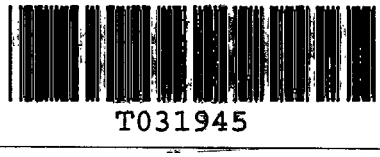
อินเซอร์กิตอีมูเลเตอร์ราคาประหยัดสำหรับ 8098 ไมโครคอนโทรลเลอร์

A LOW COST IN-CIRCUIT EMULATOR FOR 8098 MICROCONTROLLER

อิศราพงศ์ สินันตา

ITSARAPONG SINUNTA



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2541

ISBN 974-622-339-9

A LOW COST IN-CIRCUIT EMULATOR FOR 8098 MICROCONTROLLER

ITSARAPONG SINUNTA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1998

ISBN 974-622-339-9

หัวข้อวิทยานิพนธ์	อินเซอร์กิตอีมูเลเตอร์ราคาประหยัดสำหรับ 8098 ไมโครอนโทรลเลอร์
นักศึกษา	นายอิศราพงศ์ สินันตา
รหัสประจำตัว	36061032
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2541
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร.กอบชัย เดชหาญ

บทคัดย่อ

เครื่องมือที่ใช้สำหรับพัฒนาโครงงานซึ่งควบคุมการทำงานด้วยไมโครโปรเซสเซอร์ (μP) หรือไมโครคอนโทรลเลอร์ (μC) ที่มีอยู่ในปัจจุบันได้แก่ ซิงเกิ้ลบอร์ด, โปรแกรมซิมูเลเตอร์, โปรแกรมรีโมตมอนิเตอร์, อีพีรอมอีมูเลเตอร์, และอินเซอร์กิตอีมูเลเตอร์ เป็นต้น และเป็นที่ยอมรับกันโดยทั่วไปว่า อินเซอร์กิตอีมูเลเตอร์ เป็นเครื่องมือที่มีความคล่องตัว และให้ประสิทธิภาพในการพัฒนาโครงงานสูงที่สุด แต่ว่ามีราคาแพง จึงไม่เป็นที่นิยมใช้กันมากนัก วิทยานิพนธ์ฉบับนี้ได้ทำการวิจัยเกี่ยวกับเทคนิคการออกแบบ และสร้างอินเซอร์กิตอีมูเลเตอร์ ให้มีราคาถูก เพื่อใช้ในการพัฒนาโครงงานที่ควบคุมการทำงานด้วย 8098 ไมโครคอนโทรลเลอร์ขนาด 16 บิทตระกูล MCS-96 ของบริษัทอินเทล ซึ่งเป็นไมโครคอนโทรลเลอร์ที่มีขอบเขต และความสามารถสูง, สามารถหาซื้อได้ภายในประเทศ และเหมาะแก่การนำไปประยุกต์ใช้งานในด้านต่าง ๆ เช่น ระบบควบคุมแบบปิด, การประมวลผลสัญญาณดิจิตอลย่านความถี่กลาง, โมเด็ม, ระบบควบคุมมอเตอร์, ระบบควบคุมเครื่องยนต์, พริ้นท์เตอร์, เครื่องสำเนารูปถ่าย, ระบบเบรคแอนติล็อก, ระบบควบคุมเครื่องปรับอากาศ, เครื่องขับจานแม่เหล็ก และเครื่องมือวัดทางการแพทย์ เป็นต้น

Thesis Title	A Low Cost In-Circuit Emulator for 8098 Microcontroller
Student	Mr. Itsarapong Sinunta
Student ID.	36061032
Degree	Master of Engineering
Programme	Electrical Engineering
Year	1998
Thesis Advisor	Assoc. Prof. Dr.Kobchai Dejhan

ABSTRACT

There are many tools to develope electronic projects, which are controlled by microprocessor (μP) or microcontroller (μC) such as single board, simulator program, remote monitor program, EPROM emulator and in-circuit emulator. In-circuit emulator is the most powerful tool to provide the high efficiency for developing the electronic projects. The drawback is more expensive, so it is not preferable to use. This thesis concerns with the technical concept to design and implement of a low cost in-circuit emulator. It can be used to develope many electronic projects that have 8098 microcontroller as the control unit. 8098 is a member of MCS - 96 produced by Intel Corporation. It is very powerful, and easily to implement for any applications, such as closed-loop control system, mid-range digital signal processing, modem, motor control system, engine control system, printer, photo copier, anti-lock break system, air conditioner control system, disk drive, medical instrumentations and etc.

กิตติกรรมประกาศ

การจัดทำวิทยานิพนธ์ ในครั้งนี้สำเร็จลุล่วงไปได้ด้วยดีเพราะได้รับความเมตตาจาก รองศาสตราจารย์ ดร.กอบชัย เดชหาญ ที่ได้ให้ความกรุณาแนะนำแก่ผู้วิจัยตลอดมา อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือต่าง ๆ ในการทำวิทยานิพนธ์ฉบับนี้ ผู้วิจัยรู้สึกซาบซึ้ง และขอกราบขอบพระคุณเป็นอย่างสูง

และที่ลืมเสียมิได้คือ ขอกราบขอบพระคุณ คุณแม่ ที่เคารพรักยิ่ง ตลอดจนถึงญาติผู้ใหญ่ พี่ ๆ น้อง ๆ และเพื่อนร่วมงานทุก ๆ คน ที่ให้การสนับสนุน และเป็นกำลังใจที่ยิ่งใหญ่ ในการทำวิทยานิพนธ์ฉบับนี้

สุดท้ายนี้ ผู้วิจัยขอขอบพระคุณ โครงการวิทยาศาสตร์ และเทคโนโลยีเพื่อการพัฒนา สำนักงานพัฒนาวิทยาศาสตร์ และเทคโนโลยีแห่งชาติ ที่มอบทุน เพื่อสนับสนุนการศึกษาในระดับบัณฑิตศึกษาภายในประเทศ ทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงลงด้วยดี

อิศราพงศ์ สินันตา

สารบัญ

หน้า

บทคัดย่อภาษาไทย..... I

บทคัดย่อภาษาอังกฤษ.....II

กิตติกรรมประกาศ.....III

สารบัญ.....IV

สารบัญตาราง.....VIII

สารบัญภาพ..... IX

บทที่ 1 บทนำ..... 1

1.1 ความเป็นมาและความสำคัญของปัญหา..... 1

1.2 วัตถุประสงค์ของวิทยานิพนธ์..... 6

1.3 ประโยชน์ที่ได้รับจากวิทยานิพนธ์..... 7

1.4 ขอบเขตของวิทยานิพนธ์..... 7

บทที่ 2 ไมโครคอนโทรลเลอร์ตระกูล MCS-96..... 9

2.1 จุดเด่นของ MCS-96..... 10

2.2 สถาปัตยกรรมของ 8096..... 11

2.3 การจัดการหน่วยความจำ..... 17

2.4 โครงสร้างการอินเตอร์รัพต์..... 21

2.5 พอร์ตอินพุตเอาท์พุต 0, 1, 2, 3 และ 4..... 24

2.6 พอร์ตอนุกรม..... 27

2.7 ไทเมอร์..... 29

2.8 พอร์ตอินพุตความเร็วสูง (HSI)..... 30

2.9 พอร์ตเอาต์พุตความเร็วสูง (HSO)..... 32

2.10 การรับสัญญาณอะนาลอก..... 33

2.11 การสร้าง Pulse Width Modulation (PWM)..... 35

2.12 Chip Configuration Register (CCR)..... 36

2.13 แพคเก็จของไมโครคอนโทรลเลอร์ตระกูล MCS-96..... 36

2.14 หน้าที่การทำงานของขาต่าง ๆ..... 37

สารบัญ (ต่อ)

หน้า

บทที่ 3 การออกแบบทางด้านฮาร์ดแวร์และซอฟแวร์ 40

3.1 การออกแบบทางด้านฮาร์ดแวร์ 40

3.1.1 รายละเอียดของการออกแบบพ็อด (POD) 41

3.1.2 รายละเอียดของการออกแบบวงจรควบคุม (Control Circuit) 42

3.1.3 รายละเอียดการออกแบบ วงจรหน่วยความจำ (Memory Circuit) 45

3.2 การออกแบบทางด้านซอฟแวร์ 47

บทที่ 4 การทดลองใช้งาน 50

4.1 รายละเอียดทางด้านฮาร์ดแวร์ของบอร์ดเป้าหมายที่นำมาต่อใช้งาน 50

4.2 การเชื่อมต่อระหว่างอินเซอร์กิตอีมูเลเตอร์และบอร์ดเป้าหมาย 51

4.3 การเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์ 51

4.4 การเตรียมพร้อมอินเซอร์กิตอีมูเลเตอร์ก่อนการใช้งาน 52

4.5 การทดลองใช้งานคำสั่งต่างๆ 54

4.5.1 คำสั่ง New 55

4.5.2 คำสั่ง Open 57

4.5.3 คำสั่ง Open List File 59

4.5.4 คำสั่ง Save 61

4.5.5 คำสั่ง Save As 62

4.5.6 คำสั่ง Exit 64

4.5.7 คำสั่ง ASM96 66

4.5.8 คำสั่ง OH 67

4.5.9 คำสั่ง Auto 68

4.5.10 Run ในรูปแบบ Go 70

4.5.11 Run ในรูปแบบ Go forever 71

4.5.12 Run ในรูปแบบ Go from... 72

4.5.13 Run ในรูปแบบ Go from...forever... 74

4.5.14 Run ในรูปแบบ Go from...till... 76

4.5.15 Run ในรูปแบบ Go from...till...or... 78

สารบัญ (ต่อ)

หน้า

4.5.16 คำสั่ง Setup... (Breakpoints)..... 81

4.5.17 คำสั่ง Disable all (Breakpoints)..... 83

4.5.18 คำสั่ง Delete all (Breakpoints)..... 85

4.5.19 คำสั่ง Config (Setting...)..... 86

4.5.20 คำสั่ง Reset (Reset Hardware, Software)..... 88

4.5.21 คำสั่ง Data, Register..... 90

4.5.22 คำสั่ง About... 92

4.5.23 กรอบ Status :..... 93

4.5.24 คำสั่ง Down load..... 94

4.5.25 คำสั่ง Go..... 96

4.5.26 คำสั่ง Continue..... 97

4.5.27 คำสั่ง Start Address..... 99

4.5.28 คำสั่ง Edit..... 102

4.5.29 คำสั่ง Fill..... 105

4.5.30 คำสั่ง Move..... 108

4.5.31 คำสั่ง Change Resister..... 111

บทที่ 5 การประยุกต์ใช้งาน..... 114

5.1 ตัวอย่างที่ 1. ใช้ประกอบการศึกษาวิธีการกำหนดเลขที่อยู่ในโหมดต่างๆ (Addressing Modes) ของไมโครคอนโทรลเลอร์ตระกูล MCS-96 114

5.2 ตัวอย่างที่ 2. การพัฒนาโปรแกรมประยุกต์ใช้งานโดยอินเซอร์กิตอีมูเลเตอร์..... 129

บทที่ 6 สรุปผลและข้อเสนอแนะ..... 137

เอกสารอ้างอิง..... 142

สารบัญ (ต่อ)

หน้า

ภาคผนวก..... 144
ก. 8X9X Quick Reference..... 145
ข. โฟลว์ชาร์ตแสดงการทำงานของคำสั่งต่าง ๆ..... 160
ค. ผลงานวิจัยที่ได้รับการตีพิมพ์..... 173
ประวัติผู้เขียน..... 182

สารบัญตาราง

ตารางที่ หน้า

2.1 แสดงการกำหนดค่าของรีจิสเตอร์หน้าที่พิเศษ (SFR) ภายหลังเสร็จสิ้นการรีเซต.....16

2.1 (ต่อ) แสดงการกำหนดค่าของรีจิสเตอร์หน้าที่พิเศษ (SFR) ภายหลังเสร็จสิ้นการรีเซต..... 17

2.2 แสดงตำแหน่งและระดับความสำคัญของอินเตอร์รัพต์เวกเตอร์.....22

2.3 แสดงการกำหนดฟังก์ชั่นใช้งานพอร์ต 2 25

2.4 แสดงตำแหน่งขาที่ต่อกับบัสของระบบใน MCS-96.....25

2.5 แสดงหมายเลขซีพียูของไมโครคอนโทรลเลอร์ในตระกูล MCS-96.....36

3.1 แสดงแผนภูมิพื้นที่หน่วยความจำของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ.....42

3.2 แสดงตารางความจริงของไอซี 1..... 44

3.3 แสดงตารางความจริงของไอซี 2..... 44

5.1 แสดงผลลัพธ์ในการทำงานของบอร์ดเป้าหมายที่ต้องการตามตัวอย่างที่ 2..... 129

5.2 แสดงผลลัพธ์ในการทำงานของบอร์ดเป้าหมายในขณะที่กำลังพัฒนาโปรแกรม
โดยใช้อินเซอร์กิตอีมูเลเตอร์..... 135

5.3 แสดงผลลัพธ์ในการทำงานของบอร์ดเป้าหมายที่ควบคุมการทำงาน
ด้วยโปรแกรมมอนิเตอร์ ซึ่งบรรจุอยู่ภายในอีพีรอม.....136

6.1 แสดงการเปรียบเทียบราคาของอินเซอร์กิตอีมูเลเตอร์..... 137

6.2 แสดงการเปรียบเทียบคุณสมบัติและฟังก์ชั่นการใช้งานของอินเซอร์กิตอีมูเลเตอร์.....138

สารบัญภาพ

ภาพที่ หน้า

1.1 แสดงซิงเกิลบอร์ด ET-BOARD V5.0 ของบริษัท ETT CO., LTD..... 3
1.2 แสดงอีพีรอมอีมูเลเตอร์ Em128 ของบริษัท ETT CO., LTD. 4
1.3 แสดงจอภาพการทำงานของโปรแกรม ChipView-x96 ซิมูเลเตอร์
ของบริษัท ChipTools, Inc. 5
1.4 แสดงอินเซอร์กิตอีมูเลเตอร์ USP-51 ของบริษัท Signum Systems.....6
2.1 แสดงพัฒนาการเทคโนโลยีไมโครคอนโทรลเลอร์ของอินเทล.....10
2.2 แสดงโครงสร้างของชิพ 8096.....12
2.3 แสดงบล็อกไดอะแกรมของ RALU.....14
2.4 แสดงการต่อคริสตอลเพื่อสร้างความถี่ให้กับชิพ 8096.....15
2.5 แสดงสัญญาณนาฬิกาเมื่อเทียบกับ XTAL1..... 15
2.6 แสดงสภาวะการรีเซต.....16
2.7 แสดงพื้นที่หน่วยความจำของ 8096..... 18
2.8 แสดงพื้นที่ของรีจิสเตอร์ไฟล์.....19
2.9 แสดงแหล่งกำเนิดของอินเตอร์รัพต์.....22
2.10 แสดงบล็อกไดอะแกรมระบบอินเตอร์รัพต์ของ 8096..... 23
2.11 แสดงรีจิสเตอร์ Interrupt Pending.....24
2.12 แสดงรีจิสเตอร์ควบคุม 0 (IOC0).....26
2.13 แสดงรีจิสเตอร์ควบคุม 1 (IOC1).....26
2.14 แสดงรีจิสเตอร์แสดงสถานะ I/O 0 (IOS 0)..... 26
2.15 แสดงรีจิสเตอร์แสดงสถานะ I/O 1 (IOS 1)..... 27
2.16 แสดงรีจิสเตอร์คำสั่ง/สถานะของพอร์ตอนุกรม.....27
2.17 แสดงไดอะแกรมเวลาพอร์ตอนุกรมโหมด 0..... 28
2.18 แสดงตัวอย่างวงจรพอร์ตอนุกรมในโหมด 0..... 28
2.19 แสดงเฟรมของข้อมูลพอร์ตอนุกรม โหมด 1..... 28
2.20 แสดงเฟรมข้อมูลพอร์ตอนุกรมโหมด 2 และ 3..... 29
2.21 แสดงการป้อนสัญญาณนาฬิกาและการรีเซตไทเมอร์ 2..... 30
2.22 แสดงบล็อกไดอะแกรมของหน่วย HSI..... 31
2.23 แสดงบิตของรีจิสเตอร์กำหนดโหมด.....31

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

2.24 แสดงการควบคุมผ่านรีจีสเตอร์ IOC0..... 32
2.25 แสดงรีจีสเตอร์แสดงสถานะของ HSI..... 32
2.26 แสดงบล็อกไดอะแกรมของหน่วย HSO..... 33
2.27 แสดงรูปแบบคำสั่งของ HSO COMMAND..... 33
2.28 แสดงรีจีสเตอร์คำสั่งของ A/D..... 34
2.29 แสดงผลลัพธ์ของ A/D เก็บไว้ในรีจีสเตอร์ 02H และ 03H..... 34
2.30 แสดงบล็อกไดอะแกรมของ PWM..... 35
2.31 แสดงเอาท์พุตที่ได้จาก PWM..... 35
2.32 แสดง Chip Configuration Register..... 36
2.33 แสดงตัวอย่างของแพคเก็จแบบ 48 ขา..... 37
2.34 แสดงตัวอย่างของแพคเก็จแบบ 68 ขา..... 37
3.1 แสดงฮาร์ดแวร์ของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ..... 40
3.2 แสดงบล็อกไดอะแกรมของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ..... 41
3.3 แสดงพ็อดของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ..... 42
3.4 แสดงวงจรควบคุมของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ..... 43
3.5 แสดงวงจรลอจิกเกตภายในไอซี 3 ที่ออกแบบ..... 45
3.6 แสดงวงจรหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ..... 45
4.1 แสดง KS-96 BOARD MCS-96 MICROCONTROLLER..... 50
4.2 แสดงการเชื่อมต่อระหว่างอินเซอร์กิตอีมูเลเตอร์และบอร์ดเป้าหมาย..... 51
4.3 แสดงการเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์
โดยการใช้สายสัญญาณของพอร์ตอนุกรม RS-232..... 52
4.4 แสดงการเรียกใช้งานโปรแกรม ICE8089 ภายใต้ระบบปฏิบัติการ WINDOWS95..... 53
4.5 แสดงหน้าต่างหลักของโปรแกรม ICE8098..... 53
4.6 แสดงเมนู File และส่วนประกอบของคำสั่งภายใน..... 54
4.7 แสดงการใช้งานคำสั่ง New..... 55
4.8 แสดงหน้าต่างของแฟ้มข้อมูล “Untitled” 56
4.9 แสดงการใช้งานคำสั่ง OPEN..... 57
4.10 แสดงหน้าต่างสำหรับให้ผู้ใช้งานระบุชื่อของโปรแกรมที่ต้องการพัฒนา..... 57

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

4.11 แสดงตัวอย่างของแฟ้มข้อมูลเก่าที่ทำการเปิดขึ้นมาด้วยคำสั่ง OPEN เพื่อทำการพัฒนา.....58
4.12 แสดงการใช้งานคำสั่ง Open List File.....59
4.13 แสดงตัวอย่างแฟ้มข้อมูลแบบ List ของโปรแกรมที่กำลังพัฒนาอยู่ในขณะนั้น.....60
4.14 แสดงการใช้งานคำสั่ง Save.....61
4.15 แสดงการใช้งานคำสั่ง Save As.....62
4.16 แสดงหน้าต่างที่ให้ผู้ใช้ระบุชื่อใหม่สำหรับการบันทึกแฟ้มข้อมูล
ที่กำลังพัฒนาอยู่ในขณะนั้น.....62
4.17 แสดงตัวอย่างแฟ้มข้อมูลที่ได้จากการบันทึกด้วยคำสั่ง Save As.....63
4.18 แสดงการใช้งานคำสั่ง Exit.....64
4.19 แสดงเมนู Assembler และส่วนประกอบของคำสั่งภายใน.....65
4.20 แสดงการใช้งานคำสั่ง ASM96.....66
4.21 แสดงหน้าต่างที่แจ้งให้ผู้ใช้ทราบว่าได้ทำการแอสแซมเบลอร์เสร็จสิ้นแล้ว.....66
4.22 แสดงหน้าต่างที่แจ้งให้ผู้ใช้ทราบผลของการแอสแซมเบลอร์.....66
4.23 แสดงการใช้งานคำสั่ง OH.....67
4.24 แสดงหน้าต่างที่แจ้งให้ผู้ใช้ทราบว่าได้ทำการแปลงแฟ้มข้อมูลแบบ Object File ไปเป็น
Hex File เสร็จสิ้นแล้ว.....67
4.25 แสดงการใช้งานคำสั่ง Auto.....68
4.26 แสดงหน้าต่างในการ Down Load ข้อมูลแบบ Hex File ของโปรแกรมที่กำลังพัฒนา
อยู่ในขณะนั้น จากเครื่องไมโครคอมพิวเตอร์ไปยังหน่วยความจำของอีมูเลเตอร์.....68
4.27 แสดงเมนู Run.....69
4.28 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go.....70
4.29 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go forever.....71
4.30 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from... 72
4.31 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้นที่ต้องการ
ในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from... 73
4.32 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...forever.....74
4.33 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้นที่ต้องการ
ในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...forever.....75

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

4.34 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...till... 76
4.35 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น และกำหนดแอดเดรสสุดท้ายที่ต้องการหยุดทำงาน ในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...till..... 77
4.36 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...till...or... 78
4.37 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น และกำหนดแอดเดรสสุดท้าย จำนวน 2 ตำแหน่งที่ต้องการหยุดทำงานในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...till...or... 79
4.38 แสดงเมนู Breakpoints.....80
4.39 แสดงการใช้งานคำสั่ง Setup... 81
4.40 แสดงหน้าต่างที่ให้ผู้ใช้กำหนดแอดเดรสที่ต้องการหยุดทำงาน..... 82
4.41 แสดงการใช้งานคำสั่ง Disable all..... 83
4.42 แสดงหน้าต่างให้ผู้ใช้สามารถเรียกค่าเบรคพอยบัฟเฟอร์มาใช้งานได้อีก โดยคลิกเมาส์ที่ Check box ของเบรคพอยบัฟเฟอร์ที่ต้องการใช้งาน.....84
4.43 แสดงการใช้งานคำสั่ง Delete all..... 85
4.44 แสดงการใช้งานคำสั่ง Config (Setting...)..... 86
4.45 แสดงหน้าต่างที่ให้ผู้ใช้กำหนดตำแหน่งไดเร็กทอรีของแฟ้มข้อมูลแอสแซมบลี, แอดเดรสเริ่มต้นของหน่วยความจำข้อมูล และหมายเลขพอร์ตอนุกรมของเครื่องไมโครคอมพิวเตอร์ ที่ใช้ในการเชื่อมต่อกับบอร์ดเป้าหมาย..... 87
4.46 แสดงการใช้งานคำสั่ง Reset (Reset Hardware, Software)..... 88
4.47 แสดงหน้าต่างของขั้นตอนการรีเซตฮาร์ดแวร์และซอฟแวร์..... 88
4.48 แสดงการใช้งานเมนู Window..... 89
4.49 แสดงการใช้งานคำสั่ง Data, Register..... 90
4.50 แสดงหน้าต่างของหน่วยความจำข้อมูล และรีจิสเตอร์ของอินเซอร์กิตอีมูเลเตอร์.....91
4.51 แสดงการใช้งานคำสั่ง About... 92
4.52 แสดงหน้าต่างของรายละเอียดเกี่ยวกับโปรแกรม ICE8098.....93
4.53 แสดงการใช้งานปุ่ม Down load 94
4.54 แสดงหน้าต่างของขั้นตอนการ Down Load ที่ผู้ใช้ต้องปฏิบัติตาม เมื่อคลิกเมาส์ที่ปุ่ม Down Load.....94

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

4.55 แสดงผลลัพธ์ที่ได้จากการใช้งานปุ่ม Down Load.....95

4.56 แสดงการใช้งานปุ่ม Go.....96

4.57 แสดงตัวอย่างหน้าต่างวิธีการหยุดการทำงานของอินเซอร์กิตอีมูเลเตอร์ เมื่อใช้งานคำสั่ง Go ในรูปแบบการทำงานแบบ Go forever และ Go from... forever ตามลำดับ.....96

4.58 แสดงการใช้งานปุ่มคำสั่ง Continue.....97

4.59 แสดงผลลัพธ์เมื่อคลิกเมาส์ที่ปุ่ม Continue..... 98

4.60 แสดงการใช้คำสั่ง Start Address ในหน้าต่าง Data Memory.....99

4.61 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดตำแหน่งแอดเดรสเริ่มต้นของข้อมูล ในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์..... 100

4.62 แสดงข้อมูลในหน้าต่าง Data Memory หลังจากที่ได้ทำการกำหนดตำแหน่ง แอดเดรสเริ่มต้นเป็น 1000H โดยใช้คำสั่ง Start Address.....101

4.63 แสดงการใช้งานคำสั่ง Edit ในหน้าต่าง Data Memory.....102

4.64 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรส และข้อมูลที่ต้องการแก้ไข ในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์.....103

4.65 แสดงหน้าต่าง Data Memory หลังจากทำการแก้ไขข้อมูลที่แอดเดรส 1000H โดยใช้คำสั่ง Edit..... 104

4.66 แสดงการใช้งานคำสั่ง Fill ในหน้าต่าง Data Memory..... 105

4.67 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น, แอดเดรสสุดท้าย และข้อมูล ที่ต้องการเขียนลงในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์..... 106

4.68 แสดงผลลัพธ์ที่ได้จากการทำงานของคำสั่ง Fill ภายในหน้าต่าง Data Memory.....107

4.69 แสดงการใช้งานคำสั่ง Move ภายในหน้าต่าง Data Memory.....108

4.70 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น, แอดเดรสสุดท้าย และแอดเดรสปลายทางของหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์.....109

4.71 แสดงผลลัพธ์ที่ได้จากการใช้งานคำสั่ง Move ภายในหน้าต่าง Data Memory.....110

4.72 แสดงการใช้งานคำสั่ง Change Register ภายในหน้าต่าง Registers.....111

4.73 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดค่าใหม่ของรีจิสเตอร์.....112

4.74 แสดงผลลัพธ์ที่ได้จากการใช้งานคำสั่ง Change Register ภายในหน้าต่าง Registers.....113

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

5.1 แสดงแฟ้มข้อมูล Exam_a.a96....116

5.2 แสดงแฟ้มข้อมูลแบบ List ที่ได้จากการแอสแซมเบลอร์แฟ้มข้อมูล Exam_a.a96.... 117

5.3 แสดงข้อมูลในหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์ เมื่อได้รับการโหลดข้อมูล เสร็จสิ้นแล้ว โดยเริ่มจากแอดเดรส 2080H....118

5.4 แสดงการกำหนดเบรคพอยน์แอดเดรส หรือตำแหน่งที่ต้องการให้อินเซอร์กิตอีมูเลเตอร์ หยุดทำงาน....119

5.5 แสดงการเปลี่ยนตำแหน่งเริ่มต้นของหน่วยความจำภายในหน้าต่าง Data Memory ให้เป็น 0050H โดยใช้คำสั่ง Start Address....120

5.6 แสดงการป้อนข้อมูล 00H ให้กับหน่วยความจำระหว่างแอดเดรส 0050H ถึง 008FH โดยใช้คำสั่ง Fill....121

5.7 แสดงข้อมูลในหน้าต่าง Data Memory ภายหลังการใช้คำสั่ง Fill....122

5.8 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Go....123

5.9 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง LD REG2, #1234H)....124

5.10 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง LD REG1, REG2).... 125

5.11 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง ST REG1, REG3)....126

5.12 แสดงหน้าต่าง Go Forever เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง SJMP HERE).... 127

5.13 แสดงสภาวะเตรียมพร้อมของอินเซอร์กิตอีมูเลเตอร์ นั่นคือ Status : จะแสดง ข้อความ Ready และตัวนับโปรแกรม (PC) ในหน้าต่าง Register จะมีค่าเท่ากับ 2080H... 128

5.14 แสดงโฟลว์ชาร์ตการทำงานของโปรแกรมในตัวอย่างที่ 2.... 130

5.15 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์....131

5.16 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)..... 132

5.17 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)..... 133

5.18 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)..... 133

5.19 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)..... 134

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

5.20 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ).....134
5.21 แสดงการใช้ออสซิลโลสโคปตรวจวัดสัญญาณนาฬิกาที่พอร์ตเอาท์พุตความเร็วสูง (HSO.0) เมื่อปรับสวิทช์หมายเลข 6 ที่ต่อกับอินพุตพอร์ตซี (C) ของไอซี 8255 ให้มีระดับลอจิกสูง ในขณะที่กำลังพัฒนาโปรแกรมโดยใช้อินเซอร์กิตอีมูเลเตอร์.....135
5.22 แสดงการใช้ออสซิลโลสโคปตรวจวัดสัญญาณนาฬิกาที่พอร์ตเอาท์พุตความเร็วสูง (HSO.0) เมื่อปรับสวิทช์หมายเลข 6 ที่ต่อกับอินพุตพอร์ตซี (C) ของไอซี 8255 ให้มีระดับลอจิกสูง ในขณะที่บอร์ดเป้าหมายถูกควบคุมการทำงานด้วยโปรแกรมมอนิเตอร์ซึ่งบรรจุอยู่ภายในอีพีรอม.....136

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา [1], [2], [3]

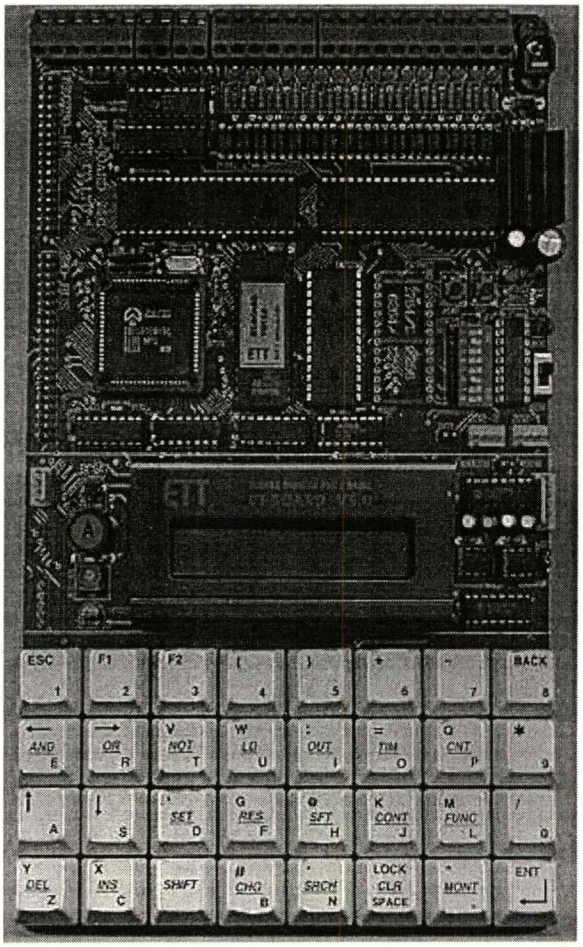
จากอดีตที่ผ่านมาเมื่อต้องการศึกษาในเรื่องไมโครโปรเซสเซอร์ขนาด 8 บิตวงการอิเล็กทรอนิกส์มักจะกล่าวถึง "Z80 CPU" และอาจมีหลายท่านยกย่องให้ Z80 CPU เป็น "ชิพบรมครู" เนื่องจากการที่ท่านเหล่านี้มีความรู้ ความชำนาญทางด้านซอฟแวร์และฮาร์ดแวร์อยู่ได้มาจนถึงทุกวันนี้ ก็เพราะได้ศึกษา และทำความเข้าใจในรายละเอียดต่างๆ จาก Z80 CPU นั่นเอง ในปัจจุบัน Z80 CPU ยังคงเป็นที่นิยมใช้สำหรับผู้ที่ต้องการเริ่มต้นศึกษาในเรื่องเกี่ยวกับไมโครโปรเซสเซอร์อยู่ ทั้งนี้เพราะ Z80 CPU มีโครงสร้างที่ไม่ซับซ้อน และใช้ชุดคำสั่งที่เข้าใจง่าย แต่ในขณะเดียวกัน ไมโครคอนโทรลเลอร์ขนาด 8 บิตตระกูล MCS-51 ของบริษัท INTEL ก็เริ่มเข้ามามีบทบาทที่สำคัญต่อการศึกษา และการนำไปประยุกต์ใช้งานทางอุตสาหกรรม ดังจะเห็นได้จากการที่สถานศึกษาหลายๆ แห่งได้จัดให้มีการเรียนการสอนเกี่ยวกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 ให้กับนักศึกษาผู้ที่มีพื้นฐานความรู้เกี่ยวกับไมโครโปรเซสเซอร์ Z80 CPU เหตุผลประการสำคัญที่ไมโครคอนโทรลเลอร์ตระกูล MCS-51 เข้ามามีบทบาทอย่างรวดเร็ว และมีผู้นิยมนำไปประยุกต์ใช้งานกันอย่างแพร่หลายนั้น เนื่องจากไมโครคอนโทรลเลอร์ตระกูล MCS-51 นี้มีขอบเขต และความสามารถสูงกว่าไมโครโปรเซสเซอร์ธรรมดาทั่วๆ ไปอยู่หลายประการ เช่น ประกอบด้วยหน่วยความจำแบบ ROM และ RAM ภายในชิพ, มีวงจรกำเนิดสัญญาณนาฬิกาและวงจรนับ, มีพอร์ตอินพุตเอาท์พุตแบบขนานและอนุกรม ทำให้สามารถต่อใช้งานได้ทันทีเป็นต้น และในปัจจุบันนี้ผู้ที่มีประสพการณ์ในการใช้งานไมโครคอนโทรลเลอร์ตระกูล MCS-51 เป็นอย่างดีแล้ว เริ่มให้ความสนใจที่จะศึกษาไมโครคอนโทรลเลอร์ตระกูล MCS-96 ของบริษัท INTEL เพื่อยกระดับการพัฒนาไมโครคอนโทรลเลอร์ และใช้คุณสมบัติที่ดีกว่าของไมโครคอนโทรลเลอร์ตระกูลนี้ในการเพิ่มประสิทธิภาพของการประยุกต์ใช้งาน เช่น มีการขยายโครงสร้างระบบบัสและรีจิสเตอร์ที่ใช้ในการคำนวณจากขนาด 8 บิตเป็น 16 บิต, มีพื้นที่หน่วยความจำแบบ ROM ภายในชิพขนาด 8 กิโลไบต์ทำให้สามารถเก็บโปรแกรมที่มีความซับซ้อนและยุ่งยากมากขึ้น, มีหน่วยความจำแบบ RAM ภายในชิพขนาด 232 ไบต์, มีวงจรฮาร์ดแวร์การคูณและหาร ทำให้สามารถทำการคำนวณได้รวดเร็วขึ้น, มีพอร์ตอินพุตเอาต์พุตความเร็วสูง, มีวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลภายในชิพทำให้สามารถเชื่อมต่อกับอินพุตที่เป็นสัญญาณอะนาลอกได้โดยตรง, มีเอาต์พุต PWM ที่สามารถควบคุมความกว้างของพัลส์ได้, มีวงจรวอตช์ดอกซึ่งทำหน้าที่เตือนและกระตุ้นให้ระบบทำงานโดยอัตโนมัติ เมื่อระบบเกิดการผิดพลาด นอกจากนี้ระบบ

การอินเตอร์รัพต์ของไมโครคอนโทรลเลอร์ตระกูล MCS-96 ยังได้รับการปรับปรุงให้มีประสิทธิภาพสูงขึ้นทำให้การเขียนโปรแกรมประยุกต์ใช้งานสะดวกและง่ายขึ้น เป็นต้น

วิธีการพัฒนาโครงงานที่ใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุมนั้น สามารถทำได้หลายวิธีได้แก่

1. การพัฒนาโครงงานด้วยซิงเกิลบอร์ด (Single Board)

วิธีนี้ผู้ใช้จะต้องทำการแปลโปรแกรมที่เขียนขึ้นให้เป็นภาษาเครื่อง (Opcode) โดยการเปิดรหัสจากตารางชุดคำสั่ง เมื่อได้ Opcode แล้งจึงนำไปป้อนให้กับซิงเกิลบอร์ด เพื่อทดสอบการทำงานของโปรแกรมว่าถูกต้องตรงตามที่ต้องการหรือไม่ หากพบว่าโปรแกรมยังไม่สมบูรณ์ก็จะต้องทำการแก้ไขโปรแกรม และเริ่มต้นตามขั้นตอนเดิมอีก จนกว่าจะได้โปรแกรมที่สมบูรณ์ และมีความถูกต้องที่สุด ข้อดีของการพัฒนาด้วยซิงเกิลบอร์ดคือ มีราคาถูก, เหมาะสำหรับผู้ที่เริ่มต้นศึกษาไมโครคอนโทรลเลอร์ เนื่องจากโปรแกรมมอนิเตอร์ของซิงเกิลบอร์ดจะประกอบด้วยโปรแกรมสนับสนุนในการเรียนรู้ทำให้ง่ายต่อการเขียนโปรแกรม และสามารถนำเอาซิงเกิลบอร์ดไปประยุกต์ใช้เป็นเครื่องมือต่างๆ ได้นอกเหนือจากการใช้สำหรับพัฒนาโครงงานเช่น ใช้เป็นเครื่องกำเนิดความถี่, ใช้เป็นตัวนับพัลซ์ หรือใช้เป็นตัวนับความถี่เป็นต้น ข้อเสียของการพัฒนาด้วยซิงเกิลบอร์ดได้แก่ ผู้ใช้ไม่สามารถเขียนโปรแกรมในการใช้งานไมโครคอนโทรลเลอร์ได้ทุกส่วน เนื่องจากโปรแกรมมอนิเตอร์ของซิงเกิลบอร์ดได้ใช้งานบางส่วนของไมโครคอนโทรลเลอร์ไปแล้วเช่น พื้นที่หน่วยความจำโปรแกรม, อินเตอร์รัพพ์เวกเตอร์เป็นต้น นอกจากนี้แล้วการเปิดรหัสจากตารางชุดคำสั่งเพื่อทำการแปลโปรแกรมนั้นใช้เวลานาน ทำให้ไม่สะดวกต่อการพัฒนาโปรแกรมที่ซับซ้อนหรือมีขนาดใหญ่



ภาพที่ 1.1 แสดงชิงเกิลบอร์ด ET-BOARD V5.0 ของบริษัท ETT CO., LTD.

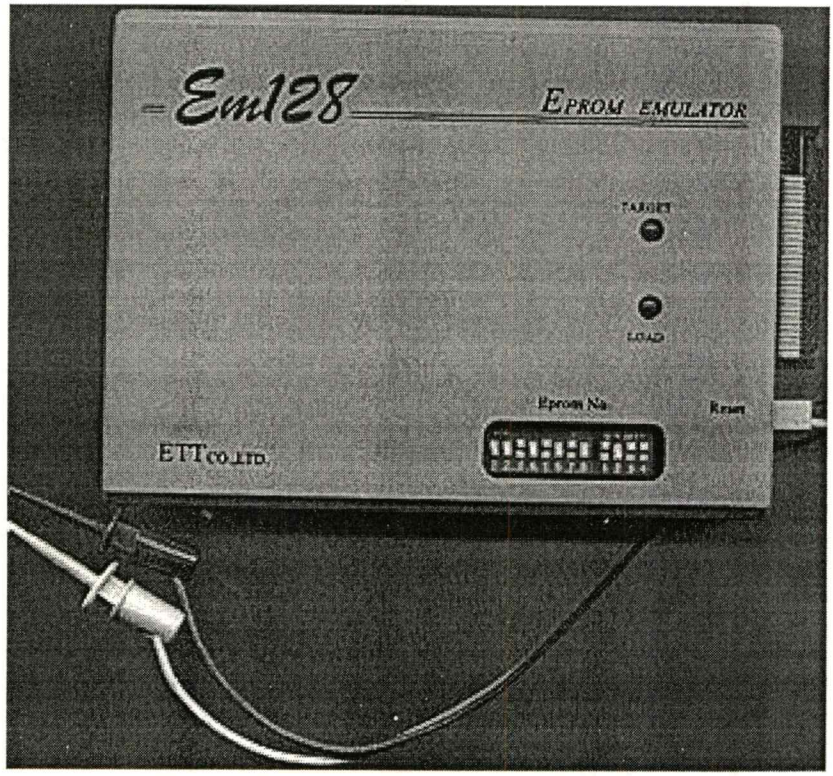
2. การพัฒนาโดยใช้โปรแกรมรีโมตมอนิเตอร์ (Remote Monitor)

โปรแกรมรีโมตมอนิเตอร์จะเก็บอยู่ภายใน อีพีรอม (EPROM) วิธีนี้จะต้องต่ออีพีรอมเข้ากับหน่วยความจำของไมโครคอนโทรลเลอร์บนระบบที่ผู้ใช้กำลังทำการพัฒนาอยู่ และจำเป็นต้องมีพอร์ตอนุกรมเพื่อใช้ในการสื่อสารด้วย ระบบที่กำลังพัฒนาจะอยู่ภายใต้การควบคุมของโปรแกรมรีโมตมอนิเตอร์ ซึ่งจะทำการติดต่อกับเครื่องไมโครคอมพิวเตอร์เพื่อแสดงผลบนจอภาพของเครื่อง และรับคำสั่งควบคุมมาจากเครื่องไมโครคอมพิวเตอร์ การพัฒนาด้วยวิธีนี้ผู้ใช้สามารถเขียนโปรแกรม, แก้ไข และทดลองการทำงานได้โดยที่ไม่มีการโยกย้ายทางกายภาพ สิ่งที่ปรากฏบนจอภาพจะเป็นสิ่งที่เกิดขึ้นจริงในระบบที่กำลังพัฒนา ลักษณะการใช้งานจะคล้ายคลึงกับโปรแกรม DEBUG ของ DOS ข้อดีของการพัฒนาด้วยโปรแกรมรีโมตมอนิเตอร์คือ มีราคาถูกและมีความยุ่งยากน้อยมาก เนื่องจากประกอบด้วยอีพีรอมเพียงตัวเดียว ส่วนข้อเสียได้แก่ ผู้ใช้ไม่สามารถเขียนโปรแกรมในการใช้งานไมโครคอนโทรลเลอร์ได้ทุกส่วน เนื่องจากโปรแกรมรีโมตมอนิเตอร์ได้ใช้งานบางส่วนของไมโครคอนโทรลเลอร์ไปแล้วเช่น พื้นที่หน่วยความจำโปรแกรม, อินเตอร์รัพพ์เวกเตอร์เป็นต้น นอกจากนี้ตำแหน่งของโปรแกรมที่ทำการพัฒนาจะต้องเริ่มต้นที่

แอดเดรสอื่นๆ ของหน่วยความจำโปรแกรมซึ่งไม่ใช่แอดเดรสเริ่มต้นที่แท้จริง เนื่องจากแอดเดรสเริ่มต้นนั้นถูกใช้โดยโปรแกรมรีโมตมอนิเตอร์ไปแล้ว

3. การพัฒนาโดยใช้อีพีรอมอีมูเลเตอร์ (EPROM Emulator)

อีพีรอมอีมูเลเตอร์จะทำหน้าที่เสมือนตัว EPROM หรือ RAM ของโครงงานที่กำลังพัฒนาอยู่ ในการใช้งานนั้นแผงวงจรด้านหนึ่งของอีพีรอมอีมูเลเตอร์ซึ่งมีสายสัญญาณขนาดเท่ากับตัว EPROM จะถูกต่อเข้ากับหน่วยความจำโปรแกรมของระบบที่กำลังพัฒนาอยู่ สำหรับปลายอีกด้านหนึ่งจะเชื่อมต่อเข้ากับเครื่องไมโครคอมพิวเตอร์ผ่านทางพอร์ตขนาน หรือพอร์ตอนุกรม เพื่อทำการส่งรหัสข้อมูลของโปรแกรมที่ผ่านการแอสแซมเบลอร์แล้วไปเก็บไว้ในตัวอีพีรอมอีมูเลเตอร์ ข้อดีของการพัฒนาด้วยอีพีรอมอีมูเลเตอร์คือ มีราคาถูก, สามารถพัฒนาโปรแกรมได้อย่างรวดเร็ว และลดขั้นตอนในการเขียนโปรแกรมหรือลบโปรแกรมให้กับตัวอีพีรอมที่ใช้งานจริง ข้อเสียของการพัฒนาด้วยวิธีนี้คือ มีข้อจำกัดของเบอร์อีพีรอมที่สามารถใช้แทนได้, ใช้แหล่งจ่ายไฟร่วมกับระบบที่กำลังพัฒนาหากแหล่งจ่ายไฟไม่เพียงพอจะส่งผลให้ไม่สามารถทำการพัฒนาได้

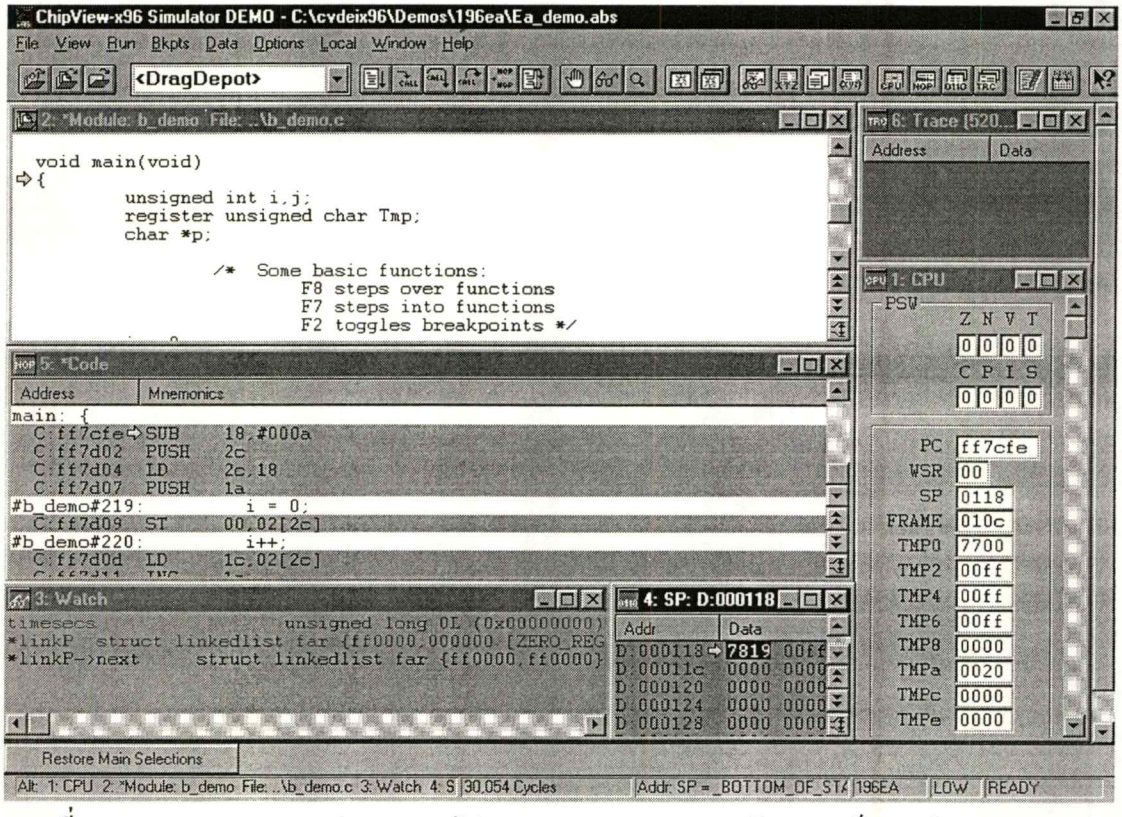


ภาพที่ 1.2 แสดงอีพีรอมอีมูเลเตอร์ Em128 ของบริษัท ETT CO., LTD.

4. การพัฒนาโดยใช้โปรแกรมซิมูเลเตอร์ (Program Simulator)

โปรแกรมซิมูเลเตอร์เป็นการจำลองการทำงานของไมโครคอนโทรลเลอร์โดยใช้ซอฟแวร์ที่ให้ผลลัพธ์ได้เช่นเดียวกับเมื่อทำงานด้วยฮาร์ดแวร์ เพียงแต่สภาวะของสัญญาณจากวงจรจริงๆ นั้นเป็นการจำลองการทำงานด้วยซอฟแวร์ที่ทำงานบนเครื่องไมโครคอมพิวเตอร์ทั้งสิ้น

ข้อดีของการพัฒนาด้วยโปรแกรมซิมูเลเตอร์คือ มีราคาถูก, เหมาะสำหรับผู้ที่ต้องการเริ่มศึกษาไมโครคอนโทรลเลอร์ เนื่องจากสามารถกำหนดรายละเอียดของการแสดงผลส่วนประกอบต่างๆ ของไมโครคอนโทรลเลอร์ เช่นรีจิสเตอร์, ข้อมูลภายในหน่วยความจำ และส่วนอื่นๆ ทำให้การทดสอบ และตรวจหาข้อผิดพลาดภายในโปรแกรมที่กำลังพัฒนาง่ายขึ้นโดยไม่ต้องใช้ฮาร์ดแวร์อื่นเพิ่มเติม ข้อเสียของการพัฒนาด้วยวิธีนี้คือ มีขอบเขตของการพัฒนาที่จำกัด เนื่องจากเป็นการจำลองสัญญาณต่างๆ ด้วยซอฟแวร์ ในบางกรณีเมื่อนำเอาโปรแกรมที่พัฒนาเสร็จแล้วไปใช้งานจริงไม่ได้

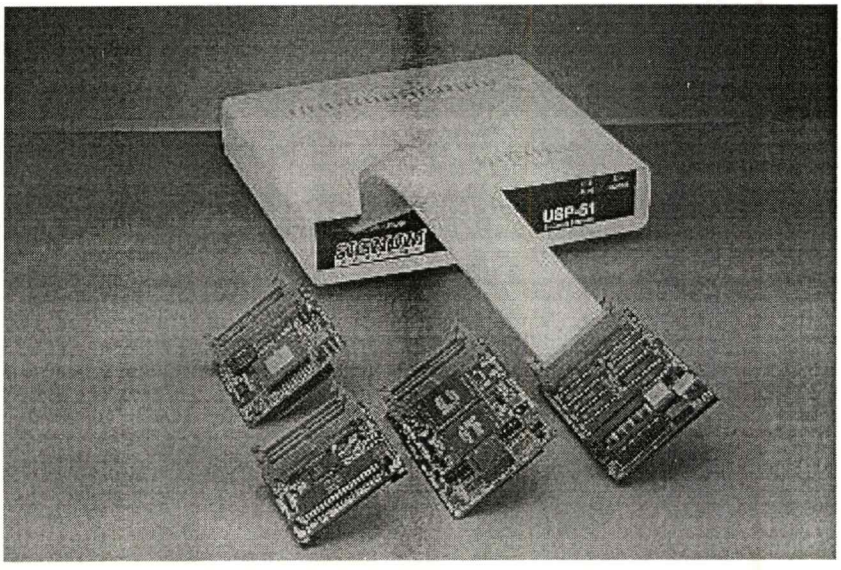


ภาพที่ 1.3 แสดงจอภาพการทำงานของโปรแกรม ChipView-x96 ซิมูเลเตอร์ของบริษัท ChipTools, Inc.

5. การพัฒนาโดยใช้อินเซอร์กิตอีมูเลเตอร์ (In-circuit Emulator)

อินเซอร์กิตอีมูเลเตอร์ประกอบด้วยซอฟแวร์และฮาร์ดแวร์ที่ทำงานร่วมกันโดยสามารถจำลองการทำงานของไมโครคอนโทรลเลอร์ในสภาวะที่ใกล้เคียงความจริง (Real Time) มากที่สุด ฮาร์ดแวร์ของอินเซอร์กิตอีมูเลเตอร์สามารถเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์ได้ทางพอร์ตอนุกรม และพอร์ตขนาน ปลายด้านหนึ่งของฮาร์ดแวร์จะต่อสายสัญญาณออกมาเป็นไอซีคอนเน็กเตอร์ขนาดเท่ากับไมโครคอนโทรลเลอร์ ในการใช้งานนั้นจะต้องถอดไมโครคอนโทรลเลอร์บนระบบที่กำลังพัฒนาออก และนำไอซีคอนเน็กเตอร์ดังกล่าวเสียบเข้าไปแทน ซึ่ง

อินเซอร์กิตอีมูเลเตอร์จะประพฤติตัวเสมือนกับเป็นไมโครคอนโทรลเลอร์จริงๆ และกำเนิดสัญญาณที่เหมือนกับสัญญาณของไมโครคอนโทรลเลอร์ในแต่ละขาสัญญาณ ข้อดีของการพัฒนาด้วยอินเซอร์กิต อีมูเลเตอร์คือ สามารถควบคุมการทำงาน และตรวจสอบการทำงานของระบบที่กำลังพัฒนาได้โดยง่าย เช่นการควบคุมให้ทำงานทีละคำสั่ง, การกำหนดจุดที่ต้องการให้หยุดทำงาน, การตรวจสอบหรือแก้ไขค่าของรีจิสเตอร์ และข้อมูลในหน่วยความจำเป็นต้น ข้อเสียของการพัฒนาด้วยวิธีนี้คือ มีราคาสูงมากจึงไม่เหมาะกับการพัฒนาระบบขนาดเล็กๆ และนักพัฒนาที่ไม่ได้ยึดถือเป็นอาชีพ



ภาพที่ 1.4 แสดงอินเซอร์กิตอีมูเลเตอร์ USP-51 ของบริษัท Signum Systems

1.2 วัตถุประสงค์ของวิทยานิพนธ์

1. สร้างอินเซอร์กิตอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ตระกูล MCS-96 เนื่องจากอินเซอร์กิตอีมูเลเตอร์เป็นเครื่องมือที่มีประสิทธิภาพสูงในการพัฒนาโครงงาน เมื่อเปรียบเทียบกับเครื่องมืออื่นๆ และในปัจจุบันนี้เครื่องมือที่ใช้ในการศึกษา และพัฒนาเกี่ยวกับไมโครคอนโทรลเลอร์ตระกูลนี้ยังมีไม่มากนัก

2. นำเอาหลักการใช้อุปกรณ์ร่วมกันมาประยุกต์สร้างอินเซอร์กิตอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ตระกูล MCS-96 เพื่อให้ราคาในการผลิตต่ำกว่าอินเซอร์กิตอีมูเลเตอร์ที่สั่งซื้อเข้ามาจากต่างประเทศ แต่สามารถใช้ในการพัฒนาโปรแกรมได้อย่างมีประสิทธิภาพ

1.3 ประโยชน์ที่ได้รับจากวิทยานิพนธ์

1. มีเครื่องมือที่ใช้สำหรับศึกษา และพัฒนาเกี่ยวกับไมโครคอนโทรลเลอร์ตระกูล MCS-96 รองรับการใช้งานในอนาคต เมื่อมีนักวิจัย นักศึกษาและผู้ที่สนใจจะศึกษาไมโครคอนโทรลเลอร์ขนาด 16 บิต และส่งเสริมความก้าวหน้าในการศึกษา ตลอดจนพัฒนาเทคโนโลยีไมโครคอนโทรลเลอร์ของนักวิจัย นักศึกษาและผู้ที่สนใจให้รวดเร็วขึ้น เนื่องจากมีเครื่องมือพร้อมให้การสนับสนุนการวิจัย ค้นคว้าและการนำไปประยุกต์ใช้งาน

2. ลดการสูญเสียเงินตราต่างประเทศจากการนำเข้าอินเซอร์กิตอีมูเลเตอร์ อีกทั้งยังเป็นการยกระดับเทคโนโลยีในการสร้างอินเซอร์กิตอีมูเลเตอร์ขึ้นใช้ภายในประเทศให้สูงขึ้น

1.4 ขอบเขตของวิทยานิพนธ์

ทำการพัฒนาซอฟแวร์และสร้างฮาร์ดแวร์ของอินเซอร์กิตอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ตระกูล MCS-96 โดยซอฟแวร์ที่พัฒนาขึ้นนี้ทำงานบนระบบปฏิบัติการวินโดว์ซึ่งมีการติอต่อกับผู้ใช้ในรูปของกราฟฟิก (Graphical User Interface, GUI) จึงทำให้เป็นซอฟแวร์ที่ใช้งานได้ง่าย และมีประสิทธิภาพ สำหรับฮาร์ดแวร์ที่พัฒนาขึ้นนั้นมีขอบเขตความสามารถดังต่อไปนี้

- ใช้เป็นอินเซอร์กิตอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ตระกูล MCS-96 ที่มีตัวถัง (Package) เป็นแบบ DIP-48 เบอร์ 8098

- มีหน่วยความจำ RAM ที่สามารถรองรับโปรแกรมยูสเซอร์ได้สูงถึง 58 กิโลไบต์ โดยแบ่งพื้นที่หน่วยความจำออกเป็นช่วงๆ ช่วงละ 2 กิโลไบต์ ผู้ใช้สามารถกำหนดแอดเดรสที่ต้องการใช้งานได้โดยง่าย

- มีวงจรควบคุมให้ชิพไมโครคอนโทรลเลอร์ เริ่มต้นทำงานตามโปรแกรมมอนิเตอร์ที่แอดเดรส 0800H ~ 0FFFH โดยอัตโนมัติทุกครั้งที่มีการป้อนแหล่งจ่ายไฟให้กับอีมูเลเตอร์ ทำให้ผู้ใช้สามารถพัฒนาโปรแกรมยูสเซอร์ ณ แอดเดรสเริ่มต้นที่แท้จริงคือ 2080H ได้ เมื่อเสร็จขั้นตอนการพัฒนาได้โปรแกรมยูสเซอร์ที่ถูกต้องสมบูรณ์แล้ว ผู้ใช้สามารถนำโปรแกรมยูสเซอร์นั้นไปบันทึกลงอีพีรอมเพื่อใช้งานได้ทันที โดยไม่จำเป็นต้องแก้ไขแอดเดรสใดๆ ทั้งสิ้น

- ในการใช้งานอินเซอร์กิตอีมูเลเตอร์ จะต้องถอดชิพไมโครคอนโทรลเลอร์ของบอร์ดเป้าหมาย (Target Board) ออกก่อน จากนั้นจึงต่อพ็อด (POD) ของอินเซอร์กิตอีมูเลเตอร์เข้าไปแทนที่ และนำชิพไมโครคอนโทรลเลอร์ติดตั้งลงบนพ็อดนั้นอีกที วิธีการนี้ทำให้ชิพไมโครคอนโทรลเลอร์อยู่ในตำแหน่งที่ใกล้กับบอร์ดเป้าหมายมากที่สุด ช่วยลดปัญหาต่างๆ ที่เกี่ยวข้องกับการเชื่อมต่อระหว่างอินเซอร์กิตอีมูเลเตอร์และบอร์ดเป้าหมายได้ เช่น ปัญหาการสั่นกระเพื่อมของสัญญาณในสายแพที่อาจทำให้ระดับลอจิกผิดเพี้ยนไป เป็นต้น

- อินเซอร์กิตอีมูเลเตอร์ที่สร้างขึ้น ใช้งานฮาร์ดแวร์บางส่วนร่วมกับบอร์ดเป้าหมาย เช่น วงจรกำเนิดสัญญาณนาฬิกา, แหล่งจ่ายไฟ และที่สำคัญคือวงจรเชื่อมต่อผ่านพอร์ตอนุกรม RS-232

บทที่ 2

ไมโครคอนโทรลเลอร์ตระกูล MCS-96

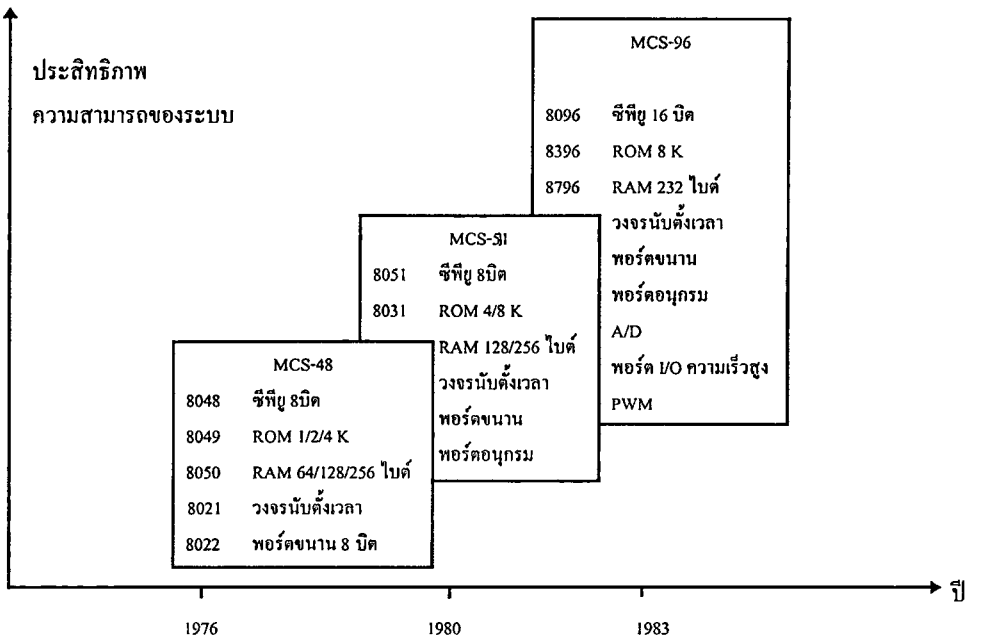
หลังจากที่อินเทลประสบผลสำเร็จในการสร้างไมโครโปรเซสเซอร์ 8080 ในปี 2516 แล้ว อินเทลก็พัฒนาต่อมาเป็น 8085 ในขณะนั้นโครงสร้างบัสของ 8085 เปลี่ยนไปในลักษณะที่เป็นแอดเดรสและข้อมูลมัลติเพล็กซ์กัน อินเทลประสบผลสำเร็จในการขายเป็นอย่างดีเพราะมีชิพประกอบร่วมอยู่ด้วยมาก อินเทลพัฒนาชิพสนับสนุนด้วยโครงสร้างที่ต่อร่วมกับตระกูล 8085 ไว้เป็นจำนวนมาก อย่างไรก็ตามโครงสร้างการพัฒนา 8085 เป็นโครงสร้างที่ต้องประกอบการทำงานร่วมกับชิพอีกหลายตัวโดยเฉพาะต้องมีรอม (ROM), แรม (RAM) ภายนอก และยังต้องมีพอร์ตอินพุตเอาต์พุทภายนอก ดังนั้นไมโครโปรเซสเซอร์กลุ่มนี้จึงไม่เหมาะที่จะใช้ในงานควบคุมขนาดเล็กหรืออุปกรณ์ฮาร์ดแวร์ที่ประยุกต์ใช้ในเครื่องมือต่าง ๆ

อินเทลจึงหันมาสร้างชิพไมโครคอมพิวเตอร์ โดยรวมซีพียู หน่วยความจำ และพอร์ตอินพุตเอาต์พุทไว้ในชิพเดียว และตั้งหมายเลขตระกูลเป็น MCS-48 เบอร์หลักที่กำหนดคือ 8048 เป็นชิพไมโครคอนโทรลเลอร์ที่อินเทลตั้งใจจะให้มีการประยุกต์ใช้ในวงจรที่ต้องการให้ต้นทุนต่ำ มีปริมาณการผลิตเป็นจำนวนมากเช่น สินค้าเครื่องใช้ไฟฟ้าภายในบ้านทั่วไป

ไมโครคอนโทรลเลอร์ตระกูล 8048 ได้นำไปประยุกต์ใช้งานด้านต่าง ๆ มากมาย ต่อมาอินเทลได้พัฒนาเทคโนโลยีให้ก้าวหน้าขึ้น และได้พัฒนากลุ่มชิพใหม่โดยใช้ชื่อ MCS-51 ชิพหลักในตระกูลนี้ได้แก่ 8051 ซึ่งเป็นซีพียูขนาด 8 บิต

MCS-51 มีรอมภายในชิพ มีหน่วยความจำแรมขนาด 256 ไบต์ มีวงจรกำเนิดสัญญาณนาฬิกาและวงจรนับ มีพอร์ตขนานและพอร์ตอนุกรม ทำให้ต่อพอร์ตออกไปใช้ได้ทันที และยังมีระบบซอฟแวร์ที่สนับสนุนให้สามารถทำงานได้เป็นอย่างดี

ครั้นเมื่อเทคโนโลยีทางด้านการผลิตชิพก้าวหน้าขึ้นไปอีก ทำให้สามารถผลิตชิพที่มีทรานซิสเตอร์ภายในได้เป็นแสนตัว อินเทลจึงพัฒนา MCS-96 ชิพหลักคือ 8096 ซึ่งเป็นซีพียูขนาด 16 บิต มีหน่วยความจำสำหรับเก็บโปรแกรม 8 กิโลไบต์ ส่วนเก็บข้อมูล 232 ไบต์ มีวงจรแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอล (A/D) จึงทำให้มีอินพุตเอาท์พุตเป็นทั้งแบบอะนาลอกและแบบดิจิตอล พัฒนาการของเทคโนโลยีดังกล่าวแสดงได้ดังภาพที่ 2.1



ภาพที่ 2.1 แสดงพัฒนาการเทคโนโลยีไมโครคอนโทรลเลอร์ของอินเทล

บริษัทอินเทลผลิตชิพในกลุ่มไมโครโปรเซสเซอร์ และไมโครคอนโทรลเลอร์มาแล้วหลายกลุ่ม พัฒนาการของชิพขึ้นอยู่กับเทคโนโลยี อินเทลได้พัฒนาให้ก้าวหน้า และดีขึ้นเป็นลำดับ จากเดิมที่พัฒนา MCS-48 ปรับปรุงมาเป็น MCS-51 และในที่สุดก็พัฒนามาเป็น MCS-96

MCS-48 พัฒนามาด้วยเทคโนโลยีขนาด 8 บิต มีช่วงเวลาเฉลี่ยของการประมวลผลต่อคำสั่ง 2 ไมโครวินาที การอ้างอิงแอดเดรสของหน่วยความจำใช้ 12 บิต จึงอ้างอิงได้ทั้งหมด 4 กิโลไบต์

เมื่อปรับปรุงมาเป็น MCS-51 ขอบเขตของการออกแบบก็ยังคงใช้โครงสร้างเดิมคือ ใช้โครงสร้างซีพียูขนาด 8 บิต ช่วงเวลาเฉลี่ยของการประมวลผลได้รับการปรับปรุงให้ดีขึ้น มีความเร็วประมาณ 1 ไมโครวินาทีต่อคำสั่ง มีการคูณและหารข้อมูล มีพอร์ตอนุกรม และสามารถอ้างอิงแอดเดรสในหน่วยความจำได้ 64 กิโลไบต์

สำหรับ MCS-96ได้รับการพัฒนามาให้เป็นซีพียูขนาด 16 บิตเต็ม ทำงานที่คำสั่งสลับซับซ้อนได้มากขึ้น ช่วงเวลาเฉลี่ยของการประมวลผลต่อคำสั่งคือ 1.25 ไมโครวินาที แต่ขนาดของข้อมูลเพิ่มเป็น 2 ไบต์ มีส่วนของวงจรฮาร์ดแวร์การคูณและการหาร มีพอร์ตอนุกรม มีวงจร A/D และสามารถอ้างอิงแอดเดรสได้ 64 กิโลไบต์

2.1 จุดเด่นของ MCS-96

เมื่อเทคโนโลยีการผลิตชิพทำได้ดีขึ้นโครงสร้างการออกแบบ MCS-96 จึงรวมจุดเด่นการใช้งานหลาย ๆ อย่างไว้ด้วยกัน เช่น ขยายโครงสร้างระบบนับและรีจิสเตอร์ที่ใช้ในการคำนวณให้เป็นแบบ 16 บิต เพื่อประสิทธิภาพการทำงานโดยรวมที่ดีขึ้น MCS-96 มีหน่วยความจำ

โปรแกรมแบบรอมถึง 8 กิโลไบต์ซึ่งก็เท่ากับเป็นการขยายพื้นที่โปรแกรมทำให้ใช้กับโปรแกรมที่สลับซับซ้อนและยุ่งยากขึ้น มีรีจิสเตอร์ไฟล์ซึ่งเป็นแรมขนาด 232 ไบต์ มีวงจรฮาร์ดแวร์การคูณและหาร ทำให้คำนวณได้เร็ว โดยสามารถคูณเลขขนาด 16 บิต กับ 16 บิตได้ 1 ล้านครั้งภายในเวลาเพียง 6.5 ไมโครวินาที (เมื่อให้ซีพียูทำงานที่ความถี่สัญญาณนาฬิกา 12 เมกะเฮิรตซ์)

ระบบการอ้างแอดเดรสทำได้ 6 โหมดเพื่อให้อ้างอิงกับหน่วยความจำได้อย่างมีประสิทธิภาพ สำหรับการติดต่อกับอินพุทเอาท์พุต มีพอร์ตที่มีระดับทีทีแอล (TTL) จำนวนถึง 40 พอร์ต นอกจากนี้ยังมีพอร์ตที่เป็นแบบพอร์ตความเร็วสูง เพื่อใช้ในการวัดหรือสร้างพัลส์ที่มีความละเอียดได้ถึง 2 ไมโครวินาที ที่ความถี่ 12 เมกะเฮิรตซ์

MCS-96 ประกอบด้วยพอร์ตอนุกรมแบบฟูลดูเพล็กซ์ที่มีการรับส่งข้อมูลแบบอะซิงโครนัส การต่อเชื่อมแบบอะซิงโครนัสนี้ทำได้โดยตรงโดยใช้ซอฟแวร์ควบคุมทำให้สามารถติดต่อระบบอื่นได้เป็นอย่างดี

ระบบการอินเตอร์รัพต์ของ MCS-96 ได้รับการปรับปรุงให้มีประสิทธิภาพที่สูงขึ้น สามารถควบคุมอินเตอร์รัพต์ในลักษณะการจัดลำดับความสำคัญได้ถึง 8 ระดับส่งผลให้การเขียนโปรแกรมได้ง่ายสามารถแยกการควบคุมเหตุการณ์ได้ถึง 8 ระดับ

นอกจากนี้ MCS-96 ยังประกอบด้วยวงจรแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอลขนาด 10 บิต ทำให้ MCS-96 สามารถรับข้อมูลอะนาลอกได้โดยตรง และมีส่วนของเอาท์พุตแบบ PWM ที่สามารถควบคุมให้ความกว้างของพัลส์แปรค่าตามปริมาณที่กำหนดได้

เพื่อเพิ่มความเชื่อถือได้ของระบบในการทำงาน MCS-96 จึงมีวงจรวอตช์ด็อกเพื่อควบคุมการทำงาน เมื่อระบบเกิดการผิดพลาดวงจรวอตช์ด็อกจะเตือน และกระตุ้นให้ระบบทำงานอย่างอัตโนมัติ

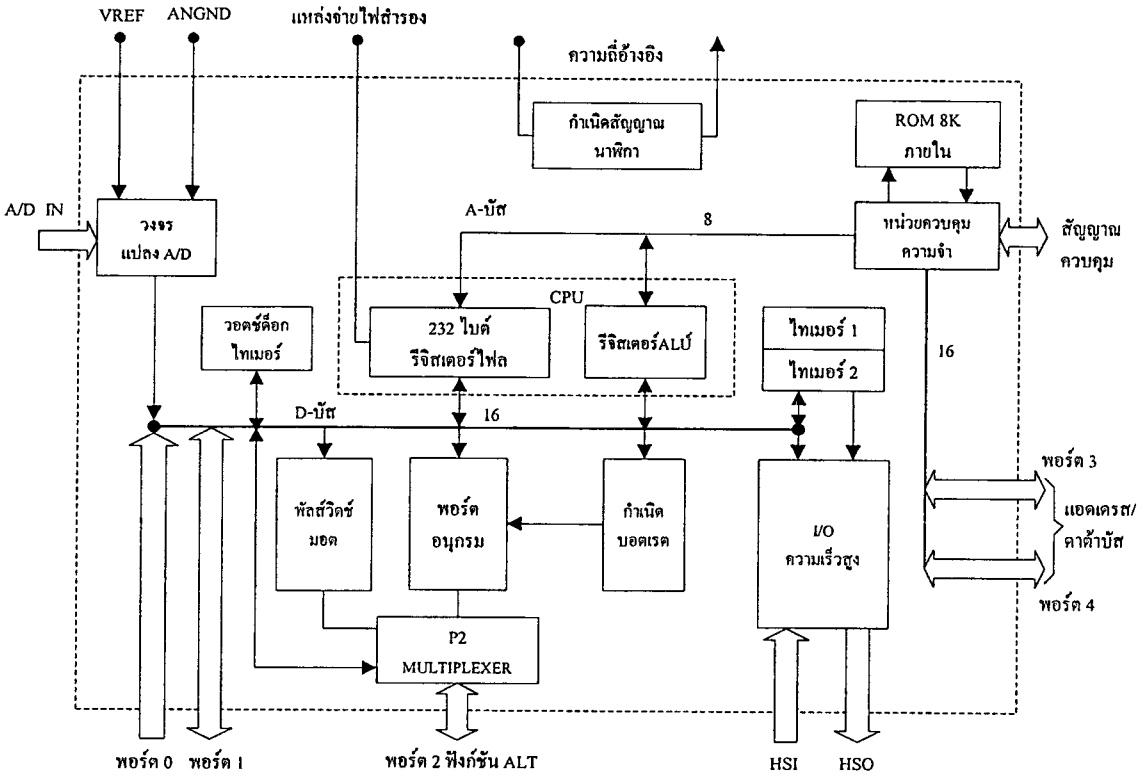
2.2 สถาปัตยกรรมของ 8096 [4], [5], [6]

ผู้ที่คุ้นเคยกับ 8048 มาแล้วจะพบว่า 8096 ขยายโครงสร้างของ 8048 เดิมออกมา โดยให้โครงสร้างบัสภายในเป็นแบบ 16 บิต ลักษณะการเชื่อมต่อภายนอกก็ใช้บัสแบบ 16 บิต ทำให้การขยายระบบทำได้มากขึ้น และใช้งานได้ดีกว่าเดิมมาก ประสิทธิภาพเชิงความเร็วของการทำงานโดยรวมจะสูงกว่า 8048 มาก เพราะ 8096 ใช้สัญญาณนาฬิกาได้สูงถึง 12 เมกะเฮิรตซ์ ประจวบกับชุดคำสั่งที่ประมวลผลใช้ได้กับข้อมูลที่มีขนาด 16 บิต จึงทำให้ประสิทธิภาพการคำนวณทำได้เร็วกว่า 8 บิต ภายในชิพของ 8096 แบ่งออกเป็นส่วน ๆ ตามฟังก์ชั่นการทำงานได้ดังนี้

- ส่วนซีพียู เป็นส่วนหลักของการประมวลผล
- หน่วยความจำ เป็นที่เก็บของโปรแกรมซึ่ง 8096 มีหน่วยความจำแบบรอมภายในชิพ
- หน่วยอินพุตเอาท์พุตความเร็วสูงที่โปรแกรมได้ เป็นส่วนของอินพุตเอาท์พุตพิเศษที่รับข้อมูลหรือส่งข้อมูลในลักษณะพัลส์เป็นสัญญาณนาฬิกาความเร็วสูง

- วงจร A/D เป็นส่วนวงจรที่ทำหน้าที่เปลี่ยนสัญญาณอะนาลอกให้เป็นสัญญาณดิจิตอล
- พอร์ตอนุกรม เป็นพอร์ตรับส่งข้อมูลแบบอะซิงโครนัส, ซิงโครนัส
- วงจร PWM ทำหน้าที่สร้างสัญญาณมอดูเลชันทางความกว้างของพัลส์

สถาปัตยกรรมของ 8096 นั้น รีจิสเตอร์ไฟล์, หน่วยคำนวณ และหน่วยควบคุมถือเป็นหัวใจหลักของซีพียู การส่งผ่านข้อมูลซึ่งกันและกันระหว่างหน่วยเหล่านี้ มีทั้งที่เป็นแบบ 8 บิตบัส และ 16 บิตบัส โครงสร้างสถาปัตยกรรมของ 8096 แสดงได้ดังภาพที่ 2.2



ภาพที่ 2.2 แสดงโครงสร้างของชิพ 8096

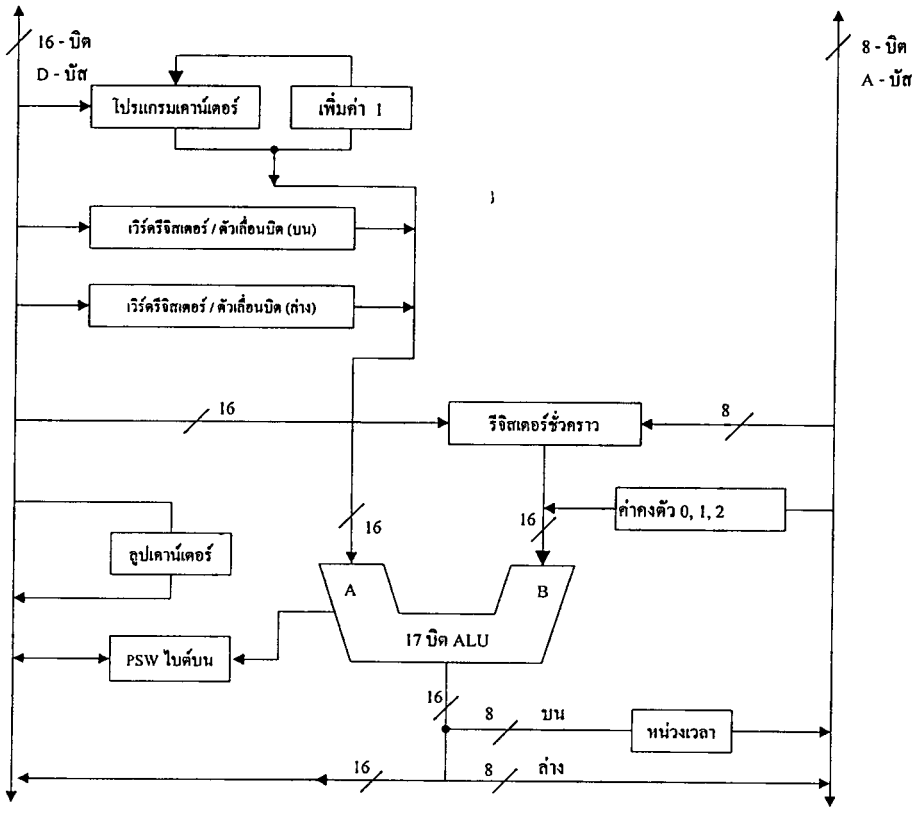
ส่วนประกอบที่สำคัญของซีพียู 8096 คือ รีจิสเตอร์ไฟล์ ซึ่งทำหน้าที่เสมือนรีจิสเตอร์ทั่วไป และหน่วยความจำภายในสำหรับเก็บข้อมูล, RALU (Register/Arithmetic Logic Unit) เป็นหน่วยคำนวณทางคณิตศาสตร์และตรรกศาสตร์, รีจิสเตอร์ SFR (Special Function Register) ซีพียูติดต่อกับอุปกรณ์ภายนอกโดยผ่านทางรีจิสเตอร์นี้ ส่งผลให้การควบคุมอินพุตเอาท์พุตสามารถทำได้อย่างมีประสิทธิภาพและทำงานได้รวดเร็ว นอกจากนี้ยังประกอบด้วยตัวควบคุมหน่วยความจำ (Memory Controller) อีกด้วย ภายใน RALU ไม่มีแอกคิวมูเลเตอร์แต่ทำงานร่วมกับรีจิสเตอร์ต่าง ๆ ในรีจิสเตอร์ไฟล์ได้โดยตรง นอกจากนี้ RALU ยังทำงานโดยตรงกับ SFR ด้วยเช่นกัน จำนวนรีจีสเตอร์ในรีจีสเตอร์ไฟล์ และ SFR มีทั้งหมด 256 ไบต์

โครงสร้างหลักของระบบบัสอยู่ที่การต่อเชื่อมระหว่างรีจิสเตอร์ RALU ที่ต่อกับบัส 2 บัส บัสทั้งสองมีชื่อ A-บัส และ D-บัส A-บัสเป็นบัสขนาด 8 บิต ส่วน D-บัส เป็นบัสขนาด 16 บิต D-บัส มีไว้สำหรับส่งถ่ายข้อมูลระหว่าง RALU กับรีจิสเตอร์ไฟล์ หรือ SFR ส่วน A-บัส ใช้เป็นแอดเดรสบัสสำหรับส่งถ่ายข้อมูลระหว่าง RALU กับรีจิสเตอร์ไฟล์หรือ SFR หรือเป็นบัสที่มีสัญญาณมัลติเพล็กซ์ระหว่างแอดเดรสและข้อมูลที่ติดต่อกับตัวควบคุมหน่วยความจำ ตัวควบคุมหน่วยความจำทำหน้าที่ติดต่อกับรอมภายในหรือหน่วยความจำโปรแกรมภายนอก ภายในตัวควบคุมหน่วยความจำจะมีโปรแกรมเคาน์เตอร์เฉพาะที่เรียกว่า สเลฟพีซี (Slave PC) ซึ่งมีค่าตรงกับค่า PC ของซีพียู ในการเฟตช์ข้อมูล จากหน่วยความจำโปรแกรมก็ใช้สเลฟพีซีตัวนี้เป็นตัวชี้ ซึ่งทำให้ประหยัดเวลาของซีพียู เพราะไม่ต้องส่งค่าแอดเดรสไปให้ตัวควบคุมหน่วยความจำแต่ถ้ามีการทำคำสั่ง JUMP จะต้องมีการโหลดค่ามายังสเลฟพีซีด้วยค่าใหม่เสียก่อน

ส่วนของรีจิสเตอร์ที่อยู่ในรีจิสเตอร์ไฟล์มีขนาด 232 ไบต์ ผู้ใช้งานสามารถเรียกเข้าถึงข้อมูลเป็นไบต์ เป็นเวิร์ด หรือเวิร์ดคู่ได้ RALU สามารถใช้รีจิสเตอร์ไฟล์ได้ทุกไบต์จึงคล้ายกับว่ามีแอกคิวมูเลเตอร์ 232 ตัว เวิร์ดแรกของกลุ่มรีจิสเตอร์ทั้งหมดใช้เป็นสแต็กพอยน์เตอร์ ดังนั้นเมื่อมีการกระทำเกี่ยวข้องกับสแต็กเราจะใช้เวิร์ดนี้ไม่ได้ แอดเดรสของรีจิสเตอร์ และ SFR จะได้รับการเก็บไว้ที่รีจิสเตอร์ชั่วคราวที่ชื่อว่า ADDRESS A และ ADDRESS B

ในการทำงานของซีพียูเมื่อเฟตช์คำสั่ง รหัสคำสั่งจะผ่านเข้ามาทาง A-บัส และจะเก็บไว้ที่ IR (Instruction Register) เพื่อส่งต่อให้หน่วยควบคุม (Control Unit) ถอดรหัสคำสั่งและสร้างลำดับของสัญญาณเพื่อให้ RALU ทำงานตามคำสั่งที่รับมา

การคำนวณส่วนใหญ่ของ 8096 จะทำใน RALU ลักษณะของ RALU แสดงได้ดังภาพที่ 2.3 RALU ประกอบด้วย ALU ขนาด 17 บิต, โปรแกรมสเตตัสเวิร์ด (PSW), โปรแกรมเคาน์เตอร์ (PC), ลูปเคาน์เตอร์ และรีจิสเตอร์ชั่วคราว รีจิสเตอร์ทั้งหมดจะมีขนาด 16 บิต หรือ 17 บิต (16 บิต + บิตเครื่องหมาย) รีจิสเตอร์บางตัวยังสามารถทำงานบางอย่างได้โดยไม่ต้องโหลดข้อมูลเข้าใน ALU เลย เช่น การเพิ่มค่าให้กับรีจิสเตอร์ PC แต่เมื่อมีการกระโดดด้วยคำสั่ง JUMP PC จะเปลี่ยนค่าไปจากปกติที่เคยเพิ่มขึ้นทีละหนึ่ง



ภาพที่ 2.3 แสดงบล็อกไดอะแกรมของ RALU

ส่วนของเวิร์ดรีจิสเตอร์สำหรับเลื่อนบิต (บนและล่าง) ใช้สำหรับการทำงานตามคำสั่ง Shift, รวมถึงกลไกการคูณและหารที่ต้องมีการเลื่อนบิตด้วย เวิร์ดรีจิสเตอร์สำหรับเลื่อนบิตส่วนล่าง (Lower) ใช้เฉพาะกรณีการเลื่อนบิตแบบเวิร์ดคู่ ส่วนเวิร์ดรีจิสเตอร์สำหรับเลื่อนบิตส่วนบน (Upper) ใช้สำหรับทำตามคำสั่ง Shift หรือใช้เป็นรีจิสเตอร์ชั่วคราวสำหรับคำสั่งอื่นๆ ส่วนลูปเคาน์เตอร์ทำหน้าที่นับรอบเมื่อมีการเลื่อนบิตซ้ำ ๆ กัน

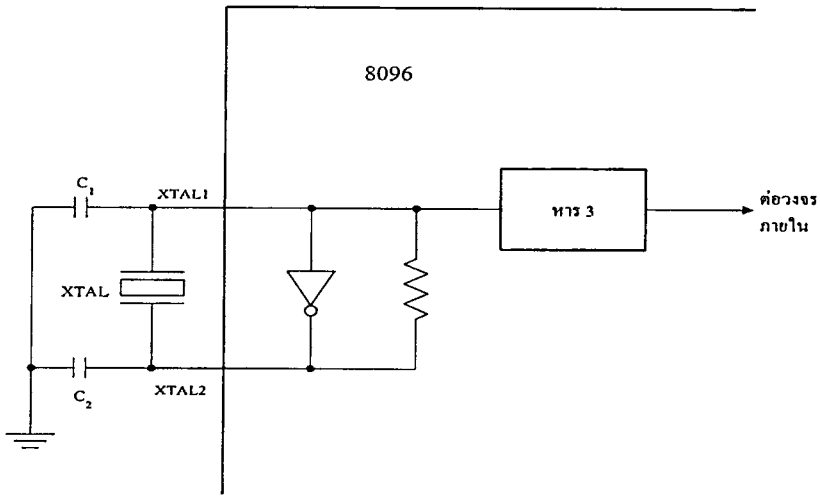
รีจิสเตอร์ชั่วคราวใช้สำหรับเก็บโอเปอร์แรนด์ตัวที่สอง ในกรณีคำสั่งนั้นมีโอเปอร์แรนด์สองตัว และจะทำหน้าที่เป็นตัวคูณ เมื่อใช้ในคำสั่งคูณ และเป็นตัวหารเมื่อใช้ในคำสั่งหาร ส่วนคำสั่งลบนั้นก็จะทำการคอมพลีเมนต์ก่อนแล้วจึงส่งไปยังอินพุต B ของ ALU

ส่วนของการหน่วงเวลาที่แสดงในภาพที่ 2.3 ใช้เพื่อเปลี่ยนจาก 16 บิต ไปเป็น 8 บิต เพราะแอดเดรสและข้อมูลจะได้รับการส่งผ่านทาง A-บัส ขนาด 8 บิต สำหรับวงจรส่วนค่าคงตัวนั้นใช้สำหรับเก็บค่าคงที่ที่ใช้ในการคำนวณเช่น 0, 1 และ 2 ทำให้การคำนวณรวดเร็วขึ้น

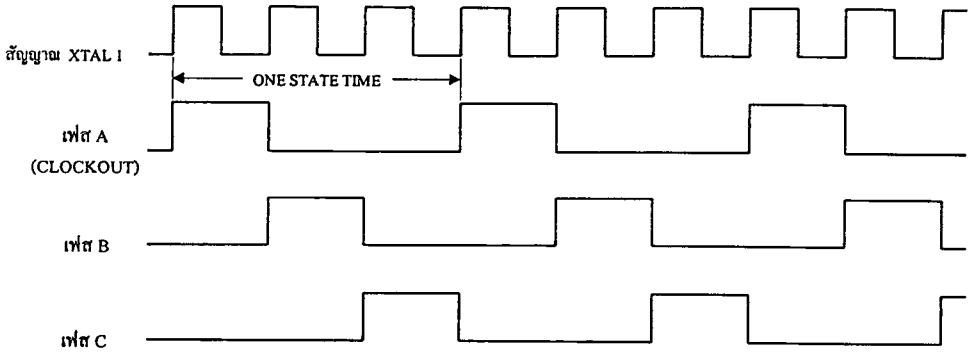
RALU ติดต่อกับหน่วยความจำ (ยกเว้นรีจิสเตอร์ไฟล์ และ SFR) ผ่าน A-บัส โดยใช้ตัวควบคุมหน่วยความจำ และตัวควบคุมหน่วยความจำนี้จะใช้สเลฟพีซีเป็นแอดเดรสเพื่อทำการเฟตช์ข้อมูลโดยไม่ต้องรอให้ RALU สั่ง และสเลฟพีซีจะเพิ่มค่าหลังจากการเฟตช์แต่ละครั้ง ในกรณีที่มี

การ JUMP สเลฟพีซีจะได้รับค่าใหม่โดยรับผ่านมาทาง A-บัส จากนั้นจึงจะทำการเฟตช์ข้อมูลต่อไป ตัวควบคุมหน่วยความจำนี้จะส่งสัญญาณไปเฟตข้อมูลมาทีละ 3 ไบต์ส่งผลให้การเอ็กซีคิวท์ทำได้เร็วขึ้น คำสั่งแต่ละคำสั่งสามารถทำการเอ็กซีคิวท์ได้ทันทีโดยไม่ต้องมีสถานะรอคอย ยกเว้นคำสั่ง JUMP ซึ่งจำเป็นต้องมีการโหลดค่าสเลฟซีพีใหม่ซึ่งใช้เวลาไปประมาณ 4 สเตต

8096 ต้องการสัญญาณนาฬิกาที่ความถี่ระหว่าง 6 เมกะเฮิรตซ์ ถึง 12 เมกะเฮิรตซ์ การประกอบวงจรสร้างสัญญาณนาฬิกาทำได้ด้วยการต่อคริสตอลที่กำเนิดสัญญาณนาฬิกาตามความถี่ที่กำหนดในตัวคริสตอล โดยต่อเข้าที่ขา XTAL1 และ XTAL2 ดังภาพที่ 2.4 ความถี่ของสัญญาณนาฬิกาที่ป้อนให้กับ 8096 จะได้รับการหารภายในให้ความถี่ลดลงสามเท่า พร้อมกับสร้างสัญญาณในรูปที่มีเฟสต่างกัน โดยแบ่งออกเป็น 3 เฟสคือ เฟส A, เฟส B และ เฟส C แต่ละเฟสจะมีความกว้างพัลส์ 33 % ของคาบเวลา เฟส A เป็นสัญญาณส่งออกมาที่ขาต่าง ๆ ของ 8096 ส่วนเฟส B และเฟส C เป็นสัญญาณที่ใช้ภายใน แสดงดังภาพที่ 2.5



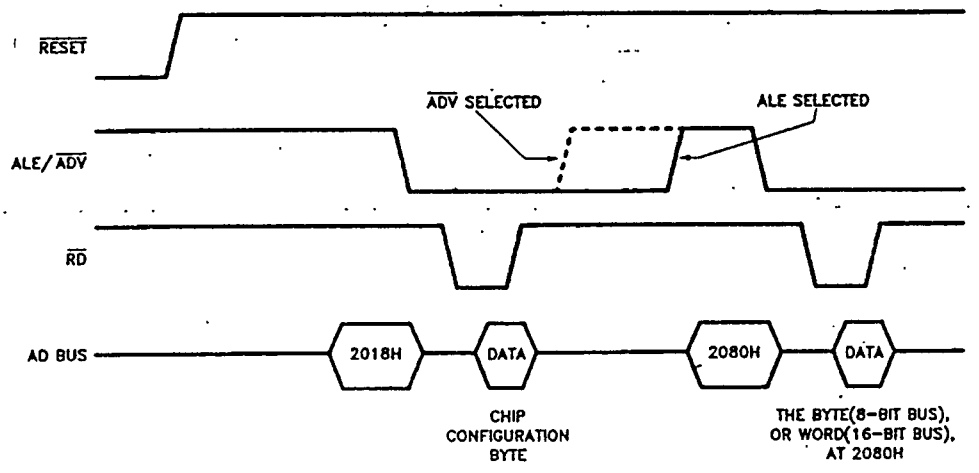
ภาพที่ 2.4 แสดงการต่อคริสตอลเพื่อสร้างความถี่ให้กับชิพ 8096



ภาพที่ 2.5 แสดงสัญญาณนาฬิกาเมื่อเทียบกับ XTAL1

เมื่อเริ่มป้อนไฟเลี้ยง 8096 จะทำการรีเซตตัวเอง การรีเซตจะเป็นไปอย่างสมบูรณ์ต้องใช้เวลาหลังจากที่ระดับของแรงดัน V_{CC} และสัญญาณนาฬิกาอยู่ในสภาวะคงที่แล้วอย่างน้อย 2 สเตต ช่วงเวลาที่ใช้ในการรีเซตที่เหมาะสมของชิพ 8096 จะอยู่ในช่วงประมาณ 10 สเตต

หลังจากสัญญาณรีเซตอยู่ในสถานะ High แล้ว ในช่วง 10 สเตตของการรีเซต ซีพียูจะทำการอ่านค่า Chip Configuration Byte (CCB) ซึ่งอยู่ที่ตำแหน่ง 2018H และเขียนค่าดังกล่าวให้กับ Chip Configuration Register (CCR) ถ้าแรงดันที่ต่ออยู่กับขา EA ถูกเลือกให้อยู่ในโหมดการเอ็กซีคิวท์ข้อมูลภายใน (Internal Execution) ค่า CCB จะถูกอ่านจากรอม/อีพีรอมภายในชิพ แต่ถ้าแรงดันที่ขา EA ถูกเลือกให้อยู่ในโหมดการเอ็กซีคิวท์ข้อมูลภายนอก ค่า CCB จะถูกอ่านจากหน่วยความจำภายนอก



ภาพที่ 2.6 แสดงสภาวะการรีเซต

ตารางที่ 2.1 แสดงการกำหนดค่าของรีจิสเตอร์หน้าที่พิเศษ (SFR) ภายหลังเสร็จสิ้นการรีเซต

SFR	Reset value
Port 1	11111111B
Port 2	110XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receiver)	undefined
Baud Rate Register	undefined
Serial Control Status	undefined
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B

ตารางที่ 2.1 (ต่อ) แสดงการกำหนดค่าของรีจิสเตอร์หน้าที่พิเศษภายหลังเสร็จสิ้นการรีเซต

SFR	Reset value
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	11111111B
HSI Status	undefined
IOS 0	00000000B
IOS 1	00000000B
IOC 0	X0X0X0X0B
IOC 1	X0X0XXX1B
HSI FIFO	empty
HSO CAM	empty
HSO lines	000000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H
RD	high
WR	high
ALE	low
BHE	low
INST	high

2.3 การจัดการหน่วยความจำ

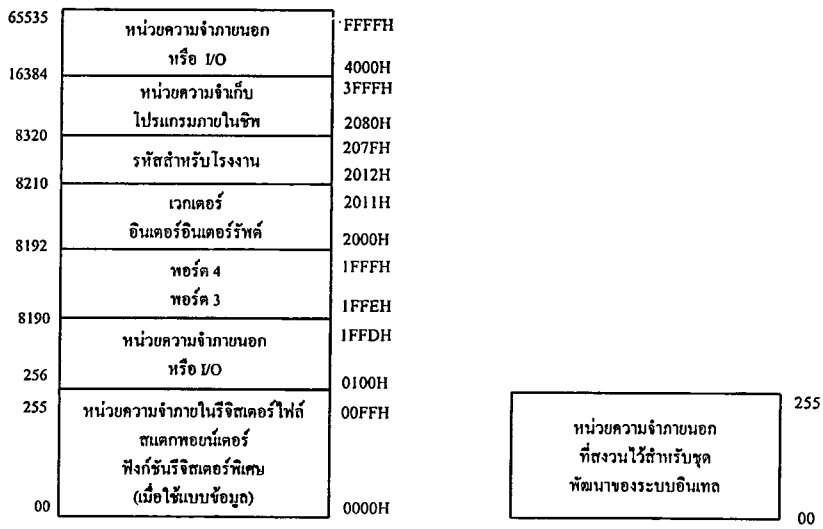
8096 เป็นซีพียูขนาด 16 บิตที่มีประสิทธิภาพในการอ้างแอดเดรสได้ดีกว่า 8048 โดยสามารถอ้างแอดเดรสได้ถึง 64 กิโลไบต์ และแอดเดรสของหน่วยความจำแรมและรอมจะใช้ร่วมกัน ไม่แยกเหมือนกับ 8048 พื้นที่ของหน่วยความจำจาก 0000H ถึง 00FFH เป็นพื้นที่ของหน่วยความจำแรมที่อยู่ภายในชิพ แอดเดรสจาก 0100H ถึง 1FFDH เป็นพื้นที่ของ I/O หรือหน่วยความจำภายนอก สำหรับแอดเดรสที่ 1FFEH และ 1FFFH เป็นหมายเลขพอร์ตของพอร์ต 3 และพอร์ต 4 ตามลำดับ

ตั้งแต่แอดเดรส 2000H ถึงแอดเดรส 2011H เป็นแอดเดรสที่อินเทลวางไว้สำหรับเวกเตอร์การอินเตอร์รัพต์ซึ่งมีอยู่ 8 ระดับ จากแอดเดรส 2012H ถึง 207FH เป็นส่วนที่โรงงานผู้ผลิตใส่รหัสไว้สำหรับการทดสอบ แอดเดรส 2080H ถึง 3FFFH เป็นส่วนของหน่วยความจำรอมสำหรับเก็บโปรแกรมภายในชิพ แอดเดรส 4000H เป็นต้นไป จนถึงแอดเดรส 0FFFFH คือ ส่วนที่ผู้ใช้จะกำหนดให้เป็น I/O หรือ หน่วยความจำที่ต่ออยู่ภายนอกก็ได้ โครงสร้างการจัดหน่วยความจำของ 8096 แสดงดังภาพที่ 2.7

สิ่งที่น่าสังเกตคือ หน่วยความจำ 64 กิโลไบต์ของ 8096 ได้รับการจัดสลับกันทั้ง I/O แรมและรอม พื้นที่หน่วยความจำ 256 ไบต์แรกเป็นส่วนของรีจิสเตอร์ไฟล์จำนวน 232 ไบต์ที่เหลือเป็นของ SFR พื้นที่ส่วนนี้ซีพียูไม่สามารถอ่านรหัสเพื่อทำการเอ็กซีคิวต์ได้ แต่ถ้าจะเอ็กซีคิวต์รหัสจากแอดเดรส 0000H ถึง 00FFH นี้จะต้องเฟตช์มาจากหน่วยความจำภายนอก ซึ่งอินเทลได้สงวนไว้สำหรับการพัฒนาระบบ (การเอ็กซีคิวต์รหัสของหน่วยความจำภายนอกในตำแหน่งนี้จะกระทำเมื่อเกิดการอินเตอร์รัพต์แบบนอนมาสเคเบิล)

หน่วยความจำตำแหน่ง 1FFEH และ 1FFFH ใช้สำหรับเป็นหมายเลขพอร์ต 3 และ 4 โดยเมื่อมีการอ้างที่แอดเดรสนี้จะหมายถึง การติดต่อกับพอร์ตทำให้การสร้างฮาร์ดแวร์เพิ่มเติมจากพอร์ตนี้ทำได้ง่ายขึ้น ส่วนตำแหน่ง 2000 ถึง 2011H เป็นที่เก็บเวกเตอร์ของการอินเตอร์รัพต์ เพื่อระบุตำแหน่งของแอดเดรสปลายทางที่ต้องการให้ซีพียูเริ่มทำงาน

ตำแหน่งที่ 2012H ถึง 207FH จะเก็บรหัสทดสอบของโรงงาน เมื่อมีการรีเซตระบบซีพียูจะเริ่มต้นทำงานที่ตำแหน่ง 2080H



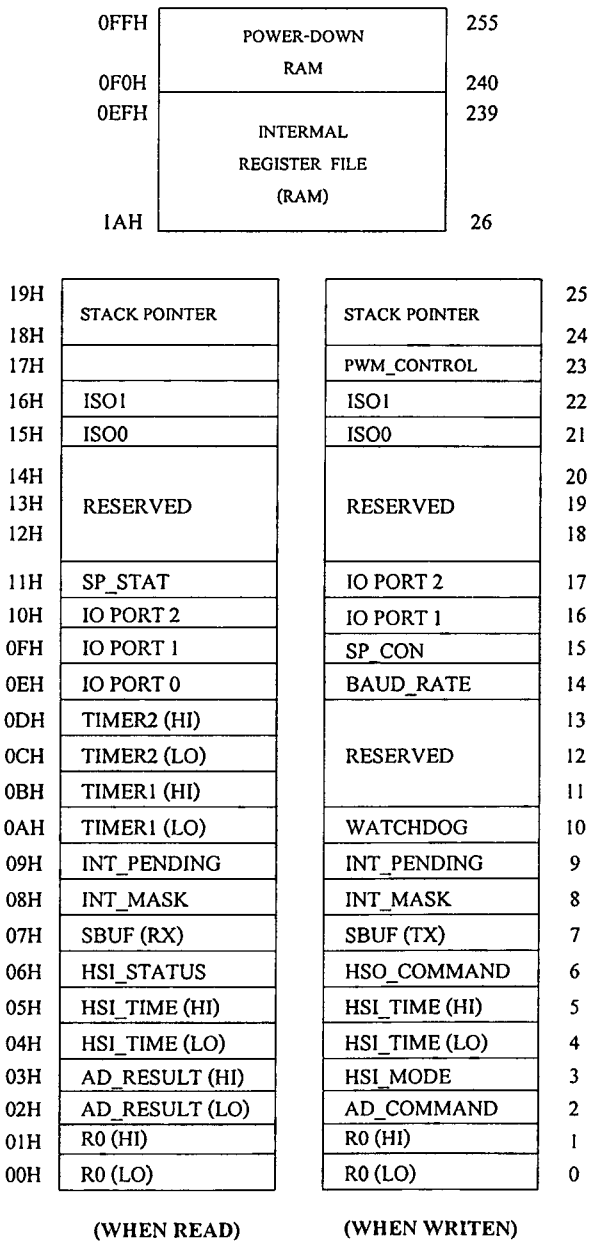
ภาพที่ 2.7 แสดงพื้นที่หน่วยความจำของ 8096

8096 มีแรมภายในอยู่ 256 ไบต์แบ่งออกเป็นสองกลุ่มคือ รีจีสเตอร์ไฟล์ และ SFR (Special Function Register) โดยที่ตำแหน่งแอดเดรส 00H ถึง 17H ใช้เป็น SFR ตำแหน่งที่ 18H, 19H ใช้เป็นสแตกพอยน์เตอร์ ซึ่งผู้ใช้สามารถกำหนดสแตกไว้ที่ตำแหน่งใดก็ได้ภายในหน่วยความจำ 64 กิโลไบต์ เนื่องจากตัวชี้สแตกมีขนาด 16 บิต

หน้าที่ของ SFR คือควบคุม I/O SFR จึงเป็นรีจีสเตอร์ที่จะเขียนหรืออ่านข้อมูลก็ได้ ดังนั้นเมื่อเขียนหรืออ่านจึงมีความหมายที่แตกต่างกันดังแสดงในภาพที่ 2.8 ส่วนของ SFR ที่

อินเทลสงวนไว้ (RESERVE) นั้น อินเทลตั้งใจที่จะใช้สำหรับการขยายโครงสร้างบางอย่างเพิ่มเติมในอนาคตหรือเพื่อการทดสอบในขณะทำการผลิตชิพ

ส่วนที่ทำหน้าที่เป็นรีจีสเตอร์ทั่วไปเริ่มจากแอดเดรส 1AH เป็นต้นไป ส่วนนี้ผู้เขียนโปรแกรมสามารถอ้างอิง และนำไปใช้ในการคำนวณได้



ภาพที่ 2.8 แสดงพื้นที่ของรีจีสเตอร์ไฟล์

พื้นที่แรมในส่วนแอดเดรส 0F0H ถึง 0FFH เรียกว่า Power Down RAM ใช้เก็บข้อมูลที่สำคัญ ที่แม้ว่าไฟฟ้าเลี้ยงวงจรหายไปแล้วก็ยังเก็บข้อมูลอยู่ได้ โดยข้อมูลในแรมส่วนนี้จะได้รับ

แรงดันจากไฟเลี้ยง 2 ชุดคือ จาก V_{CC} และจาก V_{PD} ในสภาวะปกติหน่วยความจำส่วนนี้ได้รับไฟเลี้ยงจาก V_{CC} แต่เมื่อ V_{CC} หายไปขารีเซตจะมีลอจิกเป็น 0 เพื่อควบคุมให้ซีพียูหยุดการทำงาน และเป็นการป้องกันไม่ให้มีการเขียนข้อมูลลงในหน่วยความจำ V_{PD} จะทำการจ่ายไฟเลี้ยงให้กับหน่วยความจำแทน V_{CC} ทันทีทำให้ข้อมูลในส่วนนี้ยังคงอยู่ต่อไปได้ การใช้พลังงานไฟฟ้าในส่วนของ V_{PD} นี้ใช้กระแสเพียง 1 มิลลิแอมป์ ดังนั้นจึงสามารถใช้แบตเตอรี่เล็ก ๆ ใส่ให้กับระบบไว้ได้ครั้นเมื่อมีไฟเลี้ยง V_{CC} กลับมาใหม่ เมื่อ V_{CC} อยู่ในภาวะคงที่ขารีเซตจะมีลอจิกเป็น 1 ซีพียูจะได้รับการรีเซตและทำงานตามปกติต่อไป V_{CC} ก็จะจ่ายไฟเลี้ยงแทน V_{PD} อีกครั้ง

ในขณะอยู่ที่โหมด Power Down ซีพียูของ 8096 ไม่ต้องการใช้สัญญาณนาฬิกา และการเก็บข้อมูลในแรมส่วนแอดเดรส 0F0H ถึง 0FFH นี้มีลักษณะเป็นแบบสแตกติกแรม

หน้าที่ต่าง ๆ ของรีจีสเตอร์ SFR มีดังนี้

R0 เรียกว่า *Zero register* จะเก็บค่า 0 ไว้ ดังนั้นทุกครั้งที่อ่านจะได้ข้อมูล 0 มีไว้สำหรับเป็นค่าคงตัวที่ใช้ในการคำนวณหรือเปรียบเทียบ และใช้เป็นฐาน (base) ในการอ้างแอดเดรสโดยใช้ดรรชนี

AD_RESULT ใช้เก็บผลลัพธ์ของวงจร A/D ประกอบด้วยไบต์สูงและต่ำรวม 2 ไบต์ รีจีสเตอร์ตัวนี้สามารถอ่านได้อย่างเดียว

AD_COMMAND รีจีสเตอร์คำสั่งของ A/D ทำหน้าที่ควบคุมการทำงานของวงจร A/D

HSI_MODE ทำหน้าที่กำหนดโหมดการทำงานของหน่วยอินพุตความเร็วสูง

HSI_TIME เป็นรีจีสเตอร์สำหรับกำหนดเวลาให้อินพุตความเร็วสูง รีจีสเตอร์ตัวนี้สามารถอ่านได้อย่างเดียว

HSO_TIME เป็นรีจีสเตอร์สำหรับกำหนดเวลาให้เอาท์พุตความเร็วสูง ทำงานตามคำสั่งที่อยู่ในรีจีสเตอร์คำสั่ง

HSO_COMMAND เป็นรีจีสเตอร์คำสั่ง ทำหน้าที่กำหนดเงื่อนไขของการโหลดข้อมูลให้กับ HSO_TIME

HSI_STATUS เป็นรีจีสเตอร์แสดงสถานะของอินพุตความเร็วสูง

SBUF (TX) เป็นบัฟเฟอร์สำหรับการส่งข้อมูลออกทางพอร์ตอนุกรม

SBUF (RX) เป็นบัฟเฟอร์สำหรับการรับข้อมูลที่เข้ามาทางพอร์ตอนุกรม

INT_MASK เป็นรีจีสเตอร์มาสก์อินเตอร์รัพต์ เพื่อใช้สำหรับการอีนาเบิล หรือดิสเอเบิลการอินเตอร์รัพต์

INT_PENDING เป็นรีจีสเตอร์แสดงสถานะการอินเตอร์รัพต์ เพื่อให้ซีพียูรับรู้ว่าเกิดการอินเตอร์รัพต์จากภายนอกขึ้นแล้ว

WATCHDOG เป็นรีจีสเตอร์ที่ใช้ในการตรวจสอบการทำงานของระบบ จะได้รับการเขียนข้อมูลเป็นจังหวะตลอด และจะทำการรีเซตระบบถ้าหากว่าไม่ได้รับการเขียนข้อมูล

TIMER1 ใช้แสดงข้อมูลของไทเมอร์ 1 รีจีสเตอร์ตัวนี้สามารถอ่านได้อย่างเดียว

TIMER2 ใช้แสดงข้อมูลของไทเมอร์ 2 รีจีสเตอร์ตัวนี้สามารถอ่านได้อย่างเดียว

IOPORT0 รีจีสเตอร์พอร์ต 0 ใช้สำหรับเขียนหรืออ่านข้อมูลกับพอร์ต 0

BAUD_RATE เป็นรีจีสเตอร์สำหรับกำหนดอัตราความเร็วของการรับและส่งข้อมูลผ่านพอร์ตอนุกรม

IOPORT1 รีจีสเตอร์พอร์ต 1 ใช้สำหรับเขียนหรืออ่านข้อมูลกับพอร์ต 1

IOPORT2 รีจีสเตอร์พอร์ต 2 ใช้สำหรับเขียนหรืออ่านข้อมูลกับพอร์ต 2

SP_STAT รีจีสเตอร์แสดงสถานะของพอร์ตอนุกรม

SP_CON รีจีสเตอร์สำหรับการควบคุมพอร์ตอนุกรม

IOS0 รีจีสเตอร์แสดงสถานะของพอร์ตเอาท์พุตความเร็วสูง

IOS1 รีจีสเตอร์แสดงสถานะของพอร์ตอินพุตความเร็วสูง และไทเมอร์

IOC0 รีจีสเตอร์ใช้สำหรับควบคุมการทำงานของพอร์ตอินพุตความเร็วสูงและไทเมอร์ 2

IOC1 รีจีสเตอร์ใช้สำหรับควบคุมการทำงานของพอร์ต 2, การอินเตอร์รัพต์ของไทเมอร์และพอร์ตอินพุตความเร็วสูง

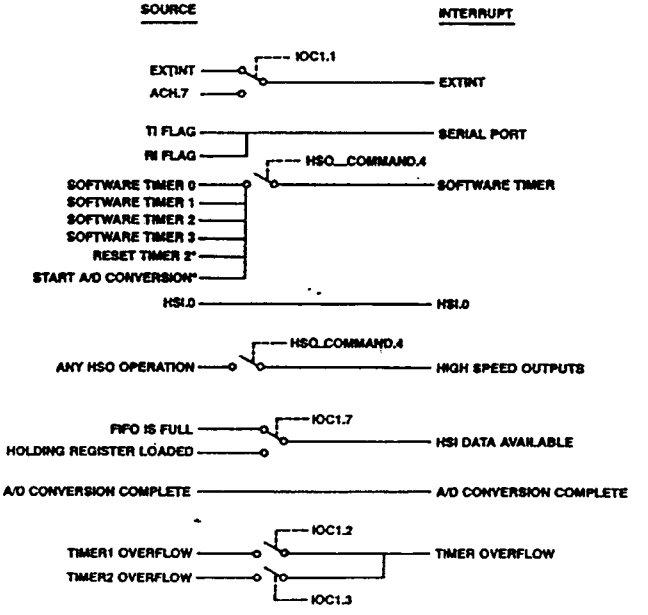
PWM_CONTROL รีจีสเตอร์ควบคุมการมอดูเลตทางความกว้างของพัลส์

2.4 โครงสร้างการอินเตอร์รัพต์

8096 ได้รับการออกแบบมาให้ใช้ในงานควบคุม ดังนั้นจึงมีแหล่งของการอินเตอร์รัพต์หลายทาง การอินเตอร์รัพต์เหล่านี้มีการชี้ไปกระทำแอดเดรสของหน่วยความจำ โดยอาศัยอินเตอร์รัพต์เวกเตอร์ ซึ่งแต่ละเวกเตอร์จะประกอบด้วยข้อมูลสองไบต์ ใช้เป็นแอดเดรสสำหรับระบุให้ซีพียูไปทำงานที่ตำแหน่งใด ลักษณะของอินเตอร์รัพต์จากแหล่งต่าง ๆ กำหนดตามลำดับความสำคัญได้เป็นระดับแสดงได้ดังตารางที่ 2.2 และเมื่อเขียนเป็นไดอะแกรมแหล่งของสัญญาณอินเตอร์รัพต์ที่เกิดขึ้นแสดงได้ดังภาพที่ 2.9

ตารางที่ 2.2 แสดงตำแหน่งและระดับความสำคัญของอินเตอร์รัพต์เวกเตอร์

แหล่งอินเตอร์รัพต์	ตำแหน่งเวกเตอร์		ลำดับความสำคัญ
	ไบต์สูง	ไบต์ต่ำ	
Software Interrupt	2011H	2010H	ยังไม่ประยุกต์ให้ใช้
External Interrupt	200FH	200EH	7 (สูงสุด)
Serial port Interrupt	200DH	200CH	6
Software Timer	200BH	200AH	5
HSI .0 Interrupt	2009H	2008H	4
HSO Interrupt	2007H	2006H	3
HSI data avalable	2005H	2004H	2
A/Dconversion complete	2003H	2002H	1
Timer overflow	2001H	2000H	0 (ต่ำสุด)

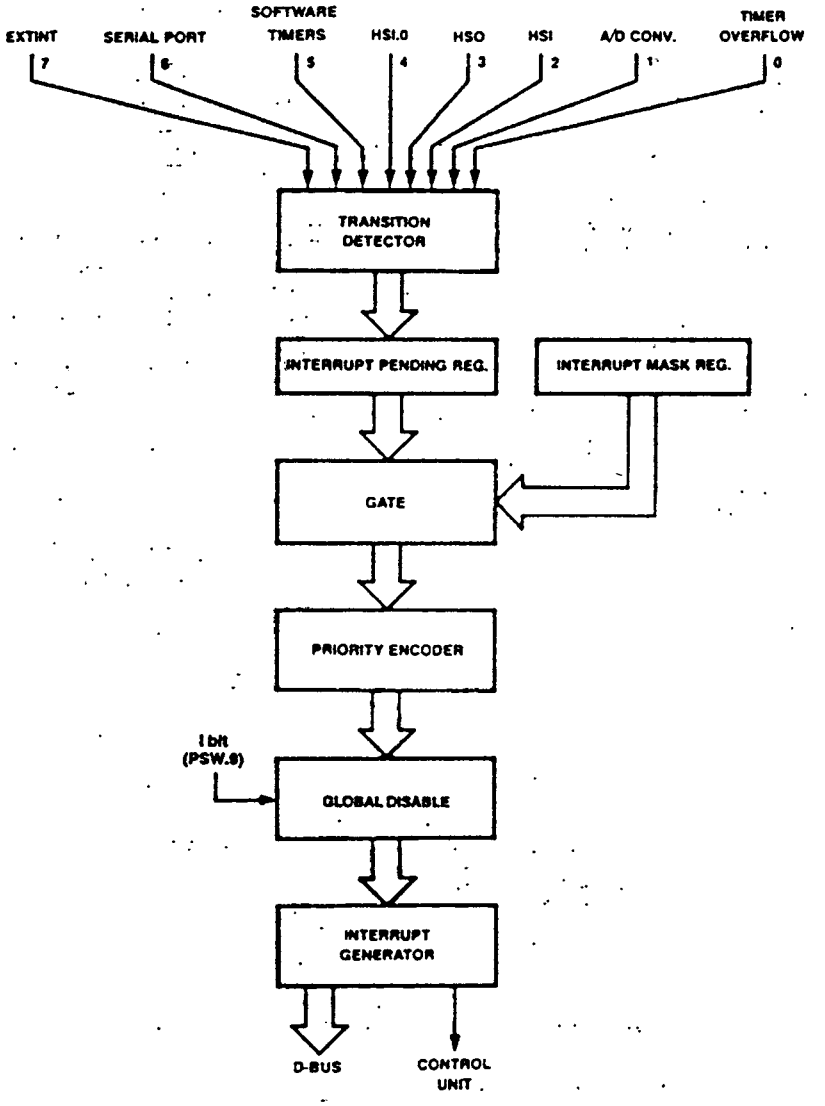


Note: * Only when initiated by the HSO unit.

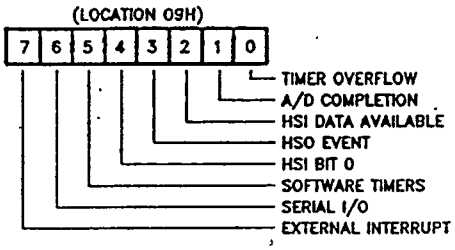
ภาพที่ 2.9 แสดงแหล่งกำเนิดของอินเตอร์รัพต์

จากที่กล่าวแล้วว่าระบบอินเตอร์รัพต์มีการจัดระดับความสำคัญ อินเทลจึงออกแบบโครงสร้างสถาปัตยกรรมของการอินเตอร์รัพต์ที่มีการควบคุมและถอดรหัสตามระดับความสำคัญหรือ แหล่งที่มาของอินเตอร์รัพต์นั้น ๆ โครงสร้างภายในจะทำการทดสอบแหล่งที่อินเตอร์รัพต์ว่าสัญญาณที่บิตนั้นมีการเปลี่ยนแปลงจาก "0" ไปเป็น "1" แล้วหรือยัง ถ้ามีก็จะเซตบิตในรีจิสเตอร์ Interrupt pending เพื่อบอกว่ามีการอินเตอร์รัพต์เข้ามาแล้ว และหากมีการตอบสนองการ

อินเตอร์รัพต์ไบต์ข้อมูลในรีจิสเตอร์นี้ก็จะได้รับการเคลียร์ อย่างไรก็ตามผู้เขียนโปรแกรมสามารถเซตหรือเคลียร์รีจิสเตอร์นี้ด้วยคำสั่งทางซอฟแวร์ได้ การอีนาเบิล และดิสเอเบิลของแต่ละอินเตอร์รัพต์สามารถทำได้โดยการกำหนดค่าในอินเตอร์รัพต์มาสก์รีจิสเตอร์ซึ่งอยู่ที่ตำแหน่งแอดเดรส 0008H ส่วนการกำหนดการอีนาเบิล และดิสเอเบิลของอินเตอร์รัพต์ทั้งหมดนั้นสามารถทำได้โดยการเซตหรือเคลียร์บิตที่ 9 ในรีจิสเตอร์ PSW ด้วยคำสั่ง EI และ DI บล็อกไดอะแกรมระบบอินเตอร์รัพต์ของ 8096 แสดงในภาพที่ 2.10



ภาพที่ 2.10 แสดงบล็อกไดอะแกรมระบบอินเตอร์รัพต์ของ 8096



ภาพที่ 2.11 แสดงรีจิสเตอร์ Interrupt Pending

การจัดลำดับความสำคัญของการอินเตอร์รัพต์จะทำการถอดรหัสจากรีจิสเตอร์ Interrupt pending และดูว่าอยู่ในสภาวะอีนาเบิลหรือไม่ ถ้าอีนาเบิลก็จะจัดระดับลำดับความสำคัญจาก 0 ถึง 7 ตามแหล่งที่เกิด เมื่อเกิดอินเตอร์รัพต์ 8096 จะกระโดดไปทำงานที่จุดเริ่มต้นของโปรแกรมย่อยของการบริการอินเตอร์รัพต์ โดยต้องมีการทำงานตามคำสั่งอย่างน้อยหนึ่งคำสั่งก่อนที่จะตอบสนองการอินเตอร์รัพต์อื่น ๆ อีก ปกติคำสั่งแรกของโปรแกรมบริการอินเตอร์รัพต์จะเป็น คำสั่ง DI หรือ PUSHF เป็นการกำหนดให้ดิสเอเบิลทุกอินเตอร์รัพต์ไว้ก่อน เพื่อสนองตอบอินเตอร์รัพต์ที่กำลังบริการอยู่ การดิสเอเบิลสามารถทำได้ 2 วิธี คือ เคลียร์ PSW.9 หรือเคลียร์รีจิสเตอร์ Interrupt mask เมื่อเสร็จสิ้นโปรแกรมบริการอินเตอร์รัพต์แล้วคำสั่งสุดท้ายจะเป็น POPF และตามด้วย RET

2.5 พอร์ตอินพุตเอาท์พุต 0, 1, 2, 3 และ 4

8096 จะมีพอร์ตอินพุตเอาท์พุตขนาด 8 บิตอยู่ 5 พอร์ต ผู้ใช้สามารถกำหนดให้เป็นพอร์ตอินพุตหรือพอร์ตแบบสองทิศทางก็ได้ บางพอร์ตยังสามารถเลือกฟังก์ชั่นพิเศษได้ด้วย อินพุตพอร์ตจะต่อโดยตรงกับบัฟเฟอร์แล้วต่อเข้ากับบัส สำหรับเอาท์พุตพอร์ตจะต่อกับบัฟเฟอร์และรีจีสเตอร์ภายใน ส่วนพอร์ตสองทิศทางจะต่อเข้ากับบัฟเฟอร์อินพุต, เอาท์พุต และรีจีสเตอร์ภายใน อินเทลกำหนดคุณสมบัติของพอร์ตต่าง ๆ ไว้ดังนี้

พอร์ต 0 เป็นอินพุตอย่างเดียว ขาของพอร์ต 0 จะใช้ร่วมกับวงจร A/D

พอร์ต 1 เป็นอินพุตเอาท์พุตแบบสองทิศทาง

พอร์ต 2 เป็นพอร์ตที่ทำงานได้หลายฟังก์ชั่นการใช้งานพอร์ตนี้จะแสดงดังตารางที่ 2.3

พอร์ต 3 และ 4 มี 2 หน้าที่คือ เป็นพอร์ตแบบสองทิศทาง และใช้เป็นบัสของระบบในการเชื่อมต่อกับหน่วยความจำภายนอก การใช้ขาร่วมกันแสดงได้ดังตารางที่ 2.4

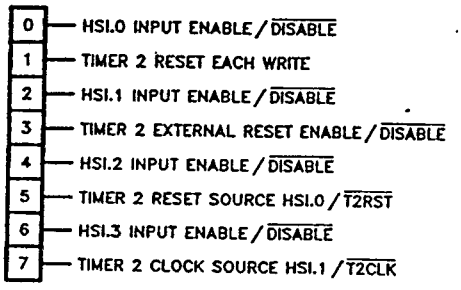
ตารางที่ 2.3 แสดงการกำหนดฟังก์ชั่นใช้งานพอร์ต 2

พอร์ต	ฟังก์ชั่น	ฟังก์ชั่นอื่น	ควบคุมโดย
P2.0	เอาท์พุต	พอร์ตอนุกรม TXD	IOC1.5
P2.1	อินพุต	พอร์ตอนุกรม RXD	N/A
P2.2	อินพุต	EXTINT อินเตอร์รัพต์ภายนอก	IOC1.1
P2.3	อินพุต	T2CLK ไทเมอร์ 2 อินพุต	IOC0.7
P2.4	อินพุต	T2RST ไทเมอร์ 2 รีเซต	IOC0.5
P2.5	เอาท์พุต	PWM	IOC1.0
P2.6	พอร์ตแบบสองทิศทาง		
P2.7	พอร์ตแบบสองทิศทาง		

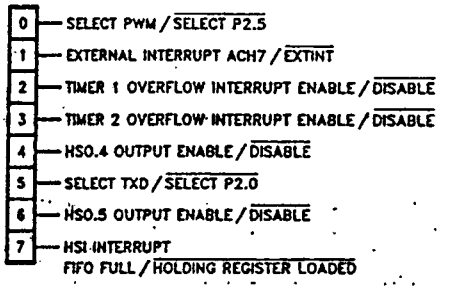
ตารางที่ 2.4 แสดงตำแหน่งขาที่ต่อกับบัสของระบบใน MCS-96

พอร์ต	บัสของระบบ
P3.0	AD 0
P3.1	AD 1
P3.2	AD 2
P3.3	AD 3
P3.4	AD 4
P3.5	AD 5
P3.6	AD 6
P3.7	AD 7
P4.0	AD 8
P4.1	AD 9
P4.2	AD 10
P4.3	AD 11
P4.4	AD 12
P4.5	AD 13
P4.6	AD 14
P4.7	AD 15

รีจีสเตอร์ที่ใช้ควบคุมอินพุตเอาท์พุตมีสองตัวคือ IOC0 และ IOC1 IOC0 ใช้ควบคุมไทเมอร์ 2 และ HSI ส่วน IOC1 ใช้ควบคุมการทำงานของขาบางขา และการกำหนดการอีนาเบิลต้นกำเนิดของสัญญาณอินเตอร์รัพต์แต่ละแหล่ง ภาพที่ 2.12 และ 2.13 แสดงรายละเอียดของ IOC0 และ IOC1

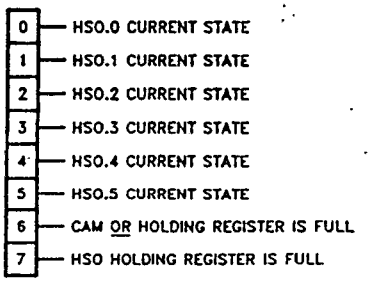


ภาพที่ 2.12 แสดงรีจีสเตอร์ควบคุม 0 (IOC0)

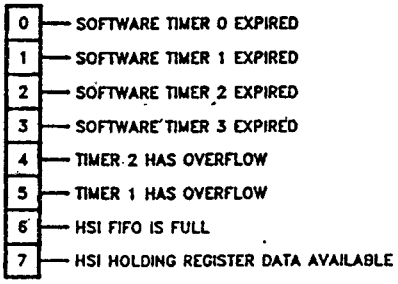


ภาพที่ 2.13 แสดงรีจีสเตอร์ควบคุม 1 (IOC1)

สำหรับรีจีสเตอร์แสดงสถานะของ I/O มีสองตัวคือ IOS0 และ IOS1 IOS0 จะแสดงสถานะของขา HSO และ CAM ดังแสดงในภาพที่ 2.14 ส่วน IOS1 แสดงสถานะของขา HSI ดังแสดงในภาพที่ 2.15



ภาพที่ 2.14 แสดงรีจีสเตอร์แสดงสถานะ I/O 0 (IOS0)

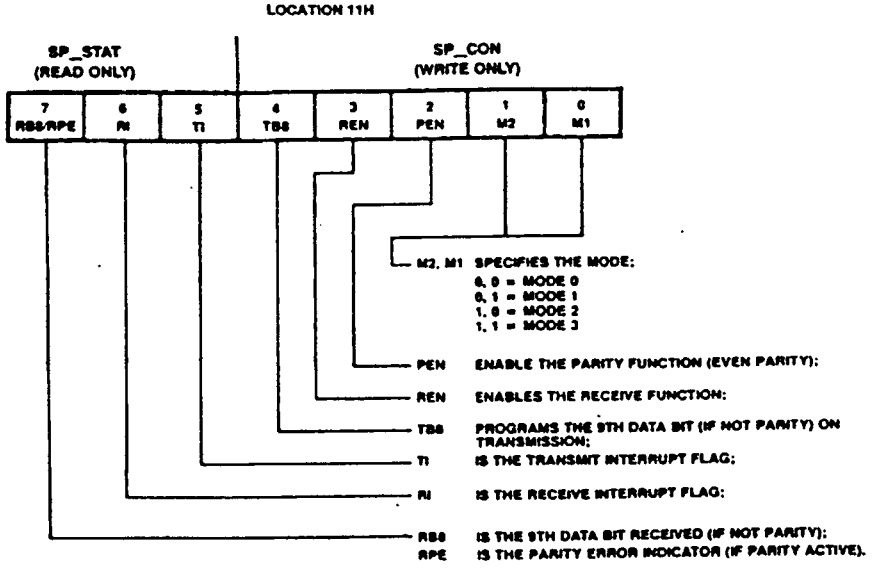


ภาพที่ 2.15 แสดงรีจีสเตอร์แสดงสถานะ I/O 1 (IOS1)

2.6 พอร์ตอนุกรม

พอร์ตอนุกรมของ 8096 มีลักษณะเหมือนกับพอร์ตที่ใช้ในชิพ 8051 คือมีทั้ง Asynchronous mode และ Synchronous mode สำหรับ Asynchronous mode เป็นการทำงานแบบฟูลดูเพล็กซ์ นั่นคือสามารถรับและส่งข้อมูลได้ในเวลาเดียวกัน เมื่อทำการรับข้อมูลจะถูกเก็บในบัฟเฟอร์ของอินพุต ดังนั้นก่อนที่ข้อมูลจะวิ่งเข้ามาอีกครั้งจะต้องทำการอ่านไบต์ที่อยู่ในบัฟเฟอร์ออกไปเสียก่อน มิฉะนั้นจะเกิดข้อผิดพลาดในการทำงานขึ้น บัฟเฟอร์ของการเขียนอยู่ที่ตำแหน่งแอดเดรส 07H สำหรับการส่งข้อมูลนั้นจะต้องเขียนข้อมูลที่จะส่งลงไปในตำแหน่งแอดเดรส 07H นี้

รีจีสเตอร์ที่ใช้ควบคุมและแสดงสถานะของพอร์ตอนุกรมอยู่ที่แอดเดรส 11H โดยแบ่งเป็นสองส่วน ส่วนหนึ่งเป็นส่วนที่เขียนอย่างเดียว อีกส่วนหนึ่งเป็นส่วนที่อ่านอย่างเดียว ส่วนที่เขียนเป็นส่วนที่ใช้สำหรับควบคุมการทำงาน ส่วนที่อ่านคือ ส่วนแสดงสถานะการทำงานของพอร์ตอนุกรมนี้ รายละเอียดของคำสั่งและสถานะของการใช้รีจีสเตอร์นี้แสดงดังภาพที่ 2.16



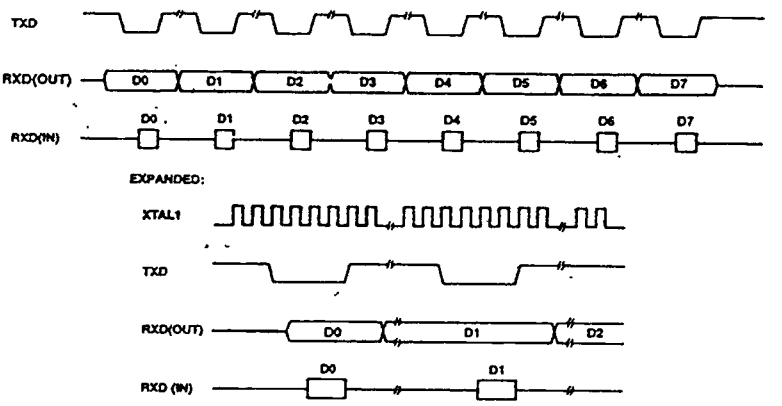
NOTE: TI and RI are cleared when SP_STAT is read

ภาพที่ 2.16 แสดงรีจีสเตอร์คำสั่ง/สถานะของพอร์ตอนุกรม

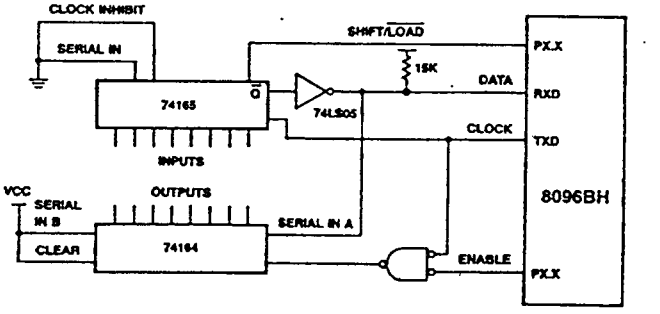
โหมดการทำงานของพอร์ตอนุกรมมี 4 โหมด คือ

โหมด 0 การทำงานจะเป็นแบบ Synchronous ที่มีการทำงานแบบเลื่อนบิต จะส่งหรือรับข้อมูลครั้งละ 8 บิต อย่างต่อเนื่องภายใน 8 ลูกของสัญญาณนาฬิกา โดยที่ขา TXD จะส่งสัญญาณพัลส์ออกมา และที่ขา RXD จะใช้ในการรับหรือส่งข้อมูล ไดอะแกรมเวลาของโหมดนี้แสดงในภาพที่ 2.17 ตัวอย่างวงจรการทำงานในโหมดนี้แแสดงในภาพที่ 2.18

หมายเหตุ ในโหมดการทำงานนี้เท่านั้นที่ขา RXD เป็น output ได้

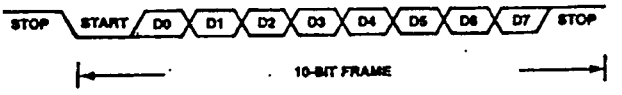


ภาพที่ 2.17 แสดงไดอะแกรมเวลาพอร์ตอนุกรมโหมด 0



ภาพที่ 2.18 แสดงตัวอย่างวงจรพอร์ตอนุกรมในโหมด 0

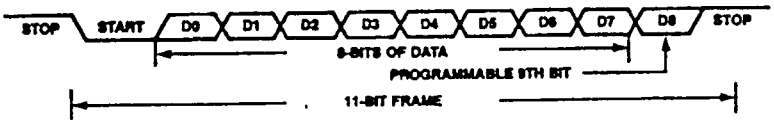
โหมด 1 เป็นโหมดที่มีการทำงานแบบ Asynchronous จะมีการรับหรือส่งครั้งละ 10 บิต เฟรมของข้อมูลจะมีลักษณะดังภาพที่ 2.19 โดยจะมีบิตเริ่มต้นเป็น "0" ตามด้วยข้อมูล 8 บิต บิตสต๊อป 1 บิต และถ้าให้มีการส่งพาริตี้โดยให้ PEN เป็น 1 จะมีการส่งพาริตี้ โหมดนี้จึงเหมาะที่จะใช้รับส่งข้อมูลกับ CRT



ภาพที่ 2.19 แสดงเฟรมของข้อมูลพอร์ตอนุกรม โหมด 1

โหมด 2 เป็นโหมดที่มีการทำงานแบบ Asynchronous จะมีการรับส่งข้อมูลครั้งละ 11 บิต เฟรมของข้อมูลจะมีลักษณะดังภาพที่ 2.20 เริ่มจาก บิตสตาร์เป็น "0" บิตข้อมูล 8 บิต บิตที่ 9 เป็นบิตที่สามารถโปรแกรมค่าได้และตามด้วยสต๊อปบิต ในการส่งบิตที่ 9 นี้จะเป็น 0 หรือ 1 ขึ้นอยู่กับการกำหนด TB8 และบิตนี้จะได้รับการเคลียร์ให้เป็น 0 ในแต่ละครั้งของการส่ง สำหรับในการรับจะทำให้ไม่ตอบสนองต่อการอินเตอร์รัพต์ยกเว้นว่าบิตที่ 9 จะมีค่าเป็น 1

โหมด 3 เป็นโหมดที่มีการทำงานแบบ Asynchronous จะมีการรับส่งข้อมูลครั้งละ 11 บิต โหมดนี้ก็จะคล้าย ๆ กับโหมด 2 ในการส่งบิตที่ 9 จะกำหนดด้วย TB8 และถ้า PEN = 1 บิตที่ 9 จะเป็นบิตพาริตี้ สำหรับการรับจะเก็บบิตที่ 9 ไว้ด้วย พอร์ตอนุกรมจะตอบสนองการอินเตอร์รัพต์โดยไม่ขึ้นกับบิตที่ 9



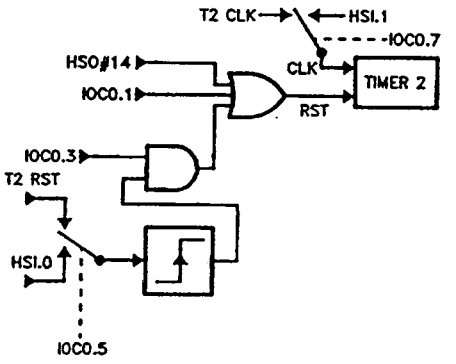
ภาพที่ 2.20 แสดงเฟรมข้อมูลพอร์ตอนุกรมโหมด 2 และ 3

2.7 ไทเมอร์

8096 ประกอบด้วยไทเมอร์ขนาด 16 บิต อยู่ 2 ตัว คือ ไทเมอร์ 1 และ ไทเมอร์ 2 โดยไทเมอร์ 1 ทำงานซิงโครไนซ์กับสัญญาณสเตตของซีพียู ส่วนไทเมอร์ 2 จะทำงานซิงโครไนซ์กับสัญญาณจากภายนอก

ไทเมอร์ 1 เป็นวงจรนับที่ทำการนับทุก ๆ 8 สเตตของสัญญาณซีพียู เมื่อถูกรีเซตไทเมอร์ 1 จะได้รับการเคลียร์ หากต้องการเปลี่ยนค่าของไทเมอร์ 1 สามารถทำได้โดยการเขียนข้อมูลที่ต้องการลงในแอดเดรสตำแหน่งที่ 000CH ขณะอยู่ในโหมดทดสอบ

ไทเมอร์ 2 จะเพิ่มค่าโดยสัญญาณจาก T2CLK หรือ HSI.1 เราสามารถกำหนดหน้าที่ของไทเมอร์ 2 ด้วยการกำหนดค่าใน IOC0 บิตที่ 7 (IOC0.7) เมื่อถูกรีเซตไทเมอร์ 2 จะได้รับการเคลียร์ นอกจากการเคลียร์ด้วยวิธีรีเซตแล้วยังสามารถทำได้อีกหลายวิธีเช่น เคลียร์โดยการกำหนดบิต 1 ของ IOC0 ให้เป็น "1" , การทริกให้กับ HSO#14, ทำให้ T2RST ให้เป็น "1" ภาพที่ 2.21 เป็นการแสดงการรีเซตไทเมอร์ 2



ภาพที่ 2.21 แสดงการป้อนสัญญาณนาฬิกาและการรีเซตไทเมอร์ 2

เมื่อเกิดสภาวะโอเวอร์โฟลว์ไทเมอร์ 1 และ ไทเมอร์ 2 จะส่งสัญญาณอินเตอร์รัพต์ให้กับซีพียู การควบคุมไทเมอร์อินเตอร์รัพต์นี้สามารถทำได้โดยการกำหนดค่าใน IOC1 บิต 2 และบิต 3

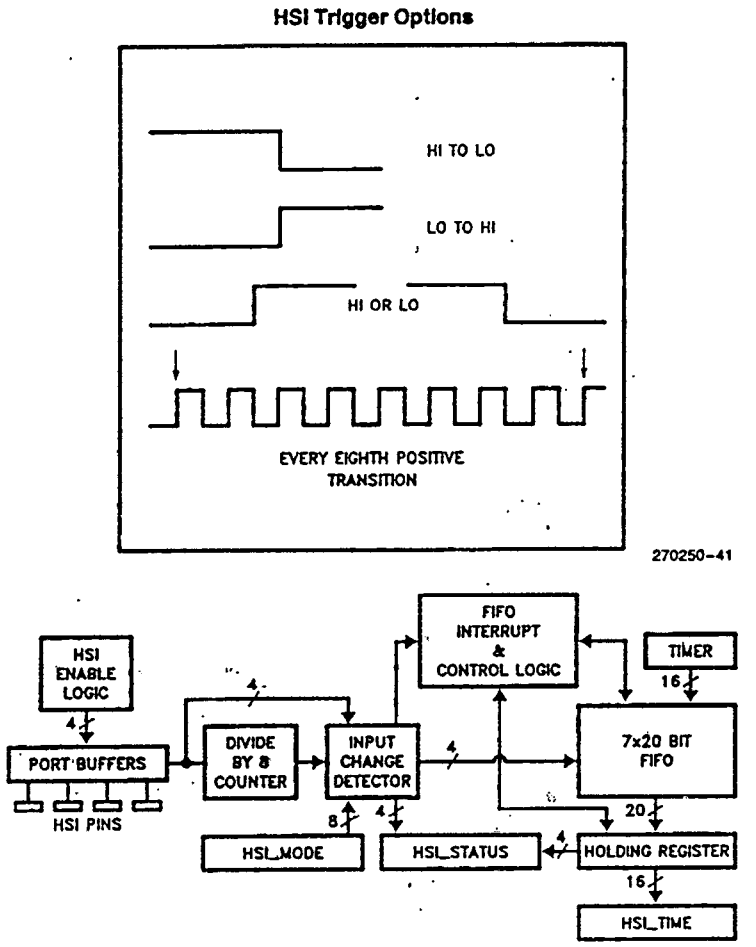
2.8 พอร์ตอินพุตความเร็วสูง (HSI)

8096 สามารถรับสัญญาณนาฬิกาที่มีความถี่สูงผ่านทางพอร์ตอินพุตที่มีชื่อว่า HSI และจะเก็บข้อมูลนี้ร่วมกับค่าของไทเมอร์ 1 ไว้ภายในรีจิสเตอร์ FIFO HSI มีทั้งหมด 4 อินพุตสามารถบันทึกข้อมูลได้ 8 ค่า ขาอินพุต HSI.2 และ HSI.3 จะใช้งานร่วมกับ HSO.4 และ HSO.5 บล็อกไดอะแกรมของ HSI แสดงในภาพที่ 2.22

HSI สามารถทำงานได้ 4 โหมด การทำงานแต่ละโหมดกำหนดโดยรีจิสเตอร์ HSI Mode การทริกให้กับ HSI สามารถทำได้หลายรูปแบบ เช่น ทริกเมื่อสัญญาณนาฬิกาเปลี่ยนจาก “1” เป็น “0” หรือเปลี่ยนจาก “0” เป็น “1” หรือเป็นพัลส์

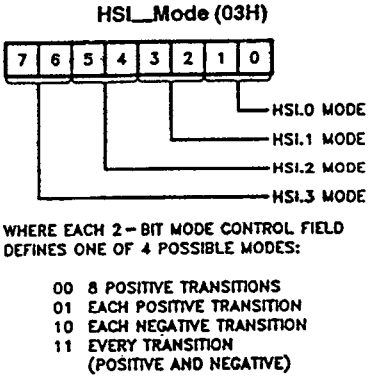
รีจิสเตอร์ FIFO ทำหน้าที่เก็บข้อมูลอินพุตและข้อมูลเวลาที่ได้จากไทเมอร์ 1 โครงสร้างของ FIFO มีขนาด 7x20 บิต โดยใช้ 16 บิตสำหรับบันทึกข้อมูลของไทเมอร์ 1 ที่เหลืออีก 4 บิตใช้สำหรับบันทึกค่าอินพุตของ HSI การเก็บข้อมูลส่วนนี้จะเก็บได้สูงสุด 8 ค่า แบ่งเป็น 7 ค่าเก็บไว้ที่ FIFO และอีก 1 ค่าเก็บไว้ที่โฮลดิ้งรีจิสเตอร์

จากที่กล่าวแล้วว่าการทริกของ HSI สามารถทำได้หลายรูปแบบ โดยรูปแบบการทริกนั้นขึ้นอยู่กับการกำหนดค่าในแอดเดรสตำแหน่งที่ 03H ซึ่งเป็นรีจิสเตอร์สำหรับการกำหนดโหมด (HIS Mode) รายละเอียดการกำหนดรูปแบบเป็นดังนี้ 00 หมายถึง ให้ทริกด้วยพัลส์ เมื่ออินพุตมีการเปลี่ยนสถานะไป 8 ครั้ง, 01 หมายถึงการทริกขณะอินพุตเปลี่ยนระดับลอจิกจาก “0” ไป “1”, 10 หมายถึง ให้ทริกเมื่ออินพุตเปลี่ยนระดับลอจิกจาก “1” ไป “0” และ 11 หมายถึง เปลี่ยนทุก ๆ การเปลี่ยนระดับลอจิก ภาพที่ 2.23 เป็นการแสดงบิตของรีจิสเตอร์กำหนดโหมด



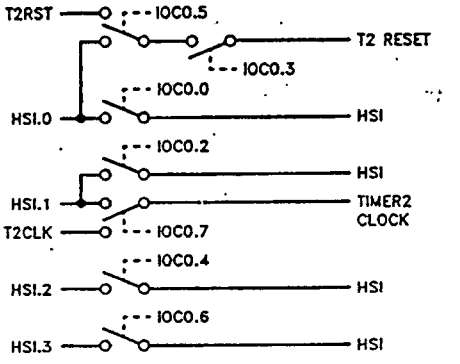
ภาพที่ 2.22 แสดงบล็อกไดอะแกรมของหน่วย HSI

การเกิดอินเตอร์รัพต์โดย HSI สามารถทำได้ 2 กรณีกำหนดโดยรีจิสเตอร์ IOC1 บิตที่ 7 (IOC1.7) ถ้าบิตนี้เป็น "0" หมายถึงให้มีการอินเตอร์รัพต์ทุกครั้งที่มีการโอนย้ายข้อมูลจาก FIFO ไปเก็บไว้ที่โฮลดิ้งรีจีสเตอร์ แต่ถ้าบิตนี้เป็น "1" หมายถึงจะเกิดการอินเตอร์รัพต์เมื่อข้อมูลอินพุต HSI ลำดับที่ 6 ถูกนำไปเก็บไว้ใน FIFO

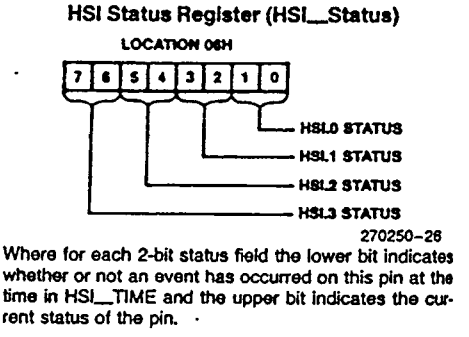


ภาพที่ 2.23 แสดงบิตของรีจีสเตอร์กำหนดโหมด

ในการควบคุมการทำงานของ HSI สามารถกำหนดได้ด้วยค่าที่อยู่ในรีจีสเตอร์ของ IOC0 โดยบิตต่าง ๆ จะหมายถึง การควบคุมสวิตซ์ ดังภาพที่ 2.24 และถ้าต้องการดูสถานะการทำงานของ HSI สามารถตรวจสอบได้จากแอดเดรสตำแหน่งที่ 06H ซึ่งค่าที่ได้แบ่งเป็น 2 บิตสำหรับ HSI แต่ละตัว แสดงดังภาพที่ 2.25 สถานะของ HSI แต่ละตัวใช้บิตต่ำแสดงถึงเหตุการณ์การทริกได้เกิดขึ้นแล้วหรือไม่ ส่วนอีกบิตใช้แสดงสถานะปัจจุบันของอินพุตแต่ละขา



ภาพที่ 2.24 แสดงการควบคุมผ่านรีจีสเตอร์ IOC0



ภาพที่ 2.25 แสดงรีจีสเตอร์แสดงสถานะของ HSI

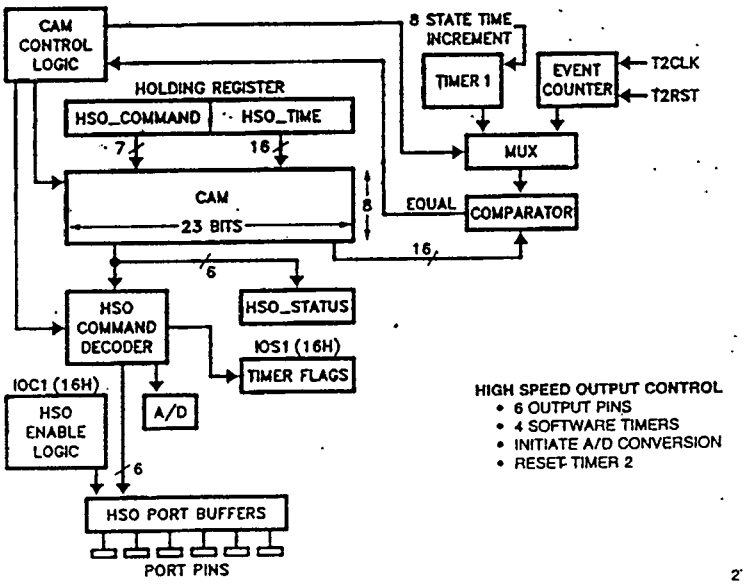
2.9 พอร์ตเอาท์พุตความเร็วสูง (HSO)

HSO ให้เอาท์พุตออกไป ณ เวลาที่กำหนด และจะส่งสัญญาณไปอินเตอร์รัพต์ซีพียู เพื่อบอกซีพียูให้รับรู้ว่าได้ส่งข้อมูลออกไปแล้ว สัญญาณเอาท์พุตของ HSO สามารถนำไปควบคุมได้หลายทาง เช่น สั่งให้วงจร A/D เริ่มต้นทำงาน, ทำการรีเซตไทเมอร์ 2, ควบคุมซอฟแวร์ไทเมอร์ และควบคุมสถานะของเอาท์พุต HSO ทั้ง 6 ขา

8096 ถูกออกแบบให้มีการใช้ขาร่วมกันระหว่าง HSI กับ HSO ด้วยเหตุนี้จึงเรียกขาเหล่านี้ว่า HSIO ตัวอย่างเช่น HSO.4 และ HSO.5 จะใช้งานร่วมกับ HSI.2 และ HSI.3 เป็นต้น

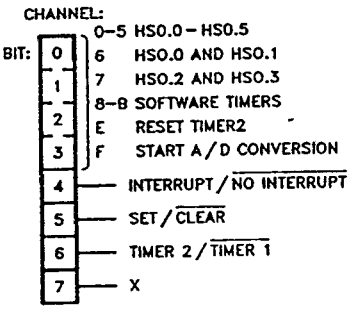
การทำงานของ HSO อาศัยหน่วยความจำที่เรียกว่า CAM (Content Addressable Memory) โดยที่ค่าใน CAM จะถูกนำไปเปรียบเทียบกับค่าของไทเมอร์เพื่อตรวจสอบว่าสามารถส่งข้อมูลออกไปได้หรือยัง บล็อกไดอะแกรมของหน่วย HSO แสดงในภาพที่ 2.26

HSO ถูกกำหนดการทำงานโดยรีจีสเตอร์ HSO_COMMAND ในการสั่งงานนั้นจำเป็นต้องตรวจสอบว่ารีจีสเตอร์โฮลดิ้งว่างอยู่หรือไม่ เพื่อป้องกันไม่ให้เกิดสภาวะการเขียนทับซ้อนขึ้น



ภาพที่ 2.26 แสดงบล็อกไดอะแกรมของหน่วย HSO

การควบคุมการทำงานของ HSO จะสั่งผ่านรีจิสเตอร์ HSO_COMMAND โดยรูปแบบของคำสั่ง แสดงดังภาพที่ 2.27



ภาพที่ 2.27 แสดงรูปแบบคำสั่งของ HSO_COMMAND

2.10 การรับสัญญาณอะนาลอก

อินเทลได้ออกแบบ 8096 ให้มีวงจรแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอลขนาด 10 บิต และมีจำนวนอินพุตทั้งหมด 8 ช่องผลลัพธ์ที่ได้จากการแปลง A/D จะเป็นค่าโดยประมาณ

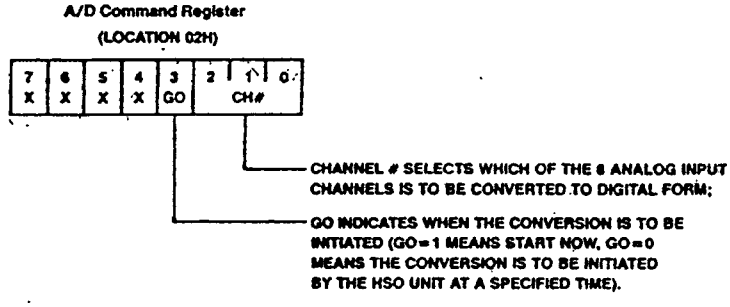
ของอัตราส่วนระหว่างสัญญาณอินพุตต่อแหล่งจ่ายไฟเปรียบเทียบกัน ถ้าอัตราส่วนนี้เป็น 1 ผลลัพธ์ที่ได้จากการแปลง A/D จะเป็น 1 ทั้ง 10 บิต

ในการแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอลแต่ละครั้งใช้เวลาประมาณ 88 สเตต หรือ 22 ไมโครวินาทีที่ความถี่ 12 เมกะเฮิรตซ์ ช่วงเวลานี้ไม่ได้ขึ้นกับค่าความละเอียด และค่าของอินพุต ระดับแรงดันอินพุตจะต้องอยู่ระหว่าง 0 โวลท์ถึง V_{REF} โดยค่า V_{REF} ที่เหมาะสมนั้นควรมีค่าประมาณ 4.5 ถึง 5.5 โวลต์ ผลลัพธ์ที่ได้จาก A/D คำนวณได้ดังนี้

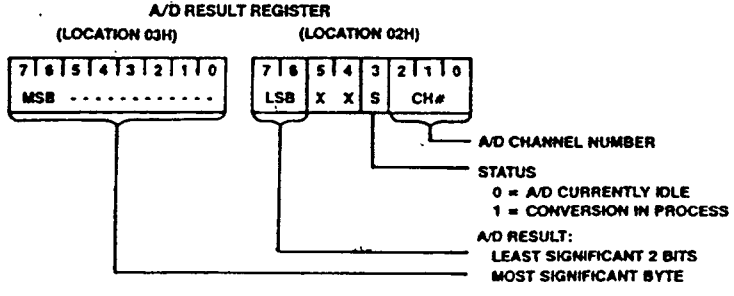
$$\text{ผลลัพธ์ของ A/D} = 1023 \times (\text{แรงดันอินพุต} - \text{ANGND}) / (V_{REF} - \text{ANGND}) \qquad (2.1)$$

จะเห็นว่าเมื่อแรงดัน V_{REF} หรือ ANGND เปลี่ยนไปจะส่งผลต่อผลลัพธ์ของ A/D หากต้องการความเที่ยงตรงสูง อาจต้องแยกใช้แหล่งจ่ายไฟโดยเฉพาะให้กับ A/D

ในการควบคุม A/D จะใช้รีจีสเตอร์ A/D COMMAND แสดงในภาพที่ 2.28 โดยรีจีสเตอร์ A/D COMMAND มีแอดเดรสอยู่ที่ตำแหน่ง 02H ทำหน้าที่เลือกช่องของอินพุตที่ต้องการให้ทำการแปลงสัญญาณ และกำหนดเงื่อนไขในการเริ่มแปลงสัญญาณ ผลลัพธ์ของ A/D จะเก็บไว้ที่แอดเดรสตำแหน่ง 02H และ 03H ดังแสดงในภาพที่ 2.29



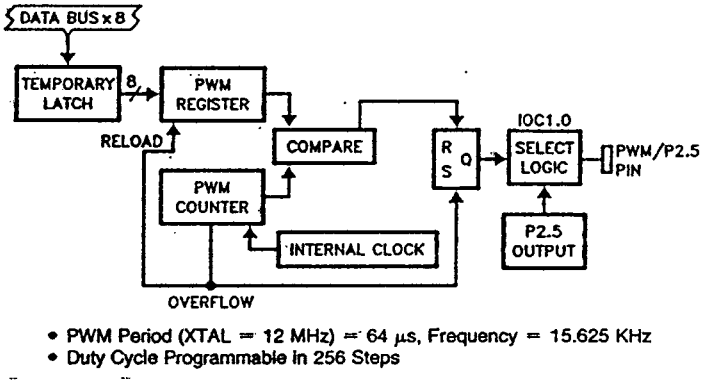
ภาพที่ 2.28 แสดงรีจีสเตอร์คำสั่งของ A/D



ภาพที่ 2.29 แสดงผลลัพธ์ของ A/D เก็บไว้ในรีจีสเตอร์ 02H และ 03H

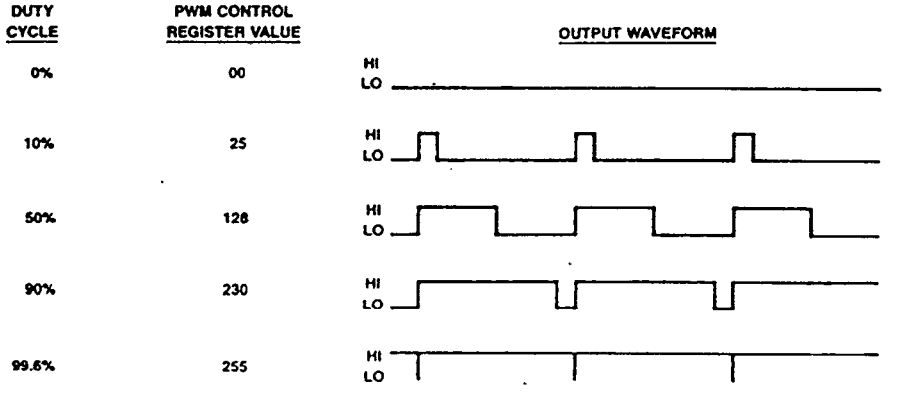
2.11 การสร้าง Pulse Width Modulation (PWM)

การเปลี่ยนสัญญาณดิจิตอลให้เป็นอะนาลอกสามารถทำได้โดยใช้วงจร PWM บล็อกไดอะแกรมของวงจร PWM แสดงในภาพที่ 2.30 ภายใน PWM จะประกอบด้วยวงจรนับขนาด 8 บิตซึ่งจะเพิ่มค่าในทุก ๆ ช่วงสเตต ค่าของวงจรนับนี้จะถูกนำไปเปรียบเทียบกับค่าของรีจิสเตอร์ PWM หากผลการเปรียบเทียบเท่ากัน ระดับแรงดันที่ขาเอาท์พุตของ PWM จะเปลี่ยนจาก "1" เป็น "0" และเมื่อวงจรนับเกิดโอเวอร์โฟลว์ ระดับแรงดันที่ขาเอาท์พุตของ PWM จะเปลี่ยนจาก "0" เป็น "1" สลับกันไป ค่าของ PWM สามารถกำหนดได้โดยเริ่มจาก 0 ถึง 255 รูปแบบของสัญญาณเอาท์พุตที่ได้แสดงดังภาพที่ 2.31



ภาพที่ 2.30 แสดงบล็อกไดอะแกรมของ PWM

เอาท์พุตของ PWM จะใช้งานร่วมกับขา 5 ของพอร์ต 2 การเลือกหน้าที่ใช้งานนั้นสามารถกำหนดได้โดยรีจิสเตอร์ IOC1 บิตที่ 0 (IOC1.0)

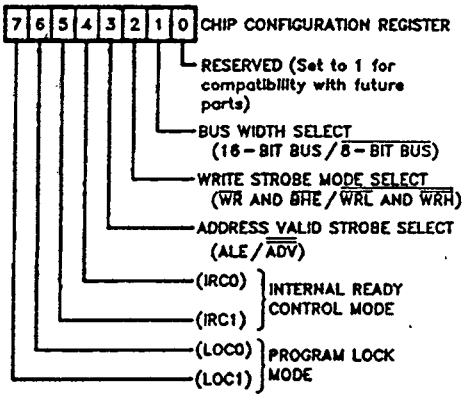


ภาพที่ 2.31 แสดงเอาท์พุตที่ได้จาก PWM

2.12 Chip Configuration Register (CCR)

Chip Configuration Register (CCR) เป็นรีจิสเตอร์ที่ใช้ควบคุมการทำงานของระบบบัส, กำหนดเวตสเตต (wait state) ของอินพุต READY และกำหนดโหมดการป้องกันการอ่านและเขียนข้อมูลหน่วยความจำภายในชิพ โครงสร้างของรีจิสเตอร์CCR สามารถแสดงได้ดังภาพที่ 2.32

เมื่อได้รับการรีเซต ซีพียูจะทำการอ่านค่า Chip Configuration Byte (CCB) จากแอดเดรสตำแหน่งที่ 2018H และทำการโหลดข้อมูลนี้ให้กับรีจิสเตอร์ CCR โดยจะได้รับการโหลดข้อมูลเพียงครั้งเดียวในช่วงเวลารีเซตและไม่สามารถเปลี่ยนแปลงได้จนกว่าซีพียูจะได้รับการรีเซตอีกครั้ง ถ้าระดับแรงดันที่ขา $\overline{EA}$ มีลอจิกเป็น "0" ซีพียูจะทำการอ่านค่า CCB จากหน่วยความจำภายนอก แต่ถ้าระดับแรงดันที่ขา $\overline{EA}$ มีลอจิกเป็น "1" ซีพียูจะทำการอ่านค่า CCB จากหน่วยความจำภายใน



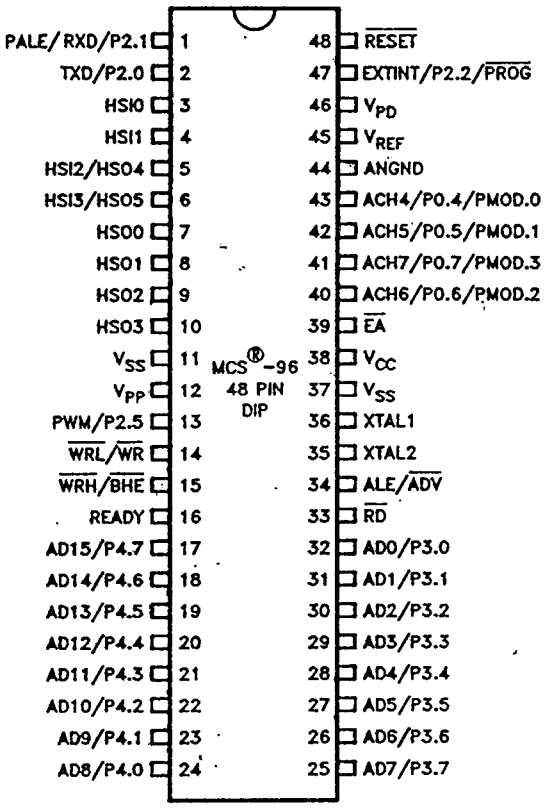
ภาพที่ 2.32 แสดง Chip Configuration Register

2.13 แพคเก็จของไมโครคอนโทรลเลอร์ตระกูล MCS-96

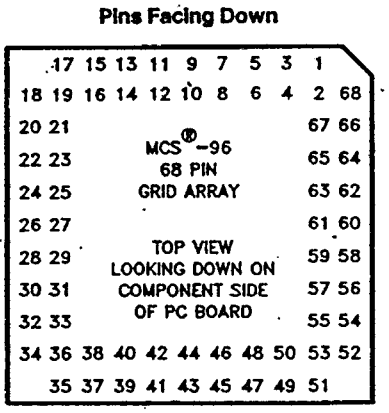
ไมโครคอนโทลเลอร์ตระกูล MCS-96 นี้ อินเทลได้ผลิตออกมาใช้งานหลายแพคเก็จ เช่น DIP-48, DIP-64 และ PGA-68 เป็นต้น ระบบการให้หมายเลขของไมโครคอนโทรลเลอร์ตระกูลนี้แสดงได้ดังตารางที่ 2.5 ในภาพที่ 2.33 และ 2.34 แสดงตัวอย่างของแพคเก็จแบบ DIP-48 และ PGA-68 ตามลำดับ

ตารางที่ 2.5 แสดงหมายเลขซีพียูของไมโครคอนโทรลเลอร์ในตระกูล MCS-96

	Factory Masked ROM			CPU			User Programmable					
							EPROM			OTP		
	68-pin	64-pin	48-pin	68-pin	64-pin	48-pin	68-pin	64-pin	48-pin	68-pin	64-pin	48-pin
ANALOG	8397BH	8397BH	8395BH	8097BH	8097BH	8095BH	8798BH		8795BH	8797BH	8797JF	8798
	8397JF	8397JF	8398	8097JF	8097JF	8098			8798	8797JF	8797BH	
NO ANALOG	8396BH			8X9X								



ภาพที่ 2.33 แสดงตัวอย่างของแพคเก็จแบบ 48 ขา



ภาพที่ 2.34 แสดงตัวอย่างของแพคเก็จแบบ 68 ขา

2.14 หน้าที่การทำงานของขาต่าง ๆ

หน้าที่การทำงานของขาต่างๆ สำหรับไมโครคอนโทรลเลอร์ตระกูล MCS-96 ที่มีแพคเก็จเป็นแบบ 48 ขานั้น สามารถอธิบายได้ดังนี้

V_{CC} รับแรงดันไฟเลี้ยงขนาด 5 โวลท์สำหรับจ่ายให้ส่วนต่างๆ ภายในซีพียู

V_{SS} เป็นดิจิตอลกราวด์มีจำนวน 2 ขา ในการใช้งานเราจะต้องต่อกราวด์ให้กับทั้ง 2 ขา

V_{PD} รับแรงดันไฟเลี้ยง สำหรับจ่ายให้รีจิสเตอร์ไฟล์ที่มีแอดเดรสระหว่าง 0F0H ถึง 0FFH จำนวน 16 ไบต์ ในกรณีที่ซีพียูทำงานในโหมดเพาเวอร์เดาน์

V_{REF} รับแรงดันไฟเลี้ยง เพื่อจ่ายเป็นแรงดันอ้างอิงให้กับวงจรแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอล

ANGND เป็นอะนาลอกกราวด์สำหรับวงจรแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอล ในการใช้งานนั้นจะต้องขานี้เข้ากับขา V_{SS}

V_{PP} รับแรงดันขนาด 12.75V เมื่อทำการโปรแกรมอีพีรอมของซีพียูที่มีหน่วยความจำ ROM หรือ EPROM ภายในชิพ

XTAL1 เป็นขาอินพุตของวงจรกำเนิดสัญญาณนาฬิกาภายใน

XTAL2 เป็นขาเอาท์พุตของวงจรกำเนิดสัญญาณนาฬิกาภายใน

RESET เป็นขาอินพุตใช้สำหรับรีเซตซีพียู

EA เป็นขาอินพุตใช้สำหรับเลือกหน่วยความจำที่แอดเดรสระหว่าง 2000H ถึง 3FFFH หากระดับแรงดันที่ขานี้มีลอจิกเป็น 1 ซีพียูจะทำการติดต่อกับหน่วยความจำแบบรอมภายในชิพ แต่ถ้ามีลอจิกเป็น 0 ซีพียูจะทำการติดต่อกับหน่วยความจำแบบรอมภายนอก และเมื่อใดที่ขานี้ได้รับแรงดันขนาด 12.75V ซีพียูจะเข้าสู่โหมดการโปรแกรมหน่วยความจำภายในชิพ

ALE/ADV เป็นขาเอาท์พุตที่ต่อเข้ากับวงจรดีมัลติเพล็กซ์เพื่อทำการแยกแอดเดรสบัส A0 ถึง A7 ออกจาก AD0 ถึง AD7 โดยขานี้จะถูกใช้งานเมื่อซีพียูติดต่อกับหน่วยความจำภายนอกเท่านั้น

RD เป็นขาเอาท์พุต จะทำงานเมื่อซีพียูทำการอ่านข้อมูลจากหน่วยความจำภายนอกเท่านั้น

WR เป็นขาเอาท์พุต จะทำงานเมื่อซีพียูทำการเขียนข้อมูลลงในหน่วยความจำภายนอกเท่านั้น

READY เป็นขาอินพุต เมื่อใดที่ขานี้มีลอจิกเป็น 0 ซีพียูจะทำการแทรกเวทสเตตเข้าไปในบัสไซเกิลของการติดต่อกับหน่วยความจำภายนอก โดยจำนวนของเวทสเตตสามารถกำหนดได้ด้วยค่าของชิพคันฟิกิวเรชันรีจิสเตอรบิตที่ 4 และ 5

HSI เป็นขาอินพุตของส่วนอินพุตความเร็วสูงมีทั้งหมด 4 ขาได้แก่ HSI.0, HSI.1, HSI.2 และ HSI.3 โดย 2 ขาสุดท้ายจะใช้งานร่วมกันกับส่วนเอาท์พุตความเร็วสูง

HSO เป็นขาเอาท์พุตของส่วนเอาท์พุตความเร็วสูงมีทั้งหมด 6 ขาได้แก่ HSO.0, HSO.1, HSO.2, HSO.3, HSO.4 และ HSO.5 โดย 2 ขาสุดท้ายจะใช้งานร่วมกันกับส่วนอินพุตความเร็วสูง

Port 0 เป็นขาอินพุตที่มีค่าอินพุตอิมพิแดนซ์สูง ประกอบด้วย P0.0 ถึง P0.7 โดยสามารถใช้เป็นได้ทั้งดิจิตอลอินพุตและอะนาลอกอินพุตของวงจรแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอล

Port 2 มีทั้งหมด 8 ขาเป็นได้ทั้งอินพุตและเอาท์พุต นอกจากนี้ยังมีหน้าที่พิเศษอื่นๆ อีกด้วย

Port 3 และ **Port 4** มีขนาด 8 บิตโครงสร้างเป็นแบบโอเพ็น-เดรน ใช้งานร่วมกันระหว่างแอดเดรสบัส และดาต้าบัส นอกจากนี้ยังใช้เป็นทางผ่านของคำสั่ง, แอดเดรส และดาต้าสำหรับซีพียูที่มีหน่วยความจำ ROM หรือ EPROM ภายในชิพ ขณะทำงานในโหมดโปรแกรมหน่วยความจำ

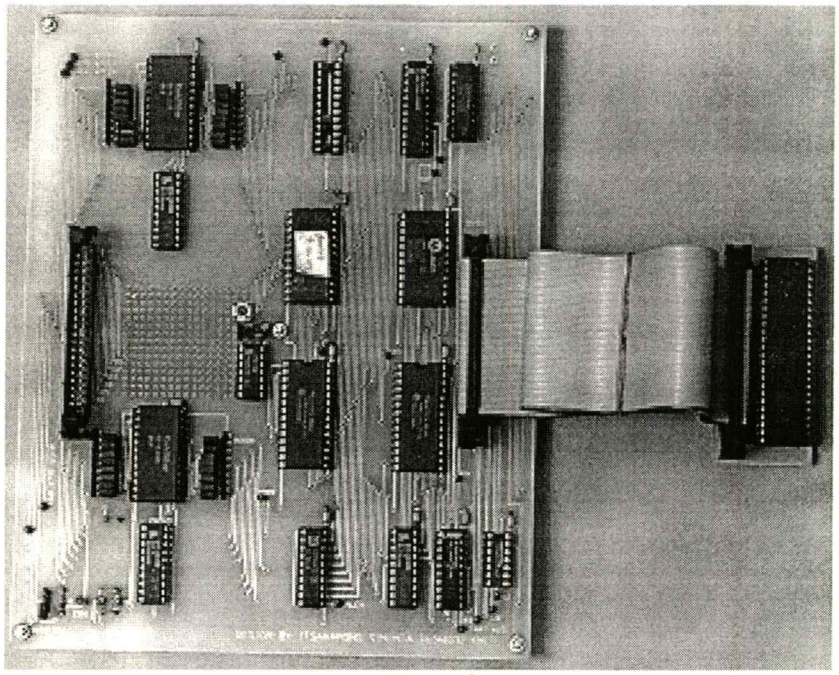
สำหรับขา PMODE, PACT, PVAL, SALE, SPROG, SID, PALE, PROG, PVER, PVAL และ PDO นั้นใช้สำหรับควบคุมการโปรแกรมหน่วยความจำภายในชิพ ซึ่งมีอยู่ในซีพียูที่มีหน่วยความจำ EPROM ภายในชิพเท่านั้น และจะไม่ใช้งานขาเหล่านี้ในกรณีที่โครงสร้างของซีพียูไม่มีหน่วยความจำภายในชิพ

บทที่ 3

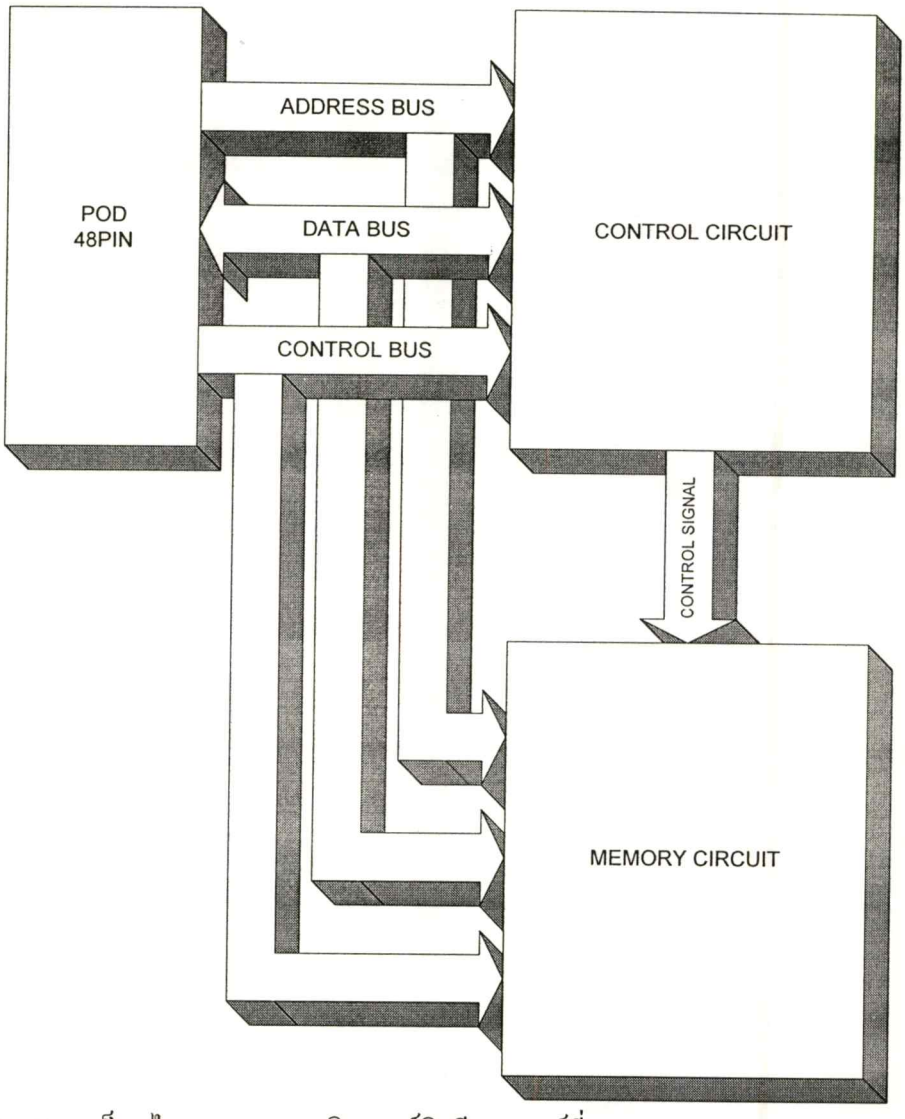
การออกแบบทางด้านฮาร์ดแวร์และซอร์ฟแวร์

3.1 การออกแบบทางด้านฮาร์ดแวร์ [7], [8], [9], [10]

แนวคิดในการออกแบบฮาร์ดแวร์นั้นได้นำเอาหลักการการใช้งานอุปกรณ์ต่างๆ ร่วมกันมาประยุกต์ใช้เพื่อลดขนาดของวงจร และประหยัดต้นทุนในการสร้างอินเซอร์กิตอีมูเลเตอร์ ดังนั้นอินเซอร์กิตอีมูเลเตอร์ที่ทำการออกแบบจึงประกอบด้วยส่วนต่างๆ ที่สำคัญเพียง 3 ส่วนด้วยกันคือ พ็อด (POD) ขนาด 48 ขา, วงจรควบคุม และวงจรหน่วยความจำ สำหรับแหล่งจ่ายไฟเลี้ยง, วงจรออสซิลเลเตอร์, พอร์ตอนุกรม และซีพียู ที่มีอยู่บนบอร์ดเป้าหมายนั้น อินเซอร์กิตอีมูเลเตอร์จะใช้งานร่วมกันกับบอร์ดเป้าหมาย



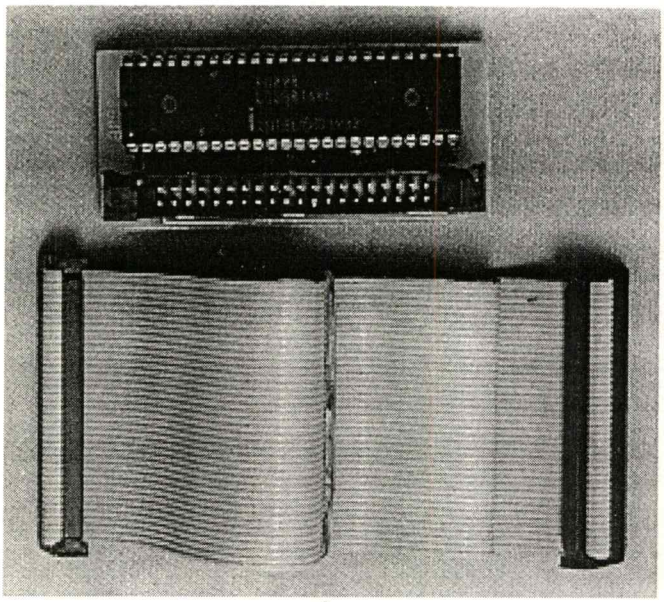
ภาพที่ 3.1 แสดงฮาร์ดแวร์ของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ



ภาพที่ 3.2 แสดงบล็อกไดอะแกรมของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ

3.1.1 รายละเอียดของการออกแบบพ็อด (POD)

พ็อดทำหน้าที่ในการเชื่อมต่อสัญญาณต่างๆ ระหว่างบอร์ดเป้าหมายกับอินเซอร์กิตอีมูเลเตอร์ประกอบด้วยคอนเน็คเตอร์ตัวผู้ขนาด 40 ขา และซ็อคเค็ทไอซีแบบดูอัลอินไลน์ (DIP) ขนาด 48 ขา พื้นที่ของพ็อดจำเป็นต้องจำกัดให้มีขนาดเล็ก เพื่อให้สามารถติดตั้งลงบนบอร์ดเป้าหมายได้โดยไม่ติดกับอุปกรณ์ใดๆ ของบอร์ดเป้าหมาย ในการใช้งานนั้นผู้ใช้จะต้องถอดซีพียูของบอร์ดเป้าหมายออกก่อน จากนั้นให้ติดตั้งพ็อดลงบนซ็อคเค็ทไอซีที่ว่างอยู่แล้วจึงติดตั้งซีพียูลงบนซ็อคเค็ทไอซีของพ็อดอีกที ในการเชื่อมต่อสัญญาณต่างๆ ระหว่างพ็อดกับอินเซอร์กิตอีมูเลเตอร์นั้นจะใช้สายแพขนาด 40 เส้น โดยที่ปลายทั้งสองด้านของสายแพจะต่อเข้ากับคอนเน็คเตอร์ตัวเมียขนาด 40 ขาดังแสดงในภาพที่ 3.3



ภาพที่ 3.3 แสดงพ็อดของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ

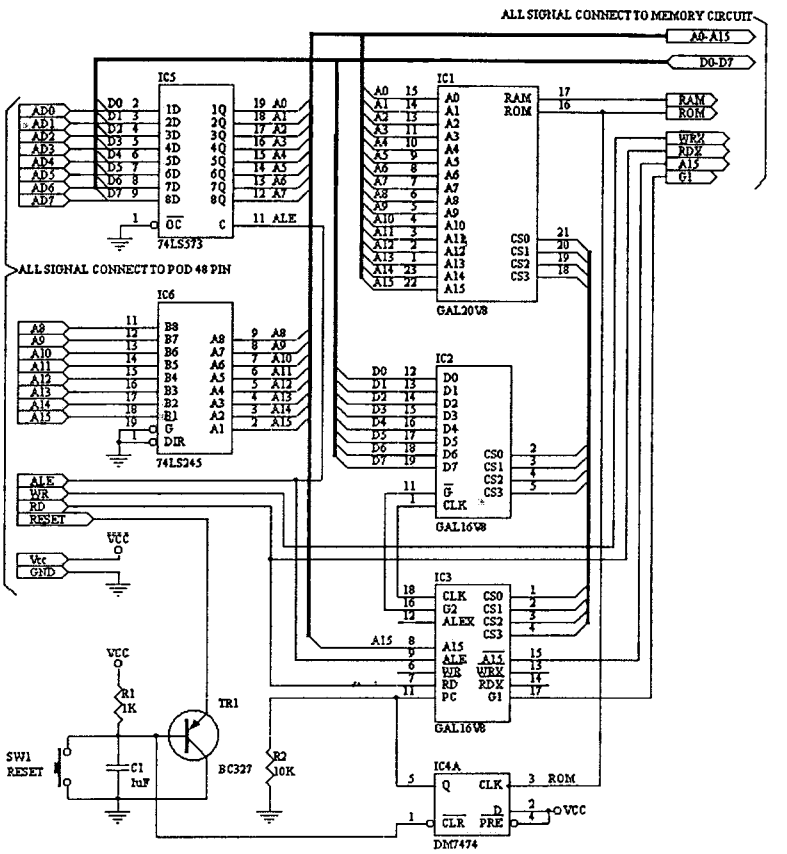
3.1.2 รายละเอียดของการออกแบบวงจรควบคุม (Control Circuit) [11], [12]

แผนภูมิพื้นที่หน่วยความจำของอินเซอร์กิตอีมูเลเตอร์แสดงดังตารางที่ 3.1 เมื่อพิจารณาข้อมูลในตารางพบว่าแอดเดรสเริ่มต้นของโปรแกรมมอนิเตอร์ (Program Monitor) ที่ใช้ควบคุมอินเซอร์กิตอีมูเลเตอร์นั้นกำหนดไว้ที่ตำแหน่ง 0800H ดังนั้นจึงต้องออกแบบวงจรควบคุมที่สามารถสั่งงานให้ซีพียูของบอร์ดเป้าหมายกระโดดไปที่ตำแหน่ง 0800H เพื่อทำงานตามคำสั่งในโปรแกรมมอนิเตอร์ได้โดยอัตโนมัติทุกๆ ครั้งที่ทำการรีเซตอินเซอร์กิตอีมูเลเตอร์ วงจรควบคุมที่ออกแบบไว้แสดงดังภาพที่ 3.4

ตารางที่ 3.1 แสดงแผนภูมิพื้นที่หน่วยความจำของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ

แอดเดรส	หน้าที่การใช้งาน
0000H - 07FFH	อินพุต, เอาท์พุตพอร์ต
0800H - 0FFFH	โปรแกรมมอนิเตอร์
1000H - 17FFH	แรมมอนิเตอร์
1800H - 1FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
2000H - 27FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
2800H - 2FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
3000H - 37FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
3800H - 3FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
4000H - 47FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
4800H - 4FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
5000H - 57FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
5800H - 5FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
6000H - 67FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
6800H - 6FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
7000H - 77FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
7800H - 7FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์

แอดเดรส	หน้าที่การใช้งาน
8000H - 87FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
8800H - 8FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
9000H - 97FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
9800H - 9FFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
A000H - A7FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
A800H - AFFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
B000H - B7FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
B800H - BFFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
C000H - C7FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
C800H - CFFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
D000H - D7FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
D800H - DFFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
E000H - E7FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
E800H - EFFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
F000H - F7FFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์
F800H - FFFFH	อินพุต, เอาท์พุตพอร์ต หรือโปรแกรมยูสเซอร์



ภาพที่ 3.4 แสดงวงจรควบคุมของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ

GAL20V8 (ไอซี 1) ทำหน้าที่ในการถอดรหัสสัญญาณแอดเดรสบัส A0 ถึง A15 GAL16V8 (ไอซี 2) ทำหน้าที่ในการเก็บข้อมูลชิพคอนฟิกิวเรชันไบต์ (Chip Configuration Byte) และออปโคด (Opcode) ของคำสั่งการกระโดดไปที่แอดเดรส 0800H สำหรับ GAL16V8 (ไอซี 3), 7474 (ไอซี4) และทรานซิสเตอร์ BC327 ทำหน้าที่ร่วมกันในการสร้างสัญญาณควบคุม 74LS573 (ไอซี 5) ทำหน้าที่ในการแลทช์สัญญาณแอดเดรสบัสด้านไบต์ต่ำ (A0 ถึง A7) ส่วน 74LS245 (ไอซี 6) ทำหน้าที่เป็นบัฟเฟอร์ให้กับสัญญาณแอดเดรสบัสด้านไบต์สูง (A8 ถึง A15) รายละเอียดของการออกแบบไอซี แต่ละตัวแยกอธิบายได้ดังต่อไปนี้

ไอซี 1 มีจำนวนขาอินพุตทั้งหมด 16 ขาโดยจะต่อเข้ากับสัญญาณแอดเดรสบัส A0 ถึง A15 ส่วนขาเอาท์พุตนั้นมีทั้งหมด 6 ขาประกอบด้วย $\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$, $\overline{CS3}$, $\overline{ROM}$ และ $\overline{RAM}$ โดยขา $\overline{CS0}$ จะแอกทีฟเมื่อสัญญาณแอดเดรสที่ขาอินพุตของไอซี 1 มีตำแหน่งเป็น 2018H (ตำแหน่งของชิพคอนฟิกิวเรชันไบต์) ส่วนขา $\overline{CS1}$, $\overline{CS2}$ และ $\overline{CS3}$ นั้นจะแอกทีฟเมื่อสัญญาณแอดเดรสที่ขาอินพุตของไอซี 1 มีตำแหน่งเป็น 2080H, 2081H, 2082H ตามลำดับ สำหรับขา $\overline{ROM}$ และ $\overline{RAM}$ จะแอกทีฟเมื่อสัญญาณแอดเดรสที่ขาอินพุตของไอซี 1 มีตำแหน่งอยู่ในช่วง 0800H ถึง 0FFFH (แอดเดรสของโปรแกรมมอนิเตอร์) และ 1000H ถึง 17FFH (แอดเดรสของแรมมอนิเตอร์) ตามลำดับ ตารางความจริงที่ใช้ในการออกแบบไอซี 1 แสดงได้ดังตารางที่ 3.2

ตารางที่ 3.2 แสดงตารางความจริงของไอซี 1

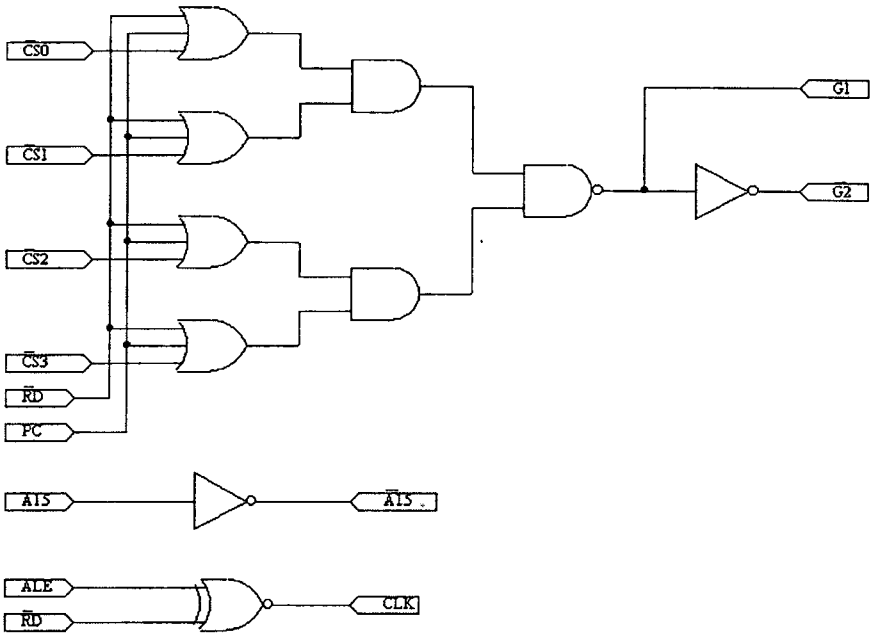
INPUT																OUTPUT						REMARK
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	CS0	CS1	CS2	CS3	ROM	RAM	ADDRESS
0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	2018H
0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	1	2080H
0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	1	1	1	2081H
0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	0	1	1	2082H
0	0	0	0	1	0	0	0	X	X	X	X	X	X	X	X	1	1	1	1	0	1	0800H - 0FFFH
0	0	0	1	0	X	X	X	X	X	X	X	X	X	X	X	1	1	1	1	1	0	1000H - 17FFH

ไอซี 2 มีจำนวนขาอินพุตทั้งหมด 6 ขาประกอบด้วย *CLK*, $\overline{G}$, $\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$ และ $\overline{CS3}$ สำหรับขาเอาท์พุตนั้นเป็นแบบสามสถานะมีทั้งหมด 8 ขา (D0 ถึง D7) ซึ่งทั้ง 8 ขานี้จะถูกต่อเข้ากับดาต้าบัส D0 ถึง D7 ของซีพียูบนบอร์ดเป้าหมายโดยตรง เมื่อสัญญาณอินพุตเป็นไปตามเงื่อนไขที่กำหนดไว้ไอซี 2 จะส่งข้อมูลออกไปยังดาต้าบัส ซึ่งข้อมูลเหล่านี้ได้แก่ ชิพคอนฟิกิวเรชันไบต์ และออปโคดของคำสั่งการกระโดดไปที่แอดเดรส 0800H นั่นเอง ตารางความจริงที่ใช้ในการออกแบบไอซี 2 แสดงได้ดังตารางที่ 3.3

ตารางที่ 3.3 แสดงตารางความจริงของไอซี 2

INPUT						OUTPUT								REMARK
CLK	G	CS3	CS2	CS1	CS0	D7	D6	D5	D4	D3	D2	D1	D0	DATA
↑	1	x	x	x	x	z	z	z	z	z	z	z	z	HIGH IMPEDANCE
↑	0	1	1	1	0	0	0	1	1	0	1	0	1	35H
↑	0	1	1	0	1	1	1	1	0	0	1	1	1	E7H
↑	0	1	0	1	1	0	1	1	1	1	1	0	1	7DH
↑	0	0	1	1	1	1	1	1	0	0	1	1	1	E7H
↑	0	1	1	1	1	1	1	1	1	1	1	1	1	FFH

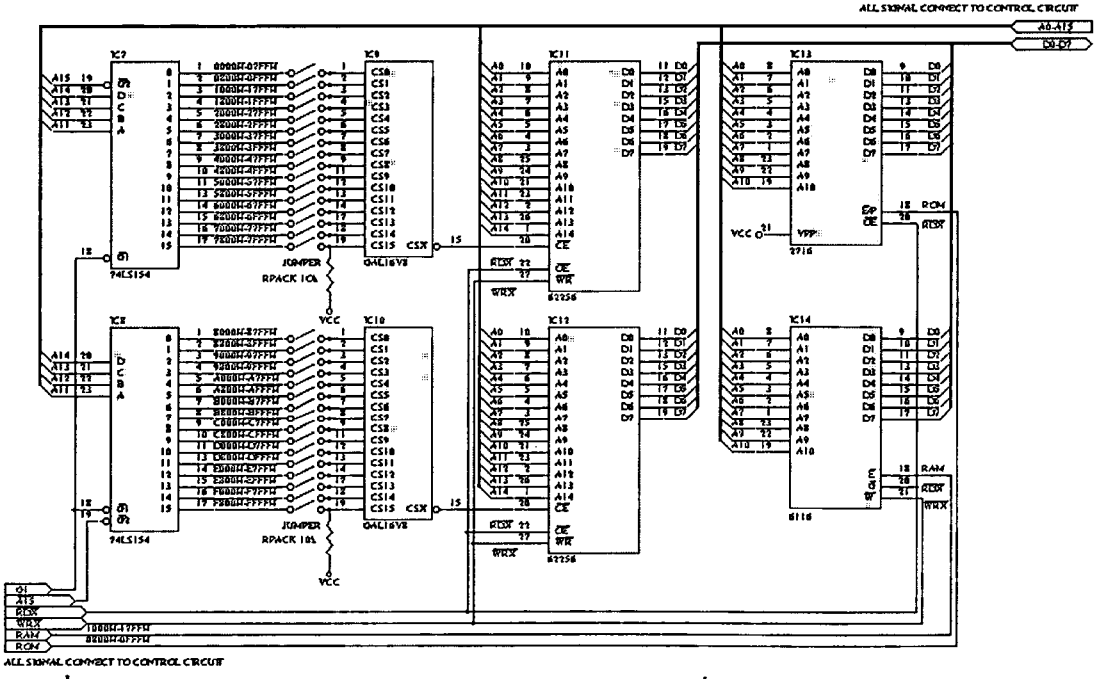
ไอซี 3 มีจำนวนขาอินพุตทั้งหมด 8 ขาประกอบด้วย $\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$, $\overline{CS3}$, $\overline{RD}$, *A*15, *ALE* และ *PC* ส่วนขาเอาท์พุตมีทั้งหมด 4 ขาประกอบด้วย $\overline{A15}$, $\overline{G2}$, $\overline{G1}$ และ *CLK* ภายในไอซี 3 ถูกออกแบบให้มีวงจรลอจิกเกตอยู่จำนวน 3 ชุดแสดงได้ดังภาพที่ 3.5



ภาพที่ 3.5 แสดงวงจรลอจิกเกตภายในไอซี 3 ที่ออกแบบ

3.1.3 รายละเอียดการออกแบบวงจรหน่วยความจำ (Memory Circuit)

อีพีรอมที่นิยมใช้มีด้วยกันหลายเบอร์เช่น 2716, 2732, 2764, 27128 และเบอร์ 27256 โดยสามารถเก็บข้อมูลได้ 2, 4, 8, 16 และ 32 กิโลไบต์ตามลำดับ ด้วยเหตุนี้จึงได้ทำการแบ่งพื้นที่หน่วยความจำแรมขนาด 64 กิโลไบต์ของอินเซอร์กิตอีมูเลเตอร์ออกเป็นช่วงๆ ช่วงละ 2 กิโลไบต์ เพื่อให้สอดคล้องกับขนาดของอีพีรอมที่มีอยู่ วงจรหน่วยความจำที่ออกแบบไว้แสดงดังภาพที่ 3.6



ภาพที่ 3.6 แสดงวงจรหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์ที่ออกแบบ

74LS154, GAL16V8 (ไอซี 7, 8, 9 และ 10) ทำหน้าที่ร่วมกันเป็นวงจรถอดรหัส 62256 (ไอซี 11 และไอซี 12) เป็นหน่วยความจำแรมขนาด 32 กิโลไบต์ใช้สำหรับเก็บโปรแกรมยูสเซอร์ 2716 (ไอซี 13) เป็นหน่วยความจำอีพีรอมขนาด 2 กิโลไบต์ใช้สำหรับเก็บโปรแกรมมอนิเตอร์ของอินเซอร์กิตอีมูเลเตอร์ ส่วน 6116 (ไอซี 14) เป็นหน่วยความจำแรมขนาด 2 กิโลไบต์ใช้เป็นแรมมอนิเตอร์ของอินเซอร์กิตอีมูเลเตอร์ รายละเอียดของการออกแบบไอซีแต่ละตัวแยกอธิบายได้ดังต่อไปนี้

ไอซี 7 และไอซี 8 ทำหน้าที่เป็น 4-16 Line Decoder มีจำนวนอินพุตทั้งหมด 6 อินพุต ได้แก่ $D, C, B, A, \overline{G2}$ และ $\overline{G1}$ และมีจำนวนเอาท์พุตทั้งหมด 16 เอาท์พุตได้แก่ 0 ถึง 15 เนื่องจากพื้นที่หน่วยความจำแรมขนาด 64 กิโลไบต์ของอินเซอร์กิตอีมูเลเตอร์ถูกแบ่งออกเป็นช่วงๆ ช่วงละ 2 กิโลไบต์ ดังนั้นจึงจำเป็นต้องใช้สัญญาณในการถอดรหัสหน่วยความจำทั้งหมด 32 เส้น โดยกำหนดให้ไอซี 7 ทำหน้าที่ถอดรหัสหน่วยความจำ 32 กิโลไบต์แรกซึ่งมีแอดเดรสอยู่ระหว่าง 0000H - 7FFFH และไอซี 8 ทำหน้าที่ถอดรหัสหน่วยความจำ 32 กิโลไบต์ที่เหลือซึ่งมีแอดเดรสอยู่ระหว่าง 8000H - 0FFFFH

ไอซี 9 และไอซี 10 ทำหน้าที่เป็นแอนด์เกตมีจำนวนอินพุตทั้งหมด 16 อินพุตได้แก่ $CS0$ ถึง $CS15$ โดยอินพุตทั้งหมดนี้จะต่อผ่านจัมเปอร์ (Jumper) เข้ากับเอาท์พุตของไอซี 7 และไอซี 8 สำหรับเอาท์พุตของไอซี 9 และไอซี 10 นี้มีเพียง 1 เอาท์พุตคือ $\overline{CSX}$

ไอซี 11 และไอซี 12 เป็นหน่วยความจำแรมขนาด 32 กิโลไบต์ ในการออกแบบกำหนดให้ไอซี 11 ทำหน้าที่ในการเก็บโปรแกรมยูสเซอร์ที่มีแอดเดรสอยู่ระหว่าง 0000H ถึง 7FFFH โดยอินพุต $\overline{CE}$ ของไอซี 11 จะต่อเข้ากับเอาท์พุต $\overline{CSX}$ ของไอซี 9 สำหรับโปรแกรมยูสเซอร์ที่มีแอดเดรสอยู่ระหว่าง 8000H ถึง 0FFFFH นั้นจะเก็บไว้ในไอซี 12 โดยอินพุต $\overline{CE}$ ของไอซี 12 จะต่อเข้ากับเอาท์พุต $\overline{CSX}$ ของไอซี 10

ไอซี 13 เป็นหน่วยความจำอีพีรอมขนาด 2 กิโลไบต์ทำหน้าที่ในการเก็บโปรแกรมมอนิเตอร์ซึ่งใช้ควบคุมการทำงานของอินเซอร์กิตอีมูเลเตอร์ อินพุต $\overline{CE}$ ของไอซี 13 จะต่อเข้ากับเอาท์พุต $\overline{ROM}$ ของไอซี 1 ซึ่งใช้ในการถอดรหัสหน่วยความจำที่มีแอดเดรสอยู่ระหว่าง 0800H ถึง 0FFFH

ไอซี 14 เป็นหน่วยความจำแรมขนาด 2 กิโลไบต์ ทำหน้าที่เป็นแรมมอนิเตอร์ซึ่งใช้เก็บข้อมูลต่างๆ ของอินเซอร์กิตอีมูเลเตอร์เช่น มอนิเตอร์สแตก, ยูสเซอร์สแตก, ยูสเซอร์รีจิสเตอร์ และเบรคพอยบัฟเฟอร์ เป็นต้น อินพุต $\overline{CE}$ ของไอซี 14 จะต่อเข้ากับเอาท์พุต $\overline{RAM}$ ของไอซี 1 ซึ่งใช้ในการถอดรหัสหน่วยความจำที่มีแอดเดรสอยู่ระหว่าง 1000H ถึง 17FFH

3.2 การออกแบบทางด้านซอฟแวร์ [13], [14], [15~17], [18], [19]

ซอฟแวร์ของอินเซอร์กิตอีมูเลเตอร์นั้นประกอบด้วย 2 ส่วนได้แก่ โปรแกรมมอนิเตอร์ ทำหน้าที่ควบคุมการทำงานของฮาร์ดแวร์ของอินเซอร์กิตอีมูเลเตอร์ และโปรแกรมการใช้งานบนเครื่องไมโครคอมพิวเตอร์ซึ่งทำหน้าที่ในการเชื่อมต่อระหว่างผู้ใช้กับอินเซอร์กิตอีมูเลเตอร์ ในการออกแบบโปรแกรมมอนิเตอร์ของอินเซอร์กิตอีมูเลเตอร์ได้เลือกใช้ภาษาแอสแซมบลี (Assembly) ซึ่งพัฒนาด้วยโปรแกรม ASM96 assembler เนื่องจากเป็นภาษาที่นิยมใช้งานกันอย่างแพร่หลาย และมีความรู้พื้นฐานการพัฒนาด้วยโปรแกรม ASM51 assembler อยู่แล้ว สำหรับการออกแบบโปรแกรมการใช้งานบนเครื่องไมโครคอมพิวเตอรนั้น เลือกใช้ภาษาเบสิก (Basic) โดยในขั้นแรกพัฒนาด้วยโปรแกรม QBASIC เพราะมีชุดคำสั่งที่เพียงพอต่อความต้องการอีกทั้งยังง่าย และสะดวกในการพัฒนา เมื่อโปรแกรมการใช้งานบนเครื่องไมโครคอมพิวเตอร์สามารถทำการเชื่อมต่อระหว่างผู้ใช้กับอินเซอร์กิตอีมูเลเตอร์ได้สมบูรณ์แล้ว จึงพัฒนาต่อด้วยโปรแกรม Visual Basic V.4 เพื่อเพิ่มความสะดวกและความสามารถในการใช้งาน เนื่องจากโปรแกรมนี้ทำงานภายใต้ระบบปฏิบัติการ Windows ที่เป็นการใช้งานแบบ Graphic User Interface (GUI)

โปรแกรมมอนิเตอร์ และโปรแกรมการใช้งานบนเครื่องไมโครคอมพิวเตอร์ที่ออกแบบและพัฒนาไว้สามารถรองรับคำสั่งต่างๆ ได้ 4 กลุ่มคือ กลุ่มคำสั่งในโหมดอิดิตเตอร์ (Editer), กลุ่มคำสั่งในโหมดแอสแซมเบลอร์ (Assembler), กลุ่มคำสั่งในโหมดดีบักเกอร์ (Debuger), กลุ่มคำสั่งในโหมดทั่วไป แยกอธิบายได้ดังนี้

กลุ่มคำสั่งในโหมดอิดิตเตอร์ ได้แก่

คำสั่ง **New** ใช้สำหรับเปิดแฟ้มข้อมูล เมื่อต้องการพัฒนาโปรแกรมยูสเซอร์โปรแกรมใหม่

คำสั่ง **Open** ใช้สำหรับเปิดแฟ้มข้อมูลของโปรแกรมยูสเซอร์ที่มีอยู่แล้ว

คำสั่ง **Open List File** ใช้สำหรับ เปิดแฟ้มข้อมูลแบบ List ของโปรแกรมยูสเซอร์ที่กำลังพัฒนาอยู่ในขณะนั้น ซึ่งผ่านการแอสแซมเบลอร์ (Assembler) แล้ว

คำสั่ง **Save** ใช้ในการเก็บบันทึกแฟ้มข้อมูลของโปรแกรมยูสเซอร์ที่กำลังพัฒนาอยู่ในขณะนั้นจากหน่วยความจำของเครื่องไมโครคอมพิวเตอร์ลงในหน่วยเก็บข้อมูลแบบแข็ง (Hard disk หรือ Floppy Disk)

คำสั่ง **Save As** มีหน้าที่เหมือนกับคำสั่ง Save แต่สามารถระบุชื่อแฟ้มข้อมูลเป็นชื่อใหม่ได้

คำสั่ง **Exit** ใช้สำหรับออกจากโปรแกรมการใช้งานบนเครื่องไมโครคอมพิวเตอร์

กลุ่มคำสั่งในโหมดแอสแซมเบลอร์ ได้แก่

คำสั่ง **ASM96** ใช้สำหรับการแอสแซมเบลอร์แฟ้มข้อมูลของโปรแกรมยูสเซอร์ที่กำลังพัฒนาอยู่ในขณะนั้นจาก Source Program ไปเป็น Object Program

คำสั่ง **OH** ใช้สำหรับเปลี่ยน Object Program ของโปรแกรมยูสเซอร์ที่กำลังพัฒนาอยู่ในขณะนั้นไปเป็น Intel HEX file

คำสั่ง **Down Load** ใช้ในการส่งข้อมูล Intel HEX file ของโปรแกรมยูสเซอร์ที่กำลังพัฒนาอยู่ในขณะนั้น ผ่านพอร์ตอนุกรมไปเก็บไว้ในหน่วยความจำข้อมูลบนฮาร์ดแวร์ของอินเซอร์กิตอีมูเลเตอร์

คำสั่ง **Auto** จะทำงานตามคำสั่ง ASM96, OH และ Down Load ตามลำดับโดยอัตโนมัติ

กลุ่มคำสั่งในโหมดดีบักเกอร์ ได้แก่

คำสั่ง **Data, Registers** ใช้สำหรับเปิดหรือปิดหน้าต่างที่แสดงข้อมูลในหน่วยความจำข้อมูล และรีจิสเตอร์

คำสั่ง **Edit** ใช้สำหรับแก้ไขหรือเปลี่ยนแปลงข้อมูลในหน่วยความจำข้อมูลทีละหนึ่งแอดเดรส

คำสั่ง **Fill** ใช้สำหรับแก้ไขหรือเปลี่ยนแปลงข้อมูลในหน่วยความจำข้อมูล ระหว่างช่วงแอดเดรส

คำสั่ง **Move** ใช้สำหรับคัดลอกหรือเคลื่อนย้ายข้อมูลในหน่วยความจำข้อมูลจากแอดเดรสหนึ่งไปยังอีกแอดเดรสหนึ่ง

คำสั่ง **Dump** ใช้สำหรับแสดงข้อมูลในหน่วยความจำข้อมูล

คำสั่ง **Change Register** ใช้สำหรับแก้ไขหรือเปลี่ยนแปลงค่าของรีจิสเตอร์ได้แก่ ตัวนับโปรแกรม (Program Counter : PC), ตัวชี้สแตก (Stack Pointer : SP) และรีจิสเตอร์คำแสดงสถานะโปรแกรม (Program Status Word : PSW)

คำสั่ง **Go** อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์ โดยใช้ค่าของตัวนับโปรแกรม และเบรคพอยบัฟเฟอร์ ที่มีอยู่ในขณะนั้น

คำสั่ง **Go forever** อินเซอร์กิตอีมูเลเตอร์จะทำการเคลียร์ค่าของเบรคพอยบัฟเฟอร์ และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์ โดยใช้ค่าของตัวนับโปรแกรมที่มีอยู่ในขณะนั้น

คำสั่ง **Go from <code_address>** อินเซอร์กิตอีมูเลเตอร์จะทำการโหลดค่าของตัวนับโปรแกรมให้มีค่าเท่ากับ <code_address> และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์ โดยใช้ค่าของเบรคพอยบัฟเฟอร์ที่มีอยู่ในขณะนั้น

คำสั่ง **Go from <code_address> forever** อินเซอร์กิตอีมูเลเตอร์จะทำการโหลดค่าของตัวนับโปรแกรมให้มีค่าเท่ากับ <code_address>, ทำการเคลียร์ค่าของเบรคพอยบัฟเฟอร์ และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์

คำสั่ง **Go from <code_address1> till <code_address2>** อินเซอร์กิตอีมูเลเตอร์จะทำการโหลดค่าของตัวนับโปรแกรมให้มีค่าเท่ากับ <code_address1>, ทำการเซตเบรคพอยบัฟเฟอร์ลำดับที่ 15 ให้มีค่าเท่ากับ <code_address2> และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์

คำสั่ง **Go from <code_address1> till <code_address2> or <code_address3>** คำสั่งนี้อินเซอร์กิตอีมูเลเตอร์จะทำงานเหมือนกับคำสั่งก่อนหน้านี้ แต่จะทำการเซตเบรคพอยบัฟเฟอร์ลำดับที่ 14 ให้มีค่าเท่ากับ <code_address3>

คำสั่ง **Setup** (Break Point) ใช้สำหรับแสดงค่า หรือทำการเปลี่ยนแปลงค่าของเบรคพอยบัฟเฟอร์ที่มีอยู่ในขณะนั้น

คำสั่ง **Disable All** (Break Point) ใช้เมื่อต้องการยกเลิกค่าเบรคพอยบัฟเฟอร์ทั้งหมดที่มีอยู่ในขณะนั้นชั่วขณะ และสามารถเรียกค่าเบรคพอยบัฟเฟอร์มาใช้งานได้อีกในภายหลัง

คำสั่ง **Deleate All** (Break Point) ใช้สำหรับลบค่าเบรคพอยบัฟเฟอร์ทั้งหมดที่มีอยู่ในขณะนั้น โดยไม่สามารถเรียกค่าเบรคพอยบัฟเฟอร์มาใช้งานได้อีก

กลุ่มคำสั่งในโหมดทั่วไป ได้แก่

คำสั่ง **Config** ใช้สำหรับกำหนดข้อมูลที่สำคัญของโปรแกรมการใช้งานบนเครื่องไมโครคอมพิวเตอร์ได้แก่ หมายเลขของพอร์ตอนุกรมที่ต่อใช้งาน, แอดเดรสเริ่มต้นของหน่วยความจำข้อมูลที่ต้องการให้แสดงค่า และตำแหน่งที่อยู่ของโปรแกรมยูสเซอร์

คำสั่ง **Reset** ใช้สำหรับรีเซตฮาร์ดแวร์และซอฟแวร์ของอินเซอร์กิตอีมูเลเตอร์ เพื่อเริ่มต้นทำงานใหม่ในกรณีที่เกิดข้อผิดพลาดขึ้น

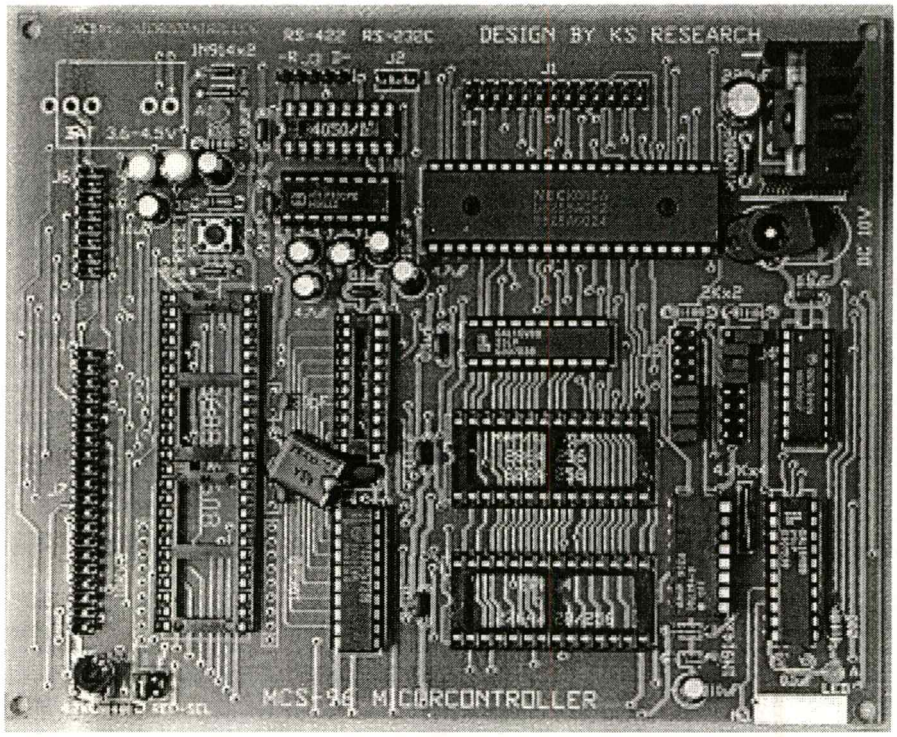
บทที่ 4

การทดลองใช้งาน

4.1 รายละเอียดทางด้านฮาร์ดแวร์ของบอร์ดเป้าหมายที่นำมาต่อใช้งาน

ในการทดลองใช้งานนั้น ได้นำเอาอินเซอร์กิตอีมูเลเตอร์ต่อร่วมกับบอร์ดเป้าหมาย "KS-96 BOARD MCS-96 MICROCONTROLLER" ซึ่งพัฒนาขึ้นโดยบริษัทไลท์แอนด์ซาวด์ ดีไซน์จำกัด มีรายละเอียดทางด้านฮาร์ดแวร์ดังนี้

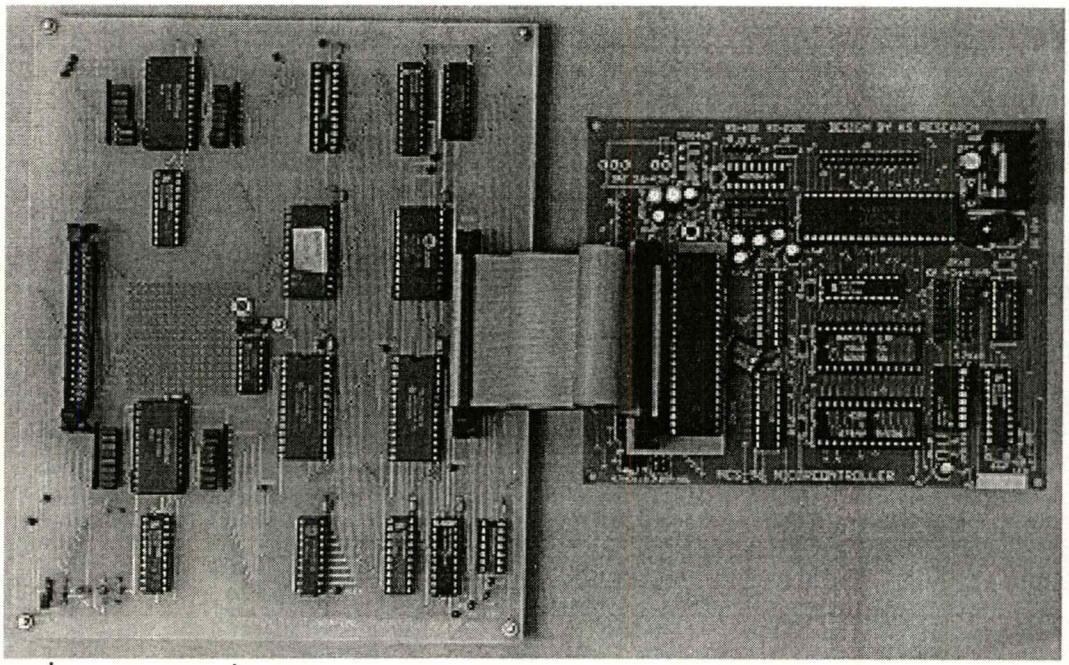
- ไมโครคอนโทรลเลอร์	8098 (ตัวถังแบบ DIP-48)
- ความถี่ของสัญญาณนาฬิกา	12 เมกะเฮิรตซ์
- หน่วยความจำแบบอีพีรอม	8, 16, 32 กิโลไบต์ (2764, 27128, 27256)
- หน่วยความจำแบบแรม	8, 32 กิโลไบต์ (6264, 62256)
- อินพุต/เอาท์พุตพอร์ต	8255
- พอร์ตอนุกรม	RS-232, RS-422
- แรงจ่ายไฟกระแสตรง	10 โวลท์



ภาพที่ 4.1 แสดง KS-96 BOARD MCS-96 MICROCONTROLLER

4.2 การเชื่อมต่อระหว่างอินเซอร์กิตอีมูเลเตอร์และบอร์ดเป้าหมาย

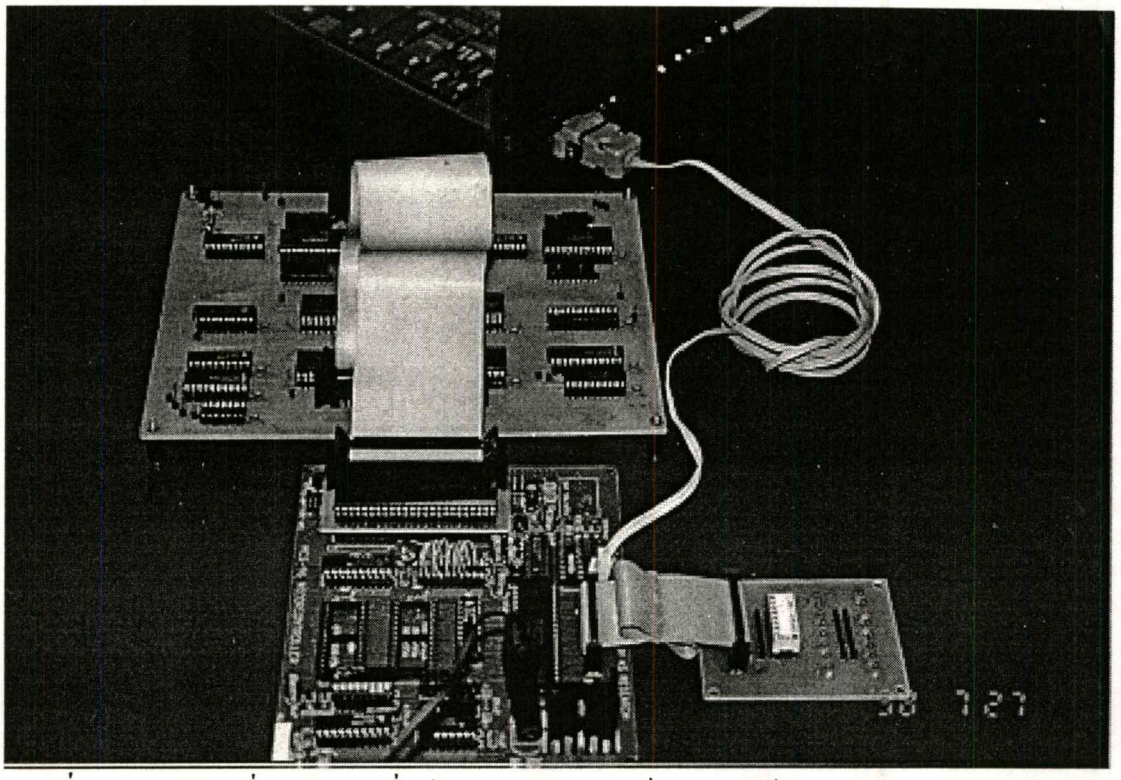
การเชื่อมต่อสามารถทำได้โดยการถอดซีพียู 8098 ของบอร์ดเป้าหมายออกแล้วนำพ็อดของอินเซอร์กิตอีมูเลเตอร์ติดตั้งลงบนซ็อคเค็ทไอซีที่ว่างอยู่ จากนั้นจึงติดตั้งซีพียู 8098 ลงบนซ็อคเค็ทไอซีของพ็อดอีกที สำหรับหน่วยความจำแบบอีพีรอมและแบบแรมของบอร์ดเป้าหมายนั้น จะต้องถอดออกเพราะในการพัฒนาโปรแกรมด้วยอินเซอร์กิตอีมูเลเตอร์นั้น ข้อมูลและโปรแกรมของผู้ใช้จะเก็บไว้ในหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์



ภาพที่ 4.2 แสดงการเชื่อมต่อระหว่างอินเซอร์กิตอีมูเลเตอร์และบอร์ดเป้าหมาย

4.3 การเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์

การเชื่อมต่อสามารถทำได้โดยใช้สายสัญญาณของพอร์ตอนุกรม RS-232 โดยต่อปลายด้านที่เป็นคอนเน็กเตอร์แบบ DB-9 เข้ากับพอร์ตอนุกรมหมายเลข 1 ของเครื่องไมโครคอมพิวเตอร์ (ในกรณีที่เป็นคอนเน็กเตอร์แบบ DB-25 จะต่อเข้ากับพอร์ตอนุกรมหมายเลข 2) ส่วนปลายอีกด้านที่เป็นคอนเน็กเตอร์ตัวเมียแบบ 3 ขานั้นจะต่อเข้ากับคอนเน็กเตอร์ตัวผู้แบบ 3 ขาซึ่งอยู่บนบอร์ดเป้าหมาย (พอร์ตอนุกรม RS-232)



ภาพที่ 4.3 แสดงการเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์ โดยการใช้สายสัญญาณของพอร์ตอนุกรม RS-232

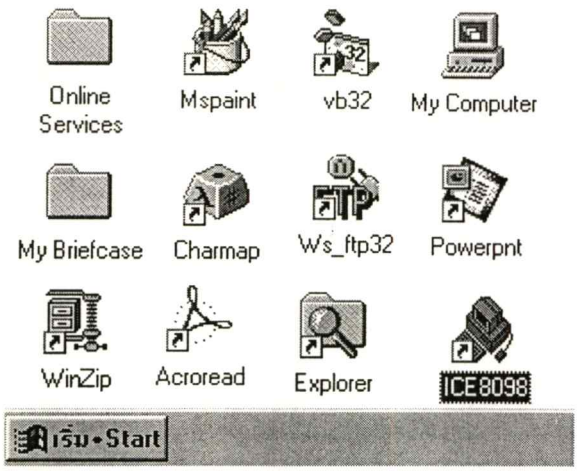
4.4 การเตรียมพร้อมอินเซอร์กิตอีมูเลเตอร์ก่อนการใช้งาน

เมื่อทำการติดตั้งอินเซอร์กิตอีมูเลเตอร์เข้ากับบอร์ดเป้าหมาย และเชื่อมต่อเข้ากับเครื่องไมโครคอมพิวเตอร์เสร็จเรียบร้อยแล้ว ขั้นตอนต่อไปคือการเตรียมพร้อมอินเซอร์กิตอีมูเลเตอร์ก่อนการใช้งานซึ่งอธิบายได้เป็นลำดับดังนี้

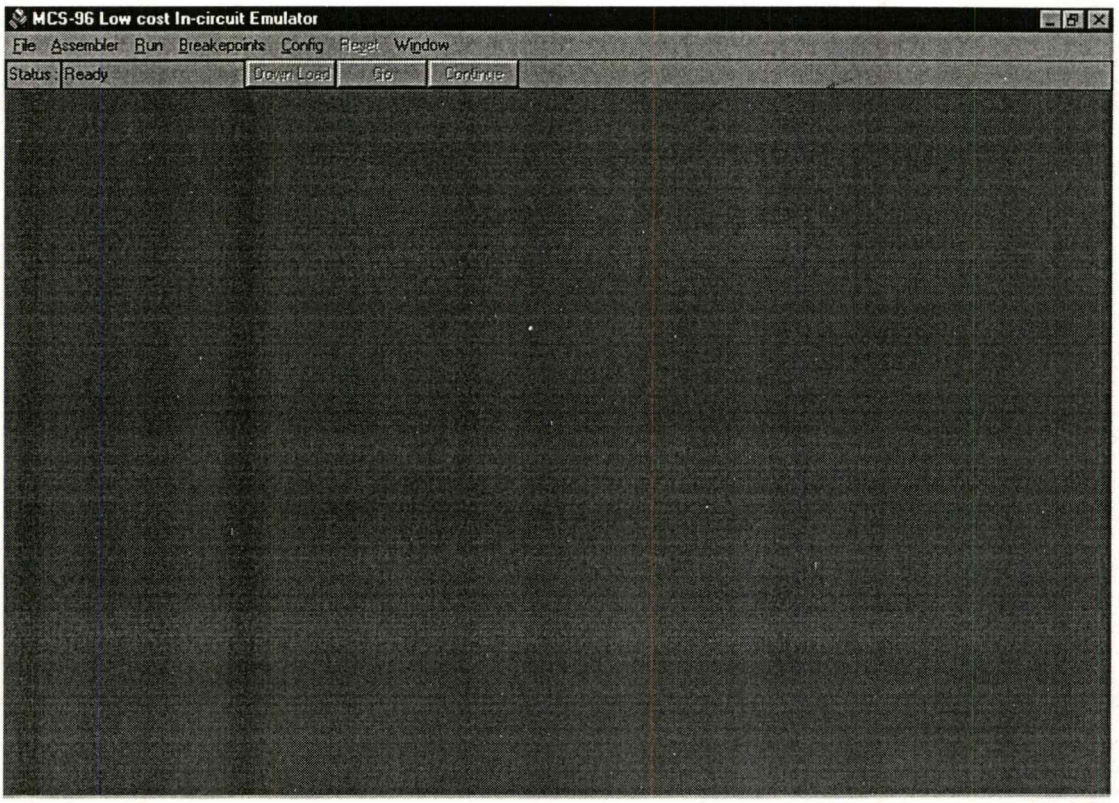
1. ต่อแหล่งจ่ายแรงดันไฟฟ้ากระแสตรง 10 โวลท์ให้กับบอร์ดเป้าหมาย

2. กดสวิทช์ รีเซตของอินเซอร์กิตอีมูเลเตอร์

3. ภายใต้ระบบปฏิบัติการ WINDOWS95 ทำการเรียกใช้งานโปรแกรม ICE8089 โดยคลิกเมาส์ที่ ICON ของโปรแกรม ICE8098 2 ครั้ง



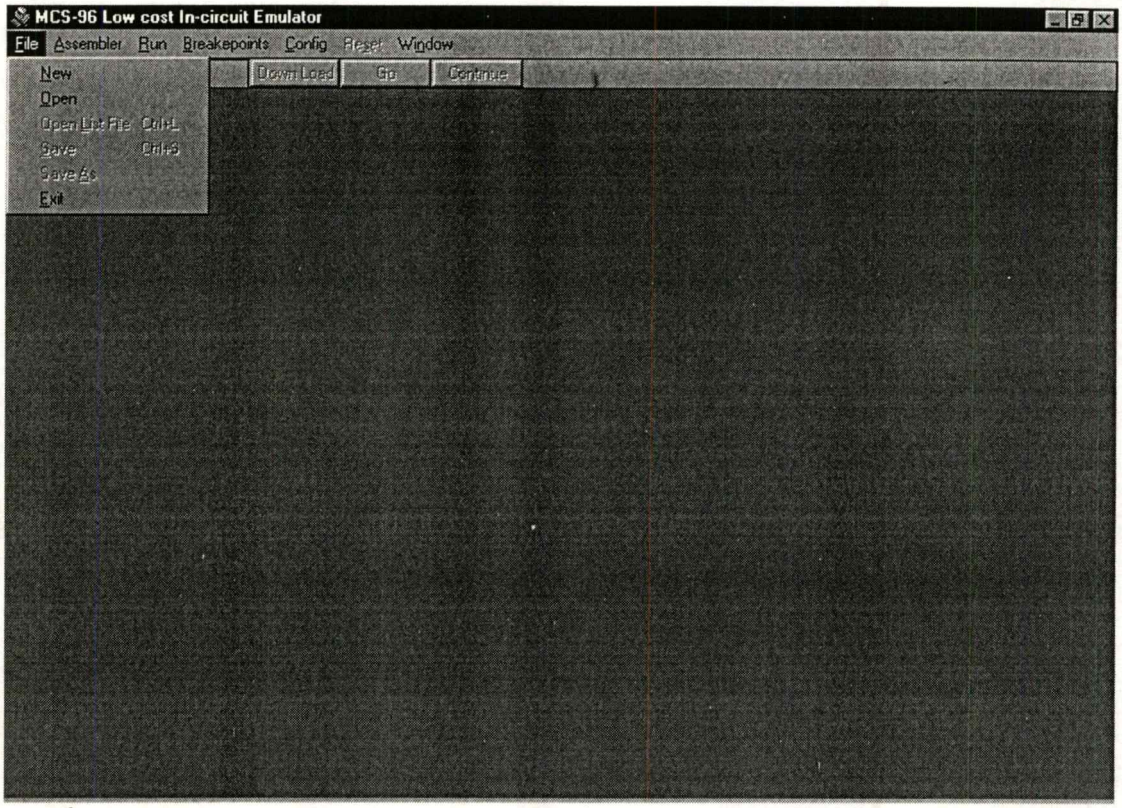
ภาพที่ 4.4 แสดงการเรียกใช้งานโปรแกรม ICE8089 ภายใต้ระบบปฏิบัติการ WINDOWS95



ภาพที่ 4.5 แสดงหน้าต่างหลักของโปรแกรม ICE8098

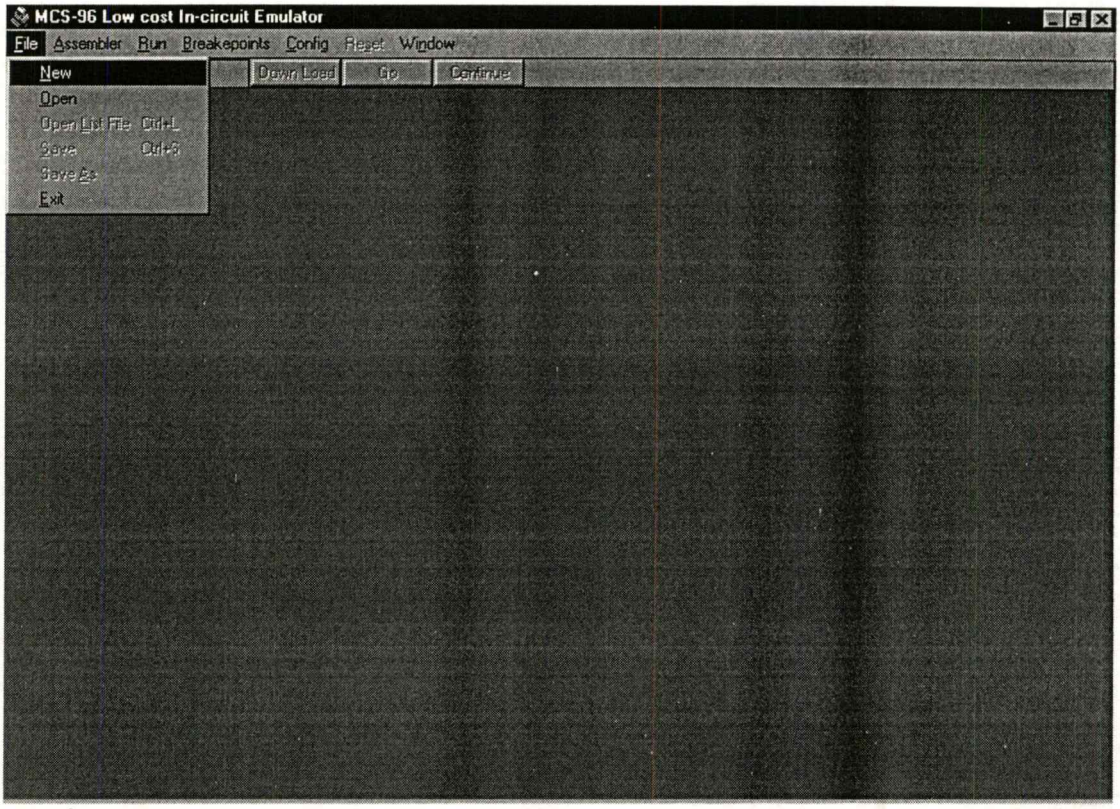
4.5 การทดลองใช้งานคำสั่งต่างๆ

เมนู File ภายในประกอบด้วยคำสั่งต่างๆ จำนวน 6 คำสั่งแยกอธิบายได้ดังนี้

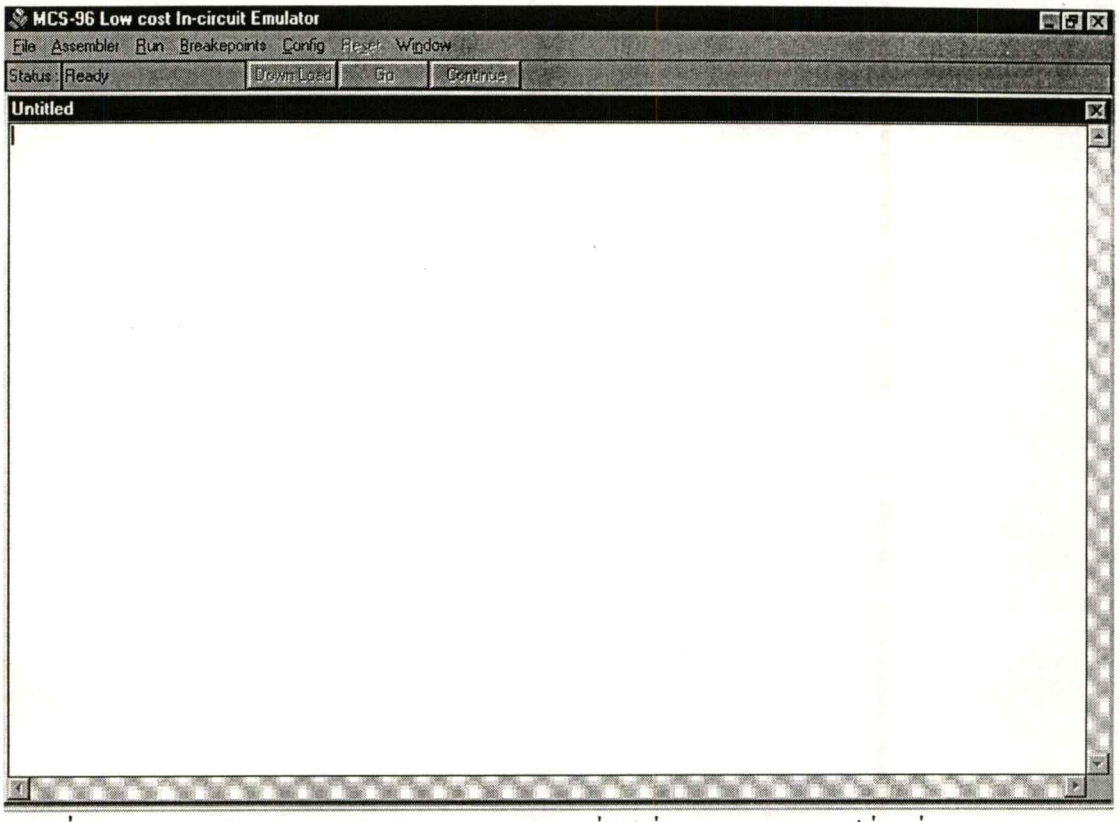


ภาพที่ 4.6 แสดงเมนู File และส่วนประกอบของคำสั่งภายใน

4.5.1 คำสั่ง New ใช้สำหรับเปิดแฟ้มข้อมูล ในกรณีที่ต้องการเริ่มพัฒนาโปรแกรมใหม่ เมื่อคลิกเมาส์ที่คำสั่งนี้โปรแกรมจะสร้างหน้าต่างสำหรับให้ผู้ใช้งานเขียนโปรแกรมที่ต้องการพัฒนาขึ้นมา โดยกำหนดชื่อของแฟ้มข้อมูลนี้เป็น “Untitled”

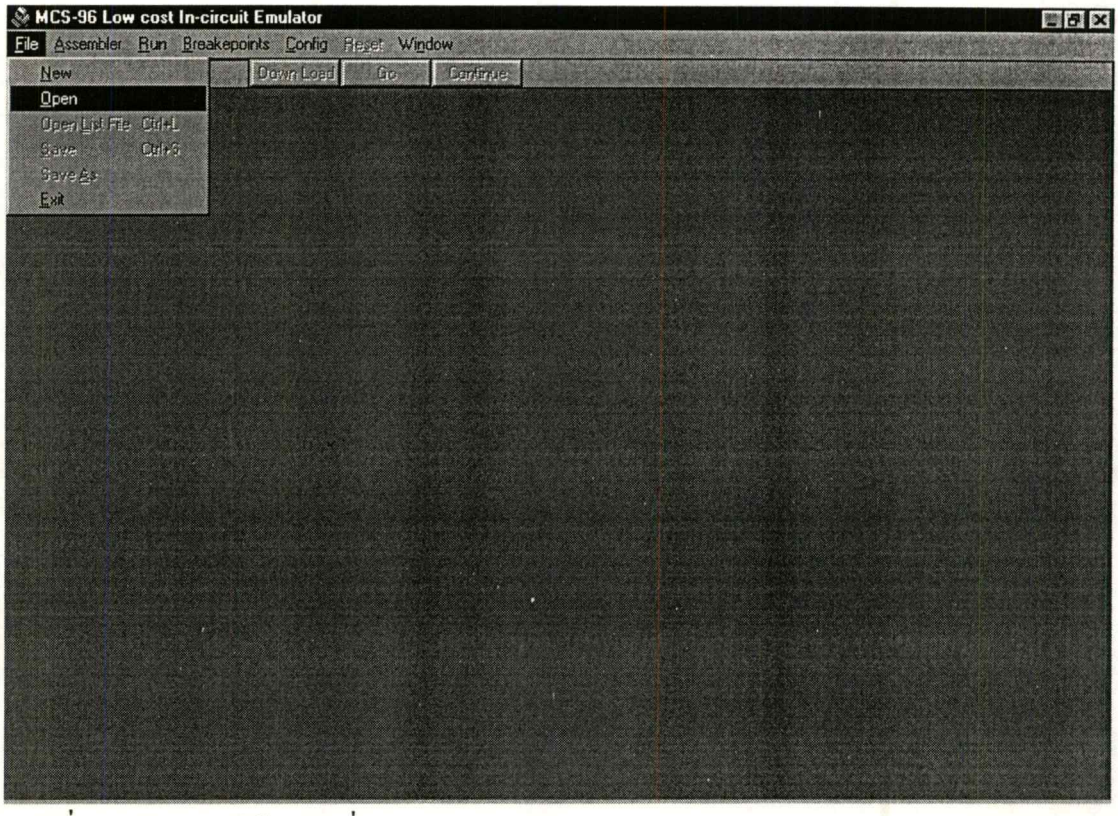


ภาพที่ 4.7 แสดงการใช้งานคำสั่ง New

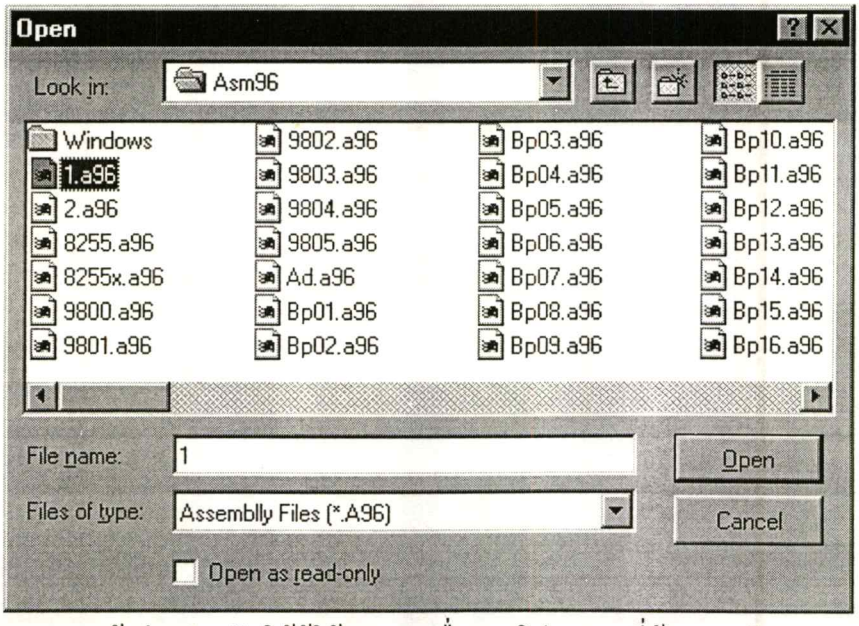


ภาพที่ 4.8 แสดงหน้าต่างของแฟ้มข้อมูล “Untitled” ที่ได้เมื่อทำการคลิกเมาส์ที่คำสั่ง New

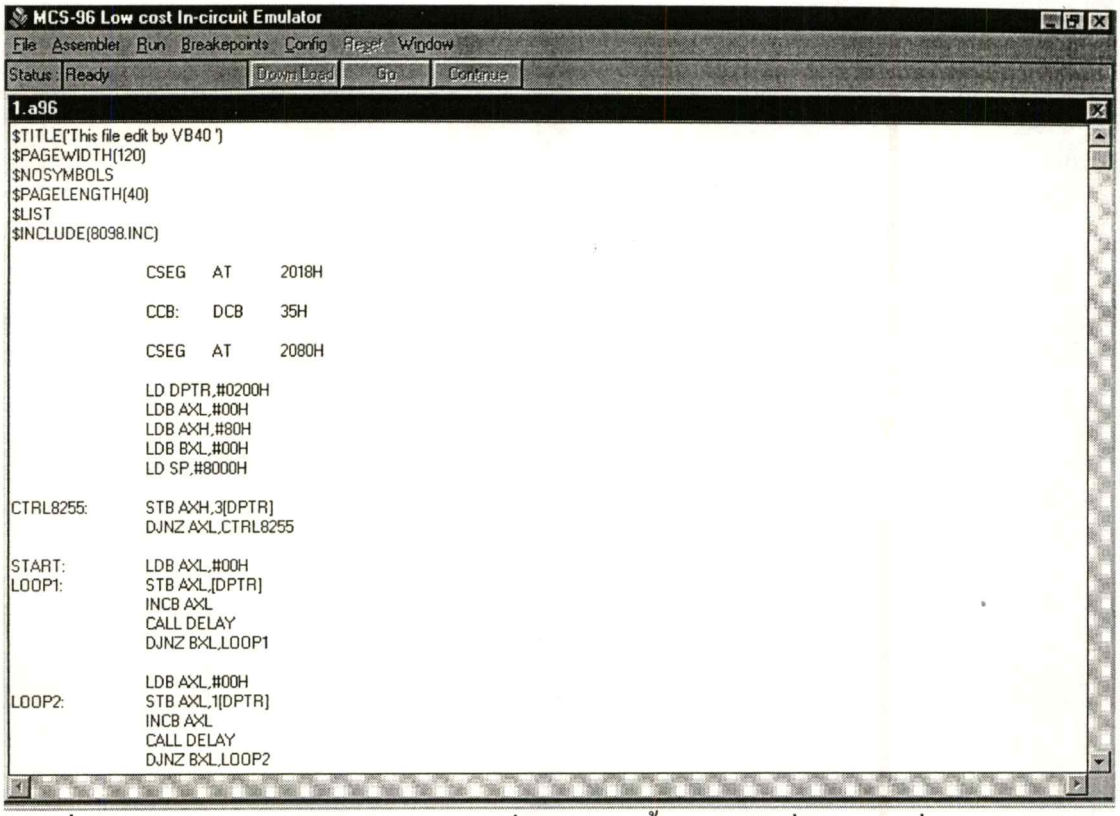
4.5.2 คำสั่ง Open ใช้สำหรับเปิดแฟ้มข้อมูลเก่าที่มีอยู่แล้ว ในกรณีที่ต้องการพัฒนาโปรแกรมนั้นอีกครั้ง เมื่อคลิกเมาส์ที่คำสั่งนี้โปรแกรมจะเปิดหน้าต่างสำหรับให้ผู้ใช้งานระบุชื่อของแฟ้มข้อมูลที่ต้องการ และจะอ่านข้อมูลนั้นขึ้นมาเพื่อให้ผู้ใช้ทำการพัฒนาโปรแกรมต่อไป



ภาพที่ 4.9 แสดงการใช้งานคำสั่ง OPEN

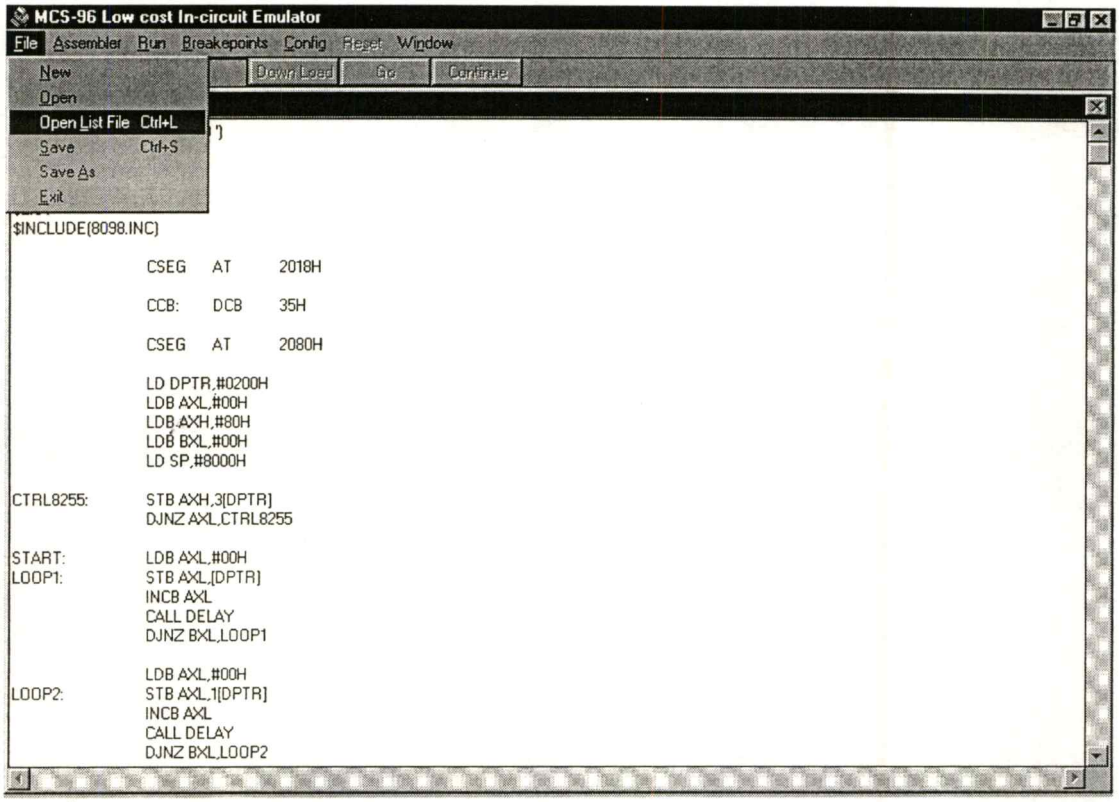


ภาพที่ 4.10 แสดงหน้าต่างสำหรับให้ผู้ใช้งานระบุชื่อของโปรแกรมที่ต้องการพัฒนา

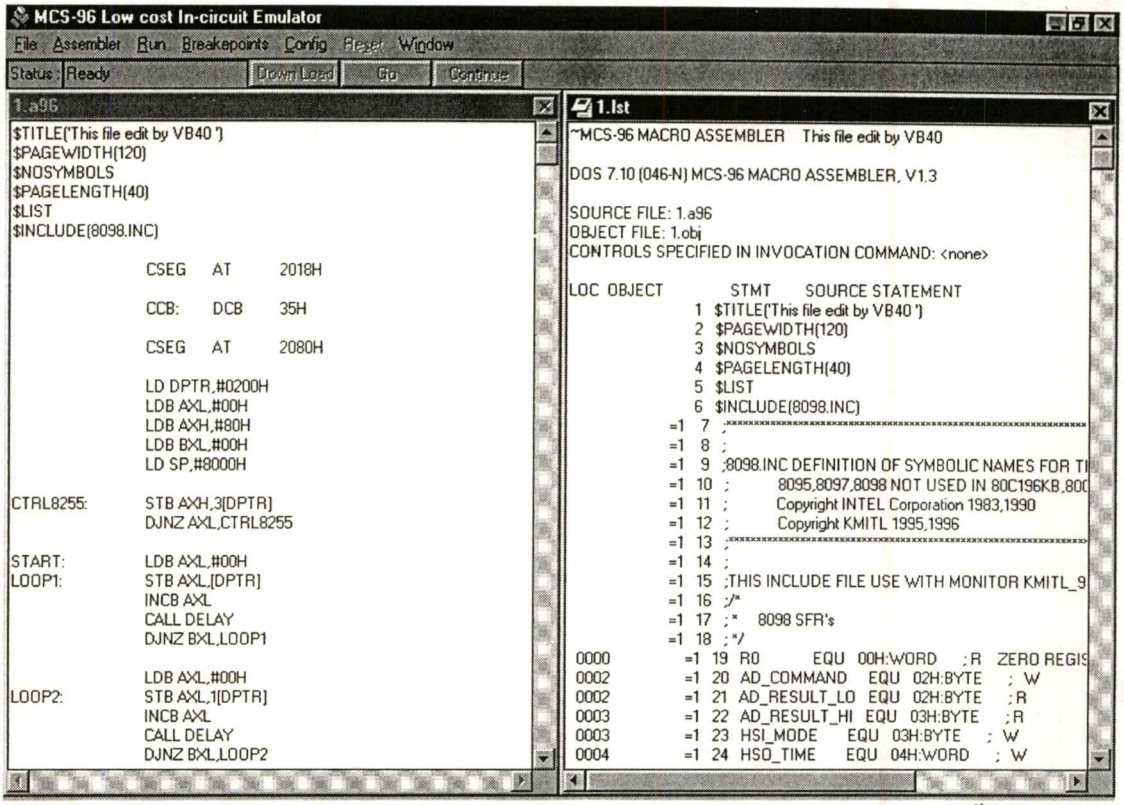


ภาพที่ 4.11 แสดงตัวอย่างของแฟ้มข้อมูลเก่าที่ทำการเปิดขึ้นมาด้วยคำสั่ง OPEN เพื่อทำการพัฒนา

4.5.3 คำสั่ง Open List File ใช้สำหรับเปิดแฟ้มข้อมูลแบบ List ที่ได้จากการแอสแซมเบลอร์โปรแกรมที่กำลังพัฒนาอยู่ในขณะนั้น เพื่อทำการตรวจสอบรายละเอียดต่างๆ ภายในโปรแกรม ตลอดจนข้อผิดพลาดที่อาจเกิดขึ้นได้

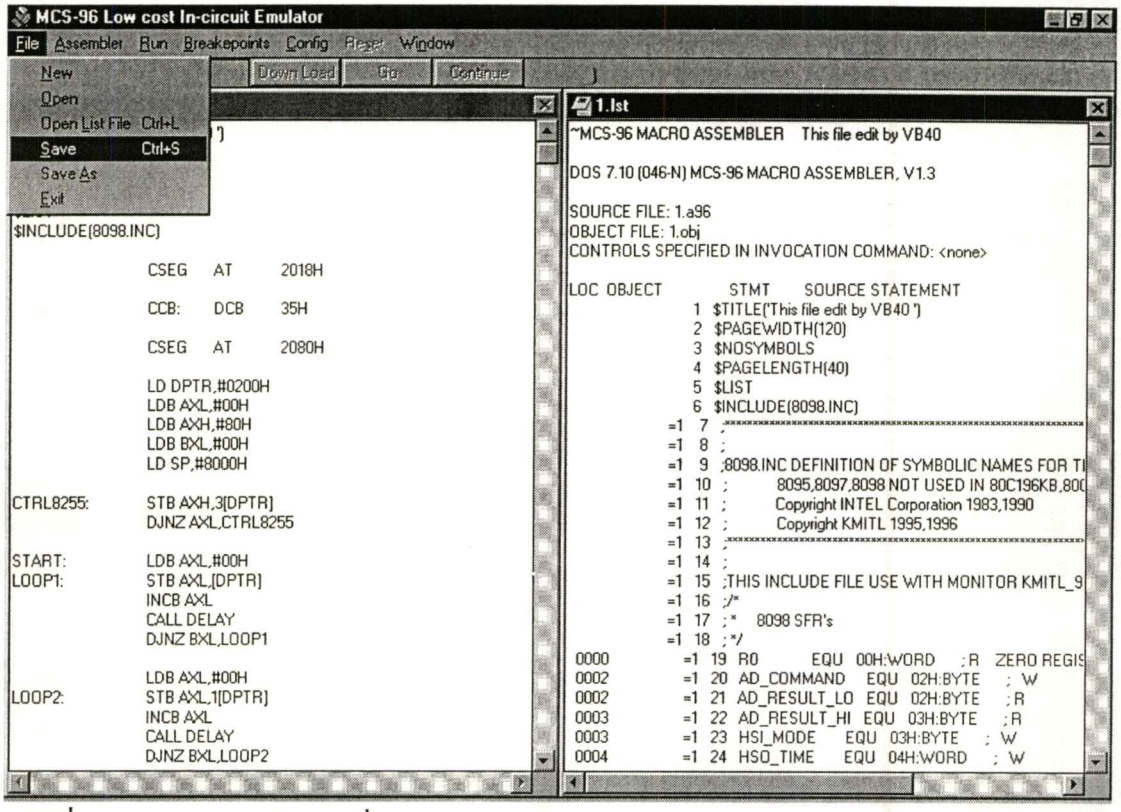


ภาพที่ 4.12 แสดงการใช้งานคำสั่ง Open List File



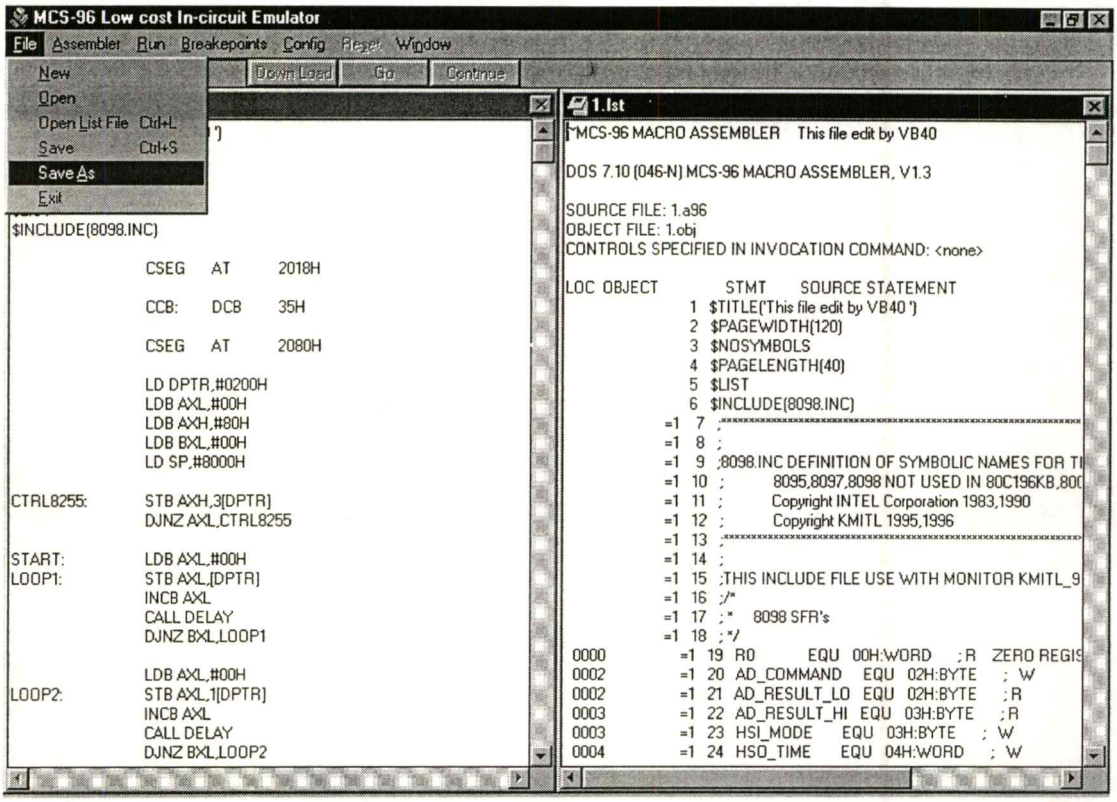
ภาพที่ 4.13 แสดงตัวอย่างแฟ้มข้อมูลแบบ List ของโปรแกรมที่กำลังพัฒนาอยู่ในขณะนั้น

4.5.4 คำสั่ง Save ใช้สำหรับบันทึกแฟ้มข้อมูลของโปรแกรมที่กำลังพัฒนาอยู่ในขณะนั้น โดยจะใช้ชื่อเดิมของแฟ้มข้อมูลในการบันทึก

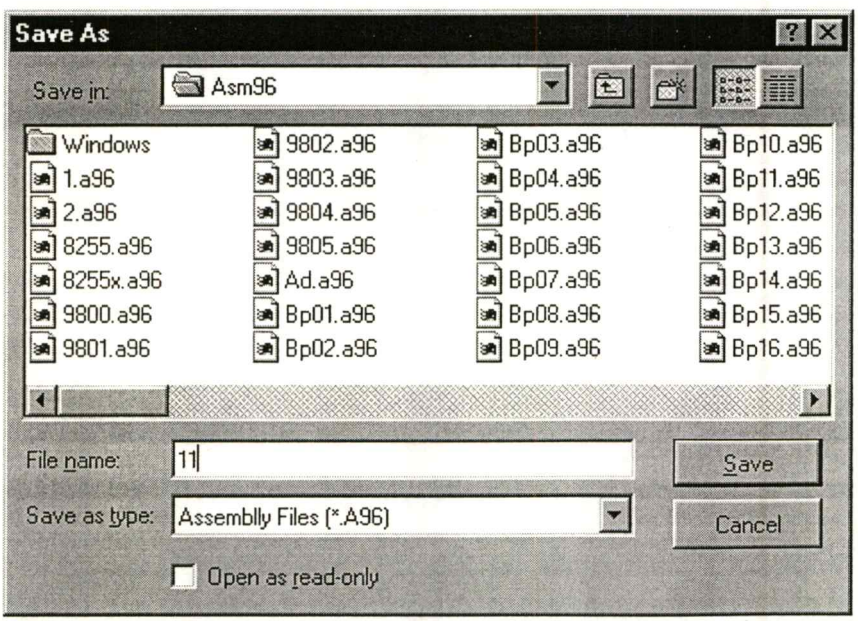


ภาพที่ 4.14 แสดงการใช้งานคำสั่ง Save

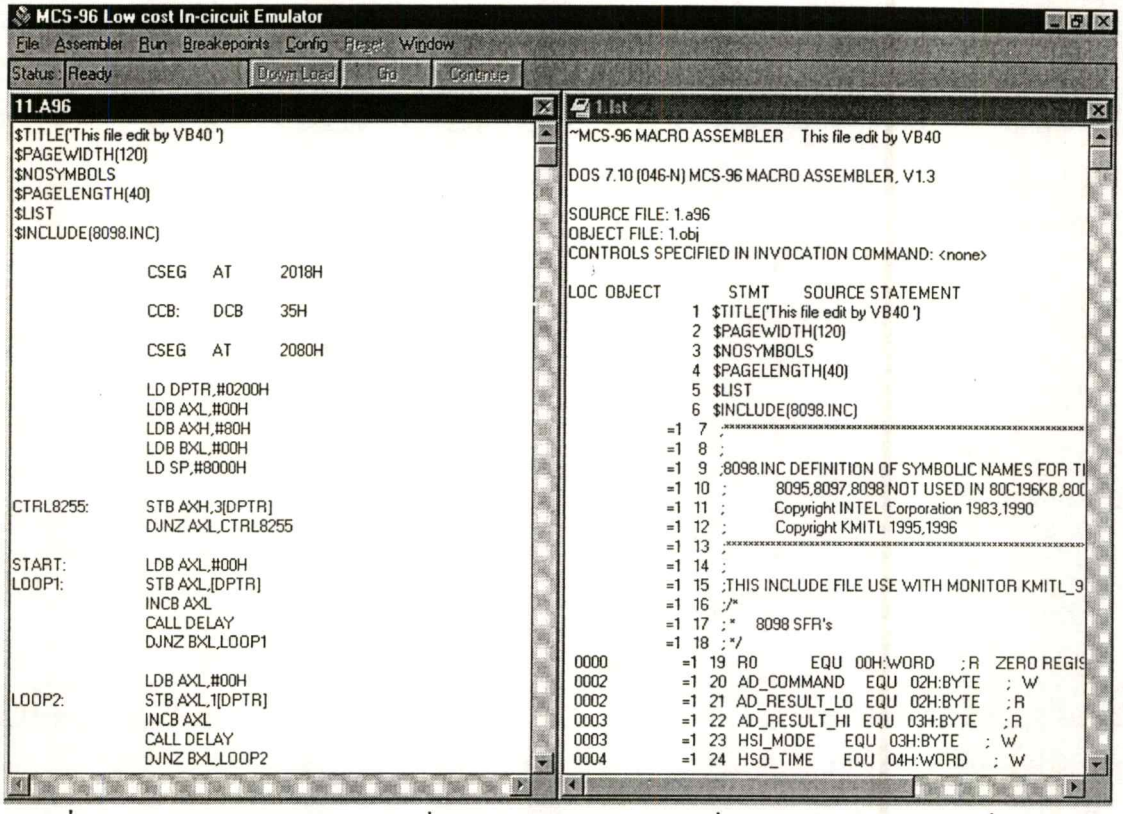
4.5.5 คำสั่ง Save Asใช้สำหรับบันทึกแฟ้มข้อมูลของโปรแกรมที่กำลังพัฒนาอยู่ในขณะนั้น โดยผู้ใช้สามารถระบุชื่อของแฟ้มข้อมูลในการบันทึกได้



ภาพที่ 4.15 แสดงการใช้งานคำสั่ง Save As

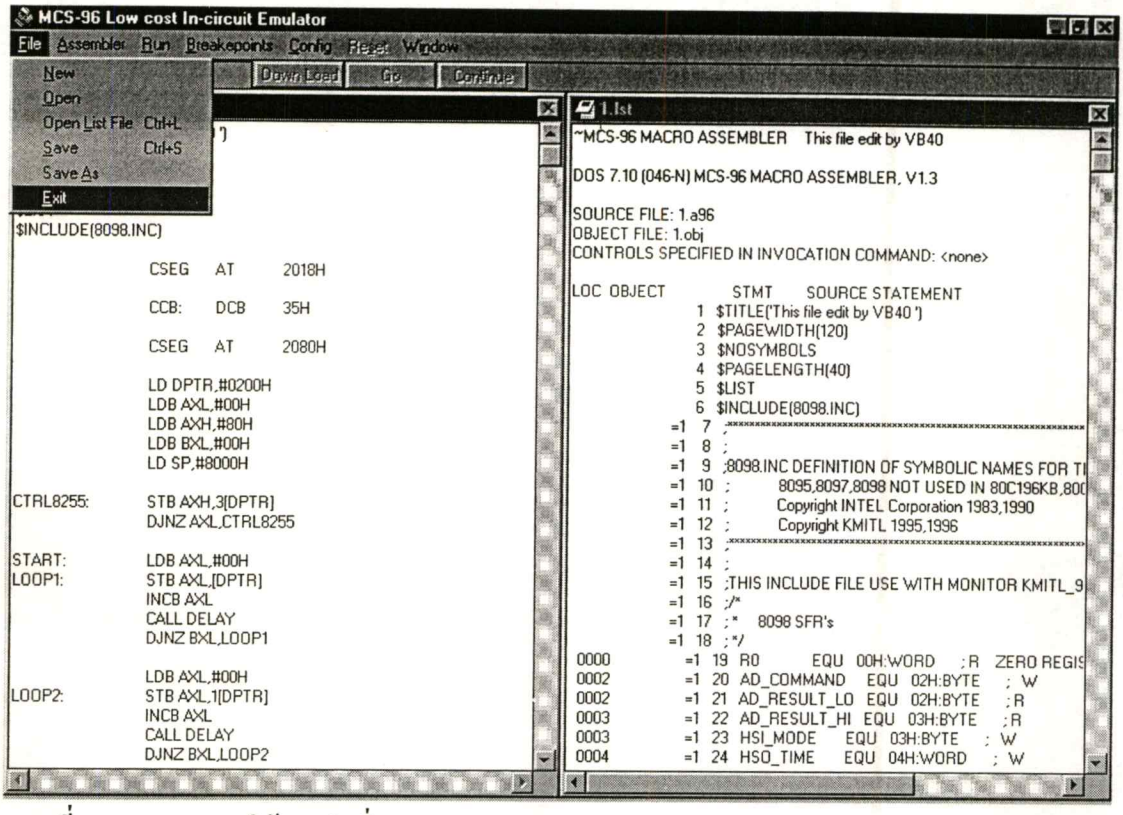


ภาพที่ 4.16 แสดงหน้าต่างที่ให้ผู้ใช้ระบุชื่อใหม่สำหรับการบันทึกแฟ้มข้อมูลที่กำลังพัฒนาอยู่ในขณะนั้น



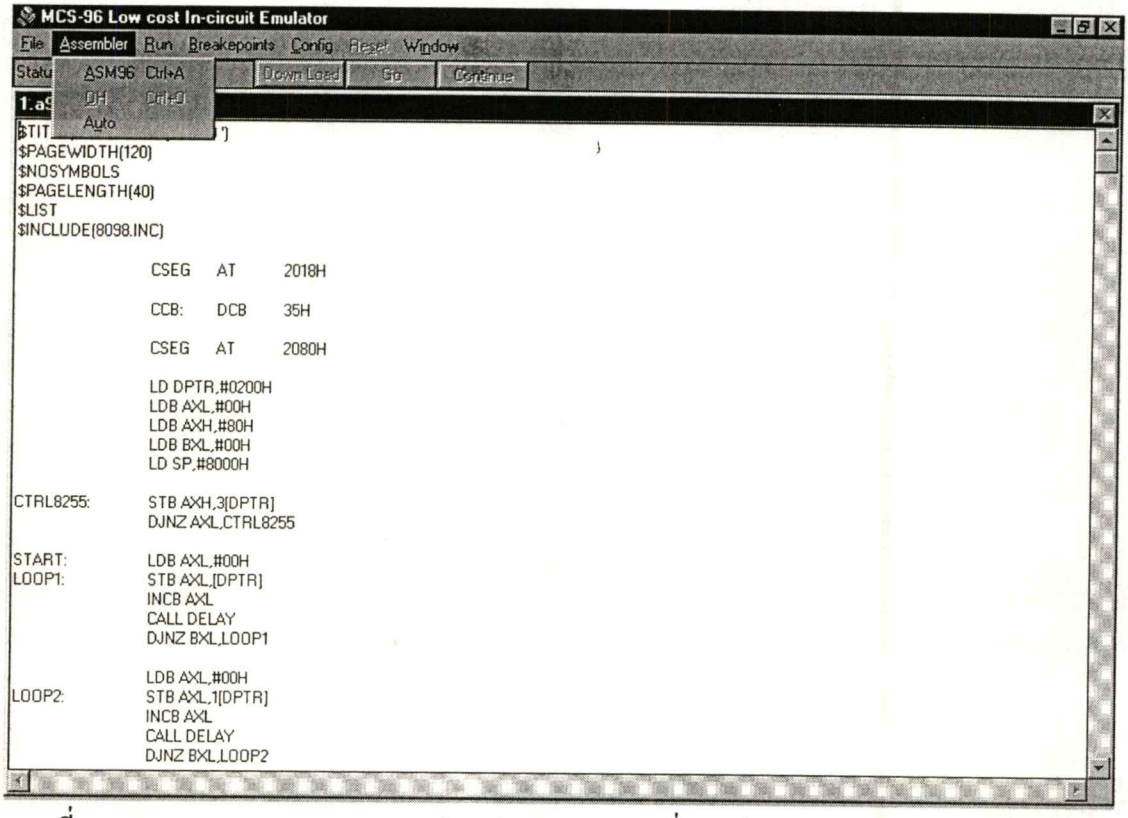
ภาพที่ 4.17 แสดงตัวอย่างแฟ้มข้อมูลที่ได้จากการบันทึกด้วยคำสั่ง Save As จะเห็นได้ว่าชื่อของแฟ้มข้อมูลจะเป็นชื่อใหม่ตามที่ระบุไว้

4.5.6 คำสั่ง Exit ใช้สำหรับออกจากโปรแกรม ICE8098 เมื่อสิ้นสุดการพัฒนาโปรแกรม



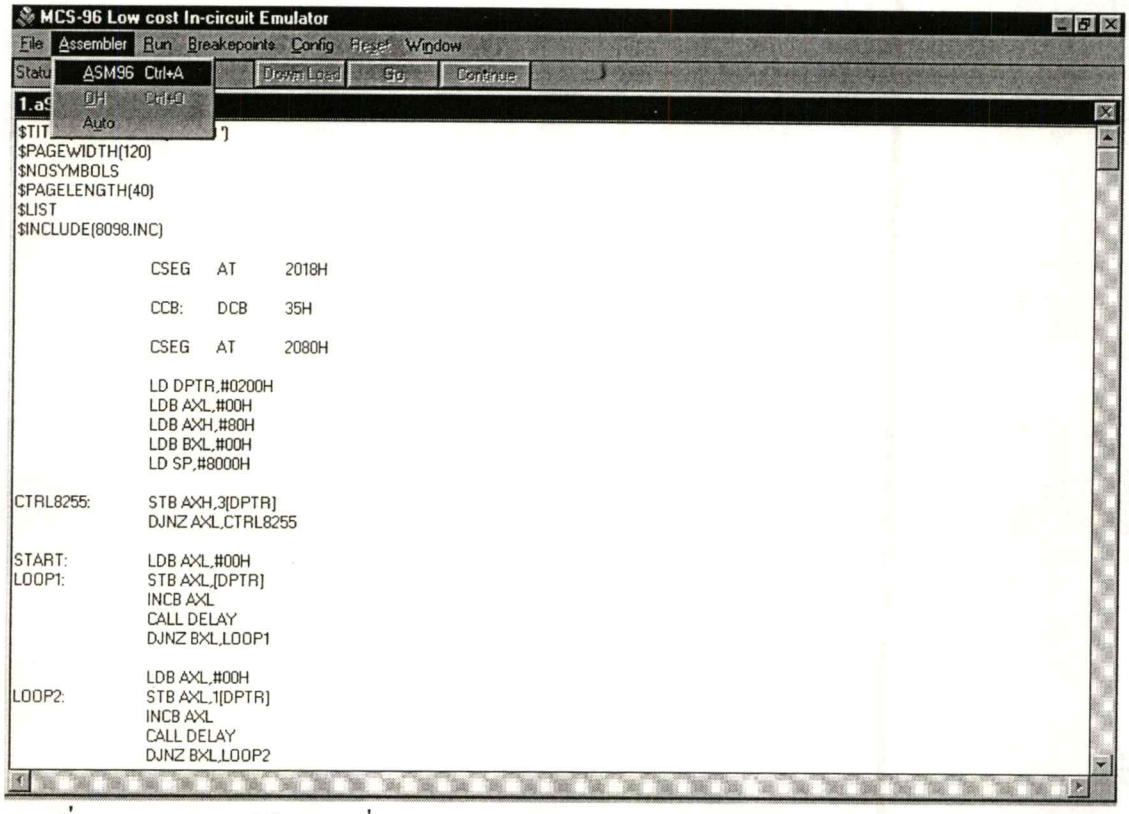
ภาพที่ 4.18 แสดงการใช้งานคำสั่ง Exit

เมนู Assembler ภายในประกอบด้วยคำสั่งต่างๆ จำนวน 3 คำสั่งแยกอธิบายได้ดังนี้

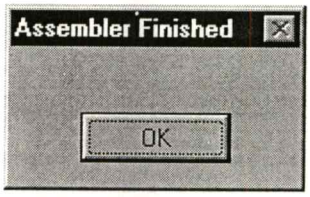


ภาพที่ 4.19 แสดงเมนู Assembler และส่วนประกอบของคำสั่งภายใน

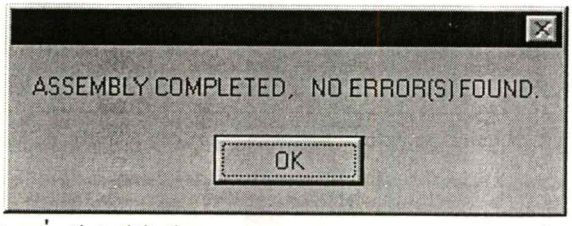
4.5.7 คำสั่ง ASM96 ใช้ในการแอสแซมเบลอร์โปรแกรมที่กำลังพัฒนาอยู่ให้เป็นแฟ้มข้อมูลแบบ Object File



ภาพที่ 4.20 แสดงการใช้งานคำสั่ง ASM96

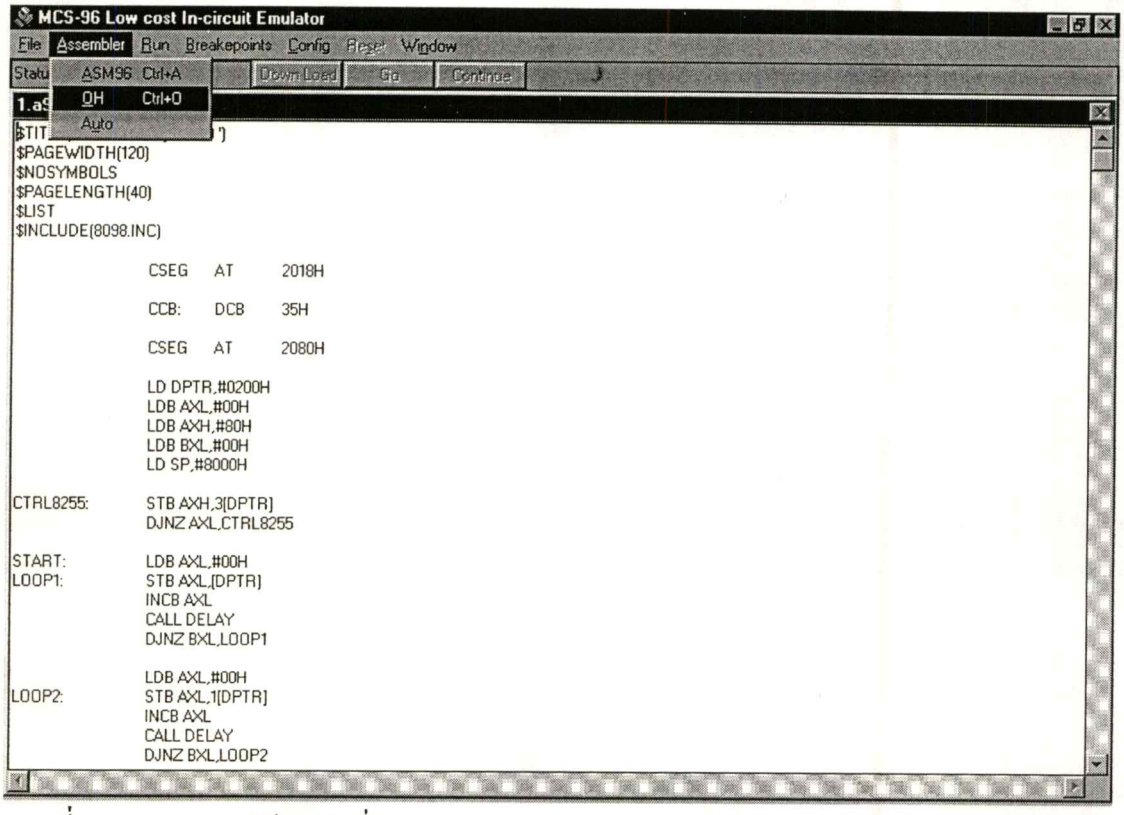


ภาพที่ 4.21 แสดงหน้าต่างที่แจ้งให้ผู้ใช้ทราบว่าได้ทำการแอสแซมเบลอร์เสร็จสิ้นแล้ว

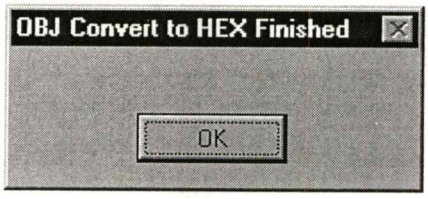


ภาพที่ 4.22 แสดงหน้าต่างที่แจ้งให้ผู้ใช้ทราบผลของการแอสแซมเบลอร์

4.5.8 คำสั่ง OH ใช้ในการแปลงแฟ้มข้อมูลแบบ Object File ที่ได้จากการแอสแซมเบลอร์โปรแกรมที่กำลังพัฒนาอยู่ให้เป็นแฟ้มข้อมูลแบบ Hex File

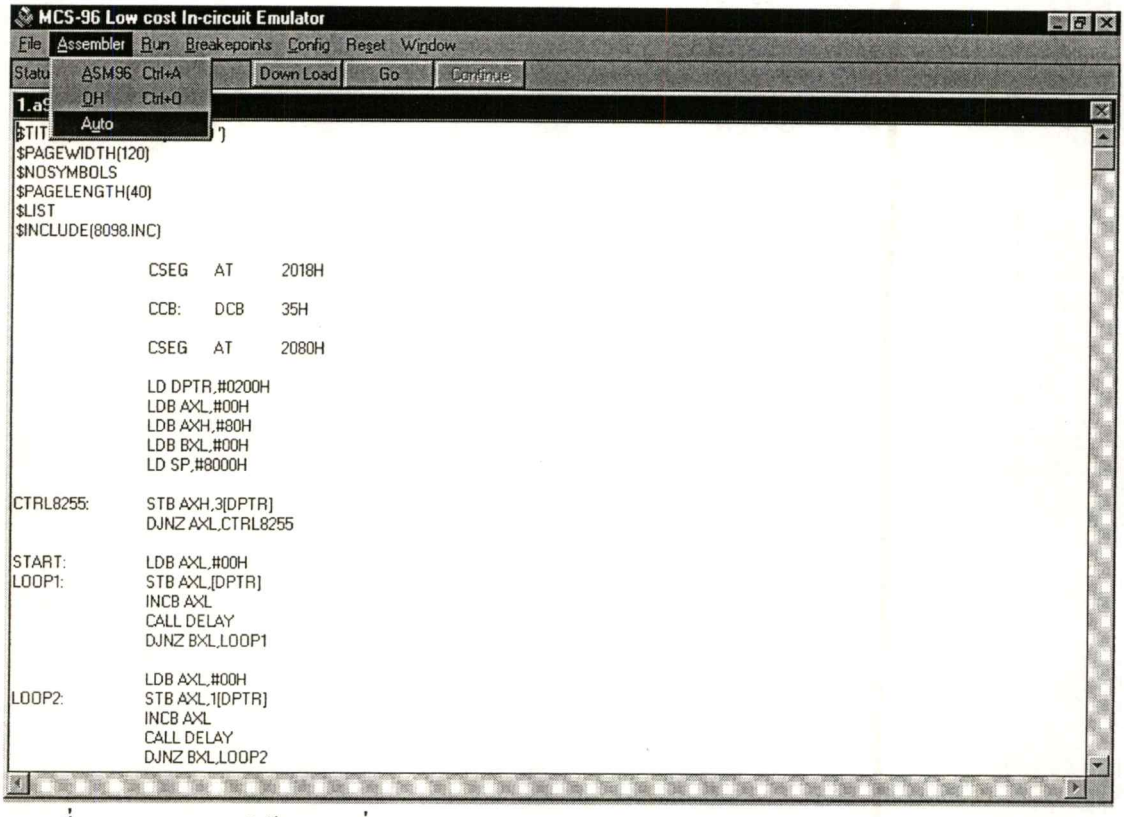


ภาพที่ 4.23 แสดงการใช้งานคำสั่ง OH

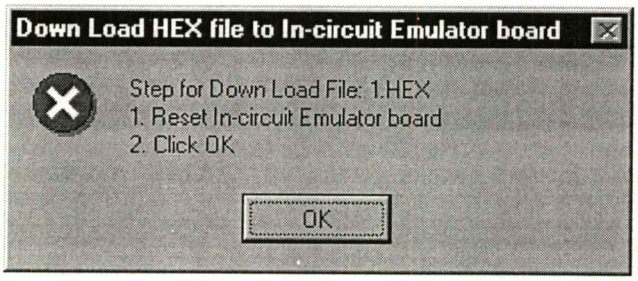


ภาพที่ 4.24 แสดงหน้าต่างที่แจ้งให้ผู้ใช้ทราบว่าได้ทำการแปลงแฟ้มข้อมูลแบบ Object File ไปเป็น Hex File เสร็จสิ้นแล้ว

4.5.9 คำสั่ง Auto จะทำงานตามคำสั่ง ASM96 และ OH ตามลำดับ จากนั้นจะทำการ Down Load ข้อมูลแบบ Hex File ของโปรแกรมที่กำลังพัฒนาอยู่ในขณะนั้น จากเครื่องไมโครคอมพิวเตอร์ไปยังหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์โดยอัตโนมัติ

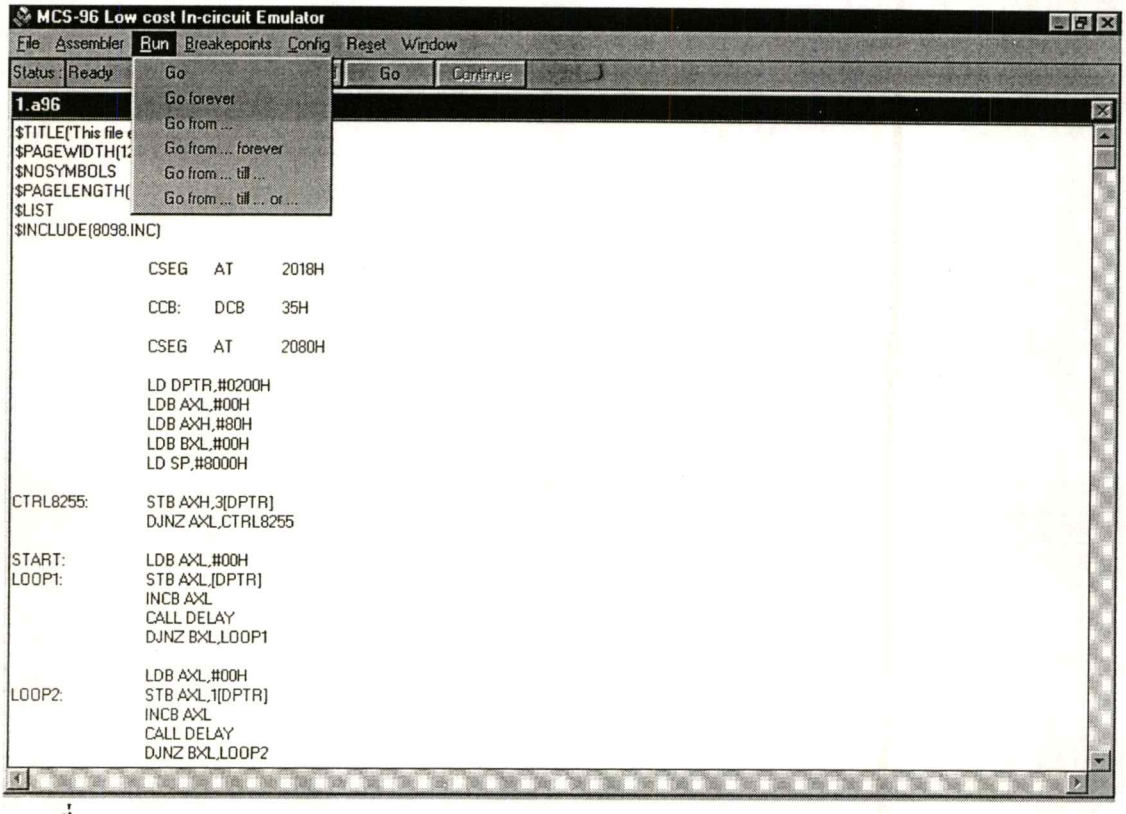


ภาพที่ 4.25 แสดงการใช้งานคำสั่ง Auto



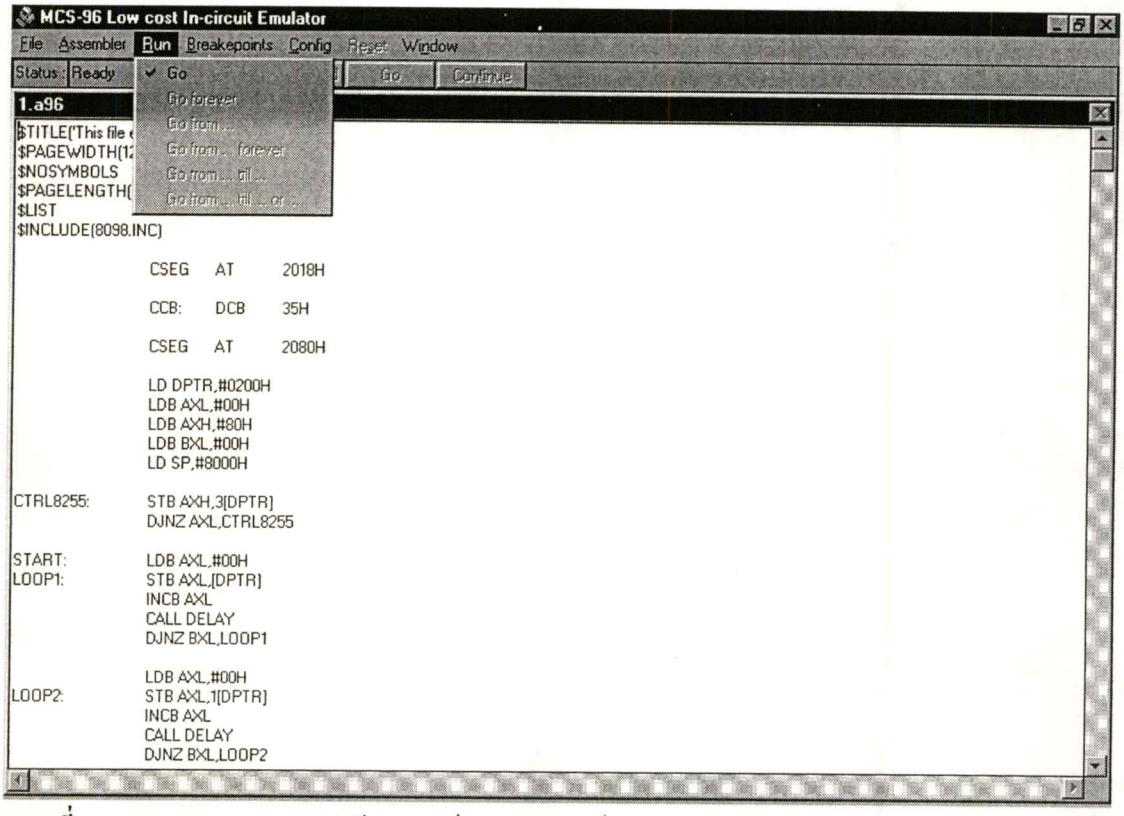
ภาพที่ 4.26 แสดงหน้าต่างในการ Down Load ข้อมูลแบบ Hex File ของโปรแกรมที่กำลังพัฒนาอยู่ในขณะนั้น จากเครื่องไมโครคอมพิวเตอร์ไปยังหน่วยความจำของอีมูเลเตอร์

เมนู Run ใช้กำหนดรูปแบบการทำงานของคำสั่ง Go โดยสามารถเลือกรูปแบบใดก็ได้จากทั้งหมด 6 รูปแบบ แยกอธิบายได้ดังนี้



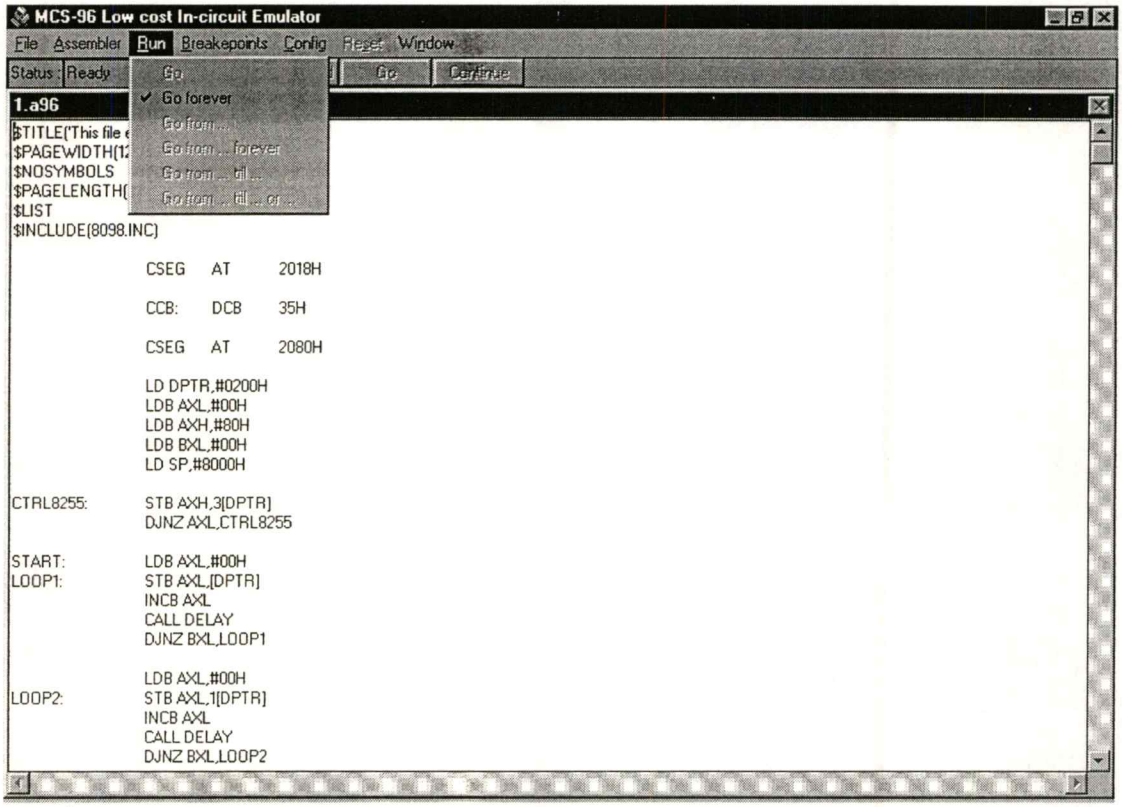
ภาพที่ 4.27 แสดงเมนู Run

4.5.10 Run ในรูปแบบ Go เมื่อคลิกเมาส์เลือกรูปแบบนี้ในการ Run อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์ โดยใช้ค่าของตัวนับโปรแกรม และเบรคพอยบัฟเฟอร์ ที่มีอยู่ในขณะนั้น



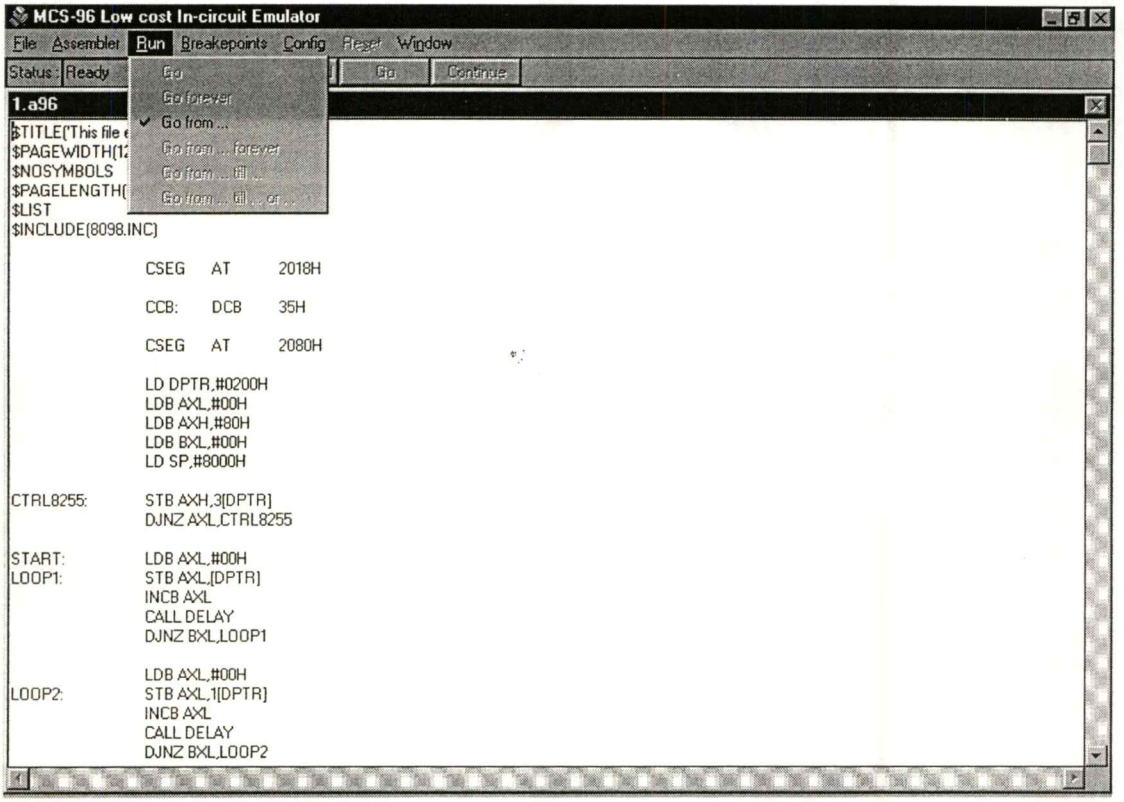
ภาพที่ 4.28 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go

4.5.11 Run ในรูปแบบ Go forever เมื่อคลิกเมาส์เลือกรูปแบบนี้ในการ Run อินเซอร์กิตอีมูเลเตอร์จะทำการเคลียร์ค่าของเบรคพอยบัฟเฟอร์ และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์ โดยใช้ค่าของตัวนับโปรแกรมที่มีอยู่ในขณะนั้น

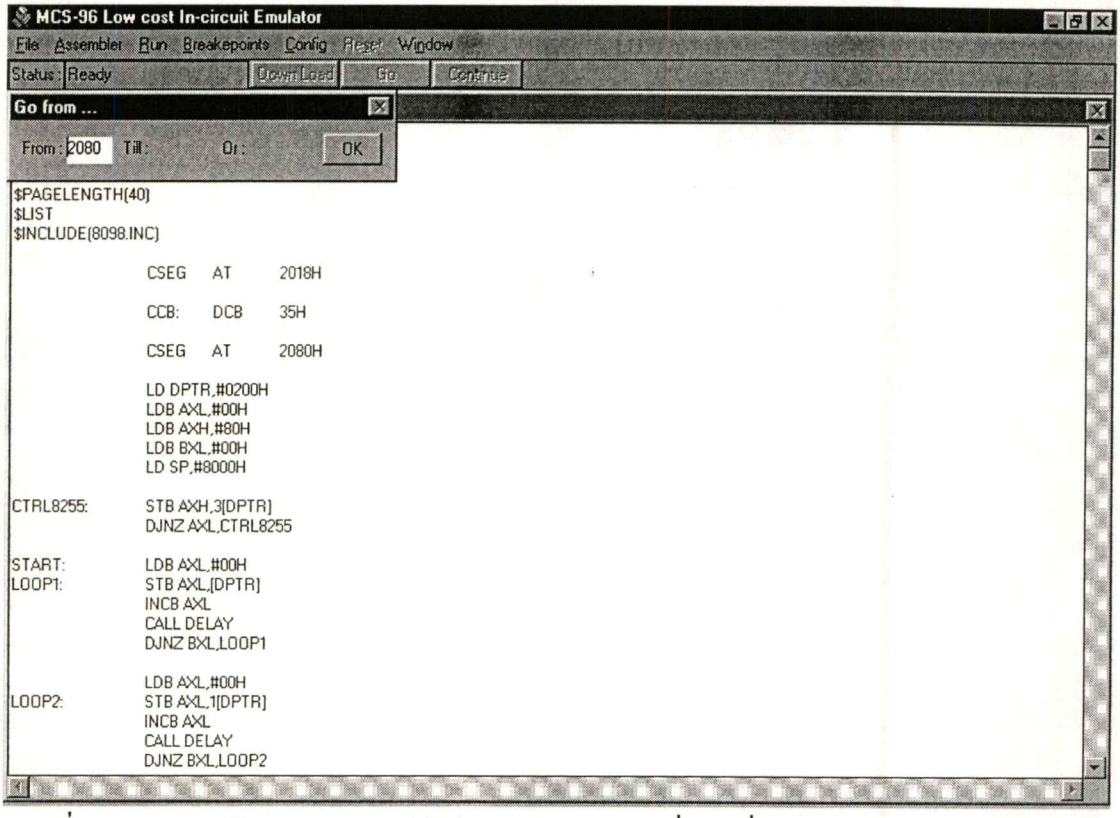


ภาพที่ 4.29 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go forever

4.5.12 Run ในรูปแบบ Go from... เมื่อคลิกเมาส์เลือกรูปแบบนี้ โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้นที่ต้องการขึ้นมา ในการ Run อินเซอร์กิตอีมูเลเตอร์จะทำการโหลดค่าของตัวนับโปรแกรมให้มีค่าเท่ากับแอดเดรสเริ่มต้นที่กำหนดไว้ และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์ โดยใช้ค่าของเบรคพอยบัฟเฟอร์ที่มีอยู่ในขณะนั้น

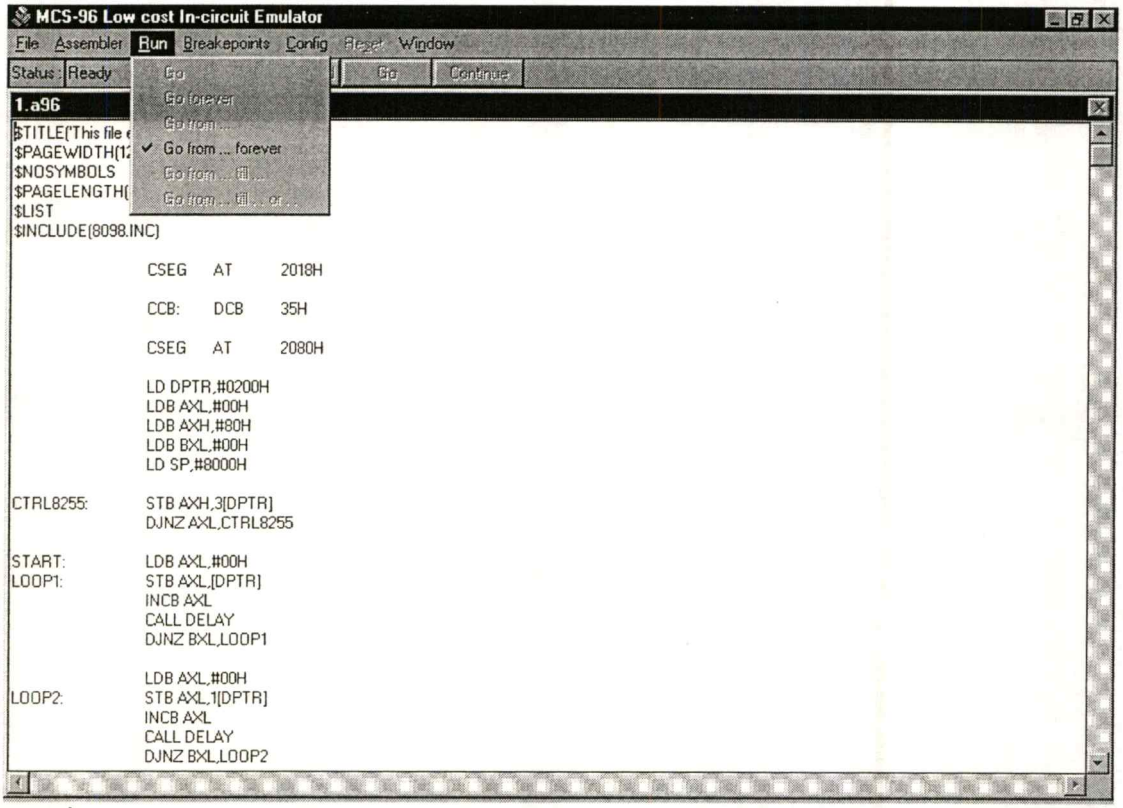


ภาพที่ 4.30 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...

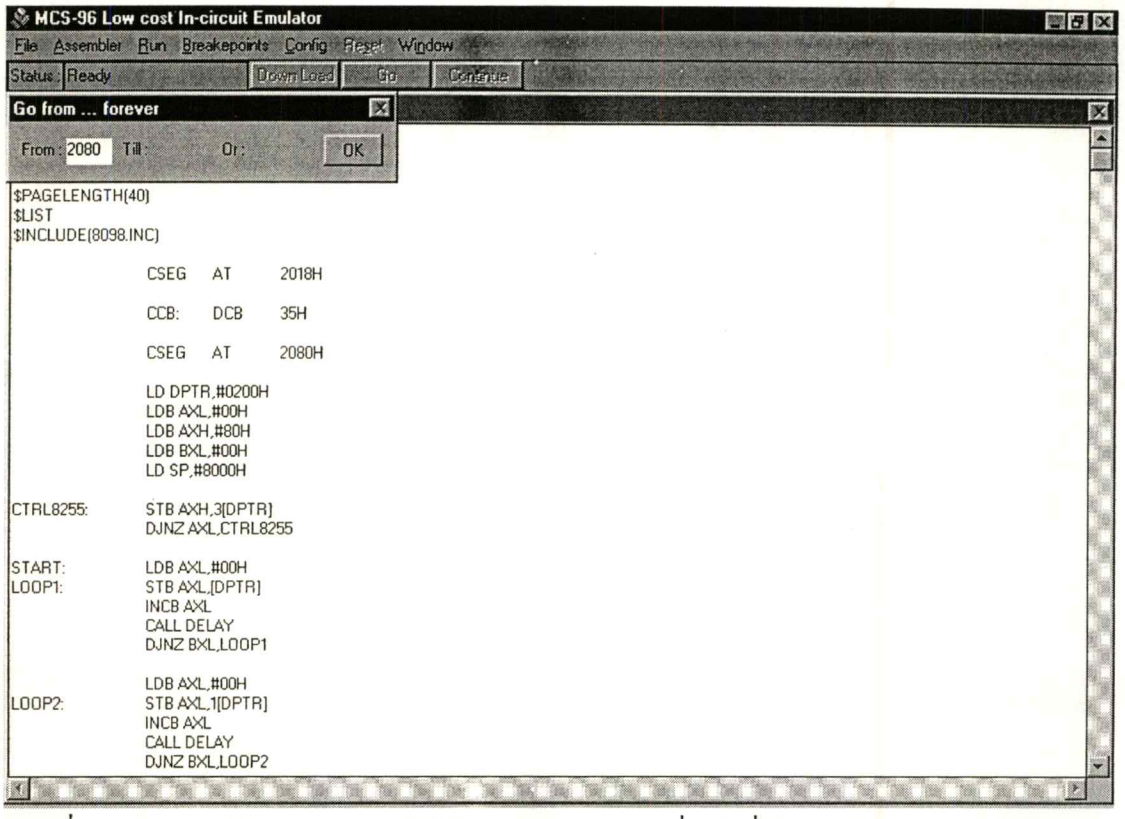


ภาพที่ 4.31 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้นที่ต้องการ ในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from…

4.5.13 Run ในรูปแบบ Go from... forever เมื่อคลิกเมาส์เลือกรูปแบบนี้ โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้นที่ต้องการขึ้นมา ในการ Run อินเซอร์กิตอีมูเลเตอร์จะทำการโหลดค่าของตัวนับโปรแกรมให้มีค่าเท่ากับแอดแดรสเริ่มต้นที่กำหนดไว้, ทำการเคลียร์ค่าของเบรคพอยบัฟเฟอร์ และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์

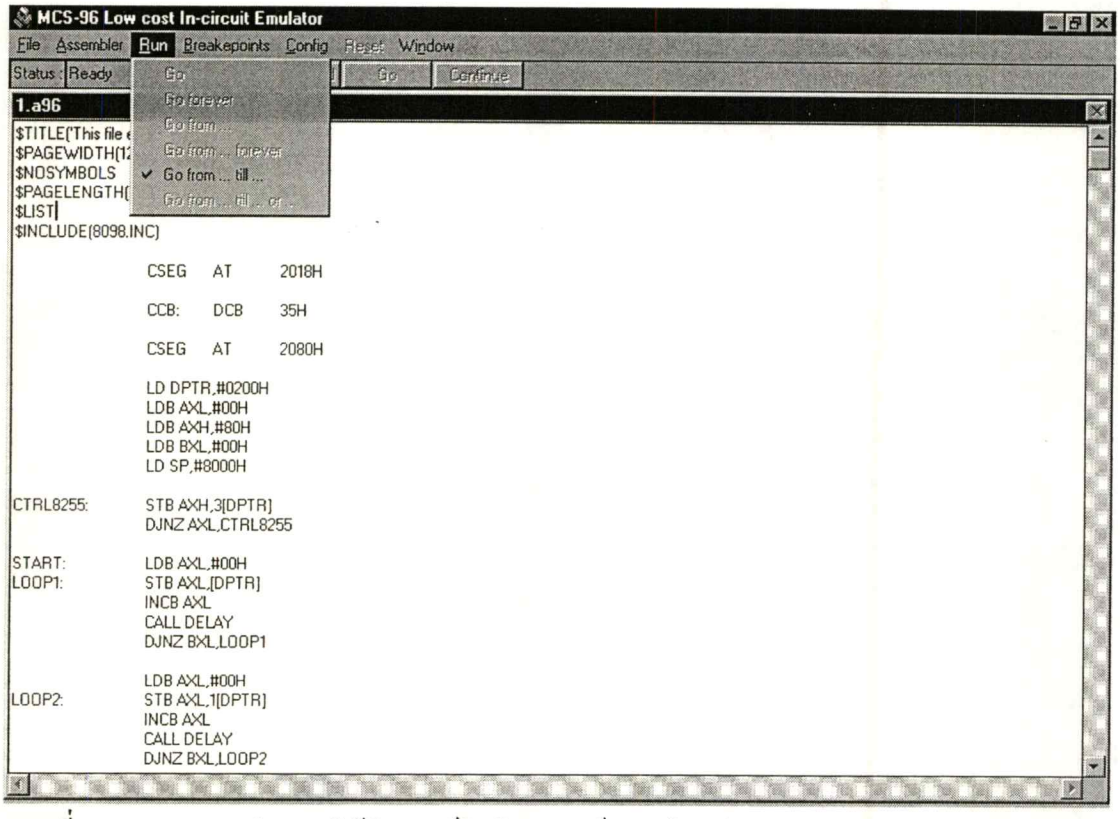


ภาพที่ 4.32 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...forever

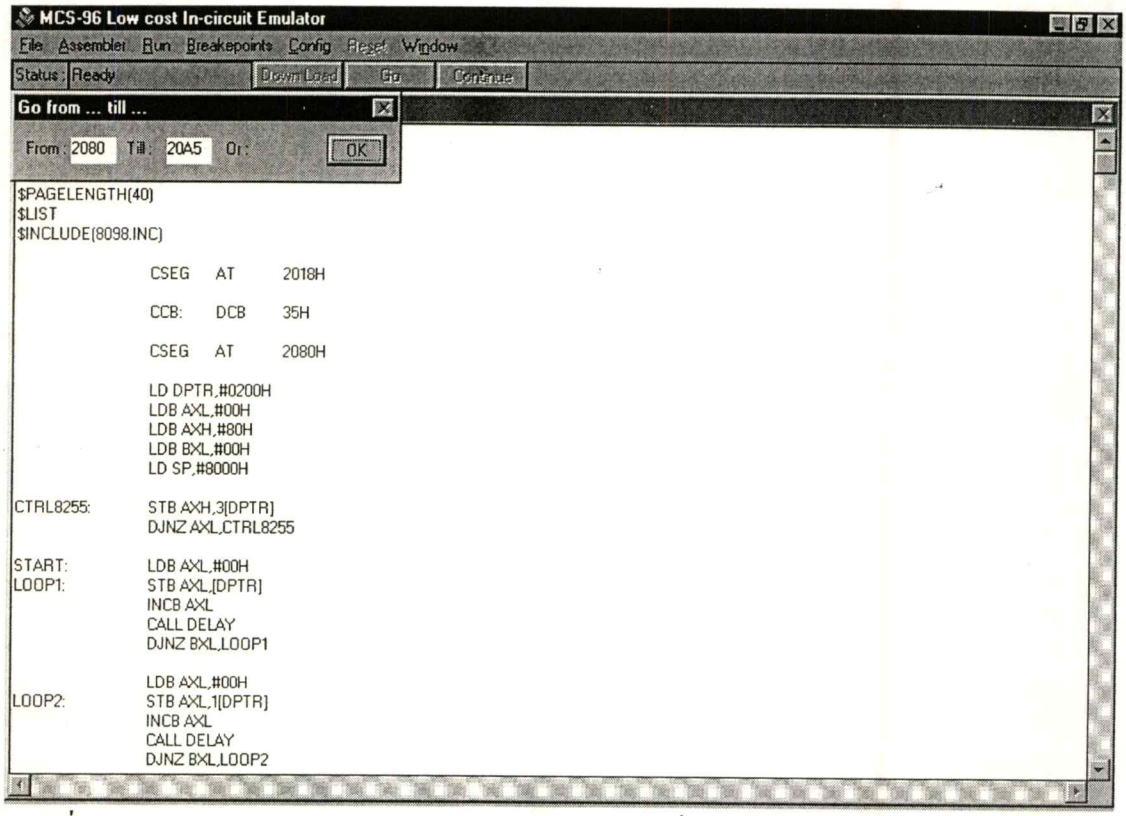


ภาพที่ 4.33 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้นที่ต้องการ ในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from…forever

4.5.14 Run ในรูปแบบ Go from... till... เมื่อคลิกเมาส์เลือกรูปแบบนี้ โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้น และกำหนดแอดเดรสสุดท้ายที่ต้องการหยุดทำงานขึ้นมา ในการ Run อินเซอร์กิตอีมูเลเตอร์จะทำการโหลดค่าของตัวนับโปรแกรมให้มีค่าเท่ากับแอดแดรสเริ่มต้นที่กำหนดไว้, ทำการเซตเบรคพอยบัฟเฟอร์ลำดับที่ 15 ให้มีค่าเท่ากับแอดเดรสสุดท้ายที่ต้องการหยุดทำงาน และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์

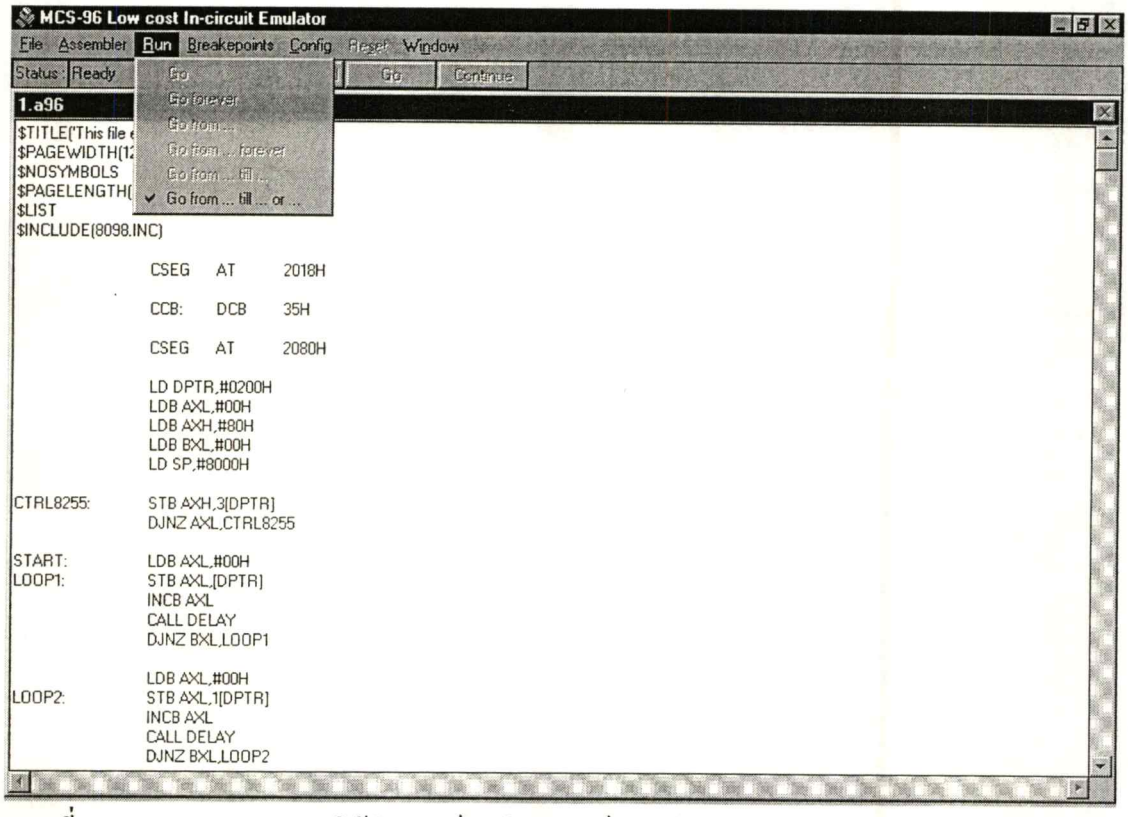


ภาพที่ 4.34 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from…till…

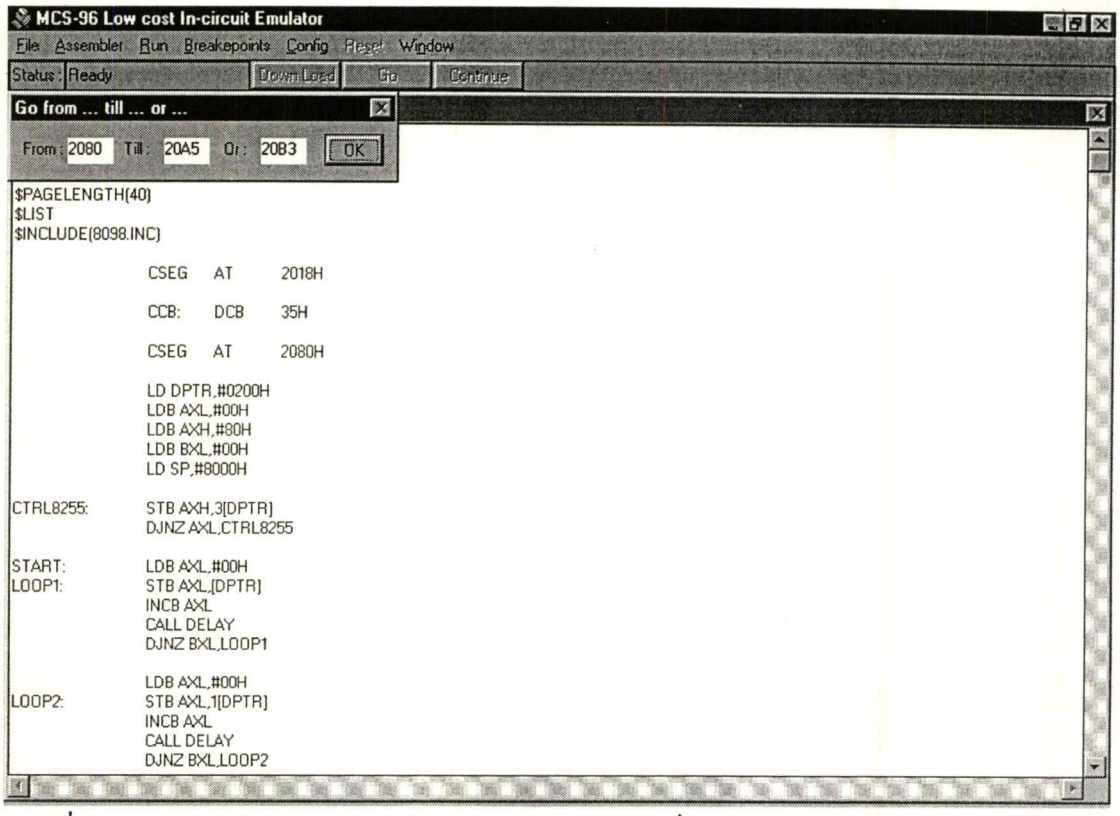


ภาพที่ 4.35 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้น และกำหนดแอดเดรสสุดท้ายที่ต้องการหยุดทำงาน ในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from…till…

4.5.15 Run ในรูปแบบ Go from... till...or... เมื่อคลิกเมาส์เลือกรูปแบบนี้ โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้น และกำหนดแอดเดรสสุดท้าย จำนวน 2 ตำแหน่งที่ต้องการหยุดทำงานขึ้นมา ในการ Run อินเซอร์กิตอีมูเลเตอร์จะทำการโหลดค่าของตัวนับโปรแกรมให้มีค่าเท่ากับแอดแดรสเริ่มต้นที่กำหนดไว้, ทำการเซตเบรคพอยบัฟเฟอร์ลำดับที่ 14 และ 15 ให้มีค่าเท่ากับแอดเดรสสุดท้ายจำนวน 2 ตำแหน่งที่ต้องการหยุดทำงาน และเริ่มทำงานตามคำสั่งของโปรแกรมยูสเซอร์

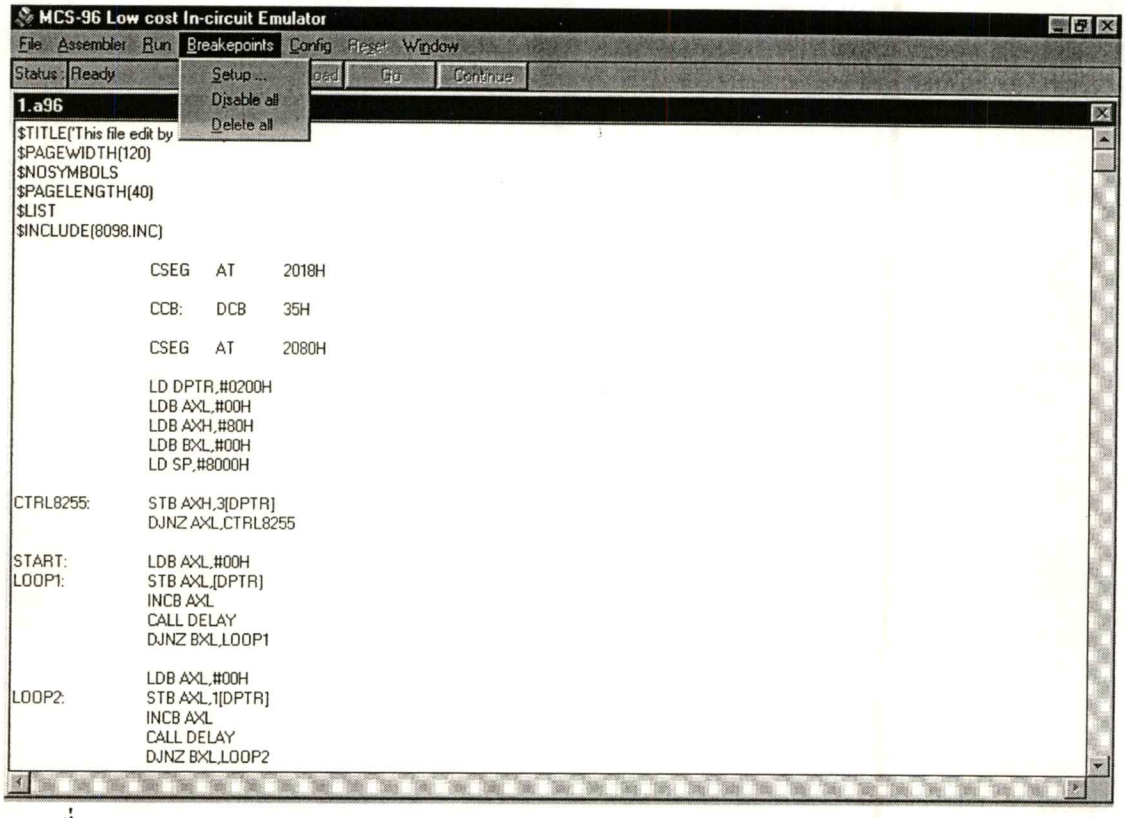


ภาพที่ 4.36 แสดงการกำหนดให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from...till...or...



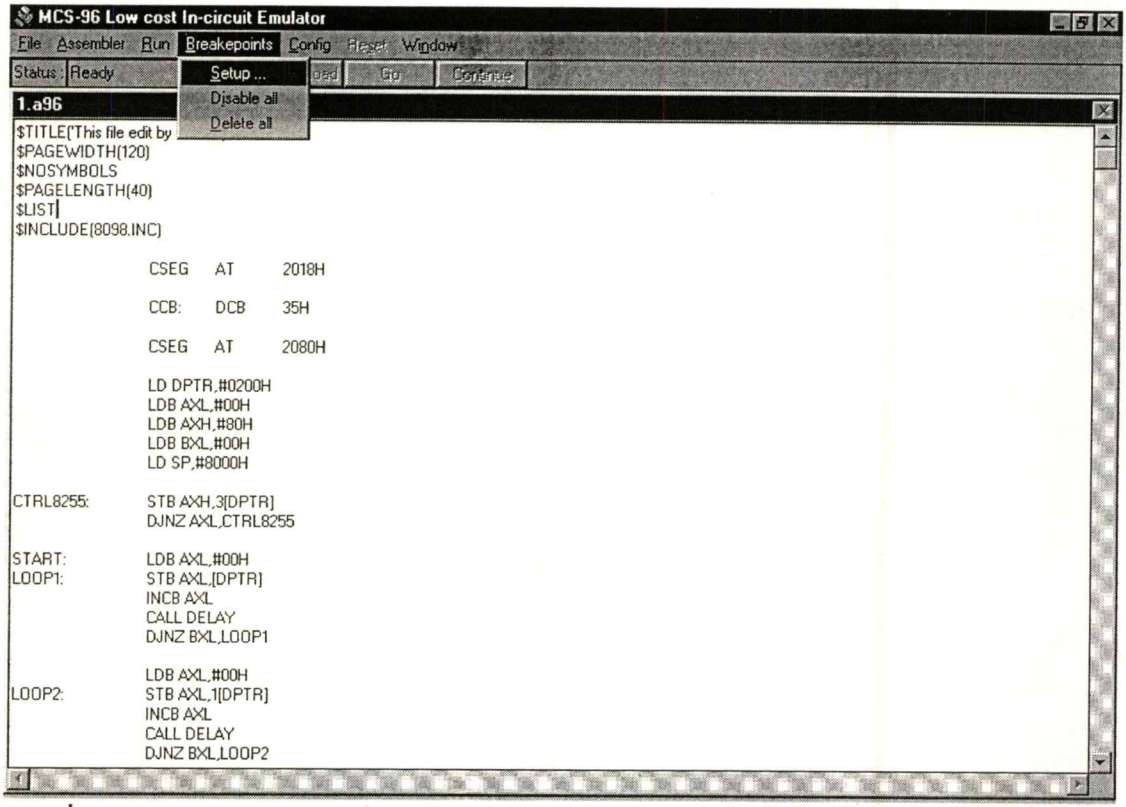
ภาพที่ 4.37 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดแดรสเริ่มต้น และกำหนดแอดเดรสสุดท้าย จำนวน 2 ตำแหน่งที่ต้องการหยุดทำงาน ในกรณีเลือกให้อินเซอร์กิตอีมูเลเตอร์ Run ในรูปแบบ Go from…till…or…

เมนู Breakpoints ภายในประกอบด้วยคำสั่งต่างๆ จำนวน 3 คำสั่งแยกอธิบายได้ดังนี้

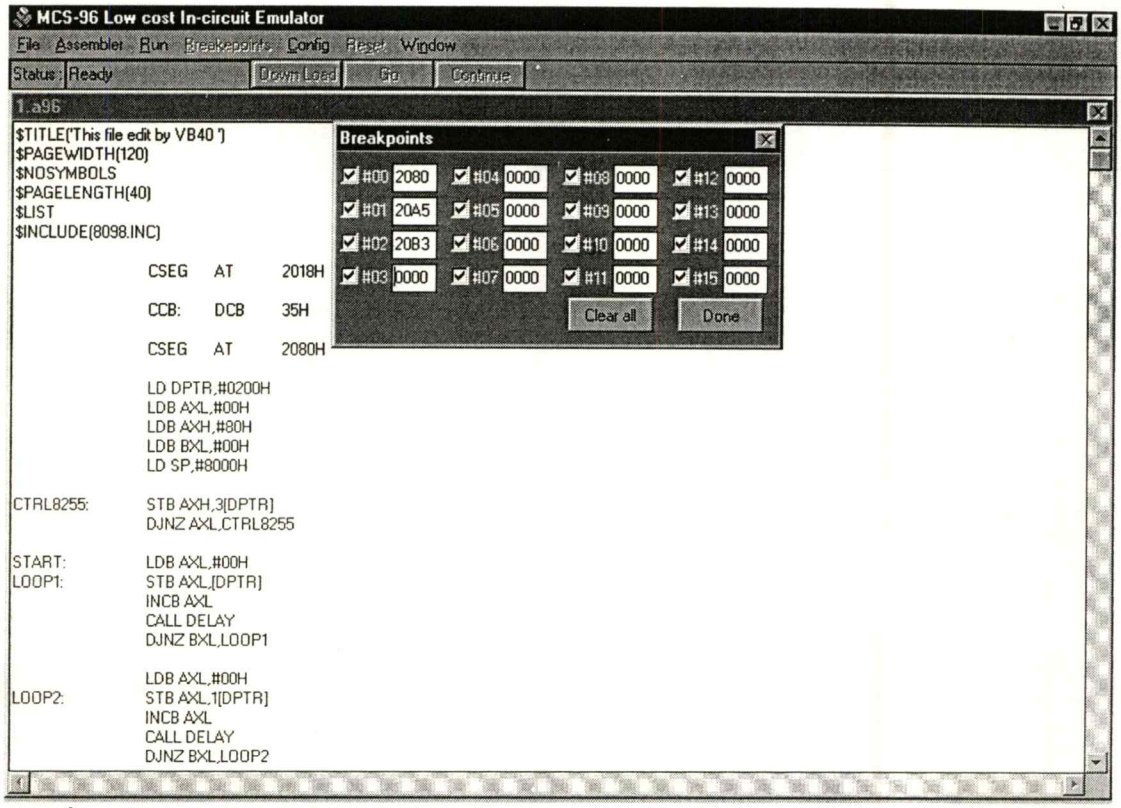


ภาพที่ 4.38 แสดงเมนู Breakpoints

4.5.16 คำสั่ง Setup...(Breakpoints) ใช้สำหรับแสดงค่า หรือทำการเปลี่ยนแปลงค่าของเบรคพอยบัฟเฟอร์ที่มีอยู่ในขณะนั้น เมื่อคลิกเมาส์ที่คำสั่งนี้โปรแกรม ICE8098 จะเปิดหน้าต่างให้ผู้ใช้กำหนดแอดเดรสที่ต้องการหยุดทำงาน โดยสามารถกำหนดได้สูงสุดถึง 16 ตำแหน่ง

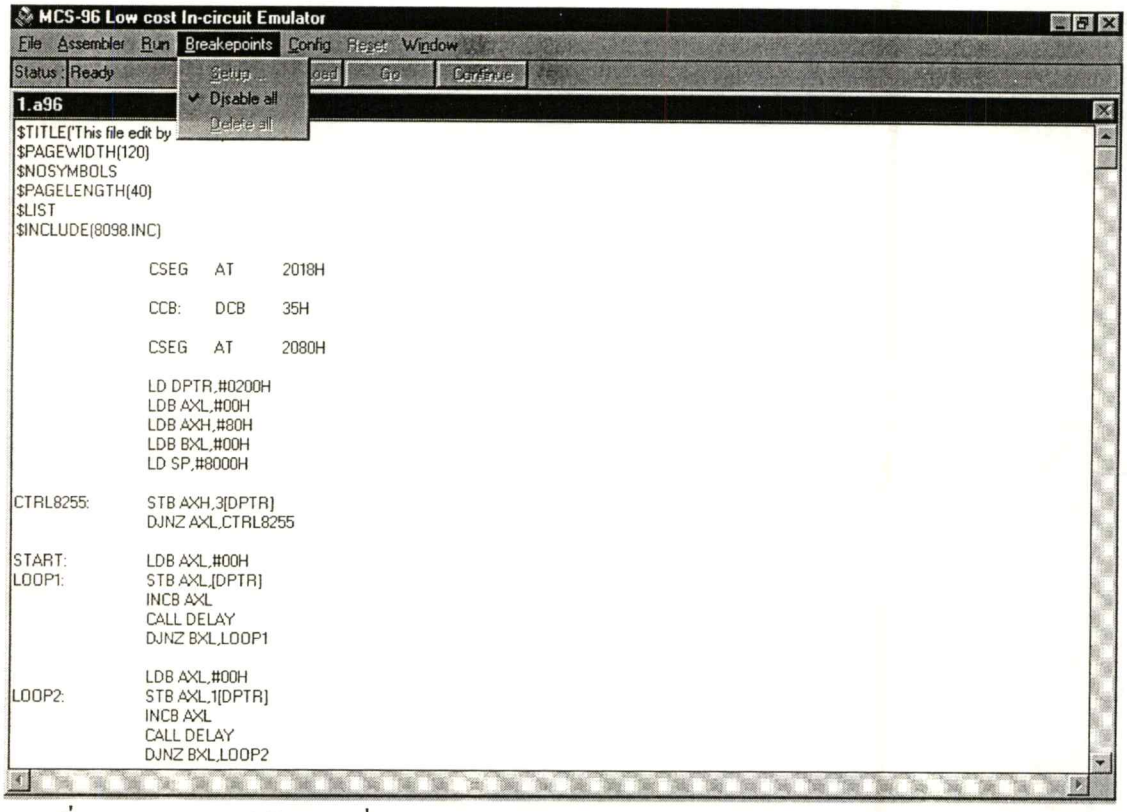


ภาพที่ 4.39 แสดงการใช้งานคำสั่ง Setup...

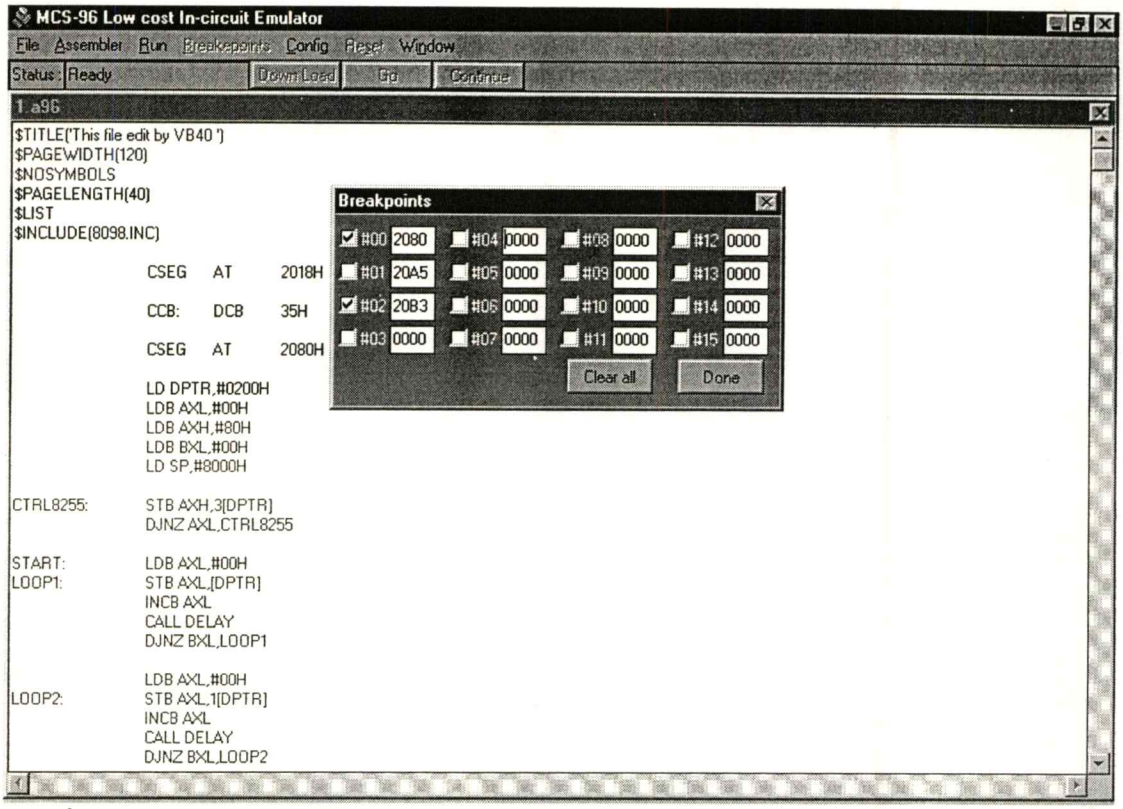


ภาพที่ 4.40 แสดงหน้าต่างที่ให้ผู้ใช้กำหนดแอดเดรสที่ต้องการหยุดทำงาน

4.5.17 คำสั่ง Disable all (Breakpoints) เมื่อคลิกเมาส์ที่คำสั่งนี้ อินเซอร์กิตอีมูเลเตอร์จะทำการยกเลิกค่าเบรคพอย บัฟเฟอร์ทั้งหมดที่มีอยู่ในขณะนั้นชั่วขณะ และผู้ใช้สามารถเรียกค่าเบรคพอยบัฟเฟอร์มาใช้งานได้อีกโดยคลิกเมาส์ที่คำสั่ง Disable all อีกครั้งหนึ่ง

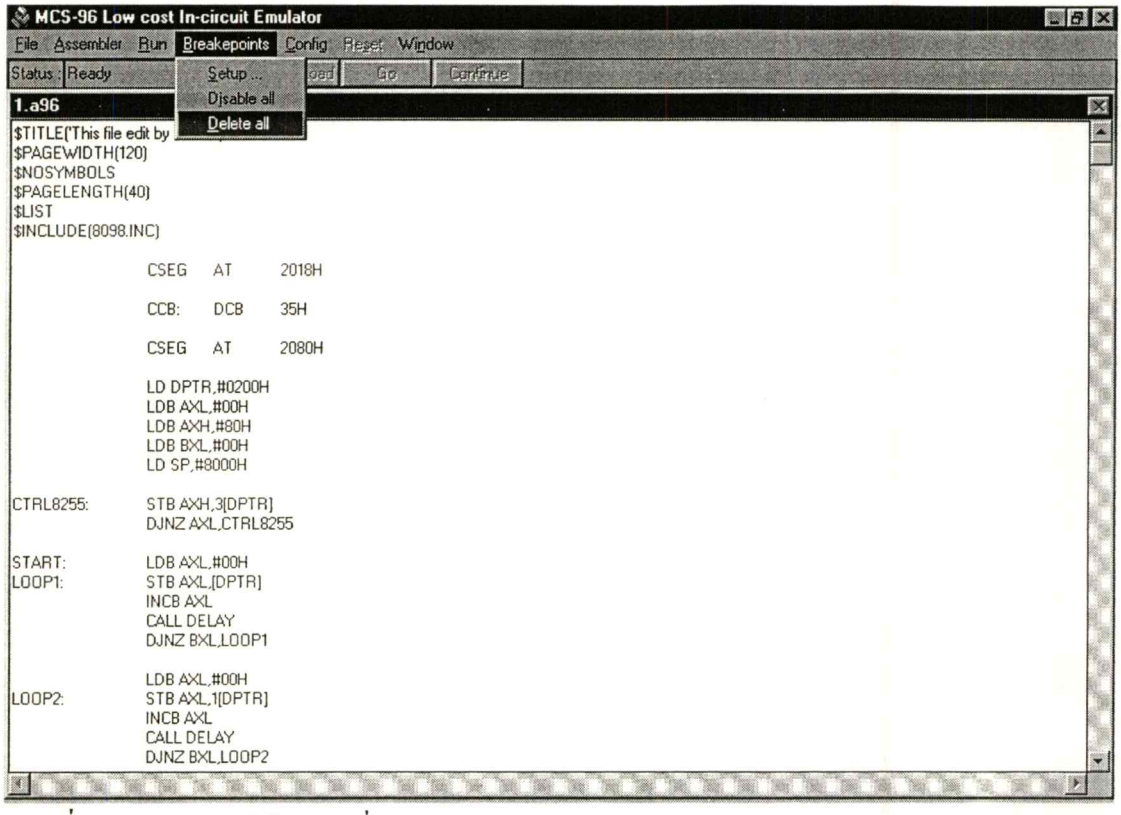


ภาพที่ 4.41 แสดงการใช้งานคำสั่ง Disable all



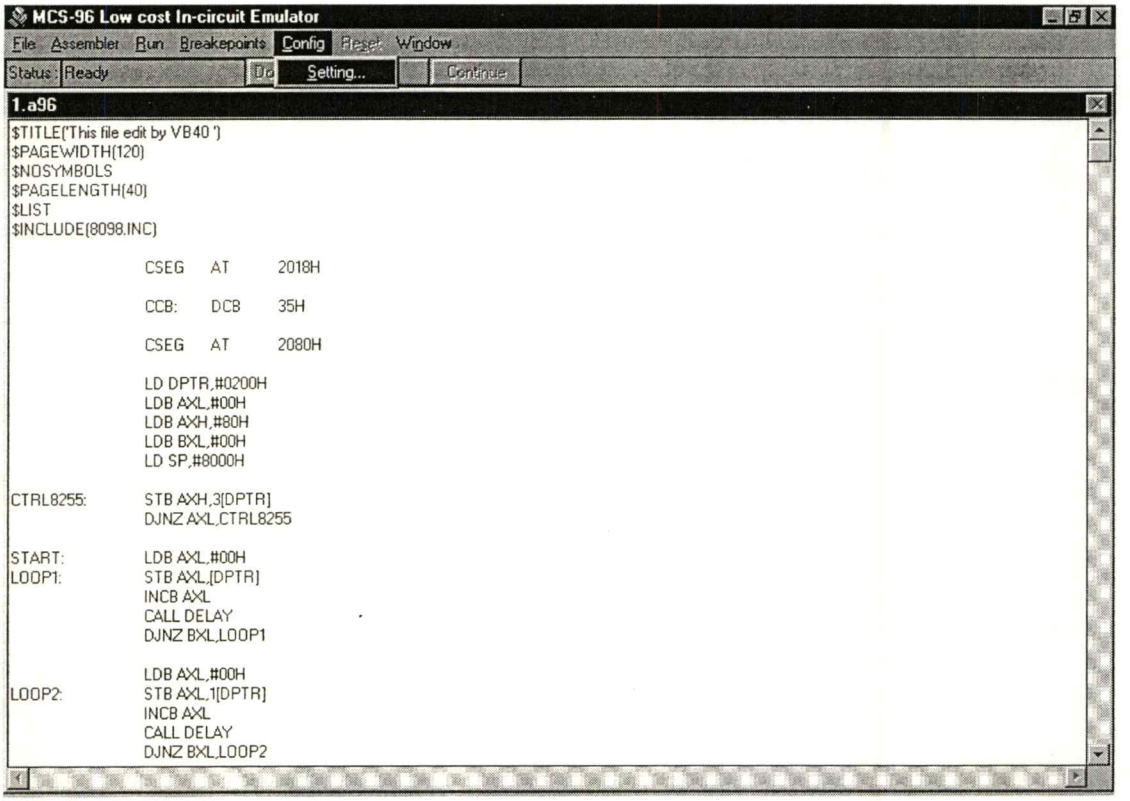
ภาพที่ 4.42 แสดงหน้าต่างให้ผู้ใช้สามารถเรียกค่าเบรคพอยบัฟเฟอร์มาใช้งานได้อีก โดยคลิกเมาส์ที่ Check box ของเบรคพอยบัฟเฟอร์ที่ต้องการใช้งาน

4.5.18 คำสั่ง Delete all (Breakpoints) เมื่อคลิกเมาส์ที่คำสั่งนี้ อินเซอร์กิตอีมูเลเตอร์จะทำการลบค่าเบรคพอยบัฟเฟอร์ทั้งหมดที่มีอยู่ในขณะนั้น โดยไม่สามารถเรียกค่าเบรคพอยบัฟเฟอร์มาใช้งานได้อีก

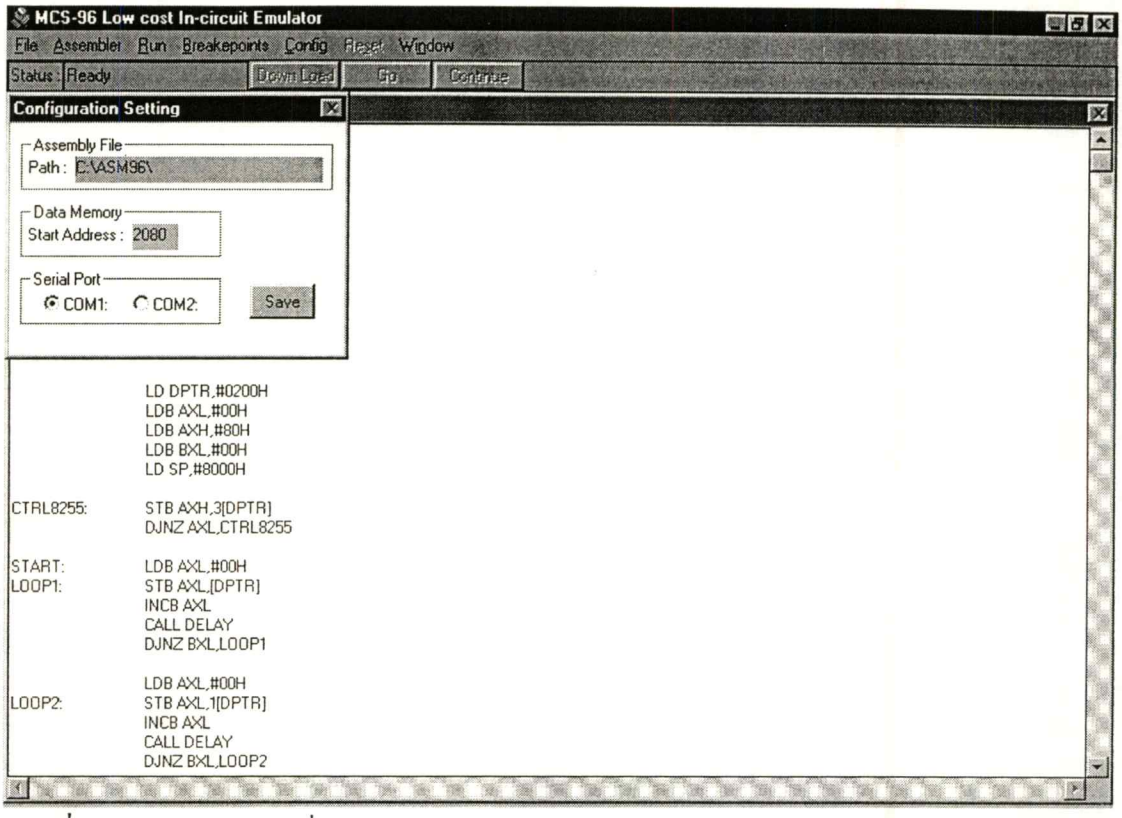


ภาพที่ 4.43 แสดงการใช้งานคำสั่ง Delete all

4.5.19 คำสั่ง Config (Setting...) มีไว้สำหรับให้ผู้ใช้งานได้กำหนดรายละเอียดที่สำคัญดังนี้ ตำแหน่งไดเร็กทอรีของแฟ้มข้อมูลแอสแซมบลี, แอดเดรสเริ่มต้นของหน่วยความจำข้อมูล และหมายเลขพอร์ตอนุกรมของเครื่องไมโครคอมพิวเตอร์ที่ใช้ในการเชื่อมต่อกับบอร์ดเป้าหมาย โดยผู้ใช้จะต้องระบุข้อมูลต่างๆ ให้ถูกต้อง มิฉะนั้นจะไม่สามารถใช้งานอินเซอร์กิตอีมูเลเตอร์ได้ รายละเอียดเหล่านี้จะบันทึกเก็บไว้ในแฟ้มข้อมูลชื่อ "ICE8098.INI" และทุกครั้งที่เริ่มใช้งานโปรแกรม ICE8098 โปรแกรมจะทำการอ่านข้อมูลจากแฟ้มนี้ เพื่อกำหนดเป็น Configuration ของโปรแกรมต่อไป เมื่อคลิกเมาส์ที่คำสั่งนี้ โปรแกรมจะเปิดหน้าต่างให้ผู้ใช้กำหนดรายละเอียดต่างๆ ดังที่ได้กล่าวไปแล้ว

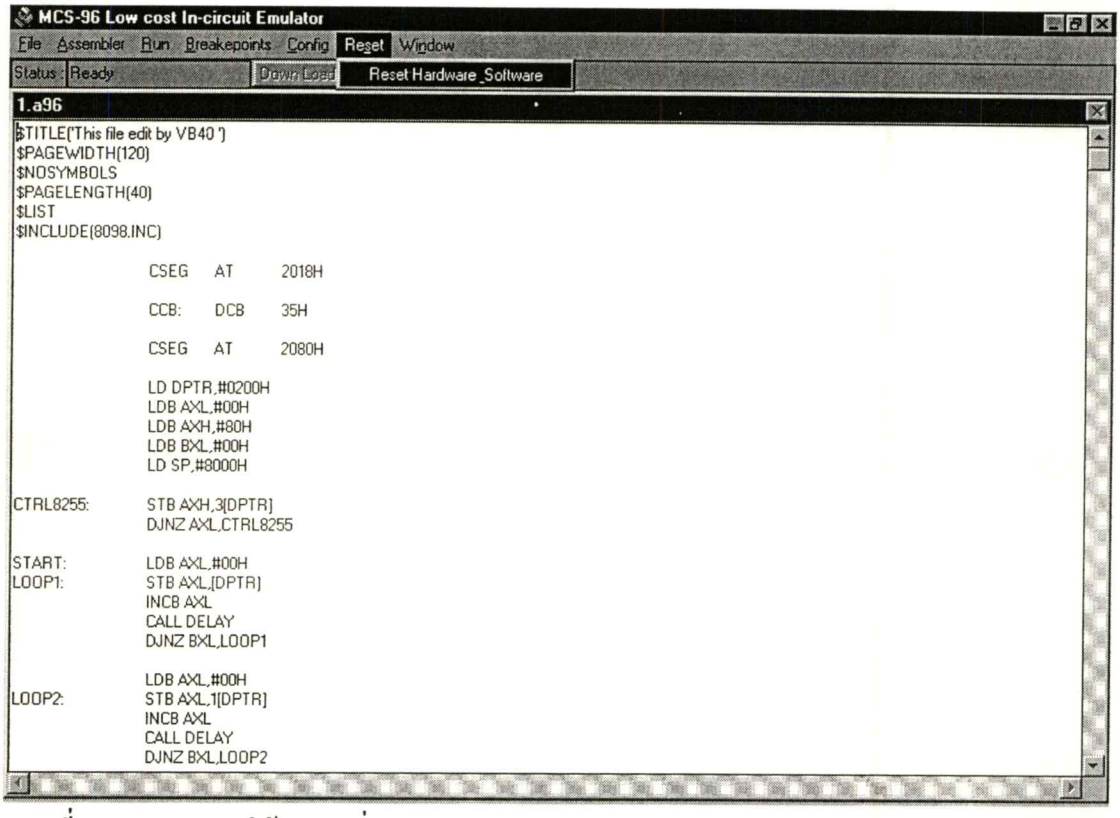


ภาพที่ 4.44 แสดงการใช้งานคำสั่ง Config (Setting...)

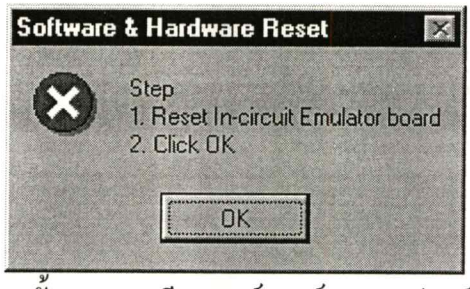


ภาพที่ 4.45 แสดงหน้าต่างที่ให้ผู้ใช้กำหนดตำแหน่งไดเร็กทอรีของแฟ้มข้อมูลแอสแซมบลี, แอดเดรสเริ่มต้นของหน่วยความจำข้อมูล และหมายเลขพอร์ตอนุกรมของเครื่องไมโครคอมพิวเตอร์ ที่ใช้ในการเชื่อมต่อกับบอร์ดเป้าหมาย

4.5.20 คำสั่ง Reset (Reset hardware, Software) ใช้สำหรับรีเซตฮาร์ดแวร์และซอฟแวร์ กรณีที่เกิดความผิดพลาดขึ้นในขณะทำการพัฒนาโปรแกรม เมื่อคลิกเมาส์ที่คำสั่งนี้ โปรแกรมจะเปิดหน้าต่างแสดงขั้นตอนการรีเซตฮาร์ดแวร์และซอฟแวร์เพื่อให้ผู้ใช้ปฏิบัติตาม

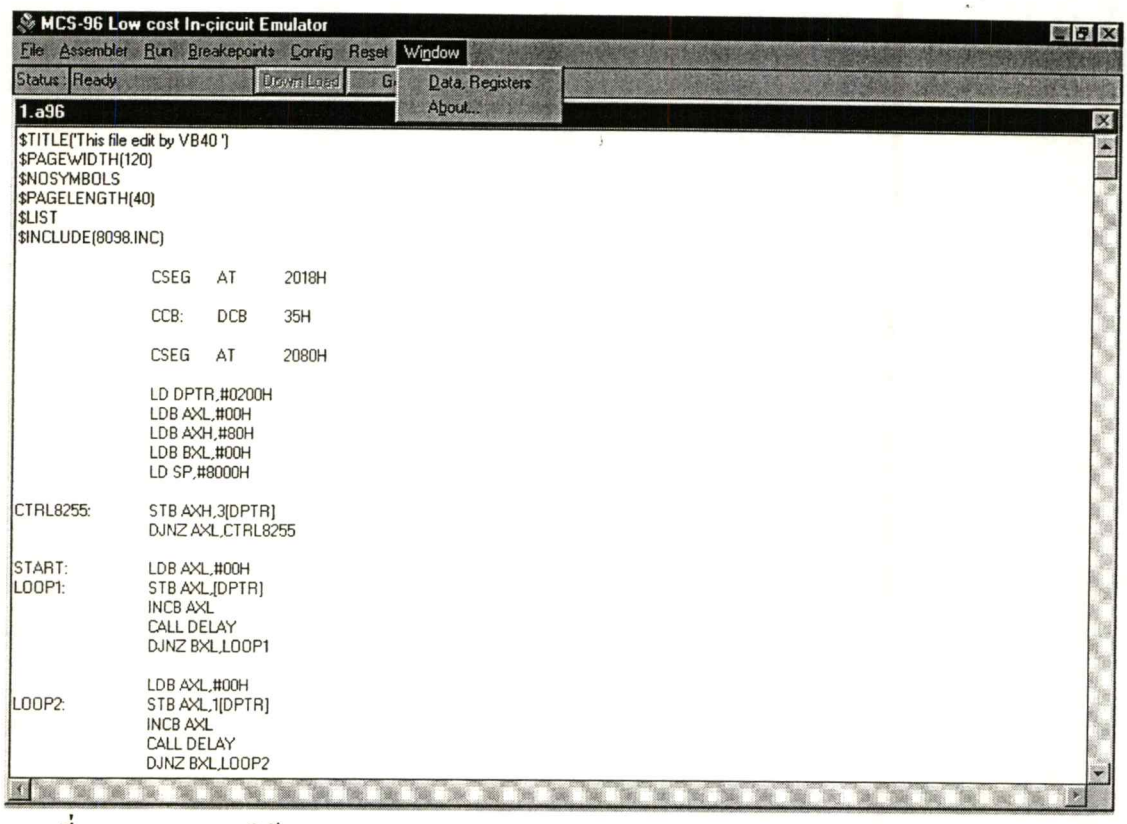


ภาพที่ 4.46 แสดงการใช้งานคำสั่ง Reset (Reset Hardware, Software)



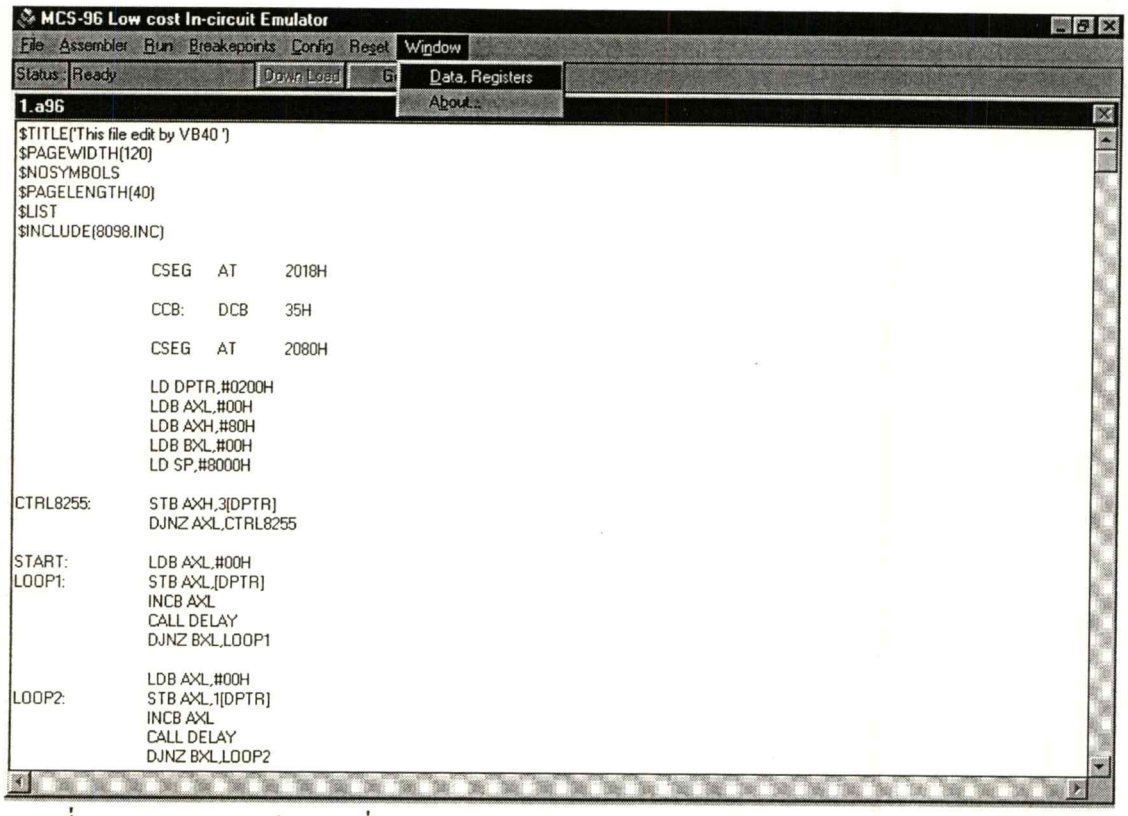
ภาพที่ 4.47 แสดงหน้าต่างของขั้นตอนการรีเซตฮาร์ดแวร์และซอฟแวร์

เมนู Window ภายในประกอบด้วยคำสั่ง 2 คำสั่งแยกอธิบายได้ดังนี้

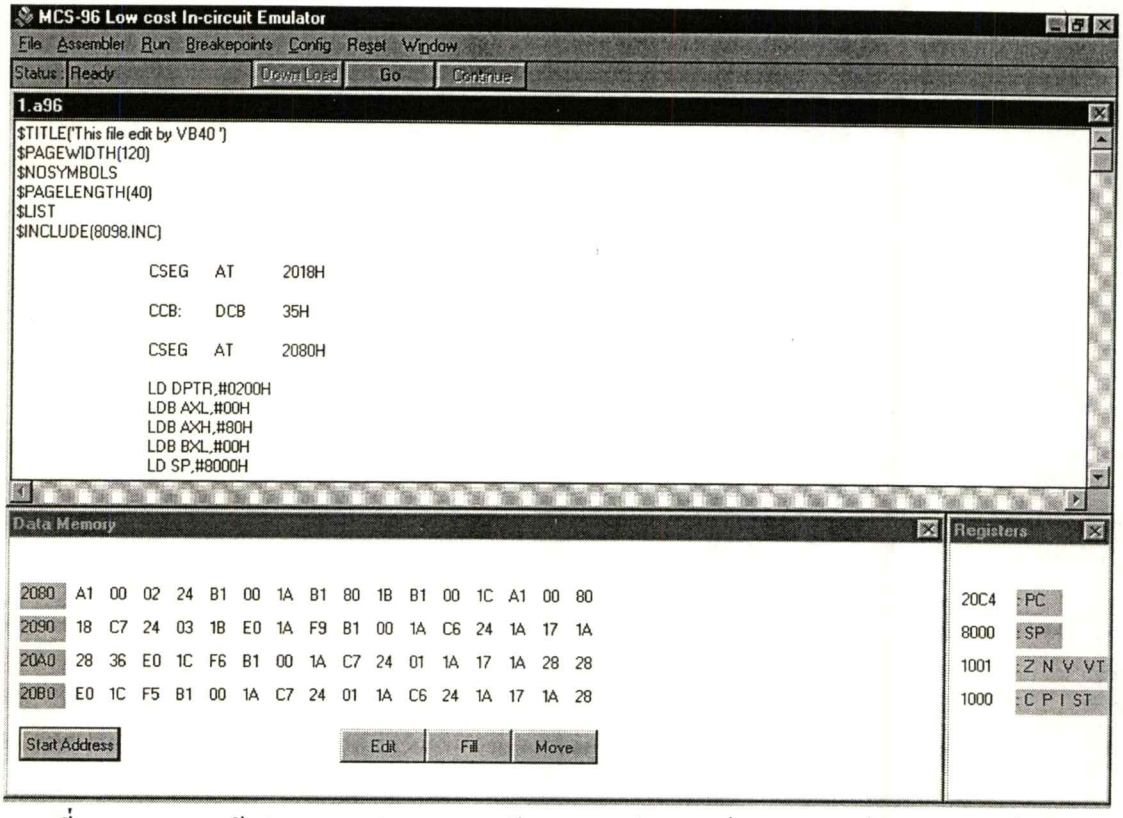


ภาพที่ 4.48 แสดงการใช้งานเมนู Window

4.5.21 คำสั่ง Data, Registers เมื่อคลิกเมาส์ที่คำสั่งนี้ โปรแกรมจะทำการเปิดหน้าต่างเพื่อแสดง, แก้ไขหรือเปลี่ยนแปลงข้อมูลภายในหน่วยความจำข้อมูลและรีจิสเตอร์ของอินเซอร์กิตอีมูเลเตอร์ขึ้นมา ได้แก่ ตัวนับโปรแกรม , ตัวชี้สแตก และรีจิสเตอร์คำแสดงสถานะโปรแกรม

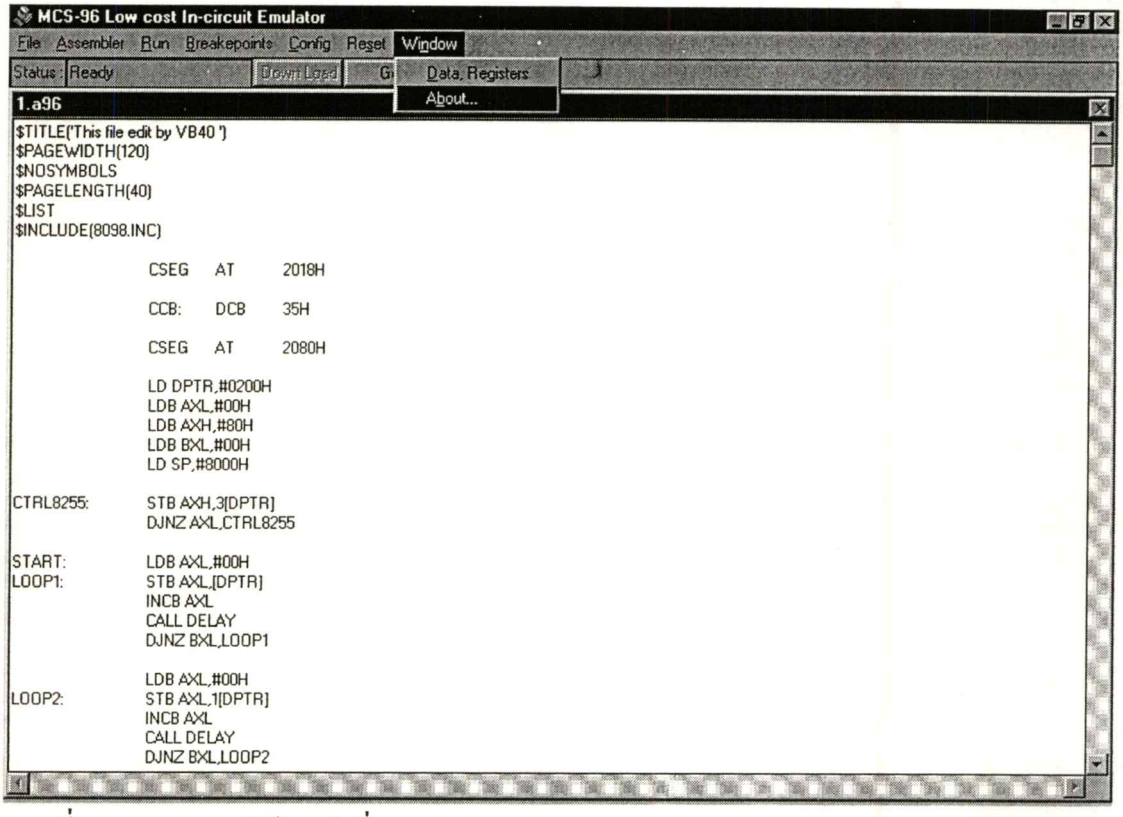


ภาพที่ 4.49 แสดงการใช้งานคำสั่ง Data, Registers

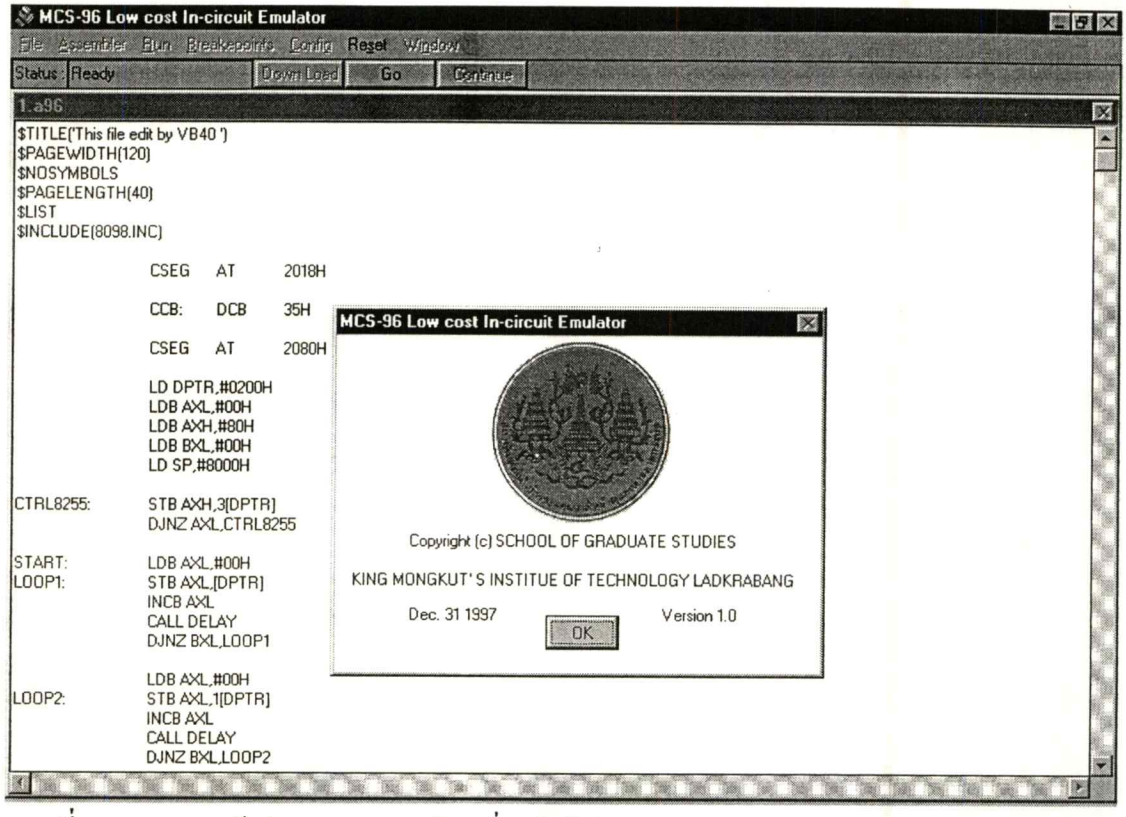


ภาพที่ 4.50 แสดงหน้าต่างของหน่วยความจำข้อมูล และรีจิสเตอร์ของอินเซอร์กิตอีมูเลเตอร์

4.5.22 คำสั่ง About... เมื่อคลิกเมาส์ที่คำสั่งนี้ โปรแกรมจะเปิดหน้าต่างที่แสดงรายละเอียดเกี่ยวกับโปรแกรม ICE8098



ภาพที่ 4.51 แสดงการใช้งานคำสั่ง About...

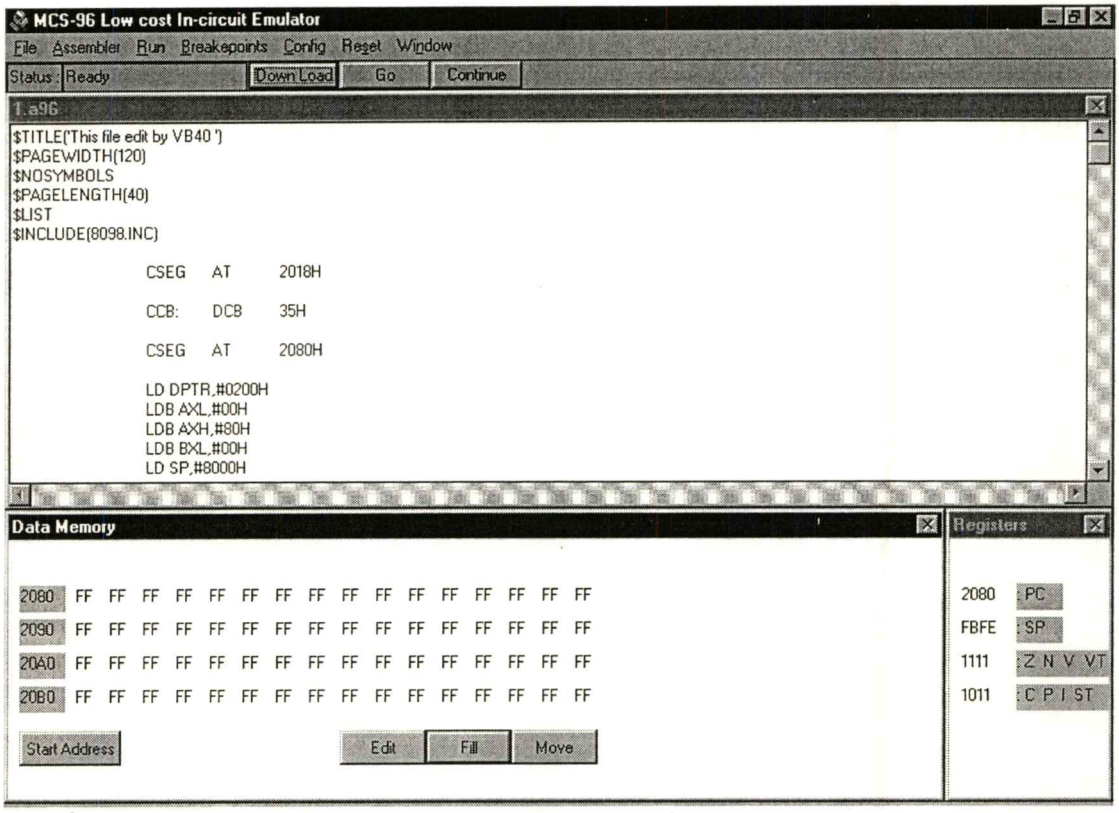


ภาพที่ 4.52 แสดงหน้าต่างของรายละเอียดเกี่ยวกับโปรแกรม ICE8098

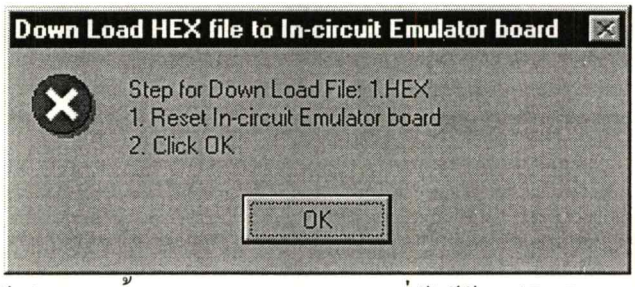
4.5.23 กรอบ Status : ใช้แสดงสภาวะการทำงานของอินเซอร์กิตอีมูเลเตอร์ ประกอบด้วย 3 สภาวะดังนี้

- สภาวะเตรียมพร้อม (Ready) ในสภาวะนี้อินเซอร์กิตอีมูเลเตอร์จะรอรับคำสั่งของผู้ใช้
- สภาวะทำงาน (Run) ในสภาวะนี้อินเซอร์กิตอีมูเลเตอร์กำลังทำงานตามคำสั่งของผู้ใช้
- สภาวะหยุดการทำงาน (Break at address...) เมื่อมีการกำหนดเบรคแอดเดรส และอินเซอร์กิตอีมูเลเตอร์ทำงานจนถึงแอดเดรสที่ระบุไว้อินเซอร์กิตอีมูเลเตอร์จะหยุดทำงาน กรอบ Status : จะแสดงตำแหน่งของแอดเดรสที่หยุดทำงานนั้นให้ผู้ใช้ทราบ

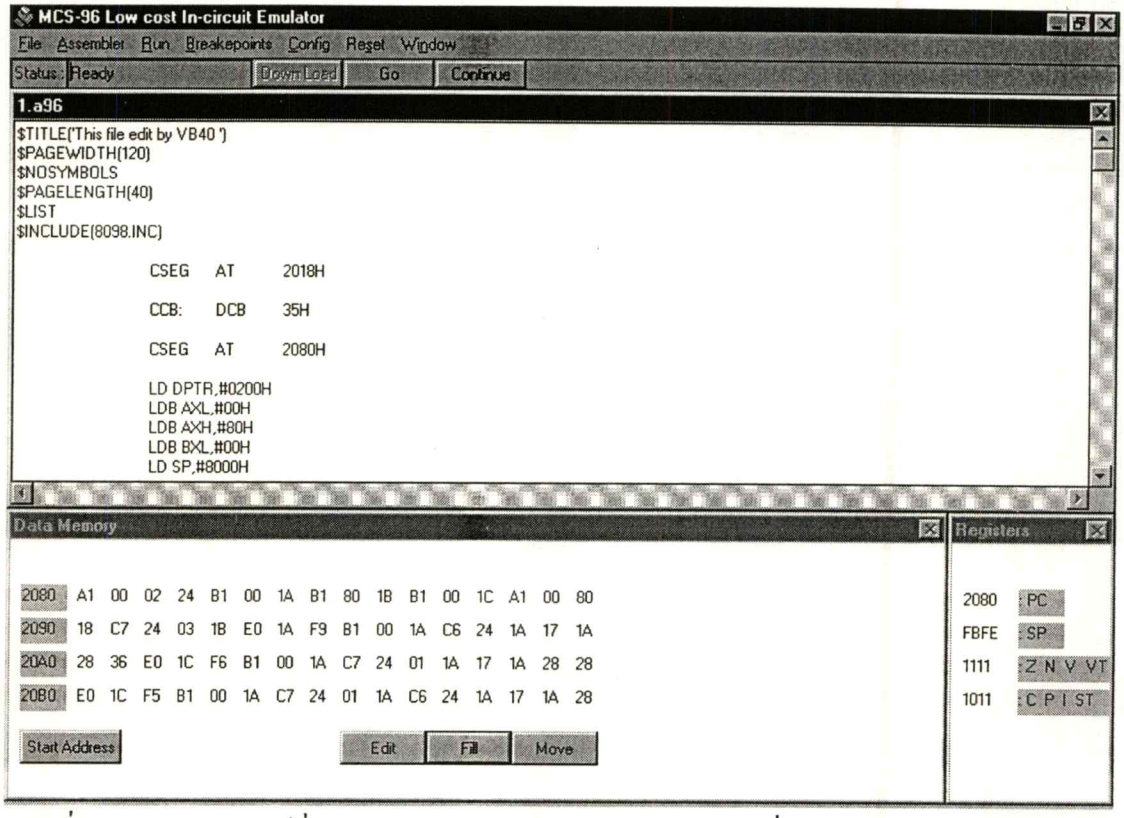
4.5.24 คำสั่ง Down load เมื่อคลิกเมาส์ที่ปุ่มนี้ โปรแกรม ICE8098 จะทำการส่งแฟ้มข้อมูลแบบ HEX ของโปรแกรมที่กำลังพัฒนาอยู่ผ่านพอร์ตอนุกรมไปเก็บไว้ในหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์



ภาพที่ 4.53 แสดงการใช้งานปุ่ม Down load ขอให้สังเกตุที่หน้าต่าง Data Memory ข้อมูลภายในหน่วยความจำ มีค่าเป็น FFH ทั้งหมด

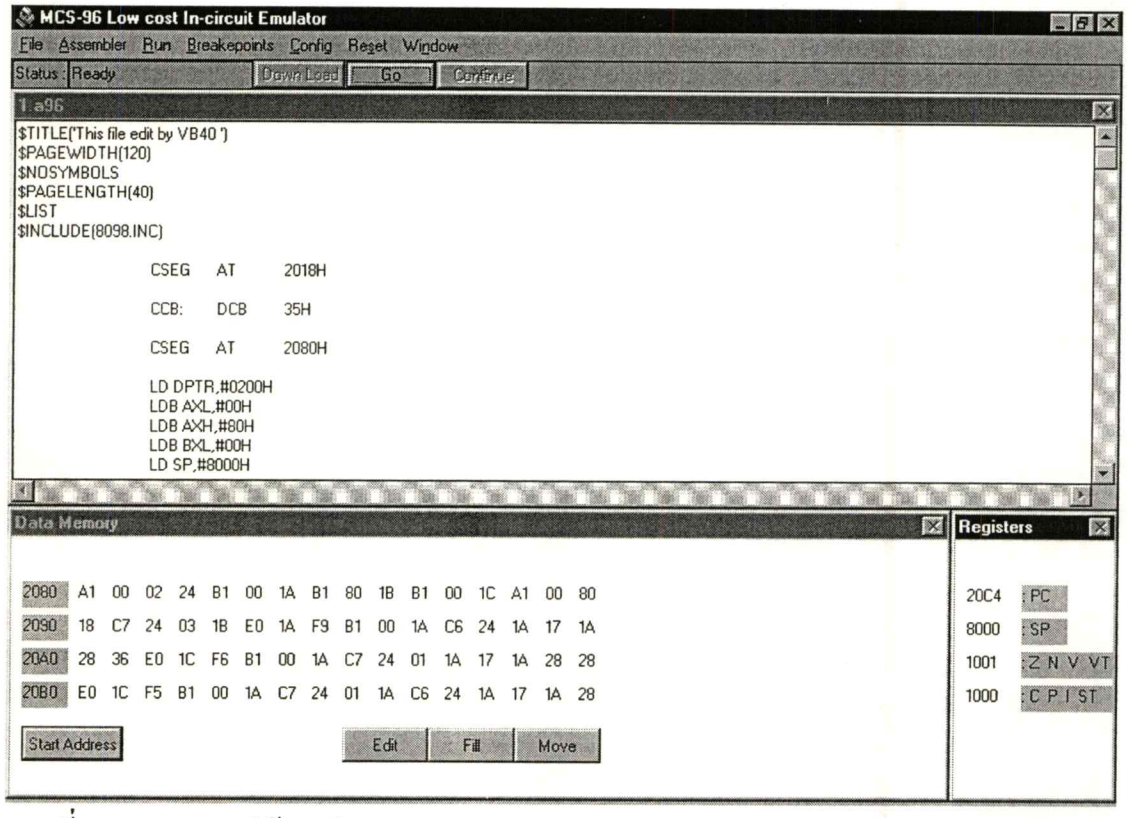


ภาพที่ 4.54 แสดงหน้าต่างของขั้นตอนการ Down Load ที่ผู้ใช้ต้องปฏิบัติตาม เมื่อคลิกเมาส์ที่ปุ่ม Down Load

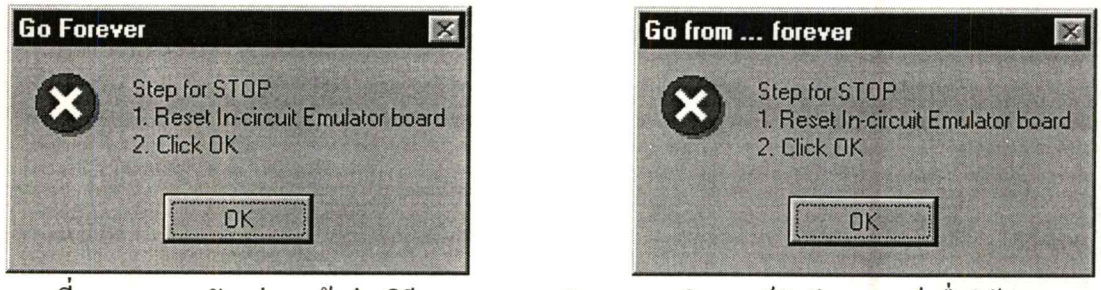


ภาพที่ 4.55 แสดงผลลัพธ์ที่ได้จากการใช้งานปุ่ม Down Load สังเกตุที่หน้าต่าง Data Memory ข้อมูลภายในหน่วยความจำจะเปลี่ยนจาก FFH เป็นข้อมูลของโปรแกรมที่ผู้ใช้กำลังพัฒนาอยู่

4.5.25 คำสั่ง Go เมื่อคลิกเมาส์ที่ปุ่มนี้ อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานตามคำสั่งของโปรแกรมที่ผู้ใช้กำลังพัฒนาอยู่ ซึ่งรูปแบบของการทำงานนั้นจะขึ้นอยู่กับการกำหนดภายในเมนู Run โดยผู้ใช้

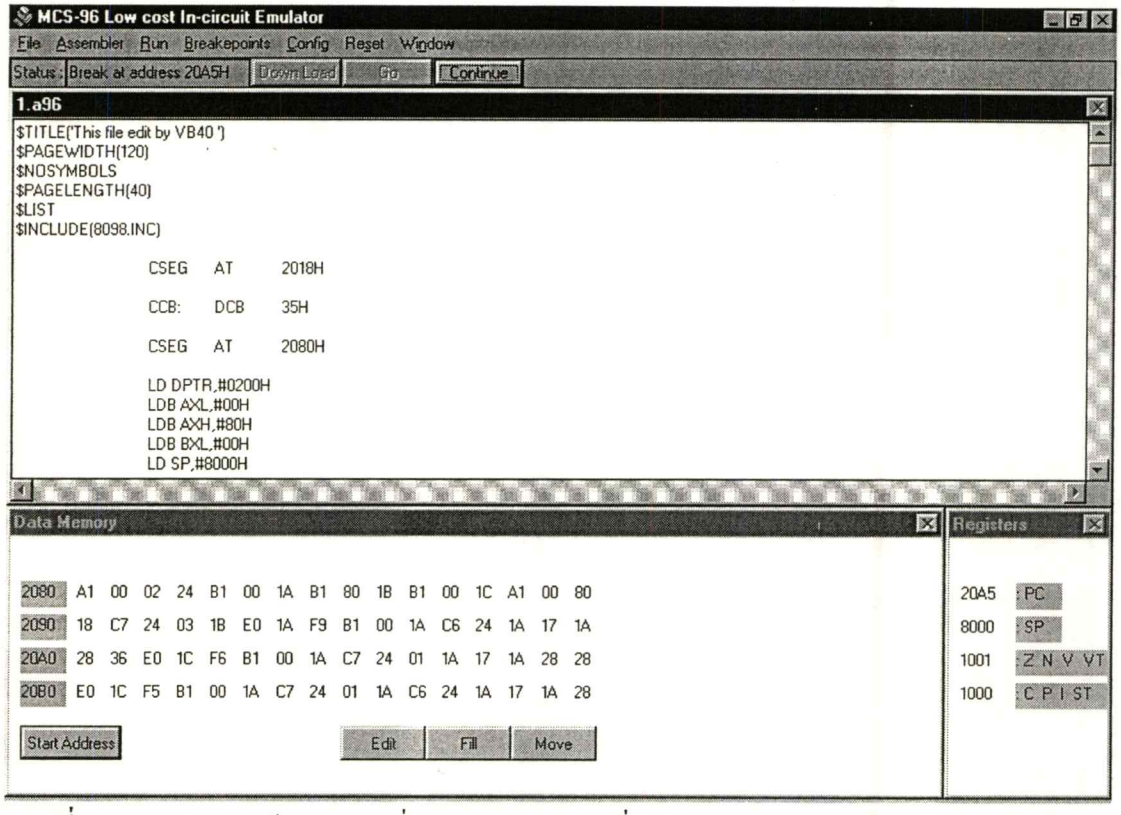


ภาพที่ 4.56 แสดงการใช้งานปุ่ม Go

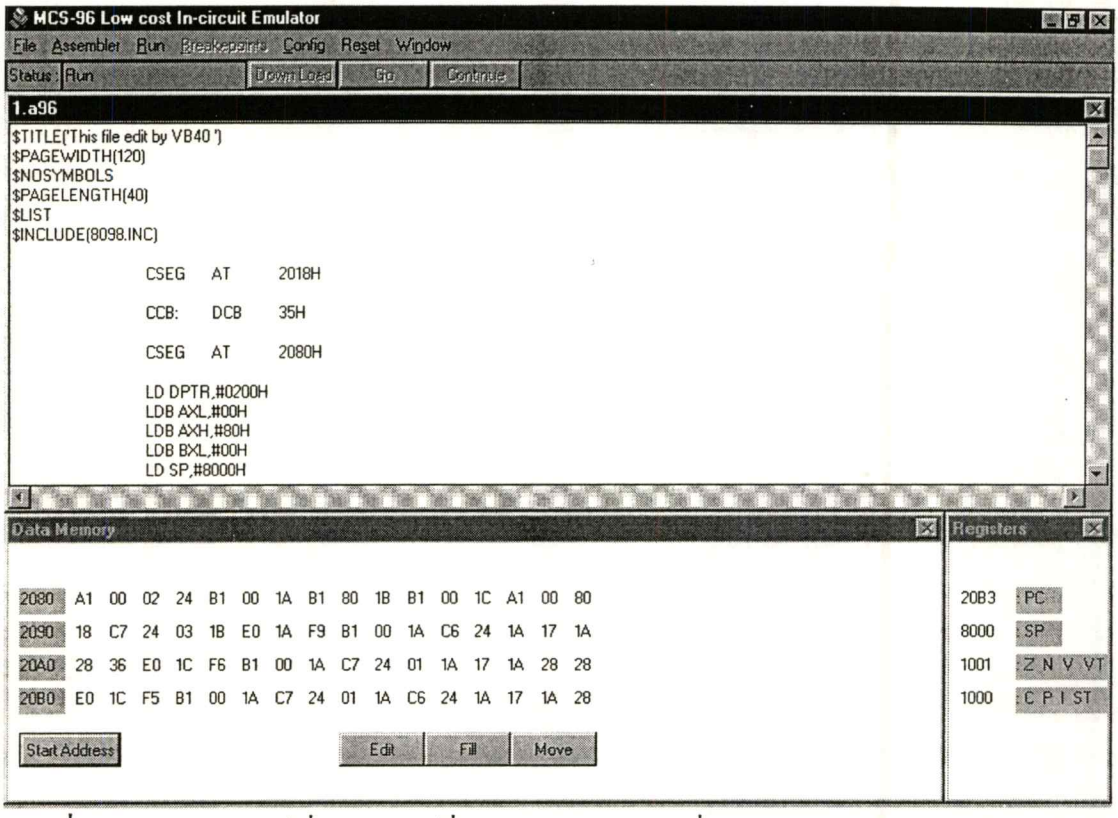


ภาพที่ 4.57 แสดงตัวอย่างหน้าต่างวิธีการหยุดการทำงานของอินเซอร์กิตอีมูเลเตอร์ เมื่อใช้งานคำสั่ง Go ใน รูปแบบการทำงานแบบ Go forever และ Go from... forever ตามลำดับ

4.5.26 คำสั่ง Continue เมื่อมีการกำหนดเบรคแอดเดรส และอินเซอร์กิตอีมูเลเตอร์ทำงานจนถึงคำสั่ง ณ ตำแหน่งนั้น อินเซอร์กิตอีมูเลเตอร์จะหยุดทำงาน ภายในกรอบ Status : จะแสดงตำแหน่งของแอดเดรสที่หยุดการทำงาน เมื่อผู้ใช้งานต้องการสั่งให้อินเซอร์กิตอีมูเลเตอร์ทำงานต่อไปก็สามารถทำได้ โดยการคลิกเมาส์ที่ปุ่ม Continue อินเซอร์กิตอีมูเลเตอร์ก็จะเริ่มทำงานตามคำสั่งของแอดเดรสที่หยุดไว้นั้น และจะหยุดการทำงานอีกครั้ง ณ ตำแหน่งของเบรคแอดเดรสต่อไป

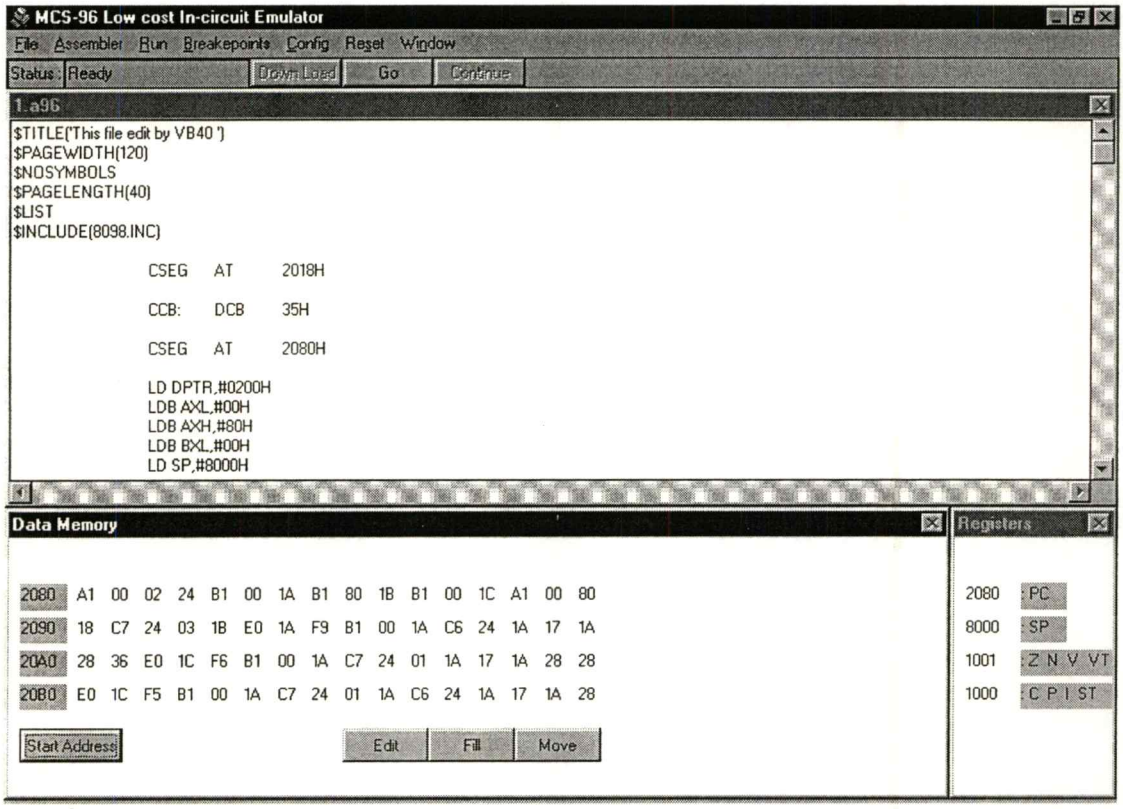


ภาพที่ 4.58 แสดงการใช้งานปุ่มคำสั่ง Continue สังเกตุที่กรอบ Status : จะแสดงตำแหน่งของแอดเดรสที่กำลังหยุดอยู่

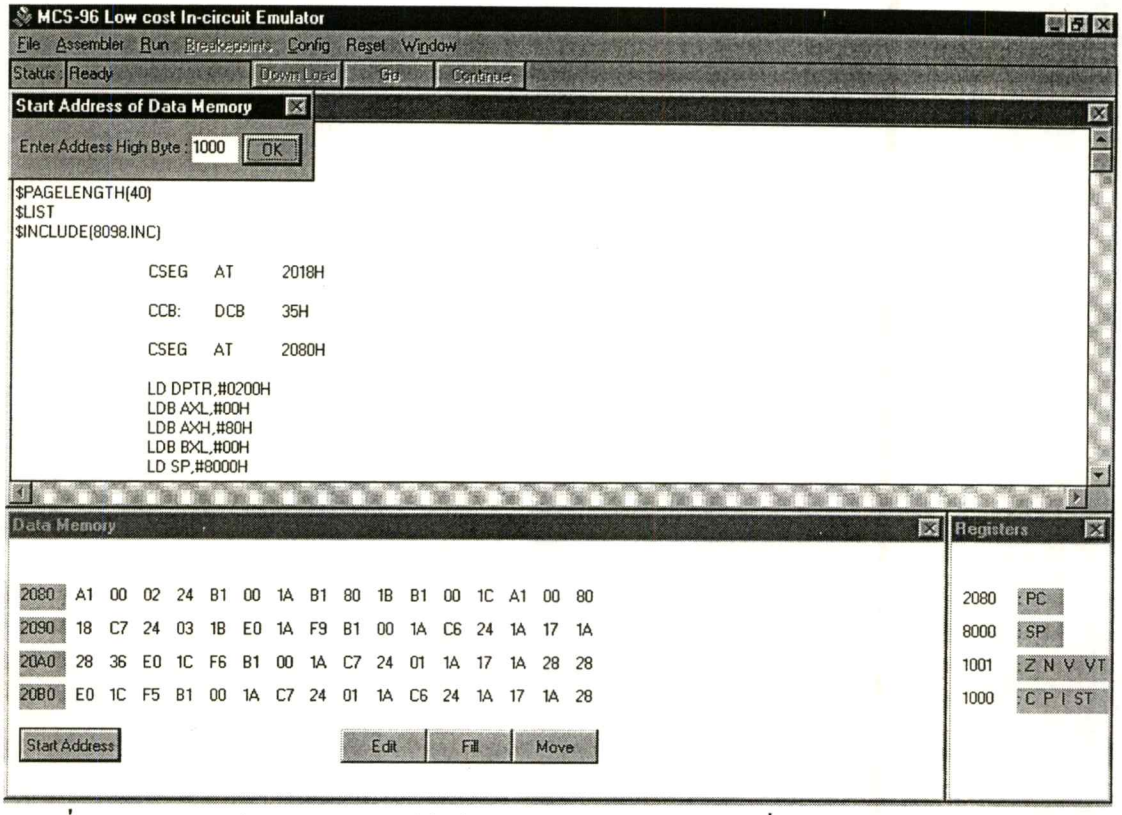


ภาพที่ 4.59 แสดงผลลัพธ์เมื่อคลิกเมาส์ที่ปุ่ม Continue สังเกตุที่กรอบ Status : จะแสดงสภาวะ Run นั่นคือ อินเซอร์กิตอีมูเลเตอร์เริ่มทำงานตามคำสั่งต่อไป

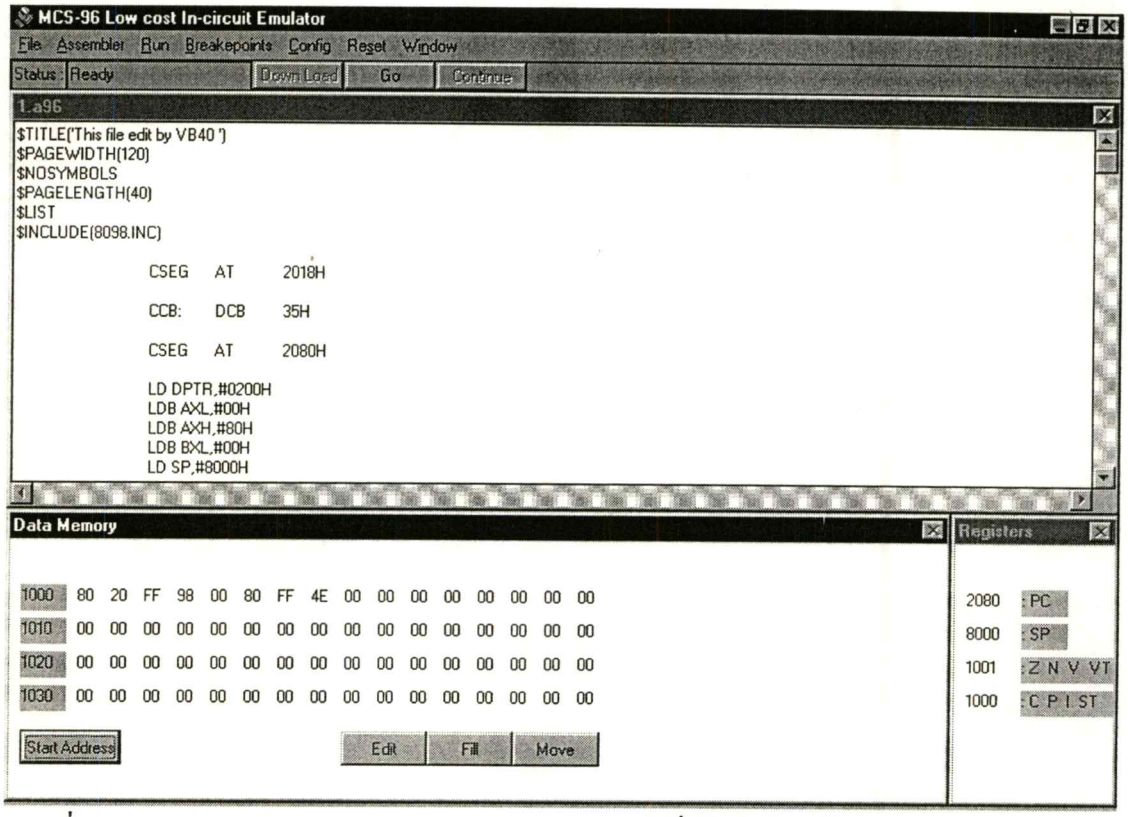
4.5.27 คำสั่ง Start Address ปุ่มนี้เป็นส่วนประกอบของหน้าต่าง Data Memory เมื่อคลิกเมาส์ที่คำสั่งนี้โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดตำแหน่งแอดเดรสเริ่มต้นของข้อมูลในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์ ซึ่งผู้ใช้ต้องการแสดงบนหน้าต่าง Data Memory โดยสามารถแสดงข้อมูลได้ครั้งละ 64 ไบต์



ภาพที่ 4.60 แสดงการใช้คำสั่ง Start Address ในหน้าต่าง Data Memory

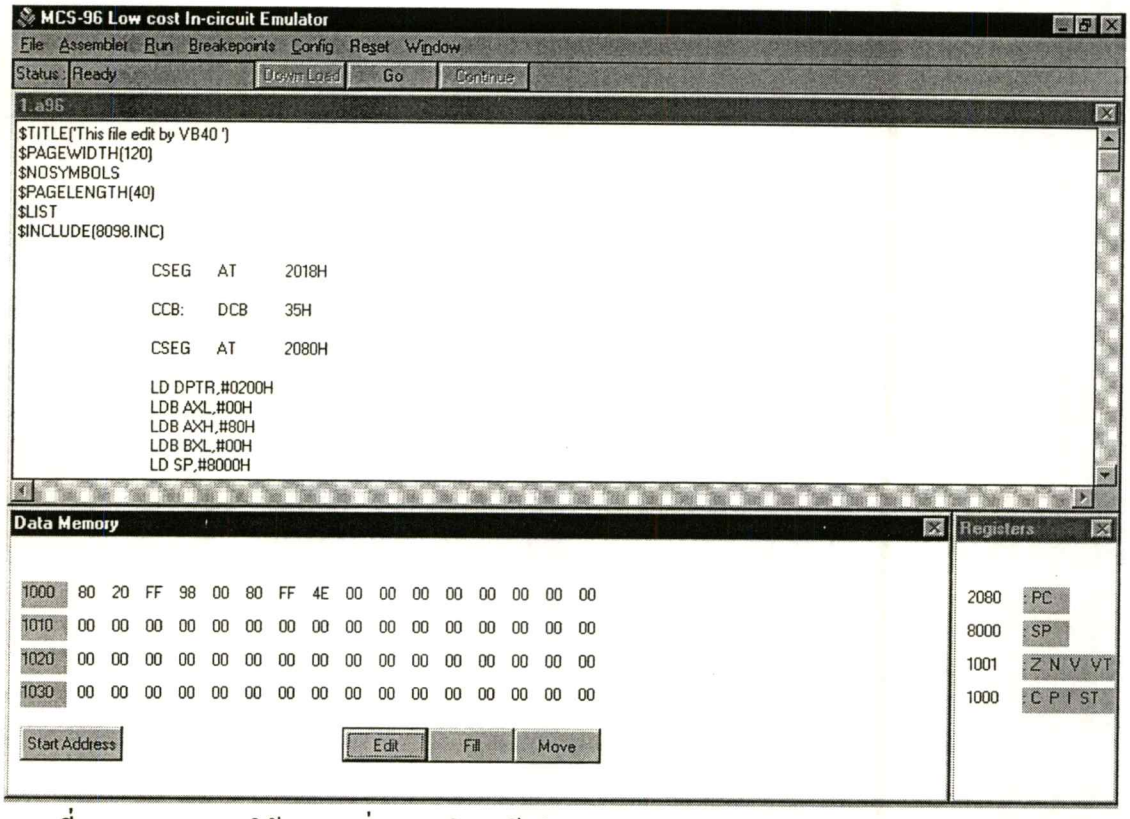


ภาพที่ 4.61 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดตำแหน่งแอดเดรสเริ่มต้นของข้อมูลในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์ ในตัวอย่างนี้แอดเดรสเริ่มต้นกำหนดให้เป็น 1000H

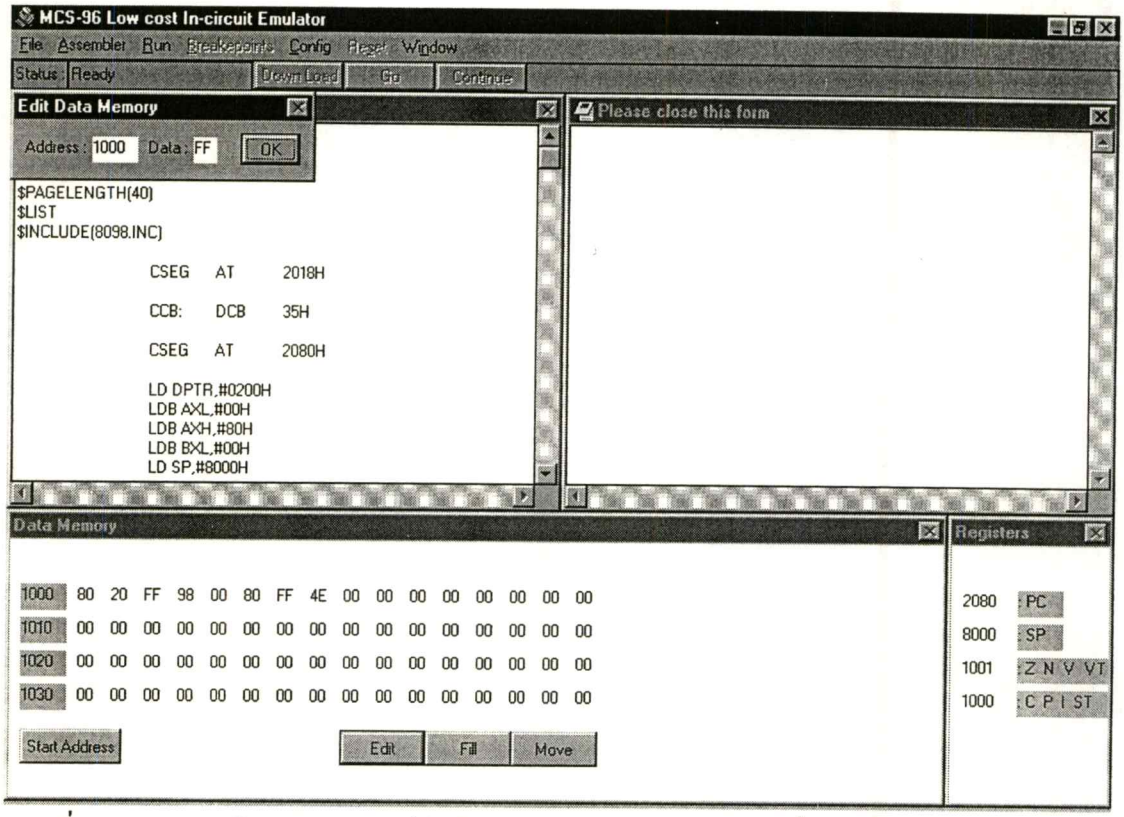


ภาพที่ 4.62 แสดงข้อมูลในหน้าต่าง Data Memory หลังจากที่ได้ทำการกำหนดตำแหน่งแอดเดรสเริ่มต้นเป็น 1000H โดยใช้คำสั่ง Start Address

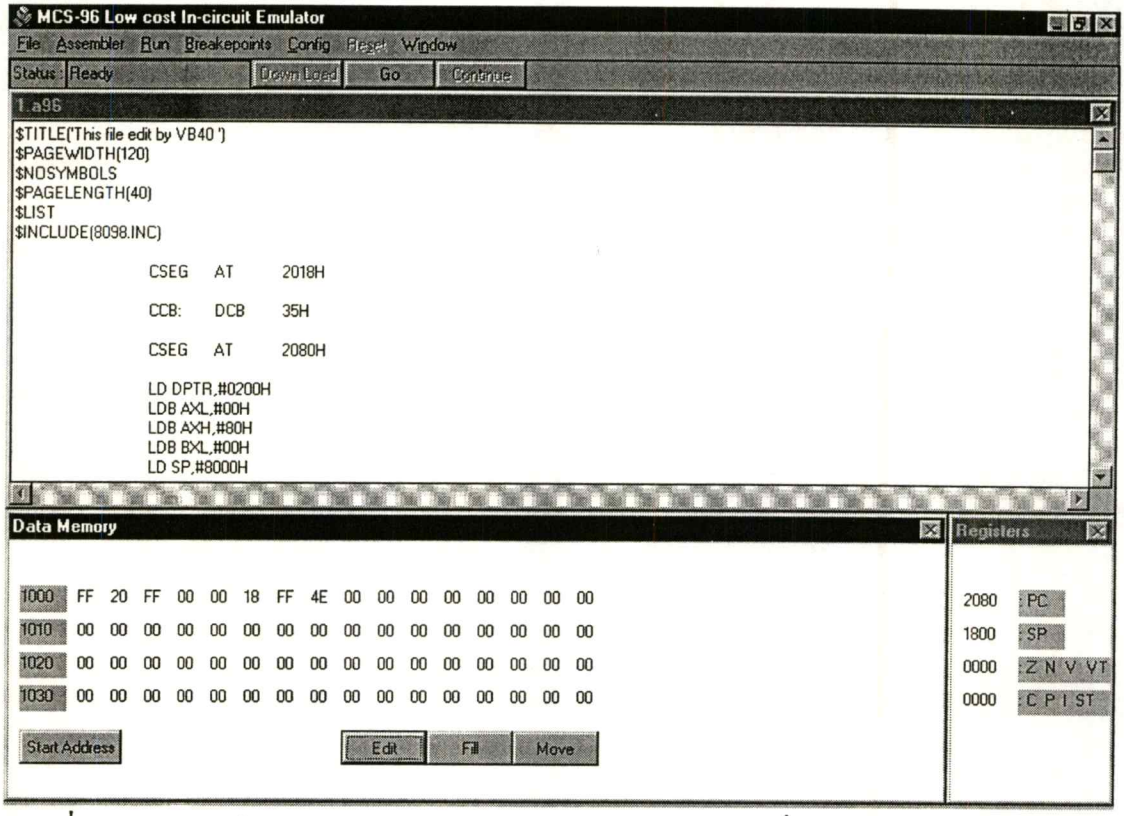
4.5.28 คำสั่ง Edit ปุ่มนี้เป็นส่วนประกอบของหน้าต่าง Data Memory เมื่อคลิกเมาส์ที่คำสั่งนี้โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสและข้อมูลที่ต้องการแก้ไขในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์ โดยผู้ใช้สามารถทำการแก้ไขข้อมูลได้ครั้งละ 1 ไบต์



ภาพที่ 4.63 แสดงการใช้งานคำสั่ง Edit ในหน้าต่าง Data Memory

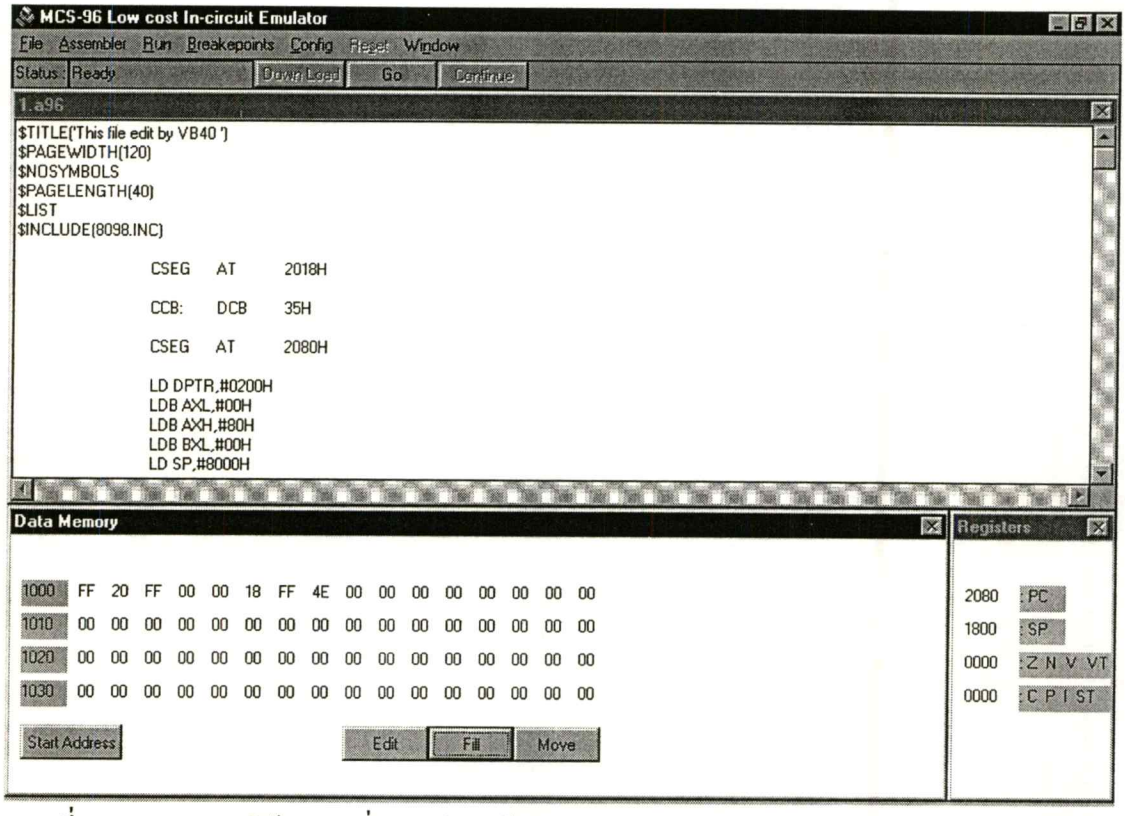


ภาพที่ 4.64 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรส และข้อมูลที่ต้องการแก้ไขในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์ สำหรับตัวอย่างนี้เป็นการแก้ไขข้อมูลที่แอดเดรส 1000H จาก 80H เป็น 0FFH

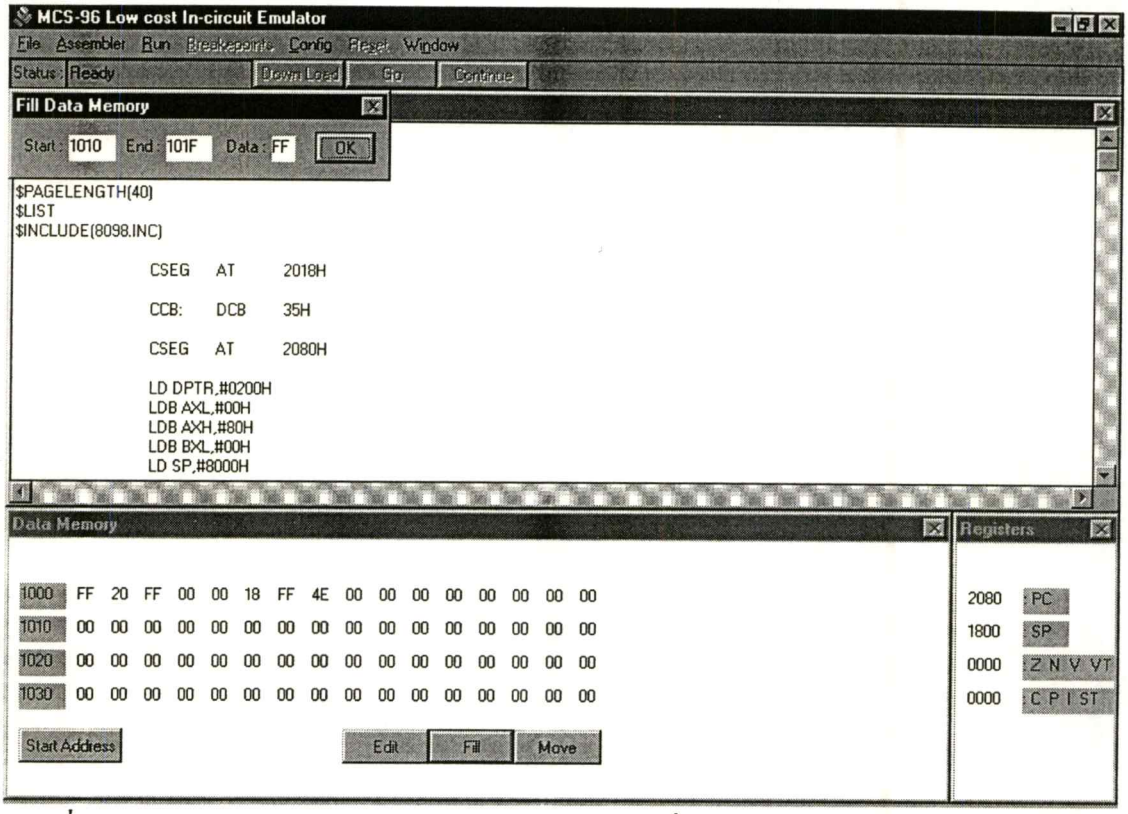


ภาพที่ 4.65 แสดงหน้าต่าง Data Memory หลังจากทำการแก้ไขข้อมูลที่แอดเดรส 1000H โดยใช้คำสั่ง Edit

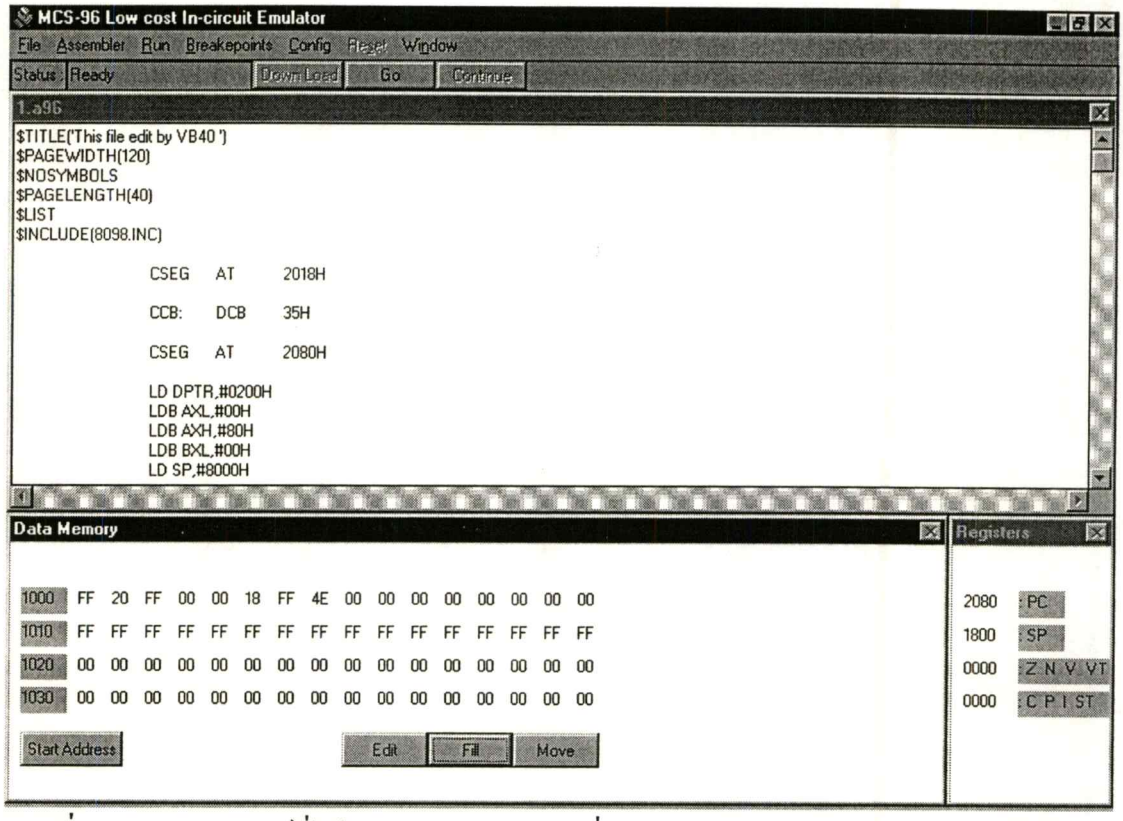
4.5.29 คำสั่ง Fill ปุ่มนี้เป็นส่วนประกอบของหน้าต่าง Data Memory เมื่อคลิกเมาส์ที่คำสั่งนี้โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น, แอดเดรสสุดท้าย และข้อมูลที่ต้องการเขียนลงในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์



ภาพที่ 4.66 แสดงการใช้งานคำสั่ง Fill ในหน้าต่าง Data Memory

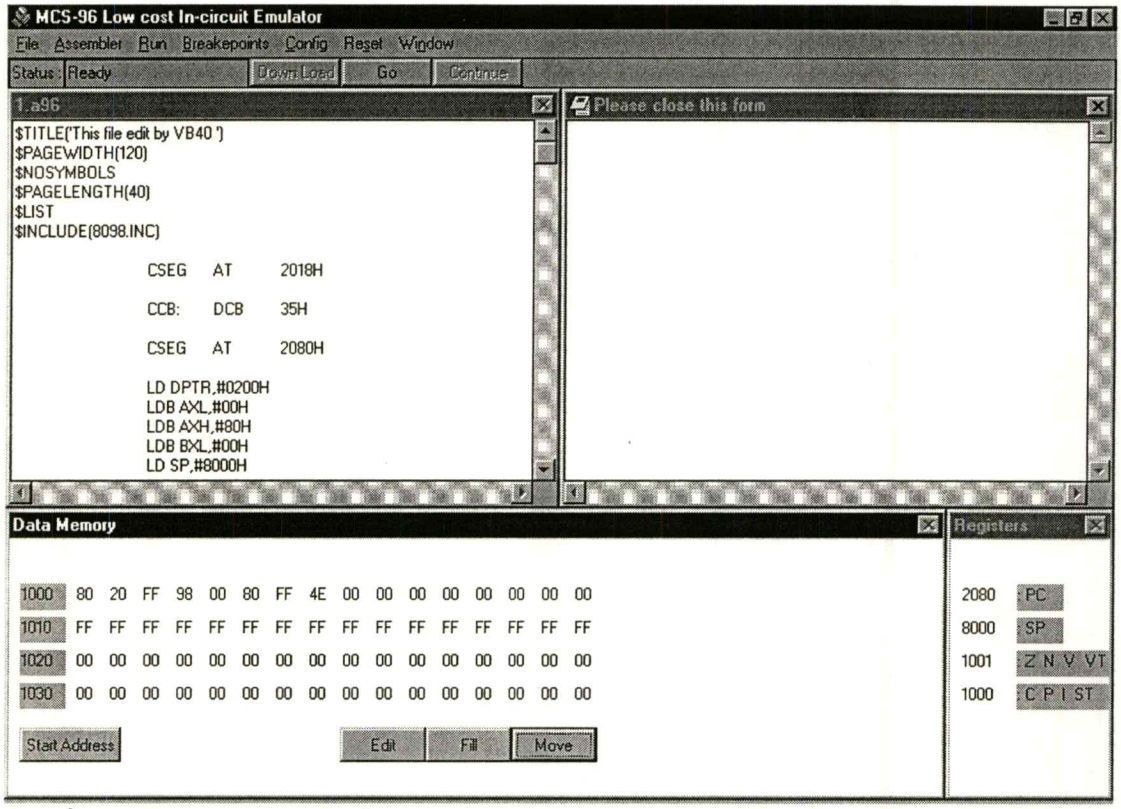


ภาพที่ 4.67 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น, แอดเดรสสุดท้าย และข้อมูลที่ต้องการเขียนลงในหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์ ในตัวอย่างนี้เป็นการเขียนข้อมูล FFH ลง ในหน่วยความจำที่แอดเดรสตำแหน่ง 1010H จนถึง 101FH

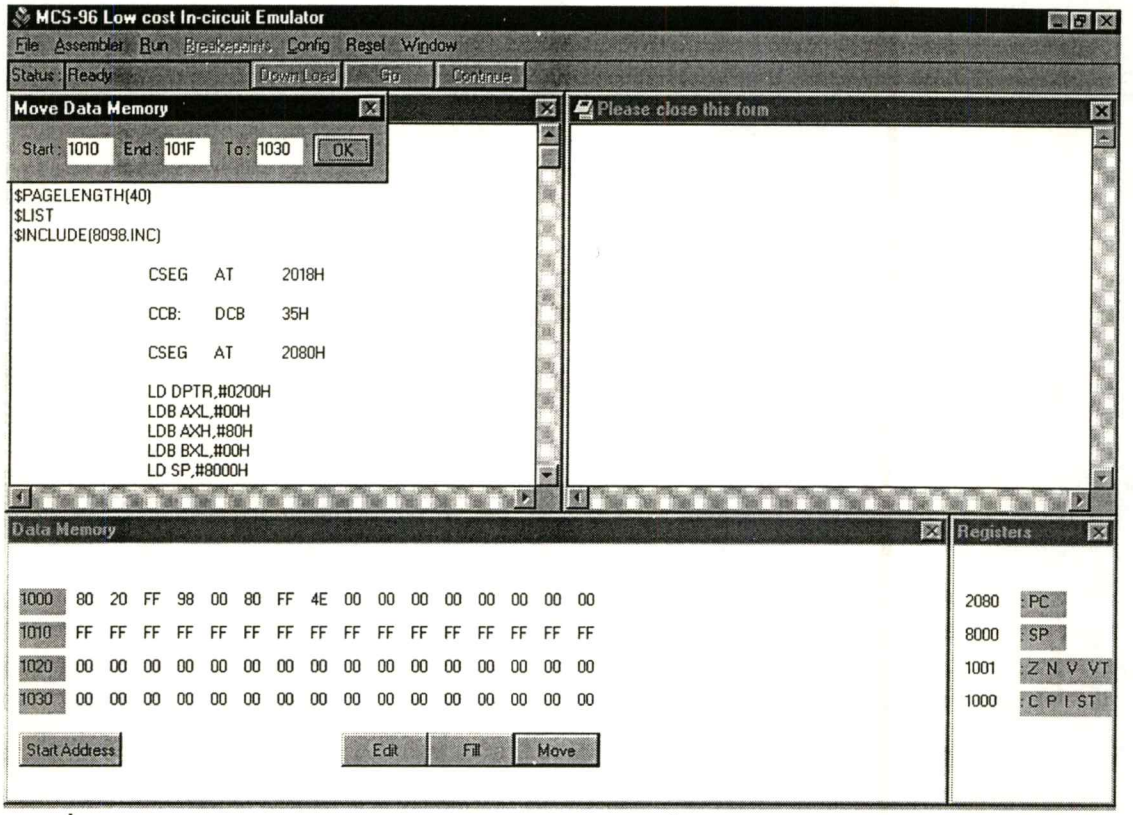


ภาพที่ 4.68 แสดงผลลัพธ์ที่ได้จากการทำงานของคำสั่ง Fill ภายในหน้าต่าง Data Memory ข้อมูลระหว่างแอดเดรส 1010H ถึง 101FH มีค่าเป็น FFH

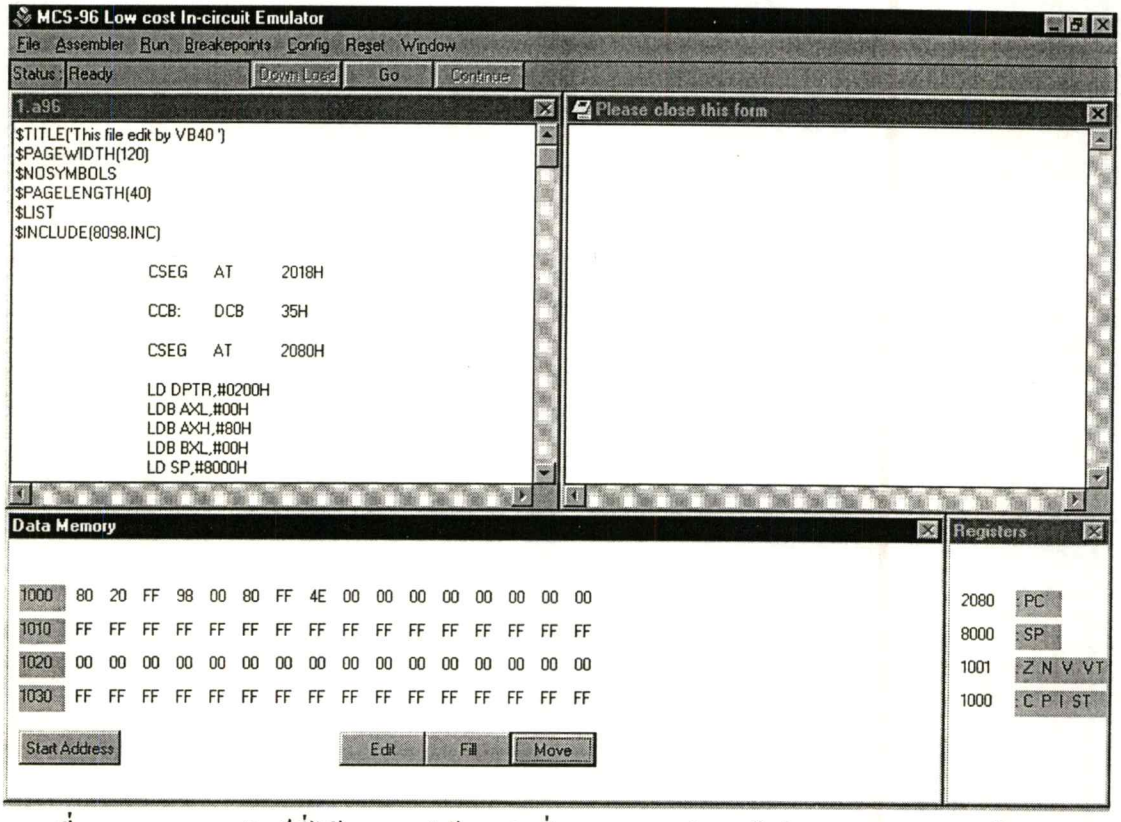
4.5.30 คำสั่ง Move ปุ่มนี้เป็นส่วนประกอบของหน้าต่าง Data Memory เมื่อคลิกเมาส์ที่คำสั่งนี้โปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น, แอดเดรสสุดท้าย และแอดเดรสปลายทางของหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์ที่ผู้ใช้ต้องการเคลื่อนย้ายข้อมูลเป็นกลุ่ม



ภาพที่ 4.69 แสดงการใช้งานคำสั่ง Move ภายในหน้าต่าง Data Memory

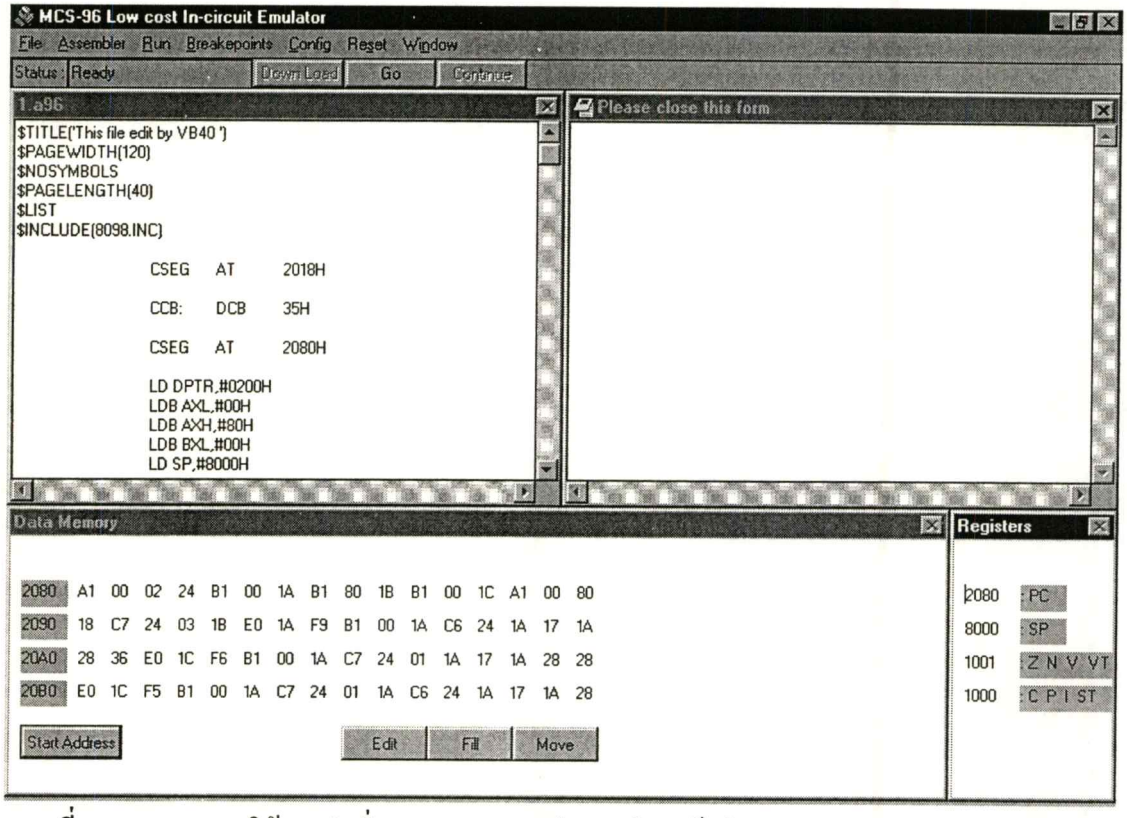


ภาพที่ 4.70 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดแอดเดรสเริ่มต้น, แอดเดรสสุดท้าย และแอดเดรสปลายทางของหน่วยความจำภายในของอินเซอร์กิตอีมูเลเตอร์ ในตัวอย่างนี้เป็นการเคลื่อนย้ายข้อมูลที่อยู่ระหว่างแอดเดรส 1010H ถึง 101FH ไปยังแอดเดรสปลายทางที่ตำแหน่ง 1030H

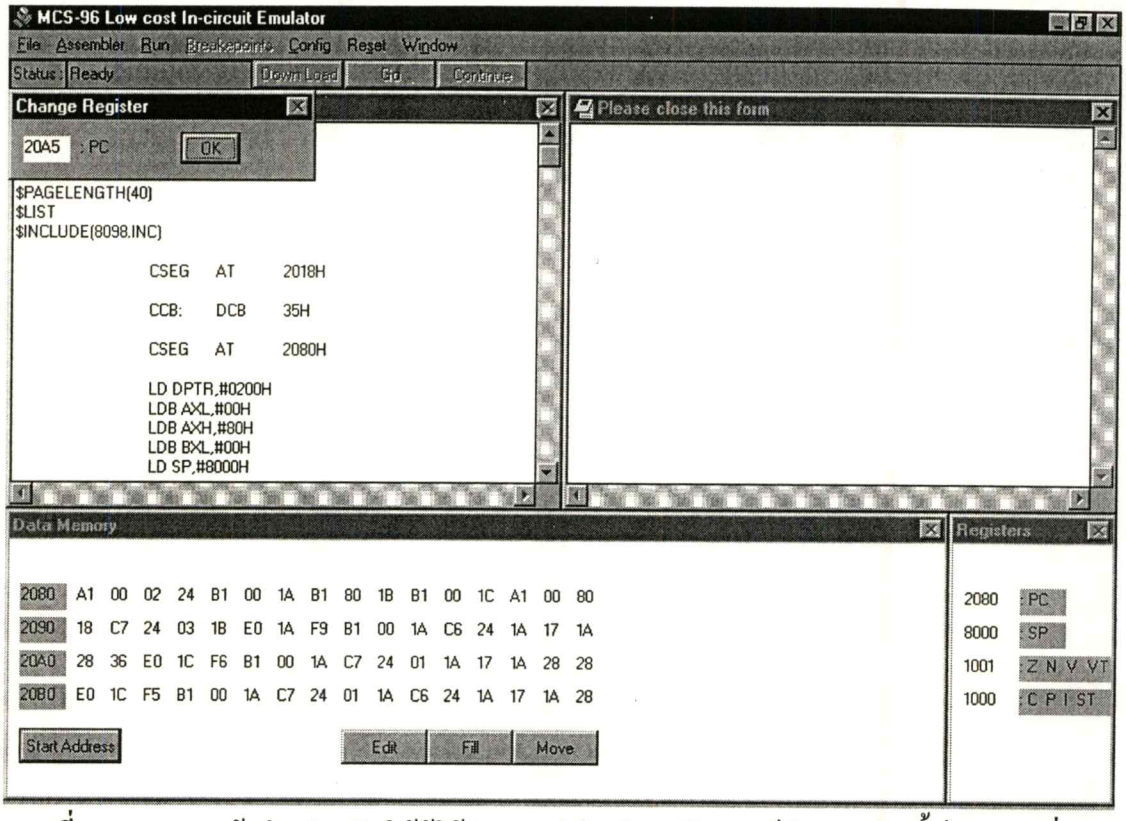


ภาพที่ 4.71 แสดงผลลัพธ์ที่ได้จากการใช้งานคำสั่ง Move ภายในหน้าต่าง Data Memory ข้อมูลระหว่างแอดเดรส 1030H ถึง 103FH มีค่าเป็น FFH

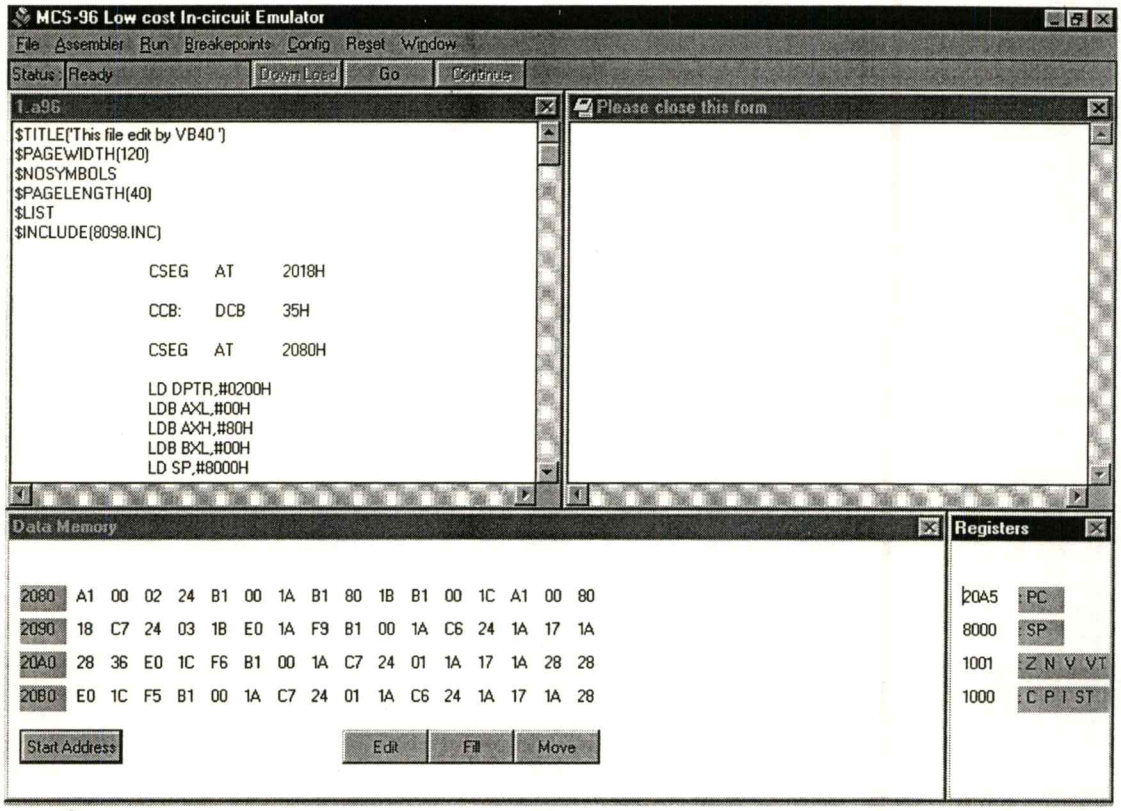
4.5.31 คำสั่ง Change Resister คำสั่งนี้เป็นส่วนประกอบของหน้าต่าง Registers ในกรณีที่ผู้ใช้ต้องการเปลี่ยนแปลงค่าในรีจิสเตอร์ต่างๆ ของอินเซอร์กิตอีมูเลเตอร์นั้น สามารถทำได้โดยการคลิกเมาส์บนค่าปัจจุบันของรีจิสเตอร์ที่ต้องการ จากนั้นโปรแกรม ICE8098 จะเปิดหน้าต่างสำหรับให้ผู้ใช้กำหนดค่าใหม่ของรีจิสเตอร์นั้นๆ



ภาพที่ 4.72 แสดงการใช้งานคำสั่ง Change Register ภายในหน้าต่าง Registers



ภาพที่ 4.73 แสดงหน้าต่างสำหรับให้ผู้ใช้กำหนดค่าใหม่ของรีจิสเตอร์ ในตัวอย่างนี้เป็นการเปลี่ยนแปลงค่าของตัวนับโปรแกรมจาก 2080H ไปเป็น 20A5H



ภาพที่ 4.74 แสดงผลลัพธ์ที่ได้จากการใช้งานคำสั่ง Change Register ภายในหน้าต่าง Registers ตัวนับโปรแกรมมีค่าเป็น 20A5H

บทที่ 5

การประยุกต์ใช้งาน

ในบทนี้จะกล่าวถึงตัวอย่างการนำเอาอินเซอร์กิตอีมูเลเตอร์ไปประยุกต์ใช้งาน โดยเน้นให้เห็นถึงประโยชน์และข้อดีที่ได้รับเมื่อทำการพัฒนาโปรแกรมโดยใช้อินเซอร์กิตอีมูเลเตอร์

5.1 ตัวอย่างที่ 1. ใช้ประกอบการศึกษาวิธีการกำหนดเลขที่อยู่ในโหมดต่างๆ (Addressing Modes) ของไมโครคอนโทรลเลอร์ตระกูล MCS-96

การกำหนดเลขที่อยู่ของไมโครคอนโทรลเลอร์ตระกูล MCS-96 นั้นมีด้วยกันหลายโหมดได้แก่ [13]

- โหมดการกำหนดเลขที่อยู่โดยตรง (Direct Addressing Mode) ตัวอย่างเช่น

```
ADD     AX,BX,CX    ;AX:=BX+CX
MUL     AX,BX       ;AX:=AX*BX
INCB    CL          ;CL:=CL+1
```

- โหมดการกำหนดเลขที่อยู่โดยอ้อม (Indirect Addressing Mode) ตัวอย่างเช่น

```
LD      AX,[AX]     ;AX:=MEM_WORD (AX)
ADDB    AL,BL,[CX]  ;AL:=BL+MEM_BYTE (CX)
POP     [AX]        ;MEM_WORD (AX) :=MEM_WORD (SP)
                    ;SP:=SP+2
```

- โหมดการกำหนดเลขที่อยู่โดยทันที (Immediate Addressing Mode) ตัวอย่างเช่น

```
ADD     AX,#340     ;AX:=AX+340
PUSH    #1234H      ;SP:=SP-2 ;MEM_WORD (SP) :=1234H
DIVB    AX,#10      ;AL:=AX/10 ;AH:=AX MOD 10
```

- โหมดการกำหนดเลขที่อยู่โดยอ้อมที่มีการเพิ่มค่าอัตโนมัติ (Indirect With Auto-increment Addressing Mode) ตัวอย่างเช่น

```
LD      AX,[BX]+    ;AX:=MEM_WORD (BX) ;BX:=BX+2
ADDB    AL,BL,[CX]+ ;AL:=BL+MEM_BYTE (CX) ;CX:=CX+2
PUSH    [AX]+       ;SP:=SP-2
                    ;MEM_WORD (SP):=MEM_WORD (AX)
                    ;AX:=AX+2
```

- โหมดการกำหนดเลขที่อยู่โดยใช้ดรรชนี (Indexed Addressing Mode) ตัวอย่างเช่น

```
LD      AX,12[BX]          ;AX:=MEM_WORD (BX+12)
MULB    AX,BL,3[CX]        ;AX:=BL*MEM_BYTE (CX+3)
ST      AX,TABLE[BX]       ;MEM_WORD (TABLE+BX):=AX
```

- โหมดการกำหนดเลขที่อยู่โดยรีจีสเตอร์ศูนย์ (Zero Register Addressing Mode) ตัวอย่างเช่น

```
ADD     AX,1234[0]     ;AX:=AX+MEM_WORD (1234)
POP     5678[0]        ;MEM_WORD (5678):=MEM_WORD (SP)
                       ;SP:=SP+2
```

- โหมดการกำหนดเลขที่อยู่โดยตัวชี้สแต็ก (Stack Pointer Register Addressing Mode) ตัวอย่างเช่น

```
PUSH    [SP]           ;DUPLICATE TOP_OF_STACK
LD      AX,2[SP]       ;AX:=NEXT_TO_TOP
```

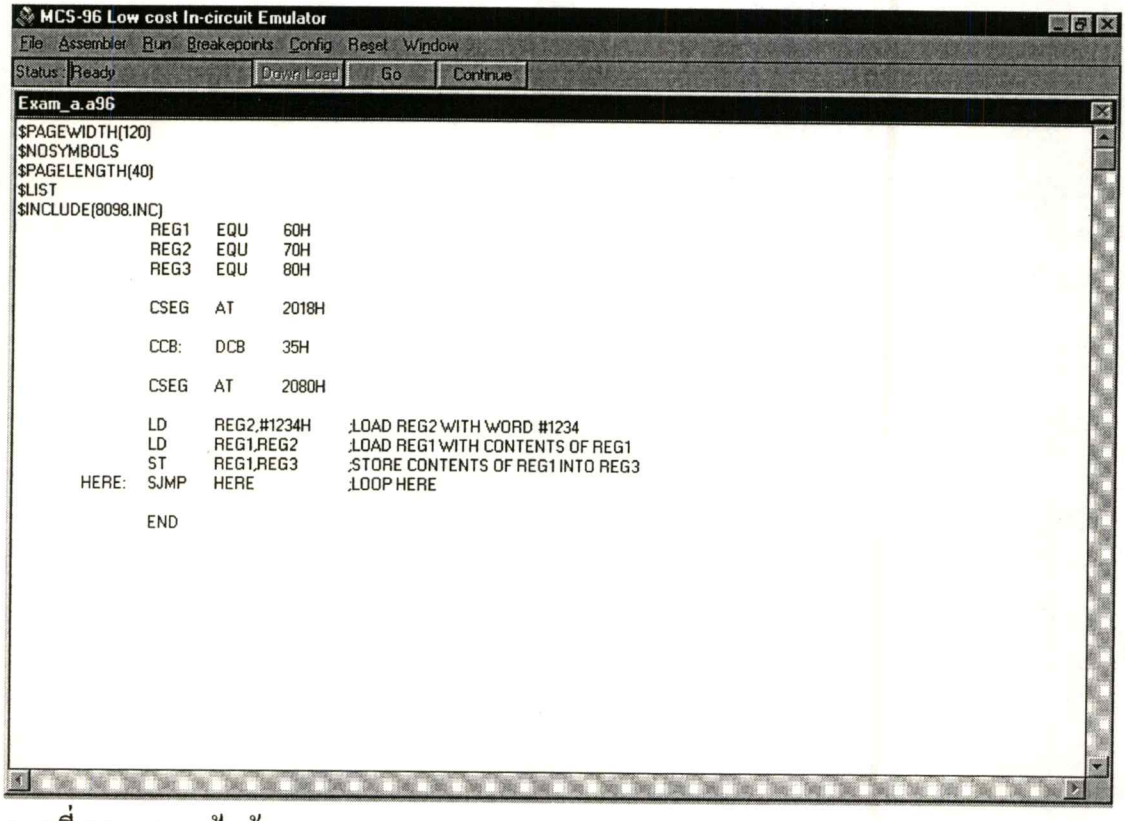
ในตัวอย่างที่ 1 นี้จะขอกล่าวถึงการศึกษาโหมดการกำหนดเลขที่อยู่โดยตรง และโหมดการกำหนดเลขที่อยู่โดยทันที โดยใช้อินเซอร์กิตอีมูเลเตอร์ อธิบายได้โดยโปรแกรมต่อไปนี้

```
        REG1  EQU   60H
        REG2  EQU   70H
        REG3  EQU   80H
        CSEG  AT    2018H
        CCB:  DCB   35H
        CSEG  AT    2080H
        LD    REG2,#1234H  ;LOAD REG2 WITH WORD #1234
        LD    REG1,REG2    ;LOAD REG1 WITH CONTENTS OF REG2
        ST    REG1,REG3    ;STORE CONTENTS OF REG1 INTO REG3
HERE:   SJMP  HERE         ;LOOP HERE
        END
```

เมื่อพิจารณาโปรแกรมด้านบนพบว่ามีการทำงานดังนี้ ค่าตัวเลข #1234 จะถูกโหลดเข้าไปเก็บไว้ในเวิร์ดรีจีสเตอร์ REG2 (แสดงตัวอย่างโหมดการกำหนดเลขที่อยู่โดยทันที ด้วยคำสั่ง LD) จากนั้นจะถูกโหลดเข้าไปเก็บไว้ในเวิร์ดรีจีสเตอร์ REG1 โดยผ่านทางเวิร์ดรีจีสเตอร์ REG2 (แสดงตัวอย่างโหมดการกำหนดเลขที่อยู่โดยตรง ด้วยคำสั่ง LD) และทำการโหลดเข้าไปเก็บไว้ในเวิร์ดรีจีสเตอร์ REG3 ผ่านทางเวิร์ดรีจิสเตอร์ REG1 (แสดงตัวอย่างโหมดการกำหนดเลขที่อยู่โดยตรงด้วยคำสั่ง ST) และในคำสั่งสุดท้ายโปรแกรมจะทำงานเป็นลูปโดยจะกระโดดซ้ำอยู่ที่ตำแหน่ง

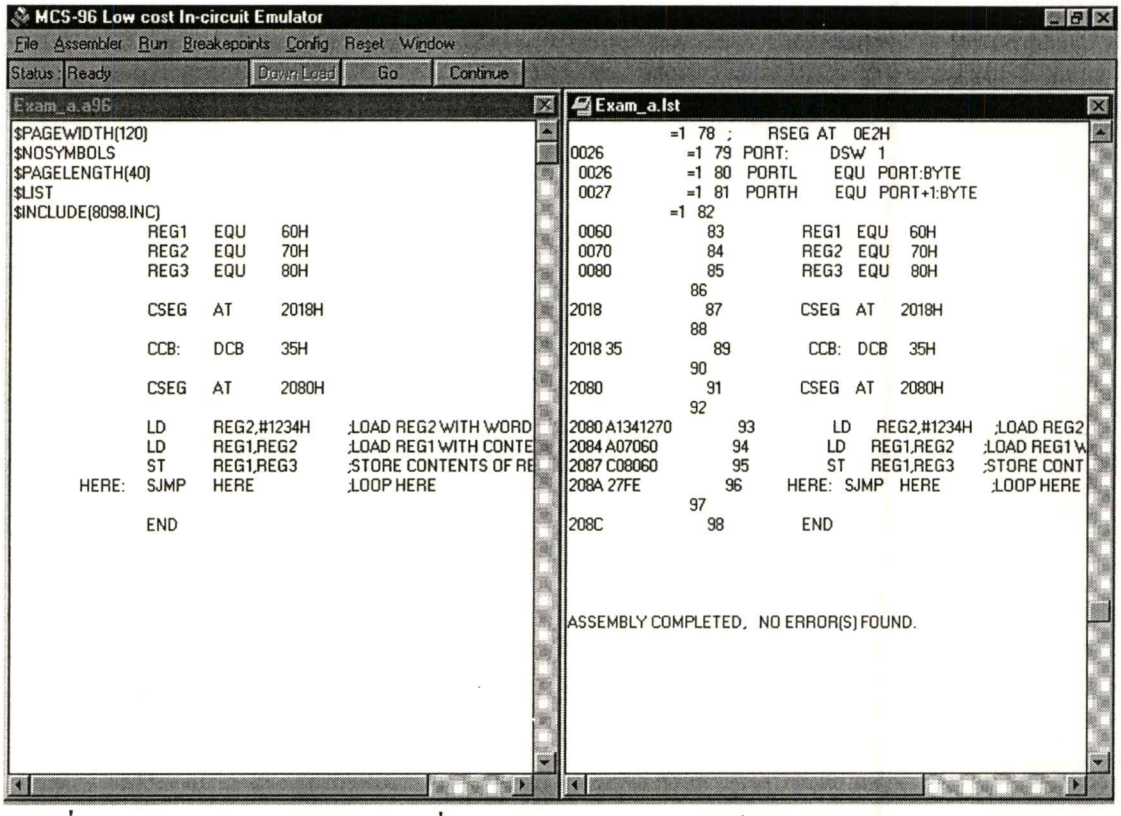
HERE การกำหนดเลขที่อยู่โดยทันทีนั้นโอเปอร์แรนด์จะประกอบด้วยค่าตัวเลข ในขณะที่การกำหนดเลขที่อยู่โดยตรงจะใช้วิธีการส่งผ่านค่าระหว่างรีจิสเตอร์ต่างๆ ณ ตำแหน่งใดก็ได้ภายในรีจีสเตอร์ไฟล์ที่มีอยู่จำนวน 256 รีจิสเตอร์ เพื่อให้ง่ายต่อการเข้าใจเราจะนำเอาอินเซอร์กิตอีมูเลเตอร์มาประกอบการศึกษาโดยการใช้งานคำสั่งต่างๆ ของอินเซอร์กิตอีมูเลเตอร์ที่มีอยู่ ทำให้เราสามารถมองเห็นความเป็นไปหรือการทำงานที่เกิดขึ้นภายในตัวไมโครคอนโทรลเลอร์ได้อย่างชัดเจน มีขั้นตอนการใช้งานดังนี้

ขั้นตอนที่ 1 ใช้คำสั่ง New เพื่อทำการเปิดแฟ้มข้อมูลใหม่ จากนั้นเขียนโปรแกรมตามตัวอย่างด้านบน เมื่อเขียนเสร็จให้ทำการบันทึกแฟ้มข้อมูลด้วยคำสั่ง Save ในชื่อ Exam_a.a96



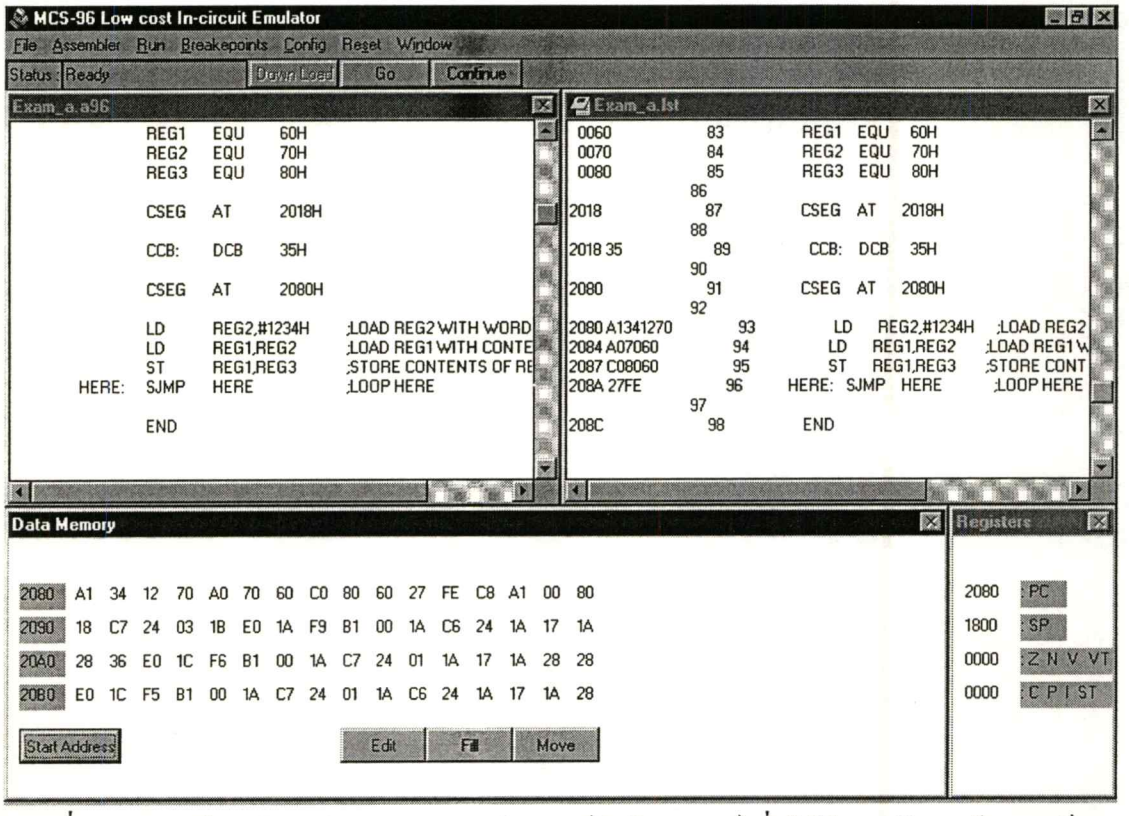
ภาพที่ 5.1 แสดงแฟ้มข้อมูล Exam_a.a96

ขั้นตอนที่ 2 ทำการแอสแซมเบลอร์โดยใช้คำสั่ง Assembler จากนั้นตรวจสอบความถูกต้อง และรายละเอียดต่างๆ ของโปรแกรมโดยใช้คำสั่ง Open List File ทำการเปิดแฟ้มข้อมูลแบบ List ขึ้นมา หากมีข้อผิดพลาดให้ทำการแก้ไขแล้วใช้คำสั่ง Save เพื่อบันทึกแฟ้มข้อมูลก่อนทำการแอสแซมเบลอร์ใหม่อีกครั้ง (ให้ทำซ้ำตามขั้นตอนนี้จนกว่าโปรแกรมไม่เกิดข้อผิดพลาด)



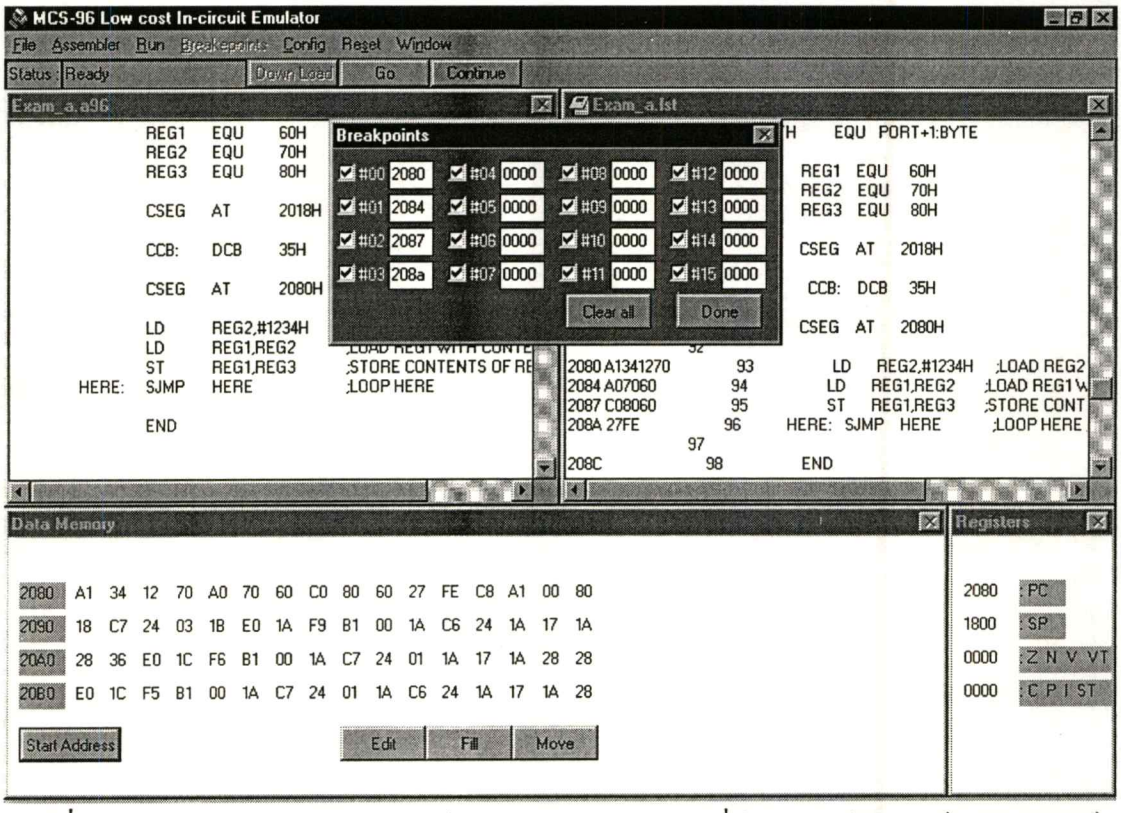
ภาพที่ 5.2 แสดงแฟ้มข้อมูลแบบ List ที่ได้จากการแอสแซมเบลอร์แฟ้มข้อมูล Exam_a.a96

ขั้นตอนที่ 3 ทำการแปลงแฟ้มข้อมูลแบบ Object ที่ได้จากการแอสแซมเบลอร์ให้เป็นแฟ้มข้อมูลแบบ Hex File โดยใช้คำสั่ง OH จากนั้นทำการโหลดข้อมูลให้กับอินเซอร์กิตอีมูเลเตอร์โดยใช้คำสั่ง Down Load และตรวจสอบข้อมูลที่เก็บไว้ในหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์โดยใช้คำสั่ง Data Memory & Register ซึ่งอยู่ภายในคำสั่ง Window โดยตำแหน่งของหน่วยความจำที่แสดงในหน้าต่าง Data Memory & Register จะเริ่มต้นที่แอดเดรส 2080H (ผู้ใช้สามารถเปลี่ยนแปลงได้ตามความต้องการ)



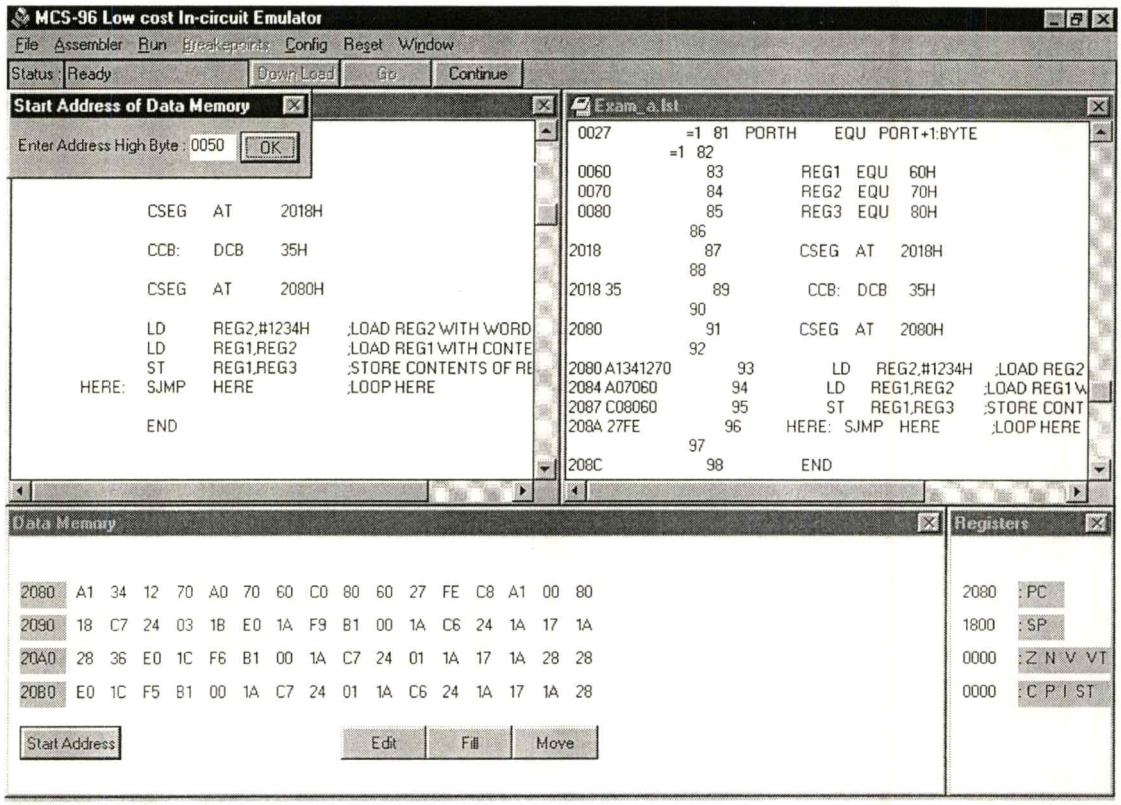
ภาพที่ 5.3 แสดงข้อมูลในหน่วยความจำของอินเซอร์กิตอีมูเลเตอร์ เมื่อได้รับการโหลดข้อมูลเสร็จสิ้นแล้ว โดยเริ่มจากแอดเดรส 2080H

ขั้นตอนที่ 4 กำหนดเบรคพอยน์แอดเดรสโดยใช้คำสั่ง Setup... ที่อยู่ภายในคำสั่ง Break points โดยอ้างอิงตำแหน่งของคำสั่งที่ต้องการหยุดได้จากแฟ้มข้อมูลแบบ List File ในตัวอย่างนี้ กำหนดให้อินเซอร์กิตอีมูเลเตอร์หยุดทำงานที่ตำแหน่งต่อไปนี้ 2080H, 2084H, 2087H และ 208AH เพื่อศึกษาการทำงานของแต่ละคำสั่ง



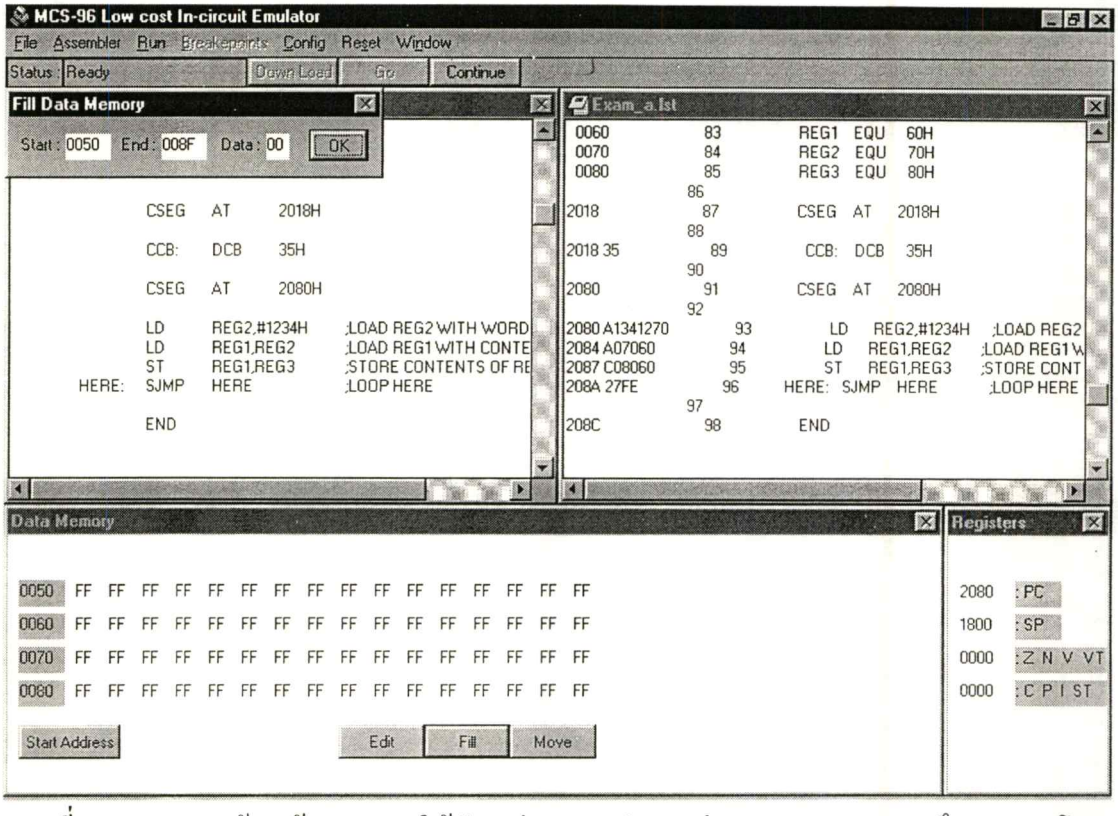
ภาพที่ 5.4 แสดงการกำหนดเบรคพอยน์แอดเดรส หรือตำแหน่งที่ต้องการให้อินเซอร์กิตอีมูเลเตอร์หยุดทำงาน

ขั้นตอนที่ 5 เนื่องจากโปรแกรมได้กำหนดตำแหน่งของรีจิสเตอร์ไฟล์ที่ใช้งานคือ REG1, REG2 และ REG3 ไว้ ณ แอดเดรส 0060H, 0070H และ 0080H ตามลำดับ ดังนั้นเราจำเป็นต้องเปลี่ยนแอดเดรสเริ่มต้นของหน่วยความจำภายในหน้าต่าง Data Memory จาก 2080H ให้เป็น 0050H โดยใช้คำสั่ง Start Address เพื่อให้ง่ายต่อการตรวจสอบผลการทำงานของโปรแกรม

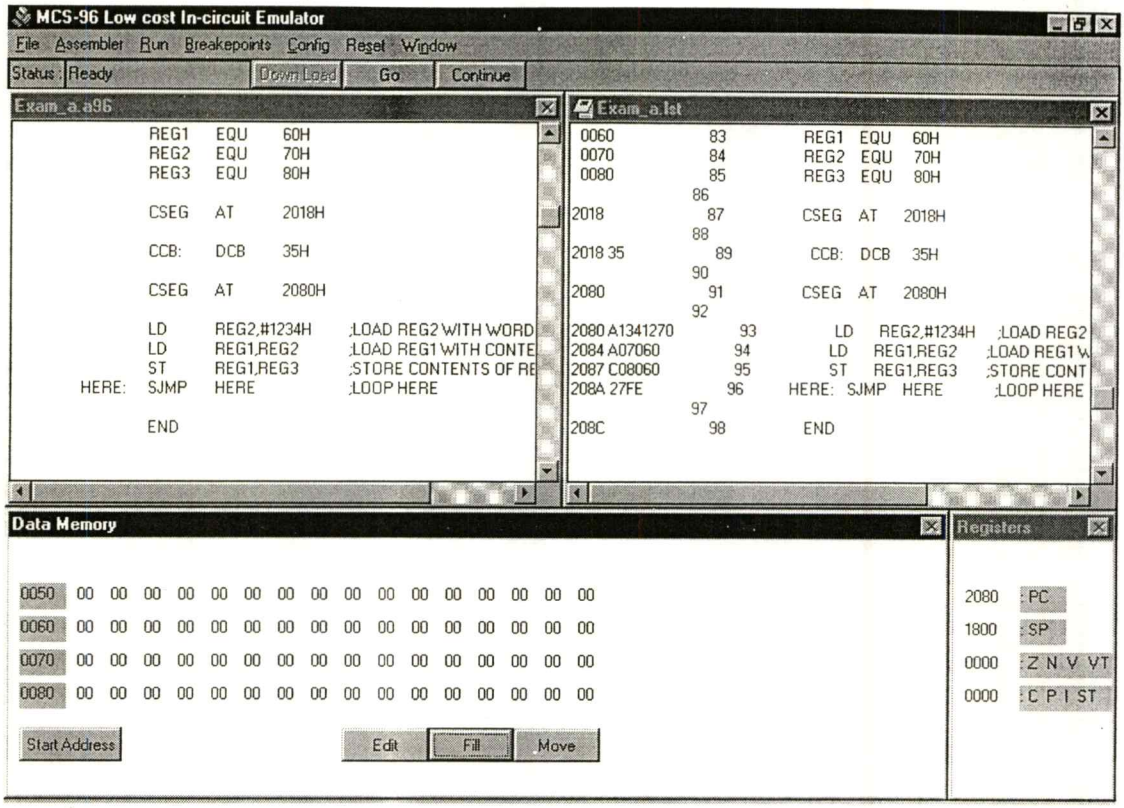


ภาพที่ 5.5 แสดงการเปลี่ยนตำแหน่งเริ่มต้นของหน่วยความจำภายในหน้าต่าง Data Memory ให้เป็น 0050H โดยใช้คำสั่ง Start Address

ขั้นตอนที่ 6 ทำการป้อนข้อมูล 00H ให้กับหน่วยความจำระหว่างแอดเดรส 0050H ถึง 008FH โดยใช้คำสั่ง Fill ทั้งนี้เพื่อให้ง่ายต่อการสังเกตุผลการทำงาน

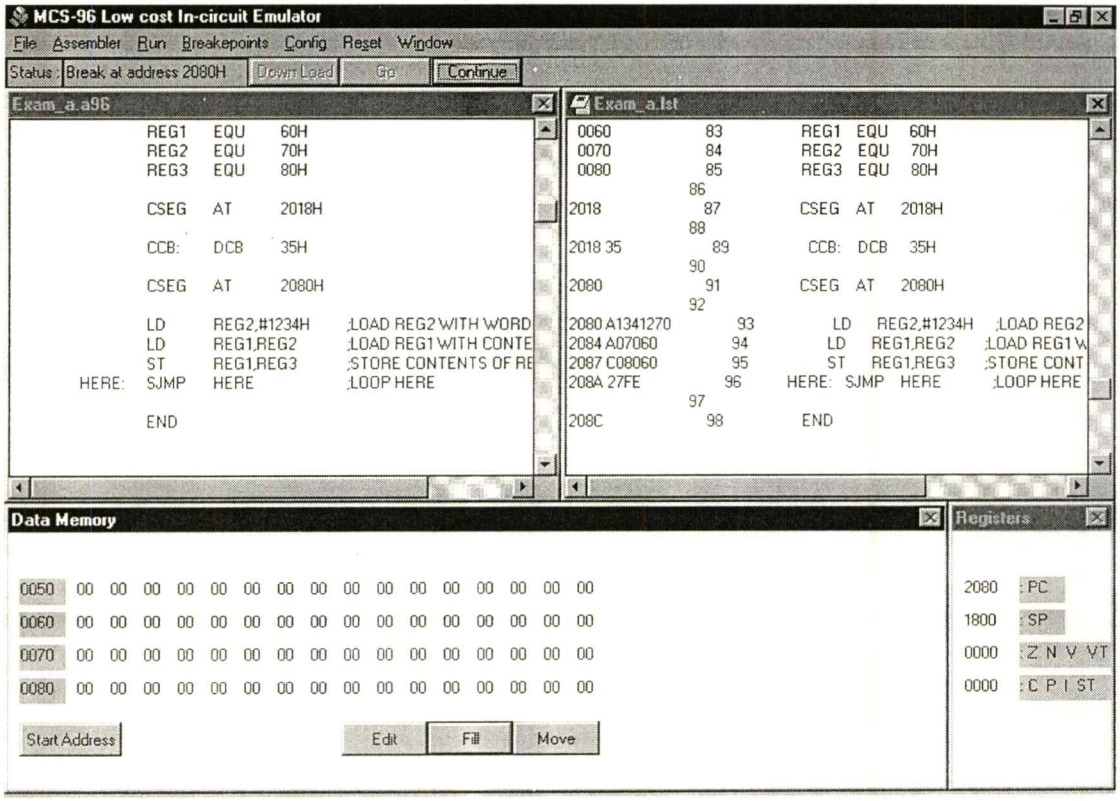


ภาพที่ 5.6 แสดงการป้อนข้อมูล 00H ให้กับหน่วยความจำระหว่างแอดเดรส 0050H ถึง 008FH โดยใช้คำสั่ง Fill



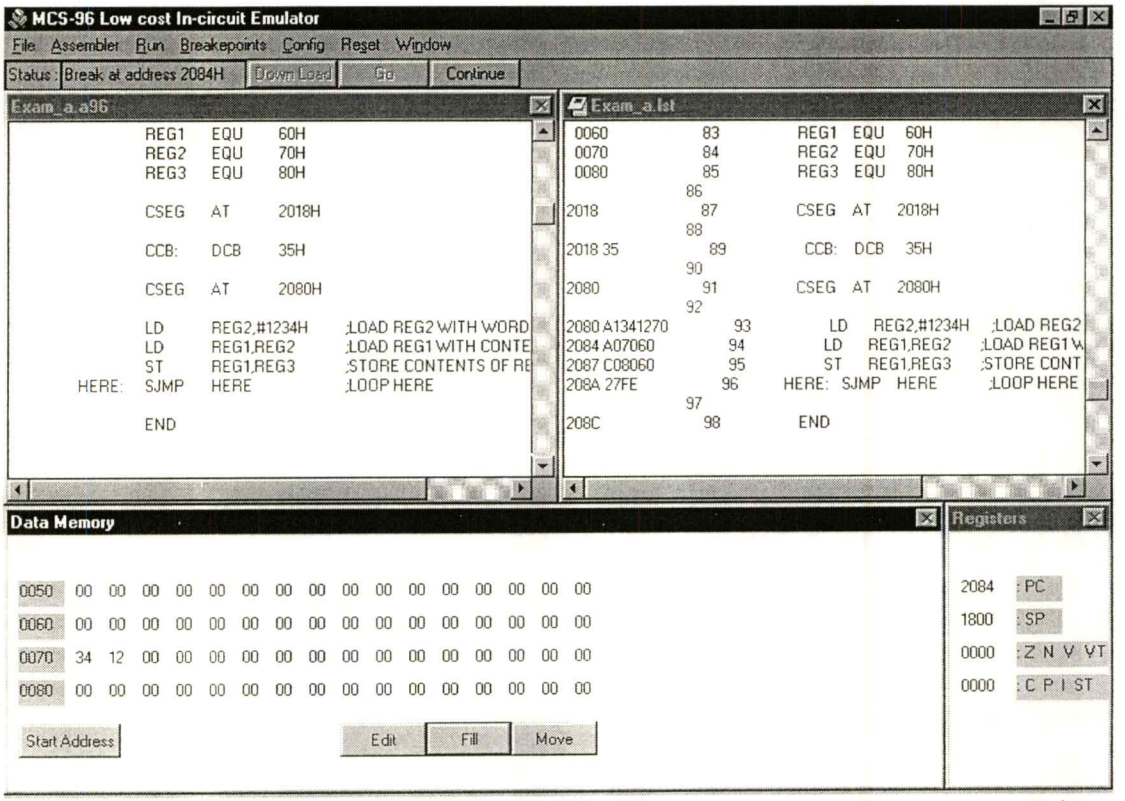
ภาพที่ 5.7 แสดงข้อมูลในหน้าต่าง Data Memory ภายหลังการใช้คำสั่ง Fill

ขั้นตอนที่ 7 จากนี้ไปจะเริ่มทำการศึกษาวิธีการกำหนดเลขที่อยู่โดยตรง และโดยทันที โดยใช้อินเซอร์กิตอีมูเลเตอร์ (ในขั้นตอนที่ 1 ถึง 6 ถือเป็นการเตรียมพร้อม) เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Go อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานตามโปรแกรมในตัวอย่างที่ 1 ให้สังเกตุที่ Status : จะเปลี่ยนเป็น Run และ Break at address 2080H ตามลำดับ ทั้งนี้เนื่องจากเราได้กำหนดให้อินเซอร์กิตอีมูเลเตอร์หยุดทำงานที่ตำแหน่ง 2080H (ตามที่กำหนดไว้ในขั้นตอนที่ 4) ค่าของตัวนับโปรแกรม (PC) ในหน้าต่าง Register มีค่าเป็น 2080H ซึ่งเป็นแอดเดรสแรกของคำสั่ง LD REG2,#1234H และข้อมูลในหน้าต่าง Data Memory ยังมีค่าเป็น 00H ทุกตำแหน่ง



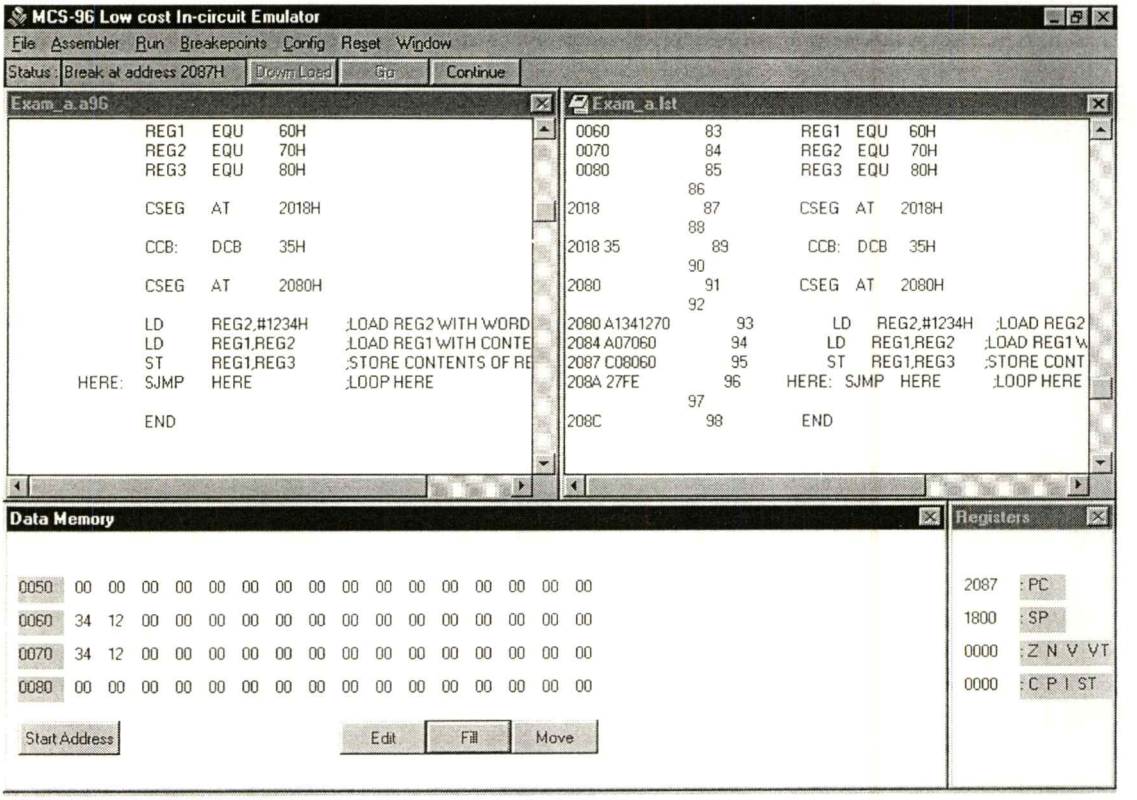
ภาพที่ 5.8 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Go

ขั้นตอนที่ 8 ทำการคลิกเมาส์ที่ปุ่ม Continue อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานต่อให้สังเกตุที่ Status : จะเปลี่ยนเป็น Run และ Break at address 2084H ตามลำดับ นั่นหมายถึงอินเซอร์กิตอีมูเลเตอร์ได้ทำงานตามคำสั่ง LD REG2,#1234H (โหมดการกำหนดเลขที่อยู่โดยทันทีด้วยคำสั่ง LD) เสร็จเรียบร้อยแล้ว และหยุดการทำงานที่ตำแหน่ง 2084H (ตามที่กำหนดไว้ในขั้นตอนที่ 4) ผลลัพธ์ที่ได้จากการทำงานตามคำสั่งนี้คือ ค่าของตัวนับโปรแกรม (PC) ในหน้าต่าง Register มีค่าเป็น 2084H ซึ่งเป็นแอดเดรสของคำสั่งที่สอง (LD REG1,REG2) ข้อมูลในหน้าต่าง Data Memory ตำแหน่งที่ 0070H และ 0071H ซึ่งถูกกำหนดให้เป็นที่อยู่ของเวิร์ดรีจิสเตอร์ REG2 จะมีค่าเท่ากับ 34H และ 12H ตามลำดับ โดยซิพจะทำการเก็บข้อมูลตัวเลข 34H เป็นไบต์ต่ำ ส่วนข้อมูลตัวเลข 12H ถูกเก็บเป็นไบต์สูง



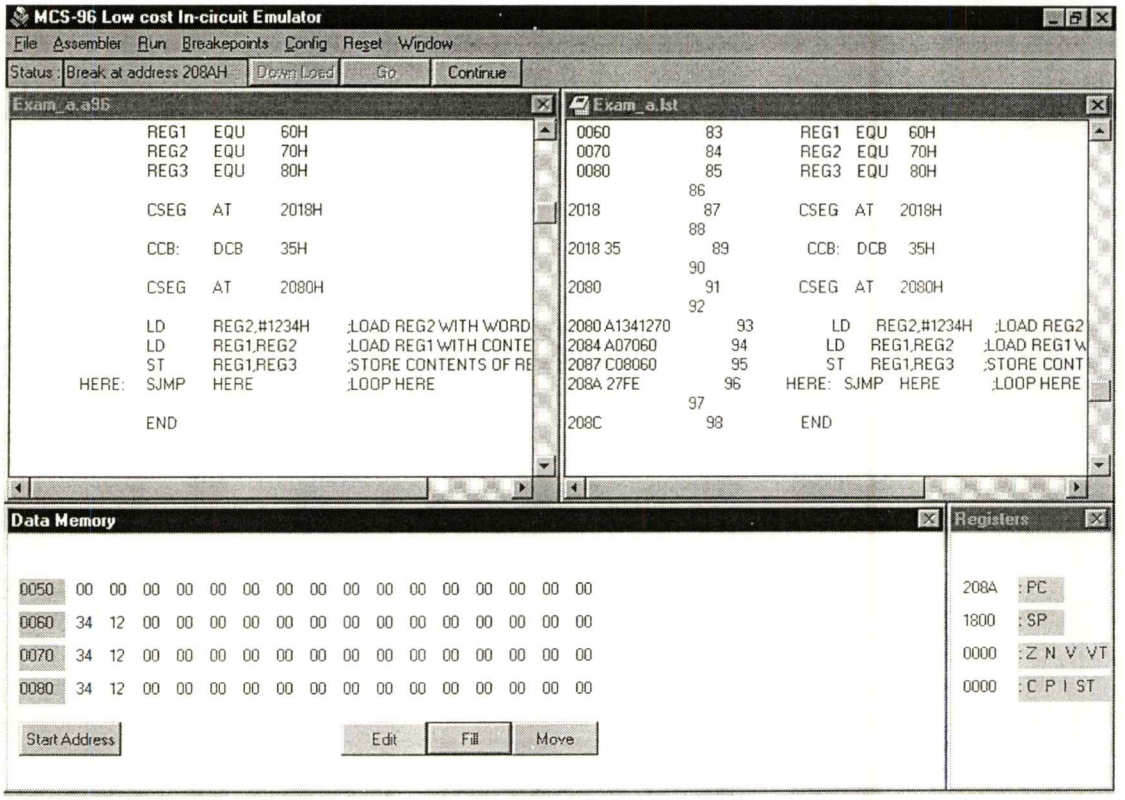
ภาพที่ 5.9 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง LD REG2, #1234H)

ขั้นตอนที่ 9 ทำการคลิกเมาส์ที่ปุ่ม Continue อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานต่อให้สังเกตุที่ Status : จะเปลี่ยนเป็น Run และ Break at address 2087H ตามลำดับ นั่นหมายถึงอินเซอร์กิตอีมูเลเตอร์ได้ทำงานตามคำสั่ง LD REG1,REG2 (โหมดการกำหนดเลขที่อยู่โดยตรง ด้วยคำสั่ง LD) เสร็จเรียบร้อยแล้ว และหยุดการทำงานที่ตำแหน่ง 2087H (ตามที่กำหนดไว้ในขั้นตอนที่ 4) ผลลัพธ์ที่ได้จากการทำงานตามคำสั่งนี้คือ ค่าของตัวนับโปรแกรม (PC) ในหน้าต่าง Register มีค่าเป็น 2087H ซึ่งเป็นแอดเดรสของคำสั่งที่สาม (ST REG1,REG3) ข้อมูลในหน้าต่าง Data Memory ตำแหน่งที่ 0060H และ 0061H ซึ่งถูกกำหนดให้เป็นที่อยู่ของเวิร์ดรีจิสเตอร์ REG1 จะมีค่าเท่ากับ 34H และ 12H ตามลำดับ โดยซิพจะทำการเก็บข้อมูลตัวเลข 34H เป็นไบต์ต่ำ ส่วนข้อมูลตัวเลข 12H ถูกเก็บเป็นไบต์สูง



ภาพที่ 5.10 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง LD REG1, REG2)

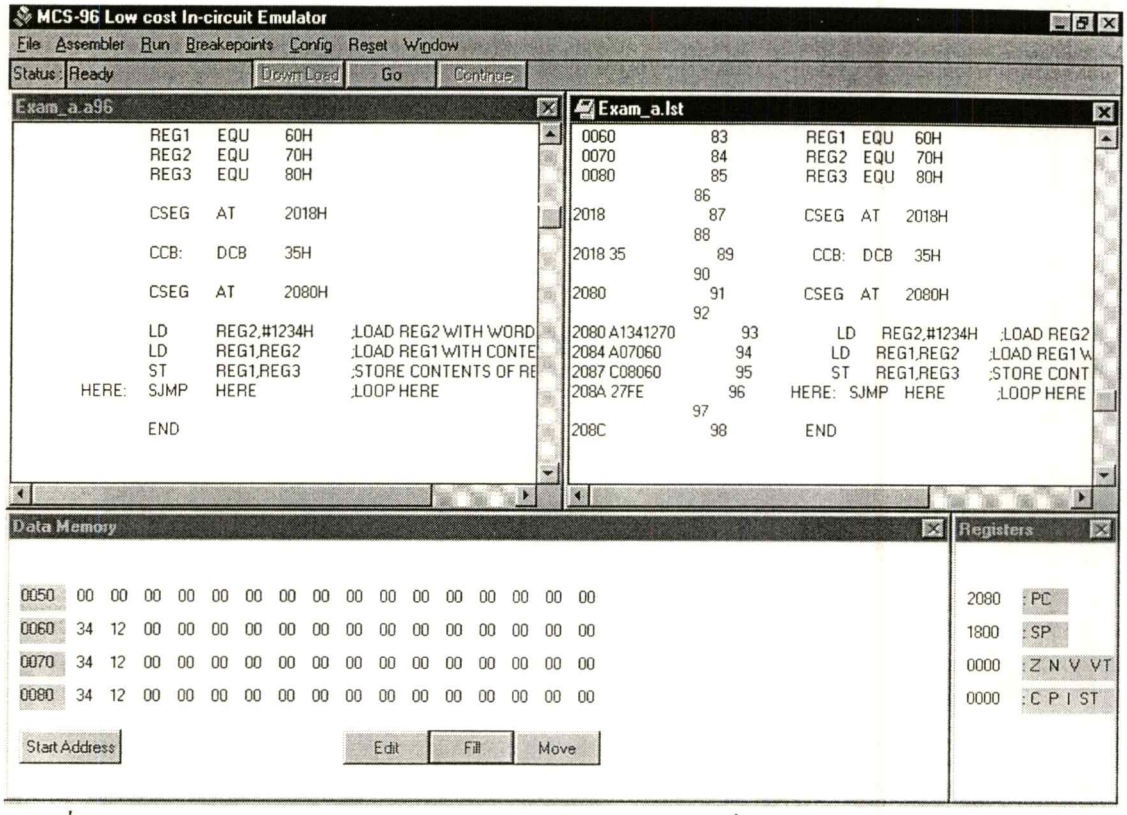
ขั้นตอนที่ 10 ทำการคลิกเมาส์ที่ปุ่ม Continue อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานต่อให้สังเกตุที่ Status : จะเปลี่ยนเป็น Run และ Break at address 208AH ตามลำดับ นั่นหมายถึงอินเซอร์กิตอีมูเลเตอร์ได้ทำงานตามคำสั่ง ST REG1,REG3 (โหมดการกำหนดเลขที่อยู่โดยตรง ด้วยคำสั่ง ST) เสร็จเรียบร้อยแล้ว และหยุดการทำงานที่ตำแหน่ง 208AH (ตามที่กำหนดไว้ในขั้นตอนที่ 4) ผลลัพธ์ที่ได้จากการทำงานตามคำสั่งนี้คือ ค่าของตัวนับโปรแกรม (PC) ในหน้าต่าง Register มีค่าเป็น 208AH ซึ่งเป็นแอดเดรสของคำสั่งที่สี่ SJMP HERE ข้อมูลในหน้าต่าง Data Memory ตำแหน่งที่ 0080H และ 0081H ซึ่งถูกกำหนดให้เป็นที่อยู่ของเวิร์ดรีจิสเตอร์ REG3 จะมีค่าเท่ากับ 34H และ 12H ตามลำดับ โดยซิพจะทำการเก็บข้อมูลตัวเลข 34H เป็นไบต์ต่ำ ส่วนข้อมูลตัวเลข 12H ถูกเก็บเป็นไบต์สูง



ภาพที่ 5.11 แสดงผลลัพธ์เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง ST REG1, REG3)

ขั้นตอนที่ 11 ทำการคลิกเมาส์ที่ปุ่ม Continue อินเซอร์กิตอีมูเลเตอร์จะเริ่มทำงานต่อให้สังเกตุที่ Status : จะเปลี่ยนเป็น Run นั่นหมายถึงอินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง SJMP HERE อยู่และจะปรากฏหน้าต่าง Go Forever ขึ้นมาเนื่องจากแอดเดรสที่ได้กำหนดไว้ภายในเบรคพอยน์บัฟเฟอร์ถูกใช้งานหมดแล้ว ในตอนนี้ผู้ใช้สามารถหยุดการทำงานของอินเซอร์กิตอีมูเลเตอร์ได้โดยปฏิบัติตามคำแนะนำในหน้าต่าง Go Forever เมื่อปฏิบัติดังนั้นแล้วที่ Status : จะแสดงข้อความ Ready ตัวนับโปรแกรม (PC) ในหน้าต่าง Register จะมีค่าเท่ากับ 2080H แสดงสภาวะเตรียมพร้อมของอินเซอร์กิตอีมูเลเตอร์ในการรอรับคำสั่ง

ภาพที่ 5.12 แสดงหน้าต่าง Go Forever เมื่อผู้ใช้คลิกเมาส์ที่ปุ่ม Continue (อินเซอร์กิตอีมูเลเตอร์ทำงานตามคำสั่ง SJMP HERE) ผู้ใช้สามารถหยุดการทำงานของอินเซอร์กิตอีมูเลเตอร์ได้โดยปฏิบัติตามคำแนะนำในหน้าต่างนี้



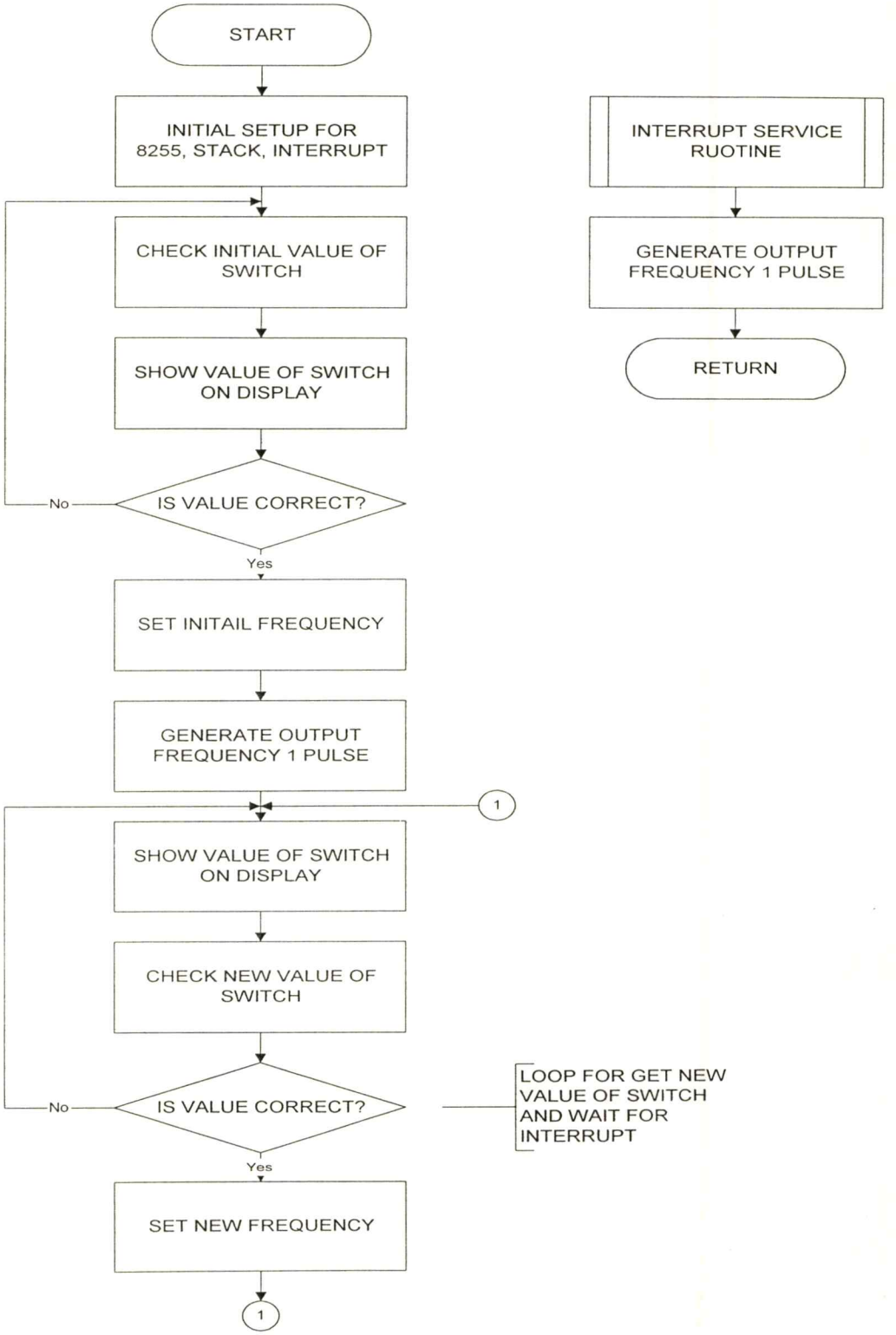
ภาพที่ 5.13 แสดงสภาวะเตรียมพร้อมของอินเซอร์กิตอีมูเลเตอร์ นั่นคือ Status : จะแสดงข้อความ Ready และตัวนับโปรแกรม (PC) ในหน้าต่าง Register จะมีค่าเท่ากับ 2080H (ผลลัพธ์ของโปรแกรมใน ตัวอย่างที่ 1 ปรากฏที่หน้าต่าง Data Memory ตำแหน่งที่ 0060H, 0070H และ 0080H)

5.2 ตัวอย่างที่ 2. การพัฒนาโปรแกรมประยุกต์ใช้งานโดยอินเซอร์กิตอีมูเลเตอร์ ในตัวอย่างนี้จะทำการเขียนโปรแกรมเพื่อควบคุมให้บอร์ดเป้าหมายทำหน้าที่เป็นเครื่องกำเนิดสัญญาณนาฬิกา (Clock generater) ที่สามารถผลิตความถี่ออกทางเอาท์พุตพอร์ตความเร็วสูง (HSO.0) ได้ 6 ความถี่ คือ 30 Hz, 60 Hz, 120 Hz, 240 Hz, 480 Hz และ 960 Hz โดยสามารถเลือกความถี่เอาท์พุตที่ต้องการได้จากการปรับสวิทช์ที่ต่ออยู่กับอินพุตพอร์ตซี (C) ของไอซี 8255 บนบอร์ดเป้าหมาย และ LED ซึ่งต่อเข้ากับเอาท์พุตพอร์ตเอ (A) ของไอซี 8255 นี้จะติดสว่างในตำแหน่งที่สัมพันธ์กับสวิทช์ที่ต่ออยู่กับอินพุตพอร์ตซี (C)

ตารางที่ 5.1 แสดงผลลัพธ์ในการทำงานของบอร์ดเป้าหมายที่ต้องการตามตัวอย่างที่ 2

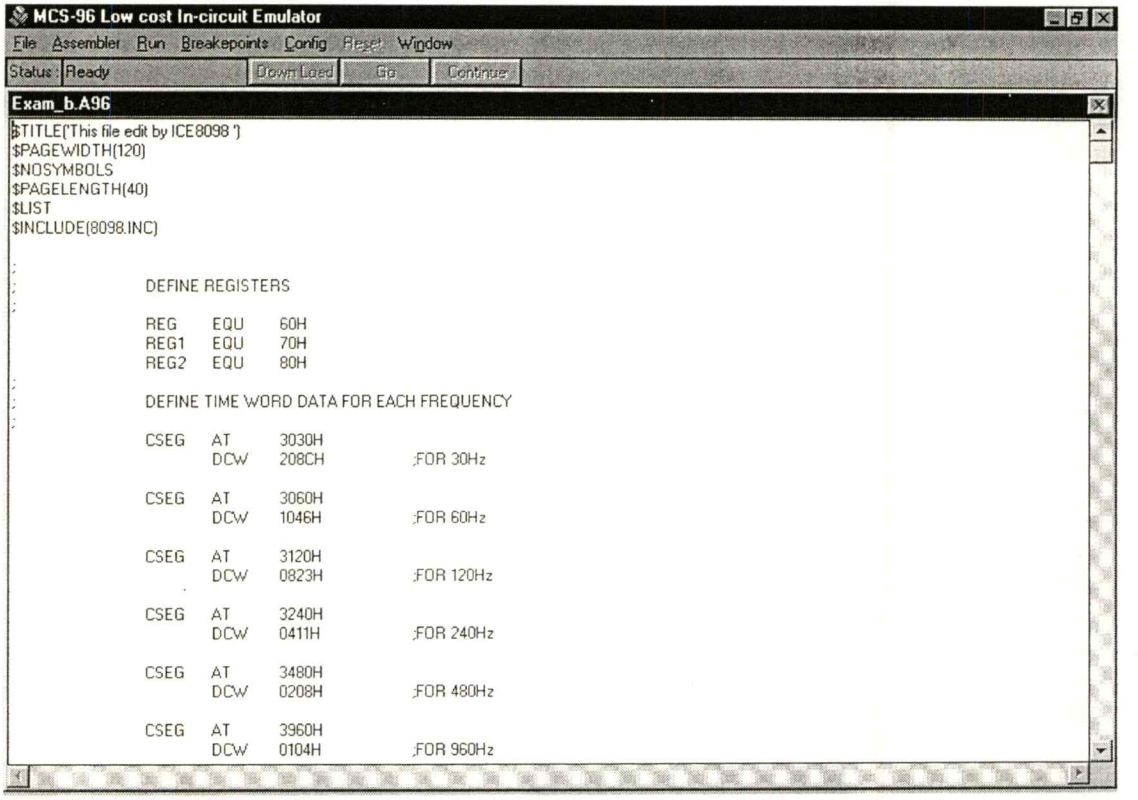
สถานะของสวิทช์ที่อินพุตพอร์ตซี (C) ของ 8255								สัญญาณนาฬิกาที่พอร์ตเอาท์พุตความเร็วสูง (HSO.0) ของชิพ 8098
#8	*#7*	*#6*	*#5*	*#4*	*#3*	*#2*	*#1*	
0	0	0	0	0	0	0	1	30 Hz
0	0	0	0	0	0	1	0	60 Hz
0	0	0	0	0	1	0	0	120 Hz
0	0	0	0	1	0	0	0	240 Hz
0	0	0	1	0	0	0	0	480 Hz
0	0	1	0	0	0	0	0	960 Hz

จากความต้องการที่ได้กล่าวไปแล้ว เราสามารถนำมาเขียนเป็นโฟลว์ชาร์ตแสดงการทำงานของโปรแกรมได้ดังนี้

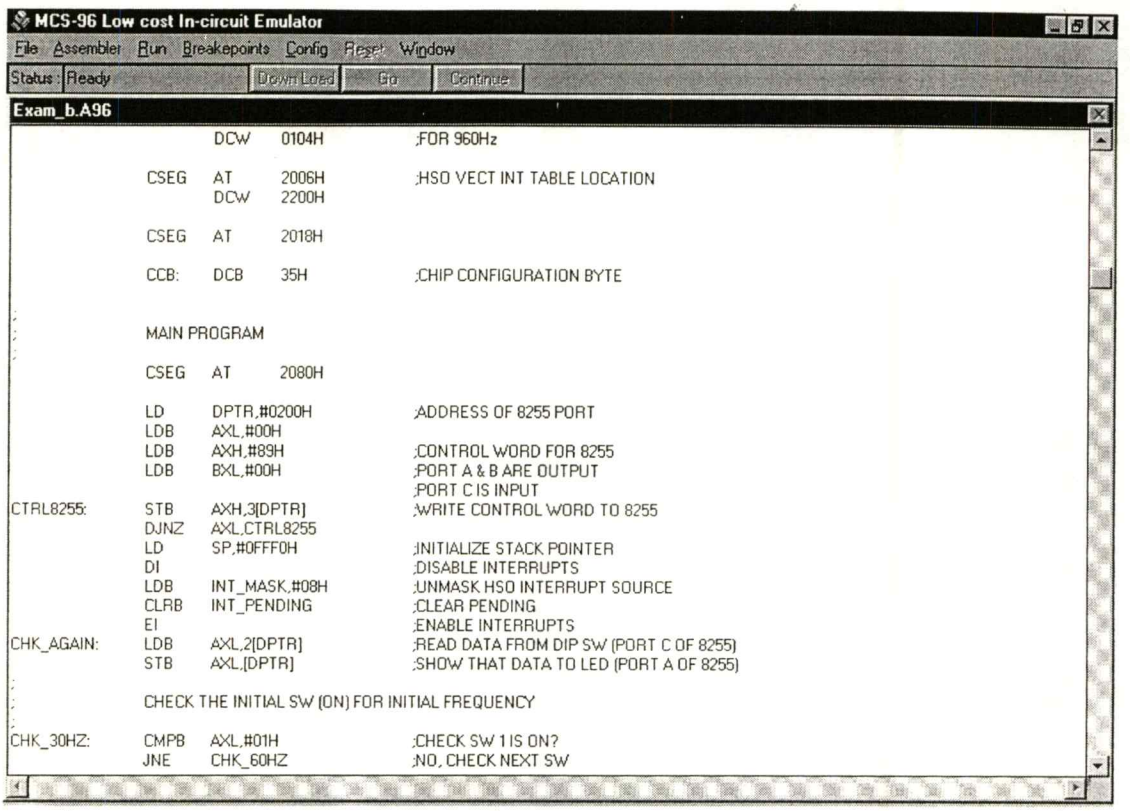


ภาพที่ 5.14 แสดงโฟลว์ชาร์ตการทำงานของโปรแกรมในตัวอย่างที่ 2

จากการทดลองใช้คำสั่งต่างๆ ของอินเซอร์กิตอีมูเลเตอร์ในการพัฒนาโปรแกรมตามโฟลว์ชาร์ตแสดงการทำงานด้านบน โดยเริ่มจากขั้นตอนการเขียนโปรแกรมไปจนถึงขั้นตอนที่บอร์ดเป้าหมายสามารถทำงานได้ตามที่ต้องการ พบว่าการพัฒนาโปรแกรมโดยอินเซอร์กิตอีมูเลเตอร์นี้ให้ความสะดวก และความคล่องตัวแก่ผู้ใช้งานเป็นอย่างมาก เมื่อเปรียบเทียบกับการพัฒนาโปรแกรมโดยใช้อีพีรอมอีมูเลเตอร์หรือวิธีการอื่นๆ ที่มีอยู่ในปัจจุบัน และเมื่อนำเอาโปรแกรมที่พัฒนาโดย อินเซอร์กิตอีมูเลเตอร์เสร็จสิ้นแล้ว ไปทำการบรรจุลงในอีพีรอมเพื่อประกอบลงบนบอร์ดเป้าหมายเป็นโปรแกรมมอนิเตอร์ พบว่าบอร์ดเป้าหมายสามารถทำงานได้ตรงตามต้องการที่กำหนดไว้ทุกประการ โปรแกรมยูสเซอร์ที่พัฒนาเสร็จสมบูรณ์แล้วแสดงได้ดังภาพที่ 5.15 ถึง 5.20

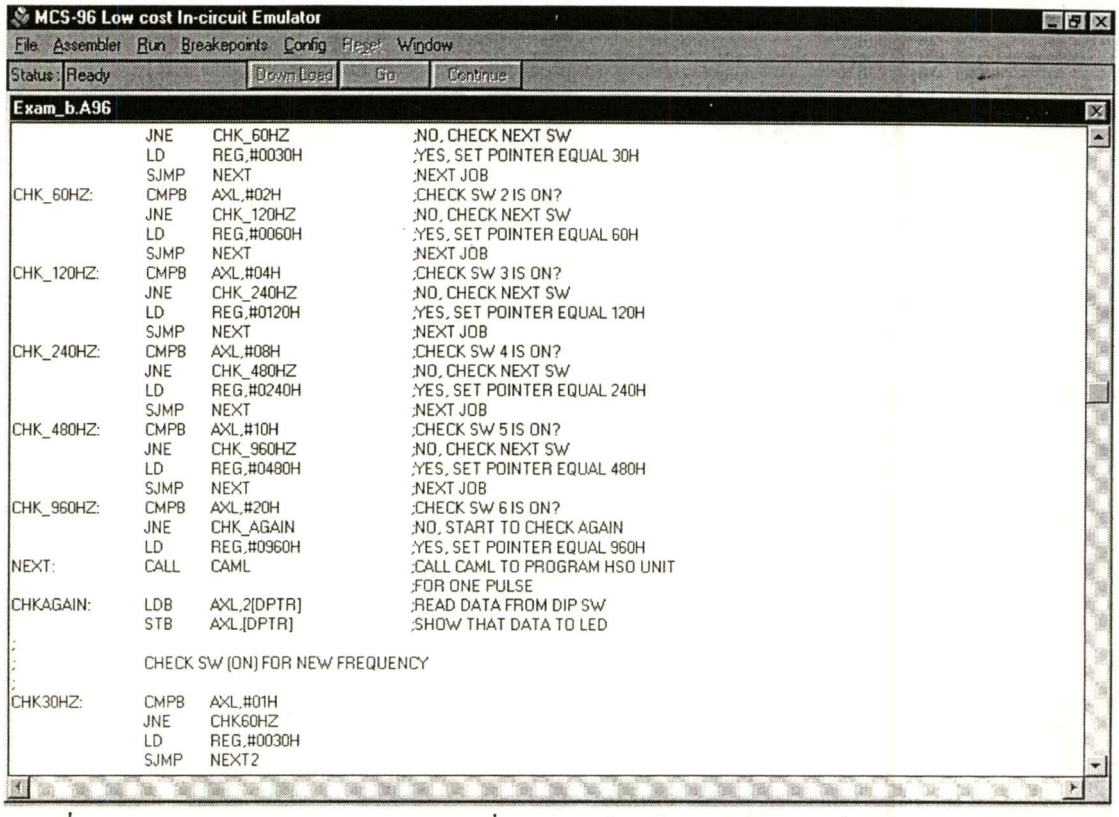


ภาพที่ 5.15 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์

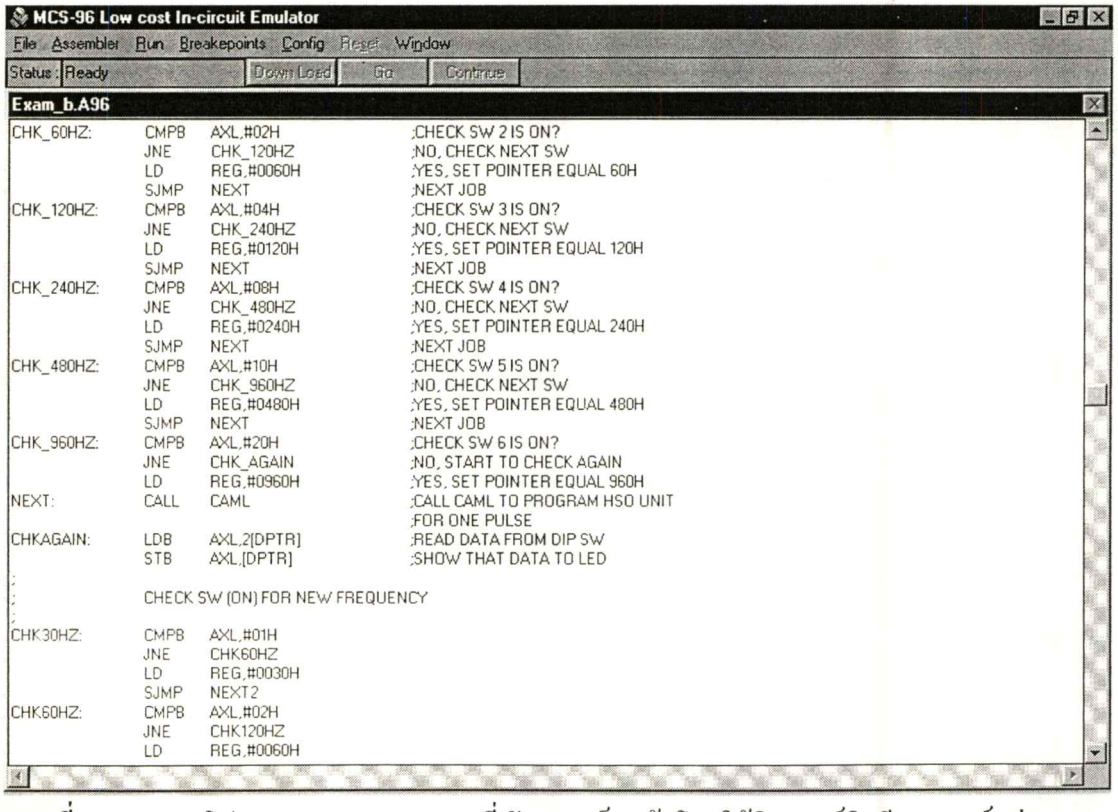



```
                    DCW     0104H           ;FOR 960Hz

                CSEG    AT      2006H       ;HSO VECT INT TABLE LOCATION
                    DCW     2200H

                CSEG    AT      2018H

                CCB:    DCB     35H         ;CHIP CONFIGURATION BYTE

;               MAIN PROGRAM
;
                CSEG    AT      2080H

                LD      DPTR,#0200H         ;ADDRESS OF 8255 PORT
                LDB     AXL,#00H
                LDB     AXH,#89H            ;CONTROL WORD FOR 8255
                LDB     BXL,#00H            ;PORT A & B ARE OUTPUT
                                            ;PORT C IS INPUT
CTRL8255:       STB     AXH,3[DPTR]         ;WRITE CONTROL WORD TO 8255
                DJNZ    AXL,CTRL8255
                LD      SP,#0FFF0H          ;INITIALIZE STACK POINTER
                DI                          ;DISABLE INTERRUPTS
                LDB     INT_MASK,#08H       ;UNMASK HSO INTERRUPT SOURCE
                CLRB    INT_PENDING         ;CLEAR PENDING
                EI                          ;ENABLE INTERRUPTS
CHK_AGAIN:      LDB     AXL,2[DPTR]         ;READ DATA FROM DIP SW (PORT C OF 8255)
                STB     AXL,[DPTR]          ;SHOW THAT DATA TO LED (PORT A OF 8255)
;
;               CHECK THE INITIAL SW (ON) FOR INITIAL FREQUENCY
;
CHK_30HZ:       CMPB    AXL,#01H            ;CHECK SW 1 IS ON?
                JNE     CHK_60HZ            ;NO, CHECK NEXT SW
```

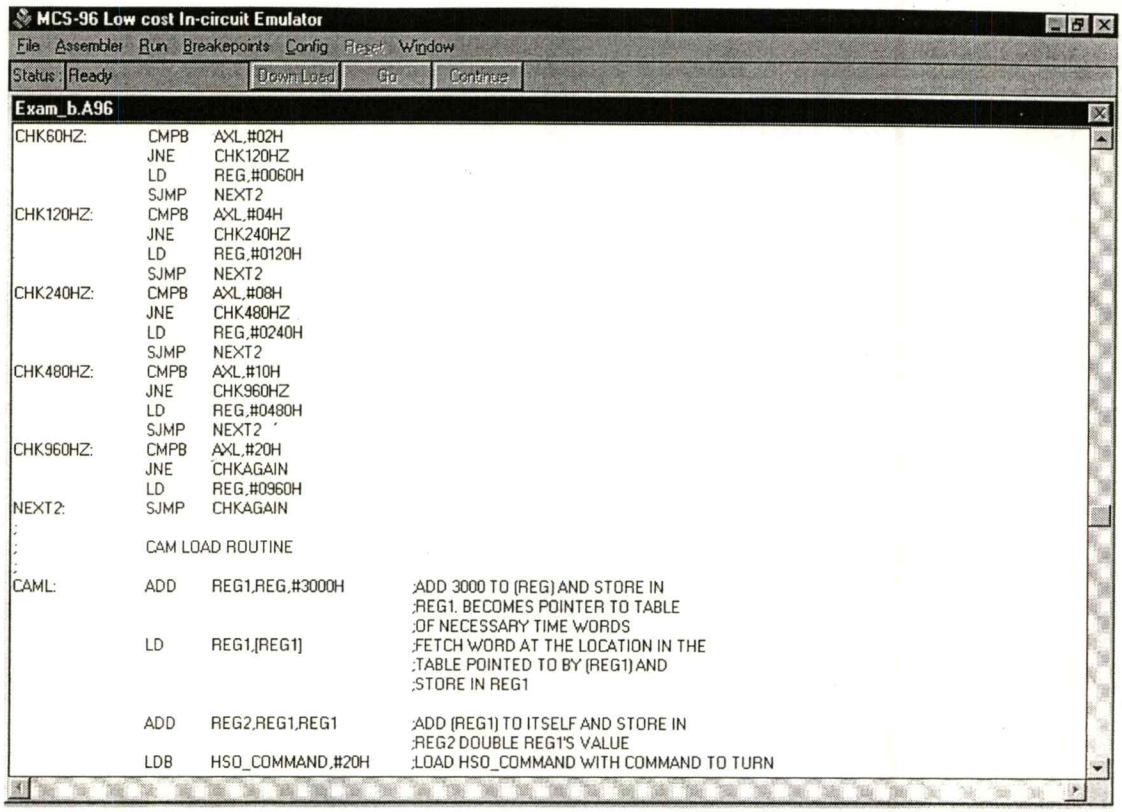
ภาพที่ 5.16 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)



ภาพที่ 5.17 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)

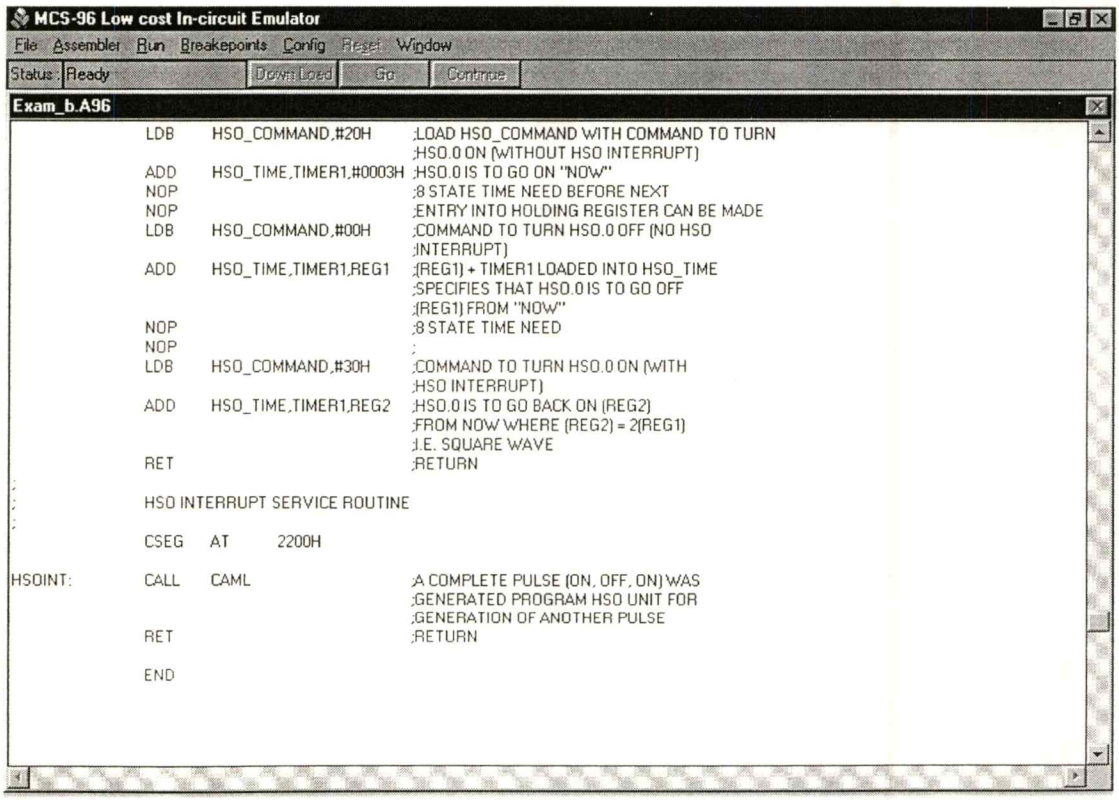


ภาพที่ 5.18 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)



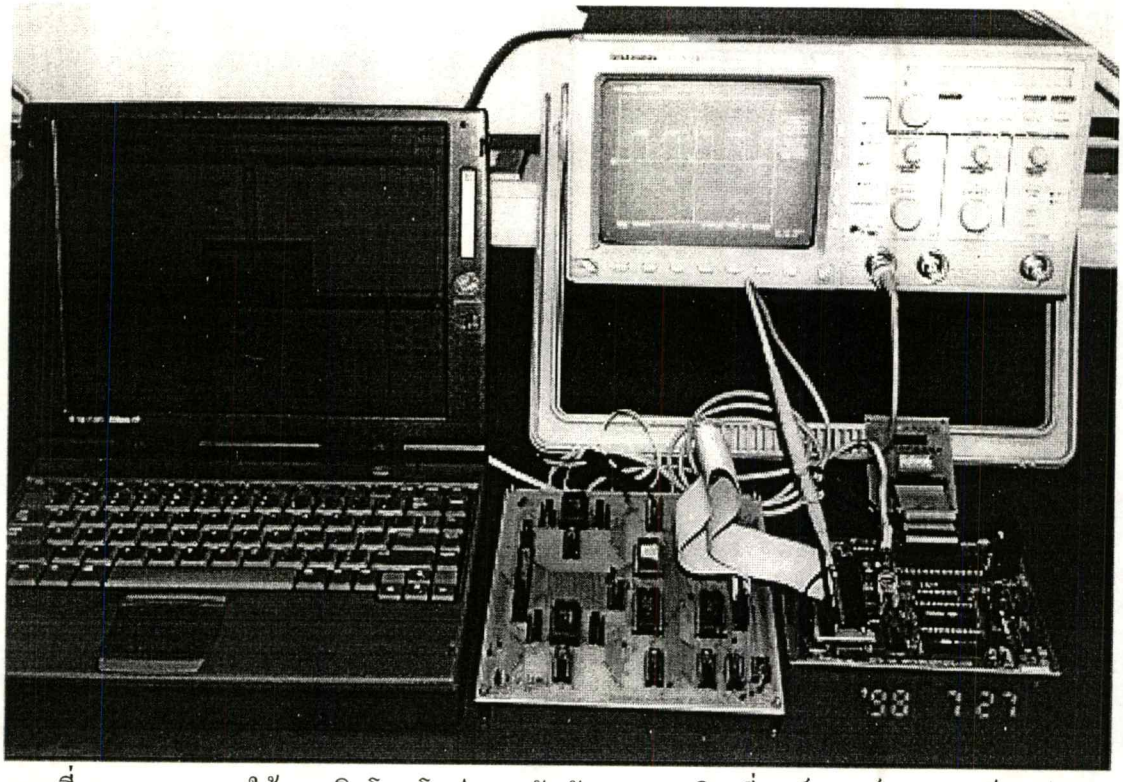

```
CHK60HZ:    CMPB    AXL,#02H
            JNE     CHK120HZ
            LD      REG,#0060H
            SJMP    NEXT2
CHK120HZ:   CMPB    AXL,#04H
            JNE     CHK240HZ
            LD      REG,#0120H
            SJMP    NEXT2
CHK240HZ:   CMPB    AXL,#08H
            JNE     CHK480HZ
            LD      REG,#0240H
            SJMP    NEXT2
CHK480HZ:   CMPB    AXL,#10H
            JNE     CHK960HZ
            LD      REG,#0480H
            SJMP    NEXT2
CHK960HZ:   CMPB    AXL,#20H
            JNE     CHKAGAIN
            LD      REG,#0960H
NEXT2:      SJMP    CHKAGAIN
;
;           CAM LOAD ROUTINE
;
CAML:       ADD     REG1,REG,#3000H     ;ADD 3000 TO (REG) AND STORE IN
                                        ;REG1. BECOMES POINTER TO TABLE
                                        ;OF NECESSARY TIME WORDS
            LD      REG1,[REG1]         ;FETCH WORD AT THE LOCATION IN THE
                                        ;TABLE POINTED TO BY (REG1) AND
                                        ;STORE IN REG1

            ADD     REG2,REG1,REG1      ;ADD (REG1) TO ITSELF AND STORE IN
                                        ;REG2 DOUBLE REG1'S VALUE
            LDB     HSO_COMMAND,#20H    ;LOAD HSO_COMMAND WITH COMMAND TO TURN
```

ภาพที่ 5.19 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)




```
            LDB     HSO_COMMAND,#20H    ;LOAD HSO_COMMAND WITH COMMAND TO TURN
                                        ;HSO.0 ON (WITHOUT HSO INTERRUPT)
            ADD     HSO_TIME,TIMER1,#0003H ;HSO.0 IS TO GO ON "NOW"
            NOP                         ;8 STATE TIME NEED BEFORE NEXT
            NOP                         ;ENTRY INTO HOLDING REGISTER CAN BE MADE
            LDB     HSO_COMMAND,#00H    ;COMMAND TO TURN HSO.0 OFF (NO HSO
                                        ;INTERRUPT)
            ADD     HSO_TIME,TIMER1,REG1 ;(REG1) + TIMER1 LOADED INTO HSO_TIME
                                        ;SPECIFIES THAT HSO.0 IS TO GO OFF
                                        ;(REG1) FROM "NOW"
            NOP                         ;8 STATE TIME NEED
            NOP                         ;
            LDB     HSO_COMMAND,#30H    ;COMMAND TO TURN HSO.0 ON (WITH
                                        ;HSO INTERRUPT)
            ADD     HSO_TIME,TIMER1,REG2 ;HSO.0 IS TO GO BACK ON (REG2)
                                        ;FROM NOW WHERE (REG2) = 2(REG1)
                                        ;I.E. SQUARE WAVE
            RET                         ;RETURN
;
;           HSO INTERRUPT SERVICE ROUTINE
;
            CSEG    AT      2200H

HSOINT:     CALL    CAML                ;A COMPLETE PULSE (ON, OFF, ON) WAS
                                        ;GENERATED PROGRAM HSO UNIT FOR
                                        ;GENERATION OF ANOTHER PULSE
            RET                         ;RETURN

            END
```

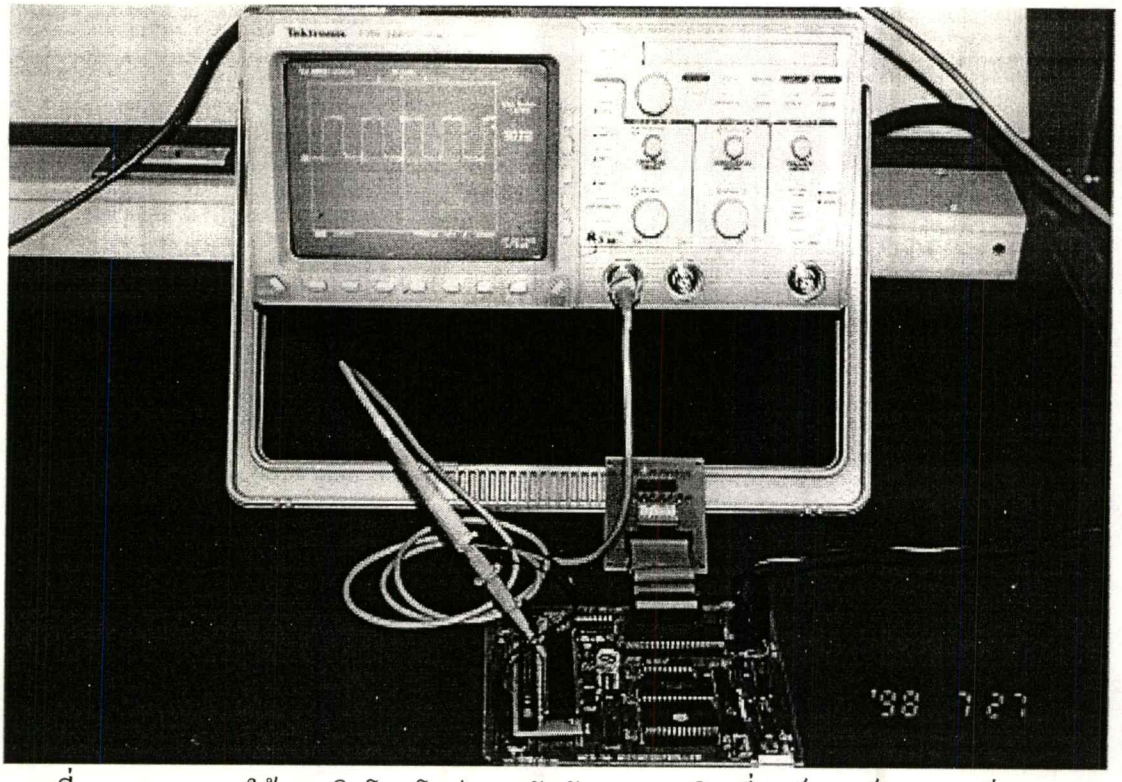
ภาพที่ 5.20 แสดงโปรแกรม Exam_b.A96 ที่พัฒนาเสร็จแล้วโดยใช้อินเซอร์กิตอีมูเลเตอร์ (ต่อ)



ภาพที่ 5.21 แสดงการใช้ออสซิลโลสโคปตรวจวัดสัญญาณนาฬิกาที่พอร์ตเอาท์พุตความเร็วสูง (HSO.0) เมื่อปรับสวิทช์หมายเลข 6 ที่ต่อกับอินพุตพอร์ตซี (C) ของไอซี 8255 ให้มีระดับลอจิกสูง ในขณะที่กำลังพัฒนาโปรแกรมโดยใช้อินเซอร์กิตอีมูเลเตอร์

ตารางที่ 5.2 แสดงผลลัพธ์ในการทำงานของบอร์ดเป้าหมายในขณะที่กำลังพัฒนาโปรแกรมโดยใช้อินเซอร์กิตอีมูเลเตอร์

สถานะของสวิทช์ที่อินพุตพอร์ตซี (C) ของ 8255								สัญญาณนาฬิกาที่พอร์ตเอาท์พุตความเร็วสูง (HSO.0) ของชิพ 8098
#8	*#7*	*#6*	*#5*	*#4*	*#3*	*#2*	*#1*	
0	0	0	0	0	0	0	1	30.11 Hz
0	0	0	0	0	0	1	0	60.31 Hz
0	0	0	0	0	1	0	0	119.1 Hz
0	0	0	0	1	0	0	0	240.4 Hz
0	0	0	1	0	0	0	0	479.5 Hz
0	0	1	0	0	0	0	0	960.3 Hz



ภาพที่ 5.22 แสดงการใช้ออสซิลโลสโคปตรวจวัดสัญญาณนาฬิกาที่พอร์ตเอาท์พุตความเร็วสูง (HSO.0) เมื่อปรับสวิทช์หมายเลข 6 ที่ต่อกับอินพุตพอร์ตซี (C) ของไอซี 8255 ให้มีระดับลอจิกสูง ในขณะที่บอร์ดเป้าหมายถูกควบคุมการทำงานด้วยโปรแกรมมอนิเตอร์ซึ่งบรรจุอยู่ภายในอีพีรอม

ตารางที่ 5.3 แสดงผลลัพธ์ในการทำงานของบอร์ดเป้าหมายที่ควบคุมการทำงานด้วยโปรแกรมมอนิเตอร์ ซึ่งบรรจุอยู่ภายในอีพีรอม

สถานะของสวิทช์ที่อินพุตพอร์ตซี (C) ของ 8255								สัญญาณนาฬิกาที่พอร์ตเอาท์พุตความเร็วสูง (HSO.0) ของชิพ 8098
#8	*#7*	*#6*	*#5*	*#4*	*#3*	*#2*	*#1*	
0	0	0	0	0	0	0	1	30.12 Hz
0	0	0	0	0	0	1	0	60.26 Hz
0	0	0	0	0	1	0	0	119.2 Hz
0	0	0	0	1	0	0	0	240.2 Hz
0	0	0	1	0	0	0	0	479.4 Hz
0	0	1	0	0	0	0	0	960.2 Hz

บทที่ 6

สรุปผลและข้อเสนอแนะ

วิทยานิพนธ์ฉบับนี้มีวัตถุประสงค์ในการศึกษาและวิจัยเพื่อสร้างอินเซอร์กิตอีมูเลเตอร์สำหรับ 8098 ไมโครคอนโทรลเลอร์ ซึ่งเป็นซีพียูขนาด 16 บิตในตระกูล MCS-96 โดยมีเป้าหมายให้ราคาต้นทุนในการผลิตต่ำ เมื่อเปรียบเทียบกับอินเซอร์กิตอีมูเลเตอร์ที่เรานำเข้ามาจากต่างประเทศ แต่สามารถใช้ในการพัฒนาโปรแกรมได้อย่างมีประสิทธิภาพ

หลักการและแนวความคิดที่นำมาประกอบการวิจัยของวิทยานิพนธ์นี้ คือการใช้งานอุปกรณ์ต่าง ๆ ร่วมกันระหว่างบอร์ดเป้าหมายและอินเซอร์กิตอีมูเลเตอร์ ด้วยเหตุนี้วงจรของอินเซอร์กิตอีมูเลเตอร์จึงมีเฉพาะส่วนที่จำเป็นต้องใช้งาน, มีขนาดเล็กและง่ายต่อการออกแบบ ส่งผลให้ใช้อุปกรณ์น้อยชิ้นและมีราคาถูก ข้อมูลในตารางที่ 6.1 แสดงการเปรียบเทียบราคาระหว่างอินเซอร์กิตอีมูเลเตอร์ที่ได้สร้างขึ้นและอินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศโดยผ่านบริษัทตัวแทนจำหน่ายภายในประเทศ

ตารางที่ 6.1 แสดงการเปรียบเทียบราคาของอินเซอร์กิตอีมูเลเตอร์

รายการ	อินเซอร์กิตอีมูเลเตอร์ที่วิจัยและสร้างขึ้น	อินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศ*	รายละเอียดเพิ่มเติม
ฮาร์ดแวร์	4,000	514,080	EMUL196-KC,KD
ซอฟแวร์	-	125,300	BSOTSK/C 196PKG
ราคารวม	4,000	639,380	

*หมายเหตุ ข้อมูลได้จากบริษัท COMPLEX TECHNOLOGY (Jan. 1997)

จากตัวเลขในตารางจะเห็นได้ว่าราคาของอินเซอร์กิตอีมูเลเตอร์ที่ได้ทำการวิจัยและสร้างขึ้นมานั้นมีราคาถูกกว่าอินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศประมาณ 100 ~ 150 เท่า และเมื่อพิจารณาให้ดีราคาของอินเซอร์กิตอีมูเลเตอร์ที่ได้ทำการวิจัยและสร้างขึ้นมานั้นเป็นราคาของเครื่องต้นแบบ ดังนั้นหากมีผู้สนใจที่จะนำไปพัฒนาและผลิตออกจำหน่าย ก็จะทำให้ราคาต้นทุนการผลิตต่ำกว่าตัวเลขที่ปรากฏในตารางที่ 6.1

อย่างไรก็ดีอินเซอร์กิตอีมูเลเตอร์ที่ได้ทำการวิจัยและสร้างขึ้นมานี้ ยังมีคุณสมบัติและฟังก์ชั่นการใช้งานหลายประการที่ด้อยกว่าอินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศ ดังจะเห็นได้จากข้อมูลในตารางที่ 6.2 ซึ่งแสดงการเปรียบเทียบคุณสมบัติและฟังก์ชั่นการใช้งานระหว่างอินเซอร์กิตอีมูเลเตอร์ทั้งสอง

ตารางที่ 6.2 แสดงการเปรียบเทียบคุณสมบัติและฟังก์ชั่นการใช้งานของอินเซอร์กิตอีมูเลเตอร์

รายการ	อินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศ	อินเซอร์กิตอีมูเลเตอร์ที่วิจัยและสร้างขึ้น
Supported 80C196 Derivative:	- 80C196KC, KD, KQ, KR, KT, JQ, JR, JT, NT and more	- 8098
Hosts :	- 386 or better computer - Sun, HP and other workstation with LanICE	- 486 or better computer - None
High level Debugging :	- Window for source level debugging - Single step or Line step with breakpoints mark directly in the code - Full support of local and global variables - Currently support for Intel OMF, Intel Extended OMF, BSO/Tasking and more	- Windows 95 for source level debugging - None - None - None

ตารางที่ 6.2 (ต่อ) แสดงการเปรียบเทียบคุณสมบัติและฟังก์ชั่นการใช้งานของอินเซอร์กิต อีมูเลเตอร์

รายการ	อินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศ	อินเซอร์กิตอีมูเลเตอร์ที่วิจัยและสร้างขึ้น
Symbolic Support :	- Full symbolic with type information - Same symbols can be used for different modules - All special functions registers supported	- None - None - None
Real-Time Emulation :	- Full speed emulation up to the chip limit - No wait states and no intrusion on Memory, I/O or interrupt pins	- Real time emulation - None
In-Line Assembler and Disassembler :	- Full instruction set and symbols supported	- None
Emulation Memory :	- 64K Data memory and 64K code Memory, using 16 bit address - 256K memory if using 24 bits address	- 58K Data memory and code memory, using 16 bit address - None
Memory Mapping :	- Mappable down to even byte	- Mappable down to 2K byte
Breakpoints :	- Hard ware and Soft ware breakpoints - Unlimited program breakpoints - Break on internal access on both Data and Address	- Soft ware breakpoints only - Maximum 16 breakpoints - None

ตารางที่ 6.2 (ต่อ) แสดงการเปรียบเทียบคุณสมบัติและฟังก์ชั่นการใช้งานของอินเซอร์กิตอีมูเลเตอร์

รายการ	อินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศ	อินเซอร์กิตอีมูเลเตอร์ที่วิจัยและสร้างขึ้น
	- With the trace board option, it 's possible to break on any Address, Data or External signal	- None
Single Stepping :	- Single or multiple instruction stepping - Step over calls and interrupts - Line stepping in high-level languages	- None - None - None
Real-Time Trace :	- Up to 512K deep, 104 bits wide, with time stamp - Three trigger levels, trigger on address and data - Instruction queue decoding prevents false triggers	- None - None - None

เมื่อพิจารณาข้อแตกต่างระหว่างอินเซอร์กิตอีมูเลเตอร์ทั้งสอง พบว่าคุณสมบัติและฟังก์ชั่นการใช้งานของอินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศนั้นมีข้อดีกว่าอินเซอร์กิตอีมูเลเตอร์ที่ได้ทำการวิจัยและสร้างขึ้นอยู่หลายประการ เช่น รองรับซีพียูในตระกูล MCS-96 ได้หลายเบอร์, สามารถทำงานแบบ Single Step ได้หลายรูปแบบ และความสามารถในการตรวจสอบการทำงานของซีพียูด้วย Trace Board เป็นต้น ด้วยเหตุนี้จึงทำให้อินเซอร์กิตอีมูเลเตอร์ที่นำเข้ามาจากต่างประเทศมีประสิทธิภาพสูง และสะดวกต่อการใช้งานเป็นอย่างมาก

อย่างไรก็ดีคุณสมบัติและฟังก์ชั่นการใช้งานของอินเซอร์กิตอีมูเลเตอร์ที่ได้ทำการวิจัยและสร้างขึ้นมานี้ มีความเหมาะสมและเพียงพอต่อการศึกษาและนำไปประยุกต์ใช้งานในระดับหนึ่ง ซึ่งในอนาคตจำเป็นต้องมีการพัฒนาให้มีขีดความสามารถ และประสิทธิภาพสูงขึ้นต่อไป โดยสามารถทำได้หลายแนวทางดังนี้

1. พัฒนาโปรแกรมมอนิเตอร์ให้สามารถรองรับคำสั่งได้มากขึ้นเช่นคำสั่ง Single Step, Disassembler, Up load และคำสั่งอื่น ๆ เพื่อเพิ่มประสิทธิภาพ และความคล่องตัวในการใช้งานของอินเซอร์กิตอีมูเลเตอร์

2. พัฒนาโปรแกรมที่ควบคุมการทำงานบนเครื่องไมโครคอมพิวเตอร์ให้สามารถเขียนหรือเปิดแฟ้มข้อมูลของโปรแกรมยูสเซอร์ได้ทีละหลาย ๆแฟ้มพร้อมกัน เพื่อความสะดวกในการพัฒนาโปรแกรม

3. พัฒนาฮาร์ดแวร์ให้มีวงจรที่สามารถทำการ Trace ได้เพื่อความสะดวกในการตรวจสอบการทำงานของซีพียู

4. พัฒนาฮาร์ดแวร์ให้มีวงจรการรับส่งข้อมูลแบบอนุกรม เพื่อให้สามารถใช้งานร่วมกับบอร์ดเป้าหมายที่ไม่มีวงจรรับส่งข้อมูลแบบอนุกรมได้

5. พัฒนาฮาร์ดแวร์ให้สามารถรองรับชิพเบอร์ 8096 ที่บัสข้อมูลสามารถควบคุมให้ทำงานในแบบ 8 บิตบัสหรือ 16 บิตบัสได้

เอกสารอ้างอิง

[1] กอบชัย เดชหาญ และคณะ. "ไมโครคอนโทรลเลอร์ในระบบควบคุมแบบแยกส่วนผ่านพอร์ทอนุกรมเพื่องานควบคุมในโรงงานขนาดเล็ก." การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 15. หน้า 4-17 - 4-23.

[2] สุนทร วิทูสุรพจน์. **การใช้งานไมโครคอนโทรลเลอร์ตระกูล 8051**. กรุงเทพฯ : เอช.เอ็น. กรุ๊ป. ม.ป.ป.

[3] อิศราพงศ์ สินันตา และคณะ. "แผงช่วยพัฒนาโปรแกรม สำหรับไมโครคอนโทรลเลอร์เบอร์ 8031 ราคาประหยัด ที่อาศัยเทคนิคการใช้หน่วยประมวลผลกลางร่วมกัน." การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 17. หน้า 368 - 373.

[4] Intel. **8X9X User's Manual**. 1993.

[5] Intel. **MCS-96 Architectural Overview and Quick References**. 1991.

[6] Intel. **Embedded Controller Handbook**. 1988.

[7] Katz Ron, Howard Boyet. **The 16-bit 8096 Programming, Interfacing, Applications**. New York : Microprocessor Training Inc. 1986.

[8] Peatman John B. **Design with Microcontrollers**. Singapore : McGraw-Hill. 1988.

[9] ชรินทร์ บัณฑุกุล. "8051 เรียลไทม์ อิน-เซอร์กิต อีมูเลเตอร์." วิทยานิพนธ์วิศวกรรมศาสตร์มหาบัณฑิต สาขาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2534.

[10] ชำนาญ เฉลิมยุทร และ ดำรงศักดิ์ ไชยกิตติโสภณ. "อีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51." ปริญญานิพนธ์วิศวกรรมศาสตร์บัณฑิต ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2537.

[11] National Semiconductor. **Programmable Logic Devices Databook and Design Guide**. 1990.

[12] Uffenbeck John. **Microcomputers and Microprocessors**. 2nd Ed. n.p., n.d.

[13] Intel. **MCS-96 Macro Assembler User's Guide for DOS System**. 1990.

[14] Intel. **EV8097BH Microcontroller Evaluation Board USER'S MANUAL**. 1989.

[15] Microsoft. **GW-BASIC Interpreter User's Guide**. 1987.

[16] Maxwell Taylor, Bryon Scott. **Visual BASIC Superbible**. Waite Group Press. 1992.

[17] Mansfield Richard. The Visual Guide to Visual BASIC for Windows. New York : Ventana Press Inc. 1993.

[18] Scott D.J. Visaul BASIC for MS-DOS by Example. Que Corporation. 1992.

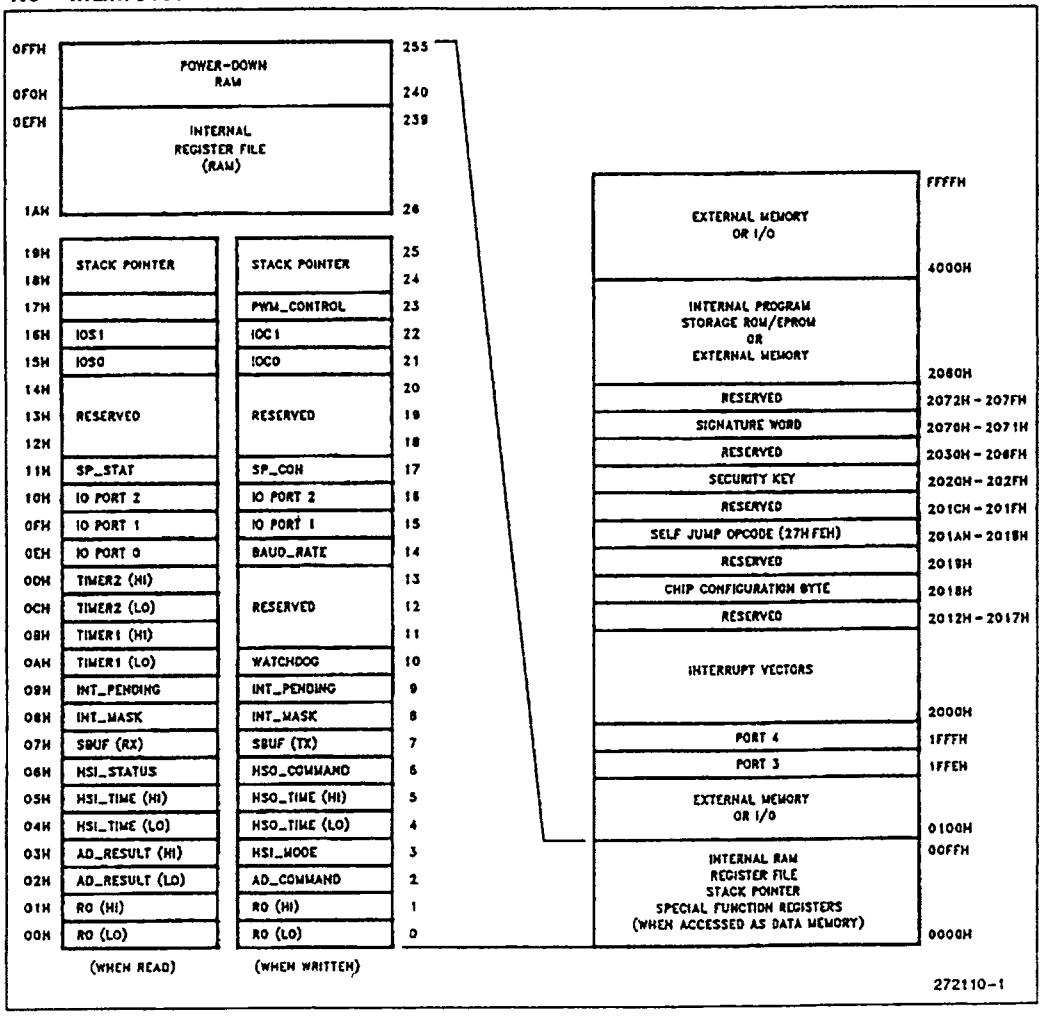
[19] จรณิต แก้วกังวาล. 20 ชั่วโมงกับ Quick BASIC. กรุงเทพฯ : เอช.เอ็น. กรุ๊ป. 2521.

ภาคผนวก

ภาคผนวก ก

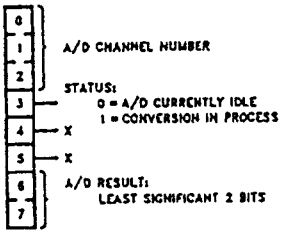
8X9X Quick Reference

1.0 MEMORY AND SFR MAP



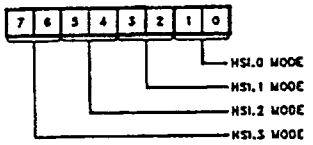
2.0 SFR BIT SUMMARY

A/D Result LO (02H)



272110-2

HSI_Mode (03H)

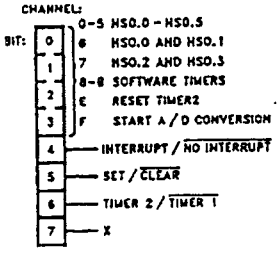


WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

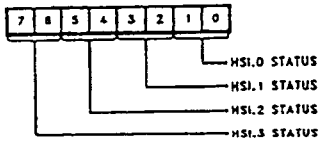
272110-3

HSO Command (06H)



272110-4

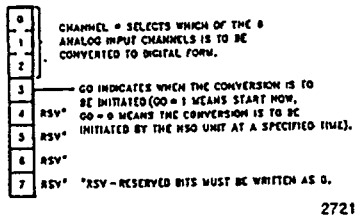
HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURRED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

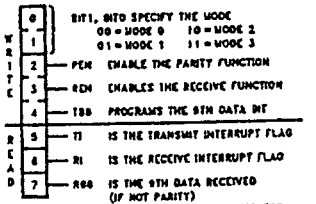
272110-5

A/D Command (02H)



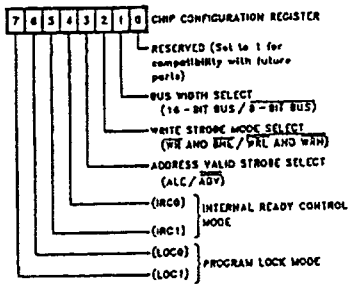
272110-6

SPCON/SPSTAT (11H)



272110-7

Chip Configuration



272110-8

Internal Ready Control

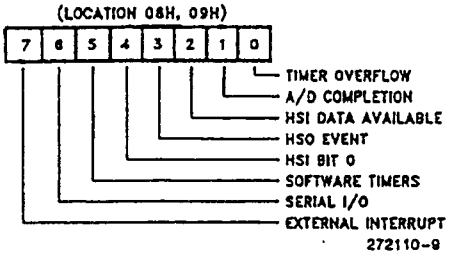
IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

Program Lock Modes

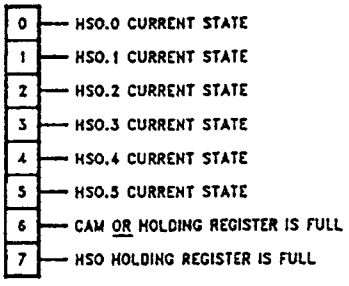
LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Interrupt Pending/Mask Register

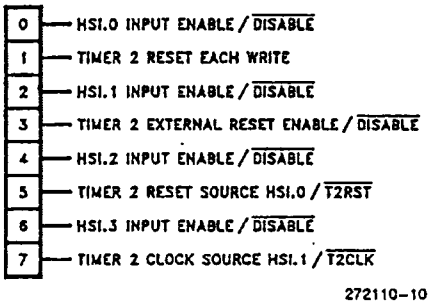
(LOCATION 08H, 09H)



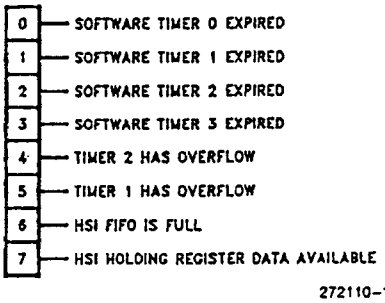
IOS0 (15H)



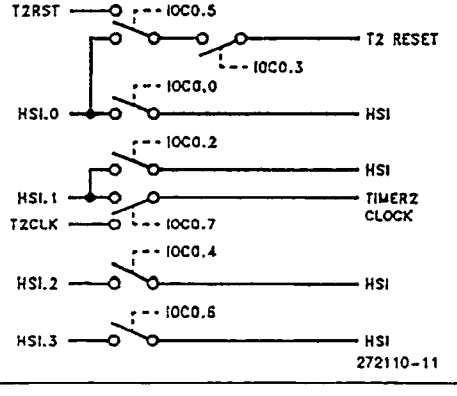
IOC0 (15H)



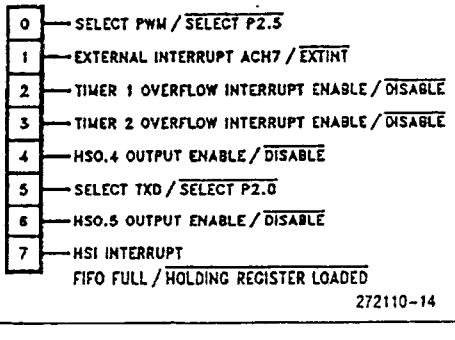
IOS1 (16H)



IOC0 (15H)



IOC1 (16H)



PSW Register

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

6.0 OPCODE TABLE

Opcode	Instruction
00	SKIP
01	CLR
02	NOT
03	NEG
04	XCH
05	DEC
06	EXT
07	INC
08	SHR
09	SHL
0A	SHRA
0B	RESERVED**
0C	SHRL
0D	SHLL
0E	SHRAL
0F	NORML
10	RESERVED**
11	CLRB
12	NOTB
13	NEGB
14	XCHB
15	DECB
16	EXTB
17	INCB
18	SHRB
19	SHLB
1A	SHRAB
1B	RESERVED**
1C	RESERVED**
1D	RESERVED**
1E	RESERVED**
1F	RESERVED**
20	SJMP
21	SJMP
22	SJMP
23	SJMP
24	SJMP
25	SJMP
26	SJMP
27	SJMP
28	SCALL
29	SCALL
2A	SCALL
2B	SCALL
2C	SCALL
2D	SCALL
2E	SCALL
2F	SCALL
30	JBC
31	JBC
32	JBC
33	JBC

Opcode	Instruction
34	JBC
35	JBC
36	JBC
37	JBC
38	JBS
39	JBS
3A	JBS
3B	JBS
3C	JBS
3D	JBS
3E	JBS
3F	JBS
40	AND DIRECT (3 OPS)
41	AND IMMEDIATE (3 OPS)
42	AND INDIRECT (3 OPS)
43	AND INDEXED (3 OPS)
44	ADD DIRECT (3 OPS)
45	ADD IMMEDIATE (3 OPS)
46	ADD INDIRECT (3 OPS)
47	ADD INDEXED (3 OPS)
48	SUB DIRECT (3 OPS)
49	SUB IMMEDIATE (3 OPS)
4A	SUB INDIRECT (3 OPS)
4B	SUB INDEXED (3 OPS)
4C	MULU DIRECT (3 OPS)
4D	MULU IMMEDIATE (3 OPS)
4E	MULU INDIRECT (3 OPS)
4F	MULU INDEXED (3 OPS)
50	ANDB DIRECT (3 OPS)
51	ANDB IMMEDIATE (3 OPS)
52	ANDB INDIRECT (3 OPS)
53	ANDB INDEXED (3 OPS)
54	ADDB DIRECT (3 OPS)
55	ADDB IMMEDIATE (3 OPS)
56	ADDB INDIRECT (3 OPS)
57	ADDB INDEXED (3 OPS)
58	SUBB DIRECT (3 OPS)
59	SUBB IMMEDIATE (3 OPS)
5A	SUBB INDIRECT (3 OPS)
5B	SUBB INDEXED (3 OPS)
5C	MULUB DIRECT (3 OPS)
5D	MULUB IMMEDIATE (3 OPS)
5E	MULUB INDIRECT (3 OPS)
5F	MULUB INDEXED (3 OPS)
60	AND DIRECT (2 OPS)
61	AND IMMEDIATE (2 OPS)
62	AND INDIRECT (2 OPS)
63	AND INDEXED (2 OPS)
64	ADD DIRECT (2 OPS)
65	ADD IMMEDIATE (2 OPS)
66	ADD INDIRECT (2 OPS)
67	ADD INDEXED (2 OPS)

6.0 OPCODE TABLE (Continued)

Opcode	Instruction
68	SUB DIRECT (2 OPS)
69	SUB IMMEDIATE (2 OPS)
6A	SUB INDIRECT (2 OPS)
6B	SUB INDEXED (2 OPS)
6C	MULU DIRECT (2 OPS)
6D	MULU IMMEDIATE (2 OPS)
6E	MULU INDIRECT (2 OPS)
6F	MULU INDEXED (2 OPS)
70	ANDB DIRECT (2 OPS)
71	ANDB IMMEDIATE (2 OPS)
72	ANDB INDIRECT (2 OPS)
73	ANDB INDEXED (2 OPS)
74	ADDB DIRECT (2 OPS)
75	ADDB IMMEDIATE (2 OPS)
76	ADDB INDIRECT (2 OPS)
77	ADDB INDEXED (2 OPS)
78	SUBB DIRECT (2 OPS)
79	SUBB IMMEDIATE (2 OPS)
7A	SUBB INDIRECT (2 OPS)
7B	SUBB INDEXED (2 OPS)
7C	MULUB DIRECT (2 OPS)
7D	MULUB IMMEDIATE (2 OPS)
7E	MULUB INDIRECT (2 OPS)
7F	MULUB INDEXED (2 OPS)
80	OR DIRECT
81	OR IMMEDIATE
82	OR INDIRECT
83	OR INDEXED
84	XOR DIRECT
85	XOR IMMEDIATE
86	XOR INDIRECT
87	XOR INDEXED
88	CMP DIRECT
89	CMP IMMEDIATE
8A	CMP INDIRECT
8B	CMP INDEXED
8C	DIVU DIRECT
8D	DIVU IMMEDIATE
8E	DIVU INDIRECT
8F	DIVU INDEXED
90	ORB DIRECT
91	ORB IMMEDIATE
92	ORB INDIRECT
93	ORB INDEXED
94	XORB DIRECT
95	XORB IMMEDIATE
96	XORB INDIRECT
97	XORB INDEXED
98	CMPB DIRECT
99	CMPB IMMEDIATE
9A	CMPB INDIRECT
9B	CMPB INDEXED

Opcode	Instruction
9C	DIVUB DIRECT
9D	DIVUB IMMEDIATE
9E	DIVUB INDIRECT
9F	DIVUB INDEXED
A0	LD DIRECT
A1	LD IMMEDIATE
A2	LD INDIRECT
A3	LD INDEXED
A4	ADDC DIRECT
A5	ADDC IMMEDIATE
A6	ADDC INDIRECT
A7	ADDC INDEXED
A8	SUBC DIRECT
A9	SUBC IMMEDIATE
AA	SUBC INDIRECT
AB	SUBC INDEXED
AC	LDBZE DIRECT
AD	LDBZE IMMEDIATE
AE	LDBZE INDIRECT
AF	LDBZE INDEXED
B0	LDB DIRECT
B1	LDB IMMEDIATE
B2	LDB INDIRECT
B3	LDB INDEXED
B4	ADDCB DIRECT
B5	ADDCB IMMEDIATE
B6	ADDCB INDIRECT
B7	ADDCB INDEXED
B8	SUBCB DIRECT
B9	SUBCB IMMEDIATE
BA	SUBCB INDIRECT
BB	SUBCB INDEXED
BC	LDBSE DIRECT
BD	LDBSE IMMEDIATE
BE	LDBSE INDIRECT
BF	LDBSE INDEXED
C0	ST DIRECT
C1	RESERVED**
C2	ST INDIRECT
C3	ST INDEXED
C4	STB DIRECT
C5	RESERVED**
C6	STB INDIRECT
C7	STB INDEXED
C8	PUSH DIRECT
C9	PUSH IMMEDIATE
CA	PUSH INDIRECT
CB	PUSH INDEXED
CC	POP DIRECT
CD	RESERVED**
CE	POP INDIRECT
CF	POP INDEXED

6.0 OPCODE TABLE (Continued)

Opcode	Instruction
D0	JNST
D1	JNH
D2	JGT
D3	JNC
D4	JNVT
D5	JNV
D6	JGE
D7	JNE
D8	JST
D9	JH
DA	JLE
DB	JC
DC	JVT
DD	JV
DE	JLT
DF	JE
E0	DJNZ
E1	RESERVED**
E2	RESERVED**
E3	BR (INDIRECT)
E4	RESERVED**
E5	RESERVED**
E6	RESERVED**
E7	LJMP

Opcode	Instruction
E8	RESERVED**
E9	RESERVED**
EA	RESERVED**
EB	RESERVED**
EC	RESERVED**
ED	RESERVED**
EE	RESERVED**
EF	LCALL
F0	RET
F1	RESERVED**
F2	PUSHF
F3	POPF
F4	RESERVED**
F5	RESERVED**
F6	RESERVED**
F7	TRAP
F8	CLRC
F9	SETC
FA	DI
FB	EI
FC	CLRVT
FD	NOP
FE	*DIV/DIVB/MUL/MULB
FF	RST

*Two Byte Instruction
**Opcodes which do not have a corresponding instruction will not generate an interrupt if executed.

7.0 INSTRUCTION SUMMARY

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	

7.0 INSTRUCTION SUMMARY (Continued)

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↕	↕	↕	↕	↑	—	
NEG/NEGB	1	D ← 0 − D	↕	↕	↕	↕	↑	—	
INC/INCB	1	D ← D + 1	↕	↕	↕	↕	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↕	↕	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↕	↕	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↕	↕	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — — lsb ← 0	↕	?	↕	↕	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — — lsb → C	↕	?	↕	0	—	↕	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — — lsb → C	↕	↕	↕	0	—	↕	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	

7.0 INSTRUCTION SUMMARY (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

8.0 OPCODES, INSTRUCTION LENGTH AND STATE TIMES

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊙ NORMAL			INDIRECT⊙ AUTO-INC.		INDEXED⊙ SHORT			INDEXED⊙ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES①	BYTES	STATE① TIMES①
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

272110-20

NOTES:
*Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.
① Number of state times shown for internal/external operands.
② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
① State times shown for 16-bit bus.

8.0 OPCODES, INSTRUCTION LENGTH AND STATE TIMES (Continued)

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊙ NORMAL			INDIRECT⊙ AUTO-INC.		INDEXED⊙ SHORT			INDEXED⊙ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES④	BYTES	STATE① TIMES④
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP②	F7	1	21/24

272110-21

NOTES:
① Number of state times shown for internal/external operands.
② The assembler does not accept this mnemonic.
④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
⑤ State times for stack located internal/external.
⑥ State times shown for 16-bit bus.

8.0 OPCODES, INSTRUCTION LENGTH AND STATE TIMES (Continued)

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.(8)

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	D8	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3 byte instructions. They require 9 state times if the jump is taken, 5 if it is not.(8)

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	E0	3	5/9 STATE TIME (NOT TAKEN/TAKEN)(8)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES(8)
	OP	B		OP	B		OP	B	
SHL	09	3	SHL8	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

8.0 OPCODES, INSTRUCTION LENGTH AND STATE TIMES (Continued)

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST(6)	FF	1	166	SKIP	00	2	4

NORMALIZE

MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:
6. This instruction takes 2 states to pull $\overline{\text{RESET}}$ low, then holds it low for at least one state time to initiate a reset. The reset takes 13 states, at which time the program restarts at location 2080H.
7. Execution will take at least 8 states, even for 0 shift.
8. State times shown for 16-bit bus.

9.0 INTERRUPT TABLE

Vector	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software Trap	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

10.0 FORMULAS

Baud Rate Calculations

Using XTAL1:

Mode 0: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{4 \cdot (B + 1)}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64 \cdot (B + 1)}$

Using T2CLK:

Mode 0: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{16 \cdot B}; B \neq 0$

Note that B cannot equal 0, except when using XTAL1 in other than Mode 0.

8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

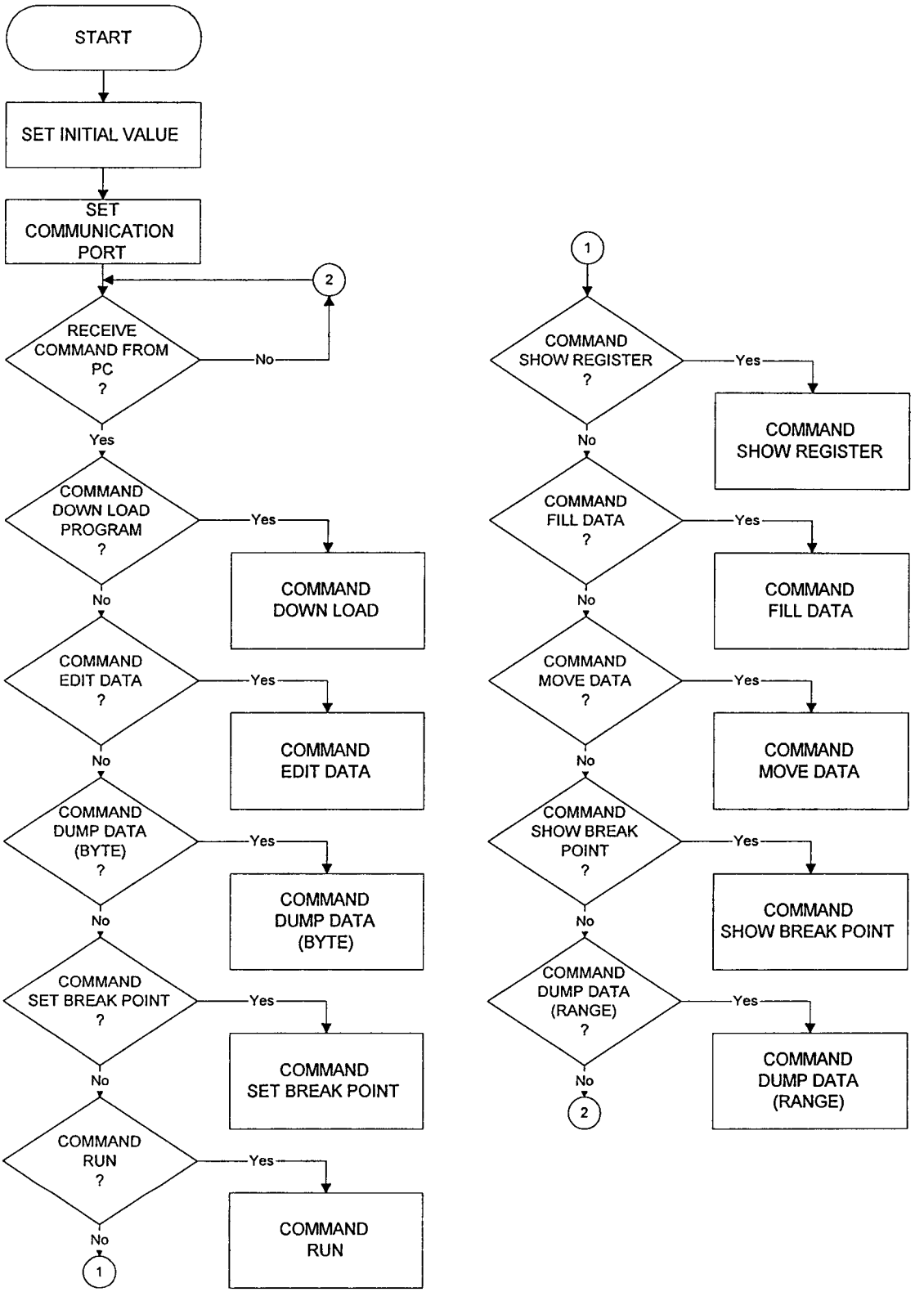
11.0 RESET STATUS

Register	RESET Value
Port 1	XXXXXXXXB
Port 2	XX0XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Port Control	XXXX0XXXB
Serial Port Status	X00XXXXXB
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B
Timer 1	0000H
Timer 2	0000H
WDT	0000H
HSI Mode	XXXXXXXXB
HSI Status	undefined
IOS0	00000000B
IOS1	00000000B
IOC0	X0X0X0X0B
IOC1	X0X0XXX1B
HSI FIFO	empty
HSI CAM	empty
HSO SFR	000000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H
Port 1	weak pullups
Port 2.6, Port 2.7	weak pullups
Ports 3 and 4	floating
HSO.0, HSO.1, HSO.2, HSO.3	low
HSO.4, HSO.5	floating
$\overline{RD}$	high
$\overline{WR}/\overline{WRL}$	high
ALE/$\overline{ADV}$	high
$\overline{BHE}/\overline{WRH}$	high
INST	low

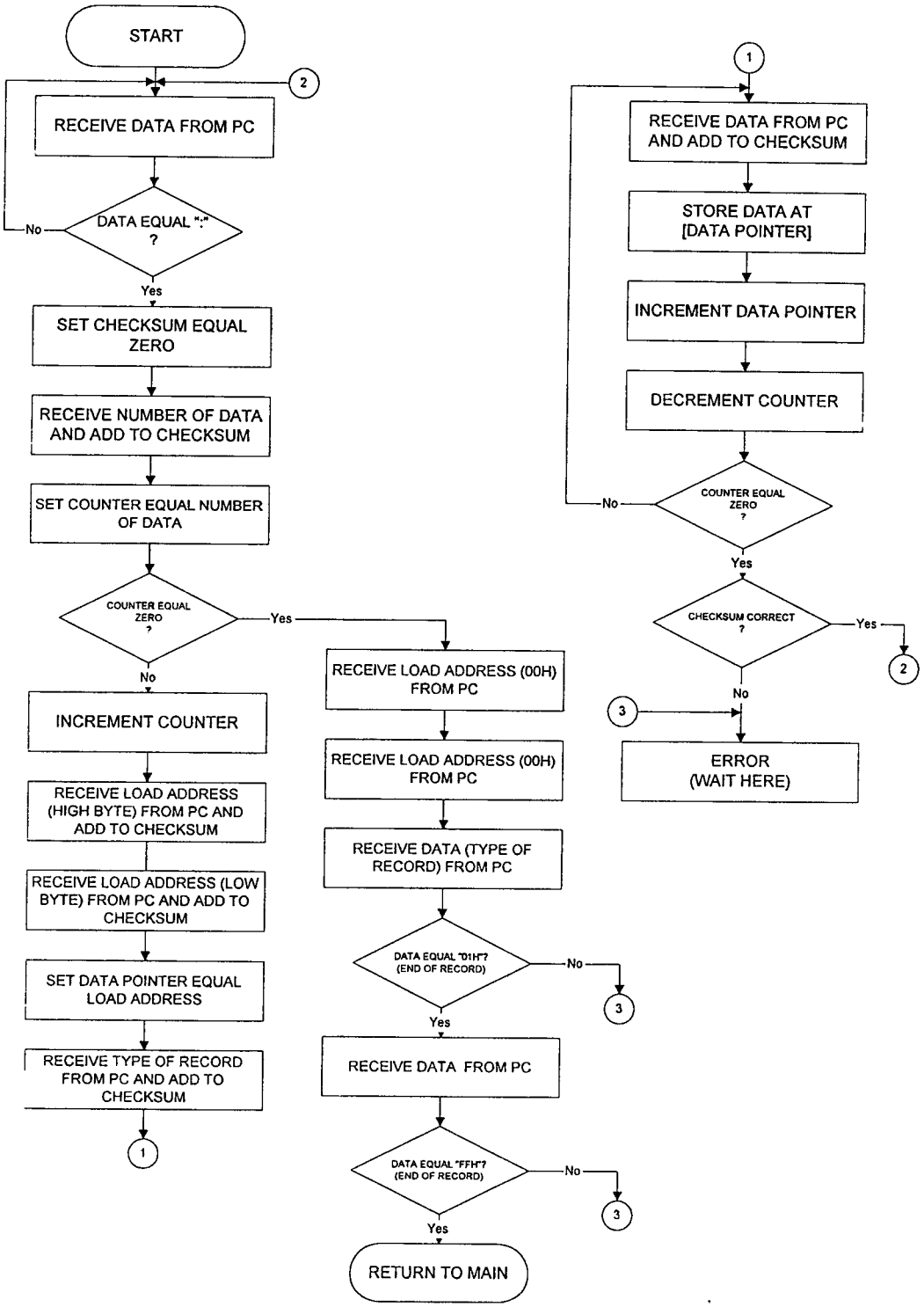
ภาคผนวก ข

โฟลว์ชาร์ตแสดงการทำงานของคำสั่งต่าง ๆ

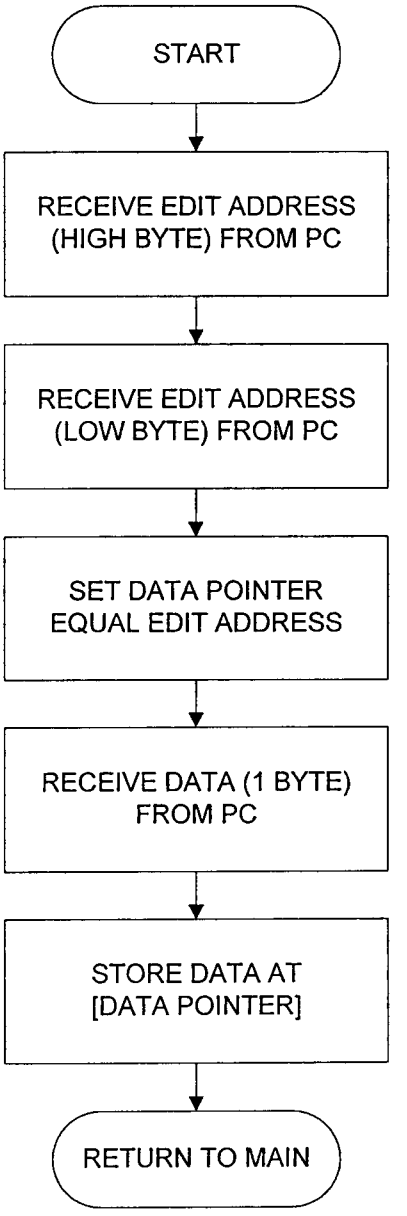
แสดงโฟลว์ชาร์ตของโปรแกรมหลัก (Main Program)



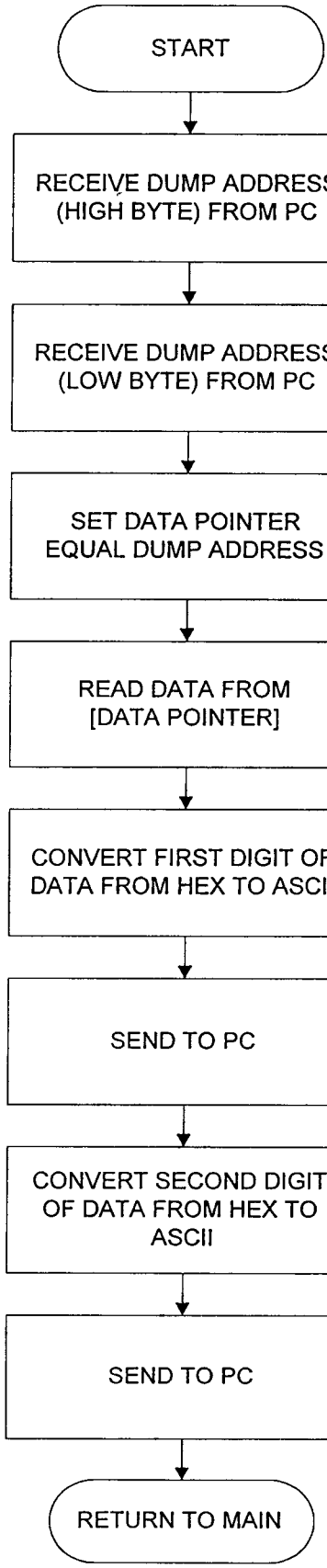
แสดงโฟลว์ชาร์ตการ Down Load Data



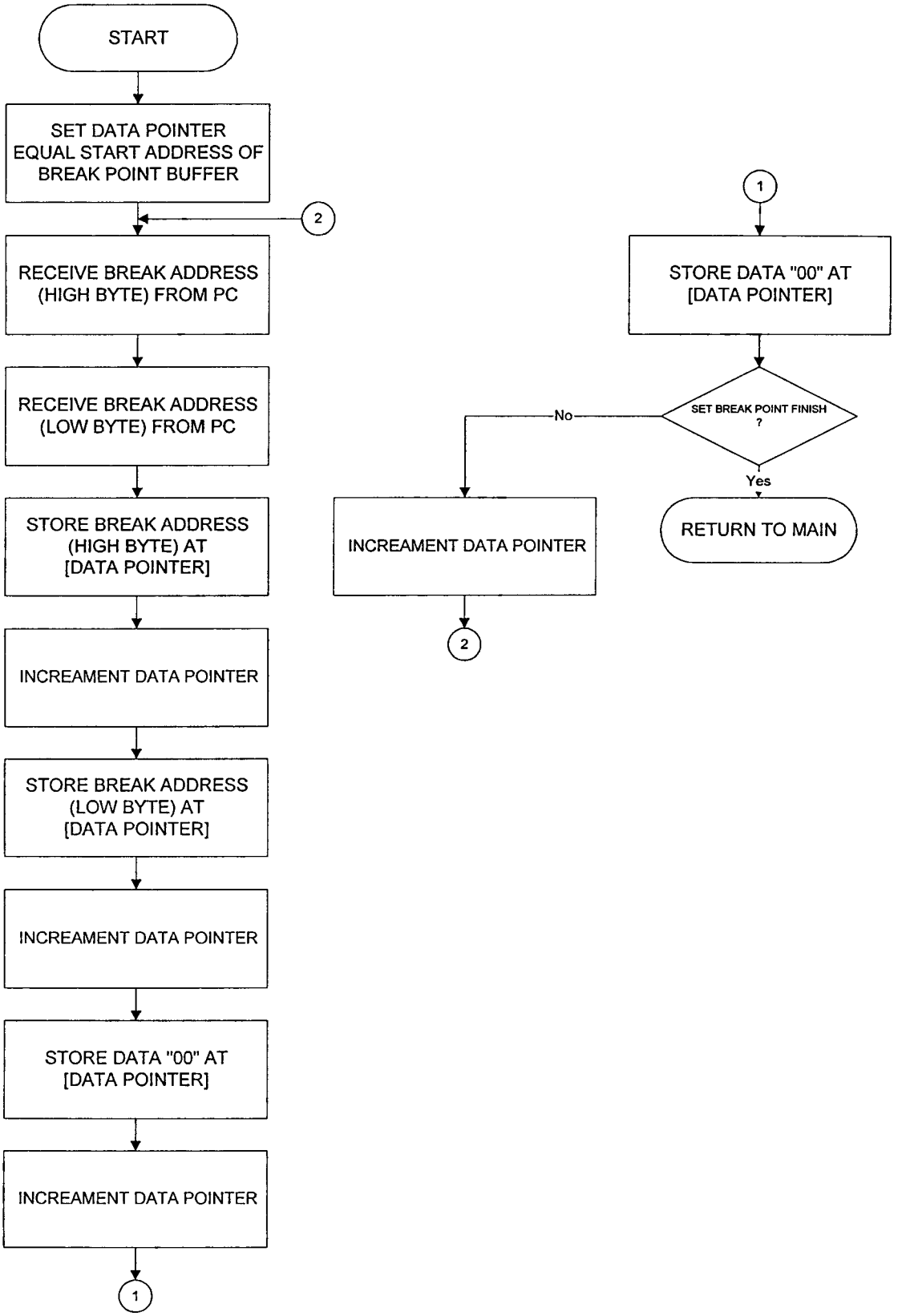
แสดงโฟลว์ชาร์ตการ Edit Data



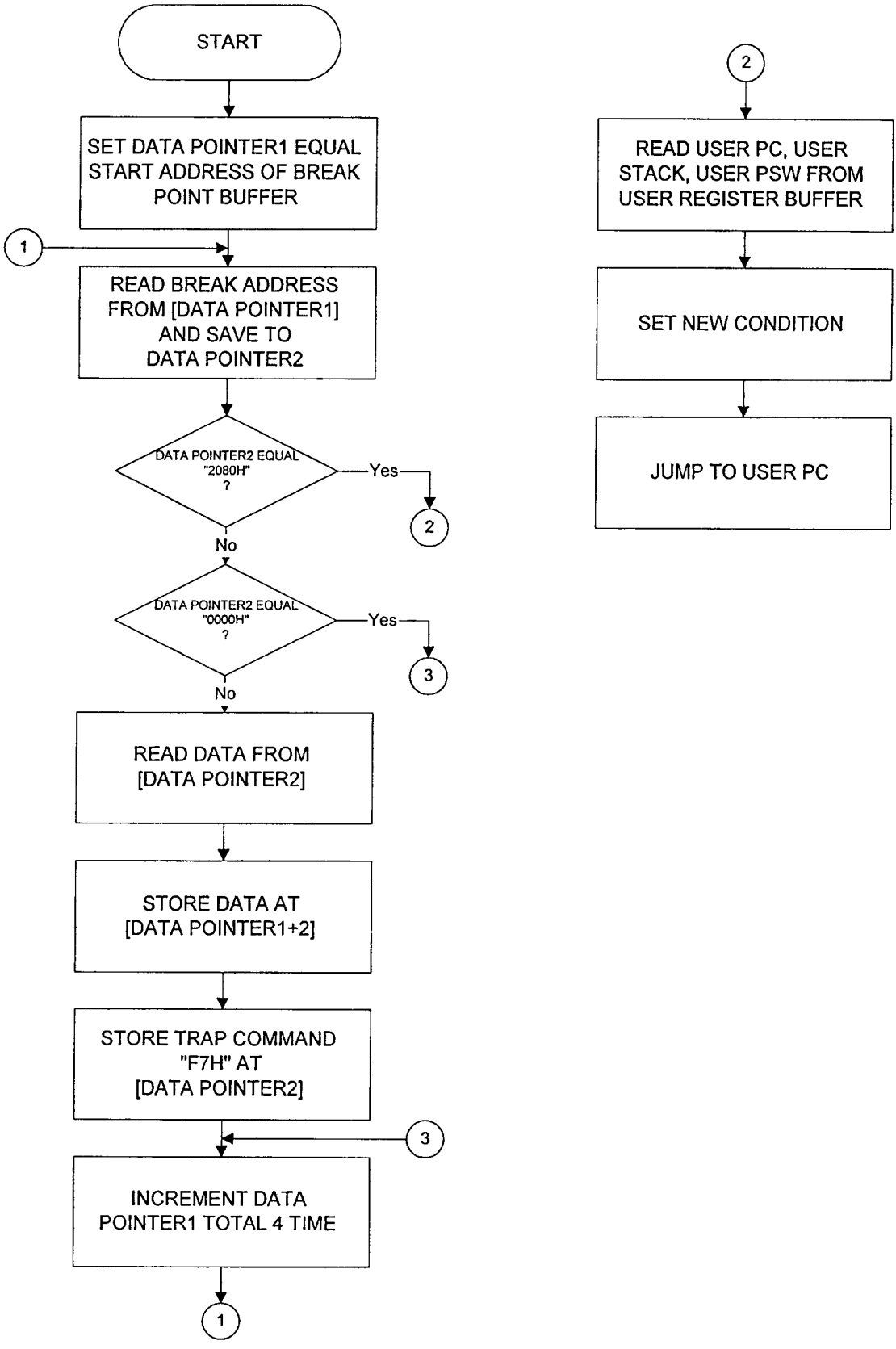
แสดงโฟลว์ชาร์ตการ Dump Memory (byte)



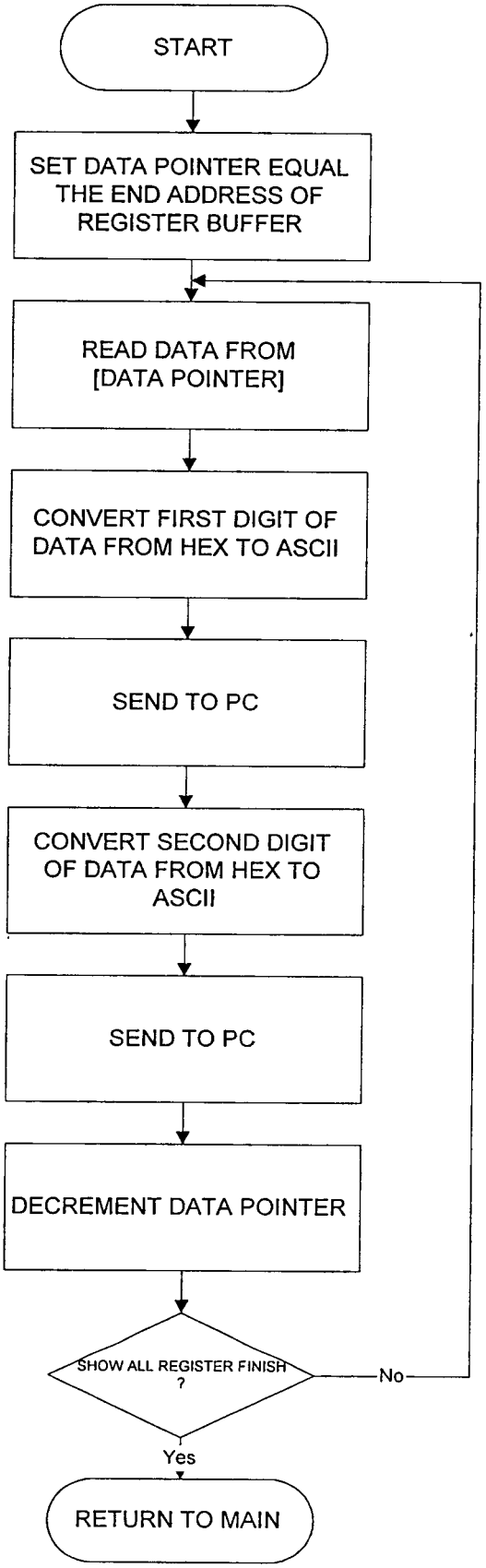
แสดงโฟลว์ชาร์ตการ Set Breakpoints



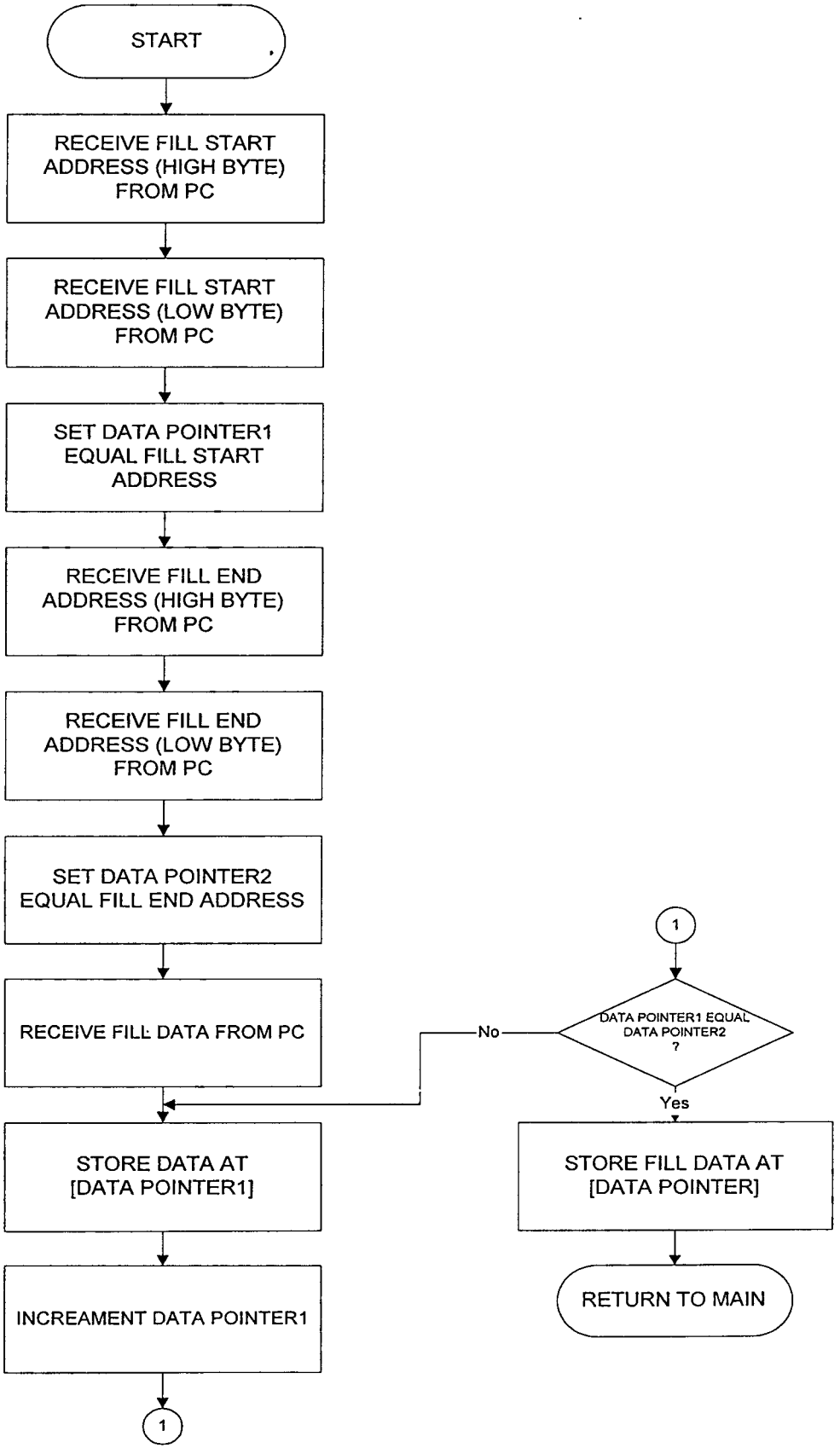
แสดงโฟลว์ชาร์ตการ Go (all condition)



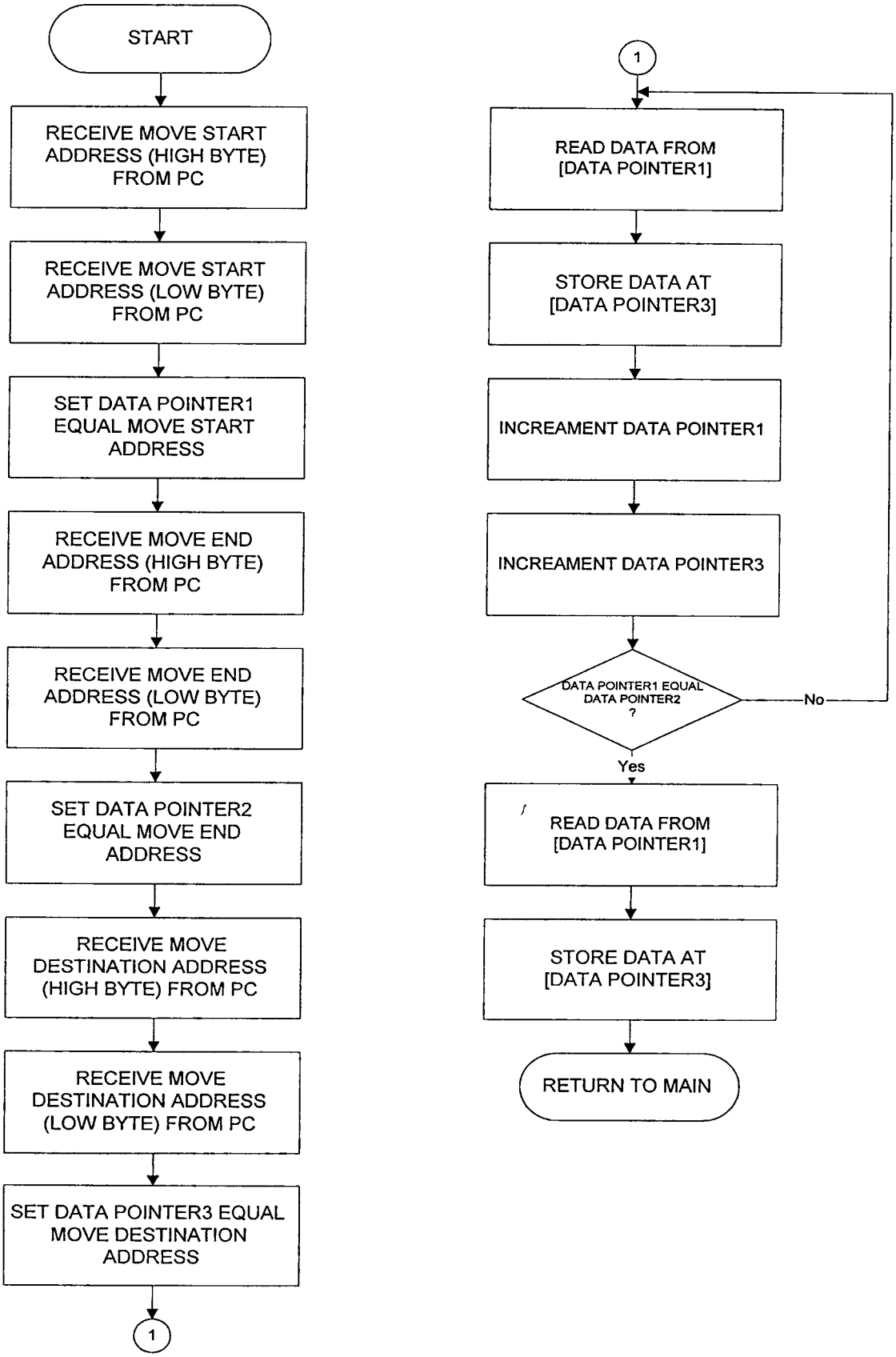
แสดงโฟลว์ชาร์ตการ Show Registers



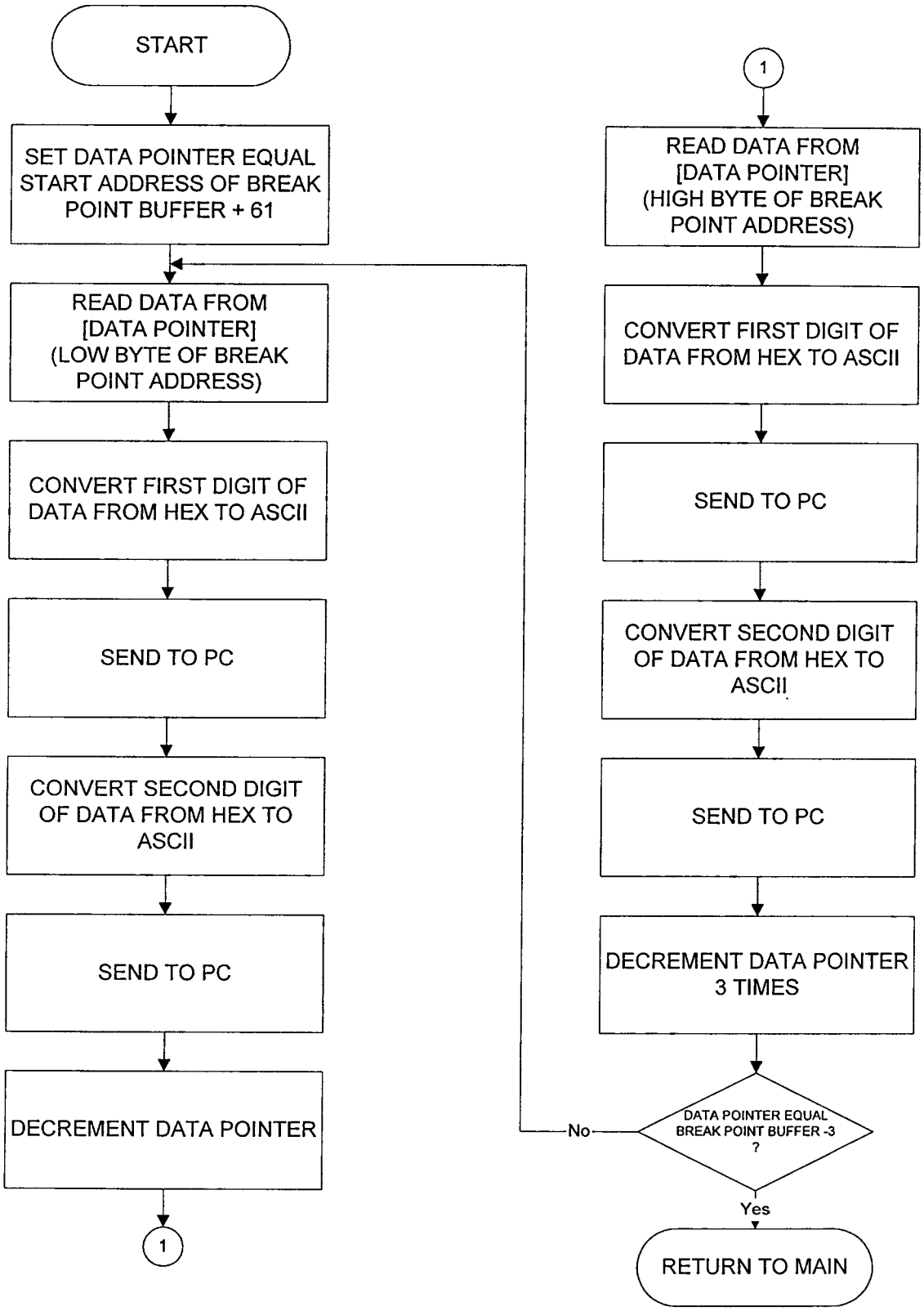
แสดงโฟลว์ชาร์ตการ Fill Data



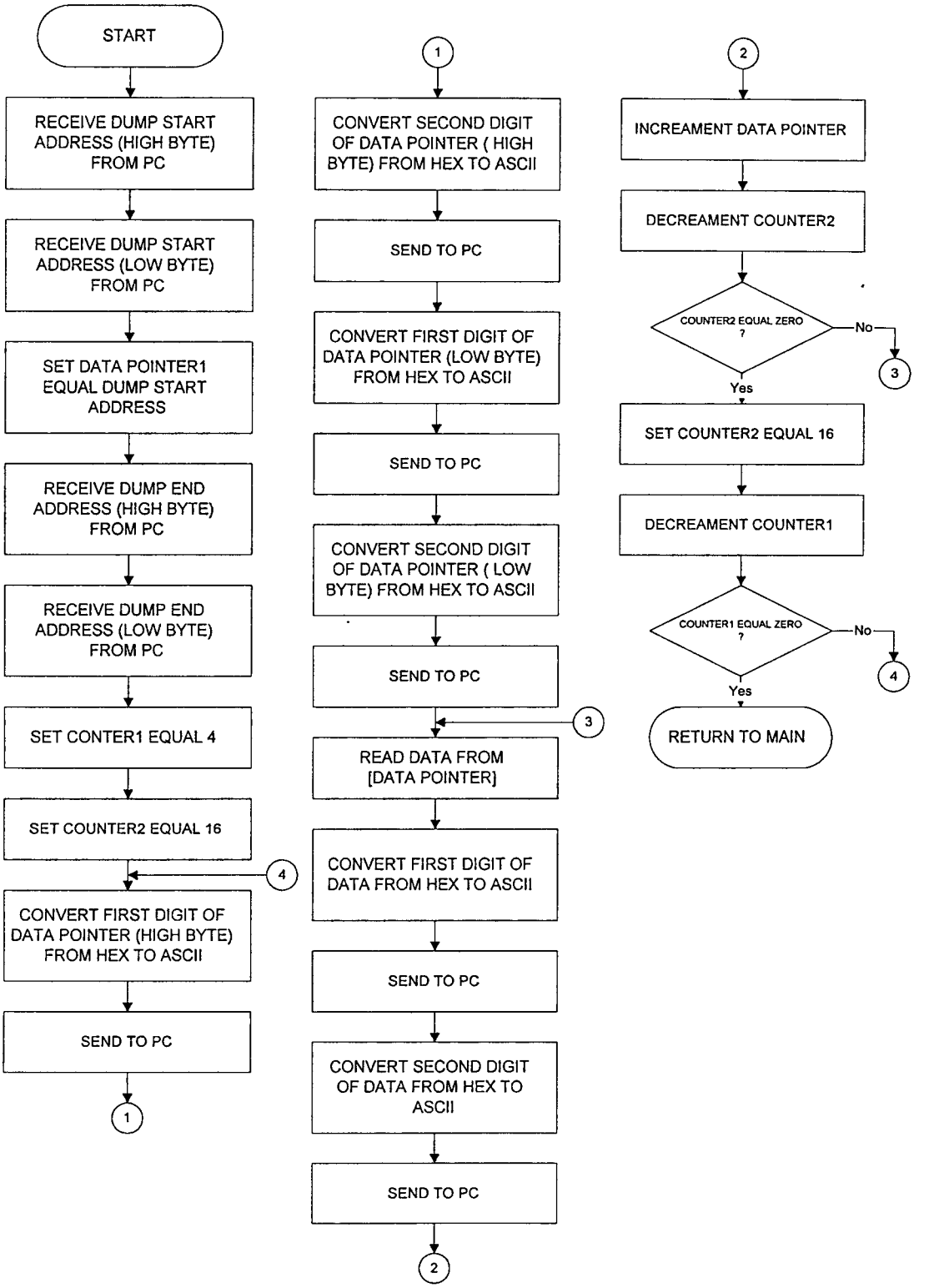
แสดงโฟลว์ชาร์ตการ Move Data



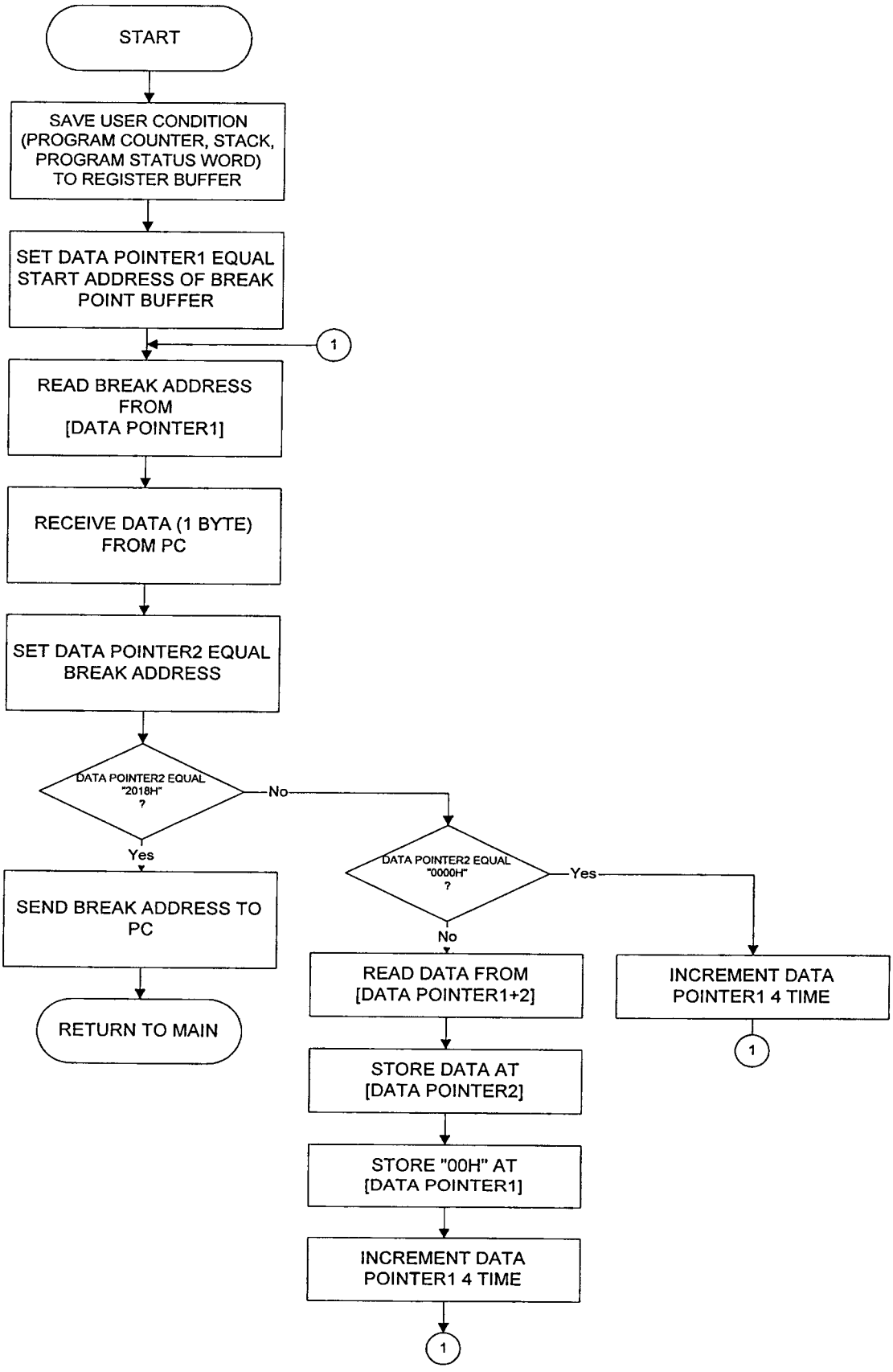
แสดงโฟลว์ชาร์ตการ Show Breakpoints



แสดงโฟลว์ชาร์ตการ Dump Memory (range)



แสดงโฟลว์ชาร์ตการทำงานเมื่อเกิด Breakpoints (Trap)



ภาคผนวก ค

ผลงานวิจัยที่ได้รับการตีพิมพ์

การประชุมวิชาการทางวิศวกรรมไฟฟ้า

ครั้งที่ ๑๗

๑ - ๒ ธันวาคม ๒๕๓๗

ณ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

การประชุมวิชาการวิศวกรรมไฟฟ้า ครั้งที่ 17

The 17th Conference of Electrical Engineering

1 - 2 ธันวาคม 2537

ณ ภาควิชาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

แผงช่วยพัฒนาโปรแกรม สำหรับไมโครคอนโทรลเลอร์ เบอร์ 8031 ราคาประหยัด ที่อาศัยเทคนิคการใช้หน่วยประมวลผลกลางร่วมกัน

Common CPU technique for a low cost 8031 µC development card

อิศราพงศ์ สินันตา * สุรพันธ์ ยิ้มมั่น ** กอบชัย เดชหาญ ***

* นักศึกษาปริญญาโท บันฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
** นักศึกษาปริญญาโท บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ
*** รองศาสตราจารย์ ภาควิชาโทรคมนาคม คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ ในปัจจุบัน µC ตระกูล MCS - 51 ได้เข้ามามีบทบาทสำคัญในด้านการศึกษา และการนำไปประยุกต์ใช้งาน ดังจะเห็นได้จาก โครงงานส่วนใหญ่ในช่วงเวลาหลายปีที่ผ่านมา เมื่อต้องใช้ CPU ทำหน้าที่ควบคุมการทำงาน จะนิยมใช้ µC เบอร์ 8031 แทน µP เบอร์ Z80 ทั้งนี้เนื่องจาก µC นั้นมีขอบเขตความสามารถในการใช้งานที่ดีกว่า µP หลายประการ วิธีการพัฒนาโปรแกรมควบคุมการทำงานของโครงงานโดยทั่วไปจะใช้ EPROM EMULATOR เพราะให้ความคล่องตัวในการทำงาน และให้ประสิทธิภาพที่สูงกว่าวิธีอื่นๆ แต่ว่า EPROM EMULATOR ที่มีจำหน่ายในปัจจุบัน มีราคาค่อนข้างสูง บทความนี้ได้นำเสนอแผงช่วยพัฒนาโปรแกรมสำหรับไมโครคอนโทรลเลอร์ เบอร์ 8031 ซึ่งให้ประสิทธิภาพในการทำงานที่ดี และที่สำคัญใช้ต้นทุนในการผลิตที่ต่ำมาก เนื่องจากได้รับการออกแบบให้มีเทคนิคในการใช้หน่วยประมวลผลกลาง (CPU) ร่วมกัน

Abstract Microcontroller series MCS - 51 have an important role for study and applications. Many projects in several years ago used CPU to control the electronic components, They used µC 8031 instead of µP Z80 because of many advantages. To develop the monitor program of project, it is easy to use several tools. The common tool uses "EPROM EMULATOR" because it provides high efficiency and easy to use, but the only disadvantage of EPROM EMULATOR is high cost. This paper presents a new and simple tool that provides good efficiency, very low cost and use common CPU technique to get advantage.

1.บทนำ

จากอดีตที่ผ่านมา เมื่อต้องการศึกษาในเรื่องไมโครโปรเซสเซอร์ขนาด 8 บิท วงการอิเล็กทรอนิกส์มักจะกล่าวถึง "Z80 CPU" และอาจมีหลายท่านที่ยกย่องให้ Z80 CPU เป็น "Chip บรมครู" เนื่องจากการที่ท่านเหล่านี้มีความรู้ ความชำนาญทางด้าน Soft Ware และ Hard Ware อยู่ได้มาจนถึงทุกวันนี้ ก็เพราะได้ศึกษา และทำความเข้าใจในรายละเอียดต่างๆ จาก Z80 CPU นั้นเอง ในปัจจุบัน Z80 CPU ยังคงเป็นที่นิยมใช้สำหรับผู้ที่ต้องการเริ่มต้นศึกษาในเรื่องเกี่ยวกับไมโครโปรเซสเซอร์อยู่ ทั้งนี้เพราะ Z80 CPU มีโครงสร้างที่ไม่ซับซ้อน และใช้ชุดคำสั่งที่เข้าใจง่าย แต่ในขณะเดียวกัน CPU ตระกูล MCS - 51 ของบริษัท INTEL ก็เริ่มเข้ามามีบทบาทที่สำคัญต่อการศึกษา และการนำไปประยุกต์ใช้งานทางอุตสาหกรรม [1] ดังจะเห็นได้จากการที่สถานศึกษาหลายๆ แห่งได้จัดให้มีการเรียนการสอนเกี่ยวกับ CPU ตระกูล MCS - 51 เพิ่มเติมให้กับนักศึกษาผู้ที่มีพื้นฐานความรู้ทางไมโครโปรเซสเซอร์จาก Z80 CPU เหตุผลประการสำคัญที่ CPU ตระกูล MCS - 51 เข้ามามีบทบาทอย่างรวดเร็ว และมีผู้นิยมนำไปใช้งานกันอย่างแพร่หลายนั้น เนื่องจาก CPU ตระกูล MCS - 51 นี้เป็นไมโครคอนโทรลเลอร์ ซึ่งมีขอบเขต และความสามารถสูงกว่าไมโครโปรเซสเซอร์ ธรรมดาทั่วๆ ไปอยู่หลายประการ [2]

ในการพัฒนาโครงงานที่ใช้ไมโครคอนโทรลเลอร์ เบอร์ 8031 เป็นตัวควบคุมนั้น เท่าที่มีผู้นิยมใช้กันมีหลายวิธีดังนี้

1. การพัฒนาโครงงานโดยอาศัย Single Board การพัฒนาด้วยวิธีนี้ ผู้ใช้จะต้องทำการแปลโปรแกรมที่เขียนขึ้นให้เป็นภาษาเครื่อง (Opcode) โดยการเปิดรหัสจากตารางชุดคำสั่ง เมื่อได้ Opcode แล้วจึงนำไปป้อนให้กับ Single Board เพื่อทดสอบการทำงานของโปรแกรมว่าถูกต้องตรงตามที่ต้องการหรือไม่ หากพบว่าโปรแกรมยังไม่สมบูรณ์ก็จะต้องทำการแก้ไขโปรแกรม และเริ่มต้นตามขั้นตอนเดิมอีก จนกว่าจะได้โปรแกรมที่สมบูรณ์ และมีความถูกต้องที่สุด

2. การพัฒนาโครงงานโดยใช้เครื่องคอมพิวเตอร์เชื่อมต่อร่วมกับอุปกรณ์ประกอบอื่นๆ ได้แก่

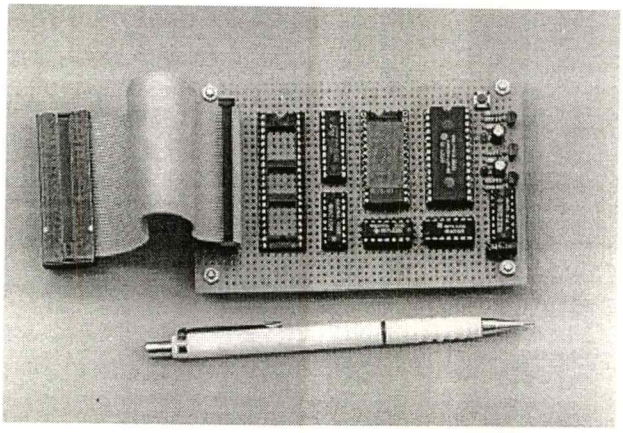
2.1 การพัฒนาโดยใช้ EPROM ที่เก็บ Monitor Program ของไมโครคอนโทรลเลอร์ เบอร์ 8031 ซึ่งใช้พอร์ทอนุกรมของเครื่องคอมพิวเตอร์สื่อสารกับผู้ใช้ โดยอาศัยจอภาพ และคีย์บอร์ดร่วมในการพัฒนาโปรแกรมด้วย การพัฒนาด้วยวิธีนี้ ผู้ใช้สามารถเขียนโปรแกรม แก้ไข และทดลองการทำงานได้โดยที่ไม่มีการโยกย้ายทางกายภาพ สิ่งที่ปรากฏบนจอภาพจะเป็นสิ่งที่เกิดขึ้นจริงในบอร์ดคอนโทรลของโครงงานที่กำลังพัฒนา ลักษณะการใช้งานจะคล้ายคลึงกับโปรแกรม DEBUG ของ DOS [3]

2.2 การพัฒนาโดยใช้ EPROM EMULATOR ที่ทำหน้าที่เสมือนตัว EPROM หรือ RAM ในบอร์ดคอนโทรลของโครงงานที่กำลังพัฒนาอยู่ การใช้งานจะใช้ร่วมกับเครื่องคอมพิวเตอร์ทั่วไป โดยมีทั้งแบบที่เชื่อมต่อผ่านทางพอร์ทขนาน (LPT) หรือผ่านทางพอร์ทอนุกรม (COM1, COM2) การใช้ EPROM EMULATOR ทำให้สามารถพัฒนาโปรแกรมได้อย่างรวดเร็ว และลดขั้นตอนในการพัฒนาโครงงานด้วยวิธีเดิมที่เคยใช้ในอดีตลงไปได้มาก [4], [5]

จากตัวอย่างวิธีการพัฒนาโครงงานบางส่วนที่กล่าวถึงไปแล้วนั้น พบว่าในแต่ละวิธียังมีข้อเสียอยู่บางประการ เช่น ในการใช้ Single Board ผู้ใช้ไม่สามารถเขียนโปรแกรมควบคุมการทำงาน 8031 ได้ครบทุกส่วน เนื่องจากบางส่วนของ 8031 ถูกใช้งานโดย Monitor Program ของ Single Board ไปแล้ว ในกรณีที่ใช้ EPROM ซึ่งเก็บ Monitor Program ของ 8031 นั้นจากประสบการณ์ที่เคยใช้พบว่า ขั้นตอนในการพัฒนาโปรแกรมด้วยวิธีนี้ยังไม่อำนวยความสะดวกให้กับผู้ใช้มากนัก เช่น ในการอ้าง Label ของคำสั่งกระโดด (Jump) ต้องระบุเป็นตัวเลข Address ของตำแหน่งปลายทาง, การแก้ไขค่า Origin Address ของโปรแกรมที่สมบูรณ์จาก 8100H ให้เป็น 0000H ก่อนนำโปรแกรมไปบรรจุลง EPROM เป็นต้น สำหรับการพัฒนาโดยใช้ EPROM EMULATOR นั้นสามารถทำการพัฒนาโปรแกรมได้คล่องตัวกว่าวิธีการอื่นๆ ที่กล่าวถึงไปแล้ว แต่ว่าประสบปัญหาในเรื่องของราคาที่ยังสูงอยู่มาก

2. แนวคิด และหลักการ

บทความนี้มีวัตถุประสงค์เพื่อนำเสนอรายละเอียดในการสร้างแผงช่วยพัฒนาโปรแกรมสำหรับไมโครคอนโทรลเลอร์ เบอร์ 8031 ซึ่งเหมาะสำหรับผู้ที่ต้องการลดค่าใช้จ่ายในการพัฒนาโครงงาน เพราะต้นทุนในการผลิตต่ำกว่า EPROM EMULATOR ที่มีจำหน่ายมาก ในขณะที่สามารถใช้พัฒนาโปรแกรมได้อย่างมีประสิทธิภาพ วิธีการใช้งานแผงช่วยพัฒนาฯ นี้จะคล้ายคลึงกับการใช้ EPROM EMULATOR แตกต่างตรงที่สามารถทำงานได้เพียง Down Load Mode เท่านั้น ในขณะที่ EPROM EMULATOR ยังสามารถทำงานในลักษณะ Remote Editor Mode ได้อีกด้วย (จากประสบการณ์ในการใช้งาน EPROM EMULATOR พบว่า Mode ที่ใช้บ่อย และสำคัญที่สุดคือ Down Load Mode สำหรับ Remote Editor Mode นั้น มีโอกาสใช้งานน้อยกว่า) เหตุผลประการสำคัญที่ทำให้ต้นทุนในการผลิตแผงช่วยพัฒนาฯ ต่ำกว่าราคาของ EPROM EMULATOR ก็คือ ได้นำเอาเทคนิคการใช้อุปกรณ์ร่วมกันมาประยุกต์ใช้ ทั้งนี้เนื่องจากบอร์ดคอนโทรลที่ผู้ใช้สร้างขึ้น หรือที่มีจำหน่ายอยู่ในปัจจุบัน (ผลิตภัณฑ์ของบริษัท ETT, SILA) จะมี CPU เบอร์ 8031 และส่วนสื่อสารผ่านพอร์ทอนุกรมรวมอยู่บนบอร์ดคอนโทรลนั้นแล้ว จึงสามารถนำ 2 ส่วนนี้มาใช้งานร่วมกับแผงช่วยพัฒนาฯ ได้ ส่งผลให้ส่วนประกอบของแผงช่วยพัฒนาฯ ลดลง ดังแสดงในรูปที่ 1.



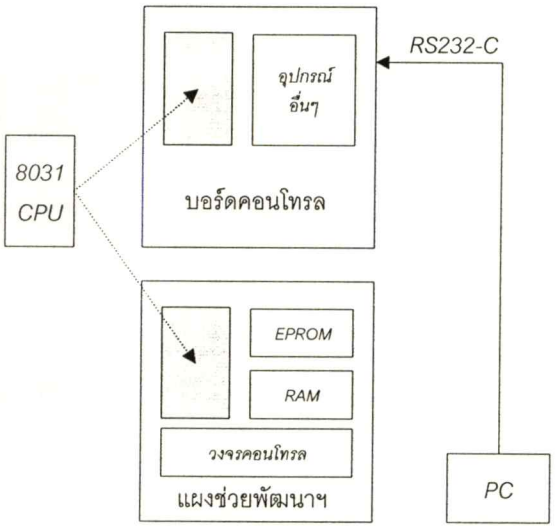
รูปที่ 1. แผงช่วยพัฒนาโปรแกรมสำหรับไมโครคอนโทรลเลอร์เบอร์ 8031

จากรูปที่ 2. แสดงเทคนิคการใช้งาน CPU ร่วมกันระหว่างบอร์ดคอนโทรลกับแผงช่วยพัฒนาฯ โดย CPU (8031) จะมีการทำงานอยู่ 2 สภาวะคือ "สภาวะ Ready" และ "สภาวะ Run"

สภาวะ Ready CPU จะทำงานตาม Monitor Program ที่บรรจุอยู่ใน EPROM (2716) บนแผงช่วยพัฒนาฯ โดยจะรอรับ INTEL HEX FILE ที่ Down Load มาจากเครื่องคอมพิวเตอร์ ผ่านทางพอร์ทอนุกรมของบอร์ดคอนโทรล แล้วนำข้อมูลที่ได้ไปเก็บไว้ภายใน RAM (6264) ของแผงช่วยพัฒนาฯ

สภาวะ Run เมื่อ CPU สิ้นสุดการรับข้อมูลจากเครื่องคอมพิวเตอร์ CPU จะสั่งงานให้วงจรคอนโทรล (ประกอบด้วย IC TTL

เบอร์ 7432, 7476, 74123) ทำการ Reset CPU และควบคุมให้ CPU ทำงานตาม Monitor Program จาก RAM (6264) แทน Monitor Program เดิม (Monitor Program ในสภาวะนี้ก็คือ INTEL HEX FILE ที่ได้จากการ Down Load นั้นเอง) ผู้ใช้สามารถควบคุมให้ CPU กลับไปทำงานในสภาวะ Ready เพื่อรอรับ INTEL HEX FILE ได้ใหม่โดยกด Receive SW1 บนแผงช่วยพัฒนาฯ



รูปที่ 2. เทคนิคการใช้งาน CPU ร่วมกันระหว่างบอร์ดคอนโทรลกับแผงช่วยพัฒนาฯ

3. ผลการทดลอง

จากข้อมูลในตารางที่ 1. แสดงถึงการเปรียบเทียบต้นทุนในการผลิต EPROM EMULATOR ทั้ง 3 แบบ พบว่าต้นทุนของแผงช่วยพัฒนาฯต่ำกว่า EPROM EMULATOR ที่มีจำหน่ายในปัจจุบันประมาณ 3 ถึง 7 เท่า

EPROM EMULATOR	ต้นทุนต่อหน่วย(บาท)
EE - 232	2450[1]
ET - EM8	1150[2]
แผงช่วยพัฒนาฯ	350[3]

ตารางที่ 1. แสดงการเปรียบเทียบต้นทุนการผลิตของ EPROM EMULATORS

หมายเหตุ (1) ข้อมูลจาก Catalog ของบริษัท SILA RESEARCH

(2) ข้อมูลจาก Catalog ของบริษัท ETT

(3) ข้อมูลจากการสำรวจ และเลือกซื้ออุปกรณ์ราคาถูกที่สุดจากร้านจำหน่ายอุปกรณ์อิเล็กทรอนิกส์ย่านบ้านหม้อ (กรกฎาคม 2537)

SPECIFICATION	EPROM EMULATOR		
	EE - 232	ET - EM8	แผงช่วยพัฒนาฯ
-ทำงานเป็น EPROM	5[1]	5[1]	5[2]
-ทำงานเป็น RAM	-	3[3]	-
-พอร์ทที่เชื่อมต่อกับเครื่องคอมพิวเตอร์	อนุกรม	ขนาน	อนุกรม
-อัตราการรับข้อมูล (BPS)	4[4]	-	9,600
-ชนิดของ HEX FILE ที่ใช้โหลดข้อมูล	INTEL	INTEL ,BIN	INTEL
-Auto Reset	ปากคีบ	-	มี
-การใช้งาน Down Load Mode	มี	มี	มี
-การใช้งาน Remote Editor Mode	มี	มี	-
-LED แสดงสภาวะการทำงาน	มี	มี	มี
-CPU ขนาด 8 บิทที่ใช้งานร่วมได้	ALL	ALL	4[5]
-บอร์ดคอนโทรลที่ใช้งานร่วมได้	ALL	ALL	ALL[6]
-ตัวอย่าง Assembler ที่ใช้งานร่วมได้	SXA51	A51	SXA51

ตารางที่ 2. แสดงการเปรียบเทียบคุณสมบัติต่างๆ ของ EPROM EMULATORS

หมายเหตุ (1) 2716, 2732, 2764, 27128 และ 27256

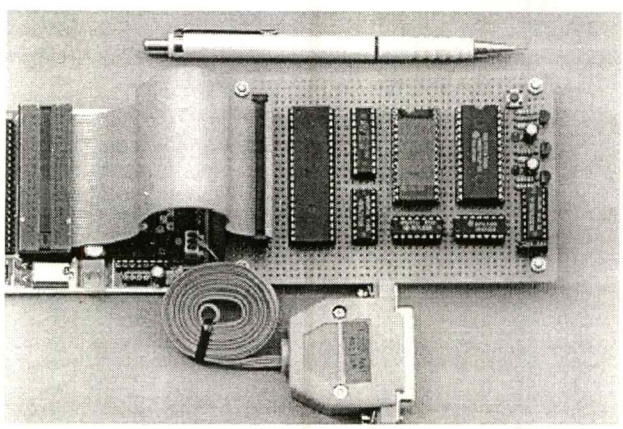
(2) เหมือน (1) แต่ในการเปลี่ยนเบอร์ EPROM จะต้องปรับปรุงวงจรบางส่วน

(3) 6116, 6264 และ 62256

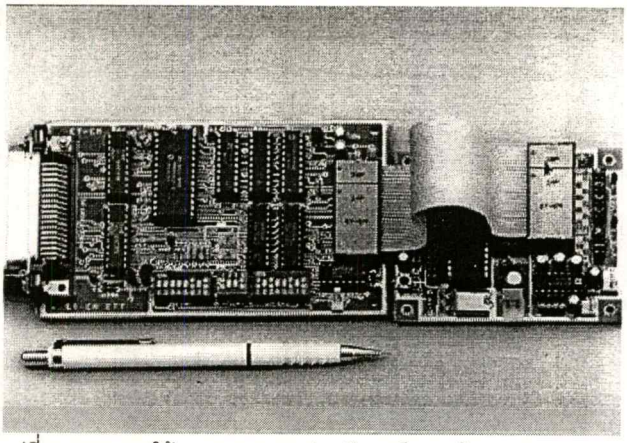
(4) 19200, 9600, 4800 และ 2400 BPS

(5) 8031, 8032, 8051 และ 8052 เท่านั้น

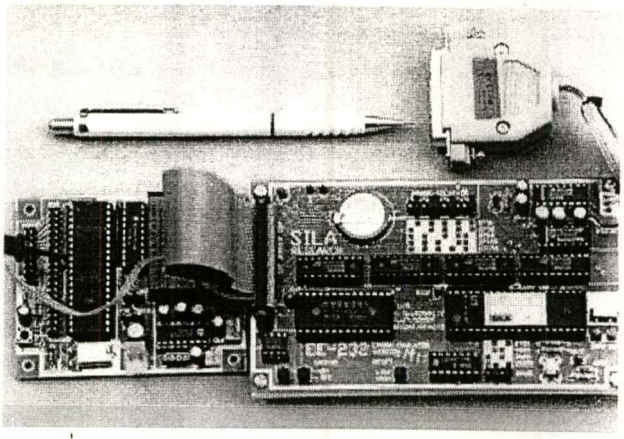
(6) จะต้องมีพอร์ทอนุกรม



รูปที่ 3. แสดงการใช้งานแผงช่วยพัฒนาฯ ร่วมกับบอร์ดคอนโทรล (V31)



รูปที่ 4. แสดงการใช้งาน ET - EM8 ร่วมกับบอร์ดคอนโทรล (V31)



รูปที่ 5. แสดงการใช้งาน EE - 232 ร่วมกับบอร์ดคอนโทรล (V31)

PROGRAM	SIZE	EPROM EMULATOR	เวลาที่ใช้ในการ Down Load File (วินาที)										
			1	2	3	4	5	6	7	8	9	10	AV
PORT1_1K.HEX	1KB	EE - 232	3.3	3.3	3.4	3.2	3.3	3.3	3.3	3.3	3.3	3.3	3.3
		ET - EM8	0.4	0.4	0.5	0.4	0.5	0.4	0.5	0.5	0.5	0.4	0.45
		แผงช่วยพัฒนาฯ	3.4	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.31
PORT1_8K.HEX	8KB	EE - 232	23.8	23.9	23.8	23.9	23.8	23.8	23.8	23.7	23.7	23.8	23.8
		ET - EM8	1.0	1.1	1.1	1.2	0.8	1.1	1.2	1.1	1.1	1.0	1.07
		แผงช่วยพัฒนาฯ	23.8	23.8	23.8	23.7	23.7	23.8	23.8	23.7	23.7	23.7	23.75

ตารางที่ 3. แสดงการเปรียบเทียบเวลาที่ใช้ในการ Down Load File ของ EPROM EMULATORS

PROGRAM	SIZE	EPROM EMULATOR	การทำงานของบอร์ดคอนโทรลเมื่อสิ้นสุดการ Down Load File									
			1	2	3	4	5	6	7	8	9	10
PORT1_1K.HEX	1KB	EE - 232	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK
		ET - EM8	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK
		แผงช่วยพัฒนาฯ	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
PORT1_8K.HEX	8KB	EE - 232	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK	HW_OK
		ET - EM8	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK	RS_OK
		แผงช่วยพัฒนาฯ	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

ตารางที่ 4. แสดงการเปรียบเทียบการทำงานของบอร์ดคอนโทรลเมื่อสิ้นสุด การ Down Load File ของ EPROM EMULATORS

หมายเหตุ 1. HW_OK หมายถึง จะต้องต่อปากคีบของ EE - 232 เข้ากับขา Reset ของ CPU เพื่อให้บอร์ดคอนโทรลทำงานได้อย่างถูกต้อง

2. RS_OK หมายถึง จะต้องทำการ Reset CPU ทุกครั้ง เพื่อให้บอร์ดคอนโทรลทำงานได้อย่างถูกต้อง

3. PORT1_1K.HEX เป็นโปรแกรมขนาด 1KB ที่สั่งงานให้ PORT 1 ของ 8031 มีลอจิกเป็น "01010101B"

4. PORT1_8K.HEX เหมือน 3. แต่มีขนาด 8KB

ข้อมูลในตารางที่ 3. ได้จากการทดลอง Down Load File 2 ขนาด (1KB, 8KB) จากเครื่องคอมพิวเตอร์ (CPU 80486DX - 33) ไปยังบอร์ดคอนโทรล (V31) ไฟล์ละ 10 ครั้งพบว่า ET - EM8 ใช้เวลาในการ Down Load น้อยที่สุด เนื่องจากมีการเชื่อมต่อกับเครื่องคอมพิวเตอร์ผ่านทางพอร์ทขนาน (LPT) สำหรับ EE - 232 และแผงช่วยพัฒนาฯ นั้นพบว่าใช้เวลาในการ Down Load ใกล้เคียงกัน

ผลการทดลองในตารางที่ 4. พบว่าบอร์ดคอนโทรลที่ใช้ EE - 232 หรือ ET - EM8 เมื่อสิ้นสุด การ Down Load File จะต้องทำการ Reset CPU ทุกครั้ง บอร์ดคอนโทรลจึงจะทำงานได้อย่างถูกต้อง แต่ถ้า Down Load File โดยใช้แผงช่วยพัฒนาฯ บอร์ดคอนโทรลจะทำงานตามโปรแกรมทันทีเมื่อ Down Load File เสร็จเนื่องจากโครงสร้าง และหลักการทำงานของแผงช่วยพัฒนาฯ ได้ออกแบบให้มีส่วนของ Auto Reset อยู่ภายในแล้ว

4. บทสรุป

บทความนี้กล่าวถึง แผงช่วยพัฒนาฯ ซึ่งสามารถใช้พัฒนาโครงงานได้เป็นอย่างดี และเหมาะสำหรับผู้ที่ต้องการลดค่าใช้จ่ายในการทำโครงงาน เพราะเมื่อเปรียบเทียบต้นทุนในการผลิตกับ EPROM EMULATOR ที่มีจำหน่ายในปัจจุบัน พบว่าแตกต่างกันมาก แต่ถ้าผู้ใช้ไม่ประสบปัญหาเรื่องงบประมาณ และต้องการเพิ่มความคล่องตัวตลอดจนเพิ่มประสิทธิภาพในการพัฒนาโครงงาน ก็ขอแนะนำให้เลือกใช้ EPROM EMULATOR ที่มีจำหน่ายอยู่แล้ว

5. เอกสารอ้างอิง

[1] จิระศักดิ์ ชาญวุฒิธรรม, อิศราพงศ์ สินันตา, กอบชัย เดชหาญ, *"ไมโครคอนโทรลเลอร์ในระบบควบคุมแบบแยก ส่วนผ่านพอร์ทอนุกรมเพื่องานควบคุมโรงงานขนาดเล็ก"*, การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 15, หน้า 4_17 - 4_23, 3 - 4 ธันวาคม 2535.

[2] MacKenzie I. Scott, *"The 8051 microcontroller"*, Macmillan Publishing Company, 1992.

[3] บริษัท ศิลารีเสิร์ช จำกัด, *"คู่มือการใช้งาน REM31"*, ตุลาคม 2534.

[4] บริษัท อีทีที จำกัด, *"คู่มือการใช้งาน ET - EM8 "*, สิงหาคม 2533.

[5] บริษัท ศิลารีเสิร์ช จำกัด, *"คู่มือการใช้งาน EE - 232 V3.0"*, มีนาคม 2536.

6. ภาคผนวก

6.1 ภาคผนวก (ก)

Monitor Program ของแผงช่วยพัฒนาฯ

```
        ORG 0000H
        MOV tmod, #20H    ;Set baud rate
        MOV scon, #50H    ;9600, n, 8, 1, p
        MOV th1, #0FDH
        SETB tr1
LOAD1:  ACALL INCHAR      ;Get code "0DH"
        ACALL INCHAR      ;Get code "0AH"
        ACALL INCHAR      ;Get code "3Ah"
        CJNE a, #':', ERROR
        MOV r1, #00H      ;Initial check sum
        ACALL GETHEX      ;Get number of data
        MOV b, a
        JZ DONE
LOAD2:  INC b
        ACALL GETHEX      ;RAM start addres(high)
        MOV dph, a
        ACALL GETHEX      ;RAM start address(low)
        MOV dpl, a
        ACALL GETHEX      ;Get type of record
LOAD3:  ACALL GETHEX      ;Get data
        MOVX @dptr, a     ;Store data in RAM
        INC dptr
        DEC b
        MOV a, b
        JNZ LOAD3
        MOV a, r1
        JZ LOAD1          ;Test check sum
ERROR:  SJMP ERROR        ;Error reset and start again
DONE:   ACALL GETHEX
        ACALL GETHEX
        ACALL GETHEX      ;Get type of record
        CJNE a, #01H, ERROR
        SJMP RUN          ;If end of record then run
INCHAR: JNB ri, INCHAR    ;Receive data from PC.
        CLR ri
        MOV a, sbuf
        RET
```

```
RUN:    MOVX a, @dptr    ;Reset CPU and change
                         ;Monitor Program
GETHEX: ACALL INCHAR     ;Convert 2 bytes of ASCII
        ACALL ATOH       ;to HEX 1 byte
        SWAP a
        MOV r0, a
        ACALL INCHAR
        ACALL ATOH
        ORL a, r0
        MOV r2, a
        ADD a, r1
        MOV r1, a
        MOV a,r2
        RET
ATOH:   CJNE a, #'A', $ + 3  ;Convert ASCII to HEX
        JC ATOH2
        ADD a, #09H
ATOH2:  ANL a, #0FH
        RET
        END
```

6.2 ภาคผนวก.(ข)

ขั้นตอนการใช้งานแผงช่วยพัฒนาฯ ในการ Down Load File

1. กด Receive SW1 ของแผงช่วยพัฒนาฯ ให้ LED "Ready" สว่าง
2. ใช้คำสั่งของ DOS ในการ Down Load ตามรูปแบบนี้
   COPY FILENAME.HEX COM1 (ใช้พอร์ทอนุกรม 1)
   COPY FILENAME.HEX COM2 (ใช้พอร์ทอนุกรม 2)
3. ในขณะที่แผงช่วยพัฒนาฯ กำลังรับข้อมูลอยู่จะสังเกตุเห็น LED "Ready" กระพริบ
4. เมื่อ Down Load File เสร็จ LED "Run" จะสว่าง ในสภาวะนี้บอร์ดคอนโทรลจะทำงานตามโปรแกรมที่ผู้ใช้ต้องการ

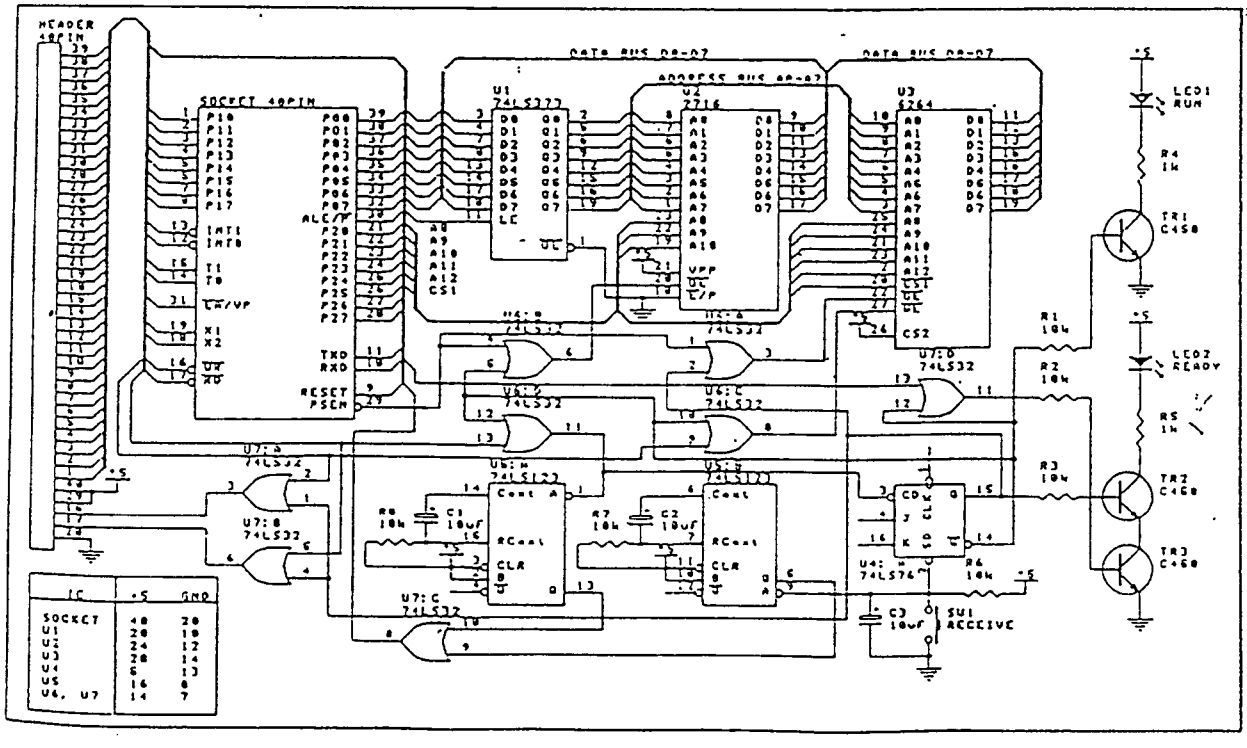
<u>หมายเหตุ</u> ก่อนทำการ Down Load File จะต้องทำการกำหนดค่าต่างๆ ให้พอร์ทอนุกรมด้วยคำสั่ง MODE (ทำครั้งแรกก็พอ)

MODE COM1:96, n, 8, 1, p (ใช้พอร์ทอนุกรม 1)

MODE COM2:96, n, 8, 1, p (ใช้พอร์ทอนุกรม 2)

6.3 ภาคผนวก (ค)

วงจรที่สมบูรณ์ของแผงช่วยพัฒนาฯ แสดงดังรูปที่ 6. จุดที่ต้องทำความเข้าใจคือ การต่อ Header 40 Pin เข้ากับ 8031 จะต้องพิจารณาตำแหน่งขาของ Dip Jumper 40 Pin ประกอบด้วย.



รูปที่ 6. แสดงวงจรที่สมบูรณ์ของแผงช่วยพัฒนาฯ

ประวัติผู้เขียน

นายอิศราพงศ์ สินันตา เกิดเมื่อวันที่ 23 กรกฎาคม พ.ศ. 2514 ที่จังหวัดเชียงใหม่ สำเร็จการศึกษาปริญญาอุตสาหกรรมศาสตร์บัณฑิต สาขาเทคโนโลยีไฟฟ้าอุตสาหกรรม คณะวิศวกรรมศาสตร์ จากสถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ ปีการศึกษา 2535 เข้าศึกษาต่อระดับปริญญาวิศวกรรมศาสตร์มหาบัณฑิต สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2536 มีผลงานทางวิชาการที่ได้รับการยอมรับ นอกจากงานวิทยานิพนธ์ จำนวน 2 เรื่อง ได้แก่ ไมโครคอนโทรลเลอร์ในระบบควบคุมแบบแยกส่วนผ่านพอร์ตอนุกรมเพื่องานควบคุมในโรงงานขนาดเล็ก และแผงช่วยพัฒนาโปรแกรม สำหรับไมโครคอนโทรลเลอร์ เบอร์ 8031 ราคาประหยัด ที่อาศัยเทคนิคการใช้หน่วยประมวลผลกลางร่วมกัน ประสบการณ์ในการทำงาน เคยเป็นอาจารย์ประจำสาขาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษมบัณฑิต เมื่อปี พ.ศ. 2536 - 2537 ปัจจุบันเป็นหัวหน้าแผนกวิศวกรรมการผลิต บริษัทไทย เอ็น เจ อาร์ จำกัด