ปีการศึกษา 2537

อีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51

โดย

นายชำนาญ เฉลิมยุทธ 35.102093

นายคำรงศักดิ์ ไชยกิตติโสภณ 35.103272

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

เรื่อง อีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51

ผู้จัดทำ

1.นายชำนาญ เฉลิมยุทธ 35.102093

2.นายคำรงศักดิ์ ไชยกิตติโสภณ 35.103272

..... อาจารย์ที่ปรึกษา

(อาจารย์ สิงห์ทอง พัฒนเศษฐานนท์)

อีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51

นายชำนาญ เฉลิมยุทธ
นายคำรงศักดิ์ ไชยกิตติโสภณ

อาจารย์สิงห์ทอง พัฒนเศรษฐานนท์ อาจารย์ที่ปรึกษา

ปีการศึกษา 2537

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการเสนอแนวทางในการพัฒนาอุปกรณ์ที่เรียกว่า อีมูเลเตอร์ (Emulator) ของไมโครโปรเซสเซอร์(Microprocessor) 8085 และ ไมโครคอนโทรลเลอร์(Microcontroller) MCS-51 โดยการประยุกต์ใช้ เครื่องไมโครคอมพิวเตอร์ เป็นเทอร์มินัล(Terminal)ในการติดต่อกับผู้ใช้ การใช้งานอีมูเลเตอร์ จะใช้เป็นเครื่องมือ ในการทดสอบ ตรวจสอบ หรือแก้ไข ระบบไมโครคอมพิวเตอร์ ที่มีไมโครโปรเซสเซอร์ 8085 หรือ ไมโครคอนโทรลเลอร์ MCS-51 เป็นหน่วยประมวลผลกลาง ที่มีข้อผิดพลาดด้านซอฟต์แวร์(Software) หรือ ฮาร์ดแวร์(Hardware) ทั้งนี้เนื่องจากอีมูเลเตอร์สามารถนำเอาข้อมูลต่าง ๆ ภายในระบบที่ต้องการตรวจสอบขึ้นมาตรวจสอบได้ เช่น ข้อมูลภายในรีจิสเตอร์(Register) ข้อมูลที่อยู่ในหน่วยความจำ เป็นต้น และเมื่อผู้ใช้สามารถตรวจสอบข้อมูลได้อย่างถูกต้องแล้ว การวิเคราะห์ปัญหาที่เกิดขึ้นก็จะทำได้อย่างรวดเร็วทำให้ระยะเวลาที่ใช้ในการแก้ปัญหาลดลง อีมูเลเตอร์ตัวนี้สามารถนำไปเสียบเข้ากับซ็อกเก็ต(Socket) ของไมโครโปรเซสเซอร์บนระบบที่ต้องการตรวจสอบ ซึ่งอีมูเลเตอร์จะทำหน้าที่แทนหน่วยประมวลผลกลางของระบบที่ต้องการตรวจสอบ ผลการทดลองการใช้งานปรากฏว่า อีมูเลเตอร์ที่สร้างขึ้นสามารถใช้งานได้

8085 AND MCS-51 EMULATOR

CHAMNAN CHALERMYUT
DUMRONGSAK CHAIYAKITTISOPHON

SINGTHONG PATANASETHANON ADVISOR
1994

ABSTRACT

8085 and MCS-51 emulator development are presented in this thesis. This emulator system is processed by personal computer. An emulator is used for digonoustic testing in microcomputer. System with using a 8085 microprocessor or a microcontroller MCS-51. An emulator can check failure on hardware and software in whole system. All result after a digonoustic test has been shown massage on windows. The emulator can plugged directly into the central processing unit socket of target system , and it can replace central processing unit of target system.

สารบัญ

สารบัญรูป

หน้า

สารบัญตาราง

หน้า

บทที่ 1

คำนำและวัตถุประสงค์

ในปัจจุบันมีการนำเอาระบบไมโครคอมพิวเตอร์แบบชิงเกิลชิพ (Single chip Microcomputor) หรือเรียกอีกอย่างหนึ่งตามลักษณะการใช้งาน คือ ไมโครคอนโทรลเลอร์ (Microcontroller) มาเป็นส่วนประกอบในการควบคุมระบบอย่างแพร่หลาย เช่น อุปกรณ์ควบคุมการทำงานอัตโนมัติในโรงงานอุตสาหกรรมซึ่งไมโครคอมพิวเตอร์แบบชิงเกิลชิพ หรือไมโครคอนโทรลเลอร์จะมีข้อดีกว่าระบบไมโครคอมพิวเตอร์ทั่วๆไป คือ อุปกรณ์สนับสนุนต่างๆ ที่ต้องใช้งานร่วมกับไมโครโปรเซสเซอร์จะอยู่ภายในชิพเดียวกัน ด้วยจุดเด่นข้อนี้ทำให้ไมโครคอมพิวเตอร์แบบชิงเกิลชิพ หรือไมโครคอนโทรลเลอร์ถูกนำมาใช้ในงานควบคุม ซึ่งจะทำให้ระบบควบคุม หรืออุปกรณ์ต่างๆ มีขนาดเล็กลง แต่จะมีการทำงานที่สลับซับซ้อนมากกว่า

จะพิจารณาเห็นว่าการใช้ไมโครคอมพิวเตอร์แบบชิงเกิลชิพ หรือ ไมโครคอนโทรลเลอร์ เป็นส่วนประกอบในการควบคุม เมื่อต้องการทดสอบ หรือ ต้องการตรวจสอบระบบและ อุปกรณ์เหล่านั้น การตรวจสอบทางฮาร์ดแวร์ จะทำได้โดยอาศัยเครื่องมือ เช่น ลอจิกโพรบ และ ออสซิลโลสโคป ตรวจหาจุดบกพร่อง แต่สำหรับการทดสอบ หรือตรวจสอบทางซอฟต์แวร์อาจจะต้องอาศัย ชิงเกิลบอร์ด (Single Board) โดยการเขียนโปรแกรมเพื่อทดสอบจะต้องเขียนโปรแกรมคำสั่งเป็นเลขฐาน 16 ทำให้เกิดความไม่สะดวกในการตรวจสอบ และหาจุดบกพร่องต่างๆ จึงเกิดแนวความคิดที่ว่า การนำเอาการเขียนโปรแกรม และการทดสอบมาทำบนเครื่องไมโครคอมพิวเตอร์ โดยการการติดตั้งอุปกรณ์เพิ่มภายนอกแล้วนำสัญญาณไปควบคุมระบบที่ต้องการทดสอบ หรือ ตรวจสอบ ซึ่งทำให้เกิดความสะดวกในการตรวจซ่อมมากขึ้น

ปริญญานิพนธ์นี้จะกล่าวถึง หลักการในการสร้างอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และ ไมโครคอนโทรลเลอร์ MCS-51 การใช้งาน การทดลองและบทสรุปผลและรวมไปถึงข้อจำกัดของอีมูเลเตอร์ที่สร้างขึ้น

ผู้จัดทำหวังเป็นอย่างยิ่งว่า ปริญญานิพนธ์ฉบับนี้คงเป็นประโยชน์แก่ผู้ที่สนใจศึกษาไมโครโปรเซสเซอร์และไมโครคอนโทรลเลอร์บ้าง หากปริญญานิพนธ์ฉบับนี้มีข้อบกพร่องประการใด คณะผู้จัดทำยินดีที่จะน้อมรับคำแนะนำเพื่อเป็นประโยชน์ต่อไป

1.1 วัตถุประสงค์

การทำปริญญานิพนธ์ฉบับนี้มีวัตถุประสงค์ที่สำคัญดังนี้ ประการแรกเพื่อสร้างอีมูเลเตอร์ที่ใช้ในการทดสอบ ตรวจสอบหาจุดบกพร่อง ในระบบควบคุมที่มีไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 เป็นหน่วยประมวลผลกลาง ประการที่สองเพื่อสำนวยความสะดวกในการที่จะศึกษาโครงสร้างการทำงาน และการเขียนโปรแกรม ของไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 โดยปริญญานิพนธ์ฉบับนี้เอื้อสำนวยต่อผู้ใช้ในการหรือพัฒนาระบบไมโครโปรเซสเซอร์ และ ไมโครคอนโทรลเลอร์ บนเครื่องไมโครคอมพิวเตอร์ และประการสุดท้ายเพื่อศึกษาถึงเทคนิคต่างๆในการเขียนโปรแกรมด้วยภาษาระดับสูงเพื่อใช้ในการควบคุมอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51

1.2 ความเป็นมา

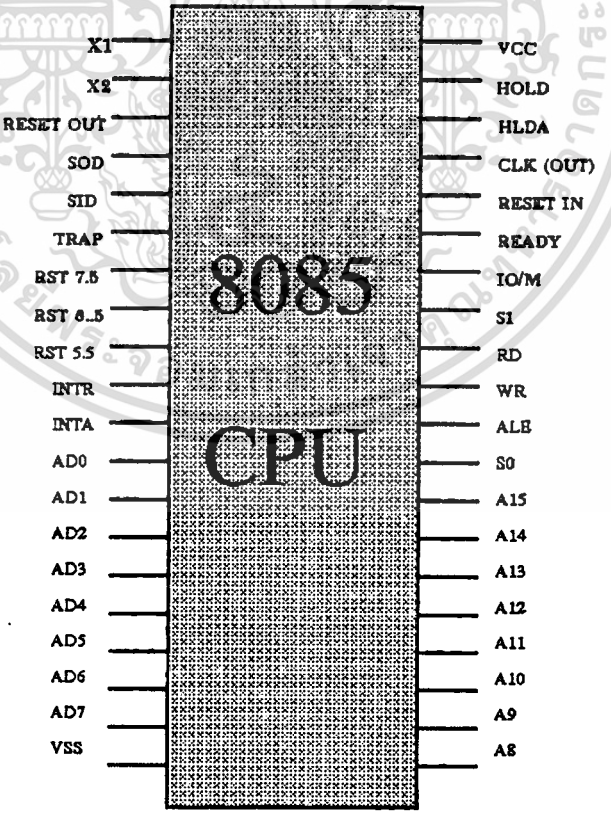
ในการตรวจสอบระบบที่ใช้ไมโครโปรเซสเซอร์ และ ไมโครคอนโทรลเลอร์เป็นส่วนควบคุม ถ้าหากระบบเกิดการทำงานผิดพลาดต้องการจะตรวจสอบ หรือ ต้องการที่จะทดสอบระบบนั้นๆ จะพิจารณาเห็นว่าความผิดพลาดอาจจะเกิดจากส่วนประกอบของระบบซึ่งจะแบ่งออกเป็น 2 ส่วน คือ ในส่วนของฮาร์ดแวร์ท่านอาจจะวิเคราะห์หาสาเหตุจากวงจรได้ แต่ถ้าเป็นความผิดพลาดที่เกิดจากซอฟท์แวร์ท่านไม่อาจจะทำการตรวจสอบได้เลย เนื่องจากระบบที่มีไมโครโปรเซสเซอร์ และ ไมโครคอนโทรลเลอร์เป็นส่วนควบคุม เมื่อทำการรันโปรแกรมจะเสมือนกับว่าตัวของมันเองตัดขาดจากภายนอกทำให้ไม่สามารถทราบได้เลยว่า ไมโครโปรเซสเซอร์ หรือ ไมโครคอนโทรลเลอร์ กำลังทำงานที่บันทึกไว้ในหน่วยความจำคำสั่งใดเกิดข้อผิดพลาดอะไร ด้วยเหตุนี้จึงนำเทคนิคของการอีมูเลท (Emulate) มาใช้งานโดยอาศัยหลักการ คือ ทำการสร้างไมโครคอนโทรลเลอร์หรือไมโครโปรเซสเซอร์ขึ้นมาอีกชุดหนึ่งเพื่อทำงานเลียนแบบไมโครโปรเซสเซอร์ หรือไมโครคอนโทรลเลอร์ที่เป็นส่วนควบคุมที่อยู่ภายในระบบที่ต้องการทดสอบ และขณะเดียวกันผู้ใช้งานก็สามารถตรวจสอบการทำงานของโปรแกรมได้ตลอดเวลา

บทที่ 2

การจัดการทางสถาปัตยกรรม

2.1 โครงสร้างทางสถาปัตยกรรมของไมโครโปรเซสเซอร์ 8085

ไมโครโปรเซสเซอร์ เบอร์ 8085 ถูกผลิตขึ้นโดยบริษัท INTEL ซึ่งถูกพัฒนามาจาก ไมโครโปรเซสเซอร์เบอร์ 8080 โดยโครงสร้างภายใน นั้นจะประกอบไปด้วย ตัวกำเนิดสัญญาณนาฬิกา,ระบบควบคุม อินเทอร์รัพต์เวคเตอร์ 5 อินพุต และมีโปรแกรมที่เกี่ยวกับการรับส่งข้อมูลอนุกรม ชุดคำสั่งของ 8085 ทั้งหมด แต่ได้เพิ่มคำสั่งใหม่อีก 2 คำสั่งเข้าไป การเรียกใช้รีจีสเตอร์ และการอ้างแอดเดรสจะยังคงเหมือนกับ 8080 และอุปกรณ์สนับสนุน (Chips Support) ที่ผลิตขึ้นโดยบริษัท INTEL สามารถนำมา อินเทอร์เฟส กับ 8085 ได้ง่าย



รูปที่ 2.1 แสดงลักษณะการจัดวางขาของไมโครโปรเซสเซอร์ เบอร์ 8085

2.1.1 ความหมายและลักษณะการจัดวางขาของไมโครโปรเซสเซอร์ เบอร์ 8085

ตำแหน่งขา	สัญญาณ	จุดประสงค์	ชนิด	คุณสมบัติอื่น ๆ
12-19	AD0-AD7	Address/data bus	Bidirectional	Three-state
21-28	A8-A15	Address bus	Output	Three-state
30	ALE	Address latch enable	Output	Three-state
32	$\overline{RD}$	Read control	Output	Three-state
31	$\overline{WR}$	Write control	Output	Three-state
34	IO/$\overline{M}$	I/O Memory switch	Output	Three-state
29,33	S0,S1	Bus state indicator	Output	
35	$\overline{READY}$	Wait state indicator	Input	
5	SID	Serial input data	Input	
4	SOD	serial output data	Output	
39	HOLD	Hold acknowledge	Input	
38	HLDA	Hold request	Output	
10	INTR	Interrupt request	Input	
6	TRAP	Nonmaskable interrupt	Input	
9	RST 5.5	Vectored interrupt	Input	
8	RST 6.5	Vectored interrupt	Input	
7	RST 7.6	Vectered interrupt	Input	
11	$\overline{INTA}$	Interrupt acknowledge	Output	
36	$\overline{RESET\ IN}$	System reset	Input	
3	$\overline{RESET\ OUT}$	Peripheral reset	Output	
1,2	X1,X2	Crystal contacts	Input	
37	CLK	System clock	Output	
40	Vcc	Power	Input	
20	Vss	Ground	Input	

ตารางที่ 2.1 ความหมายและลักษณะการจัดวางขาของไมโครโปรเซสเซอร์ เบอร์ 8085

2.1.2 รายละเอียดเกี่ยวกับสัญญานทาง Output ต่าง ๆ มีดังนี้

- **ALE (Address Latch Enable)**

ขาสัญญานนี้จะถูกส่งออกเป็นสถานะลอจิกสูง เมื่อไมโครโปรเซสเซอร์ ได้ทำการส่งแอดเดรส ออกมาที่ขา AD0-AD7แล้วสัญญาน ALE จะถูกใช้ให้เป็นตัวแลทช์สัญญาน AD0-AD7ได้โดยที่อุปกรณ์ภายนอกจะทำการแลทช์แอดเดรส เมื่อถึงขอบขาลงของสัญญาน ALE

- **A7-A15 (Address bus)**

คือ แอสเดรสขนาด 8 บิต ที่ต้องส่งออกไปเพื่อใช้งานการอ้างแอดเดรส ซึ่งจะไปรวมกับ AD0-AD7 เป็นแอดเดรสขนาด 16 บิต ส่วนแอดเดรสที่ใช้ติดต่อกับ อุปกรณ์อินพุต เอาท์พุตนั้น จะถูกจำกัดให้ใช้งานได้เพียง 00H-FFH ซึ่งก็จะใช้เพียงแค่สัญญาณ AD0-AD7 เท่านั้น

- **S0,S1**

จะมีสถานะลอจิกสูง หรือ ลอจิกต่ำ ซึ่งเป็นขาเอาท์พุตที่ไมโครโปรเซสเซอร์ส่งออกมาเพื่อใช้แสดงว่าคำสั่งที่เกิดขึ้นในระบบเป็นคำสั่งประเภทใดดังแสดงไว้ในตารางที่ 2.2

S0	S1	การใช้งาน
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

ตารางที่ 2.2 แสดงสถานะสัญญาณ S0,S1 ของไมโครโปรเซสเซอร์เบอร์ 8085

- **$\overline{\text{RD}}$ (Read)**

ขา RD นี้จะเป็นสัญญานเอาท์พุต ที่แอกทีฟที่สถานะลอจิกต่ำ ซึ่งมันจะแอกทีฟขณะที่ทำการอ่านข้อมูลในหน่วยความจำหรือจากการใช้คำสั่ง อ่านค่าจากพอร์ต และยังสามารถใช้เป็นสัญญาน Read Strobe memory และอุปกรณ์อินพุต เอาท์พุต ได้อีกด้วย

- **$\overline{\text{WR}}$ (Write)**

ขา WR จะแอกทีฟที่สถานะลอจิกต่ำ เมื่อไมโครโปรเซสเซอร์ทำคำสั่งเกี่ยวกับการเขียนออกไปบนบัสข้อมูล หรือส่งข้อมูลให้กับพอร์ต

- **IO/M** (Input-Output/Memory)

สัญญาณที่ส่งออกมาจากขานี้จะเป็นตัวบอกว่าคำสั่งที่กำลังทำงานอยู่นั้นเป็นคำสั่งที่ติดต่ออยู่กับหน่วยความจำหรือ อุปกรณ์อินพุตเอาท์พุตเมื่อสัญญาน IO/M มีสถานะเป็นลอจิกสูง ข้อมูลบนแอดเดรสบัส จะเป็นค่าของตำแหน่งที่ต้องการติดต่อกับอุปกรณ์อินพุตเอาท์พุต ซึ่งจะทำงานโดยตรง ในการอ่านหรือเขียนพอร์ต และเมื่อสัญญาณนี้มีสถานะเป็นลอจิกต่ำ ข้อมูลบนแอดเดรสบัส จะแสดงถึงตำแหน่งของหน่วยความจำที่ไมโครโปรเซสเซอร์ต้องการติดต่อด้วย ซึ่งจะเกิดการอ่านหรือเขียนโดยตรงกับหน่วยความจำ สัญญาณ IO/M นี้ จะใช้ร่วมกับสัญญาณควบคุมการอ่านหรือเขียน

- $\overline{\text{RESET OUT}}$ (Reset out)

สัญญาน RESET OUT นี้จะแอกทีฟที่สถานะลอจิกต่ำ เมื่อ ไมโครโปรเซสเซอร์อยู่ในสภาวะที่ถูกรีเซ็ต ซึ่งสามารถนำสัญญานจากขานี้ไปใช้ในการรีเซ็ต อุปกรณ์สนับสนุน (chips support) อื่น ๆ ที่ต่ออยู่ภายในระบบได้

- $\overline{\text{INTA}}$ (Interrupt Acknowledge)

สัญญาณ INTA จะแอกทีฟที่สถานะลอจิกต่ำ ก็ต่อเมื่อไมโครโปรเซสเซอร์ตอบรับการขออินเทอร์รัพต์

- **HLDA** (Hold Acknowledge)

เป็นสัญญานตอบรับการขอใช้บัสซึ่งจะมีสถานะลอจิกสูง และจะทำให้ไมโครโปรเซสเซอร์ลอยตัวออกจากระบบบัส

- **SOD** (Serial Output Data)

คือข้อมูลแบบอนุกรมที่ส่งออกไป ซึ่งถูกควบคุมจากโปรแกรมโดยค่าที่อยู่ในบิตที่ 7 ของแอกคิวมูเลเตอร์ จะถูกส่งออกไปยังขา SOD เมื่อมีการ set interrupt msaks คำสั่ง SIM จะถูกทำงาน ผู้ใช้โปรแกรมจำเป็นจะต้องกำหนดให้อยู่ในตำแหน่งที่ 7 และลองทดสอบการทำงานของคำสั่ง SIM

2.1.3 รายละเอียดเกี่ยวกับขาสัญญานทาง Input ต่าง ๆ มีดังนี้

- **TRAP**

สัญญานที่เข้ามาทางขา TRAP จะเป็นสัญญาน nonmaskable interrupt จะทำงานที่แอกทีฟที่สถานะลอจิกสูง ซึ่งจะเป็นการอินเทอร์รัพต์ที่มีลำดับความสำคัญสูงสุด และจะทำงานที่ขอบขาขึ้น ในขณะที่สัญญาณนี้แอกทีฟ ไมโครโปรเซสเซอร์จะทำงานตามคำสั่งปัจจุบัน

จนเสร็จ แล้วจะเก็บค่าตำแหน่งของแอดเดรสถัดไปลงในสแตค แล้วจะกระโดดไปยังตำแหน่ง 0024H ซึ่งผู้ออกแบบจะต้องเขียนโปรแกรมบริการการอินเทอร์รัพต์ เริ่มต้นไว้ที่ตำแหน่งนี้

- **RST X.5** (Restart 7.5,6.5,5.5)

เป็นสัญญานการอินเทอร์รัพต์ แบบ maskable จะทำงานเมื่อมีสถานะเป็นลอจิกสูง เมื่อมีสัญญาณการขออินเทอร์รัพต์ เข้ามาที่ขา RST X.5 ไมโครโปรเซสเซอร์จะทำงานตามคำสั่งที่กำลังทำอยู่ให้เสร็จเสียก่อน จากนั้นจึงจะทดสอบสถานะของ IFF (Interrupt enable flip-flop) ถ้า IFF ตัวนี้อยู่ในสภาวะไม่สนใจต่อการอินเทอร์รัพต์อยู่ ไมโครโปรเซสเซอร์จะไม่สนใจต่อการขออินเทอร์รัพต์นั้น ๆ ซึ่งมันจะทำงานตามคำสั่งถัดไป แต่ถ้า IFF ตัวนี้อยู่ในสภาวะที่สนใจต่อการอินเทอร์รัพต์อยู่ ไมโครโปรเซสเซอร์จะเก็บค่าตัวนับโปรแกรม (Program counter) ลงบนสแตค และจะกระโดดไปตามตำแหน่งแอดเดรส ที่กำหนดไว้ดังตารางที่ 2.3

สัญญาณ	ตำแหน่งที่กระโดด
RST7.5	003Ch
RST6.5	0034h
RST5.5	002Ch

ตารางที่ 2.3 แสดงตำแหน่งที่จะกระโดดไปทำงานเมื่อมีการขออินเทอร์รัพต์

สัญญานที่ขา RST 7.5 จะมีความสำคัญสูงสุด และRST 5.5 จะมีความสำคัญต่ำสุด การอินเทอร์รัพต์แบบนี้อาจทำได้จากโปรแกรมโดยใช้คำสั่ง SIM (set interrupt mask) สภาวะในการอินเทอร์รัพต์จะถูกเก็บไว้ในอินเทอร์รัพต์รีจิสเตอร์ จะถูกอ่านด้วยคำสั่ง RIM ดังตารางที่ 2.4

Bit	7	6	5	4	3	2	1	0
	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5

ตารางที่ 2.4 แสดงสภาวะในการอินเทอร์รัพต์รีจิสเตอร์ก่อนมีการขออินเทอร์รัพต์

บิตที่มี I นำหน้าจะเก็บสถานะการขออินเทอร์รัพต์ที่เข้ามาแต่ละอินพุต ส่วนบิตที่มี M นำหน้าจะเป็นการเซ็ตการขออินเทอร์รัพต์ จากโปรแกรมโดยใช้คำสั่ง SIM ซึ่งจะ

ทำการโหลดข้อมูลที่อยู่ในแอกคิวมูเลเตอร์ ลงในรีจีสเตอร์ บิตที่ 3 นั้นจะใช้เป็นบิตที่ควบคุมการสนใจ หรือไม่สนใจการอินเทอร์รัพต์ ด้วยคำสั่ง DI และ EI ตามลำดับ เมื่อมีการขออินเทอร์รัพต์ ค่าที่บรรจุอยู่ในอินเทอร์รัพต์รีจีสเตอร์จะเป็นดังตารางที่ 2.5

Bit	7	6	5	4	3	2	1	0
	SID	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5

ตารางที่ 2.5 แสดงสภาวะในการอินเทอร์รัพต์รีจีสเตอร์เมื่อมีการอินเทอร์รัพต์

SOE คือ การควบคุมการ enable ข้อมูลที่จะถูกส่งออกมาที่ขา SID

X คือ bit ที่ไม่ได้ใช้งาน

MSE คือ เงื่อนไขของ master interrupt enable.

- **INTR** (Interrupt Request)

เป็นสัญญาณอินเทอร์รัพต์ แบบ maskable ซึ่งทำงานที่สถานะลอจิกสูง เมื่ออุปกรณ์ภายนอกส่งสัญญาณมาที่ขานี้ ไมโครโปรเซสเซอร์จะทำคำสั่งปัจจุบัน จนเสร็จก่อน หลังจากนั้น จะทำการตรวจสอบสถานะของ IFF ถ้า IFF นี้ถูกรีเซ็ตอยู่ สัญญาณอินเทอร์รัพต์ก็จะไม่ได้รับการตอบสนองและไมโครโปรเซสเซอร์จะทำคำสั่งถัดไป แต่ถ้า IFF ถูกรีเซ็ตอยู่ ไมโครโปรเซสเซอร์ ก็จะตอบสนองต่อการขออินเทอร์รัพต์ โดยที่ไมโครโปรเซสเซอร์จะคิดว่าคำสั่งถัดไปเป็นออปโค้ดจากภายนอก คำสั่งแรกจะถูกทำงานหลังจากการอินเทอร์รัพต์ได้รับการตอบสนอง ไมโครโปรเซสเซอร์จะทำการเก็บค่าตัวนับโปรแกรมลงในสแตค จากนั้นจะทำงานตามโปรแกรมบริการการอินเทอร์รัพต์ และเมื่อพบคำสั่ง RST ก็จะกลับมาที่ตำแหน่งหลังสุดที่ถูกเก็บไว้ในสแตคก่อนที่จะตอบสนองการอินเทอร์รัพต์ แล้วจะทำคำสั่งถัดไป IFF สามารถควบคุมได้จากโปรแกรมด้วยการ EI (enable interrupt) และ DI (disable interrupt) สัญญาณอินเทอร์รัพต์ที่เข้ามาที่ขานี้ จะมีความสำคัญต่ำกว่าสัญญาณอินเทอร์รัพต์อื่น ๆ ทั้งหมด

- $\overline{\text{READY}}$ (Ready)

เมื่อสัญญาน READY นี้ ถ้าแอกทีฟที่สถานะต่ำขึ้นมา ไมโครโปรเซสเซอร์จะสร้างสัญญาน Wait states จนกระทั่ง READY กลับเป็นสสถานะลอจิกสูง ซึ่งจะใช้ในกรณีที่อุปกรณ์อื่นทำงานช้ากว่าระบบ

- **HOLD** (Hold)

เมื่อขาสัญญานนี้ มีสถานะเป็นลอจิกสูง จะทำให้ไมโครโปรเซสเซอร์หยุดการทำงานเมื่อทำคำสั่งปัจจุบันจนครบแมชชีนไซเคิลแล้ว และจะลอยตัวออกจากระบบ เพื่อให้อุปกรณ์ภายนอกเข้ามาใช้ แอดเดรสและดาต้าบัสได้ หรืออาจจะนำสัญญาณนี้มาใช้ในการทำขบวนการ DMA หรือการรีเฟรสไดนามิคแรมได้

- **RESET IN** (Reset in)

เมื่อสัญญาน RESET IN นี้จะรับสัญญาณที่มีสถานะเป็นลอจิกต่ำเพื่อใช้ในการรีเซ็ต การทำงานของระบบและตัวนับโปรแกรมจะถูกโหลดให้เป็นตำแหน่งที่ 0000H ดังนั้นผู้ใช้งานควรเขียนโปรแกรมให้เริ่มทำงานที่แอดเดรส 0000H ขณะเดียวกันสัญญาณ INTA และ HLDA,IFF จะถูกรีเซ็ตด้วย

- SID (Serial Input Data)

เป็นขาสัญญาณอินพุต แบบอนุกรม ซึ่งถูกควบคุมด้วยโปรแกรมเมื่อมีการอ่านสัญญาณการอินเทอร์รัพต์ คำสั่ง RIM จะถูกทำงาน จะมีข้อมูลเพียงบิตเดียวที่ขา SID และจะถูกส่งไปยังบิตที่ 7 ของแอกคิวมูเลเตอร์ มันสามารถที่จะถูกส่งไปยังตำแหน่งหน่วยความจำที่ผู้ใช้เป็นผู้กำหนด

- Vcc

ขาอินพุตขานี้ อีมูเลเตอร์จะใช้เป็นตัวตรวจสอบว่าระบบภายนอกจ่ายไฟ (+5V) มาที่ขานี้หรือไม่ ถ้ามีโปรแกรม EMU8085 จะทำงาน แต่ถ้าไม่มีก็จะไม่สามารถเรียกโปรแกรม EMU8085 มาทำงานได้หรืออาจจะไม่สามารถรันโปรแกรมของ CPU-8085 ได้

- Vss

เป็นขาอินพุตที่ต้องต่อกับกราวด์ของระบบภายนอกกับเครื่องไมโครคอมพิวเตอร์

2.2 โครงสร้างทางสถาปัตยกรรมของไมโครคอนโทรลเลอร์ MCS-51

ลักษณะทั่วๆ ไปของ MCS-51 จะประกอบด้วย

1.สร้างโดยใช้ HMOS และ CHMOS เทคโนโลยีและการทำงานด้วยแหล่งจ่ายไฟขนาด 5 V. เพียงแหล่งเดียว

2. ซีพียูมีขนาด 8บิต

3 มีวงจรออสซิลเลเตอร์ และวงจรนาฬิกาบนชิพ

4. ชุดแบงก์ (BANK) รีจิสเตอร์มี 4 ชุด แต่ละชุดมีรีจิสเตอร์ 8 ตัวทำงาน

5. มีตัวจับเวลา/ตัวนับขนาด 16 บิต 2ชุด และสำหรับเบอร์ 8032/8052 มี 3 ชุด

6. มีพอร์ตอินพุตเอาท์พุตแบบขนาน 2 ทิศทางจำนวน 4 พอร์ต พอร์ตละ 8 บิต รวมทั้งหมดเป็น 32 เส้นแต่จะเหลือเพียง 16 เส้นสำหรับเบอร์ 8031 อีก 16 เส้นใช้ในการเข้าถึงทางแอดเดรสและข้อมูล
7. พอร์ตแบบอนุกรมสามารถที่จะใช้โปรแกรมการรับส่งแบบฟูลดูเพล็กซ์ ที่ความเร็วสูง
8. หนึ่งวัฏจักรคำสั่งจะใช้เวลา 1 ไมโครวินาที ด้วยการใช้คริสตอล 12 เมกกะเฮิรตซ์
9. แอดเดรสข้อมูลภายนอกได้ 64 กิโลไบต์
10. แอดเดรสโปรแกรมภายนอกได้ 64 กิโลไบต์
11. สามารถกำหนดเลขที่อยู่ข้อมูลขนาดไบต์หรือบิตได้โดยตรง
12. มีซอร์ฟแวร์แฟล็กสำหรับผู้ใช้ที่จะกำหนดเองได้ถึง 128 ตำแหน่งบิต
13. โครงสร้างอินเทอร์รัปต์ทำได้ 5 แหล่งและ 6 แหล่งสำหรับ 8032/8052 พร้อมด้วยการจัดไพรโอริตี้ (Priority) ได้ 2 ระดับ
14. ตัวโปรเซสเซอร์สามารถใช้งานแบบบูลีน (Boolean) ได้สำหรับการใช้งานควบคุม
15. มีคำสั่งคูณและหารทางฮาร์ดแวร์ทำได้ภายใน 4 ไมโครวินาที
16. ตัวเลขทางคณิตศาสตร์ ใช้ได้ทั้งแบบไบนารี และเดซิมัล
17. การใช้พื้นที่สแต็กสำหรับโปรแกรมย่อยต่างๆ ทำได้กว้างขึ้น

ไมโครคอนโทรลเลอร์ ตระกูล MCS-51 จะมีทั้งแบบมี หน่วยความจำโปรแกรมภายในตัว หรือไม่มีบนชิพเดียวกันและจะมีตำแหน่งที่ขาเหมือนกัน ตารางที่ 2.6 แสดงถึงตารางรายละเอียดของเบอร์ต่างๆ ในตระกูล MCS-51 ที่มีจำหน่ายในท้องตลาด

8751H อยู่ในกลุ่มรุ่นเดียวกับ 8051AH ที่เราสามารถโปรแกรมได้ด้วยระบบไฟและสามารถลบโปรแกรมออกได้ด้วยแสงอุลตร้าไวโอเลต นอกเหนือจากไอซีที่แสดงในตารางที่ 2.6 ที่ใช้เทคโนโลยี HMOS แล้วยังมีตระกูลอื่นที่ใช้เทคโนโลยี CHMOS ที่ประหยัดพลังงานได้มากกว่า 4 เท่าของ HMOS ที่มีจำหน่ายขณะนี้คือเบอร์ 80C51,80C31 และ 87C51

เบอร์	หน่วยความจำภายใน		ตัวนับ/ตัวนับจำนวน	อินเตอร์รัพต์
	โปรแกรม	ข้อมูล		
8052 AH	8K X 8 ROM	256 X 8 RAM	3 X 16 BIT	6
8051 AH	4K X 8 ROM	128 X 8 RAM	2 X 16 BIT	5
8051	4K X 8 ROM	128 X 8 RAM	2 X 16 BIT	5
8032 AH	ไม่มี ROM	256 X 8 RAM	2 X 16 BIT	5
8031 AH	ไม่มี ROM	128 X 8 RAM	2 X 16 BIT	5
8031	ไม่มี ROM	128 X 8 RAM	2 X 16 BIT	5
8751 H	4K X 8 EPROM	128 X 8 RAM	2 X 16 BIT	5
8751 H-12	4K X 8 EPROM	128 X 8 RAM	2 X 16 BIT	5

ตารางที่ 2.6 แสดงรายละเอียดของไมโครคอนโทรลเลอร์ ตระกูล MCS-51

2.2.1 การจัดขาลักษณะภายนอกของ MCS-51

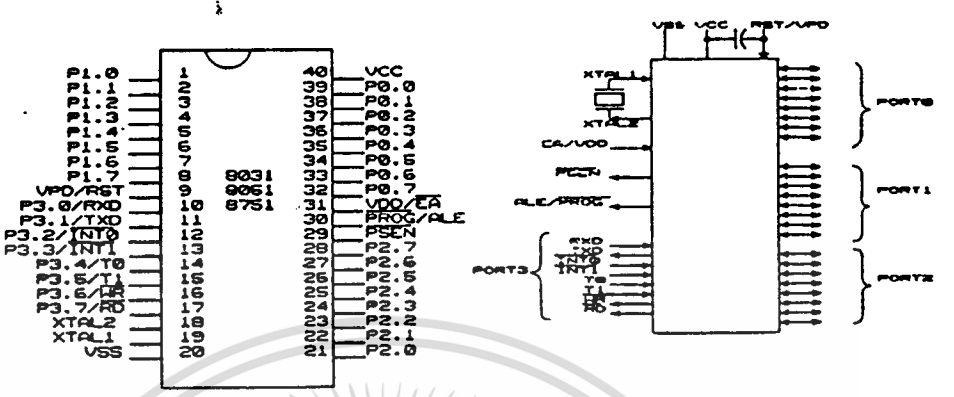
รูปที่ 2.1 แสดงการจัดขาตามลักษณะภายนอกของชิพ MCS-51 ซึ่งจะมีการแบ่งกลุ่มการจัดการขาตามสถาปัตยกรรมของ MCS-51 อยู่ 4 กลุ่มคือ

1. กลุ่มขารับแหล่งจ่ายไฟฟ้า และสัญญาณนาฬิกา
2. กลุ่มขาแอดเดรสและข้อมูล
3. กลุ่มขาควบคุม
4. กลุ่มขาพอร์ตแบบขนานและอนุกรม

ขาบางขาจะมีหน้าที่ได้สองหน้าที่ขึ้นอยู่กับการติดตั้งด้วยซอร์ฟแวร์หรือฮาร์ดแวร์ ซึ่งมีรายละเอียดดังนี้คือ

- ขา Vss (ขา 20) เป็นขาสำหรับต่อลงดิน
- ขา Vcc (ขา 40) เป็นขาที่ต่อแรงดันไฟกระแสตรงขนาด 5 V. และใช้สำหรับการโปรแกรมแบบ Open Drain Bidirectional สามารถที่จะรับโหลดทีทีแอลได้ 8 ตัว การเขียนค่า "1" ไปที่พอร์ตนี้ จะเป็นการปล่อยลอย (Float) ขาของพอร์ตนี้ ทำให้มันทำงานเป็นอินพุตมีสถานะอิมพีแดนซ์สูง ในการให้พอร์ตนี้บริการแบบไอโอ พอร์ต 0 จะทำงานเป็นมัลติเพล็กซ์ด้วยสัญญาณแอดเดรสไบต์ต่ำกับบัสข้อมูล สำหรับการใช้งานด้านหน่วยความจำภายนอก ในการใช้งานแบบนี้จะใช้ลักษณะภายในเป็นตัวพูลอัพ พอร์ต 0 ยังใช้งาน เป็นตัวส่งข้อมูลออกทางพอร์ตนี้ เมื่อใช้บริการทางด้านการตรวจสอบโปรแกรม ROM

ภายใน และการโปรแกรมตัว EPROM ภายในถ้าใช้งานในลักษณะนี้การพูลอัพภายนอกจะต้องต่อด้วยค่า 10 กิโลโอห์ม



รูปที่ 2.2 (A) ลักษณะการจัดวางขาของ MCS-51 (B) สัญญลักษณ์ทางตรรกของ MCS-51

- ขา Port 1 (P1.0 - P1.7) (ขา 1 - 8) เป็นพอร์ตไอโอ 8 บิตแบบ Open Drian Bidirectional พร้อมด้วยพูลอัพภายใน ถ้าเป็นพอร์ตเอาต์พุต บัฟเฟอร์สามารถขับโหลดทีทีแอลตระกูลแอลเอสไอได้ 4 ตัว พอร์ต 1 เมื่อถูกเขียนค่า "1" ด้วยโปรแกรมมันจะมีสถานะสูงด้วยการพูลอัพภายใน การให้สถานะเช่นนี้จะเป็นการกำหนดใช้งานพอร์ตนี้ให้เป็น อินพุต ขณะที่พอร์ต 1 เป็นอินพุตการให้สัญญาณต่ำลงจะเป็นการจ่ายกระแสออกเนื่องจากการพูลอัพภายใน ในเบอร์ 8052 ขา P1.0 และ P1.7 จะใช้งานเป็น T2 และ T2EX โดยขา T2 จะทำหน้าที่รับสัญญาณจากภายนอกให้ตัวตั้งเวลา 2 ทำงาน และขา T2EX จะเป็นอินพุตผ่านเข้าตัวตั้งเวลา 2 ถูกกระตุ้นให้ทำงานแบบปกติตามโปรแกรมที่ตั้งไว้ หรือแบบแคปเจอร์ (Capture)
- ขา Port 2 (P2.0 - P2.7) (ขา 21 - 28) เป็นพอร์ตไอโอ 8 บิตแบบ Open Drian Bidirectional ด้วยการพูลอัพภายในพอร์ต 2 ที่ทำหน้าที่เป็นบัฟเฟอร์เอาท์พุตสามารถจ่ายโหลดทีทีแอลตระกูลแอลเอสไอได้ 4 ตัวพอร์ตจะถูกใช้งานเป็นตัวส่งแอดเดรสไบต์สูงด้วย เมื่อใช้งานร่วมกับหน่วยความจำภายนอก เพื่อให้สามารถอ้างแอดเดรส ได้ถึง 16 บิต ด้วยการใช้งานแบบนี้มันจะมีพูลอัพภายในที่ช่วยในการส่งค่า "1" ได้ระดับที่แน่นอน นอกจากการใช้งานสำหรับแอดเดรสอันดับสูงใช้เป็นขาควบคุมในการใช้งานตรวจสอบ และเขียนโปรแกรมเบอร์ 8751 และตรวจ สอบโปรแกรมภายใน 8051

- ขา Port (P3.0 - P3.7) (ขา 10 - 17) เป็นพอร์ตไอโอ 8 บิตแบบพูลอัพภายใน นอกจากทำเป็นพอร์ตไอโอ ที่สามารถรับโหลดทีทีแอลพวกตระกูลแอลเอสไอได้ 4 ตัวแล้ว ยังใช้งานเป็นพิเศษสำหรับตระกูล MCS-51 ดังตารางที่ 2.7

ชาพอร์ต	ขา	การทำงานตามฟังก์ชันพิเศษ
P3.0	10	RXD พอร์ตอนุกรมอินพุต
P3.1	11	TXD พอร์ตอนุกรมเอาท์พุต
P3.2	12	INT0 อินเทอร์รัพท์ภายนอกตัวที่ 1
P3.3	13	INT1 อินเทอร์รัพท์ภายนอกตัวที่ 2
P3.4	14	T0 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลาและตัวนับ 0
P3.5	15	T1 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลาและตัวนับ 1
P3.6	16	WR สัญญาณควบคุมการเขียน
P3.7	17	RD สัญญาณควบคุมการอ่าน

ตารางที่ 2.7 รายละเอียดการทำงานของขา P3.0-P3.7

การที่จะให้ทำงานตามฟังก์ชันข้างบน จะต้องเริ่มโปรแกรมด้วยการส่งค่า "1" ไปแลทช์ไว้ก่อนที่จะให้ทำงานตามฟังก์ชันข้างบน

- ขา RST (ขา 9) ต้องคงสถานะค่าสูงเป็นเวลาประมาณอย่างน้อย 2 วัฏจักรระหว่างที่ออสซิลเลเตอร์ทำงานที่ต้องการเซ็ตทั้งระบบงาน โดยจะต่อรีจิสเตอร์พูลดาวน์ (8.2 กิโลโอห์ม) จากขา RST ไปลงดินและเพื่อให้ตัวชิพรีเซ็ตได้โดยอัตโนมัติ ขณะเปิดจะใช้คาปาซิเตอร์ (10 ไมโครฟารัด) ต่อคร่อมระหว่างขา RST กับขา Vcc
- ขา ALE/PROG (ขา 30) เป็นขาแอดเดรสแลทช์อีนาเบิ้ลด้วยการส่งพัลส์ออก ไปใช้สำหรับแลทช์ค่าแอดเดรสไบต์ต่ำจากพอร์ต 0 ในระหว่างการเข้าถึงข้อมูล จากหน่วยความจำภายใน ALE จะถูกส่งสัญญารออกมาในอัตราความเร็วคงที่ ที่ 1/8 ของความถี่ออสซิลเลเตอร์ตลอดเวลา แม้ว่าจะไม่มีการเข้าถึงข้อมูลจากภายใน ดังนั้นจึงสามารถที่จะใช้สัญญาณจากขานี้เป็นตัวตั้งเวลาภายนอก หรือเป็นความถี่สัญญาณนาฬิกา แต่อย่างไรก็ตามความถี่สัญญาณนี้จะลดความถี่ช้าลงไปเท่าหนึ่ง ระหว่างการทำงานแบบการเข้าถึงของหน่วยความจำข้อมูลภาย

นอก ขานี้ยังใช้เป็นสัญญาณพัลส์เข้าสำหรับการควบคุมการโปรแกรม EPROM ภายในชิพ

- ขา PSEN (ขา 29) Program Storage Enable เป็นสโตรบอ่านข้อมูลจากโปรแกรมหน่วยความจำภายนอก เมื่อชิพทำงานด้วยโปรแกรมภายนอก ขา PSEN จะสร้างสโตรบต่ำสองครั้งภายในแต่ละวัฏจักรแมชชีน สัญญาณจะมีสถานะสูง หรือพัลส์ต่ำทั้งสองลูกจะหายไป เมื่อทำงานในช่วงการอ่านหรือเขียนข้อมูลจากหน่วยความจำข้อมูลภายนอก และ PSEN จะไม่มีพัลส์ส่งออก ถ้าชิพทำงานด้วยโปรแกรมหน่วยความจำภายใน
- ขา EA/Vdd (ขา 31) มีสถานะสูง ตัวซีพียูในชิพจะทำงานตามโปรแกรมที่อยู่ในหน่วยความจำภายใน (โดยที่โปรแกรมจะต้องไม่ยาวกว่า 4 กิโลไบต์สำหรับเบอร์ 8051 AH และ 8 กิโลไบต์สำหรับเบอร์ 8052 AH) การทำให้ EA มีสถานะต่ำ จะเป็นการควบคุมให้ซีพียูทำงานตามโปรแกรมหน่วยความจำภายนอก ซึ่งขยายโปรแกรมได้ยาวถึง 64 กิโลไบต์ ในตัว 8031 AH และ 8032 AH ขา EA จะต้องต่อลงดินเช่นกันแม้ว่าจะไม่มี ROM อยู่ภายในก็ตาม ในตัว 8751 AH จะใช้ขานี้จ่ายแรงดันขนาด 21 V. ขณะทำการเขียนโปรแกรมเข้า EPROM ของชิพ
- ขา XTAL1 (ขา 19) ใช้เป็นตัวอินพุตเข้าสู่ออสซิลเลเตอร์ขยายแบบ Invert
- ขา XTAL2 (ขา 18) ใช้เป็นเอาท์พุตจากตัวออสซิลเลเตอร์ขยายแบบ Invert

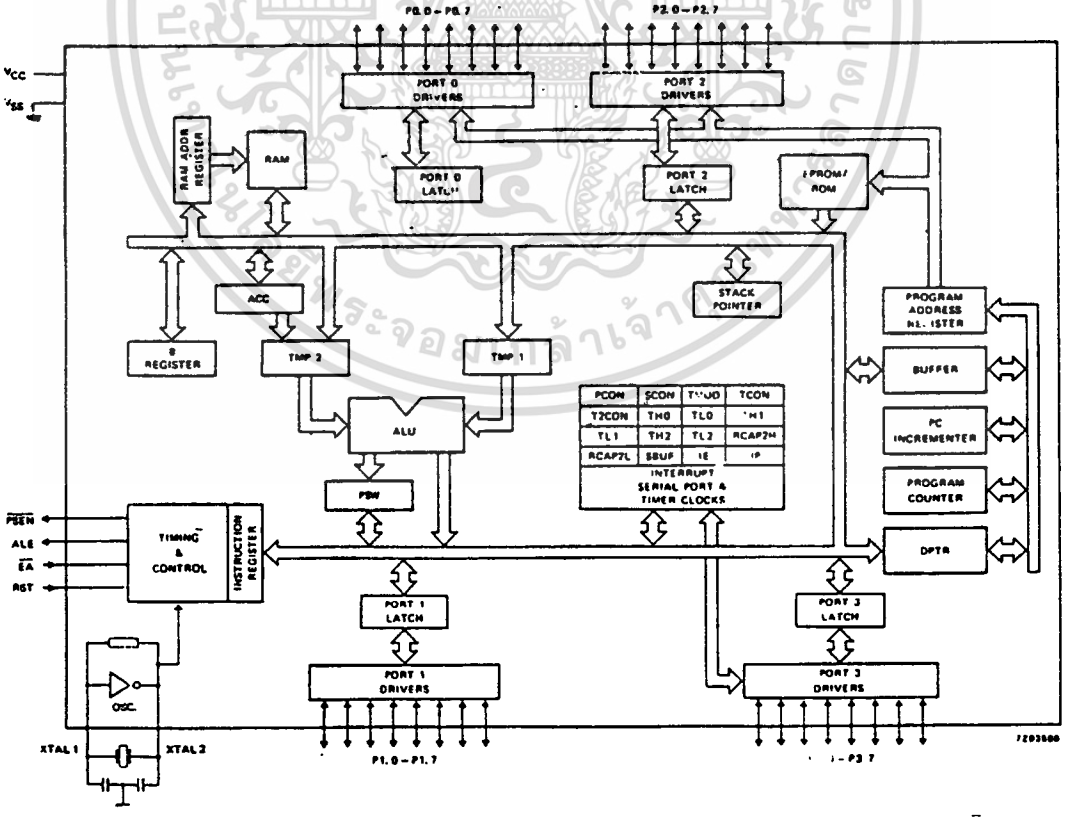
ตามตาราง 2.6 MCS-51 ทั้งสามกลุ่มคือ กลุ่มที่มี ROM และไม่มี ROM และพวก EPROM จะมีที่ขาใช้งานเหมือนกันหมดยกเว้นขา 1 จะใช้งานเป็น T2 และขา 2 เป็น T2EX ในเบอร์ 8032/ 8052 ตลอดถึงจังหวะเวลา (Timing Diagram) และคุณสมบัติทางไฟฟ้าทั้งสามจะแตกต่างกันเฉพาะการโปรแกรมบนชิพ MCS-51 เท่านั้นซึ่งแต่ละแบบจัดไปตามความต้องการของผู้ใช้ เช่น 8751 AH จะมีหน่วยความจำ EPROM ขนาด 4 กิโลไบต์เหมาะสำหรับการพัฒนาเครื่องต้นแบบและการผลิตอุปกรณ์ที่มีจำนวนจำกัด เมื่อต้องการจะเขียนโปรแกรมเข้า EPROM จะมีตัวเขียนโปรแกรมพิเศษสำหรับเขียนโปรแกรมที่ผู้ออกแบบเขียนขึ้นมาถ้าโปรแกรมมีส่วนผิดพลาดที่ต้องการจะแก้ไขได้โดยการนำตัว 8751 AH นี้ไปล้างโปรแกรมเดิมออกด้วยแสงอุลตร้าไวโอเลตและอัดโปรแกรมที่ได้แก้ไขแล้วเข้าไปใหม่ ทำเช่นนี้จนกระทั่งได้โปรแกรมสมบูรณ์ และเมื่อต้องการผลิตจำนวนมากก็สามารถที่จะใช้ MCS-51 เบอร์ 8051 ที่มี 4 กิโลไบต์ของ ROM ซึ่งจะถูกอัดข้อมูลโปรแกรมตามความต้องการของผู้ออกแบบ โดย

โรงงานผู้ผลิตชิพเบอร์นี้ การผลิตลักษณะนี้จะถูกกว่าการใช้เบอร์ 8751 แต่โปรแกรมภายในจะไม่สามารถลบ และโปรแกรมใหม่ได้หลังจากการผลิตไปแล้ว

ส่วนเบอร์ 8031 จะไม่มีหน่วยความจำของโปรแกรมบนชิพ แต่อาจต่อหน่วยความจำโปรแกรมจากภายนอกด้วย ROM EPROM หรือ PROM ได้ถึง 64 กิโลไบต์ดังนั้น 8031 จึงเหมาะสำหรับการใช้งานที่โปรแกรมมีขนาดใหญ่กว่า 4 กิโลไบต์ และสำหรับผู้ออกแบบที่ต้องการแยกส่วนของโปรแกรมออกจากชิพ

2.2.2 การจัดการทางสถาปัตยกรรม

รูปที่ 2.3 เป็นบล็อกไดอะแกรมที่บ่งตามลักษณะงานทางด้านสถาปัตยกรรมภายในของ MCS-51 โดยชิงเกิลชิพ แต่ละตัวของตระกูลนี้ จะประกอบด้วยหน่วยศูนย์กลางการประมวลหน่วยความจำสองชนิดคือ แบบ RAM และ ROM หรือ EPROM พอร์ต อินพุต เอาต์พุต โหมดรีจิสเตอร์สถานะข้อมูล ส่วนวงจรตรรกในการสุ่ม ที่จำเป็นสำหรับตัวแปรของฟังก์ชันการต่อพ่วง ส่วนต่างๆ ที่กล่าวนี้จะติดต่อกันด้วยบัสขนาด 8 บิต และจะมีบัฟเฟอร์สำหรับการติดต่อกับภายนอกผ่านพอร์ตไอโอ เมื่อต้องการขยายหน่วยความจำหรือพอร์ตไอโอ



รูปที่ 2.3 โครงสร้างสถาปัตยกรรมภายในของ MCS-51

- **แอกคิวมูเลเตอร์ (Accumulator : Acc)**

MCS-51 ก็เช่นเดียวกับ MCS-48 ที่ใช้แอกคิวมูเลเตอร์ที่มีขนาด 8 บิต เป็นแอกคิวมูเลเตอร์หลัก คำสั่งส่วนใหญ่จะอ้างถึงตัวรีจิสเตอร์นี้โดยถือค่าภายในเป็นค่าตัวตั้งและรับค่าผลลัพธ์ที่ได้จากคำสั่งทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร เข้ามาเก็บไว้ในตัวแอกคิวมูเลเตอร์ ยังสามารถใช้ตัวแหล่งกระทำ หรือถูกกระทำในการทำงานทางตรรก และใช้เป็นตัวกลางในการถ่ายเทข้อมูลในการติดต่อกับอุปกรณ์ภายนอกไอโอและหน่วยความจำภายนอกรวมถึงการตรวจสอบข้อมูล

- **รีจิสเตอร์ B**

เป็นรีจิสเตอร์พิเศษที่ใช้งานสำหรับคำสั่งของการคูณและหาร โดยใช้เป็นที่เก็บตัวคูณหรือหาร และเป็นที่เก็บผลลัพธ์ตัวที่สองหลังการคูณและเศษของการหาร

- **รีจิสเตอร์คำแสดงสถานะโปรแกรม (Program Status Word : PSW)**

รีจิสเตอร์ PSW เป็นรีจิสเตอร์ที่แสดงผลที่ได้หลังจากการใช้คำสั่งต่างๆ และใช้เป็นตัวเลือกกลุ่มการทำงานของรีจิสเตอร์กลุ่มต่างๆ

- **ตัวชี้สแตค (Stack Pointer : SP)**

MCS-51 จะใช้หน่วยความจำข้อมูลภายใน เป็นบริเวณสแตคทางฮาร์ดแวร์สำหรับ การเชื่อมโปรแกรมหลักสแตคการผ่านพารามิเตอร์ระหว่างงาน ในแต่ละส่วนโปรแกรม และสแตคเก็บตัวแปรข้อมูลชั่วคราว หรือสแตคการเก็บสถานะระหว่างการบริการงานอินเทอร์รัพต์ไว้ภายในชิพโดยที่ตัวชี้สแตคจะมีขนาด 8 บิต จะเพิ่มค่าอัตโนมัติก่อนที่ข้อมูลจะนำมาเก็บในหน่วยความจำระหว่างการใช้คำสั่ง PUSH และ CALL และจะลดค่าของตัวชี้สแตค ลงจากที่ได้ถ่ายเทข้อมูลออกไปแล้วในคำสั่ง POP หรือ RETURN โดยทฤษฎีทางสถาปัตยกรรม MCS-51 สามารถใช้สแตคให้มีเนื้อที่ถึง 128 ไบต์ แต่ในทางปฏิบัติสำหรับโปรแกรมทั่วไปจะใช้น้อยกว่านี้ตัวชี้สแตค จะเริ่มที่ตำแหน่ง 07H ดังนั้นสแต็กจะเริ่มบรรจุข้อมูลที่ตำแหน่ง 08H MCS-51 สามารถเปลี่ยนแปลงค่าในตัวชี้สแตคได้ ซึ่งจะเป็นการเปลี่ยนตำแหน่งสแตคไปยังที่ใดๆ ของหน่วยความจำข้อมูลภายในชิพ

- **ตัวชี้ข้อมูล (Data Pointer : DPTR)**

ตัวชี้ข้อมูล เป็นรีจิสเตอร์ขนาด 16 บิตที่ประกอบด้วยไบต์สูง (DPH) และไบต์ต่ำ (DPL) ที่เราสามารถเลือกแบ่งออกเป็นรีจิสเตอร์ 8 บิตสองตัวที่ใช้ได้อย่างอิสระ หรือจะใช้ร่วมกันทั้ง 16 บิตก็ได้ ในการเพิ่มค่า หรือ ลดค่าเพื่อประโยชน์ในการใช้เป็นฐานของเลขที่อยู่ในรีจิสเตอร์ในการกระโดดโดยทางอ้อมในการใช้คำสั่งเกี่ยวกับตารางข้อมูลและชี้ตำแหน่งของหน่วยความจำภายนอก

- **พอร์ต 0 ถึง 3**

รีจิสเตอร์ P0, P1, P2 และ P3 **ของกลุ่มรีจิสเตอร์ฟังก์ชั่นพิเศษ** (Special Function Register : SFR) **จะเป็นตัวรีดิสเตอร์ที่แลทช์ค่าของพอร์ต** 0,1,2 และ 3**ตามลำดับในขณะใช้งาน**

- **รีจิสเตอร์ CAPTURE**

ไอซีเบอร์ 8035/8052 **จะมีคู่รีจิสเตอร์** (RCAP2H,RCAP2L) **เพิ่มเติมเป็นรีจิส-เตอร์เค็ปเจอร์ สำหรับตัวตั้งเวลาหมายเลข** 2 **ในโหมดการใช้งานของรีจิสเตอร์ตัวนี้จะรับการเปลี่ยนแปลงที่เข้ามาที่ขา** T2EX **ตัว** TH2 **และ** TL2 **จะลอกข้อมูลเข้าไปในรีจิสเตอร์คู่** RCAP2H **และ** RCAP2L **โดยการใช้ตัวตั้งเวลา จะมีโหมดการบรรจุอัตโนมัติขนาด** 16 **บิตสำหรับการใช้ตัวตั้งเวลา/ตัวนับ** 2

- **บัฟเฟอร์ข้อมูลอนุกรม (Serial Data Buffer : SBUF)**

บัฟเฟอร์ข้อมูลอนุกรมแบ่งออกเป็นรีจิสเตอร์สองตัว ตัวหนึ่งเป็นบัฟเฟอร์การส่งและอีกตัวหนึ่งเป็นบัฟเฟอร์การรับ เมื่อข้อมูลถ่ายเทเข้า SBUF **มันจะถ่ายเข้าบัฟเฟอร์การส่ง ซึ่งเป็นตัวจัดการส่งข้อมูลอนุกรม วิธีการเคลื่อนย้ายเข้า** SBUF **ขึ้นอยู่กับการกำหนดคิดตั้งโปรแกรม การส่งข้อมูลย้ายออกจาก** SBUF **จะเป็นการรับข้อมูลจากบัฟเฟอร์ตัวรับ**

- **รีจิสเตอร์ควบคุม (Control Register)**

กลุ่ม SFR **ที่เป็น** IP,IE,TMOD,TCON,T2CON,SCON **และ** PCON **จะประกอบด้วยบิตที่ใช้ในการควบคุมและแสดงสถานะของการใช้งานในระบบอินเตอร์รัปต์ตัวตั้งเวลา/ตัวนับและพอร์ตอนุกรม**

2.2.3 การจัดการหน่วยความจำ

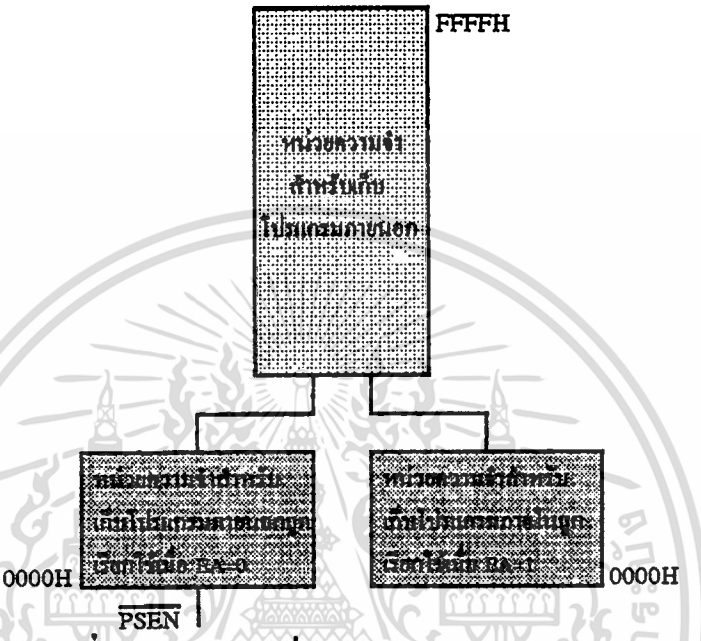
MCS-51 แบ่งตามพื้นฐานหน่วยความจำของการกำหนดเลขที่อยู่แอดเดรสได้เป็น 3 **ส่วนประกอบด้วยเนื้อที่**

- 64 **กิโลไบต์ หน่วยความจำโปรแกรม**
- 64 **กิโลไบต์ หน่วยความจำข้อมูลภายนอก**
- 256 **ไบต์ เป็นหน่วยความจำข้อมูลภายใน ส่วนเบอร์** 8032/8052 **มีขนาด** 384 **ไบต์**

2.2.4 เนื้อที่หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมจะประกอบด้วย ส่วนภายในและภายนอกชิฟ ถ้าขา EA **มีสถานะสูง** MCS-51 **จะบริการโปรแกรมภายใน ถ้าโปรแกรมมีความยาวไม่เกิน** 0FFFH (4K)

หรือ 1FFFFH (8K) สำหรับตัว 8052 ตำแหน่งตั้งแต่ 1000H ถึง 0FFFFH (หรือ 2000H - 0FFFFH สำหรับ 8052) จะเป็นการเฟทช์ข้อมูลภายนอก ถ้าขา EA มีสถานะต่ำ MCS-51 จะเฟทช์ข้อมูลภายนอกทั้งหมด ในทุกกรณีตัวนับโปรแกรมขนาด 16 บิต จะเป็นตัวกำหนดเลขที่อยู่โปรแกรม ซึ่งจะแสดงโครงสร้างของหน่วยความจำได้ดังรูปที่ 2.4



รูปที่ 2.4 แสดงแผนที่ของหน่วยความจำโปรแกรม

ตำแหน่ง 00H ถึง 32H (หรือ 00H ถึง 2BH สำหรับเบอร์ 8032/8052) ในหน่วยความจำโปรแกรมจะสำรองสำหรับใช้บริการอินเทอร์รัพต์ตามตารางที่ 2.8

ชนิดของการอินเทอร์รัพต์ใน MCS-51	ตำแหน่งเริ่มต้นของโปรแกรมบริการอาร์ลินเทอร์รัพต์ (Vector Interrupt)
อินเทอร์รัพต์ภายนอกชนิด 0 (IE0)	0003H
อินเทอร์รัพต์ของไทม์เมอร์ 0 (TF0)	000BH
อินเทอร์รัพต์ภายนอกชนิด 1 (IE1)	0013H
อินเทอร์รัพต์ของไทม์เมอร์ 1 (TF1)	001BH
อินเทอร์รัพต์ของพอร์ตสื่อสารอนุกรม (TI+RI)	0023H
อินเทอร์รัพต์ของไทม์เมอร์ 2 (TF2+EXF2)	002BH

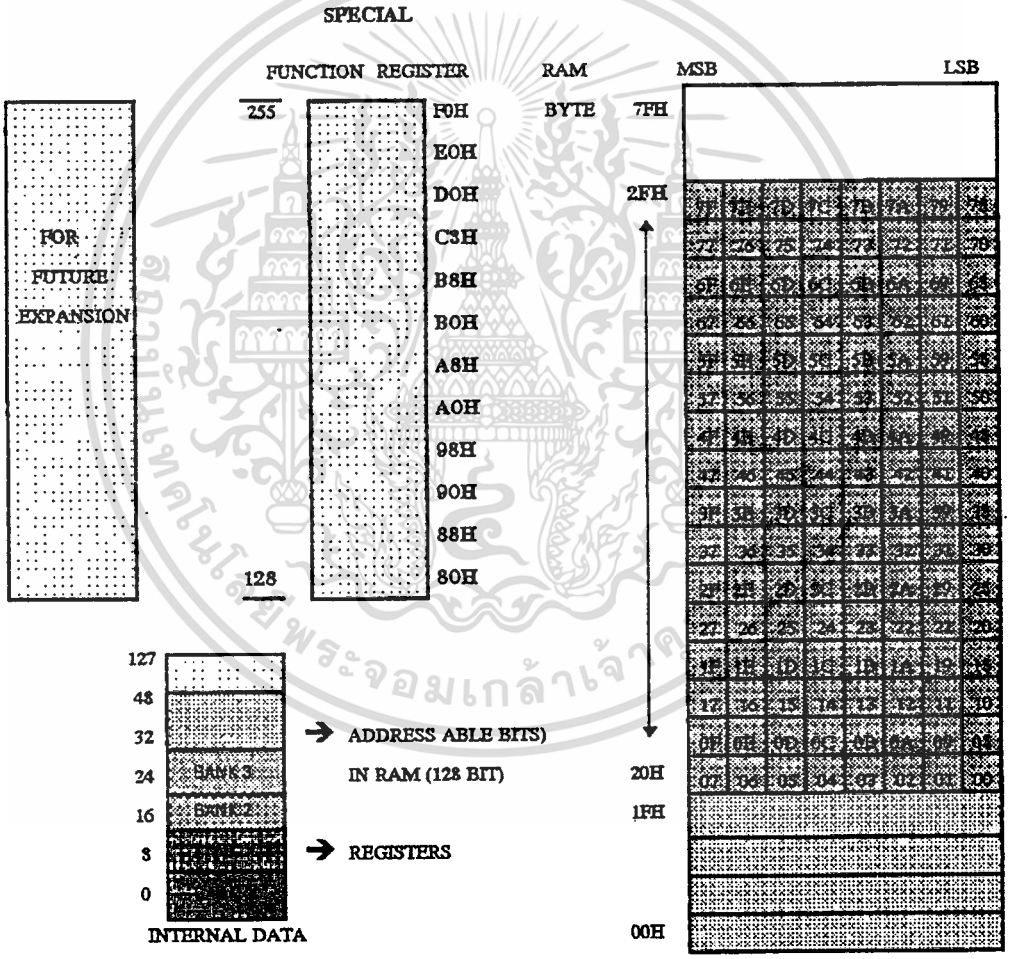
ตารางที่ 2.8 แสดงตำแหน่งเริ่มต้นของโปรแกรมบริการอินเทอร์รัพต์

2.2.5 เนื้อหน่วยความจำข้อมูล

หน่วยความจำข้อมูลจะประกอบด้วยความจำข้อมูลภายในและภายนอก หน่วยความจำข้อมูลภายนอกจะเข้าถึงได้ด้วยการใช้คำสั่ง MOVX

หน่วยความจำข้อมูลภายในจะแบ่งเป็นลักษณะงานดังนี้คือ

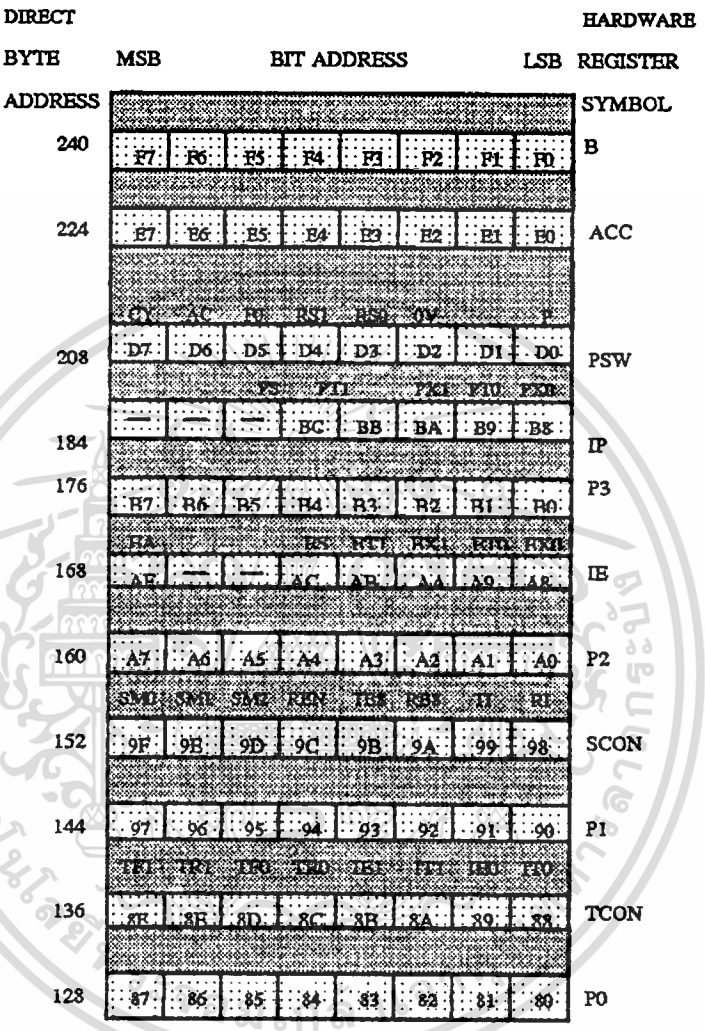
1. จำนวน 128 ไบต์ของบริเวณตำแหน่งล่างในเนื้อที่ RAM ภายใน
2. อีก 128 ไบต์เป็นของบริเวณตำแหน่งบนของ หน่วยความจำข้อมูลภายใน



รูปที่ 2.5 (A) แสดงแผนที่ของหน่วยความจำข้อมูล (B) แสดงแผนที่การกำหนดตำแหน่งบิต

ส่วนบนนี้จะมีเฉพาะในเบอร์ 8032/8052 เท่านั้น และส่วนของ 128 ไบต์อีกบริเวณหนึ่งใช้เป็นรีจิสเตอร์ฟังก์ชั่นพิเศษ ขณะที่ใช้ส่วนบน ของหน่วยความจำข้อมูลภายในและ

บริเวณของ SFR ทั้ง 2 ส่วนนี้จะถูกปันส่วนให้ใช้ค่าแอดเดรสภายในแต่จะเข้าถึงในแต่ละบริเวณของทั้ง 2 บริเวณนี้ได้โดยการใช้โหมดการกำหนดเฉพาะที่อยู่ต่างกัน

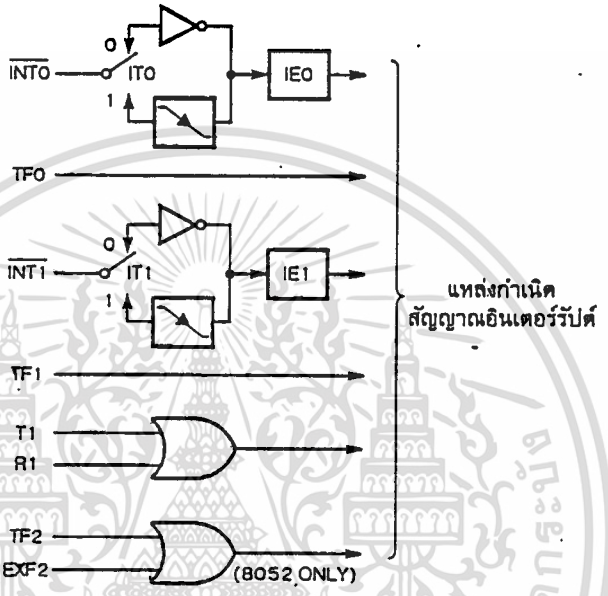


รูปที่ 2.6 แสดงตำแหน่งของรีจิสเตอร์ SFR และบิตแอดเดรสของ SFR

รูปที่ 2.5 (A) แสดงถึงแผนที่ของหน่วยความจำข้อมูลโดยแบ่งเป็น 4 แบงค์ ในแต่ละแบงค์มีรีจิสเตอร์ 8 ตัว มีตำแหน่งตั้งแต่ 0H-31H ในบริเวณของหน่วยความจำข้อมูลแบงค์เหล่านี้จะถูกเลือกใช้ให้อินาเบิ้ลได้คราวละ 1 แบงค์ด้วยการกำหนดเริ่มแรกภายใน 2 บิตของรีจิสเตอร์ PSW ว่าจะเลือกใช้ในแบงก์ใดภายใน 4 แบงก์ และบริเวณตำแหน่งตั้งแต่ 20H - 2FH จำนวน 16 ตำแหน่ง ตำแหน่งละ 1 ไบต์ สามารถที่จะกำหนดเฉพาะที่อยู่ของแต่ละบิตได้ดังแสดงในรูปที่ 2.5(B) เป็นบิตแรกแอดเดรส นี้อที่รีตจสเตอร์ SFR สามารถที่จะกำหนดตำแหน่งได้เช่นกัน ดังรูปที่ 2.6

2.2.6. โครงสร้างทางอินเตอร์รัพต์ MCS-51

MCS-51 สามารถรับสัญญาณอินเตอร์รัพต์ที่เกิดขึ้นอย่างน้อย 5 ชนิด (MCS-51 บางเบอร์ในตระกูลนี้สามารถรับสัญญาณอินเตอร์รัพต์ได้มากกว่า 5 ชนิด เช่นเบอร์ 8052สามารถรับได้ 6 ชนิด) แหล่งกำเนิดสัญญาณอินเตอร์รัพต์ทั้ง 5 ชนิดที่ MCS-51 สามารถรับได้มีดังแสดงในรูปที่ 2.7



รูปที่ 2.7 แหล่งกำเนิดสัญญาณอินเตอร์รัพต์ทั้ง 5 ชนิดที่ MCS-51 สามารถรับได้

อินเตอร์รัพต์แต่ละชนิดที่ MCS-51 สามารถรับได้มีรายละเอียดดังต่อไปนี้

- อินเตอร์รัพต์ที่เกิดจากภายนอก (External Interrupt) เป็นอินเตอร์รัพต์ที่เกิดขึ้นจากภายนอก MCS-51 มีอยู่ 2 ชนิดด้วยกันคือ
  - อินเตอร์รัพต์ภายนอกชนิด 0 รับได้จากขา INT0
  - อินเตอร์รัพต์ภายนอกชนิด 1 รับได้จากขา INT1

ผู้ใช้สามารถกำหนดให้ MCS-51 ตรวจสอบสัญญาณอินเตอร์รัพต์ทั้งสองชนิดที่เกิดขึ้นที่ขา INT0 , INT1 2 แบบด้วยกันคือ

- ตรวจสอบจากระดับสัญญาณ (level-activated)
- ตรวจสอบจากการเปลี่ยนสถานะสัญญาณ (transition-activated)

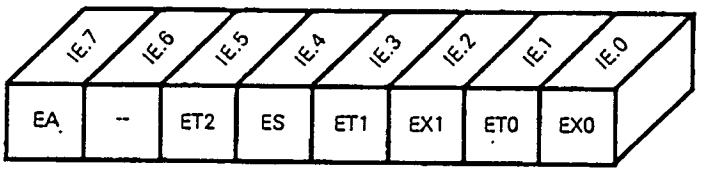
การตรวจสอบสถานะของสัญญาณอินเตอร์รัพต์ภายนอกที่ขาทั้งสอง สามารถเลือกได้เพียงอย่างใดอย่างหนึ่งขึ้นอยู่กับการกำหนดค่าบิต IT0 , IT1 (Interrupt Type Control bit) ในรีจิสเตอร์ใช้งานเฉพาะ TCON ดังแสดงในรูปที่ 2.7

เมื่อ MCS-51 ตรวจพบสัญญาณอินเตอร์รัพต์จากภายนอก จะมีผลทำให้บิต IE0 (จากขา INT0) หรือ IE1 (จากขา INT1) ของรีจิสเตอร์ใช้งานเฉพาะ TCON ถูกเซ็ต บิตทั้งสองจะเป็นตัวบอกสถานะของสัญญาณอินเตอร์รัพต์ที่เกิดจากภายนอก โดยถูกเซ็ตเมื่อเกิดสัญญาณอินเตอร์รัพต์และจะถูกเคลียร์โดยฮาร์ดแวร์ภายใน MCS-51 เอง เมื่อซีพียูย้ายไปทำงานที่โปรแกรมบริการอินเตอร์รัพต์ ต่อเมื่อ สัญญาณอินเตอร์รัพต์ภายนอกที่เกิดขึ้นเป็นชนิดที่ตรวจสอบได้จากการเปลี่ยนสถานะสัญญาณ แต่ถ้าสัญญาณอินเตอร์รัพต์ภายนอกที่เกิดขึ้นได้มาจากการตรวจสอบระดับสัญญาณเมื่อซีพียูย้ายไปทำงานที่โปรแกรมบริการอินเตอร์รัพต์จะไม่มีการเคลียร์บิต IE0 หรือ IE1 ให้ ในกรณีนี้จะเป็นหน้าที่ของวงจรกำเนิดสัญญาณอินเตอร์รัพต์ภายนอกที่ต้องทำหน้าที่ควบคุมสถานะของสัญญาณที่ขา INTx (INT0 หรือ INT1) ให้กลับสู่สภาพเดิมเอง มิฉะนั้นโปรแกรมหลักที่ทำงานอยู่จะถูกอินเตอร์รัพต์ไปเรื่อย ๆ จนกระทั่งสัญญาณอินเตอร์รัพต์กลับมีค่าเป็น 1 อีกครั้ง

- **อินเตอร์รัพต์ของไทม์เมอร์ 0 และไทม์เมอร์ 1** อินเตอร์รัพต์ของไทม์เมอร์ 0 หรือไทม์เมอร์ 1 ถูกทำให้เกิดขึ้นโดยบิต TF0 หรือ TF1 ซึ่งถูกเซ็ตเมื่อไทม์เมอร์ 0 หรือไทม์เมอร์ 1 เกิดโอเวอร์โฟลว์ (overflow) (มีการเปลี่ยนค่าจาก 1 ทั้งหมดมาเป็น 0 ทั้งหมดในรีจิสเตอร์ที่ใช้เป็นไทม์เมอร์หรือเคาน์เตอร์ของไทม์เมอร์ 0 หรือไทม์เมอร์ 1) ยกเว้นไทม์เมอร์ 0 ในโหมด 3 ซึ่งหยุดการทำงานเมื่อมีอินเตอร์รัพต์จากไทม์เมอร์เกิดขึ้น บิต TF0 และ TF1 จะถูกเคลียร์โดยฮาร์ดแวร์ภายใน MCS-51 เองเมื่อซีพียูย้ายไปทำงานที่ภโปรแกรมบริการอินเตอร์รัพต์
- **อินเตอร์รัพต์ของพอร์ตสื่อสารอนุกรม (Serail Port Interrupt)** พอร์ตสื่อสารอนุกรมของ MCS-51 สามารถทำให้เกิดสัญญาณอินเตอร์รัพต์ได้ สัญญาณอินเตอร์รัพพต์ที่เกิดขึ้นได้มาจากบิต TI หรือ RI ที่นำมาผ่านเกตออร์ (ดังแสดงในรูปที่ 2.7) และบิตที่ควบคุมการอินเตอร์รัพต์ทั้งสองนี้จะไม่ถูกเคลียร์โดยฮาร์ดแวร์ใน MCS-51 เมื่อซีพียูไปทำงานในโปรแกรมบริการการอินเตอร์รัพต์ เพราะการเกิดอินเตอร์รัพต์ของพอร์ตสื่อสารอนุกรมอาจจะเกิดจากบิต RI หรือ TI ก็ได้ ดังนั้นโปรแกรมในส่วนบริการการอินเตอร์รัพต์จะต้องตรวจสอบเองว่าสัญญาณอินเตอร์รัพต์ที่เกิดขึ้นได้มาจากบิต RI หรือ TI และบิตทั้งสองจะถูกเคลียร์โดยซอร์ฟแวร์เท่านั้น

อินเตอร์รัพต์แต่ละชนิดที่กล่าวไปแล้ว สามารถถูกควบคุมให้สามารถอินเตอร์รัพต์ MCS-51 ได้หรือไม่ โดยการควบคุมจากบิตต่าง ๆในรีจิสเตอร์ใช้งานเฉพาะ IE ดังแสดงในรูปที่ 2.8

- **รีจิสเตอร์ใช้งานเฉพาะ IE (Interrupt Enable-Register)** เข้าถึงข้อมูลได้ในระดับบิต การกำหนดให้บิตควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพต์แต่ละชนิดมีค่าเป็น 0 หมายถึงไม่ให้ MCS-51 ตอบสนองต่อสัญญาณอินเตอร์รัพต์ชนิดนั้น หากกำหนดให้บิตควบคุมการตอบสนองต่ออสัญญาณอินเตอร์รัพต์แต่ละชนิดมีค่าเป็น 1 หมายถึงให้ MCS-51 ตอบสนองต่อสัญญาณอินเตอร์รัพต์ชนิดนั้น (บิต EA ต้องถูกเซ็ตไว้ก่อนด้วย)



บิต	ชื่อบิต	
IE.7	EA	ใช้ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัปต์ทั้งหมด 0: MCS-51 จะไม่ตอบสนองต่อสัญญาณอินเตอร์รัปต์ใด ๆ ทั้งสิ้น 1 : อินเตอร์รัปต์แต่ละชนิดจะถูกควบคุมการตอบสนองอย่างอิสระจากบิตในรีจิสเตอร์นี้
IE.6	--	ไม่ถูกกำหนดการใช้งาน (สำรองไว้ใช้ใน MCS-51 เบอร์ใหม่ ๆ ในอนาคต)
IE.5	ET2	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของไทม์เมอร์ 2 เมื่อเกิด overflow (มีใช้เฉพาะ MCS-51 บางเบอร์ที่มีไทม์เมอร์ 2 เช่น 8052)
IE.4	ES	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของพอร์ตสื่อสารอนุกรม
IE.3	ET1	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของไทม์เมอร์ 1 เมื่อเกิด overflow
IE.2	EX1	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัปต์ภายนอกชนิด 1
IE.1	ET0	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของไทม์เมอร์ 0 เมื่อเกิด overflow
IE.0	EX0	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัปต์ภายนอกชนิด 0

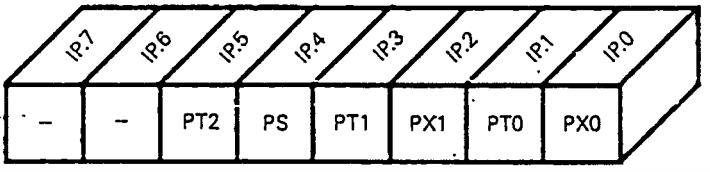
รูปที่ 2.8 รีจีสเตอร์ใช้งานเฉพาะ IE

บิต EAในรีจีสเตอร์ใช้งานเฉพาะ IE สามารถควบคุมการอินเตอร์รัพต์ใน MCS-51 ได้ทั้งหมด หากบิตนี้มีค่าเป็น 0 สัญญาณอินเตอร์รัพต์ทุกชนิดที่เกิดขึ้น จะไม่สามารถอินเตอร์รัพต์ MCS-51 ได้ แต่หากบิตนี้มีค่าเป็น 1 สัญญาณอินเตอร์รัพต์แต่ละชนิดจะถูกควบคุมให้อินเตอร์รัพต์MCS-51 ได้อย่างอิสระ (ควบคุมจากบิต IE.0-IE.5)

บิต IE.5 , IE.6 ไม่ถูกใช้ใน 8051 เพราะถูกสงวนไว้ใช้ใน MCS-51 เบอร์อื่น ๆ ที่สามารถรับอินเตอร์รัพต์ได้เพิ่มขึ้น ดังนั้นซอร์ฟแวร์ของผู้ใช้ไม่ควรจะมีคำสั่งเขียนค่า 1 ลงไปในบิตเหล่านี้ เพื่อให้โปรแกรมยังคงสามารถใช้กับชิพเบอร์ใหม่ ๆ ในตระกูลนี้ได้

2.2.7 โครงสร้างระดับความสำคัญในการบริการอินเตอร์รัพต์ (Priority Level Structure) อินเตอร์รัพต์แต่ละชนิดสามารถถูกเลือกระดับความสำคัญในการบริการได้ 2 ระดับ โดยการเซ็ตหรือเคลียร์บิตในรีจีสเตอร์ใช้งานเฉพาะ IP ดังแสดงในรูปที่ 2.9

- **รีจีสเตอร์ใช้งานเฉพาะ IP (Interrupt Priority Register)** เข้าถึงข้อมูลได้ในระดับบิต การให้บิตกำหนดความสำคัญของอินเตอร์รัพต์เป็น 0 หมายถึง ให้อินเตอร์รัพต์ชนิดนั้นมีลำดับความสำคัญต่ำ ส่วนการให้บิตกำหนดลำดับความสำคัญของอินเตอร์รัพต์เป็น 1 หมายถึงให้อินเตอร์รัพต์ชนิดนั้นมีลำดับความสำคัญสูง



บิต	ชื่อบิต	
IP.7	–	ไม่ถูกกำหนดการใช้งาน (สำรองไว้ใช้ใน MCS-51 เบอร์ใหม่ ๆ ในอนาคต)
IP.6	–	ไม่ถูกกำหนดการใช้งาน (สำรองไว้ใช้ใน MCS-51 เบอร์ใหม่ ๆ ในอนาคต)
IP.5	PT2	กำหนดลำดับความสำคัญในการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของไทม์เมอร์ 2
IP.4	PS	กำหนดลำดับความสำคัญในการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของพอร์ตสื่อสารอนุกรม
IP.3	PT1	กำหนดลำดับความสำคัญในการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของไทม์เมอร์ 1
IP.2	PX1	กำหนดลำดับความสำคัญในการตอบสนองต่อสัญญาณอินเตอร์รัปต์ภายนอกชนิด 1
IP.1	PT0	กำหนดลำดับความสำคัญในการตอบสนองต่อสัญญาณอินเตอร์รัปต์ของไทม์เมอร์ 0
IP.0	PX0	กำหนดลำดับความสำคัญในการตอบสนองต่อสัญญาณอินเตอร์รัปต์ภายนอกชนิด 0

รูปที่ 2.9 รีจีสเตอร์ใช้งานเฉพาะ IP

ระดับความสำคัญในการบริการอินเตอร์รัพต์ที่เกิดขึ้นมีได้ 2 ระดับ ได้แก่

- **อินเตอร์รัพต์ความสำคัญต่ำ (Low Priority Interrupt)** : อินเตอร์รัพต์ชนิดนี้สามารถถูกอินเตอร์รัพต์จากสัญญาณอินเตอร์รัพต์ระดับความสำคัญสูงได้ แต่จะไม่สามารถถูกอินเตอร์รัพต์โดยสัญญาณอินเตอร์รัพต์ระดับความสำคัญต่ำตัวอื่น ๆ ได้

- **อินเตอร์รัพต์ความสำคัญสูง (High Priority Interrupt)** : อินเตอร์รัพต์ประเภทนี้ไม่สามารถถูกอินเตอร์รัพต์โดยสัญญาณอินเตอร์รัพต์ชนิดอื่นได้เลย นั่นคือมีระดับความสำคัญสูงสุด

ถ้ามีสัญญาณอินเตอร์รัพต์เกิดขึ้นพร้อมกัน 2 ชนิด โดยมีระดับความสำคัญในการบริการอินเตอร์รัพต์ไม่เท่ากัน สัญญาณอินเตอร์รัพตที่มีระดับความสำคัญสูงกว่าจะได้รับการบริการก่อนแต่ถ้ามีการขออินเตอร์รัพต์พร้อมกัน 2 ชนิด ซึ่งมีระดับความสำคัญเท่าเทียมกัน ลำดับการบริการอินเตอร์รัพต์ภายในจะเป็นตัวกำหนดเองว่าอินเตอร์รัพต์ชนิดใดควรถูกบริการก่อน ดังนั้นภายในระดับความสำคัญของการบริการอินเตอร์รัพต์หนึ่ง ๆ จะมีระดับความสำคัญในการบริการอินเตอร์รัพต์ย่อยลงไปอีกระดับ (second priority structure) ดังแสดงในตารางที่ 2.9

การจัดการกับสัญญาณอินเตอร์รัพต์ที่มีระดับความสำคัญเท่าเทียมกัน ที่เกิดขึ้นพร้อมกัน (priority within level structure) ถูกใช้เพียงเพื่อแก้ปัญหาสัญญาณอินเตอร์รัพต์ที่มีระดับความสำคัญเท่ากันที่เกิดขึ้นพร้อมกันเท่านั้น (simultaneous reques of the same priority level)

เหตุการณ์ที่เกิดอินเตอร์รัพต์	ลำดับความสำคัญภายใน MCS-51
อินเตอร์รัพต์ภายนอกชนิด 0 (IE0)	สูงสุด
อินเตอร์รัพต์ไทม์เมอร์ 0 (TF0)	↑
อินเตอร์รัพต์ภายนอกชนิด 1 (IE1)	
อินเตอร์รัพต์ไทม์เมอร์ 1 (TF1)	
อินเตอร์รัพต์ของพอร์ทสื่อสารอนุกรม(TI+RI)	
อินเตอร์รัพต์ไทม์เมอร์ 2 (TF2+EXF2)	ต่ำสุด

ตารางที่ 2.9 ระดับความสำคัญในการบริการอินเตอร์รัพต์

รีจิสเตอร์ใช้งานเฉพาะ IP มีบิตที่ไม่ถูกใช้งานอยู่บางบิต คือ IP.7 และ IP.6 โดยไม่ถูกใช้ใน 8051 และ 8052 ใน 8051 จะมีบิตที่ว่างเพิ่มมาอีก 1 บิตคือ IP.5 ดังนั้นซอร์ฟแวร์ของผู้ใช้ไม่ควรมีการเขียนค่า 1 ไปที่ตำแหน่งบิตเหล่านี้ เพราะมันอาจถูกนำไปใช้ใน MCS-51 เบอร์ใหม่ ๆ ในอนาคตต่อไป

2.2.8 ตัวจับเวลา/ตัวนับ (Timer/Coumter)

MCS-51 มี 16 บิต ตัวจับเวลา/ตัวนับ 2 ตัว คือ ตัวจับเวลา/ตัวนับ 0 และตัวจับเวลา/ตัวนับ 1 ส่วน 8032/8052 มีเพิ่มอีกหนึ่งชุด คือ ตัวจับเวลา/ตัวนับ 2 ขณะที่แต่ละ ตัวจับเวลา/ตัวนับ สามารถที่จะติดตั้งให้ทำงานเป็นตัวจับเวลาหรือตัวนับก็ได้

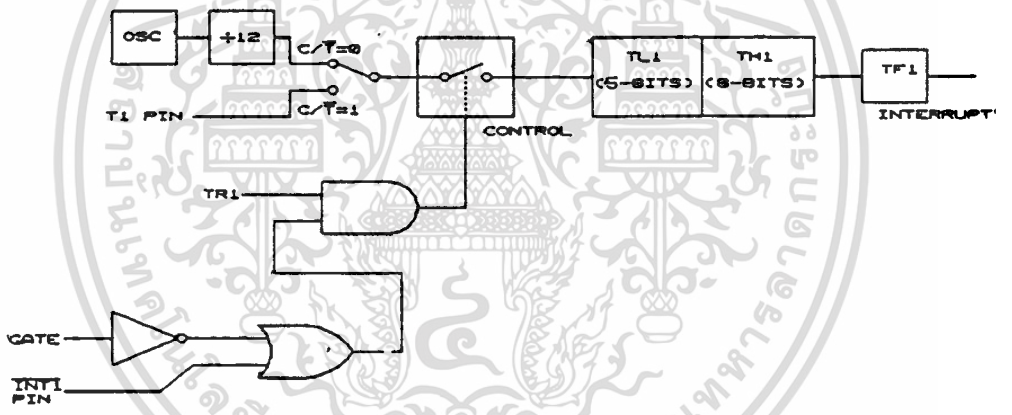
2.2.8.1 ตัวจับเวลา/ตัวนับ 0 และตัวจับเวลาตัวนับ 1

แต่ละตัวจะถูกติดตั้งให้ทำงานเป็นตัวจับเวลาหรือตัวนับ ได้ด้วยการเซ็ตหรือเคลียร์บิตที่ตัวควบคุมในรีจิสเตอร์ TMOD ในกลุ่ม SFR ในฟังก์ชันตัวจับเวลาตัวรีจิสเตอร์จะเพิ่มค่าทุก ๆ วัฏจักรแมชชีน ดังนั้นตัวเลขในรีจิสเตอร์จะเป็นจำนวนของวัฏจักรแมชชีน เนื่องจากแต่ละวัฏจักรแมชชีนประกอบด้วย 12 คาบของออสซิลเลเตอร์ อัตราการนับแต่ละครั้งจะกินเวลาเป็น 1/12 ของความถี่ออสซิลเลเตอร์

ในฟังก์ชันตัวนับรีจิสเตอร์จะเพิ่มค่าทุกครั้งที่ที่การเปลี่ยนแปลงสถานะจาก '1' เป็น '0' ที่เข้ามาที่ขา T0 หรือ T1 ในฟังก์ชันนี้สัญญาณภายนอกที่เข้ามาจะถูกรับแซมปลิ้ง (Sampling) ระหว่างช่วง S5P2 ของทุกวัฏจักรแมชชีน โดยถ้าแซมปลิ้งสัญญาณเข้าเป็นระดับสูงในวัฏจักรหนึ่ง ดังนั้นถ้าในวัฏจักรตัวต่อมาของสัญญาณเข้าเป็นระดับต่ำรีจิสเตอร์จะนับเพิ่มหนึ่งค่า

โดยที่ค่าไหม่ของตัวนับ จะปรากฏที่รีจีสเตอร์ช่วง S3P1 ของวัฏจักร ซึ่งค่าหนึ่งที่รับเข้าไป จะใช้ช่วง 2 วัฏจักรแมชชีน (เท่ากับ 24 คาบ) ในการรับค่าช่วงการเปลี่ยน 1 เป็น 0 ดังนั้นค่าสูงสุดในการนับจะมีอัตรา 1/24 เท่าของความถี่ออสซิลเลเตอร์และสัญญาณอินพุตที่จะนับนั้นจะไม่มีช่วงระยะห่างที่แน่นอนของดูตี้ไซเคิ้ล (Duty cycle) แต่จะถูกนับเมื่อระดับแรงดันที่ถูกแซมปลิ้งในแต่ละครั้ง จะต้องมีช่วงคงที่อย่างน้อย 1 วัฏจักรแมชชีนก่อนที่จะเปลี่ยนค่าระดับแรงดันใหม่ ในการเลือกทำงานระหว่างตัวนับกับตัวจับเวลา จะเลือกได้ 4 โหมด คือ โหมด 0 , 1 และ 2 เลือกได้ทั้งสองตัวของ ตัวจับเวลา/ตัวนับ ส่วนโหมด 3 ทำงานแตกต่างออกไป

- โหมด 0 การใช้ตัวจับเวลา/ตัวนับ 0 หรือ 1 อยู่ในโหมด 0 จะคล้ายกับการทำงานของ MCS-48 โดยตัวจับเวลาของ MCS-48 มีขนาด 8 บิต มีตัวพรีสเคเลอร์ (Prescaler) เป็นตัวหาร 12 รูปที่ 2.10 แสดงการทำงานในโหมด 0 ของจับเวลา/ตัวนับ 1



รูปที่ 2.10 แสดงการทำงานในโหมด 0 ของตัวจับเวลา/ตัวนับ 1 ขนาด 13 บิต

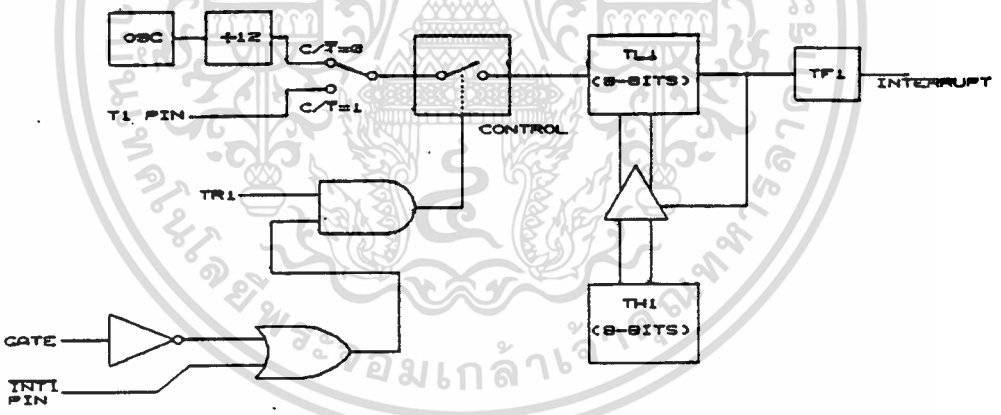
ในโหมดนี้รีจีสเตอร์ตัวจับเวลาถูกกำหนดให้มี 13 บิต ด้วยการนับขึ้นเมื่อเป็น'1' หมดทุกบิต จะกลับมาที่ '0' ทุกบิตใหม่ เมื่อกลับเป็น '0' ทุกบิตจะเกิดการโอเวอร์โฟลว์ (Over-flow) ไปทดไหแฟลกอินเทอร์รัพต์ TF1 ปรับเป็น '1' การควบคุมให้เริ่มนับตัวอินพุตจะควบคุมด้วยการอีนาเบิ้ล TR1 = 1 , GATE = 0 และขา $\overline{INT1}$ = 1 การปรับ GATE = 1 เป็นการติดตั้งตัวนับให้นับด้วยสัญญาณจากภายนอกที่เข้ามาที่ขา $\overline{INT1}$ TR1 จะเป็นบิตควบคุมในรีจีสเตอร์ TMOD ของสัญญาณ SFR

รีจีสเตอร์ตัวนับจะมี 13 บิต ประกอบด้วย TH1 8 บิต และ TL1 อีก 5 บิตอันดับต่ำ ส่วนอีก 3 บิตที่เหลือในอันดับสูงของ TL1 จะไม่ใช้ การเซ็ตแฟลก TR1 ให้ทำงานจะไม่ได้

เคลียร์ค่าในรีจีสเตอร์ของ TH1 และ TL1 การทำงานในโหมด 0 ในตัวจับเวลา/ตัวนับ 0 จะทำงานเหมือนกับตัวจับเวลา/ตัวนับ 1 โดยใช้ TR0 และ INT0 รวมกันควบคุมแทนสัญญาณต่าง ๆ ในรูปที่ 2.10 มีความแตกต่างในการควบคุมคือบิตของ GATE ทั้งสองตัวหนึ่งจะแทนตัวจับเวลา/ตัวนับ 1 (TMOD.7) และอีกตัวจะแทนตัวจับเวลา/ตัวนับ 0 (TMOD.3)

- โหมด 1 โหมด 1 ทำงานเหมือนกับโหมด 0 ต่างกันเฉพาะการใช้รีจีสเตอร์ตัวจับเวลา/ตัวนับ จะทำงานด้วยขนาด 16 บิต โดยไม่มีพรีสเคเลอร์ คือความถี่ 1/12 ของออสซิลเลเตอร์ เป็นความถี่ที่เข้ามาถูกหารด้วยค่า 16 บิตในรีจีสเตอร์ตัวนับ

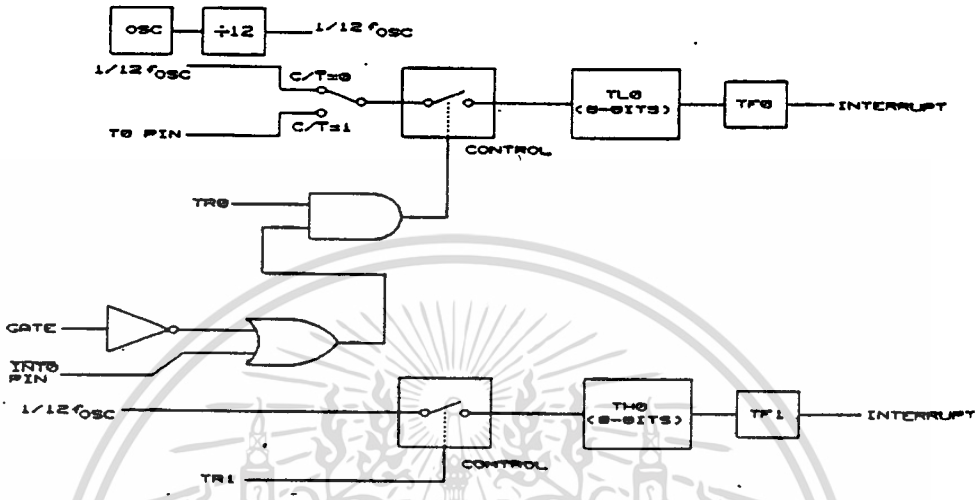
- โหมด 2 โหมด 2 มีการทำงานโดยการกำหนดให้ตัวนับ 8 บิตของ TL! และจะโหลดใหท่โดยอัตโนมัติทุกครั้ง เมื่อมีการโอเวอร์โฟลว์จาก TL1 ดังรูปที่ 2.11 ไม่เพียงแต่ TF1 จะปรับเป็น '1' แต่ TL1 จะถูกโหลดโดยอัตโนมัติจากค่าที่ตั้งไว้ใน TH1 ซึ่งค่าใน TH1 สามารถตั้งค่าได้ด้วยซอฟต์แวร์ คือการใช้คำสั่ง MOV และบรรจุเข้าไปใหม่ที่ TL1 ทุกครั้งที่เกิดโอเวอร์โฟลว์TH0 และ TF0 จะเป็นตัวร่วมทำงานในโหมดนี้



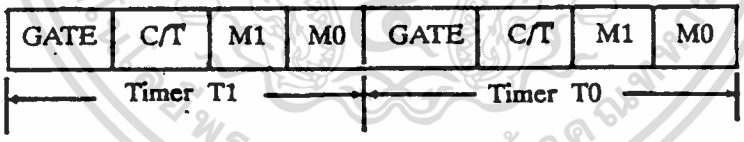
รูปที่ 2.11 ตัวจับเวลา/ตัวนับ 1 ทำงานในโหมด 2 แบบโหลดใหม่ 8 บิต

- โหมด 3 ถ้าใช้ตัวจับเวลา/ตัวนับ 1 ในโหมด 3 มีการทำงานเป็นตัวนับ มีผลเช่นเดียวกับการตั้ง TR1 = 0 และใช้ตัวจับเวลาตัวนับ 0 ในโหมด 3 จัดการให้ TL0 ใช้ตัวจับเวลา/ตัวนับ 0 ร่วมกับบิตควบคุมของ C/T ,GATE ,TR0 ,INT0 และ TF0 ตัว TH0 จะถูกล็อคให้ทำงานในฟังก์ชันตัวจับเวลา ตามรูปที่ 2.12 และใช้บิตแฟลก TR1 และ TF1 เข้าร่วมทำงานในโหมด 3 ดังนั้นตัว TH0 ในโหมดนี้จะควบคุมการอินเทอร์รัพต์ของตัวจับเวลา 1 เป็นกลุ่มจับเวลาและขนาด 8 บิตสองตัว ตัวจับเวลา (เป็นวัฏจักรแมชชีนได้) และจะใช้บิตที่ TR1 และ TF1ของ ตัวจับเวลา 1

เป็นตัวควบคุมดังนั้นจึงใช้ TH0 เป็นตัวจับเวลาเป็นการควบคุมการอินเทอร์รัพต์ด้วยตัวจับเวลา 1



รูปที่ 2.12 ใช้ตัวจับเวลา/ตัวนับ 1 ในโหมด 3 เป็นกลุ่มตัวนับขนาด 8 บิต



GATE : ควบคุมเกต เมื่อเซตเป็น '1' จะเป็นอินาเบิ้ล ตัวจับเวลา/ตัวนับเท่านั้น ขณะที่ขา INTx มีสถานะสูง และขาควบคุม TRx ใน TCON จะถูกเซตเป็น '1'เมื่อตัวนับภายในถูกเคลียร์ให้เป็นอินาเบิ้ล เมื่อไรก็ตามที่บิตควบคุม TRx ถูกเซตเป็น '1'

C/T: เลือกการทำงานแบบตัวจับเวลาหรือตัวนับ ถ้าเป็น '0' จะเลือกทำงานเป็นตัวจับเวลา (โดยใช้สัญญาณนาฬิกาภายในเป็นสัญญาณเข้าอ้างอิงถึง) ถ้าเป็น '1' จะเป็นการทำงานแบบตัวนับ และรับสัญญาณเข้าที่ขา Tx

M1	M0	การทำงาน
0	0	ทำงานแบบตัวจับเวลาของ MCS-48 ใช้ TLx เป็นตัวป้อนบิตอีก 5 บิต ;
0	1	การใช้ตัวจับเวลา/ตัวนับ ขนาด 16 บิต จะใช้ THx และ TLx เป็นตัวนับไม่มี Prescaler
1	0	การโหลดขนาด 8 บิตโดยอัตโนมัติที่ตัวนับและตัวจับเวลา โดยใช้ THx เก็บค่าที่ตั้งไว้และจะถ่ายเข้าไปที่ TLx ใหม่ทุกครั้งที่เกิด Overflow คือ TLx ถูกนับเป็น '0' หมด
1	1	ตัวจับเวลา 0 ทำงาน โดยให้ TL0 และ TH0 เป็นตัวนับแยกกัน

ตารางที่ 2.10 TMOD : Timer/Counter Mode Control Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	ตัวจับเวลา1 แฟลกเป็น '1' เมื่อเกิด Overflow ถูกเซตเป็นหนึ่งด้วยฮาร์ดแวร์ทางสัญญาณ เมื่อ ตัวจับเวลา/ตัวนับ Overflow และจะเคลียร์ตัวเองเมื่ออินเตอร์รัพตไปแล้ว
TR1	TCON.6	ตัวจับเวลา1 เป็นตัวควบคุมบิตให้เริ่มทำงาน จะเซตหรือเคลียร์ด้วยซอฟต์แวร์ที่จะมาทำให้ ตัวจับเวลา/ตัวนับ 1 เริ่มหรือหยุดการทำงาน
TF0	TCON.5	ตัวจับเวลา0 แฟลกเป็น '1'เมื่อเกิด Overflow ถูกเซตเป็นหนึ่งด้วยฮาร์ดแวร์ทางสัญญาณเมื่อตัวจับเวลา/ตัวนับ Overflow เคลียร์ตัวเองเมื่อเข้าอินเตอร์รัพตไปแล้ว
TR0	TCON.4	ตัวจับเวลา0 เป็นตัวควบคุมบิตให้เริ่มทำงาน จะเซตหรือเคลียร์ด้วยซอฟต์แวร์ที่จะมาทำให้ตัวจับเวลา/ตัวนับ เริ่มหรือหยุดการทำงาน
IE1	TCON.3	เป็นแฟลกขอบสัญญาณอินเตอร์รัพต์1 จะเซตด้วยฮาร์ดแวร์เมื่อสัญญาณขอบการอินเตอร์รัพต์ปรากฏเข้าที่ขา INT1 และเคลียร์เมื่อการทำงานอินเตอร์รัพตสิ้นสุด
IT1	TCON.2	รูปแบบการควบคุมบิตของอินเตอร์รัพต์1 จะเซตหรือเคลียร์ได้ด้วยซอฟต์แวร์ที่จะเป็นตัวกำหนดให้มีการกระตุ้นอินเตอร์รัพต์จากภายนอกที่เป็นขอบขาลงหรือเป็นระดับแรงดันต่ำ โดยถ้า IT1 =1 จะควบคุมอินเตอร์รัพต์แบบขอบขาลง และถ้า IT =0 จะควบคุมอินเตอร์รัพต์แบบระดับแรงดันต่ำ
IE0	TCON.1	เป็นแฟลกขอบสัญญาณอินเตอร์รัพต์0 เซตด้วยฮาร์ดแวร์เมื่อสัญญาณขอบการอินเตอร์รัพต์ปรากฏเข้าที่ขา INT0 และเคลียร์เมื่อการทำงานอินเตอร์รัพตสิ้นสุด
IT0	TCON.0	รูปแบบการควบคุมบิตของอินเตอร์รัพต์จะเซตหรือเคลียร์ได้ด้วยซอฟต์แวร์ที่จะเป็นตัวกำหนดให้มีการกระตุ้นอินเตอร์รัพต์จากภายนอกที่แบบขอบขาลงหรือเป็นแบบระดับแรงดันต่ำ

ตารางที่ 2.11 TCON : Timer/Counter Control Register

โหมด 3 สามารถที่จะใช้ในงานที่ต้องการตัวจับเวลา/ตัวนับ ขนาด 8 บิตที่เพิ่มขึ้นด้วยการใช้ตัวจับเวลา 0 ในโหมด 3 ดังนั้น 8051 สามารถที่จะทำงานใช้ตัวจับเวลา/ตัวนับ ได้เป็น 3 ชุดขณะที่ 8052 ก็จะใช้งานได้ 4 ชุด เมื่อใช้ตัวจับเวลา 0 อยู่ในโหมด 3 ตัวจับเวลา 1 สามารถที่จะเปิด-ปิดให้เข้าสู่หรือออกจากการทำงานของโหมด 3 หรือสามารถที่จะยังคงใช้เป็นตัวสร้างอัตราบ๊อตของการส่งข้อมูลอนุกรมหรือการใช้งานใด ๆ ที่ไม่ต้องการการอินเทอร์รัพต์

2.2.9 การต่อเชื่อมแบบอนุกรม

พอร์ตอนุกรมเป็นแบบฟูลดูเพล็กซ์ สามารถที่จะส่งและรับพร้อมกันได้ โดยทำหน้าที่เป็นบัฟเฟอร์การรับ หมายถึงพอร์ตสามารถที่จะรับไบต์ที่สองก่อนที่ตัวแรกจะถูกรับไปจาก

รีจีสเตอร์ตัวรับ อย่างไรก็ตามไบต์ตัวแรกจะต้องถูกอ่านไปก่อนที่ช่วงเวลาการรับไบต์ที่สองจะสิ้นสุดมิฉะนั้นไบต์ตัวแรก จะถูกซ้อนและสูญหายไปได้ในพอร์ตอนุกรมรีจีสเตอร์ตัวรับและส่ง จะเข้าถึงติดต่อกันด้วยรีจีสเตอร์ SBUF ใน SFR แม้ว่าโครงสร้างทางรีจีสเตอร์ทั้งสองจะแยกกันอยู่ก็ตาม

พอร์ตอนุกรมที่จะเลือกทำงานในโหมดต่าง ๆ ได้ใน 4 โหมด

● โหมด 0 ข้อมูลจะเข้าออกผ่น RXD TXD ด้วยการเลื่อนสัญญาณนาฬิกาเอาท์พุต ข้อมูลจะเป็นลักษณะ 8 บิต ในการรับส่งแต่ละครั้ง โดยที่ส่งค่า LSB ก่อนอัตราบ็อตจะคงที่ที่ 1/12 ของความถี่ออสซิลเลเตอร์

● โหมด 1 จะเป็นการส่งข้อมูลขนาด 10 บิตผ่านออก TXD หรือรับเข้ามาผ่าน RXD โดยรูปแบบบิตจะประกอบด้วยหนึ่งบิตเริ่มต้นเป็น '0' แปดบิตข้อมูลโดยLSBเป็นตัวแรกที่รับและส่งและอีกหนึ่งบิตสิ้นสุดมีค่า '1' การรับบิตสิ้นสุดจะนำไปเก็บที่บิต RB8 ของ SFR รีจีสเตอร์ SCON อัตราบ็อดแปรผันได้ตามการตั้งตัวจับเวลาซึ่งจะกล่าวในหัวข้อต่อไป

● โหมด 2 เป็นการส่งข้อมูลขนาด 11 บิตผ่านออกขา TXD หรือรับเข้ามาผ่านขา RXD ประกอบด้วยหนึ่งบิตเริ่มต้นมีค่า '0' แปดบิตข้อมูลโดย LSB เป็นตัวแรกที่รับและส่งข้อมูล บิตที่เก้าของข้อมูลสามารถที่จะโปรแกรมเลือกได้ และบิตสิ้นสุดค่า '1' อีกหนึ่งบิต ในการส่งบิตที่เก้าที่อยู่ในบิต TB8 ของรีจีสเตอร์ SCON สามารถที่จะกำหนดเลือกเป็น '1' หรือ '0' ได้ตัวอย่าง เช่น การใช้งานแบบบิตพาริตี้ โดยการเลื่อนเอาบิต P ของ PSW มาไว้ใน TB8 เพื่อเป็นการส่งข้อมูลแบบมีการตรวจพาริตี้ของข้อมูลที่ส่ง ในการรับข้อมูลบิตที่เก้าจะเข้าไปเก็บที่ RB* ใน SFR รีจีสเตอร์ SCON ขณะที่บิตสิ้นสุดจะไปรับเข้ามาเก็บ อัตราบ็อดสามารถเลือกเป็น 1/32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์SCON เป็น SFR ที่ใช้ในการติดตั้งโหมดทำงานของพอร์ตอนุกรม เช่น การกำหนดค่า RB8 จะเป็นการใช้ตัวรับบิตที่เก้าด้วยหรือไม่ เป็นต้น ตารางที่ 5-6 เป็นตารางการใช้ SCON ในการควบคุมการทำงานของพอร์ตอนุกรม

● โหมด 3 เป็นการส่งข้อมูลขนาด 11 บิตผ่านออกขา TXD หรือรับเข้ามาผ่านขา RXD ประกอบด้วยบิตเริ่มต้นมีค่าเป็น '0' ข้อมูลแปดบิตโดย LSB เป็นบิตแรกที่รับและส่งข้อมูลที่บิตเก้าของข้อมูลสามารถที่จะโปรแกรมเลือกได้ และบิตสิ้นสุดค่า '1' อีกหนึ่งบิต ในความเป็นจริงโหมด 3 จะคล้ายกับโหมด 2 ทุกประการ ยกเว้นอัตราบ็อด โดยอัตราบ็อดในโหมด 3 จะแปรผันได้ไปตามการโปรแกรมเลือกตัวจับเวลา

ทั้งสี่โหมดนี้ การส่งข้อมูลจะเริ่มติดตั้ง Initiated ด้วยคำสั่งใด ๆ ที่ใช้ตัวรีจีสเตอร์ SBUF เป็นรีจีสเตอร์ตัวรับข้อมูลจากซีพียู และในโหมด 0 การรับข้อมูลเริ่มติดตั้งด้วยการใช้สถานะ RI=0 และ REN = 1 ส่วนในโหมดอื่นการรับข้อมูลจะเริ่มติดตั้งด้วยการรับบิตเริ่มต้นเข้ามาตรวจสอบถ้า REN = 1 การทำงานทั้งสี่โหมดของพอร์ตอนุกรมได้สรุปไว้ในตารางที่ 2.12 ต่อไปนี้เป็นการทำงานโดยละเอียดในโหมดต่าง ๆ ของพอร์ตอนุกรม

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

โดย SM0 , SM1 เป็นตัวกำหนดใช้โหมดต่าง ๆ ของพอร์ตอนุกรม ดังนี้

SM0	SM1	โหมด	ลักษณะการทำงาน อัตราบ๊อด
0	0	0	เลื่อนเรจิสเตอร์ fosc/12
0	1	1	8-บิต UART แปรผันได้ตามการเลือกตัวจับเวลา
1	0	2	9-บิต UART fosc/64 หรือ fosc/32
1	1	3	9-บิต UART แปรผัน

* UART : Universal Asynchronous Receiver / Tranceiver

SM2 ควบคุมอีนาเบิ้ล การใช้โพรเซสเซอร์หลายตัวในการสื่อสารซึ่งกันและกันในโหมด 2 และ 3 ถ้า SM2 เซตเป็น 1 ดังนั้น RI จะต้องไม่แอกทีฟ ถ้ามีการรับบิตที่เก้า ทำให้บิต RB8 นี้ เป็น 0ใน โหมด 1 ถ้า SM2 เซตเป็น 1 ดังนั้น RI จะไม่แอกทีฟถ้า STOP บิตไม่ถูกรับเข้ามาในโหมด 0 SM2 ควรมีค่าเท่ากับ 0

REN ตัวอีนาเบิ้ลอนุกรมการรับ เซตเป็น '1' ด้วยโปรแกรมในการเลือกอีนาเบิ้ลก รรับและเป็น '0' ด้วย โปรแกรม เมื่อให้เป็นดิสเอเบิ้ลการรับ

TB8 เป็นข้อมูลบิตที่เก้า ซึ่งจะถูกส่งในโหมด 2 และ 3 ซึ่งจะให้เป็น '1' หรอ '0'ได้ด้วยการโปรแกรม

RB8 ในโหมด 2 และ 3 ข้อมูลบิตที่เก้าจะถูกรับไปในโหมด 1 ถ้า SM2 = 0 RB8 จะกลายเป็น STOP บตที่ถูกรับไปในโหมด 0 RB8 ไม่ใช้

TI เป็นแฟลกอินเตอร์รัพต์การส่ง เซตด้วยฮาร์ดแวร์คือสัญญาณปลายช่วงเวลาของบิตที่แปดในโหมด 0 หรือที่จุดเริ่มต้นของบิต STOP ในโหมดอื่น ในการส่งแบบอนุกรมของทุกโหมดจะต้องเคลียร์บิตนี้ ด้วยโปรแกรมหลงการสมแล้ว

RI เป็นแฟลกอินเตอร์รัพต์การรับ เซตด้วย ฮาร์ดแวร์คือสัญญาณที่ปลายช่วงเวลาของบิตที่แปดในโหมด 0 หรือที่จุดครึ่งทางของช่วงบิต STOP ในโหมดอื่น ในการรับแบบอนุกรมยกเว้นกรณีใช้ SM2 จะ ต้องเคลียร์บิตด้วยโปรแกรมหลังการรับข้อมูลไปแล้ว

ตารางที่ 2.12 SCON : รีจีสเตอร์ควบคุมพอร์ตอนุกรม

2.2.9.1 อัตราบ๊อด

อัตราบ๊อดในโหมด 0 ของการใช้พอร์ตอนุกรมจะคงที่ที่ความถี่ของออสซิลเลเตอร์คือ

อัตราบ๊อดในโหมด 0 = Oscillator Frequency / 12

อัตราบ๊อดในโหมด 2 จะขึ้นอยู่กับการปรับค่าบิตใน SMOD ของ SFR ในรีจีสเตอร์ PCON ถ้า SMOD = 0 ซึ่งจะเป็นค่าที่ถูกรีเซ็ตแต่แรก หลังการรีเซ็ตอัตราบ๊อดจะเป็น 1/64 ของความถี่ออสซิลเลเตอร์ ถ้า SMOD = 1 อัตราบ๊อดจะเป็น 1/32 ของความถี่ออสซิลเลเตอร์มีสูตรเป็น

อัตราบ๊อดในโหมด 2 = (2SMOD / 64) * Oscillator Frequency

- ใน MCS-51 อัตราบ๊อดในโหมด 1 และ 3 ถูกกำหนดได้ด้วยอัตราโอเวอร์โฟลว์ที่เกิดขึ้นจากการกำหนดค่าในรีจีสเตอร์ TH1 ของตัวจับเวลา 1 ส่วนใน 8052 อัตราบ๊อดเหล่านี้สามารถคำนาณได้จากตัวจับเวลา 1 หรือตัวจับเวลา 2 หรือใช้ทั้งสองตัวโดยตัวหนึ่งสำหรับส่งและอีกตัวสำหรับรับ

- การใช้ตัวจับเวลา 1 เป็นตัวสร้างอัตราบ๊อด เมื่อใช้ตัวจับเวลา 1 เป็นตัวสร้างอัตราบ๊อด อัตราบ๊อดในโหมด 1 และ 3 จะถูกคำนวณด้วยอัตราโอเวอร์โฟลว์ที่เกิดขึ้นในตัวจับเวลา 1 และค่าบิตในSMODซึ่งสูตรการคำนวณเป็นดังนี้

อัตราบ๊อดในโมหด 1,3 = (2 SMOD / 32) * Timer1 Overflow Rate

การอินเทอร์รัพต์ตัวจับเวลา 1 ควรที่จะดีสเอเบิ้ลในการใช้งานแบบนี้ ตัวจับเวลาในตัวมันเองสามารถที่จะถูกกำหนดให้ใช้เป็นตัวจับเวลาหรือตัวนับการทำงานในโหมด 3 ในการใช้งานลักษณะนี้มันจะถูกกำหนดให้เป็นตัวจับเวลาในโหมดแบบบรรจุอัตโนมัติ (โดยตั้งให้ HIGH NIBBLE ของ TMOD = 0010B)ในกรณีนี้อัตราบ๊อดคำนวณได้ดังสูตร

อัตราบ๊อดในโหมด 1,3 = (2 SMOD / 32) * (Oscillator Frequency / 12*(256-TH1))

			TIMER1		
BAUD RATE	fosc	SMOD	C/T	MODE	RELOAD VALUE
MODE 0 MAX. 1MHz	12 MHz	X	X	X	X
MODE 2 MAX. 375K	12 MHz	1	X	X	X
MODES 1,3 62.5K	12 MHz	1	0	2	FFH
19.2K	11.059MHz	1	0	2	FDH
9.6K	11.059MHz	0	0	2	FDH
4.8K	11.059MHz	0	0	2	FAH
2.4K	11.059MHz	0	0	2	F4H
1.2K	11.059MHz	0	0	2	E8H
137.5	11.059MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

ตารางที่ 2.13 เป็นรายการอัตราบ๊อดที่ใช้ตัวจับเวลา 1

ในอัตราการบ๊อดมีค่าต่ำมาก ก็สามารถที่จะทำได้โดยการตั้งตัวจับเวลา 1 ให้สามารถรับการอินเทอร์รัพต์ได้ และกำหนดให้ตัวจับเวลาทำงานเป็น 16 บิต (โดยตั้งค่า HIGH NIBBLE ของ TMOD = 0001B) และใช้ตัวจับเวลา 1 ให้ทำการอินเทอร์รัพต์เมื่อเกิดโอเวอร์โฟลว์ และบรรจุค่า 16 บิตไปใหม่ ในกรณีนี้ต้องการอินเทอร์รัพต์ที่ตัวจับเวลา 1 จึงให้ IE.3 = 1 ในกรณีถ้าตัวจับเวลา 1 กำลังทำงานที่บิต C/T = 0 อัตราการนับเป็น 1/12 ของความถี่ออสซิลเลเตอร์ ถ้าตัวจับเวลาทำงานที่บิต C/T = 1 อัตราการนับจะใช้ความถี่ภายนอกที่ส่งเข้ามา ซึ่งจะมีความถี่สูงสุดที่จะใช้ได้คือ 1/24 ของความถี่ออสซิลเลเตอร์

ตารางที่ 2.13 เป็นรายการอัตราบ๊อดที่ใช้กันทั่วไป และค่าต่าง ๆ ที่จะใส่ในตัวจับเวลา 1 แสดงแผนภูมิฟังก์ชันของพอร์ตอนุกรมในโหมด 3 จากการคำนวณ เพื่อให้ได้อัตราบ๊อดตามกำหนด

บทที่ 3

การออกแบบฮาร์ดแวร์และซอร์ฟแวร์ที่ใช้ในการควบคุมอีมูเลเตอร์

3.1 ส่วนประกอบและการออกแบบทางด้านฮาร์ดแวร์

หลักการทำงานที่สำคัญของฮาร์ดแวร์ที่ใช้ในการสร้างอีมูเลเตอร์ คือ จะต้องให้เอาท์พุตของอีมูเลเตอร์มีการแสดงสถานะทางลอจิก (logic) ได้เหมือนกับ ไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ ตระกูล MCS-51 ได้จริงในการทำงานแต่ละคำสั่ง ดังนั้นการในการออกแบบทางฮาร์ดแวร์นี้จึงต้องศึกษาถึงลักษณะของสัญญาณต่างๆ ของไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ ตระกูล MCS-51 และพอจะจำแนกคุณลักษณะของขาสัญญาณได้ 4 ลักษณะดังนี้

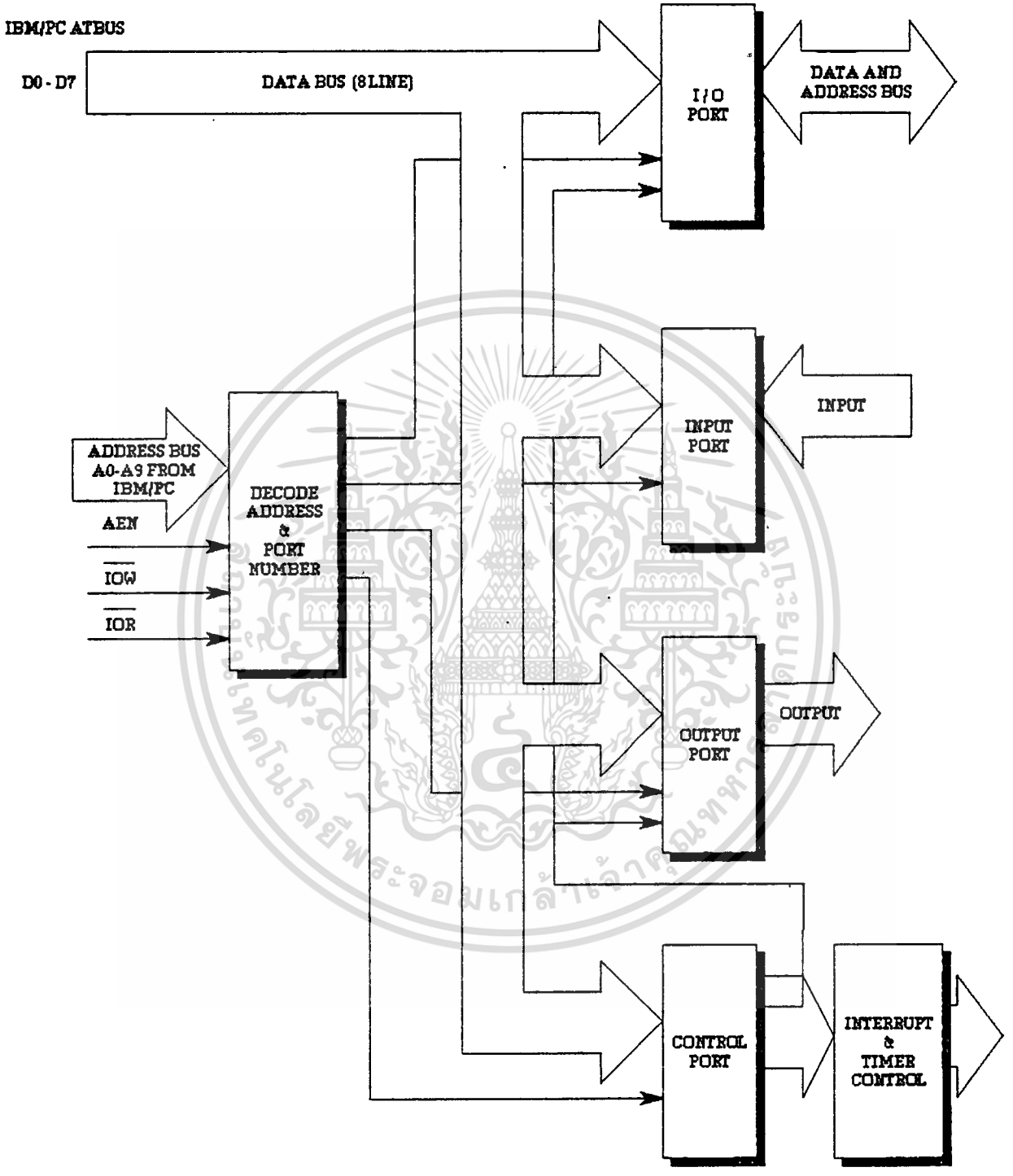
1. สัญญานที่มีลักษณะที่มีการส่งและรับ (bidirectional)
2. สัญญานที่มีลักษณะเป็นการส่งออกอย่างเดียว (output)
3. สัญญานที่มีลักษณะเป็นการรับอย่างเดียว (input)
4. สัญญาณการสื่อสารพอร์ทอนุกรม (Seriel port)

จากลักษณะต่าง ๆ ประเภทของสัญญานที่เกิดขึ้นในแต่ละขาของไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ ตระกูล MCS-51 ตัวจริง และขาต่างๆ นี้จะต้องควบคุมให้มีการทำงานเป็นไปตามคำสั่งต่างๆตามชุดคำสั่งนั้น ในที่นี้จะใช้ไมโครคอมพิวเตอร์เป็นตัวควบคุมการทำงานของระบบ เพื่อที่จะทำให้เข้าใจได้ดียิ่งขึ้นขอให้พิจารณาจากรูปที่ 3.1 ซึ่งแสดงถึงบล็อกไดอะแกรมของอีมูเลเตอร์ และจากบล็อกไดอะแกรมนี้ก็ได้นำไปสร้างเป็นวงจรใช้งานจริง ดังรูปที่ 3.2

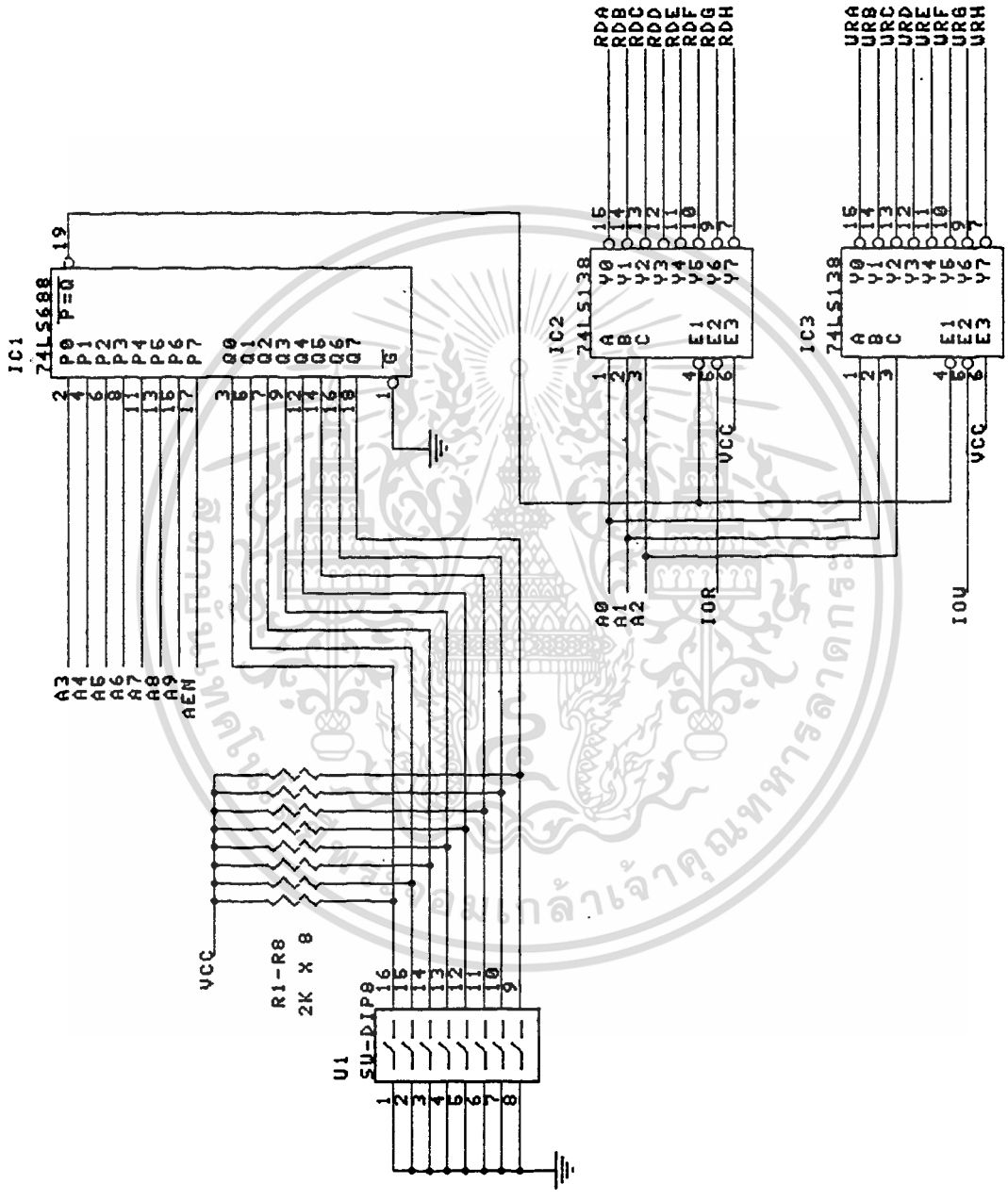
3.2 รายละเอียดของการออกแบบแลวงจรในแต่ละส่วน

3.2.1 วงจรที่ใช้ในการถอดรหัสหมายเลขพอร์ต (Decode port number)

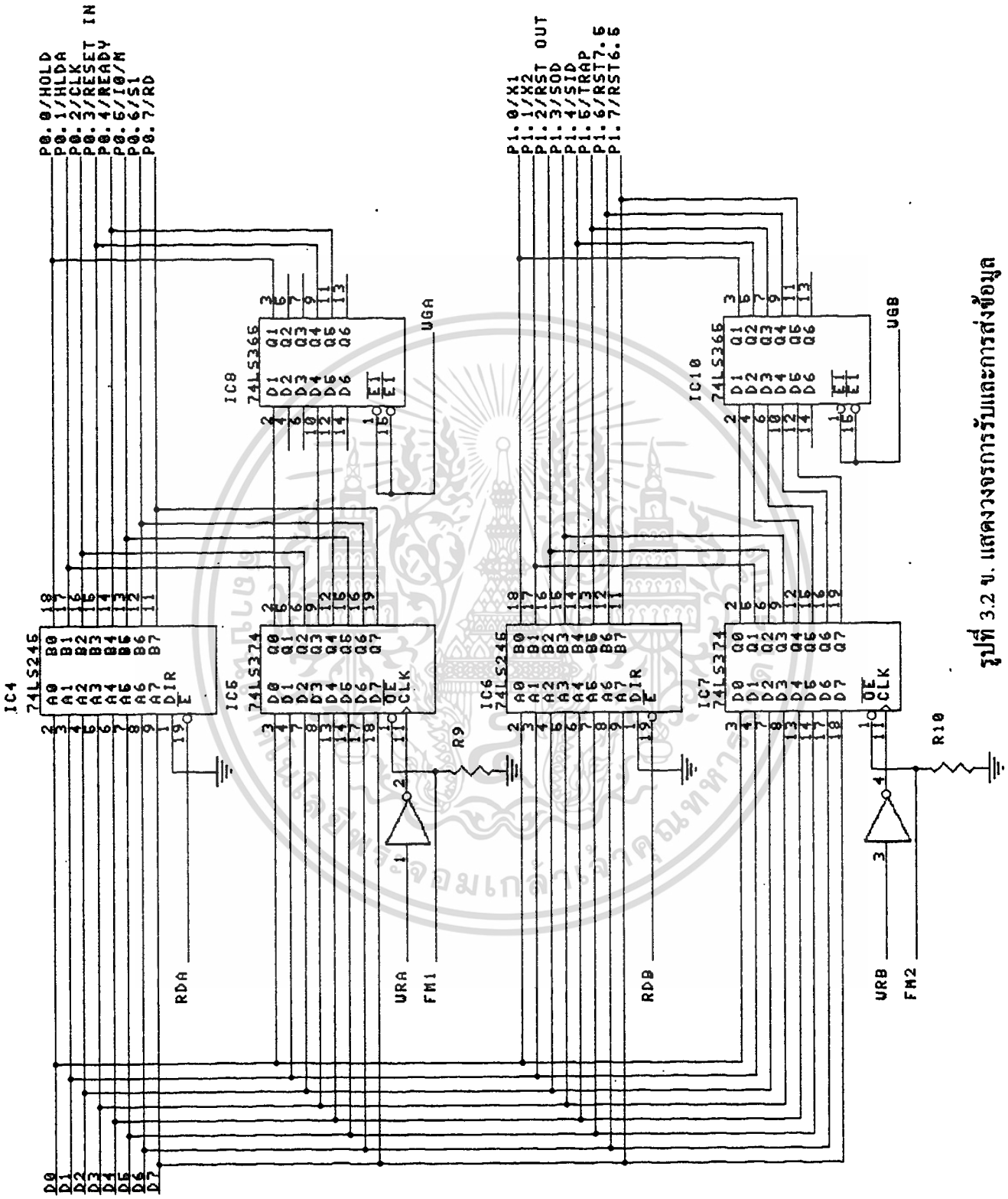
หน้าที่ของวงจรในส่วนนี้ก็คือ เป็นส่วนที่ใช้ในการถอดรหัสหมายเลขพอร์ตที่จะใช้ในการติดต่อและควบคุมระหว่างไมโครคอมพิวเตอร์กับการ์ดอีมูเลเตอร์ที่สร้างขึ้นโดยสัญญาณที่ใช้ในการถอดรหัสจะใช้สัญญาณแอดเดรส AEN , IOR และ IOW จากไมโครคอมพิวเตอร์ซึ่งจากวงจรแสดงให้เห็นใน รูปที่ 3.2 ก. ซึ่งจะอธิบายการทำงานโดยสังเขปดังนี้



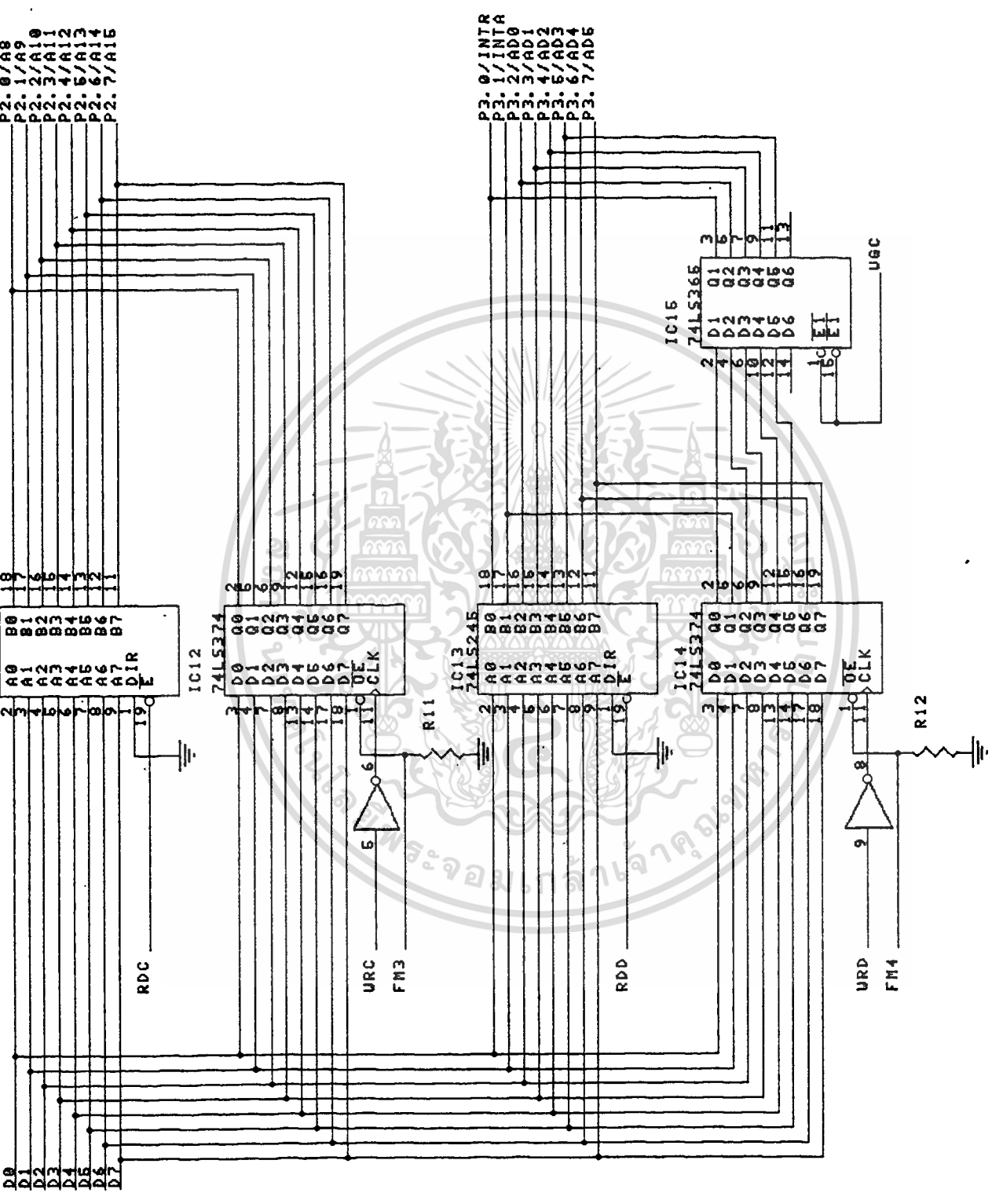
รูปที่ 3.1 แสดงบล็อกไดอะแกรมของอีมูเลเตอร์สาหรับ ไมโครโปรเซสเซอร์ เบอร์ 8085 และ ไมโครคอนโทรลเลอร์ ตระกูล MCS-51



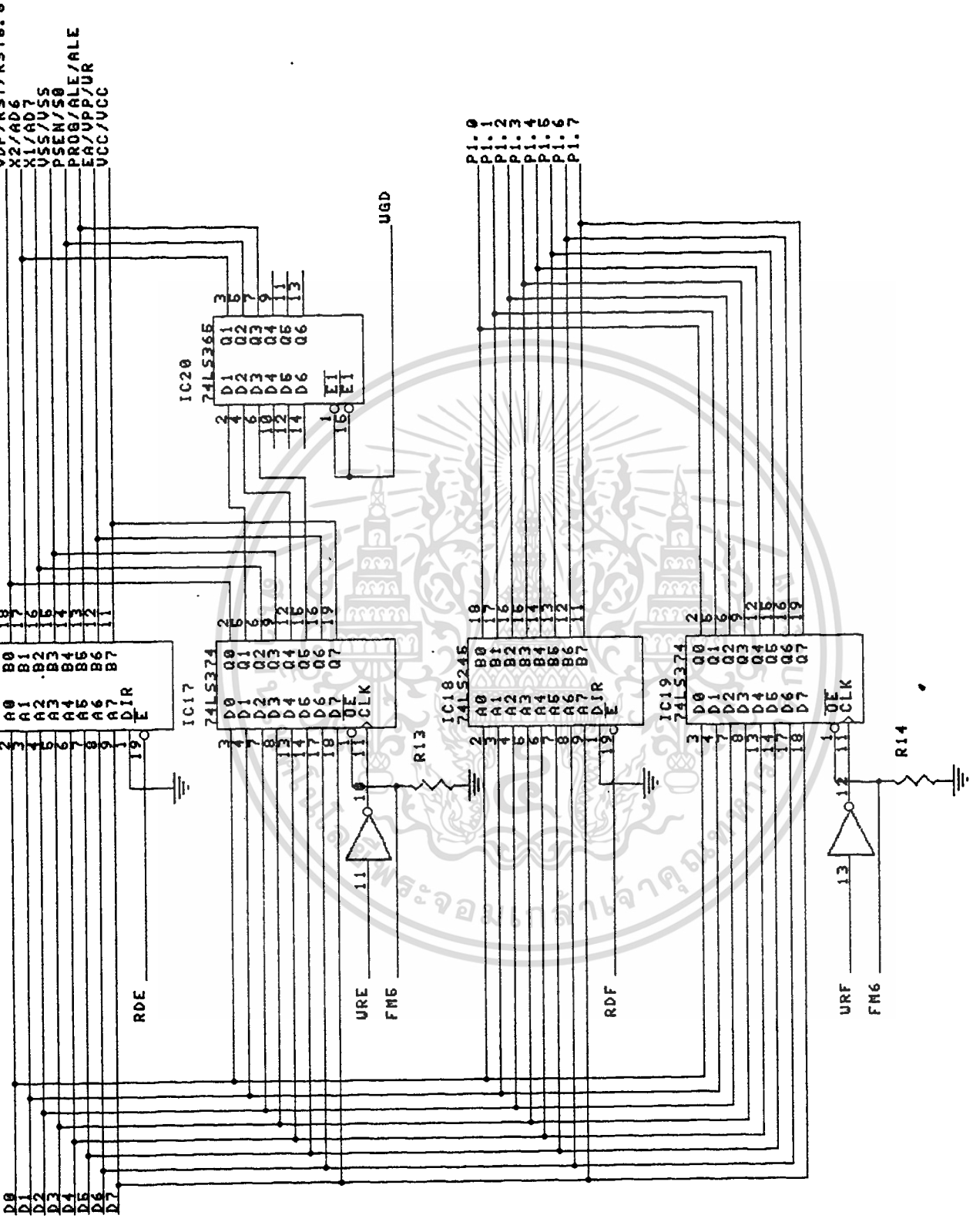
รูปที่ 3.2 ก. แสดงวงจรถอดรหัสหมายเลขพอร์ท



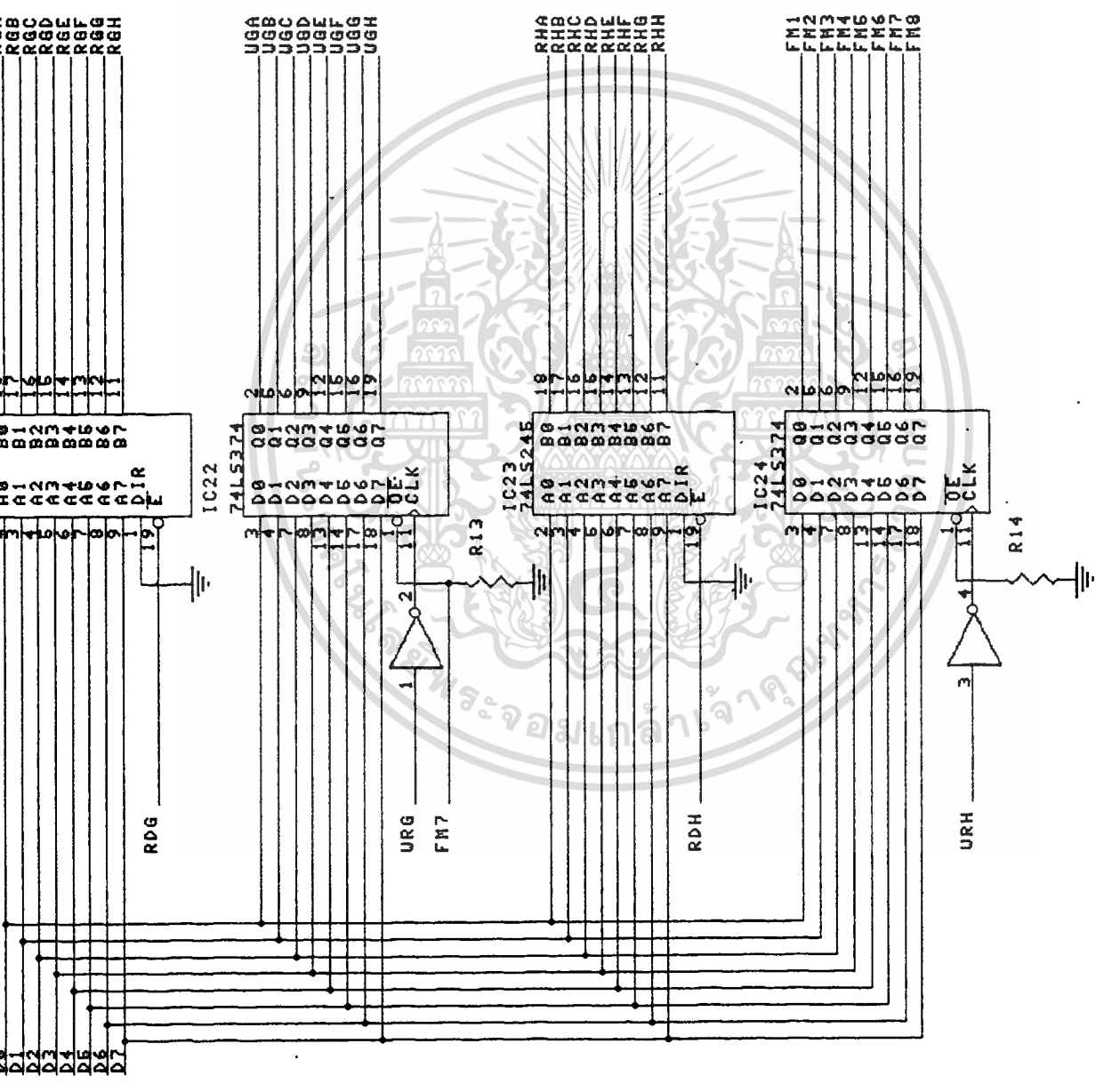
รูปที่ 3.2 ข. แสดงวงจรการรับและการส่งข้อมูล



รูปที่ 3.2 ค. แสดงวงจรการรับและการส่งข้อมูล



รูปที่ 3.2 ง. แสดงวงจรการรับและการส่งข้อมูล



รูปที่ 3.2 จ. แสดงวงจรการรับและการส่งข้อมูล

วงจรในส่วนนี้จะใช้ไอซี (IC) 74LS688 ทำหน้าที่ในการเปรียบเทียบค่าอินพุตจำนวน 2 ชุด ที่ถูกส่งเข้ามาทางขา P_0-P_7 และขา Q_0-Q_7 ถ้าอินพุตทั้ง 2 ชุด นี้เท่ากันแล้วเอาท์พุตที่ขา P=Q (19) จะให้เอาท์พุตเป็นลอจิก "0" จากวงจรขา P_0-P_7 ของ IC 74LS688 จะต่อกับขาแอดเดรสที่บิต A_3-A_9 และขา AEN ในขณะที่ขา Q_0-Q_7 ต่อกับความต้านทานที่ทำหน้าที่รักษาระดับแรงดันให้เป็นลอจิก "1" ไว้ในกรณีที่ไม่มีอินพุตใดๆเข้ามา และที่ขา Q_0-Q_7 นี้จะต่อกับปลายข้างหนึ่ง ของ DIP Switch ด้วย ส่วนปลายอีกข้างหนึ่งของ DIP Switch นั้น จะต่อลงกราวด์ไว้ดังนั้นถ้าเราทำการ "ON" DIP Switch ที่ต่อเข้ากับขาใด อยู่ขานั้นก็จะได้รับลอจิก "0" ในขณะที่ถ้า DIP Switch ที่ต่อกับขาใดถูก "OFF" ขานั้นก็จะได้รับลอจิก "1" และเนื่องจากอิน-พุตที่ขา P_0-P_7 ต้องเท่ากับอินพุตที่ขา Q_0-Q_7 ดังนั้นถ้าเราเปลี่ยนแปลงการตั้งตำแหน่งของDip Switch เหล่านี้ก็จะทำให้แอดเดรสบิตที่ A_3-A_9 ซึ่งต่อกับขา P_0-P_6 นั้น ต้องเปลี่ยนแปลงตามไปด้วยจึงจะทำให้เอาท์พุตของ 74LS688 ที่ขา 19 อยู่ในสภาวะแอกทีฟได้ทำให้เราสามารถนำหมายเลขหมายเลขพอร์ตที่ได้จากการถอดรหัสไปใช้งาน ในปริญญานิพนธ์ฉบับนี้เลือกใช้หมายเลขพอร์ตตั้งแต่พอร์ต 300H-307H ซึ่งเอาท์พุตที่ได้จากการถอดรหัสจะถูกนำมาใช้ในการกำหนดการอีนาเบิล 74LS138 ร่วมกับสัญญาณ IOR และ IOWจากไมโครคอมพิวเตอร์ตารางที่ 3.1 จะแสดงให้เห็นถึงสัญลักษณ์ที่กำหนดหมายเลขพอร์ต

สัญญลักษณ์	หมายเลขพอร์ต
PORT A	300H
PORT B	301H
PORT C	302H
PORT D	303H
PORT E	304H
PORT F	305H
PORT G	306H
PORT H	307H

ตารางที่ 3.1 สัญญลักษณ์และการกำหนดหมายเลขพอร์ต

3.2.2. วงจรที่ใช้เป็นส่วนที่สามารถทำการรับและส่ง

จากบล็อกไดอะแกรม ในรูปที่ 3.1 นั้น ในส่วนนี้ทำหน้าที่ในการรับส่งข้อมูลในพอร์ตเดียวกัน ซึ่งจะใช้ทำหน้าที่เป็นขาสัญญาณที่ใช้เป็นพอร์ต อินพุต เอาท์พุต หรือ พอร์ตที่ใช้เป็นบัสข้อมูลและบัสแอดเดรส ซึ่งในกรณีที่ใช้งานเป็นบัสข้อมูลและบัสแอดเดรส ในช่วงแรกที่ไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ ตระกูล MCS-51 ต้องทำการติดต่อกับ

อุปกรณ์ภายนอก มันจะส่งแอดเดรสไบต์ต่ำออกมาทางขาสัญญาณ AD0-AD7 เพื่อใช้ในการชี้ตำแหน่งของหน่วยความจำที่ต้องติดต่อ ซึ่งขณะเดียวกันไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ MCS-51 จะส่งสัญญาณ ALE (Address latch enable) เพื่อแสดงว่าได้ทำการส่งแอดเดรสไบต์ต่ำออกมาแล้ว อุปกรณ์ภายนอกจะต้องทำการแลทช์แอดเดรสไบต์ต่ำไว้ และช่วงเวลาต่อมาเมื่อวงจรภายนอกแลทช์แอดเดรสไบต์ต่ำไว้แล้ว จากนั้นขาสัญญาน AD0-AD7 จะใช้สำหรับรับ-ส่งข้อมูล ที่ต้องการอ่านหรือต้องการเขียนแทน

สำหรับวงจรจริงที่ใช้ในการ์ดอินเตอร์เฟสเตอร์ ที่ทำหน้าที่แทนขาพอร์ตที่สามารถทำการรับส่งข้อมูลนี้ ประกอบไปด้วยชิพสนับสนุนที่เป็นอินพุตและเอาท์พุต อย่างละหนึ่งตัว คือ ตัวที่ใช้เป็นอินพุตจะใช้ IC 74LS245 ซึ่งทำหน้าที่เป็น Octal Bus Transceivers with 3-state Outputs โดยใช้ขาสัญญาน RDA ที่ได้จากการ Decode ของ 74LS138 เป็นสัญญาน Enable ให้ทำงานส่วนชิพสนับสนุน ตัวที่ทำหน้าที่เป็นเอาท์พุต ก็คือ 74LS374 ซึ่งทำหน้าที่เป็น Octal D-Type Transparent Latch and Edge-Triggered Flip-Flops เหตุที่ต้องใช้พอร์ตอินพุตและเอาท์พุตแยกกัน เนื่องจากเหตุผลที่กล่าวไว้ในตอนต้นคือ ขาสัญญานที่ใช้เป็น แอดเดรส และ ดาต้า บัสนั้นในขณะที่ส่งแอดเดรสไบต์ต่ำออกไปจะต้องคงค่าไว้จนกว่าวงจรภายนอกจะแลทช์ ค่าแอด-เดรสไว้ก่อนแล้วจึงจะทำให้พอร์ตอยู่ในสภาวะอิมพีแดนซ์สูง (high impedance) หลังจากนั้นจึงเป็นขบวนการรับ หรือส่งข้อมูลต่อไป ซึ่งขาสัญญานที่ใช้ ในการควบคุมการอีนาเบิล 74LS374 นี้จะใช้สัญญาณที่ได้จากการถอดรหัสจากสัญญาณ IOW ส่วนสัญญานจากพอร์ตควบคุม จะใช้เป็นตัวควบคุมให้ 74LS374 อยู่ในสภาวะอิมพี-แดนซ์สูง

3.2.3. วงจรที่ใช้เป็นส่วนอินพุต

ในส่วนที่เป็นอินพุตนี้จะใช้ไอซีเบอร์ 74LS244 ซึ่งทำหน้าที่เป็น Octal Buffers / Line Drivers/Line Receivers และ สัญญาณที่ใช้ควบคุม จะใช้สัญญาณที่ได้จากการถอดรหัส จากสัญญาณ IOR

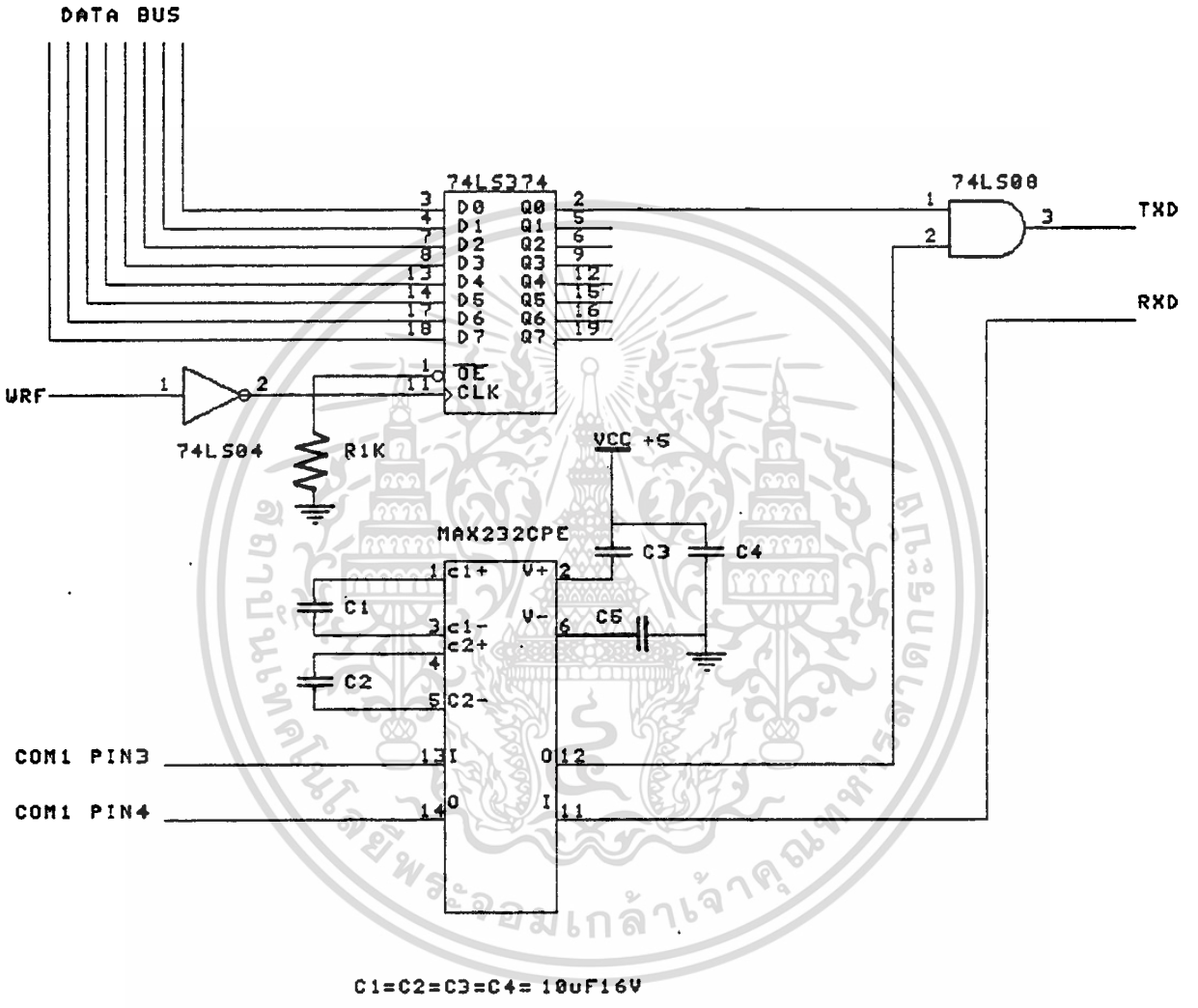
3.2.4. วงจรที่ใช้เป็นส่วนเอาท์พุต (Output port)

ในส่วนของบล็อกไดอะแกรมจะแยกเป็น 2 ส่วน คือ ส่วนที่ใช้ทำหน้าที่การส่งสัญญานต่างๆออกทางขาสัญญานที่เป็น เอาท์พุตของไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ MCS-51 และส่วนที่ใช้เป็นเป็นส่วนควบคุมภายใน ตัวอย่างเช่น ในขณะที่ไมโครโปรเซสเซอร์ 8085 กระทำคำสั่ง HALT ขาเอาท์พุตทั้งหมดควรอยู่ในสภาวะอิมพแดนซ์สูงโดยพอร์ตควบคุมจะทำหน้าที่ทำให้เอาท์พุตทั้งหมดอยู่ในสภาวะอิมพีแดนซ์สูง

3.2.5 สัญญาณการสื่อสารพอร์ทอนุกรม (Seriel port)

จากรูปที่ 3.2ฉการรับส่งข้อมูลอนุกรมจะใช้พอร์ทสื่อการอนุกรมจากคอมพิวเตอร์ (COM 1) มาต่อการ์ดอินเตอร์เฟสเตอร์ที่จุด J1 เพื่อเปลี่ยนแรงดัน 12 โวลต์ เป็นแรงดัน 5 โวลต์ หรือ

เปลี่ยนแรงดัน 5 โวลต์ เป็นแรงดัน 12 โวลต์ โดยใช้ ไอซีเบอร์ MAX232CPE จากนั้นมาเข้าแอนเกต 7408 เพื่อควบคุมการรับส่งจากซอฟต์แวร์



รูปที่ 3.2 ฉ แสดงการรับส่งข้อมูลอนุกรม

3.3 โปรแกรมที่ใช้ในการติดต่อและควบคุมอีมูเลเตอร์

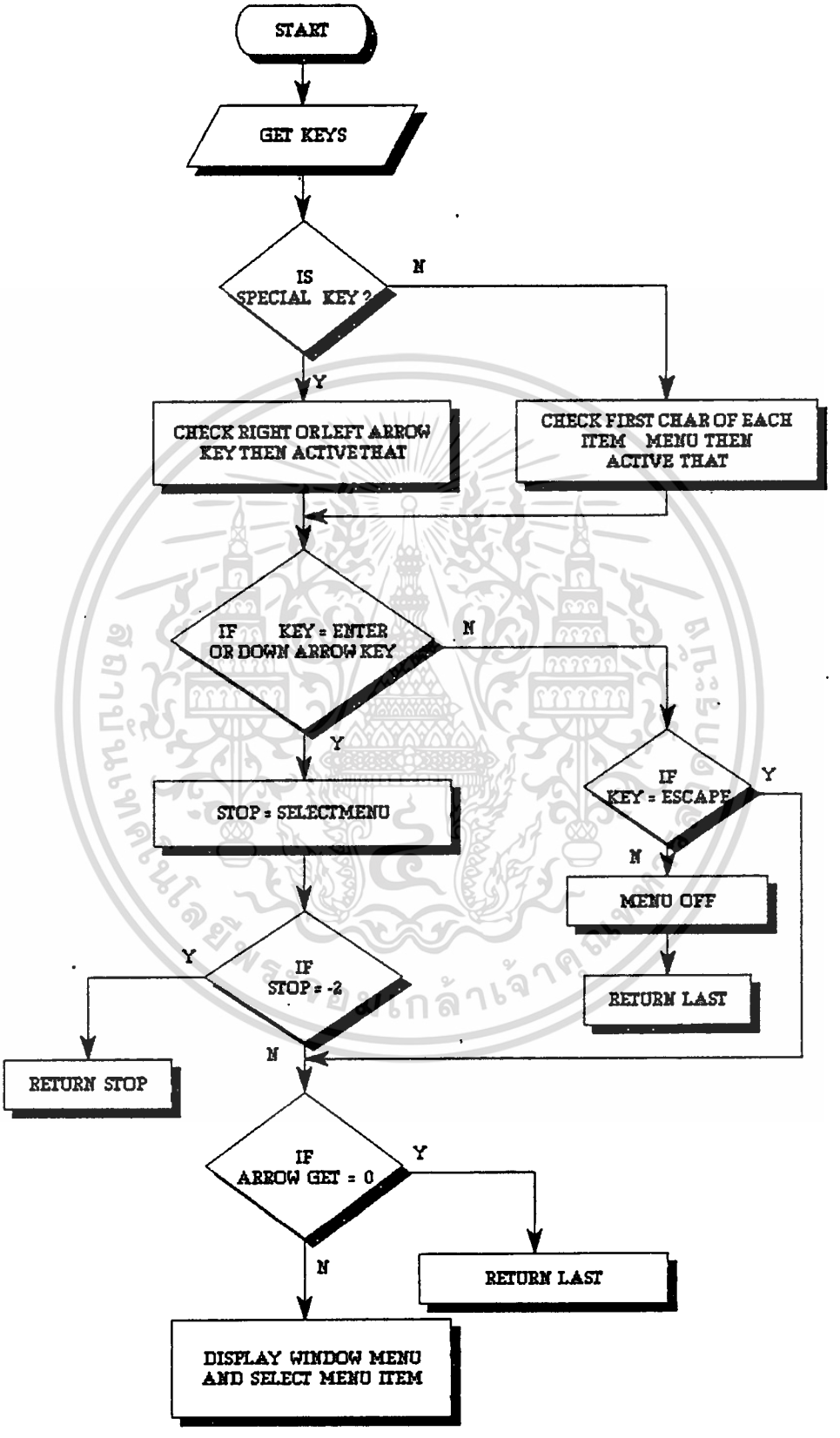
จุดประสงค์หลักของการเขียนโปรแกรมอีมูเลเตอร์นี้ก็เพื่อนำไปใช้ในการทดสอบและทดลองระบบควบคุมอัตโนมัติที่มีไมโครโปรเซสเซอร์ 8085 หรือ ไมโครคอนโทรลเลอร์ MCS-51 เป็นหน่วยประมวลผลกลาง โดยใช้การ์ดอีมูเลเตอร์เป็นตัวกลางในการติดต่อระหว่าง

ระบบที่ต้องการทดสอบกับไมโครคอมพิวเตอร์ ซึ่งผู้ใช้สามารถควบคุมโดยใช้โปรแกรมที่ผู้ใช้เขียนขึ้นมา โดยแนวทางในการเขียนโปรแกรมนี้จะเป็นไปในทางที่จะทำให้ผู้ที่นำโปรแกรมไปใช้สามารถที่จะใช้ได้ง่าย และสะดวกต่อการใช้งาน สำหรับภาษาที่ใช้เขียนโปรแกรมนี้จะใช้ภาษาซีในการเขียนโปรแกรม ซึ่งโปรแกรมที่ใช้ติดต่อ และ ควบคุมอีมูเลเตอร์จะแบ่งแยกออกเป็น 2 โปรแกรมหลัก คือ ส่วนของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 (ไฟล์ EMU8085.C) และอีกส่วนหนึ่งเป็นส่วนของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 (ไฟล์ EMUMCS.C) ซึ่งโครงสร้างของโปรแกรมหลักทั้งสองจะมีลักษณะคล้ายกัน แต่จะแตกต่างกันบ้างในส่วนของแต่ละเมนูย่อยซึ่งแนวความคิดในการเขียนโปรแกรมนี้จะแบ่งออก เป็น 2 ส่วนคือ ส่วนแรกจะเป็นส่วนของระบบเมนูหลักและอีกส่วนหนึ่งจะเป็นระบบของเมนูย่อย ดังนั้นในขั้นตอนแรกในการออกแบบก็คือ จะต้องออกแบบโครงสร้างของระบบเมนูออกมาก่อน หน้าที่หลักของโปรแกรมในส่วนนี้จะทำ หน้าที่ในการจัดการ และ ควบคุมการเลือกเมนูต่างๆ โดยการเลือกเมนูในขั้นนี้จะทำให้มีการเรียกฟังก์ชันย่อยจะแยกอธิบายโดยโฟลว์ชาร์ทการทำงานของแต่ละเมนูในแต่ละหัวข้อ

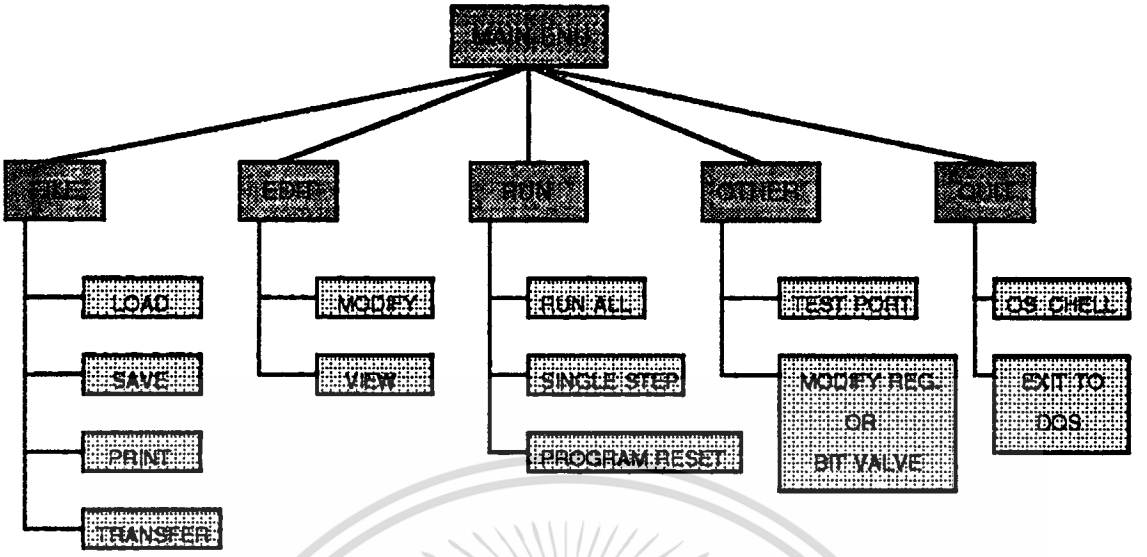
3.3.1. โครงสร้างของโปรแกรมเมนูและส่วนที่จัดการเกี่ยวกับการเลือกเมนู

หน้าที่หลักของโปรแกรมในส่วนนี้คือ จะทำการควบคุมการใช้งานในการเลือกหัวข้อเมนูย่อยต่างๆ ซึ่งตัวโปรแกรมทั้งหมดนี้ ได้รวมเอาฟังก์ชันต่างๆที่เกี่ยวกับวินโดว์เมนู เป็นต้น ซึ่งจากรูปที่ 3.3 จะแสดงให้เห็นถึงโฟลว์ชาร์ทของโปรแกรมย่อยที่ใช้ในการเลือกเมนู จากโฟลว์ชาร์ทจะเห็นว่ามีขั้นตอนการทำงานดังนี้เริ่มต้นด้วยการตรวจสอบการกดคีย์ว่ามีการ กดคีย์ใดคีย์หนึ่งหรือไม่ถ้ามีก็จะมีฟังก์ชันที่ทำหน้าที่ตรวจสอบว่าคีย์ที่กดนั้น เป็นคีย์พิเศษหรือไม่ถ้าใช่ก็จะตรวจสอบว่าเป็นคีย์ลูกศรซ้ายหรือขวา แล้วแอกทีฟไปตามหัวข้อเมนูหลักตามทิศทางของคีย์ลูกศรนั้นแต่ถ้าคีย์ที่กดนั้นไม่ใช่คีย์พิเศษก็ให้ตรวจสอบว่าตัวอักษรที่กดนั้นตรงกับตัวอักษรตัวแรกของแต่ละหัวข้อเมนูหรือไม่ถ้าใช่ก็ให้ทำการแอกทีฟตามหัวข้อเมนูนั้นหลังจากนั้นก็ทำการตรวจสอบว่าคีย์ที่กดคีย์ต่อมานั้นเป็นคีย์ Enter หรือคีย์ลูกศรลง (Down arrow) หรือไม่ ถ้าใช่ก็จะมี Pull down menu ของหัวข้อเมนูย่อยนั้นขึ้นมาทำการเลือกแต่ถ้าไม่ใช่คีย์ทั้งสอง ก็จะตรวจสอบว่าเป็นคีย์ Esc หรือไม่ถ้าใช่ก็ใช้ยกเลิกการแอกทีฟที่หัวข้อเมนูย่อยนั้นแล้วก็กลับสู่โปรแกรมที่ทำการเรียกใช้โปรแกรมย่อยนี้มาพร้อมส่งค่าให้กับตัวแปร Stop ซึ่งจะใช้ในการตรวจสอบการออกจากโปรแกรมเพื่อเข้าสู่ระบบปฏิบัติการ

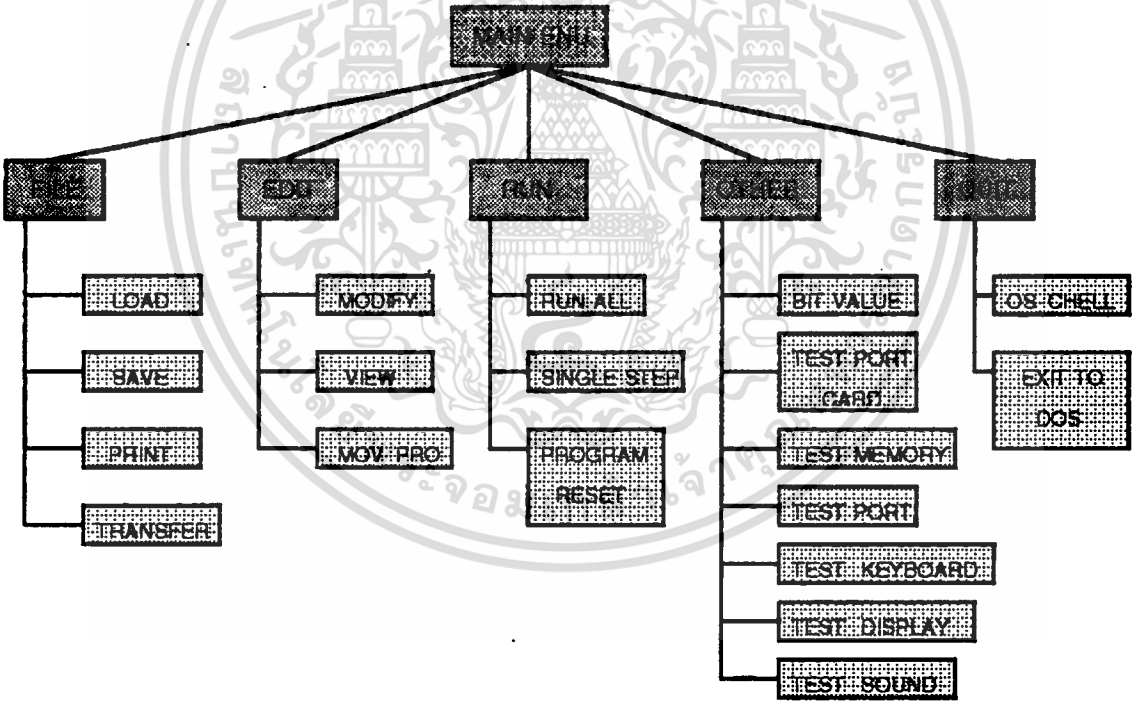
จากรูปที่ 3.4 และรูปที่ 3.5 เป็นการแสดงโครงสร้างของระบบเมนูของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 ตามลำดับ ซึ่งจะมีเมนูหลักจำนวน 5 หัวข้อเมนู คือ เมนู File , เมนู Edit , เมนู Run , เมนู Other และ เมนู Quit ซึ่งจะแสดงรายละเอียดและลำดับขั้นตอนการทำงานแต่ละไฟล์จะแยกอธิบายตามโฟลว์ชาร์ทดังนี้



รูปที่ 3.3. แสดงโฟล์วชาร์ทของโปรแกรมที่จัดการเกี่ยวกับการเลือกเมนู



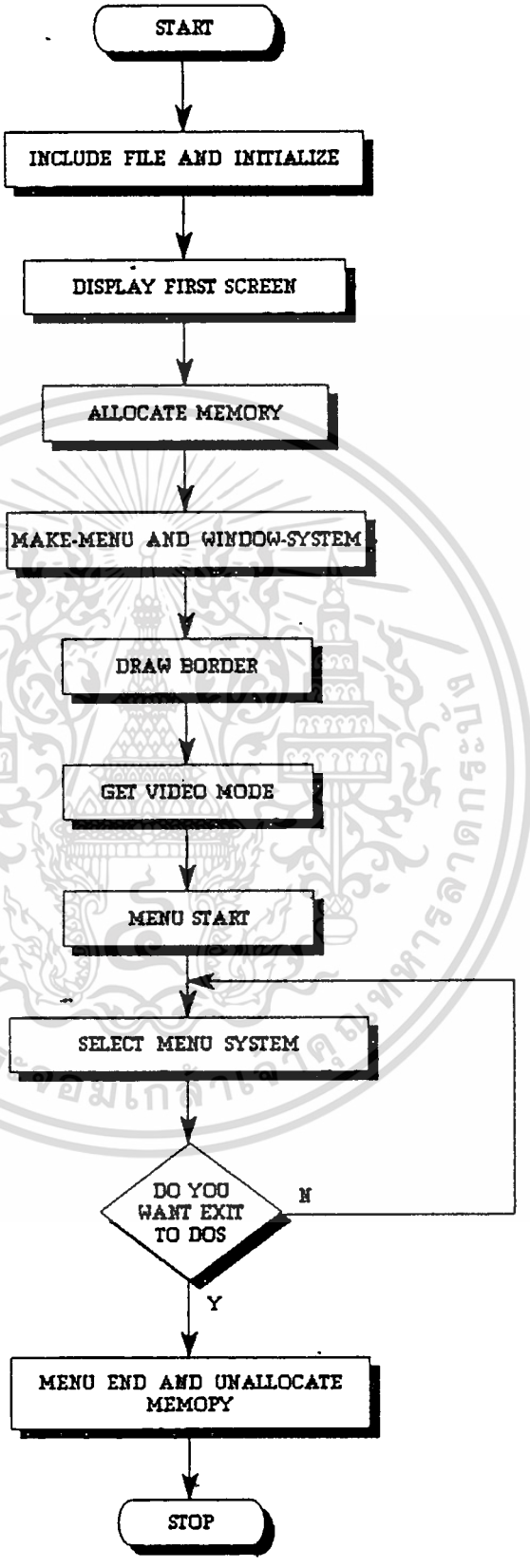
รูปที่ 3.4 แสดงโครงสร้างของระบบเมนูของอีมูเลเตอร์



รูปที่ 3.5 แสดงโครงสร้างของระบบเมนูของอีมูเลเตอร์

3.3.2. โปรแกรมหลัก (Main program)

โปรแกรมหลักนี้จะเป็นโปรแกรมที่จัดการเกี่ยวกับการจองหน่วยความจำมีการจัดการเกี่ยว กับระบบวินโดว์ เมนูสร้างกรอบ จัดการเกี่ยวกับจอภาพ เรียกใช้ฟังก์ชันย่อยเกี่ยวกับ



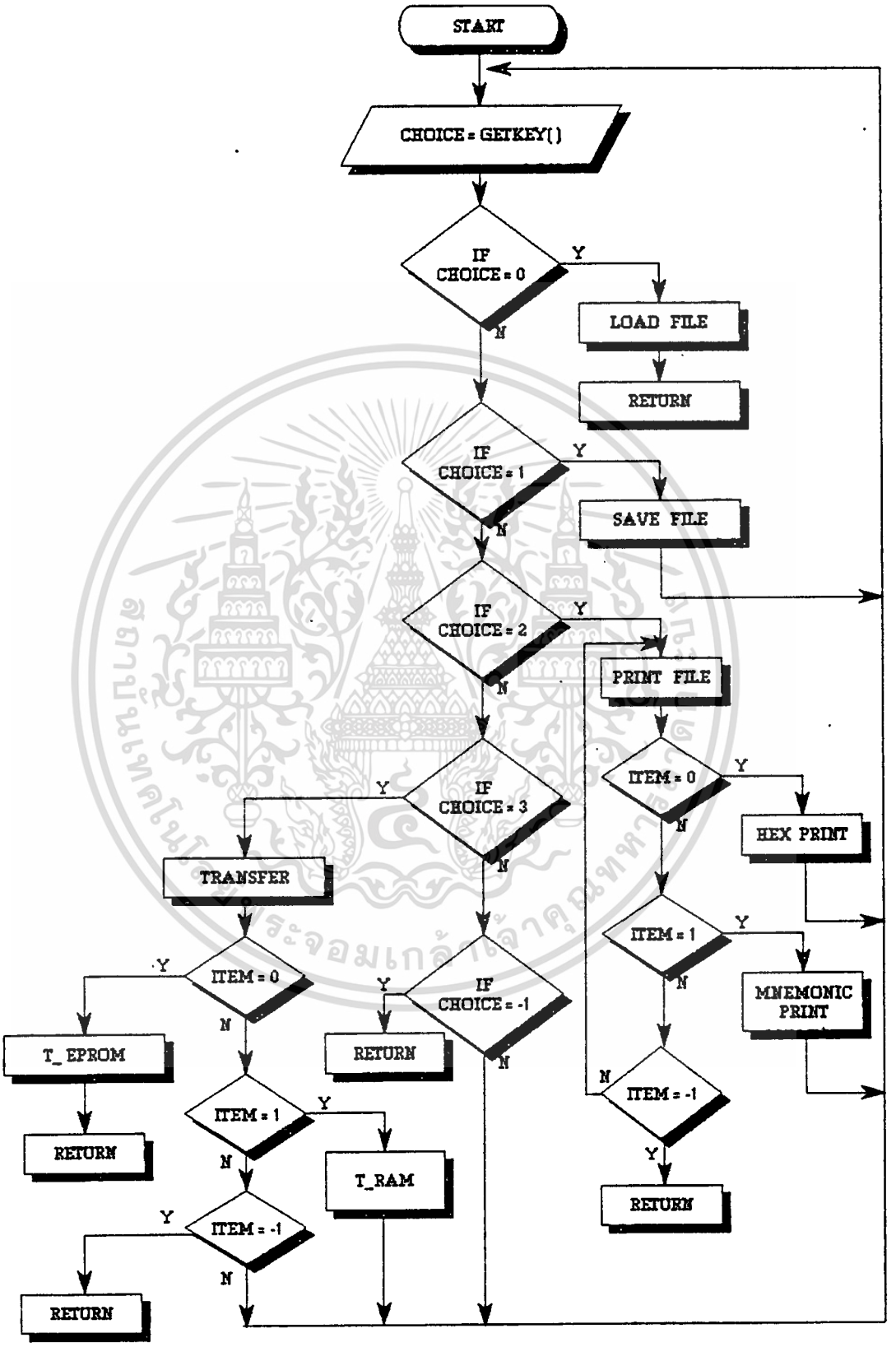
รูปที่ 3.6 แสดงโฟล์วชาร์ทของโปรแกรม EMU8085.C และ EMUMCS.C

เมนู และฟังก์ชันย่อยในการทำการแต่ละเมนู รูปที่ 3.6 จะแสดงโฟลว์ชาร์ทของโปรแกรมของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 ตามลำดับ

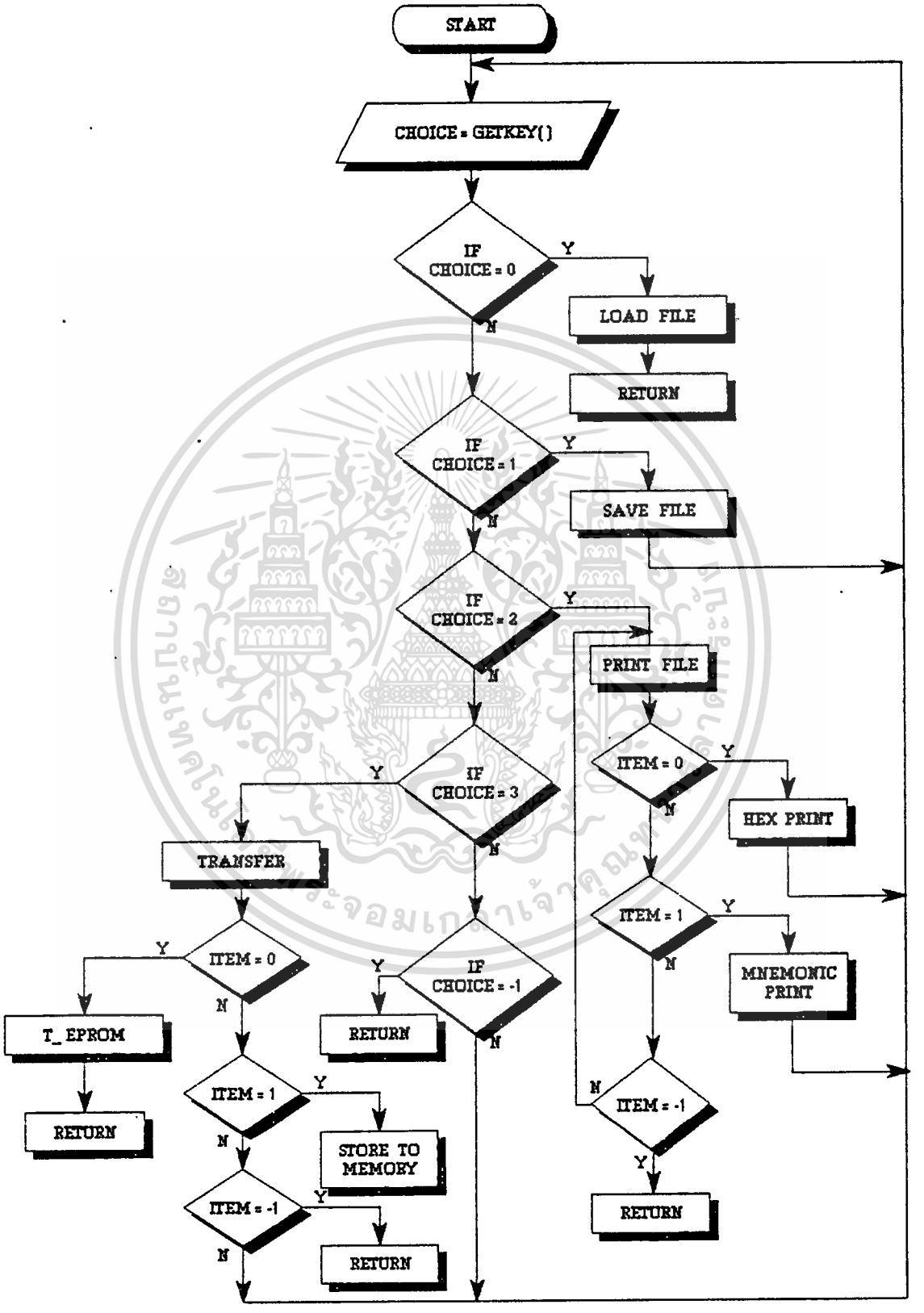
จากโฟลว์ชาร์ทจะเห็นว่าโครงสร้างการทำงานของโปรแกรม EMU8085.C และ EMUMCS.C ซึ่งเป็นโปรแกรมหลักของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 นั้นจะมีลักษณะเหมือนกันคือ ในส่วนแรกจะเป็นการแสดงหน้าจอแรก มีการจองหน่วยความจำจัดการเกี่ยวกับระบบเมนูและระบบวินโดว์ที่ใช้ในโปรแกรมทั้งหมด สร้างกรอบและจัดการเกี่ยวกับจอภาพ ตรวจสอบการกดคีย์ F10 เพื่อเริ่มต้นเข้าสู่การเลือกใช้หัวข้อเมนู หลังจากนั้นจะเป็นการเลือกใช้เมนูตามต้องการ และโปรแกรมหลักจะมีการตรวจสอบการออกจากโปรแกรม เพื่อเข้าสู่ระบบปฏิบัติการของดอสด้วยซึ่งเมื่อต้องการยกเลิกการใช้โปรแกรมก็จะจัดการคืนหน่วยความจำที่ทำการจองไว้ ให้กับระบบไมโครคอมพิวเตอร์ด้วย

3.3.3. โครงสร้างของโปรแกรมเมนู File

เมนูไฟล์นี้จะเป็นการกระทำเกี่ยวกับไฟล์ต่างๆ ซึ่งได้แบ่งหัวข้อเมนูย่อยได้ 4 หัวข้อคือ Load, Save, Print และ Transfer จากรูปที่ 3.7 และรูปที่ 3.8 จะแสดงถึงโฟลว์ชาร์ทของฟังก์ชั่นที่เกี่ยวกับเมนูไฟล์ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 ตามลำดับ ซึ่งจากโฟลว์ชาร์ทจะเห็นว่าโครงสร้างการทำงานของโปรแกรมจะมีลักษณะคล้ายกันแต่จะแตกต่างกันบ้างในส่วนของการทำงานในระดับเมนูย่อยเท่านั้น ซึ่งโปรแกรม ในส่วนนี้จะตรวจสอบการเลือกเมนูย่อยของหัวข้อเมนู File จากนั้นจะส่งค่าของหัวข้อเลือกเมนูย่อยให้กับตัวแปร choice เพื่อทำการตรวจสอบว่า หัวข้อเมนูที่เลือกนั้นตรงกับกรณีใดบ้าง แล้วจึงเรียกฟังก์ชันไปใช้งาน ซึ่งจากโฟลว์ชาร์ทจะเห็นว่าถ้าค่าของ choice เท่ากับ 0 จะเลือกทำหัวข้อเมนูย่อย Load file ซึ่งฟังก์ชันนี้จะทำหน้าที่ในการโหลดไฟล์ข้อมูลจากแผ่นดิสก์เก็ตเข้ามาไว้ในหน่วยความจำของระบบไมโครคอมพิวเตอร์ หลังจากนั้นก็จะออกจากเมนู File แต่ถ้า choice เท่ากับ 1 ก็จะเป็นการเรียกฟังก์ชัน Save file มาทำงาน เพื่อทำการเก็บข้อมูลลงบนแผ่นดิสก์เก็ตเสร็จเรียบร้อยแล้ว ก็จะกลับไปที่ขั้นตอนการเลือกหัวข้อเมนูย่อยในหัวข้อเมนู File และถ้าค่า choice เท่ากับ 2 ก็จะเข้าสู่เมนูที่ทำการพิมพ์ไฟล์ข้อมูลออกทางเครื่องพิมพ์ซึ่งสามารถที่จะเลือกลักษณะของการพิมพ์ได้ คือพิมพ์ในลักษณะเลขฐาน 16 หรือพิมพ์แบบรหัสนีมอนิค (Mnemonic) ส่วนหัวข้อสุดท้าย คือ ถ้าค่าในตัวแปร choice เท่ากับ 3 ก็จะเป็นการเลือกฟังก์ชัน Transfer มาใช้งาน ซึ่งฟังก์ชันนี้จะทำการถ่ายโอนข้อมูลจากหน่วยความจำของระบบที่ต้องการทดลองหรือตรวจสอบและถ้ามีการกดคีย์ Esc จะทำให้ตัวแปร choice เท่ากับ -1 ก็จะเป็นการยกเลิกการเลือกหัวข้อเมนูนี้และกลับไปยังโปรแกรมหลักที่เรียกไฟล์นี้



รูปที่ 3.7 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนูไฟล์สำหรับไมโครโปรเซสเซอร์ 8085



รูปที่ 3.8 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนูโฟล์สำหรับไมโครคอนโทรลเลอร์ MCS-51

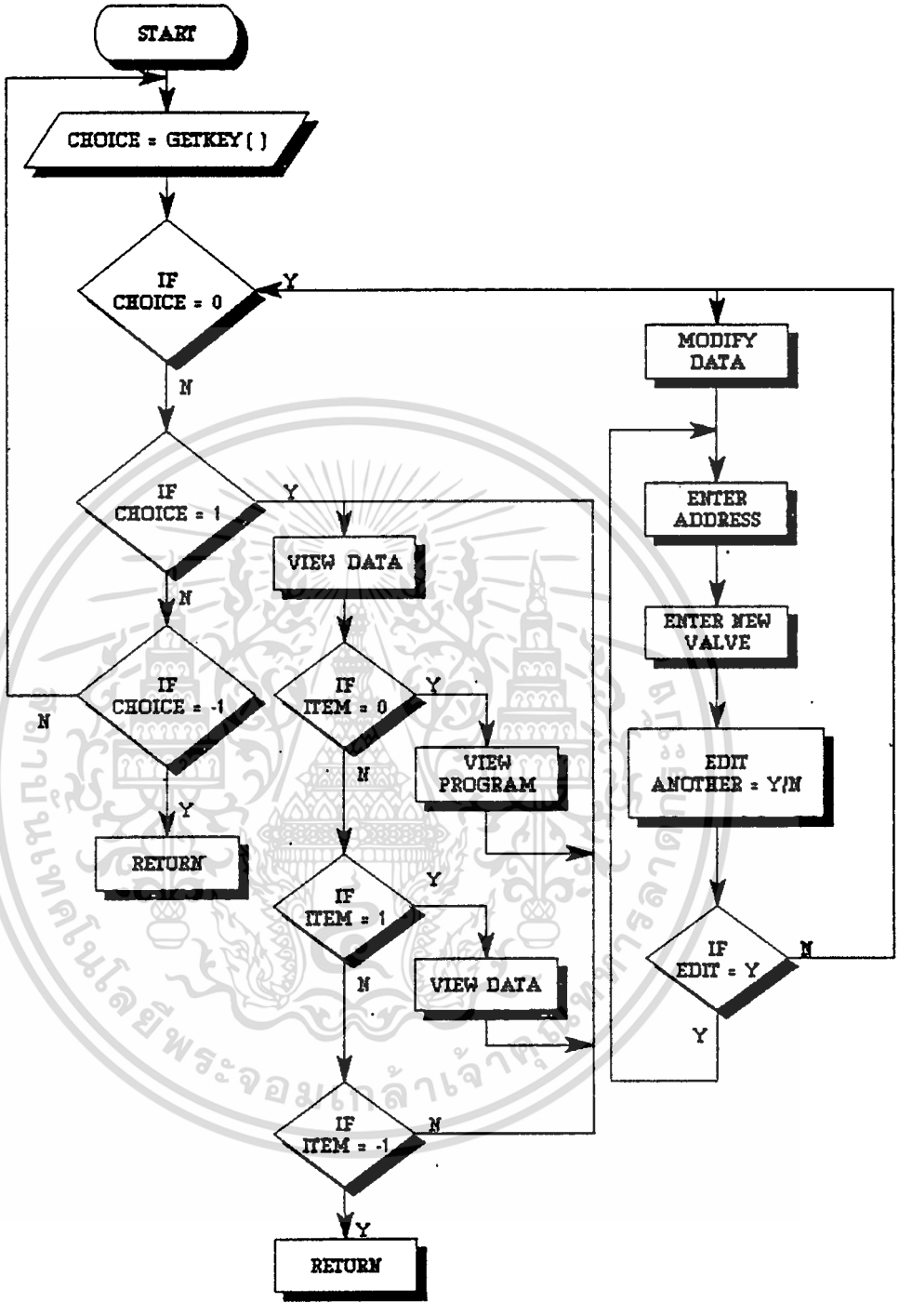
3.3.4. โครงสร้างของโปรแกรมเมนู Edit

หน้าที่หลักของหัวข้อเมนูนี้จะมีอยู่ 2 อย่างคือ การแก้ไขหรือเปลี่ยนแปลงข้อมูลในแอดเดรสต่างๆ และการขอดูค่าของข้อมูลหรือตัวโปรแกรมในแอดเดรสต่างๆตามที่ต้องการ จากรูปที่ 3.9 และรูปที่ 3.10 จะแสดงถึงโฟลว์ชาร์ทของฟังก์ชั่นที่เกี่ยวกับเมนู Edit ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 โครงสร้างของโปรแกรมจะมีลักษณะคล้ายกันแต่จะแตกต่างกันในส่วนของการทำงานในเมนูย่อยบ้างเท่านั้นจากโฟลว์ชาร์ทจะเห็นว่าเริ่มต้นการทำงานก็มีการตรวจคีย์ถ้ามีการเลือกหัวข้อใดหัวข้อหนึ่งก็จะมีการส่งค่าของหัวข้อเมนูนั้นๆไปยังตัวแปร choice ซึ่งก็จะทำการตรวจสอบว่ามีการเลือกฟังก์ชันใดซึ่งในส่วนนี้มีอยู่ 3 หัวข้อเมนูคือ Modify,View และ Mov_pro ซึ่งถ้า choice เท่ากับ 0 ก็จะเป็นการเลือกใช้ฟังก์ชั่น Modify ซึ่งฟังก์ชั่นนี้จะทำการแก้ไขค่าของข้อมูลในรีจิสเตอร์หรือค่าของข้อมูลในหน่วยความจำ ถ้า choice เท่ากับ 1 ก็จะเป็นการเลือกใช้ฟังก์ชัน View ซึ่งในฟังก์ชันนี้จะมีหน้าที่ในการแสดงค่าของข้อมูลตามแอสเดรสที่ต้องการ และ choice เท่ากับ 2 ทำหน้าที่ในการย้ายบล็อกโปรแกรมไปยังตำแหน่งที่กำหนด และเช่นเดียวกันกับฟังก์ชันอื่นถ้าหากว่ามีการกดคีย์ Esc โปรแกรมก็จะกลับไปสู่โปรแกรมหลักที่เรียกมา

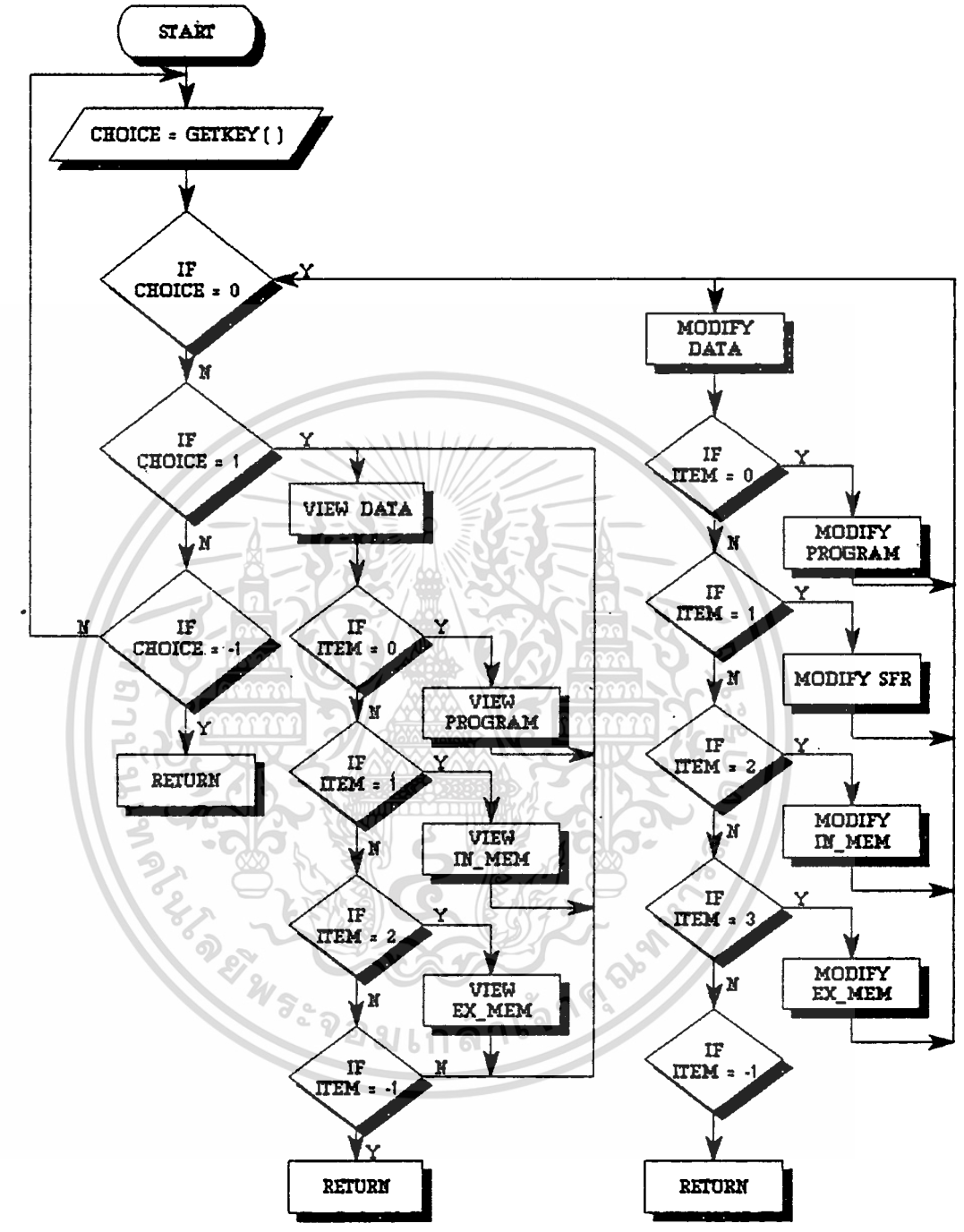
3.3.5. โครงสร้างของโปรแกรมเมนู Run

หน้าที่การทำงานหลักของหัวข้อเมนูนี้จะเป็นหัวใจของอีมูเลเตอร์เพราะใช้เมนูนี้ในการรันโปรแกรมที่ต้องการทดลองหรือตรวจสอบ ซึ่งแบ่งการทำงานออกเป็น 3 หัวข้อเมนูย่อย คือ Program reset, Single step และ Run all ซึ่งจากรูปที่ 3.11 และรูปที่ 3.12 จะแสดงให้เห็นโฟลว์ชาร์ทของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 ตามลำดับ ซึ่งจากโฟลว์ชาร์ทจะเห็นว่า เริ่มต้นการทำงานก็มีการตรวจคีย์ถ้ามีการเลือกหัวข้อใดหัวข้อหนึ่งก็จะมีการส่งค่าของหัวข้อเมนูนั้นๆไปยังตัวแปร choice ซึ่งก็จะทำการตรวจสอบว่ามีการเลือกฟังก์ชันใดเนื่องจากฟังก์ชั่นในโปรแกรมนี้มีความสำคัญในการทำงานของอีมูเลเตอร์ดังนั้นจึงจะอธิบายการทำงานในส่วนของฟังก์ชันอย่างละเอียดดังนี้

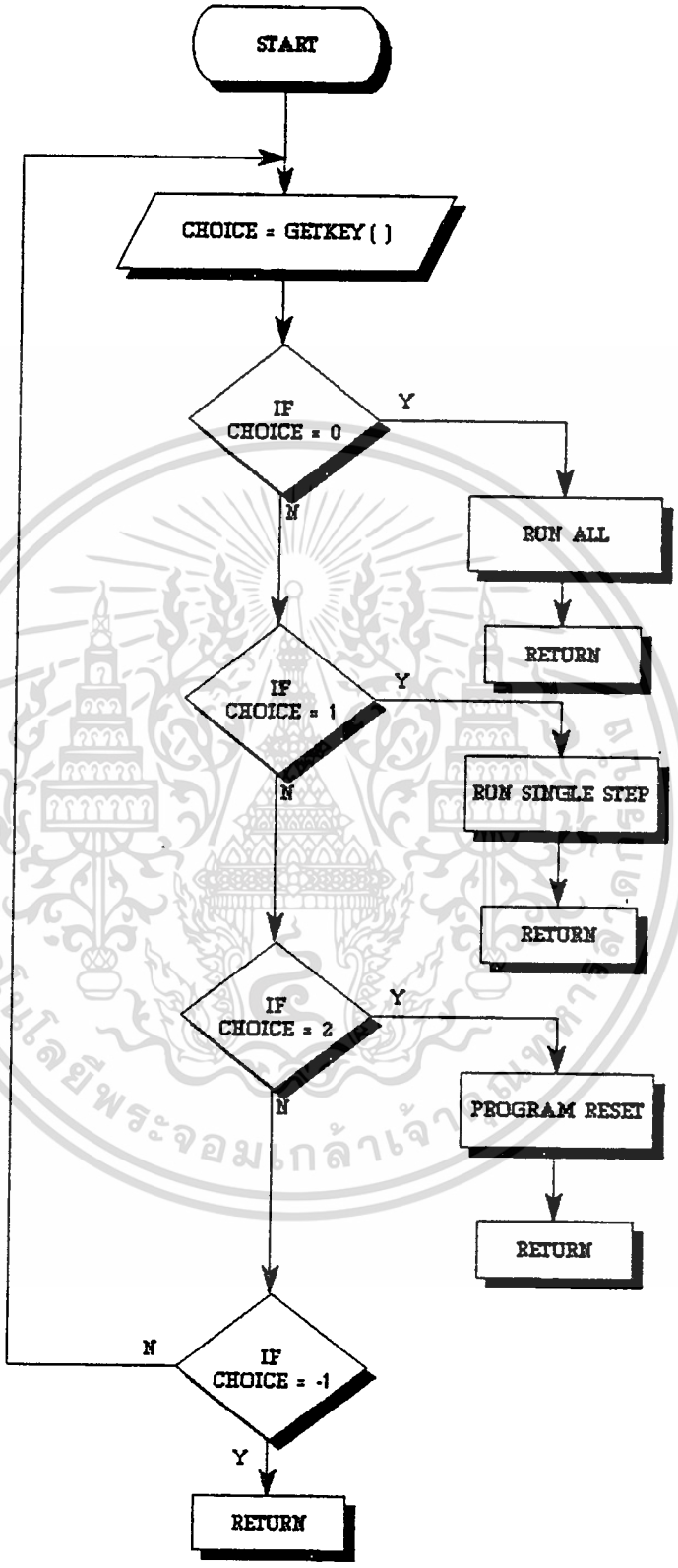
- Program_reset เป็นฟังก์ชันที่ใช้ในการรีเซ็ตค่าที่อยู่ในรีจิสเตอร์ต่างๆให้อยู่ในสภาวะเริ่มต้น นอกจากนี้ยังมีการส่งสัญญาณ Reset เพื่อใช้ในการรีเซ็ตระบบภายนอกด้วย
- Run_all เป็นฟังก์ชันที่ใช้ในการควบคุมให้โปรแกรมทำงานตามชุดคำสั่ง โดยอาศัยหลักการคือ เริ่มจากการดึงค่าของข้อมูลที่อยู่ในแอสเดรสเริ่มต้นซึ่งจะต้องมีการกำหนดให้ถูกต้องทุกครั้งว่าจะให้โปรแกรมเริ่มทำงานที่แอสเดรสใด เมื่อได้ค่าข้อมูลมาแล้วก็นำมาตรวจสอบว่าเป็นคำสั่งใดจะต้องมีโอเปอแรนด์ (Operand) ตามมาอีกหรือไม่ถ้าเป็นคำสั่งที่ต้องตามด้วยโอเปอแรนด์แล้วนำข้อมูลมาประมวลผลต่อไปซึ่งชุดคำสั่งของไมโครโปรเซสเซอร์ 8085 และ



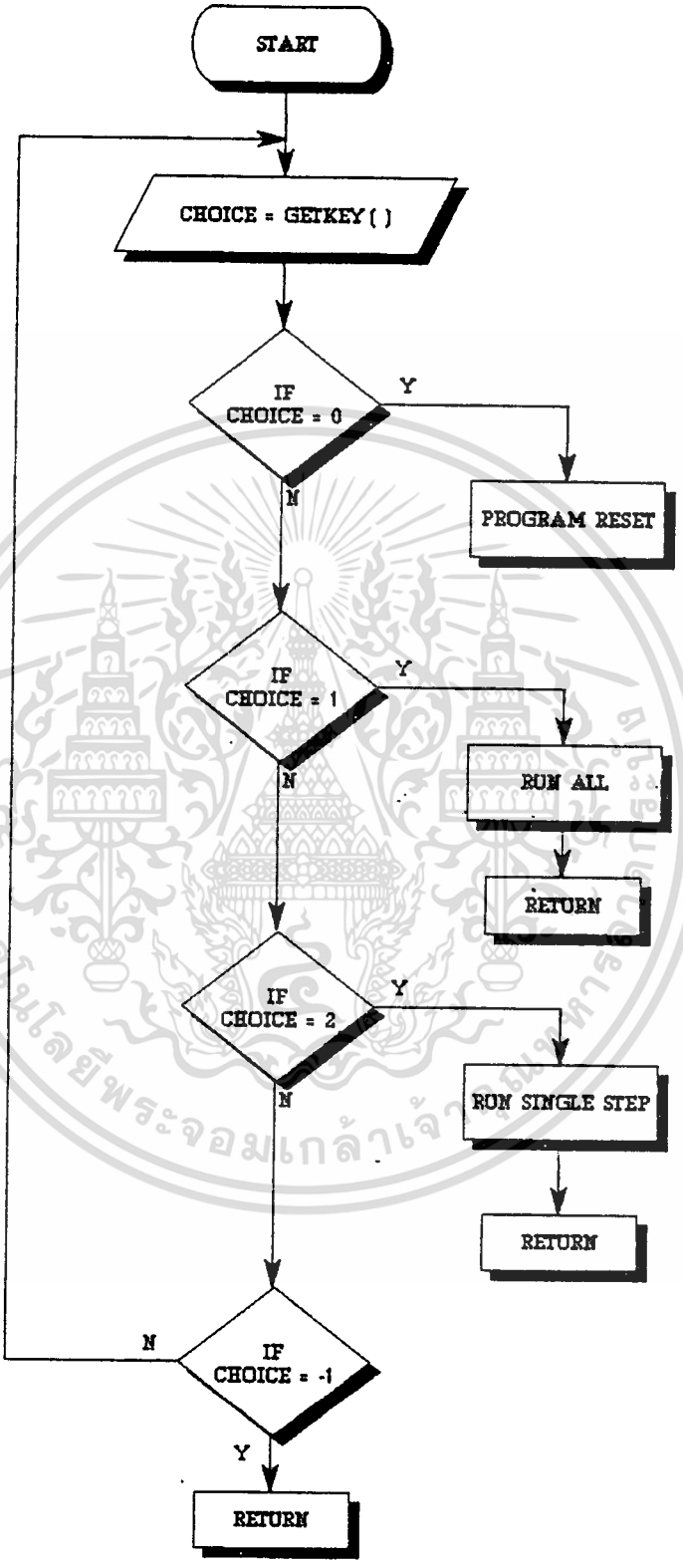
รูปที่ 3.9 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนู EDIT สำหรับไมโครโปรเซสเซอร์ 8085



รูปที่ 3.10 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนู EDIT สำหรับไมโครคอนโทรลเลอร์ MCS-51



รูปที่ 3.11 **แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนู** RUN **สำหรับไมโครโปรเซสเซอร์** 8085



รูปที่ 3.12 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนู RUN สำหรับไมโครคอนโทรลเลอร์ MCS-51

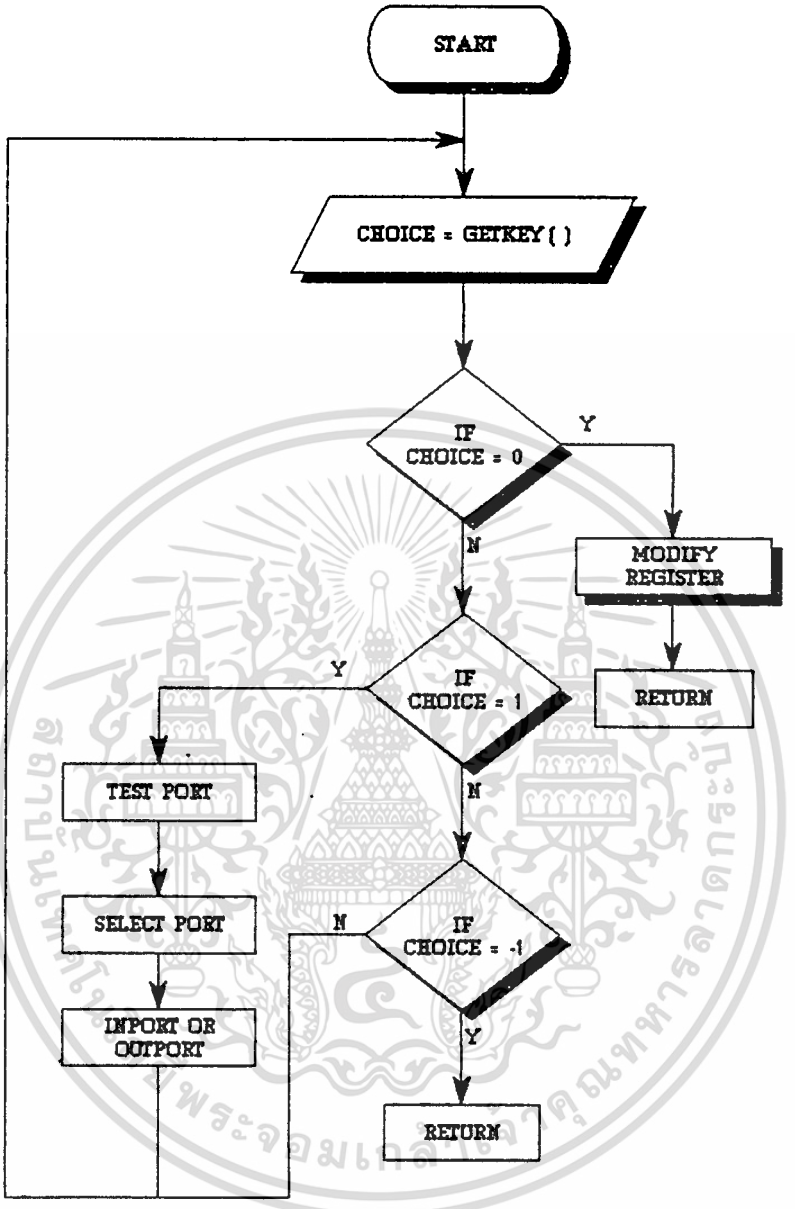
ไมโครคอนโทรลเลอร์ MCS-51จะถูกรวบรวมเป็นฟังก์ชันที่สามารถทำการเรียกใช้ได้หากต้องการที่จะหยุดการทำงานก่อนที่จะดึงค่าแอสเดรสที่ตั้งไว้ก็อาจทำได้โดยวิธีแรกนั้นให้กดคีย์Esc โปรแกรมก็จะถูกเบรค (Break)ค้างไว้ที่แอสเดรสที่ทำการหยุดนั้นส่วนอีกวิธีหนึ่งคือทำการรีเซ็ตจากระบบภายนอกแต่การหยุดวิธีนี้จะทำให้โปรแกรมทำการรีเซ็ตค่าเริ่มต้นใหม่ทั้งหมด

- Step ฟังก์ชันนี้จะมีหลักการทำงานที่เหมือนกับฟังก์ชันที่ผ่านมาแต่แตกต่างกันตรงที่ว่าในฟังก์ชันนี้จะเป็นการทำงานทีละคำสั่ง โดยหลังจากที่ทำงานไปแล้วคำสั่งหนึ่งก็จะมีการแสดงค่าแอสเดรสที่จะทำการรันถัดไปปละจะรอให้ผู้ใช้กดคีย์ F8 เพื่อสั่งให้โปรแกรมทำการรันโปรแกรมในแอสเดรสถัดไป

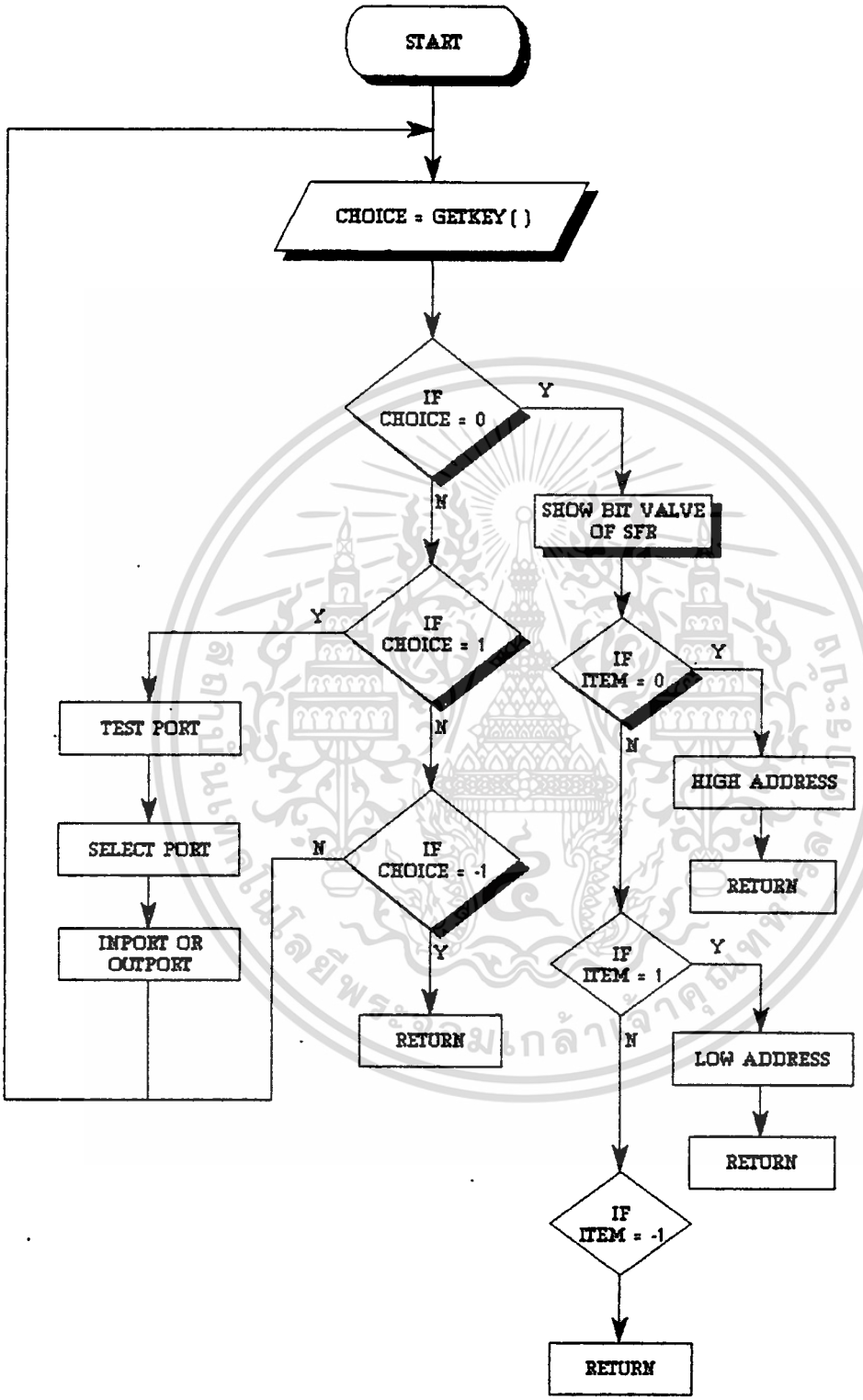
3.3.6. โครงสร้างของโปรแกรมเมนู Other

จากรูปที่ 3.13 จะแสดงถึงโฟลว์ชาร์ทการทำงานของฟังก์ชั่นเมนู Other ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 ซึ่งจากโฟลว์ชาร์ทจะเห็นว่า เริ่มต้นโปรแกรมจะรอการเลือกหัวข้อที่จะทำงานถ้าหากมีการเลือกเกิดขึ้นจะมีการเก็บค่าหัวข้อที่เลือก ไว้ในตัวแปร choice หลังจากนั้นก็จะทำการตรวจสอบดูว่ามีการเรียกใช้ฟังก์ชั่นใด ซึ่งแบ่งหัวข้อของเมนูย่อยออกเป็น 2 หัวข้อเมนูย่อย คือ เมนู Modify registor และเมนู Test port card เมื่อเลือกหัวข้อเมนูจะมีการส่งค่าให้กับตัวแปร choice ถ้า choice เท่ากับ 0 จะเป็นการเลือกใช้ฟังก์ชั่น Modify registor สำหรับการเปลี่ยนแปลงแก้ไขค่าข้อมูลหรือสภาวะต่างๆของรีจีสเตอร์ ถ้า choice เท่ากับ 1 จะเป็นการทดสอบอุปกรณ์ฮาร์ดแวร์ภายในการ์ด

จากรูปที่ 3.14 จะแสดงถึงโฟลว์ชาร์ทการทำงานของฟังก์ชั่นเมนู Other ของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 ซึ่งจากโฟลว์ชาร์ทจะเห็นว่า เริ่มต้นโปรแกรมจะรอการเลือกหัวข้อที่จะทำงานถ้าหากมีการเลือกเกิดขึ้นจะมีการเก็บค่าหัวข้อที่เลือก ไว้ในตัวแปร choice หลังจากนั้นก็จะทำการตรวจสอบดูว่ามีการเรียกใช้ฟังก์ชั่นใด ซึ่งแบ่งหัวข้อของเมนูย่อยออกเป็น 7 หัวข้อเมนูย่อย เมื่อเลือกหัวข้อเมนูจะมีการส่งค่าให้กับตัวแปร choice ถ้า choice เท่ากับ 0 จะเป็นการเลือกใช้ฟังก์ชั่น Bit value สำหรับแสดงค่าหน่วยความจำระดับบิต ถ้า choice เท่ากับ 1 จะเป็นการเลือกใช้ฟังก์ชั่น Test port card เป็นการทดสอบอุปกรณ์ฮาร์ดแวร์ภายในการ์ด ถ้า choice เท่ากับ 2 จะเป็นการเลือกใช้ฟังก์ชั่น Test memory เป็นการทดสอบหน่วยความจำภานนอก 64 ไบต์ ถ้า choice เท่ากับ 3 จะเป็นการเลือกใช้ฟังก์ชั่น Test port i/o เป็นการทดสอบพอร์ทภายนอกสามารถระบุหมายเลขพอร์ท และชนิดการควบคุมได้ ถ้า choice เท่ากับ 4 จะเป็นการเลือกใช้ฟังก์ชั่น Test keyboard เป็นการทดสอบคีย์บอร์ดของเครื่อง Jazz-31 ถ้า choice เท่ากับ 5 จะเป็นการเลือกใช้ฟังก์ชั่น Test display เป็นการทดสอบส่วนแสดงผลของเครื่อง Jazz-31 ถ้า choice เท่ากับ 6 จะเป็นการเลือกใช้ฟังก์ชั่น Test sound เป็นการทดสอบลำโพงของ



รูปที่ 3.13 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนู OTHER สำหรับไมโครโปรเซสเซอร์ 8085

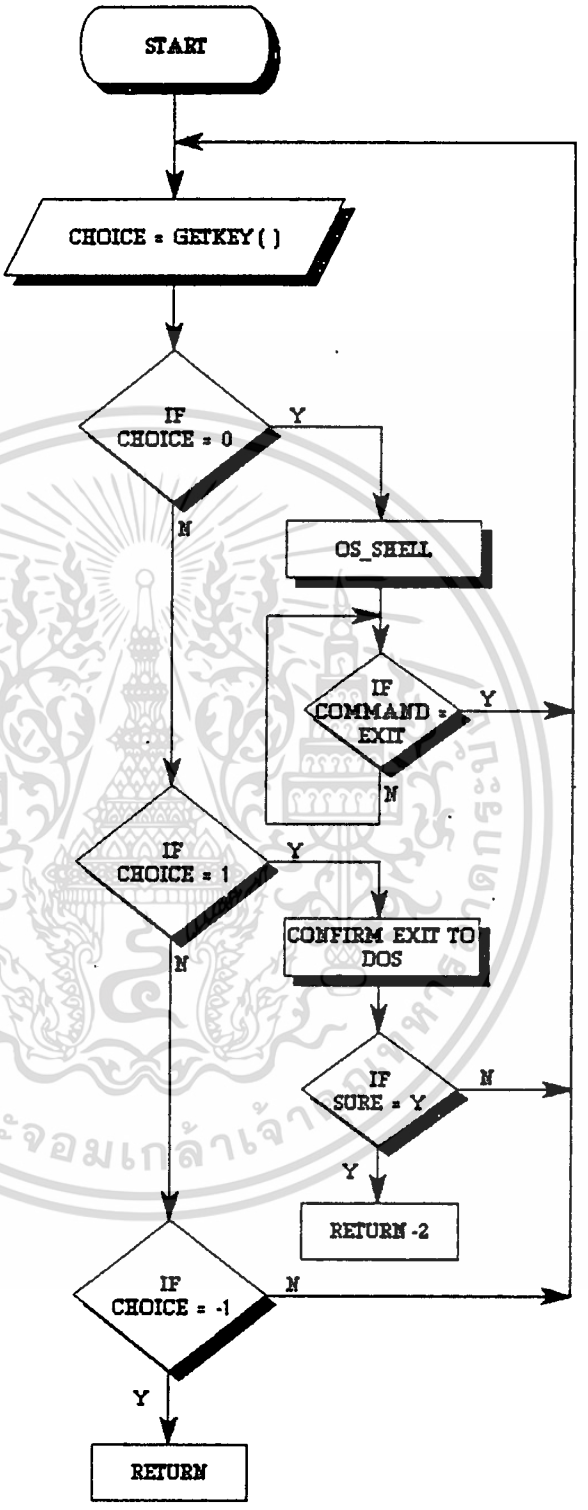


รูปที่ 3.14 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนู OTHER สำหรับไมโครคอนโทรลเลอร์ MCS-51

เครื่อง Jazz-31 ถ้าหากว่ามีการกดคีย์ Esc โปรแกรมก็จะกลับไปสู่โปรแกรมหลักที่เรียกมา

3.3.6. โครงสร้างของโปรแกรมเมนู Quit

เมนูนี้จะถูกใช้งานก็ต่อเมื่อต้องการที่จะเลิกทำงานจากโปรแกรมอีมูเลเตอร์แล้ว เพื่อที่จะออกไปสู่ระบบปฏิบัติการดอส ซึ่งจากรูปที่ 3.15 จะแสดงโฟลว์ชาร์ทการทำงานของฟังก์ชั่นเมนู Quit ซึ่งจะเห็นว่าทั้งอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 จะมีลักษณะเหมือนกัน ซึ่งจากโฟลว์ชาร์ทจะเห็นว่ามีขั้นตอนการทำงานคือ เริ่มต้นจะตรวจสอบการกดคีย์ว่าต้องการที่เรียกใช้เมนูย่อย Os_shell หรือ Exit to dos เมื่อตรวจสอบแล้วก็ทำงานตามฟังก์ชันนั้น แต่ถ้าการกดคีย์ Esc จะทำให้โปรแกรมกลับสู่โปรแกรมที่เรียกใช้ฟังก์ชันนี้



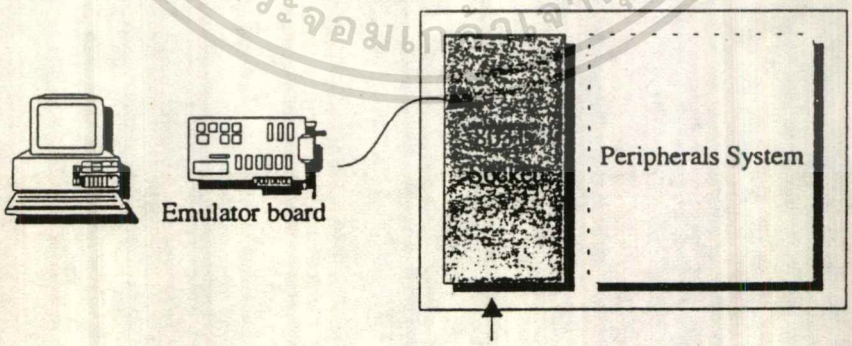
รูปที่ 3.15 แสดงโฟล์วชาร์ทของฟังก์ชั่นเมนู QUIT สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51

บทที่ 4

การใช้งานอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51

4.1 ลักษณะการเชื่อมต่อทางฮาร์ดแวร์

ลักษณะการเชื่อมต่อทางฮาร์ดแวร์ของอีมูเลเตอร์นี้ คือสามารถที่จะนำเอาการ์ดนี้ไปเสียบในสล็อตที่อยู่ในเครื่องไมโครคอมพิวเตอร์ได้เลย แล้วนำสายสัญญาณซึ่งมีขนาด 40 เส้น ต่อเข้ากับคอนเนคเตอร์(Connecter)ของการ์ดอีมูเลเตอร์ ส่วนปลายอีกด้านหนึ่งก็จะนำเข้าไปเสียบเข้ากับซ็อกเก็ตของหน่วยประมวลผลกลางที่มีไมโครโปรเซสเซอร์ 8085หรือไมโครคอนโทรเลอร์ MCS-51 เป็นหน่วยประมวลผลกลางในระบบที่ต้องการที่จะทำการทดสอบ หรือตรวจสอบ โดยต้องถอดหน่วยประมวลผลกลางตัวจริงออกจากระบบนั้นๆก่อน



รูปที่ 4.1 แผนภาพแสดงการเชื่อมต่อระบบกับการ์ดอีมูเลเตอร์

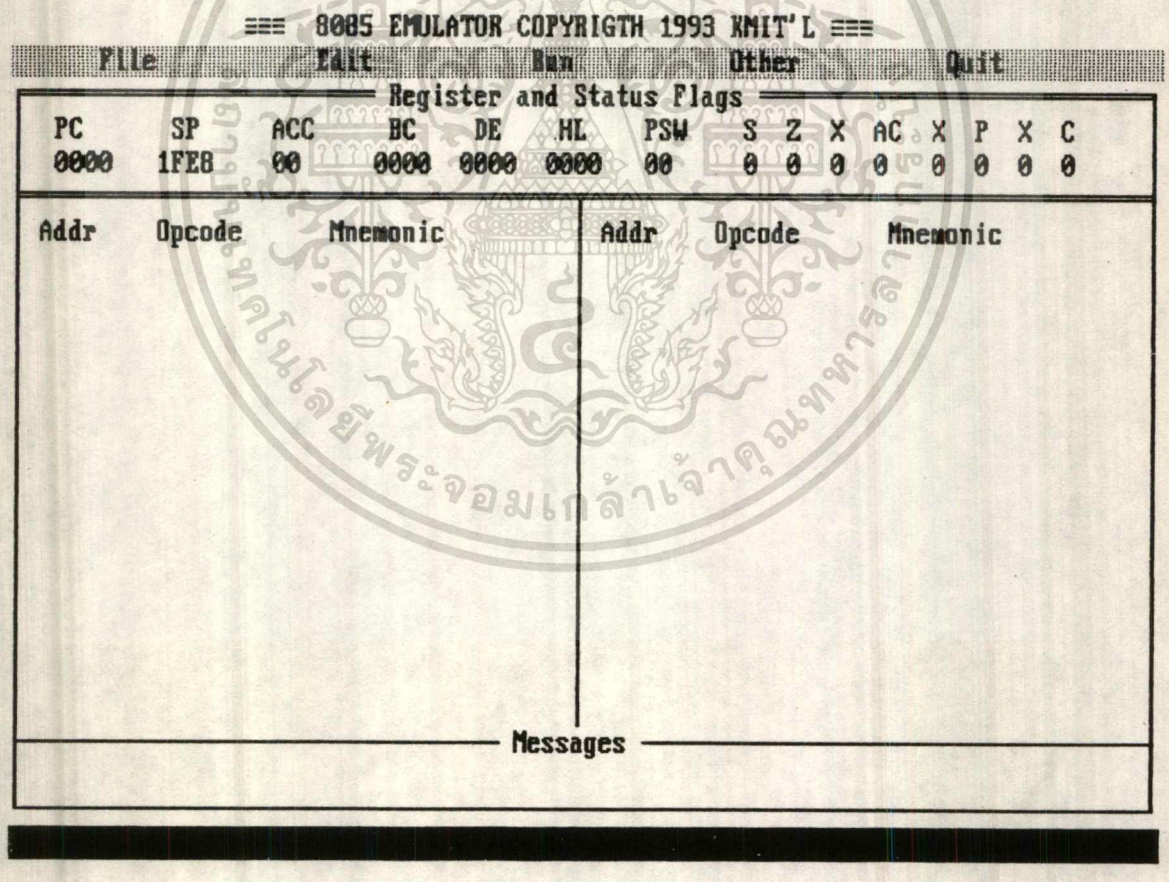
4.2 การเรียกใช้โปรแกรมของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51

เมื่อเข้าสู่ระบบการทำงานบนดอสแล้วให้ผู้ใช้พิมพ์ชื่อโปรแกรม (c:\>emulator) โปรแกรมจะเข้าสู่โปรแกรมหน้าจอแรกดังรูปที่ 4.2

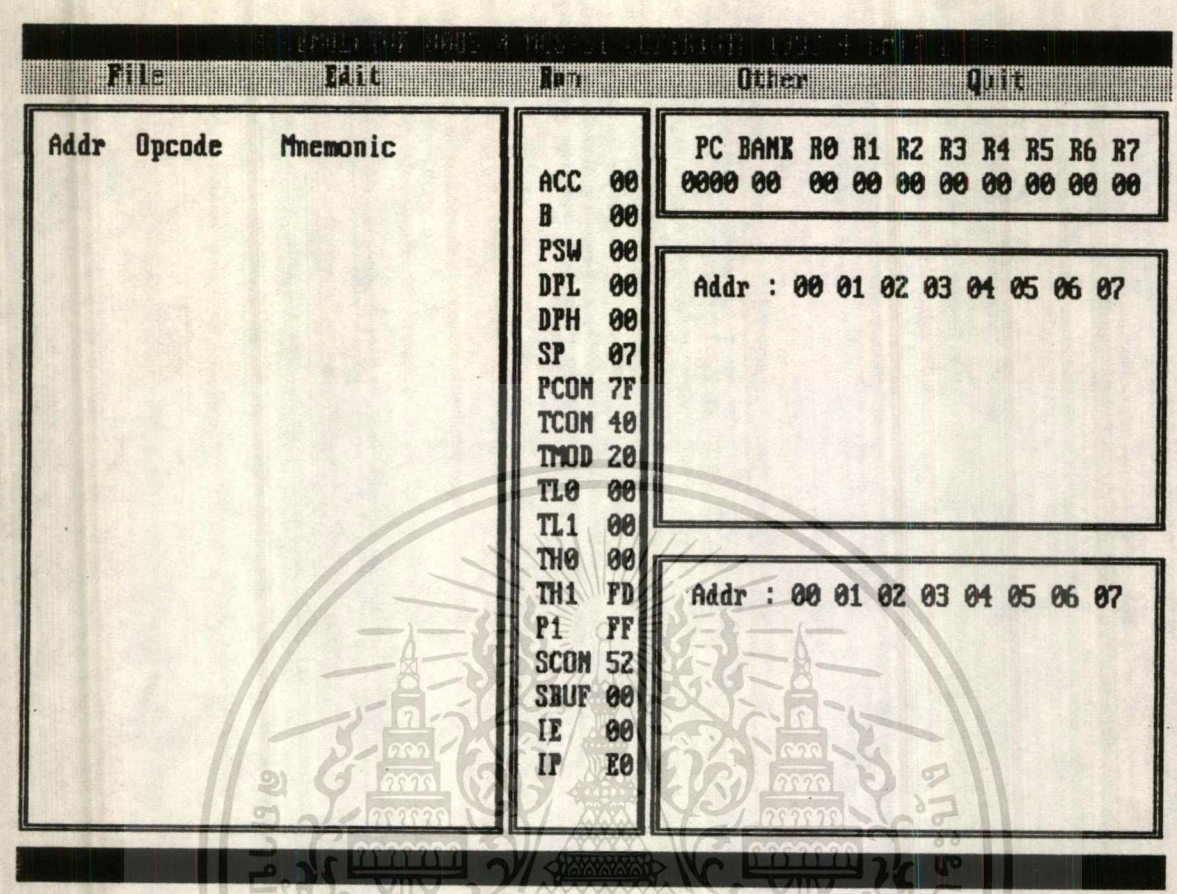
รูปที่ 4.2 แสดงหน้าจอแรกเมื่อใช้งานอีมูเลเตอร์

จากนั้นให้ผู้ใช้กดคีย์ใด ๆเพื่อเข้าสู่เมนูการเลือกใช้อีมูเลเตอร์ทั้งสองชนิดดังรูปที่ 4.3 ถ้าต้องการเมนูใดก็เลื่อนคีย์ลูกศรแล้วกดเอนเทอร์ก็จะเข้าสู่ระบบอีมูเลเตอร์ชนิดนั้นดังรูปที่ 4.4 แสดงหน้าจอแรกของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และรูปที่ 4.5 แสดงหน้าจอแรกของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 แต่ถ้าหน่วยความจำของเครื่องคอมพิวเตอร์ไม่พอโปรแกรมจะแสดงดังรูปที่ 4.6 ก. หรือวงจรภายนอกไม่ได้จ่ายไฟเลี้ยงวงจรโปรแกรมจะแสดงดังรูปที่ 4.6 ข. ซึ่งแต่ละโปรแกรมจะรวมอธิบายการเลือกใช้งานเมนูของอีมูเลเตอร์พอสังเขปดังนี้

รูปที่ 4.3 แสดงเมนูเลือกชนิดอีมูเลเตอร์



รูปที่ 4.4 แสดงหน้าจอแรกของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085



รูปที่ 4.5 แสดงหน้าจอแรกของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51

Out of memory
Memory not enoung for run program

รูปที่ 4.6 ก แสดงหน่วยความจำของเครื่องคอมพิวเตอร์ไม่พอสำหรับการเรียกใช้โปรแกรม

Singel board not +VCC

รูปที่ 4.6 ข แสดงระบบภายนอกไม่ได้จ่ายไฟเลี้ยงวงจร

4.2.1 การเลือกใช้งานระบบเมนู

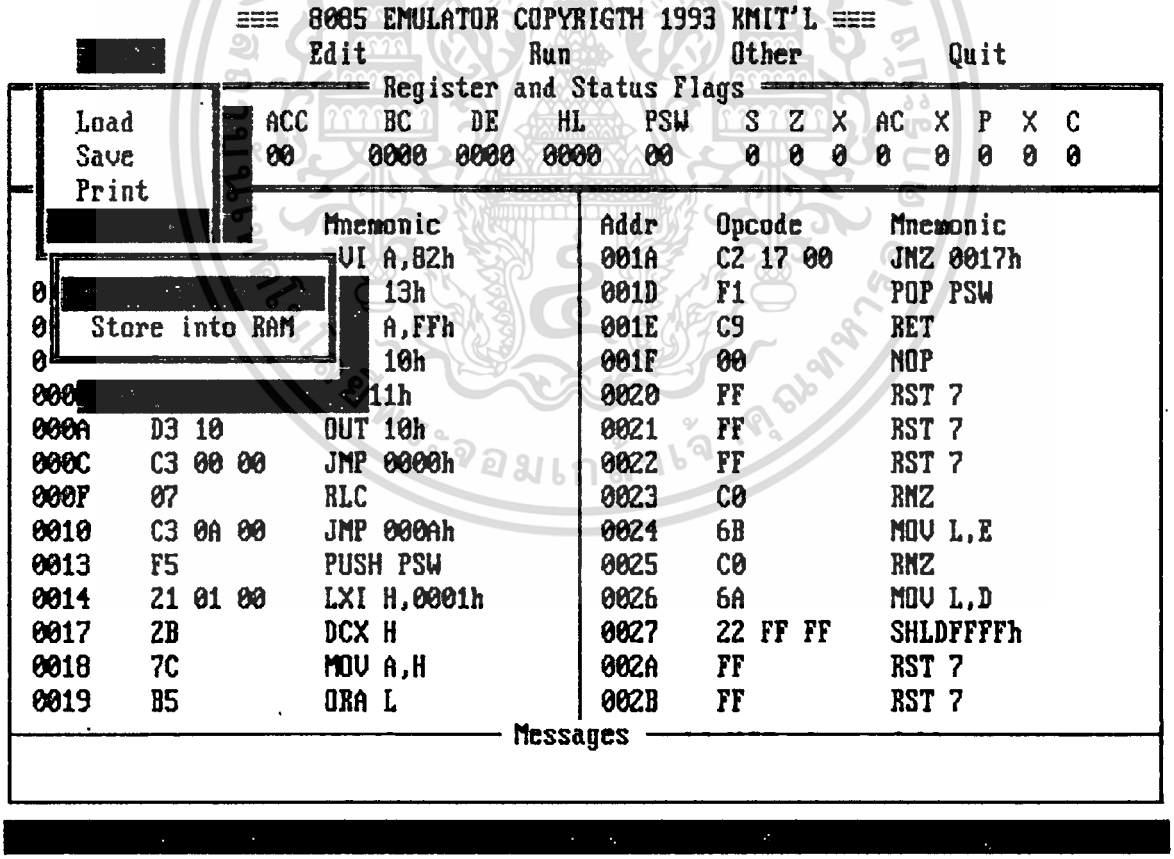
ระบบเมนูจะมีหัวข้อดังนี้ File, Edit, Run, Other และ Quit สำหรับการเรียกเมนูต่างๆขึ้นมาทำงานนั้นทำได้โดยกดฟังก์ชันคีย์ F10 หรือถ้าต้องการคำอธิบายการใช้งานโดยย่อก็สามารถเรียกใช้ได้โดยยกดฟังก์ชันคีย์ F2 เมื่อทำการกดฟังก์ชันคีย์ F10 แล้วจะปรากฏหัวข้อเมนูหลักที่แอกทีฟขึ้นมาเพียงเมนูเดียวเท่านั้น จากนั้นให้ทำการเลือกหัวข้อเมนูที่ต้องการทำงานโดยวิธีเลือกมีอยู่ 2 วิธีคือ วิธีแรกกดตัวอักษรตัวแรกที่เป็นตัวอักษรพิเศษของแต่ละเมนูหลัก หรืออีกวิธีหนึ่งคือ ใช้คีย์ลูกศรเลื่อนไปยังหัวข้อที่ต้องการแล้วกดคีย์ Enter หรือคีย์ลูกศรลง (Down arrow) และลักษณะการเลือกหัวข้อเมนูนี้จะเป็นลักษณะแบบวนรอบกล่าวคือ เมื่อเลื่อนแถบเคอร์เซอร์ไปที่เมนูหลักซ้ายสุด หรือขวาสุดของจอภาพ และถ้ายังมีการกดคีย์ลูกศรไปทางซ้ายหรือขวาอีกแถบเคอร์เซอร์ ก็จะไปปรากฏอยู่ที่หัวข้อเมนูด้านถัดไปของจอภาพ ส่วนรายละเอียดในการทำงานในแต่ละหัวข้อเมนูจะได้อธิบายในหัวข้อต่อไป

4.2.1.1. เมนู File

- เมื่อเลือกหัวข้อเมนูหลักนี้จะเกิด pull down menu ขึ้นมาและจะมีเมนูย่อยให้เลือกในการทำงานต่างๆที่เกี่ยวกับการจัดการเกี่ยวกับไฟล์ จากรูปที่ 4.7 และ 4.8 จะแสดงให้เห็นถึงไฟล์เมนูของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 ตามลำดับ ซึ่งทั้งสองจะมีลักษณะคล้ายกันแต่จะแตกต่างกันบ้างในส่วนของเมนูย่อยซึ่งจะได้อธิบายดังต่อไปนี้
  - Load จะทำหน้าที่ในการโหลดไฟล์ข้อมูลที่ต้องการจากแผ่นดิสก์เก็ต (Disket) เข้ามาไว้ในหน่วยความจำของระบบไมโครคอมพิวเตอร์ ซึ่งไฟล์ข้อมูลที่จะทำการโหลดเข้ามานี้ จะต้องเป็นไฟล์ของโปรแกรมคำสั่งของไมโครโปรเซสเซอร์ 8085 หรือคำสั่งของไมโครคอนโทรลเลอร์ MCS-51 ซึ่งอยู่ในรูปของ Hexadecimal file (ไฟล์ที่ได้จากการแอสแซมเบลอร์โดยโปรแกรมสำเร็จรูป) ซึ่งการโหลดไฟล์ข้อมูลนี้จะต้องระบุทั้งไดรฟ์ (Drive)และพาร์ท (Path) ที่อยู่ของไฟล์ให้ถูกต้องโดยโปรแกรมอีมูเลเตอร์ จะทำการแปลโปรแกรมคำสั่งให้เป็นรหัสนีมอนิค (Mnemonic) ของคำสั่งไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ MCS-51 โดยอัตโนมัติซึ่งโปรแกรมจะเริ่มต้นที่ 0000H
  - Save จะทำหน้าที่ในการเก็บไฟล์ข้อมูลที่อยู่ในหน่วยความจำของไมโครคอมพิวเตอร์ ซึ่งไฟล์นั้นอาจจะทำการโหลดมาจากแผ่นดิสก์เก็ต

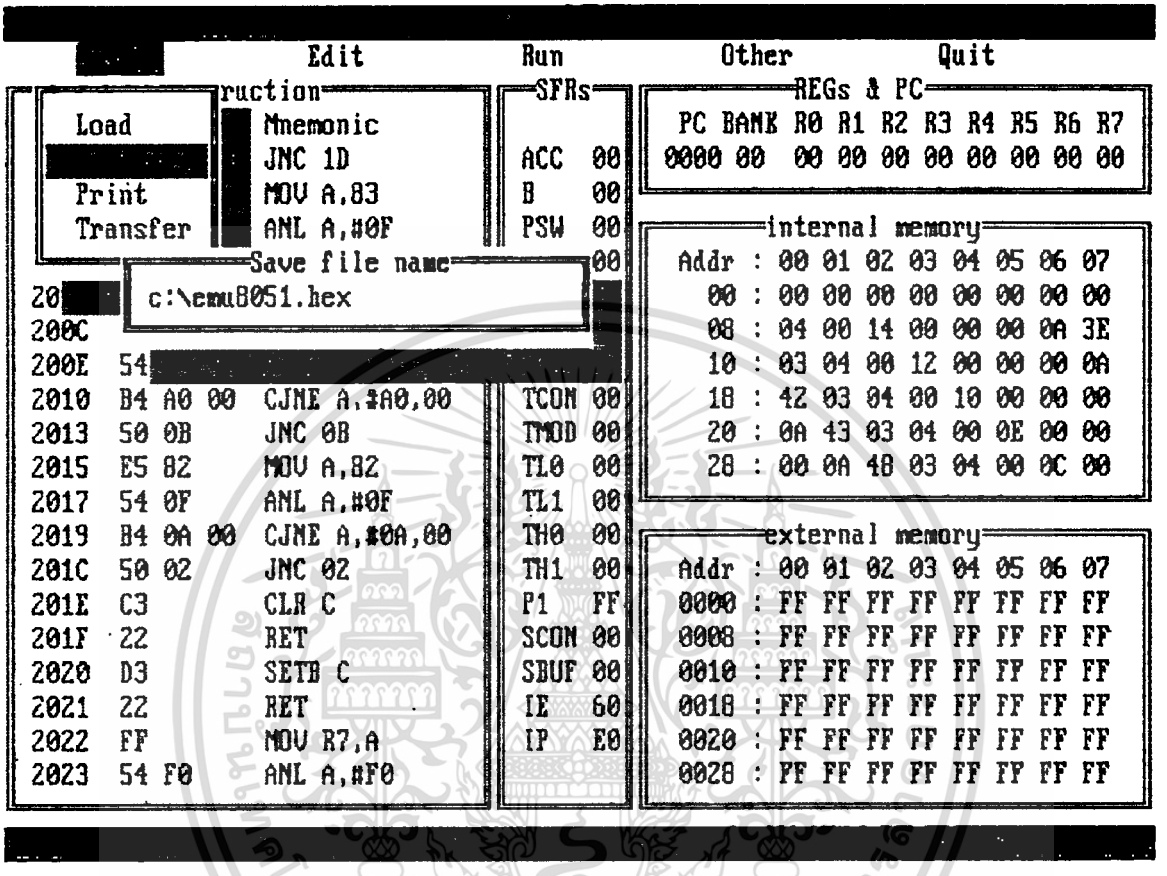
หรือทำการโหลดโปรแกรมมาจากหน่วยความจำโปรแกรมของระบบภายนอกที่ต้องการตรวจสอบ หรือทดสอบก็ได้ซึ่งหลักในการตั้งชื่อไฟล์จะอาศัยหลักการเดียวกับการตั้งชื่อไฟล์ในระบบปฏิบัติการดอส และในการเก็บไฟล์ข้อมูลจสามารถระบุแอดเดรสเริ่มต้นและแอดเดรสสุดท้ายที่ต้องการจะเก็บได้

- **Print** จะทำหน้าที่ในการพิมพ์ไฟล์ข้อมูลออกมาทางเครื่องพิมพ์ซึ่งสามารถที่จะเลือกพิมพ์ได้ 2 ลักษณะคือลักษณะแรกเป็นการพิมพ์แบบ Hexadecimal (พิมพ์เฉพาะเลขฐาน 16) หรือพิมพ์ในลักษณะนีมอนิค (ซึ่งเป็นการพิมพ์มิทั้งออปโค้ต (Opcode) และแอดเดรสของออปโค้ตนั้น) ลักษณะการเลือกใช้เมนูย่อยนี้คือ เมื่อเลือกใช้เมนูนี้แล้วจะมีการ



รูปที่ 4.7 แสดงเมนู Flie ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085

Pull down menu ย่อยขึ้นมาอีกเพื่อให้ระบุลักษณะการพิมพ์ที่ต้องการ แอดเดรสเริ่มต้น และแอดเดรสสุดท้ายที่ต้องการจะพิมพ์

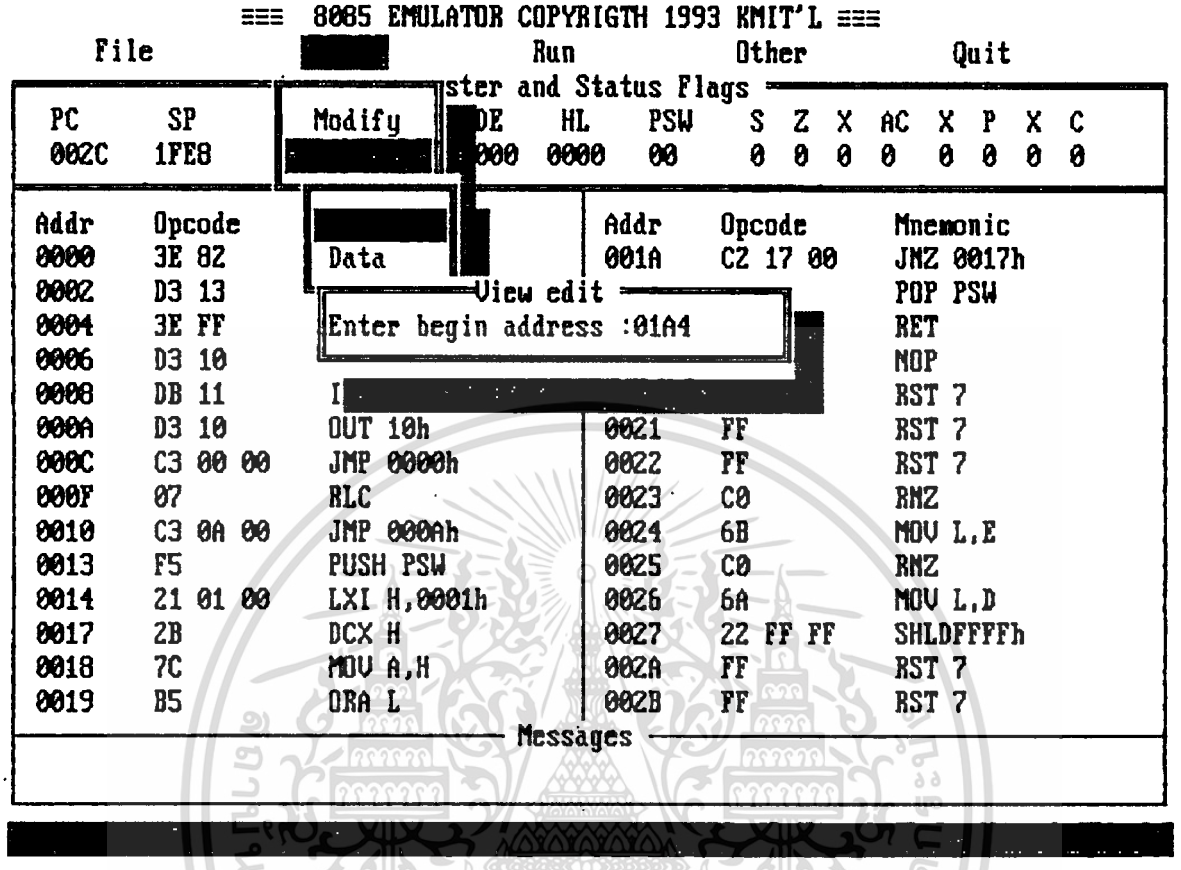


รูปที่ 4.8 แสดงเมนู File ของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51

- **Transfer** จะทำหน้าที่ในการโหลดข้อมูลหรือคำสั่งที่เก็บอยู่ภายในหน่วยความจำของระบบ ที่ต้องการ (จาก ROM หรือ EPROM) เข้ามาเก็บในหน่วยความจำของ ไมโครคอมพิวเตอร์ นอกจากนี้เมนูยังมีหน้าที่ในการถ่ายโอนข้อมูลให้กับหน่วยความจำข้อมูลภายนอกด้วย

4.2.1.2. เมนู Edit

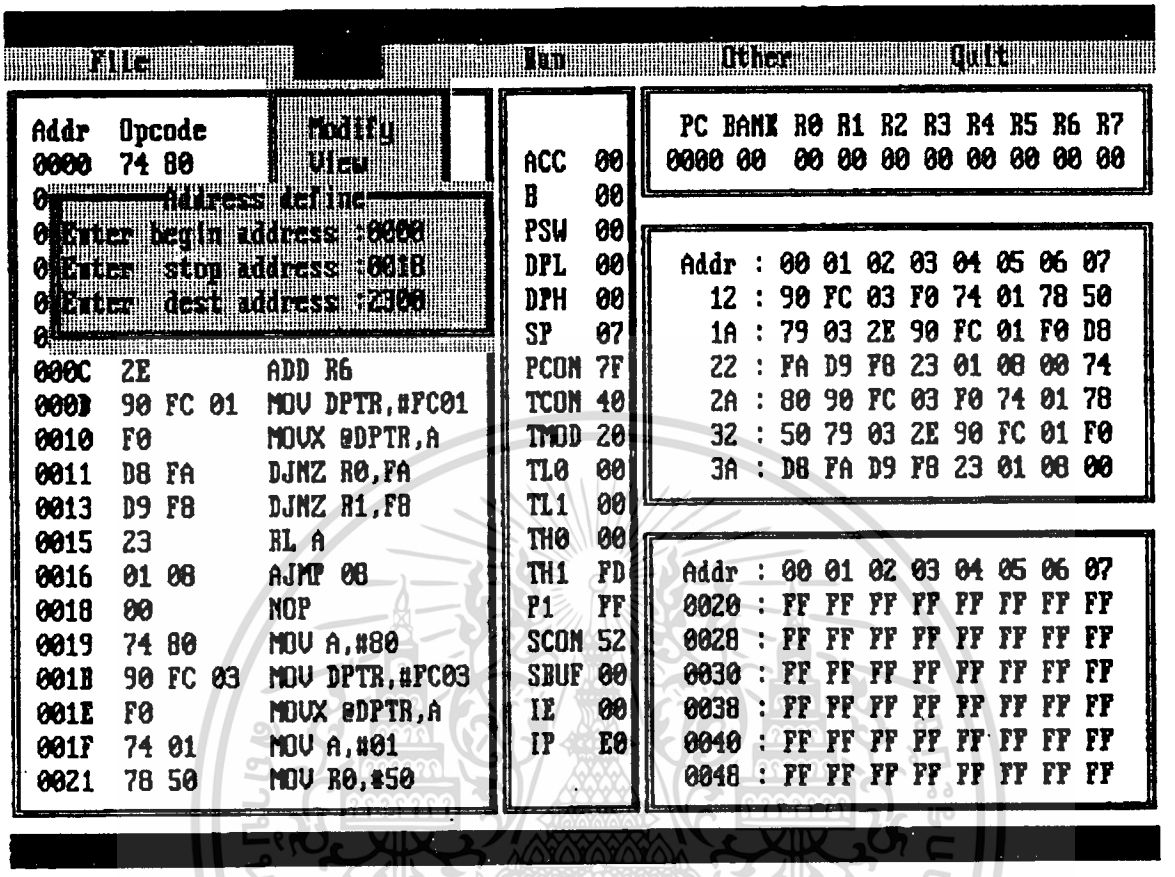
เมื่อเลือกใช้หัวข้อเมนูนี้ จะปรากฏ Pull down menu ขึ้นมา ซึ่งไม่ว่าจะเป็นอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 ก็จะมีลักษณะคล้ายกันดังแสดงในรูปที่ 4.9 และ รูปที่ 4.10 ตามลำดับ ซึ่งเมนูย่อยที่อยู่ภายในหัวข้อนี้ จะมีเมนูย่อยให้



รูปที่ 4.9 แสดงเมนู Edit ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085

เลือก 2 เมนูคือ Modify และ View และข้อสำคัญก็คือ เมนูนี้จะถูกใช้งานก็ต่อเมื่อมีการโหลดไฟล์ข้อมูลมาจากแผ่นดิสก์เก็ต หรือมีการถ่ายโอนหน่วยข้อมูลมาจากหน่วยความจำของระบบที่ต้องการทดสอบมาเก็บอยู่ในหน่วยความจำของไมโครคอมพิวเตอร์แล้ว

- **Modify** เมนูนี้จะทำหน้าที่สำหรับการเปลี่ยนแปลงข้อมูล หรือแก้ไขข้อมูลในหน่วยความจำต่างๆ ตามแอดเดรสที่ต้องการของโปรแกรมที่ได้ทำการโหลดข้อมูลเข้ามา และเมื่อเลือกหัวข้อเมนูย่อยนี้จะมีการให้กำหนดแอดเดรส ที่ต้องการจะเปลี่ยนแปลง หรือแก้ไขโดยค่าของแอดเดรสที่ป้อนเข้าไปจะต้องเป็นเลขฐาน 16 เท่านั้น หลังจากป้อนค่าแอดเดรสที่ต้องการแก้ไขแล้ว จะมีการแสดงค่าข้อมูลเดิมออกมา แล้วให้ทำการป้อนค่าใหม่ที่ต้องการ



รูปที่ 4.10 แสดงเมนู Edit ของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51

- View เมนูนี้จะมีหน้าที่สำหรับแสดงโปรแกรมที่ทำการโหลดหรือถ่ายโอนเข้ามาจากหน่วยความจำต่างๆ ตามแอดเดรสที่ต้องการกำหนดค่าแอดเดรสของโปรแกรมที่ต้องการแสดงนั้น จะต้องป้อนเป็นเลขฐาน 16 เท่านั้น

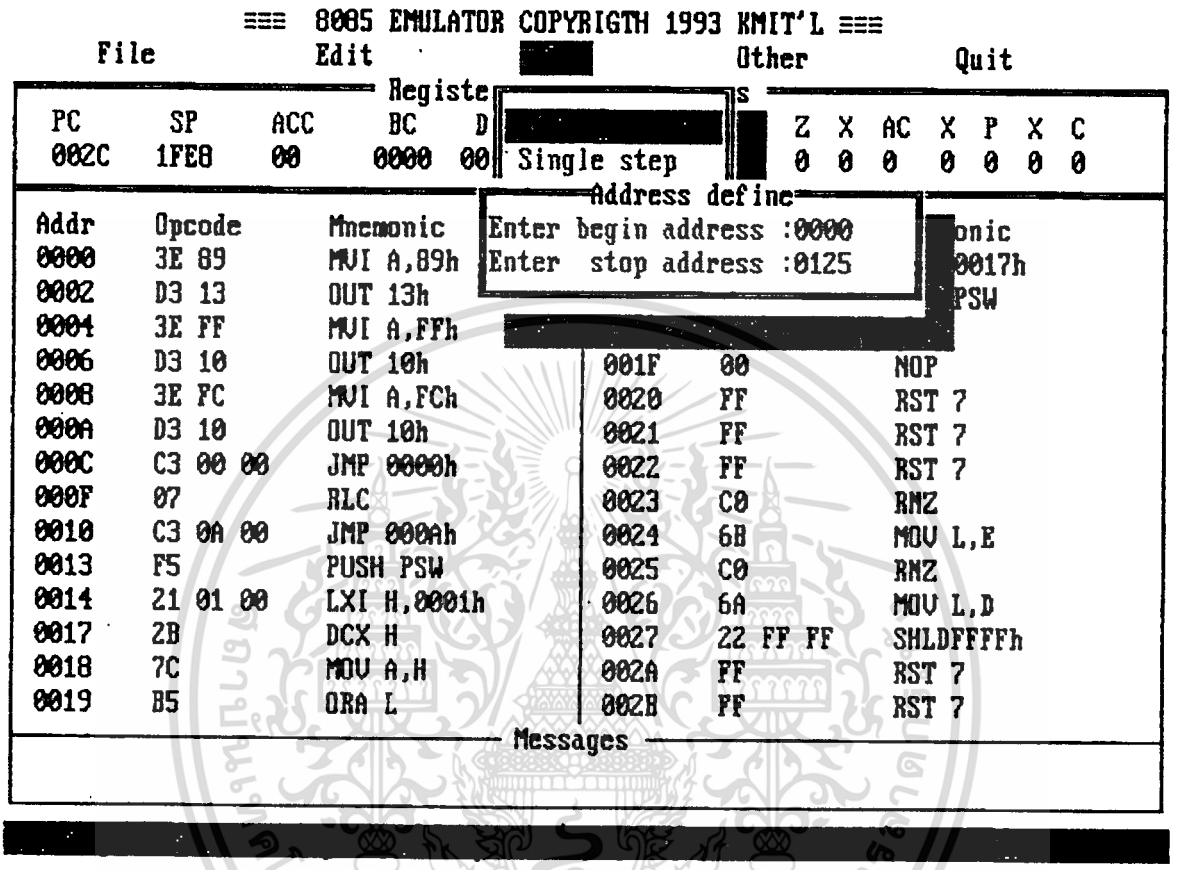
4.2.1.3. เมนู Run

หน้าที่ของหัวข้อเมนูนี้จะเป็นการนำข้อมูลที่โหลดออกมาจากแผ่นดิสก์เก็ต หรือถ่ายโอนมาจากระบบหน่วยความจำของระบบที่ต้องการตรวจสอบหรือทดสอบมาทำการรัน ซึ่งก่อนที่จะใช้เมนูนี้จะต้องแน่ใจว่าได้ทำการโหลดหรือถ่ายโอนไฟล์ข้อมูลของคำสั่งไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ MCS-51 เข้ามาในหน่วยความจำของระบบ ไมโครคอมพิวเตอร์แล้ว ซึ่งจากรูปที่ 4.11 และรูปที่ 4.12 จะแสดงให้เห็นเมนู Run ซึ่งในหัวข้อเมนูนี้จะมีอยู่ 3 เมนูย่อย คือ

- **Program reset** เมนูนี้จะทำหน้าที่เปรียบเสมือนในการรีเซ็ตการทำงานของไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ MCS-51 ซึ่งจะมีการส่งสัญญาณรีเซ็ตออกไปกับระบบภายนอกด้วยและผลจากการใช้เมนูนี้ค่าของรีจีสเตอร์ หรือหน่วยความจำข้อมูลภายในมีค่าเริ่มต้นด้วย
- **Single step** การทำงานในเมนูนี้จะเป็นการสั่งให้โปรแกรมที่เขียนด้วยคำสั่งของไมโครโปรเซสเซอร์ 8085 หรือ ไมโครคอนโทรลเลอร์ MCS-51 ทำงานทีละคำสั่งไปตามลำดับจากค่าแอดเดรสเริ่มต้นจนถึงค่าแอดเดรสสุดท้ายที่ได้กำหนด ซึ่งเมื่อเลือกใช้เมนูนี้จะมีการกำหนดแอดเดรสเริ่มต้น และแอดเดรสสุดท้ายซึ่งแอดเดรสที่กำหนดนั้นจะต้องเป็นเลขฐาน 16 และเมื่อป้อนค่าแอดเดรสแล้ว แอดเดรสเริ่มต้นจะถูกรันและจะรอจนกระทั่งมีการกดคีย์ F8 เพื่อทำการรันทีละคำสั่งต่อไป ในระหว่างการรันทีละคำสั่งนี้สามารถที่จะเลือกการทำงานในฟังก์ชันอื่นๆได้อีก โดยจะแยกอธิบายฟังก์ชันที่สามารถใช้ได้กับอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS 51

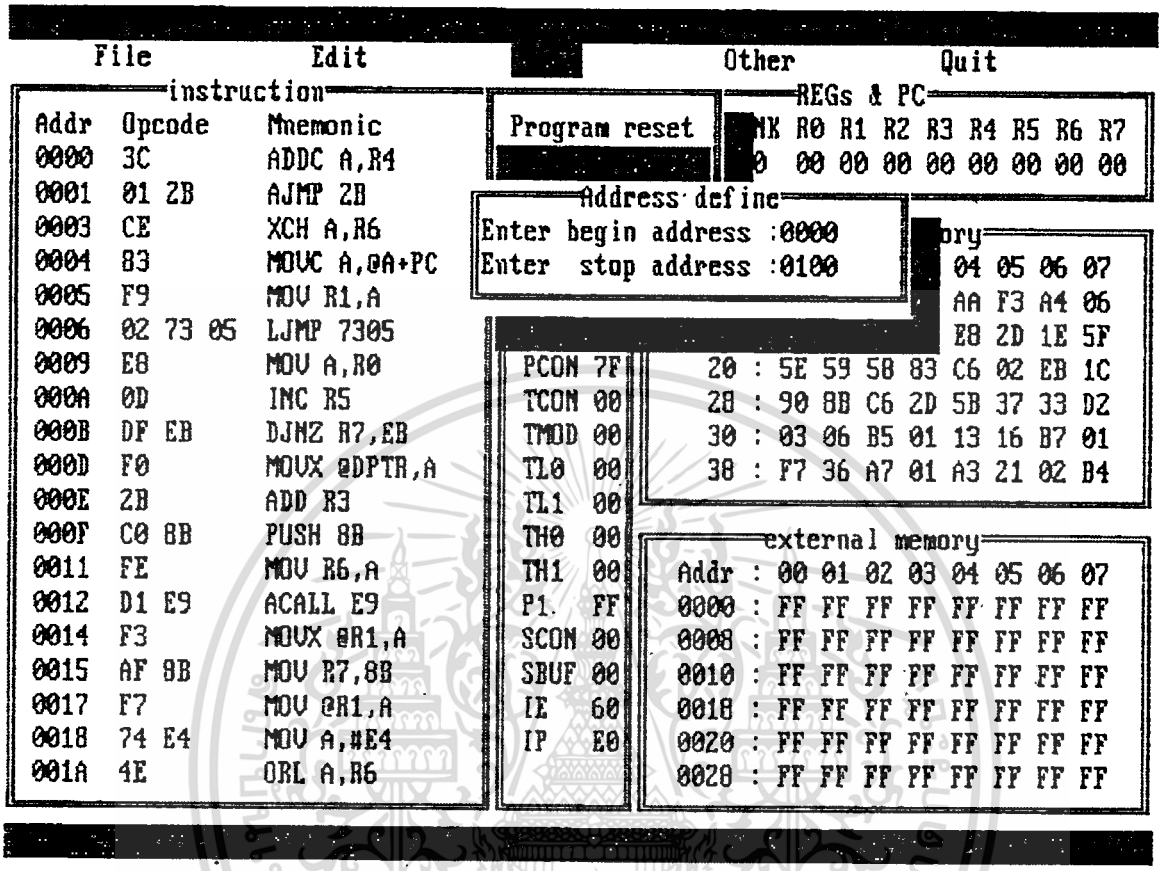
  F5 - เป็นการแสดงตัวโปรแกรมในแอดเดรสอื่นๆที่ผู้ใช้ต้องการ
  F6 - เป็นการแสดงข้อมูลในแอดเดรสอื่นๆที่ผู้ใช้ต้องการ
  F7 - เป็นการเปลี่ยนแปลงค่าในรีจีสเตอร์ต่างๆ
  F8 - เป็นคำสั่งที่ให้มีการทำงานทีละคำสั่งต่อไป
  Esc- เป็นการยกเลิกการทำงานของโปรแกรม

- **Run all** ลักษณะการใช้งานในเมนูนี้ จะเป็นการนำโปรแกรมที่เขียนขึ้นมาด้วยชุดคำสั่งของไมโครโปรเซสเซอร์ 8085 หรือไมโครคอนโทรลเลอร์ MCS-51 มาทำการรัน โดยหลังจากที่เลือกทำงานในหัวข้อเมนูนี้แล้วจะปรากฏวินโดว์ขึ้นมา เพื่อให้ผู้ใช้กำหนดค่าแอดเดรสเริ่มต้น และแอดเดรสสุดท้ายที่จะทำการรันโปรแกรมก่อนที่จะทำการรันโปรแกรมบนเมนูนี้ควรจะทำการ reset program ก่อนทุกครั้ง เพื่อความแน่ใจว่า โปรแกรมที่เขียนขึ้นมาจากชุดคำสั่งทำงานไม่ผิดพลาด และถ้าหากต้องการที่จะหยุด ก่อนที่จะถึงค่าแอดเดรสสิ้นสุดที่กำหนด



รูปที่ 4.11 แสดงเมนู Run ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085

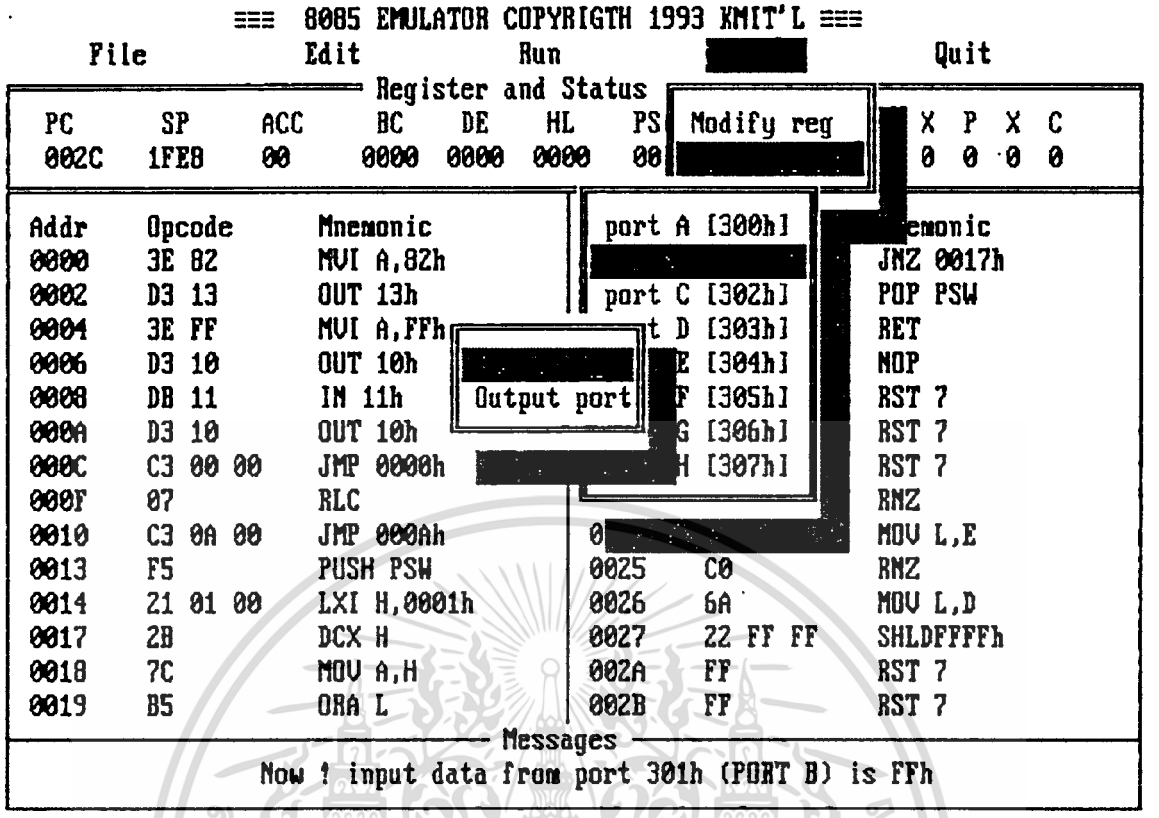
ไว้ทำได้ 2 วิธีคือ วิธีแรกให้ทำการกดคีย์ Esc และเมื่อมีการหยุดการทำงานของโปรแกรมแล้วค่าสภาวะต่าง ๆ ที่ได้ จากการสั่งให้โปรแกรมทำงานจะค้างไว้ จนกว่าผู้ใช้จะทำการรันโปรแกรมต่อไปหรือทำการรีเซ็ตโปรแกรม ส่วนอีกวิธีหนึ่ง คือการสั่งให้โปรแกรมหยุดการทำงาน โดยการให้สัญญาณรีเซ็ตกับหน่วยประมวลผลกลาง ซึ่งผลจากการหยุดในลักษณะนี้จะทำให้ค่าสภาวะต่างๆถูกรีเซ็ตให้เป็นค่าเริ่มต้น



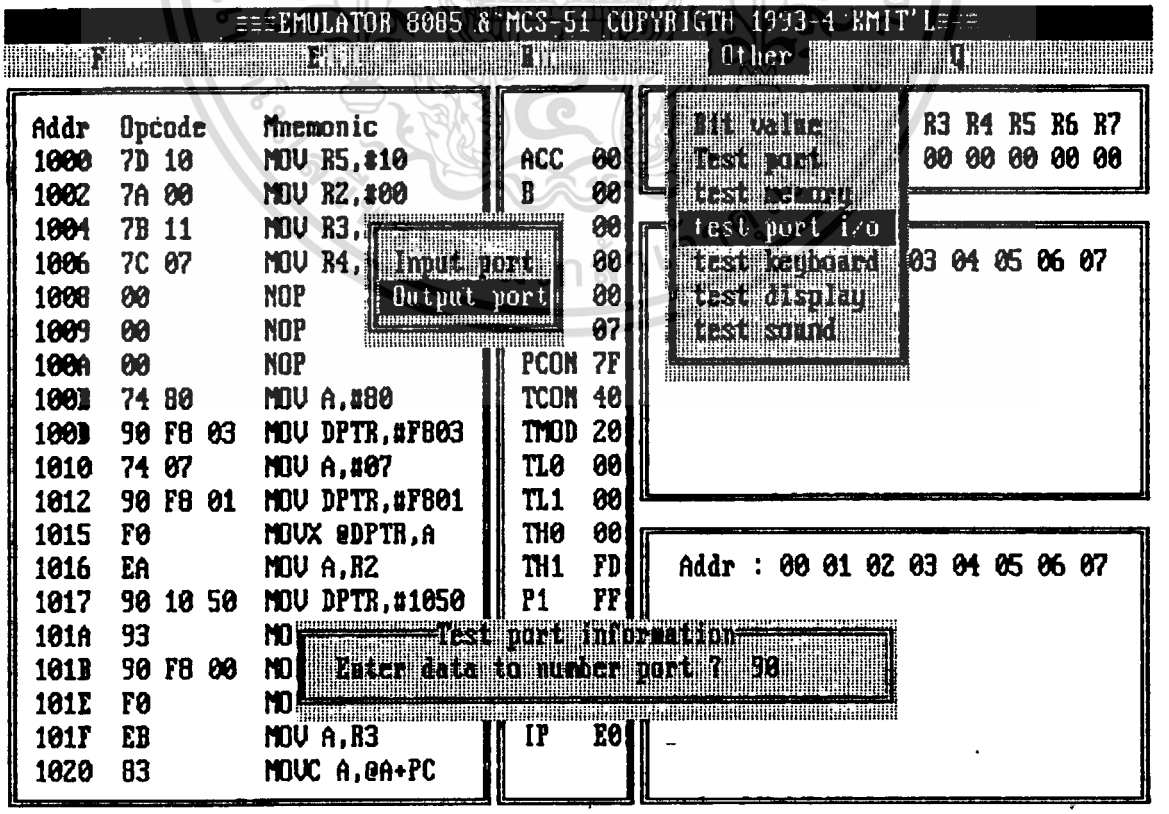
รูปที่ 4.12 แสดง เมนู Run ของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51

4.2.1.4. เมนู Other

สำหรับหัวข้อเมนูนี้ จะมีหน้าที่ในการใช้งานอยู่ 2 ลักษณะ คือ ในอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 จะมีการเปลี่ยนแปลงหรือแก้ไขค่าข้อมูลในรีจีสเตอร์ต่างๆ แต่สำหรับอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 นั้น จะเป็นการแสดงค่าของข้อมูลในระดับบิตของหน่วยความจำข้อมูลภายในที่สามารถเข้าถึงในระดับบิตได้ ส่วนอีกลักษณะหนึ่งคือการทดสอบพอร์ตต่าง ๆ ของอีมูเลเตอร์ โดยตรงซึ่งดูได้จากวงจรภาพในอีมูเลเตอร์นี้ซึ่งประโยชน์ในการทดสอบพอร์ตโดยใช้เมนูนี้ คือ สามารถทำการทดสอบการส่งหรือรับข้อมูลผ่านทางพอร์ตของการ์ดได้โดยตรง ดังนั้นจึงสามารถที่จะทำการตรวจสอบสภาวะต่างๆได้โดยยังไม่ได้ทำการรันโปรแกรม จากรูปที่ 4.13 และ 4.14 แสดงถึงเมนู Other ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 ตามลำดับ



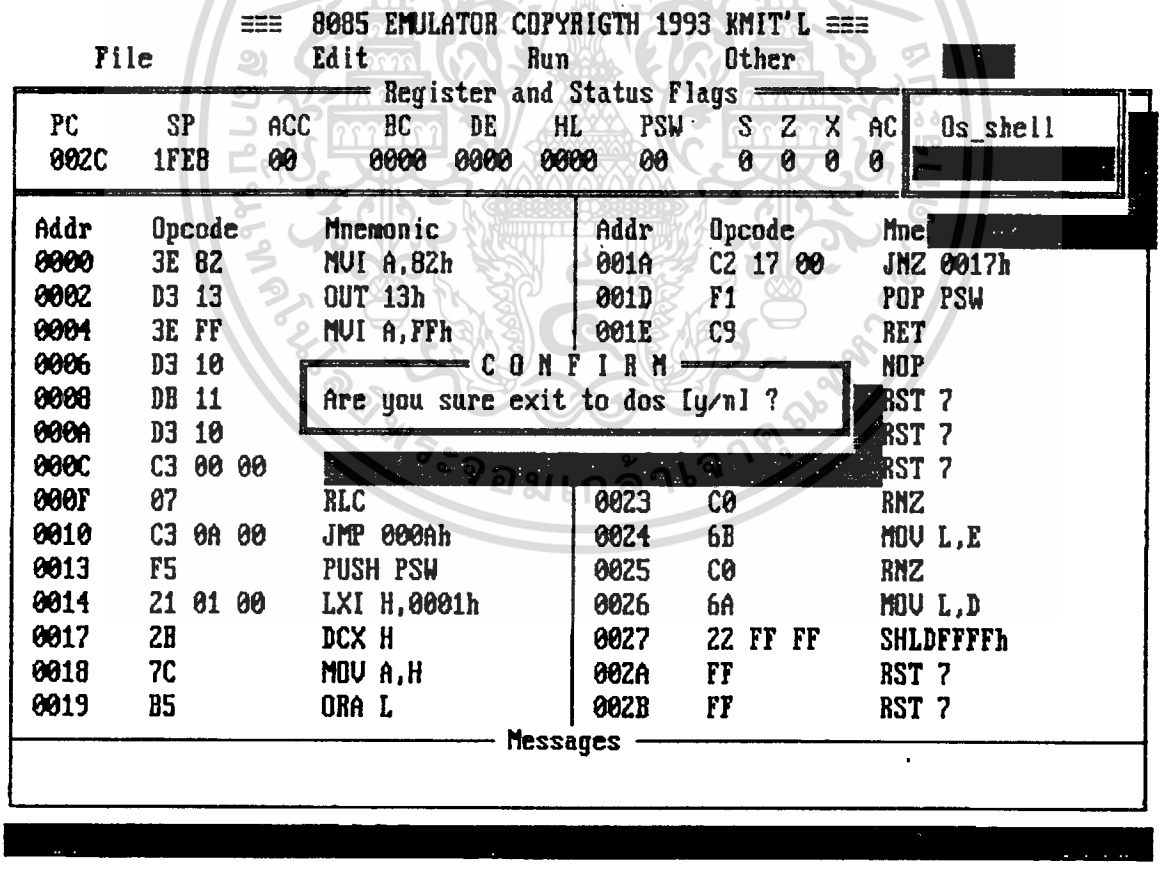
รูปที่ 4.13 แสดงเมนู Other ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085



รูปที่ 4.14 แสดงเมนู Other ของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51

4.2.1.5. เมนู Quit

เมนูนี้จะใช้เมื่อต้องการเลิกทำงาน โดยจะออกจากโปรแกรมไปที่ระบบปฏิบัติการ ซึ่งสามารถที่จะไปที่ระบบปฏิบัติการเป็นการชั่วคราวหรือออกไประบบปฏิบัติการเป็นการถาวรเลยก็ได้ ซึ่งเมนูย่อยที่มีให้เลือกในเมนูนี้คือ Os-shell จะใช้ในการที่ต้องการออกจากระบบปฏิบัติการเป็นการชั่วคราว สามารถกลับมาใช้โปรแกรมอีมูเลเตอร์อีกได้โดยการป้อนคำสั่ง Exit ที่ระบบปฏิบัติการดอส ส่วนอีกเมนูหนึ่ง คือ Exit to dos จะถูกใช้ในกรณีที่ต้องการจะออกจากระบบปฏิบัติการเป็นการถาวร โดยจะมีคำถามเพื่อย้ำให้แน่ใจอีกครั้ง โดยจะแสดงในรูปที่ 4.15 แสดงถึงเมนูของ Quit ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ซึ่งจะมีลักษณะเหมือนกัน



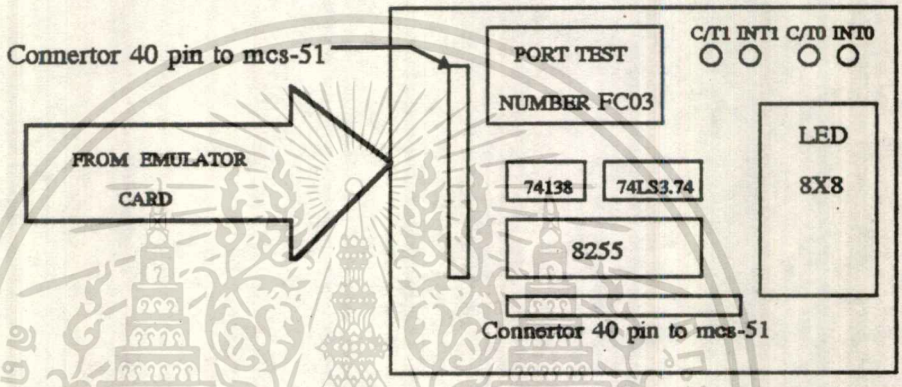
รูปที่ 4.15 แสดงเมนู Quit ของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085

4.3 ตัวอย่างโปรแกรมและวงจรฮาร์ดแวร์

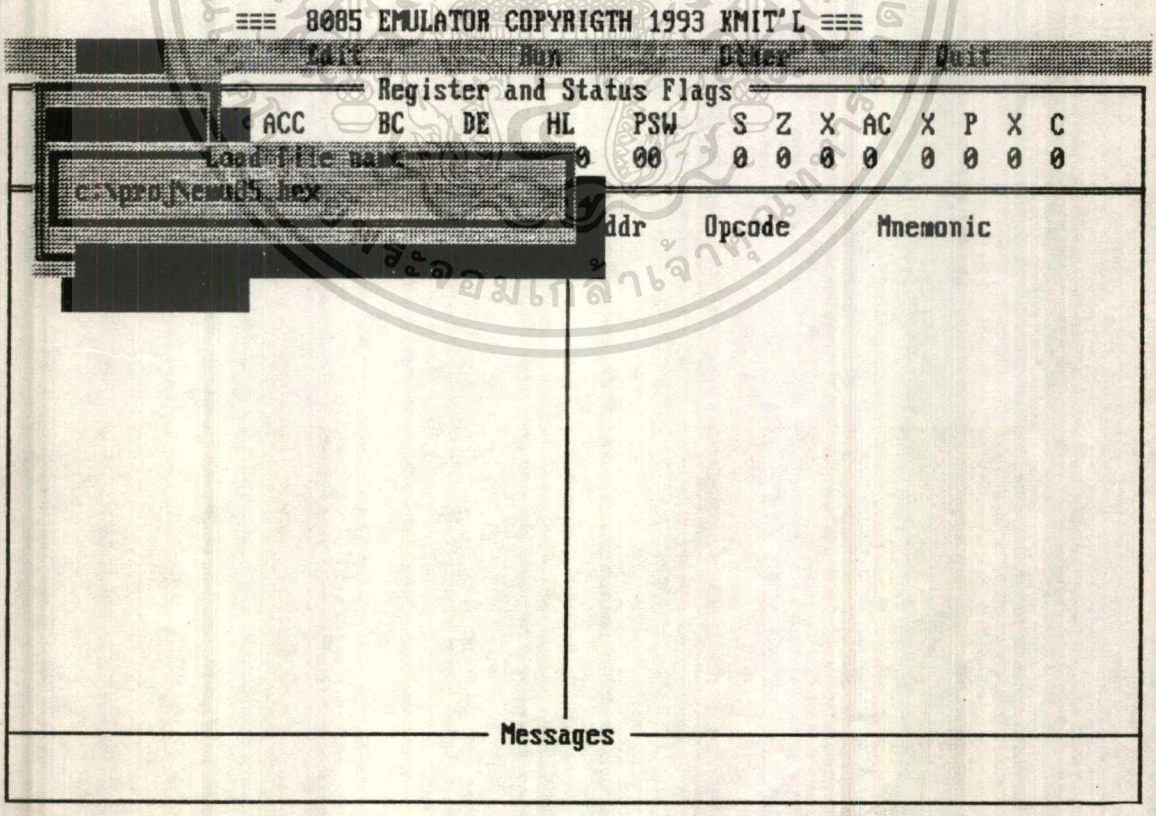
4.3.1 โปรแกรมและวงจรฮาร์ดแวร์เกี่ยวกับไมโครโปรเซสเซอร์ 8085

- โปรแกรมไฟวิ่งแบบเมตริกซ์ขนาด 8X8

ให้ผู้ใช้ต่อสาย 40 พิน (emulator cpu 8085 & mcs-51) จากการ์ดเข้ากับวงจรทดลอง (connector emulator for cpu 8085) ดังรูปที่ 4.16 และให้ผู้ใช้เลือกเมนู Load ของอีมูลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 เรียกโปรแกรม emu85.hex ดังรูปที่ 4.17 โปรแกรมนี้เป็นโปรแกรมแสดงไฟวิ่งแบบขนาด 8X8 บิต



รูปที่ 4.16 แสดงการเชื่อมต่ออีมูลเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 กับวงจรทดลอง

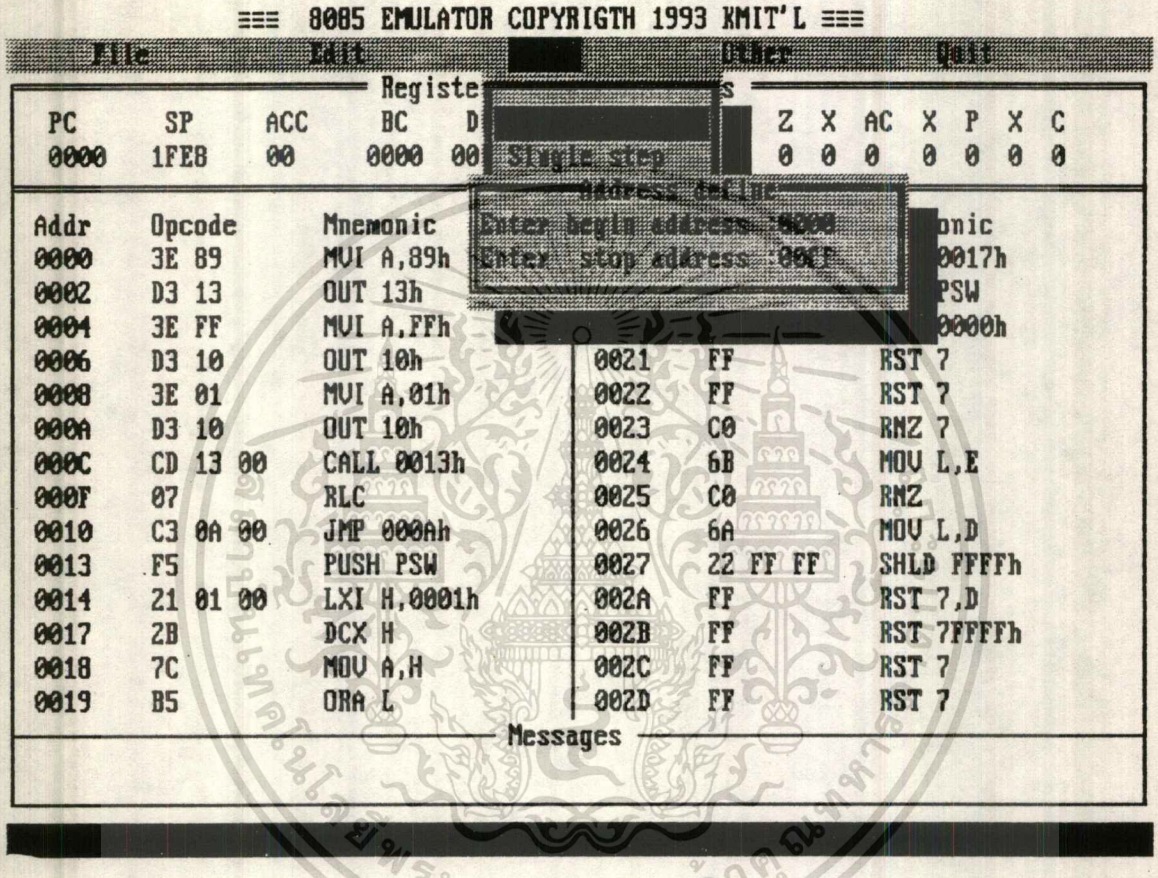


รูปที่ 4.17 แสดงการเรียกใช้โปรแกรมภาษาแอสแซมบลี้ 8085

โปรแกรมข้างล่างนี้ชื่อ EMU85.HEX เป็นโปรแกรมทดลองสำหรับไมโครโปรเซสเซอร์ 8085 กับวงจรทดลองในรูปที่ 4.16

Address	Label	Opcode	Mnemonic
0000	Start:	3E 89	MVI A,89h
0002		D3 13	OUT 13h
0004		3E FF	MVI A,FFh
0006		D3 10	OUT 10h
0008		3E 01	MVI A,01h
000A	Loop1:	D3 10	OUT 10h
000C		CD 13 00	Loop2
000F		07	RLC
0010		C3 0A 00	JMP Loop1
0013	Loop2:	F5	PUSH PSW
0014		21 01 00	LXI H,0001h
0017		2B	DCX H
0018		7C	MOV A,H
0019	End:	B5	ORA L

จากนั้นเลือกไปที่เมนู RUN ดังรูปที่ 4.18 โดยจะรันโปรแกรมแบบทั้งหมดหรือรันโปรแกรมแบบที่ละคำสั่งโดยกดคีย์ F8 ก็ให้เริ่มรันที่ตำแหน่ง 0000h ถึง 0020h จะสังเกตเห็นไฟ LED วิ่งวนเรื่อย เมื่อกดคีย์ ESC โปรแกรมทดลองก็จะหยุดทำงาน

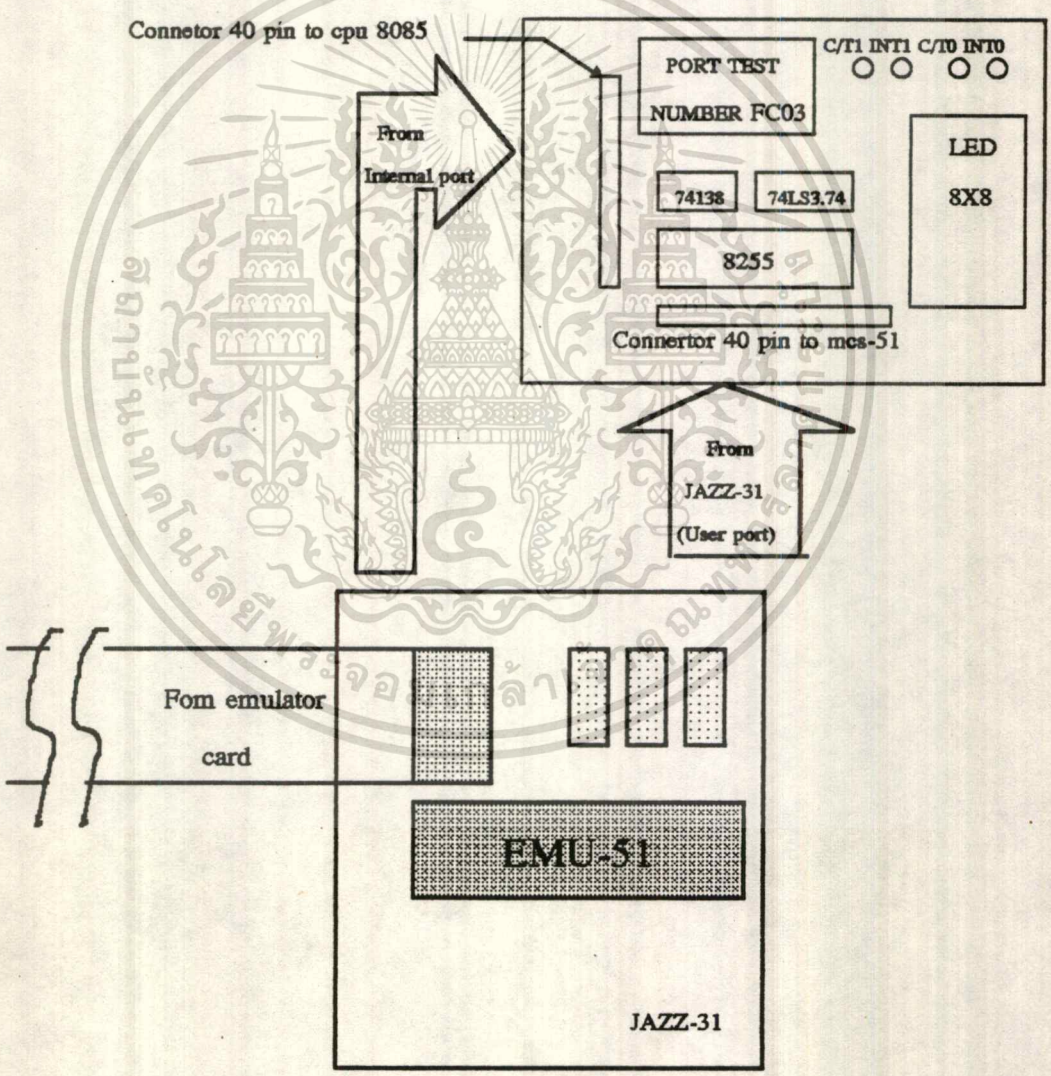


รูปที่ 4.18 แสดงเมนูการรันโปรแกรม EMU85.HEX

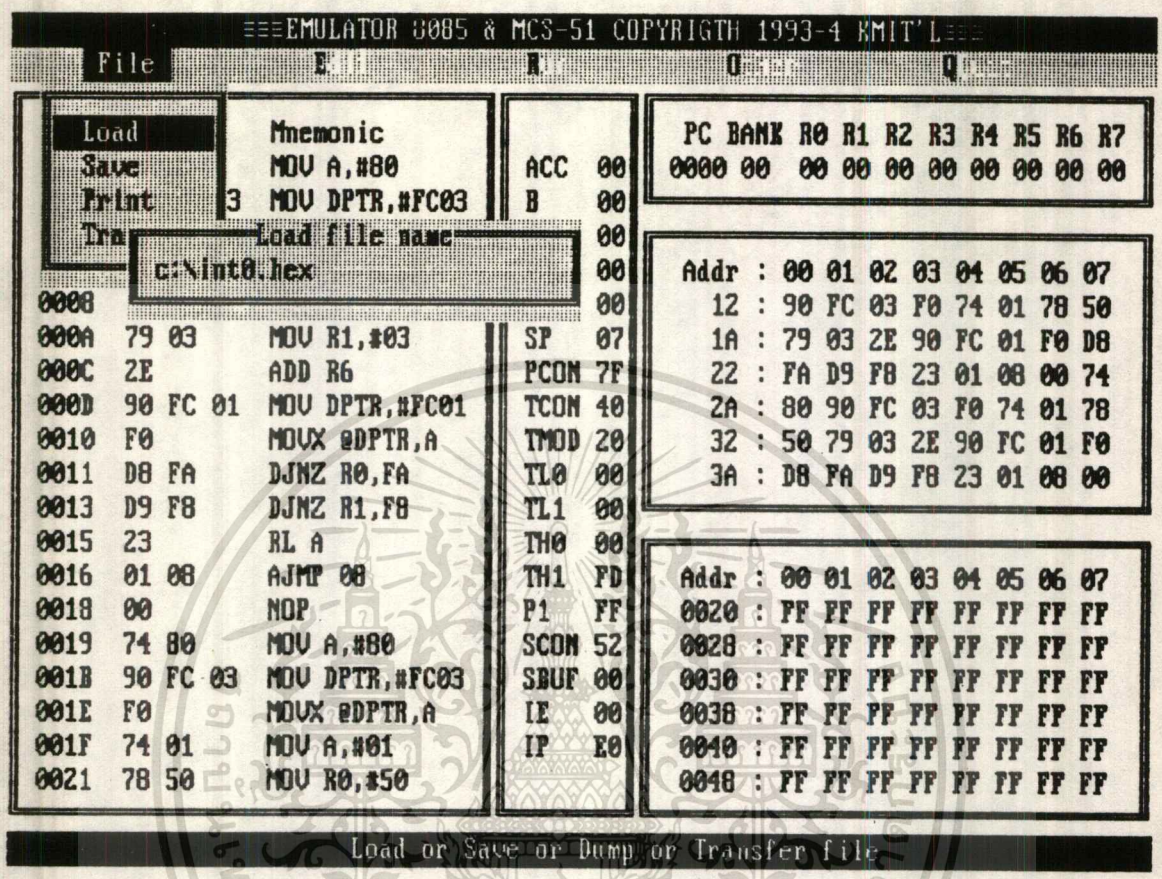
4.3.2 โปรแกรมและวงจรฮาร์ดแวร์เกี่ยวกับไมโครคอนโทรลเลอร์ MCS-51

- โปรแกรมทดลองการอินเตอร์รัพต์

ให้ผู้ใช้ต่อสาย 40 พิน(emulator cpu 8085 & mcs-51) จากการ์ดเข้ากับเครื่อง jazz-31 สาย 40 พิน (user port) จากเครื่อง jazz-31 กับวงจรทดลอง (connector emulator for mcs-51) สาย 16 พิน(internal port) จากเครื่อง jazz-31 เข้ากับวงจรทดลอง (connector emulator for cpu 8085) ดังรูปที่ 4.19 และให้ผู้ใช้เลือกเมนูของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 แล้วเรียกใช้โปรแกรม int0.hex ดังรูปที่ 4.20 โปรแกรมนี้เป็นโปรแกรมแสดงคำว่า " EMU-51" ที่ส่วนแสดงผลของเครื่อง jazz-31 และรอรับการอินเทอร์รัพ

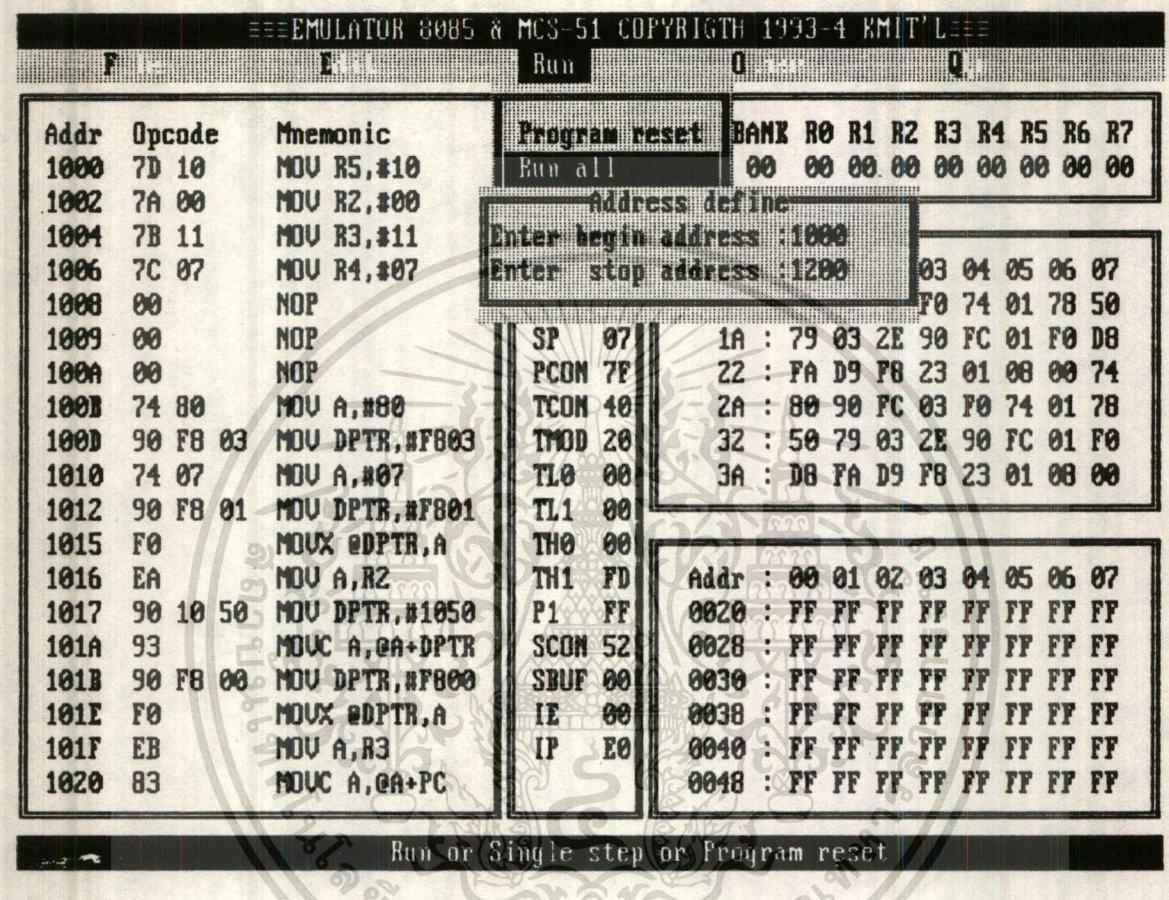


รูปที่ 4.19 แสดงการเชื่อมต่ออีมูลเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 กับวงจรทดลอง

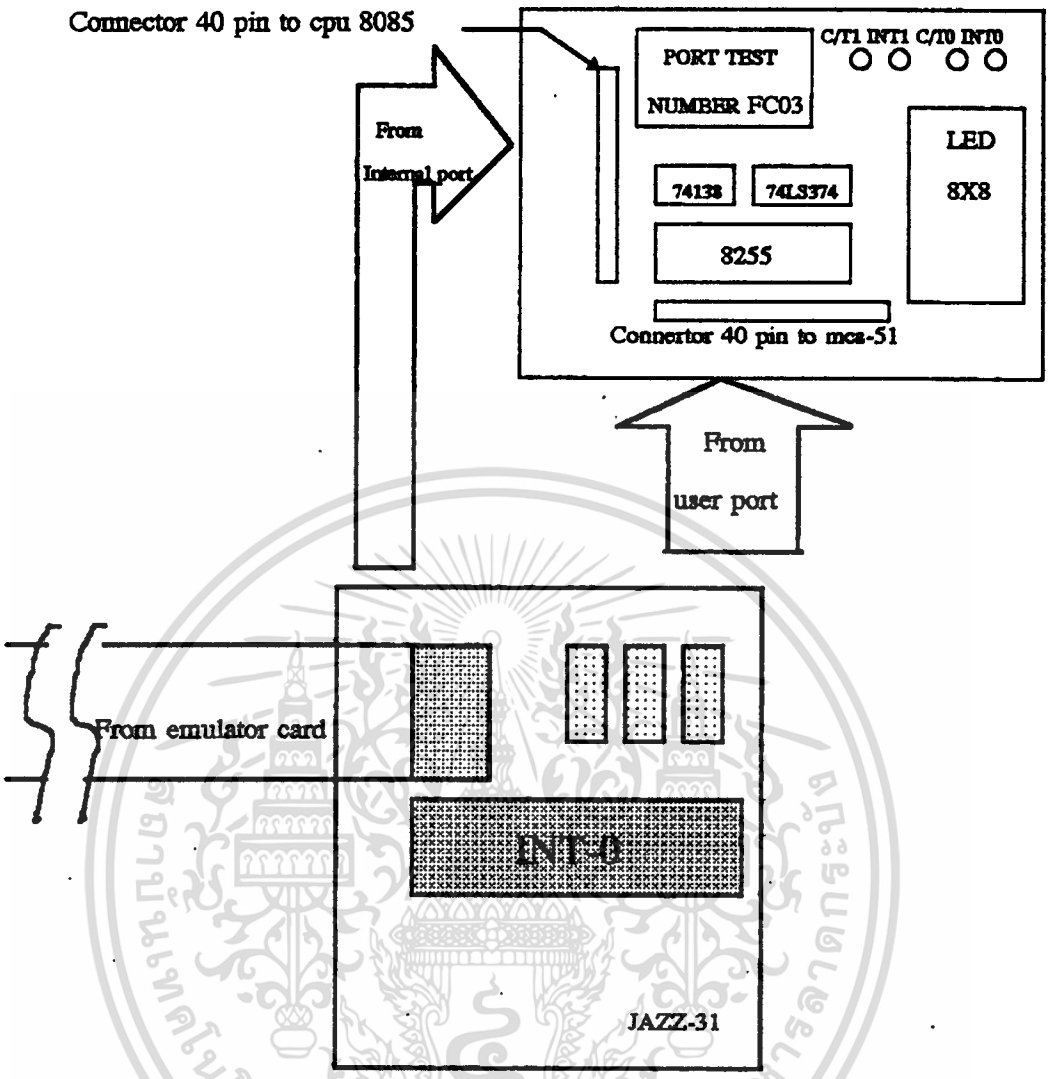


รูปที่ 4.20 แสดงการเรียกใช้โปรแกรมภาษาแอสแซมบลี้ MCS-51

จากนั้นเลือกไปที่เมนู RUN ดังรูปที่ 2.21 โดยจะรันโปรแกรมแบบทั้งหมดหรือรันโปรแกรมแบบที่ละคำสั่งโดยกดคีย์ F8 ก็ให้เริ่มรันที่ตำแหน่ง 1000h ถึง 1200h จะเห็นคำว่า "EMU-51" ที่ส่วนแสดงผลเมื่อกดสวิตช์ INT0 ที่วงจรทดลอง โปรแกรมหลักจะกระโดดไปทำงานที่ส่วนโปรแกรมบริการอินเทอร์รัพ INT0 ซึ่งจะแสดงคำว่า "INT0" ดังรูปที่ 4.22 และเมื่อบริการการอินเทอร์รัพเสร็จแล้วก็จะกลับมาแสดงคำว่า "EMU-51" ในโปรแกรมหลักต่อ ถ้าเกิดการอินเทอร์รัพแบบใดโปรแกรมจะแสดงคำนั้น ทั้งนี้การอินเทอร์รัพสามารถกำหนดให้เกิดกับชนิดใดบ้างโดยการกำหนดค่าในรีจีสเตอร์ IE (Modify registor) เช่น ให้ IE เท่ากับ 0x9Fh คืออนุญาตให้เกิดการอินเทอร์รัพทุกชนิด เมื่อกดคีย์ ESC โปรแกรมทดลองก็จะหยุดทำงาน



รูปที่ 4.21 แสดงเมนูการรันโปรแกรม int0.hex



รูปที่ 4.22 แสดงวงจรเมื่อเกิดการอินเตอร์รัพต์ int0

โปรแกรมข้างล่างนี้ชื่อ INT0.HEX เป็นโปรแกรมทดลองสำหรับไมโครคอนโทรลเลอร์ MCS-51 กับวงจรทดลองในรูปที่ 4.19

Address	Label	Opcode	Mnemonic
0000	Start:	00	NOP
0001		00	NOP
0002		00	NOP
0003		12 01 00	LCALL INT0
0006		32	RETI

Address	Label	Opcode	Mnemonic
0007		00	NOP
0008		00	NOP
0009		00	NOP
000A		00	NOP
000B		12 03 00	LCALL C
000E		32	RETI
000F		00	NOP
0010		00	NOP
0011		00	NOP
0012		00	NOP
0013		12 02 00	LCALL 0200 INT1
0016		32	RETI
0017		00	NOP
0018		00	NOP
0019		00	NOP
001A		00	NOP
001B		12 04 00	LCALL 0400 C
001E		32	RETI
001F		00	NOP
0020		00	NOP
0021		00	NOP
0022		00	NOP
0023		12 0500	LCALL 0500 R1
0026		32	RETI
0027		00	NOP
0028		00	NOP
0029		00	NOP
002A		00	NOP
002B		00	NOP
002C		00	NOP

Address	Label	Opcode	Mnemonic
002D		00	NOP
002E		00	NOP
002F		00	NOP
0100	INT0:	7D 10	MOV R5,#10
0102	Loop1:	7A 00	MOV R2,#00
0104		7B 11	MOV R3,#11
0106		7C 07	MOV R4,#07
0108		74 80	MOV A,#80
010A		90 F8 03	MOV DPTR,CONP
010D	Loop2:	74 07	MOV A,#07
010F		90 F8 01	MOV DPTR,PortA
0112		F0	MOVX @DPTR,A
0113		EA	MOV A,R2
0114		90 06 00	MOV DPTR,Font1
0117		93	MOVC A,@A+DPTR
0118		90 F8 00	MOV DPTR,PortA
011B		F0	MOVX @DPTR,A
011C		EB	MOV A,R3
011D		83	MOVC A,@A+PC
011E		90 F8 01	MOV DPTR,PortB
0121		F0	MOVX @DPTR,A
0122		0B	INC R3
0123		0A	INC R2
0124		DC E7	DJNZ R4,Loop2
0126		DD DA	DJNZ R5,DA Loop1
0128		22	RET
0129		00	NOP
012A		00	NOP
012B		00	NOP
012C		00	NOP

Address	Label	Opcode	Mnemonic
012D		00	NOP
012E		00	NOP
012F	Digit1:	00	NOP
0130		01 02	AJMP 02
0132		03	RR A
0133		04	INC A
0134		05 03	INC 03
0200	C/T0:	7D 10	MOV R5,#10
0202	Loop1:	7A 00	MOV R2,#00
0204		7B 11	MOV R3,#11
0206		7C 07	MOV R4,#07
0208		74 80	MOV A,#80
020A		90 F8 03	MOV DPTR,CONP
020D	Loop2:	74 07	MOV A,#07
020F		90 F8 01	MOV DPTR,PortB
0212		F0	MOVX @DPTR,A
0213		EA	MOV A,R2
0214		90 06 06	MOV DPTR,Font2
0217		93	MOVC A,@A+DPTR
0218		90 F8 00	MOV DPTR,PortA
021B		F0	MOVX @DPTR,A
021C		EB	MOV A,R3
021D		83	MOVC A,@A+PC
021E		90 F8 01	MOV DPTR,PortB
0221		F0	MOVX @DPTR,A
0222		0B	INC R3
0223		0A	INC R2
0224		DC E7	DJNZ R4,Loop2
0226		DD DA	DJNZ R5,Loop1

Address	Label	Opcode	Mnemonic
0228		22	RET
0229		00	NOP
022A		00	NOP
022B		00	NOP
022C		00	NOP
022D		00	NOP
022E		00	NOP
022F	Digit2:0	00	NOP
0230		01 02	AJMP 02
0232		03	RR A
0233		04	INC A
0234		05	INC 05
0300	INT1:	7D 10	MOV R5,#10
0302	Loop1:	7A 00	MOV R2,#00
0304		7B 11	MOV R3,#11
0306		7C 07	MOV R4,#07
0308		74 80	MOV A,#80
030A		90 F8 03	MOV DPTR,CONP
030D	Loop2:	74 O7	MOV A,#07
030F		90 F8 01	MOV DPTR,PortB
0312		F0	MOVX @DPTR,A
0313		EA	MOV A,R2
0314		90 06 0C	MOV DPTR,Font3
0317		93	MOVC A,@A+DPTR
0318		90 F8 00	MOV DPTR,PortA
031B		F0	MOVX @DPTR,A
031C		EB	MOV A,R3
031D		83	MOVC A,@A+PC
031E		90 F8 01	MOV DPTR,PortB

Address	Label	Opcode	Mnemonic
0321		F0	MOVX @DPTR,A
0322		0B	INC R3
0323		0A	INC R2
0324		DC E7	DJNZ R4,Loop2
0326		DD DA	DJNZ R5,Loop1
0328		22	RET
O329		00	NOP
O32A		00	NOP
O32B		00	NOP
O32C		00	NOP
002D		00	NOP
032E		00	NOP
032F	Digit3:	00	NOP
0330		01 02	AJMP 02
0332		03	RR A
0333		04	INC A
0334		05	INC 05
0400	C/T1:	7D 10	MOV R5,#10
0402	Loop1:	7A 00	MOV R2,#00
0404		7B 11	MOV R3,#11
0406		7C 07	MOV R4,#07
0408		74 80	MOV A,80
040A		90 F8 03	MOV DPTR,CONP
040D	Loop2:	74 07	MOV A,#07
040F		90 F8 01	MOV DPTR,PortB
0412		F0	MOVX @DPTR,A
0413		EA	MOV A,R22
0414		90 06 12	MOV DPTR,Font4

Address	Label	Opcode	Mnemonic
0417		93	MOVC A,@A+DPTR
0418		90 F8 00	MOV DPTR,PortA
041B		F0	MOVX @DPTR,A
041C		EB	MOV A,R3
041D		83	MOVC A,@A+PPC
041E		90 F8 01	MOV DPTR,PortB
0421		F0	MOVX @DPTR,A
0422		0B	INC R3
0423		0A	INC R2
0424		DC E7	DJNZ R4,Loop2
0426		DD DA	DJNZ R5,Loop1
0428		22	RET
0429		00	NOP
042A		00	NOP
042B		00	NOP
042C		00	NOP
042D		00	NOP
042E		00	NOP
042F	Digit4:	00	NOP
0430		01 02	AJMP 02
0432		03	RR A
0433		04	INC A
0434		05	INC 05
0500	R1:	7D 00	MOV R5,#00
0502	Loop1:	7A 00	MOV R2,#00
0504		7B 11	MOV R3,#11
0506		7C 07	MOV R4,#07
0508		74 80	MOV A,#80
050A		90 F8 03	MOV DPTR,#CONP
050D	Loop2:	74 07	MOV A,#07
050F		90 F8 01	MOV DPTR,PortB

Address	Label	Opcode	Mnemonic
0512		F0	MOVX @DPTR,A
0513		EA	MOV A,R2
0514		90 06 19	MOV DPTR,Font5
0517		93	MOVC A,@A+DPTR
0518		90 F8 00	MOV DPTR,PortA
051B		F0	MOVX @DPTR,A
051C		EB	MOV A,R3
051D		83	MOVC A,@A+PC
051E		90	MOV DPTR,PortB
0521		F0	MOVX @DPTR,A
0522		0B	INC R3
0523		0A	INC R2
0524		DC E7	DJNZ R4,Loop2
0526		DD DA	DJNZ R5,Loop1
0528		22	RET
0529		00	NOP
052A		00	NOP
052B		00	NOP
052C		00	NOP
052D		00	NOP
052E		00	NOP
052F	Digit5:	00	NOP
0530		10 02	AJMP 02
0532		03	RR A
0533		04	INC A
0534		05	INC 05
0600	Font1:	00	NOP
0601		06	INC @R0
0602		37	ADDC A,@R1

Address	Label	Opcode	Mnemonic
0603		B1 3F	ACALL 3F
0605		00	NOP
0606	Font2:	00	NOP
0607		06	INC @R0
0608		37	ADDC A,@R1
0609		B1 06	ACALL 06
060B		00	NOP
060C	Font3:	00	NOP
060D		39	ADDC A,R1
060E		31 40	ACALL 40
0610		3F	ADDC A,R7
0611		00	NOP
0612	Font4:	00	NOP
0613		39	ADDC A,R1
0614		31 40	ACALL 40
0616		06	INC @R0
0617		00	NOP
0618		00	NOP
0619	Font5:	73	JMP @A+DPTR
061A		06	INC @R0
061B		40 31	JC 31
061D		06	INC @R0
061E		80 00	SJMP 00
0620		00	NOP
0621		00	NOP
0622		00	NOP
0623		00	NOP
0624		00	NOP
0625		00	NOP
0626		00	NOP

Address	Label	Opcode	Mnemonic
0627		00	NOP
0628		00	NOP
0629		00	NOP
062A		00	NOP
062B		00	NOP

/*********** MAIN PROGRAM **********/

Address	Label	Opcode	Mnemonic
1000	Main:	7D 10	MOV R5,#10
1002	Loop1:	7A 00	MOV R2,#00
1004		7B 11	MOV R3,#11
1006		7C 07	MOV R4,#07
1008		00	NOP
1009		00	NOP
100A		00	NOP
100B		74 80	MOV A,#80
100D		90 F8 03	MOV DPTR,CONP
1010	Loop2:	74 07	MOV A,#07
1012		90 F8 01	MOV DPTR,PortB
1015		F0	MOVX @DPTR,A
1016		EA	MOV A,R2
1017		90 10 50	MOV DPTR,Font6
101A		93	MOVC A,@A+DPTR
101B		90 F8 00	MOV DPTR,PortA
101E		F0	MOVX @DPTR,A
101F		EB	MOV A,R3
1020		83	MOVC A,@A+PC
1021		90 F8 01	MOV DPTR,PortB
1024		F0	MOVX @DPTR,A
1025		0B	INC R3

Address	Label	Opcode	Mnemonic
1026		0A	INC R2
1027		DC E7	DJNZ R4,Loop2
1029		DD D7	DJNZ R5,Loop1
102B		01 00	AJMP 00
102D		00	NOP
102E		00	NOP
102F		00	NOP
1030		00	NOP
1031		00	NOP
1032	Digit0:	00	NOP
1033		01 02	AJMP 02
1035		03	RR A
1036		04	INC A
1037		05	INC 05
1050	Font6:	79 37	MOV R1,#37
1052		3E	ADDC A,R6
1053		40 6D	JC 8D
1055		86	MOV 85,@R0

/********** END MAIN PROGRAM **********/

บทที่ 5

บทวิจารณ์และสรุปผล

จากการพัฒนาและสร้างวงจรทดสอบภายนอกดังบทที่ 3 ติดต่อกับโครงงานแล้วทำการเขียนโปรแกรมทดสอบ และทำการแอสแซมเบลอร์โปรแกรมที่เขียนโดยคำสั่งของไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS-51 โดยใช้โปรแกรม SXA51 หรือ CROSS16 ก่อนที่จะโหลดเข้ามาเก็บไว้ในหน่วยความจำของระบบไมโครคอมพิวเตอร์แล้วทดสอบดู จากการทดลองใช้อีมูเลเตอร์ในการพัฒนาวงจรสามารถดำเนินการได้อีกทั้งยังสามารถทำการแก้ไขโปรแกรมได้โดยตรงทำให้เกิดความรวดเร็วในการที่จะพัฒนาโปรแกรม แต่ก็มีข้อจำกัดในการใช้งานอีมูเลเตอร์อยู่หลายประการซึ่งต้องจะต้องพิจารณาเพื่อนำไปปรับปรุงในภายหลัง

5.1 ข้อจำกัดทางด้านฮาร์ดแวร์

เนื่องจากลักษณะของฮาร์ดแวร์ที่ได้จัดสร้างขึ้นมานั้น จะเห็นว่ามีส่วนที่ทำหน้าที่หลักอยู่ 3 ส่วนดังนี้คือ ส่วนแรกคือส่วนที่ทำการถอดรหัสแอดเดรสที่ใช้ติดต่อกับระบบไมโครคอมพิวเตอร์ ส่วนที่สองนั้นจะเป็นวงจรในส่วนที่ทำหน้าที่แทนขาสัญญาณอินพุตและขาเอาท์พุต ส่วนสุดท้ายก็เป็นส่วนที่แทนขาสัญญาณที่เป็นแบบสองทิศทาง สำหรับข้อจำกัดทางด้านฮาร์ดแวร์นี้ก็คือ ความรวดเร็วในการทำงานของไอซีแต่ละตัว ในการที่จะส่งสภาวะตามขาสัญญาณต่างๆให้เหมือนกับไมโครโปรเซสเซอร์และไมโครคอนโทรลเลอร์ตัวจริงและการทำงานในแต่ละพอร์ตนั้นสามารถที่จะส่งหรือรับข้อมูลได้ครั้งละ 8 บิต ซึ่งไม่สามารถที่จะเขียนโปรแกรมให้ควบคุมทีละ 1 บิตได้ ดังนั้นในการที่จะทำให้สัญญาณใดๆแอกทีฟหรือไม่ให้แอกทีฟนั้นก็จะต้องทำการตรวจดูก่อนว่า ขาสัญญาณนั้นอยู่ในกลุ่มของพอร์ตใด แล้วจึงทำการส่งค่าของข้อมูลที่ทำให้บิตนั้นแอกทีฟออกไปซึ่งทำให้โปรแกรมในการควบคุมการทำงานทำได้ช้าและไม่สะดวกในการเขียนโปรแกรม ข้อจำกัดอีกด้านคือการสื่อสารข้อมูลต้องใช้พอร์ทสื่อสารข้อมูลจากคอมพิวเตอร์ ทำให้อัตราการส่งข้อมูลจำกัดเพียง 9,600 บิตต่อวินาที และต่ำสุดคือ 110 บิตต่อวินาที ถ้าหากการส่งข้อมูลเกินกว่าที่กำหนดนี้จะมีการผิดพลาดเกิดขึ้น

5.2 ข้อจำกัดทางด้านซอฟต์แวร์

ในการออกแบบโครงสร้างทางด้านซอฟต์แวร์นั้นกล่าวโดยสรุปก็คือ ได้ออกแบบเป็น 2 ส่วนคือ ในส่วนแรกนั้นจะเป็นการออกแบบระบบโครงสร้างของระบบเมนูเพื่อทำให้ผู้ใช้ ใช้นำไปใช้งานได้ง่าย โดยใช้เวลาศึกษาวิธีการใช้โปรแกรมไม่นานนัก ก็สามารถนำโปรแกรมไปใช้ได้อย่างมีประสิทธิภาพ เพราะระบบโครงสร้างของเมนูที่เขียนขึ้นมาเป็นโปรแกรมที่เขียนเป็นแบบ Pull-up และ Pul-down menu ซึ่งง่ายต่อการใช้งาน เพราะได้แบ่งเป็นหัวข้อไว้แล้วว่าทำงานเกี่ยวกับอะไร โดยแบ่งเป็นหัวข้อเมนูหลักได้ 5 หัวข้อดังนี้ คือ File, Edit, Run, Other และ Quit ซึ่งคำอธิบายในการใช้งานสามารถเรียกดูได้จากโปรแกรมคำอธิบายในการช่วยใช้งาน ส่วนที่สองของการออกแบบโครงสร้างของโปรแกรมนั้นก็คือ ส่วนที่ทำหน้าที่เป็นฟังก์ชันการทำงานต่างๆในแต่ละหัวข้อเลือกเมนู ซึ่งจะเป็นการกำหนดรายละเอียดว่าในแต่ละฟังก์ชันนั้นทำงานอย่างไร โดยได้ออกแบบเป็นโฟลว์ชาร์ทซึ่งได้กล่าวไปแล้วในบทที่ 3 ซึ่งเกี่ยวกับการออกแบบโครงสร้างของโปรแกรมที่ใช้ในการติดต่อและควบคุมอีมูเลเตอร์

ส่วนข้อจำกัดทางด้านซอฟต์แวร์ที่มีนั้นก็เป็นทางด้านความเร็วของการรันโปรแกรม เนื่องจากการเขียนฟังก์ชันที่ทำหน้าที่เป็นชุดคำสั่งของไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์นั้นใช้ภาษาระดับสูงเขียน ดังนั้นจึงต้องเสียเวลาในการปฏิบัติตามคำสั่งในฟังก์ชันนั้นๆ ดังนั้นโอกาสที่จะโปรแกรมที่เขียนขึ้นมาเพื่อแทนชุดคำสั่งนั้นทำงานได้ตรงเวลาจึงแทบไม่มีเลย แต่ถ้าเรานำโปรแกรมชุดคำสั่งของไมโครโปรเซสเซอร์ หรือไมโครคอนโทรลเลอร์นี้มาทำการรันทีละคำสั่ง ดังนั้นจึงไม่จำเป็นต้องรันตามเวลาจริงเพราะเป็นการทำงานทีละคำสั่งหรือทีละฟังก์ชันในภาษาระดับสูงนั่นเอง ซึ่งจะต้องแสดงข้อมูลในรีจีสเตอร์หรือหน่วยความจำข้อมูลภายในที่เปลี่ยนไปตามการปฏิบัติตามชุดคำสั่งนั้นๆ ด้วย และข้อจำกัดของการเขียนโปรแกรมอีกอย่างหนึ่งก็คือ การส่งสัญญาณออกไปตามขาสัญญาณต่างๆ นั้นจะต้องส่งออกไปทีละพอร์ต หรือ 8 บิตไม่สามารถจะควบคุมทีละ 1 บิตได้

5.3 บทสรุป

โครงงานชิ้นนี้ มีหน้าที่หลักในการที่จะจำลองหรือเลียนแบบการทำงานของไมโครโปรเซสเซอร์ 8085 และไมโครคอนโทรลเลอร์ MCS 51 เพื่อที่จะนำเอาอีมูเลเตอร์นี้ไปใช้ในการวิเคราะห์ ตรวจสอบ ตรวจซ่อม ระบบควบคุม หรือนำไปใช้ในการทดลองพัฒนาระบบที่ใช้ไมโครโปรเซสเซอร์ 8085 หรือ ไมโครคอนโทรลเลอร์ MCS-51 เป็นหน่วยประมวลกลางเพราะจะทำให้สามารถช่วยลดเวลาในการวิเคราะห์ตรวจซ่อมลงไปได้มาก เนื่องจากว่าสามารถที่จะนำไปช่วยวิเคราะห์ข้อผิดพลาดที่เกิดขึ้นทั้งจากทางด้าน ฮาร์ดแวร์และ

ซอฟต์แวร์ได้อย่างสะดวกและรวดเร็ว และลักษณะการใช้งานโปรแกรมอีมูเลเตอร์ก็ถูกออกแบบให้ใช้งานในระบบ Pop up และ Pull down menu ซึ่งทำให้ง่ายต่อการนำไปใช้งาน ถึงแม้ว่าอีมูเลเตอร์ที่สร้างขึ้นมานี้ สามารถที่นำจะใช้งานได้ แต่ในการทำงานของอีมูเลเตอร์ตัวนี้ยังจะ ต้องการการพัฒนาอีกมาก เพื่อให้อีมูเลเตอร์ตัวนี้มีประสิทธิภาพในการทำงานเพิ่มมากขึ้น เช่นการพัฒนาในเรื่องของ การทำงานในแบบตรงตามเวลาจริง(Real time)ซึ่งเป็นสิ่งที่จำเป็นถ้าต้องการให้โครงงานนี้มีความสามารถ ในการทำงานในลักษณะ อินเซอร์กิต อีมูเลเตอร์ (IN-CIRCUIT EMULATOR) ได้ซึ่งจะต้องมีการพัฒนาต่อไป

ภาคผนวก

ปริญญานิพนธ์ฉบับนี้ได้แบ่งภาคผนวกเป็น 2 ส่วนคือ ภาคผนวก ก. ภาคผนวก ข. โดยภาคผนวก ก. นั้นจะเป็นโปรแกรมของอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ 8085 ซึ่งอยู่ไฟล์ข้อมูลชื่อ EMU8085.C และโปรแกรมของอีมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 ซึ่งอยู่ในไฟล์ข้อมูลชื่อ EMUMCS.C และภาคผนวก ข. เป็นรายละเอียดเพิ่มเติมเกี่ยวกับไมโครคอนโทรลเลอร์ MCS-51

ภาคผนวก ก.

```
/*************************************************************/
/********************* Include file and initialize ***************/
/*************************************************************/
#include <stdio.h>
#include <conio.h>
#include "c:\mcs\menu.h"
#include "c:\mcs\display.h"

extern unsigned char *ex_ram,*pcrom,*in_ram,*sfr,*pcram;
extern int arrow_choice ,key_geted;

unsigned char r_acc,r_b,r_psw,r_dpl,r_dph,r_sp,r_pcon,r_tcon,r_tmod,r_tl0,
              r_tl1,r_th0,r_th1,r_p0,r_p1,r_p2,r_p3,r_scon,r_sbuf,r_ie,
              r_ip,reg_r0,reg_r1,reg_r2,reg_r3,reg_r4,reg_r5,reg_r6,reg_r7;
unsigned char  psw,reg_a,reg_b,reg_c,reg_d,reg_e,reg_h,reg_l;
unsigned char *ex_mem,*pro_mem,*in_mem,*sfr,*pcram;
unsigned int s_point,pc,p_break;
char over_f,aux_f,parity_f,carry_f,rs0,rs1,f0,res_f;
char screen[80*25*2];
int  xpos,ypos,bank;

MENU   menu0[] = {
        OPTION(" File "," Load or Save or Dump or Transfer file ",'F')
        OPTION(" Edit "," Modify or view program in memory ",'E')
        OPTION(" Run "," Run or Single step or Program reset",'R')
        OPTION(" Other"," Bit value or Test port ",'O')
        OPTION(" Quit ","Display Version or Os shell and Exit to dos",'Q')
        OPTIONEND
        };
char *file[]={
        " Load    ",
        " Save    ",
        " Print   ",
        " Transfer ",
};
char *t_mode[]={
        " Load from EX-ROM  ",
```

```
            " store into EX-RAM ",
            };
char *disp_file[]={
            " Hexadecimal ",
            " Mnemonic    ",
            };
char *edit[]={
            " Modify ",
          " View   ",
            };
char  *view_mode[]={
            " Program ",
            " Data    ",
            };
char *run[]={
          " Program reset ",
            " Run all       ",
            " Single step   ",
            };
char  *other[]={
            " Bit value   ",
            " Test port   ",
            };
char *quit[]={
            " Os_shell    ",
            " Exit to dos ",
            };
char *testport[]={
            " port A [300h] ",
            " port B [301h] ",
            " port C [302h] ",
            " port D [303h] ",
            " port E [304h] ",
            " port F [305h] ",
            " port G [306h] ",
            " port H [307h] ",
            };
```

```
char *type_port[]={
        " Input port ",
        " Output port",
        };
char *modi_datamc[]={
        " Program memory ",
      " Sfr          ",
        " Int. memory  ",
        " Ext. memory  ",
        };
char *view_datamc[]={
        " Program memory ",
        " Int. memory  ",
        " Ext. memory  ",
        };
char *emu_type[]={
        "  Emulator for 8085    ",
        "  Emulator for MCS-51   ",
        };
char *bit[]={
        " Lower  Addr [00-7F]  ",
        " Higher Addr [80-FF]  "
        };

/*****************************************************************/
/************************ Start main program *********************/
/*****************************************************************/

main()
{
        int choice,chk;
        int back, fore, mode;

        xpos=wherex();
        ypos=wherey();
        gettext(1,1,80,25,screen);
```

```
    arrow_choice = 0;
    key_geted = 0;

/*************************************************************/
/********************  Allocate memory  **********************/
/*************************************************************/

sfr   = (unsigned char *) malloc(256*sizeof(unsigned char));
in_mem= (unsigned char *) malloc(256*sizeof(unsigned char));
pro_mem = (unsigned char *) malloc(NUM*sizeof(unsigned char));
ex_mem = (unsigned char *) malloc(NUM*sizeof(unsigned char));
if(!ex_mem && !pro_mem && !sfr && !in_mem)
        {
        printf("out of memory\n");
        exit(0);
        }

/*************************************************************/
/*************  Make menu and window system  *****************/
/*************************************************************/

make_window(0,"PROJECT",8,19,18,60,BORDER);
make_window(1,"Out port sevice ",16,19,18,60, BORDER);
make_window(2,"Modify register service",5,15,7,64,BORDER);
make_window(3," CONFIRM ",10,20,12,59, BORDER);
make_window(4,"Load file name",4,3,6,40,BORDER);
make_window(5,"Save file name",5,3,7,40,BORDER);
make_window(6,"View edit ",8,21,10,54,BORDER);
make_window(7,"Edit Function",4,19,7,60,BORDER);
make_window(8,"Dump data",8,10,10,40,BORDER);
make_window(9,"Address define",5,32,8,63,BORDER);
make_window(10,"Bit value",4,24,22,42,BORDER);
make_window(11,"Address define",5,3,8,34,BORDER);
make_window(12,"Help Function",3,1,24,52,BORDER);
make_window(13,"Test port information",18,25,20,62,BORDER);
make_menu(0, file,"lsdt",4,2,2,BORDER);
```

```
make_menu(1, edit,"mv",2,2,18,BORDER);
make_menu(2, run,"prs",3,2,33,BORDER);
make_menu(3, other,"bt",2,2,46,BORDER);
make_menu(4, quit,"oe",2,2,62,BORDER);
make_menu(5, testport,"abcdefgh",8,5,40,BORDER);
make_menu(6, type_port,"io",2,9,31,BORDER);
make_menu(7, t_mode,"ps",2,7,3,BORDER);
make_menu(8, disp_file,"hm",2,6,3,BORDER);
make_menu(9, view_mode,"pd",2,5,20,BORDER);
make_menu(10, modi_datamc,"psie",4,4,25,BORDER);
make_menu(11, view_datamc,"pie",3,4,25,BORDER);
make_menu(12, emu_type,"em",2,13,27,BORDER);
make_menu(13, modi_datamc,"pise",4,8,5,BORDER);
make_menu(14, bit,"lh",2,4,43,BORDER);clrscr();
/*************************************************************/
/*********************** Draw border *************************/
/*************************************************************/
clrscr();
cursor(1,17);
putchar(C_HT);putchar(C_HT);putchar(C_HT);
Paintstr(20,1,BANNER,15,0);
cursor(1,64);
putchar(C_HT);putchar(C_HT);putchar(C_HT);
initialize();
borde();
initialmcs();

/*************************************************************/
/********************** Get video mode ***********************/
/*************************************************************/
        mode = getvmode();
        if (mode == 1 || mode == 3) {
                fore = 0;
                back = 7;
        } else {
                fore = 0;
                back = 7;
```

```
        };
/*************************************************************/
/****************** Menu start and display *******************/
/*************************************************************/
menu_start(menu0,fore,back);
 do {
        first_active(menu0,fore,back);
/*************************************************************/
/****************** Select menu system** *********************/
/*************************************************************/
        chk = menu_choice(menu0,fore,back);
/*************************************************************/
/********************** If exit to dos ***********************/
/***************  menu end unallocate memory *****************/
/*************************************************************/
        if(chk == -2) {
                menu_end();
                cls();
                cursor(1,1);
                free(ex_mem);
                free(pro_mem);
                free(sfr);
                free(in_mem);
        exit(0);
         };
 }while(TRUE);
}
/*********************************************************************/
/************************ End of main program ************************/
/*********************************************************************/
```

```
#include <stdio.h>
#include <string.h>
#include "c:\mcs\menu.h"
#include "c:\mcs\display.h"
extern unsigned char *in_mem,*ex_mem,*pro_mem,*sfr,chk_esc;
extern unsigned char  r_acc,r_b,r_psw,r_dpl,r_dph,r_sp,r_pcon,r_tcon,r_tmod,r_tl0,r_tl1,r_th0,r_th1,
                      r_p0,r_p1,r_p2,r_p3,r_scon,r_sbuf,r_ie,r_ip,reg_r0,reg_r1,reg_r2,reg_r3,reg_r4,
                      reg_r5,reg_r6,reg_r7;
extern unsigned int pc;
extern int  arrow_item,key_geted,key_recive,arrow_choice,arrow_get,bank;
int save_addr[20];
char temp[5];
/*************************************************************/
/***************** Start of menu_editmcs *********************/
/*************************************************************/
menu_editmcs()
{
        int choice;
        arrow_choice = 0;
        key_geted = 0;
        pulldown(1);
        while((choice = get_resp(1)) != -1){
                switch(choice){         /*******************************/
                        case 0: modi_datamcs(); /***** Define value *****/
                                break;          /***** with user.   *****/
                        case 1: view_datamcs(); /***** Show data in *****/
                                break;  /** program or internal memory**/
                        }               /**   or external memory.    **/
                }                       /*******************************/
        restore_video(1);
}
/*************************************************************/
/****************** End of menu_editmcs **********************/
/*************************************************************/
```

```
#include <stdio.h>
#include <string.h>
#include "c:\mcs\menu.h"
#include "c:\mcs\display.h"

extern unsigned char  *pcram,*pro_mem,*in_mem,*ex_mem,*sfr,chk_esc;
extern unsigned int pc,p_break;
extern int arrow_choice,arrow_get,arrow_item,key_geted,key_recive;
extern int arrow_item1,norm_key,key_recive1;
extern int save_addr[20];

unsigned int init_pc[10];
int index;
/*************************************************************/
/******************* Start of menu file **********************/
/*************************************************************/
menu_filemcs()
{
        int choice;
          arrow_choice = 0;
          key_geted = 0;
        pulldown(0);
        while((choice=get_resp(0)) != -1){
            ClrText(1,25,80,25);
                switch(choice){
                        case 0: loadfilemcs();          /*******************/
                                        if(chk_esc==1)  /*** Load file ***/
                                          return; /****from diskette.*****/
                                        else        /*********************/
                                          break; /*********************/
                      case 1:   savefilemcs(); /*****  Save file  ******/
                                break;          /***** to diskette.******/
                      case 2: dump_filemcs();/*****  Dump file ******/
                                break;        /**** hex or mnemonic.***/
                      case 3: transfermcs();  /**** transfer data  *****/
                                break;        /****  with  memory.*****/
                    }                       /*****************************/
```

```
        }
        restore_video(0);
        disp_pcmcs();
}

/*************************************************************/
/********************* End of menu file **********************/
/*************************************************************/
```

```
#include <stdio.h>
#include <string.h>
#include "c:\mcs\menu.h"
#include "c:\mcs\display.h"

extern unsigned char  psw,reg_a,reg_b,reg_c,reg_d,reg_e,reg_h,reg_l,*pcram;
extern unsigned int s_point,pc,p_break;
extern int arrow_choice,arrow_get,key_geted,save_addr[20];
extern char sign_f,zero_f,aux_f,parity_f,carry_f;

unsigned char chk_esc;

/*********************************************************************/
/********************** start of menu_runmcs *************************/
/*********************************************************************/

menu_runmcs()
{
        int choice,test;
                arrow_choice = 0;
                key_geted = 0;
        pulldown(2);
        while((choice=get_resp(2)) != -1){
                switch(choice){
                        case 0: restore_video(2);          /************************/
                                initialize();       /*****Reset all register****/
                                initialmcs();       /*****  first status.  *****/
                                pulldown(2);       /************************/
                                hide_cursor();
                                break;
                        case 1: test = run_allmcs();        /************************/
                                if(test==1)                 /****  Run all address ****/
                                initialize();       /**** with user define.*****/
                                hide_cursor();      /************************/
                                return;
                        case 2: stepmcs();                  /************************/
                                hide_cursor();      /***** Run singal step.*****/
```

```
                return;
            }
        }
        restore_video(2);
}
/*************************************************************************/
/*********************** End of menu_runmcs ***************************/
/*************************************************************************/
```

```
#include "c:\mcs\display.h"
#include "c:\mcs\menu.h"

extern unsigned char  psw,reg_a,reg_b,reg_c,reg_d,reg_e,reg_h,reg_l;
extern unsigned char  *in_mem,*ex_mem,*pro_mem;
extern unsigned int s_point,pc;
extern int arrow_choice,arrow_get,key_geted,arrow_item,key_recive, arrow_item1,key_recive1;
extern char sign_f,zero_f,aux_f,parity_f,carry_f;

int norm_key,port_type;

/*****************************************************************************/
/****************************** Start of menu_other ***************************/
/*****************************************************************************/

menu_othermcs()
{
        int choice;
                arrow_choice = 0;
                key_geted = 0;
        pulldown(3);
        while((choice=get_resp(3)) != -1){
                switch(choice){                 /*****************************/
                  case 0: bit_value();          /*****  Show bit address. *****/
                          break;                /*****************************/
                  case 1: menu_testport();      /*****  TEST Input &    ******/
                          break;                /*****    Output Port.     ******/
                  }                             /************** **************/
        }
        restore_video(3);
}
/*****************************************************************************/
/***************************** End of menu_other *****************************/
/*****************************************************************************/
```

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <ctype.h>
#include <process.h>
#include <stdlib.h>
#include <dir.h>
#include <string.h>
#include "c:\mcs\display.h"

extern int arrow_choice,key_geted;
extern char *screen[80*25*2];
extern int xpos,ypos;

/*************************************************************/
/********************** Start of menu_quit **********************/
/********************** ***********************************/

menu_quit()
{
        int choice;
        char ch;
                arrow_choice=0;
                key_geted=0;
        pulldown(4);
        while((choice=get_resp(4)) != -1) {
                switch(choice) {
                        case 0: os_shell(&xpos,&ypos,screen);   /**********************/
                                hide_cursor();                  /****** Go to dos ******/
                                break;                          /**********************/
                        case 1: window_(3);window_cls(3);       /****** Exit to dos.*****/
                                window_xy(3,0,1);               /**********************/
                                window_puts(3,"Are you sure exit to dos [y/n] ? "
                                ,A_INVERSE);
                                window_xy(3,0,36);
                                ch = window_getche(3);
                                if(tolower(ch)=='y') {
```

```
                                        deactivate_win(3);
                                        restore_video(3);
                                        return -2;
                                    }else { deactivate_win(3);hide_cursor(); }
                                    break;
                                }
                            }
                        restore_video(4);
}
/*************************************************************/
/************************ End of menu_quit *******************/
/*************************************************************/
```

```
#include <stdio.h>
#include <conio.h>
#include "c:\mcs\menu.h"
#include "c:\mcs\display.h"

extern unsigned char  *pcram;
extern unsigned int s_point,p_count,p_break,select_type;

unsigned char  byte2,byte3,reg_ir,reg_im;
int  increment,last,menucount,menulen,mouse_here,*nptr,pcol,prow;
int  arrow_get,arrow_item,arrow_item1;
int  key_geted,key_recive,key_recive1;
int  arrow_choice,len_deso;
char *cptr;
char far *vid_mem;

struct menu_frame {
        int startx, endx, starty,endy;
        unsigned char *p;
        char **menu;
        char *keys;
        int border ,count;
        int active;
        }frame[MAX_FRAME],i;

struct window_frame {
        int startx, endx, starty,endy;
        int curx,cury;
        unsigned char *p;
        char *header;
        int border;
        int active;
        }frame_win[MAX_FRAME];
```

```
/*****************************************************************/
/************************ Start of menu choice *******************/
/*****************************************************************/

void menu_start(MENU s[],int fore, int back)
{
        int buttons,i,r,c,wide;
                wide = getvcols()-1;
                find_cursor(&prow,&pcol);
                c_scroll(prow,pcol-4,wide,1,0,6,back << 4);
        last = 0;
        nptr = (int *) calloc(wide,sizeof(int));
        cptr = (char *) calloc(wide,sizeof(char));

        for(menulen = i = 0; s[i].prompt;i++) {
                find_cursor(&r,&c);
                nptr[i] = c;
                cptr[i] = s[i].letter;
                colorstr(s[i].prompt,fore,back << 4);
                        if (i != 4)
                                colorstr(MARGIN,fore,back << 4);
                                menulen += s[i].plen;
        }

        /************ Active first character ****************/
        cursor(2,7);
        colorchr(s[0].letter,4,back<<4);
        cursor(2,22);
        colorchr(s[1].letter,4,back<<4);
        cursor(2,37);
        colorchr(s[2].letter,4,back<<4);
        cursor(2,51);
        colorchr(s[3].letter,4,back<<4);
        cursor(2,66);
        colorchr(s[4].letter,4,back<<4);
        menucount = 1;
```

```
        menulen += (strlen(MARGIN) * (menucount-1));
        BackColor(1,25,80,25,7);

    }
    /*********************************************************************/
    /*********************** End of menu choice **************************/
    /*********************************************************************/
```

```
/*****************************************************************
**********  The is a header file that use for display system  *********
*****************************************************************/

#define C_UL  218
#define C_UR  191
#define C_LL  192
#define C_LR  217
#define C_XD  194
#define C_XU  193
#define C_XL  195
#define C_XR  180
#define C_XX  197
#define C_H   196
#define C_V   179
#define C_ULD 201
#define C_URD 187
#define C_LLD 200
#define C_LRD 188
#define C_XDD 203
#define C_XUD 202
#define C_XLD 204
#define C_XRD 185
#define C_XXD 206
#define C_HD  205
#define C_VD  186
#define C_ULSD 213
#define C_URSD 184
#define C_LLSD 212
#define C_LRSD 190
#define C_XDSD 209
#define C_XUSD 207
#define C_XLSD 198
#define C_XRSD 181
#define C_XXSD 216
#define C_ULDS 214
#define C_URDS 183
```

```
#define C_LLDS 211
#define C_LRDS 189
#define C_XDDS 210
#define C_XUDS 208
#define C_XLDS 199
#define C_XRDS 182
#define C_XXDS 215
#define C_HATCH 176
#define C_HT    240
#define C_SPACE 255

#define VIDRAM     0xB8000000
#define BLACK      0
#define LIGHTGRAY  15
#define A_NORM       0x07
#define A_INVERSE   0x70
#define A_INTENSE    0x0F
#define A_BLINK      0x87
#define A_UNDER      0x01

#define PORT_A 768
#define PORT_B 769
#define PORT_C 770
#define PORT_D 771
#define PORT_E 772
#define PORT_F 773
#define PORT_G 774
#define PORT_H 775

#define NUM 8192
#define MAG 65536
#define HEIGHT 24
#define J1_HEIGHT 6
#define J2_HEIGHT 22
#define WIDTH 80
#define PRINT 1
#define SCREEN 0
```

```
#define P_BC 26
#define P_DE 32
#define P_HL 38
#define P_PSW 45
#define P_ACC 19
#define P_PC  4
#define P_SP  11
#define P_ROW 5
#define PH_A0 16
#define PH_AC 17

#define DISP_S gotoxy(52,P_ROW);printf("%d",sign_f);
#define DISP_Z gotoxy(55,P_ROW);printf("%d",zero_f);
#define DISP_AC gotoxy(61,P_ROW);printf("%d",aux_f);
#define DISP_P gotoxy(68,P_ROW);printf("%d",parity_f);
#define DISP_C gotoxy(74,P_ROW);printf("%d",carry_f);
#define DISP_ALL_F DISP_S;DISP_Z;DISP_AC;DISP_P;DISP_C

#define BANNER  "EMULATOR 8085 & MCS-51 COPYRIGTH 1993 KMITL"
#define BORDER  1
#define MAX_FRAME 25
#define BKSP  8
#define TAB  3
#define TRUE 1
#define MAX_COM   30

#define NOTE_KEY "F5-View int.memory F6-Modify int.memory F7-Modify sfr F8-step ESC-break"
#define NOTE_FILE1 "File not found !!!"
#define NOTE_FILE2 "Loading File ..."
#define NOTE_FILE3 "Can't open file !!! "
#define NOTE_FILE4 "Saving File ... "
#define NOTE_FILE5 "ESC-dumping cancel "
#define NOTE_FILE6 "Check printer for ready then press ENTER "
#define NOTE_FILE7 "Now dumping file ... "
#define NOTE_FILE8 "Press any key to continue dumping next page ..."
#define NOTE_KEY1 "Press ESCAPE key to break..."
```

```
#define P_COLUMN1   42
#define P_PCMCS     47
#define BANK        52
#define P_R0        56
#define P_R1        59
#define P_R2        62
#define P_R3        65
#define P_R4        68
#define P_R5        71
#define P_R6        74
#define P_R7        77
#define P_ROW1       5

#define MES3  "instruction"
#define MES4  "SFRs"
#define MES5  "internal memory"
#define MES6  "REGs & PC"
#define MES7  "external memory"
#define UP_L  218
#define UP_R  191
#define DN_L  192
#define DN_R  217
#define HR_L  196
#define VL_L  179

#define P0    0x80
#define SP    0x81
#define DPL   0x82
#define DPH   0x83
#define PCON  0x87
#define TCON  0x88
#define TMOD  0x89
#define TL0   0x8A
#define TL1   0x8B
#define TH0   0x8C
#define TH1   0x8D
#define P1    0x90
```

```
#define SCON 0x98
#define SBUF 0x99
#define P2   0xA0
#define IE   0xA8
#define P3   0xB0
#define IP   0XB8
#define B    0xF0
#define ACC  0xE0
#define PSW  0xD0
```

```
/*******************************************************
********************  menu.h  ************************
*******************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <dos.h>
#include <alloc.h>
#include <mem.h>
#include <string.h>

typedef struct {
        char  *prompt;
        int   plen;
        char *deso;
        int dlen;
        char letter;
        }MENU;
#ifdef  EXTERN
        extern
#endif

/********************  macros ************************/

#define TRUE    1
#define  HOME  71
#define  UARROW 72
#define  PAGEUP 73
#define  LARROW 75
#define  RARROW 77
#define  END    79
#define  DARROW 80
#define  PAGEDN 81

#define  F1 59
#define  F2 60
```

```
#define F3 61
#define F4 62
#define F5 63
#define F6 64
#define F7 65
#define F8 66
#define F9 67
#define F10 68

#define LBUTTON 1
#define RBOTTON 2
#define MBUTTON 4
#define SIDEWAYS 0
#define UPDOWN  1
#define SINGLEBAR 1
#define DOUBLEBAR 2
#define UPA  24
#define DOWNA 25
#define ENTER '\r'
#define ESCAPE 27

#define OPTION(a,b,c) { a , sizeof(a) - 1 , b , sizeof(b) - 1 , c } ,
#define OPTIONEND {0}

#define MARGIN "      "
#define BELL  7

#define MAXWIDE  80
#define MAXDEEP  24

#define DELTA  50
#define ifbars(x,y)     ( c = ( bars == 1 ) ? ( x ) : ( y ) )

#define MAXLINE  256
#define MAXITEM  50
#define EVER    ;;
#define UP      72
```

```
#define  DOWN     80
#define  WIDTH    80
#define  ROW      5

unsigned int read_key(int button);
int
        box_menu_start(MENU s[],int fore,int bk,int frame,int bar,int wdeep),
        box_select(MENU m[],int fore,int bk,int button,int limit,int market),
        box_menu_start(MENU s[],int fore,int bk,int frame,int bar,int wdeep),

        menu_choice(MENU m[],int fore,int back);

void active(char *m,int row,int col,int fore,int back),
        active_off(char *m,int row,int col,int fore,int bakc),
        cbox(int row,int col,int wide,int deep,int color,int bars),
        close_box(void),
        hide_cursor(void),
        highlight(MENU m[],int row,int fore,int back),
        indicator(int fore,int marker,int limit),
        menu_end(void),
        menu_off(MENU m[],int fore,int back),
        menu_on(MENU m[],int fore,int back),
        menu_start(MENU s[],int fore,int back),
        first_active(MENU s[],int fore,int back),
        prepare(void),
        redraw_window(MENU m[],int first,int fore,int limit),
        restore(MENU m[],int row,int fore,int back),
        save_screen(int row,int col,int wide,int deep,char *s),
        write_screen(int row,int col,int wide,int deep, char *s);
```

ภาคผนวก ข.

intel

PRELIMINARY

8031AH/8051AH
SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8031AH — Control-Oriented CPU with RAM and I/O
- 8051AH — An 8031AH with Factory Mask-Programmable ROM
- Fabricated with Intel's HMOS II Process

- 4K × 8 ROM (8051AH only)
- 128 × 8 RAM
- 32 I/O Lines (Four 8-Bit Ports)
- Two 16-Bit Timer/Counters
- Programmable Full-Duplex Serial Channel
- 128K Accessible External Memory
- Boolean Processor
- 4 μs Multiply and Divide
- 218 User Bit-Addressable Locations
- 100 mA Typical Supply Current

The 8031AH/8051AH is Intel's HMOS II version of the high performance 8-bit 8031/8051 microcomputer. While the 8031AH/8051AH features the same powerful architecture and instruction set as its HMOS I predecessor, it offers the additional benefit of lower power supply current.

The 8031AH/8051AH provides a cost-effective solution for those controller applications requiring up to 64K bytes of program and/or 64K bytes of data storage. Specifically, the 8031AH contains 128 bytes of read/write data memory; 32 I/O lines configured as four 8-bit parallel ports; two 16-bit timer/counters; a five-source, two-priority level, nested interrupt structure; a programmable serial I/O port; and an on-chip oscillator with clock circuitry. The 8051AH has all of these 8031AH features plus 4K bytes of nonvolatile read only program memory. Both microcomputers can use standard TTL compatible memories and most byte-oriented MCS®-80 and MCS-85 peripherals for additional I/O and memory capabilities.

The 8031AH/8051AH microcomputer, characteristic of the entire MCS-51 family, is efficient in both control and computational type applications. This results from extensive BCD/binary arithmetic and bit-handling facilities. The 8031AH/8051AH also makes efficient use of its program memory space with an instruction set consisting of 44% one-byte, 41% two-byte, and 15% three-byte instructions. At 12MHz CPU operation, over half of the instructions execute in just 1.0μs, while the longest instructions, multiply and divide, require only 4μs.

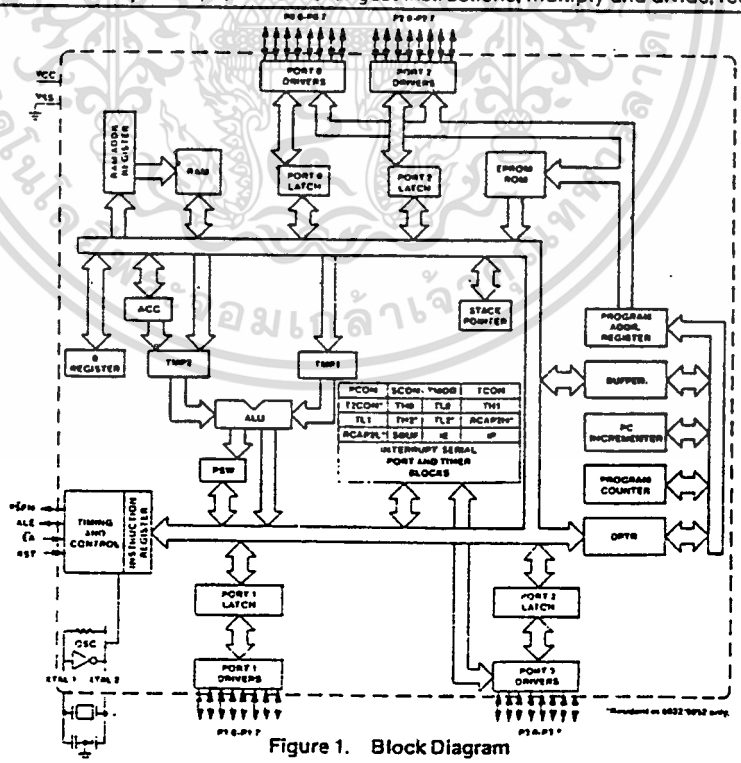


Figure 1. Block Diagram

8031AH/8051AH PIN DESCRIPTIONS

V_{SS}
Circuit ground potential.

V_{CC}
5V power supply input for normal operation and program verification.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

Port 1
Port 1 is an 8-bit quasi-bidirectional I/O port. It is also used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

Port 2
Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (Timer/counter 0 external input)
P3.5	T1 (Timer/counter 1 external input)
P3.6	WR (external Data Memory write strobe)
P3.7	RD (external Data Memory read strobe)

The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

RST/V_{PD}
A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (≈8.2kΩ) from RST/VPD to V_{SS} permits power-on reset when a capacitor (≈10 μf) is also connected from this pin to V_{CC}.

If RST/V_{PD} is held within the V_{PD} spec.while V_{CC} drops below its spec, RST/V_{PD} will provide standby power to the RAM. When RST/V_{PD} is low, the RAM's current is drawn from V_{CC}.

ALE
Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs.

$\overline{PSEN}$
The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods except during external data memory access. $\overline{PSEN}$ remains high during internal program execution.

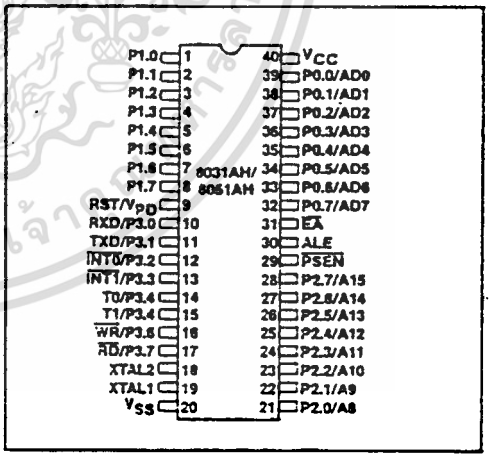


Figure 2. Pin Configuration

$\overline{EA}$

When held at a TTL high level, the 8051AH executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the 8031AH/8051AH fetches all instructions from external Program Memory. Do not float $\overline{EA}$ during normal operation.

XTAL1

Input to the inverting amplifier that forms part of the oscillator. This pin should be connected to ground when an external oscillator is used.

XTAL2

Output of the inverting amplifier that forms part of the oscillator, and input to the internal clock generator. XTAL2 receives the oscillator signal when an external oscillator is used.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ... 0°C to 70°C

Storage Temperature ..... −65°C to +150°C

Voltage on Any Pin With Respect to Ground (V_{SS}) ..... −0.5V to +7V

Power Dissipation ..... 1 Watt

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

DC CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 4.5V to 5.5V; V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	−0.5	0.8	V	
VIH	Input High Voltage (Except RST/V_{PD} and XTAL2)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to RST/V_{PD} For Reset, XTAL2	2.5	V_{CC} + 0.5	V	XTAL1 to V_{SS}
V_{PD}	Power Down Voltage to RST/V_{PD}	4.5	5.5	V	V_{CC} = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = −80μA
VOH1	Output High Voltage Port 0, ALE, $\overline{PSEN}$	2.4		V	IOH = −400μA
IIL	Logical 0 Input Current Ports 1, 2, 3		-800	μA	Vin = 0.45V
IIL2	Logical 0 Input Current for XTAL 2		-2.5	mA	XTAL1 = V_{SS}, Vin = 0.45V
ILI	Input Leakage Current To Port 0, $\overline{EA}$		±10	μA	0.45V < Vin < V_{CC}
IIH1	Input High Current to RST/V_{PD} For Reset		500	μA	Vin < V_{CC} − 1.5V
ICC	Power Supply Current		125	mA	All outputs disconnected
IPD	Power Down Current		10	mA	V_{CC} = 0V
CIO	Capacitance of I/O Buffer		10	pF	f_C = 1MHz, T_A = 25°C

See page 4 for Notes.

AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = 0V, C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100 pF; C_L for all other outputs = 80 pF)

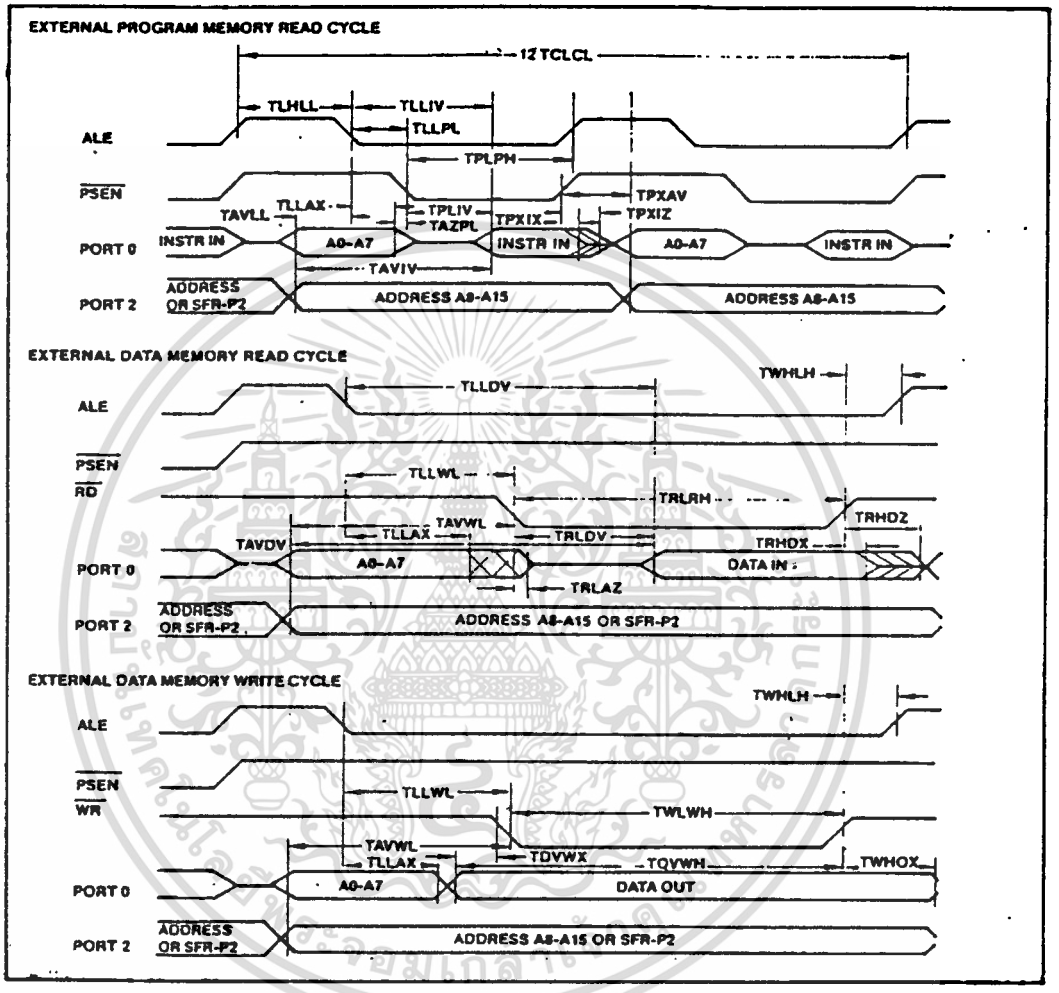
EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{PSEN}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{PSEN}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		63	ns		TCLCL-20	ns
TPXAV	Address Valid After $\overline{PSEN}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{PSEN}$	0		ns	0		ns

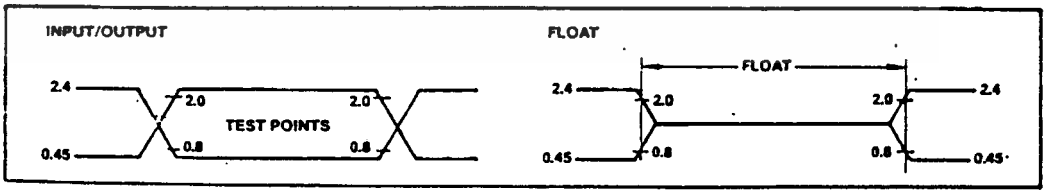
EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TRLRH	$\overline{RD}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		
TRLDV	$\overline{RD}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{RD}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{WR}$ or $\overline{RD}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{WR}$ or $\overline{RD}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{WR}$ or $\overline{RD}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{WR}$ Transition	23		ns	TCLCL-60		ns
TQVWH	Data Setup Before $\overline{WR}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{RD}$		0	ns		0	ns

AC TIMING DIAGRAMS



AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



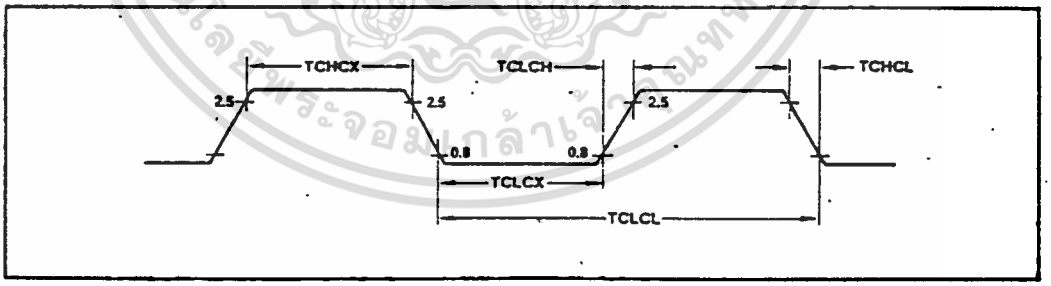
AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 2.4mA or sources 400 μA at the voltage test levels.

Note 1: VOL is degraded when the 8031AH/8051AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8031AH/8051AH as possible.

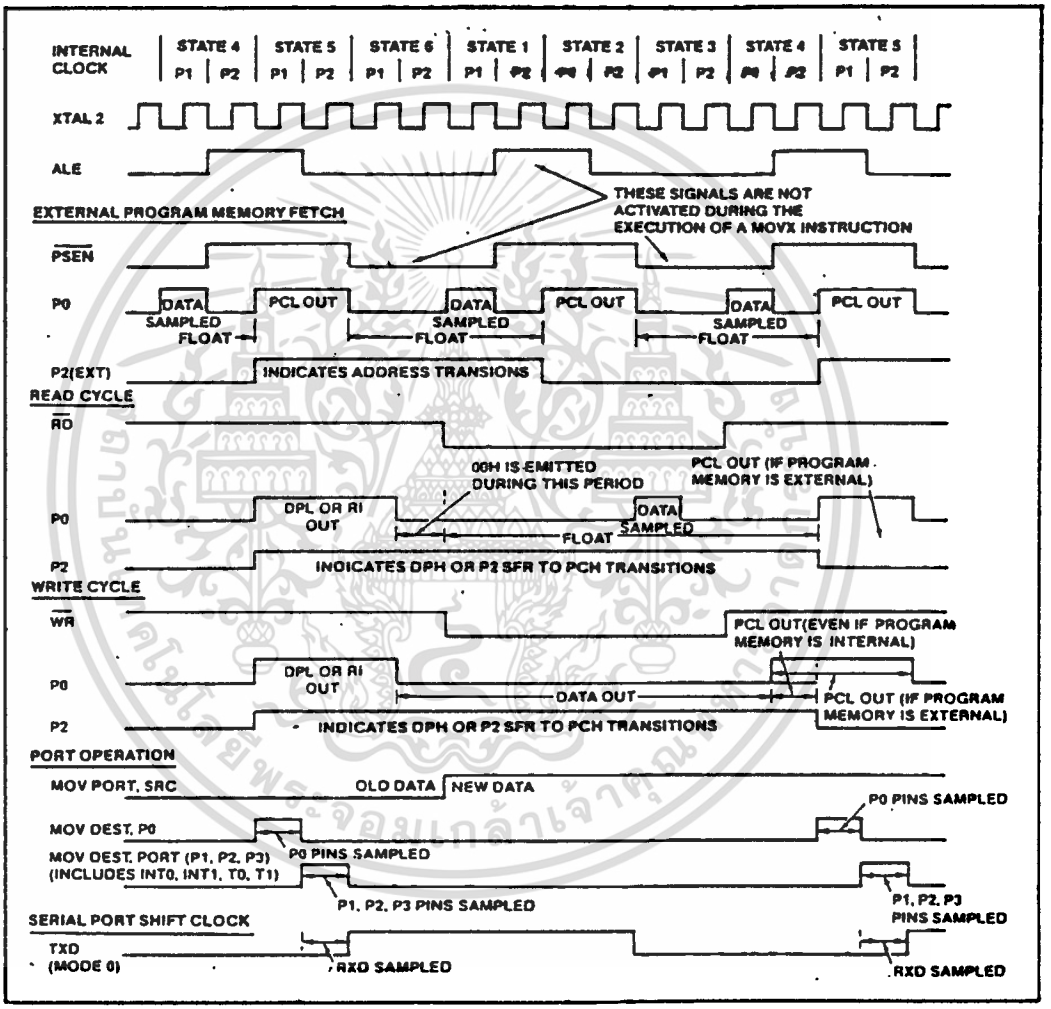
Datum	Emitting Ports	Degraded 1/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Variable Clock freq = 1.2 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	833.3	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25°C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS™-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,/bit	AND complement of direct bit to Carry	2	2
ORL	C,bit	OR direct bit to Carry flag	2	2
ORL	C,/bit	OR complement of direct bit to Carry	2	2
MOV	C,bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp. immed. to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp. immed. to ind. & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0-R7
direct —128 internal RAM locations, any I/O port, control or status register
@Ri —Indirect internal RAM location addressed by register R0 or R1
#data —8-bit constant included in instruction
#data16 —16-bit constant included as bytes 2 & 3 of instruction
bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
rel —SJMP and all conditional jumps include an 8-bit offset byte. Range is +127-128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	

Hex Code	Number of Bytes	Mnemonic	Operands
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr,A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr,data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0

Hex Code	Number of Bytes	Mnemonic	Operands
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0, data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A