ปีการศึกษา 2530

เครื่องพัฒนาระบบคอมพิวเตอร์

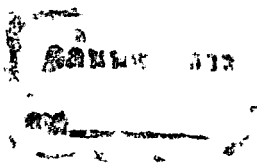
โดย

นาย บัญชา วัฒนโสภณวงศ์

นาย สันติเดช แม้นสวัสดิ์

อาจารย์ที่ปรึกษา

อาจารย์ วิริยะ กองรัตน์



ปริญญานิพนธ์ปีการศึกษา 2530

เรื่อง เครื่องพัฒนาระบบคอมพิวเตอร์

ผู้จัดทำ

1. นาย บัญชา วัฒนโสภณวงศ์

2. นาย สันติเดช แม้นสวัสดิ์

อาจารย์ วิริยะ กองรัตน์ อาจารย์ที่ปรึกษา

(.....)

เครื่องพัฒนาระบบไมโครโปรเซสเซอร์

นาย บัญชา วัฒนโสภณวงศ์

นาย สันติเดช แม้นสวัสดิ์

อาจารย์ วิริยะ กองรัตน์ อาจารย์ที่ปรึกษา

ปีการศึกษา 2530

บทคัดย่อ

การพัฒนาระบบคอมพิวเตอร์ในงานทั่ว ๆ ไป มักจะประสพปัญหา ใจส่วนของการเตรียมข้อมูลหรือโปรแกรม เพื่อที่จะให้ระบบที่ต้องการจะพัฒนา มีการดำเนินการตามขั้นตอนต่าง ๆ ของผู้ใช้ โดยทั่วไปขั้นตอนต่างๆจะถูกกระทำบนเครื่องไมโครคอมพิวเตอร์ และนำข้อมูลที่ได้เก็บลงบนหน่วยความจำแบบใหแบบหนึ่งเพื่อ นำมาทดลองกับระบบ

ในกรณีที่มีการผิดพลาดของการดำเนินงานแล้วจะต้องมีการเปลี่ยนแปลงหรือเตรียมข้อมูลใหม่ ทำให้เกิดความไม่คล่องตัวในการทดลอง

วิทยานิพนธ์ฉบับนี้ได้นำเสนอ ระบบช่วยพัฒนาคอมพิวเตอร์ กล่าวคือ การเตรียมข้อมูลจะถูกกระทำบนเครื่องไมโครคอมพิวเตอร์และทำการส่งผ่านข้อมูลแบบอนุกรม (Serial Communication) แบบ อาร์เอส 232 ซี (RS-232C) และยังสามารถที่จะตรวจสอบ แก้ไข เพิ่มเติมค่าในหน่วยความจำ ดูสถานะของการทำงานของแต่ละคำสั่ง หรือสั่งให้มีการทำงานในส่วนใดๆของโปรแกรมในระบบได้ ซึ่งการกระทำตามขั้นตอนดังกล่าวมีการตอบรับกับเครื่องไมณครคอมพิวเตอร์ตลอดเวลา

THE MICROCOMPUTER DEVELOPMENT SYSTEM

BANCHA WATANASOPONWONG

SANTIDEJ MANSAWASDI

VIRIYA KONGRATANA ADVISOR

Abstract

The development of Microcomputer system usually encounters the problem of the data entry. Generally,this procedure are done on computer and the data is stored in memory under tested system. In case of system operation failure,it is necessary to entry new data, which cause inconvenience during operation.

This paper presents,the microcomputer development system. The data is sent via serial communication interface. This system can test and improve the data in memory unit, etc.

สารบัญ

บทที่ 1. บทนำ.....1

บทที่ 2. หลักการทำงาน.....4

ความสามารถของระบบ

1.ในเรื่องการดูข้อมูลขณะโปรแกรมกำลังทำงานอยู่.....5

2.ในเรื่องการบันทึกข้อมูลภาษาเครื่องลงแผ่นดิสก์.....8

3.ในเรื่องของการเอาข้อมูลภาษาเครื่องลงสู่หน่วยความจำ.....9

4.ในเรื่องของการแก้ไขข้อมูลในหน่วยความจำ.....10

5.ในเรื่องของการแสดงแฟ้มข้อมูล.....13

บทที่ 3. การสร้าง.....14

โครงสร้างของเครื่องพัฒนาระบบคอมพิวเตอร์.....14

1.ส่วนฮาร์ดแวร์.....14

1.1.ส่วนจัดการ(ชิงเกิ้ลบอร์ด).....14

-ส่วนของแรม.....16

-ส่วนของรอม.....17

-ส่วนของไอโอพอร์ท.....17

-ส่วนของยูซาส.....20

1.2.ส่วนอินพุตเอาท์พุต(เครื่องไอบีเอ็ม).....24

2.ส่วนซอฟแวร์.....24

-ส่วนซอฟแวร์ภาษาปาสคาล.....24

-ส่วนซอฟแวร์ภาษาแอสแซมบลี้.....36

บทที่ 4. การทดลองและผลการทดลอง.....45

บทที่ 5. บทวิจารณ์และบทสรุป.....46

ภาคผนวก.....47

กิตติกรรมประกาศ

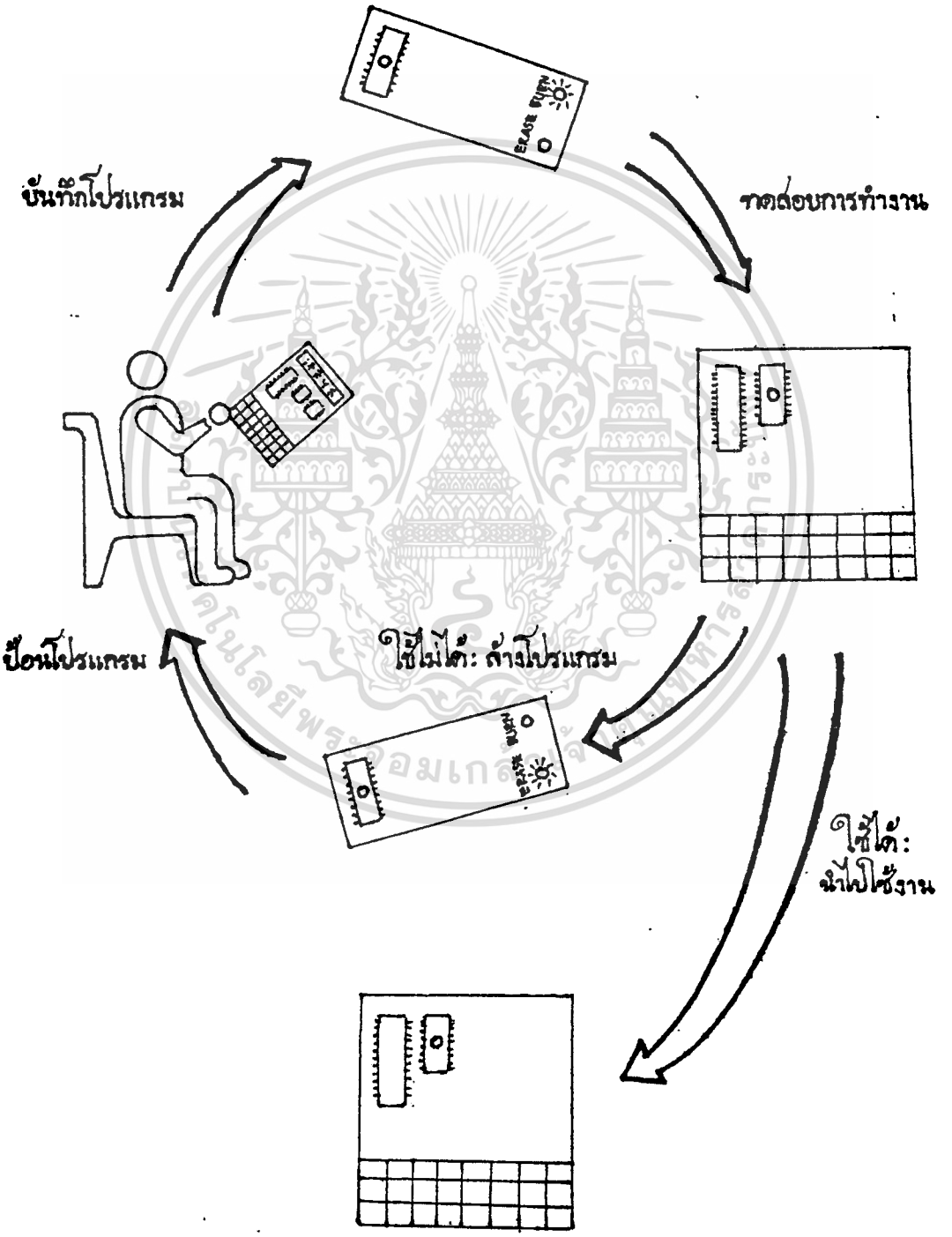
หนังสืออ้างอิง

บทที่ 1

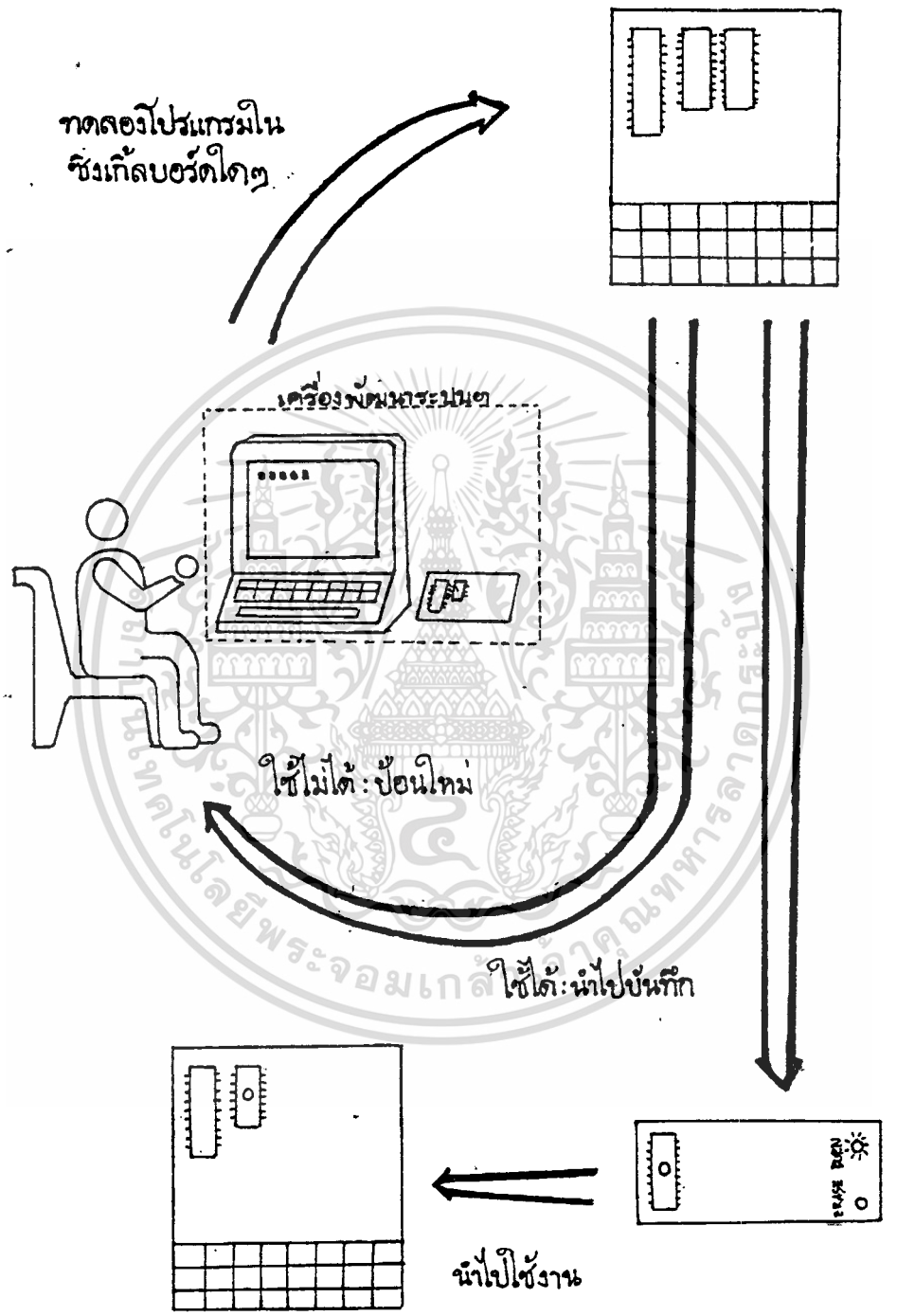
บทนำ

ก่อนอื่นขอกล่าวถึงการพัฒนามอนิเตอร์โปรแกรม (MONITOR PROGRAM) ของระบบคอมพิวเตอร์ที่ใช้ไมโครโปรเซสเซอร์เป็นหน่วยประมวลผลกลาง (CENTRAL PROCESSING UNIT : CPU) ซึ่งไมโครโปรเซสเซอร์ขนาด 8 บิทที่ใช้กันอย่างแพร่หลายกันที่สุด ก็เห็นจะเป็น เบอร์ Z-80 บริษัทผู้ผลิตก็คือ บริษัท ไซล็อค (ZILOG CORPORATION) โดยที่บริษัทนี้ก็เป็นทีมงานที่แยกตัวมาจาก บริษัท อินเทล (INTEL CORPORATION) ผู้ผลิตไมโครโปรเซสเซอร์ขนาด 8 บิท เบอร์ 8080 แต่เพราะเหตุที่ว่า Z-80 มีคำสั่งมากกว่าและมีประสิทธิภาพดีกว่า จึงเป็นที่นิยมใช้กันอย่างแพร่หลายดังที่ได้กล่าวมาแล้ว ที่ผ่านๆ มานั้นการพัฒนามอนิเตอร์โปรแกรม สามารถทำได้โดย การเขียนโปรแกรมด้วยภาษาแอสแซมบลี้แล้วนำโปรแกรมดังกล่าวไปแปลเป็นภาษาเครื่อง (MACHINE LANGUAGE) ขั้นตอนต่อไปก็คือ นำเอาโปรแกรมภาษาเครื่องนี้ไปบันทึก (BURN) ลงในหน่วยความจำ ประเภทนอนโวล์ลาไทน์ (NONVOLATILE) เช่น พรอม (PROGRAMABLE READ ONLY MEMORY : PROM),อีพรอม (ERASABLE PROGRAMABLE READ ONLY MEMORY : EPROM) จากนั้นจึงนำไปทดลองทำงานดู ถ้าหากมีการผิดพลาด (BUG) เกิดขึ้นหรือยังไม่เป็นที่น่าพอใจของโปรแกรมเมอร์ (PROGRAMER) ก็จะต้องเริ่มทำใหม่ตั้งแต่ต้นตามที่กล่าวมา จนกระทั่งการทำงานของโปรแกรมเป็นที่น่าพอใจหรือไม่มีข้อผิดพลาดเกิดขึ้น อีกประการหนึ่งทั้งยังทำการตรวจสอบหาข้อผิดพลาด (DEBUG) ที่เกิดขึ้นได้ยากอีกด้วย ทำให้เสียเวลา ซึ่งวิธีการแบบเก่าข้างต้นนี้ ก่อให้เกิดความยุ่งยากและไม่สะดวกต่อการพัฒนามอนิเตอร์โปรแกรมเป็นอย่างมากและด้วยเหตุนี้เอง เพื่อที่จะขจัดความยุ่งยากและความไม่สะดวกที่เกิดขึ้นดังกล่าวให้หมดสิ้นไป จึงน่าจะเป็นการที่ดีกว่าที่จะสามารถ พัฒนามอนิเตอร์โปรแกรมไปพร้อมๆ กันกับการทดสอบการทำงานของมอนิเตอร์โปรแกรมไปด้วย เมื่อทำการพัฒนามอนิเตอร์โปรแกรมจนเป็นที่น่าพอใจแล้ว ก็ค่อย

นำเอามอนิเตอร์โปรแกรมที่เป็นภาษาเครื่องแล้วนี้ ไปบันทึกลงในหน่วยความจำดังกล่าวเพียงครั้งเดียว



รูปที่ 1.1 แสดงการพัฒนามอนิเตอร์โปรแกรมด้วย วิธีเก่า

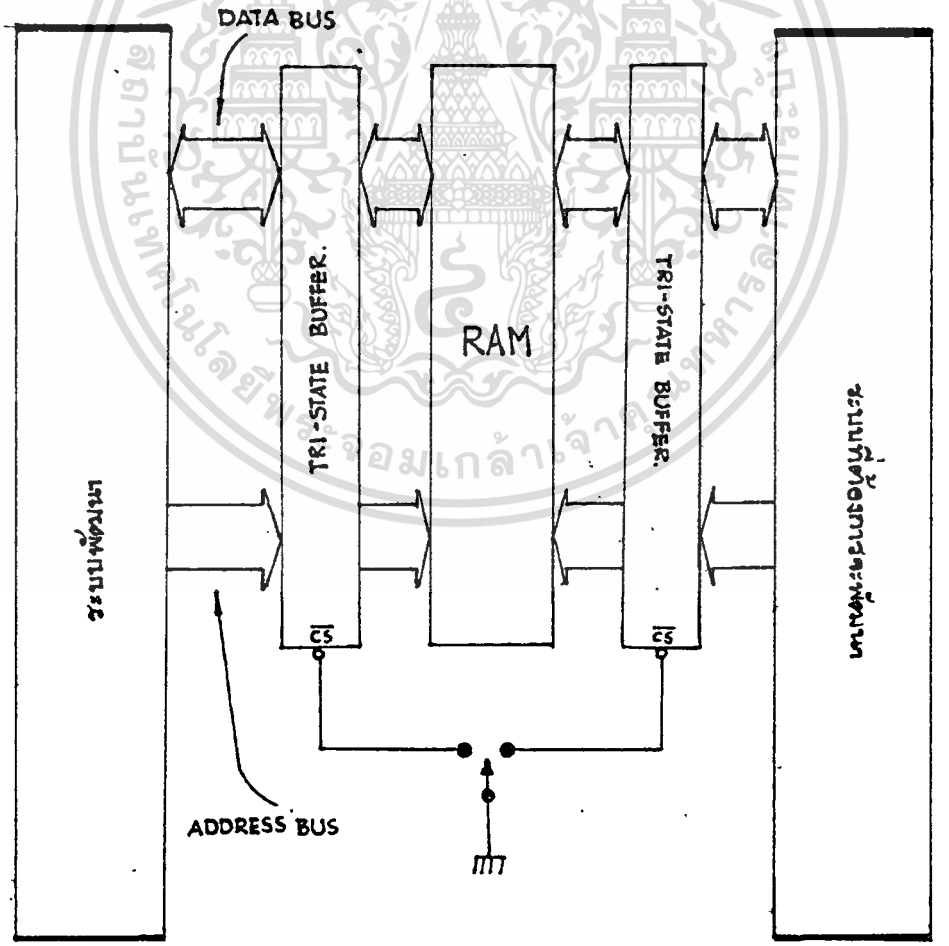


รูปที่ 1.2 แสดงการพัฒนาอินเตอร์โปรแกรมด้วย วิธีใหม่

บทที่ 2

หลักการทำงานอย่างกว้างๆ ของเครื่องพัฒนาระบบคอมพิวเตอร์

จากบทนำจะเห็นได้ว่า เพื่อให้บรรลุวัตถุประสงค์ดังที่กล่าวมานั้น จำเป็นอย่างยิ่งที่จะต้องมีตัวกลางตัวหนึ่ง ที่คอยทำหน้าที่ในการอำนวยความสะดวกต่อการพัฒนามอนิเตอร์โปรแกรม วิธีการหนึ่งที่ใช้กันอยู่ทั่วไปก็คือ การใช้หน่วยความจำชนิดแรม 2 ทาง ดังที่แสดงไว้ในรูปที่ 2.1



รูปที่ 2.1 การใช้หน่วยความจำชนิดแรม 2 ทาง.

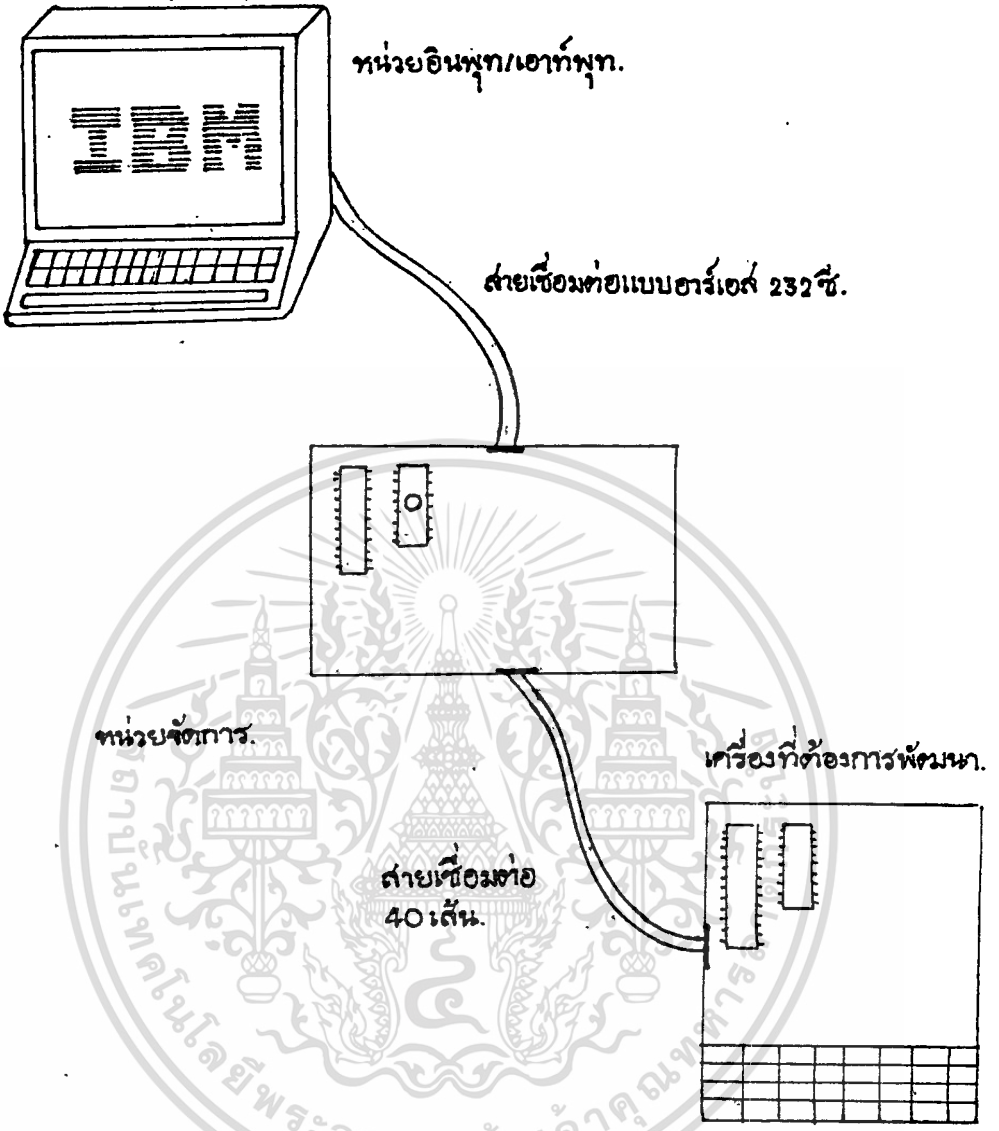
จากรูปจะเห็นได้ว่ามีทางหนึ่งที่ต่อเข้ากับ เครื่องที่ต้องการจะพัฒนา โดยเป็นหน่วยความจำของเครื่องนั้นๆ และอีกทางหนึ่งจะต่อเข้ากับเครื่องมือใดๆ ที่สามารถจะเปลี่ยนแปลงข้อมูลในตำแหน่งแอดเดรสต่างๆ ของหน่วยความจำนั้นได้ ซึ่งเครื่องมือนั้นอาจจะเป็น คอมพิวเตอร์-เทอมินัล แต่วิธีการนี้ไม่อ่อนตัว (FLEXIBLE) ในการใช้งานมากนัก เพราะว่ามีข้อจำกัดคือ ข้อจำกัดที่เกิดจากความแตกต่างกันด้านฮาร์ดแวร์ (HARDWARE) และยังไม่มีฟังค์ชั่นต่างๆที่จะคอยอำนวยความสะดวกต่อผู้ใช้ เช่น ไม่สามารถแสดงผลข้อมูลที่มีการเปลี่ยนแปลงตลอดเวลาได้ เป็นต้น แต่ก็ยังมีวิธีอีกวิธีหนึ่งที่การใช้งานมีความอ่อนตัวกว่า วิธีการนี้เป็นวิธีการที่มีการทำงานค่อนข้างอ่อนตัว หรืออาจจะกล่าวได้ว่าอำนวยความสะดวกแก่ผู้ใช้มากกว่า กล่าวคือ มีความอ่อนตัวต่อการใช้งาน เป็นเพราะว่า เหียงแต่มีเครื่องคอมพิวเตอร์ของไอบีเอ็ม (IBM COMPUTER) หรือเครื่องที่เข้ากันได้ (COMPATIBLE) ซึ่งจะถูกใช้เป็นหน่วยอินพุต/เอาท์พุต (INPUT OUT UNIT) ของเครื่องพัฒนาระบบคอมพิวเตอร์ โดยมีคุณสมบัติดังนี้คือ

1. สามารถที่จะทำงานด้วยโปรแกรมปาสคาลได้

2. จะต้องมีพอร์ท (PORT) สำหรับการสื่อสารแบบ อนุกรม ที่เป็นไปตามมาตรฐาน อาเอส 232-ซี (RS 232-C SERIAL COMMUNICATION STANDARD) ของ สมาคมผู้ผลิตอุตสาหกรรมทางอิเลคทรอนิคส์ (EIA.: ELECTRONICS INDUSTRY ASSOCIATION)

และยังมีฟังค์ชั่นต่างๆ อีกมากมายในการอำนวยความสะดวก ซึ่งนอกจากนี้แล้วยังสามารถแสดงผลข้อมูลที่มีการเปลี่ยนแปลงตลอดเวลาได้อีกด้วย และหลักการทำงานของเครื่องพัฒนาระบบคอมพิวเตอร์นั้น เนื่องจากว่า ประกอบด้วยฟังค์ชั่นการทำงานหลายอย่างจึงจะอธิบายหลักการของแต่ละฟังค์ชั่นเป็นหลักการรวมของเครื่อง ซึ่งสามารถอ้างได้จากโครงสร้างในรูปที่ 2.2 จะแบ่งความสามารถของ เครื่องพัฒนาระบบคอมพิวเตอร์ นี้ออกตามหน้าที่ของฟังค์ชั่นต่างๆได้เป็น 5 ฟังค์ชั่นสำคัญๆ ดังนี้

1. ความสามารถในเรื่องของ การดูข้อมูลขณะโปรแกรมกำลังทำงานอยู่ (MONITORING FUNCTION)

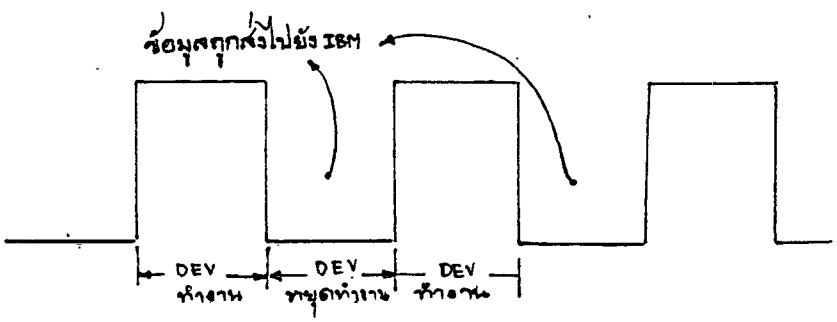


รูปที่ 2.2 ลักษณะโครงสร้างของเครื่องพัฒนาระบบคอมพิวเตอร์.

ในการทำงานของโปรแกรมใดๆ นั้น ส่วนมากจะมีการเปลี่ยนแปลงข้อมูลในหน่วยความจำอยู่เสมอ ถ้าหากว่าการเปลี่ยนแปลงของข้อมูลไม่ใช่สิ่งสำคัญที่น่าสนใจแล้ว ฟังค์ชั่นนี้ของ เครื่องพัฒนาระบบคอมพิวเตอร์ คงจะไม่ได้นำไปใช้ประโยชน์อะไร แต่ในลักษณะของงานบางอย่าง ที่เกี่ยวข้องกันกับการเปลี่ยนข้อมูลในหน่วยความจำโดยตรง เช่นว่า การตรวจสอบความถูกต้องในการทำงานของโปรแกรม ซึ่งเราสามารถตรวจสอบได้จากข้อมูลในหน่วยความจำที่แอดเดรสใดๆ เป็นต้น

สำหรับหลักของ การทำงานในฟังค์ชั่นนี้ ถ้าดูโครงสร้างจากรูปที่ 2.2

แล้วก็จะต้องตอบคำถามที่ว่า 'จะทำอย่างไร จึงจะเอาข้อมูลในแอดเดรสที่สนใจ ซึ่งมีการเปลี่ยนแปลงตลอดเวลานี้ มาแสดงที่หน้าจอภาพได้ ?' คำตอบก็คือ จะต้องให้เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนซิงเกิ้ลบอร์ดนี้ ทำการขอใช้บัส(โดยการส่งสัญญาน บัสรีเควท ออกไป) ไปยังเครื่องที่ต้องการพัฒนาในขณะที่กำลังทำงานอยู่ เมื่อเครื่องที่ต้องการจะพัฒนาตอบรับโดยการส่งสัญญาน บัสแอคโนว์เลท กลับมา เครื่องพัฒนาระบบคอมพิสเตอร์เองก็จะเข้าไปควบคุมแอดเดรสบัสและดาต้าบัสแทน จากนั้นจึงส่งค่าแอดเดรสที่ต้องการรู้ข้อมูลนั้นออกไป เพื่อที่จะได้อ่านข้อมูลเข้ามายังเครื่องพัฒนาระบบคอมพิวเตอร์ หลังจากนั้นก็ทำการยกเลิกการขอใช้บัสที่ขอไปยังเครื่องที่ต้องการจะพัฒนา เพื่อที่จะได้ไม่เป็นการเสียเวลาในการทำงานของเครื่องที่ต้องการจะพัฒนามากนัก แล้วจึงนำเอาข้อมูลดังกล่าว ไปแสดงที่หน้าจอภาพของเครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของเครื่องไอบีเอ็ม ต่อไป



รูปที่ 2.3. แสดงหลักการของการดูข้อมูลขณะโปรแกรมกำลังทำงานอยู่.

2.ความสามารถในเรื่องของการบันทึกข้อมูลภาษาเครื่องลงแผ่นดิสซ์

(SAVE FUNCTION)

ฟังค์ชั่นนี้จะใช้ประโยชน์ในการบันทึกข้อมูลภาษาเครื่องที่ต้องการเก็บเอาไว้ ซึ่งจะบันทึกเก็บเอาไว้ในรูปของแฟ้มข้อมูลแบบ ไบท์ (FILE OF BYTE) จะทำให้มีความสะดวกในการที่จะนำเอามาพัฒนาได้อีกในภายหลัง หรือ จะเป็นการสำรองข้อมูลที่สำคัญเอาไว้ เพื่อป้องกันความเสียหายที่อาจจะเกิดขึ้นได้

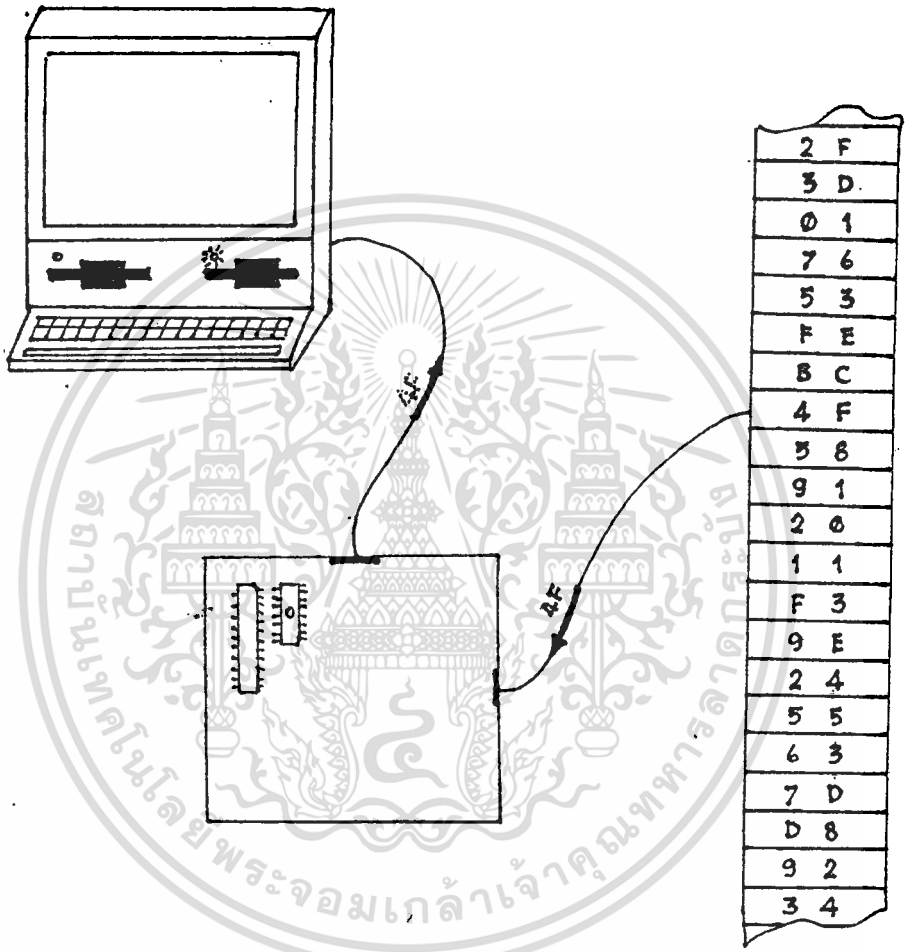
หลักการทำงานของฟังค์ชั่นนี้ จะมีการกำหนดชื่อแฟ้มที่จะนำเอาข้อมูลไปเก็บ พร้อมทั้งระบุแอดเดรสเริ่มต้นของข้อมูลในหน่วยความจำและความยาวของข้อมูล (เป็นไบท์) ด้วย ในส่วนของการระบุข้อกำหนดต่างๆนี้ จะเป็นหน้าที่ของเครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของเครื่องไอบีเอ็ม หลังจากนั้น เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนซิงเกิ้ลบอร์ด ก็จะทำการขอใช้บัสไปยัง เครื่องที่ต้องการจะพัฒนา อีกเช่นกัน เพื่อที่จะได้เข้าถึงหน่วยคงามจำของ เครื่องที่ต้องการจะพัฒนา โดยตรง เมื่อขอใช้บัสของ เครื่องที่ต้องการจะพัฒนา ได้แล้ว ก็จะเริ่มอ่านเอาข้อมูล ณ. ตำแหน่งแอดเดรสที่ต้องการเข้ามาที่ เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนซิงเกิ้ลบอร์ด จนกระทั่งครบตามความยาวของข้อมูลที่กำหนดไว้ในตอนแรก แล้วจึงนำเอาข้อมูลดังกล่าวส่งไปให้ เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของเครื่องไอบีเอ็ม เก็บลงแฟ้มข้อมูลต่อไป

3.ความสามารถในเรื่องของการนำเอาข้อมูลภาษาเครื่องจากแผ่นดิสซ์ลงสู่หน่วยความจำ

(DOWN LOAD FUNCTION)

ฟังค์ชั่นนี้จะใช้ประโยชน์ ในการนำเอาข้อมูลภาษาเครื่องที่เก็บเอาไว้ในรูปของแฟ้มข้อมูลแบบ ไบท์ ลงไปสู่หน่วยความจำของ เครื่องที่ต้องการจะพัฒนา ในตำแหน่งเอดเดรสที่ต้องการ หลักการทำงานของฟังค์ชั่นนี้ จะมีการกำหนดชื่อแฟ้มที่จะนำเอาข้อมูลออกมา พร้อมทั้งระบุแอดเดรสเริ่มต้นในหน่วยความจำของ เครื่องที่ต้องการจะพัฒนา และความยาว ของข้อมูล (เป็นไบท์) ด้วย ในส่วนของการระบุข้อกำหนดต่างๆนี้ จะเป็นหน้าที่ของ

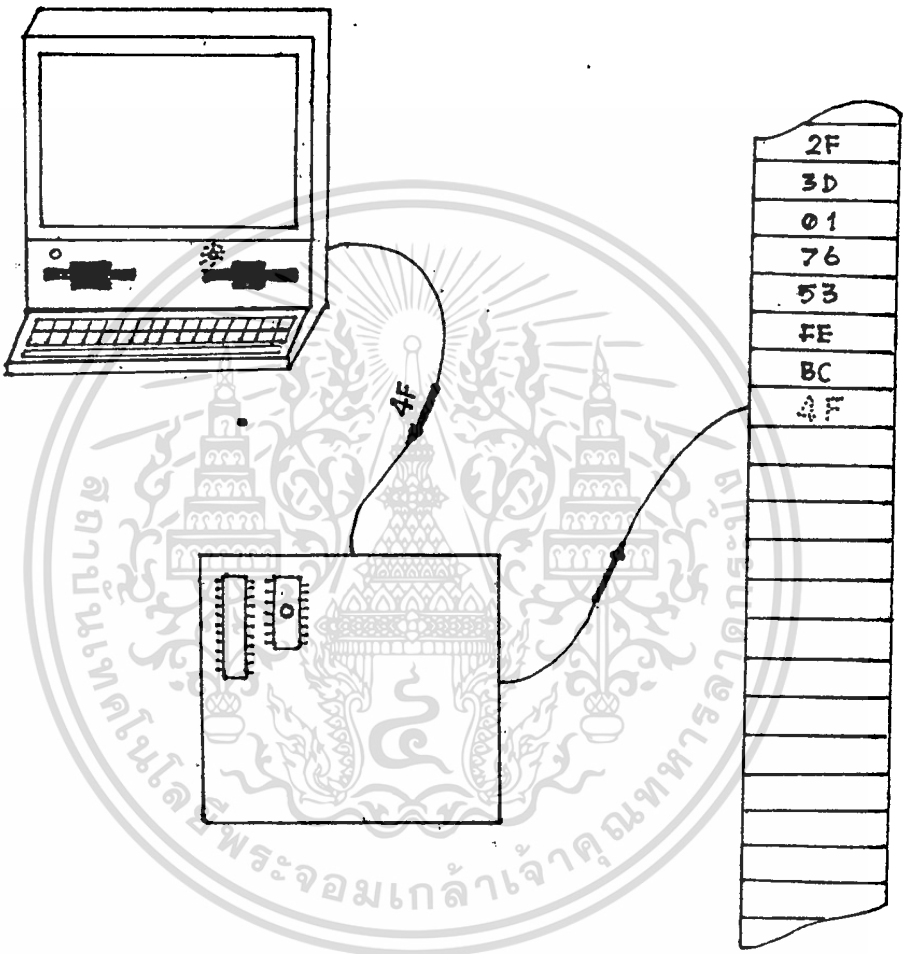
เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของเครื่องไอบีเอ็ม เช่นกัน ซึ่งหลังจากนั้น เครื่องพัฒนาระบบ



รูปที่ 2.4. แสดงหลักการของการบันทึกข้อมูลภาษาเครื่องลงแผ่นดิสซ์.

คอมพิวเตอร์ในส่วนชิงเกิ้ลบอร์ด ก็จะทำการขอใช้บัสไปยัง เครื่องที่ต้องการจะพัฒนา เพื่อที่จะได้เข้าถึงหน่วยความจำของ เครื่องที่ต้องการจะพัฒนา ได้โดยตรง เมื่อขอใช้บัสของเครื่องที่ต้องการจะพัฒนา ได้แล้ว ก็จะเริ่มอ่านเอาข้อมูลจากแผ่นข้อมูลออกมา แล้วก็ส่งไปยังเครื่องที่ต้องการจะพัฒนา ณ. ตำแหน่งแอดเดรสที่ต้องการ เครื่องพัฒนาระบบคอมพิวเตอร์

ในส่วนชิงเก้ลบอร์ด จนกระทั่งครบตามตามความยาวของข้อมูลที่กำหนดไว้ในตอนแรก



รูปที่ 2.5. แสดงหลักการของการนำเอาข้อมูลภาษาเครื่องจากแผ่นดิสซ์ลงสู่หน่วยความจำ.

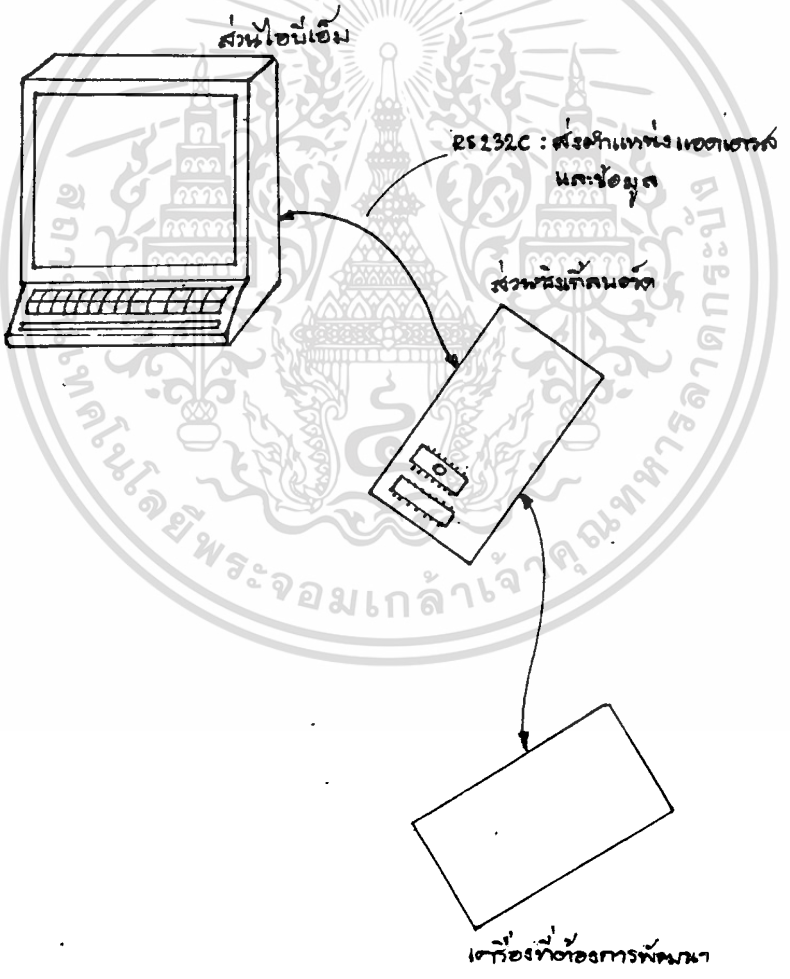
4.ความสามารถ ในการแก้ไขข้อมูล ณ.แอดเดรสใดๆ ในหน่วยความจำ

(FILL DATA FUNCTION)

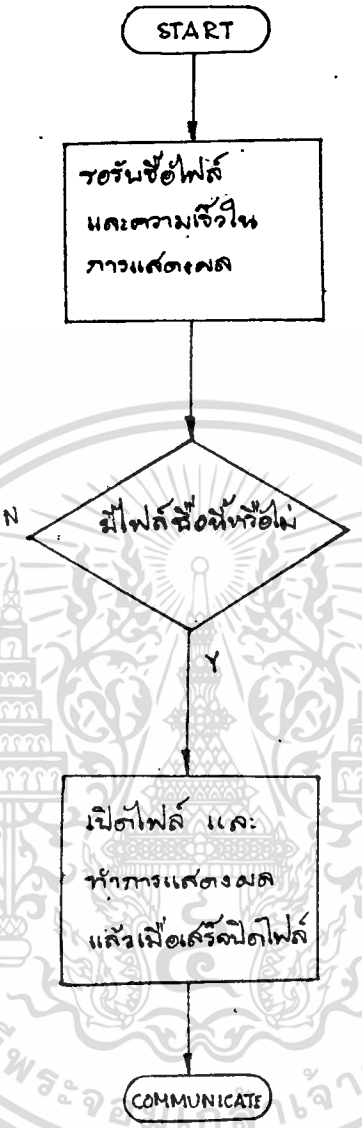
ในการแก้ไขข้อมูล ณ.แอดเดรสใดๆ ในหน่วยความจำของ เครื่องที่ต้อง

การจะพัฒนา นี้มีหลักในการทำงานคือ จะใช้ เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของ

เครื่องไอบีเอ็ม ในการรับค่าของแอดเดรส และ ข้อมูลที่ต้องการจะแก้ไขเปลี่ยนแปลง หลังจากนั้นก็จะเป็นหน้าที่ของ เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนซิงเกิ้ลบอร์ด ที่จะทำการส่งสัญญาณขอใช้บัส เมื่อขอใช้บัสได้แล้ว เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของเครื่องไอบีเอ็ม ก็จะส่งค่าแอดเดรสไปก่อนตามด้วยค่าของข้อมูลไปให้ เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนซิงเกิ้ลบอร์ด ๆ เองก็จะผ่านค่าแอดเดรสและข้อมูลไปให้ เครื่องที่ต้องการจะพัฒนาต่อไป



รูปที่ 2.6. แสดงหลักการของการแก้ไขข้อมูล ณ.แอดเดรสใดๆ ในหน่วยความจำ.



รูปที่ 2.7. แสดงหลักการของการแสดงแฟ้มข้อมูล.

5.ความสามารถในเรื่องของการแสดงแฟ้มข้อมูล

(LIST TEXT FILE FUNCTION)

ในการแสดงข้อมูลต่างๆ ของแฟ้มข้อมูล บนจอภาพของ เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของเครื่องไอบีเอ็ม นี้จะมีประโยชน์ในด้านการตรวจสอบโปรแกรมไปพร้อมๆ กันกับการทำงานของโปรแกรมนั้นๆ เพื่อดูความถูกต้องในการทำงาน ฟังค์ชั่นนี้จะทำ

งานบน เครื่องพัฒนาระบบคอมพิวเตอร์ในส่วนของเครื่องไอบีเอ็ม เท่านั้นเพราะเกี่ยวกับการแสดงผลอย่างเดียว

หลักการก็คือ ค้นหาไฟล์ที่ต้องการ ตามที่ได้ป้อนชื่อเข้าไปนั้นบนแผ่นดิสซ์ ถ้าหากค้นพบก็จะทำการเปิดแฟ้มนั้น แล้วก็อ่านข้อมูลออกมาแสดงผล แต่ถ้าค้นหาแฟ้งดังกล่าวไม่เจอ ก็จะต้องแจ้งให้ทราบ

จากหลักการ การทำงานของฟังค์ชั่นต่างๆทั้งหมดนั้น (ยกเว้นฟังค์ชั่นที่ 5 : การแสดงข้อมูลของแฟ้มข้อมูล) จะมีหลักการที่สำคัญร่วมกันอยู่ประการหนึ่ง ก็คือ การขอใช้ระบบบัสของ เครื่องที่ต้องการจะพัฒนา ให้ได้นั่นเอง ซึ่งเมื่อขอใช้ระบบบัสได้แล้ว ก็จะเป็นการทำงานในส่วนย่อยๆไป ตามแต่ละฟังค์ชั่น

บทที่ 3

การสร้างเครื่องพัฒนาระบบคอมพิวเตอร์

ตามโครงสร้างที่ระบุไว้แต่แรกแล้วนั้น จึงสามารถที่จะแบ่งการสร้างเครื่องพัฒนาระบบคอมพิวเตอร์ ได้เป็นสองส่วนคือ

1. ฮาร์ดแวร์ (HARDWARE)

2. ซอฟแวร์ (SOFTWARE)

ในรายละเอียดนั้นจะขออธิบายถึงหลักการสร้างและการทำงานดังนี้ คือ

1.ฮาร์ดแวร์ ฮาร์ดแวร์ของ เครื่องพัฒนาระบบคอมพิวเตอร์ นี้จะใช้ SINGLE BOARD ที่มี Z-80 เป็นซีพียู (CPU) เป็นตัวจัดการสำคัญโดยมีคอมพิวเตอร์ IBM เป็นส่วน อินพุตเอาท์พุต(INPUT OUTPUT)ดังนั้นจึงจะขออธิบายส่วนเป็น 2 ส่วนย่อยๆ ลงไปคือ

1.1. ส่วนของตัวจัดการ (ซิงเกิ้ลบอร์ด)

ในส่วนนี้ก็คงเหมือนกันกับซิงเกิ้ลบอร์ดทั่วๆ ไป คือ มีหน่วยประมวลผลกลาง (ซีพียู) ซึ่งในที่นี้ใช้ซพยูเบอร์ Z-80 ของบริษัทไซล็อค,หน่วยความจำที่เป็นทั้งแรม (RAM : RANDOM ACCESS MEMORY) สำหรับเก็บข้อมูลต่างๆ ,หน่วยความจำที่เป็นรอม (ROM :READ ONLY MEMORY) สำหรับเก็บมอนิเตอร์โปรแกรม (MONITER PROGRAM) ของเครื่อง , หน่วยติดต่อกับอุปกรณ์ภายนอก หรือที่รู้จักกันดีในชื่อ ไอโอพอร์ท (I/O PORT : INPUT/OUTPUT PORT),และ สิ่งสุดท้ายก็คือ ตัวติดต่อสื่อสารแบบอนุกรม ซึ่งในที่นี้ใช้ยูซาส (USART : UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER) เบอร์ 8251 ซึ่งวงจรใช้งานจะแสดงดัง รูปที่ 3.1

และ ก่อนที่จะกล่าวถึงรายละเอียดของแต่ละส่วนดังนั้น จะขอทำ

จะต่อขาชิพซีเลค เข้ากับเอาท์พุตที่ 0 ของดีโคดเดอร์ ซึ่งสามารถที่จะแสดง เม็มโมรี่แมพได้ดัง รูปที่ 3.4

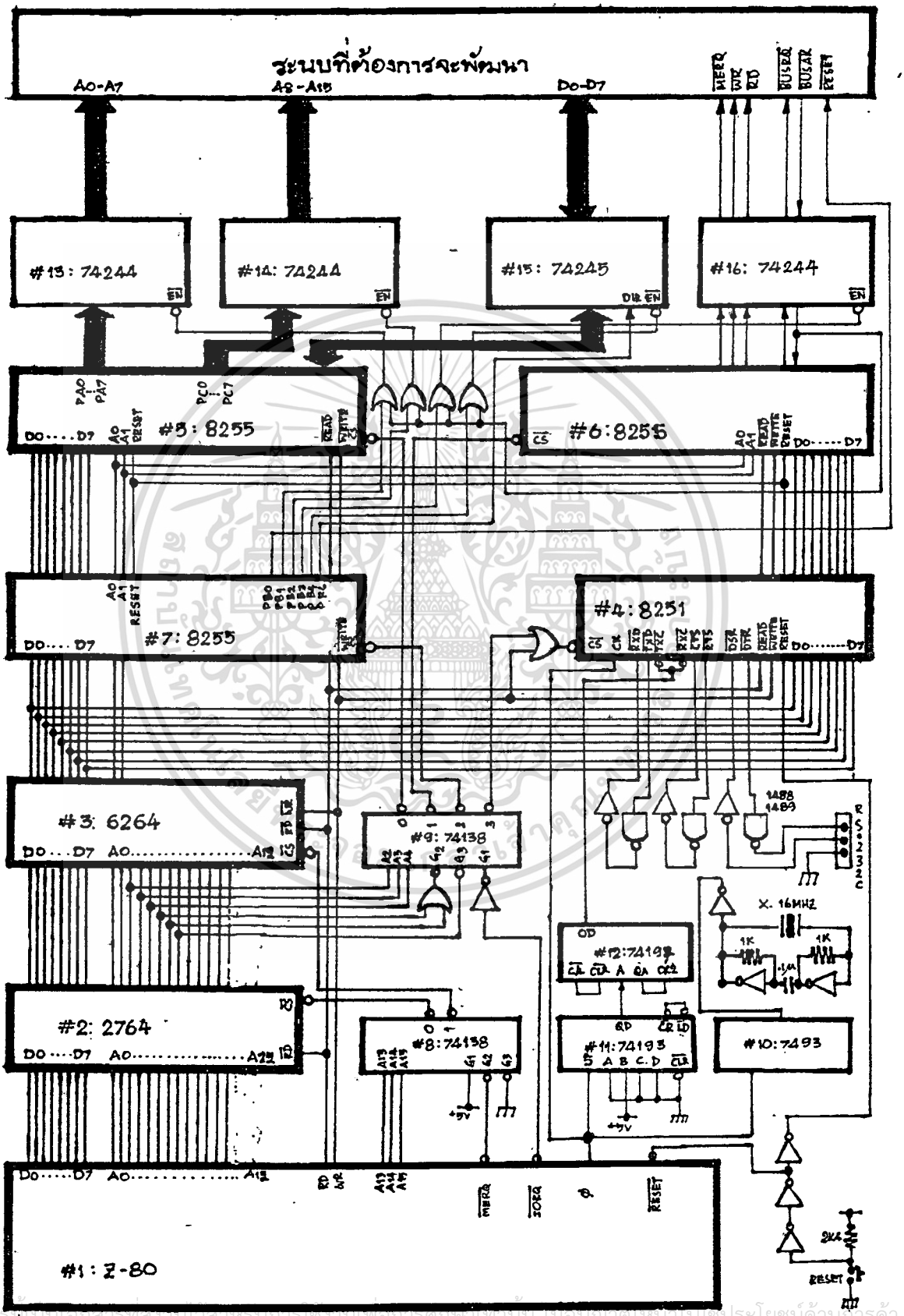
สายที่ใช้ถอดรหัส	●	●	●													
หมายเลขแอดเดรส	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ค่าต่ำสุด	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
ค่าสูงสุด	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(2000H - 3FFFH)

รูปที่ 3.2. เม็มโมรี่แมพของ [illegible] ไอซีหมายเลข 3/.

และสำหรับหน้าที่ของไอซีตัวนี้ก็คือ จ่ายทั้งสัญญาณแอดเดรส(16 BIT ADDRESS) และสัญญาณดาต้า (8 BIT DATA) ให้แก่เครื่องที่ต้องการจะพัฒนาซึ่งจะมีการจัดใช้พอร์ทต่างๆ ดังนี้คือ พอร์ทเอ (PORT A) จ่ายสัญญาณแอดเดรส ด้านไบท์ต่ำ (LOW ORDER BYTE ADDRESS),พอร์ทบี (PORT B) จ่ายสัญญาณดาต้า (DATA),พอร์ทซี (PORT C) จ่ายสัญญาณแอดเดรสไบท์ด้านสูง (HIGH ORDER BYTE ADDRESS)

ไอซีหมายเลขที่ 6 ก็จะใช้สายแอดเดรสที่ 2,3,4,5,6,และ 7 ทำการถอดรหัสร่วมกันกับสายสัญญาณร้องขอใช้อุปกรณ์ภายนอกด้วยไอซีหมายเลขที่ 9 ตัวเดียวกันกับไอซีหมายเลขที่ 5 แต่ไอซีหมายเลขที่ 6 นี้ จะต่อขาชิพซิเลค เข้ากับเอาท์พุตที่ 1 ของดีโคดเดอร์



รูปที่ 3.1. วงจรใช้งานจริงของเครื่องพัฒนาระบบคอมพิวเตอร์.

ความเข้าใจอยู่ประเด็นหนึ่งที่ว่า การจัดวงจรทำงานของ แรม,รอม,ไอโอพอร์ทและยูซาส นั้น จะจัดแบบ เม็มโมรี่เม็พ (MEMORY MAPPED) ทั้งสิ้น ซึ่งต่อไป นี้ก็จะเป็นรายละเอียดในการจัดวงจรทำงานของแต่ละส่วน คือ

- **ส่วนของแรม (RAM SECTION)** ในเครื่องพัฒนาระบบคอมพิวเตอร์นี้จะใช้หน่วยความจำแบบ แรม เพียงตัวเดียวคือ ไอซีหมายเลขที่ 3 ซึ่งเป็น แรม ขนาด 8 บิทคูณ 8 เค (8*8 Kbytes) เบอร์ 6264 ซึ่งการใช้งานในเครื่องพัฒนาระบบคอมพิวเตอร์นี้ ให้แรมตัวนี้เริ่มทำงานตั้งแต่ แอดเดรส (ADDRESS) ที่ 2000H ถึง 3FFFH วิธีที่จะให้เป็นไปตามข้อกำหนดดังกล่าวนี้ ก็จะใช้ไอซีดีโคดเดอร์ (DECODER) เข้ามาช่วยในการคอยถอดรหัสของสายแอดเดรส (ADDRESS LINE) เมื่อค่าแอดเดรสอยู่ในย่าน 2000H ถึง 3FFFH แล้วก็จะต้องจ่ายสัญญาณที่เป็นลอจิก '0' ให้แก่ขาชิพซีเลค (CHIP SELECT) ของแรม ดังนั้นจึงต้องใช้สายแอดเดรสที่ 13,14 และ 15 ต่อเข้ากับขาอินพุทของดีโคดเดอร์ ซึ่งสามารถที่จะแสดง เม็มโมรี่แมพได้ดัง รูปที่ 3.2

สายที่ใช้ถอดรหัส	●	●	●													
หมายเลขแอดเดรส	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ค่าต่ำสุด	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ค่าสูงสุด	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

(0000H - 1FFFH)

รูปที่ 3.3. เม็มโมรี่แมพของ รอม (ไอซีหมายเลขที่ 2)

*...หมายเหตุ ไอซีดีโคดเดอร์ดังกล่าวนั้นคือ ไอซีหมายเลขที่ 8 เป็นเบอร์ 74LS138 ซึ่งเป็นดีโคดเดอร์จาก 3 ให้เป็น 8 (3 TO 8 LINE DECODER)

- ส่วนของรอม (ROM SECTION) และสำหรับ หน่วยความจำแบบ รอมนั้นก็มีเพียงตัวเดียว เช่นกันคือ ไอซีหมายเลขที่ 2 ซึ่งเป็น รอม ขนาด 8 บิทคูณ 8 เค (8*8 Kbytes) เบอร์ 2764 ซึ่งการใช้งานในเครื่องพัฒนาระบบคอมพิวเตอร์นี้ ให้รอมตัวนี้เริ่มทำงานตั้งแต่ แอดเดรส ที่ 0000H ถึง 1FFFH วิธีที่จะให้เป็นไปตามข้อกำหนดดังกล่าวนี้ ก็จะใช้ไอซีดีโคดเดอร์ตัวเดียวกันคือไอซีหมายเลข 8 เข้ามาช่วยในการคอยถอดรหัสของสายแอดเดรส (ADDRESS LINE) เมื่อค่าแอดเดรสอยู่ในย่าน 0000H ถึง 1FFFH แล้ว ก็จะต้องจ่ายสัญญาณที่เป็นลอจิก '0' ให้แก่ขาชิพซีเลค (CHIP SELECT) ของรอม ดังนั้นจึงต้องใช้สายแอดเดรสที่ 13,14 และ 15 เช่นกันต่อเข้ากับขาอินพุทของดีโคดเดอร์ ซึ่งสามารถที่จะแสดง เม็มโมรี่แมพได้ดัง รูปที่ 3.3

- ส่วนของไอโอพอร์ท (I/O SECTION)

ในส่วนนี้จะใช้ไอโอพอร์ทแบบ พีพีไอ (PPI : PROGRAMABLE PERIPHERAL INTERFACE) เบอร์ 8255 ทั้งสิ้น 3 ตัวคือ ไอซีหมายเลขที่ 5,6 และ 7 เหตุผลหนึ่งที่มีความจำเป็นต้องใช้ไอโอพอร์ทจำนวนมากเช่นนี้ก็เพื่อ ต้องการใช้ไอโอพอร์ทดังกล่าวนี้ เชื่อมต่อกันกับขาทุกขาของ ซีพียู (Z-80) ในเครื่องที่ต้องการจะพัฒนานั่นเอง เพราะจะมีบางช่วงที่ เครื่องพัฒนาระบบคอมพิวเตอร์นี้ต้องเข้าไปควบคุมระบบบัส (BUS SYSTEM) ของเครื่องที่ต้องการจะพัฒนาด้วย

ดังนั้นจึงจะขอกล่าวรายละเอียดของ การจัดวงจรและหน้าที่การทำงานของแต่ละตัวดังนี้

ไอซีหมายเลขที่ 5 จะใช้สายแอดเดรสที่ 2,3,4,5,6,และ 7 ทำการถอดรหัสร่วมกันกับสายสัญญานร้องขอใช้อุปกรณ์ภายนอก (IORQ ; I/O REQUEST) ด้วยไอซีหมายเลขที่ 9 (เบอร์ 74LS138 ซึ่งเป็นดีโคดเดอร์จาก 3 ให้เป็น 8) ซึ่งไอซีหมายเลขที่ 5 นี้

ซึ่งสามารถที่จะแสดง เม็มโมรี่แมพได้ดัง รูปที่ 3.5

สายที่ใช้ถอดรหัส									●	●	●	●	●	●		
หมายเลขแอดเดรส	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ค่าต่ำสุด	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ค่าสูงสุด	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1

(0000H-FF03H)

รูปที่ 3.4. เม็มโมรี่แมพของ พีพีไอ (ไอซีหมายเลขที่ 5).

(** .....ดูที่หมายเหตุ)

และสำหรับหน้าที่ของไอซีตัวนี้ก็คือ รับและส่งสัญญานควบคุมการ ทำงานต่าง ซึ่งจะมีการใช้งานของแต่ละพอร์ท คือ ใช้พอร์ทเอ บิทที่ 2 เป็นขาจ่ายสัญญานร้องขอใช้หน่วยความจำ (MREQ : MEMORY REQUEST),พอร์ทเอ. บิทที่ 3 เป็นขาจ่ายสัญญานการเขียนหน่วยความจำ (WR : WRITE),พอร์ทเอ บิทที่ 4 เป็น ขาจ่ายสัญญานอ่านหน่วยความจำ(RD : READ),พอร์ทซี บิทที่ 7 เป็นขาจ่ายสัญญานร้องขอใช้บัส (BUSRQ : BUS REQUEST) และ พอร์ทซี บิทที่ 0 เป็นขารับสัญญานตอบรับการร้องขอใช้บัส (BUSAK : BUS ACKNOWLEDGE)

สายที่ใช้ถอดรหัส									●	●	●	●	●	●		
หมายเลขแอดเดรส	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ค่าต่ำสุด	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
ค่าสูงสุด	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1

(0004H - FF07H)

รูปที่ 3.5. เม็มโมรี่แมพของ พีพีไอ (ไอซีหมายเลขที่ 6).

(** .....ดูที่หมายเหตุ)

ไอซีหมายเลขที่ 7 ก็ยังคงใช้สายแอดเดรสที่ 2,3,4,5,6,และ 7 ทำการถอดรหัสร่วมกันกับสายสัญญาณร้องขอใช้อุปกรณ์ภายนอก ด้วยไอซีหมายเลขที่ 9 ตัวเดียวกันกับไอซีหมายเลขที่ 5 ใช้แต่ไอซีหมายเลขที่ 7 นี้ จะต่อขาซีพซีเลค เข้ากับเอาท์พุตที่ 2 ของดิโคดเดอร์ ซึ่งสามารถที่จะแสดง เม็มโมรี่แมพได้ดัง รูปที่ 3.6

สำหรับหน้าที่ของพีพีไอตัวนี้ก็คือ ทำหน้าที่ในการทำให้ ไทร์สเตทบัฟเฟอร์ (TRI-SATATE BUFFER) ปิดกั้นสัญญาณต่างๆหรือยอมให้สัญญาณต่างๆผ่าน ซึ่งในที่นี้จะใช้เป็นตัวป้องกันการลัดวงจรแบบชั่วขณะ ระหว่างเครื่องที่ต้องการพัฒนากับเครื่องพัฒนาระบบคอมพิวเตอร์โดยที่จะมีการใช้พอร์ทในการควบคุมต่างๆ ดังนี้คือ ใช้พอร์ทบี บิทที่ 1 ควบคุมไอซีหมายเลขที่ 13 (ซึ่งไอซีหมายเลขที่ 13 นี้จะไปควบคุม ว่าจะให้สัญญาณจาก พอร์ทเอของ พีพี

ไอ (IC # 5) ผ่านออกไปได้หรือไม่ อีกขั้นหนึ่ง),พอร์ทบี บิทที่ 2 ควบคุมไอซีหมายเลขที่ 14 (ซึ่งไอซีหมายเลขที่ 14 นี้จะไปควบคุม ว่าจะให้สัญญาณจาก พอร์ทซีของ พีพีไอ (IC # 5) ผ่านออกไปได้หรือไม่ อีกขั้นหนึ่ง) ,พอร์ทบี บิทที่ 3 ควบคุมไอซีหมายเลขที่ 16 (ซึ่งไอซีหมายเลขที่ 16 นี้จะไปควบคุม ว่าจะให้สัญญาณจาก พอร์ทเอของ พีพีไอ (IC # 6) ผ่านออกไปได้หรือไม่ อีกขั้นหนึ่ง) ,พอร์ทบี บิทที่ 4 ควบคุมไอซีหมายเลขที่ 15 (ซึ่งไอซีหมายเลขที่ 15 นี้จะไปควบคุม ว่าจะให้สัญญาณจาก พอร์ทบีของ พีพีไอ

สายที่ใช้ถอดรหัส									●	●	●	●	●	●		
หมายเลขแอดเดรส	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ค่าต่ำสุด	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
ค่าสูงสุด	1	1	1	1	1	1	1	1	0	0	0	0	1	0	1	1

(0008H - FF0BH)

<u>รูปที่ 3.6. เม็มโมรี่แมพของ พีพีไอ (ไอซีหมายเลขที่ 7).</u>

(** .....ดูที่หมายเหตุ)

(IC # 5) ผ่านออกไปได้หรือไม่ อีกขั้นหนึ่ง) ,พอร์ทบี บิทที่ 6 ควบคุมขากำหนดทิศทางของไอซีหมายเลขที่ 15 (ซึ่งไอซีหมายเลขที่ 15 นี้สามารถรับส่งข้อมูลได้สองทิศทาง เพื่อให้สา

มารถใช้ได้กับการรับส่งดาต้านั่นเอง ซึ่งถ้าพอร์ทบี บิทที่ 6 มีลอจิกเป็น '1' ก็จะทำให้ทิศทางของข้อมูลสามารถที่จะ ส่งเข้ามายังเครื่องพัฒนาระบบเท่านั้น นั่นคือ อยู่ในสถานะการอ่านนั่นเองแต่ถ้าพอร์ทบี บิทที่ 6 มีลอจิกเป็น '0' ก็จะทำให้ทิศทางของข้อมูลสามารถที่จะส่งออกไปจากเครื่องพัฒนาระบบเท่านั้น นั่นคือ อยู่ในสถานะการเขียนนั่นเอง)

**...หมายเหตุ สำหรับความหมายของสายแอดเดรสที่ 0 และ 1 ร่วมกับสัญญานอ่าน (RD) และสัญญานเขียน (WR) สามารถที่จะสรุปได้ดังรูปที่ 3.7

RD	WR	A0	A1	REGISTER NAME
1	0	0	0	WRITE PORT A DATA
0	1	0	0	READ PORT A DATA
1	0	0	1	WRITE PORT B DATA
0	1	0	1	READ PORT B DATA
1	0	1	0	WRITE PORT C DATA
0	1	1	0	READ PORT C DATA
1	0	1	1	WRITE CONTROL WORD
0	1	1	1	ILLIGAL READ REGISTER

<u>รูปที่ 3.7.ความหมายของสายแอดเดรสที่ 0 และ 1 ร่วมกับสัญญานอ่าน (RD) และสัญญานเขียน (WR)</u>

- **ส่วนของยูซาส (USART SECTION)**

ยูซาส (ไอซีหมายเลขที่ 4) จะมีหน้าที่ในการรับและส่งข้อมูลระหว่างเครื่องพัฒนาระบบคอมพิวเตอร์ส่วนที่เป็นซิงเกิ้ลบอร์ดกับส่วนของเครื่อง IBM ซึ่งจะใช้การส่งแบบอนุกรม ส่วนการจัดวงจรใช้งานนั้นจะใช้สายแอดเดรสที่ 2,3,4,5,6,และ 7 ทำการถอดรหัสร่วมกันกับสายสัญญานร้องขอใช้อุปกรณ์ภายนอก ด้วยไอซีหมายเลขที่ 9 ตัวเดียว

กันกับไอซีหมายเลขที่ 5 แต่ไอซีหมายเลขที่ 4 นี้ จะต่อขาชิพซีเลค เข้ากับเอาท์พุตที่ 3 ของดีโคดเดอร์ ซึ่งสามารถที่จะแสดง เม็มโมรี่แมพได้ดัง รูปที่ 3.8

สายที่ใช้ดีโคดเดอร์									O	O	O	O	O	O	O	
หมายเลขแอดเดรส	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ค่าต่ำสุด	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
ค่าสูงสุด	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0

(000CH-0

รูปที่ 3.8. เม็มโมรี่แมพของยูซาส (ไอซีหมายเลขที่ 4).

สำหรับทางด้านของขาสัญญาณอินพุตและเอาท์พุตจัดไว้ดังนี้ คือ จะจัดให้ขา Rxd ลัดวงจรเข้าหากันกับขา Txd เพราะในการรับส่งข้อมูลระหว่างชิงเกิ้ลบอร์ดและเครื่อง IBM นั้น จะใช้ขารับส่งส่วนที่ใช้กับโมเด็ม (MODEM : MODULATE AND DEMODULATE) แทน, ให้ขาเคลียร์ทูเซนต์ (CTS : CLEAR TO SEND) ลัดวงจรเข้าหากันกับขารีเควททูเซนต์ (RTS : REQUEST TO SEND) เพื่อให้เครื่องชิงเกิ้ลบอร์ดส่งข้อมูลออกไป ถ้าขา TxE มีลอจิกเป็น '1' , ใช้ขาดาต้าเทอมินัลรีดดี้ (DTR : DATA TERMINAL READY) ส่ง

ข้อมูลออกไป และใช้ขาดาต้าเซตรีดดี้ (DSR : DATA SET READY) เป็นขารับข้อมูลเข้ามายังเครื่องซิงเกิ้ลบอร์ด

1.2. ส่วนอินพุตเอาท์พุต (เครื่อง IBM)

ส่วนนี้เป็นแต่เพียงส่วนอินพุต (ในที่นี้คือ KEY BOARD) และเป็นส่วนเอาท์พุต (ในที่นี้คือ DISPLAY MONITOE) ให้กับซิงเกิ้ลบอร์ทเท่านั้น ซึ่งการทำงานทั้งหมดในการแสดงผลหน้าจอ จะถูกเขียนขึ้นด้วยโปรแกรมภาษาปาสคาลทั้ง สิ้นและต่อไปนี้จะขอกล่าวถึงส่วนที่เป็นซอฟแวร์บ้างซึ่งมีรายละเอียดดังนี้

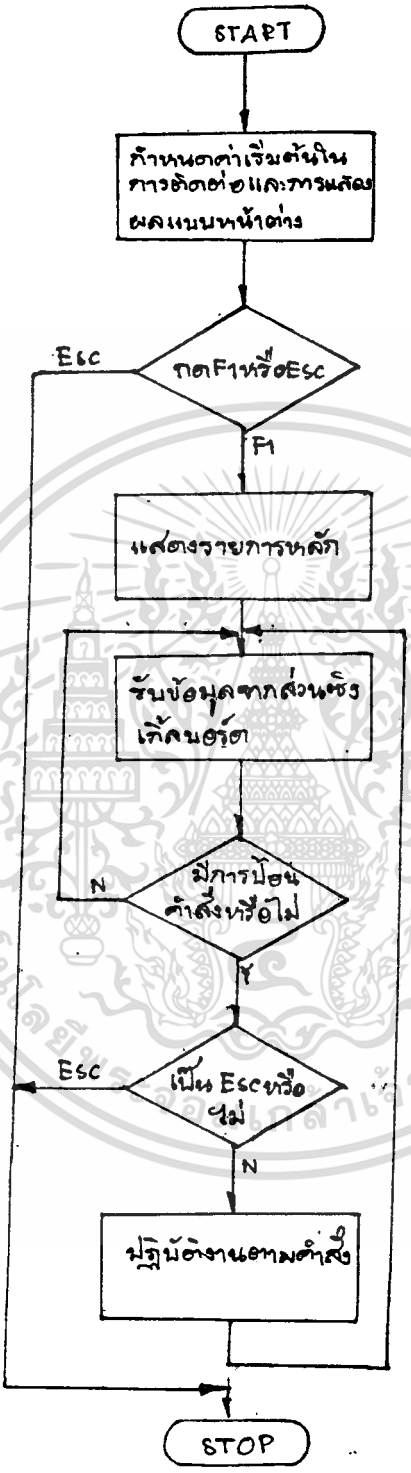
2. ส่วนซอฟแวร์

ในส่วนของซอฟแวร์นี้จะประกอบไปด้วยโปรแกรม 2 ส่วน คือ โปรแกรมภาษาแอสแซมบลี้ที่ใช้ควบคุมการทำงานเครื่องพัฒนาระบบคอมพิวเตอร์ส่วนที่เป็นซิงเกิ้ลบอร์ด (ซึ่งต่อไปนี้จะเรียกสั้นๆ ว่า 'ซิงเกิ้ลบอร์ด') และโปรแกรมภาษาปาสคาล เพื่อใช้ในการควบคุมการทำงานเครื่องพัฒนาระบบคอมพิวเตอร์ ส่วนของเครื่อง IBM (ซึ่งต่อไปนี้จะเรียกสั้นๆ ว่า 'เครื่อง IBM') และต่อไปนี้เป็นโปรแกรมส่วนของภาษาปาสคาล

- โปรแกรมของส่วนภาษาปาสคาล

ในโปรแกรมส่วนนี้ จะใช้โปรแกรมสำเร็จรูป คือ เทอร์โบ เพาว์เวอร์ทูลซ์ เข้ามาช่วยในการแสดงหน้าต่าง (WINDOW) ทำให้สามารถที่จะแสดงผลของฟังค์ชั่น (FUNCTION) ต่างๆอย่างอิสระแยกออกจากกันได้และการทำงานของโปรแกรมภาษาปาสคาลนี้สามารถที่จะเขียนเป็นผังการทำงานได้ดังรูปที่ 3.9

ในการทำงานของโปรแกรมภาษาปาสคาลนี้ จะประกอบขึ้นด้วยโปรแกรมย่อยหลายๆ โปรแกรม โดยการทำงานในขั้นต้นนั้น จะเริ่มทำงานที่โปรแกรมหลัก (MAIN PROGRAM) ก่อน เพื่อทำการหน้าที่ติดต่อกับซิงเกิ้ลบอร์ด ซึ่งจะทำการส่งข้อมูลคือค่ารหัสแอสกี้ (ASCII CODE) ของคีย์ (KEY) ที่กดเข้าไป จากนั้นก็ จะเริ่มไปทำงานต่อที่ โปรแกรย่อย

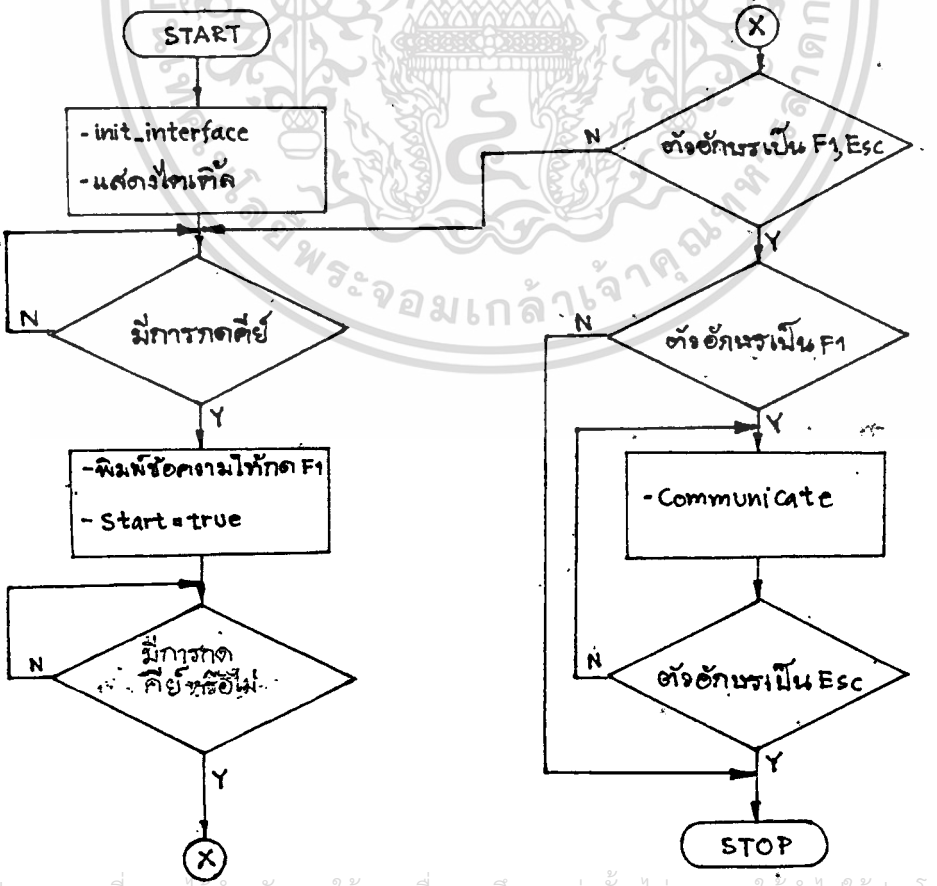


รูปที่ 3.9. โฟว์ชาร์ทการทำงานของโปรแกรมภาษาปาสคาล.

ต่างๆส่วนข้อมูลที่ส่งไปให้ชิงเกิ้ลบอร์ด นั้นตัวชิงเกิ้ลบอร์ดก็จะไปปฏิบัติตามคำสั่งที่กดเข้าไปในตอนแรกนั้น และต่อไปนี้จะเป็นรายละเอียดของการทำงานคือ จากโฟว์ชาร์ตการทำงานในรูปที่ 3.10 จะเห็นได้ว่ารูปที่ 3.10 นี้เป็นเพียงหลักการกว้างๆเท่านั้น โดยมีการกำหนดลักษณะการทำงานของการติดต่อแบบอนุกรมในขั้นแรก จากนั้นก็จะแสดงข้อความให้ผู้ใช้กดคีย์ใดๆ ก่อน เมื่อมีการกดคีย์จากผู้ใช้แล้วจึงจะแสดงข้อความต่อไปว่า

```
'Press F1 to Main Menu....'
```

เพื่อที่จะแสดงเมนู (Menu) ให้ผู้ใช้ทราบ จากนั้นการทำงานของโปรแกรมก็จะไปทำการกำหนดค่าให้แก่ตัวแปร start ให้เป็นจริง (true) ที่มีการกำหนดตัวแปร start ขึ้นมานี้เพื่อที่จะใช้ในการแสดงว่าเป็นการเริ่มต้นการทำงานของโปรแกรมครั้งแรกที่สุด สำหรับรายละเอียดจะอธิบายต่อไปอีกครั้งหนึ่งในโปรแกรมย่อยส่วน 'Communicate'



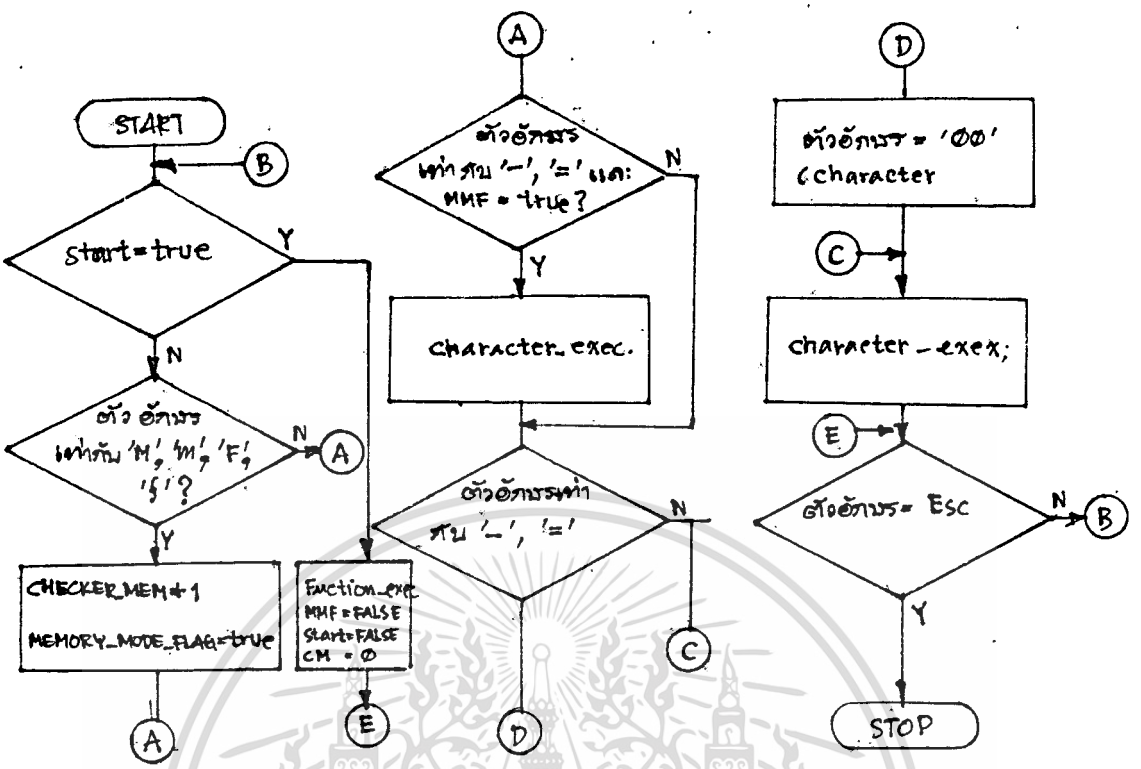
รูปที่ 3.10. โฟว์ชาร์ตการทำงานของ Main Program.

เมื่อพิมพ์ข้อความ 'Press F1 to Main Menu' เรียบร้อยแล้วก็จะมาทำการรอรับคีย์จนกว่าคีย์ที่กดเข้ามานั้นเป็นคีย์ F1 หรือ Esc เท่านั้น ซึ่งถ้าไม่ใช่คีย์ดังกล่าวนี้ก็จะยังคอยรอรับคีย์ต่อไปอีก นั่นหมายความว่า จะแสดงเมนเมนูก่อนเท่านั้น มิฉนั้นก็กดคีย์ Esc ออกจากการทำงานของโปรแกรมไปเลย เมื่อผู้ใช้กดคีย์ F1 แล้วการทำงานของโปรแกรมก็จะเรียกใช้โปรแกรมย่อยส่วน 'Commnicate' ในลักษณะการทำงานแบบวน (Loop) จนกระทั่งตัวแปร Character เท่ากับค่ารหัสของคีย์ Esc เท่านั้นจึงจะเป็นการสิ้นสุดของการทำงานในส่วนณปรแกรมภาษาปาสคาลทั้งหมด

และต่อไปนี้ก็จะเป็นรายละเอียดของโปรแกรมย่อยส่วน 'Communicate' ซึ่งสามารถแสดงการทำงานด้วยโฟว์ชาร์ตดัง รูปที่ 3.11 โปรแกรมย่อยส่วน 'Communicate' จะทำหน้าที่ในการรับส่งข้อมูลหรือคำสั่งต่างๆระหว่างเครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม กับ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนชิงเด็ลบอร์ดนี้ซึ่งจะทำงานอยู่ที่โปรแกรมย่อยส่วนนี้จนกระทั่ง ค่าของคีย์ที่กดเข้ามานั้นมีค่าเท่ากับ รหัสเอสเคป (Esc:ESCAPE) จึงจะเป็นการจบสิ้นการทำงานทั้งหมดของโปรแกรมส่วนภาษาปาสคาล จะเห็นได้ว่าแรกที่สุดก็จะทำการตรวจสอบกันก่อนเลยว่าตัวแปร start เป็นจริงหรือเป็นเท็จ

ถ้า ตัวแปร start เป็นจริง : นั่นแสดงว่าเป็นการเริ่มต้นการทำงานของโปรแกรมภาษาปาสคาลนั่นเอง เพราะลักษณะการทำงานของโปรแกรมนี้ ตัวแปร start จะเป็นจริงได้เฉพาะแต่ครั้งแรกของการทำงานของโปรแกรมเท่านั้นนั่นเอง เมื่อเป็นเช่นนี้ก็จะมีการเรียกใช้โปรแกรมย่อยส่วน 'Function_exec' ต่อเพื่อไปทำการแยกแยะการทำงานต่างๆ ต่อไปอีก

ถ้า ตัวแปร start เป็นเท็จ : เมื่อเป็นเช่นนี้ ต่อไปก็จะทำการเรียกใช้โปรแกรมย่อยส่วน 'Receive_massage' เพื่อรอรับข้อมูลต่างๆจาก เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของชิงเก็ลบอร์ด จนกระทั่งมีการกดคีย์เกิดขึ้น (ซึ่งการกดคีย์ในครั้งนี้ก็จะนำเอาค่าที่ได้จากการกดนี้ให้แก่ ตัวแปร Character ไปพร้อมเลย) ต่อจากนั้นก็ทำการตรวจสอบดูว่าค่าของ ตัวแปร Character เท่ากับ 27 หรือไม่



รูปที่ 3.11. โฟว์ชาร์ตของโปรแกรมย่อยส่วน 'Communicate'.

ถ้าใช่ : ก็แสดงว่าคีย์ที่กดเข้ามานั้นเป็น ฟังค์ชั่นคีย์จึงจะทำการกำหนดให้ตัวแปร Check_Mode_flag มีค่าเป็นเท็จและ ตัวแปร Checker_Memory มีค่าเท่ากับ 0 และหลังจากนั้นก็ทำการเรียกใช้โปรแกรมย่อยส่วน 'Function_exec' ต่อไป (ดูหน้าที่ของตัวแปรเหล่านี้ในตอนท้าย)

ถ้าไม่ใช่ : ก็จะทำการตรวจอีกครั้งว่าค่าของ ตัวแปร Character เท่ากับรหัสของอักษร 'M','F','m','f' หรือไม่

ถ้าใช่ : ก็จะทำการกำหนดค่าให้ ตัวแปร Checker_Memory เท่ากับ 1 และ ตัวแปร Memory_Mode_flag เป็นจริง

ถ้าไม่ใช่ : จะมาทำการตรวจสอบว่าเป็นเครื่องหมายเท่ากับ ('=') หรือ เครื่องหมายลบ ('-') และตัวแปร Memory_Mode_flag เป็นจริงหรือไม่ ถ้าเป็นจริงก็จะไปทำงานที่โปรแกรมส่วนย่อย 'Character_exec' ต่อไป

หลังจากนั้นก็จะมาทำการตรวจดูว่า ตัวแปร Character มีค่าเท่ากับรหัส

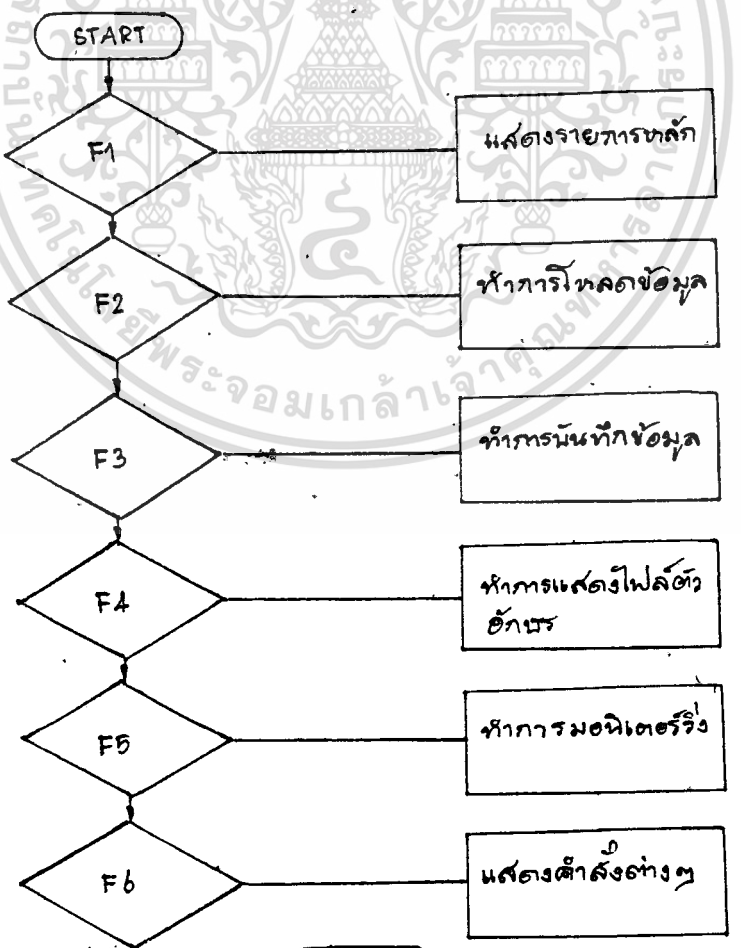
ของ Esc หรือไม่ ถ้าเท่า ก็จะจบการทำงานของโปรแกรมย่อยส่วน 'Communicate'แต่ถ้าไม่เท่าก็จะเริ่มทำงานใหม่อีกครั้งหนึ่ง

หน้าที่ของตัวแปรต่างๆมีดังนี้

ตัวแปร Memory_Mode_flag เพื่อป้องกันไม่ให้ การทำงานของเครื่องหมายลบหรือเครื่องหมายเท่ากับ ไปทำงานที่หน้าต่างอื่นๆ กล่าวคือ เครื่องหมายทั้งสองนี้จะมีผล ใช้งานได้ขณะที่เครื่องกำลังทำงานอยู่ในโหมดของการแก้ไขข้อมูลเท่านั้น

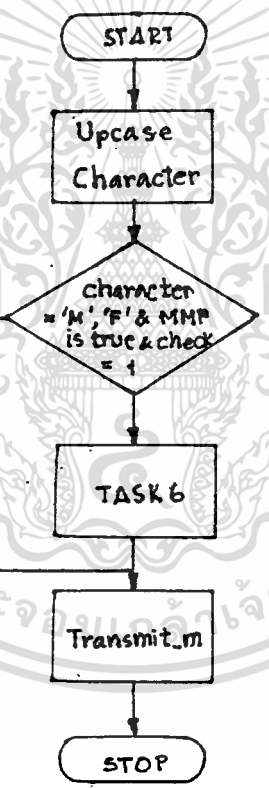
ตัวแปร Checker_memory เป็นตัวแปรที่มีหน้าที่ในการป้องกันการลบหน้าต่างของการทำงานเกี่ยวกับการแก้ไขข้อมูล

และรูปที่ 3.12เป็นโฟว์ชาร์ตของโปรแกรมย่อยส่วน 'Function_exec'



รูปที่ 3.12. โฟว์ชาร์ตของโปรแกรมย่อยส่วน 'Function_exec'.

และโปรแกรมย่อยส่วน 'Character_exec' แสดงโฟว์ชาร์ตได้ดังรูปที่ 3.13 สามารถอธิบายได้คือ ช่วงแรกเมื่อเข้ามาก็จะทำการแปลงให้ตัวอักษรเป็นตัวใหญ่ก่อนเสมอจากนั้นก็ทำการตรวจดูว่าอักษรนั้นเป็น 'M' หรือ 'F'และตัวแปร Memory_Mode_flag เป็นจริง และ ตัวแปร Checker_Memory เท่ากับ 1 หรือไม่ ถ้าเป็นจริงก็จะไปทำงานที่หน้าต่างที่ 6 แล้วไปทำงานที่โปรแกรมย่อยส่วน 'Transmit_massage' แต่ถ้าไม่เป็นจริงก็จะไปทำงานที่โปรแกรมย่อยส่วน 'Transmit_massage' ทันที

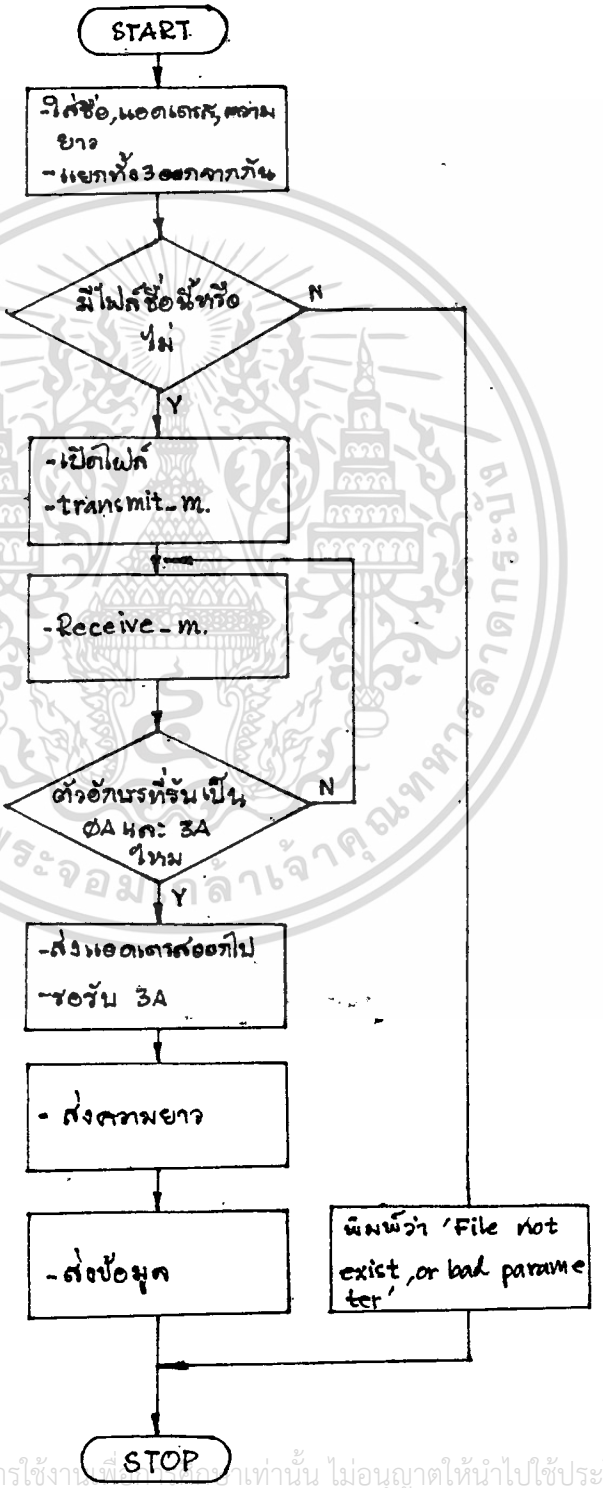


รูปที่ 3.13. โฟว์ชาร์ตของโปรแกรมย่อยส่วน 'Character_exec'.

ส่วนโปรแกรมย่อย 'Function_exec' มีหน้าที่ในการตรวจดูว่า ฟังค์ชั่นคีย์ที่กดเข้ามานั้นเป็นฟังค์ชั่นใด ถ้าเป็น

ฟังค์ชั่น F1. จะแสดงรายการหลัก (MAIN MENU) ที่จอแสดงผล ณ. หน้าต่างเดียวกันกับการแสดงผลข้อมูลของหน่วยความจำ

ฟังค์ชั่น F2. จะเป็นการนำเอาข้อมูลที่เป็นภาษาเครื่อง (MACHINE CODE) ที่เก็บไว้ในแฟ้มข้อมูล (FILE) ลงไปสู่หน่วยความจำของเครื่องที่ต้องการจะพัฒนา ณ. ตำแหน่งแอดเดรสที่ต้องการ ซึ่งสามารถที่จะอธิบายดารทำงานดังโฟว์ชาร์ตในรูปที่ 3.12



รูปที่ 3.14. โฟว์ชาร์ตการทำงานของ การโหลดไฟล์.

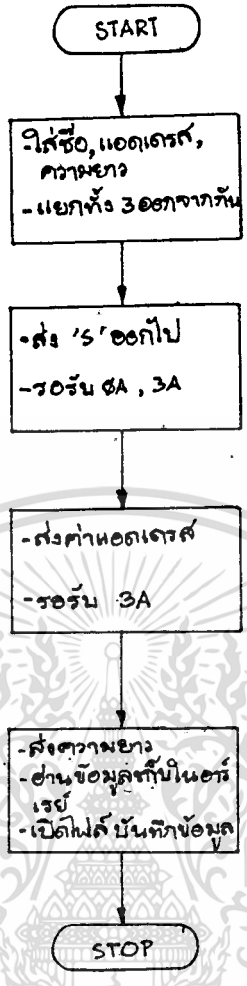
จากโฟว์ชาร์ตจะเห็นว่า จะต้องใส่ชื่อของไฟล์ที่ต้องการโหลด, แอดเดรสเริ่มต้นของหน่วยความจำในเครื่องที่ต้องการจะพัฒนา และ ความยาวของไฟล์โดยประมาณ ซึ่งทั้งหมดนี้จะใส่เข้ามาพร้อมกัน โดยแยกออกจากกันด้วยเครื่องหมายคอมม่า (Comma) เมื่อป้อนเสร็จ โปรแกรมก็เริ่มทำการแยกชื่อ, จุดเริ่มต้น และ ความยาวของไฟล์ออกจากกัน โดยเริ่มทำการแยกเอาแต่เฉพาะชื่อของไฟล์ออกมาก่อน เมื่อแยกชื่อของไฟล์ออกมาได้แล้ว ก็เริ่มทำการตรวจสอบดูว่ามีไฟล์ชื่อนี้อยู่ในแผ่นดิสซ์หรือไม่ ถ้าไม่มีไฟล์ชื่อดังกล่าวนี้ก็จะแจ้งให้ทราบโดยพิมพ์ว่า

'File not exist,or bad parameter'

แต่ถ้ามีไฟล์ตามชื่อที่ป้อนเข้าไปนี้ ก็จะทำการเปิดไฟล์ แล้วจากนั้นก็ส่งตัวอักษร 'L' ไปให้ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด เพื่อให้ทราบว่า ต่อไปนี้เป็นขบวนการทำงานโหลดไฟล์ เมื่อส่งไปแล้วก็จะรอรับคำตอบจาก เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด ด้วยการรอรับรหัสไลน์ฟีดและโคลอน (Colon :) เพื่อที่เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม จะได้ทราบว่า เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด ได้รับรู้ว่าต่อไปนี้เป็นการโหลดไฟล์แล้วนั่นเอง เมื่อ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ได้รับรหัสไลน์ฟีดและโคลอนแล้ว ก็จะส่งค่าของแอดเดรสเริ่มต้นในหน่วยความจำของ เครื่องที่ต้องการจะพัฒนา ไปให้แก่ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด ตามมาด้วยความยาวของไฟล์ แล้วจึงค่อยส่งข้อมูลจากไฟล์นั้นตามไป ก็เป็นการจบการทำงานในฟังค์ชั่นนี้

ฟังค์ชั่น F3. จะเป็นการนำเอาข้อมูลที่เป็นภาษาเครื่อง (MACHINE CODE) หน่วยความจำของเครื่องที่ต้องการจะพัฒนา ณ. ตำแหน่งแอดเดรสที่ต้องการ มาเก็บไว้ในแฟ้มข้อมูล (FILE) ซึ่งสามารถอธิบายการทำงานของฟังค์ชั่นนี้ได้ดังรูปที่ 3.15

การทำงานก็เริ่มจากการให้ใส่ชื่อของไฟล์ที่ต้องการจะนำเอาข้อมูลไปเก็บไว้, ค่าของแอดเดรสเริ่มต้นในหน่วยความจำของ เครื่องที่ต้องการจะพัฒนา ที่จะเอาข้อมูลมา

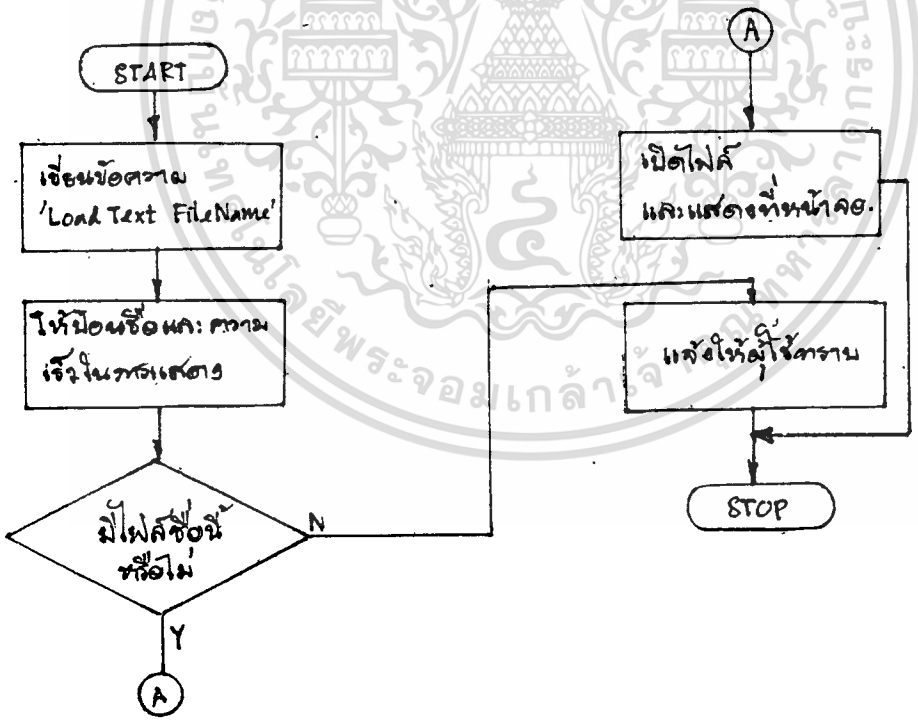


<u>รูปที่ 3.15. โฟว์ชาร์ตการทำงานของ การบันทึกไฟล์.</u>

เก็บบันทึกไว้ และ ความยาวโดยประมาณของข้อมูลที่ต้องการจะบันทึก ซึ่งทั้งหมดนี้ก็ป้อนเข้าไปทีเดียวเช่นกัน โดยแยกออกจากกันด้วยเครื่องหมายคอมม่าเช่นกัน เมื่อป้อนสิ่งที่กำหนดมาดังกล่าวหมดแล้ว ก็จะทำการแยกเอาชื่อ.แอดเดรสเริ่มต้นและความยาวออกจากกัน จากนั้น เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ก็จะส่งตัวอักษร 'S' ออกไปให้แก่ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด เพื่อให้ทราบว่าต่อไปนี้เป็นการทำงานในส่วนของการบันทึกไฟล์ เมื่อส่งออกไปแล้ว เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ก็จะรอรับเอารหัสไลน์ฟีดและโคลอนจาก เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด อีกเช่นกัน เมื่อได้รับแล้วจึงจะทำการส่งค่าของแอดเดรสเริ่มต้น,ความยาวของข้อมูลไปให้ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด เพื่อที่จะให้ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด นี้ไปนำเอาข้อมูลมาจาก เครื่องที่ต้องการจะพัฒนา

ส่งมาเก็บไว้ที่ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ก่อนในแบบ อาร์เรย์ (Array) เมื่อครบตามความยาวที่กำหนดให้แล้ว เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ก็จะทำการบันทึกลงแผ่นดิสซ์ และปิดไฟล์เมื่อทำการบันทึกหมด ก็จะเป็นการสิ้นสุดการทำงานของฟังค์ชั่นนี้แต่เพียงเท่านี้

ฟังค์ชั่น F4. ในฟังค์ชั่นนี้ จะเป็นการนำเอาข้อมูลจากแฟ้มข้อมูลประเภทตัวอักษร (TEXT FILE) มาแสดงที่จอภาพ ประโยชน์ที่ได้รับจากฟังค์ชั่นนี้ก็คือ นำเอาไว้ใช้ในการตรวจสอบการทำงานของโปรแกรม เช่น นำเอาข้อมูลที่เป็นภาษาแอสแซมบลี้มาแสดงเปรียบเทียบกับการทำงานของโปรแกรมภาษาเครื่อง ว่ามีความสัมพันธ์กันอย่างไร ซึ่งจะทำให้มีความสะดวกต่อการพัฒนาโปรแกรมเป็นอย่างมาก สำหรับการทำงานของโปรแกรมในส่วนนี้สามารถอธิบายได้ดังรูปที่ 3.16



<u>รูปที่ 3.16. โฟว์ชาร์ตการทำงานของการแสดงข้อมูลของไฟล์.</u>

จากรูปที่ 3.16 นี้จะเป็นได้ว่า เริ่มแรกก็จะมีการใส่ชื่อไฟล์ที่ต้องการจะนำเอาข้อมูลออกมาแสดงก่อน จากนั้นก็เป็นการป้อนค่าความเร็วในการแสดง ซึ่งค่านี้ที่จริงเป็น

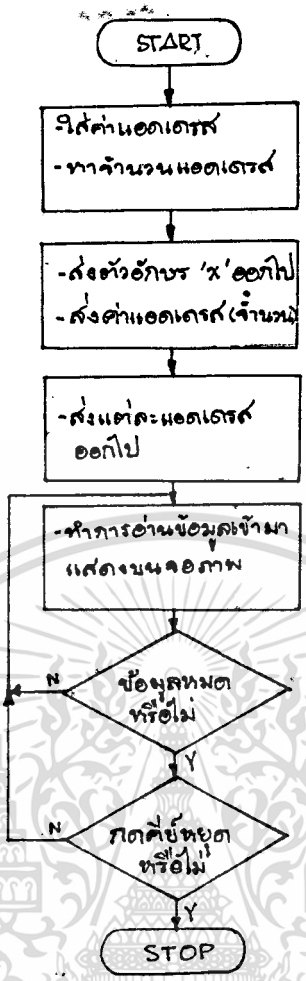
การหน่วงการทำงานในช่วงของการแสดงผลที่หน้าจอ ดังนั้นถ้าค่านี้มีค่ามากการแสดงผลที่หน้าจอก็จะช้า แต่ถ้ามีค่าน้อยๆ การแสดงผลที่หน้าจอจะเร็วขึ้น เมื่อเสร็จสิ้นจากการป้อนชื่อไฟล์และความเร็วในการแสดงผลแล้ว โปรแกรมก็จะทำการตรวจหาไฟล์ชื่อดังกล่าวนี้ในแผ่นดิสซ์ ถ้าไม่มีไฟล์ชื่อนี้ก็จะแจ้งให้ทราบ โดยการแสดงข้อความว่า

'File not exist,or bad parameter'

แต่ถ้ามีไฟล์ชื่อดังกล่าวอยู่ ก็จะทำการเปิดไฟล์และอ่านข้อมูลออกมาแสดงที่หน้าจอภาพจนหมด ก็จะเป็นการสิ้นสุดการทำงานของฟังค์ชั่นนี้ทันที

ฟังค์ชั่น F5. เป็นการแสดงผลของข้อมูลในหน่วยความจำ ณ.ตำแหน่งแอดเดรสที่ต้องการ ของเครื่องที่จะพัฒนา ในขณะที่เครื่องที่ต้องการจะพัฒนากำลังทำงานอยู่ สำหรับการทำงานของฟังค์ชั่นนี้นั้นสามารถแสดงได้ดังรูปที่ 3.17

การทำงานจะเริ่มจากการรอรับค่าแอดเดรสที่ต้องการจะดูข้อมูลนั้น ซึ่งแต่ละแอดเดรสจะต้องแยกออกจากกันด้วยเครื่องหมายคอมม่าและจบด้วยเครื่องหมายจุด เมื่อทำการป้อนแอดเดรสต่างๆเสร็จแล้ว โปรแกรมก็จะทำการหาจำนวนของแอดเดรสที่ป้อนเข้ามาโดยการนับจำนวนของเครื่องหมายคอมม่าจนกระทั่งพบเครื่องหมายจุดนั่นเอง เมื่อได้จำนวนแอดเดรสที่ต้องการแล้ว เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ก็จะส่งตัวอักษร 'X' ไปแจ้งให้ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด ทราบว่าต่อไปนี้เป็นการขอดูข้อมูลในหน่วยความจำของ เครื่องที่ต้องการจะพัฒนา แล้วก็จะมารอการตอบรับว่าทราบแล้วจาก เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด ด้วยการรอรับตัวอักษร 'X' เช่นเดียวกัน จนกระทั่งได้รับ ตัวอักษร'X'นั่นเอง เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม จึงจะเริ่มทำการส่งค่าของจำนวนแอดเดรสที่ต้องการออกไปก่อน แล้วจึงค่อยส่งค่าแอดเดรสแต่ละตัวออกไปให้แก่ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด



<u>รูปที่ 3.17. โฟว์ชาร์ตแสดงการทำงานของ การแสดงผลของข้อมูลในหน่วยความจำ.</u>

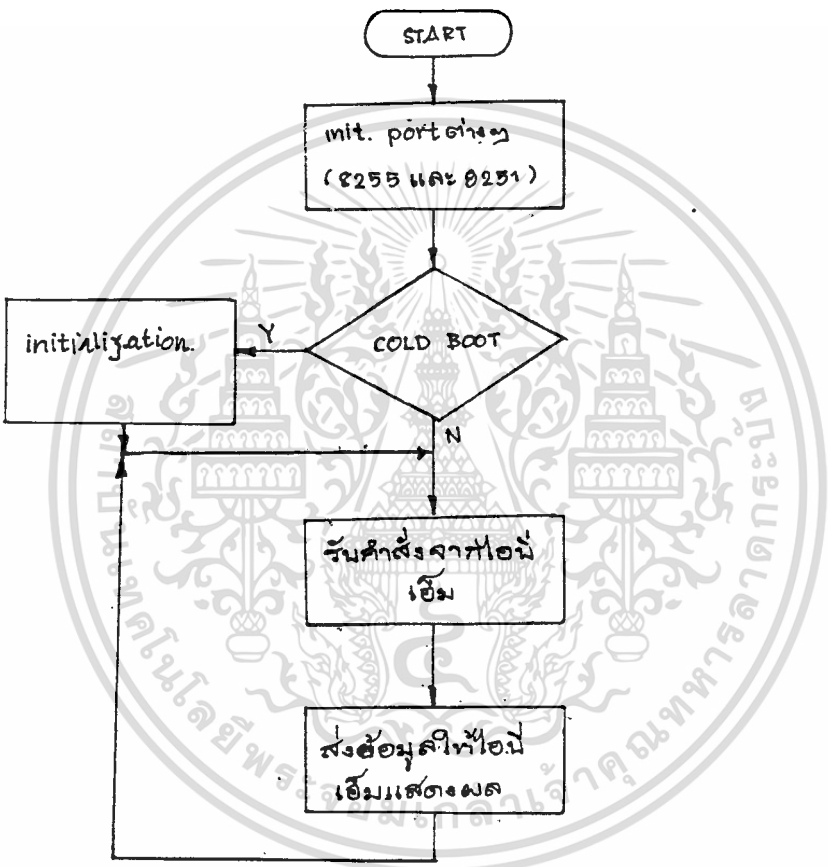
เมื่อการทำงานถึงจุดนี้แล้ว เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ก็จะเริ่มรับเอาข้อมูลที่ส่งมาจาก เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด (ซึ่ง เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของซิงเกิ้ลบอร์ด จะไปนำเอาข้อมูลดังกล่าวนี้มาจาก เครื่องที่ต้องการจะพัฒนา นั่นเอง) เมื่อได้รับข้อมูลแล้วก็จะตรวจดูว่าครบตามจำนวนที่ป้อนเข้าไปหรือไม่ ถ้าไม่ครบก็จะทำการรอรับเอาข้อมูลตัวต่อๆ ไปเข้ามาอีก แต่ถ้าครบแล้วก็จะตรวจดูว่ามีการกดคีย์เพื่อหยุดการทำงานหรือไม่ ถ้าไม่มีก็จะเริ่มทำงานใหม่อีกครั้งหนึ่ง แต่ถ้ามีก็จะปยุดการทำงานของโปรแกรมส่วนนี้ทันที

- โปรแกรมของส่วนภาษาแอสเซมบลี้

ในโปรแกรมส่วนนี้เขียนขึ้นมา(ด้วยภาษาของ Z-80) เพื่อควบคุมการทำงานทั้ง

หมดของ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของชิงเกิ้ลบอร์ด ซึ่งมีหน้าที่หลักอยู่ 2 ประการ คือ ติดต่อรับส่งข้อมูลและคำสั่งกับ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม และอย่างที่สิ่งคือ รับส่งข้อมูลกับ เครื่องที่ต้องการจะพัฒนา

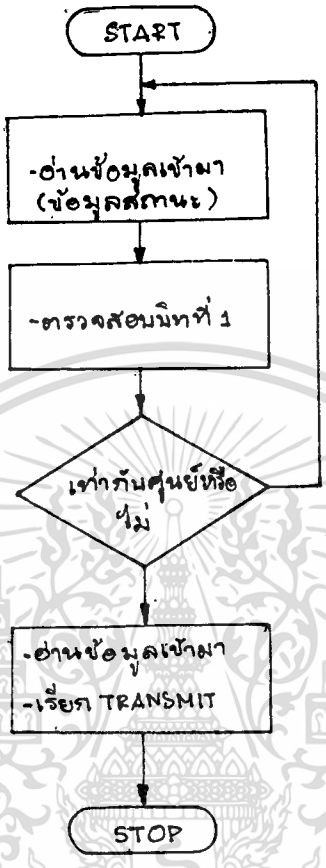
สำหรับการทำงานของโปรแกรมในส่วนนี้นั้นสามารถแสดงได้ดังรูปที่ 3.18



รูปที่ 3.18. โฟว์ชาร์ตแสดงการทำงานของโปรแกรมแอสแซมบลี้.

จากรูปจะเห็นได้ว่า การทำงานเริ่มขึ้นด้วยการกำหนดการทำงานของพอร์ทต่างๆก่อน จากนั้นจึงเข้ามาทำงานที่โปรแกรมย่อยส่วน 'MAIN' ในโปรแกรมย่อยส่วนนี้จะทำการรอรับข้อมูลที่ส่งมาจาก เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม โดยการเรียกใช้โปรแกรมย่อยส่วน 'SCAN' ต่ออีกครั้งหนึ่ง ในโปรแกรมย่อยส่วนนี้ ('SCAN') จะมีการตรวจสอบเงื่อนไขในการรับข้อมูลด้วย ซึ่งถ้าไม่ตรงไม่มีข้อมูลส่งเข้ามาก็จะยังรอรับอยู่ จนกระทั่งได้รับ

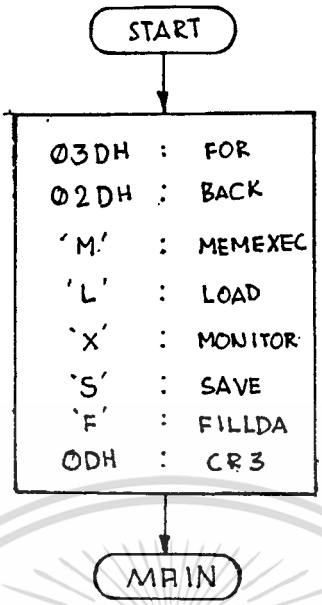
ข้อมูลจึงกลับมาสู่โปรแกรมส่วน 'MAIN' อีกครั้งหนึ่ง



รูปที่ 3.19. โฟว์ชาร์ตแสดงการทำงานของโปรแกรมย่อยส่วน 'SCAN'.

หลังจากนั้นก็จะทำการเคลียร์บัฟเฟอร์ (Clear buffer) โดยการเรียกใช้โปรแกรมย่อยส่วน 'CLRBF' ซึ่งจะเป็นการกำหนดให้ตำแหน่งแอดเดรส OUTPTR เก็บค่าแอดเดรสของ INPBF (INPUT BUFFER) นั่นเอง เสร็จแล้วโปรแกรม 'MAIN' จึงเรียกใช้โปรแกรมย่อยส่วน 'KEYEXEC' (KEY EXECUTE) ต่อ ซึ่งโฟว์ชาร์ตการทำงานของโปรแกรมย่อยส่วน 'KEYEXEC' แสดงได้ดังรูปที่ 3.20

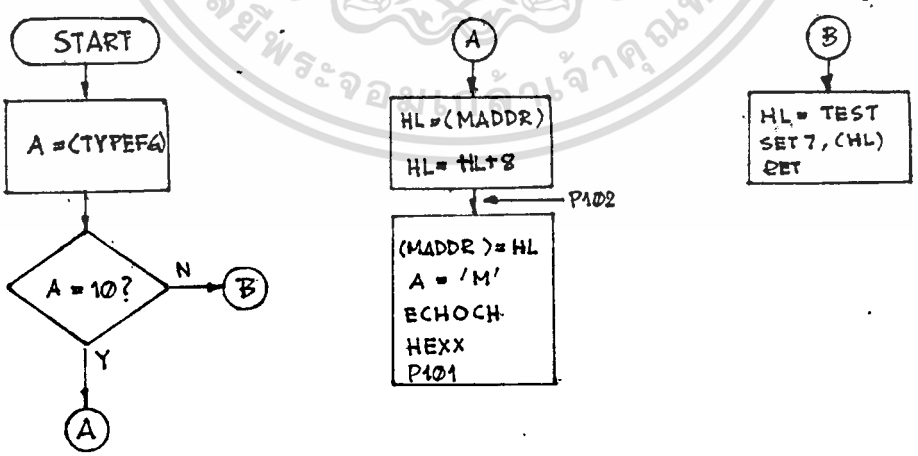
เมื่อเข้ามาสู่โปรแกรมย่อยในส่วนนี้ ก็จะเป็นการนำเอาค่าที่ได้จากการกดคีย์ที่เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนเครื่องไอบีเอ็มส่งมาให้นั้น มาตีความหมายเพื่อที่จะได้ทำงานตามที่ต้อง



รูปที่ 3.20. โฟว์ชาร์ตแสดงการทำงานของโปรแกรมย่อยส่วน 'KEYEXEC'.

การต่อไป จากรูปที่ 3.20 จะเห็นได้ว่าสามารถที่จะอธิบายการทำงานตามค่าที่กดเข้ามาได้ดังนี้ คือ

1. เมื่อได้รับเครื่องหมายเท่ากับ ก็จะเข้าไปทำโปรแกรมย่อยส่วน 'FOR' ซึ่งก็คือการแสดงผลของข้อมูลภายในแอดเดรสถัดขึ้นไปไปอีก 7 แอดเดรส (รวมแอดเดรสปัจจุบันด้วยอีกหนึ่ง ก็จะเป็น 8 แอดเดรส) โปรแกรมย่อยในส่วนนี้สามารถอธิบายได้ดังรูปที่ 3.21



รูปที่ 3.21. โฟว์ชาร์ตแสดงการทำงานของโปรแกรมย่อยส่วน FOR.

จะเห็นได้ว่า เมื่อเริ่มแรกนั้นจะเริ่มทำการตรวจสอบค่าที่เก็บไว้ในแอดเดรส ØFF8ØH (TYPEFG) ซึ่งแอดเดรสนี้จะเก็บค่าที่แสดงว่าขณะนี้กำลังทำงานอยู่ในโหมดใด แล้วนำเอาค่า

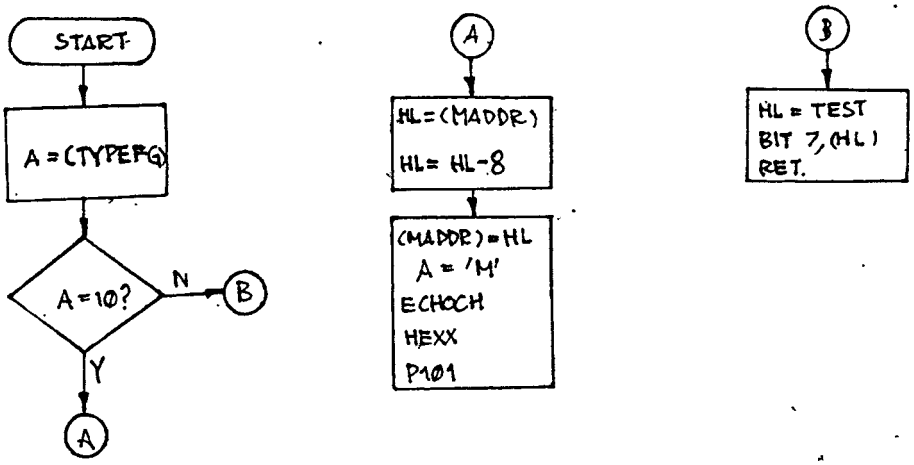
ที่อยู่ในแอดเดรสนี้มาเปรียบเทียบกับ 10H ว่าเท่ากันหรือไม่

ถ้าไม่เท่ากัน ก็จะแสดงว่าไม่ได้อยู่ในโหมดของการกระทำกับหน่วยความจำ (เพราะว่า เมื่อเริ่มการทำงานในโหมดของหน่วยความจำ จะทำการกำหนดให้ที่แอดเดรสนี้ (FF80H) มีค่าเท่ากับ 10H) จึงไปทำงานต่อที่ โปรแกรมย่อยส่วน IGNORE เพื่อที่จะได้กับเข้าโปรแกรมเมนต่อไป

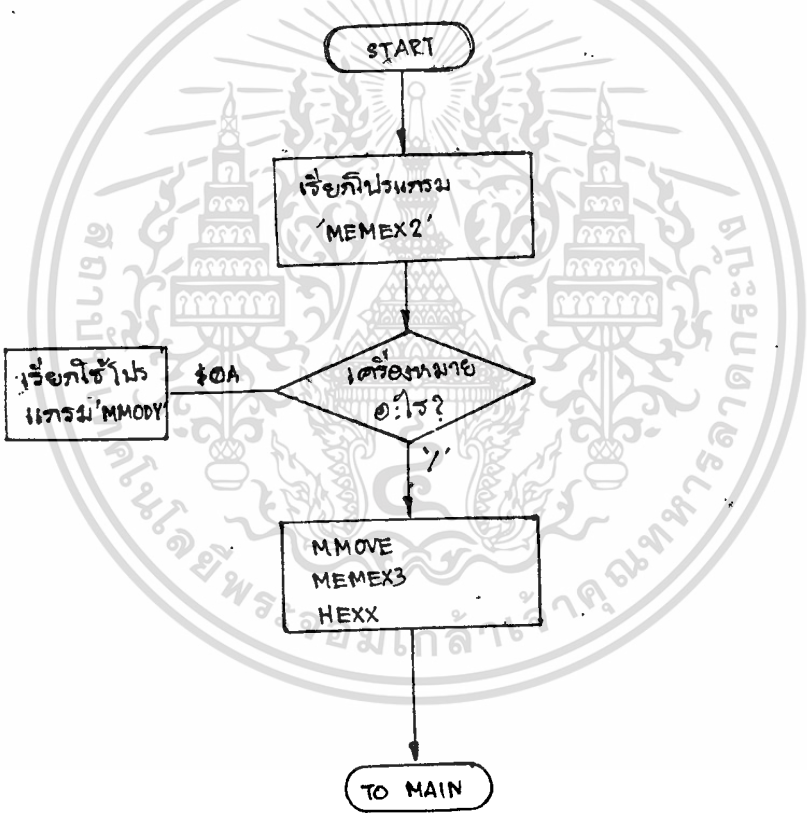
ถ้าเท่ากัน ก็จะไปทำงานต่อที่โปรแกรมย่อยส่วน 'MFOR' ที่โปรแกรมย่อยส่วน 'MFOR' นี้ จะนำเอาข้อมูลที่เก็บเอาไว้ใน MADDR เข้ามาให้แก่ รีจิสเตอร์เอชแอล (REGISTER HL) แล้วเพิ่มค่าขึ้นอีกเท่ากับ 8 จากนั้นจึงนำเอาค่าที่เพิ่มขึ้นนี้ กลับเข้าไปเก็บไว้ที่ MADDR ตามเดิม แล้วจึงค่อยเรียกใช้โปรแกรมย่อยส่วน 'ECHOCH' เพื่อ ทำการสร้างเครื่องหมาย '>' และ '=' เข้าไปเก็บไว้ที่ INPUT BUFFER และเรียกใช้โปรแกรมย่อยส่วน HEXX ต่อ เพื่อที่จะที่จะทำการเปลี่ยนค่าของแอดเดรสที่เก็บเอาไว้ใน รีจิสเตอร์เอชแอล (ซึ่งก็เป็นค่าของแอดเดรสที่ต้องการจะนำเอาข้อมูลออกมานั่นเอง) เป็นค่ารหัสแอสกี้ แล้วจึงไปทำงานต่อที่ P101

2. เมื่อได้รับเครื่องหมายลบ ก็มีการทำงานคล้ายกันกับเมื่อกดคีย์เครื่องหมายเท่ากับนั่นเอง แต่ต่างกันที่จุดมุ่งหมายคือ เมื่อกดคีย์เครื่องหมายลบนี้แล้ว จะเป็นการแสดงข้อมูลของแอดเดรสที่ถัดลงไปจากแอดเดรสปัจจุบันอีก 7 แอดเดรส ดังนั้นการทำงานของโปรแกรมย่อยส่วนนี้จึงต่างจากโปรแกรมย่อย 'FOR' ตรงที่ จะทำการลดค่าของ ข้อมูลที่เก็บไว้ในแอดเดรส FF80H (แทนที่จะเพิ่ม) ซึ่งสามารถแสดงการทำงานได้ดังรูปที่ 3.22

3. เมื่อรับตัวอักษร 'M' จาก เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนเครื่องไอบีเอ็ม ก็จะเป็นการกระทำเกี่ยวกับหน่วยความจำ ซึ่งมีการทำงานแสดงดังรูปที่ 3.23 เมื่อได้รับตัวอักษร 'M' ก็จะทำการกำหนดให้ค่า TYPEFG เท่ากับ 10H แล้วก็จะไปทำ



รูปที่ 3.22. โฟว์ชาร์ต การทำงานของโปรแกรมย่อยส่วน 'BACK'.

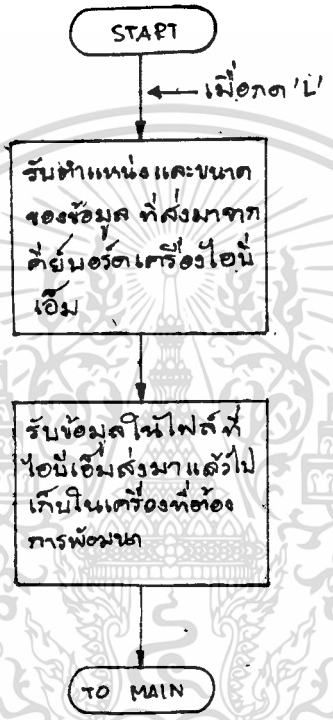


รูปที่ 3.23. โฟว์ชาร์ตแสดงการทำงานเกี่ยวกับหน่วยความจำ.

งานที่ โปรแกรมย่อยส่วน 'MEMEXEC' ซึ่งจะทำการเรียกใช้โปรแกรมย่อยส่วน 'MEMEX2' หาค่ารหัสของเครื่องหมายไลน์ฟีด ถ้าเป็นเครื่องหมายไลน์ฟีดแล้วก็จะไปทำงานต่อที่ โปรแกรมย่อยส่วน 'MMODY1' แต่ถ้าเป็นเครื่องหมายสแลท (/) ก็จะไปทำงานที่ โปรแกรมย่อยส่วน 'MMOVE' จากนั้นก็เรียกใช้โปรแกรมย่อยส่วน MEMEX3 เพื่อและโปรแกรม

ย่อยส่วน HEXX ต่อจากนั้นจึงเรียกใช้โปรแกรมย่อยส่วน MEM3

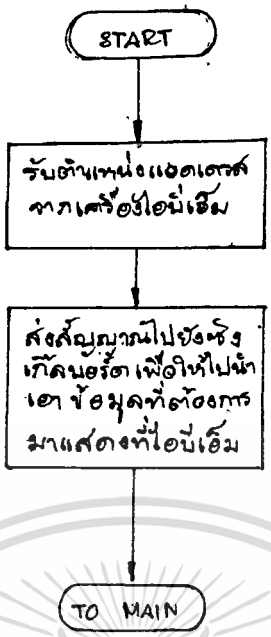
4. เมื่อรับตัวอักษร 'L' จะเป็นการทำงานเกี่ยวกับการโหลดข้อมูลจากเครื่องที่ต้องการจะพัฒนา เกื่อส่งต่อไปให้ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนเครื่องไอบีเอ็ม ซึ่งสามารถแสดงการทำงานได้ดังรูปที่ 3.24



รูปที่ 3.24. โฟว์ชาร์ตการทำงานของการโหลดข้อมูล.

ก่อนอื่นจะต้องนำเอาความยาวของข้อมูลที่ต้องการเข้ามาก่อน จากนั้นจึงทำการขอใช้บัสไปยัง เครื่องที่ต้องการจะพัฒนา เมื่อขอได้แล้วจึงทำการอ่านข้อมูลตามแอดเดรสที่ต้องการ

5. เมื่อรับตัวอักษร 'X' จะเป็นการทำงานเกี่ยวกับการขอดูข้อมูลที่เปลี่ยนแปลงเสมอของเครื่องที่ต้องการจะพัฒนาตามที่แอดเดรสระบุมาจากเครื่องพัฒนารับบคอมพิวเตอร์ส่วนเครื่องไอบีเอ็มส่งมา ซึ่งรูปที่ 3.25 แสดงการทำงานของการขอดูข้อมูลนี้

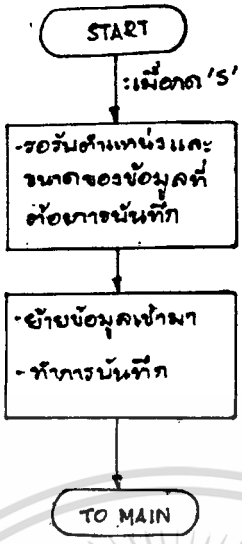


รูปที่ 3.25. โฟว์ชาร์ตการทำงานของการขอดูข้อมูลของเครื่องที่ต้องการจะพัฒนา.

การทำงานของโปรแกรมย่อยส่วนนี้ จะเริ่มด้วยการรับเอาจำนวนของแอดเดรสที่ต้องการเข้ามาก่อน จากนั้นก็ทำการขอใช้บัสไปยังเครื่องที่ต้องการจะพัฒนาอีกเช่นกัน แล้วจึงจะเริ่มทำการอ่านข้อมูลเข้ามา เพื่อส่งต่อไปให้เครื่องพัฒนาระบบส่วนของเครื่องไอบีเอ็ม ให้นำไปแสดงที่หน้าจอภาพต่อไป

6. เมื่อรับตัวอักษร 'S' สำหรับโปรแกรมย่อยในส่วนนี้ สามารถแสดงการทำงานของโปรแกรมได้ดังรูปที่ 3.26

ในโปรแกรมส่วนนี้นั้นจะทำการอ่นข้อมูลที่ต้องการจะบันทึกเก็บเอาไว้ ส่งต่อไปให้ เครื่องพัฒนาระบบส่วนของเครื่องไอบีเอ็ม ทำการบันทึกลงแผ่นดิสซ์ต่อไป การทำงานจะเริ่มด้วยการรับค่าของแอดเดรสเริ่มต้นที่ต้องการจะบันทึกเข้ามาก่อน ตามมาด้วยค่าความยาวของข้อมูลที่ต้องการ หลังจากนั้นก็ทำการขอใช้บัสไปยังเครื่องที่ต้องจะพัฒนา เพื่อที่จะได้อ่านข้อมูลออกมาส่งต่อไปให้ เครื่องพัฒนาระบบส่วนของเครื่องไอบีเอ็ม ต่อจนกระทั่งครบตามความยาวของข้อมูลที่กำหนดเอาไว้



รูปที่ 3.26. โฟว์ชาร์ตแสดงการทำงานของการบันทึกข้อมูล.

7. เมื่อรับตัวอักษร 'F' จะเป็นโปรแกรมสำหรับใช้ในการแก้ไขข้อมูลที่แอดเดรสต่างๆ โดยช่วงแรกของการทำงาน จะทำการตรวจสอบดูว่า แอดเดรสที่ป้อนเข้ามานั้น เป็นตำแหน่งของหน่วยความจำแบบแรมหรือไม่ ซึ่งถ้าใช่ก็จะเริ่มทำการเปลี่ยนแปลงข้อมูลที่แอดเดรสนั้นๆ ตามค่าที่ส่งมาจาก เครื่องพัฒนาระบบส่วนของเครื่องไอบีเอ็ม ต่อไป

บทที่ 4

การทดลองและผลการทดลอง

การทดลองการทำงานของเครื่องพัฒนาระบบคอมพิวเตอร์นี้ เริ่มจากการเชื่อมต่อสายต่างๆ ที่จำเป็นดังรูปที่ 4.1 และเริ่มตรวจสอบการทำงานของเครื่อง ผลที่ได้จากการทดลองสรุปได้ว่า มีการทำงานที่ถูกต้องเป็นไปตามที่ต้องการทุกอย่าง แต่ก็ยังมีอยู่ฟังค์ชั่นหนึ่ง ที่มีการทำงานล่าช้าอยู่ คือ การหยุดการทำงานของการขอดูข้อมูล (มอนิเตอร์ริ่ง) ซึ่งผู้ใช้จำเป็นที่จะต้องกดคีย์เอ็นเตอร์จนกว่าจะมีเสียงเตือนให้ปล่อย ทำให้เกิดความล่าช้าเล็กน้อย และอีกสิ่งหนึ่งคือ การขอดูข้อมูลนี้ยังไม่สามารถที่จะขอดูข้อมูลที่มีการเปลี่ยนแปลงอย่างรวดเร็วได้ แต่การทำงานก็ยังคงถูกต้องตามความต้องการอยู่

บทที่ 5

บทวิจารณ์และสรุป

บทวิจารณ์

สิ่งที่ยังมีข้อบกพร่องอยู่อีกประการหนึ่งก็คือ การทำงานของเครื่องในฟังค์ชั่นการขอดูหน่วยความจำ (มอนิเตอร์ริ่ง) ในสาเหตุที่ว่าการหยุดการทำงานโดยกดคีย์เอ็นเตอร์นั้นไม่แน่นอนเท่าที่ควร ซึ่งผู้ใช้จะต้องกดคีย์นานกว่าปกติ จนกว่าจะมีเสียงเตือนให้ปล่อยคีย์ และอีกประการหนึ่งก็คือ การแสดงข้อมูลที่เปลี่ยนแปลงตลอดเวลานั้น ยังไม่สามารถที่จะนำเอาข้อมูลที่มีการเปลี่ยนแปลงอย่างรวดเร็วมาแสดงที่หน้าจอได้ สาเหตุเนื่องมาจาก ความล่าช้าในการทำงานของเครื่องไอบีเอ็มนั่นเอง

บทสรุป

การทำงานของเครื่องพัฒนาระบบคอมพิวเตอร์ที่สร้างขึ้นนี้ มีประสิทธิภาพเป็นที่น่าพอใจ ซึ่งสามารถทำงานในฟังค์ชั่นต่างๆ ได้ถูกต้องและรวดเร็วในขีดความสามารถเท่าที่จะทำได้ ซึ่งสาเหตุของความเร็วในการทำงานของเครื่องคอมพิวเตอร์ที่ใช้นี้ก็ทำให้ ขีดความสามารถของเครื่องลดลงไปเล็กน้อย แต่ก็เป็นการทำงานที่ถูกต้องเช่นกัน

ภาคผนวก

วิธีการใช้เครื่องพัฒนาระบบคอมพิวเตอร์

ก่อนที่จะใช้งานของระบบนี้ จะต้องทำการเชื่อมโยงระหว่างเครื่องพัฒนาระบบคอมพิวเตอร์ส่วนเครื่องไอบีเอ็มกับเครื่องพัฒนาระบบคอมพิวเตอร์ส่วนซิงเกิ้ลบอร์ดให้ถูกต้องเสียก่อนโดยการนำเอาสายรับส่งแบบอาร์เอส 232-ซี ที่จัดไว้ให้ต่อเข้าที่กอร์ทสื่อสารอนุกรมของ เครื่องพัฒนาระบบคอมพิวเตอร์ส่วนเครื่องไอบีเอ็ม เมื่อเสร็จแล้วจึงทำการเชื่อมโยงเครื่องที่ต้องการจะพัฒนา เข้ากับเครื่องพัฒนาระบบคอมพิวเตอร์ส่วนซิงเกิ้ลบอร์ด ที่จัดซ็อคเก็ตที่จัดไว้ เมื่อเริ่มเปิดเครื่อง ก็จะมีการแสดงข้อความว่า

'Press F1 to Main Menu....'

ซึ่งผู้ใช้จะต้องกด F1 เพื่อเข้าสู่เมนเมนู โดยที่เมนเมนูจะแสดงข้อความต่อไปอีกดังนี้

MAIN MENU

F1 : Main Menu

F2 : Load File

F3 : Save File

F4 : Load Text file

F5 : Monitoring

F6 : Help !

M : Memory

F : Fill Memory

R : Reset

Esc : Exit to DOS

Select Menu...

ซึ่งสามารถจะอธิบายได้ตามลำดับดังนี้คือ

1. เมื่อกดฟังค์ชั่นคีย์ที่ 1 จะเป็นการแสดงเมนเมนูตามรูปข้างบนนี้นั่นเอง

2. เมื่อกดฟังค์ชั่นคีย์ที่ 2 จะเป็นการโหลดไฟล์จากแผ่นดิสซ์ เมื่อกดคีย์นี้จะแสดงข้อความให้ผู้ใช้ป้อนสิ่งที่จำเป็นดังนี้คือ

Load FileName:

ซึ่งสิ่งที่ผู้ใช้จะต้องป้อนในขณะนี้ก็คือ ชื่อไฟล์ เว้นวรรคด้วยเครื่องหมายคอมม่า แล้วตามด้วยแอดเดรสเริ่มต้นในหน่วยความจำของเครื่องที่ต้องการจะพัฒนาที่ต้องการโหลดข้อมูลเข้าไปเก็บเอาไว้ จากนั้นก็ตามมาด้วยเครื่องหมายคอมม่าอีกครั้งจึงจะป้อนความยาวโดยประมาณของไฟล์ที่ต้องการจะโหลดลงมา

ตัวอย่างเช่น ถ้าต้องการโหลดไฟล์ชื่อ PROJ.TSK ที่อยู่ในดิสซ์ไดร์วเครื่อง B ลงมาสู่หน่วยความจำของเครื่องที่ต้องการจะพัฒนา โดยให้โหลดลงมาเก็บไว้เริ่มต้นตั้งแต่แอดเดรสที่ F800H และไฟล์ข้อมูลดังกล่าวมีความยาวโดยประมาณเท่ากับ 500 ไบท์ สิ่งที่ผู้ใช้จะต้องป้อนก็คือ

Load FileName:B:PROJ.TSK,F800,500

แล้วก็กดคีย์เอ็นเตอร์ (หรือที่รู้จักกันในชื่อ คีย์รีเทอน :RETURN KEY) ก็จะเป็นการสิ้นสุดของการโหลดไฟล์ แต่ถ้าชื่อไฟล์ที่ป้อนเข้าไปนั้นไม่มีอยู่ในแผ่นดิสซ์ หรือป้อนค่าต่างๆ เกิน (หรือขาดไป) ตามที่โปรแกรมกำหนด โปรแกรมก็จะทำการแจ้งให้ผู้ใช้ทราบ โดยการแสดงข้อความว่า

File not exist,or bad parameter

3. เมื่อกดฟังค์ชั่นคีย์ที่ 3 จะเป็นการบันทึกไฟล์ลงไปในแผ่นดิสซ์นั่นเอง เมื่อผู้ใช้กดฟังค์ชั่นคีย์นี้แล้วจะมีการแสดงข้อความดังนี้

Save Filename:

คอมม่าอีกครั้งจึงจะป้อนความยาวโดยประมาณของข้อมูลที่ต้องการจะบันทึก

ตัวอย่างเช่น ถ้าต้องการเก็บข้อมูลของเครื่องที่ต้องการจะพัฒนา ไว้เริ่มต้นตั้งแต่แอดเดรสที่ F800H และข้อมูลดังกล่าวมีความยาวโดยประมาณเท่ากับ 500 ไบท์ แล้วนำไปเก็บที่ไดร์ว B ในไฟล์ชื่อ PROJ.TSK สิ่งที่ผู้ใช้จะต้องป้อนก็คือ

Save Filename:B:PROJ.TSK,F800,500

แล้วก็กดคีย์เอ็นเตอร์ ก็จะเป็นการสิ้นสุดของการบันทึกข้อมูล แต่ถ้าการป้อนค่าต่างๆเกิน (หรือขาดไป) ตามที่โปรแกรมกำหนด โปรแกรมก็จะทำการแจ้งให้ผู้ใช้ทราบ โดยการแสดงข้อความว่า

File not exist,or bad parameter

4. เมื่อกดฟังค์ชั่นคีย์ที่ 4 ก็จะเป็นการโหลดไฟล์ข้อมูลของตัวอักษรออกมาแสดงที่จอภาพซึ่งเมื่อกดคีย์นี้แล้ว สิ่งที่จะแสดงบนจอภาพคือ

Load Text FileName:

ผู้ใช้จะต้องป้อนชื่อของไฟล์ที่ต้องการจะโหลดเข้าไป แล้วกดเอ็นเตอร์ เครื่องก็จะแสดงข้อความต่อไปอีกว่า

********* Speed [1..50]:

เป็นการให้ผู้ใช้ป้อนค่าความเร็วในการโหลดไฟล์ออกมาแสดงที่จอภาพนั่นเอง ค่านี้ถ้ามีค่ามากจะทำให้ความเร็วของการแสดงบนหน้าจอช้า แต่ถ้ามีค่าน้อยการแสดงจะเร็วขึ้น เมื่อป้อนเสร็จแล้วก็กดคีย์เอ็นเตอร์อีกครั้งหนึ่ง เครื่องก็จะทำการตรวจสอบว่ามีไฟล์นี้อยู่หรือไม่ ถ้ามีก็จะทำการอ่านออกมา แต่ถ้าไม่มีก็จะทำการแจ้งให้ผู้ใช้ทราบ โดยแสดงข้อความว่า

File not exist,or bad parameter

5. เมื่อกดฟังค์ชั่นคีย์ที่ 5 ฟังค์ชั่นการขอดูข้อมูลในหน่วยความจำของเครื่องที่ต้องการจะพัฒนา เมื่อกดคีย์นี้แล้ว สิ่งที่เครื่องจะแสดงบนหน้าจอคือ

Monitoring Address:

ผู้ใช้จะต้องป้อนค่าแอดเดรสที่ต้องการเข้าไป โดยแยกแต่ละแอดเดรสด้วย เครื่องหมายคอมม่าและจบด้วยเครื่องหมายจุด เช่น เมื่อต้องการทำการดูข้อมูลในแอดเดรสที่ F800H, F801H,F802H และ F803H สิ่งที่ผู้ใช้จะต้องป้อนก็คือ

Monitoring Address:F800,F801,F802,F803.

เครื่องก็จะทำการทำงานต่อไป โดยการแสดงแอดเดรสและข้อมูลที่หน้าภาพเช่น

MONITORING : F800 ==> 23

MONITORING : F801 ==> 33

MONITORING : F802 ==> 43

MONITORING : F803 ==> 53

ซึ่งเครื่องจะทำการแสดงเช่นนี้จนกระทั่งผู้ใช้กดคีย์เอ็นเตอร์จึงจะหยุดการทำงาน

6. เมื่อกดฟังค์ชั่นคีย์ที่ 6 เป็นฟังค์ชั่นที่ทำหน้าที่ คอยแสดงข้อความที่จำเป็น เมื่อผู้ใช้ไม่ทราบว่าจะทำงานต่อไปได้อย่างไร ซึ่งเมื่อกดคีย์นี้แล้วจะแสดงขอความที่เป็นรูปแบบของคำสั่งต่างๆ

7. เมื่อกดคีย์ตัวอักษร 'M' จะเป็นการทำงานในโหมดของการแสดงในแอดเดรสที่ต้องการ เมื่อผู้ใช้กดคีย์นี้แล้ว ก็จะรอนับค่าแอดเดรสจากผู้ใช้ เช่นว่าต้องการดูข้อมูลที่แอดเดรส F800H เป็นต้นไป ผู้ใช้ก็ป้อนเพียงแอดเดรสนี้ค่าเดียวเท่านั้น คือ

F800

แล้วกดคีย์เอ็นเตอร์ ก็จะได้ข้อมูลแสดงอยู่ที่หน้าจอภาพดังนี้ (ข้อมูลที่แสดงในตัวอย่างนี้สมมุติขึ้นเท่านั้น)

<M>= F800 23 45 67 89 0A BC DE F1

แต่ถ้า ในระหว่างที่ผู้ใช้ป้อนค่าแอดเดสเข้าไปนั้น ทำการป้อนแอดเดรสผิดไปหรือไม่ใช่แอดเดรสที่ต้องการเช่น

F8GØ

ซึ่งในที่นี้ป้อนผิดที่ตัวอักษร G ผู้ใช้ก็สามารถที่จะใช้คีย์แบคสเปท (BACK SPACE KEY) กลับมาแก้ไขได้ตามต้องการ

8. เมื่อกดคีย์ตัวอักษร 'F' ป้อนแอดเดรสเริ่มต้น แอดเดรสสุดท้าย แอดเดรสเริ่มต้นของปลายทาง โดยที่ค่าแอดเดรสเริ่มต้นจะต้องน้อยกว่าแอดเดรสปลายทางเสมอ

9. เมื่อกดคีย์ตัวอักษร 'R' จะเป็นการรีเซท (RESET) เครื่องที่ต้องการจะพัฒนา ผ่านทางเครื่องพัฒนาระบบคอมพิวเตอร์ส่วนของเครื่องไอบีเอ็ม ซึ่งจะทำให้เกิดความสะดวกในการทำงานเป็นอย่างมาก และไม่ว่าผู้ใช้กำลังทำงานใดอยู่ก็ตาม หากกดคีย์นี้แล้วจะเป็นการรีเซทเครื่องที่ต้องการจะพัฒนาทุกครั้ง

10. เมื่อกดคีย์เอสเคป (Esc) จะเป็นการหยุดการทำงานของเครื่องพัฒนาระบบส่วนของเครื่องไอบีเอ็ม ทันที

```
Program Microprocessor_Development_System;

  {*************************************************************************
  *                                                                        *
  *    Program Monitor for Microprocessor Development System (MDS.)        *
  *                                                                        *
  *         Programmer  : Mr. Santidej Mansawasdi                          *
  *                     : Mr. Bancha   Watanasoponwong                     *
  *         Advisor     : Mr. Viriya   Kongratana                          *
  *         Section     : Computer                                         *
  *         Department  : Industial Instrumention Technology               *
  *         Faculty     : Engineering                                      *
  *                                                                        *
  *         King Mongkut's Institue of Technology Ladkrabang               *
  *                           KMITL.                                       *
  *                                                                        *
  *************************************************************************

  *************************************************************************
  *                                                                        *
  * Turbo POWER TOOLS Required Procedures and Functions                    *
  * (C)Copyright Blaise Computing 1985                                     *
  * Version 1.0  July 15, 1985                                            *
  *                                                                        *
  * Created on Wednesday July 22, 1987  7:49 AM                            *
  *                                                                        *
  *************************************************************************)

  {$I C:\TOOLS.INC    }
  {$I C:\RESETSCN.INC}
  {$I C:\MOVESCN.INC }
  {$I C:\CSIZESCN.INC}
  {$I C:\COFFSCN.INC }
  {$I C:\GOTOSCN.INC }
  {$I C:\WHERESCN.INC}
  {$I C:\RCHARSCN.INC}
  {$I C:\VERTSCN.INC }
  {$I C:\BOXSCN.INC  }
  {$I C:\MAKEWIN.INC }
  {$I C:\VERTWIN.INC }
  {$I C:\DISPWIN.INC }
  {$I C:\CURWIN.INC  }
  {$I C:\REMWIN.INC  }
  {$I C:\GOTOWIN.INC }
  {$I C:\CLEARWIN.INC}
  {$I C:\DOSINKEY.INC}
  {$I C:\PAUSEKEY.INC}

const
  F1            = #59;
  F2            = #60;
  F3            = #61;
  F4            = #62;
  F5            = #63;
  F6            = #64;
  Esc           = #27;
  Window1X1     = 21;
```

```
Window1Y1      = 3;
Window1X2      = 58;
Window1Y2      = 21;
Window1Fore    = lightRed;
Window1Back    : integer = Brown;
Window1BorType = 10;
Window1BorFore = LightGreen;
Window1BorBack = Black;
Window2X1      = 20;
Window2Y1      = 2;
Window2X2      = 60;
Window2Y2      = 10;
Window2Fore    = White;
Window2Back    : integer = Blue;
Window2BorType = 10;
Window2BorFore = Yellow;
Window2BorBack = Black;
Window3X1      = 20;
Window3Y1      = 14;
Window3X2      = 60;
Window3Y2      = 22;
Window3Fore    = LightGreen;
Window3Back    : integer = Red;
Window3BorType = 10;
Window3BorFore = LightMagenta;
Window3BorBack = Black;
Window4X1      = 22;
Window4Y1      = 9;
Window4X2      = 79;
Window4Y2      = 23;
Window4Fore    = White;
Window4Back    : integer = Blue;
Window4BorType = 10;
Window4BorFore = LightRed;
Window4BorBack = Black;
Window5X1      = 50;
Window5Y1      = 4;
Window5X2      = 75;
Window5Y2      = 21;
Window5Fore    = Yellow;
Window5Back    : integer = LightGray;
Window5BorType = 10;
Window5BorFore = LightMagenta;
Window5BorBack = Black;
Window6X1      = 4;
Window6Y1      = 5;
Window6X2      = 40;
Window6Y2      = 24;
Window6Fore    = Yellow;
Window6Back    : integer = Green;
Window6BorType = 10;
Window6BorFore = LightRed;
Window6BorBack = Black;
Window7X1      = 25;
Window7Y1      = 12;
Window7X2      = 55;
```

```
  Window7Y2      = 12;
  Window7Fore    = Lightgreen;
  Window7Back    : integer = Black;
  Window7BorType = 10;
  Window7BorFore = LightRed;
  Window7BorBack = DarkGray;
  Window8X1      = 2;
  Window8Y1      = 2;
  Window8X2      = 79;
  Window8Y2      = 24;
  Window8Fore    = White;
  Window8Back    : integer = Blue;
  Window8BorType = 10;
  Window8BorFore = LightMagenta;
  Window8BorBack = Black;
type
  string80       = string [80];
  DeviceType     = (CRT,PRN);
  __Range_String = string[4];
  __File_Down_Up = file of byte;
  __File_Text    = text;
var
  DocumentFile     : Text;
  Ch               : char;
  Device           : DeviceType;
  DeviceName       : string [4];
  __Check          : boolean;
  Window1          : _WindowPtr;
  Window2          : _WindowPtr;
  Window3          : _WindowPtr;
  Window4          : _WindowPtr;
  Window5          : _WindowPtr;
  Window6          : _WindowPtr;
  Window7          : _WindowPtr;
  Window8          : _WindowPtr;
  __Line           : string80;
  __Databit        : integer;
  __Size_File_Char : string[4];
  __Data_Index     : integer;
  __Size_File      : integer;
  __Command_Word   : byte;
  bit01            : byte;
  bit2             : byte;
  bit3             : byte;
  bit4             : byte;
  bit567           : byte;
  __Data           : byte;
  __Length,i       : byte;
  __Speed          : byte;
  __Character      : char;
  __FileVar        : __File_Down_Up;
  __File_Name      : string[15];
  __Filename_Address_Range     : string[30];
  __Memory_Address : string[4];
  __Memory_Range   : string[4];
  __Size_File_Dec  : integer;
  __Data_Save      : array[1..8192] of byte;
```

```
  __Start          : boolean;
  __Memory_Mode_Flag : Boolean;
  __Checker_Memory   : integer;
  __Range_File     : integer;
  __Temp_Counter   : integer;
  __Size_Value     : byte;
  __Size_Each_Char : string[1];

{************************** end of variable ****************************}

procedure __Initial_8250;
var LSB , MSB : byte;

{    Buad Rate  =  9600    }
{    __Data     =  8 bits  }
{    Parity     =  Odd     }
{    Stop bit   =  2       }

begin
  MSB := $00;
  LSB := $0C;
  Port [$3FB] := $80;  { Buad Rate Divisor Register }
  Port [$3F8] := LSB;  { LSB Value for BuadRate }
  Port [$3F9] := MSB;  { MSB Value for BuadRate }
  bit01 := $03;
  bit2 := $04;
  bit3 := $08;
  bit4 := $00;
  bit567 := $00;
  __Command_Word := bit01 OR bit2 OR bit3 OR bit4 OR bit567;
  Port [$3FB] := __Command_Word;
end;

procedure Disable_Interrupt;
begin
  Port [$3F9] := $00;
end;

procedure __Transmit;
begin { Teansmit }
  Port [$3F8] := __Data;
end; {__Transmit}

procedure __Init_Interface;
begin
  __Initial_8250 ;
  Disable_Interrupt;
end; { __Init_Interface }

procedure __Receive;
begin { __Receive }
  __Character := Chr(Port[$3F8]);
end { __Receive };

procedure __Receive_Message;
begin
  if port[$03FD] AND $01 = $01 then
```

```
    begin
      __Receive;
      write (__Character);
    end;
end;

procedure __Receive_Message_mon;
begin
  if port[$03FD] AND $01 = $01 then
     __Receive;
end;

procedure __Transmit_Message;
begin
   __Character := upcase(__Character);
   __Data := ord(__Character);
   __Transmit;
end;

procedure Converse_Hex (var __Number : integer;
                        var __Memory_Range : __Range_String);
type
   __Character = string[1];
var
  __Hex             : array [1..4] of __Character;
  __Num             : array [1..4] of byte;
  __Number1,__Number2 : integer;
  i                 : byte;

procedure __Check_byte (__Number:byte);
begin
   __Num[1] := __Number mod 16;
   __Num[2] := __Number div 16;
end;
procedure __Check_integer(__Number:integer);
begin
   __Number1 := __Number div 256;
   __Number2 := __Number mod 256;
   __Check_byte (__Number2);
   __Num[3] := __Number1 mod 16;
   __Num[4] := __Number1 div 16;
end;
procedure __Converse_Hexchar;
begin
   for i := 1 to 4 do
   begin
     if __Num[i] >= 10 then
     case __Num[i] of
     10 : __Hex[i] := 'A';
     11 : __Hex[i] := 'B';
     12 : __Hex[i] := 'C';
     13 : __Hex[i] := 'D';
     14 : __Hex[i] := 'E';
     15 : __Hex[i] := 'F';
     end
     else
     str (__Num[i],__Hex[i]);
```

```
    end;
  end;

  begin
    __Num[1] := 0; __Num[2] := 0; __Num[3] := 0; __Num[4] := 0;
    if __Number <= 255
      then
        __Check_byte (__Number)
      else
        __Check_integer (__Number);
    __Converse_Hexchar;
    __Memory_Range := concat (__Hex[4],__Hex[3],__Hex[2],__Hex[1]);
  end;

  function __Check_IO (var fi : __File_Down_Up) : boolean;
  begin
   {$I-} reset (fi); {$I+}
   __Check_IO := IOresult = 0;
  end;

  function __Check_IOtext (var fi : __File_Text) : boolean;
  begin
   {$I-} reset (fi); {$I+}
   __Check_IOtext := IOresult = 0;
  end;

  procedure __Load_Textfile;
  begin
    write ('Load Text FileName:');read (__File_Name);
    write (' ----> __Speed [1..50]:');readln (__Speed);
    assign (DocumentFile,__File_Name);
    if __Check_IOtext (DocumentFile) then
      begin
        reset (DocumentFile);
        __COffScn (true);
        while NOT EOF(DocumentFile) do
          begin
            Readln (DocumentFile,__Line);
            __Length := length(__Line);
            for i := 1 to __Length do
              begin
                Ch := copy (__Line,i,1);
                write (Ch);
                delay(__Speed);
              end;
            write(Chr($0D));write(Chr($0A))
          end;
        Close (DocumentFile)
      end
      else writeln ('File not exist, or bad parameter');
      write('<',Chr($0D),Chr($0A));
  end;

  procedure __Transfer_file;

  begin
    write ('Load FileName:');
    readln (__Filename_Address_Range);
```

```
  writeln;   { one line feed }
  __File_Name := copy(__Filename_Address_Range,1,
                      pos(',',__Filename_Address_Range)-1);
  delete (__Filename_Address_Range,1,length(__File_Name)+1);
  __Memory_Address := __Filename_Address_Range;
  assign (__FileVar,__File_Name);
  if __Check_IO (__FileVar) then
    begin
      reset (__FileVar);
        __Size_File := filesize(__FileVar);
        converse_hex (__Size_File,__Memory_Range);
      __Character := 'L';
      __Transmit_Message;
      repeat
        __Receive_Message;
      until ord(__Character) = $0A;
      repeat
        __Receive_Message;
      until ord(__Character) = $3A;
      __Receive_Message;
      __Character := Chr($20); __Transmit_Message;
      delay(100); __Receive_Message;
      for i := 1 to 4 do
        begin
          __Character := copy(__Memory_Address,i,1);
          __Transmit_Message; delay(100); __Receive_Message;
        end;
      __Character := Chr($0D); __Transmit_Message;
      repeat
        __Receive_Message;
      until ord(__Character) = $3A;
      __Receive_Message;
      __Character := Chr($20); __Transmit_Message;
      delay(100); __Receive_Message;
      for i := 1 to 4 do
        begin
          __Character := copy(__Memory_Range,i,1);
          __Transmit_Message; delay(100); __Receive_Message;
        end;
      __Character := Chr($0D); __Transmit_Message;
      for __Data_Index := 1 to __Size_File do
        begin
          read (__FileVar,__Data);
          __Transmit;
        end;
      close (__FileVar);
    end { if __Check_IO (__FileVar) }
    else  writeln ('File not exist, or bad parameter');
end;

procedure __Receive_Message_Save;
begin
  repeat
  until port[$03FD] and $01 = $01 ;
    __Data := Port[$3F8];
end;
```

```
procedure  __Save_file;
var
  j,l,t : integer;
begin
  write ('Save Filename:');
  readln (__Filename_Address_Range);
  writeln;  { one line feed }
  __File_Name := copy(__Filename_Address_Range,1,
                      pos(',',__Filename_Address_Range)-1);
  delete (__Filename_Address_Range,1,length(__File_Name)+1);
  t := 0;
  repeat
    __Character := copy(__Filename_Address_Range,t+1,1);
    t := t+1;
  until __Character = ',';
  __Memory_Address := copy(__Filename_Address_Range,1,t-1);
  delete(__Filename_Address_Range,1,t);
  __Character := 'S';
  __Transmit_Message;
  repeat
    __Receive_Message;
  until ord(__Character) = $0A;
  repeat
    __Receive_Message;
  until ord(__Character) = $3A;
  __Receive_Message;
  __Character := Chr($20);
  __Transmit_Message; delay(100);
  __Receive_Message;
  j := length(__Memory_Address);
  for i := 1 to j do
    begin
      __Character := copy(__Memory_Address,i,1);
      __Transmit_Message;  delay(80);
      __Receive_Message;
    end;
  __Character := Chr($0D);
  __Transmit_Message;
  repeat
    __Receive_Message;
  until ord(__Character) = $3A;
  __Receive_Message;
  __Character := Chr($20); __Transmit_Message; delay(100);
  __Receive_Message;
  __Size_File_Dec:=length(__Filename_Address_Range);
  __Range_File:=0;
  for j := __Size_File_Dec  downto 1  do
    begin
      case j of
        1 : l:=1;
        2 : l:=16;
        3 : l:=256;
        4 : l:=4096;
      end;
      __Size_Each_Char := copy(__Filename_Address_Range,
                               __Size_File_Dec-j+1,1);
      __Size_Each_Char := upcase(__Size_Each_Char);
      __Size_Value := ord(__Size_Each_Char);
```

```
      if  __Size_Each_Char <= '9'
        then  __Size_Value:= __Size_Value-48
        else  __Size_Value:= __Size_Value-55;
       __Range_File:= __Range_File+(__Size_Value*1);
    end;
  for i := 1 to __Size_File_Dec do
    begin
      __Character := copy(__Filename_Address_Range,i,1);
      __Transmit_Message; delay(100);
      __Receive_Message;
    end;
  __Character := Chr($0D);__Transmit_Message;delay(5);
  __Range_File :=__Range_File+3;
  for __Data_Index := 1 to __Range_File do
    begin
      __Receive_Message_Save;
      __Data_Save[__Data_Index] := __Data;
    end;
  assign (__FileVar,__File_Name);
  rewrite (__FileVar);
  for __Data_Index := 4 to __Range_File do
    begin
      write(__FileVar,__Data_Save[__Data_Index]);
    end;
  close (__FileVar);
  writeln;
  __Character:=Chr($20);__Transmit_Message;
end;

procedure  __Set_Cursor(top,bottom:byte);
type
  reg   =  record
             case integer of
              1 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS : integer );
              2 : (AL,AH,BL,BH,CL,CH,DL,DH          : byte    );
           end;
var  __Register   : reg  ;
begin
  __Register.AX := $0100  ;
  __Register.CH := top    ;
  __Register.CL := bottom ;
  intr($10,__Register);
end;

procedure  __Check_Key_Pressed(var __Key_Pressed : boolean);
begin
    for i := 1 to 100 do
      if KeyPressed then
        begin
          read(KBD,__Character);
          __Key_Pressed := true
        end;
end;

procedure __Monitoring;
var
  __Number_Address_Mem : byte;
  __Memory_Address : string[100];
  __Buffer_Mem_Address   : array[1..20] of string[4];
```

```
  __Check_Num,__Char : char;
    i,j,k   : byte;
  __Number     : array[1..2] of byte;
  __Hex     : array[1..2] of string[1];
  __Num_Addr_Hex_String : string[2];
  __Key_Pressed    : boolean;
begin
  write('Monnitoring Address:');
  __Memory_Address := '';
  repeat
    read(KBD,__Char);write(__Char);
    __Char := upcase(__Char);
    if __Char <> #13 then
      __Memory_Address := concat(__Memory_Address,__Char);
  until __Char = #13;
  __Number_Address_Mem :=1; i:=1;
  repeat
    __Check_Num := copy(__Memory_Address,i,1);
    if __Check_Num = ',' then
      __Number_Address_Mem := __Number_Address_Mem+1;
    i:= i+1;
  until  __Check_Num ='.';
  for j:= 1 to __Number_Address_Mem do
    begin
      i:= 1;
      repeat
        __Check_Num := copy(__Memory_Address,i,1);
        i:=i+1;
      until __Check_Num IN ['.',','];
      __Buffer_Mem_Address[j] := copy(__Memory_Address,1,i-2);
      delete(__Memory_Address,1,i-1)
    end;
  __Number[1] :=0;__Number[2]:= 0;
  __Number[1] := __Number_Address_Mem mod 16;
  __Number[2] := __Number_Address_Mem div 16;
  if __Number[1] >= 10 then
     case __Number[1] of
       10 : __Hex[1]:='A';
       11 : __Hex[1]:='B';
       12 : __Hex[1]:='C';
       13 : __Hex[1]:='D';
       14 : __Hex[1]:='E';
       15 : __Hex[1]:='F';
     end
   else
    str(__Number[1],__Hex[1]);
   str(__Number[2],__Hex[2]);
   __Num_Addr_Hex_String :=concat(__Hex[2],__Hex[1]);
   __Character := 'X';
   __Transmit_Message;
   repeat
     __Receive_Message_mon;
   until  __Character = 'X';
   __Character := copy(__Num_Addr_Hex_String,1,1);
   __Transmit_Message;delay(80);
   __Character := copy(__Num_Addr_Hex_String,2,1);
   __Transmit_Message;delay(80);
   __Character := Chr($0D);
```

```
  __Transmit_Message;
  for j:= 1 to __Number_Address_Mem do
    begin
      i:= length(__Buffer_Mem_Address[j]);
      for k:= 1 to i do
        begin
          __Character := copy(__Buffer_Mem_Address[j],k,1);
          __Transmit_Message;delay(80);
      end; {for k }
      __Character := Chr($0D);
      __Transmit_Message;delay(80);
      repeat
        __Receive_Message_mon;
      until __Character ='Z';
    end;  { for j }
  delay(80);
  __Set_Cursor(7,6);   { cursor OFF }
  __ClearWin (Window5Fore,Window5Back);__GotoWin(1,1);
  __GotoWin(12,Window5Y2-Window5Y1+1);
  Textcolor(White+Blink);
  write('Run Monitoring');
  __GotoWin(1,1);
  Textcolor(Yellow);
  __Key_Pressed := false;
  repeat
    for j:= 1 to __Number_Address_Mem do
      begin
        i:= length(__Buffer_Mem_Address[j]); delay(1);
        case i of
          1: write(' Monitoring : ','000',__Buffer_Mem_Address[j],' ==> ');
          2: write(' Monitoring : ','00',__Buffer_Mem_Address[j],' ==> ');
          3: write(' Monitoring : ','0',__Buffer_Mem_Address[j],' ==> ');
          4: write(' Monitoring : ',__Buffer_Mem_Address[j],' ==> ');
        end; {case}
        repeat
          __Character := 'Z';
          __Transmit_Message;delay(50);
          __Receive_Message;
        until __Character = Chr($0A);
      end;
      __Check_Key_Pressed(__Key_Pressed);
      __GotoWin(1,1);
  until  (__Key_Pressed = true) and (__Character = Chr($0D));
  sound(500);delay(1000);nosound;
  Textcolor(Blue);
  __GotoWin(12,Window5Y2-Window5Y1+1);
  write('                ');
  Textcolor(Yellow);
  __GotoWin(1,1);
  __Set_Cursor(6,7); { cursor ON }
  for j := 1 to __Number_Address_Mem+1 do
     write('<',Chr($0D),Chr($0A));
  __Character := 'B';
  __Transmit_Message;
end;

procedure __Init;
begin { __Init }
```

```
if __Start then
  __ResetScn (-1);

if (_CurMode = 7) then Window1Back := Black;
Window1 := __MakeWin (Window1X1,Window1Y1,Window1X2,Window1Y2,12,13,
Window1Fore,Window1Back,Window1BorType,Window1BorFore,Window1BorBack);

if (_CurMode = 7) then Window2Back := Black;
Window2 := __MakeWin (Window2X1,Window2Y1,Window2X2,Window2Y2,12,13,
Window2Fore,Window2Back,Window2BorType,Window2BorFore,Window2BorBack);

if (_CurMode = 7) then Window3Back := Black;
Window3 := __MakeWin (Window3X1,Window3Y1,Window3X2,Window3Y2,12,13,
Window3Fore,Window3Back,Window3BorType,Window3BorFore,Window3BorBack);

if (_CurMode = 7) then Window4Back := Black;
Window4 := __MakeWin (Window4X1,Window4Y1,Window4X2,Window4Y2,12,13,
Window4Fore,Window4Back,Window4BorType,Window4BorFore,Window4BorBack);

if (_CurMode = 7) then Window5Back := Black;
Window5 := __MakeWin (Window5X1,Window5Y1,Window5X2,Window5Y2,12,13,
Window5Fore,Window5Back,Window5BorType,Window5BorFore,Window5BorBack);

if (_CurMode = 7) then Window6Back := Black;
Window6 := __MakeWin (Window6X1,Window6Y1,Window6X2,Window6Y2,12,13,
Window6Fore,Window6Back,Window6BorType,Window6BorFore,Window6BorBack);

if (_CurMode = 7) then Window7Back := Black;
Window7 := __MakeWin (Window7X1,Window7Y1,Window7X2,Window7Y2,12,13,
Window7Fore,Window7Back,Window7BorType,Window7BorFore,Window7BorBack);

if (_CurMode = 7) then Window8Back := Black;
Window8 := __MakeWin (Window8X1,Window8Y1,Window8X2,Window8Y2,12,13,
Window8Fore,Window8Back,Window8BorType,Window8BorFore,Window8BorBack);

Device := CRT;
DeviceName := 'CON:';
if __Start then
  begin
     __Check := __DispWin (Window8);
     Assign (DocumentFile,'TITLE.TXT');
     Reset (DocumentFile);
     __COffScn (true);
     while not EOF(DocumentFile) do
       begin
         Readln (DocumentFile,__Line);
         Writeln (__Line);
       end;
     Close (DocumentFile);
     __GotoWin (49,24);
     __COffScn (true);
     TextColor (LightMagenta+Blink);
     Ch := __PauseKey ('');
     __ClearWin (Window8Fore,Window8Back);__gotowin(1,1);
  end;
end { __Init };

procedure __Help;
```

```
var
  i : byte;
procedure __Return;
begin
    TextColor(LightRed);write(' ',Chr($1B),'-')
end;
begin
  __ResetScn(-1);
  TextBackGround(Black);
  TextColor(Brown);
  for i:= 1 to 23 do
    begin
      gotoXY(39,i);
      write(Chr($C7));
    end;
  for i:= 1 to 79 do
    begin
      gotoXY(i,23);write(Chr($CD));
      gotoXY(i,1);write(Chr($CD));
      gotoXY(i,25);write(Chr($CD));
    end;
  gotoXY(39,1);write(Chr($CB));
  gotoXY(39,23);write(Chr($CA));
  gotoXY(11,2);TextColor(LightCyan);write('Verify Memory');
  TextColor(LightRed);
  for i:= 11 to 23 do
    begin
      gotoXY(i,3);write(Chr($CD));
    end;
  gotoXY(2,5);TextColor(LightMagenta);write('M');
  TextColor(LightGreen);write('XXXX');
  __Return;
  TextColor(Yellow);write(' Display 8 Location');
  gotoXY(5,6);TextColor(LightCyan);write('=');
  TextColor(Yellow);write('     Display Next 8 Location');
  gotoXY(5,7);TextColor(White);write('-');
  TextColor(Yellow);write('     Display Last 8 Location');
  gotoXY(11,9);TextColor(LightRed);write('Modify Memory');
  TextColor(White);
  for i:= 11 to 23 do
    begin
      gotoXY(i,10);write(Chr($CD));
    end;
  gotoXY(2,12);TextColor(LightMagenta);write('M');
  TextColor(LightGreen);write('XXXX');
  TextColor(White+Blink);write(':');
  TextColor(Yellow);
  write('dd1 dd2');
  __Return;TextColor(LightBlue);write(' Fill Data dd1 dd2');
  gotoXY(19,13);write('to XXXX XXXX+1');
  gotoXY(2,14);TextColor(LightMagenta);write('M');
  TextColor(LightGreen);write('XXXX');
  TextColor(White+Blink);write('/');
  TextColor(LightCyan);write('YYYY ');
  TextColor(Cyan);write('ZZZZ');
  __Return;TextColor(LightBlue);write(' Move Block');
```

```
  gotoXY(19,15);write('XXXX ',Chr($1D),' YYYY to ZZZZ');
  gotoXY(2,16);TextColor(LightMagenta);write('F');
  TextColor(LightGreen);write('XXXX');
  TextColor(LightCyan);write(' YYYY ');
  TextColor(Yellow);write('dd');
  __Return;TextColor(LightBlue);write(' Fill Block Data dd');
  gotoXY(19,17);write('to XXXX ',Chr($1D),' YYYY');
  gotoXY(2,19);TextColor(LightGreen);
  write('Note : for Move & Fill Block');
  gotoXY(9,20);write('XXXX must be < = YYYY');
  gotoXY(42,2);TextColor(White);write('F2');TextColor(LightGray);
  write(' Load File');
  gotoXY(46,3);TextColor(LightRed);write('[Path:]');
  TextColor(Yellow);write('FileName,');TextColor(LightGreen);
  write('XXXX,');TextColor(White);write('YYYY');__Return;
  gotoXY(42,5);TextColor(White);write('F3');TextColor(LightBlue);
  write(' Save File');
  gotoXY(46,6);TextColor(LightRed);write('[Path:]');
  TextColor(Yellow);write('FileName,');TextColor(LightGreen);
  write('XXXX,');TextColor(White);write('YYYY');__Return;
  TextColor(LightCyan);
  gotoXY(46,8); write('XXXX     --> Location');
  gotoXY(46,9); write('YYYY     --> Range');
  gotoXY(42,10);TextColor(White);write('F4');TextColor(LightGray);
  write(' Load Text File');
  gotoXY(46,11);TextColor(LightRed);write('[Path:]');
  TextColor(Yellow);write('FileName');__Return;TextColor(LightGreen);
  write('  Speed');__Return;TextColor(White);
  gotoXY(46,12);write('Speed    --> High to Low ');
  gotoXY(46,13);write('Path     --> Drive A,B,C or D');
  gotoXY(46,14);write('FileName --> nnnnnnnn.sss');
  gotoXY(42,16);TextColor(White);write('F5');TextColor(LightGreen);
  write(' Monitoring');gotoXY(46,17);TextColor(LightRed);
  write('VVVV,');TextColor(LightMagenta);write('WWWW,');
  TextColor(LightBlue);write('XXXX,');
  TextColor(Brown);write('YYYY,,,,');TextColor(White);
  write('ZZZZ');TextColor(LightMagenta+Blink);write('.');__Return;
  gotoXY(45,19);TextColor(LightCyan);write('Display Data Continuous');
  gotoXY(45,20);TextColor(Yellow+Blink);write('Stop');
  TextColor(LightCyan);write(' by Return Key');
  gotoXY(13,24);TextColor(LightMagenta);
  write('VVVV WWWW XXXX YYYY ZZZZ dd dd1 dd2  ALL in Hexadecimal');
  __Init;
end;

procedure __Task1;
begin { __Task1 }
  __Check := __DispWin (Window1);
  __GotoWin (1,1);
end { __Task1 };

procedure __Task2;
begin { __Task2 }
  __Check := __DispWin (Window2);
  __GotoWin (1,1);
end { __Task2 };

procedure __Task3;
```

```
begin { __Task3 }
  __Check := __DispWin (Window3);
  __GotoWin (1,1);
end { __Task3 };

procedure __Task4;
begin { __Task4 }
  __Check := __DispWin (Window4);
  __GotoWin (1,1);
end { __Task4 };

procedure __Task5;
begin { __Task5 }
  __Check := __DispWin (Window5);
  __GotoWin (1,1);
end { __Task5 };

procedure __Task6;
begin { __Task6 }
  __Check := __DispWin (Window6);
  __GotoWin (1,1);
end { __Task6 };

procedure __Task7;
begin { __Task7 }
  __Check := __DispWin (Window7);
  __GotoWin (1,1);
end { __Task7 };

procedure MenuSelection;
begin { MenuSelection }
  __COffScn (true);
  Writeln;
  Writeln ('           Main Menu');
  Writeln ('           ---------');
  Writeln;
  Writeln ('         F1 : Main Menu');
  Writeln ('         F2 : Load File');
  Writeln ('         F3 : Save File');
  Writeln ('         F4 : Load Text file');
  Writeln ('         F5 : Monitoring');
  Writeln ('         F6 : __Help !');
  Writeln;
  Writeln ('         M  : Memory');
  Writeln ('         F  : Fill Memory');
  Writeln ('         R  : Reset Board');
  TextColor(LightMagenta);
  Writeln ('       Esc : Exit to DOS');
  TextColor (White);
  __GotoWin (24,Window1Y2-Window1Y1+1);
  Write ('Select Menu...');
end;

procedure Function_Exec (__Character:char);
begin
    case __Character of
    F1 : begin
           __Task1;
```

```
            MenuSelection;
          end;
      F2 : begin
            __Task2;
            __Transfer_file;
          end;
      F3 : begin
            __Task3;
            __Save_file;
          end;
      F4 : begin
            __Task4;
            __Load_Textfile;
          end;
      F5 : begin
            __Task5;
            __Monitoring;
          end;
      F6 : __Help;
    end;
end;

procedure __Character_Exec (__Character:char);
begin
   __Character := upcase(__Character);
   if (__Character IN ['M','F']) and  (__Memory_Mode_Flag)
       and (__Checker_Memory = 1)  then  __Task6;
   __Transmit_Message;
end;

procedure __Comunication;
begin
  repeat
  begin
     if NOT __Start then
       begin
         repeat
           __Receive_Message;
         until KeyPressed;
         read (KBD,__Character);
         if (__Character = #27) AND KeyPressed then
           begin
             Read (KBD,__Character);
             __Memory_Mode_Flag := false;
             __Checker_Memory := 0;
             Function_Exec (__Character);
           end
         else
           begin
             if __Character IN ['M','F','m','f']
               then begin
                 __Checker_Memory := __Checker_Memory+1;
                 __Memory_Mode_Flag := true;
               end;
             if (ord(__Character) IN [$2D,$3D]) AND (__Memory_Mode_Flag)
             then
                 __Character_Exec(__Character)
             else
               begin
```

```
            if ord(__Character) IN [$2D,$3D]
              then __Character := Chr($00);
            __Character_Exec(__Character);
          end;
        end;
      end
    else
      begin
       Function_Exec(__Character);
       __Memory_Mode_Flag := false;
       __Checker_Memory := 0;
      end;
    __Start := false
  end;
 until __Character = Esc;
 __Check := __ResWin;
end; { communicate }

begin { Main }
  __Start := true;
  __Init;
  __Init_interface;
  __Task7;
  write ('   Press F1 to Main Menu...');
  repeat
    repeat
    until KeyPressed;
   read(KBD,__Character);
    if (__Character = #27) and KeyPressed then
      read(KBD,__Character);
  until __Character IN [F1,Esc];
  if __Character = F1 then
     begin
       repeat
         __Comunication;
       until __Character = Esc;
     end;
  __ResetScn (-1);
end { Main }.
```

```
2500 A.D. Z80 CROSS ASSEMBLER   -   VERSION 3.00b
-------------------------------------------------

      INPUT  FILENAME :  PROJ_Z80.ASM
      OUTPUT FILENAME :  PROJ_Z80.OBJ


                ;*********************************************************
                ;*                                                       *
                ;*       PROGRAM MONITOR FOR MICROPROCESSOR              *
                ;*        DEVELOPMENT SYSTEM (Z-80) (MDS.)               *
                ;*   Programmer : Mr. Santidej Mansawasdi                *
                ;*              : Mr. Bancha   Watanasoponwong           *
                ;*   Advisor    : Mr. Viriya                             *
                ;*   Section    : Industial Computer                     *
                ;*   Department : Industial Instrumentation Technology   *
                ;*   Faculty    : Engineering                            *
                ;*        King Mongkut's Institue of Technology          *
                ;*        Ladkrabang   ----------  KMITL                 *
                ;*                                                       *
                ;*********************************************************

0000                    ORG   0000H
      03 00     PB255N00  EQU   03H
      07 00     PB255N01  EQU   07H
      0B 00     PB255N02  EQU   0BH
      00 00     LOWADDR   EQU   00H
      02 00     HIGHADDR  EQU   02H
      01 00     DATA      EQU   01H
      09 00     ENABLE    EQU   09H
      04 00     CONTROL   EQU   04H
      06 00     BUSRQAK   EQU   06H
      0D 00     URTCNT    EQU   0DH
      0C 00     URTDA     EQU   0CH
      A5 00     PWCODE    EQU   0A5H

                ;I/O port assignment : (PB255N00)
                ;
                ;port A (address 00H ) The  low byte  address bus
                ;                      of cpu z-80.
                ;       bit0 -- A0
                ;       bit1 -- A1
                ;       bit2 -- A2
                ;       bit3 -- A3
                ;       bit4 -- A4
                ;       bit5 -- A5
                ;       bit6 -- A6
                ;       bit7 -- A7
```

```
;port C (address 02H) The high byte address bus
;                     of cpu z-80.
;       bit0 -- A8
;       bit1 -- A9
;       bit2 -- A10
;       bit3 -- A11
;       bit4 -- A12
;       bit5 -- A13
;       bit6 -- A14
;       bit7 -- A15
;port B (address 01H) The data bus of cpu z-80.
;       bit0 -- D0
;       bit1 -- D1
;       bit2 -- D2
;       bit3 -- D3
;       bit4 -- D4
;       bit5 -- D5
;       bit6 -- D6
;       bit7 -- D7
;*********************************************************
;I/O port assignment : (P8255N01)
;
;port B (address 04H) The control bus of cpu z-80.
;       bit2 -- MEMRQ
;       bit3 -- WR
;       bit4 -- RD
;port C (address 06H) The control bus of cpu z-80.
;       bit0 -- BUSRQ
;       bit7 -- BUSAK
;*********************************************************
;I/O port assignment : (P8255N02)
;
;port B (address 09H) The enable buffer control.
;       bit0 -- Reset DEV board.
;       bit1 -- Enable low byte address buffer.
;       bit2 -- Enable high byte address buffer.
;       bit3 -- Enable control signal buffer.
;       bit4 -- Enable data buffer.
;       bit6 -- Control direction of data bus.
;*********************************************************
;I/O port assignment : (8251)
;
;       staus -- 0DH
;       data  -- 0CH
;
; ---reset---
; There are two cases that will genrate a RESET signal:
;      (i) power-up
```

```
                        ;     (ii) 'RESET' key pressed
                        ;   In both cases, the following actions will be taken:
                        ;      a) disable interrupt,set intrrupt mode to 0
                        ;         set I register to 00 and start execution
                        ;         at address 0000H ( by z-80 cpu itself).
                        ;      b) set user's SP to FEA0H
                        ;   Memory location POWERUP is used to distinguish
                        ;   power-up from RESET-key. (POWERUP) contains a
                        ;   random data when  power-up and contains PWCODE
                        ;   (0A5H) thereafter.

0000   01 00 03                LD BC,0300H     ; Power-up delay
0003   ED A9           RESTART: CPD
0005   EA 03 00                JP PE,RESTART
                        ;
                        ; Initial P8255NO2 to mode 0 with all port to output.
                        ; The control word is 80H
                        ;
0008   3E 80                   LD A,80H
000A   D3 0B                   OUT (P8255NO2),A
                        ;
                        ;  When the control word is sent out to 8255, all output
                        ;  ports are cleared to 0. It is necessary to disable
                        ;  all buffer and no reset signal by sending 0FFH to
                        ;  port B of P8255NO2.
                        ;
000C   3E FF                   LD A,0FFH       ; Disable all buffer.
000E   D3 09                   OUT (ENABLE),A  ; and no reset signal to dev.
0010   CD C0 00                CALL INITRS232
0013   31 9A FF                LD SP,SYSSTK    ; Initial system stack.
                        ;
                        ;  If the content of location POWERUP is not equal to
                        ;  PWCODE, call subroutine INI. Continue otherwise.

0016   3A C0 FF                LD A,(POWERUP)
0019   FE A5                   CP PWCODE
001B   C4 C9 00                CALL NZ,INI     ; Cold start

                        ;*********************************************************
                        ; Scan the keyboard and display on IBM. When a key is
                        ; is deteced, IBM  will ASCII code by serial interface.
                        ; This routine will be take proper action according to
                        ; the key pressed.

001E   31 9A FF         MAIN:  LD SP,SYSSTK    ; Initial system stack.
0021   CD 52 02                CALL SCAN       ; Scan ASCII code send from IBM
                                               ; keyborad and echo ASCII code
                                               ; to IBM monitor display.
```

```
0024  F5                    PUSH AF
0025  CD 14 07              CALL CLRBF
0028  F1                    POP AF
0029  CD 59 00              CALL KEYEXEC    ; Input key dispatch routine.
002C  CD 62 02              CALL DISPLAY    ; Send ASCII code in INPBF
                                            ; to IBM monitor display until
                                            ; charactor = ODH (return key)
                                            ; , send OAH (line feed)
                                            ; thereafter.
002F  18 ED                 JR MAIN         ; Back to MAIN, get more keys
                                            ; and execute them.

                    ;*******************************************************
                    ;
                    ; SOFTWARE ESC command -- reenter monitor.
                    ; Executed by depressing the ESC key.
                    ; The ESC command escapes from thje existing command
                    ; and returns to monitor.
                    ; ESC is operative only in the commands that sample
                    ; the key board.
                    ; MDS will responds to ESC by displaying the MDS
                    ; monitor prompt <.

0031  31 9A FF      ESCAPE: LD SP,SYSSTK
0034  CD 14 07              CALL CLRBF
0037  CD AA 07              CALL CR3
003A  18 E2                 JR MAIN
                    ;
                    ;*******************************************************
                    ;
                    ; Executed when equal sign (=) or minus sign (-)
                    ; key is pressed.

003C  3A FD FF      FOR:    LD A,(TYPEFG)
003F  FE 10                 CP 10H
0041  CA 96 06              JP Z,MFOR       ; Display next eight memory
                                            ; contents.
0044  18 08                 JR IGNORE
0046  3A FD FF      BACK:   LD A,(TYPEFG)
0049  FE 10                 CP 10H
004B  CA AF 06              JP Z,MBACK      ; Display last eight memory
                                            ; contents.
004E  CD B2 07      IGNORE: CALL CLEAR
0051  3E 3C                 LD A,'<'
0053  CD 9F 07              CALL CHRWR
0056  C3 AA 07              JP CR3
                    ;
                    ;*******************************************************
```

```
                        ;
                        ; Input key dispatch routine.
                        ; This routine uses the key codes returned by subroutine
                        ; SCAN, which is one byte ( ASCII code) stored in A.
                        ;
0059   FE 3D            KEYEXEC: CP 3DH
005B   28 DF                    JR Z,FOR        ; Equal sign.
005D   FE 2D                    CP 2DH
005F   28 E5                    JR Z,BACK       ; Minus sign.
0061   FE 4D                    CP 'M'
0063   CA C7 04                 JP Z,MEMEXEC    ; Memory display and modify.
0066   FE 58                    CP 'X'
0068   CA 9A 03                 JP Z,MONITOR    ; Monitoring routine
006B   FE 53                    CP 'S'
006D   CA DD 02                 JP Z,SAVE       ; Save data into File
0070   FE 46                    CP 'F'
0072   CA BD 06                 JP Z,FILLDA     ; Fill data into memory
0075   FE 4C                    CP 'L'
0077   CA 7D 02                 JP Z,LOAD       ; Load file into memory
007A   FE 52                    CP 'R'
007C   CA B7 00                 JP Z,RESETDEV   ; Reset development board.
007F   FE 0D                    CP 0DH
0081   CA AA 07                 JP Z,CR3
0084   C3 1E 00                 JP MAIN
                        ;
                        ;*******************************************************

                        RESETDEV:               ; Generate reset signal
0087   3E FE                    LD A,0FEH
0089   D3 09                    OUT (ENABLE),A
008B   CD 71 04                 CALL DELAY
008E   CD 71 04                 CALL DELAY
0091   3E FF                    LD A,0FFH
0093   D3 09                    OUT (ENABLE),A
0095   3E 0D                    LD A,0DH
0097   CD 63 04                 CALL TRANSMIT
009A   21 A3 00                 LD  HL,RESETMSG
009D   CD 3F 04                 CALL TRANSMSG
00A0   C3 4E 00                 JP IGNORE

00A3   3C 52 3E 3D      RESETMSG: DEFB '<R>= Reset Development Board',0DH
00A7   20 52 65 73
00AB   65 74 20 44
00AF   65 76 65 6C
00B3   6F 70 6D 65
00B7   6E 74 20 42
00BB   6F 61 72 64
00BF   0D
```

```
                        ;*******************************************************
                        ; Function : Initial 8251 Universal Synchronous
                        ;           Asynchronous Reciever Transmitter
                        ;           ( USART ).

00C0                    INITRS232:
00C0   3E DE                    LD A,0DEH
00C2   D3 0D                    OUT (URTCNT),A
00C4   3E 37                    LD A,37H
00C6   D3 0D                    OUT (URTCNT),A
00C8   C9                       RET

                        ;*******************************************************
                        ;
                        ; Cold boot initialization.

00C9                    INI:
00C9   CD EC 00                 CALL STARTPG
00CC   21 FF FF                 LD HL,0FFFFH
00CF   01 00 20         RAMT1:  LD BC,2000H
00D2   CD 1D 07         RAMT2:  CALL RAMCHK
00D5   28 03                    JR Z,TNEXT
00D7   C3 E5 00                 JP RAMFAULT
00DA   ED A9            TNEXT:  CPD
00DC   EA D2 00                 JP PE,RAMT2
00DF   3E A5            INIS:   LD A,PWCODE
00E1   32 C0 FF                 LD (POWERUP),A
00E4   C9                       RET

                        ;*******************************************************
                        ;
                        ; Display RAM fault massage , system halt
                        ; thereafter.

00E5                    RAMFAULT:
00E5   21 12 02                 LD HL,RFLTMSG
00E8   CD 3F 04                 CALL TRANSMSG
00EB   76                       HALT

                        ;*******************************************************
                        ;
                        ; Display title massage.

00EC   CD 4B 04         STARTPG: CALL ENDMSG
00EF   21 20 01                 LD HL,TITLEMSG
00F2   CD 3F 04                 CALL TRANSMSG
00F5   21 3E 01                 LD HL,TITLEMSG1
```

```
00F8   CD 3F 04                  CALL TRANSMSG
00FB   21 5D 01                  LD HL,TITLEMSG2
00FE   CD 3F 04                  CALL TRANSMSG
0101   21 6B 01                  LD HL,TITLEMSG3
0104   CD 3F 04                  CALL TRANSMSG
0107   21 B9 01                  LD HL,TITLEMSG4
010A   CD 3F 04                  CALL TRANSMSG
010D   21 AC 01                  LD HL,TITLEMSG5
0110   CD 3F 04                  CALL TRANSMSG
0113   21 CD 01                  LD HL,TITLEMSG6
0116   CD 3F 04                  CALL TRANSMSG
0119   21 F2 01                  LD HL,TITLEMSG7
011C   CD 3F 04                  CALL TRANSMSG
011F   C9                        RET

0120   20 20 20 20       TITLEMSG:  DEFB '      Z80 Development System',0DH
0124   20 20 20 5A
0128   38 30 20 44
012C   65 76 65 6C
0130   6F 70 6D 65
0134   6E 74 20 53
0138   79 73 74 65
013C   6D 0D
013E   20 44 65 73    TITLEMSG1: DEFB ' Designed By INS.Research Team',0DH
0142   69 67 6E 65
0146   64 20 42 79
014A   20 49 4E 53
014E   2E 52 65 73
0152   65 61 72 63
0156   68 20 54 65
015A   61 6D 0D
015D   20 50 72 6F       TITLEMSG2: DEFB ' Programmer :',0DH
0161   67 72 61 6D
0165   6D 65 72 20
0169   3A 0D
016B   20 20 20 20       TITLEMSG3: DEFB '      Mr. Santidej Mansawasdi',0DH
016F   20 20 4D 72
0173   2E 20 53 61
0177   6E 74 69 64
017B   65 6A 20 4D
017F   61 6E 73 61
0183   77 61 73 64
0187   69 0D
0189   20 20 20 20       TITLEMSG4: DEFB '      Mr. Bancha   Watanasoponwong',0DH
018D   20 20 4D 72
0191   2E 20 42 61
0195   6E 63 68 61
0199   20 20 20 57
```

```
019D   61 74 61 6E
01A1   61 73 6F 70
01A5   6F 6E 77 6F
01A9   6E 67 0D
01AC   20 41 64 76      TITLEMSG5: DEFB ' Advisor : Mr. Viriya Kongratana',0DH
01B0   69 73 6F 72
01B4   20 3A 20 4D
01B8   72 2E 20 56
01BC   69 72 69 79
01C0   61 20 4B 6F
01C4   6E 67 72 61
01C8   74 61 6E 61
01CC   0D
01CD   20 44 65 70      TITLEMSG6: DEFB ' Dept. of Instrumentation Technology'
01D1   74 2E 20 6F
01D5   66 20 49 6E
01D9   73 74 72 75
01DD   6D 65 6E 74
01E1   61 74 69 6F
01E5   6E 20 54 65
01E9   63 68 6E 6F
01ED   6C 6F 67 79
01F1   0D                       DEFB 0DH
01F2   20 20 20 46      TITLEMSG7: DEFB '   Faculty of Engineering KMITL',0DH
01F6   61 63 75 6C
01FA   74 79 20 6F
01FE   66 20 45 6E
0202   67 69 6E 65
0206   65 72 69 6E
020A   67 20 4B 4D
020E   49 54 4C 0D
0212   53 79 73 74      RFLTMSG:   DEFB 'System Failure HALT ! check Ram '
0216   65 6D 20 46
021A   61 69 6C 75
021E   72 65 20 48
0222   41 4C 54 20
0226   21 20 63 68
022A   65 63 6B 20
022E   52 61 6D 20
0232   20 20 20 20                DEFB '     Location 0E000H  All      ',0DH
0236   20 4C 6F 63
023A   61 74 69 6F
023E   6E 20 30 45
0242   30 30 30 48
0246   20 20 41 6C
024A   6C 20 20 20
024E   20 20 20 0D
```

```
                        ;*******************************************************
                        ;
                        ; Function : Scan the ASCII code from the keyboard IBM
                        ;           and echo it to IBM monitor display. If the
                        ;           some key is already pressed when this
                        ;           routine starts execution return when next
                        ;           key is entered.
                        ; Input:    None
                        ; Output:   A -- ASCII code send from IBM.
                        ; Reg affected: AF HL.
                        ; Call:     TRANSMIT

0252   37               SCAN:    SCF
0253   DB 0D                     IN A,(URTCNT)
0255   CB 4F                     BIT 1,A
0257   CA 52 02                  JP Z,SCAN        ;R
025A   DB 0C                     IN A,(URTDA)     ; Get charactor from IBM.
025C   CD 63 04                  CALL TRANSMIT    ; Echo this charactor back
                                                   ; to IBM monitor display.
025F   37                        SCF
0260   3F                        CCF
0261   C9                        RET

                        ;*******************************************************
                        ;
                        ; Sent The ASCII code in the input buffer (INPBF)
                        ; to the IBM monitor display until character is
                        ; a carriage return code (0DH),sent line feed code
                        ; (0AH) thereafter.

0262   3E 0D            DISPLAY: LD A,0DH
0264   CD 63 04                  CALL TRANSMIT
0267   21 C6 FF                  LD HL,INPBF
026A   CD 3F 04                  CALL TRANSMSG
026D   C9                        RET

                        ;*******************************************************
                        ;
                        ; Function : Change the ASCII code 1,2,3 or 4 character
                        ;           to be the hexadecimal 4 digits. And then
                        ;           store it in the HL register.
                        ; Input:  None
                        ; Output: HL <-- hexadecimal 4 digits.
                        ; Reg affected: AF DE HL
                        ; Call: ECHOCH GET CHKHE2 LOADAGN.

026E   CD 31 07         ADDRLD:  CALL ECHOCH
0271   CD F7 06         LOADAGN: CALL GET
```

```
0274   21 CA FF              LD HL,INPBF+4
0277   CD 6F 07              CALL CHKHE2
027A   38 F5                 JR C,LOADAGN
027C   C9                    RET

                      ;*******************************************************
                      ;
                      ; Load a file from the disk drive into the
                      ; specific memory address.

027D   CD D7 02       LOAD:   CALL SETTYPE
0280   3E 0D                  LD A,0DH
0282   CD 63 04                CALL TRANSMIT
0285   21 7E 04                LD HL,LDMSG
0288   CD 3F 04                CALL TRANSMSG
028B   21 B7 04                LD HL,LDMSG1
028E   CD 56 04                CALL TRANSMSG1
0291   CD 6E 02                CALL ADDRLD
0294   22 BC FF                LD (START),HL
0297   CD 4B 04                CALL ENDMSG
029A   21 A0 04                LD HL,LDMSG2
029D   CD 56 04                CALL TRANSMSG1
02A0   CD 14 07                CALL CLRBF
02A3   CD 6E 02                CALL ADDRLD
02A6   22 BE FF                LD (RANGE),HL
02A9   CD 4B 04                CALL ENDMSG
02AC   2A BC FF                LD HL,(START)
02AF   ED 4B BE FF             LD BC,(RANGE)
02B3   CD FE 04                CALL INITDO
02B6   DB 0D          LOAD1:   IN A,(URTCNT)
02B8   CB 4F                   BIT 1,A
02BA   28 FA                   JR Z,LOAD1
02BC   DB 0C                   IN A,(URTDA)
02BE   CD BB 05                CALL WRITEDATA
02C1   23                      INC HL
02C2   0B                      DEC BC
02C3   78                      LD A,B
02C4   B1                      OR C
02C5   C2 B6 02                JP NZ,LOAD1     ; Load file OK !.
02C8   CD 15 05                CALL DISABLE
02CB                  LOADRDY:
02CB   CD 4B 04                CALL ENDMSG
02CE   21 B9 04                LD HL,LDRDYMSG
02D1   CD 3F 04                CALL TRANSMSG
02D4   C3 4E 00                JP IGNORE

                      ;*******************************************************
                      ;
```

```
                        ; Set type flag not equal to memory mode.

02D7  3E A5             SETTYPE: LD A,0A5H
02D9  32 FD FF                   LD (TYPEFG),A
02DC  C9                         RET

                        ;*******************************************************
                        ;
                        ; Save a file from the specific memory address
                        ; to the diskette.

02DD  CD D7 02          SAVE:    CALL SETTYPE
02E0  3E 0D                      LD A,0DH
02E2  CD 63 04                   CALL TRANSMIT
02E5  21 52 03                   LD HL,SVMSG
02E8  CD 3F 04                   CALL TRANSMSG
02EB  21 5B 03                   LD HL,SVMSG1
02EE  CD 56 04                   CALL TRANSMSG1
02F1  CD 6E 02                   CALL ADDRLD
02F4  22 BC FF                   LD (START),HL
02F7  CD 4B 04                   CALL ENDMSG
02FA  21 74 03                   LD HL,SVMSG2
02FD  CD 56 04                   CALL TRANSMSG1
0300  CD 14 07                   CALL CLRBF
0303  CD 6E 02                   CALL ADDRLD
0306  22 BE FF                   LD (RANGE),HL
0309  CD 4B 04                   CALL ENDMSG
030C  2A BC FF                   LD HL,(START)
030F  ED 4B BE FF                LD BC,(RANGE)
0313  CD E7 04                   CALL  INITDI
0316                    SAVE1:
0316  CD A8 05                   CALL READDATA
0319  F5                        PUSH AF
031A  DB 0D             SAVE2:   IN  A,(URTCNT)
031C  CB 47                      BIT  0,A
031E  28 FA                      JR  Z,SAVE2
0320  F1                        POP AF
0321  D3 0C                      OUT (URTDA),A
0323  CD 71 04                   CALL DELAY
0326  23                        INC HL
0327  0B                        DEC BC
0328  78                        LD  A,B
0329  B1                        OR  C
032A  C2 16 03                   JP  NZ,SAVE1    ; Save file OK !.
032D  CD 15 05                   CALL DISABLE
0330  CD 52 02          TESTIN:  CALL SCAN
0333  FE 20                      CP ' '
0335  20 F9                      JR NZ,TESTIN
```

```
0337                    SAVEREDY:
0337   CD 4B 04                 CALL ENDMSG
033A   21 43 03                 LD HL,SVRDYMSG
033D   CD 3F 04                 CALL TRANSMSG
0340   C3 4E 00                 JP IGNORE

0343   54 52 41 4E      SVRDYMSG:    DEFB 'TRANSMIT READY',0DH
0347   53 4D 49 54
034B   20 52 45 41
034F   44 59 0D
0352   4F 4B 20 52      SVMSG:       DEFB 'OK READY',0DH
0356   45 41 44 59
035A   0D
035B   53 54 41 52      SVMSG1:      DEFB 'START ADDRESS (HEX)   : ',0DH
035F   54 20 41 44
0363   44 52 45 53
0367   53 20 28 48
036B   45 58 29 20
036F   20 20 3A 20
0373   0D
0374   52 41 4E 47      SVMSG2:      DEFB 'RANGE  (HEX)          : ',0DH
0378   45 20 20 28
037C   48 45 58 29
0380   20 20 20 20
0384   20 20 20 20
0388   20 20 3A 20
038C   0D

038D   CD 14 07          ADDRMON: CALL CLRBF
0390   CD 31 07                  CALL ECHOCH
0393   CD F7 06                  CALL GET
0396   CD 6C 07                  CALL CHKHEX
0399   C9                        RET

                        ;*****************************************************
                        ;
                        ; Display contents of group specific memory
                        ; continuous.

039A   CD D7 02          MONITOR: CALL SETTYPE
039D   21 79 FF                   LD HL,PTR
03A0   22 76 FF                   LD (DPTR),HL
03A3   CD 8D 03                   CALL ADDRMON
03A6   32 75 FF                   LD (N),A
03A9   47                         LD B,A
03AA   C5                GETADDR: PUSH BC
03AB   CD 14 07                   CALL CLRBF
03AE   CD 8D 03                   CALL ADDRMON
```

```
03B1   EB                        EX DE,HL
03B2   2A 76 FF                  LD HL,(OPTR)
03B5   73                        LD (HL),E
03B6   23                        INC HL
03B7   72                        LD (HL),D
03B8   23                        INC HL
03B9   22 76 FF                  LD (OPTR),HL
03BC   3E 5A                     LD A,'Z'
03BE   CD 63 04                  CALL TRANSMIT   ;Transait Character sync.
03C1   C1                        POP BC
03C2   10 E6                     DJNZ GETADDR
03C4   06 0A                     LD B,10
03C6   3E 5A              WAIT:  LD  A,'Z'
03C8   CD 63 04                  CALL TRANSMIT   ;TRANSMIT
03CB   10 F9                     DJNZ WAIT
03CD                      MONITOR1:
03CD   3A 75 FF                  LD A,(N)
03D0   47                        LD B,A
03D1   DD 21 79 FF               LD IX,PTR
03D5                      MONITOR2:
03D5   C5                        PUSH BC
03D6   DD 6E 00                  LD L,(IX+0)
03D9   DD 66 01                  LD H,(IX+1)
03DC   CD E7 04                  CALL INITDI
03DF   CD A8 05                  CALL READDATA
03E2   F5                        PUSH AF
03E3   CD 15 05                  CALL DISABLE
03E6   F1                        POP  AF
03E7   DD 23                     INC IX
03E9   DD 23                     INC IX
03EB   DD E5                     PUSH IX
03ED   21 C6 FF                  LD  HL,INPBF
03F0   22 F9 FF                  LD  (OUTPTR),HL
03F3   CD 0A 08                  CALL HEX2
03F6   3E 0D                     LD A,0DH
03F8   CD 9F 07                  CALL CHRWR
03FB   21 C6 FF                  LD HL,INPBF
03FE   CD 08 04                  CALL TMONITOR
0401   DD E1                     POP IX
0403   C1                        POP BC
0404   10 CF                     DJNZ MONITOR2
0406   18 C5                     JR  MONITOR1
0408                      TMONITOR:
0408   7E                        LD A,(HL)
0409   FE 0D                     CP 0DH
040B   CA 14 04                  JP Z,ENDMASSAGE
040E   CD 1F 04                  CALL TX_DATA
0411   23                        INC HL
```

```
0412   18 F4                    JR TMONITOR
0414                         ENDMASSAGE:
0414   3E 0D                    LD A,0DH
0416   CD 1F 04                 CALL TX_DATA
0419   3E 0A                    LD A,0AH
041B   CD 1F 04                 CALL TX_DATA
041E   C9                       RET
041F                         TX_DATA:
041F   F5                       PUSH AF
0420   DB 0D                 TX1:    IN A,(URTCNT)
0422   CB 4F                    BIT 1,A
0424   CA 20 04                 JP Z,TX1
0427   DB 0C                    IN A,(URTDA)
0429   FE 42                    CP 'B'
042B   CA 1E 00                 JP Z,MAIN
042E   FE 5A                    CP 'Z'
0430   20 EE                    JR NZ,TX1
0432   DB 0D                 TX2:    IN A,(URTCNT)
0434   CB 47                    BIT 0,A
0436   28 FA                    JR Z,TX2
0438   F1                       POP AF
0439   D3 0C                    OUT (URTDA),A
043B   CD 71 04                 CALL DELAY
043E   C9                       RET

                            ;*********************************************************
                            ;
                            ; Send the ASCII code in input buffer (INPBF)
                            ; to serial interface until last charactor
                            ; equal to carriage return (0DH) , send line
                            ; feed charactor (0AH) thereafter.

043F   7E                   TRANSMSG:LD A,(HL)
0440   FE 0D                    CP 0DH
0442   CA 4B 04                 JP Z,ENDMSG
0445   CD 63 04                 CALL TRANSMIT
0448   23                       INC HL
0449   18 F4                    JR TRANSMSG
044B   3E 0D                ENDMSG:  LD A,0DH
044D   CD 63 04                 CALL TRANSMIT
0450   3E 0A                    LD A,0AH
0452   CD 63 04                 CALL TRANSMIT
0455   C9                       RET

                            ;*********************************************************

0456                        TRANSMSG1:
0456   7E                       LD A,(HL)
```

```
0457  FE 0D                     CP 0DH
0459  CA 62 04                  JP Z,ENDMSG1
045C  CD 63 04                  CALL TRANSMIT
045F  23                        INC HL
0460  18 F4                    JR TRANSMSG1
0462  C9                 ENDMSG1: RET

                         ;*****************************************************
                         ;
                         ; Send data 1 byte to the serial port.

0463                     TRANSMIT:
0463  F5                        PUSH AF
0464                     TRANSMIT1
0464  DB 0D                     IN A,(URTCNT)
0466  CB 47                     BIT 0,A
0468  28 FA                     JR Z,TRANSMIT1
046A  F1                        POP AF
046B  D3 0C                     OUT (URTDA),A
046D  CD 71 04                  CALL DELAY
0470  C9                       RET
0471  C5                 DELAY:  PUSH BC
0472  F5                        PUSH AF
0473  01 00 02                  LD BC,200H
0476  0B                 DELAY1:  DEC BC
0477  78                        LD A,B
0478  B1                        OR C
0479  20 FB                     JR NZ,DELAY1
047B  F1                        POP AF
047C  C1                        POP BC
047D  C9                        RET

047E  4F 4B 20 52          LDMSG:    DEFB 'OK READY',0DH
0482  45 41 44 59
0486  0D
0487  53 54 41 52          LDMSG1:   DEFB 'START ADDRESS (HEX)  :  ',0DH
048B  54 20 41 44
048F  44 52 45 53
0493  53 20 28 48
0497  45 58 29 20
049B  20 3A 20 20
049F  0D
04A0  52 41 4E 47          LDMSG2:   DEFB 'RANGE  (HEX)        :  ',0DH
04A4  45 20 20 28
04A8  48 45 58 29
04AC  20 20 20 20
04B0  20 20 20 20
04B4  20 3A 20 20
```

```
04B8   0D
04B9   52 45 43 45      LDRDYMSG: DEFB 'RECEIVE READY',0DH
04BD   49 56 45 20
04C1   52 45 41 44
04C5   59 0D

                        ;*******************************************************
                        ;
                        ; Exeuted when 'M' key is pressed.
                        ;  Enter the hexadecimal address of the first of the
                        ; four memory locations to be displayed.
                        ;    (1) Type <CR> -- Display specified memory contents.
                        ;    (2) Type :    -- Alter memory contents.
                        ;    (3) TYPE /    -- Move data block from one area to
                        ;                     another.

04C7   CD 1C 05         MEMEXEC: CALL MEMEX2
04CA   3E 10                     LD A,10H
04CC   32 FD FF                  LD (TYPEFG),A   ; Set memory type.
04CF   1A                        LD A,(DE)
04D0   FE 3A                     CP 3AH          ; :
04D2   CA 37 05                  JP Z,MMODFY
04D5   FE 2F                     CP 2FH          ; /
04D7   CA 0A 06                  JP  Z,MMOVE
04DA   CD 2E 05                  CALL MEMEX3     ; Display specified memory
                                                 ; contents.
04DD   CD FC 07                  CALL HEXX
04E0   CD 7E 06         P101:    CALL MEM3
04E3   CD AA 07                  CALL CR3
04E6   C9                        RET

                        ;*******************************************************
                        ;
                        ; Initial all bus in order to  get data from
                        ; DEV. to MDS.

04E7   3E B2            INITDI:  LD  A,082H
04E9   D3 03                     OUT (P8255ND0),A
04EB   3E 81                     LD  A,081H
04ED   D3 07                     OUT (P8255ND1),A
04EF   DB 06            CHECKK:  IN  A,(BUSRQAK)
04F1   CB 47                     BIT 0,A
04F3   20 FA                     JR  NZ,CHECKK
04F5   3E FF                     LD  A,0FFH
04F7   D3 04                     OUT (CONTROL),A
04F9   3E 41                     LD  A,041H
04FB   D3 09                     OUT (ENABLE),A
04FD   C9                        RET
```

```
                        ;*******************************************************
                        ;
                        ; Initial all bus in order to send data from
                        ; MDS to DEV.

04FE   3E B0            INITDO:  LD  A,0B0H
0500   D3 03                    OUT (PB255N00),A
0502   3E B1                    LD  A,0B1H
0504   D3 07                    OUT (PB255N01),A
0506   DB 06            CHECK:  IN  A,(BUSRQAK)
0508   CB 47                    BIT 0,A
050A   20 FA                    JR  NZ,CHECK
050C   3E FF                    LD  A,0FFH
050E   D3 04                    OUT (CONTROL),A
0510   3E 01                    LD  A,01H
0512   D3 09                    OUT (ENABLE),A
0514   C9                       RET

                        ;*******************************************************
                        ;
                        ; Disable all bus.

0515   3E FF            DISABLE: LD  A,0FFH
0517   D3 09                    OUT (ENABLE),A  ;DISABLE BUFFER
0519   D3 06                    OUT (BUSRQAK),A ;DISABLE CTRL SIGNAL BUSRQ
051B   C9                       RET

                        ;*******************************************************
                        ;
                        ; The input data must be hexadecimal values. MDS will
                        ; ignore this command if there exists at least one digit
                        ; which is not a hexadecimal value. The user can use
                        ; the BACESPACE (<--) key to correct the data.

051C   CD 31 07         MEMEX2:  CALL ECHODH     ; Echo the input character and
                                                 ; prompt.
051F   CD F7 06         MEMEX1:  CALL GET        ; Get a string of characters
                                                 ; and end the input with <CR>.
0522   CD 25 07                  CALL CHKINP     ; Check hexadecimal values.
0525   38 F8                    JR   C,MEMEX1   ; Jump to MEMEX1 if the input
                                                 ; data is illegal.
0527   CD 6C 07                  CALL CHKHEX     ; Get the hexadecimal address
                                                 ; of the first of four memory
                                                 ; locations to be displayed.
052A   22 C4 FF                  LD (MADDR),HL
052D   C9                        RET
```

```
                        MEMEX3:                 ; Reset the counters of INPUT
                                                ; BUFFER.
052E   E5                       PUSH HL
052F   21 CA FF                 LD HL,INPBF+4
0532   22 F9 FF                 LD (OUTPTR),HL
0535   E1                       POP HL
0536   C9                       RET

                        ;*******************************************************
                        ; If you want to change the values in these location,
                        ; just type a colon and the values separated gy spaces.
                        ; The final command look like this:
                        ;   <M>=<start>:<data1> <data2><CR>

0537   E5               MMODFY: PUSH HL
0538   CD CE 05                 CALL RAMDCHK    ; Check this location,is't RAM ?
053B   CA 56 05                 JP Z,RAMFAULT1
053E   E1                       POP HL
053F   CD FE 04                 CALL INITDO
0542   E5               MMLOOP: PUSH HL
0543   CD 72 07                 CALL GETHL      ; Get data.
0546   E1                       POP HL
0547   CD BB 05                 CALL WRITEDATA
054A   CA 50 05                 JP Z,COMPLETE
054D   23                       INC HL
054E   1B F2                    JR MMLOOP
0550                    COMPLETE:
0550   CD 15 05                 CALL DISABLE
0553   C3 AA 07                 JP CR3

                        ;*******************************************************

0556                    RAMFAULT1:
0556   CD 4B 04                 CALL ENDMSG     ; One line feed.
0559   21 68 05                 LD HL,DOCUMENT1
055C   CD 3F 04                 CALL TRANSMSG
055F   21 86 05                 LD HL,DOCUMENT2
0562   CD 3F 04                 CALL TRANSMSG
0565   C3 4E 00                 JP IGNORE

0568   20 20 20 20      DOCUMENT1: DEFB '       This Location Non RAM',0DH
056C   20 20 20 20
0570   54 68 69 73
0574   20 4C 6F 63
0578   61 74 69 6F
057C   6E 20 4E 6F
0580   6E 20 52 41
0584   4D 0D
```

```
0586   20 20 4F 72      DOCUMENT2: DEFB '  Or Board Failure ! Check  Again',0DH
058A   20 42 6F 61
058E   72 64 20 46
0592   61 69 6C 75
0596   72 65 20 21
059A   20 43 6B 65
059E   63 6B 20 20
05A2   41 67 61 69
05A6   6E 0D

                        ;*********************************************************
                        ;
                        ; Function:   Get data 1 byte from DEV.
                        ; Input:      HL -- Point to specified memory.
                        ; Output:     A <-- data 1 byte
                        ; Reg affected: AF.
                        ; Call :      None.

05A8                    READDATA:
05A8   7D                       LD A,L
05A9   D3 00                    OUT (LOWADDR),A
05AB   7C                       LD A,H
05AC   D3 02                    OUT (HIGHADDR),A
05AE   3E EB                    LD A,0EBH
05B0   D3 04                    OUT (CONTROL),A
05B2   DB 01                    IN A,(DATA)
05B4   F5                       PUSH AF
05B5   3E FF                    LD A,0FFH
05B7   D3 04                    OUT (CONTROL),A
05B9   F1                       POP AF
05BA   C9                       RET

                        ;*********************************************************
                        ;
                        ; Function:   Send data 1 byte to DEV.
                        ; Input:      A -- data to be send
                        ;             HL -- Point to specified memory.
                        ; Output:     None.
                        ; Reg affected: AF.
                        ; Call :      None.

05BB                    WRITEDATA:
05BB   F5                       PUSH AF
05BC   7D                       LD A,L
05BD   D3 00                    OUT (LOWADDR),A
05BF   7C                       LD A,H
05C0   D3 02                    OUT (HIGHADDR),A
05C2   3E E3                    LD A,0F3H
```

```
05C4   D3 04                    OUT (CONTROL),A
05C6   F1                       POP AF
05C7   D3 01                    OUT (DATA),A
05C9   3E FF                    LD A,0FFH
05CB   D3 04                    OUT (CONTROL),A
05CD   C9                       RET

                        ;*****************************************************
                        ;
                        ; Function: The same as RAMCHK, but check DEV. board.
                        ; Input:    HL -- Point to specified address location
                        ;           on DEV. board.
                        ; Output:   Zero flag -- 0, ROM or nonexistant;
                        ;                        1, RAM.
                        ; Destroyed reg. : AF.
                        ; Call: INITDI INITDO READDATA WRITEDATA DISABLE .

05CE   C5               RAMDCHK: PUSH BC
05CF   CD E7 04                  CALL INITDI
05D2   CD A8 05                  CALL READDATA
05D5   F5                        PUSH AF
05D6   CD 15 05                  CALL DISABLE
05D9   F1                        POP AF
05DA   2F                        CPL
05DB   CD FE 04                  CALL INITDO
05DE   CD BB 05                  CALL WRITEDATA
05E1   CD 15 05                  CALL DISABLE
05E4   CD E7 04                  CALL INITDI
05E7   CD A8 05                  CALL READDATA
05EA   F5                        PUSH AF
05EB   CD 15 05                  CALL DISABLE
05EE   F1                        POP AF
05EF   2F                        CPL
05F0   CD FE 04                  CALL INITDO
05F3   CD BB 05                  CALL WRITEDATA
05F6   CD 15 05                  CALL DISABLE
05F9   F5                        PUSH AF
05FA   CD E7 04                  CALL INITDI
05FD   CD A8 05                  CALL READDATA
0600   F5                        PUSH AF
0601   CD 15 05                  CALL DISABLE
0604   F1                        POP AF
0605   47                        LD  B,A
0606   F1                        POP AF
0607   B8                        CP  B
0608   C1                        POP BC
0609   C9                        RET
```

```
                        ;*******************************************************
                        ;
                        ; You can treat a range of memory (specifed by two
                        ; address separated by a slash),move it from one place
                        ; to another in memory by using the MOVE   command.
                        ; The final command look like this:
                        ;   .<M>=<start>/<end> <destination><CR>

060A   22 9A FF         MMOVE:  LD (SYSSTK),HL  ; The starting address in HL.
060D   CD 72 07                 CALL GETHL      ; Get the ending address.
0610   22 9C FF                 LD (SYSSTK+2),HL
0613   CD 72 07                 CALL GETHL      ; Get the destination address.
0616   22 9E FF                 LD (SYSSTK+4),HL
0619   CD 1F 06                 CALL GMV
061C   C3 AA 07                 JP CR3

                        ;*******************************************************

061F   21 9A FF         GMV:    LD HL,SYSSTK
0622   CD 04 07                 CALL GETP       ; Load parameter from
                                                ; stack area into registers.
                                                ; Also check if the parameters
                                                ; are legal. After GETP,
                                                ; HL = start address of source
                                                ; BC = length to MOVE.
0625   DA FE 06                 JP C,ERROR      ; Jump to ERROR if the
                                                ; parameters are illegal.(i.e.,
                                                ; ending address < starting.)
0628   ED 5B 9E FF              LD DE,(SYSSTK+4) ; Load destination
                                                ; address into DE.
062C   ED 52                    SBC HL,DE       ; Compare HL and DE to
                                                ; determine to move up or down.
062E   30 0C                    JR NC,MVUP
                                                ; Move down
0630   EB                      EX DE,HL        ; HL = destination address.
0631   09                      ADD HL,BC       ; HL = dest.address+length.
0632   2B                      DEC HL          ; HL = end address of dest.
0633   EB                      EX DE,HL        ; DE = end address of dest.
0634   2A 9C FF                LD HL,(SYSSTK+2) ; HL = end address of source.
0637   CD 42 06                CALL LDDRE      ; Block transfer subroutine.
063A   13                      INC DE          ; DE = last address moved.
063B   C9                      RET
                        MVUP:                   ; Move up
063C   19                      ADD HL,DE       ; HL is changed by
                                                ; SBC HL,DE. Restore HL.
063D   CD 60 06                 CALL LDINC      ; Block transfer subroutine.
0640   1B                      DEC DE          ; DE = last address moved.
0641   C9                      RET
```

```
                        ;*******************************************************

0642  CD E7 04      LDDRE:  CALL INITDI
0645  CD A8 05              CALL READDATA
0648  F5                    PUSH AF
0649  CD 15 05              CALL DISABLE
064C  EB                    EX DE,HL
064D  CD FE 04              CALL INITDO
0650  F1                    POP AF
0651  CD BB 05              CALL WRITEDATA
0654  CD 15 05              CALL DISABLE
0657  EB                    EX DE,HL
0658  2B                    DEC HL
0659  1B                    DEC DE
065A  0B                    DEC BC
065B  78                    LD A,B
065C  B1                    OR C
065D  20 E3                 JR NZ,LDDRE
065F  C9                    RET

0660  CD E7 04      LDINC:  CALL INITDI
0663  CD A8 05              CALL READDATA
0666  F5                    PUSH AF
0667  CD 15 05              CALL DISABLE
066A  EB                    EX DE,HL
066B  CD FE 04              CALL INITDO
066E  F1                    POP AF
066F  CD BB 05              CALL WRITEDATA
0672  CD 15 05              CALL DISABLE
0675  EB                    EX DE,HL
0676  23                    INC HL
0677  13                    INC DE
0678  0B                    DEC BC
0679  78                    LD A,B
067A  B1                    OR C
067B  20 E3                 JR NZ,LDINC
067D  C9                    RET

                        ;*******************************************************
                        ;
                        ; To display eight consecutive memory contents.

067E  CD E7 04      MEM3:   CALL INITDI
0681  2A C4 FF              LD HL,(MADDR)
0684  06 08                 LD B,8
0686  CD 05 08      MEM5:   CALL SPACE1     ; Insert a space.
0689  CD A8 05              CALL READDATA
```

```
068C   CD 0A 08              CALL HEX2
068F   23                   INC HL
0690   10 F4                 DJNZ MEM5
0692   CD 15 05              CALL DISABLE
0695   C9                    RET

                    ;*******************************************************
                    ;
                    ; Executed when equal sign (=) or minus sign (-) key is
                    ; pressed. Executed in memory mode only.

                    MFOR:                 ; Display nest eight memory
                                          ; contents.
0696   2A C4 FF             LD HL,(MADDR)
0699   23                   INC HL
069A   23                   INC HL
069B   23                   INC HL
069C   23                   INC HL
069D   23                   INC HL
069E   23                   INC HL
069F   23                   INC HL
06A0   23                   INC HL
06A1   22 C4 FF      P102:  LD (MADDR),HL
06A4   3E 4D                LD A,'M'
06A6   CD 31 07             CALL ECHOCH      ; Get pattern '<M>='
06A9   CD FC 07             CALL HEXX
06AC   C3 E0 04             JP P101
                    MBACK:                ; Display last eight memory
                                          ; contents.
06AF   2A C4 FF             LD HL,(MADDR)
06B2   2B                   DEC HL
06B3   2B                   DEC HL
06B4   2B                   DEC HL
06B5   2B                   DEC HL
06B6   2B                   DEC HL
06B7   2B                   DEC HL
06B8   2B                   DEC HL
06B9   2B                   DEC HL
06BA   C3 A1 06             JP P102

                    ;*******************************************************
                    ;
                    ; Executed when 'F' key is pressed.
                    ; Store the data byte into all memory locations from
                    ; addr1 to addr2.
                    ; The final command look like this :
                    ;   <F>=<addr1> <addr2> <data><CR>
```

```
06BD                    FILLDA:
06BD   3E 10                    LD A,10H
06BF   32 FD FF                 LD (TYPEFG),A
06C2   3E 46                    LD A,'F'
06C4   CD 1C 05                 CALL MEMEX2     ; Get starting address.
06C7   CD CE 05                 CALL RAMDCHK
06CA   CA 56 05                 JP Z,RAMFAULT1 ; Jump to ERROR if the
                                               ; memory location of the
                                               ; starting address is not RAM.
06CD   E5                       PUSH HL
06CE   CD 72 07                 CALL GETHL      ; Get ending address.
06D1   E5                       PUSH HL
06D2   CD 72 07                 CALL GETHL      ; Get data.
06D5   7D                       LD A,L
06D6   A7                       AND A
06D7   E1                       POP HL
06D8   D1                       POP DE
06D9   EB                       EX DE,HL
06DA   F5                       PUSH AF
06DB   CD FE 04                 CALL INITDO
06DE   F1                       POP AF
06DF   CD BB 05                 CALL WRITEDATA
06E2   CD 15 05                 CALL DISABLE
06E5   EB                       EX DE,HL
06E6   ED 52                    SBC HL,DE
06E8   C8                       RET Z
06E9   DA FE 06                 JP C,ERROR      ; Jump to ERROR if starting
                                               ; address > ending address.
06EC   44                       LD B,H
06ED   4D                       LD C,L
06EE   62                       LD H,D
06EF   6B                       LD L,E
06F0   13                       INC DE
06F1   CD 60 06                 CALL LDINC
06F4   C3 AA 07                 JP DR3

                        ;*****************************************************
                        ;
                        ;  Function: Refer to READLN.
                        ;            C -- Get a string of characters.
                        ;            NC-- Reset the content of INPTR.


06E7   DA C1 07         GET:    JP C,RDLOOP
06FA   CD BB 07         GETT:   CALL READLN
06FD   C9                       RET
                        ;
                        ;*****************************************************
```

```
                        ;
                        ; Function : Print ERROR message.
                        ; Input : None
                        ; Output: None
                        ; Reg affected : AF HL .
                        ; Call : None

06FE  21 3C 08          ERROR:  LD HL,ERRSMSG
0701  C3 3F 04                  JP TRANSMSG
                        ;
                        ;*******************************************************
                        ;
                        ; Get parameter from stack buffer.

0704  5E                GETP:   LD E,(HL)       ; Load starting address into
                                                ; DE.
0705  23                        INC HL
0706  56                       LD D,(HL)
0707  23                       INC HL
0708  4E                       LD C,(HL)
0709  23                       INC HL
070A  66                       LD H,(HL)       ; Load ending address into
                                                ; HL.
070B  69                       LD L,C
070C  B7                       OR A            ; Clear carry flag.
070D  ED 52                    SBC HL,DE       ; Find difference.
                                               ; Carry flag is changed here.
070F  4D                       LD C,L
0710  44                       LD B,H
0711  03                       INC BC          ; Now BC contains the length.
0712  EB                      EX  DE,HL        ; Now HL contains the starting
                                               ; starting.
0713  C9                      RET
                        ;
                        ;*******************************************************
                        ;
                        ; Function: Clear display buffer and display prompt.
                        ; Input: None
                        ; Output: (OUTPTR) <-- INPBF
                        ; Reg affected: AF .
                        ; Call: CLEAR CHRWR .

0714  CD B2 07          CLRBF:  CALL CLEAR
0717  3E 3C                     LD A,3CH
0719  CD 9F 07                  CALL CHRWR
071C  C9                        RET

                        ;*******************************************************
```

```
                        ;
                        ; Function: check if a memory address is in RAM.
                        ; Input: HL -- address to be check.
                        ; Output: Zero flag -- 0, ROM or nonexistant;
                        ;                      1, RAM.
                        ; Destroyed reg. : AF.
                        ; Call: None

071D   7E               RAMCHK:  LD A,(HL)
071E   2F                        CPL
071F   77                        LD (HL),A
0720   7E                        LD A,(HL)
0721   2F                        CPL
0722   77                        LD (HL),A
0723   BE                        CP (HL)
0724   C9                        RET
                        ;
                        ;*******************************************************

                        CHKINP:                  ; Check all the datum in input
                                                 ; buffer are hexadecimal values
                                                 ; or not until <CR> met. Carry
                                                 ; flag is set if there exists
                                                 ; at least one non hexadecimal
                                                 ; value
0725   CD 6C 07                 CALL CHKHEX
0728   D8                       RET C
0729   C8                       RET Z
072A   CD 72 07         CHKIN1: CALL GETHL
072D   D8                       RET C
072E   C8                       RET Z
072F   18 F9                    JR  CHKIN1

                        ECHODCH:                 ; Echo the input character
                                                 ; with <?>=
                                                 ; ? is the input character.
0731   CD 9F 07                 CALL CHRWR      ; ?
0734   3E 3E                    LD A,3EH        ; >
0736   CD 9F 07                 CALL CHRWR
0739   3E 3D                    LD A,3DH        ; =
073B   CD 9F 07                 CALL CHRWR
073E   C9                       RET

                        ;*******************************************************
                        ;
                        ; Function: Use (GETPT) as a pointer increase HL until
                        ;           (HL-1) is one of the following delimeters:
                        ;           SPACE TAB : / = and (HL+1) is not SPACE
```

```
                        ;          or TAB.
                        ; Input : HL = (GETPT) -- Starting address.
                        ; Output: HL <-- HL+?
                        ;         (GETPT) <-- (GETPT)+?
                        ; Reg affected: AF HL.
                        ; Call: None

073F   2A FB FF         GETCHR:  LD HL,(GETPT)
0742   7E               LDA:     LD A,(HL)
0743   FE 20                     CP ' '          ; SPACE.
0745   28 04                     JR Z,SKIP
0747   FE 09                     CP 09H          ; TAB.
0749   20 0E                     JR NZ,EOS
074B   23               SKIP:    INC HL
074C   7E                        LD A,(HL)
074D   FE 20                     CP ' '          ; SPACE.
074F   28 FA                     JR Z,SKIP
0751   FE 09                     CP 09H          ; TAB.
0753   28 F6                     JR Z,SKIP
0755   22 FB FF         STPTR:   LD (GETPT),HL
0758   C9                        RET
0759   FE 0D            EOS:     CP 0DH          ; End of string?
075B   28 F8                     JR Z,STPTR      ; Yes.
075D   FE 3A                     CP 3AH          ; :
075F   28 EA                     JR Z,SKIP
0761   FE 3D                     CP 3DH          ; =
0763   28 E6                     JR Z,SKIP
0765   FE 2F                     CP 2FH          ; /
0767   28 E2                     JR Z,SKIP
0769   23                        INC HL
076A   18 D6                     JR  LDA
076C                     CHKHEX:
076C   21 C6 FF                  LD HL,INPBF
076F                     CHKHE2:
076F   22 FB FF                  LD (GETPT),HL

                        ;*********************************************************
                        ;
                        ; Function: Call GETCHR and convert ASSCII codes to
                        ;           hexadecimal values and store them int HL.
                        ; Input: (GETPT)
                        ; Output: (GETPT) <-- (GETPT)+?
                        ;         A <-- L
                        ;         H = 0 if there is only one hexadecimal digit.
                        ;         Carry flag = 1 if the data is not hexadecimal
                        ;                        digits.
                        ;         Zero  flag = 1 if the last ASCII code is <CR>.
                        ; Reg affected: AF DE HL.
```

```
                        ; Call : GETCHR ONE.

                        GETHL:                  ; Get 4 digit number to HL
                                                ; & L = A
                                                ; C (Non hexadecimal values)
                                                ; Z (ODH)
0772  21 00 00                LD HL,0000H       ; Assume input 0000
0775  E5                      PUSH HL           ; Temporary store in (SP),
                                                ; (SP+1)
0776  39                      ADD HL,SP         ; HL = SP
0777  EB                      EX DE,HL          ; Borrow SP for temporary
                                                ; buffer
0778  CD 3F 07                CALL GETCHR
077B  EB                      EX DE,HL
077C  FE 30             CV3:  CP '0'
077E  30 0A                   JR NC,CVT
0780  FE 0D                   CP 0DH
0782  20 03                   JR NZ,CV2
0784  E1                CV1:  POP HL
0785  7D                      LD A,L            ; String end.
0786  C9                      RET
0787                    CV2:
0787  A7                      AND A
0788  18 FA                   JR CV1
078A  FE 3A              CVT:  CP 3AH            ; :
078C  28 F9                   JR Z,CV2
078E  CD 28 08           CVTHEX: CALL ONE       ; ASCII to HEX
0791  38 0A                   JR C,NOTHEX
0793  ED 6F                   RLD               ; Rotate into (HL) i.e.
                                                ; (SP)
0795  23                      INC HL            ; SP+1
0796  ED 6F                   RLD
0798  2B                      DEC HL
0799  13                      INC DE
079A  1A                      LD A,(DE)
079B  18 DF                   JR CV3
                        NOTHEX:                 ; Error
079D  E1                      POP HL
079E  C9                      RET

                        ;*******************************************************
                        ;
                        ; Function: Store ASCII code in A register to input
                        ;           buffer.
                        ; Input: A -- a byte of ASCII code.
                        ;        (OUTPTR) -- Point to the result address in
                        ;                    input buffer.
                        ; Output: Store the ASCII code into (OUTPTR)
```

```
                        ;       (OUTPTR) <-- (OUTPTR)+1
                        ; Reg affected: AF
                        ; Call: None

079F   E5               CHRWR:  PUSH HL
07A0   2A F9 FF                 LD HL,(OUTPTR)
07A3   77                       LD (HL),A
07A4   23                       INC HL
07A5   22 F9 FF                 LD (OUTPTR),HL
07A8   E1                       POP HL
07A9   C9                       RET

                        ;*******************************************************

07AA   E5               CR3:    PUSH HL
07AB   2A F9 FF                 LD HL,(OUTPTR)
07AE   36 0D                    LD (HL),0DH
07B0   E1                       POP HL
07B1   C9                       RET

                        ;*******************************************************
                        ;
                        ; Function : Set the contents of (OUTPTR) to starting
                        ;            address of input buffer.
                        ; Input: None
                        ; Output: (OUTPTR) <-- INPBF
                        ; Reg affected : None
                        ; Call : None

07B2   E5               CLEAR:  PUSH HL
07B3   21 C6 FF                 LD HL,INPBF
07B6   22 F9 FF                 LD (OUTPTR),HL
07B9   E1                       POP HL
07BA   C9                       RET

                        ;*******************************************************
                        ;
                        ; Function: Get a string of characters and end with <CR>.
                        ; Input:    (OUTPTR) <-- Point to the result address in
                        ;           input buffer.
                        ; Output:   (INPTR) <-- (OUTPTR)
                        ;           (OUTPTR) <-- (OUTPTR)+?
                        ;           ? is the number of input characters.
                        ;           Zero flag -- Set if only <CR> is depressed.
                        ; Reg affected: AF BC DE HL AF' BC' DE' HL'.
                        ; Call : SCAN CHRWR.

07BB                    READLN:
```

```
07BB  2A F9 FF            LD HL,(OUTPTR)
07BE  22 FE FF            LD (INPTR),HL   ; Set input pointer.
07C1                RDLOOP:
07C1  CD 52 02            CALL SCAN
07C4  FE 1B               CP 1BH          ; Software Escape ( Esc ).
07C6  CA 31 00            JP Z,ESCAPE
07C9  FE 0D               CP 0DH          ; CR
07CB  28 11               JR Z,RDEND
07CD  FE 08               CP 08H          ; Back Space
07CF  28 18               JR Z,LEFT
07D1  FE 2D               CP 2DH          ; Minus sign (-)
07D3  28 EC               JR Z,RDLOOP
07D5  FE 3D               CP 3DH          ; Equal sign (=)
07D7  28 E8               JR Z,RDLOOP
07D9  CD 9F 07            CALL CHRWR
07DC  18 E3               JR RDLOOP
07DE                RDEND:
07DE  2A F9 FF            LD HL,(OUTPTR)
07E1  77                  LD (HL),A       ; Store 0DH.
07E2  ED 5B FE FF         LD DE,(INPTR)
07E6  ED 52               SBC HL,DE       ; Zero flag.
07E8  C9                  RET
                    LEFT:                 ; Back Space key service
                                          ; routine.
07E9  2A FE FF            LD HL,(INPTR)
07EC  ED 5B F9 FF         LD DE,(OUTPTR)
07F0  A7                  AND A
07F1  ED 52               SBC HL,DE
07F3  30 CC               JR NC,RDLOOP    ; Ignore if exceeding LEFT end.
07F5  EB                  EX DE,HL
07F6  2B                  DEC HL          ; Decrease the pointer of
                                          ; input buffer by one.
07F7  22 F9 FF            LD (OUTPTR),HL
07FA  18 C5               JR RDLOOP

                    ;*****************************************************
                    ;
                    ; Function: Convert binary data in HL to ASCII code.
                    ; Input: HL -- Two bytes of hexadecimal values in HL.
                    ;        (OUTPTR) -- Point to the result address in
                    ;        input buffer.
                    ; Output: Four ASCII code in (OUTPTR)-(OUTPTR)+3
                    ;         (OUTPTR) <-- (OUTPTR)+4
                    ; Reg affected : AF
                    ; Call: HEX2

07FC  7C            HEXX:   LD A,H
07FD  CD 0A 08              CALL HEX2
```

```
0800   7D                   LD A,L
0801   CD 0A 08             CALL HEX2
0804   C9                   RET

                    ;*******************************************************
                    ;
                    ; Write space in the input buffer.

0805   3E 20        SPACE1: LD A,' '
0807   C3 9F 07             JP CHRWR

                    ;*******************************************************
                    ;
                    ; Function: Convert binary data to ASCII code.
                    ; Input: A -- a byte in Aregister.
                    ;        (OUTPTR) -- Point to the result address in
                    ;        input buffer.
                    ; Output: The first ASCII code in (OUTPTR) and the
                    ;         second ASCII code in (OUTPTR)+1 .
                    ;         (OUTPTR) <-- (OUTPTR)+2
                    ; Reg affected: AF
                    ; Call: HEX1

080A   E5            HEX2:  PUSH HL
080B   21 C2 FF             LD HL,TEMP
080E   77                   LD (HL),A
080F   AF                   XOR A
0810   ED 6F                RLD
0812   CD 1D 08             CALL HEX1
0815   AF                   XOR A
0816   ED 6F                RLD
0818   CD 1D 08             CALL HEX1
081B   E1                   POP HL
081C   C9                   RET

                    ;*******************************************************
                    ;
                    ; Function: Convert binary data to ASCII code.
                    ; Input: A -- LSB 4 bits contains the binary data.
                    ;        (OUTPTR) -- Point to the result address in
                    ;        input buffer.
                    ; Output: ASCII code in (OUTPTR).
                    ;         (OUTPTR) <-- (OUTPTR)+1.
                    ; Reg affected:  AF
                    ; Call: CHRWR

081D   C6 30         HEX1:  ADD A,'0'
081F   FE 3A                CP 3AH
```

```
0821   38 02                JR C,HHH
0823   C6 07                ADD A,7
0825   C3 9F 07      HHH:   JP CHRWR

                     ;*******************************************************
                     ;
                     ; Function: Convert a byte (ASCII code) in A register
                     ;           to hexadecimal digit.
                     ; Input:    A -- ASCII code.
                     ; Output:   A -- Hexadecimal values.
                     ;           Carry flag = 1 if the data is not a
                     ;           hexadecimal digit.
                     ;           (HEXFLAG) is not zero if the content of A
                     ;           within 'A' and 'F'.
                     ; Reg affected: AF
                     ; Call : None

0828   FE 47         ONE:   CP 47H          ; 'F'+1
082A   3F                   CCF
082B   D8                   RET C
082C   D6 30                SUB '0'
082E   D8                   RET C
082F   FE 0A                CP 10
0831   3F                   CCF
0832   D0                   RET NC
0833   D6 07                SUB 7
0835   FE 0A                CP 10
0837   D8                   RET C
0838   32 BB FF             LD  (HEXFLAG),A
083B   C9                   RET

083C   20 45 52 52   ERRSMSG: DEFB ' ERRORS      ',0DH
0840   4F 52 53 20
0844   20 20 20 20
0848   20 20 0D
E000                        ORG 0E000H
       75 FF         N      EQU 0FF75H      ;DEFB  1
       76 FF         OPTR   EQU 0FF76H      ;DEFB  2
       79 FF         PTR    EQU 0FF79H      ;DEFB  32 (16 BUFFER)
       9A FF         SYSSTK EQU 0FF9AH      ;DEFB  32 (16 LEVEL STACK)
       BB FF         HEXFLAG EQU 0FFBBH     ;DEFS  1
       BC FF         START  EQU 0FFBCH      ;DEFS  2
       BE FF         RANGE  EQU 0FFBEH      ;DEFS  2
       C0 FF         POWERUP EQU 0FFC0H     ;DEFS  1
       C1 FF         PRTFLG EQU 0FFC1H      ;DEFS  1
       C2 FF         TEMP   EQU 0FFC2H      ;DEFS  2
       C4 FF         MADDR  EQU 0FFC4H      ;DEFS  2
       C6 FF         INPBF  EQU 0FFC6H      ;DEFS  50 (32H BYTE)
```

```
      F9 FF           OUTPTR    EQU  OFFF9H      ;DEFS  2
      FB FF           GETPT     EQU  OFFFBH      ;DEFS  2
      FD FF           TYPEF6    EQU  OFFFDH      ;DEFS  1
      FE FF           INPTR     EQU  OFFFEH      ;DEFS  2
E000                            END
```

********** SYMBOLIC REFERENCE TABLE **********

ADDRLD	026E	ADDRMON	038D	BACK	0046	BUSRQAK	= 0006
CHECK	0506	CHECKK	04EF	CHKHE2	076F	CHKHEX	076C
CHKIN1	072A	CHKINP	0725	CHRWR	079F	CLEAR	07B2
CLRBF	0714	COMPLETE	0550	CONTROL	= 0004	CR3	07AA
CV1	0784	CV2	0787	CV3	077C	CVT	078A
CVTHEX	078E	DATA	= 0001	DELAY	0471	DELAY1	0476
DISABLE	0515	DISPLAY	0262	DOCUMENT1	0568	DOCUMENT2	0586
ECHOCH	0731	ENABLE	= 0009	ENDMASSAGE	0414	ENDMSG	044B
ENDMSG1	0462	EOS	0759	ERROR	06FE	ERRSMSG	083C
ESCAPE	0031	FILLDA	06BD	FOR	003C	GET	06F7
GETADDR	03AA	GETCHR	073F	GETHL	0772	GETP	0704
GETPT	= FFFB	GETT	06FA	GMV	061F	HEX1	081D
HEX2	080A	HEXFLAG	= FFBB	HEXX	07FC	HHH	0825
HIGHADDR	= 0002	IGNORE	004E	INI	00C9	INI5	00DF
INITDI	04E7	INITDO	04FE	INITRS232	00C0	INPBF	= FFC6
INPTR	= FFFE	KEYEXEC	0059	LDA	0742	LDDRE	0642
LDINC	0660	LDMSG	047E	LDMSG1	0487	LDMSG2	04A0
LDRDYMSG	04B9	LEFT	07E9	LOAD	027D	LOAD1	02B6
LOADAGN	0271	LOADRDY	02CB	LOWADDR	= 0000	MADDR	= FFC4
MAIN	001E	MBACK	06AF	MEM3	067E	MEM5	06B6
MEMEX1	051F	MEMEX2	051C	MEMEX3	052E	MEMEXEC	04C7
MFOR	0696	MMLOOP	0542	MMODFY	0537	MMOVE	060A
MONITOR	039A	MONITOR1	03CD	MONITOR2	03D5	MVUP	063C
N	= FF75	NOTHEX	079D	ONE	0828	OPTR	= FF76
OUTPTR	= FFF9	P101	04E0	P102	06A1	P8255NO0	= 0003
P8255NO1	= 0007	P8255NO2	= 000B	POWERUP	= FFC0	PRTFLG	= FFC1
PTR	= FF79	PWCODE	= 00A5	RAMCHK	071D	RAMDCHK	05CE
RAMFAULT	00E5	RAMFAULT1	0556	RAMT1	00CF	RAMT2	00D2
RANGE	= FFBE	RDEND	07DE	RDLOOP	07C1	READDATA	05A8
READLN	07BB	RESETDEV	0087	RESETMSG	00A3	RESTART	0003
RFLTMSG	0212	SAVE	02DD	SAVE1	0316	SAVE2	031A
SAVEREDY	0337	SCAN	0252	SETTYPE	02D7	SKIP	074B
SPACE1	0805	START	= FFBC	STARTP6	00EC	STPTR	0755
SVMSG	0352	SVMSG1	035B	SVMSG2	0374	SVRDYMSG	0343
SYSSTK	= FF9A	TEMP	= FFC2	TESTIN	0330	TITLEMSG	0120
TITLEMSG1	013E	TITLEMSG2	015D	TITLEMSG3	016B	TITLEMSG4	01B9
TITLEMSG5	01AC	TITLEMSG6	01CD	TITLEMSG7	01F2	TMONITOR	0408
TNEXT	00DA	TRANSMIT	0463	TRANSMIT1	0464	TRANSMSG	043F
TRANSMSG1	0456	TX1	0420	TX2	0432	TX_DATA	041F
TYPEFG	= FFFD	URTCNT	= 000D	URTDA	= 000C	WAIT	03C6
WRITEDATA	05BB						

0000 ASSEMBLY ERRORS

กิตติกรรมประกาศ

วิทยานิพนธ์เรื่องนี้สำเร็จได้ด้วยคำแนะนำต่างๆจากอาจารย์ จึงขอขอบคุณ อาจารย์วิริยะ กองรัตน์ และอาจารย์ในภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม ทุกท่าน มา ณ. ที่นี้เป็นอย่างสูง

หนังสืออ้างอิง

1. "Turbo Pascal version 3.0 reference manual",Borland International Inc.,1985
2. James W. Coffron,"Z80 Application",Sybex Inc.,1983
3. David C.Willen and Jefferey I.Krantz , "8088 Assembly Language ProgrammingThe IBM PC",Howard W.Sam & Co.Inc,1983
4. "MPF-IP User Manual",Multitehc Industrial Co,1983
5. "MPF-IP Monitor Program Source List",Multitehc Industrial Co,1983
6. "IBM Technical Reference", International Business Machines Corp,1983
7. ยืน ภู่วรวรรณ และ วิทยา เชียงกูล,"ไมโครโปรเซสเซอร์และไมโครคอมพิวเตอร์ (Z-80 MICROPROCESSOR)",บริษัท ซีเอ็ดยูเคชั่น จำกัด,2527
8. "คู่มือไอซีทีทีแอล",บริษัท ซีเอ็ดยูเคชั่น จำกัด,2526
9. "คู่มือไอซีไมโครโปรเซสเซอร์",บริษัท ซีเอ็ดยูเคชั่น จำกัด,2529
10. "คู่มือไอซีซีพซัพพอร์ท",บริษัท ซีเอ็ดยูเคชั่น จำกัด,2529
