อีมูเลเตอร์สำหรับ CPU-8085

8085 EMULATOR

โดย

นายปริญญา เอกโพธิ์

นายกิติชัย ธาตรีทอง

นายสมชาย แก้วลัดดากร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2534

ปริญญานิพนธ์ปีการศึกษา 2534

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง 8085 EMULATOR

ผู้จัดทำ

1. นายปริญญา	เอกโพธิ์	33.162213
2. นายกิติชัย	ธาตรีทอง	33.161201
3. นายสมชาย	แก้วลัดดากร	33.161226

อ. สิงห์ทอง พัฒเสรษฐานนท์ อาจารย์ที่ปรึกษา

(.....)

อีมูเลเตอร์สำหรับ CPU - 8085

ปริญญา เอกโพธิ์
กิติชัย ธาตรีทอง
สมชาย แก้วลัดดากร
อ.สิงห์ทอง พัฒนเศรษฐานนท์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2534

บทคัดย่อ

ปริญญานิพนธ์นี้กล่าวถึง การออกแบบและสร้างระบบเครื่องต้นแบบของ อีมูเลเตอร์ สำหรับ CPU เบอร์ 8085 โดยได้ประยุกต์เอา IBM PC/XT/AT เป็นเทอร์มินัลในการติดต่อกับผู้ใช้ เพื่อใช้ในการตรวจสอบและแก้ไขโปรแกรม (DEBUG) ที่สามารถทำได้ทั้งแบบ ซิงเกิล สะเต็ป (SINGLE STEP) ซึ่งสามารถกำหนดจุดหยุด (BREAK POINT) ของโปรแกรมได้ และในการโหมดการ RUN โปรแกรมตั้งแต่ต้นจนจบ

ความสามารถของตัวอีมูเลเตอร์นี้ สามารถที่จะนำไปเสียบเข้าโดยตรงกับซ็อกเก็ต (SOCKET) ของไมโครโปรเซสเซอร์ ที่ติดต่ออยู่กับระบบที่จะทำการตรวจสอบหาข้อผิดพลาด โดยให้ตัวอีมูเลเตอร์นี้ จะทำหน้าที่แทนไมโครโปรเซสเซอร์ ซึ่งจะมีประโยชน์อย่างมาก เพราะมันสามารถที่จะตรวจสอบหาข้อผิดพลาดทั้งทางด้านฮาร์ดแวร์ และซอฟแวร์ของระบบที่ได้ออกแบบไว้แล้ว

8085 EMULATOR

PRARINYA EKAPHO

KITICHAI THATREETONG

SOMCHAY KAEWLADDAKORN

SINGTONG PATTANASAGETHANON ADVISOR

1992

Abstract

This thesis deals with the design and construction of emulator for the 8085 microplrocessor. User interface can be implemented on IBM PC/XT/AT, to be terminal. It USED to testing and debug the program of the prototype. Which can work as a single step and can determine the break point in the practical run all.

The capability of emulator helping us is that it can plugged directly into the microprocessing socket of the prototype or production user's equipment extending into this hard ware the debugging system. It's very powerful, since it providcs the capability for hardware and software debugging in the final environment of the system.

สารบัญ

หน้า

หน้า

สารบัญรูป

หน้า

หน้า

บทที่ 1

บทนำ

ในการทดสอบและออกแบบวงจร โดยใช้ไมโครโปรเซสเซอร์เป็นตัวควบคุม เช่น ใช้ CPU -8085 ในการออกแบบวงจร ซึ่งในปัจจุบันต้องใช้ซิงเกิลบอร์ด (SINGLE BOARD) ในการเขียนโปรแกรมเพื่อการทดสอบ โดยในการเขียนโปรแกรมจะต้องป้อนคำสั่งเป็นเลขฐาน 16 ทำให้เกิดความไม่สะดวกในการตรวจสอบและหาจุดบกพร่องหรือความผิดพลาดต่าง ๆ จึงเกิดแนวคิดที่ว่า จะนำเอาการเขียนโปรแกรมและการทดสอบมาทำบนเครื่องไมโครคอมพิวเตอร์ โดยการต่ออุปกรณ์เพิ่มภายนอก แล้วนำสัญญาณไปใช้ในการควบคุมวงจร โดยหลักการนี้ทำให้ผู้ใช้สามารถเขียนโปรแกรมและตรวจสอบการทำงานได้สะดวก ในที่นี้จะใช้ CPU - 8085 เนื่องจากเป็นเบอร์ที่นิยมนำไปใช้ในการสร้างระบบควบคุมต่าง ๆ (Microcontroller) การเขียนและแก้ไขโปรแกรมจะทำบนเครื่องไมโครคอมพิวเตอร์ มีการต่อสัญญาณจากขาของ CPU -8085 ไปใช้ในการทดสอบกับระบบภายนอก

ปริญญานิพนธ์นี้จะกล่าวถึง หลักการในการสร้างการทำงานเลียนแบบ CPU เบอร์ 8085 (8085 EMULATOR) โดยสังเขป ซึ่งจะแยกเป็น 2 ส่วนคือ ส่วนของ Hardware ซึ่งจะอธิบายตาม Block diagram และวงจรใช้งานจริง อีกส่วนหนึ่งจะเป็นทางด้านของ Solfware ซึ่งเป็นระบบ Pop - up และ Pull down menu โดยได้จัดแบ่งเป็นโครงสร้างของระบบ Menu ต่าง ๆ และจะได้อธิบายแยกไว้ทีละหัวข้อ บทถัดไปนั้นจะกล่าวถึงลำดับขั้น การทำงานของโปรแกรมที่เขียนขึ้นมาเพื่อควบคุมเครื่อง 8085 อีมูเลเตอร์นี้ ส่วนบทสุดท้ายจะเป็นบทสรุปของปริญญานิพนธ์ และรวมไปถึงข้อจำกัดของปริญญานิพนธ์นี้ สำหรับในภาคผนวกได้รวบรวมการออกแบบลายวงจรพิมพ์ของเครื่องอีมูเลเตอร์นี้ ตลอดจนตัวโปรแกรมทั้งหมดที่ได้เขียนขึ้นมา

ผู้จัดทำหวังเป็นอย่างยิ่งว่า ปริญญานิพนธ์ฉบับนี้คงจะเป็นประโยชน์แก่ผู้ที่สนใจในการใช้งานไมโครโปรเซสเซอร์ไม่มากก็น้อย หากรายงานที่ทางกลุ่มได้จัดทำขึ้นนี้มีสิ่งใดสิ่งหนึ่งที่บกพร่อง คณะผู้จัดทำยินดีที่จะน้อมรับคำแนะนำเพื่อเป็นประโยชน์ในการจัดทำในครั้งต่อไป

1.1 วัตถุประสงค์

ในการทำปริญญานิพนธ์นี้มีวัตถุประสงค์ที่สำคัญอยู่ 4 ประการด้วยกันคือ

1. สร้าง เครื่อง EMULATOR 8085 ที่สามารถติดต่อกับระบบบัส ของ IBM/PC/XT/AT ได้
2. ใช้ IBM/PC ในการตรวจซ่อมเครื่องมือ เครื่องจักรอัตโนมัติที่มี 8085 เป็น CPU และจำลองการทำงานโดยใช้ 8085 EMULATOR เป็นตัวกลางในการติดต่อได้
3. สามารถนำเครื่อง 8085 EMULATOR ไปใช้ในการทดสอบการทำงานของโปรแกรมในระบบที่ต้องการจะทดสอบ ในแบบ Single step ได้
4. เพื่อศึกษาถึงเทคนิคต่างๆ ในการเขียนโปรแกรมด้วยภาษาในระดับสูง (High Level) ในที่นี้คือ ภาษา C นำไปควบคุมและตรวจสอบ เครื่องมือ หรือเครื่องจักรอัตโนมัติที่มี CPU เบอร์ 8085 เป็นตัวควบคุมได้

1.2 ความเป็นมา

สำหรับเหตุผลหรือความเป็นมาที่เลือกทำโครงงานชิ้นนี้ ก็เพราะว่า ในการที่จะทำการตรวจสอบ ระบบใดระบบหนึ่งที่มี CPU เป็นตัวควบคุมอยู่นั้น ถ้าหากระบบนั้น ๆ เกิดทำงานผิดพลาดขึ้นมา ซึ่งสาเหตุนั้นอาจจะเกิดจากส่วนที่เป็น Hard ware หรือ Soft ware ก็ได้ ถ้าเกิดจาก Hard ware ท่านอาจจะวิเคราะห์หาสาเหตุจากวงจรได้ แต่ถ้าข้อผิดพลาดนั้นเกิดจาก Soft ware ท่านไม่อาจที่จะตรวจสอบได้เลย และนี่ก็คือเหตุผลที่เลือกทำโครงงาน 8085 EMULATOR นี้ ซึ่งสามารถที่จะนำมาช่วยแก้ปัญหาดังกล่าวได้ ส่วนสาเหตุที่เลือก CPU เบอร์ 8085 ก็เพราะว่า ระบบที่ใช้ CPU ตัวนี้ควบคุมได้ถูกใช้มานานแล้ว ดังนั้นระบบดังกล่าวจึงเริ่มที่จะมีปัญหา ซึ่งถ้าหากใช้เครื่องอีมูเลเตอร์ 8085 นี้มาใช้เพื่อช่วยในการตรวจซ่อม ก็จะสามารถที่จะลดเวลาในการตรวจซ่อม ทั้งทางด้าน Hard ware และ Soft ware ลงไปได้เป็นอย่างมาก

บทที่ 2

โครงสร้างทาง Hardware ของ 8085 EMULATOR

บทนำ

ในส่วนการออกแบบวงจรทางด้าน Hardware ของ เครื่อง อีมูเลเตอร์นี้ จะเป็นวงจร Digital ที่ทำหน้าที่เสียนแบบหรือจำลองการทำงานในการรับ - ส่งสัญญาน ต่าง ๆ ที่จำเป็น ที่เหมือนกับสัญญานที่ส่งออกมาจากขาต่าง ๆ จากตัว CPU เบอร์ 8085 ของบริษัท Intel โดยที่วงจรภายในเครื่องนี้ ส่วนหลักจะเป็นลักษณะของ port input และ port output โดยโครงสร้างหลัก ๆ ของวงจรจะแสดงได้ดัง Block Diagram ดังรูปที่ 2.1

2.1 รายละเอียดเกี่ยวกับตัว CPU เบอร์ 8085

8085 ถูกผลิตขึ้นโดยบริษัท Intel ซึ่งถูกพัฒนามาจาก 8080 โดยโครงสร้างภายใน 8085 นั้นจะประกอบไปด้วย ตัวกำเนิดสัญญานนาฬิกา, ระบบควบคุม, interrupt vactor 5 input และมีโปรแกรมที่เกี่ยวกับการรับส่งข้อมูลอนุกรม ชุดคำสั่งของ 8085 จะเหมือนกับ 8080 ทั้งหมด แต่ได้เพิ่มคำสั่งใหม่อีก 2 คำสั่งเข้าไป การเรียกใช้ Register และการอ้าง Addres จะยังคงเหมือนกับ 8080 และอุปกรณ์สนับสนุน (Chips support) ที่ผลิตขึ้นโดย บริษัท Intel สามารถนำมา Interface กับ 8085 ได้ง่าย

ความหมายและลักษณะการจัดวางขาของ CPU - 8085

ตำแหน่งขา	ชื่อขาสัญญาน	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	READY	Wait state indicator	Input	
5	SID	Serial input data	Input	
4	SOD	Serial output data	Output	
39	HOLD	Hold request	Input	
38	HLDA	Hold acknowledge	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 7.5	Vectored interrupt	Input	
11	$\overline{INTA}$	Interrupt acknowledge	Output	
36	$\overline{RESETIN}$	System reset	Input	
3	$\overline{RESETOUT}$	Peripheral reset	Output	
1,2	X1,X2	Crystal contacts	Input	
37	CLK	System clock	Output	
40	Vcc	Power	Input	
20	Vss	Ground	Input	

รายละเอียดเกี่ยวกับขาสัญญานทาง output ต่าง ๆ มีดังนี้

ALE (Address Latch Enable)

ขาสัญญานนี้จะถูกส่งออกเป็น high เมื่อ CPU ได้ทำการส่ง Address ออกมาที่ขา AD0-AD7 แล้ว สัญญาน ALE จะ ถูกใช้ให้เป็น ตัว Latch สัญญาน AD0-AD7 ได้ โดยที่อุปกรณ์ ภายนอกจะทำการ Latch Address เมื่อถึงขอบขาลงของสัญญาน ALE

A7 - A15 (Address bus)

คือ Address ขนาด 8 bit ที่ต้องส่งออกไปเพื่อใช้ใน การอ้าง Address ซึ่งจะไปรวมกับ AD0-AD7 เป็น Address ขนาด 16 bit ส่วน Address ที่ใช้ติดต่อกับ อุปกรณ์อินพุท-เอาท์ พุทนั้น จะถูกจำกัดให้ใช้งานได้เพียง 00-FFh ซึ่งก็จะใช้เพียงแค่ Address AD0-AD7 เท่านั้น

S0,S1

จะมีสถานะ 0 หรือ 1 ซึ่งเป็นขา output ที่ CPU ส่งออกมาเพื่อใช้แสดงว่า คำสั่งที่เกิดขึ้นในระบบเป็นคำสั่งประเภท ไหน ดังแสดงไว้ในตารางข้างล่างนี้

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

$\overline{RD}$ (Read)

ขา $\overline{RD}$ นี้จะเป็นสัญญาน output ที่ Active "low" ซึ่งมันจะ Active ขณะที่ทำการอ่านข้อมูล ในหน่วยความจำหรือ จากการใช้คำสั่ง input และยังสามารถใช้เป็นสัญญาน Read strobe memory และ อุปกรณ์ I/O ได้อีกด้วย

$\overline{WR}$ (Write)

ขา $\overline{WR}$ จะ Active "low"เมื่อ CPU ทำคำสั่งเกี่ยว กับการเขียนหรือส่ง output ออกไป บนบัสข้อมูล

IO/$\overline{M}$ (Input-Output/Memory)

สัญญานที่ส่งออกจากขานี้ จะเป็นตัวบอกว่า คำสั่งที่กำลัง ทำงานอยู่นั้น เป็นคำสั่งที่ติดต่ออยู่กับ Memory หรือ อุปกรณ์ I/O เมื่อสัญญาน IO/$\overline{M}$ เป็น "high"ข้อมูลบน Address bus จะเป็น ค่าของตำแหน่งที่ติดต่อกับอุปกรณ์ I/O ซึ่งจะทำงานโดยตรงใน การอ่านหรือเขียน port และเมื่อสัญญานนี้เป็น "low" ข้อมูลบน Address bus จะแสดงถึงตำแหน่งของหน่วยความจำ ที่ CPU ต้องการติดต่อด้วย ซึ่งจะเกิดการอ่านหรือ เขียนโดยตรงกับ RAM หรือ ROM สัญญาน IO/$\overline{M}$ นี้จะใช้ร่วมกับ สัญญานควบคุมการอ่าน หรือเขียน

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

สัญญาน $\overline{RESET\ OUT}$ นี้จะ Active "low" เมื่อ CPU อยู่ในสภาวะที่ถูก Reset ซึ่งสามารถนำสัญญานนี้ไปใช้ในการ Reset อุปกรณ์สนับสนุน อื่น ๆ (Chips support)ที่ต่ออยู่ภายใน ระบบได้

$\overline{INTA}$ (Interrupt Acknowledge)

สัญญาน $\overline{INTA}$ จะ Active เป็น "low" ก็ต่อเมื่อ CPU ตอบรับการขอ interrupt

HLDA (Hold Acknowledge)

เป็นสัญญานตอบรับการขอใช้บัส ซึ่งจะ Active "high" และจะทำให้ระบบลอยตัว ออกจากระบบบัส

SOD (Serial Output Data)

คือข้อมูลแบบอนุกรมที่ส่งออกไป ซึ่งถูกควบคุมจากโปรแกรมรีดยค่าที่อยู่ในบิท ที่ 7 ของ Accumulator จะถูกส่งออก ไปยังขา SOD เมื่อมีการ set interrupt masks คำสั่ง SIM จะถูกทำงาน ผู้ใช้โปรแกรมจำเป็น จะต้องกำหนดให้อยู่ในตำ แหน่งที่ 7 และลองทดสอบการทำงานของคำสั่ง SIM

รายละเอียดเกี่ยวกับขาสัญญานทาง input ต่าง ๆ มีดังนี้

TRAP

สัญญานที่เข้ามาทางขา TRAP จะเป็นสัญญาน nonmaskable interrupt จะทำงานที่ Active "high" ซึ่ง จะเป็นการ interrupt ที่มีลำดับความสำคัญสูงสุดและจะทำงานที่ ขอบขาขึ้น ในขณะที่สัญญานนี้ Active CPU จะทำงานตามคำสั่ง ปัจจุบันจนเสร็จ แล้วจะเก็บค่าตำแหน่งของ Address ถัดไปลงใน Stack และจะกระโดดไปยังตำแหน่ง 0024h ซึ่งผู้ออกแบบ จะ ต้องเขียนโปรแกรมบริการการ interrupt เริ่มต้นไว้ที่ตำแหน่งนี้

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

เป็นสัญญานการ interrupt แบบ maskable จะทำ งานเมื่อเป็นลอจิก "high"เมื่อมีสัญญานการขอ interrupt เข้า มาที่ขา RST X.5 CPU จะทำงานตามคำสั่งที่กำลังทำอยู่ ให้เสร็จ เสียก่อน จากนั้นจึงจะทดสอบสถานะของ interrupt enable flip-flop ถ้า interrupt flip-flop ตัวนี้อยู่ในสภาวะ disable อยู่ CPU จะไม่สนใจต่อการขอ interrupt นั้น ๆ ซึ่ง มันจะทำงานตามคำสั่งถัดไป แต่ถ้า interrupt flip-flop enable อยู่ CPU จะเก็บค่า Program counter ลงบน Stack และจะกระโดดไป ตามตำแหน่ง Address ที่กำหนดไว้ดังต่อไปนี้

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

สัญญาณที่ขา RST7.5 จะมีความสำคัญสูงสุด และ 5.5 จะมีความสำคัญต่ำสุด การ interrupt แบบนี้อาจทำได้จากโปรแกรมโดยใช้คำสั่ง SIM (set interrupt mask) สถานะใน การ interrupt จะถูกเก็บไว้ใน interrupt register จะ ถูกอ่านด้วยคำสั่ง RIM ดังนี้

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

บิทที่มี I นำหน้าจะเก็บสถานะการขอ interrupt ที่ เข้ามาแต่ละ inpu t ส่วนบิทที่มี M นำหน้าจะเป็นการ set การ ขอ interrupt จากโปรแกรมโดยใช้คำสั่ง SIM ซึ่งจะทำการ load ข้อมูลที่อยู่ใน Accumulator ลงใน register Bit ที่ 3 นั้นจะใช้เป็น Bit ที่ควบคุมการ disable หรือ enable ด้วยคำสั่ง DI และ EI ตามลำดับ เมื่อมีการขอ interrupt ค่า ที่บรรจุอยู่ใน interrupt register จะเป็นดังนี้คือ

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

SOE คือ การควบคุมการenable ข้อมูลที่จะถูกส่งออกมาที่ขา SID

X คือ bit ที่ไม่ได้ใช้งาน และ

MSE คือ เงื่อนไขของ master interrupt enable.

INTR (Interrupt Request)

เป็นสัญญาณ interrupt แบบ maskable ซึ่งทำงานที่ Active "high" เมื่ออุปกรณ์ภายนอกส่งสัญญาณมาที่ขานี้ CPU จะทำคำสั่งปัจจุบัน จนเสร็จก่อน หลังจากนั้น จะทำการตรวจ สอบสถานะของ I flip-flop ถ้า I flip-flop นี้ถูก Reset อยู่ สัญญาณ interrupt ก็จะไม่ได้รับการตอบสนองและ CPU จะ ทำคำสั่งถัดไป แต่ถ้า I flip-flop ถูก set อยู่ CPU ก็จะ ตอบสนองต่อการขอ interrupt โดยที่ CPU จะคิดว่า คำสั่งถัด ไปเป็น op-code จากภายนอก คำสั่งแรกจะถูกทำงาน หลังจาก การ interrupt ได้รับการตอบสนอง CPU จะทำการเก็บค่า Program counter ลงใน Stack จากนั้นจะทำงานตาม service routine และเมื่อพบคำสั่ง

RST ก็จะกลับมาที่ตำแหน่งหลังสุดที่เก็บไว้ใน Stack ก่อนที่จะตอบสนองการ interrupt และจะทำคำสั่งถัดไป Flag I สามารถควบคุมได้จากโปรแกรม ด้วยการ enable interrupt,EI และ disable, DI สัญญาน interrupt ที่เข้าที่ขานี้ จะมีความสำคัญต่ำกว่า สัญญาน interrupt อื่น ๆ ทั้งหมด

READY (Ready)

เมื่อสัญญาน READY นี้ Active เป็น "low" ขึ้นมา CPU จะสร้างสัญญาน Wait states จนกระทั่ง READY กลับเป็น "high" ซึ่งจะใช้ในกรณีที่อุปกรณ์อื่น ทำงานช้ากว่าระบบ

HOLD (Hold)

เมื่อขาสัญญานนี้เป็น "high" จะทำให้ CPU หยุดการทำงาน เมื่อทำคำสั่งปัจจุบัน จนครบ machine cycle แล้ว และมันจะลอยตัวออกจากระบบ เพื่อให้อุปกรณ์ภายนอกเข้ามาใช้ Address และ data bus ได้ หรือ อาจจะนำสัญญานนี้มาใช้ใน การทำงานกระบวนการ DMA หรือการ refresh dymamic ram ได้

RESET IN (Reset in)

เมื่อสัญญาน RESET IN นี้จะรับ input ที่เป็น "low" เพื่อใช้ในการ Reset การทำงานของระบบ และ Program counter จะถูก load เป็นตำแหน่งที่ 0000h ดังนั้นผู้ใช้งาน ควรเขียนโปรแกรม ให้เริ่มทำงานที่ Address 0000h ขณะ เดียวกัน INTA และ HLDA flip-flops จะถูก Clear ด้วย

SID (Serial Input Data)

คือ ขาสัญญาน input แบบอนุกรม ซึ่งถูกควบคุมด้วย โปรแกรม เมื่อมีการอ่านสัญญานการ interrupt คำสั่ง RIM จะ ถูกทำงาน จะมีข้อมูลเพียงบิทเดียวที่ขา SID และจะถูกส่งไปยัง bit ที่ 7 ของ Accumulator มันสามารถที่จะถูกส่งไปยังตำแหน่ง หน่วยความจำที่ผู้ใช้เป็นผู้กำหนด

Vcc

ขา input นี้ 8085 Emulator จะใช้เป็นตัว check ว่า ระบบภายนอกจ่ายไฟ (+5V) มาที่ขานี้หรือไม่ ถ้ามี โปรแกรม EMU8085 จะทำงาน แต่ถ้าไม่มีก็ไม่สามารถเรียกโปรแกรม EMU8085 มาทำงานได้หรืออาจจะไม่สามารถ Run โปรแกรมของ CPU - 8085 ได้

Vss

เป็นขาinput ที่ต้องต่อกับ Ground ของระบบภายนอกกับ IBM/PC

2.2 ส่วนประกอบทาง HARD WARE ของ 8085 EMULATOR

หลักการทำงานที่สำคัญที่สุดของ HARD WARE ก็คือจะต้องให้ OUTPUT ที่มีการแสดงสถานะทาง LOGIC ได้เหมือนกับ CPU 8085 ตัวจริงในแต่ละคำสั่ง ดังนั้นในการออกแบบทาง HARD WARE นี้ จึงต้องศึกษาถึงลักษณะของสัญญาณต่าง ๆ ของ CPU เบอร์ 8085 เสียก่อน ซึ่งได้กล่าวถึงในหัวข้อ 2.1 ไปแล้ว และพอจะจำแนกคุณลักษณะของขาสัญญาณได้ 3 ลักษณะดังนี้ คือ

1. สัญญาณที่มีลักษณะที่มีการส่งและรับ (2 ทิศทาง)

   เช่น DATA BUS

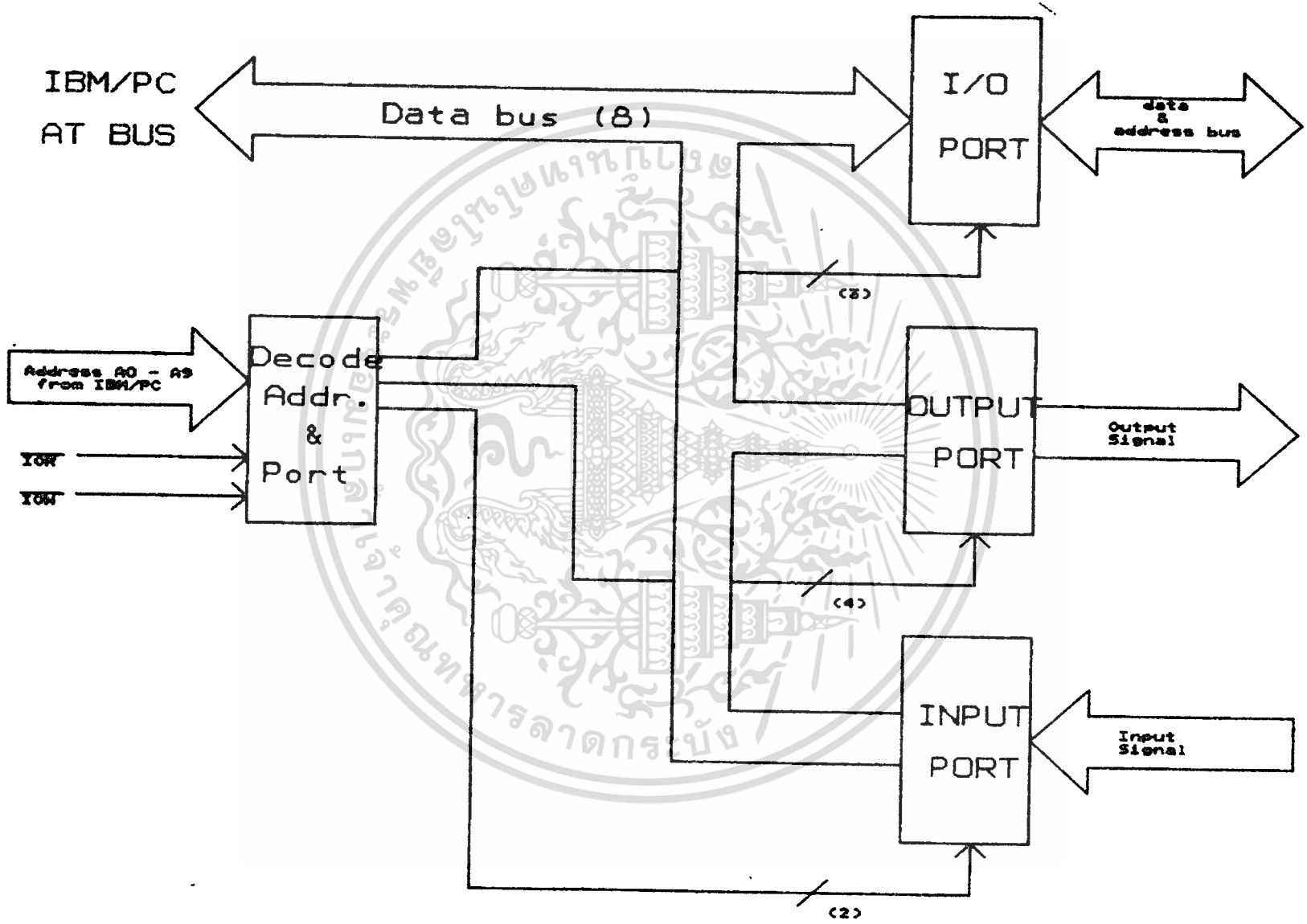
2. สัญญาณที่มีลักษณะเป็นการส่งออกอย่างเดียว (1 ทิศทาง)

   เช่น ADDRESS BUS, ALE, S0, S1, $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$, $\overline{RESET\ OUT}$, $\overline{INTA}$, HLDA และ SOD

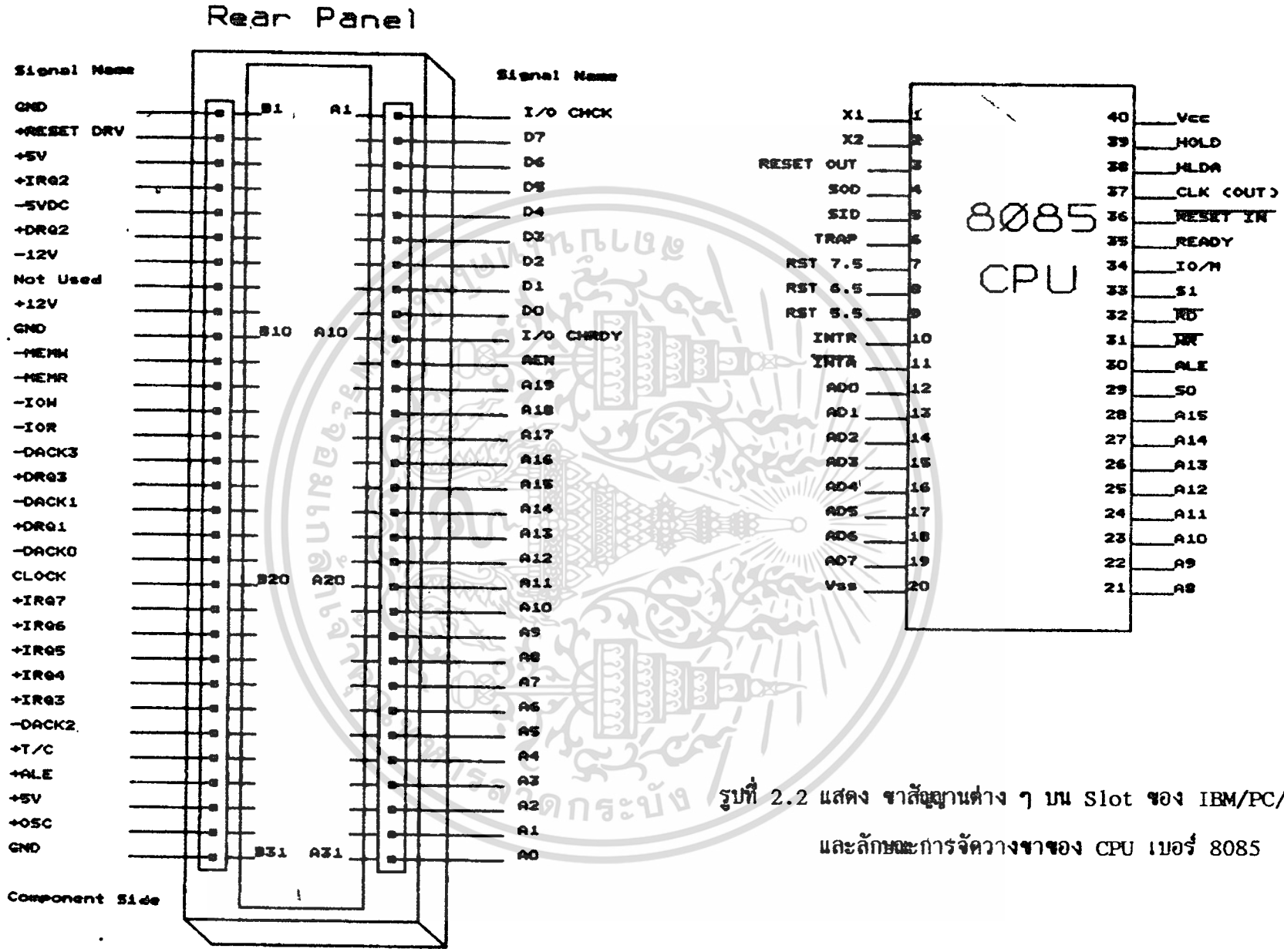
3. สัญญาณที่มีลักษณะเป็นการรับอย่างเดียว (1 ทิศทาง)

   เช่น TRAP, RST X.5, INTR, READY, HOLD, $\overline{RESET\ IN}$, SID, VCC และ Vss

จากที่กล่าวมานี้เป็นลักษณะต่าง ๆ ของประเภทของสัญญาณที่เกิดขึ้นกับขาของ CPU 8085 ตัวจริงและขาต่าง ๆนี้ จะต้องถูกควบคุมให้มีการทำงานเป็นไปตามคำสั่งต่าง ๆ ตามชุดคำสั่งของ CPU 8085 ในที่นี้จะใช้ IBM/PC เป็นตัวควบคุมการทำงานของระบบ เพื่อที่จะทำให้เข้าใจได้ดียิ่งขึ้น ขอให้พิจารณาจากรูปที่ 2.1 ซึ่งแสดงถึง BLOCK

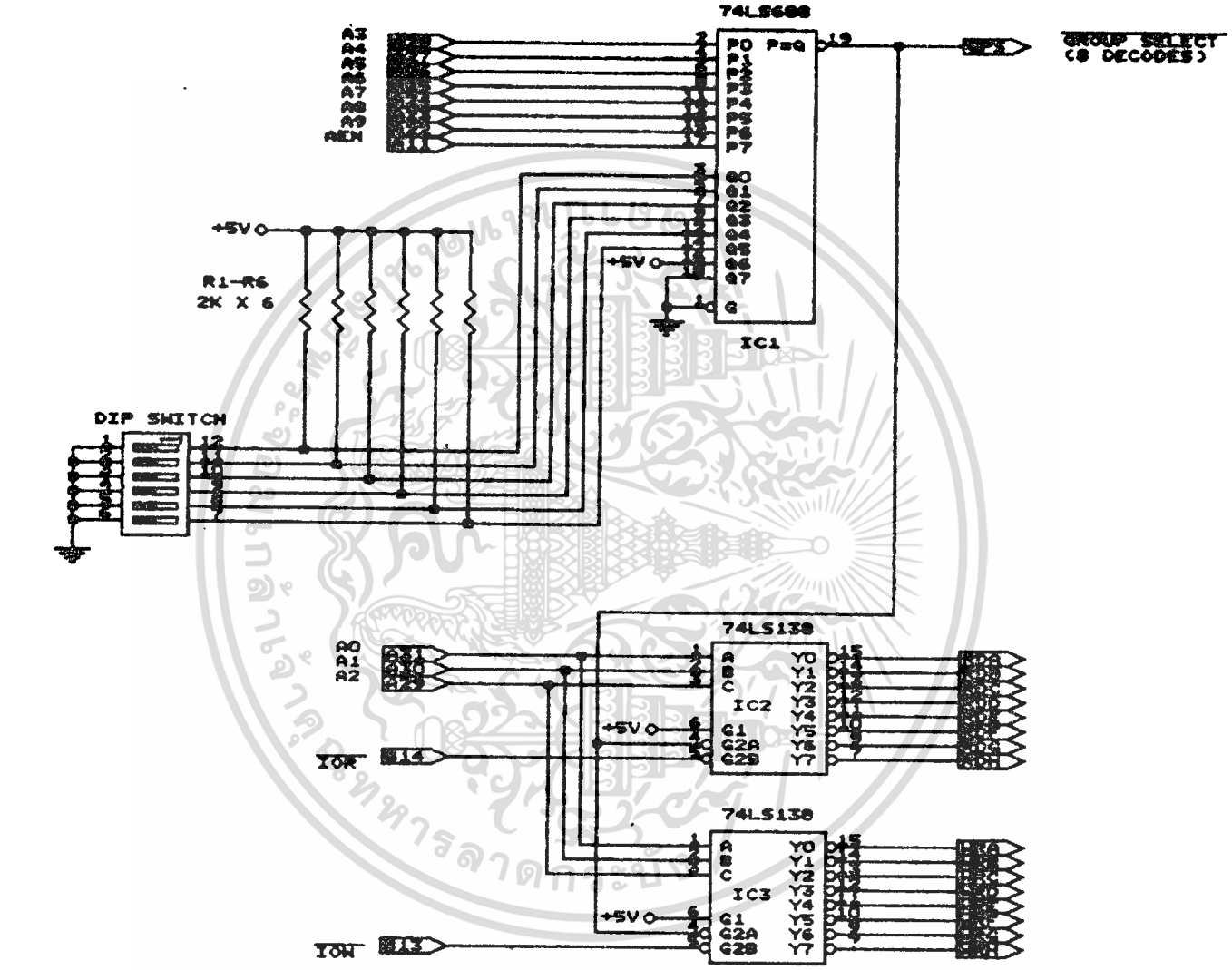


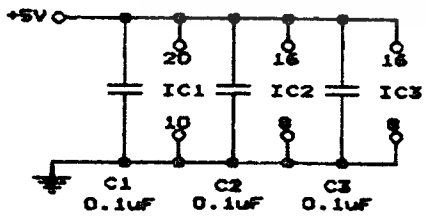
รูปที่ 2.1 แสดง Block Diagram ของเครื่อง 8085 EMULATOR



รูปที่ 2.2 แสดง ขาสัญญาณต่าง ๆ บน Slot ของ IBM/PC/XT/AT และลักษณะการจัดวางขาของ CPU เบอร์ 8085

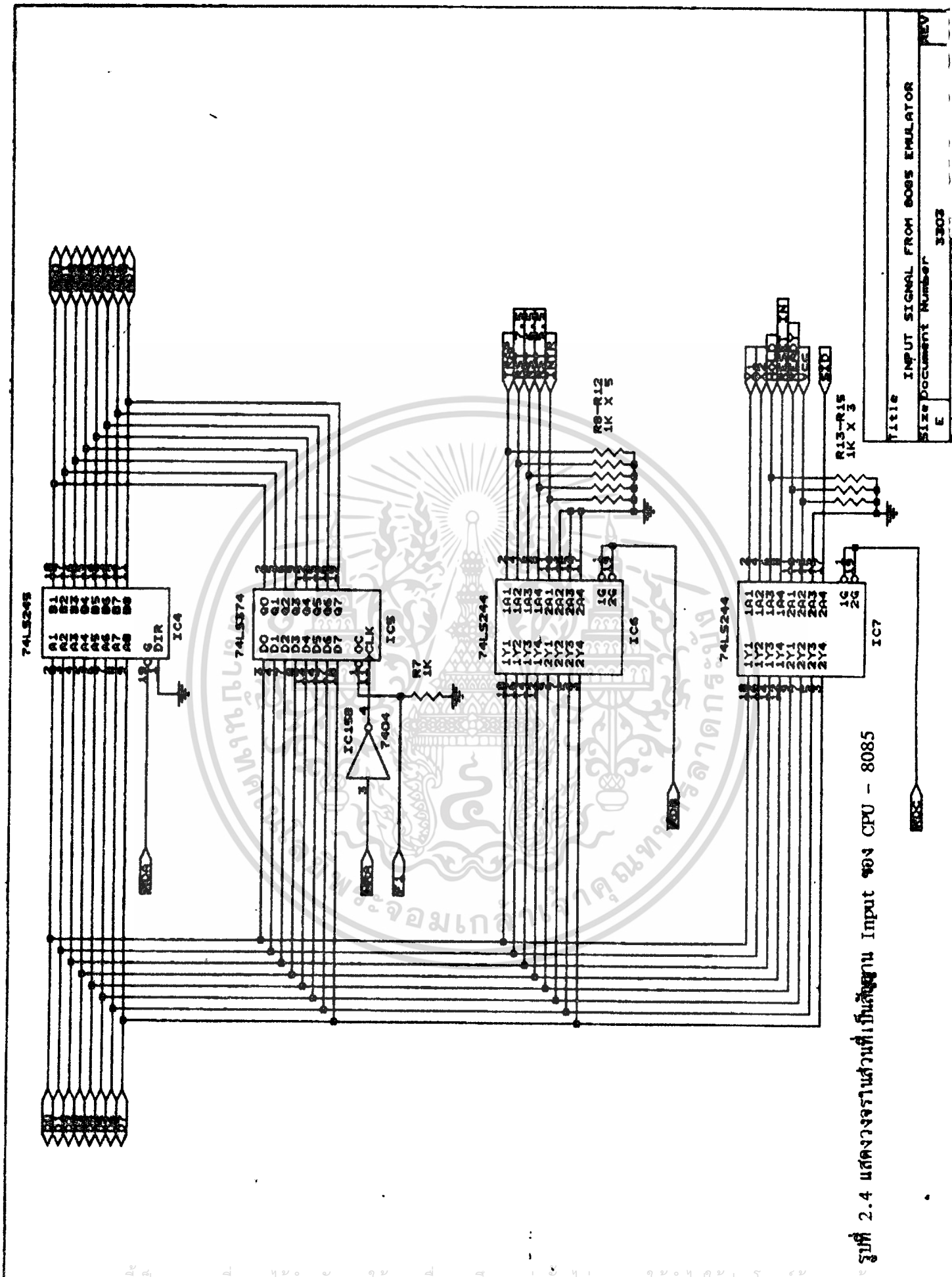
Title	SLOT IBM PC AND 8085 CPU	
Size E	Document Number 3301	REV
Date:	October 22, 1991	



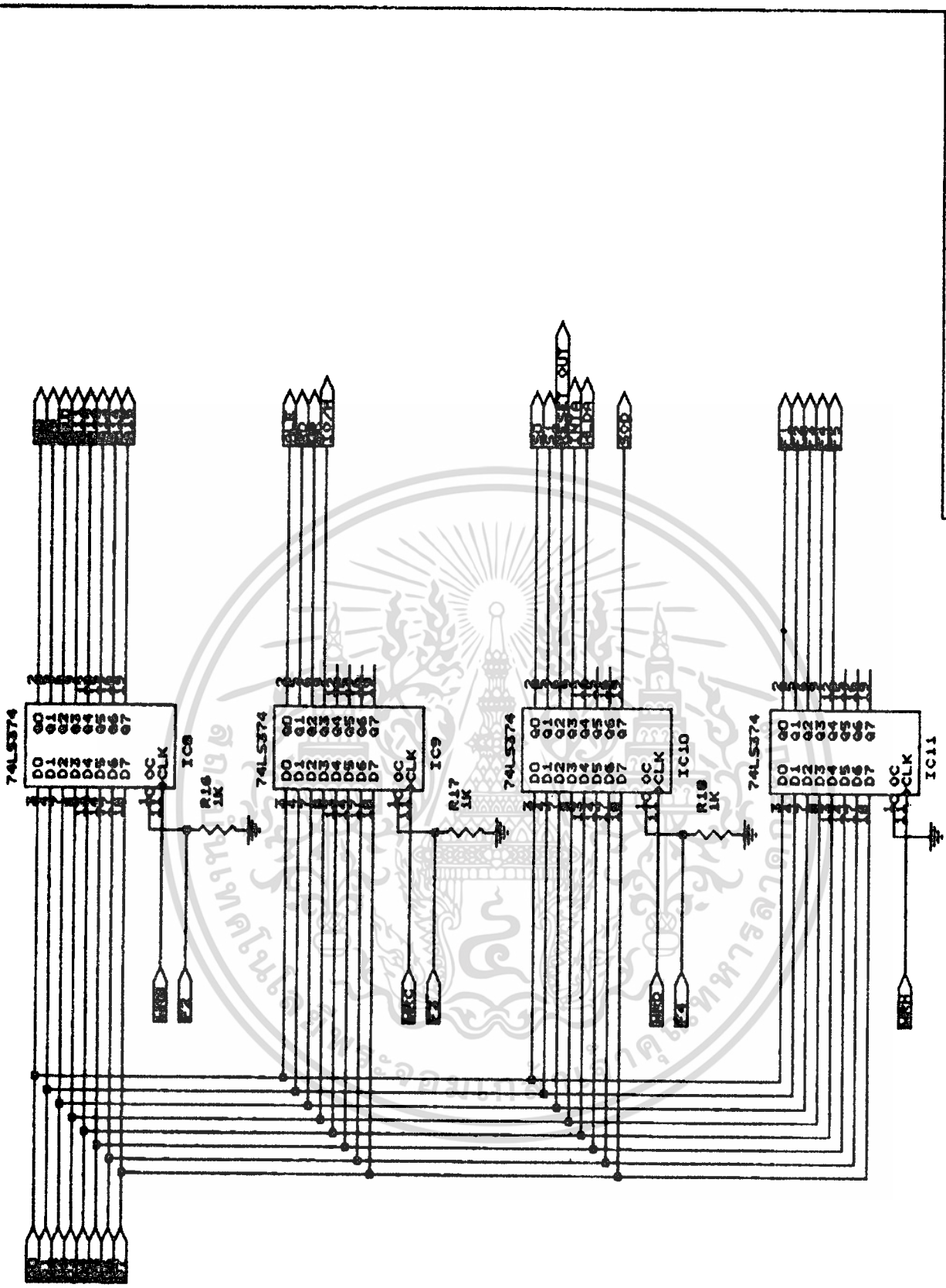


รูปที่ 2.3 แสดงวงจร Decoder number port ที่ใช้ติดต่อกับ 8085 EMULATOR

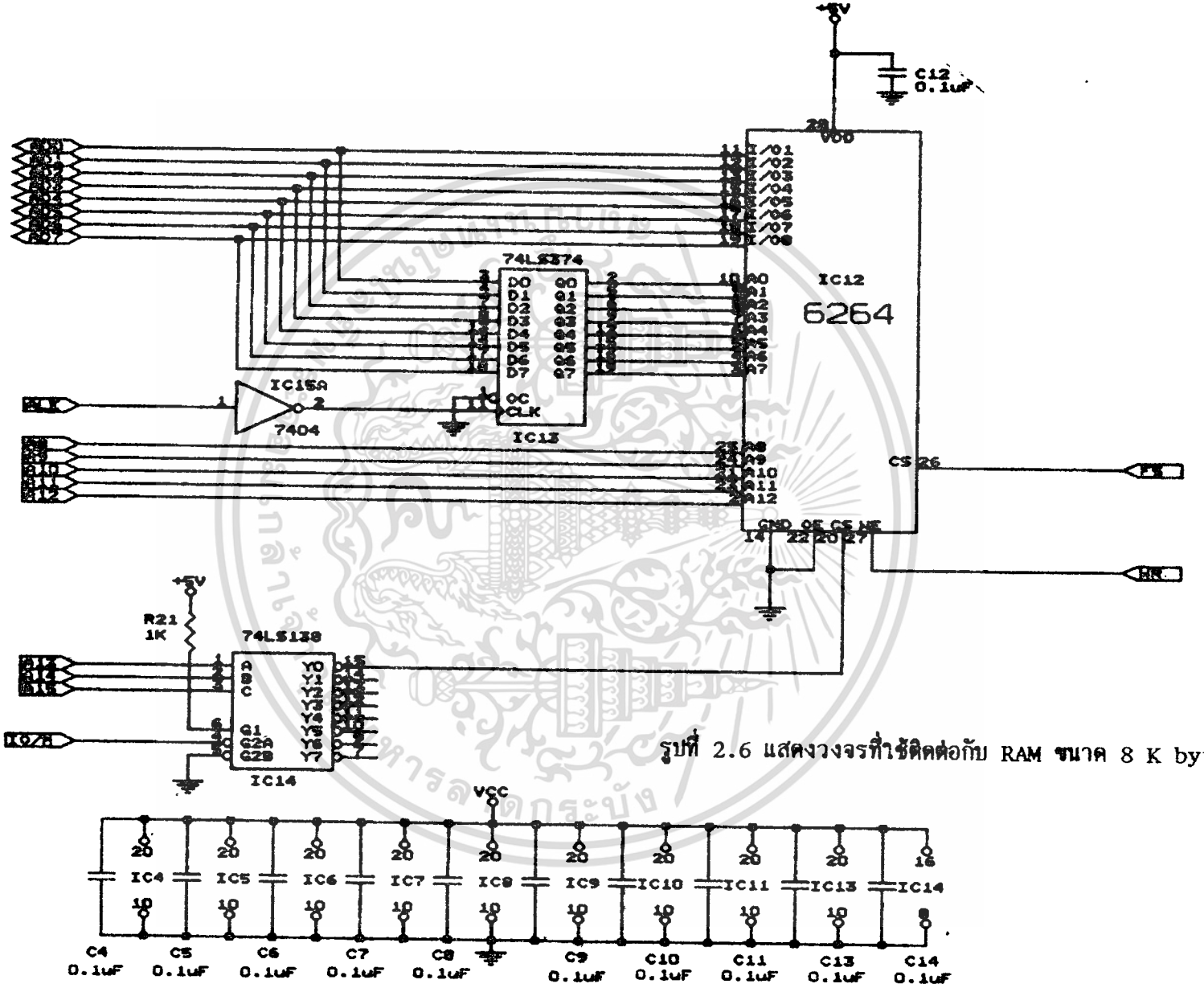
TITLE	DECODER WITH DIP SWITCH	
Size E	Document Number 3302	REV



รูปที่ 2.4 แสดงวงจรในส่วนที่เกี่ยวข้องกับ Input ของ CPU - 8085



รูปที่ 2.5 แสดงวงจรในส่วนที่เป็นสัญญาณ Output จาก CPU - 8085



รูปที่ 2.6 แสดงวงจรที่ใช้ติดต่อกับ RAM ขนาด 8 K bytes

DIAGRAM ของ EMULATOR 8085 และจาก BLOCK DIAGRAM นี้ ก็ได้นำไปสร้างเป็นวงจรที่สามารถ ใช้งานได้จริงดังรูปที่ 2.3 - 2.6

วงจรในการ Decoder address and number port

หน้าที่ของวงจรในส่วนนี้ก็คือ นำค่า address จากระบบ AT BUS มาทำการ Decode เบอร์ port ที่จะใช้ในการติดต่อกับ 8085 Emulator หลังจากนั้นก็จะส่งสัญญาน Enable ต่าง ๆ เพื่อเลือก chip ที่ต้องการจะติดต่อด้วย ซึ่งจะเป็น input หรือ output ก็ขึ้นอยู่กับว่า IBM/PC ทำคำสั่ง Read หรือ Write ในขณะนั้น โดยจะส่งสัญญานการ Active มาที่ขา $\overline{IOR}$ หรือ $\overline{IOW}$ ไปยังชุด Decoder ให้ทำงาน สำหรับรายละเอียดอยู่ในวงจรในรูปที่ 2.3 ซึ่งมีการทำงานโดยสังเขป ดังนี้

จากวงจร จะเห็นว่าเป็นวิธีการ Decode โดยใช้สวิทช์เลือก (DIP Switch) ซึ่งต่างจากการ Decode ในแบบ Fixed คือใน แบบ Fixed นั้นมีข้อเสียคือ address ที่เราเลือกใช้งานไว้นั้นอาจจะซ้ำกับ address ของ Card อื่นๆ ที่เรานำมาเสียบเพิ่มเข้าไปในระบบ ในภายหลังได้ ในกรณีเช่นนี้ เราต้องแก้ไขวงจรเพื่อหลีกเลี่ยงไปใช้ Address อื่น ที่ยังว่างอยู่ และไม่ถูกใช้งาน โดย Card ที่จะเพิ่มเข้าไปใหม่ ซึ่งยุ่งยาก และ ต้องเสียเวลามากขึ้น ปัญหาเช่นนี้เราสามารถแก้ไขได้ โดยใช้วงจร Decode ที่สามารถเปลี่ยนแปลงค่า Address ได้ โดยเพียงแต่เปลี่ยน ตำแหน่งของสวิทช์ (ในที่นี้คือ DIP Switch) ที่ set ไว้ในวงจรเท่านั้น

จากรูปวงจรจะทำการ Decode กลุ่ม Address ขนาด 8 Address ซึ่ง การเลือกกลุ่ม Address ที่จะทำการ Decode นี้ จะทำได้ โดยการ Set DIP Switch ที่ขา Q_0 - Q_5 ของ 74LS688 สำหรับหน้าที่ของ 74LS688 นี้ จะทำการเปรียบเทียบ ค่าของอินพุท 2 ชุด ที่ถูกส่งเข้ามาทางขา P_0 - P_7 และขา Q_0 - Q_7 ถ้าอินพุททั้ง 2 ชุดนี้เท่ากันแล้ว เอาท์พุทที่ขา $\overline{P = Q}$ (19) จะให้เอาท์พุทเป็นลอจิก "0" จากในวงจรขา P_0 - P_6 ของ 74LS688 จะต่อกับขา Address ที่บิต A_3 - A_9 ในขณะที่ขา Q_0 - Q_5 ต่อกับ ความต้านทาน ที่ทำหน้าที่เป็น Pull up (รักษาระดับแรงดันให้เป็นลอจิก "1" ไว้ ในกรณีที่ไม่มี อินพุทใด ๆ เข้ามา) และขา Q_0 - Q_5 นี้จะต่อกับปลายข้างหนึ่งของ DIP Switch ด้วย ส่วนปลายอีกข้างหนึ่งของ DIP Switch นั้นจะต่อลง Ground (ลอจิก "0") ไว้ ดังนั้นถ้าเราทำการ "ON" DIP Switch ที่ต่อกับขาใดอยู่ ขานั้นก็จะได้รับลอจิก "0" ในขณะที่ถ้า DIP Switch ที่ต่อกับ

ขาใดถูก "OFF" ขานั้นก็จะได้รับลอจิก "1" และเนื่องจากอินพุทที่ขา P_0 - P_5 (Address A_3 - A_9) ต้องเท่ากับอินพุทที่ขา Q_0 - Q_5 ดังนั้นถ้าเราเปลี่ยนแปลงการ set DIP Switch เหล่านี้ ก็จะทำให้ Address บิต A_3 - A_8 ซึ่งต่อกับขา P_0 - P_5 นั้น ต้องเปลี่ยนแปลงตามไปด้วย จึงจะทำให้เอาท์พุทของ 74LS688 Active ได้ ทำให้เราสามารถ เปลี่ยนค่า Address ที่ต้องการจะ Decode ได้ง่ายกว่าวิธีการ Decode แบบ Fixed

สำหรับขา Q_6 นั้นจะต่อกับลอจิก "1" (+5V) และขา P_6 ต่อกับ Address บิต A_9 ในกรณีเช่นนี้ จึงเท่ากับเป็นการบังคับให้ Address ที่จะทำการ Decode ได้นั้น จะต้องมี Address บิต A_9 เป็น ลอจิก "1" เท่านั้น ทั้งนี้ก็เนื่องมาจากว่า ใน IBM/PC ได้ใช้งานเส้น Address เพียง 10 เส้น (คือ A_0 - A_9) ดังนั้นจึงสามารถที่จะอ้าง Address ของ port ได้สูงสุด เพียง 1024 พอร์ท (จากจำนวน 64K พอร์ท) เท่านั้น นอกจากนี้ ในกรณีที่เป็นการอ่านข้อมูล จากพอร์ทของ IBM/PC ข้อมูลในบิต A_9 จะถูกจัดให้มีหน้าที่ในการแบ่งพอร์ททั้ง 1024 พอร์ท ออกเป็น 2 ส่วน (ส่วนละ 512 พอร์ท) อีกด้วย กล่าวคือ ถ้าข้อมูลในบิต A_9 เป็น "0" แล้ว เราจะทำการอ่านข้อมูลได้เฉพาะจากพอร์ทของอุปกรณ์ หรือ ชิบชัพพอร์ทต่าง ๆ ที่อยู่บนเมนบอร์ด (Main Board) ของ IBM/PC เช่น 8253-5,8237-5 หรือ 8259A เท่านั้น แต่ถ้าข้อมูลในบิต A_9 นี้เป็น "1" ก็จะทำการอ่านข้อมูลได้เฉพาะพอร์ทที่อยู่บน Card ต่าง ๆ เท่านั้น ดังนั้นในการกำหนดค่า Address ให้กับพอร์ทที่ถูกสร้างขึ้นบน Card ต่าง ๆ จึงควรจะใช้ค่า Address ที่ Address บิต A_9 มีค่าเป็น "1" คือ Address 0FE00H จนถึง OFFFFH เท่านั้น (Address บิต A_{10}-A_{15} ไม่ถูกใช้ในการ Decode แต่เพื่อความสะดวกจึงกำหนดให้มีค่าเป็น "1" ในฐานสอง ทั้งหมด แต่ในการใช้งานจริงอาจเปลี่ยนให้ Address A_{10}-A_{15} แต่ละบิตมีค่าเป็น "1" หรือ "0" ก็ได้)

ส่วนขา P_7 จะต่อกับสัญญาน AEN โดยมีขา Q_7 ต่อกับลอจิก "0" การต่อในลักษณะนี้ก็เพื่อป้องกันไม่ให้ 74LS688 ทำการ Decode ในระหว่างขบวนการ DMA นั่นเอง เอาท์พุทจากขา $\overline{P = Q}$ ของ 74LS688 นี้ จะถูกนำไปใช้ในการ Enable 74LS138 ซึ่งทำหน้าที่ในการ Decode Address 8 Address ของกลุ่ม Address ที่เราเลือก (โดยใช้ DIP Switch ดังที่ได้กล่าวไว้แล้ว)ซึ่งเอาท์พุทจากขา $\overline{P = Q}$ ของ 74LS688 จะถูกต่อเข้ากับขา G_{2A} ของ 74LS138 ทั้งสองตัว เพราะฉนั้นจึงสามารถ Decode Address ได้ชุดละ 8 Address โดยทำการเปลี่ยนค่าใน Address บิต A_0 - A_2 ได้ 8 ค่า (000_2 - 111_2) ส่วนสัญญานที่ใช้เลือกว่าจะให้ 74LS138 ตัว

ไหน Active นั้น คือสัญญาน IOR และ IOW ที่ต่ออยู่กับขา G_{2B} ของ 74LS138 แต่ละตัว กล่าวคือ 74LS138 แต่ละตัวจะแยกกันเป็นตัว Decode Address port ว่าตัวไหนจะถูก Enable เป็น input port หรือ output port โดยขึ้นอยู่กับขา $\overline{IOR}$ หรือ $\overline{IOW}$ ตามลำดับ

สำหรับการ Decode Address ที่ใช้ใน 8085 Emulator Card นี้ได้เลือกใช้เบอร์ 300H - 307H ซึ่งไม่ได้ถูกใช้งานใน IBM/PC ดังนั้น วงจรนี้จึงสามารถติดต่อกับ port input และ output ได้อย่างละ 8 port คือ port A - port H ซึ่งกำหนดให้แต่ละ port มีค่าตรงกับ Address บิตทั้ง 8 ดังนี้

สัญลักษณ์	Number port
port A	300h
port B	301H
port C	302h
port D	303h
port E	304h
port F	305h
port G	306h
port H	307h

แต่ในการใช้งานจริงไม่ได้ใช้ครบหมดทั้ง 8 port ซึ่ง port ที่เหลือนี้เพื่อไว้ใช้ในการแก้ไขหรือเพิ่มเติมวงจรส่วนอื่น ๆ ในภายหลัง

วงจรในส่วนที่เป็น Input/Output port

จาก Block diagram ในรูป 1.1 นั้น ในส่วนนี้จะทำหน้าที่ในการรับส่งข้อมูลใน port เดียวกัน ซึ่งจะใช้ทำหน้าที่เป็นขาสัญญาน AD0 - AD7 ขาเหล่านี้จะเป็น Address ของข้อมูลขนาด 8 บิตและจะถูก Multiplex กับ Data bus

ขนาด 8 บิต ในช่วงแรกที่ CPU ทำการติดต่อกับ อุปกรณ์ภายนอก 8085 จะส่ง Address ไบท์ต่ำ (A0 - A7) ออกมาทาง ขา AD0 - AD7 เพื่อที่จะชี้ตำแหน่งของหน่วยความจำที่ ต้องการติดต่อ ซึ่งขณะนี้ CPU จะ set ให้ขา ALE (Address Latch Enable)เป็น high เพื่อแสดงว่าได้ทำการส่ง Address ไบท์ต่ำออกมาแล้ว และช่วงเวลาต่อมา เมื่อ Address ถูกวงจรภายนอก Latch Address ไบท์ต่ำไว้แล้ว จากนั้นขาสัญญาน AD0 - AD7 จะใช้สำหรับรับ-ส่งข้อมูล ที่ต้องการอ่าน หรือ ต้องการเขียนแทน

สำหรับวงจรที่ใช้งานจริงในเครื่อง 8085 Emulator ที่ทำหน้าที่ แทนขา AD0 - AD7 ของ 8085 CPU นี้ประกอบไปด้วย Chip support ที่เป็น input และ output port อย่างละ 1 ตัวคือ ตัวที่ใช้เป็น input จะใช้ 74LS245 ซึ่งเป็น Octal Bus Transceivers with 3-state Outputs โดยใช้ขาสัญญาน $\overline{RDA}$ ที่ได้จากการ Decode ของ IC 74LS138 เป็นสัญญาน Enable ให้ทำงาน ส่วน Chip support ตัวที่ทำหน้าที่เป็น output ก็คือ 74LS374 ซึ่งเป็น Octal D-Type Transparent Latches and Edge-Triggered Flip-Flops เหตุที่ต้องใช้input และ output port แยกกันคนละตัวก็เนื่องจาก เหตุผลที่กล่าวไว้ตอนต้นคือ ขาสัญญาน AD0 - AD7 นี้ ในขณะที่ส่ง Address ไบท์ต่ำออกไปจะต้อง Latches ค่าไว้จนกว่า วงจรภายนอกจะนำค่า Address ไป Latches ไว้ก่อน แล้วถึงจะทำให้เป็นสภาวะ high Impedance หลังจากนั้นจึงเป็นขบวนการรับหรือส่ง ข้อมูลต่อไป ซึ่งขาสัญญานที่ใช้ในการควบคุมการ Enable 74LS374 นี้คือ $\overline{WRA}$ ส่วนสัญญาน F1จะใช้เป็นตัวควบคุมให้ 74LS374 อยู่ในสภาวะ high Impedance

วงจรในส่วนที่เป็น Input port

ในส่วนที่เป็น input นี้จะใช้ IC เบอร์ 74LS244 จำนวน 2 ตัวซึ่งเป็น Octal Buffers/Line Drivers/Line Receivers ซึ่งขาสัญญานต่าง ๆ ที่ต่อกับขา input ของ 74LS244 ได้แก่ TRAP, RST7.5, RST6.5, RST5.5, INTR, X1, X2, HOLD, $\overline{RESET\ IN}$, READY และ Vcc

วงจรในส่วนที่เป็น Output port

ในส่วนของ Block ที่เป็น Output port จะแยกเป็น 2 ส่วน คือ ส่วนที่ใช้ทำหน้าที่ ส่งสัญญานต่าง ๆ ออกทางขาสัญญานที่เป็น Output ของ CPU 8085 ซึ่งได้แก่ $\overline{\text{RESET OUT}}$, $\overline{\text{INTA}}$, HLDA, $\overline{\text{RD}}$, $\overline{\text{WR}}$, IO/$\overline{\text{M}}$, SO,S1,SOD และ ALE และอีกส่วนหนึ่งก็คือ ส่วนที่ใช้ควบคุมภายใน คือขา F1-F4 ตัวอย่างเช่น ในขณะที่ CPU กระทำคำสั่ง Halt ขาเอาท์พุท ทั้งหมดควรจะอยู่ในสภาวะ high Impedance โดยการส่งค่า "1" ออกไปที่ขา F1-F4 ก็จะทำให้ ขาสัญญานที่เป็น Output ทั้งหมดกลายเป็น high Impedance

วงจรในส่วนที่เป็น RAM ขนาด 8 K bytes

วงจรในส่วนนี้เป็นการต่อ RAM ขนาด 8 K bytes เข้ากับระบบของเครื่อง Emulator โดยได้ Decode Address ไว้ตั้งแต่ Address ที่ 0000H จนถึง Address 1FFFH ซึ่งการควบคุมการติดต่อกับ RAM ตัวนี้นอกจากจะขึ้นอยู่กับค่า Address ที่ได้ Decode ไว้แล้วยังต้องขึ้นกับขาสัญญาน F1 จาก port H ด้วย ซึ่งประโยชน์ของการต่อ RAM นี้ก็เพื่อให้ CPU 8080 ตัวจริงดึงโปรแกรมที่บรรจุอยู่ใน RAM ไปทำงาน เพราะตัวโปรแกรมที่อยู่ใน RAM นี้สามารถทำการเปลี่ยนแปลงแก้ไขได้จาก ตัวโปรแกรม EMU8085 ที่ต้องใช้คู่กับเครื่อง EMULATOR นี้ โดยหลังจากที่ได้แก้ไขโปรแกรมในส่วนที่เป็นชุดคำสั่งของ CPU - 8085 จนเป็นที่พอใจแล้ว ก่อนที่จะให้ CPU - 8085 ตัวจริง RUN จะต้องทำการ Transfer ข้อมูลลงมาเก็บไว้ใน RAM ตัวนี้ก่อน โดยใช้หัวข้อ menu File/ Transfer/Store into RAM จึงจะสามารถให้ CPU ตัวจริง RUN ได้

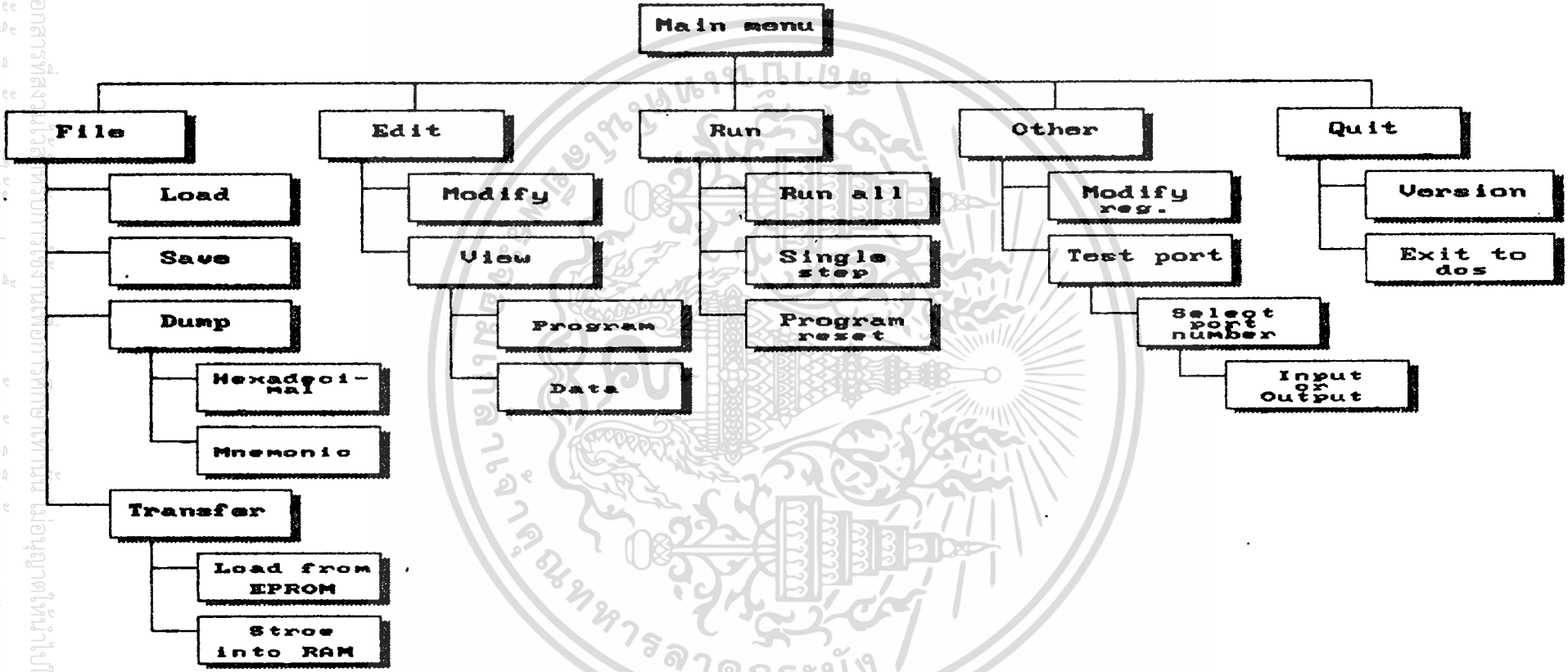
บทที่ 3

โปรแกรมที่ใช้ติดต่อและควบคุม อีมูเลเตอร์ 8085

บทนำ

จุดประสงค์หลักของการเขียน Software ตัวนี้ ก็เพื่อนำไปใช้ในการตรวจซ่อมเครื่องมือ หรือเครื่องจักรอัตโนมัติ โดยใช้ 8085 Emulator เป็นตัวกลางในการติดต่อ โดยแนวทางการเขียนโปรแกรม จะเน้นไปในทางที่จะทำให้ ผู้ที่จะนำโปรแกรมไปใช้ สามารถที่จะใช้ได้โดยง่ายและสะดวกต่อการใช้งาน สำหรับภาษาโปรแกรมที่ใช้เขียนโปรแกรมนี้ จะเลือกใช้ภาษา C ในการเขียน เพราะเนื่องจาก ภาษา C เป็นภาษาที่มีความอ่อนตัว และยังมี Function ต่าง ๆ ให้เลือกใช้งานอีกมากมาย อีกทั้งยังสามารถที่จะเขียนโปรแกรมให้เข้าถึง ภาษา Assembly ได้อีกด้วย

และเนื่องจากความต้องการที่จะทำให้โปรแกรม หรือ Software ที่เขียนขึ้นมานี้ ต้องใช้งานได้ง่าย ผู้ใช้ไม่จำเป็นต้องใช้เวลาในการศึกษามากนัก ก็สามารถที่จะใช้โปรแกรมได้อย่างสะดวก ดังนั้นจึงได้เขียนโปรแกรมให้ทำงานในระบบ Pop-up และ Pull down menu ซึ่งจะมีลักษณะเหมือนกับ ตัวเอดิเตอร์ของเทอร์โบซี หรือของเทอร์โบปาสคาล นั่นเอง ซึ่งเป็นระบบ Menu ที่ให้เลือกคำสั่งโดยใช้ปุ่มลูกศร (Arrow key) 4 ทิศทาง โดยที่ในจอภาพหนึ่งอาจจะมีได้หลาย Menu แต่จะมีเพียงหัวข้อ Menu เดียวเท่านั้นที่จะ Active หรือ กำลังถูกเลือกใช้งานอยู่ หากจะเรียกใช้ Menu อื่น ๆ ก็ต้อง กดปุ่มลูกศรซ้าย หรือ ขวา Menu เก่าจะหายไป บนจอภาพจะปรากฏ Menu ใหม่ขึ้นมาแทน ส่วนข้อเลือกในแต่ละ Menu ก็ต้องใช้คีย์ลูกศรขึ้นหรือลง ข้อไหนถูกเลือกอยู่ ก็จะเห็นเป็น รีเวิร์สหรือขีดเส้นใต้ หรือแบบอื่น ๆ แล้วแต่ ผู้ออกแบบโปรแกรมจะกำหนด แต่สำหรับโปรแกรมนี้ จะใช้วิธีแรก คือ เมื่อเลื่อนคีย์ลูกศรขึ้นหรือลง ไปที่ข้อเลือกอันไหน ก็จะเห็นเป็น รีเวิร์สที่หัวข้อนั้น และยังได้เพิ่มเทคนิคในการสร้าง Attribute ของหัวข้อเลือกให้เด่นชัดขึ้นมา (high video) อีกด้วย ดังนั้นเมื่อกดปุ่มลูกศรขึ้นหรือลง หนึ่งครั้ง ก็จะเกิดการปรับ Attribute ของข้อเลือก 2 ข้อ คือ ปรับ Attribute เดิมให้กลับสู่สภาวะปกติ และปรับ Attribute ของหัวข้อเลือกใหม่ ที่เลื่อนไปหาให้เด่นชัดขึ้น



รูปที่ 3.1 แสดงโครงสร้างระบบ menu ของโปรแกรม EMU8085

การเลื่อนแถบเคอร์เซอร์ยังต้องมีลักษณะวนรอบ (Wrap around) ได้ด้วย กล่าวคือ ถ้าเลื่อนลงไปจนสุดข้อเลือกสุดท้าย แถบเคอร์เซอร์จะวนไปปรากฏ ที่ข้อเลือกแรก หรือถ้าแถบเคอร์เซอร์เลื่อนไปสุดหัวข้อแรก ถ้ากดเลื่อนขึ้นอีก แถบเคอร์เซอร์ก็จะไปปรากฏที่ข้อเลือกตัวสุดท้าย

3.1 โครงสร้างของระบบ Menu

โครงสร้างของโปรแกรมและระบบการทำงานของ แต่ละหัวข้อ Menu นั้น มีโครงสร้างดังแสดงได้ดังรูปที่ 3.1 โดยจะแบ่งเป็นหัวข้อ Menu หลัก (main menu) ได้ 5 หัวข้อ คือ File, Edit, Run, Other และ Quit โดยหน้าที่การทำงานในแต่ละหัวข้อ menu นี้จะแตกต่างกันไปดังนี้คือ

Menu File

- จะเป็นการกระทำเกี่ยวกับ File ต่าง ๆ ซึ่งได้แบ่งเป็นหัวข้อ Menu ย่อยได้อีก 4 หัวข้อคือ Load, Save, Dump และ Transfer การ Load และ Save นั้นจะเป็นการทำงานติดต่อกับ Disk ส่วนการ Dump นั้นจะเป็นการพิมพ์ File program ที่ได้ Load เข้ามาอยู่ในหน่วยความจำของ IBM/PC แล้ว พิมพ์ออกทางเครื่องพิมพ์ ซึ่งก็สามารถก็จะเลือกได้ด้วยว่า จะให้พิมพ์ในรูปแบบใด ระหว่างการพิมพ์ในแบบ Hexadecimal (Dump ออกมาเป็นเลขฐาน 16) กับการพิมพ์ในแบบ Mnemonic คือจะมีทั้งค่า Address Opcode และรหัส Mnemonic ออกมาซึ่งจะสะดวกมากในการที่จะนำมาศึกษาการทำงานของโปรแกรม สำหรับหัวข้อ Transfer นั้น จะมี 2 หัวข้อ Menu ให้เลือกคือ Load from EPROM คือการดึงค่าข้อมูลที่อยู่ในตัว EPROM หรือ ROM ของระบบที่ต้องการจะทดสอบนั้น เข้ามาเก็บไว้ยังหน่วยความจำของ IBM/PC ซึ่งโปรแกรม EMU8085 ได้ทำการ Allocate ไว้ ส่วนอีกหัวข้อหนึ่งคือ การนำตัวโปรแกรมที่อยู่ในหน่วยความจำของ IBM/PC ไปเก็บไว้บน RAM ในเครื่องอีมูเลเตอร์ ซึ่งก็คือหัวข้อ Stroe into RAM

Menu Edit

- หน้าที่หลักของหัวข้อ Menu นี้จะมีอยู่ 2 อย่างคือ การแก้ไขหรือเปลี่ยนแปลงค่าของข้อมูลใน Address ต่าง ๆ (Modify) และการขอดูค่าของข้อมูล หรือตัวโปรแกรมใน Address ต่าง ๆ ตามที่ต้องการได้ (View)

Menu Run

- หน้าที่การทำงานของหัวข้อ Menu นี้ เป็นหัวใจสำคัญของเครื่องอีมูเลเตอร์ เพราะจะใช้ Menu นี้ในการที่จะ Run ตัวโปรแกรมที่ต้องการจะตรวจหาข้อผิดพลาด ของทั้งทางด้าน Hard ware และ Soft ware ของระบบที่ต้องการจะตรวจซ่อมนั้น ๆ ได้ ซึ่งก็ได้ทำการแบ่งเป็น 3 หัวข้อ Menu ย่อยคือ Run all, Single step และ Program reset ส่วนรายละเอียดในแต่ละหัวข้อ Menu ย่อยนั้น จะได้กล่าวถึงในส่วนถัดไป

Menu Other

- ในหัวข้อ Menu นี้จะจัดแบ่งเป็น 2 หัวข้อหลัก ๆ คือ การเปลี่ยนแปลงค่าข้อมูลหรือค่าสภาวะต่าง ๆ ของ Register แต่ละตัวได้ โดยเลือกหัวข้อ Menu Modify reg. ส่วนอีกหัวข้อหนึ่งคือ การทดสอบ Input - Output port ของเครื่อง 8085 EMULATOR ซึ่งลักษณะการใช้งานบางอย่างนั้นบางครั้งก็อาจจะใช้ หัวข้อ Menu นี้ทำการทดสอบได้

Menu Quit

- Menu นี้ จะถูกใช้งานก็ต่อเมื่อ ต้องการที่จะเลิกการทำงาน จากโปรแกรม EMU8085 แล้ว เพื่อที่จะกลับออกไประบบปฏิบัติการหรือดอส

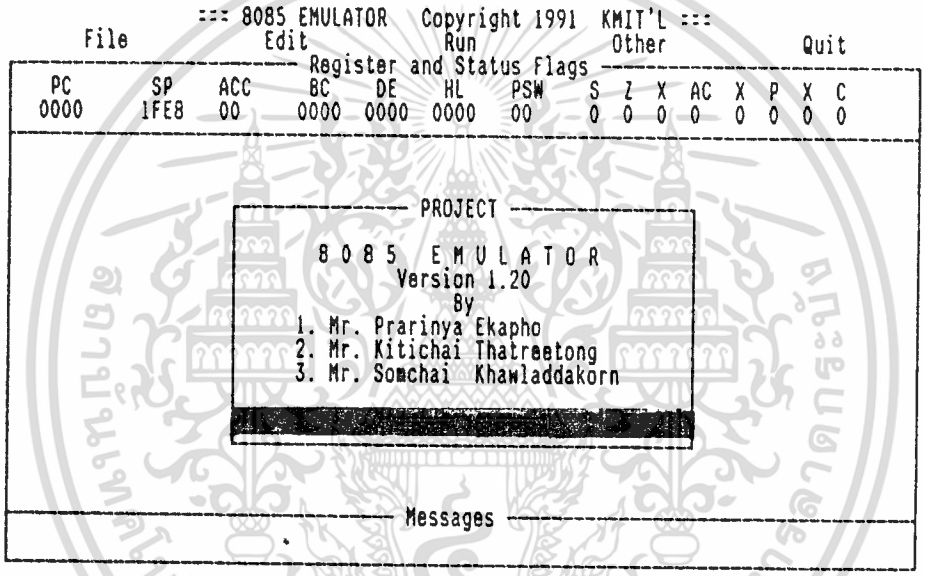
3.2 วิธีการเรียกโปรแกรม EMU8085 มาใช้งาน

โปรแกรม EMU8085 นี้สามารถที่จะเรียกขึ้นมาทำงานจาก ดอส ได้โดยตรง เช่น C:\>emu8085 <ENTER> แต่ก่อนที่จะเรียกโปรแกรมนี้จะต้องแน่ใจเสียก่อนว่าระบบที่จะทำการทดสอบนั้น มีไฟจ่ายให้กับขา Vcc ของ CPU-8085 เพราะโปรแกรมนี้จะ

ทำการตรวจสอบก่อนว่าถ้าไม่มีแรงดันจ่ายให้กับขา Vcc ของ CPU-8085 จะไม่สามารถเรียกโปรแกรมนี้มาทำงานได้ และจะมีข้อความบอกดังนี้คือ

Your system is not ready !

แต่ถ้าหากเข้าโปรแกรมได้แล้วจะปรากฏหน้าจอแรก ดังแสดงในรูปที่ 3.2



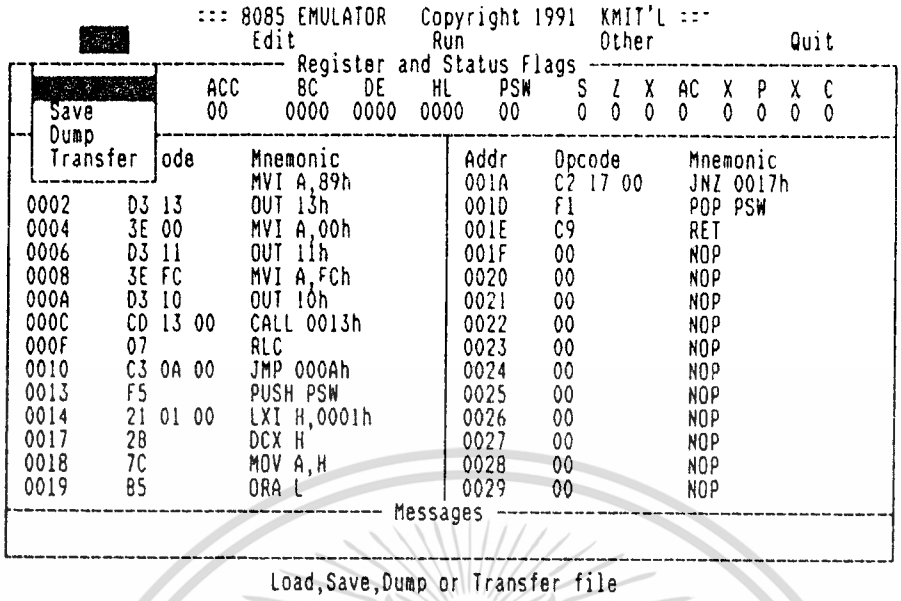
รูปที่ 3.2 The First menu screen

3.3 การใช้งานระบบ menu

Main menu นั้นจะมีหัวข้อหลัก ๆ ดังนี้คือ File, Edit, Run, Debug และ Quit สำหรับการเรียก Menu ต่าง ๆ ขึ้นมาทำงานนั้นทำได้โดยการกดฟังก์ชั่นคีย์ F10 ซึ่งก็จะมีเพียงหัวข้อ menu เดียวเท่านั้นที่ Active (reverse video) จากนั้นให้ทำการเลือกหัวข้อ menu ที่ต้องการทำงานโดยวิธีการเลือกมีอยู่ 2 วิธีคือ วิธีแรกให้กดตัวอักษรตัวแรกที่เป็นอักษรตัวใหญ่ของแต่ละ menu แล้วกด Enter หรือ กดคีย์ลูกศรลง (DOWN ARROW) ส่วนอีกวิธีหนึ่งคือใช้คีย์ลูกศร ซ้าย-ขวา (LEFT/RIGHT ARROW KEY) เลื่อนไปยังหัวข้อ menu ที่ต้องการแล้วกด Enter หรือ กดคีย์ลูกศรลง และลักษณะการเลือกหัวข้อ menu นี้จะเป็นแบบวนรอบ (WRAP AROUND) กล่าวคือเมื่อเลื่อนหัวข้อ menu ที่ Active ไปยังหัวข้อซ้ายสุดของจอภาพและถ้ากดคีย์ลูกศรไปทางซ้ายอีกก็จะไป Active ที่หัวข้อ menu ทางขวาสุด หรือ ถ้าหัวข้อ menu ที่ Active อยู่ อยู่ทางขวาสุดแล้วกดคีย์ลูกศรไปทางขวาอีกก็จะเป็นในทำนองเดียวกัน ส่วนรายละเอียดการทำงานในแต่ละหัวข้อ menu นั้นจะได้อธิบายในส่วนต่อไป

Menu File

เมื่อเลือกหัวข้อ menu นี้จะเกิด pull-down menu ขึ้นมา และจะมีหัวข้อ menu ต่าง ๆ ให้เลือกทำงานต่าง ๆ ที่เกี่ยวกับ file อีกเช่น Load file จากแผ่น disk, Save file ไปเก็บไว้ใน disk หรือ อาจจะพิมพ์ file ออกทางเครื่องพิมพ์ (Printer)ซึ่งสามารถที่จะเลือกได้แบบ Mnemonic ส่วนหัวข้อสุดท้ายคือการ Transfer file ซึ่งก็จะมีทั้งการ Transfer ข้อมูลจาก Eprom เข้ามาเก็บไว้ใน RAM ของ PC และ Transfer file ที่เก็บไว้ใน PC ไปไว้ใน RAM บนเครื่อง EMULATOR 8085 สำหรับรายละเอียดในแต่ละหัวข้อ menu มีดังนี้



รูปที่ 3.3 The File menu

Load

Menu นี้จะเป็นการ load file ที่เก็บไว้ใน disk เข้ามาไว้ใน program โดยจะต้องบอกทั้ง drive และ path ที่ file นั้นอยู่ และจะต้องพิมพ์ file และส่วนขยาย (Extension)ให้ถูกต้องด้วย แต่ถ้าไม่ใส่ชื่อ drive และ path มันจะหาเฉพาะ drive และ directory ปัจจุบันเท่านั้นโดย file ที่ load เข้ามานี้จะต้องเป็น file program ของ CPU เบอร์ 8085 เท่านั้น และต้องเก็บในรูปแบบ binary file โดยตัวโปรแกรม CPU 8085 นี้จะทำการ Unassemble เป็นรหัส Mnemonic ของ CPU เบอร์ 8085 โดยอัตโนมัติ ซึ่งจะแสดงให้เห็นตั้งแต่ Address 0000h เป็นต้นไปที่ช่องหน้าต่าง Edit

Save

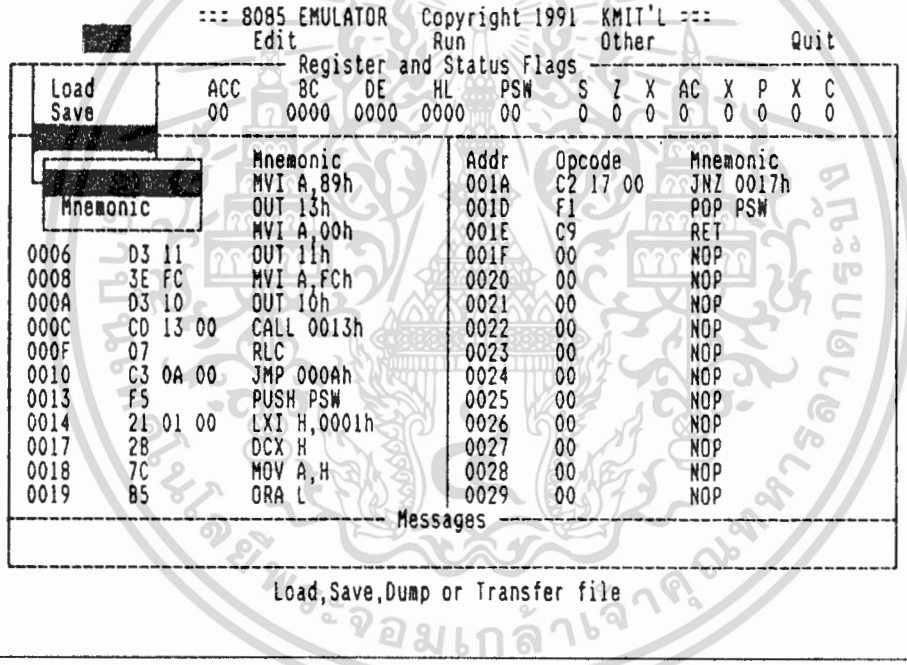
หัวข้อนี้จะเป็นการ save file ที่ถูก load เข้ามาในหน่วยความจำของ PC ซึ่งอาจจะ load มาจากแผ่น disk หรือจาก EPROM ก็ตาม และยังสามารถแก้ไขโปรแกรมที่ load เข้ามานี้ได้โดยใช้ menu edit ส่วนหลักการในการตั้งชื่อ file ที่จะ

save จะต้องใช้หลักการเดียวกับการตั้งชื่อ file ของ dos

Dump

หัวข้อเลือกใน menu นี้จะทำให้สามารถพิมพ์ตัวโปรแกรมที่ถูก load เข้ามาแล้วออกทางเครื่องพิมพ์ได้ ซึ่งสามารถที่จะเลือกพิมพ์ได้ 2 แบบคือ แบบ Hexadacimal และแบบ Mnemonic

* Hexadecimal
* Mnemonic



รูปที่ 3.4 The File / Dump menu

Hexadecimal

การเลือกพิมพ์ในลักษณะนี้จะเป็นการ dump file ออกทางเครื่องพิมพ์ในรูปแบบเลขฐาน 16 ซึ่งเมื่อเลือก menu นี้แล้วจะมีการถามค่า Address เริ่มต้นที่จะให้พิมพ์ โดยให้ป้อนเป็นเลขฐาน 16 จำนวน 2 ไบต์ หลังจากนั้นให้กดคีย์ Enter จะมีคำเตือนให้ check เครื่องพิมพ์ในอยู่สภาวะพร้อม (READY) ก่อนที่จะเริ่มพิมพ์ เมื่อเครื่องพิมพ์

พร้อมแล้วให้กดคีย์ใด ๆ เพื่อเริ่มพิมพ์ โดยลักษณะการพิมพ์ใน menu นี้จะส่งข้อมูลออกไปยังเครื่องพิมพ์ทีละ 1 หน้ากระดาษ แล้วรอกดคีย์ใด ๆ เมื่อเครื่องพิมพ์พร้อมที่จะพิมพ์ต่อไป ซึ่งในแต่ละหน้าจะมีหมายเลขหน้าพิมพ์ออกมาด้วย โดยจะพิมพ์ทั้ง Address และข้อมูล จำนวน 8 ไบต์ต่อ 1 บรรทัด ใน 1 หน้าจะพิมพ์ได้ 50 บรรทัด เมื่อพิมพ์ไปครบ 1 หน้า แล้วหากต้องการจะยกเลิกการพิมพ์ให้กดคีย์ ESCAPE แต่ถ้าต้องการพิมพ์หน้าต่อไปก็กดคีย์ใด ๆ เมื่อเครื่องพิมพ์พร้อม

Mnemonic

ความสามารถอีกอย่างหนึ่งในการส่งพิมพ์ตัวโปรแกรมที่ load เข้ามาของโปรแกรม EMU8085 นี้คือการพิมพ์ในลักษณะ Mnemonic ซึ่งจะมีทั้งค่า Address, opcode และรหัส Mnemonic ของ CPU เบอร์ 8085 ทำให้สะดวกในการตรวจสอบโปรแกรม ข้อดีอีกอย่างหนึ่งก็คือ ตัวโปรแกรมที่บรรจุอยู่ใน EPROM หรือ ROM ของระบบที่ต้องการจะตรวจสอบนั้น เราไม่สามารถรู้ได้เลยว่าตัวโปรแกรมเขียนไว้อย่างไร ดังนั้น function นี้จะช่วยแก้ปัญหานี้ได้มาก

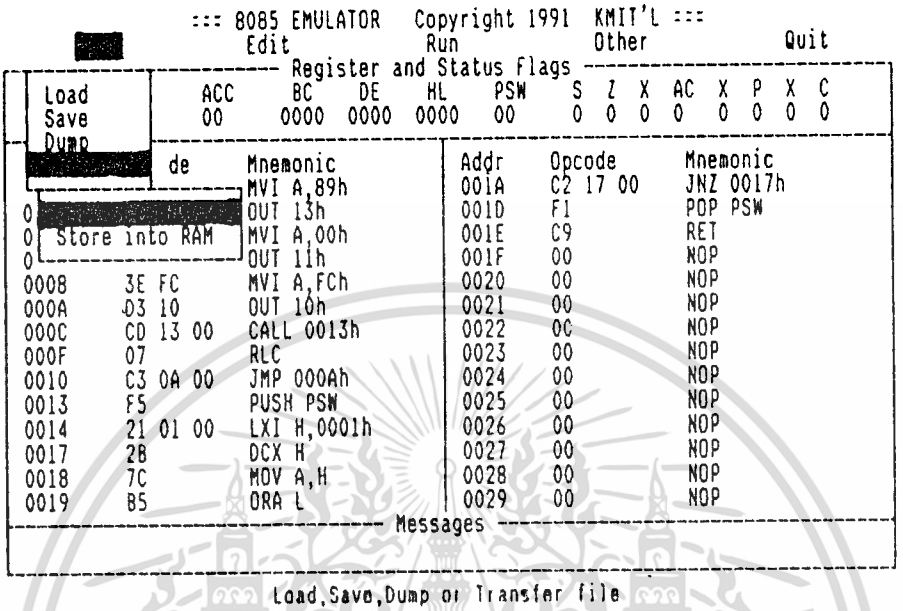
ส่วนวิธีการใช้งาน menu นี้จะคล้ายกับการส่งพิมพ์ในแบบ Hexadecimal คือ จะมีการถามค่า Address เริ่มต้นที่ต้องการจะเริ่มพิมพ์ ลักษณะการส่งข้อมูลไปยังเครื่องพิมพ์ก็จะส่งไปทีละ 1 หน้าเช่นกันโดยจะพิมพ์ได้หน้าละ 52 บรรทัด (ขนาดกระดาษ A-4) แล้วจะหยุดรอให้ป้อนกระดาษใหม่เพื่อเริ่มพิมพ์หน้าถัดไป แต่ถ้าหากว่าต้องการเลิกพิมพ์ก่อนจบ file ให้กด ESCAPE เพื่อยกเลิกการพิมพ์

Transfer

Menu นี้เป็นหัวข้อการทำงานแรกที่จะต้องเลือกในการที่จะเริ่มตรวจสอบระบบที่ต้องการ หน้าที่ของหัวข้อนี้ก็คือ การ load ตัวโปรแกรมหรือข้อมูลที่อยู่ในตัว EPROM หรือ ROM ของระบบนั้น ๆ เข้ามายังหน่วยความจำของ PC และหลังจากนั้นก็จะต้องนำโปรแกรมที่ load เข้ามาแล้วนี้นำไปเก็บไว้ในหน่วยความจำชั่วคราว (RAM) ของเครื่อง EMULATOR 8085 ด้วยซึ่งแบ่งเป็นหัวข้อ menu ย่อยได้ 2 หัวข้อดังนี้

* Load from EPROM

* Store into RAM



รูปที่ 3.5 The File / Transfer menu

Load from EPROM

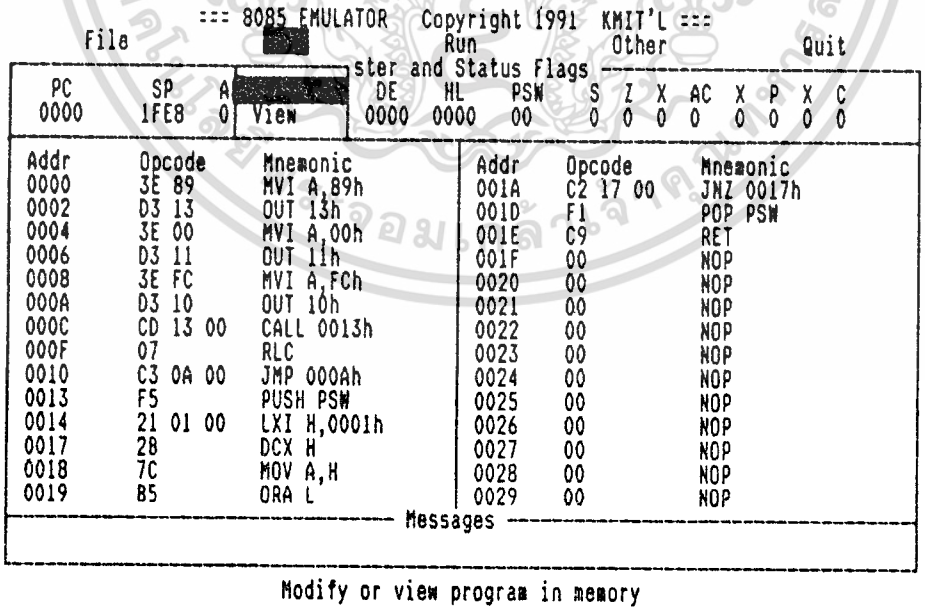
การใช้งาน menu นี้ก็เพื่อที่จะทำการ load ตัวโปรแกรมหรือข้อมูลที่อยู่ในตัว EPROM ของระบบที่ต้องการจะตรวจสอบโดยจะใช้หลักการดึงค่าข้อมูลเข้ามาทีละ Address แล้วนำเข้ามาเก็บไว้ในหน่วยความจำส่วนหนึ่งใน PC ที่ได้ทำการ Allocate ไว้ซึ่งก่อนเลือกทำงานในหัวข้อ menu นี้ต้องแน่ใจว่าได้เชื่อมต่อสายทาง HARD WARE ระหว่าง PC กับเครื่อง EMULATOR 8085 และจากเครื่อง EMULATOR กับระบบที่ต้องการจะตรวจสอบอย่างถูกต้องแล้วจึงจะสามารถใช้ function นี้ได้ และหลังจาก load ข้อมูลเข้ามาเสร็จเรียบร้อยแล้วควรจะถอดตัว EPROM ของระบบนั้น ๆ ออก เพราะอาจจะมีการรบกวนกับตัววงจร EMULATOR ที่ได้สร้างขึ้น

Store into RAM

menu นี้จะต้องเลือกหลังจากที่ได้ทำการ load ตัวโปรแกรมหรือข้อมูลจาก EPROM ของระบบภายนอกแล้ว ซึ่งหน้าที่ของ menu นี้ก็คือจะนำตัวโปรแกรมหรือข้อมูลที่ได้ถูก load มาแล้วนำไปเก็บไว้ในหน่วยความจำชั่วคราว (RAM) ของเครื่อง EMULATOR 8085 ซึ่งจะถูกเก็บไว้จนกว่าจะมีการ RUN โปรแกรมในแบบ Read time ซึ่ง CPU ที่อยู่ในเครื่อง EMULATOR จะมาอ่านตัวโปรแกรมใน RAM ตัวนี้ไป Execute

Menu Edit

เมื่อเลือกหัวข้อ menu นี้จะเกิด pull-down menu ขึ้นมาซึ่งจะมีหัวข้อ menu ย่อยให้เลือกอีก 2 หัวข้อ คือ Modify และ View และ menu นี้จะทำงานได้ก็ต่อเมื่อได้ทำการ load ตัวโปรแกรมของ CPU 8085 เข้ามาใน PC แล้วเท่านั้น ไม่ว่าตัวโปรแกรมนี้จะถูก load มาจาก EPROM ของระบบภายนอก หรือจากแผ่น disk ก็ตาม ส่วนรายละเอียดการทำงานในแต่ละหัวข้อ menu จะได้กล่าวในส่วนถัดไป



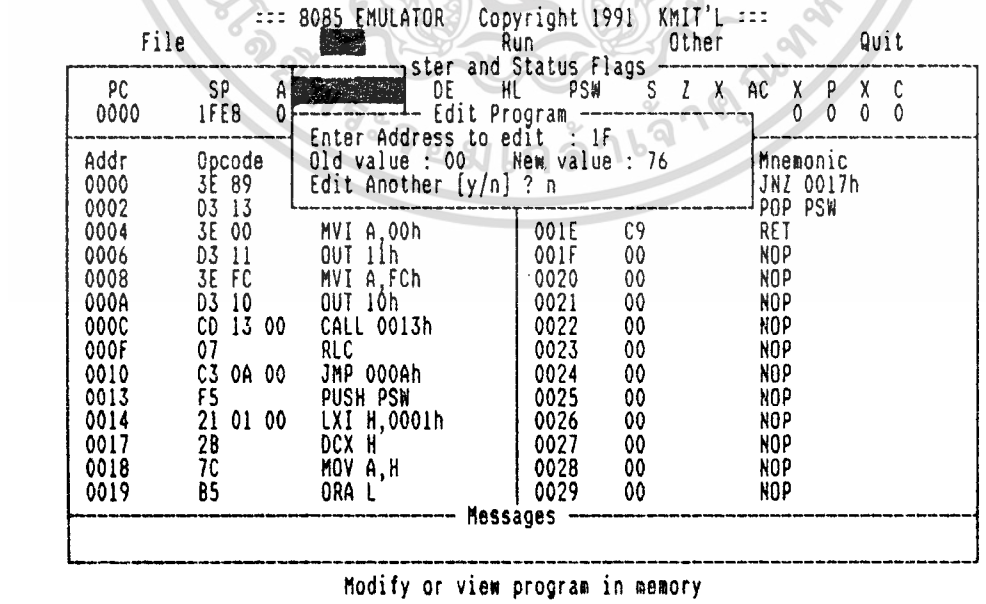
รูปที่ 3.6 The Edit menu

Modify

หัวข้อ menu นี้จะเป็นการเปลี่ยนแปลงหรือแก้ไขค่าข้อมูลใน Address ต่างๆ ของโปรแกรมที่ load เข้ามา ซึ่งเมื่อเลือกแล้วจะมีหน้าต่าง Edit Program pop-up ขึ้นมา และจะมีคำถามให้ป้อนค่า Address ที่ต้องการจะแก้ไข โดยค่า Address ที่ป้อนเข้าไปต้องเป็นเลขฐาน 16 เท่านั้น หลังจากป้อนค่า Address ที่ต้องการจะแก้ไขแล้วจะมีการแสดงค่าข้อมูลเดิมออกมา แล้วให้ทำการป้อนค่าใหม่ที่ต้องการจะเปลี่ยน หลังจากนั้นจะมีคำถามว่าต้องการจะแก้ไขใน Address อื่นอีกหรือไม่ โดยให้ตอบ Y หรือ N โดยถ้าตอบ Y ก็จะให้ป้อนค่า Address ที่จะทำการแก้ไขต่อไป แต่ถ้าตอบ N ก็จะเป็นการยกเลิกการแก้ไขตัวโปรแกรม

หมายเหตุ

หลังจากที่ได้ทำการเปลี่ยนแปลงแก้ไขค่าของข้อมูลใน Address ต่าง ๆ เสร็จ แล้ว ควรจะเลือกหัวข้อ Menu View ทุกครั้ง เพื่อให้แสดงตัวโปรแกรมที่หน้าจอใหม่อีก เพราะการแก้ไขข้อมูลตาม Address ต่าง ๆ ที่ได้ทำไปแล้วนั้น จะไปแก้ไขเปลี่ยนแปลงข้อมูลใน Memory แต่ข้อมูลเดิมที่แสดงอยู่ที่หน้าจอนั้น ยังเป็นข้อมูลเก่าอยู่ ดังนั้นจึงควร View Address ดูใหม่ทุกครั้งที่มีการ Modify ค่าใน Address ต่าง ๆ

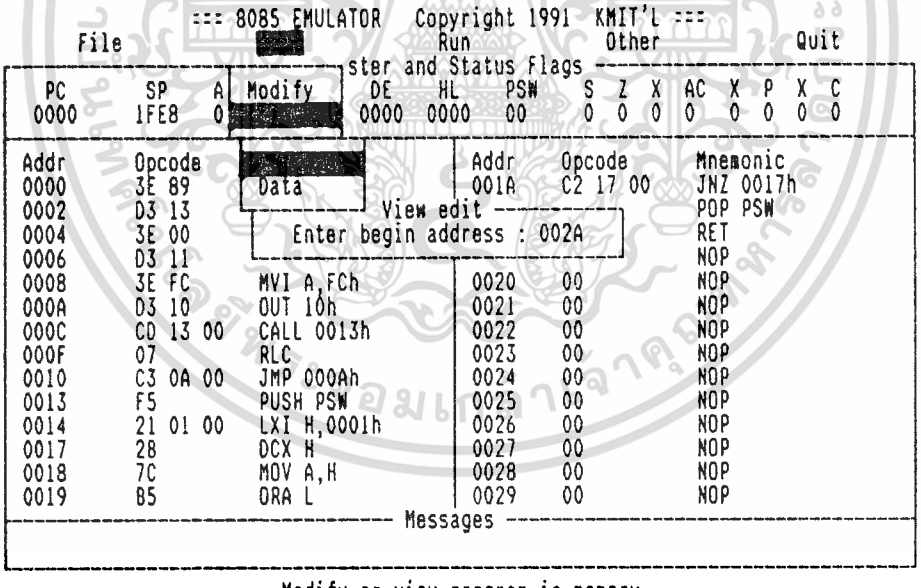


รูปที่ 3.7 The Edit / Modify menu

View

เนื่องจากการแสดงตัวโปรแกรมที่ได้ load เข้ามาจาก EPROM ของระบบภายนอกหรือจากแผ่น disk ก็ตามจะแสดงได้เพียง 26 บรรทัดโดยแบ่งแสดงเป็น 2 ส่วน คือทางด้านซ้ายและด้านขวา ซึ่งแสดงได้ด้านละ 13 บรรทัด ดังนั้นวิธีที่จะดูโปรแกรมใน Address ถัดไป หรือ Address ใด ๆ ของโปรแกรมก็ทำได้โดยการเลือกหัวข้อ menu นี้ ซึ่งจะถามค่า Address เริ่มต้นที่ต้องการจะดู (View) หลังจากนั้นก็จะแสดงตัวโปรแกรมโดยเริ่มตั้งแต่ Address นั้นไปจนครบ 26 บรรทัด ดังได้กล่าวไว้ข้างต้น แต่ค่า Address ที่ป้อนเข้าไปจะต้องเป็นเลขฐาน 16 (Hexadecimal) เท่านั้น และในหัวข้อ Menu นี้นั้น จะแบ่งเป็นหัวข้อ Menu ย่อย 2 หัวข้อคือ Program และ Data

* Program
* Data



รูปที่ 3.8 The Edit / View menu

Program

หลังจากเลือกหัวข้อ Menu นี้แล้วจะมี pop-up window ขึ้นมาเพื่อจะถามค่า Address เริ่มต้นของตัวโปรแกรมที่ต้องการจะให้แสดงที่หน้าจอ และค่า Address ที่ป้อนนี้จะต้องเป็นเลขฐานสิบหก (Hexadecimal) เท่านั้น

หมายเหตุ

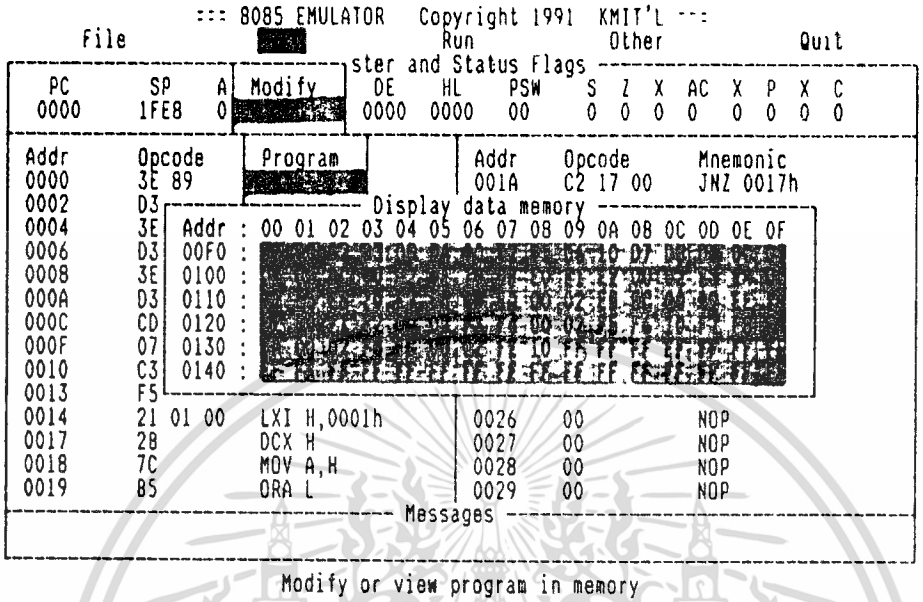
ค่า Address ที่ป้อนเข้าไปนี้จะต้องเป็น Address ที่เป็นค่าเริ่มต้นของชุดคำสั่งของ CPU เบอร์ 8085 เท่านั้น จะเป็นค่า Address ที่บรรจุ Operand ตัวที่ 1 หรือ Operand ตัวที่ 2 ไม่ได้เพราะหลังจากป้อนค่า Address ไปแล้วโปรแกรม EMU8085 จะทำการ Unassembler ทุกครั้งแล้วจึงจะแสดงที่หน้าจอ ดังนั้นถ้าหาก ป้อนค่า Address ที่บรรจุ Operand ตัวที่ 1 หรือ Operand ตัวที่ 2 อยู่ก็จะทำให้การแสดงตัวโปรแกรม ที่หน้าจอผิดพลาดได้

Data

หลังจากเลือกหัวข้อ Menu นี้แล้วจะมี pop-up window ขึ้นมาเพื่อจะถามค่า Address เริ่มต้นของตัวโปรแกรมที่ต้องการจะให้แสดงที่หน้าจอ และค่า Address ที่ป้อนนี้จะต้องเป็นเลขฐานสิบหก (Hexadecimal) เท่านั้น

หมายเหตุ

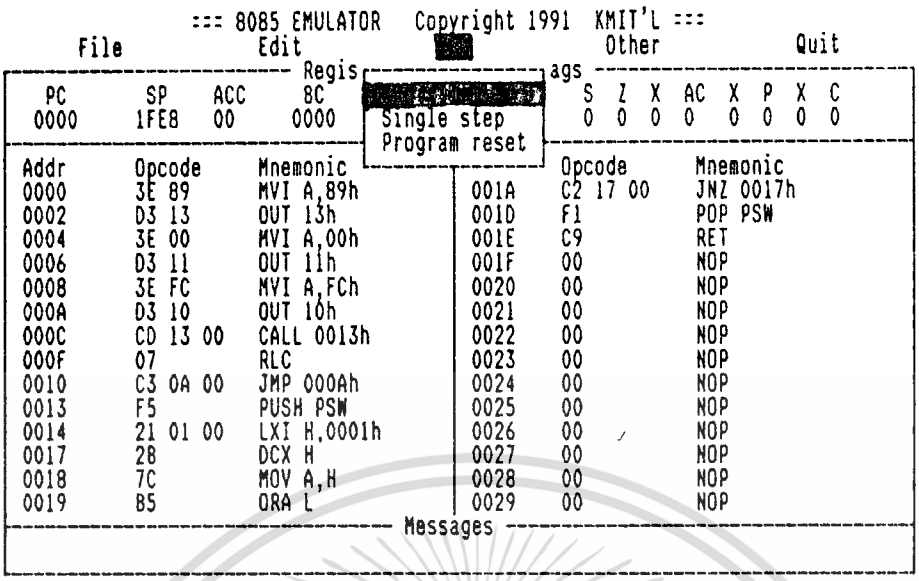
ค่า Address ที่ป้อนเข้าไปนี้จะต้องเป็น Address ที่เป็นค่าเริ่มต้นของชุด Address ครั้งละ 16 บิต ตัวอย่างเช่นถ้าต้องการดูค่าข้อมูลใน Address 1FE8h ก็ควรจะป้อนค่า Address เป็น 1FE0h, 1FD0h, 1FC0h, 1FB0h หรือ 1FA0h อันใดอันหนึ่ง อย่างไรก็ตามการแสดงค่าข้อมูลตาม Address ที่ต้องการนั้น จะแสดงได้ครั้งละ 6 แถว ๆ ละ 16 ค่า (00 - 0F) ดังนั้นจึงแสดงได้ครั้งละ 96 ค่า ซึ่งการแสดงค่าข้อมูลนี้จะเป็นลักษณะ pop-up window ขึ้นมาแสดง และเมื่อกดคีย์ ใด ๆ หน้าต่างแสดงค่าข้อมูลนี้ก็จะหายไป



รูปที่ 3.9 The Edit / View / Data menu

Menu Run

หน้าที่ของ Menu นี้จะเป็นการนำโปรแกรมที่ Load มาจาก EPROM หรือจาก File ใน Disk มา Run ซึ่งก่อนที่จะใช้ Menu นี้ได้จะต้องแน่ใจว่าได้ทำการ Load File Program ของ CPU เบอร์ 8085 เข้ามาในตัวโปรแกรม EMU8085 แล้ว และในหัวข้อ Memu นี้ จะมีอีก 3 หัวข้อ Menu ย่อยคือ Run all , Single step และ Program reset ส่วนรายละเอียดในแต่ละหัวข้อ Menu นั้น จะได้กล่าวต่อไป



รูปที่ 3.10 The Run menu

Run all

ลักษณะการใช้งาน Menu นี้จะเป็นการนำโปรแกรมที่เขียนขึ้นมาด้วย ชุดคำสั่งของ CPU เบอร์ 8085 มา Run ทีละ Address โดยหลังจากที่ได้เลือกทำงานในหัวข้อ Menu นี้แล้ว จะมี pop-up window Address define ขึ้นมาเพื่อถามค่า Address เริ่มต้นและค่า Address สุดท้ายที่จะให้โปรแกรม Run หลังจากนั้น โปรแกรมของ 8085 ก็จะถูกทำงานไปทีละคำสั่งตั้งแต่ Address เริ่มต้นจนถึงค่า Address สุดท้าย แต่ก่อนที่จะทำงานในหัวข้อ Menu Run all นี้ควรจะทำการ Reset Program ก่อนทุกครั้ง เพื่อความแน่ใจว่าโปรแกรมที่เขียนขึ้นมาจากชุดคำสั่งของ CPU เบอร์ 8085 จะทำงานได้อย่างไม่ผิดพลาด ซึ่งในระหว่างที่โปรแกรมของ CPU เบอร์ 8085 กำลังทำงานอยู่นั้น ค่าสภาวะต่าง ๆ เช่น Program counter, Stack pointer หรือค่าใน Register ต่าง ๆ จะเปลี่ยนแปลงไปตามผลของการกระทำชุดคำสั่งแต่ละคำสั่งไปโดยตลอด และถ้าหากต้องการจะหยุดการทำงานก่อนที่จะถึงค่า Address สิ้นสุดที่กำหนดไว้นั้น จะทำได้ 2 วิธีดังนี้คือ วิธีแรกถ้าหากต้องการหยุดการทำงานดังกล่าว ให้ทำการกดคีย ESCAPE ซึ่งจะมีข้อความนี้บอกในขณะที่โปรแกรมกำลังทำงานอยู่ ในช่องหน้าต่าง Messages นี้อยู่แล้ว และเมื่อทำการหยุดการทำงาน (Break) ของโปรแกรมได้แล้ว

ค่าสภาวะต่าง ๆ ที่ได้จากการสั่งให้โปรแกรมทำงาน จะค้างไว้อยู่หลังจากถูกสั่งให้โปรแกรมหยุดการทำงานแล้ว ส่วนอีกวิธีหนึ่งคือการสั่งให้โปรแกรมหยุดทำงานโดยการให้สัญญาน Reset กับ CPU หรือตัดแหล่งจ่ายไฟที่จ่ายให้กับตัว CPU ออก โปรแกรมที่กำลังทำงานอยู่ก็จะถูก Reset ไปโดยอัตโนมัติ ซึ่งค่าสภาวะต่าง ๆ เช่น Program counter, Stack pointer หรือค่าใน Register ต่าง ๆ จะถูก Set ให้เป็นค่าเริ่มต้น

Single step

การทำงานในลักษณะนี้นั้นจะเป็นการสั่งให้โปรแกรม ที่เขียนด้วยชุดคำสั่งของ CPU เบอร์ 8085 ทำงานทีละคำสั่งไปตามลำดับจากค่า Address เริ่มต้นจนถึงค่าสุดท้ายที่ได้กำหนดไว้ ซึ่งถ้าหากมีการเลือก Menu นี้ จะมีการถามค่า Address เริ่มต้นและค่า Address สิ้นสุดก่อนทุกครั้ง และหลังจากป้อนค่า Address สุดท้ายที่จะให้โปรแกรมทำงานแล้ว Address แรกจะถูก Run จากนั้นจะรอให้ผู้ใช้ทำการกดคีย์ F8 เพื่อให้ ทำงานในคำสั่งถัดไป ซึ่งในขณะที่รอให้ผู้ใช้กดคีย์ F8 เพื่อทำการ Run ในคำสั่งถัดไปนั้น ผู้ใช้สามารถที่จะทำงานใน Function อื่น ๆ ได้ อีกดังนี้คือ

F5 - Other Address เป็นการขอดูตัวโปรแกรมใน Address อื่น ๆ

F6 - Display data memory เป็นการขอดูค่าข้อมูลใน Address ต่าง ๆ

F7 - Modify register เป็นการเปลี่ยนแปลงค่าสภาวะใน Register ต่าง ๆ รวมทั้งค่าใน Program counter และ Stack pointer ด้วย

F8 - Next step เป็นการสั่งให้โปรแกรมทำงานในคำสั่งถัดไป

ESCAPE - Break เป็นการยกเลิกการทำงานของโปรแกรม

Program reset

Menu นี้จะเปรียบเสมือนการ reset การทำงานของ CPU ซึ่งจะมีการส่งสัญญาน reset ออกไปให้กับระบบภายนอกด้วย และจะทำการ Reset ค่าสภาวะของ Register ต่าง ๆ ให้เป็นค่าเริ่มต้นด้วย

Menu Other

สำหรับหัวข้อ menu นี้จะมีหน้าที่การใช้งานอยู่ 2 ลักษณะคือ การแก้ไขหรือเปลี่ยนแปลงค่าข้อมูลใน register ต่าง ๆ ได้ซึ่งจะช่วยเพิ่มประสิทธิภาพในการตรวจหาข้อผิดพลาดของโปรแกรมได้ง่ายขึ้น ส่วนอีกอย่างหนึ่งก็คือการทดสอบ port ต่าง ๆ ของเครื่อง EMULATOR โดยตรงซึ่งดูได้จากวงจรภายในเครื่อง EMULATOR นี้ ซึ่งประโยชน์ของการ test port โดยใช้ menu นี้คือสามารถทำการทดสอบ ส่ง หรือ รับ ข้อมูลผ่านทาง port ของเครื่อง EMULATOR ได้โดยตรง ซึ่ง port แต่ละ port นั้นก็จะเป็นขาสัญญาณของ CPU เบอร์ 8085 นั่นเอง เพราะฉะนั้นถ้าเราต้องการทดสอบขาสัญญาณบางขาของ CPU เบอร์ 8085 โดยในขณะที่ยังไม่ต้องการ RUN โปรแกรมก็สามารถเลือกใช้ menu นี้ได้ ซึ่งหัวข้อ menu ทั้ง 2 ที่กล่าวมานี้ มีดังนี้ คือ Modefy reg และ Test port

```
              === 8085 EMULATOR   Copyright 1991  KMIT'L ===
  File              Edit                Run                           Quit
 ---------------------- Register and Status Fla -------------------------
  PC      SP     ACC    BC    DE    HL    PSW                X   P   X   C
  0000    1FE8   00     0000  0000  0000  00    Test port    0   0   0   0
 ------------------------------------------------------------------------
  Addr    Opcode     Mnemonic         Addr   Opcode     Mnemonic
  0000    3E 89      MVI A,89h        001A   C2 17 00   JNZ 0017h
  0002    D3 13      OUT 13h          001D   F1         POP PSW
  0004    3E 00      MVI A,00h        001E   C9         RET
  0006    D3 11      OUT 11h          001F   00         NOP
  0008    3E FC      MVI A,FCh        0020   00         NOP
  000A    D3 10      OUT 10h          0021   00         NOP
  000C    CD 13 00   CALL 0013h       0022   00         NOP
  000F    07         RLC              0023   00         NOP
  0010    C3 0A 00   JMP 000Ah        0024   00         NOP
  0013    F5         PUSH PSW         0025   00         NOP
  0014    21 01 00   LXI H,0001h      0026   00         NOP
  0017    2B         DCX H            0027   00         NOP
  0018    7C         MOV A,H          0028   00         NOP
  0019    B5         ORA L            0029   00         NOP
 ---------------------------- Messages ----------------------------------

 ------------------------------------------------------------------------
                     Modify register or Test port
```

รูปที่ 3.11 The Other menu

Modify reg

เมื่อเลือกหัวข้อ menu นี้จะมี pop-up window ขึ้นมาพร้อมทั้งรอให้กดตัวอักษรตัวแรกของ register ที่ต้องการจะแก้ไข ซึ่งมีวิธีการเลือกดังนี้

* ถ้าต้องการแก้ไขค่าใน Program counter ให้กดตัวอักษร "P"
* ถ้าต้องการแก้ไขค่าใน Stack pointer ให้กดตัวอักษร "S"
* ถ้าต้องการแก้ไขค่าใน Accumulator ให้กดตัวอักษร "A"
* ถ้าต้องการแก้ไขค่าใน Register B ให้กดตัวอักษร "B"
* ถ้าต้องการแก้ไขค่าใน Register C ให้กดตัวอักษร "C"
* ถ้าต้องการแก้ไขค่าใน Register D ให้กดตัวอักษร "D"
* ถ้าต้องการแก้ไขค่าใน Register E ให้กดตัวอักษร "E"
* ถ้าต้องการแก้ไขค่าใน Register H ให้กดตัวอักษร "H"
* ถ้าต้องการแก้ไขค่าใน Register L ให้กดตัวอักษร "L"
* ถ้าต้องการแก้ไขค่าใน Program Status Word ให้กดตัวอักษร "F"

<u>หมายเหตุ</u>

สำหรับค่าข้อมูลที่จะแก้ไขใน Program Status Word (PSW) นั้นจะต้องเป็นข้อมูลที่มีความหมายตาม flag ต่าง ๆ ของ CPU เบอร์ 8085 ด้วย กล่าวคือค่าข้อมูลในแต่ละบิตใน PSW มีความหมายดังนี้

บิตที่ 0 ใช้แสดงเป็น flag ตัวทด (carry flag)

บิตที่ 1 ไม่ใช้

บิตที่ 2 ใช้แสดงเป็น flag parity (parity flag)

บิตที่ 3 ไม่ใช้

บิตที่ 4 ใช้แสดงเป็น flag ตัวทดช่วย (Auxiliary flag)

บิตที่ 5 ไม่ใช้

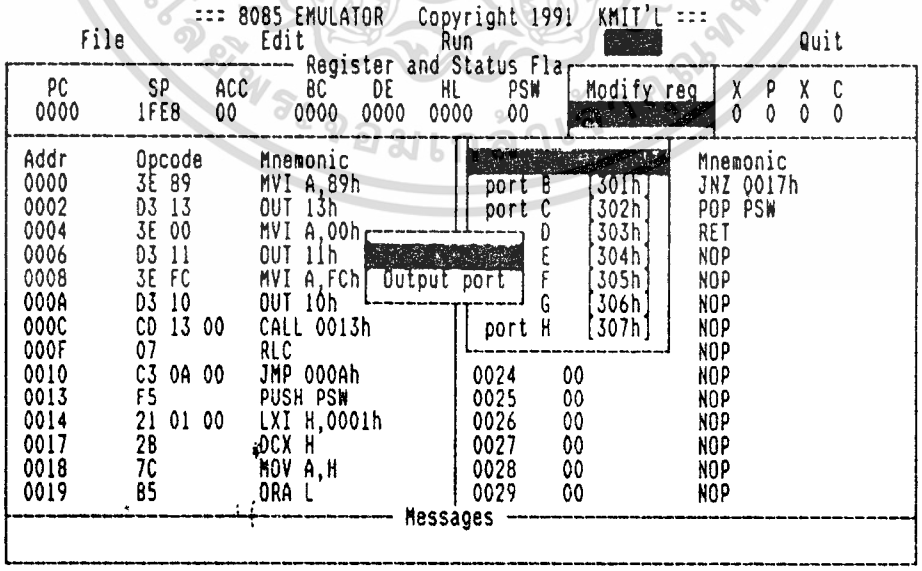
บิตที่ 6 ใช้เป็น ZERO flag

บิตที่ 7 ใช้เป็น flag แสดงเครื่องหมาย (sign flag)

ถ้าหากข้อมูลที่ป้อนเข้าไปใน PSW มีบางบิตที่ไม่มีความหมายว่าเป็น flag ใด ๆโปรแกรมจะแจ้งข้อผิดพลาดพร้อมเสียงเตือนให้ทราบแล้วให้ทำการป้อนค่าข้อมูลที่ถูกต้องลงไปใหม่

Test port

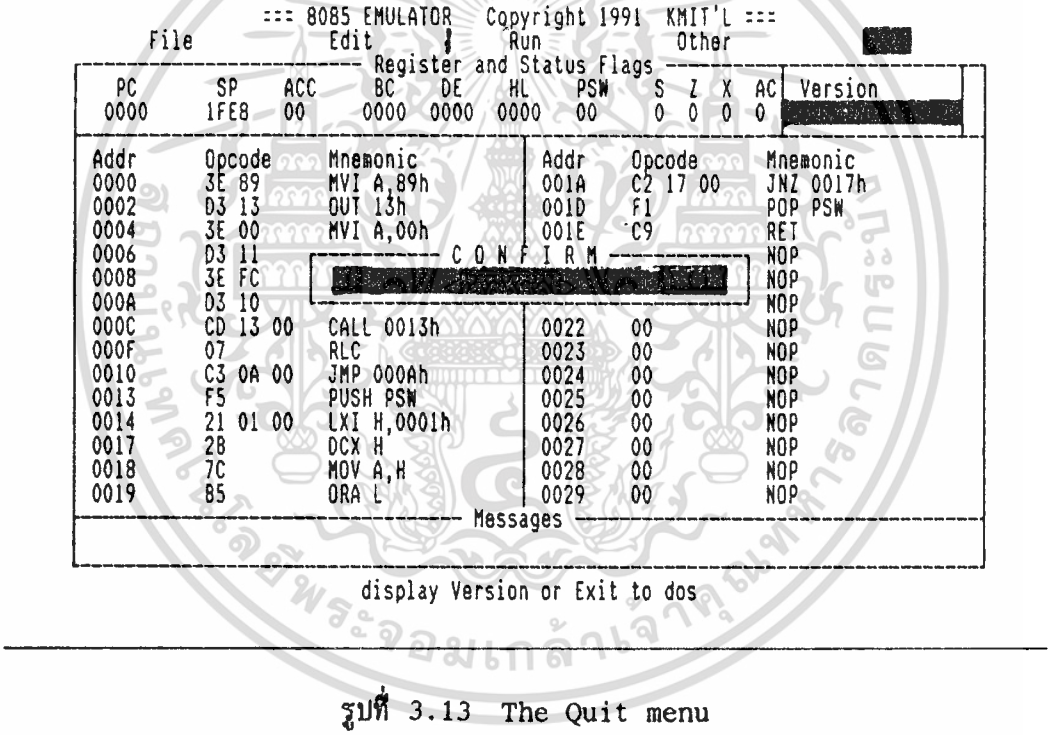
ดังที่ได้กล่าวไว้ตอนต้นแล้วว่า menu test port นี้จะเป็นการทดสอบส่งหรือรับค่าข้อมูลโดยผ่านทางเครื่อง EMULATOR 8085 โดยตรง ดังนั้นผู้ที่จะใช้หัวข้อ menu นี้จำเป็นจะต้องรู้วงจรทาง HARD WARE ของเครื่อง EMULATOR 8085 นี้ว่า port แต่ละ port นั้นต่อไว้เป็นขาสัญญาณอะไรของ CPU เบอร์ 8085 เพื่อที่จะได้ทำการทดสอบรับหรือส่งข้อมูล โดยผ่านทางขาสัญญาณต่าง ๆ เหล่านั้นได้อย่างถูกต้องซึ่งเมื่อเลือก menu นี้จะมี pull-down menu ขึ้นมาให้เลือก port ที่ต้องการจะทดสอบ ส่วนวิธีการเลือกอาจจะใช้คีย์ลูกศรเลื่อนไปยัง port ที่ต้องการ หรือกดตัวอักษรที่เป็นชื่อ port ก็ได้ หลังจากนั้นจะมี pull-down menu ขึ้นมาอีก โดยจะให้เลือกว่าจะทดสอบ input หรือ output port ถ้าหากเป็นการทดสอบ input port ข้อมูลที่รับเข้ามาได้จะแสดงในช่องหน้าต่าง massages โดยจะแสดงในเลขฐาน 16 แต่ถ้าเป็นการทดสอบ output port แล้วจะมี pop-up window ขึ้นมาให้ป้อนค่าข้อมูลที่ต้องการจะส่งออกไป โดยให้ป้อนในรูปแบบเลขฐาน 16 คือ 00-FF ซึ่งถ้าข้อมูลที่ป้อนเข้าไปผิดพลาดก็จะมีข้อความแจ้งพร้อมกับมีเสียงเตือนให้ทำการป้อนข้อมูลที่ถูกต้องเข้าไปใหม่



รูปที่ 3.12 The Other / Test port menu

Menu Quit

menu จะถูกใช้เมื่อต้องการจะเลิกการทำงานโดยออกไปที่ระบบปฏิบัติการหรืออาจจะต้องการขอดู Version ของตัวโปรแกรม EMU 8085 ซึ่งจะแสดง Version และรายชื่อผู้ร่วมงานขึ้นมาให้ดู และถ้าหากต้องการเลิกการทำงานโดยเลือกหัวข้อ Exit to dos จะมี pop-up window ขึ้นมาถามเพื่อความแน่ใจอีกครั้ง ถ้าแน่ใจว่าจะเลิกการทำงานให้ตอบ "Y" แต่ถ้าต้องการจะยกเลิกให้กด "N" หรือคีย์ใด ๆ ก็ได้



รูปที่ 3.13 The Quit menu

บทที่ 4

หลักการทำงานของโปรแกรม EMU8085

บทนำ

เนื่องจากโปรแกรม EMU8085 ที่เขียนขึ้นมานี้ใช้ภาษา C ในการเขียนทั้งหมดไม่ว่าจะเป็นระบบ Pop - up และ Pull - down menu หรือแม้กระทั่งโปรแกรมย่อยที่เป็นชุดคำสั่งของ CPU เบอร์ 8085 ก็ตาม เพราะเหตุผลที่ว่าภาษา C เป็นภาษาที่มีความอ่อนตัวและมีความสะดวกในการแก้ไขและเปลี่ยนแปลง ซึ่งภาษา C ก็เป็นภาษาหนึ่งที่มีลักษณะเป็นภาษาโครงสร้าง ดังนั้นจึงเป็นการง่ายที่จะหาข้อผิดพลาดหรือมีความคล่องตัวต่อการพัฒนาโปรแกรมต่อไปในอนาคต สำหรับตัวโปรแกรมนี้นั้นส่วนที่สำคัญของการทำงานของโปรแกรมก็คือ การ input/output port ที่เชื่อมต่อกับ IBM/PC/XT/AT เพื่อทำให้สามารถทำการควบคุมระบบที่ต้องการจะตรวจสอบได้

4.1 หลักการที่ใช้ในการเขียนโปรแกรม EMU8085

แนวความคิดเบื้องต้นสำหรับการเขียนโปรแกรมนี้ จะแบ่งเป็น 2 ส่วน โดยส่วนแรกคือส่วนที่เป็นระบบ Menu และหัวข้อย่อยในแต่ละ Menu หลัก ซึ่งก่อนที่จะเริ่มทำการเขียนโปรแกรมนี้ ก็จะต้องมีการกำหนดไว้แล้วว่าในหัวข้อ Menu หลักนั้น จะมีหัวข้ออะไรบ้าง และจากหัวข้อหลักในแต่ละ Menu นั้น จะประกอบด้วยหัวข้อ Menu ย่อยอะไรบ้าง และในแต่ละหัวข้อ Menu ย่อยนั้น อาจจะมี Menu ย่อยลงไปอีก ดังนั้นขั้นแรกในการออกแบบก็คือ จะต้องออกแบบโครงสร้างของระบบ Menu ออกมาก่อน ดังนั้นหน้าที่หลักของโปรแกรมในส่วนนี้ก็จะต้องคอยจัดการและควบคุม การเลือกหัวข้อ Menu ต่างๆ และอีกส่วนหนึ่งจะเป็นโปรแกรมหรือ Function ที่จะถูกเรียกใช้เมื่อมีการเลือกจากระบบ Menu ซึ่งจะมีอยู่มากมายหลาย Function โดยจะได้แยกอธิบายในส่วนถัดไป

4.2 ลำดับขั้นการทำงานของโปรแกรม

โปรแกรมทั้งหมดที่ได้เขียนขึ้นนี้ ได้รวบรวมไว้ในภาคผนวก ผ.2 ซึ่งทั้งหมดมี 12 ไฟล์ ด้วยกันคือ

- EMU8085.C
- FILE_MEN.C
- EDIT_MEN.C
- RUN_MENU.C
- RUN_SUB.C
- OTHER_ME.C
- QUIT_MEN.C
- INSTRUCT.C
- UNASSEM.C
- OPCODE1.C
- OPCODE2.C
- WIN_MENU.C

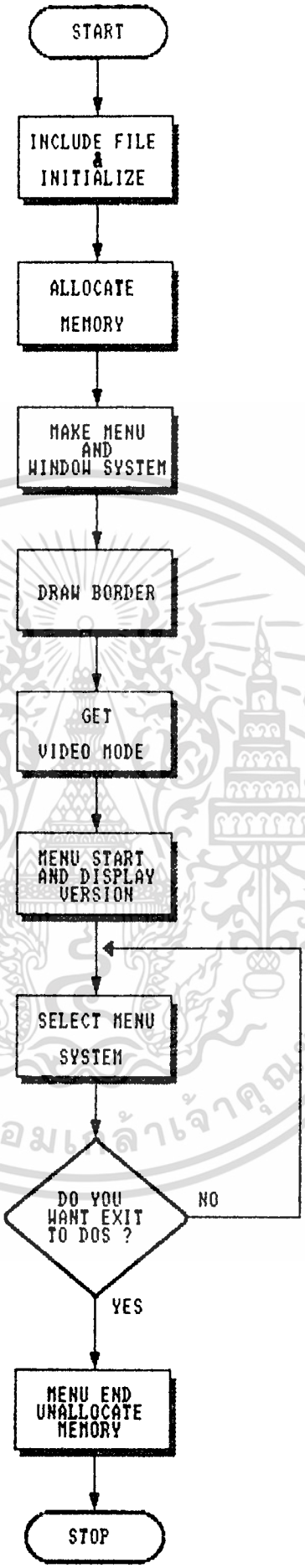
และไฟล์จำหน้า (Header file) อีก 2 ไฟล์ คือ

- DISPLAY.H
- MENU.H

ซึ่งลำดับขั้นการทำงานในแต่ละไฟล์นั้น จะขอแยกอธิบายตาม Flow Chart ทีละไฟล์ในส่วนถัดไป

4.2.1 MAIN PROGRAM (EMU8085)

โปรแกรมนี้จะเป็นโปรแกรมหลักของภาษา C ซึ่งจะเรียกใช้โปรแกรมย่อยต่าง ๆ ที่ได้จากการ include file ไว้ที่ส่วนต้นของโปรแกรม ซึ่งในส่วนนี้ได้ทำการ Initial ค่าตัวแปรต่างๆ ไว้ด้วย และยังมีการตรวจสอบด้วยว่าระบบที่กำลังจะทำการตรวจสอบนั้น มีแรงดันเป็น High ที่ขา 40 (Vcc) ของ Socket CPU - 8085 หรือไม่ ถ้าขา 40 (Vcc) จาก Socket ของ CPU - 8085 นี้ไม่มีแรงดันจ่ายมาให้ โปรแกรมก็จะไม่ทำงานต่อ แล้วจะออกไปยัง Dos โดยจะมีข้อความแสดงที่หน้าจอดังนี้



รูปที่ 4.1 แสดง Flow chart การทำงานของโปรแกรม EMU8085.C

Your system is not ready !

แต่ถ้ามี โปรแกรมก็จะทำงานต่อไป หลังจากนั้นจึงจะทำการ Allocate memory เพื่อจองพื้นที่ในหน่วยความจำภายใน IBM/PC ไว้สำหรับตัวโปรแกรมที่เป็นชุดคำสั่งของ 8085 ซึ่งขนาดที่ได้ทำการจองพื้นที่นี้ไว้คือ 8 K bytes หรือเท่ากับ 8192 bytes ซึ่งก็คิดว่าน่าจะเพียงพอสำหรับที่จะรองรับตัวโปรแกรมของระบบที่ต้องการจะตรวจสอบจากระบบภายนอกได้แล้ว เมื่อผ่านการ Allocate แล้วก็จะเข้าสู่ขั้นตอนของการสร้างระบบ Window และ Menu หลังจากนั้น ก็จะเป็นการสร้างกรอบของตัวโปรแกรม ซึ่งจะประกอบไปด้วยช่องหน้าต่าง 3 ช่องคือ ช่องหน้าต่างแสดงค่าข้อมูลใน Register และค่าสภาวะของ flags ต่าง ๆ ส่วนช่องหน้าต่างที่ 2 จะเป็นช่องหน้าต่างที่ใช้แสดงตัวโปรแกรมที่เป็นชุดคำสั่งของ CPU เบอร์ 8085 ที่ได้ทำการ load เข้ามายังหน่วยความจำของ IBM/PC ส่วนช่องหน้าต่างที่ 3 นั้นจะเป็นช่องหน้าต่างที่ใช้แสดงข้อความเพื่อบอกให้ผู้ใช้ได้ทราบและทำตาม ส่วนต่อไปก็จะเป็นการตรวจสอบ Mode ของการแสดงผลทางจอภาพ เพื่อที่จะทำการแสดงหน้าจอแรก หลังจากนั้นก็จะเข้าสู่ขั้นตอนของการเลือกหัวข้อ Menu ที่ต้องการจะทำงาน โดยจะมีอยู่ 5 หัวข้อคือ File, Edit, Run, Other และ Quit ซึ่งโครงสร้างการทำงานของแต่ละโปรแกรมนั้นจะได้กล่าวถึงในส่วนถัดไป และโปรแกรมก็จะทำงานอยู่ตลอดจนกว่าจะมีการเลือกหัวข้อ Menu Quit แล้วเลือก Exit to dos เพื่อต้องการเลิกการทำงาน ซึ่งจะมีคำถามย้ำเพื่อความแน่ใจอีกครั้ง ถ้าหากยังไม่ต้องการออกจากโปรแกรมก็ให้ตอบ 'n' แต่ถ้าแน่ใจว่าจะเลิกการทำงานจากโปรแกรมแล้วให้ตอบ 'y' หลังจากนั้นโปรแกรมก็จะทำการ Unallocate Memory เพื่อเป็นการคืนหน่วยความจำที่ได้จองไว้คืนให้กับระบบปฏิบัติการ

4.2.2 โปรแกรมที่ใช้ควบคุมระบบการเลือก Menu (WIN_MENU.C)

หน้าที่หลักของโปรแกรมในส่วนนี้คือ จะทำการควบคุมการใช้งานในการเลือกหัวข้อ Menu ต่าง ๆ ซึ่งตัวโปรแกรมทั้งหมดนี้จะอยู่ใน File ชื่อ WIN_MENU.C โดยได้รวมเอา Function ต่าง ๆ ที่เกี่ยวกับ Window เข้าไว้ด้วย ในส่วนต้นของโปรแกรมนั้นจะมีการกำหนด Array ของ char เพื่อเก็บไว้เป็นหัวข้อ Menu ย่อยต่าง ๆ ซึ่งเป็นลักษณะของ Box Menu ส่วนหัวข้อหลักที่เป็น Main Menu นั้นจะกำหนดเป็น Array ชื่อ Menu0 โดยเป็นชนิด MENU ซึ่งได้กำหนดไว้ใน Header file ชื่อ MENU.H ส่วนต่อไปจะเป็นหน้าที่การทำงานของ Function ต่าง ๆ ใน file WIN_MENU.C

Menu Function

- menu_start(MENU s[],int fore,int back)

  Function นี้จะทำการสร้างและแสดงหัวข้อ Main Menu ออกมาในแบบ slide bar menu

- fist_active(MENU s[],int fore,int back)

  Funciton นี้จะทำการตรวจสอบคีย์ว่ามีการกดคีย์ F10 หรือไม่ ถ้ามีก็จะทำให้หัวข้อ Menu Active เป็น Reverse Video

- menu_choice(MENU s[],int fore,int back)

  Function นี้จะเป็นหัวใจสำคัญของระบบ menu ทั้งหมด กล่าวคือ จะทำหน้าที่ในการตรวจสอบการกดคีย์ที่ใช้ควบคุมการเลือกหัวข้อ menu ต่าง ๆ เช่น คีย์ลูกศรซ้าย - ขวา หรือ คีย์ที่เป็นตัวอักษรตัวแรกของแต่ละหัวข้อ menu แล้วให้ active ตามหัวข้อ menu ที่เลื่อนคีย์ลูกศรไป หลังจากนั้นให้กดคีย์ Enter หรือกดคีย์ลูกศรลง (down arrow) เพื่อเป็นการเลือกหัวข้อ menu นั้น ๆ

- select_menu()

  Function นี้จะเป็นการเลือกไปทำงานตามหัวข้อเลือกต่าง ๆ

- border()

  Function นี้จะเป็นการสร้างกรอบหน้าต่างที่หน้าจอ โดยได้แบ่งเป็น 3 ช่องหน้าต่างคือช่องแรกจะเป็นส่วนแสดงค่าของข้อมูลใน Register และค่าสภาวะใน flags ต่าง ๆ ส่วนช่องหน้าต่างตรงกลางจะเป็นช่องหน้าต่างที่ใช้แสดงตัวโปรแกรมของ CPU - 8085 ที่ได้ load เข้ามายังหน่วยความจำของ IBM/PC ส่วนช่องหน้าต่างที่ 3 นั้นจะใช้เป็นช่องหน้าต่างที่ใช้แสดงข้อความต่าง ๆ แจ้งให้ผู้ใช้ได้ทราบและทำตามข้อความนั้น ๆ

Screen Function

Funciton ในส่วนนี้จะเป็นการใช้งานเกี่ยวกับการแสดงผลทางจอภาพต่าง ๆ เช่น cls(), active(), active_off(), c_scroll(), colorstr(), colorchr(), getvpage(), getvmode(), video_mode(), write_string(), write_char(), save_video(), restore_video(), display_menu(), draw_border(),

Cursor Function

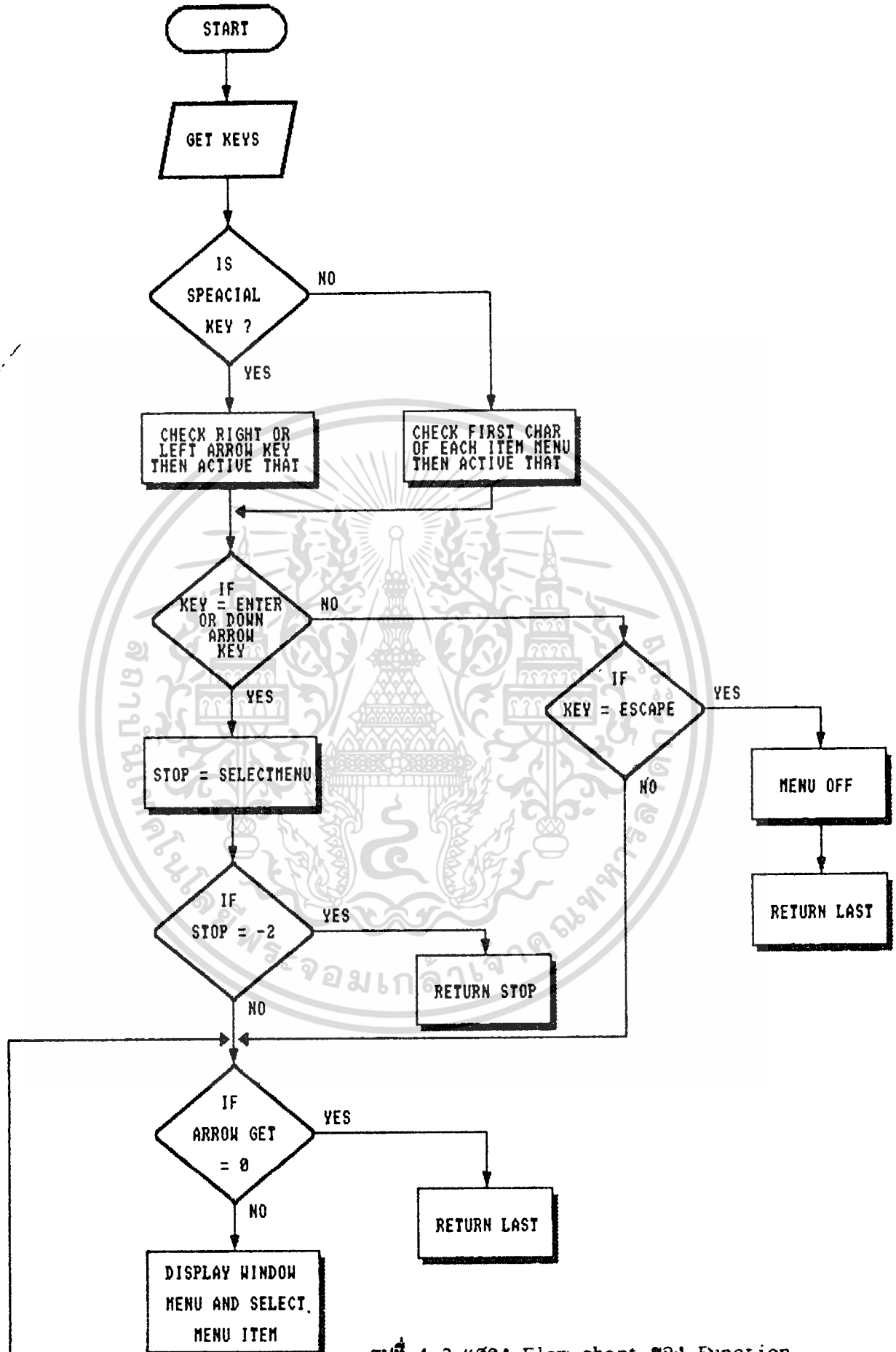
Function ในส่วนนี้จะเป็นการทางานเกี่ยวกับการจัดการกับ cursor ต่างๆ เช่น goto_xy(), cursor(), hide_cursor(), getvcols(), find_cursor()

Key Function

เป็น function ที่ทำหน้าที่เกี่ยวกับคีย์ควบคุมต่าง ๆ เช่น ตรวจสอบการกดคีย์ หรือตรวจสอบคีย์ที่ใช้ในการควบคุมการเลือกในแต่ละหัวข้อ menu ซึ่งมี function ต่าง ๆ ดังนี้คือ get_key(), get_resp(), get_resp_sub(), get_resp_sub1()

Window Function

Function ที่อยู่ในส่วนนี้ จะเป็น Function ที่ทำหน้าที่เกี่ยวกับการใช้งาน window ต่าง ๆ เช่น การเรียก window มาใช้งาน หรือการสร้าง window หรืออาจจะเป็นการเขียนตัวอักษรหรือข้อความต่าง ๆ ภายใน window ซึ่งมี function ต่าง ๆ ดังนี้คือ window_(), make_window(), deactive_win(), display_header(), draw_border1(), window_puts(), window_putchar(), window_xy(), window_gets(), window_getchar(), window_cls(), window_cleol(), window_upline(), window_downline(), window_bksp(), save_video_win(), restore_video_win(), get_special()



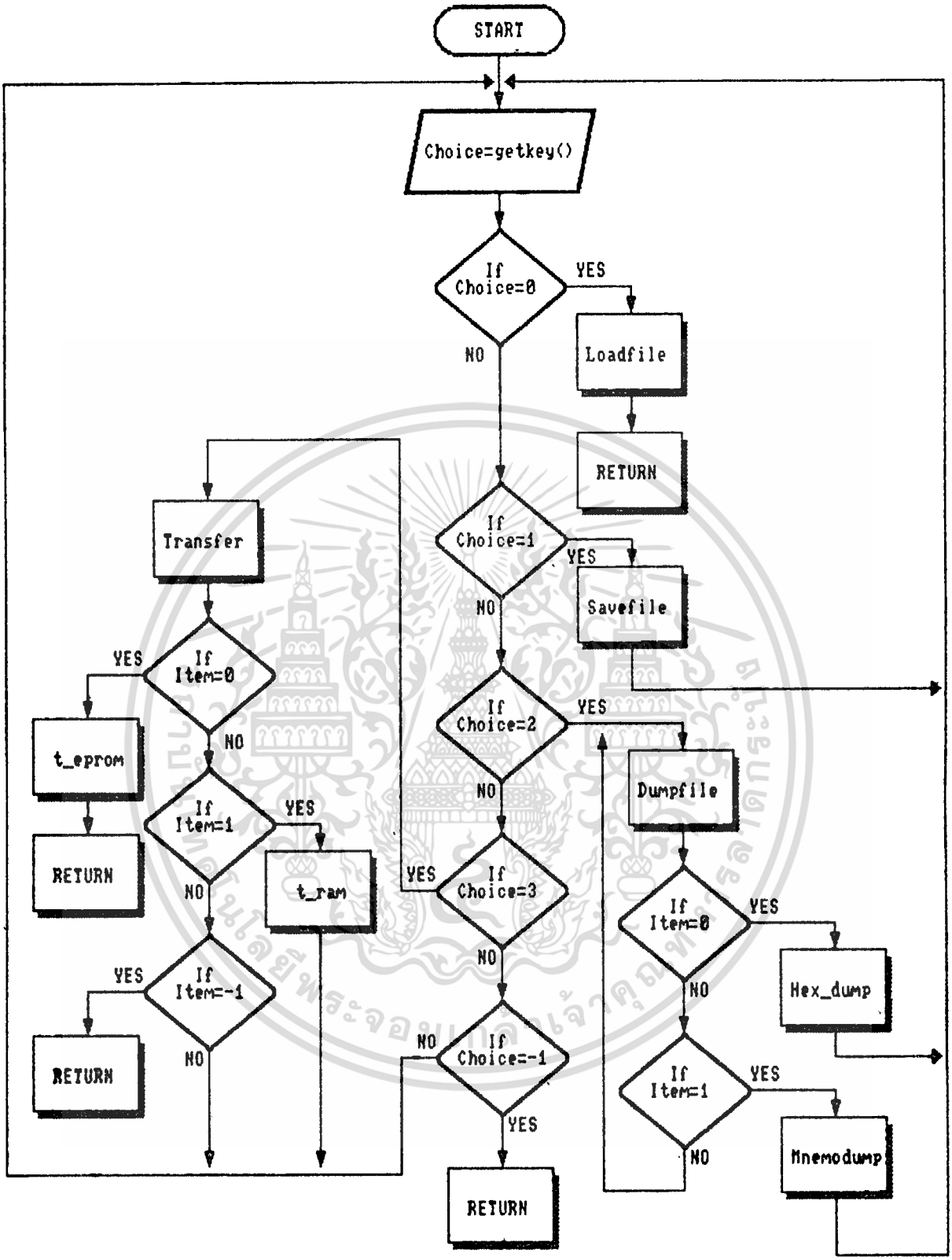
รูปที่ 4.2 แสดง Flow chart ของ Function menu_choice()

4.2.3 การทำงานของ Function menu_choice()

จากรูปที่ 4.2 จะเป็น Flow chart ของ Function menu_choice() ซึ่งเป็นหัวใจของการเลือกทำงานตามหัวข้อ menu ต่าง ๆ ซึ่งมีขั้นตอนการทำงานดังต่อไปนี้ เริ่มต้นด้วยการตรวจสอบการกดคีย์ ว่ามีการกดคีย์ใดคีย์หนึ่งหรือไม่ ถ้ามีก็จะมี Function ที่ทำหน้าที่ตรวจสอบว่าคีย์ที่กดนั้น เป็นคีย์พิเศษหรือไม่ ถ้าใช่ก็จะตรวจสอบว่าเป็นคีย์ลูกศรซ้ายหรือขวา แล้ว Active ไปตามหัวข้อ menu หลัก ตามทิศทางของคีย์ลูกศรนั้น ๆ แต่ถ้าคีย์ที่กดนั้นไม่ใช่คีย์พิเศษ ก็ให้ตรวจสอบว่า ตัวอักษรที่กดนั้นตรงกับตัวอักษรตัวแรกของแต่ละหัวข้อ main menu หรือไม่ ถ้าใช่ก็ให้ทำการ Active ตามหัวข้อ menu นั้น ๆ หลังจากนั้นก็จะทำการตรวจสอบว่าคีย์ที่กดต่อไป เป็นคีย์ ENTER หรือ คีย์ลูกศรลงหรือไม่ ถ้าใช่ก็จะมี pull down menu ของหัวข้อ menu หลัก นั้น ๆ ขึ้นมาให้ทำการเลือก แต่ถ้าไม่ใช่คีย์ทั้งสอง ก็จะตรวจสอบว่าเป็นคีย์ ESCAPE หรือไม่ ถ้าใช่ก็ให้ deactive หัวข้อ main menu นั้น แล้ว RETURN ค่า LAST กลับไป ในขณะที่กำลังทำงานในระบบ menu อยู่นั้น ถ้าหากเลือกหัวข้อ menu Quit และ Exit to dos แล้วจะมีการ Set ค่าตัวแปร stop ให้เท่ากับ -2 ซึ่งถ้าหากค่าตัวแปรนี้มีค่าเท่ากับ -2 แล้ว Function นี้ จะ RETURN ค่า -2 นี้กลับออกไป

4.2.4 การทำงานของโปรแกรม FILE_MEN.C

จากรูปที่ 4.3 ซึ่งแสดง Flow chart การทำงานของโปรแกรม FILE_MEN หลังจากที่ Function ใน File นี้ถูกเรียกใช้งานแล้วจะปรากฏ Pull down menu ขึ้นมาให้เลือกหัวข้อการทำงานอีก 4 หัวข้อ โดยที่โปรแกรมจะคอยตรวจสอบการกดคีย์ จากนั้นจะ RETURN ค่าของหัวข้อเลือกให้กับตัวแปร Choice หลังจากนั้นก็จะเข้าสู่การตรวจสอบโดยอาศัย คำสั่ง Switch - case เพื่อทำการตรวจสอบว่า หัวข้อที่เลือกนั้นตรงกับ case ไหน แล้วจึงไปเรียก Function นั้น ๆ มาทำงาน ซึ่งจาก Flow chart นั้น ถ้าค่า Choice เป็น 0 แสดงว่าเป็นการเลือกหัวข้อ menu Loadfile ซึ่ง Function นี้ก็จะทำหน้าที่ในการ Load file จากแผ่น Disk เข้ามาไว้ยังหน่วยความจำของ IBM/PC หลังจากนั้นก็จะ RETURN ออกจาก Function file นี้ แต่ถ้าค่า Choice เท่ากับ 1 ก็จะเป็นการเรียก Function Save file มาทำงาน โดยเมื่อทำการ Save เสร็จแล้วก็จะกลับไปที่ขั้นตอนการเลือกหัวข้อ menu ย่อย ใน menu File และถ้าค่า Choice

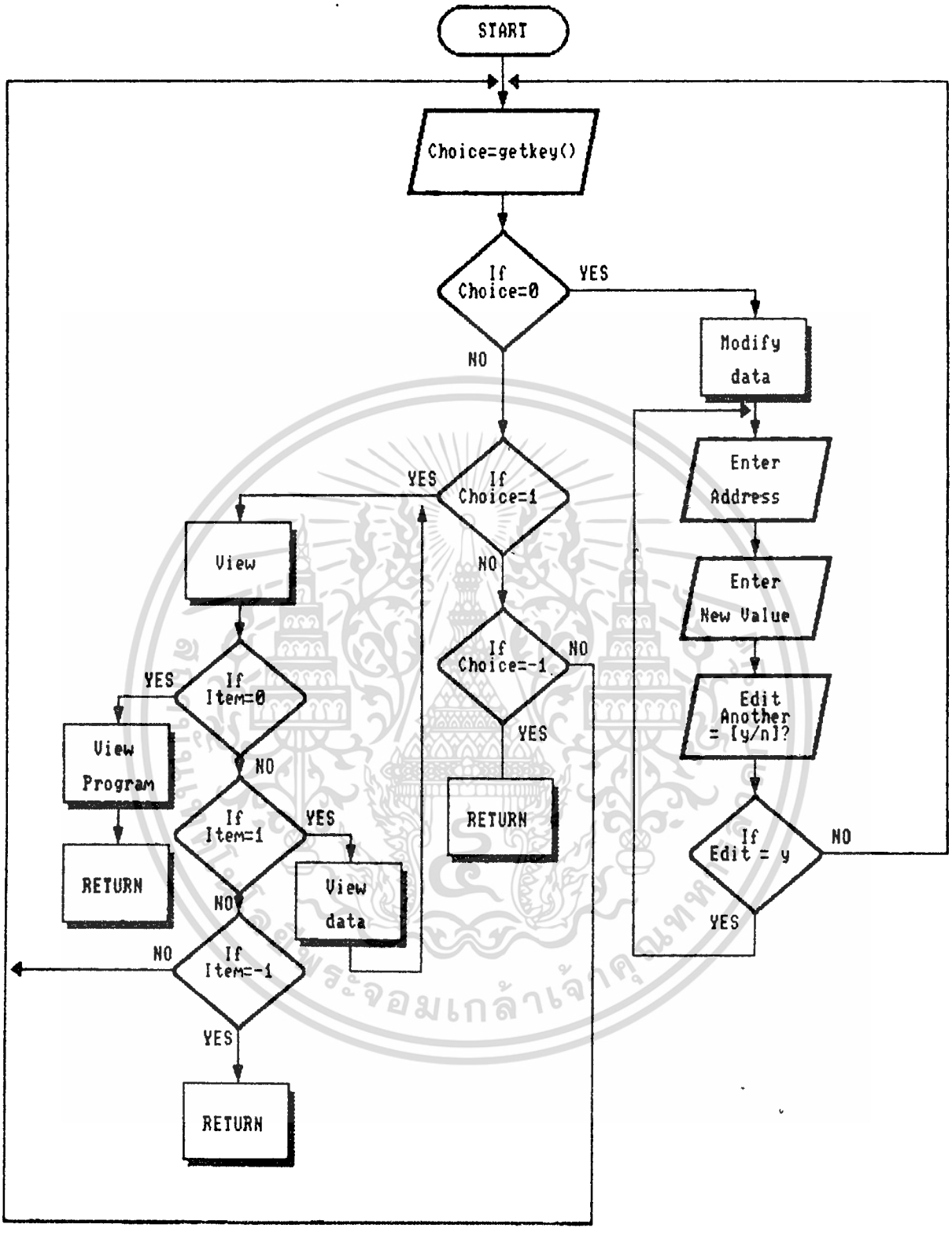


รูปที่ 4.3 แสดง Flow chart ของโปรแกรม FILE_MEN.C

เป็น 2 ก็จะเข้าไปสู่ menu การ Dump file ซึ่ง Function ใน menu นี้จะทำหน้าที่ในการส่งไฟล์ ที่ได้ถูก Load มาไว้ยังหน่วยความจำของ IBM/PC แล้ว นำไปพิมพ์ออกทางเครื่องพิมพ์ ซึ่งสามารถที่จะเลือกรูปแบบการพิมพ์ได้อีก 2 แบบ โดยโปรแกรมจะตรวจสอบค่าที่อยู่ในตัวแปร Item ซึ่งได้จากการ getkey มา ถ้า Item = 0 ก็จะเป็นการส่งพิมพ์ไฟล์ในแบบ Hexadecimal แต่ถ้าหากว่าค่า Item = 1 ก็จะพิมพ์ไฟล์ออกในแบบ Mnemonic คือจะมีทั้งค่าของ Address ,Opcode และรหัส Mnemonic ส่วนหัวข้อสุดท้ายคือ ถ้าค่าในตัวแปร Choice = 3 ก็จะเป็นการเรียก Function Transfer มาทำงาน ซึ่งใน Function นี้ก็จะมีให้เลือกอีกว่า จะเป็นการ Load ข้อมูลจาก EPROM หรือจะเป็นการนำข้อมูลในหน่วยความของ IBM/PC ไปเก็บไว้ใน RAM ภายในเครื่อง 8085 EMULATOR ซึ่งจะทำการเลือกด้วย คำสั่ง Switch - case เช่นเดียวกัน และถ้าหากว่าตัวแปร Choice มีค่าเป็น -1 ซึ่งก็แสดงว่ามีการกดคีย์ ESCAPE Function file นี้ก็จะถูก Return กลับไปยังโปรแกรมหลักที่เป็นฝ่ายเรียกมา

4.2.5 การทำงานของโปรแกรม EDIT_MEN.C

ลำดับขั้นการทำงานใน Function นี้นั้นจะอธิบายได้ตาม Flow chart ในรูปที่ 4.4 ซึ่งหลังจากมีการเรียก Function นี้ไปใช้งานแล้ว ก็จะมีการคอยตรวจสอบว่ามีการเลือกทำงานในหัวข้อ menu ใด ซึ่งใน Function นี้จะมีอยู่ 2 หัวข้อคือ Modify และ View โดยค่าของหัวข้อที่เลือกนั้นจะถูก Return ค่าไปไว้ในตัวแปร Choice แล้วจึงทำการ Check ว่า ค่าที่อยู่ในตัวแปร Choice นี้มีค่าเป็นอะไรโดยใช้ Switch-case ซึ่งถ้า Choice มีค่าเป็น 0 ก็จะเป็นการเลือก Function Modify มาทำงานโดยจะมีการถามค่า Address ที่ต้องการจะแก้ไข จากนั้นจะแสดงค่าข้อมูลเดิมออกมา พร้อมทั้งให้ทำการป้อนค่าใหม่เข้าไป หลังจากนั้นก็จะมีคำถามอีกว่าต้องการจะแก้ไขค่าข้อมูลใน Address อื่นอีกหรือไม่ โดยให้ตอบ 'y' หรือ 'n' แต่ถ้าตัวแปร Choice = 1 ก็จะเป็นการเลือก Function View ซึ่งใน Function ก็จะมี menu ย่อยให้เลือกอีก 2 หัวข้อคือ Program หรือ Data ในส่วนของการ View Program นั้นจะทำหน้าที่ในการแสดงตัวโปรแกรมที่แสดงอยู่ที่หน้าจอใหม่ โดยจะต้องมีการป้อนค่า Address เริ่มต้นด้วยทุกครั้ง ส่วนในการ View Data นั้นจะเป็นการขอดูค่าของข้อมูลใน Address ช่วงใดช่วงหนึ่ง และเช่นเดียวกับ Function อื่น ๆ คือถ้าหากว่าในขณะที่โปรแกรมกำลังรอให้เลือกหัวข้อใดหัวข้อหนึ่งอยู่นั้นแล้วมีการกดคีย์ ESCAPE โปรแกรมก็จะถูก Return กลับไป



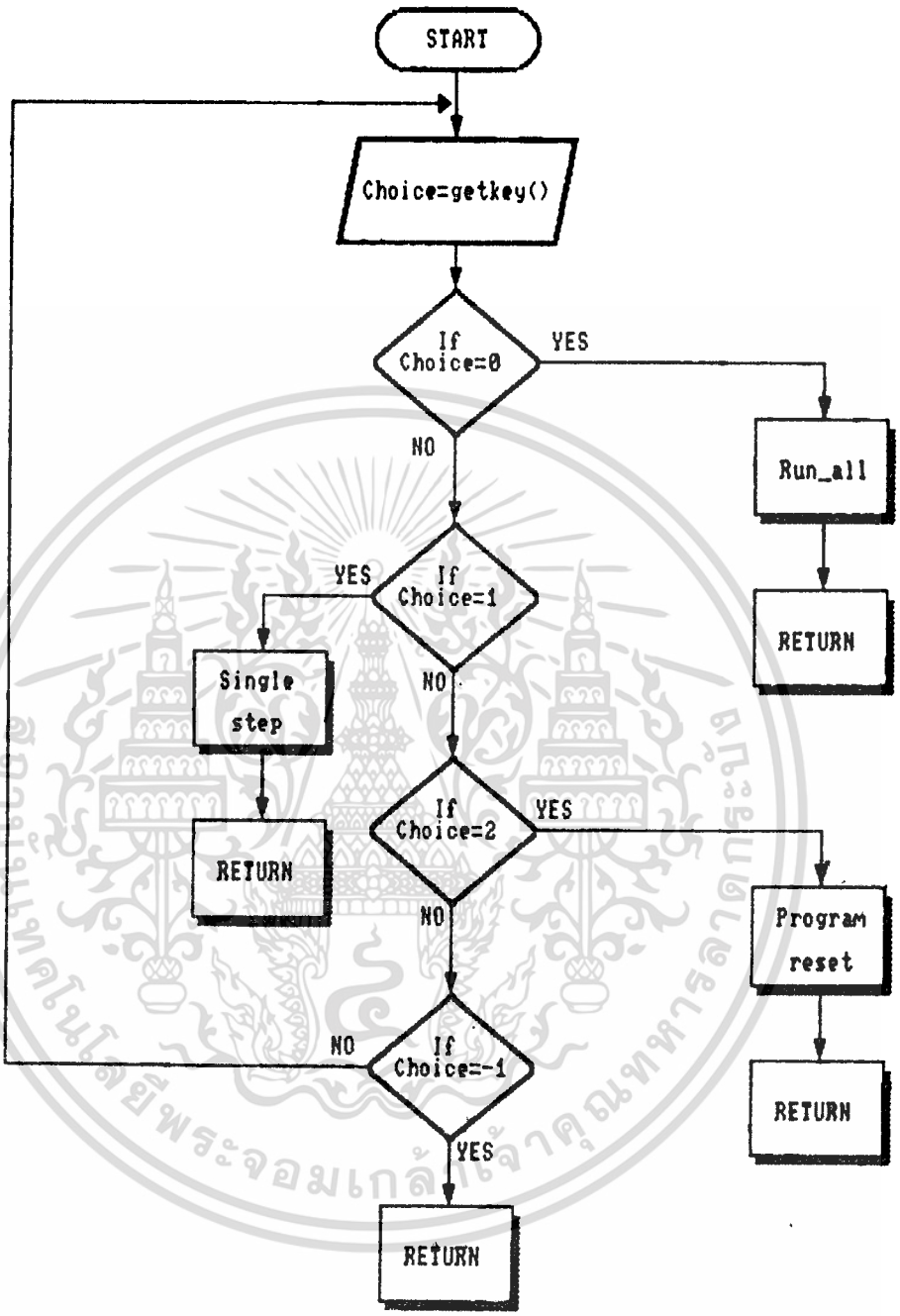
รูปที่ 4.4 แสดง Flow chart ของโปรแกรม EDIT_MEN.C

4.2.6 การทำงานของโปรแกรม RUN_MENU.C

หลังจากที่ Function นี้ถูกเลือกไปใช้งานแล้ว ก็จะมีลักษณะเหมือนกับ Function อื่น ๆ คือ จะมีหัวข้อ menu ย่อยให้เลือกอีก โดยใน Function นี้จะมีให้เลือกอีก 3 menu ย่อยคือ Run all, Single step และ Program reset สำหรับวิธีการเลือกก็จะเหมือนกับ Function ที่แล้ว ๆ มา กล่าวคือ ถ้ามีการเลือกหัวข้อใดหัวข้อหนึ่งก็จะมีการ Return ค่าของหัวข้อ menu นั้น ๆ ไปยังตัวแปร Choice ซึ่งก็จะถูกนำไปเข้า Switch - case เพื่อเป็นตัวเลือก Function ที่จะถูกทำงานต่อไป

เนื่องจากว่า Function ในโปรแกรม RUN_MENU.C นี้เป็นหัวใจสำคัญในการทำงานของเครื่องอีมูเลเตอร์นี้ ดังนั้นจึงขอกล่าวถึง Function ที่ใช้ในการ Run ตัวโปรแกรมของ CPU เบอร์ 8085 ดังนี้คือ

- run_all() ซึ่งหลักการในการที่จะสั่งให้โปรแกรมทำงานตามชุดคำสั่งก็คือเริ่มจากการดึงค่าข้อมูลที่อยู่ใน Address เริ่มต้น ซึ่งจะต้องมีการกำหนดทุกครั้งที่จะมีการสั่งให้โปรแกรมในส่วนนี้ทำงาน เมื่อได้ค่าข้อมูลมาแล้ว ก็นำมาเข้า Switch-case เพื่อตรวจสอบว่าค่าของข้อมูลนั้น เป็นคำสั่งอะไร และจะต้องมี Operand ตามมาอีกหรือไม่ ถ้าเป็นคำสั่งที่ต้องตามด้วย Operand ก็จะต้องทำการเพิ่มค่า Address แล้วนำข้อมูลเข้ามาประมวลผลต่อไป ซึ่งคำสั่งทุกคำสั่งของ CPU - 8085 นี้จะถูกรวบรวมเป็น Function ไว้ใน Switch - case ทั้งหมด และ File ที่เก็บรวบรวม Function ต่าง ๆ เหล่านี้ถูกรวบรวมไว้ใน File ชื่อ OPCODE1.C และ OPCODE2.C และถ้าคำสั่งใดมีผลต่อ Flag ก็จะมีการแสดงค่าสภาวะของ Flag นั้น ๆ หลังจากปฏิบัติคำสั่งนั้นแล้วด้วย ซึ่งในระหว่างที่โปรแกรมกำลังทำงานอยู่นั้น หากต้องการที่จะหยุดการทำงานก่อนที่จะถึงค่า Address ที่ตั้งไว้ อาจทำได้ 2 วิธีดังนี้คือ วิธีแรกนั้นให้กดคีย์ ESCAPE โปรแกรมจะถูก Break ค้างไว้ที่ Address ที่ทำการ Break นั้น โดยจะแสดงค่าสภาวะของ Register ต่าง ๆ หลังจากที่ได้ทำการ Break ค้างเอาไว้ ส่วนอีกวิธีหนึ่งที่จะทำการหยุดโปรแกรมได้คือ การกด Switch ของระบบภายนอกที่กำลังตรวจสอบอยู่นั้น แต่การกด Switch นี้นอกจากจะเป็นการหยุดการทำงานของ โปรแกรมได้แล้วยังจะทำการ Clear ค่าที่อยู่ใน Register ต่าง ๆ ให้อยู่ในสภาวะเริ่มต้นอีกด้วย เช่น Set ให้ค่าใน Program counter มีค่าเป็น 0000H ส่วน Stack pointer มีค่าเป็น 1FE8H และ Clear ใน Register ACC, BC, DE, HL และค่า Flags ต่าง ๆ ให้เป็น 0 นอกจากการที่กด Switch จากระบบภายนอกจะทำให้เกิดขบวนการดังกล่าวแล้ว ยังจะต้องมีการส่งสัญญาน



รูปที่ 4.5 แสดง Flow chart ของโปรแกรม RUN_MENU.C

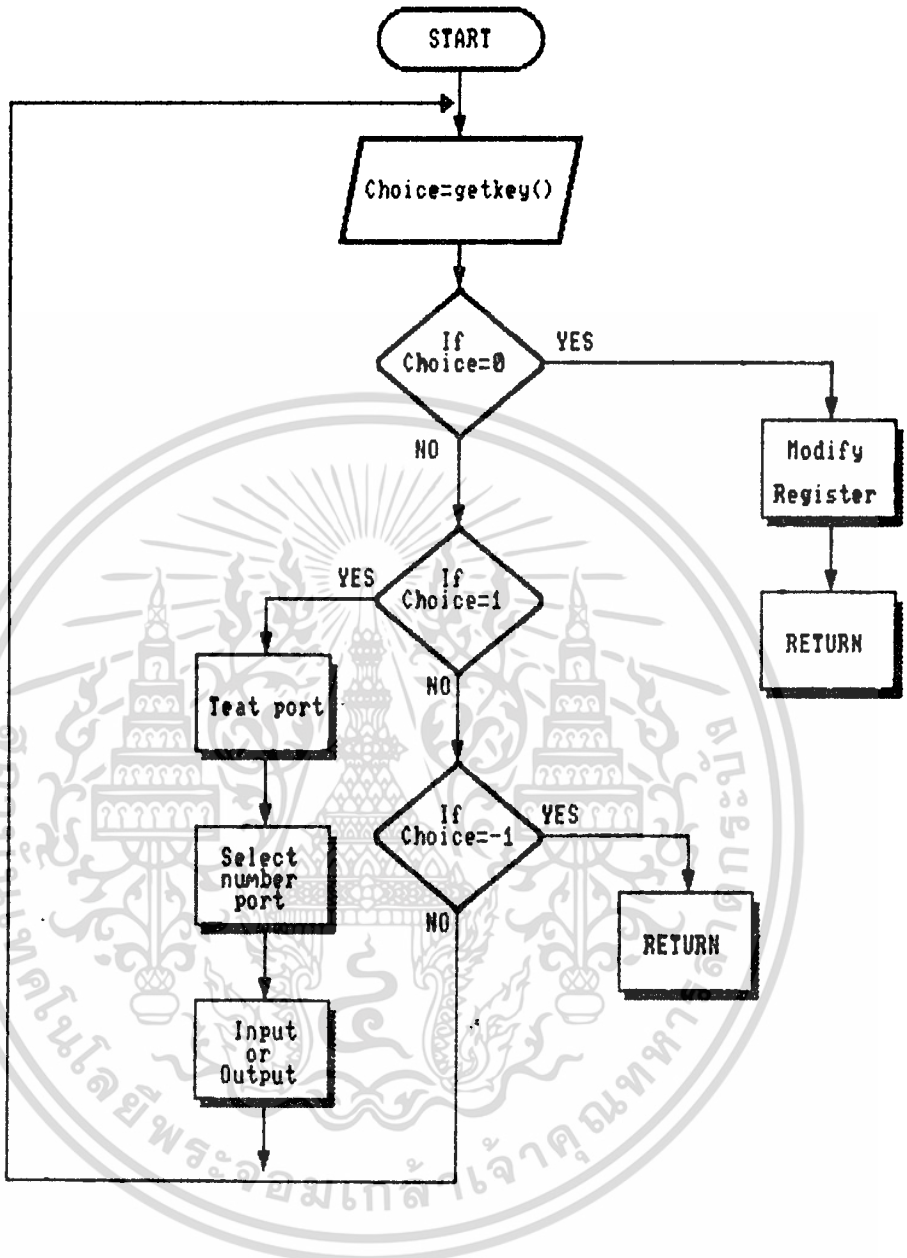
การ Reset ซึ่ง Active "Low" ออกไปยังระบบภายนอกอีกด้วย

- Step() Function นี้จะมีหลักการทำงานที่เหมือนกับ Function ที่แล้ว แต่แตกต่างกันตรงที่ว่า ใน Function นี้นั้นจะเป็นการทำงานทีละคำสั่ง โดยหลังจากที่ได้ Run คำสั่งหนึ่งไปแล้ว จะมีการแสดงค่า Address ถัดไปที่จะ Run ในช่องหน้าต่าง Messages และจะรอให้ผู้ใช้กดคีย์ F8 เพื่อทำงานใน Address ถัดไป ซึ่งหน้าที่ของ Function คีย์ต่าง ๆ จะมีการอธิบายที่บรรทัดล่างสุดของจอภาพและได้กล่าวไปแล้วในบทที่ 3 ในส่วนของการใช้งานโปรแกรม Run Single step ว่ามี Function คีย์อะไรบ้างและแต่ละอย่างทำหน้าที่อะไร

- Process_rst() ถ้าหากมีการเรียก Function นี้ไปใช้งานก็จะเป็นการ Clear ค่าที่อยู่ใน Register ต่าง ๆ ให้อยู่ในสภาวะเริ่มต้น เช่น Set ให้ค่าใน Program counter มีค่าเป็น 0000H ส่วน Stack pointer มีค่าเป็น 1FE8H และ Clear ใน Register ACC, BC, DE, HL และค่า Flags ต่าง ๆ ให้เป็น 0 นอกจากนั้น ยังจะต้องมีการส่งสัญญาน การ Reset ซึ่ง Active "Low" ออกไปที่ขา 3 ของ Socket ที่ทำหน้าที่แทนขาสัญญานต่าง ๆ ของ CPU - 8085 เพื่อไปทำการ Resetระบบภายนอกอีกด้วย

4.2.7 การทำงานของโปรแกรม OTHER_ME.C

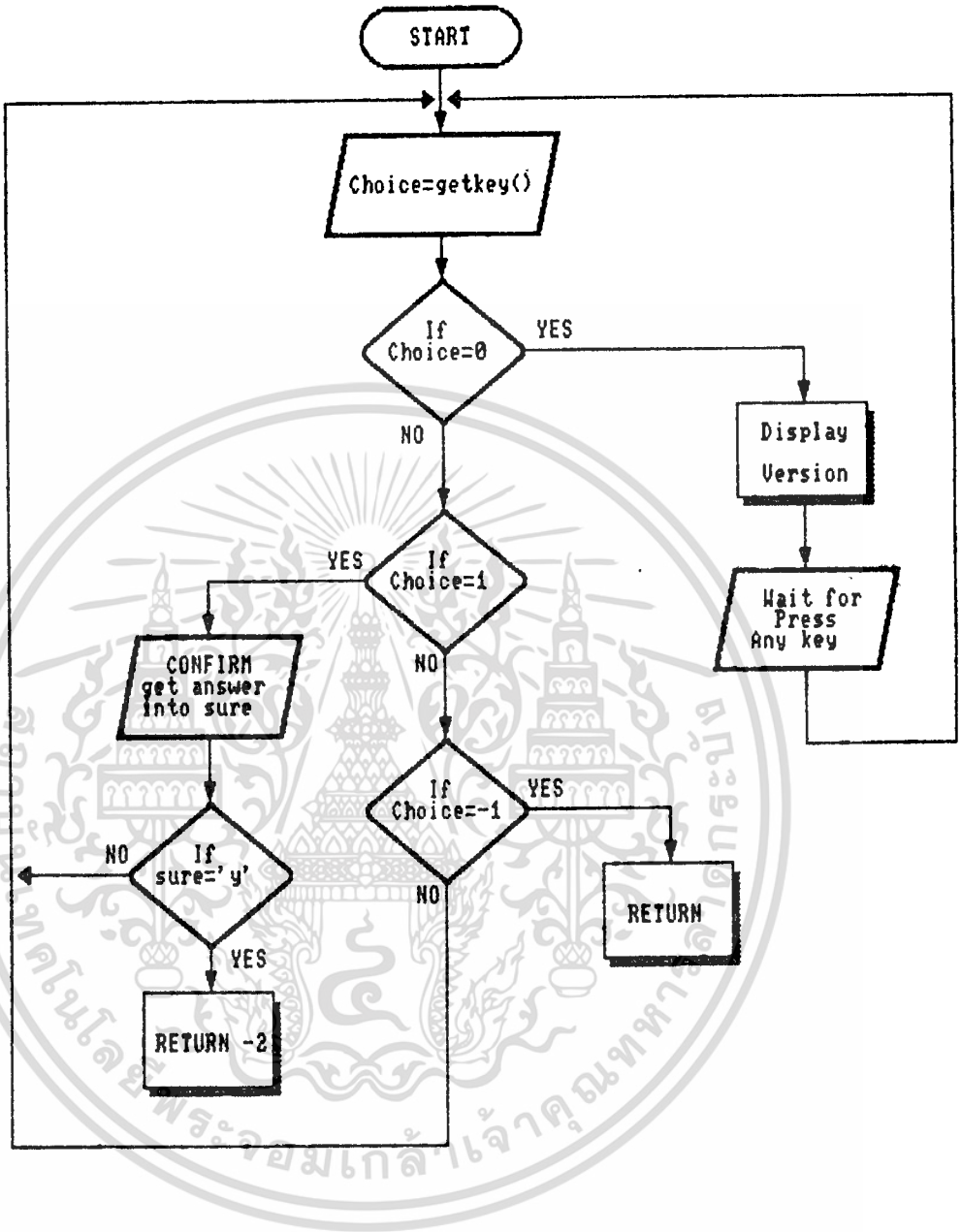
หลักการทำงานในโปรแกรมนี้ก็จะเหมือนกับหัวข้ออื่น ๆ คือเริ่มต้นโปรแกรมจะรอการเลือกหัวข้อที่จะทำงาน ถ้าหากมีการเลือกเกิดขึ้นก็จะมีการ ส่งค่าหัวข้อที่เลือกนั้นไปเก็บไว้ในตัวแปร Choice หลังจากนั้นก็จะไปเข้า Switch case เพื่อ Check ว่าเป็นเลือกหัวข้ออะไร ซึ่งถ้าค่า Choice = 0 ก็จะเป็นการเรียก Function Modify reg มาทำงาน เพื่อทำการเปลี่ยนแปลงค่าข้อมูลที่อยู่ใน Register ต่าง ๆ หลังการนั้นก็จะ Return กลับออกไป แต่ถ้าค่าในตัวแปร Choice = 1 ก็จะเป็นการเรียก Function ของการ Test port ของเครื่อง 8085 EMULATOR มาทำงาน โดยจะมี menu ย่อยให้เลือกเบอร์ port ที่ต้องการจะทดสอบ หลังจากเลือกแล้วก็จะมี menu ให้เลือกอีกว่าจะทำการ Test port ที่เป็น Output หรือ Input ซึ่งจะไปเรียก Function ในการ Test port ออกมาทำงาน และในระหว่างที่รอการเลือกหัวข้อ menu อยู่นั้น มีการกดคีย์ ESCAPE ซึ่งก็จะทำให้ค่าในตัวแปร Choice มีค่าเป็น -1 Function ในโปรแกรม OTHER_ME.C นี้ก็จะถูก Return กลับไปยังโปรแกรมหลักที่เป็นฝ่ายเรียกมา



รูปที่ 4.6 แสดง Flow chart ของโปรแกรม OTHER_ME.C

4.2.8 การทำงานของโปรแกรม QUIT_MEN.C

สำหรับการเลือกหัวข้อเลือกทำงานใน Function นี้ก็จะเหมือนกับทุก ๆ Function ที่แล้วมา ซึ่งจะมีหัวข้อให้ทำการเลือก 2 หัวข้อ คือ การขอดู Version ของโปรแกรม (disp_version()) และการเลิกการทำงานโดยเลือกหัวข้อ Exit to dos ซึ่งก็จะมีการคำถามย้ำเพื่อความแน่ใจอีกครั้ง โดยให้ตอบ 'y' หรือ 'n' ถ้าแน่ใจว่าจะเลิกการทำงานแล้วก็ให้ตอบ 'y' Function ในโปรแกรมนี้ก็จะทำการ Return ค่า -2 กลับออกไปให้ยัง Main Program เพื่อไปเข้าเงื่อนไขในการ Check เพื่อเลิกการทำงานของโปรแกรม



รูปที่ 4.7 แสดง Flow chart ของโปรแกรม QUIT_MEN.C

บทที่ 5

วิธีการนำเครื่อง 8085 EMULATOR ไปใช้งาน

5.1 การเชื่อมต่อทางฮาร์ดแวร์

ลักษณะการเชื่อมต่อทางฮาร์ดแวร์ของเครื่อง 8085 EMULATOR ในการที่จะนำไปใช้งานนี้ จะต้องนำสายสัญญาณที่ต่ออยู่กับ Card ที่จะต้องนำไปเสียบเข้ากับ Slot ของ IBM/PC/XT/AT นั้นนำไปเสียบเข้ากับ Slot ที่ยังว่างอยู่ภายในเครื่องไมโครคอม-พิวเตอร์แล้วนำปลายอีกด้านหนึ่งไปต่อเข้ากับ CONNECTOR ตัวบนของเครื่อง EMULATOR จากนั้นนำสายสัญญาณอีกเส้นซึ่งจะเป็นขนาด 40 ขา ต่อเข้ากับ CONNECTOR ตัวล่างที่อยู่ด้านหน้าของเครื่อง 8085 EMULATOR ส่วนปลายอีกด้านนั้นจะมีลักษณะเป็นขาแบบตัว IC ขนาด 40 ขา ซึ่งจะต้องนำไปเสียบเข้ากับ Socket ของ CPU เบอร์ 8085 ของระบบที่จะทำการตรวจซ่อม โดยที่ต้องถอด CPU 8085 ตัวจริงออกจากระบบนั้น ๆ ก่อน

5.2 การเรียกโปรแกรม EMU8085 มาใช้งานเพื่อควบคุมเครื่อง EMULATOR

ก่อนที่จะเรียกตัวโปรแกรมนี้มาใช้งานนั้นควรจะต้องเปิด Switch ที่ด้านหน้าของเครื่องอีมูเลเตอร์และจ่ายไฟให้กับระบบที่ต้องการจะทดสอบก่อน เพราะเมื่อเริ่มต้นเรียกโปรแกรมมาใช้งานจาก Dos นั้น หลังจากพิมพ์ชื่อโปรแกรม (C:\>emu8085) แล้วนั้นเริ่มต้นก่อนที่โปรแกรมจะเข้าไปทำงานตามระบบ menu นั้น จะมีการตรวจสอบว่าที่ขา 40 ของ Socket CPU - 8085 มีแรงดันเป็น "high" จ่ายมาให้หรือไม่ ถ้ามีก็จะโปร-แกรมก็จะทำงานต่อไป โดยจะแสดงหน้าจอแรกออกมา แต่ถ้าหากว่าในขณะที่ทำการเรียกโปรแกรมมาใช้งานนั้น ไม่มีแรงดันจ่ายมาให้ที่ขา 40 (Vcc) ที่ Socket ของ CPU โปรแกรมก็จะหยุดการทำงาน แล้วออกมาที่ระบบปฏิบัติการ (dos) โดยจะมีข้อความแสดงที่จอภาพดังนี้

Your system is not ready !

เมื่อเกิดเหตุการเช่นนี้ ให้ทำการจ่ายไฟให้กับระบบนั้น ๆ ก่อนที่จะเรียกโปร-แกรมมาใช้งาน

หลังจากที่เรียกตัวโปรแกรมมาใช้งานได้แล้ว ซึ่งจะเป็นหน้าจอแรกจะมีหน้าต่างแสดง Version และ รายชื่อผู้ร่วมงาน แสดงให้เห็น จากนั้นให้กดคีย์ใด ๆ เพื่อทำให้ช่องหน้าต่างนี้หายไป หลังจากนั้นจึงเริ่มทำการ Load ตัวโปรแกรมที่เป็นชุดคำสั่งของ CPU เบอร์ 8085 เข้ามายังหน่วยความจำของ IBM/PC ไม่ว่าจะเป็นการ Load มาจาก EPROM ของระบบนั้น ๆ หรือ จากแผ่น Disk ก็ตาม แล้วจึงเลือกทำงานตามหัวข้อ menu ต่าง ๆ ตามที่ต้องการได้ สำหรับวิธีการใช้งานในหัวข้อ menu ต่าง ๆ นั้น ให้ศึกษาจากวิธีการใช้งานระบบ menuในบทที่ 3 ซึ่งได้อธิบายไว้แล้ว อย่างละเอียด

บทที่ 6

บทสรุปและวิจารณ์

6.1 ข้อจำกัดทางด้านฮาร์ดแวร์

เนื่องจากลักษณะวงจรทางฮาร์ดแวร์ที่ทางกลุ่มได้จัดสร้าง ขึ้นมานั้นจะมีส่วนที่ทำหน้าที่หลัก ๆ อยู่ 3 ส่วนดังนี้คือ ส่วนแรกคือส่วนที่เป็นการ Decode Address ที่ใช้ติดต่อกับ IBM/PC/XT/AT ซึ่งสามารถที่จะต่อกับระบบ AT bus ได้โดยเสียบ Card ที่ใช้เชื่อมต่อเข้ากับ Slot ที่ยังว่างอยู่ในเครื่องไมโครโปรเซสเซอร์ หน้าที่หลักของวงจรในส่วนนี้ก็คือจะต้องนำค่า Address ของ IBM/PC มา Decode เพื่อเลือกเบอร์ port ที่จะใช้ติดต่อกับเครื่อง 8085 EMULATOR นี้ โดยในโครงงานนี้ได้กำหนดไว้ที่ Address ตั้งแต่ 300h - 307h แต่ทั้งนี้ค่า Address ดังกล่าวสามารถที่จะทำการเปลี่ยนแปลงได้โดยการติดตั้งค่า Dip-Switch ใหม่ ในกรณีที่ฮาร์ดแวร์ของ IBM/PC ได้นำค่า Address นี้ไปใช้งานแล้ว และจะต้องแก้ไขโปรแกรมในส่วนที่ได้ define Port ไว้ให้ตรงกับค่า Address ใหม่ที่ได้กำหนดขึ้นด้วย ส่วนที่ 2 นั้นจะเป็นวงจรในส่วนที่ทำหน้าที่แทนขาสัญญาณทางอินพุทและขาสัญญาณแบบสองทิศทางของ CPU เบอร์ 8085 โดยส่วนใหญ่จะใช้ IC ที่เป็น Buffer และสำหรับส่วนสุดท้ายนั้นจะเป็นวงจรในส่วนที่เป็น เอาท์พุท ซึ่งทำหน้าที่แทนขาสัญญาณต่าง ๆ ทางเอาท์พุทของ CPU เบอร์ 8085 ซึ่งคุณสมบัติของ IC ที่ใช้จะต้องมีคุณสมบัติในการที่จะ Latch ข้อมูลได้ด้วย

ที่กล่าวมาขั้นต้นนั้น เป็นการกล่าวสรุปถึงฮาร์แวร์ที่ได้ออกแบบไว้ สำหรับข้อจำกัดทางด้านฮาร์ดแวร์ในโครงงานนี้นั้น จะเป็นทางด้านของความเร็วในการทำงานของ IC แต่ละตัว ในการที่จะส่งสภาวะตามขาสัญญาณต่าง ๆ ให้เหมือนกับ CPU ตัวจริง และเนื่องจากการ Decode Address ได้ทำการ Decode เบอร์ port ไว้ 8 ค่าด้วยกันคือ ตั้งแต่ 300h - 307h และใน port แต่ละเบอร์นั้นก็สามารถที่จะรับหรือส่งข้อมูลได้ครั้งละ 8 บิต ซึ่งไม่สามารถที่จะเขียนโปรแกรมให้ควบคุมทีละ 1 บิตได้ ดังนั้นในการที่จะทำสัญญาณใด Active หรือไม่ให้ Active นั้น ก็จะต้องทำการตรวจดูก่อนว่า ขาสัญญาณนั้นอยู่ในกลุ่มของ port ใด แล้วจึงทำการส่งค่าของข้อมูลที่ทำให้บิตนั้น Active ออกไป ซึ่งทำให้โปรแกรมในการควบคุมทำงานได้ช้าและไม่สะดวกในการเขียนโปรแกรม

6.2 ข้อจำกัดทางด้านซอฟแวร์

ในด้านการออกแบบโครงสร้างทางด้านซอฟแวร์นั้นกล่าวโดยสรุปก็คือ ได้ออกแบบ เป็น 2 ส่วนคือ ในส่วนแรกนั้นจะเป็นการออกแบบระบบโครงสร้างของระบบ menu เพื่อทำให้ผู้นำไปใช้ ได้ใช้งานได้ง่ายโดยใช้เวลาศึกษาวิธีการใช้โปรแกรมไม่นานนักก็สามารถที่จะนำโปรแกรมไปใช้ได้อย่างมีประสิทธิภาพ เพราะระบบโครงสร้าง menu ที่เขียนขึ้นเป็น โปรแกรมนี้เป็นแบบ pop-up และ pull down menu ซึ่งง่ายต่อการใช้งาน เพราะได้แบ่งเป็นหัวข้อไว้แล้วว่าจะทำงานเกี่ยวกับอะไร โดยแบ่งเป็นหัวข้อ menu หลักได้ 5 หัวข้อดังนี้คือ File, Edit, Run, Other และ Quit หลังจากที่มีการกดคีย์ F10 ให้มีการ Active ที่หัวข้อใดหัวข้อหนึ่งแล้ว ก็จะมีคำอธิบายสั้น ๆ ที่บรรทัดล่างสุดของหน้าจอว่าแต่ละหัวข้อ menu นั้นมีการใช้งานอะไรได้บ้าง ซึ่งช่วยทำให้ผู้ใช้ไม่ต้องจำว่าในแต่ละหัวข้อ menu นั้นมีการใช้งานทางด้านไหนบ้าง

ส่วนที่ 2 ของการออกแบบโครงสร้างของโปรแกรมนั้นก็คือ ส่วนที่ทำหน้าที่เป็น Function การทำงานต่าง ๆ ในแต่ละหัวข้อเลือก menu ซึ่งจะเป็นการกำหนดรายละเอียดว่าในแต่ละ Function นั้นทำงานอย่างไร โดยได้ออกแบบเป็น Flow chart ดังที่ได้กล่าวไปแล้วในบทที่ 4 ในหัวข้อ 4.2

ส่วนข้อจำกัดทางการซอฟแวร์ที่มีนั้นก็เป็นทางด้าน ความเร็วของการ Run โปรแกรม เพราะเนื่องจากการเขียน Function ที่ทำหน้าที่เป็นชุดคำสั่งของ CPU เบอร์ 8085 นั้น ใช้ภาษาระดับสูงเขียน (ภาษา C) ดังนั้นจึงต้องเสียเวลาในการปฏิบัติตามคำสั่งใน Function นั้น ๆ เช่นอาจจะต้องมีการทำคำสั่ง inport หรือ outport หลายคำสั่งเพื่อให้ทำหน้าที่แทนชุดคำสั่งของ CPU - 8085 เพียงคำสั่งเดียว ดังนั้นโอกาสที่จะทำให้โปรแกรมหรือ Function ที่เขียนขึ้นมาเพื่อแทนชุดคำสั่งของ CPU - 8085 นั้นให้ทำงานได้ตรงตามเวลาจริง (Real time) แทบจะไม่มีเลย แต่ถ้าเรานำชุดคำสั่งของ CPU - 8085 ที่เขียนขึ้นด้วยภาษาระดับสูงนี้มาทำงานในแบบทีละคำสั่ง (Single step) ดังนั้นจึงไม่จำเป็นต้อง Run ตามเวลาจริง เพราะเป็นการทำงานทีละคำสั่งหรือทีละ Function ในภาษาระดับสูงนั่นเอง ซึ่งจะต้องแสดงข้อมูลในของ Register และค่าสภาวะของ Flags ต่าง ๆ รวมถึงข้อมูลในหน่วยความจำที่เปลี่ยนแปลงไปตามการปฏิบัติตามชุดคำสั่งของ CPU - 8085 นั้น ๆ ด้วย และข้อจำกัดอีกอย่างหนึ่งก็คือการเขียนโปรแกรมในการส่งสัญญาณออกตามขาสัญญาณต่าง ๆ ของ CPU - 8085 เพราะจะต้องส่งออกไปทีละ port หรือทีละ 8 บิต ไม่สามารถที่จะเขียนควบคุมทีละบิตได้

6.3 บทสรุป

เนื้อหาในโครงงานชิ้นนี้คือ เครื่อง 8085 EMULATOR ซึ่งหน้าที่หลักในการที่จะจำลองหรือเลียนแบบการทำงานของ CPU เบอร์ 8085 ของบริษัท Intel เพื่อใช้ในการที่จะนำไปใช้ช่วยวิเคราะห์ในการตรวจซ่อมระบบควบคุมระบบใดระบบหนึ่งที่มี CPU เบอร์ 8085 เป็นตัวควบคุมการทำงานอยู่ได้ เพราะจะทำให้สามารถช่วยลดเวลาในการตรวจซ่อมลงไปได้มาก เนื่องจากว่าสามารถที่จะนำไปช่วยวิเคราะห์ข้อผิดพลาดที่เกิดขึ้นทั้งจากทางด้านฮาร์ดแวร์และซอฟแวร์ได้อย่างสะดวกและรวดเร็ว เพราะตัวโปรแกรมได้ถูกออกแบบให้ใช้งานในระบบ pop-up และ pull down menu ซึ่งทำให้ง่ายต่อการนำไปใช้งาน

เครื่องต้นแบบที่ได้ทำการออกแบบ และสร้างพร้อมกับทดลองการทำงานแล้วนี้ได้ทำสำเร็จตามวัตถุประสงค์ของท่านอาจารย์ที่ปรึกษา ที่ได้ให้หลักการในการสร้างโครงงานชิ้นนี้ขึ้นมาเรียบร้อยแล้ว และ Function การใช้งานในบาง Function นั้น ยังมีความสามารถที่จะทำงานได้เกินขอบเขต จากที่อาจารย์ที่ปรึกษาได้กำหนดไว้ คือ Function ในการส่งไฟล์ พิมพ์ออกทางเครื่องพิมพ์ และจัดเก็บไฟล์ที่ Load มาได้จากระบบภายนอกนำไปเก็บไว้ในแผ่น Disk ได้ รวมทั้งยังสามารถที่จะ Load เข้ามาจากแผ่น Disk เพื่อนำมาใช้งานในครั้งต่อไปได้อีกด้วย ส่วนความสามารถในการที่จะพิมพ์ตัวโปรแกรม ออกทางเครื่องพิมพ์นั้นยังสามารถที่จะเลือกได้อีกว่า จะให้พิมพ์ในลักษณะการ Dump ไฟล์ออกมาในรูปแบบเลขฐานสิบหก (Hexadecimal) หรือต้องการที่จะพิมพ์ออกมาในรูปแบบมาตราฐานที่ใช้ในการเขียนโปรแกรมภาษา Assembly ได้อีกด้วย แต่ความสามารถอาจจะไม่ตรงตามรูปแบบมาตราฐาน 100 เปอร์เซ็น ซึ่งจะมีเพียง ค่า Address ตัว Opcode และรหัส Mnemonic เท่านั้น ส่วนหัวข้อ Comment นั้น ต้องมาใส่การทำงานของโปรแกรมแล้วเขียนเอาเอง

อย่างไรก็ตาม ถึงแม้ว่าทางกลุ่มได้ทำสำเร็จตามวัตถุประสงค์ ตามที่ท่านอาจารย์ที่ปรึกษากำหนดแล้วก็ตาม ความสามารถของเครื่องอีมูเลเต่อร์เครื่องนี้ยังจะต้องมีการพัฒนาต่อไปเพื่อให้มีประสิทธิภาพเพิ่มมากขึ้น อย่างเช่นในเรื่องของการทำงานในแบบตรงตามเวลาจริง (Real time) โครงงานนี้ยังมีความสามารถไม่ถึงขั้นการทำงานในลักษณะ IN-CIRCUIT EMULATOR ได้ ซึ่งเป็นส่วนที่จะต้องมีการพัฒนาต่อไป

ภาคผนวก

ในภาคผนวกนี้ได้แบ่งเป็น 3 ส่วนคือ ผ-1 ผ-2 และ ผ-3 โดยใน ผ-1 นั้นจะเป็นวงจรในส่วนที่เป็นการทดลองการทำงานของเครื่อง 8085 EMULATOR และใน ผ-2 นั้น จะเป็นตัวโปรแกรมภาษา C ที่เขียนขึ้นมาเพื่อใช้ในการควบคุมการทำงานของเครื่อง 8085 EMULATOR นี้ ส่วนใน ผ-3 นั้นจะแผ่นปรินซ์ที่แสดงลายทองแดงทั้งด้านบนและด้านล่าง

ผ-1

วงจรที่ใช้ในการทดลองนี้ใช้ LED แบบ Matrix ขนาด 8x8 ดวง เพื่อแสดงเป็นไฟวิ่ง ซึ่งแสดงได้ดังรูป ผ-1.1 โดยมีหลักการดังนี้คือ เนื่องจากชุด LED นี้มีการ CONTROL ได้ทั้ง 2 ด้านแต่ไฟวิ่งแถวเดียวต้องการจุด CONTROL เพียงด้านเดียวก็พอแล้ว ดังนั้นเราก็ให้ด้านหนึ่ง ACTIVE ตลอดไปเลย ส่วนการติดดับของไฟก็ใช้ทางด้าน Data ทั้ง 8 เส้น โดยครั้งแรกให้หลักแรกกับหลักที่สองติดก่อน แล้วต่อไปจึงให้หลักที่สามและสี่ติด ซึ่งการที่ไฟวิ่งได้ก็เพราะการหน่วงเวลาระหว่างการเปลี่ยนหลักที่จะติดนี่เอง จึงทำให้ตาเราเห็นการเปลี่ยนแปลง และลำดับขั้นการเขียนโปรแกรมควบคุมแสดงได้ดัง Flowchart ในรูปที่ ผ-1.2 ส่วนตัวโปรแกรมใช้งานจริงนั้นอยู่ในรูปที่ ผ-1.3 ซึ่งได้จากการพิมพ์จากโปรแกรม EMU8085 โดยเลือกจากหัวข้อ menu Dump/Memonic

```
Addr      Opcode      Mnemonic        Comment

0000      3E 89       MVI A,89h
0002      D3 13       OUT 13h
0004      3E 00       MVI A,00h
0006      D3 11       OUT 11h
0008      3E FC       MVI A,FCh
000A      D3 10       OUT 10h
000C      CD 13 00    CALL 0013h
000F      07          RLC
0010      C3 0A 00    JMP 000Ah
0013      F5          PUSH PSW
0014      21 01 00    LXI H,0001h
0017      2B          DCX H
0018      7C          MOV A,H
0019      B5          ORA L
001A      C2 17 00    JNZ 0017h
001D      F1          POP PSW
001E      C9          RET
001F      00          NOP
0020      00          NOP
0021      00          NOP
0022      00          NOP
```

รูปที่ ผ-1.3 แสดงตัวโปรแกรมควบคุมไฟวิ่งโดยใช้ภาษา Assembly 8085

ผ-2

ในภาคผนวก ผ-2 นี้จะเป็นตัวโปรแกรม EMU8085.C ซึ่งจะต้องใช้ไฟล์อื่น ๆ ร่วมในการ Complier ร่วมกันทั้งหมด 12 ไฟล์ คือ EMU8085.C, FILE_MEM.C, EDIT_MEN.C, RUN_MENU.C, RUN_SUB.C, OTHER_ME.C, QUIT_MEN.C, OPCODE1.C, OPCODE2.C, UNASSEM.C, INSTRUCT.C, WIN_MENU.C และไฟล์จำหน้า (header file) อีก 2 ไฟล์คือ DISPLAY.H และ MENU.H ซึ่งรายละเอียดจะอยู่ในหน้าถัดไป

DISPLAY.H

```
/*********************************************************
*   The is a heder file that use for Display system     *
*********************************************************/


#define C_UL      218       /* upper left  */
#define C_UR      191       /* upper right */
#define C_LL      192       /* lower left  */
#define C_LR      217       /* lower right */
#define C_XD      194       /* T-intersection down */
#define C_XU      193       /* T-intersection up */
#define C_XL      195       /* T-intersection left */
#define C_XR      180       /* T-intersection right */
#define C_XX      197       /* 4-way intersection */

#define C_H       196       /* horizontal line */
#define C_V       179       /* vertical line */

#define C_ULD     201       /* upper left  */
#define C_URD     187       /* upper right  */
#define C_LLD     200       /* lower left  */
#define C_LRD     188       /* lower right */
#define C_XDD     203       /* T-intersection down */
#define C_XUD     202       /* T-intersection up */
#define C_XLD     204       /* T-intersection left */
#define C_XRD     185       /* T-intersection right */
#define C_XXD     206       /* 4-way intersection */

#define C_HD      205       /* horizontal line */
#define C_VD      186       /* vertical line */

#define C_ULSD    213       /* single and double upper left */
#define C_URSD    184       /* single and double upper right */
```

```
#define C_LLSD     212          /* single and double lower left */
#define C_LRSD     190          /* single and double lower right */
#define C_XDSD     209          /* T-intersection down */
#define C_XUSD     207          /* T-intersection up */
#define C_XLSD     198          /* T-intersection left */
#define C_XRSD     181          /* T-intersection right */
#define C_XXSD     216          /* 4-way intersection */

#define C_ULDS     214          /* double and single upper left */
#define C_URDS     183          /* double and single upper right */
#define C_LLDS     211          /* double and single lower left */
#define C_LRDS     189          /* double and single lower right */
#define C_XDDS     210          /* T-intersection down */
#define C_XUDS     208          /* T-intersection up */
#define C_XLDS     199          /* T-intersection left */
#define C_XRDS     182          /* T-intersection right */
#define C_XXDS     215          /* 4-way intersection */

#define C_HATCH    176          /* hatching */
#define C_HT       240          /* 3-horizontal line */
#define C_SPACE    255          /* alternate space */

#define A_NORM     0x07         /* normal */
#define A_INVERSE  0x70         /* inverse */
#define A_INTENSE  0x0F         /* intense */
#define A_BLINK    0x87         /* blink */
#define A_UNDER    0x01         /* underline */
```

MENU . H

```
/*************************************************************************
*                                menu.h                                 *
* This header file contains the neccessary over hesd information for *
* the slidebar and box menuing sustems.                                 *
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <dos.h>
#include <alloc.h>          /* May be alloc.h in some complilers */
#include <mem.h>            /* May be mem.h in some complilers */
#include <string.h>

/********* Data structures and Definitions **************/

/* Slidebar menu structure  */
typedef struct {
    char  *prompt;          /* First line menu prompt      */
    int   plen;             /* Its length                  */
    char  *deso;            /* Second line menu prompt     */
    int   dlen;             /* Its length                  */
    char  letter;           /* Singel letter choice option */
    } MENU;

#ifdef  EXTERN
    extern
#endif
    char   *cptr;           /* Vector of single char choices   */
#ifdef  EXTERN
    extern
#endif
```

```
    int   increment,          /* Relative movement of marker  */
    last,                     /* The last menu choice made  */
    menucount,                /* The number of items in the menu  */
    menulen,                  /* The length of all menu choices  */
    mouse_here,               /* Is there a mouse present  */
    *nptr,                    /* Pointer to menu column positions */
    pcol,                     /* Primary column  */
    prow;                     /* Primary row  */
/************* Macros *************/

#ifndef   TRUE
    #define TRUE    1
#endif

#define   HOME      71    /* Extended key codes for keypad    */
#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 Function keys */
#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 mouse buttons  */
#define  RBOTTON     2
#define  MBUTTON     4

#define  SIDEWAYS    0        /* Direction of mouse travel  */
#define  UPDOWN      1

#define  SINGLEBAR   1        /* For box drawing  */
#define  DOUBLEBAR   2

#define  UPA         24
#define  DOWNA       25

#define  ENTER       '\r'   /* This may be '\n' on some compilers */
#define  ESCAPE      27

/* Macros for two-line menu bar  */

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

#define  MARGIN      "        " /* Spacing between menu items   */
#define  BELL        7        /* Terminal Alarm                */

#define  MAXWIDE     80       /* Maximum screen width-depth    */
#define  MAXDEEP     24

#define  DELTA       50       /* Mickeys/menu change           */
#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 marker),
 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 back),
     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),
     prepars(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);
```

EMU8085.C

```
/*************************************************************************
*    This is a 8085 emulator program that uses the menuing system.      *
*                                                                       *
*    written  By.....    Mr. Prarinya Ekapho    code. 33162213         *
*    Faculty of  " ENGINEERING "                                        *
*    Major:  Industial computer technology                              *
*    King Mongkut's Institute of Technology Ladkrabang.                 *
*************************************************************************/

#include <stdio.h>
#include "menu.h"
#include "display.h"

#include "b:\instruct.c"
#include "b:\win_menu.c"
#include "b:\file_men.c"
#include "b:\edit_men.c"
#include "b:\run_menu.c"
#include "b:\other_me.c"
#include "b:\quit_men.c"
#include "b:\unassem.c"
#include "b:\opcode1.c"
#include "b:\opcode2.c"
#include "b:\run_sub.c"

int main()
{
  unsigned char vcc;
  int back, chk=0, fore, mode;
  p_count=0;s_point=0x1FE8;
  reg_a=0;reg_b=0;reg_c=0;reg_d=0;reg_e=0;
  reg_h=0;reg_l=0;psw=0;
  pcram = (unsigned char *) malloc(NUM*sizeof(unsigned char));
```

```
if(!pcram) {
  printf(" out of memory\n");
  exit(0);
}
vcc = inportb(PORT_C);
vcc&=0x20;
if(vcc!=0x20) {
   printf("Your system is not ready !!!\n");
   exit(0);
}
initial();
/*  create all menu and window system */
make_window(0, " PROJECT ", 8, 19, 18, 60, BORDER);
make_window(1, " Out port service ", 16, 19, 18, 60, BORDER);
make_window(2, " Modify register service ", 5, 15, 9, 64, BORDER);
make_window(3, " C O N F I R M ", 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 Program ", 4, 19, 8, 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," Display data memory ", 8, 12, 16, 71, BORDER);
make_menu(0, file, "lsdt", 4, 2, 2, BORDER);
make_menu(1, run, "rsp", 3, 2, 31, BORDER);
make_menu(2, other, "mt", 2, 2, 49, BORDER);
make_menu(3, quit, "ve", 2, 2 , 62, BORDER);
make_menu(4, testport, "abcdefgh", 8, 5, 40, BORDER);
make_menu(5, type_port, "io", 2, 9, 31, BORDER);
make_menu(6, edit, "mv", 2, 2, 19, BORDER);
make_menu(7, t_mode, "ls", 2, 7, 3, BORDER);
make_menu(8, disp_file, "hm", 2, 6, 3, BORDER);
make_menu(9, view_mode, "pd", 2, 5, 20, BORDER);
```

```
    cls();
    cursor(1,17);
    putchar(C_HT);putchar(C_HT);putchar(C_HT);
    printf(" %s ",BANNER);
    putchar(C_HT);putchar(C_HT);putchar(C_HT);
    border();
    cursor(2,5);
    mode = getvmode();
    if (mode == 1 || mode == 3) {              /* Color... */
        fore = 0;                              /* Blue (1) */
        back = 7;                              /* Cyan (3) */
    } else {                                   /* or B&W   */
        fore = 0;
        back = 7;
    };
    menu_start(menu0, fore, back);
    disp_version();
    do {
    first_active(menu0, fore, back);
    chk = menu_choice(menu0, fore, back);
      if(chk == -2) {
        menu_end();
        free(pcram);
        cls();cursor(1,1);
        exit(0);
      };
    } while(TRUE);
}
/*************** END OF MAIN ******************/
```

FILE_MEN.C

```
/*****************************
*         MENU FILE          *
*****************************/
menu_file()
{
  int choice,chk;

/* now, activate as needed */
  arrow_choice = 0;
  key_geted = 0;
  chk = 0;
  pulldown(0);
  while((choice=get_resp(0)) != -1) {
   switch(choice)  {
    case 0:
      loadfile();
      arrow_get=10;
      return;
    case 1:
      savefile();
      break;
    case 2:
      dump_file();
      break;
    case 3:
      chk = transfer();
      if(chk==1) return;
      break;
   }
  }
  restore_video(0);
  disp_pc();
}
```

```
/*** Load file from disk ***/
loadfile()
{
  int i;
  char fn[80];
  FILE *point;

  window_(4);window_cls(4);
  window_xy(4, 0, 1);
  window_gets(4, fn);
  deactivate_win(4);
  restore_video(0);
  hide_cursor();
  if(*fn == -1) return;
  if((point=fopen(fn,"rb"))==NULL) {
   putchar(BELL);
   clsmess();
   cursor(23,31);
   printf("File not found !!!");
   cursor(23,31);
   get_key();
   hide_cursor();
 } else {
  clsmess();
  cursor(23,32);
  printf("Loading File ...");
  cursor(23,32);
  for(i=0;i<NUM;++i) {
    *(pcram+i)=fgetc(point);
    p_count++;
  }
  disp_pc();
  clsmess();
```

```
    cursor(23,23);
    printf("Data %d bytes are Transfered ...",NUM);
    unassem();
  }
 }

/*** Save file to disk ***/
savefile()
{
  int i;
  char fn[20];
  FILE *fp;

  window_(5);window_cls(5);
  window_xy(5, 0, 1);
  window_gets(5, fn);
  deactivate_win(5);
  hide_cursor();
  if(*fn == -1) return;
  if((fp=fopen(fn,"wb"))==NULL) {
   putchar(BELL);
   clsmess();
   cursor(23,30);
   printf("Can't open file !!!");
   cursor(23,30);
   get_key();
   clsmess();
   hide_cursor();
  } else {
  clsmess();
  cursor(23,32);
  printf("Saving File ...");
  cursor(23,32);
```

```
    for(i=0;i<NUM;++i) {
     fputc(*(pcram+i),fp);
    }
   fclose(fp);
   clsmess();
   hide_cursor();
   }
}
/*** dump file in thexl ***/
dump_file()
{
   int item,stop_win;

   arrow_item = 0;key_recive = 0;stop_win=0;
   pulldown(8);
   while((item=get_resp_sub(8)) != -1) {
     switch(item) {
       case 0:
              stop_win=mess_dump();
              if(stop_win==-1) break;
              hex_dump();
              mess_end();
              break;
       case 1:
              stop_win=mess_dump();
              if(stop_win==-1) break;
              mnemo_dump();
              mess_end();
              break;
     }
   } /* end of loop while */
   restore_video(8);
}
```

```
mess_dump()
{

  char temp[4],chk_p;

 window_(8);
 window_cls(8);
 window_xy(8, 0, 2);
 window_puts(8, "Enter start address : ",A_INVERSE);
 window_gets(8, temp,A_INVERSE);
 if(temp[0]==-1) { deactivate_win(8); hide_cursor(); return -1; }
 deactivate_win(8);
 hide_cursor();
 p_count=ch_to_hex(temp);
 cursor(23,13);
 printf("Check printer for ready then press ENTER to start ...");
 chk_p = get_key();
 if(tolower(chk_p) != ENTER) {
   clsmess();
   hide_cursor();
   return -1;
 }
 clsmess();
 cursor(23,31);
 printf("Now dumping file ...");
 cursor(23,31);
}

mess_end()
{
      clsmess();
      cursor(23,31);
      printf("End of dumping ...");
```

```
        get_key();
        clsmess();
        hide_cursor();
}
hex_dump()
{
   int i,j,page;
   unsigned char k;
   char stop;

   i=0; page=1;

   while(p_count!=NUM) {
     fprintf(stdprn,"\n\n\n");
     fprintf(stdprn,"                                        ");
     fprintf(stdprn,"                    Page %d\n",page);
     fprintf(stdprn,"\n\n");
     fprintf(stdprn,"          ");
     fprintf(stdprn,"                Data from file [hex]\n\n");
     fprintf(stdprn,"           Addr : ");
     for(k=0;k<=0xf;k++)  fhex_up8(k);
     fprintf(stdprn,"\n\n");
     while(i!=50 && p_count!=NUM) {
             fprintf(stdprn,"          ");
             fhex_up16(p_count);
             for(j=0; j<=15 ; ++j,++p_count)
                  fhex_up8(*(pcram+p_count));
             fprintf(stdprn,"\n");
             i++;
     };
     if(i==50) {
        i=0;
        ++page;
```

```
            clsmess();
            cursor(23,22);
            printf("Press any key to continue dumping page %d ...",page);
            stop = get_key();
            if(stop==ESCAPE) {
                 clsmess();
                 hide_cursor();
                 return;
            }
            clsmess();
            hide_cursor();
       }
    }
    fprintf(stdprn,"\n");
    fprintf(stdprn,"          Data %d bytes has been dumped ...",NUM);
}

mnemo_dump()
{
    int r,page;
    char stop;

    page=1;

    while(p_count!=NUM) {
       fprintf(stdprn,"\n\n\n");
       fprintf(stdprn,"                                          ");
       fprintf(stdprn,"                    Page %d\n",page);
       fprintf(stdprn,"\n\n");
       fprintf(stdprn,"          ");
       fprintf(stdprn,"Addr");fprintf(stdprn,"      ");
       fprintf(stdprn,"Opcode");fprintf(stdprn,"      ");
       fprintf(stdprn,"Mnemonic");fprintf(stdprn,"       ");
```

```
        fprintf(stdprn,"Comment");
        fprintf(stdprn,"\n\n");
        r=0;
        while(r!=52) {
          fprintf(stdprn,"              ");
          fmhex_up16(p_count);
          fprintf(stdprn,"      ");chk_op(PRINT);
          fprintf(stdprn,"\n");
          r++;
          p_count++;
        }
       if(r==52) {
          ++page;
          clsmess();
          cursor(23,22);
          printf("Press any key to continue dumping page %d ...",page);
          stop = get_key();
       if(stop==ESCAPE) {
          clsmess();
          hide_cursor();
          return;
       }
       clsmess();
       hide_cursor();
       }
    }
}
/***  Transfer data  ***/
transfer()
{
  int item;
  arrow_item = 0;key_recive = 0;
  pulldown(7);
```

```
    while((item=get_resp_sub(7)) != -1) {
      switch(item) {
        case 0:
          t_eprom();
          arrow_get=10;
          restore_video(7);
          restore_video(0);
          unassem();
          return 1;
        case 1:
          t_ram();
          break;
      }
    }
    restore_video(7);
}

t_eprom()
{
    unsigned char prv_h,prv_l;
    int i,t;
    prv_h=reg_h; prv_l=reg_l;
    reg_h=0; reg_l=0;
    p_count=0;

    for(p_count=0;p_count<NUM;++p_count,++reg_l) {
      *(pcram+p_count) = RD_r_M_F();
      if(reg_l==0xff)
      ++reg_h;
    }
    reg_h=prv_h; reg_l=prv_l;
}
```

```
t_ram()
{
  unsigned char prv_h,prv_l;
  int i;
  prv_h=reg_h; prv_l=reg_l;
  reg_h=0; reg_l=0;
  p_count=0;
  cursor(23,28);
  printf("Saving data into RAM ...");
  cursor(23,28);
  for(p_count=0;p_count<NUM;++p_count,++reg_l) {
    WR_r_M_F(*(pcram+p_count));
    if(reg_l==0xff)
    ++reg_h;
  }
  clsmess();
  cursor(23,30);
  printf("Save successful !!! ");
  get_key();
  clsmess();
  hide_cursor();
  reg_h=prv_h; reg_l=prv_l;
}

/*** Unassembler file ***/
unassem()
{
  int r;

      p_count = 0;
      disp_pc();
      r = J1_HEIGHT;
      cursor(r,40);putchar(C_XDSD);
```

```
        ++r;
        cursor(r,3);printf("Addr");cursor(r,11);printf("Opcode");
        cursor(r,23);printf("Mnemonic");cursor(r,40);putchar(C_V);
        cursor(r,42);printf("Addr");cursor(r,50);printf("Opcode");
        cursor(r,62);printf("Mnemonic");
      for(r+=1;r<=21;p_count++,r++) {
          cursor(r,3);
          printf("                                    ");
          cursor(r,3);hex_up16(p_count);
          printf("    ");chk_op(SCREEN);
        }
        r = J1_HEIGHT+1;
        for(r+=1;r<=21;p_count++,r++) {
          cursor(r,40);putchar(C_V);
          printf("                                    ");
          cursor(r,42);hex_up16(p_count);
          printf("    ");chk_op(SCREEN);
        }
        disp_pc();
        hide_cursor();
}
/***** End of menu file *******/
```

EDIT_MEN.C

```
/*********************************
*          MENU EDIT            *
*********************************/
menu_edit()
{
   int choice;
   arrow_choice = 0; key_geted = 0;
   pulldown(6);
   while((choice = get_resp(6)) != -1) {
     switch(choice) {
       case 0:
            modi_data();
            break;
       case 1:
            view_data();
            break;
     }
   }
   restore_video(6);
}

modi_data()
{
   unsigned char n0,n1;
   unsigned int addr;
   char temp_pc[5],new[3],any[2];
   char new_var;
   int  r;

   addr=p_count;
   window_(7);
   do {
     window_cls(7);
```

```
        window_xy(7, 0, 0);
        window_puts(7, " Enter Address to edit  :  ",A_INVERSE);
        window_gets(7, temp_pc);
        if(temp_pc[0]== -1) {
              deactivate_win(7); hide_cursor(); return;
        }
        p_count = ch_to_hex(temp_pc);
        window_xy(7, 1, 0);
        window_puts(7, " Old value : ",A_INVERSE);
        n1 = n0 = *(pcram+p_count);
        n0 &= 0x0f;
        n1 >>= 4;
        n1 += (n1 < 0xA) ? 0x30 : 0x37;
        n0 += (n0 < 0xA) ? 0x30 : 0x37;
        window_putchar(7, n1,A_INVERSE);
        window_putchar(7, n0,A_INVERSE);
        window_puts(7, "    New value : ",A_INVERSE);
        window_gets(7, new);
        new_var = ch_to_hex(new);
        *(pcram+p_count) = new_var;
        window_xy(7, 2, 0);
        window_puts(7, " Edit Another [y/n] ? ",A_INVERSE);
        window_gets(7, any);
    } while (any[0] == 'y' || any[0] == 'Y');
    deactivate_win(7);
    p_count=addr;
    hide_cursor();
}

/******* End of modi data ********/

view_data()
{
```

```
    int item;
    arrow_item=0;key_recive=0;
    pulldown(9);
    while((item=get_resp_sub(9)) != -1) {
     switch(item) {
       case 0:
              prog_list();
              return;
       case 1:
              data_list();
              break;
     }
    }
    restore_video(9);
}

enter_addr()
{
   char temp[5];

     window_(6);window_cls(6);
     window_xy(6, 1, 0);
     window_puts(6, "   Enter begin address : ",A_INVERSE);
     window_gets(6, temp);
     if(temp[0] == -1) { deactivate_win(6); hide_cursor(); return; }
     deactivate_win(6);
     hide_cursor();
     p_count = ch_to_hex(temp);
}

data_list()
{
  unsigned int addr;
```

```
    int r,c;

    enter_addr();
    addr=p_count;
      window_(10);
      window_cls(10);
      window_xy(10, 0, 2);
      window_puts(10,"Addr : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D
                              0E 0F",A_INVERSE);
      r=1;
      while(r!=7) {
       window_xy(10, r, 2);
       win_hex16(p_count);
       window_puts(10," : ",A_INVERSE);
       for(c=0;c<=15;++c,++p_count) {
              win_hex8(*(pcram+p_count));
              if(c!=15)
              window_puts(10," ",A_INTENSE);
       };
       r++;
      };
      hide_cursor();
      get_key();
      deactivate_win(10);
      p_count=addr;
}

prog_list()
{
  int r;

    enter_addr();
    restore_video(9);
```

```
    r = J1_HEIGHT;
    cursor(r,40);putchar(C_XDSD);++r;
    cursor(r,3);printf("Addr");cursor(r,11);printf("Opcode");
    cursor(r,23);printf("Mnemonic");cursor(r,40);putchar(C_V);
    cursor(r,42);printf("Addr");cursor(r,50);printf("Opcode");
    cursor(r,62);printf("Mnemonic");
    for(r+=1;r<=21;p_count++,r++) {
      cursor(r,3);
      printf("                                    ");
      cursor(r,3);hex_up16(p_count);
      printf("    ");chk_op(SCREEN);
    }
    r = J1_HEIGHT+1;
    for(r+=1;r<=21;p_count++,r++) {
      cursor(r,40);putchar(C_V);
      printf("                                    ");
      cursor(r,42);hex_up16(p_count);
      printf("    ");chk_op(SCREEN);
    }
    disp_pc();
    hide_cursor();
}
```

RUN_MENU.C

```
/*****************************
*          MENU RUN           *
*****************************/
menu_run()
{
  int choice,test;

/* now, activate as needed */
arrow_choice = 0;key_geted = 0;
pulldown(1);
while((choice=get_resp(1)) != -1) {
  switch(choice)  {
    case 0:
      test = run_all();
      if(test==1) process_rst();
      hide_cursor();
      arrow_get=10;
      return;
    case 1:
      step();
      hide_cursor();
      arrow_get=10;
      return;
    case 2:
      initial();
      p_count=0;s_point=0x1FE8;
      reg_a=0;reg_b=0;reg_c=0;reg_d=0;reg_e=0;
      reg_h=0;reg_l=0;psw=0;
      sign_f=0;zero_f=0;aux_f=0;parity_f=0;carry_f=0;
      restore_video(1);
      disp_pc();disp_sp();disp_acc();disp_bc();
      disp_de();disp_hl();disp_psw();DISP_ALL_F;
      hide_cursor();
```

```
        arrow_get=10;
        return;
    }
}
    restore_video(1);
}

process_rst()
{
        initial();
        p_count=0;s_point=0x1FE8;
        reg_a=0;reg_b=0;reg_c=0;reg_d=0;reg_e=0;
        reg_h=0;reg_l=0;psw=0;
        sign_f=0;zero_f=0;aux_f=0;parity_f=0;carry_f=0;
        disp_pc();disp_sp();disp_acc();disp_bc();
        disp_de();disp_hl();disp_psw();DISP_ALL_F;
}
/******* End of menu run ********/
```

RUN_SUB.C

```
step()
{
  int test,rst;
  union scan {
  int c;
  char ch[2];
  } sc;

  addr_define();
  note_key();
  while(p_count < p_break) {
    rst = inportb(PORT_C);
    rst&=0x28;
    if(rst!=0x28) { clsdeso(); process_rst(); return; }
    current_addr();
    test=fatch();
    if(test==-1) { clsdeso(); return; }
    ++p_count;
    disp_pc();
    clsmess();
    cursor(23,22);
    printf("The address  ");
    hex_up16(p_count);
    printf("h ");
    printf(" will be executed !");
    do {
      sc.c = get_key();
      if(sc.ch[0]==ESCAPE) { clsdeso(); return; }
      switch(sc.ch[1]) {
        case F5: nextpage(); break;
        case F6: disp_mem(); break;
        case F7: modi_reg(); break;
        default : break;
```

```
        };
        cursor(23,60);
      } while(sc.ch[1] != F8);
    }   /* end of loop while */
    clsdeso();
}    /* end of function step */

nextpage()
{
    int r,pbuff;
    char temp[5];

      pbuff=p_count;
      window_(6);window_cls(6);
      window_xy(6, 1, 0);
      window_puts(6, "   Enter begin address : ",A_INVERSE);
      window_gets(6, temp);
      deactivate_win(6);
      hide_cursor();
      p_count = ch_to_hex(temp);
      r = J1_HEIGHT;
      cursor(r,40);putchar(C_XDSD);++r;
      cursor(r,3);printf("Addr");cursor(r,11);printf("Opcode");
      cursor(r,23);printf("Mnemonic");cursor(r,40);putchar(C_V);
      cursor(r,42);printf("Addr");cursor(r,50);printf("Opcode");
      cursor(r,62);printf("Mnemonic");
      for(r+=1;r<=21;p_count++,r++) {
        cursor(r,3);
        printf("                                    ");
        cursor(r,3);hex_up16(p_count);
        printf("    ");chk_op(SCREEN);
      }
      r = J1_HEIGHT+1;
```

```
    for(r+=1;r<=21;p_count++,r++) {
      cursor(r,40);putchar(C_V);
      printf("                                        ");
      cursor(r,42);hex_up16(p_count);
      printf("    ");chk_op(SCREEN);
    }
    p_count=pbuff;
    disp_pc();
}
disp_mem()
{
  unsigned int addr;
  char temp_pc[5],any[2];
  int r,c;

  addr = p_count;
    window_(10);
    window_cls(10);
    window_xy(10, 1, 10);
    window_puts(10, "Enter start address to view : ",A_INVERSE);
    window_gets(10, temp_pc);
    if(temp_pc[0] == -1) { deactivate_win(10); hide_cursor(); return; }
    p_count = ch_to_hex(temp_pc);
    window_cls(10);
    window_xy(10, 0, 2);
    window_puts(10,"Addr : 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D
                          0E 0F",A_INVERSE);
    r=1;
    while(r!=7) {
      window_xy(10, r, 2);
      win_hex16(p_count);
      window_puts(10," :",A_INVERSE);
        for(c=0;c<=15;++c,++p_count) {
```

```
            win_hex8(*(pcram+p_count));
            if(c!=15)
            window_puts(10," ",A_INTENSE);
           );
         r++;
       );
       hide_cursor();
       get_key();
       deactivate_win(10);
       p_count=addr;
}

addr_define()
{
  char temp_s[5],temp_b[5];

      window_(9);window_cls(9);
      window_xy(9, 0, 0);
      window_puts(9, "  Enter begin address : ",A_INVERSE);
      window_gets(9, temp_s);
      if(temp_s[0] == -1) {
         deactivate_win(9); restore_video(1);
         hide_cursor(); return;
      );
      p_count = ch_to_hex(temp_s);
      window_xy(9, 1, 0);
      window_puts(9, "  Enter stop  address : ",A_INVERSE);
      window_gets(9, temp_b);
      if(temp_b[0] == -1) {
         deactivate_win(9); restore_video(1);
         hide_cursor(); return;
      );
      p_break = ch_to_hex(temp_b);
```

```
    deactivate_win(9);
    restore_video(1);
    hide_cursor();
}

win_hex16(r16)
unsigned int r16;
{
  unsigned int n0,n1,n2,n3;
  n0=n1=n2=n3=r16;
  n0 &= 0x000f; n1 &= 0x00f0; n1 >>= 4;
  n2 &= 0x0f00; n2 >>= 8; n3 &= 0xf000;
  n3 >>= 12;
  n0 += (n0 < 0xA) ? 0x30 : 0x37;
  n1 += (n1 < 0xA) ? 0x30 : 0x37;
  n2 += (n2 < 0xA) ? 0x30 : 0x37;
  n3 += (n3 < 0xA) ? 0x30 : 0x37;
  window_putchar(10, n3,A_INVERSE);
  window_putchar(10, n2,A_INVERSE);
  window_putchar(10, n1,A_INVERSE);
  window_putchar(10, n0,A_INVERSE);
}
win_hex8(r8)
unsigned char r8;
{
  unsigned char n0,n1;
  n0=n1=r8;
  n0 &= 0x0f; n1 >>= 4;
  n1 += (n1 < 0xA) ? 0x30 : 0x37;
  n0 += (n0 < 0xA) ? 0x30 : 0x37;
  window_putchar(10, n1,A_INTENSE);
  window_putchar(10, n0,A_INTENSE);
}
```

```
note_key()
{
   clsdeso();
   cursor(25,1);
   printf("F5-Other address  F6-Data memory  F7-Modify reg.  F8-step
          ESC-Break");
   hide_cursor();
}
clsdeso()
{
   int i;
   for(i=1;i<=79;i++) {
    cursor(25,i);
    printf(" ");
   }
   hide_cursor();
}
current_addr()
{
  outport(PORT_A,p_count);
  outportb(PORT_H,PH_AO);
  outportb(PORT_C,7);
  outport(PORT_B,p_count>>8);
  outportb(PORT_C,6);
  outportb(PORT_D,15);
}
fatch()
{
   switch(*(pcram+p_count)) {
     case 0x00: NOP(); break;
     case 0x01: LXI_B_d16(); break;
     case 0x02: STAX_B(); break;
     case 0x03: INX_B(); break;
```

```
case 0x04: INR_B(); break;
case 0x05: DCR_B(); break;
case 0x06: MVI_B_d8(); break;
case 0x07: RLC(); break;
case 0x08: break;
case 0x09: DAD_B(); break;
case 0x0A: LDAX_B(); break;
case 0x0B: DCX_B(); break;
case 0x0C: INR_C(); break;
case 0x0D: DCR_C(); break;
case 0x0E: MVI_C_d8(); break;
case 0x0F: RRC(); break;
case 0x10: break;
case 0x11: LXI_D_d16(); break;
case 0x12: STAX_D(); break;
case 0x13: INX_D(); break;
case 0x14: INR_D(); break;
case 0x15: DCR_D(); break;
case 0x16: MVI_D_d8(); break;
case 0x17: RAL(); break;
case 0x18: break;
case 0x19: DAD_D(); break;
case 0x1A: LDAX_D(); break;
case 0x1B: DCX_D(); break;
case 0x1C: INR_E(); break;
case 0x1D: DCR_E(); break;
case 0x1E: MVI_E_d8(); break;
case 0x1F: RAR(); break;
case 0x20: RIM(); break;
case 0x21: LXI_H_d16(); break;
case 0x22: SHLD_Adr(); break;
case 0x23: INX_H(); break;
case 0x24: INR_H(); break;
```

```
case 0x25: DCR_H(); break;
case 0x26: MVI_H_d8(); break;
case 0x27: DAA(); break;
case 0x28: break;
case 0x29: DAD_H(); break;
case 0x2A: LHLD_Adr(); break;
case 0x2B: DCX_H(); break;
case 0x2C: INR_L(); break;
case 0x2D: DCR_L(); break;
case 0x2E: MVI_L_d8(); break;
case 0x2F: CMA(); break;
case 0x30: SIM(); break;
case 0x31: LXI_SP_d16(); break;
case 0x32: STA_Adr(); break;
case 0x33: INX_SP(); break;
case 0x34: INR_M(); break;
case 0x35: DCR_M(); break;
case 0x36: MVI_M_d8(); break;
case 0x37: STC(); break;
case 0x38: break;
case 0x39: DAD_SP(); break;
case 0x3A: LDA_Adr(); break;
case 0x3B: DCX_SP(); break;
case 0x3C: INR_A(); break;
case 0x3D: DCR_A(); break;
case 0x3E: MVI_A_d8(); break;
case 0x3F: CMC(); break;
case 0x40: MOV_B_B(); break;
case 0x41: MOV_B_C(); break;
case 0x42: MOV_B_D(); break;
case 0x43: MOV_B_E(); break;
case 0x44: MOV_B_H(); break;
case 0x45: MOV_B_L(); break;
```

```
case 0x46: MOV_B_M(); break;
case 0x47: MOV_B_A(); break;
case 0x48: MOV_C_B(); break;
case 0x49: MOV_C_C(); break;
case 0x4A: MOV_C_D(); break;
case 0x4B: MOV_C_E(); break;
case 0x4C: MOV_C_H(); break;
case 0x4D: MOV_C_L(); break;
case 0x4E: MOV_C_M(); break;
case 0x4F: MOV_C_A(); break;
case 0x50: MOV_D_B(); break;
case 0x51: MOV_D_C(); break;
case 0x52: MOV_D_D(); break;
case 0x53: MOV_D_E(); break;
case 0x54: MOV_D_H(); break;
case 0x55: MOV_D_L(); break;
case 0x56: MOV_D_M(); break;
case 0x57: MOV_D_A(); break;
case 0x58: MOV_E_B(); break;
case 0x59: MOV_E_C(); break;
case 0x5A: MOV_E_D(); break;
case 0x5B: MOV_E_E(); break;
case 0x5C: MOV_E_H(); break;
case 0x5D: MOV_E_L(); break;
case 0x5E: MOV_E_M(); break;
case 0x5F: MOV_E_A(); break;
case 0x60: MOV_H_B(); break;
case 0x61: MOV_H_C(); break;
case 0x62: MOV_H_D(); break;
case 0x63: MOV_H_E(); break;
case 0x64: MOV_H_H(); break;
case 0x65: MOV_H_L(); break;
case 0x66: MOV_H_M(); break;
```

```
case 0x67: MOV_H_A(); break;
case 0x68: MOV_L_B(); break;
case 0x69: MOV_L_C(); break;
case 0x6A: MOV_L_D(); break;
case 0x6B: MOV_L_E(); break;
case 0x6C: MOV_L_H(); break;
case 0x6D: MOV_L_L(); break;
case 0x6E: MOV_L_M(); break;
case 0x6F: MOV_L_A(); break;
case 0x70: MOV_M_B(); break;
case 0x71: MOV_M_C(); break;
case 0x72: MOV_M_D(); break;
case 0x73: MOV_M_E(); break;
case 0x74: MOV_M_H(); break;
case 0x75: MOV_M_L(); break;
case 0x76: HLT(); return -1;
case 0x77: MOV_M_A(); break;
case 0x78: MOV_A_B(); break;
case 0x79: MOV_A_C(); break;
case 0x7A: MOV_A_D(); break;
case 0x7B: MOV_A_E(); break;
case 0x7C: MOV_A_H(); break;
case 0x7D: MOV_A_L(); break;
case 0x7E: MOV_A_M(); break;
case 0x7F: MOV_A_A(); break;
case 0x80: ADD_B(); break;
case 0x81: ADD_C(); break;
case 0x82: ADD_D(); break;
case 0x83: ADD_E(); break;
case 0x84: ADD_H(); break;
case 0x85: ADD_L(); break;
case 0x86: ADD_M(); break;
case 0x87: ADD_A(); break;
```

```
case 0x88: ADC_B(); break;
case 0x89: ADC_C(); break;
case 0x8A: ADC_D(); break;
case 0x8B: ADC_E(); break;
case 0x8C: ADC_H(); break;
case 0x8D: ADC_L(); break;
case 0x8E: ADC_M(); break;
case 0x8F: ADC_A(); break;
case 0x90: SUB_B(); break;
case 0x91: SUB_C(); break;
case 0x92: SUB_D(); break;
case 0x93: SUB_E(); break;
case 0x94: SUB_H(); break;
case 0x95: SUB_L(); break;
case 0x96: SUB_M(); break;
case 0x97: SUB_A(); break;
case 0x98: SBB_B(); break;
case 0x99: SBB_C(); break;
case 0x9A: SBB_D(); break;
case 0x9B: SBB_E(); break;
case 0x9C: SBB_H(); break;
case 0x9D: SBB_L(); break;
case 0x9E: SBB_M(); break;
case 0x9F: SBB_A(); break;
case 0xA0: ANA_B(); break;
case 0xA1: ANA_C(); break;
case 0xA2: ANA_D(); break;
case 0xA3: ANA_E(); break;
case 0xA4: ANA_H(); break;
case 0xA5: ANA_L(); break;
case 0xA6: ANA_M(); break;
case 0xA7: ANA_A(); break;
case 0xA8: XRA_B(); break;
```

```
case 0xA9: XRA_C(); break;
case 0xAA: XRA_D(); break;
case 0xAB: XRA_E(); break;
case 0xAC: XRA_H(); break;
case 0xAD: XRA_L(); break;
case 0xAE: XRA_M(); break;
case 0xAF: XRA_A(); break;
case 0xB0: ORA_B(); break;
case 0xB1: ORA_C(); break;
case 0xB2: ORA_D(); break;
case 0xB3: ORA_E(); break;
case 0xB4: ORA_H(); break;
case 0xB5: ORA_L(); break;
case 0xB6: ORA_M(); break;
case 0xB7: ORA_A(); break;
case 0xB8: CMP_B(); break;
case 0xB9: CMP_C(); break;
case 0xBA: CMP_D(); break;
case 0xBB: CMP_E(); break;
case 0xBC: CMP_H(); break;
case 0xBD: CMP_L(); break;
case 0xBE: CMP_M(); break;
case 0xBF: CMP_A(); break;
case 0xC0: RNZ(); break;
case 0xC1: POP_B(); break;
case 0xC2: JNZ_Adr(); break;
case 0xC3: JMP_Adr(); break;
case 0xC4: CNZ_Adr(); break;
case 0xC5: PUSH_B(); break;
case 0xC6: ADI_d8(); break;
case 0xC7: RST_0(); break;
case 0xC8: RZ(); break;
case 0xC9: RET(); break;
```

```
case 0xCA: JZ_Adr(); break;
case 0xCB: break;
case 0xCC: CZ_Adr(); break;
case 0xCD: CALL_Adr(); break;
case 0xCE: ACI_d8(); break;
case 0xCF: RST_1(); break;
case 0xD0: RNC(); break;
case 0xD1: POP_D(); break;
case 0xD2: JNC_Adr(); break;
case 0xD3: OUT(); break;
case 0xD4: CNC_Adr(); break;
case 0xD5: PUSH_D(); break;
case 0xD6: SUI_d8(); break;
case 0xD7: RST_2(); break;
case 0xD8: RC(); break;
case 0xD9: break;
case 0xDA: JC_Adr(); break;
case 0xDB: IN(); break;
case 0xDC: CC_Adr(); break;
case 0xDD: break;
case 0xDE: SBI_d8(); break;
case 0xDF: RST_3(); break;
case 0xE0: RPO(); break;
case 0xE1: POP_H(); break;
case 0xE2: JPO_Adr(); break;
case 0xE3: XTHL(); break;
case 0xE4: CPO_Adr(); break;
case 0xE5: PUSH_H(); break;
case 0xE6: ANI_d8(); break;
case 0xE7: RST_4(); break;
case 0xE8: RPE(); break;
case 0xE9: PCHL(); break;
case 0xEA: JPE_Adr(); break;
```

```
        case 0xEB: XCHG(); break;
        case 0xEC: CPE_Adr(); break;
        case 0xED: break;
        case 0xEE: XRI_d8(); break;
        case 0xEF: RST_5(); break;
        case 0xF0: RP(); break;
        case 0xF1: POP_PSW(); break;
        case 0xF2: JP_Adr(); break;
        case 0xF3: DI(); break;
        case 0xF4: CP_Adr(); break;
        case 0xF5: PUSH_PSW(); break;
        case 0xF6: ORI_d8(); break;
        case 0xF7: RST_6(); break;
        case 0xF8: RM(); break;
        case 0xF9: SPHL(); break;
        case 0xFA: JM_Adr(); break;
        case 0xFB: EI(); break;
        case 0xFC: CM_Adr(); break;
        case 0xFD: break;
        case 0xFE: CPI_d8(); break;
        case 0xFF: RST_7(); break;
    }  /* end of switch case */
}
run_all()
{
  unsigned char rst;
  int test;

  addr_define();
  clsmess();
  cursor(23,26);
  printf("Press ESCAPE keys to break !!!");
  while(p_count < p_break) {
```

```
        current_addr();
        fatch();
        ++p_count;
        disp_pc();
        rst = inportb(PORT_C);
        rst&=0x28;
        if(rst!=0x28) return 1;
        test=kbhit();
        if(test!=0) break;
    }   /* end of loop while */
}   /* end of function step */
```

OTHER_ME.C

```
void  menu_testport(),menu_type_port();
int   norm_key,port_type;

/*********************************
*          MENU OTHER           *
*********************************/
menu_other()
{
  int choice;
/* now, activate as needed */
arrow_choice = 0;key_geted = 0;
pulldown(2);
 while((choice=get_resp(2)) != -1) {
  switch(choice)  {
    case 0:
      restore_video(2);
      modi_reg();
      arrow_get = 10;
      return;
    case 1:
      menu_testport();
      break;
  }
 }
 restore_video(2);
}

/***  Modify the contain of any ragister  ***/
modi_reg()
{
   int   old_f,out_f=0;
   char answer,temp_reg[5];
```

```
window_(2);window_cls(2);
window_xy(2, 1, 4);
window_puts(2," Type the first letter of register ? ",A_INVERSE);
answer=window_getche(2);
switch(tolower(answer)) {
  case 'p':
    window_xy(2, 1, 0);
    window_cleol(2);
    window_xy(2, 1, 8);
    window_puts(2," Enter new value of PC : ",A_INVERSE);
    window_gets(2, temp_reg);
    if(temp_reg[0] == -1) { deactivate_win(2); return; }
    p_count = ch_to_hex(temp_reg);
    break;
  case 's':
    window_xy(2, 1, 0);
    window_cleol(2);
    window_xy(2, 1, 8);
    window_puts(2," Enter new value of SP : ",A_INVERSE);
    window_gets(2, temp_reg);
    if(temp_reg[0] == -1) { deactivate_win(2); return; }
    s_point = ch_to_hex(temp_reg);
    break;
  case 'a':
    window_xy(2, 1, 0);
    window_cleol(2);
    window_xy(2, 1, 9);
    window_puts(2," Enter new value of ACC : ",A_INVERSE);
    window_gets(2, temp_reg);
    if(temp_reg[0] == -1) { deactivate_win(2); return; }
    reg_a = ch_to_hex(temp_reg);
    break;
```

```
case''b':
  window_xy(2, 1, 0);
  window_cleol(2);
  window_xy(2, 1, 9);
  window_puts(2,"  Enter new value of B : ",A_INVERSE);
  window_gets(2, temp_reg);
  if(temp_reg[0] == -1) { deactivate_win(2); return; }
  reg_b = ch_to_hex(temp_reg);
  break;
case 'c':
  window_xy(2, 1, 0);
  window_cleol(2);
  window_xy(2, 1, 9);
  window_puts(2,"  Enter new value of C : ",A_INVERSE);
  window_gets(2, temp_reg);
  if(temp_reg[0] == -1) { deactivate_win(2); return; }
  reg_c = ch_to_hex(temp_reg);
  break;
case 'd':
  window_xy(2, 1, 0);
  window_cleol(2);
  window_xy(2, 1, 9);
  window_puts(2,"  Enter new value of D : ",A_INVERSE);
  window_gets(2, temp_reg);
  if(temp_reg[0] == -1) { deactivate_win(2); return; }
  reg_d = ch_to_hex(temp_reg);
  break;
case 'e':
  window_xy(2, 1, 0);
  window_cleol(2);
  window_xy(2, 1, 9);
  window_puts(2,"  Enter new value of E : ",A_INVERSE);
  window_gets(2, temp_reg);
```

```
            if(temp_reg[0] == -1) { deactivate_win(2); return; }
            reg_e = ch_to_hex(temp_reg);
            break;
          case 'h':
            window_xy(2, 1, 0);
            window_cleol(2);
            window_xy(2, 1, 9);
            window_puts(2,"  Enter new value of H : ",A_INVERSE);
            window_gets(2, temp_reg);
            if(temp_reg[0] == -1) { deactivate_win(2); return; }
            reg_h = ch_to_hex(temp_reg);
            break;
          case 'l':
            window_xy(2, 1, 0);
            window_cleol(2);
            window_xy(2, 1, 9);
            window_puts(2,"  Enter new value of L : ",A_INVERSE);
            window_gets(2, temp_reg);
            if(temp_reg[0] == -1) { deactivate_win(2); return; }
            reg_l = ch_to_hex(temp_reg);
            break;
          case 'f':
            window_xy(2, 1, 0);
            window_cleol(2);
            window_xy(2, 1, 9);
            window_puts(2," Enter new value of PSW : ",A_INVERSE);
            window_gets(2, temp_reg);
            if(temp_reg[0] == -1) { deactivate_win(2); return; }
            old_f = psw;
            psw = ch_to_hex(temp_reg);
    switch(psw) {
     case 0x00: sign_f=0;zero_f=0;aux_f=0;parity_f=0;carry_f=0;break;
     case 0x01: sign_f=0;zero_f=0;aux_f=0;parity_f=0;carry_f=1;break;
```

```
 case 0x04: sign_f=0;zero_f=0;aux_f=0;parity_f=1;carry_f=0;break;
 case 0x05: sign_f=0;zero_f=0;aux_f=0;parity_f=1;carry_f=1;break;
 case 0x10: sign_f=0;zero_f=0;aux_f=1;parity_f=0;carry_f=0;break;
 case 0x11: sign_f=0;zero_f=0;aux_f=1;parity_f=0;carry_f=1;break;
 case 0x14: sign_f=0;zero_f=0;aux_f=1;parity_f=1;carry_f=0;break;
 case 0x15: sign_f=0;zero_f=0;aux_f=1;parity_f=1;carry_f=1;break;
 case 0x40: sign_f=0;zero_f=1;aux_f=0;parity_f=0;carry_f=0;break;
 case 0x41: sign_f=0;zero_f=1;aux_f=0;parity_f=0;carry_f=1;break;
 case 0x44: sign_f=0;zero_f=1;aux_f=0;parity_f=1;carry_f=0;break;
 case 0x45: sign_f=0;zero_f=1;aux_f=0;parity_f=1;carry_f=1;break;
 case 0x50: sign_f=0;zero_f=1;aux_f=1;parity_f=0;carry_f=0;break;
 case 0x51: sign_f=0;zero_f=1;aux_f=1;parity_f=0;carry_f=1;break;
 case 0x54: sign_f=0;zero_f=1;aux_f=1;parity_f=1;carry_f=0;break;
 case 0x55: sign_f=0;zero_f=1;aux_f=1;parity_f=1;carry_f=1;break;
 case 0x80: sign_f=1;zero_f=0;aux_f=0;parity_f=0;carry_f=0;break;
 case 0x81: sign_f=1;zero_f=0;aux_f=0;parity_f=0;carry_f=1;break;
 case 0x84: sign_f=1;zero_f=0;aux_f=0;parity_f=1;carry_f=0;break;
 case 0x85: sign_f=1;zero_f=0;aux_f=0;parity_f=1;carry_f=1;break;
 case 0x90: sign_f=1;zero_f=0;aux_f=1;parity_f=0;carry_f=0;break;
 case 0x91: sign_f=1;zero_f=0;aux_f=1;parity_f=0;carry_f=1;break;
 case 0x94: sign_f=1;zero_f=0;aux_f=1;parity_f=1;carry_f=0;break;
 case 0x95: sign_f=1;zero_f=0;aux_f=1;parity_f=1;carry_f=1;break;
 case 0xC0: sign_f=1;zero_f=1;aux_f=0;parity_f=0;carry_f=0;break;
 case 0xC1: sign_f=1;zero_f=1;aux_f=0;parity_f=0;carry_f=1;break;
 case 0xC4: sign_f=1;zero_f=1;aux_f=0;parity_f=1;carry_f=0;break;
 case 0xC5: sign_f=1;zero_f=1;aux_f=0;parity_f=1;carry_f=1;break;
 case 0xD0: sign_f=1;zero_f=1;aux_f=1;parity_f=0;carry_f=0;break;
 case 0xD1: sign_f=1;zero_f=1;aux_f=1;parity_f=0;carry_f=1;break;
 case 0xD4: sign_f=1;zero_f=1;aux_f=1;parity_f=1;carry_f=0;break;
 case 0xD5: sign_f=1;zero_f=1;aux_f=1;parity_f=1;carry_f=1;break;
 default :  psw = old_f; out_f = 1; break;
}
break;
```

```
    default:
    deactivate_win(2);
    hide_cursor();
    return;
    }
    deactivate_win(2);
      if(out_f==1) {
         hide_cursor();
         putchar(BELL);
         clsmess();
         cursor(23,23);
         printf("!!! Can not modify or Invalid data !!!");
         get_key();
         hide_cursor();
         clsmess();
      }
        disp_pc();disp_sp();disp_acc();disp_bc();
        disp_de();disp_hl();disp_psw();DISP_ALL_F;
        hide_cursor();
}
ch_to_hex(c)
char *c;
{
   char *chklen,temp;
   int i,result,ans,len=0;
   chklen = c;
   while(*chklen) {
     len++;
     chklen++;
   }
   if(len>4) return 0;
   for(i=0;i<len;i++,c++) {
     temp = *c;
```

```
switch(tolower(temp)) {
  case '0':
    result = 0;
    break;
  case '1':
    result = 1;
    break;
  case '2':
    result = 2;
    break;
  case '3':
    result = 3;
    break;
  case '4':
    result = 4;
    break;
  case '5':
    result = 5;
    break;
  case '6':
    result = 6;
    break;
  case '7':
    result = 7;
    break;
  case '8':
    result = 8;
    break;
  case '9':
    result = 9;
    break;
  case 'a':
    result = 0xa;
```

```
        break;
      case 'b':
        result = 0xb;
        break;
      case 'c':
        result = 0xc;
        break;
      case 'd':
        result = 0xd;
        break;
      case 'e':
        result = 0xe;
        break;
      case 'f':
        result = 0xf;
        break;
      default:
        putchar(BELL);
        ans = 0;
      return ans;
    }
    if(len==1)  return result;
    else
    if(len==2) {
     if(i==0) { result <<= 4; ans = result; }
     else if(i==1) ans |= result;
    }
    else if(len==3) {
     if(i==0) { result <<= 8; ans = result; }
     else if(i==1) { result <<= 4; ans |= result; }
     else if(i==2) ans |= result;
    }
    else if(len==4) {
```

```
        if(i==0) { result <<= 12; ans = result; }
        else if(i==1) { result <<= 8; ans |= result; }
        else if(i==2) { result <<= 4; ans |= result; }
        else if(i==3) ans |= result;
    }
    } /* end of loop for */
  return ans;
}

/*** Program test port on emu8085 card ***/
void menu_testport()
{
  int item;
arrow_item = 0;key_recive= 0;
pulldown(4);
while((item=get_resp_sub(4)) != -1) {
    switch(item) {
      case 0:
          menu_type_port();
          if(norm_key==-1 || norm_key==-10) break;
          port_io(PORT_A, 'A');
          break;
      case 1:
          menu_type_port();
          if(norm_key==-1 || norm_key==-10) break;
          port_io(PORT_B, 'B');
          break;
      case 2:
          menu_type_port();
          if(norm_key==-1 || norm_key==-10) break;
          port_io(PORT_C, 'C');
          break;
```

```
        case 3:
            menu_type_port();
            if(norm_key==-1 || norm_key==-10) break;
            port_io(PORT_D, 'D');
            break;
        case 4:
            menu_type_port();
            if(norm_key==-1 || norm_key==-10) break;
            port_io(PORT_E, 'E');
            break;
        case 5:
            menu_type_port();
            if(norm_key==-1 || norm_key==-10) break;
            port_io(PORT_F, 'F');
            break;
        case 6:
            menu_type_port();
            if(norm_key==-1 || norm_key==-10) break;
            port_io(PORT_G, 'G');
            break;
        case 7:
            menu_type_port();
            if(norm_key==-1 || norm_key==-10) break;
            port_io(PORT_H, 'H');
            break;
        default:
            break;
    }
  }
    restore_video(4);
}
void menu_type_port()
{
```

```
int choice;
arrow_item1= 0;key_recive1= 0;norm_key=0;
pulldown(5);
choice=get_resp_sub1(5);
  switch(choice)  {
    case 0:
      port_type = 0;
      break;
    case 1:
      port_type = 1;
      break;
    case -1:
      norm_key=choice;
      break;
    case -10:
      norm_key=choice;
      break;
  }
    restore_video(5);
}
port_io(port_id, c)
int port_id;
char c;
{
   int data;
   unsigned char data_input;
   char ary[4];

    switch(port_type) {
      case 0:
       data_input = inportb(port_id);
       clsmess();cursor(23,18);
       printf("Now ! input data from port %xh (PORT %c) is ",
```

```
        port_id,c);
        hex_up8(data_input);printf("h ");
        hide_cursor();
        break;
      case 1:
        do {
          window_(1);window_cls(1);
          window_puts(1," Enter number in hex (00h-FFh) ? ",A_INVERSE);
          window_gets(1, ary);
          if(ary[0] == -1 || ary[0] == '\0') {
             deactivate_win(1);
             hide_cursor();
             return;
          }
          data = ch_to_hex(ary);
          if(data > 0xff) {
            putchar(BELL);
            clsmess();cursor(23,22);
            printf("     !!! data out of range !!!      ");
            hide_cursor();
            delay(1500);
            clsmess();
          }
        } while(data > 0xff);
        deactivate_win(1);
        clsmess();cursor(23,17);
        printf("Now ! out data [%2xh] into port no. %xh (PORT %c) ",
        data,port_id,c);
        hide_cursor();
        outportb(port_id,data);
        break;
    }
} /*****  End of menu debug  ******/
```

QUIT_MEN.C

```
/**********************************
*          MENU QUIT              *
**********************************/
menu_quit()
{
  int choice;
  char ch;
/* now, activate as needed */
arrow_choice = 0;key_geted = 0;
pulldown(3);
while((choice=get_resp(3)) != -1) {
  switch(choice)  {
    case 0:
      disp_version();
      break;
    case 1:
      window_(3);window_cls(3);
      window_xy(3, 0, 1);
      window_puts(3,"  Are you sure exit to dos [y/n] ?  ",A_INTENSE);
      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;
  } /* end switch */
} /* end while */
  restore_video(3);
}
```

```
#define   MES_LINE1 "8 0 8 5   E M U L A T O R"
#define   MES_LINE2 "Version 1.20"
#define   MES_LINE3 "By"
#define   MES_LINE4 "1. Mr. Prarinya Ekapho        "
#define   MES_LINE5 "2. Mr. Kitichai Thatreetong   "
#define   MES_LINE6 "3. Mr. Somchai  Khawladdakorn"
#define   MES_LINE7 "          << F10 to open menu >>          "

disp_version()          /*  display version of this program */
{
  window_(0); window_cls(0);
  window_xy(0,1,(40-strlen(MES_LINE1))/2);
  window_puts(0,MES_LINE1,A_INVERSE);
  window_xy(0,2,(40-strlen(MES_LINE2))/2);
  window_puts(0,MES_LINE2,A_INVERSE);
  window_xy(0,3,(40-strlen(MES_LINE3))/2);
  window_puts(0,MES_LINE3,A_INVERSE);
  window_xy(0,4,(40-strlen(MES_LINE4))/2);
  window_puts(0,MES_LINE4,A_INVERSE);
  window_xy(0,5,(40-strlen(MES_LINE5))/2);
  window_puts(0,MES_LINE5,A_INVERSE);
  window_xy(0,6,(40-strlen(MES_LINE6))/2);
  window_puts(0,MES_LINE6,A_INVERSE);
  window_xy(0,8,(40-strlen(MES_LINE7))/2);
  window_puts(0,MES_LINE7,A_INTENSE);
  hide_cursor();
  get_key();
  deactivate_win(0);
}
/****** end menu quit *****/
```

OPCODE1.C

```
/**** NOP ****/
NOP()
{
}
/**** LXI B,data16 ****/
LXI_B_d16()
{
  ++p_count; reg_c = *(pcram+p_count);
  ++p_count; reg_b = *(pcram+p_count);
  disp_bc();
}
/**** STAX B ****/
STAX_B()
{
  unsigned char r1,r2;
  r1 = reg_b; r2 = reg_c;
  staxout(r1,r2);
}
/**** STAX D ****/
STAX_D()
{
  unsigned char r1,r2;
  r1 = reg_d; r2 = reg_e;
  staxout(r1,r2);
}

/**** INX B ****/
INX_B()
{
  int xc;
  xc = reg_c; ++reg_c; ++xc;
  reg_b += (xc > 0xFF) ? 1 : 0;
  disp_bc();
```

```
}
/**** INX D ****/
INX_D()
{
  int xe;
  xe = reg_e; ++reg_e; ++xe;
  reg_d += (xe > 0xFF) ? 1 : 0;
  disp_de();
}
/**** INX H ****/
INX_H()
{
  int xl;
  xl = reg_l; ++reg_l; ++xl;
  reg_h += (xl > 0xFF) ? 1 : 0;
  disp_hl();
}
/**** INX SP ****/
INX_SP()
{
  ++s_point;
  disp_sp();
}
/**** DCX B ****/
DCX_B()
{
  int bx = 0x100;
  bx |= reg_c; --reg_c; --bx;
  reg_b -= (bx < 0x100) ? 1 : 0;
  disp_bc();
}
/**** DCX D ****/
DCX_D()
```

```
/**** INR C ****/
INR_C()
{
  int old,cx;
  old = cx = reg_c; ++reg_c; ++cx;
  disp_bc(); inr_flag(cx,old);
}
/**** INR D ****/
INR_D()
{
  int old,dx;
  old = dx = reg_d; ++reg_d; ++dx;
  disp_de(); inr_flag(dx,old);
}
/**** INR E ****/
INR_E()
{
  int old,ex;
  old = ex = reg_e; ++reg_e; ++ex;
  disp_de(); inr_flag(ex,old);
}
/**** INR H ****/
INR_H()
{
  int old,hx;
  old = hx = reg_h; ++reg_h; ++hx;
  disp_hl(); inr_flag(hx,old);
}
/**** INR L ****/
INR_L()
{
  int old,lx;
  old = lx = reg_l; ++reg_l; ++lx;
```

```
  disp_hl(); inr_flag(lx,old);
}
/**** INR M ****/
INR_M()
{
  int old,m;
  unsigned char in_data;
  old = m = in_data = RD_r_M();
  ++m; ++in_data;
  WR_r_M(in_data);
  inr_flag(m,old);
}
/**** INR A ****/
INR_A()
{
  int old,acc16;
  old = acc16 = reg_a; ++acc16; ++reg_a;
  disp_acc(); inr_flag(acc16,old);
}
dcr_flag(int ans,int prv)
{
  sign(ans); zero(ans);
  aux_sub(prv,1); parity(ans);
  flag_to_psw();disp_psw();DISP_ALL_F;
}
/**** DCR B ****/
DCR_B()
{
  int old,bx;
  old = bx = reg_b; --reg_b; --bx;
  disp_bc(); dcr_flag(bx,old);
}
/**** DCR C ****/
```

```
DCR_C()
{
  int old,cx;
  old = cx = reg_c; --reg_c; --cx;
  disp_bc(); dcr_flag(cx,old);
}
/**** DCR D ****/
DCR_D()
{
  int old,dx;
  old = dx = reg_d; --reg_d; --dx;
  disp_de(); dcr_flag(dx,old);
}
/**** DCR E ****/
DCR_E()
{
  int old,ex;
  old = ex = reg_e; --reg_e; --ex;
  disp_de(); dcr_flag(ex,old);
}
/**** DCR H ****/
DCR_H()
{
  int old,hx;
  old = hx = reg_h; --reg_h; --hx;
  disp_hl(); dcr_flag(hx,old);
}
/**** DCR L ****/
DCR_L()
{
  int old,lx;
  old = lx = reg_l; --reg_l; --lx;
  disp_hl(); dcr_flag(lx,old);
```

```
}
/**** DCR M ****/
DCR_M()
{
  int old,m;
  unsigned char in_data;
  old = m = in_data = RD_r_M();
  m += 0xFF; --in_data;
  WR_r_M(in_data);
  dcr_flag(m,old);
}
/**** DCR A ****/
DCR_A()
{
  int old,acc16;
  old = acc16 = reg_a; --acc16; --reg_a;
  disp_acc(); dcr_flag(acc16,old);
}
/**** MVI B,d8 ****/
MVI_B_d8()
{
  ++p_count;
  reg_b = *(pcram+p_count);
  disp_bc();
}
/**** MVI C,d8 ****/
MVI_C_d8()
{
  ++p_count;
  reg_c = *(pcram+p_count);
  disp_bc();
}
/**** MVI D,d8 ****/
```

```
MVI_D_d8()
{
  ++p_count;
  reg_d = *(pcram+p_count);
  disp_de();
}
/**** MVI E,d8 ****/
MVI_E_d8()
{
  ++p_count;
  reg_e = *(pcram+p_count);
  disp_de();
}
/**** MVI H,d8 ****/
MVI_H_d8()
{
  ++p_count;
  reg_h = *(pcram+p_count);
  disp_hl();
}
/**** MVI L,d8 ****/
MVI_L_d8()
{
  ++p_count;
  reg_l = *(pcram+p_count);
  disp_hl();
}
/**** MVI M,d8 ****/
MVI_M_d8()
{
  unsigned char d8;
  ++p_count;
  d8 = *(pcram+p_count);
```

```
  WR_r_M(d8);
}
/**** MVI A,d8 ****/
MVI_A_d8()
{
  ++p_count;
  reg_a = *(pcram+p_count);
  disp_acc();
}
dad_rotate_flag()
{
  flag_to_psw(); disp_psw();
  DISP_ALL_F;
}
/**** RLC ****/
RLC()
{
  unsigned char temp;
  temp = reg_a;
  reg_a <<= 1;temp >>= 7;
  reg_a |= temp;
  carry_f = (temp == 1) ? 1 : 0;
  dad_rotate_flag();
  disp_acc();
}
/**** RRC ****/
RRC()
{
  unsigned char temp;
  temp = reg_a;
  reg_a >>= 1;temp <<= 7;
  reg_a |= temp;
  carry_f = (temp == 0x80) ? 1 : 0;
```

```
  dad_rotate_flag();
  disp_acc();
}
/**** RAL ****/
RAL()
{
  int temp;
  temp = reg_a;
  temp <<= 1;reg_a <<= 1;
  reg_a |= (carry_f == 1) ? 1 : 0;
  temp &= 0x100;
  carry_f = (temp == 0x100) ? 1 : 0;
  dad_rotate_flag();
  disp_acc();
}
/**** RAR ****/
RAR()
{
  int temp;
  temp = reg_a;
  temp <<= 7;reg_a >>= 1;
  reg_a |= (carry_f == 1) ? 0x80 : 0;
  temp &= 0x80;
  carry_f = (temp == 0x80) ? 1 : 0;
  dad_rotate_flag();
  disp_acc();
}
/**** LXI D,data16 ****/
LXI_D_d16()
{
  ++p_count; reg_e = *(pcram+p_count);
  ++p_count; reg_d = *(pcram+p_count);
  disp_de();
```

```
/**** LXI H,data16 ****/
LXI_H_d16()
{
  ++p_count; reg_l = *(pcram+p_count);
  ++p_count; reg_h = *(pcram+p_count);
  disp_hl();
}
/**** LXI SP,data16 ****/
LXI_SP_d16()
{
  int temp; s_point=0;
  ++p_count; byte2 = *(pcram+p_count); s_point = byte2;
  ++p_count; byte3 = *(pcram+p_count); temp = byte3;
  s_point |= (temp<<=8);
  disp_sp();
}
/**** LDAX B ****/
LDAX_B()
{
  unsigned char r1,r2;
  r1 = reg_b; r2 = reg_c;
  ldaxout(r1,r2);
  disp_acc();
}
/**** LDAX D ****/
LDAX_D()
{
  unsigned char r1,r2;
  r1 = reg_d; r2 = reg_e;
  ldaxout(r1,r2);
  disp_acc();
}
```

```
/**** DAD B ****/
DAD_B()
{
  int temp;
  long int hl,bc;
  hl = reg_h; hl <<= 8; hl |= reg_l;
  bc = reg_b; bc <<= 8; bc |= reg_c;
  hl += bc; cy_16(hl);temp = hl;
  reg_h = (temp>>8); reg_l = (temp & 0xff);
  dad_rotate_flag();
  disp_hl();
}
/**** DAD D ****/
DAD_D()
{
  int temp;
  long int hl,de;
  hl = reg_h; hl <<= 8; hl |= reg_l;
  de = reg_d; de <<= 8; de |= reg_e;
  hl += de; cy_16(hl);temp = hl;
  reg_h = (temp>>8); reg_l = (temp & 0xff);
  dad_rotate_flag();
  disp_hl();
}
/**** DAD H ****/
DAD_H()
{
  int temp;
  long int hl;
  hl = reg_h; hl <<= 8; hl |= reg_l;
  hl += hl; cy_16(hl);temp = hl;
  reg_h = (temp>>8); reg_l = (temp & 0xff);
  dad_rotate_flag();
```

```
    disp_hl();
  }
  /**** DAD SP ****/
  DAD_SP()
  {
    int temp;
    long int hl;
    hl = reg_h; hl <<= 8; hl |= reg_l;
    hl += s_point; cy_16(hl); temp = hl;
    reg_h = (temp)>>8); reg_l = temp;
    dad_rotate_flag();
    disp_hl();
  }
  /**** RIM ****/
  RIM()
  {
    reg_a = reg_im;
    disp_acc();
  }
  /**** SIM ****/
  SIM()
  {
    reg_im = reg_a;
  }
  /**** SHLD Address ****/
  SHLD_Adr()
  {
    ++p_count; byte2 = *(pcram+p_count);
    ++p_count; byte3 = *(pcram+p_count);
    WR_M(reg_l);
    byte3 += (byte2 == 0xFF) ? 1 : 0;
    ++byte2;
    WR_M(reg_h);
```

```
}
/**** LHLD Address ****/
LHLD_Adr()
{
  ++p_count; byte2 = *(pcram+p_count);
  ++p_count; byte3 = *(pcram+p_count);
  reg_l = RD_M();
  byte3 += (byte2 == 0xFF) ? 1 : 0;
  ++byte2;
  reg_h = RD_M();
  disp_hl();
}
/**** DAA ****/
DAA()
{
  int chk;
  unsigned char temp;
  temp = reg_a; temp &= 0x0F;
  temp += (temp > 9 || aux_f == 1) ? 6 : 0;
  aux_f = (temp > 0x0F) ? 1 : 0;
  reg_a &= 0xF0; reg_a >>= 4;
  reg_a += (reg_a > 9 || carry_f == 1) ? 6 : 0;
  carry_f = (reg_a > 0x0F) ? 1 : 0;
  reg_a <<= 4; reg_a |= temp; chk = reg_a;
  sign(chk); zero(chk); parity(chk);
  flag_to_psw(); disp_acc();
  disp_psw(); DISP_ALL_F;
}
/**** CMA ****/
CMA()
{
  reg_a = ~reg_a;
  disp_acc();
```

```
}
/**** CMC ****/
CMC()
{
  carry_f = (carry_f == 0) ? 1 : 0;
  if(carry_f == 0) psw &= 0xFE;
   else   psw |= 0x01;
  disp_psw();DISP_C;
}
/**** STA Address ****/
STA_Adr()
{
  ++p_count; byte2 = *(pcram+p_count);
  ++p_count; byte3 = *(pcram+p_count);
  WR_M(reg_a);
}
/**** LDA Address ****/
LDA_Adr()
{
  ++p_count; byte2 = *(pcram+p_count);
  ++p_count; byte3 = *(pcram+p_count);
  RD_M();
  disp_acc();
}
/**** STC ****/
STC()
{
  carry_f = 1;
  psw |= 0x01;
  disp_psw();DISP_C;
}
```

```
/**** JMP Address ****/
JMP_Adr()
{
  ++p_count; byte2 = *(pcram+p_count);
  ++p_count; byte3 = *(pcram+p_count);
  p_count = byte3; p_count<<=8; p_count+=byte2;
  --p_count;
  disp_pc();
}
/**** JNZ Address ****/
JNZ_Adr()
{
  if(zero_f == 0) JMP_Adr();
  else p_count+=2;
}
/**** JZ Address ****/
JZ_Adr()
{
  if(zero_f == 1) JMP_Adr();
  else p_count+=2;
}
/**** JC Address ****/
JC_Adr()
{
  if(carry_f == 1) JMP_Adr();
  else p_count+=2;
}
/**** JNC Address ****/
JNC_Adr()
{
  if(carry_f == 0) JMP_Adr();
  else p_count+=2;
}
```

```
/**** JP Address ****/
JP_Adr()
{
  if(sign_f == 0) JMP_Adr();
  else p_count+=2;
}
/**** JM Address ****/
JM_Adr()
{
  if(sign_f == 1) JMP_Adr();
  else p_count+=2;
}
/**** JPE Address ****/
JPE_Adr()
{
  if(parity_f == 1) JMP_Adr();
  else p_count+=2;
}
/**** JPO Address ****/
JPO_Adr()
{
  if(parity_f == 0) JMP_Adr();
  else p_count+=2;
}

/**** CALL Address ****/
CALL_Adr()
{
  call_sub();
  JMP_Adr();
  disp_sp();
}
```

```
/**** CC Address ****/
CC_Adr()
{
  if(carry_f == 1) CALL_Adr();
  else p_count+=2;
}
/**** CNC Address ****/
CNC_Adr()
{
  if(carry_f == 0) CALL_Adr();
  else p_count+=2;
}
/**** CZ Address ****/
CZ_Adr()
{
  if(zero_f == 1) CALL_Adr();
  else p_count+=2;
}
/**** CNZ Address ****/
CNZ_Adr()
{
  if(zero_f == 0) CALL_Adr();
  else p_count+=2;
}
/**** CP Address ****/
CP_Adr()
{
  if(sign_f == 0) CALL_Adr();
  else p_count+=2;
}
/**** CM Address ****/
CM_Adr()
{
```

```
    if(sign_f == 1) CALL_Adr();
    else p_count+=2;
  }
  /**** CPE Address ****/
  CPE_Adr()
  {
    if(parity_f == 1) CALL_Adr();
    else p_count+=2;
  }
  /**** CPO Address ****/
  CPO_Adr()
  {
    if(parity_f == 0) CALL_Adr();
    else p_count+=2;
  }
  /**** RET ****/
  RET()
  {
    unsigned char ph,pl;
    pl = pop_rp();
    pl = *(pcram+s_point);
    ++s_point;
    ph = pop_rp();
    ph = *(pcram+s_point);
    p_count = ph; p_count <<= 8; p_count |= pl;
    ++s_point;
    --p_count;
    disp_pc();disp_sp();
  }
  /**** RC ****/
  RC()
  {
    if(carry_f == 1) RET();
```

```
}
/**** RNC ****/
RNC()
{
  if(carry_f == 0) RET();
}
/**** RZ ****/
RZ()
{
  if(zero_f == 1) RET();
}
/**** RNZ ****/
RNZ()
{
  if(zero_f == 0) RET();
}
/**** RM ****/
RM()
{
  if(sign_f == 1) RET();
}
/**** RP ****/
RP()
{
  if(sign_f == 0) RET();
}
/**** RPE ****/
RPE()
{
  if(parity_f == 1) RET();
}
/**** RPO ****/
RPO()
```

```
{
  if(parity_f == 0) RET();
}
/**** XTHL ****/
XTHL()
{
  unsigned char temp;
  temp = reg_l;
  reg_l = pop_rp();
  reg_l = *(pcram+s_point);
  push_rp(temp);
  temp = reg_h;
  ++s_point;
  reg_h = pop_rp();
  reg_h = *(pcram+s_point);
  push_rp(temp);
  ++s_point;
  disp_sp();disp_hl();
}
/**** PUSH B ****/
PUSH_B()
{
  push_bdhp(reg_b,reg_c);
  disp_sp();
}
/**** PUSH D ****/
PUSH_D()
{
  push_bdhp(reg_d,reg_e);
  disp_sp();
}
/**** PUSH H ****/
PUSH_H()
```

```
{
  push_bdhp(reg_h,reg_l);
  disp_sp();
}
/**** PUSH PSW ****/
PUSH_PSW()
{
  push_bdhp(psw,reg_a);
  disp_sp();
}
/**** POP B ****/
POP_B()
{
  reg_c = pop_rp();
  reg_c = *(pcram+s_point);
  ++s_point;
  reg_b = pop_rp();
  reg_b = *(pcram+s_point);
  ++s_point;
  disp_sp(); disp_bc();
}
/**** POP D ****/
POP_D()
{
  reg_e = pop_rp();
  reg_e = *(pcram+s_point);
  ++s_point;
  reg_d = pop_rp();
  reg_d = *(pcram+s_point);
  ++s_point;
  disp_sp(); disp_de();
}
/**** POP H ****/
```

```
POP_H()
{
  reg_l = pop_rp();
  reg_l = *(pcram+s_point);
  ++s_point;
  reg_h = pop_rp();
  reg_h = *(pcram+s_point);
  ++s_point;
  disp_sp(); disp_hl();
}
/**** POP PSW ****/
POP_PSW()
{
  reg_a = pop_rp();
  reg_a = *(pcram+s_point);
  ++s_point;
  psw = pop_rp();
  psw = *(pcram+s_point);
  psw_to_flag();
  ++s_point;
  disp_sp(); disp_acc();
  disp_psw(); DISP_ALL_F;
}
/**** SPHL ****/
SPHL()
{
 s_point = reg_h; s_point <<= 8;
 s_point |= reg_l;
 disp_sp();
}
/**** PCHL ****/
PCHL()
{
```

```
 p_count = reg_h; p_count <<= 8;
 p_count |= reg_l;
 --p_count;
 disp_pc();
}
/**** XCHG ****/
XCHG()
{
  unsigned char temp;
  temp = reg_h;
  reg_h = reg_d; reg_d = temp;
  temp = reg_l;
  reg_l = reg_e; reg_e = temp;
  disp_hl(); disp_de();
}
/**** ADI data8 ****/
ADI_d8()
{
  unsigned char m;
  int old,ans;
  ++p_count;
  old = reg_a; m = *(pcram+p_count);
  reg_a += m; ans = reg_a;
  add_flag(ans,old,m);
}
/**** ACI data8 ****/
ACI_d8()
{
  unsigned char m;
  int old,ans;
  ++p_count; old = reg_a;
  m = *(pcram+p_count); reg_a += m;
  reg_a += (carry_f == 1) ? 1 : 0;
```

```
    ans = reg_a;
    adc_flag(ans,old,m);
  }
  /**** SUI data8 ****/
  SUI_d8()
  {
    int old,ans;
    unsigned char m;
    ++p_count; m = *(pcram+p_count);
    old = reg_a; reg_a += (~m+1);
    ans = reg_a;
    sub_flag(ans,old,m);
  }
  /**** SBI data8 ****/
  SBI_d8()
  {
    int old,ans;
    unsigned char m;
    ++p_count; m = *(pcram+p_count);
    old = reg_a; reg_a += (~m+1);
    reg_a += (carry_f == 1) ? 0xFF : 0;
    ans = reg_a;
    sbb_flag(ans,old,m);
  }
  /**** ANI data8 ****/
  ANI_d8()
  {
    int ans;
    unsigned char m;
    ++p_count; m = *(pcram+p_count);
    reg_a &= m; ans = reg_a;
    ana_flag(ans);
  }
```

```
/**** XRI data8 ****/
XRI_d8()
{
  int ans;
  unsigned char m;
  ++p_count; m = *(pcram+p_count);
  reg_a ^= m; ans = reg_a;
  or_xor_flag(ans);
}
/**** ORI data8 ****/
ORI_d8()
{
  int ans;
  unsigned char m;
  ++p_count; m = *(pcram+p_count);
  reg_a |= m; ans = reg_a;
  or_xor_flag(ans);
}
/**** CPI data8 ****/
CPI_d8()
{
  int old,ans;
  unsigned char m;
  ++p_count; m = *(pcram+p_count);
  old = reg_a; ans = reg_a +(~m+1);
  cmp_flag(ans,old,m);
}
/**** RST 0 ****/
RST_0()
{
  call_rst();
  p_count = 0x0000; --p_count;
  disp_pc();disp_sp();
```

```
}
/**** RST 1 ****/
RST_1()
{
  call_rst();
  p_count = 0x0008; --p_count;
  disp_pc();disp_sp();
}
/**** RST 2 ****/
RST_2()
{
  call_rst();
  p_count = 0x0010; --p_count;
  disp_pc();disp_sp();
}
/**** RST 3 ****/
RST_3()
{
  call_rst();
  p_count = 0x0018; --p_count;
  disp_pc();disp_sp();
}
/**** RST 4 ****/
RST_4()
{
  call_rst();
  p_count = 0x0020; --p_count;
  disp_pc();disp_sp();
}
/**** RST 5 ****/
RST_5()
{
  call_rst();
```

```
  p_count = 0x0028; --p_count;
  disp_pc();disp_sp();
}
/**** RST 6 ****/
RST_6()
{
  call_rst();
  p_count = 0x0030; --p_count;
  disp_pc();disp_sp();
}
/**** RST 7 ****/
RST_7()
{
  call_rst();
  p_count = 0x0038; --p_count;
  disp_pc();disp_sp();
}
/**** IN PORT ****/
IN()
{
  unsigned char no_port;
  ++p_count;
  no_port = *(pcram+p_count);
  outportb(PORT_A,no_port);
  outportb(PORT_H,0);
  outportb(PORT_C,15);
  outportb(PORT_D,14);
  outportb(PORT_C,14);
  outportb(PORT_H,1);
  outportb(PORT_C,12);
  reg_a = inportb(PORT_A);
  outportb(PORT_C,14);
  disp_acc();
```

```
}
/**** OUT PORT ****/
OUT()
{
  unsigned char no_port;
  ++p_count;
  no_port = *(pcram+p_count);
  outportb(PORT_A,no_port);
  outportb(PORT_H,0); /* Write data */
  outportb(PORT_C,15);
  outportb(PORT_D,13);
  outportb(PORT_C,14);
  outportb(PORT_A,reg_a);
  outportb(PORT_C,10);
  outportb(PORT_C,14);
  outportb(PORT_H,1);
}
/**** DI ****/
DI()
{
 NOP();
}
/**** EI ****/
EI()
{
 NOP();
}
```

OPCODE2.C

```
/**** MOV B,B ****/
MOV_B_B()
{
}
/**** MOV B,C ****/
MOV_B_C()
{
  reg_b = reg_c;
  disp_bc();
}
/**** MOV B,D ****/
MOV_B_D()
{
  reg_b = reg_d;
  disp_bc();
}
/**** MOV B,E ****/
MOV_B_E()
{
  reg_b = reg_e;
  disp_bc();
}
/**** MOV B,H ****/
MOV_B_H()
{
  reg_b = reg_h;
  disp_bc();
}
/**** MOV B,L ****/
MOV_B_L()
{
  reg_b = reg_l;
  disp_bc();
```

```
}
/**** MOV B,M ****/
MOV_B_M()
{
  reg_b = RD_r_M();
  disp_bc();
}
/**** MOV B,A ****/
MOV_B_A()
{
  reg_b = reg_a;
  disp_bc();
}
/**** MOV C,B ****/
MOV_C_B()
{
  reg_c = reg_b;
  disp_bc();
}
/**** MOV C,C ****/
MOV_C_C()
{
}
/**** MOV C,D ****/
MOV_C_D()
{
  reg_c = reg_d;
  disp_bc();
}
/**** MOV C,E ****/
MOV_C_E()
{
  reg_c = reg_e;
```

```
  disp_bc();
}
/**** MOV C,H ****/
MOV_C_H()
{
  reg_c = reg_h;
  disp_bc();
}
/**** MOV C,L ****/
MOV_C_L()
{
  reg_c = reg_l;
  disp_bc();
}
/**** MOV C,M ****/
MOV_C_M()
{
  reg_c = RD_r_M();
  disp_bc();
}
/**** MOV C,A ****/
MOV_C_A()
{
  reg_c = reg_a;
  disp_bc();
}
/**** MOV D,B ****/
MOV_D_B()
{
  reg_d = reg_b;
  disp_de();
}
/**** MOV D,C ****/
```

```
MOV_D_C()
{
  reg_d = reg_c;
  disp_de();
}
/**** MOV D,D ****/
MOV_D_D()
{
}
/**** MOV D,E ****/
MOV_D_E()
{
  reg_d = reg_e;
  disp_de();
}
/**** MOV D,H ****/
MOV_D_H()
{
  reg_d = reg_h;
  disp_de();
}
/**** MOV D,L ****/
MOV_D_L()
{
  reg_d = reg_l;
  disp_de();
}
/**** MOV D,M ****/
MOV_D_M()
{
  reg_d = RD_r_M();
  disp_de();
}
```

```
/**** MOV D,A ****/
MOV_D_A()
{
  reg_d = reg_a;
  disp_de();
}
/**** MOV E,B ****/
MOV_E_B()
{
  reg_e = reg_b;
  disp_de();
}
/**** MOV E,C ****/
MOV_E_C()
{
  reg_e = reg_c;
  disp_de();
}
/**** MOV E,D ****/
MOV_E_D()
{
  reg_e = reg_d;
  disp_de();
}
/**** MOV E,E ****/
MOV_E_E()
{
}
/**** MOV E,H ****/
MOV_E_H()
{
  reg_e = reg_h;
  disp_de();
```

```
}
/**** MOV E,L ****/
MOV_E_L()
{
  reg_e = reg_l;
  disp_de();
}
/**** MOV E,M ****/
MOV_E_M()
{
  reg_e = RD_r_M();
  disp_de();
}
/**** MOV E,A ****/
MOV_E_A()
{
  reg_e = reg_a;
  disp_de();
}
/**** MOV H,B ****/
MOV_H_B()
{
  reg_h = reg_b;
  disp_hl();
}
/**** MOV H,C ****/
MOV_H_C()
{
  reg_h = reg_c;
  disp_hl();
}
/**** MOV H,D ****/
MOV_H_D()
```

```
{
  reg_h = reg_d;
  disp_hl();
}
/**** MOV H,E ****/
MOV_H_E()
{
  reg_h = reg_e;
  disp_hl();
}
/**** MOV H,H ****/
MOV_H_H()
{
}
/**** MOV H,L ****/
MOV_H_L()
{
  reg_h = reg_l;
  disp_hl();
}
/**** MOV H,M ****/
MOV_H_M()
{
  reg_h = RD_r_M();
  disp_hl();
}
/**** MOV H,A ****/
MOV_H_A()
{
  reg_h = reg_a;
  disp_hl();
}
/**** MOV L,B ****/
```

```
MOV_L_B()
{
  reg_l = reg_b;
  disp_hl();
}
/**** MOV L,C ****/
MOV_L_C()
{
  reg_l = reg_c;
  disp_hl();
}
/**** MOV L,D ****/
MOV_L_D()
{
  reg_l = reg_d;
  disp_hl();
}
/**** MOV L,E ****/
MOV_L_E()
{
  reg_l = reg_e;
  disp_hl();
}
/**** MOV L,H ****/
MOV_L_H()
{
  reg_l = reg_h;
  disp_hl();
}
/**** MOV L,L ****/
MOV_L_L()
{
}
```

```
/**** MOV L,M ****/
MOV_L_M()
{
  reg_l = RD_r_M();
  disp_hl();
}
/**** MOV L,A ****/
MOV_L_A()
{
  reg_l = reg_a;
  disp_hl();
}
/**** MOV M,B ****/
MOV_M_B()
{
  unsigned char r8;
  r8 = reg_b; WR_r_M(r8);
}
/**** MOV M,C ****/
MOV_M_C()
{
  unsigned char r8;
  r8 = reg_c; WR_r_M(r8);
}
/**** MOV M,D ****/
MOV_M_D()
{
  unsigned char r8;
  r8 = reg_d; WR_r_M(r8);
}
/**** MOV M,E ****/
MOV_M_E()
{
```

```
  unsigned char r8;
  r8 = reg_e; WR_r_M(r8);
}
/**** MOV M,H ****/
MOV_M_H()
{
  unsigned char r8;
  r8 = reg_h; WR_r_M(r8);
}
/**** MOV M,L ****/
MOV_M_L()
{
  unsigned char r8;
  r8 = reg_l; WR_r_M(r8);
}
/**** MOV M,A ****/
MOV_M_A()
{
  unsigned char r8;
  r8 = reg_a; WR_r_M(r8);
}
/**** HLT ****/
HLT()
{
  unsigned char interrup,reset;

  outportb(PORT_H,7);
  outportb(PORT_D,12);
  for(;;) {
    interrup = inportb(PORT_B);
    interrup &= 0;
    if(interrup != 0) break;
    reset = inportb(PORT_C);
```

```
    reset &= 0x28;
    if(reset != 0x28) break;
  };
  initial();
  p_count=0;
  disp_pc();
  hide_cursor();
}
/**** MOV A,B ****/
MOV_A_B()
{
  reg_a = reg_b;
  disp_acc();
}
/**** MOV A,C ****/
MOV_A_C()
{
  reg_a = reg_c;
  disp_acc();
}
/**** MOV A,D ****/
MOV_A_D()
{
  reg_a = reg_d;
  disp_acc();
}
/**** MOV A,E ****/
MOV_A_E()
{
  reg_a = reg_e;
  disp_acc();
}
/**** MOV A,H ****/
```

```
MOV_A_H()
{
  reg_a = reg_h;
  disp_acc();
}
/**** MOV A,L ****/
MOV_A_L()
{
  reg_a = reg_l;
  disp_acc();
}
/**** MOV A,M ****/
MOV_A_M()
{
  reg_a = RD_r_M();
  disp_acc();
}
/**** MOV A,A ****/
MOV_A_A()
{
}
/**** ADD B ****/
ADD_B()
{
  int old,ans;
  old = reg_a; reg_a += reg_b;
  ans = reg_a;
  add_flag(ans,old,reg_b);
}
/**** ADD C ****/
ADD_C()
{
  int old,ans;
```

```
  old = reg_a; reg_a += reg_c;
  ans = reg_a;
  add_flag(ans,old,reg_c);
}
/**** ADD D ****/
ADD_D()
{
  int old,ans;
  old = reg_a; reg_a += reg_d;
  ans = reg_a;
  add_flag(ans,old,reg_d);
}
/**** ADD E ****/
ADD_E()
{
  int old,ans;
  old = reg_a; reg_a += reg_e;
  ans = reg_a;
  add_flag(ans,old,reg_e);
}
/**** ADD H ****/
ADD_H()
{
  int old,ans;
  old = reg_a; reg_a += reg_h;
  ans = reg_a;
  add_flag(ans,old,reg_h);
}
/**** ADD L ****/
ADD_L()
{
  int old,ans;
  old = reg_a; reg_a += reg_l;
```

```
  ans = reg_a;
  add_flag(ans,old,reg_l);
}
/**** ADD M ****/
ADD_M()
{
  int old,ans;
  unsigned char m;
  m = RD_r_M();
  old = reg_a; reg_a += m;
  ans = reg_a;
  add_flag(ans,old,m);
}
/**** ADD A ****/
ADD_A()
{
  int old,ans;
  old = reg_a; reg_a += reg_a;
  ans = reg_a;
  add_flag(ans,old,old);
}
/**** ADC B ****/
ADC_B()
{
  int old,ans;
  old = reg_a; reg_a += reg_b;
  reg_a += (carry_f == 1) ? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,reg_b);
}
/**** ADC C ****/
ADC_C()
{
```

```
  int old,ans;
  old = reg_a; reg_a += reg_c;
  reg_a += (carry_f == 1) ? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,reg_c);
}
/**** ADC D ****/
ADC_D()
{
  int old,ans;
  old = reg_a; reg_a += reg_d;
  reg_a += (carry_f == 1) ? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,reg_d);
}
/**** ADC E ****/
ADC_E()
{
  int old,ans;
  old = reg_a; reg_a += reg_e;
  reg_a += (carry_f == 1) ? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,reg_e);
}
/**** ADC H ****/
ADC_H()
{
  int old,ans;
  old = reg_a; reg_a += reg_h;
  reg_a += (carry_f == 1) ? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,reg_h);
}
```

```
/**** ADC L ****/
ADC_L()
{
  int old,ans;
  old = reg_a; reg_a += reg_l;
  reg_a += (carry_f == 1)? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,reg_l);
}
/**** ADC M ****/
ADC_M()
{
  int old,ans;
  unsigned char m;
  m = RD_r_M();
  old = reg_a; reg_a += m;
  reg_a += (carry_f == 1) ? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,m);
}
/**** ADC A ****/
ADC_A()
{
  int old,ans;
  old = reg_a; reg_a += reg_a;
  reg_a += (carry_f == 1) ? 1 : 0;
  ans = reg_a;
  adc_flag(ans,old,old);
}
/**** SUB B ****/
SUB_B()
{
  int old,ans;
```

```
  old = reg_a; reg_a += (~reg_b+1);
  ans = reg_a;
  sub_flag(ans,old,reg_b);
}
/**** SUB C ****/
SUB_C()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_c+1);
  ans = reg_a;
  sub_flag(ans,old,reg_c);
}
/**** SUB D ****/
SUB_D()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_d+1);
  ans = reg_a;
  sub_flag(ans,old,reg_d);
}
/**** SUB E ****/
SUB_E()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_e+1);
  ans = reg_a;
  sub_flag(ans,old,reg_e);
}
/**** SUB H ****/
SUB_H()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_h+1);
```

```
  ans = reg_a;
  sub_flag(ans,old,reg_h);
}
/**** SUB L ****/
SUB_L()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_l+1);
  ans = reg_a;
  sub_flag(ans,old,reg_l);
}
/**** SUB M ****/
SUB_M()
{
  int old,ans;
  unsigned char m;
  m = RD_r_M();
  old = reg_a; reg_a += (~m+1);
  ans = reg_a;
  sub_flag(ans,old,m);
}
/**** SUB A ****/
SUB_A()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_a+1);
  ans = reg_a;
  sub_flag(ans,old,old);
}
/**** SBB B ****/
SBB_B()
{
  int old,ans;
```

```
  old = reg_a; reg_a += (~reg_b+1);
  reg_a += (carry_f == 1) ? 0xFF : 0;
  ans = reg_a;
  sbb_flag(ans,old,reg_b);
}
/**** SBB C ****/
SBB_C()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_c+1);
  reg_a += (carry_f == 1) ? 0xFF : 0;
  ans = reg_a;
  sbb_flag(ans,old,reg_c);
}
/**** SBB D ****/
SBB_D()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_d+1);
  reg_a += (carry_f == 1) ? 0xFF : 0;
  ans = reg_a;
  sbb_flag(ans,old,reg_d);
}
/**** SBB E ****/
SBB_E()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_e+1);
  reg_a += (carry_f == 1) ? 0xFF : 0;
  ans = reg_a;
  sbb_flag(ans,old,reg_e);
}
/**** SBB H ****/
```

```
SBB_H()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_h+1);
  reg_a += (carry_f == 1) ? 0xFF : 0;
  ans = reg_a;
  sbb_flag(ans,old,reg_h);
}
/**** SBB L ****/
SBB_L()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_l+1);
  reg_a += (carry_f == 1) ? 0xFF : 0;
  ans = reg_a;
  sbb_flag(ans,old,reg_l);
}
/**** SBB M ****/
SBB_M()
{
  int old,ans;
  unsigned char m;
  m = RD_r_M();
  old = reg_a; reg_a += (~m+1);
  reg_a += (carry_f == 1) ? 0xFF : 0;
  ans = reg_a;
  sbb_flag(ans,old,m);
}
/**** SBB A ****/
SBB_A()
{
  int old,ans;
  old = reg_a; reg_a += (~reg_a+1);
```

```
    reg_a += (carry_f == 1) ? 0xFF : 0;
    ans = reg_a;
    sbb_flag(ans,old,old);
  }
  /**** ANA B ****/
  ANA_B()
  {
    int ans;
    reg_a &= reg_b; ans = reg_a;
    ana_flag(ans);
  }
  /**** ANA C ****/
  ANA_C()
  {
    int ans;
    reg_a &= reg_c; ans = reg_a;
    ana_flag(ans);
  }
  /**** ANA D ****/
  ANA_D()
  {
    int ans;
    reg_a &= reg_d; ans = reg_a;
    ana_flag(ans);
  }
  /**** ANA E ****/
  ANA_E()
  {
    int ans;
    reg_a &= reg_e; ans = reg_a;
    ana_flag(ans);
  }
  /**** ANA H ****/
```

```
ANA_H()
{
  int ans;
  reg_a &= reg_h; ans = reg_a;
  ana_flag(ans);
}
/**** ANA L ****/
ANA_L()
{
  int ans;
  reg_a &= reg_l; ans = reg_a;
  ana_flag(ans);
}
/**** ANA M ****/
ANA_M()
{
  int ans;
  unsigned char m;
  m = RD_r_M();
  reg_a &= m; ans = reg_a;
  ana_flag(ans);
}
/**** ANA A ****/
ANA_A()
{
  int ans;
  reg_a &= reg_a; ans = reg_a;
  ana_flag(ans);
}
/**** XRA B ****/
XRA_B()
{
  int ans;
```

```
  reg_a ^= reg_b; ans = reg_a;
  or_xor_flag(ans);
}
/**** XRA C ****/
XRA_C()
{
  int ans;
  reg_a ^= reg_c; ans = reg_a;
  or_xor_flag(ans);
}
/**** XRA D ****/
XRA_D()
{
  int ans;
  reg_a ^= reg_d; ans = reg_a;
  or_xor_flag(ans);
}
/**** XRA E ****/
XRA_E()
{
  int ans;
  reg_a ^= reg_e; ans = reg_a;
  or_xor_flag(ans);
}
/**** XRA H ****/
XRA_H()
{
  int ans;
  reg_a ^= reg_h; ans = reg_a;
  or_xor_flag(ans);
}
/**** XRA L ****/
XRA_L()
```

```
{
  int ans;
  reg_a ^= reg_l; ans = reg_a;
  or_xor_flag(ans);
}
/**** XRA M ****/
XRA_M()
{
  int ans;
  unsigned char m;
  m = RD_r_M();
  reg_a ^= m; ans = reg_a;
  or_xor_flag(ans);
}
/**** XRA A ****/
XRA_A()
{
  int ans;
  reg_a ^= reg_a; ans = reg_a;
  or_xor_flag(ans);
}
/**** ORA B ****/
ORA_B()
{
  int ans;
  reg_a |= reg_b; ans = reg_a;
  or_xor_flag(ans);
}
/**** ORA C ****/
ORA_C()
{
  int ans;
  reg_a |= reg_c; ans = reg_a;
```

```
  or_xor_flag(ans);
}
/**** ORA D ****/
ORA_D()
{
  int ans;
  reg_a |= reg_d; ans = reg_a;
  or_xor_flag(ans);
}
/**** ORA E ****/
ORA_E()
{
  int ans;
  reg_a |= reg_e; ans = reg_a;
  or_xor_flag(ans);
}
/**** ORA H ****/
ORA_H()
{
  int ans;
  reg_a |= reg_h; ans = reg_a;
  or_xor_flag(ans);
}
/**** ORA L ****/
ORA_L()
{
  int ans;
  reg_a |= reg_l; ans = reg_a;
  or_xor_flag(ans);
}
/**** ORA M ****/
ORA_M()
{
```

```
    int ans;
    unsigned char m;
    m = RD_r_M();
    reg_a |= m; ans = reg_a;
    or_xor_flag(ans);
}
/**** ORA A ****/
ORA_A()
{
    int ans;
    reg_a |= reg_a; ans = reg_a;
    or_xor_flag(ans);
}
/**** CMP B ****/
CMP_B()
{
    int old,ans;
    old = reg_a; ans = reg_a +(~reg_b+1);
    cmp_flag(ans,old,reg_b);
}
/**** CMP C ****/
CMP_C()
{
    int old,ans;
    old = reg_a; ans = reg_a +(~reg_c+1);
    cmp_flag(ans,old,reg_c);
}
/**** CMP D ****/
CMP_D()
{
    int old,ans;
    old = reg_a; ans = reg_a +(~reg_d+1);
    cmp_flag(ans,old,reg_d);
```

```
}
/**** CMP E ****/
CMP_E()
{
  int old,ans;
  old = reg_a; ans = reg_a +(~reg_e+1);
  cmp_flag(ans,old,reg_e);
}
/**** CMP H ****/
CMP_H()
{
  int old,ans;
  old = reg_a; ans = reg_a +(~reg_h+1);
  cmp_flag(ans,old,reg_h);
}
/**** CMP L ****/
CMP_L()
{
  int old,ans;
  old = reg_a; ans = reg_a +(~reg_l+1);
  cmp_flag(ans,old,reg_l);
}
/**** CMP M ****/
CMP_M()
{
  int old,ans;
  unsigned char m;
  m = RD_r_M();
  old = reg_a; ans = reg_a +(~m+1);
  cmp_flag(ans,old,m);
}
```

```
/**** CMP A ****/
CMP_A()
{
  int old,ans;
  old = reg_a; ans = reg_a +(~reg_a+1);
  cmp_flag(ans,old,reg_a);
}
```

UNASSEM .C

```
char *Mnemonic;
chk_op(int mode)
{
    switch(*(pcram+p_count)) {
      case 0x00: Mnemonic = "NOP"; disp_f1(mode); break;
      case 0x01: Mnemonic = "LXI  B,";disp_f3(mode); break;
      case 0x02: Mnemonic = "STAX B";disp_f1(mode);break;
      case 0x03: Mnemonic = "INX B";disp_f1(mode); break;
      case 0x04: Mnemonic = "INR B";disp_f1(mode); break;
      case 0x05: Mnemonic = "DCR B";disp_f1(mode); break;
      case 0x06: Mnemonic = "MVI B,";disp_f2(mode); break;
      case 0x07: Mnemonic = "RLC";disp_f1(mode); break;
      case 0x08: break;
      case 0x09: Mnemonic = "DAD B";disp_f1(mode); break;
      case 0x0A: Mnemonic = "LDAX B";disp_f1(mode); break;
      case 0x0B: Mnemonic = "DCX B";disp_f1(mode); break;
      case 0x0C: Mnemonic = "INR C";disp_f1(mode); break;
      case 0x0D: Mnemonic = "DCR C";disp_f1(mode); break;
      case 0x0E: Mnemonic = "MVI C,";disp_f2(mode); break;
      case 0x0F: Mnemonic = "RRC";disp_f1(mode); break;
      case 0x10: break;
      case 0x11: Mnemonic = "LXI D,";disp_f3(mode); break;
      case 0x12: Mnemonic = "STAX D";disp_f1(mode); break;
      case 0x13: Mnemonic = "INX D";disp_f1(mode); break;
      case 0x14: Mnemonic = "INR D";disp_f1(mode); break;
      case 0x15: Mnemonic = "DCR D";disp_f1(mode); break;
      case 0x16: Mnemonic = "MVI D,";disp_f2(mode); break;
      case 0x17: Mnemonic = "RAL";disp_f1(mode); break;
      case 0x18: break;
      case 0x19: Mnemonic = "DAD D";disp_f1(mode); break;
      case 0x1A: Mnemonic = "LDAX D";disp_f1(mode); break;
      case 0x1B: Mnemonic = "DCX D";disp_f1(mode); break;
      case 0x1C: Mnemonic = "INR E";disp_f1(mode); break;
```

```
case 0x1D: Mnemonic = "DCR E";disp_f1(mode); break;
case 0x1E: Mnemonic = "MVI E,";disp_f2(mode); break;
case 0x1F: Mnemonic = "RAR";disp_f1(mode); break;
case 0x20: Mnemonic = "RIM";disp_f1(mode); break;
case 0x21: Mnemonic = "LXI H,";disp_f3(mode); break;
case 0x22: Mnemonic = "SHLD ";disp_f3(mode); break;
case 0x23: Mnemonic = "INX H";disp_f1(mode); break;
case 0x24: Mnemonic = "INR H";disp_f1(mode); break;
case 0x25: Mnemonic = "DCR H";disp_f1(mode); break;
case 0x26: Mnemonic = "MVI H,";disp_f2(mode); break;
case 0x27: Mnemonic = "DAA";disp_f1(mode); break;
case 0x28: break;
case 0x29: Mnemonic = "DAD H";disp_f1(mode); break;
case 0x2A: Mnemonic = "LHLD ";disp_f3(mode); break;
case 0x2B: Mnemonic = "DCX H";disp_f1(mode); break;
case 0x2C: Mnemonic = "INR L";disp_f1(mode); break;
case 0x2D: Mnemonic = "DCR L";disp_f1(mode); break;
case 0x2E: Mnemonic = "MVI L,";disp_f2(mode); break;
case 0x2F: Mnemonic = "CMA";disp_f1(mode); break;
case 0x30: Mnemonic = "SIM";disp_f1(mode); break;
case 0x31: Mnemonic = "LXI SP,";disp_f3(mode); break;
case 0x32: Mnemonic = "STA ";disp_f3(mode); break;
case 0x33: Mnemonic = "INX SP";disp_f1(mode); break;
case 0x34: Mnemonic = "INR M";disp_f1(mode); break;
case 0x35: Mnemonic = "DCR M";disp_f1(mode); break;
case 0x36: Mnemonic = "MVI M,";disp_f2(mode); break;
case 0x37: Mnemonic = "STC";disp_f1(mode); break;
case 0x38: break;
case 0x39: Mnemonic = "DAD SP";disp_f1(mode); break;
case 0x3A: Mnemonic = "LDA ";disp_f3(mode); break;
case 0x3B: Mnemonic = "DCX SP";disp_f1(mode); break;
case 0x3C: Mnemonic = "INR A";disp_f1(mode); break;
case 0x3D: Mnemonic = "DCR A";disp_f1(mode); break;
```

```
case 0x3E: Mnemonic = "MVI A,";disp_f2(mode); break;
case 0x3F: Mnemonic = "CMC";disp_f1(mode); break;
case 0x40: Mnemonic = "MOV B,B";disp_f1(mode); break;
case 0x41: Mnemonic = "MOV B,C";disp_f1(mode); break;
case 0x42: Mnemonic = "MOV B,D";disp_f1(mode); break;
case 0x43: Mnemonic = "MOV B,E";disp_f1(mode); break;
case 0x44: Mnemonic = "MOV B,H";disp_f1(mode); break;
case 0x45: Mnemonic = "MOV B,L";disp_f1(mode); break;
case 0x46: Mnemonic = "MOV B,M";disp_f1(mode); break;
case 0x47: Mnemonic = "MOV B,A";disp_f1(mode); break;
case 0x48: Mnemonic = "MOV C,B";disp_f1(mode); break;
case 0x49: Mnemonic = "MOV C,C";disp_f1(mode); break;
case 0x4A: Mnemonic = "MOV C,D";disp_f1(mode); break;
case 0x4B: Mnemonic = "MOV C,E";disp_f1(mode); break;
case 0x4C: Mnemonic = "MOV C,H";disp_f1(mode); break;
case 0x4D: Mnemonic = "MOV C,L";disp_f1(mode); break;
case 0x4E: Mnemonic = "MOV C,M";disp_f1(mode); break;
case 0x4F: Mnemonic = "MOV C,A";disp_f1(mode); break;
case 0x50: Mnemonic = "MOV D,B";disp_f1(mode); break;
case 0x51: Mnemonic = "MOV D,C";disp_f1(mode); break;
case 0x52: Mnemonic = "MOV D,D";disp_f1(mode); break;
case 0x53: Mnemonic = "MOV D,E";disp_f1(mode); break;
case 0x54: Mnemonic = "MOV D,H";disp_f1(mode); break;
case 0x55: Mnemonic = "MOV D,L";disp_f1(mode); break;
case 0x56: Mnemonic = "MOV D,M";disp_f1(mode); break;
case 0x57: Mnemonic = "MOV D,A";disp_f1(mode); break;
case 0x58: Mnemonic = "MOV E,B";disp_f1(mode); break;
case 0x59: Mnemonic = "MOV E,C";disp_f1(mode); break;
case 0x5A: Mnemonic = "MOV E,D";disp_f1(mode); break;
case 0x5B: Mnemonic = "MOV E,E";disp_f1(mode); break;
case 0x5C: Mnemonic = "MOV E,H";disp_f1(mode); break;
case 0x5D: Mnemonic = "MOV E,L";disp_f1(mode); break;
case 0x5E: Mnemonic = "MOV E,M";disp_f1(mode); break;
```

```
case 0x5F: Mnemonic = "MOV E,A";disp_f1(mode); break;
case 0x60: Mnemonic = "MOV H,B";disp_f1(mode); break;
case 0x61: Mnemonic = "MOV H,C";disp_f1(mode); break;
case 0x62: Mnemonic = "MOV H,D";disp_f1(mode); break;
case 0x63: Mnemonic = "MOV H,E";disp_f1(mode); break;
case 0x64: Mnemonic = "MOV H,H";disp_f1(mode); break;
case 0x65: Mnemonic = "MOV H,L";disp_f1(mode); break;
case 0x66: Mnemonic = "MOV H,M";disp_f1(mode); break;
case 0x67: Mnemonic = "MOV H,A";disp_f1(mode); break;
case 0x68: Mnemonic = "MOV L,B";disp_f1(mode); break;
case 0x69: Mnemonic = "MOV L,C";disp_f1(mode); break;
case 0x6A: Mnemonic = "MOV L,D";disp_f1(mode); break;
case 0x6B: Mnemonic = "MOV L,E";disp_f1(mode); break;
case 0x6C: Mnemonic = "MOV L,H";disp_f1(mode); break;
case 0x6D: Mnemonic = "MOV L,L";disp_f1(mode); break;
case 0x6E: Mnemonic = "MOV L,M";disp_f1(mode); break;
case 0x6F: Mnemonic = "MOV L,A";disp_f1(mode); break;
case 0x70: Mnemonic = "MOV M,B";disp_f1(mode); break;
case 0x71: Mnemonic = "MOV M,C";disp_f1(mode); break;
case 0x72: Mnemonic = "MOV M,D";disp_f1(mode); break;
case 0x73: Mnemonic = "MOV M,E";disp_f1(mode); break;
case 0x74: Mnemonic = "MOV M,H";disp_f1(mode); break;
case 0x75: Mnemonic = "MOV M,L";disp_f1(mode); break;
case 0x76: Mnemonic = "HLT";disp_f1(mode); break;
case 0x77: Mnemonic = "MOV M,A";disp_f1(mode); break;
case 0x78: Mnemonic = "MOV A,B";disp_f1(mode); break;
case 0x79: Mnemonic = "MOV A,C";disp_f1(mode); break;
case 0x7A: Mnemonic = "MOV A,D";disp_f1(mode); break;
case 0x7B: Mnemonic = "MOV A,E";disp_f1(mode); break;
case 0x7C: Mnemonic = "MOV A,H";disp_f1(mode); break;
case 0x7D: Mnemonic = "MOV A,L";disp_f1(mode); break;
case 0x7E: Mnemonic = "MOV A,M";disp_f1(mode); break;
case 0x7F: Mnemonic = "MOV A,A";disp_f1(mode); break;
```

```
case 0x80: Mnemonic = "ADD B";disp_f1(mode); break;
case 0x81: Mnemonic = "ADD C";disp_f1(mode); break;
case 0x82: Mnemonic = "ADD D";disp_f1(mode); break;
case 0x83: Mnemonic = "ADD E";disp_f1(mode); break;
case 0x84: Mnemonic = "ADD H";disp_f1(mode); break;
case 0x85: Mnemonic = "ADD L";disp_f1(mode); break;
case 0x86: Mnemonic = "ADD M";disp_f1(mode); break;
case 0x87: Mnemonic = "ADD A";disp_f1(mode); break;
case 0x88: Mnemonic = "ADC B";disp_f1(mode); break;
case 0x89: Mnemonic = "ADC C";disp_f1(mode); break;
case 0x8A: Mnemonic = "ADC D";disp_f1(mode); break;
case 0x8B: Mnemonic = "ADC E";disp_f1(mode); break;
case 0x8C: Mnemonic = "ADC H";disp_f1(mode); break;
case 0x8D: Mnemonic = "ADC L";disp_f1(mode); break;
case 0x8E: Mnemonic = "ADC M";disp_f1(mode); break;
case 0x8F: Mnemonic = "ADC A";disp_f1(mode); break;
case 0x90: Mnemonic = "SUB B";disp_f1(mode); break;
case 0x91: Mnemonic = "SUB C";disp_f1(mode); break;
case 0x92: Mnemonic = "SUB D";disp_f1(mode); break;
case 0x93: Mnemonic = "SUB E";disp_f1(mode); break;
case 0x94: Mnemonic = "SUB H";disp_f1(mode); break;
case 0x95: Mnemonic = "SUB L";disp_f1(mode); break;
case 0x96: Mnemonic = "SUB M";disp_f1(mode); break;
case 0x97: Mnemonic = "SUB A";disp_f1(mode); break;
case 0x98: Mnemonic = "SBB B";disp_f1(mode); break;
case 0x99: Mnemonic = "SBB C";disp_f1(mode); break;
case 0x9A: Mnemonic = "SBB D";disp_f1(mode); break;
case 0x9B: Mnemonic = "SBB E";disp_f1(mode); break;
case 0x9C: Mnemonic = "SBB H";disp_f1(mode); break;
case 0x9D: Mnemonic = "SBB L";disp_f1(mode); break;
case 0x9E: Mnemonic = "SBB M";disp_f1(mode); break;
case 0x9F: Mnemonic = "SBB A";disp_f1(mode); break;
case 0xA0: Mnemonic = "ANA B";disp_f1(mode); break;
```

```
case 0xA1: Mnemonic = "ANA C";disp_f1(mode); break;
case 0xA2: Mnemonic = "ANA D";disp_f1(mode); break;
case 0xA3: Mnemonic = "ANA E";disp_f1(mode); break;
case 0xA4: Mnemonic = "ANA H";disp_f1(mode); break;
case 0xA5: Mnemonic = "ANA L";disp_f1(mode); break;
case 0xA6: Mnemonic = "ANA M";disp_f1(mode); break;
case 0xA7: Mnemonic = "ANA A";disp_f1(mode); break;
case 0xA8: Mnemonic = "XRA B";disp_f1(mode); break;
case 0xA9: Mnemonic = "XRA C";disp_f1(mode); break;
case 0xAA: Mnemonic = "XRA D";disp_f1(mode); break;
case 0xAB: Mnemonic = "XRA E";disp_f1(mode); break;
case 0xAC: Mnemonic = "XRA H";disp_f1(mode); break;
case 0xAD: Mnemonic = "XRA L";disp_f1(mode); break;
case 0xAE: Mnemonic = "XRA M";disp_f1(mode); break;
case 0xAF: Mnemonic = "XRA A";disp_f1(mode); break;
case 0xB0: Mnemonic = "ORA B";disp_f1(mode); break;
case 0xB1: Mnemonic = "ORA C";disp_f1(mode); break;
case 0xB2: Mnemonic = "ORA D";disp_f1(mode); break;
case 0xB3: Mnemonic = "ORA E";disp_f1(mode); break;
case 0xB4: Mnemonic = "ORA H";disp_f1(mode); break;
case 0xB5: Mnemonic = "ORA L";disp_f1(mode); break;
case 0xB6: Mnemonic = "ORA M";disp_f1(mode); break;
case 0xB7: Mnemonic = "ORA A";disp_f1(mode); break;
case 0xB8: Mnemonic = "CMP B";disp_f1(mode); break;
case 0xB9: Mnemonic = "CMP C";disp_f1(mode); break;
case 0xBA: Mnemonic = "CMP D";disp_f1(mode); break;
case 0xBB: Mnemonic = "CMP E";disp_f1(mode); break;
case 0xBC: Mnemonic = "CMP H";disp_f1(mode); break;
case 0xBD: Mnemonic = "CMP L";disp_f1(mode); break;
case 0xBE: Mnemonic = "CMP M";disp_f1(mode); break;
case 0xBF: Mnemonic = "CMP A";disp_f1(mode); break;
case 0xC0: Mnemonic = "RNZ";disp_f1(mode); break;
case 0xC1: Mnemonic = "POP B";disp_f1(mode); break;
```

```
case 0xC2: Mnemonic = "JNZ ";disp_f3(mode); break;
case 0xC3: Mnemonic = "JMP ";disp_f3(mode); break;
case 0xC4: Mnemonic   "CNZ ";disp_f3(mode); break;
case 0xC5: Mnemonic = "PUSH B";disp_f1(mode); break;
case 0xC6: Mnemonic = "ADI ";disp_f2(mode); break;
case 0xC7: Mnemonic = "RST 0";disp_f1(mode); break;
case 0xC8: Mnemonic = "RZ";disp_f1(mode); break;
case 0xC9: Mnemonic = "RET";disp_f1(mode); break;
case 0xCA: Mnemonic = "JZ ";disp_f3(mode); break;
case 0xCB: break;
case 0xCC: Mnemonic = "CZ ";disp_f3(mode); break;
case 0xCD: Mnemonic = "CALL ";disp_f3(mode); break;
case 0xCE: Mnemonic = "ACI ";disp_f2(mode); break;
case 0xCF: Mnemonic = "RST 1";disp_f1(mode); break;
case 0xD0: Mnemonic = "RNC";disp_f1(mode); break;
case 0xD1: Mnemonic = "POP D";disp_f1(mode); break;
case 0xD2: Mnemonic = "JNC ";disp_f3(mode); break;
case 0xD3: Mnemonic = "OUT ";disp_f2(mode); break;
case 0xD4: Mnemonic = "CNC ";disp_f3(mode); break;
case 0xD5: Mnemonic =
case 0xD6: Mnemonic =
case 0xD7: Mnemonic = "RST 2";disp_f1(mode); break;
case 0xD8: Mnemonic = "RC";disp_f1(mode); break;
case 0xD9: break;
case 0xDA: Mnemonic = "JC ";disp_f3(mode); break;
case 0xDB: Mnemonic = "IN ";disp_f2(mode); break;
case 0xDC: Mnemonic = "CC ";disp_f3(mode); break;
case 0xDD: break;
case 0xDE: Mnemonic = "SBI ";disp_f2(mode); break;
case 0xDF: Mnemonic = "RST 3";disp_f1(mode); break;
case 0xE0: Mnemonic = "RPO";disp_f1(mode); break;
case 0xE1: Mnemonic = "POP H";disp_f1(mode); break;
case 0xE2: Mnemonic = "JPO ";disp_f3(mode); break;
```

```
        case 0xE3: Mnemonic = "XTHL";disp_f1(mode); break;
        case 0xE4: Mnemonic = "CPO ";disp_f3(mode); break;
        case 0xE5: Mnemonic = "PUSH H";disp_f1(mode); break;
        case 0xE6: Mnemonic = "ANI ";disp_f2(mode); break;
        case 0xE7: Mnemonic = "RST 4";disp_f1(mode); break;
        case 0xE8: Mnemonic = "RPE";disp_f1(mode); break;
        case 0xE9: Mnemonic = "PCHL";disp_f1(mode); break;
        case 0xEA: Mnemonic = "JPE ";disp_f3(mode); break;
        case 0xEB: Mnemonic = "XCHG";disp_f1(mode); break;
        case 0xEC: Mnemonic = "CPE ";disp_f3(mode); break;
        case 0xED: break;
        case 0xEE: Mnemonic = "XRI ";disp_f2(mode); break;
        case 0xEF: Mnemonic = "RST 5";disp_f1(mode); break;
        case 0xF0: Mnemonic = "RP";disp_f1(mode); break;
        case 0xF1: Mnemonic = "POP PSW";disp_f1(mode); break;
        case 0xF2: Mnemonic = "JP ";disp_f3(mode); break;
        case 0xF3: Mnemonic = "DI";disp_f1(mode); break;
        case 0xF4: Mnemonic = "CP ";disp_f3(mode); break;
        case 0xF5: Mnemonic = "PUSH PSW";disp_f1(mode); break;
        case 0xF6: Mnemonic = "ORI ";disp_f2(mode); break;
        case 0xF7: Mnemonic = "RST 6";disp_f1(mode); break;
        case 0xF8: Mnemonic = "RM";disp_f1(mode); break;
        case 0xF9: Mnemonic = "SPHL";disp_f1(mode); break;
        case 0xFA: Mnemonic = "JM ";disp_f3(mode); break;
        case 0xFB: Mnemonic = "EI";disp_f1(mode); break;
        case 0xFC: Mnemonic = "CM ";disp_f3(mode); break;
        case 0xFD: break;
        case 0xFE: Mnemonic = "CPI ";disp_f2(mode); break;
        case 0xFF: Mnemonic = "RST 7";disp_f1(mode); break;
    }
}
```

```
disp_f1(int pos)
{
  if(pos==PRINT) {
      fmhex_up8(*(pcram+p_count));
      fprintf(stdprn,"          %s",Mnemonic);
  } else {
      hex_up8(*(pcram+p_count));
      printf("          %s",Mnemonic);
  };
}
disp_f2(int pos)
{
  if(pos==PRINT) {
     fmhex_up8(*(pcram+p_count));
     fprintf(stdprn," ");
     p_count+=1;
     fmhex_up8(*(pcram+p_count));
     fprintf(stdprn,"      %s",Mnemonic);
     fmhex_up8(*(pcram+p_count));
     fprintf(stdprn,"h");
  } else {
     hex_up8(*(pcram+p_count));
     printf(" ");
     p_count+=1;
     hex_up8(*(pcram+p_count));
     printf("       %s",Mnemonic);
     hex_up8(*(pcram+p_count));
     printf("h");
  };
}
disp_f3(int pos)
{
  if(pos==PRINT) {
```

```
        fmhex_up8(*(pcram+p_count));fprintf(stdprn," ");
        p_count+=1;
        fmhex_up8(*(pcram+p_count));fprintf(stdprn," ");
        p_count+=1;
        fmhex_up8(*(pcram+p_count));
        fprintf(stdprn,"    %s",Mnemonic);
        fmhex_up8(*(pcram+(p_count)));
        fmhex_up8(*(pcram+p_count-1));
        fprintf(stdprn,"h");
    } else {
        hex_up8(*(pcram+p_count));printf(" ");
        p_count+=1;
        hex_up8(*(pcram+p_count));printf(" ");
        p_count+=1;
        hex_up8(*(pcram+p_count));
        printf("    %s",Mnemonic);
        hex_up8(*(pcram+(p_count)));
        hex_up8(*(pcram+p_count-1));
        printf("h");
    };
}
```

INSTRUCT.C

```
/*****************************************************************/
/* INSTRUCT.C  - Programmer..... Mr. Prarinya Ekapho  */
/* This program is collection utility routine of EMU8085.C     */
/*****************************************************************/
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#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  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_AO       16        /* enable port A */
#define  PH_AC       17        /* disable port A */

#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

unsigned char *pcram,byte2,byte3,reg_ir,reg_im;
unsigned char psw,reg_a,reg_b,reg_c,reg_d,reg_e,reg_h,reg_l;
int  ra16,rb16,rc16,rd16,re16,rh16,rl16,flag16,ir16;
char sign_f,zero_f,aux_f,parity_f,carry_f;
unsigned int  s_point,p_count,p_break;

hex_up_rp(reg1,reg2)
unsigned char reg1,reg2;
{
  unsigned char n11,n10,n21,n20;
  n11=n10=reg1; n21=n20=reg2;
  n10 &= 0x0f; n20 &= 0x0f;
  n11 >>= 4; n21 >>= 4;
  n11 += (n11 < 0xA) ? 0x30 : 0x37;
  n10 += (n10 < 0xA) ? 0x30 : 0x37;
  n21 += (n21 < 0xA) ? 0x30 : 0x37;
  n20 += (n20 < 0xA) ? 0x30 : 0x37;
  printf("%c%c%c%c",n11,n10,n21,n20);
}

hex_up16(r_16)
unsigned int r_16;
```

```
{
  unsigned int n0,n1,n2,n3;
  n0=n1=n2=n3=r_16;
  n0 &= 0x000f; n1 &= 0x00f0; n1 >>= 4;
  n2 &= 0x0f00; n2 >>= 8; n3 &= 0xf000;
  n3 >>= 12;
  n0 += (n0 < 0xA) ? 0x30 : 0x37;
  n1 += (n1 < 0xA) ? 0x30 : 0x37;
  n2 += (n2 < 0xA) ? 0x30 : 0x37;
  n3 += (n3 < 0xA) ? 0x30 : 0x37;
  printf("%c%c%c%c",n3,n2,n1,n0);
}
hex_up8(r_8)
unsigned char r_8;
{
  unsigned char n1,n0;
  n1=n0=r_8;
  n0 &= 0x0f;
  n1 >>= 4;
  n1 += (n1 < 0xA) ? 0x30 : 0x37;
  n0 += (n0 < 0xA) ? 0x30 : 0x37;
  printf("%c%c",n1,n0);
}
fnhex_up16(r_16)
unsigned int r_16;
{
  unsigned int n0,n1,n2,n3;
  n0=n1=n2=n3=r_16;
  n0 &= 0x000f; n1 &= 0x00f0; n1 >>= 4;
  n2 &= 0x0f00; n2 >>= 8; n3 &= 0xf000;
  n3 >>= 12;
  n0 += (n0 < 0xA) ? 0x30 : 0x37;
  n1 += (n1 < 0xA) ? 0x30 : 0x37;
```

```
  n2 += (n2 < 0xA) ? 0x30 : 0x37;
  n3 += (n3 < 0xA) ? 0x30 : 0x37;
  fprintf(stdprn,"%c%c%c%c",n3,n2,n1,n0);
}
fmhex_up8(r_8)
unsigned char r_8;
{
  unsigned char n1,n0;
  n1=n0=r_8;
  n0 &= 0x0f;
  n1 >>= 4;
  n1 += (n1 < 0xA) ? 0x30 : 0x37;
  n0 += (n0 < 0xA) ? 0x30 : 0x37;
  fprintf(stdprn,"%c%c",n1,n0);
}
fhex_up8(hex)
unsigned char hex;
{
  unsigned char high,low;
  high = low = hex;
  low &= 0x0f; high >>= 4;
  high += (high < 0xA) ? 0x30 : 0x37;
  low  += (low  < 0xA) ? 0x30 : 0x37;
  fprintf(stdprn,"%c%c ",high,low);
}
fhex_up16(hex16)
unsigned int hex16;
{
  unsigned int byte0,byte1,byte2,byte3;
  byte0 = byte1 = byte2 = byte3 = hex16;
  byte0 &= 0x000f; byte1 &= 0x00f0; byte1 >>= 4;
  byte2 &= 0x0f00; byte2 >>= 8; byte3 &= 0xf000;
  byte3 >>= 12;
```

```
  byte0 += (byte0 < 0xA) ? 0x30 : 0x37;
  byte1 += (byte1 < 0xA) ? 0x30 : 0x37;
  byte2 += (byte2 < 0xA) ? 0x30 : 0x37;
  byte3 += (byte3 < 0xA) ? 0x30 : 0x37;
  fprintf(stdprn,"%c%c%c%c : ",byte3,byte2,byte1,byte0);
}
disp_bc()
{
   gotoxy(P_BC,P_ROW);
   hex_up_rp(reg_b,reg_c);
}
disp_de()
{
   gotoxy(P_DE,P_ROW);
   hex_up_rp(reg_d,reg_e);
}
disp_hl()
{
  gotoxy(P_HL,P_ROW);
  hex_up_rp(reg_h,reg_l);
}
disp_psw()
{
  gotoxy(P_PSW,P_ROW);
  hex_up8(psw);
}
disp_acc()
{
  gotoxy(P_ACC,P_ROW);
  hex_up8(reg_a);
}
disp_pc()
{
```

```
    gotoxy(P_PC,P_ROW);
    hex_up16(p_count);
  }
  disp_sp()
  {
    gotoxy(P_SP,P_ROW);
    hex_up16(s_point);
  }
  /**** test sign flags ****/
  sign(int test)
  {
    test &= 0x80;
    sign_f = (test == 0x80) ? 1 : 0;
  }
  /**** test zero flags ****/
  zero(int test)
  {
    test &= 0xFF;
    zero_f = (test == 0) ? 1 : 0;
  }
  /**** test auxiliary carry ****/
  aux_add(int st,int nd)
  {
    st &= 0x0F; nd &= 0x0F;
    st += nd; st &= 0x10;
    aux_f = (st == 0x10) ? 1 : 0;
  }
  aux_adc(int st,int nd)
  {
    st &= 0x0F; nd &= 0x0F;
    st += nd; st += (carry_f == 1) ? 1 : 0;
    st &= 0x10; aux_f = (st == 0x10) ? 1 : 0;
  }
```

```
aux_sub(int st,int nd)
{
  st &= 0x0F; nd = (~nd+1);
  nd &= 0x0F; st += nd; st &= 0x10;
  aux_f = (st == 0x10) ? 1 : 0;
}
aux_sbb(int st,int nd)
{
  st &= 0x0F; nd = (~nd+1);
  nd &= 0x0F; st += nd;
  if(carry_f == 1) { st &= 0x0F; st += 0x0F; }
  st &= 0x10; aux_f = (st == 0x10) ? 1 : 0;
}
/**** test parity status (1 for even ,0 for odd) ****/
parity(int test)
{ .
  int i,count,check;
  for(i=0,count=0,check=0;i<8;i++) {
    check = test & 0x01;
    if(check == 0x01) count++;
    test >>= 1;
  };
  count %= 2;
  parity_f = (count == 0) ? 1 : 0;
}
/**** test carry flags ****/
carry_add(int test)
{
  carry_f = (test > 0xFF) ? 1 : 0;
}
carry_sub(int test)
{
  carry_f = (test > 0xFF) ? 0 : 1;
```

```
}
add_flag(int result,int prv,unsigned char r8)
{
  int op2;op2=r8;
  sign(result); zero(result);
  aux_add(prv,op2); parity(result); prv += r8;
  carry_add(prv); flag_to_psw();
  disp_psw();DISP_ALL_F;disp_acc();
}
adc_flag(int result,int prv,unsigned char r8)
{
  int op2;op2 = r8;
  sign(result); zero(result);
  aux_adc(prv,op2); parity(result);
  prv += op2;prv += (carry_f == 1) ? 1 : 0;
  carry_add(prv); flag_to_psw();
  disp_psw();DISP_ALL_F;disp_acc();
}
sub_flag(int result,int prv,unsigned char r8)
{
  int op2;op2=r8;
  sign(result); zero(result);
  aux_sub(prv,op2); parity(result); prv += (~r8+1);
  carry_sub(prv); flag_to_psw();
  disp_psw();DISP_ALL_F;disp_acc();
}
sbb_flag(int result,int prv,unsigned char r8)
{
  int op2;op2=r8;
  sign(result); zero(result);
  aux_sbb(prv,op2); parity(result);
  prv += (~r8+1);
  prv += (carry_f == 1) ? 0xFF : 0;
```

```
  carry_sub(prv); flag_to_psw();
  disp_psw();DISP_ALL_F;disp_acc();
}
ana_flag(int result)
{
  sign(result); zero(result);
  aux_f = 1; parity(result); carry_f = 0;
  flag_to_psw(); disp_psw();
  DISP_ALL_F;disp_acc();
}
or_xor_flag(int result)
{
  sign(result); zero(result);
  aux_f = 0; parity(result); carry_f = 0;
  flag_to_psw(); disp_psw();
  DISP_ALL_F;disp_acc();
}
cmp_flag(int result,int prv,unsigned char r8)
{
  int op2;op2=r8;
  sign(result); zero(result);
  aux_sub(prv,op2); parity(result);
  carry_sub(result);flag_to_psw();
  disp_psw();DISP_ALL_F;
}
flag_to_psw()
{
   if(sign_f==0 && zero_f==0 && aux_f==0 && parity_f==0 && carry_f==0)
psw=0x00;  else
   if(sign_f==0 && zero_f==0 && aux_f==0 && parity_f==0 && carry_f==1)
psw=0x01;  else
   if(sign_f==0 && zero_f==0 && aux_f==0 && parity_f==1 && carry_f==0)
psw=0x04;  else
```

```
   if(sign_f==0 && zero_f==0 && aux_f==0 && parity_f==1 && carry_f==1)
psw=0x05;  else
   if(sign_f==0 && zero_f==0 && aux_f  1 && parity_f==0 && carry_f==0)
psw=0x10;  else
   if(sign_f==0 && zero_f==0 && aux_f==1 && parity_f==0 && carry_f==1)
psw=0x11;  else
   if(sign_f==0 && zero_f==0 && aux_f==1 && parity_f==1 && carry_f==0)
psw=0x14;  else
   if(sign_f==0 && zero_f==0 && aux_f==1 && parity_f==1 && carry_f==1)
psw=0x15;  else
   if(sign_f==0 && zero_f==1 && aux_f==0 && parity_f==0 && carry_f==0)
psw=0x40;  else
   if(sign_f==0 && zero_f==1 && aux_f==0 && parity_f==0 && carry_f==1)
psw=0x41;  else
   if(sign_f==0 && zero_f==1 && aux_f==0 && parity_f==1 && carry_f==0)
psw=0x44;  else
   if(sign_f==0 && zero_f==1 && aux_f==0 && parity_f==1 && carry_f==1)
psw=0x45;  else
   if(sign_f==0 && zero_f==1 && aux_f==1 && parity_f==0 && carry_f==0)
psw=0x50;  else
   if(sign_f==0 && zero_f==1 && aux_f==1 && parity_f==0 && carry_f==1)
psw=0x51;  else
   if(sign_f==0 && zero_f==1 && aux_f==1 && parity_f==1 && carry_f==0)
psw=0x54;  else
   if(sign_f==0 && zero_f==1 && aux_f==1 && parity_f==1 && carry_f==1)
psw=0x55;  else
   if(sign_f==1 && zero_f==0 && aux_f==0 && parity_f==0 && carry_f==0)
psw=0x80;  else
   if(sign_f==1 && zero_f==0 && aux_f==0 && parity_f==0 && carry_f==1)
psw=0x81;  else
   if(sign_f==1 && zero_f==0 && aux_f==0 && parity_f==1 && carry_f==0)
psw=0x84;  else
   if(sign_f==1 && zero_f==0 && aux_f==0 && parity_f==1 && carry_f==1)
```

```
psw=0x85;  else
   if(sign_f==1 && zero_f==0 && aux_f==1 && parity_f==0 && carry_f==0)
psw=0x90;  else
   if(sign_f==1 && zero_f==0 && aux_f==1 && parity_f==0 && carry_f==1)
psw=0x91;  else
   if(sign_f==1 && zero_f==0 && aux_f==1 && parity_f==1 && carry_f==0)
psw=0x94;  else
   if(sign_f==1 && zero_f==0 && aux_f==1 && parity_f==1 && carry_f==1)
psw=0x95;  else
   if(sign_f==1 && zero_f==1 && aux_f==0 && parity_f==0 && carry_f==0)
psw=0xC0;  else
   if(sign_f==1 && zero_f==1 && aux_f==0 && parity_f==0 && carry_f==1)
psw=0xC1;  else
   if(sign_f==1 && zero_f==1 && aux_f==0 && parity_f==1 && carry_f==0)
psw=0xC4;  else
   if(sign_f==1 && zero_f==1 && aux_f==0 && parity_f==1 && carry_f==1)
psw=0xC5;  else
   if(sign_f==1 && zero_f==1 && aux_f==1 && parity_f==0 && carry_f==0)
psw=0xD0;  else
   if(sign_f==1 && zero_f==1 && aux_f==1 && parity_f==0 && carry_f==1)
psw=0xD1;  else
   if(sign_f==1 && zero_f==1 && aux_f==1 && parity_f==1 && carry_f==0)
psw=0xD4;  else
   if(sign_f==1 && zero_f==1 && aux_f==1 && parity_f==1 && carry_f==1)
psw=0xD5;
}
7
psw_to_flag()
{
   switch(psw) {
   case 0x00: sign_f=0;zero_f=0;aux_f=0;parity_f=0;carry_f=0;break;
   case 0x01: sign_f=0;zero_f=0;aux_f=0;parity_f=0;carry_f=1;break;
   case 0x04: sign_f=0;zero_f=0;aux_f=0;parity_f=1;carry_f=0;break;
   case 0x05: sign_f=0;zero_f=0;aux_f=0;parity_f=1;carry_f=1;break;
```

```
case 0x10: sign_f=0;zero_f=0;aux_f=1;parity_f=0;carry_f=0;break;
case 0x11: sign_f=0;zero_f=0;aux_f=1;parity_f=0;carry_f=1;break;
case 0x14: sign_f=0;zero_f=0;aux_f=1;parity_f=1;carry_f=0;break;
case 0x15: sign_f=0;zero_f=0;aux_f=1;parity_f=1;carry_f=1;break;
case 0x40: sign_f=0;zero_f=1;aux_f=0;parity_f=0;carry_f=0;break;
case 0x41: sign_f=0;zero_f=1;aux_f=0;parity_f=0;carry_f=1;break;
case 0x44: sign_f=0;zero_f=1;aux_f=0;parity_f=1;carry_f=0;break;
case 0x45: sign_f=0;zero_f=1;aux_f=0;parity_f=1;carry_f=1;break;
case 0x50: sign_f=0;zero_f=1;aux_f=1;parity_f=0;carry_f=0;break;
case 0x51: sign_f=0;zero_f=1;aux_f=1;parity_f=0;carry_f=1;break;
case 0x54: sign_f=0;zero_f=1;aux_f=1;parity_f=1;carry_f=0;break;
case 0x55: sign_f=0;zero_f=1;aux_f=1;parity_f=1;carry_f 1;break;
case 0x80: sign_f=1;zero_f=0;aux_f=0;parity_f=0;carry_f=0;break;
case 0x81: sign_f=1;zero_f=0;aux_f=0;parity_f=0;carry_f=1;break;
case 0x84: sign_f=1;zero_f=0;aux_f=0;parity_f=1;carry_f=0;break;
case 0x85: sign_f=1;zero_f=0;aux_f=0;parity_f=1;carry f=1;break;
case 0x90: sign_f=1;zero_f=0;aux_f=1;parity_f=0;carry_f=0;break;
case 0x91: sign_f=1;zero_f=0;aux_f=1;parity_f=0;carry_f=1;break;
case 0x94: sign_f=1;zero_f=0;aux_f=1;parity_f=1;carry_f=0;break;
case 0x95: sign_f=1;zero_f=0;aux_f=1;parity_f=1;carry_f=1;break;
case 0xC0: sign_f=1;zero_f=1;aux_f=0;parity_f=0;carry_f=0;break;
case 0xC1: sign_f=1;zero_f=1;aux_f=0;parity_f=0;carry_f=1;break;
case 0xC4: sign_f=1;zero_f=1;aux_f=0;parity_f=1;carry_f=0;break;
case 0xC5: sign_f=1;zero_f=1;aux_f=0;parity_f=1;carry_f=1;break;
case 0xD0: sign_f=1;zero_f=1;aux_f=1;parity_f=0;carry_f 0;break;
case 0xD1: sign_f=1;zero_f=1;aux_f=1;parity_f=0;carry_f=1;break;
case 0xD4: sign_f=1;zero_f=1;aux_f=1;parity_f=1;carry_f=0;break;
case 0xD5: sign_f=1;zero_f=1;aux_f=1;parity_f=1;carry_f=1;break;
default :  break;
}
}
```

```
initial()
{
    outportb(PORT_A,0);      /* out data 0 to A0-A7                    */
    outportb(PORT_H,PH_AO); /* enable port A                          */
    outportb(PORT_C,15);     /* ALE=RD=WR=IO/M=1                       */
    outportb(PORT_B,0);      /* out data 0 to A8-A15                   */
    outportb(PORT_C,14);     /* ALE=0, RD=WR=IO/M=1                    */
    outportb(PORT_D,11);     /* S0=S1=INTA=1,RSTOUT=0,HLDA=SOD=0       */
    outportb(PORT_D,15);     /* S0=S1=RSTOUT=INTA=1,HLDA=SOD=0         */
}
/**** Write memory ****/
WR_M(unsigned char data)
{
  unsigned int addr;
  addr=byte3; addr<<=8;
  addr|=byte2;
  *(pcram+addr)=data;
  outportb(PORT_A,byte2);  /* out A0-A7 to port A       */
  outportb(PORT_H,PH_AO);  /* enable port A (outport)   */
  outportb(PORT_B,byte3);  /* out A8-A15 to port B      */
  outportb(PORT_C,7);      /* ALE=1,IO/M=0              */
  outportb(PORT_D,13);     /* S0=1,S1=0                 */
  outportb(PORT_C,6);      /* ALE=0,IO/M=0              */
  outportb(PORT_A,data);   /* out data 1 byte to port A*/
  outportb(PORT_C,2);      /* WR=0 (active)             */
  outportb(PORT_C,6);      /* WR=1 (deactive)           */
  outportb(PORT_H,PH_AC);  /* disable port A            */
}
/**** Read memory ****/
RD_M()
{
  unsigned char data;
  unsigned int addr;
```

```
  addr=reg_h; addr<<=8;
  addr|=reg_l;
  data = *(pcram+addr);
  outportb(PORT_A,byte2);  /* out A0-A7 to port A          */
  outportb(PORT_H,PH_A0);  /* enable port A (outport)      */
  outportb(PORT_B,byte3);  /* out A8-A15 to port B         */
  outportb(PORT_C,7);      /* ALE=1,IO/M=0                 */
  outportb(PORT_D,14);     /* S0=0,S1=1                    */
  outportb(PORT_C,6);      /* ALE=0,IO/M=0                 */
  outportb(PORT_H,PH_AC);  /* disable port A               */
  outportb(PORT_C,4);      /* RD=0 (active)                */
  data = inportb(PORT_A);  /* input data 1 byte into port A */
  outportb(PORT_C,6);      /* RD=1 (deactive)              */
  return(data);            /* retuen data 1 byte           */
}
/**** READ REGISTER TO MEMORY ****/
RD_r_M()
{
  unsigned char data;
  unsigned int addr;
  addr=reg_h; addr<<=8;
  addr|=reg_l;
  data = *(pcram+addr);
  outportb(PORT_A,reg_l);  /* out A0-A7 to port A          */
  outportb(PORT_H,PH_A0);  /* enable port A (outport)      */
  outportb(PORT_B,reg_h);  /* out A8-A15 to port B         */
  outportb(PORT_C,7);      /* ALE=1,IO/M=0                 */
  outportb(PORT_D,14);     /* S0=0,S1=1                    */
  outportb(PORT_C,6);      /* ALE=0,IO/M=0                 */
  outportb(PORT_H,PH_AC);  /* disable port A               */
  outportb(PORT_C,4);      /* RD=0 (active)                */
  data = inportb(PORT_A);  /* input data 1 byte into port A */
  outportb(PORT_C,6);      /* RD=1 (deactive)              */
```

```
  return(data);              /* retuen data 1 byte              */
}
RD_r_M_F()
{
  unsigned char data;
  outportb(PORT_A,reg_l);  /* out A0-A7 to port A  */
  outportb(PORT_H,PH_A0);  /* enable port A (outport) */
  outportb(PORT_B,reg_h);  /* out A8-A15 to port B */
  outportb(PORT_C,7);      /* ALE=1,IO/M=0  */
  outportb(PORT_D,14);     /* S0=0,S1=1     */
  outportb(PORT_C,6);      /* ALE=0,IO/M=0    */
  outportb(PORT_H,PH_AC);  /* disable port A (set to high impedance)*/
  outportb(PORT_C,4);      /* RD=0 (active)    */
  data = inportb(PORT_A);  /* input data 1 byte into port A */
  outportb(PORT_C,6);      /* RD=1 (deactive)   */
  return(data);            /* retuen data 1 byte */
}
/**** WRITE REGISTER TO MEMORY ****/
WR_r_M(unsigned char data)
{
  unsigned int addr;
  addr=reg_h; addr<<=8;
  addr|=reg_l;
  *(pcram+addr)=data;
  outportb(PORT_A,reg_l);  /* out A0-A7 to port A             */
  outportb(PORT_H,PH_A0);  /* enable port A (outport)         */
  outportb(PORT_B,reg_h);  /* out A8-A15 to port B            */
  outportb(PORT_C,7);      /* ALE=1,IO/M=0                    */
  outportb(PORT_D,13);     /* S0=1,S1=0                       */
  outportb(PORT_C,6);      /* ALE=0,IO/M=0                    */
  outportb(PORT_A,data);   /* out data 1 byte to port A       */
  outportb(PORT_C,2);      /* WR=0 (active)                   */
  outportb(PORT_C,6);      /* WR=1 (deactive)                 */
```

```
  outportb(PORT_H,PH_AC);  /* disable port A                    */
}
WR_r_M_F(unsigned char data)
{
  outportb(PORT_A,reg_l);  /* out A0-A7 to port A */
  outportb(PORT_H,PH_AO);  /* enable port A (outport) */
  outportb(PORT_B,reg_h);  /* out A8-A15 to port B */
  outportb(PORT_C,7);      /* ALE=1,IO/M=0    */
  outportb(PORT_D,13);     /* S0=1,S1=0       */
  outportb(PORT_C,6);      /* ALE=0,IO/M=0    */
  outportb(PORT_A,data);   /* out data 1 byte to port A */
  outportb(PORT_C,2);      /* WR=0 (active)       */
  outportb(PORT_C,6);      /* WR=1 (deactive)     */
  outportb(PORT_H,PH_AC);  /* disable port A (set to high impedance) */
}
/**** STAXOUT ****/
staxout(hb,lb)
unsigned char hb,lb;
{
  unsigned int addr;
  addr=hb; addr<<=8;
  addr|=lb;
  *(pcram+addr) = reg_a;
  outportb(PORT_A,lb);     /* out A0-A7 to port A              */
  outportb(PORT_H,PH_AO);  /* enable port A (outport)          */
  outportb(PORT_B,hb);     /* out A8-A15 to port B             */
  outportb(PORT_C,7);      /* ALE=1,IO/M=0                     */
  outportb(PORT_D,13);     /* S0=1,S1=0                        */
  outportb(PORT_C,6);      /* ALE=0,IO/M=0                     */
  outportb(PORT_A,reg_a);  /* out data 1 byte to port A        */
  outportb(PORT_C,2);      /* WR=0 (active)                    */
  outportb(PORT_C,6);      /* WR=1 (deactive)                  */
  outportb(PORT_H,PH_AC);  /* disable port A (high impedance)  */
```

```
}
/**** LDAXOUT ****/
ldaxout(hb,lb)
unsigned char hb,lb;
{
  unsigned int addr;
  addr=hb; addr<<=8;
  addr|=lb;
  reg_a = *(pcram+addr);
  outportb(PORT_A,lb);      /* out A0-A7 to port A                */
  outportb(PORT_H,PH_AO);   /* enable port A (outport)            */
  outportb(PORT_B,hb);      /* out A8-A15 to port B               */
  outportb(PORT_C,7);       /* ALE=1,IO/M=0                       */
  outportb(PORT_D,14);      /* S0=0,S1=1                          */
  outportb(PORT_C,6);       /* ALE=0,IO/M=0                       */
  outportb(PORT_H,PH_AC);   /* disable port A (high impedance)    */
  outportb(PORT_C,4);       /* RD=0 (active)                      */
  reg_a = inportb(PORT_A); /* input data 1 byte into port A       */
  outportb(PORT_C,6);       /* RD=1 (deactive)                    */
}
/**** CHECK CARRY 16 BIT ****/
cy_16(long int hx)
{
  carry_f = (hx > 0xFFFF) ? 1 : 0;
}
/* call subrutine */
call_sub()
{
  unsigned char ph,pl;
  unsigned int px;
  --s_point;
  px = p_count+3; ph=(px>>8); pl=px;
  push_rp(ph);
```

```
  --s_point;
  push_rp(pl);
}
/* call restart */
call_rst()
{
  unsigned char ph,pl;
  unsigned int px;
  --s_point;
  px = p_count+1; ph=(px)>>8); pl=px;
  push_rp(ph);
  --s_point;
  push_rp(pl);
}
pop_rp()
{
  unsigned char data;
  data = *(pcram+s_point);
  outport(PORT_A,s_point);
  outportb(PORT_H,PH_AO);
  outport(PORT_B,s_point)>>8);
  outportb(PORT_C,7);
  outportb(PORT_D,14);
  outportb(PORT_C,6);
  outportb(PORT_H,PH_AC);
  outportb(PORT_C,4);
  data = inportb(PORT_A);
  outportb(PORT_C,6);
  return(data);
}

push_bdhp(r1, r2)
unsigned char r1,r2;
```

```
{
  --s_point;
  push_rp(r1);
  --s_point;
  push_rp(r2);
}
push_rp(unsigned char r8)
{
  *(pcram+s_point) = r8;
  outport(PORT_A,s_point);
  outportb(PORT_H,PH_AO);
  outport(PORT_B,s_point>>8);
  outportb(PORT_C,7);
  outportb(PORT_D,13);
  outportb(PORT_C,6);
  outportb(PORT_A,r8);
  outportb(PORT_C,2);
  outportb(PORT_C,6);
  outportb(PORT_H,PH_AC);
}
```

WIN_MENU.C

```
#define   BANNER      "8085 EMULATOR   Copyright 1991  KMIT'C"
#define   BORDER      1
#define   MAX_FRAME   12
#define   BKSP        8
#define   TAB         3

void window_(),window_cleol(),window_gets(),window_cls();
void display_menu(),save_video(),restore_video(),restore_video_win();
void goto_xy(),cursor(),find_cursor(),cls(),save_video_win();
void write_string(),write_char(),colorstr(),c_scroll();
int  arrow_get,arrow_item,arrow_item1;
int  key_geted,key_recive,key_recive1;
int  arrow_choice,len_deso;

char far *vid_mem;

char *file[]= {
  " Load     ",
  " Save     ",
  " Dump     ",
  " Transfer ",
};

char *t_mode[]= {
  " Load from EPROM ",
  " Store into RAM  ",
};

char *disp_file[]= {
  " Hexadecimal ",
  " Mnemonic    ",
```

```
 char *edit[]= {
   " Modify  ",
   " View    ",
 };

 char *view_mode[]= {
   " Program  ",
   " Data     ",
 };

 char *run[]= {
   " Run all       ",
   " Single step   ",
   " Program reset ",

 };

 char  *other[]= {
   " Modify reg ",
   " Test port  ",
};

char *quit[]= {
  " Version      ",
  " 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 ",
};

MENU    menu0[] = {
        OPTION(" File ", "  Load,Save,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 "," Modify register or Test port      ", 'o')
        OPTION(" Quit ", "  display Version or Exit to dos   ", 'q')
        OPTIONEND
        };

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; /* current cursor positon in window */
    unsigned char *p; /* pointer to buffer */
    char *header; /* header message */
```

```
    int border;  /* border on/off */
    int active; /* on screen yes/no */
  } frame_win[MAX_FRAME];

  /*****************************************
  *            MENU FUNCTION               *
  *****************************************/
  void menu_start(MENU s[],int fore, int back)
  {
    int buttons,i,r,c,wide;

    wide = getvcols()-1;

    find_cursor(&prow,&pcol);          /* find out where to put it */
    c_scroll(prow,pcol-4,wide,1,0,6,back << 4);
    last = 0;

    nptr = (int *) calloc(wide, sizeof(int));  /* Store col positions */
    cptr = (char *) calloc(wide,sizeof(char)); /* Store letter options*/

    for(menulen = i = 0; s[i].prompt; i++) {
       find_cursor(&r,&c);
       nptr[i] = c;                        /* Fill in colums */
       cptr[i] = s[i].letter;              /* Single-letter choices */

       colorstr(s[i].prompt, fore, back << 4);
       if(i != 4)
         colorstr(MARGIN, fore, back << 4);
       menulen += s[i].plen;               /* find total menu length  */
    }
    menucount = 1;                         /* Item in this menu       */
    menulen += (strlen(MARGIN) * (menucount-1)); /* Length of menu line */
  }
```

```
void first_active(MENU s[],int fore, int back)
{
    union scan {
    int  c;
    char ch[2];
  } sc;

  do  {
    hide_cursor();
    sc.c = get_key();
  } while(sc.ch[1] != F10 );      /* F10 call menu */
  switch(last) {
   case 0:
    pcol=5;
    break;
   case 1:
    pcol=21;
    break;
   case 2:
    pcol=37;
    break;
   case 3:
    pcol=52;
    break;
   case 4:
    pcol=69;
    break;
   default:
    break;
  }
  cursor(prow,pcol);
  colorstr(s[last].prompt, back, A_INTENSE);
  len_deso = (80-strlen(s[last].deso)) / 2;
```

```
  cursor(prow + 23,len_deso);
  colorstr(s[last].deso, fore, A_NORM);
}

void menu_on(MENU m[], int fore, int back)
{
  active(m[last].prompt,prow,nptr[last],back,A_INTENSE);
  active(m[last].deso,prow + 23,len_deso,fore,back);
}

void menu_off(MENU m[], int fore, int back)
{
  active_off(m[last].prompt, prow, nptr[last], fore, back);
  c_scroll(prow+23,len_deso,m[last].dlen,1,1,6,A_INTENSE);
}

void menu_end(void)      /* close all menu */
{
    free(nptr);
    free(cptr);
}
int  menu_choice(MENU m[], int fore, int back)
{
   int stop=0;
   union scan {
    int  c;
    char ch[2];
  } sc;

  while(TRUE) {
    arrow_get=0;                /* reset every time */
    hide_cursor();
    sc.c = get_key();
```

```
if(sc.ch[0] == 0) {
 switch (sc.ch[1])' {
   case RARROW:              /* Used the right arrow */
      menu_off(m, fore, back); /* Restore to not active */
      if(last + 1 >= menucount) / *Wrap menu around */
      {
       if(last == 4)
        last = 0;
       else
        last++;
      }                        /* Move one to the right */
      else
        last = 0;
        menu_on(m, fore, back);
        break;
   case LARROW:                / *Used the left arrow */
        menu_off(m,fore,back);
        if(last - 1 < 0)
                last   menucount + 3;
        else                    /* Move one to the left */
                last--;
        menu_on(m,fore,back);
        break;
   default:                     / *Don't know what it was */
        break;
 } / *end switch */
} else
   switch(tolower(sc.ch[0]))  {
      case 'f':
        menu_off(m,fore,back);
        last=0;
        menu_on(m,fore,back);
```

```
            break;
          case 'e':
            menu_off(m,fore,back);
            last=1;
            menu_on(m,fore,back);
            break;
          case 'r':
            menu_off(m,fore,back);
            last=2;
            menu_on(m,fore,back);
            break;
          case 'o':
            menu_off(m,fore,back);
            last=3;
            menu_on(m,fore,back);
            break;
          case 'q':
            menu_off(m,fore,back);
            last=4;
            menu_on(m,fore,back);
            break;
          default:                    / *Don't know what it
            break;
      } /* end switch */
   / *condition of loop while */
   if (sc.ch[0] == ENTER || sc.ch[1] == DARROW)
      {
         stop = select_menu();
         if(stop == -2) return stop;
      }
   else if(sc.ch[0] == ESCAPE) {
         menu_off(m,fore,back);
         clsmess();
```

```
        hide_cursor();
        break;
      }
  do {
        switch(arrow_get) {
          case LARROW:          /* Move one to the left */
              menu_off(m,fore,back);
              if(last - 1 < 0)
                    last = menucount + 3;
              else
                    last--;
              menu_on(m,fore,back);
              stop = select_menu();
              if(stop == -2) return stop;
              break;
         case RARROW:
              menu_off(m, fore, back); /* Restore to not active */
              if(last + 1 >= menucount) /* Wrap menu around */
              {
               if(last == 4)
                    last = 0;
               else
                    last++;
              }                         /* Move one to the right */
              else
                 last = 0;
              menu_on(m, fore, back);
              stop = select_menu();
              if(stop == -2) return stop;
              break;
         case 10: /* if you select menu modify reg */
              menu_off(m, fore, back);
              return last;
```

```
        } /* end switch */
    } while(arrow_get!=0);
  }  /* end while main */
  return last;
}
select_menu()              /* get choice of your selection */
{
   int sure=0;

        switch(last) {
          case 0:
             menu_file();
             clsmess();
             break;
          case 1:
             menu_edit();
             clsmess();
             break;
          case 2:
             menu_run();
             clsmess();
             break;
          case 3:
             menu_other();
             break;
          case 4:
             sure = menu_quit();
             clsmess();
             return sure;
          }
  return 0;
}
```

```
/*********************************************
*       DISPLAY EDIT WINDOW & BORDER          *
*********************************************/

#define     MES1          " Register and Status Flags "
#define     MES2          " Messages "

border()        /* draw the fram or border of main menu */
{
  int r,c,temp;

  cursor(3,1);    /* format function is cursor(row,col) */
  putchar(C_ULSD); /* print chr$ single and double upper left */
  for(c = 2; c < WIDTH; c++)
      putchar(C_HD); /* draw double horizontal line */
  putchar(C_URSD); /* print chr$ single and double upper right */
  for(r= 4; r < J1_HEIGHT; r++) {
    putchar(C_V);
     if(r==4) {
          cursor(r,5);printf("PC");cursor(r,12);printf("SP");
          cursor(r,19);printf("ACC");cursor(r,27);printf("BC");
          cursor(r,33);printf("DE");cursor(r,39);printf("HL");
          cursor(r,45);printf("PSW");cursor(r,52);printf("S");
          cursor(r,55);printf("Z");cursor(r,58);printf("X");
          cursor(r,61);printf("AC");cursor(r,65);printf("X");
          cursor(r,68);printf("P");cursor(r,71);printf("X");
          cursor(r,74);printf("C");
     }
     if(r==5) {
          disp_pc();disp_sp();disp_acc();
          disp_bc();disp_de();disp_hl();
          disp_psw();DISP_S;DISP_Z;
          cursor(r,58);printf("0");DISP_AC;
```

```
            cursor(r,65);printf("0");DISP_P;
            cursor(r,71);printf("0");DISP_C;
        }
      cursor(r,WIDTH);
      putchar(C_V);
    }
    putchar(C_XLSD);
    for(c = 2; c < WIDTH; c++)
      putchar(C_HD);
    putchar(C_XRSD);  /* end of Register and Status Flags window */
    temp =  J1_HEIGHT+1;
    for(r = temp; r < J2_HEIGHT; r++) {
      putchar(C_V);
      cursor(r,WIDTH);
      putchar(C_V);
    }
    putchar(C_XL);
    for(c = 2; c < WIDTH; c++)
      putchar(C_H);
    putchar(C_XR);
    for(r = 23; r < HEIGHT; r++) {
      putchar(C_V);
      cursor(r,WIDTH);
      putchar(C_V);
    }
    putchar(C_LL);
    for(c = 2; c < WIDTH; c++)
      putchar(C_H);
    putchar(C_LR);

    write_head((80-strlen(MES1))/2, 3,  MES1, 0, A_NORM);
    write_head((80-strlen(MES2))/2, 22, MES2, 0, A_INVERSE);
}
```

```
write_head( x, y, h, f, attr)
int x,y,f,attr;
char *h;
{
  int i;

  for(i=0; h[i]; i++,x++) {
     cursor(y,x);
     colorchr(h[i], f, attr);
  }
}

cls_edit(r,c)
int r,c;
{
 int i,j;
 for(i=r;i<20;i++) {
   for(j=c;j<79;j++) {
     cursor(i,j);
     printf(" ");
   }
 }
}

clsmess()
{
    cursor(23,2);
    printf("
}
/******** END OF MENU FUNCTION ********/
```

```
/****************************************
*        SCREEN FUNCTION               *
****************************************/
/* clear the screen */
void cls()
{
  union REGS r;

  r.h.ah=6;  /* screen scroll code */
  r.h.al=0;  /* clear screen code */
  r.h.ch=0;  /* start row */
  r.h.cl=0;  /* start column */
  r.h.dh=24; /* end row */
  r.h.dl=79; /* end column */
  r.h.bh=7;  /* blank line is blank */
  int86(0x10, &r, &r);
}

void active_off(char *m, int row, int col, int fore, int back)
{
    cursor(row,col);
    colorstr(m,fore,back << 4);
}

void active(char *m, int row, int col, int fore, int back)

    cursor(row,col);
    colorstr(m,fore,back); /* A_INTENSE */
}
void c_scroll(introw,int col,int wide,intdeep,intnum,int f int color)
{
   union REGS ireg;
```

```
    row--;                          /* Index from(1,1)  */
    col--;
    ireg.h.ah = f;
    ireg.h.al = num;
    ireg.h.ch = row;
    ireg.h.cl = col;
    ireg.h.dh = row + deep - 1;
    ireg.h.dl = col + wide;
    ireg.h.bh = color;
    int86(0x10, &ireg, &ireg);
}
/* write string in color */
void colorstr(char *s, int fcolor, int bcolor)
{
  int col, row, wide;
  wide = getvcols();                    /* width of display */

  find_cursor(&row, &col);
  while (*s)  {
    switch (*s)  {
      case '\n':
            s++;
            row++;
            col = 1;
            break;
      case '\b':
            s++;
            col--;
            break;
      case '\r':
            s++;
            col = 1;
            break;
```

```
        case '\t':
                s++;
                col += TAB;
                break;
        case '\f':
                s++;
                break;
        case '\a':
                s++;
                putchar(BELL);
                continue;
        case '\\':
                col++;
                break;
        default:
                col++;
                break;
      }
      cursor(row, col);
      colorchr(*s++,fcolor,bcolor);
      if (col > wide) {
        row++;
        col = 1;
      }
    }
    cursor(row,col);
}
colorchr(int c, int fcolor, int bcolor) /* write charecter in color */
{
    union REGS ireg;

    ireg.h.ah = 0x09;                   /* Function 9 */
    ireg.x.cx = 1;                      /* Number of character *
```

```
  ireg.h.bh = getvpage();          /* Video page */
  ireg.h.al = (char) c;
  ireg.h.bl = (char) (fcolor | bcolor);
  int86(0x10, &ireg, &ireg);
}

scroll(int row, int col, int wide, int deep, int num, int function)
{
   union REGS ireg;

   ireg.h.ah = function;
   ireg.h.al = num;
   ireg.h.ch = row - 1;                /* Index from (0, 0)  */
   ireg.h.cl = col - 1;
   ireg.h.dh = row + deep;
   ireg.h.dl = col + wide;
   ireg.h.bh = 0;                      /* attribute byte */
   int86(0x10, &ireg, &ireg);
}
int getvpage(void)
{
  union REGS ireg;

  ireg.h.ah = 0x0f;                    /*  Function 0x0f  */
  int86(0x10, &ireg, &ireg);
  return (ireg.h.bh);
}
int getvmode(void)
{
    union  REGS ireg;

    ireg.h.ah = 0x0f;
```

```
    int86(0x10, &ireg, &ireg);
    return(ireg.h.al);
}
/*returns the current video mode  */
video_mode()
{
  union REGS r;

  r.h.ah = 15;   /* get video mode  */
  return int86(0x10, &r, &r) & 255;
}

/* display a string with specified attribute */
void write_string(x, y, p, attrib)
int x, y;
char *p;
int attrib;
{
  register int i;
  char far *v;

  v = vid_mem;
  v += (x*160) + y*2;  /* compute the address */
  for(i=y; *p; i++)  {
    *v++ = *p++;  /*write the character */
    *v++ = attrib;    /*write the attribute  */
  }
}

/* write character with specified attribute */
void write_char(x, y, ch, attrib)
int x, y;
char ch;
```

```
int attrib;
{
  register int i;
  char far *v;

  v = vid_mem;
  v += (x*160) + y*2;  /*  compute the address */
  *v++   ch;  /* write the character */
  *v = attrib;    /* write the attribute */
}
/* save a portion of the screen */
void save_video(num)
int num;
{
  register int i,j;
  char *buf_ptr;

  buf_ptr = frame[num].p;
  v = vid_mem;
  for(i=frame[num].starty; i<frame[num].endy; i++)
    for(j=frame[num].startx; j<frame[num].endx+1; j++) {
      t = v + (j*160) + i*2;  /* compute the address  */
      *buf_ptr++ = *t++;  /* read the character  */
      *buf_ptr++ = *t;    /* read the attribute  */
      *(t-1) = ' ';  /* clear the window  */
    }

/* restore a portion of the screen  */
void restore_video(num)
int num;
{
```

```
register int i,j;
char far *v, far *t;
char *buf_ptr;

buf_ptr = frame[num].p;
v = vid_mem;
t = v;
for(i=frame[num].starty; i<frame[num].endy; i++)
  for(j=frame[num].startx; j<frame[num].endx+1; j++) {
    v = t;
    v += (j*160) + i*2;  /* compute the address  */
    *v++ = *buf_ptr++;  /*  write the character  */
    *v = *buf_ptr++;     /* write the attribute  */
  }
  frame[num].active = 0; /* deactivate */
}
/* display a pull-down menu and return selection */
int pulldown(num)
int num;  /* menu number */
{
  int vmode, choice;

  vmode = video_mode();
  if((vmode!=2) && (vmode!=3) && (vmode!=7))
    printf("video must be in 80 column text mode");
    exit(1);
  }
  /* set proper address of video RAM */
  if(vmode==7) vid_mem = (char far *) 0xB0000000;
  else vid_mem = (char far *) 0xB8000000;
  /* get active window */
  if(!frame[num].active) {  /* not currently in use */
    save_video(num);        /* save the current screen */
```

```
    frame[num].active = 1;  /* set active flag */
  }
  if(frame[num].border) draw_border(num);

  display_menu(num);        /*display the menu */
}

/*construct a puul down menu frame
   1 is returned if menu frame can be constructed
   otherwise 0 is returned.
*/

make_menu(num, menu, keys, count, x, y, border)
int num;        /*menu number */
char *menu[];   /* menu text */
char *keys;     /*hot keys */
int count;      /*number of menu items */
int x, y;       /*X,Y coordinates of left hand corner */
int border;     /*no border if 0 */
{
  register int i, len;
  int endx, endy, choice, vmode;
  unsigned char *p;

  if(num>MAX_FRAME)  {
    printf("Too many menus\n");
    return 0;


  if((x>24) || (x<0) || (y>79) || (y<0))  {
    printf("range error");
    return 0;
  }
```

```
/* compute the size */
len = 0;
for(i=0; i<count; i++)
 if(strlen(menu[i]) > len) len = strlen(menu[i]);
          endy = len + 2 + y;
          endx = count + 1 + x;
if((endx+1>24) || (endy+1>79)) {
  printf("menu won't fit");
  return 0;

 /* allocate enough memory to hold it */
 p = (unsigned char *) malloc(2 * (endx-x+1) * (endy-y+1));
 if(!p) exit(1);   /* put your own error handler here */

 /* construct the frame */
 frame[num].startx = x; frame[num].endx = endx;
 frame[num].starty = y; frame[num].endy = endy;
 frame[num].p = p;
 frame[num].menu = (char **) menu;
 frame[num].border = border;
 frame[num].keys = keys;
 frame[num].count = count;
 frame[num].active = 0;
 return 1;
}

/* display the menu in its proper location */
void display_menu(num)
int num;
{
  int x, y;
  register int i, x_;
```

```
    char **m;

    x_ = frame[num].startx+1;
    m  = frame[num].menu;

    for(i=0; i<frame[num].count; i++, x_++)
      write_string(x_, frame[num].starty+1, m[i], A_INVERSE)

     x = frame[num].startx+1;
     y = frame[num].starty+1;

     /* highlight the first selection */
     goto_xy(x, y);
     write_string(x, y, frame[num].menu[0], A_INTENSE);  /*  everse video */
     hide_cursor();
}

draw_border(num)
int num;
{
  register int i;
  char far *v, far *t;

  v = vid_mem;
  t = v;
  for(i=frame[num].startx+1; i<frame[num].endx; i++)  {
     v += (i*160) + frame[num].starty*2;
     *v++ = C_VD;
     *v = A_INVERSE;
     v = t;
     v += (i*160) + (frame[num].endy-1)*2;
     *v++ = C_VD;
     *v = A_INVERSE;
```

```
    v = t;
  }

  for(i=frame[num].starty+1; i<frame[num].endy; i++) {
    v += (frame[num].startx*160) + i*2;
    *v++ = C_HD;
    *v = A_INVERSE;
    v = t;
    v += (frame[num].endx*160) + i*2;
    *v++ = C_HD;
    *v = A_INVERSE;
    v = t;
  }
  write_char(frame[num].startx, frame[num].starty, C_ULD,  _INVERSE);
  write_char(frame[num].startx, frame[num].endy-1, C_URD,  _INVERSE);
  write_char(frame[num].endx, frame[num].starty, C_LLD, A_ NVERSE);
  write_char(frame[num].endx, frame[num].endy-1, C_LRD, A_ NVERSE);
}
/****************** END OF SCREEN FUNCTION *************** ***/
/*************************************
*         CURSOR FUNCTION            *
*************************************/
/* send the cursor to x,y */
void goto_xy(x,y)
int x,y;
{
  union REGS r;
  r.h.ah=2;  /* cursor addressing function */
  r.h.dl=y;  /* column coordinate */
  r.h.dh=x;  /* row coordinate */
  r.h.bh=0;  /* video page */
  int86(0x10, &r, &r);
}
```

```
void cursor(int row, int col)     / *Sent cursor to position x,y */
{
   union REGS ireg;

   ireg.h.ah = 0x02;              / *Function number 2 */
   ireg.h.bh = getvpage();        /* Page ? */
   ireg.h.dh = row   1;
   ireg.h.dl = col   1;
 / int86(0x10,&ireg,&ireg);
/

void hide_cursor(void)
{
   cursor(26,1);
}
int getvcols(void)
{
  union REGS ireg;
  ireg.h.ah = 0x0f;

  int86(0x10, &ireg, &ireg);
  if(ireg.h.al < 2)
    return 40;                   / *40-column mode */
  if(ireg.h.al > 3 && ireg.h.al != 7)
    return 0;                    / *Return 0 for any graphi   mode */
  else
    return 80;                   / *80-column mode */
}
/ *get current position of cursor */
void find_cursor(int *row, int *col)
{
   union REGS ireg;

                                 / *function 3 */
```

```
  ireg.h.bh = getvpage();      /* Page ?      */
  int86(0x10, &ireg, &ireg);
  *row = (int) ireg.h.dh + 1;
  *col = (int) ireg.h.dl + 1;



/****************************************
*           KEY FUNCTION               *
****************************************/
get_key()
{
 union REGS r;

 r.h.ah = 0;
 return int86(0x16, &r, &r);
}
/* input user's selection */
get_resp(num)
int num;
{
   union inkey {
     char ch[2];
     int i;
   } c;
   int key_choice;
   int x,y;

   arrow_get = 0;
   x = frame[num].startx+1;
   y = frame[num].starty+1;
```

```
do {
  c.i = get_key();
                                          / *is normal key *
  if(c.ch[0]) {
    if(c.ch[0] -- ESCAPE) return -1;
    if(c.ch[0] -- ' ')    break;
    if(c.ch[0] -- ENTER)
     {
       key_geted    = arrow_choice;
/      return arrow_choice;

     key_choice = is_in(frame[num].keys, tolower(c.ch[0]));
     if(key_choice==0) return -10;
     / *reset the selection to normal video */
     goto_xy(x+key_geted, y);
     write_string(x+key_geted, y,
     frame[num].menu[key_geted], A_INVERSE); /* redisplay *
     / *see if it is a hot key */
     key_geted = key_choice-1;
     if(key_geted)
       if(key_geted==frame[num].count) key_geted=0;
       if(key_geted<0) key_geted=frame[num].count-1;
      / *highlight the next selection */
       goto_xy(x+key_geted, y);
       write_string(x+key_geted, y,
       frame[num].menu[key_geted],A_INTENSE);
       hide_cursor();
     ) else {
  /* highlight the next selection  if key_geted = 0 */
         goto_xy(x+key_geted, y);
         write_string(x+key_geted, y,
         frame[num].menu[key_geted],A_INTENSE);
         hide_cursor();
        )
```

```
      arrow_choice=key_geted;
      return key_geted;
    } else {  /* is special key */
/* reset the selection to normal video */
        goto_xy(x+arrow_choice, y);
        write_string(x+arrow_choice, y,
        frame[num].menu[arrow_choice], A_INVERSE); /* redisplay */

  switch(c.ch[1]) {
    case  UARROW:   /* up arrow */
       arrow_choice--;
       break;
    case  DARROW:  /* down arrow */
       arrow_choice++;
       break;
     case  LARROW:                  /* left arrow */
       arrow_get = LARROW;
       return -1;
     case  RARROW:                          /* right arrow */
       arrow_get = RARROW;
       return -1;

   if(arrow_choice==frame[num].count) arrow_choice=0;
   if(arrow_choice<0) arrow_choice=frame[num].count-1;
   key_geted    = arrow_choice;
  /* highlight the next selection */
   goto_xy(x+arrow_choice, y);
   write_string(x+arrow_choice, y,
   frame[num].menu[arrow_choice],A_INTENSE);
   hide_cursor();
  }
 } while(TRUE);
}
```

```
/* input user's selection */
get_resp_sub(num)
int num;
{
   union inkey  {
     char ch[2];
     int i;
   } c;
   int key_choice;
   int x,y;

   arrow_get = 0;
   x = frame[num].startx+1;
   y = frame[num].starty+1;
   do {
     c.i = get_key();
     if(c.ch[0]) {                          /* is normal key */
       if(c.ch[0] == ESCAPE) return -1;
       if(c.ch[0] == ' ')    break;
       if(c.ch[0] == ENTER)
       {
            key_recive= arrow_item;
            return arrow_item;
       }
       key_choice = is_in(frame[num].keys, tolower(c.ch[0]));
       if(key_choice==0) return -10;
       /* reset the selection to normal video */
       goto_xy(x+key_recive, y);
       write_string(x+key_recive, y,
       frame[num].menu[key_recive], A_INVERSE); /* redisplay */
       /* see if it is a hot key */
       key_recive= key_choice-1;
       if(key_recive)  {
```

```
            if(key_recive==frame[num].count) key_recive=0;
            if(key_recive<0) key_recive=frame[num].count-1;
          /* highlight the next selection */
            goto_xy(x+key_recive, y);
            write_string(x+key_recive, y,
            frame[num].menu[key_recive],A_INTENSE);
            hide_cursor();
        } else {
/* highlight the next selection  if key_recive= 0 */
            goto_xy(x+key_recive, y);
            write_string(x+key_recive, y,
            frame[num].menu[key_recive],A_INTENSE);
            hide_cursor();
        }
        arrow_item=key_recive;
        return key_recive;
      } else {  /* is special key */
/* reset the selection to normal video */
            goto_xy(x+arrow_item, y);
            write_string(x+arrow_item, y,
            frame[num].menu[arrow_item], A_INVERSE); /* redisplay */

    switch(c.ch[1]) {
      case  UARROW:   /* up arrow */
            arrow_item--;
            break;
      case  DARROW:  /* down arrow */
            arrow_item++;
            break;
    }
    if(arrow_item==frame[num].count) arrow_item=0;
    if(arrow_item<0) arrow_item=frame[num].count-1;
    key_recive= arrow_item;
```

```
    /* highlight the next selection */
    goto_xy(x+arrow_item, y);
    write_string(x+arrow_item, y,
    frame[num].menu[arrow_item],A_INTENSE);
          hide_cursor();
    }
   } while(TRUE);
}
/* input user's selection */
get_resp_sub1(num)
int num;
{
    union inkey  {
      char ch[2];
      int i;
    } c;
    int key_choice;
    int x,y;

    arrow_get = 0;
    x = frame[num].startx+1;
    y = frame[num].starty+1;
    do {
      c.i = get_key();
      if(c.ch[0]) {                          /* is normal key */
        if(c.ch[0] == ESCAPE) return -1;
        if(c.ch[0] == ' ')    break;
        if(c.ch[0] == ENTER)
        {
             key_recive1= arrow_item1;
             return arrow_item1;
        }
        key_choice = is_in(frame[num].keys, tolower(c.ch[0]));
```

```
    if(key_choice==0) return -10;
    /* reset the selection to normal video */
    goto_xy(x+key_recive1, y);
    write_string(x+key_recive1, y,
    frame[num].menu[key_recive1], A_INVERSE); /* redisplay */
    /* see if it is a hot key */
    key_recive1= key_choice-1;
    if(key_recive1)  {
        if(key_recive1==frame[num].count) key_recive1=0;
        if(key_recive1<0) key_recive1=frame[num].count-1;
        /* highlight the next selection */
        goto_xy(x+key_recive1, y);
        write_string(x+key_recive1, y,
        frame[num].menu[key_recive1],A_INTENSE);
        hide_cursor();
    } else {
        /* highlight the next selection  if key_recive= 0 */
        goto_xy(x+key_recive1, y);
        write_string(x+key_recive1, y,
        frame[num].menu[key_recive1],A_INTENSE);
        hide_cursor();
    }
    arrow_item1=key_recive1;
    return key_recive1;
  } else {  /* is special key */
/* reset the selection to normal video */
goto_xy(x+arrow_item1, y);
write_string(x+arrow_item1, y,
frame[num].menu[arrow_item1], A_INVERSE); /* redisplay */
switch(c.ch[1]) {
  case  UARROW:   /* up arrow */
        arrow_item1--;
        break;
```

```
      case  DARROW:  /* down arrow */
            arrow_item1++;
            break;
    }
    if(arrow_item1==frame[num].count) arrow_item1=0;
    if(arrow_item1<0) arrow_item1=frame[num].count-1;
    key_recive1= arrow_item1;
    /* highlight the next selection */
    goto_xy(x+arrow_item1, y);
    write_string(x+arrow_item1, y,
    frame[num].menu[arrow_item1],A_INTENSE);
    hide_cursor();
   }
  } while(TRUE);
}
/********** END OF KEY FUNCTION ***********/

is_in(s, c)
char *s, c;
{
  register int i;

  for(i=0; *s; i++)
   if(*s++==c) return i+1;
  return 0;
}
/*************************
*  Window functions      *
*************************/

/* Display a pull-down window. */
void window_(num)
int num; /* window number */
```

```
{
  int  vmode, choice;
  int x, y;

  vmode   video_mode();
  if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
    printf("video must be in 80 column text mode");
    exit(1);
  }
  /*  set proper address of video RAM  */
  if(vmode==7) vid_mem = (char far *) 0xb0000000;
  else vid_mem = (char far *) 0xb8000000;

  /* get active window */
  if(!frame_win[num].active) { /* not currently in use */
    save_video_win(num);       /* save the current screen */
    frame_win[num].active = 1; /* set active flag */
  }

  if(frame_win[num].border) draw_border1(num);
  display_header(num); /* display the window */

  x = frame_win[num].startx + frame_win[num].curx + 1;
  y = frame_win[num].starty + frame_win[num].cury + 1;
  goto_xy(x, y);
}

/* Construct a pull down window frame
   1 is returned if window frame can be constructed;
   otherwise 0 is returned.
*/

make_window(num, header, startx, starty, endx, endy, border)
```

```
int num;        /* window number */
char *header;   /* hwader text */
int startx, starty; /* X,Y coordinates of upper left corner */
int endx, endy;  /* X,Y coordinates of lower right corner */
int border;     /* no borde if 0 */
{
  unsigned char *p;

  if(num>MAX_FRAME) {
    printf("Too many windows\n");
    return 0;
  }
  if((startx>24) || (startx<0) || (starty>78) || (starty<0)) {
    printf("range error");
    return 0;
  }
  if((endx>24) || (endy>79)) {
    printf("window won't fit");
    return 0;
  }
  /* allocate enough memory to hold it */
  p = (unsigned char *) malloc(2*(endx-startx+1)*(endy-starty+1));
  if(!p) exit(1); /* put your own error handler here */
  /* construct the frame */
  frame_win[num].startx = startx; frame_win[num].endx = endx;
  frame_win[num].starty = starty; frame_win[num].endy = endy;
  frame_win[num].p = p;
  frame_win[num].header = header;
  frame_win[num].border = border;
  frame_win[num].active = 0;
  frame_win[num].curx = 0; frame_win[num].cury = 0;
  return 1;
}
```

```
/* Deactivate a window and removw it from the screen. */
deactivate_win(num)
int num;
{
  /* reset the cursor postion to upper left corner */
  frame_win[num].curx = 0;
  frame_win[num].cury = 0;
  restore_video_win(num);
}

/* Display the header message in its proper location. */
display_header(num)
int num;
{
  register int y, len;

  y = frame_win[num].starty;

  /* calculate the correct starting position to center
     the header message - if negative, message won't
     fit.
  */
  len = strlen(frame_win[num].header);
  len = (frame_win[num].endy - y - len) / 2;
  if(len<0) return; /* don't display it */
  y = y + len;

  write_string(frame_win[num].startx, y,
       frame_win[num].header, A_INVERSE);
}

draw_border1(num)
int num;
```

```
{
  register int i;
  char far *v, far *t;

  v = vid_mem;
  t = v;
  for(i=frame_win[num].startx+1; i<frame_win[num].endx; i++) {
     v += (i*160) + frame_win[num].starty*2;
     *v++ = C_VD;
     *v = A_INVERSE;
     v = t;
     v += (i*160) + frame_win[num].endy*2;
     *v++ = C_VD;
     *v = A_INVERSE;
     v = t;
  }
  for(i=frame_win[num].starty+1; i<frame_win[num].endy; i++) {
    v += (frame_win[num].startx*160) + i*2;
    *v++ = C_HD;
    *v = A_INVERSE;
    v = t;
    v += (frame_win[num].endx*160) + i*2;
    *v++ = C_HD;
    *v = A_INVERSE;
    v = t;
  }
  write_char(frame_win[num].startx, frame_win[num].starty, C_ULD,
   A_INVERSE);
  write_char(frame_win[num].startx, frame_win[num].endy, C_URD,
   A_INVERSE);
  write_char(frame_win[num].endx, frame_win[num].starty, C_LLD,
   A_INVERSE);
  write_char(frame_win[num].endx, frame_win[num].endy, C_LRD,
```

```
    A_INVERSE);
}
/*****************************************************************
*  Window I/O functions                                          *
*                                                                *
*  Write a string at the current cursor position                 *
*  in the specified wnidow.                                      *
*  Returns 0 if window not active;                               *
*****************************************************************/

window_puts(num, str,attr)
int num,attr;
char *str;
{
   /* make sure window is active */
   if(!frame_win[num].active) return 0;

   for(; *str; str++)
     window_putchar(num,*str,attr);
   return 1;
}

/*****************************************************************
*  Window I/O functions                                          *
*                                                                *
*  Write a character at the current cursor position              *
*  in the specified window.                                      *
*  Returns 0 if window not active;                               *
*  1 otherwise.                                                  *
*****************************************************************/

window_putchar(num, ch, at)
int num,at;
```

```
char ch;
{
  register int x, y;
  char far *v;

  /* make sure window is active */
  if(!frame_win[num].active) return 0;
  x = frame_win[num].curx + frame_win[num].startx + 1;
  y = frame_win[num].cury + frame_win[num].starty + 1;
  v = vid_mem;
  v += (x*160) + y*2; /* compute the address */
  if(y>=frame_win[num].endy) {
    return 1;
  }
  if(x>=frame_win[num].endx) {
    return 1;
  }
  if(ch=='\n') { /* newline char */
    x++;
    y = frame_win[num].startx+1;
    v = vid_mem;
    v += (x*160) + y*2; /* compute the address */
    frame_win[num].curx++; /* increment X */
    frame_win[num].cury = 0; /* reset Y */
  }
  else {
    frame_win[num].cury++;
    *v++ = ch;  /* write the character */
    *v++ = at;  /* normal video attribute */
  }
  window_xy(num, frame_win[num].curx, frame_win[num].cury);
  return 1;
}
```

```
/*****************************************************************
*  Window I/O functions                                          *
*                                                                *
*  Position cursor in a window at specified location.            *
*  Returns 0 if out of range;                                    *
*  non-zero otherwise.                                           *
*****************************************************************/

window_xy(num, x, y)
int num, x, y;
{
  if(x<0 || x+frame_win[num].startx>=frame_win[num].endx-1)
    return 0;
  if(y<0 || y+frame_win[num].starty>=frame_win[num].endy-1)
    return 0;
  frame_win[num].curx = x;
  frame_win[num].cury = y;
  goto_xy(frame_win[num].startx+x+1, frame_win[num].starty+y+1);
  return 1;
}

/*********************************
* Read a string from a window.  *
*********************************/
void window_gets(num, s)
int num;
char *s;
{
  char ch, *temp;

  temp = s;
  for(;;) {
    ch = window_getche(num);
```

```
      switch(ch) {
        case ESCAPE :
            *s = -1;
            return;
        case ENTER  :  /* the ENTER key is pressed */
            *s='\0';
            return;
       case BKSP   : /* backspace */
            if(s>temp) {
              s--; frame_win[num].cury--;
              if(frame_win[num].cury<0) frame_win[num].cury = 0;
              window_xy(num, frame_win[num].curx, frame_win[num].cury);
              write_char(frame_win[num].startx+ frame_win[num].curx+1,
              frame_win[num].starty+frame_win[num].cury+1, ' ',
              A_INVERSE);
            }
            break;
       default   : *s = ch;
            s++;
      }
    }
  }
/*************************************
* Input keystrokes inside a window. *
* Returns full 16 bit scan code.    *
*************************************/
window_getche(num)
int num;
  {
    union inkey {
      char ch[2];
      int i;
    } c;
```

```
    if(!frame_win[num].active) return 0; /* window no active */
    window_xy(num, frame_win[num].curx, frame_win[num].cury);
    c.i = bioskey(0);     /* read the key */
    if(c.ch[0]) {
      switch(c.ch[0]) {
          case ENTER: /* the ENTER key is pressed */
            break;
          case BKSP: /* back space */
            break;
          default:
            if(frame_win[num].cury+frame_win[num].starty <
            frame_win[num].endy-1) {
              write_char(frame_win[num].startx+frame_win[num].curx+1,
              frame_win[num].starty+frame_win[num].cury+1, c.ch[0],
              A_INVERSE);
              frame_win[num].cury++;
            }
      }
      if(frame_win[num].curx < 0) frame_win[num].curx = 0;
      if(frame_win[num].curx+frame_win[num].startx >
         frame_win[num].endx-2)
      frame_win[num].curx--;
      window_xy(num, frame_win[num].curx, frame_win[num].cury);
    }
    return c.i;
}

/******************
* Clear a window. *
******************/
void window_cls(num)
int num;
{
```

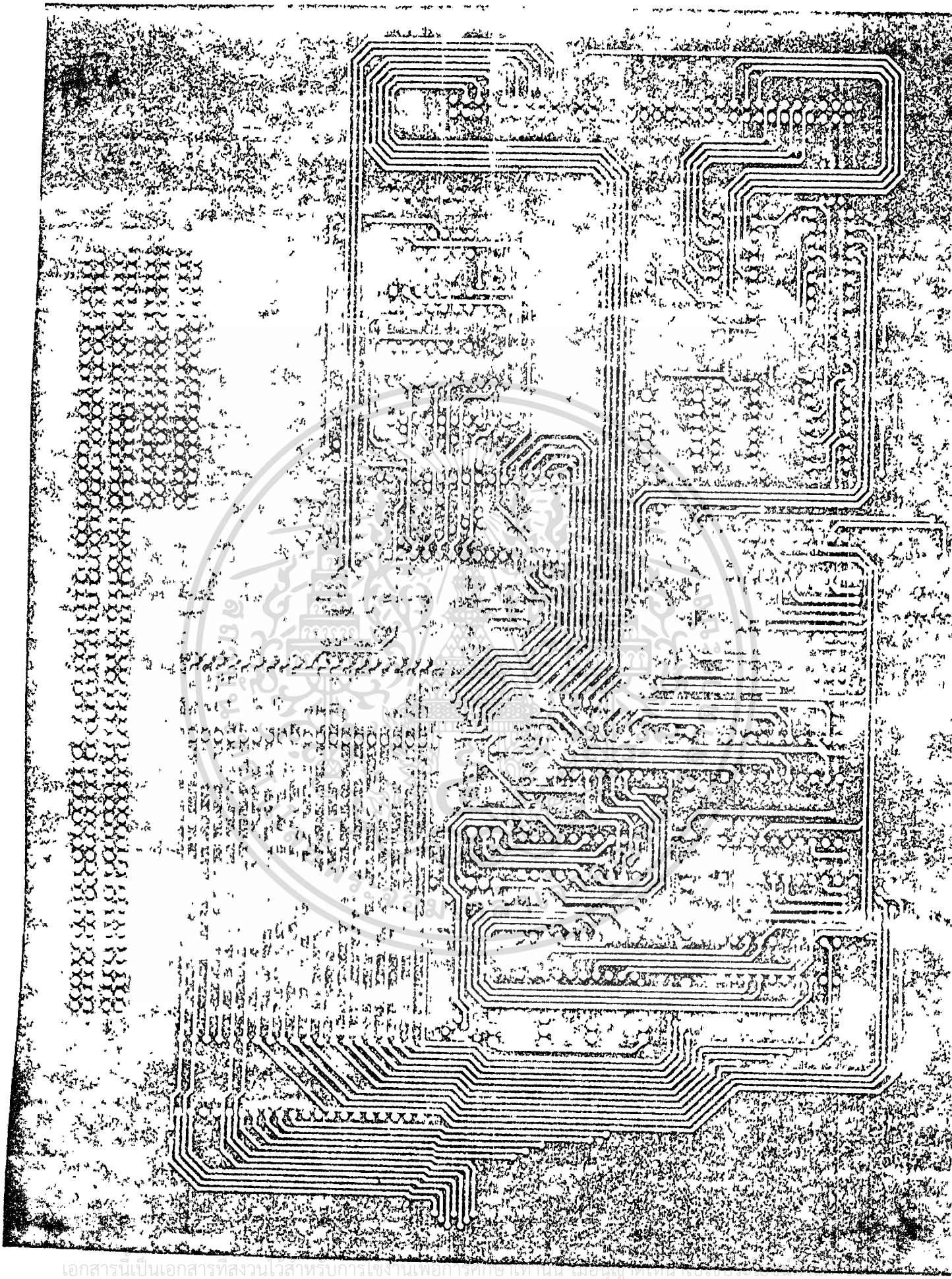
```
    register int i,j;
    char far *v, far *t;

    v = vid_mem;
    t = v;
    for(i=frame_win[num].starty+1; i<frame_win[num].endy; i++)
      for(j=frame_win[num].startx+1; j<frame_win[num].endx; j++) {
        v = t;
        v += (j*160) + i*2;
        *v++ = ' '; /* write a space */
        *v = A_INVERSE;  /* normal */
    }
    frame_win[num].curx = 0;
    frame_win[num].cury = 0;
}

/*************************
* Clear to end of line. *
*************************/
void window_cleol(num)
int num;
{
  register int i, x, y;

  x = frame_win[num].curx;
  y = frame_win[num].cury;
  window_xy(num, frame_win[num].curx, frame_win[num].cury);

  for(i=frame_win[num].cury; i<frame_win[num].endy-1; i++)
    window_putchar(num,' ',A_INVERSE);
  window_xy(num, x, y);
}
```

```
/************************************
*  Move cursur up one line.        *
*  Returns non-zero if successful; *
*  0 otherwise.                    *
************************************/
window_upline(num)
int num;
{
  if(frame_win[num].curx>0) {
    frame_win[num].curx--;
    window_xy(num, frame_win[num].curx, frame_win[num].cury);
    return 1;
  }
  return 0;
}
/************************************
* Move cursor down one line.       *
*  Returns non-zero if successful; *
*  0 otherwise.                    *
************************************/
window_downline(num)
int num;
{
  if(frame_win[num].curx<frame_win[num].endx-frame_win[num].startx-1)
  {
    frame_win[num].curx++;
    window_xy(num, frame_win[num].curx, frame_win[num].cury);
    return 1;
  }
  return 0;
}
```

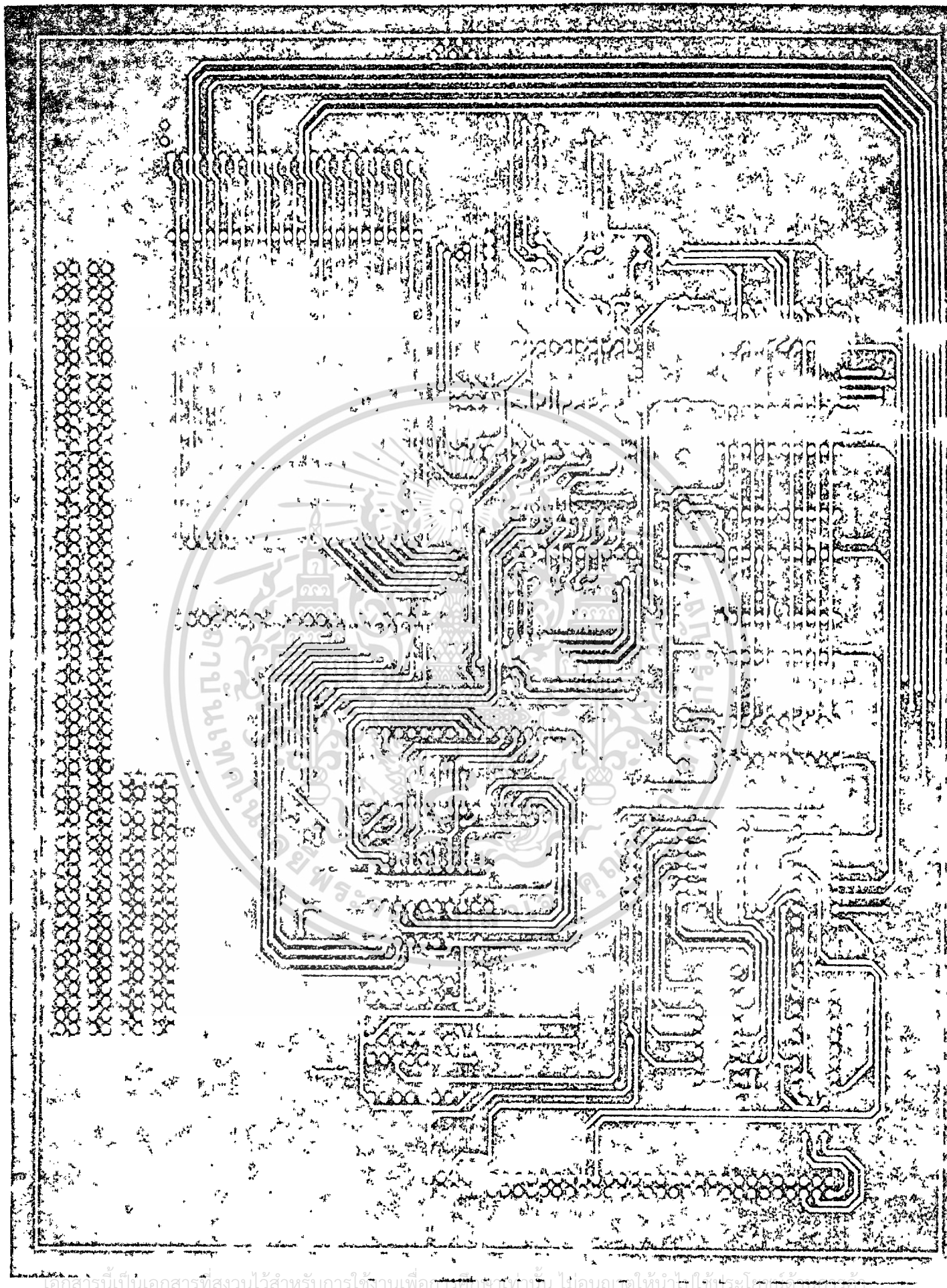
```
/*************************
* back up one character. *
*************************/
window_bksp(num)
int num;
{
  if(frame_win[num].cury>0) {
    frame_win[num].cury--;
    window_xy(num, frame_win[num].curx, frame_win[num].cury);
    window_putchar(num, ' ');
    frame_win[num].cury--;
    window_xy(num, frame_win[num].curx, frame_win[num].cury);
  }
}
/*********************************
* save a portion of the screen *
*********************************/
void save_video_win(num)
int num;
{
  register int i,j;
  char *buf_ptr;
  char far *v, far *t;
  buf_ptr = frame_win[num].p;
  v = vid_mem;
  for(i=frame_win[num].starty; i<frame_win[num].endy+1; i++)
    for(j=frame_win[num].startx; j<frame_win[num].endx+1; j++) {
      t = v + (j*160) + i*2;  /* compute the address  */
      *buf_ptr++ = *t++;  /* read the character  */
      *buf_ptr++ = *t;    /* read the attribute  */
      *(t-1) = ' ';  /* clear the window  */
    }
}
```

```
/************************************
* restore a portion of the screen  *
************************************/
void restore_video_win(num)
int num;
{
  register int i,j;
  char far *v, far *t;
  char *buf_ptr;
  buf_ptr = frame_win[num].p;
  v = vid_mem; t = v;
  for(i=frame_win[num].starty; i<frame_win[num].endy+1; i++)
    for(j=frame_win[num].startx; j<frame_win[num].endx+1; j++) {
      v = t;
      v += (j*160) + i*2;  /* compute the address  */
      *v++ = *buf_ptr++;  /*  write the character  */
      *v = *buf_ptr++;     /* write the attribute  */
    }
    frame_win[num].active = 0; /* deactivate */
}
/*********************************************************
* Return the position code of arrow and function keys. *
*********************************************************/
get_special()
{
  union inkey {
    char ch[2];
    int i;
  } c;
  while(!bioskey(1)); /* wait for key stroke */
  c.i = bioskey(0);   /* read the key */
  return c.ch[1];
}
```

แสดงถึงด้านลายทองแดงของแผ่นวงจรพิมพ์ ทั้งด้านหน้าและด้านล่างของของเครื่อง 8085 EMULATOR ซึ่งแสดงได้ดังรูปที่ ผ-3.1 และ ผ-3.2 ในหน้าถัดไป



รูปที่ ผ-3.1 แสดงลายทองแดงด้านบน



รูปที่ ผ-3.2 แสดงลายทองแดงด้านล่าง

กิตติกรรมประกาศ

การทดลองวิจัยและสร้าง อีมูเลเตอร์ 8085 นี้ได้รับความกรุณาและช่วยเหลือเป็นอย่างดีในการจัดหาอุปกรณ์ เครื่องมือพร้อมแหล่งข้อมูล และคำปรึกษาจากท่านอาจารย์ สิงห์ทอง พัฒนเศรษฐานนท์ อาจารย์ที่ปรึกษา นอกจากนี้ยังได้รับคำแนะนำในการวางหลักการของโครงงานนี้จาก คุณนิพนธ์ ศิริรัตน์ และการอำนวยความสะดวกอีกหลายประการจากเจ้าหน้าที่ และอาจารย์ประจำภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม ตลอดจนรุ่นพี่และเพื่อน ๆ อีกหลายคน

จึงขอขอบคุณมา ณ ที่นี้ ที่มีส่วนช่วยให้งานบริญญานิพนธ์ครั้งนี้สำเร็จลุล่วงไปด้วยดี

คณะผู้จัดทำ

กิตติกรรมประกาศ

การทดลองวิจัยและสร้าง อีมูเลเตอร์ 8085 นี้ได้รับความกรุณาและช่วยเหลือเป็นอย่างดีในการจัดหาอุปกรณ์ เครื่องมือพร้อมแหล่งข้อมูล และคำปรึกษาจากท่านอาจารย์ สิงห์ทอง พัฒนเศรษฐานนท์ อาจารย์ที่ปรึกษา นอกจากนี้ยังได้รับคำแนะนำในการวางหลักการของโครงงานนี้จาก คุณนิพนธ์ ศิริรัตน์ และการอำนวยความสะดวกอีกหลายประการจากเจ้าหน้าที่ และอาจารย์ประจำภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม ตลอดจนรุ่นพี่และเพื่อน ๆ อีกหลายคน

จึงขอขอบคุณมา ณ ที่นี้ ที่มีส่วนช่วยให้งานปริญญานิพนธ์ครั้งนี้สำเร็จลุล่วงไปด้วยดี

คณะผู้จัดทำ

เอกสารอ้างอิง

1. ธานินทร์ ถาวรศาสนวงศ์,ทินกร ดุ๊ก,"การอินเตอร์เฟส IBM/PC",พิสิกส์เซ็นเตอร์การพิมพ์
2. ไมโครคอมพิวเตอร์ ฉบับพิเศษ ยูทิลิตี้",บริษัท ซีเอ็ดยูเคชั่น จำกัด
3. ยืน ภู่วรรวณ,ทฤษฎีและการประยุกต์ใช้งาน Z-80,บริษัท ซีเอ็ดยูเคชั่น จำกัด
4. Ramirez/Weiss ," MICROPROCESSING FUNDAMENTALS HARDWARE AND SOFTWARE"
5. DAVID LALOND, "The 8080, 8085 and Z80 hardware, software, programing interfacing and Troubleshooting",prentiec-Hall international
6. Elmer C.Poe, "The MICROPROCESSOR HAND BOOK",Howard W. Sams & Co.,Inc
7. Jack Purdun,"C Programer's Toolkit ",Que Corporation,1989
8. Herbert Schildt, "C: Power User's Guide",Osborne McGraw-Hill, 1988
9. Herbert Schildt, " ADVANCE C Second Edition ",Osborne Mcgraw Hill,1988
10. Brian W.kernighan,Dennis M.Ritchie,"The C PROGRAMMING LANGUAGE ",Prentiec-Hall,inc