านไม่ต้องเสียเวลาในการออกแบบสิ่งต่างๆ เหล่านี้ โดยท่านสามารถนำไปศึกษาเขียนโปรแกรมทดลองได้เลย อีกทั้งยังมีหนังสือคู่มือ พร้อมตัวอย่างต่างๆ ที่จะช่วยให้ท่านเกิดการเรียนรู้ที่เร็วขึ้น

สิ่งที่ท่านจะได้จากการศึกษาเรียนรู้ในการควบคุมหุ่นยนต์นี้ก็คือ ทักษะในการเขียนโปรแกรม เสริมสร้างความคิดจินตนาการ จนถึงขั้นที่ท่านสามารถนำเอาความรู้จากการควบคุมหุ่นยนต์ไปประยุกต์ใช้งานจริง เช่น สร้างเป็นระบบควบคุมต่างๆ เนื่องจากอุปกรณ์ที่ใช้ในหุ่นยนต์ เช่น Servo motor , เซนเซอร์อินฟราเรด และ ส่วนอื่นๆ เป็นอุปกรณ์ที่มีการใช้งานจริงๆ ในอุตสาหกรรมต่างๆ ดังนั้นจะเห็นได้ว่าหุ่นยนต์ที่ท่านจะได้ศึกษานี้ไม่ได้เป็นแค่ของเล่นเท่านั้น

ภายในหนังสือคู่มือเล่มนี้จะกล่าวถึงรายละเอียดต่างๆ ของหุ่นยนต์ ตั้งแต่ชิ้นส่วน ,การประกอบ, วิธีการควบคุมในรูปแบบต่างๆ พร้อมตัวอย่างโปรแกรมที่ใช้ในการทดลอง และ ยังมีรายละเอียดของชุดคำสั่งของ CPU Basic Stamp ให้อีกด้วยเพื่อความสะดวกในการศึกษาทดลอง เหตุที่เลือก CPU เป็นตระกูล Basic Stamp เนื่องจากเป็น CPU ที่ใช้ภาษาเบสิกในการพัฒนา ซึ่งมีรูปแบบภาษาที่ไม่ยากนัก เหมาะสำหรับผู้เริ่มต้นที่ต้องการเรียนรู้เกี่ยวกับไมโครคอนโทรลเลอร์ ทั้งนี้หวังว่าหนังสือเล่มนี้จะช่วยให้ท่านเกิดการเรียนรู้ ตลอดจนสามารถนำความรู้ที่ได้นี้ไปประยุกต์ใช้ตามความต้องการของท่านได้เป็นอย่างดี

ทีมงานอีทีที

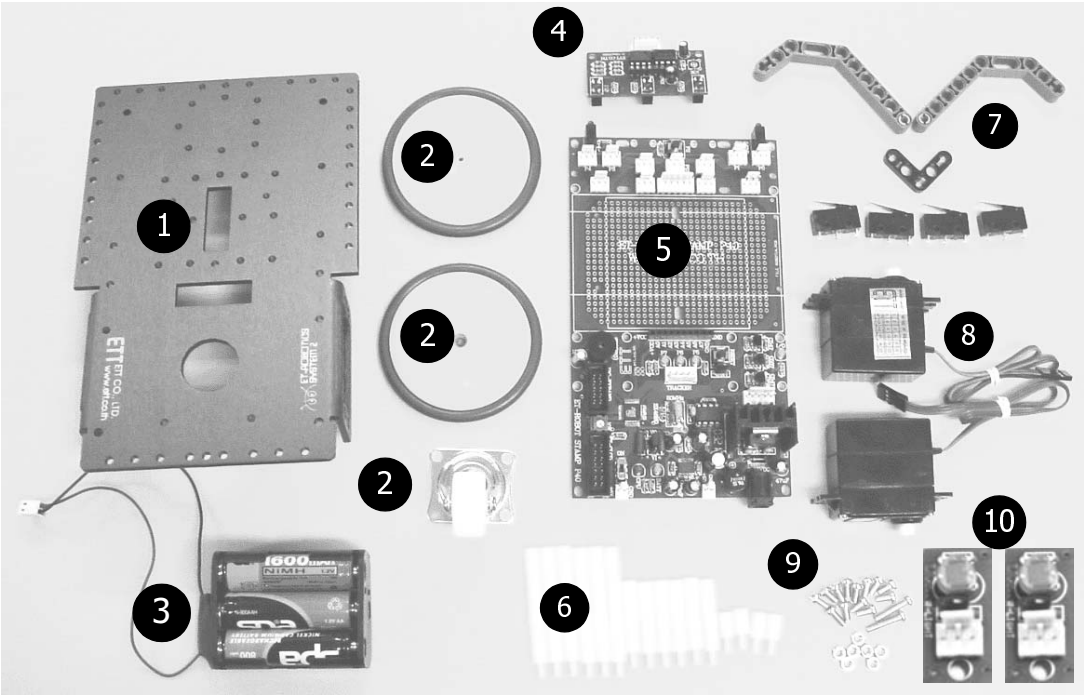
กรกฎาคม 2546

สารบัญ

บทที่ 1

ส่วนประกอบของหุ่นยนต์ ET-ROBOT STAMP

หุ่นยนต์ ET-ROBOT นี้ถูกออกแบบสำหรับการเรียนรู้ทดลองในเรื่องของการควบคุมหุ่นยนต์ด้วยการเขียนโปรแกรมภาษาเบสิก ซึ่งในส่วนของฮาร์ดแวร์นั้นได้พยายามออกแบบให้มีความอ่อนตัวมากที่สุด โดยจะแบ่งเป็นส่วนประกอบต่างๆ ออกเป็นส่วนๆ ดังรูปที่ 1.1



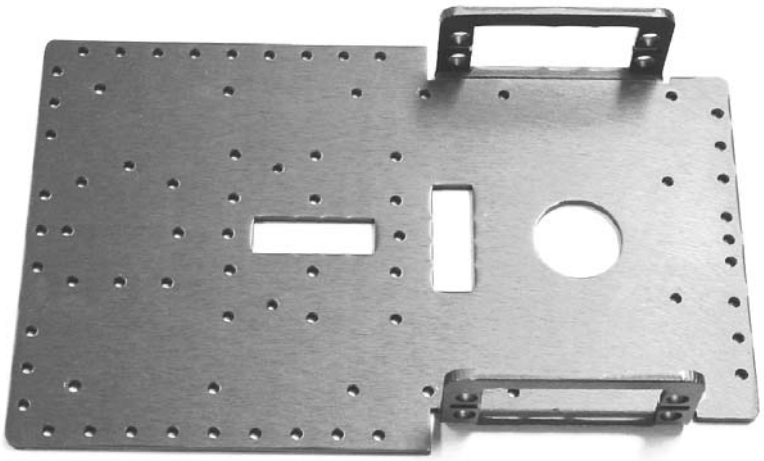
รูปที่ 1.1 ส่วนประกอบต่างๆ ของ ET-ROBOT

ประกอบไปด้วย

1. โครงหุ่นยนต์ (Body)
2. ล้อหุ่นยนต์
3. แบตเตอรี่
4. ชุดเซ็นเซอร์ตรวจจับเส้น
5. บอร์ดควบคุม (Basic Stamp)
6. ฐานรอง
7. สวิตช์กันชนหน้าหลัง (Micro Switchs)
8. เซอร์โวมอเตอร์ (Servo Motor)
9. น็อต – สกรู
10. เซนเซอร์แสง (Light sensor)

1.1 โครงหุ่นยนต์ (Body)

เป็นโครงสร้างหลักของหุ่นยนต์ ทำจาก สแตนเลส ทำให้มีความแข็งแรง และ น้ำหนักไม่มากนัก โดยจะมีการออกแบบพื้นที่สำหรับทำการติดตั้งอุปกรณ์ต่างๆ ซึ่งการยึดอุปกรณ์ต่างๆ เข้ากับโครงหุ่นยนต์นี้จะใช้วิธีการขันน็อต หรือ สกรู ทำให้ชิ้นส่วนต่างๆ ที่ประกอบนั้นมีความมั่นคงแน่นหนาเป็นอย่างดีจะมีลักษณะดังรูปที่ 1.2



รูปที่ 1.2 โครงหุ่นยนต์ (Body)

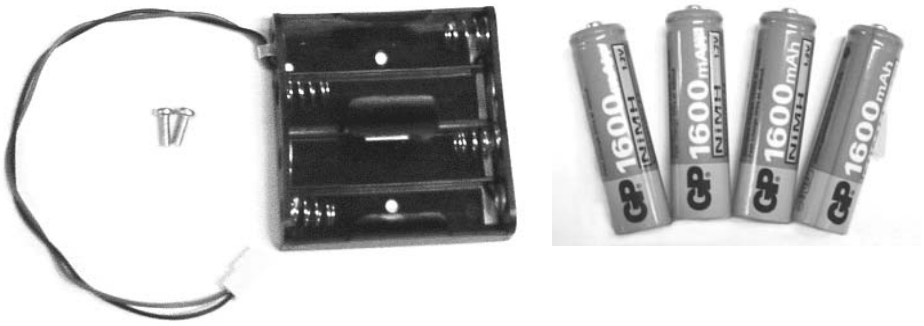
1.2 ล้อหุ่นยนต์

ล้อหุ่นยนต์จะมีทั้งหมด 3 ล้อ คือ ล้อหลัง 2 ล้อ และ ล้อหน้าอีกหนึ่งล้อ ในส่วนของล้อหน้านั้นจะมีการหมุนที่อิสระตามการเคลื่อนที่และทิศทางของหุ่นยนต์ โดยจะทำการยึดติดกับโครงของหุ่นยนต์ ส่วนล้อหลังทั้งสองล้อจะต้องประกอบเข้ากับตัว Servo Motor ทำให้การหมุนถูกควบคุมด้วยมอเตอร์ทั้ง 2 ตัว โดยจะมีลักษณะดังรูปที่ 1.3

รูปที่ 1.3 ล้อหุ่นยนต์

1.3 แบตเตอรี่ (Battery)

เป็นส่วนที่ทำหน้าที่จ่ายพลังงานไปเลี้ยงส่วนต่างๆ ของหุ่นยนต์ ไม่ว่าจะเป็นชุดควบคุม , มอเตอร์ หรือวงจรเซนเซอร์ต่างๆ โดยจะต้องใช้ถ่านขนาด 1.5 V จำนวน 4 ก้อนต่ออนุกรมกันเพื่อให้ได้แรงดันรวมกันไม่ต่ำกว่า 5 โวลท์ โดยจะต่ออนุกรมกันอยู่ในรังถ่าน ไม่ควรใช้ถ่านขนาด 1.2 V เพราะจะทำให้แรงดันไฟที่ได้ไม่เพียงพอกับการทำงานของหุ่นยนต์ ส่วนการติดตั้งกับโครงของหุ่นยนต์นั้นจะใช้วิธีการขันยึดด้วยน๊อต หรือ สกรู



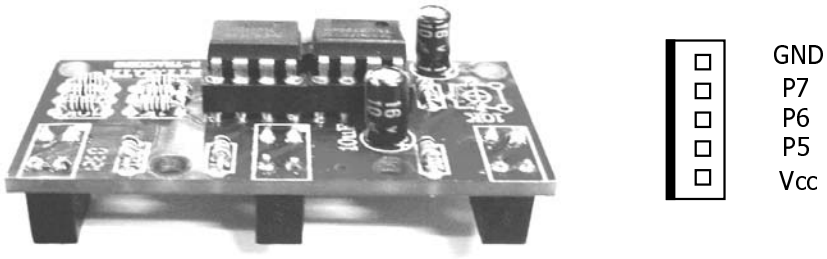
รูปที่ 1.4 รังถ่านแบตเตอรี่ขนาด 1.5 V Size AA (R-AA-TYPE4)

รูปที่ 1.5 รังถ่านขนาด 1.5 V Size D (R-D-TYPE4)

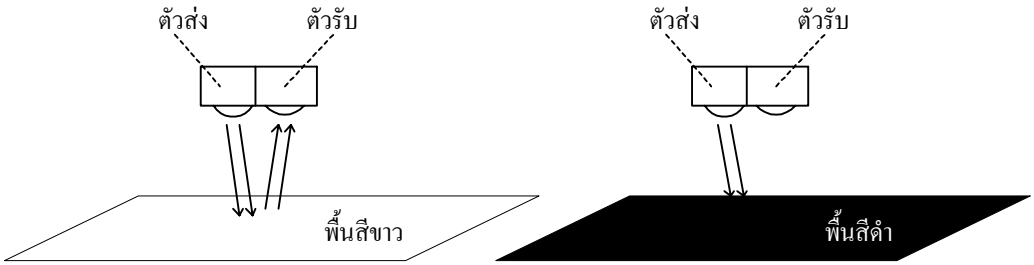
โดยรังถ่านที่เราจัดไว้ให้จะมี 2 ขนาด คือ รุ่น R-AA-TYPE4 และ รุ่น R-D-TYPE4 ทั้งสองรุ่นนี้สามารถบรรจุถ่านได้ 4 ก่อน เหมือนกันตางกันตรงขนาดของถ่านที่จะนำมาบรรจุ โดยรังถ่านรุ่น R-AA-TYPE4 จะใช้บรรจุถ่านขนาด Size AA ส่วนรังถ่านรุ่น R-D-TYPE4 จะใช้บรรจุถ่านขนาด Size D ซึ่งจะมีขนาดใหญ่กว่า Size AA อยู่พอสมควร หรือ หากไม่ต้องการใช้ถ่านเหล่านี้ก็สามารถเปลี่ยนไปใช้แบตเตอรี่ประเภทอื่นๆ ก็ได้ที่สามารถจ่ายไฟได้ไม่ต่ำกว่า 5 โวลท์

1.4 ชุดเซ็นเซอร์ตรวจจับเส้น (TRACKER)

เป็นวงจรเซนเซอร์ที่ใช้ในการตรวจจับเส้น โดยจะใช้หลักการรับและส่งของวงจรอินฟราเรด ซึ่งในโมดูลนี้จะมีการออกแบบไว้ 3 จุด เพื่อเพิ่มความสามารถในการตรวจจับให้มากขึ้น โดยจะอาศัยหลักการในการสะท้อนกลับของสัญญาณอินฟราเรด ซึ่งจะไม่มีการสะท้อนกลับในวัตถุ หรือ พื้นผิวที่เป็นสีดำ ดังนั้นเราจึงสามารถทำการตรวจสอบหรือแยกความแตกต่างระหว่างวัตถุหรือพื้นผิวที่เป็นสีขาว และ สีดำได้ แต่เนื่องจากในสถานที่ต่างๆ นั้นจะมีความเข้มข้น หรือ ปริมาณของแสงที่ไม่เท่ากัน ซึ่งจะมีผลต่อการทำงานของวงจรอินฟราเรด ดังนั้นเราจึงออกแบบวงจรให้สามารถทำการปรับระดับความไวในการตรวจจับได้ โดยการปรับที่ตัวต้านทานปรับค่าได้ 10 Kohm เพื่อให้มีความอ่อนตัวเมื่อนำไปใช้ในสถานที่ต่างๆ มากขึ้น



รูปที่ 1.6 แผงวงจรตรวจจับเส้น



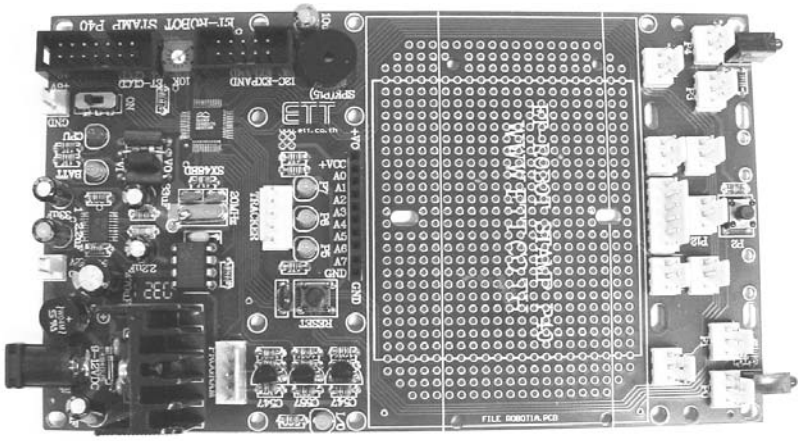
รูปที่ 1.7 ลักษณะการสะท้อนของคลื่นอินฟราเรด

จากรูปที่ 1.7 ในตัวตรวจจับเส้น 1 ชุดจะมีอินฟราเรดจำนวน 2 ตัว คือ ตัวรับและตัวส่ง ซึ่งในการทำงานตัวส่งจะทำการส่งคลื่นอินฟราเรดออกมาตลอดเวลา และ เมื่อกระทบกับวัตถุก็จะมีการสะท้อนกลับของสัญญาณไปยังตัวรับแต่ขนาดความแรงของสัญญาณที่สะท้อนกลับนั้นจะขึ้นอยู่กับสีพื้นผิวของวัสดุ โดยในพื้นผิวที่มีสีทึบ เช่น สีดำ จะมีการสะท้อนกลับของสัญญาณน้อยมาก ส่วนสีที่มีความสว่าง เช่น สีขาว จะมีการสะท้อนสัญญาณได้ดี จากคุณสมบัติดังกล่าวเราจึงสามารถแยก ความแตกต่างของเส้นได้ โดยในโมดูลนี้ถูกออกแบบให้สถานะเอาต์พุตของวงจร (PIN 7,6 และ 5) มีการเปลี่ยนแปลงตามการสะท้อนของสัญญาณอินฟราเรดดังนี้

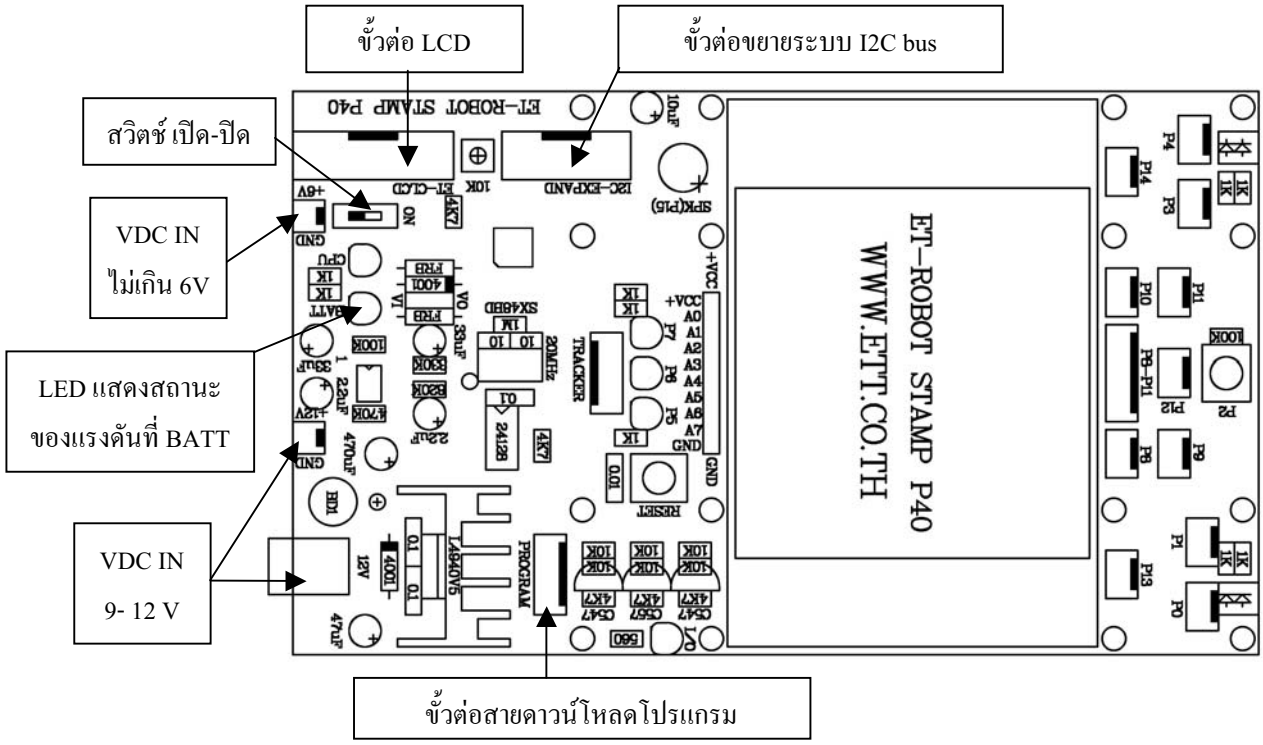
- เมื่อมีการสะท้อน (พื้นผิวสีขาว) ที่ขาสัญญาณเอาต์พุตจะมี Logic 0
- เมื่อไม่มีการสะท้อน (พื้นผิวสีดำ) ที่ขาสัญญาณเอาต์พุตจะมี Logic 1

1.5 บอร์ดควบคุม (Basic Stamp)

เป็นบอร์ดไมโครคอนโทรลเลอร์ในตระกูล Basic Stamp ภาษาของโปรแกรมที่ใช้ในการพัฒนาเป็นภาษาเบสิก ซึ่งไมโครคอนโทรลเลอร์ประเภทนี้จะมีตัวแปลภาษาอยู่ภายใน โดยในบอร์ดรุ่นนี้จะใช้ CPU เบอร์ SX48BD/TQ (BS2P40) ทำงานที่ความถี่ 20 MHz หรือที่ 12,000 คำสั่ง/วินาที เขียนโปรแกรมได้ 2KB X 8 โปรแกรม (16 KB) และ เมื่อเขียนโปรแกรมเสร็จสามารถดาวน์โหลดโปรแกรมลงบนตัว CPU ได้จาก PC ผ่านทาง Serial Port COM1 หรือ COM2 ก็ได้

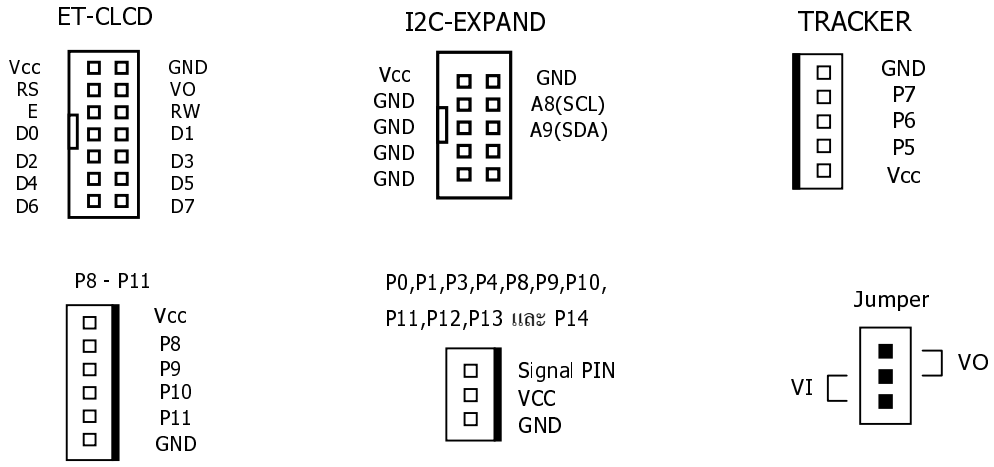


รูปที่ 1.8 ลักษณะบอร์ด ET-ROBOT STAMP P40



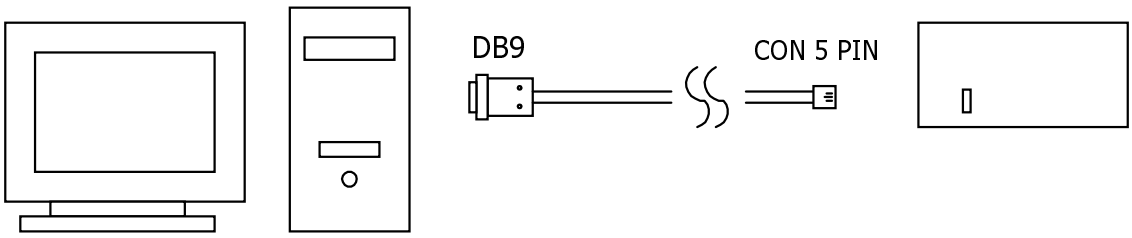
รูปที่ 1.9 ลักษณะการจัดวางอุปกรณ์ต่างๆ บนบอร์ด ET-ROBOT STAMP P40

จากรูปที่ 1.9 ในส่วนของการจ่ายไฟ (VDC IN) สามารถต่อได้หลายทางโดยขั้วต่อ VDC IN 9-12 นั้น จะไปผ่านวงจรปรับระดับแรงดันเป็น 5 VDC เพื่อเลี้ยงวงจร แต่ในส่วนของขั้วต่อ VDC IN +6V,GND นั้นจะเชื่อมต่อกับวงจรภายในโดยตรงดั้งนั้นควรจะจ่ายแรงดันเข้าที่จุดนี้ไม่เกิน 6 โวลท์เพื่อป้องกันความเสียหายที่จะเกิดขึ้นกับวงจร



รูปที่ 1.10 แสดงตำแหน่งขาสัญญาณของ Connector ต่างๆ

จากรูปที่ 1.10 ในส่วนของจัมเปอร์ VI/VO มีไว้สำหรับเลือกใช้งานวงจรยกระดับของแรงดันซึ่งจะใช้ไอซี TSP 60130-TSSOP20 เมื่อเลือกไปทาง VO หากแรงดันอินพุต (VDC IN) ต่ำกว่า 5 โวลท์ วงจรนี้จะทำการยกระดับแรงดันที่ต่ำนั้นขึ้นเป็น 5 V แต่แรงดันอินพุตจะต้องไม่ต่ำกว่า 4.2 V จึงจะสามารถยกระดับเป็น 5 โวลท์ได้ จะมีประโยชน์ในกรณีที่ใช้แบตเตอร์รี่ ซึ่งจะช่วยให้สามารถใช้งานได้นานขึ้น ส่วนกรณีเลือกเป็น VI แรงดัน VDC IN จะถูกต่อตรงเข้าวงจรภายใน ไม่ผ่านวงจรยกระดับในส่วนนี้ ควรใช้ในกรณีที่แรงดัน VDC IN มีมากพออยู่แล้ว



รูปที่ 1.11 แสดงการเชื่อมต่อสายดาวน์โหลดโปรแกรม

จากรูปที่ 1.11 เป็นการเชื่อมต่อสายดาวน์โหลดโปรแกรมโดยด้านที่เป็น DB9 นั้นจะต้องนำไปต่อที่พอร์ตอนุกรมของคอมพิวเตอร์ เช่น COM1 หรือ COM2 ส่วนปลายสายอีกด้านหนึ่งเป็นคอนเนคเตอร์ขนาด 5 PIN ให้นำไปต่อกับบอร์ด ET-ROBOT STAMP ที่ขั้วต่อ PROGRAM

1.6 ฐานรอง

ทำจากพลาสติก ทั้งนี้ก็เพื่อให้เกิดความปลอดภัยในการประกอบติดตั้งบอร์ดวงจรต่างๆ ซึ่งหากเป็นโลหะอาจจะทำให้เกิดการลัดวงจรของแผงวงจรควบคุม หรือ วงจรอื่นๆ ได้ ซึ่งจะมีอยู่หลายขนาดด้วยกัน และ ยังสามารถนำฐานรองต่างๆ มาต่อเข้าด้วยกันได้อีกด้วย

1.7 สวิตช์กันชนหน้าหลัง (Micro Switchs)

จะแบ่งออกเป็น 2 ส่วนคือ สวิตช์ที่ติดตั้งด้านหน้า และ สวิตช์ที่ติดตั้งด้านหลัง ซึ่งทั้งหมดจะเป็นไมโครสวิตช์ แต่สำหรับด้านหน้านั้นจะมีพลาสติกที่ช่วยให้สามารถตรวจสอบการชนได้ดีขึ้นดังรูปที่ 1.12

รูปที่ 1.12 แสดงลักษณะสวิตช์กันชนต่างๆ ของหุ่นยนต์

1.8 เซอร์โวมอเตอร์ (Servo Motor)

Servo motor คือ มอเตอร์ไฟฟ้ากระแสตรง (DC motor) ที่ถูกประกอบรวมกับ ชุดเกียร์ และ ส่วนควบคุมต่างๆ ไว้ในโมดูลเดียวกัน หรือ ภายในกล่องพาสติกเดียวกัน โดยมอเตอร์ชนิดนี้จะมีสายต่อใช้งานเพียง 3 เส้นเท่านั้น คือ VCC,GND และ สายสัญญาณควบคุม(Control Line) ซึ่งสามารถควบคุมให้มอเตอร์หมุนซ้าย หรือ ขวาได้จากสายสัญญาณเพียงเส้นเดียว โดยสัญญาณที่ใช้ควบคุมนี้จะเป็นสัญญาณ พัลส์วิดมอด (PWM) แบบ TTL Level ระดับแรงดันที่จ่ายให้มอเตอร์นี้จะอยู่ในช่วงประมาณ 4 ถึง 6 โวลท์ ขึ้นอยู่กับคุณสมบัติของมอเตอร์แต่ละตัว ข้อดีของมอเตอร์ชนิดนี้ก็คือ จะมีขนาดเล็กน้ำหนักเบา, ให้แรงบิตสูง ,กินพลังงานน้อย และ สามารถควบคุม ด้วยแรงดันลอจิกที่เป็น TTL ได้โดยตรงไม่จำเป็นต้องต่อวงจรขับ (Driver) อื่นๆ เพราะ มอเตอร์ชนิดนี้จะมีวงจรควบคุมบรรจุไว้ภายในอยู่แล้ว ซึ่งมอเตอร์ชนิดนี้สามารถควบคุมให้หมุนไปในตำแหน่ง หรือ ทิศทางองศาที่ต้องการได้ โดยอาศัยสัญญาณความกว้างพัลส์ ที่ป้อนให้มอเตอร์ โดยทั่วไปเซอร์โวมอเตอร์นี้จะหมุนได้แค่เพียงในช่วงประมาณ 180º หรือ ครึ่งรอบเท่านั้น หรือ บางรุ่นอาจหมุนได้ถึง 210˚ แต่จะไม่สามารถหมุนเป็นวงรอบได้ แต่เซอร์โวมอเตอร์ในชุดที่จัดให้กับ ET-ROBOT STAMP นี้เราได้ทำการปรับแต่ง (Modify) ให้สามารถหมุนเป็น วงรอบ 360องศาเรียบร้อยแล้ว ดังนั้นจึงสามารถนำไปใช้งานได้ทันที โดยเซอร์โวมอเตอร์ทั้งสองตัวนี้จะถูกยึดติดกับโครง และ ล้อของหุ่นยนต์ เพื่อใช้ในการขับเคลื่อนให้หุ่นยนต์สามารถเคลื่อนที่ได้

รูปที่ 1.13 ลักษณะของภายนอก Servo Motor

1.9 น็อต – สกรู (SCREWS)

ใช้ในการยึดชิ้นส่วนต่างๆ ของหุ่นยนต์ ซึ่งจะมีอยู่ด้วยกันหลายขนาด ตามหน้าที่และการใช้งาน ซึ่งการยึดชิ้นส่วนต่างๆ ด้วยน๊อต หรือ สกรูนี้จะช่วยทำให้ชิ้นส่วนต่างๆ ของหุ่นยนต์มีความมั่นคงแข็งแรง โดยจะมีลักษณะดังรูปที่ 1.14

รูปที่ 1.14 แสดงลักษะของน็อต-สกรูขนาดต่างๆ

1.10 เซนเซอร์แสง (Light sensor)

เซนเซอร์แสง คือ วงจรเซนเซอร์ที่ใช้ในการตรวจจับระดับความเข้มของแสงซึ่งตัวรับแสงที่เราใช้นี้จะเรียกว่าตัว LDR หรือ Photoresistor โดยจะมีคุณสมบัติในการเปลี่ยนแปลงค่าความต้านทานภายในตัวเองตามขนาดความเข้มของแสงที่ตกกระทบ ซึ่งการนำเอา LDR มาต่อเป็นวงจรเซนเซอร์แสงนั้นจะมีหลายวิธี แต่ในที่นี้เราจะนำมาต่อเป็นวงจร RC โดยในการตรวจสอบสภาวะของแสงจะใช้วิธีการอ่านค่าเวลาของวงจร RC (RC time) ซึ่งความสัมพันธ์ ของค่าเวลา , ค่า R และ C จะเป็นดังสมการต่อไปนี้

$$\frac{t}{RxC} = \ln\left(\frac{V_{initial}}{V_{final}}\right)$$

$$\frac{t}{R \times 0.01 \times 10^{-6}} = \ln\left(\frac{5V}{1.4V}\right) \quad \ldots \text{Sec}$$

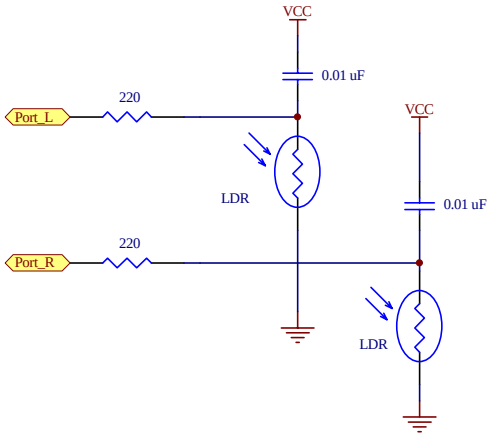
$$t = \ln(3.57) \times R \times 0.01 \times 10^{-6} \quad \ldots \text{Sec}$$

$$t = 1.27 \times 10^{-8} \times R \quad \ldots \text{Sec}$$

Vinitial = แรงดันค่าเริ่มต้น (เท่ากับแหล่งจ่าย Vcc)

Vfinal = แรงดันจุด threshold voltage ที่ขาสัญญาณ I/O ของ BASIC Stamp หากต่ำกว่า 1.4 V จะอ่านได้เป็น Logic 0

จากสมการจะเห็นว่าเมื่อเรากำหนดค่า C เป็นแบบคงที่ (0.01uF) จะทำให้ค่าเวลา RC time (t) มีการเปลี่ยนแปลงขึ้นอยู่กับค่าความต้านทาน (R) เพียงตัวเดียว ดังนั้นเมื่อความเข้มของแสงเปลี่ยนแปลงค่าความต้านทานของ LDR ก็จะเปลี่ยนไปด้วย และ มีผลทำให้ค่าเวลาของวงจร RC เปลี่ยนแปลง จากจุดนี้เราจึงสามารถนำเอา LDR มาต่อเป็นวงจรตรวจจับแสงในรูปแบบของวงจร RC time ได้ดังรูปที่ 1.15



รูปที่ 1.15 แสดงวงจรของเซนเซอร์แสง (Light Sensor)

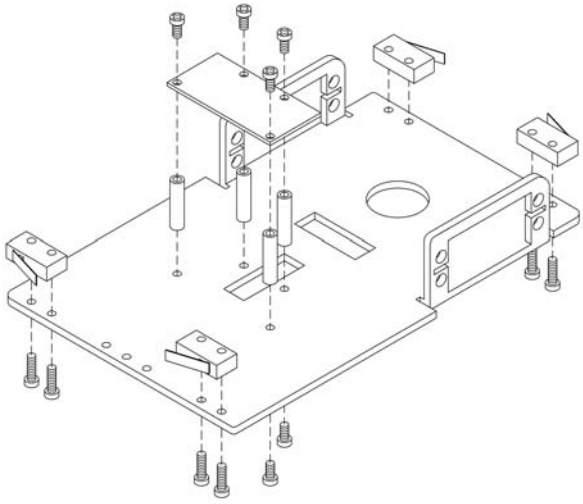
รูปที่ 1.16 แสดงลักษณะโมดูลเซนเซอร์แสง (Light Sensor)

บทที่ 2
การประกอบหุ่นยนต์

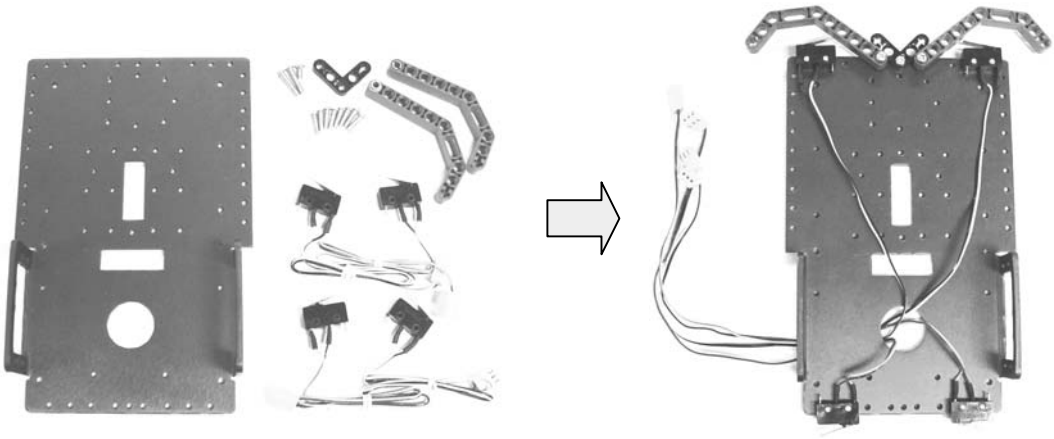
การติดตั้ง หรือ การประกอบอุปกรณ์ต่างๆ เข้ากับโครงของหุ่นยนต์นั้น จะใช้วิธีการยึดติดด้วยการขันสกรู ซึ่งชิ้นส่วนต่างๆ ของหุ่นยนต์นั้นจะมีด้วยกันหลายส่วนดังนั้นเพื่อให้เกิดความเข้าใจที่ดีขึ้นเราจะขออธิบายทีละส่วนเป็นลำดับดังนี้

2.1 การประกอบชุดเซนเซอร์สวิตช์กันชนหน้าหลัง

จะต้องทำการติดตั้งบนตัว Robot ในพื้นที่ด้านล่าง ซึ่งสวิตช์ทั้งหมดจะมี 4 ตัว แบ่งเป็นด้านหน้า 2 ตัว และด้านหลัง 2 ตัว เป็นด้านซ้ายและขวา ส่วนหนวด หรือ พลาสติกกันชน ให้ทำการติดตั้งที่ด้านหน้าของหุ่นยนต์เพื่อให้มีการทำงานร่วมกันกับสวิตช์กันชนด้านหน้า โดยส่วนต่างๆ เหล่านี้จะต้องทำการยึดด้วยสกรูให้แน่น ส่วนสายไฟให้สอดเข้าไปในช่องวงกลมให้ทะลุไปยังอีกด้านหนึ่งของตัว Robot ดังรูปที่ 2.2



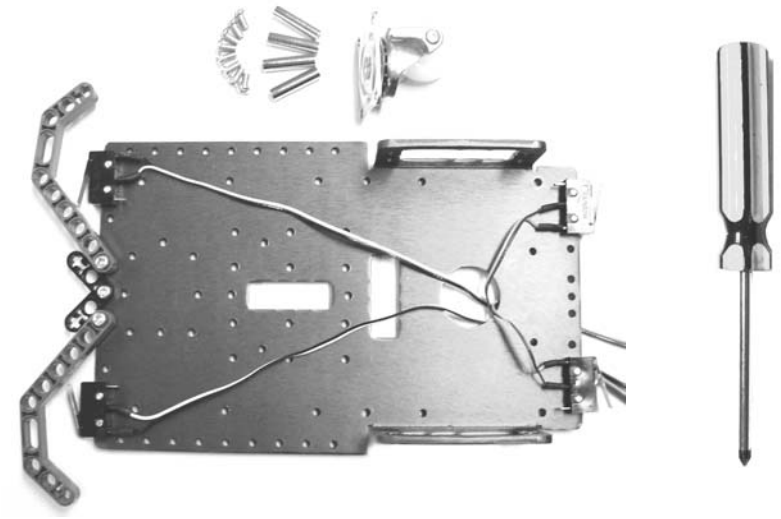
รูปที่ 2.1 ส่วนประกอบต่างๆ ของการติดตั้งสวิตช์กันชนหน้าหลัง



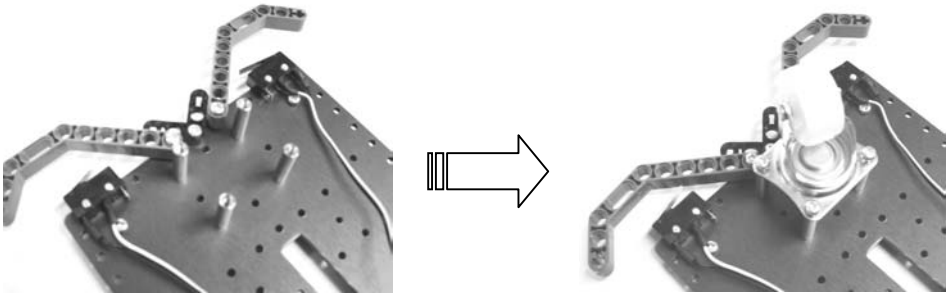
รูปที่ 2.2 แสดงลักษณะการติดตังสวิตช์กันชนให้กับ Robot

2.2 การติดตั้งล้อหน้า

ในส่วนของการติดตั้งล้อหน้านี้จะมีส่วนประกอบต่างๆที่ต้องทำการติดตั้งดังรูปที่ 2.3 โดยจะต้องทำการใส่ฐานรองเข้าไปก่อนเพื่อยกระดับของล้อให้สูงขึ้นจากพื้นโดยตำแหน่งในการยึดจะมีทั้งหมด 4 จุดดังในรูปที่ 2.4(ก) เมื่อครบแล้วให้นำล้อหน้าที่เตรียมไว้มาประกอบเข้ากับฐานที่ได้ประกอบไปก่อนหน้านี้แล้วขันสกรูให้แน่นโดยจะมีลักษณะในรูปที่ 2.4(ข)



รูปที่ 2.3 ส่วนประกอบต่างๆ ในการติดตั้งล้อหน้า

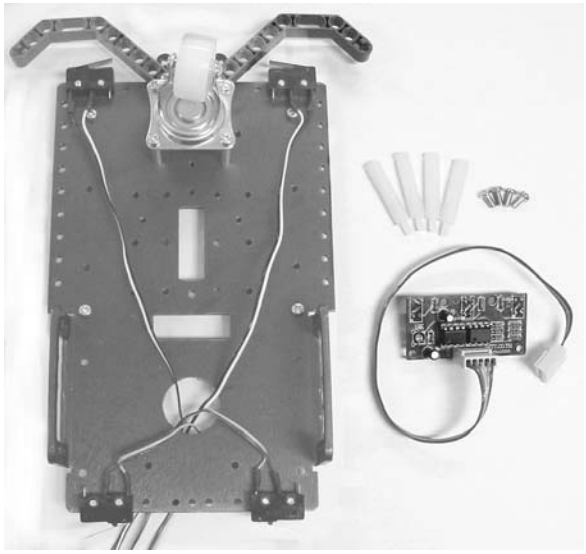


(ก) (ข)

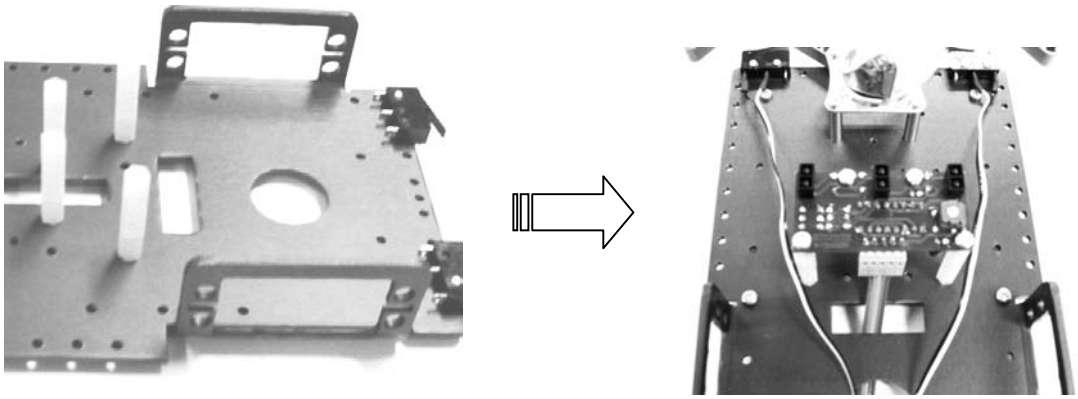
รูปที่ 2.4 ลักษณะการติดตั้งล้อหน้าของ Robot

2.3 การติดตั้งชุดเซนเซอร์ตรวจจับเส้น

ในส่วนของการติดตั้งชุดเซนเซอร์ตรวจจับเส้น นั้นจะต้องทำการติดตั้งฐานรองก่อน เพื่อให้ระยะห่างระหว่างตัวเซนเซอร์กับพื้นไม่ไกลกันมากนัก เพื่อให้วงจรเซนเซอร์ตรวจจับเส้นทำงานได้ดีขึ้น การวางตำแหน่งการติดตั้งนั้นจะทำการติดตั้งในพื้นที่ส่วนกลางด้านล่างของตัว Robot ดังรูปที่ 2.6(ก) จากนั้นทำการประกอบชุดเซนเซอร์เข้าไปกับฐานรองที่เตรียมไว้โดยจะต้องให้ด้านที่มีตัวเซนเซอร์อยู่หงายขึ้นเพื่อให้อยู่ในตำแหน่งที่สามารถตรวจจับเส้นได้ ส่วนสายไฟให้สอดขึ้นไปด้านบนของตัวหุ่นยนต์เพื่อเตรียมติดตั้งกับบอร์ดควบคุม ดังรูปที่ 2.6(ข) จากนั้นขันสกรูให้แน่น



รูปที่ 2.5 ส่วนประกอบต่างๆ ในการติดตั้งชุดเซนเซอร์ตรวจจับเส้น

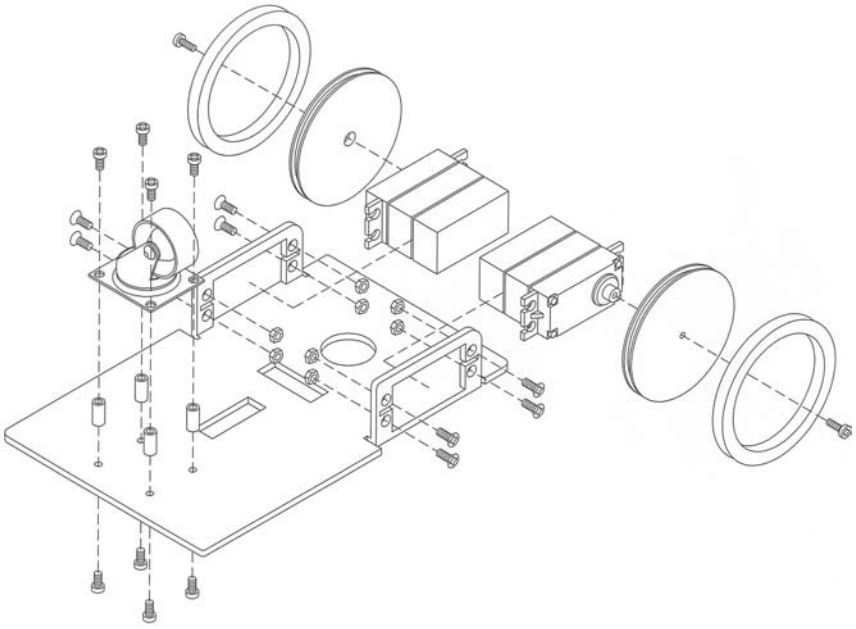


(ก) (ข)

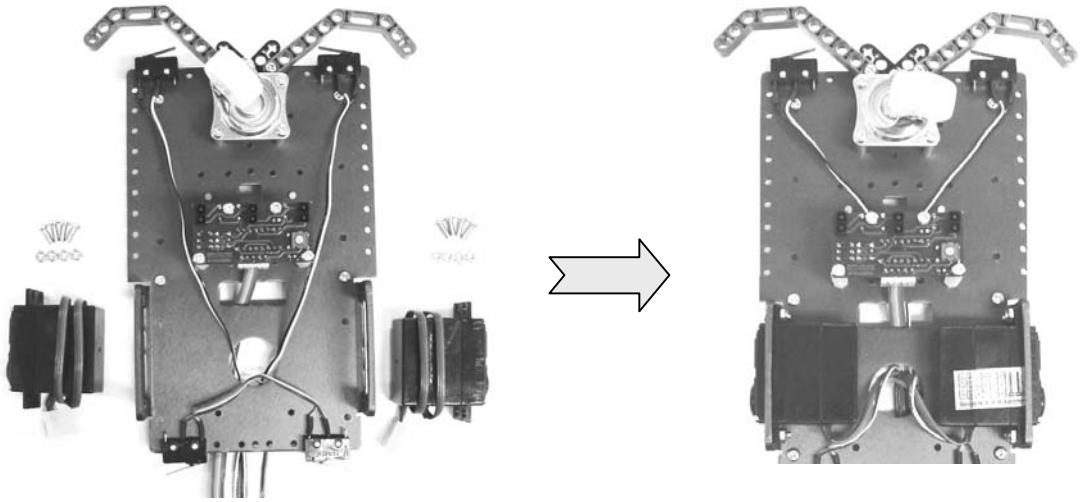
รูปที่ 2.6 แสดงการติดตั้งชุดเซนเซอร์ตรวจจับเส้น

2.4 การติดตั้งเซอร์โวมอเตอร์และชุดล้อหลัง

การติดตั้งตัวเซอร์โวมอเตอร์นี้ ในส่วนของตัวโครง Robot จะมีพื้นที่ที่ออกแบบสำหรับทำการติดตั้งมอเตอร์ให้อยู่แล้ว โดยให้เรานำเอาเซอร์โวมอเตอร์เข้าไปประกอบทั้งสองด้านในพื้นที่สำหรับติดตั้งมอเตอร์แล้วทำการขันยึดสกรูให้แน่น โดยจะต้องให้ด้านที่มีแกนหมุนของมอเตอร์หันไปทางด้านท้ายของตัว Robot เหมือนกันทั้งสองด้าน ดังรูปที่ 2.7 และนำสายไฟรอดผ่านช่อง วงกลมออกไปด้านบนของตัว Robot

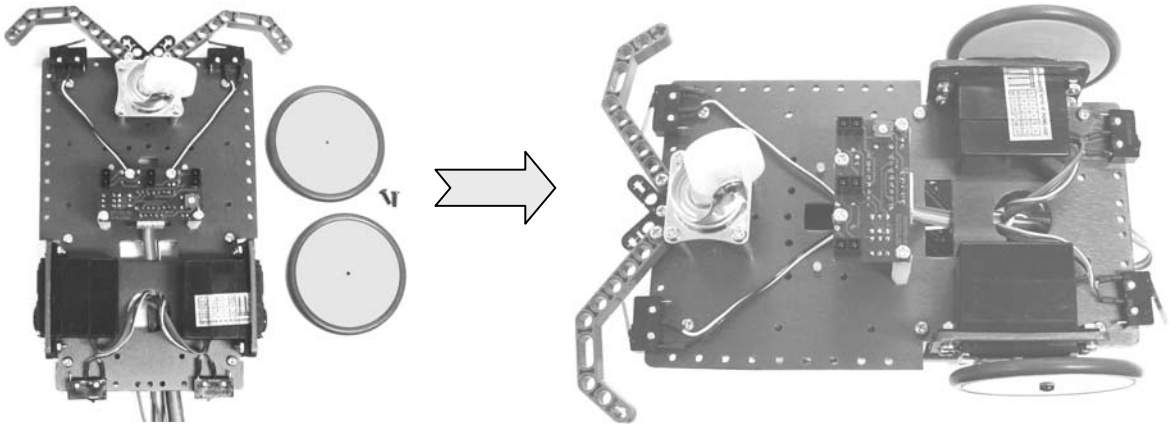


รูปที่ 2.7 แสดงภาพโดยรวมของการติดตั้ง Servo motor และ ชุดล้อหลัง



รูปที่ 2.8 ลักษณะการประกอบ Servo Motor

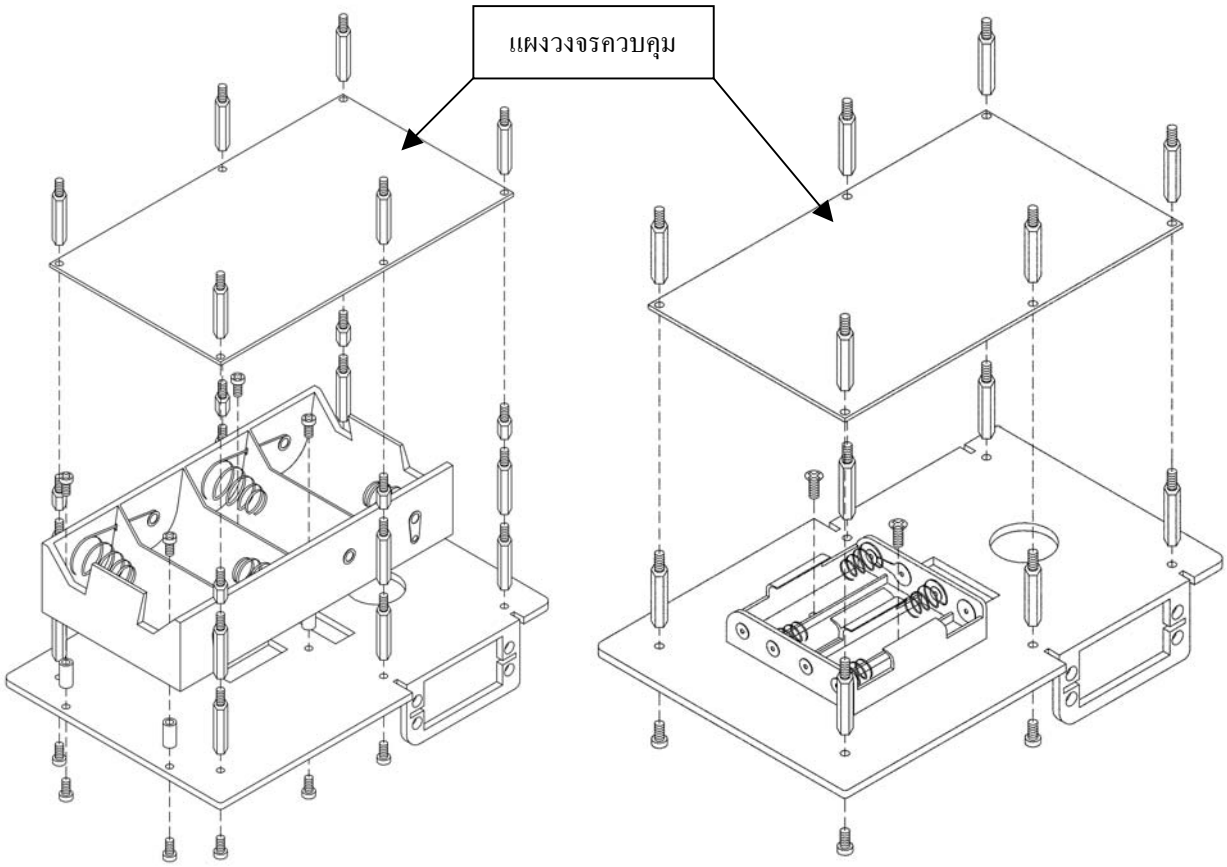
โดยจะต้องประกอบเข้ากับปลายแกนหมุนของมอเตอร์จากนั้นทำการขันสกรูให้แน่นดังรูปที่ 2.9



รูปที่ 2.9 ลักษณะการประกอบล้อหลังเข้ากับ Servo Motor

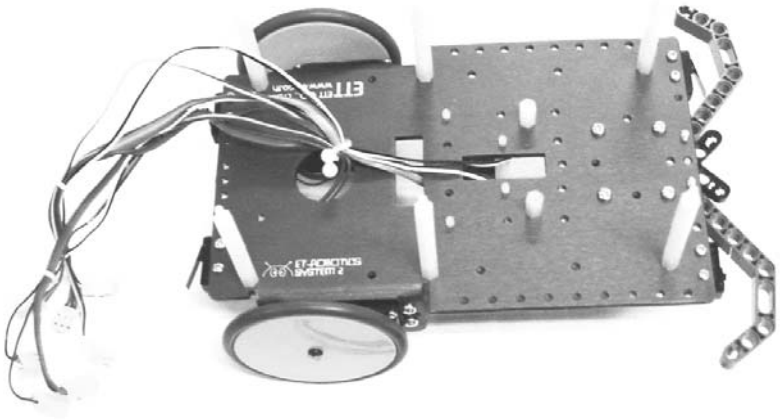
2.5 การติดตั้งรังถ่านและบอร์ดควบคุม

ในส่วนของด้านบนนี้จะมีชิ้นส่วนหลักๆ ที่ต้องทำการติดตั้งนั่นก็คือ รังถ่าน และ แผงวงจรควบคุม โดยรังถ่านที่ให้ไปกับ Robot นี้จะมี 2 ขนาด คือ R-D-TYPE4 และ R-AA-TYPE4 โดยจะมีการติดตั้งโดยรวมดังรูปที่ 2.10



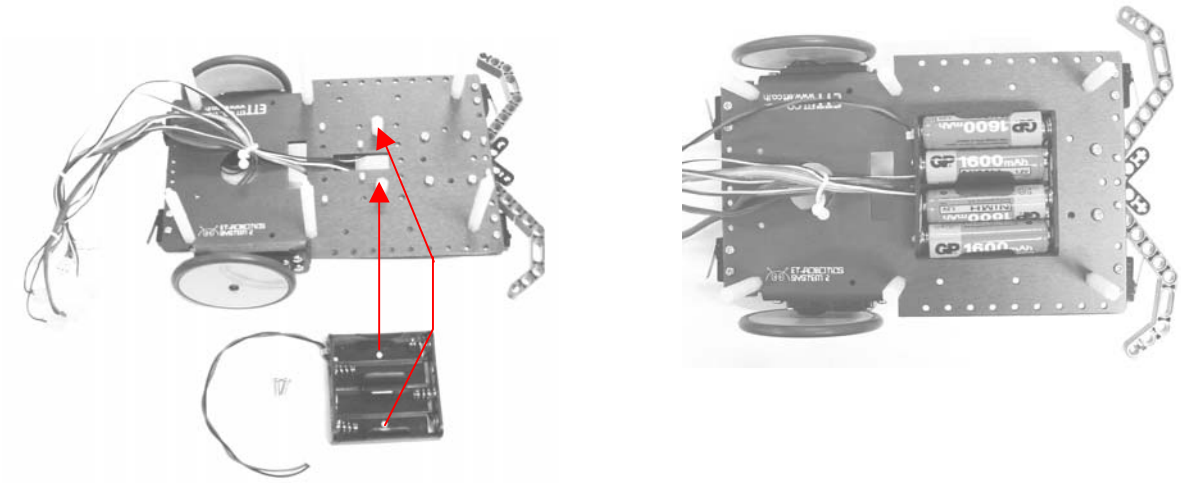
รูปที่ 2.10 แสดงลักษณะภาพโดยรวมของการประกอบบอร์ดควบคุมและ รังถ่านทั้ง 2 แบบ

ซึ่งก่อนที่จะทำการติดตั้งชิ้นส่วนทั้งสองนี้จะต้องทำการติดตั้งฐานรองลงไปบนตัวโครงหุ่นยนต์ก่อน เพื่อรองรับชิ้นส่วนทั้งสองดังรูปที่ 2.11



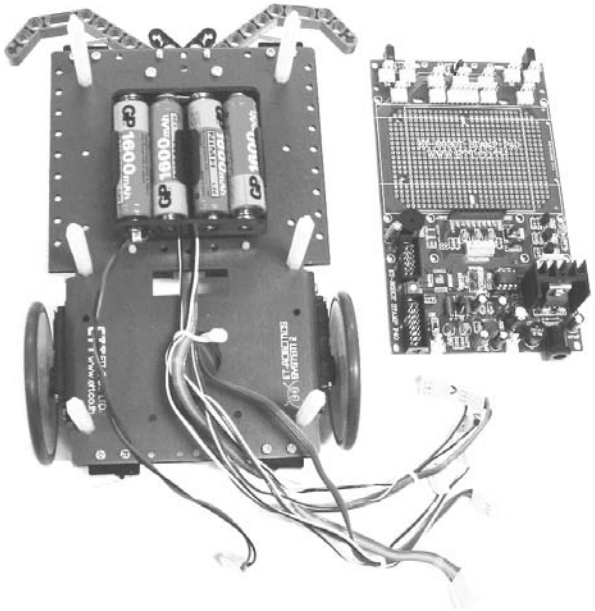
รูปที่ 2.11 แสดงการติดตั้งฐานรองด้านบนของ Robot

จากนั้นให้ทำการประกอบรังถ่านลงไปก่อนในตำแหน่งดังรูปที่ 2.12 ซึ่งจะทำการยึดด้วยสกรูจำนวน 2 ตัว



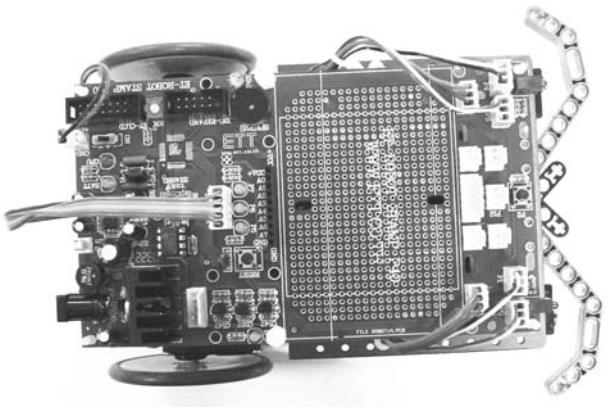
รูปที่ 2.12 ลักษณะการติดตั้งรังถ่าน

ต่อไปทำการประกอบบอร์ดควบคุมเข้าไปในตำแหน่งที่ได้ทำการติดตั้งฐานรองเอาไว้ซึ่งจะมีทั้งหมด 6 จุด โดยให้หันด้านที่มีคอนเนคเตอร์สำหรับต่ออุปกรณ์ต่างๆ ไปทางด้านหน้าของ Robot ดังรูปที่ 2.13



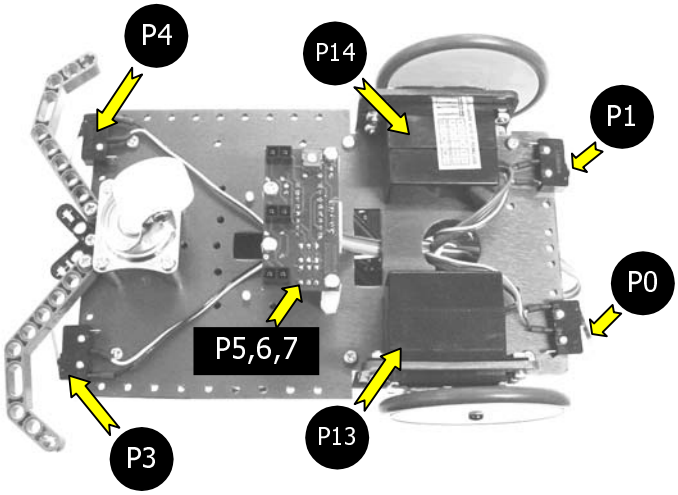
รูปที่ 2.13 ลักษณะการติดตั้งบอร์ดควบคุม

หลังจากนั้นทำการเชื่อมต่อสายสัญญาณต่างๆ เข้ามาในบอร์ดควบคุม โดยเราได้ทำการออกแบบไว้เป็นคอนเน็คเตอร์ เพื่อให้เกิดความสะดวก และ ป้องกันความผิดพลาดจากการใส่กลับด้านต่างๆ ในส่วนของคอนเน็คเตอร์สำหรับชุดเซนเซอร์ตรวจจับเส้นนั้น จะเป็นแบบ 5 PIN สามารถทำการเชื่อมต่อกับชุดตรวจจับเส้นได้โดยตรง และเนื่องจากเราได้จัดเตรียมคอนเน็คเตอร์ที่เป็น I/O ของ CPU ส่วนอื่นๆ เอาไว้มากมายหลายจุด และ การวางขาสัญญาณต่างๆ จะเป็นลักษณะเดียวกัน ทำให้สามารถใช้จุดใดก็ได้ในการเชื่อมต่อกับอุปกรณ์เซนเซอร์ หรือ เซอร์โวมอเตอร์ ทั้งนี้จะต้องสัมพันธ์สอดคล้องกับซอร์ฟแวร์ที่ได้ทำการออกแบบด้วย ซึ่งในที่นี้จะยกตัวอย่างการต่อดังนี้



รูปที่ 2.14 ลักษณะของ Robot เมื่อประกอบและเชื่อมต่อสายสัญญาณต่างๆเสร็จเรียบร้อย

การเชื่อมต่อจุดต่างๆ เข้ากับแผงวงจรบอร์ดควบคุม ET-ROBOT STAMP P40



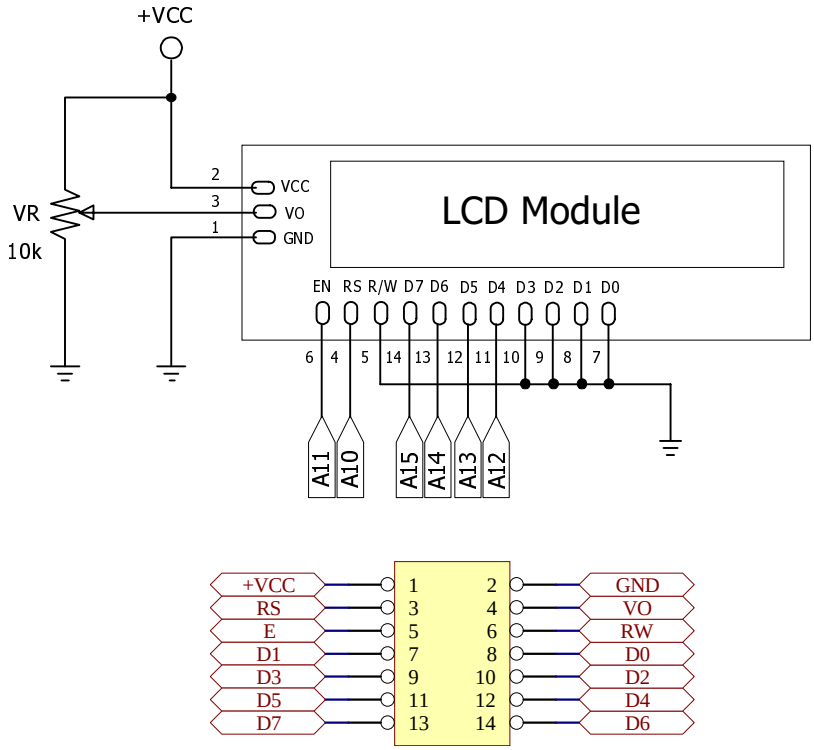
รูปที่ 2.15 การเชื่อมต่อระหว่างขา MCU กับอุปกรณ์ต่างๆ

จากรูปที่ 2.15 จะมีการเชื่อมต่อกันดังนี้

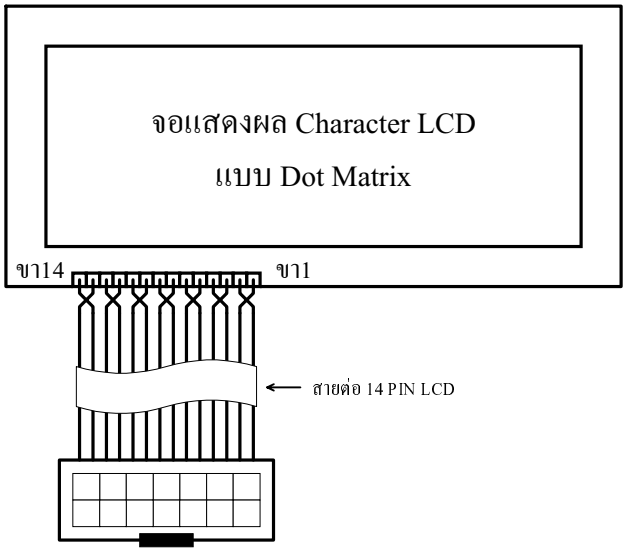
1. สวิตช์กันชนหน้าซ้าย ⟶ P4
2. สวิตช์กันชนหน้าขวา ⟶ P3
3. สวิตช์กันชนหลังซ้าย ⟶ P1
4. สวิตช์กันชนหลังขวา ⟶ P0
5. เซอร์โวมอเตอร์ (Servo Motor) ด้านซ้าย ⟶ P14
6. เซอร์โวมอเตอร์ (Servo Motor) ด้านขวา ⟶ P13
7. ชุดเซ็นเซอร์ตรวจจับเส้น (TRAKER) ⟶ P5 , P6 และ P7

2.6 การต่อ ET-ROBOT STAMP กับจอ LCD

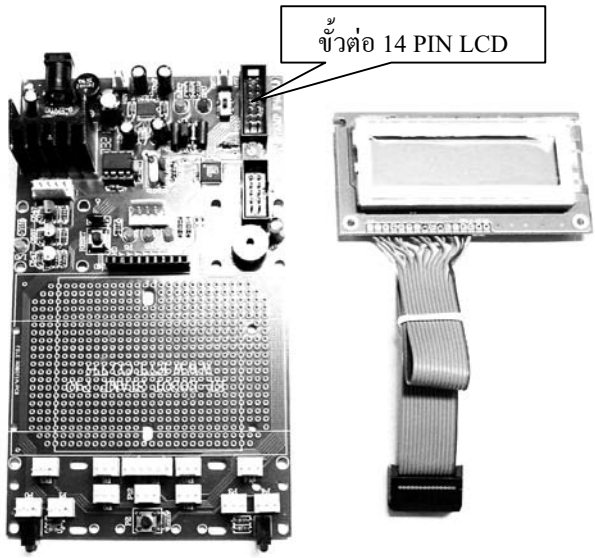
จากการที่บอร์ด ET-ROBOT STAMP P40 ถูกออกแบบให้สามารถทำการเชื่อมต่อจอแสดงผล LCD ได้ โดยจะมีขั้วต่อ 14 PIN และ วงจรตัวต้านทานปรับค่าสำหรับปรับความสว่างของจอ LCD ไว้เรียบร้อยแล้ว ซึ่งในส่วนของการนำไปใช้งานอาจจะนำไปแสดงผลข้อความต่างๆ ในการทำงานของ Robot เพื่อเพิ่มความน่าสนใจ และ เพื่อฝึกทักษะการเรียนรู้ใช้งานจอแสดงผลแบบ LCD โดยจอ LCD ที่จะนำมาต่อกับขั้วต่อ 14 PIN ET-CLCD ได้นั้นจะต้องเป็นชนิดตัวอักษรเท่านั้น (Character) เท่านั้น ไม่สามารถนำไปต่อกับ LCD ประเภทกราฟฟิกได้ ซึ่งรูปแบบในการติดต่อกับ LCD ที่เราจัดไว้ให้นี้จะเป็นแบบ 4 บิตข้อมูล รวมกับขาสัญญาณ RS และ EN แล้วจะใช้ขาสัญญาณในการควบคุมจากตัว Basic Stamp ทั้งหมด 6 ขา ดังวงจรในรูปที่ 2.16



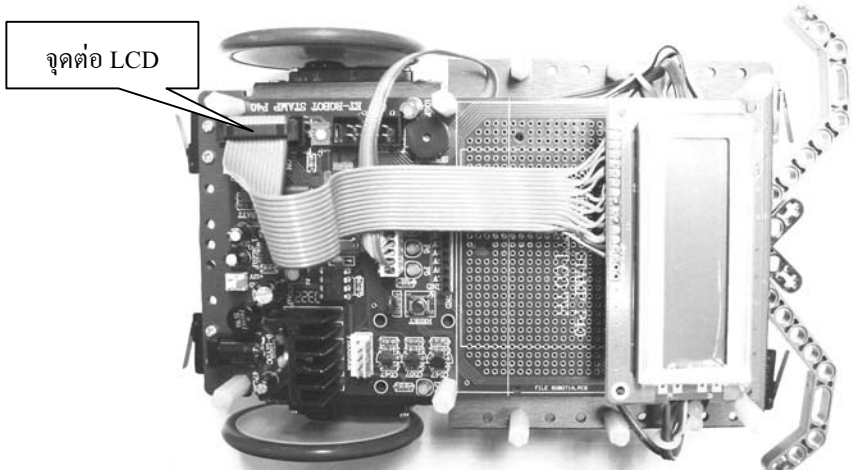
รูปที่ 2.16 แสดงวงจรการต่อ LCD กับบอร์ด ET-ROBOT STAMP P40



รูปที่ 2.17 แสดงการต่อสายแพรมายัง LCD



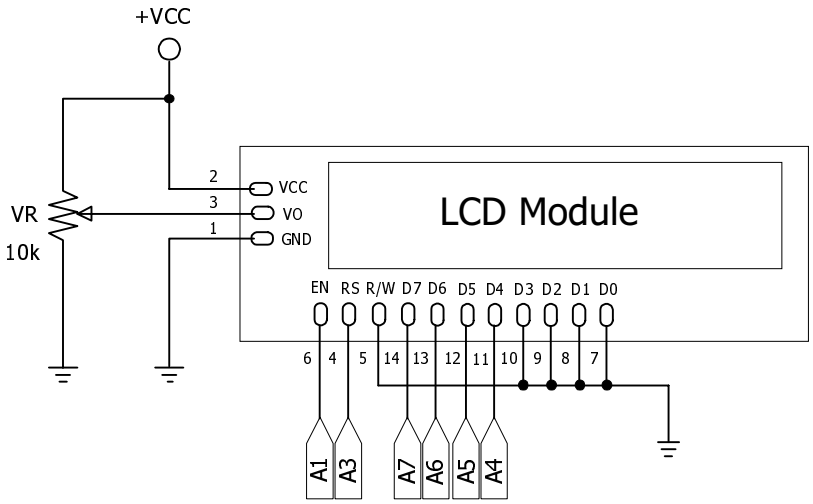
รูปที่ 2.18 แสดงบอร์ด ET-ROBOT STAMP และ จอ LCD



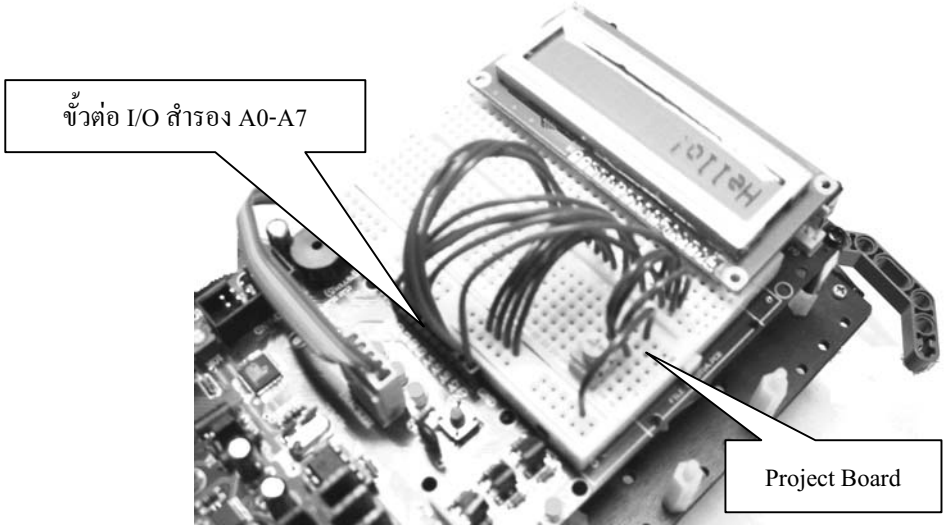
รูปที่ 2.19 แสดงลักษณะการต่อจอ LCD เข้ากับ Robot

ก่อนการใช้งาน LCD ควรมีการปรับค่าความสว่างในการแสดงผลของจอ LCD ซึ่งสามารถทำได้โดยปรับค่าที่ตัวต้านทานปรับค่า 10 k จนได้ระดับที่สามารถมองเห็นตัวอักษรได้อย่างชัดเจน

เนื่องจากขั้วต่อ ET-CLCD มีการจัดวงจรที่ไม่สอดคล้องกับคำสั่ง LCDOUT หรือ คำสั่ง LCDCMD ดังนั้น หากต่อ LCD ที่ขั้วต่อดังกล่าวนี้จะต้องใช้วิธีการเขียนโปรแกรมควบคุมจอ LCD แบบเบื้องต้นทั่วไป แต่หากต้องการใช้คำสั่งสำเร็จ เช่น LCDOUT หรือ LCDCMD เหล่านี้จะต้องจัดวงจรใหม่โดยอาจจะใช้ Project Board ต่อก็ได้ เช่น ต่อขาสัญญาณจาก A0 ถึง A7 ไปควบคุม ซึ่งหากจัดวงจรตามรูปแบบของคำสั่ง LCDOUT แล้วจะสามารถทำให้การใช้งาน LCD ง่ายยิ่งขึ้น โดยแสดงตัวอย่างวงจรดังรูปที่ 2.20 และ ลักษณะการต่อดังรูปที่ 2.21



รูปที่ 2.20 วงจรเชื่อมต่อ LCD เพื่อใช้กับคำสั่ง LCDOUT



รูปที่ 2.21 ลักษณะการเชื่อมต่อ LCD บนโปรเจ็คบอร์ด

* โดยการใช้งาน LCD สามารถศึกษาได้จากตัวอย่างที่ 4.14 , 4.15 และ 4.16 ในท้ายบทที่ 4

บทที่ 3

การติดตั้งโปรแกรมและการดาวน์โหลดโปรแกรม

3.1 การเตรียมการในส่วนของฮาร์ดแวร์

ก่อนจะมีการเขียนโปรแกรม และ ดาวน์โหลดโปรแกรมควรมีการเตรียมพร้อมในส่วนของฮาร์ดแวร์เสียก่อนโดยสามารถทำได้ตามขั้นตอนต่างๆ ดังต่อไปนี้

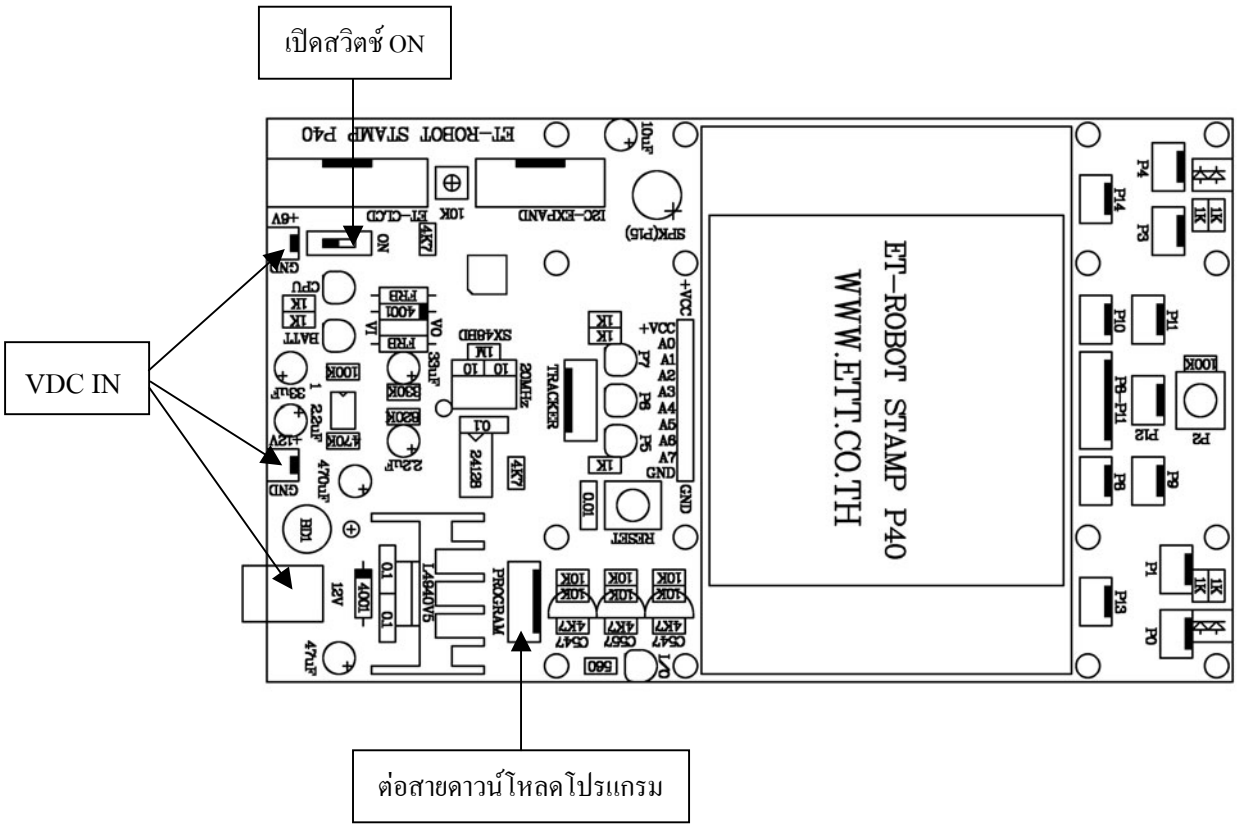
ขั้นตอนที่ 1 ทำการต่อวงจรหรือ เซนเซอร์ต่างๆ ของหุ่นยนต์ตามต้องการให้เรียบร้อย

ขั้นตอนที่ 2 เชื่อมต่อบอร์ด ET-ROBOT STAMP P40 เข้ากับเครื่องคอมพิวเตอร์ทางพอร์ตอนุกรม (COM Port)

ขั้นตอนที่ 3 เชื่อมต่อแหล่งจ่ายไฟให้กับบอร์ด (VDC IN) ซึ่งสามารถใส่ได้หลายจุดให้ใส่เพียงจุดใดจุดหนึ่งเท่านั้น

ขั้นตอนที่ 4 เชื่อมต่อคอมพิวเตอร์ทาง Serial Port เข้ากับขั้วต่อ PROGRAM (Connector 5 Pin) ตามตำแหน่งดังรูป

ขั้นตอนที่ 5 เปิดสวิตช์ ON



รูปที่ 3.1 ลักษณะของบอร์ด ET-ROBOT STAMP P40

3.2 การติดตั้งโปรแกรม BASIC STAMP

การที่เราจะสามารถเขียนโปรแกรมเพื่อควบคุมให้ Robot ทำงานได้นั้น ก่อนอื่นจะต้องทำการติดตั้งซอร์ฟแวร์ สำหรับใช้เป็นตัวเขียนโปรแกรม (Editor) และ ใช้ในการโปรแกรม (Programming) ข้อมูลลงในตัว CPU เบสิกแสตมป์ก่อน โดยซอร์ฟแวร์ดังกล่าวนี้มีชื่อเรียกว่า Basic Stamp Editor มีด้วยกันอยู่หลายเวอร์ชันสามารถเลือกใช้ได้ตามความต้องการเช่น เวอร์ชัน V1.33 และ V2.0 จะทำงานบนระบบปฏิบัติการวินโดว์โดยมีความต้องการของระบบ (System Requirements) ดังต่อไปนี้

- Windows 95, Windows 98, Windows ME, Windows NT 4.0, Windows 2000 or Windows XP
- 80486 (or greater) processor
- 16 MB RAM (24 MB Recommended)
- 6 MB Free Hard Drive Space
- 256 Color VGA Video Card (24-bit SVGA recommended)
- 1 Available COM port

1. ให้ใส่แผ่น CD โปรแกรมแล้วเข้าไปใน Folders ที่อยู่ของไฟล์ Setup สำหรับทำการติดตั้งโปรแกรมซึ่งจะใช้เวอร์ชันไหนก็ได้ โดยให้ดับเบิลคลิกเมาส์ ที่ไอคอนของเวอร์ชันที่ต้องการติดตั้งซึ่งในตัวอย่างนี้จะติดตั้งเวอร์ชัน V1.33

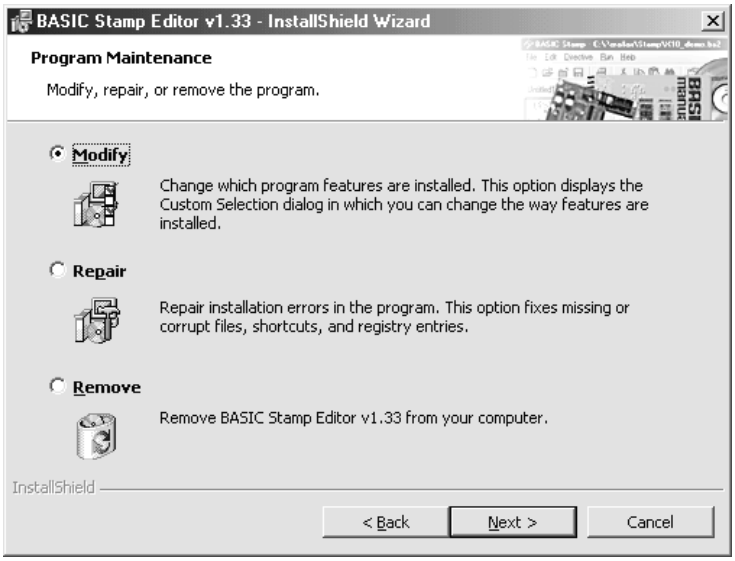
Setup_Stamp_Editor_6MB.exe : สำหรับเวอร์ชัน V1.33

Setup_Stamp_Editor_v2.0_Beta_1_6MB.exe : สำหรับเวอร์ชัน V2.0

2. จากนั้นจะปรากฏหน้าต่างดังรูปที่ 3.2 ให้คลิกเลือก Next> เพื่อเข้าสู่ขั้นตอนต่อไป

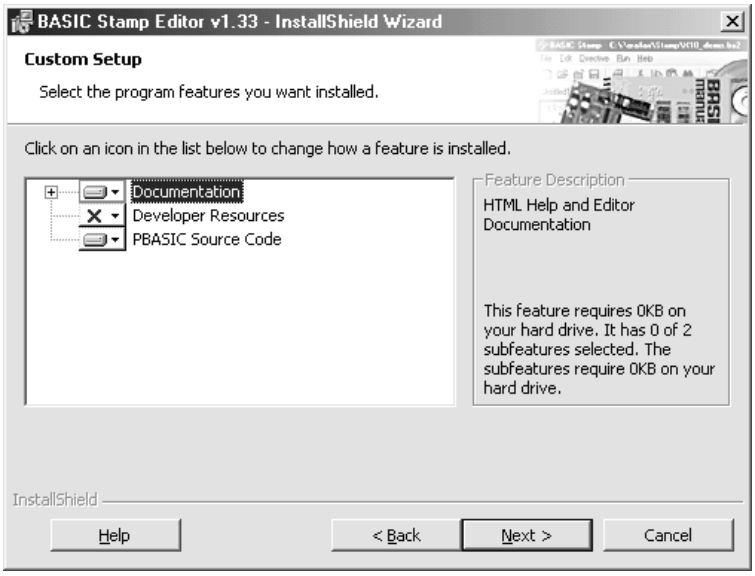
รูปที่ 3.2 แสดงหน้าต่างตอบรับการติดตั้งโปรแกรม

3. เลือกเป็น Modify แล้วคลิกเมาส์ไปที่ NEXT>



รูปที่ 3.3 หน้าต่างสำหรับเลือกรูปแบบของการติดตั้งโปรแกรม

4. จะปรากฏหน้าต่างดังรูปที่ 3.4 ให้คลิกเมาส์ที่ Next>

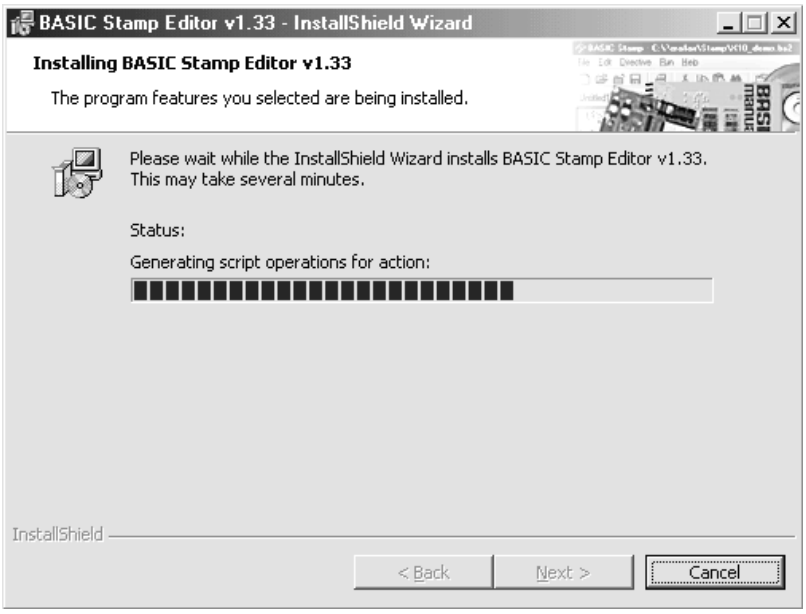


รูปที่ 3.4 แสดงหน้าต่าง Custom Setup

5. จะปรากฏหน้าต่างแสดงรายละเอียดต่างๆ ดังรูปที่ 3.5 ให้คลิกเลือกเมาส์ที่ Install เพื่อเริ่มต้นการติดตั้งโปรแกรม

รูปที่ 3.5 หน้าต่างแสดงรายละเอียดของการติดตั้งโปรแกรม

7. จากนั้นโปรแกรมจะทำการติดตั้งให้เราเองโดยอัตโนมัติโดยจะแสดงแถบความก้าวหน้าดังรูปที่ 3.6



รูปที่ 3.6 หน้าต่างแสดงความก้าวหน้าของการติดตั้งโปรแกรม

8. เมื่อกระบวนการติดตั้งเสร็จสิ้นจะปรากฏหน้าต่างดังรูปที่ 3.7 ให้คลิกเมาส์เลือกที่ Finish เป็นการเสร็จสิ้นการติดตั้งโปรแกรม

รูปที่ 3.7 หน้าต่างแสดงการสิ้นสุดการติดตั้งโปรแกรม

3.3 การเขียนโปรแกรมและการดาวน์โหลด

หลังจากทำการติดตั้งโปรแกรม และ การเตรียมการในส่วนของฮาร์ดแวร์เรียบร้อยแล้วต่อไปจะเป็นการแสดงตัวอย่างการใช้งานโปรแกรม Basic Stamp Editor โดยสามารถทำได้ดังขั้นตอนต่างๆ ต่อไปนี้

- เปิดโปรแกรม Basic Stamp Editor ซึ่งสามารถทำได้โดยดับเบิลคลิกที่ไอคอน BASIC StampEditor หรือคลิกเลือกที่ Start → Programs→Parallax Inc ดังรูปที่ 3.8

รูปที่ 3.8 แสดงการเข้าสู่โปรแกรม Stamp Editor

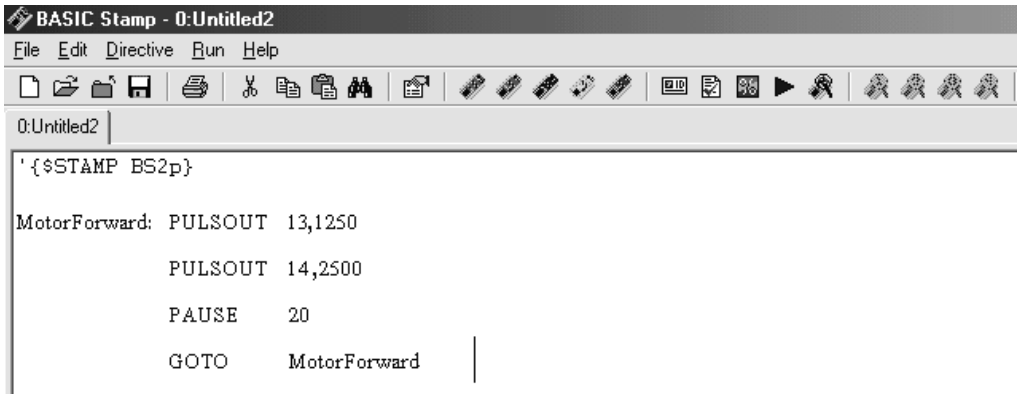
- จะปรากฏหน้าต่างการทำงานดังรูปที่ 3.9 ให้เลือกเบอร์ของ CPU Basic Stamp ในที่นี้คือ BS2p

รูปที่ 3.9 แสดงการเลือกเบอร์ของตัว Basic Stamp ที่ใช้

หลังจากเลือกเบอร์แล้วจะปรากฏข้อความในพื้นที่ของการเขียนโปรแกรมดังรูปที่ 3.10

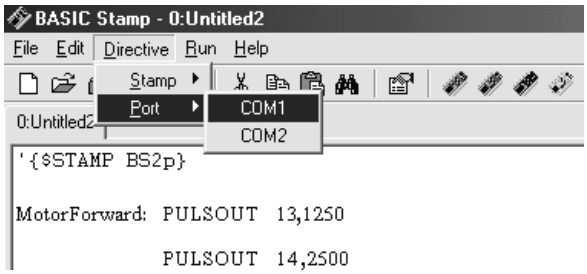
รูปที่ 3.10 แสดงเบอร์ของ Basic Stamp ที่ถูกเลือก

- เขียนโค้ดโปรแกรมที่ต้องการ ดังเช่นตัวอย่างด้านล่าง



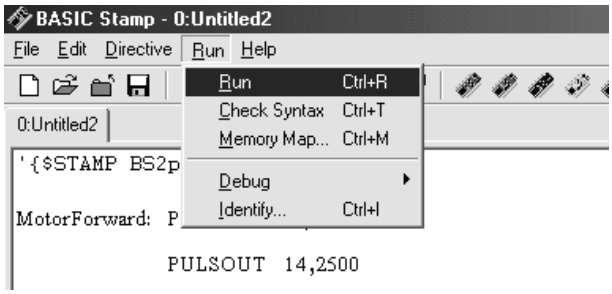
รูปที่ 3.11 แสดงตัวอย่างการเขียนโค้ดโปรแกรม

- เลือกพอร์ตสื่อสารอนุกรม COM1 หรือ COM2 เลือกให้ตรงกับที่ใช้งานอยู่ (พอร์ตที่ต่ออยู่กับบอร์ด)



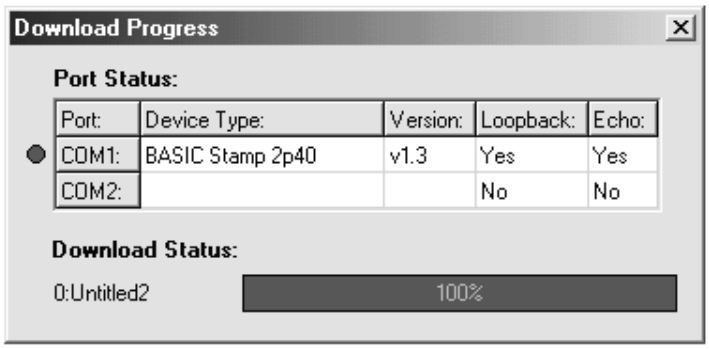
รูปที่ 3.12 แสดงการเลือกพอร์ตการสื่อสารอนุกรมสำหรับการดาวน์โหลดโปรแกรม

- เมื่อเรียบร้อยแล้วสั่ง RUN (Ctrl+R) เพื่อดาวน์โหลดโปรแกรม ดังรูปที่ 3.13



รูปที่ 3.13 การสั่ง Run เพื่อดาวน์โหลดโปรแกรม

ซึ่งจะปรากฏหน้าต่างแสดงการโปรแกรมข้อมูลลง CPU ดังรูปด้านล่างนี้ ตัวเบสิกแสตมป์ก็จะเริ่มทำงานตามคำสั่งที่เราเขียนขึ้น ซึ่งเราสามารถถอดสายดาวน์โหลดออกได้ เพื่อปล่อยให้ Robot วิ่งได้อย่างอิสระตามโปรแกรมที่เราออกแบบขึ้น



รูปที่ 3.14 หน้าต่างแสดงสภาวะการโปรแกรมข้อมูลลง CPU Basic Stamp

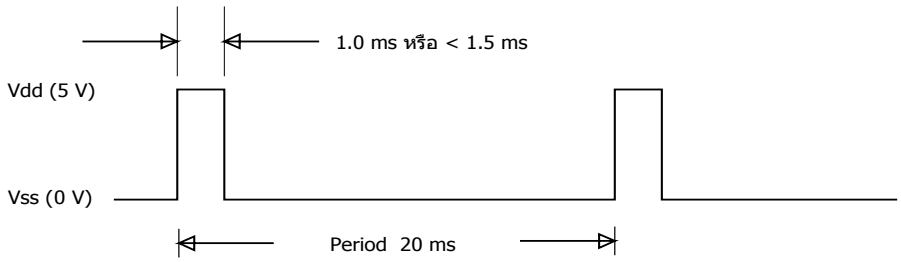
บทที่ 4
วิธีการควบคุมหุ่นยนต์

การเคลื่อนที่ของหุ่นยนต์นี้จะอาศัยมอเตอร์เป็นตัวขับเคลื่อนซึ่งมอเตอร์ที่ใช้นี้จะเป็นชนิด เซอร์โวมอเตอร์ (Servo Motor) ดังนั้นก่อนที่เราจะเขียนโปรแกรมเพื่อควบคุมให้หุ่นยนต์เคลื่อนที่ได้นั้นเราจะต้องทราบหลักการควบคุมมอเตอร์ชนิดนี้เสียก่อน ดังรายละเอียดต่อไปนี้

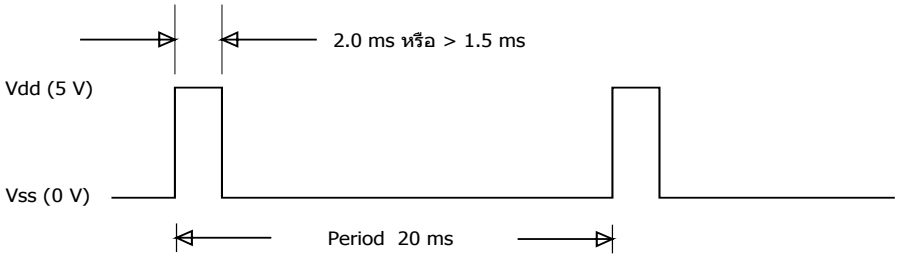
4.1 หลักการทำงานของเซอร์โวมอเตอร์

เซอร์โวมอเตอร์ที่เราได้จัดให้ในชุดของ ET-ROBOT นี้ได้ผ่านการปรับแต่งการทำงานของเซอร์โวมอเตอร์ให้สามารถหมุนเป็นวงรอบ (360 องศา) เรียบร้อยแล้วโดยวิธีในการควบคุมให้มอเตอร์หมุน จะมีลักษณะดังนี้

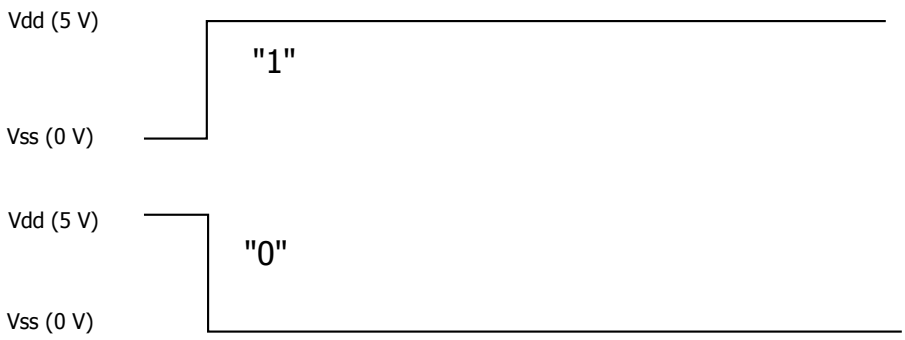
- การควบคุมให้มอเตอร์หมุนทางด้านซ้ายจะต้องป้อนสัญญาณพัลส์ที่มีขนาดความกว้างพัลส์ 1 ms หรือ ให้น้อยกว่า 1.5 ms โดยจะต้องป้อนสัญญาณพัลส์นี้ทุกๆ 20 ms (หรือในช่วงประมาณ 20ms – 30ms)



- การควบคุมให้มอเตอร์หมุนทางด้านขวาจะต้องป้อนสัญญาณพัลส์ที่มีขนาดความกว้างพัลส์ 2 ms หรือ ไม่ต่ำกว่า 1.5 ms และจะต้องป้อนสัญญาณพัลส์นี้ทุกๆ 20 ms (หรือในช่วงประมาณ 20ms – 30ms) เช่นกัน



- การควบคุมให้มอเตอร์หยุดหมุน ทำได้โดยการส่งลอจิก “0” หรือ “1” ให้กับมอเตอร์ หรือ ก็คือการไม่จ่ายสัญญาณพัลส์ให้กับมอเตอร์นั่นเอง



STOP Motor

การควบคุมการทำงานของ Servo motor จะใช้หลักการสร้างสัญญาณพัลส์ขนาดความกว้างต่างๆ ส่งไปควบคุมการทำงานของมอเตอร์ โดยใน CPU เบสิกแสตมป์ที่ใช้ภาษาเบสิกนั้นจะใช้คำสั่ง PULSOUT Pin,Period เพื่อสร้างสัญญาณพัลส์ โดยค่าเวลาหนึ่งหน่วยของค่า Period ของ CPU เบสิกแสตมป์ (SX48BD) ที่ความถี่ 20 MHz จะเท่ากับ 0.8 uS เช่น คำสั่ง PULSOUT Pin,1000 ก็จะได้ค่าเวลาเท่ากับ 1000 x 0.8 uS = 800 uS หรือ 0.8 mS แต่ในการควบคุมเซอร์โวมอเตอร์ในที่นี้จะยึดที่ 2 mS และ 1 mS โดยค่า Period จะเป็นดังนี้

ที่ค่าเวลา **1 ms** สั่งให้มอเตอร์หมุนตามเข็มนาฬิกา (CW)

1 ms / 0.8 uS = 1250

* ดังนั้นจะต้องใช้คำสั่ง PULSOUT Pin,1250

ที่ค่าเวลา **2 mS** สั่งให้มอเตอร์หมุนทวนเข็มนาฬิกา (CCW)

2 mS / 0.8 uS = 2500

* ดังนั้นจะต้องใช้คำสั่ง PULSOUT Pin,2500

*** หมายเหตุ** รายละเอียดของคำสั่ง PULSOUT ดูได้จากภาคผนวก

4.2 การควบคุมการเคลื่อนที่ของหุ่นยนต์

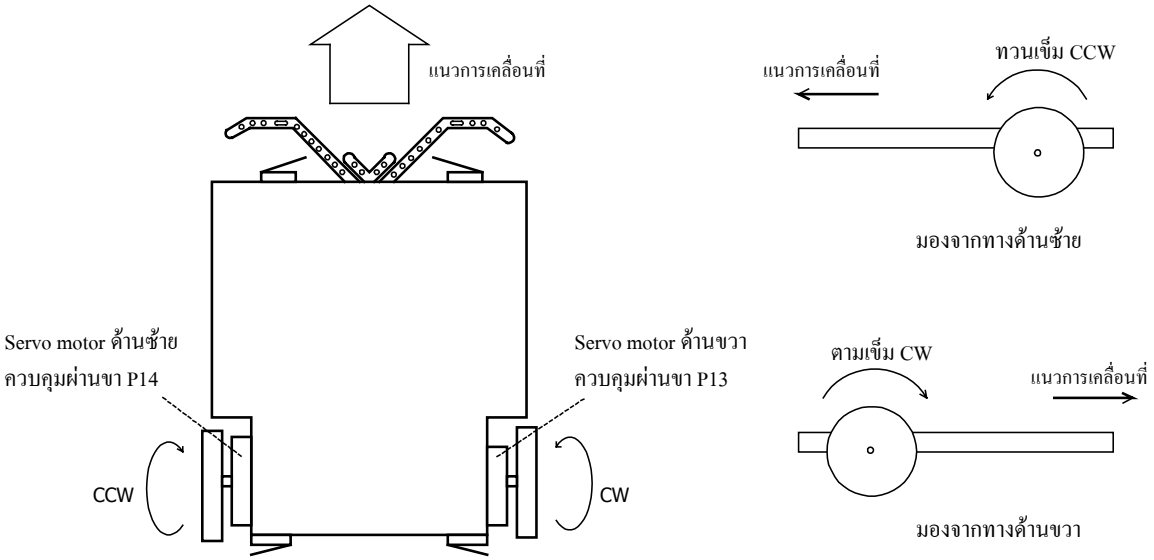
ในการเคลื่อนที่ของหุ่นยนต์นั้นสามารถทำได้หลายทิศทางโดยใช้วิธีการสั่งงานมอเตอร์ให้หมุนในรูปแบบต่างๆ ซึ่งพอจะสรุปได้ดังนี้

- การเคลื่อนที่ไปด้านหน้า (Forward)

การเคลื่อนที่ในรูปแบบนี้จะต้องควบคุมให้มอเตอร์ทั้งสองตัวมีทิศทางการหมุนไปเป็นแนวเดียวกัน แต่จากการที่ตำแหน่งในการติดตั้งเซอร์โวมอเตอร์นั้นวางในทิศทางที่กลับด้านกันอยู่ดังนั้นในการสั่งให้มอเตอร์หมุนเพื่อให้เดินหน้านั้นจึงต้องสั่งให้มอเตอร์ตัวหนึ่งหมุนในทิศทางตามเข็มนาฬิกา และ อีกตัวหนึ่งทวนเข็มนาฬิกา ซึ่งจะมีลักษณะการหมุนดังรูปที่ 4.1 (เพราะตัว Motor ทั้ง 2 ตัวนั้นอยู่คนละด้านกัน คือ ด้านซ้ายและขวา เพราะฉนั้นทิศทางการหมุนจึงกลับด้านกันอยู่)

ตัวอย่างที่ 4.1 การควบคุมให้หุ่นยนต์เคลื่อนที่ไปด้านหน้า

```
MotorForward:  PULSOUT   13,1250      'ส่งพัลส์ 1 mS ให้มอเตอร์ด้านขวา หมุนตามเข็มนาฬิกา (CW)
               PULSOUT   14 ,2500     'ส่งพัลส์ 2 mS ให้มอเตอร์ด้านซ้าย หมุนทวนเข็มนาฬิกา (CCW)
               PAUSE     20           ' หน่วงเวลา 20 mS
               GOTO      MotorForward ' วนลูป
```



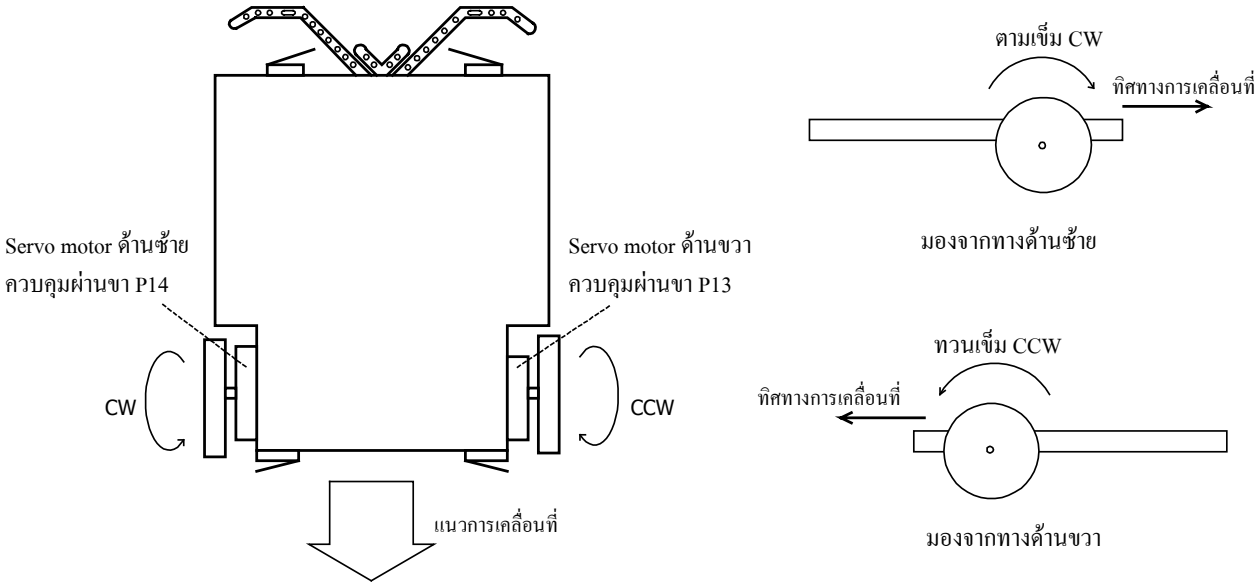
รูปที่ 4.1 แสดงลักษณะการเคลื่อนที่ของหุ่นยนต์ไปด้านหน้า (Forward)

- การเคลื่อนที่ไปด้านหลัง (Backward)

ในการเคลื่อนที่ไปด้านหลังนี้ จะมีการควบคุมมอเตอร์ให้มีการทำงานตรงข้ามกันกับการเคลื่อนที่ไปด้านหน้าตามในตัวอย่างที่ผ่านมา โดยลักษณะการควบคุมจะเป็นดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 4.2 การควบคุมหุ่นยนต์ให้เคลื่อนที่ไปด้านหลัง

```
MotorBackward:  PULSOUT   13, 2500     'ส่งพัลส์ 2 mS ให้มอเตอร์ด้านขวาหมุนทวนเข็มนาฬิกา (CCW)
                PULSOUT   14 , 1250    'ส่งพัลส์ 1 mS ให้มอเตอร์ด้านซ้ายหมุนตามเข็มนาฬิกา (CW)
                PAUSE     20           ' หน่วงเวลา 20 mS
          GOTO      MotorForward       ' วนลูป
```



รูปที่ 4.2 แสดงลักษณะการเคลื่อนที่ของหุ่นยนต์ไปด้านหลัง (Backward)

- การเลี้ยวซ้าย (Turn Left)

ในส่วนของการเลี้ยวนั้นสามารถกระทำได้ใน 2 ลักษณะ คือ ให้ล้อด้านหนึ่งหยุดหมุน แล้ว สั่งให้ล้ออีกด้านหนึ่งหมุน และ อีกวิธีคือการสั่งให้ล้อทั้งสองข้างหมุนในทิศทางตรงข้ามกัน ซึ่งในแบบแรกจะควบคุมได้ง่ายกว่า แต่ในแบบที่สองจะมีความคล่องตัวมากกว่าโดยเฉพาะการเลี้ยวในมุมแคบๆ ซึ่งในส่วนของการเลี้ยวไปทางด้านซ้ายนั้นจะมีลักษณะการควบคุมตามตัวอย่างดังนี้

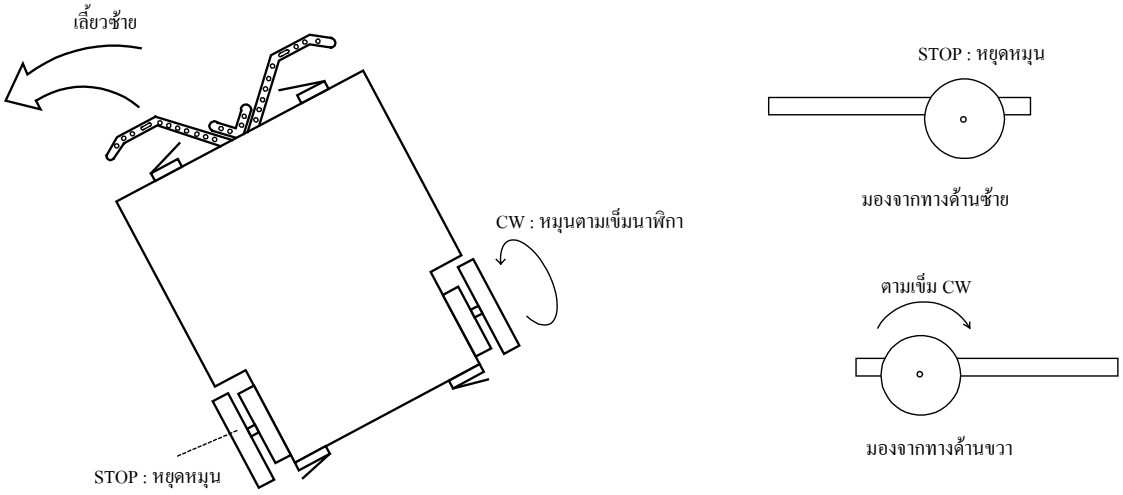
ตัวอย่างที่ 4.3 การเลี้ยวซ้ายแบบล้อหมุนด้านเดียว

```
TurnLeft:   LOW      14          ‘ หยุดการหมุนของมอเตอร์ด้านซ้าย
            PULSOUT  13 , 1250   ‘ส่งพัลส์ 1 mS ให้มอเตอร์ด้านขวาหมุนตามเข็มนาฬิกา (CW)
            PAUSE    20          ‘ หน่วงเวลา 20 mS
            GOTO     TurnLeft    ‘ วนลูปทำให้หุ่นยนต์หมุนวนทางด้านซ้าย
```

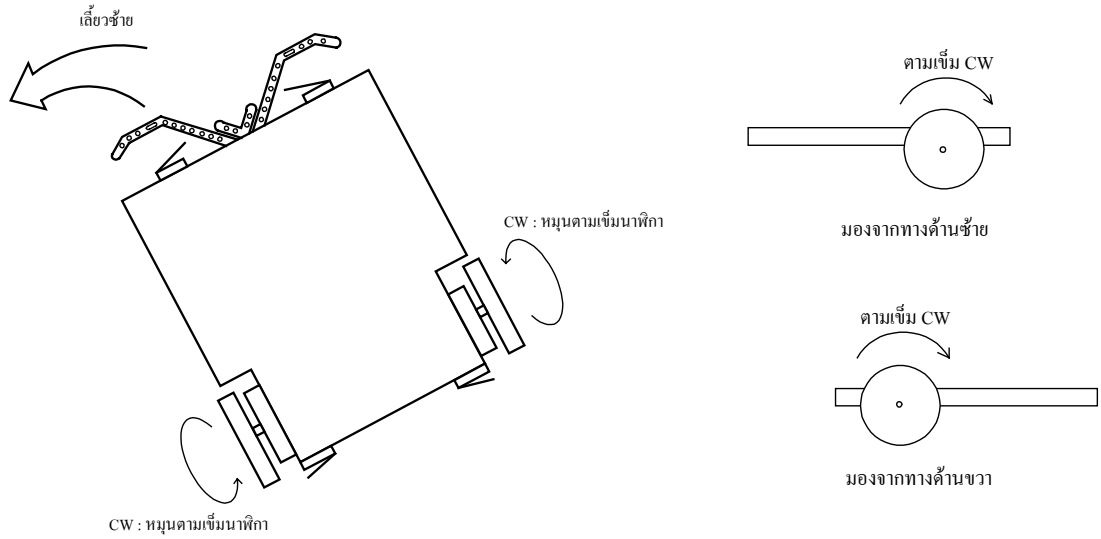
ตัวอย่างที่ 4.4 การเลี้ยวซ้ายแบบล้อหมุนทั้งสองด้าน

```
TurnLeft:   PULSOUT  14 ,1250    ‘ส่งพัลส์ 1 mS ให้มอเตอร์ด้านซ้ายหมุนตามเข็มนาฬิกา (CW)
            PULSOUT  13 , 1250   ‘ส่งพัลส์ 1 mS ให้มอเตอร์ด้านขวาหมุนตามเข็มนาฬิกา (CW)
            PAUSE    20          ‘ หน่วงเวลา 20 mS
            GOTO     TurnLeft    ‘ วนลูปทำให้หุ่นยนต์หมุนวนทางด้านซ้าย
```

จากตัวอย่างที่ 4.4 จะเห็นว่ามีการสั่งให้มอเตอร์หมุนในทิศทางตามเข็มนาฬิกาทั้ง 2 ตัว ทั้งนี้เพราะการวางตำแหน่งของมอเตอร์บนตัวหุ่นยนต์นั้นกลับด้านกัน ซึ่งจากตัวอย่างที่ 2 นั้นจะทำให้ล้อด้านซ้ายหมุนไปด้านหลัง ส่วนล้อด้านขวาจะหมุนไปด้านหน้า ทำให้ทิศทางของหุ่นยนต์หมุนไปทางด้านซ้ายและหากเราส่งพัลส์ในลักษณะเช่นนี้ไปเลื่อยๆ หุ่นยนต์ก็จะหมุนวนไปทางซ้ายตลอด ดังนั้นเมื่อสั่งให้หุ่นยนต์เลี้ยวไปถึงจุดที่ต้องการแล้วให้ทำการหยุดการหมุนของมอเตอร์ทั้งสองด้วยการส่งลอจิก “0” ให้กับมอเตอร์ทั้งคู่ (P13 = 0,P14 =0)



รูปที่ 4.3 ลักษณะการเลี้ยวของหุ่นยนต์ไปทางด้านซ้ายด้วยการหมุนล้อเดียว



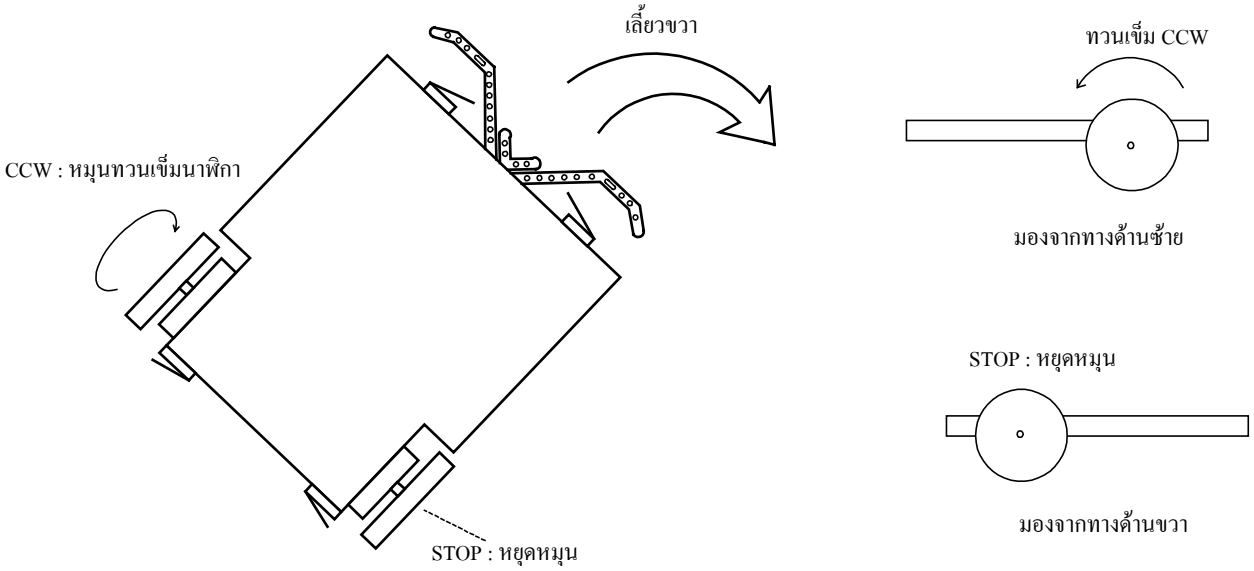
รูปที่ 4.4 ลักษณะการเลี้ยวของหุ่นยนต์ไปทางด้านซ้ายโดยการหมุนล้อทั้งสองข้าง

- การเลี้ยวขวา (Turn Right)

การควบคุมให้หุ่นยนต์เลี้ยวไปด้านขวา นั้นจะมีลักษณะตรงข้ามกับ การควบคุมแบบเลี้ยวซ้าย แต่จะมีลักษณะรูปแบบการเลี้ยวเป็น 2 รูปแบบ เช่นเดียวกันตามตัวอย่างดังนี้

ตัวอย่างที่ 4.5 การเลี้ยวขวาแบบหมุนล้อเดียว

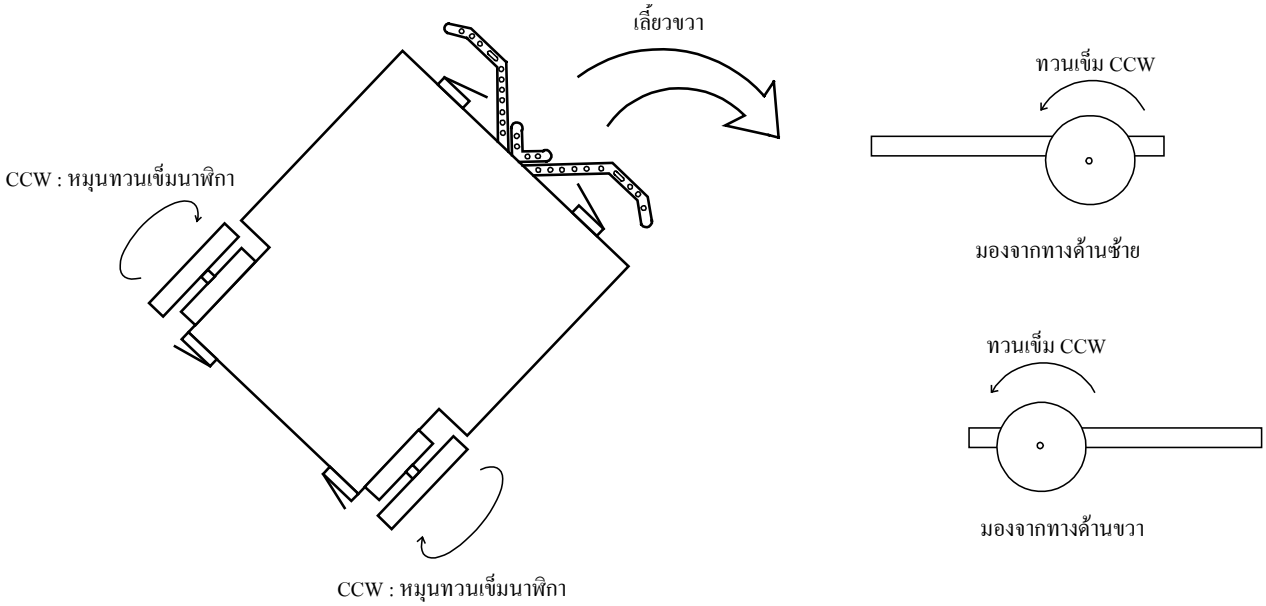
```
TurnRight:    LOW      13          ‘ หยุดการหมุนของมอเตอร์ด้านขวา
              PULSOUT  14 , 2500   ‘ ส่งพัลส์ 2 mS ให้มอเตอร์ด้านซ้ายหมุนทวนเข็มนาฬิกา (CCW)
              PAUSE    20          ‘ หน่วงเวลา 20 mS
              GOTO     TurnRight   ‘ วนลูปทำให้หุ่นยนต์หมุนวนทางด้านขวา
```



รูปที่ 4.5 ลักษณะการเลี้ยวขวาด้วยการหมุนเพียงล้อเดียวตามตัวอย่างที่ 4.5

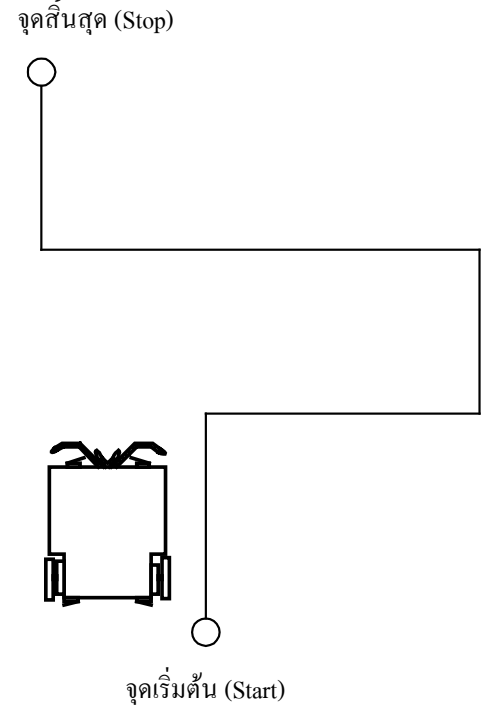
ตัวอย่างที่ 4.6 การเลี้ยวแบบล้อหมุนทั้งสองด้าน

```
TurnRight:    PULSOUT  14 , 2500   ‘ส่งพัลส์ 2 mS ให้มอเตอร์ด้านซ้ายหมุนทวนเข็มนาฬิกา (CCW)
              PULSOUT  13 , 2500   ‘ส่งพัลส์ 2 mS ให้มอเตอร์ด้านขวาหมุนทวนเข็มนาฬิกา (CCW)
              PAUSE    20          ‘ หน่วงเวลา 20 mS
              GOTO     TurnRight   ‘ วนลูปทำให้หุ่นยนต์หมุนวนทางด้านขวา
```



รูปที่ 4.6 ลักษณะการเลี้ยวขวาแบบหมุนสองล้อตามตัวอย่างที่ 4.6

เมื่อได้เรียนรู้การควบคุมหุ่นยนต์ในลักษณะต่างๆ แล้วเราจะยกตัวอย่างการควบคุมให้หุ่นยนต์เคลื่อนที่ไปในแนวหรือ เส้นทางที่ต้องการอย่างง่ายๆ โดยไม่มีการใช้เซนเซอร์ใดๆ ตามเส้นทางในรูปที่ 4.7



รูปที่ 4.7 แสดงลักษณะเส้นทางที่จะให้หุ่นยนต์เคลื่อนที่

ตัวอย่างที่ 4.7 โปรแกรมการควบคุมหุ่นยนต์ให้เคลื่อนที่ไปตามเส้นทางตามรูปที่ 4.7

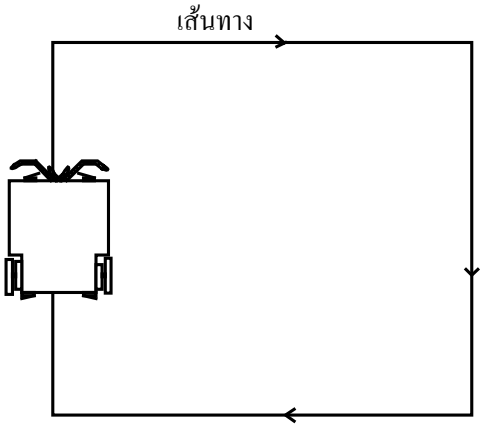
```
'{$STAMP BS2p}
LoopLeft        VAR     WORD
LoopRight       VAR     WORD
LoopDelay       VAR     WORD

'########## โปรแกรมหลัก ##########
GOSUB MotorForward              ' Robot เดินไปข้างหน้า
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOSUB MotorReturnRight90        ' Robot เลี้ยวขวา
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOSUB MotorReturnLeft90         ' Robot เลี้ยวซ้าย
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOSUB MotorReturnLeft90         ' Robot เลี้ยวซ้าย
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOSUB MotorReturnRight90        ' Robot เลี้ยวขวา
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOSUB MotorForward              ' Robot เดินไปข้างหน้า
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOSUB MotorStop                 ' หยุด

Delay:                          ' เดินหน้าไประยะหนึ่ง
FOR LoopDelay = 1 TO 50
 PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                       ' สร้าง Pulse Off = 20 mS
NEXT
RETURN

MotorForward:
PULSOUT 13,1250                 ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                        ' สร้าง Pulse Off = 20 mS
RETURN
```

```
MotorStop:
LOW 13                  ' เมื่อ Motor หยุดต้องป้อน Low ทั้งคู่ไม่เช่นนั้นเมื่อทำเงื่อนไขนี้แล้วจะทำให้โปรแกรมรวน
LOW 14
END

MotorReturnLeft90:
FOR LoopLeft = 1 TO 40  ' รถเลี้ยวซ้ายมุม 90 องศา
 PULSOUT 13,1250        ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,1250        ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20               ' สร้าง Pulse Off = 20 mS
NEXT
RETURN

MotorReturnRight90:
FOR LoopRight = 1 TO 40 ' เลี้ยวขวามุม 90 องศา
 PULSOUT 13,2500        ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,2500        ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
```

คำอธิบายโปรแกรม

จากตัวอย่างที่ 4.7 เป็นโปรแกรมตัวอย่างการควบคุมให้ Robot มีทิศทางการเคลื่อนที่ในรูปแบบที่เราต้องการโดยในที่นี้จะควบคุมให้หุ่นยนต์เดิน เป็นรูปแบบตามรูปที่ 4.7 ในโปรแกรมจะประกอบไปด้วย 2 ส่วนหลักๆ ก็คือ ส่วนของโปรแกรมหลัก และ โปรแกรมย่อย ซึ่งในโปรแกรมย่อยจะประกอบไปด้วยโปรแกรมที่ควบคุมให้หุ่นยนต์มีทิศทางการเคลื่อนที่ในรูปแบบต่างๆ เช่น เดินหน้า , ถอยหลัง , เลี้ยวซ้าย และ เลี้ยวขวา เป็นต้น ส่วนโปรแกรมหลักก็คือ รูปแบบการเดินของหุ่นยนต์ที่เราต้องการโดยอาศัยการเรียกใช้โปรแกรมย่อยต่างๆ ที่ผ่านมา ซึ่งจากตัวอย่างจะมีลำดับดังนี้ เดินหน้า → เลี้ยวขวา→ เลี้ยวซ้าย→เลี้ยวซ้าย→เลี้ยวขวา→เดินหน้า→หยุด ซึ่งจะเห็นว่าในตัวอย่างเมื่อมีการเรียกใช้โปรแกรมย่อยต่างๆ เหล่านี้แล้วจะมีการเรียกใช้โปรแกรมย่อยที่ชื่อ Delay โปรแกรมในส่วนนี้มีหน้าที่ควบคุมให้หุ่นยนต์เดินหน้าเป็นระยะหนึ่ง ซึ่งจะไปไกลเท่าใดนั้นขึ้นอยู่กับค่าจำนวนลูปของการทำซ้ำ (LoopDelay) ในที่นี้คือ 50 ลูป (1 to 50) หากเราต้องการเปลี่ยนแปลงระยะทางในแต่ละช่วงให้เปลี่ยนค่าของจำนวนลูปดังกล่าวนี้

ตัวอย่างที่ 4.8 การควบคุมหุ่นยนต์ให้เคลื่อนที่เป็นรูปสี่เหลี่ยม



รูปที่ 4.8 แสดงลักษณะรูปแบบการเดินเป็นเส้นทางสี่เหลี่ยม

```
'{$STAMP BS2p}
' โปรแกรมควบคุม Robot ให้มีการเคลื่อนที่เป็นรูปสี่เหลี่ยมตามรูปที่ 4.8

LoopRight   VAR         WORD
LoopDelay   VAR         WORD
'########## โปรแกรมหลัก ##########
Wait_start:
        if  IN2 = 1 then  Wait_start      ' รอการกดปุ่ม Start  (สวิตช์ที่ต่ออยู่กับ P2 บนบอร์ด ET-ROBOT)
Loop:
GOSUB MotorForward              ' Robot เดินไปข้างหน้า
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOSUB MotorReturnRight90        ' Robot เลี้ยวขวา 90 องศา
GOSUB Delay                     ' เดินหน้าไประยะหนึ่ง
GOTO Loop
'########## Delay##########
Delay:
FOR LoopDelay = 1 TO 50
 PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                       ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
```

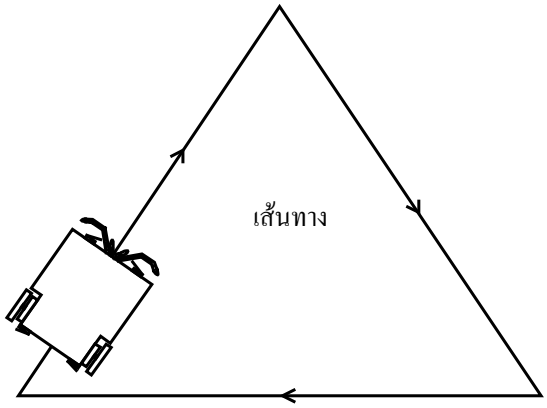
```
'########## MotorForward ##########
MotorForward:
PULSOUT 13,1250                 ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                        ' สร้าง Pulse Off = 20 mS
RETURN
'########## MotorReturnRight90 ##########
MotorReturnRight90:             ' เดินตรงไปข้างหน้าแล้วเลี้ยวขวามุม 90 องศา
FOR LoopRight = 1 TO 20
 PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                        ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopRight = 1 TO 40         ' เลี้ยวขวามุม 90 องศา
 PULSOUT 13,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                        ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
```

คำอธิบายโปรแกรม

จากโปรแกรมตัวอย่างที่ 4.8 เป็นการควบคุมหุ่นยนต์ให้เคลื่อนที่ในรูปแบบที่เราต้องการเช่นเดียวกันกับตัวอย่างที่ผ่านมา เพียงแต่ในตัวอย่างนี้จะให้ Robot เคลื่อนที่เป็นรูป สี่เหลี่ยมตามรูปที่ 4.8 โดยจะมีลำดับการเคลื่อนที่ดังนี้ เดินหน้า→เดินหน้าไปอีกระยะ→เลี้ยวขวา 90 องศา→เดินหน้าไปอีกระยะ แล้ววนลูป ทำให้หุ่นยนต์มีการเคลื่อนที่เป็นเส้นทางรูปสี่เหลี่ยมตลอดเวลา

ในโปรแกรมนี้ได้เพิ่มเงื่อนไขการเริ่มต้นการทำงานของหุ่นยนต์ด้วยการเช็คสภาวะการกดสวิตช์ที่ต่ออยู่กับขา P2 ของ CPU Stamp ซึ่งในที่นี้เราจะกำหนดให้สวิตช์นี้เป็นสวิตช์ Start เพื่อเริ่มการทำงานของหุ่นยนต์ เมื่อ Run โปรแกรมหุ่นยนต์จะยังไม่ทำงานทันทีจนกว่าจะมีการกดสวิตช์ดังกล่าวนี้ ทั้งนี้เพราะในบางครั้งเมื่อเราสั่ง Run จากคอมพิวเตอร์แล้ว เราอาจจะยังไม่อยากให้หุ่นยนต์เดินในทันที จนกว่าเราจะพร้อมเสียก่อน ซึ่งก็อาจนำเอาวิธีนี้ไปใช้ประโยชน์ได้ สภาวะของสวิตช์เมื่อถูกกดจะเป็น Logic “0” และ ในสภาวะปกติไม่มีการกดจะเป็น Logic “1”

ตัวอย่างที่ 4.9 การควบคุมหุ่นยนต์ให้เคลื่อนที่เป็นรูปสามเหลี่ยม



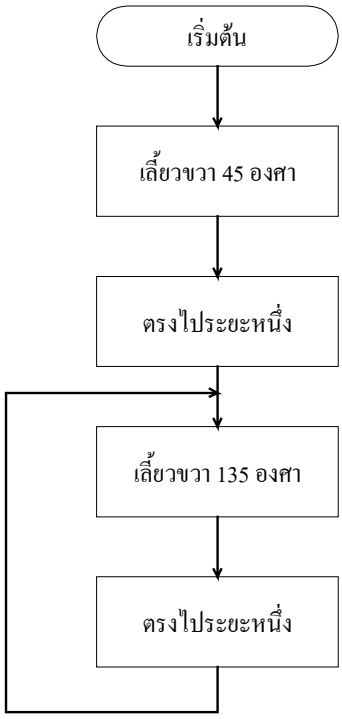
รูปที่ 4.9 ลักษณะรูปแบบการเดินเป็นเส้นทางรูปสามเหลี่ยม

```
'{$STAMP BS2p}
LoopRight   VAR          WORD
LoopDelay   VAR          WORD
'########## โปรแกรมหลัก ##########
GOSUB MotorReturnRight45
GOSUB Delay
Loop:
GOSUB  MotorReturnRight135
GOSUB  Delay
GOTO    Loop
'########## Delay ##########
Delay:
FOR LoopDelay = 1 TO 50
 PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                       ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
'########## MotorForward ##########
MotorForward:
PULSOUT 13,1250                 ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                        ' สร้าง Pulse Off = 20 mS
RETURN
```

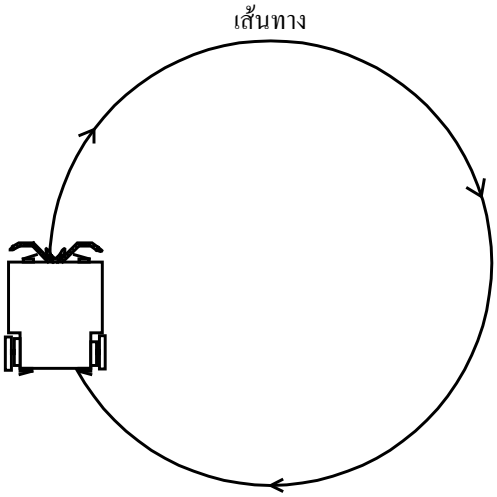
```
'########## MotorReturnRight45 ##########
MotorReturnRight45:
FOR LoopRight = 1 TO 20              ' เลี้ยวขวาประมาณ 45 องศา
PULSOUT 13,2500                      ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PULSOUT 14,2500                      ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                            ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
'########## MotorReturnRight135 ##########
MotorReturnRight135:
FOR LoopRight = 1 TO 50              ' เลี้ยวขวามุม 135 องศา
 PULSOUT 13,2500                     ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,2500                     ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                             ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
```

คำอธิบายโปรแกรม

จากโปรแกรมตัวอย่างที่ 4.9 เป็นตัวอย่างการควบคุมการเคลื่อนที่ของหุ่นยนต์ให้มีรูปแบบทิศทางตามรูปที่ 4.9 ซึ่งเป็นรูปสามเหลี่ยมมุม 45 องศา โดยมีลำดับการเคลื่อนที่ดังนี้



ตัวอย่างที่ 4.10 การควบคุมหุ่นยนต์ให้เคลื่อนที่เป็นวงกลม



รูปที่ 4.10 ลักษณะรูปแบบการเดินเป็นเส้นทางรูปวงกลม

```
'{$STAMP BS2p}
LoopRight   VAR         WORD
LoopDelay   VAR         WORD

'########## โปรแกรมหลัก ##########
Wait_start:
            if   IN2 = 1 then  Wait_start
Loop:
GOSUB MotorReturnRight          ' เลี้ยวขวา 45 องศา
GOSUB Delay                     ' เดินหน้าระยะทางสั้นๆ
GOTO Loop                       ' วนรูป

'########## Delay ##########
Delay:
FOR LoopDelay = 1 TO 5
 PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                       ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
```

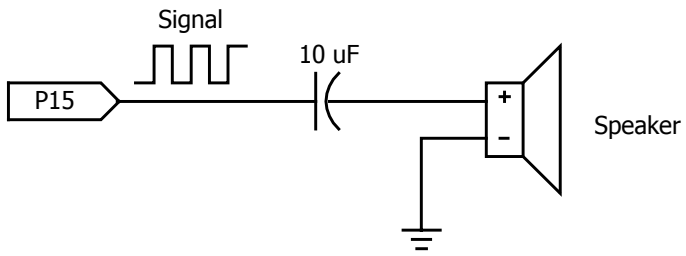
```
'########## MotorReturnRight ##########
MotorReturnRight:
FOR LoopRight = 1 TO 5              ' เลี้ยวขวาประมาณ 45 องศา
PULSOUT 13,2500                     ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PULSOUT 14,2500                     ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                           ' สร้าง Pulse Off = 20 mS
NEXT
RETURN
```

คำอธิบายโปรแกรม

จากโปรแกรมตัวอย่างที่ 4.10 เป็นการควบคุมการเคลื่อนที่ของหุ่นยนต์ให้เป็นเส้นทางวงกลมตามรูปที่ 4.10 โดยลักษณะการเคลื่อนที่ เนื่องจากหุ่นยนต์มีข้อจำกัดในการเลี้ยว ดังนั้นการเลี้ยวเป็นวงกลมจึงอาจจะดูไม่เป็นเส้นโค้งนัก ซึ่งจะใช้วิธีการเลี้ยวครั้งละ 45 องศาแล้วเดินหน้าไปเล็กน้อย วนลูปลักษณะนี้จนได้การเคลื่อนที่เป็นวงกลม

นอกจากนี้ยังได้ใส่เงื่อนไขการ Start หรือ การเริ่มการทำงานของหุ่นยนต์ด้วยการกดสวิตช์ P2 เข้าไปเหมือนกับในตัวอย่างที่ 4.8 อีกด้วย ดังนั้นเมื่อเปิดสวิตช์ ON แล้ว หุ่นยนต์จะยังไม่ทำการเคลื่อนที่ในทันที โดยจะต้องรอการกดสวิตช์ P2 เพื่อเริ่ม Start การทำงานหุ่นยนต์

ตัวอย่างที่ 4.11 การสร้างเสียง Beep ของลำโพง



รูปที่ 4.11 แสดงวงจรการต่อลำโพงของบอร์ด ET-ROBOT STAMP P40

ในการกำเนิดเสียงของลำโพง จะใช้หลักการในการสร้างพัลส์เป็นความถี่ป้อนให้กับสำโพง โดยจะต้องเป็นความถี่ที่อยู่ในย่านที่ลำโพงสามารถตอบสนองได้ ซึ่งการกำเนิดเป็นเสียงต่างๆ นั้นจะขึ้นอยู่กับขนาดของความถี่นั้นๆ ดังตัวอย่างต่อไปนี้ จะเป็นการสร้างเสียง บี๊บ (Beep) ติดและดับสลับกันตามโปรแกรมต่อไปนี้

```
'{$STAMP BS2p}
Loop:       FREQOUT 15,1000,1000   ‘กำเนิดความถี่ 3.77 KHz ไปที่ขา P15 เป็นเวลา 256 ms
            PAUSE 200              ‘ หน่วงเวลาเพื่อหยุดกำเนิดเสียงเป็นเวลา 200 ms
            GOTO Loop
```

คำอธิบายโปรแกรม

จากโปรแกรมตัวอย่างที่ 4.11 เป็นการสร้างความถี่ 3.77KHz ไปยังขาสัญญาณ P15 ซึ่งต่ออยู่กับลำโพงขนาดเล็กบนบอร์ด ET-ROBOT Stamp โดยใช้คำสั่ง FREQOUT เป็นเวลา 256 ms หลังจากนั้นจะทำการหน่วงเวลาไว้ 200 ms ในระหว่างการหน่วงเวลานี้ลำโพงจะหยุดกำเนิดเสียง แล้วทำการวนลูปไปกำเนิดเสียงใหม่ ผลที่ได้ก็คือ ลำโพงจะดัง และ หยุด สลับกันเป็นช่วงๆ ตลอดเวลาโดยหนึ่งหน่วยของค่าความถี่จะมีค่าเป็น 3.77 Hz และ หนึ่งหน่วยของค่า Period จะเป็น 0.256 ms จากตัวอย่างจะได้ความถี่เป็น 1000x 3.77 = 3770 Hz หรือ 3.77 KHz และ จะได้ค่าเวลา Period เป็น 0.256 ms x 1000 = 256 ms ซึ่งรายละเอียดของคำสั่ง FREQOUT สามารถดูได้จากภาคผนวก

ตัวอย่างที่ 4.12 การสร้างเป็นเสียงเพลง

จากตัวอย่างที่ 4.11 เป็นการกำเนิดเสียงให้กับลำโพงแบบง่ายๆ โดยต่อไปในตัวอย่างนี้ เราจะทำการกำเนิดเสียงเป็นรูปแบบของเสียงเพลง ดังตัวอย่างโปรแกรมต่อไปนี้

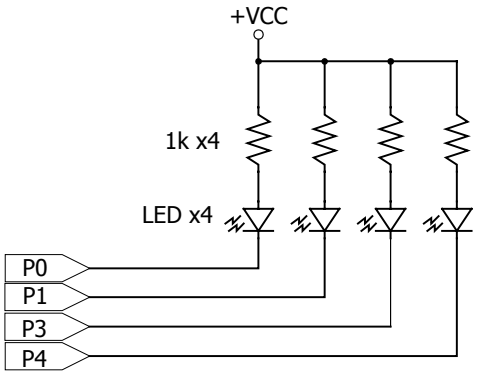
```
'{$STAMP BS2p}
I       VAR     BYTE        ‘ ตัวแปรสำหรับทำการชี้ค่าเพื่อเปิดตารางโน้ตดนตรี
F       VAR     WORD        ‘ ตัวแปรแบบเวิร์ดสำหรับเก็บค่าความถี่ที่ได้จากการเปิดตารางข้อมูล
C       CON     867         ‘ ตัวแปร C เก็บค่าคงที่ 867 สำหรับกำเนิดเสียงโน้ตดนตรี
D       CON     973         ‘ ตัวแปร D เก็บค่าคงที่ 973 สำหรับกำเนิดเสียงโน้ตดนตรี
E       CON     1092        ‘ ตัวแปร E เก็บค่าคงที่ 1092 สำหรับกำเนิดเสียงโน้ตดนตรี
G       CON     1300        ‘ ตัวแปร G เก็บค่าคงที่ 1300 สำหรับกำเนิดเสียงโน้ตดนตรี
R       CON     0           ‘ ตัวแปร R เก็บค่าคงที่ 0 ไม่มีการกำเนิดเสียง
Loop:
FOR I=0 TO 28                       ‘ เล่นเพลงจาก 29 ความถี่ในตารางข้อมูล
 LOOKUP I,[E,D,C,D,E,E,E,R,D,D,D,R,E,G,G,R,E,D,C,D,E,E,E,E,D,D,E,D,C],F   ‘เปิดตารางความถี่
 FREQOUT 15,1000,F,(F-2) MAX 16384   ‘ส่งสัญญานความถี่ออกที่ขา P15 ตามค่าจากการเปิดตาราง
NEXT
GOTO Loop
```

คำอธิบายโปรแกรม

จากตัวอย่างโปรแกรมที่ 4.12 เป็นโปรแกรมกำเนิดเสียงเพลงแบบง่ายๆ โดยอาศัยการส่งความถี่ขนาดต่างๆ กันไปที่ลำโพง ซึ่งผลของความถี่ที่แตกต่างกันนั้นจะทำให้เราได้โทนเสียงของลำโพงที่ต่างกัน และ เมื่อนำมาเรียงต่อกันในลำดับที่เหมาะสมเราก็จะได้เสียงเพลงออกมา

ในเบื้องต้นเราจะกำหนดตัวแปรและค่าคงที่ขนาดต่างๆ คือ C,D,E,G และ R ซึ่งแต่ละตัวจะเก็บค่าคงที่ไม่เท่ากัน ซึ่งอาจมองตัวแปรต่างๆ เหล่านี้เป็นตัวโน้ตที่ให้เสียงต่างๆ ก็ได้ ตัวแปรต่างๆ เหล่านี้จะถูกนำไปเรียงต่อกันในตารางตามรูปแบบของเสียงเพลงที่เราจะสร้างขึ้น จากนั้นจะทำการเปิดตารางเพื่อดึงเอาตัวแปรที่เก็บค่าคงที่แต่ละตัวออกมาโดยจะเก็บที่ตัวแปร F แล้วใช้คำสั่ง FREQOUT สร้างความถี่ตามขนาดของค่าคงที่ที่ได้จากการเปิดตารางนี้ ออกไปที่ลำโพงทำให้เกิดเสียงออกมา

ตัวอย่างที่ 4.13 การควบคุมการติดดับของ LED



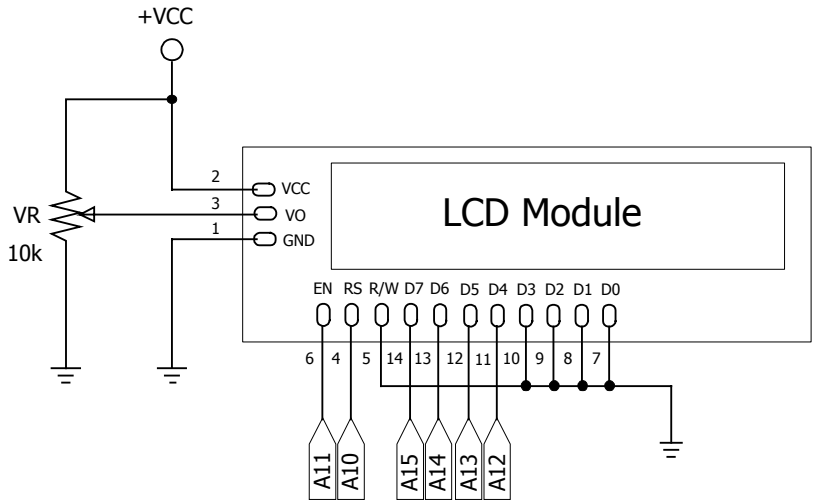
รูปที่ 4.12 แสดงวงจร LED ที่ต่ออยู่กับขาของ CPU Stamp

```
'{$STAMP BS2p}
Loop:
LOW 0           ‘ ส่ง Logic 0 ไปที่ขา P0 ทำให้ LED ติด
LOW 1           ‘ ส่ง Logic 0 ไปที่ขา P1 ทำให้ LED ติด
LOW 3           ‘ ส่ง Logic 0 ไปที่ขา P3 ทำให้ LED ติด
LOW 4           ‘ ส่ง Logic 0 ไปที่ขา P4 ทำให้ LED ติด
PAUSE 100       ‘ หน่วงเวลา
HIGH 0          ‘ ส่ง Logic 1 ไปที่ขา P0 ทำให้ LED ดับ
HIGH 1          ‘ ส่ง Logic 1 ไปที่ขา P1 ทำให้ LED ดับ
HIGH 3          ‘ ส่ง Logic 1 ไปที่ขา P3 ทำให้ LED ดับ
HIGH 4          ‘ ส่ง Logic 1 ไปที่ขา P4 ทำให้ LED ดับ
PAUSE 100       ‘ หน่วงเวลา
GOTO Loop       ‘ วนลูป
```

คำอธิบายโปรแกรม

LED ที่ว่านี้จะอยู่ที่ด้านหน้าของบอร์ด ET - ROBOT STAMP มีทั้งหมด 4 ดวงโดยจะต่อกับขาสัญญาณ P0,P1,P3 และ P4 ของ CPU Stamp ดังวงจรรูปที่ 4.12 การควบคุมให้ LED ติดสว่างทำได้โดยการส่ง Logic “0” ออกไปที่ขาสัญญาณดังกล่าว และ เช่นกัน หากต้องจากให้ LED ดับจะต้องส่ง Logic “1” โดยตัวอย่างต่อไปนี้ จะควบคุมให้ LED ทั้ง 4 ตัวติดและดับสลับกัน เป็นรูปแบบของไฟกระพริบ

ตัวอย่างที่ 4.14 การใช้งาน LCD กับบอร์ด ET-ROBOT



รูปที่ 4.13 แสดงวงจรการต่อ LCD กับขาสัญญาณของ Basic Stamp (ET-CLCD)

```
' ตัวอย่างการส่งตัวอักษรออกจอ LCD แบบพื้นฐาน ใช้กับ LCD แบบ  16 ตัวอักษร 2 บรรทัด
'{$STAMP BS2p}
 ASCII       VAR   BYTE

'********** โปรแกรมหลัก **********
 AUXIO                          ‘ เลือกใช้งาน I/O สำรอง (Auxiliary I/O)
 PAUSE   1000                   ‘ หน่วงเวลาเพื่อรอให้ LCD พร้อมทำงาน
 GOSUB   InitialLCD             ‘ กำหนดค่าเริ่มต้นการทำงานของ LCD
 ASCII =   "T"                  ‘ ข้อมูลรหัสแอสกีตัวอักษร T
 GOSUB   SendASCII              ‘ ส่งข้อมูลตัว T ออกไปแสดงผลที่จอ LCD
 ASCII = "e"                    ‘ ข้อมูลรหัสแอสกีตัวอักษร e
 GOSUB SendASCII                ‘ ส่งข้อมูลตัว e  ออกไปแสดงผลที่จอ LCD
```

```
 ASCII = "s"                    ‘ ข้อมูลรหัสแอสกีตัวอักษร  s
 GOSUB SendASCII                ‘ ส่งข้อมูลตัว s  ออกไปแสดงผลที่จอ LCD
ASCII = "t"                     ‘ ข้อมูลรหัสแอสกีตัวอักษร t
 GOSUB SendASCII                ‘ ส่งข้อมูลตัว t ออกไปแสดงผลที่จอ LCD
ASCII = " "                     ‘ ข้อมูลรหัสแอสกีช่องว่าง
 GOSUB SendASCII                ‘ ส่งข้อมูลตัวแอสกีของช่องว่าง ออกไปแสดงผลที่จอ LCD
 ASCII = "L"                    ‘ ข้อมูลรหัสแอสกีตัวอักษร  L
 GOSUB SendASCII                ‘ ส่งข้อมูลตัว L ออกไปแสดงผลที่จอ LCD
ASCII = "C"                     ‘ ข้อมูลรหัสแอสกีตัวอักษร C
 GOSUB SendASCII                ‘ ส่งข้อมูลตัว C ออกไปแสดงผลที่จอ LCD
 ASCII = "D"                    ‘ ข้อมูลรหัสแอสกีตัวอักษร D
 GOSUB SendASCII                ‘ ส่งข้อมูลตัว D ออกไปแสดงผลที่จอ LCD
 STOP
'********** โปรแกรม InitialLCD **********
InitialLCD:
 ASCII = $33                    ‘ 00110011 =  8 bit mode
 GOSUB SendCommandLCD           ‘ ส่งข้อมูลควบคุมให้ LCD
 ASCII = $32                    ‘ 00110010 = 8 bit mode
 GOSUB SendCommandLCD           ‘ ส่งข้อมูลควบคุมให้ LCD
 ASCII = $28                    ‘ 00101000 = 4 bit mode , 2 บรรทัด , 5x7 จุด
 GOSUB SendCommandLCD           ‘ ส่งข้อมูลควบคุมให้ LCD
 ASCII = $C                     ‘ เปิดจอแสดงผล, ไม่แสดงเคอร์เซอร์
 GOSUB SendCommandLCD           ‘ ส่งข้อมูลควบคุมให้ LCD
 ASCII = $6                     ‘ เลื่อนตำแหน่งเคอร์เซอร์ขึ้นเมื่อมีการเขียนข้อมูลบนจอ LCD
 GOSUB SendCommandLCD           ‘ ส่งข้อมูลควบคุมให้ LCD
 ASCII = $1                     ‘ เคลียร์หน้าจอ LCD
 GOSUB SendCommandLCD           ‘ ส่งข้อมูลควบคุมให้ LCD
 RETURN

'********** โปรแกรม SendASCII และ SendCommandLCD **********
SendASCII:
        HIGH 10  : GOTO  NextSend    ‘ บิต RS = 1 เลือกเป็นการส่งข้อมูลเพื่อแสดงผล : โดดไปทำงานที่ NextSend
SendCommandLCD:
        LOW 10                       ‘ บิต RS = 0 เลือกเป็นการส่งข้อมูลคำสั่ง
```

```
‘ *******  ส่งข้อมูล 4 บิตบน  *******
NextSend:
      HIGH 15 : IF ASCII.BIT7 = 1 THEN NextSend1 : LOW 15    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.7 ให้ LCD
NextSend1:
      HIGH 14 : IF ASCII.BIT6 = 1 THEN NextSend2 : LOW 14    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.6 ให้ LCD
NextSend2:
      HIGH 13 : IF ASCII.BIT5 = 1 THEN NextSend3 : LOW 13    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.5 ให้ LCD
NextSend3:
      HIGH 12 : IF ASCII.BIT4 = 1 THEN NextSend4 : LOW 12    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.4 ให้ LCD
NextSend4:
      PULSOUT 11,1 : PAUSE 1                                 ‘ ส่งพัลส์ Enable ให้ LCD

‘ ******** ส่งข้อมูล 4 บิตล่าง  *********
     HIGH 15 : IF ASCII.BIT3 = 1 THEN NextSend5 : LOW 15   ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.3 ให้ LCD
NextSend5:
     HIGH 14 : IF ASCII.BIT2 = 1 THEN NextSend6 : LOW 14    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.2 ให้ LCD
NextSend6:
     HIGH 13 : IF ASCII.BIT1 = 1 THEN NextSend7 : LOW 13    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.1 ให้ LCD
NextSend7:
     HIGH 12 : IF ASCII.BIT0 = 1 THEN NextSend8 : LOW 12    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.0 ให้ LCD
NextSend8:
          PULSOUT 11,1 : PAUSE 1                            ‘ ส่งพัลส์ Enable ให้ LCD
          RETURN
```

คำอธิบายโปรแกรม

จากตัวอย่างโปรแกรมที่ 4.14 เป็นโปรแกรมส่งข้อมูลออกไปแสดงผลที่จอ LCD เป็นรูปแบบการส่งแบบ 4 บิตข้อมูล คือ ใช้ขาสัญญาณข้อมูลเพียง 4 เส้น ในการส่งข้อมูล 1 Byte (8Bit) วิธีการก็คือ จะส่ง 4 บิตบน ออกไปก่อนจากนั้นจึงค่อยส่ง 4 บิตล่างตามไป ก็จะได้ข้อมูล 1 Byte พอดี ข้อมูลที่ส่งให้ LCD จะแบ่งออกเป็น 2 ส่วนคือ ข้อมูลคำสั่ง และ ข้อมูลแสดงผล โดยใช้บิต RS (P10) เป็นตัวแยกข้อมูลทั้ง 2 ส่วน โดยหากต้องการส่งข้อมูลคำสั่งต้องเรียกใช้โปรแกรมย่อย SendCommandLCD ส่วนข้อมูลแสดงผลจะใช้ SendASCII

จากโปรแกรมในส่วนแรกจะเป็นการกำหนดค่าการทำงานให้กับ LCD (Initial LCD) โดยจะต้องกำหนดเป็นแบบ 8 บิต ก่อนจากนั้นจึงเปลี่ยนมาเป็น 4 บิตในภายหลัง ซึ่งข้อมูลต่างๆ เหล่านี้จะเป็นข้อมูลคำสั่ง จากนั้นจะเป็นการส่งข้อมูลไปแสดงผลซึ่งข้อความคือ “Test LCD” และ เนื่องจากขาสัญญาณที่นำมาใช้ควบคุมจอ LCD เป็นขาสัญญาณ I/O สำรอง (Auxiliary I/O) ดังนั้นในส่วนแรกของโปรแกรมต้องใช้คำสั่ง AUXIO ด้วย

ตัวอย่างที่ 4.15 การส่งข้อความออก LCD แบบใช้การเปิดตารางข้อมูล (วงจรตามรูปที่ 4.13)

```
' ตัวอย่างการส่งข้อความออกแสดงผลที่จอ LCD โดยใช้วิธีการเปิดตาราง ใช้กับ LCD แบบ 16 ตัวอักษร 2 บรรทัด
'{$STAMP BS2p}
 ASCII          VAR   BYTE
 OrderASCII     VAR   BYTE

'********** โปรแกรมหลัก **********
 AUXIO                                                  ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O)
 PAUSE 1000                                             ‘ หน่วงเวลา 1000 ms เพื่อรอให้ LCD พร้อมทำงาน
 GOSUB InitialLCD                                       ‘ กำหนดค่าเริ่มต้นการทำงานของ LCD
FOR OrderASCII = 0 TO 10                                ‘ ลูปเปิดตารางข้อมูล 11 ตัวอักษร
    LOOKUP OrderASCII,["BASIC Stamp"],ASCII             ‘ เปิดตารางข้อมูลเก็บไว้ในตัวแปร ASCII
    GOSUB   SendASCII                                   ‘ ส่งข้อมูลออกไปแสดงผล LCD
NEXT                                                    ‘ OrderASCII เพิ่มขึ้นหนึ่ง และทำซ้ำลูป
 STOP                                                   ‘ สิ้นสุดการทำงาน
'********** โปรแกรม InitialLCD **********
InitialLCD:                                             ‘ กำหนดค่าการทำงานให้ LCD
       FOR OrderASCII = 0 TO 5                          ‘ เปิดตารางข้อมูลคำสั่ง 6 ค่า
         LOOKUP OrderASCII,[$33,$32,$28,$C,$6,$1],ASCII ‘ เปิดตารางเก็บค่าไว้ที่ตัวแปร ASCII
         GOSUB   SendCommandLCD                         ‘ ส่งข้อมูลคำสั่งให้ LCD
       NEXT                                             ‘ OrderASCII เพิ่มขึ้นหนึ่ง และทำซ้ำลูป
       RETURN
'********** โปรแกรม SendASCII และ SendCommandLCD **********
SendASCII:
      HIGH 10  : GOTO NextSend                          ‘ RS = 1  ส่งข้อมูลแสดงผล : โดดไปทำงานที่ NextSend
SendCommandLCD:
      LOW 10                                            ‘ RS = 0 ส่งข้อมูลคำสั่ง
NextSend:
   HIGH 15 : IF ASCII.BIT7 = 1   THEN  NextSend1 : LOW 15   ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.7 ให้ LCD
NextSend1:
   HIGH 14 : IF  ASCII.BIT6 = 1  THEN  NextSend2 : LOW 14   ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.6 ให้ LCD
NextSend2:
    HIGH 13 : IF ASCII.BIT5 = 1  THEN NextSend3 : LOW 13   ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.5 ให้ LCD
NextSend3:
    HIGH 12 : IF ASCII.BIT4 = 1  THEN NextSend4 : LOW 12   ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.4 ให้ LCD
```

```
NextSend4:
        PULSOUT 11,1 : PAUSE 1                          ‘ ส่งพัลส์ Enable LCD
    HIGH 15 : IF ASCII.BIT3 = 1  THEN NextSend5 : LOW 15    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.3 ให้ LCD
NextSend5:
    HIGH 14 : IF ASCII.BIT2 = 1  THEN NextSend6 : LOW 14    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.2 ให้ LCD
NextSend6:
    HIGH 13 : IF ASCII.BIT1 = 1  THEN NextSend7 : LOW 13    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.1 ให้ LCD
NextSend7:
    HIGH 12 : IF ASCII.BIT0 = 1  THEN NextSend8 : LOW 12    ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.0 ให้ LCD
NextSend8:
      PULSOUT 11,1 : PAUSE 1                            ‘ ส่งพัลส์ Enable LCD
      RETURN
```

คำอธิบายโปรแกรม

ตัวอย่างโปรแกรมที่ 4.15 นี้จะมีลักษณะคล้ายกับตัวอย่างโปรแกรมที่ 4.14 เพียงแต่ใช้การเปิดตารางข้อมูลแทนทำให้ การเขียนโปรแกรมสั้นลง และ มีความสะดวกยิ่งขึ้นส่วนรูปแบบของการส่งนั้นยังเหมือนเดิม โดยในตัวอย่างนี้จะแสดงข้อความ "BASIC Stamp" สามารถลองทดสอบเปลี่ยนเป็นข้อความต่างๆ ได้ตามต้องการ แต่จะต้องกำหนดจำนวน OrderASCII ให้สอดคล้องกับจำนวนข้อมูลที่อยู่ในตารางด้วย และ จากโปรแกรมจะเห็นว่ามีการเขียนคำสั่งหลายๆ คำสั่งไว้ในบรรทัดเดียวกัน เช่น

```
HIGH 12 : IF ASCII.BIT0 = 1  THEN NextSend8 : LOW 12
```

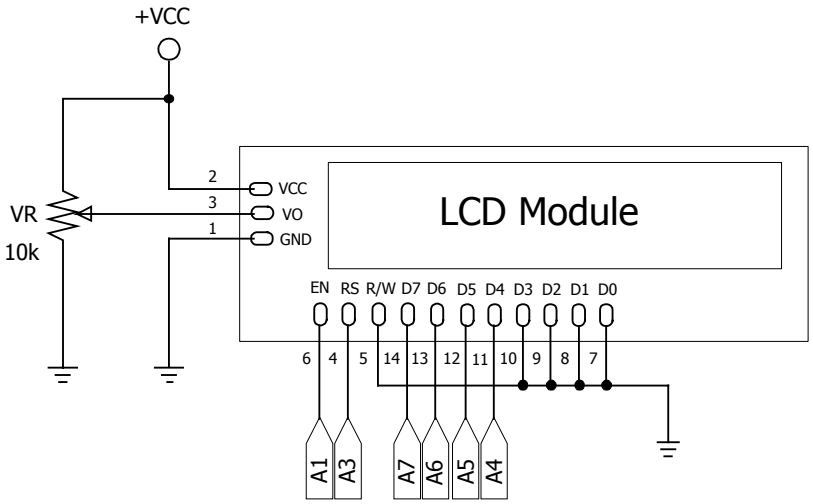
จะเหมือนกับ

```
HIGH 12
 IF ASCII.BIT0 = 1  THEN NextSend8
 LOW 12
```

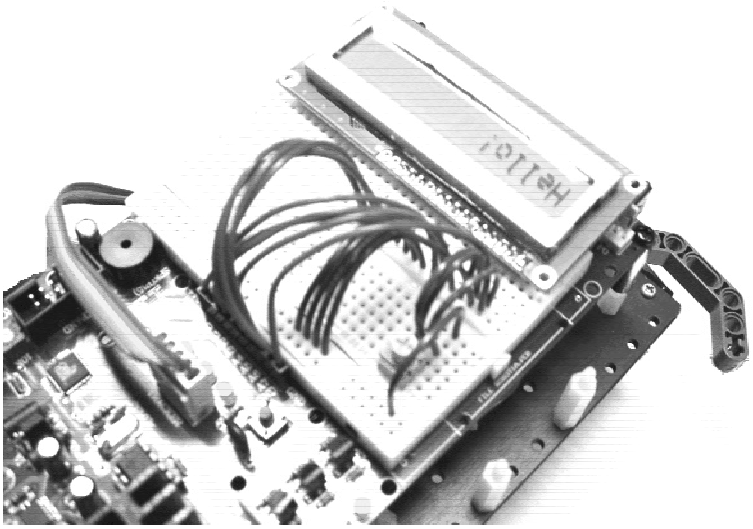
ซึ่งทั้งนี้ที่เขียนอยู่ในบรรทัดเดียวกันก็เพื่อความสะดวก โดยในการเขียนคำสั่งต่างๆไว้ในบรรทัดเดียวกันนั้นจะต้องทำการคั้น คำสั่งต่างๆ นั้นด้วยเครื่องหมาย Colon (:) แต่ก็ควรระวังเพราะอาจจะทำให้ไปซับสนกับตัวลาเบลได้เช่นกัน

ตัวอย่างที่ 4.16 การต่อ LCD กับขา auxiliary I/O เพื่อใช้คำสั่ง LCDOUT

เนื่องจากเราไม่ได้จัดวงจรในส่วนของ LCD ให้รองรับการใช้งานคำสั่ง LCDOUT ดังนั้นหากต้องการต่อใช้งาน LCD ที่ขั้วต่อ 14 PIN ET-CLCD จะต้องใช้วิธีการและวงจรตามตัวอย่างที่ 4.14 และ 4.15 แต่หากต้องการใช้คำสั่ง LCDOUT เพื่อควบคุมการแสดงผลของจอแอลซีดีสามารถทำได้ตามวงจร และ ตัวอย่างโปรแกรมต่อไปนี้ โดยการต่อขาสัญญาณสำรอง (Auxiliary I/O) เข้ากับขาสัญญาณต่างๆ ของ LCD ด้วยโปรเจ็คบอร์ดดังต่อไปนี้



รูปที่ 4.14 วงจรเชื่อมต่อ LCD เพื่อใช้กับคำสั่ง LCDOUT



รูปที่ 4.15 ลักษณะการเชื่อมต่อ LCD บนโปรเจ็คบอร์ด

```
‘ ตัวอย่างการใช้งาน LCD โดยใช้คำสั่ง LCDCMD และ LCDOUT ใช้กับ LCD แบบ 16 ตัวอักษร 2 บรรทัด
'{$STAMP BS2p}
        AUXIO           ' เลือกใช้งาน I/O สำรอง หรือ auxiliary I/O pins
InitLCD:
        PAUSE 1000      ' หน่วงเวลา 1 วินาทีเพื่อรอให้ LCD พร้อมทำงาน
        LCDCMD 1, 32    ' กำหนดโหมดการส่งข้อมูลเป็นแบบ 4-bit mode
        LCDCMD 1, 40    ' กำหนดชนิดของ LCD เป็นแบบ 2 บรรทัด 5x7 จุด
        LCDCMD 1, 8     ' ปิดจอแสดงผล LCD
        LCDCMD 1, 12    ' เปิดจอแสดงผล และ ไม่แสดงเคอร์เซอร์
        LCDCMD 1, 6     ' เพิ่มตำแหน่งเคอร์เซอร์ขึ้นเมื่อมีการเขียนข้อมูลแต่ไม่แสดงเคอร์เซอร์
        LCDCMD 1, 1     ' เคลียร์หน้าจอแสดงผล

        LCDOUT 1,1, ["Hello! ET-ROBOT"]   ‘ เคลียร์หน้าจอและส่งข้อความไปแสดงที่ LCD
        STOP
```

คำอธิบายโปรแกรม

จากตัวอย่างที่ผ่านๆ ในการเขียนโปรแกรมควบคุมการแสดงผลของจอ LCD จะใช้วิธีการควบคุมแบบพื้นฐานทางด้านฮาร์ดแวร์ของ LCD จริงๆ แต่ในความเป็นจริงเราสามารถทำได้อีกวิธีหนึ่งก็คือใช้คำสั่งสำเร็จรูปที่มีอยู่แล้วของ Basic Stamp คือ คำสั่ง LCDOUT ซึ่งจะง่ายกว่าวิธีที่ผ่านๆ มาซึ่งหากจะใช้วิธีการนี้จะต้องทำการต่อวงจรดังรูปที่ 4.13 ซึ่งจะใช้ขาสัญญาณ I/O แบบ Auxiliary หรือ เรียกอีกอย่างหนึ่งว่า I/O สำรองจะสามารถใช้ได้ก็ต่อเมื่อใช้คำสั่ง AUXIO ก่อนเท่านั้น ซึ่งในขณะที่เราเลือกใช้งาน I/O สำรองดังกล่าวนี้ เราจะไม่สามารถควบคุมการทำงานของ I/O หลักได้ (Main I/O) การกลับไปยัง I/O หลัก ทำได้โดยใช้คำสั่ง MAINIO และ ก็เช่นกันเมื่ออยู่ในฟังก์ชันของ I/O หลัก เราก็ไม่สามารถควบคุมการทำงานของ I/O สำรองได้ ดังนั้นหากมีการใช้งาน I/O ทั้งสองส่วนร่วมกัน คือ I/O หลัก (Main I/O) และ I/O สำรอง (Auxiliary I/O) จะต้องมีการสลับกันไปมาโดยใช้วิธีการต่างๆ ตามที่กล่าวมาซึ่งปกติแล้วจะ default ไว้ที่ I/O หลัก (Main I/O)

บทที่ 5
การควบคุมหุ่นยนต์แบบมีเงื่อนไข

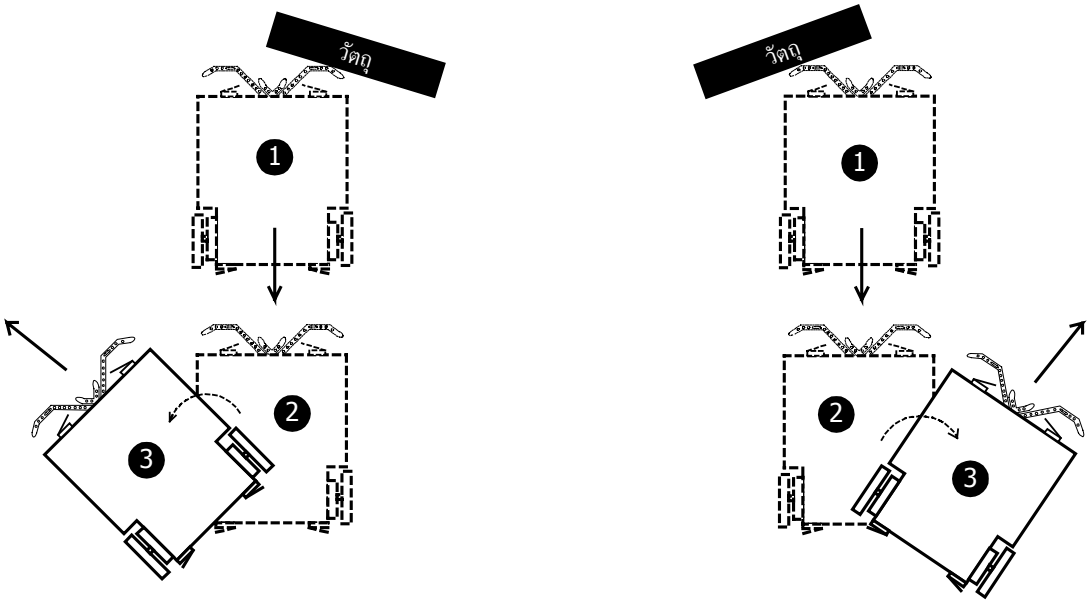
การควบคุมหุ่นยนต์แบบมีเงื่อนไขที่ว่านี้ หมายถึง การสั่งงานให้หุ่นยนต์ปฏิบัติงานต่างๆ พร้อมกับทำการตรวจสอบสภาวะของอุปกรณ์อินพุตเซนเซอร์ต่างๆ ด้วยว่ามีสภาวะเป็นอย่างไร จากนั้นจึงนำสภาวะต่างๆ ที่ได้มาตรวจสอบกับเงื่อนไขที่เราสร้างขึ้น เช่น หากสวิตช์กันชนหน้าทั้งสองจุดถูกกด ก็สั่งให้หุ่นยนต์ถอยหลังเป็นต้น

ซึ่งอุปกรณ์เซ็นเซอร์ต่างๆ นี้จะเปรียบเสมือนกับเป็นเส้นประสาทในการรับรู้สิ่งต่างๆ ของหุ่นยนต์ ยิ่งมีมากเท่าไรก็จะยิ่งทำให้หุ่นยนต์นั้นมีความสามารถมากขึ้นเท่านั้นส่วนความฉลาดของหุ่นยนต์นั้นก็จะขึ้นอยู่กับโปรแกรมที่เราสร้างให้กับหุ่นยนต์ว่าจะมีวิธีคิด หรือ จัดการกับสิ่งต่างๆที่เกิดขึ้นนี้อย่างไร ซึ่งอุปกรณ์เซ็นเซอร์ต่างๆ จะมีดังนี้

- สวิตช์เซ็นเซอร์กันชน หน้า-หลัง รวม 4 ตัว (Micro Switch Sensor)
- เซ็นเซอร์ตรวจจับแสง (Light Sensor)
- เซ็นเซอร์สำหรับตรวจจับเส้น หรือ แถบดำ (Track Sensor)

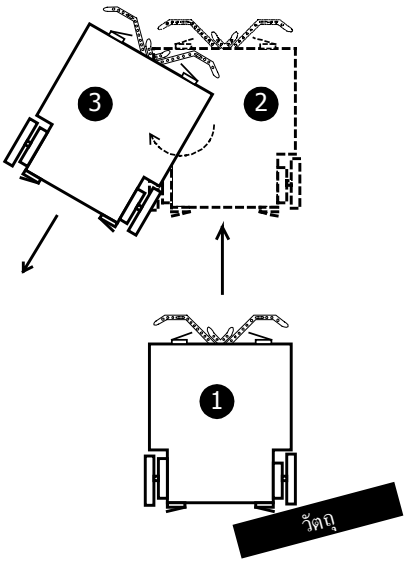
5.1 การควบคุมหุ่นยนต์ให้หลบหลีกสิ่งกีดขวาง

โดยในเบื้องต้นนี้จะยกตัวอย่างการเขียนโปรแกรมควบคุมการทำงานของหุ่นยนต์ให้มีการหลบหลีกสิ่งกีด-ขวาง โดยมีการเช็คเงื่อนไขจากสวิตช์กันชนหน้า-หลังทั้ง 4 ตัว ซึ่งตัวอย่างโปรแกรมนี้จะสั่งให้หุ่นยนต์เดินไปด้านหน้าพร้อมกับเช็คสถานะของสวิตช์กันชน ซึ่งหากเกิดการชนอาจจะเป็นวัตถุใดๆ ก็ตามหุ่นยนต์จะทำการถอยหลังแล้วหักหลบสิ่งกีดขวางนั้นและเดินหน้าต่อไป

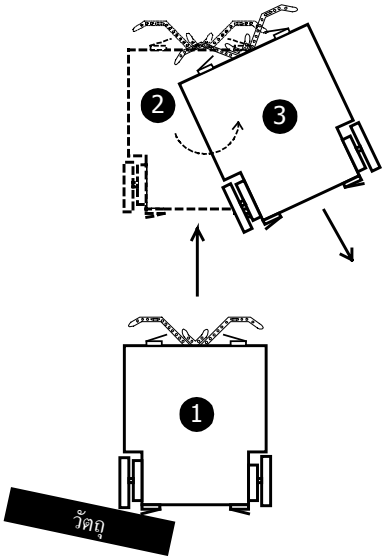


การชนจากด้านหน้าขวา การชนจากด้านหน้าซ้าย

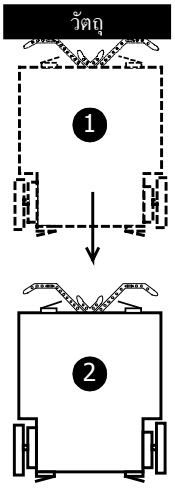
รูปที่ 5.1 ลักษณะการชนและการหลบหลีกวัตถุ



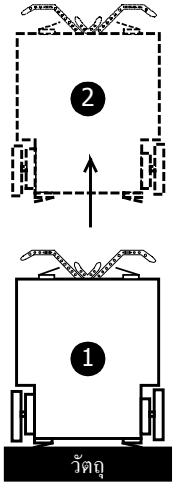
การชนจากด้านหลังขวา



การชนจากด้านหลังซ้าย



การชนจากด้านหน้าทั้งซ้ายและขวา



การชนจากด้านหลังทั้งซ้ายและขวา

รูปที่ 5.1 ลักษณะการชนและการหลบหลีกวัตถุ (ต่อ)

จากรูปที่ 5.1 เป็นรูปแบบการชนและการหลบหลีกวัตถุในรูปแบบต่างๆ โดยอาศัยสวิตช์กันชนหน้า-หลังทั้ง 4 ตัวเป็นตัวตรวจสอบการชนโดยองศาในการเลี้ยวหลบนั้นขึ้นอยู่ที่เราจะเขียนโปรแกรมอย่างไร และในส่วนของการชนด้านหน้าทั้งซ้ายและขวา เมื่อทำการถอยแล้วขั้นตอนต่อไปก็อยู่ที่เราว่าจะให้ทำอย่างไรเช่น เลี้ยวซ้าย-ขวา หรือ หมุนกลับหลังแล้วเดินหน้าต่อก็ได้ ขึ้นอยู่กับความต้องการของเรา การถอยไปชนวัตถุทั้งทางด้านซ้ายและขวาก็เช่นกันเมื่อเดินหน้าแล้วจะควบคุมให้หุ่นยนต์ทำอะไรต่อไปนั้นก็ขึ้นอยู่กับการออกแบบของเรา เพื่อให้หุ่นยนต์สามารถเคลื่อนที่ผ่านสิ่งกีดขวางนั้นไปได้ โดยสามารถสรุปตามเงื่อนไขต่างๆ เป็นตารางดังนี้

กรณีหุ่นยนต์เคลื่อนที่ไปด้านหน้า

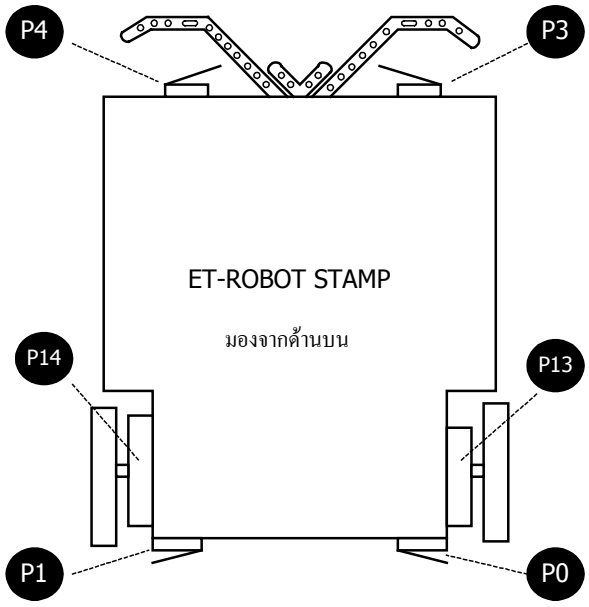
อินพุตเซ็นเซอร์สวิตช์ด้านหน้า		เอาต์พุตสำหรับความคุมมอเตอร์		สถานะการทำงานของหุ่นยนต์
ซ้าย (P4)	ขวา (P3)	ซ้าย (P14)	ขวา (P13)	
0	0	CCW	CW	ถอยหลัง
0	1	[*] CCW	[*] CCW	ถอยหลัง + เลี้ยวขวา(45º)
1	0	[*] CW	[*] CW	ถอยหลัง + เลี้ยวซ้าย(45º)
1	1	CW	CCW	เดินหน้า

* จะต้องทำการถอยหลังออกมาก่อนสักระยะหนึ่ง (P14 = CCW , P13 = CW) ก่อนที่จะทำการเลี้ยว

กรณีหุ่นยนต์เคลื่อนที่ไปด้านหลัง

อินพุตเซ็นเซอร์สวิตช์ด้านหลัง		เอาต์พุตสำหรับความคุมมอเตอร์		สถานะการทำงานของหุ่นยนต์
ซ้าย (P1)	ขวา (P0)	ซ้าย (P14)	ขวา (P13)	
0	0	CW	CCW	เดินหน้า
0	1	[*] CCW	[*] CCW	เดินหน้า + เลี้ยวขวา(45º)
1	0	[*] CW	[*] CW	เดินหน้า + เลี้ยวซ้าย(45º)
1	1	CW	CCW	ถอยหลัง

* จะต้องทำการเดินหน้าก่อนสักระยะหนึ่ง (P14 = CW , P13 = CCW) ก่อนที่จะทำการเลี้ยว



รูปที่ 5.2 แสดงตำแหน่งของการต่อพอร์ตต่างๆ กับเซนเซอร์และมอเตอร์

จากตารางการทำงานต่างๆ สามารถนำมาเขียนเป็นโปรแกรมดังตัวอย่างโปรแกรมต่อไปนี้

ตัวอย่างที่ 5.1 โปรแกรมควบคุมการทำงานของหุ่นยนต์ให้หลบลีกสิ่งกีดขวาง

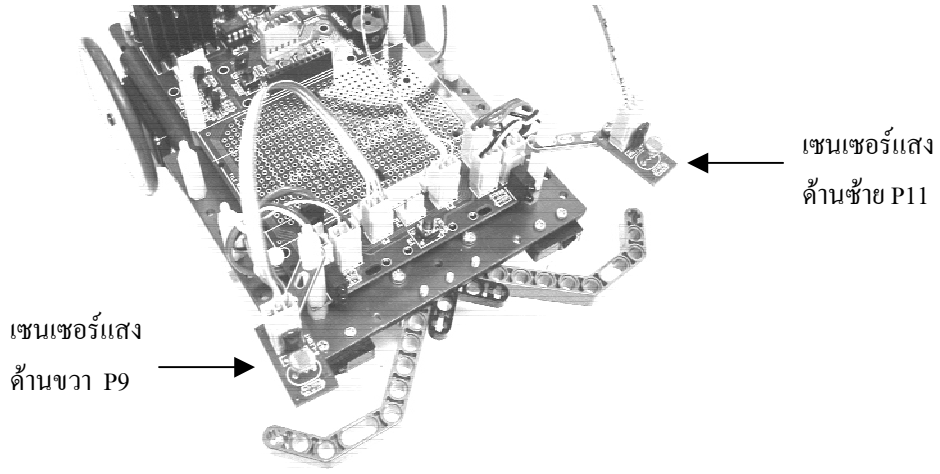
```
'{$STAMP BS2p}
LoopLeft      VAR      WORD      ' ตัวแปรชนิด WORD ใช้สำหรับกลุ่มโปรแกรมการเคลื่อนตัวไปทางซ้าย
LoopRight     VAR      WORD      ' ตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางขวา
LoopStraight  VAR      WORD      ' ตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวในแนวเส้นตรง

'########## โปรแกรมหลัก ##########
LoopMain:
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN MotorBackward
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorEvadeLeft
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN MotorEvadeRight
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 0 ) THEN MotorForward
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 1 ) THEN MotorForward
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 0 ) THEN MotorForward
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
GOTO LoopMain          ' วนตรวจสอบสถาวะ Input

 '########## MotorForward (เดินหน้า)  ##########
MotorForward:
PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                       ' สร้าง Pulse Off = 20 mS
GOTO LoopMain                  ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorBackward (ถอยหลัง)  ##########
MotorBackward:
FOR LoopStraight = 1 TO 50             ' ถอยหลังไประยะหนึ่ง
 PULSOUT 13,2500                       ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,1250                       ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PAUSE 20                               ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain                          ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

```
'########## MotorEvadeRight (ถ่อยแล้วเลี้ยวซ้าย 45 องศาเพื่อหลบสิ่งกีดขวาง) ##########
MotorEvadeRight:
FOR LoopRight = 1 TO 40          ' ถอยหลังไประยะหนึ่ง
 PULSOUT 13,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,1250                 ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                        ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopRight = 1 TO 20          ' เลี้ยวซ้ายประมาณ 45 องศา เพื่อหลบสิ่งกีดขวาง
PULSOUT 13,1250                  ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,1250                  ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PAUSE 20                         ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain                    ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
'########## MotorEvadeLeft (ถ่อยแล้วเลี้ยวขวา 45 องศาเพื่อหลบสิ่งกีดขวาง) ##########
MotorEvadeLeft:
FOR LoopLeft = 1 TO 40           ' ถอยหลังไประยะหนึ่ง
 PULSOUT 13,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,1250                 ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PAUSE 20                         ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopLeft = 1 TO 20           ' เลี้ยวขวาประมาณ 45 องศา เพื่อหลบสิ่งกีดขวาง
PULSOUT 13,2500                  ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PULSOUT 14,2500                  ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                        ' สร้าง Pulse Off = 20 mS
NEXT
GOTO    LoopMain                 ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

5.2 การควบคุมหุ่นยนต์ให้เดินตามแสง

การควบคุมหุ่นยนต์ให้สามารถเดินตามแสงได้นั้นปัจจัยสำคัญก็คือ ตัวเซนเซอร์ตรวจจับแสงซึ่งเปรียบไปแล้วก็เหมือนกับเป็นดวงตาของหุ่นยนต์ ที่ใช้ในการมองเห็นแสงเพราะฉนั้นก่อนอื่นเราต้องทราบหลักการทำงานของวงจรเซนเซอร์แสงเสียก่อนซึ่งสามารถศึกษารายละเอียดได้ในหัวข้อที่ 1.10 บทที่ 1

การติดตั้งตัวเซนเซอร์แสงนั้นจะต้องให้ตัวเซนเซอร์ทั้งสองมีระยะห่างกันพอสมควรเพื่อให้สามารถแยกความแตกต่างของแสงได้ดีขึ้น ในตัวอย่างรูปที่ 5.1 เราจะทำการติดตั้งไว้ด้านหน้าทางด้านซ้ายและขวา เพื่อตรวจจับระดับความเข้มแสงของทั้งสองด้าน



รูปที่ 5.1 การติดตั้งตัวเซ็นเซอร์แสง

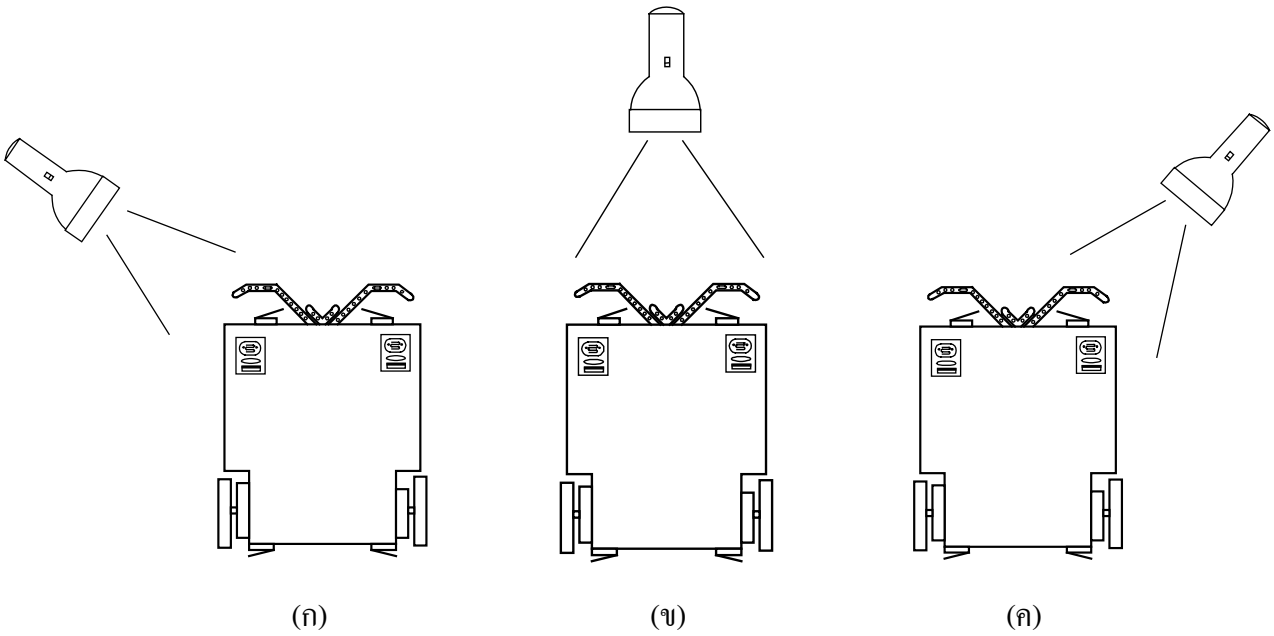
โดยหลักการของวงจร RC time ก็คือ เมื่อค่า R หรือ C มีการเปลี่ยนแปลงจะทำให้ค่าเวลาของวงจร RC เปลี่ยนแปลงด้วยแต่ในที่นี้ค่าที่สามารถเปลี่ยนแปลงได้คือค่า R (LDR) เท่านั้น และค่า R นี้ก็จะมีการเปลี่ยนแปลงไปตามความเข้มของแสง ดังนั้นในการตรวจสอบสภาวะของแสงเราจึงตรวจสอบได้จากค่าเวลาของวงจร RC โดยใช้คำสั่งในการอ่านค่าเวลาจากวงจร RC time ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 5.2 โปรแกรมวัดค่า RC time จากวงจรเซนเซอร์แสง

```
left_photo      var     word
right_photo     var     word

main:
        'วัดค่า RC time จากวงจรเซนเซอร์แสงด้านขวา
        High    9                       ' ส่งลอจิก "1" ไปที่ขา PIN 9
        Pause   3                       ' หน่วงเวลา 3 ms เพื่อให้ C เก็บประจุได้เต็มที่
        Rctime  9,1,right_photo         ' อ่านค่า RC time จากขา PIN 9
```

```
'วัดค่า RC time จากวงจรเซนเซอร์แสงด้านซ้าย
High    11                          ' ส่งลอจิก "1" ไปที่ขา PIN 11
Pause   3                           ' หน่วงเวลา 3 ms
Rctime  11,1,left_photo             ' อ่านค่า RC time จากขา PIN 11
Debug   home, "L ", dec5 left_photo, " R ",dec5 right_photo   ' ส่งค่าที่ได้ออกไปแสดงที่หน้าจอ
Goto    main
```

จากตัวอย่างโปรแกรมเป็นการวัดค่าเวลาจากวงจร RC time (ตัวเซนเซอร์แสง) ที่ติดตั้งอยู่ทางด้านซ้ายและขวาของหุ่นยนต์ (ดังรูปที่ 5.1) ซึ่งทางด้านขวาจะถูกต่อเข้ากับขา PIN 9 ส่วนด้านซ้ายจะต่อกับขา PIN 11 จากนั้นจะนำค่าที่อ่านได้ไปแสดงที่หน้าจอด้วยคำสั่ง Debug ซึ่งค่าที่อ่านได้นี้จะขึ้นอยู่กับขนาดความเข้มของแสงที่ตกกระทบตัว LDR ซึ่งเราสามารถใช้ค่าต่างๆ เหล่านี้มาเป็นเงื่อนไขในการควบคุมการทำงานของหุ่นย์ยนต์ให้มีการเคลื่อนที่ไปตามรูปแบบต่างๆ โดยมีลักษณะทิศทางการรับแสงดังรูปที่ 5.2



(ก) (ข) (ค)

รูปที่ 5.2 แสดงลักษณะการรับแสงในรูปแบบต่างๆ ของ Robot

จากรูปที่ 5.2(ก) ความเข้มของแสงทางด้านซ้ายจะมีขนาดมากกว่าด้านขวา ในกรณีนี้หากต้องการให้หุ่นยนต์เดินตามแสงจะต้องควบคุมให้หุ่นยนต์เดินไปทางซ้าย ส่วนรูปที่ 5.2(ข) ระดับความเข้มของแสงทั้งด้านซ้ายและ ขวาจะมีค่าใกล้เคียงกันกรณีนี้จะสั่งให้หุ่นยนต์เดินไปด้านหน้า และ ในรูปที่ 5.2(ค) เป็นการรับแสงจากทางด้านขวาทำให้ความเข้มแสงทางด้านขวามากกว่าด้านซ้าย ดังนั้นจะต้องควบคุมหุ่นยนต์ให้เคลื่อนที่ไปทางขวาเพื่อเข้าไปหาแหล่งที่มาของแสง ในกรณีที่แสงเข้ามาด้านหน้านั้นไม่ได้หมายความว่าความเข้มของแสงที่ตัวเซ็นเซอร์ทั้งซ้ายและขวาจะเท่ากันเสียทีเดียว เนื่องจากมีปัจจัยอื่นๆ อีกมากที่ไม่สามารถทำให้เกิดการเท่ากันของเซ็นเซอร์ทั้งสองด้านได้ดังนั้นในการเขียนโปรแกรมเราจะต้องทำการเพื่อค่าไว้ โดยหากค่าที่อ่านได้จากเซ็นเซอร์ทั้งสองมีค่าใกล้เคียงกันก็ถือได้ว่าแสงมาจากทิศทางตรงหน้า แต่หากมีความแตกต่างกันมากก็แสดงว่าแสงมาจากทางด้านซ้ายหรือขวาเป็นต้น ซึ่งจะยกตัวอย่างโปรแกรมควบคุมให้หุ่นยนต์เดินตามแสงดังนี้

ตัวอย่างที่ 5.3 โปรแกรมควบคุมหุ่นยนต์ให้เดินตามแสง

```
'{$STAMP BS2p}
' โปรแกรม Control Robot

ValueRCTimePIN9       VAR     WORD
ValueRCTimePIN11      VAR     WORD

'########## โปรแกรมหลัก ##########
LoopMain:
HIGH 9                              ' ขา PIN 9 มีสภาวะ high (“1”)
PAUSE 1                             ' หน่วงเวลาเพื่อให้ C เก็บประจุเต็มที่
RCTIME  9,1,ValueRCTimePIN9     ' อ่านค่า Rctime จากขา PIN 9 เก็บในตัวแปร ValueRCTimePIN9
HIGH 11                             ' ขา PIN 11 มีสภาวะ high (“1”)
PAUSE 1                             ' หน่วงเวลาเพื่อให้ C เก็บประจุเต็มที่
RCTIME 11,1,ValueRCTimePIN11    ' อ่านค่า Rctime จากขา PIN 11 เก็บในตัวแปร ValueRCTimePIN11

IF ( ValueRCTimePIN9 > 200 AND ValueRCTimePIN11 > 200 ) THEN MotorStop
IF ( ValueRCTimePIN9 > 200 AND ValueRCTimePIN11 < 200 ) THEN MotorReturnLeft
IF ( ValueRCTimePIN9 < 200 AND ValueRCTimePIN11 > 200 ) THEN MotorReturnRight
IF ( ValueRCTimePIN9 < 200 AND ValueRCTimePIN11 < 200 ) THEN MotorForward
GOTO LoopMain           ' วนตรวจสอบสภาวะ Input
```

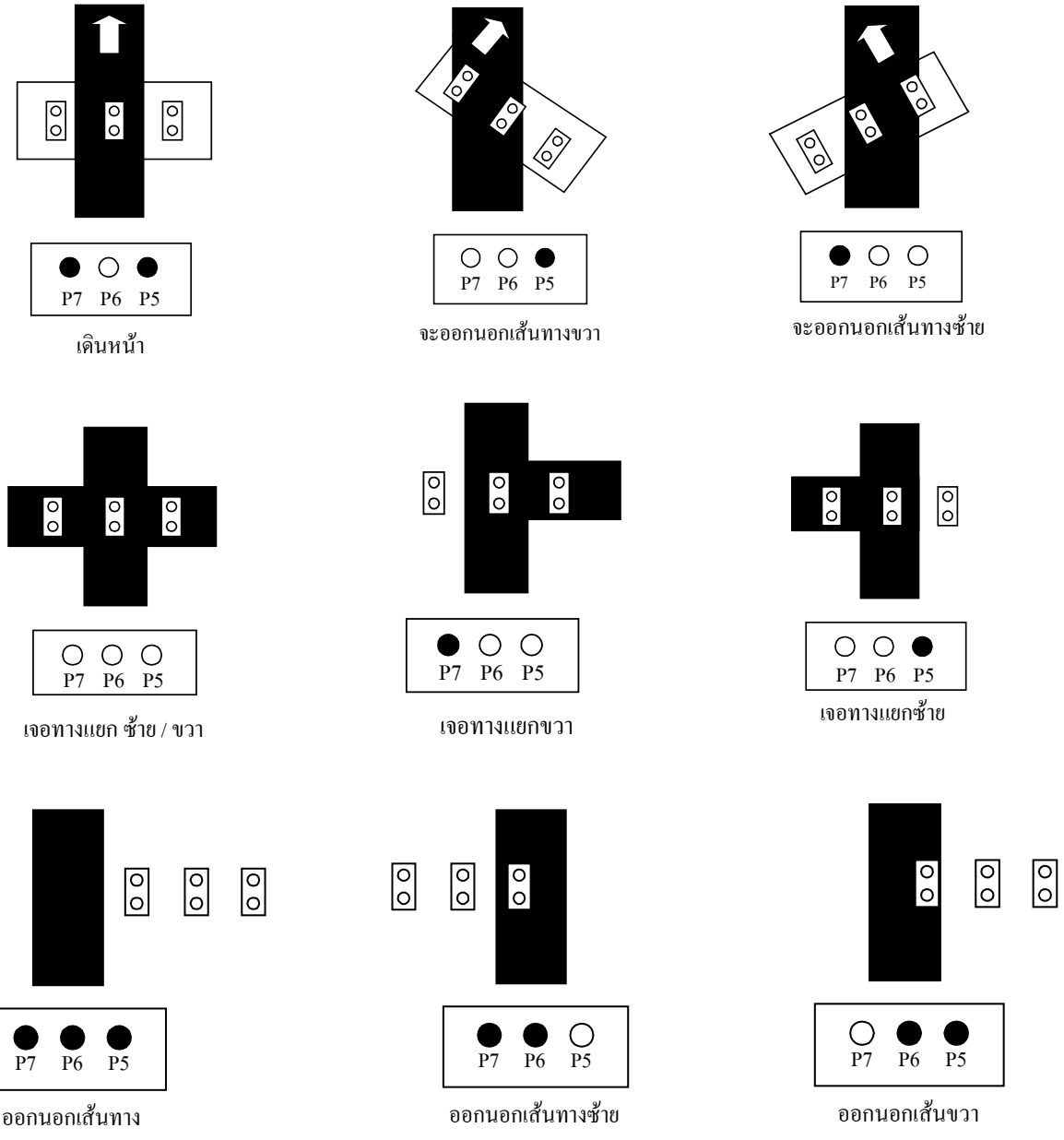
```
'########## MotorForward ##########
MotorForward:
PULSOUT 13,1250                 ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                        ' สร้าง Pulse Off = 20 mS
GOTO LoopMain                   ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorStop ##########
MotorStop:
LOW 13              ' เมื่อ Motor หยุดต้องป้อน Low ทั้งคู่ไม่เช่นนั้นเมื่อทำเงื่อนไขนี้แล้วจะทำให้โปรแกรมรวน
LOW 14
GOTO LoopMain       ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnLeft ##########
MotorReturnLeft:
PULSOUT 13,1250     ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,1250     ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20           ' สร้าง Pulse Off = 20 mS
GOTO LoopMain       ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnRight ##########
MotorReturnRight:
PULSOUT 13,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20           ' สร้างช่วง Pulse Off = 20 mS
GOTO LoopMain       ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

จากตัวอย่างโปรแกรมที่ 5.3 เป็นการควบคุมให้หุ่นยนต์เดินตามแสงโดยอาศัยการอ่านค่าความเข้มแสงของเซนเซอร์ทางด้านซ้ายและขวามาเปรียบเทียบกัน แล้วให้หุ่นยนต์เคลื่อนที่ตามเงื่อนไขที่เรากำหนดขึ้น ดังนั้นเราอาจจะเปลี่ยนการทำงานใหม่ก็ได้ เช่น ให้หุ่นยนต์เดินหนีแสง ซึ่งก็สามารถทำได้ทั้งทางซอร์ฟแวร์ คือ แก้ไขโปรแกรมใหม่ หรือ ทางฮาร์ดแวร์ ก็คือการสลับที่ของตัวเซนเซอร์ทางด้านซ้ายและขวา หรือ อาจจะเพิ่มความฉลาดของหุ่นยนต์ขึ้นอีก ด้วยการเพิ่มฟังก์ชันการหลบหลีกสิ่งกีดขวาง โดยการเช็คเงื่อนไขจากสวิตช์กันชนเพิ่มเข้าไป เป็นต้น

5.3 การควบคุมหุ่นยนต์ให้เดินตามเส้น

หลักการของการควบคุมหุ่นยนต์เดินตามเส้นนั้น ส่วนประกอบสำคัญก็คือวงจรเซ็นเซอร์ที่ใช้ตรวจจับเส้น ซึ่งจะเป็นวงจรอินฟราเรด โดยรายละเอียดและหลักการของวงจรเซ็นเซอร์ตรวจจับเส้นนี้ดูได้จากหัวข้อที่ 1.4

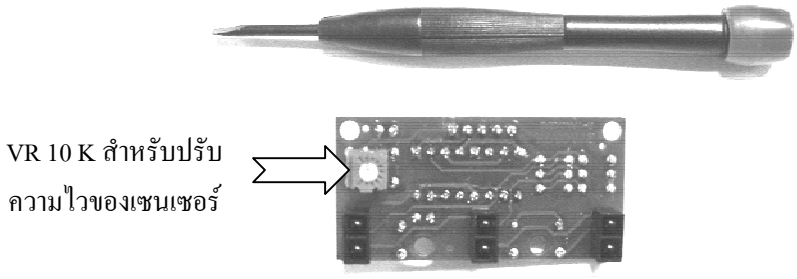
ในการควบคุมทิศทางของหุ่นยนต์ให้เคลื่อนที่ไปตามเส้นนั้นจะต้องออกแบบโปรแกรมให้มีการเช็คเงื่อนไขจากวงจรตรวจจับเส้น ซึ่งโมดูลนี้มีอยู่ด้วยกัน 3 จุด คือ ซ้าย(P7) , กลาง(P6) และ ขวา(P5) ทำให้เงื่อนไขต่างๆ มีความเป็นไปได้หลายรูปแบบ (2^3 = 8 สภาวะ) ซึ่งรูปแบบของสภาวะต่างๆ ในการตรวจจับเส้นจะมีลักษณะดังนี้



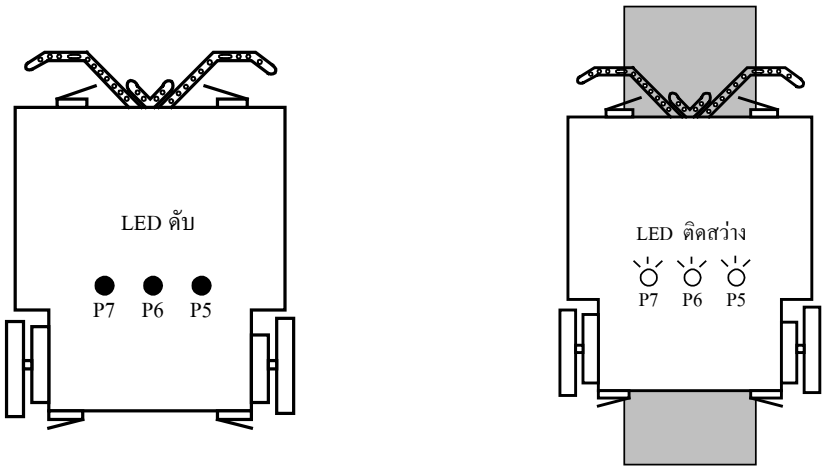
รูปที่ 5.4 แสดงลักษณะรูปแบบต่างๆ ของการตรวจจับเส้น

ก่อนการใช้งานตัวเซนเซอร์ตรวจจับเส้น จะต้องทำการปรับระดับความไวในการตรวจจับเส้นของโมดูลอินฟราเรดให้อยู่ในระดับที่จะสามารถทำงานได้ก่อน เนื่องจากแต่ละพื้นที่ที่เราจะนำไปใช้งานนั้นมีขนาดของแสงหรือ พื้นผิวของวัตถุแตกต่างกันทำให้มีผลต่อการทำงานของวงจรเซนเซอร์ตรวจจับเส้น ซึ่งสามารถทำได้โดยการปรับค่าที่ตัวความต้านทานปรับค่า VR 10K ที่อยู่ด้านล่างของแผงวงจรเซนเซอร์ โดยใช้ไขควงหมุน VR 10 K แล้วทำการทดสอบด้วยการนำเอา Robot และตัวเซนเซอร์ไปวางขนานกับพื้นสนามที่เราต้องการในส่วนที่เป็นสีดำ และสีขาว โดยจะมีวิธีการพิจารณาได้ดังนี้

- กรณีวางตัวเซนเซอร์อยู่บนพื้นสีขาวจะมีการสะท้อนกลับของสัญญาณทำให้เอาต์พุตของวงจรเซนเซอร์นั้นมี Logic “0” ดูได้จาก LED สถานะ P7,P6 และ P5 บนบอร์ด ET-ROBOT STAMP ซึ่งจะต้องติดสว่าง
- กรณีวางตัวเซนเซอร์บนพื้นสีดำ (เส้นทางเดินของหุ่นยนต์) จะไม่มีการสะท้อนกลับของสัญญาณหรือ มีแต่อยู่ในระดับที่ต่ำมากทำให้ สถานะเอาต์พุตของวงจรเซนเซอร์มี Logic “1” และ สถานะของ LED บนบอร์ด P7,P6 และ P5 จะดับ



รูปที่ 5.5 แสดงตำแหน่งของ VR 10k ที่ใช้ปรับความไวของเซนเซอร์



รูปที่ 5.6 แสดงลักษณะการปรับค่าความไวของเซนเซอร์บนพื้นผิวของสนาม

ในการออกแบบโปรแกรมให้หุ่นยนต์เดินตามเส้นนั้น จะต้องทำให้ครอบคลุมสภาวะต่างๆ ดังรูปที่ 5.4 ทุกสภาวะซึ่งอาจจะเขียนเป็นตารางความจริง (True Table) ดังตัวอย่างต่อไปนี้

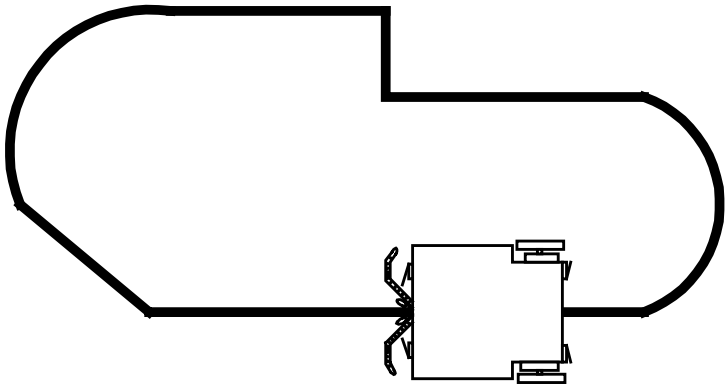
เงื่อนไขที่	INPUT TRACKER Sensor			OUTPUT Servo Motor		หมายเหตุ
	ซ้าย P7	กลาง P6	ขวา P5	ซ้าย P14	ขวา P13	
1	0	0	0	CCW	CW	เดินหน้า
2	0	0	1	CCW	CCW	เลี้ยวขวาเข้าเส้น
3	0	1	0	CCW	CW	เดินหน้า
4	0	1	1	CCW	CCW	เลี้ยวขวา 90 องศา
5	1	0	0	CW	CW	เลี้ยวซ้ายเข้าเส้น
6	1	0	1	CCW	CW	เดินหน้า
7	1	1	0	CW	CW	เลี้ยวซ้าย 90 องศา
8	1	1	1	CCW	CW	เดินหน้า

ตัวอย่างที่ 5.4 โปรแกรมควบคุมหุ่นยนต์เดินตามเส้น

```
'{$STAMP BS2p}
LoopLeft     VAR    WORD      ' ประกาศตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางซ้าย
LoopRight    VAR    WORD      ' ประกาศตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางขวา
LoopStraight VAR    WORD      ' ประกาศตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวในแนวเส้นตรง

'########## โปรแกรมหลัก ##########
LoopMain:
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 0 ) THEN MotorForward              'Robot เดินไปข้างหน้า
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 1 ) THEN MotorReturnRightGoLine 'Robot เลี้ยวขวาเข้าเส้น
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 0 ) THEN MotorForward              'Robot เดินไปข้างหน้า
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 1 ) THEN MotorReturnRight90        'Robot เลี้ยวขวา
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 0 ) THEN MotorReturnLeftGoLine   'Robot เลี้ยวซ้ายเข้าเส้น
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 1 ) THEN MotorForward              'Robot เดินไปข้างหน้า
IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 0 ) THEN MotorReturnLeft90         'Robot เลี้ยวซ้าย
IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 1 ) THEN MotorForward              'Robot เดินไปข้างหน้า
GOTO     LoopMain                                                     ' วนตรวจสอบสภาวะ Input
```

```
'########## MotorForward เดินหน้า ##########
MotorForward:
PULSOUT 13,1250                    ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,2500                    ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                  ' สร้าง Pulse Off = 20 mS
GOTO LoopMain          ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnLeft90 เดินหน้าระยะหนึ่งแล้วเลี้ยวซ้าย ##########
MotorReturnLeft90:
FOR LoopLeft = 1 TO 30    ' เดินตรงไปข้างหน้าอีกระยะก่อนเลี้ยวซ้าย 90 องศาเพื่อให้ ตัวรถและ Sensor อยู่กึ่งกลางเส้น
  PULSOUT 13,1250          ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
  PULSOUT 14,2500          ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                  ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopLeft = 1 TO 40    ' รถเลี้ยวซ้ายมุม 90 องศา
  PULSOUT 13,1250          ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
  PULSOUT 14,1250          ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                  ' สร้าง Pulse Off = 20 mS
NEXT
GOTO    LoopMain          ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnLeftGoLine เลี้ยวซ้ายเข้าเส้น ##########
MotorReturnLeftGoLine:
PULSOUT 13,1250          ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,1250          ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                ' สร้าง Pulse Off = 20 mS
GOTO LoopMain            ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnRight90 ##########
MotorReturnRight90:
FOR LoopRight = 1 TO 30    ' เดินตรงไปข้างหน้าอีกระยะแล้วเลี้ยวขวา 90 องศาเพื่อให้ ตัวรถ และ Sensor อยู่กึ่งกลางเส้น
  PULSOUT 13,1250          ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
  PULSOUT 14,2500          ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                   ' สร้าง Pulse Off = 20 mS
NEXT
```

```
FOR LoopRight = 1 TO 40    ' เลี้ยวขวามุม 90 องศา
  PULSOUT 13,2500          ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
  PULSOUT 14,2500          ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                   ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain              ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnRightGoLine ##########
MotorReturnRightGoLine:
 PULSOUT 13,2500           ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,2500           ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                  ' สร้างช่วง Pulse Off = 20 mS
GOTO LoopMain              ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```



รูปที่ 5.7 ตัวอย่างเส้นทาง หรือ สนามที่จะให้หุ่นยนต์วิ่ง

จากตัวอย่างที่ผ่านๆ มาเราได้ทำการศึกษาการเขียนโปรแกรมในแต่ละส่วนไปแล้ว และ ในตัวอย่างนี้เราจะนำเอาแต่ละส่วนมารวมกันเพื่อให้หุ่นยนต์ของเรามี ศักยภาพและความฉลาดมากขึ้น ซึ่งจะทำการสั่งให้หุ่นยนต์เดินตามเส้นพร้อมกับทำการเช็คสภาวะกันชนหน้าหลังด้วย เพื่อหลบหลีกสิ่งกีดขวาง โดยจะเขียนเป็นตารางสภาวะ หรือตารางความจริงก่อน แล้วจึงนำเอาเงื่อนไขในตารางไปเขียนเป็นโปรแกรม ดังตัวอย่างต่อไปนี้

เงื่อนไข	INPUT Sensor							OUTPUT		หมายเหตุ
	เซนเซอร์อ่านเส้น			เซนเซอร์กันชน				Servo Motor		
	P7	P6	P5	P4	P3	P1	P0	ซ้าย P14	ขวา P13	
1	X	X	X	0	0	0	0	หยุด	หยุด	หยุด
2	X	X	X	0	0	0	1	หยุด	หยุด	หยุด
3	X	X	X	0	0	1	0	หยุด	หยุด	หยุด
4	X	X	X	0	0	1	1	CW	CCW	ถอยหลัง
5	X	X	X	0	1	0	0	หยุด	หยุด	หยุด
6	X	X	X	0	1	0	1	หยุด	หยุด	หยุด
7	X	X	X	0	1	1	0	หยุด	หยุด	หยุด
8	X	X	X	0	1	1	1	CCW	CCW	เลี้ยวขวาหลบสิ่งกีดขวางทางซ้าย
9	X	X	X	1	0	0	0	หยุด	หยุด	หยุด
10	X	X	X	1	0	0	1	หยุด	หยุด	หยุด
11	X	X	X	1	0	1	0	หยุด	หยุด	หยุด
12	X	X	X	1	0	1	1	CW	CW	เลี้ยวซ้ายหลบสิ่งกีดขวางทางขวา
13	X	X	X	1	1	0	0	CCW	CW	เดินหน้า
14	X	X	X	1	1	0	1	CCW	CW	เดินหน้า
15	X	X	X	1	1	1	0	CCW	CW	เดินหน้า
16	X	X	X	1	1	1	1	CCW	CW	เดินหน้า
17	0	0	0	1	1	1	1	CCW	CW	เดินหน้า
18	0	0	1	1	1	1	1	CCW	CCW	เลี้ยวขวาเข้าเส้นทาง
19	0	1	0	1	1	1	1	CCW	CW	เดินหน้า
20	0	1	1	1	1	1	1	CCW	CCW	เลี้ยวขวา 90 องศา
21	1	0	0	1	1	1	1	CW	CW	เลี้ยวซ้ายเข้าเส้นทาง
22	1	0	1	1	1	1	1	CCW	CW	เดินหน้า
23	1	1	0	1	1	1	1	CW	CW	เลี้ยวซ้าย 90 องศา
24	1	1	1	1	1	1	1	CCW	CW	เดินหน้า

*** หมายเหตุ** เมื่อ Robot หยุดเดินจะมีการกำเนิดเสียงที่ลำโพง (P15)

ตัวอย่างที่ 5.5 โปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นพร้อมกับ หลบเลี่ยงสิ่งกีดขวาง

```
'{$STAMP BS2p}
LoopLeft      VAR   WORD        ' ตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางซ้าย
LoopRight     VAR   WORD        ' ตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางขวา
LoopStraight  VAR   WORD        ' ตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวในแนวเส้นตรง
```

```
'########## โปรแกรมหลัก ##########
LoopMain:
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 0 AND IN0 = 0 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 0 AND IN0 = 1 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 1 AND IN0 = 0 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN MotorBackward            ' Robot ถอยหลัง
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 0 AND IN0 = 0 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 0 AND IN0 = 1 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 1 AND IN0 = 0 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorEvadeLeft           ' Robot หลบสิ่งกีดขวางที่อยู่ทางซ้าย
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 0 AND IN0 = 0 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 0 AND IN0 = 1 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 1 AND IN0 = 0 ) THEN MotorStop                ' Robot หยุดเดิน
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN MotorEvadeRight          ' Robot หลบสิ่งกีดขวางที่อยู่ทางขวา
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 0 ) THEN MotorForward             ' Robot เดินไปข้างหน้า
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 1 ) THEN MotorForward             ' Robot เดินไปข้างหน้า
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 0 ) THEN MotorForward             ' Robot เดินไปข้างหน้า
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnRightGoLine
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnRight90
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnLeftGoLine
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnLeft90
IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
GOTO    LoopMain                                                                 ' วนตรวจสอบสถาวะ Input
'########## MotorForward ##########
MotorForward:
PAUSE 10                    ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน  แล้วตรวจสอบสภาวะ Input อีกครั้ง
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 0 ) THEN NextMotorForward
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 1 ) THEN NextMotorForward
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 0 ) THEN NextMotorForward
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorForward
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorForward
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorForward
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorForward
GOTO LoopMain                       ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input
NextMotorForward:
        PULSOUT 13,1250             ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
        PULSOUT 14,2500             ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
        PAUSE 20                    ' สร้าง Pulse Off = 20 mS
GOTO LoopMain                       ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

```
'########## MotorBackward ##########
MotorBackward:
    PAUSE 10                                    ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
                                                ' ตรวจสอบสภาวะ Input อีกครั้งเพื่อป้องกันสัญญาณรบกวน
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorBackward
   GOTO  LoopMain                               ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input ทั้งหมด
NextMotorBackward:
  FOR LoopStraight = 1 TO 50                    ' ถอยหลังไประยะหนึ่ง
    PULSOUT 13,2500                             ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PULSOUT 14,1250                             ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
    PAUSE 20                                    ' สร้าง Pulse Off = 20 mS
  NEXT
GOTO LoopMain                                   ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorStop ##########
MotorStop:
PAUSE 10     ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
             ' ตรวจสอบสภาวะ Input อีกครั้งเพื่อป้องกันสัญญาณรบกวน
             IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 0 AND IN0 = 0 ) THEN NextMotorStop
             IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 0 AND IN0 = 1 ) THEN NextMotorStop
             IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 1 AND IN0 = 0 ) THEN NextMotorStop
             IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 0 AND IN0 = 0 ) THEN NextMotorStop
             IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 0 AND IN0 = 1 ) THEN NextMotorStop
             IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 1 AND IN0 = 0 ) THEN NextMotorStop
             IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 0 AND IN0 = 0 ) THEN NextMotorStop
             IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 0 AND IN0 = 1 ) THEN NextMotorStop
             IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 1 AND IN0 = 0 ) THEN NextMotorStop
             GOTO LoopMain                      ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input
NextMotorStop:
             LOW 13                             ' เมื่อ Motor หยุดต้องป้อน Low ทั้งคู่ไม่เช่นนั้นอาจจะทำให้โปรแกรมรวนได้
             LOW 14
             FREQOUT 15,1000,1000,1000          ' กำเนิดความถี่เสียงป้อนให้กับลำโพง
             GOTO LoopMain                      ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'@@@@@@@@@@ กลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางซ้าย @@@@@@@@@@
'########## MotorReturnLeft90 ##########
MotorReturnLeft90:
             PAUSE 10                           ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
                                                ' ตรวจสอบสภาวะ Input อีกครั้งเพื่อป้องกันสัญญาณรบกวน
  IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorReturnLeft90
  IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorReturnLeft90
    GOTO    LoopMain                            ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

```
NextMotorReturnLeft90:
 FOR LoopLeft = 1 TO 30        ' เดินตรงไปข้างหน้าอีกหน่อยเพื่อให้รถเลี้ยวซ้ายมุม 90 องศา ตัวรถและ Sensor จะได้อยู่กึ่งกลางเส้น
  PULSOUT 13,1250               ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
  PULSOUT 14,2500               ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
  PAUSE 20                      ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopLeft = 1 TO 40          ' รถเลี้ยวซ้ายมุม 90 องศา
 PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                       ' สร้าง Pulse Off = 20 mS
NEXT
GOTO   LoopMain                 ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorEvadeRight ##########
MotorEvadeRight:
PAUSE 10                        ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
                                ' ตรวจสอบสภาวะ Input อีกครั้งเพื่อป้องกันสัญญาณรบกวน
   IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorEvadeRight
  GOTO LoopMain                 ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input ทั้งหมด
NextMotorEvadeRight:
  FOR LoopRight = 1 TO 40       ' ถอยหลังไประยะหนึ่ง
    PULSOUT 13,2500             ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PULSOUT 14,1250             ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
    PAUSE 20                    ' สร้าง Pulse Off = 20 mS
 NEXT
 FOR LoopRight = 1 TO 20        ' เลี้ยวซ้ายประมาณ 45 องศา เพื่อหลบสิ่งกีดขวาง
   PULSOUT 13,1250              ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
   PULSOUT 14,1250              ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
   PAUSE 20                     ' สร้าง Pulse Off = 20 mS
 NEXT
 GOTO LoopMain                  ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnLeftGoLine ##########
MotorReturnLeftGoLine:
   PAUSE 10                     ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorReturnLeftGoLine
   GOTO LoopMain                ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input
NextMotorReturnLeftGoLine:
   PULSOUT 13,1250              ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
   PULSOUT 14,1250              ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
   PAUSE 20                     ' สร้าง Pulse Off = 20 mS
  GOTO LoopMain                 ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

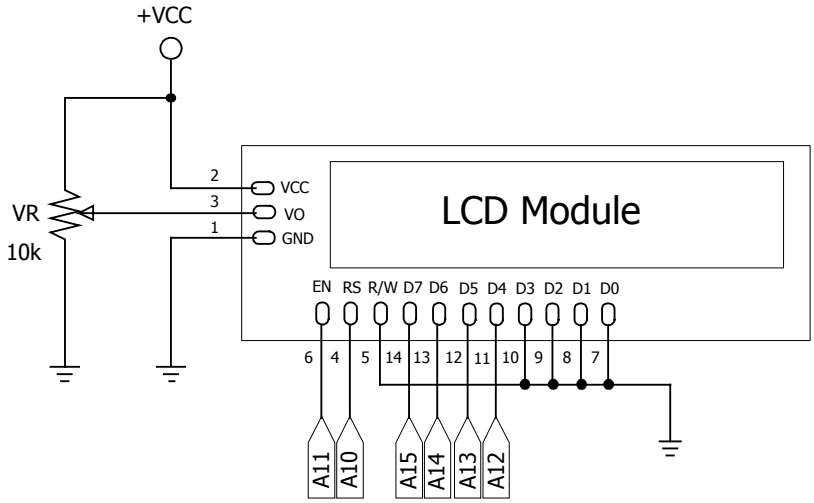
```
'@@@@@@@@@@ กลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางขวา @@@@@@@@@@
'########## MotorReturnRight90 ##########
MotorReturnRight90:
        PAUSE 10                    ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
                                    ' ตรวจสอบสภาวะ Input อีกครั้งเพื่อป้องกันสัญญาณรบกวน
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorReturnRight90
   GOTO LoopMain                    ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input
NextMotorReturnRight90:
   FOR LoopRight = 1 TO 30    ' เดินตรงไปข้างหน้าอีกหน่อยเพื่อให้รถเลี้ยวขวามุม 90 องศา ตัวรถและ Sensor จะได้อยู่กึ่งกลางเส้น
     PULSOUT 13,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
     PULSOUT 14,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
     PAUSE 20                       ' สร้าง Pulse Off = 20 mS
   NEXT
  FOR LoopRight = 1 TO 40           ' เลี้ยวขวามุม 90 องศา
    PULSOUT 13,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PAUSE 20                        ' สร้าง Pulse Off = 20 mS
  NEXT
GOTO LoopMain                       ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorEvadeLeft ##########
MotorEvadeLeft:
   PAUSE 10                         ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
                                    ' ตรวจสอบสภาวะ Input อีกครั้งเพื่อป้องกันสัญญาณรบกวน
   IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorEvadeLeft
  GOTO LoopMain                     ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input
NextMotorEvadeLeft:
   FOR LoopLeft = 1 TO 40           ' ถอยหลังไประยะหนึ่ง
     PULSOUT 13,2500                ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
     PULSOUT 14,1250                ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
     PAUSE 20                       ' สร้าง Pulse Off = 20 mS
   NEXT
  FOR LoopLeft = 1 TO 20            ' เลี้ยวขวาประมาณ 45 องศา เพื่อหลบสิ่งกีดขวาง
    PULSOUT 13,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PAUSE 20                        ' สร้าง Pulse Off = 20 mS
  NEXT
 GOTO   LoopMain                    ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

```
'########## MotorReturnRightGoLine ##########
MotorReturnRightGoLine:
    PAUSE 10                        ' หน่วงเวลาป้องกันสัญญาณรบกวนหรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN NextMotorReturnRightGoLine
    GOTO LoopMain                   ' ถ้าเป็นสัญญาณรบกวนให้กลับไปโปรแกรมหลักตรวจสภาวะ Input
NextMotorReturnRightGoLine:
    PULSOUT 13,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PULSOUT 14,2500                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
    PAUSE 20                        ' สร้างช่วง Pulse Off = 20 mS
    GOTO LoopMain                   ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

*** หมายเหตุ**

1. ในแต่ละสถานที่จะมีความเข้มของแสงแตกต่างกัน ดังนั้นต้องปรับความไวในการรับแสงของเซนเซอร์ Infrared ทุกครั้ง โดยปรับที่ VR 10k ที่ภาค Sensor ตรวจจับเส้น
2. จะเห็นว่ามีการหน่วงเวลาหลังจากพบว่าสวิตช์กันชนทำงาน (Pause 10) ทั้งนี้เพื่อป้องกันสัญญาณรบกวนจากหน้าสัมผัสสวิตช์ (Bounce) หรือรอให้หน้าสัมผัสกระเด้งเสร็จก่อน และ จะมีการตรวจสอบสภาวะ Input อีกครั้งเพื่อป้องกันสัญญาณรบกวนเพื่อให้ Robot ตรวจ Input ได้ไวขึ้น

ตัวอย่างที่ 5.6 โปรแกรมควบคุมหุ่นยนต์เดินตามเส้นพร้อมกับแสดงผลข้อความบน LCD



รูปที่ 5.8 แสดงวงจรการต่อ LCD กับขาสัญญาณของ Basic Stamp (ET-CLCD)

โดย LCD ที่จะใช้กับตัวอย่างนี้เป็นแบบ 16 ตัวอักษร 2 บรรทัด ซึ่งจะแสดงข้อความตามการทำงานของหุ่นยนต์ดังนี้

การเคลื่อนที่ของหุ่นยนต์	การแสดงข้อความบนจอ LCD
เดินหน้า	"Forward"
เลี้ยวซ้าย 90 องศา	"Return Left 90"
เลี้ยวขวา 90 องศา	"Return Right 90"
ถอยหลัง	"Backward "
หยุด	"Stop"
หลบสิ่งกีดขวางทางขวา	"Evade Right"
หลบสิ่งกีดขวางทางซ้าย	"Evade Left"

```
'{$STAMP BS2p}
LoopLeft      VAR       WORD      ' ประกาศตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางซ้าย
LoopRight     VAR       WORD      ' ประกาศตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางขวา
LoopStraight  VAR       WORD      ' ประกาศตัวแปรชนิด WORD ใช้ในกลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวในแนวเส้นตรง
ASCII                   VAR BYTE
OrderASCII    VAR       BYTE

'########## โปรแกรมหลัก ##########
 AUXIO                          ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
 PAUSE 1000                     ‘ หน่วงเวลา 1000 ms เพื่อรอให้ LCD พร้อมทำงาน
 GOSUB InitialLCD               ‘ กำหนดค่าเริ่มต้นการทำงานของ LCD
MAINIO                          ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
LoopMain:
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 0 AND IN0 = 0 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 0 AND IN0 = 1 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 1 AND IN0 = 0 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN MotorBackward      ' Robot ถอยหลัง
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 0 AND IN0 = 0 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 0 AND IN0 = 1 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 1 AND IN0 = 0 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 0 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorEvadeLeft     ' Robot หลบสิ่งกีดขวางที่อยู่ทางซ้าย
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 0 AND IN0 = 0 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 0 AND IN0 = 1 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 1 AND IN0 = 0 ) THEN MotorStop          ' Robot หยุดเดิน
IF ( IN4 = 1 AND IN3 = 0 AND IN1 = 1 AND IN0 = 1 ) THEN MotorEvadeRight    ' Robot หลบสิ่งกีดขวางที่อยู่ทางขวา
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 0 ) THEN MotorForward       ' Robot เดินไปข้างหน้า
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 0 AND IN0 = 1 ) THEN MotorForward       ' Robot เดินไปข้างหน้า
IF ( IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 0 ) THEN MotorForward       ' Robot เดินไปข้างหน้า
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward   'Robot เดินไปข้างหน้
IF ( IN7 = 0 AND IN6 = 0 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnRightGoLine
```

```
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
IF ( IN7 = 0 AND IN6 = 1 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnRight90
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnLeftGoLine
IF ( IN7 = 1 AND IN6 = 0 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 0 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorReturnLeft90
IF ( IN7 = 1 AND IN6 = 1 AND IN5 = 1 AND IN4 = 1 AND IN3 = 1 AND IN1 = 1 AND IN0 = 1 ) THEN MotorForward
GOTO    LoopMain                                    ' วนตรวจสอบสถาวะ Input

'########## MotorForward  (เดินหน้า)  ##########
MotorForward:
AUXIO                                               ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
ASCII = $1                                          ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
GOSUB SendCommandLCD                                ‘ ส่งข้อมูลคำสั่ง
FOR OrderASCII = 0 TO 6                             ‘ 7 ลูปเพื่อส่งข้อความในตาราง 7 ตัวอักษร
 LOOKUP OrderASCII,["Forward"],ASCII                ‘  เปิดตารางส่งข้อความคำว่า Forward
 GOSUB SendASCII                                    ‘ ส่งข้อมูลตัวอักษรไปแสดงผลที่ LCD
NEXT
MAINIO                                              ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
PULSOUT 13,1250                                     ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,2500                                     ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                                            ' สร้าง Pulse Off = 20 mS
GOTO LoopMain                                       ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
'########## MotorBackward (ถอยหลัง) ##########
MotorBackward:
AUXIO                                               ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
ASCII = $1                                          ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
GOSUB SendCommandLCD                                ‘ ส่งข้อมูลคำสั่ง
FOR OrderASCII = 0 TO 7                             ‘ 8  ลูปเพื่อส่งข้อความในตาราง 8 ตัวอักษร
 LOOKUP OrderASCII,["Backward"],ASCII               ‘ เปิดตารางข้อมูลเก็บไว้ที่ตัวแปร ASCII
 GOSUB SendASCII                                    ‘ ส่งข้อมูลตัวอักษรไปแสดงผลที่ LCD
NEXT
MAINIO                                              ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
FOR LoopStraight = 1 TO 50                          ' ถอยหลังไประยะหนึ่ง
 PULSOUT 13,2500                                    ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,1250                                    ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PAUSE 20                                            ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain                                       ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

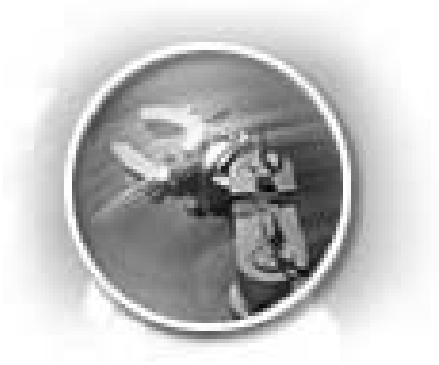
```
'@@@@@@@@@@ กลุ่มโปรแกรมที่ทำให้ Robot หยุดเคลื่อนที่ @@@@@@@@@@
'########## MotorStop ##########
MotorStop:
AUXIO                                          ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
ASCII = $1                                     ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
GOSUB SendCommandLCD                           ‘ ส่งข้อมูลคำสั่ง
FOR OrderASCII = 0 TO 3                        ‘ 4  ลูปเพื่อส่งข้อความในตาราง 4  ตัวอักษร
 LOOKUP OrderASCII,["Stop"],ASCII              ‘ เปิดตารางข้อมูลเก็บไว้ที่ตัวแปร ASCII
 GOSUB  SendASCII                              ‘ ส่งข้อมูลตัวอักษรไปแสดงผลที่ LCD
NEXT
MAINIO                                         ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
LOW 13                                 ' เมื่อ Motor หยุดต้องป้อน Low ทั้งคู่ไม่เช่นนั้นเมื่อทำเงื่อนไขนี้แล้วจะทำให้โปรแกรมรวน
LOW 14
FREQOUT 15,1000,1000,1000                      ‘ กำเนิดความถี่ออกไปที่ลำโพง
GOTO LoopMain                                  ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'@@@@@@@@@@ กลุ่มโปรแกรมที่ทำให้ Robot เคลื่อนตัวไปทางซ้าย @@@@@@@@@@
'########## MotorReturnLeft90 (เลี้ยวซ้าย 90 องศา)  ##########
MotorReturnLeft90:
AUXIO                                          ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
ASCII = $1                                     ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
GOSUB SendCommandLCD                           ‘ ส่งข้อมูลคำสั่ง
FOR OrderASCII = 0 TO 13                       ‘ 14  ลูปเพื่อส่งข้อความในตาราง 14 ตัวอักษร
 LOOKUP OrderASCII,["Return Left 90"],ASCII    ‘ เปิดตารางข้อมูลเก็บไว้ที่ตัวแปร ASCII
 GOSUB SendASCII                               ‘ ส่งข้อมูลตัวอักษรไปแสดงผลที่ LCD
NEXT
MAINIO                  ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
FOR LoopLeft = 1 TO 30  ' เดินตรงไปข้างหน้าอีกหน่อยเพื่อให้รถเลี้ยวซ้ายมุม 90 องศา ตัวรถและ Sensor จะได้อยู่กึ่งกลางเส้น
 PULSOUT 13,1250        ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,2500        ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20               ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopLeft = 1 TO 40  ' รถเลี้ยวซ้ายมุม 90 องศา
 PULSOUT 13,1250        ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PULSOUT 14,1250        ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20               ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain           ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
```

```
'########## MotorEvadeRight (หลบสิ่งกีดขวางทางขวา) ##########
MotorEvadeRight:
AUXIO                                   ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
ASCII = $1                              ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
GOSUB SendCommandLCD                    ‘ ส่งข้อมูลคำสั่ง
FOR OrderASCII = 0 TO 10                ‘ 11  ลูปเพื่อส่งข้อความในตาราง 11 ตัวอักษร
 LOOKUP OrderASCII,["Evade Right"],ASCII ‘ เปิดตารางข้อมูลเก็บไว้ที่ตัวแปร ASCII
 GOSUB SendASCII                        ‘ ส่งข้อมูลตัวอักษรไปแสดงผลที่ LCD
NEXT
MAINIO                                  ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
FOR LoopRight = 1 TO 40                 ' ถอยหลังไประยะหนึ่ง
  PULSOUT 13,2500                       ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
  PULSOUT 14,1250                       ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                               ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopRight = 1 TO 20                 ' เลี้ยวซ้ายประมาณ 45 องศา เพื่อหลบสิ่งกีดขวาง
PULSOUT 13,1250                         ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,1250                         ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                               ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain                           ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
'########## MotorReturnLeftGoLine (เลี้ยวซ้ายเข้าเส้น)  ##########
MotorReturnLeftGoLine:
PULSOUT 13,1250                         ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
PULSOUT 14,1250                         ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                               ' สร้าง Pulse Off = 20 mS
GOTO LoopMain                           ' กลับไปโปรแกรมหลักตรวจสภาวะ Input

'########## MotorReturnRight90 (เลี้ยวขวา 90 องศา)  ##########
MotorReturnRight90:
AUXIO                                   ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
ASCII = $1                              ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
GOSUB SendCommandLCD                    ‘ ส่งข้อมูลคำสั่ง
FOR OrderASCII = 0 TO 14                ‘ 15  ลูปเพื่อส่งข้อความในตาราง 15 ตัวอักษร
 LOOKUP OrderASCII,["Return Right 90"],ASCII ‘ เปิดตารางข้อมูลเก็บไว้ที่ตัวแปร ASCII
 GOSUB SendASCII                        ‘ ส่งข้อมูลตัวอักษรไปแสดงผลที่ LCD
NEXT
MAINIO                  ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
FOR LoopRight = 1 TO 30 ' เดินตรงไปข้างหน้าอีกหน่อยเพื่อให้รถเลี้ยวขวามุม 90 องศา ตัวรถและ Sensor จะได้อยู่กึ่งกลางเส้น
  PULSOUT 13,1250       ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
  PULSOUT 14,2500       ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                ' สร้าง Pulse Off = 20 mS
NEXT
```

```
FOR LoopRight = 1 TO 40                          ' เลี้ยวขวามุม 90 องศา
 PULSOUT 13,2500                                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,2500                                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PAUSE 20                                         ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain                                    ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
'########## MotorEvadeLeft  (หลบสิ่งกีดขวางทางซ้าย) ##########
MotorEvadeLeft:
AUXIO                                            ‘ เลือกใช้ขาสัญญาณ I/O สำรอง (Auxiliary I/O  A0-A15)
ASCII = $1                                       ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
GOSUB SendCommandLCD                             ‘ ส่งข้อมูลคำสั่ง
FOR OrderASCII = 0 TO 9                          ‘ 10 ลูปเพื่อส่งข้อความในตาราง 10 ตัวอักษร
 LOOKUP OrderASCII,["Evade Left"],ASCII          ‘ เปิดตารางข้อมูลเก็บไว้ที่ตัวแปร ASCII
 GOSUB SendASCII                                 ‘ ส่งข้อมูลตัวอักษรไปแสดงผลที่ LCD
NEXT
MAINIO                                           ‘ กลับมาใช้งาน I/O หลัก (P0-P15)
FOR LoopLeft = 1 TO 40                           ' ถอยหลังไประยะหนึ่ง
 PULSOUT 13,2500                                 ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PULSOUT 14,1250                                 ' สร้าง Pulse Width = 1 mS ทำให้ Motor หมุนตามเข็มนาฬิกา
 PAUSE 20                                        ' สร้าง Pulse Off = 20 mS
NEXT
FOR LoopLeft = 1 TO 20                           ' เลี้ยวขวาประมาณ 45 องศา เพื่อหลบสิ่งกีดขวาง
PULSOUT 13,2500                                  ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PULSOUT 14,2500                                  ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                                        ' สร้าง Pulse Off = 20 mS
NEXT
GOTO LoopMain                                    ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
'########## MotorReturnRightGoLine (เลี้ยวขวาเข้าเส้น )  ##########
MotorReturnRightGoLine:
PULSOUT 13,2500                                  ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
PULSOUT 14,2500                                  ' สร้าง Pulse Width = 2 mS ทำให้ Motor หมุนทวนเข็มนาฬิกา
 PAUSE 20                                        ' สร้างช่วง Pulse Off = 20 mS
GOTO LoopMain                                    ' กลับไปโปรแกรมหลักตรวจสภาวะ Input
'********** โปรแกรม InitialLCD **********
InitialLCD:
 ASCII = $33                                     ‘ 00110011 =  8 bit mode
 GOSUB SendCommandLCD                            ‘ ส่งข้อมูลคำสั่ง
 ASCII = $32                                     ‘ 00110010 = 8 bit mode
 GOSUB SendCommandLCD                            ‘ ส่งข้อมูลคำสั่ง
 ASCII = $28                                     ‘ 00101000 = 4 bit mode , 2 บรรทัด , 5x7 จุด
 GOSUB SendCommandLCD                            ‘ ส่งข้อมูลคำสั่ง
 ASCII = $C                                      ‘ เปิดจอแสดงผล, ไม่แสดงเคอร์เซอร์
```

```
GOSUB SendCommandLCD                ‘ ส่งข้อมูลคำสั่ง
 ASCII = $6                         ‘ เลื่อนตำแหน่งเคอร์เซอร์ขึ้นเมื่อมีการเขียนข้อมูลบนจอ LCD
 GOSUB SendCommandLCD               ‘ ส่งข้อมูลคำสั่ง
 ASCII = $1                         ‘ ข้อมูลคำสั่งสำหรับเคลียร์หน้าจอ LCD
 GOSUB SendCommandLCD               ‘ ส่งข้อมูลคำสั่ง
 RETURN
'********** โปรแกรม SendASCII และ SendCommandLCD **********
SendASCII:
 HIGH 10  : GOTO NextSend           ‘ บิต RS = 1 เลือกเป็นการส่งข้อมูลเพื่อแสดงผล : โดดไปทำงานที่ NextSend
SendCommandLCD:                     ‘ ส่งข้อมูลคำสั่ง
 LOW 10                             ‘ บิต RS = 0 เลือกเป็นการส่งข้อมูลคำสั่ง
NextSend:
 HIGH 15 : IF ASCII.BIT7 = 1 THEN NextSend1 : LOW 15        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.7 ให้ LCD
NextSend1:
 HIGH 14 : IF ASCII.BIT6 = 1 THEN NextSend2 : LOW 14        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.6 ให้ LCD
NextSend2:
 HIGH 13 : IF ASCII.BIT5 = 1 THEN NextSend3 : LOW 13        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.5 ให้ LCD
NextSend3:
 HIGH 12 : IF ASCII.BIT4 = 1 THEN NextSend4 : LOW 12        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.4 ให้ LCD
NextSend4:
 PULSOUT 11,1 : PAUSE 1                                     ‘ ส่งพัลส์ Enable ให้ LCD
 HIGH 15 : IF ASCII.BIT3 = 1 THEN NextSend5 : LOW 15        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.3 ให้ LCD
NextSend5:
 HIGH 14 : IF ASCII.BIT2 = 1 THEN NextSend6 : LOW 14        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.2 ให้ LCD
NextSend6:
 HIGH 13 : IF ASCII.BIT1 = 1 THEN NextSend7 : LOW 13        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.1 ให้ LCD
NextSend7:
 HIGH 12 : IF ASCII.BIT0 = 1 THEN NextSend8 : LOW 12        ‘ ส่งข้อมูล Logic 0 หรือ 1 ตามข้อมูลใน ASCII.0 ให้ LCD
NextSend8:
 PULSOUT 11,1 : PAUSE 1                                     ‘ ส่งพัลส์ Enable ให้ LCD
 RETURN
```

ภาคผนวก



ชุดคำสั่งต่างๆ ของเบสิกแสตมป์

FUNCTIONS AND OPERATOR

FUNCTION

PBASIC มีฟังก์ชั่นให้ใช้งานถึง 6 ฟังก์ชั่นแต่ไม่สามารถคำนวณค่าทศนิยมได้ โดยมีฟังก์ชั่นดังนี้

ABS เป็นคำสั่งที่ใช้ในการแปลงค่าตัวเลขแบบคิดเครื่องหมายเป็นแบบค่าสัมบูรณ์ (แบบไม่คิดเครื่องหมาย) เช่น

```
W1=-99
W1=ABS W1
DEBUG DEC?W1
“จะได้ค่า W1=99”
```

SQR เป็นคำสั่งที่ใช้ถอดรากที่ 2 (Square Root) เช่น

DEBUG DEC? SQR 100 “จะได้ SQR100=10

DCD เป็นคำสั่งเลือกเซตบิทที่ต้องการของข้อมูลขนาด 16 บิท

NCD เป็นคำสั่งแสดงบิทนัยสำคัญของข้อมูลขนาด 16 บิท

SIN เป็นคำสั่งที่ใช้หาค่า SIN โดยจะให้ค่าออกมาเป็น 2’S Complement เช่น

```
B0 = 45  (45 Degree)
DEBUG “SIN 45 is”,dec sin B0
```

COS เป็นคำสั่งที่ใช้หาค่า COS โดยจะให้ค่าออกมาเป็น 2’S Complement เช่น

```
B0 = 90 (90Degree)
DEBUG “COS 45 is”,dec cos B0
```

ตารางแสดงความหมายของ OPERATOR ต่างๆ

เครื่องหมาย	ความหมาย
+	การบวก
-	การลบ
*	การคูณแล้วให้ผลลัพธ์ 16 บิท
**	การคูณแล้วให้ผลลัพธ์ 16 บิท ที่ทางด้านสูง (Bit16-Bit31)
*/	คูณแล้วให้ผลลัพธ์ 16 บิท ตรงกลางของผลลัพธ์
/	การหารเอาเฉพาะจำนวนเต็ม
//	การหารเอาเศษ
MIN	กำหนดข้อมูลต่ำสุดของกลุ่มข้อมูลที่ต้องการตรวจสอบ
MAX	กำหนดข้อมูลสูงสุดของกลุ่มข้อมูลที่ต้องการตรวจสอบ

ตารางแสดงความหมายของ OPERATOR ต่างๆ (ต่อ)

เครื่องหมาย	ความหมาย
DIG	เรียกค่าของข้อมูลในหลักที่กำหนดในรูปของเลขฐานสิบ
<<	เลื่อนข้อมูลไปทางซ้าย 1 บิท
>>	เลื่อนข้อมูลไปทางขวา 1 บิท
REV	สลับบิทตามจำนวนหลักที่กำหนด
&	คำสั่งทางลอจิกที่กำหนดให้ทำการ AND
|	คำสั่งทางลอจิกที่กำหนดให้ทำการ OR
^	คำสั่งทางลอจิกที่กำหนดให้ทำการ EX-OR
~	คำสั่งทางลอจิกที่กำหนดให้ทำการ NOT หรือ Complement แบบบิทต่อบิท
=	เท่ากัน
<>	ไม่เท่ากัน
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าหรือเท่ากับ
<=	น้อยกว่าหรือเท่ากับ

รายละเอียดชุดคำสั่ง PBASIC ของ Basic Stamp BS2p อย่างย่อ

คำสั่งที่จะอธิบายในที่นี้เป็นแบบย่อๆ เท่านั้นหากต้องการรายละเอียดของชุดคำสั่งเพิ่มเติมสามารถดูได้จากโปรแกรม Basic Stamp Editor V1.32 ในส่วนของ Help Menu หรือ จากคู่มือที่เป็นไฟล์ PDF ในแผ่น CD ROM

AUXIO

เป็นคำสั่งที่ให้เปลี่ยนการควบคุมจากคำสั่งต่างๆ ที่อ้างถึงขา I/O ใน MAIN I/O เป็นขา AUX I/O ซึ่งมีเฉพาะใน BS2P เท่านั้น

ตัวอย่าง
```
HIGH 0      ' ขา P0 ของ MAIN I/O มีสถานะเป็น HIGH
AUXIO       ' คำสั่งต่างๆ จะเป็นการอ้างถึงขา AUX I/O
LOW 0       ' ขา A0 ของ AUX I/O มีสถานะเป็น LOW
```

BRANCH

BRANCH *offset, [address0, address1,… addressN]*

ใช้กระโดดไปยังแอดเดรสที่กำหนดตามค่า offset

- **Offset** คือ ตัวแปรหรือค่าคงที่ มีค่า 0-255 ที่ใช้ชี้ตำแหน่งที่จะกระโดดไปตามรายการที่กำหนดใน address โดยค่าของ address จะต้องอยู่ในช่วง 0-N

●**address** เป็นชื่อ Label ที่ใช้กำหนดตำแหน่งของ Address ที่จะกระโดดไป

ตัวอย่าง BRANCH B2[CAT,RAT,BAT] ‘ ถ้าB2=0 ไปที่ CAT
‘ ถ้าB2=1 ไปที่ RAT
‘ ถ้าB2=2 ไปที่ BAT

CAT: …….

RAT: …….

BAT: …….

ถ้าค่าข้อมูลไม่อยู่ในย่านที่กำหนดคือค่ารีจิสเตอร์ B2 มากกว่า 2 คำสั่ง BRANCH จะไม่ทำการใดๆ และโปรแกรมจะกระทำคำสั่งถัดไป

BUTTON

BUTTON ***pin,downState,delay,rate,byteVariable,targetState,address***

เป็นคำสั่งอ่านค่าของสวิทช์อินพุท สามารถกำหนดให้มีการป้องกันการกระเพื่อมอันเนื่องมาจากการกดสวิทช์หรือการ Bounce โดยคำสั่งนี้จะกำหนดให้เบสิกแสตมป์ สร้างสัญญาณการกดสวิทช์ซ้ำ (Auto - Repeat) และกระโดดไปยังตำแหน่งที่ระบุถ้าปุ่มที่กำหนดนั้นตรงกับค่าที่กำหนดไว้ที่ tagetstate โดยวงจรสร้างสัญญาณการกดคีย์นี้สามารถให้แอกทีฟหรือทำงานที่ลอจิก 0 หรือ 1 ก็ได้

● **pin** เป็นค่าตัวแปรหรือค่าคงที่มีค่า 0-15 เพื่อกำหนดตำแหน่งขาพอร์ทของเบสิกแสตมป์ให้เป็นขาอินพุทที่ต้องการใช้งาน

● **downState** เป็นค่าตัวแปรหรือค่าคงที่มีค่า 0 หรือ 1 ใช้สำหรับกำหนดสถานะลอจิกเมื่อสวิทช์ถูกกด ถ้าเป็น 0 หมายความว่าจะเกิดลอจิก 0 เมื่อมีการกดสวิทช์ และ ถ้าเป็น 1 หมายความว่า เมื่อมีการกดสวิทช์จะเกิดลอจิอ 1 ขึ้นที่ขาพอร์ทที่กำหนด

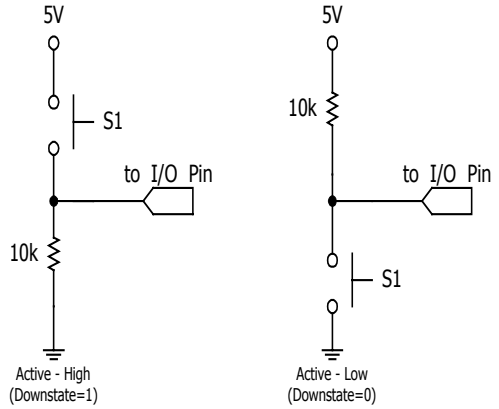
● **delay** เป็นตัวแปรหรือค่าคงที่มีค่า 0-255 เพื่อใช้กำหนดระยะห่างของช่วงเวลา ก่อนที่จะสร้างสัญญาณการกดสวิทช์ซ้ำ มีหน่วยเป็น มิลลิวินาทีถ้าค่า Delay = 0 จะไม่มีการแก้อาการ Bounce อันมาจากการกดสวิทช์ หรือไม่เกิด Debounce ขึ้น จะมีการสร้างสัญญาณการกดสวิทช์ซ้ำทันที ทำให้ค่าที่อ่าน อาจเกิดความผิดพลาดได้ ถ้าค่า Delay = 255 จะมีการแก้อาการ Bounce ของการกดสวิทช์ แต่จะไม่มีการสร้างสัญญาณการกดสวิทช์ซ้ำ

● **rate** เป็นค่าตัวแปรหรือค่าคงที่มีค่าตั้งแต่ 0-255 เพื่อกำหนดจำนวนรอบในการสร้างสัญญาณการกดสวิทช์ซ้ำ

● **byteVariable** เป็นตำแหน่งพื้นที่ทำงานของคำสั่ง BUTTON ค่านี้จะต้องให้เป็นศูนย์ก่อนใช้คำสั่ง BOTTON ในครั้งแรก

● **targetState** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0 หรือ 1 เพ่อใช้กำหนดเงื่อนไขของการกดสวิทช์ ถ้ามีค่าเป็น 0 หมายถึง ไม่มีการกดสวิทช์ และถ้ามีค่าเป็น 1 หมายถึงสวิทช์ถูกกด

● **Address** เป็น label ที่กำหนดตำแหน่งที่โปรแกรมจะกระโดดไป



ตัวอย่าง

```
Loop
BUTTON  7,1,255,250,0,0,Nopress     ‘กำหนดให้ไม่มีการ Debounce , ไม่มี Auto Repeat , BUTTON ทำงานที่
                                    ‘สภาวะ High
DEBUG  “*”
Goto LOOP
```

COUNT

COUNT ***pin,period,variable***

ใช้นับจำนวน Cycle ของสัญญาณอินพุทที่ขาพอร์ทของเบสิกแสตมป์ตามระยะเวลาที่กำหนด และเก็บค่าที่นับได้ไว้ในตัวแปร

- **pin** เป็นค่าตัวแปรหรือค่าคงที่ มีค่าตั้งแต่ 0-15 เพื่อกำหนดขาพอร์ทของเบสิกแสตมป์ที่ใช้งานเป็นขาอินพุท โดยการเขียนลอจิก 0 ไปยังบิตที่กำหนดภายในรีจิสเตอร์ DIRS
- **period** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 1-65,535 ใน BS2P มีหน่วยเป็น 0.287ms เพื่อใช้กำหนดช่วงเวลาในการนับ นั่นคือ ถ้า period = 10 ช่วงเวลาในการนับจะเป็น 10 x 0.287ms = 2.87 ms
- **variable** ปกติจะเป็นค่าตัวแปรแบบ Word จะใช้ในการเก็บค่าที่ได้จากในการนับ

ตัวอย่าง COUNT 0,1000,CAT ‘เป็นการนับ Cycle ในช่วงเวลา 287 ms แล้วเก็บไว้ในตัวแปร CAT

DEBUG

DEBUG ***outputData {,outputData…}***

ใช้ในการแสดงค่าตัวแปรและข้อความบนหน้าจอของคอมพิวเตอร์ด้วยโปรแกรมBasic Stamp Editor

- **outputData** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-65,535ส่วนประกอบของข้อความ,ตัวแปร,ค่าคงที่และตัวอักษรควบคุม

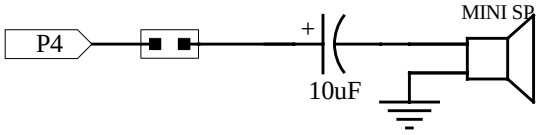
ตัวอย่าง DEBUG “ETT CO.,LTD.” ‘กำหนดให้แสดงข้อความที่หน้าต่าง debug เป็น ETT CO.,LTD.

DTMFOUT

DTMFOUT ***pin,{ontime,offtime,}{,tone..}***

เป็นคำสั่งสำหรับสร้างสัญญาณสองโทนหลายความถี่หรือ DTMF (Dual-tone multi-frequency) ซึ่งเป็นสัญญาณเสียงการกดปุ่มของเครื่องโทรศัพท์สมัยใหม่ที่เรียกว่าระบบโทน

- **Pin** เป็นค่าตัวแปรหรือค่าข้อมูลมีค่า 0-15 ใช้ระบุขาที่ต้องการใช้งานเป็นขาเอาท์พุทชั่วคราวเพื่อส่งสัญญาณ DTMF หลังจากการสร้างสัญญาณ DTMF แล้ว ขาที่ถูกกำหนดใช้นี้จะเปลี่ยนกลับเป็นขาอินพุทในทันที แม้ว่าก่อนหน้านี้จะถูกกำหนดเป็นขาเอาท์พุทไว้ก็ตาม
- **ontime** เป็นตัวแปรหรือค่าคงที่ กำหนดค่าได้ตั้งแต่ 0-65,535 ใช้ในการกำหนดระยะเวลาสร้างสัญญาณ DTMF ใน BS2P ค่านี้จะมีหน่วยเป็น 0.265 ms
- **offtime** เป็นตัวแปรหรือค่าคงที่ กำหนดค่าได้ตั้งแต่ 0-65,535 ใช้กำหนดช่วงเวลาหยุดหลังจากการสร้างสัญญาณ DTMF แล้ว มีหน่วยเป็น 0.265 ms
- **tone** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-15 ใช้สำหรับกำหนดค่าโทนของสัญญาณ DTMF ที่จะส่งออกไป โดยค่า 0-11 จะกำหนดให้สัญญาณเสียงที่ส่งออกไปนั้นตรงกับปุ่มโทรศัพท์มาตราฐาน นั่นคือค่า 0-9 คือปุ่มเลข 0-9 ค่า 10 คือปุ่ม * ค่า 11 คือปุ่ม # ส่วนค่า 12-15 จะเป็นสัญญาณโทน ซึ่งส่วนใหญ่ใช้อยู่ในเครื่องทดสอบโทรศัพท์และวิทยุรับส่ง



ตัวอย่าง DTMFOUT 4,[7,1,2,1,1,2,0] ‘สร้างสัญญาณโทรศัพท์หมายเลข 7121120 (ETT)

** หมายเหตุ ** ทั้ง ontimeและ offtime มีหน่วยเป็น 0.4 ms และถ้าไม่ใส่ ontimeจะเท่ากับ 200 ms และofftime จะเท่ากับ 50 ms

END

ใช้เมื่อต้องการจบโปรแกรม นำเบสิกแสตมป์เข้าสู่โหมดประหยัดพลังงาน

FOR…NEXT

FOR ***counter=startValue to endValue { step{-} stepValue} …. NEXT{counter}***

ใช้สำหรับสร้างลูปการทำงานของโปรแกรมซึ่งอยู่ระหว่างคำสั่ง For กับ Next การเพิ่มค่าหรือการลดค่าตัวแปร variable ขึ้นอยู่กับค่า StepVal โปรแกรมจะวนลูปซ้ำไปเรื่อยๆจนกว่าค่าที่เพิ่มหรือลดนั้นมีค่าเท่ากับค่า End

- **counter** ปกติจะเป็นตัวแปรแบบไบต์หรือเวิร์ด ใช้เป็นตัวนับการวนลูป
- **startValue** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-65,535 ใช้กำหนดค่าเริ่มต้นให้กับ counter
- **end** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-65,535 ใช้กำหนดค่าสิ้นสุดของ counter เมื่อค่าของ counter มีค่าเท่ากับค่า end การวนลูปของคำสั่ง For…Next จะสิ้นสุดและโปรแกรมจะไปทำงานในคำสั่งถัดไป

● **StepValue** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-65,535 ใช้กำหนดจำนวนของการเพิ่มขึ้นหรือลดลงของค่า counterถ้าค่า start มีค่ามากกว่า End เบสิกแสตมป์ จะเข้าใจว่าค่า stepValue มีค่าเป็นลบ แม้จะไม่มีเครื่องหมายลบระบุไว้ก็ตาม

ตัวอย่าง
```
Bat     VAR     BYTE
FOR Bat=0 to 65500  step 3000     ‘ แต่ละลูปให้ทำการบวกเพิ่มที่ละ 3000
DEBUG dec ? bat                   ‘ ที่หน้าต่าง Debug Terminalจะแสดงค่า bat
NEXT                              ‘ กระทำตามคำสั่งไปจนกระทั่งค่า Bat>65500
```

FREQOUT

FREQOUT ***pin,period,freq1,{freq2}***

ใช้สำหรับกำเนิดสัญญาณ Sine หนึ่งหรือสองสัญญาณในช่วงของคาบเวลาที่กำหนด

● **pin** เป็นค่าตัวแปรหรือค่าคงที่มีค่า 0-15 เพื่อกำหนดขา I/O ที่จะใช้งาน โดยขา I/O นี้จะได้รับการกำหนดให้เป็นขาเอาท์พุทเพื่อกำหนดสัญญาณ และจะกลับไปสถานะเดิมหลังจากกำเนิดสัญญาณแล้ว

● **period** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-65,535 ใช้กำหนดช่วงเวลาของสัญญาณที่กำเนิด ใน BS2P ค่านี้มีหน่วยเป็น 0.265 ms

● **freq1** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-32,767 ใช้เพื่อกำหนดค่าความถี่ของโทนเสียงแรก ใน BS2P ค่านี้จะมีหน่วยเป็น 3.77 Hz ดังนั้นจึงสามารถกำเนิดสัญญาณได้ตั้งแต่ 0 -123.531 kHz

● **freq2** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-32,767 ใช้เพื่อกำหนดค่าความถี่ของโทนเสียงที่ 2 ใน BS2P ค่านี้จะมีหน่วยเป็น 3.77 Hz ดังนั้นจึงสามารถกำเนิดสัญญาณได้ตั้งแต่ 0 -123.531 kHz โดยจะทำการผสมเสียงเข้าด้วยกันที่ขา I/O ที่เลือกใช้งาน

สำหรับการหยุดกำเนิดเสียงทำได้โดยกำหนดให้ค่าความถี่เท่ากับ 0

ตัวอย่าง
```
FREQOUT  13,1000,1000,1200   ‘สร้างสัญญาณ 3,770 Hz และ 4,524 Hz ผสมกันออกขา 13 นาน 265 ms
```

GET

GET ***location,variable***

เป็นคำสั่งอ่านค่า จาก Scratch Pad RAM ไปเก็บไว้ในตัวแปรที่กำหนด

● **location** เป็นตัวเป็นหรือค่าคงที่แสดงตำแหน่งที่จะอ่านค่าในหน่วยความจำ Scratch Pad RAM ในเบสิกแสตมป์ BS2P ค่านี้จะมีค่า 0-127 (128 Byte)

● **variable** โดยปกติแล้วเป็นตัวแปรแบบ Byte ที่ใช้เก็บข้อมูลที่อ่านได้

ตัวอย่าง
```
GET 25,Temp          ‘อ่านค่าจาก Scratch Pad RAM ที่ location 25 ไปเก็บไว้ที่ตัวแปร Temp
```

GOSUB

GOSUB *address*

เป็นคำสั่งใช้กำหนดให้เบสิกแสตมป์กระโดดไปทำงานยัง address ของโปรแกรมย่อยที่กำหนด และมีการเก็บค่าสุดท้ายไว้ใน Stack เมื่อสิ้นการทำงาน จะทำการเรียกค่าที่เก็บไว้ในแสต็กออกมา กลับไปทำงานยัง address นั้น

- **address** เป็นการระบุตำแหน่ง label ที่จะกระโดดไป

การกระทำตามคำสั่ง GOSUB นั้นเมื่อได้กระทำคำสั่งในโปรแกรมย่อยเสร็จสิ้นแล้ว บรรทัดสุดท้ายของโปรแกรมย่อยจะต้องจบด้วยคำสั่ง RETURN เสมอ ทั้งนี้เพื่อเป็นการกำหนดให้เบสิกแสตมป์ออกจากโปรแกรมย่อยแล้วกลับไปทำงานยังโปรแกรมหลักต่อไป

ตัวอย่าง

```
GOSUB  Hello     'เป็นคำสั่งที่กำหนดให้เบสิกแสตมป์กระโดดไปทำงานที่โปรแกรมย่อย Hello
```

GOTO

GOTO *address*

เป็นคำสั่งที่กำหนดให้เบสิกแสตมป์กระโดดตามจุดที่ต้องการ โดยจะกำหนดเป็น address

- **address** คือชื่อโปรแกรมหรือตำแหน่ง LABEL ที่จะกระโดดไปทำงาน

ตัวอย่าง

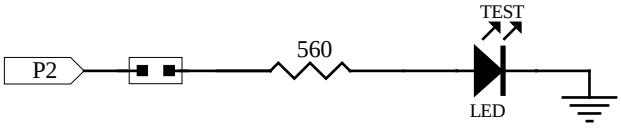
```
LOOP:
        DEBUG  "ETT CO.,LTD."     ' กำหนดข้อความที่จะแสดงที่หน้าต่าง Debug Terminal
        GOTO  LOOP                ' กระโดดไปทำงานที่ โปรแกรม LOOP
```

HIGH

HIGH *pin*

ใช้สั่งให้ขา I/O ที่ระบุใช้งานมีสถานะเป็น HIGH คือมีลอจิกเป็น "1"

- **pin** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-15 ใช้กำหนดขา I/O ที่ต้องการใช้งาน โดยจะถูกเซตให้เป็นสถานะ HIGH ดังตัวอย่างต่อไปนี้



ตัวอย่าง

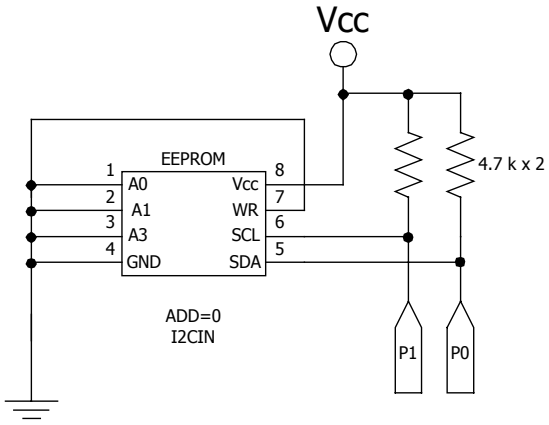
```
HIGH  2          ' ทำให้ขา 2 มีสถานะเป็น HIGH หรือมีลอจิก "1" ทำให้ LED ติด
```

I2CIN

I2CIN ***pin,slaveID,Address{\LOW Address},[InputData]***

เป็นคำสั่งที่ใช้รับข้อมูลจากการสื่อสารข้อมูลด้วยระบบบัส I^2C ซึ่งเป็นการสื่อสารแบบ 2 ทิศทาง

- **pin** เป็นตัวแปรหรือค่าคงที่มีค่า 0 หรือ 8 แสดงขา I/O ที่เลือกใช้ โดยเมื่อเราทำการเลือกขา 0 ขา 0 นี้จะทำหน้าที่เป็นขา SDA และจะมีขา 1 เป็นขา SCL แต่ถ้าหากทำการเลือกขา 8 ขา 8 นี้จะทำหน้าที่เป็นขา SDA และจะมีขา 9 เป็นขา SCL เสมอ



- **slaveID** เป็นค่าคงที่หรือตัวแปรที่มีค่าตั้งแต่ 0 – 255 แสดงให้เห็นถึงเลขประจำตัว (ID)เฉพาะของไอซี
- **Address** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-255 เป็นการอ้างถึงตำแหน่ง Address ภายใน IC ที่จะใช้ในการรับข้อมูล
- **LOW Address** เป็นตัวแปรหรือค่าคงที่ที่มีค่า 0 – 255 แสดงถึง Low-Byte ภายใน IC ที่จะใช้รับข้อมูล โดยจะต้องทำการอ้างถึง Address ที่ต้องการจะใช้ด้วย
- **InputData** เป็นรายการของตัวแปรที่บอกการกระทำของ I2CIN กับข้อมูลที่เข้ามา โดยจะสามารถเก็บข้อมูลในรูปของ Variable,Array,เลขฐาน 2,เลขฐาน10,เลขฐาน 16

การใช้งานระบบบัส I^2C นั้น จะใช้ขา I/O 2 ขาในการสื่อสารข้อมูล โดยขาแรกจะใช้สำหรับเชื่อมต่อกับขา SDA ของไอซี และขาที่สองจะใช้เชื่อมต่อกับขา SCL ของไอซี ในระหว่างที่กระทำตามคำสั่งอยู่นั้นขาทั้ง 2 นี้ จะทำงานเป็นอินพุทและเอาท์พุทสลับไปมา และจะถูกเซตเป็นอินพุทเมื่อเสร็จสิ้นการกระทำตามคำสั่ง

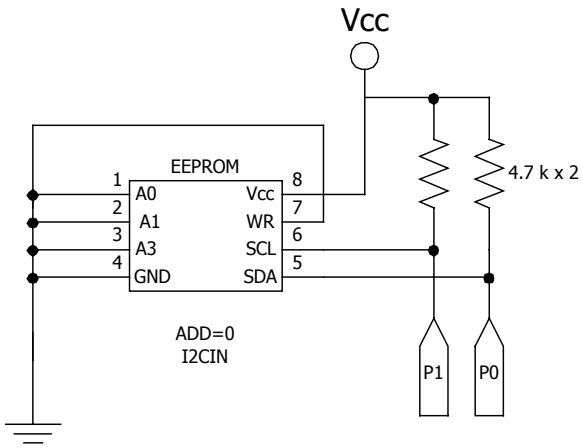
```
ตัวอย่าง        I2CIN   0,$A1,0,[Result]    ' รับค่าข้อมูลจากการใช้ I2C โดยใช้ขา 0 เป็น SDA และขา 1 เป็น SCL
                                            ' มีหมายเลข ID = $A1 ใช้ Address 0 ในการรับข้อมูล เก็บค่าที่ได้ใน
                                            ' ตัวแปร Result
```

I2COUT

I2COUT ***pin,slaveID,Address{\LOW Address},[OutputData]***

เป็นคำสั่งที่ใช้ส่งข้อมูลให้กับการสื่อสารข้อมูลด้วยระบบบัส I^2C ซึ่งเป็นการสื่อสารแบบ 2 ทิศทาง

- **pin** เป็นตัวแปรหรือค่าคงที่มีค่า 0 หรือ 8 แสดงขา I/O ที่เลือกใช้ โดยเมื่อเราทำการเลือกขา 0 ขา 0 นี้จะทำหน้าที่เป็นขา SDA และจะมีขา 1 เป็นขา SCL แต่ถ้าหากทำการเลือกขา 8 ขา 8 นี้จะทำหน้าที่เป็นขา SDA และจะมีขา 9 เป็นขา SCL เสมอ
- **slaveID** เป็นค่าคงที่หรือตัวแปรที่มีค่าตั้งแต่ 0 - 255 แสดงให้เห็นถึงเลขประจำตัว (ID)เฉพาะของไอซี



- **Address** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-255 เป็นการอ้างถึงตำแหน่ง Address ภายใน IC ที่จะใช้ในการส่งข้อมูล
- **LOW Address** เป็นตัวแปรหรือค่าคงที่ที่มีค่า 0 – 255 แสดงถึง Low-Byte ภายใน IC ที่จะใช้ส่งข้อมูล โดยจะต้องทำการอ้างถึง Address ที่ต้องการจะใช้ด้วย
- **OutputData** เป็นรายการของตัวแปรหรือค่าคงที่ ที่บอกถึงรูปแบบของข้อมูลที่จะทำการส่งออกไป I2COUT สามารถส่งผ่านได้เฉพาะไบต์หรือไบต์ซ้ำ ๆ กัน และทำการแปลงค่าในระบบเลขฐาน 10 ,16 หรือฐาน 2

การใช้งานระบบบัส I^2C นั้น จะใช้ขา I/O 2 ขาในการสื่อสารข้อมูล โดยขาแรกจะใช้สำหรับเชื่อมต่อกับขา SDA ของไอซี และขาที่สองจะใช้เชื่อมต่อกับขา SCL ของไอซี ในระหว่างที่กระทำตามคำสั่งอยู่นั้นขาทั้ง 2 นี้ จะทำงานเป็นอินพุทและเอาท์พุทสลับไปมา และจะถูกเซตเป็นอินพุทเมื่อเสร็จสิ้นการกระทำตามคำสั่ง

```
ตัวอย่าง        I2COUT  0,$A0,5,[100]      ' จะทำการเขียนคำสั่งไปที่ IC โดยใช้ขา 0 เป็น SDA และ
                                           ' ขา 1 เป็น SCL มีหมายเลข ID = $A0 ส่งเลข 100ไปที่Address 5
```

IF…THEN

IF ***condition*** **THEN** ***address***

เป็นคำสั่งที่ใช้ในการตรวจสอบเงื่อนไข และถ้าเงื่อนไขเป็นจริง ก็จะทำการกระโดดไปทำงานยังตำแหน่ง address ที่ระบุเอาไว้

- **condition** เป็นการกำหนดเงื่อนไข เพื่อตรวจสอบสมการว่าเป็นจริงหรือเท็จ กำหนดสมการได้เช่น "X=4"
- **address** เป็นตำแหน่งที่ label จะกระโดดไปถ้าเงื่อนไขเป็นจริง

ข้อมูลที่นำมาเปรียบเทียบสามารถใช้ได้กับข้อมูลทุกชนิดทั้งระดับบิท , ไบต์ , เวิร์ดหรือข้อมูลที่เป็นค่าคงที่ การเปรียบเทียบข้อมูลทั้งหมดจะใช้ชนิดข้อมูลเป็นแบบ 16 บิทไม่คิดเครื่องหมาย

ตัวอย่าง
```
IF  Value <4000   THEN  Loop     ' ถ้าค่า Value น้อยกว่า 4,000 จริงให้กระโดดไปทำงานที่ตำแหน่ง Loop
```

การใช้คำสั่ง IF….THEN ประมวลผลทางลอจิก

กรณีกระทำการ NOT เป็นการกลับเงื่อนไข เปลี่ยนจากจริงเป็นเท็จ หรือเท็จเป็นจริง

ตัวอย่าง
```
IF  NOT A = 100  THEN  NOT_EQUAL     ' ถ้า A ไม่เท่ากับ 100 ให้กระโดดไปทำงานที่ตำแหน่ง NOT_EQUAL
```

กรณีที่กระทำการ AND จะเป็นจริงเมื่อเงื่อนไขทั้งคู่เป็นจริง

ตัวอย่าง
```
            A = 5
            B = 10
            IF   A = 5 AND  B = 10 THEN TRUE      ' ถ้า A และ B มีเงื่อนไขเป็นจริงทั้งคู่ ให้กระโดดไปทำงาน
                                                  ' ที่ตำแหน่ง TRUE
            DEBUG "NOT TRUE    ' ถ้าเงื่อนไขตัวใดตัวหนึ่งหรือทั้งหมดไม่เป็นจริงแสดงข้อความ NOT  TRUE
                               ' ที่หน้าต่าง Debug Terminal
            STOP
TRUE:       DEBUG "TRUE"       ' ถ้าเงื่อนไขเป็นจริงทั้งคู่ให้แสดงข้อความ TRUE ที่หน้าต่าง Debug Terminal
            STOP
```

กรณีกระทำการ OR จะได้ผลเป็นจริงเมื่อเงื่อนไขที่กำหนดตัวใดตัวหนึ่งเป็นจริง

กรณีกระทำตามคำสั่ง XOR จะได้ผลเป็นจริงเมื่อมีเงื่อนไขใดเงื่อนไขหนึ่งเป็นจริงเท่านั้น ทดลองเปลี่ยนตัวอย่างคำสั่งด้านบนจาก AND เป็น OR และ XOR

INPUT

INPUT *pin*

เป็นคำสั่งที่ทำให้ขาที่ระบุใช้งานมีสถานะเป็นขาอินพุท

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 ใช้เลือกขา I/O ที่ต้องการจะใช้งาน โดยจะถูกเซตให้เป็นโหมดอินพุท

ตัวอย่าง
```
INPUT   7        ' กำหนดให้ขา7 เป็นขาอินพุท
```

IOTERM

IOTERM port

เป็นคำสั่งที่ใช้เปลี่ยนการควบคุมของขา MAIN I/O หรือ AUX I/O

- **port** เป็นตัวแปรหรือค่าคงที่ที่มีค่า 0-1 ซึ่งเป็น I/O Port ที่ใช้ (MAIN I/O = 0,AUXILIARY I/O = 1)

ตัวอย่าง

```
HIGH 2        ‘ กำหนดขา P2 ของ MAIN I/O มีสถานะเป็น High
IOTERM 1      ‘ ให้คำสั่งต่อไปทั้งหมด กระทำกับขา AUX I/O
LOW 2         ‘ กำหนดขา A2 ของ AUX I/O มีสถานะเป็น LOW
```

LCDCMD

LCDCMD *pin,command*

ใช้ส่งคำสั่งไปที่ LCD Display

- **pin** เป็นตัวแปรหรือค่าคงที่ ที่มีค่า 0-1 หรือ 8-9 ซึ่งเป็นขา I/O ที่ใช้
- **command** เป็นตัวแปรหรือค่าคงที่ ที่มีค่า 0-255 ซึ่งชี้ให้เห็นการส่งคำสั่ง LCD

ตัวอย่าง

```
LCDCMD 1,16     ‘ กำหนดให้ขา 1 เป็นขา I/O ที่ใช้ และตัวอักษรตัวแรกจะทำการเลื่อนไปทางซ้าย
```

LCDIN

LCDIN *pin,command,[InputData]*

รับข้อมูลจากจอแสดงผล LCD

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่าระหว่าง 0-1 หรือ 8-9 ซึ่งเป็นขา I/O ที่ใช้
- **command** เป็นตัวแปรหรือค่าคงที่ ที่มีค่าระหว่าง 0-255 แสดงให้เห็นการส่งคำสั่งให้ LCD
- **InputData** แสดงรายการตัวแปรหรือค่าคงที่และบอกรูปแบบของข้อมูลที่เข้ามา

ตัวอย่าง

```
Char  VAR  BYTE
LCDIN  1,128,[Char]
```

LCDOUT

LCDOUT *pin,command,[OutputData]*

ส่งข้อมูลไปแสดงผลที่ LCD

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่าระหว่าง 0-1 หรือ 8-9 ซึ่งเป็นขา I/O ที่ใช้
- **command** เป็นตัวแปรหรือค่าคงที่ ที่มีค่าระหว่าง 0-255 แสดงให้เห็นการส่งคำสั่งให้ LCD
- **InputData** แสดงรายการตัวแปรหรือค่าคงที่และบอกรูปแบบของข้อมูลที่ส่งออกไป

ตัวอย่าง

```
LCDOUT  1,128,[“Hello  world”]
```

LOOKDOWN

LOOKDOWN ***target,{comparisionOp,}[value0,value1,…valueN],resultVariable***

เป็นคำสั่งที่เก็บค่า INDEX ของตัวเลขที่นำมาเปรียบเทียบกับค่าเป้าหมาย แล้วผลของการเปรียบเทียบเป็นจริง

- **target** เป็นค่าตัวแปรหรือค่าคงที่สำหรับเปรียบเทียบกับค่าที่อยู่ใน value0…. valueN
- **comparisionOp** เป็นเครื่องหมายในการเปรียบเทียบ(ถ้าไม่ใส่จะถือว่าเป็น "=")

  = equal

  <> not equal

  > greater than

  < less than

  >= greater than or equal to

  <= less than or equal to

- **value0,value1…** รายชื่อของค่าข้อมูลที่กำหนดซึ่งเป็นตัวแปรหรือค่าคงที่ ที่มีขนาด 16 บิท
- **resultVariable** ตัวแปรที่ใช้เก็บค่า index หากการเปรียบเทียบพบค่าที่ตรงกัน

รายละเอียดเพิ่มเติม

คำสั่ง LOOKDOWN ทำงานเหมือนสารบัญในหนังสือ ซึ่งสามารถค้นหาเรื่องราวที่ต้องการในหนังสือจากสารบัญ คำสั่ง LOOKDOWN จะทำการเปรียบเทียบกลุ่มข้อมูลในรายการ จากนั้นจะเก็บค่าตัวเลขที่ชี้ไปยังรายการที่ตรวจสอบพบว่า มีค่าตรงตามเงื่อนไขของการเปรียบเทียบ

ตัวอย่าง

```
VALUE  =  20
RESULT = 15
LOOKDOWN  VALUE,[5,10,20,25,30,35,40],RESULT
                ‘  ทำการเปรียบเทียบรายการข้อมูลกับค่า VALUE แล้วเก็บค่า INDEX ไว้ที่ตัวแปร Result

DEBUG   “VALUE MATCHES ITEM “,DEC RESULT,” IN LIST”
```

เมื่อทำการ RUN โปรแกรมนี้ หน้าต่าง Debug Terminal จะแสดงข้อความ VALUE MATCHES ITEM 2 IN LIST เนื่องจากค่า VALUE ที่เท่ากับ 20 มีค่าตรงกับข้อมูลตัวที่ 2

LOOKUP

LOOKUP ***index,[value0, value1,…valueN], resultVariable***

เป็นคำสั่งที่ใช้เปิดตารางข้อมูล โดยคำสั่ง LOOKUP จะมองหาค่าที่ชี้โดย INDEX และนำค่าที่ได้ไปเก็บไว้ใน ResultVariable ถ้าค่า INDEX มีค่ามากกว่าค่าที่อยู่ในรายการของ VALUE ค่า ResultVariable จะไม่เกิดการเปลี่ยนแปลง

- **index** เป็นค่าตัวแปรหรือค่าคงที่เพื่อกำหนดเป็น INDEX ของ VALUE ที่ต้องการหาในรายการ
- **value0,value1 ….** เป็นค่าตัวแปรหรือค่าคงที่สำหรับสร้างรายการข้อมูล มีขนาดสูงสุดได้ถึง 16 บิท
- **resultVariable** ตัวแปรที่ใช้เก็บ VALUE ที่มีการตรวจพบ

ตัวอย่าง
```
INDEX  = 5
RESULT = 255
LOOKUP INDEX,[5,10,20,25,30,35,40],RESULT
        ' มองหาค่าที่ชี้โดย INDEX และนำค่าที่ ได้ไปเก็บไว้ในตัวแปร Result
DEBUG " ITEM ",DEC INDEX," IS ", DEC RESULT
```

เมื่อทำการ RUN โปรแกรมนี้ หน้าต่าง Debug Terminal จะแสดงข้อความ “ITEM 5 IS: 35” เนื่องจากค่า INDEX ตัวที่ 5 มีค่าเท่ากับ 35

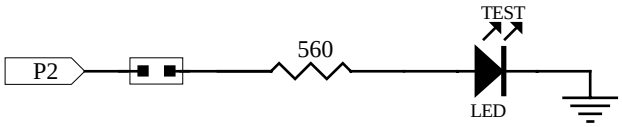
LOW

LOW *pin*

เป็นการทำให้ขา I/O ที่เลือกใช้ มีสถานะเป็น LOW หรือ ลอจิก “0”

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 ใช้เลือกขาที่ต้องการใช้งาน โดยจะถูกเซตเป็น LOW

ตัวอย่าง
```
LOW 2        ' กำหนดให้ ขา 2 มีสถานะเป็น LOW
```



MAINIO

เป็นคำสั่งที่ให้เปลี่ยนการควบคุมจากคำสั่งต่างๆ ที่อ้างถึงขา I/O ใน AUX I/O เป็นขา MAIN I/O ซึ่งมีเฉพาะใน BS2P เท่านั้น

ตัวอย่าง
```
AUXIO        ' คำสั่งต่างๆ จะเป็นการอ้างถึงขา AUX I/O
HIGH 2       ' ขา A2 ของ AUX I/O มีสถานะเป็น HIGH
MAINIO       ' คำสั่งต่างๆ จะเป็นการอ้างถึงขา MAIN I/O
LOW 2        ' ขา A2 ของ MAIN I/O มีสถานะเป็น LOW
```

NAP

NAP *period*

เป็นคำสั่งที่ให้เบสิกแสตมป์ เข้าสู่โหมดประหยัดพลังงาน หรือ Sleep Mode ในช่วงเวลาสั้นๆ ซึ่งจะทำให้เบสิกแสตมป์กินกระแสลดลงเหลือประมาณ 50 μA ในขณะที่ไม่มีการขับโหลด

- **period** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-7 ซึ่งใช้กำหนดคาบเวลาในขณะที่เข้าสู่ Sleep Mode โดยค่าคาบเวลาของการเข้าสู่โหมดประหยัดพลังงานจะหาได้จาก $2^{period} \times 18$ ms

Period	2^period	Length of Nap
0	1	18 ms
1	2	36 ms
2	4	72 ms
3	8	144 ms
4	16	288 ms
5	32	576 ms
6	64	1152 ms (1.152 seconds)
7	128	2304 ms (2.304 seconds)

หากใช้คำสั่ง NAP ในขณะที่มีการขับโหลดอยู่ กระแสที่ไหลผ่านจะหยุดชะงักในระยะเวลาสั้นๆ 18 ms โดยประมาณ

ตัวอย่าง
```
NAP  4        ' กำหนดให้เบสิกแสตมป์เข้าสู่โหมดประหยัดพลังงานในช่วงเวลา 288 ms
```

OUTPUT

OUTPUT ***pin***

เป็นคำสั่งที่ทำให้ขา I/O ที่ใช้นั้นเป็นขา Output (เป็นการเขียน “1” ไปที่ DIRS ของ pin นั้นๆ)

- **pin** เป็นค่าตัวแปรหรือค่าคงที่มีค่า 0-15 ใช้เลือกขา I/O ของเบสิกแสตมป์ที่ต้องการใช้งานเป็นขาเอาท์พุท

หลังจากที่ขา I/O ของเบสิกแสตมป์ได้ถูกกำหนดให้เป็นขาเอาท์พุทแล้ว เราสามารถเขียนโปรแกรมเพื่อกำหนดให้ขา I/O ของเบสิกแสตมป์ที่ติดต่ออยู่นั้นมีสถานะเป็น 0 หรือ 1 ก็ได้ เช่น

ตัวอย่าง
```
OUTPUT  1     ' กำหนดให้ขา 1 เป็นขา OUTPUT
OUT1 = 1      ' ทำให้ขา 1 มีสถานะเป็น HIGH หรือมีลอจิก 1
```

OWIN

OWIN ***pin,mode,[InputData]***

รับข้อมูลที่เกิดจากการใช้การสื่อสารของระบบบัส 1-Wire คำสั่งนี้มีใน BS2P เท่านั้น

- **pin** เป็นตัวแปรหรือค่าคงที่ ที่มีค่า 0-15 ซึ่งแสดงให้เห็นขา I/O ที่ใช้ โดยการสื่อสารของระบบบัส 1-Wire จะใช้ขา I/O ในการติดต่อเพียงขาเดียวเท่านั้นเรียกว่าขา DQ
- **mode** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 แสดงให้เห็นโหมดของการส่งข้อมูล แสดงรายละเอียดของโหมดต่างๆ ตามตารางต่อไปนี้

Mode	Effect
0	No Reset , Byte Mode , Low Speed
1	Reset Before data , Byte Mode , Low Speed
2	Reset after data , Byte Mode , Low Speed
3	Reset Before and after data ,Byte Mode , Low Speed
4	No Reset , Bit Mode , Low Speed
5	Reset before data , Bit mode , low speed
8	No Reset , Byte mode , High speed
9	Reset before data , Byte mode , High Speed

- **InputData** เป็นรายการของตัวแปร และบอกถึงการเปลี่ยนแปลงที่กระทำกับข้อมูลที่รับเข้ามา

ตัวอย่าง
```
Result          VAR     Byte
OWIN  0,1[Value]
        ‘ จะทำการส่ง Reset Pulse ไปที่ 1-Wire โดยเชื่อมต่อกับขา 1 และรับค่าที่ ได้ไปเก็บในตัวแปร Value
```

OWOUT

OWOUT ***pin,mode,[OutputData]***

ส่งข้อมูลเข้าสู่การสื่อสารของระบบบัส 1-Wire คำสั่งนี้มีใน BS2P เท่านั้น

- **pin** เป็นตัวแปรหรือค่าคงที่ ที่มีค่า 0-15 ซึ่งแสดงให้เห็นขา I/O ที่ใช้ โดยการสื่อสารของระบบบัส 1-Wire จะใช้ขา I/O ในการติดต่อเพียงขาเดียวเท่านั้นเรียกว่าขา DQ
- **mode** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 แสดงให้เห็นถึงโหมดของการถ่ายโอนข้อมูล
- **OutputData** เป็นรายการของตัวแปร และบอกถึงการเปลี่ยนแปลงของข้อมูลที่ออกไป

ตัวอย่าง
```
Value = 65
OWOUT  0,1,[DEC Value]
‘ ส่งค่า Value ที่เป็นตัวอักขระเลขฐานสิบคือ 6 กับ 5 เข้าสู่การสื่อสารของระบบบัส 1-Wire
```

PAUSE

PAUSE ***millisecond***

เป็นคำสั่งหน่วงเวลาเพื่อหยุดการทำงานของโปรแกรมชั่วคราวตามช่วงเวลาที่ระบุ

- **millisecond** เป็นตัวแปรหรือค่าคงที่ ใช้สำหรับกำหนดค่าช่วงเวลาที่จะหยุดการทำงานของโปรแกรมชั่วคราว มีหน่วยเป็น ms สามารถกำหนดช่วงเวลาได้นานสูงสุด 65,535 ms

ตัวอย่าง

```
PAUSE 1000          ' เป็นการหยุดการทำงานของโปรแกรมชั่วคราวเป็นเวลา 1 วินาที
```

POLLIN

POLLIN ***pin,state***

เป็นคำสั่งที่ระบุถึงขาที่จะได้รับเลือกเป็นขาอินพุท และเลือกสถานะ Active

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 ซึ่งแสดงขา I/O ที่ถูกเลือกใช้เป็นขาอินพุท
- **state** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-1 แสดงสถานะของขา I/O โดยเป็น LOW (0) หรือ HIGH (1)

ตัวอย่าง

```
POLLIN  0,1         ' ทำการเลือกขา 0 เป็นขาอินพุท โดยมีการ Active ที่สถานะ HIGH (1)
```

POLLMODE

POLLMODE ***mode***

ระบุโหมดที่จะใช้ในคำสั่ง Polled

- **mode** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 แสดงให้เห็นถึงการดำเนินการของคำสั่ง Polled โดยแสดงรายละเอียดของโหมดต่างได้ดังตารางต่อไปนี้

Mode	Effect
0	Deactivate polling,clear polled-input and output configuration.
1	Deactivate polling,save polled-input and output configuration.
2	Activate polling with polled-output action (and polled-wait) only.
3	Activate polling with polled-run action only.
4	Activate polling with polled-output /polled wait and polled run action.
5	Clear polled-input configuration.
6	Clear polled-output configuration.
7	Clear polled-input configuration and output configuration.
8-15	Same 0-7 except polled – output states are latched.

ตัวอย่าง

```
POLLIN 0,1          ' กำหนดให้ ขา 0 เป็นขาอินพุท และActive ที่สถานะ HIGH
POLLOUT 1,0         ' กำหนดให้ ขา 1 เป็นขาเอาท์พุท และActive ที่สถานะ LOW
POLLMODE  2         ' เลือกโหมด 2 เป็นการเลือกสถานะของ POLLOUT เท่านั้น ในที่นี้คือสถานะ LOW (0)
```

POLLOUT

POLLOUT ***pin,state***

ระบุถึงขาที่ได้ถูกเลือกเป็นขาเอาท์พุท และสถานะของการ Active

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 ซึ่งแสดงขา I/O ที่ถูกเลือกใช้เป็นขาเอาท์พุท
- **state** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-1 แสดงสถานะของขา I/O โดยเป็น LOW (0) หรือ HIGH (1)

ตัวอย่าง
```
POLLOUT 1,0          ‘ กำหนดให้ขา 1 เป็นขาเอาท์พุทและActive ที่สถานะเป็น LOW
```

POLLRUN

POLLRUN ***slot***

ระบุถึงโปรแกรมที่ใช้ในการ Run บนขาอินพุทที่ถูกเลือก

- **slot** เป็นตัวแปรหรือค่าคงที่ มีค่าตั้งแต่ 0-7 เป็นรายละเอียดของโปรแกรม Slot ที่จะทำการ Run

ตัวอย่าง
```
POLLIN 0,1           ‘ กำหนดให้ขา 1 เป็นขาอินพุทและActive ที่สถานะเป็น HIGH
POLLRUN  1           ‘ ทำการ Run โปรแกรม Slot 1
POLLMODE 3           ‘ เลือกโหมดที่ 3
```

POLLWAIT

POLLWAIT ***period***

หยุดการปฏิบัติโปรแกรมให้อยู่ในโหมด Low - Power ในหน่วยของคาบเวลาจนกระทั่งขา Input ที่ถูกเลือกไปถึงสถานะที่ต้องการ

- **period** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-8 เป็นรายละเอียดของสถานะ Low-Power ซึ่งระบุไว้ตามตารางต่อไปนี้

Period	Length of Low-Power Mode
0	18 ms
1	36 ms
2	72 ms
3	144 ms
4	288ms
5	576ms
6	1.152 s
7	2.304 s
8	No Power-Down

```
ตัวอย่าง    POLLIN 0,1          ‘ กำหนดให้ขา 1 เป็นขาอินพุทและActive ที่สถานะเป็น HIGH
      Loop:
            POLLWAIT 0          ‘ กำหนดคาบเวลา 0 ซึ่งจะมีความยาวของโหมด Low-Power18 ms
                                หมายถึงทุกๆ 18 ms เบสิกแสตมป์จะทำการเช็คสถานะของขา 0 ที่มีสถานะ HIGH
                                ถ้ามีสถานะเป็น HIGH ก็จะทำการ Run โปรแกรมในบรรทัดต่อไป แต่ถ้าเป็น LOW
                                ก็จะกลับไปทำงานใน Sleep Mode อีก 18ms

            TOGGLE 1            ‘ ทำการกลับสถานะของขา 1
            GOTO  Loop
```

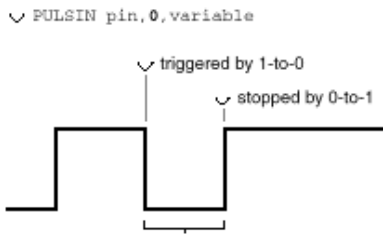
PULSIN

PULSIN *pin,state,resultVariable*

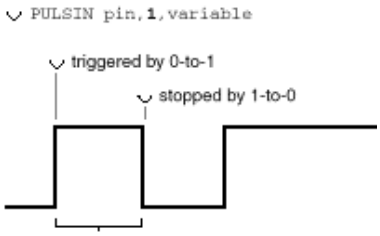
เป็นคำสั่งที่ใช้ในการวัดความกว้างของสัญญาณ Pulse ที่ป้อนเข้ามายังขา I/O ของเบสิกแสตมป์ และเก็บค่าไว้ในตัวแปร

- **pin** เป็นค่าตัวแปรหรือค่าคงที่ที่ มีค่า0-15 ใช้ระบุตำแหน่งขา I/O ของเบสิกแสตมป์ที่ต้องการใช้สำหรับการนับจำนวนพัลส์ โดยขา I/O ที่ระบุจะเข้าสู่โหมดอินพุทโดยอัติโนมัติและเมื่อกระทำตามคำสั่งเสร็จเรียบร้อยก็จะคืนกลับสู่สถานะเดิม
- **state** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0 หรือ 1 ใช้สำหรับกำหนดค่าลอจิกเริ่มต้นในการวัด ถ้ากำหนดค่า state = 0 (LOW) จะกำหนดให้นับเมื่อพัลส์เปลี่ยนจากลอจิก 1 เป็น 0 ถ้าค่า state = 1 (HIGH) จะกำหนดให้นับเมื่อพัลส์เปลี่ยนจากลอจิก 0 เป็น 1
- **Variable** เป็นตัวแปรที่ใช้เก็บผลลัพธ์ของจำนวนพัลส์ที่นับได้ ใน BS2P ค่านี้จะมีหน่วยเป็น 0.75μs

คำสั่ง PLUSIN สามารถวัดค่าของสัญญาณพัลส์ความถี่สูงสุดได้ไม่เกิน 625 kHz ที่ค่าของ resultVariable = 1 และ วัดความถี่ต่ำสุดได้ 9.537 Hz ที่ค่าของ resultVariable = 65,535



measured in 0.75 μs units
and stored in variable



measured in 0.75 μs units
and stored in variable

shows how the state bit controls triggering of Pulsin.

ตัวอย่าง PULSIN 10,1,B0 ‘ วัดความกว้างของ pulse ที่มีสถานะ HIGH ที่ขา 10
แล้วเก็บค่าที่ได้ไว้ในตัวแปร B0

PULSOUT

PULSOUT ***pin,period***

เป็นคำสั่งที่ใช้กำเนิดสัญญาณพัลส์ที่ขา I/O ใดๆ ที่เลือกใช้ กับความกว้างของคาบเวลา

- **pin** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-15 เพื่อเลือกขา I/O ของเบสิกแสตมป์ ซึ่งขา I/O นี้จะถูกกำหนดให้เป็นเอาท์พุทก่อน และกลับสถานะเดิมหลังจากกระทำตามคำสั่ง PULSOUT เสร็จสิ้น
- **period** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-65,535 ใช้กำหนดคาบเวลาของสัญญาณพัลส์ ในเบสิกแสตมป์ BS2P จะมีค่าหน่วยของคาบเวลาเป็น 1.18 μS

ตัวอย่าง PULSOUT 7,5000 ‘ ส่งพัลส์ 5900 μS ไปที่ขา 7

PUT

PUT ***location,value***

เป็นคำสั่งสำหรับนำค่า data ในตัวแปรหรือค่าคงที่ไปเก็บไว้ในหน่วยความจำ Scratch Pad RAM

- **location** เป็นตัวแปรหรือค่าคงที่แสดงตำแหน่งของ หน่วยความจำ Scratch Pad RAM ภายในเบสิกแสตมป์ BS2P มีค่า 0-127
- **value** เป็นตัวแปรหรือค่าคงที่ ที่ต้องการเก็บไว้ในหน่วยความจำ Scratch Pad RAM มีค่า 0-255

ตัวอย่าง PUT 3,100 ‘ จะนำค่า 100 ไปเก็บไว้ที่ Location 3 ของหน่วยความจำ Scratch Pad RAM

PWM

PWM ***pin, duty,cycles***

เป็นคำสั่งที่ใช้สำหรับเปลี่ยนข้อมูลแบบดิจิตอลเป็นอนาล็อกผ่านการ PWM

- **pin** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-15 เป็นขา I/O ของเบสิกแสตมป์ที่ถูกเลือกใช้งาน โดยกำหนดให้เป็นขา OUTPUT ก่อนที่จะสร้างพัลส์ และจะกลับเป็นขา INPUT เมื่อกระทำตามคำสั่งแล้ว
- **duty** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-255 ใช้กำหนดระดับอนาล็อกทางเอาท์พุท (0-5 V)
- **cycles** เป็นค่าที่กำหนดค่าคาบเวลา มีค่า 0-255 ค่าของสัญญาณ PWM ใน BS2P จะมีหน่วยเป็น 0.625 ms สามารถหาค่าแรงดันที่เกิดขึ้นได้จากการกำหนดค่าดิวตี้ไซเคิลได้จากสูตร Vout = (duty/255) x 5V

ตัวอย่าง PWM 0,100,10 เปลี่ยนข้อมูลดิจิตอลเป็นอนาล็อกผ่านการ PWM ที่ขา 0 ที่ค่า duty 100
และกำหนดค่าคาบเวลาโดยประมาณของสัญญาณ PWM ที่ 6.52 ms

RANDOM

RANDOM ***variable***

เป็นคำสั่งจำลองการสุ่มตัวเลข

- **variable** เป็นค่าตัวแปรแบบ Byte หรือ Word ใช้กำหนดจำนวนสูงสุดของการสุ่มตัวเลข

คำสั่ง RANDOM จะทำการจำลองการสุ่มตัวเลข มีค่าตั้งแต่ 0-65,535 สาเหตุที่เป็นเพียงการจำลองการสุ่มเพราะตัวเลขที่สร้างขึ้นได้มาจากการกระทำทางลอจิก ซึ่งจะให้ผลลัพธ์เหมือนกันทุกครั้งสำหรับอินพุทค่าๆ หนึ่ง

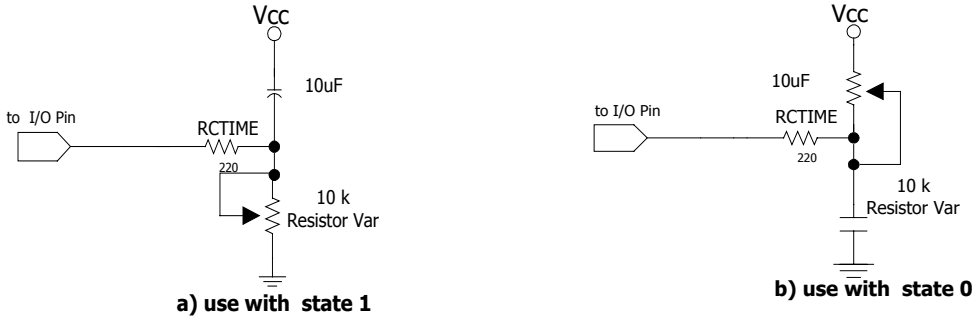
ตัวอย่าง RANDOM Result ‘ Result เป็นตัวแปรที่ใช้ในการสุ่มเลข

RCTIME

RCTIME *pin,state,Variable*

เป็นคำสั่งสำหรับนับค่า เวลาที่ขา I/O ยังคงสถานะเดิมอยู่ ใช้ในการวัดค่าเวลาของการเก็บประจุและการคายประจุของตัว Capacitor ในวงจร RC

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 สำหรับกำหนดขา I/O ของเบสิกแสตมป์ที่ใช้งาน โดยขา I/O นั้นจะถูกกำหนดให้เป็นขาอินพุท และหลังจากใช้งานแล้วก็ยังคงสถานะเป็นขาอินพุทเช่นเดิม
- **state** เป็นค่าตัวแปรหรือค่าคงที่ มีค่าเป็น 0 - 1 เพื่อกำหนดสถานะของขาที่ต้องการจะวัด และเมื่อทำงานจนจบคำสั่งแล้วจะทำการเก็บผลลัพธ์ที่ได้ไว้ในตัวแปร
- **Variable** ปกติจะเป็นค่าตัวแปรแบบเวิร์ด ที่ใช้เก็บค่าเวลาที่วัดได้ ในเบสิกแสตมป์ BS2P จะมีหน่วยเป็น 0.9 μS



ตัวอย่าง

RCTIME 8,1,Result ทำการวัดค่าเวลาการชาร์จประจุในวงจร RCและเก็บค่าที่วัดได้ในตัวแปร Result

READ

READ *location,variable*

เป็นคำสั่งที่ใช้ในการอ่านข้อมูลจากหน่วยความจำ EEPROM มาเก็บไว้ในตัวแปร

- **location** เป็นค่าตัวแปรหรือค่าคงที่ มีค่า 0-2047 เพื่อใช้กำหนดตำแหน่งของ address ที่ใช้ในการอ่านหน่วยความจำ EEPROM
- **variable** ปกติจะเป็นตัวแปรแบบไบต์ ใช้สำหรับเก็บข้อมูลที่อ่านได้จากหน่วยความจำ EEPROM

ตัวอย่าง READ 100,B0 ‘ ทำการอ่านข้อมูลในหน่วยความจำ EEPROM ตำแหน่ง location 100จากนั้นนำค่าที่อ่านได้ไปเก็บไว้ในตัวแปร B0

RETURN

RETURN

เป็นคำสั่งที่กำหนดให้เบสิกแสตมป์ กลับไปทำงานใน Main Program หลังจากเสร็จสิ้นการทำงานใน Subroutine โดยคำสั่ง RETURN นี้ต้องใช้คู่กับคำสั่ง GOSUB

ตัวอย่าง

```
GOSUB   Hello
DEBUG “How are you?”
END
Hello :
DEBUG “Hello my friend.”,cr
RETURN
```

จากตัวอย่าง การใช้คำสั่ง RETURN ต้องใช้ร่วมกับคำสั่ง GOSUB เนื่องจากเมื่อเบสิกแสตมป์กระทำตามคำสั่ง GOSUB จะกระโดดมากทำงานใน Subroutine จากนั้นจะไม่สามารถกลับไปทำงานยังโปรแกรมหลักได้จนกว่าจะพบคำสั่ง RETURN

REVERSE

REVERSE ***pin***

ทำการกลับสถานะการทำงานของขา I/O จากอินพุทเป็นเอาท์พุท หรือจากเอาท์พุทเป็นอินพุท

- **pin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 ใช้เลือกขา I/O ที่ต้องการกลับการทำงาน

ตัวอย่าง

```
REVERSE 13    ‘ ทำการกลับสถานะของขา 13
```

RUN

RUN ***program***

เป็นคำสั่งให้เบสิกแสตมป์ไปทำงานในโปรแกรมที่กำหนด

- **program** เป็นตัวแปรหรือค่าคงที่ สำหรับกำหนดตำแหน่งของโปรแกรมที่ต้องการให้เบสิกแสตมป์ทำงาน มีค่าตั้งแต่ Program 0 - 7

ตัวอย่าง

```
RUN 4          ‘ สั่งให้เบสิกแสตมป์ไปทำงานในโปรแกรมที่ 4
```

SERIN

SERIN ***rpin{\fpin},baudmode,{plabel,}{timeout,tlabel,}[inputData]***

เป็นคำสั่งที่ใช้รับข้อมูลแบบ Asynchronous เช่นข้อมูลจากพอร์ตอนุกรม RS232

- **rpin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-16 ใช้เลือกขา I/O ที่ใช้สำหรับรับข้อมูลอินพุทอนุกรม เมื่อต้องการจะใช้รับข้อมูล ขานั้นจะถูกกำหนดให้เป็นขาอินพุท และจะคงสถานะนี้ไว้หลังจากที่ทำคำสั่งเสร็จเรียบร้อยและหากขา rpin นี้ถูกกำหนดให้เท่ากับ 16 เบสิกแสตมป์จะใช้ขา S_{IN} ในการรับข้อมูล ซึ่งปกติขานี้จะใช้ในการดาวน์โหลดข้อมูลลงในตัวเบสิกแสตมป์
- **fpin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 ใช้ในการเลือกขา I/O ที่จะใช้งานเพิ่มมาเป็นขา Flow Control เมื่อมีการกระทำตามคำสั่ง โดยขาที่ถูกเลือกจะทำงานเป็นขาเอาท์พุทและคงสถานะไว้จนจบคำสั่ง

● **baudmode** เป็นค่าตัวแปรหรือค่าคงที่ขนาด 16 บิท มีค่า 0-65,535 เพื่อกำหนดคุณสมบัติต่างๆของการสื่อสารข้อมูลแบบอนุกรม ใช้กำหนดค่าอัตราเร็วในการถ่ายทอดข้อมูล หรือ Baud Rate

Data Speed	Direct Connection		Through Line Driver	
Baud Rate	8 data bits, no parity	7 data bits, even parity	8 data bits, no parity	7 data bits, even parity
600	20530	28722	4146	12338
1200	18447	26639	2063	10255
2400	17405	25597	1021	9213
4800	16884	25076	500	8692
9600	16624	24816	240	8432
19200	16494	24686	110	8302
38400	16429	24621	45	8237

● **plabel** เป็นตัวเลือกเพิ่มเติมใช้กำหนดลาเบลที่โปรแกรมจะกระโดดไปกรณีที่เกิด Parity Error โดยคำสั่งนี้จะทำงานก็ต่อเมื่อกำหนดค่า Baudmode ให้ใช้กับจำนวนข้อมูล 7 บิท และพาริตี้คู่

● **timeout** เป็นตัวเลือกเพิ่มเติม ซึ่งเป็นตัวแปรหรือค่าคงที่ มีค่า 0-65,535 เพื่อกำหนดเวลาในการรอรับข้อมูลหลังจากกระทำตามคำสั่ง ในเบสิกแสตมป์ BS2P จะมีหน่วยเป็น 0.4 ms และเมื่อถึงเวลาที่กำหนดไว้แล้วยังไม่มีข้อมูลเข้ามาทางขาอินพุท โปรแกรมจะกระโดดไปยัง address ที่ระบุไว้ใน tlable

● **tlabel** เป็นตัวเลือกเพิ่มสำหรับกำหนดตำแหน่ง label ที่โปรแกรมจะกระโดดไปเมื่อเกิด timeout หรือไม่มีการรับข้อมูลเข้ามาเมื่อถึงเวลาที่กำหนดไว้ใน timeout

● **InputData** ตัวแปรที่ใช้เก็บข้อมูลที่รับเข้ามา

<u>ตัวอย่าง</u>

```
SERDATA     var     byte
SERIN  1,16624,[DEC SERDATA]    ' รับข้อมูลจากขา 1 ที่ baud rate 9,600 บิท/วินาที
                                  จำนวนบิทข้อมูล 8 บิท ไม่มีพาริตี้ และมีการกลับขั้วสัญญาณที่รับเข้ามา
                                  และจะเก็บค่าที่อ่านได้ไว้ในตัวแปร SERDATA เป็นเลขฐาน 10
```

SEROUT

SEROUT ***tpin,baudmode,{pace,}[outputData]***

SEROUT tpin\fpin,baudmode,{timeout,tabel,}[outputdata]

ส่งข้อมูลแบบ Asynchronous เช่นข้อมูลจากพอร์ตอนุกรม RS232

● **tpin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-16 ใช้เลือกขา I/O ที่ใช้สำหรับส่งข้อมูลและจะถูกกำหนดให้เป็นขาเอาท์พุท และจะคงสถานะนี้ไว้หลังจากที่ทำคำสั่งเสร็จเรียบร้อย และหากขา tpin นี้ถูกกำหนดให้เท่ากับ 16 เบสิกแสตมป์จะใช้ขา S_{OUT} ในการส่งข้อมูล ซึ่งปกติขานี้จะใช้ในการดาวน์โหลดโปรแกรมจาก PC กับตัวเบสิกแสตมป์

● **fpin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 ใช้ในการเลือกขา I/O ที่จะใช้งานเป็นขา Flow Control เมื่อมีการกระทำตามคำสั่ง โดยขาที่ถูกเลือกจะทำงานเป็นขาอินพุทและคงสถานะแม้ว่าจะจบคำสั่งแล้วก็ตาม

● **baudmode** เป็นค่าตัวแปรหรือค่าคงที่ขนาด 16 บิท เพื่อกำหนดคุณสมบัติต่างๆของการสื่อสารข้อมูลแบบอนุกรม โดยข้อมูล 13 บิทด้านต่ำ (บิท 0-12) ใช้กำหนดค่าอัตราเร็วในการถ่ายทอดข้อมูล หรือ Baud Rate

Data Speed	Direct Connection		Through Line Driver	
Baud Rate	8 data bits, no parity	7 data bits, even parity	8 data bits, no parity	7 data bits, even parity
600	20530	28722	4146	12338
1200	18447	26639	2063	10255
2400	17405	25597	1021	9213
4800	16884	25076	500	8692
9600	16624	24816	240	8432
19200	16494	24686	110	8302
38400	16429	24621	45	8237

● **pace** เป็นตัวเลือกเพิ่มเติมอาจเป็นตัวแปรหรือค่าคงที่ มีค่า 0-65,535 ใช้กำหนดค่าเวลาในการส่งข้อมูล มีหน่วยเป็น 0.4 ms

● **OutputData** เป็นตัวแปรหรือค่าคงที่ของข้อมูลที่ใช้ในการส่ง

● **timeout** เป็นตัวเลือกเพิ่มเติม ซึ่งเป็นตัวแปรหรือค่าคงที่ มีค่า 0-65,535 เพื่อกำหนดเวลาในการรอสัญญาณ Flow Control ในภาครับทางขา fpin มีหน่วยเป็น 0.4 ms และเมื่อถึงเวลาที่กำหนดไว้แล้วยังไม่มีสัญญาณ Flow Control จากภาครับ โปรแกรมจะกระโดดไปยัง address ที่ระบุไว้ใน tlable

● **tlabel** เป็นตัวเลือกเพิ่มเติมซึ่งใช้กับขา fpin และ timeout ใช้สำหรับกำหนดตำแหน่งที่โปรแกรมจะกระโดดไปเมื่อเกิด timeout

ตัวอย่าง SEROUT 7,240,[“A”] ‘ ส่งตัว “A” ออกจากขา 7 ที่ baud rate 9,600 บิท/วินาที
จำนวนบิทข้อมูล 8 บิท ไม่มีพาริตี้ และไม่มีการกลับขั้วสัญญาณ

SHIFTIN

SHIFTIN *dpin,cpin,mode,[variable{\bits}{,variable{\bits}…}]*

เป็นการอ่านข้อมูลจากอุปกรณ์ที่มีการสื่อสารอนุกรม ซิงโครนัส และในเบสิกแสตมป์ BS2P40 มีอัตราการผ่านประมาณ 42 kBits/sec

● **dpin** เป็นตัวแปรหรือค่าคงที่มีค่า 0-15 ใช้เลือกขา I/O ของเบสิกแสตมป์ สำหรับรับข้อมูลซิงโครนัส โดยขาที่เลือกใช้นี้จะถูกเซตเป็นโหมดอินพุท

● **cpin** เป็นตัวแปรหรือค่าคงที่มีค่า 0-15 ใช้เลือกขา I/O ของเบสิกแสตมป์ สำหรับเชื่อมต่อกับขาอินพุทของสัญญาณนาฬิกาของอุปกรณ์ซิงโครนัสที่ต้องการจะติดต่อด้วย โดยขานี้จะถูกเซตเป็นโหมดเอาท์พุท

● **mode** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-3 ช่วยบอกถึงรูปแบบของการจัดการบิทข้อมูลที่มีความเกี่ยวข้องกับ Clock Pulse โดยแสดงรายละเอียดของโหมดต่างๆ ดังนี้

0 MSBPRE ข้อมูลบิทนัยสำคัญสูงสุดถูกส่งมาก่อน,ข้อมูลถูกส่งออกมาก่อนสัญญาณนาฬิกา

1 LSBPRE ข้อมูลบิทนัยสำคัญต่ำสุดถูกส่งมาก่อน,ข้อมูลถูกส่งออกมาก่อนสัญญาณนาฬิกา

2 MSBPORT ข้อมูลบิทนัยสำคัญสูงสุดถูกส่งมาก่อน,ข้อมูลถูกส่งหลังจากได้รับสัญญาณนาฬิกา

3 LSBPORT ข้อมูลบิทนัยสำคัญต่ำสุดถูกส่งมาก่อน, ข้อมูลถูกส่งหลังจากได้รับสัญญาณนาฬิกา

● **variable** เป็นตัวแปรแบบนิปเปิล บิท ไบต์หรือเวิร์ด ที่ใช้เก็บบิทข้อมูลที่รับเข้ามา

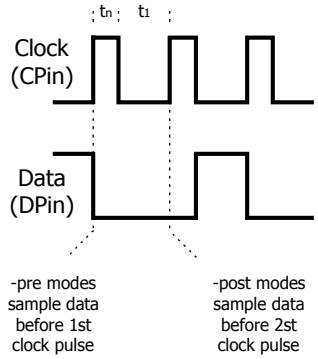
● **bit** ใช้กำหนดจำนวนอินพุทที่จะรับเข้ามา มีค่า 1-16 ถ้าไม่ใส่ ถือว่าเป็น 8

คำสั่งนี้เป็นคำสั่งที่ช่วยอำนวยความสะดวกในการติดต่อกับอุปกรณ์ที่ใช้การสื่อสารข้อมูลแบบอะซิงโครนัส อุปกรณ์ที่มีลักษณะการสื่อสารข้อมูลแบบอะซิงโครนัสได้แก่ ADC0831 , ADC0832 , LTC1298 , DS1620, MAX7219 เป็นต้น

คำสั่ง SHIFTIN 1 คำสั่งจะมีการทำงานดังนี้

1. ทำให้ขา Cpin ซึ่งเป็นขาสัญญาณนาฬิกา มีสถานะเป็น 0
2. ทำให้ขาข้อมูล Dpin เป็นอินพุท
3. อ่านค่าสถานะของบิทข้อมูลเข้ามาเก็บ
4. ส่งสัญญาณนาฬิกาออกไปก่อนหรือหลังจากการรับบิทข้อมูล
5. เลื่อนบิตผลลัพธ์ทางซ้ายหรือขวา ขึ้นอยู่ที่การกำหนดตัวแปรที่โหมด

ทำตามขั้นตอนเดิม จนกว่าข้อมูลที่รับเข้ามาครบตามจำนวนบิทที่กำหนดไว้



ตัวอย่าง

```
Result          VAR   BYTE
SHIFTIN  0,1,MSBPRE,[Result]    'ขา I/O Pin 0 ถูกเซตเป็นอินพุท , ขา I/O Pin 1 ถูกเซตเป็นเอาท์พุท
                                ซึ่งจะส่งบิทสำคัญสูงสุดออกมาก่อน ,ข้อมูลถูกส่งออกมาก่อน-
                                สัญญาณนาฬิกาและเก็บข้อมูลที่รับมาไว้ที่ Result
```

SHIFTOUT

SHIFTOUT ***dpin,cpin,mode,[OutputData{\bits}{, OutputData {\bits}…}]***

ใช้เลื่อนข้อมูลออกไปยังอุปกรณ์ที่มีการสื่อสารอนุกรมแบบซิงโครนัส ในเบสิกแสตมป์ BS2P40 มีอัตราการส่งผ่านประมาณ 42 kBits/se

- **dpin** เป็นตัวแปรหรือค่าคงที่มีค่า 0-15 ใช้เลือกขา I/O ของเบสิกแสตมป์ สำหรับส่งข้อมูลซิงโครนัส โดยทำงานเป็นเอาท์พุท มีสถานะเป็นเอาท์พุท
- **cpin** เป็นตัวแปรหรือค่าคงที่มีค่า 0-15 ใช้เลือกขา I/O ของเบสิกแสตมป์ สำหรับส่งสัญญาณนาฬิกาไปยังอุปกรณีที่มีการสื่อสารข้อมูลแบบซิงโครนัส โดยขานี้จะถูกเซตเป็นโหมดเอาท์พุท
- **mode** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-1 ช่วยบอกถึงลำดับของการจัดการบิทข้อมูล โดยแสดงรายละเอียดของโหมดต่างๆ ดังนี้

```
0       LSBFIRST    ' ส่งข้อมูลออกไปโดยส่งบิตในสำคัญต่ำสุดออกไปก่อน
1       MSBFIRST    ' ส่งข้อมูลออกไปโดยส่งบิตในสำคัญสูงสุดออกไปก่อน
```

(Msb is most-significant bit; the highest or leftmost bit of a nibble,byte, or word. Lsb is the least-significant bit; the lowest or rightmostbit of a nibble, byte, or word.)

- **outputData** เป็นตัวแปรหรือค่าคงที่ๆ เป็นข้อมูลที่จะส่งออกไป
- **bit** เป็นตัวเลขที่ระบุว่าจะส่งข้อมูลกี่ bit มีค่า 1-16 ถ้าไม่ใส่ ถือว่าเป็น 8 bit

ตัวอย่าง SHIFTOUT 0,1,MSBFIRST,[250] ‘ สร้างสัญญาณ Pulse 8 บิท ในขณะที่ทำการเขียนค่าแต่ละบิท (ของ 8 บิทที่ค่า 250) บนขา Dpin โดยเริ่มบิทนัยสำคัญสูงสุด บิทแรกมีการจัดการบิทข้อมูลแบบ MSBFIRST

คำสั่ง SHIFTOUT นี้เป็นคำสั่งที่ช่วยอำนวยความสะดวกในการติดต่อกับอุปกรณ์ที่ใช้การสื่อสารข้อมูลแบบอะซิงโครนัส อุปกรณ์ที่มีลักษณะการสื่อสารข้อมูลแบบอะซิงโครนัสเช่น ADC0831,ADC0832,LTC1298, DS1620, MAX7219 เป็นต้น

คำสั่ง SHIFTON 1 คำสั่ง จะมีการทำงานดังนี้

1. ทำให้ขา Cpin ซึ่งเป็นขาสัญญาณนาฬิกา มีสถานะเป็น 0
2. นำบิตของข้อมูลที่จะส่งป้อนไปยังขาข้อมูล
3. กำหนดขาข้อมูล Dpinให้เป็นขาเอาท์พุท
4. ให้ขาสัญญาณนาฬิกาเป็น 1

ทำตามขั้นตอนเดิม จนกว่าข้อมูลที่จะส่งออกไปครบตามจำนวนบิทที่กำหนดไว้

SLEEP

SLEEP ***period***

เป็นคำสั่งที่ทำให้เบสิกแสตมป์ อยู่ในโหมด LOW - POWER

- **period** เป็นตัวแปร ค่าคงที่ มีค่า1-65,535 แสดงเวลาที่ต้องการให้เบสิกแสตมป์ อยู่ใน Sleep Mode กี่วินาทีความละเอียดประมาณ 2.3 Sec ซึ่งคำสั่ง sleep นี้สามารถทำได้นานสุด 18 ช.ม.

ตัวอย่าง SLEEP 10 ‘ กำหนดให้เบสิกแสตมป์อยู่ใน Sleep Mode 10 วินาที

STOP

หยุดการทำงานของโปรแกรมแต่ไม่เข้าสู่โหมด Low-Power

STORE

STORE ***programslot***

ระบุชื่อโปรแกรม slot สำหรับกระทำการ Read/Write

- **programslot** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-7 แสดงโปรแกรม slot ที่ใช้ในการ Read/Write

ตัวอย่าง STORE 1 ‘ ทำการเลือก Read/Write Slot 1

TOGGLE

TOGGLE ***pin***

เป็นคำสั่งที่กลับสถานะของขา I/O ที่ใช้ คือ เปลี่ยนจาก ”1” เป็น “0” หรือ จาก “0” เป็น “1”

- **pin** เป็นตัวแปรหรือค่าคงที่ เป็นขาที่ต้องการกลับสถานะ

ตัวอย่าง TOGGLE 2 ‘ กำหนดให้ทำการกลับสถานะขา 2

WRITE

WRITE ***Location,DataItem***

ใช้เขียนรายการข้อมูล เข้าไปใน Location ของ EEPROM

- Location เป็นตัวแปรหรือค่าคงที่ มีค่า0–2047ซึ่งแสดงตำแหน่งของEEPROMที่จะเขียนข้อมูลลงไป
- DataItem เป็นตัวแปรหรือค่าคงที่ แสดงค่าข้อมูลที่ต้องการจะเก็บ

ตัวอย่าง WRITE 100,245 ‘ เก็บค่า 245 ไว้ที่ตำแหน่ง 100 ของ EEPROM

XOUT

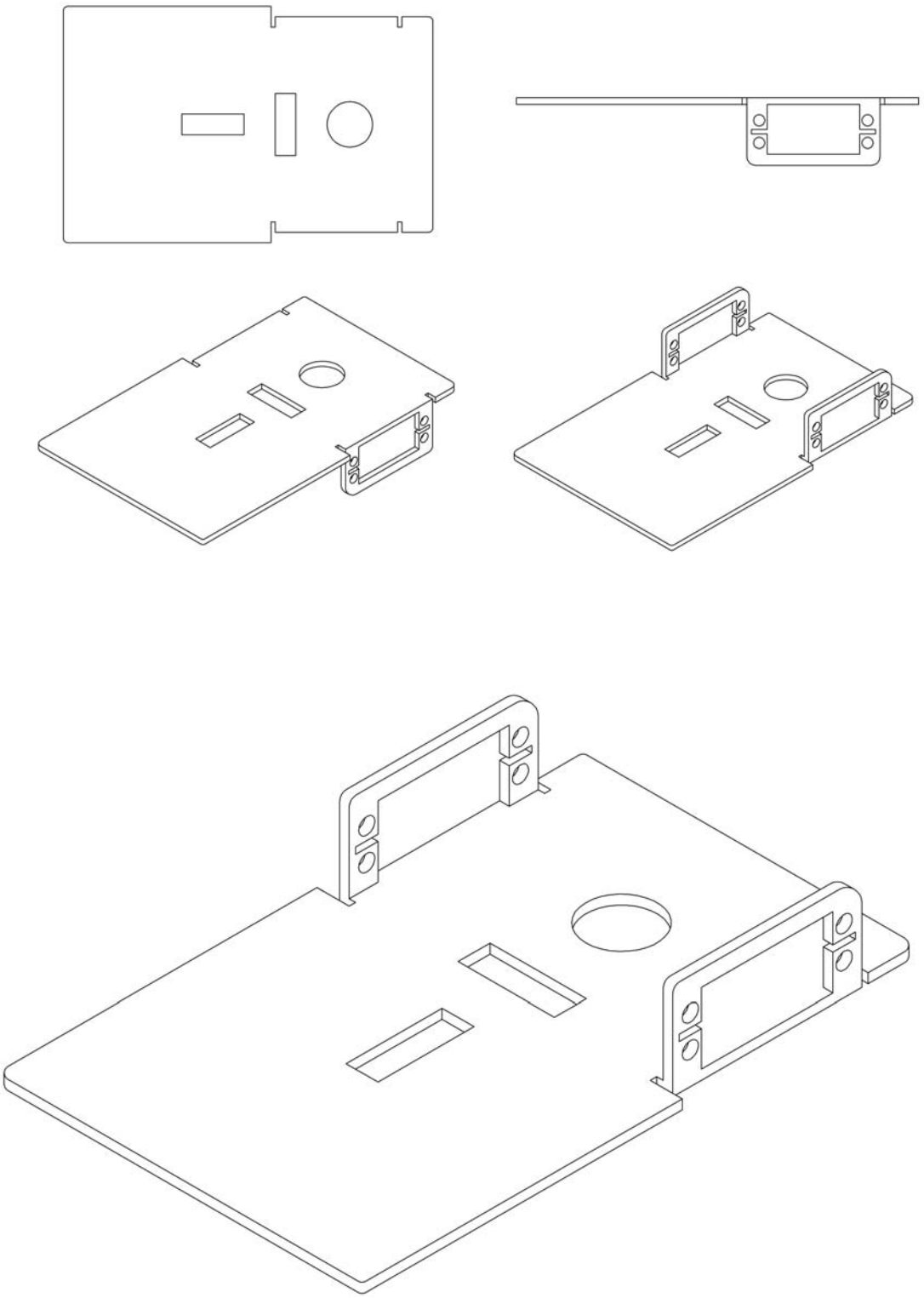
XOUT ***mpin,zpin,[house\command{\cycles}{,house\command{\cycles}…}]***

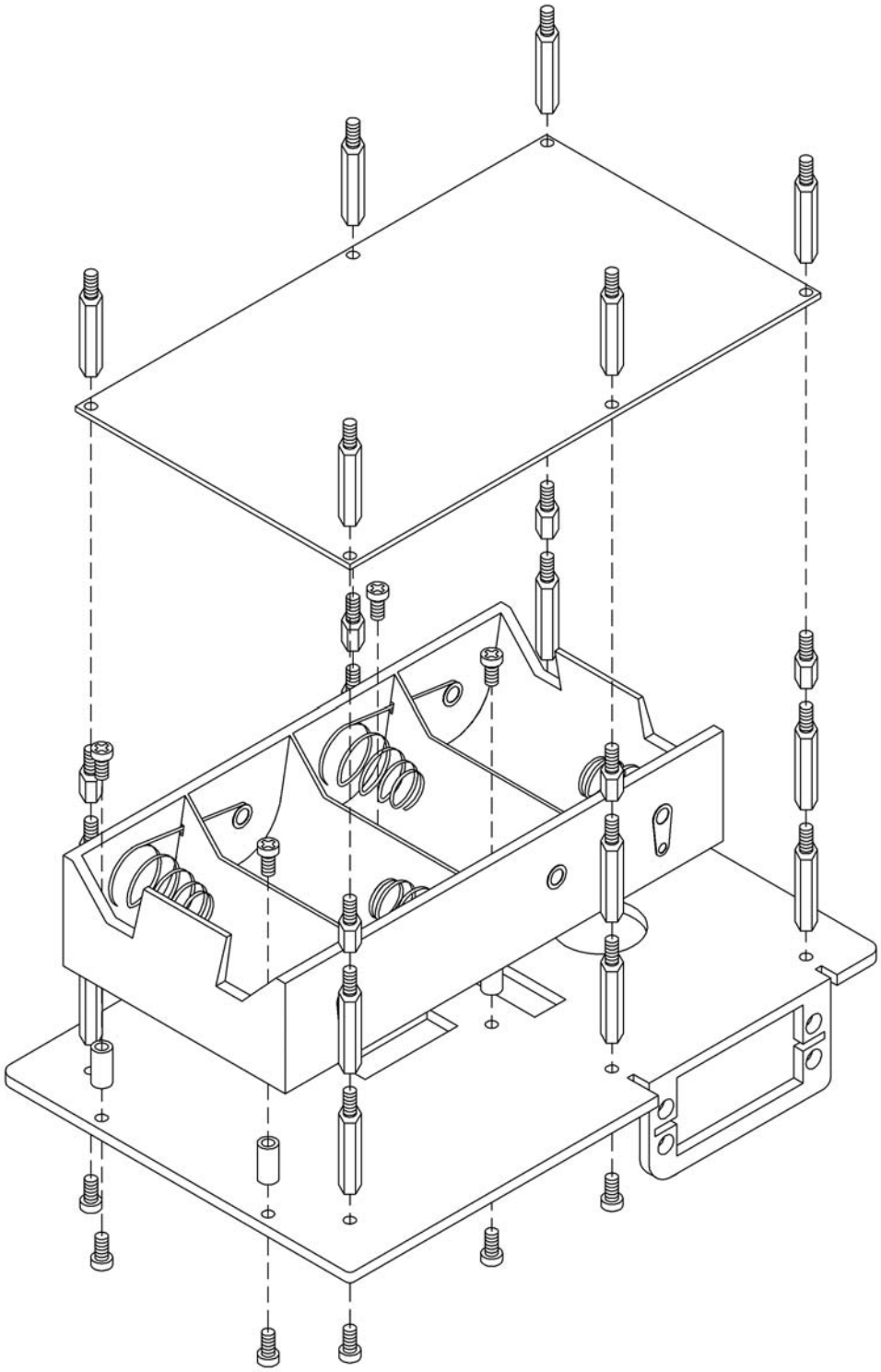
เป็นคำสั่งที่ใช้ส่งเพื่อไปทำการควบคุมเครื่องใช้ไฟฟ้าผ่านสายไฟฟ้าสลับมาตราฐาน X-10

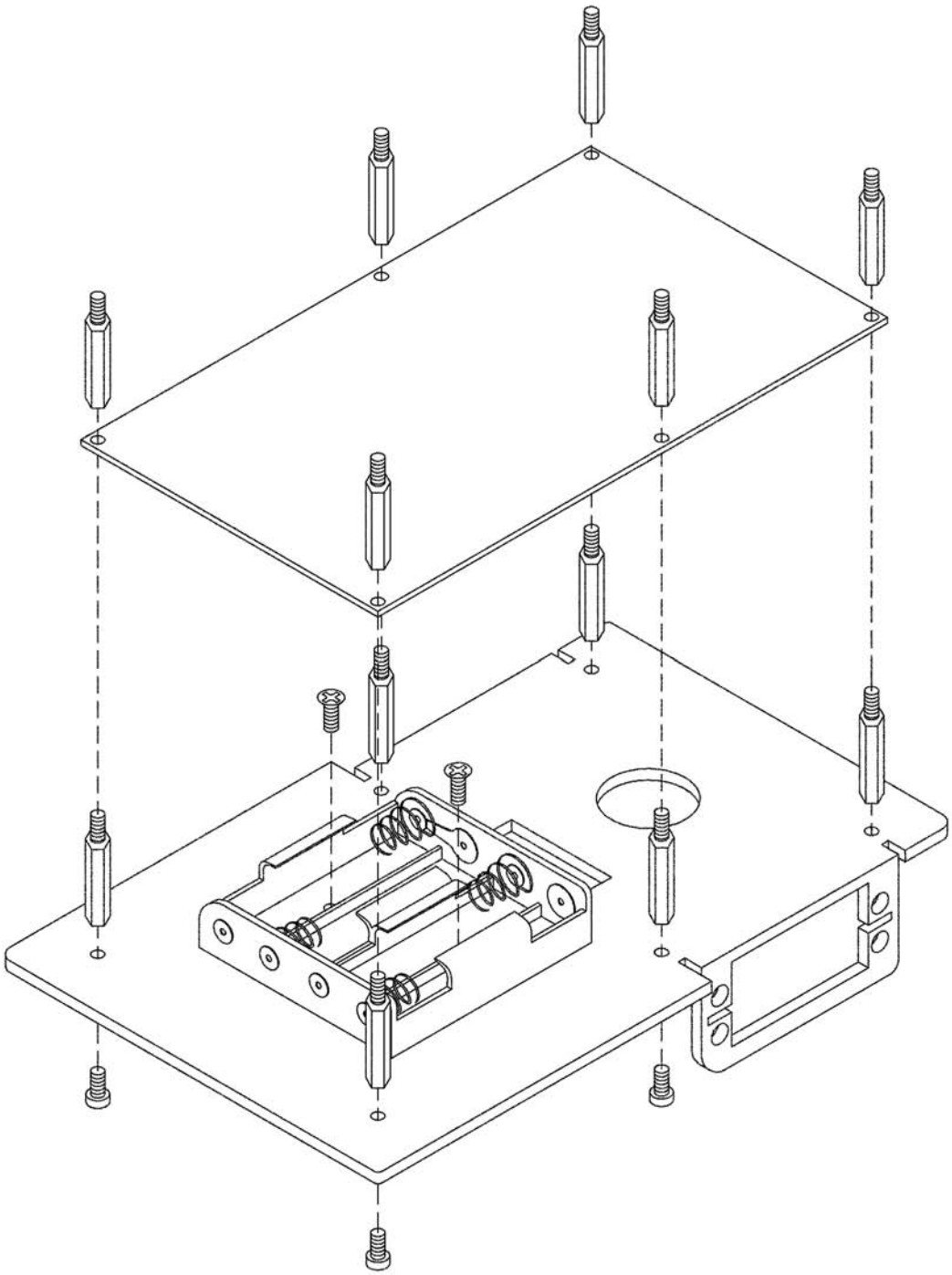
- **Mpin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 แสดงขา I/O ที่ใช้เป็นขาเอาท์พุทที่เชื่อมต่อสายไฟฟ้า
- **Zpin** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 แสดงขา I/O ที่ใช้เป็นขาอินพุทรับสัญญาณจุดฉนวนที่จุดศูนย์ (Zero crossing) จากอุปกรณ์เชื่อมต่อไฟฟ้ากระแสสลับ
- **House** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-15 แทนรหัสของอุปกรณ์ไฟฟ้าที่ใช้มาตราฐานการควบคุมแบบ X-10 ใช้แทนตัวอักษร A ถึง P
- **Command** เป็นตัวแปรหรือค่าคงที่ มีค่า 0-30 แสดงรายละเอียดของคำสั่งที่จะใช้ส่ง
- **Cycles** เป็นค่าเวลาในการส่ง command มีค่า 1-255 ถ้าไม่ใส่ค่าในคำสั่งจะมีค่าเป็น 2

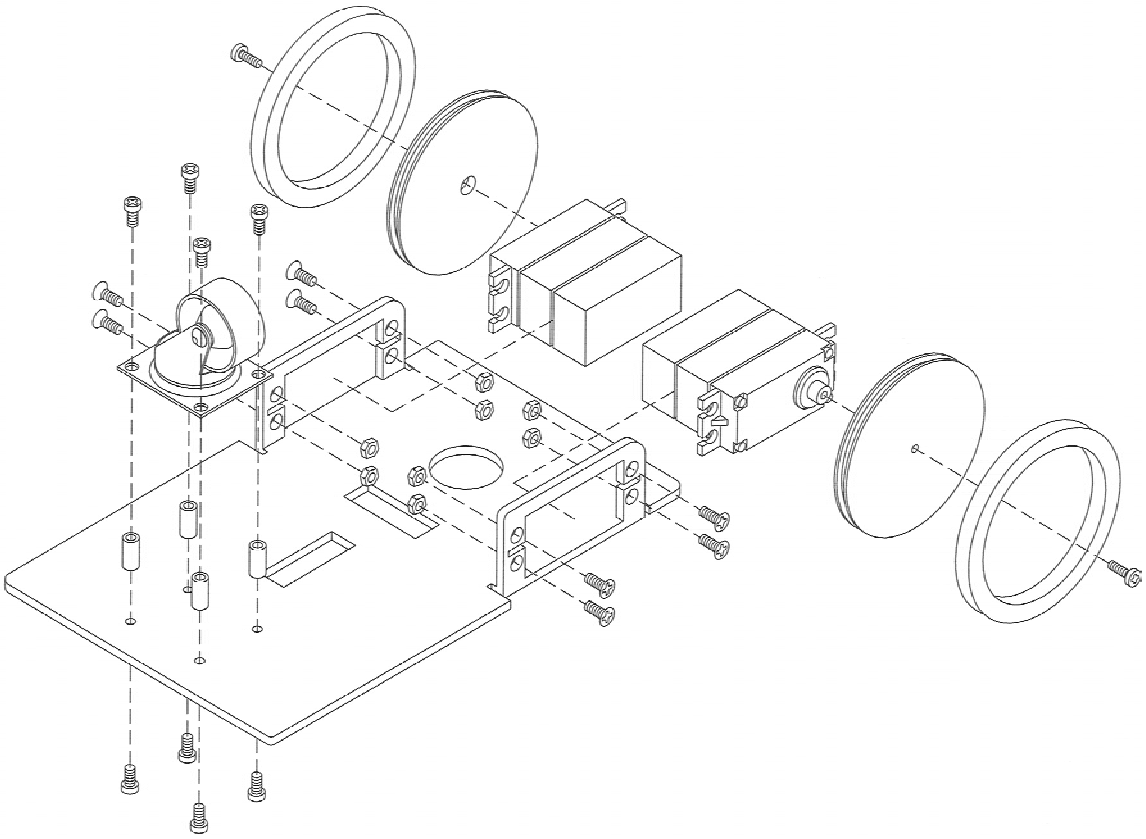
ตัวอย่าง XOUT 0,1,[HouseA\Unit1] ‘ ใช้ขา 0 ส่งข้อมูลคำสั่ง และขา 1 รับสัญญาณ Zero crossing จากอุปกรณ์เชื่อมต่อสายไฟสลับ ควบคุมอุปกรณ์ไฟฟ้า House Code : A = 1 ที่ Unit Code : 1 = 2

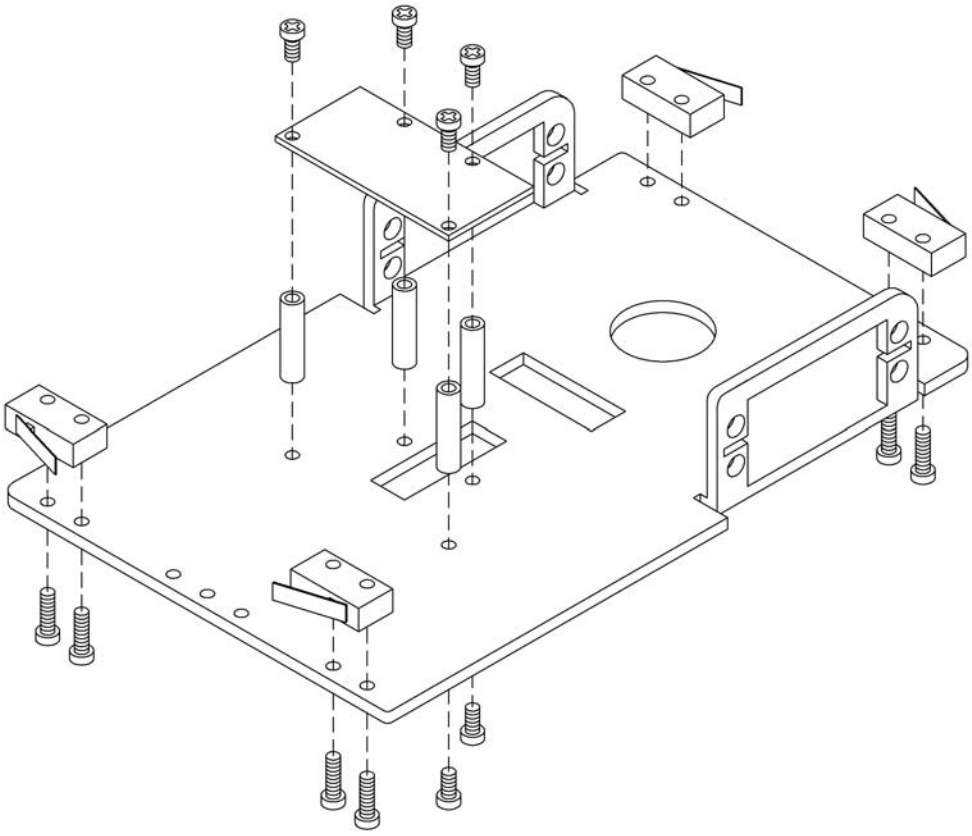
โครงหุ่นยนต์ และ การประกอบติดตั้ง



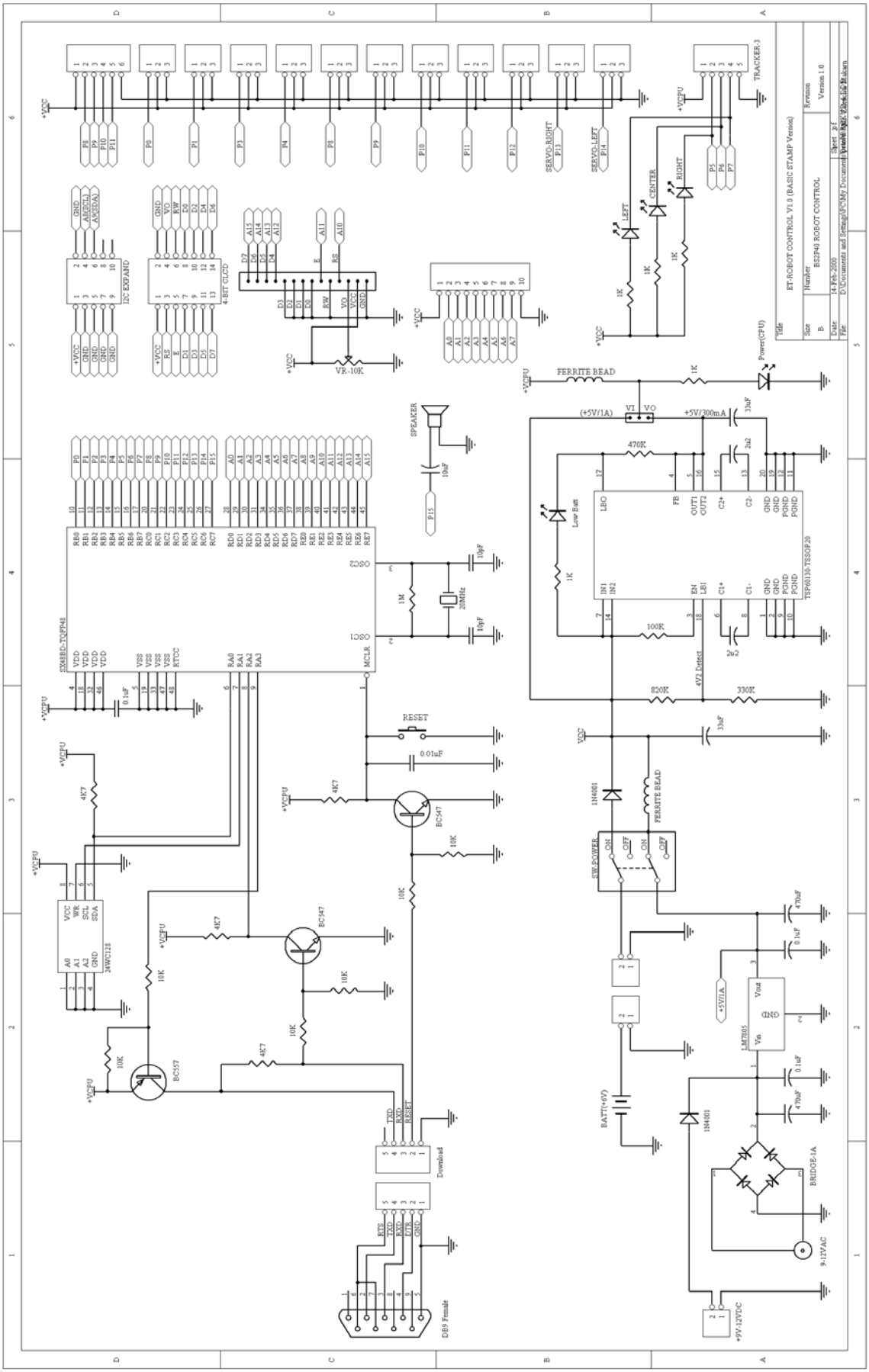


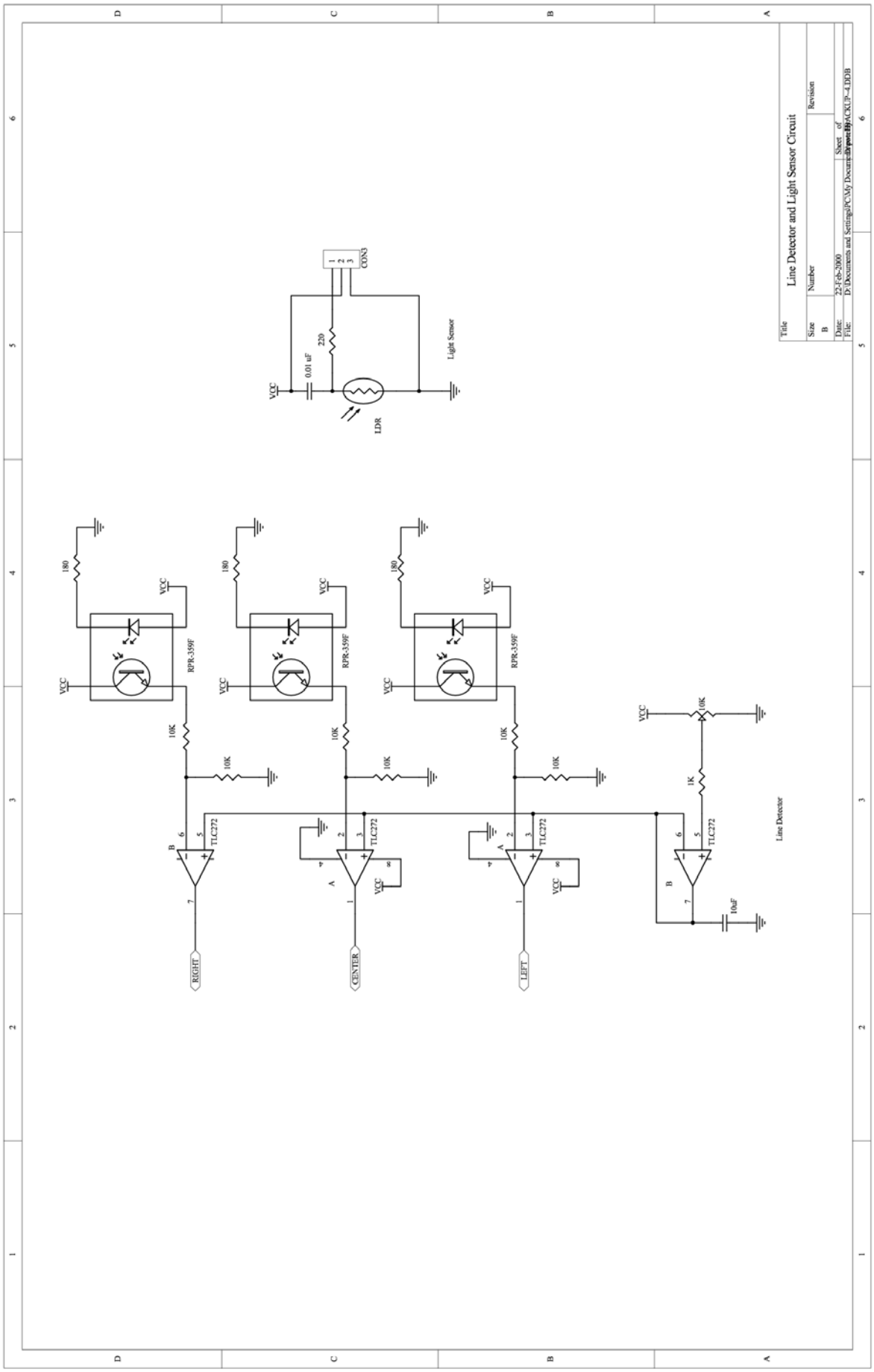






วงจร (Circuit)




Line Detector and Light Sensor Circuit
Title
Number
Revision
Size
B
Date: 23-Feb-2000
Sheet of
File: D:\Documents and Settings\PCM\My Documents\ET-ROBOT\ACKL\P~4.DDB
CON3
Light Sensor
0.01 uF
220
LDR
VCC
180
RPR-359F
10K
TLC272
1K
10uF
RIGHT
CENTER
LEFT
Line Detector

อุปกรณ์สนับสนุนระบบ ROBOT มีให้เลือกใช้งาน จาก อีทีที

ชุดล้อต่าง ๆ

- ล้อรุ่น R-WHEEL SERVO ชุดล้อพร้อมยางอย่างดี ขึ้นรูปด้วยการฉีดพลาสติก ขนาดล้อเส้นผ่าศูนย์กลาง 6.7 CM ออกแบบมาให้ต่อเข้ากับตัว SERVO MOTOR รุ่น S03N,S03T ราคาล้อละ **ราคา 100.-**
- ล้อรุ่น R-WHEEL LEGO 43.2 ชุดล้อพร้อมยางและชุด MODIFY เข้ากับตัว SERVO MOTOR รุ่น S03N,S03T ขนาดล้อเส้นผ่าศูนย์กลาง 4.32 CM ราคาล้อละ **ราคา 160.-***
- ล้อรุ่น R-WHEEL LEGO 81.6 ชุดล้อพร้อมยางและชุด MODIFY เข้ากับตัว SERVO MOTOR รุ่น S03N,S03T ขนาดล้อเส้นผ่าศูนย์กลาง 8.16 CM ราคาล้อละ **ราคา 240.-***
- ล้อรุ่น R-WHEEL FRONT 25 ชุดล้อหน้าเป็นแบบแป้นหมุนรอบตัว ตัวล้อแบบไนล่อนขนาดล้อเส้นผ่าศูนย์กลาง 2.5 CM าคาล้อละ **ราคา 45.-**

* สินค้ามีจำนวนจำกัด กรุณาโทรสอบถามก่อนสั่งซื้อ

ชุดลังถ่าน ,BATTERY และ เครื่องชาร์จ BATTERY

- ลังถ่าน รุ่น R-D-TYPE4 เป็นชุดลังถ่านขนาด 4 ก้อน ใช้กับถ่านขนาด SIZE D พร้อมสายต่อและขาตั้งเข้ากับชุดฐาน R-BASE **ราคา 45.-**
- ลังถ่าน รุ่น R-AA-TYPE4 เป็นชุดลังถ่านขนาด 4 ก้อน ใช้กับถ่านขนาด SIZE AA พร้อมสายต่อเข้ากับชุดฐาน R-BASE **ราคา 35.-**
- BATTERY รุ่น SL6-5 เป็น BATTERY แบบ ชาร์จได้ SEALED RECHARGEABLE ขนาด 6 V 5 AMP ขนาด 70x47x102x106 mm น้ำหนัก 0.81 Kg พร้อมสายต่อ **ราคา 170.-**
- BATTERY รุ่น SL6-1.3 เป็น BATTERY แบบ ชาร์จได้ SEALED RECHARGEABLE ขนาด 6 V 1.3 AMP ขนาด 97x24x51x55 mm น้ำหนัก 0.32 Kg พร้อมสายต่อ **ราคา 165.-**
- ถ่านไฟฉายรุ่น 1.5V SIZE D แบบ ALKALINE SANYO บรรจุขายขนาด 2 ก้อน **ราคา 110.-***
- CHARGERS รุ่น PACB-0004 เป็นเครื่อง CHARGER BATTERY ขนาด 6V 1 AMP สามารถใช้กับ BATTERY รุ่น SL6-5 และ SL6-1.3 ได้ โดยจะหยุดชาร์จไฟเมื่อเติมแล้วเอง **ราคา 170.-**

ระยะเวลาการใช้งานชุด ET-ROBOT STAMP P40 กับ BATTERY รุ่น SL6-5,SL6-1.3 และ 1.5V SIZE D

BATTERY รุ่น SL6-5 ใช้งานวิ่งตามเส้น โดยจะสามารถวิ่งอยู่ได้ประมาณ 6 ชม. 30 นาที

BATTERY รุ่น SL6-1.3 ใช้งานวิ่งตามเส้น โดยจะสามารถวิ่งอยู่ได้ประมาณ 3 ชม.

ถ่านไฟฉาย 1.5V SIZE D แบบ ALKALINE SANYO 4 ก้อน โดยจะสามารถวิ่งอยู่ได้ประมาณ 3 ชม.

SL6-5

SL6-1.3

ALKALINE SANYO

PACB-0004

ชุดตัวรถ,เส้าต่อ,และบอร์ด SENSOR ต่าง ๆ

- ชุด R-BASE เป็นชุดโครงตัวรถ ทำด้วย ALUMINUM อย่างดี ชุบ ANODIZED สีน้ำเงิน ขนาดหนา 3 mm เป็นตัวรถที่มีน้ำหนักเบา ออกแบบให้ใช้กับชุด SERVO MOTOR รุ่น S03N,S03T **ราคา 320.-**
- ชุดเส้าต่อพลาสติกแบบตัวหนุนเกลียวนอกในมีหลายขนาดความยาว ใช้เสริมต่อเข้ากับ ชุด R-BASE เกลียวแบบ M3 มีจำหน่ายเป็นชุด ชุดและ 10 ตัว

HTS-306 (สูง 6 mm) **ราคา 27.-** HTS-310 (สูง 10 mm) **ราคา 30.-** HTS-315 (สูง 15 mm) **ราคา 35.-**
HTS-320 (สูง 20 mm) **ราคา 38.-** HTS-325 (สูง 25 mm) **ราคา 40.-** หัวน็อตพลาสติก M3 **ราคา 35.-**

- ชุด R-TRACKER 3 เป็นชุดตรวจจับแสงตามเส้น ใช้ตัว SENSOR จำนวน 3 ชุด พร้อม IC OP AMP มีระยะตรวจสอบ 3 CM พร้อม VR ปรับค่าให้ OUTPUT เป็น TTL HI/LO พร้อมสายต่อ **ราคา 350.-**
- ชุด R-TRACKER 1 เป็นชุดตรวจจับแสงตามเส้น ใช้ตัว SENSOR จำนวน 1 ชุด พร้อม IC OP AMP มีระยะตรวจสอบ 3 CM พร้อม VR ปรับค่าให้ OUTPUT เป็น TTL HI/LO พร้อมสายต่อ **ราคา 210.-**
- ชุด R-LIGHT เป็นชุดตรวจจับ SENSOR แสง 1 ช่อง พร้อมสายต่อ **ราคา 30.-**
- ชุด R-SW เป็นชุดตรวจจับการชน 1 ช่อง พร้อมสายต่อ **ราคา 30.-**
- ชุด R-OPA 1 เป็นชุดขยายสัญญาณด้วย OPAMP 1 ช่อง เปลี่ยนสัญญาณต่างๆ ให้อยู่ในรูป OUTPUT แบบ TTL HI/LO พร้อมสายต่อ **ราคา 120.-**
- ชุด R-MOTOR เป็นชุดขับ DC MOTOR ขนาด 2 ตัวโดยใช้ IC เบอร์ L293D พร้อมสายต่อ **ราคา 210.-**
- ชุด PROJECT PCB M4 เป็น PCB เอนกประสงค์ SIZE 15.3x9cm ที่ต่อเข้ากับชุด R-BASE **ราคา 140.-**

ชุด STEPPING MOTOR

- STEPPING MOTOR รุ่น M42SP-5 เป็น STEPPING ขนาด 4 PHASE 12 VDC (สามารถใช้กับ 5VDC ได้) จำนวน 7.5 องศาต่อ STEP ขนาด 42x23 mm **ราคา 50.-***
- STEPPING MOTOR รุ่น PM55L-048 เป็น STEPPING ขนาด 4 PHASE 24 VDC (สามารถใช้กับ 5VDC ได้) จำนวน 7.5 องศาต่อ STEP ขนาด 55x25.7 mm **ราคา 90.-***
- STEPPING MOTOR รุ่น PH266L เป็น STEPPING ขนาดใหญ่ 4 PHASE 6 VDC 1.2A 1.8 องศาต่อ STEP ขนาด 55x55 mm ของบริษัท VEXTA (ORIENTAL MOTOR) **ราคา 300.-***

R-WHEEL SERVO

R-WHEEL LEGO 43.2

R-WHEEL FRONT 25

R-WHEEL LEGO 81.6

R-D-TYPE4

R-AA-TYPE4

ต่อใช้กับ BATTERY SL6-5

ต่อใช้กับ BATTERY SL6-5

R-BASE

เส้าต่อพลาสติก

R-OPA 1

R-TRACKER 3

R-LIGHT

PROJECT PCB M4

PH266L

PM55L-048

M42SP-5

บริษัท อีทีที จำกัด 1112/96-98 ถนนสุขุมวิท แขวงพระโขนง เขตคลองเตย กรุงเทพฯ 10110
Tel : 02-7121120 Fax : 02-3917216 www.ett.co.th email : sale@etteam.com

ETT
www.ett.co.th