เครื่องเก็บข้อมูลสำหรับเครื่องจักรในโรงงานอุตสาหกรรม

DATA ACQUISITION MACHINE

โดย

นายก้องเกียรติ บุญธปัญญา 41012044

นางสาวพรพิมล ทัพชัย 41012062

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

สาขาวิชาโทรคมนาคม ภาควิชาเทคนิคอุตสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

หัวข้อปริญญานิพนธ์ เครื่องเก็บข้อมูลสำหรับเครื่องจักรในโรงงานอุตสาหกรรม
(DATA ACQUISITION MACHINE)

โดย 1. นายก้องเกียรติ บุญธปัญญา 41012044
2. นางสาวพรพิมล ทัพชัย 41012062

ภาควิชา เทคนิคอุตสาหกรรม
อาจารย์ที่ปรึกษา อ.อรรถสิทธิ์ หล่าสกุล

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเหล้าเจ้าคุณทหารลาดกระบังอนุมัติให้ปริญญานิพนธ์ฉบับนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

..... อ.ที่ปรึกษา
(.....)

..... กรรมการ
(.....)

..... กรรมการ
(.....)

..... กรรมการ
(.....)

..... กรรมการ
(.....)

..... กรรมการ
(.....)

หัวข้อปริญญานิพนธ์ เครื่องเก็บข้อมูลสำหรับเครื่องจักรในโรงงานอุตสาหกรรม

โดย 1. นายก้องเกียรติ บุญธปัญญา 41012044
2. น่างสาวพรพิมล ทัพชัย 41012062

อาจารย์ที่ปรึกษา อ.อรรถสิทธิ์ หล่าสกุล
ระดับการศึกษา ปริญญาอุตสาหกรรมศาสตร์บัณฑิต
ปีการศึกษา 2542

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ได้กล่าวถึง การออกแบบสร้างเครื่องเก็บข้อมูลด้วยไมโครโปรเซสเซอร์ เพื่อเป็นการศึกษาถึงการประยุกต์ใช้งานของ Z80180 ทำให้เกิดประโยชน์ใช้งานจริง ซึ่งได้ศึกษาพัฒนาทั้งด้าน ซอร์ฟแวร์และ ฮาร์ดแวร์เพื่อการเก็บรับข้อมูลต่าง ๆ จากเครื่องทำความเย็นในโรงงานอุตสาหกรรมโดยอัตโนมัติไม่ต้องอาศัยมนุษย์มาควบคุมดูแลลักษณะของโครงงานจะใช้งานร่วมกับบอร์ด CP – JR180 V2.0 และยังสามารถแสดงผลที่ LCD ได้ สำหรับช่องในการรับข้อมูลมี 4 ช่องทางพร้อมทั้งช่องที่ติดต่อกับคอมพิวเตอร์เพื่อทำการรับ - ส่งข้อมูล

PROJECT DATA ACQUISITION MACHINE

NAME 1. MR.KONGKIAT BOONTHAPUNYA
2. MISS.PORNPIMON TUPCHAI

ADVISOR MR.ATTHASIT HLASKUN
LEVEL OF STUDY BACHALOR'S DEGREE IN INDUSTRIAL
ACADEMIC YEAR 1998

ABSTRACT

The one equipment that very important for maintaining and controlling the efficiency in factory is the data acquisition machine. This equipment gives a capability of viewing and recognition the statistic of data, which is, received form many kind of measuring machine such as the cooler machine. The one problem is most of data is directed to printer to make the document all the time. Most of data cannot be keeping for long time. The data acquisition machine that can automatically acquits data and send to computer is a vary importance machine.

In this thesis the construction of the low cost data acquisition machine is presented. This machine consist of two parts, one is the multiplex for multi-channel input and another one is Microprocessor Z80180 which can be used to keep a large of data and send data to computer (PC). This machine can be developed to receive data form many kind of measuring machine that use RS-232C for communication.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ สำเร็จลุล่วงไปได้ด้วยดี เกิดจากความร่วมมือร่วมใจของคณะผู้จัดทำ โดยได้รับความช่วยเหลือทางด้านคำแนะนำต่าง ๆ เป็นอย่างดีจากอาจารย์อรรถสิทธิ์ หล่าสกุล และอาจารย์อีกหลายท่านในภาคเทคนิคอุตสาหกรรม และเพื่อน ๆ ภาคอื่น ที่ได้ให้ความช่วยเหลือทางด้านคำแนะนำหลาย ๆ ด้าน

ท้ายนี้คณะผู้จัดทำ ขอขอบพระคุณ อาจารย์ทุกท่านที่ช่วยประสาทวิชาให้ความรู้ต่าง ๆ จนสามารถทำโครงงานนี้ได้

คณะผู้จัดทำ

9 กุมภาพันธ์ 2543

สารบัญ

สารบัญรูป

สารบัญตาราง

หน้า

บทที่1
บทนำ

1.1 แนวความคิดและที่มา

ในปัจจุบันการควบคุมอุปกรณ์ต่าง ๆ ในโรงงานอุตสาหกรรม จะเป็นลักษณะของการควบคุมด้วยอุปกรณ์อัตโนมัติ เช่น อุปกรณ์จำพวก PLC (Programmable Logic Control) หรือการใช้คอมพิวเตอร์แบบอื่น ๆ มาใช้ควบคุมกัน ทั้งนี้เพื่อให้ใช้งานเพื่อให้ได้ผลผลิตที่มากที่สุดในขณะที่รายจ่ายต้องพยายามให้น้อยที่สุด วิธีการหนึ่งที่จะลดรายจ่ายของโรงงานก็คือการควบคุมสภาวะการใช้อุปกรณ์ที่ไม่จำเป็นในบางเวลา ซึ่งในส่วนนี้ระบบที่เป็นอัตโนมัติจะมีราคาแพงมากและมีใช้น้อยในเมืองไทย ดังนั้นวิศวกรบุคคลที่ทำหน้าที่วิเคราะห์และจัดการระบบการทำงานของโรงงานเพื่อให้สิ้นเปลืองพลังงานน้อยที่สุดจึงมีความจำเป็นอย่างมาก ตัวอย่างเช่นการควบคุมอุณหภูมิโดยเครื่องทำความเย็น (Chiller) ภายในโรงงาน ซึ่งแต่ละจุดจะมีค่าที่ต้องใช้อุณหภูมิที่ไม่เท่ากัน เหล่านี้เป็นต้น

และสำหรับ Chiller (เครื่องทำความเย็น) นั้นปกติจะส่งผลของการวัดข้อมูลต่าง ๆ หลายค่าออกมาจากภายในโรงงานทุก ๆ หนึ่งชั่วโมง (สามารถเซ็ตได้) และค่าที่ได้นั้นจะถูกส่งเข้าสู่เครื่องพิมพ์โดยตรงทุกชั่วโมงโดยจะพิมพ์ค่าต่าง ๆ ที่จำเป็นเหล่านั้น ซึ่งต่อมาผู้วิเคราะห์ระบบก็จะนำเอาข้อมูลนี้ไปเขียนเป็นกราฟเพื่อจะได้ทราบถึงอัตราต่าง ๆ ของการใช้เครื่องทำความเย็นในช่วงระยะเวลาต่าง ๆ เช่น หนึ่งวัน หนึ่งอาทิตย์ หรือแม้หนึ่งปีเพื่อจะได้ควบคุมให้เครื่องทำงานได้อย่างถูกต้องที่สุด (ปกติจะมีเครื่อง Chiller 4 ตัวขึ้นไปต่อหนึ่งโรงงาน) และในเมืองไทยหลาย ๆ โรงงานก็มีการใช้งานเครื่องในลักษณะนี้มากมาย ปัญหาที่พบก็คือ

1. ผู้วิเคราะห์จะพบกับความยุ่งยากมากในการนำข้อมูลแต่ละตัว (ซึ่งมีหลายข้อมูล) มาทำการเขียนเป็นกราฟ

2. ข้อมูลไม่สามารถถูกเก็บได้อย่างมีประสิทธิภาพอันเนื่องจากข้อมูลจะถูกพิมพ์ออกมาอย่างต่อเนื่องตลอด หากกระดาษสิ่งพิมพ์เหล่านั้นหายก็ไม่สามารถทราบข้อมูลเหล่านั้นได้เลย

3. สิ้นเปลืองกระดาษของเครื่องพิมพ์เป็นอย่างมาก

ดังนั้นโครงงานนี้จึงถูกนำเสนอเพื่อสร้างเครื่องเก็บข้อมูล เพื่อแก้ปัญหาดังที่กล่าวมาข้างต้น

1.2 วัตถุประสงค์และจุดมุ่งหมายในการทำปริญญานิพนธ์

1. เพื่อสร้างเครื่องต้นแบบในการเก็บข้อมูลจากเครื่องทำความเย็นในโรงงานอุตสาหกรรมโดยอัตโนมัติด้วยไมโครโปรเซสเซอร์

2. เพื่อศึกษาถึงระบบอินพุต – เอาท์พุต ระบบบัส (BUS) ของคอมพิวเตอร์พร้อมทั้งการอินเทอร์เฟส (Interface) กับอุปกรณ์ภายนอก

3. เพื่อศึกษาถึงการโปรแกรมภาษาแอสแซมบลี

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1. เครื่องเก็บข้อมูลสำหรับเครื่องจักรในโรงงานอุตสาหกรรมซึ่งมีคุณสมบัติดังนี้
   - สามารถรอรับเก็บข้อมูลได้ตลอด 24 ชั่วโมง
   - มีความจุในการเก็บข้อมูลได้มาก
   - รับอินพุตได้อย่างน้อย 4 อินพุตจากเครื่องทำความเย็น
   - สามารถติดต่อกับเครื่องไมโครคอมพิวเตอร์เพื่อจัดเก็บข้อมูลได้หากมีการเรียกใช้ดูทีหลัง ได้ทุกเวลา
   - มีส่วนของการติดต่อกับผู้ใช้เพื่อการใช้งานที่สะดวก
2. ช่วยอำนวยความสะดวกให้แก่วิศวกรในการนำข้อมูลไปวิเคราะห์

1.4 เนื้อหาปริญญานิพนธ์

บทที่ 1 กล่าวถึง แนวความคิดและที่มา วัตถุประสงค์ และประโยชน์

บทที่ 2 กล่าวถึงทฤษฎีที่จำเป็นในโครงงานนี้ เช่น ระบบคอมพิวเตอร์พื้นฐาน ลักษณะทั่วไปของบอร์ด CP-JR180 V2.0

บทที่ 3 กล่าวถึงหลักการทำงานของฮาร์ดแวร์ ซอร์ฟแวร์ และวงจรมัลติเพล็ก

บทที่ 4 สรุปผลและวิจารณ์

1.5 โครงงานประกอบด้วยส่วนสำคัญ 2 ส่วน คือ

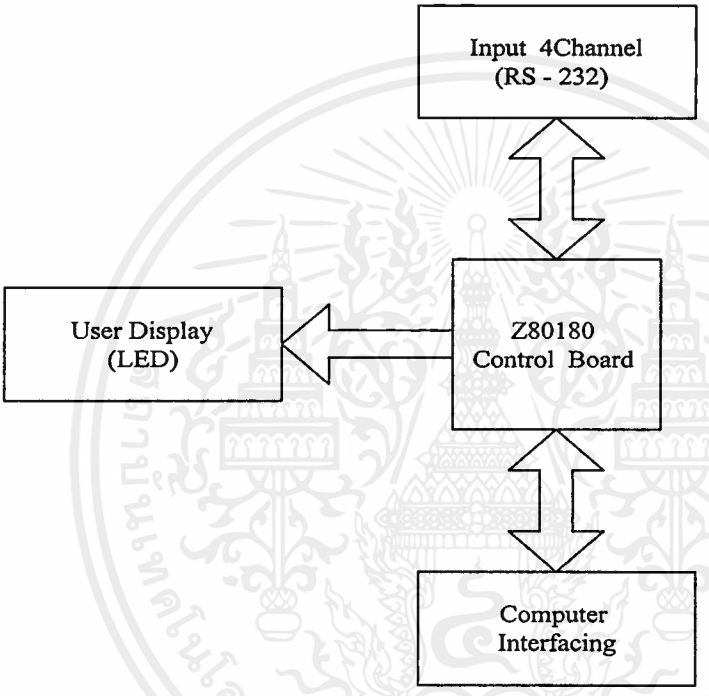
1.5.1 ซอร์ฟแวร์

ส่วนที่เป็นซอร์ฟแวร์ คือส่วนที่เป็นโปรแกรมใช้ในการมัลติเพล็กเซอร์รับและจัดเรียงข้อมูลแสดงผลในสภาวะต่าง ๆ พร้อมทั้งโปรแกรมในส่วนของการติดต่อกับคอมพิวเตอร์

1.5.2 ฮาร์ดแวร์

- บอร์ดไมโครคอนโทรลเลอร์ CP – JR180 V2.0
- วงจรมัลติเพล็กเซอร์
- ส่วนของการแสดงผล ได้แก่ LCDแสดงสภาวะการทำงาน LED

เราสามารถเขียนเป็นบล็อกไดอะแกรมของฮาร์ดแวร์ ได้ดังนี้



รูปที่ 1 แสดงบล็อกไดอะแกรมของฮาร์ดแวร์

บทที่ 2
ทฤษฎี

2.1 ระบบคอมพิวเตอร์พื้นฐาน (Basic computer system)

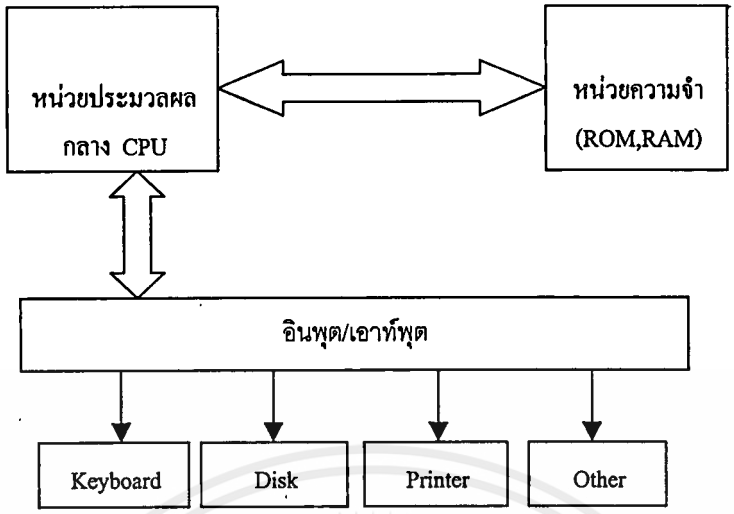
มีบล็อกไดอะแกรมการทำงาน ดังรูปที่ 2 หัวใจหลักคือ หน่วยประมวลผลกลางหรือ CPU(Central processing unit) จะทำหน้าที่ประมวลผลข้อมูลต่าง ๆ เพื่อนำผลลัพธ์ที่ได้ไปทำการควบคุมระบบต่อไป ภายใน CPU ประกอบด้วย

2.1.1 หน่วยจัดการทางคณิตศาสตร์ (Arithmatic Logic Unit : ALU) ทำหน้าที่จัดการด้านการคำนวณทางคณิตศาสตร์และลอจิก แล้วนำผลลัพธ์หรือข้อมูลที่ได้ส่งให้แก่ CPU เพื่อประมวลผลและควบคุมการทำงานต่อไป

2.1.2 หน่วยเก็บข้อมูลชั่วคราว หรือ รีจิสเตอร์ (Register) ทำหน้าที่เก็บข้อมูล เพื่อเตรียมเข้าสู่กระบวนการประมวลผล เก็บข้อมูลที่ได้จากการประมวลผลโดย ALU นำข้อมูลเพื่อส่งออกไปยังอุปกรณ์ภายนอก

2.1.3 แฟลก (Flag) ทำหน้าที่แสดงสภาวะการทำงานของ CPU ว่าขณะนี้เกิดอะไรขึ้นมีสภาวะลอจิกเป็นอย่างไร ซึ่งสามารถใช้ประโยชน์จากแฟลกนี้ในการตั้งเงื่อนไขในการทำงานของระบบคอมพิวเตอร์

2.1.4 สแต็กและโปรแกรมเคาท์เตอร์ (Stack & Program counter) สแต็กทำหน้าที่เป็นที่พักข้อมูล เมื่อ CPU กระโดดไปทำโปรแกรมย่อย ก็จะเก็บข้อมูลที่ประมวลผลค้างไว้ที่สแต็ก ส่วนโปรแกรมเคาท์เตอร์เป็นรีจิสเตอร์เก็บค่าการทำงานของ CPUว่าขณะนี้ CPU ทำงานไปถึงแอดเดรสใด หรือเป็นเสมือนมอนิเตอร์แสดงตำแหน่งการทำงานของ CPU นั่นเอง



รูปที่ 2 ระบบคอมพิวเตอร์เบื้องต้น

ส่วนที่สอง คือ หน่วยความจำซึ่งก็มีทั้งแบบรอม (Read Only Memory: ROM) และแรม (Read Access Memory : RAM) โดยรอมจะทำหน้าที่เก็บโปรแกรมควบคุมการทำงานและควบคุมฮาร์ดแวร์ทั้งหมดหรือที่เรียกว่า โปรแกรมมอนิเตอร์ (moniter program) และหน่วยความจำแรมใช้เป็นที่พักข้อมูลระหว่างรอการประมวลผล หรือเก็บข้อมูลเพื่อเตรียมการส่งออกไปยังอุปกรณ์ภายนอก

ส่วนพอร์ตอินพุตจะใช้เชื่อมต่ออุปกรณ์อินพุตที่จะส่งข้อมูลมายังคอมพิวเตอร์อันได้แก่ คีย์บอร์ด เป็นต้น

ส่วนพอร์ตเอาต์พุตจะใช้เชื่อมต่อกับอุปกรณ์เอาต์พุตเพื่อแสดงผลการทำงานได้แก่ จอมอนิเตอร์ ปรินท์เตอร์ เป็นต้น

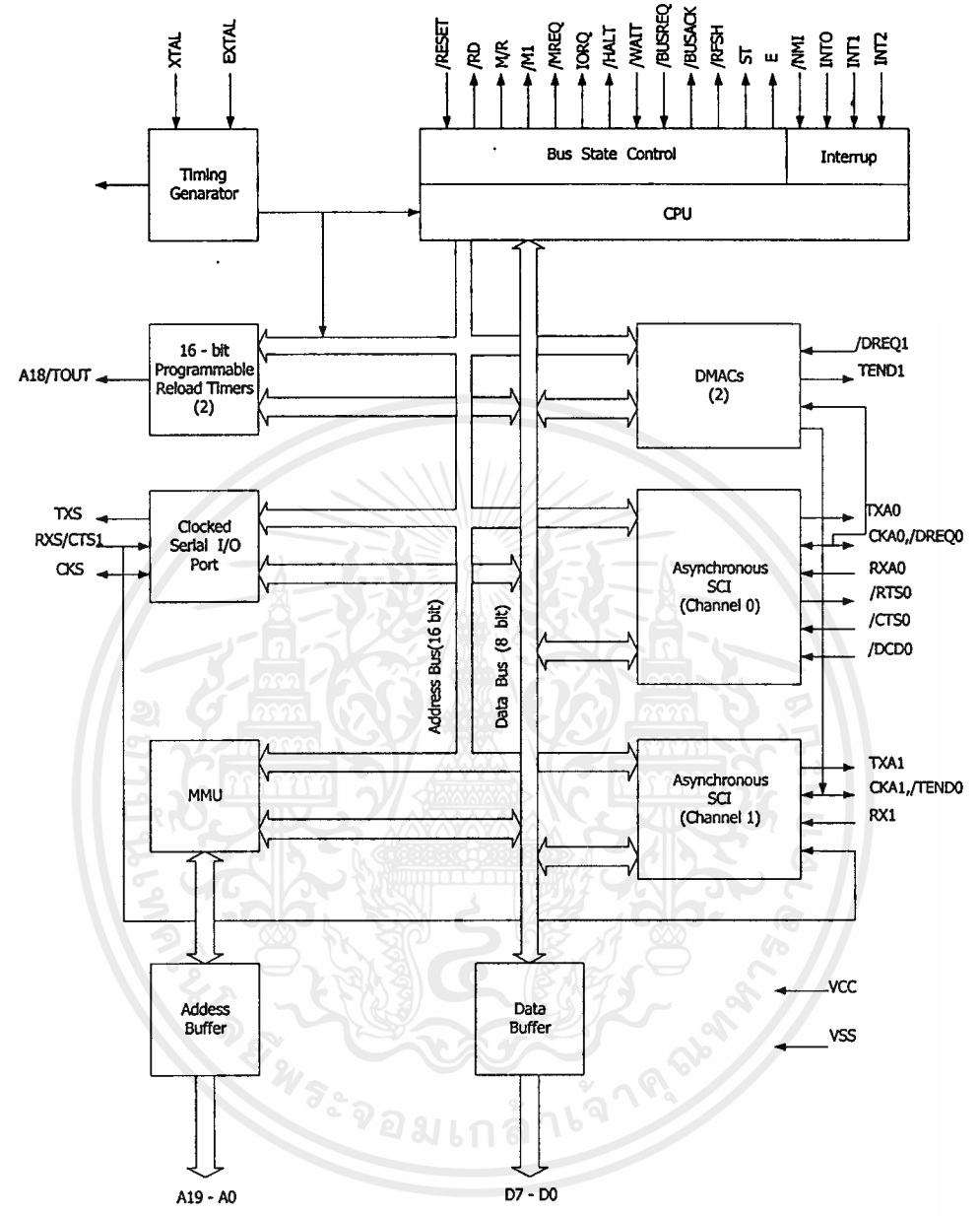
นอกเหนือไปจากนั้น คอมพิวเตอร์ยังมีพอร์ตสำหรับการสื่อสารข้อมูลอีกด้วย ที่นิยมใช้กันมากคือ พอร์ตสื่อสารอนุกรม RS – 233C RS422 และอื่น ๆ

2.2 Z80180/Z180MPU

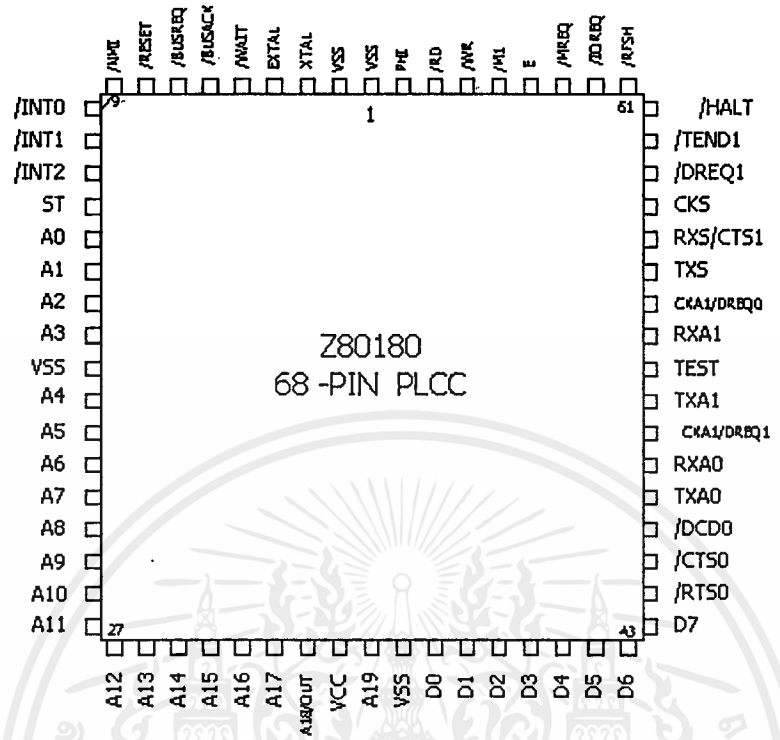
Z80180 เป็น CPU ที่มีความสามารถสูงที่ได้รวมชิพ (CHIP) สำคัญอื่นๆไว้ใน CPU CHIP เดียว จึงทำให้มีลักษณะคล้ายกับ CPU ที่ใช้งานควบคุมในจำพวก “SINGLE CHIP” แต่เนื่องจากชิงเกิ้ลชิพ มีข้อดีคือเป็นระบบเล็กราคาถูก แต่ข้อเสียคือการโปรแกรมควบคุมค่อนข้างยากในตอนเริ่มต้นและกับระบบงานที่ใหญ่ขึ้นจะเขียนโปรแกรมได้ยาก แต่ Z80180 ทางด้านโปรแกรมจะ

สะดวกอย่างมากเพราะคำสั่งที่ใช้มีมาก และตรงไปตรงมาทั้งคู่มือภาษาไทยและตัวอย่างการใช้งานอย่างมากมายเพราะ CPU Z80180 นี้เป็นซุปเปอร์คอมแพท (SUPPER COMPAT) Z80 คือคำสั่งทั้งหมดยังเป็น Z80และได้เพิ่มชุดคำสั่งขึ้นมาเพื่อเพิ่มความสะดวกในการใช้งานขึ้นอีก

เมื่อมองดูระบบไมโครคอนโทรลเลอร์ "SINGLE CHIP" แล้ว Z80180 จะด้อยกว่าตรงที่ไม่มี ROM,RAM และพอร์ตแต่ถ้าเป็นระดับงานอุตสาหกรรมแล้วระบบของ Z80180 กับชิพไมโครคอนโทรลเลอร์ แล้วจะไม่ต่างกันเลยเพราะความต้องการเนื้อที่ในการเก็บข้อมูลมากและพอร์ต มากตามมาจึงทำให้ต้องต่อเพิ่มวงจรภายนอกขึ้น จึงทำให้ Z80180 ในระดับงานควบคุมอุตสาหกรรมคล่องตัวกว่ามากเพราะภายใน Z80180 ประกอบด้วยเป็น CMOS,OSCILATOR ในตัว RUN ที่ 10 MHz - 20 MHz , MMU ชิพอ้างหน่วยความจำ (MEMORY) ได้ 1 MBYTE , DMA 2 CHANNEL , PORT สื่อสาร UART 2 CHANEL , CLOCK SERIAL I/O , 16 BIT TIMER COUNTER และเกี่ยวกับพอร์ตสื่อสารสามารถทำมัลติโปรเซสเซอร์คอมมูนิเคชั่น (MULTI PROCESSOR COMMUNICATION) ซึ่งโครงสร้างของ ชิพนี้จะเป็นดังรูป



รูปที่ 3 โครงสร้างภายในของชิพ Z80180/Z180 MPU



รูปที่ 4 แสดงตำแหน่งการวางขาของ Z80180/Z180 MPU

ขาการใช้งาน

AO-A19

ADDRESS BUS 20 เส้น ระหว่าง RESET จะเป็น HIGH IMPEDANCE

BUSAK

BUS ACKNOWLEDGE เป็นขา OUTPUT ACTIVE LOW ทำงานก็ต่อเมื่อ Z80180ตอบสนองต่อการขอBUS ของ BUSRQ และจะทำให้ BUS ข้อมูล BUS ADDRESS และสัญญาณ CONTROL บางเส้นเป็น HIGH IMPEDANCE

BUSRQ

BUS REQUEST เป็นขา INPUT ACTIVE LOW ซึ่งจะมีความสำคัญสูงกว่า NMI โดยจะมีการตรวจสอบสัญญาณนี้ทุกๆการสิ้นสุดของ MACHINE CYCLE

CKA0,CKA1

ASYNCHRONOUS CLOCK 0 และ 1 เป็นขาสัญญาณ CLOCK แบบ2ทิศทางคือจะใช้เป็นขาอินพุตหรือเอาต์พุตก็ได้

CKS

SERIAL CLOCK เป็นขา CLOCK 2 ทิศทางของ CSI/O

CLOCK

เป็นขา OUTPUT โดยจะเป็นครึ่งหนึ่งของ X'TALหรือ CLOCK OUT เช่น X'TAL 12MHz Z80180 จะ RUN ที่ 6MHz

CTS0-CTS1

CLEAR TO SEND 0 และ 1 เป็นขาอินพุต ACTIVE LOW ใช้ในการควบคุม MODEM

D0-D7

DATA BUS เป็นแบบ 2 ทิศทาง

DCDO

DATA CARRIER DETECT 0 เป็นขา INPUT ACTIVE LOW ใช้ควบคุมในการติดต่อกับ MODEM ของ ASCI CHANEL 0

DREQ0-DREQ1

DMA REQUEST 0 และ 1 เป็นขาอินพุต ACTIVE LOW ใช้ในการขอ DMA และขานี้จะโปรแกรมได้ว่าให้ตรวจสัญญาณที่ขอบหรือระดับได้

E

ENABLE CLOCK เป็นขาเอาต์พุต ACTIVE HIGH ซึ่งใช้ซิงค์การทำงานกับอุปกรณ์ภายนอกระหว่างการทำงานเกี่ยวกับ BUS และใช้เชื่อมต่อกับอุปกรณ์ในตระกูล 68XX และ 80XX

HALT

เป็นขาเอาต์พุต ACTIVE LOW จะทำงานเมื่อทำคำสั่ง HALT หรือ SLP

INTO

MASKABLE INTERRUPT 0 เป็นขาอินพุต ACTIVE LOW สัญญาณที่ขานี้จะถูกตรวจทุกๆการสิ้นสุดของคำสั่ง

INT1,INT2

เช่นเดียวกับ INT0 แต่มีระดับความสำคัญรองลงมาตามลำดับ

IORQ

เป็นขาเอาต์พุต เพื่อบอกว่ากำลังติดต่อ I/O หรือขา IOE ใน 64180

M1

MACHINE CYCLE 1 เป็นขาเอาต์พุต ACTIVE LOW จะทำงานเมื่อ FETCH OP-CODE หรือเป็นขาของ 64180 LIR

NM1

NON MASKABLE INTERRUPT เป็นขาอินพุต ACTIVE LOW ขานี้จะตอบรับการ INTERRUPT เสมอโดยไม่สามารถหยุดด้วย SOFTWARE

RD

เป็นขาที่ใช้ทำการอ่านข้อมูลจาก MEMORY หรือ I/O

RFSH

เป็นขาที่ให้ ADDRESS LOW (A0-A7) ไป REFRESH DYNAMIC RAM หรือขา REF ของ 64180

RTS0

REQUEST TO SEND เป็นขาเอาต์พุต ACTIVE LOW ขานี้ใช้โปรแกรมสัญญาณควบคุมโมเด็มของ ASCI CANEL 0

RXA0,RXA1

RECEIVE DATA 0 และ 1 เป็นขารับสัญญาณจาก SERIAL ของ ASCI

RXS

CLOCK SERIAL RECEIVE DATA เป็นขารับสัญญาณ SERIAL ของ CSIO

ST

STATUS เป็นขาเอาต์พุต ACTIVE HIGH ใช้แสดงสถานะการทำงานของ CPU โดยร่วมกับ M1และ HALT ดังตาราง

ST	HALT	M1	OPERATION
0	1	0	CPU operation(1 st op-code fetch)
1	1	0	CPU operation(2 nd op-codeand 3 rd op-code fetch)
1	1	1	CPU operation(MC except for op-code fetch)
0	x	1	DMA operation
0	0	0	HALT mode
1	0	1	SLEEP mode (including SYSTEM STOP MODE)

NOTE X: Don't care

MC :Machine cycle

ตารางที่ 1 แสดงสถานะการทำงานของ CPU โดยร่วมกับ M1 และ HALT

TEND0-TEND1

TRANSFER END 0 และ 1 เป็นขาเอาต์พุต ACTIVE LOW ใช้แสดงถึงว่าทำ DMA สิ้นสุดลงแล้ว

TOUT

TIMER OUT ใช้กำเนิดพัลส์จาก PRT CHANEL 1

TXA0,TXA1

TRANSMIT DATA 0 และ 1 เป็นขาส่งข้อมูล SERIAL ของ ASCI

TXS

CLOCK SERIAL TRANSMIT DATA เป็นขาส่งข้อมูล SERIAL ของ CSIO

WAIT

ขาอินพุต ACTIVE LOW จะถูกตรวจที่ขอบขาลงของ CLOCK ลูกที่ 2 ของทุก ๆ MACHINE เพื่อเป็นการรอให้อุปกรณ์ภายนอกทำงานให้ ทันกับการทำงานของ CPU

WR

ใช้สำหรับการส่งข้อมูลไปยัง I/O หรือ MEMORY

X'TAL

เป็นขาที่ใช้ต่อกับ X'TAL

ขาที่ MULTIPLEX

A18/TOUT

ระหว่าง RESET จะเป็น A18 แต่ถ้ามีการเลือก SET BIT TOC1 หรือ TOC0 ใน TIMER CONTROL REGISTOR (TCR) ก็จะทำหน้าที่เป็น TOUT

CKA0/DREQ0

ระหว่าง RESET ขานี้จะเป็น CKA0 แต่ถ้า DM1 หรือ SM1 ใน DMA MODE REGISTOR (DMODE) ถูก SET เป็น 1 จะเป็นขา DREQ0

CKA1/TEND0

ระหว่าง RESET จะเป็นขา CKA1 แต่ถ้า BIT CKA1D ใน ASCI ถูก SET จะเป็นขา TEND0

RXS/CTS1

ระหว่าง RESET ขานี้จะเป็นขา RXS ถ้า BIT CTS1E ใน ASCI ถูก SET จะเป็นขา CTS1

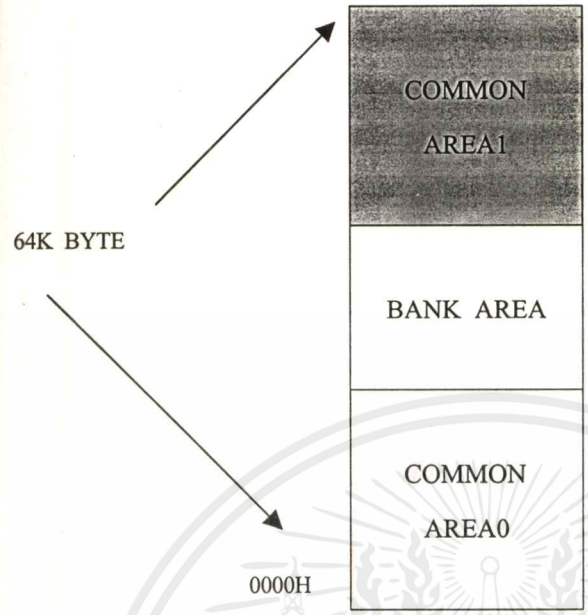
MEMORY MANAGEMENT UNIT (MMU)

เนื่องจาก Z80180 สามารถอ้างถึงหน่วยความจำได้ 1024KBYTE แต่ในชุดคำสั่งของZ80 นั้นไม่มีคำสั่งใดที่จะอ้างข้ามเกิน 64 KBYTE ได้และด้วยเป็นการที่ไม่ให้กระเทือนต่อผู้ที่เคยใช้ Z80 อยู่แล้วการเขียนโปรแกรมและการอ้างถึงหน่วยความจำก็ยังคงสภาพเดิมใน 64KBYTE แต่ในการปฏิบัติงานของ CPU Z80180 จริงๆที่จะกระทำหน่วยความจำทั้ง 1024KBYTE จะมี MMU มาเป็นผู้จัดการในการเข้าถึงตำแหน่งอันแท้จริง 00000H-FFFFFH ซึ่งในการปฏิบัติงานจะถูกแยกเป็น 2 แบบคือ

-ส่วนของ USER เรียกว่า LOGICAL ADDRESS (0000H-FFFFH) เป็นส่วนที่ถูกเรียกใช้ในโปรแกรมที่ผู้ใช้เขียนขึ้นมีขอบเขต 64KBYTE

-ส่วนของ CPU เรียกว่า PHYSICAL ADDRESS (00000H-FFFFFH) เป็นตำแหน่งที่ CPU ใช้ในการปฏิบัติงานจริง1024KBYTE

LOGICAL ADDRESS 64 KBYTE จะถูกแยกเป็น 3 ส่วนดังนี้ 64 K BYTE



รูปที่ 5 LOGICAL ADDRESS

COMMON AREA0,1 ส่วนนี้เมื่อเรียกตำแหน่งทาง LOGICAL ที่กำหนดไว้ไม่ว่าการทำงานของ CPU จะอยู่ใน PHYSICAL จริงที่ใดก็ตามจะกลับมายังตำแหน่งที่กำหนดเป็น COMMON นั้น นั่นก็คือ COMMON จะตาม CPU ไปทุกๆ ตำแหน่งที่กำลังปฏิบัติงานอยู่

BANK AREA จะมีลักษณะการทำงานเป็น PAGE เมื่อย้ายตำแหน่งที่เกิน PAGE ที่กำหนดไปยัง PAGE อื่นก็จะไม่สามารถติดต่อ PAGE ก่อนหน้านี้ได้

การกำหนดตำแหน่งในการใช้งานของ LOGICAL ADDRES

จะมี REGISTER ชื่อ COMMON/BANKAREAREGISTER (BAR:I/O ADDRESS= 3AH) ซึ่งใช้กำหนดตำแหน่งการเรียกใช้ในทางโปรแกรม (LOGICAL) ขนาด 1 BYTE โดยแบ่งออกเป็น 2 NIBBLES คือ

HIGH NIBBLE ใช้กำหนดค่าของ CA (COMMON AREA1(D7-D4)

LOW NIBBLE ใช้กำหนดค่าของ BA (BANK AREA)(D3-D0)

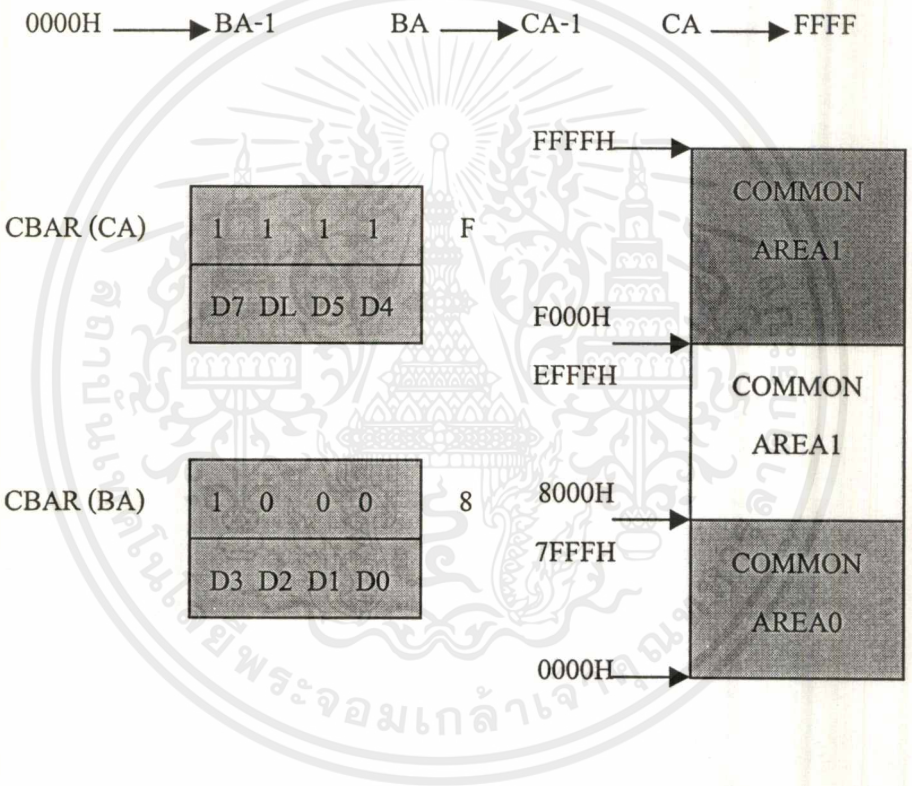
การมองค่าใน CBAR ค่าแต่ละ NIBBLE จะ X 000H จาก LOGICAL ที่ถูกแยกเป็น 3 ส่วนจะได้การกำหนดค่านี้

ตำแหน่ง 0000H ⟶ วิ่งไปชน BANK AREA (ค่าที่กำหนดใน BA-1)

ตำแหน่งใน BA ⟶ วิ่งไปชน COMMON AREA1 (ค่าที่กำหนดใน CA-1)

ตำแหน่งใน CA ⟶ วิ่งไปชนตำแหน่ง FFFFH

<u>ตัวอย่าง</u> กำหนดให้ CBAR=F8 ดังนั้น CA=F และ BA=8 ตำแหน่ง LOGICAL จะเป็นดังนี้



<u>COMMON AREA0</u> เมื่อถูกเรียกใช้ทางโปรแกรมอยู่ในขอบเขต 32K BYTE จาก ADDRESS 0000H-7FFFH

<u>BANK AREA</u> จะมีขอบเขตในการเรียกใช้ทาง LOGICAL 28K BYTE จาก ADDRESS 8000H-EFFFH

<u>COMMON AREA1</u> จะมีขอบเขตในการเรียกใช้ทาง LOGICAL 4 K BYTE จาก ADDRESS F000H-FFFFH

ในส่วนของ BANK AREA และ COMMON AREA1 สามารถกำหนดตำแหน่งการใช้งานจริงว่าให้อยู่ส่วนใดของหน่วยความจำขนาด 1024 K BYTE ได้จากค่า REGISTER ขนาด 8 BIT คือ BANK BASE REGISTER (BBR:ADDRESS 39H) และ COMMON BASE REGISTER (CBR:ADDRESS 38H) ตามลำดับโดย

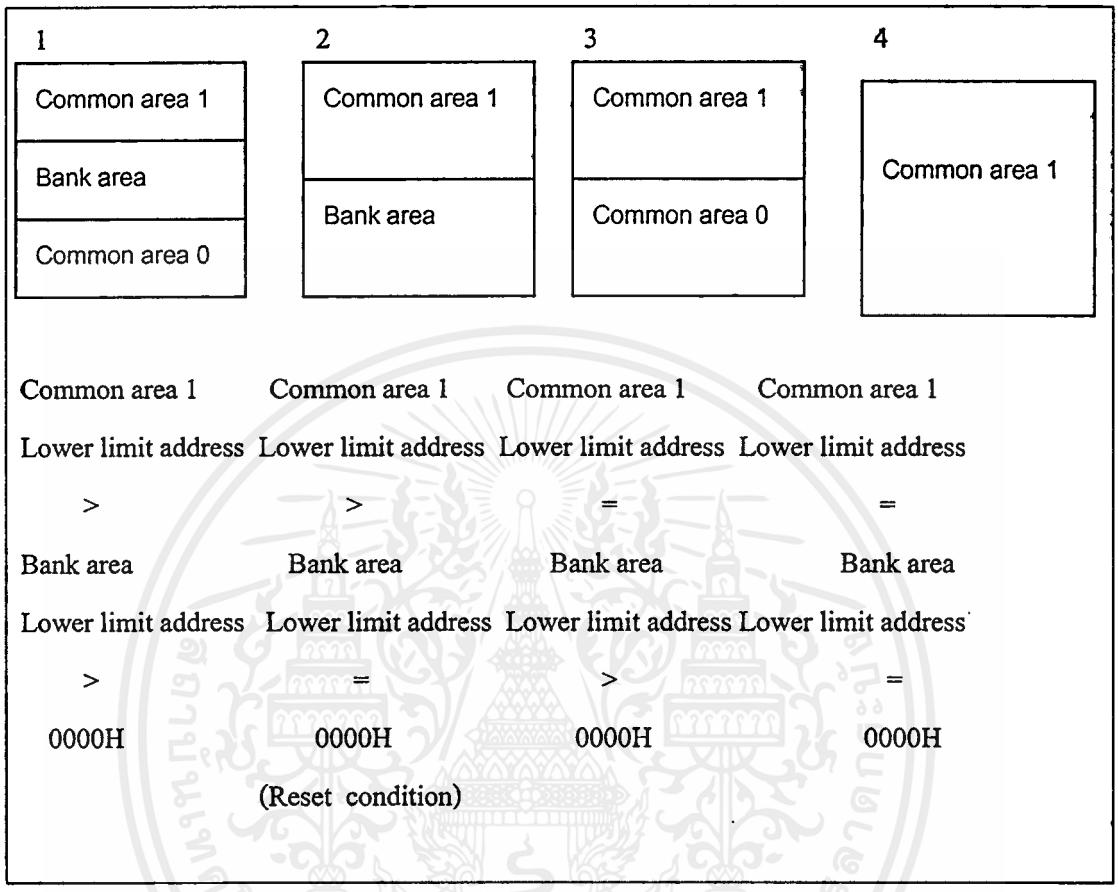
ตำแหน่งใน 1024 K BYTE (PHYSICAL ADDRESS)= BANK AREA หรือ COMMON AREA1+ (BBR หรือ CBR*1000H) จาก LOGICAL ที่กำหนดต้องการ BANK AREA อยู่ที่ตำแหน่ง 18000 และ COMMON AREA 1 อยู่ที่ตำแหน่ง 30000H ดังนั้น PHYSICAL ADDRESS เราทราบแล้วสิ่งที่ต้องการหาคือค่าใน BBR และ CBR จากสูตรด้านบนเปลี่ยนใหม่เป็น

$$BBR = \frac{18000H-8000H}{1000H} = 10H \qquad H \text{ เลขฐาน } 16$$

$$CBR = \frac{30000H-F000H}{1000H} = 21H$$

เช่นเมื่อมีการอ้างถึงตำแหน่ง 8000H ในโปรแกรมตำแหน่ง 18000H BANK AREA จะทำงานและถ้าอ้างถึงตำแหน่ง F007H จากโปรแกรมตำแหน่ง 30007H จะทำงานและถ้ามีการเปลี่ยนค่าใน BBR หรือ CBR ตำแหน่ง PHYSICAL ของ BANK AREA หรือ COMMON AREA1 จะเปลี่ยนไปในขณะการอ้างถึงทาง LOGICAL ยังคงเดิมเช่นเปลี่ยนค่า BBR จาก 10H เป็น 08H เมื่อมีการอ้างถึงตำแหน่ง 8000H ในโปรแกรม PHYSICAL ของ BANK AREA จะถูกทำงานที่ตำแหน่ง 10000H แทน

LOGICAL MEMORY สามารถจัดได้ใน 4 ลักษณะ



รูปที่ 6 Logical Memory organization

<u>แบบที่1</u> 0000H น้อยกว่า BA,BA น้อยกว่า CA ทำให้มี AREA ใช้งานแยกเป็น 3 ส่วนดังเช่น การให้ CBAR=F8H

0000H ⟶ BA-1	BA ⟶ CA-1	CA ⟶ FFFFH
0000H-7FFFH	8000H-EFFFH	F000H-FFFFH
(COMMON 0)	(BANK)	(COMMON 1)

<u>แบบที่2</u> 0000H=BA,BA น้อยกว่า CA ทำให้มี AREA ใช้งานเป็น 2 ส่วนดังเช่นการให้ CBAR=F0H (กรณีเกิดการ RESET)

0000H=BA⟶ CA-1	CA ⟶ FFFFH
0000H-EFFFH (BANK)	F000H-FFFFH (COMMON-1)

จะเห็นว่า COMMON AREA0 ไม่มีเนื่องจากการคิดจะเริ่มจากซ้ายไปขวา นั่นก็คือสิ่งที่อยู่ทางขวาจะถือเป็นค่าที่ใหม่กว่า เมื่อตำแหน่ง 0000H=BA แล้วก็จะเป็นค่าของ BA แทนในกรณีนี้เมื่อ RESET CPU จะเริ่มทำงานที่ ADDRESS 0000H ซึ่งเป็น MONITOR โปรแกรมถ้ามีคำสั่งย้าย BANK จะทำให้โปรแกรมทำงานไม่ได้เพราะ COMMON AREA0 ไม่มีแต่ถ้าต้องการย้าย BANK จำเป็นต้องกระทำการย้ายโปรแกรมไป RUN ในส่วน COMMON1 ก่อนแล้วจึงทำการย้าย BANK

แบบที่ 3 0000H น้อยกว่า BA,BA=CA AREA ใช้งานก็จะมี 2 ส่วน ดังเช่นการให้ CBAR=88H

0000H ⟶ BA-1 BA=CA ⟶ FFFFH

0000H-7FFFH (COMMON 0) 8000-FFFFH (COMMON 1)

ประโยชน์สามารถย้าย COMMON AREA1 เป็น BLOCK ๆ ได้ BLOCKจะ 32 K ทำให้ง่ายต่อการคำนวณตำแหน่งย้าย BLOCK และการเขียนโปรแกรมแต่ในส่วน COMMON AREA 0 จะต้องมีที่เก็บของ STACK ด้วย

แบบที่ 4 0000H=BA=CA AREA ใช้งานจึงมีเพียงส่วนเดียว ดังนั้น CBAR=00H

0000H=BA=CA ⟶ FFFFH

0000H-FFFFH (COMMON 1)

การ SETแบบนี้เป็นการใช้พื้นที่ส่วนเดียว 64 K ซึ่งจะทำการย้าย COMMON AREA1 ก็ไม่ได้ซึ่งจะเป็นการทำงานแบบ Z80 นั่นเอง คือ PHYSICAL จะเริ่มจาก 00000H-0FFFFH

ตัวอย่าง ต้องการให้

MONITOR PROGRAM อยู่ที่	0000H-7FFFH	ADDRESS จริง 0000H-7FFFH
RAM ใช้งานอยู่ที่	8000H-EFFFH	ADDRESS จริง 10000-16FFFH
RAM ที่เก็บ STACK	F000H-FFFFH	ADDRESS จริง 17000-17FFFH

จะเขียนโปรแกรมดังนี้

```
LD      A,0F8H      ; COMMON AREA 0
OUT0    (CBAR),A    ; MONITOR ADDRESS 0000-7FFFH NOTREMOVE
LD      A,08H
OUT0    (BBR),A     ; BANK AREA 10000H CAN REMOVE
OUT0    (CBR),A     ; COMMON AREA 1 10000H NOT REMOVE
```

สรุป

1. เมื่อ RESET ค่าใน CBAR=F0H
2. จะต้องกำหนด LOGICAL ADDRESS ใน REGISTER CBAR(3AH) ก่อน
3. PHYSICAL = LOGICAL + (BBR(39H) หรือ CBR(38H)*1000H)
4. ในทางกลับกันรู้ PHYSIAL ต้องการหาค่าของ BBR หรือ CBR

   BBR หรือ CBR = $\frac{\text{PHYSICAL-LOCICAL}}{1000H}$

5. เมื่อต้องการย้าย BANK หรือ COMMON โปรแกรมในขณะ RUN นั้นจะต้องอยู่ในส่วนของ COMMON

2.3 ลักษณะทั่วไปของบอร์ด CP – JR180 V 2.0

บอร์ด CP –JR180 V2.0 เป็นบอร์ดไมโครคอนโครลเลอร์ ที่ถูกออกมาและพัฒนาต่อเนื่องมาจากบอร์ด CP – JR180 V1.0 โดยได้ทำการปรับปรุงและเพิ่มเติมอุปกรณ์ต่าง ๆ เพื่อสนับสนุนและส่งเสริมประสิทธิภาพให้มีความอ่อนตัวมากขึ้น

2.4 คุณสมบัติของบอร์ด CP – JR180 V2.0

- CPU เบอร์ Z80180 CMOS/RUN ความถี่ 6.144 MHz
- 32KB/64KB EPROM MONITOR(27256/27512)
- 32KB RAM ON BOARD(62256)
- 512KB ROM/RAM EXPANSION
  (RAM 62256/628128/628512/EPROM27512/271001)
- 20 PIN LCD CONNECTOR(Graphic & Character Dotmatrix)
- 10 PIN KEYBOARD CONNECTOR
- 10 PIN SPD8 CONNECTOR

- RTC6242 REAL TIME CLOCK
- UART RS-232x2 CHANNEL(RS422/RS485 OPTION)
- SERIAL IO A/D 12 BITx2 CHANNEL(LTC1298 OPTION)
- SERIAL IO EEPROM 1KB(9346)/2KB(9356)/4KB(9366)
- POWER ON RESET + WATCHDOG + BACKUP(MAX691)
- 48 BIT I/O 8255 PORT 34 PIN (72IOZ80)x2
- 7805 REGULATOR POWER SUPPLY ON BOARD (9VDC)
- PCB SIZE 10x16 cm
- MINI SPEAKER

หน่วยประมวลผล หรือ CPU บอร์ด CP-JR180 V2.0 จะใช้ CPU ของ ZILOG เบอร์ Z80180 เป็น หน่วยประมวลผลหลักของระบบ ทั้งนี้เนื่องจาก CPU เบอร์ Z80180 จัดว่าเป็น CPU ที่มีความสามารถสูงตัวหนึ่งที่ได้พัฒนาปรับปรุงมาจาก CPU ยอดนิยม เบอร์ Z80 ซึ่งชุดคำสั่งทั้งหมดยังคงเป็นของ Z80 อยู่เหมือนเดิม เพียงแต่ได้มีการเพิ่มเติมชุดคำสั่งและความสามารถเพิ่มเติมอื่น ๆ ให้สามารถใช้งานได้กว้างขวางและสะดวกมากขึ้น โดยได้รวมเอาส่วนประกอบที่สำคัญ ๆ หลายอย่างไว้ภายในตัว CPU อย่างครบถ้วน ซึ่งเมื่อพิจารณาดูแล้วนับว่ามีความเหมาะสมกับการใช้งานในด้านระบบการควบคุมต่าง ๆ เป็นอย่างมาก ไม่ว่าจะเป็นระบบการควบคุมในโรงงานอุตสาหกรรม หรือระบบการสื่อสารข้อมูล เพราะ CPU เบอร์ Z80180 สามารถอ้างหน่วยความจำได้สูงถึง 1MB มีระบบการทำ DMA ถึง 2 วงจร มีพอร์ตสื่อสารแบบอนุกรม UART 2 ชุด ซึ่งสามารถทำการสื่อสารแบบ Multi Process Communication ได้โดยตรงและยังมีระบบฐานเวลา Timer/Counter และอินเตอร์รัพท์ต่าง ๆ อีก

หน่วยความจำ บอร์ด CP-JR180 V2.0 สามารถเชื่อมต่อกับหน่วยความจำทั้งแบบ ROM และ RAM ได้ Page ละ 64KB จำนวน 6 Page โดยวงจรในส่วนของการ Decode ตำแหน่งของหน่วยความจำนั้น เราจะเลือกใช้การ Decode ออกเป็น 6 Page โดยมีขนาด Page ละ 64KB โดยเราสามารถจัดแบ่งหน่วยความจำออกเป็นดังนี้คือ

"Socket U2" เป็นส่วนของหน่วยความจำโปรแกรม สำหรับเก็บค่ารหัสคำสั่งของโปรแกรมมอนิเตอร์ของระบบ โดยในส่วนนี้ต้องใช้กับหน่วยความจำถาวรแบบ ROM หรือ EPROM ซึ่งสามารถเลือกใช้ขนาดของหน่วยความจำในส่วนนี้ได้ 2 ขนาด คือ 32KB (27256)

และ 64KB (27512) โดยการเลือกกำหนดจาก Jumper JP1 โดยหน่วยความจำในส่วนนี้มีตำแหน่งการทำงานอยู่ระหว่าง 00000H-0FFFFH (Physical Address)

"Socket U3" เป็นส่วนของหน่วยความจำข้อมูล ใช้สำหรับเก็บข้อมูลหรือค่าต่าง ๆ ของการประมวลผลข้อมูลรวมทั้งใช้เป็นพื้นที่ STACK ของระบบ โดยกำหนดเป็นหน่วยความจำชั่วคราวแบบ RAM ขนาด 32KB (62256) มีตำแหน่งการทำงานอยู่ที่ 10000H-1FFFFH (Physical Address) ดังที่กล่าวมาแล้วว่าวงจรของการ Decode หน่วยความจำนั้น จะทำการ Decode ไว้ช่วงละ 64KB ดังนั้นจึงส่งผลให้หน่วยความจำในส่วนนี้ ซึ่งมีขนาด 32KB จึงเกิดปรากฏการณ์ของ "แอดเดรสเงา" หรือ การทับซ้อนตำแหน่งกันอยู่ ซึ่งไม่ว่าเราจะอ้างตำแหน่งของหน่วยความจำไปยังตำแหน่ง 32KB ส่วนล่าง หรือ 32KB ในส่วนบน เราก็จะมองเห็นเป็นตำแหน่งเดียวกันเสมอ กล่าวคือ เราจะมองเห็นหน่วยความจำ RAM นี้เป็น 2 ช่วง ช่วงละ 32KB คือ

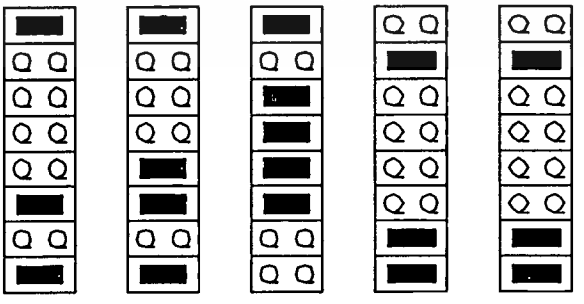
10000H-17FFFH (32KB ล่าง)

18000H-1FFFFH (32KB บน)

ดังนั้นจะเห็นได้ว่า ไม่ว่าเราจะทำการสั่งให้ CPU อ้างถึงตำแหน่งของหน่วยความจำเป็น 10000H หรือ 18000H ก็จะเป็นการติดต่อกับหน่วยความจำที่ตำแหน่งเดียวกันเสมอ

"Socket U4" ในส่วนนี้จะเป็นส่วนของหน่วยความจำ ที่ใช้ขยายเพิ่มเติม ซึ่งสามารถใส่ได้ทั้งหน่วยความจำแบบ RAM และ EPROM โดยถ้าเป็นหน่วยความจำแบบ RAM นั้นเราสามารถทำการ Backup ข้อมูลไว้ได้ด้วย โดยถ้าต้องการ Backup ข้อมูลในหน่วยความจำส่วนนี้ไว้ให้เลือก Jumper JP3 (VCC/VBAT) ไปยังตำแหน่ง VBAT แต่ถ้าไม่ต้องการ Backup ข้อมูลหรือใช้กับหน่วยความจำแบบ EPORM ต้องทำการเลือกตำแหน่ง Jumper JP3 ไว้ที่ตำแหน่ง VCC เสมอ

สำหรับการเลือกขนาดและชนิดของหน่วยความจำนั้น เราสามารถทำการเลือกได้โดย Jumper JP2 ซึ่งมีดังนี้



62256 6281282 6285122 27512 271001

รูปที่ 7 แสดง MEMORY MAP ของบอร์ด CP-JR180 V2.0

EEPROM สำหรับหน่วยความจำในส่วนนี้จะไม่สัมพันธ์เกี่ยวข้องกับตำแหน่งของหน่วยความจำที่ Decode ไว้ในระบบแต่อย่างใด ทั้งนี้เพราะเราใช้ หน่วยความจำแบบ EEPROM ชนิด Serial-I/O ซึ่งจะต้องใช้สัญญาณจากระบบ Clock Serial I/O Port ของ CPU ในการอ่านและเขียนข้อมูลกับ EEPROM ซึ่งในส่วนของหน่วยความจำแบบ EEPROM นี้สามารถเลือกใส่ EEPROM ได้ทั้งหมด 3 ขนาด คือ 1KB/2KB/4KB แต่เนื่องจาก EEPROM ที่มีจำหน่ายกันอยู่ทั่วไปตามท้องตลาดนั้น ในปัจจุบันจะมีอยู่มากมายหลายยี่ห้อ ซึ่งในบางยี่ห้อก็ใช้วิธีการอ่านเขียนข้อมูลที่แตกต่างกันไปถึงแม้ว่าจะมีขนาดความจุของหน่วยความจำที่เท่ากันก็ตามที กล่าวคือ ในบางยี่ห้อใช้วิธีการอ่านเขียนข้อมูลครั้งละ 8 บิต บางยี่ห้อใช้วิธีการอ่านเขียนข้อมูลแบบ 16 บิต หรือบางยี่ห้อก็สามารถทำการอ่านเขียนข้อมูลได้ทั้งแบบ 8 บิตและ 16 บิต โดยการเลือกจากขาสัญญาณควบคุม (ORG) ก็ได้ดังตัวอย่างในตาราง

EEPRON เบอร์	อ่าน/เขียนข้อมูล แบบ 8 บิต	อ่าน/เขียนข้อมูล แบบ 16 บิต	ขนาด ความจุ
93xx46	128ไบท์ x 8 บิต	64ไบท์ x 16 บิต	1 KB
93xx56	256ไบท์ x 8 บิต	128ไบท์ x 16 บิต	2 KB
93xx66	512ไบท์ x 8 บิต	256ไบท์ x 16 บิต	4 KB

ตารางที่ 2 แสดงการอ่านเขียนข้อมูลของ EEPROM

ซึ่งโดยทั่วๆ ไป EEPROM ที่สามารถเลือกวิธีการอ่านและเขียน ข้อมูลได้ทั้งแบบ 8 บิต และ 16 บิต จะใช้ขาสัญญาณ ORG (ขา 16) เป็นตัวกำหนด โดยการเลือก Logic ให้กับขาสัญญาณนี้โดย

- ถ้าขา ORG ได้รับ Logic" 0" จะเป็นการอ่าน/เขียนแบบ 8 บิต
- ถ้าขา ORG ได้รับ Logic" 1" จะเป็นการอ่าน/เขียนแบบ 16 บิต

สำหรับบอร์ด CP-JR180 V2.0 นี้ ในส่วนวงจรของการเชื่อมต่อกับ EEPROM จะมี Jumper อยู่ 2 ชุด คือ Jumper JP5 ซึ่งใช้สำหรับเลือกสัญญาณ CS# ให้กับ EEPROM ว่าจะใช้สัญญาณ RTS0 จาก CPU หรือ PB0 จากพอร์ต B ของ 8255 มาควบคุมการทำงานของสัญญาณ CS# ของ EEPROM และใช้ Jumper JP6 สำหรับเลือกกำหนดวิธีการอ่านและเขียนข้อมูลกับ EEPROM ว่าจะใช้แบบ 8 บิต หรือ 16 บิต แต่ถ้าหากเราใช้กับ EEPROM รุ่นที่มีวิธีการอ่านเขียนข้อมูลแบบเดียวตายตัวนั้นการเลือก Jumper JP6 นี้ก็จะไม่มีผลต่อการทำงานของ EEPROM แต่อย่างใด เพราะไม่ว่าจะเลือกไว้ตำแหน่งใด เพราะไม่ว่าจะเลือกไว้ตำแหน่งใด

การอ่าน/เขียนข้อมูล ก็ยังคงเหมือนเดิม ซึ่งการจะทราบได้ว่า EEPROM เบอร์ใด ยี่ห้อใด มีวิธีการอ่านและเขียนใดนั้น ต้องดูจากรหัสและเบอร์ที่พิมพ์ติดมากับตัวของ EEPROM นั้น ๆ โดยการเทียบจาก Data Sheet ของ EEPROM ในแต่ระยี่ห้อก็จะทราบได้

<u>วงจร A/D</u> สำหรับการทำงานของวงจร A/D นี้ เราจะใช้ A/D ขนาด 12 บิต 2 Channel ชนิด Serial I/O เบอร์ LTC1298 ซึ่งใช้วิธีการอ่านเขียนข้อมูลแบบอนุกรม ในลักษณะที่คล้ายกับ EEPROM แต่สัญญาณในการอ่านเขียนจะใช้พอร์ต B ของ 8255 และสัญญาณ DCD0 ของ CPU แทน ซึ่งเราสามารถต่อสัญญาณ Analog ขนาด 0-5 โวลท์ เข้ามาให้วงจร A/D นี้ได้โดยตรง ถึง 2 ชุด

<u>วงจรฐานเวลานาฬิกา</u> RTC6242 สำหรับวงจรในส่วนของฐานเวลานาฬิกานี้เราจะใช้ IC RTC6242 ซึ่งเป็น Real Time Clock/Calender ในตัวเดียวกัน ซึ่งจะทำให้เราสามารถอ่านเขียนค่าเวลา วินาที นาที ชั่วโมง วัน เดือน ปี วันในรอบสัปดาห์ และยังมีส่วนของวันในปีอธิกสุรทินอีกด้วย กล่าวคือสามารถตรวจสอบและปรับเวลาของปีที่มี 28 หรือ 29 วัน ได้เองโดยอัตโนมัติ ซึ่งการเชื่อมต่อของวงจรในส่วนนี้จะใช้วิธีการ Decode ตำแหน่งไว้ เราจึงสามารถทำการอ่านเขียนข้อมูลกับ RTC ผ่านทางพอร์ตได้เลย ซึ่ง RTC นี้จะใช้ตำแหน่งพอร์ตในการอ่านเขียนข้อมูลกับส่วนต่าง ๆ ไม่ว่าจะเป็น เวลา วัน เดือน หรือ ปี ซึ่งมีด้วยกันทั้งหมด 16 ตำแหน่งด้วยกัน โดยมีตำแหน่งการทำงานอยู่ที่ A0H-AFH ตามลำดับดังแสดงในตาราง

ตำแหน่ง Adress	ชื่อ รีจีสเตอร์	ค่าของบิตข้อมูล D3	D2	D1	D0	ค่า นับ	หน้าที่และ การทำงาน
A0H	Sec1	S8	S4	S2	S1	0-9	ค่าวินาทีหลักหน่วย
A1H	Sec10	—	S40	S20	S10	0-5	ค่าวินาทีหลักสิบ
A2H	Min1	Mi8	mi4	mi2	mi1	0-9	ค่าวินาทีหลักหน่วย
A3H	Min10	—	mi40	mi20	mi10	0-5	ค่าวินาทีหลักสิบ
A4H	Hour1	h1	h4	h2	h1	0-9	ค่าชั่วโมงหลักหน่วย
A5H	Hour10	—	PM/ AM	h20	h10	0-2 0-1	ค่าชั่วโมงหลักสิบ & สถานะ AM/PM
A6H	Date1	d8	d4	d2	d1	0-9	ค่าวันหลักหน่วย
A7H	Date10	—	—	d20	d10	0-3	ค่าวันหลักสิบ
A8H	Mon1	mo8	mo4	mo2	mo1	0-9	ค่าเดือนหลักหน่วย
A9H	Mon10	—	—	—	mo10	0-1	ค่าเดือนหลักสิบ
AAH	Year1	y8	y4	y2	y1	0-9	ค่าปีหลักหน่วย
ABH	Year10	y80	y40	y20	y10	0-9	ค่าปีหลักสิบ
ACH	Week		w4	w2	w1	0-6	ค่าสัปดาห์
ADH	Control Reg. D	30 sec ADJ	IRQ FLAG	BUSY	HOLD	—	รีจิสเตอร์ควบคุม D
AEH	Control Reg. E	t1	t0	ITRPT /STND	MASK	—	รีจิสเตอร์ควบคุม E
AFH	Control Reg. F	TEST	24/12.	STOP	RESET	—	รีจิสเตอร์ควบคุม F

หมายเหตุ ITRPT/STND = Interrupt/Standard

ตารางที่ 3 แสดงตำแหน่งและหน้าที่การทำงานของ RTC6242

จอแสดงผล LCD สำหรับส่วนของการแสดงผล แบบ LCD นี้เราสามารถเลือกใช้จอแสดงผล LCD ได้ 2 ชนิด คือ ทั้งแบบ รูปภาพ (Graphic) และแบบอักขระตัวอักษร (character

Dot-Matrix) โดยมี Jumper jp7 เป็นตัวเลือกว่าจะต่อกับ LCD ชนิดใด และใช้ตัวต้านทานแบบปรับค่าได้แบบเกือกม้าทำหน้าที่เป็นตัวปรับระดับความสว่างของหน้าจอ LCD โดยในการต่อใช้งาน LCD แต่ละชนิดจะใช้สัญญาณในการเชื่อมต่อไม่เท่ากัน คือ ถ้าเป็นจอ LCD ชนิดตัวอักษร Character Dot-Matrix จะใช้สัญญาณ 14 เส้น แต่ถ้าเป็นจอ LCD ชนิดรูปภาพ Graphic ต้องใช้สัญญาณทั้งหมด 18 เส้น แต่ขั้วต่อของ LCD ที่ออกแบบไว้ในบอร์ด CP-JR180 V2.0 นี้จะเป็นขนาด 20 ขา เพื่อให้สามารถติดต่อกับ LCD ได้ทั้ง 2 แบบดังนั้นถ้าต้องการใช้กับ LCD ชนิดตัวอักษร Character Dot-Matrix ก็ให้ต่อสายสัญญาณเพียงแค่ 14 เส้นแรกเท่านั้น ส่วนสัญญาณที่เกินมาก็ไม่ต้องสนใจ ให้ปล่อยว่างไว้

หน้าที่การทำงานของ LCD	ตำแหน่งพอร์ต Adress	
	Character หรือ Page1 Graphic	เฉพาะ Page2 Graphic เท่านั้น
Write Instruction	C0H	C1H
Write Data to CG or DD RAM	C2H	C3H
Read Busy Flag and Adress	C4H	C5H

ตารางที่ 4 แสดงหน้าที่การทำงานและตำแหน่งพอร์ต ADDRESS ของ LCD

<u>จอแสดงผล 7-Segment</u> ในกรณีที่ต้องการแสดงผลตัวเลขด้วยจอแสดงผลแบบ 7-Segment นั้นทางทีมงาน อีทีที ได้ออกแบบชุดแสดงผลแบบ 7-Segment ขนาด 8 หลัก ในรุ่น " ET-SDP8 " ไว้สนับสนุนการทำงานในส่วนนี้อยู่แล้ว ซึ่งชุดแสดงผลรุ่น ET-SDP8 นี้จะก่อให้เกิดความสะดวกต่อผู้ใช้เป็นอย่างมากในการทำงานและเขียนโปรแกรมควบคุมเพราะไม่จำเป็นต้องเขียนโปรแกรมเพื่อ SCAN หน้าจออยู่ตลอดเวลาเพียงแต่ส่งค่ารหัสของตัวเลขที่ต้องการจะแสดงผลไปให้วงจร ET-SDP8 หลังจากนั้นวงจร ET-SDP8 ก็จะจัดการเรื่อง SCAN หน้าจอเองทั้งหมด เมื่อต้องการเปลี่ยนค่าการแสดงผลที่หน้าจอใหม่ก็ส่งรหัสตัวเลขสำหรับแสดงผลชุดใหม่ไปอีก ทำให้ CPU มีเวลาว่างเพื่อไปประมวลผลอย่างอื่นมากขึ้น ไม่ต้องคอยวนรอบกลับมา SCAN หน้าจออยู่บ่อย ๆ ตลอดเวลา และถ้าหาก CPU ต้องทำงานหลาย ๆ อย่างพร้อมกับการ SCAN หน้าจอในเวลาเดียวกันด้วยแล้วอาจเกิดความไม่สม่ำเสมอหรือการกระตุกของหน้าจอแสดงผลและเขียนโปรแกรมได้ยากลำบากมากขึ้นอีกด้วย ซึ่งในส่วนของบอร์ด CP- JR180 V2.0 นี้ก็ได้จัดวงจรใน

ส่วนของการเชื่อมต่อกับบอร์ดแสดงผล 7-Segment รุ่น "ET-STD8" นี้ ทางขั้วต่อขนาด 10 PIN ได้โดยตรงโดยไม่ต้องเพิ่มเติมหรือดัดแปลงวงจรให้ยุ่งยากเสียเวลา

วงจร PowerOn Reset,WatchDog และ BackUp สำหรับวงจรในส่วนนี้จะใช้ IC เบอร์ MAX691 ทำหน้ามี่ทั้งหมด โดยในส่วนของการรีเซ็ตนั้นจะสามารถเลือกได้ว่าจะใช้วงจร WatchDog เข้ามาควบคุมการรีเซ็ตด้วยหรือไม่ โดยเลือกกำหนดจาก Jumper JP4 (CLK/WD)

-ถ้าต้องการใช้ WatchDog ให้เลือก JP4 = WD

-ถ้าไม่ต้องการใช้ WatchDog ให้เลือก JP4 = CLK

ซึ่งลักษณะการทำงานของวงจร WatchDog นั้นพอจะอธิบายได้ดังนี้คือ วงจร WatchDog จะเป็นวงจรนับประเภทหนึ่งซึ่งมีคาบเวลาของการนับที่แน่นอนตามที่เรากำหนดไว้ โดยภายในคาบเวลาที่เรากำหนดไว้นั้นหากไม่ส่งสัญญาณไปรีเซ็ตการนับของ WatchDog ก็จะทำให้ WatchDog ส่งสัญญาณ RESET ออกมาในระบบ ซึ่งประโยชน์ของ WatchDog ในระบบไมโครโปรเซสเซอร์นั้น จะใช้ WatchDog เป็นตัวเฝ้าติดตามการทำงานของ CPU ให้ทำงานตามข้อกำหนดของโปรแกรม ที่เราเขียนไว้เท่านั้น โดยการแทรกโปรแกรมสำหรับรีเซ็ต WatchDog ไว้ในส่วนต่างของโปรแกรมตามความเหมาะสม ถ้าหากว่าเมื่อใดก็ตามที่ CPU เกิดหยุดการทำงานลงอย่างกระทันหัน หรือทำงานผิดพลาด หรือกระโดดไปทำงานในตำแหน่งที่ไม่ได้กำหนดไว้ในเงื่อนไขแล้ว CPU ก็จะไม่สามารถส่งสัญญาณไปรีเซ็ตการนับของ WatchDog ได้ทันตามกำหนดเวลา ซึ่ง WatchDog ก็จะส่งสัญญาณ RESET เพื่อบังคับให้ CPU กลับมาเริ่มต้นทำงานใหม่ได้อีกครั้งหนึ่งอย่างถูกต้อง

โดยสัญญาณ ที่ใช้สำหรับการรีเซ็ตค่าของ WatchDog นี้จะใช้สัญญาณจากการ Decode พอร์ต เช่นเดียวกับอุปกรณ์อินพุตเอาต์พุตอื่น ๆ ของบอร์ด ซึ่งวิธีการรีเซ็ต WatchDog นั้นสามารถทำได้โดยการสั่งให้สัญญาณที่ Decode ไว้ทำงาน โดยอาจเป็นการอ่านหรือเขียนค่ากับตำแหน่ง Decode ของ WatchDog ก็ได้ ซึ่งค่าของข้อมูลที่จะเขียนหรืออ่านนั้นจะเป็นเท่าไรก็ได้ไม่ต้องสนใจ เพราะเราเพียงต้องการสัญญาณมารีเซ็ต WatchDog เท่านั้นไม่สนใจค่าของข้อมูลว่าจะเป็นอะไร ตำแหน่งที่เราต้องเขียนข้อมูลเข้าไปเพื่อให้ CPU ส่งสัญญาณมารีเซ็ต WatchDog มีตำแหน่งอยู่ที่ E0H สำหรับตัวอย่างของโปรแกรม การรีเซ็ต WatchDog นั้นสามารถแสดงให้เห็นได้ดังตัวอย่างต่อไปนี้

```
WATCHDOG:       OUT    ( 0E0H ),A          ;Active Port E0H
                RET
```

<u>คีย์บอร์ด</u> ขั้วต่อคีย์บอร์ดของบอร์ด CP-JR180 V2.0 นี้ เป็นขั้วต่อคอนเน็คเตอร์ขนาด 10 PIN โดยสัญญาณจะต่อมาจากพอร์ต C ของ 8255 ซึ่งสามารถต่อใช้งานได้กับคีย์บอร์ดขนาด 16 คีย์ (Matrix KEY 4x4)

<u>I/O PORT</u> อุปกรณ์อินพุตเอาต์พุตของบอร์ด CP-JR180 V2.0 นี้จะใช้ Chips Support เบอร์ 8255 จำนวน 2 ตัว ซึ่งขั้วต่อของพอร์ตจะเป็นขั้วต่อมาตรฐานอีทีที (72-IOZ80) สามารถเชื่อมต่อกับอุปกรณ์อินพุตเอาต์พุตของอีทีทีได้ทันที โดย 8255 นี้จะมีตำแหน่งพอร์ตใช้งานดังนี้ คือ

8255 System (72IOZ80-1)		8255 System (72IOZ80-1)	
Port A	60H	Port A	80H
Port B	61H	Port B	81H
Port C	62H	Port C	82H
Port Control	63H	Port Control	83H

ตารางที่ 5 แสดงตำแหน่งพอร์ตการใช้งานของ 8255

ซึ่งพอร์ต 8255 นี้จะแบ่งออกเป็น 2 ส่วน คือ 8255 System เป็นพอร์ต 8255 ซึ่งสัญญาณบางส่วนจาก 8255 นี้จะถูกนำไปใช้ในระบบ เช่น ควบคุมการอ่านเขียนของวงจร EEPROM วงจร A/D วงจรแสดงผล SDP8 และคีย์บอร์ดกับลำโพง เป็นต้น โดยถ้าหากว่าผู้ใช้ไม่ได้ใช้งานอุปกรณ์ดังที่กล่าวมาแล้วข้างต้นก็สามารถนำสัญญาณของ 8255 System นี้ไปใช้งานอื่นตามความต้องการได้ ซึ่งถ้าหากมีความต้องการจะนำสัญญาณจาก 8255 System นี้ไปใช้งานภายนอกแล้วจะต้องไม่ใส่อุปกรณ์ที่กล่าวมาแล้วข้างต้นในบอร์ด เพราะจะทำให้สัญญาณถูกชนกันได้โดย พอร์ต 8255 System นี้จะมีขั้วต่อของสัญญาณเป็นแบบ 34 PIN (72IOZ80-1)

ชื่อสัญญาณ	สัญญาณการเชื่อมต่อ	หน้าที่
PA0-PA7	ว่างสำหรับใช้งานตามต้องการอย่างอิสระ	—
PB0	สัญญาณ CS# ของ EEPROM	OUTPUT
PB1	สัญญาณ CLK ของ A/D (LTC1298)	OUTPUT
PB2	สัญญาณ DI ของ A/D (LTC1298)	OUTPUT
PB3	สัญญาณ CS# ของ A/D (LTC1298)	OUTPUT
PB4	สัญญาณ CLK ของชุดแสดงผล SDP8	OUTPUT
PB5	สัญญาณ DIN ของชุดแสดงผล ADP8	OUTPUT
PB6	สัญญาณ LOAD ของชุดแสดงผล SDP8	OUTPUT
PB7	สัญญาณขับลำโพงเล็กเพื่อกำเนิดเสียง	OUTPUT
PC0-PC7	ใช้สำหรับ Scan Keyborad ขนาด 4x4	IN & OUTPUT

ตารางที่ 6 แสดงหน้าที่ของสัญญาณ 8255 System

ส่วนพอร์ต 8255 User นั้นสัญญาณทุกเส้นจาก 8255 ในส่วนนี้จะว่างไว้สำหรับผู้ใช้ทั้งหมด ผู้ใช้สามารถต่อไปใช้งานได้อย่างอิสระ โดยมีขั้วต่อสัญญาณเป็นแบบ 34 PIN(72IOZ80-2)

<u>พอร์ตอนุกรม UART</u> สำหรับวงจร ในส่วนของการรับข้อมูลแบบอนุกรม UART นั้น ในบอร์ด CP-JR180 V2.0 จะมีอยู่ 2 Channel โดย ในวงจรของ Channel0 (RS232-1) จะใช้การติดต่อสื่อสารแบบ RS232 ได้อย่างเดียวซึ่งใน Channel0 นี้สามารถรับส่งข้อมูลแบบ Full Duplex คือสามารถทำการรับส่งข้อมูลพร้อมกันได้ทั้งสองทิศทาง โดยมีขั้วต่อของสัญญาณสำหรับทำการรับส่งข้อมูลกันเป็นแบบ CPA-4PIN (RS232-1)

สำหรับในส่วนของวงจร Channel1 (RS232-2) จะสามารถเลือกใช้วงจร Line Driver ได้หลายลักษณะไม่ว่าจะเป็น RS232 หรือ RS422 หรือ RS485 ก็ได้ตามความเหมาะสมและความต้องการของผู้ใช้ โดยถ้าต้องการวิธีการรับส่งข้อมูลแบบ RS232 ชนิด Full Duplex ต้องต่อสายสัญญาณสำหรับรับส่งข้อมูลกับขั้วต่อแบบ CPA-4PIN (RS232-2) และต้องเลือก Jumper JP8 (MAX/75176) ไว้ยังตำแหน่ง MAX เพื่อเลือกวงจร Line Driver เป็นเบอร์ MAX232 สำหรับรับส่งข้อมูลแบบ RS232 นั่นเอง

ซึ่งในการรับส่งข้อมูลแบบ RS232 นั้นจะมีข้อจำกัดในเรื่องของระยะทางอยู่มาก คือสามารถรับส่งกันได้ไกลสุดประมาณ 50 ฟุต เท่านั้น ถ้าระยะทางไกลกว่านี้จะเกิดความผิดพลาดสูง

หรือรับส่งกันไม่ได้เลย ถ้าผู้ใช้มีความต้องการที่จะรับส่งข้อมูลให้ได้ระยะทางที่ไกลขึ้นกว่า 50 ฟุต แล้วต้องเปลี่ยนวงจร Line Driver ให้เป็นแบบ RS422 ซึ่งจะทำให้สามารถรับส่งข้อมูลกันได้ในระยะทางที่ไกลขึ้นกว่าแบบ RS232 มาก กล่าวคือ การรับส่งข้อมูลแบบ RS422 สามารถทำการรับส่งข้อมูลกันได้ไกลสุดถึง 4,000 ฟุต หรือ ประมาณ 1.2 กิโลเมตร เลยทีเดียว ซึ่งการเปลี่ยนวงจรการรับส่งแบบ RS232 มาเป็นแบบ RS422 นี้ยังจำไม่เป็นต้องเปลี่ยนแปลงโปรแกรมสำหรับควบคุมการรับส่งข้อมูลใหม่แต่อย่างใด ยังคงใช้โปรแกรมเดียวกับการรับส่งแบบ RS232 อยู่เช่นเดิม เพียงแต่ว่าทั้งฝ่ายรับและฝ่ายส่งต้องเปลี่ยนวงจร Line Driver ของวงจรการรับส่งให้เป็นแบบ RS422 (ใช้ Line Driver เบอร์ 75176) ให้เหมือนกันทั้งคู่เท่านั้นก็สามารถรับส่งข้อมูลกันได้แล้ว ซึ่งในบอร์ด CP-JR180 V2.0 นี้สามารถเปลี่ยนได้โดยการนำไอซี Line Driver เบอร์ 75176 มาใส่ลงในบอร์ดแล้วเลือก Jumper ตำแหน่ง JP8 (MAX/75176) มาไว้ที่ตำแหน่ง 75176 และเลือก Jumper ตำแหน่ง JP9 (422/485) มาไว้ในตำแหน่ง 422 เท่านั้น ส่วนสายสัญญาณในการรับส่งก็ให้ต่อเข้าที่ขั้วต่อแบบ CPA-6PIN (422/485)

แต่ในกรณีที่ต้องการต่อวงจรหลาย ๆ ชุดเข้าด้วยกันเพื่อทำการรับส่งข้อมูลซึ่งกันและกัน (Multi Processor Communication System) วงจรการรับส่งแบบ RS232 และแบบ RS422 ไม่สามารถใช้ได้ในกรณีนี้เพราะจะเกิดการชนกันของข้อมูลในสายทำให้เกิดการผิดพลาดและรับส่งกันไม่รู้เรื่อง ดังนั้นจึงต้องหลีกเลี่ยงมาใช้วงจรการรับส่งแบบ RS485 แทนโดยในการรับส่งข้อมูลแบบ RS485 นี้จะสามารถรับส่งข้อมูลกันได้ในระยะทางไกลสุดถึง 4,000 ฟุต หรือประมาณ 1.2 กิโลเมตร เช่นเดียวกับแบบ RS422 แต่ไม่สามารถทำการรับและส่งข้อมูลพร้อมกันในเวลาเดียวกันได้ ต้องใช้วิธีการผลัดกันรับและผลัดกันส่งทีละครั้ง หรือเรียกว่า “Half Duplex” ซึ่งในการรับส่งข้อมูลแบบนี้จะมีข้อดีคือสามารถเชื่อมต่อระบบไมโครโปรเซสเซอร์หลาย ๆ ตัว เข้าด้วยกันได้ แต่มีข้อเสียคือต้องมีโปรแกรมสำหรับควบคุมการรับส่งโดยเฉพาะ เพื่อจัดลำดับและกำหนดทิศทางการรับส่งข้อมูลว่าเมื่อใดควรจะให้วงจรตัวใดทำหน้าที่รับและให้วงจรตัวใดทำหน้าที่ส่งข้อมูล ซึ่งต้องใช้วิธีการที่สลับซับซ้อนมากขึ้น

2.5 แนวทางการพัฒนาโปรแกรม

บอร์ด CP-JR180 V2.0 สามารถทำการพัฒนาโปรแกรมได้หลายวิธีตามความต้องการ ซึ่งการที่จะตัดสินใจเลือกแนวทางการพัฒนาโปรแกรมโดยวิธีใดนั้นต้องพิจารณาตามความเหมาะสมและความถนัดของแต่ละคน เพราะแต่ละวิธีล้วนมีข้อดี ข้อเสีย และความเหมาะสมที่แตกต่างกันไป

การพัฒนาโปรแกรมด้วยชุด ET-DEBUGGER JR180

การพัฒนาโปรแกรมด้วยวิธีนี้ เหมาะสำหรับผู้ที่ต้องการพัฒนาการทำงานของบอร์ดโดยใช้ ภาษาแอสเซมบลี ซึ่งต้องใช้งานร่วมกับคอมพิวเตอร์ PC โดยเชื่อมต่อผ่านทางพอร์ตอนุกรม RS232 ซึ่งการพัฒนาโปรแกรมด้วยวิธีนี้เหมาะสำหรับการพัฒนาในช่วงเริ่มต้นหรือระยะแรกของการทำงาน ทั้งนี้ก็เนื่องจากว่าชุด ET-DEBUGGER JR180 มีคำสั่งสำหรับสนับสนุนการทำงาน ที่สามารถอำนวยความสะดวกต่อการพัฒนาโปรแกรมได้มากมาย เช่นการ DownLoad โปรแกรมที่เขียนและแปลเป็นรหัสคำสั่งไว้แล้วลงมาไว้ยังหน่วยความจำชั่วคราวในตัวบอร์ด CP-JR180 V2.0 เพื่อสั่งให้โปรแกรมทำงานนอกจากนี้ยังสามารถทำการตรวจสอบความถูกต้องของโปรแกรมในส่วนต่าง ๆ ไม่ว่าจะเป็นคำสั่งให้ CPU ปฏิบัติงานทีละคำสั่ง (Single Step) เพื่อตรวจสอบผลการทำงานของแต่ละคำสั่ง หรือการสั่งหยุดการทำงานของโปรแกรม (Break) ในตำแหน่งที่เราต้องการเพื่อทำการตรวจสอบค่าในรีจิสเตอร์หรือหน่วยความจำหลังจากที่ CPU ปฏิบัติงานตามโปรแกรมที่เราเขียนไว้แล้วนั้น ว่าเป็นไปตามเงื่อนไขที่กำหนดไว้หรือไม่ ฯลฯ แต่การพัฒนาโปรแกรมด้วยวิธีนี้ ก็คงมีข้อจำกัดอยู่บ้าง เช่น ไม่สามารถใช้ทรัพยากรทั้งหมดที่มีอยู่บนบอร์ดได้ทั้งหมด เช่น พอร์ตอนุกรม UART Channel-1 หรือหน่วยความจำในบางตำแหน่ง รวมทั้งตำแหน่งที่เขียนโปรแกรม หรือ Interrupt Vector ต่าง ๆ ก็ไม่สามารถใช้งานที่ตำแหน่งจริงได้ เนื่องจากเราต้องเขียนโปรแกรมของเราให้ซ้อนทับอยู่บนโปรแกรมของชุด DEBUGGER อีกทีหนึ่ง ซึ่งโปรแกรมของชุด DEBUGGER เองก็ต้องใช้พื้นที่ในการทำงานด้วยส่วนหนึ่งเหมือนกัน แต่ปัญหานี้ก็ไม่ใช่ปัญหาหลักที่เป็นอุปสรรคสำคัญมากนัก เพราะโดยหลักการทั่วไปแล้ว การเขียนโปรแกรมบนเครื่องคอมพิวเตอร์ PC นั้น จะต้องเขียนเป็นข้อความ (Text Files) ไว้ก่อนแล้วใช้โปรแกรมจำพวกคอมไพเลอร์เช่น Cross32 V2.0 หรือ Z180ASM เพื่อทำการแปลข้อความของโปรแกรมนั้นให้เป็นรหัสคำสั่งอีกทีหนึ่ง ซึ่งโปรแกรมคอมไพเลอร์นั้นสามารถอ้างตำแหน่งเป็นสัญลักษณ์ข้อความ (Label) ได้ ซึ่งทำให้เราสามารถเปลี่ยนแปลงตำแหน่งการทำงานของโปรแกรมต่าง ๆ ได้ง่ายโดยเปลี่ยนแปลงที่ค่าของสัญลักษณ์ข้อความ (Label) เพียงแห่งเดียวเท่านั้น แล้วจึงทำการสั่งแปลโปรแกรมนั้นใหม่อีกครั้งหนึ่งก็จะได้โปรแกรมที่ถูกต้องสมบูรณ์เพื่อนำไปโปรแกรมลงใน EPROM จริงเพื่อใช้งานได้แล้ว

ข้อดีของการพัฒนาโปรแกรมด้วย DEBUGGER

1. สามารถทำการแก้ไขโปรแกรมและสั่ง RUN ได้ทันที โดยไม่ต้องทำการลบและทำการโปรแกรมข้อมูลใหม่เหมือนการใช้ EPROM

2. สามารถสั่งหยุดการทำงาน (Break) หรือสั่งให้โปรแกรมทำงานทีละคำสั่ง (Single Step) เพื่อตรวจสอบค่าของรีจิสเตอร์หรือหน่วยความจำได้
3. สามารถทำการติดต่อกับหน่วยความจำหรืออุปกรณ์อินพุตเอาต์พุตของบอร์ดได้โดยตรง
4. มีคำสั่งสำหรับอำนวยความสะดวกในการพัฒนาโปรแกรมมากมาย

ข้อเสียของการพัฒนาโปรแกรมด้วย DEBUGGER

1. ไม่สามารถเขียนโปรแกรมที่ตำแหน่งใช้งานจริงของ CPU ได้
2. ไม่สามารถใช้ทรัพยากรของบอร์ดได้ทั้งหมด เพราะทรัพยากรบางอย่าง ถูกสงวนไว้สำหรับโปรแกรม DEBUGGER เอง
3. ไม่สามารถนำชุด DEBUGGER ไปใช้งานกับบอร์ดไมโครโปรเซสเซอร์ รุ่นอื่น ๆ ที่มีระบบ Hardware ที่แตกต่างกันได้

การพัฒนาโปรแกรมด้วย EPROM EMULATOR (ET-EM8/32/128)

การพัฒนาโปรแกรมด้วยวิธีนี้ ก็เป็นที่นิยมอีกใช้งานกันอย่างแพร่หลายอีกวิธีหนึ่ง ซึ่งเหมาะสำหรับผู้ที่มีความชำนาญในการเขียนโปรแกรม อยู่บ้างแล้วพอสมควร หรือกล่าวอธิบายง่าย ๆ ก็คือ เมื่ออ่านข้อความคำสั่งในโปรแกรมแล้วสามารถทำความเข้าใจความหมายและหน้าที่ของการทำงานคำสั่งต่าง ๆ ในโปรแกรมได้ทันทีว่าเมื่อ CPU ปฏิบัติงานคำสั่งนั้น ๆ แล้วจะให้ผลลัพท์เป็นอย่างไร ซึ่งลักษณะของ EPROM EMULATOR ก็คือ จะเป็นแผงวงจรอิเลคทรอนิกส์วงจรหนึ่งที่ทำหน้าที่เหมือนกับ EPROM จริง ๆ หรืออาจเรียกว่า เป็นวงจรจำลองการทำงานของ EPROM ก็ได้ ซึ่งโดยทั่วไปแล้วจะสามารถปรับเลือกได้ว่าจะให้เป็น EPROM เบอร์อะไร โดยเราจะใช้ EPROM EMULATOR ET-EM แทน EPROM จริงในช่วงของการพัฒนาโปรแกรม เพราะถ้าหากใช้ EPROM จริงในการพัฒนาโปรแกรมจะเสียเวลามาก ไม่ว่าจะเป็นการโปรแกรมหรือการนำ EPROM ไปฉายแสงอุลตร้าไวโอเล็ต (U-V) เพื่อลบข้อมูลกลับมาโปรแกรมใหม่ ซึ่งในแต่ละครั้งอาจใช้เวลาหลายสิบนาที หากต้องทำหลาย ๆ ครั้งก็จะเสียเวลาเป็นอันมาก แต่ EPROM EMULATOR สามารถลดความยุ่งยากเหล่านั้นได้ โดยสามารถสั่งโปรแกรมข้อมูลซ้ำได้ทันที และไม่จำกัดจำนวนครั้งในการสั่งโปรแกรม ซึ่งในการใช้แต่ละครั้งจะใช้เวลาที่รวดเร็วมาก

ข้อดีของการพัฒนาโปรแกรมด้วย EPROM EMULATOR

1. สามารถเขียนโปรแกรมที่ตำแหน่งใช้งานจริงของ CPU ได้ทันที เมื่อพัฒนาโปรแกรมเสร็จสมบูรณ์แล้วสามารถนำโปรแกรมนั้นไปโปรแกรมลง EPROM จริงเพื่อใช้งานได้ทันทีโดยไม่ต้องแก้ไขใหม่
2. สามารถแก้ไขโปรแกรมและสั่ง DownLoad ข้อมูลใหม่ได้ทันที ภายในเวลาอันรวดเร็วโดยไม่ต้องเสียเวลา นำ EPROM ไปล้างและโปรแกรมใหม่
3. สามารถใช้ทรัพยากรทั้งหมดที่มีอยู่ในบอร์ดได้
4. สามารถแก้ไขโปรแกรมบางส่วนโดยไม่ต้องเสียเวลาทำการ Assembler หรือแปลโปรแกรมใหม่
5. สามารถนำ EPROM EMULATOR ไปใช้ได้กับบอร์ดไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ได้ทุกรุ่น โดยไม่จำกัดว่าจะใช้ CPU เบอร์อะไร ยี่ห้อใด ตระกูลใด เพียงแต่ขอให้บอร์ดไมโครโปรเซสเซอร์ นั้น ๆ ใช้ EPROM ที่มีขนาดความจุหรือเบอร์ตรงกับความสามารถที่ EPROM EMULATOR สามารถปรับเปลี่ยนและรองรับได้
6. สะดวกรวดเร็วในการพัฒนาโปรแกรม

ข้อเสียของการพัฒนาโปรแกรมด้วย EPROM EMULATOR

1. การแก้ไขความผิดพลาดของโปรแกรมทำได้ยาก เนื่องจากไม่สามารถติดตามตรวจสอบการทำงานของโปรแกรมได้ ซึ่งเมื่อโปรแกรมไม่ทำงานหรือทำงานไม่ถูกต้องตามต้องการ ผู้ใช้ต้องทำการตรวจสอบจาก Source โปรแกรมที่เขียนขึ้นเอง ว่าความผิดพลาดเกิดจากส่วนใดของโปรแกรม ซึ่งหากโปรแกรมมีความยาวและซับซ้อนมาก ๆ อาจต้องใช้การคาดเดาและใช้ความชำนาญหรือประสพการณ์ประกอบกับความรู้ความเข้าใจที่ดีพอสมควร ในการพิจารณาและวิเคราะห์หาสาเหตุความผิดพลาดนั้น
2. ราคาของ EPROM EMULATOR จะมีราคาที่ค่อนข้างสูงเมื่อเปรียบเทียบกับ DEBUGGER แต่ก็นับว่าคุ้มค่า เพราะเป็นการลงทุนเพียงครั้งเดียวแต่สามารถใช้งานได้ตลอดไม่ว่าจะเป็นบอร์ดไมโครโปรเซสเซอร์ รุ่นใด ๆ ก็ใช้ได้ทั้งหมด

การพัฒนาโปรแกรมด้วยชุด ET-BASIC180

สำหรับผู้ที่ต้องการพัฒนาโปรแกรมด้วยภาษา BASIC โดยใช้งานร่วมกับชุด ET-BASIC 180 นั้นจะมีขั้นตอนและกรรมวิธีคล้ายกับ DEBUGGER เพียงแต่แตกต่างกันในเรื่องของภาษาที่ใช้เท่านั้น ซึ่งภาษา BASIC ก็มีข้อดี ตรงที่มีความสะดวกและง่ายต่อการใช้งาน ผู้ใช้ไม่จำเป็นต้องทราบรายละเอียดของ CPU ต่าง ๆ มากมายนัก เพียงศึกษาหน้าที่และคำสั่งของภาษา BASIC ให้เข้าใจก็สามารถใช้งานได้แล้ว แต่การใช้งานด้วยภาษา BASIC นี้อาจมีข้อจำกัดในเรื่องของความเร็วอยู่บ้าง ดังนั้นงานบางลักษณะที่ต้องการความเร็วในการทำงานมาก ๆ จึงไม่เหมาะสมที่จะพัฒนาด้วยภาษา BASIC เช่น งานเกี่ยวกับการ SCAN จำพวกทำไฟวิ่งโฆษณาต่าง ๆ ทั้งนี้ก็เนื่องจากภาษา BASIC จะใช้วิธีการนำรหัสคำสั่งของ BASIC ที่ผู้ใช้เขียนไว้มาทำการแปลให้เป็นรหัสคำสั่งที่ CPU รับรู้ได้เสียก่อน แล้วจึงสั่งให้ CPU ปฏิบัติงานตามคำสั่งที่แปลได้อีกทีหนึ่งซึ่งทุกครั้งที่สั่ง RUN โปรแกรมด้วยภาษา BASIC นี้จะต้องเสียเวลาสำหรับการแปลคำสั่งจาก BASIC ให้เป็นรหัสคำสั่งของ CPU ทุกครั้งไป ดังนั้นการทำงานของภาษา BASIC จึงช้ากว่าภาษาแอสเซมบลีเสมอ

จะเห็นได้ว่าบอร์ด CP-JR180 V2.0 มีความอ่อนตัวในการใช้งานเป็นอย่างมาก สามารถประยุกต์ใช้งานได้อย่างกว้างขวาง และสามารถเลือกแนวทางการพัฒนาโปรแกรมได้หลายแนวทาง ซึ่งแต่ละแนวทางต่างก็มีข้อดี ข้อเสีย ที่แตกต่างกันไป ดังนั้นในการที่จะตัดสินใจว่าจะเลือกใช้แนวทางหรือวิธีการแบบใดในการพัฒนาโปรแกรมนั้น คงต้องพิจารณาจากองค์ประกอบหลาย ๆ อย่าง ไม่ว่าจะเป็นลักษณะของงานหรือพื้นฐานความรู้ความเข้าใจ รวมทั้งประสบการณ์ต่าง ๆ แล้วจึงพิจารณาตัดสินเลือกใช้แนวทางที่เหมาะสม

สรุปหน้าที่ของ Jumper ต่าง ๆ

JP1 ใช้สำหรับเลือกขนาด EPROM MONTITOR (U2) ว่าจะใช้หน่วยความจำขนาด 32 KB หรือ 64 KB (27256/27512)

JP2 ใช้กำหนดเลือกเบอร์ของหน่วยความจำ Expandsion U4

JP3 ใช้เลือกแหล่งจ่ายไฟให้หน่วยความจำ U4 ว่าจะใช้แหล่งจ่าย +VCC จากระบบหรือ +VBAT จาก Battery (สำหรับ Backup)

JP4 ใช้เลือกว่าจะใช้ WatchDog ควบคุมการรีเซ็ตหรือไม่ (WD/CLK)

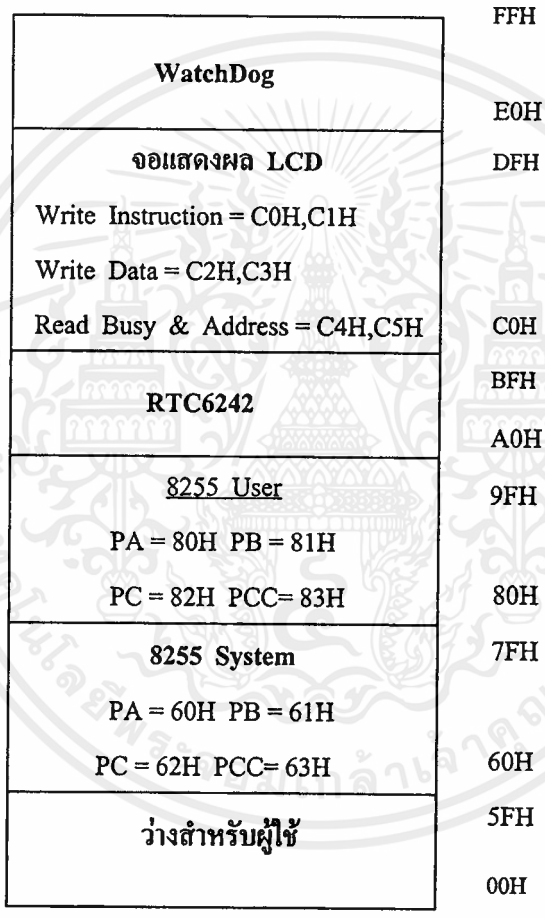
JP5 ใช้เลือกสัญญาณ CS# ของหน่วยความจำ EEPROM ว่าจะใช้สัญญาณ RTS0 จาก CPU หรือ สัญญาณ PB0 ของ 8255 (System)

JP6 ใช้เลือกวิธีการเขียนอ่าน EEPROM ว่าจะใช้แบบ 8 บิตหรือ 16 บิต

JP7 ใช้เลือกชนิดของจอแสดงผล LCD ว่าจะใช้รุ่นรูปภาพ Graphic หรือรุ่นตัวอักษร Dotmatrix (GR/CH)

JP8 ใช้เลือกวงจร Line Driver ของพอร์ตสื่อสาร UART Channel-1 ว่าจะใช้ MAX232 สำหรับการรับส่งแบบ RS232 หรือใช้ 75176 สำหรับการรับส่งแบบ RS422 และ RS485

JP9 ใช้เลือกรูปแบบการรับส่งของพอร์ตสื่อสาร UART Channel-1 ในกรณีที่ใช้ Line Driver เป็น 75176 แล้วว่าจะเป็นแบบ RS422 (Full Duplex) หรือ RS485 (Half Duplex)



รูปที่ 8 แสดง I/O MAP ของบอร์ด CP-JR180 V2.0

บทที่ 3
ฮาร์ดแวร์

พอร์ต RS 232

พอร์ต RS 232 นี้จะทำหน้าที่รับและส่งข้อมูลในแบบอนุกรมเรียกว่า Universal Asynchronous Adapter เหตุผลที่มีชื่อเรียกว่า RS232C ก็เนื่องจากสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์ของอเมริกาหรือ EIA ได้กำหนดมาตราฐานของอุปกรณ์การสื่อสารแบบอนุกรมภายใต้ชื่อว่า RS232C

<u>หน้าที่สำคัญของการสื่อสารแบบอะซิงโครนัสก็คือ</u>

<u>รับสัญญาณ</u>

1. เปลี่ยนสัญญาณเข้ามาเป็นแบบอนุกรมให้เป็นแบบขนาน
2. ตรวจสอบความผิดพลาดของสัญญาณที่รับ
3. ตัดสต๊อปบิตและพาริตี้บิตออก
4. ส่งสัญญาณให้ซีพียูรู้ว่ารับสัญญาณไว้แล้ว

<u>ส่งสัญญาณ</u>

1. เปลี่ยนสัญญาณแบบขนานจากซีพียูค่อยทยอยส่งออกเป็นแบบอนุกรม
2. เพิ่มสต๊อปบิตและพาริตี้
3. เพิ่มสัญญาณควบคุมโมเด็มที่ต่อเชื่อม (ถ้ามี)

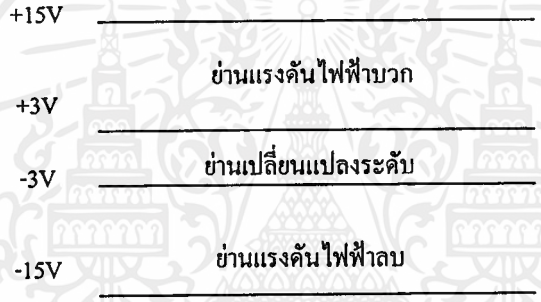
มาตรฐาน RS232C ก็เพื่อบรรยายคุณลักษณะของการเชื่อมต่ออุปกรณ์รับส่งข้อมูลปลายทาง (Datas Terminal Equipement :DET) กับอุปกรณ์สื่อสาร (Data Communication Equipement : DCE) สำหรับผู้ใช้ไมโครคอมพิวเตอร์ DTE ก็หมายถึงตัวไมโครคอมพิวเตอร์และ DCE ก็หมายถึงโมเด็ม อุปกรณ์อื่น ๆ เช่น เครื่องพิมพ์ที่รับสัญญาณแบบอนุกรมอาจจะเป็นได้ทั้ง DTE และ DCE ขึ้นอยู่กับผู้ผลิต

ความจริงอีกประการหนึ่งของ RS 232C ก็คือความเร็วและระยะทางของการเชื่อม ต่อ RS 232C สามารถเชื่อมต่อการถ่ายโอนข้อมูลได้จาก 0 - 20,000 บิตต่อวินาที ซึ่งเพียงพอสำหรับ

ไมโครคอมพิวเตอร์ที่มีขนาดอัตราบอด 110 ถึง 9600 บอด ความยาวของสายเชื่อมต่อโดยสัญญาณตามมาตราฐานของ RS 232 จำกัดอยู่แค่ 50 ฟุต

ลักษณะของสัญญาณ RS232C เพื่อเป็นหลักประกันว่าข้อมูลถูกส่งออกไปอย่างถูกต้องและอุปกรณ์ถูกควบคุมอย่างถูกต้อง จึงจำเป็นต้องมีข้อตกลงกันในเรื่องของสัญญาณที่ใช้ มาตรฐาน RS 232C กำหนดย่านของแรงดันไฟฟ้าในสัญญาณเพื่อสนองจุดประสงค์ดังแสดงในตาราง

มาตราฐานการใช้แรงดันไฟฟ้า			
แรงดันไฟฟ้า	สถานภาพลอจิก	สถานภาพของสัญญาณ	ฟังก์ชั่นในการควบคุม
บวก	0	SPACE	ON
ลบ	1	MARK	OFF

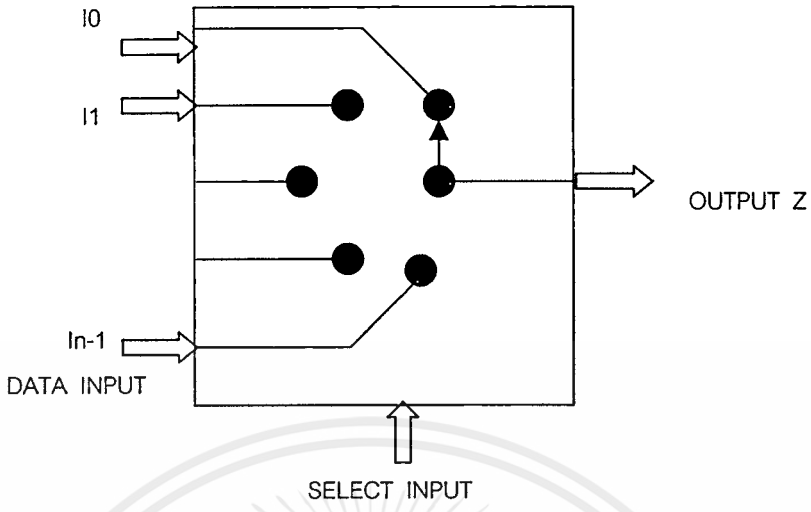


รูปที่ 9 ย่านของแรงดันไฟฟ้าที่ใช้สัญญาณ RS232C

สำหรับไมโครคอมพิวเตอร์บางเครื่อง ใช้แต่สัญญาณลอจิกออกมาเป็นสัญญาณของ RS232C เลย อย่างเช่นอะซิงโครนัสอะแดปเตอร์ของ IBM PC ในกรณีเช่นนี้ระยะทางของสายที่เชื่อมต่ออาจจะไปได้สั้นกว่า 50 ฟุต เนื่องจากระดับของกราวด์เปลี่ยนแปลงไปอันเนื่องจากการสูญเสียไปในความต้านฐานของสาย

<u>การมัลติเพล็กเซอร์</u>

มัลติเพล็กเซอร์ (MUX) หรืออุปกรณ์เลือกข้อมูล (Data Selector) เป็นวงจรที่มีหลายอินพุตแต่มีเอาท์พุตเดียว ที่เวลาใดเวลาหนึ่งอินพุต SELECT จะทำหน้าที่ควบคุมข้อมูลอินพุต (Data Input) แล้วส่งมายังเอาต์พุต

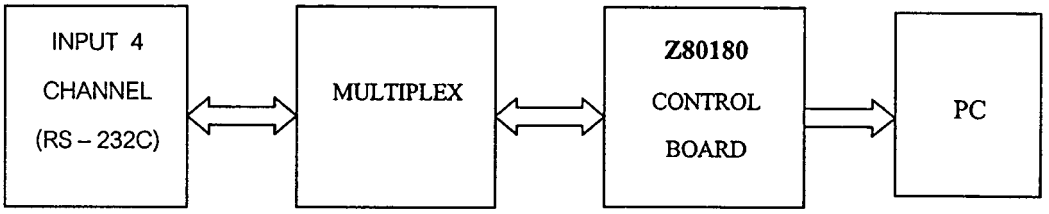


รูปที่ 10 แผนภูมิการทำงานการมัลติเพล็กเซอร์

จากรูปที่ 10 แสดงแผนภูมิการทำงานของ MUX สังเกตว่าอินพุต SELECT ของ MUX มีการทำงานเหมือนกับสวิทช์หลายตำแหน่งที่ใช้ควบคุมการทำงานทางดิจิตอลนั่นคือเมื่อป้อนรหัสดิจิตอลหรือข้อมูลอินพุตเข้าไปยังอินพุต SELECT สวิทช์จะควบคุมหรือเลือกข้อมูลอินพุตนี้แล้วส่งไปยังเอาต์พุต เช่นที่เวลาหนึ่งเอาต์พุต Z กับข้อมูลอินพุต I0 หรืออีกเวลาหนึ่งเอาต์พุต I1 เป็นต้น

อีกนัยหนึ่ง MUX จะทำหน้าที่เลือกข้อมูลอินพุต 1 ในจำนวน N ข้อมูลจากนั้นจึงเคลื่อนย้ายข้อมูลที่เลือกไปยังเอาต์พุตซึ่งเราเรียกว่า การมัลติเพล็กเซอร์

ส่วนประกอบทางฮาร์ดแวร์

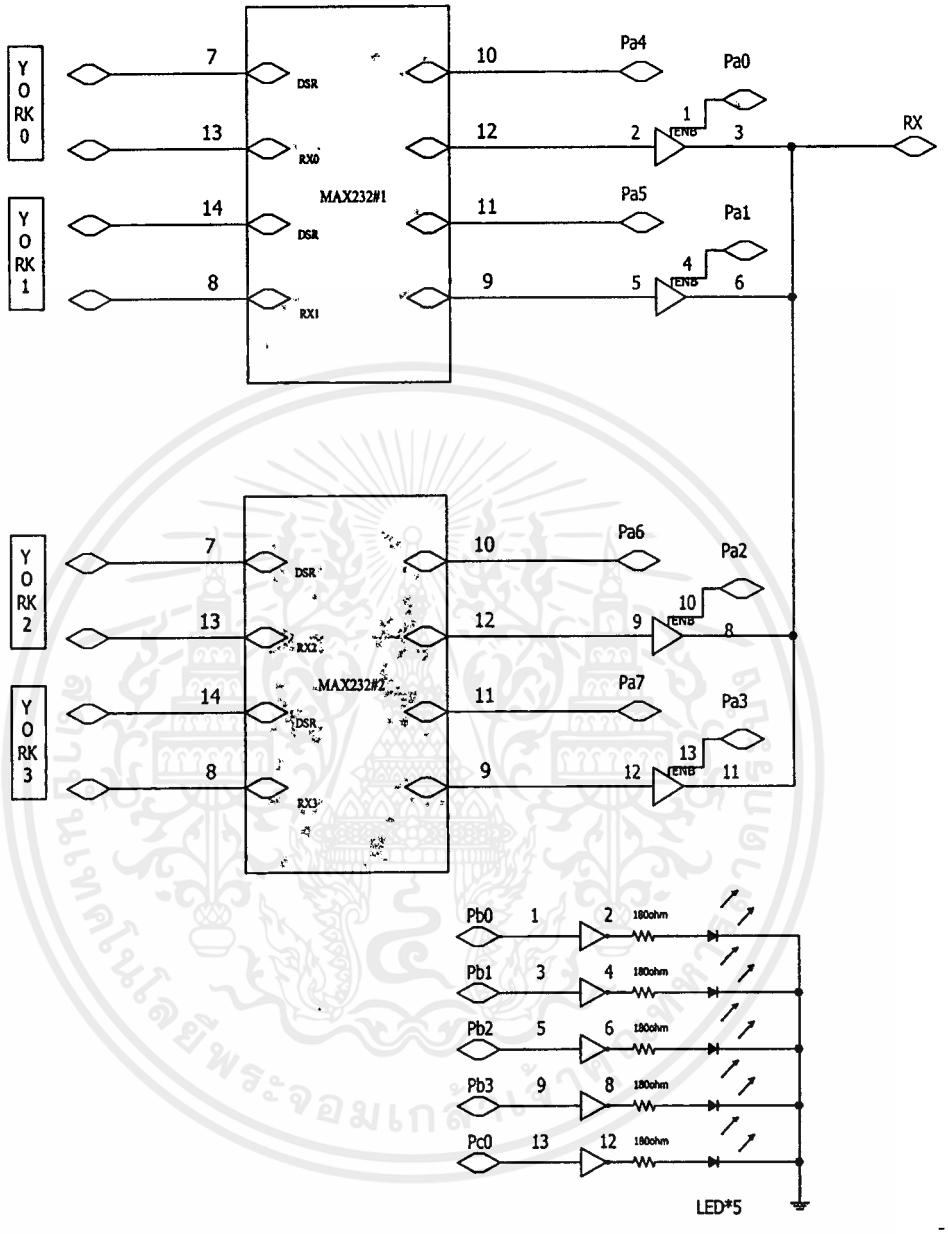


รูปที่ 11 บล็อคไดอะแกรมแสดงส่วนประกอบทางฮาร์ดแวร์

หลักการทำงานทางฮาร์ดแวร์

จากบล็อคไดอะแกรมของรูปที่ 11 แสดงการ Multiplex ข้อมูลของเครื่องทำความเย็น จำนวน 4 เครื่อง ซึ่งมีอินพุต 4 สายคือ Rx0 ถึง Rx3 จะทำการส่งข้อมูลผ่านวงจร Multiplex เพื่อเลือกข้อมูลอินพุตที่ต้องการส่งต่อไปยัง บอร์ด Z80180 ซึ่ง บอร์ด Z80180 จะทำการประมวลผลข้อมูลที่ต้องการเก็บไว้ จนเมื่อถึงระยะเวลาหนึ่งเมื่อข้อมูลมีจำนวนมากเราสามารถส่งข้อมูลไปสู่คอมพิวเตอร์เพื่อทำการวิเคราะห์ข้อมูล

วงจรมัลติเพล็กเซอร์



รูปที่ 12 แสดงวงจรมัลติเพล็ก

หลักการทำงานของวงจรมัลติเพล็ก

เมื่อเครื่องทำความเย็นส่งสัญญาณข้อมูลจะส่งเข้าผ่าน IC MAX232 เพื่อเปลี่ยนแปลงระดับสัญญาณให้สามารถรับส่งข้อมูลแบบอนุกรมเพื่อติดต่อสื่อสารกับพอร์ต RS 232 ได้ เมื่อสัญญาณผ่าน IC MAX232 จะส่งสัญญาณไปที่ IC 74SL125 เพื่อทำการเลือกสัญญาณข้อมูลที่ต้องการ ในการเลือกสัญญาณแต่ละครั้งเราจะควบคุมสัญญาณการเลือกข้อมูลด้วยโปรแกรม ซึ่งเราจะสามารถทราบได้ว่าขณะนี้เครื่องทำความเย็นเครื่องใดที่กำลังส่งข้อมูลให้กับบอร์ด Z80180 เพื่อทำการประมวลผลโดยดูได้จากสัญญาณไฟ LED

หลักการทำงานทางซอร์ฟแวร์

ในส่วนของโปรแกรมจะทำการเขียนโปรแกรมเพื่อทำการรับข้อมูลจากเครื่องทำความเย็น (YORK) 4 เครื่องทำการสแกนรับทีละเครื่อง ซึ่งเครื่องทำความเย็นจะทำการส่งข้อมูลมายังเครื่องเก็บข้อมูลต้นแบบที่สร้างขึ้น โดยเครื่องเก็บข้อมูลต้นแบบจะทำการรับข้อมูลที่เครื่องทำความเย็นส่งมาถึงคำว่า PUR และจะเลือกเก็บเฉพาะข้อมูลที่เป็นตัวเลข ซึ่งข้อมูลที่เครื่องต้นแบบเก็บไว้นี้สามารถที่จะส่งไปยังเครื่องคอมพิวเตอร์ได้ด้วย

ตัวอย่างไฟล์ข้อมูลที่เก็บไว้ที่ได้จากผลการทดลอง

```
;1;1990;.;.01;.11;12:00P;06/07/98;53.0;61.2;11.8;21.8;46.2;374;333;272;384;349;385;
91.2;29.7;90;;.;3741;71;51.0;91;421;0;100;0;20;55.6;100.0;112.5;150.4END
;2;1990;.;.01;.11;12:05A;06/07/98;53.0;61.2;7.8;20.8;41.2;378;393;372;389;389;385;
97.2;89.7;91;.;3748;77;52.0;90;431;0;100;0;20;52.6;100.0;112.3;120.4END
;1;1990;.;.01;.11;12:00P;06/07/98;53.0;61.2;11.8;21.8;46.2;374;333;272;384;349;385;
91.2;29.7;90;;.;3741;71;51.0;91;421;0;100;0;20;55.6;100.0;112.5;150.4END
;2;1990;.;.01;.11;12:05A;06/07/98;53.0;61.2;7.8;20.8;41.2;378;393;372;389;389;385;
97.2;89.7;91;.;3748;77;52.0;90;431;0;100;0;20;52.6;100.0;112.3;120.4END
;1;1990;.;.01;.11;12:00P;06/07/98;53.0;61.2;11.8;21.8;46.2;374;333;272;384;349;385;
91.2;29.7;90;;.;3741;71;51.0;91;421;0;100;0;20;55.6;100.0;112.5;150.4END
;2;1990;.;.01;.11;12:05A;06/07/98;53.0;61.2;7.8;20.8;41.2;378;393;372;389;389;385;
97.2;89.7;91;.;3748;77;52.0;90;431;0;100;0;20;52.6;100.0;112.3;120.4END
;1;1990;.;.01;.11;12:00P;06/07/98;53.0;61.2;11.8;21.8;46.2;374;333;272;384;349;385;
91.2;29.7;90;;.;3741;71;51.0;91;421;0;100;0;20;55.6;100.0;112.5;150.4END
```

YORK SYSTEM 1 UPDATE

c1990 YORK INTERNATIONAL CORP.

VERSION C.01F.11

TODAY IS SUN 12:00 PM 06/07/98

CHILLED LEAVING = 53.0iF; RETURN = 61.2iF

EVAP = 11.8 PSIA; COND = 21.8 PSIA

OIL PRESSURE = 46.2 PSID

NO OPTIONS INSTALLED

A AMPS = 374; B AMPS = 333; C AMPS = 272

V A-B = 384; V B-C = 349; V C-A = 385

COND LEAVING = 91.2 iF; RETURN = 29.7 iF

MOTOR CURRENT = 90% FLA

OPER. HOURS = 3741; START COUNTER = 71

LEAVING SETPOINT = 51.0 iF

CURRENT LIMIT = 91% FLA; MTR CUR = 421 FLA

SETPOINT = .0MIN @ 100% FLA, 0 MIN LEFT

S M T W T F S HOLIDAY NOTED BY *

REMOTE TEMP SETPOINT RANGE = 20 iF

SAT TEMPS EVAP = 55.6 iF, COND = 100.0 iF

DISCHARGE TEMP = 112.5 iF,OIL TEMP = 150.4 iF

PURGE PRESSURE = 15.5 PSIA

PURGES LAST HOUR = 0;MAX PURGES/HR= 20

SUN START = 00:00 AM, STOP 00:00 AM

MON START = 00:00 AM, STOP 00:00 AM

TUE START = 00:00 AM, STOP 00:00 AM

ไฟล์ตัวอย่างข้อมูลที่เครื่อง YORK

บทที่ 4
สรุปผลและวิจารณ์

1. สรุปผลการทดลอง

ปริญญานิพนธ์ฉบับนี้ ได้ทำการสร้างเครื่องเก็บข้อมูลสำหรับเครื่องจักรในโรงงานอุตสาหกรรม ซึ่งได้ทำการสร้างเป็นเครื่องต้นแบบในการเก็บข้อมูลจากเครื่องทำความเย็นในโรงงานอุตสาหกรรมโดยอัตโนมัติด้วยไมโครโปรเซสเซอร์เบอร์ Z80180 ของบริษัท ZILOG และทำการโปรแกรมการทำงานด้วยภาษาแอสเซมบลี เพื่อใช้ในการวิเคราะห์ข้อมูลและทำการเก็บข้อมูล ซึ่งเครื่องต้นแบบที่ทำการสร้างขึ้นจะสามารถรับอินพุตจากเครื่องทำความเย็นได้ทั้งหมด 4 อินพุตและสามารถรอรับการเก็บข้อมูลได้ตลอด 24 ชั่วโมงมีความจุในการเก็บข้อมูลได้มาก จากการทดลองเครื่องต้นแบบสามารถติดต่อกับเครื่องไมโครคอมพิวเตอร์โดยใช้หลักการในการอินเตอร์เฟสเพื่อจัดเก็บข้อมูลได้หากมีการเรียกดูทีหลังได้ทุกเวลา ซึ่งช่วยอำนวยความสะดวกให้แก่วิศวกรและช่วยลดทั้งเวลาและต้นทุนให้กับโรงงานอุตสาหกรรม

2. แนวทางในการพัฒนาและประยุกต์ใช้งาน

2.1 ควรประยุกต์ใช้งานเครื่องเก็บข้อมูลกับเครื่องจักรในโรงงานอุตสาหกรรมที่มีลักษณะการทำงานคล้ายกับเครื่องทำความเย็น

2.2 ควรทำการพัฒนาโปรแกรมเพื่อให้เครื่องเก็บข้อมูลสามารถทำการติดต่อการเครื่องพิมพ์ได้โดยไม่ต้องผ่านเครื่องคอมพิวเตอร์

2.3 ควรเพิ่มจำนวนในการรับอินพุตของเครื่องเก็บข้อมูลให้เหมาะสมกับเครื่องจักรในโรงงานอุตสาหกรรม

3. ปัญหาที่เกิดจากการทำโครงงาน

ในการทดลองปัญหาที่เกิดขึ้นประการหนึ่งจะขึ้นอยู่กับอุปกรณ์การทดลอง และอีกประการหนึ่งคือการเขียนโปรแกรมเพื่อควบคุมบอร์ดไมโครโปรเซสเซอร์ ซึ่งจะต้องทำการศึกษาคุณสมบัติและโครงสร้างของบอร์ดให้เข้าใจ อาทิเช่น หน่วยความจำและคำสั่งต่าง ๆ ก่อนที่จะนำไปใช้งานในโครงงานนี้ได้ใช้ภาษาแอสเซมบลีซึ่งยากแก่การเข้าใจแต่ก็มีข้อดีที่สามารถทำงานได้อย่างรวดเร็ว

หนังสืออ้างอิง

เข้าใจ/สร้าง/เล่น ไมโครโปรเซสเซอร์1, บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน), กรุงเทพมหานครฯ , 2538

คู่มือไมโครโปรเซสเซอร์ CPU Z80180, บริษัท อีทีที จำกัด, กรุงเทพมหานครฯ

คู่มือการใช้บอร์ด CP-JR180 V2.0, บริษัท อีทีที จำกัด, กรุงเทพมหานครฯ

ภาคผนวก ก.

```
;*********************************************
;       PROGRAM Z80180                        *
;       ASM SOFTWARE A180Z                    *
;       MONITOR PROGRAM COMMON AREA 0         *
;       (LOGICAL ADDRESS)  0000H-7FFFH        *
;       (PHYSICAL ADDRESS) 0000H-7FFFH        *
;       RAM  BANK AREA                        *
;       (LOGICAL ADDRESS)  8000H-EFFFH        *
;       (PHYSICAL ADDRESS) 10000H-16FFFH      *
;       RAM COMMON AREA 1                     *
;       (LOGICAL ADDRESS)  F000H-FFFFH        *
;       (PHYSICAL ADDRESS) 17000H-1FFFFH      *
;*********************************************
;
;******************** ASCI **********************
;
    .EQU    FR1,0DBH
    .EQU    FR2,0B0H
    .EQU    CNTLA0,0        ;ASCI CONTROL REGISTER A CH 0
    .EQU    CNTLA1,1        ;ASCI CONTROL REGISTER A CH 1
    .EQU    CNTLB0,2        ;ASCI CONTROL REGISTER B CH 0
    .EQU    CNTLB1,3        ;ASCI CONTROL REGISTER B CH 1
    .EQU    STAT0,4         ;ASCI STATUS REGISTER CH 0
    .EQU    STAT1,5         ;ASCI STATUS REGISTER CH 1
    .EQU    TDR0,6          ;ASCI TRANSMIT DATA REGISTER CH 0
    .EQU    TDR1,7          ;ASCI TRANSMIT DATA REGISTER CH 1
    .EQU    RDR0,8          ;ASCI RECEIVE DATA REGISTER CH 0
    .EQU    RDR1,9          ;ASCI RECEIVE DATA REGISTER CH 1
```

```
;
;********************** CSI/O ********************
;
    .EQU    CNTR,0AH            ;CONTROL REFGISTER
    .EQU    TRDR,0BH            ;TRANSMIT/RECEIVE DATA REGISTER
;
;************************ TIMER ******************
;
    .EQU    TMDR0L,0CH          ;TIMER DATA REGISTER CH 0L
    .EQU    TMDR0H,0DH          ;TIMER DATA REGISTER CH 0H
    .EQU    RLDR0L,0EH          ;RELOAD REGISTER CH 0L
    .EQU    RLDR0H,0FH          ;RELOAD REGISTER CH 0H
    .EQU    TCR,10H             ;TIMER CONTROLREGISTER
    .EQU    TMDR1L,14H          ;TIMER DATA REGISTER CH 1L
    .EQU    TMDR1H,15H          ;TIMER DATA REGISTER CH 1H
    .EQU    RLDR1L,16H          ;RELOAD REGISTER CH 1L
    .EQU    RLDR1H,17H          ;RELOAD REGISTER CH 1H
;
;************************ OTHER ******************
;
    .EQU    FRC,18H             ;FREE RUNNING COUNTER
;
;************************ DMA ********************
;
    .EQU    SAR0L,20H   ;DMA SOURCE ADDRESS REGISTER CH 0L
    .EQU    SAR0H,21H   ;DMA SOURCE ADDRESS REGISTER CH 0H
    .EQU    SAR0B,22H   ;DMA SOURCE ADDRESS REGISTER CH 0B
    .EQU    DAR0L,23H   ;DMA DESTINATION ADDRESS REGISTER CH 0L
    .EQU    DAR0H,24H   ;DMA DESTINATION ADDRESS REGISTER CH 0H
    .EQU    DAR0B,25H   ;DMA DESTINATION ADDRESS REGISTER CH 0B
```

```
    .EQU    BCR0L,26H    ;DMA BYTE COUNT REGISTER CH 0L
    .EQU    BCR0H,27H    ;DMA BYTE COUNT REGISTER CH 0H
    .EQU    MAR1L,28H    ;DMA MEMORY ADDRESS REGISTER CH 1L
    .EQU    MAR1H,29H    ;DMA MEMORY ADDRESS REGISTER CH 1H
    .EQU    MAR1B,2AH    ;DMA MEMORY ADDRESS REGISTER CH 1B
    .EQU    IAR1L,2BH    ;DMA I/O ADDRESS REGISTER CH 1L
    .EQU    IAR1H,2CH    ;DMA I/O ADDRESS REGISTER CH 1H
    .EQU    BCR1L,2EH    ;DMA BYTE COUNT REGISTER CH 1L
    .EQU    BCR1H,2FH    ;DMA BYTE COUNT REGISTER CH 1H
    .EQU    DSTAT,30H    ;DMA STATUS REGISTER
    .EQU    DMODE,31H    ;DMA MODE REGISTER
    .EQU    DCNTL,32H    ;DMA/WAIT CONTROL REGISTER
;
;************************ INT ********************
;
    .EQU    IL,33H       ;IL REGISTER
    .EQU    ITC,34H      ;INT/TRAP CONTROL REGISTER
;
;********************** REFRESH ********************
;
    .EQU    RCR,36H      ;REFRESH CONTROL REGISTER
;
;************************ MMU ********************
;
    .EQU    CBR,38H      ;MMU COMMON BASE REGISTER
    .EQU    BBR,39H      ;MMU BANK BASE REGISTER
    .EQU    CBAR,3AH     ;MMU COMMON/BANK AREA REGISTER
```

```
;
;********************** I/O ********************
;
    .EQU    OMCR,3EH    ;OPERATION MODE CONTROL REGISTER
    .EQU    ICR,3FH     ;I/O CONTROL REGISTER
;
;********************** LCD ********************
;
    .EQU    PDATA,0C0H
    .EQU    PSING,0C2H
    .EQU    PREAD,0C4H
;
;**************END   I/O MAP REG Z80180**************
;
    .EQU    ORIGIN,0000H
    .EQU    DATA_BUF,8400H      ;END ADDRESS DATA BUFFER
    .EQU    SPACE_BUF,0F000H    ;RAM COMMON AREA1
    .EQU    SPACE_CON,08000H    ;START ADDRESS DATA BUFFER
    .ORG    ORIGIN
;
;*********************************************
;           POWER UP DELAY                  *
;*********************************************
INIT:
            LD      HL,08000H   ;DELAY
INIT1:
            DEC     HL
            LD      A,H
            OR      L
            JR      NZ,INIT1
```

```
;
;*************** GET MEMORY MAP *****************
;
INIT2:
                LD      A,0F8H
                OUT0    (CBAR),A        ;MONITOR 0-7FFF NOT REMOVE
                LD      A,8H
                OUT0    (BBR),A         ;BANK 10000H CAN REMOVE
                LD      A,8H
                OUT0    (CBR),A         ;STACK 10000H NOT REMOVE
                LD      SP,0FFFFH       ;SYSTEM STACK
;
;************** SET INTERRUPT VECTOR *************
;
                LD      A,0
                LD      I,A             ;VECTOR HIGH
                LD      A,40H           ;VECTOR LOW
                OUT0    (IL),A          ;SET VECTOR INTERRUPT
                IM      1               ;FOR INT0
                LD      A,8             ;INT ASCI CHANEL 1
                OUT0    (STAT1),A
                LD      A,0             ;INT ASCI CHANEL 0
                OUT0    (STAT0),A
                LD      A,44H           ;RX,8BIT,1STOP  CHANEL 1
                OUT0    (CNTLA1),A
                LD      A,64H           ;TX,RX,8BIT,1STOP CHANEL 0
                OUT0    (CNTLA0),A
                LD      A,2H            ;9600  AT X'TAL 12.288 MHZ
                OUT0    (CNTLB0),A
                LD      A,5H            ;1200  AT X'TAL 12.288 MHZ
```

```
                OUT0   (CNTLB1),A
;
;*********** SET INITIAL INTERNAL I/O ***********
;
                LD     A,40H
                OUT0   (OMCR),A       ;SAME Z80
                LD     A,83H
                OUT0   (RCR),A         ;ENABLE REFRESH
                IN0    A,(DCNTL)
                OR     70H
                OUT0   (DCNTL),A      ;MEM 2 WIAT  IO 4
;
; ******** I/O CONTROL REGISTER ADDRESS *********
;***************** 8255 USER PORT ******************
;
        .EQU   PA,80H
        .EQU   PB,81H
        .EQU   PC,82H
        .EQU   PCC,83H
        .EQU   PA0,60H
        .EQU   PB0,61H
        .EQU   PC0,62H
        .EQU   PCC0,63H
;
;************ COMPARISION CHARACTER *************
;
        .EQU   CHA1,050H              ; 'P'
        .EQU   CHA2,055H              ; 'U'
        .EQU   CHA3,052H              ; 'R'
        .EQU   CHA_W,057H             ; 'W'
```

```
        .EQU    CHA_X,058H          ; 'X'
        .EQU    CHA_Y,059H          ; 'Y'
        .EQU    CHA_Z,05AH          ; 'Z'
;
;***************** CONTROL PORT *****************
;
                LD      A,PA
                OUT     (PCC),A
                LD      A,90H
                OUT     (PCC0),A
;
;******************** INITIAL LCD ****************
;
                LD      A,00111000B   ;FUNCTION SET  38H
                                      ;DL=1 8 BIT,N=1 1/16 DUTY,F=0 5X7
                OUT     (PDATA),A
                CALL    DELAY1
                CALL    DELAY1        ;DELAY > 4.1 MS
                LD      A,00001111B   ;DISPLAY ON/OFF CONTROL
                                      ;D=1 OFF,C=1 CURSOR ON,B=1 BLINK
                OUT     (PDATA),A
                CALL    DELAY1
                LD      A,00000110B   ;ENTRY MODE SET
                                      ;I/D=1 INCREMENT,S=0 RIGHT
                OUT     (PDATA),A
                CALL    DELAY1
;
;************************************************
;
```

```
MAIN:
            CALL  SOUND_START
            LD    HL,DATA_BUF
            DEC   HL
            LD    (AD_END),HL          ;INITIAL FIRST DATA ADDRESS AREA
            LD    A,'W'                ;CONTROL WORD FOR CHANNEL 1
            LD    (CHANNEL),A
MAIN0:
            LD    A,0
            LD    (COUNT_PUR),A        ;START TO COUNT P-U-R
            LD    (COUNT_CHA),A        ;START TO COUNT FOR INSERT ";"
            LD    HL,SPACE_CON         ;START TO KEEP DATA BUFFER
            LD    (AD_ENDBX),HL        ;INITAL SPACE_CON ADDRESS
            CALL  TABLE1_DISPLAY       ;DISPLAY 'DATA ACQUISITION'
;
;----------------------SCAN CHANNEL----------------
SCAN_DATA:  CALL  SCAN                 ;SCAN CHANNEL
            ;
            LD    A,0FFH               ;FOR CLEAR PORT A,B,C
            CALL  CLEAR_LED
            ;
            LD    A,(HL)               ;ENABLE GATE
            OUT0  (PA),A
            OUT0  (PB),A
            CALL  DELAY1
            ;
            INC   HL
            LD    A,(HL)               ;ENABLE DSR SIGNAL
            OUT0  (PA),A
            LD    HL,SPACE_CON
```

```
;
;---------- IN DATA ----------
;
IN_DATA:                                ;WAIIT FOR DATA 256 LOOP
        LD      DE,0H
RXBYTE1:
        DEC     DE
        LD      A,E
        OR      D
        JP      Z,SEND_DATA             ;NO DATA IN SO GO TO SEND TO PC
        IN0     C,(STAT1)               ;RXREADY
        BIT     7,C
        JP      Z,RXBYTE1
        IN0     C,(RDR1)                ;RECEIVE
        LD      A,C
;
;---------- KEEP ALL DATA ----------
;
        LD      (HL),A
        INC     HL
        CP      CHA1
        JP      Z,ITS_P
        CP      CHA2
        JP      Z,ITS_U
        CP      CHA3
        JP      Z,ITS_R
        LD      A,0
        LD      (COUNT_PUR),A           ;CLEAR COUNTER PUR
        JP      IN_DATA
```

```
ITS_P:
        LD      IY,COUNT_PUR        ;CHECK STATUS 'P'
        INC     (IY+0)              ;INCREASE ONE FOR "P"
        LD      (IX+0),'P'
        JP      IN_DATA
ITS_U:
        LD      IY,COUNT_PUR        ;CHECK STATUS 'U'
        INC     (IY+0)              ;INCREASE ONE FOR "U"
        LD      (IX+1),'U'
        JP      IN_DATA
ITS_R:
        LD      (IX+2),'R'          ;CHECK STATUS 'R'
        CALL    CHECK_END           ;CALL CHECK P-U-R
        LD      IY,FLAG1
        BIT     2,(IY+0)
        JP      Z,IN_DATA           ;NO END SO GET DATA
        LD      (AD_ENDBX),HL       ;ENDING ADDRESS SPACE_CON
        PUSH    HL
        LD      HL,TABLE2           ;DISPLAY 'RECEIVE  O.K.'
        CALL    WRP
        CALL    SOUND_ONE
        CALL    DELAY1              ;WAIT DATA END
        CALL    DELAY1
        CALL    DELAY1
        CALL    DELAY1
        LD      HL,TABLE3           ;DISPLAY 'PLEASE WAIT'
        CALL    WRP
        POP     HL
```

```
;
;---------- FINISHED KEEP ALL DATA ON SPACE_CON ----------
;
                LD      DE,SPACE_CON
                LD      HL,(AD_ENDBX)
                XOR     A
                SBC     HL,DE
                LD      B,H                     ;AMOUNT OF DATA IN
                LD      C,L
                LD      HL,SPACE_CON
                LD      DE,(AD_END)             ;DE <-- LAST ADDRESS DATA AREA
                LD      A,D
                CP      084H
                JR      NZ,INC_DE
                LD      A,E
                CP      000H
                JR      NZ,INC_DE       ;INCREASE DE TO POINT START NEXT BLOCK
                JP      CH_NUM
INC_DE:
                INC     DE
;
;---------------------------- ONLY NUMBER -------------------------
;
CH_NUM:
                LD      A,(HL)
                INC     HL
                EX      AF,AF'
                DEC     BC
                LD      A,B
                OR      C
```

```
                JP      Z,CH_LIMIT          ;CHECK AMOUNT OF DATA
                EX      AF,AF'
                CALL    CHECK_NUM
                LD      IX,FLAG1
                BIT     0,(IX+0)
                JP      NZ,CH_CHA
                BIT     3,(IX+0)
                JP      NZ,IN_COMMA
                LD      (DE),A
                INC     DE
                JP      CH_NUM
IN_COMMA:
                RES     3,(IX+0)
                EX      AF,AF'
                LD      A,';'
                LD      (DE),A
                EX      AF,AF'
                INC     DE
                LD      (DE),A
                INC     DE
                JP      CH_NUM
CH_CHA:
                SET     3,(IX+0)
                JP      CH_NUM
;
;---------- INPUT END & KEEP PARAMETER ----------
;
CH_LIMIT:
                LD      A,'E'
                LD      (DE),A
```

```
        INC     DE
        LD      A,'N'
        LD      (DE),A
        INC     DE
        LD      A,'D'
        LD      (DE),A
        INC     DE
        LD      A,0DH
        LD      (DE),A
        INC     DE
        LD      A,0AH
        LD      (DE),A
        LD      (AD_END),DE
        LD      IX,BL_COUNT
        LD      B,0
        LD      A,(BL_COUNT)        ;GET BLANK NUMBER
        LD      C,A
        ADD     IX,BC
        LD      DE,(AD_END)
        LD      IY,BL_MUL
        ADD     IY,BC               ;POSITION OF MULTIPLY
        LD      C,(IY+0)
        ADD     IX,BC
        LD      (IX+1),E        ;KEEPING LAST ADDRESS IN THAT BLOCK
        LD      (IX+2),D
;
;---------- CHECK LIMIT OF BLANK0 BUFFER MEMORY ----------
;
        XOR     A
        LD      HL,(AD_END)
```

```
                LD      DE,0D7FFH           ;DEAD LINE IN BUFFER BLOCK
                SBC     HL,DE
                JP      M,DETECT1           ;IF AD_END < 0DFFFH STILL GET DATA
                JP      MAIN                ;IF OVER CLEAR ALL
DETECT1:        LD      HL,(AD_END)
                LD      (AD_ENDB0),HL   ;KEEP LAST ADD FOR BUFFTER BLACK0
                CALL    SDELAY              ;WAIT A WHILE FOR ALL DATA OUT
                CALL    SOUND_ONE
                CALL    SOUND_ONE
;---------------------- SEND DATA TO PC --------------------------------
;
SEND_DATA:      PUSH    HL
                LD      HL,(AD_END)
                LD      DE,DATA_BUF
                XOR     A
                SBC     HL,DE
                POP     HL
                JP      M,SCAN_DATA     ;IF NO DATA--> SO GET NEXT CHANNEL
S_DATA:         LD      HL,(AD_END)
                LD      D,H
                LD      E,L
                LD      HL,DATA_BUF     ;HL IS START DATA
                IN0     C,(PC)
                LD      A,C
                BIT     0,A
                JP      NZ,SEND_ONLY
                LD      C,'X'
                CALL    TXBYTE              ;SEND CODE FOR SENDING DATA
                CALL    TXBYTE
                LD      B,0H
```

```
S_DATA1:
        DJNZ    S_DATA1
        LD      B,02H
RXBYTE2:
        PUSH    BC
        LD      BC,0H
RXBYTE3:
        DEC     BC
        LD      A,B
        OR      C
        JP      NZ,CC_B0
        POP     BC
        DJNZ    RXBYTE2
        JP      MAIN0           ;THERE IS NOT COMPUTER IN
CC_B0:
        IN0     A,(STAT0)       ;RXREADY
        BIT     7,A
        JP      Z,RXBYTE3
        IN0     A,(RDR0)        ;RECEIVE
        CP      'X'
        JR      NZ,RXBYTE3
        POP     BC
;---------------------------------------------
SEND_ONLY: CALL SOUND_ONE
        CALL    SOUND_ONE
        CALL    SOUND_ONE
        LD      A,0FFH
        RES     4,A
        OUT0    (PB),A
```

```
S_DATA2:
        LD      C,(HL)          ;GET DATA TO SEND
        CALL    TXBYTE
        PUSH    BC
        PUSH    AF
        LD      BC,00FFH
B_DOWN1:
        DEC     BC
        LD      A,B
        OR      C
        JR      NZ,B_DOWN1
        POP     AF
        POP     BC
        INC     HL
        LD      A,H
        CP      D
        JP      NZ,S_DATA2
        LD      A,L
        CP      E
        JP      NZ,S_DATA2
        LD      C,(HL)
        CALL    TXBYTE          ;SEND LAST CHARACTER
        LD      C,'J'
        CALL    TXBYTE
        PUSH    HL
        LD      HL,TABLE4       ;DISPLAY 'TRANSMITTING...'
        CALL    WRP
        CALL    DELAY1
        CALL    DELAY1
```

```
                CALL   DELAY1
                CALL   DELAY1
                POP    HL
                JP     MAIN                ;NEW LOOP
;************************************
;       CHECK GATE                  *
;************************************
;
CHECK_GATE:
                PUSH   HL
OPEN_GATE:
                IN0    C,(PC)
                LD     A,C
                BIT    0,A
                JP     Z,END_GATE
                ;
                CALL   SOUND_ONE
                CALL   SOUND_ONE
                CALL   SOUND_ONE
                CALL   SOUND_ONE
                ;
                LD     HL,TABLE5
                CALL   WRP
                LD     A,0FFH
                CALL   CLEAR_LED
                CALL   DELAY1
                LD     A,0
                CALL   CLEAR_LED
                CALL   DELAY1
                JP     OPEN_GATE
```

```
END_GATE:
                POP    HL
                RET
;
;----------------------SUBROUNTINE----------------------
;*********************************************
;     DISPLAY DATA ACQUISITION                *
;*********************************************
;
TABLE1_DISPLAY:
                PUSH   HL
                LD     HL,TABLE1
                CALL   WRP
                POP    HL
                RET
;
;*****************************************
;                 TEST LCD                *
;          16 CHARACTERS * 1 LINE         *
;*****************************************
;
WRP:
                LD     A,00H
                CALL   GOTO
                CALL   WRLINE
                LD     A,40H
                CALL   GOTO
                CALL   WRLINE
                RET
```

```
;
;*********************************************
;                    CLOSE LCD                *
;*********************************************
CLSLCD:
            LD      A,00000001B         ;CLEAR ALL DISPLAY
            OUT     (PDATA),A
            CALL    READ
            RET
;
;*********************************************
;              WRITE LINE LCD 16*1            *
;*********************************************
WRLINE:
            LD      B,8
TEST11:
            LD      A,(HL)
            LD      D,A
            CALL    WRBYTE
            INC     HL
            DEC     B
            JP      NZ,TEST11
            RET
;
;*********************************************
;              WRITE BYTE SUB                 *
;*********************************************
WRBYTE:
            LD      A,D
```

```
                OUT     (PSING),A
                CALL    READ
                RET
;
;***********************************************
;               GOTO ADDR LCD                   *
;***********************************************
GOTO:
                SET     7,A
                OUT     (PDATA),A
                CALL    READ
                RET
;
;***********************************************
;               READ BUSY BIT                   *
;***********************************************
READ:
                IN      A,(PREAD)
                NOP
                BIT     7,A
                JP      NZ,READ
                RET
;
;***********************************************
;               SYS CALL GOTOLCD                *
;***********************************************
;  INPUT  C = GOTO
GOTOLCD:
                LD      A,C
                CALL    GOTO
```

```
                RET
;
;***********************************************
;               SYS CALL WBYTE                  *
;***********************************************
;  INPUT  C = ASCII
WBYTELCD:
                LD      D,C
                CALL    WRBYTE
                RET
;
;***********************************************
;               SYS CALL STGLCD                 *
;***********************************************
;  INPUT  DB XX-00H
LCDSTG:
                POP     HL
WRLCDSTG1:
                LD      C,(HL)
                INC     HL
                XOR     A
                CP      C
                JP      Z,WRSTG2
                CALL    WBYTELCD
                JP      WRLCDSTG1
WRSTG2:
                PUSH    HL
                RET
```

```
;
;*********************************************
;               SYS CALL CLCD                 *
;*********************************************
;INPUT C = HEX
DSPCLCD:
            LD      A,C
            CALL    ASCHEX
            PUSH    DE
            LD      C,D
            CALL    WBYTELCD
            POP     DE
            LD      C,E
            CALL    WBYTELCD
            RET
;
;*********************************************
;               SYS CALL DELCD                *
;*********************************************
DSPDELCD:
            LD      C,D
            PUSH    DE
            CALL    DSPCLCD
            POP     DE
            LD      C,E
            CALL    DSPCLCD
            RET
```

```
;
;*********************************************
;                BINARY ACC TO ASCII          *
;*********************************************
;INPUT  = A
;OUTPUT = DE (ASCII)
;REG    = ADE
ASCHEX:
        PUSH   AF
        RRCA
        RRCA
        RRCA
        RRCA
        CALL   ASCH1
        LD     D,A
        POP    AF
ASCH1:
        AND    0FH
        ADD    A,90H
        DAA
        ADC    A,40H
        DAA
        LD     E,A
        RET
```

```
;
;***********************************************
;                   CHANNEL NUMBER              *
;***********************************************
SCAN:
            LD      A,(CHANNEL)
            CP      CHA_W
            JP      Z,CHAN_01
            CP      CHA_X
            JP      Z,CHAN_02
            CP      CHA_Y
            JP      Z,CHAN_03
            CP      CHA_Z
            JP      Z,CHAN_04
            JP      MAIN0
CHAN_01:
            LD      HL,ON_1
            LD      A,'X'
            LD      (CHANNEL),A
            JP      END_SCAN
CHAN_02:
            LD      HL,ON_2
            LD      A,'Y'
            LD      (CHANNEL),A
            JP      END_SCAN
CHAN_03:
            LD      HL,ON_3
            LD      A,'Z'
            LD      (CHANNEL),A
            JP      END_SCAN
```

```
CHAN_04:
                LD      HL,ON_4
                LD      A,'W'
                LD      (CHANNEL),A
END_SCAN:
                RET
;
;***********************************************
;               CLEAR_LED                       *
;***********************************************
CLEAR_LED:
                OUT0    (PA),A
                OUT0    (PB),A
                OUT0    (PC),A
                RET
;
;***********************************************
;           CHECK ENDING OF BLOCK               *
;***********************************************
CHECK_END:
                PUSH    AF
                LD      A,(COUNT_PUR)
                CP      02H
                JP      NZ,NO_END
                LD      IX,PAT_PUR
                LD      A,'P'
                CP      (IX+0)
                JP      NZ,NO_END
                LD      A,'U'
                CP      (IX+1)
```

```
                JP      NZ,NO_END
                LD      A,'R'
                CP      (IX+2)
                JP      NZ,NO_END
                LD      IY,FLAG1
                SET     2,(IY+0)            ;IT IS THE END
                JP      CHECK_OUT
NO_END:
                LD      IY,FLAG1
                RES     2,(IY+0)            ;IT IS NOT THE END
CHECK_OUT:
                POP     AF
                RET
;
;***********************************************
;                       CHECK NUM?              *
;***********************************************
; IN:  A
; OUT: A,FLAG1 BIT 0
;    0 = NUM
;    1 = CHARACTER
CHECK_NUM:
                PUSH    IX
                CP      '.'
                JP      Z,ITS_NUM
                CP      ':'
                JP      Z,ITS_COLON
                CP      '/'
                JP      Z,ITS_NUM
                CP      '0'
```

```
        JP      Z,ITS_NUM
        CP      '1'
        JP      Z,ITS_NUM
        CP      '2'
        JP      Z,ITS_NUM
        CP      '3'
        JP      Z,ITS_NUM
        CP      '4'
        JP      Z,ITS_NUM
        CP      '5'
        JP      Z,ITS_NUM
        CP      '6'
        JP      Z,ITS_NUM
        CP      '7'
        JP      Z,ITS_NUM
        CP      '8'
        JP      Z,ITS_NUM
        CP      '9'
        JP      Z,ITS_NUM
        LD      IX,FLAG1
        SET     0,(IX+0)            ;IT'S NOT NUMBER
        JP      GET_OUT
ITS_COLON:
        PUSH    BC
        LD      C,A
        EX      AF,AF'
        LD      A,C
        LD      (DE),A
        INC     DE
        LD      A,(HL)
```

```
                LD      (DE),A
                INC     DE
                INC     HL
                LD      A,(HL)
                LD      (DE),A
                INC     DE
                INC     HL
                LD      A,(HL)
                LD      (DE),A
                INC     DE
                INC     HL
                LD      A,(HL)
                LD      (DE),A
                INC     HL
                LD      A,C
                EX      AF,AF'
                POP     BC
ITS_NUM:
                LD      IX,FLAG1
                RES     0,(IX+0)            ;IT'S NUMBER
GET_OUT:
                POP     IX
                RET
;
;***********************************************
;               GENERATE SOUND                *
;***********************************************
SOUND_END:
                PUSH    DE
                PUSH    BC
```

```
                PUSH   HL
                LD     D,05
LOX:
                LD     B,50
LOOXX:
                LD     C,50
                LD     HL,50          ;GENERATE SOUND ONE TIME
                CALL   SOUND
                DEC    B
                JR     NZ,LOOXX
                CALL   DELAY1
                DEC    D
                JR     NZ,LOX
                POP    HL
                POP    BC
                POP    DE
                RET
;
;*********************************************
;                GENERATE SOUND                *
;*********************************************
SOUND_ONE:
                PUSH   DE
                PUSH   BC
                PUSH   HL
                LD     D,01
LOX1:
                LD     B,80
LOOXX1:
                LD     C,50
```

```
                LD      HL,50           ;GENERATE SOUND ONE TIME
                CALL    SOUND
                DEC     B
                JR      NZ,LOOXX1
                CALL    DELAY1
                DEC     D
                JR      NZ,LOX1
                POP     HL
                POP     BC
                POP     DE
                RET
;
;***********************************************
;       GENERATE TONE TO SPEAKER                *
;***********************************************
; REG   C=FREQ
;     HL=LENGTH
SOUND:
                PUSH    HL
                PUSH    DE
                PUSH    AF
                PUSH    BC
SOUND1:
                LD      D,0
                LD      A,0FFH
                CALL    SOUND2
                XOR     A
                CALL    SOUND2
                DEC     D
                JR      NZ,SOUND1
```

```
                POP     BC
                POP     AF
                POP     DE
                POP     HL
                RET

SOUND2:
                OUT     (61H),A
                LD      E,C
SOUND3:
                DEC     HL
                LD      A,H
                OR      L
                JR      NZ,SOUND4
                INC     D
SOUND4:
                DEC     E
                JR      NZ,SOUND3
                RET
;
;*********************************************
;                TX ASCII ONE BYTE              *
;*********************************************
; INPUT  C
; OUTPUT -
; REG   AC
TXBYTE:
                PUSH    AF
TX0:
                IN0     A,(STAT0)
```

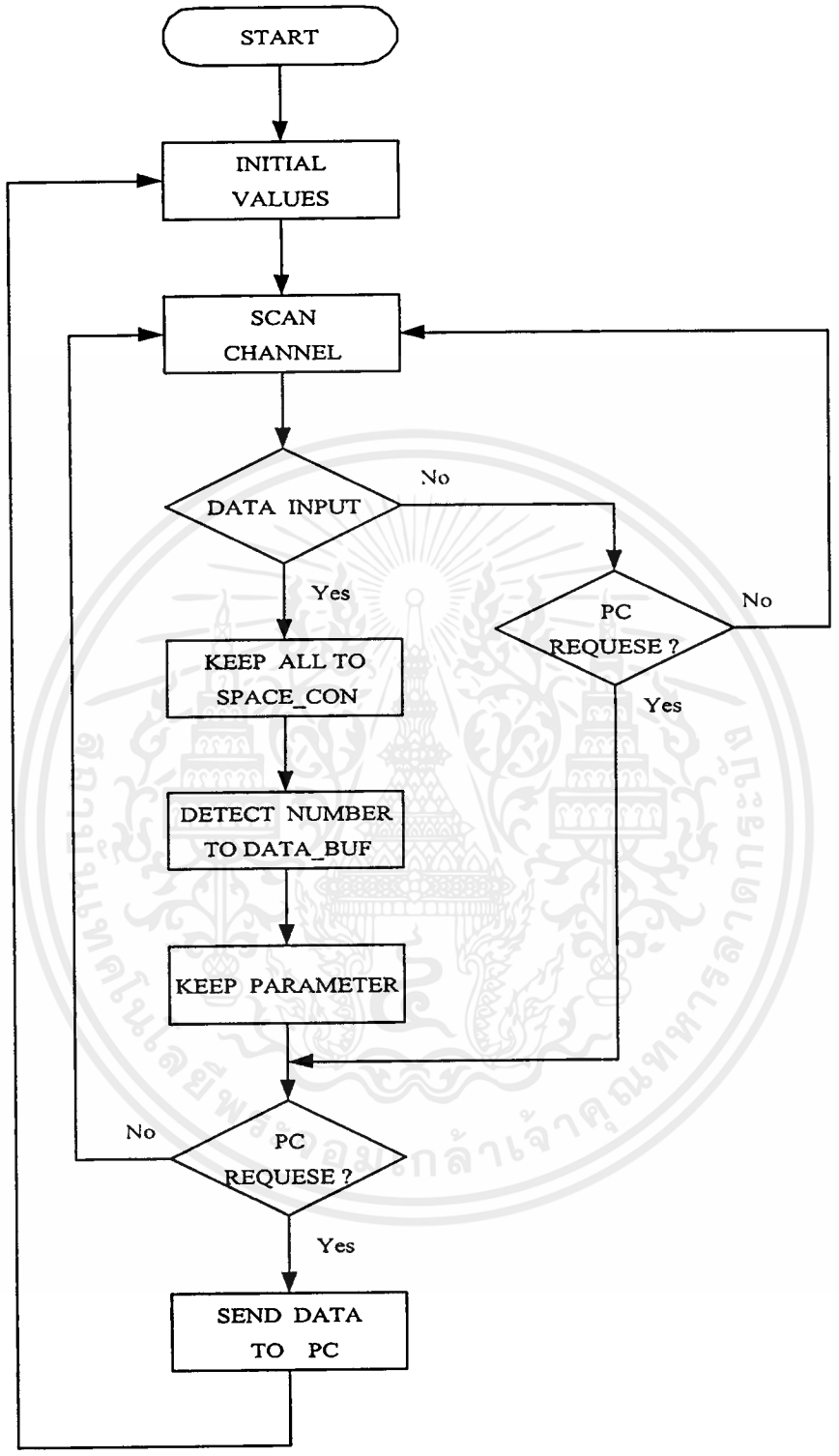
```
                BIT     1,A
                JP      Z,TX0
                OUT0    (TDR0),C
                POP     AF
                RET
;
;***********************************************
;                 DELAY HL = 8000H                *
;***********************************************
DELAY1:
                PUSH    HL
                PUSH    AF
                LD      HL,8000H            ;DELAY
LOOP1:
                DEC     HL
                LD      A,H
                OR      L
                JP      NZ,LOOP1
                POP     AF
                POP     HL
                RET
;
;***********************************************
;                 GENERATE SOUND START            *
;***********************************************
SOUND_START:
                PUSH    BC
                PUSH    HL
                LD      B,128
```

```
LO:
            LD      C,200
            LD      HL,500          ;GENERATE SOUND ONE TIME
            CALL    SOUND
            DEC     B
            JR      NZ,LO
            POP     HL
            POP     BC
            RET
;
;***********************************************
;           DELAY HL=0FFFFH                 *
;***********************************************
SDELAY:
            PUSH    AF
            PUSH    BC
            PUSH    HL
            EXX
            LD      BC,01FH
SDEL1:
            LD      HL,0
SDEL2:
            DEC     HL
            LD      A,L
            OR      H
            JP      NZ,SDEL2
            DEC     BC
            LD      A,B
            OR      C
            JP      NZ,SDEL1
```

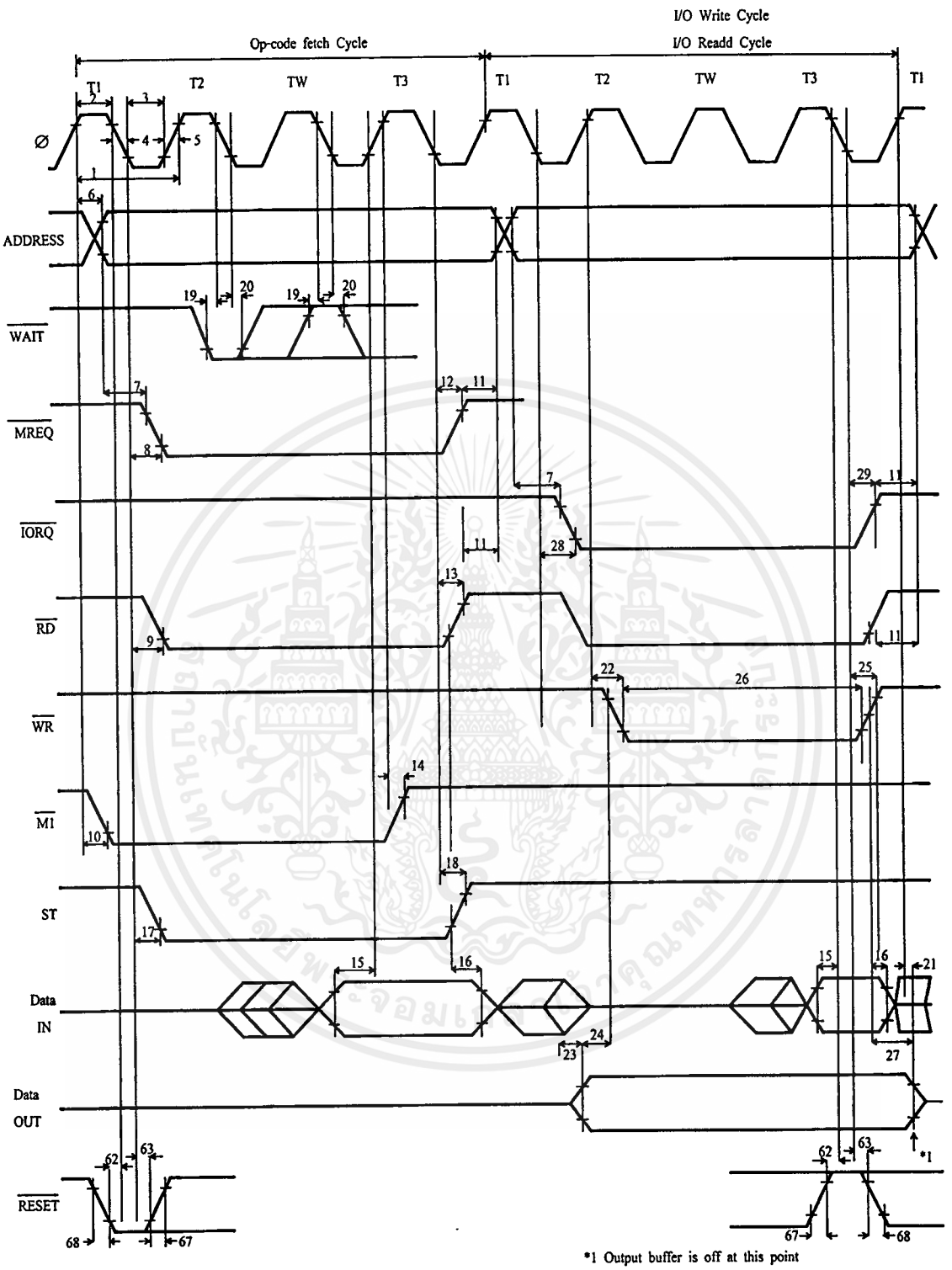
```
                EXX
                POP     HL
                POP     BC
                POP     AF
                RET

;
;***********************************************
;                   END SUBRUTINE                 *
;***********************************************

ON_1:           .DB     0FEH,0EEH       ;ENABLE CHANNEL 1
ON_2:           .DB     0FDH,0DDH       ;ENABLE CHANNEL 2
ON_3:           .DB     0FBH,0BBH       ;ENABLE CHANNEL 3
ON_4:           .DB     0F7H,077H       ;ENABLE CHANNEL 4

TABLE1:         .DB     "DATA ACQUISITION"
TABLE2:         .DB     "RECEIVING  O.K. "
TABLE3:         .DB     " PLEASE WIAT... "
TABLE4:         .DB     "TRANSMITTING... "

BL_MUL:         .DB     00H,01H,02H,03H,04H,05H,06H,07H,08H
BL_MUL1:        .DB     00H,02H,04H,06H,08H,0AH,0CH,0EH,10H,12H
CH_TAB:         .DB     0FEH,0FDH,0FBH,0F7H,0FFH,0EEH,0DDH,0BBH,077H,0FFH
BBR_DAT:        .DB     08H,018H,01EH,024H,02AH,030H,036H,00H,00H,010H,020H,030H,
                        040H,050H;DATA FOR SET BLANK&LED 24 KBYTES BLANK

                .ORG    SPACE_CON
DATA_CON:       .RS     03FFH           ;FOR CONVERT NUMBER
```

```
                .ORG    SPACE_BUF
FLAG1:          .RS     1       ;FOR CHECKING NUMBER?
COUNT_CHA:      .RS     1       ;COUNTER FOR INSERT";"
COUNT_PUR:      .RS     1       ;COUNTER OF P,U,R
COUNT_YOR:      .RS     1       ;COUNTER OF Y O R
PAT_PUR:        .RS     3       ;FOR CHECKING CONTINUE OF P U R
PAT_YOR:        .RS     3       ;FOR CHECKING CONTINUE OF Y O R
AD_END:         .RS     2       ;END OF GROUP DATA
AD_ENDB0:       .RS     2       ;END OF BLANK0 BEFORE MOVE
AD_ENDBX:       .RS     2       ;END OF SPACE_CON BEFORE MOVE
BBR_BUF:        .RS     1       ;NUMBER OF BBR
BL_COUNT:       .RS     20      ;
CHANNEL:        .RS     1       ;NUMBER CHANNEL

                .ORG    DATA_BUF
DATA_IN:        .RS     06000H      ;24KB FOR DATA
END_PRO:        .END
```

ภาคผนวก ข.

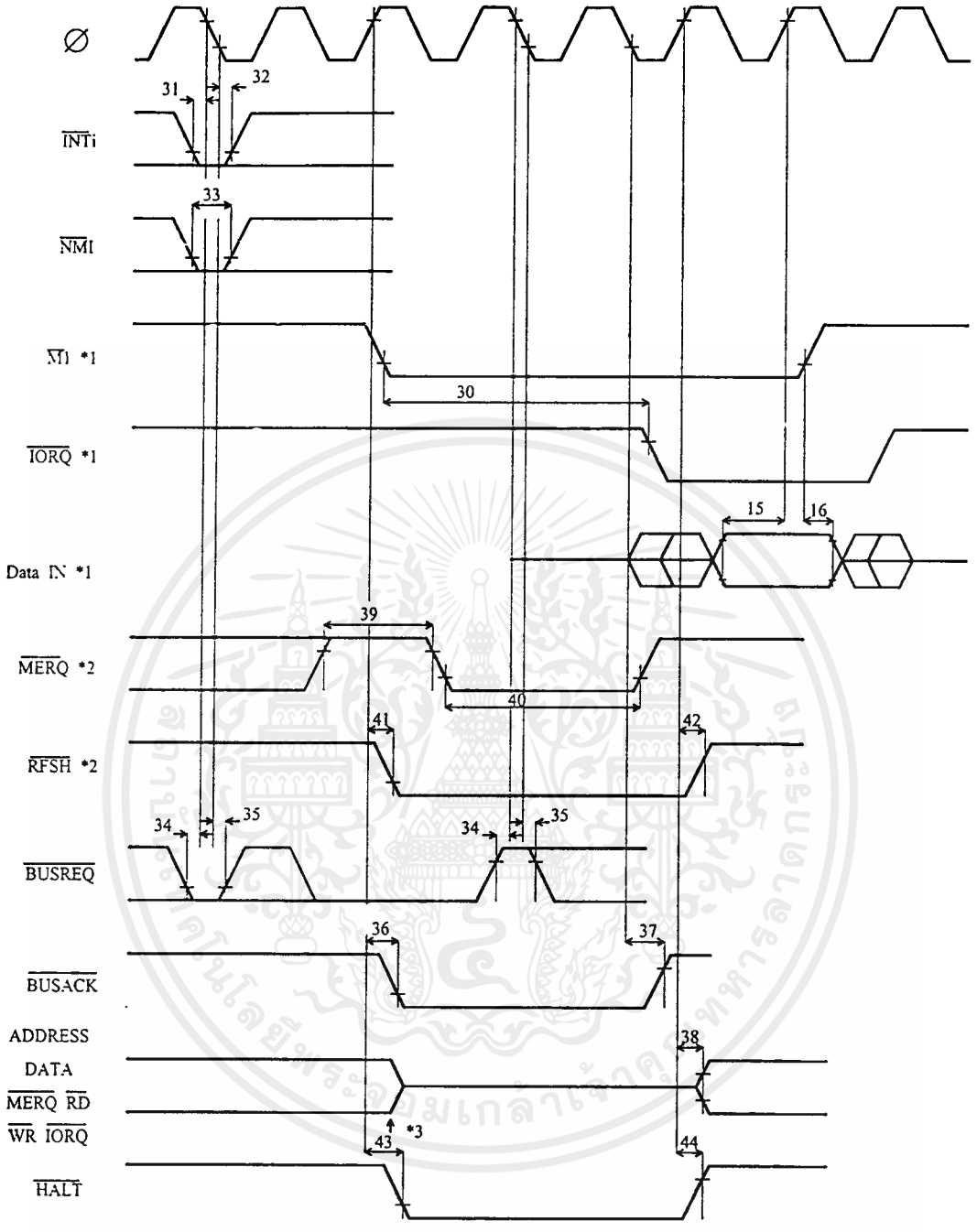


โฟลว์ชาร์ตแสดงการทำงานเครื่องเก็บข้อมูล



CPU Timing

- Op -code fetch Cycle
- I/O write Cycle
- I/O Read Cycle

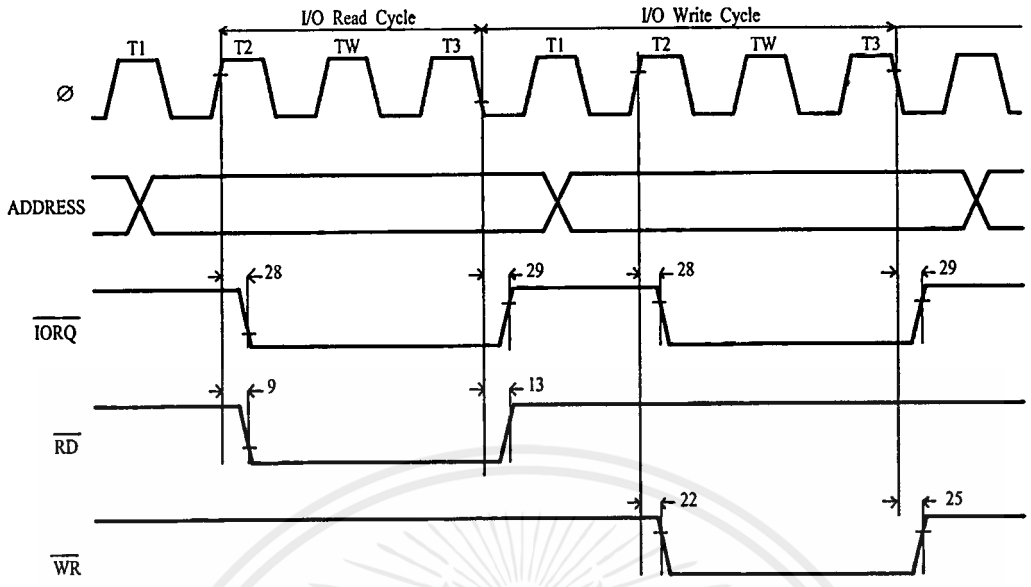


*1 during INTo acknowledge cycle

*2 during refresh cycle

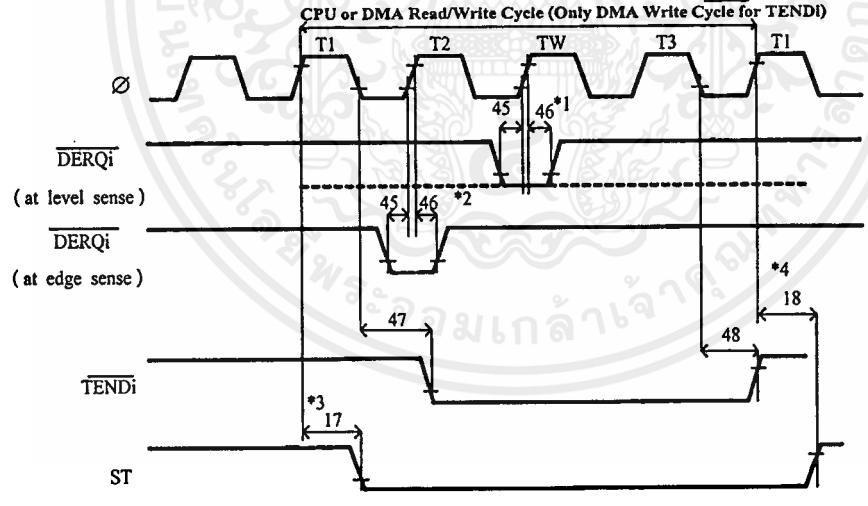
*3 Output buffer is off at this point

CPU Timing



CPU Timing (IOC=0)

[I/O Read Cycle
I/O Write Cycle]



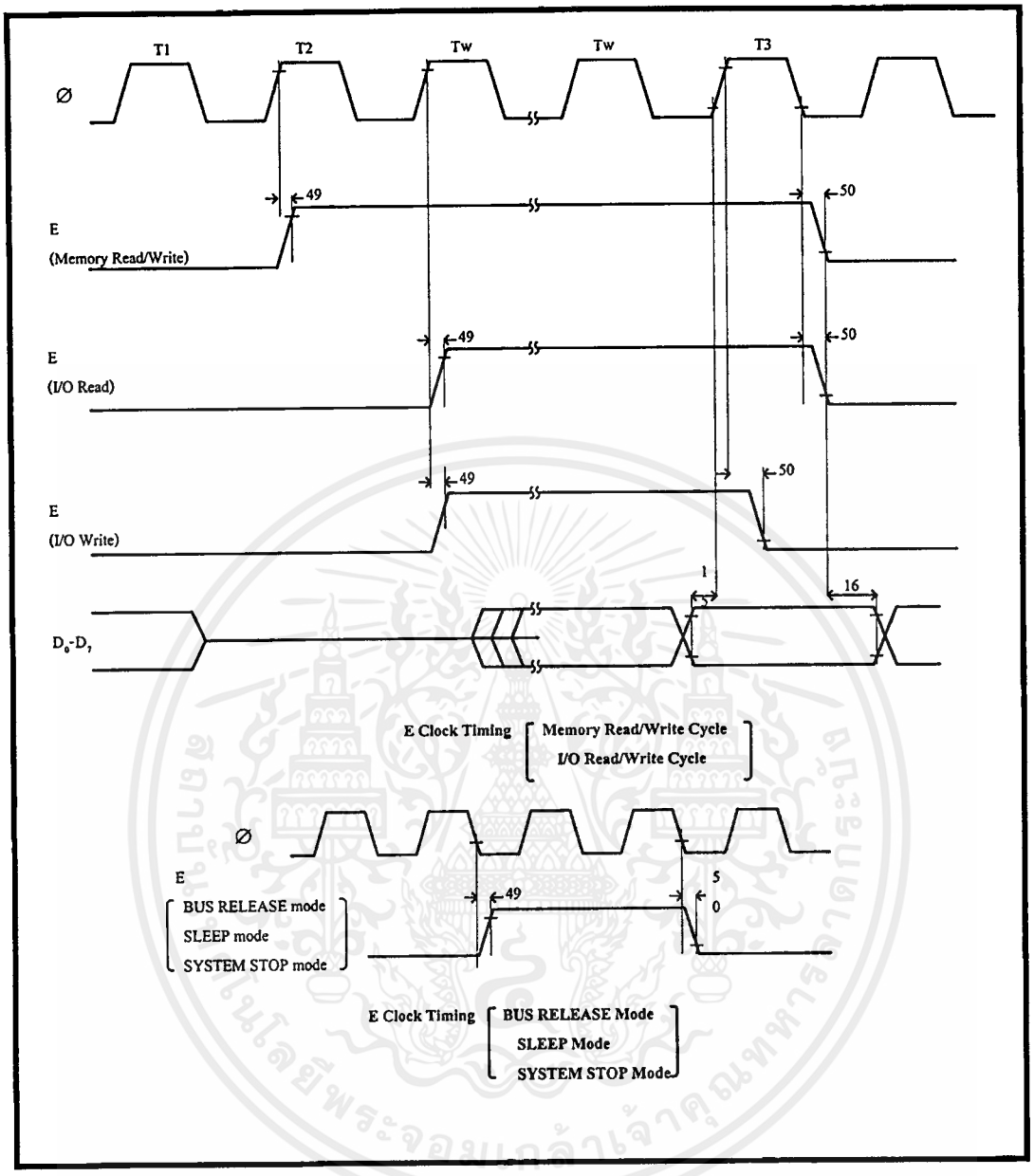
DMA Control Signals

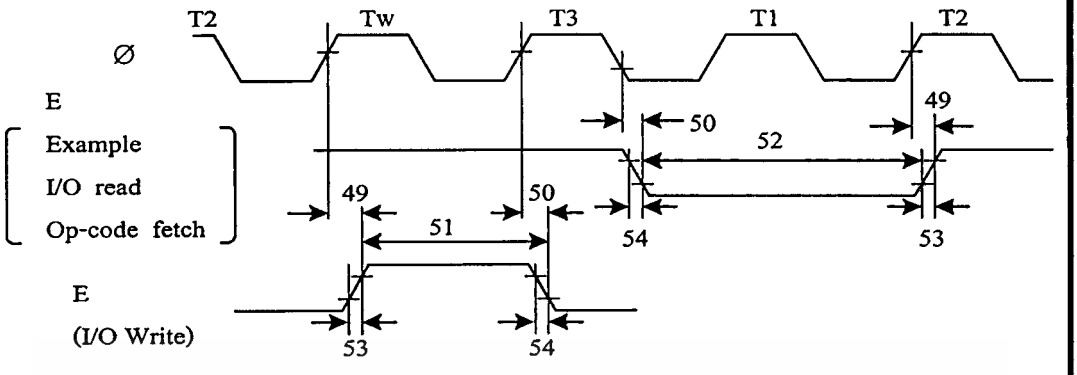
*1 t_{DRQS} and t_{DRQH} are specified for the rising edge of clock follow by T_3.

*2 t_{DRQS} and t_{DRQH} are specified for the rising edge of clock.

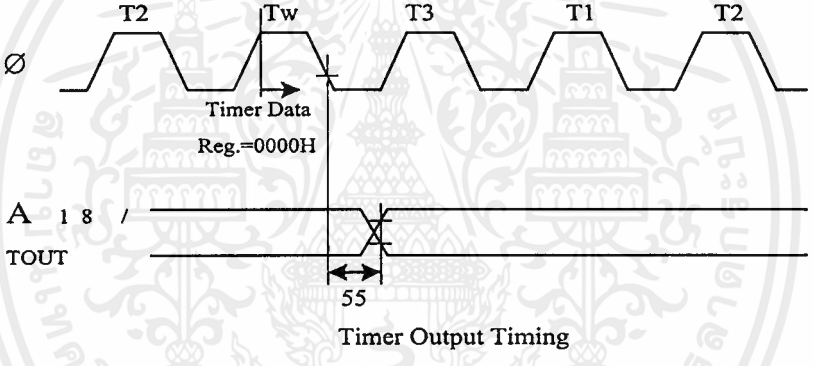
*3 DMA cycle starts.

*4 CPU cycle starts.

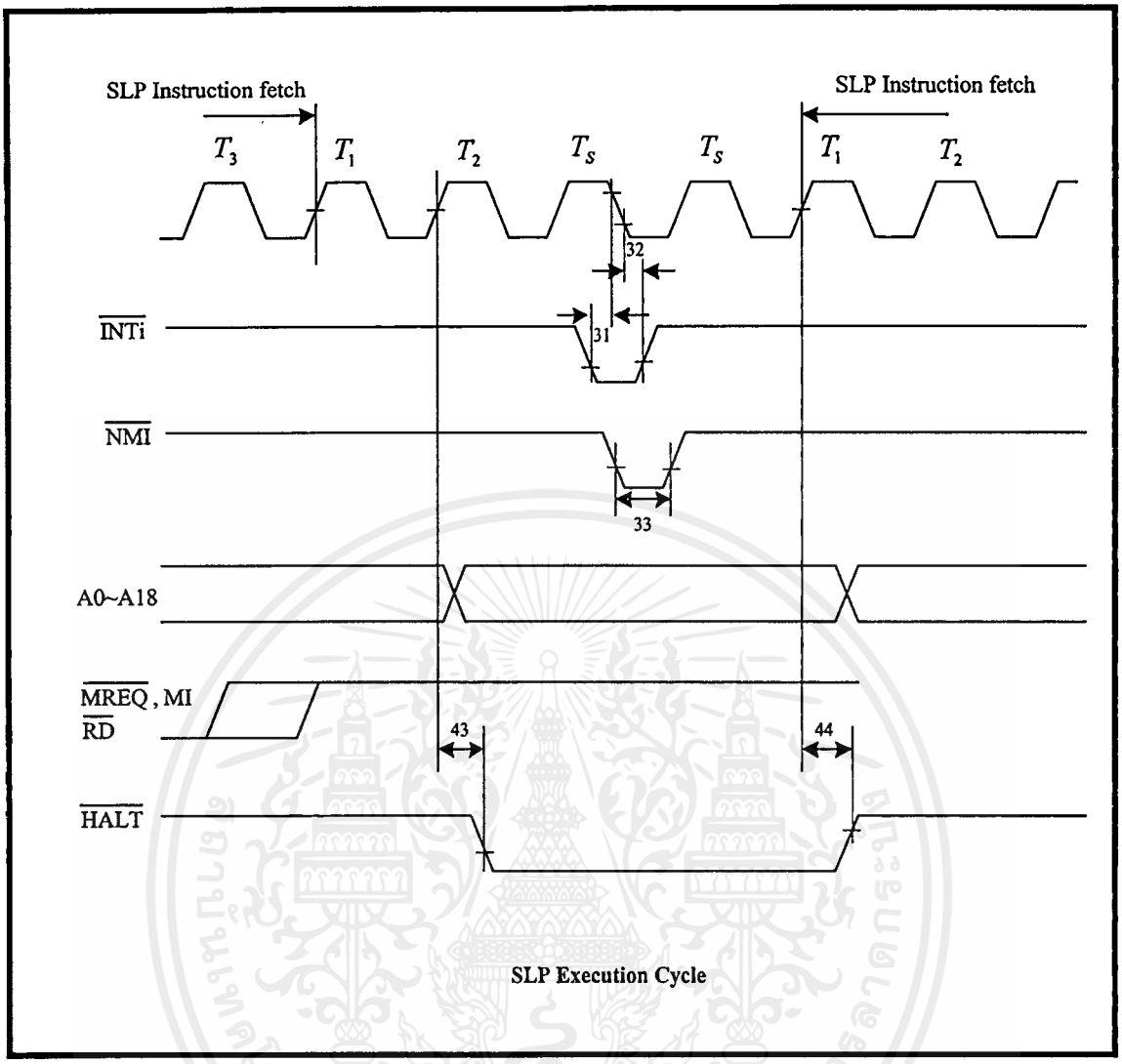




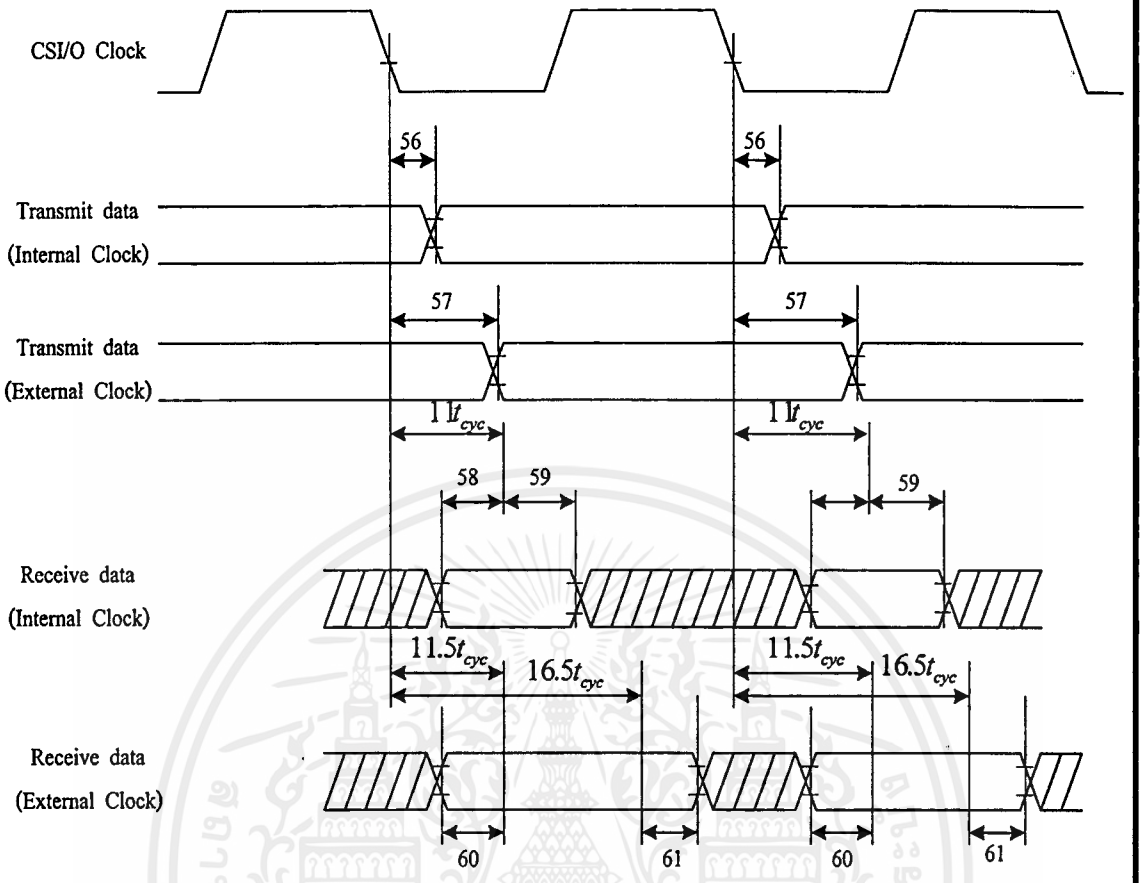
E clock Timing (Minimum timing example of P WEL and PWEH)



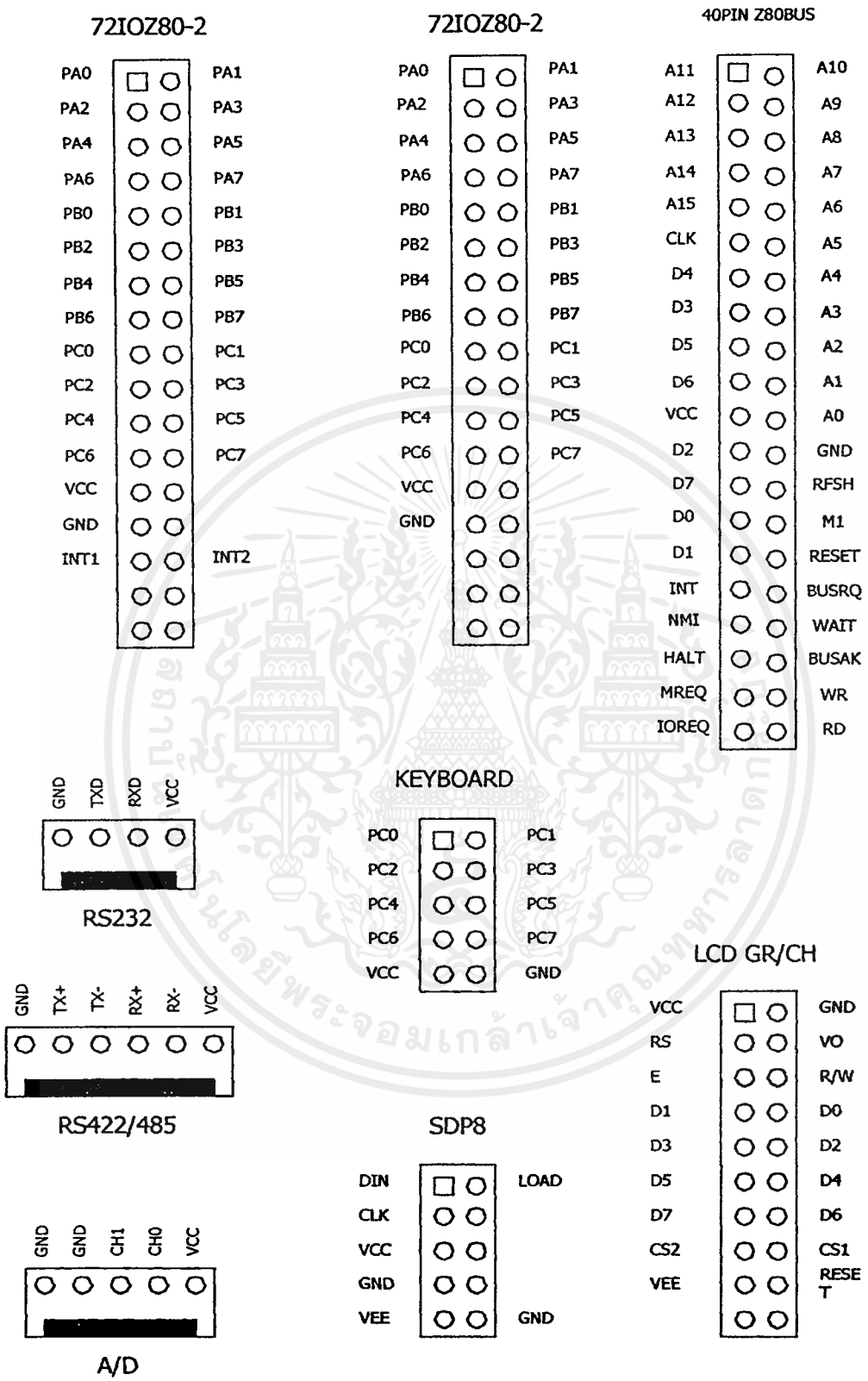
Timer Output Timing



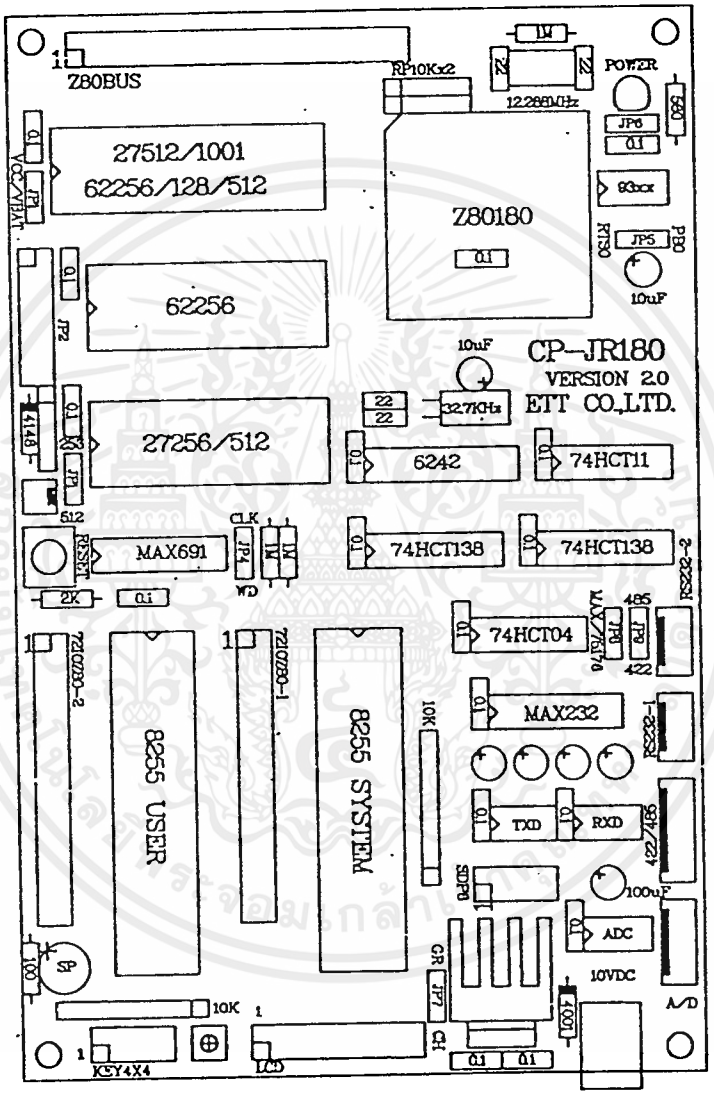
SLP Execution Cycle

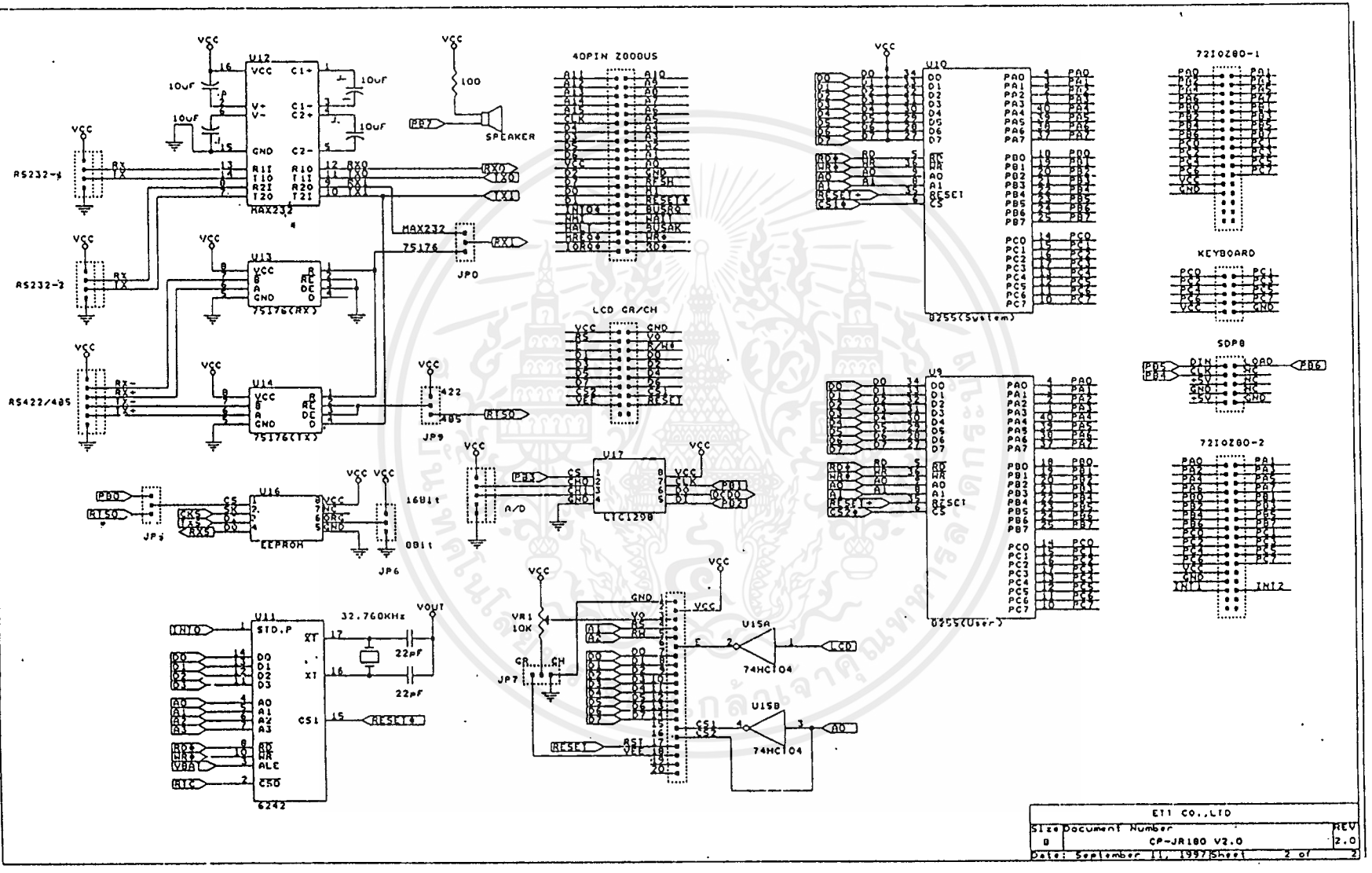


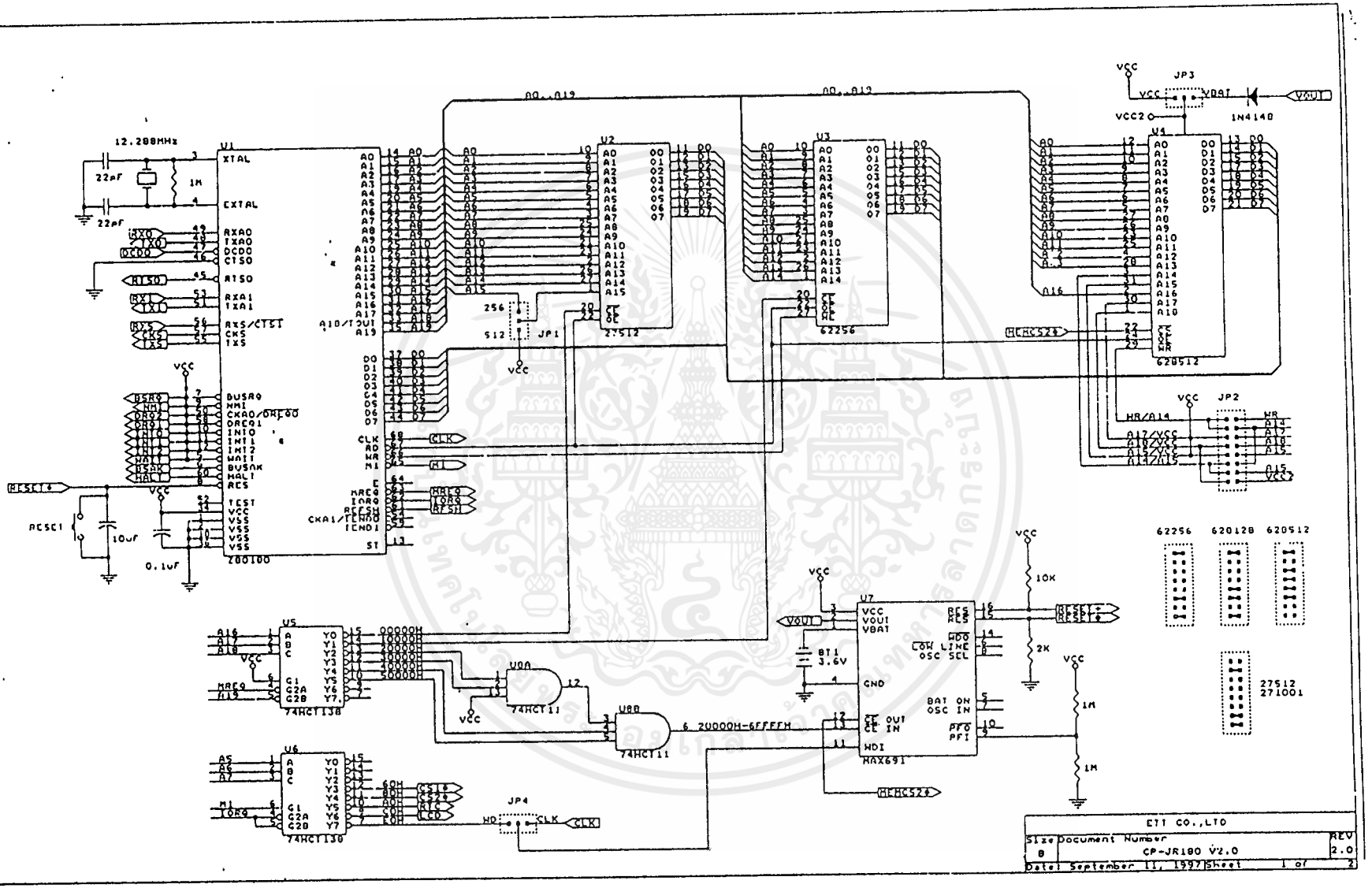
CSI/O Receive/Transmit Timing



รูปแสดงคอนเน็คเตอร์ต่าง ๆ




RS232-1
RS232-2
RS422/485
MAX232
75176
JP0
JP9
JP5
JP6
JP7
EEPROM
LTC1298
A/D
16bit
8bit
40PIN Z80BUS
LCD CR/CH
SPEAKER
100
10uF
U12
U13
U14
U16
U17
U11
U10
U9
8255(System)
8255(User)
72I0Z80-1
72I0Z80-2
KEYBOARD
SDP8
32.760KHz
22pF
VR1
10K
U15A
U15B
74HC104
6242
VCC
GND
RESET
ET1 CO.,LTD
Size
Document Number
CP-JR180 V2.0
REV
2.0
Date: September 11, 1997
Sheet
2 of
2


12.288MHz
22pF
1M
22pF
Z80180
XTAL
EXTAL
RESET
RESET
10uF
0.1uF
VCC
A0..A19
27512
62256
620512
JP1
JP2
JP3
JP4
1N4148
VOUT
VBAT
74HCT138
74HCT11
MAX691
BT1
3.6V
10K
2K
1M
20000H-6FFFFH
62256
620128
620512
27512
271001
ETT CO.,LTD
CP-JR180 V2.0
REV
2.0
Size
B
Date: September 11, 1997
Sheet
1 of
2

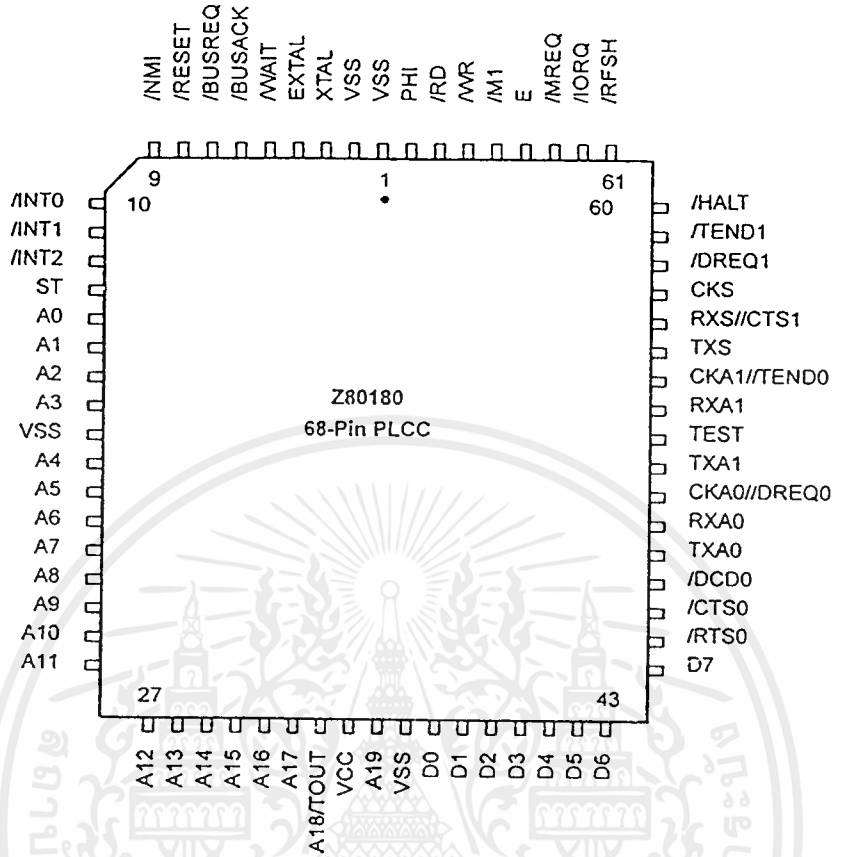
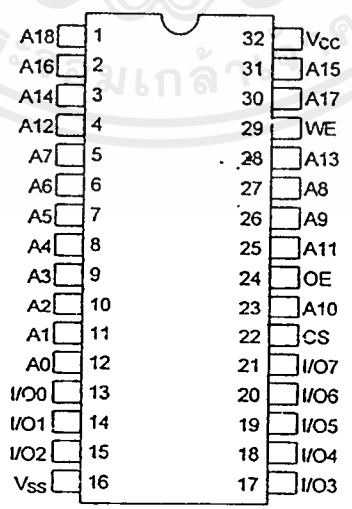


Figure 3. Z80180 68-Pin PLCC Pin Configuration

HM628512P/LP Series
HM628512FP/LFP Series

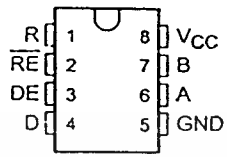


(Top View)

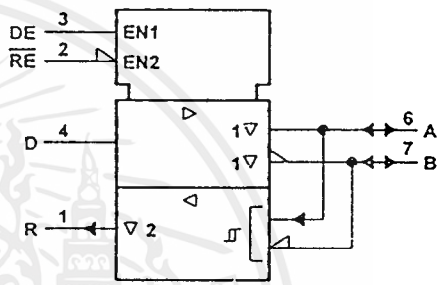
SLLS101A – JULY 1985 – REVISED MAY 1995

- Bidirectional Transceivers
- Meet or Exceed the Requirements of ANSI Standards EIA/TIA-422-B and RS-485 and ITU Recommendations V.11 and X.27
- Designed for Multipoint Transmission on Long Bus Lines in Noisy Environments
- 3-State Driver and Receiver Outputs
- Individual Driver and Receiver Enables
- Wide Positive and Negative Input/Output Bus Voltage Ranges
- Driver Output Capability . . . ±60 mA Max
- Thermal Shutdown Protection
- Driver Positive and Negative Current Limiting
- Receiver Input Impedance . . . 12 kΩ Min
- Receiver Input Sensitivity . . . ±200 mV
- Receiver Input Hysteresis . . . 50 mV Typ
- Operate From Single 5-V Supply

D OR P PACKAGE
(TOP VIEW)

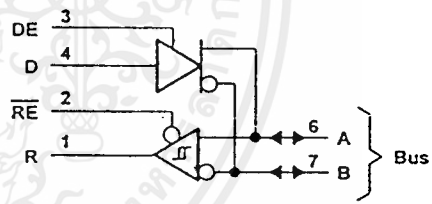


logic symbol†



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

logic diagram (positive logic)



description

The SN65176B and SN75176B differential bus transceivers are monolithic integrated circuits designed for bidirectional data communication on multipoint bus transmission lines. They are designed for balanced transmission lines and meet ANSI Standards EIA/TIA-422-B and RS-485 and ITU Recommendations V.11 and X.27.

The SN65176B and SN75176B combine a 3-state differential line driver and a differential input line receiver, both of which operate from a single 5-V power supply. The driver and receiver have active-high and active-low enables, respectively, that can be externally connected together to function as a direction control. The driver differential outputs and the receiver differential inputs are connected internally to form differential input/output (I/O) bus ports that are designed to offer minimum loading to the bus whenever the driver is disabled or V_{CC} = 0. These ports feature wide positive and negative common-mode voltage ranges making the device suitable for party-line applications.

Function Tables

DRIVER

INPUT D	ENABLE DE	OUTPUTS A	OUTPUTS B
H	H	H	L
L	H	L	H
X	L	Z	Z

RECEIVER

DIFFERENTIAL INPUTS A – B	ENABLE $\overline{RE}$	OUTPUT R
$V_{ID} \geq 0.2$ V	L	H
$-0.2\text{ V} < V_{ID} < 0.2$ V	L	?
$V_{ID} \leq -0.2$ V	L	L
X	H	Z
Open	L	H

H = high level, L = low level, ? = indeterminate, X = irrelevant, Z = high impedance (off)

19-0054; Rev 6; 7/95

MAXIM

Microprocessor Supervisory Circuits

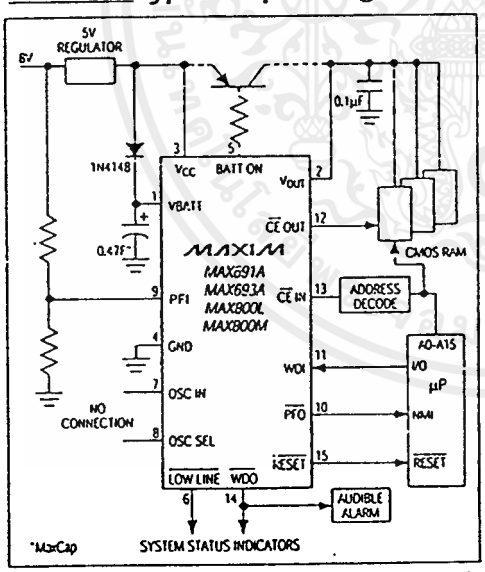
General Description

The MAX691A/MAX693A/MAX800L/MAX800M microprocessor (µP) supervisory circuits are pin-compatible upgrades to the MAX691, MAX693, and MAX695. They improve performance with 30µA supply current, 200ms typ reset active delay on power-up, and 6ns chip-enable propagation delay. Features include write protection of CMOS RAM or EEPROM, separate watchdog outputs, backup-battery switchover, and a $\overline{RESET}$ output that is valid with V_{CC} down to 1V. The MAX691A/MAX800L have a 4.55V typical reset-threshold voltage, and the MAX693A/MAX800M's reset threshold is 4.4V typical. The MAX800L/MAX800M guarantee power-fail accuracies to ±2%.

Applications

Computers
Controllers
Intelligent Instruments
Automotive Systems
Critical µP Power Monitoring

Typical Operating Circuit



Features

- 200ms Power-OK/Reset Timeout Period
- 1µA Standby Current, 30µA Operating Current
- On-Board Gating of Chip-Enable Signals, 10ns Max Delay
- MaxCap™ or SuperCap™ Compatible
- Guaranteed $\overline{RESET}$ Assertion to V_{CC} = 1V
- Voltage Monitor for Power-Fail or Low-Battery Warning
- Power-Fail Accuracy Guaranteed to ±2% (MAX800L/M)
- Available in 16-Pin Narrow SO and Plastic DIP Packages

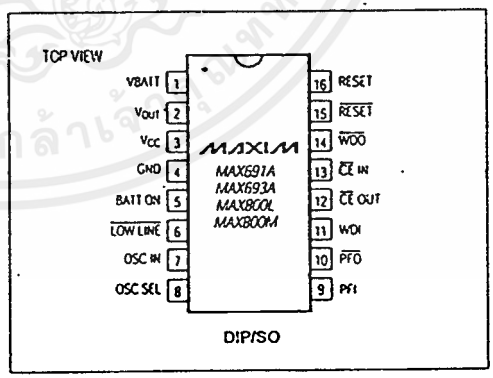
Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX691ACPE	0°C to +70°C	16 Plastic DIP
MAX691ACSE	0°C to +70°C	16 Narrow SO
MAX691ACWE	0°C to +70°C	16 Wide SO
MAX691AC/D	0°C to +70°C	Dice*
MAX691AEPE	-40°C to +85°C	16 Plastic SO
MAX691AESE	-40°C to +85°C	16 Narrow SO
MAX691AEWE	-40°C to +85°C	16 Wide SO
MAX691AEJE	-40°C to +85°C	16 CERDIP
MAX691AMJE	-55°C to +125°C	16 CERDIP

Ordering Information continued on last page.

** Dice are specified at T_A = +25°C.*

Pin Configuration



™ SuperCap is a registered trademark of Baknor Industries. ™ MaxCap is a registered trademark of The Carborundum Corp.

LINEAR TECHNOLOGY

LTC1286/LTC1298

Micropower Sampling 12-Bit A/D Converters In SO-8 Packages

FEATURES

- 12-Bit Resolution
- 8-Pin SOIC Plastic Package
- Low Cost
- Low Supply Current: 250µA Typ.
- Auto Shutdown to 1nA Typ.
- Guaranteed ±3/4LSB Max DNL
- Single Supply 5V to 9V Operation
- On-Chip Sample-and-Hold
- 60µs Conversion Time
- Sampling Rates:
  - 12.5 ksps (LTC1286)
  - 11.1 ksps (LTC1298)
- I/O Compatible with SPI, Microwire, etc.
- Differential Inputs (LTC1286)
- 2-Channel MUX (LTC1298)
- 3V Versions Available: LTC1285/LTC1288

APPLICATIONS

- Battery-Operated Systems
- Remote Data Acquisition
- Battery Monitoring
- Handheld Terminal Interface
- Temperature Measurement
- Isolated Data Acquisition

DESCRIPTION

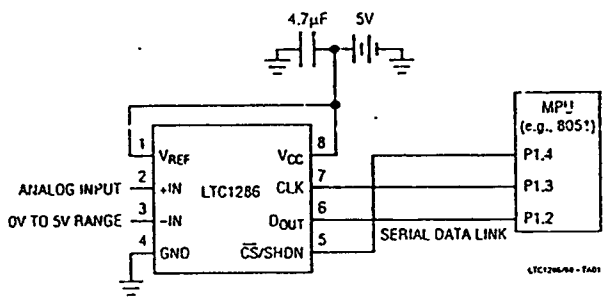
The LTC1286/LTC1298 are micropower, 12-bit, successive approximation sampling A/D converters. They typically draw only 250µA of supply current when converting and automatically power down to a typical supply current of 1nA whenever they are not performing conversions. They are packaged in 8-pin SO packages and operate on 5V to 9V supplies. These 12-bit, switched-capacitor, successive approximation ADCs include sample-and-holds. The LTC1286 has a single differential analog input. The LTC1298 offers a software selectable 2-channel MUX.

On-chip serial ports allow efficient data transfer to a wide range of microprocessors and microcontrollers over three wires. This, coupled with micropower consumption, makes remote location possible and facilitates transmitting data through isolation barriers.

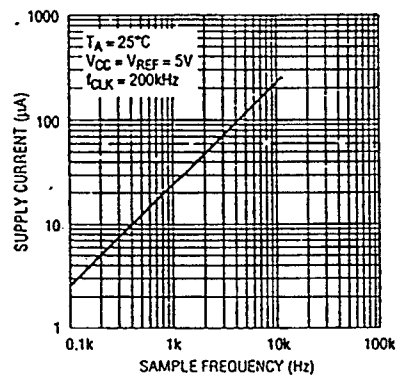
These circuits can be used in ratiometric applications or with an external reference. The high impedance analog inputs and the ability to operate with reduced spans (to 1.5V full scale) allow direct connection to sensors and transducers in many applications, eliminating the need for gain stages.

TYPICAL APPLICATIONS

25µW, SO-8 Package, 12-Bit ADC
Samples at 200Hz and Runs Off a 5V Supply



Supply Current vs Sample Rate



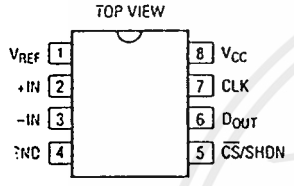
ABSOLUTE MAXIMUM RATINGS (Notes 1 and 2)

Supply Voltage (V_{CC}) to GND 12V
Voltage
- Analog and Reference ..... −0.3V to V_{CC} + 0.3V
- Digital Inputs ..... −0.3V to 12V
- Digital Output ..... −0.3V to V_{CC} + 0.3V

Power Dissipation 500mW
Operating Temperature Range
- LTC1286C/LTC1298C ..... 0°C to 70°C
- LTC1286I/LTC1298I ..... −40°C to 85°C

Storage Temperature Range ..... −65°C to 150°C
Lead Temperature (Soldering, 10 sec.) ..... 300°C

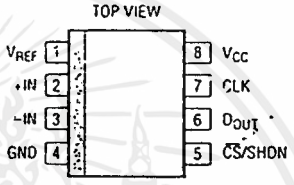
PACKAGE/ORDER INFORMATION



N8 PACKAGE
8-LEAD PLASTIC DIP

T_{JMAX} = 150°C, θ_{JA} = 130°C/W

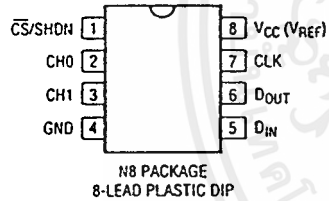
ORDER PART NUMBER
LTC1286CN8
LTC1286IN8



S8 PACKAGE
8-LEAD PLASTIC SOIC

T_{JMAX} = 150°C, θ_{JA} = 175°C/W

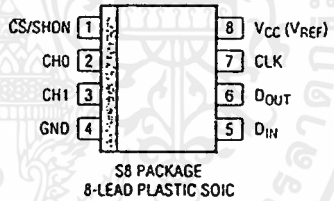
ORDER PART NUMBER
LTC1286CS8
LTC1286IS8
PART MARKING
1286C
1286I



N8 PACKAGE
8-LEAD PLASTIC DIP

T_{JMAX} = 150°C, θ_{JA} = 130°C/W

ORDER PART NUMBER
LTC1298CN8
LTC1298IN8



S8 PACKAGE
8-LEAD PLASTIC SOIC

T_{JMAX} = 150°C, θ_{JA} = 175°C/W

ORDER PART NUMBER
LTC1298CS8
LTC1298IS8
PART MARKING
1298C
1298I

Consult factory for military grade parts.

RECOMMENDED OPERATING CONDITIONS

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
V_{CC}	Supply Voltage (Note 3)	LTC1286 LTC1298	4.5 4.5		9.0 5.5	V V
f_{CLK}	Clock Frequency	V_{CC} = 5V	(Note 4)		200	kHz
t_{CYC}	Total Cycle Time	LTC1286, f_{CLK} = 200kHz LTC1298, f_{CLK} = 200kHz	80 90			µs µs
t_{hDI}	Hold Time, D_{IN} After CLK↑	V_{CC} = 5V	150			ns
$t_{su\overline{CS}}$	Setup Time $\overline{CS}$↓ Before First CLK↑ (See Operating Sequence)	LTC1286, V_{CC} = 5V LTC1298, V_{CC} = 5V	2 2			µs µs
t_{suDI}	Setup Time, D_{IN} Stable Before CLK↑	V_{CC} = 5V	400			ns
t_{WHCLK}	CLK High Time	V_{CC} = 5V	2			µs
t_{WLCLK}	CLK Low Time	V_{CC} = 5V	2			µs
$t_{WH\overline{CS}}$	$\overline{CS}$ High Time Between Data Transfer Cycles	V_{CC} = 5V	2			µs
$t_{WL\overline{CS}}$	$\overline{CS}$ Low Time During Data Transfer	LTC1286, f_{CLK} = 200kHz LTC1298, f_{CLK} = 200kHz	75 85			µs µs

MICROCHIP

93AA46/56/66

1K/2K/4K 1.8V CMOS Serial EEPROM

FEATURES

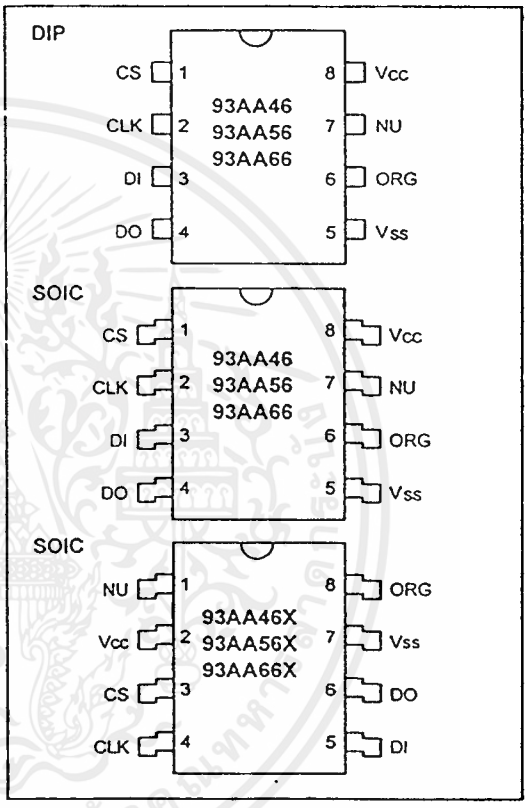
- Single supply with programming operation down to 1.8V
- Low power CMOS technology
  - 70 µA typical active READ current at 1.8V
  - 2 µA typical standby current at 1.8V
- ORG pin selectable memory configuration
  - 128 x 8 or 64 x 16 bit organization (93AA46)
  - 256 x 8 or 128 x 16 bit organization (93AA56)
  - 512 x 8 or 256 x 16 bit organization (93AA66)
- Self-timed ERASE and WRITE cycles (including auto-erase)
- Automatic ERAL before WRAL
- Power on/off data protection circuitry
- Industry standard 3-wire serial I/O
- Device status signal during ERASE/WRITE cycles
- Sequential READ function
- **10,000,000 ERASE/WRITE cycles guaranteed on 93AA56 and 93AA66**
- **1,000,000 E/W cycles guaranteed on 93AA46***
- Data retention > 200 years
- 8-pin PDIP/SOIC (SOIC in JEDEC and EIAJ standards)

DESCRIPTION

The Microchip Technology Inc. 93AA46/56/66 are 1K, 2K and 4K low voltage serial Electrically Erasable PROMs. The device memory is configured as x8 or x16 bits depending on the ORG pin setup. Advanced CMOS technology makes these devices ideal for low power non-volatile memory applications. The 93AA Series is available in standard 8-pin DIP and surface mount SOIC packages. The rotated pin-out 93AA46X/56X/66X are offered in the "SN" package only.

*Future: 10,000,000 cycles guaranteed

PACKAGE TYPE



BLOCK DIAGRAM