ชุดจำลอง พีแอลซี

นายนิวัฒน์ โทนลักษณ์ 39013276

นายนุสนธิ์ ศรีปราชญ์ 39013277

นายประหยัด โชติภัทรกุล 39013280

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

PLC EMULATOR

Mr . Niwat Thonlusana 39013276

Mr . Nuson Siprat 39013277

Mr . Prayad Chotipattarakool 39013280

Project Report Submitted in Partial Fulfillment of the Requirement

for the Bachelor 's Degree

Department of Industrial Technology

Faculty of Engineering

King Mongkut 's Institute of Technology Ladkrabang

1998

หัวข้อปริญญานิพนธ์	ชุดทำลอง พีแอลซี
ชื่อนักศึกษา	นายนิวัฒน์ โทนล้กษณ์
	นายนุสนธิ์ ศรีปราชญ์
	นายประหยัด โชติภัทรกุล
อาจารย์ที่ปรึกษา	ดร.ปีติเขต สู้รักษา
	อ.บุญชนะ ภู่ระหงษ์
ภาควิชา	เทคนิคอุตสาหกรรม
ปีการศึกษา	2541

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์ฉบับนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

..... หัวหน้าภาควิชาเทคนิคอุตสาหกรรม
( )

คณะกรรมการสอบ

..... ประธานคณะกรรมการ
( )

..... กรรมการ
( )

..... กรรมการ
( )

..... กรรมการ
( )

Project Report	PLC EMULATOR
By	Mr . Niwat Thonlusana
	Mr . Nuson Siprat
	Mr . Prayad Chotipattarakool
Department	Industrial Technology
Project Report Advisor	Dr . Pitikhate Sooraksa
	Mr . Bunchana Purahong

Accepted by the Faculty of Engineering , King Mongkut 's Institute of Technology , Ladgrabang in partial fulfillment of the requirements for the bachelor 's degree

Pitikhate Sooraksa

..... Chairman

( )

Project Report Committee

..... Member

( )

..... Member

( )

..... Member

( )

..... Member

( )

หัวข้อปริญญานิพนธ์	ชุดจำลอง พีแอลซี
ชื่อนักศึกษา	นายนิวัฒน์ โทนลักษณ์
	นายนุสนธิ์ ศรีปราชญ์
	นายประหยัด โชติภัทรกุล
อาจารย์ที่ปรึกษา	ดร.ปิติเขต สู้รักษา
	อ.บุญชนะ ภู่ระหงษ์
ภาควิชา	เทคนิคอุตสาหกรรม
ปีการศึกษา	2541

บทคัดย่อ

จุดมุ่งหมายของปริญญานิพนธ์นี้เพื่อศึกษาการทำงาน ทั้งด้านฮาร์ดแวร์ และซอร์ฟแวร์ของ ไมโครคอนโทรลเลอร์ AT89C52 ในการใช้งานจริง รวมทั้งการใช้งานขั้นพื้นฐานของ พีแอลซี เพื่อนำความรู้ไปใช้งานต่อไป ปริญญานิพนธ์นี้กล่าวถึงทฤษฎี การทำงานของไมโครคอนโทรลเลอร์ รวมทั้งการเชื่อมต่อกับอุปกรณ์พื้นฐานรวมทั้งลักษณะการทำงานของ พีแอลซี และแนวความคิดที่ได้ใช้ในการจำลอง ไมโครคอนโทรลเลอร์ AT89C52 เป็น พีแอลซี ผลที่ได้จากปริญญานิพนธ์นี้สามารถนำไปใช้เพื่อเริ่มการพัฒนาอุตสาหกรรม พีแอลซี เพื่อการผลิตได้เองในประเทศ

Project Report Title	PLC EMULATOR
Name	Mr . Niwat Thonlusana
	Mr . Nuson Siprat
	Mr . Prayad Chotipattarakool
Project Report Advisor	Dr . Pitikhate Sooraksa
	Mr . Bunchana Purahong
Department	Industrial Technology
Academic Year	1998

Abstract

The objective of this project is to study hardware and software of the AT89C52 microcontroller for basic knowledge and real applications . The thesis presents theory and architecture of MCS-51 family . The emulation of MCS-51 as a simple PLC is implemented . The emulator could be applied to initiate the PLC industry for domestic markets .

กิตติกรรมประกาศ

เนื่องจากชุดจำลองพีแอลซี นี้ประกอบด้วยส่วนสำคัญ 2 ส่วนคือฮาร์ดแวร์ และซอฟท์แวร์ จึงจำเป็นต้องใช้ความรู้และการค้นคว้าอย่างมาก ซึ่งความซับซ้อนทางกลวิธีทางการโปรแกรมนั้นมากเป็นพิเศษ เพราะเทียบกับการสร้างตัวแปลภาษาขึ้นมาอีกภาษาหนึ่ง ซึ่งคณะผู้จัดทำมีความถนัดในด้านนี้น้อยมาก แต่หากได้รับความอนุเคราะห์ด้านคำปรึกษาชี้แนะในแนวทางที่ถูกต้องของอาจารย์ผู้มีพระคุณ ซึ่งนอกจากจะได้รับการสั่งสอน ถ่ายทอดความรู้แล้ว ยังได้รับความห่วงใยต่อคณะผู้จัดทำฉันท์ศิษย์และอาจารย์อีกด้วย

ฉะนั้น ในโอกาสอันเหมาะสมนี้ คณะผู้จัดทำขอได้กล่าวคำขอบพระคุณในความรู้ความห่วงใย และปรารถนาดี ที่คณะผู้จัดทำได้รับ จาก ท่านอาจารย์ ดร. ปิติเขต สู้รักษา และท่านอาจารย์ บุญชนะ ภู่ระหงษ์ มา ณ. ที่นี้ด้วยความสำนึกในพระคุณอย่างสูง โดยส่วนดีของปริญญานิพนธ์นี้ที่อาจได้มีโอกาสเผยแพร่ ไปยังผู้แสวงหาความรู้ในด้านนี้ คณะผู้จัดทำขอมอบให้เป็นเกียรติต่อ ท่านอาจารย์ทั้งสองท่าน และคณะผู้จัดทำยังขอกล่าวคำขอบคุณอย่างจริงใจต่อบุคคลที่มีส่วนช่วยเหลือต่อผลงานชิ้นนี้ และขออภัยที่มิได้เอ่ยนาม ณ. ที่นี้ ส่วนข้อผิดพลาดหรือแนวทางที่ผิดนั้น คณะผู้จัดทำขอน้อมรับไว้เพื่อการแก้ไขต่อไปในอนาคต

คณะผู้จัดทำ

20 ตุลาคม 2541

สารบัญ

หน้า

บทที่ 1

บทนำ

ปัจจุบันวิวัฒนาการของโลกก้าวไกลขึ้น ยุคต้นของงานอุตสาหกรรม มีการใช้แรงงานคนจำนวนมากในงานผลิต แต่ต่อมาเมื่อมีไฟฟ้าเกิดขึ้นมนุษย์รู้จักคิดค้นประดิษฐ์กรรม ที่นำมาใช้ร่วมกับไฟฟ้า พัฒนาตั้งแต่ระบบการทำงานด้วยมือ (Manual) มาเป็นระบบการทำงานด้วยรีเลย์ (Relay) จนกระทั่งมีอุปกรณ์อิเล็กทรอนิกส์ที่มีขนาดเล็ก และความสามารถสูงเข้ามาเกี่ยวข้องกับงานอุตสาหกรรมมากขึ้น และมีปริมาณเพิ่มขึ้นเรื่อย ๆ เกิดเป็น พีซี (พีซี : Programable Controller) ซึ่งเป็นต้นแบบของ พีแอลซี (Programable Logic Controller) ในปัจจุบัน พีแอลซี คือ อุปกรณ์กึ่งอิเล็กทรอนิกส์ไฟฟ้าแบบใหม่ที่ใช้แทนรีเลย์ และใช้การเขียนชุดคำสั่งทำนองเดียวกันกับคอมพิวเตอร์มีประโยชน์การใช้งานมากมาย เพราะเหตุผล ดังต่อไปนี้

1. ใช้ในระบบการควบคุมที่สามารถแก้ไขดัดแปลงได้ง่าย
2. ระบบ พีแอลซี มีความน่าเชื่อถือสูง
3. ใช้ในเนื้อที่จำกัดได้
4. สามารถขยายจำนวน อินพุต และเอ้าท์พุต ในอนาคตได้
5. มีการเก็บรวบรวมข้อมูลการผลิตในตัวมันเอง
6. การแก้ไขลักษณะการควบคุมบ่อยครั้ง และสามารถใช้เวลาน้อยที่สุด
7. การควบคุมมีลักษณะคล้ายกับการใช้เครื่องจักรหลาย ๆ เครื่องพร้อมกัน
8. ระบบการควบคุมมีการขยายตัวในอนาคต

ลักษณะการใช้งานไม่ยุ่งยากเหมือนคอมพิวเตอร์ สะดวกต่อการบำรุงรักษาเป็นเวลานาน ชุดคำสั่งพีแอลซี สามารถพัฒนาให้มีความสามารถในการตัดสินใจสูงมากขึ้นเรื่อย ๆ เป็นที่มาของการสร้าง " ชุดจำลอง พี-แอล-ซี " ซึ่งชุดจำลอง พีแอลซี นั้นเป็นวงจรที่ใช้ ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นตัวประมวลผล มาจำลองเป็นพีแอลซี เพื่อให้ได้คุณลักษณะของการชุดคำสั่งที่ง่าย โดยผู้นำไปใช้จะต้องมีความรู้พื้นฐานการใช้งาน พีแอลซี และ แผนผังขั้นบันได (Ladder Diagram)

1.1 วัตถุประสงค์

1. เพื่อศึกษาการทำงานของ พีแอลซี (PLC) และประยุกต์ใช้งานได้
2. เพื่อสร้างชุดควบคุมทางอิเล็กทรอนิกส์ ที่สามารถนำไปใช้ควบคุมกระบวนการทางอุตสาหกรรมเบา
3. เพื่อศึกษา พีแอลซี การทำงานของ ไมโครคอนโทรลเลอร์ AT89C52 และประยุกต์

สร้างวงจรควบคุมร่วมกับ Chip Support

1.2 ขอบเขตของโครงงาน

1. ออกแบบวงจรจำลอง พีแอลซี
2. สร้างโปรแกรมสื่อสารระหว่างวงจรที่สร้างขึ้น กับเครื่องคอมพิวเตอร์ส่วนบุคคล ให้กับวงจร
3. สร้างโปรแกรมแปลภาษาบูลลีนขั้นพื้นฐานให้เป็นภาษาแอสเซมบลีของไมโครคอนโทรลเลอร์ตระกูล MCS-51

1.3 แนวโน้มในการประยุกต์ใช้งานอื่น

จากการศึกษาวงจรควบคุมการทำงานของอุปกรณ์ และ เครื่องจักรไฟฟ้าโดยทั่วไป ที่เป็นระบบอัตโนมัติ ส่วนใหญ่มีเงื่อนไขในการเลือกกระบวนการทำงานตามลักษณะเงื่อนไขทางตรรกะ จึงเป็นไปได้ว่า ชุดจำลองการทำงาน พีแอลซี นี้จะสามารถนำไปใช้งานได้ตามคุณสมบัติพื้นฐานของ พีแอลซี

บทที่ 2

ทฤษฎี พีแอลซี

2.1 ความหมายของ พีแอลซี

PROGRAMMABLE LOGIC CONTROLLER หรือ " พีแอลซี " คือ อุปกรณ์ที่ใช้ควบคุมการทำงานของเครื่องจักร อุปกรณ์ไฟฟ้า หรือระบบกระบวนการต่าง ๆ ที่มีลักษณะการทำงาน เป็นแบบลอจิก หรือแบบซีเควินซ์ คือ เป็น 0 กับ 1 นั้นเอง แต่พบว่าในท้องตลาด อาจใช้คำว่า ' พีซี ' แทน ซึ่งจริง ๆ แล้ว พีซี ก็คือ PROGRAMMABLE CONTROLLER ซึ่งเป็นอุปกรณ์ที่ใช้ในงานควบคุมเหมือนกับ พีแอลซี แต่ พีซี จะรวมเอาการควบคุมที่มี สัญญาณเป็นแบบอนาล็อก รวมถึงการควบคุมแบบ พีไอดี (PID) เข้าไปด้วย จึงสรุปได้ว่า พีแอลซี เป็นส่วนหนึ่งของ พีซี นั้นเอง

2.2 การทำงานของ พีแอลซี

การทำงานของ พีแอลซี จะเป็นควบการทำงานของอุปกรณ์อิเล็กทรอนิกส์ โดยมี ไมโครโปรเซสเซอร์ เป็นหน่วยประมวลผล ซึ่งจะทำหน้าที่รับค่าจากภายนอกเข้ามาทำการประมวลผล เมื่อได้ผลลัพธ์แล้วก็จะส่งออกสู่ภายนอก เพื่อใช้ควบคุมอุปกรณ์ที่เราต้องการ หลังจากนั้นก็จะกลับมารับค่าสภาวะจากภายนอกใหม่ ซึ่งลักษณะการทำงานแบบนี้จะเป็นการวนรอบของชุดคำสั่งที่เรียกว่า " การสแกน " โดยเวลาการสแกนจะเริ่มตั้งแต่การรับค่าสภาวะเข้ามาทำการประมวลผล และส่งผลลัพธ์ออกไป

2.3 การใช้งานของ พีแอลซี

การที่เราจะทำการควบคุม พีแอลซี ให้ทำงานตามความต้องการได้นั้น ก็จะต้องมีภาษาหรือคำสั่งที่ใช้ในการเขียนชุดคำสั่งควบคุมเครื่อง พีแอลซี ซึ่งภาษา หรือคำสั่งที่ใช้มีอยู่หลายภาษาด้วยกัน เช่น ภาษาแลดเดอร์ , ภาษาบูลลีน , ภาษาบล็อก , คำสั่งข้อความภาษาอังกฤษ , ภาษาฟังก์ชั่นชาร์ท แต่ภาษาที่ใช้งานได้ง่าย และเป็นที่นิยมกันมากที่สุด ก็คือ ภาษาแลดเดอร์ และภาษาบูลลีน ซึ่งวิธีการใช้ภาษาก็คือ ทำการเขียนภาษาแลดเดอร์ขึ้นมาก่อน หลังจากนั้นจึงค่อยแปลงจากภาษาแลดเดอร์มาเป็นภาษาบูลลีนอีกที เพื่อใช้ทำการคีย์แล้วป้อนเข้าเครื่อง พีแอลซี

2.4 ภาษาแลดเดอร์

ภาษาแลดเดอร์หรือที่นิยมเรียกกันว่า แผนผังขั้นบันได จะประกอบด้วย สัญลักษณ์หน้าสัมผัส มีลักษณะคล้ายวงจรรีเลย์ ซึ่งการเขียนชุดคำสั่งต้องระบุตำแหน่ง หรือหมายเลขของอุปกรณ์ต่าง ๆ ให้ ถูกต้องภาษาแลดเดอร์จะมีลักษณะคล้ายขั้นบันได ที่มีการอ่าน หรือเขียนจากบนลงมาล่าง ซึ่งโดยทั่ว ๆ ไปแล้วภาษาแลดเดอร์ของ พีแอลซี แต่ละเครื่อง อาจแตกต่างกันบ้างเล็กน้อย

2.5 ภาษาบูลลีน

ภาษาบูลลีน เป็นภาษาพื้นฐานของ พีแอลซี มีรูปแบบ หรือการสื่อความหมายที่เป็นตรรกะ ที่เข้าใจได้ง่าย เช่น LD , OR , NOT , OUT เป็นต้น การเขียนคำสั่งในการควบคุมอุปกรณ์ต่าง ๆ ที่เขียนด้วยคำสั่งของไมโครโปรเซสเซอร์ ต้องศึกษาทำความเข้าใจถึงโครงสร้างคำสั่ง , สถาปัตยกรรมภายในของไมโครโปรเซสเซอร์เบอร์นั้น ๆ ฮาร์ดแวร์ (HARDWARE) ของระบบซึ่งจะมีความสลับซับซ้อนในการเขียนงานควบคุมในขั้นต้นมากพอสมควร แต่ แอลพีซี จะสั่งงานโดยวิธีการแบบตรรกะ หรือคำสั่งต่าง ๆ ที่ใช้จะมองในลักษณะ ไอซีเกต (IC Gate) ต่าง ๆ ที่มีอินพุตเข้า และจ่ายเอาต์พุตออกในลักษณะ 0 หรือ 1 เท่านั้น ทำให้ความซับซ้อนของการนำคำสั่งมาใช้งานน้อยกว่าแต่จะต้องทำความเข้าในการวางลำดับของหน้าสัมผัส ที่จะมาทำเงื่อนไขในการทำงานซึ่งกันและกันเพราะลำดับ และการทำงานของคำสั่งจะถูกทำงานไล่เรียงเป็นลำดับ จากบนลงล่างมาตลอด จนกว่าจะพบคำสั่งสิ้นสุดการทำงาน (END) ก็จะวนกลับขึ้นไปบรรทัดแรกใหม่เช่นนี้เรื่อยไป ซึ่งการทำงานตั้งแต่บรรทัดบนสุดจนถึงคำสั่ง END พีแอลซี จะใช้เวลานี้ไป ถือเป็น 1 รอบการทำงาน ซึ่งเรียกว่าช่วงเวลาการสแกน (SCAN TIME) ดังนั้นในการเขียนชุดคำสั่งต้องคำนึงถึงเวลาในการสแกน 1 รอบการทำงานนี้ด้วย เช่น ถ้าอินพุตมีการตรวจจับ มีการเปลี่ยนสภาวะทุก 10 ms ถ้าเราเขียนชุดคำสั่ง RUN จนถึงคำสั่ง END ใช้เวลาไป 20 ms ก็จะทำให้การตรวจจับสัญญาณเกิดการผิดพลาดขึ้น ในการเขียนคำสั่งของ พีแอลซี ส่วนใหญ่ จะเขียนเป็นแลดเดอร์ไดอะแกรม ซึ่งจะมีการรับค่าสภาวะของอินพุตเข้ามา ไม่ว่าจะเป็นกี่สภาวะก็ตาม ที่มีความสัมพันธ์กันทางตรรกะจะมีผลลัพธ์เพียงค่าสภาวะเดียวคือ ภาวะสุดท้ายเท่านั้น และค่าสภาวะนี้จะถูกส่งออกทางเอาต์พุต โดยที่ตัวเอาต์พุตอาจจะมีมากกว่าหนึ่งจุดก็ได้และเราจะเรียกแลดแดอร์ไดอะแกรมนั้นว่าเป็น 1 รังก์ (1 RUNG)

2.6 ส่วนประกอบของพีแอลซี

พีแอลซี ก็เป็นคอมพิวเตอร์เครื่องหนึ่ง ซึ่งประกอบด้วยส่วนสำคัญ 3 ส่วนด้วยกัน

1. หน่วยประมวลผล (Processing Unit)
2. หน่วยความจำ (Memory Unit)
3. หน่วยอินพุต / เอาต์พุต (Input Output Unit)

1.หน่วยประมวลผล มีหน้าที่นำชุดคำสั่งผู้ใช้ (USER PROGRAM) มาปฏิบัติควบคุมการติดต่อรับส่งข้อมูลกับอุปกรณ์อินพุต เอาต์พุต และหน่วยความจำ

2. หน่วยความจำ จะแบ่งออกเป็น 2 ส่วนใหญ่ ๆ คือ

1. หน่วยความจำระบบ (SYSTEM MEMORY)
2. หน่วยความจำผู้ใช้ (USER MEMORY)

1. หน่วยความจำระบบ เป็นส่วนที่ใช้ในการเก็บชุดคำสั่งควบคุมการทำงานของ

เครื่อง พีแอลซี ในการติดต่อกับผู้ใช้ โดยหน่วยความจำในส่วนนี้ ผู้ใช้จะไม่สามารถเปลี่ยนแปลงแก้ไขได้ จึงอยู่ในรูปแบบของ ROM หรือ EPROM และยังมีส่วนของหน่วยความจำที่ใช้ในการเก็บสภาวะ หรือผลการทำงานที่เกิดจากการปฏิบัติของ ชุดคำสั่งบริหารระบบ (Monitor Program) การปฏิบัติงานจากชุดคำสั่งผู้ใช้ ซึ่งจะใช้หน่วยความจำแรม (RAM) เนื่องจากมีการเปลี่ยนแปลงอยู่ตลอดเวลา

2. หน่วยความจำผู้ใช้ เป็นส่วนที่ใช้เก็บชุดคำสั่งบูลลีนที่ผู้ใช้เขียนขึ้นทำการแปลแล้วถ่าย โอนมายังชุดจำลองการทำงานพีแอลซี เพื่อนำไปปฏิบัติงานตามเงื่อนไขต่าง ๆ ที่กำหนดไว้ ซึ่งจะเป็นหน่วยความจำแบบ แรม

3.1 หน่วยอินพุต ทำหน้าที่รับค่าทางไฟฟ้าของอุปกรณ์ภายนอก และเปลี่ยนสภาพทางตรรกะ เพื่อเก็บไว้ในหน่วยความจำระบบที่กำหนดเป็นส่วนอินพุต โดยจะมีค่าเป็น 1 เมื่ออุปกรณ์อินพุตอยู่ในสภาวะ ON หรือปิดวงจรไฟฟ้า และมีค่าเป็น 0 เมื่ออุปกรณ์อินพุตอยู่ใน สภาวะ OFF หรือเปิดวงจรไฟฟ้า

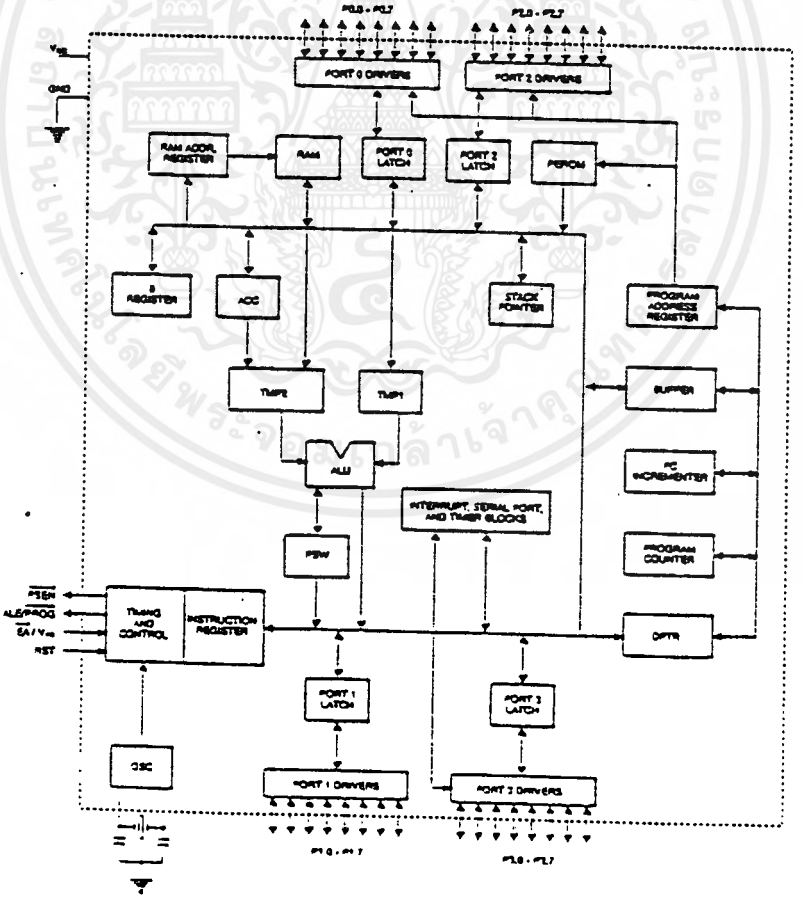
3.2 หน่วยเอาต์พุต จะรับสภาวะทางตรรกะ จากหน่วยความจำระบบที่กำหนดเป็นส่วนของเอาต์พุต แล้วเปลี่ยนเป็นค่าทางไฟฟ้า เพื่อควบคุมอุปกรณ์ภายนอกอีกทีหนึ่ง โดยค่า 1 หมายถึงการ ON หรือการต่อวงจรไฟฟ้าส่วนค่า 0 จะเท่ากับ OFF หรือการตัดวงจรไฟฟ้า

บทที่ 3

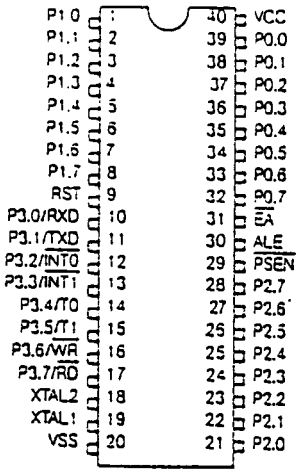
ทฤษฎีของไมโครคอนโทรลเลอร์ 89C52

3.1 สถาปัตยกรรมของ 89C52

89C52 ใช้สถาปัตยกรรมเดียวกันกับ 8052 แต่ภายในของ 89C52 จะเปลี่ยนแปลงจากบล็อกที่เป็น ROM (Read Only Memory) ของ 8052 เป็นบล็อก PE ROM ซึ่งมีขนาด 8 Kbyte ฉะนั้นเราจึงขอกล่าวว่า 89C52 เป็นตระกูลเดียวกันกับ 8052 จากรูปที่ 3.1 จะเป็นสถาปัตยกรรมภายในของ 89C52 ซึ่งจะอธิบายถึงส่วนย่อย ๆ ของภายใน 89C52 เพียงชิพเดียว และสัญญาณจากภายในจะต่อออกสู่ภายนอกทางขา (Pin) ของ 89C52 ที่มีอยู่ 40 ขา ดังรูปที่ 3.1



รูปที่ 3.1 สถาปัตยกรรมภายในของ 89C52



รูปที่ 3.2 ไดอะแกรมขาของ 89C52 แบบ DIP

89C52 เป็นไมโครคอนโทรลเลอร์ที่บรรจุอยู่ในวงจรรวมแบบ Dual Inline Package (DIP) ซึ่งแต่ละข้างของ 89C52 มีขาอยู่ข้างละ 20 ขา รวมกันทั้งหมด 2 ข้าง ได้ 40 ขา มีการใช้งานต่าง ๆ กัน ดังต่อไปนี้

VCC (ที่ขา 40)

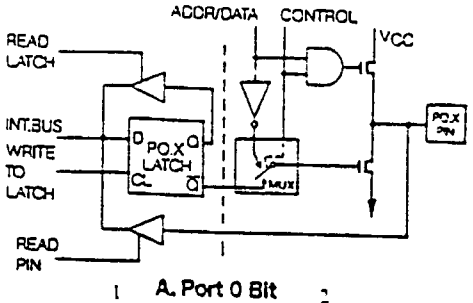
เป็นขาที่ต้องใช้ไฟเลี้ยง + 5 โวลท์ เข้าไปเพื่อให้วงจรรวมทำงานได้ ระดับโวลเตจของลอจิก 0 และ 1 ของ 89C52 จึงต่อเข้ากับอุปกรณ์ลอจิกแบบ TTL ได้โดยตรง

VSS (ที่ขา 20)

เป็นขาที่ต้องต่อกับกราวด์ (Ground)ของแหล่งจ่ายไฟ การต่ออุปกรณ์ทั้งหมดจะต้องมีกราวด์ของอุปกรณ์ต่อเข้าด้วยกัน

Port 0 (ที่ขา 32 - 39)

เป็นพอร์ทขนานขนาด 8 บิท อยู่ที่ขา 32 ถึง 32 เริ่มจากบิท 0 ถึงบิท 7 ตามลำดับ ดังในรูปที่ 3.2 แต่ละขาจะเขียนว่า P0.0 , P0.1 , ... , P0.7 P0.7 นั้น หมายถึงบิท 7 ของพอร์ท 0 ซึ่งเป็นบิทที่มีนัยสำคัญสูงสุด (Most Significant) และ P0.0 คือบิท 0 ของพอร์ท 0 เป็นบิทที่มีนัยสำคัญต่ำสุด (Least Significant) พอร์ท 0 นี้ใช้ได้ทั้งการรับ-ส่งตำแหน่งและข้อมูลกับหน่วยความจำ หรือใช้เป็นพอร์ทรับ-ส่งข้อมูลก็ได้ ข้อมูลที่ส่งออกทางพอร์ท 0 จะถูกแลตช์ (Latch) ไว้ที่ขาของพอร์ท โครงสร้างแต่ละบิทของพอร์ท 0 เป็น แบบ Open Drain Bidirectional ดังรูปที่ 3.3



รูปที่ 3.3 โครงสร้างของพอร์ท 0

ในรูปที่ 3.3 เมื่อเปรียบเทียบกับรูปที่ 3.1 ส่วนที่ 1 ของรูปที่ 3.3 ก็คือ Port 0 Latch ในรูปที่ 3.1 และส่วนที่ 2 ของรูปที่ 3.3 ก็คือ Port 0 Driver ของรูปที่ 3.1 นั่นเอง

จากโครงสร้างในรูปที่ 3.3 เมื่อมีคำสั่งของการเขียนข้อมูลมายังพอร์ท 0 ข้อมูลจาก Internal Data Bus จะถูก Latch ไว้ที่ D-FF โดยสัญญาณ " Write to Latch " ที่ถูกสร้างมาจากส่วน Timing and Control และในการอ่านข้อมูลจากพอร์ท 0 จะอ่านได้ 2 แบบคือ การอ่านข้อมูลที่ส่งไปเก็บไว้ที่พอร์ทก็จะมีสัญญาณ Read Latch มาเพื่ออ่านข้อมูลจาก D-FF กลับเข้าไปยัง Internal Data Bus การอ่านข้อมูลอีกแบบก็คือ การอ่านสภาวะของสัญญาณที่เข้ามาทางพอร์ท 0 ก็จะมีสัญญาณ Read Pin มาควบคุมการอ่าน พอร์ท 0 จะใช้งานหลายอย่างดังนี้

1. ใช้สำหรับส่งค่าตำแหน่ง หน่วยความจำภายนอกที่ต้องการติดต่อด้วย ตำแหน่งหน่วยความจำสูงสุดที่จะติดต่อได้ก็คือ 64 Kbyte จึงมีค่าตำแหน่งหน่วยความจำ 16 บิท ของเลขฐาน 2 ค่าตำแหน่งหน่วยความจำ 8 บิทล่างจะถูกส่งออกไปทางพอร์ท 0 และ 8 บิทบนจะส่งออกไปทางพอร์ท 2

2. ใช้รับ-ส่ง ข้อมูลกับ Data Memory หรือใช้รับข้อมูลจาก Program Memory

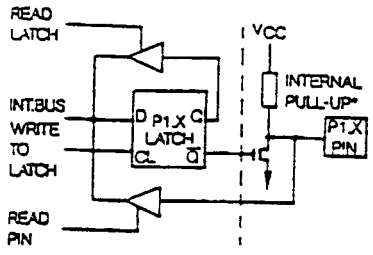
3. ใช้รับ-ส่ง ข้อมูลผ่านทางพอร์ทโดยตรง ในกรณีที่ไม่มีการใช้หน่วยความจำของ Program Memory หรือ Data Memory ภายนอก

วงจรภายในส่วน Timing and Control จะเป็นตัวสร้างสัญญาณมาควบคุมวงจรในรูปที่ 3.3 เพื่อให้การทำงานแต่ละอย่างข้างต้น เมื่อแต่ละบิทของพอร์ท 0 ทำงานตามข้อ 1 และ 2 ข้างต้น วงจร Timing and Control จะทำให้สภาวะลอจิกของขา Control เป็น 1 ซึ่งทำให้สวิทช์ MUX อยู่ในตำแหน่งข้างบน เมื่อพอร์ท 0 จะส่งข้อมูลซึ่งเป็นค่าตำแหน่งหน่วยความจำ หรือข้อมูลที่จะเขียนออกไปยังหน่วยความจำภายนอก ก็จะส่งค่าดังกล่าวมายัง ADDR/DATA ถ้าข้อมูลที่ส่งมาเป็น 1 จะทำให้สัญญาณออกจาก AND GATE เป็น 1 และสัญญาณที่ออกจาก Inverter เป็น 0 ดังนั้น FET ตัวบน ON (สภาวะ ON ของ FET คือความต้านทานระหว่างขา D กับ S มีค่าต่ำมากเสมือนกับเป็นวงจรปิด) ส่วน FET ตัวล่าง OFF (สภาวะ OFF ของ FET คือความต้านทานระหว่างขา D กับ S มีค่าสูงมากเสมือนกับเป็นวงจรเปิด) สภาวะลอจิกที่ขา P0.X

PIN จะเป็น 1 แต่ถ้าข้อมูลที่ส่งออกมายัง ADDR/DATA เป็น 0 ก็จะทำให้สัญญาณจาก AND GATE เป็น 0 และสัญญาณที่ออกจาก Inverter เป็น 1 ดังนั้น FET ตัวบนจะ OFF ส่วน FET ตัวล่างจะ ON ทำให้สภาวะลอจิกที่ขา P0.X PIN เป็น 0 เมื่อ 89C52 ต้องการใช้พอร์ท 0 สำหรับการอ่านข้อมูลจากหน่วยความจำภายนอก หรือใช้ทำงานในข้อ 3 ข้างบน ก็จะทำได้โดยวงจร Timing and Control ทำให้สภาวะลอจิกของสัญญาณ Control ในรูปเป็น 0 ทำให้เอาต์-พุตจาก AND GATE เป็น 0 FET ตัวบนจะ OFF และสวิตช์ MUX จะอยู่ในตำแหน่งข้างล่าง ดังนั้น FET ตัวล่างจะ ON หรือ OFF ก็แล้วแต่ข้อมูลที่ขา Q ของ D-FF เมื่อมีการเขียนข้อมูลจาก Internal Data Bus มายัง D-FF ก็จะมีสัญญาณ Write to Latch มายัง D-FF ด้วย ถ้าข้อมูลที่เขียนมาเป็น 1 ก็จะทำให้ขา Q มีสภาวะลอจิกเป็น 0 ทำให้ FET ตัวล่าง OFF ดังนั้นขา P0.X จะอยู่ในสภาวะอิมพีแดนซ์สูง (High Impedance) เพราะ FET ทั้ง 2 ตัว OFF แต่ถ้าข้อมูลที่เขียนมายัง D-FF เป็น 0 จะทำให้ FET ตัวล่าง ON แต่ตัวบน OFF ทำให้สภาวะลอจิกที่ขา P0.X เป็น 1 ดังนั้น PORT 0 เมื่อให้ทำงานเป็นพอร์ทส่งข้อมูล (ไม่ใช่ส่งตำแหน่งหน่วยจำ) จะไม่สามารถแสดงสภาวะลอจิก 1 ได้จึงต้องต่อตัวต้านทาน Pull Up ไว้ภายนอก ระหว่างขา P0.X กับไฟเลี้ยงวงจร ถ้าจะใช้พอร์ท 0 สำหรับรับข้อมูลเข้าจะต้องเขียน 1 มาเก็บไว้ยัง D-FF เสียก่อนเพื่อให้ขา P0.X อยู่ในสภาวะ High Impedance แล้วจึงใช้คำสั่งอ่านสภาวะลอจิกเข้าไปยัง Internal Data Bus ต่อไป โดยคำสั่งอ่านสภาวะลอจิกทางพอร์ท 0 ก็จะทำให้วงจร Timing and Control สร้างสัญญาณ Read Pin สำหรับการอ่านสภาวะลอจิกข้างต้น ถ้าไม่เขียน 1 มาเก็บไว้ยัง D-FF ก่อนที่จะอ่านข้อมูลแล้วอาจมีข้อมูลค้างอยู่ที่ D-FF ทำให้ Q เป็น-0 และ $\overline{Q}$ เป็น 1 ซึ่งทำให้ FET ตัวล่าง ON สัญญาณที่ต่อเข้ามาที่ขา P0.X ไม่ว่าจะมีสภาวะลอจิกใดจะถูกดึงลงกราวด์ ดังนั้นเมื่ออ่านข้อมูลไปก็จะพบว่าเป็น 0 เสมอ ในการอ่านข้อมูลจากหน่วยความจำภายนอกนั้นวงจร Timing and Control ก็จะเขียนข้อมูลมายัง D-FF ให้เป็น 1 และสร้างสัญญาณ Control ให้มีลอจิกเป็น 0 ก่อนจะอ่านข้อมูลเข้าไปด้วย

Port 1 (ที่ขา 1 - 8)

เป็นพอร์ทขนานขนาด 8 บิท ในรูปที่ 3.2 คือขา P1.0 ถึง P1.7 (ขา 1 - 8) P1.0 หมายถึงบิท 0 ของพอร์ท 1 ซึ่งเป็นบิท Least Significant Bit และบิท P1.7 หมายถึงบิทที่ 7 ของพอร์ท 1 ซึ่งเป็น Most Significant bit โครงสร้างของพอร์ท 1 แต่ละบิทมีดังรูปที่ 3.4

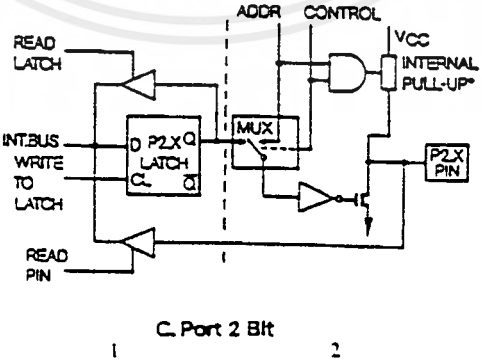


รูปที่ 3.4 โครงสร้างของพอร์ท 1

ส่วนที่ 1 คือ Port 1 Latch ในรูปที่ 3.1 ซึ่งจะมีการทำงานเหมือนส่วนที่ 1 ของพอร์ท 0 ในรูปที่ 3.3 ส่วนที่ 2 คือ Port 1 Driver ในรูปที่ 3.1 Port 1 Driver นี้จะมีตัวต้านทานต่ออยู่เป็น Internal Pull Up พอร์ท 1 นี้จะใช้ทำหน้าที่เป็นตัวรับ-ส่งข้อมูลเท่านั้น ข้อมูลที่ส่งออกมาทางพอร์ท 1 จะถูก Latch ไว้แล้วส่งออกไปทางแต่ละขา ก่อนที่จะอ่านข้อมูลเข้าไปทางพอร์ท 1 จะต้องเขียน 1 ไปยังทุกบิทของพอร์ท 1 เสียก่อน เพื่อให้ FET อยู่ในสภาวะ OFF ก่อน มิฉะนั้นแล้วถ้ามีข้อมูล 0 ส่งออกมาค้างอยู่ที่ D-FF จะทำให้ FET อยู่ในสภาวะ ON ดังนั้นถ้าสัญญาณภายนอกส่งเข้ามาที่ขานี้ก็จะถูกลัดวงจรลงกราวด์ โดยไม่สนใจว่าสภาวะลอจิกของสัญญาณที่เข้ามาจะเป็นอะไร ข้อมูลที่อ่านเข้าไปจึงจะเป็น 0 เสมอ

Port 2 (ที่ขา 21 - 28)

พอร์ทขนานขนาด 8 บิท คือขา P2.0 ถึง P2.7 (บิท 0 ถึงบิท 7 ของพอร์ท 2) ในรูปที่ 3.3 โครงสร้างของพอร์ท 2 แต่ละบิทจะมีดังรูปที่ 3.5



รูปที่ 3.5 โครงสร้างของพอร์ท 2

ลักษณะโครงสร้างจะเหมือนกับ Port 0 แตกต่างกันที่ใน Port 2 นั้นภาค Driver จะใช้งานเพียง 2 ลักษณะคือ

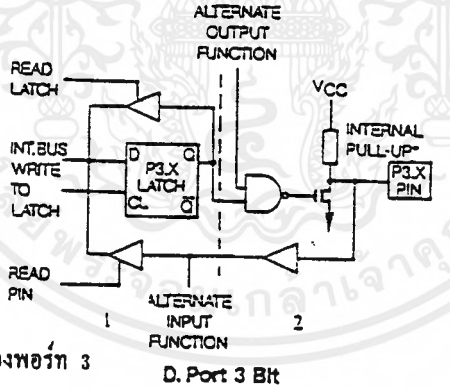
1. ใช้ส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อ ค่าตำแหน่งนี้เป็น 8 บิท บนของค่าตำแหน่ง

2. ใช้เป็นพอร์ทรับ และส่งข้อมูลกับภายนอก

ดังนั้นภาค Driver ของพอร์ท 2 จึงแตกต่างจาก Driver ของพอร์ท 0 โดยที่ในพอร์ท 2 นั้นจะมีเฉพาะ ADDR (ตำแหน่งหน่วยความจำ) เข้ามาที่ MUX (Multiplexer) เท่านั้น นอกนั้นแล้วการทำงานจะเหมือนกัน และที่เอาต์พุตของพอร์ท 2 จะมี Internal pull-up ซึ่งเป็นตัวต้านทาน และจะทำให้เอาต์พุตของพอร์ท 2 แสดงสถานะลอจิกเป็น 1 ได้ ถ้า FET อยู่ในสภาวะ OFF บางครั้งเรียกว่า " Quasi-bidirectional " เมื่อใช้เป็นพอร์ทอินพุตก็สามารถทำได้โดยการต่อสัญญาณภายนอกเข้ามาโดยตรง ถ้าสัญญาณภายนอกเป็น 0 ก็จะมีกระแสไหลออกจากพอร์ท (Source Current) ในการที่จะใช้พอร์ทนี้เป็นพอร์ทรับข้อมูลเข้า จะต้องเขียน 1 ไปยังแต่ละบิทของพอร์ทเสียก่อน ดังได้อธิบายในเรื่อง Port 0 และ Port 1

Port 3 (ที่ขา 10 - 17)

คือขา P3.0 ถึง P3.7 หรือขา 10 - 17 ตามลำดับ ในรูปที่ 3.2 พอร์ทนี้มีโครงสร้างดังรูปที่ 3.6



รูปที่ 3.6 โครงสร้างของพอร์ท 3

ส่วนที่ 1 ในรูปที่ 3.6 เป็นส่วน Latch ข้อมูลที่เขียนมายังพอร์ท 3 ทาง Internal Bus เหมือนกับพอร์ทอื่น ๆ และพอร์ท 3 จะมี Internal pull up อยู่ทุกบิท แต่พอร์ท 3 นี้แต่ละบิทจะใช้การทำงานอื่นได้โดยใช้คำสั่งควบคุมการทำงาน ในส่วนที่ 2 จะมีสัญญาณ Alternative Output Function เป็นสัญญาณที่ส่งออกในกรณีที่ใช้พอร์ท 3 ทำงานในฟังค์ชั่นอื่น และจุด Alternative Output Function เป็นจุดที่จะเอาสัญญาณไปเข้ากับส่วนอื่นตามการทำงานของบิทนั้น

แต่ละบิทของพอร์ท 3 จะมีฟังก์ชั่นอื่นดังนี้

P3.0/ RXD (Serial Input Port) เป็นขาที่ใช้รับข้อมูลแบบอนุกรม

P3.1/ TXD (Serial Output Port) เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม

P3.2/ $\overline{\text{INT0}}$ (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

P3.3/ $\overline{\text{INT1}}$ (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

P3.4/ TO (Time / Counter 0 External Input) ขารับสัญญาณเข้าไปยังวงจร Time / Counter 0 ที่ทำหน้าที่นับจำนวนไซเคิลของสัญญาณ TO นี้ หรือสัญญาณนาฬิกาก็ได้

P3.5/ T1 (Timer / Counter 1 External Input) ขารับสัญญาณเข้าไปยังวงจร Time / Counter 1 ซึ่งมีการทำงานเหมือนกับ To

P3.6/ $\overline{\text{WR}}$ (External Data Memory Write Strobe) ขาสัญญาณควบคุมการเขียนข้อมูลไปยังหน่วยความจำสำหรับข้อมูลภายนอก 89C52

P3.7/ $\overline{\text{RD}}$ (External Data Memory Read Strobe) ขาสัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลภายนอก

RST (ที่ขา 9)

ขารีเซ็ทขานี้จะใช้ทำการรีเซ็ทการทำงานของ 89C52 ที่ขา RST ภายใน 89C52 จะมีตัวต้านทานต่อระหว่างขานี้กับกราวด์ (Ground) ถ้าป้อนสัญญาณที่มีสภาวะลอจิก 1 เข้าไปที่ขานี้จะเป็นการรีเซ็ทการทำงานของ 89C52 ดังนั้นจึงสามารถต่อตัวเก็บประจุ (Capacitor) ภายนอกระหว่างขา RST กับไฟเลี้ยง + 5 โวลท์ เพื่อให้เกิดการรีเซ็ท เมื่อเริ่มป้อนไฟเลี้ยงให้กับ 89C52 ซึ่งเรียกว่า Power on reset การรีเซ็ทจะทำให้ค่าในรีจิสเตอร์ต่าง ๆ เปลี่ยนแปลงไปเป็นค่าหนึ่งดังตารางรูปที่ 3.7

REGISTER	CONTENT
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	00H
DPTR	0000H
P0-P3	0FFH
IP	00H
IE	0X0000000B
TMOD	00H
TCON	00H
T2CON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2	00H
TL2	00H
RCAP2H	00H
RCAP2L	00H
SCON	00H
SBUF	Indeterminate
IOCON	00H

รูปที่ 3.7 ค่าในรีจิสเตอร์เมื่อเกิดการรีเซ็ท 89C52

ในตารางรูปที่ 3.7 ช่องทางขวาเป็นค่าของรีจิสเตอร์ที่อยู่ทางซ้ายเมื่อสิ้นสุดการรีเซ็ท ในรีจิสเตอร์ SBUF เมื่อสิ้นสุดการรีเซ็ทจะมีค่าที่ไม่แน่นอน และพอร์ทจะอยู่ในสภาวะลอจิก 1 ทุกบิทตลอดเวลาที่สัญญาณของขา RST เป็น HIGH อยู่

เมื่อสัญญาณที่ขา RST กลับเป็น 0 ก็จะออกจากการรีเซ็ท 89C52 จะเริ่มทำงานจากคำสั่งที่อยู่ใน Program memory ตำแหน่ง 0000H เพราะค่าของรีจิสเตอร์ PC (Program Counter) ซึ่งใช้ชี้ตำแหน่งโปรแกรมที่จะทำงาน ถูกเปลี่ยนให้เป็น 0000H ดังนั้นผู้ใช้จะต้องเขียนโปรแกรมมาเก็บไว้ที่ตำแหน่ง 0000H ในเครื่องไมโครคอมพิวเตอร์แบบบอร์ดเดี่ยว (Single Board Micro-computor) จะมีโปรแกรมที่เขียนเก็บไว้เริ่มจากตำแหน่ง 0000H นี้ เรียกว่า มอนิเตอร์โปรแกรม (Monitor program) ที่จะคอยรับการกดแป้นพิมพ์ (Keyboard) และแสดงผลทางตัวแสดงผล (Display) แบบ 7 Segment

ALE (ที่ขา 30)

Address Latch Enable ขานี้จะส่งสัญญาณที่มีความถี่ 1/6 เท่าของสัญญาณนาฬิกาจากออสซิลเลเตอร์ สัญญาณนี้จะส่งออกมาตลอดเวลา ยกเว้นบางครั้งของการติดต่อกับหน่วยความจำสำหรับข้อมูลภายนอก 89C52 สัญญาณนี้จะใช้บอกกับอุปกรณ์ภายนอก 89C52 ว่าขณะนี้สัญญาณนี้ Active (เป็นลอจิก 1) จะมีการส่งข้อมูลที่เป็น 8 บิทล่างของตำแหน่งหน่วยความจำภายนอก 89C52 ที่ต้องการติดต่อออกไปทางพอร์ท 0 อุปกรณ์ภายนอกจะใช้สัญญาณนี้ในการ Latch ข้อมูลไว้เพราะพอร์ท 0 จะส่งค่าตำแหน่งหน่วยความจำออกมาเพียงชั่วขณะเท่านั้น ซึ่งในเวลาต่อมาพอร์ท 0 จะใช้รับ-ส่งข้อมูลกับหน่วยความจำภายนอก สัญญาณ ALE จะสามารถต่อเข้ากับอุปกรณ์ TTL ชนิด LS ได้ถึง 8 อินพุต

$\overline{\textbf{PSEN}}$ (ที่ขา 29)

Program Store Enable เป็นขาที่ 29 ในรูปที่ 3.2 ขานี้ปกติจะให้ลอจิก 1 แต่จะส่งลอจิก 0 เมื่อต้องการอ่านคำสั่ง (Fetch Instruction) ที่จะนำไปทำงานมาจากหน่วยความจำสำหรับโปรแกรมภายนอก 89C52 ในกรณีที่อ่านคำสั่งซึ่งเก็บอยู่ในหน่วยความจำสำหรับโปร-แกรมภายใน 89C52 แล้วสัญญาณนี้จะไม่เปลี่ยนลอจิกเป็น 0 ขา PSEN นี้สามารถต่อไปยังขาอินพุตของ TTL ชนิด LS ได้ถึง 8 อินพุต

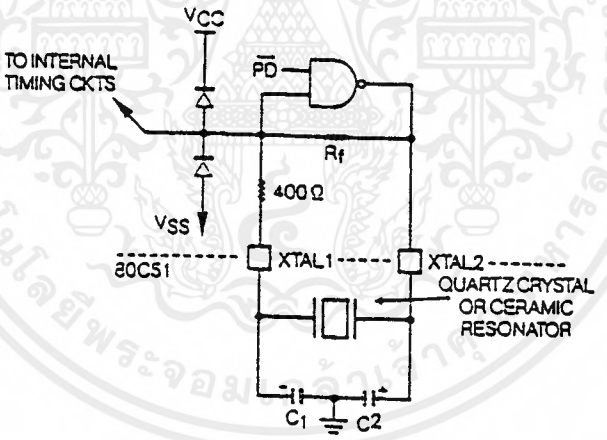
$\overline{\textbf{EA}}$ (ที่ขา 31)

External Access ขา 31 ของรูปที่ 3.2 ขานี้เป็นขาอินพุตที่ต่อเข้าไปยังวงจร Timing

and Control ในรูปที่ 3.1 เพื่อควบคุมการสร้างสัญญาณ $\overline{\text{PSEN}}$ ถ้าป้อนสัญญาณลอจิก 0 เข้าไปที่ขา $\overline{\text{EA}}$ นี้แสดงว่าโปรแกรมในตำแหน่ง 0000H ถึง 0FFFH ที่ต้องการให้ทำงานถูกเก็บไว้ภายนอก 89C52 จะต้องสร้างสัญญาณ $\overline{\text{PSEN}}$ ออกไปยังภายนอก เพื่อทำการ FETCH คำสั่งเข้ามาทำงาน แต่ถ้าสัญญาณที่ป้อนให้ขา $\overline{\text{EA}}$ เป็น 1 หมายความว่าโปรแกรมในตำแหน่ง 0000H ถึง 0FFFH ถูกเก็บไว้ภายใน 89C52 การทำงานในตำแหน่งหน่วยความจำช่วงนี้จะอ่านคำสั่งต่าง ๆ จาก ROM ภายใน 89C52

XTAL 1 (ที่ขา 19)

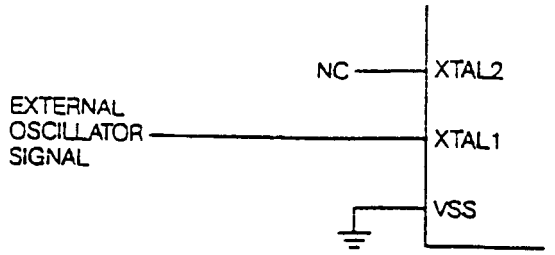
ขาที่ 19 ของรูปที่ 3.2 ขานี้จะต่อเข้ากับขาของ Inverting Amplifier (วงจรขยายแบบป้อนกลับเฟสสัญญาณ) ที่ประกอบเป็นวงจรออสซิสเลเตอร์ ในรูปที่ 3.8 จะเห็นวงจรภายในของออสซิลเลเตอร์ NAND Gate จะทำหน้าที่เป็นวงจรขยายแบบกลับเฟสของสัญญาณที่จะควบคุมให้มีการออสซิสเลตหรือไม่ก็ขึ้นกับสัญญาณ PD ซึ่งต่อมาจากบิท PD ของรีจิสเตอร์ PCON ถ้าต้องการใช้สัญญาณนาฬิกา (Clock Signal) จากภายนอกมาเป็นสัญญาณนาฬิกา มาควบคุมการทำงานของ 89C52 ก็ให้ป้อนสัญญาณเข้ามาที่จุดนี้ แต่ถ้าต้องการใช้วงจรออสซิสเล-เตอร์ภายในก็ให้ต่อ Crytal หรือเซรามิคเรโซเนเตอร์ ดังรูปที่ 3.8 คาปาซิเตอร์ในวงจรควรมีค่าประมาณ 20 PF



รูปที่ 3.8 วงจรออสซิสเลเตอร์ภายใน 89C52

XTAL 2 (ที่ขา 18)

ขาที่ 18 ของรูปที่ 3.2 ขานี้เป็นจุดเอาต์พุตของวงจรขยายแบบกลับสัญญาณที่ประกอบเป็นวงจรออสซิสเลเตอร์ (อินพุตคือขา XTAL 1) ถ้าจะใช้สัญญาณนาฬิกาที่สร้างมาจากภายนอกมาเป็นสัญญาณนาฬิกาของ 89C52 แล้ว ให้ปล่อยขานี้ลอยไว้แล้วป้อนสัญญาณนาฬิกาจากภายนอกเข้ามาที่ขา XTAL 1 ดังรูปที่ 3.9



รูปที่ 3.9 89C52 ที่ทำงานโดยสัญญาณที่มาจากภายนอก

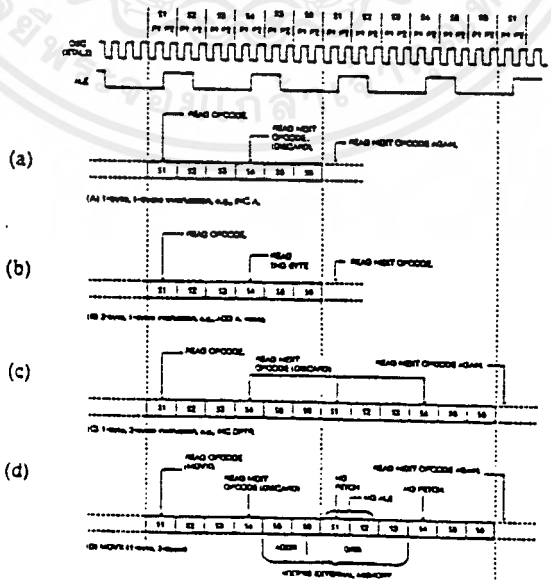
3.2 การทำงานของ 89C52

คอมพิวเตอร์จะทำงานด้วยวงจรที่เรียกว่าฮาร์ดแวร์ (Hardware) ประกอบขึ้นมาเพียงอย่างเดียวไม่ได้ จะต้องมีโปรแกรม หรือคำสั่งที่จัดเรียงกันไว้ให้คอมพิวเตอร์ทำงานตามลำดับ ใน 89C52 ก็เช่นกัน ผู้ใช้ต้องเขียนโปรแกรมเป็นภาษาเครื่อง ซึ่งอยู่ในรูปของเลขฐาน 2 เก็บไว้ในหน่วยความจำประเภท Program Memory แต่ละคำสั่งของ 89C52 อาจประกอบด้วย 1 , 2 หรือ 3 ไบท์ แล้วแต่ว่าจะเป็นคำสั่งให้ทำงานอะไร คอมพิวเตอร์ก็จะเหมือนกับคนที่จะต้องทำงานตามคำสั่ง เมื่อรับคำสั่งแล้วก็จะไปทำตามคำสั่งนั้นเสร็จสิ้นแล้วก็กลับมารับคำสั่งต่อไป

จากรูปที่ 3.1 เมื่อเริ่มป้อนไฟเลี้ยงให้กับ 89C52 ซึ่งมีวงจรนับ Power on reset ต่ออยู่จะมีการรีเซ็ทเกิดขึ้น การทำงานภายใน 89C52 จะเริ่มจากบล็อก Program Counter ซึ่งเป็นวงจรนับ (Counter Circuit) ชนิดหนึ่ง ส่งค่าตำแหน่งหน่วยความจำสำหรับโปรแกรมลงไปบนบัส (BUS) หมายเลข 1 บัสนี้มีขนาด 16 บิท ค่าตำแหน่งหน่วยความจำนี้จะถูกส่งไปเก็บไว้ที่ Program ADDR Register ที่เป็นวงจร Latch ข้อมูลซึ่งเป็นค่าตำแหน่งหน่วยความจำ จะปรากฏที่บนบัส 16 บิท หมายเลข 2 ถ้าเป็นค่าตำแหน่งหน่วยความจำแรกหลังจากรีเซ็ท ค่าตำแหน่งหน่วยความจำจะเป็น 0000H หน่วยความจำสำหรับโปรแกรมจะเลือกได้ว่าเป็น ROM ภายในหรือภายนอก 89C52 โดยการป้อนสภาวะลอจิกเข้าไปที่ 89C52 ทางขา $\overline{EA}$ ซึ่งต่ออยู่กับส่วน Timing and Control ทำหน้าที่เป็นวงจรถอดรหัส (Decoder) แล้วสร้างสัญญาณควบคุมต่อไป ถ้าป้อนสัญญาณลอจิก 0 เข้าไปที่ขา $\overline{EA}$ จะเป็นการเลือกใช้ ROM ภายใน 89C52 โดยที่วงจร Timing and Control จะสร้างสัญญาณไปยัง ROM ภายในให้ส่งข้อมูลที่เป็นคำสั่งจากตำแหน่งที่ถูกชี้ด้วยค่าตำแหน่งที่ส่งมาทางบัสหมายเลข 2 ข้อมูลจาก ROM จะถูกส่งลงไปยังบัสหมายเลข 3 ที่เรียกว่า Internal Data Bus แล้วนำไปเก็บไว้ที่ Instruction Register (เป็นวงจร Latch) เพื่อส่งต่อไปให้กับวงจร Timing and Control ทำการถอดรหัสแล้วควบคุมการทำงานส่วนอื่น ๆ ต่อไปแล้วแต่ว่าเป็นคำสั่งให้ทำงานอะไร ในกรณีที่เลือก ROM ภายนอก 89C52 โดยป้อนสัญญาณลอจิก 1 เข้าไปที่ขา $\overline{EA}$ จะทำให้วงจร Timing and Control ส่งสัญญาณไปยังพอร์ท 0 และพอร์ท

2 เพื่อส่งค่าตำแหน่งหน่วยความจำบนบัสหมายเลข 2 ออกไปชี้หน่วยความจำภายนอก จากนั้นจะอ่านข้อมูลที่เป็นคำสั่งกลับเข้ามาทางพอร์ท 0 ไปยัง Internal Data Bus แล้วนำไปเก็บไว้ที่ Instruction Register เพื่อทำงานต่อไปเหมือนกับตอนอ่านคำสั่งจาก ROM ภายในการทำงานในช่วงส่งค่าตำแหน่งหน่วยความจำไปยังหน่วยความจำ แล้วอ่านข้อมูลที่เป็นคำสั่งกลับเข้ามาเก็บไว้ใน Instruction Register เรียกว่าเป็นช่วงของการ Fetch (Fetch Cycle) ช่วงต่อไปจะเป็นช่วงของการทำงานตามคำสั่งเรียกว่า Execute Cycle เช่นถ้าเป็นคำสั่งให้บวกข้อมูลในรีจิสเตอร์ Accumulator กับข้อมูลจากหน่วยความจำ Data Memory ภายใน RAM ตำแหน่ง 23H วงจร Timing and Control ก็จะส่งสัญญาณให้ Instruction Register ส่งค่าตำแหน่งหน่วยความจำ 23 H ลงไปยัง Internal Data Bus แล้วนำข้อมูลไปเก็บไว้ที่ RAM ADDR Register เพื่อใช้ชี้ตำแหน่งหน่วยความจำ RAM จากนั้น Timing and Control จะสั่งให้ RAM ส่งข้อมูลที่เก็บอยู่ในหน่วยความจำตำแหน่ง 23H ลงมายัง Internal Data Bus แล้วนำข้อมูลไปเก็บไว้ที่ TMP1 (วงจร Latch) ขณะเดียวกันวงจร Timing and Control ก็จะส่งสัญญาณไปยัง ACC ให้ส่งข้อมูลมายัง TMP2 (วงจร Latch) วงจร ALU ซึ่งโครงสร้างเป็นวงจรทำการคำนวณทางคณิตศาสตร์ (บวก , ลบ , คูณ , หาร) และยังสามารถทำงานทางลอจิก (AND , OR , NOT , XOR) จะทำการบวกเลขจาก TMP1 และ TMP2 เข้าด้วยกัน ผลลัพธ์ที่ได้จะส่งผ่าน Internal Data Bus กลับไปเก็บยัง ACC PSW (Program Status Word) ซึ่งจะทำหน้าที่เก็บสถานะผลลัพธ์ของการทำงานใน ALU เช่น ผลลัพธ์การบวกมีค่าเกิน 8 บิท ก็จะทำให้บิทหนึ่งใน PSW ถูก SET เป็น 1

การทำงานที่กล่าวมาข้างต้นจะขึ้นกับสัญญาณควบคุมที่สร้างมาจากวงจร Timing and Control และสัญญาณที่สร้างขึ้นนี้จะอ้างอิงกับสัญญาณนาฬิกาที่สร้างมาจากวงจร Oscillator ทำให้การทำงานต่าง ๆ เป็นไปตามลำดับที่ผู้ผลิตได้ออกแบบไว้ ดังในรูปที่ 3.10



รูปที่ 3.10 ลำดับสถานะการทำงานใน MCS-51

คำสั่งแต่ละคำสั่งของ 89C52 จะใช้เวลาทำงาน 1 , 2 หรือ 3 ไซเคิลของเครื่อง (Machine Cycle) แล้วแต่ว่าเป็นคำสั่งประเภทใด 1 ไซเคิลของเครื่องจะใช้เวลา 12 ไซเคิล ของสัญญาณนาฬิกา ดังนั้นแต่ละคำสั่งของ 89C52 จะใช้เวลาการทำงาน 12 , 24 หรือ 36 ไซเคิลของสัญญาณนาฬิกานั้นเอง แต่ละไซเคิลของเครื่องจะถูกแบ่งออกเป็น 6 State คือ S1 , S2 , S3 , S4 , S5 , และ S6 แต่ละ State จะประกอบด้วย 2 ไซเคิล ของสัญญาณนาฬิกา ในไซเคิลแรกจะเรียกว่าเฟส 1 (P1) และไซเคิลที่ 2 เรียกว่าเฟส 2 (P2) ในแต่ละเฟสจะนับตั้งแต่ขอบขาลงของสัญญาณนาฬิกา ถึงขอบขาลงของสัญญาณนาฬิกาที่อยู่ถัดไปดังในรูปที่ 3.10 เมื่อ 89C52 ทำงานเสร็จ 1 ไซเคิลของเครื่องก็จะเริ่มทำงาน State 1 Phase 1 (S1 P1) ของไซเคิลต่อไป ใน 1 ไซเคิล ของเครื่องวงจร Timing and Control จะสร้างสัญญาณ ALE ออกมา 2 ไซเคิล เพื่อ Fetch คำสั่งเข้าไป 2 ครั้งเสมอ ที่บริเวณขอบขาขึ้นของสัญญาณ ALE คำสั่งใดจะมีกี่ไบท์ หรือใช้เวลาทำงานกี่ไซเคิล จะดูได้จากตารางชุดคำสั่ง 89C52

คำสั่งประเภท 1 ไบท์ 1 ไซเคิลของเครื่องได้แก่คำสั่ง INC A จะมีการอ่านคำสั่งจากหน่วยความจำสำหรับโปรแกรม 2 ครั้ง ที่เวลาประมาณขอบขาขึ้นของสัญญาณ ALE เมื่อคำสั่งแรกถูกอ่านเข้าไปที่เวลา ที่เวลาขอบขาขึ้นของสัญญาณ ALE แรก แล้วนำไปเก็บที่ Instruction Register เพื่อให้วงจร Timing and Control ถอดรหัสแล้วเข้าอยู่การ Execute ขณะเดียวกันก็จะเริ่มต้นการ Fetch คำสั่งที่อยู่ในหน่วยความจำคำสั่งถัดไปเข้ามา และคำสั่งที่ 2 จะถูกอ่านเข้ามาที่เวลาขอบขาขึ้นของสัญญาณ ALE ถัดไป วงจร Timing and Control เมื่อถอดรหัสคำสั่งแรกก็จะทราบว่า การทำงานคำสั่งนี้ให้สิ้นสุดจะใช้คำสั่งเพียง 1 ไบท์ ดังนั้นคำสั่งที่ถูกอ่านมาไบท์ที่ 2 จะไม่ถูกนำมาทำงาน เพียงแต่อ่านเข้ามาแล้วทิ้งไป (Discard) ดังในรูปที่ 3.10a

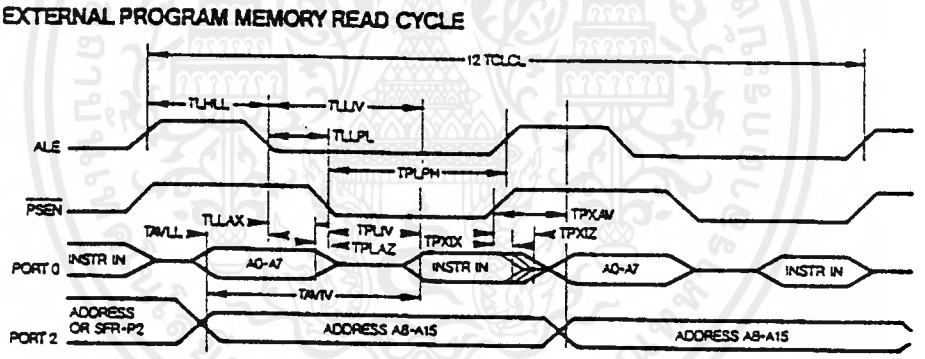
คำสั่งประเภท 2 ไบท์ และใช้เวลาเพียง 1 ไซเคิล ของเครื่องได้แก่คำสั่ง ADD A , # Data ในหนึ่งไซเคิลของเครื่องนี้ จะมีการอ่านคำสั่งเข้ามา 2 ไบท์ เหมือนกับคำสั่งประเภท 1 ไบท์ 1 ไซเคิลของเครื่อง แตกต่างกันที่ไบท์ที่ 2 จะถูกนำมาใช้งานด้วยไม่ได้ถูกทิ้งไปดังในรูปที่ 3.10b ตัวอย่างของคำสั่ง ADD A , # 33H จะเขียนเป็นภาษาเครื่องได้ 2 ไบท์ คือ 24 33 เมื่ออ่านคำสั่งไบท์แรกคือ 24 เข้าไปไว้ที่ Instruction Register และ Timing and Control จะถอดรหัสพบว่าเป็นคำสั่งบวกเลข ก็จะส่งสัญญาณไปยัง Accumulator ให้เอาข้อมูลไปไว้ที่ TMP1 เมื่อคำสั่งที่ 2 ถูกอ่านเข้ามาที่ Instruction Register และ Timing and Control จะส่งให้เอาข้อมูลไบท์ที่ 2 ส่งลงไปยัง Internal Data Bus ไปเก็บยัง TMP1 จากนั้นวงจร ALU จะนำเอาข้อมูล TMP1 และ TMP2 มาบวกกัน ผลลัพธ์ที่ได้จะส่งออกมาจาก ALU ไปยัง Internal Data Bus แล้วไปเก็บไว้ที่ Accumulator

คำสั่งประเภท 1 , 2 หรือ 3 ไบท์ ที่ใช้เวลาทำงาน 2 ไซเคิล ของเครื่อง เช่น คำสั่ง

INC DPTR จะมีการอ่านคำสั่งเข้าไป 4 ครั้ง ทุก ๆ ขอบขาขึ้นของสัญญาณ ALE ที่มี 2 ครั้งต่อ 1 ไซเคิล ของเครื่อง ถ้าเป็นคำสั่งประเภท 1 , 2 หรือ 3 ไบท์ วงจร Timing and Control จะเอาคำสั่ง 1 , 2 หรือ 3 ไบท์ แรกเท่านั้นไปทำงาน ส่วนคำสั่งที่เหลือทิ้งไปดังในรูปที่ 3.10c คำสั่ง 1 ไบท์ ที่ใช้เวลาทำงาน 2 ไซเคิล ของเครื่องที่กล่าวมาแล้วจะไม่รวมถึงคำสั่ง MOVX ซึ่งใช้ในการอ่าน หรือเขียนข้อมูลกับหน่วยความจำ Data Memory ภายนอก การทำงานของคำสั่งนี้จะมีการ Fetch คำสั่งเข้าไป 2 ไบท์ ในไซเคิลของเครื่องแรก ในไซเคิลของเครื่องที่ 2 จะไม่มีการ Fetch คำสั่งเข้าไป แต่จะเป็นช่วงเวลาของการอ่าน หรือเขียนข้อมูลกับ Data Memory ภายนอก สัญญาณ ALE ซึ่งปกติจะเปลี่ยนเป็น 1 ที่ S1 P2 ก็จะไม่เปลี่ยนเป็น 1 ในไซเคิลของเครื่องที่ 2 โดยจะเป็น 0 อยู่จนกว่าจะถึงเวลา S4 P2 ของไซเคิลของเครื่องที่ 2 สัญญาณ ALE จะเปลี่ยนเป็น 1 เพื่อการอ่าน หรือเขียนข้อมูลกับ Data Memory ภายนอก

3.3 ไดอะแกรมเวลาของการติดต่อกับหน่วยความจำ

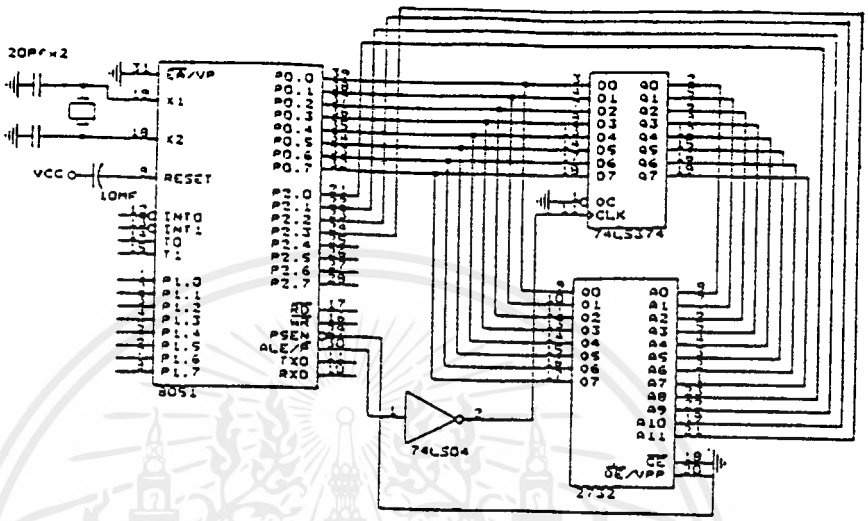
การอ่านข้อมูลจากหน่วยความจำสำหรับโปรแกรมภายนอก 89C52 นั้น ลำดับสัญญาณตามเวลา (Timing Diagram) ของสัญญาณที่ทำการอ่านคำสั่ง ดังรูปที่ 3.11



รูปที่ 3.11 Timing Diagram ของการอ่านโปรแกรมจากหน่วยความจำภายนอก

การอ่านคำสั่ง (Fetch) จาก Program area ภายนอก จะเริ่มจาก 89C52 ส่งสัญญาณลอจิก 1 ออกมาทางขา ALE ขณะนี้สัญญาณที่ขา $\overline{\text{PSEN}}$ จะเป็น 1 จากนั้น Port 0 จะส่งค่าตำแหน่งหน่วยความจำ 8 บิทล่าง และพอร์ท 2 จะส่งตำแหน่งหน่วยความจำ 8 บิทบนออกมา แล้วสัญญาณ ALE จะกลับเป็น 0 อุปกรณ์ภายนอกจะสามารถใช้ขอบขาลงของสัญญาณ ALE เพื่อ Latch ตำแหน่งหน่วยความจำที่พอร์ท 0 ไว้ จากนั้นพอร์ท 0 ก็จะยกเลิกการส่งค่าตำแหน่งหน่วยความจำ เข้าสู่สภาวะ High Impedance และสัญญาณ $\overline{\text{PSEN}}$ จะเป็น 0 เพื่อเตรียมรับคำสั่ง ที่ส่งออกจากหน่วยความจำภายนอก เข้าไปยัง 89C52 เพื่อทำงานต่อไป เมื่อคำสั่งถูกอ่านเข้าไปเก็บใน Instruction Register (ดูรูปที่ 3.1) แล้วสัญญาณ $\overline{\text{PSEN}}$ จะกลับเป็น 1 พร้อมกับ

สัญญาณ ALE ก็จะกลับเป็น High (เพื่อเตรียมคำสั่งต่อไปทำงาน ข้อมูลในพอร์ท 2 จะคงที่ตลอดเวลา ตั้งแต่สัญญาณ ALE เป็น 1 จนกระทั่งสัญญาณ ALE เปลี่ยนเป็น 0 และกลับเป็น 1 อีกครั้งหนึ่ง จากนั้นจะเริ่มลำดับการ Fetch ข้อมูลไบท์ที่ 2 จากหน่วยความจำสำหรับโปรแกรม ซึ่งจะมีการเปลี่ยนแปลงของสัญญาณตามเวลา เหมือนกับการ Fetch ไบท์แรกนั้นเอง จาก Timig Diagram ดังกล่าว จะออกแบบวงจรที่มี Program Memory อยู่ภายนอก 89C52 ได้ดังตัวอย่างในรูปที่ 3.12



รูปที่ 3.12 วงจรที่มี Program Memory อยู่ภายนอก 89C52

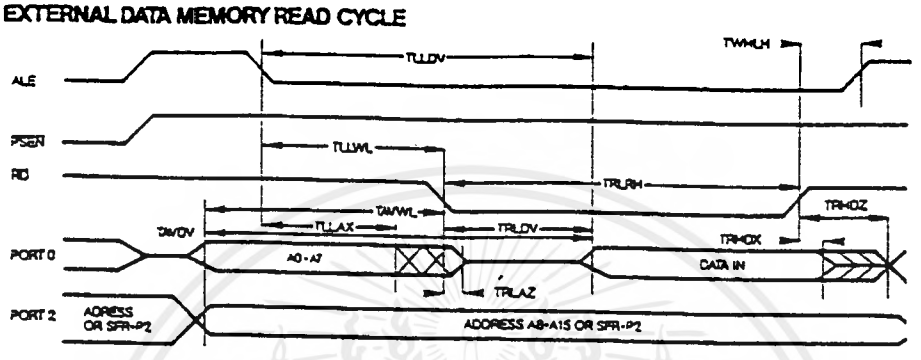
74LS374 ในรูปที่ 3.12 จะทำหน้าที่ Latch ตำแหน่งในหน่วยความจำ 8 บิทล่าง ที่เวลาขอบขาลงของสัญญาณ ALE ซึ่งสัญญาณ ALE จะถูกกลับให้เป็นตรงกันข้ามโดย Inverter 74LS04 ก่อนที่จะป้อนให้กับขา CK ของ 74LS374 และที่ขอบขาขึ้นของสัญญาณที่ออกจาก 74LS04 และ Latch ตำแหน่งหน่วยความจำ ข้อมูลที่ออกจาก 74LS374 จะเป็นค่า 8 บิทล่างของตำแหน่งหน่วยความจำที่ต้องการติดต่อ ในวงจรได้ต่อค่าตำแหน่งหน่วยความจำ 8 บิท เข้ากับ A0 ถึง A7 ของ EPROM และข้อมูลจากพอร์ท 2 บิท P2.0 ถึง P2.3 จะต่อเข้ากับ A8 - A11 ของ EPROM โดยตรง เพราะค่าตำแหน่งหน่วยความจำ 8 บิท บนที่ออกมาจากพอร์ท 2 จะคงที่ตลอดเวลา ขา $\overline{\text{PSEN}}$ ของ 89C52 จะถูกต่อเข้ากับขา $\overline{\text{OE}}$ ของ EPROM 2716 ดังนั้นเมื่อสัญญาณ $\overline{\text{PSEN}}$ มีสภาวะลอจิกเป็น 0 ก็ส่งคำสั่งที่เก็บไว้ใน EPROM ณ ตำแหน่งที่ชี้โดยข้อมูลที่ขา A0 ถึง A11 ออกมายังพอร์ท 0 และถูก 89C52 เก็บไปทำงานต่อไป

การอ่าน - เขียนข้อมูลกับหน่วยความจำสำหรับข้อมูลภายนอก 89C52

การอ่าน - เขียนข้อมูลกับ Data Memory ภายใน 89C52 นั้นจะมีสัญญาณสร้างมาจากส่วน Timing and Control โดยที่ผู้ใช้ไม่จำเป็นที่จะต้องทำความเข้าใจ แต่การอ่าน - เขียนข้อ

มูลกับ Data Memory ภายนอก อันเนื่องมาจากคำสั่ง MOVX นั้น เมื่อคำสั่งดังกล่าวถูกอ่านเข้ามายัง Instruction Register แล้ว Timing and Control จะทำการถอดรหัส และสร้างสัญญาณควบคุมดังนี้

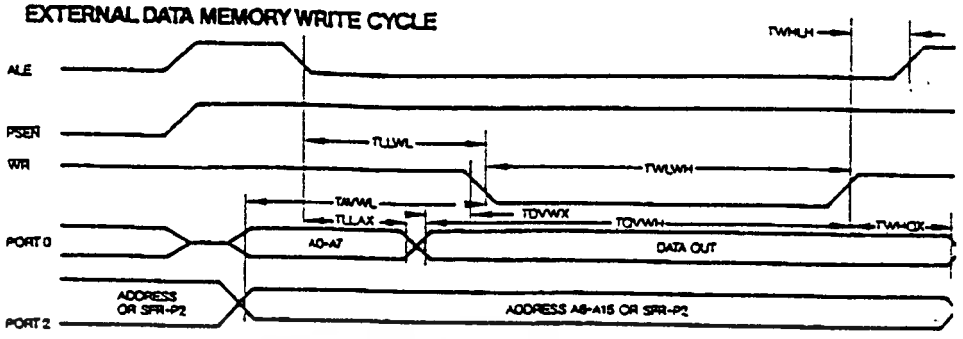
การอ่านข้อมูลจาก External Data Memory จะมีไดอะแกรมสัญญาณตามเวลาดังรูปที่ 3.13



รูปที่ 3.13 Timing diagram ของการอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูล ภายนอก 89C52

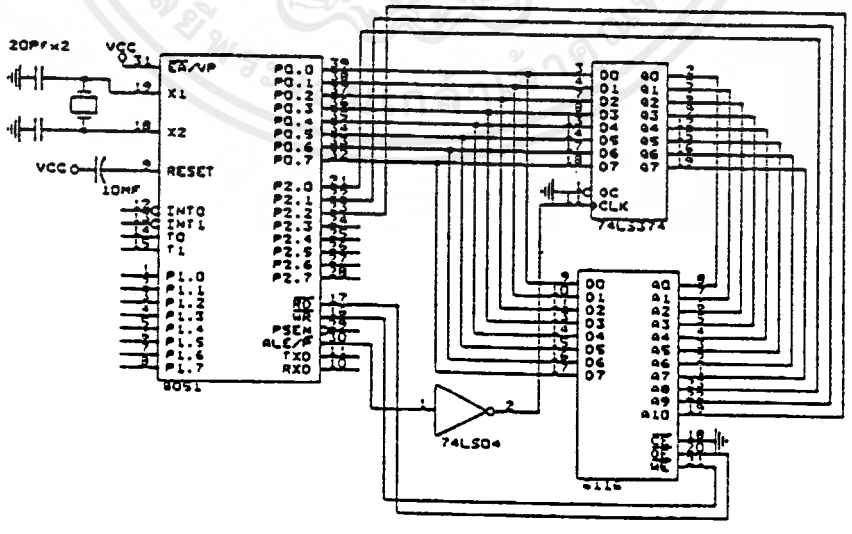
การทำงานจะเริ่มจากการส่งค่าตำแหน่งหน่วยความจำภายนอก 8 บิทล่าง ออกพอร์ท 0 และ 8 บิทบนออกทางพอร์ท 2 เมื่อส่งค่าตำแหน่งแล้ว สัญญาณ ALE ซึ่งเดิมมีลอจิกเป็น 1 จะกลับมาเป็น 0 เพื่อให้อุปกรณ์ภายนอกสามารถ Latch ตำแหน่งหน่วยความจำไว้เหมือนกับในการอ่านข้อมูลจากหน่วยความจำสำหรับโปรแกรมภายนอก 89C52 เพื่อส่งไปยังหน่วยความจำ แม้ว่าข้อมูลทางพอร์ท 0 จะเปลี่ยนแปลงไปก็ยังมีค่าตำแหน่งหน่วยความจำส่งไปยังหน่วยความจำในระหว่างการติดต่อกับ Data Memory นี้สัญญาณ $\overline{\text{PSEN}}$ จะเป็น 1 ตลอดเพราะสัญญาณ $\overline{\text{PSEN}}$ จะเป็น Active (เป็น 0) ก็ต่อเมื่อเป็นการติดต่อหน่วยความจำ สำหรับโปรแกรมภายนอก 89C52 เท่านั้น 89C52 จะส่งสัญญาณ ลอจิก 0 ออกมาทางขา $\overline{\text{RD}}$ (P3.7) เพื่อบอกกับหน่วยความจำภายนอกว่าต้องการอ่านข้อมูลเข้าไปเมื่อ 89C52 ส่งสัญญาณ $\overline{\text{RD}}$ เป็นลอจิก 0 ทำให้พอร์ท 0 เข้าสู่สถานะ High Impedance พร้อมที่จะให้หน่วยความจำภายนอกส่งข้อมูลมาบน พอร์ท 0 ซึ่งส่งมาจากหน่วยความจำภายนอกจะถูกอ่านเข้าไปเก็บที่เวลาขอบขาขึ้นของสัญญาณ $\overline{\text{RD}}$ จากนั้นสัญญาณ ALE ก็จะกลับเป็น 1 เพื่อเริ่มการทำงานในคำสั่งต่อไปในระหว่างการอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลภายนอกนี้ พอร์ท 2 จะส่งค่าตำแหน่งหน่วยความจำ 8 บิทบนออกมาตลอดเวลา

การเขียนข้อมูลไปยังหน่วยความจำสำหรับข้อมูลภายนอก 89C52 จะมีไดอะแกรมสัญญาณตามเวลาดังรูป 1.14



รูปที่ 3.14 Timing diagram ของการเขียนข้อมูลไปยังหน่วยความจำสำหรับข้อมูลภายนอก 89C52

เมื่อ 89C52 ส่งค่าตำแหน่งหน่วยความจำ 8 บิทล่างไปทางพอร์ท 0 และหน่วยความจำ 8บิทบนลงไปทางพอร์ท 2 แล้ว สัญญาณ ALE ก็จะกลับเป็น 0 อุปกรณ์ภายนอกจะสามารถใช้สัญญาณนี้ในการ Latch ค่าตำแหน่งหน่วยความจำบนพอร์ท 0 เหมือนกับการอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลภายนอก เมื่อสัญญาณ ALE เป็น 0 แล้ว 89C52 จะส่งข้อมูลที่ต้องการเขียนไปยังพอร์ท 0 แล้วจะให้สัญญาณ $\overline{WR}$ เปลี่ยนสภาวะลอจิกเป็น 0 ขณะนี้หน่วยความจำภายนอกจะต้องเขียนข้อมูลไปเก็บยังตำแหน่งที่กำหนด จากนั้น สัญญาณ $\overline{WR}$ จะกลับเป็น 1 เพื่อเป็นการบอกสิ้นสุดการเขียนข้อมูลแล้วสัญญาณ ALE ก็จะกลับเป็น 1 เพื่อ Fetch คำสั่งต่อไปมาทำงาน หน่วยความจำสำหรับข้อมูลภายนอก ที่สามารถอ่านและเขียนข้อมูลได้ดังรูปที่ 3.15

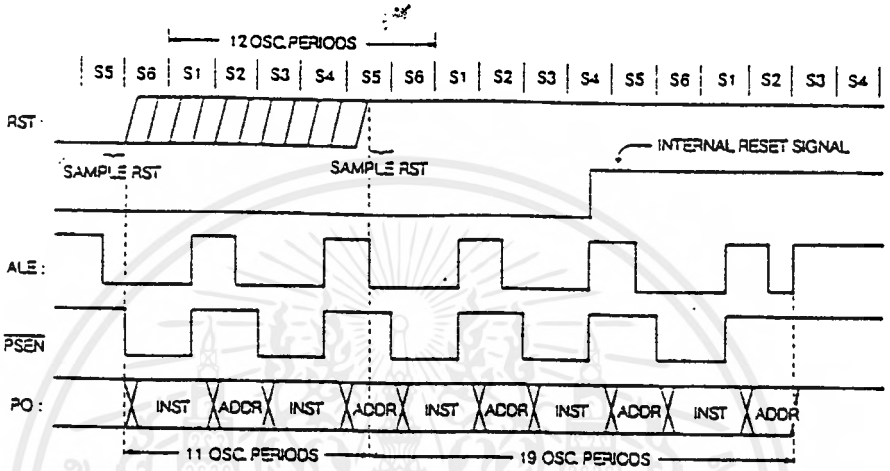


รูปที่ 3.15 วงจรที่มีหน่วยความจำสำหรับข้อมูลที่อยู่ภายนอก 89C52

74LS374 ในรูปจะใช้สำหรับ Latch ค่าตำแหน่งหน่วยความจำ 8 บิทล่างไว้ แม้ว่าข้อมูลบนพอร์ท 2 จะเปลี่ยนไป สัญญาณ RD และ WR จะอ่านหรือเขียนข้อมูลจากหน่วยความจำภายนอก 6116 เป็นหน่วยความจำแบบ RAM ที่สามารถจะอ่านและเขียนข้อมูลได้

3.4 การรีเซ็ท

เมื่อป้อนสัญญาณที่มีสภาวะลอจิกเป็น 1 เข้าไปทางขา **RST** จะไม่ได้เกิดการรีเซ็ทขึ้นทันทีทันใด แต่ลำดับการเกิดรีเซ็ทจะแสดงได้ดังไดอะแกรมตามเวลาในรูปที่ 3.16



รูปที่ 3.16 ไดอะแกรมตามเวลาของการรีเซ็ท

ในรูปที่ 3.16 เป็น Timing Diagram ของการรีเซ็ท สภาวะลอจิกของสัญญาณที่ขา RST ที่ถูกอ่านเข้ามาที่เวลา S5P2 (เฟส 2 State 5) ของทุก ๆ ไซเคิลของเครื่อง ในกรณีที่เป็นคำสั่งซึ่งมีการทำงานเสร็จสิ้นใน 2 ไซเคิล ของเครื่องก็จะตรวจสอบเฉพาะสัญญาณที่อ่านเข้ามาในไซเคิลที่ 2 ของการทำงาน ดังนั้นในการรีเซ็ท จะต้องป้อนสัญญาณที่มีสภาวะลอจิก 1 เข้าไปที่ขาเป็นเวลาอย่างน้อย 2 ไซเคิลของเครื่องหรือ 24 ไซเคิล ของสัญญาณนาฬิกาที่สร้างจากวงจรออสซิลเลเตอร์ภายใน 89C52 เพื่อให้แน่ใจว่าสัญญาณรีเซ็ท จะถูกอ่านเข้าไปตรวจสอบและทำงาน ขณะที่ทำงานรีเซ็ท 89C52 ออสซิลเลเตอร์จึงจะต้องทำงานอยู่ด้วย เมื่อ 89C52 สุ่มข้อมูลที่ขา RST แล้วตรวจพบว่าเป็นสภาวะลอจิก 1 ก็จะสร้างสัญญาณรีเซ็ทขึ้นภายใน ที่เวลา S2P4 ของไซเคิลเครื่องถัดไป ข้อมูลแต่ละพอร์ทที่ส่งออกมา จะยังคงปรากฏที่พอร์ทจนกว่าจะเกิดการรีเซ็ทขึ้นซึ่งต้องใช้เวลา 19 ไซเคิลของสัญญาณจากออสซิลเลเตอร์นับตั้งแต่เวลา S5P2 ในไซเคิลของเครื่องที่พบสัญญาณรีเซ็ท ในระหว่างเวลา 19 ไซเคิล นี้จะยังคงมีการ Fetch คำสั่งเข้าไปทำงานได้อยู่

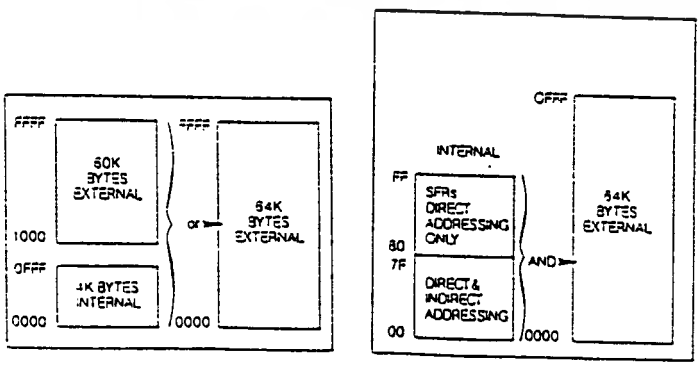
สภาวะของสัญญาณลอจิกที่ขา RST จะถูกอ่านเข้าไปตรวจสอบที่เวลา S5P2 ของทุก ๆ ไซเคิล ของเครื่อง ดังนั้นถึงแม้ว่าสัญญาณที่ขา RST จะมีลอจิกเป็น 1 มาก่อนก็ยังไม่เกิดการตรวจสอบสัญญาณรีเซ็ท ดังในรูปที่ 3.16 สัญญาณที่ขา RST อาจเป็น 1 มาตั้งแต่ State ที่ 6 ก็จะไม่เกิด

อะไรขึ้นจนกระทั่ง 1 ไซเคิลของออสซิลเลเตอร์ต่อมาซึ่งเป็นเวลา S5P2 จึงจะเกิดการตรวจสอบสัญญาณที่ขา RST ถ้าคำสั่งนั้นมีการทำงานมากกว่า 1 ไซเคิลของเครื่อง 89C52 ก็จะต้องทำงานในคำสั่งนั้นให้เสร็จสิ้นเสียก่อนจึงจะเริ่มการรีเซ็ทได้ โดย 89C52 จะดูสภาวะของสัญญาณที่ขา RST ของ S5P2 ในไซเคิลของเครื่องสุดท้ายเท่านั้น ดังนั้นใน S5P2 ของไซเคิลเครื่องแรก ๆ ในคำสั่งอาจมีสภาวะลอจิกที่ขา RST เป็น 1 แต่ที่ S5P2 ของไซเคิลเครื่องสุดท้าย มีสภาวะลอจิกที่ขา RST เป็น 0 ก็จะไม่เกิดการรีเซ็ท

ขึ้นที่เวลา S5P2 เมื่อตรวจสอบสภาวะสัญญาณที่ขา RST แล้วพบว่าเป็น 1 จะต้องรอไปจนถึงเวลา S4P2 ที่ตรวจพบสัญญาณ RST มีลอจิกเป็น 1 จนถึง S4P2 ของไซเคิลเครื่องถัดไปจะยังคงมีการ Fetch คำสั่งเข้าไปทำงานอีก 2 คำสั่ง เมื่อสัญญาณรีเซ็ทภายในเปลี่ยนเป็น 1 ก็จะเริ่มการรีเซ็ท โดยการเขียนข้อมูล 0 ไปยัง Special Function Register ทุกตัวยกเว้นพอร์ท 0 ถึงพอร์ท 3 Stack Pointer และรีจิสเตอร์ SBUF ดังตารางในรูปที่ 3.7 ระหว่างนี้ข้อมูลใน RAM ภายใน 89C52 จะไม่เปลี่ยนแปลงในระหว่างการเขียนข้อมูลลงไปยัง SFR จะยังมีการ Fetch คำสั่งเข้ามาทำงานอีก 1 คำสั่งจนกว่าจะถึง S3P1 ของไซเคิลที่ 2 (นับแต่ไซเคิลของเครื่องที่ตรวจพบลอจิก 1 ที่ขา RST) ก็จะทำให้สภาวะลอจิกที่ขา ALE และ PSEN ค้างอยู่ที่สภาวะลอจิก 1 และจะเป็นอย่างนี้ไปจนกว่าสภาวะลอจิกที่ขา RST เป็น 0 เวลาตั้งแต่พบสัญญาณลอจิก 1 ที่ขา RST ที่เวลา S5P2 จนถึงเวลาที่ ALE และ PSEN ค้างอยู่ที่ 1 จะเท่ากับ 19 ไซเคิลของออสซิลเลเตอร์เมื่อสัญญาณที่ขา RST ถูกเปลี่ยนกลับเป็นลอจิก 0 89C52 จะรออีก 1 ถึง 2 ไซเคิลของเครื่องสัญญาณ ALE และ PSEN จะเริ่มเปลี่ยนแปลงเพื่อเริ่มกระบวนการ Fetch คำสั่งเข้าไปทำงานเริ่มจากคำสั่งในหน่วยความจำสำหรับโปรแกรมตำแหน่ง 0000H

3. 5 รีจิสเตอร์ของ 89C52

หน่วยความจำของ 89C52 แบ่งออกเป็น 2 แบบ คือ หน่วยความจำสำหรับโปรแกรม (Program Area) และหน่วยความจำสำหรับเก็บข้อมูล (Data Area) ดังแสดงในไดอะแกรมรูปที่ 3.17



รูปที่ 3.17 ไดอะแกรมภาพของหน่วยความจำ 89C52

หน่วยความจำสำหรับโปรแกรมเป็นหน่วยความจำที่ 89C52 ใช้สำหรับโปรแกรมภาษาเครื่องที่ 89C52 จะทำงานเมื่อเริ่มป้อนไฟเลี้ยงให้ 89C52 หรือมีการรีเซ็ท (Reset) 89C52 จะทำให้เริ่มการทำงานจากคำสั่งในโปรแกรมตำแหน่งที่ 0000H เมื่อทำงาน 1 คำสั่งก็จะทำให้รีจิสเตอร์ PC ที่ชี้ตำแหน่งโปรแกรมมีค่าเพิ่มขึ้นเพื่อชี้ตำแหน่งของคำสั่งต่อไป ตำแหน่งสุดท้ายของหน่วยความจำคือ FFFFH หน่วยความจำสำหรับโปรแกรมนี้สามารถที่จะเลือกได้ว่าเป็นหน่วยความจำที่อยู่ภายใน 89C52 หรือภายนอก 89C52 ก็ได้ หน่วยความจำสูงสุดสำหรับโปรแกรมภายนอก 89C52 มีได้ถึง 64 KByte ทำให้สามารถใช้งานได้อย่างกว้างขวาง หน่วยความจำในช่วงนี้ 89C52 สามารถอ่านข้อมูลได้อย่างเดียว ไม่สามารถเขียนข้อมูลเข้าไปได้ระหว่างการทำงาน

หน่วยความจำสำหรับข้อมูลเป็นหน่วยความจำที่ 89C52 ใช้สำหรับเก็บหรือพักข้อมูลระหว่างที่ทำงาน หน่วยความจำสำหรับข้อมูลมี 2 แบบ แบบหนึ่งมีขนาด 128 ไบท์อยู่ภายใน 89C52 หน่วยความจำอีกแบบหนึ่งจะมีขนาด 64 กิโลไบท์ (Kbyte) ต้องต่อเพิ่มเติมเข้าไปภายนอก 89C52 หน่วยความจำภายในตำแหน่ง 0 ถึง 7FH นี้สามารถอ้างถึงได้โดยตรงคือมีการสั่งให้อ่านหรือเขียนข้อมูลไปยังตำแหน่งนั้นได้โดยตรง แต่หน่วยความจำตำแหน่ง 80H ถึง FFH นั้นเป็นรีจิสเตอร์ฟังก์ชั่นพิเศษ (Special Function Register , SFR) หน่วยความจำภายในช่วงนี้ใช้เป็นรีจิสเตอร์สำหรับงานเฉพาะอย่าง

หน่วยความจำสำหรับข้อมูลภายใน 89C52 ช่วง 00H ถึง 07FH สามารถแบ่งออกได้เป็น 3 กลุ่มคือ

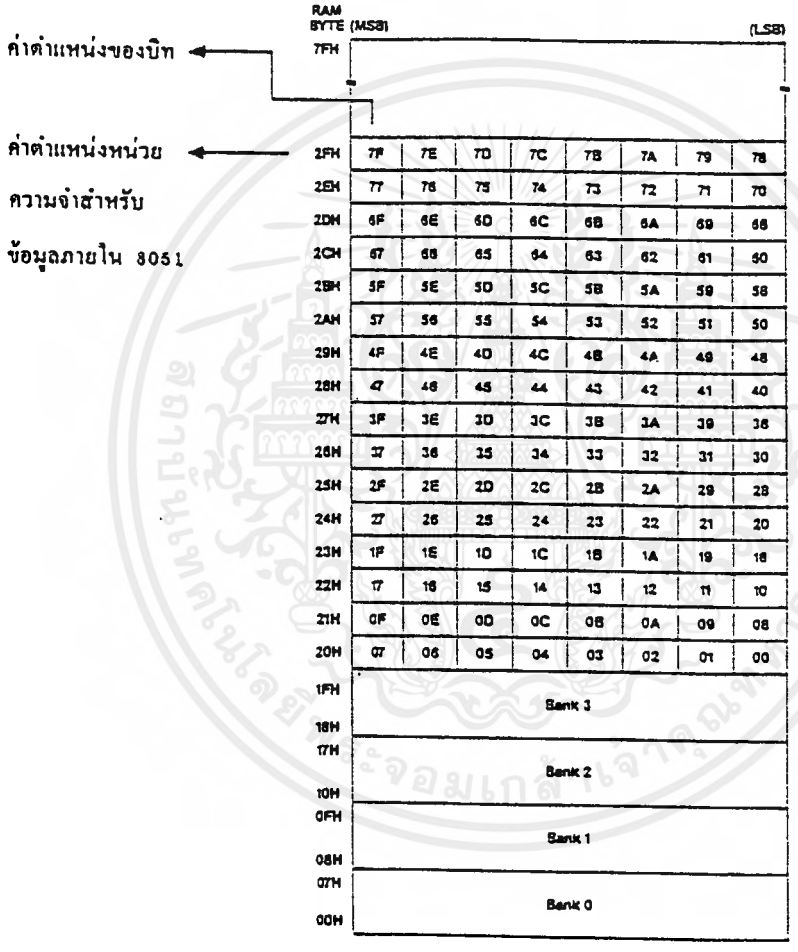
1. Register bank 0.3 อยู่ในหน่วยความจำช่วงตำแหน่ง 00H ถึง 1FH หน่วยความจำนี้แบ่ง ออกเป็น 4 ชุด ชุดละ 8 ไบท์ แต่ละชุดเราเรียกว่า BANK แต่ละไบท์ใน 1 BANK จะมีชื่อของรีจิสเตอร์ว่า R0,R1,R2,R3,R4,R5,R6 และR7 รีจิสเตอร์เหล่านี้จะเรียกใช้งานในระหว่างการทำงานของโปรแกรมได้อย่างสะดวก และรีจิสเตอร์เหล่านี้จะเป็นชื่อซ้ำกันในทุก BANK การใช้งานจึงต้องเรียกใช้งานทีละ BANK เท่านั้น โดยการกำหนดให้รีจิสเตอร์ PSW ที่จะกล่าวถึงต่อไปในบทนี้ เมื่อมีการ Reset การทำงานของ 89C52 จะเริ่มการใช้งานรีจิสเตอร์ R0 ถึง R7 ที่ BANK 0 ซึ่งรีจิสเตอร์ R0 ถึง R7 ในแต่ละ BANK นั้นจะอ้างอิงในหน่วยความจำสำหรับข้อมูลภายใน 89C52 ดังในตาราง

รีจิสเตอร์	ตำแหน่งหน่วยความจำ			
	BANK 0	BANK 1	BANK 2	BANK 3
R0	0	8	10	18
R1	1	9	11	19
R2	2	A	12	1A
R3	3	B	13	1B
R4	4	C	14	1C
R5	5	D	15	1D
R6	6	E	16	1E
R7	7	F	17	1F

ตัวอย่าง เมื่อกำลังมีการใช้งานในหน่วยความจำ BANK 1 และมีการอ้างถึงรีจิสเตอร์ R7 เช่นคำสั่ง MOV A , R7 (รหัสภาษาเครื่องคือ EFH)

การทำงานของคำสั่งนี้คือการเอาข้อมูลจากตำแหน่ง FH ของหน่วยความจำภายใน 89C52 ไปไว้ยังรีจิสเตอร์ A นั่นเอง

2 Bit Address Area เป็นหน่วยความจำในช่วงตำแหน่ง 20H ถึง 2FH หน่วยความจำแต่ละบิท ในช่วงของหน่วยความจำดังกล่าวจะสามารถตรวจสอบหรือตั้งค่าเป็น 1 หรือ 0 ได้โดยการโปรแกรมภาษาเครื่อง แต่ละบิทของหน่วยความจำช่วงนี้ จะมีค่าของตำแหน่งดังใน Memory Map รูปที่ 3.18 เช่นบิทที่ 7 ของหน่วยความจำในตำแหน่ง 2FH จะมีค่าตำแหน่งเป็น 7FH นั่นเอง



รูปที่ 3.18 ค่าตำแหน่งของแต่ละบิท

ในรูปที่ 3.18 ตัวเลขทางซ้ายเป็นค่าตำแหน่งของหน่วยความจำสำหรับข้อมูลภายใน 89C52 ซึ่งแต่ละบิทในตำแหน่งนั้นจะมีค่าเป็นเลขฐาน 16 ที่จะใช้เป็นค่าอ้างอิงในคำสั่งจัดการกับข้อมูลบิทนั้น

3. Scratched Pod Area เป็นช่วงของหน่วยความจำตำแหน่ง 30H ถึง 7FH หน่วยความจำช่วงนี้จะใช้สำหรับเก็บข้อมูลทั่วไป ถ้ารีจิสเตอร์ Stack Pointer ชี้มายังหน่วยความจำ ช่วงนี้จะต้องระวังไม่ให้เกิดการเขียนทับของข้อมูลอันจะทำให้การทำงานของโปรแกรมผิดพลาดได้

ที่กล่าวมาแล้วคงจะพอเข้าใจถึงลักษณะการจัดการหน่วยความจำของ 89C52 ได้พอที่จะเริ่มเขียนโปรแกรมภาษาแอสเซมบลีของ 89C52 สรุปชุดคำสั่งของ 89C52 จะอยู่ในภาคผนวก ก

จากตารางคำสั่งในหน้า 12-13 ของภาคผนวก ก จะมีคำอธิบายการใช้งานหรือการทำงานของแต่ละคำสั่งใน 89C52 ไว้ด้วย คำสั่งของ 89C52 เป็นคำสั่งที่ประสิทธิภาพการทำงานสูงมาก ในขณะที่ 89C52 ทำงานจะมีรีจิสเตอร์ตัวหนึ่งที่เก็บสภาวะ (Flag) ที่เกิดขึ้นระหว่างการคำนวณ เช่น ตัวทด (Carry) หรือจะใช้เลือก BANK ของรีจิสเตอร์ภายใน 8052 ก็ได้ รีจิสเตอร์นี้คือ Program Status Word (PSW) มีขนาด 8 บิท แต่ละบิทจะใช้เก็บสถานะการทำงานต่าง ๆ ไว้ดังรูปที่ 3.19

(MSB) (LSB)

CY	AC	F0	RS1	RS0	OV	–	P

Symbol	Position	Name and Significance
CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary Carry flag. (For BCD operations).
F0	PSW.5	Flag 0 (Available to the user for general purposes).
RS1 RS0	PSW.4 PSW.3	Register bank select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).
OV	PSW.2	Overflow flag.
–	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the Accumulator, i.e., even parity.

Note :
The contents of (RS1, RS0) enable the working register banks as follows :

(0.0)—Bank 0 (00H–07H)
(0.1)—Bank 1 (08H–0FH)
(1.0)—Bank 2 (10H–17H)
(1.1)—Bank 3 (18H–1FH)

รูปที่ 3.19 Program Status Word (PSW)

PSW.0 บิท 0 เรียกว่าบิทพาริตี้บิทนี้จะบอกว่าในรีจิสเตอร์ Accumulator หรือรีจิสเตอร์ A มี 1 เป็นจำนวนคี่หรือคู่ เช่นในรีจิสเตอร์ A ขนาด 8 บิทมี 1 อยู่ 3 ตัวและมี 0 อยู่ 5 ตัวก็จะทำให้บิท PSW.0 นี้มีค่าเป็น 1 ถ้าใน Accumulator มี 1 อยู่เป็นจำนวนคู่ก็จะทำให้บิทนี้มีค่าเป็น 0

PSW.1 บิท 1 บิทนี้ไม่มีการใช้งาน

PSW.2 บิท 2 เรียกว่า Overflow Flag เป็นบิทที่บอกการคำนวณนั้นทำให้เกิดตัวทดขึ้นในระหว่างการคำนวณ ตัวทดนี้เป็นตัวทดที่เกิดจากบิท 6 ไปยังบิท 7 มีประโยชน์เมื่อทำการคำนวณแบบ Signed Integer

PSW.3 , PSW.4 บิท 3 และ 4 2 บิทนี้จะใช้งานร่วมกันเพื่อเป็นตัวบอกว่าขณะนี้ใช้รีจิสเตอร์ R0 ถึง R7 ใน BANK ได้ดังตาราง

บิท 4 (RB1)	บิท 3 (RB0)	Register bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

ตัวอย่างเช่น บิท 4 และบิท 3 มีค่าเป็น 10_2 เป็นการเลือกรีจิสเตอร์ BANK 2 หมายความว่าในรหัสคำสั่งช่วยจำที่อ้างอิงถึง R0 ก็จะอ้างถึงหน่วยความจำภายในที่ตำแหน่ง 10H

PSW.5 บิท 5 เรียกว่าบิทเอนกประสงค์เป็นบิทที่ผู้ใช้สามารถใช้คำสั่งกำหนดค่าให้เป็น 0 หรือ 1 ก็ได้ โดยที่การทำงานของคำสั่งอื่น จะไม่ทำให้บิทนี้มีค่าเปลี่ยนแปลง บิทนี้มีประโยชน์สำหรับในการส่งสถานะของโปรแกรมระหว่างเรียกการทำงานของโปรแกรมย่อย (Subroutine)

PSW.6 บิท 6 เรียกว่า Auxiliary Carry Flag เป็นบิทที่ใช้สำหรับเก็บตัวทดที่เกิดขึ้นระหว่างการคำนวณ โดยตัวทดนี้เป็นตัวทดที่เกิดการคำนวณของบิท 3 ข้ามไปยังบิท 4

PSW.7 บิท 7 เรียกว่า Carry Flag เป็นบิทที่บอกสถานะการคำนวณทางคณิตศาสตร์ว่าผลลัพธ์นั้นทำให้เกิดตัวทดขึ้นหรือไม่ เช่นการบวกเลข 2 จำนวนเข้าด้วยกันแล้วผลลัพธ์มีค่ามากกว่า 255 ก็จะทำให้เกิดตัวทดขึ้น เนื่องมาจากว่า Accumulator ที่ทำการบวกนี้สามารถเก็บข้อมูลได้เพียง 8 บิทเท่านั้น และทำให้บิทนี้มีค่าเป็น 1

3.6 พื้นที่หน่วยความจำข้อมูลแฝงของ AT89C52

เป็นพื้นที่ที่อยู่ซ้อนทับกับค่าตำแหน่งเดียวกับ หมายเลขรีจิสเตอร์ฟังชั่นพิเศษเช่นเดียวกับ ไมโครคอนโทรลเลอร์ 8052 การเข้าถึงจะสามารถเข้าถึงโดยอ้อม (indirect) มีขนาด 128 ไบต์ อยู่ที่ตำแหน่ง 80H - FFH

3.7 รีจิสเตอร์ฟังก์ชั่นพิเศษ (Special Function Register , SFR)

ใน 89C52 จะใช้วิธีการกำหนดชื่อให้กับตำแหน่งของหน่วยความจำ สำหรับข้อมูลภายใน (Internal Data Memory) ที่เรียกว่า Symbolize เช่น การให้ชื่อหน่วยจำแต่ละตำแหน่งในแต่ละ Bank ซึ่งอยู่ในช่วงหน่วยความจำตำแหน่ง 00H ถึง 1FH แล้วในคำสั่งจะอ้างอิงหน่วยความจำแต่ละตำแหน่งโดยการใช้ชื่อ R0 , R1 , R2 , R3 , R4 , R5 , R6 และ R7 หน่วยความจำตำแหน่งเหล่านี้ จะเรียกอีกอย่างหนึ่งว่าเป็นรีจิสเตอร์ ซึ่งมีหน้าที่ในการเก็บหรือพักข้อมูล หรือใช้สำหรับการกระทำบางอย่าง รีจิสเตอร์บางอย่างใน 89C52 ที่เรียกว่า Specail Function Register (SFR) เป็นรีจิสเตอร์ที่ใช้สำหรับงานเฉพาะ คือข้อมูลที่นำไปเก็บไว้รีจิสเตอร์เหล่านี้จะมีความหมายเฉพาะตัวของรีจิสเตอร์ ที่แต่ละตำแหน่งของ SFR อาจจะไม่ใช้เป็นหน่วยความจำ (RAM) แต่อาจเป็นตัวนับ (Register Counter) , Shift Register หรือ Latch ซึ่งการอ้างอิงข้อมูลในแต่ละตำแหน่งนั้น 89C52 จะถือเสมือนว่าเป็นหน่วยความจำตำแหน่งหนึ่ง จึงเรียกการมองข้อมูลแต่ละตำแหน่งนี้ว่า Memory Map I/O รีจิสเตอร์กลุ่มนี้มีดังในรูปที่ 3.20

Table 1

Symbol	Name	Address
*ACC	Accululator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 2 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H
*IOCON (1)	IO Control	0F8H

+ 80C52 and 83C154 only * bit addressable
(1) 83C154 only

รูปที่ 3.20 Special Function Register (SFR)

ในรูปที่ 3.20 ช่อง Symbol ทางซ้ายจะเป็นสัญลักษณ์ของรีจิสเตอร์ ในช่องถัดมาคือชื่อของรีจิสเตอร์ตามสัญลักษณ์ที่อยู่ทางซ้าย ในช่องขวาสุดจะเป็นตำแหน่งของหน่วยความจำสำหรับข้อมูลภาย

ใน 89C52 ที่แทนด้วยชื่อหรือสัญลักษณ์ทางซ้ายนั่นเอง เช่น ในบรรทัดแรกคือรีจิสเตอร์ชื่อ Accummulater ที่มีสัญลักษณ์ ACC รีจิสเตอร์นี้คือ หน่วยความจำสำหรับข้อมูลภายใน 89C52 ที่ตำแหน่ง 0E0H การอ่านหรือเขียนข้อมูลกับรีจิสเตอร์เหล่านี้ สามารถทำได้โดยการใช้คำสั่งในกลุ่มการย้ายข้อมูล (เช่น MOV A ,# 25H หรือ MOV 0E0H , #25H) และรีจิสเตอร์บางตัวในกลุ่มนี้ยังสามารถใช้คำสั่งในกลุ่ม Boolean Instruction เพื่อการทำงานในแต่ละบิทในรีจิสเตอร์เหล่านั้นได้ จากตารางในรูปที่ 3.20 รีจิสเตอร์ที่มีเครื่องหมาย * อยู่ข้างหน้าสามารถจะใช้คำสั่งในกลุ่ม Boolean Instruction จัดการกับแต่ละบิทได้ รีจิสเตอร์ที่มีเครื่องหมาย + นำหน้าหมายความว่า รีจิสเตอร์นั้นมีเฉพาะใน 80C52 และ 83C154 เท่านั้นไม่มีใน 89C52

Direct Byte Address	Bit Address (MSB)							(LSB)	Special Function Register Symbol
	WDT	T32	SERR	IDC	P3HZ	P2HZ	P1HZ	ALE	
0F8H	FF	FE	FD	FC	FB	FA	F9	F8	IOCON
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	CY	AC	F0	RS1	RS0	OV	F1	P	
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0CDH	Not Bit Addressable								TH2
0CCH	Not Bit Addressable								TL2
0CBH	Not Bit Addressable								RCAP2H
0CAH	Not Bit Addressable								RCAP2L
	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2	
0C8H	CF	CE	CD	CC	CB	CA	C9	C8	T2CON
	PCT		PT2	PS	PT1	PX1	PT0	PX0	
0B8H	BF	-	B0	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA		ET2	ES	ET1	EX1	ET0	EX0	
0A8H	AF	-	AD	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
99H	Not Bit Addressable								SBUF
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
8DH	Not Bit Addressable								TH1
8CH	Not Bit Addressable								TH0
8BH	Not Bit Addressable								TL1
8AH	Not Bit Addressable								TL0
89H	Not Bit Addressable								TMOD
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
87H	Not Bit Addressable								PCON
83H	Not Bit Addressable								DPH
82H	Not Bit Addressable								DPL
81H	Not Bit Addressable								SP
80H	87	86	85	84	83	82	81	80	P0

รูปที่ 3.21 แผนภาพค่าตำแหน่งหน่วยความจำแต่ละบิท

รูปที่ 3.21 ในช่องสี่เหลี่ยมเล็ก ๆ จะเป็นตำแหน่งของบิทนั้นในแต่ละรีจิสเตอร์ เช่น ในช่องซ้ายของรีจิสเตอร์ TCON มีค่า ซึ่งเป็นตำแหน่งค่าบิท 7 ของหน่วยความจำ 88H ถ้าต้องการให้บิทนี้มีค่าเป็น 0 ก็สามารถทำได้โดยใช้คำสั่ง

CLR 8FH

หรือจะทำให้เป็น 1ก็ทำได้โดยคำสั่ง

SETB 8FH

Special Function Register

รีจิสเตอร์ในกลุ่ม Special.Function Register มีดังนี้

1. Accumulator ตำแหน่งหน่วยความจำภายในเท่ากับ 0E0H

รีจิสเตอร์นี้มีขนาด 8 บิท เป็นรีจิสเตอร์ที่ใช้มาก ซึ่งในรหัสคำสั่งช่วยจำจะอ้างถึงรีจิสเตอร์นี้โดยใช้สัญลักษณ์ A เช่น MOV A , #15H คำสั่งที่จะอ่านหรือเก็บข้อมูลกับหน่วยความจำภายนอกจะต้องกระทำผ่านรีจิสเตอร์นี้เท่านั้น เช่น MOVX @R0 , A หรือ MOVX A , @R0 เป็นต้น และข้อมูลที่อยู่ภายในรีจิสเตอร์นี้ ก็สามารถที่จะให้โปรแกรมตรวจสอบเพื่อกระโดดการทำงานไปยังตำแหน่งอื่นได้ เช่น JZ rel

2. B Register ตำแหน่งหน่วยความจำภายในเท่ากับ 0F0H

เป็นรีจิสเตอร์ขนาด 8 บิท ที่ใช้ในคำสั่งการคูณ (MUL AB) และคำสั่งการหาร (DIV AB) เท่านั้น โดยรีจิสเตอร์ B นี้จะเก็บตัวคูณและผลลัพธ์บิท 8 ถึง 15 ในคำสั่งการคูณ ส่วนในคำสั่งการหารนั้นรีจิสเตอร์ B จะเก็บตัวหารและผลการหาร การเขียนข้อมูลไปยังรีจิสเตอร์นี้จะต้องใช้คำสั่งเคลื่อนย้ายข้อมูลไปยังตำแหน่ง 0F0H เช่น MOV 0F0H , 25H จะเป็นการกำหนดค่า 25H ให้กับรีจิสเตอร์ B

3. Program status word ตำแหน่งหน่วยความจำภายในเท่ากับ 0D0H

เป็นรีจิสเตอร์ขนาด 8 บิท ที่แต่ละบิทจะบอกสภาวะต่าง ๆ แต่ละบิทของ PSW จะสามารถกำหนดให้เป็น 1 หรือ 0 ได้ด้วยคำสั่ง SETB หรือ CLR ตามลำดับค่าตำแหน่งบิท 0 ถึงบิท 7 ของรีจิสเตอร์ PSW เท่ากับ D0H ถึง D7H ตามลำดับ

4. Stack Pointer ตำแหน่งหน่วยความจำในเท่ากับ 081H

เป็นรีจิสเตอร์ขนาด 8 บิท รีจิสเตอร์นี้จะใช้ตำแหน่งหน่วยความจำภายใน 89C52 ที่ใช้เก็บตำแหน่ง (Address) เดิมของโปรแกรมก่อนทำงานคำสั่ง CALL หรือตำแหน่งที่จะใช้เก็บข้อมูลด้วยคำ

สั่ง PUSH และ ตำแหน่งที่จะอ่านข้อมูลออกมาในคำสั่ง POP เมื่อทำการรีเซ็ท 89C52 โดยการป้อนสัญญาณสภาวะลอจิก 1 เข้าไปที่ขา RST ของ 89C52 จะทำให้ข้อมูลในรีจิสเตอร์นี้มีค่าเป็น 07H หมายความว่ารีจิสเตอร์ SP ชี้หน่วยความจำภายใน 89C52 ที่ตำแหน่ง 07H ค่าของ SP จะเปลี่ยนแปลงไปโดยการใช้คำสั่งเคลื่อนย้ายข้อมูลหรือการทำงานของคำสั่ง PUSH , POP และ CALL

5. Data Pointer Register ตำแหน่งหน่วยความจำภายในเท่ากับ 82H และ 83H

รีจิสเตอร์ DTPR มีขนาด 16 บิท หน้าที่ของรีจิสเตอร์นี้ก็คือใช้สำหรับชี้ตำแหน่งในหน่วยความจำรีจิสเตอร์ DPTR นี้สามารถใช้อ้างถึงตำแหน่งหน่วยความจำได้สูงสุด 60×1024 ตำแหน่ง เช่นคำสั่ง MOVX A , @DPTR หรือ ใช้ชี้ตำแหน่งโปรแกรมที่ต้องการกระโดดข้ามไปทำงาน เช่นคำสั่ง JMP @A+DPTR รีจิสเตอร์ DPTR นี้ประกอบด้วยรีจิสเตอร์ขนาด 8 บิท 2 ตัวคือ DPH ซึ่งอยู่ที่ตำแหน่ง 83H และ DPL ซึ่งอยู่ตำแหน่ง 82H ในหน่วยความจำสำหรับข้อมูลภายใน 89C52 ดังนั้นการแก้ไขข้อมูลในรีจิสเตอร์ DPTR จึงทำได้ทั้งทีละ 16 บิท เช่น คำสั่ง MOV DPTR , #DATA16 หรือจัดการทีละ 8 บิทโดยการแก้ไขข้อมูลใน DPH หรือ DPL ด้วยคำสั่ง MOV 83H , #DATA8 หรือ MOV 83H , #DATA8

6. PORT 0 ถึง 3 ตำแหน่งหน่วยความจำภายในเท่ากับ 80H, 90H, 0A0H, 0B0H

Special Function Register ชื่อ P0 , P1 , P2 และ P3 เป็นรีจิสเตอร์ขนาด 8 บิท ของหน่วยความจำสำหรับข้อมูลภายใน 89C52 ที่ตำแหน่งเป็นการส่งข้อมูลไปยังพอร์ทนั้น ๆ ของ 89C52 ข้อมูลที่เขียนออกไปจะถูก LATCH ค้างไว้และปรากฏที่แต่ละบิทของพอร์ท เช่น MOV 80H , #18H จะปรากฏสภาวะลอจิก LLLHHLLL ที่ขาบิท 7 ถึง 0 ของพอร์ท 0 ตามลำดับ ในการอ่านข้อมูลจากรีจิสเตอร์แต่ละตัวก็จะเป็นการอ่านสภาวะลอจิกของสัญญาณที่ปรากฏอยู่ที่แต่ละขาของพอร์ทนั้นเช่น MOV A , 80H เป็นการอ่านสภาวะลอจิกจากพอร์ท 0 เข้ามายัง Accumulator การอ่านข้อมูลจากพอร์ทจะต้องเขียนข้อมูล 11111111B ไปไว้ที่พอร์ทนั้น ๆ เสียก่อน ทุกบิทของพอร์ท 0 ถึง 3 จะสามารถแก้ไขเปลี่ยนแปลงได้โดยใช้คำสั่ง SETB bit และ CLR bit

7. Serial Data Buffer ตำแหน่งหน่วยความจำภายในเท่ากับ 99H

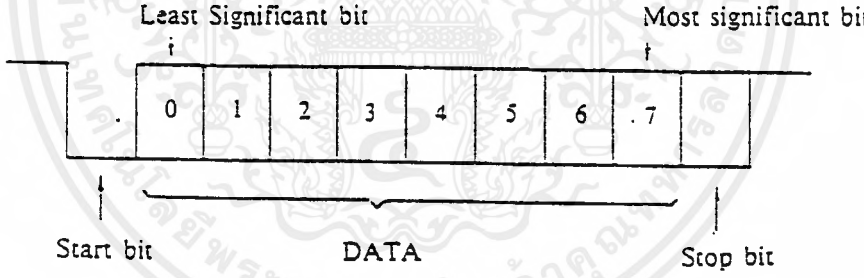
รีจิสเตอร์นี้มีขนาด 8 บิทและมีตำแหน่งของหน่วยความจำสำหรับข้อมูลภายใน 89C52 เท่ากับ 99H โครงสร้างภายในแล้วรีจิสเตอร์นี้มี 2 ตัว ที่มีชื่อเดียวกัน ตัวหนึ่งสำหรับเก็บข้อมูลที่จะส่งแบบอนุกรมออกจาก 89C52 และอีกตัวหนึ่งสำหรับข้อมูลแบบอนุกรมที่เข้ามา ดังนั้น Serial Port ของ 89C52 จึงเรียกว่ามีการทำงานแบบ Full Duplex เพราะสามารถส่งและรับข้อมูลได้ในเวลาเดียวกัน เนื่องจากรีจิสเตอร์สำหรับส่งและรับแยกออกจากกัน ข้อมูลที่ต้องการจะส่งออกก็ให้เขียนไปยังรีจิส-

เตอร์ SBUF แล้วสั่งงานให้ส่งข้อมูลออกมา ข้อมูลในรีจิสเตอร์จะเริ่มส่งออกโดยเริ่มจากบิท 0 ถึง 7 ตามลำดับ ถ้าข้อมูลมีข้อมูลเข้ามาทางขา RXD ก็จะถูกเก็บไปไว้ในรีจิสเตอร์นี้โดยถือว่าข้อมูลบิทแรกที่เข้ามาคือบิท 0

Serial Port จะสามารถกำหนดให้การทำงานรับ-ส่งข้อมูลแบบอนุกรมได้ 4 โมด (MODE) โดยการกำหนดในรีจิสเตอร์ SCON (Serial Port Control Register) ซึ่งจะอธิบายต่อไปในข้อ 5.18 แต่ละโมดการทำงานของ Serial Port มีดังนี้

MODE 0 : ในโมดนี้ จะมีการับหรือส่งข้อมูลแบบอนุกรมทางขา RXD และ ขา TXD จะส่งสัญญาณ Clock ที่ใช้สำหรับเลื่อน (Shift) ข้อมูล 1 ชุด ของข้อมูลจะประกอบด้วยข้อมูล 8 บิทเท่านั้น และจะเริ่มการรับ-ส่งข้อมูลจากบิท 0 จนถึงบิท 7 ตามลำดับ อัตราการส่งข้อมูลแบบอนุกรมจะเท่ากับ 1/12 เท่าของความถี่สัญญาณนาฬิกาที่ใช้กับ 89C52

MODE 1 : ข้อมูลที่รับ-ส่ง 1 ชุดในโมดนี้จะมี 10 บิท ผ่านทางขา RXD และ TXD ตามลำดับ เริ่มต้นการรับส่งข้อมูลด้วย Start bit 1 บิท (ลอจิกเป็น 0) , ข้อมูล 8 บิท (เริ่มจากบิท 0) , Stop bit 1 บิท (ลอจิก 0) การส่งข้อมูลโมดนี้มีดังรูปที่ 3.22

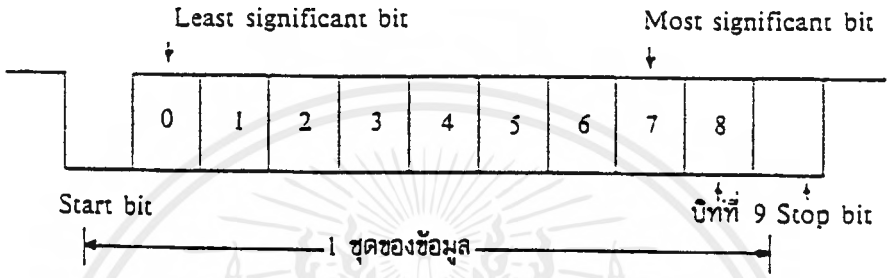


รูปที่ 3.22 ชุดข้อมูลอนุกรมในโมด 1

เมื่อรับข้อมูลอนุกรมเข้ามาข้อมูล 8 บิทจะถูกเก็บในรีจิสเตอร์ SBUF และ Stop bit จะถูกเก็บไปที่บิท RB8 ในรีจิสเตอร์ SCON ในการส่งข้อมูลออกก็จะเขียนข้อมูลที่ต้องการส่งไปยังรีจิสเตอร์ SBUF อัตราการส่งข้อมูลในโมดนี้ สามารถกำหนดได้ตามต้องการ โดยจะขึ้นกับการเกิด Overflow ใน Timer 1

MODE 2 : การรับ-ส่งข้อมูลของโมด 2 1 ชุดจะมี 11 บิท ข้อมูลจะส่งออกผ่านทางขา TXD และรับเข้ามาทางขา RXD ข้อมูลแต่ละชุดจะเริ่มต้นด้วย Start bit 1 บิท , ข้อมูล 8 บิท (เริ่มจาก

บิท 0) , ข้อมูลบิทที่ 9 จำนวน 1 บิท และ Stop bit อีก 1 บิทข้อมูลบิทที่ 9 ที่จะส่งออกนี้สามารถกำหนดได้ว่าจะให้เป็น 1 หรือ 0 พาริตี้ของข้อมูลไปเป็นบิทที่ 9 เพื่อว่าปลายทางรับข้อมูลแล้วจะได้ใช้ตรวจสอบว่าข้อมูลที่รับเข้ามา 8 บิทมีพาริตี้บิทตรงกับบิทที่ 9 หรือไม่ ถ้าไม่ตรงก็แสดงว่ามีข้อผิดพลาดเกิดขึ้นระหว่างการส่งข้อมูลเข้ามานั้นข้อมูลบิทที่ 9 ก็จะถูกนำไปเก็บในบิท RB8 ของรีจิสเตอร์ SCON ชุดข้อมูลที่รับ-ส่งจะมีดังรูปที่ 3.23



รูปที่ 3.23 ชุดข้อมูลอนุกรมในโมด 2

อัตราการส่งข้อมูลจะกำหนดให้เป็น 1/32 หรือ 1/64 เท่า ของความถี่สัญญาณนาฬิกาที่ใช้กับ 89C52 โดยการกำหนดบิท SMOD ในรีจิสเตอร์ PCON

MODE 3 : การส่งข้อมูลโมดนี้ 1 ชุดมี 11 บิท เหมือนกับโมด 2 ทุกประการแตกต่างกันตรงอัตราการส่งข้อมูลเท่านั้น คืออัตราการส่งข้อมูลในโมด 3 นี้ สามารถกำหนดได้ตามต้องการโดยจะขึ้นกับการเกิด Overflow ใน Timer 1 เหมือนกับโมด 1

8. SCON (Serial Port Control Register) ตำแหน่งหน่วยความจำภายในเท่ากับ 98H

รีจีสเตอร์ SCON มีขนาด 8 บิท ใช้สำหรับควบคุมการส่งและรับข้อมูลผ่านทาง Serial Port แต่ละบิทของข้อมูลในรีจิสเตอร์นี้มีความหมายเฉพาะดังรูปที่ 3.24

SCON : SERIAL PORT CONTROL REGISTER, BIT ADDRESSABLE.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial Port mode specifier.(NOTE 1).
SM1	SCON.6	Serial Port mode specifier. (NOTE 1).
SM2	SCON.5	Enables the multiprocessor communication feature in mode 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0. (See Table 9).
REN	SCON.4	Set/Cleared by software to Enable/Disable reception.
TB8	SCON.3	The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.
RB8	SCON.2	In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.

NOTE 1 :

SM0	SM1	Mode	Description	Baud Rate
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8-Bit UART	Variable
1	0	2	9-Bit UART	Fosc./64 OR Fosc./32
1	1	3	9-Bit UART	Variable

SERIAL PORT SET-UP : Table 9

MODE	SCON	SM2 VARIATION
0	10H	Single Processor Environment (SM2 = 0)
1	50H	
2	90H	
3	D0H	
0	NA	Multiprocessor Environment (SM2 = 1)
1	70H	
2	B0H	
3	F0H	

รูปที่ 3.24 Serial Port Control Register (SCON)

ในรูปที่ 3.24 บิท R1 จะเป็นชื่อของบิท 0 และ SNO จะเป็นบิท 7 ของรีจิสเตอร์ SCON ซึ่งความหมายหรือการทำงานของแต่ละบิทมีดังนี้

RI Receive Interrupt Flag

บิทนี้จะถูกกำหนดโดยฮาร์ดแวร์ให้มีค่าเป็น 0 หรือ 1 โดยที่ในการรับข้อมูลโมด 0 นั้น บิท RB8 จะมีค่าเป็น 1 เมื่อมีข้อมูลเข้ามาครบทั้ง 8 บิท ส่วนในโมดอื่นบิท RB8 จะเป็น 1 ก็ต่อเมื่อข้อมูลเข้ามาถึงเวลาครึ่งหนึ่งของ Stop bit (ยกเว้นบางกรณีให้ดูที่เรื่องบิท SM2 ของรีจิสเตอร์ SCON) บิทนี้จะสามารถ Clear ให้มีค่าเป็น 0 ได้ โดยใช้คำสั่ง CLR bit โดยค่าตำแหน่งของบิทมีค่าเท่ากับ 98H บิทนี้มีประโยชน์ให้รู้ว่าข้อมูลได้เข้ามาอยู่ใน SBUF ครบทั้งชุดแล้วพร้อมที่ CPU จะอ่านไปเก็บใน

นับจำนวนไซเคิลของสัญญาณที่เข้ามาทางขา T0 หรือ T1 ของ 89C52 สัญญาณที่เข้ามาทางขาลอจิก 0 หรือ 1 เป็นแบบ TTL คือลอจิก 0 จะต้องมีโวลเตจไม่เกิน 0.6 โวลท์ และลิจิก 1 จะต้องมีโวลเตจมากกว่า 2.4 โวลท์

10. TMOD Timer/ Counter mode register

ตำแหน่งหน่วยความจำภายในเท่ากับ 89H

TMOD เป็นรีจิสเตอร์ขนาด 8 บิทที่มีหน้าที่ควบคุมการทำงานของ Timer 0 และ Timer1 แต่ละบิทในรีจิสเตอร์นี้มีความหมายเฉพาะดังรูปที่ 3.25

GATE1	C/T	M1	M0	GATE0	C/T	M1	M0
	TIMER 1				TIMER 0		

GATE — When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

C/T — Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

M1 — Mode selector bit. (NOTE 1)

M0 — Mode selector bit. (NOTE 1)

NOTE 1 :

M1	M0		Operating Mode
0	0	0	13-bit Timer
0	1	1	16-bit Timer/Counter
1	0	2	8-bit Auto-Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

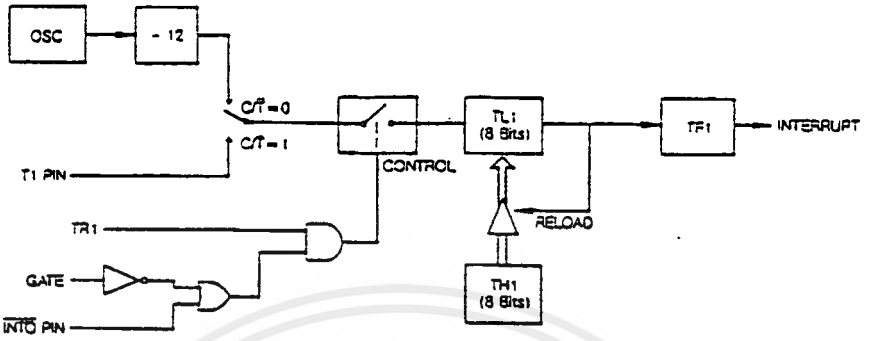
รูปที่ 3.25 TMOD Timer/ Counter mode register

ในรูปที่ 3.25 M0 เป็นชื่อของบิท 0 และ GATE ทางซ้ายสุดเป็นชื่อของบิท 7 รีจิสเตอร์นี้แบ่งข้อมูลออกเป็น 2 ชุด ชุดละ 4 บิท คือบิท 0-3 ใช้สำหรับควบคุมการทำงานของ Timer 0 และ บิท 4-7 ใช้ควบคุมการทำงานของ Timer 1 หน้าที่ในการควบคุม Timer ของแต่ละบิทที่มีชื่อเดียวกันจะเหมือนกัน

GATEx เป็นบิทที่ใช้ควบคุมให้ Timer x ทำงานหรือไม่ ถ้าบิทนี้ของ Timer x ถูกตั้งเป็น 1 จะทำให้ Timer ทำงานก็ต่อเมื่อที่ขา INTx มีสภาวะลอจิกเป็น 1 และบิท TRx ในรีจิสเตอร์ TCON เป็น 1 ด้วย

C/$\overline{T}$ บิทนี้ใช้สำหรับเลือกการทำงานของ Timer ว่าจะใช้เป็น Timer หรือ Counter ถ้าบิทนี้เป็น 1 ก็หมายความว่าเลือกการทำงานเป็น Counter ซึ่งจะนับจำนวนไซเคิลของสัญญาณที่เข้ามาทางขา Tx

โมด 2



รูปที่ 3.27 Timer mode 2

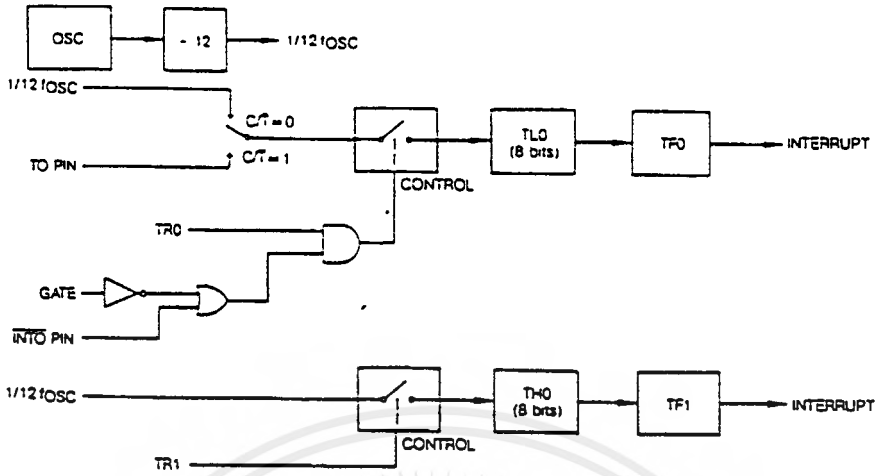
ในรูปที่ 3.27 เป็นไดอะแกรมของวงจร Timer 1 ใน 89C52 ที่ทำงานโมด 2 Timer 0 และ Timer 1 มีการทำงานในโมด 2 เหมือนกันโดยจะสามารถกำหนดให้ทำหน้าที่เป็น Timer หรือ Counter ได้โดยบิท C/T และควบคุมการนับได้โดยข้อมูลในบิท TR1 และ GATE ในรีจิสเตอร์ TMOD กับสัญญาณที่เข้า INTx เมื่อเริ่มการทำงานข้อมูลในรีจิสเตอร์ TH1 จะถูกโหลด (Load) ไปยังรีจิสเตอร์ TL1 ทำให้รีจิสเตอร์ TH1 และ TL1 มีค่าเหมือนกัน เมื่อเกิดการนับจำนวนไซเคิลของสัญญาณที่ออกจากสวิตช์ Control จะทำให้ค่าจากการนับในรีจิสเตอร์ TL1 เพิ่มขึ้นไปเรื่อย ๆ ทีละ 1 จนถึง 0FFH ในการนับครั้งต่อไปจะทำให้บิท TF1 ในรีจิสเตอร์ TCON ไม่เป็น 1 และข้อมูลในรีจิสเตอร์ TH1 จะถูกโหลดไปยังรีจิสเตอร์ TL1 เพื่อเป็นค่าเริ่มต้นการนับต่อไป

โมด 3

การทำงานโมด 3 ของ Timer 0 และ 1 ต่างกัน

Timer 1 ในโมด 3 จะไม่ทำงาน

Timer 0 ในโมด 3 จะทำงานเป็นตัวนับที่เสมือนมีตัวนับ 8 บิทอยู่ 2 ตัว คือ TL0 และ TH0 ทำงานแยกกันดังรูปที่ 3.28



รูปที่ 3.28 Timer 0 mode 3

รีจิสเตอร์ TL0 จะเป็นตัวนับ 8 บิท ที่มีการนับสัญญาณจากออสซิลเลเตอร์หารด้วย 12 หรือนับสัญญาณที่เข้ามาทางขา T0 ขึ้นกับบิท $C/\overline{T}$ ในรีจิสเตอร์ TMOD และการนับจะควบคุมโดยบิท TR0 และ GATE ในรีจิสเตอร์ TMOD กับสภาวะลอจิกของสัญญาณที่ขา INT0 เหมือนกับในการทำงานโมด 0 , 1 และ 2 แต่ค่าจากการนับนี้สูงสุดจะเพียง 255 เท่านั้น เมื่อค่าการนับเปลี่ยนจาก 0FFH เป็น 00H คือเกิดการ overflow จะทำให้บิท TF0 ถูก SET เป็น 1 และอาจเกิดการขัดจังหวะ(Interrupt) การทำงานของโปรแกรมได้ถ้ามีการกำหนดค่าในรีจิสเตอร์ IE และ IP

ตัวนับอีกตัวคือ รีจิสเตอร์ TH0 จะทำงานในโมดของ Timer เท่านั้น คือจะนับจำนวนไซเคิลของสัญญาณที่ออกจากออสซิลเลเตอร์แล้วหารด้วย 12 การนับจะควบคุมได้ด้วยบิท TR1 ในรีจิสเตอร์ TMOD ถ้าบิทนี้เป็น 1 ก็จะมีสัญญาณเข้าไปยัง TH0 แต่ถ้าบิทนี้เป็น 0 ก็จะไม่มีสัญญาณเข้าไปยัง TH0

11. TCON Timer Control Regster ตำแหน่งหน่วยความจำภายในเท่ากับ 088H

รีจิสเตอร์ ขนาด 8 บิทนี้ใช้ควบคุมการทำงาน และบอกสภาวะของ Timer 0 และ Timer 1 แต่บิทของรีจิสเตอร์นี้ จะทำงานต่างกันดังรูปที่ 3.29

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn Timer/Counter 1 ON/OFF.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
IE1	TCON.3	External interrupt 1 edge flag. Set by hardware when External interrupt edge is detected. Cleared by hardware when interrupt is processed.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered External interrupt.
IE0	TCON.1	External interrupt 0 edge flag. Set by hardware when External interrupt edge detected. Cleared by hardware when interrupt is processed.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External interrupt.

รูปที่ 3.29 TCON Timer Control Regster

ในรูปที่ 3.29 T0 เป็นชื่อของบิท 0 และ TF เป็นชื่อของบิท 7 ในรีจิสเตอร์ TCON แต่ละบิทมีหน้าที่การทำงานดังนี้

IT0 Interrupt 0 เป็นบิทที่จะใช้กำหนดวิธีการขัดจังหวะโปรแกรม อันเนื่องจากสภาวะของสัญญาณที่เข้ามาทางขา $\overline{INT0}$

ถ้า IT0 เป็น 1 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา $\overline{INT0}$ เปลี่ยนจาก 1 เป็น 0

ถ้า IT0 เป็น 0 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา $\overline{INT0}$ เป็น 0

IE0 บิทนี้จะเป็น 1 ถ้าสัญญาณที่เข้ามาทางขา INT0 มีสภาวะลอจิกของสัญญาณ ตามที่กำหนดในบิท IT0 แล้วทำให้เกิดการขัดจังหวะโปรแกรม เมื่อเกิดการกระโดด ไปทำงานยังโปรแกรมตอบสนองการขัดจังหวะแล้ว จะทำให้บิทนี้กลับเป็น 0

IT1 Interrupt 1 เป็นบิทที่จะใช้กำหนดวิธีการขัดจังหวะโปรแกรม อันเนื่องจากสภาวะของสัญญาณที่เข้ามาทางขา $\overline{INT1}$

ถ้า IT1 เป็น 1 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา $\overline{INT1}$ เปลี่ยนจาก 1 เป็น 0

ถ้า IT1 เป็น 0 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา $\overline{INT1}$ เป็น 0

IE0 บิทนี้จะเป็น 1 ถ้าสัญญาณที่เข้ามาทางขา $\overline{INT1}$ มีสภาวะลอจิกของสัญญาณ ตามที่กำหนดในบิท IT1 แล้วทำให้เกิดการขัดจังหวะโปรแกรม เหมือนกับ IT0 ที่ทำงานกับสัญญาณ $\overline{INT0}$

TR0 Timer 0 Run Control Bit บิทนี้ถ้าเป็น 0 Timer 0 ไม่ทำการนับสัญญาณไม่ว่ากรณีใด ๆ ทั้งสิ้น แต่ถ้าบิทนี้เป็น 1 จะทำให้ Timer 0 ทำงานโดยขึ้นกับสัญญาณ GATE , $\overline{INT0}$ ข้อมูลในบิทนี้จะสามารถ Set เป็น 1 หรือ Clear เป็น 0

TF0 Timer 0 Overflow flag บิทนี้จะเป็น 1 เมื่อการนับของรีจิสเตอร์ใน Timer 0 (TL0 หรือ TH0 ขึ้นอยู่กับโมดของการทำงาน) เกิด Overflow ขึ้น คือเอาการนับเพิ่มไปจนถึงค่าสูงสุดแล้วนับต่อไปทำให้ค่าการนับกลับมาเริ่มต้นใหม่ที่ 0 หรือค่า Reload เมื่อ 89C52 กระโดดไปทำงานที่โปรแกรมตอบสนองการขัดจังหวะทำให้บิทนี้กลับเป็น 0

TR1 Timer 1 Run Control Bit การทำงานจะเหมือนกับการทำงานของบิท TR0 แต่บิทนี้จะทำงานกับ Timer 1

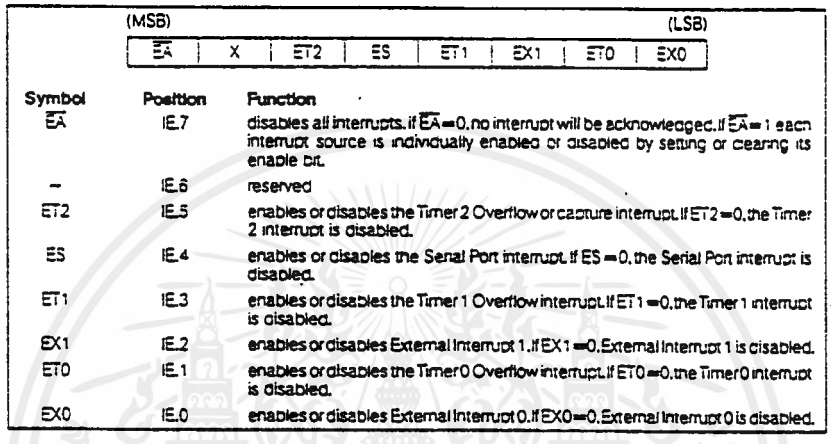
TF1 Timer 1 Overflow flag บิทนี้เหมือนกับบิท TF0 ต่างกันที่ขึ้นกับการทำงานของ Timer1

4 บิทแรกที่กล่าวมานั้นจะเกี่ยวข้องกับการขัดจังหวะ (Interrupt) ส่วน 4 บิทหลังนั้นได้กล่าวมาแล้วอย่างละเอียดในเรื่องโมดการทำงานของ Timer

ในขณะที่ Timer ทำงานในโมดของ Timer นั้น รีจิสเตอร์ที่ทำหน้าที่เป็นตัวนับจะมีค่าเพิ่มขึ้น 1 ทุก ๆ 1 ไซเคิลของเครื่อง ซึ่งเท่ากับ 12 คาบของสัญญาณจากออสซิลเลเตอร์ ในกรณีที่ Timer ทำงานเป็น Counter เพื่อนับจำนวนไซเคิลของสัญญาณที่เข้ามาทางขา T0 หรือ T1 รีจิสเตอร์จะเพิ่มค่าไป 1 เมื่อมีการเปลี่ยนสถานะของสัญญาณที่ขาดังกล่าวจาก 1 เป็น 0 โดยวงจรภายใน 89C52 จะตรวจสอบสถานะของสัญญาณที่ขาดังกล่าวในช่วงเวลาเฟส 2 ของ State5 (S5P2) ในทุก ๆ 1 ไซเคิลของเครื่อง เช่นในเวลา S5P2 ครั้งหนึ่งพบว่าสัญญาณที่ขา T0 มีสภาวะลอจิกเป็น 1 และในเวลา S5P2 ของไซเคิลของเครื่องถัดมาพบว่าสัญญาณที่ขา T0 มีสภาวะลอจิกเป็น 0 ก็จะทำให้ค่าในรีจิสเตอร์ตัวนับเพิ่มค่าไป 1 แต่ละไซเคิลของเครื่องจะเท่ากับ 12 ไซเคิลของสัญญาณออสซิลเลเตอร์ ดังนั้นสัญญาณที่จะนับได้จะต้องเป็น 1 อย่างน้อยให้ถูกจับได้ใน 1 ไซเคิลของเครื่องและเป็น 0 อย่างน้อย ก็ต้องให้ถูกตรวจสอบได้

TF0 , TF1 เป็น 1 ซึ่งสัญญาณจาก 2 บิทนี้จะสามารถทำให้เกิดการขัดจังหวะได้เช่นกัน ดังเช่นสัญญาณขัดจังหวะที่ 2 และ 4 ในรูปที่ 3.30

แหล่งกำเนิดสัญญาณทั้ง 6 ที่สามารถทำให้เกิดการขัดจังหวะได้ 5 แบบนี้ ผู้ใช้สามารถกำหนดให้สัญญาณใดบ้างเกิดการขัดจังหวะเรียกว่า Enable หรือไม่ให้เกิดการขัดจังหวะเรียกว่า Disable โดยการกำหนดในรีจีสเตอร์ IE (Interrupt Enable Register) ซึ่งมี 8 บิท แต่ละบิทสามารถ Enable ให้ขัดจังหวะได้จากแต่ละสัญญาณ ดังรูปที่ 3.31



(MSB) (LSB)

$\overline{EA}$	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
$\overline{EA}$	IE.7	disables all interrupts. If $\overline{EA}$=0, no interrupt will be acknowledged. If $\overline{EA}$=1 each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
–	IE.6	reserved
ET2	IE.5	enables or disables the Timer 2 Overflow or capture interrupt. If ET2=0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES=0, the Serial Port interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1=0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External Interrupt 1. If EX1=0, External Interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0=0, the Timer 0 interrupt is disabled.
EX0	IE.0	enables or disables External Interrupt 0. If EX0=0, External Interrupt 0 is disabled.

รูปที่ 3.31 Interrupt Enable Register

ถ้าต้องการ Enable บิทใดก็ให้โปรแกรมกำหนดค่าในบิทนั้นเป็น 1 ถ้าค่าในบิทนั้นเป็น 0 หมายถึง Disable การ Disable จะทำให้มีการขัดจังหวะการทำงานของโปรแกรมเนื่องจากสัญญาณขอขัดจังหวะนั้น ๆ EX0 เป็นชื่อบิท 0 และ EA เป็นชื่อของบิท 7

EX0 บิทนี้ใช้สำหรับการ Enable สัญญาณที่เข้ามาทางขา INTO ให้เกิดการขัดจังหวะ หรือไม่

ET0 Timer 0 Interrupt Enable Bit ข้อมูลบิทนี้จะใช้ Enable Disable สัญญาณขัดจังหวะที่มาจากวงจร Timer 0 (TF0)

EX1 บิทนี้จะใช้ Enable หรือ Disable สัญญาณที่เข้ามาทางขา INT1 ให้เกิดการขัดจังหวะหรือไม่

ET1 Timer 1 Interrupt Enable Bit บิทนี้จะใช้ Enable หรือ Disable สัญญาณขัดจังหวะจาก Timer 1 (TF1)

ES ข้อมูลในบิทนี้จะ Disable หรือ Enable การขัดจังหวะจาก Serial Port อันเนื่องจากมีข้อมูลเข้ามายัง SBUF หรือข้อมูลจาก SBUF ได้ส่งออกไปทาง Serial Port หมดแล้ว

ET2 Timer 2 Internal Enable Bit จะใช้เฉพาะใน 8052 และ 83152 เท่านั้น บิทนี้จะใช้ Enable หรือ Disable สัญญาณขอขัดจังหวะที่มาจาก Timer 2 (สัญญาณที่ 6 ในรูปที่ 3.30)

EA บิทนี้จะควบคุมทั้ง 6 บิทที่กล่าวมาแล้ว ถ้าข้อมูลในบิทนี้เป็น 0 จะเป็นการ Disable ทุกบิทที่กล่าวมาแล้ว ทำให้ไม่เกิดการขัดจังหวะโปรกแรมได้เลย แต่ถ้าบิทนี้ปืน 1 การ Enable/Disable ใน 6 บิทที่กล่าวมาแล้วจะขึ้นกับข้อมูล รีจิสเตอร์นั้นคือ IP Interrupt Priority Register

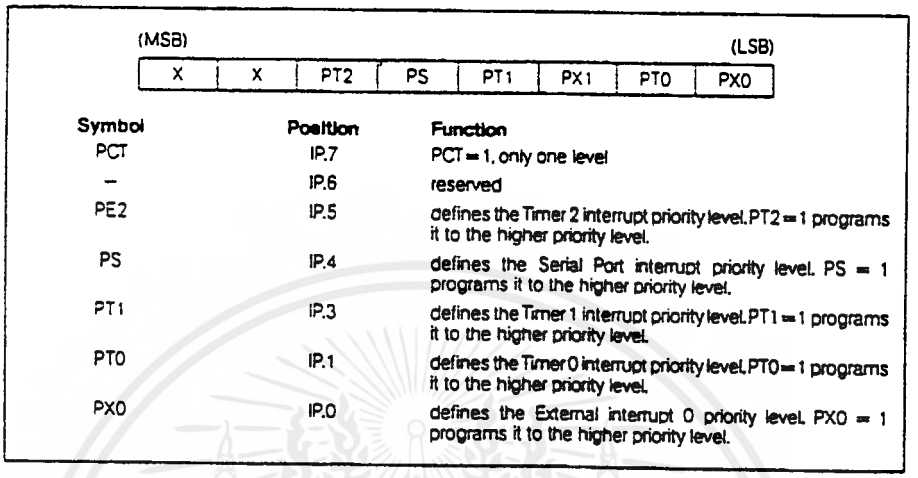
การกำหนดให้บิทใด Enable หรือ Diasble นั้นจะเป็นไปโดยอิสระไม่ขึ้นแก่กัน จึงสามารถกำหนดให้บิทใดหรือมากกว่า 1 บิท Enable ก็ได้ ดังนั้น 89C52 จึงมี Register อีกตัวที่ใช้เลือกว่าถ้ามีสัญญาณขอการขัดจังหวะโปรแกรมเข้ามาพร้อมกันมากกว่า 1 แล้วจะทำให้โปรแกรมตอบสนองการขัดจังหวะอันใดก่อน รีจิสเตอร์นั้น คือ IP Interrupt Priority Register

13. IP Interrupt Priority register ตำแหน่งหน่วยความจำภายในเท่ากับ 0B8H

ในการตอบสนองต่อสัญญาณขัดจังหวะของ 89C52 นั้น ถ้าสัญญาณขัดจังหวะทั้งหมดเข้ามาพร้อมกัน 89C52 จะต้องเลือกทำงานโปรแกรมตอบสนองการขัดจังหวะโดยการตรวจสอบสัญญาณเรียงตามลำดับ ซึ่งเรียกว่าวิธีการ Polling สัญญาณจังหวะหนึ่งจะถูกตรวจสอบก่อนแล้วสัญญาณอื่น ๆ จะถูกตรวจสอบต่อมา ถ้าสัญญาณนั้นขอขัดจังหวะ 89C52 จะสร้างคำสั่ง CALL เป็นพิเศษขึ้นมาเพื่อไปทำงานโปรแกรมตอบสองการขัดจังหวะของสัญญาณนั้น เมื่อเสร็จสิ้นแล้วจะกลับมาทำงานในโปรแกรมเดิมก่อนการขัดจังหวะ ทำให้เสมือนว่าสัญญาณแต่ละสัญญาณมีลำดับความสำคัญไม่เท่ากัน สัญญาณขัดจังหวะจะมีลำดับความสำคัญดังนี้ โดยเรียงจากลำดับความสำคัญสูงสุดถึงต่ำสุด

1. IE0
2. TF0
3. IE1
4. TF1
5. RI + TI

แต่ถ้าในการใช้งานบางครั้งจำเป็นที่จะต้องให้สัญญาณใดสัญญาณหนึ่งมีลำดับความสำคัญสูงสุด (Highest Priority) เพื่อจะทำงานโปรแกรมตอบสนองการขัดจังหวะได้ก่อนการขัดจังหวะของสัญญาณอื่น จะสามารถกำหนดลำดับความสำคัญของการขัดจังหวะได้ใหม่โดยการกำหนดข้อมูลในบิทของรีจิสเตอร์ IP (Interrupt Priority Register) ตามตำแหน่งของแต่ละบิทในรูปที่ 3.32

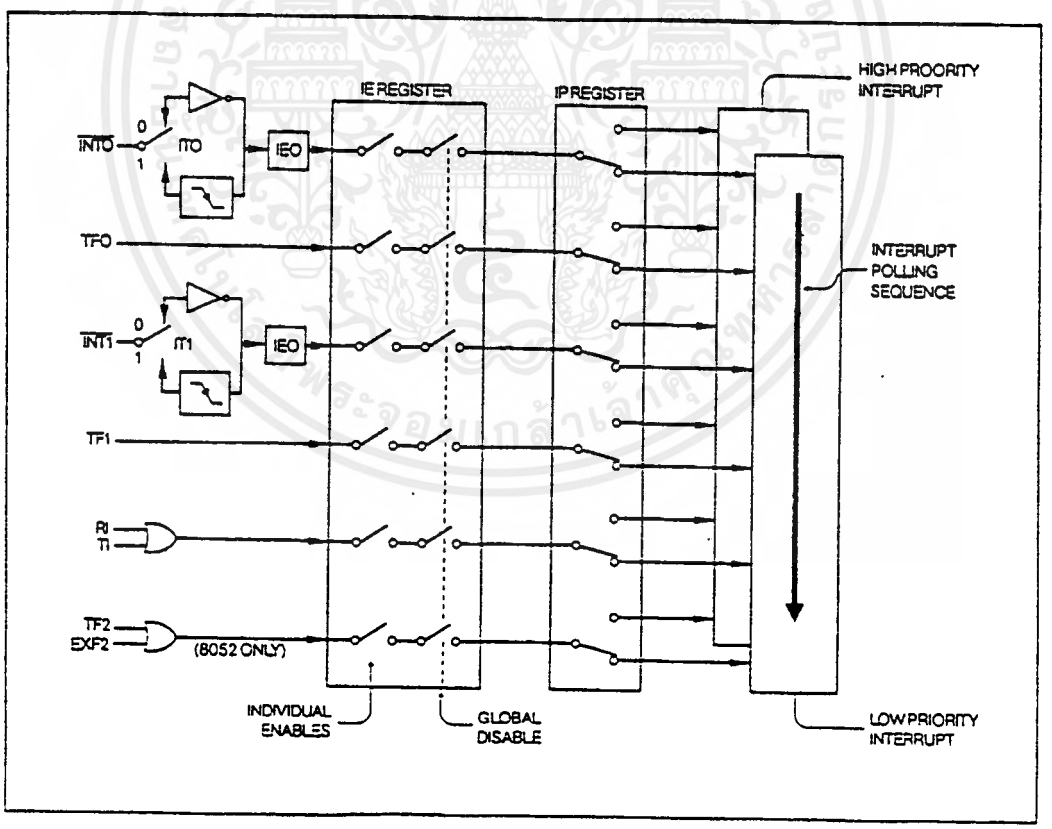


(MSB) (LSB)

X	X	PT2	PS	PT1	PX1	PT0	PX0

Symbol	Position	Function
PCT	IP.7	PCT = 1, only one level
–	IP.6	reserved
PE2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
PT1	IP.3	defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	defines the External interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

รูปที่ 3.32 IP : Interrupt Priority Register



รูปที่ 3.33 ระบบการขัดจังหวะของ AT89C52

รูปที่ 3.33 เป็นแผนภาพแสดงระบบขัดจังหวะของ 8052 ซึ่งแตกต่างจากของ 89C52 ตรงที่ 8052 จะมีสัญญาณขัดจังหวะมาจาก TF2 , EFX2 คือชุดล่างในภาพ

ในรูปจะเห็นว่าแต่ละสัญญาณจะมีสวิตช์ควบคุมอยู่ 3 ตัว 2 ตัวแรกอยู่ในกรอบสี่เหลี่ยม IE Register และอีก 1 สวิตช์อยู่ในกรอบ IP Register สวิทช์ตัวแรกทางซ้ายสุดจะควบคุมข้อมูลด้วยข้อมูลแต่ละบิท บิท 0 ถึงบิท 5 ของรีจิสเตอร์ IE ถ้าข้อมูลเป็น 1 จะทำให้สวิตช์นั้นปิดวงจร (Closed circuit) การควบคุมสวิตช์ทางซ้ายสุดของแต่ละสัญญาณจะไม่ขึ้นแก่กัน (Individual) สวิตช์ที่ 2 ถัดมาของทุกสัญญาณจะควบคุมร่วมกันด้วยบิท EA ในรีจิสเตอร์ IE ถ้าบิทนี้เป็น 0 สวิตช์ที่ 2 ของทุกสัญญาณจะเปิดวงจร (Opened Circuit) ทำให้ไม่มีสัญญาณขอขัดจังหวะผ่านไปได้ สวิตช์ที่ 3 ทางขวาสุดจะใช้สำหรับเลือกว่าสัญญาณนั้นจะอยู่ในกลุ่มลำดับความสำคัญสูง (High Priority Interrupt) หรือลำดับความสำคัญต่ำ (Low Priority Interrupt)

ถ้าต้องการให้สัญญาณใดมีลำดับความสำคัญสูงก็ให้กำหนดบิทนั้นในรีจิสเตอร์ IP เป็น 1 สวิตช์ที่ 3 จะเลื่อนไปอยู่ในตำแหน่งบน ถ้าไม่ต้องการก็กำหนดให้บิทนั้นเป็น 0 บิทใดเป็น 1 เรียกว่าสัญญาณนั้นจะอยู่ในกลุ่มลำดับความสำคัญสูงและบิทใดเป็น 0 เรียกว่าสัญญาณนั้นอยู่ในกลุ่มลำดับความสำคัญต่ำ ถ้าในกลุ่มลำดับความสำคัญสูงมีเพียง 1 สัญญาณก็จะเรียกว่าสัญญาณนั้นมีลำดับความสำคัญสูงสุด ในกลุ่มลำดับความสำคัญเดียวกันก็จะมีการจัดลำดับความสำคัญเฉพาะกลุ่มโดยวิธี Polling เหมือนเดิม เช่นกรณีที่มีการกำหนดในบิทของรีจิสเตอร์ IP ให้มีลำดับความสำคัญสุงหรือต่ำเหมือนกันแล้ว เกิดมีความต้องการขอการขัดจังหวะจากสัญญาณนั้น ๆ มาพร้อมกัน 89C52 ก็จะทำงานในโปรแกรมตอบสนองการขัดจังหวะของ Timer 1 , External interrupt 1 และ Timer 0 พร้อมกัน 89C52 กำลังทำงานตอบสนองการขัดจังหวะของสัญญาณขัดจังหวะที่ลำดับความสำคัญต่ำอยู่ ถ้ามีสัญญาณขัดจังหวะที่มีลำดับความสำคัญสุงกว่าเกิดขึ้น การทำงานของโปรแกรมก็จะกระโดดไปทำงานในตำแหน่งโปรแกรมตอบสนองการขัดจังหวะของสัญญาณที่มีลำดับความสำคัญสูง เสร็จแล้วจึงกลับมาทำงานที่โปรแกรมตอบสนองการขัดจังหวะลำดับความสำคัญต่ำต่อไป แต่ละบิทของรีจิสเตอร์ IP นั้นจะบอกลำดับความสำคัญของแหล่งกำเนิดสัญญาณขัดจังหวะดังนี้

PX0 บิท 0 เป็นลำดับความสำคัญของสัญญาณขอขัดจังหวะภายนอก 89C52 คือ INT0

PT0 บิท 1 เป็นลำดับความสำคัญของสัญญาณขอขัดจังหวะจาก Timer 0

PX1 บิท 2 เป็นลำดับความสำคัญของสัญญาณขอขัดจังหวะภายนอกของ 89C52 คือ INT1

PT1 บิท 3 เป็นลำดับความสำคัญของสัญญาณขอขัดจังหวะจาก Timer 1

PT2 บิท 5 เป็นลำดับความสำคัญของสัญญาณขอขัดจังหวะจาก Timer 2 บิทนี้ใช้เฉพาะใน 8052 ที่มี Timer 2

PS บิท 3 เป็นลำดับความสำคัญของสัญญาณขอขัดจังหวะจาก Serial Port ในกรณีที่มีข้อมูลเข้ามาหรือส่งข้อมูลออกสิ้นสุดแล้ว

บิทที่เหลือกจะไม่มีการใช้งาน

รายละเอียดของการขัดจังหวะจะกล่าวต่อไปในหัวข้อ 3.9

14. PCON (Power Control Register) ตำแหน่งหน่วยความจำภายในเท่ากับ 87H

Symbol	Position	Name and Function
SMOD	PCON.7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1,2 or 3.
HPD	PCON.6 (83C154 only)	Hard Power Down bit. Setting this bit allows CPU to enter in Power Down state on an external event (1 to 0 transition) on bit T1 (p.3-5) the CPU quit the Hard Power Down mode when bit T1 (p. 3-5) go high or when reset is activated.
RPD	PCON.5 (83C154 only)	Recover from idle or Power Down bit. When 0 RPD has no effect. When 1, RPD permits to exit from idle or Power Down with any non enabled interrupt source (except timex 2). In this case the program start at the next address. When interrupt is enabled the appropriate interrupt routine is serviced.
–	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF0	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Ide mode bit. Setting this bit activates idle mode operation.

รูปที่ 3.34 PCON : Power Control Register

89C52 เป็นไมโครโปรคอนโทรลเลอร์ที่สร้างขึ้นด้วยเทคโนโลยี่ทั้งแบบ CHMOS และ HMOS ซึ่งแบบ CHMOS มีข้อดีตรงที่ใช้กำลังไฟต่ำกว่าแบบ HMOS ดังนั้นต่อไปในอนาคตจึงจะมีแต่เฉพาะรุ่น CHMOS เท่านั้น นอกจากนี้แล้ว 89C52 ยังมีข้อดีอีกตรงที่สามรถลดการใช้กำลังไฟลงได้โดยการทำงานใน Idle Mode และ Power Down Mode ใน Idle Mode นั้นสัญญาณนาฬิกาจากออสซิลเลเตอร์จะป้อนให้เฉพาะส่วน Interrupt , Serial Port และ Timer ในส่วนอื่นจะไม่มีสัญญาณนาฬิกาไปเลี้ยงแต่มีไฟเลี้ยงให้กับทุกส่วนในวงจร การใช้กำลังไฟจึงลดลงมาก ส่วนใน Power Down Mode นั้น ออสซิลเลเตอร์จะหยุดทำงานทำให้ไม่มีสัญญาณนาฬิกาไปเลี้ยงส่วนใด ๆ ในวงจรเลยแต่ข้อมูลภายในรีจิสเตอร์จะยังคงอยู่ไม่สูญหายไป รายละเอียดของแต่ละโหมดจะได้กล่าวต่อไป

การสั่งงานให้ 89C52 ทำงานในโหมดของ Idle หรือ Power Down จะสามารถทำได้โดยใช้กำหนดค่าในรีจิสเตอร์ PCON (Power Control Register) แต่ละบิทในรีจิสเตอร์ PCON มีดังนี้

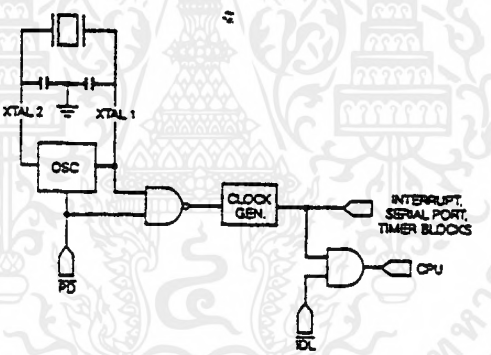
IDC บิท 0 ถ้าบิทนี้ถูก Set ให้เป็น 1 89C52 จะเข้าสู่การทำงานใน Idle Mode ทันที

PD บิท 1 ถ้าบิทนี้ถูก Set ให้เป็น 1 89C52 จะเข้าสู่การทำงานใน Power Down ทันที

GF0, GF1 บิท 2,3 เป็น General Purpose Flag bit บิทนี้สามารถกำหนดให้มี ค่าเป็น 0 หรือ 1 ได้ โดยโปรแกรม เพื่อส่งผ่านสถานะการทำงานของ 89C52 ระหว่างแต่ละโปรแกรมย่อย

SMOD บิท 7 เป็นบิทที่ใช้ร่วมในการกำหนดส่งข้อมูล (Baud Rate) ผ่านทางพอร์ทอนุกรม ซึ่งในการรับ-ส่งข้อมูลผ่านทางพอร์ทอนุกรมโหมด 1 และ 3 จะสามารถกำหนดอัตราการส่งข้อมูลได้ตามอัตราการเกิด Overflow ใน Timer 1 ถ้าบิทนี้เป็น 1 จะทำให้อัตราการส่งข้อมูล เพิ่มขึ้น 2 เท่า รายละเอียดการส่งข้อมูลผ่านพอร์ทอนุกรมดูในข้อการรับส่งข้อมูลทางพอร์ทอนุกรม

บิท 4,5,6 ไม่ได้ใช้งาน



รูปที่ 3.35 Power down และ Idle mode

Idle Mode

ในรูปที่ 3.35 ขณะที่ 89C52 ทำงานตามปกติไปจนถึงคำสั่งที่ทำให้บิท 0 ของรีจิสเตอร์ PMOD มีค่าเป็น 1 ก็จะเข้าสู่การทำงานใน Idle Mode โดยสัญญาณ IDL จะเป็น LOW (สัญญาณจะตรงข้ามกับข้อมูลในบิท 0) ขณะนี้สัญญาณนาฬิกาออสซิลเลเตอร์ไม่ออกจาก AND GATE ไปยังส่วน CPU โดยจ่ายเฉพาะส่วน Interrupt , Timer และ Serial Port ในขณะนี้ 89C52 จะเสมือนหยุดการทำงานโดยข้อมูลใน Stack Pointer , Program Counter , Program Status Word , Accumulator และรีจิสเตอร์อื่น ๆ จะ

ไม่เปลี่ยนแปลงข้อมูลพอร์ทต่าง ๆ จะยังคงค่าเดิมไว้เหมือนกับก่อนเข้าสู่ Idle mode และสัญญาณ ALE กับ PSEN จะเป็นลอจิก High ขณะนี้การใช้กระแสไฟของ 89C52 จะไม่มีการเปลี่ยนสถานะลอจิก การที่จะออกจาก Idle Mode ทำได้ 2 วิธี

วิธีที่ 1 โดยการขัดจังหวะจากสัญญาณขัดจังหวะทั้ง 6 ที่กล่าวมาแล้ว เมื่อมีสัญญาณขอขัดจังหวะจากแหล่งใดก็ตาม จะทำให้บิท 0 ของรีจิสเตอร์ PCON มีค่าเป็น 0 และการทำงานของ 89C52 จะออกจาก Idle Mode โดยกระโดดไปทำงานยังตำแหน่งของโปรแกรมตอบสนองการขัดจังหวะนั้น ๆ เมื่อเสร็จสิ้นการทำงานของโปรแกรมตอบสนองการขัดจังหวะโดยการทำงานคำสั่ง RETI ก็จะกลับมาทำงานยังคำสั่งที่อยู่ต่อจากคำสั่งที่ทำให้บิท 0 ของรีจิสเตอร์ PMOD เป็น 1 ซึ่งทำให้การทำงานเข้าสู่ Idle Mode เช่นคำสั่งที่ตำแหน่ง 200H คือ MOV PCON , #1H ที่เป็นคำสั่งที่ทำให้บิท IDC มีค่า 1 ดังนั้นเมื่อทำงานที่คำสั่งนี้เสร็จสิ้นก็จะหยุดการทำงาน และเมื่อเกิดการขัดจังหวะ เนื่องจากสัญญาณขัดจังหวะใด ๆ ก็ตาม 89C52 จะออกจาก Idle Mode ไปทำงานที่โปรแกรมตอบสนองการขัดจังหวะเมื่อเสร็จสิ้นการทำงานโปรแกรมตอบสนองการขัดจังหวะแล้วจะกระโดดมาทำงานที่ตำแหน่งของคำสั่ง MOV PCON , #1H

วิธีที่ 2 ก็คือการป้อนสัญญาณที่มีสภาวะลอจิก 1 เข้าไปยังขา RST เพื่อทำการรีเซ็ท 89C52 สัญญาณรีเซ็ทนี้จะต้องมีลอจิกเป็น 1 ในระหว่างนี้ 89C52 จะทำงานในคำสั่งต่อจากคำสั่งที่ทำให้บิท 0 ของ ของ PCON เป็น 1 เข้าสู่ Idle Mode ต่อไปอีก 2-3 คำสั่ง ก่อนที่ทุกอย่างจะเข้าสู่การรีเซ็ท (ดูรายละเอียดการรีเซ็ทในหัวข้อ 3.4) ดังนั้นจะต้องระวังคำสั่งที่อยู่ต่อจากคำสั่งที่ทำให้เข้าสู่ Idle Mode อาจทำให้ข้อมูลบนพอร์ทเปลี่ยนแปลงจนทำให้อุปกรณ์ที่มาต่อเสียหายเมื่อกลับออกจาก Idle Mode

ในวิธีที่ 1 นั้นแสดงว่าการเข้าสู่โปรแกรมตอบสนองการขัดจังหวะจะเป็นได้ 2 กรณีคือขณะที่ทำงานตามปกติแล้วมีสัญญาณขัดจังหวะก็จะกระโดดไปทำงานในโปรแกรมตอบสนองการขัดจังหวะ หรือในกรณีที่อยู่ใน Idle Mode แล้วมีสัญญาณขัดจังหวะก็จะกระโดดไปทำงานในโปรแกรมตอบสนองการขัดจังหวะ จึงอาจทำให้โปรแกรมตอบสนองการขัดจังหวะนั้นมาจากกรณีใด

Power Down Mode

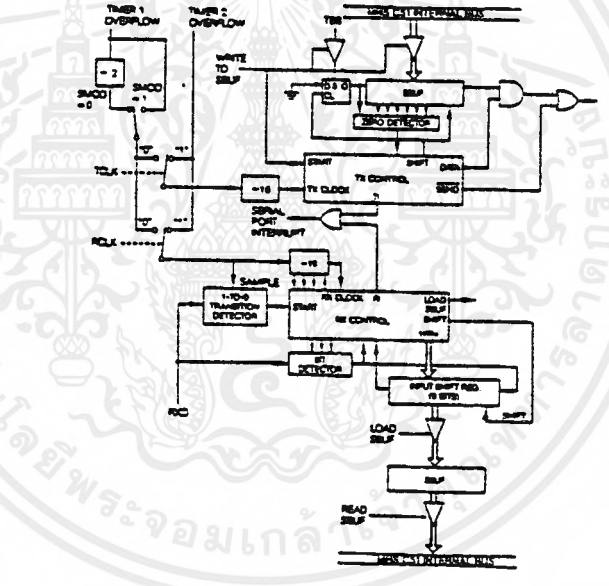
ในการเข้าสู่ Power down mode นั้นจะทำได้โดยการใช้โปรแกรมกำหนดให้บิท PD หรือบิท 1 ของรีจิสเตอร์ PCON มีค่าเป็น 1 เช่น MOV PCON , #2 เมื่อ 89C52 ทำงานที่คำสั่งนี้เสร็จสิ้น สัญญาณ PD ในรูปที่ 3.35 จะเป็น 0 เพราะจะตรงข้ามกับข้อมูลในบิท PD ทำให้การทำงานจะเข้าสู่ Power Down Mode ทันที ในโหมดนี้ออสซิลเลเตอร์จะหยุดการทำงานทำให้มีสัญญาณนาฬิกาไปยังส่วนต่าง ๆ ภายใน 89C52 ดังนั้นจะไม่มีการทำงานใด ๆ รวมทั้งข้อมูลในรีจิสเตอร์ทุกตัวจะไม่เปลี่ยนแปลง และข้อมูลใน แรมภายใน ก็จะไม่เปลี่ยนแปลง ขณะนี้สัญญาณออกจากขา ALE และ PSEN จะเป็น 0 การใช้กำลัง

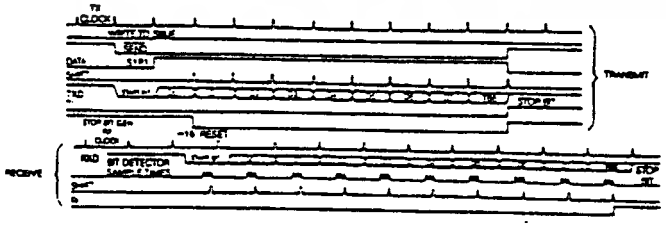
ไฟของ AT89C52 จะต่ำมากอีกทั้งสามารถลดไฟเลี้ยงวงจรที่ขา VCC ลงได้จนถึง 2 โวล์ทโดยไม่ทำให้ข้อมูลใด ๆ ใน AT89C52 สูญหายไป การออกจาก Power Down Mode ทำได้วิธีเดียวคือ การป้อนสัญญาณลอจิก 1 เข้าไปยังขา RST ซึ่งทำให้เข้าสู่สภาวะการรีเซ็ท AT89C52 แต่จะทำให้ข้อมูลใน SFR เปลี่ยนแปลงไป ถ้าในขณะที่อยู่ใน Power Down Mode มีการลดไฟเลี้ยงวงจรจะต้องให้ไฟเลี้ยงวงจรกับมาอยู่ที่ 5 โวล์ท ก่อนที่จะเข้าสู่การรีเซ็ท

3.8 การรับ-ส่งข้อมูลทางพอร์ทอนุกรม

ในการรับ-ส่งข้อมูลแบบอนุกรมผ่านทางพอร์ทอนุกรมนั้น จะต้องมีการกำหนดโหมดการทำงานในรีจิสเตอร์ SCON และในบางโหมดของการทำงานจะสามารถกำหนดอัตราการส่งข้อมูลได้โดยการโปรแกรมใน Timer ข้อมูลที่จะส่งออกหรือรับเข้าทางพอร์ทอนุกรมจะอยู่ที่รีจิสเตอร์ SBUF การทำงานของวงจรภายในแต่ละโหมดมีดังนี้ โดยแสดงการทำงานในโหมด 1 เท่านั้น

การทำงานโหมด 1





รูปที่ 3.36 Serial Port mode 1

การส่งข้อมูล

จากรูปที่ 3.36 บิท SMOD จะเป็นตัวเลือกว่า สัญญาณ Timer 1 Overflow ที่ส่งไปยังวงจรหาร 16 จะถูกหาร 2 ก่อนหรือไม่ ถ้า SMOD เป็น 1 สัญญาณ Timer 1 จะไม่ถูกหาร แต่ถ้า SMOD เป็นสัญญาณ 1 Timer Overflow จะถูกหาร 2 ก่อนที่จะเข้าวงจรหาร 16 การส่งข้อมูลจะเริ่มจากการที่มีคำสั่งเขียนข้อมูลไปยังรีจิสเตอร์ SBUF จะมีสัญญาณ Write to SBUF เกิดขึ้นเพื่อรับข้อมูลจาก Internal Bus ด้านบนไปเก็บยังรีจิสเตอร์ SBUF และทำให้เอาต์พุตของ D FLIP FLOP ทางซ้ายของ SBUF มีค่าเป็น 1 และเป็นบิทที่ 9 ของการส่งข้อมูล สัญญาณ Write to SBUF ยังส่งไปยัง TX control ด้วย ขณะนี้ข้อมูลในวงจรหาร 16 มีค่าเป็นอะไรไม่ทราบจึงจะรอจนกว่าข้อมูลในวงจรหาร 16 นับเพิ่มขึ้นจนถึงค่าสูงสุดแล้ววนกลับมาเป็น 0 คือเกิดการวนกลับทำให้เริ่มการส่งข้อมูลที่เวลา S1P1 ของไซเคิลเครื่องถัดไป (การส่งข้อมูลออกจะสัมพันธ์กับการเกิด Overflow ในวงจรหาร 16) สัญญาณ SEND จาก TX Control เปลี่ยนสภาวะลอจิกเป็น 0 แล้วเริ่มส่งข้อมูลที่เป็น Start bit (0) ออกไป เมื่อส่ง Start Bit ออกไปแล้ววงจร Tx Control ก็จะทำให้สัญญาณ DATA เป็น 1 เพื่อเลื่อนข้อมูลใน SBUF ออกไป เริ่มจากบิท 0 จนถึงบิทที่ 7 การส่งข้อมูลนี้จะเกิดขึ้นเมื่อสัญญาณ Tx Clock เปลี่ยนสถานะจาก 0 เป็น 1 ดังในรูปที่ 3.36 ขณะที่ข้อมูลถูกเลื่อนออกไปนั้นจะมี 0 ถูกเลื่อนเข้ามาทางซ้ายของรีจิสเตอร์ เมื่อข้อมูลเลื่อนออกไปทั้ง 8 บิทแล้วบิทที่ 9 ซึ่งเป็น 1 และตอนต้นอยู่ทางซ้ายจะถูกเลื่อนมาอยู่ในตำแหน่งสุดท้ายทางขวาของรีจิสเตอร์ SBUF และทางซ้ายของหลักนี้จะมี 0 อยู่ทั้ง 8 บิทใน SBUF ทำให้ Zero Detector ส่งสัญญาณ Shift ออกไปเป็นการส่งข้อมูลบิทสุดท้าย (บิท 7) ออกไป ก็จะรออีก 1 TX Clock (Bit Clock) ก็จะทำให้ขา TXD ส่งข้อมูล Stop Bit (1) ออกมา สัญญาณ DATA ซึ่งมีสภาวะลอจิกเป็น 1 มาตั้งแต่เริ่มส่งข้อมูลบิท 0 ก็จะกลับเป็น 0 และบิท TI จะเป็น 1 เพื่อบอกการสิ้นสุดการส่งข้อมูลทั้งหมดจะสิ้นสุดเมื่อสัญญาณ TX Clock ไซเคิลที่ 10 นับตั้งแต่สัญญาณ SEND เปลี่ยนสภาวะลอจิกเป็น 0

การรับข้อมูล

การรับข้อมูลจะขึ้นกับอัตราการเกิด Overflow ใน Timer 1 แล้วหาร 2 หรือไม่ขึ้นกับค่าของบิท SMOD สัญญาณนี้จะไปเข้าวงจรหาร 15 และเป็นตัวกำหนดอัตราการรับข้อมูล การรับข้อมูล จะเริ่มจากวงจร 1-TO-0 Transition Detector พบว่าสัญญาณที่ขา RXD เปลี่ยนจาก 1 เป็น 0 ซึ่งหมายถึงข้อมูล Start bit เข้ามา การตรวจสอบนี้จะกระทำด้วยอัตราเดียวกับสัญญาณที่เข้าวงจรหาร 16 เมื่อพบการเปลี่ยนสถานะลอจิกที่ขา RXD ก็จะเริ่มการับข้อมูล ขณะนี้จะรีเซ็ทวงจรหาร 16 ให้มีค่าเป็น 0 เพื่อสร้างสัญญาณ RX Clock ให้เข้าจังหวะ (Synchronous) กับข้อมูลที่เข้ามาโดยสัญญาณ RX Clock จะเป็น 1 เมื่อการนับของวงจรหาร 16 มีค่าเป็น 15 ขณะที่วงจรหาร 16 นับถึง 7 , 8 และ 9 จะมีการตรวจสอบข้อ

มูลที่เข้ามาทางขา RXD เพื่อเป็นการตรวจข้อมูลนั้นเป็นอะไร ถ้าอย่างน้อยข้อมูล 2 ใน 3 เป็นค่าใดก็จะถือว่าข้อมูลที่เข้ามาเป็นค่านั้น ถ้าในการตรวจสอบ Start Bit แล้วพบว่าผิดพลาด คือไม่เป็น 0 ก็จะรีเซ็ทการทำงานเพื่อไปตรวจสอบการเปลี่ยนสถานะจาก 1 เป็น 0 ของข้อมูลที่ขา RXD ใหม่ แต่ถ้าพบ Start Bit ก็จะเก็บข้อมูลทั้งหมดที่เข้ามาโดยเลื่อนข้อมูลเข้าไปยัง Input Shift Register ที่มีสัญญาณควบคุมการเลื่อนข้อมูล (Shift) ส่งมาจาก RX Control ในเริ่มต้นการรับข้อมูลจะมีการเขียนข้อมูล 1FFH ไปเก็บใน Input Shift Register ขณะที่ข้อมูลถูกเลื่อนเข้าไปทางขวาของ Input Shift Register ก็จะมี 1 ถูกเลื่อนออกไปทางซ้ายทุกครั้งที่มีข้อมูลเข้ามา เมื่อ Start Bit ที่รับเข้ามาถูกเลื่อนไปทางซ้ายสุดของ Input Shift Register ก็จะมีสัญญาณไปบอก RX Control Block หลังจากข้อมูลบิทสุดท้ายเข้ามาแล้วก็จะโหลด (Load) เอาข้อมูล 8 บิทไปเก็บในรีจิสเตอร์ SBUF พร้อมทั้ง Set ค่าในบิท RI และRB8 ของรีจิสเตอร์ SCON แต่การโหลดข้อมูลไปเก็บนี้จะเกิดขึ้นได้ก็ต่อเมื่อ

1. RI = 0 และ
2. SM2 = 0 หรือถ้า SM2 = จะต้องได้รับ stop bit เป็น 1

ถ้าไม่มีสภาวะใดสภาวะหนึ่งดังกล่าวแล้ว ข้อมูลที่รับเข้ามาจะถูกทิ้งไปคือ ไม่โหลดไปเก็บในรีจิสเตอร์ SBUF ถ้ามีสภาวะดังกล่าวถูกต้อง stop bit จะถูกนำไปเก็บในรีจิสเตอร์ SBUF และบิท RI จะเป็น 1

แต่ไม่ว่าทั้ง 2 กรณีจะเกิดหรือไม่ก็จะกลับไปสู่การตรวจสภาวะเปลี่ยนจาก 1 เป็น 0 ที่ขา RXD เพื่อรับข้อมูลต่อไป

ในการรับข้อมูลแบบอนุกรมโหมด 1 นี้อัตราการส่งข้อมูลแต่ละบิทท (Band Rate) จะขึ้นกับ

$$Baudrate = \frac{2^{SMOD}}{32} \times (Timer_1 Overflow Rate)$$

อัตราการเกิด overflow ใน Timer 1 ดังสมการ

ในขณะที่ใช้ Timer 1 เป็นตัวกำหนด Baud Rate นี้จะต้อง Disable ไม่ให้เกิดการขัดจังหวะเนื่องมาจากการ Overflow Timer 1 อาจใช้ในโหมดของ Timer หรือ Counter ก็ได้ ซึ่งเมื่อการนับในรีจิสเตอร์ตัวนับมีค่าสูงสุดแล้วกลับมาเป็น 0 ก็จะเกิด Overflow เช่นเดียวกัน แต่โดยปกติแล้วจะใช้ Timer 1

นี้ในโหมดของ Timer ที่มีการทำงานแบบ Auto Reload โหมด 2 เพื่อว่าเมื่อค่าในการนับโดย

รีจิสเตอร์ TL1 ถึงค่าสูงสุดก็จะโหลดค่าในรีจิสเตอร์ TH1 มาไว้ใน TL1 สำหรับเป็นค่าเริ่มต้นการนับต่อไป ซึ่ง Baud rate จะมีค่า

$$Baudrate = \frac{2^{SMOD}}{32} \times \frac{OscillatorFrequency}{12 \times [256 - (TH1)]}$$

โดยที่ SMOD เป็นบิทหนึ่งในรีจิสเตอร์ PCO

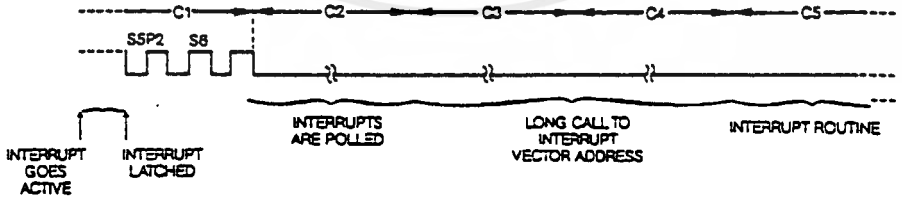
3.9 การขัดจังหวะ (Interrupt)

การขัดจังหวะคือสภาวะหนึ่งที่คอมพิวเตอร์กำลังทำงานอยู่แล้วถูกขัดจังหวะด้วยสัญญาณหรือคำสั่งพิเศษที่ทำให้คอมพิวเตอร์ต้องละจากงานที่กำลังทำอยู่ ไปทำงานในโปรแกรมตอบสนองการขัดจังหวะนั้น เมื่อเสร็จแล้วก็จะกลับมาทำงานเดิมต่อไปได้ ใน 89C52 จะสามารถขัดจังหวะกาทำงานได้ 6 แหล่งคือ

1. INT0 , INT1 เป็น 2 ขาของ 89C52 ที่จะรับสัญญาณจากภายนอก การขัดจังหวะจะเกิดขึ้น ถ้าสัญญาณที่ขาดังกล่าวมีสภาวะลอจิกเป็น 0 หรือเปลี่ยนจาก 1 เป็น 0 โดยเลือกด้วยการกำหนดในบิท IT0 หรือ IT1 ในรีจิสเตอร์ TCON

2. TF0 , TF1 เป็นบิทหนึ่งที่จะบอกการทำงานของ Timer 0 , Timer 1 เมื่อเกิด Overflow ขึ้นใน Timer จะทำให้บิทนี้เป็น 1 และเกิดการขัดจังหวะการทำงานของ 89C52 ได้

3. TI , RI เป็น 2 บิท ในรีจิสเตอร์ SCON ถ้าบิทนี้ถูกเซตให้เป็น 1 โดยฮาร์ดแวร์ อันเนื่องจากเสร็จสิ้นการส่งหรือรับข้อมูลจะสามารถทำให้เป็นการขัดจังหวะได้



รูปที่ 3.37 ไดอะแกรมเวลาของการตอบสนองการขัดจังหวะ

89C52 จะทำการอ่านสัญญาณจากทั้ง 6 แหล่งที่เวลา S5P2 ของทุก ๆ ไซเคิลของเครื่อง (Machine Cycle) เข้ามาเก็บและในช่วงของไซเคิลของเครื่องถัดไปก็จะตรวจสอบสถานะของสัญญาณทั้ง 6 ที่เก็บเข้ามา ถ้าสัญญาณนั้นมีการขัดจังหวะที่ถูกต้อง 89C52 ก็จะละทิ้งการทำงานเดิมไว้ชั่วคราว แล้วสร้างคำสั่ง LCALL ขึ้นมาภายใน 89C52 เพื่อไปทำงานในโปรแกรมตอบสนองการขัดจังหวะแต่ละสัญญาณนั้น เมื่อทำงานในโปรแกรมตอบสนองการขัดจังหวะเสร็จสิ้นก็จะสามารถกลับมาทำงานเดิมได้ โดยคำสั่ง RET1 เป็นคำสั่งสุดท้ายในโปรแกรมตอบสนองการขัดจังหวะ สัญญาณขัดจังหวะจากแต่ละแหล่งจะมีตำแหน่งหน่วยความจำที่จะเก็บโปรแกรมตอบสนองการขัดจังหวะไว้ต่างกันดังนี้

สัญญาณที่ขอขัดจังหวะ	ตำแหน่งเริ่มต้นโปรแกรมตอบสนองการขัดจังหวะ	
1	INT0	0003H
2	TFO	000BH
3	INT1	0013H
4	TF1	001BH
5	TI,RI	0023H

ตำแหน่งเริ่มต้นโปรแกรมนี้เป็นตำแหน่งใน Program area เช่น ถ้ามีสัญญาณของ INT0 เข้ามาแล้ว 89C52 ตรวจสอบว่ามีการขอขัดจังหวะถูกต้อง ก็จะละทิ้งการทำงานเดิม แล้วไปทำงานที่โปรแกรมตอบสนองการขัดจังหวะที่มีตำแหน่งเริ่มต้นอยู่ที่ตำแหน่ง 0003H เมื่อเสร็จสิ้นการทำงานของโปรแกรมตอบสนองการขัดจังหวะจะต้องมีคำสั่ง RETI อยู่เพื่อกลับมาสู่การทำงานเดิมได้ 89C52 จะทำการตรวจสอบสัญญาณดังกล่าวว่ามีสัญญาณใดขอการขัดจังหวะมาบ้างได้โดยวิธี Polling คือการตรวจสอบเรียงตามลำดับจาก 1 , 2 , 3 , 4 และ 5 ตามลำดับ ดังนั้นถ้ามีการขอขัดจังหวะเข้ามาพร้อม ๆ กัน AT89C52 ซึ่งตรวจสอบการขอขัดจังหวะแบบ Polling จะพบว่าสัญญาณมีการขอขัดจังหวะจากสัญญาณต้น ๆ ก่อนจึงตอบสนองต่อการขอขัดจังหวะของสัญญาณต้น ๆ ก่อนหรืออีกนัยหนึ่งก็คือสัญญาณขอการขัดจังหวะต้น ๆ จะมีลำดับความสำคัญสูงสุด (Highest Priority) และในสัญญาณที่ 5 จะมีลำดับความสำคัญต่ำที่สุด (Lowest Priority) อย่างไรก็ตามสามารถที่จะจัดลำดับความสำคัญของสัญญาณขัดจังหวะนี้ใหม่ เพื่อให้มีการตอบสนองการขัดจังหวะสัญญาณขอการขัดจังหวะลำดับหลังได้ โดยการโปรแกรมในรีจิสเตอร์ IP (Interrupt Priority Register) และสามารถกำหนดว่าจะให้ทำโปรแกรมตอบสนองการขัดจังหวะเมื่อมีสัญญาณขอขัดจังหวะเข้ามาหรือไม่ก็ได้ โดยการโปรแกรมในรีจิสเตอร์ IE (Interrupt Enable Register)

เมื่อ 89C52 ทำการตรวจสัญญาณขอการขัดจังหวะที่เก็บเข้ามาเมื่อเวลา S5P2 แล้วพบว่ามีการขอขัดจังหวะนั้น แม้ว่ามีการ Enable ในรีจิสเตอร์ IE ถูกต้อง แต่จะต้องมีเงื่อนไขดังนี้ด้วย

1. ไม่ได้กำลังทำงานในโปรแกรมตอบสนองการขัดจังหวะของสัญญาณ ที่มีลำดับความสำคัญสูงกว่าหรือเท่ากัน เช่น กำลังทำงานในโปรแกรมตอบสนองการขัดจังหวะของสัญญาณ INT0 อยู่ แล้วมีการขอขัดจังหวะจากสัญญาณ INT1 อีก จะไม่เกิดการทิ้งงานเดิม คือไม่มีการไปทำงานที่โปรแกรมตอบสนองการขัดจังหวะของสัญญาณ INT1

2. เนื่องจากการส่งสัญญาณเข้าไปเพื่อตรวจสอบนั้นจะทำให้เวลา S5P5 ของในไซเคิลสุดท้ายของคำสั่ง และคำสั่งที่อยู่ถัดมาจะต้องในเวลาทำงาน 2 ไซเคิลของเครื่อง ดังนั้นการตรวจสอบจะกระทำในไซเคิลแรก แม้ว่าจะมีการขอการขัดจังหวะเข้ามา ก็จะไม่ทำโปรแกรมตอบสนองการขัดจังหวะจะต้องอ่านสัญญาณที่เวลา S5P2 อีกครั้งแล้วไปตรวจสอบที่ไซเคิลที่ 2 ของคำสั่ง ถ้ามีการขอขัดจังหวะถูกต้องจึงจะข้ามไปทำงานในโปรแกรมตอบสนองการขัดจังหวะ

3. คำสั่งที่กำลังทำงานอยู่ขณะที่ตรวจสอบสัญญาณการขอขัดจังหวะ จะต้องไม่ใช่คำสั่ง RET หรือคำสั่งใด ๆ ก็ตามที่พยายามเขียนข้อมูลไปยังรีจิสเตอร์ IE หรือ IP

สัญญาณขอขัดจังหวะที่ถูกอ่านเข้าไปที่เวลา S5P2 นี้ไม่ว่าได้รับการตอบสนองหรือไม่จะถูกทิ้งไป แล้วอ่านเข้าไปใหม่ทุกเวลา S5P2

3.10 เมโมรีแม็พไอโอเทคนิค

เป็นวิธีการเพิ่มหมายเลขพอร์ตอินพุตเอาต์พุต โดยการกำหนดหมายเลขพอร์ตให้แทนที่ที่ตำแหน่งของหน่วยความจำ ตำแหน่งใดตำแหน่งหนึ่ง ซึ่งเป็นหน่วยความจำประเภทหน่วยความจำภายนอก ซึ่ง ณ. ตำแหน่งนั้นไม่มีหน่วยความจำจริงต่ออยู่ เพียงแต่ใช้หมายเลขของหน่วยความจำเป็นตัวอ้างอิงในการติดต่อ ในกรณีของ ชุดจำลอง พีแอลซี จะต้องใช้หมายเลขพอร์ตซึ่งได้จากวิธีการดังกล่าวนี้เป็นหมายเลขที่แน่นอน (แอ็ดทีฟที่หมายเลขเพียงหมายเลขเดียว เพื่อป้องกันความผิดพลาดที่จะเกิดขึ้นกับอุปกรณ์ภายนอก)

บทที่ 4

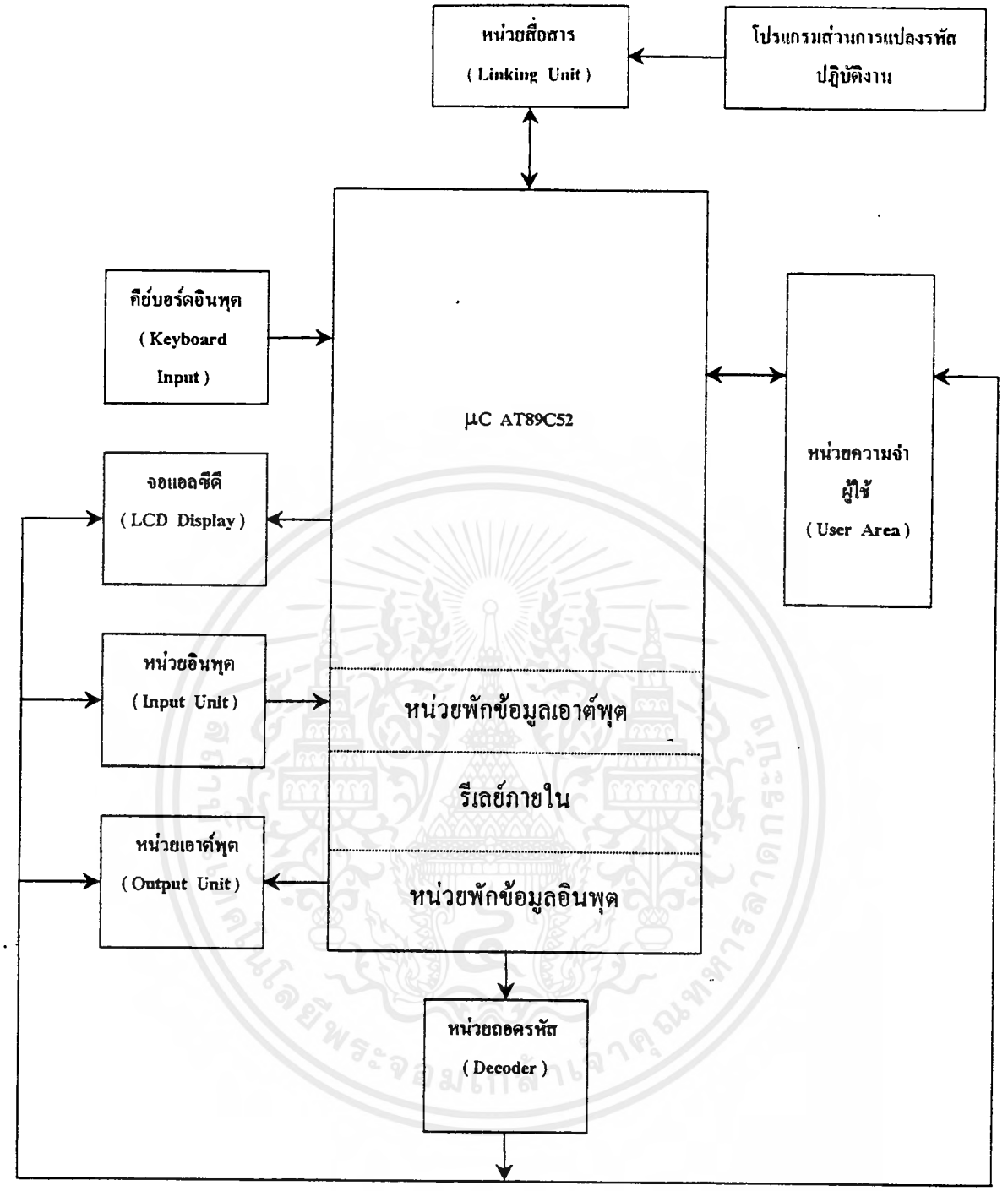
ส่วนประกอบของโครงงาน

4.1 ส่วนประกอบสำคัญ โครงงานประกอบด้วย 2 ส่วนใหญ่ ๆ คือ

1. ส่วนที่เป็น ฮาร์ดแวร์ (HARDWARE) คือส่วนของวงจร ที่ทำหน้าที่จำลองการทำงาน พีแอลซี
2. ส่วนที่เป็น ซอร์ฟแวร์ (SOFTWARE) แบ่งย่อยเป็น 2 ส่วน คือส่วนควบคุมการทำงานและส่วนสร้างชุดคำสั่งการปฏิบัติงานของผู้ใช้ (EXECUTABLE CODE)
   2.1 ชุดควบคุมการทำงานถูกเขียนขึ้นให้สามารถติดต่อสื่อสารกับเครื่องคอมพิวเตอร์ส่วนบุคคลเพื่อเลือกสภาวะการทำงาน (SRCOMLCD.ASM)
   2.2 ชุดสร้างโปรแกรมการปฏิบัติงานของผู้ใช้ ใช้ในการแปลคำสั่งของภาษาบูลลีนของพีแอลซี ให้เป็นรหัสปฏิบัติงานของไมโครคอนโทรลเลอร์ตระกูล MCS-51 (EXPLC.EXE)

4.2 หน้าที่การทำงานของส่วนประกอบโครงสร้างหลักของวงจร

1. ส่วนควบคุม และประมวลผล (AT89C52) เป็นตัวกลางในการประมวลผลการทำงานของวงจรทั้งหมด และจำลองส่วนประกอบ คือ รีเลย์ภายใน และ หน่วยพักของข้อมูลจากหน่วยอินพุตและเอาต์พุต
   (รีเลย์ภายใน ใช้เป็นเอาต์พุตเทียมซึ่งอำนวยความสะดวกในการเขียนชุดคำสั่ง)
2. แป้นพิมพ์ (คีย์บอร์ดอินพุต) เพื่อพัฒนาให้สามารถรับการสั่งงาน และรับส่วนของชุดคำสั่ง จากผู้ใช้ได้ในอนาคต
3. จอแสดงผลแอลซีดี ใช้แสดงสภาวะปัจจุบัน
4. หน่วยอินพุต เป็นส่วนรับสัญญาณจากอุปกรณ์ภายนอกเข้ามาประมวลผล
5. หน่วยเอาต์พุต เป็นส่วนส่งสัญญาณออกไปควบคุมอุปกรณ์ภายนอก
6. หน่วยสื่อสาร เป็นส่วนที่ช่วยให้วงจรสามารถติดต่อสื่อสารกับคอมพิวเตอร์ส่วนบุคคลได้
7. หน่วยความจำผู้ใช้ เป็นหน่วยความจำภายนอก ที่เป็นทั้งหน่วยความจำประเภทหน่วยความจำข้อมูล และหน่วยความจำชุดคำสั่ง
8. หน่วยถอดระหัส เป็นตัวกำหนดว่าช่วงเวลาใด จะให้หน่วยใดมีการติดต่อกับส่วนควบคุม และประมวลผล (AT89C52) ดังแสดงในรูปที่ 4.1



รูปที่ 4.1 แผนผังโครงสร้างหลักของชุดจำลองการทำงาน พีแอลซี

4.3 ระบบการทำงานภายในวงจรร่วมกับโปรแกรมมอนิเตอร์

ระบบการทำงานทั้งหมดมี 3 สภาวะ

1. สภาวะมอนิเตอร์ (Monitor Mode)

พิจารณาจากแผนผังโครงสร้างหลักของชุดจำลองการทำงานพีแอลซีได้ดังนี้ หลังจากระบบได้มีการติดตั้งค่าเริ่มต้นต่าง ๆ แล้วระบบจะนำเข้าสู่สภาวะมอนิเตอร์โหมดซึ่งเป็นการเตรียมพร้อมรับการสั่งงานจากผู้ใช้ ผ่านการสื่อสารทางพอร์ตอนุกรมกับคอมพิวเตอร์ส่วนบุคคล ว่าจะให้ระบบรอรับชุดคำสั่งจากคอมพิวเตอร์ส่วนบุคคล (PC) ในสภาวะโหมดสื่อสาร(Receive Mode) ซึ่งหลังจากได้ดำเนินการรับชุดคำสั่งจากวิธีดังกล่าว ระบบจะกลับสู่มอนิเตอร์โหมดอีกครั้งเพื่อรอรับการเปลี่ยนโหมดครั้งต่อไป ซึ่งอาจเป็นการเปลี่ยนเข้าสู่โหมดการทำงานเข้าสู่โหมดปฏิบัติการ (Running Mode) ได้

2. สภาวะโหมดสื่อสาร (Receive Mode)

สภาวะนี้จะทำงานร่วมกับคอมพิวเตอร์ส่วนบุคคล เพื่อบันทึกรหัสปฏิบัติการที่สร้างและได้รับการแปลจากโปรแกรมแปลชุดคำสั่งภาษาบูลลีนลงในหน่วยความจำผู้ใช้เพื่อนำไปปฏิบัติ

3. สภาวะโหมดปฏิบัติการ (Running Mode)

ระบบจะเข้าสู่สภาวะนี้เมื่อ ผู้ใช้สั่งงานผ่านแป้นพิมพ์ของคอมพิวเตอร์ส่วนบุคคล และระบบมีความพร้อม เมื่อระบบเข้าสู่สภาวะนี้ ระบบจะนำรหัสปฏิบัติงานที่รับได้จากโหมดสื่อสารมาเริ่มปฏิบัติ โดยเริ่มจากตำแหน่งที่ผู้ใช้ระบุเรียงลำดับตามที่ผู้ใช้กำหนด ขณะนี้ผู้ใช้สามารถตรวจสอบขั้นตอนการทำงานที่กระทำอยู่ว่าถูกต้องตามที่ต้องการหรือไม่

4.4 การทำงานของชุดคำสั่งของสภาวะมอนิเตอร์

หลังจากจ่ายพลังงานให้กับบอร์ดจำลองการทำงานพีแอลซีแล้ว หน่วยประมวลผลกลางจะเริ่มอ่านคำสั่งจากหน่วยความจำชุดคำสั่งภายใน ซึ่งสั่งให้เปิดการแสดงผลทางหน้าจอ LCD ก่อน แล้วทำการติดตั้งค่าเริ่มต้นสำหรับการสื่อสารกับคอมพิวเตอร์ส่วนบุคคลโดยผ่านโปรแกรม xtalk จากนั้นจะสั่งให้แสดงข้อความว่า พร้อมสำหรับการรอรับรหัสคำสั่งจากผู้ใช้ ว่าจะให้รอรับการบรรจุชุดคำสั่งที่ส่งมาจากคอมพิวเตอร์ส่วนบุคคลลงหน่วยความจำผู้ใช้ หรือเพื่อเข้าสู่การปฏิบัติชุดคำสั่งดังกล่าวนั้นต่อไป กรณีที่มีการค่ารหัสอื่นที่ไม่ได้กำหนดไว้เป็นระหัสคำสั่งระบบจะแสดงข้อความว่าคำสั่งที่สั่งไปยังวงจร จำลองการทำงานพีแอลซีนั้นไม่ถูกต้อง (Invalid Command) ทางหน้าจอคอมพิวเตอร์ส่วนบุคคล การรับคำสั่งจะรับจากการสื่อสารผ่านโปรแกรม xtalk.exe โดยมีเงื่อนไขในการรับคำสั่ง 2 ลักษณะ คือ ระหัสคำสั่งแรกคือคำสั่งรอรับชุดคำสั่ง ผ่านการสื่อสารทางพอร์ตอนุกรม คำสั่งที่สอง คือคำสั่งให้เริ่มปฏิบัติการตามชุดคำสั่งที่ได้รับผ่านการสื่อสารจาก

คอมพิวเตอร์ส่วนบุคคล และหน้าจอแสดงผลชนิดผลึกเหลวก็จะแสดงผล ด้วยว่าขณะนั้นระบบกำลังทำงานในโหมดใดอยู่

4.5 การทำงานของชุดคำสั่งสภาวะโหมดสื่อสาร

1. รอรับชุดคำสั่งจากการสื่อสารกับคอมพิวเตอร์ส่วนบุคคลผ่านพอร์ตอนุกรม
2. แสดงสถานะของการสื่อสารให้ผู้ใช้รับทราบว่า การถ่ายโอนชุดคำสั่งจากเครื่องคอมพิวเตอร์ส่วนบุคคลสำเร็จหรือผิดพลาด
3. คืนกลับสู่สภาวะมอนิเตอร์โหมด

4.6 การทำงานของชุดคำสั่งโหมดปฏิบัติการ

กำหนดตำแหน่งการปฏิบัติการกระโดดไปยังตำแหน่งปฏิบัติการที่กำหนด (โดยอาศัยหลักของคำสั่ง ret ของ MCS-51)

4.7 ขั้นตอนการสร้างรหัสปฏิบัติของชุดจำลองการทำงาน พีแอลซี

ผู้ใช้จะต้องสร้างชุดคำสั่งภาษาบูลลีนพื้นฐาน ด้วยโปรแกรมแก้ไขแฟ้มข้อมูลตัวอักษรใด ๆ (text editor) แล้วบันทึกให้มีส่วนขยายหรือนามสกุล PLC (Filename.PLC) จากนั้นเรียกตัวแปลภาษาชื่อ explc ตามด้วยชื่อไฟล์โดยไม่ต้องใส่นามสกุล (explc Filename) แล้วกดคีย์ย้อนกลับ (enter or return) โปรแกรม explc.exe นี้จะแปลภาษาบูลลีนจากแฟ้มข้อมูลต้นทางให้เป็นแฟ้มข้อมูลที่เป็นภาษาแอสเซมบลีของไมโครคอนโทรลเลอร์ตระกูล MCS-51 หากการแปลพบความผิดพลาดใดๆจะแสดงออกทางหน้าจอ และบันทึกลงแฟ้มข้อมูลอ้างอิงความผิดพลาด (โดยไม่สร้างแฟ้มข้อมูลภาษาแอสเซมบลี)เพื่อให้ผู้ใช้รับทราบข้อผิดพลาดนั้นแล้วทำการแก้ไขแฟ้มชุดคำสั่งก่อนจะทำการแปลครั้งต่อไป แต่หากการแปลกระทำได้เสร็จสมบูรณ์โปรแกรมจากสร้างชุดคำสั่งเป็นภาษาแอสเซมบลีออกมา เป็นชื่อแฟ้มข้อมูลเดียวกับแฟ้มข้อมูลภาษาบูลลีน แต่มีส่วนขยายเป็น ASM จากนั้น โปรแกรมจะเรียกโปรแกรมแปลภาษาแอสเซมบลีของไมโครคอนโทรลเลอร์ตระกูล MCS-51 ชื่อว่า sxa51.exe ให้ทำการแปลรหัสคำสั่งภาษาแอสเซมบลีที่ได้จากการแปลครั้งแรกให้เป็นแฟ้มข้อมูลที่บันทึกรหัสปฏิบัติการของ ไมโครคอนโทรลเลอร์ MCS-51 ในรูปแบบของอินเทลเฮ็กซ์ไฟล์ และจะได้แฟ้มข้อมูลเป็นชื่อเดียวกับแฟ้มข้อมูลชุดคำสั่งภาษาบูลลีนแต่มีส่วนขยายเป็น HEX (Filename.HEX) ซึ่งแฟ้มข้อมูลที่มีส่วนขยายเป็น .HEX นี้จะต้องทำการถ่ายโอนไปยังวงจรชุดจำลองการทำงานพีแอลซี ซึ่งจะต้องรอรับการถ่ายโอนในสภาวะโหมดรอรับก่อนเสมอ

4.8 ไวยากรณ์ของภาษาบูลลีน ชุดคำสั่งที่อนุญาตให้ผู้ใช้เลือกใช้งานได้

1. STR
2. STR NOT
3. AND
4. AND NOT
5. OR
6. OR NOT
7. OUT
8. OUT NOT
9. END

ซึ่งไวยากรณ์แต่ละคำสั่งจะเป็นดังต่อไปนี้

1. STR XXXX ชุดคำสั่งภาษาบูลลีนในแต่ละรังก์จะเริ่มต้นด้วยคำสั่งนี้ (หรือคำสั่ง STR NOT XXXX) เสมอซึ่งเป็นการระบุตัวตั้ง หากตัวแปลภาษาบูลลีนพบคำสั่งแรกของรังก์ที่ไม่ใช่คำสั่งนี้จะเกิดความผิดพลาดขึ้น ตัวแปรที่อยู่หลังจากคำสั่ง STR จะเป็นหมายเลขอินพุต หรือ รีเลย์ภายในซึ่งแสดงสถานะของเอาต์พุต หากตัวเลขที่ใส่ไม่อยู่ในย่านใด ๆ เลย จะเกิดความผิดพลาดของการแปลขึ้น
2. STR NOT XXXX เป็นคำสั่งสั่งระบุตัวตั้งเช่นเดียวกับคำสั่ง STR แต่จะทำการคอมพลีเมนต์ตัวตั้งก่อนนำไปกระทำกับตัวกระทำ สำหรับตัวแปร XXXX มีเงื่อนไขในการระบุเช่นเดียวกับคำสั่ง STR

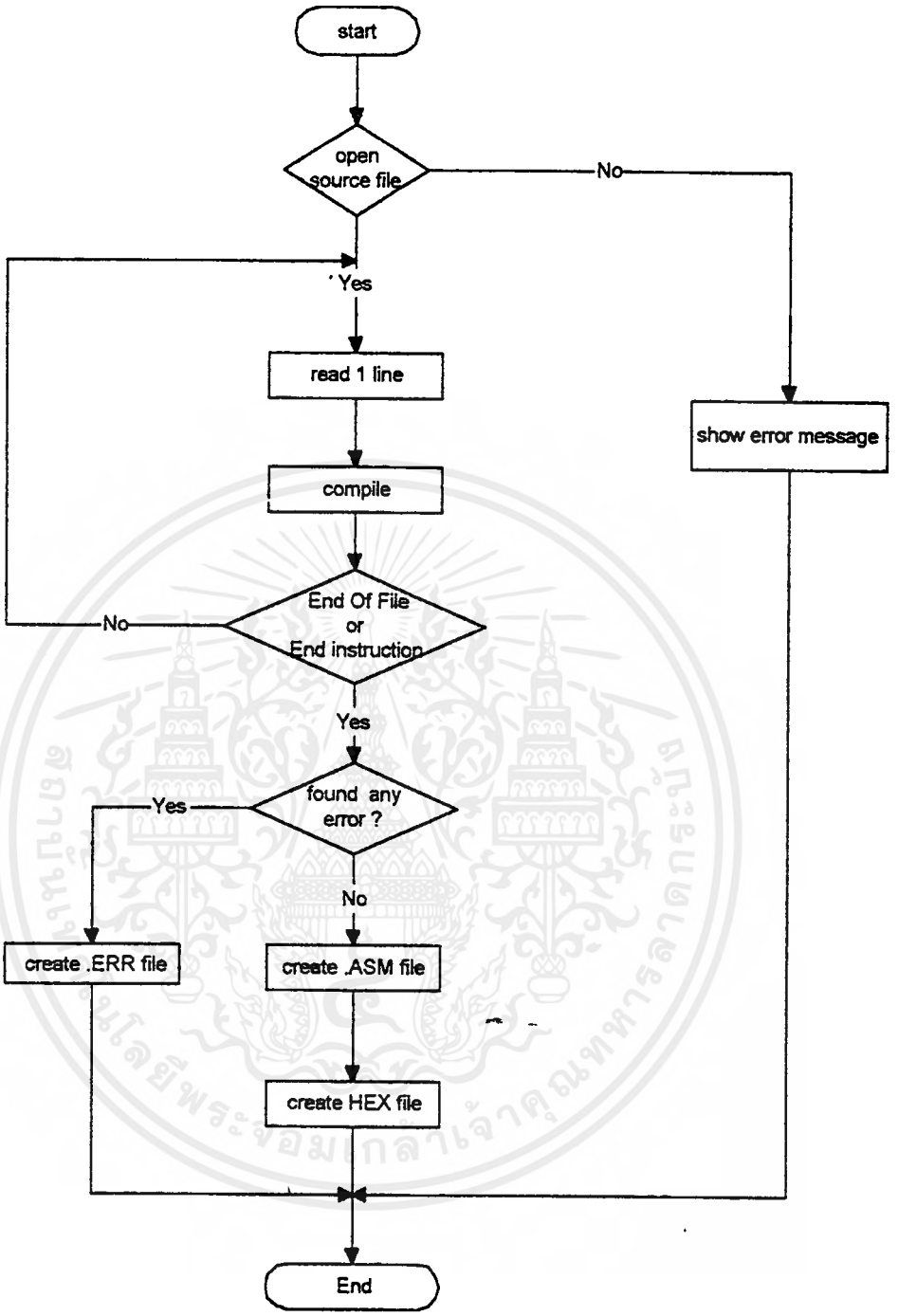
หมายเหตุ การใช้คำสั่ง STR หรือ STR NOT จะใช้ซ้อนกันเกิน 2 คำสั่งไม่ได้

3. AND XXXX เป็นคำสั่งระบุตัวกระทำซึ่งจะนำไปกระทำกับตัวตั้ง สำหรับตัวแปร XXXX มีเงื่อนไขในการระบุเช่นเดียวกับคำสั่ง STR หลังจากการกระทำทางลอจิกแอนด์แล้วจะถือว่าผลลัพธ์นั้นเป็นตัวตั้งต่อไป
4. AND NOT XXXX เป็นคำสั่งระบุตัวกระทำแต่จะทำการคอมพลีเมนต์ตัวกระทำนั้นก่อนสำหรับตัวแปร XXXX มีเงื่อนไขในการระบุเช่นเดียวกับคำสั่ง STR หลังจากการกระทำทางลอจิกแอนด์แล้วจะถือว่าผลลัพธ์นั้นเป็นตัวตั้งต่อไป
5. OR XXXX เป็นคำสั่งระบุตัวกระทำซึ่งจะนำไปกระทำกับตัวตั้ง สำหรับตัวแปร XXXX มีเงื่อนไขในการระบุเช่นเดียวกับคำสั่ง STR หลังจากการกระทำทางลอจิกออร์ แล้วจะถือว่าผลลัพธ์นั้นเป็นตัวตั้งต่อไป

6. OR NOT XXXX เป็นคำสั่งระบุตัวกระทำแต่จะทำการคอมพลีเมนต์ตัวกระทำนั้นก่อนสำหรับตัวแปร XXXX มีเงื่อนไขในการระบุเช่นเดียวกับคำสั่ง STR หลังจากการกระทำทางลอจิกออร์แล้วจะถือว่าผลลัพธ์นั้นเป็นตัวตั้งต่อไป
7. OUT YYYY เป็นคำสั่งส่งข้อมูลลอจิกที่ได้จากการทำงานทั้งหมดในรังก์ไปที่เอาต์พุต และเป็นการสิ้นสุดรังก์ โดยตัวแปร YYYY จะต้องอยู่ในย่านของหมายเลขเอาต์พุตหรือรีเลย์ภายในเท่านั้นมิฉะนั้นการแปลจะเกิดความผิดพลาด
8. OUT NOT YYYY เป็นคำสั่งส่งข้อมูลลอจิกที่ได้จากการทำงานทั้งหมด ในรังก์มาทำคอมพลีเมนต์ก่อนส่งไปที่เอาต์พุต และเป็นการสิ้นสุดรั้ง โดยตัวแปร YYYY จะต้องอยู่ในย่านของหมายเลขเอาต์พุตหรือรีเลย์ภายในเท่านั้น มิฉะนั้นการแปลจะเกิดความผิดพลาด
9. END เป็นคำสั่งสุดท้ายเพื่อบอกตัวแปรภาษาว่าเป็นการสิ้นสุดชุดคำสั่ง หากตัวแปรภาษาไม่พบคำสั่งนี้จะเกิดความผิดพลาดในการแปล

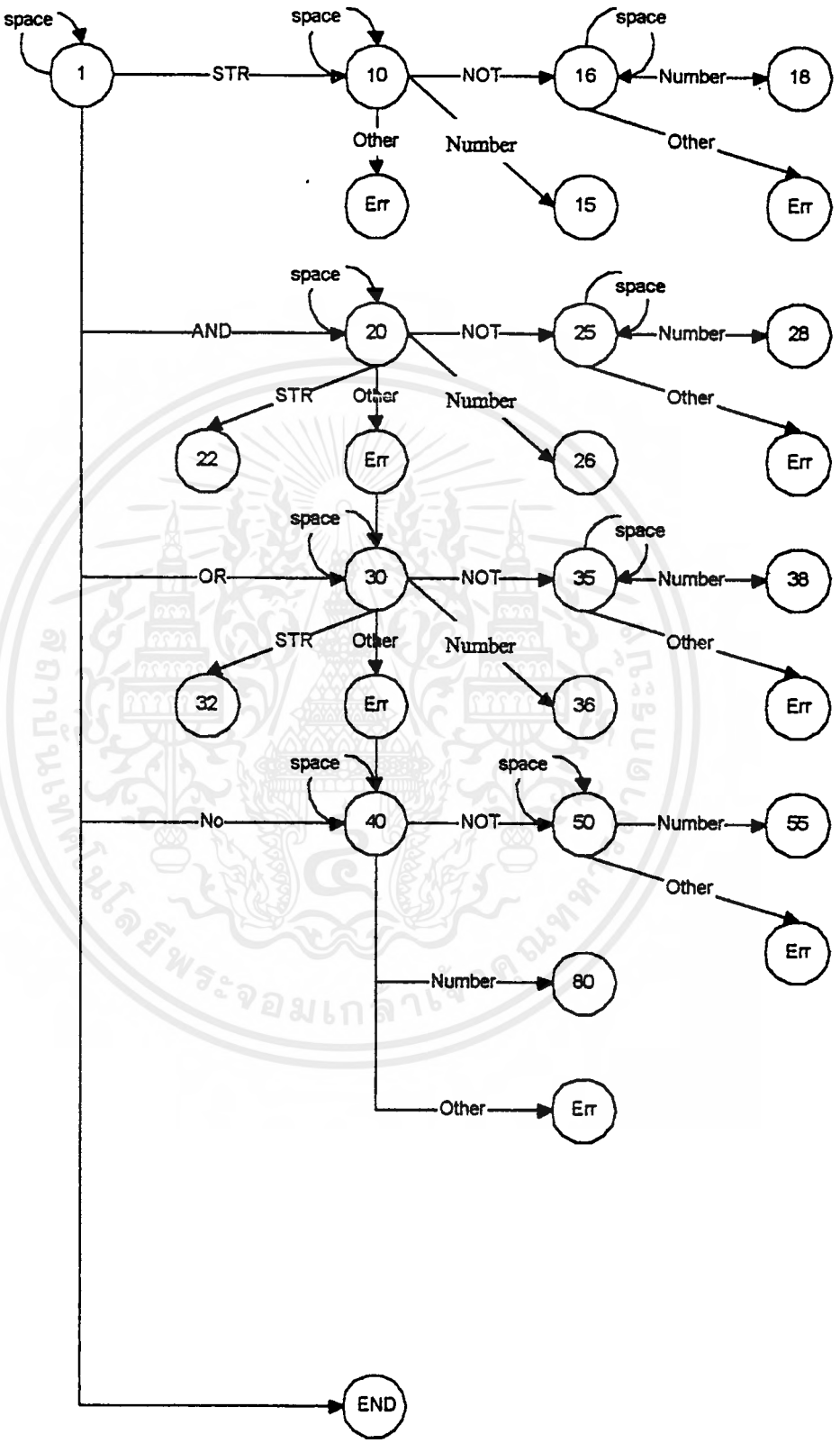
4.9 การทำงานของโปรแกรมแปลภาษามูลลีนในเครื่องคอมพิวเตอร์ส่วนบุคคล

โปรแกรมจะเรียกหาแฟ้มข้อมูลที่บรรจุในแฟ้มข้อมูลที่ผู้ใช้ระบุ อ่านแล้วนำมาแปลทีละบันทัดหากพบข้อผิดพลาดทางไวยากรณ์จะแจ้งให้ทราบทางจอภาพ และบันทึกข้อผิดพลาดนั้นลงแฟ้มข้อมูลชนิด .ERR แต่หากสามารถแปลได้ถูกต้องโปรแกรมจะสร้างแฟ้มข้อมูลที่บรรจุภาษาแอสเซมบลีของไมโครคอนโทรลเลอร์ MCS-51 ลงในแฟ้มข้อมูลชื่อเดียวกับแฟ้มต้นทางแต่ส่วนขยายเป็น .ASM จากนั้นจะเรียกโปรแกรมแปลภาษาแอสเซมบลี (assembler) มาแปลในเป็นแฟ้มข้อมูลรูปแบบของ Intel Hex File ซึ่งมีส่วนขยายเป็น .HEX ภายในประกอบตัวอักษรที่แทนระหัสปฏิบัติการของ ไมโครคอนโทรลเลอร์ MCS-51 (op-code) ซึ่งใช้จำลองการทำงานของพีแอลซีได้ แสดงแผนผังการทำงานของการแปลดังแผนผังรูปที่ 4.2



รูปที่ 4.2 แผนผังการทำงานของตัวแปลภาษาบูลลีน

4.10 การทำงานในสเตทแมทชีนเพื่อการแปลภาษาบูลลีน



รูปที่ 4.3 สเตทแมชชีนการตรวจไวยากรณ์ภาษาบูลลีน

จากการเรียกโปรแกรมย่อยเพื่อการตรวจสอบค้นหาคำสั่ง หรือตัวกระทำเมื่อพบประเภทของไวยากรณ์ใดจะส่งออกไปสู่โปรแกรมภายนอก โดยการทำงานเริ่มจากสเตทที่ 1

1. จะตรวจหาคำสั่ง STR หากพบคำสั่ง STR จะเปลี่ยน สเตทไปยังสเตทที่ 10 หากไม่ใช่คำสั่ง STR แต่หากเป็นช่องว่างจะวนการทำงานที่ สเตทเดิม หากไม่ใช่คำสั่ง STR หรือช่องว่างจะตรวจสอบหาคำสั่งต่อไปในข้อ 2

1.1 หากมีการเปลี่ยนสเตทมายังสเตทที่ 10 ในสเตทที่ 10 จะมีการตรวจหา คำสั่ง NOT หรือ ตัวกระทำที่เป็นตัวเลข แต่หากคำที่ได้เป็นช่องว่างจะวนทำงานที่สเตท 10 เหมือนเดิม หากเป็นตัวเลขจะเปลี่ยนสเตทไปยังสเตทที่ 15 หากเป็นคำสั่ง NOT จะเปลี่ยนสเตทไปยังสเตทที่ 16 หากไม่ได้เป็นทั้งสองอย่างแสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์ จะเปลี่ยนสเตทไปยัง err สเตทพร้อมให้รหัสความผิดพลาด

1.1.1 จาก สเตทที่ 16 จะมีการตรวจหาตัวกระทำที่เป็นตัวเลข หากเป็นช่องว่างจะวนทำงานที่สเตท 16 ไม่ใช่ทั้งช่องว่างตัวตัวเลข แสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์ หากเป็นตัวเลข เปลี่ยนสเตทไปยังสเตทที่ 18 ซึ่งสเตทที่ 18 นี้จะให้ค่าสำหรับอ้างอิงกับการแปล

2. จะตรวจหาคำสั่ง AND หากพบคำสั่ง AND จะเปลี่ยน สเตทไปยังสเตทที่ 20หากไม่ใช่คำสั่ง AND จะตรวจสอบหาคำสั่งต่อไปในข้อ 3.

2.1 หากมีการเปลี่ยนสเตทมายังสเตทที่ 20 ในสเตทที่ 20 จะมีการตรวจหา คำสั่ง NOT หรือ ตัวกระทำที่เป็นตัวเลข แต่หากคำที่ได้เป็นช่องว่างจะวนทำงานที่ สเตท 20เหมือนเดิมหากเป็นตัวเลขจะเปลี่ยนสเตทไปยังสเตทที่ 25 หากเป็นคำสั่ง NOT จะเปลี่ยนสเตทไปยังสเตทที่ 26 หากไม่ได้เป็นทั้งสองอย่างแสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์จะเปลี่ยนสเตทไปยัง err สเตทพร้อมให้รหัสความผิดพลาด

2.1.1 จากสเตทที่ 26 จะมีการตรวจหาตัวกระทำที่เป็นตัวเลข หากคำที่เข้ามาไม่ใช่ตัวเลขหรือช่องว่างแสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์ หากเป็นช่องว่างจะวนทำงานที่สเตทที่ 26 เหมือนเดิม หากเป็นตัวเลขเปลี่ยนสเตทไปยังสเตทที่ 28 ซึ่งสเตทที่ 28 นี้จะให้ค่าสำหรับอ้างอิงกับการแปล

3. จะตรวจหาคำสั่ง OR หากพบคำสั่ง จะเปลี่ยน สเตทไปยังสเตทที่ 30 หากไม่ใช่คำสั่ง OR จะตรวจสอบหาคำสั่งต่อไปในข้อที่ 4

3.1 หากมีการเปลี่ยนสเตทมายังสเตทที่ 30 ในสเตทที่ 30 จะมีการตรวจหา คำสั่ง NOT หรือ ตัวกระทำที่เป็นตัวเลข แต่หากคำที่ได้เป็นช่องว่างจะวนทำงานที่สเตท 30 เหมือนเดิม หากเป็นตัวเลขจะเปลี่ยนสเตทไปยังสเตทที่ 35 หากเป็นคำสั่ง NOT จะเปลี่ยน

สเตทไปยังสเตทที่ 36 หากไม่ได้เป็นทั้งสองอย่างแสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์จะเปลี่ยนสเตทไปยัง err สเตทพร้อมให้รหัสความผิดพลาด

3.1.1 จาก สเตทที่ 36 จะมีการตรวจหาตัวกระทำที่เป็นตัวเลขหากคำที่เข้ามาไม่ใช่ช่องว่างหรือตัวเลข แสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์ หากเป็นช่องว่างจะวนทำงานที่สเตทที่ 26 เหมือนเดิม หากเป็นตัวเลขจะเปลี่ยนสเตทไปยังสเตทที่ 38 ซึ่งสเตทที่ 38 นี้จะให้ค่าสำหรับอ้างอิงกับการแปล

4. จะตรวจหาคำสั่ง OUT หากพบจะเปลี่ยน สเตทไปยังสเตทที่ 40 หากไม่ใช่คำสั่ง OUT จะตรวจสอบหาคำสั่งต่อไปในข้อ 5

4.1 หากมีการเปลี่ยนสเตทมายังสเตทที่ 40 ในสเตทที่ 40 จะมีการตรวจหา คำสั่ง NOT หรือ ตัวกระทำที่เป็นตัวเลข อย่างใดอย่างหนึ่ง แต่หากคำที่ได้เป็นช่องว่างจะวนทำงานที่ สเตท 40 เหมือนเดิมหากไม่ได้เป็นทั้งหมดที่กล่าวมาแสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์จะเปลี่ยนสเตทไปยัง err สเตทพร้อมให้รหัสความผิดพลาด กรณีที่ เป็นคำสั่ง NOT จะเปลี่ยน สเตทไปยังสเตทที่ 50 หากเป็นตัวเลขจะเปลี่ยนสเตทไปยังสเตทที่ 80

4.1.1 หากมีการเปลี่ยนสเตทมาที่สเตทที่ 50 จะตรวจหาตัวกระทำที่เป็นตัวเลข หากคำที่เข้ามาไม่ใช่ช่องว่างหรือตัวเลข แสดงว่าคำที่ระบุเข้ามาไม่ถูกต้องตามไวยากรณ์หากเป็นช่องว่างจะวนทำงานที่สเตทที่ 50 เหมือนเดิม หากเป็นตัวเลขจะเปลี่ยนสเตทไปยังสเตทที่ 55 ซึ่งสเตทที่ 55 นี้จะให้ค่าสำหรับอ้างอิงกับการแปล

4.1.2 หากมีการเปลี่ยนสเตทมาที่สเตทที่ 80 จะให้อ้างอิงกับการแปลทันที

5. จะตรวจสอบหาเครื่องหมายเซมิโคลอน ซึ่งแสดงว่า ให้ข้ามการแปลในบรรทัดนั้นไปหากไม่เครื่องหมายนั้นจะตรวจหาคำสั่งต่อไปในข้อ 6

6. จะตรวจสอบหาคำสั่ง END (จบชุดคำสั่ง) หากพบจะสั่งให้หยุดการแปลทั้งหมด หากไม่พบแสดงว่าคำที่รับเข้ามาไม่ใช่ลักษณะไวยากรณ์ของที่ถูกต้อง

หมายเหตุ ระหว่างคำต่างๆ ที่อยู่บนบรรทัดแต่ละคำจะถูกคั่นด้วยช่องว่าง (space)

4.11 สรุปการทำงานการแปลคำสั่งโดยการทำงานของสเตทแมชชีน

เริ่มจากค้นหาคำสั่งในบรรทัดนั้นๆ จากนั้นค้นหาตัวกระทำซึ่งเป็นหมายเลข โดยโปรแกรมค้นหาคำซึ่งมีการทำงานเป็นลำดับ (state machiene) คล้ายวงจรซีเควนเชียล หลังจากที่สามารถเปลี่ยนแปลงสเตทไปยังสเตทสิ้นสุดของแต่ละกิ่ง ซึ่งสเตทดังกล่าวจะต้องไม่ใช่สเตทเออเรอร์ (Err state) ในสเตทสิ้นสุดของแต่ละกิ่งจะให้ค่าที่ใช้อ้างอิงกับการแปลที่แตกต่างกัน ซึ่งหากมีค่า

สัมพันธ์กับค่าใดก็จะนำชุดคำสั่งที่สัมพันธ์กันนั้นมาบรรจุไว้ในหน่วยความจำหากพบข้อผิดพลาดจะแจ้งออกทางหน้าจอให้ผู้ใช้รับทราบและบันทึกข้อผิดพลาดลงแฟ้มชนิด .ERR

4.12 แนวคิดในการสร้างชุดคำสั่งภาษาแอสเซมบลี

อาศัยการสร้างที่พักข้อมูลในระดับบิทในหน่วยความจำภายในชนิดแรมไว้ที่ตำแหน่ง 20H-22H (byte) เป็นหน่วยความจำบัฟเฟอร์การติดต่อกับ อินพุตและเอาต์พุตทุกครั้งของคำสั่งที่สร้างขึ้นจะติดต่อกับหน่วยความจำเหล่านี้ ดังนั้นในการสแกนแต่ละวงรอบจะเกิดการติดต่อกับหน่วยอินพุตและเอาต์พุตอย่างละหนึ่งครั้ง คือก่อนการสแกนคำสั่งทั้งหมดจะมีการอ่านอินพุตเข้ามาหนึ่งมาบันทึกลงในหน่วยความจำพักข้อมูลอินพุต หลังจากปฏิบัติการทางลอจิกที่ถูกกำหนดขึ้น จะอ่านข้อมูลจากเอาต์พุตบัฟเฟอร์ส่งออกไป หนึ่งครั้ง และจากการใช้พื้นที่หน่วยความจำแรมภายในเป็นพื้นที่พักข้อมูลทั้งหมดนี้ทำให้สามารถกำหนดการเข้าถึงแบบบิทของข้อมูลได้ ดังนั้นการกระทำลอจิกใดๆที่จำลองการทำงาน พีแอลซี จะอาศัยการกระทำระดับบิท และกำหนดให้ตัวกระทำหลักเป็น แฟลกตัวทด (carry flag) เช่น คำสั่ง STR 1 คำสั่งภาษา แอสเซมบลีที่ได้จากการแปลคือ mov c , 0 หมายถึง การโอนย้ายข้อมูลบิทที่ 0 มายัง แฟลกตัวทดหรือ คำสั่ง AND 5 คำสั่งภาษาแอสเซมบลีที่ได้จากการแปลคือ anl c , 4 หมายเลขหน้าสัมผัสจะต้องลบด้วย 1 เสมอเมื่ออ้างอิงเป็นหน่วยความจำภายในระดับบิท

ประเภทหน้าสัมผัส	หมายเลขหน้าสัมผัส	หมายเลขหน่วยความจำระดับบิท
อินพุต	1-8	0-7
รีเลย์ภายใน	9-16	8-15
เอาต์พุต	17-24	16-23

รูปที่ 4.4 ตารางความสัมพันธ์หน้าสัมผัสกับหน่วยความจำบัฟเฟอร์

บทที่ 5

สรุปผล และวิจารณ์

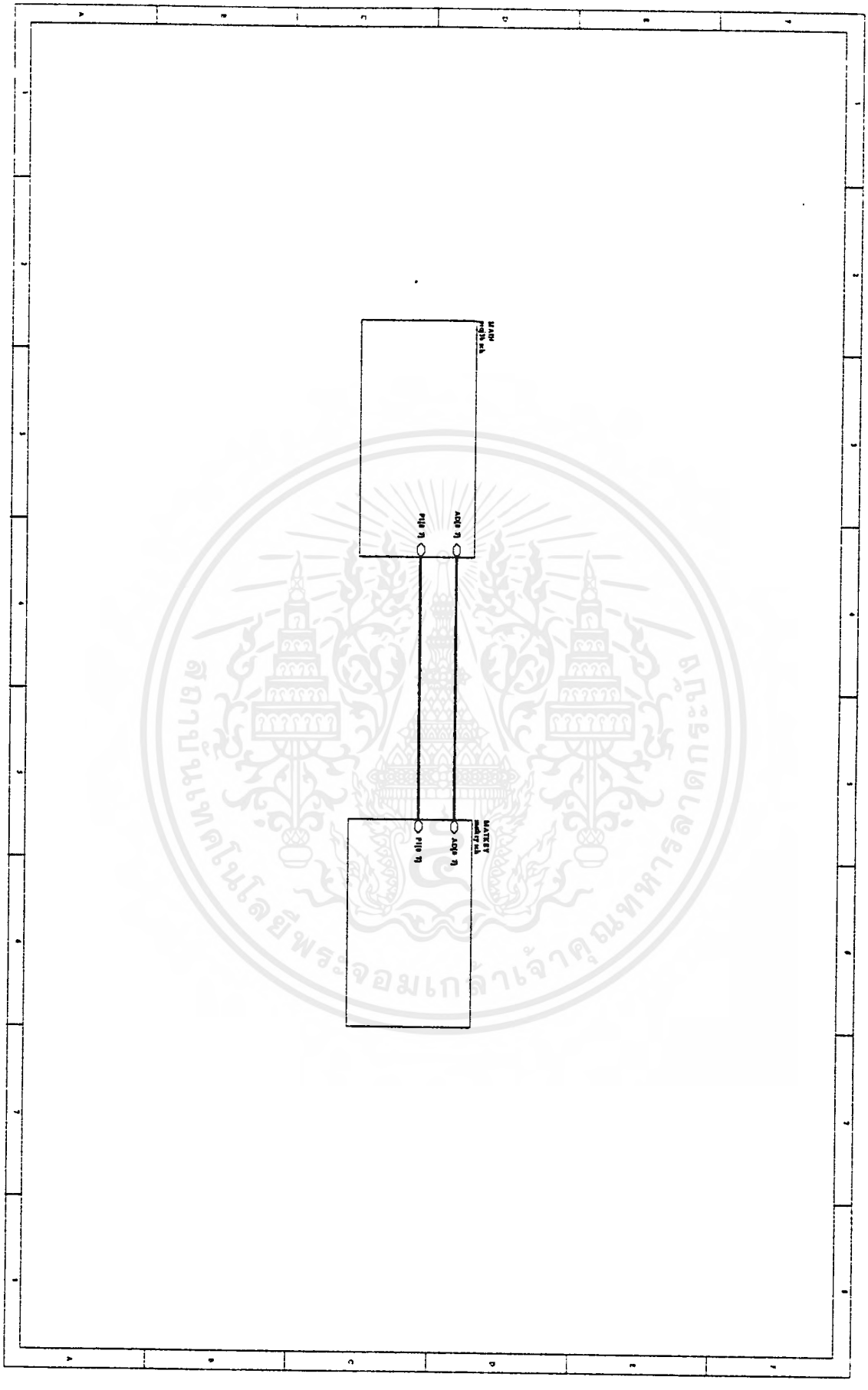
ปริญญานิพนธ์นี้ ได้จัดสร้างวงจรควบคุมโดยใช้ตัวประมวลผลกลาง AT89C52 พร้อมทั้งเขียนโปรแกรมเพื่อควบคุมการทำงานของวงจร โดยใช้ทฤษฎีการประยุกต์ใช้งานของไมโครคอนโทรลเลอร์ ตระกูล MCS-51 นอกจากนี้ยังจัดสร้างโปรแกรมตัวแปลภาษาบูลลีนขึ้น เพื่อใช้ในการแปลภาษาบูลลีนดังกล่าว ให้เป็นรหัสปฏิบัติการของไมโครคอนโทรลเลอร์ AT89C52 ซึ่งได้ใช้ทฤษฎีการสร้างคอมไพล์อย่างง่าย

ผลที่ได้จากปริญญานิพนธ์นี้ได้แสดงให้เห็นว่า การประยุกต์ใช้งานไมโครคอนโทรลเลอร์เพื่อจัดสร้างวงจรจำลองการทำงานของ พีแอลซี มีความเหมาะสม และสามารถใช้งานได้จริงในระดับหนึ่ง

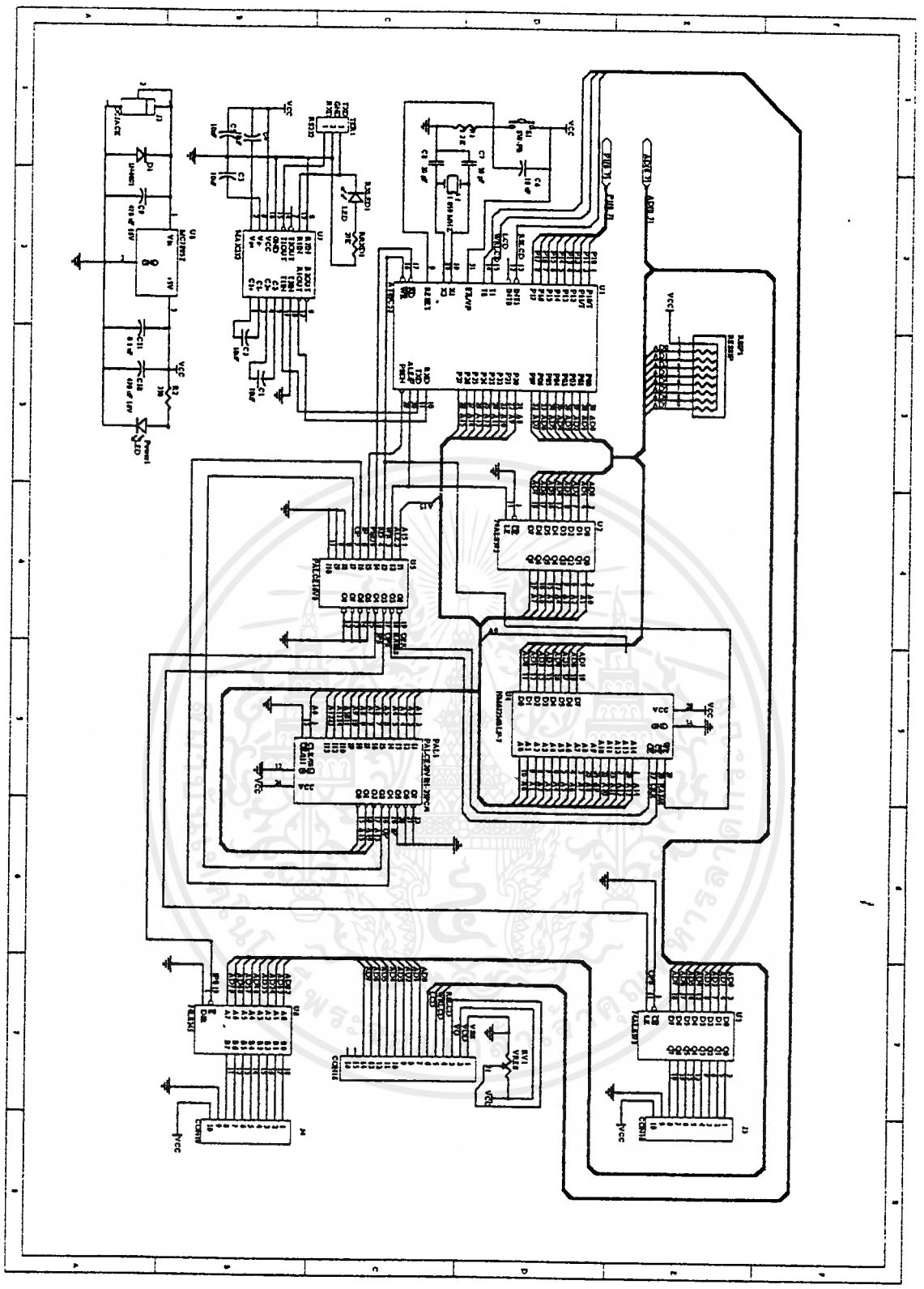
ดังนั้นหากได้รับการพัฒนาให้มีความสามารถด้านตัวนับ (Counter) และตัวจับเวลา (Timer) จะมีประโยชน์ในการประยุกต์ใช้งานได้สูงยิ่งขึ้น

ภาคผนวก

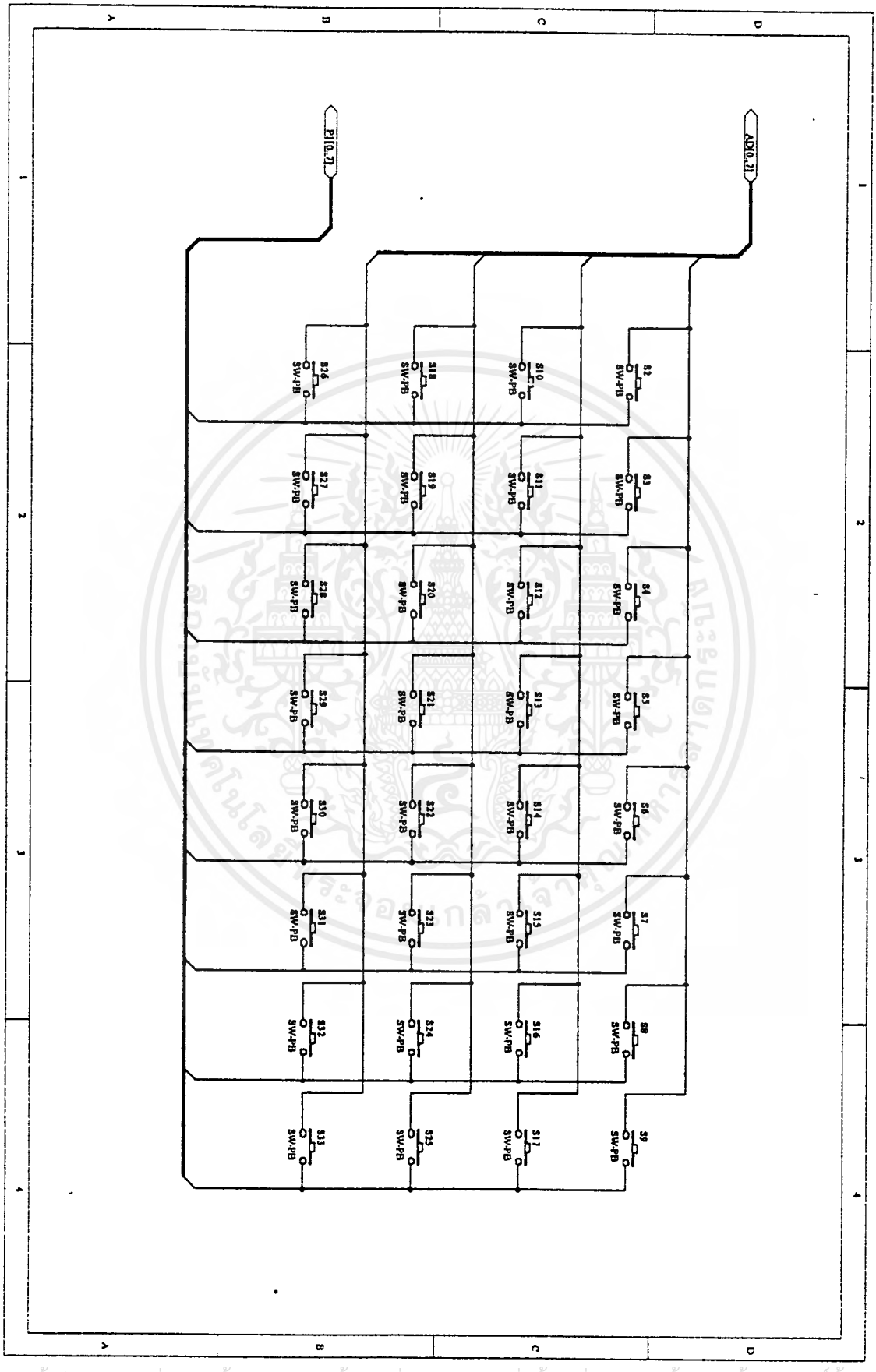
รายละเอียดวงจรชุดจำลอง พีแอลซี



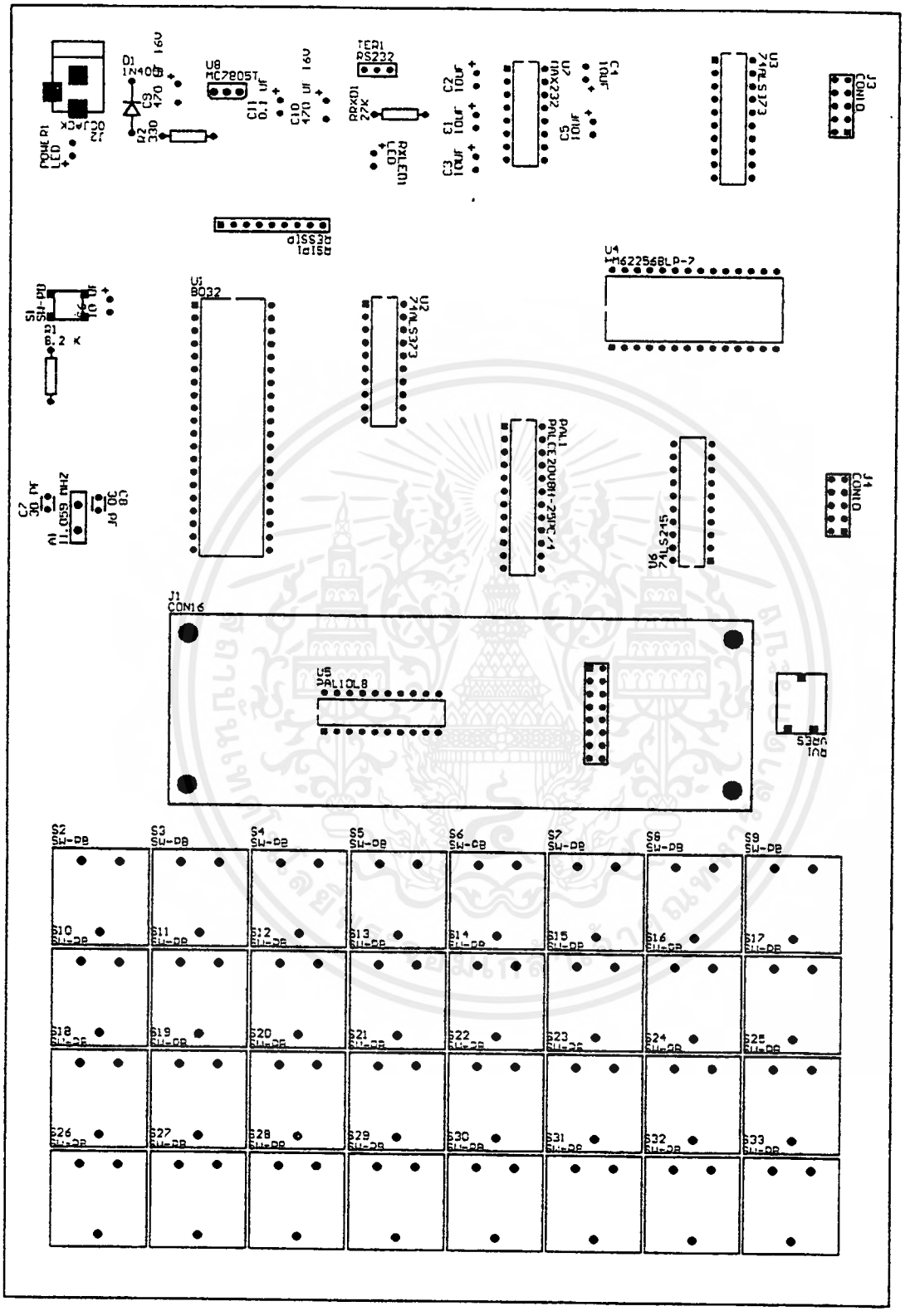
รูป A1.1 แผนผังวงจรหลัก ชุดจำลองพีแอลซี (main.sch)



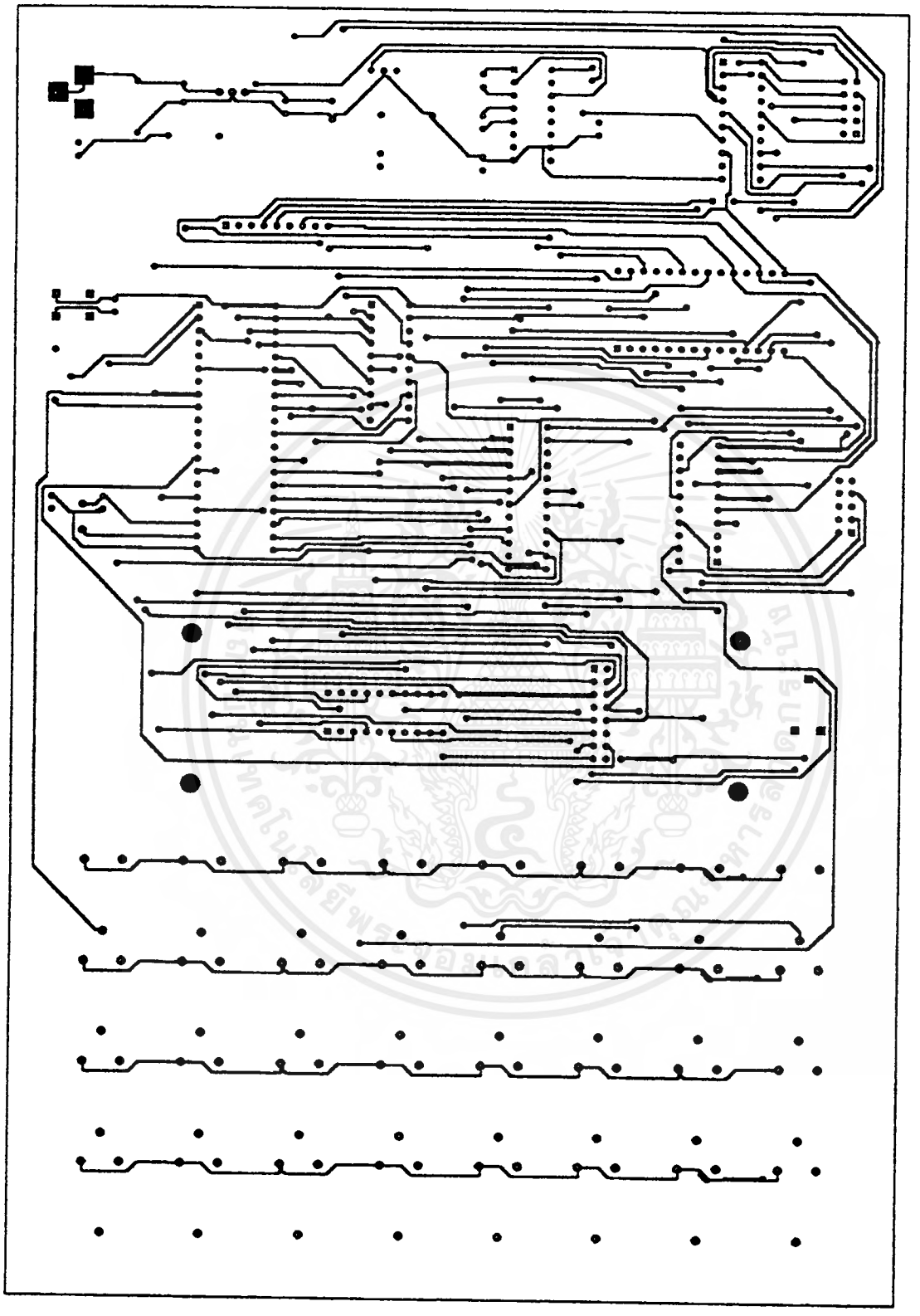
รูป A1.2 แผนผังวงจรส่วนควบคุมหลัก ชุดจำลองพีแอลซี (proj3b.sch)



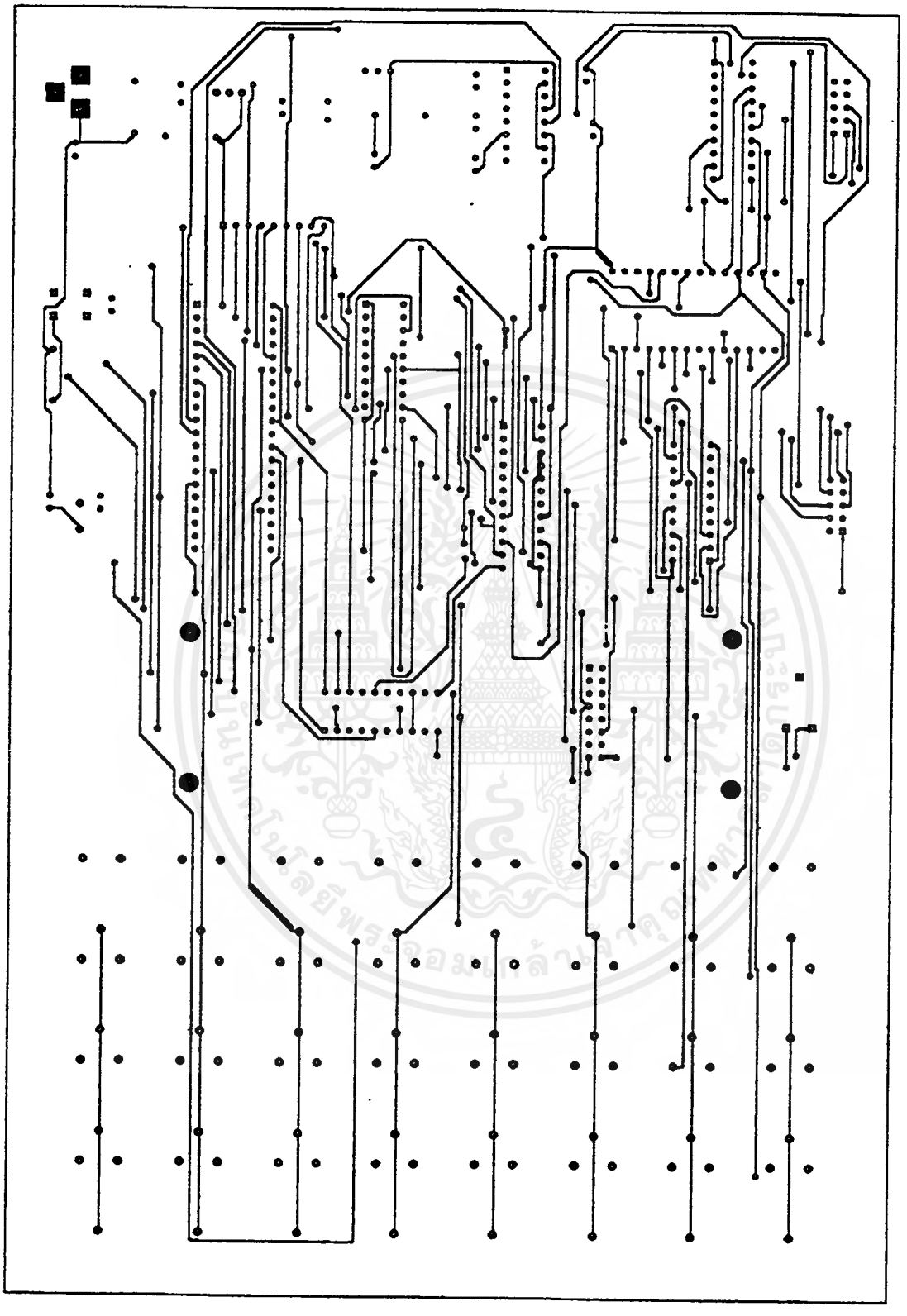
รูป A1.3 แผนผังวงจรส่วน แมททริกซ์คีย์ (matkey.sch)



รูป A1.4 การจัดวางอุปกรณ์ในแผ่นวงจร (scale 1 : 0.8)



รูป A1.5 ลายทองแดงด้านบนของแผ่นวงจร (scale 1 : 0.8)



รูป A1.6 ลายทองแดงด้านล่างของแผ่นวงจร (scale 1 : 0.8)

คู่มือการใช้งานชุดจำลอง พีแอลซี

คู่มือการใช้งานโครงงานชุดจำลองพีแอลซี

ก่อนอื่นผู้ใช้ต้องมีความเข้าใจในขั้นพื้นฐานของ พีแอลซีพอสมควร เช่น แลดเดอร์ไดอะแกรมคืออะไร อินพุต , เอาต์พุต และ รีเลย์ภายในของพีแอลซีมีลักษณะการทำงานรวมทั้งความรู้พื้นฐานด้านคอมพิวเตอร์

ก. ขั้นตอนการใช้งานของโครงงาน

แบ่งเป็น 2 ส่วน คือ ส่วนของโปรแกรมแปลภาษาบูลลีน และส่วนตัวเครื่อง ซึ่งมีลำดับขั้นตอนการใช้งานดังนี้

1. ส่วนของโปรแกรมแปลภาษาบูลลีน

1.1 ให้ผู้ใช้กำหนดเงื่อนไขในระบบที่จะใช้งาน

1.2 ให้ผู้ใช้เขียนแผนผังขั้นบันได (LADDER DIAGRAM) การกำหนดหมายเลขหน้าสัมผัสให้ดูตารางรูป A2.1 โดยตัวแปร XX และ YY เป็นหมายเลขของหน้าสัมผัส (XX มีย่านหมายเลขอยู่ที่ 1-24 และ YY มีย่านหมายเลขอยู่ที่ 9-24)

คำสั่ง	หมายเลขอินพุต (1-8)	หมายเลขรีเลย์ภายใน (9-16)	หมายเลขเอาต์พุต (17-24)
STR XX	ใช้ได้	ใช้ได้	ใช้ได้
STR NOT XX	ใช้ได้	ใช้ได้	ใช้ได้
AND XX	ใช้ได้	ใช้ได้	ใช้ได้
AND NOT XX	ใช้ได้	ใช้ได้	ใช้ได้
OR XX	ใช้ได้	ใช้ได้	ใช้ได้
OR NOT XX	ใช้ได้	ใช้ได้	ใช้ได้
OUT YY	ใช้ไม่ได้	ใช้ได้	ใช้ได้
OUT NOT YY	ใช้ไม่ได้	ใช้ได้	ใช้ได้
END	-	-	-

รูป A2.1 ตารางอ้างอิงการใช้หมายเลขหน้าสัมผัส

1.3 สร้างไฟล์ชุดคำสั่งภาษาบูลลีนซึ่งอ้างอิงจากแลดเดอร์ไดอะแกรมที่สร้างขึ้นในข้อ 1.2 โดยใช้เท็กซ์เอดิเตอร์ใด ๆ แล้วบันทึกลงในแฟ้มข้อมูลชนิด .PLC ซึ่งถือว่าเป็นไฟล์ต้นทาง (ต้องใส่คำสั่ง END เป็นคำสั่งสุดท้าย)

1.4 เรียกโปรแกรมตัวแปลภาษาบูลลีน ตามด้วยชื่อไฟล์ต้นทาง โดยไม่ต้องใส่นามสกุล เช่น explc demo1 แล้วกด enter (ซึ่ง demo1 คือไฟล์ต้นทาง)

1.5 หากแปลได้ถูกต้องจะได้ไฟล์ 2 ไฟล์ ชื่อเดียวกับไฟล์ต้นทาง แต่นามสกุลเป็น .ASM และ .HEX

1.6 หากการแปลพบความผิดพลาดใดๆ จะปรากฏข้อความบนหน้าจอเครื่องคอมพิวเตอร์ส่วนบุคคล และข้อความเหล่านั้นยังถูกบันทึกลงในแฟ้มข้อมูลอ้างอิงความผิดพลาด ซึ่งมีชื่อเดียวกับไฟล์ต้นทางแต่ส่วนขยาย เป็น .ERR เมื่อแก้ไขความผิดพลาดของไฟล์ต้นทางแล้วให้ทำการคอมไพล์ใหม่

หมายเหตุ กรณีที่ต้องการใส่คำอธิบายโปรแกรม (comment) ให้ใส่เครื่องหมายเซมิคลอน (;) หน้าคำอธิบาย

หลังจากเสร็จสิ้นขั้นตอนทั้งหมดของข้อ 1 แล้ว จะได้รหัสปฏิบัติการอยู่ในแฟ้มข้อมูลชนิด .HEX ซึ่งจะต้องส่งให้ตัวเครื่องเพื่อทำการรัน (RUN) ต่อไป โดยปฏิบัติตามขั้นตอนทั้งหมดในข้อ 2

2. การใช้งานตัวเครื่อง

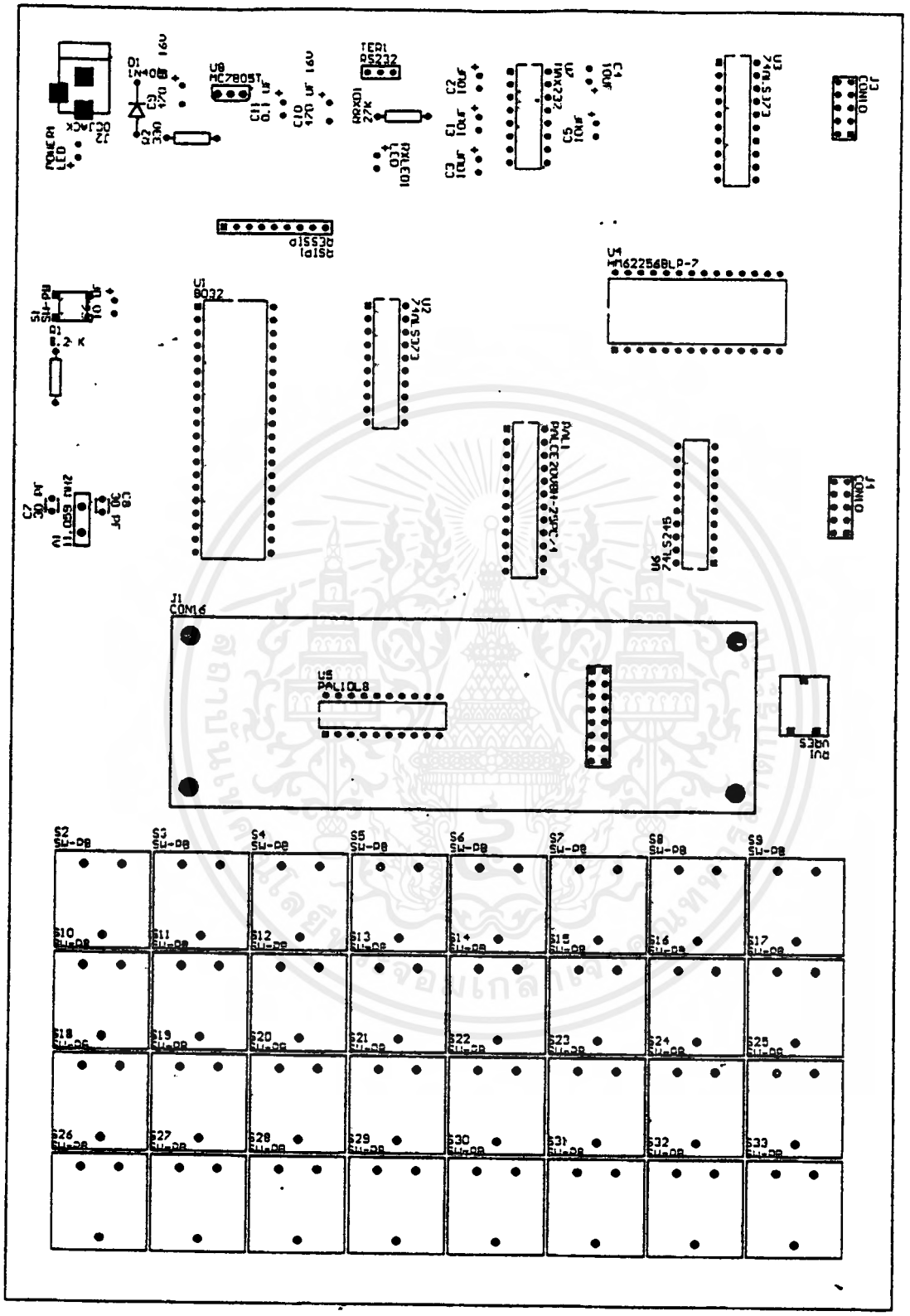
โปรแกรมที่ใช้ในการสื่อสารสำหรับสั่งงานชุดจำลองพีแอลซี คือโปรแกรม xtalk.exe โหมดการทำงานของโปรแกรม xtalk ที่ใช้มี 2 โหมดด้วยกัน ดังนี้คือ

1. โหมดการสื่อสาร 1 ตัวอักษร คือหากมีการกดแป้นพิมพ์ใด ๆ ของเครื่องคอมพิวเตอร์ส่วนบุคคล รหัสแอ็สกีของแป้นพิมพ์นั้นจะถูกส่งออกไปทางพอร์ตอนุกรม ซึ่งการเรียกโปรแกรม xtalk ทุกครั้งในสภาวะเริ่มแรกของการรันโปรแกรม xtalk นี้จะอยู่ในโหมดนี้ก่อนเสมอ

2. โหมดคอมมานด์ หรือ โหมดคำสั่ง ซึ่งมีคำสั่งต่าง ๆ หลายคำสั่ง แต่ที่ใช้ในที่นี้ คือ คำสั่งสำหรับโอนย้ายข้อมูลแฟ้มข้อมูลส่งออกไปทางพอร์ตอนุกรม (ส่งแฟ้มข้อมูล) โดยใช้คำสั่ง SEnd ของ xtalk

การทำงานของตัวเครื่องชุดจำลองพีแอลซีมี 3 โหมด คือ

1. โหมดมอนิเตอร์รอรับการสั่งงานจากผู้ใช้ผ่านโปรแกรมสื่อสาร xtalk (การสั่งงานชุดจำลองพีแอลซีใช้โหมดการสื่อสาร 1 ตัวอักษรของ xtalk)
2. โหมดรอรับ (Receive mode)
3. โหมดการปฏิบัติการ (Running mode)



ลำดับขั้นตอนการใช้งานและสั่งงานตัวเครื่องมีดังนี้

2.1 เชื่อมต่อวงจรชุดจำลอง พีแอลซี กับวงจรภายนอกผ่านคอนเน็คเตอร์โดยอุปกรณ์ที่เป็นเอาต์พุตเชื่อมต่อกับคอนเน็คเตอร์ J3 และ และอุปกรณ์ที่เป็นอินพุตเชื่อมต่อกับคอนเน็คเตอร์ J4 (พิจารณารูป A1.4)

2.2 เชื่อมต่อการสื่อสารโดยใช้สาย link ที่ให้มากับโครงงาน ระหว่างวงจรและเครื่องคอมพิวเตอร์ส่วนบุคคลผ่านพอร์ตอนุกรม (COM1 หรือ COM2) อาจเชื่อมต่อโดยตรงกับ 9DB ของเครื่องคอมพิวเตอร์ส่วนบุคคลเลย หรือ ใช้ร่วมกับ 25DB to 9DB converter ที่ให้มาพร้อมสาย link

2.3 เรียกโปรแกรมการสื่อสาร xtalk บนเครื่องคอมพิวเตอร์ส่วนบุคคล(PC) โดยใช้คำสั่ง X1 หรือ X2 ตามหมายเลขพอร์ตอนุกรมที่ใช้ PC จะรันโปรแกรม xtalk อยู่ในโหมดการสื่อสาร 1 ตัวอักษรก่อนเป็นอันดับแรก

2.4 จ่ายพลังงานให้กับเครื่อง จนเครื่องอยู่ในสภาวะพร้อม (แสดง READY ที่หน้าจอ LCD ของชุดจำลองพีแอลซี และ เครื่องหมายคำถาม (?) ที่หน้าจอของพีซี กรณีที่หน้าจอพีซีไม่แสดง เครื่องหมายคำถามให้ดูการแก้ปัญหาการสื่อสารของชุดจำลองพีแอลซี)

2.5 กดแป้นพิมพ์ตัวL (ตัวแอลใหญ่ หมายถึง Load) ของเครื่องคอมพิวเตอร์ส่วนบุคคลเพื่อสั่งให้ตัวเครื่องชุดจำลองพีแอลซี (PLC EMULATOR) เข้าสู่โหมดการรอรับข้อมูล (receive mode) ซึ่งที่หน้าจอเครื่องคอมพิวเตอร์ส่วนบุคคลมีข้อความ " Please send program... " ปรากฏขึ้น

2.6 กดแป้นพิมพ์ Ctrl + A ของเครื่องคอมพิวเตอร์ส่วนบุคคลเพื่อเข้าสู่คอมมานด์โหมดของ xtalk

2.7 ที่คอมมานด์โหมดของ xtalk ใช้คำสั่ง se filename.HEX เพื่อส่งรหัสที่ได้จากการแปลไปยังตัวเครื่อง เช่น se demo1.hex หลังจาก xtalk โอนย้ายข้อมูลเฮ็กซ์ไฟล์เรียบร้อยแล้วจะกลับสู่โหมด xtalk จะกลับเข้าสู่การสื่อสารโหมด 1 ตัวอักษรอีกครั้ง (ที่หน้าจอของเครื่องคอมพิวเตอร์ส่วนบุคคลจะปรากฏข้อความว่า " Download success " และปรากฏเครื่องหมายคำถามใต้ข้อความ) แต่หากการโอนย้ายแฟ้มข้อมูลเกิดความผิดพลาดจะแสดงข้อความที่จอ LCD และหน้าจอพีซีว่า " Download Error ! " ก่อนกลับเข้าสู่ monitor mode ของชุดจำลองพีแอลซี

2.8 กดแป้นพิมพ์ G (ตัวจีใหญ่ หมายถึง Go หรือ run) ของเครื่องคอมพิวเตอร์ส่วนบุคคลเพื่อสั่งให้ตัวเครื่องชุดจำลองพีแอลซี (PLC EMULATOR) เข้าสู่โหมดการเข้าสู่สภาวะการรันโปรแกรมซึ่งได้รับการดาวน์โหลดจากโดยผ่านโปรแกรม xtalk ในข้อ 2.7 โปรแกรมจะเริ่มรันเป็นวงรอบที่ไม่สิ้นสุด

2.9 กดสวิทช์รีเซ็ท (S1ในรูปที่ A1.4) บนตัวเครื่อง เมื่อต้องการหยุดการรันโปรแกรม เครื่องจะกลับสู่สภาวะพร้อมใหม่ หากต้องการรันโปรแกรมเดิมให้ทำตามข้อ 2.6 แต่หากต้องการให้เครื่องทำงานโปรแกรมอื่นที่แปลไว้แล้วโดยโปรแกรมแปลภาษาบูลลีน (explc) ให้ทำตามตั้งแต่ข้อ 2.3

หมายเหตุ 1. ควรทำการสำเนาแฟ้มข้อมูลในแผ่นดิสค์ PLCEM ลงในไดเร็คทอรีปัจจุบันที่ใช้ในการคอมไพล์ และใช้เรียกโปรแกรมสื่อสาร xtalk ก่อน

2. การออกจากโปรแกรม xtalk ให้เข้าสู่คอมมานด์โหมด โดยกด Ctrl + A แล้วใช้คำสั่ง QUit โดยพิมพ์ QU แล้วกด enter

3. เมื่อมีการแก้ไขชุดคำสั่งภาษาบูลลีนในแฟ้มข้อมูลชนิด .PLC (source file) ทุกครั้งหลังจากแก้ไขแล้ว ต้องทำการคอมไพล์ใหม่ด้วยเพื่อให้รหัสปฏิบัติการมี ความเป็นปัจจุบัน

4. หากผู้ใช้ถนัดการใช้โปรแกรมการสื่อสารโปรแกรมอื่นนอกจาก xtalk เช่น procomplus ก็สามารถใช้แทนโปรแกรม xtalk ได้

ข. การแก้ปัญหาการสื่อสารของชุดจำลองพีแอลซี

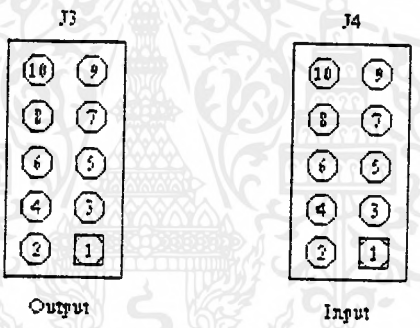
1. เมื่อจ่ายพลังงานให้กับชุดจำลองพีแอลซี ตามขั้นตอน 2.1 - 2.4 แล้ว ที่หน้าจอของเครื่องคอมพิวเตอร์ส่วนบุคคล (PC) ไม่ปรากฏเครื่องหมายคำถามแสดงว่าการสื่อสารระหว่างชุดจำลองพีแอลซี และเครื่องคอมพิวเตอร์ส่วนบุคคลยังไม่เกิดขึ้น ซึ่งอาจเนื่องมาจากสาเหตุที่เครื่องคอมพิวเตอร์แต่ละเครื่องมีการต่อพอร์ตอนุกรมไม่เหมือนกัน ซึ่งโดยปกติพอร์ตอนุกรมแบบ 9DB จะถูกต่อให้เป็น COM1 และ 25DB จะถูกต่อให้เป็น COM2 ดังนั้นตามปกติหากสาย link ของ ชุดจำลองพีแอลซีต่อกับ พอร์ตอนุกรมแบบ 9DB การเรียกใช้โปรแกรมสื่อสาร xtalk จะใช้คำสั่ง X1 หากสาย link ของชุดจำลองพีแอลซีต่อกับพอร์ตอนุกรมแบบ 25DB การเรียกใช้โปรแกรมสื่อสาร xtalk จะใช้คำสั่ง X2 แต่หากการเรียกโปรแกรมตามลักษณะดังกล่าวแล้วการสื่อสารระหว่าง PC และ PLC emulator ยังไม่เกิดขึ้นให้ทดลองสลับการเรียกใช้โปรแกรมสื่อสารดู คือ หากสาย link ของชุดจำลอง

พีแอลซีต่อกับ พอร์ตอนุกรมแบบ 25DB การเรียกใช้โปรแกรมสื่อสาร xtalk จะใช้คำสั่ง X1 และหาก หากสาย link ของชุดจำลองพีแอลซีต่อกับ พอร์ตอนุกรมแบบ 9 DB การเรียกใช้โปรแกรมสื่อสาร xtalk จะใช้คำสั่ง X2 แทน

2. หากการสื่อสารระหว่าง PC และ PLC emulator ยังไม่เกิดขึ้นให้ทดลอง shut down ระบบปฏิบัติการ windows 95 (หรือ version ที่สูงกว่า) เข้าสู่ MS-DOS mode แล้วทดลองเชื่อมต่อการสื่อสารอีกครั้งตามที่กล่าวมาในข้อ 1

ค. การเชื่อมต่ออุปกรณ์ภายนอกกับหน่วยอินพุท เอ้าต์พุต

การเชื่อมต่อจะเชื่อมต่อผ่าน connector 10 pins J3 และ J4 (พิจารณารูป A1.4) โดยกำหนดให้สัญญาณที่เข้ามาเชื่อมต่อต้องเข้ากันได้กับไอซีชนิด TTL สำหรับขาและหมายเลขหน้าสัมผัสให้พิจารณาจากรูป A2.2

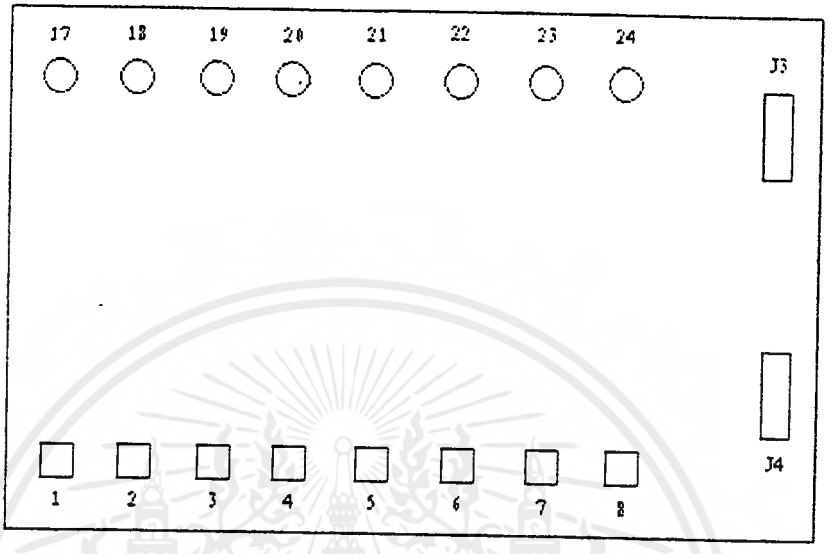


คอนเน็คเตอร์	ขาที่	หมายเลขหน้าสัมผัส	หมายเหตุ
J3	1-8	17-24	Output contact
J3	9	-	GND
J3	10	-	VCC 5 volts
J4	1-8	1-8	Input contact
J4	9	-	GND
J4	10	-	VCC 5 volts

รูป A2.2 ตารางข้อมูลสำหรับการเชื่อมต่อกับอุปกรณ์ภายนอก

ง. ตัวอย่างสาธิตการใช้งานชุดจำลองพีแอลซี

วงจรที่จัดสร้างขึ้นเพื่อเป็นการสาธิตการใช้งานโครงงาน ชุดจำลองพีแอลซี เป็นวงจรที่ต่อมต่อกับหน่วยอินพุต และเอาต์พุตโดยตรง ผ่านคอนเน็คเตอร์ J4 (I/P) และ J3 (O/P) โดยมีการจัดวางอุปกรณ์เปรียบเทียบกับหมายเลขหน้าสัมผัส ดังรูป A2.3



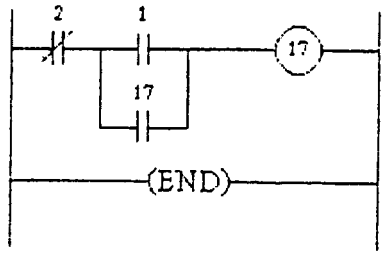
รูป A2.3 การจัดวางอุปกรณ์ในวงจรสาธิตเปรียบเทียบกับหมายเลขหน้าสัมผัส

เมื่อ 1-8 เป็นหน้าสัมผัสอินพุต ต่อกับสวิตช์กดติดปล่อยดับ และ 17-24 เป็น ตัวแสดงผลแบบ แอลอีดี

<u>ตัวอย่างที่ 1</u> สมมติว่าต้องการให้สวิตช์ 1 (หมายเลขหน้าสัมผัส 1) เป็นสวิตช์สำหรับสตาร์ทมอเตอร์ซึ่งต่ออยู่กับหน้าสัมผัสที่ 17 และสวิตช์ 2 เป็นสวิตช์ที่ใช้สต็อปมอเตอร์ โดยมีการทำงานคือ สภาวะเริ่มแรกมอเตอร์ไม่มีการหมุน(OFF) เมื่อมีการกดสวิตช์สตาร์ทมอเตอร์จะเริ่มหมุน และยังคงหมุนต่อไปจนกว่าจะมีการกดสวิตช์สต็อป

<u>วิธีทำ</u>

จากแนวทางการทำงานที่ได้กล่าวมาสามารถเขียนเป็นแลดเดอร์ไดอะแกรมได้ดังแสดงในรูป A2.4



รูป A2.4 แลดเดอร์ไดอะแกรมวงจรสตาร์ทมอเตอร์

จากแลดเดอร์ไดอะแกรมเขียนเป็นภาษาบูลลีน บันทึกลงแฟ้มข้อมูล DEMO1.PLC ได้ดังนี้

```
;-------------------------------------------------------------------------------;
; Filename      demo1.plc
; Description   start/stop  motor circuit
; Date          11/2/1998
; By            s9013280  (Mr. Prayad Chotipattarakool)
;-------------------------------------------------------------------------------;

    str not 2
    str 1
    or  17
    and str
    out 17
    end
```

เมื่อผู้ใช้ทราบรายละเอียดของซอร์สไฟล์ภาษาบูลลีนแล้วกระทำตามขั้นตอนต่อไปนี้

1. ขั้นตอนการเตรียมการใช้งาน

1.1 ให้ผู้ใช้สร้างไดเร็คทอรีใหม่ ชื่อว่า PLCEM ขึ้นมาในไดรฟ์ที่ต้องการเช่น ไดรฟ์ C (ไม่ควรใช้งานจากแผ่น PLCEM โดยตรง)

1.2 ให้ผู้ใช้ทำการสำเนา (copy) แฟ้มข้อมูลทุกแฟ้มจากไดรฟ์ A (หรือ B) ลงในไดเร็คทอรีที่สร้างขึ้นในข้อ 1.1 เมื่อสำเนาเสร็จสิ้นจะปรากฏแฟ้มข้อมูลหนึ่งชื่อ DEMO1.PLC ในไดเร็คทอรีที่สร้างขึ้น ซึ่งภายในแฟ้มบรรจุชุดคำสั่งภาษาบูลลีนที่แสดงข้างต้น

2. ขั้นตอนการเริ่มการใช้งาน

2.1 ให้ผู้ใช้เรียก MS-DOS prompt (หากใช้ระบบปฏิบัติการ windows 95 ขึ้นไป) หรือหากเป็นไปได้ให้ shut down เข้า MS-DOS mode เลย แต่หากใช้ระบบปฏิบัติการ MS-DOS อยู่แล้วให้ข้ามขั้นตอนในข้อ 2.1 นี้ไป

2.2 เปลี่ยนไดเร็คทอรีไปยังไดเร็คทอรีที่สร้างขึ้นโดยใช้คำสั่ง CD\PLCEM

2.3 เชื่อมต่อแผงวงจรสาธิตเข้ากับชุดจำลองพีแอลซีที่คอนเน็คเตอร์ J3 และ J4 พิจารณารูป A2.3 และ รูป A1.4 โดยเชื่อมต่อระหว่าง J3 กับ J3 และ J4 กับ J4

2.4 ต่อสาย link ระหว่างชุดจำลองพีแอลซี (TER1 ในรูป A1.4) เข้ากับพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ส่วนบุคคล (PC)

2.5 คอมไพล์โปรแกรมภาษาบูลลีน โดยใช้คำสั่ง explc DEMO1 (DEMO1 เป็นชื่อแฟ้มชุดคำสั่งภาษาบูลลีนซึ่งมีอยู่แล้วเนื่องจากการสำเนามา)

2.6 เรียกโปรแกรม xtalk โดยใช้คำสั่ง X1 หรือ X2 ตามหมายเลขพอร์ตอนุกรมที่ใช้ (COM1 ใช้คำสั่ง X1 COM2 ใช้คำสั่ง X2)

2.7 จ่ายพลังงานให้ชุดจำลองพีแอลซี (เสียบอแดปเตอร์ให้กับบอร์ด)

2.8 สังเกตหน้าจอของ PC จะปรากฏเครื่องหมายคำถาม ('?') ออกมา 1 ตัว หากไม่เป็นดังนี้ ให้แก้ปัญหาตามวิธีการในหัวข้อการแก้ปัญหาการสื่อสารชุดจำลองพีแอลซี

2.9 ทดลองกดแป้นพิมพ์ enter หลาย ๆ ครั้ง หน้าจอของ PC จะปรากฏข้อความว่า " Invalid Command " ตามจำนวนครั้งของการกดคีย์ enter

2.10 ให้กดแป้นพิมพ์ 'L' (ตัวแอลใหญ่ หมายถึง Load) 1 ครั้ง หน้าจอของ PC จะปรากฏข้อความว่า " Please send program... "

2.11 ให้กดแป้นพิมพ์ Ctrl+A (กด Ctrl ค้างไว้แล้วกดตัว A จากนั้นปล่อยคีย์ทั้งสอง)

2.12 พิมพ์คำสั่ง (ในบรรทัดล่างสุด)ว่า se demo1.hex แล้วกด enter (SE หมายถึง send)

2.13 หน้าจอ PC จะปรากฏเครื่องหมาย '&' หลายตัวและปรากฏข้อความใต้บรรทัดของเครื่องหมาย '&' ว่า " Download success " แต่หากปรากฏข้อความว่า " Download error !!! " ให้ทำตามข้อ 2.10 - 2.12 ใหม่

2.14 ให้กดแป้นพิมพ์ 'G' (ตัวจีใหญ่ หมายถึง Go) เพื่อรันโปรแกรม DEMO1

2.15 ให้ทดลองกดสวิตช์หมายเลข 1 LED หมายเลข 17 จะติดและค้างอยู่อย่างนั้น

2.16 ให้ทดลองกดสวิตช์หมายเลข 3 . 4 . 5 . 6 . 7 . 8 ดูตามลำดับ จะไม่มีผลต่อ LED หมายเลข 17

2.17 ให้ทดลองกดสวิตช์หมายเลข 2 LED จะดับลง

2.18 ให้ทดลองกดสวิตช์หมายเลข 1 อีกครั้ง LED หมายเลข 17 จะติดและค้างอีกครั้ง ทั้งนี้เนื่องจากเป็นไปตามเงื่อนไขที่ตั้งขึ้น

2.19 ให้กดสวิตช์ S2 (ดูรูป A1.4) เพื่อยุติการรันโปรแกรม DEMO1

2.20 ให้ทดลองกดสวิตช์หมายเลข 1-8 จะไม่มีผลต่อ LED หมายเลข 17 เพราะโปรแกรมถูกยกเลิกการรันไปแล้ว

หมายเหตุ ขั้นตอนเตรียมการใช้งานที่ปฏิบัติตามข้อ 1 ทั้งหมดเป็นขั้นตอนที่ปฏิบัติเพียงครั้งเดียว ต่อไปหากผู้ใช้ทดลองสร้างโปรแกรมชุดคำสั่งภาษาบูลลีนเอง หรือทดสอบโปรแกรมตัวอย่าง DEMO2 ที่ได้จากการสำเนาจากไดรฟ์ A เช่นกันสามารถปฏิบัติตามข้อ 2 ได้เลยโดยข้ามขั้นตอนที่ 1 ทั้งหมดไป

ตัวอย่างที่ 2 เป็นการสร้างวงจรบวกเลขไบนารีขนาด 4 บิท สองจำนวนมาบวกด้วยกัน โดยอาศัยหลักของวงจรบวกเลขทางดิจิตอล คือ การสร้างสมการบวกแบบฮาล์ฟแอดเดอร์ร่วมกับการบวกแบบฟูลแอดเดอร์ สมการการบวกแสดงดังนี้

sum1 = !A1*B1 + A1*!B1

C1 = A1*B1 (carry 1)

temp1 = !A2*B2 +A2*!B2

sum2 = !temp1*C1 + temp1*!C1

C2 = A2*B2 + temp1*C1 (carry 2)

temp2 = !A3*B3 +A3*!B3

sum3 = !temp2*C2 + temp2*!C2

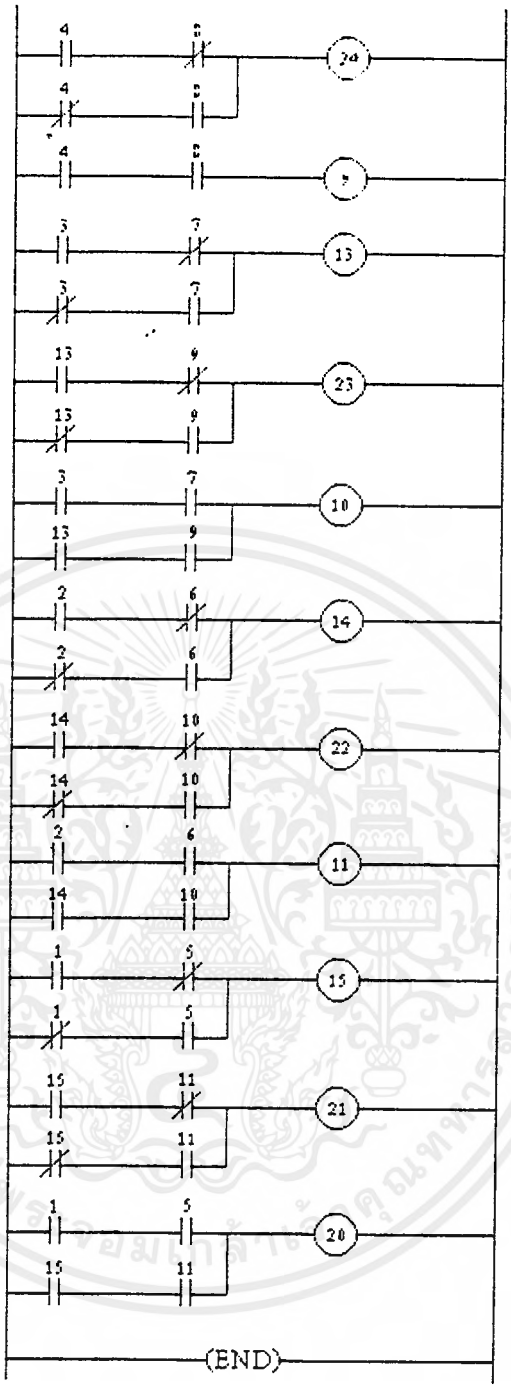
C3 = A3*B3 + temp2*C2 (carry 3)

temp3 = !A4*B4 +A4*!B4

sum4 = !temp3*C3 + temp3*!C3

C4 = A4*B4 + temp3*C3 (carry 4)

จากสมการทั้งหมดจะเห็นว่าทั้งหมดอาศัยหลักการของวงจรฮาล์ฟแอดเดอร์ เพราะวงจรฟูลแอดเดอร์ ก็ประกอบด้วยวงจรฮาล์ฟแอดเดอร์สองวงจร คือใช้บวกค่าในหลักนั้น1 วงจร และบวกผลลัพธ์ที่ได้ (temp?) กับตัวทดจากหลักที่แล้วอีกหนึ่งวงจร (temp?+C?) แต่การบวกในหลักแรกเป็นการบวกแบบไม่คิดตัวทด สำหรับตัวแปร temp1 , temp2 และ temp3 กำหนดขึ้นเพื่อให้สอดคล้องกับเขียน แลดเดอร์ไดอะแกรม และโปรแกรมภาษาบูลลีน สามารถทำได้ง่ายขึ้นเพราะสมการแต่ละสมการจะมีตัวแปรอยู่เพียงแค่สองตัวแปร ส่วนหลักการบวกก็เป็นหลักการบวกเลขทางดิจิตอลธรรมดา จากสมการทั้งหมดด้านบนสามารถเขียนเป็นแลดเดอร์ไดอะแกรมได้ดังรูป A2.5



รูป A2.5 แลดเดอร์ไดอะแกรมของวงจรบวกเลขไบนารี 4 บิท

จากแอดเดรสไดอะแกรมหน้าสัมผัสอินพุต (สวิตช์อินพุต) แบ่งออกเป็น 2 ชุด คือ ชุดของตัวตั้ง (1-4 มีรหัสเป็น 8 , 4 , 2 ,1 ตามลำดับ)และชุดตัวบวก (5-8 มีรหัสเป็น 8 , 4 , 2 ,1 ตามลำดับ) ส่วนของเอาต์พุตใช้หน้าสัมผัสหมายเลข 24 , 23 , 22 , 21 แสดงผลบวกในรูปแบบรหัส 8 , 4 , 2 , 1 ตามลำดับ และ หน้าสัมผัสหมายเลข 20 แสดงตัวทดที่เกิดจากการบวกกันทั้ง 4 หลัก หน้าสัมผัสที่นำมาช่วยในการบวกคือ 13 เป็น temp1 , 14 เป็น temp2 , 15 เป็น temp3 ในสมการการบวกหน้าสัมผัสที่ใช้เก็บตัวทดการบวกคือ 9 , 10 , 11 เป็นตัวเก็บตัวทดในการบวก พิจารณารูป A2.6 ประกอบ

หมายเลขหน้าสัมผัส	หน้าที่	หมายเหตุ
1	ตัวตั้ง	Weight = 8
2	ตัวตั้ง	Weight = 4
3	ตัวตั้ง	Weight = 2
4	ตัวตั้ง	Weight = 1
5	ตัวบวก	Weight = 8
6	ตัวบวก	Weight = 4
7	ตัวบวก	Weight = 2
8	ตัวบวก	Weight = 1
9	ตัวทด 1	C1
10	ตัวทด 2	C2
11	ตัวทด 3	C3
12	-	-
13	หน้าสัมผัสช่วย	temp1
14	หน้าสัมผัสช่วย	temp2
15	หน้าสัมผัสช่วย	temp3
20	ตัวทดผลบวกทั้งหมด	Weight = 16
21	ผลบวก 4	Weight = 8
22	ผลบวก 3	Weight = 4
23	ผลบวก 2	Weight = 2
24	ผลบวก 1	Weight = 1

รูป A2.6 หน้าที่ของหน้าสัมผัสในการจำลองวงจรบวกเลขไบนารี 4 บิท

จากข้อมูลที่กล่าวมาทั้งหมดนำมาเขียนเป็นชุดคำสั่งภาษาบูลีนได้ดังนี้

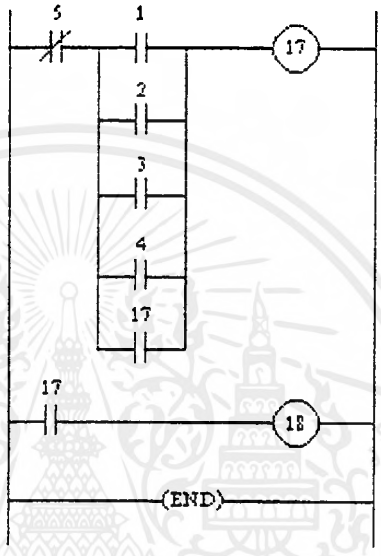
```
;-------------------------------------------------------------------------------;
: Filename      DEMO2.PLC
: Description   EMULATE  FOUR BIT BINARY ADDER
; Date          10/29/1998                                                     :
;-------------------------------------------------------------------------------;


        str not 4     ;!A1*B1 + A1*!B1
        and 8
        str 4
        and not 8
        or str
        out 24        ;sum 1 to o/p

        str 4         ;A1*B1
        and 8
        out 9         ;C1 ( carry 1 to internal relay )


        str not 3     ;!A2*B2 + A2*!B2
        and 7
        str 3
        and not 7
        or  str
        out 13        ;temp1 to internal relay 13

        str not 13    ;!temp1*C1 + temp1*!C1
        and 9
        str 13
        and not 9
        or  str
```

```
out 23      ;sum 2  to o/p

str 3       ;A2*B2 + temp1*C1
and 7
str 13
and 9
or  str
out 10      ;carry 2 to internal relay

str not 2   ;!A3*B3 + A3*!B3
and 6
str 2
and not 6
or  str
out 14      ;temp2 to internal relay 14

str not 14  ;!temp2*C2 + temp2*!C2
and 10
str 14
and not 10
or  str
out 22      ;sum 3 to o/p

str 2       ;A3*B3 + temp2*C2
and 6
str 14
and 10
or  str
out 11      ;carry 3 to internal relay

str not  1  ;!A4*B4 + A4*!B4
and 5
```

```
str 1
and not 5
or  str
out 15        ;temp3 to internal relay 15

str not 15    ;!temp3*C3 + temp3*!C3
and 11
str 15
and not 11
or  str
out 21        ;sum 4 o/p

str 1         ;A4*B4 + temp3*C3
and 5
str 15
and 11
or  str
out 20        ;carry 4 to o/p
end
```

เมื่อได้แฟ้มชุดคำสั่งภาษาบูลลีนแล้วให้กระทำตามขั้นตอนเช่นเดียวกับที่กล่าวมาแล้วในตัวอย่างที่ 1

<u>ตัวอย่างที่ 3</u> ให้จำลองการทำงานของสัญญาณกันขโมยรถยนต์ คือ เมื่อเปิดประตูใดประตูหนึ่งใน 4 ประตูของรถยนต์ซึ่งติดตั้ง sensor ไว้สัญญาณเตือนจะดังขึ้นและค้างไว้ และไฟหน้าของรถจะติดและค้างไว้ จนกว่าจะมีการกดสวิตช์ยกเลิก เมื่อให้ sensor ทั้ง 4 ตัว แทนด้วยสวิตช์ หมายเลข 1-4 และสวิตช์หมายเลข 5 แทนสวิตช์รีเซ็ท หน้าสัมผัสเอาต์พุตหมายเลข 17 แทนสัญญาณเตือนภัย และ หน้าสัมผัสเอาต์พุตหมายเลข 18 แทนสวิตช์ไฟหน้าของรถยนต์จากรายละเอียดทั้งหมด สามารถเขียนแลดเดอร์ไดอะแกรมดังรูป A2.7



รูป A2.7 แลดเดอร์ไดอะแกรมจำลองวงจรสัญญาณกันขโมย

จากแลดเดอร์ไดอะแกรมจะเขียนเป็นชุดคำสั่งภาษาบูลลีนได้ดังนี้

```
str not 5
str     1
or      2
or      3
or      4
or      17
and     str
out     17
str     17
out     18
end
```

ให้ผู้ใช้ทดลองพิมพ์โปรแกรมดังกล่าวแล้วบันทึกลงแฟ้มข้อมูลชื่อ ALARM.PLC แล้วให้ทดลอง compile และส่งรหัสปฏิบัติการที่ได้ลงไปทดสอบรันดูในชุดจำลองพีแอลซี โดยกระทำตามขั้นตอนในหัวข้อที่ 1 และ 2

หมายเหตุ การใช้คำสั่ง out ในโปรแกรมจะเห็นว่า เมื่อต้องการ out ข้อมูลจากที่เดียวกันไปยังเอาต์พุตสองจุด ต้องใช้การ ป้อนกลับของสัญญาณโดยใช้คำสั่ง str ก่อน (บรรทัดที่ 8-10 ของโปรแกรม) แล้วจึงใช้คำสั่ง out ได้อีกครั้ง และโปรแกรมนี้ไม่ได้ให้ซอร์สไฟล์มาเพื่อให้ผู้ใช้ทดลองสร้างซอร์สไฟล์เอง

ชุดคำสั่งของโครงงาน และการใช้งาน

ชุดคำสั่งของโครงงานที่จัดสร้างขึ้นและการใช้งาน

ภายในแผ่นดิสค์ PLCEM ซึ่งแนบส่งพร้อมกับโครงงาน บรรจุแฟ้มข้อมูลที่เป็นชุดคำสั่งที่สร้างขึ้น และใช้สำหรับการพัฒนาต่อไปดังนี้

1. srcomled.asm เป็นโปรแกรม monitor ที่ใช้กับ AT89C52 อยู่ในไดเร็คทอรีย่อย 51 ซึ่งใช้กับตัวแปล SXA51.EXE

2. ชุดคำสั่งภาษาซี เป็นโปรแกรมคอมไพล์ภาษาบูลลีนของผู้ใช้ อยู่ในไดเร็คทอรีย่อย C การคอมไพล์ใช้คอมไพเลอร์ของเทอร์โบซีเวอร์ชั่น 3 ขึ้นไป ซึ่งมีขั้นตอนเตรียมการคอมไพล์ดังนี้

2.1 สำเนาแฟ้ม token.c และ tokfile.c ไปยังไดเร็คทอรีของแฟ้มข้อมูลต้นทางของเทอร์โบซี (source directory)

2.2 สำเนาแฟ้ม token.h และ tokfile.h ไปยังไดเร็คทอรีอินคลูดของเทอร์โบซี (include directory)

2.3 สร้างโปรเจ็คโดยใช้เมนูโปรเจ็ค แล้วใช้คำสั่ง new จัดการตั้งชื่อว่า explc.prj

2.4 เพิ่มแฟ้มให้กับโปรเจ็ค (Add) ดังนี้ คือ token.c และ tokfile.c

2.5 จัดการบันทึกโปรเจ็ค จะได้แฟ้มข้อมูล explc.prj

2.6 หากต้องการคอมไพล์แฟ้ม explc ให้กดแป้นพิมพ์ F9 จะได้แฟ้ม explc.exe เหมือนกับที่อยู่ใน root directory ของแผ่นดิสค์ PLCEM

หมายเหตุ การดำเนินการด้านโปรเจ็คไฟล์ อาจหาทราบรายละเอียดจากหนังสือภาษาซีทั่วไป

หน้าที่การทำงานฟังก์ชั่นในโปรแกรมแปลภาษาบูลลีน

โปรแกรมแปลภาษาบูลลีน ซึ่งสร้างจากภาษาซีประกอบด้วยแฟ้มชุดคำสั่ง 2 แฟ้มคือ token.c และ tokfile.c ซึ่งประกอบด้วยฟังก์ชั่นต่าง ๆ ที่มีหน้าที่ดังนี้

1. save_str สร้างชุดคำสั่งภาษาแอสเซมบลีเพื่อเก็บรักษาข้อมูลในแฟลกตัวทด ทำให้ใช้คำสั่ง str ครั้งที่ 2 ได้
2. restore_str สร้างชุดคำสั่งภาษาแอสเซมบลีเพื่อดึงข้อมูลของแฟลกตัวทดที่เก็บรักษาไว้กลับคืนมา
3. contact_renge_check ตรวจสอบค่าตัวเลขว่าอยู่ในย่านของหน้าสัมผัสทั้งหมดหรือไม่
4. output_reang_check ตรวจสอบค่าตัวเลขว่าอยู่ในย่านของหน้าสัมผัสเอาต์พุตหรือไม่
5. and_str สร้างชุดคำสั่งภาษาแอสเซมบลีเพื่อเลียนแบบการทำงานของคำสั่งภาษาบูลลีน and_str โดยนำแฟลกตัวทดตัวเก่าที่เก็บไว้กับตัวทดปัจจุบันมา แอนด์กัน

6. or_str สร้างชุดคำสั่งภาษาแอสเซมบลีเพื่อเลียนแบบการทำงานของคำสั่งภาษาบูลลีน or_str โดยนำแฟลกตัวหดตัวเก่าที่เก็บไว้กับตัวหดปัจจุบันมา ออร์กัน
7. gen_head สร้างชุดคำสั่งภาษาแอสเซมบลีที่เป็น header ซึ่งไม่เป็นคำสั่งที่เกี่ยวข้องกับการแปลภาษาบูลลีนโดยตรง
8. gen_state15 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 15
9. gen_state18 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 18
10. gen_state25 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 25
11. gen_state28 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 28
12. gen_state35 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 35
13. gen_state38 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 38
14. gen_state50 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 50
15. gen_state80 สร้างชุดคำสั่งภาษาแอสเซมบลีซึ่งอ้างอิงกับการทำงานในสเตทที่ 80
16. gen_end สร้างชุดคำสั่งภาษาแอสเซมบลีที่เป็น ส่วนลงท้ายโปรแกรมที่สร้างขึ้น ซึ่งไม่เป็นคำสั่งที่เกี่ยวข้องกับการแปลภาษาบูลลีนโดยตรง
17. get_token จะทำหน้าที่ระบุประเภทของคำต่างๆ (token) ที่ตัวแปลภาษาบูลลีน ตีความออกมาว่าเป็น คำชนิดใด ตัวแปลภาษารู้จักหรือไม่
18. word_sep จะทำหน้าที่แยกคำต่าง ๆ ในบรรทัดที่อ่านเข้ามาออกจากกันทีละคำเพื่อส่งให้ฟังก์ชั่น get_token ตีความและเปลี่ยนสเตทตามประเภทของคำ
19. read_line จะทำหน้าอยู่ข้อมูลจาก source file เข้ามาทีละ 1 บรรทัด

ข้อมูลบางประการในไฟล์ส่วนหัวภาษาซี

header file ที่สร้างขึ้น 2 ไฟล์ คือ token.h และ tokfile.h ภายในประกอบด้วยข้อมูลดังนี้

token.c

1. ค่าคงที่ระบุ ชนิดของคำ
2. ค่าคงที่ระบุประเภทคำ (token) ซึ่งสัมพันธ์กับหมายเลขของสเตทที่จะต้องมีการเปลี่ยนไป
3. ค่าคงที่ที่ระบุความผิดพลาด ซึ่งใช้บอกสาเหตุความผิดพลาดในการแปล

tokfile.h

1. ส่วนกำหนดสตรัคเจอร์ที่ใช้เป็นตัวแปลซึ่งเก็บค่าที่ใช้ในการแปลเช่นประเภทของคำ (token type) และ ชนิดของคำ
2. ต้นแบบของฟังก์ชั่นที่ใช้ในแฟ้ม tokfile.c

รายละเอียดของ แฟ้มส่วนหัว tokfile.h

```
#include <token.h>
typedef struct {
    int  word_type;
    int  index;
    int  token_type;
    int  line_no;
    int  number;
    int  contact_num;
    int  timer_num;
    int  reg_displace;
    int  Loop;
    char  string_line[100];
    char  string_sep[100];  }words; //end struct

extern void word_sep(words *word);
extern char* read_line(FILE *Source_File,words *word);
extern void get_token(words *word);
```

รายละเอียดของแฟ้มส่วนหัว token.h

```
# define    Space_type     50
# define    Comment_type   60
# define    Letter_type    70
#define     EOL            5
// word define

#define              TRUE          1
#define     FALSE          0
#define     End_of_file    30

#define     Token_space    600
#define     Token_number   700

#define     Token_STR      10
#define     Token_and      20
#define     Token_or       30
#define     Token_out      40
#define     Token_tim      92
#define     Token_cnt      100
#define     Token_not      110
#define     Token_L        120
#define     Token_U        130

#define     Token_end      200
#define     Token_unknow   300

#define     TokenEOL       400
#define     Token_comment  500
#define     TokenEOF       0
//token define

#define     STR_required           10
#define     Reg_required           20
#define     Num_required               30
```

รายละเอียดของแฟ้มโปรแกรม tokfile.c

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <token.h>
#include <tokfile.h>

 int Current_type;
 int Old_type;

void get_token(words *word)
{  int i;
    word_sep(word);
    switch(Old_type)     //word type of what ?
    {
        case Space_type :
          (*word).token_type=Token_space;
        break;

        case Comment_type :
          (*word).token_type=Token_comment;
        break;

        case EOL :
          (*word).token_type=TokenEOL;
        break;

    /*  case Empty_type :
         (*word).token_type=Token_space;
        break;*/

    case End_of_file :
          (*word).token_type=TokenEOF;
    break;
```

```
default :
      strupr((*word).string_sep);

     //if( (*word).number = atoi((*word).string_sep)!=0)
     //i=atoi((*word).string_sep);
     if( ( (*word).number = atoi((*word).string_sep) ) !=0)
       (*word).token_type=Token_number;
     else

     if(!strcmp( (*word).string_sep,"STR"))
      (*word).token_type=Token_STR;
     else

     if(!strcmp( (*word).string_sep,"AND"))
      (*word).token_type=Token_and;
     else

     if(!strcmp( (*word).string_sep,"OR"))
      (*word).token_type=Token_or;
     else

     if(!strcmp( (*word).string_sep,"OUT"))
      (*word).token_type=Token_out;
     else

     if(!strcmp( (*word).string_sep,"TIM"))
      (*word).token_type=Token_tim;
     else

     if(!strcmp( (*word).string_sep,"CNT"))
      (*word).token_type=Token_cnt;
     else

     if(!strcmp( (*word).string_sep,"NOT"))
      (*word).token_type=Token_not;
```

```
            else

            if(!strcmp( (*word).string_sep,"END"))
              (*word).token_type=Token_end;
            else
              (*word).token_type=Token_unknow;

          break;

      }

}

void word_sep(words *word)
{ char ch_ptr= '@';
  int i = 0,Loop=1;

  Old_type = (*word).word_type;
   do

    { ch_ptr=(*word).string_line[(*word).index];
        switch (ch_ptr)
        {  case ';':
             (*word).word_type=Comment_type;
             Loop=0;
           break;

           case ' ':
            if(Old_type==Space_type)
             { (*word).string_sep[i] = ch_ptr;
                (*word).index++          ;
                 i++;
                 (*word).string_sep[i] = '\0';
             }
```

```
            Current_type=Space_type;
            (*word).word_type=Space_type;
           break;
//add
/*         case '\t':
            if(Old_type==Space_type)
             { (*word).string_sep[i] = ' ' ;
                  (*word).index++          ;
                  i++;
                  (*word).string_sep[i] = '\0';
             }

            Current_type=Space_type;
            (*word).word_type=Space_type;
           break;*/
//add
           case '\n':
            (*word).word_type=EOL;
            Loop=0;
           break;
           case '\0':
            (*word).word_type=End_of_file;
            Loop=0;
           break;
           default :
            if(Old_type==Letter_type)
             { (*word).string_sep[i] = ch_ptr;
                  (*word).index++    ;
                  i++;
                  (*word).string_sep[i] = '\0';
             }
            Current_type=Letter_type;
            (*word).word_type=Letter_type;
       }
    }while ( (Loop) && (Old_type==Current_type) && (*word).index <85);
```

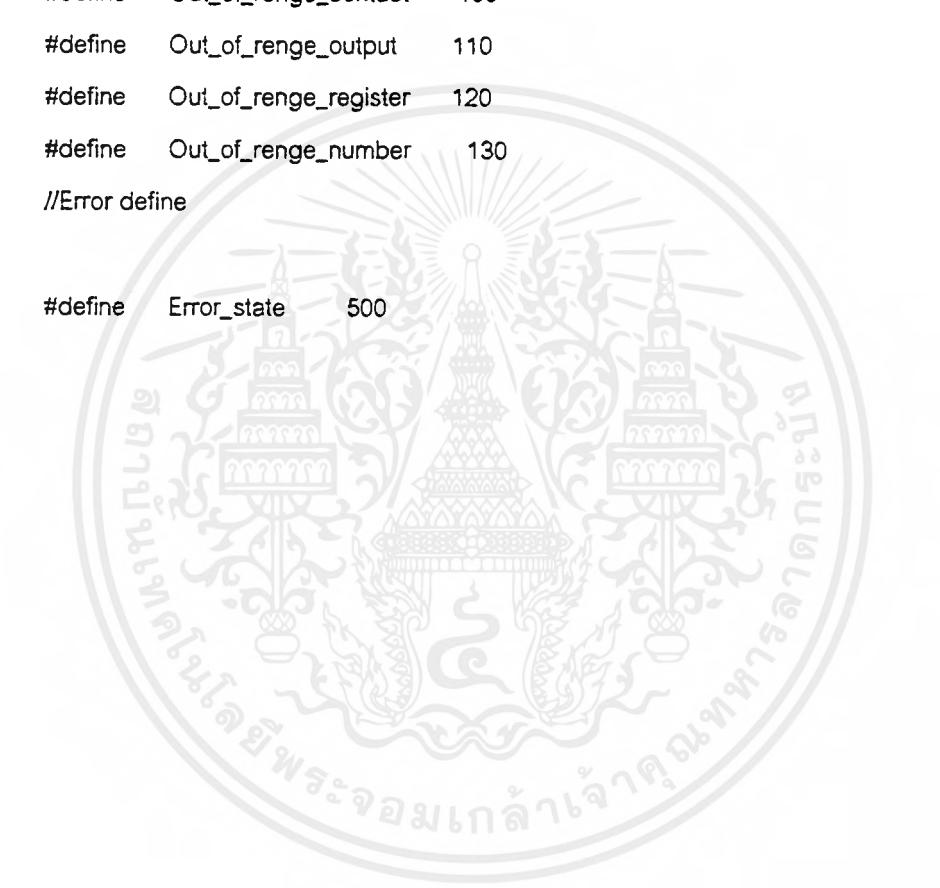
```
}
char* read_line(FILE *Source_File,words *word)
{ char* CH;
strcpy((*word).string_line,"\0\0\0\0\0\0\0\0\0\0\0\0\0\0");
 CH = fgets((*word).string_line,85,Source_File);
 if(CH!=NULL)
  (*word).line_no++;
 return(CH);
}
```

รายละเอียดของแฟ้มโปรแกรม token.c

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <process.h>
#include <token.h>
#include <tokfile.h>


void save_str(FILE* Des_File);
void restore_STR(FILE* Des_File);

int contact_renge_check(words *word);
int output_renge_check(words *word);



void and_str(FILE *Des_File);
void or_str(FILE *Des_File);

void gen_head(FILE* Des_File);
void gen_state15(FILE *Des_File,words *word);
void gen_state18(FILE *Des_File,words *word);

void gen_state25(FILE *Des_File,words *word);
void gen_state28(FILE *Des_File,words *word);

void gen_state35(FILE *Des_File,words *word);
void gen_state38(FILE *Des_File,words *word);

void gen_state55(FILE *Des_File,words *word);
void gen_state80(FILE *Des_File,words *word);

void gen_end(FILE* Des_File);

void main(int argc,char *argv[])
```

```
{
 words word;
 FILE  *Source_File,*Des_File,*Err_File;
 int   Next_state,STR_count=0,Error_code=0,End_status=0,Error_status=0;
 char  *Loopc;
 char  source_file[30];
 char  des_file[30];
 char  err_file[30];

 clrscr();
 printf("\n\n\t\t\tKMITL PLC EMULATE V1.0 COMPILER\n");
 if (argc <2)
 { printf("\n\n\t\t\t[Command line...]   EXPLC filename\n\a");
  exit(0);
 }
 strcpy(source_file,argv[1]);
 strcat(source_file,".plc");

 strcpy(des_file,argv[1]);
 strcat(des_file,".asm");

 strcpy(err_file,argv[1]);
 strcat(err_file,".err");


if((Source_File=fopen(source_file,"r"))==NULL)
  { printf("\t\t\tCan not open %s \a\n",source_file);
    printf("\n\n\t\t\t[Command line...]   EXPLC filename\n");
    exit(1);
  }
if((Des_File=fopen(des_file,"wt"))==NULL)
  { printf("\t\t\tCan not create %s\a",des_file);
    printf("\t\t\tCheck write protect & disk space please !\a");
    exit(1);
  }
```

```
#define     End_required            40
#define     Toomany_STR             50
#define     Toofew_STR              55
#define     Not_or_contact_required    60
#define     Not_or_str_contact_required 65
#define     Contact_required        70
#define     Out_param_required      80
#define     Instruct_error          90
#define     Out_of_renge_contact     100
#define     Out_of_renge_output      110
#define     Out_of_renge_register    120
#define     Out_of_renge_number      130
//Error define

#define     Error_state       500
```



```
if((Err_File=fopen(err_file,"wt"))==NULL)
  { printf("\t\t\tCan not create %s \a",err_file);
    printf("\t\t\tCheck write protect & disk space please !\a");
  }


gen_head(Des_File);
word.token_type=Token_space;
word.line_no=0;
while(!feof(Source_File) && word.token_type!=Token_end )

{ /*each branch have to terminate when success or error*/

  Loopc=read_line(Source_File,&word);
  word.index=0;
  Next_state=1;

  word.word_type=Space_type;
  while(Loopc)
    { switch(Next_state){
        case 1: { // state 1

                  get_token(&word);
                  switch(word.token_type){

                  case Token_comment :
                        Loopc=0;
                        Next_state=1;
                  break;

                  case TokenEOF :
                        Error_code=End_required;
                        Loopc=0;
```

```
                Next_state=Error_state;
            break;

            case TokenEOL :
                Loopc=0;
                Next_state=1;
            break;

            case Token_space :
//          (stay at this state (state 1)  )
            break;

            case Token_STR :
                switch(STR_count){

                  case 0:
                    Next_state=10;
                  break;

                  case 1:
//                save_str if success
                    Next_state=10;
                  break;

                  default :
                    Error_code=Toomany_STR;
                    Next_state=Error_state;
                }

            break;

            case Token_and :
                if(STR_count<1){
                  Error_code=STR_required;
                  Next_state=Error_state;
```

```
        }
        else
          Next_state=20;
break;

case Token_or :
        if(STR_count<1){
          Error_code=STR_required;
          Next_state=Error_state;
        }
        else
          Next_state=30;
break;

case Token_out :
        if(STR_count<1){
          Error_code=STR_required;
          Next_state=Error_state;
        }
        else
          Next_state=40;
break;

case Token_tim :
  if (STR_count <1){
          Error_code=STR_required;
          Next_state=Error_state;
  }
  else
          Next_state=90;
break;

case Token_cnt :
  if (STR_count <1){
          Error_code=STR_required;
```

```
                    Next_state=Error_state;
                }
                else
                    Next_state=100;
            break;

            case Token_end :
                End_status=1;   //find end instruction
                Loopc=0;
            break;

            default :

// Not , Number and other word that unknow
                Error_code = Instruct_error;
                Next_state = Error_state;
            } //end of  switch Token_type

        }break; //end case of Next_state = state 1

/*********************STR INSTRUCTION************************/

        case 10: {
        get_token(&word);
          switch(word.token_type){
          case Token_space:
                //Next_state=Next_state
          break;

          case Token_not:
                Next_state=16;
          break;

          case Token_number:
```

```
            Next_state=15;
        break;

        default:
            Error_code=Not_or_contact_required;
            Next_state=Error_state;
        }
    }break;         //end case of state 10

    case 15:            //subcase of case 10 (STR)
     if (contact_renge_check(&word)){
        if (STR_count>0)      //STR_count=1
            save_str(Des_File);
        gen_state15(Des_File,&word);  //gen asm code
        STR_count++;
        Loopc=0;            //end state OF STR
     }
     else{
        Error_code=Out_of_renge_contact;
        Next_state=Error_state;
     }

    break;    //end of case 15 (sub case of STR ins)

    case 16: //subcase of case 10 (STR NOT)
     get_token(&word);
     switch(word.token_type){

     case Token_space:
     //Next_state=Next_state
     break;

     case Token_number:
       Next_state=18;
     break;
```

```
 default:
  Error_code=Contact_required;
  Next_state=Error_state;
 }
break; //end of case 16


case 18:
if (contact_renge_check(&word)){
   if(STR_count>0)
         save_str(Des_File);
   gen_state18(Des_File,&word);       //gen asm code
   STR_count++;
   Loopc=0;          //end of success STR
}
 else{
    Error_code=Out_of_renge_contact;
    Next_state=Error_state;
 }
break;   //end of case 18 (sub case of STR ins)



/**************AND INSTRUCTION*************************/

case 20: {
 get_token(&word);
   switch(word.token_type){
   case Token_space:
         //Next_state=Next_state
   break;

   case Token_not:
         Next_state=26;
   break;
```

```
    case Token_number:
            Next_state=25;
    break;
    case Token_STR:
            switch(STR_count) {
             case  2:
               Next_state=22;
             //if sucess  STR_count must be decresed (--)
             break;

             default:
               Error_code=Toofew_STR;
               Next_state=Error_state;

            }
    break;

    default:
            Error_code=Not_or_str_contact_required; // add STR later
            Next_state=Error_state;
    }

}break; //end of case 20


case 22:

     and_str(Des_File);   // call function to gen ASM code to and str
     Next_state=1;
     STR_count--;
     Loopc=0;

break;

case 25:
```

```
  if (contact_renge_check(&word)){
    gen_state25(Des_File,&word);       //gen asm code
    Loopc=0;            //end of AND
  }
  else{
    Error_code=Out_of_renge_contact;
    Next_state=Error_state;
  }
 break;

 case 26: //subcase of case 20 (AND)
  get_token(&word);
  switch(word.token_type){

  case Token_space:
  //Next_state=Next_state     //stay at state 26
  break;

  case Token_number:
   Next_state=28;
  break;

  default:
   Error_code=Contact_required;
   Next_state=Error_state;
  }
 break; //end of case 26

 case 28:
 if (contact_renge_check(&word)){
    gen_state28(Des_File,&word);       //gen asm code
    Loopc=0;            //end of AND
  }
  else{
    Error_code=Out_of_renge_contact;
```

```
            Next_state=Error_state;
          }
        break;   //end of case 26 (sub case of AND ins)

/***************END of AND INSTRUCTION*************************/


/***************OR INSTRUCTION*************************/

        case 30: {
          get_token(&word);
            switch(word.token_type){

            case Token_space:
                  //Next_state=Next_state
            break;

            case Token_not:
                  Next_state=36;
            break;

            case Token_number:
                  Next_state=35;
            break;

            case Token_STR:
                  switch(STR_count) {
                    case  2:
                      Next_state=32;
                    //if sucess  STR_count must be decresed (--)
                    break;

                    default:
                      Error_code=Toofew_STR;
                      Next_state=Error_state;
```

```
            }
    break;


     default:
            Error_code=Not_or_contact_required;
            Next_state=Error_state;
    }

}break; //end of case 30

case 32:

     or_str(Des_File); // call function to gen ASM code to or str
     Next_state=1;
     STR_count--;
     Loopc=0;

break;

case 35:
   if (contact_renge_check(&word)){
     gen_state35(Des_File,&word);       //gen asm code
     Loopc=0;            //end of OR
   }
   else{
     Error_code=Out_of_renge_contact;
     Next_state=Error_state;
   }
 break;

 case 36: //subcase of case 30 ( NOT OR)
   get_token(&word);
   switch(word.token_type){
```

```
            case Token_space:
            //Next_state=Next_state     //stay at state 36
            break;

            case Token_number:
              Next_state=38;
            break;

            default:
             Error_code=Contact_required;
             Next_state=Error_state;
            }
           break; //end of case 36

           case 38:
           if (contact_renge_check(&word)){
              gen_state38(Des_File,&word);       //gen asm code
              Loopc=0;            //end of OR
            }
            else{
              Error_code=Out_of_renge_contact;
              Next_state=Error_state;
            }
           break;    //end of case 38 (sub case of OR ins)

/***************END  OR INSTRUCTION*************************/

/*****************OUT INSTRUCTION*************************/

           case 40: {
             get_token(&word);
              switch(word.token_type){

              case Token_space:
```

```
            //Next_state=Next_state
        break;

        case Token_not:
                Next_state=50;
        break;

        case Token_L:
                Next_state=60;
        break;

        case Token_U:
                Next_state=70;
        break;

        case Token_number:
                Next_state=80;
        break;

        default:
                Error_code=Out_param_required;
                Next_state=Error_state;
      }

    }break; //end of case 40

    case 50: //subcase of case 40 (OUT NOT)
      get_token(&word);
      switch(word.token_type){

      case Token_space:
      //Next_state=Next_state     //stay at state 50
      break;

      case Token_number:
```

```
    Next_state=55;
  break;

  default:
   Error_code=Out_of_renge_output;
   Next_state=Error_state;
  }
 break; //end of case 50

 case 55:
 if (output_renge_check(&word)){
     gen_state55(Des_File,&word);       //gen asm code
     STR_count--;
     if(STR_count>0)
           restore_STR(Des_File);
     Loopc=0;            //end of OUT NOT
  }
  else{
     Error_code=Out_of_renge_contact;
     Next_state=Error_state;
  }
 break;    //end of case 55 (sub case of OUT ins)

 case 60:
   printf("Not yet definite operand\n");
   Loopc=0;
 break;

 case 70:
   printf("Not yet definite operand\n");
   Loopc=0;
 break;

 case 80:
```

```
            if (output_renge_check(&word)){
              gen_state80(Des_File,&word);       //gen asm code
              STR_count--;
              if(STR_count>0)
                    restore_STR(Des_File);
              Loopc=0;            //end of OUT
            }
            else{
              Error_code=Out_of_renge_output ;
              Next_state=Error_state;
            }
          break;


/*********************END OF OUT INSTRUCTION*******************/


/*********************TIM INSTRUCTION*************************/
          case 90:
            printf("Not yet definite instruction \n");
            Loopc=0;
            Next_state=1;
          break;

    /*    case 90: {
            get_token(&word);
            switch(word.token_type){

              case Token_space:
                    //Next_state=Next_state
              break;

              case Token_number:
                     Next_state=95;
              break;
```

```
        default:
                Error_code=Contact_required;
                Next_state=Error_state;
        }

    }break; //end of case 90 (Timer case)

    case 95:
      if(output_renge_check(&word)){
        get_token(&word);
        switch(word.token_type){

        case Token_space:
              //stay at this state
        break;

        case Token_number:
                Next_state=98;
        break;

        default:
                Error_code=Reg_required;
                Next_state=Error_state;

        }

      }

      else{
        Error_code=Out_of_renge_output;
        Next_state=Error_state;
      }
    break; //end of case 95
```

```
case 98:
  if(treg_renge_check(&word)){
    get_token(&word);
    switch(word.token_type){

          case Token_space:
          //stay at this state
          break;

          case Token_number:
            Next_state=99;
          break;

          default:
            Error_code=Out_of_renge_register;
            Next_state=Error_state;

    }

  }
  else{
        Error_code=Out_of_renge_register;
        Next_state=Error_state;
  }
break; //end of case 98

case 99:
  if(num_check(&word)){
    gen_state99(Des_File,&word);
    STR_count--;
    if(STR_count>0)
          restore_STR(Des_File);
    Loopc=0;

  }
```

```
            else {
              Error_code=Out_of_renge_number;
              Next_state=Error_state;
            }

          break;*/ //end of case 99

/**********Not yet definite COUNTER INSTRUCTION**********/
          case 100:
             printf("Not yet definite instruction \n");
             Loopc=0;
             Next_state=1;
          break;              //end of case 100 (Counter)

/**********END OF COUNTER **********/

  case Error_state: { // Error state
          Error_status=TRUE;

          Loopc=0;

    switch(Error_code){

    case Toomany_STR:
     printf("Line %d Too many STR instruction\n\a",word.line_no);
     fprintf(Err_File,"Line %d Too many STR instruction\n",word.line_no);
    break;

    case Toofew_STR:
     printf("Line %d Too few STR instruction\n\a",word.line_no);
     fprintf(Err_File,"Line %d Too few STR instruction\n",word.line_no);
```

```
break;

case Reg_required:
 printf("Line %d Register NO. is required \n\a",word.line_no);
 fprintf(Err_File,"Line %d Register NO. is required\n",word.line_no);
break;

case STR_required:
 printf("Line %d STR instruction is required \a\n",word.line_no);
 fprintf(Err_File,"Line %d STR instruction is required\n",word.line_no);
break;

case Not_or_contact_required:
 printf("Line %d [NOT] contact number is required \a\n",word.line_no);
 fprintf(Err_File,"Line %d [NOT] contact number is required \n",word.line_no);
break;

case Contact_required:
 printf("Line %d contact number is required \a\n",word.line_no);
 fprintf(Err_File,"Line %d contact number is required \n",word.line_no);
break;

case Out_param_required:
 printf("Line %d [NOT] contact number is required \a\n",word.line_no);
 fprintf(Err_File,"Line %d NOT contact number is required \n",word.line_no);
break;
case Instruct_error:
 printf("Line %d Instruction error\n",word.line_no);
 fprintf(Err_File,"Line %d Instruction error\n",word.line_no);
break;

case Out_of_renge_contact:
 printf("Line %d Out of renge contact\n",word.line_no);
 fprintf(Err_File,"Line %d Out of renge contact\n",word.line_no);
break;
```

```
case Not_or_str_contact_required:
 printf("Line %d [NOT] contact number or [STR] is required \a\n",word.line_no);
 fprintf(Err_File,"Line %d [NOT] contact NO. or [STR] is required \a\n",word.line_no);
break;

case Out_of_renge_output:
 printf("Line %d Out of renge output\n ",word.line_no);
 fprintf(Err_File,"Line %d Out of renge output ",word.line_no);
break;

case Out_of_renge_number:
 printf("Line %d Out of renge number\n",word.line_no);
 fprintf(Err_File,"Line %d Out of renge number\n ",word.line_no);
break;

case Out_of_renge_register:
 printf("Line %d Out of renge register\n",word.line_no);
 fprintf(Err_File,"Line %d Out of renge register\n ",word.line_no);
break;

default :
 printf("Other Error \a\n");
 fprintf(Err_File,"Other Error \a\n");

} //end of switch Error_code
    }break; //end case of Error state


    }//end of switch(Next_state)

} // End of Loopc

} //  End of EOF , END instruction
if(End_status==0){
```

```
    Error_status=1;
    printf("Error !!! No END instruction \n\a");
    fprintf(Err_File,"Error !!! No END instruction \n");
 }
gen_end(Des_File);

fclose(Source_File);

if (fclose(Err_File)!=0 && Error_status==1)
   printf("Writing  error reference file ERROR \n\a");
if (fclose(Des_File)!=0)
 { printf("Writing output file error !!! \a\n");
   Error_status=1;
 }

  if (Error_status)
   remove(des_file);     // delete asm file
  else{

   remove(err_file);     // remove(); delete error file
   execl("sxa51","sxa51",des_file,NULL);
  }
}


int contact_renge_check(words *word)
{ int i;
        if( (*word).number<1   || (*word).number>24 )
          i=0;

        else{
          i=1;
          (*word).contact_num=(*word).number-1;
        }
```

```
  return(i);
}
int output_renge_check(words *word)
{  int i;

        if (  (*word).number<9 || (*word).number>24  )
          i=0;

        else{
          i=1;
          (*word).contact_num=(*word).number-1;
        }

    return(i);
}

/*int treg_renge_check(words *word)
{ int i;
    if ( ((*word).number<201) || ((*word).number>208) )
        i=0;
    else {
        i=1;
        (*word).reg_displace=(*word).number-201; //register store time
    }
    return(i);
} */

/*int creg_renge_check(words *word)
{ int i;
    if ( ((*word).number<300) || ((*word).number>308) )
        i=0;
    else
```

```
        i=1;
    return(i);
} */

void gen_state15(FILE *Des_File,words *word)
{
  fprintf(Des_File,"\t\t\t MOV C,%d\n\n",(*word).contact_num  );
}

void gen_state18(FILE *Des_File,words *word)
{
  fprintf(Des_File,"\t\t\t MOV C,%d\n",(*word).contact_num  );
  fprintf(Des_File,"\t\t\t CPL C\n\n"  );
}

void gen_state25(FILE *Des_File,words *word)
{
  fprintf(Des_File,"\t\t\t\ MOV 65H,C\n");
  fprintf(Des_File,"\t\t\t\ MOV C,%d\n",(*word).contact_num  );
  fprintf(Des_File,"\t\t\t\ ANL C,65H\n\n",(*word).contact_num  );
}

void gen_state28(FILE *Des_File,words *word)
{
  fprintf(Des_File,"\t\t\t\ MOV 65H,C\n");
  fprintf(Des_File,"\t\t\t\ MOV C,%d\n",(*word).contact_num  );
  fprintf(Des_File,"\t\t\t\  CPL C\n"  );
  fprintf(Des_File,"\t\t\t\ ANL C,65H\n\n",(*word).contact_num  );
}

void gen_state35(FILE *Des_File,words *word)
{
  fprintf(Des_File,"\t\t\t\ MOV 65H,C\n");
  fprintf(Des_File,"\t\t\t\ MOV C,%d\n",(*word).contact_num  );
  fprintf(Des_File,"\t\t\t\ ORL C,65H\n\n",(*word).contact_num  );
```

```
}
void gen_state38(FILE *Des_File,words *word)
{
 fprintf(Des_File,"\t\t\t\ MOV 65H,C\n");
 fprintf(Des_File,"\t\t\t\ MOV C,%d\n",(*word).contact_num  );
 fprintf(Des_File,"\t\t\t\  CPL C\n"  );
 fprintf(Des_File,"\t\t\t\ ORL C,65H\n\n",(*word).contact_num  );

}

void gen_state55(FILE *Des_File,words *word)
{
 fprintf(Des_File,"\t\t\t\ CPL C\n"  );
 fprintf(Des_File,"\t\t\t\ MOV %d,C\n\n",(*word).contact_num  );
}

void gen_state80(FILE *Des_File,words *word)
{
 fprintf(Des_File,"\t\t\t\ MOV %d,C\n\n",(*word).contact_num  );
}

void gen_state99(FILE *Des_File,words *word)
{
 fprintf(Des_File,"\t\t\t\  SETB %xH \n",0x18+(*word).reg_displace); //set gate
 fprintf(Des_File,"\t\t\t\  MOV  %xH,C \n",0x40+(*word).reg_displace); //store status
 fprintf(Des_File,"\t\t\t\  MOV %xH,#%x\n",0x88+(*word).reg_displace,(*word).timer_num);
//time sec
 fprintf(Des_File,"\t\t\t\  MOV
%xH,#%d\n\n",0x80+(*word).reg_displace,(*word).contact_num);
}

void and_str(FILE *Des_File)
{
 fprintf(Des_File,"\t\t\t\ ANL C,64H\n\n " );
```

```
}
void or_str(FILE *Des_File)
{
  fprintf(Des_File,"\t\t\t ORL C,64H\n\n " );
}

void save_str(FILE* Des_File)
{
 fprintf(Des_File,"\t\t\t MOV  64H,C\n\n " );
}

void restore_STR(FILE* Des_File)
{
 fprintf(Des_File,"\t\t\t MOV  C,64H\n\n " );
}

void gen_head(FILE* Des_File)
{
   fprintf(Des_File,"\t\t\t  ORG 9000H\n\n");
   fprintf(Des_File,"\t\t\t  MOV 20h,#0\n");
   fprintf(Des_File,"\t\t\t  MOV 21h,#0\n");
   fprintf(Des_File,"\t\t\t  MOV 22h,#0\n");
   fprintf(Des_File,"\t;**********BODY**********\n\n");
   fprintf(Des_File,"Loops:\n");
   fprintf(Des_File,"\t\t\t mov  dptr,#0\n");
   fprintf(Des_File,"\t\t\t movx A,@dptr\n");
   fprintf(Des_File,"\t\t\t mov  20h,A\n\n");
}
void gen_end(FILE* Des_File)
{  fprintf(Des_File,"\t\t\t mov  dptr,#1000h\n");
   fprintf(Des_File,"\t\t\t mov  A,22h\n");
   fprintf(Des_File,"\t\t\t movx @dptr,A\n");
   fprintf(Des_File,"\t\t\t jmp  Loops\n");
   fprintf(Des_File,"\t\t\t END " );
}
```

ข้อเสนอแนะในการพัฒนา

ข้อเสนอแนะในการพัฒนา

เนื่องจาก ชุดจำลองพีแอลซี ที่ได้จัดสร้างขึ้นนี้มีเพียงฟังก์ชั่นการใช้งานทางด้านลอจิกเท่านั้น ในการที่จะเพิ่มฟังก์ชั่นหน่วยตั้งเวลา , หน่วยนับจำนวน และการแก้ไขข้อมูลบางประการของโปรแกรมโดยผ่านแมททริกซ์คีย์ของวงจร จะต้องกระทำในสิ่งต่อไปนี้

1. เพิ่มสเตทแมชชีนในโปรแกรมตัวแปรภาษา ให้สามารถรับคำสั่งที่เพิ่มเข้าไปได้ รายละเอียดของการเพิ่มสเตทแมชชีนนี้ขึ้นอยู่กับการวางหลักไวยากรณ์ของภาษาบูลลีนที่เพิ่มเข้าไปว่าคำสั่งต้องการไวยากรณ์ใดบ้าง เช่น TIM 23 200 5 เป็นคำสั่ง ให้ตัวนับเวลานับเวลา 5 วินาที โดยใช้รีจิสเตอร์หมายเลข 200 เป็นตัวบันทึกข้อมูล และเมื่อนับได้ครบจะ ON ขดลวดหมายเลข 23 เป็นต้น (ในโปรแกรม token.c ใน ภาคผนวก ค จะมี stateที่ยังไม่ระบุการทำงานอยู่สามารถเพิ่มเติมในส่วนนั้นได้)
2. สร้างโปรแกรมการตอบสนองการอินเตอร์รัพท์ของไทม์เมอร์ 0 ใน AT89C52 เพราะการนับเวลาในหน่วยนับเวลาที่จำลองขึ้นมีจำนวนมากกว่าหนึ่ง จึงต้องใช้โปรแกรม มาจำลองหน่วยนับเวลาทั้งหมดที่จัดสร้างขึ้นและไม่ควรมีจำนวนมากเกินไป และในโปรแกรมการตอบสนองการอินเตอร์รัพท์ ดังกล่าวจะต้องมีส่วนการจัดการกับหน่วยนับจำนวนของชุดจำลองพีแอลซีนี้ด้วย
3. สร้างโปรแกรมตอบสนองการกดคีย์แบบแมททริกซ์คีย์ เพิ่มเติมใส่ลงในส่วนของมอนิเตอร์โปรแกรม ซึ่งอาศัยหลักการสแกนแมททริกซ์คีย์ทั่วไป ซึ่งมีประโยชน์ช่วยให้ไม่ต้องแก้ไขชุดคำสั่งภาษาบูลลีนทุกครั้งที่มีการแก้ไขข้อมูลเล็ก ๆ น้อย ๆ (คำนึงไว้เสมอว่า ขณะอยู่ในโหมดการรันไม่สามารถกดแมททริกซ์คีย์ได้เนื่องจาก พอร์ต 0 ของ AT89C52 ถูกใช้เป็นดาต้าบัสสำหรับติดต่อกับโปรแกรมเมโมรีภายนอก)

ข้อมูลของอุปกรณ์ที่ใช้ และข้อมูลทางไฟฟ้าของ IC TTL

ตารางคำสั่ง HD44780

■ Instructions

Instruction	Code										Description	Execution Time (max) (when fcp or fosc is 250 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter.	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	*	Sets DD RAM address 0 in address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40μs
Display On/Off Control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of entire display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor and shifts display without changing DD RAM contents.	40μs
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (L) and character font (F).	40μs
Set CG RAM Address	0	0	0	1	ACG						Sets CG RAM address. CG RAM data is sent and received after this setting.	40μs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. DD RAM data is sent and received after this setting.	40μs
Read Busy Flag & Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0μs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40μs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40μs
	I/D=1: Increment I/D=0: Decrement S =1: Accompanies display shift S/C=1: Display shift S/C=0: Cursor move R/L=1: Shift to the right R/L=0: Shift to the left DL=1: 8 bits, DL=0: 4 bits N =1: 2 lines, N=0: 1 line F =1: 5×10 dots, F=0: 5×7 dots BF=1: Internally operating BF=0: Can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM ACG: CG RAM address ADD: DD RAM Address: Corresponds to cursor address AC: Address counter used for both DD and CG RAM address.	Execution time changes when frequency changes Example: When fcp or fosc is 270 kHz: $40\mu s \times \frac{250}{270} = 37\mu s$

* No effect

รายละเอียดของคำสั่ง HD44780

1. CLEAR DISPLAY

Clear display

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	0	0	1

คำสั่งนี้จะเป็นการเขียนช่องว่างหรือ SPACE (ASCII 20H) เข้าไปใน DD RAM ทั้งหมดและทำการ SET DD RAM ADDRESSER เป็นศูนย์ ตัว CURSOR จะกลับไปอยู่ ตำแหน่งบนสุดซ้ายมือของจอภาพ SET I/D = 1, S ไม่มีการเปลี่ยน

2. RETURN HOME

Return home

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	0	0	*

* No effect

คำสั่งนี้จะทำการ SET DD RAM ADDRESSER เป็นศูนย์ ตัว CURSOR จะกลับไปอยู่ตำแหน่งบนสุดซ้ายมือของจอภาพข้อมูลในจอภาพไม่เปลี่ยน

3. ENTRY MODE SET

Entry mode set

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	0	I/D	S

BIT I/D : โดยจะเป็นตัวกำหนดให้ว่าเมื่อเขียนหรืออ่านข้อมูลแล้วจะทำให้ DD RAM ADDRESS เพิ่มขึ้นหนึ่งหรือลดลงหนึ่งโดย

1 = เพิ่ม

0 = ลดลงหนึ่ง

BIT S : เป็นกำหนดแสดงผลโดยถ้า S = 1 จะเป็นการใส่ข้อมูลแล้วตัว CURSOR อยู่กับที่ข้อมูลจะถูกดันไปทางซ้าย ถ้า S = 0 ข้อมูลจะ อยู่กับที่ตัว CURSOR จะถูกดันไปทางขวามือ

4. DISPLAY ON/OFF CONTROL

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	1	D	C	B

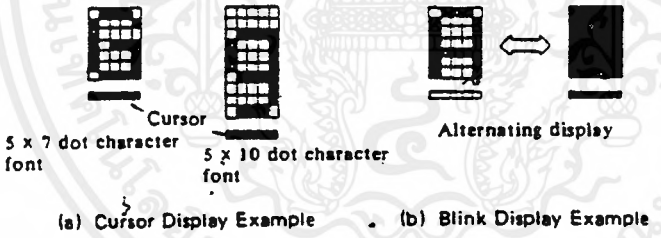
BIT D : เป็น BIT ให้เปิดปิดหน้าจอภาพโดยถ้า

D = 1 จะ ON และ

D = 0 จะ OFF

BIT C : จะให้แสดง CURSOR ให้ BIT C = 1 และถ้าไม่ต้องการแสดง CURSOR BIT C = 0 โดยตัว CURSOR จะอยู่ที่ LINE ที่ 8 ในแบบ 5X7 DOT และจะอยู่ LINE ที่ 11 ในแบบ 5X10 DOT

BIT B : เป็น BIT SET การกระพริบของ CURSOR โดย B = 1 การกระพริบ B = 0 ไม่มีการกระพริบ โดยมีระยะเวลาการกระพริบประมาณ 379.2 ms



(a) Cursor Display Example (b) Blink Display Example

5. CURSOR OR DISPLAY SHIFT

Cursor or display shift

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	1	S/C	R/L	*	*

* No effect

เป็นคำสั่งกำหนดให้ตำแหน่ง CURSOR หรือข้อมูลไปเกิดทางซ้ายหรือขวาโดยไม่ต้องใช้คำสั่งเขียนหรืออ่าน โดย

S/C	R/L	
0	0	ทำการย้าย CURSOR ไปจากตำแหน่งเดิมไปซ้ายมือ 1 ตำแหน่ง
0	1	ทำการย้าย CURSOR ไปจากตำแหน่งเดิมไปขวามือ 1 ตำแหน่ง
1	0	เป็นการดันตัวอักษรที่เกิดไปทางซ้าย
1	1	เป็นการดันตัวอักษรที่เกิดไปทางขวามือ

6. FUNCTION SET

Function set

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	1	DL	N	F	•	•

* No effect

BIT DL : เป็นการ SET การติดต่อว่าจะให้เป็นแบบ 8 BIT หรือ 4 BIT

โดยถ้าต้องการติดต่อ 4 BIT DL = 0 และ 8 BIT DL = 1

N : เป็นการ SET บรรทัดการแสดงผล N = 0 แสดง 1 บรรทัด

N = 1 แสดง 2 บรรทัด ในกรณีมากกว่า 2 บรรทัด ก็ให้ SET N = 1

F : เป็นการ SET ชนิด DOT การแสดงผล 5X7 หรือ 5X10 โดย

F = 0 เป็นแบบ 5X7 และ F = 1 เป็นแบบ 5X10

N F	No. of display lines	Character font	Duty factor	Remarks
0 0	1	5 x 7 dots	1/8	
0 1	1	5 x 10 dots	1/11	
1 •	2	5 x 7 dots	1/16	Cannot display 2 lines with 5 x 10 dot character font.

* No effect

7. SET CG RAM ADDRESS

Set CG RAM address

	RS	R/W	DB_7						DB_0
Code	0	0	0	1	A	A	A	A	A	A

← Higher Order Bits — Lower Order Bits →

ใน HD44780 นั้นจะมีหน่วยความจำอยู่ 2 ชุด คือ DISPLAY DATA RAM (DD RAM) จำนวน 80X8 BIT และ CHARACTER GENERATOR ROM CG RAM จำนวน 512 BIT และ 7200 BIT คำสั่งนี้จะเป็นการ SET ADDRESS ใน CG RAM โดยต้องทำการ SET ADDRESS ก่อนเขียนหรืออ่านข้อมูลจาก CG RAM ด้วย

8. SET DD RAM ADDRESS

Set DD RAM address

	RS	R/W	DB_7						DB_0
Code	0	0	1	A	A	A	A	A	A	A

← Higher Order Bits — lower Order Bits →

เป็นคำสั่ง SET ค่า ADDRESS ใน DD RAM ในการเขียนหรืออ่านค่าจาก DD RAM(DD RAM คือ ส่วนที่จะแสดงผลหน้าจอ LCD) โดยจำนวน ADDRESS ที่จะเกิดขึ้นบนจอ LCD จะอยู่กับ SET ค่า N ด้วย

N = 0 (1 บรรทัด) ADDRESS จะอยู่ 00H-4FH

N = 1 (2 บรรทัด) ADDRESS จะอยู่ 00H-27H สำหรับบรรทัดที่ 1 และ 40H-67H สำหรับ บรรทัดที่ 2

แบบการจัด ADDRESS ของ DD RAM หน้าจอ LCD แบบ 16 ตัวอักษร 1 บรรทัด ,16 ตัวอักษร 2 บรรทัด ,16 ตัวอักษร 4 บรรทัด , 20 ตัวอักษร 1 บรรทัด, 20 ตัวอักษร 2 บรรทัด และ 40 ตัวอักษร 2 บรรทัด

16 ตัวอักษร 1 บรรทัด

00	01	02	03	04	05	06	07	40	41	42	43	44	45	46	47

16 ตัวอักษร 2 บรรทัด

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

16 ตัวอักษร 4 บรรทัด

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F

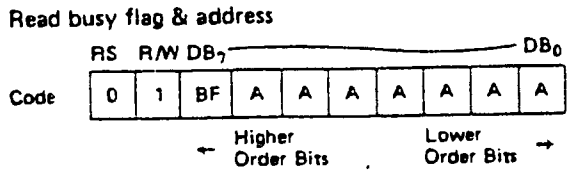
20 ตัวอักษร 1 บรรทัด

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13

20 ตัวอักษร 2 บรรทัด

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53

9. READ BUSY FLAG AND ADDRESS



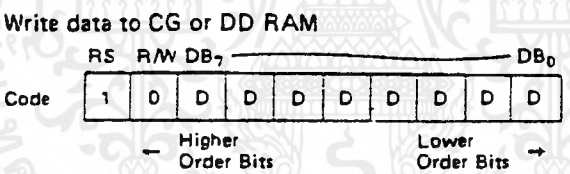
เป็นคำสั่งอ่านค่า BUSY FLAG ซึ่งจะเป็นตัวบอกว่าตัว HD44780 นี้อยู่ในขบวนการทำงานภายในอยู่หรืออยู่ในสภาพพร้อมจะรับข้อมูล โดย

BF = 1 อยู่ในขบวนการทำงานภายในไม่พร้อมจะรับข้อมูลหรือคำสั่ง

BF = 0 พร้อมจะรับข้อมูลหรือคำสั่งได้

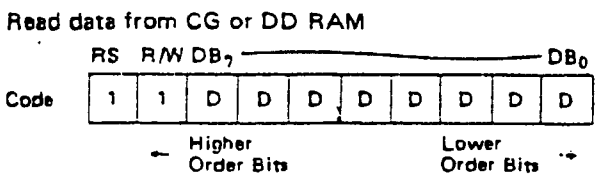
และนอกจากนี้ยังเป็นคำสั่งอ่านค่าข้อมูล ADDRESS ของ CG RAM หรือ DD RAM ด้วย

10. WRITE DATA TO CG หรือ DD RAM



เป็นคำสั่งเขียนข้อมูลเข้าไปใน CG หรือ DD RAM โดยเมื่อเขียนข้อมูลแล้ว ADDRESS จะเพิ่มหรือลดโดยอัตโนมัติตามคำสั่งที่ SET ใน ENTRY MODE ข้อกำหนดที่จะรู้ว่าเป็นการเขียนข้อมูลของ CG RAM หรือ DD RAM ทำได้โดยการ SET ADDRESS ของ CG RAM หรือ DD RAM ขึ้นมาก่อนจะเขียนข้มูล

11. READ DATA FROM CG OR DD RAM



ขาต่างๆในการต่อใช้งาน HD44780

1. RS (REGISTOR SELECTION) จะเป็นขาเลือก REGISTOR ภายในซึ่งมี อยู่ 2 ตัวคือ **INSTRUCTION REGISTOR (IR)** และ **DATA REGISTOR (DR)** โดยถ้าเป็น 1 จะเป็นการเลือก DATA และถ้าเป็น 0 จะเป็นการเลือก INSTRUCTION

2. R/W (READ/WRITE) เป็นตัวเลือกว่าจะเขียนหรือจะอ่านข้อมูลจากตัว IC โดยอ่า นข้อมูล = 1, เขียนข้อมูล = 0

3. E (ENA BLE SIGNAL) เป็นขากำหนดสภาพการรับเขียนอ่านข้อมูล

Register selection

RS	R/W	E	Operation
0	0		IR write as internal operation (Display clear, etc.)
0	1		Read busy flag (DB_7) and address counter (DB_0 ~ DB_6)
1	0		DR write as internal operation (DR to DD or CG RAM)
1	1		DR read as internal operation (DD or CG RAM to DR)

4. DB0-DB7 เป็นขารับส่งข้อมูลจากตัว IC

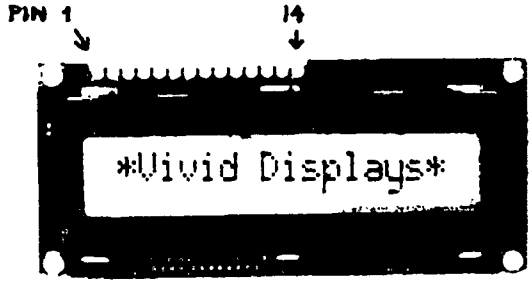
5. VDD ไฟเลี้ยงตัววงจร +5V

6. VSS เป็นขา GND

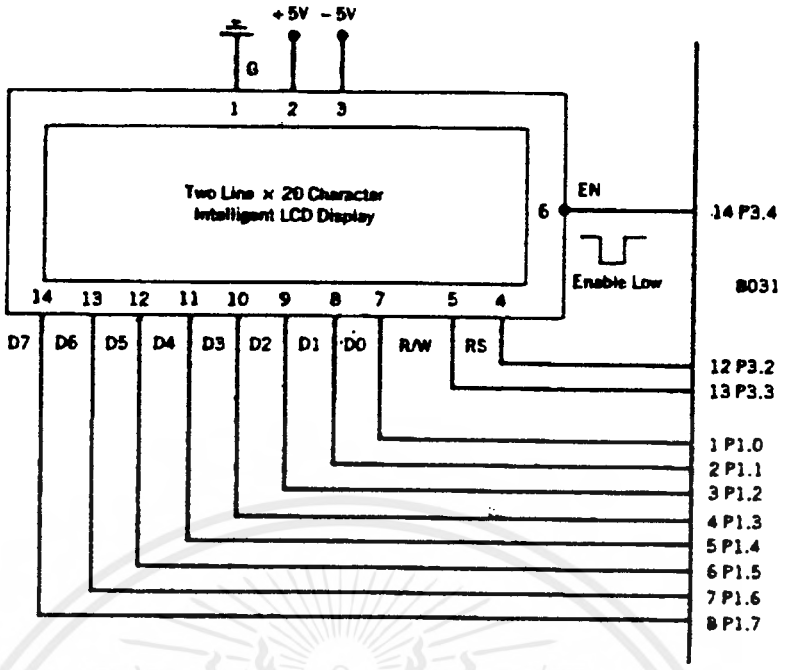
7. VO เป็นขารับ VOLTAGE ในการขับ LCD ให้สว่างหรือมืด

PIN CONNECTION

Pin No	Symbol	Level	Function
1	V_{SS}	—	0V
2	V_{DD}	—	+5V
3	V_O	—	—
4	RS	H/L	L: Instruction code input H: Data input
5	R/W	H/L	H: Data read (LCD module → MPU) L: Data write (LCD module ← MPU)
6	E	H, H→L	Enable signal
7	DB0	H/L	Data bus line Note (1), (2)
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	



Intelligent LCD Circuit for "Lcdisp" Program




```
ADDRESS    MNEMONIC        COMMENT
            org 0000h
lcdisp:     clr p3.2        ;select the command register
            clr p3.3        ;select write level
            mov a,#3fh      ;command 8 bits/char.  2 rows, 5 x 10
            acall strobe    ;strobe command to display
            mov a,#0eh      ;command screen and cursor on, no blink
            acall strobe
            mov a,#06h      ;command cursor right as data displayed
            acall strobe
            mov a,#01h      ;clear all and home cursor
            acall strobe
            setb p3.2       ;select display data RAM register
            mov a,#'h'      ;say "hello"
            acall strobe
            mov a,#'e'
            acall strobe
            mov a,#'l'
            acall strobe
            acall strobe
            mov a,#'o'
            acall strobe
here:       sjmp here       ;message sent
;
;the subroutine "strobe" is used to check for a display busy
;condition, and pulse P3.3 high-low-high to enable the display
;write or read
;
strobe:     mov p1,#0ffh    ;configure port 1 as an input
            setb p3.3       ;set read level
wait:       setb p3.4       ;generate read strobe
            clr p3.4        ;enable the display
            jb p1.7,wait    ;check for busy when BF = 1
            setb p3.4       ;end of read strobe
            clr p3.3        ;write character to display
            setb p3.2       ;choose data RAM
            mov p1,a        ;character to port 1
            clr p3.4        ;generate write strobe
            setb p3.4
            clr p3.2        ;return with display as before call
            ret
             end
```

Features

- Compatible with 80C51 and 80C52 Products
- 8 Kbytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
  - Data Retention: 10 Years
- Fully Static Operation: 0 Hz to 24 MHz
- Three-Level Program Memory Lock
- 256 x 8-Bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-Bit Timer/Counters
- Eight Interrupt Sources
- Programmable Serial Channel
- Low Power Idle and Power Down Modes

Description

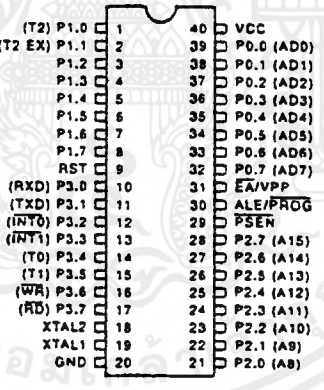
The AT89C52 is a low-power, high-performance CMOS 8-bit microcomputer with 8 Kbytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard 80C51 and 80C52 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C52 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

The AT89C52 provides the following standard features: 8 Kbytes of Flash, 256 bytes of RAM, 32 I/O lines, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89C52 is

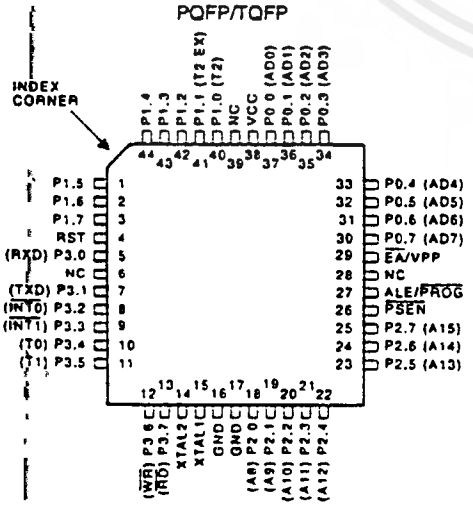
continued

Pin Configurations

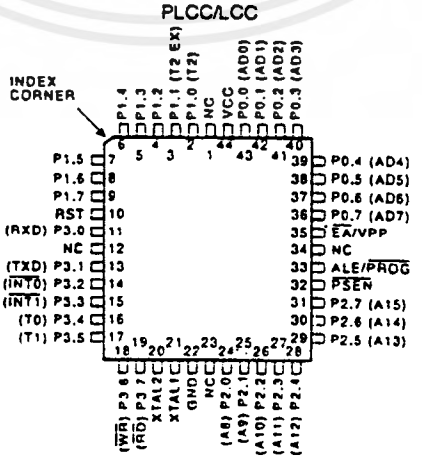
PDIP/Cerdip



PQFP/TQFP



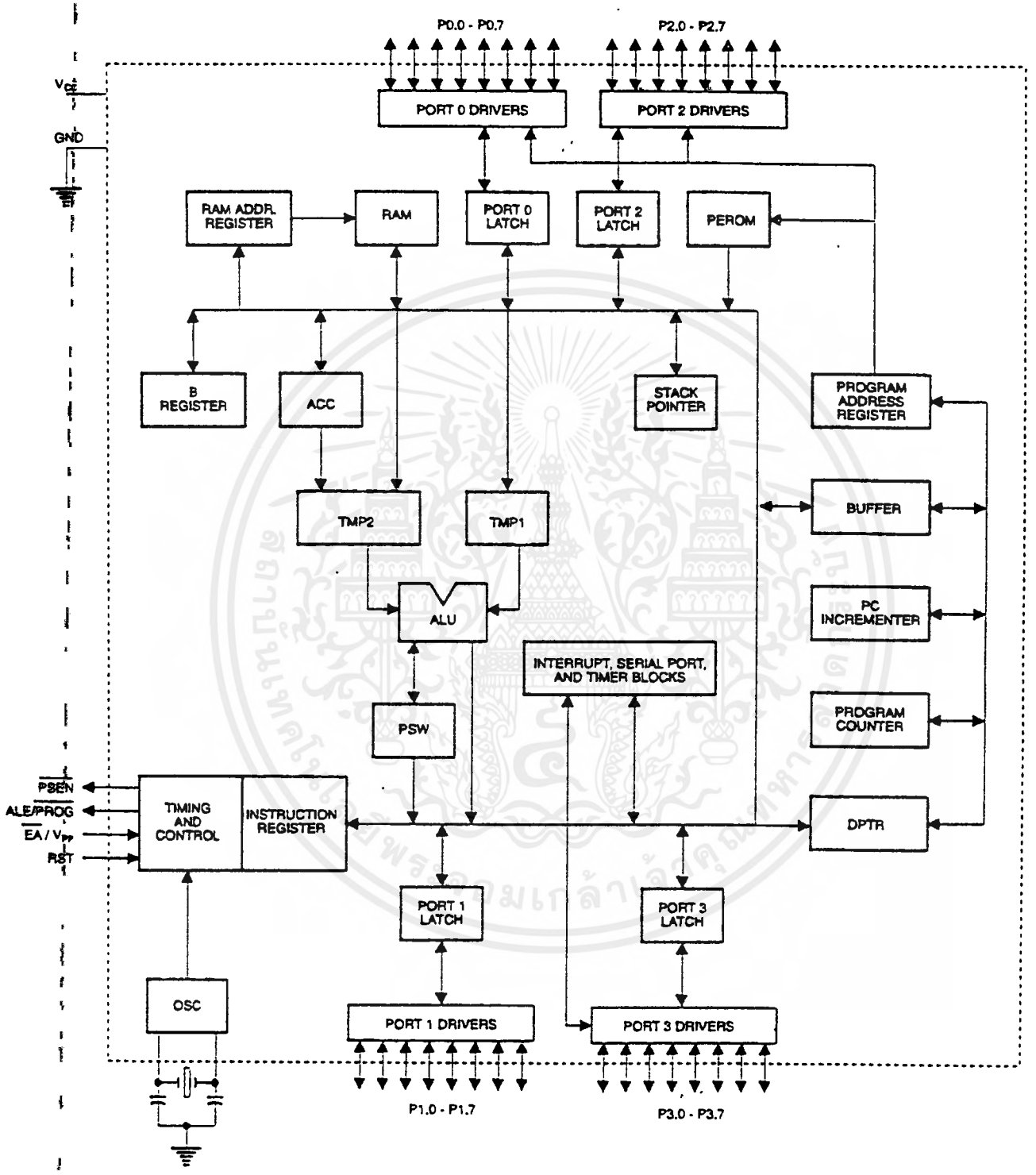
PLCC/LCC



8-Bit Microcontroller with 8 Kbytes Flash

AT89C52

Block Diagram



Description (Continued)

designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next hardware reset.

Pin Description

V_{CC}

Supply voltage.

GND

Ground.

Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

In addition, P1.0 and P1.1 can be configured to be the timer/counter 2 external count input (P1.0/T2) and the timer/counter 2 trigger input (P1.1/T2EX), respectively, as shown in the following table.

Port Pin	Alternate Functions
P1.0	T2 (external count input to Timer/Counter 2), clock-out
P1.1	T2EX (Timer/Counter 2 capture/reload trigger and direction control)

Port 1 also receives the low-order address bytes during Flash programming and program verification.

Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51, as shown in the following table.

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and programming verification.

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$

Address Latch Enable is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external data memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcrontroller is in external execution mode.

$\overline{PSEN}$

Program Store Enable is the read strobe to external program memory.

When the AT89C52 is executing code from external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external data memory.

$\overline{EA}$/V_{PP}

External Access Enable. $\overline{EA}$ must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to V_{CC} for internal program executions.

This pin also receives the 12-volt programming enable voltage (V_{PP}) during Flash programming when 12-volt programming is selected.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Special Function Registers

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in Table 1.

Note that not all of the addresses are occupied, and unoccupied addresses may not be implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect.

User software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

Timer 2 Registers Control and status bits are contained in registers T2CON (shown in Table 2) and T2MOD (shown in Table 4) for Timer 2. The register pair (RCAP2H, RCAP2L) are the Capture/Reload registers for Timer 2 in 16-bit capture mode or 16-bit auto-reload mode. *(continued)*

Table 1. AT89C52 SFR Map and Reset Values

0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000								0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFH
0C0H									0C7H
0B8H	IP X0000000								0BFH
0B0H	P3 11111111								0B7H
0A8H	IE 00000000								0AFH
0A0H	P2 11111111								0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000			8FH
80H	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 00000000	87H

Table 2. T2CON—Timer/Counter 2 Control Register

T2CON Address = 0C8H — Reset Value = 0000 0000B

Bit Addressable

Bit	7	6	5	4	3	2	1	0
	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

Symbol	Function
TF2	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.
EXF2	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software. EXF2 does not cause an interrupt in up/down counter mode (DCEN = 1).
RCLK	Receive clock enable. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in serial port Modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
TCLK	Transmit clock enable. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in serial port Modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
EXEN2	Timer 2 external enable. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.
TR2	Start/Stop control for Timer 2. TR2 = 1 starts the timer.
C/$\overline{T2}$	Timer or counter select for Timer 2. C/$\overline{T2}$ = 0 for timer function. C/$\overline{T2}$ = 1 for external event counter (falling edge triggered).
CP/$\overline{RL2}$	Capture/Reload select. CP/$\overline{RL2}$ = 1 causes captures to occur on negative transitions at T2EX if EXEN2 = 1. CP/$\overline{RL2}$ = 0 causes automatic reloads to occur when Timer 2 overflows or negative transitions occur at T2EX when EXEN2 = 1. When either RCLK or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

Special Function Registers (Continued)

Interrupt Registers The individual interrupt enable bits are in the IE register. Two priorities can be set for each of the six interrupt sources in the IP register.

Data Memory

The AT89C52 implements 256 bytes of on-chip RAM. The upper 128 bytes occupy a parallel address space to the Special Function Registers. That means the upper 128 bytes have the same addresses as the SFR space but are physically separate from SFR space.

When an instruction accesses an internal location above address 7FH, the address mode used in the instruction specifies whether the CPU accesses the upper 128 bytes of RAM or the SFR space. Instructions that use direct addressing access SFR space.

For example, the following direct addressing instruction accesses the SFR at location 0A0H (which is P2).

```
MOV 0A0H, #data
```

Instructions that use indirect addressing access the upper 128 bytes of RAM. For example, the following indirect addressing instruction, where R0 contains 0A0H, accesses the data byte at address 0A0H, rather than P2 (whose address is 0A0H).

```
MOV @R0, #data
```

Note that stack operations are examples of indirect addressing, so the upper 128 bytes of data RAM are available as stack space.

Timer 0 and 1

Timer 0 and Timer 1 in the AT89C52 operate the same way as Timer 0 and Timer 1 in the AT89C51.

Timer 2

Timer 2 is a 16-bit Timer/Counter that can operate as either a timer or an event counter. The type of operation is selected by bit $C/\overline{T2}$ in the SFR T2CON (shown in Table 2). Timer 2 has three operating modes: capture, auto-reload (up or down counting), and baud rate generator. The modes are selected by bits in T2CON, as shown in Table 3.

Timer 2 consists of two 8-bit registers, TH2 and TL2. In the Timer function, the TL2 register is incremented every machine cycle. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the Counter function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since two machine cycles (24 oscillator periods) are required to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. To ensure that a given level is sampled at least once before it changes, the level should be held for at least one full machine cycle.

Table 3. Timer 2 Operating Modes

RCLK + TCLK	$CP/\overline{RL2}$	TR2	MODE
0	0	1	16-Bit Auto-Reload
0	1	1	16-Bit Capture
1	X	1	Baud Rate Generator
X	X	0	(Off)

Capture Mode

In the capture mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 is a 16-bit timer or counter which upon overflow sets bit TF2 in T2CON. This bit can then be used to generate an interrupt. If EXEN2 = 1, Timer 2 performs the same operation, but a 1-to-0 transition at external input T2EX also causes the current value in TH2 and TL2 to be captured into RCAP2H and RCAP2L, respectively. In addition, the transition at T2EX causes bit EXF2 in T2CON to be set. The EXF2 bit, like TF2, can generate an interrupt. The capture mode is illustrated in Figure 1.

Auto-Reload (Up or Down Counter)

Timer 2 can be programmed to count up or down when configured in its 16-bit auto-reload mode. This feature is invoked by the DCEN (Down Counter Enable) bit located in the SFR T2MOD (see Table 4). Upon reset, the DCEN bit is set to 0 so that timer 2 will default to count up. When DCEN is set, Timer 2 can count up or down, depending on the value of the T2EX pin.

Figure 2 shows Timer 2 automatically counting up when DCEN = 0. In this mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 counts up to 0FFFFH and then sets the TF2 bit upon overflow. The overflow also causes the timer registers to be reloaded with the 16-bit value in RCAP2H and RCAP2L. The values in RCAP2H and RCAP2L are preset by software. If EXEN2 = 1, a 16-bit reload can be triggered either by an overflow or by a 1-to-0 transition at external input T2EX. This transition also sets the EXF2 bit. Both the TF2 and EXF2 bits can generate an interrupt if enabled.

Setting the DCEN bit enables Timer 2 to count up or down, as shown in Figure 3. In this mode, the T2EX pin controls the direction of the count. A logic 1 at T2EX makes Timer 2 count up. The timer will overflow at 0FFFFH and set the TF2 bit. This overflow also causes the 16-bit value in RCAP2H and RCAP2L to be reloaded into the timer registers, TH2 and TL2, respectively.

(continued)

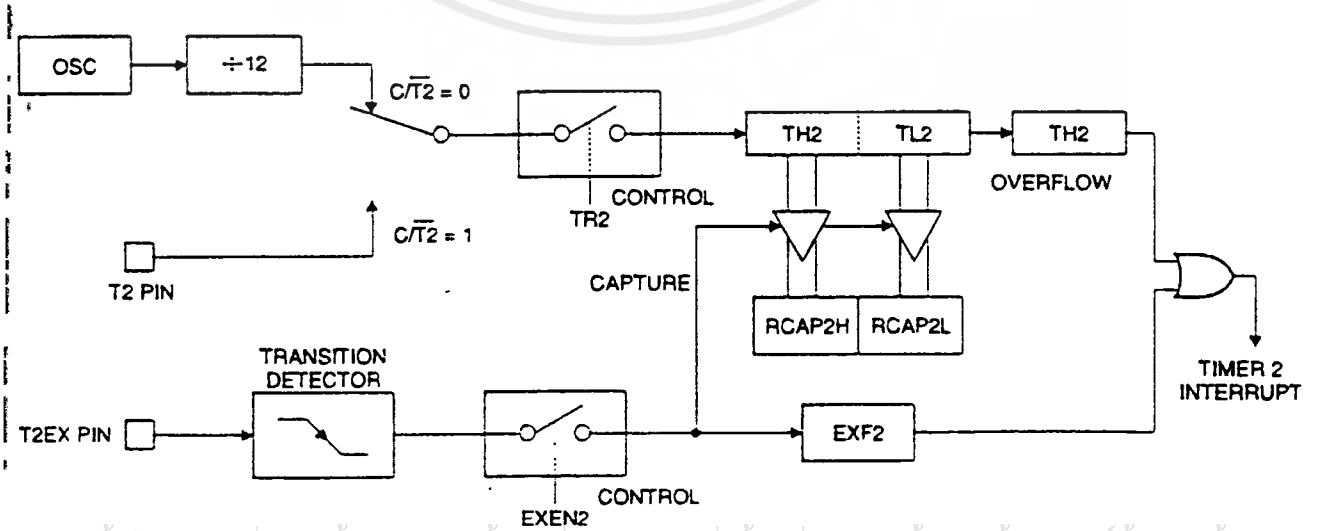


Figure 1. Timer 2 in Capture Mode

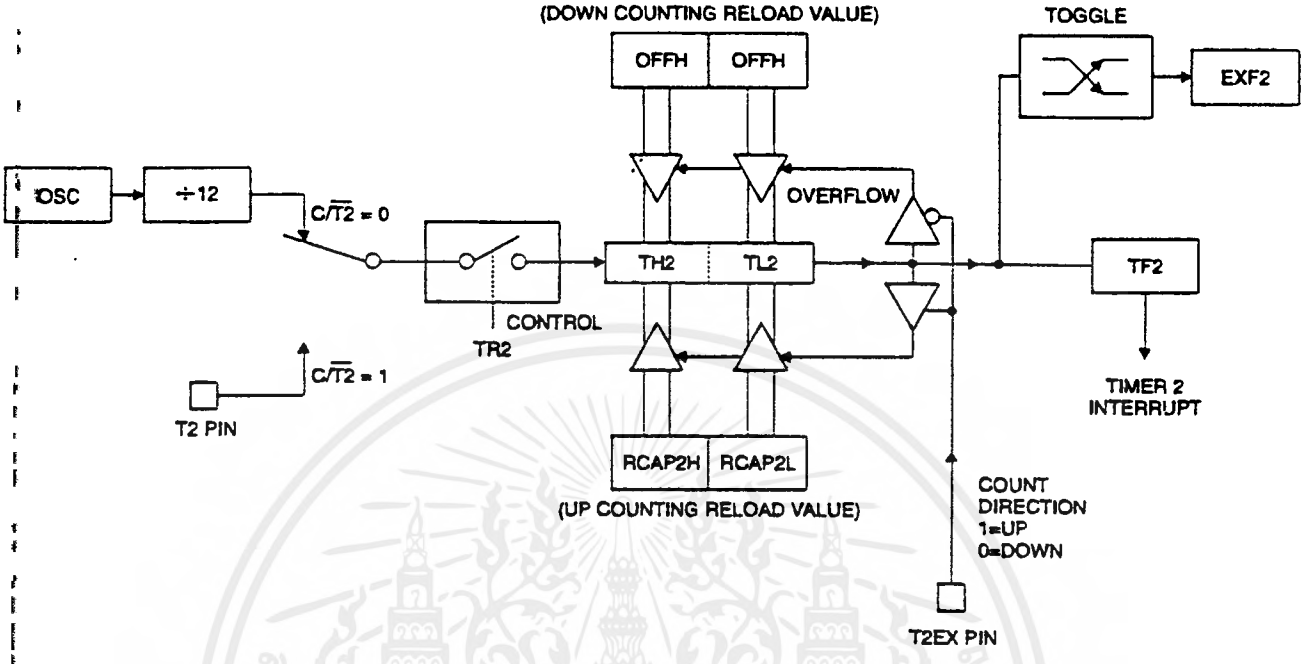


Figure 3. Timer 2 Auto Reload Mode (DCEN = 1)

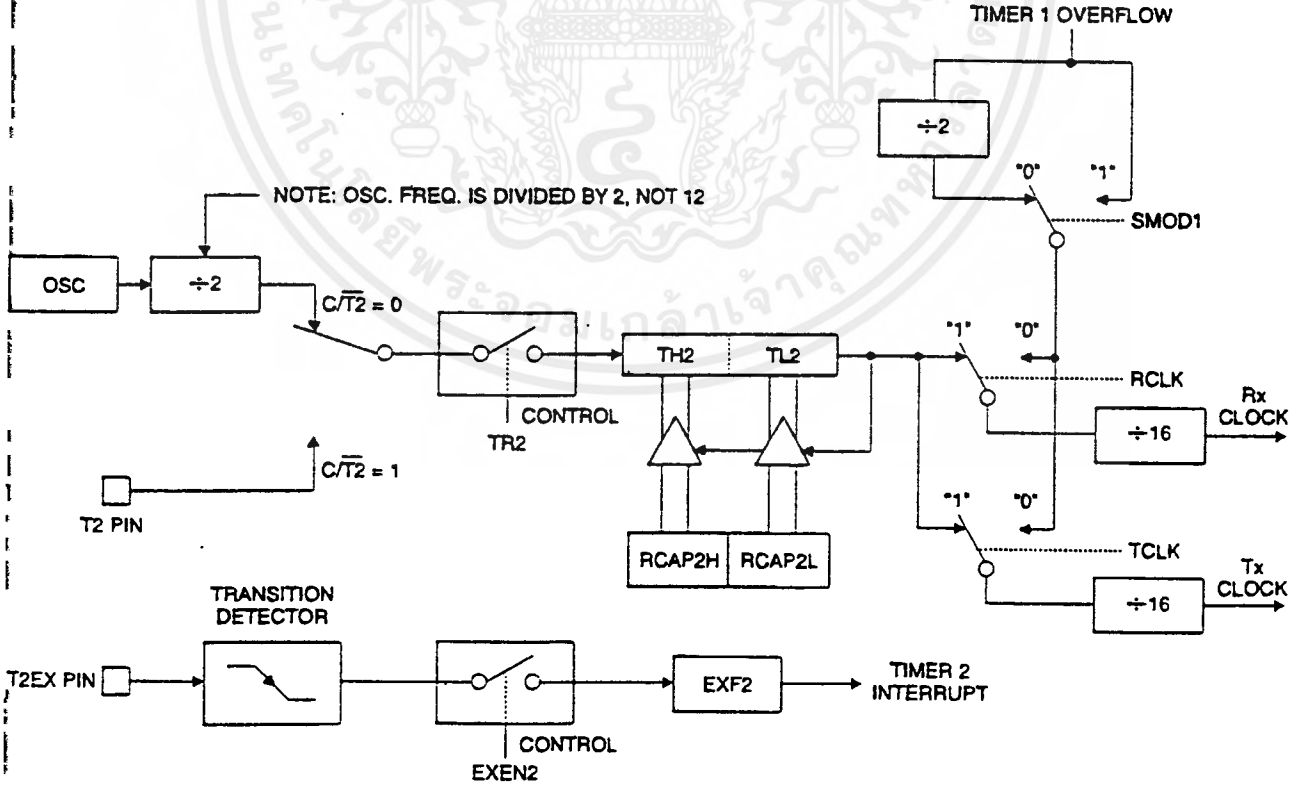


Figure 4. Timer 2 in Baud Rate Generator Mode

Baud Rate Generator

Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Table 2). Note that the baud rates for transmit and receive can be different if Timer 2 is used for the receiver or transmitter and Timer 1 is used for the other function. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 4.

The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

The baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate according to the following equation.

$$\text{Modes 1 and 3 Baud Rates} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

The Timer can be configured for either timer or counter operation. In most applications, it is configured for timer operation ($C/\overline{T2} = 0$). The timer operation is different for Timer 2 when it is used as a baud rate generator. Normally, as a timer, it increments every machine cycle (at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (at 1/2 the oscillator frequency). The baud rate formula is given below.

$$\frac{\text{Modes 1 and 3}}{\text{Baud Rate}} = \frac{\text{Oscillator Frequency}}{32 \times [65536 - (\text{RCAP2H, RCAP2L})]}$$

where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

Timer 2 as a baud rate generator is shown in Figure 4. This figure is valid only if RCLK or TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2 and will not generate an interrupt. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt.

Note that when Timer 2 is running (TR2 = 1) as a timer in the baud rate generator mode, TH2 or TL2 should not be read from or written to. Under these conditions, the Timer is incremented every state time, and the results of a read or write may not be accurate. The RCAP2 registers may be read but should not be written to, because a write might overlap a reload and cause write and/or reload errors. The timer should be turned off (clear TR2) before accessing the Timer 2 or RCAP2 registers.

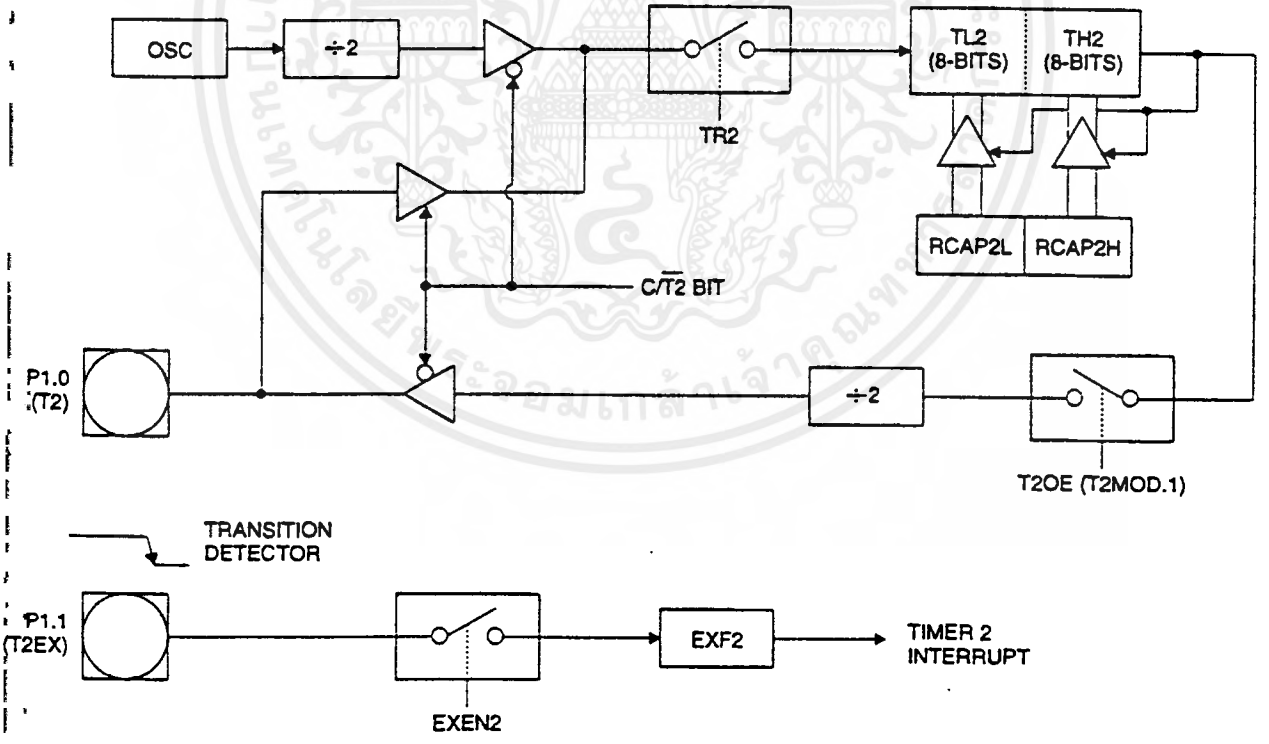


Figure 5. Timer 2 in Clock-Out Mode

Programmable Clock Out

A 50% duty cycle clock can be programmed to come out on P1.0, as shown in Figure 5. This pin, besides being a regular I/O pin, has two alternate functions. It can be programmed to input the external clock for Timer/Counter 2 or to output a 50% duty cycle clock ranging from 61 Hz to 4 MHz at a 16 MHz operating frequency.

To configure the Timer/Counter 2 as a clock generator, bit C/$\overline{T2}$ (T2CON.1) must be cleared and bit T2OE (T2MOD.1) must be set. Bit TR2 (T2CON.2) starts and stops the timer.

The clock-out frequency depends on the oscillator frequency and the reload value of Timer 2 capture registers (RCAP2H, TCAP2L), as shown in the following equation.

$$\text{Clock-Out Frequency} = \frac{\text{Oscillator Frequency}}{4 \times [65536 - (\text{RCAP2H, RCAP2L})]}$$

In the clock-out mode, Timer 2 roll-overs will not generate an interrupt. This behavior is similar to when Timer 2 is used as a baud-rate generator. It is possible to use Timer 2 as a baud-rate generator and a clock generator simultaneously. Note, however, that the baud-rate and clock-out frequencies cannot be determined independently from one another since they both use RCAP2H and RCAP2L.

UART

The UART in the AT89C52 operates the same way as the UART in the AT89C51.

Interrupts

The AT89C52 has a total of six interrupt vectors: two external interrupts ($\overline{INT0}$ and $\overline{INT1}$), three timer interrupts (Timers 0, 1, and 2), and the serial port interrupt. These interrupts are all shown in Figure 6.

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. IE also contains a global disable bit, EA, which disables all interrupts at once.

Note that Table 5 shows that bit position IE.6 is unimplemented. In the AT89C51, bit position IE.5 is also unimplemented. User software should not write 1s to these bit positions, since they may be used in future AT89 products.

Timer 2 interrupt is generated by the logical OR of bits TF2 and EXF2 in register T2CON. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and that bit will have to be cleared in software.

The Timer 0 and Timer 1 flags, TF0 and TF1, are set at S5P2 of the cycle in which the timers overflow. The values are then polled by the circuitry in the next cycle. However, the Timer 2 flag, TF2, is set at S2P2 and is polled in the same cycle in which the timer overflows.

Table 5. Interrupt Enable (IE) Register

(MSB)							(LSB)
EA	—	ET2	ES	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt.

Enable Bit = 0 disables the interrupt.

Symbol	Position	Function
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	Reserved.
ET2	IE.5	Timer 2 interrupt enable bit.
ES	IE.4	Serial Port interrupt enable bit.
ET1	IE.3	Timer 1 interrupt enable bit.
EX1	IE.2	External interrupt 1 enable bit.
ET0	IE.1	Timer 0 interrupt enable bit.
EX0	IE.0	External interrupt 0 enable bit.
User software should never write 1s to unimplemented bits, because they may be used in future AT89 products.		

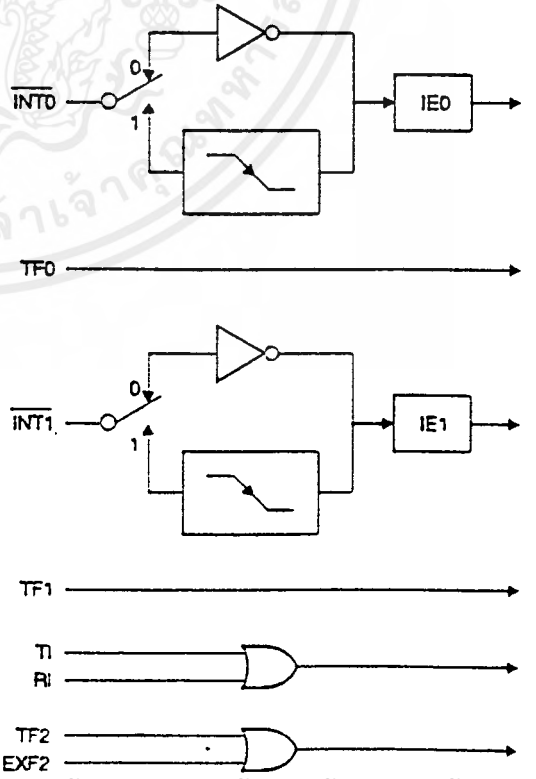
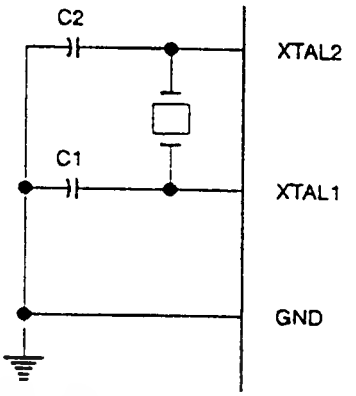


Figure 6. Interrupt Sources

Oscillator Characteristics

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier that can be configured for use as an on-chip oscillator, as shown in Figure 7. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven, as shown in Figure 8. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Figure 7. Oscillator Connections



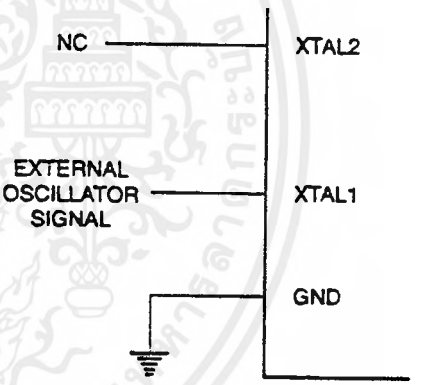
Notes: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

Note that when idle mode is terminated by a hardware reset, the device normally resumes program execution from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when idle mode is terminated by a reset, the instruction following the one that invokes idle mode should not write to a port pin or to external memory.

Figure 8. External Clock Drive Configuration



Power Down Mode

In the power down mode, the oscillator is stopped, and the instruction that invokes power down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power down mode is terminated. The only exit from power down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Status of External Pins During Idle and Power Down

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power Down	Internal	0	0	Data	Data	Data	Data
Power Down	External	0	0	Float	Data	Data	Data

Program Memory Lock Bits

The AT89C52 has three lock bits that can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the following table.

When lock bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value and holds that value until reset is activated. The latched value of $\overline{EA}$ must agree with the current logic level at that pin in order for the device to function properly.

Lock Bit Protection Modes

Program Lock Bits				
	LB1	LB2	LB3	Protection Type
1	U	U	U	No program lock features.
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, EA is sampled and latched on reset, and further programming of the Flash memory is disabled.
3	P	P	U	Same as mode 2, but verify is also disabled.
4	P	P	P	Same as mode 3, but external execution is also disabled.

Programming the Flash

The AT89C52 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents = FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volt) or a low-voltage (V_{CC}) program enable signal. The low voltage programming mode provides a convenient way to program the AT89C52 inside the user's system, while the high-voltage programming mode is compatible with conventional third party Flash or EPROM programmers.

The AT89C52 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

	V_{PP} = 12 V	V_{PP} = 5 V
Top-Side Mark	AT89C52 xxxx yyww	AT89C52 xxxx-5 yyww
Signature	(030H)=1EH (031H)=52H (032H)=FFH	(030H)=1EH (031H)=52H (032H)=05H

The AT89C52 code memory array is programmed byte-by-byte in either programming mode. *To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.*

Programming Algorithm: Before programming the AT89C52, the address, data and control signals should be set up according to the Flash programming mode table and Figures 9 and 10. To program the AT89C52, take the following steps.

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise $\overline{EA}$/V_{PP} to 12 V for the high-voltage programming mode.
5. Pulse ALE/$\overline{PROG}$ once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 1.5 ms. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

$\overline{Data}$ Polling: The AT89C52 features $\overline{Data}$ Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written data on PO.7. Once the write cycle has been completed, true data is valid on all outputs, and the next cycle may begin. $\overline{Data}$ Polling may begin any time after a write cycle has been initiated.

Ready/$\overline{Busy}$: The progress of byte programming can also be monitored by the RDY/$\overline{BSY}$ output signal. P3.4 is pulled low after ALE goes high during programming to indicate $\overline{BUSY}$. P3.4 is pulled high again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE/$\overline{PROG}$ low for 10 ms. The code array is written with all 1s. The chip erase operation must be executed before the code memory can be reprogrammed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 030H,

031H; and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(030H) = 1EH indicates manufactured by Atmel
(031H) = 52H indicates 89C52
(032H) = FFH indicates 12 V programming
(032H) = 05H indicates 5 V programming

Programming Interface

Every code byte in the Flash array can be written, and the entire array can be erased, by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Flash Programming Modes

Mode	RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	EA/V_{PP}	P2.6	P2.7	P3.6	P3.7
Write Code Data	H	L	⎍	H/12V(1)	L	H	H	H
Read Code Data	H	L	H	H	L	L	H	H
Write Lock Bit - 1	H	L	⎍	H/12V	H	H	H	H
Bit - 2	H	L	⎍	H/12V	H	H	L	L
Bit - 3	H	L	⎍	H/12V	H	L	H	L
Chip Erase	H	L	⎍(2)	H/12V	H	L	L	L
Read Signature Byte	H	L	H	H	L	L	L	L

Notes: 1. The signature byte at location 032H designates whether V_{PP} = 12 V or V_{PP} = 5 V should be used to enable programming.
2. Chip Erase requires a 10 ms $\overline{PROG}$ pulse.

Figure 9. Programming the Flash Memory

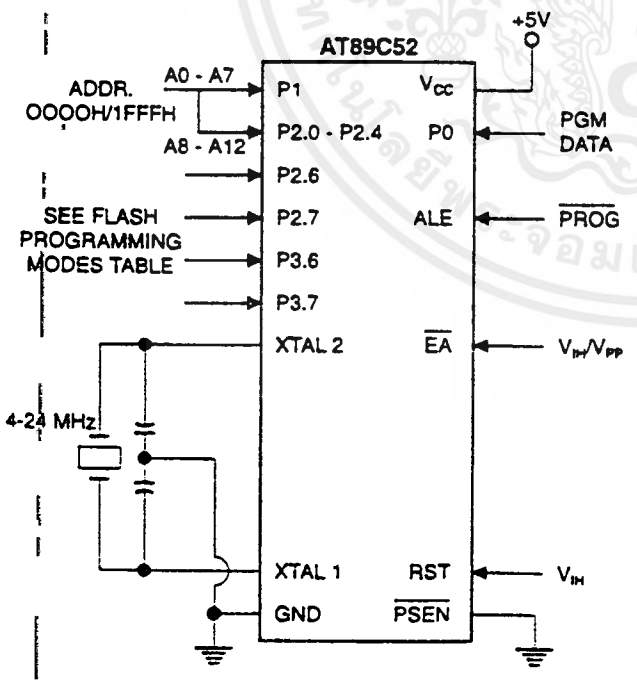
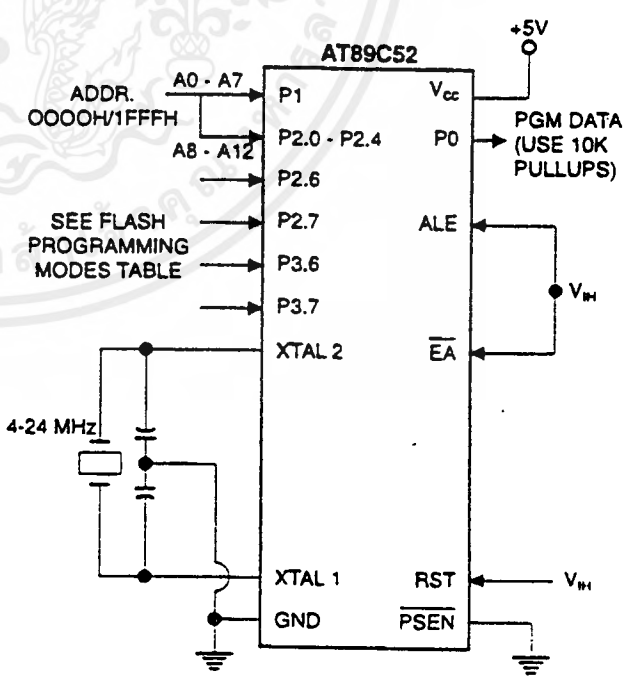


Figure 10. Verifying the Flash Memory



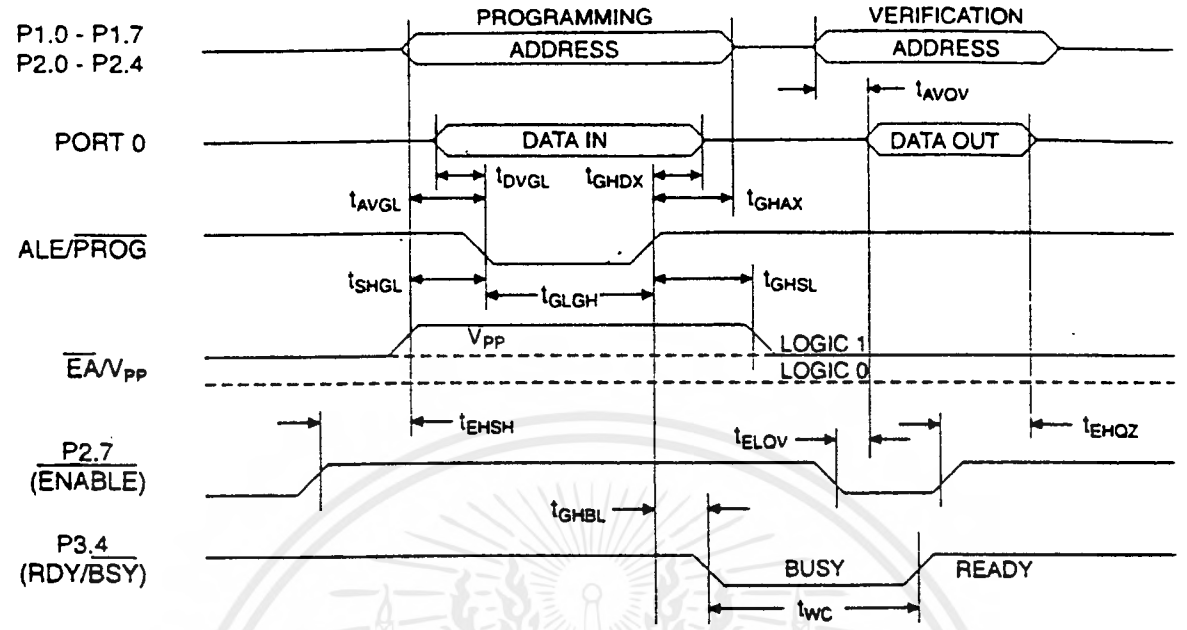
Flash Programming and Verification Characteristics

T_A = 21°C to 27°C, V_{CC} = 5.0 ± 10%

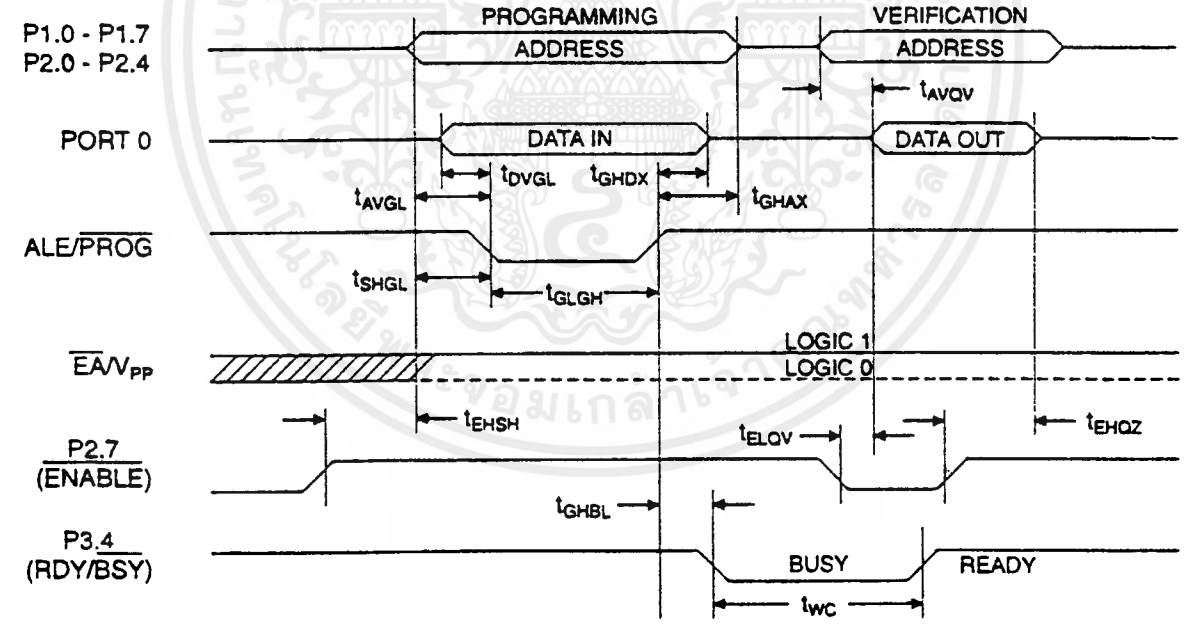
Symbol	Parameter	Min	Max	Units
V_{PP}[1]	Programming Enable Voltage	11.5	12.5	V
I_{PP}[1]	Programming Enable Current		1.0	mA
$1/t_{CLCL}$	Oscillator Frequency	4	24	MHz
t_{AVGL}	Address Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHAX}	Address Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHDX}	Data Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{EHSH}	P2.7 ($\overline{ENABLE}$) High to V_{PP}	$48t_{CLCL}$		
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}[1]	V_{PP} Hold After $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{AVQV}	Address to Data Valid		$48t_{CLCL}$	
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		$48t_{CLCL}$	
t_{EHQV}	Data Float After $\overline{ENABLE}$	0	$48t_{CLCL}$	
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		1.0	µs
t_{WC}	Byte Write Cycle Time		2.0	ms

Note: 1. Only used in 12-volt programming mode.

Flash Programming and Verification Waveforms - High Voltage Mode



Flash Programming and Verification Waveforms - Low Voltage Mode



Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0 V to +7.0 V
Maximum Operating Voltage	6.6 V
DC Output Current	15.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. Characteristics

The values shown in this table are valid for T_A = -40°C to 85°C and V_{CC} = 5.0 V ± 20%, unless otherwise noted.

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low Voltage	(Except $\overline{EA}$)	-0.5	0.2 V_{CC}-0.1	V
V_{IL1}	Input Low Voltage ($\overline{EA}$)		-0.5	0.2 V_{CC}-0.3	V
V_{IH}	Input High Voltage	(Except XTAL1, RST)	0.2 V_{CC}+0.9	V_{CC}+0.5	V
V_{IH1}	Input High Voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC}+0.5	V
V_{OL}	Output Low Voltage[1] (Ports 1,2,3)	I_{OL} = 1.6 mA		0.45	V
V_{OL1}	Output Low Voltage[1] (Port 0, ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Output High Voltage (Ports 1,2,3, ALE, $\overline{PSEN}$)	I_{OH} = -60 µA, V_{CC} = 5 V ± 10%	2.4		V
		I_{OH} = -25 µA	0.75 V_{CC}		V
		I_{OH} = -10 µA	0.9 V_{CC}		V
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)	I_{OH} = -800 µA, V_{CC} = 5 V ± 10%	2.4		V
		I_{OH} = -300 µA	0.75 V_{CC}		V
		I_{OH} = -80 µA	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1,2,3)	V_{IN} = 0.45 V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1,2,3)	V_{IN} = 2 V		-650	µA
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)	0.45 < V_{IN} < V_{CC}		±10	µA
RRST	Reset Pulldown Resistor		50	300	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz		25	mA
		Idle Mode, 12 MHz		6.5	mA
	Power Down Mode[2]	V_{CC} = 6 V		100	µA
		V_{CC} = 3 V		40	µA

Notes: 1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
Maximum I_{OL} per port pin:10 mA
Maximum I_{OL} per 8-bit port:
Port 0:26 mA
Ports 1,2, 3:15 mA
Maximum total IOL for all output pins:71 mA
If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum V_{CC} for Power Down is 2 V.

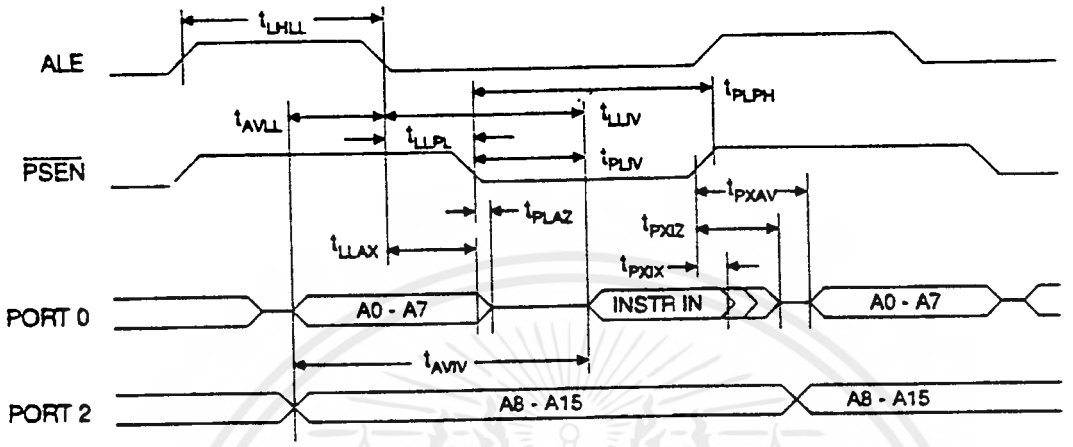
A.C. Characteristics

Under operating conditions, load capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; load capacitance for all other outputs = 80 pF.

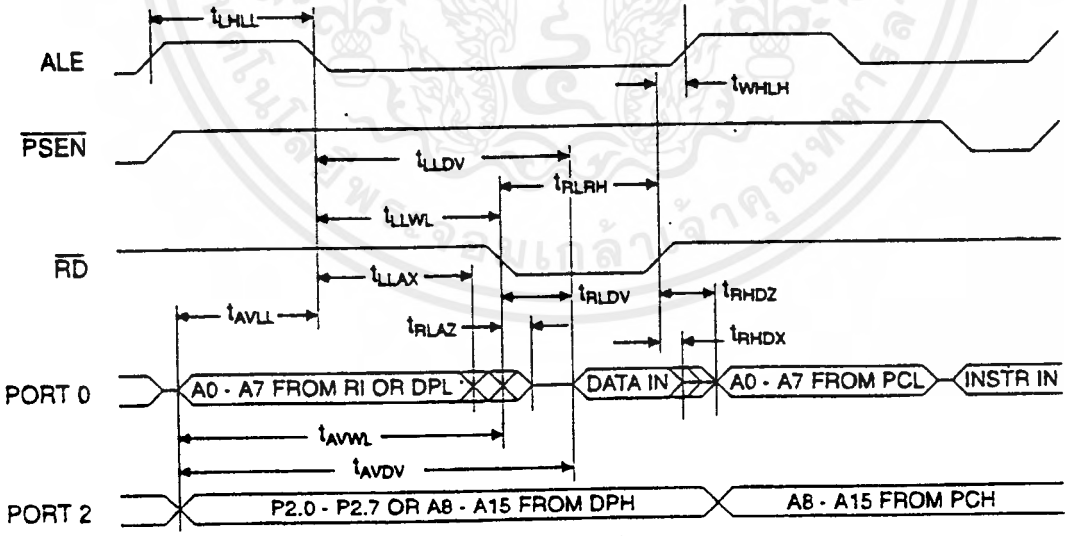
External Program and Data Memory Characteristics

Symbol	Parameter	12 MHz Oscillator		Variable Oscillator		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency			0	24	MHz
t_{LHLL}	ALE Pulse Width	127		$2t_{CLCL}-40$		ns
t_{AVLL}	Address Valid to ALE Low	28		$t_{CLCL}-13$		ns
t_{LLAX}	Address Hold After ALE Low	48		$t_{CLCL}-20$		ns
t_{LLIV}	ALE Low to Valid Instruction In		233		$4t_{CLCL}-65$	ns
t_{LLPL}	ALE Low to $\overline{PSEN}$ Low	43		$t_{CLCL}-13$		ns
t_{PLPH}	$\overline{PSEN}$ Pulse Width	205		$3t_{CLCL}-20$		ns
t_{PLIV}	$\overline{PSEN}$ Low to Valid Instruction In		145		$3t_{CLCL}-45$	ns
t_{PXIX}	Input Instruction Hold After $\overline{PSEN}$	0		0		ns
t_{PXIZ}	Input Instruction Float After $\overline{PSEN}$		59		$t_{CLCL}-10$	ns
t_{PXAV}	$\overline{PSEN}$ to Address Valid	75		$t_{CLCL}-8$		ns
t_{AVIV}	Address to Valid Instruction In		312		$5t_{CLCL}-55$	ns
t_{PLAZ}	$\overline{PSEN}$ Low to Address Float		10		10	ns
t_{RLRH}	$\overline{RD}$ Pulse Width	400		$6t_{CLCL}-100$		ns
t_{WLWH}	$\overline{WR}$ Pulse Width	400		$6t_{CLCL}-100$		ns
t_{RLDV}	$\overline{RD}$ Low to Valid Data In		252		$5t_{CLCL}-90$	ns
t_{RHDX}	Data Hold After $\overline{RD}$	0		0		ns
t_{RHDZ}	Data Float After $\overline{RD}$		97		$2t_{CLCL}-28$	ns
t_{LLDV}	ALE Low to Valid Data In		517		$8t_{CLCL}-150$	ns
t_{AVDV}	Address to Valid Data In		585		$9t_{CLCL}-165$	ns
t_{LLWL}	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	$3t_{CLCL}-50$	$3t_{CLCL}+50$	ns
t_{AVWL}	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		$4t_{CLCL}-75$		ns
t_{QVWX}	Data Valid to $\overline{WR}$ Transition	23		$t_{CLCL}-20$		ns
t_{QVWH}	Data Valid to $\overline{WR}$ High	433		$7t_{CLCL}-120$		ns
t_{WHQX}	Data Hold After $\overline{WR}$	33		$t_{CLCL}-20$		ns
t_{RLAZ}	$\overline{RD}$ Low to Address Float		0		0	ns
t_{WHLH}	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	$t_{CLCL}-20$	$t_{CLCL}+25$	ns

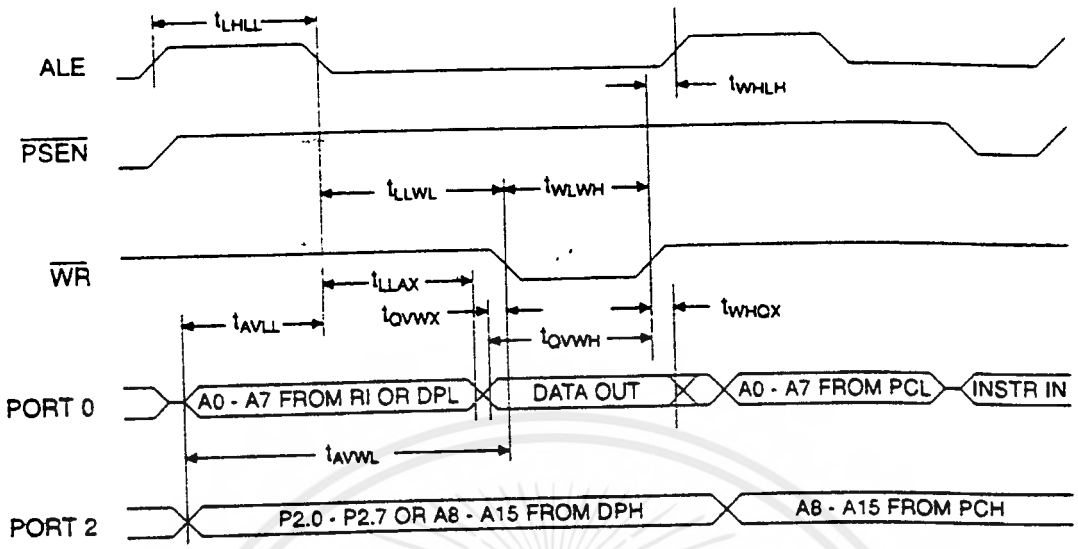
External Program Memory Read Cycle



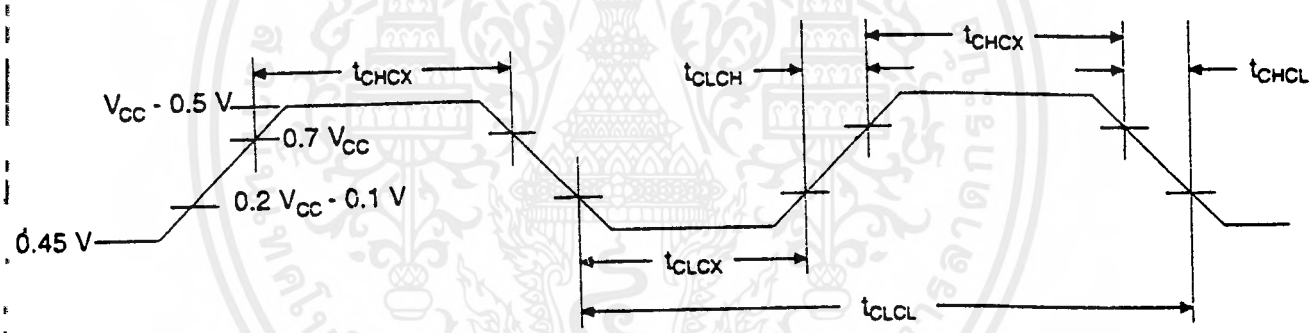
External Data Memory Read Cycle



External Data Memory Cycle



External Clock Drive Waveforms



External Clock Drive

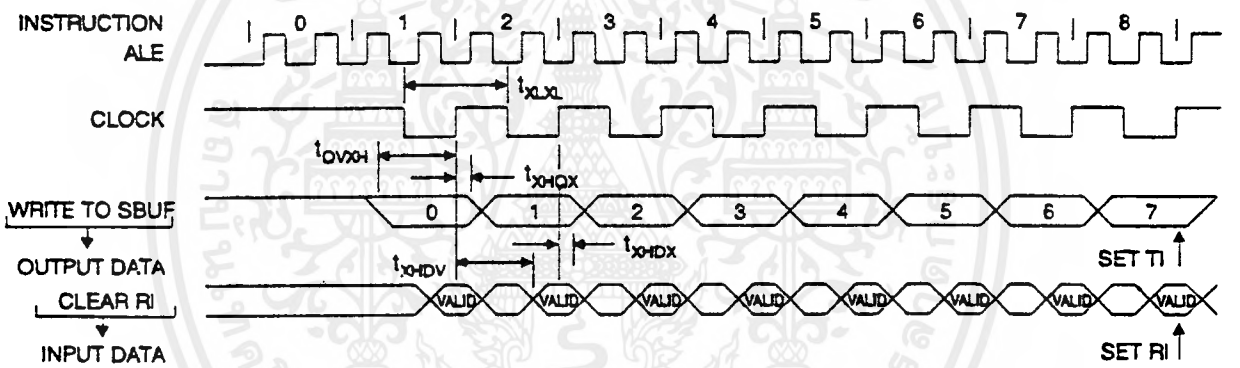
Symbol	Parameter	Min	Max	Units
1/tCLCL	Oscillator Frequency	0	24	MHz
tCLCL	Clock Period	41.6		ns
tCHCX	High Time	15		ns
tCLCX	Low Time	15		ns
tCLCH	Rise Time		20	ns
tCHCL	Fall Time		20	ns

Serial Port Timing: Shift Register Mode Test Conditions

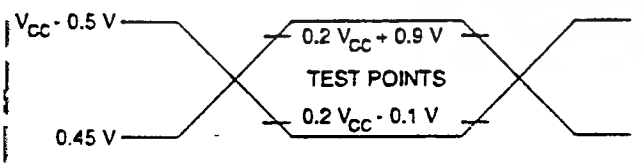
The values in this table are valid for V_{CC} = 5.0 V ± 20% and Load Capacitance = 80 pF.

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		µs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}$-133		ns
t_{XHQX}	Output Data Hold After Clock Rising Edge	50		$2t_{CLCL}$-33		ns
t_{XHDX}	Input Data Hold After Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		$10t_{CLCL}$-133	ns

Shift Register Mode Timing Waveforms



AC Testing Input/Output Waveforms [1]



Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5 V for a logic 1 and 0.45 V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Float Waveforms [1]

Note: 1. For timing purposes, a port pin is no longer floating wh a 100-mV change from load voltage occurs. A port pin begins to float when a 100-mV change from the loaded V_{OH}/V_{OL} level occurs.

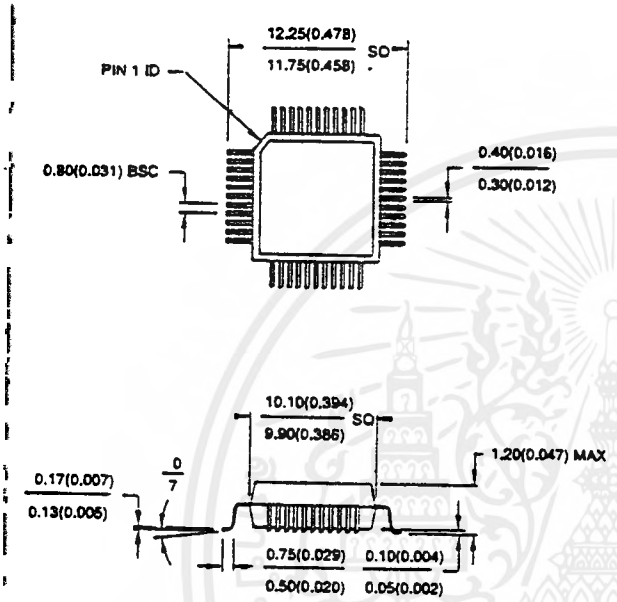
Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
12	5 V ± 10%	AT89C52-12DM AT89C52-12LM	40D6 44L	Military (-55°C to 125°C)
		AT89C52-12DM/883 AT89C52-12LM/883	40D6 44L	Military/883C Class B, Fully Compliant (-55°C to 125°C)
16	5 V ± 20%	AT89C52-16AA AT89C52-16JA AT89C52-16PA AT89C52-16QA	44A 44J 40P6 44Q	Automotive (-40°C to 125°C)
20	5 V ± 20%	AT89C52-20AC AT89C52-20JC AT89C52-20PC AT89C52-20QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C52-20AI AT89C52-20JI AT89C52-20PI AT89C52-20QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)
24	5 V ± 20%	AT89C52-24AC AT89C52-24JC AT89C52-24PC AT89C52-24QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C52-24AI AT89C52-24JI AT89C52-24PI AT89C52-24QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)

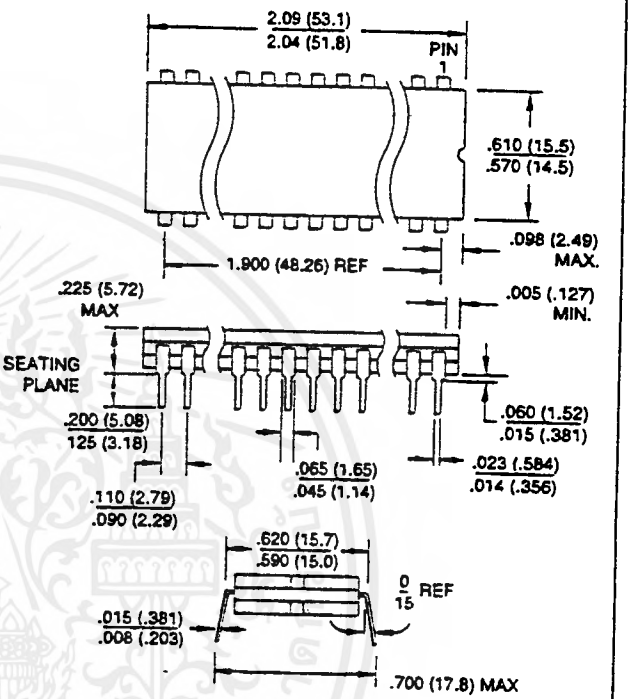
Package Type	
44A	44 Lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
40D6	40 Lead, 0.600" Wide, Non-Windowed, Ceramic Dual Inline Package (Cerdip)
44J	44 Lead, Plastic J-Leaded Chip Carrier (PLCC)
44L	44 Pad, Non-Windowed, Ceramic Leadless Chip Carrier (LCC)
40P6	40 Lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
44Q	44 Lead, Plastic Gull Wing Quad Flatpack (PQFP)

Packaging Information

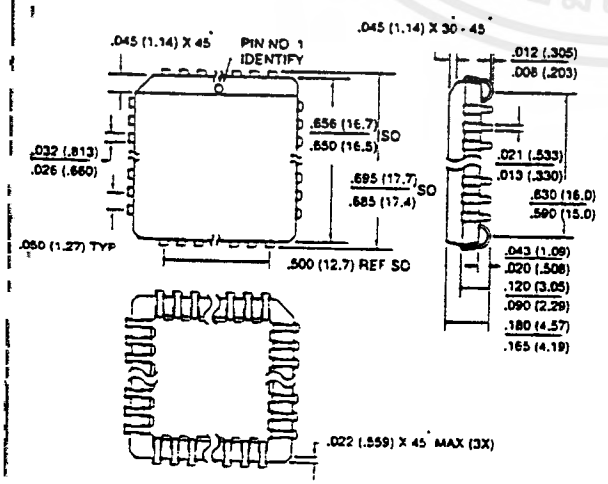
44A, 44 Lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
Dimensions in Millimeters and (Inches)



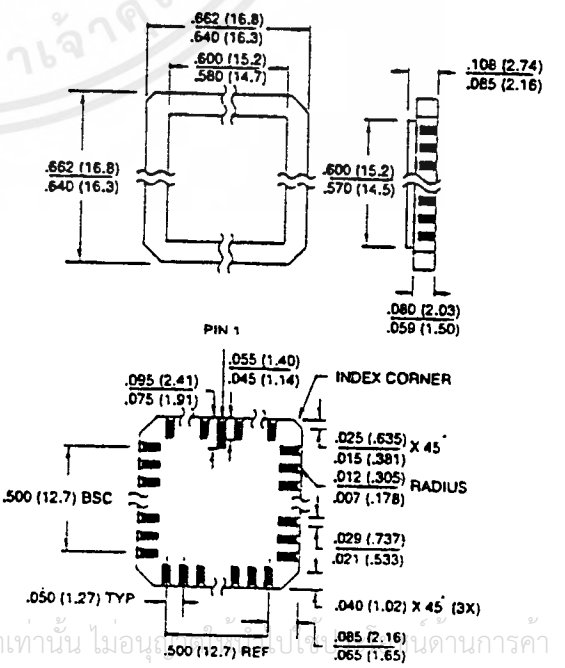
40D6, 40 Lead, 0.600" Wide, Non-Windowed, Ceramic Dual Inline Package (Cerdip)
Dimensions in Inches and (Millimeters)
MIL-M-38510 D-5 CONFIG 1



44J, 44 Lead, Plastic J-Leaded Chip Carrier (PLCC)
Dimensions in Inches and (Millimeters)
JEDEC OUTLINE MO-047 AC

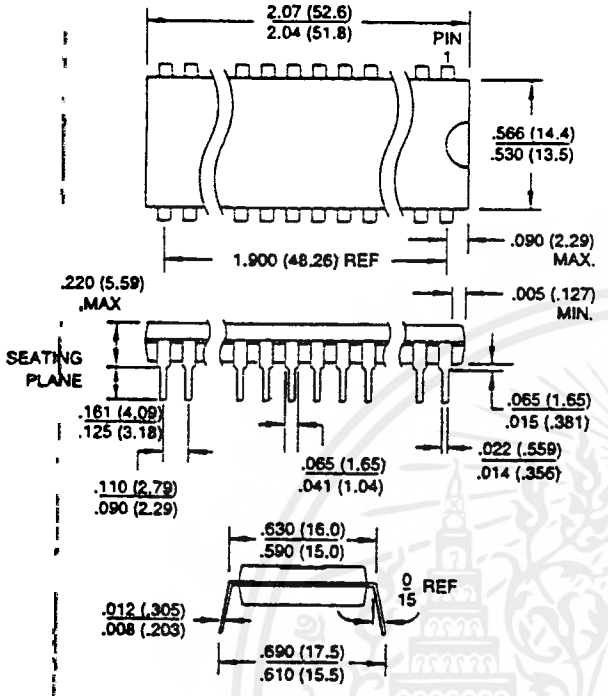


44L, 44 Pad, Non-Windowed, Ceramic Leadless Chip Carrier (LCC)
Dimensions in Inches and (Millimeters)

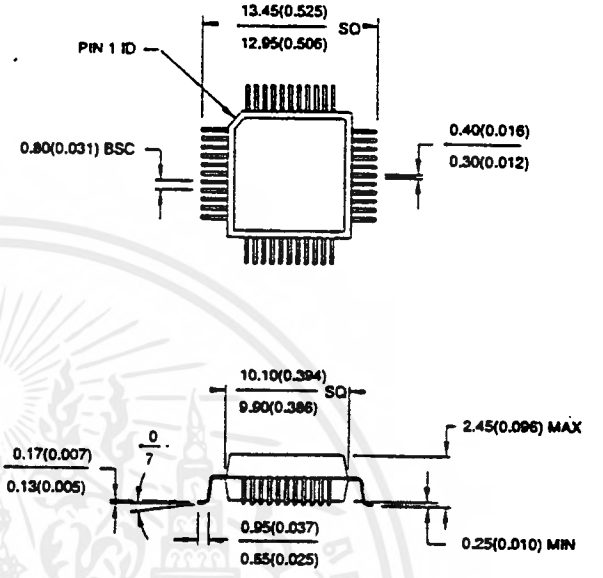


Packaging Information

40P6, 40 Lead,, 0.600" Wide,
Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)



44Q, 44 Lead, Plastic Gull Wing Quad Flatpack (PQFP)
Dimensions in Millimeters and (Inches)



54373 / 74373 Octal D-Type Transparent Latches and Edge-Triggered Flip-Flops

	Schottky TTL					High-Speed TTL					Low-Power Schottky TTL					Standard TTL					Low-Power T				
	Device Type	Package				Device Type	Package				Device Type	Package				Device Type	Package				Device Type	P			
		C	P	M	CF		C	P	M	CF		C	P	M	CF		C	P	M	CF		C			
T.I.	SN54S373	J (J)									SN54LS373	J (J)													
	SN74S373	J (J,N)									SN74LS373	J (J,N)													
FAIRCHILD																									
MOTOROLA																									
N.S.C.																									
PHILIPS																									
SIGNETICS																									
SIEMENS																									
FUJITSU																									
HITACHI																									
MITSUBISHI																									
NEC																									
TOSHIBA																									

Electrical Characteristics SN54LS373/SN74LS373

absolute maximum ratings over operating free-air temperature range

Supply voltage, V_{CC}	7V	Operating free-air temperature range	SN54LS	−55°C to 125°C
Input voltage	7V		SN74LS	0°C to 70°C
		Storage temperature range		−65°C to 150°C

recommend operating conditions

	SN54LS373			SN74LS373			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−1			−2.6	mA
High-level output voltage, V_{OH}			5.5			5.5	V
Pulse width, t_w Clock enable high	15			15			ns
Pulse width, t_w Clock enable high	15			15			ns
Setup time t_{SUTUP}	0			0			ns
Hold time, t_{HOLD}	10			10			ns
Operating free-air temperature, T_A	−55		125	0		70	°C

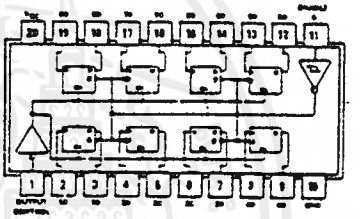
electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS‡	MIN	TYP‡	MAX	UNIT
V_{IH}	High-level input voltage		2			V
V_{IL}	Low-level input voltage				0.8	V
V_{IK}	Input clamp voltage	V_{CC} = MIN, I_I = −18mA			−1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max, I_{OH} = MAX	2.4	3.1		V
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max, I_{OL} = 24mA		0.35	0.5	V
I_{OZH}	Off-state output current, high-level voltage applied	V_{CC} = MAX, V_{IH} = 2V, V_O = 2.7V			20	μA
I_{OZL}	Off-state output current, low-level voltage applied	V_{CC} = MAX, V_{IH} = 2V, V_O = 0.4V			−20	μA
I_I	Input current at maximum input voltage	V_{CC} = MAX, V_I = 7V			0.1	mA
I_{IH}	High-level input current	V_{CC} = MAX, V_I = 2.7V			20	μA
I_{IL}	Low-level input current	V_{CC} = MAX, V_I = 0.4V			−0.4	mA
I_{OS}	Short-circuit output current §	V_{CC} = MAX	−30		−130	mA
I_{CC}	Supply current	V_{CC} = MAX, Output control at 4.5V, LS373		24	40	mA

switching characteristics, V_{CC} = 5V, T_A = 25°C

PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_{max}							MHz
t_{PLH}	Data	Any Q	C_L = 45pF, R_L = 667Ω, See Note 2 and 3		12	18	ns
t_{PHL}					12	18	
t_{PLH}	Clock or enable	Any Q			20	30	ns
t_{PHL}					18	30	
t_{PZH}	Output control	Any Q			15	28	ns
t_{PZL}					25	36	
t_{PHZ}	Output control	Any Q	C_L = 5pF, R_L = 667Ω, See Note 2		12	20	ns
t_{PLZ}					15	24	

Pin Assignments (Top View)



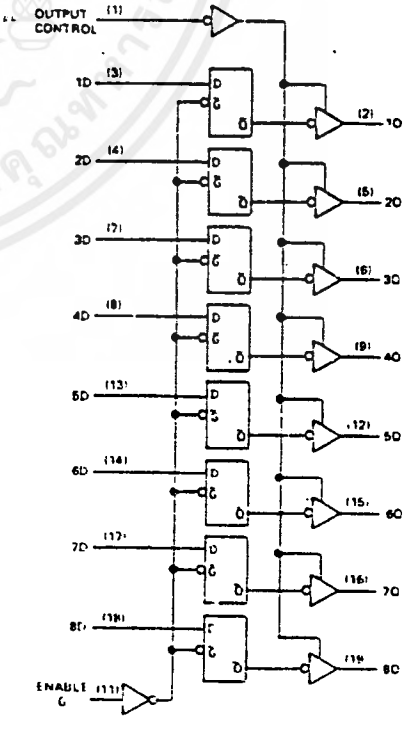
SN54LS373 (J) SN74LS373 (J, N)
SN54S373 (J) SN74S373 (J, N)

'LS373, 'S373
FUNCTION TABLE

OUTPUT CONTROL	ENABLE G	D	Q
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

'LS373, 'S373
TRANSPARENT LATCHES

'LS373, 'S373
TRANSPARENT LATCHES



A245 74245 Octal Bus Transceivers with 3-state Outputs

	Schottky TTL				High-Speed TTL				Low-Power Schottky TTL				Standard TTL				Low-Power TTL			
	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF
											SN54LS245			W	W										
											SN74LS245		J	N											
HILD																									
OROLA																									
C																									
PS																									
ETICS																									
ENS																									
TSU																									
OH																									
UBISHI																									
NBA																									

Electrical Characteristics SN54LS245/SN74LS245

absolute maximum ratings over operating free-air temperature range

y voltage, V_{CC}	7 V	Operating free-air temperature range	SN54LS	55°C to 125°C	
t voltage	7 V		SN74LS	0°C to 70°C	
		Storage temperature range		65°C to 150°C	

recommended operating conditions

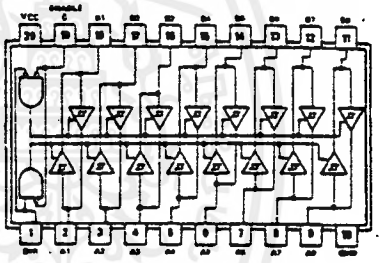
	SN54LS245			SN74LS245			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
y voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
igh-level output current, I_{OH}			-12			-15	mA
ow-level output current, I_{OL}			12			24	mA
ing free-air temperature, T_A	-55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range

PARAMETER	TEST CONDITIONS†		SN74LS245 MIN	TYP‡	MAX	UNIT
High-level input voltage			2			V
Low-level input voltage					0.8	V
Input clamp voltage	V_{CC} = MIN, I_I = -18mA				-1.5	V
Hysteresis ($V_{T+} - V_{T-}$) A or B input	V_{CC} = MIN		0.2	0.4		V
High-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max	I_{OH} = -3mA	2.4	3.4		V
		I_{OH} = MAX	2			
Low-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max	I_{OL} = 12mA			0.4	V
		I_{OL} = 24mA			0.5	
Off-state output current, high-level voltage applied	V_{CC} = MAX, G at 2V	V_O = 2.7 V			10	μA
Off-state output current, low-level voltage applied		V_O = 0.4V			-200	
Input current at maximum input voltage: A or B	V_{CC} = MAX,	V_I = 5.5V			0.1	mA
Input current at maximum input voltage: DIR or $\overline{G}$		V_I = 7V			0.1	
High-level input current	V_{CC} = MAX, V_{IH} = 2.7V				20	μA
Low-level input current	V_{CC} = MAX, V_{IL} = 0.4V				-0.2	mA
Short-circuit output current	V_{CC} = MAX		-40		-225	mA
Supply current: Total, outputs high	V_{CC} = MAX, Outputs open			48	70	mA
Supply current: Total, outputs low				62	90	
Supply current: Outputs at Hi-Z				64	95	

switching characteristics, V_{CC} = 5V, T_A = 25°C

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Propagation delay time, low-to-high-level output	C_L = 45pF, R_L = 667Ω, See Note 2		8	12	ns
Propagation delay time, high-to-low-level output			8	12	ns
Output enable time to low level			27	40	ns
Output enable time to high level			25	40	ns
Output disable time from low level	C_L = 5pF, R_L = 667Ω, See Note 2		15	25	ns
Output disable time from high level			15	25	ns



SN54LS245 (J) SN74LS245 (J, N)

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡ All typical values are at V_{CC} = 5V, T_A = 25°C.
§ Not more than one output should be shorted at a time, and duration of the short-short should not not exceed one second

บรรณานุกรม

1. ปรเมษฐ์ ประมยานันท์ และ ปิยะพงศ์ เผ่าวณิช , " คู่มือและการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ MCS-51 " (หน้า 9-94 , 110-116) , บริษัทซีเอ็ดยูเคชัน จำกัด มหาชน , 2537
2. ไพบูลย์ บุปผา , " เทคนิคการใช้แอลซีดีโมดูล " , เซมิคอนดัคเตอร์อิเล็กทรอนิกส์ (หน้า 48-55 ฉบับที่ 166) ธันวาคม 2539
3. สุเชียร เกียรตสุนทร , " PC/PLC " , (หน้า 9-104) , บริษัทซีเอ็ดยูเคชัน จำกัด มหาชน , 2536
4. สุนทร วิทูสุรพจน์ , " การใช้งานไมโครคอนโทรลเลอร์ตระกูล 8051 " , บริษัทซีเอ็ดยูเคชัน จำกัด มหาชน , 2537
5. ธันวา ศรีประมง , " การเขียนโปรแกรมภาษาซี สำหรับวิศวกรรม " ,มหาลัยเทคโนโลยีมหานคร
6. Frank D. Petruzella , " Programable Logic Controllers " , McGraw-Hill (pages 49-61)
7. Seth Bergmann , " Complier Design " , WM.C. Brown Publisher (pages 30-67)