ระบบควบคุมสัญญาณไฟการจราจร

TRAFFIC LIGHT CONTROL SYSTEM

โดย

นาย ณัฐกร ดิศพงศ์

นาย ทวีชัย สุรเจริญชัยกุล

นาย ทศพร หุ่นแก้ว

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ระบบควบคุมสัญญาณไฟการจราจร
TRAFFIC LIGHT CONTROL SYSTEM

โดย

นาย ณัฐกร ดิศพงศ์ 38014141
นาย ทวีชัย สุรเจริญชัยกุล 38014173
นาย ทศพร หุ่นแก้ว 38014174

อาจารย์ที่ปรึกษา

อาจารย์ โกศล ชวนขยัน

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

ปริญญานิพนธ์ ปีการศึกษา 2541

ภาควิชา อิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบควบคุมสัญญาณไฟการจราจร

ผู้จัดทำ

1. นาย ณัฐกร ดิศพงศ์
2. นาย ทวีชัย สุรเจริญชัยกุล
3. นาย ทศพร หุ่นแก้ว

..... อาจารย์ที่ปรึกษา

(นาย ไกรสร ขวนขยัน)

ระบบควบคุมสัญญาณไฟการจราจร

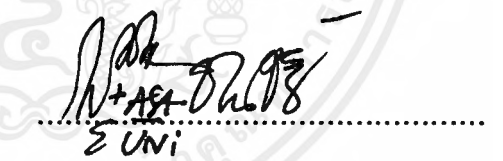
TRAFFIC LIGHT CONTROL SYSTEM

นาย ณัฐกร ดิศพงศ์ 38014141

นาย ทวีชัย สุรเจริญชัยกุล 38014173

นาย ทศพร หุ่นแก้ว 38014174

ปริญญานิพนธ์ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้



(นายโกศล ชวนชาญ)

อาจารย์ที่ปรึกษา

ระบบควบคุมสัญญาณไฟการจราจร

ณัฐกร ดิศพงศ์
ทวีชัย สุรเจริญชัยกุล
ทศพร หุ่นแก้ว
อ.โกศล ชวนขยัน อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์นี้มีวัตถุประสงค์เพื่อช่วยแก้ปัญหาการจราจร ซึ่งเป็นปัญหาทางสังคมที่ยังแก้ไขไม่ได้เนื่องจากระบบที่มีอยู่แล้วยังต้องใช้มนุษย์เข้าไปควบคุมอีกที ซึ่งจะเกิด Human error เนื่องจากการทำงานของมนุษย์ไม่มีความแม่นยำและความแน่นอน คอมพิวเตอร์ เป็นอุปกรณ์ที่ทำงานได้แม่นยำ

ปริญญานิพนธ์นี้มีรูปแบบการเปลี่ยนสัญญาณไฟเป็นชุดๆ 8 ชุดด้วยกัน โดยมีลักษณะเฉพาะคือ เมื่อจะทำการเปลี่ยนจากไฟเขียวเป็นไฟแดงจะใช้วิธีให้ไฟเขียวกระพริบแทนการใช้ไฟเหลือง ทำให้ช่วยในการตัดสินใจได้ดีขึ้น เพราะการลดรูปแบบการเปลี่ยนสัญญาณไฟจราจรให้สั้นลงทำให้เข้าใจความหมายจากระยะไกลได้มากขึ้น การติดต่อกับคอมพิวเตอร์ทำได้โดยการใช้ อินเตอร์เฟส การ์ด โดยใช้ข้อมูลจากบัสผ่าน ไอซี 8255 ควบคุมการเลือกโหมดโดยใช้โปรแกรมภาษาซี และโหมดการเปิด-ปิดไฟแบบต่างๆทั้ง 8 โหมดนี้จะควบคุมโดย ไมโครคอนโทรลเลอร์ PIC และข้อมูลจากเซนเซอร์ โดยคำนึงถึงการไหลของรถให้สะดวกที่สุด

Traffic Light Control System

Nuttakorn Dispong

Taweechai Surajaroenchaikul

Tosaporn Hunkaew

Kosol Chounkayan Advisor

1998

ABSTACT

This thesis has a purpose that it can help traffic problem which is a social problem. Because the traffic system that we have controlled by human for this reason it may be come to human error. So computer is the reasonable choice. This traffic light control system has 8 modes to change light sign. The unique light signal when after green light to red light this system will brink green light and change to red light , in stead of yellow light for this reason we can decision good enough because of change new pattern of traffic signal make us visible clear for long distance. The connection with computer IBM can be able by use Interface Card , Data form Bus pass 8255 control the mode selection C language is used for this thesis and mode patterns are controlled by Microcontroller PIC . This Thesis consider very much about flow rate of vehicle.

สารบัญ

บทที่ 1
บทนำ

ในปัจจุบันปัญหาสำคัญอย่างหนึ่งของเมืองใหญ่ๆของโลก ก็คงหนีไม่พ้นปัญหาด้านการจราจร เพราะในเมืองใหญ่ๆ จะมีประชากรอยู่กันอย่างหนาแน่น ทำให้มีปริมาณการคมนาคมมากกว่าในเมืองที่มีขนาดเล็ก ซึ่งปัญหาด้านการจราจรนี้อาจนำไปสู่ ปัญหาทางด้านต่างๆอีกมากมาย ทั้งปัญหาทางด้านเศรษฐกิจ ปัญหาทางด้านสุขภาพจิต ฯลฯ การคิดค้นหาทางแก้ไขปัญหานี้นับว่าเป็นสิ่งที่สำคัญยิ่ง ถ้าย้อนกลับมามองที่กรุงเทพซึ่งก็มีปัญหานี้หมักมาหลายปี ทางกระทรวงคมนาคมได้มีการว่าจ้างบริษัทต่างประเทศมาทำการศึกษาและวิจัย เพื่อทำการติดตั้งระบบควบคุมสัญญาณไฟจราจรอัตโนมัติ ซึ่งใช้งบประมาณในการจัดหาและติดตั้งอุปกรณ์ต่างๆในการนี้ค่อนข้างมาก แต่ก็ยังคงไม่สามารถแก้ปัญหานี้ให้หมดไปได้ กล่าวกันว่าเมื่อไรก็ตามที่ทางตำรวจจราจรได้เปิดระบบควบคุมสัญญาณไฟจราจรเป็นแบบอัตโนมัติได้ไม่กี่ชั่วโมง จะทำให้การจราจรในช่วงนั้นติดขัดขึ้นมาทันทีจนในที่สุดก็ต้องให้ตำรวจจราจรประจำแต่ละสี่แยกควบคุมเอง การจราจรที่ติดขัดนั้นจึงจะค่อยเบาบางลง จึงพอพูดโดยรวมได้ว่าในปัจจุบันมีการควบคุมสัญญาณไฟจราจรอยู่ 2 รูปแบบคือแบบอัตโนมัติและแบบควบคุมโดยมนุษย์ โดยทางคณะผู้จัดทำปริญญานิพนธ์นี้จะขอยึดหลักการนี้ไปใช้ในรายละเอียดต่อไป

สำหรับปริญญานิพนธ์นี้จะเป็นการเสนอความคิดริเริ่มในการแก้ปัญหาการจราจรของระบบกลุ่มสี่แยก โดยใช้กลุ่มสี่แยกบางกะปิเป็นแนวทาง ซึ่งได้ทำแบบจำลองของกลุ่มสี่แยกนี้ขึ้นมาเพื่อให้สามารถแสดงให้เห็นได้จริงและสามารถนำไปประยุกต์ใช้ได้ทันที พร้อมทั้งได้สร้างแนวคิดเกี่ยวกับดีเทคเตอร์แบบใหม่ขึ้นมาอีกด้วย

สำหรับประโยชน์ของ ปริญญานิพนธ์นี้น่าจะจุดประกายความคิดเกี่ยวกับการพัฒนาระบบจราจรในประเทศไทยได้

จากการสังเกตที่ได้กล่าวไปข้างต้นนับว่าการแก้ปัญหาจราจรในกรุงเทพนั้นค่อนข้างยากและท้าทายยิ่งทางสำหรับคณะผู้จัดทำปริญญานิพนธ์นี้ จึงได้ขอเสนอไว้ในตอนแรกว่า ปริญญานิพนธ์นี้เป็นเพียงการเสนอแนวทางและคิดรูปแบบเพื่อใช้บรรเทาปัญหาด้านการจราจรอีกทางหนึ่งเท่านั้น มิได้มุ่งหวังว่าเมื่อสิ้นสุดปริญญานิพนธ์นี้จะสามารถหาหนทางในการแก้ไขปัญหานี้ได้ 100% หากผิดพลาดประการใดทางคณะผู้จัดทำปริญญานิพนธ์นี้ต้องขออภัยมาล่วงหน้าด้วย

บทที่ 2
ประวัติและทฤษฎีเกี่ยวกับระบบการจราจร

การคมนาคมขนส่งเริ่มมีมาตั้งแต่ก่อนสงครามโลกครั้งที่2 ซึ่งได้มีการผลิตเรือเหาะและจรวดขึ้นมา เนื่องมาจากความต้องการในการพัฒนาระบบทางหลวง(Highway System) เพื่อขนส่งสินค้าทำให้เกิดการค้นหาและพัฒนาระบบทางหลวง(Highway System)โดยมีการทำระบบการขนส่งระหว่างรัฐและมีการบำรุงรักษาทางหลวง โดยสร้างจากเงินสินเชื่อและกลุ่มของผู้ใช้ถนนที่จ่ายภาษีให้ โดยAid Highway Act ในปี ค.ศ.1956 และต่อมา 20 ปีก็ได้มีการสร้างระบบขนส่งกันอย่างกว้างขวาง

มาตรฐานต่างๆเกี่ยวกับระบบการจราจรในปัจจุบันนั้นตั้งอยู่บนมาตรฐานของการจัดวางและการออกแบบอุปกรณ์ควบคุมการจราจรโดยอยู่ใน "Manual Of Uniform Traffic Control Device (MUTCP) " โดยในอเมริกาจะมีองค์กรที่เรียกว่า Federal Highway Administration เป็นคนออกหลักการสากล (MUTCD) ซึ่งจะประกอบด้วยมาตรฐานทั่วๆไปและรูปแบบสำหรับแต่ละรัฐ โดยในแต่ละรัฐก็จะมีกฎข้อปลีกย่อยที่ต่างกันแต่ต่างก็ตั้งอยู่บนพื้นฐานของ MUTCD ที่ Federal Highway Administration กำหนดไว้ทั้งสิ้น

2.1 ประวัติและปูมหลัง

ในยุคเริ่มแรกการควบคุมจะเป็นแบบแมนนวล(Manual) โดยใช้ธงคล้ายๆกับสัญญาณที่ใช้กับระบบของรถไฟโดยใช้ครั้งแรกในกรุงลอนดอน ประมาณปี 1868 แต่ว่าสัญญาณไฟฟ้าใช้ครั้งแรกในสหรัฐอเมริกา ประมาณปี 1914 โดยได้รับการพัฒนาจาก Mr. James Hoge โดยติดตั้งในคีฟแลนด์(Cleveland , Ohio) สำหรับสัญญาณหยุด (Stop Sign) ติดตั้งครั้งแรกในดีทรอยด์ (Detroit)ในปี 1915 และติดตั้งไฟจราจร 3 สีครั้งแรกในปี 1920

โดยการสร้างมาตรฐานต่างๆในอุปกรณ์ควบคุมการจราจรนั้นเกิดอยู่ในช่วงกลางปี 1920 โดยมีองค์การที่ไม่เกี่ยวข้องก็นเลย 2 องค์การต่างพัฒนาขึ้นมาเอง จนกระทั่งในปี 1927 The American Association Of State Highway Officals (AASHO) ได้พิมพ์ The Manual And Specification For The Manufacture , Display And Erection Of U.S. Standard Road Markers And Sign โดยคู่มือจะมีเฉพาะการใช้งานแค่ในเมืองและสัญลักษ์ต่างๆเท่านั้น แต่ต่อมาในปี 1930 The National Conference On Street And Highway Safety (NCSHS) ได้พิมพ์ Manual On Street Traffic Signs , Signals , And Marking ขึ้นเพื่อใช้ในแถบชานเมืองและต่างจังหวัด

ในปี 1932 ทั้งสองกลุ่มก็มาทำการประชุมร่วมกันและทำให้เกิด Joint Committee On Uniform Traffic Control Devices และพิมพ์อย่างเสร็จสมบูรณ์เป็น MUTCD ครั้งแรกในปี 1926

และกลุ่มนี้ก็ได้รับผิดชอยในการปรับปรุงเรื่อยมาจนปี 1972 เมื่อ The Federal Highway Administration ได้ก่อตั้งขึ้นเพื่อมีหน้าที่รับผิดชอบในคู่มือนี้โดย MUTCD ของแท้ดั้งเดิมได้ถูกพิมพ์ใหม่ในปี 1927 และได้มี Edition ใหม่ๆเกิดขึ้นมาอย่างมากมายในปี 1939 , 1942 ,1948 ,1954 ,1961 ,1671 ,1978 ,1988 และอาจมีฉบับใหม่ในช่วงปี 1988-2000 นี้

2.2 หลักการทั่วไปของ MUTCD

มี 5 หลักการสำหรับอุปกรณ์ควบคุมการจราจร (Traffic Control Devices) โดยอุปกรณ์ต้อง

1. ตอบสนองความต้องการจริง
2. แจ้งให้ทราบเกี่ยวกับคำสั่ง
3. มีความหมายที่แสดงได้ชัดเจนและง่ายต่อความเข้าใจ
4. คำสั่งเกี่ยวข้องโดยตรงกับผู้ใช้ถนน
5. ให้เวลาเพียงพอสำหรับการตอบสนองที่เหมาะสม

2.3 สาระสำคัญใน MUTCD

1. รายละเอียดสำหรับการออกแบบทางกายภาพของอุปกรณ์ รวมถึงขนาด , รูปร่าง , สีประวัติรูปแบบต่างๆและประวัติเฉพาะอย่าง
2. รายละเอียดของมาตรฐานและแนวทางว่าอุปกรณ์ควรติดตั้งที่ไหนหรือวางในที่ที่มีความสัมพันธ์กับถนนอย่างไร
3. การประกันหรือสถานการณ์ที่ปรับปรุงเพื่อใช้ในอุปกรณ์เฉพาะอย่าง

2.3.1 การสื่อสารกับผู้ขับขี่ยานพาหนะ

ผู้ขับขี่ยานพาหนะจะอยู่ในสภาพที่ต้องได้รับข่าวสารที่ชัดเจนและบ่อยครั้งที่มากเกินไป กลไกที่ใช้ในการสื่อความหมายต้องตระหนักถึงข้อจำกัดของมนุษย์ด้วยโดยเฉพาะเรื่องที่เกี่ยวกับระดับสายตา โดยข้อความที่ใช้สื่อความหมายต้องใช้

สี (Color) จะเป็นสิ่งที่ถูกแสดงคุณลักษณะของอุปกรณ์ได้ง่ายที่สุด สีจะถูกเห็นและวิเคราะห์เป็นระยะเวลานานกว่าที่รูปร่างจะถูกวิเคราะห์ โดยสีจะสามารถอ่านและเข้าใจได้ง่ายที่สุด สีพื้นฐานที่ใช้ในอุปกรณ์ควบคุมการจราจร (Traffic Control Devices) มีสีดังต่อไปนี้คือ สีแดง , สีเหลือง , สีเขียว , สีส้ม , สีดำ , สีน้ำเงิน , สีน้ำตาล

รูปแบบ (Pattern) รูปแบบจะใช้แสดงสัญลักษณ์เครื่องหมายต่างๆ เช่น เส้นเฉียง , เส้นประโดยใช้สื่อความหมายของสิ่งที่ใกล้เคียงกับสิ่งที่ผู้ขับขี่ยานพาหนะคุ้นเคย

รูปทรง (Shape) หลังจากสีก็มีรูปทรง , รูปร่างที่เป็นปัจจัยต่อไปที่จะมองเห็นได้ไกลที่สุดของผู้ขับขี่ยานพาหนะ ซึ่งรูปร่างมีความสำคัญมากที่จะใช้จำแนกประเภทของข้อมูลของตัวมันเอง

ตัวอักษรที่ใช้บรรยายใต้ภาพ (Legend) เป็นปัจจัยสุดท้ายที่ให้ผู้ขับขี่ยานพาหนะรวมความแล้วสรุปความหมายของอุปกรณ์ที่ได้แสดงออกมานั้น โดยตัวอักษรที่ใช้นั้นต้องสั้นและง่ายในการเข้าใจเพื่อป้องกันไม่ให้มีสิ่งอื่นมาดึงความสนใจไปจากผู้ขับขี่ยานพาหนะขณะขับรถ เพื่อสามารถที่จะเข้าใจในคำจำกัดความที่ใช้ได้

2.3.2 สัญญาณจราจร(Traffic Signal)

คำว่าสัญญาณจราจรนิยามโดย MUTCD ได้คือ การบังคับควบคุมอุปกรณ์ควบคุมการจราจรรวมทั้งสัญญาณเตือนหรือสัญญาณไฟที่สว่างอยู่ตลอดเวลา โดยการจราจรจะถูกเตือนหรือควบคุมสำหรับการกระทำหนึ่งๆ โดยสัญญาณจราจรที่เรารู้จักกันดีก็คือสัญญาณไฟตามสี่แยกต่างๆนั้นเอง โดยสัญญาณอื่นๆก็ได้แก่ สัญญาณคนข้ามถนน , สัญญาณการใช้ช่องทาง , รวมไปถึงสัญญาณเตือนต่างๆ

2.3.2.1สัญญาณควบคุมการจราจร (Traffic Control Signal)

MUTCD ออกแบบมาเพื่อเหตุผลเฉพาะอย่างในการใช้ในสัญญาณควบคุมจราจรเหตุผลต่างๆนั้นมีรายละเอียดมากมาย สำหรับหลายๆอุปกรณ์ แต่ค่าใช้จ่ายเกี่ยวกับระบบสัญญาณจราจรมีค่าสูงเมื่อเทียบกับอุปกรณ์อื่นๆ และผลเสียที่อาจเกิดขึ้นก็ยิ่งใหญ่กว่าผลของอุปกรณ์อื่นๆ

สำหรับการควบคุมสัญญาณจราจรที่ใช้มาตรฐานของ MUTCD จะทำให้เกิดประโยชน์หลายประการดังนี้

1. สามารถออกคำสั่งให้การเคลื่อนไหวต่างๆกับระบบจราจร
2. ที่ใดก็ตามที่มีลักษณะทางกายภาพเหมาะสม และมีการติดตั้งเครื่องมือวัดต่างๆสามารถเพิ่มความสามารถในการจัดการได้
3. สามารถลดปัญหาอุบัติเหตุต่างๆ ที่เกิดขึ้นเป็นประจำได้
4. ภายใต้สถานการณ์ปรกติทั่วไป สามารถทำให้เกิดความต่อเนื่องหรือใกล้เคียงกับความต่อเนื่องของการเคลื่อนที่ของระบบจราจร ณ.ความเร็วค่าจำกัดได้ตลอดเส้นทาง
5. สามารถยับยั้งการจราจรที่คับคั่งในกรณีเพื่อให้ยานพาหนะหรือคนเดินเท้าข้ามถนนไปได้

โดย MUTCD ให้คำจำกัดความหมายต่างๆ สำหรับสัญญาณ อันหลากหลายซึ่งจำเป็นต้องใช้ดังต่อไปนี้

ไฟสัญญาณวงกลมสว่างค้าง

เขียว เมื่อยานพาหนะเจอกับ " ไฟเขียวกลม (Green Ball) "สามารถผ่านแยกนั้นๆ ได้ไม่ว่าจะเพื่อเลี้ยวซ้าย , ขวา ยกเว้นมีสัญญาณข้าม , เครื่องหมายเลน หรือการออกแบบอื่นๆ และรถที่จะเลี้ยวต้องยอมให้รถทางตรงไปก่อนเสมอ สำหรับคนข้ามถนนจะข้ามได้ก็ต่อเมื่อมีสัญญาณไฟให้ข้ามเท่านั้น

เหลือง ยานพาหนะจะถูกเตือนว่าสัญญาณไฟเขียวจะหมดไปแล้ว จะมีสัญญาณไฟแดงขึ้นมาแทน การขับผ่านไฟเหลืองถือว่าถูกต้องตามกฎหมายแต่อาจมีการฝ่าไฟแดงตามมา สำหรับคนข้ามถนนจะถูกห้ามตั้งแต่มีสัญญาณไฟเหลือง

แดง เป็นสัญญาณแสดงให้ยานพาหนะห้ามผ่านสี่แยกใดๆ โดยต้องหยุดที่เส้นข้าถนน สำหรับการเลี้ยวซ้ายควรทำด้วยความระมัดระวัง สำหรับการเลี้ยวขวาจะถูกหยุดทันที

สัญญาณไฟกลมกระพริบ

เหลือง เมื่อผู้ขับรถเจอสัญญาณไฟเหลืองกระพริบอาจขับผ่านแยกใดๆ ด้วยความระมัดระวัง

แดง ใช้แทนความหมายเดียวกับหยุด

สัญญาณไฟค้างพร้อมลูกศร

เขียว การจราจรอาจมีการเคลื่อนที่ขณะขึ้นสีเขียวพร้อมลูกศรด้วยความระมัดระวัง เช่น ต้องยอมให้มีการข้ามถนนตามกฎหมาย , สัญญาณไฟเขียวพร้อมลูกศรเลี้ยวซ้าย ใช้ในกรณีที่การเลี้ยวซ้ายแสดงว่าถูกกำหนดไว้

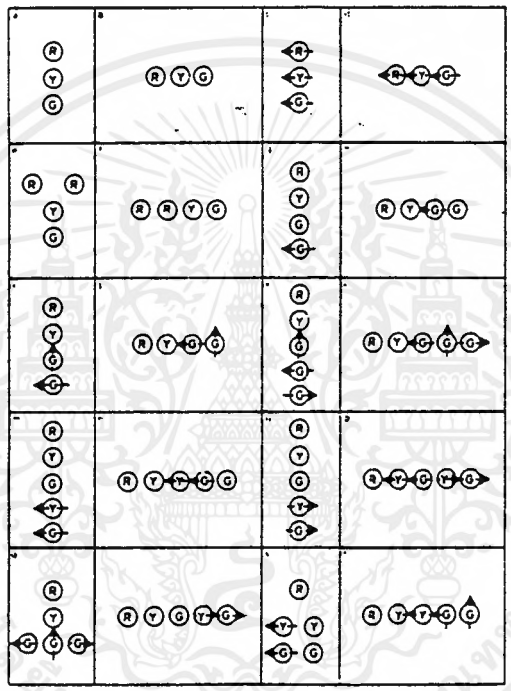
เหลือง มีความหมายเช่นเดียวกับเหลืองสว่างธรรมดา ยกเว้นมีไว้ใช้สำหรับการควบคุมการเคลื่อนที่โดยไฟเขียวพร้อมลูกศรเลี้ยว โดยจะใช้เมื่อเปลี่ยนจากสีเขียวมีลูกศรมาเป็นสีแดงมีลูกศรตามมาในทันที

แดง มีความหมายเหมือนไฟแดงสว่างธรรมดายกเว้นการใช้สำหรับการควบคุมการเคลื่อนที่ของไฟเขียวพร้อมลูกศร

สัญญาณไฟกระพริบพร้อมลูกศร

มีความหมายเหมือนไฟเหลือง , แดงกระพริบทั่วไปยกเว้นสำหรับการใช้แบบอื่นๆ โดยมีลูกศร

โดยลักษณะสัญญาณไฟจราจรออกแบบได้หลายรูปแบบ แต่ MUTCD ได้รวบรวมรูปแบบของลักษณะสัญญาณไฟที่ใช้บ่อยๆไว้ดังนี้ ซึ่งถ้าหากมีความต้องการจะรู้รายละเอียดมากขึ้นควรศึกษาเพิ่มเติมใน MUTCD

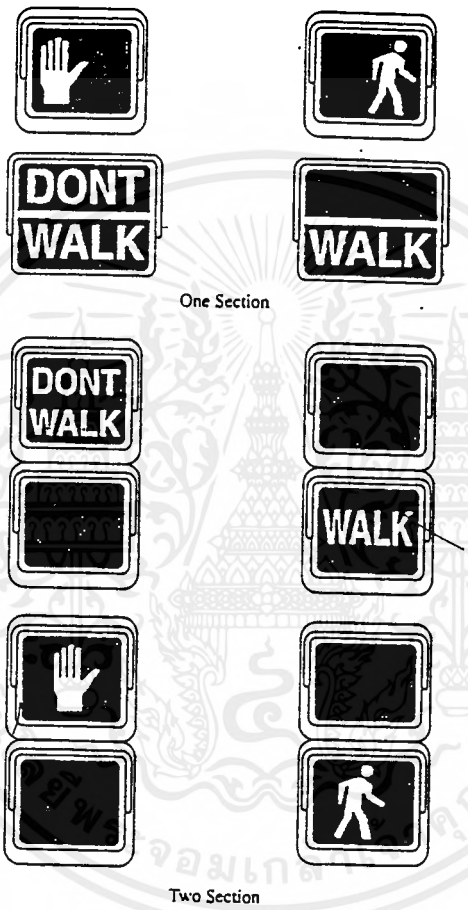


รูปที่ 2.1 แสดงรูปแบบของสัญญาณไฟจราจร

สัญญาณคนข้ามถนน

จะมีข้อมูลอย่างละเอียดมากขึ้นโดยการศึกษาเพิ่มเติมใน MUTCD โดยสามารถยก

ตัวอย่างของสัญญาณคนข้ามได้ดังรูป



รูปที่ 2.2 แสดงรูปแบบของสัญญาณคนเดินเท้า

2.4 อุปกรณ์ที่ใช้ในสัญญาณจราจรและภาคแสดงผลตามถนน

(Traffic Signal Hardware And Street Display)

อุปกรณ์ที่ใช้ในสัญญาณจราจร จะประกอบไปด้วยขอบเขตที่กว้างมากตั้งแต่จุดของวิศวกรระบบจราจร ไปจนถึงฟังก์ชั่นต่างๆของอุปกรณ์ดังนี้

1. เครื่องมือที่ใช้ในการควบคุมสี่แยก(Intersection Controllers)
2. เครื่องมือที่ใช้แสดงผลตามสี่แยก(Intersection Display Hardware)

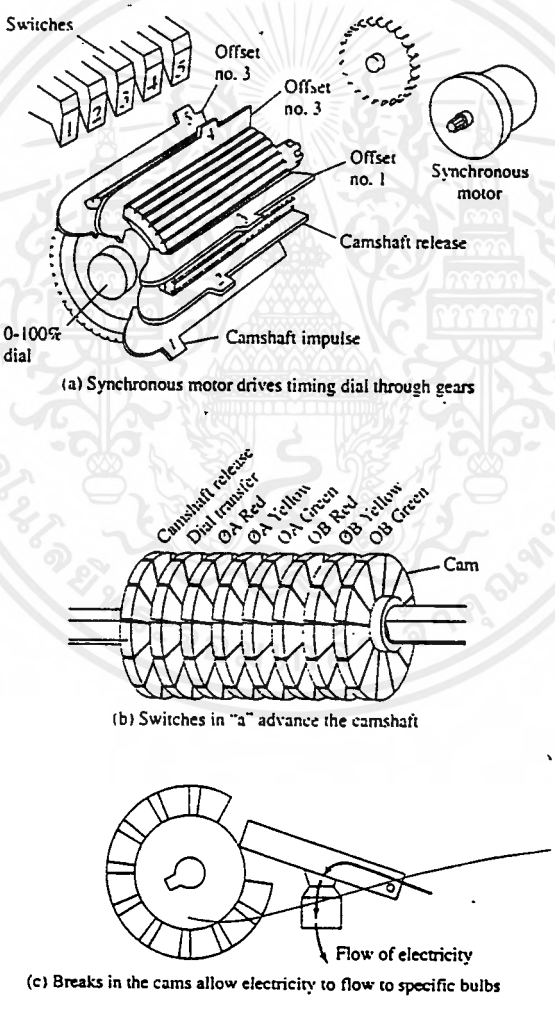
3. เครื่องมือควบคุมอุปกรณ์ระบบทั้งสี่แยก(Arterial And System Coordination Hardware)

4. เครื่องตรวจจับ(Detectors)

2.4.1. Intersection Signal Controllers

จะแบ่งออกเป็นสองแบบคือ

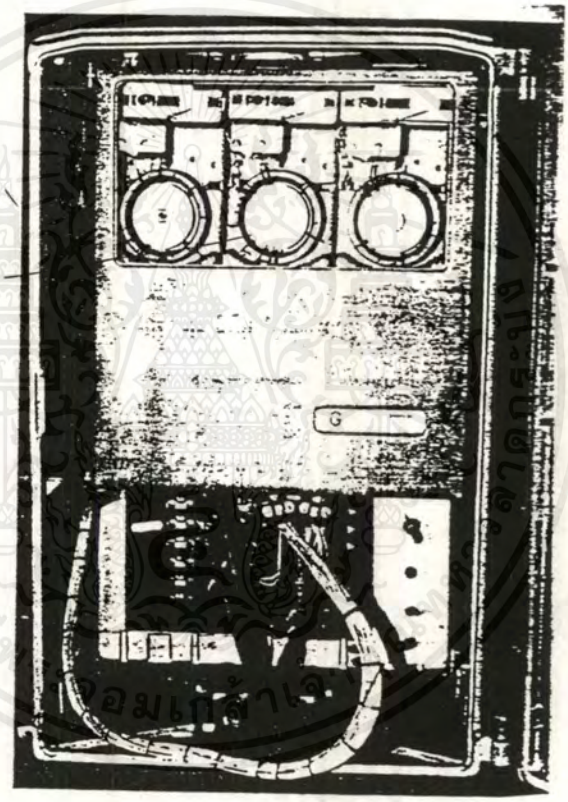
24.1.1 แบบกำหนดเวลาไว้ล่วงหน้า (Pretime) คือเครื่องควบคุมที่ให้ค่าคาบเวลาคงตัวลำดับเฟสคงตัวและความยาวเฟสคงตัว โดยเครื่องควบคุมเครื่องเดียวอาจให้เวลาได้ต่างกันหลายรูปแบบสำหรับช่วงเวลาที่ต่างกันในหนึ่งวัน เครื่องควบคุมแบบนี้ไม่ต้องการเครื่องตรวจจับ



รูปที่ 2.3 แสดงส่วนประกอบพื้นฐานของ Electromechanical Controller

Electromechanical Pretimed Controllersในยุคแรกๆเครื่องควบคุมกึ่งอิเล็กทรอนิกส์กึ่งเครื่องจักรกลแบบกำหนดเวลาไว้ล่วงหน้าแสดงให้เห็นได้ดังรูป 2.3

โดยมีซิงค์โครนัสมอเตอร์ที่พันรอบไว้คงที่ (กำหนดโดยความถี่ของแหล่งจ่ายไฟ) และมีช่วงเวลาของผู้ขับรถที่กำหนดโดยไทม์มิ่ง เกียร์ (Timing gear) โดยไทม์มิ่ง เกียร์สามารถเปลี่ยนแปลงได้ง่ายมากโดยใช้อัตราส่วนของเกียร์เป็นตัวกำหนดรอบความเร็วของการหมุนของช่วงเวลา และความยาวของช่วงเวลาและในรูปที่ 2.4 แสดงตู้ควบคุมสี่แยกที่ใช้ระบบเกียร์แบบเก่านี้ โดยในรูปจะมีสามช่วงเวลาที่ได้กำหนดไว้ตายตัว โดยจะอยู่ที่การเลือกใช้ว่าจะใช้ช่วงเวลาแบบใด ในช่วงใดของวัน

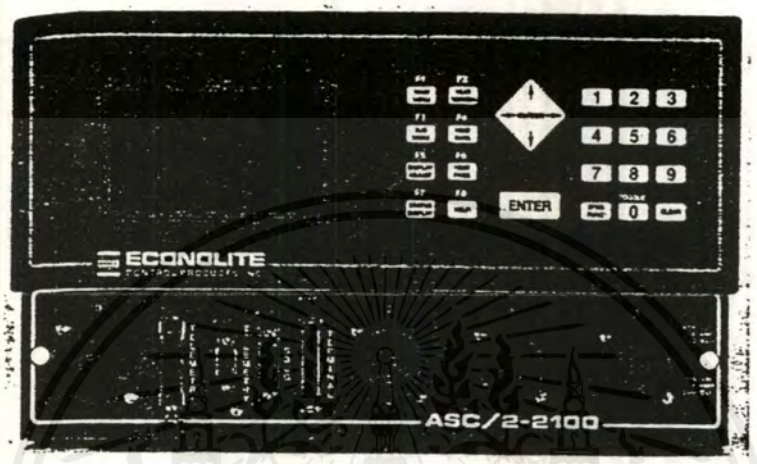


รูปที่ 2.4 แสดงเครื่องควบคุมแบบ Electromechanical

2.4.1.2แบบปรับค่าได้ (Actuated) โดยการทำงานของเครื่องควบคุมแบบนี้ส่งผลเป็นไซเคิลต่อไซเคิลพื้นฐานโดยการกระแสที่ต้องการ ดังนั้นเครื่องควบคุมแบบนี้ต้องการตัวตรวจจับ เพื่อการเก็บข้อมูล

Modern Actuated Controllers เครื่องควบคุมแบบปรับค่าได้ยุคใหม่ ได้เกิดขึ้นเนื่องจากเทคโนโลยีได้ก้าวหน้าไปเร็วมากในตลาดของการแข่งขันดังแสดงให้เห็นได้ดังรูป 2.5

ซึ่งแสดงให้เห็นถึงเครื่องควบคุมที่เพิ่งออกมาสู่ตลาดในปี 1996 ซึ่งมีลูกเล่นหลายอย่างโดยเปิดฝาครอบออกเห็นวงจรอิเล็กทรอนิกส์ภายในดังในรูป 2.6 ซึ่งมีคุณลักษณะดังต่อไปนี้



รูปที่ 2.5 แสดงเครื่องควบคุมชุดใหม่แบบปรับค่าได้

รูปที่ 2.6 แสดงเครื่องควบคุมแบบเปิดหน้ากากออก

อนุญาติให้เครื่องควบคุมได้ถึง 12 เฟส , 8 กลุ่มสี่แยก ที่มีผลต่อกันและ 2 สัญญาณเวลา และมีมาตรฐานทุกอย่างตาม (NEMA (The National Electrical Manufactures Association) ซึ่งเป็นองค์การหนึ่งในสหรัฐอเมริกา ซึ่งทำหน้าที่กำหนดการออกแบบส่วนต่างๆในระบบควบคุมการจราจร , มี 64 รูปแบบ ของสี่แยกจราจรและมีการเชื่อมต่อได้ 3 แบบโดยมีอุปกรณ์เสริมคือ มีความสามารถเชื่อมต่อกับอุปกรณ์ตรวจจับได้ 64 ตัว , มีการสื่อสารด้วยโมเด็มขนาด 1200 บิทต่อวินาที และมีเมนูรายการที่สามารถแสดงผลได้บนหน้าจอช่วยเหลือการใช้งาน

2.5 เครื่องตรวจจับ (Detector)

ในหลายๆปีที่ผ่านมาได้มีการใช้เครื่องตรวจจับที่หลากหลายเพื่อใช้ในการสื่อข้อมูลไปยังเครื่องควบคุมสัญญาณจราจรแบบปรับค่าได้และมีการเก็บข้อมูลสะสมไว้ มีดังต่อไปนี้

1. เครื่องตรวจจับ แบบขดลวดไฟฟ้า(Inductive Loop Detectors)

2.. เครื่องตรวจจับ แบบแม่เหล็ก(Magnetic Detectors)

3. เครื่องตรวจจับ แบบสนามแม่เหล็ก(Magnetometers)

4. เครื่องตรวจจับ แบบเรดาห์(Radar Detectors)

5. เครื่องตรวจจับ แบบคลื่นเสียง(Ultrasonic Detectors)

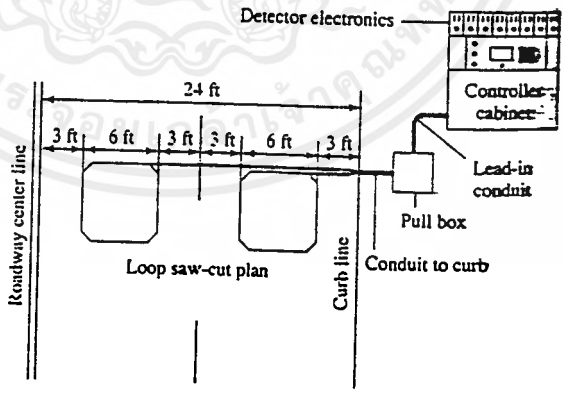
6. เครื่องตรวจจับ แบบตรวจแสง(Photocell Detectors)

7. เครื่องตรวจจับ แบบแรงดัน(Pressure Pads)

เครื่องตรวจจับที่นิยมใช้มากที่สุด คือ เครื่องตรวจจับแบบ ขดลวดไฟฟ้า

2.5.1เครื่องตรวจจับ แบบขดลวดไฟฟ้า(Inductive Loop Detectors)

มักจะสร้างขดลวดตัวนำประมาณ 2-3 รอบ วางอยู่ในช่วงของถนน และมีสายต่อไปยังเครื่องขยายกำลัง (Amplifier) และออสซิลเลเตอร์ (Oscillators) ในตู้ควบคุม โดย ออสซิลเลเตอร์ จะทำหน้าที่จ่ายพลังงานให้กับ Loop เมื่อรถยนต์ผ่านมาเหนือ Loop จะทำให้เกิดการเหนี่ยวนำขึ้นที่ Loop Inductance ซึ่งจะทำให้เกิดการเพิ่มขึ้นของความถี่ออสซิลเลเตอร์ ซึ่งค่าความถี่ที่เพิ่มขึ้นนี้จะส่งเป็นพลังงานไฟฟ้าไปยังเครื่องควบคุมต่อไป



รูปที่ 2.7 การติดตั้งทั่วไปของระบบดีเทคเตอร์

ในรูปที่ 2.7 แสดงรูปมาตรฐานบนถนน 2 เลน (โดยแต่ละเลนจะต้องมีเครื่องตรวจจับของตัวเอง โดยที่มุมจะถูกตัดออกดังรูป) เพื่อไม่ให้เป็นการยากต่อการติดตั้งซึ่งบางบริษัท อาจวาง Inductive Loop ไว้ในท่อพลาสติกเพื่อ ณ.บริเวณติดตั้ง จะมีการผนึกด้วย อีพอกซี่ (Epoxy) หรือสารอื่นๆ หลังจากใส่ Inductive Loop เข้าไปแล้วแต่ปัจจุบันมีการสร้างเป็น Inductive Loop สำเร็จรูปลงในบล็อคคอนกรีต

2.5.2 เครื่องตรวจจับ แบบแม่เหล็ก(Magnetic Detectors)

มีอยู่ 3 แบบด้วยกันคือ แบบเครื่องตรวจจับระบบแม่เหล็กมาตรฐาน (Standard), แบบเครื่องตรวจจับระบบแม่เหล็กโดยตรง (A Directional Magnetic Detectors) และแบบแมกนีโตรมิเตอร์ (MagnetoMeter) โดยทั้ง 3 แบบจะประกอบไปด้วย 2 ส่วนหลักๆ คือ เซนเซอร์ในถนน และ ส่วนภาควงจรขยาย ถึงแม้ว่าทั้ง 3 แบบจะตั้งอยู่บนพื้นฐานของการเปลี่ยนแปลงของเส้นฟลักซ์จากสนามแม่เหล็กโลกก็ตาม แต่สำหรับเครื่องตรวจจับแบบแมกนีโตมิเตอร์ จะเป็นแบบที่พิเศษกว่าแบบทั่วไปดังจะอธิบายต่อไปนี้สำหรับแบบเครื่องตรวจจับสนามแม่เหล็กโดยตรงนี้จะใช้คอยล์ (Coil) ที่ทำมาจากขดลวดกับแกนที่มีค่าซึมซับสูง วางลงไปในท่อที่ฝังไว้ใต้พื้นผิวของถนน

เมื่อมีรถผ่านมาใกล้และผ่านไปเหนือคอยล์ เส้นฟลักซ์ที่คงที่ จะถูกผ่านไปยังคอยล์จะถูกทำให้เบี่ยงเบนไปจากทิศทางเดิม ทำให้เกิดการเปลี่ยนแปลงของระดับแรงดันภายในคอยล์ ซึ่งวงจรที่มีกำลังขยายสูงจะทำการขยายแรงดันเพื่อที่จะทำการส่งผ่านไปยังเครื่องควบคุมซึ่งจะเข้าใจได้ว่ามีรถยนต์วิ่งผ่านมา สำหรับเครื่องตรวจจับแบบนี้รถยนต์ต้องมีการเคลื่อนไหว สำหรับรถยนต์ที่มีความเร็วน้อยกว่า 5 ไมล์ต่อชั่วโมง โดยทั่วไปแล้วจะไม่สามารถตรวจจับได้ สำหรับเครื่องยนต์แบบนี้จะสามารถให้ข้อมูลจากการเคลื่อนที่ได้ แต่จะไม่สามารถเก็บข้อมูลหรือแสดงข้อมูลได้

2.5.3 เครื่องตรวจจับ แบบสนามแม่เหล็ก(Magnetometers)

เครื่องแมกนีโตมิเตอร์ จะเป็นเครื่องตรวจจับระบบสนามแม่เหล็กที่พิเศษกว่าแบบอื่นซึ่งถูกออกแบบมาเพื่อแสดงถึงข้อมูลของรถที่ได้จากการวัดผลการเปลี่ยนแปลงจากสนามแม่เหล็กโลก เมื่อรถยนต์เข้ามาใกล้เครื่องตรวจจับ

โดยเซนเซอร์ที่ติดตั้งไว้ในถนนจะมีขนาดเล็กโดยมีลักษณะคล้ายกระป๋อง ที่สามารถวางไว้ในแนวตั้งในหลุมที่ขุดลงไปใต้ถนนซึ่งจะถูกเชื่อมต่อมายังภาคขยายซึ่งจะติดตั้งอยู่ในตู้ควบคุม

2.5.4 เครื่องตรวจจับ แบบเรดาห์(Radar Detectors)

สำหรับเครื่งตรวจจับแบบเรดาห์นั้นจะตั้งอยู่บนพื้นฐานของดรอปเปอร์(Doppler) โดยคลื่นไมโครเวฟจะถูกยิงเป็นเส้นตรงไปยังถนน เมื่อมีรถยนต์ผ่านมาจะทำให้เกิดดการสะท้อนของลำคลื่นไมโครเวฟไปยังเสาอากาศในความถี่ที่ต่างออกไปจากเดิม แต่สำหรับการตรวจจับแบบนี้จะสามารถตรวจจับได้แค่มีการผ่านของรถหรือไม่เท่านั้นอีกทั้งยังต้องมีการขอคลื่นความถี่ใช้งานอย่างถูกต้องอีกด้วย รวมทั้งยังต้องมีการซ่อมแซมและการบำรุงรักษาที่ซับซ้อนยุ่งยาก

2.5.5 เครื่องตรวจจับ แบบคลื่นเสียง (UltraSonic Detectors)

เป็นวิธีการที่ดีอย่างหนึ่งที่ใช้กันในยุค 1970 แต่จากในอดีตพบว่าเป็นการยากที่จะปรับบริเวณที่จะถูกตรวจจับให้ถูกต้อง รวมทั้งมีปัญหาเรื่องพื้นผิวของทางเดินรถอีกด้วยโดย

เซนเซอร์แบบนี้จะส่งคลื่นอัลต้าโซนิกส์ไปยังทางเท้าซึ่งจะสะท้อนกลับมายังต้นกำเนิดอีกครั้ง โดยช่วงเวลาที่จะสะท้อนกลับมาจะต้องถูกปรับแต่งให้เหมาะสม เมื่อมีรถเข้ามาในพื้นที่ของกรวยลำ อัลต้าโซนิกส์ มันจะสะท้อนคลื่นกลับมาเร็วกว่าปรกติเพราะหลังคารจะอยู่ใกล้กับอัลต้าโซนิกส์ดีเทกเตอร์มากกว่าพื้นถนนนั้นเอง

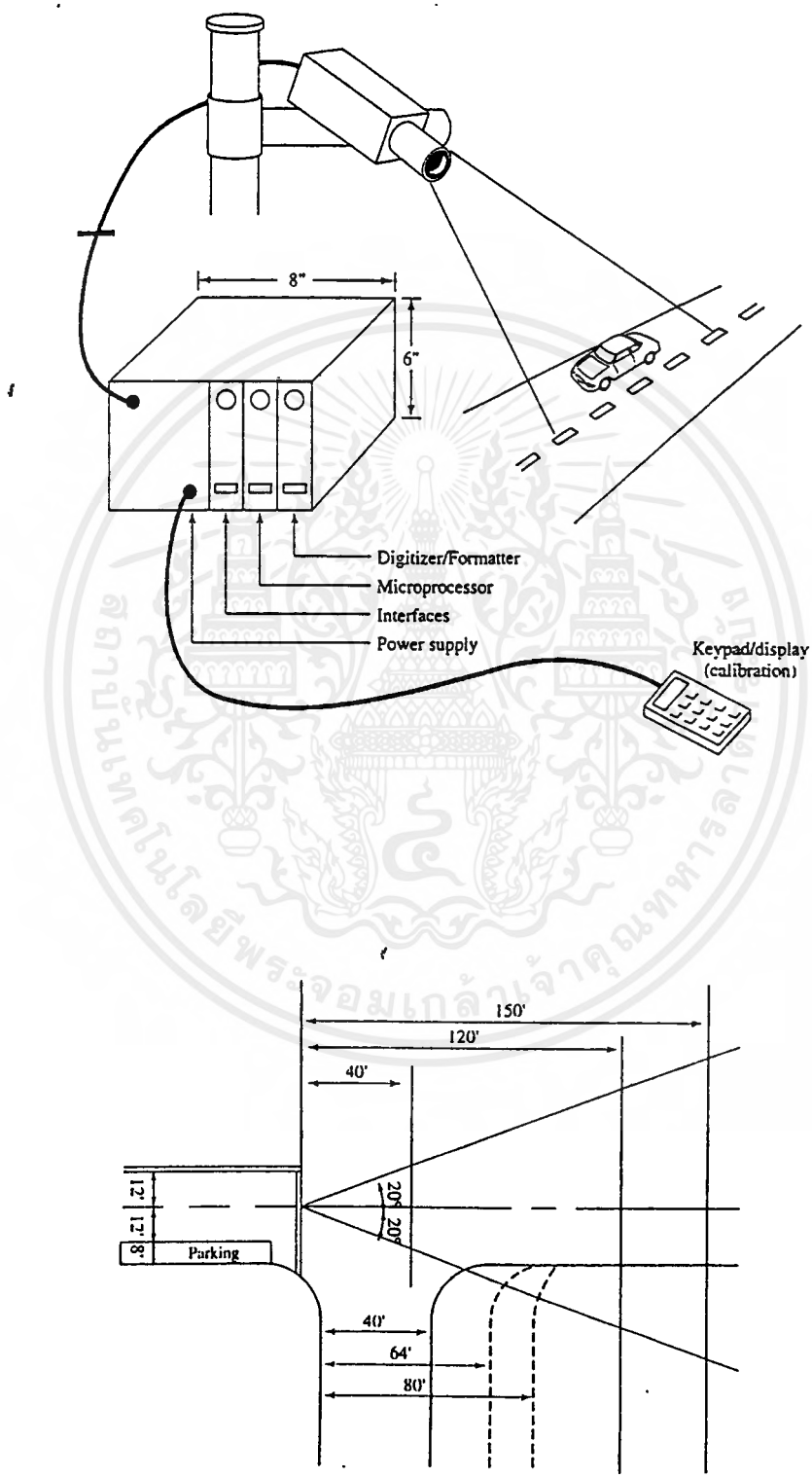
โดยปัญหาหลักของอัลต้าโซนิกส์ดีเทกเตอร์ก็คือจะวางอย่างไรให้เหมาะสมกับการทำงานและปรับกรวยดีเทกเตอร์อย่างไรให้เหมาะสมกับแต่ละสี่แยกที่แตกต่างกัน โดยต้องตรวจจับรถที่เข้ามาใกล้เลนที่ติดตั้งได้

2.5.6 เครื่องตรวจจับ แบบแรงดัน(Pressure Pads)

มีการใช้งานมาหลายปีแล้วแต่ในปัจจุบันนี้ไม่นิยมใช้โดยมันต้องการโครงโลหะที่ติดตั้งลงไปในถนนเพื่อรองรับให้กับแผ่นโลหะความดัน เมื่อมีน้ำหนักของรถกดลงบนแผ่นโลหะที่หุ้มด้วยยางไว้ด้านบนจะเกิดการสัมผัสของรอยต่อของโลหะภายในซึ่งจะส่งสัญญาณไปยังเครื่องควบคุม แต่เครื่องตรวจจับแบบนี้มีค่าติดตั้งที่แพงและโดยปรกติแล้วมักต้องติดตังใหม่เสมอเมื่อมีการทำถนนใหม่

สำหรับการวิวัฒนาการในปัจจุบันระบบที่มีความครอบคลุมและระบบอุปกรณ์ในอนาคตคือระบบ Wide-Area-Video-Imaging การใช้วิดีโอ กลายมาเป็นสิ่งที่ใช้เล่าเรื่องราวในปัจจุบันพร้อมด้วยการพัฒนาของซอฟแวร์ที่เหมาะสม และในกรณีนี้เป็นไปได้ที่จะกำหนดให้เครื่องตรวจจับติดตั้งที่ใดก็ได้พร้อมกับกล้องวีดิโอ โดยรถยนต์จะถูกตรวจจับจากความเข้มของแสง ณ. บริเวณที่ตรวจจับ (ภายในรัศมีที่กล้องวีดิโอมองเห็น) เปลี่ยนแปลงไป โดยในกรณีนี้ต้องการความเข้มของภาพของสภาวะแวดล้องที่คงที่และสามารถปรับแต่งใหม่ได้สำหรับสภาพแสงที่แตกต่างกัน แต่มีข้อควรระวังเกี่ยวกับลักษณะการวางของมุมกล้องให้ดี เพื่อป้องกันการตรวจจับได้เป็นเงาของรถแทนที่จะเป็นตัวรถเอง โดยสามารถต่อกับไมโครโปรเซสเซอร์ ในเครื่องควบคุมเพื่อให้สามารถควบคุมการปรับแต่งได้

ประโยชน์อันยิ่งใหญ่ของเทคโนโลยีการตรวจจับแบบนี้ คือความสามารถในการครอบคลุมพื้นที่เป็นบริเวณกว้าง และครอบคลุมหลายช่องทางด้วยอุปกรณ์เพียงตัวเดียวทั้งยังอาจจะระบุสถาที่ ที่เครื่องตรวจจับแต่ละตัวเพื่อง่ายต่อการเปลี่ยนการควบคุมจากศูนย์คอมพิวเตอร์โดยท่ไม่ต้องมีอุปกรณ์ใดๆ ติดตั้งอยู่ที่ถนนเลยดังรูป



รูปที่ 2.8 แสดงลักษณะการติดตั้งของระบบ Wide-Area Video imaging Detector

บทที่ 3
ทฤษฎีที่ใช้ในการออกแบบ

3.1 แนวทางการออกแบบการทำงานของระบบควบคุมสัญญาณไฟจราจร

จากในตอนแรกของบทนำได้กล่าวไปแล้วว่าในปัจจุบันระบบควบคุมสัญญาณไฟจราจรนั้นพอที่จะแบ่งออกได้เป็นสองรูปแบบคือ

1.ระบบควบคุมสัญญาณไฟจราจรแบบอัตโนมัติ

ซึ่งระบบนี้จะใช้การสั่งงานจากคอมพิวเตอร์ศูนย์กลางเป็นหลักโดยตำรวจจราจรไม่ต้องคอยมาควบคุมเองตามสี่แยก โดยจะมีระบบกล้องคอยตรวจสอบความหนาแน่นของปริมาณรถให้ทางศูนย์กลางคอยดูเป็นข้อมูล

2.ระบบควบคุมสัญญาณไฟจราจรแบบManual

ระบบนี้ในแต่ละสี่แยกตำรวจจราจรต้องมานั่งควบคุมเองตลอดเวลา

สำหรับในตอนแรกจะขอกล่าวถึงแนวทางในการหารูปแบบที่น่าจะช่วยแก้ไขปัญหานี้ได้ โดยสามารถแบ่งออกได้เป็น 2 ขั้นตอนดังนี้

3.1.1 แนวคิดเรื่องลักษณะสัญญาณไฟจราจร

ในปัจจุบันการควบคุมระบบสัญญาณไฟจราจรจะมีรูปแบบดังต่อไปนี้

สัญญาณไฟบอกให้หยุดคือ

เขียว ⟶ เหลือง ⟶ แดง

มีความหมายคือเมื่อสัญญาณไฟจราจรเปลี่ยนจากเขียวเป็นเหลือง เพื่อให้คนขับรถระวังตัวและพร้อมที่จะชะลอรถ และเมื่อสัญญาณไฟจราจรเปลี่ยนจากเหลืองเป็นแดงคือคนขับรถต้องหยุดรถ

สัญญาณไฟบอกให้ไปได้คือ

แดง ⟶ เขียว

มีความหมายคือเมื่อสัญญาณไฟจราจรเปลี่ยนจากแดงเป็นเขียวคือให้คนขับรถออกตัวได้จากระบบสัญญาณไฟจราจรข้างต้นอาจเกิดปัญหาตามมาดังนี้

1.เมื่อเป็นสัญญาณบอกให้หยุดเมื่อสัญญาณไฟจราจรเปลี่ยนจากเขียวเป็นเหลือง คนขับรถส่วนใหญ่มักจะไม่ยอมชะลอความเร็วลงเพื่อเตรียมตัวที่จะหยุดรถ กลับพยายามที่เร่งความเร็วเพื่อที่จะไปให้ทันก่อนสัญญาณไฟแดงจะสว่างขึ้น ทำให้เมื่อสัญญาณไฟเปลี่ยนเป็นสีแดงรถจะชะลอไม่ทันและเกิดการฝ่าไฟแดงอยู่บ่อยครั้ง ทำให้อาจเกิดการชนกันของรถที่ถูกปล่อยมาจากอีกช่องทางที่สัญญาณเปลี่ยนจากแดงเป็นเขียวได้

2.ในทางกลับกันเมื่อสัญญาณไฟจราจรเปลี่ยนจากแดงเป็นเขียวคนขับรถมักจะออกตัวก่อนที่สัญญาณจะเปลี่ยนจากแดงเป็นเขียวเสมอทำให้อาจเกดการชนกันถ้าอีกช่องทางหนึ่งสัญญาณยังไม่เปลี่ยนจากเหลืองเป็นแดงได้

ดังนั้นจากปัญหาดังกล่าวข้างต้นจึงได้ลองออกแบบลักษณะของสัญญาณไฟจราจรใหม่ดังนี้

สัญญาณไฟจราจรบอกให้ไปได้ จะยังคงใช้แนวทางเดิม

สัญญาณไฟบอกให้หยุดจะใช้สัญญาณไฟจราจรในรูปแบบใหม่คือ

เขียว ⟶ เขียวกระพริบ ⟶ แดง

และเปลี่ยนจากในหนึ่งช่องทางที่มีไฟจราจรสำหรับ 1เลนมี 3 ดวงคือ เขียว , เหลือง, แดงไปเป็น 1 เลนมี 2 ดวงคือ เขียว กับ แดงเท่านั้น และในแต่ละช่องทางการจราจร(มักมี 3 เลน คือ เลี้ยวซ้าย , ตรงไป , เลี้ยวขวา) จะมีสัญญาณไฟจราจรทั้งหมด 6 ดวงดังรูป

แดง	แดง	แดง
เขียว	เขียว	เขียว

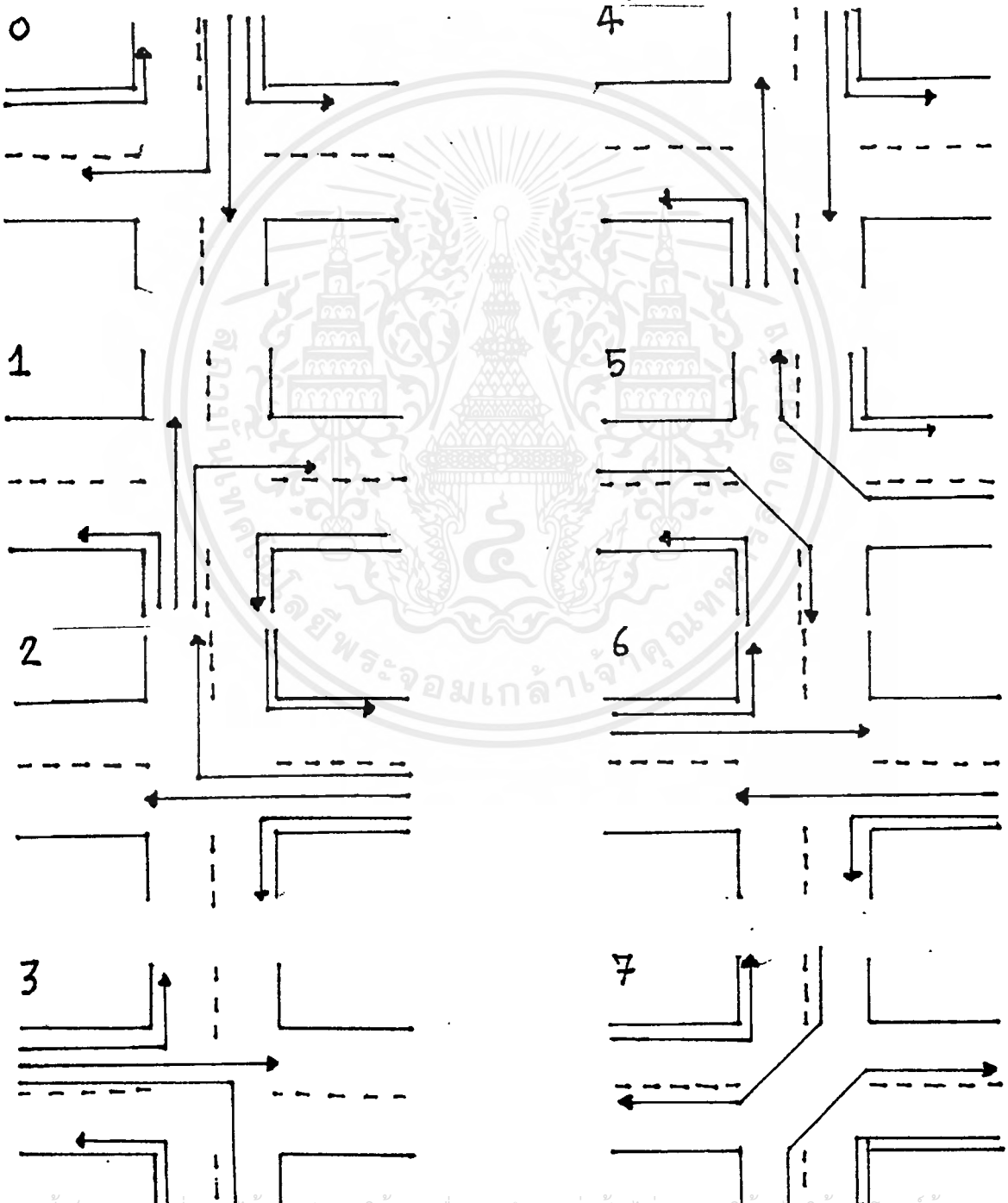
โดยในแต่ละช่วงสัญญาณจะใส่เวลาหน่วงลงไปสักครู่หนึ่งโดยในขณะที่ช่องทางหนึ่งเปลี่ยนสัญญาณไฟจากเขียวเป็นแดงเสร็จแล้ว ในอีกช่องทางที่จะให้สัญญาณไฟบอกให้ไปได้จะยังคงแดงอยู่คือยังไม่ให้สัญญาณไปจนกว่าเวลาจะผ่านระยะเวลาหนึ่งก่อน จึงจะให้สัญญาณไฟจากหยุดเป็นให้ไปได้ โดยระยะเวลาหน่วงสำหรับแต่ละสี่แยกไม่ควรเท่ากันเนื่องจากว่ามีปัจจัยหลายอย่างไม่เหมือนกันซึ่งจะเป็นเวลาเท่าไรนั้นจะได้จากการสังเกตจากของจริงเท่านั้น ซึ่งการเปลี่ยนลักษณะของสัญญาณในรูปแบบนี้จะมีข้อได้เปรียบลักษณะสัญญาณไฟจราจรแบบเก่าอยู่2 ประการคือ

1.ช่วยเพิ่มระยะเวลาห่างของการเปลี่ยนสัญญาณไฟจราจรในแต่ละช่องทาง ซึ่งน่าจะช่วยลดปัญหาที่คนขับรถไม่ยอมชะลอรถเมื่อสัญญาณเปลี่ยนจากเขียวเป็นแดง ไม่ให้ช่วงเวลานี้ไปซ้อนทับกับช่วงเวลาของอีกช่องทางที่สัญญาณเปลี่ยนจากแดงเป็นเขียวได้

2.เมื่อเปลี่ยนลักษณะสัญญาณไฟจราจรเป็นแบบนี้จะช่วยเพิ่มความรัดกุมในการตัดสินใจมากขึ้น

3.1.2 รูปแบบสัญญาณไฟจราจร

จากการที่ได้สังเกตความจำเป็นต่างๆในการรถปล่อยรถตามสี่แยกแล้วก็พอที่จะสามารถหารูปแบบของการปล่อยรถได้เท่าที่จำเป็นมาด้วยกัน 8 รูปแบบดังนี้



รูปที่ 3.1 แสดงรูปแบบต่างๆของสัญญาณไฟในระบบสี่แยก

สำหรับสี่แยก

แบบที่ 0 ใช้ในกรณีทั่วๆไปที่รถในช่องทาง 1 มีความหนาแน่นมากที่สุด

แบบที่ 1 ใช้ในกรณีทั่วๆไปที่รถในช่องทาง 2 มีความหนาแน่นมากที่สุด

แบบที่ 2 ใช้ในกรณีทั่วๆไปที่รถในช่องทาง 3 มีความหนาแน่นมากที่สุด

แบบที่ 3 ใช้ในกรณีทั่วๆไปที่รถในช่องทาง 4 มีความหนาแน่นมากที่สุด

แบบที่ 4 ใช้ในกรณีที่รถเลี้ยวขวาของช่องทาง 1 และ 3 มีความหนาแน่นน้อยกว่าทางตรงและเลี้ยวซ้าย

แบบที่ 6 ใช้ในกรณีที่รถเลี้ยวขวาของช่องทาง 2 และ 4 มีความหนาแน่นน้อยกว่าทางตรงและเลี้ยวซ้าย

แบบที่ 5 ใช้ในกรณีที่รถเลี้ยวขวาและซ้ายของช่องทาง 1 และ 3 มีความหนาแน่นมากกว่าทางตรง

แบบที่ 7 ใช้ในกรณีที่รถเลี้ยวขวาและซ้ายของช่องทาง 1 และ 3 มีความหนาแน่นมากกว่าทางตรง

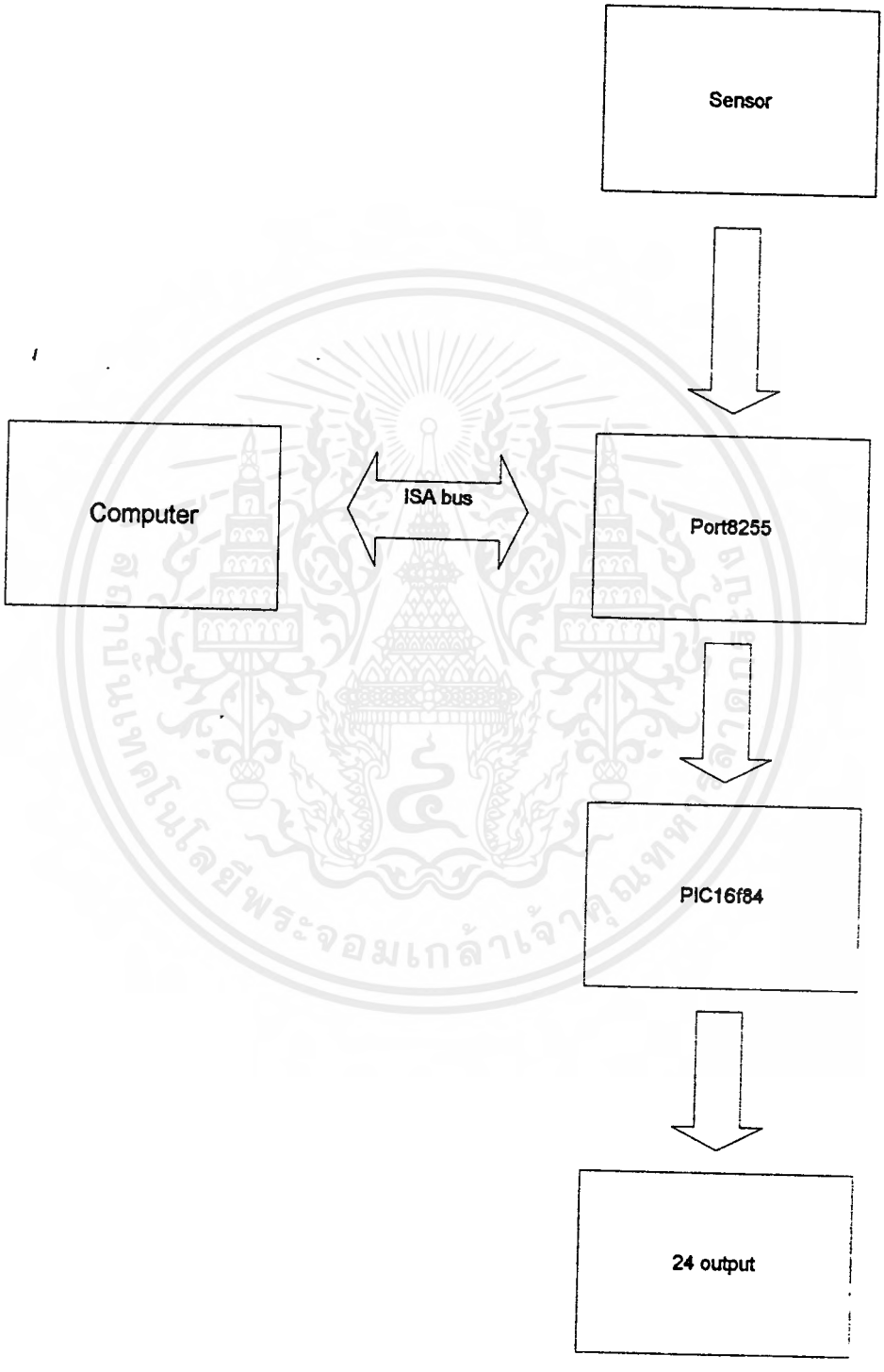
โดยที่จะสามารถรวบรวมชุดของรูปแบบเป็นโหมดการทำงานของสัญญาณไฟจราจรได้ 3 โหมดดังนี้

โหมดที่ 1 คือการวนของรูปแบบที่ 0 , 1 , 2 , 3 ตามลำดับ

โหมดที่ 2 คือการวนของรูปแบบที่ 4 , 5 , 6 , 7 ตามลำดับ

โหมดที่ 3 คือการวนของรูปแบบที่ 1 , 4 , 0 , 2 , 6 , 3 ตามลำดับ

โดยมีบล็อกไดอะแกรมของระบบควบคุมสัญญาณไฟจราจรดังรูปที่ 3.2



รูปที่ 3.2 แสดงบล็อกไดอะแกรมของระบบควบคุมสัญญาณไฟจราจร

สำหรับสามแยก

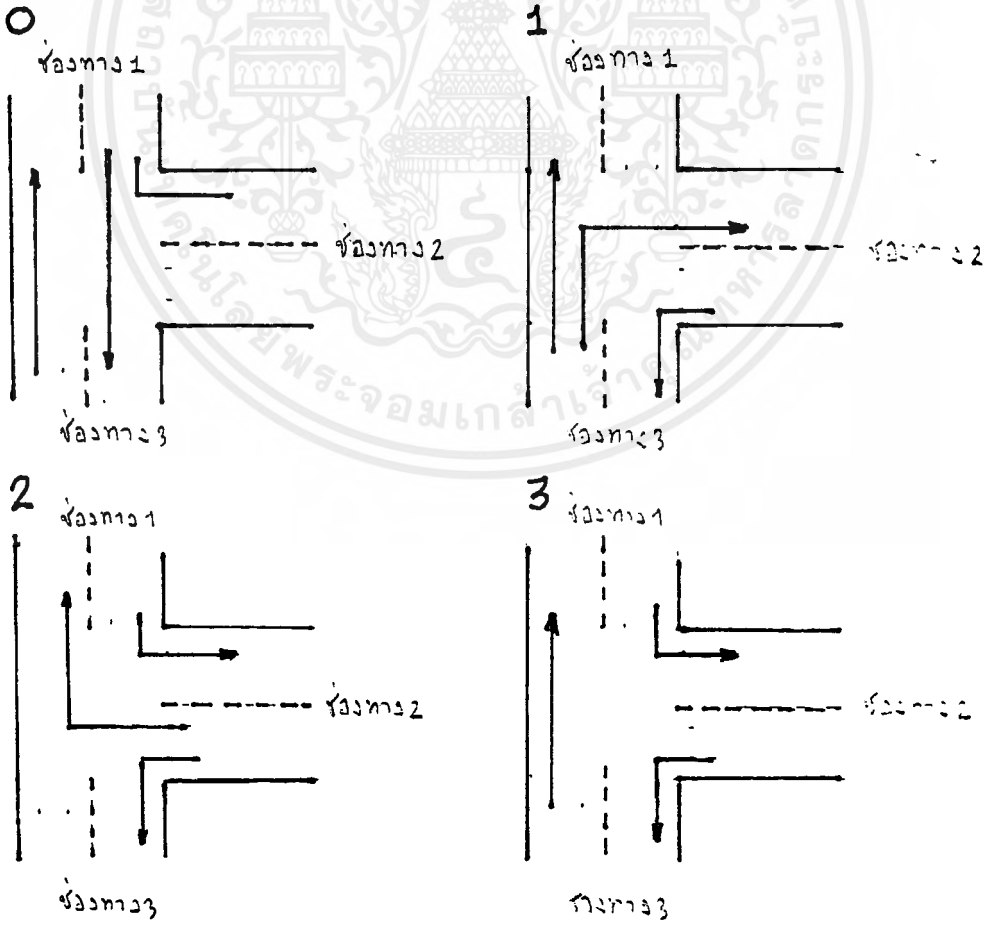
แบบที่ 0 คือปล่อยให้รถช่องทาง 1 และ 2 วิ่งสวนกัน โดยให้ช่องทาง 1 เลี้ยวซ้ายได้

แบบที่ 1 คือปล่อยให้รถช่องทาง 3 วิ่งได้ทุกเส้นทาง โดยช่องทางที่ 2 เลี้ยวซ้ายได้

แบบที่ 2 คือปล่อยให้รถช่องทาง 2 วิ่งได้ทุกเส้นทาง โดยช่องทางที่ 1 เลี้ยวซ้ายได้

แบบที่ 3 คือปล่อยให้รถช่องทาง 3 วิ่งตรงโดยช่องทางที่ 1 และ 2 เลี้ยวซ้ายได้

สำหรับสามแยกจะมีเพียงโหมดเดียวเท่านั้น คือการรวมของรูปแบบที่ 0 , 1 , 2 , 3



รูปที่ 3.3 แสดงรูปแบบต่างๆของสัญญาณไฟจราจรที่ใช้สำหรับสามแยก

3.2 การ อินเตอร์เฟส กับ คอมพิวเตอร์ ตระกูล 80x86

การติดต่อกับ คอมพิวเตอร์ นั้นจริงๆแล้ว เราต้องทำการติดต่อในระดับของภาษาเครื่องเท่านั้น ซึ่งก็จะต้องติดต่อในรูปของเลขฐานสองเป็นลำดับของตัวเลข 0 1 0 1 ไปเรื่อยๆ ขึ้นอยู่กับว่าจะเป็นการติดต่อแบบขนาน หรือแบบอนุกรม ซึ่ง คอมพิวเตอร์ จะสามารถเข้าใจได้เองเพราะวงจรภายในของคอมพิวเตอร์ก็เป็นวงจร ดิจิตอล ซึ่งประกอบไปด้วยวงจร ลอจิกเกต ที่ทำงานเมื่อมีระดับสัญญาณเข้าเป็น 1(5 โวลต์) หรือ มีระดับสัญญาณเข้าเป็น 0(0 โวลต์) แต่สำหรับคนเรานั้นไม่สามารถเข้าใจภาษาที่เป็นตัวเลขนี้ได้ จึงได้มีการคิดค้นตัวแปรภาษาจากภาษาเครื่องไปเป็นภาษาที่มนุษย์สามารถเข้าใจได้ง่ายเช่น ภาษา C ภาษา ปาสคาล ภาษา เบสิก เป็นต้น

3.2.1 โครงสร้างและสถาปัตยกรรมของ คอมพิวเตอร์ ตระกูล 80x86

การติดต่อกับ คอมพิวเตอร์ นั้นเราจำเป็นที่จะต้องเข้าใจถึงโครงสร้างและสถาปัตยกรรมภายในของมันเสียก่อน ซึ่งองค์ประกอบหลักๆของ คอมพิวเตอร์ มีดังต่อไปนี้

3.2.1.1) Central Processing Unit (CPU)

ทำหน้าที่เป็นหน่วยประมวลผลหลักของ คอมพิวเตอร์ เป็นตัวกลางคอยควบคุมส่วนประกอบอื่นๆ ใน คอมพิวเตอร์ โดยจะทำการประมวลผลข้อมูลที่ได้มาเป็นตัวเลข และส่งออกเป็นตัวเลขเช่นเดียวกัน

3.2.1.2) Memory

คือหน่วยความจำนั่นเอง คอยทำหน้าที่เก็บข้อมูลให้ทั้ง ซีพียู และอุปกรณ์I/O ต่างๆ ทั้งเมื่อเกิดการ เข้า และ ออก ของข้อมูล

3.2.1.3) ส่วน อินพุท และ เอาท์พุท

จะประกอบไปด้วยส่วนของ พอร์ต และ บัส ต่างๆที่คอยทำหน้าที่เป็นตัวรับ - ส่งข้อมูลระหว่าง ซีพียู กับส่วนต่างๆของ คอมพิวเตอร์ ทั้งภายในและภายนอก

3.2.2 ISA บัส

ย่อมาจากคำว่า (Industry Standard Architecture) เป็นระบบ บัส ที่จะปรากฏอยู่ใน คอมพิวเตอร์ ที่มีสถาปัตยกรรมแบบ AT ขึ้นไป สำหรับระบบ บัส มีคำจำกัดความที่ควรรู้ดังต่อไปนี้

Master และ slave

ในขณะที่มีการส่งสัญญาณ read และ write นั้นแต่ละอุปกรณ์ก็ต้องการ แอดเดรส ที่มีค่าของตนเอง ซึ่งอุปกรณ์ที่เป็นตัวกำหนดสถานะเริ่มต้นและควบคุมการติดต่อจะเรียกว่า มาสเตอร์ขณะที่อุปกรณ์ที่ทำการตอบสนองจะเรียกว่า สเลฟ ยกตัวอย่างเช่น ซีพียู จะถือว่าเป็น มาสเตอร์ และเมมโมรี จะถือว่าเป็น สเลฟ เป็นต้น

Bus Arbitration

บัส จะไม่สามารถทำงานกับ มาสเตอร์ 2 ตัวได้ในเวลาเดียวกัน ดังนั้นในขณะที่ มาสเตอร์ ขอ เข้า มายัง บัส จะต้องขออนุญาติจาก เซ็นทรัล บัส อาร์บิเตรเตอร์ และคอยสัญญาณตอบสนองก่อนที่จะทำงานต่อไปได้ โดยมี เซ็นทรัล บัส อาร์บิเตรเตอร์ จะคอยให้ ไพออริตี้ แก่อุปกรณ์ มาสเตอร์ และสเลฟต่างๆใน คอมพิวเตอร์ ตามหลักการที่ว่า First - Come - First - Served

Bus protocal

การติดต่อกับระบบ บัส จำเป็นต้องมีรูปแบบของ ไทม์มิ่ง และ ซิกแนล โดยเฉพาะซึ่งในระบบ บัส แบบ ISA จะมี โปรโตคอล อยู่ 2 แบบคือซิงโครนัส โปรโตคอล และอะซิงโครนัส โปรโตคอล โดย ซิงโครนัส โปรโตคอล จะมีการใช้เมื่อเกิดการ Mismatch กันของ บัส ไทม์มิ่ง ของ มาสเตอร์ และ สเลฟ ในขณะที่ อะซิงโครนัส โปรโตคอล จะใช้เมื่อ บัส ไทม์มิ่ง ของ มาสเตอร์ และ สเลฟ แมทซ์ กันซึ่งโดยปกติแล้ว ไทม์มิ่ง ของ สเลฟ จะช้ากว่า มาสเตอร์

Bus Bandwidth

เป็นอัตราในการส่งถ่ายข้อมูลจาก มาสเตอร์ ไปยัง สเลฟ โดยจะขึ้นกับทั้ง ความเร็ว บัส,ความกว้าง และ โปรโตคอล

สำหรับ การ์ด อินเตอร์เฟส ที่ได้ออกแบบไว้นี้จะใช้ บัส แบบ ISA เพราะสามารถพบได้ใน คอมพิวเตอร์ รุ่น80386 ขึ้นมาซึ่งเป็น บัส แบบ 16 บิท โดยมี Pin สัญญาณต่างๆดังต่อไปนี้

A0 - A19

เป็นการกล่าวอ้างถึง SA0 - SA9 ซึ่ง SA ก็คือ System Address นั่นเอง โดยมีขนาด 20 บิท แอดเดรส สำหรับ เข้า ถึง เมมโมรี หรือ อินพุท/เอาท์พุท

AEN

ย่อมาจาก Address Enable ใช้แสดงว่าใครเป็นผู้ควบคุม บัส อยู่ เช่น ไมโครโปรเซสเซอร์ หรือ DMA เมื่อ AEN = 1 DMAจะควบคุม Address , Data, Ior, Iow, Memr และMemw Bus และเมื่อ AEN = 0 ไมโครโปรเซสเซอร์ จะควบคุม บัส แทน

ALE

ย่อมาจาก (Address Latch Enable) ซึ่งมักกล่าวอ้างถึง BALE (Buffer ALE) เมื่อ ALE เปลี่ยนจาก Low เป็น High มันจะแสดงค่าที่ถูกต้องของ แอดเดรส บน PIN A0 - A9 และเมื่อ ALE เปลี่ยนจาก High เป็น Low จะใช้เมื่อ Latch แอดเดรส จาก ไมโครโปรเซสเซอร์

CLK

Clock เป็นสัญญาณนาฬิกาของทั้งระบบ คอมพิวเตอร์ซึ่งทั้งเมมโมรี,I/O Readและ

I/OWrite จะทำการ ซิงโครไนส์ด้วย ยิ่งเป็น คอมพิวเตอร์ รุ่นใหม่จะยิ่งสามารถทำงานที่สัญญาณนาฬิกาที่สูง ทำให้มีความเร็วในการทำงานเร็วขึ้นด้วย

D0 - D7

เป็น 8 บิท ดาต้า บัส ใช้ในการรับส่งข้อมูลสำหรับ ไมโครโปรเซสเซอร์ เมมโมรี และI/O

DRQ1 , DRQ2 ,DRQ3 ,DACK1 ,DACK2 ,DACK3

เป็นชื่อย่อของ DMA Request และ DMA Acknowledge ใช้ในการทำ DMA Service

$\overline{\text{IOCHCHK}}$

ย่อมาจาก I/O Channel Check จะทำงานเมื่อ Active -Low โดยใช้เพื่อแสดงว่า MotherBoard เกิดการ Error ขึ้นเมื่อมีการใส่ Add - In การ์ด ใน Expansion Slot

$\overline{\text{IOCHRDY}}$

ย่อมาจาก I/O Channel Ready จะทำงานเมื่อ Active - Low ใช้เมื่อมีการเสียบอุปกรณ์ I/O หรือ เมมโมรี ใน Expansion Slot เมื่อใส่ Wait States ใน เมมโมรี หรือ I/O

$\overline{\text{IOR}}$, $\overline{\text{IOW}}$

คือ I/O Read และ I/O Write ใช้ควบคุมสัญญาณในการอ่านข้อมูลจาก บัส หรือส่งข้อมูลออกจาก บัส โดยทำงานที่ Active-Low

IRQ3 - IRQ7 และ IRQ9

เป็น Interrupt Request (IRQ)Line ใช้โดยอุปกรณ์ อินพุท / เอาท์พุท เพื่อส่งสัญญาณไปยัง ซีพียู เพื่อขอให้ ซีพียู ว่างและให้มาทำงานของ อินพุท / เอาท์พุท

OSC

ใช้ส่งสัญญาณ เอาท์พุท ออกมากับความถี่ 14.31818 MHZ มี Duty Cycle 50% จะใช้ในบาง Video Boards เท่านั้น

$\overline{\text{REFRESH}}$

จะทำงานที่ Active - Low เพื่อแสดงว่าต้องการให้มีการ Refresh ในCycle ของการทำงาน

RESET DRV

ย่อมาจาก Reset Driveใช้ในการ Reset MotherBoardหรืออุปกรณ์อื่นๆที่ต่อกับ บัส

$\overline{\text{SMEMR}}$, $\overline{\text{SMEMW}}$

จะทำงานที่ Active - Lowทั้งคู่โดย SMEMR จะสั่งให้ เมมโมรี ให้ค่าข้อมูลลงบน

บัสข้อมูล ส่วน SMEMW ก็จะให้ เมมโมรี เก็บข้อมูลจาก บัสข้อมูล ไปไว้ใน เมมโมรี

TC

Terminal Count จะ High เมื่อมี DMA Channel ใดๆเข้ามาใช้ Terminal Count

0WS

ย่อมาจาก Zero Wait State จะทำงานเมื่อ Active - Low ใช้เมื่อสั่งไม่ให้ MotherBoard ใส่ WS และให้ Cycle เสร็จสมบูรณ์ภายใน 2 Clock

+5V ,-5V ,+12V ,-12V ,GND ,GND

ใช้ต่อแหล่งจ่ายไฟไปยัง การ์ด ที่ต่อกับ บัส

ภาค Expansion Slot (36 PIN)

A17 - A23

แทน Latchable แอดเดรส แต่ความจริงแล้วมีแค่ A20 - A23ส่วนA17 - A19มีไว้เพื่อ Latch ข้อมูลเพิ่มเติม แต่จริงๆแล้ว ใน PC ทั่วไปจะใช้ แอดเดรส เพียง A0 - A15 นั่นหมายความว่า มี I/O พอร์ต ได้สูงสุดเพียง 65,536 เท่านั้น

D8 - D15

เป็น บัสข้อมูล ที่ขยายเพิ่มจาก PC/XT เพื่อให้เป็น บัส แบบ 16 บิท โดย D0 - D7 จะถูกใช้บ่อยๆ แต่ D8 - D15 ใช้เมื่อต่อกับ I/O แบบ 16 บิท

DRQ0 , DRQ5 - 7 , DACK0 , DACK 5 - 7

เป็น 4 Additional Dma Channel โดย DRQ 0 คือ 8 - บิท DMA Channel ,DRQ 5 - 7 เป็น 16 - บิท Channel

IOCS16

คือ 16 - Bit Chip ซิกแนล จะทำงานที่ Active - Low เพื่อให้เกิดความ Compatibility กับระบบ 8 - บิท PC/XT

IRQ10 , IRQ11 , IRQ12 , IRQ14 , IRQ15

เป็น Additional Interrupt Request ให้กับ PC AT System โดยมี ไพออริตี้ สูงกว่า IRQ3 - 7 แต่มี ไพออริตี้ ต่ำกว่า IRQ9

MASTER

เป็น อินพุท ซิกแนล โดยจะทำงานที่ Active - Low เพื่ออนุญาติให้ Add - In การ์ด ที่เสียบลงไปใน Expansion Slot เพื่อให้เป็น มาสเตอร์ และขยายการควบคุมทั้งระบบ บัส

$\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$

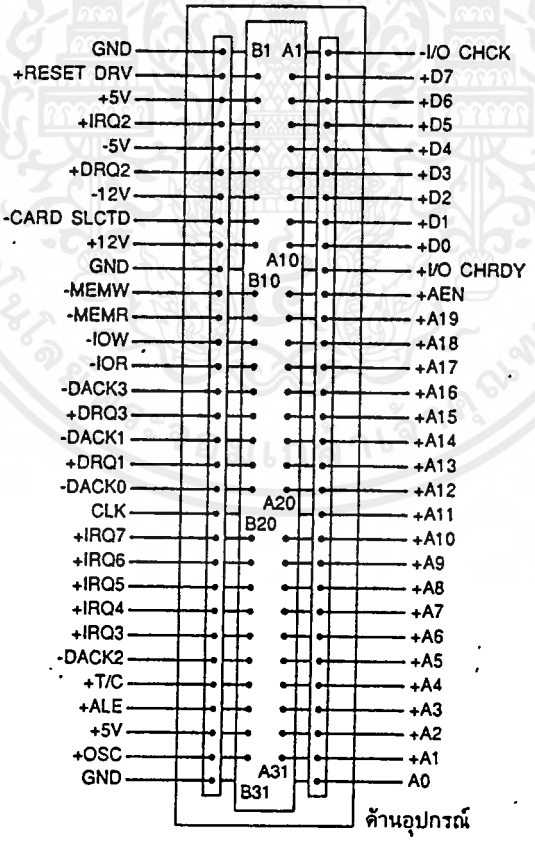
จะมีโครงสร้างเหมือนกับที่กล่าวมาแล้ว

BHE

หรือเรียกอีกอย่างหนึ่งว่า SBHE เพื่อแสดงการขนถ่ายข้อมูลบน D8 -D15 ซึ่งเป็น 8บิทบนของ บัส แบบ 16 บิท โดย D0 - D15 จะใช้เมื่ออุปกรณ์ 16 บิท ต้องการใช้บน Add - In การ์ด

+5V,GND

ใช้เพื่อป้อนไฟเลี้ยงให้แก่ส่วน 36 PINS ของ ISA บัส



(+ หมายถึง ทำงานที่ลอจิก "1" , - หมายถึง ทำงานที่ลอจิก "0")

รูปที่ 3.4 แสดงการจัดวางตำแหน่งของขาสัญญาณของบัสของคอมพิวเตอร์

3.3 ทฤษฎีและโครงสร้างของ ไมโครคอนโทรลเลอร์ ตระกูล PIC 16C8x

PIC16C84 เป็นไมโครคอนโทรลเลอร์แบบ ซีมอส ที่มีหน่วยความจำแบบ EEPROM อยู่ภายใน,มีความเชื่อถือได้สูง,เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ผลิตโดยบริษัทไมโครชิพเทคโนโลยี ใช้สถาปัตยกรรมภายในแบบ RISC คำสั่งใช้งานเพียง33 คำสั่ง ทุกคำสั่งใช้เวลาในการเอ็กซีคิวส์ 1ไซเคิล ยกเว้นในกรณีโปรแกรมบรานซ์ จะต้องใช้ 2ไซเคิล คำสั่งขนาดกว้าง 14บิต จะช่วยทำให้ผลของรหัสคำสั่งมีอัตราส่วนเป็น 2.5 :1เมื่อเทียบกับไมโครคอนโทรลเลอร์ขนาด 8บิตอื่นๆ

3.3.1 คุณสมบัติทั่วๆไป

3.3.1.1 CPUใช้รูปแบบการทำงานแบบ RISC

1. มีคำสั่งที่ต้องเรียนรู้เพียง 35 คำสั่ง
2. ทุกคำสั่งใช้เวลาปฏิบัติการ 1 ไซเคิล ยกเว้นคำสั่งแบบ บรานซ์ 2ไซเคิล
3. ความเร็วในการทำงาน : จากช่วง DC – สัญญาณนาฬิกาอินพุทเท่ากับ 10 เมกะเฮิรดซ์ หรือ จากช่วง DC –ไซเคิลเท่ากับ 400 นาโนวินาที
4. คำสั่งมีขนาดกว้าง 14 บิต
5. บัสข้อมูลกว้าง 8 บิต
6. หน่วยความจำเป็น อีพรอมขนาด 14 บิต
7. รีจิสเตอร์ใช้งานทั่วไปขนาด 8 บิตมี 38 ตัวเป็น SRAM
8. รีจิสเตอร์ฮาร์ดแวร์ที่ทำหน้าที่พิเศษมี 15ตัว
9. สแต็กฮาร์ดแวร์มีความลึก 8 ระดับ
10. มีไหมดการอ้างอิงแอดเดรสของข้อมูล และคำสั่งแบบโดยตรง,โดยอ้อม และแบบสัมพันธ์

3.3.1.2 คุณสมบัติของอุปกรณ์ต่อพ่วง

1. การควบคุมทิศทางทำโดยการใช้ขา 13 ขา
2. มีตัวนับเวลา / สัญญาณเวลาขนาด 8 บิต(RTCC)กับตัวตั้งค่าที่โปรแกรมได้ขนาด 8 บิต
3. มีตัวรีเซตกำลังไฟ (Power on reset)
4. มีตัว OST (Oscillator Start Timer)
5. มีตัว WDT (Watchdog Timer) กับออสซิลเลเตอร์แบบ RC เพื่อใช้ในการปฏิบัติงานที่เชื่อถือได้
6. มีฟิวส์อีพรอมพิเศษเพื่อป้องกันการลอกเลียนซอร์สโค้ด

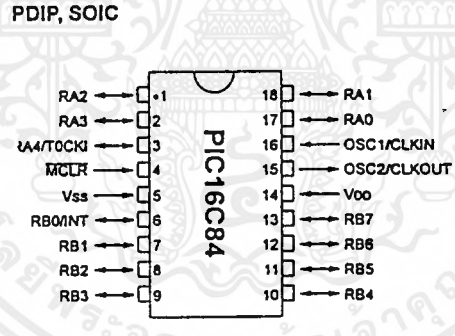
7. มีโหมด SLEEP ลดการสูญเสียพลังงานเมื่อไม่ได้ใช้งาน
8. มีออสซิเลเตอร์ให้เลือกใช้เพื่อกำหนดไปยังอีพรอมแบบต่างๆ ดังนี้

   -ออสซิเลเตอร์แบบ RC ที่มีราคาถูก

   -ออสซิเลเตอร์แบบคริสตอล / รีโซเนเตอร์มาตรฐาน XT

   -ออสซิเลเตอร์แบบคริสตอล / รีโซเนเตอร์ความเร็วสูง HS

   -ออสซิเลเตอร์แบบคริสตอล ความถี่ต่ำ,กินไฟน้อย LP



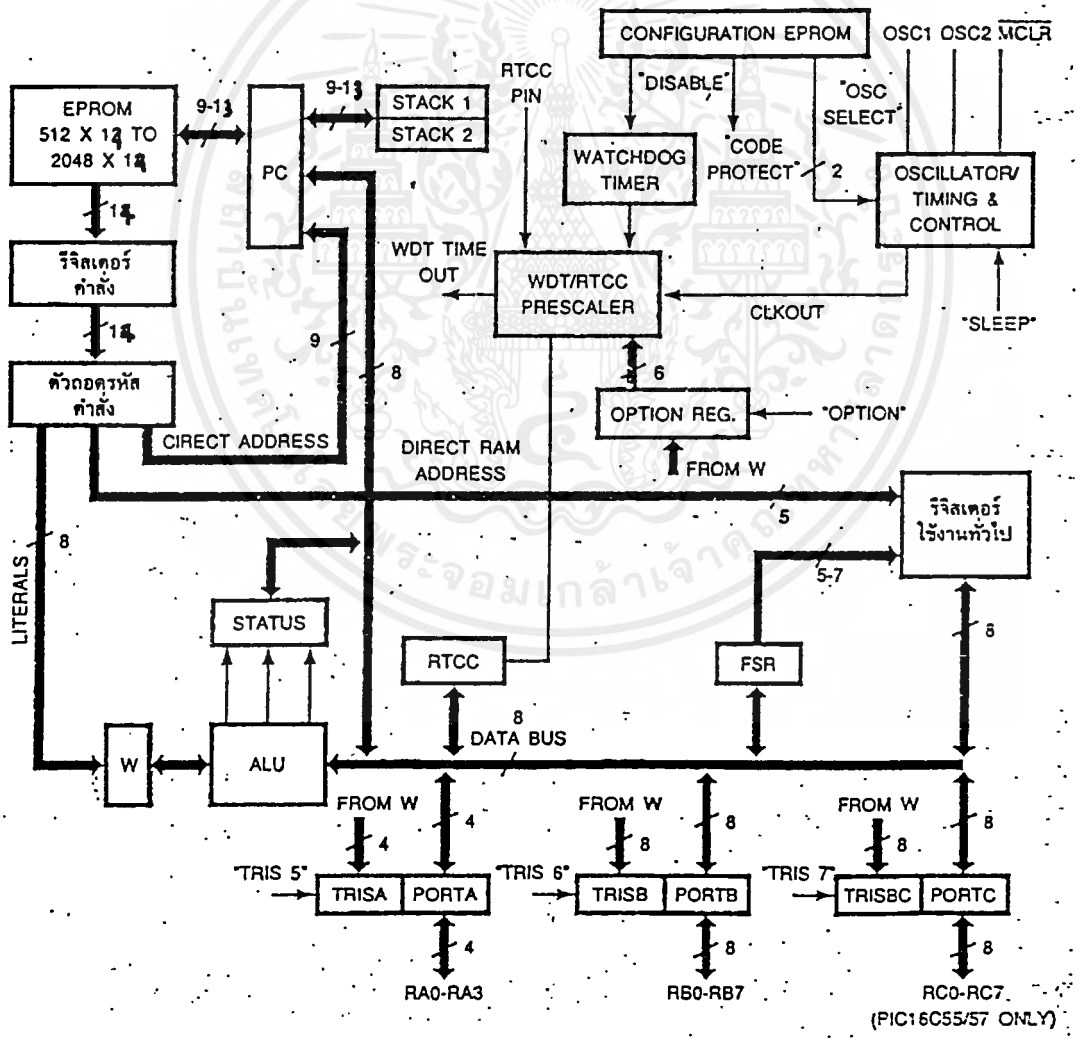
รูปที่ 3.5 แสดงการจัดวางตำแหน่งขาอุปกรณ์ของ PIC 16C84

3.3.1.3 เทคโนโลยี CMOS

1. กินไฟต่ำ,ใช้อีพรอมแบบ CMOS ความเร็วสูง
2. ออกแบบมาจากข้อมูลทางสถิติ
3. ค่าศักดาไฟในช่วงการทำงาน 2.5 โวลต์ ถึง 6.25 โวลต์
4. กินไฟต่ำน้อยกว่า 2 มิลลิแอมป์ที่แรงดัน 5 โวลต์ความถี่ 4เมกะเฮิรตซ์
5. น้อยกว่า 15 ไมโครแอมป์ที่แรงดัน 3 โวลต์ความถี่ 32 กิโลเฮิรตซ์

3.3.2 สถาปัตยกรรมภายใน

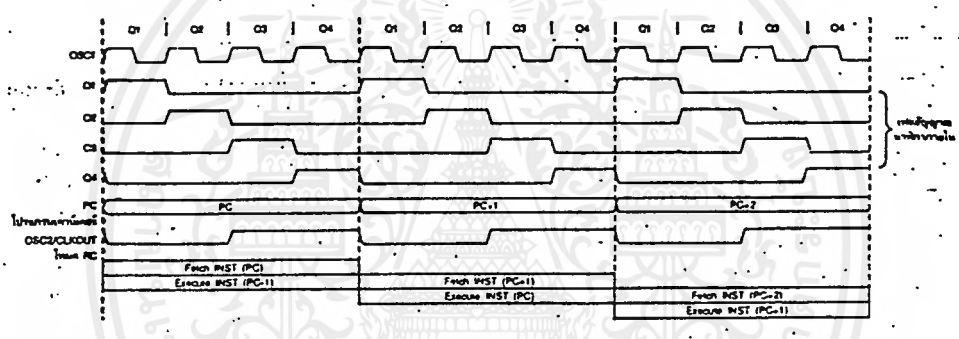
สถาปัตยกรรมภายในของ PIC16C84ใช้แนวความคิดแบบรีจิสเตอร์ไฟล์ ซึ่งจะมีการแยกบัสและหน่วยความจำสำหรับเก็บข้อมูลและเก็บคำสั่ง บัสข้อมูลและหน่วยความจำจะมีขนาดกว้าง 8 บิต ในขณะที่บัสโปรแกรมจะมีขนาดกว้าง14 บิต แนวความคิดนี้ทำให้สามารถใช้ชุดคำสั่งเพียงชุดเดียว ในการทำงานเป็นบิต ,ไบต์ และรีจิสเตอร์ด้วยความเร็วสูง โดยมีการเฟตช์และเอ็กซีคิวส์คำสั่งแบบซ้อนทับกัน ซึ่งหมายความว่า ในขณะที่คำสั่งหนึ่งกำลังถูกเอ็กซีคิวส์คำสั่งต่อไปจะถูกอ่านจากหน่วยความจำเก็บโปรแกรมมาเก็บไว้ บล็อกไดอะแกรมของไมโครคอนโทรลเลอร์ตระกูล PIC 16C8X แสดงดังรูป



รูปที่ 3.6 แสดงบล็อคไดอะแกรมแสดงการทำงานภายในของไมโครคอนโทรลเลอร์ตระกูล PIC 16C8X

3.3.3ไทมิ่งไดอะแกรมแสดงลำดับการประมวลผลคำสั่ง

ไทมิ่งไดอะแกรมแสดงการทำคำสั่งของ PIC16C8X ซึ่งสัญญาณนาฬิกาอินพุท (ป้อนเข้าที่ขา OSC1) จะถูกแบ่งออกเป็น4 ช่วงเพื่อสร้างสัญญาณนาฬิกาภายใน ที่ไม่ทับซ้อนกัน 4ค่า เรียกว่า Q1 , Q2 ,Q3 , Q4 ภายในชิปค่าโปรแกรมเคาน์เตอร์จะเพิ่มค่าที่เวลาQ1 ,คำสั่งจะถูกเฟตช์จากหน่วยความจำเก็บโปรแกรมและนำเข้าไปเก็บในรีจีสเตอร์คำสั่งที่เวลา Q4 จะถูกถอดระหัสและ เอ็กซิคิวส์ในช่วงเวลา จาก Q1 จนถึง Q4 ดังรูป



รูปที่ 3.7 แสดงไทม์มิ่งไดอะแกรมแสดงการทำคำสั่งของ PIC 16C8X

3.3.4 รีจิสเตอร์เก็บข้อมูล (Data Register file)

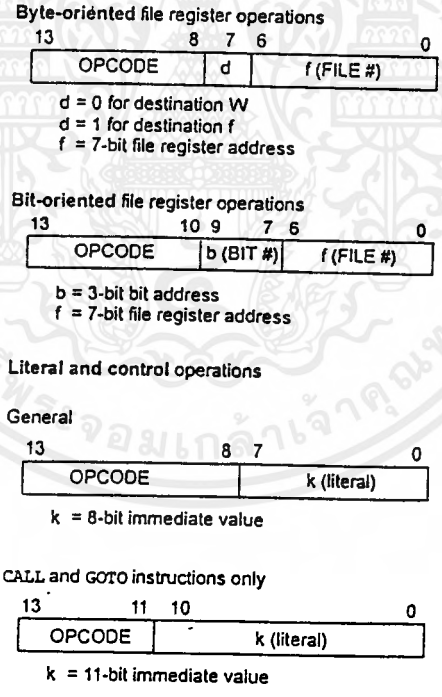
บัสข้อมูลขนาด 8 บิต จะเชื่อมต่อส่วนที่ทำหน้าที่พื้นฐาน 2 ส่วนเข้าด้วยกัน คือ รีจีสเตอร์ไฟล์ที่ประกอบด้วยรีจีสเตอร์ 8 บิตจำนวน 36 ตัวรวมถึงพอร์ตอินพุต /เอาต์พุตต่างๆ กับ ALU (Arithmetic Logic Unit) ขนาด 8 บิต แรมขนาด 32 ไบต์จะถูกอ้างอิงแอดเดรสได้โดยตรงในขณะที่พื้นที่หน่วยความจำใน “แบงค์ (bank)” ขนาด 16 ไบต์ ต่อ 1แบงค์ จะถูกใช้เพื่อขยายหน่วยความจำให้มีขนาดใหญ่ขึ้น ข้อมูลสามารถถูกอ้างอิงตำแหน่งได้โดยตรง หรือโดยอ้อม โดยใช้รีจีสเตอร์ FSR (File Select Register)ทันทีที่มีการอ้างอิงตำแหน่งข้อมูลโดยใช้คำสั่งตาม “ตัวอักษร” ข้อมูลจะถูกโหลดจากหน่วยความจำโปรแกรมเข้าไปในรีจีสเตอร์ W

3.3.5 ALU (Arithmetic Logig Unit)

ALU ขนาด 8 บิตจะมีรีจีสเตอร์ W ซึ่งใช้เก็บข้อมูลชั่วคราวรวมอยู่ด้วย ALU จะทำหน้าที่คำนวณทางคณิตศาสตร์ และบลูลีน (Boolean)ระหว่างข้อมูลที่อยู่ในรีจีสเตอร์ W กับรีจีสเตอร์ ไฟล์ใดๆได้ด้วย

3.3.6 หน่วยความจำโปรแกรม(program memory)

หน่วยความจำโปรแกรมนี้มีขนาด 1024 เวิร์ด(1เวิร์ด มีขนาด 14 บิต) เป็นอีพรอม, สามารถอ้างอิงแอดเดรสได้โดยตรง โปรแกรมขนาดใหญ่สามารถอ้างอิงแอดเดรสได้โดยการเลือกเพจหนึ่งจาก 4เพจ ลำดับคำส่งของคำสั่งขนาดเล็ก (micro instruction) จะถูกควบคุมโดยผ่ายโปรแกรมเคาน์เตอร์ที่มีการเพิ่มอย่างอัตโนมัติทุกครั้งที่มีการเอ็กซีคิวส์โปรแกรมในการปฏิบัติงานเพื่อควบคุมโปรแกรม ,การอางอิงแอดเดรสโดยตรง ,โดยอ้อม ,โหมดการอ้างอิงแบบสัมพัทธ์ สามารถกระทำกับบิตทดสอบ และคำสั่ง Skip ,คำสั่ง Call , คำสั่ง Jump หรือการคำนวณแอดเดรสเพื่อโหลดลงในโปรแกรมเคาน์เตอร์ นอกจากนี้ในชิปยังมีแสต็ก (stack) ถึง 8 ระดับ เพื่อให้ง่ายต่อการใช้โปรแกรมย่อยแบบโครงข่าย (nesting)



รูปที่ 3.8 แสดงโปรแกรมเมมโมรี

3.3.7 สภาวะรีเซต

สภาวะรีเซตอาจเกิดขึ้นจากกสนที่เริ่มป้อนไฟเลี้ยงให้แก่ชิป , ป้อนค่า ลอจิก 0 เข้าที่ขา MCLR หรือเกิดจากตัว WDT เกิดไทม์เอาต์ ชิปจะอยู่ในสภาวะรีเซตตราบเท่าที่ขา MCLR ยังคงมีค่าลอจิก 0 หรือตัว OST (ocillator starp up timer) แอกทีฟ

ตัว OST จะแอกทีฟทันทีที่ขา MCLR มีค่าลอจิกเป็น 1 สรุปได้ว่าในกรณีการรีเซตเนื่องจากการป้อนไฟเลี้ยงทำๆด้โดยการผูกขา MCLR เข้ากับขา VDD และตัว OST จะเริ่มทำงานหลังจากป้อนไฟเลี้ยง ในกรณีที่ WDT เกิดไทม์เอาต์ตัว OST เริ่มทำงานเมื่อตัว WDT เกิดไทม์เอาต์ (ในขณะที่ขา MCLR มีค่าลอจิก 1 ในกรณีที่รีเซตขา MCLR ตัว OST จะเริ่มทำงานอีกครั้ง เมื่อขา MCLR มีค่าลอจิก 1) โดยปกติแล้วตัว OST จะมีช่วงคาบเวลาเท่ากับ 18 มิลลิวินาที ซึ่งจะกล่าวถึงรายละเอียดของ OST อีกครั้งในตอนหลังในสภาวะรีเซต PIC16C8X จะมีสถานะดังนี้

1. ออสซิลลเลเตอร์จะทำงานหรือเริ่มต้นทำงาน (ป้อนไฟเลี้ยงหรือตื่นจากโหมด SLEEP)
2. ขาของพอร์ตอินพุต / เอาต์พุท ทุกขา จะอยู่ในสถานะความต้านทานสูงโดยจะรีเซตให้รีเซตให้รีจิสเตอร์ "TRIS" มีค่าเป็น 1 ทั้งหมด
3. โปรแกรมเคาน์เตอร์ถูกเซตเป็น 1 ทั้งหมด
4. รีจิสเตอร์ออปชั่นถูกเซตเป็น 1 ทั้งหมด
5. ตัว WDT และปรีสเกลเลอร์จะถูกเคลียร์
6. 3 บิตบน(เลือก เพจ) ในรีจิสเตอร์สถานะถูกเคลียเป็น 0
7. เฉพาะชิปที่ใช้ออสซิลเลเตอร์แบบ RC สัญญาณ CLKOUT ที่ขา OSC จะมีค่าเป็นลอจิก 0

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb			LSb		
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0000	0011	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff	None	1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff	None	1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff	None	
NOP	-	No Operation	1	00	0000	0xx0	0000	None	
RLF	f, d	Rotate left f through carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate right f through carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff	None	1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff	None	1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff	None	1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff	None	3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff	None	3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk	None	
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk	None	
RETFIE	-	Return from interrupt	2	00	0000	0000	1001	None	
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk	None	
RETURN	-	Return from subroutine	2	00	0000	0000	1000	None	
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (i.e., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the TMR0.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

รูปที่ 3.9 แสดงชุดคำสั่งของไมโครคอนโทรลเลอร์ PIC 16C84

บทที่ 4

การออกแบบวงจรควบคุมสัญญาณไฟจราจร

4.1 อินเตอร์เฟสการ์ด

อินเตอร์เฟสการ์ด (Interface Card) คือการ์ดที่ใช้เชื่อมต่อระหว่าง คอมพิวเตอร์ กับ อุปกรณ์ภายนอก โดย อินเตอร์เฟสการ์ด ที่ใช้ในการควบคุมสัญญาณไฟจราจรนั้นมีจุดประสงค์เพียงเพื่อเชื่อมต่อกับอุปกรณ์ภายนอกโดยเป็นตัวกลางในการส่งผ่านข้อมูลที่ได้เขียนขึ้นจากโปรแกรมภาษา C ส่งไปยังวงจรควบคุมสัญญาณไฟจราจรที่อยู่ภายนอก

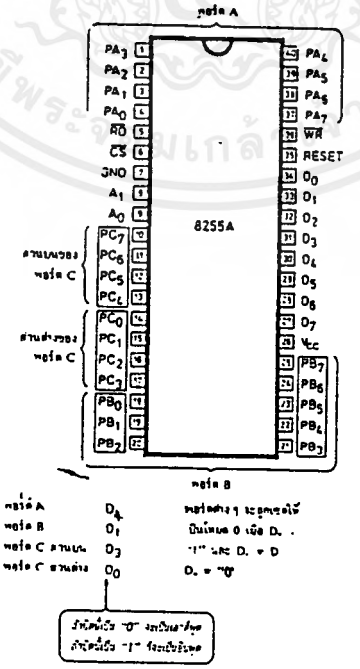
ซึ่งตัว การ์ด ที่ใช้จะมีส่วนประกอบดังต่อไปนี้

ไอซี 8255 1 ตัว

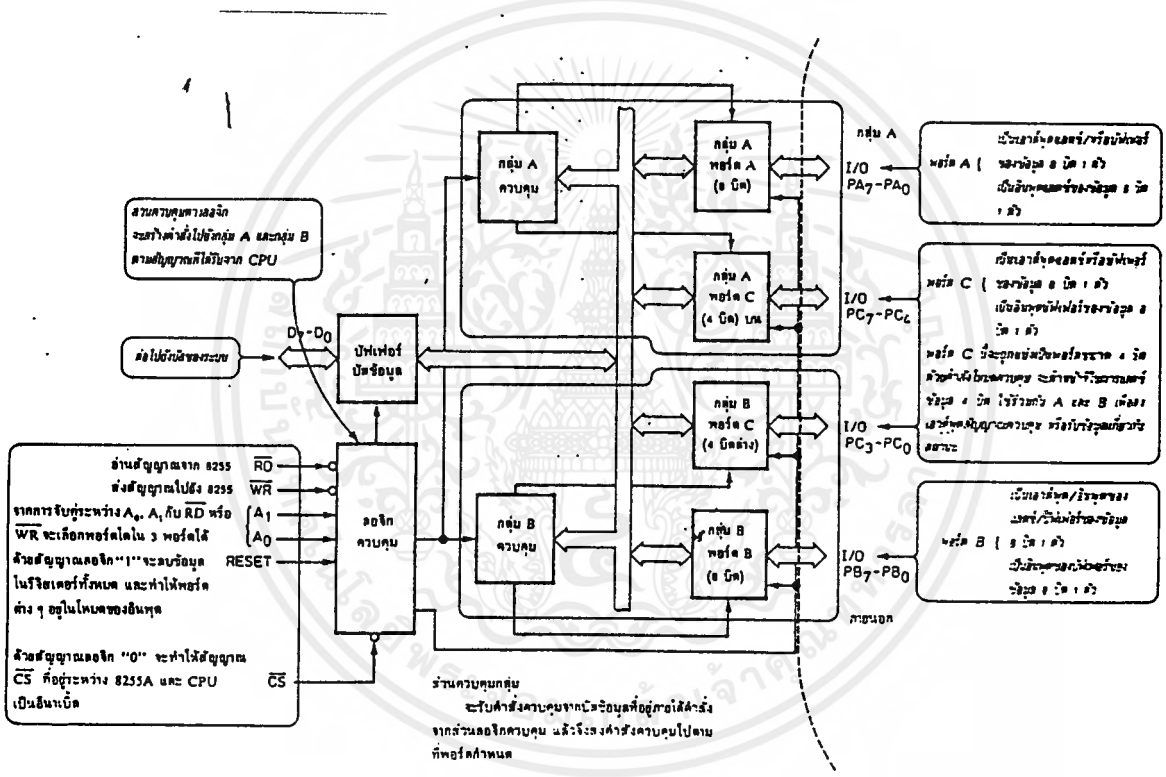
ไอซี 74LS138 2 ตัว

ไอซี 8255

จัดเป็นอุปกรณ์แบบ เพอริเฟอรอล อินเตอร์เฟส ที่สามารถโปรแกรมได้ชนิดหนึ่ง โดยจัดเป็น พอร์ต แบบขนาน โดยตัว ไอซี 8255 นี้จะมี พอร์ต ที่ใช้งานได้อยู่ทั้งหมด 3 พอร์ต (24 บิต) และสามารถ ซิงโครไนส์ กับอุปกรณ์ภายนอกได้ โดย ไอซี 8255 จะมีโครงสร้างดังรูป

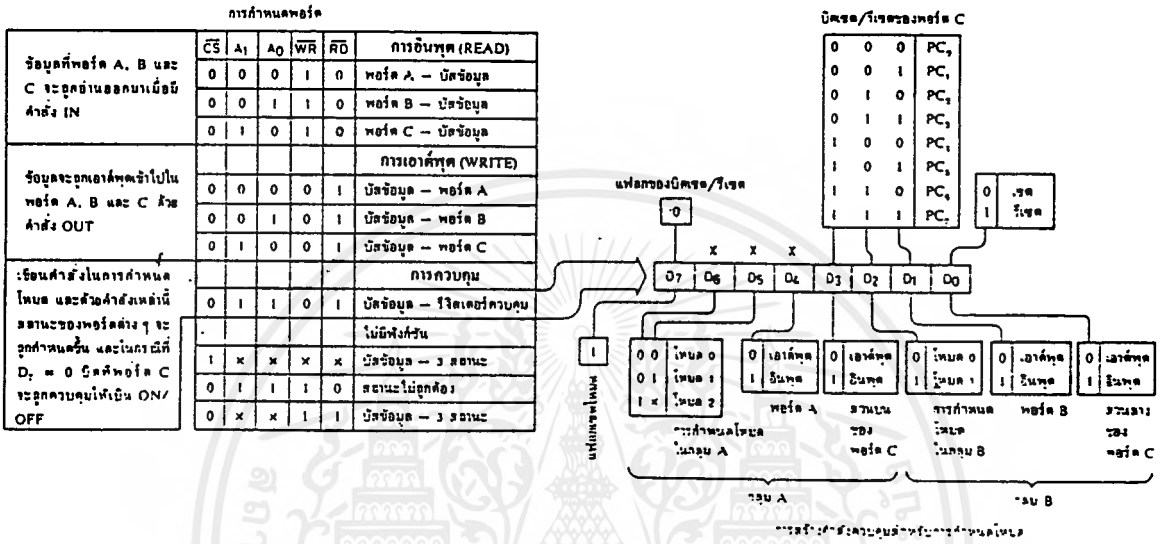


รูปที่ 4.1 แสดงโครงสร้างของการวางตำแหน่งขาของ 8255



รูปที่ 4.2 แสดงการทำงานส่วนต่างๆของ 8255 A

โดยจะมี พอร์ต 3 พอร์ต คือ A , B , C ตามลำดับและสามารถจัดแบ่งกลุ่มเป็น 2 กลุ่ม โดยที่ พอร์ต A และ พอร์ต B จะทำงานแตกต่างกันและไม่ขึ้นต่อกัน ส่วน พอร์ต C นั้นจะสามารถแบ่งสัญญาณออกเป็น 2 ส่วนๆละ 4 บิต ทำหน้าที่เป็นสัญญาณควบคุมให้กับ พอร์ต A และ B จากการที่เราใช้ ไอซี 8255 นี้ทำให้สามารถต่อ เอาท์พุท เพื่อไปควบคุมอุปกรณ์ภายนอกได้ทั้งหมด 24 ตัว โดยเราสามารถที่จะกำหนดให้ พอร์ต ใดเป็น อินพุท หรือ เอาท์พุท ก็ได้โดยทำการป้อน คอนโทรลเวิร์ด เข้าไปที่ ไอซี 8255 จากส่วนของโปรแกรมภาษา C สำหรับการต่อ 8255 แต่ละขากับอะไรบ้างนั้นจะขอกล่าวถึงในภายหลัง โดยการกำหนดคำสั่งควบคุม (Control Word) แสดงไว้ดังรูปที่ 4.3



รูปที่ 4.3 แสดงการกำหนดคำสั่งควบคุม (Control Word) ใน 8255 A

ไอซี 74LS138

ไอซี ประเภทนี้เป็น ไอซี ดีโคดเดอร์ ชนิด เข้า 3 ออก 8 นั่นคือ เราสามารถป้อนอินพุทเพียง 3 บิต ก็สามารถควบคุมอุปกรณ์ที่ต่ออยู่กับ เอาท์พุท ของมันได้ 8 ตัว โดยทำการเลือกใช้งานได้ครั้งละ 1 อุปกรณ์

สำหรับหน้าที่ของ ไอซี 74LS138นี้นั้นมีไว้เพื่อช่วยในการควบคุมการเลือกแอดเดรส ของ บัส ของ คอมพิวเตอร์ มาใช้งานโดยใช้ ดิพสวิทช์ เป็นตัวเลือก แอดเดรส ซึ่งอธิบายได้ดังต่อไปนี้

สำหรับ ไอซี 74LS138 ตัวที่ 1 นั้นจะคอยควบคุมการเลือก แอดเดรส ของ A2 – A4

สำหรับ ไอซี 74LS138 ตัวที่ 2 นั้นจะคอยควบคุมการเลือก แอดเดรส ของ A7 – A5

สำหรับ แอดเดรส ที่ A10 จะ คง ค่าไว้ที่ 0 โดยจะต่อกับ กราวนด์ของคอมพิวเตอร์

สำหรับ แอดเดรส ที่ A 9 จะ คง ค่าไว้ที่ 1 โดยจะต่อกับ +5Vcc ของคอมพิวเตอร์

สำหรับ แอดเดรส ที่ A 8 จะ คง ค่าไว้ที่ 0 โดยจะต่อกับ กราวนด์ ของคอมพิวเตอร์

ส่วน แอดเดรส ที่ A0 และ A1 จะใช้เป็นตัวเลือก พอร์ต ของ ไอซี 8255 ดังนี้

A0 = 0 ,A1 = 0 เป็นการเลือกให้ เอาท์พุท จาก บัสข้อมูล คอมพิวเตอร์ ออกไปยัง พอร์ต A ของ 8255

A0 = 0 ,A1 = 1 เป็นการเลือกให้ เอาท์พุท จาก บัสข้อมูล คอมพิวเตอร์ ออกไปยังพอร์ต B ของ 8255

A0 = 1 ,A1= 0 เป็นการเลือกให้ เอาท์พุท จาก บัสข้อมูล คอมพิวเตอร์ ออกไปยังพอร์ต C ของ 8255

A0 = 1 ,A1 = 1 เป็นการเลือกให้ เอาท์พุท จาก บัสข้อมูล คอมพิวเตอร์ ออกไปยัง

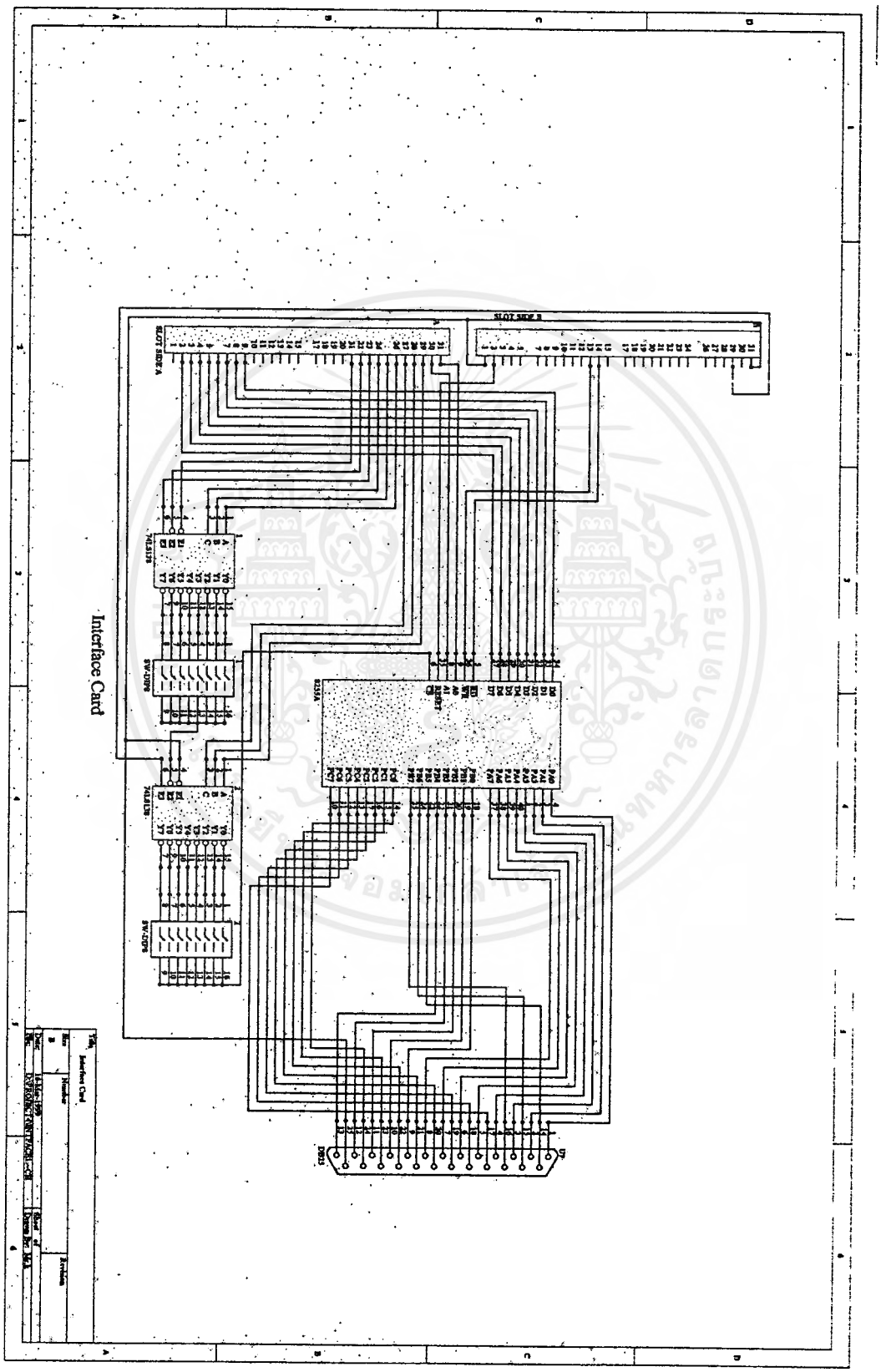
คอนโทรล พอร์ต ของ 8255 โดย 8255 จะรู้ว่า ถ้าสั่ง Data มาที่ แอดเดรส นี้จะถือว่าเป็นการ เซต ค่า คอนโทรล เวิร์ด ให้แก่ตัว 8255 เองโดยอัตโนมัติ โดยจะต้องทำการ เซต คอนโทรล เวิร์ด ให้ 8255 จากภายในโปรแกรมก่อนที่จะสั่งให้โปรแกรมทำงานส่งค่าผ่าน พอร์ต ของ 8255 ใดๆเสมอ

สำหรับตารางการ เซต ค่า ดิพสวิทซ์ มีดังต่อไปนี้

เมื่อ ดิพสวิทซ์ ไปที่	A7	A6	A5	A4	A3	A2
Y0	0	0	0	0	0	0
Y1	0	0	1	0	0	1
Y2	0	1	0	0	1	0
Y3	0	1	1	0	1	1
Y4	1	0	0	1	0	0
Y5	1	0	1	1	0	1
Y6	1	1	0	1	1	0
Y7	1	1	1	1	1	1

ตารางที่ 4.1 แสดงการกำหนดค่าแอดเดรสของอินเตอร์เฟสการ์ด

โดยเมื่อทำเช่นนี้แล้ว เราก็จะสามารถเลือก แอดเดรส ได้ในช่วงตั้งแต่ 200H – 2FFH เลยทีเดียวทั้งนี้เพื่อเป็นการ ล็อค ค่า แอดเดรส ไว้ช่วงหนึ่งซึ่งเป็นช่วงที่เราสามารถใช้งานได้โดยไม่เกิดการ คอนฟลิกซ์ กันของ แอดเดรส ของการ์ด หรือ อุปกรณ์อื่นๆ และยังป้องกันการเกิดสัญญาณรบกวนที่ไม่พึงประสงค์อันจะทำให้ แอดเดรส เปลี่ยนค่าไปจากเดิม จากการกำหนด แอดเดรส โดยตรงจากโปรแกรมไปยัง พอร์ต ได้อีกด้วย

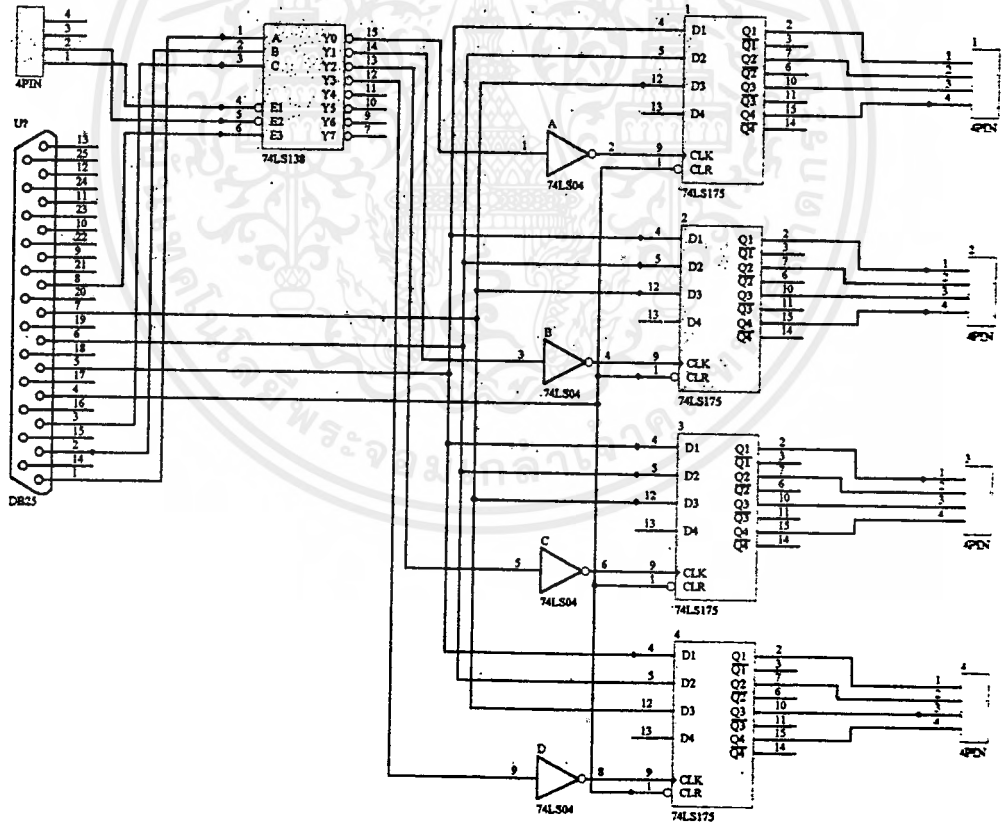


รูปที่ 4.4 แสดงวงจรของอินเตอร์เฟสการ์ด

4.2 I/O Expansion Unit

4.2.1 การทำงานการในการเลือกแยกของสัญญาณไฟ

จากที่กล่าวมาข้างต้น กล่าวถึงเพียงการบังคับไฟจราจร 1 สี่แยก ต่อจากนี้จะกล่าวถึงการควบคุมไฟจราจร 4 แยก จำนวน n แยก ซึ่งโครงงานชิ้นนี้สามารถขยายการควบคุมได้ถึง 2^n แยก เพื่อเป็นการง่ายต่อการเข้าใจ ขอจำลอง 4 แยกไฟจราจรทั้งหมด $2^3 = 8$ แยก



รูปที่ 4.5 แสดงการขยายพอร์ตอินพุท/เอาท์พุท โดยใช้ดีโคดเดอร์ 3:8

จากรูป ถ้าต้องการเลือกแยก 1 ต้องให้ P_{A2} P_{A1} P_{A0} = 0 0 0 ตามลำดับ เพื่อเลือกให้เอาท์พุท ของ 138 Y_0 = Low นอกนั้น เท่ากับ High แล้วทำการ ดิสเอเบิล 47LS138 โดย set P_{A7} = Low ฉะนั้นจะได้ คล็อกพัลส์ที่ D-FF D_1 ฉะนั้น ข้อมูลเอาท์พุท จะเท่ากับ ข้อมูลอินพุท หรือ Q = D โดยที่ขา Q ของ D-FF อื่นๆไม่มีการเปลี่ยนแปลง เนื่องจากไม่มี คล็อกมา เลือก

Q ที่เปลี่ยนไปเป็น Input ของ PIC ทำให้ รูปแบบ ที่ t-1 ไม่เท่ากับ t จึงมีการเปลี่ยนรูปแบบการสว่างเปิด-ปิดของไฟจราจร

P_{A2}	P_{A1}	P_{A0}	สี่แยก
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

ตารางที่ 4.2 แสดงการเลือกแอดเดรสให้ตรงกับตำแหน่งสี่แยก

4.3 วงจรสัญญาณไฟจราจรส่วนย่อย โดย ไมโครคอนโทรลเลอร์ ตระกูล PIC

4.3.1เหตุผลที่ใช้ไมโครคอนโทรลเลอร์มาประยุกต์การใช้งาน

1. ง่ายต่อการออกแบบ
2. สามารถแบ่งส่วนของระบบออกเป็นส่วนๆ ได้
3. ลดส่วนที่เป็นโปรแกรมหลักให้ง่ายต่อการเข้าใจ ลดความซับซ้อน
4. เพื่อการแสดงผลซึ่งจะให้เป็นแบบ เรียลไทม์ มากที่สุด
5. ลดภาระกรรมของคอมพิวเตอร์ ช่วยให้คอมพิวเตอร์ไม่จำเป็นต้องเสียเวลาประมวลผลกับส่วนแสดงผล
6. ลดจำนวนของสายสัญญาณ ซึ่งจะนำไปสู่แต่ละ 4 แยก ซึ่งจะสามารถลด ข้อผิดพลาด ที่เกิดจาก สัญญาณรบกวนได้ (สัญญาณรบกวน ชนิด คลอสทอล์ก สายสัญญาณที่มีกระแสไหลจะ เหนี่ยวนำสนามแม่เหล็กเหนี่ยวนำ สายสัญญาณเส้นอื่นให้ ข้อผิดพลาด)
7. เมื่อคอมพิวเตอร์เกิดความผิดพลาด สามารถแยกส่วน ไมโครคอนโทรลเลอร์ ทำงานอิสระได้
8. ไมโครคอนโทรลเลอร์ ทำงานเที่ยงตรงสามารถเชื่อถือได้

4.3.2 เหตุผลที่นำไมโครคอนโทรลเลอร์ตระกูล PIC (Peripheral Interface Controller)

1. เร็วด้วยเทคโนโลยี RISC คำสั่งขนาดกว้าง 12 บิต
2. เหมาะกับงานควบคุมขนาดเล็ก
3. คำสั่งที่ใช้ง่ายเพียง 35 คำ
4. ความยืดหยุ่นสูง
5. สามารถต่อ ออสซิลเลเตอร์ ได้หลายแบบ XT, HS, LP,RC ซึ่งแบบ RC มีราคาถูกมาก
6. มี โหมดสลีพ กินไฟน้อย
7. ป้องกันการ ลอกแบบได้
8. มี ออสซิลเลเตอร์สตาร์ตอัพไทม์
9. กินไฟต่ำด้วยเทคโนโลยี CMOS ,EPROM ความเร็วสูง
10. ราคาถูก

***เนื่องจากคำสั่งขนาดกว้าง 14 บิต ของ PIC จะช่วยทำให้ผลของรหัสคำสั่งมีอัตราส่วนเป็น 2.5 :1 เมื่อเทียบกับไมโครคอนโทรลเลอร์ ขนาด 8บิต อื่น ๆ มีคำสั่งที่ง่ายต่อการใช้และง่ายต่อการจำจึงช่วยลดเวลาที่ใช้ในการพัฒนา

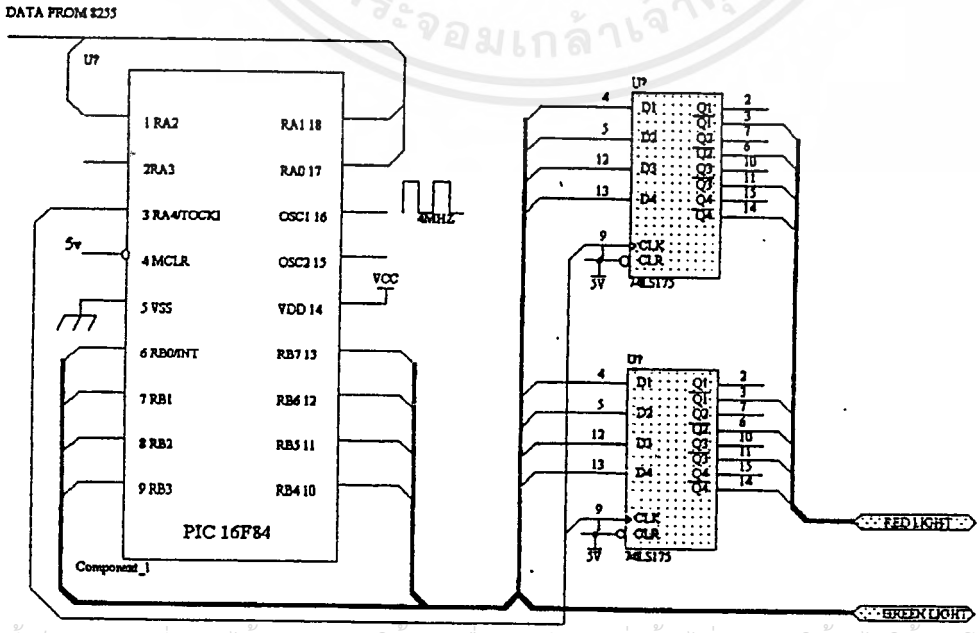
4.3.3 การทำงานในการเลือกรูปแบบของสัญญาณไฟ

ชิพ 8255 ทำหน้าที่นำค่าจาก บัสข้อมูล ถ่ายสู่ เอาท์พุทพอร์ต A ผ่านการประมวลผลโดย PIC อีกต่อหนึ่งเพื่อควบคุมไฟสัญญาณแต่ละดวง รูปแบบ ต่างๆ และการกระพริบไฟดังจะกล่าวต่อไป

P_{A6}	P_{A5}	P_{A4}	รูปแบบ
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

ตารางที่ 4.3 แสดงการเลือกรูปแบบของไฟจราจรของอินพุท PIC

ชิพ PIC 16F84 มี 2 พอร์ต โดยที่ พอร์ต A = 5 พิน , พอร์ต B = 8 พิน โดยที่การทำงานของระบบให้ พอร์ต A = อินพุท , พอร์ต B = เอาท์พุท



รูปที่ 4.6 แสดงแบบวงจรการต่อไมโครคอนโทรลเลอร์ PIC 16C84

เนื่องจากกำหนดให้ว่า เมื่อไฟเขียวเปลี่ยนเป็นไฟแดงจะทำการกระพริบไฟเขียวก่อนแล้วค่อยเปลี่ยนเป็นไฟแดง โดยที่หลักการกระพริบคือ นำ รูปแบบเก่า เอาท์พุท สลับ รูปแบบ ใหม่ จะได้ว่า

จาก 1→ 0	เขียว → แดง	เมื่อ เอาท์ พอร์ต สลับกันจะเกิดการกระพริบ
0→ 1	แดง → เขียว	ต้องบังคับมิให้เขียวกระพริบจึงต้อง เซต บิตนั้น เป็น 0 คือ 0→ 0 ก่อนแล้วจึงคืนค่าเก่ากลับคืนหลังกระพริบเรียบร้อยแล้ว

4.4 ดีเทกเตอร์ (Detector)

เป็นส่วนที่ติดตั้งขึ้นเพื่อใช้ในการตรวจสอบสภาพการจราจรในขณะนั้นว่าเป็นอย่างไร และจะส่งผลที่ได้ไปยังคอมพิวเตอร์เพื่อทำการประมวลผล

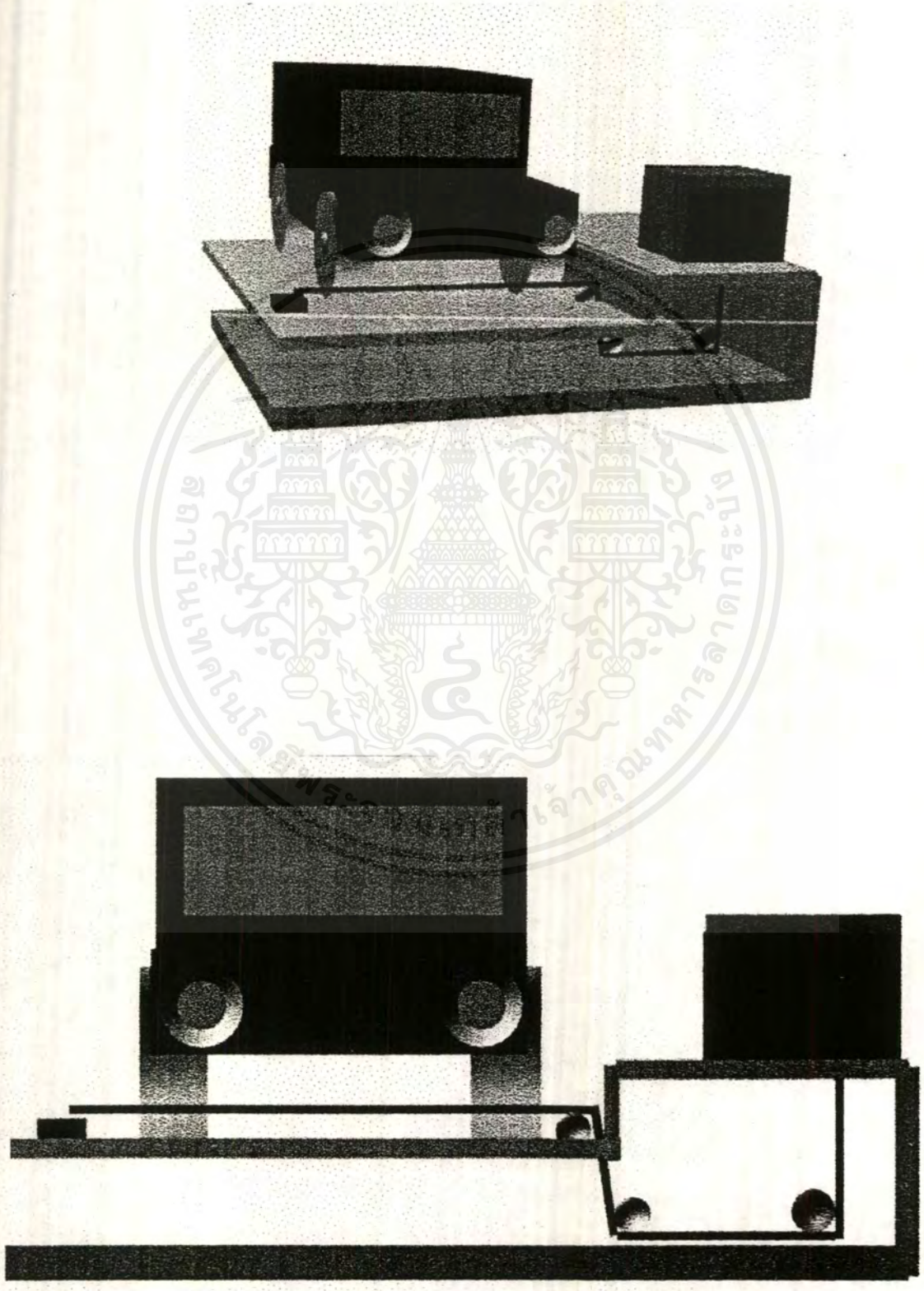
4.4.1 แนวคิดริเริ่ม

จากหลักทฤษฎีของเครื่องตรวจจับแบบต่างๆจะเห็นได้ว่าสามารถแบ่งเป็น 2 พวกใหญ่ๆได้ดังนี้ คือ เครื่องตรวจจับแบบอิเล็กทรอนิกส์ และ เครื่องตรวจจับแบบแมคคานิกส์ ซึ่งจะเห็นได้ว่าแบบแมคคานิกส์จะมีข้อดีมากกว่าโดยไม่ต้องกังวลถึงสภาวะแวดล้อมต่างๆเช่น น้ำท่วม , หมอกลง หรือมีฝุ่นละอองในอากาศมากเกินไป อันจะทำให้เครื่องตรวจจับแบบอิเล็กทรอนิกส์ เช่น อัลตราโซนิกดีเทคเตอร์ เกิดการผิดพลาดได้ จึงได้นำไปสู่การออกแบบเครื่องตรวจจับที่ใช้แบบแมคคานิกส์นี้ โดยมีหลักการดังนี้

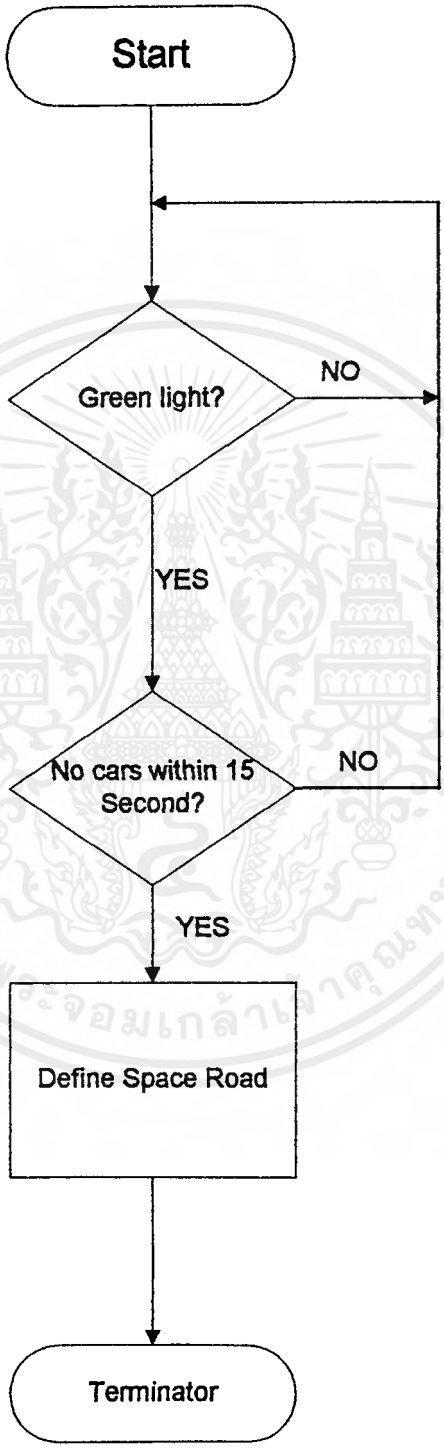
4.4.2 หลักการ

เป็นดังนี้คือ เมื่อมีช่องทางใดได้รับสัญญาณไฟเขียวจะทำการตรวจสอบการไหลของรถยนต์โดยจะมีการตั้งค่าเวลาไว้ค่าหนึ่งแล้วตรวจสอบว่า ภายในเวลาที่ตั้งไว้มีรถวิ่งผ่านหรือไม่ ถ้ามีดีเทกเตอร์จะไม่ส่งสัญญาณไปยังคอมพิวเตอร์ คอมพิวเตอร์จะทำการควบคุมสัญญาณไฟตามปกติ แต่ถ้าหากว่าภายในเวลาที่กำหนดไม่มีรถวิ่งผ่าน ดีเทกเตอร์จะส่งสัญญาณไปให้คอมพิวเตอร์เพื่อให้เปลี่ยนโหมดในการเปิดสัญญาณไฟจราจร

รูปแบบของส่วนตรวจจับจะเป็นดังรูปที่ 4.7



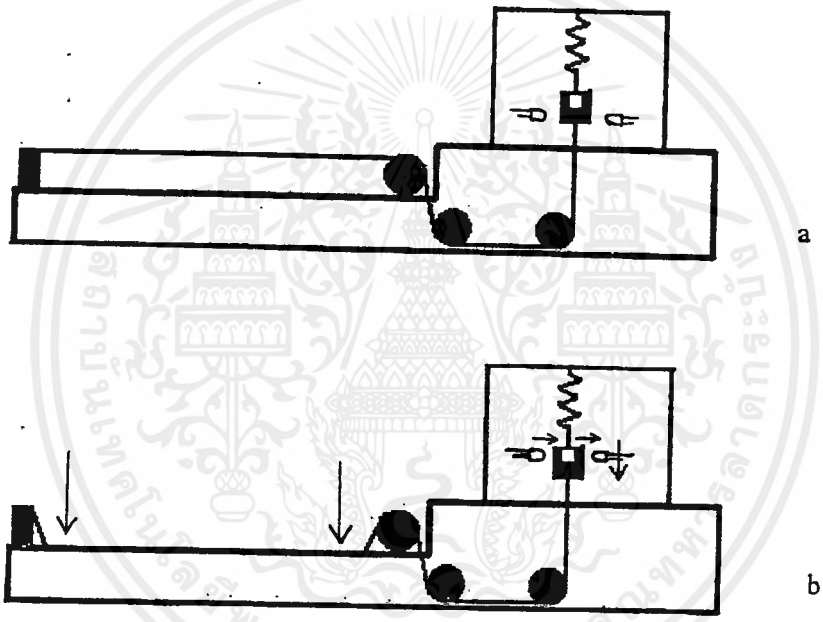
รูปที่ 4.7 แสดงรูปแบบการติดตั้งในการใช้งานของดีเทคเตอร์



รูปที่ 4.8 แสดง Flow Chart การทำงานของดีเทคเตอร์

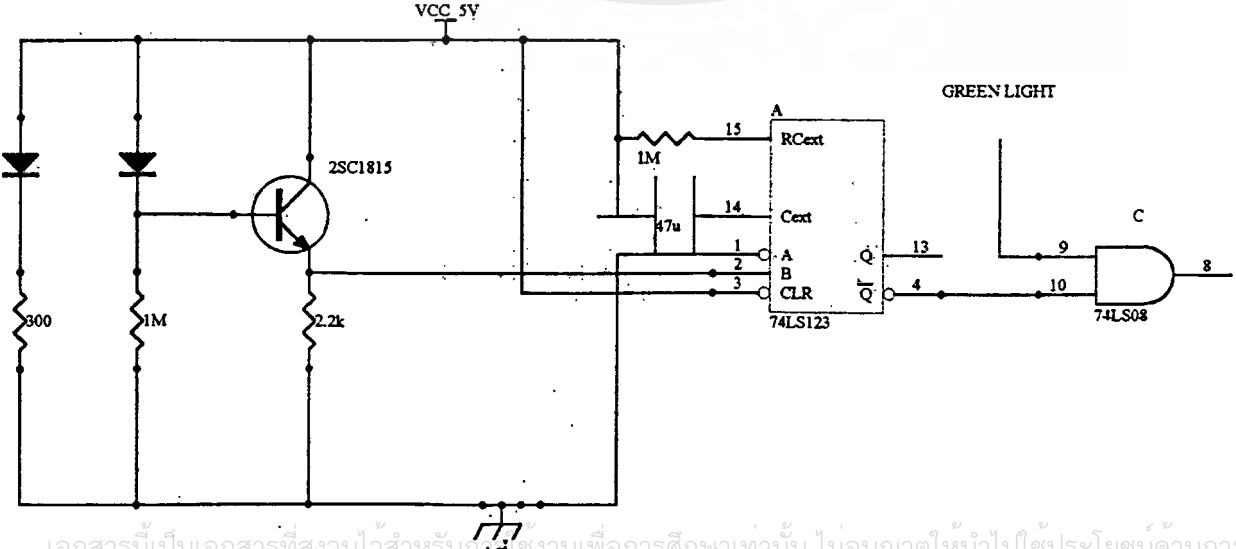
จะใช้ลวดสลิงขึงตึงสูงจากพื้นถนนเล็กน้อยใน 1 ช่องทาง โดยด้านหนึ่งจะยึดไว้ตายตัว ส่วนอีกด้านหนึ่งจะนำไปผ่านระบบรอกตายตัวลอดผ่านใต้พื้นถนน และจะมีวงจรติดตั้งอยู่บริเวณบนฟุตบาทหรือเกาะกลาง ดังรูป โดยวงจรที่ใช้เป็นวงจรส่งและรับแสงอินฟาเรด

เมื่อมีรถวิ่งผ่านรถจะเหยียบไปบนเส้นลวด ทำให้เส้นลวดเคลื่อนที่เสมือนว่าถูกดึง โดยปลายอีกด้านของเส้นลวดจะถูกยึดไว้กับสปริงเพื่อให้เมื่อรถวิ่งผ่านไปสปริงจะดึงเส้นลวดให้กลับมายังตำแหน่งเดิม โดยจะทำการติดตั้งแผ่นกั้นแสงไว้ที่ส่วนหนึ่งของเส้นลวดเพื่อทำการเปิดและกั้นช่องทางของแสง กล่าวคือเมื่อรถเหยียบเส้นลวดจะทำการเปิดแผ่นกั้นให้แสงผ่านได้นั่นเอง



รูปที่ 4.9 แสดงลักษณะการทำงานของดีเทคเตอร์เมื่อ a. ไม่มีรถผ่าน b. มีรถผ่าน

4.4.3 วงจรดีเทคเตอร์



รูปที่ 4.10 แสดงวงจรดีเทคเตอร์

อุปกรณ์ที่ใช้

1. ตัวต้านทาน 300 โอห์ม 1 ตัว
   ตัวต้านทาน 2.2K โอห์ม 1 ตัว
   ตัวต้านทาน 1M โอห์ม 2 ตัว
2. ทรานซิสเตอร์ 2SC1815 1 ตัว
3. ตัวเก็บประจุ 47 ไมโครฟารัด 1 ตัว
4. ไอซี 74LS08 1 ตัว
   ไอซี 74LS123 1 ตัว

ใช้ LED ส่งและรับอินฟาเรด จะนำเอาท์พุทที่ได้จากตัวรับแสงผ่านทรานซิสเตอร์เพื่อทำการขยายกระแส และนำไปเข้าเป็นอินพุทของไอซี 74 แอลเอส 123 ซึ่งเป็น ไอซีรีทริกเกอร์เอเบิลไมโนสเตเบิลมัลติไวเบรเตอร์ ซึ่งจะทำการตั้งค่าเวลาโดยคำนวณจากค่า ตัวต้านทาน และ ตัวเก็บประจุ ดังสมการ

$$Tr = 0.45\,RC$$

ในช่วงเวลาที่กำหนดไว้หากไม่มีสัญญาณ รีทริก จะส่งค่าเอาท์พุทออกมาและจะนำสัญญาณเอาท์พุทจากไอซี 74 แอลเอส 123 ไปแอนกับ สัญญาณไฟเขียวซึ่งมีสถานะทางลอจิกเป็น 1 และส่งเอาท์พุทไปยังคอมพิวเตอร์

บทที่ 5
โปรแกรมการทำงาน

5.1 แนวทางการเขียนโปรแกรมให้เหมาะกับระบบกลุ่มสี่แยก

สำหรับระบบกลุ่มที่นำมาใช้ศึกษาควบคู่ไปกับการเขียนโปรแกรมคือกลุ่มสี่แยกดังต่อไปนี้ สี่แยกลำสาลี , สี่แยกบางกะปิ , สามแยกบางกะปิ , สี่แยกหมู่บ้านสวนสน ซึ่งมีลักษณะทางกายภาพดังรูปที่ 5.1

ซึ่งจากการได้ลองไปเก็บข้อมูลเกี่ยวกับปริมาณรถยนต์ที่วิ่งเข้า - ออก จะได้ลักษณะของรูปแบบการปล่อยรถดังต่อไปนี้

สี่แยกลำสาลี จะใช้โหมด 3 ซึ่งมีลักษณะการปล่อยรถยนต์ตามรูปแบบที่ 1 , 4 , 0 , 2 , 6 , 3 ตามลำดับ

สี่แยกบางกะปิ ใช้โหมด 3 ซึ่งมีลักษณะการปล่อยรถยนต์ตามรูปแบบที่ 1 , 4 , 0 , 2 , 6 , 3 ตามลำดับ

สามแยกบางกะปิ จะใช้โหมดสำหรับสามแยก

สี่แยกหมู่บ้านสวนสน จะใช้โหมด 1 ซึ่งมีลักษณะการปล่อยรถยนต์ตามรูปแบบที่ 0 , 1 , 2 , 3 ตามลำดับ

โดยฐานเวลาของการปล่อยรถยนต์ของแต่ละรูปแบบจะขึ้นอยู่กับความหนาแน่นของรถยนต์ตามข้อมูลที่ได้เก็บมา

5.2 Flow Chart โปรแกรมหลักควบคุมโดย IBM คอมพิวเตอร์ ที่ใช้ภาษาซี

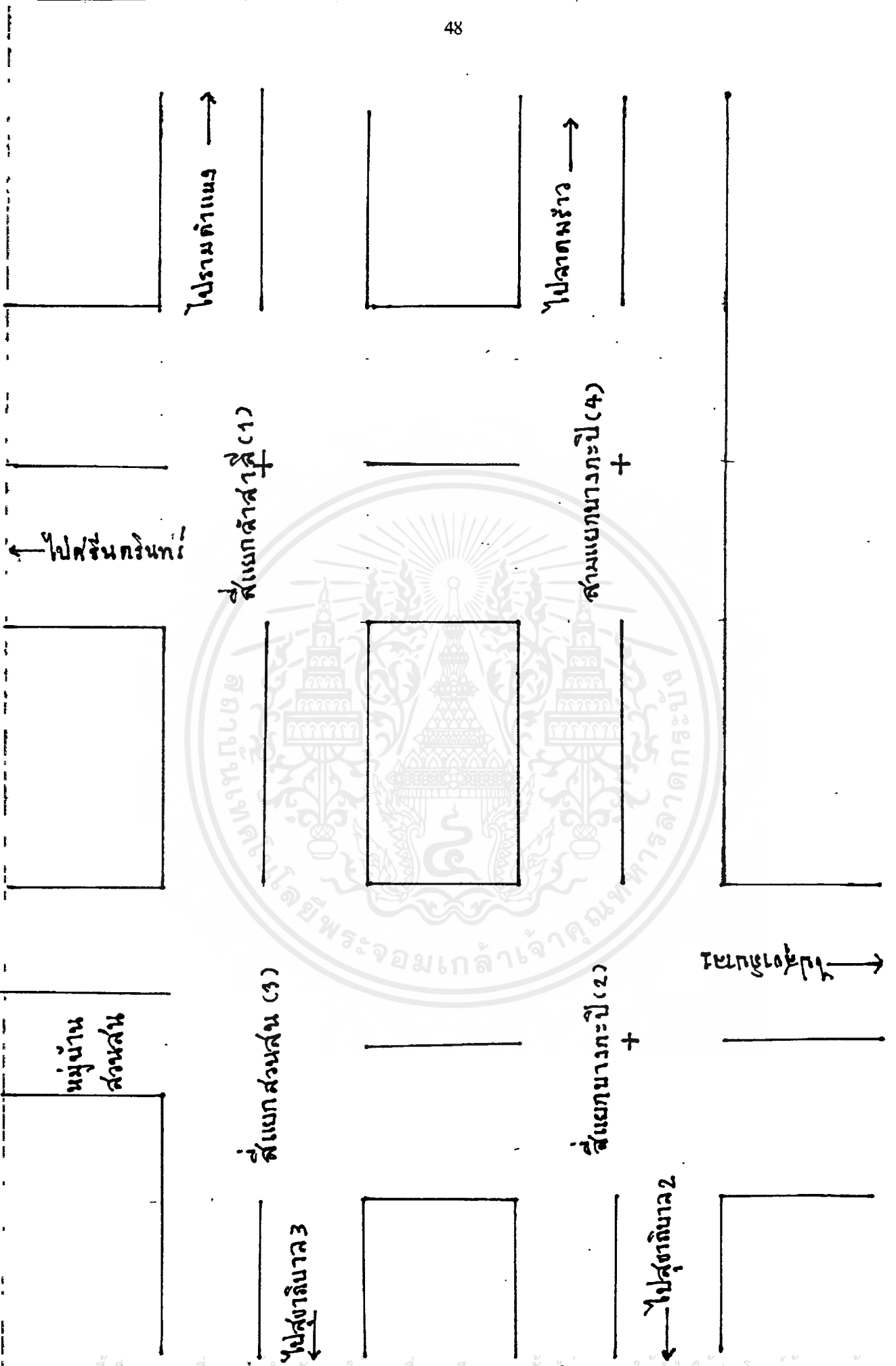
5.2.1 รูปที่ 5.2 แสดง Flow Chart โปรแกรมการควบคุมหลัก

5.3 Flow Chart โปรแกรมย่อยควบคุมโดย ไมโครคอนโทรลเลอร์ตระกูล PIC ที่ใช้ภาษาแอสแซมบลี

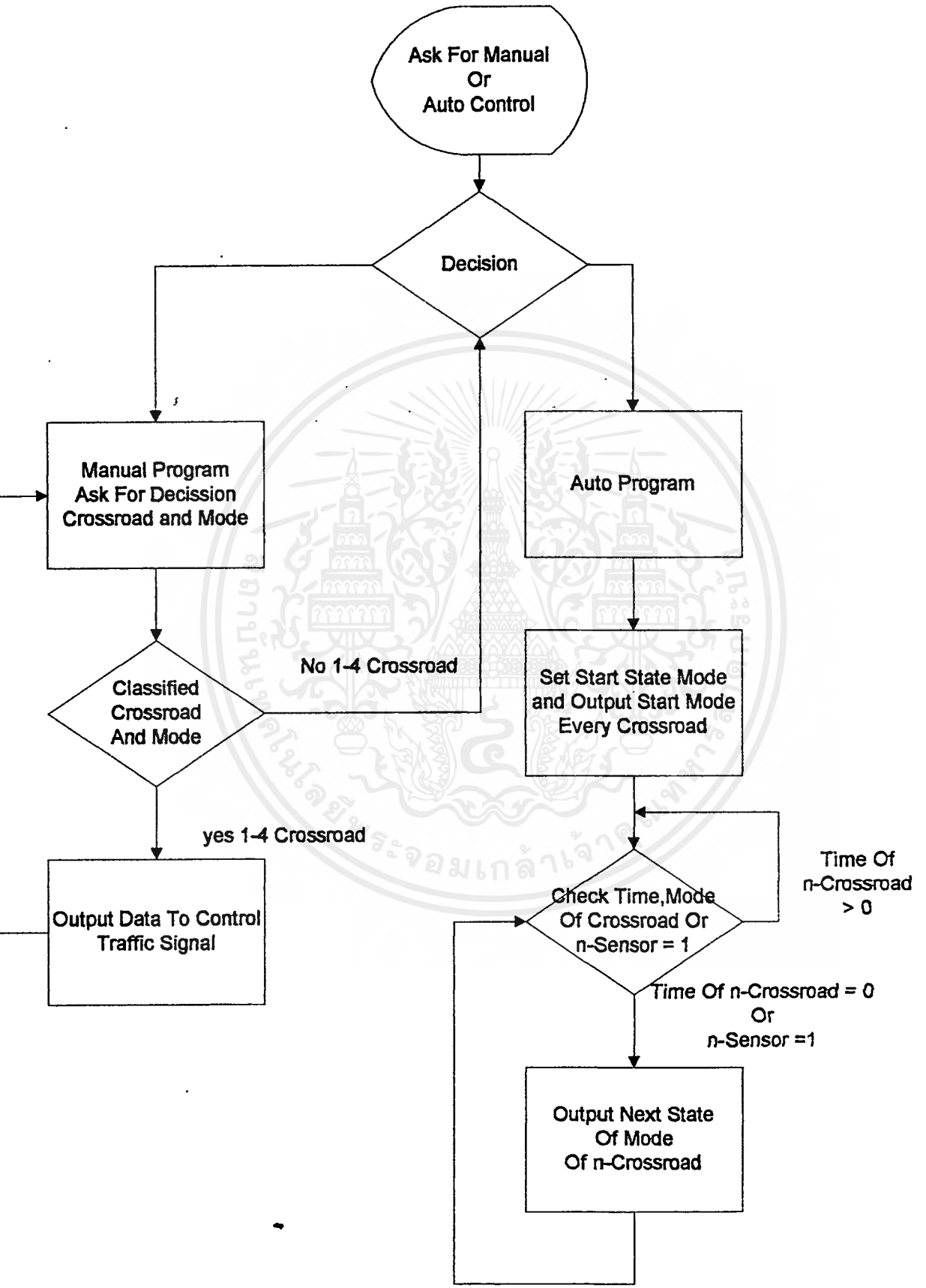
5.3.1 รูปที่ 5.3 แสดง Flow Chart โดยรวม

5.3.2 รูปที่ 5.5 แสดง Flow Chart Subroutine Check

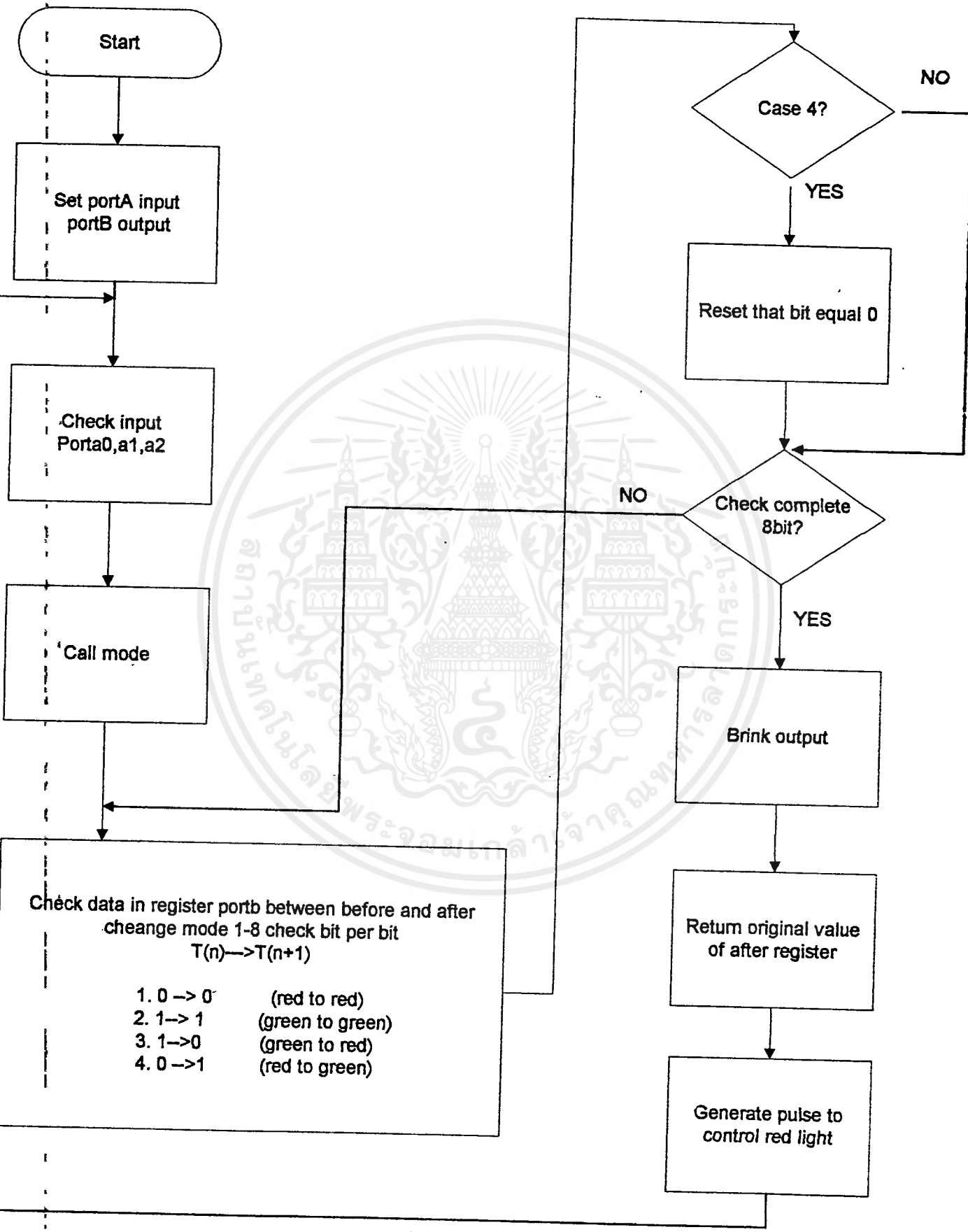
5.3.3 รูปที่ 5.5 แสดง Flow Chart Subroutine Brink



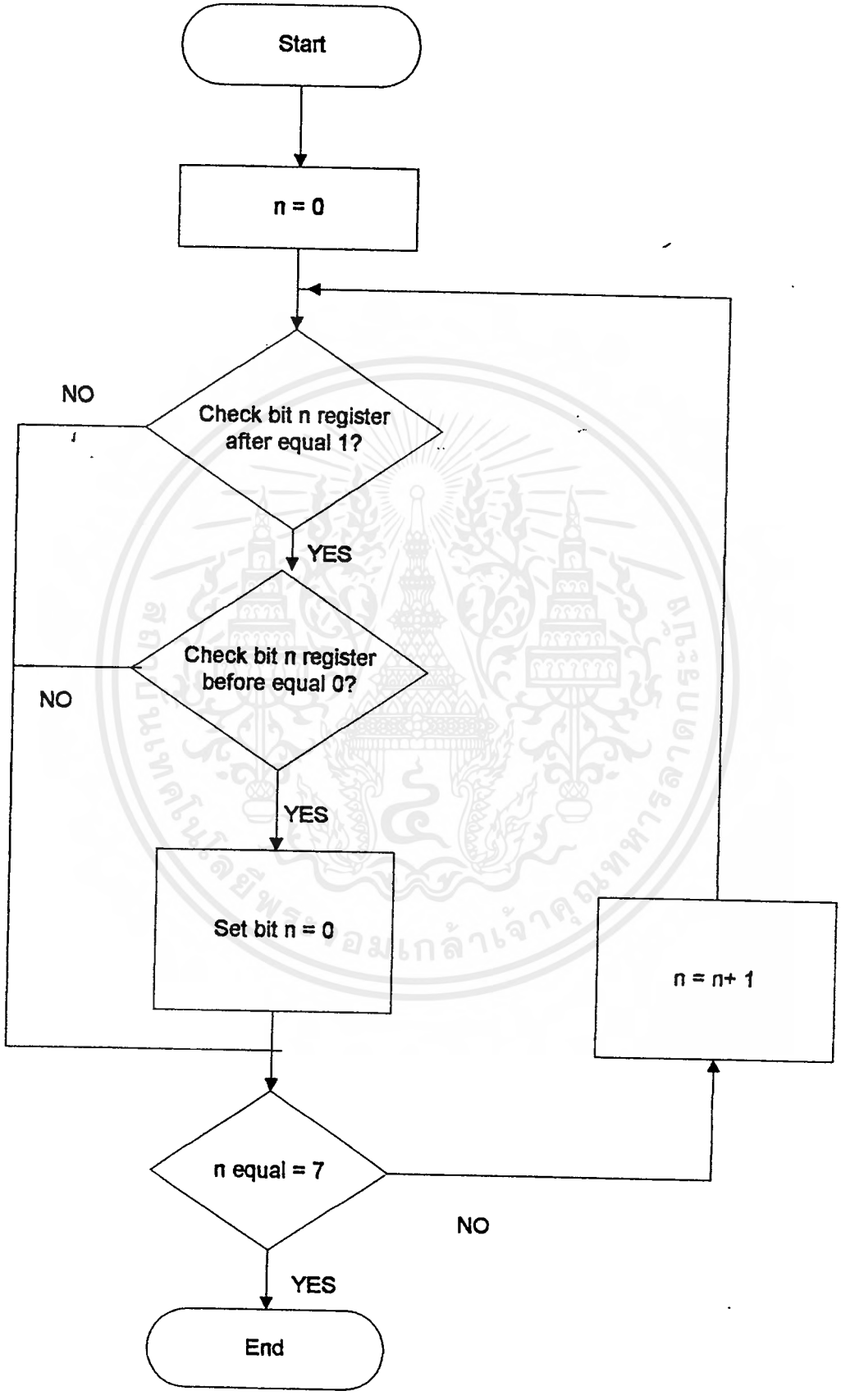
รูปที่ 5.1 แสดงลักษณะทางกายภาพของกลุ่มสี่แยกบางกะปี



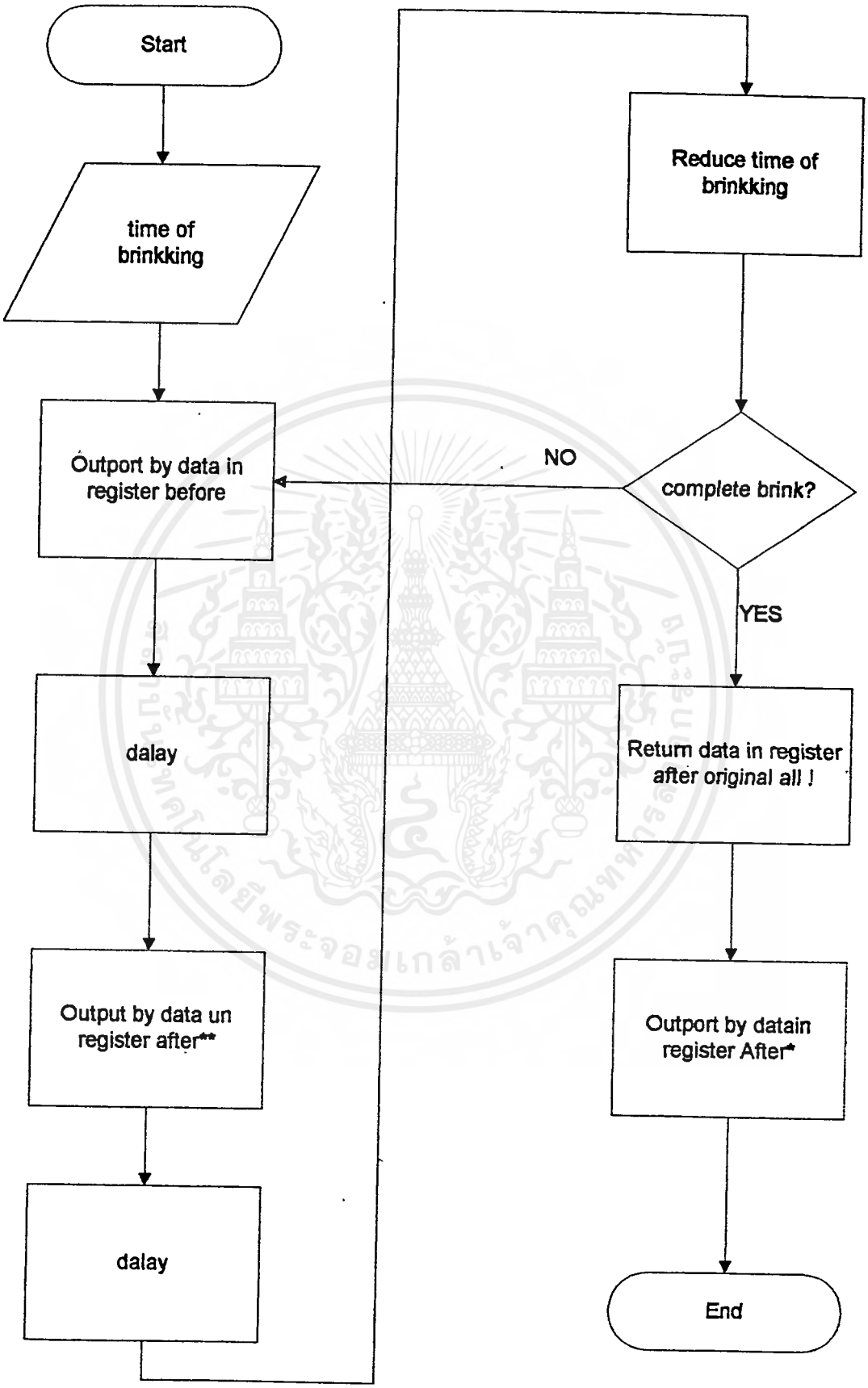
รูปที่ 5.2 แสดง Flow Chart โปรแกรมการควบคุม



รูปที่ 5.3 แสดง Flow Chart โดยรวม



รูปที่ 5.4 แสดง Flow Chart Subroutine Check



รูปที่ 5.5 แสดง Flow Chart Subroutine Brink

บทที่ 6
สรุปและวิจารณ์

6.1 สรุปและวิจารณ์

ระบบควบคุมสัญญาณไฟจราจรที่ใช้กันอยู่นั้นมีหลายประเภท ในส่วนของปริญญานิพนธ์นี้ได้ทำการเสนอแนวความคิดใหม่ทางด้านรูปแบบของไฟจราจรและดีเทคเตอร์ แต่ยังคงยึดแก่นของระบบจราจรเอาไว้ ซึ่งโปรแกรมภาษาซีที่ได้ทำการทดลองนี้ได้ทำการออกแบบมาเฉพาะกลุ่มสี่แยกย่านลำสาลีและบางกะปิ หากต้องการประยุกต์ใช้กับกลุ่มสี่แยกอื่นๆต้องทำการพิจารณาค่าตัวแปรและฐานเวลาใหม่ส่วนโปรแกรมภาษาแอสแซมบลีสามารถใช้ได้กับสี่แยกและสามแยกทุกประเภท ในส่วนของดีเทคเตอร์นั้นเป็นลักษณะของแบบจำลอง ซึ่งในการนำไปประยุกต์ใช้หรือติดตั้งจริงนั้นอาจจะต้องมีการพัฒนาให้เหมาะสมยิ่งขึ้น ซึ่งจะแยกเป็นส่วนๆอธิบายได้ดังนี้

6.2.1 ส่วนโปรแกรมการทำงาน

ในส่วนของโปรแกรมการทำงานที่ได้เขียนขึ้นมานั้นถือว่าสามารถทำงานได้ดี ถ้าจะเปรียบเทียบแล้วก็คงจะทำงานได้ไม่น้อยกว่าการตัดสินใจของตำรวจจราจร 4คนที่ควบคุมอยู่คนละ 1 สี่แยก เพราะเมื่อนำชุดทดลองไปเชื่อมต่อกับส่วนของภาคดีเทคเตอร์ โดยใช้คอมพิวเตอร์เป็นตัวสั่งการแล้วปรากฏว่าสามารถทำงานได้ตามโปรแกรมทุกอย่าง ซึ่งตัวของโปรแกรมถ้าจะผิดพลาดก็คงเป็นส่วนของการคิดฐานเวลาให้กับกลุ่มสี่แยกซึ่งต้องอาศัยการเก็บข้อมูลให้มากกว่านี้ ค่าฐานเวลาที่ได้จึงจะได้ค่าที่เหมาะสมมากขึ้น

6.2.2 ส่วนภาคแสดงผล(ชุดทดลอง)

เนื่องจากใช้ชุดทดลองที่เลียนแบบมาจากของจริง จึงไม่น่าจะมีการผิดพลาดทางการแสดงผล แต่พอทำชุดทดลองจนครบทั้ง 4 สี่แยกก็ได้ทราบถึงจุดบกพร่องซึ่งก็คือการแยกให้แต่ละสี่แยกออกจากกัน ทำให้สายที่ใช้เชื่อมต่อระหว่างสี่แยกมีความซับซ้อนตามไปด้วย ซึ่งตลอดเวลาที่ทำการทดลองก็จะเจอแบบเดียวกันตลอดก็คือการที่มีสายในการเชื่อมต่อเยอะทำให้เกิดการหลวมของขั้วหัวต่อทำให้เกิดสถานะลอย ซึ่งทำให้ทั้งการรับค่าของโปรแกรมผิดพลาดและการส่งค่าไปยังไมโครคอนโทรลเลอร์ผิดพลาดอยู่เสมอ แต่ถ้าได้มีการขยับขั้วหัวต่อให้ดีก็จะลดปัญหาตรงนี้ไปได้ ซึ่งการแก้ไขควรจะทำชุดทดลองอยู่บนชุดเดียวกันและควรจะใช้วงจรแสดงผลที่ออกแบบมาให้อยู่บนแผ่นวงจรเดียวกันด้วย ก็น่าจะขจัดปัญหาขั้วหัวต่อหลวมไปได้

6.2.3 ส่วนไมโครคอนโทรลเลอร์

การทำงานของส่วนไมโครคอนโทรลเลอร์นั้นไม่ค่อยเจอปัญหามากนักเพราะการทำลายวงจรได้ไปให้ร้านทำให้ จึงไม่ค่อยเจอปัญหาแต่ปรากฏว่าปัญหาที่เกิดขึ้นกลับมาจากการที่

ร้านกัดลายวงจรส่วนของคอนเนคเตอร์มาไม่ตรงกับรูของหัวต่อก้างปลาทำให้เกิดปัญหาลามไปถึงภาคอื่นๆด้วย ซึ่งถ้าทางร้านทำรูของคอนเนคเตอร์ให้ตรงกับที่ได้ออกแบบไว้ก็คงจะไม่เกิดปัญหาเช่นนี้ขึ้น ส่วนการทำงานก็ทำงานได้ดีตามที่ได้ลองออกแบบมาตั้งแต่อยู่ในบอร์ดทดลอง และอีกส่วนหนึ่งก็คือส่วนของการกระพริบที่ไม่ค่อยพร้อมกันในแต่ละสี่แยกนั้นเป็นผลมาจากการใช้ตัวต้านทานและตัวเก็บประจุเป็นออสซิลเลเตอร์ ซึ่งทำให้มีการผิดพลาดที่ควบคุมไม่ได้ขึ้นเนื่องจากค่าที่ใช้แม้จะเป็นค่าเดียวกันแต่ก็ยังมีค่าความผิดพลาด 5 % อยู่ดีซึ่งปัญหานี้น่าจะหมดไปถ้าใช้ออสซิลเลเตอร์สำเร็จรูปที่มีคุณภาพดีกว่านี้

6.2.4 ส่วนโปรแกรมภาษาแอสแซมบลี

การทำงานก็จะมีปัญหาในการที่จะออกแบบอย่างไรให้มีการกระพริบให้เหมาะสมเท่านั้น ซึ่งเมื่อได้ลองออกแบบไปเรื่อยๆก็สามารถสั่งให้มีการทำงานตามโปรแกรมที่ต้องการได้

6.2.5 ส่วนดีเทคเตอร์

เนื่องจากเป็นดีเทคเตอร์ที่ได้ออกแบบขึ้นมาใหม่จึงอาจมีข้อผิดพลาดได้ค่อนข้างมาก ซึ่งทางคณะผู้จัดทำได้ลองแก้ไขปัญหาในเบื้องต้นเท่าที่มองเห็นไปบ้างแล้ว ซึ่งก็คิดว่าโดยหลักการสามารถนำไปใช้งานจริงได้ เพียงแต่ต้องมีการศึกษาถึงปัจจัยภายนอกอื่นๆก่อนเท่านั้น เช่น ความสูงของเส้นลวด , ขนาดของเส้นลวด , จะออกแบบให้กลมกลืนกับผิวถนนได้อย่างไร , ความเร็วของยานพาหนะเป็นเท่าไรจึงจะตรวจสอบได้และไม่ได้ เป็นต้นซึ่งทางผู้จัทำคิดว่าหากได้มีการพัฒนาแบบจำลองนี้อย่างต่อเนื่องก็น่าจะสามารถแก้ปัญหาเหล่านี้ได้ สำหรับข้อดีที่เห็นได้ชัดก็คือ มีต้นทุนที่ต่ำ , มีเทคโนโลยีที่เกี่ยวข้องด้วยไม่สูงนัก , สามารถทำงานได้ทุกสภาพแวดล้อมไม่ว่าฝนตกหรือ น้ำท่วมหรือหมอกลงหรือมีฝุ่นละอองในอากาศสูงก็ตาม

ซึ่งผลการทำงานโดยรวมของระบบควบคุมสัญญาณไฟจราจรนี้นับว่าทำงานได้ตรงตามจุดประสงค์ที่ได้วางไว้ทุกประการ และถ้าได้มีการพัฒนาต่อไปทางคณะผู้จัดทำเชื่อว่าจะสามารถนำไปใช้ในการแก้ปัญหาการจราจรของกรุงเทพได้ โดยไม่ต้องไปซื้อเทคโนโลยีของต่างประเทศที่มีราคาแพงมาใช้ได้

บทที่ 7 ภาคผนวก

7.1 แสดงโปรแกรมการทำงานที่เขียนโดยภาษาซี

7.2 แสดงโปรแกรมภาษาแอสแซมบลี

```
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <process.h>
#define port_A 0x200
#define port_B 0x201
#define port_C 0x202
#define ctrl  0x203
main()
{     int type,tc1,tc1case,tc2,tc2case,tc3,tc3case,tc4,tc4case;
/*          int sensor11,sensor14,sensor10,sensor12,sensor16,sensor13;
            int sensor21,sensor24,sensor20,sensor22,sensor26,sensor23;
            int sensor30,sensor31,sensor32,sensor33;*/
            int/* sensor40*/sensor41/*,sensor42,sensor43*/;
            clrscr();
            outp(port_C,0x89);
            textmode(14);
loop1:      clrscr();
            printf("\n \rMain Program Of Traffic Light Control");
            printf("\n \rManual Control    :1 ");
            printf("\n \rAutomatic Control :2 ");
            printf("\n \rPlease Choose Type Of Control :");
            scanf("%d",&type);
            switch(type)
            {
            case 1:
                        spawnv(P_WAIT,"tman1234",NULL);
                        break;
            case 2:
                        printf("You are in Automatic Program Of Traffic Light Control");
                        break;
            default:
                        goto loop1;
            }
            tc1case=1;
            tc1=18;
            tc2case=1;
            tc2=6;
            tc3case=0;
            tc3=6;
            tc4case=0;
            tc4=9;
/*          sensor40=0;*/
            sensor41=0;
/*          sensor42=0;
            sensor43=0;*/
            goto loopttl;
loopsc1:switch(tc1case)
            {
                        case 1:
                        {           tc1=18;
                                    tc1case=1;
                                    outp(port_A,0x10);
                                    outp(port_A,0x90);
                                    break;
                        }
```

```
                case 4:
                {       tc1=18;
                        tc1case=4;
                        outp(port_A,0x40);
                        outp(port_A,0xc0);
                        break;
                }
                case 0:
                {       tc1=18;
                        tc1case=0;
                        outp(port_A,0x00);
                        outp(port_A,0x80);
                        break;
                }
                case 2:
                {       tc1=6;
                        tc1case=2;
                        outp(port_A,0x20);
                        outp(port_A,0xa0);
                        break;
                }
                case 6:
                {       tc1=6;
                        tc1case=6;
                        outp(port_A,0x60);
                        outp(port_A,0xe0);
                        break;
                }
                case 3:
                {       tc1=6;
                        tc1case=3;
                        outp(port_A,0x30);
                        outp(port_A,0xb0);
                        break;
                }
                default:
                        break;
        }
        goto loopct2;
loopsc2:switch(tc2case)
        {       case 1:
                {       tc2=6;
                        tc2case=1;
                        outp(port_A,0x11);
                        outp(port_A,0x91);
                        break;
                }
                case 4:
                {       tc2=9;
                        tc2case=4;
                        outp(port_A,0x41);
                        outp(port_A,0xc1);
                        break;
                }
                case 0:
                {       tc2=12;
                        tc2case=0;
```

```
                outp(port_A,0x01);
                outp(port_A,0x81);
                break;
        }
        case 2:
        {       tc2=6;
                tc2case=2;
                outp(port_A,0x21);
                outp(port_A,0xa1);
                break;
        }
        case 6:
        {       tc2=12;
                tc2case=6;
                outp(port_A,0x61);
                outp(port_A,0xe1);
                break;
        }
        case 3:
        {       tc2=6;
                tc2case=3;
                outp(port_A,0x31);
                outp(port_A,0xb1);
                break;
        }
        default:
                break;
    }
    goto loopct3;
loopsc3:switch(tc3case)
    {   case 0:
        {       tc3=6;
                tc3case=0;
                outp(port_A,0x02);
                outp(port_A,0x82);
                break;
        }
        case 1:
        {       tc3=6;
                tc3case=1;
                outp(port_A,0x12);
                outp(port_A,0x92);
                break;
        }
        case 2:
        {       tc3=6;
                tc3case=2;
                outp(port_A,0x22);
                outp(port_A,0xa2);
                break;
        }
        case 3:
        {       tc3=6;
                tc3case=3;
                outp(port_A,0x32);
                outp(port_A,0xb2);
                break;
```

```
                }
                default:
                        break;
        }
        goto loopct4;
loopsc4:switch(tc4case)
        {       case 0:
                {       tc4=15;
                        tc4case=0;
                        outp(port_A,0x03);
                        outp(port_A,0x83);
                        break;
                }
                case 1:
                {       tc4=15;
                        tc4case=1;
                        outp(port_A,0x13);
                        outp(port_A,0x93);
                        break;
                }
                case 2:
                {       tc4=15;
                        tc4case=2;
                        outp(port_A,0x23);
                        outp(port_A,0xa3);
                        break;
                }
                case 3:
                {       tc4=15;
                        tc4case=3;
                        outp(port_A,0x33);
                        outp(port_A,0xb3);
                        break;
                }
                default:
                        break;
        }
        goto loopct1;
loopttl:  while(kbhit()) goto loop1;    /*Check Time*/
loopct1:if(tc1==0&&tc1case==1/*||sensor11==5*/)
                {       tc1case=4;
                        goto loopsc1;
                }
                if(tc1==0&&tc1case==4/*||sensor14==5*/)
                {       tc1case=0;
                        goto loopsc1;
                }
                if(tc1==0&&tc1case==0/*||sensor10==5*/)
                {       tc1case=2;
                        goto loopsc1;
                }
                if(tc1==0&&tc1case==2/*||sensor12==5*/)
                {       tc1case=6;
                        goto loopsc1;
                }
                if(tc1==0&&tc1case==6/*||sensor16==5*/)
                {       tc1case=3;
```

```
                goto loopsc1;
        }
        if(tc1==0&&tc1case==3/*||sensor13==5*/)
        {       tc1case=1;
                goto loopsc1;
        }
loopct2:if(tc2==0&&tc2case==1/*||sensor21==5*/)
        {       tc2case=4;
                goto loopsc2;
        }
        if(tc2==0&&tc2case==4/*||sensor24==5*/)
        {       tc2case=0;
                goto loopsc2;
        }
        if(tc2==0&&tc2case==0/*||sensor20==5*/)
        {       tc2case=6;
                goto loopsc2;
        }
        if(tc2==0&&tc2case==2/*||sensor22==5*/)
        {       tc2case=6;
                goto loopsc2;
        }
        if(tc2==0&&tc2case==6/*||sensor26==5*/)
        {       tc2case=3;
                goto loopsc2;
        }
        if(tc2==0&&tc2case==3/*||sensor23==5*/)
        {       tc2case=1;
                goto loopsc2;
        }
loopct3:if(tc3==0&&tc3case==0/*||sensor30==5*/)
        {       tc3case=1;
                goto loopsc3;
        }
        if(tc3==0&&tc3case==1/*||sensor31==5*/)
        {       tc3case=2;
                goto loopsc3;
        }
        if(tc3==0&&tc3case==2/*||sensor32==5*/)
        {       tc3case=3;
                goto loopsc3;
        }
        if(tc3==0&&tc3case==3/*||sensor33==5*/)
        {       tc3case=0;
                goto loopsc3;
        }
loopct4:if(tc4==0&&tc4case==0/*||sensor40==5*/)
        {       tc4case=1;
                goto loopsc4;
        }
        if(tc4==0&&tc4case==1/*||sensor41==5*/)
        {       tc4case=2;
                goto loopsc4;
        }
        if(tc4==0&&tc4case==2||sensor41==1)
        {       tc4case=3;
                sensor41=0;
```

```
            goto loopsc4;
}
if(tc4==0&&tc4case==3/*||sensor43==5*/)
{           tc4case=0;
            goto loopsc4;
}
tc1=tc1-1;
tc2=tc2-1;
tc3=tc3-1;
tc4=tc4-1;
if(inp(port_C)==0xff)
{           sensor41=sensor41+1;
}
printf(" %d ",sensor41);
sleep(1);
goto looptt;
}
```

```
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#define port_A 0x200
#define port_B 0x201
#define port_C 0x202
#define ctrl  0x203
main()
{       int  time1,time2,mode,cross;
        clrscr();
        outp(port_C,0x89);
loop1:  printf("\n \rYou are in Manual Program Traffic Light Control");
        printf("\n \rCrossroad 1 :1");
        printf("\n \rCrossroad 2 :2");
        printf("\n \rCrossroad 3 :3");
        printf("\n \rCrossroad 4 :4");
        printf("\n \rType other for come back to Main Program");
        printf("\n \rEnter Number Of Crossroad To Control :");
        scanf("%d",&cross);
        switch(cross)
        {       case 1:
                {       printf("\rWhat Mode Do You Want To Use :");
                        scanf("%d",&mode);
                        break;
                }
                case 2:
                {       printf("\rWhat Mode Do You Want To Use :");
                        scanf("%d",&mode);
                        break;
                }
                case 3:
                {       printf("\rWhat Mode Do You Want To Use :");
                        scanf("%d",&mode);
                        break;
                }
                case 4:
                {       printf("\rWhat Mode Do You Want To Use :");
                        scanf("%d",&mode);
                        break;
                }
                default:
                        goto loop5;
        }
        switch(mode)
                {       case 0:
                        {       switch(cross)
                                {       case 1:
                                        {       outp(port_A,0x00);
                                                outp(port_A,0x80);
                                                break;
                                        }
                                        case 2:
                                        {       outp(port_A,0x01);
                                                outp(port_A,0x81);
                                                break;
                                        }
```

```
                    case 3:
                    {       outp(port_A,0x02);
                            outp(port_A,0x82);
                            break;
                    }
                    case 4:
                    {       outp(port_A,0x03);
                            outp(port_A,0x83);
                            break;
                    }
                    default:
                            goto loop1;

            }
            break;
    }
    case 1:
    {       switch(cross)
            {       case 1:
                    {       outp(port_A,0x10);
                            outp(port_A,0X90);
                            break;
                    }
                    case 2:
                    {       outp(port_A,0x11);
                            outp(port_A,0x91);
                            break;
                    }
                    case 3:
                    {       outp(port_A,0x12);
                            outp(port_A,0x92);
                            break;
                    }
                    case 4:
                    {       outp(port_A,0x13);
                            outp(port_A,0x93);
                            break;
                    }
                    default:
                            goto loop1;
            }
    break;
    }
    case 2:
    {       switch(cross)
            {       case 1:
                    {       outp(port_A,0x20);
                            outp(port_A,0xA0);
                            break;
                    }
                    case 2:
                    {       outp(port_A,0x21);
                            outp(port_A,0xA1);
                            break;
                    }
                    case 3:
                    {       outp(port_A,0x22);
```

```
                        outp(port_A,0xA2);
                        break;
                }
                case 4:
                {       outp(port_A,0x23);
                        outp(port_A,0xA3);
                        break;
                }
                default:
                        goto loop1;
        }
break;
}
case 3:
{       switch(cross)
        {       case 1:
                {       outp(port_A,0x30);
                        outp(port_A,0xB0);
                        break;
                }
                case 2:
                {       outp(port_A,0x31);
                        outp(port_A,0xB1);
                        break;
                }
                case 3:
                {       outp(port_A,0x32);
                        outp(port_A,0xB2);
                        break;
                }
                case 4:
                {       outp(port_A,0x33);
                        outp(port_A,0xB3);
                        break;
                }
                default:
                        goto loop1;
        }
break;
}
case 4:
{       switch(cross)
        {       case 1:
                {       outp(port_A,0x40);
                        outp(port_A,0xC0);
                        break;
                }
                case 2:
                {       outp(port_A,0x41);
                        outp(port_A,0xC1);
                        break;
                }
                case 3:
                {       outp(port_A,0x42);
                        outp(port_A,0xC2);
                        break;
                }
```

```
                    case 4:
                            goto loop2;
                    default:
                            goto loop1;
            }
    break;
    }
    case 5:
    {       switch(cross)
            {       case 1:
                    {       outp(port_A,0x50);
                            outp(port_A,0xD0);
                            break;
                    }
                    case 2:
                    {       outp(port_A,0x51);
                            outp(port_A,0xD1);
                            break;
                    }
                    case 3:
                    {       outp(port_A,0x52);
                            outp(port_A,0xD2);
                            break;
                    }
                    case 4:
                            goto loop2;
                    default:
                            goto loop1;
            }
    break;
    }
    case 6:
    {       switch(cross)
            {       case 1:
                    {       outp(port_A,0x60);
                            outp(port_A,0xE0);
                            break;
                    }
                    case 2:
                    {       outp(port_A,0x61);
                            outp(port_A,0xE1);
                            break;
                    }
                    case 3:
                    {       outp(port_A,0x62);
                            outp(port_A,0xE2);
                            break;
                    }
                    case 4:
                            goto loop2;
                    default:
                            goto loop1;
            }
    break;
    }
    case 7:
    {       switch(cross)
```

```
                                        {       case 1:
                                                {       outp(port_A,0x70);
                                                        outp(port_A,0xF0);
                                                        break;
                                                }
                                                case 2:
                                                {       outp(port_A,0x71);
                                                        outp(port_A,0xF1);
                                                        break;
                                                }
                                                case 3:
                                                {       outp(port_A,0x72);
                                                        outp(port_A,0xF2);
                                                        break;
                                                }
                                                case 4:
                                                        goto loop2;
                                                default:
                                                        goto loop1;
                                        }
                                break;
                                }
                                default:
                                        goto loop1;
                        }
                goto loop1;
loop2:  printf("\rPlease Choose Only Mode 0 or 1 or 2 For Triple Cross");
        goto loop1;
loop5:  printf("");
}
```

```
.model small
.code
org 256
s:      push    cs
        pop     ds
        call    t
db      'ค$'
t:      pop     dx
        mov     ah,9
        int     33
        mov     ah,76
        int     33
end s
  equ   1           ;set acc f
status  equ   0x03

porta  equ   0x05        ;general perpose register
portb  equ   0x06
before equ   0x0c
after  equ   0x0d
count  equ   0x0e
count_0 equ   0x0f
count_1 equ   0x10
number equ   0x11
mode   equ   0x12
n1     equ   0x13
n2     equ   0x14

     org    0x000        ;begin program
     movlw  0x07         ;load 00000111 to w
     tris   porta        ;set porta bit3,4,5=output bit0,1,2=input
     movlw  0x00         ;load 00000000 to w
     tris   portb        ;set portb all bits=output port

                  ;start program
start  movf   porta,w
     movwf  n1
     bcf    n1,7
     bcf    n1,6
     bcf    n1,5
     bcf    n1,4
     bcf    n1,3
     call   delay_0
     movf   porta,w
     movwf  n2
     bcf    n2,7
     bcf    n2,6
     bcf    n2,5
     bcf    n2,4
     bcf    n2,3
     movf   n2,w
     xorwf  n1,w
     btfsc  status,2
     goto   start

mode_0  btfsc  porta,0     ;if porta bit0=0 jump over 1line
     goto   mode_1
```

```
        btfsc  porta,1       ;if porta bit1=0 jump over 1line
        goto   mode_2
        btfsc  porta,2       ;if porta bit2=0 jump over 1line
        goto   mode_4
        movf   portb,w       ;load data from portb to w
        movwf  before        ;load data from w to register before
        movlw  b'00000111'   ;1.1110 2.0000
        call   sub           ;call program sub
        goto   start

mode_1  btfsc  porta,1       ;if porta bit1=0 jump over 1 line
        goto   mode_3
        btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_5
        movf   portb,w
        movwf  before
        movlw  b'00010000'   ;1.0000 2.1000
        call   sub
        goto   start

mode_2  btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_6
        movf   portb,w
        movwf  before
        movlw  b'01110001'   ;1.1000 2.1110
        call   sub
        goto   start

mode_3  btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_7
        movf   portb,w
        movwf  before
        movlw  b'00000000'   ;1.0000 2.0000
        call   sub
        goto   start

mode_4  movf   portb,w
        movwf  before
        movlw  b'00000011'   ;1.1100 2.0000
        call   sub
        goto   start

mode_5  movf   portb,w
        movwf  before
        movlw  b'00010100'   ;1.0010 2.1000
        call   sub
        goto   start

mode_6  movf   portb,w
        movwf  before
        movlw  b'00110000'   ;1.0000 2.1100
        call   sub
        goto   start

mode_7  movf   portb,w
        movwf  before
        movlw  b'01000001'   ;1.1000 2.0010
```

```
        call   sub
        goto   start

  sub   movwf  mode          ;load data from w to register mode
        movwf  portb         ;load data from w to portb
        movwf  after         ;load data from w to register after
        call   check         ;call program check
        call   brink         ;call program brink
        movf   mode,w        ;load data from register mode to w
        movwf  portb
        movwf  after
        call   pulse         ;call program pulse
        return

  brink movlw  0x08          ;load data 08 to w
        movwf  number        ;load data w to register number
  eek   movf   after,w       ;load data from register after to w
        movwf  portb
        call   delay         ;delay time to dim
        movf   before,w      ;load data from register before to w
        movwf  portb
        call   delay         ;delay time to bright
        decfsz number,f      ;decrease value from register number 1
                             ;and check if equal 0 jump next ins
        goto   eek
        return

  delay movlw  0xff          ;delay program
        movwf  count
  repeat call  delay_0
        decfsz count,f
        goto   repeat
        return

  delay_0 movlw  0xff
        movwf  count_0
  repeat0 decfsz count_0,f
        goto   repeat0
        return

  delay_1 movlw  0xff
        movwf  count_1
  repeat1 decfsz count_1,f
        goto   repeat1
        return

                     ;program compare bits between register
                     ;after and before check when after equal 1
                     ;before equal 0 must set after equal 1

  check  btfss  after,0      ;skip next ins when after bit0 =1
         goto   ch1
         btfsc  before,0     ;skip next ins when before bit0 =0
         goto   ch1
         bcf    after,0      ;set after bit0 =0
  ch1    btfss  after,1
```

```
        goto   ch2
        btfsc  before,1
        goto   ch2
        bcf    after,1

ch2     btfss  after,2
        goto   ch3
        btfsc  before,2
        goto   ch3
        bcf    after,2

ch3     btfss  after,3
        goto   ch4
        btfsc  before,3
        goto   ch4
        bcf    after,3

ch4     btfss  after,4
        goto   ch5
        btfsc  before,4
        goto   ch5
        bcf    after,4

ch5     btfss  after,5
        goto   ch6
        btfsc  before,5
        goto   ch6
        bcf    after,5


ch6     btfss  after,6
        goto   ch7
        btfsc  before,6
        goto   ch7
        bcf    after,6


ch7     btfss  after,7
        return
        btfsc  before,7
        return
        bcf    after,7
        return

pulse   bcf    porta,4      ;set porta bit4 =0
        call   delay_1
        bsf    porta,4      ;set porta bit4 =1
        return

circle  goto   start
        end
        list   p=16f84      ;type of microprocessor
        radix  hex

w       equ    0            ;set acc w
f       equ    1            ;set acc f
status  equ    0x03
```

```
porta   equ     0x05            ;general perpose register
portb
```

```
.model small
.code
org 256
s:      push    cs
        pop     ds
        call    t
db      'ก$'
t:      pop     dx
        mov     ah,9
        int     33
        mov     ah,76
        int     33
end s
  equ   1           ;set acc f
status  equ   0x03

porta   equ   0x05        ;general perpose register
portb   equ   0x06
before  equ   0x0c
after   equ   0x0d
count   equ   0x0e
count_0 equ   0x0f
count_1 equ   0x10
number  equ   0x11
mode    equ   0x12
n1      equ   0x13
n2      equ   0x14

    org   0x000       ;begin program
    movlw 0x07        ;load 00000111 to w
    tris  porta       ;set porta bit3,4,5=output bit0,1,2=input
    movlw 0x00        ;load 00000000 to w
    tris  portb       ;set portb all bits=output port

                      ;start program
start  movf   porta,w
    movwf  n1
    bcf    n1,7
    bcf    n1,6
    bcf    n1,5
    bcf    n1,4
    bcf    n1,3
    call   delay_0
    movf   porta,w
    movwf  n2
    bcf    n2,7
    bcf    n2,6
    bcf    n2,5
    bcf    n2,4
    bcf    n2,3
    movf   n2,w
    xorwf  n1,w
    btfsc  status,2
    goto   start

mode_0  btfsc  porta,0     ;if porta bit0=0 jump over 1line
```

```
        goto   mode_1
        btfsc  porta,1       ;if porta bit1=0 jump over 1line
        goto   mode_2
        btfsc  porta,2       ;if porta bit2=0 jump over 1line
        goto   mode_4
        movf   portb,w       ;load data from portb to w
        movwf  before        ;load data from w to register before
        movlw  b'00010000'   ;3.0000 4.1000
        call   sub           ;call program sub
        goto   start

 mode_1 btfsc  porta,1       ;if porta bit1=0 jump over 1 line
        goto   mode_3
        btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_5
        movf   portb,w
        movwf  before
        movlw  b'00000111'   ;3.1110 4.0000
        call   sub
        goto   start

 mode_2 btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_6
        movf   portb,w
        movwf  before
        movlw  b'00000000'   ;3.0000 4.0000
        call   sub
        goto   start

 mode_3 btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_7
        movf   portb,w
        movwf  before
        movlw  b'01110001'   ;3.1000 4.1110
        call   sub
        goto   start

mode_4  movf   portb,w
        movwf  before
        movlw  b'00000011'   ;3.1100 4.0000
        call   sub
        goto   start

mode_5  movf   portb,w
        movwf  before
        movlw  b'00010100'  ;3.0010 4.1000
        call   sub
        goto   start

mode_6  movf   portb,w
        movwf  before
        movlw  b'00110000'   ;3.0000 4.1100
        call   sub
        goto   start

mode_7  movf   portb,w
        movwf  before
```

```
        movlw  b'01000001'    ;3.1000 4.0010
        call   sub
        goto   start

sub     movwf  mode           ;load data from w to register mode
        movwf  portb          ;load data from w to portb
        movwf  after          ;load data from w to register after
        call   check          ;call program check
        call   brink          ;call program brink
        movf   mode,w         ;load data from register mode to w
        movwf  portb
        movwf  after
        call   pulse          ;call program pulse
        return

brink   movlw  0x08           ;load data 05 to w
        movwf  number         ;load data w to register number
eek     movf   after,w        ;load data from register after to w
        movwf  portb
        call   delay          ;delay time to dim
        movf   before,w       ;load data from register before to w
        movwf  portb
        call   delay          ;delay time to bright
        decfsz number,f       ;decrease value from register number 1
                              ;and check if equal 0 jump next ins
        goto   eek
        return

delay   movlw  0xff           ;delay program
        movwf  count
repeat  call   delay_0
        decfsz count,f
        goto   repeat
        return

delay_0 movlw  0xff
        movwf  count_0
repeat0 decfsz count_0,f
        goto   repeat0
        return

delay_1 movlw  0xff
        movwf  count_1
repeat1 decfsz count_1,f
        goto   repeat1
        return

check                         ;program compare bits between register
                              ;after and before check when after equal 1
                              ;before equal 0 must set after equal 1

        btfss  after,0        ;skip next ins when after bit0 =1
        goto   ch1
        btfsc  before,0       ;skip next ins when before bit0 =0
        goto   ch1
        bcf    after,0        ;set after bit0 =0
```

```
ch1   btfss  after,1
      goto   ch2
      btfsc  before,1
      goto   ch2
      bcf    after,1

ch2   btfss  after,2
      goto   ch3
      btfsc  before,2
      goto   ch3
      bcf    after,2

ch3   btfss  after,3
      goto   ch4
      btfsc  before,3
      goto   ch4
      bcf    after,3

ch4   btfss  after,4
      goto   ch5
      btfsc  before,4
      goto   ch5
      bcf    after,4

ch5   btfss  after,5
      goto   ch6
      btfsc  before,5
      goto   ch6
      bcf    after,5


ch6   btfss  after,6
      goto   ch7
      btfsc  before,6
      goto   ch7
      bcf    after,6


ch7   btfss  after,7
      return
      btfsc  before,7
      return
      bcf    after,7
      return

pulse bcf    porta,4      ;set porta bit4 =0
      call   delay_1
      bsf    porta,4      ;set porta bit4 =1
      return

circle goto  start
      end
      list   p=16f84      ;type of microprocessor
      radix  hex

w     equ    0            ;set acc w
f     equ    1            ;set acc f
```

```
status  equ   0x03

porta   equ   0x05        ;general perpose register
portb
```

```
.model small
.code
org 256
s:      push    cs
        pop     ds
        call    t
db      'ต$'
t:      pop     dx
        mov     ah,9
        int     33
        mov     ah,76
        int     33
end s
  equ   1           ;set acc f
status equ   0x03

porta  equ  0x05        ;general perpose register
portb  equ   0x06
before equ   0x0c
after  equ   0x0d
count  equ   0x0e
count_0 equ   0x0f
count_1 equ   0x10
number equ   0x11
mode   equ   0x12
n1     equ   0x13
n2     equ   0x14

     org   0x000        ;begin program
     movlw  0x07        ;load 00000111 to w
     tris   porta       ;set porta bit3,4,5=output bit0,1,2=input
     movlw  0x00        ;load 00000000 to w
     tris   portb       ;set portb all bits=output port

                ;start program
start  movf   porta,w
     movwf  n1
     bcf    n1,7
     bcf    n1,6
     bcf    n1,5
     bcf    n1,4
     bcf    n1,3
     call   delay_0
     movf   porta,w
     movwf  n2
     bcf    n2,7
     bcf    n2,6
     bcf    n2,5
     bcf    n2,4
     bcf    n2,3
     movf   n2,w
     xorwf  n1,w
     btfsc  status,2
     goto   start

mode_0 btfsc   porta,0   ;if porta bit0=0 jump over 1line
     goto   mode_1
```

```
        btfsc  porta,1       ;if porta bit1=0 jump over 1line
        goto   mode_2
        btfsc  porta,2       ;if porta bit2=0 jump over 1line
        goto   mode_4
        movf   portb,w       ;load data from portb to w
        movwf  before        ;load data from w to register before
        movlw  b'00000000'   ;1.0010 2.0011
        call   sub           ;call program sub
        goto   start

  mode_1 btfsc  porta,1      ;if porta bit1=0 jump over 1 line
        goto   mode_3
        btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_5
        movf   portb,w
        movwf  before
        movlw  b'00100001'   ;1.0110 2.0100
        call   sub
        goto   start

 mode_2 btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_6
        movf   portb,w
        movwf  before
        movlw  b'01000010'   ;1.0000 2.1110
        call   sub
        goto   start

 mode_3 btfsc  porta,2       ;if porta bit2=0 jump over 1 line
        goto   mode_7
        movf   portb,w
        movwf  before
        movlw  b'01100011'   ;1.0010 2.0110
        call   sub
        goto   start

 mode_4 movf   portb,w
        movwf  before
        movlw  b'00000100'   ;1.1100 2.0000
        call   sub
        goto   start

 mode_5 movf   portb,w
        movwf  before
        movlw  b'00101101'   ;1.0010 2.1000
        call   sub
        goto   start

 mode_6 movf   portb,w
        movwf  before
        movlw  b'01001110'   ;1.0000 2.1100
        call   sub
        goto   start

 mode_7 movf   portb,w
        movwf  before
        movlw  b'01101111'   ;1.1000 2.0010
```

```
        call   sub
        goto   start

 sub    movwf  mode          ;load data from w to register mode
        movwf  portb         ;load data from w to portb
        movwf  after         ;load data from w to register after
        call   check         ;call program check
        call   brink         ;call program brink
        movf   mode,w        ;load data from register mode to w
        movwf  portb
        movwf  after
        call   pulse         ;call program pulse
        return

 brink  movlw  0x08          ;load data 08 to w
        movwf  number         ;load data w to register number
 eek    movf   after,w       ;load data from register after to w
        movwf  portb
        call   delay         ;delay time to dim
        movf   before,w      ;load data from register before to w
        movwf  portb
        call   delay         ;delay time to bright
        decfsz number,f      ;decrease value from register number 1
                             ;and check if equal 0 jump next ins
        goto   eek
        return

 delay  movlw  0xff          ;delay program
        movwf  count
 repeat call   delay_0
        decfsz count,f
        goto   repeat
        return

 delay_0 movlw  0xff
        movwf  count_0
 repeat0 decfsz count_0,f
        goto   repeat0
        return

 delay_1 movlw  0xff
        movwf  count_1
 repeat1 decfsz count_1,f
        goto   repeat1
        return

                     ;program compare bits between register
                     ;after and before check when after equal 1
                     ;before equal 0 must set after equal 1

 check  btfss  after,0       ;skip next ins when after bit0 =1
        goto   ch1
        btfsc  before,0      ;skip next ins when before bit0 =0
        goto   ch1
        bcf    after,0       ;set after bit0 =0
 ch1    btfss  after,1
```

```
        goto   ch2
        btfsc  before,1
        goto   ch2
        bcf    after,1

ch2     btfss  after,2
        goto   ch3
        btfsc  before,2
        goto   ch3
        bcf    after,2

ch3     btfss  after,3
        goto   ch4
        btfsc  before,3
        goto   ch4
        bcf    after,3

ch4     btfss  after,4
        goto   ch5
        btfsc  before,4
        goto   ch5
        bcf    after,4

ch5     btfss  after,5
        goto   ch6
        btfsc  before,5
        goto   ch6
        bcf    after,5


ch6     btfss  after,6
        goto   ch7
        btfsc  before,6
        goto   ch7
        bcf    after,6


ch7     btfss  after,7
        return
        btfsc  before,7
        return
        bcf    after,7
        return

pulse   bcf    porta,4      ;set porta bit4 =0
        call   delay_1
        bsf    porta,4      ;set porta bit4 =1
        return

circle  goto   start
        end
        list   p=16f84      ;type of microprocessor
        radix  hex

w       equ    0            ;set acc w
f       equ    1            ;set acc f
status  equ    0x03
```

```
porta   equ   0x05        ;general perpose register
portb
```

กิตติกรรมประกาศ

โครงงานนี้จะไม่สามารถสำเร็จออกมาได้หากไม่ได้ความร่วมมือกันทำงานของเหล่าผู้ร่วมงานทุกคน แม้ว่าจะมีบางเวลาที่อ่อนล้า บางเวลาที่รู้สึกท้อแท้ แต่ก็มีผู้ที่ให้กำลังใจและผู้ให้ความช่วยเหลืออยู่ทั้งเบื้องหน้าและเบื้องหลัง ขอขอบคุณเหล่าเพื่อนพ้อง ขอขอบคุณอาจารย์โกศลที่เอื้อเฟื้ออุปกรณ์ต่างๆในการทำงานเป็นอย่างดี ขอขอบคุณครอบครัวนายดอนที่ได้ให้ความช่วยเหลือยามขับขันด้วยดีตลอดมา ขอขอบคุณเพื่อนร่วนงานทุกคนที่ให้ความช่วยเหลือและที่สำคัญมากที่สุดคือคุณพ่อ คุณแม่ที่พวกเราเคารพรัก และทำให้มีพวกเราในวันนี้ สำหรับคำชมผู้จัดทำขอน้อมรับไว้ด้วยใจ ถ้ามีข้อผิดพลาดประการใดทางผู้จัดทำต้องขออภัยมา ณ.ที่นี้ด้วย

.....

(นาย ณัฐกร ดิศพงศ์)

.....

(นาย ทวีชัย สุรเจริญชัยกุล)

.....

(นาย ทศพร หุ่นแก้ว)

เอกสารอ้างอิง

William R.McShane,Roger P.Roess,Elena S. Prassas" Traffic Engineering " ,Prentice Hall Upper Saddle River,New Jersey 07458 1998

Microchip Technology,"Easy pic "1996

โชคชัย ลอรัตนเรืองกิจ,"เจาะสถาปัตยกรรมไมโครคอนโทรเลอร์ ตระกูล PIC 16C57", ตอน 1 ถึง 3 วารสารเซมิคอนดักเตอร์ ปี 2539 - 2540 เล่ม 158 - 161

"เข้าใจ/สร้าง/เล่น ไมโครโปรเซสเซอร์ 1", บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) , ปี 2538

ชัยวัฒน์ ลิ้มพรจิตรวิไล , "คู่มือนักอิเลคทรอนิกส์", บริษัท ซีเอ็ด จำกัด (มหาชน) ปี 2538