PROGRAMMABLE POWER SUPPLY

โดย
นายเริงศักดิ์ ทาระพันธ์
นายอำนวย ทองนิตย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาเทคนิคอุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2534

ชื่อเรื่อง ปริญญานิพนธ์ PROGRAMMABLE POWER SUPPLY

ชื่อผู้เขียน 1. นาย เริงศักดิ์ ทาระพันธ์
2. นาย อำนวย ทองนิตย์

ปริญญานิพนธ์ อุตสาหกรรมศาสตร์บัณฑิต สาขาเทคโนโลยี-
อิเล็คทรอนิคส์ สถาบันเทคโนโลยีพระจอมเกล้า
เจ้าพระคุณทหาร ลาดกระบัง

ได้รับการพิจารณาอนุมัติให้นับเป็นส่วนหนึ่งของการศึกษา
ตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต
สาขาวิชา เทคโนโลยีอิเล็คทรอนิคส์

คณะกรรมการตรวจสอบปริญญานิพนธ์

..... ประธานกรรมการ
( )

..... กรรมการ
( )

..... กรรมการ
( )

วันที่..... เดือน..... พ.ศ. .....

PROGRAMMBLE POWER SUPPLY

นาย เริงศักดิ์ ทาระพันธ์
นาย อำนวย ทองนิตย์

อาจารย์ที่ปรึกษา อ. ชวลิต เบญจางคประเสริฐ

บทคัดย่อ

หลักการของเครื่อง Programmble Power Supply นี้อาศัยหลักการของแหล่งจ่ายไฟตรงที่ควบคุมแรงดันได้ตั้งแต่ 0 ถึง 18 โวลท์ บวก , ลบ พร้อมทั้งให้กระแสสูงสุด 3 แอมป์ วงจรแหล่งจ่ายไฟตรง 5 โวลท์ ที่คงที่เพื่อนำไปใช้ประโยชน์ใช้ไปเลี้ยง ไอซี ทีทีแอล พร้อมให้กระแสสูงสุดได้ 3 แอมป์

การทำงานของแหล่งจ่ายไฟตรงใช้ CPU ในการประมวลกลางซึ่งใช้ร่วมกับ Key Board เป็นตัวกำหนดแรงดันต่าง ๆ แล้วแสดงออกบนหน้าจอ 7 - Segment ถ้าต้องการก็ให้กด Enter จะได้โวลท์ออกทางเอาท์พุทภายในเครื่อง และจะรอการกดปุ่ม ON / OFF เพื่อจ่ายแรงดันออกเอาท์พุทภายนอกอีกที ซึ่งจะได้เอาท์พุทตามที่เรากำหนดใน Key Board ตามต้องการ

เครื่องนี้มีความสามารถพิเศษอีกอย่างหนึ่ง คือ สามารถเพิ่มหรือลดแรงดันได้ทีละ 10 มิลิโวลท์

PROGRAMMABLE POWER SUPPLY

MR. ROENGSAK TARAPANT

MR. AMNUAY THONGNIT

ADVISOR MR. CHAWALIT BENJANGKRAPRASERT

ABSTRACT

THE MAIN PRINCIPLE OF THIS PROGRAMMBLE POWER SUPPLY INSPECTION MACHINE OF REGULATOR CAN BE SUPPLY VOLTAGE SINCE ZERO VOLTS TO EIGHTEEN VOLTS , POSITIVE , NEGATIVE WITH SUPPLY AMPERE MAXIMUM 3 AMPS. THE REGULATOR CIRCUIT 5 VOLTS ON FIXE , BENIFIT SUPPLY IC TTL AND SUPPLY AMPERE MAXIMUM 3 AMP.

THE OPERATE OF REGULATOR USE CPU IN CENTRAL PROCESSING WHICH USED WITH TO KEY BOARD IS SET EACH THE VOLTS SHOW ON DISPLAY 7 - SEGMENT IF WANT PRESS ENTER BUTTOM WILL BE VOLTAGE OUT TO INTERNAL OUTPUT OF MACHINE AND IT WILL DELAY THE PRESS ON / OFF BUTTOM , FOR SUPPLY VOLTAGE OUT TO EXTERNAL OUTPUT A LITTLE MORE. WHICH IT WILL GIVE OUTPUT TO FOLLOW US TO CONTROL ON KEY BOARD.

THIS MACHINE HAVE SPECIALLY FURTHER MORE ARE CAN BE INCREAMENT OR DECREAMENT VOLTAGE AS 10 MILI - VOLTS.

คำนำ

ในปัจจุบันการนำเครื่องมือต่าง ๆ นำไปใช้งาน ส่วนมากจะมีวงจรอิเล็คทรอนิคส์เป็นตัวควบคุม ซึ่งเกี่ยวข้องกับเครื่องมือ อุปกรณ์ทางอิเล็คทรอนิคส์ และ คุณสมบัติของเครื่องมือทางอิเล็คทรอนิคส์ นั้นขึ้นอยู่กับปัจจัยหลายอย่างด้วยกัน โดยเฉพาะอย่างยิ่งแหล่งจ่ายไฟตรง (D.C. Power Supply) ที่ใช้เลี้ยง หรือ จ่ายไฟให้กับ ไอซี จำพวกตระกูล Logic เช่น ไอซี ทีทีแอล

เนื้อหาสาระของ Power II เป็นชุดควบคุมแหล่งจ่ายไฟด้วยไมโครโปรเซสเตอร์ (Programmable Power Supply) ซึ่งรวบเอาทฤษฎี อุปกรณ์ และ วงจรต่างๆ ซึ่งจะทำให้สะดวกเวลาใช้งาน ขั้นตอนการประดิษฐ์ การออกแบบอาศัยหลักการทางวงจรดิจิตอล วงจรอนาล็อก วงจรควบคุมแหล่งจ่ายไฟ ซึ่งวงจรต่าง ๆ สามารถทำงานช่วยกันได้ โดยใช้ไมโครโปรเซสเตอร์ เป็นตัวควบคุมการทำงาน

ชุดแหล่งจ่ายไฟแบบควบคุมด้วยไมโครโปเซสเซอร์ (Programmable Power Supply) สามารถปรับแรงดันได้ ตั้งแต่ 0 ถึง 18 Volts และ 0 ถึง -18 Volts กระแส 3 AMP ซึ่งสามารถเพิ่มหรือลด Volts ได้ STEP STEP ละ 10 มิลิโวลท์ (0.01 Volts) และ มีชุด Supply 5 Volts เป็นแหล่งจ่ายไฟคงที่มีไว้สำหรับ IC ชนิด TTL สามารถ SHORT OUT PUT ได้ โดยไม่ทำให้วงจรเสียหาย

การใช้งานแหล่งจ่ายไฟ 0, +18, -18 Volts เพียงแต่ป้อนแรงดันที่ต้องการผ่านคีย์บอร์ด LRD สามารถนำไปใช้งานได้ทันที และมีวงจรป้องกันในกรณี เกิดการลัดวงจร (OVERLOAD Protection)

สารบัญ

บทที่ 1

บทนำ

ปริญญานิพนธ์ เล่มนี้มีส่วนประกอบด้วย 4 ส่วน ดังนี้

ส่วนแรก วงจรควบคุม ประกอบด้วย ซีพียู ทำหน้าที่ ประมวลผล ทำการติดต่อระหว่าง เอ-ทู-ดี คอนเวอร์ทเตอร์ , คีย์บอร์ด กับ ดี-ทู-เอ คอนเวอร์ทเตอร์ ดีสเพลย์

ส่วนที่สอง ส่วนอินพุท ประกอบด้วย เอ-ทู-ดี คอนเวอร์ทเตอร์ , คีย์บอร์ด เป็น การสั่งงานให้ ซีพียู แปลงรหัสการทำงานต่าง ๆ เช่น กดหมายเลข 1 ซีพียู จะทำการประเมินผลเพื่อให้แสดงเป็นเลข " 1 "

ส่วนที่สาม ส่วนเอาท์พุท ประกอบด้วย ดี-ทู-เอ คอนเวอร์ทเตอร์ , ดีสเพลย์ เป็นการแสดงผล 7 - SEGMONT ตามที่ ซีพียู ทำการประเมินผลแล้ว ออกทางเอาท์พุท

ส่วนที่สุดท้าย วงจรแหล่งจ่ายไฟตรงที่รักษาระดับแรงดันได้ สามารถจ่ายไฟตรงได้ ตั้งแต่ 0-18 โวลท์ บวก , ลบ พร้อมทั้งให้กระแสสูงสุด 3 แอมป์

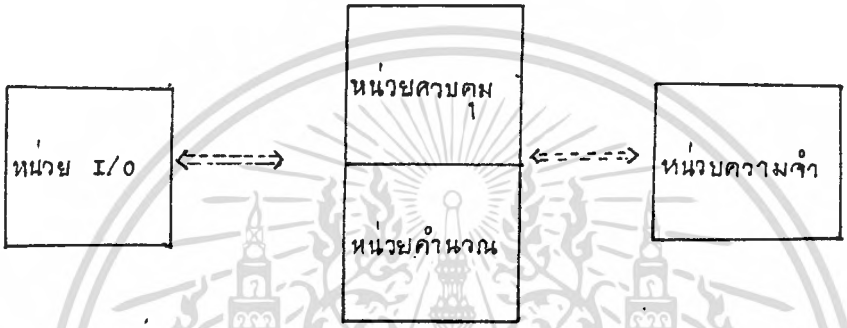
วงจรแหล่งจ่ายไฟตรง 5 โวลท์ ที่คงที่ เพื่อนำประโยชน์ไปเลี้ยง ไอซี ทีที่แอล พร้อมให้กระแสสูงสุด ได้ 3 แอมป์

คณะผู้จัดทำ

บทที่ 2. 1. ไมโครโปรเซสเซอร์และไมโครคอมพิวเตอร์

1.1 ไมโครโปรเซสเซอร์ในระบบคอมพิวเตอร์

ไมโครคอมพิวเตอร์ คือ คอมพิวเตอร์ที่มี ไมโครโปรเซสเซอร์เป็นหน่วยประ
ผลกลาง (Central Processing Unit) ที่ทำงานร่วมกับหน่วยอื่นอีก สองหน่วย ดังรูป



รูปที่ 1.1 ส่วนประกอบพื้นฐานของระบบไมโครคอมพิวเตอร์

ประกอบด้วยส่วนต่าง ๆ ดังนี้ :-

- หน่วยควบคุม
- หน่วยคำนวณ
- หน่วยความจำ
- หน่วยอินพุท เอาท์พุท

ส่วนของวงจรอิเลคทรอนิคส์ที่ทำหน้าที่ในแต่ละหน่วยเราจะเรียกว่า ฮาร์ดแวร์ (HardWare) ซึ่งฮาร์แวร์เหล่านี้จะทำตามคำสั่งของผู้ใช้เป็นชุดของโปรแกรม (Program) ในหน่ว
ความจำ หน้าที่ของแต่ละส่วนก็จะต่างกันไป

ลักษณะหน้าที่ในแต่ละส่วนเป็นดังนี้ :-

หน้าที่ควบคุม จัดเป็นหัวใจของระบบเลย เนื่องจากเป็นส่วนที่จะส่งสัญญาณควบคุมส่วนอื่น ๆ ให้ทำงานตามฟังก์ชั่นต่าง ๆ เช่น การอ่าน,เขียนหน่วยความจำ การติดต่อกับ อุปกรณ์ภายนอก อย่างถูกต้อง โดยส่วนควบคุมนี้จะทำงานตามคำสั่งที อ่านมา จากโปรแกรม และการอ่านคำสั่งมาปฏิบัติแต่ละครั้งสามารถแยกได้เป็น 2 สภาวะคิ

สภาวะ เฟทช์ คือ กระบวนการที่หน่วยควบคุมทำงานชุดคำสั่งจากหน่วยความจำมาถอดระห เพื่อรอการปฏิบัติ

สภาวะ เอ็กซ์คิ้ว คือ กระบวนการหน่วยควบคุมได้ปฏิบัติตามหน่วยนั้น ๆ ในรายละเอียดของการปฏิบัติตามคำสั่งที่ได้กล่าวถึงในหัวข้ออื่นอีกครั้ง

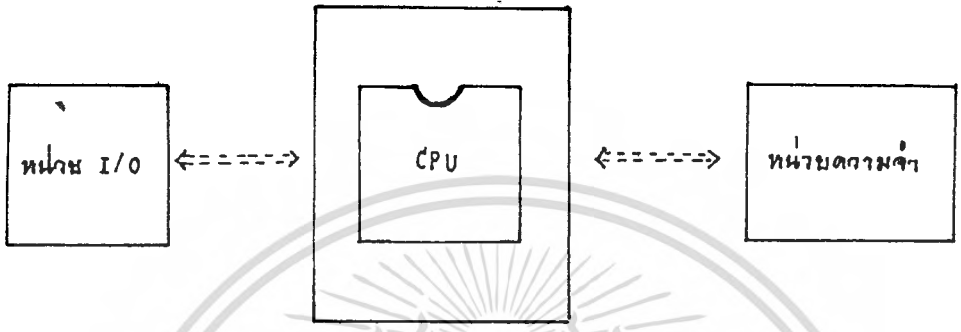
หน่วยคำนวน ทำหน้าที่ทางคณิตศาสตร์และโลจิกโดยตรง เช่นการ บวก ลบ คูณ หาร ตรวจสอบบิท โดยจะกระทำที่หน่วยความจำชั่วคราว ซึ่งโดยทั่วไปจะมีขนาด ไบท์ ที่เรียกว่า Accumulator rogister ซึ่งเมื่อกระทำเสร็จแล้วมักจ จะเก็บผลลัพท์ไว้ที่หน่วยชั่วคราวนี้

หน่วยความจำ เป็นที่เก็บโปรแกรมของผู้ใช้ คำสั่งต่าง ๆ ผลของคำสั่ง ซึ่งหน่วยความจำนี้ จะถูกติดต่อโดยส่วนควบคุม เพื่อทำการอ่านเขียนได้

หน่วย I/O เป็นส่วนของวงจร บัฟเฟอร์ ที่ให้หน่วยความจำสามารถที่จะติดต่อกับโลก ภายนอกได้นั่นหมายความว่าหน่วยควบคุมจะส่งสัญญาณใด ๆ ไปสู่โลกภายนอ จะต้องผ่านส่วนนี้ก่อน ในทางกลับกันสัญญาณจากข้างนอก จะเข้าสู่ระบบได้ก็ ผ่านวงจรส่วนนี้เช่นกัน ตัวอย่างที่ติอต่อกับโลกภายนอก ได้แก่ จอภา เครื่องพิมพ์ โมเดม เป็นต้น

จากรูปที่ 1 นั้นเป็นพื้นฐานของระบบคอมพิวเตอร์ จะเห็นว่าหน่วยควบคุมกับหน่วยคำน นั้นจะต้องทำงานร่วมกันเสมอ เพื่อให้ได้ผลการทำงานไปสู่หน่วยความจำหรืออุปกรณ์ I/O ดังนั้นหากสามาร เอาหน่วยควบคุมกับหน่วยคำนวนไว้ในหน่วยเดียวกัน ก็จะทำให้ประหยัดอุปกรณ์เชื่อมต่อระหว่างหน่วยนั้ หมายความว่าความเร็วก็สูงขึ้น ซึ่งด้วยเทคโนโลยีปัจจุบันก็สามารถกระทำได้

จึงได้ยุบหน่วยทั้งสองนั้นลงเป็น ชิฟ (Ship) เพียงตัวเดียวเรียกว่า หน่วยประมาณผลกลาง หรือ CPU (Centrol Processing Unit) ดังรูปที่ 1.2



ดังรูปที่ 1.2

ส่วนของ CPU นี่เองที่เราจะเรียนรู้เพื่อเชื่อมต่อกับระบบพื้นฐานต่าง ๆ ในการนำไปใช้ประโยชน์ต่อไป ตัวชิฟ CPU เองได้มีการผลิตออกมาหลายเบอร์และหลายบริษัท คุณสมบัติของแต่ละเบอร์ก็มีข้อดีต่างกัน ซึ่งแยกเป็นประเภทใหญ่ ๆ ได้ 2 ประเภท คือ

- แบบไบโพลา ภายในประกอบด้วยทรานซิสเตอร์ ซึ่งจะทำงานด้วยกระแสสองประเภท คือ ทั้งบวกและลบ จึงเรียกว่า ไบโพลา โดยแบ่งย่อยตามลักษณะวงจรได้ดังนี้

ECL (Emitter Conpling Logic)

IIL (Intgrated Injection Logic)

TTL (Transistor Transistor Logic)

STTL (Schottky TTL)

พวกนี้คุณสมบัติมากในเรื่องความเร็ว แต่กินกระแสสูง

- แบบ MOS (Metal Oxide Semicondector) พวกนี้ภายในเป็นลักษณะของทรานซิสเตอร์ เช่นกันแต่เป็นแบบ MOSFET แบ่งตามลักษณะของวงจรเป็น

POMS (P-channel MOS)

NMOS (N-channel MOS)

CMOS (Complementary MOS)

พวกนี้คุณสมบัติที่เด่นมาก คือ กินกระแสต่ำ

ฉะนั้นในการใช้งานต้องเลือกแบบ และ คุณสมบัติที่ตรงตามความต้องการของงานที่เอาไปใช้ ส่วนในการศึกษานี้ จะได้เลือกแบบละเบอร์ที่นักศึกษาสามารถหาและปฏิบัติได้ง่ายทั้งข้อมูลและการทดลอง จึงได้เลือกเบอร์ Z-80 ของบริษัท Zilog เป็นไมโครโปรเซสเซอร์ขนาด 8 บิท ซึ่งเมื่อได้ศึกษาแล้วอย่างเข้าใจนักศึกษาก็สามารถที่จะทำความเข้าใจในเบอร์อื่นได้ไม่ยาก

1.2 ประเภทของคอมพิวเตอร์

คอมพิวเตอร์ในปัจจุบันแบ่งย่อยออกเป็นระดับตามต้องการใช้งานดังนี้ :-

1. ระบบคอมพิวเตอร์แบบแยกส่วน (Mini COM.)
2. ระบบคอมพิวเตอร์แบบเบ็ดเสร็จ (Mini Com.)
3. ระบบคอมพิวเตอร์แบบแผ่นวงจรเดียว (Single Board Com.)
4. ระบบคอมพิวเตอร์แบบชิฟเดียว (Single Ship Com.)

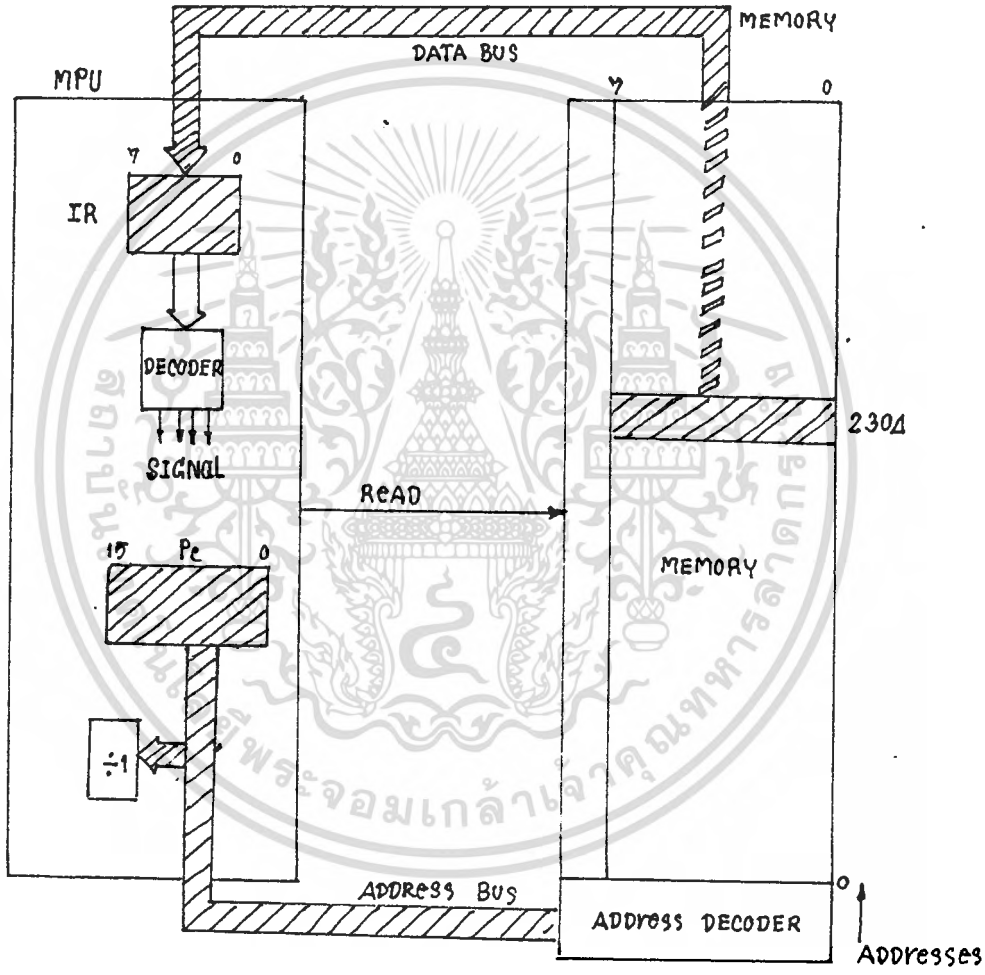
1. ระบบคอมพิวเตอร์แบบแยกส่วน จะแยกส่วนของเมนแฟมกับอุปกรณ์หน่วยความจำและอุปกรณ์ I/O ออกไปจากกันเมื่อเปลี่ยนแปลงเพิ่มเติมได้อย่างสะดวก โดยแต่ละหน่วยจะมีขนาดใหญ่และความสามารถสูงมาก โดยมากระบบนี้จะใช้งานที่เก็บข้อมูลเป็นจำนวนมาก เช่น ธนาคาร เป็นต้น

2. ระบบคอมพิวเตอร์แบบเบ็ดเสร็จ แบบนี้เห็นอยู่ทั่งไปเพราะปัจจุบันสามารถหามาใช้งานก็ง่ายและราคาก็ถูกลง เป็นแบบหน่วยความจำและอุปกรณ์ I/O ถูกบรรจุรวมอยู่ในเครื่องเดียวกันสามารถนำไปใช้งานได้โดยทันที แต่ความสามารถจะน้อยกว่าแบบแยกส่วน เนื่องจากมีขีดจำกัดของหน่วยความจำและอุปกรณ์ I/O

3. ระบบคอมพิวเตอร์แบบแผ่นวงจรเดียว แบบนี้จะมีหน่วยความจำและอุปกรณ์ I/O รวมอยู่บนแผ่นวงจรพิมพ์เพียงแผ่นเดียว ฉะนั้นขีดจำกัดในการใช้งานจะน้อยกว่าแบบเบ็ดเสร็จอีก แต่มีประโยชน์มากในการศึกษาการทำงานของระบบไมโครโปรเซสเซอร์ และการควบคุมอุปกรณ์ต่าง ๆ

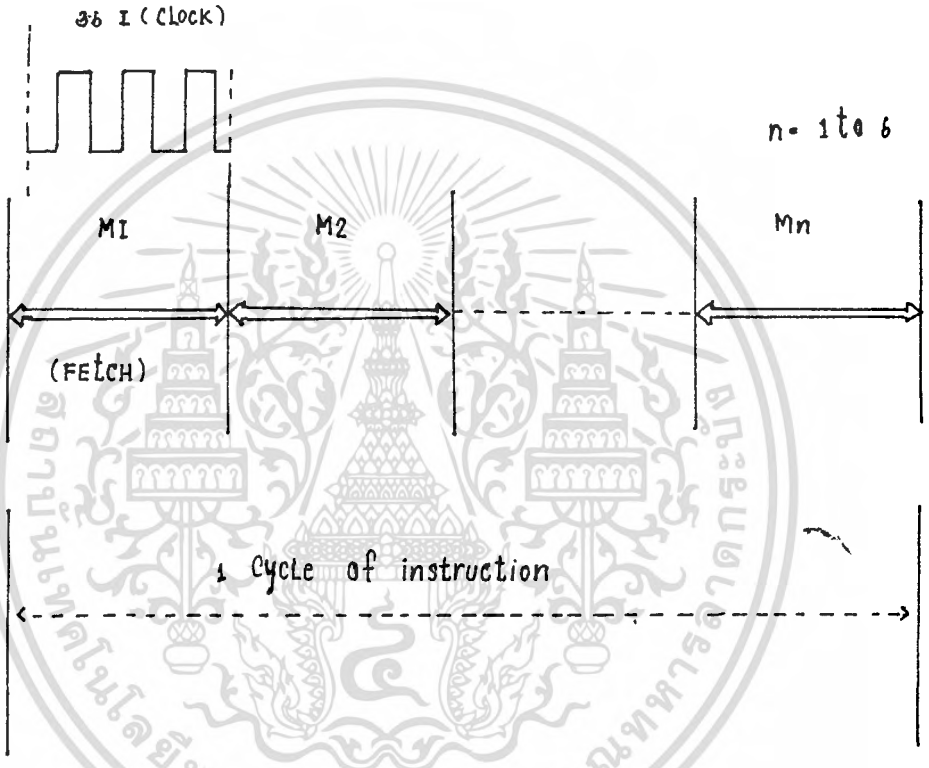
4. ระบบคอมพิวเตอร์แบบชิฟเดียว ด้วยเทคโนโลยีที่มากขึ้นของปัจจุบัน จึงได้มีการรวมเอาส่วนของหน่วยความจำและส่วนอุปกรณ์ I/O เข้าไว้ใน ชิฟ ไอซี เพียงตัวเดียว ทำให้ความสามารถโดยส่วนรวมจะน้อยลง แต่มีคุณสมบัติพิเศษเฉพาะตัวในด้านการนำไปใช้งานควบคุมต่าง ๆ มีประสิทธิภาพสูงขณะที่ขนาดเล็กลง

ดังที่ได้กล่าวมาแล้วทั้งหมด ระบบคอมพิวเตอร์แบบพิมพ์แผ่นเดียวจึงเหมาะแก่การศึกษามากที่สุดโดยการเรียน การสอนในวิชานี้ก็จะไปในเรื่องระบบคอมพิวเตอร์แบบวงจรพิมพ์แผ่นเดียวเป็นหลัก



รูปที่ 1.3 แสดงถึงการเดินทางของสัญญาณสภาวะการเฟทและเอ็กซีคิ้วคำสั่ง

รูปของคำสั่งแต่ละคำสั่งของไมโครโปรเซสเซอร์ จะประกอบด้วย แมทชีนไซเกิ้ลต่าง ๆ แต่ ต้องมีแมทชีนไซเกิ้ลของการเฟท (M1) เริ่มต้นก่อนเสมอ ในรูปที่ 1.4 ในแต่ละแมทชีนไซเกิ้ล ก็จะต้องใช้จำนวนสัญญาณนาฬิกาที่แตกต่างกันออกไปอีก และในแต่ละคำสั่งจะประกอบด้วย แมทชีนไซเกิ้ลต่าง ๆ รวมแล้วไม่เกิน 6 แมทชีนไซเกิ้ล (ของ Z-80) ซึ่งรายละเอียดจะได้ศึกษาดังต่อไป



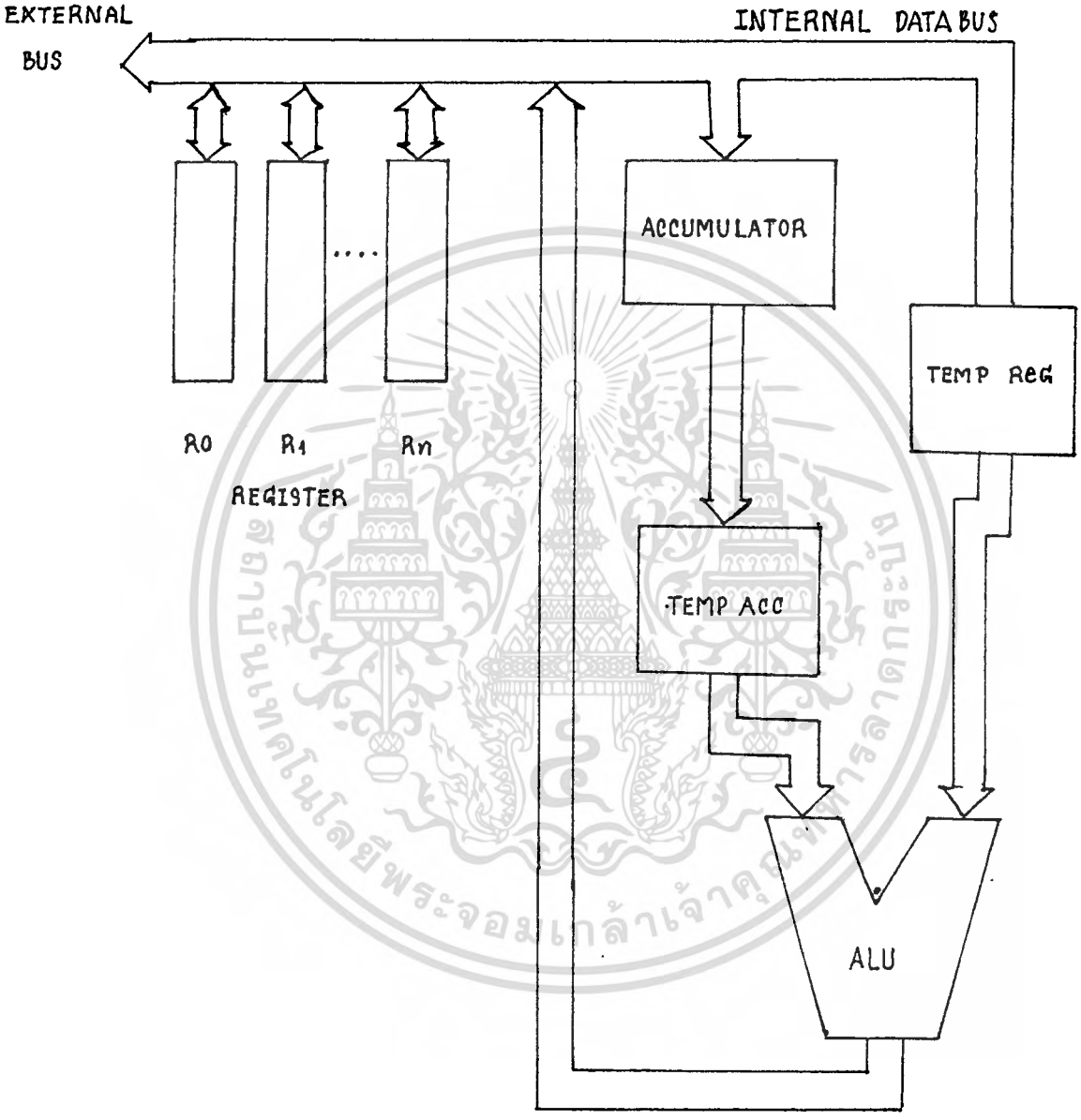
รูปที่ 1.4 ไดอะแกรมไซเกิ้ลคำสั่งของไมโครโปรเซสเซอร์

ตัวอย่างการเคลื่อนย้ายข้อมูลภายในไมโครโปรเซสเซอร์

กล่าวถึงรูปที่ 1.3 ,เป็นมาตราฐานของโครงสร้างภายในของไมโครโปรเซสเซอร์ง่าย ๆ ที่ประกอบด้วยส่วนคำนวน ALU, แอคคิวมูเลเตอร์ และรีจิสเตอร์ต่าง ๆ ,มีมีการคำนวน คือ ต้องนำข้อมูลใน register R0 มารวมกับข้อมูล register R1 และนำผลไปเก็บใน register R0 เช่นเดิม

R0 = R0 + R1

ปัญหาที่เกิดจาก timing ภายในไมโครโปรเซสเซอร์

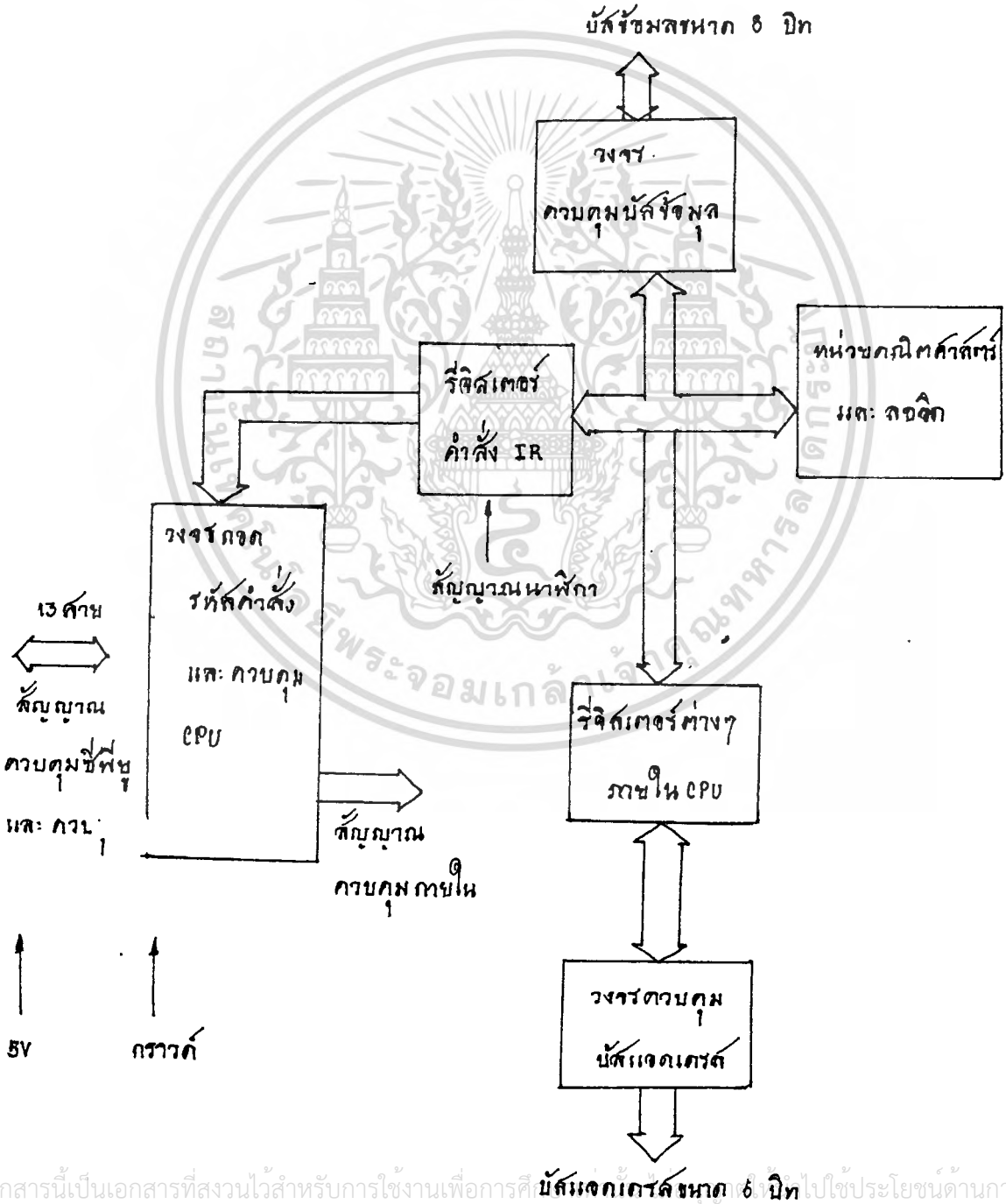


Two Buffers Are Required

รูปที่ 1.5 รูปแสดงส่วน รีจิสเตอร์ชั่วคราวที่เพิ่มขึ้นมา

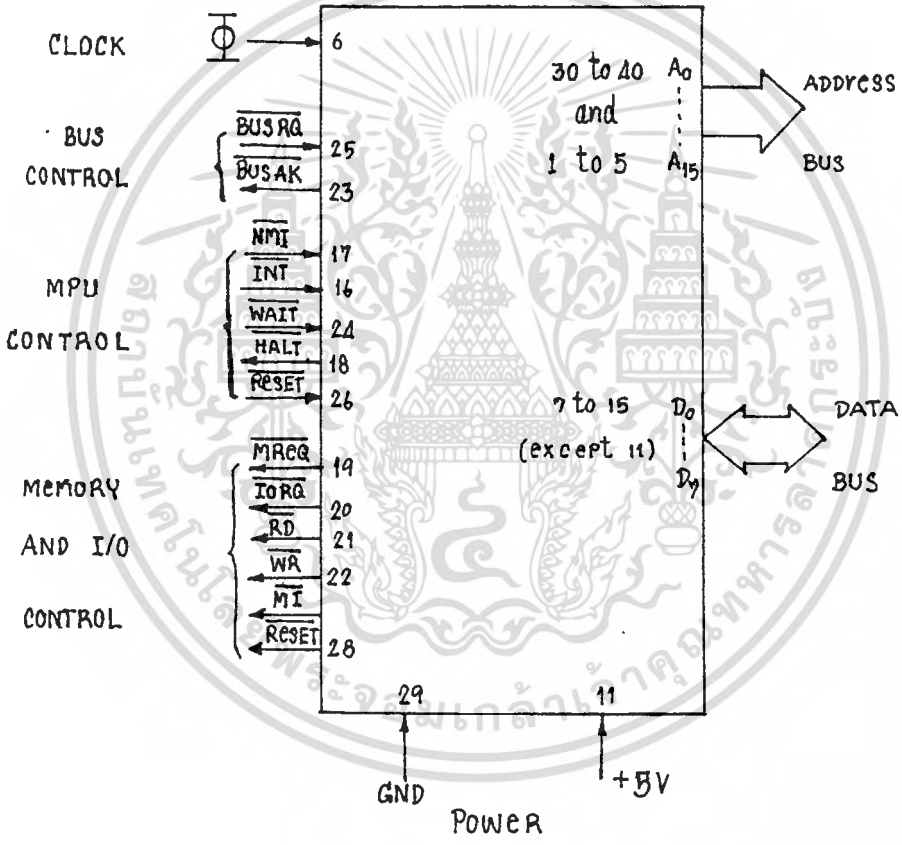
2. ลักษณะโครงสร้างของไมโครโปรเซสเซอร์ เบอร์ Z-80

Z-80 เป็นไมโครโปรเซสเซอร์ที่พัฒนามาจาก ไมโครโปรเซสเซอร์เดิมคือเบอร์ 8080 ฉะนั้นลักษณะของโครงสร้าง คำสั่ง การใช้งานจึงคล้ายกันมากแต่ว่า Z-80 ได้มีความสามารถมากขึ้นหลายอย่าง ทั้งในด้านของคำสั่งและการอินเตอร์รัพต์ เป็นต้น



2.2 การจัดขาและความหมาย

ตัว ชิฟ ซีพียู เป็นลักษณะตัวแบบสีดำที่มีขนาด 40 ขา ซึ่งแยกออกเป็นกลุ่มต่าง ๆ คือ บัสข้อมูล 8 บิท ซึ่งเป็นบัสชนิด สองทิศทางคือสามารถวิ่งเข้าออกจากซีพียูได้ และบัสแอดเดรสที่มีขนาด 16 บิท ทำให้สามารถอ้างแอดเดรสได้ถึง 64 Kbytes และบัสแอดเดรสนี้ยังเป็นส่วนใช้ในการอ้างอิงถึงแอดเดรสของ พอร์ทอินพุทเอาท์พุทด้วย และอีกกลุ่มหนึ่งคือกลุ่มของสัญญาณควบคุมการทำงาน ซึ่งจะมีทั้งหมด 13 สาย ที่เหลือเป็นขาสัญญาณนาฬิกาและแรงจ่ายไฟเลี้ยง



รูปที่ 2.3 แสดงการจัดขาต่าง ๆ ของ Z-80

รายละเอียดและหน้าที่สำคัญของสัญญาณต่าง ๆ ดังนี้

A0 - A15 เป็นกลุ่มของสายบัสขนาด 16 เส้น มีลักษณะเป็น โลจิกสามสถานะและจะมีสัญญาณที่ใช้เพื่อแยกว่าเวลาใดเป็นการอ้างแอดเดรสของบัส หรือแอดเดรสของพอร์ทอินพุทและเอาท์พุท ทั้งนี้เพราะการใช้สายสัญญาณร่วมกัน

D0-D7 เป็นกลุ่มสายสัญญาณ ขนาด 8 เส้น ลักษณะของสัญญาณเป็นแบบ สามสถานะ เช่นกัน ข้อมูลสามารถที่จะวิ่งได้สองทิศทาง ตามแต่ลักษณะของการอ่าน หรือ เขียนระหว่างตัว ซีพียูและอุปกรณ์ภายนอก

MI ลักษณะเป็นสัญญาณเอาท์พุท โดยส่งสัญญาณออกมาให้ทราบว่า ตอนนี้ทำงานอยู่ ในสภาวะเฟทช์ (fetch) (สภาวะการเอาข้อมูลคำสั่งจากหน่วย ความจำเข้าสู่ตัวซีพียู) โดยแอคตีฟที่โลจิก " 0 "

MREQ เป็นสายสัญญาณเอาท์พุทแบบสามสถานะ ซึ่งให้บ่งบอกว่าขณะนี้สัญญาณที่บัสแอค เดรสมีค่าของแอคเดรสที่ต้องการติดต่อกับหน่วยความจำ โดยการแอคตีฟที่โลจิก " 0 "

IORQ เป็นสายสัญญาณที่เอาท์พุทซึ่งบอกว่า สัญญาณในแอคเดรสบัสที่ขา A0-A7 มีค่า ของแอคเดรสพอร์ทอินพุท เอาท์พุทอยู่ ฉะนั้นจึงเป็นสัญญาณที่บ่งบอกถึงการที่ ซีพียูต้องการติดต่อกับหน่วยอินพุทเอาท์พุท

RD เป็นสายสัญญาณเอาท์พุทที่บอกว่า ขณะนี้ซีพียูต้องการอ่านข้อมูลจากหน่วยความจำ หรืออุปกรณ์อินพุทเอาท์พุท

WR เป็นสายสัญญาณเอาท์พุทที่บอกว่า ขณะนี้ซีพียูต้องการเขียนข้อมูลลงสู่หน่วยความจำ หรืออุปกรณ์อินพุทเอาท์พุท

RSFH เป็นสายที่ส่งสัญญาณเพื่อจะบอกว่า ขณะนี้บัสแอคเดรสมีค่าของแอคเดรสสำหรับ การรีเฟรชหน่วยความจำชนิดไดนามิก

HALT เป็นสายสัญญาณเอาท์พุทที่แอคตีฟเมื่อซีพียูกระทำคำสั่ง HALT โดยจะให้แอคตีฟ โลจิก " 0 "

WAIT เป็นสัญญาณที่จะบอกให้ทราบว่า ขณะนี้หน่วยความจำหรืออุปกรณ์อินพุทเอาท์พุทยัง ไม่พร้อมที่จะรับหรือส่งผ่านข้อมูล นั่นคือสัญญาณนี้เข้าไปสู่ตัวซีพียูก็จะหยุดรอจนกว่า สัญญาณ WAIT เลิกไป

INT เป็นขารับสัญญาณจากอุปกรณ์อินพุทที่ทำการอินเตอร์ซีพียู การอินเตอร์รัพท์มีอยู่ด้วย กันหลายโหมด สำหรับสัญญาณที่เข้าสู่ขานี้ จะเรียกว่ามาสเคเบิลอินเตอร์รัพท์ (Maskable Interrupt)

RESET เป็นขาสัญญาณที่รับสัญญาณจากอุปกรณ์ภายนอก ที่ต้องการรีเซตตัวซีพียู หรือต้อง การให้โปรแกรมเคาน์เตอร์มีค่าเป็น " 0 "

NMI เป็นขาสัญญาณอินเตอร์รัพท์อีกแบบหนึ่ง ที่เรียกว่า นอนมาสเคเบิลอินเตอร์รัพท์ เป็นสัญญาณอินเตอร์รัพท์ที่มีความสำคัญสูงสุด ที่ซีพียูต้องรับเสมอ

BUSRQ เป็นขาสัญญาณอินพุทที่บอกให้ซีพียูรู้ว่าขณะนี้ อุปกรณ์ภายนอกต้องการให้ระบบบัส

ซึ่งตัวซีพียูต้องตัดตัวเองออกจากระบบบัส โดยการทำให้ตัวซีพียูอยู่ในสถานะอินพีแดนซ์สูงและส่งสัญญาณบอกอุปกรณ์ที่ขอให้บัสทราบว่า ขณะนี้ซีพียูได้ตัดตัวเองออกแล้วพร้อมที่จะให้ใช้งานบัสได้แล้ว

BUSAK เป็นสัญญาณที่ส่งออกไปจากตัวซีพียู เพื่อบอกว่านี้ซีพียูไม่ได้ใช้ระบบบัสแล้ว

ϕ เป็นขารับสัญญาณนาฬิกาเพื่อเป็นฐานเวลาหลักในการทำงานของระบบ

+5 เป็นแรงดันไฟเลี้ยงตัวซีพียู

GND เป็นกราวด์ของตัวซีพียู

2.3 ลักษณะของสัญญาณไดอะแกรมเวลาในการทำงานในไซเกิ้ลต่าง ๆ ในการทำของไมโครโปรเซสเซอร์ทั้งหลายนั้น ดังที่ได้กล่าวมาแล้วข้างต้นว่ามีการทำงานอยู่สองจังหวะ คือ เฟทช์และเอ็กซีคิ้ว ซึ่งในสถานะของการเฟทช์จะมีเพียง แมทชีนไซเกิ้ลเดียวที่เรียกว่า M1 ส่วนสภาวะของการเอ็กซีคิ้วนั้น จะมีแมทชีนไซเกิ้ลอยู่หลายแมทชีนไซเกิ้ล คือ M2,M3,M4... ซึ่งแต่ละคำสั่งก็จะใช้จำนวนของแมทชีนไซเกิ้ลไม่เท่ากัน และที่จะได้กล่าวต่อไปนี้ จะกล่าวถึงแมทชีนไซเกิ้ลของแต่ละอันนั้น (M1,M2...) มีอะไรบ้างและไดอะแกรมเวลานั้นเป็นอย่างไร (ซึ่งให้นักศึกษาเข้าใจได้เลยว่าไมโครโปรเซสเซอร์จะทำงานได้เพียงตาม M ไซเกิ้ลที่มีอยู่เท่านั้น เพียงแต่คำสั่งใดจะใช้แมทชีนใดมาประกอบกันเพื่อทำให้ทำงานได้สมบูรณ์เท่านั้นเอง) ไดอะแกรมเวลาของไมโครโปรเซสเซอร์ จะอ้างถึงสัญญาณนาฬิกาของระบบ (CLOCK) ซึ่งปกติมักจะเป็น คริวตอล ที่ป้อนเข้าสู่ขา 6 นำมาเป็นฐานเวลาในการอ้างอิงเสมอและแต่ละลูกของ CLOCK จะเรียกเป็น T state, หรือ T cycle ประมาณ 3-6 ลูก ดังนั้น หากระบบใช้สัญญาณนาฬิกาที่มีขนาด 4 MHz ก็จะใช้เวลาทำงานแต่ละแมทชีนไซเกิ้ลประมาณ 750 nS-1500 nS ดังนั้นหากเราทราบว่าในแต่ละคำสั่งประกอบด้วยจำนวน T cycle มากน้อยเท่าใดเราก็สามารถคำนวนหาค่าเวลาที่ต้องใช้ได้

ในแมทชีนไซเกิ้ลของ Z-80 นั้นจะมีอยู่ด้วยกัน 8 machine cycle ดังนี้ :

1. การทำงาน เฟทช์ (fetch) เอารหัสคำสั่ง เป็นช่วง M1 cycles
2. ช่วงไซเกิ้ลการอ่านหรือเขียนหน่วยความจำ
3. ช่วงไซเกิ้ลของการรับส่งข้อมูลไอโอ
4. ช่วงไซเกิ้ลการร้องขอและตอบสนองการใช้บัส (request/acdnowledge bus)
5. ช่วงไซเกิลการขอและตอบสนองการ interrupt
6. ช่วงไซเกิลการของและตอบสนองแบบ non maskable interrupt
7. ช่วงไซเกิลของคำสั่ง HALT
8. ช่วงไซเกิลของการ รีเซท (reset)

สรุปรหัสคำสั่งที่อยู่ในรูปนีโมนิค รหัสภาษาเครื่องและช่วงเวลาในการประมวลผล

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
LD (ADDR),HL	22 pppq	3	16	SHLD ADDR	16
LD (ADDR),IX	DD 22 pppq	4	20		
LD (ADDR),IY	FD 22 pppq	4	20		
LD (ADDR),SP	ED 73 pppq	4	20		
LD (BC),A	02	1	7	STAX B	7
LD (DE),A	12	1	7	STAX D	7
LD HL,(ADDR)	2A pppq	3	16	LHLD ADDR	16
LD (HL),DATA	36 YY	2	10	MVI M,DATA	10
LD (HL),REG	01110sss	1	7	MDV M,REG	7
LD I,A	ED 47	2	9		
LD IX,(ADDR)	DD 2A pppq	4	20		
LD IX,DATA16	DD 21 YYYY	4	14		
LD (IX + DISP),DATA	DD 36 YY YY	4	19		
LD (IX + DISP),REG	DD 01110sss	3	19		
	YY				
LD IY,(ADDR)	FD 2A pppq	4	20		
LD IY,DATA16	FD 21 YYYY	4	14		
LD (IY + DISP),DATA	FD 36 YYYY	4	19		
LD (IY + DISP),REG	FD 01110sss	3	19		
	YY				
LD R,A	ED 4F	2	9		
LD REG,DATA	00ddd110	2	7	MVI REG,DATA	7
	YY				
LD REG,(HL)	01ddd110	1	7	MOV REG,M	7
LD REG,(IX + DISP)	DD	3	19		
	01ddd110				
	YY				
LD REG,(IY + DISP)	FD	3	19		
	01ddd110				
	YY				
LD REG,REG	01dddsss	1	4	MDV REG,REG	5
LD RP,(ADDR)	ED 01xx1011	4	20		
	pppq				
LD RP,DATA16	00xx0001	3	10	LXI RP,DATA16	10
	YYYY				
LD SP,HL	F9	1	6	SPHL	5
LD SP,IX	DD F9	2	10		
LD SP,IY	FD F9	2	10		
LDD	ED A8	2	16		
LDDR	ED B8	2	21/16		
LDI	ED A0	2	16		
LDIR	ED B0	2	21/16		
NEG	ED 44	2	8		
NOP	00	1	4	NOP	4
OR DATA	F6 YY	2	7	ORI DATA	7
OR (HL)	B6	1	7	ORA M	7
OR (IX + DISP)	DD B6 YY	3	19		
OR (IY + DISP)	FD B6 YY	3	19		
OR REG	10110xxx	1	4	ORA REG	5
OUT (C),REG	ED	2	12		
	01sss001				

สรุปรหัสคำสั่งที่อยู่ในรูปนีโมนิก รหัสภาษาเครื่องและช่วงเวลาในการประมวลผล

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
DEC REG	00xxx101	1	4	DCR REG	5
DI	F3	1	4	DI	4
DJNZ DISP	10 YY	2	8/13		
EI	FB	1	4	EI	4
EX AF,AF	08	1	4		
EX DE,HL	EB	1	4	XCHG	4
EX (SP),HL	E3	1	19	XTHL	18
EX (SP),IX	DD E3	2	23		
EX (SP),IY	FD E3	2	23		
EXX	D9	1	4		
HALT	76	1	4	HLT	4
IM 0	ED 46	2	8		
IM 1	ED 56	2	8		
IM 2	ED 5E	2	8		
IN A,PORT	DB YY	2	10	IN PORT	10
IN REG,(C)	ED	2	11		
	01ddd000				
INC (HL)	34	1	11	INR M	10
INC IX	DD 23	2	10		
INC (IX + DISP)	DD 34 YY	3	23		
INC IY	FD 23	2	10		
INC (IY + DISP)	FD 34 YY	3	23		
INC RP	00xx0011	1	6	INX RP	5
INC REG	00xxx100	1	4	INR REG	5
IND	ED AA	2	15		
INDR	ED BA	2	20/15		
INI	ED A2	2	15		
INIR	ED B2	2	20/15		
JP LABEL	C3 pppq	3	10	JMP LABEL	10
JP C,LABEL	DA pppq	3	10	JC LABEL	10
JP (HL)	E9	1	4	PCHL	5
JP (IX)	DD E9	2	8		
JP (IY)	FD E9	2	8		
JP M,LABEL	FA pppq	3	10	JM LABEL	10
JP NC,LABEL	D2 pppq	3	10	JNC LABEL	10
JP NZ,LABEL	C2 pppq	3	10	JNZ LABEL	10
JP P,LABEL	F2 pppq	3	10	JP LABEL	10
JP PE,LABEL	EA pppq	3	10	JPE LABEL	10
JP PO,LABEL	E2 pppq	3	10	JPO LABEL	10
JP Z,LABEL	CA pppq	3	10	JZ LABEL	10
JR C,DISP	38 YY	2	7/12		
JR DISP	18 YY	2	12		
JR NC,DISP	30 YY	2	7/12		
JR NZ,DISP	20 YY	2	7/12		
JR Z,DISP	28 YY	2	7/12		
LD A,(ADDR)	3A pppq	3	13	LDA ADDR	13
LD A,(BC)	0A	1	7	LDAX B	7
LD A,(DE)	1A	1	7	LDAX D	7
LD A,I	ED 57	2	9		
LD A,R	ED 5F	2	9		
LD (ADDR),A	32 pppq	3	13	STA ADDR	13
LD (ADDR),BC	ED 43 pppq	4	20		
LD (ADDR),DE	ED 53 pppq	4	20		

สรุปรหัสคำสั่งที่อยู่ในรูปนีโมนิค รหัสภาษาเครื่องและช่วงเวลาในการประมวลผล

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
ADC DATA	CE YY	2	7	ACI DATA	7
ADC (HL)	8E	1	7	ADC M	7
ADC HL,RP	ED 01xx1010	2	15		
ADC (IX + DISP)	DD 8E YY	3	19		
ADC (IY + DISP)	FD 8E YY	3	19		
ADC REG	10001xxx	1	4	ADC REG	4
ADD DATA	C6 YY	2	7	ADI DATA	7
ADD (HL)	86	1	7	ADD M	7
ADD HL,RP	00xx1001	1	11	DAD RP	10
ADD (IX + DISP)	DD 86 YY	3	19		
ADD IX,PP	DD 00xx1001	2	15		
ADD (IY + DISP)	FD 86 YY	3	19		
ADD IY,RR	FD 00xx1001	2	15		
ADD REG	10000xxx	1	4	ADD REG	4
AND DATA	E6 YY	2	7	ANI DATA	7
AND (HL)	A6	1	7	ANA M	7
AND (IX + DISP)	DD A6 YY	3	19		
AND (IY + DISP)	FD A6 YY	3	19		
AND REG	10100xxx	1	4	ANA REG	4
BIT B,(HL)	CB 01bbb110	2	12		
BIT B,(IX + DISP)	DD CB YY 01bbb110	4	20		
BIT B,(IY + DISP)	FD CB YY 01bbb110	4	20		
BIT B,REG	CB 01bbbxxx	2	8		
CALL LABEL	CD ppqq	3	17	CALL LABEL	17
CALL C,LABEL	DC ppqq	3	10/17	CC LABEL	11/17
CALL M,LABEL	FC ppqq	3	10/17	CM LABEL	11/17
CALL NC,LABEL	D4 ppqq	3	10/17	CNC LABEL	11/17
CALL NZ,LABEL	C4 ppqq	3	10/17	CNZ LABEL	11/17
CALL P,LABEL	F4 ppqq	3	10/17	CP LABEL	11/17
CALL PE,LABEL	EC ppqq	3	10/17	CPE LABEL	11/17
CALL PO,LABEL	E4 ppqq	3	10/17	CPO LABEL	11/17
CALL Z,LABEL	CC ppqq	3	10/17	CZ LABEL	11/17
CCF	3F	1	4	CMC	4
CP DATA	FE YY	2	7	CPI DATA	7
CP (HL)	BE	1	7	CMP M	7
CP (IX + DISP)	DD BE YY	3	19		
CP (IY + DISP)	FD BE YY	3	19	CMP REG	19
CP REG	10111xxx	1	4		
CPD	ED A9	2	16		
CPDR	ED B9	2	21/16		
CPI	ED A1	2	16		
CPIR	ED B1	2	21/16		
CPL	2F	1	4	CMA	4
DAA	27	1	4	DAA	4
DEC (HL)	35	1	11	DCR M	10
DEC IX	DD 2B	2	10		
DEC (IX + DISP)	DD 35 YY	3	23		
DEC IY	FD 2B	2	10		
DEC (IY + DISP)	FD 35 YY	3	23		
DEC RP	00xx1011	1	6	DCX RP	5

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

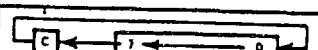
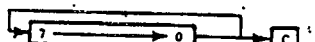
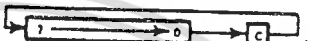
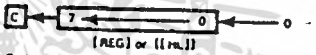
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
INTERRUPT	DI		1		Disable interrupts
	EI		1		Enable interrupts
	RST	N	1		PUSH PC, $(PC) \leftarrow (8 \cdot N)_m$
					Restart at designated location
	RETI		2		Return from interrupt
	RETN		2		Return from nonmaskable interrupt
	IM	0	2		Set interrupt mode 0, 1, or 2
		1			
		2			
STATUS	SCF		1	1 0 0	C←1
					Set Carry flag
	CCF		1	X ? 0	$C \leftarrow \overline{C}$
					Complement Carry flag
	NOP		1		No operation — volatile memories are refreshed
	HALT		1		CPU halts, executes NOPs to refresh volatile memories.

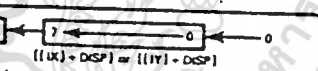
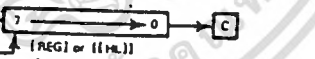
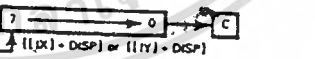
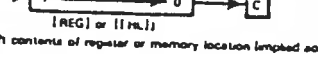
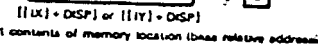
สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	RRD		2	X X P 0 0	(A) 7 4 3 0 — 7 4 3 0 — ([HL]) Rotate one BCD digit right between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.
BIT MANIPULATION	BIT	B,REG	2	X ? ? 1 0	Z—REG(B) Zero flag contains complement of the selected register bit
	BIT	B,(HL)	2	X ? ? 1 0	Z—[[HL]](B) Zero flag contains complement of selected bit of the memory location (implied addressing).
	BIT	B,(IX + DISP) B,(IY + DISP)	4	X ? ? 1 0	Z—[[IX] + DISP](B) or Z—[[IY] + DISP](B) Zero flag contains complement of the selected bit of the mayhory location (base relative addressing).
	SET	B,REG	2		REG(B)—1 Set indicated register bit
	SET	B,(HL)	2		[[HL]](B)—1 Set indicated bit of memory location (implied addressing).
	SET	B,(IX + DISP) B,(IY + DISP)	4		[[IX] + DISP](B)—1 or [[IY] + DISP](B)—1 Set indicated bit of memory location (base relative addressing).
	RES	B,REG	2		REG(B)— 0. Reset indicated register bit
	RES	B,(HL)	2		[[HL]](B)—0 Reset indicated bit in memory location (implied addressing).
	RES	B,(IX + DISP) B,(IY + DISP)	4		[[IX] + DISP](B)—0 or [[IY] + DISP](B)—0 Reset indicated bit of memory location (base relative addressing).

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
STACK	PUSH	PR	1		[[SP]-1]—[PRHI] [[SP]-2]—[PRLO] [SP]—[SP]-2 Put contents of register pair on top of Stack and decrement Stack Pointer.
	PUSH	IX IY	2		[[SP]-1]—[IXHI] or [SP]-1]—[YHI] [[SP]-2]—[IXLO]] or [[SP]-2]—[IYLO]] [SP]—[SP]-2 Put contents of Index register on top of Stack and decrement Stack Pointer.
	POP	PR	1		[PRLO]]—[[SP]] [PRHI]]—[[SP] + 1] [SP]—[SP] + 2 Put contents of top of Stack in register pair and increment Stack Pointer.
	POP	IX IY	2		[IXLO]]—[[SP]] or [IYLO]]—[[SP]] [IXHI]]—[[SP] + 1] or [IYHI]]—[[SP] + 1] [SP]—[SP] + 2 Put contents of top of Stack in index register and increment Stack Pointer.
	EX	(SP),HL	1		[H]——[[SP] + 1] [L]——[[SP]] Exchange contents of HL and top of Stack.
	EX	(SP),IX (SP),IY	2		[IXHI]——[[SP] + 1] or [IYHI]——[[SP] + 1] [IXLO]]——[[SP]] or [IYLO]]——[[SP]] Exchange contents of index register and top of Stack.

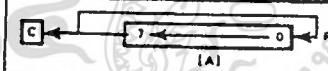
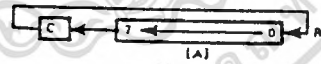
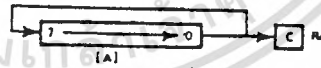
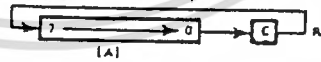
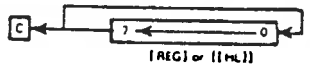
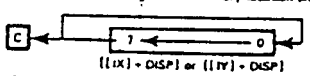
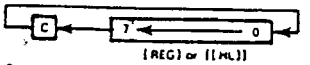
สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	RL	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Rotate contents of memory location (base relative addressing) left through Carry
	RRC	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Rotate contents of register or memory location (implied addressing) right with branch Carry
	RRC	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Rotate contents of memory location (base relative addressing) right with branch Carry
	RR	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Rotate contents of register or memory location (implied addressing) right through Carry
	RR	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Rotate contents of memory location (base relative addressing) right through Carry.
	SLA	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Shift contents of register or memory location (implied addressing) left with branch Carry.

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	SLA	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Shift contents of memory location (base relative addressing) left with branch Carry.
	SRA	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Arithmetic shift right contents of register or memory location (implied addressing).
	SRA	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Arithmetic shift right contents of memory location (base relative addressing).
	SRL	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Shift contents of register or memory location (implied addressing) right with branch Carry
	SRL	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Shift contents of memory location (base relative addressing) right with branch Carry.
	RLD		2	X X P 0 0	 Rotate one BCD digit left between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER-REGISTER OPERATE (Continued)	ADC	HL,RP	2	X X X O ? 0	(HL) ← (HL) + (RP) + C 16-bit add with Carry; register pair contents to contents of HL
	SBC	HL RP	2	X X X O ? 1	(HL) ← (HL) - (RP) - C 16-bit subtract with Carry; register pair contents from contents of HL
	ADD	IX PP	2	X ? 0	(IX) ← (IX) + (PP) 16-bit add register pair contents to contents of Index register (PP: BC, DE, IX, SP)
	ADD	IY RR	2	X ? 0	(IY) ← (IY) + (RR) 16-bit add register pair contents to contents of IY Index register (RR: BC, DE, IY, SP)
REGISTER OPERATE	DAA		1	X X X P X	Decimal adjust Accumulator, assuming that Accumulator contents are the sum or difference of BCD operands
	CPL		1	1 1	(A) ← (A) Complement Accumulator (ones complement)
	NEG		2	X X X O X 1	(A) ← (A) + 1 Negate Accumulator (twos complement)
	INC	REG	1	X X O X 0	(REG) ← (REG) + 1 Increment register contents
	INC	RP	1		(RP) ← (RP) + 1 Increment contents of register pair
	INC	IX IY	2		(IX) ← (IX) + 1 or (IY) ← (IY) + 1 Increment contents of Index register
	DEC	REG	1	X X O X 1	(REG) ← (REG) - 1 Decrement register contents
	DEC	RP	1		(RP) ← (RP) - 1 Decrement contents of register pair
	DEC	IX IY	2		(IX) ← (IX) - 1 or (IY) ← (IY) - 1 Decrement contents of Index register

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	RLCA		1	X 0 0	 Rotate Accumulator left with branch Carry
	RLA		1	X 0 0	 Rotate Accumulator left through Carry
	RRCA		1	X 0 0	 Rotate Accumulator right with branch Carry
	RRA		1	X 0 0	 Rotate Accumulator right through Carry
	RLC	REG (HL)	2	X X X P 0 0	 Rotate contents of register or memory location (implied addressing) left with branch Carry
	RLC	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 Rotate contents of memory location (base relative addressing) left with branch Carry
	RL	REG (HL)	2	X X X P 0 0	 Rotate contents of register or memory location (implied addressing) left through Carry

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
JUMP ON CONDITION	JP	COND,LABEL	3		If COND. then [PC]—LABEL Jump to instruction at address LABEL if the condition is satisfied
	JR	C,DISP	2		If C=1, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Carry flag is set
	JR	NC,DISP	2		If C=0, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Carry flag is reset
	JR	Z,DISP	2		If Z=1, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Zero flag is set
	JR	NZ,DISP	2		If Z=0, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Zero flag is reset
	DJNZ	DISP	2		[B]—[B]-1 If [B]≠0, then [PC]—[PC]+2+DISP Decrement contents of B and Jump relative to contents of Program Counter if result is not 0
REGISTER-REGISTER MOVE	LD	DST,SRC	1		[DST]—[SRC] Move contents of source register to destination register. SRC and DST may each be A, B, C, D, E, H or L.
	LD	A,IV	2	X X I 0 0	[A]—[IV] Move contents of interrupt vector to Accumulator
	LD	A,R	2	X X I 0 0	[A]—[R] Move contents of Refresh register to Accumulator
	LD	IV,A	2		[IV]—[A] Load interrupt vector from Accumulator
	LD	R,A	2		[R]—[A] Load Refresh register from Accumulator
	LD	SP,HL	1		[SP]—[HL] Move contents of HL to Stack Pointer
	LD	SP,IX SP,IY	2		[SP]—[IX] or [SP]—[IY] Move contents of Index register to Stack Pointer

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER-REGISTER MOVE (Continued)	EX	DE,HL	1		[DE]⟷[HL] Exchange DE and HL contents
	EX	AF,AF'	1		[AF]⟷[AF'] Exchange program status and alternate program status
	EXX		1		$\begin{pmatrix}[BC]\\[DE]\\[HL]\end{pmatrix}$ ⟷ $\begin{pmatrix}[BC']\\[DE']\\[HL']\end{pmatrix}$ Exchange register pairs and alternate register pairs.
REGISTER-REGISTER OPERATE	ADD	REG	1	X X X 0 X 0	[A]—[A]+[REG] Add contents of register to Accumulator
	ADC	REG	1	X X X 0 X 0	[A]—[A]+[REG]+C Add contents of register and Carry to Accumulator.
	SUB	REG	1	X X X 0 X 1	[A]—[A]-[REG] Subtract contents of register from Accumulator
	SBC	REG	1	X X X 0 X 1	[A]—[A]-[REG]-C Subtract contents of register and Carry from Accumulator
	AND	REG	1	0 X X P X 1	[A]—[A]∧[REG] AND contents of register with contents of Accumulator
	OR	REG	1	0 X X P X 0	[A]—[A]∨[REG] OR contents of register with contents of Accumulator
	XOR	REG	1	0 X X P X 0	[A]—[A]∨[REG] Exclusive-OR contents of register with contents of Accumulator
	CP	REG	1	X X X 0 X 1	[A]:[REG] Compare contents of register with contents of Accumulator
	ADD	HL,RP	1	X 7 0	[HL]—[HL]+[RP] 16-bit add register pair contents to contents of HL

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE (Continued)	INC	(IX + DISP) (IY + DISP)	3	X X O X 0	[(IX) + DISP]←[(IX) + DISP] + 1 or [(IY) + DISP]←[(IY) + DISP] + 1 Increment using base relative addressing
	DEC	(HL)	1	X X O X 1	[(HL)]←[(HL)] - 1 Decrement using implied addressing
	DEC	(IX + DISP) (IY + DISP)	3	X X O X 1	[(IX) + DISP]←[(IX) + DISP] - 1 or [(IY) + DISP]←[(IY) + DISP] - 1 Decrement using base relative addressing
IMMEDIATE	LD	REG,DATA	2		[REG]←DATA Load immediate into register
	LD	RP,DATA16	3		[RP]←DATA16 Load 16 bits of immediate data into register pair
	LD	IX,DATA16 IY,DATA16	4		[IX]←DATA16 or [IY]←DATA16 Load 16 bits of immediate data into index register
	LD	(HL),DATA	2		[(HL)]←DATA Load immediate into memory location using implied addressing
	LD	(IX + DISP),DATA (IY + DISP),DATA	4		[(IX) + DISP]←DATA or [(IY) + DISP]←DATA Load immediate into memory location using base relative addressing
JUMP	JP	LABEL	3		[PC]←LABEL Jump to instruction at address LABEL
	JR	DISP	2		[PC]←[PC] + 2 + DISP Jump relative to present contents of Program Counter
	JP	(HL)	1		[PC]←[HL] Jump to address contained in HL
	JP	(IX) (IY)	2		[PC]←[IX] or [PC]←[IY] Jump to address contained in index register

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
SUBROUTINE CALL AND RETURN	CALL	LABEL	3		[(SP)-1]←[PC(HI)] [(SP)-2]←[PC(LO)] [SP]←[SP]-2 [PC]←LABEL Jump to subroutine starting at LABEL
	CALL	COND,LABEL	3		Jump to subroutine if condition is satisfied; otherwise, continue in sequence
	RET		1		[PC(LO)]←[(SP)] [PC(HI)]←[(SP) + 1] [SP]←[SP] + 2 Return from subroutine
	RET	COND	1		Return from subroutine if condition is satisfied; otherwise, continue in sequence
IMMEDIATE OPERATE	ADD	DATA	2	X X X O X 0	[A]←[A] + DATA Add immediate to Accumulator
	ADC	DATA	2	X X X O X 0	[A]←[A] + DATA + C Add immediate with Carry
	SUB	DATA	2	X X X O X 1	[A]←[A]-DATA Subtract immediate from Accumulator
	SBC	DATA	2	X X X O X 1	[A]←[A]-DATA-C Subtract immediate with Carry
	AND	DATA	2	0 X X P X 1	[A]←[A]∧DATA AND immediate with Accumulator
	OR	DATA	2	0 X X P X 0	[A]←[A]∨DATA OR immediate with Accumulator
	XOR	DATA	2	0 X X P X 0	[A]←[A]∨DATA Exclusive-OR immediate with Accumulator
	CP	DATA	2	X X X O X 1	[A] : DATA Compare immediate data with Accumulator contents

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
BLOCK TRANSFER AND SEARCH (Continued)	CPI		2	X X X X 1	(A):[(HL)] (HL)←(HL)+1 (BC)←(BC)-1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Increment address and decrement byte count.
	CPD		2	X X X X 1	(A):[(HL)] (HL)←(HL)-1 (BC)←(BC)-1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Decrement address and byte count.
SECONDARY MEMORY REFERENCE	ADD	(HL)	1	X X X 0 X 0	(A)←(A)+[(HL)] Add to Accumulator using implied addressing.
	ADD	(IX+DISP) (IY+DISP)	3	X X X 0 X 0	(A)←(A)+[(IX)+DISP] or (A)←(A)+[(IY)+DISP] Add to Accumulator using base relative addressing
	ADC	(HL)	1	X X X 0 X 0	(A)←(A)+[(HL)]+C Add with Carry using implied addressing.
	ADC	(IX+DISP) (IY+DISP)	3	X X X 0 X 0	(A)←(A)+[(IX)+DISP]+C or (A)←(A)+[(IY)+DISP]+C Add with Carry using base relative addressing
	SUB	(HL)	1	X X X 0 X 1	(A)←(A)-[(HL)] Subtract from Accumulator using implied addressing
	SUB	(IX+DISP) (IY+DISP)	3	X X X 0 X 1	(A)←(A)-[(IX)+DISP] or (A)←(A)-[(IY)+DISP] Subtract using base relative addressing
	SBC	(HL)	1	X X X 0 X 1	(A)←(A)-[(HL)]-C Subtract with Carry using implied addressing

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE (Continued)	SBC	(IX+DISP) (IY+DISP)	3	X X X 0 X 1	(A)←(A)-[(IX)+DISP]-C or (A)←(A)-[(IY)+DISP]-C Subtract with Carry using base relative addressing
	AND	(HL)	1	0 X X P X 1	(A)←(A)∧[(HL)] AND with Accumulator using implied addressing
	AND	(IX+DISP) (IY+DISP)	3	0 X X P X 1	(A)←(A)∧[(IX)+DISP] or (A)←(A)∧[(IY)+DISP] AND with Accumulator using base relative addressing
	OR	(HL)	1	0 X X P X 0	(A)←(A)∨[(HL)] OR with Accumulator using implied addressing
	OR	(IX+DISP) (IY+DISP)	3	0 X X P X 0	(A)←(A)∨[(IX)+DISP] or (A)←(A)∨[(IY)+DISP] OR with Accumulator using base relative addressing
	XOR	(HL)	1	0 X X P X 0	(A)←(A)⊻[(HL)] Exclusive-OR with Accumulator using implied addressing
	XOR	(IX+DISP) (IY+DISP)	3	0 X X P X 0	(A)←(A)⊻[(IX)+DISP] or (A)←(A)⊻[(IY)+DISP] Exclusive-OR with Accumulator using base relative addressing
	CP	(HL)	1	0 X X O X 1	(A):[(HL)] Compare with Accumulator using implied addressing
	CP	(IX+DISP) (IY+DISP)	3	0 X X O X 1	(A):[(IX)+DISP] or (A):[(IY)+DISP] Compare with Accumulator using base relative addressing
	INC	(HL)	1	X X O X 0	[(HL)]←[(HL)]+1 Increment using implied addressing

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
PRIMARY MEMORY REFERENCE (Continued)	LD	REG,(HL)	1		(REG)←((HL)) Load register from memory location addressed by contents of HL.
	LD	(BC),A (DE),A	1		((BC))←(A) or ((DE))←(A) Store Accumulator to memory location addressed by the contents of the specified register pair.
	LD	(HL),REG	1		((HL))←(REG) Store register contents to memory location addressed by the contents of HL.
	LD	REG,(IX + DISP) REG,(IY + DISP)	3		(REG)←((IX) + DISP) or (REG)←((IY) + DISP) Load register from memory location using base relative addressing.
	LD	(IX + DISP),REG (IY + DISP),REG	3		((IX) + DISP)←(REG) or ((IY) + DISP)←(REG) Store register to memory location addressed relative to contents of index register.
BLOCK TRANSFER AND SEARCH	LDIR		2	0 0 0	Repeat until (BC)=0: ((DE))←((HL)) (DE)←(DE) + 1 (HL)←(HL) + 1 (BC)←(BC)-1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from low addresses to high. Contents of BC serve as a count of bytes to be transferred.
	LDDR		2	0 0 0	Repeat until (BC)=0: ((DE))←((HL)) (DE)←(DE)-1 (HL)←(HL)-1 (BC)←(BC)-1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from high addresses to low. Contents of BC serve as a count of bytes to be transferred.

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
BLOCK TRANSFER AND SEARCH (Continued)	LDI		2	X 0 0	((DE))←((HL)) (DE)←(DE) + 1 (HL)←(HL) + 1 (BC)←(BC)-1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Increment source and destination addresses and decrement byte count.
	LDD		2	X 0 0	((DE))←((HL)) (DE)←(DE)-1 (HL)←(HL)-1 (BC)←(BC)-1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Decrement source and destination addresses and byte count.
	CPIR		2	X X X X 1	Repeat until (A)=((HL)) or (BC)=0: (A)-((HL)) (HL)←(HL) + 1 (BC)←(BC)-1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from low addresses to high. Stop when a match is found or when the byte count becomes zero.
	CPDR		2	X X X X 1	Repeat until (A)=((HL)) or (BC)=0: (A)-((HL)) (HL)←(HL)-1 (BC)←(BC)-1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from high addresses to low. Stop when a match is found or when the byte count becomes zero.

บทที่ 3

การเขียนโปรแกรมสำหรับไมโครคอมพิวเตอร์

การทำงานของเครื่องคอมพิวเตอร์เหล่านี้ไม่ว่าจะเป็นเครื่องคอมพิวเตอร์จะต้องทำงานภายใต้เงื่อนไขของโปรแกรมที่สั่งงาน และถ้าพิจารณาให้ลึกซึ้งลงไปจะเห็นว่าระดับที่เครื่องจะรับรู้และจะกระทำตามคำสั่งได้เป็นคำสั่งในลักษณะภาษาเครื่อง (Machine Instruction) ที่เขียนเป็นเลขไบนารี่ ซีพียู จะรับคำสั่งเหล่านี้แล้วตีความในหน่วยควบคุม เพื่อกระทำงานตามคำสั่งนี้เครื่องคอมพิวเตอร์ทั่ว ๆ ไปก็มีคำสั่งภาษาเครื่องที่แตกต่างกันไป ขึ้นอยู่กับผู้ออกแบบโครงสร้าง

เรามาลองรู้จักว่าในการที่เราป้อนโปรแกรมด้วยระดับสูง เช่น FORTRAN หรือ BASIC อาศัยตัวแปรโปรแกรมเหล่านี้จนเป็นภาษาเครื่องแล้วจึงทำงานได้ ในทำนองเดียวกันการจัดการทำงานของเครื่องต่าง ๆ เช่นจัดให้ภาคแสดงผลลัพธ์ตามการคำนวณได้ หรือรับคำสั่งบางอย่างจากคีย์บอร์ด การรับรู้วิธีการควบคุมบางอย่าง เครื่องก็จะต้องรับรู้ภายใต้โปรแกรมแปลโปรแกรมเหล่านี้จนเป็นภาษาของเครื่องแล้วจึงจะทำงานได้ แต่ถ้าเราให้โปรแกรมเครื่องรับรู้ให้ทำงานได้ในส่วนจำกัดเครื่องก็จะทำงานได้ในขอบเขตจำกัด เช่น โปรแกรมมอนิเตอร์สำหรับเครื่องไมโครคอมพิวเตอร์แผงเดียว

ลักษณะของโปรแกรม

จากที่เราได้ศึกษามาแล้วในบทก่อนเกี่ยวกับชุดคำสั่งของ 8080 โดยแต่ละคำสั่งจะมีรหัสที่แทนหรือเป็นรหัสเครื่องรับรู้ในการสั่งงาน การเขียนโปรแกรมในลักษณะนี้เรียกภาษาเครื่อง หรือ ออปเจคโปรแกรม

```
00111010
01100000
00000000
01000111
00111010
01100001
00000000
10000000
00110010
01100010
00000000
```

ออปเจคโปรแกรม

การเขียนโปรแกรมด้วยภาษาเครื่องเหมาะสำหรับเครื่องไมโครคอมพิวเตอร์ที่ผู้ออกแบบลดข้อยุ่งยากนัก แต่ถ้าเราต้องการเขียนโปรแกรมที่ยาว ๆ และยุ่งยากซับซ้อน การเขียนตัวเลขไบนารี่ก็มีปัญหาเช่น

1. เป็นโปรแกรมที่ยากต่อการเขียน การแก้ไขและตรวจสอบโปรแกรมเพราะเป็นเลข 0 และ 1 หมดทำให้เราดูยาก
2. การป้อนโปรแกรมจะเป็นไปด้วยความลำบากทำให้เสียเวลาในการป้อนโปรแกรม
3. การอ่านโปรแกรมเป็นไปด้วยความลำบากทำให้เสียโอกาสในการเข้าใจตัวโปรแกรมยากด้วย
4. ต้องใช้เวลาในการเขียนโปรแกรมนาน
5. ผู้เขียนโปรแกรมจะต้องระมัดระวังในการเขียนเป็นพิเศษ เพราะโอกาสผิดพลาดมีได้ง่ายมาก และเมื่อผิดแล้วจะหาทางแก้ไขได้ยาก

จากเหตุผลดังกล่าวจึงได้มีผู้ออกแบบตัวแปรรหัสโดยการป้อนเป็นรหัสที่เราเข้าใจง่าย แล้วให้เครื่องแปลเป็นภาษาเครื่องอีกต่อหนึ่ง แนวทางการรวมเลขไบนารี่เข้าเป็นกลุ่มให้จดจำได้ง่าย ลักษณะการเขียนโปรแกรมสามารถเขียนได้ดังนี้

```
3A
60
00
47
3A
00
80
32
62
00
```

โปรแกรมดังกล่าวเมื่อเขียนด้วยตัวเลขฐานสิบหกการป้อนโปรแกรมเป็นไปได้รวดเร็วยิ่งขึ้น และสามารถตรวจสอบโปรแกรมได้ดีกว่าโดยการเปลี่ยนจากฐานเลขสิบหกที่เรากดแป้นจะเกิดขึ้นในวงจรระบบฮาร์ดแวร์ หรือซอร์ฟแวร์ ที่ผู้ออกแบบโครงไว้

แต่อย่างไรก็ตามถ้าหากว่ามีใครก็ตามเอาตัวโปรแกรมที่เขียนด้วยเลขฐานสิบหกมาให้เราตีความว่าในแต่ละคำสั่งหมายถึงอะไร เพราะรหัสเหล่านี้ยากต่อการจดจำหรือตีความหมาย จึงมีผู้พยายามที่จะหาความสะดวกสบายให้กับผู้เขียนในโปรแกรมด้วยการกำหนดตัวอักษรที่เหมาะสมกับคำสั่งที่เรียกว่า นีโมนิค ลักษณะของนีโมนิคจะกำหนดกันเป็นมาตรฐานสำหรับไมโครโปรเซสเซอร์แต่ละเบอร์ แต่มักจะยึดถือหลักการให้เข้าใจว่าง่าย ตัวอย่างโปรแกรมที่แล้วมาสามารถเขียนได้ดังนี้

```
LDA
60
00
MOV B, A
LDA
61
00
ADD B
STA
62
00
```

การเขียนโปรแกรมในรูปของตัวอักษณที่เราเข้าใจกันง่ายทำให้สามารถตรวจแก้ไขหรือเข้าใจในการทำงานของโปรแกรมได้ชัดเจน

ภาษาแอสเซมบลี้ (Assembly Language)

การวางรูปโปรแกรมดังกล่าวยังต้องเกี่ยวข้องกับระดับภาษาเครื่องโดยตรงแต่เราไม่ต้องยุ่งเกี่ยวกับตัวเลขไบนารี่หรือตำแหน่งต่าง ๆ ของแอดเดรสมากนัก ลักษณะของรูปแบบโปรแกรมจะเป็นดังนี้

```
เลเบล     นีโมนิค     โอเปอร์แรนด์          หมายเหตุ
START     LDA         VAL  1          : LOAD FIRST NUMBER INTO A
          MOV         B,A             : SAVE IN B
          LDA         VAL  2          : LOAD SECOND NUMBER INTO A
```

เลเบล	นีโมนิค	โอเปอร์แรนด์	หมายเหตุ
	STA	SUM	: STORE SUM
NEXT	?	?	: NEXT INSTUCTION
VALI	DS		
VALI	DS		
SUM	DS		

ลักษณะของภาษาแซมบลี้จะประกอบด้วยฟิลด์ที่สำคัญ ดังนี้

<u>เลเบล</u> เป็นฟิลด์ที่มีไว้สำหรับอ้างอิงถึงในส่วนตัวโปรแกรม เลเบลเปรียบเสมือนกับค่าที่ตำแหน่งแอดเดรสที่จะอ้างถึง ส่วนของเลเบลจะเขียนด้วยลักษณะของอักษรภาษาอังกฤษ และจะจัดวางไว้ในฟิลด์แรก

การที่เรากำหนดเลเบลเป็นตัวอักษรมีข้อดีทำให้ในขณะที่เรากำลังเขียนโปรแกรม เราไม่ต้องพะวงถึงตำแหน่งของแอดเดรสจริง ๆ ในหน่วยความจำที่จะเก็บโปรแกรมไว้ เพราะสิ่งเหล่านี้จะได้รับการกำหนดโดยขบวนการแอสเซมเบลอร์

<u>นีโมนิคฟิลด์</u> เป็นฟิลด์ที่ใช้แสดงการกระทำของคำสั่ง ลักษณะของนีโมนิคจะเป็นคำเฉพาะที่ ที่ได้รับการกำหนดขึ้นมาและรับรู้กันโดยทั่วไปของ 8080 เรากำหนดให้ ADD หมายถึงการบวก และ CMC หมายถึงการคอมพลีเมนท์ค่าในรีจิสเตอร์ A

<u>โอเปอร์แรนด์ฟิลด์</u> เป็นฟิลด์ที่ใช้เก็บค่าตัวแปรหรือข้อมูลที่จะนำมากระทำตามคำสั่งที่อยู่ในนีโมฟิลด์ เช่น START LDA VAL 1 ; ตัว VAL 1 หมายถึงโอเปอร์แรนท์ที่จะได้รับการโหลดมาเก็บไว้ในรีจิสเตอร์ คอมเมนท์ (comment) หรือฟิลด์หมายเหตุ เป็นฟิลด์ที่ไว้สำหรับเขียนอธิบายลักษณะของคำสั่งหรือโปรแกรมว่าส่วนนี้กำลังทำอะไรอยู่ เพื่อให้การอ่านโปรแกรมเป็นไปได้ง่าย

<u>ตัวอย่างโปรแกรมอย่างง่าย</u>

เพื่อให้เข้าใจในการเขียนโปรแกรมของ 8080 ดียิ่งขึ้น คราวนี้ลองมาพิจารณาตัวอย่างการเขียนโปรแกรมดังต่อไปนี้

ตัวอย่าง เป็นการกระทำการคอมพลีเมนท์ข้อมูลที่เก็บไว้ในตำแหน่งหน่วยความจำที่ 4016 แล้วเก็บผลลัพท์ที่ได้ในตำแหน่งหน่วยความจำที่ 4116

ภาษาเครื่องเลขฐานสิบหก		ภาษาแอสเซมเบลอร์			คอมเมนท์
ตำแหน่งหน่วยความจำ	ข้อมูลในหน่วยความจำ	เลเบล	นีโมนิค	โอเปอร์แรนท์	
00	3A	START	LDA	40H	: โหลดข้อมูลจากตำแหน่ง 40 เข้าเก็บรีจิสเตอร์ A
02	00				
03	2F		CMA		: คอมพลีเมนท์ข้อมูล A
04	32		STA	41H	: เก็บผลลัพท์ไว้ที่ตำแหน่ง 4116
05	41				
06	00				
07	C3	HEAR	JMP	HEAR	: ทำซ้ำคำสั่งไปเรื่อย ๆ
08	07				
09	00				

ตัวอย่าง จงเขียนโปรแกรมแสดงการบวกข้อมูลในตำแหน่ง หน่วยความจำ 4016 แล้วเก็บผลลัพท์ไว้ในตำแหน่งที่ 4216

แอดเดรส	ข้อมูล	เลเบล	นีโมนิค	โอเปอร์แรนท์	คอมเมนท์
00	3A		LDA	40H	: GET DATA
01	40				
02	00				
03	87		ADD	A	: SHIFT DATA LEFT
04	32		STA	41H	: STORE RESULT
05	41				
46	00				
07	C3	HEAR	JMP	HEAR	
08	07				
09	00				

ตัวอย่าง จงเขียนโปรแกรมทำการเลื่อนข้อมูลที่อยู่แอดเดรส 4016 ไปซ้าย 1 บิท แล้วเก็บผลลัพท์ไว้

แอดเดรส	ข้อมูล	เลเบล	นิโมนิค	โอเปอร์แรนท์	คอมเมนท์
00	3A				
01	40				
02	00				
03	87		ADD	A	
04	32		STA		
05	41				
06	00				
07	C3	HEAR	JMP	HEAR	
08	07				
09	00				

จากผลของการกระทำตามโปรแกรมข้างบน ถ้าสมมติว่าเดิมหน่วยความจำตำแหน่ง (40)16 ด้วยค่า DE ลักษณะการเลื่อนบิทในลักษณะนี้จะมีความหมายในการคูณด้วย 2 เราจึงใช้วิธีการบวกตัวมันเองหนึ่งครั้งได้

ตัวอย่าง จงแสดงการเขียนโปรแกรมเคลียร์ข้อมูล ในหน่วยความจำตำแหน่งที่ 4016.

แอดเดรส	ข้อมูล	เลเบล	นีโมนิค	โอเปอร์แรนท์	คอมเมนท์
00	97		SUB	A	;
01	32		STA	40H	; CLEAR LOCATION 40
02	40				
03	00				
04	C3	HEAR	JMP	HEAR	
05	04				
06	00				

ตัวอย่าง ในการแบ่งข้อมูลของหน่วยความจำตำแหน่งที่ 40 ออกเป็นสองส่วน ๆ ละ 40 บิทแล้วเก็บไว้ในหน่วยความจำตำแหน่งที่ 41 และ 42 โดยให้ส่วน 4 บิท มีนัยสำคัญมากที่สุดจากตำแหน่ง 40 มาเก็บไว้ที่ตำแหน่งที่มีความสำคัญน้อยที่สุดของตำแหน่งที่ 41 และส่วนที่มีนัยสำคัญน้อยที่สุด 4 บิท ของตำแหน่ง 40 มาเก็บไว้ที่ตำแหน่งที่มีนัยสำคัญน้อยสุดของตำแหน่ง 42 ให้ทำการเคลียร์ 4 บิทที่มีนัยสำคัญมากสุดของตำแหน่ง 4116 และ 42

แอสเดรส	ข้อมูล	เลเบล	นีโมนิค	โอเปอร์แรนท์	คอมเมนท์
00	21	START	LXI	H,40H	; GET DATA
01	40				
02	00				
03	7E		MOV	A,M	; GET DATA
04	47		MOV	B,A	
05	0F		RRC		; SHIFT DATA RIGHT 4 TIMES
06	0F		RRC		
07	0F		RRC		
08	0F		RRC		
09	E6		ANI	00001111B	; MASK OFF LSBS
0A	0F				
0B	23		INX	H	
0C	77		MOV	M,A	; STORE MSBS
0D	78		MOV	A,B	; RESTORE ORIGINAL DATA
0E	E6		ANI	00001111B	; MASK OFF MSBS
0F	0F				
10	23		INK	H	
11	77		MOV	M,A	; STORE LSBS
12	C3	HEAR	JMP	HEAR	; STORE LSBS
13	12				
14	00				

ตัวอย่าง สมมติว่า 2 ตัว จงเขียนโปรแกรมการตรวจสอบข้อมูลสองตัวนี้ว่าตัวใดมีค่ามากกว่า ในที่นี้เราสมมติว่าข้อมูลที่ต้องการตรวจสอบเก็บไว้ที่หน่วยความจำตำแหน่งแอดเดรส 40 16 และ 4116 ตัวโปรแกรมที่เขียนได้จะมีลักษณะดังนี้

แอดเดรส	ข้อมูล	เลเบล	นีโมนิค	โอเปอร์แรนท์	คอมเมนท์
00	21		LXI	H,40H	; GET FIRST OPERAND
01	40				
02	00		MOV	A,M	; GET FIRST OPERAND
03	7E		INX	H	
04	23		CMP	M	; IS SECOND OPERAND LARGER
05	BE				
06	D2				
07	0A				
08	00				
09	7E		MOV	A,M	; YES, GET SECOND INSTEAD
0A	23	DONE	INX	H	
0B	77		MOV	M,A	; STORE LARGER OPERAND
0C	C3				
0D	0C	HEAR	JMP	HEAR	
0E	00				

จากโปรแกรมนี้อาศัยหลักการทดสอบแฟลกที่มีผลจากการเปรียบเทียบในคำสั่ง CMP โดยการนำเอาข้อมูลจากหน่วยความจำเปรียบเทียบกับข้อมูลในรีจิสเตอร์ A

ตัวอย่าง สมมติว่า มีตัวเลขขนาด 16 บิท สองตัว ตัวที่หนึ่งเก็บไว้ที่หน่วยความจำตำแหน่งที่ 4016 และ 4116 ส่วนตัวที่สอง เก็บไว้ที่ตำแหน่ง 4216 และ 4316 จงเขียนโปรแกรมบวกข้อมูล 16 บิทแล้วนำผลลัพท์ที่บวกได้เก็บไว้ในหน่วยความจำตำแหน่ง 4416 และ 4516

แอดเดรส	ข้อมูล	เลเบล	นีโมนิค	โอเปอร์แรนท์	คอมเมนท์
00	2A		LHLD	40H	; LOAD TWO BYTE INTO HL REG
01	40				
02	00				
03	EB		XCHG		; EXCHANGE HL WITH DE
04	2A		LHLD	42H	; LOAD TWO BYTE INTO HL REG
05	42				
06	00				
08	22		SHLD	44H	; STORE RESULT INTO MEN
09	44				
0A	00				
0B	C3	HEAR	JMP	HERE	
0C	0B				
0D	00				

ตัวอย่างข้างบนถ้าสมมติว่าแต่เดิมข้อมูลในหน่วยความจำต่าง ๆ มีดังนี้

(40) = 2A

(41) = 67

(42) = F8

(43) = 14

ผลลัพท์ที่ได้จากโปรแกรมคือ

(44) = 22

(45) = 7C

การสร้างตารางและการมองหาตารางเบื้องต้น

ตัวอย่าง สมมติว่าต้องการให้เครื่องคำนวณค่ากำลังสองของตัวเลขจาก 0-7 ในการเขียนโปรแกรมเราอาจทำได้ด้วยการสร้างตารางขึ้นที่ส่วนหนึ่งส่วนใดของหน่วยความจำ เพื่อเก็บผลลัพท์ไว้ และเมื่อต้องการหาค่ากำลังสอง ก็ใช้วิธีมองหาตารางโดยอาศัยวิธีการสร้างอินแดด เพื่อหาว่าผลลัพท์คืออะไร เมื่อเป็นเช่นนี้การกำหนดค่าในตารางเราต้องรู้ค่าที่สัมพันธ์กับแอดเดรสที่แน่นอน

ตัวอย่าง ตาราง เช่น

ตำแหน่ง	ข้อมูล	ข้อมูลเลขฐานสิบ
60	00	0 (0^2)
61	01	1 (1^2)
62	04	4 (2^2)
63	09	9 (3^2)
64	10	16 (4^2)
65	19	25 (5^2)
66	24	36 (6^2)
67	31	49 (7^2)

การมองหาตารางเราก็ใช้วิธีสร้างอินเดค เช่น ต้องการรู้ว่าผลยกกำลังสอง ของตัวเลข 3 มีค่าเท่าไร ก็ย่อมหาได้จากตาราง ลักษณะโปรแกรมที่ใช้หาจะเป็นดังนี้

```
00    3A        LAD     40H
01    40
02    00
03    6F        MOV     L,A
04    26        MVI     H,0
05    00
06    11        LXI     D,60H
07    60
08    00
09    19        DAD     D
```

```
0A          7E              MOV     A,M
0B          32              STA     41H
0C          41
0D          00
0E          C3      HEAR    JMP     HEAR
```

การสร้างตารางเป็นเทคนิคในการเขียนโปรแกรมที่ดีมากวิธีหนึ่งและเราสามารถนำมาใช้ได้ค่อนข้างมาก เช่น สร้างตารางค่าของ sin ก็เรียกมาจากตารางโดยทราบว่าที่แอดเดรสเท่าไรเป็นค่าไซน์มุมใด หรือในบางกรณีเราอาจสร้างตารางในการใช้ถอนรหัสบางอย่าง เช่น ใช้ถอดรหัสในภาคแสดงด้วย LED เจ็ดส่วนให้เป็นตัวอักษรพิเศษบางอย่างขึ้นเองได้โดยไม่ต้องอาศัยวงจรทางฮาร์ดแวร์ในการใช้ถอดรหัสเช่น ให้รหัส 7ส่วนเป็นอักษร A-F ดังนี้ ABCDEF เป็นต้น

ในส่วนของโปรแกรมโมนิเตอร์ก็ดี โปรแกรมการแปลรหัสแอสเซมเบลอร์ก็ดี เรามักใช้วิธีการสร้างตารางเช่นว่านี้เพราะจะทำให้ประหยัดเนื้อที่ในหน่วยความจำ

<u>คำสั่งเทียมที่ใช้ในการเขียนโปรแกรมภาษาแอสเซมบลี้ (Pecudo Assembly)</u>

ในการเขียนภาษาแอสเซมบลี้จะต้องเกี่ยวข้องกับตำแหน่งต่าง ๆ ในส่วนของโปรแกรมและถึงแม้ว่าเราจะหลีกเลี่ยงลักษณะของคำสั่งในภาษาเครื่องมาเขียนด้วยนีโมนิคส์แล้วก็ตาม ความยุ่งยากในการสร้างนิยามหรือการกำหนดค่าของตัวแปรก็เกิดขึ้น เราจึงหาวิธีการทำจะทำให้การเขียนโปรแกรมเป็นไปได้ง่ายด้วยการกำหนดคำสั่งใหม่เพิ่มเติมขึ้นมา คำสั่งที่กำหนดขึ้นมาใหม่นี้จะไม่มีรหัสภาษาเครื่องแทน คำสั่งนี้จึงเป็นคำสั่งเทียม (Pseudo Instruction)

1. ORG ย่อมาจาก ORIGIN

คำสั่งนี้ไว้สำหรับให้ผู้โปรแกรมมากำหนดแอดเดรสของตัวแปรหรือโปรแกรมย่อยว่าจะอยู่ในแอดเดรสใด เช่น

```
ORG     100
MVI     SP, DAT
```

ในที่นี้หมายถึงว่าเราเริ่มต้นโปรแกรมที่แอดเดรส 100 หรือ คำสั่ง MVI อยู่ที่แอดเดรส 100

ORG อาจใช้กำหนดอยู่ในรูปของตัวแปรก็ได้เช่น

```
ORG     RESET
MVI     SP, DAT
```

ในที่นี้ค่า RESET จะต้องถูกนิยามค่ามาก่อน

2. EQU มาจากคำว่า EQUATE

คำสั่งนี้เป็นการกำหนดค่าให้กับตัวแปรนั่นเอง ลักษณะของการใช้จะเป็นดังนี้

```
RESET EQU 00
      ORG  RESET
      MVI  SP, DAT
```

ในกรณีนี้เป็นการกำหนดค่า FINAL ให้เท่ากับ TTY

3. DS มาจากคำว่า DATA STORAGE

คำสั่งนี้เป็นการกำหนดค่าว่าจะเก็บไว้ในอะเรย์ของหน่วยความจำ จำนวนกี่ไบท์นั่นเอง

```
TEMP  EQU
      ORG  TEMP
MAN   DS   2
```

มีความหมายว่า ตัวแปร MAN จะเก็บไว้ในหน่วยความจำ 2 ไบท์ คือที่แอดเดรส 00 กับ 01

4. RESERVE ลักษณะการใช้เหมือนกับ DS นั่นเองคือจะเป็นการกำหนดพื้นที่ของหน่วยความจำขึ้นมาซึ่งอาจจะใช้สำหรับเป็นกลุ่มข้อมูลเป็นตาราง หรือแสตค ลักษณะวิธีใช้จะเป็นดังนี้

```
      ORG  3000
BUF 1 RESERVE 100
```

ในที่นี้ BUF 1 จะประกอบด้วยกลุ่มหน่วยความจำจากแอดเดรส 3000 นับไปอีก 100 ที่เรียงกันไป

<u>การสร้างลูป (loop) ในโปรแกรม</u>

ในภาษาชั้นสูงเราสามารถกำหนดการทำงานเป็นลูปของโปรแกรมได้ง่ายเช่น ในภาษา BASIC เรากำหนดการบวกข้อมูล จาก 1-10 ได้ดังนี้

```
10    J    =  0
20    FOR I =  1 TO 10 STEP 1
30    J    =  I + J
40    NEXT I =
50    ANS  =  J
```

หรือกรณีภาษา FORTRAN เรามี DO กำหนด LOOP หรือการทำงานเป็นลูปได้ง่าย เพียงแต่บอกจำนวนลูปที่ต้องการจะทำตามกำหนด

ในส่วนของภาษาแอสเซมบลี้นั้นกวรกำหนดลูปเราก็ทำได้เช่นเดียวกันแต่จะต้องมีลักษณะบางอย่างที่ยุ่งยากกว่า เพราะที่เราจะต้องกำหนดในโปรแกรมประกอบด้วยส่วนสำคัญสี่ส่วนคือ

1. ส่วนการกำหนดค่าเริ่มต้น เช่นการกำหนดจำนวนที่ต้องการโดยการกำหนดขึ้นที่ส่วนของค่าในควอเตอร์ที่เราต้องกำหนดค่าตัวแปรเริ่มต้นที่เราต้องการ
2. ส่วนของการประมวลผลจากข้อมูลที่เราต้องการ
3. ส่วนควบคุมลูป ซึ่งเป็นส่วนหนึ่งของการปรับค่าในควอเตอร์เพื่อนับว่าครบตามจำนวนหรือยัง
4. ส่วนสรุปหรือเก็บผลลัพท์ที่ต้องการ

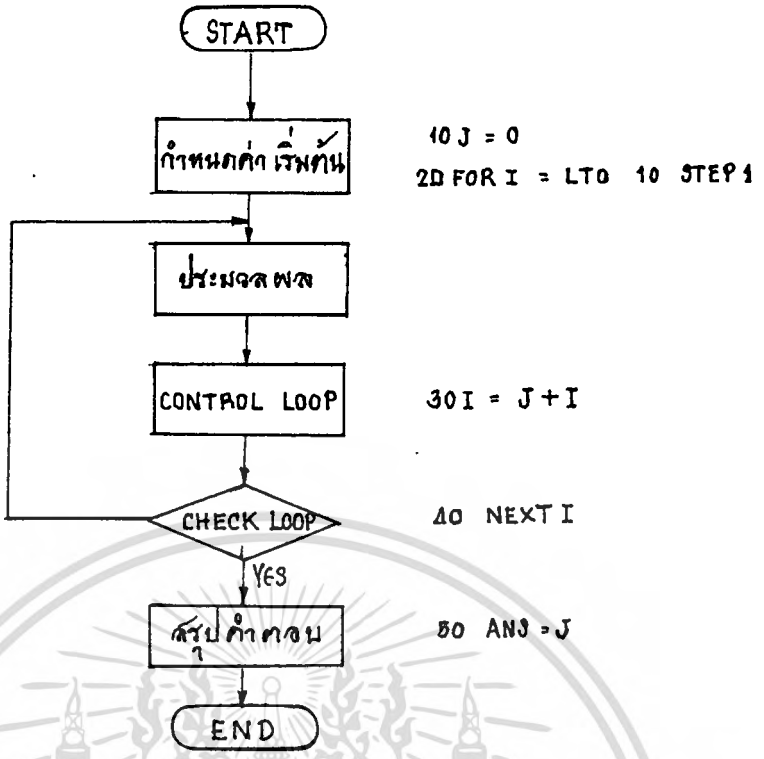
เพื่อเข้าใจในเรื่องโปรแกรมได้ดียิ่งขึ้น ดังตัวอย่างต่อไปนี้

จงเขียนโปรแกรมแสดงการบวกกันของข้อมูล โดยจำนวนครั้งที่ใช้ในการบวกจะเก็บไว้ในหน่วยความจำครั้ง เก็บผลลัพท์ที่ได้ไว้ที่หน่วยความจำตำแหน่ง

หมายเหตุ เพื่อลดข้อยุ่งยากการเขียนรหัสภาษาเครื่องฐานสิบหกจึงขอเขียนโปรแกรมด้วยสัญญลักษณ์ทางนีโมนิคแต่เพียงอย่างเดียว

```
LXI   H, 41H     ; COUNT  =  LENGTH OFSERIES OF NUMBER
MOV   B, M
SUB   A
SUMD  INX  H
ADD   M          ; SUM  = SUM + DATA
DCR   B
JNZ   SUMD       ; CHECK NO OF LOOP
STA   40H        ; STORE SUM
HLT
```

จากกรณีนี้เรามาดูที่ผังงานจะเป็นดังรูปที่

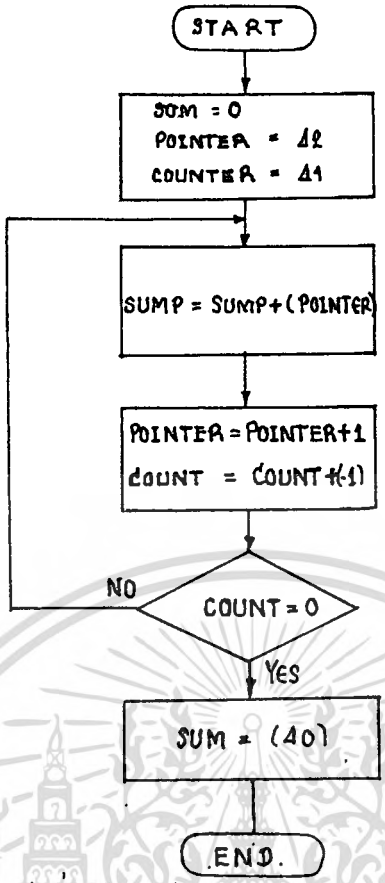


ในการบวกตัวเลขหลาย ๆ ไบท์นั้นผลบวกอาจจะเกินกว่า 8 บิทได้ เช่น C8+FA+96 จะมีค่าเป็น 0258 H ซึ่งมีค่ามากกว่า 8 บิท จากกรณีนี้เราดัดแปลงโปรแกรมที่กล่าวถึงแล้วให้เป็นได้ดังรูปที่ 5.3

จากตัวอย่างข้างบนเราสมมติว่าจำนวนลูปของการบวกกำหนดด้วยหน่วยความจำตำแหน่งที่ 4216 และข้อมูลเราเริ่มบวกกำหนดด้วยหน่วยความจำตำแหน่งที่ 4316 ส่วน SUMH เก็บที่ 4116

```
        LXI     H, 42h
        MOV     B,M             ; CONT  =  LENGTH OF SERIES
        SUB     A               ; LSB OF SUM  =  0
        MOV     C,A             ; MSB OF SUM  =  0
DSUMD   INX     H
        ADD     M               ; SUM  =  SUM + DATA
        JNC     CHCNT
        INR     C               ; ADD CARRY TO MSB OF SUM
CHCNT   DCR B
        JNZ D SUMD
        LXI     H, 40H          ; STORE LSB OF SUM
        MOV     M,A
        INX     H               ; STORE MSB OF SUM
```

```
MOV    M,C
HLT
```



ตัวอย่าง สมมติว่ามีข้อมูลอยู่หนึ่งบล๊อก ส่วนของความยาวบล๊อกเรากำหนดไว้ด้วยหน่วยความจำตำแหน่งที่ 4116 ส่วนจุดเริ่มต้นของบล๊อกอยู่ที่ 4216 จงหาค่าที่มากที่สุดของข้อมูลที่เก็บไว้ในหน่วยความจำบล๊อกที่กำหนด

สำหรับตัวโปรแกรมตามผังงานเขียนได้เป็น

```
LXI     H, 41H      ; POINT TO COUNT
MOV     B,M         ; COUNT = NUMBER OF ELEMENTS
SUB A               ; MAXIMUM  =  0
NEXT    INX  H
CMP     M           ; IS NEXT ELEMENT MAXIMUM
INC     DECN
MOV     A,M         ; YES REPLANCE MAXIMUM
DECN DCR B
JNZ     NEXT
STA     40 H        ; SAVE MAXIMUM
HLT
```

การกำหนดผังงาน

การออกแบบระบบซอร์ฟแวร์

การออกแบบระบบซอร์ฟแวร์เหมือนกับการออกแบบวงจรอิเลคทรอนิคส์ หรือเครื่องจักรกลทั่ว ๆ ไป ซึ่งจะมีเริ่มต้นด้วยการกำหนดขอบข่ายปัญหาที่ต้องการจะแก้ไข และรายละเอียดของข้อมูลที่ต้องการ หรือผลลัพท์ ในทางซอร์ฟแวร์เราจำเป็นต้องกำหนดโครงสร้างของโปรแกรมโดยยึดหลักการดังต่อไปนี้

1. แจกแจง และตีความหมายของปัญหารวมทั้งคิดแนวทาง หรือวิธีที่จะใช้ในการแก้ปัญหานั้น ๆ

2. กำหนดผังโปรแกรม

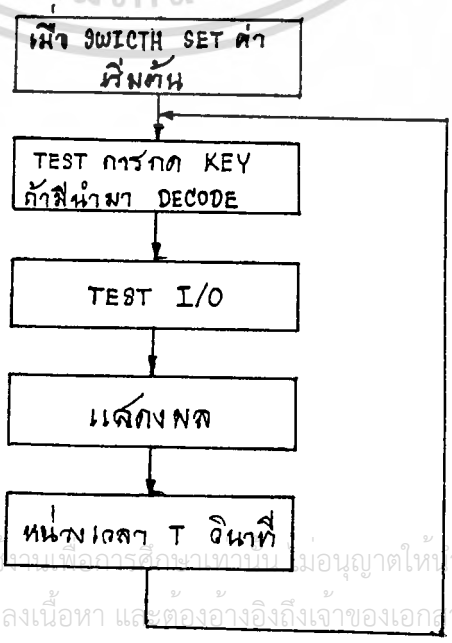
3. เขียนรายละเอียดส่วนย่อยในโปรแกรม

จากกรณีนี้เราลองยกตัวอย่างระบบซอร์ฟแวร์ง่าย ๆ ที่ใช้ไมโครคอมพิวเตอร์ให้ดูสักตัวอย่างหนึ่ง โดยสมมติว่าเรามีซีพียูที่ติดต่อกับอินพุทหลายตัว และอินพุทตัวหนึ่งอาจจะเป็นคีย์บอร์ด และเอาท์พุทอีกตัวหนึ่งอาจจะเป็นภาคแสดง LED สิ่งที่เราต้องการคือ ในการคีย์โปรแกรมหรือข้อมูลผ่านทางคีย์บอร์ดนั้นจะให้ผลของปัญหาและคิดหาวิธีได้แล้วก็จะมาเขียนผังงานตามวิธีที่เลือกไว้

วิธี พอลลิ่ง POLLING

จากกรณีผังงานรูปที่ 5.5 จะเห็นว่าจะต้องให้ซีพียูคอยตรวจดูเป็นระยะ ๆ ว่าคีย์บอร์ดมีการคีย์อะไรมาหรือไม่ ถ้าไม่มีก็ผ่านไป ทดสอบ I/O ตัวอื่นต่อไป ดังนั้นซีพียูจะต้องเสียเวลามาคอยวนอ่านคีย์บอร์ดอยู่เสมอ ๆ

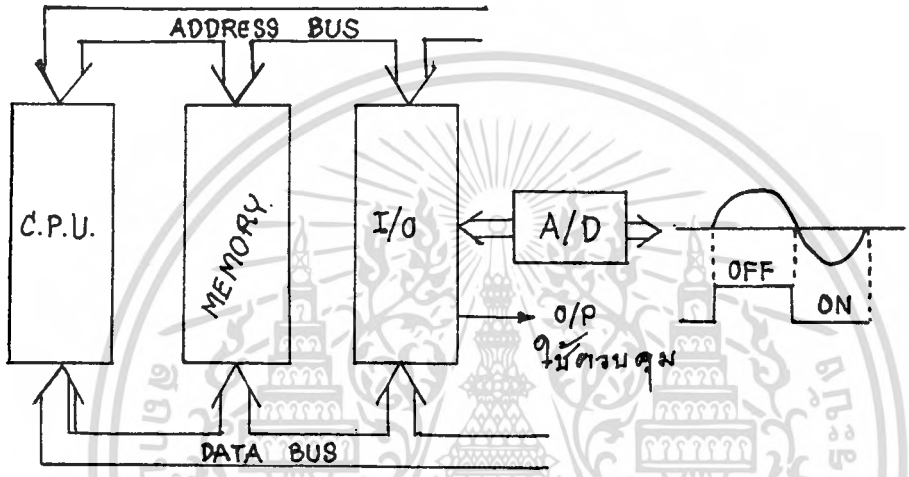
วิธีการอินเตอร์รัพท์ ในกรณีนี้ผังงานอาจจะลดข้อยุ่งยากทางซอร์ฟแวร์บ้าง แต่จะยุ่งยากในวงจรอินเตอร์รัพท์อยู่บ้าง ลักษณะของการเขียนเป็นผังงานของการกระทำในกรณีนี้ดังนี้



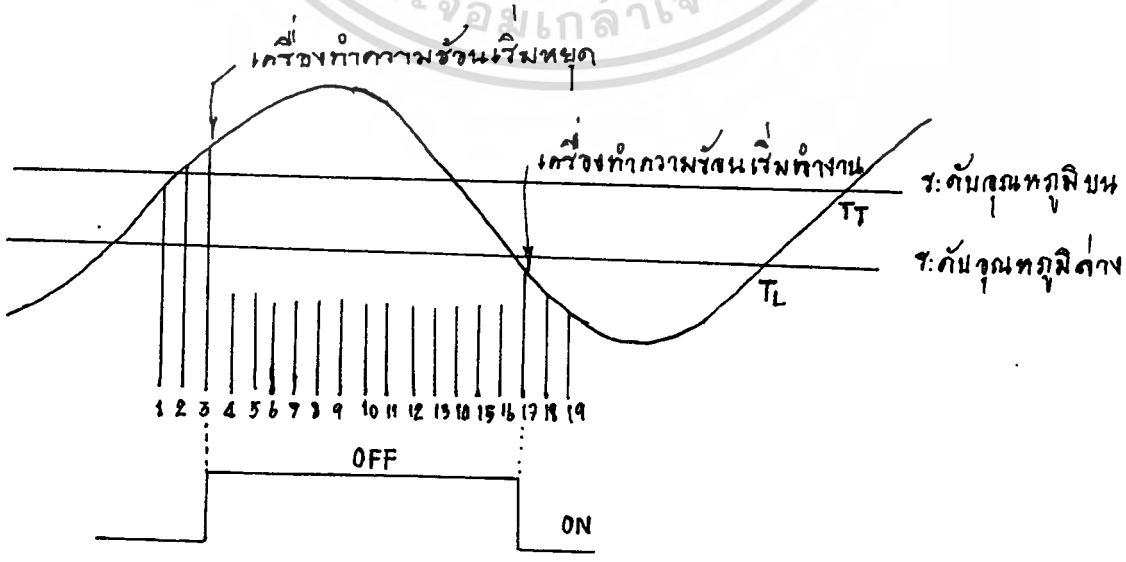
ในการอินเตอร์รัพท์ ซีพียูจะต้องสนองทันทีที่มีสัญญาณอินเตอร์รัพท์เข้ามา เรามีเทคนิครายละเอียดเกี่ยวกับอินเตอร์รัพท์อีกมาก ซึ่งจะได้อธิบายให้เห็นในบทต่อไป

<u>ตัวอย่างการวางระบบซอร์ฟแวร์</u>

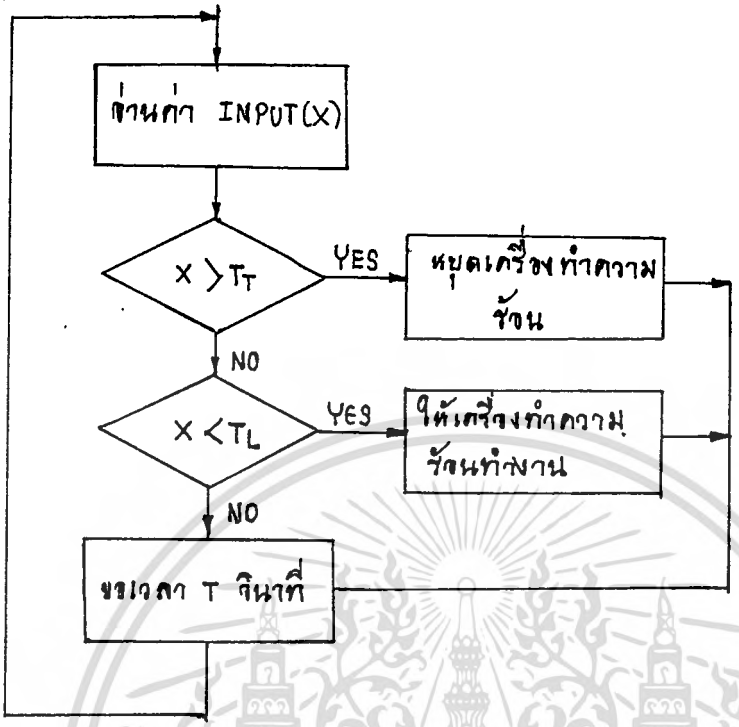
เพื่อเป็นแนวทางอย่างง่ายจึงขอยกตัวอย่างง่ายให้เป็นแนวทางในการเข้ามจ ในที่นี้สมมติว่าต้องการระบบซอร์ฟแวร์ที่ใช้ควบคุมอุณหภูมิของโรงงานอุตสาหกรรมแห่งหนึ่งระบบคอมพิวเตอร์เป็นระบบดังรูป



สิ่งที่ต้องการคือลักษณะของสัญญาณอินพุทและเอาท์พุทที่ต่ออยู่กับไมโครคอมพิวเตอร์ โดยอินพุทเป็นค่าอุณหภูมิที่เราวัดได้ ส่วนเอาท์พุทคือสัญญาณที่จะไปควบคุมการทำงานของเครื่องทำความร้อน ลักษณะของสัญญาณทั้งสองสัมพันธ์กันดังนี้



จากจุดมุ่งหมายที่เราได้วางไว้เรานำมาเขียนผังงานในการทำงานได้ดังนี้



ในส่วนของการรอเวลานั้นเราอาจใช้วิธีกำหนดลูปให้เก็บการทำงานโดยคำนวณหาจำนวนลูปที่พอดีทำให้ระยะที่ได้ตามต้องการ ลักษณะเช่นนี้เราเขียนเป็นผังงานในส่วนการรอเวลาได้ดังนี้

การเขียนโปรแกรมที่เกี่ยวข้องกับข้อมูลที่เป็นตัวอักษรหรือตัวเลข

บ่อยครั้งที่เราอดแปลกใจไม่ได้ว่า การทำงานของโปรแกรมก็ดี การใช้งานก็ดีมักจะใช้ลักษณะของคำสั่งที่ THEN Y = 3 : NEXT I ทั้ง ๆ ที่การทำงานของเครื่องคอมพิวเตอร์จะใช้รหัสคำสั่งในรูปภาษาเครื่องที่เป็นเลขไบนารี่ การที่เราแทนลักษณะคำสั่งหรือข้อความแล้วเราเข้าใจได้ดีเป็นสิ่งที่นิยมทำกันมากเช่น ADD หมายถึงการบวก

เราทราบแล้วว่าตัวอักษรในภาษาอังกฤษนั้น สามารถแทนเป็นรหัสตัวเลขไบนารี่ได้ ซึ่งขึ้นอยู่กับมาตรฐานที่ใช้ได้ และมาตรฐานได้มากถึง $2^7 = 128$ ตัวอักษร

ลักษณะเด่นของรหัส ASC II นั้นเป็นสิ่งที่เราจะต้องเข้าใจในหลักการทั่ว ๆ ไปก่อนเช่น ถ้าเราต้องการจะพิมพ์ตัวเลข 9 ที่เครื่องพิมพ์ไมโครคอมพิวเตอร์จะต้องส่งรหัส 3916 มาให้เครื่องพิมพ์ รหัสที่ใช้เป็นตัวเลขและตัวอักษรแล้ว ยังมีรหัสอีกกลุ่มหนึ่งที่ใช้เป็นรหัสในการควบคุมระบบโดยรหัสเหล่านี้เป็นที่ยอมรับกันอยู่แล้ว เช่น 0A16 หมายถึงการเลื่อนบรรทัด (LF), 0D หมายถึงรหัสแทน carriage return (CR), รหัส 20 หมายถึง space (SP) เป็นต้น

เมื่อระบบที่ใช้มีรหัสในการควบคุมที่ยอมรับแล้ว การกำหนดการทำงานระบบจึงทำให้ง่าย และจะมีประสิทธิภาพดียิ่งขึ้น เช่น เมื่อต้องการให้ตัวอักษรบนจอ CRT ขึ้นบรรทัดใหม่เครื่องคอมพิวเตอร์ก็เพียงแต่ส่งรหัสตัวอักษร CR ไปให้เท่านั้น

โดยปกติการวางรูปของข้อมูลตัวอักษรหรือตัวเลขก็ดีมักผูกรวมกันเป็นกลุ่ม ๆ ในกรณีตัวเลขก็จะรวมกลุ่มกันเป็นตัวเลขอินทีเจอร์ (integer) เลขจำนวนจริง (real) เลขโฟลตติ้งพอยท์ (floating point) สำหรับกลุ่มตัวเลขอินทีเจอร์หรือจำนวนเต็มอาจใช้จำนวนจริง 3-4 ไบท์รวมกันและตัวเลขโฟลตติ้งพอยท์ก็อาจรวมข้อมูลถึง 8 ไบท์ เป็นต้น

ในกรณีของตัวอักษรเราก็จะผูกรวมกันเป็นข้อความที่เรียกว่าสติง เช่น ข้อความว่า GOOD รวมกันเป็นสติง เพื่อให้เข้าใจเกี่ยวกับการนำรหัสข้อมูล ASCII มาใช้เป็นประโยชน์จึงขอยกตัวอย่างดังนี้

ตัวอย่าง ในการถ่ายข้อมูลจากหน่วยความจำไปยังเครื่องพิมพ์นั้นต้องการนับจำนวนสติงของข้อมูลที่ส่งไปให้เครื่องมีจำนวนตัวอักษรทั้งหมดกี่ตัว โดยข้อมูลที่จะส่งเริ่มต้นตั้งแต่แอดเดรส 4116 เป็นต้นไป การจบของสติงจะปิดท้ายด้วยสัญญาณ CR ที่นับได้ให้นำหน้าไปเก็บไว้แอดเดรส 4016

จากผังงานเราจะเขียนโปรแกรมตามผังงานได้

```
        LXI   H,   41H      ; POINTER  =  START OF STRING
        MVI   B, 0          ; LENGTH   =  0
CHKCR   CMP M               ; IS CHARACTER CR ?
```

```
JZ    DONE          ; YES END OF STRING
INR   B             ; NO, ADD 1 TO LENGTH
INX   H
JMP   CHKCR         ; EXAMINE NEXT CHARACTER
DONE MOV A,B
STA 40H .           ; SAVE STRING LENGTH HERE JMP HERE
```

ในกรณีของอุปกรณ์เฟอริเฟรัลเช่น เครื่องพิมพ์จะรับจากคอมพิวเตอร์มาพิมพ์ ส่วนตัวและรหัสควบคุม เครื่องนั้นเครื่องจะรู้เองและจะไม่มีการพิมพ์แต่จะรับรู้ และกระทำตามคำสั่งรหัสควบคุม

การเปรียบเทียบข้อมูลด้วยคำสั่ง CMP นั้น ผลที่ได้ในขณะที่เท่ากันจะทำให้แฟลกศูนย์ ได้รับการเซ็ท ครั้นเราทำการตรวจสอบ JZ มันก็จะทำงานโดยกระโดดต่อไปได้

ลักษณะของโปรแกรมดังกล่าวนี้สามารถเขียนผังงานได้อีกแบบหนึ่งเป็น

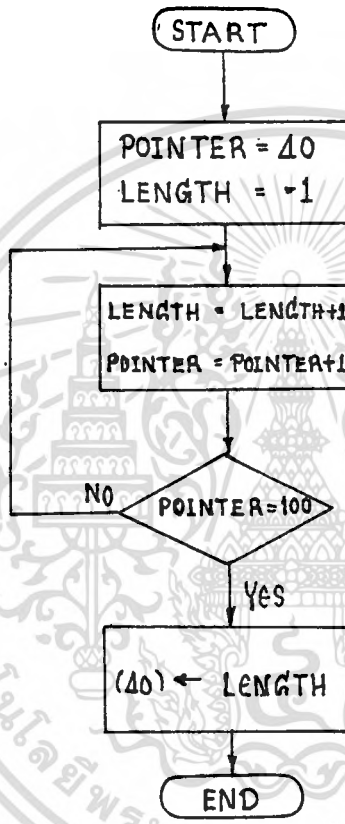
```
LXI   H, 40H        ; POINTER = BYTE BEFORE STRING
MVI   B OFFH        ; LENGTH  = -1
MIV   A,oDH
CHKCR  INX  H
INR   B             ; ADD 1 TO LENGTH
CMP   M             ; IS CHARACTER 'CR' ?
JNZ   CHKCR         ; NO, KEEP COUNTING
MOV   A,B
STA 40H             ; YES, SAVE STRING LENGTH
                      HEAR JMP HERE
```

ลักษณะการเปรียบเทียบทำการตรวจสอบบล๊อก ในตัวอย่างนี้เป็นลักษณะที่นำไปประยุกต์ใช้โหลดข้อมูลจากเทปคาสเซ็ทหน่วยความจำ ซึ่งมาตรฐานคือ

INTEL HEX FORMAT จะใช้ตัวอักษรใน ASCII คือ CR ขึ้นเป็นตัวกำหนดข้อมูลในบล๊อกนั้นแล้ว

ตัวอย่างโปรแกรมแทนเลขฐานศูนย์ที่อยู่ข้างหน้าด้วยแบลงค์

การเติมแบลงค์แทนเลขฐานศูนย์ สำหรับตัวเลขนั้นมีความจำเป็นในทางผลลัพท์ทางคณิตศาสตร์ ในกรณีของเครื่องพิมพ์เลข 0 อยู่ข้างหน้า ต่อไปนี้เป็นตัวอย่างที่ยกขึ้นมาให้เห็นจริงโดยกำหนดว่าสตริงของตัวเลขนี้เริ่มต้นที่หน่วยความจำแอดเดรส 4016 โดยที่จำนวนความยาวของสตริงจะเก็บไว้ที่แอดเดรส 4016 ลักษณะของผังงานเขียนได้เป็น



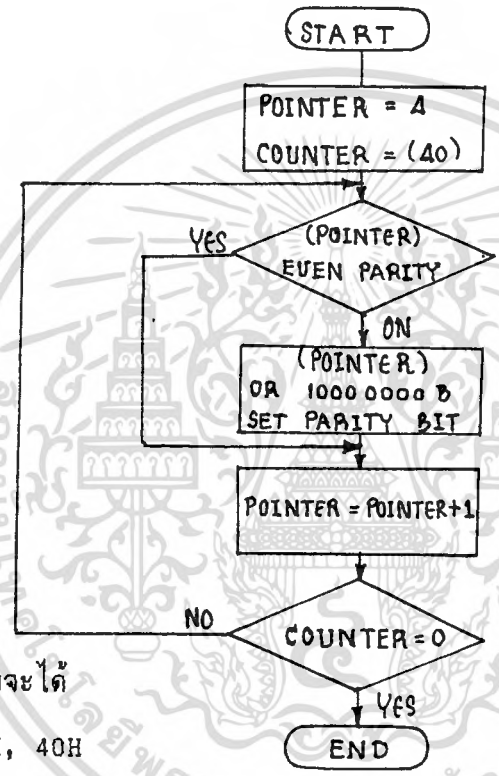
จากผังงานเราเขียนโปรแกรมได้เป็น

```
        LXI    H, 40H
        MOV    B, M              ; COUNT = STRING LENGTH
        MVI    A, 0              ; GET ASCII 0 FOR COMPARISION
CHKZ :  INK H
        CMP    M                 ; NO, THROUGH
        JNZ    DONE              ; REPLACE-LEADING ZERO WITH BLANK
        DCR    B                 ; HAVE ALL DIGITS BEEN EXAMINED ?
        JNZ    CHKZ              ; NO. GO EXAMINE NEXT HLT
```

ตัวอย่าง การเพิ่มบิทพาริตี้ ให้กับตัวอักษรใน ASCII

ในการรับส่งข้อมูลทางคอมพิวเตอร์ทั่วไปมักจะมีการผิดเพี้ยนไปจากข้อมูลเดิม การผิดนั้นจะมีโอกาสที่ผิดพลาดวิธีหนึ่งที่ง่ายก็ได้แก่การเพิ่มบิทของข้อมูล ในแง่ตัวอักษรเดิมผิดไป แต่บิทที่เติมจะเป็น "0" "1" ก็ได้ ที่พอดีทำให้ผลบวกของตัวเลข 1 ก็ได้

ในตัวอย่างนี้สมมติว่าสตริงของตัวอักษร ASCII ชนิด 7 บิท สิ่งที่เราต้องการคือในตัวอักษรทุกตัวจะเพิ่มบิทที่เป็นพาริตี้เพื่อให้แสดงว่าพาริตี้เป็นคู่ จากลักษณะเช่นว่านี้เขียนผังงานได้เป็นดังนี้



เมื่อเขียนโปรแกรมจะได้

```
LXI     H, 40H
MOV     B, M            ; GET STRING LENGTH
MVI     C, 10000000     ; GET PARITY BIT OF 1
SETPR   INX   H
MOV     A, M            ; GET A CHARACTER
ORA     C               ; SET PARITY BIT TO 1
JPO     CHCNT           ; IS PARITY NOW EVEN ?
MOV     M, A            ; YES, SAVE CHARACTER WITH EVEN PARITY
CHCNT   DCR   B
JNZ     SETPR
HLT
```

จากลักษณะที่สำคัญของโปรแกรมนี้อยู่ที่การกำหนดค่า 10000000 ขึ้นรีจิสเตอร์ C เพื่อนำมา OR นี้จะเป็นผลทำให้บิทพาริตี้เป็น 1 เสมอไม่ว่ากรณีใด ๆ ดังนั้นเราจึงมีข้อมูลมาทดสอบที่รีจิสเตอร์ A มีค่าเป็น 1 ที่บิทแรกสุด ครั้นเมื่อเราทดสอบด้วยคำสั่ง JPO หรือ JUMP IF ODD PAIRTY นั่นคือข้อมูลที่อยู่รีจิสเตอร์ A เป็นพาริตี้คู่ก็คือค่าเดิมในหน่วยความจำของมันนั่นเองที่มีค่าพาริตี้คี่ ซึ่งพาริตี้คู่นี้ ก็คือค่าเดิมในหน่วยความจำของมันที่มีค่าพาริตี้ถูกต้องแล้ว ถ้าในริจิสเตอร์ A มาเก็บไว้แทนหน่วยความจำ

เพื่อให้เข้าใจถึงขบวนการดังข้างบน เราสมมติว่าเดิมหน่วยความจำในแอดเดรสต่าง ๆ เก็บข้อมูลไว้ดังนี้

(40) = 06

(41) = 31

(42) = 32

(43) = 33

(44) = 34

(45) = 35

(46) = 36

หลังจากทำการเอ็กซิดิวแล้วผลที่ได้จะเป็นดังนี้

(41) = B1

(42) = B2

(43) = 33

(44) = B4

(45) = 35

(46) = 36

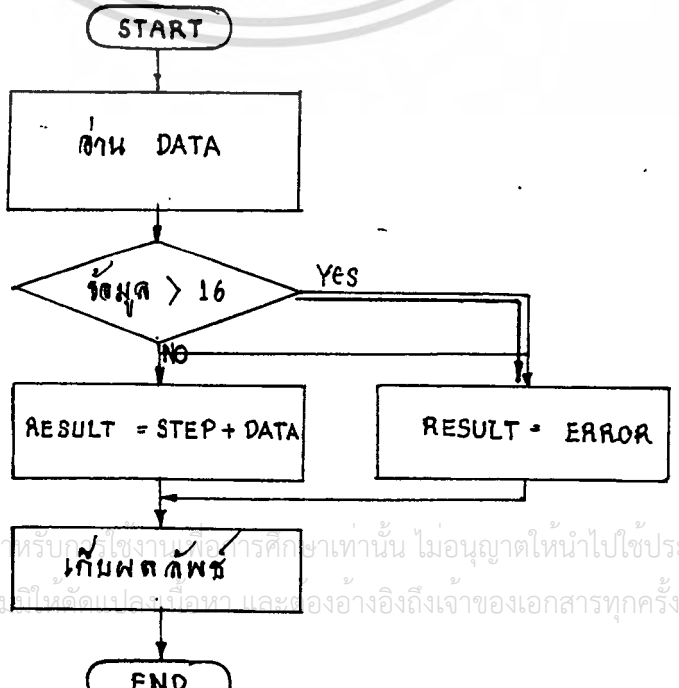
<u>การเขียนโปรแกรมเพื่อเปลี่ยนรหัสจากตัวเลขฐานสิบหกให้แสดงด้วยผลภาคแสดง 7 ส่วน</u>

ในการเขียนโปรแกรมเพื่อควบคุมภาคแสดง LED 7 ส่วน มีเทคนิคในการเขียนหลายแบบด้วยกัน ทั้งนี้มักจะมีส่วนของวงจรทางฮาร์ดแวร์มาเกี่ยวข้องด้วยเสมอ แต่อย่างไรก็ตามที่มักใช้กันมากอย่างหนึ่งก็คือกำหนดข้อมูลในแต่ละบิทแทนแต่ละส่วนของ LED 7 ก็จะแทนในส่วนของข้อมูลเหล่านั้น แล้วจึงใช้โปรแกรมมองหาข้อมูลตามที่ต้องการส่งไปยังภาคแสดง วิธีการเช่นนี้เรียก Table Look Up

ขอให้พิจารณาหลักการสร้างตารางดังนี้

ตัวเลข	รหัส
0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F
A	77
B	1F
C	4E
D	3D

การติดสว่างของ LED จะเป็นไปตามลักษณะบิทโดยที่แสดงค่า 1 จะเป็นบิทที่ติดสว่าง เมื่อสร้างตารางไว้เป็นข้อมูลในหน่วยความจำแล้ว เราก็นำข้อมูลที่อ่านมาหาดูว่าตรงช่องตารางใด ก็เอาข้อมูลออกมาก็จะได้รหัสตามภาคแสดง 7 ส่วน ซึ่งเราจะเขียนเป็นผังงานได้



```
START  MVI B, ERR          ; GET ERROR CODE
       IN   PORT 1         ; READ INPUT
       CPI  16             ; IS DATA > 16
       JCN  DONE           ; YES DONE
       LXI  H, STAB        ; LOAD STARTING TABLE ADDRESS
       MOV  C, A           ; MAKE DATA INTO A 16 BIT INDES
       DAD  BB             ; FIND ADDRESS OF TABLE
       MOV  B, M           ; LOOK UP TABLE
DONE   MOV A, B
       OUT PORT 2
STAB   DB 3FJ, 06H, 4F, 66H, TABLE CONTENT
       DB 6DH, 7DH, 07H, 7FH, 6FH
       DB 77H, 1FH, 4EH, 3DH, 4FH
       DB 47H
ERR    DB 55H              ; ERROR CODE
```

จากกรณีนี้จะเห็นว่าในการทำงานกำหนดข้อมูล (DB = Defind Byte) นั้นแอดเดรสสุดท้าย คือ 55 H นั้นเป็นรหัสของตัวบอกข้อผิดพลาดในกรณีที่มีรหัสเข้ามามากกว่า 15 ลักษณะเช่นนี้เราสามารถกำหนดอักษรต่าง ๆ จากภาคแสดง 7 ส่วนได้เป็นอย่างดี โดยการสร้างตารางของตัวอักษรเหล่านั้นขึ้น และสมมติว่าเมื่อเรากดคีย์บอร์ดของตัวอักษรตัวหนึ่งเข้ามา ซีพียูจะอ่านข้อมูลแล้วตีความมองหาข้อมูลจากตารางแล้วส่งผลลัพท์ไปที่ภาคแสดงตามที่เราคีย์เข้ามาได้

<u>การเขียนภาษาแอสเซมบลี้ในทางคณิตศาสตร์</u>

โดยหลักการทั่วไปแล้วการเขียนภาษาแอสเซมบลี้ไม่เหมาะกับการคำนวณทางคณิตศาสตร์ที่ยุ่งยากและซับซ้อน เพราะการทำงานทางคณิตศาสตร์เรามักจะมองใกล้ในแง่ของภาษาเครื่อง ทำให้เราต้องยุ่งยากต่อตัวเลขไบนารี่เป็นจำนวนมาก แต่อย่างไรก็ตามในภาษาชั้นสูงที่เหมาะในการทำงานคำนวณ เช่น BASIC FORTRAN ก็อาศัยโครงสร้างการคำนวณพื้นฐานเหล่านี้

การบวกตัวเลขที่มีความเชื่อถือได้หลายหลัก

การทำงานของ 8080 จะกระทำข้อมูลทีละไบท์ (8 บิท) ถ้าพิจารณาในแง่ตัวเลขจะเห็นว่าเป็นจำนวนเลขที่น้อยมาก แต่การคำนวณที่เราใช้นั้นอาจมีจำนวนตัวเลขฐานสิบได้มากกว่า 10 ตัว ดังนั้นเราทำได้โดยไม่ยากนักโดยพิจารณาจากตัวอย่างต่อไปนี้

เมื่อเราเขียนโปรแกรมจะได้ดังนี้

```
      LDA   30 H          ; COUNT = LENGTH OF STRINGS (IN BYTES)
      MOV   B, A
      LXI   H, 41H        ; START POINTER 1 AT FIRST WORD OF STRING 1
      LXI   D, 61H        ; START POINTER 2 AT FIRST WORD OF STRING 2
      AND   A             ; CLEAR CARRY TO START
ADD   LDAX D              ; GET WORD FROM STRING 2
      ADC   M             ; ADD WORD FROM STRING 1
      MOV   M, A          ; STORE RESULT
      INX   D
      INX   H
      DCR   B
      JNZ   ADDW
```

การบวกในลักษณะนี้เราสามารถบวกต่อข้อมูลจำนวนกี่ไบท์ต่อกันก็ได้ แต่อย่างไรก็ตามในภาษาชั้นสูง เช่น FORTRAN ตัวเลขที่เป็นจำนวนจริง เรามักใช้เพียงจำนวน 4 ไบท์เท่านั้น ส่วนตัวเลขที่เป็นจำนวนเต็ม (integer) เราอาจลดลงมาเหลือเพียง 2 ไบท์ (16 บิท)

ในชุดคำสั่งของ 8080 มีคำสั่งที่ทำไว้สำหรับการบวกตัวเลข (BCD) รหัสฐานสิบได้ ลักษณะการบวกจะเป็นไปได้เหมือนเดิมโดยสมมติว่าแต่ละไบท์เป็นของข้อมูลจะเก็บตัวฐานสิบไว้สองหลัก เช่น

```
    36701985 +
    12663459
    --------
    49365444
```

การบวกเช่นนี้กระทำได้โดยไม่ยากเย็นอะไรเลย เราเพียงแต่ใช้วิธีการบวกตามที่เคยใช้มาแล้ว แล้วใช้คำสั่งแทรกประกอบ คือ คำสั่ง DAA ตามหลังคำสั่ง ADC เพื่อปรับผลของการบวกที่ได้ให้เป็นเลขฐานสิบเสียก่อน

การกระทำตามคำสั่ง DAA นั้นมีหลักการที่น่าสนใจอยู่ที่ควรแก่การรู้ คือ ในคำสั่งนี้จะใช้ส่วนของแฟลกตัวทด (auxiliary carry flag) เป็นตัวแก้ไขตัวเลขในคำสั่ง DAA พอสรุปเป็นข้อ ๆ ดังนี้

1. ในกรณีที่ผลรวมของตัวเลขมีค่าเกิน 9 คืออยู่ระหว่าง 10-15 ในกรณีนี้คำสั่ง DAA จะกระทำโดยการบวกค่าตัวเลข 0110 เพื่อแก้ตัวเลขดังกล่าวเช่น

```
    0101        (5)
    1000        (8)
    ----
    1101        (D)
    0110        :
    ----
0001 0011       (BCD = 13)
```

2. ในกรณีที่ผลบวกเลขมีผลลัพท์ได้มากกว่า 16 ในกรณีนี้ค่าที่ปรากฏในตอนแรกจะมีค่าไม่เป็นไปตามข้อ 1 แต่ถ้าการทดปรากฏในแฟลกตัวทดช่วยหรือแฟลกตัวทด คำสั่ง DAA จะกระทำการปรับค่าด้วยการบวกตัวเลข 0110 เข้าด้วย การบวกตัวเลข เช่น

```
    1000        (8)
    1001        (9)
    ----
0001 0001   (BCD = 11)
    0110
    ----
0001 0111   (BCD = 17)
```

การคูณ หาร ตัวเลขไบนารี่

ในไมโครโปรเซสเซอร์บางเบอร์ โดยเฉพาะกลุ่ม 16 บิท จะมีคำสั่งการคูณหรือหารได้โดยเฉพาะกลุ่ม 16 บิทจะมีคำสั่งการคูณ หารได้โดยตรง แต่สำหรับ 8080 แล้วการคูณหรือหารจะไม่มี ดังนั้นหากเราต้องการใช้คำสั่งในการคูณ หรือหารเลขเราก็จำเป็นจะต้องเขียนโปรแกรมขึ้นเพื่อใช้ในการคูณ หาร

อัลกอริทึมในการคูณ

ในการคูณตัวเลขไบนารี่ด้วยคอมพิวเตอร์นั้นก็เหมือนการคูณตัวเลขด้วยมือนั่นเอง เรามีขั้นตอนประกอบอย่างไร เราก็อาศัยขั้นตอนมาใช้ในการคูณตามแบบอย่างเช่น

```
    11011011
    10010011
    11011011
   11011011
  11011011
  11011011
  1111011100001
```

จากผลคูณที่ได้ดังตัวอย่างข้างบนจะเห็นว่าการคูณก็คือการเลื่อนบิทมาทางซ้ายของตัวตั้ง แล้วบวกกันเข้าไปตามตำแหน่งเลขของตัวคูณ ดังนั้นเราสามารถวางเงื่อนไขของการคูณเป็นขั้นตอนต่าง ๆ ดังนี้

ขั้นตอนที่ 1 เซทค่าเริ่มต้น

```
PRODUCT   =  0
COUNTER   =  8
```

ขั้นตอนที่ 2 เลื่อน PRODUCT ให้ตรงพอเหมาะ PRODUCT = 2 x PRODUCT ถ้า lsb = 0

ขั้นตอนที่ 3 เลื่อนตัวคูณ (MULTIPLIER) เพื่อให้บิทเลื่อนสู่แฟลกตัวทด

```
MULTIPLIER  =  2 x MULTIPLIER
```

ขั้นตอนที่ 4 บวกตัวตั้ง (multiplicand) เข้ากับ PRODUCT ถ้าแฟลกตัวทดเท่ากับ 1

ขั้นตอนที่ 5 ลดค่า COUNTER แล้วตรวจสอบว่าเป็นศูนย์ COUNTER = COUNTER - 1

ถ้า COUNTER ไม่เท่ากับ ศูนย์ ให้ไปทำขั้นตอนที่ 2 ใหม่

เพื่อให้เห็นอัลกอริทึมในการคูณได้ชัดเจนขอให้ดูลักษณะการทำการคูณของคอมพิวเตอร์เป็นขั้น ๆ โดยสมมติว่าตัวคูณคือ 61H และตัวตั้งคือ 6FH กรรมวิธีการคูณเป็นดังนี้

ขั้นเริ่มกำหนดค่า

PRODUCT	0000
MULTIPLIER	61
MULTIPLICAND	6F
COUNTER	8

หลังจากการทำครั้งแรกตามขั้นตอนที่ 2-5 ผลที่ได้จะเป็นดังนี้

PRODUCT	0000
MULTIPLIER	C2
MULTIPLICAND	6F
COUNTER	07
CARRY FORM MULTIPLIER	0

หลังจากทำการครั้งที่ 2 ผลที่ได้

PRODUCT	006F
MULTIPLIER	84
MULTIPLICAND	6F
COUNTER	05
CARRY FORM MULTIPLIER	1

หลังจากทำครั้งที่ 3 ผลที่ได้

PRODUCT	014D
MULTIPLIER	20
MULTIPLICAND	6F
COUNTER	03
CARRY FORM MULTIPLIER	0

หลังจากทำครั้งที่ 4

PRODUCT 1	029A
MULTIPLIER	10
MULTIPLICAND	6F
COUNTER	04
CARRY FORM MULTIPLIER	0

หลังจากทำครั้งที่ 5

PRODUCT	0534
MULTIPLIER	20
MULTIPLICAND	6F
COUNTER	03
CARRY FORM MULTIPLIER	0

หลังจากทำครั้งที่ 6

PRODUCT	0A68
MULTIPLIER	40
MULTIPLICOAN	6F

หลังจากทำครั้งที่ 8

PRODUCT	2A0F
MULTIPLIER	00
MULTIPLICAND	6F
COUNTER	00
CARRY FORM MULTIPLIER	1

จากกรณีนี้ถ้าสมมติว่าต้องการคูณตัวเลขที่เก็บไว้ที่แอดเดรส 4016 กับแอดเดรส 4116 ผลลัพท์ที่ได้เก็บไว้ที่แอดเดรส 4216 และ 4316

<u>การตรวจสอบข้อมูลด้วยวิธีการทดสอบผลบวก (CHECK SUM)</u>

ในกรณีป้อนข้อมูลหรือโหลดข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง มักจะประสบปัญหาผิดเพี้ยนของข้อมูล เช่น ตัวเลข 43 ผู้ป้อนหรือป้อนเป็น 34 สลับกันในขณะกดตัวเลขเข้าไป ลักษณะการผิดเพี้ยนเช่นนี้ควรจะมีการตรวจสอบที่ได้ผลดี วิธีหนึ่งที่มีการใช้กัน ข้อมูล 7260 3525 เป็นข้อมูล 2 ตัวที่ต่ออนุกรมกันมา แต่การรับจะกลายเป็น 2603 เป็นตัวเลขตัวหนึ่ง

วิธีการทดสอบแบบนี้อาศัยหลักการบวกค่าตัวเลขเหล่านี้ไม่ได้ ทั้งนี้เพราะจะทำให้เราตรวจสอบการสลับที่ไม่ได้ การตรวจสอบจึงต้องอาศัยเทคนิควิธีการช่วยเพิ่มเติม โดยการหาตัวเลขมาคูณหลักหนึ่งหลักใด วิธีที่ใช้กันมากวิธีหนึ่งก็คือการคูณด้วย 2 ดังตัวอย่าง

ตัวเลข 54931

ผลของการทดสอบผลบวก = 5x2+4+9x2+3+3x2+1

= 40

ค่าของผลการทดสอบบวก = 0

วิธีการนี้เรามักจะเขียนขึ้นในโปรแกรมการโหลดบล๊อกข้อมูลลงเทปคาสเซท โดยแต่ละบล๊อกของข้อมูลตรวจสอบการทดสอบอีกครั้งหนึ่งว่าผลลัพท์ที่ได้ตรงกันหรือไม่ ตรงกันก็แสดงว่าการโหลดข้อมูลนั้นถูกต้อง

จากกรณีนี้เราลองยกตัวอย่างโปรแกรมการตรวจสอบผลบวกโดยข้อสมมติของการตรวจสอบข้อมูล 1 บล๊อกโดยแอดเดรส 30 ผลลัพท์ของการตรวจสอบเก็บไว้ที่ตำแหน่ง 40

ตัวอย่าง (30) = 03

(41) = 36

(42) = 68

(43) = 51

ผลลัพท์ที่ได้ CHECK JUM = 3X2+6+6X2+8+5X2+1

= 43

(40) = 03

จากกรณีเราเขียนผังงานได้ดังรูปที่ 5.19 จะเขียนโปรแกรมได้ดังนี้

```
      LDA   30H        ; COUNT  =  LENGTH OF STRING (IN BYTE)
      MOV   B,A
      MVI   C,0        ; CHECKSUM  =  0
      LXI   H, 41H     ; POINT TO START OF STRING
CHDIG MOV A,M          ; GET TWO BCD DIGITS
      MOV   D,A        ; SAVE COPY
      RAR
      RAR
      RAR
      DCR   B
      JNZ   CHDIG
```

```
        ANI   00001111B     ; MASK OFF SELF CHECK DIGIT
        HLT
```

การาเขียนโปรแกรมที่ยาว ๆ และมีส่วนของโปรแกรมที่ซ้ำ ๆ กันอยู่หลายแห่งอาจยุบส่วนของโปรแกรมย่อยก็คือโปรแกรมที่อาจจะเรียกไปใช้ได้ในขณะที่ซีพียูกำลังประมวลผลในโปรแกรมใดโปรแกรมหนึ่งอยู่ และเมื่อเรียกหาแล้วซีพียูจะกระโดดไปทำโปรแกรมย่อยจนกว่าจะมีคำสั่งหวนกลับ (return) มันจะกลับไปยังโปรแกรมเดิมที่กระโดดมา ลักษณะการทำงานของโปรแกรมย่อยเขียนเป็นไดอะแกรมได้ดังนี้

```
        ANI   0001111B      ; MASK OFF MSD
        ADD   A             ; DOUBLE MSD
        DAA                 ; KEEP IT DECIMAL
        MOV   C,A
        MOV   A,D
        ANI   00001111B     ; MASK OFF LSD
        ADD   C             ; ADD LSD TO CHECKSUM
        DAA
        MOV   C,A
        INX   H
```

จากลักษณะการทำงานข้างบนจะเห็นได้ว่าส่วนของโปรแกรมหลัก (main program) ถ้ามีคำสั่งเรียกชื่อโปรแกรมย่อยมันจะกระโดดไปทำในส่วนย่อย ครั้นทำในโปรแกรมย่อยจบแล้วจะกลับมายังโปรแกรมหลักในคำสั่ง CALL เป็นเช่นนี้

ในการเขียนโปรแกรมย่อยนั้นจะต้องมีเทคนิคที่ต้องคำนึงถึง หลายประการเช่น การคงรักษาข้อมูลบางส่วนที่ได้กระทำไว้ก่อนการกระโดดไปทำในส่วนโปรแกรมย่อย เช่น ข้อมูลในรีจิสเตอร์ต่าง ๆ (A, B,C,D,E,H,L) อาจจะต้องเก็บไว้ในที่ใดที่หนึ่งก่อน แล้วก่อนการ RET จึงจะนำเอาข้อมูลเหล่านี้กลับมาใส่ให้ใหม่

กรรมวิธีการเรียกโปรแกรมย่อยเราใช้คำสั่ง CALL ผลที่ได้คือ ซีพียูจะนำข้อมูลที่อยู่ในโปรแกรมย่อยไปให้กับโปรแกรมเคาน์เตอร์เพื่อให้เครื่องวิ่งต่อไปได้

ลองมาพิจารณาตัวอย่างโปรแกรมย่อยและโปรแกรมหลักที่สัมพันธ์กับโปรแกรมย่อย ในการนับความยาวของสตริงดังนี้

```
        LXI  SP,80          ; START STACK AT LOCATION 80
        LHLD  40 H          ; GET STARTING ADDRESS
        CALL STLEN          ; DETERMINE STRING LENGTH
        STA 42 H.           ; STORE STRING LENGTH
```

ส่วนของโปรแกรมย่อยเขียนได้

```
STLEN  MVI  B,0             ; LENGTH  =  0
       MVI  A, ODH          ; GET CR FOR COMPARISION
CHKC   CMP  M               ; IS CHARACTER 'CR'
       JZ   DONE            ; YES END OF STRING
       INR  B               ; NO ADD 1 TO LENGTH
       INX  H
       JMP  CHKC
KDONE  MOV  A,B
       RET
```

ปัญหาที่น่าสนใจอย่างหนึ่งในการใช้โปรแกรมย่อย คือ เรื่องเกี่ยวกับสแตคพอยท์เตอร์ ทั้งนี้เพราะเมื่อเราใช้คำสั่ง CALL ข้อมูลในโปรแกรมเคาน์เตอร์จะถูกถ่ายไปเก็บไว้ในหน่วยความจำที่แสตคอ้างไว้ และจะนำค่าให้ข้อมูลในแสตคพอยท์เตอร์เองผิดไปจากเดิมนั่นคือการเรียกโปรแกรมย่อยทุกครั้งจะเกี่ยวข้องกับแสตคพอยท์เตอร์

ในส่วนโปรแกรมควบคุมระบบการทำงาน (Monitor) มักเก็บไว้ใน Rom โดยส่วนของแสตคพอยท์จะอยู่ในหน่วยความจำ RAM และการเข้าถึงแสตคเหล่านี้จะต้องเกี่ยวข้องกับแอดเดรสในแสตคมากระทำ ในส่วนโปรแกรมที่ผู้ใช้เขียนขึ้นส่วนของโปรแกรมที่ผู้ใช้เขียนอาจจะมีการเกี่ยวข้องโดยตรงกับแสตคหรือ RAM ในส่วนแสตคทำให้เมื่อกระโดกลับเข้ามอนิเตอร์ใหม่จะทำให้การทำงานมอนิเตอร์จึงต้องมีการเก็บข้อมูลในแสตคพอยท์เตอร์ไว้เสียก่อน โดยใช้ส่วนโปรแกรมต่อไปนี้

```
        LXI  H,0            ; GET MONITOR STACK POINTER
        DAD  SP
        SHLD STEMP          ; AND SAVE IT
```

เมื่อเสร็จจากการเก็บข้อมูลในแสตคพอยท์เตอร์ไว้แล้วจึงเรียกไปใช้กระทำอย่างอื่น ซึ่งอาจจะมีการเซทค่าแสตคพอยท์เตอร์ใหม่อีกครั้ง เมื่อต้องการจะกลับเข้าไปในมอนิเตอร์โปรแกรมจำเป็นจะต้องเรียกค่ากลับเข้ามายังแสตคพอยท์เตอร์ใหม่ด้วยชุดคำสั่ง

```
LHLD    STEMP          ; RESTORE MONITOR STACK POINTER
SPHL
```

<u>การเขียนโปรแกรมที่เกี่ยวข้องกับตัวอุปกรณ์อินพุทและเอาท์พุท</u>

ก่อนอื่นมาทำความเข้าใจกับโครงสร้างการส่งเสริมการส่งข้อมูลเบื้องต้น ระหว่างซีพียูกับอุปกรณ์ I/O อีกครั้ง อุปกรณ์ I/O ที่ซีพียูจะมองเห็นเหมือนแอดเดรสหนึ่ง โดยความเร็วในการส่งข้อมูลก็เหมือนกับการส่งข้อมูลกับ RAM และการใช้โครงสร้างระดับแรงดันและตัวเลขไบนารี่เหมือนกัน เพียงแต่ว่าการกำหนด I/O ใน 8080 นั้นสามารถกำหนดตัว I/O ได้ถึง 28 = 256 ตัว และเอาท์พุทถึง 256 คำสั่ง ว่าซีพียูจะกระทำกับตัว I/O ใน 8080 ก็มีเพียง PORT และ OUT PORT เท่านั้น แต่อย่างไรก็ตามความเร็วในการส่งจากซีพียูไปยังอุปกรณ์เอาท์พุท หรือรับจากอินพุทมายังซีพียูด้วยความเร็วฟลูซึ่งขึ้นอยู่กับสัญญาณนาฬิกาที่เราใช้

ในการเชื่อมต่อกับอุปกรณ์ I/O นั้นลักษณะของการเชื่อมมักขึ้นอยู่กับความเร็วในการส่งข้อมูลของตัวอุปกรณ์ I/O ด้วย อัตราการส่งข้อมูลแบ่งออกเป็น 3 ชนิด คือ

1. ชนิดความเร็วช้า ได้แก่ I/O ที่จะส่งข้อมูล หรือส่งข้อมูลมาใช้ด้วยเวลาที่ค่อนข้างช้ามาก เช่น อุปกรณ์จำพวกรีเลย์ และตัวรับความรู้สึกทางกลต่าง ๆ และเราอาจรวมถึงตัวภาคแสดงด้วยแสงบางแบบ

2. ชนิดความเร็วปานกลาง ซึ่งเราอาจกำหนดขนาดความเร็วอยู่ในช่วงประมาณ 1- 10000 บิทต่อวินาที อุปกรณ์จำพวกคีย์บอร์ดได้แก่ เครื่องพิมพ์ เครื่องอ่านการ์ด เทปคาสเซท อุปกรณ์รับส่งข้อมูลทางอนาล็อก เป็นต้น

3. ชนิดความเร็วสูง พวกนี้ส่งข้อมูลได้เร็วกว่า 10000 บิทต่อวินาที อุปกรณ์พวกนี้ได้แก่ CRT จานแม่เหล็ก เทปแม่เหล็ก เครื่องพิมพ์ความเร็วสูง

สำหรับในกรณีของไมโครคอมพิวเตอร์ที่เราให้ความสนใจพิเศษ และเป็นหลักการทางเบื้องต้นนั้น เราจะสนใจใน 2 กลุ่มนี้ก่อน

ในการรับส่งอุปกรณ์ประเภทมีอัตราการส่งข้อมูลช้า ๆ อาจจะต้องมีการกำหนดเวลาหรือควบคุมสัญญาณจังหวะเวลากันบ้าง เช่น ส่งหรือรับข้อมูลจากเทปคาสเซทด้วยอัตราขนาด 300 บิทต่อวินาที ดังนั้นอต่ละบิทจะมีเวลาห่างกัน 3.33 มิลลิวินาที ถ้ามากกว่าส่งมา 8 บิท ได้ในคราวเดียวกันก็จะใช้เวลาส่งห่างกัน 8 x 3.33 = 26.64 มิลลิวินาที ซึ่งก็ใช้เวลานานพอสมควรเมื่อเทียบกับการทำงาน

ของซีพียูที่ทำงานในหน่วยของไมโครวินาที วิธีหนึ่งที่ใช้ในการหน่วงเวลานี้คือ การกำหนดช่วงเวลาด้วยซอร์ฟแวร์ เราสามารถเขียนโปรแกรมให้เกิดการดีเลย์ตามที่กำหนดได้ โดยอาศัยผังงานข้างล่างนี้

ค่าของช่วงเวลาที่ดีเลย์ในโปรแกรมนี้ขึ้นอยู่กับการเซท

MSCNT โดยโปรแกรมเขียนได้ดังนี้

```
DELAYMVI B, MSCN    ; GET COUNT FOR 1 MS
DLY 1 DCR B         ; COUNT = COUNT -1
JNZ DLY 1           ; CONTINUE UNTIL COUNT = 0
DCR A               ; UNMBER OF MS = NUMBER OF MS - 1
```

การคำนวณเวลาจากโปรแกรมทำได้ไม่ยากโดยการสร้างตารางเวลา โดยดูว่าในแต่ละคำสั่งใช้เวลาจำนวนกี่ไซเคิลของสัญญาณนาฬิกา เราจะได้ตารางคำนวณดังนี้

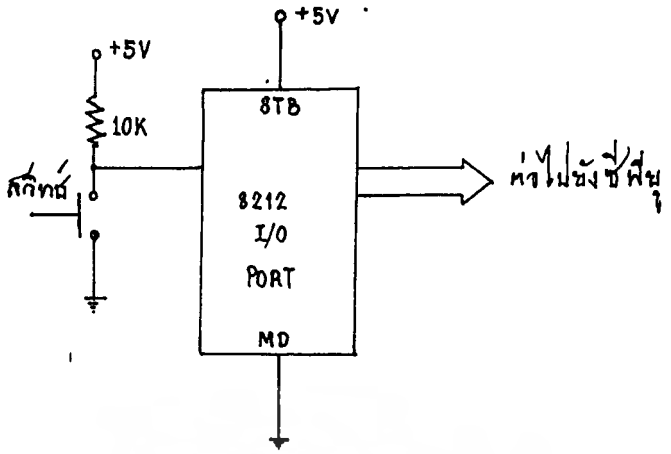
คำสั่ง		หน่วย
MVI B, MSCNT	7 X (A)	จำนวนไซเคิลของสัญญาณนาฬิกา
DCR	5 X (A) X MSCNT	"
JNZ	10 X (A) X MSCNT	"
DCR	5 X (A)	
JNZ	10 X (A)	"

การอ่านข้อมูลจากสวิทช์

การใช้อุปกรณ์อินพุทแบบพื้นฐานอย่างหนึ่งคือ การใช้สวิทช์ซึ่งแต่ละตัวสามารถเป็นอุปกรณ์อินพุทได้ 1 บิท ถ้าเราจะรวมสวิทช์ให้เป็นตัวสามารถเป็นอุปกรณ์ต้องใช้สวิทช์ 8 ตัว ลักษณะของสวิทช์ที่ต่อเป็นตัวอินพุทเมื่ออินเตอร์เฟสเข้ากับซีพียูแบบง่าย แสดให้เห็นดังรูป 5.22

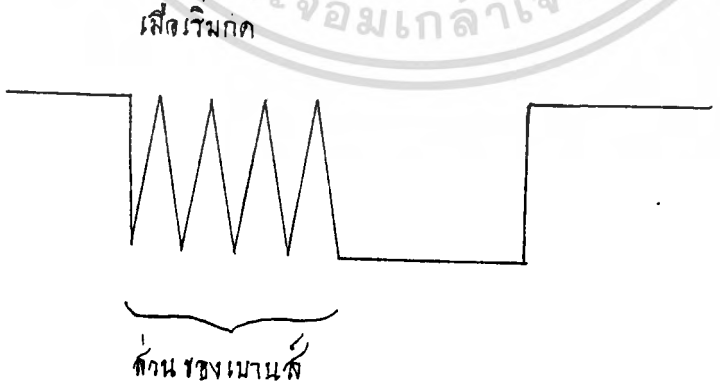
เรามีวิธีการตรวจสอบว่าสวิทช์ได้รับการปิดวงจรหรือยังด้วยคำสั่งง่าย ๆ ที่ขึ้นอยู่กับว่าสวิทช์จะต่ออยู่ที่บิทใดของบัสข้อมูล ในที่นี้สมมติว่าต่อที่บิทที่ 7 คำสั่งที่ใช้ก็คือ

```
IN   PORT     ; READ BUTTON POSSITION
RAL           ; IS BUTTON CLOSED
JNC  DONE     ; YES, DONE
```



รูปที่ 5.22

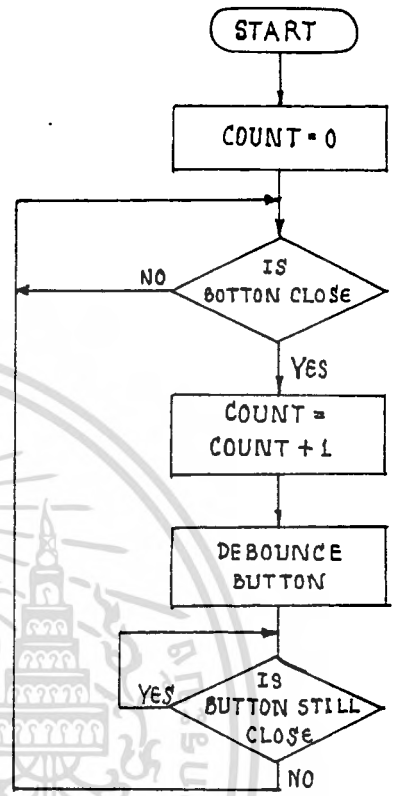
และถ้าเราต่อที่บิท 0 เราก็เพียงเปลี่ยนคำสั่ง RAL เท่านั้น ในการกดสวิทช์นั้นโดยทั่วไปมักจะต้องเกิดพัลช์ที่เป็นเบาน์ส ลักษณะของเบาน์สนี้จะมีผลต่อการอ่านของซีพียูได้ ทั้งนี้เพราะช่วงเวลาการอ่านนั้นถ้าอ่านของซีพียูได้ ข้อมูลที่ได้ก็อาจจะเป็น 0 หรือ 1 ก็ได้ไม่แน่นอน ดังนั้นจึงต้องมีการกำจัดส่วนของเบาน์ส การกำจัดส่วนของเบาน์สนี้อาจทำได้ทั้งฮาร์ดแวร์และซอฟท์แวร์ แต่วิธีที่จะเป็นที่นิยมมากวิธีหนึ่งคือวิธีกำจัดเบาน์สด้วยวิธีทางซอฟท์แวร์ เพราะไม่ต้องลงทุนเพิ่มเติม ทางฮาร์ดแวร์ลองมาพิจารณาการกำจัดเบาน์สจากตัวอย่างนี้



รูปที่ 5.23

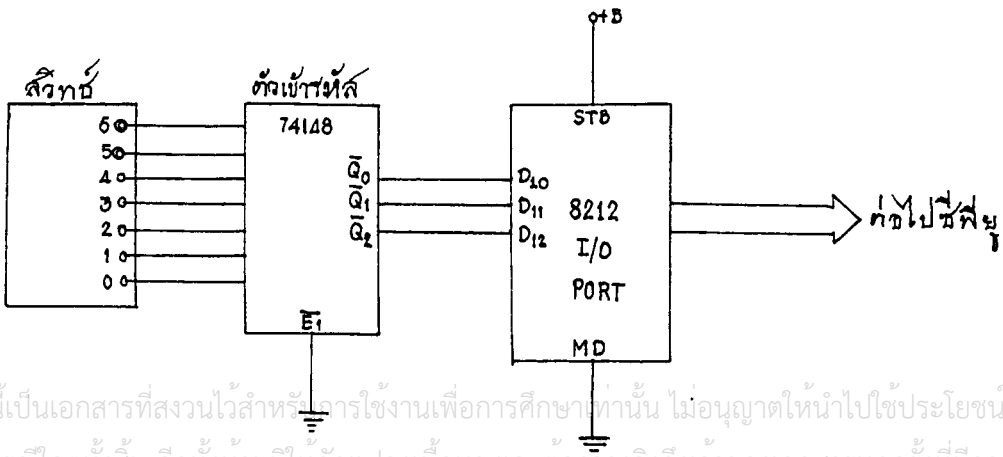
การเขียนโปรแกรมจึงต้องคำนึงถึงส่วนเหล่านี้ขอให้พิจารณาจากผังงาน และเราเขียนโปรแกรมวงจรการนับการกดของสวิทช์ได้ดังนี้

```
      LXI   H,40      ; (40) = COUNT = 0
      MVI   M,0
CHKCL IN PORT         ; IS BUTTON CLOSED (0)
     .ANI   MASK
      JNZ   CHKCT     ; NO WAIT
      INR   M         ; YES, COUNT = COUNT + 1
      MVI   A,1
      CALLDELAY       ; DEBOUNCE BUTTON BY
      -                 WAITING IM
CHKOP - IN PORT       ; ISBUTTON STILL CLOSED (0)
      ANI   MASK      ; YES WAIT
      JZ    CHKOP     ; NOLOOK FOR NEXT CLOSERE
      JMP CHKCL
```



จากโปรแกรมที่กล่าวมานี้เราจะอ่านการกดของสวิทช์ครั้งแรกแล้วรอเวลาอีกครั้งหนึ่ง ให้ผ่านช่วงเบานส์เสียก่อนแล้วค่อยอ่านอีกครั้ง วิธีการนี้เป็นเทคนิคในการอ่านดีเบานส์เสียก่อน แล้วค่อยอ่านอีกครั้ง วิธีการนี้เป็นเทคนิคในการดีเบานส์ที่ใช้กันทั่วไป (โดยทั่วไปช่วงเวลาดีเลย์จะใช้ประมาณ 10 มิลลิวินาที)

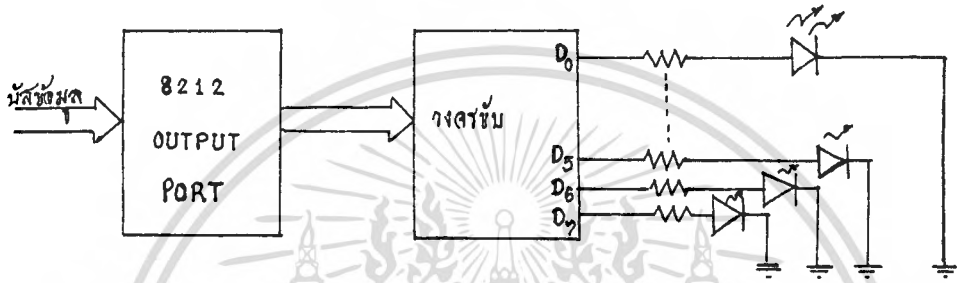
นอกจากนี้เราสามารถดัดแปลงการอ่านสวิทช์เพียงตัวเดียวเหล่านี้มาเป็นลักษณะของสวิทช์เลือกหรือสวิทช์หมุน หรือทัมบ์วิลด์สวิทช์ ด้วยการอ่านรูปลักษณะเช่นวงจรข้างล่าง



การเขียนโปรแกรมอินเตอร์เฟสเข้ากับเอาท์พุทภาคแสดง LED

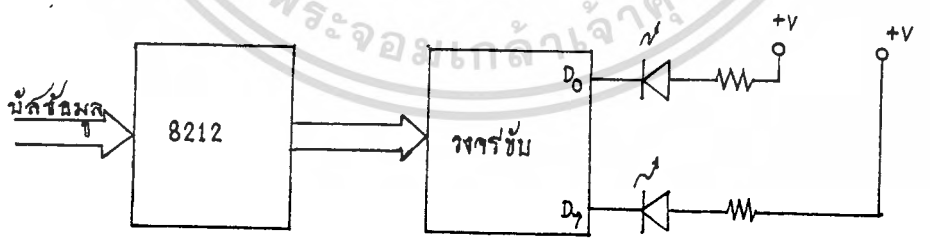
การติดสว่างของ LED จะแสดงสถานะทางลอจิกได้โดยอาศัยการใช้ LED เป็นภาคแสดงผลของไมโครคอมพิวเตอร์ได้อย่างดี การแสดงผลนั้นอาจจะออกมาในรูปของรหัสเลขฐาน 8 หรือฐานสิบหกเลยก็ได้ ขึ้นกับวิธีการถอดรหัสภาคแสดงเหล่านั้น

ลักษณะการอินเตอร์เฟสในภาคเอาท์พุท เมื่อแสดงรหัสไบนารี่จะใช้ไดอะแกรมของวงจรดังตัวอย่าง



รูป 5.26

ก. การให้ LED สว่างตามสถานะลอจิก 1



รูป 5.27

ข. การให้ LED สว่างตามสถานะลอจิก 0

การส่งข้อมูลให้ LED สว่างนั้นก็ไม่ยากเย็นอะไรเลย โดยใช้คำสั่งลักษณะนี้ คือ

```
MVI A,  MASKP
OUT PORT
```

โดยปกติที่ตัวเอาท์พุทจะมีวงจรในลักษณะที่จะต้องแสดงแลทช์ข้อมูลเอาไว้ได้ ดังนั้นเมื่อส่งข้อมูลก็จะสว่างที่ LED ครั้นส่งมาใหม่ ตัว LED ก็จะติดสว่างตามข้อมูลชุดใหม่

การส่งข้อมูลมานั้นบางครั้งมาจากข้อมูลในแอดเดรสใดแอดเดรสหนึ่ง ลักษณะการเปลี่ยนข้อมูลโดยโหลดข้อมูลมาจากที่เก็บที่อยู่ในหน่วยความจำทำได้โดย

```
LXI H, ADDRRE   ; LOAD PONTER TP DATA
MOV A,M         ; GET COPY OF DATA
OUT PORT        ; DATA TO LED
```

จากลักษณะดังกล่าวมานี้เป็นลักษณะของการส่งข้อมูลเข้าแสดงผลด้วย LED ลักษณะตัวเลข 0 และ 1 แต่ถ้าหากต้องเอาท์พุทแสดงผลด้วยตัวแสดง LED 7 ส่วน ที่สามารถใช้ของตัวเลขและถ้าหากไม่ใช่ตำแหน่งจุดทศนิยมก็จะใช้ข้อมูลเพียง 7 บิทเท่านั้น วิธีการที่จะให้ภาคแสดงจะอินเตอร์เฟสในลักษณะที่จะกล่าวนี้ได้วิธีหนึ่ง

จากวิธีนี้เราจะใช้วิธีการสร้างตารางของภาคแสดงเอาท์พุทดังที่ได้กล่าวมาแล้ว ในที่นี้จะกล่าวซ้ำอีกเล็กน้อยในแง่ของภาคแสดงที่สัมพันธ์กับรหัสในตารางได้ดังนี้

ตัวเลข	เลขฐานสิบหก	
	คาโถด	อาโนด
0	3F	40
1	06	79
2	5B	24
3	4F	30
4	66	19
5	6D	12
6	7D	02
7	07	78
8	7F	00
9	67	18
A	77	08
B	7C	03
C	39	46
D	5E	21
E	79	06
F	71	0E

ลองมาดูตัวอย่างการเขียนโปรแกรมให้แสดงผลต่อภาคแสดงในรูปตัวเลขฐานสิบหก โดยนำข้อมูลในหน่วยความจำมาแสดง ถ้าข้อมูลเก็บตัวเลขเกินกว่า 0FH ให้แสดงดับหมด ผังงานจะได้ดังนี้

เราเขียนโปรแกรมตามผังงานได้ดังนี้

```
MVI   B, BLANK          ; GET BLANK CODE
LAD   ADDR              ; GET DATA
CPI   10 H              ; IS DATA 10 H
JNC   D, SPLY           ; YES, DISPLAY BLANKS
```

```
LXI   D, SSEG          ; BASE OF 7-SEGMEN TABLE
MVI   H, 0
KMOV     L, A          ; MAKE DATA INTO 16 BIT
DAD   D                ; INDEX TABLE
MOV   B, M             ; GET 7 SEGMENT CODE
OUT   PORT             ; CODE TO DISPLAY
```

จากกรณีดังกล่าวนี้เราจะส่งรหัสไปได้เพียงตัวเดียว แต่ถ้าหากต้องการส่งรหัสไปยังภาคแสดง 7 ส่วนหลาย ๆ ตัว อาจทำได้ในลักษณะการมัลติเพล็กซ์ คือส่งไปทีละตัวเรียงกันไปโดยอาจจะถือว่า LED 7 ส่วนแต่ละส่วนคือเอาท์พุท 1 ช่องก็ได้

<u>การเขียนโปรแกรมทำการอ่านคีย์บอร์ดชนิดแมทริกซ์</u>

การใช้งานจริง ๆ ของไมโครคอมพิวเตอร์นั้นลักษณะการอ่านข้อมูลอินพุทจะมีเทคนิคและวิธีการอีกมากมาย เช่น อินพุทที่เป็นคีย์บอร์ดที่มีจำนวนมากมาย ดังตัวอย่างของคีย์บอร์ด ของ CRT หรือ TTY ที่มีจำนวนคีย์ได้มากกว่า 64 คีย์ การอ่านคีย์จึงมีวิธีการเพื่อให้การอ่านเป็นไปอย่างมีประสิทธิภาพ และรวดเร็วยิ่งขึ้น ในที่นี้จะยกตัวอย่างการเขียนโปรแกรมเพื่ออ่านข้อมูลจากคีย์บอร์ดชนิดแมททริกซ์

จะเห็นได้ชัดอย่างหนึ่งว่าถ้าหากจัดทางคีย์บอร์ดแบบขนาน เรียงกันตามข้อมูลที่อ่านจะใช้พอร์ทเอาท์พุทมากกว่าหนึ่งพอร์ทในการอ่านข้อมูล และยิ่งถ้าจำนวนคีย์บอร์ดยิ่งมากก็ต้องใช้จำนวนพอร์ทอินพุทมากยิ่งขึ้น ลักษณะเช่นนี้อาจลดจำนวนพอร์ทอินพุทลงได้มากด้วยเทคนิควิธีที่เรียกว่า "การสแกน"

ขนาดของคีย์บอร์ด	จำนวนสายถ้าวางขนานกัน	จำนวนสายถ้าวางแบบแมททริกซ์
3 x 3	9	6
4 x 4	16	8
4 x 6	24	10
5 x 5	25	10
6 x 6	30	12
6 x 8	48	14
8 x 8	64	16

สิ่งที่นักโปรแกรมจะต้องทำการค้นหาว่ามีคีย์ใดได้รับการกดลงมา การค้นหาทำได้ด้วยการสแกนทีละแถว เริ่มที่แถว 0 ก่อนแล้วตรวจแต่ละคอลัมน์ดูถ้าไม่มีก็เลื่อนมาที่แถวที่ 1 แล้วตรวจแต่ละคอลัมน์ทำเช่นนี้จนครบทุกแถวก็จะพบว่าคีย์ใดได้รับการกด ถ้าหากว่าไม่มีการกดในคีย์ใดเลยก็ให้วนตรวจใหม่ ขอให้พิจารณารูป 5.31

จากกรณีนี้เราจะส่งสัญญาณจากเอาท์พุทแลทช์ไว้ก่อน แล้วตรวจสอบด้านการอ่านข้อมูลทางด้านอินพุทว่ามีคีย์ใดที่ลัดวงจรต่อถึงกัน ในกรณีนี้สมมติให้ซีพียูส่งสัญญาณมากราวนด์ทางด้านแถวไว้โดยการสแกนทีละแถวแล้วอ่านข้อมูลทางด้านอินพุทพอร์ท

ลองดูตัวอย่างการอ่านว่าการกดคีย์เกิดขึ้น เราจะใช้วิธีการตรวจสอบได้ดังนี้

```
        OUT KBDOT       ; GROUND ALL KEYBOARD ROWS
        IN  KBIN        ; GET KEYBOARD COLUMN DATA
        ANI 00000111B   ; MASK COLUMN BITS
        CPI 00000111B   ; ARE ANY COLUMN GROUND
        JZ  WAITK       ; NO KEEP LOOKING AT KEYBOARD
DONE    JMP     DONE
```

จากโปรแกรมจะเห็นว่าเราส่งข้อมูลไปยังแต่ละแถวด้วยลอจิก "0" ก่อนแล้วอ่านข้อมูลมาทางอินพุทตรวจสอบดูว่ามีคอลัมน์ใดอ่านได้ลอจิก "0" หรือไม่อ่านก็แสดงว่ามีการกดคีย์ใดคีย์หนึ่ง

การวางรูปและการเขียนโปรแกรมสำหรับมอนิเตอร์

โดยปกติการทำงานในระบบของคอมพิวเตอร์จะต้องอยู่ภายใต้การควบคุมของโปรแกรม ถ้าหากว่าเป็นเครื่องคอมพิวเตอร์ขนาดใหญ่ โปรแกรมที่ใช้ควบคู่ระบบก็เรียกว่า OS (OPERATING SYSTEM) สำหรับไมโครคอมพิวเตอร์ขนาดเล็ก เช่น ไมโครคอมพิวเตอร์แผงเดี่ยว โปรแกรมควบคุมการจัดระบบจะเรียกว่า มอนิเตอร์โปรแกรม

โดยปกติขีดความสามารถของไมโครคอมพิวเตอร์ขนาดเล็กจะต้องประกอบด้วย ฮาร์ดแวร์ และซอฟท์แวร์ที่ดี สำหรับซอฟแวร์นั้นก็ขึ้นอยู่กับโปรแกรมมอนิเตอร์ที่ผู้เขียนจะเพิ่มฟังก์ชั่น หรือขีดความสามารถต่าง ๆ ได้เพียงไร

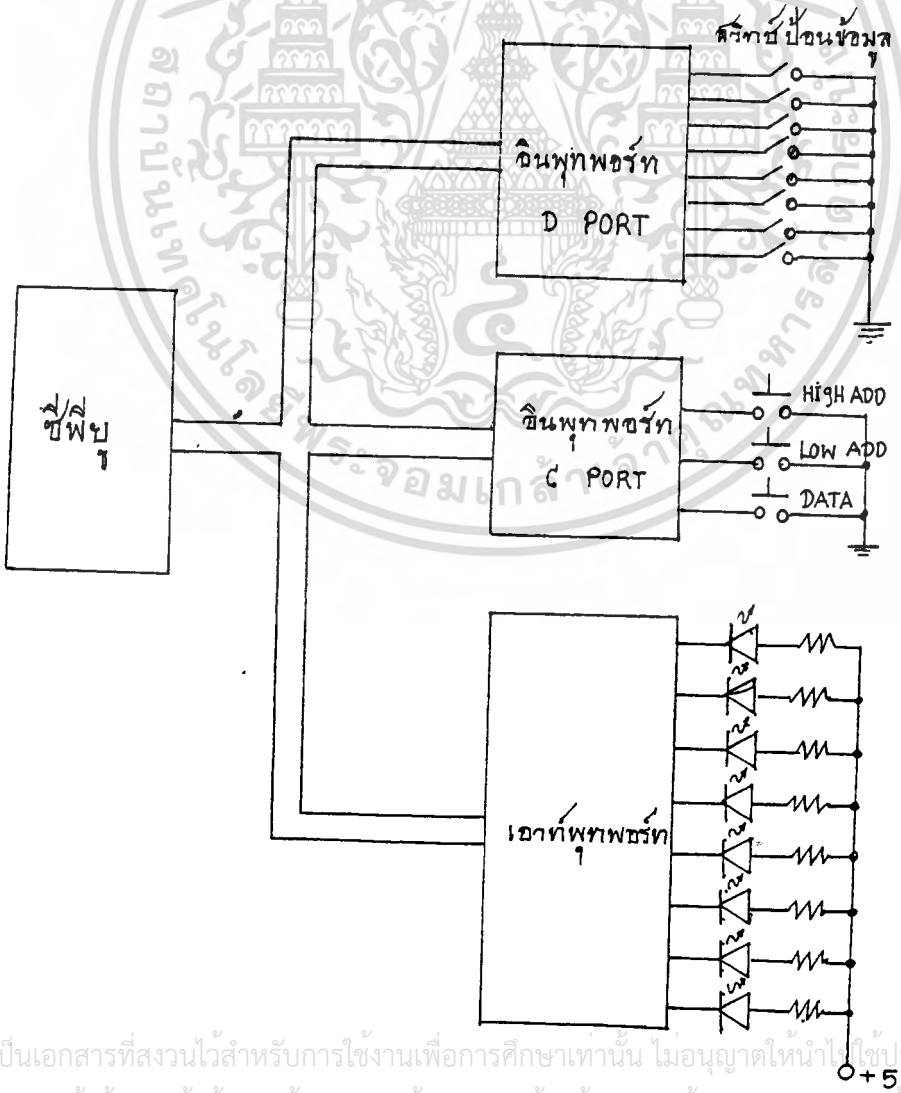
เมื่อเราเปิดสวิทช์ป้อนไฟเลี้ยงให้กับเครื่องไมโครคอมพิวเตอร์จะเห็นว่าตัววงจรทำการรีเซทตัวเองในขณะเปิดไฟ หรือถ้าไม่รีเซทตัวเองก็มักจะต้องมีสวิทช์เพื่อทำการรีเซทก่อนการใช้งาน การรีเซทโดยทั่วไปก็คือการทำให้ค่าใน PC มีค่าเป็น 0 ก่อนแล้วให้เครื่องคอมพิวเตอร์เริ่มทำตามโปรแกรมตั้งแต่

แอดเดรส 0 ดังนั้นที่ตำแหน่งแอดเดรส 0 จึงต้องมีคำสั่งให้เครื่องได้ทำงาน คำสั่งในตอนแรกนี้อาจจะเป็นคำสั่งที่ทำให้คอมพิวเตอร์กระโดดไปที่แอดเดรสอื่น ๆ ได้อีกมากมาย ครั้นเมื่อเครื่องเริ่มวิ่งโปรแกรมจะเข้าสู่โปรแกรมที่เรียกว่า มอนิเตอร์

ลักษณะของโปรแกรมมอนิเตอร์ก็มักจะจัดให้เครื่องทำงาน เช่น จัดให้อ่านข้อมูลที่คีย์มาจากคีย์บอร์ด เมื่ออ่านข้อมูลแล้วก็จะตีความหมายของคีย์บอร์ดนั้นว่าเป็นแอดเดรส เป็นข้อมูล โดยโปรแกรมมอนิเตอร์จะเอาส่วนของข้อมูลนั้นมาตีความหมายและกระทำตามอีกครั้ง

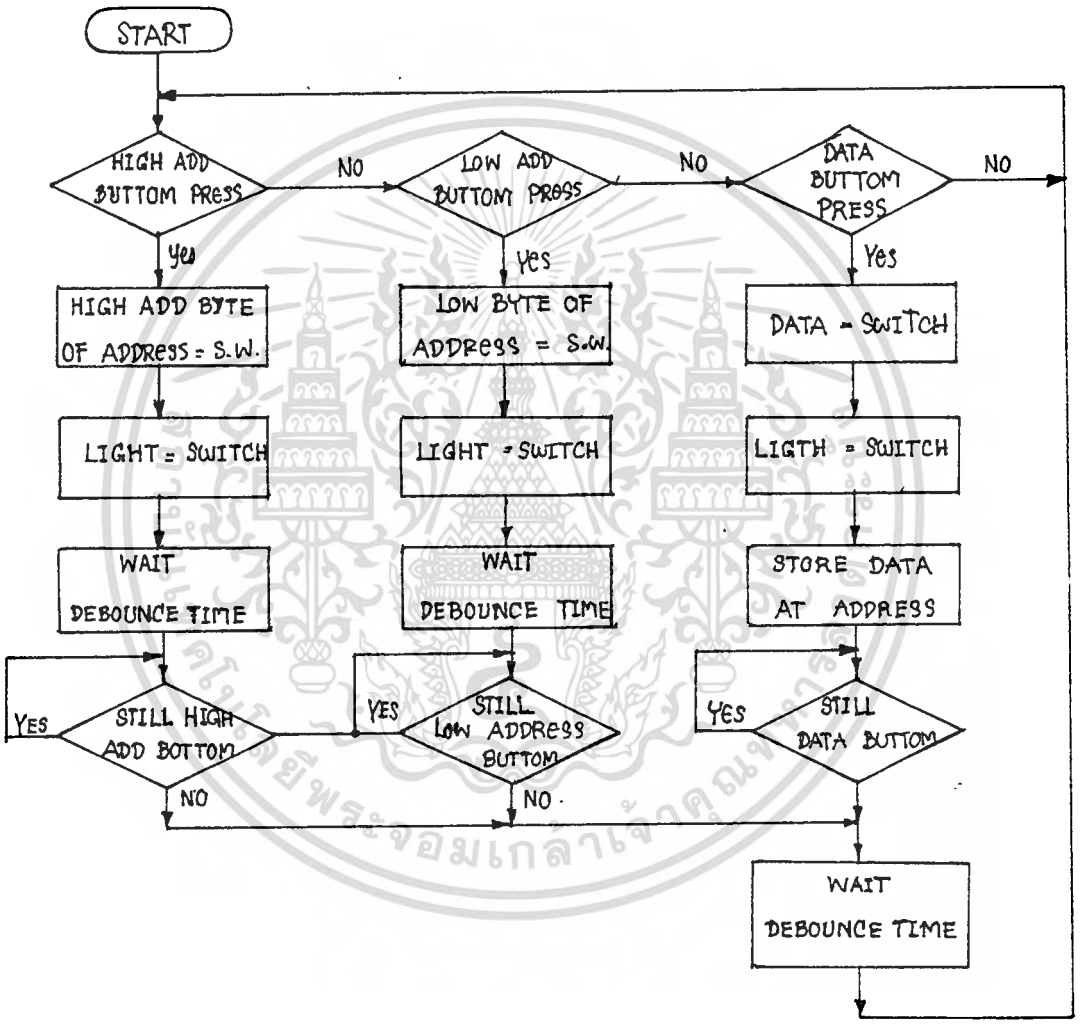
เพื่อให้เข้าใจหลักเบื้องต้นของมอนิเตอร์ จึงขอยกตัวอย่างของวงจรพื้นฐานอย่างง่าย ๆ ของไมโครโปรเซสเซอร์ในการใช้ทำเป็นเครื่องไมโครคอมพิวเตอร์แผงเดียว

รูป 5.35



ระบบนี้เป็นระบบง่าย ๆ ที่ใช้ป้อนข้อมูลจากสวิทช์ไปเก็บยังหน่วยความจำใน 1 ตำแหน่งที่ต้องการ โดยที่เราสามารถเซทค่าสวิทช์ที่ส่วนของสวิทช์ป้อนข้อมูลได้ทั้งเป็นค่าแอดเดรส หรือข้อมูลซึ่งขึ้นอยู่กับการกดสวิทช์คำสั่งควบคุม ผังงานของมอนิเตอร์ที่จะใช้เป็นตัวป้อนข้อมูลได้ดังนี้

รูป 5.35

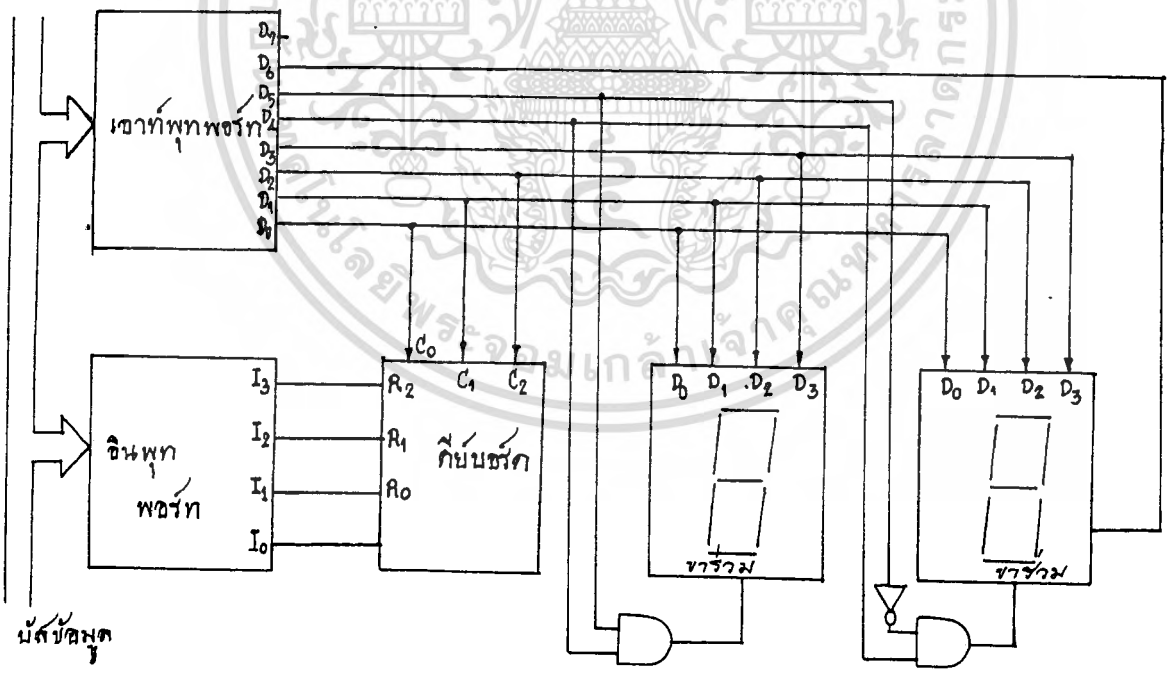


ตัวอย่างการใช้โปรแกรมการประยุกต์แทนวงจรทางฮาร์ดแวร์

ลักษณะของตัวอย่างที่จะยกขึ้นมากล่าวในที่นี้เป็นลักษณะของนาฬิกาจับเวลาที่เราสามารถโปรแกรมตัวเลขได้ 2 ตัวเลข ซึ่งเป็นหลักนาทีและหลักสิบของนาที การโปรแกรมนั้นเราจะโปรแกรมผ่านทางคีย์บอร์ดเมื่อป้อนตัวเลขครบสองตัวแล้วก็จะกดคีย์ GO วงจรจะแสดงการนับแบบถอยหลังโดยภาคแสดงผลจะค่อย ๆ ลดค่าลงที่หนึ่งจนมาเป็น 0 แล้วจะหยุด

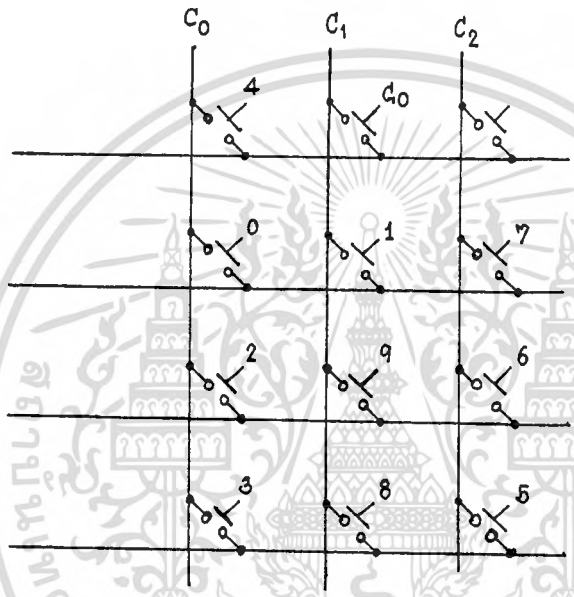
ในการที่จะโปรแกรมการทำงานให้กับวงจรนั้นเราต้องศึกษาและวางรูปโปรแกรมให้กับเครื่องก่อน ในกรณีนี้ถ้าเราให้การคีย์ตัวเลขเครื่องก็จะนำตัวเลขที่คีย์ไปแสดงบนแผง ครั้นเมื่อตัวเลปรากฏครบสองตัวแล้ว เราคีย์ GO วงจรก็จะเริ่มการเป็นนาฬิกาจับเวลาทันที โดยจะค่อย ๆ ลดค่าตัวเลขตามช่วงเวลาที่ตั้งไว้จนกระทั่งถึง 0 ผังงานของวงจรเขียนได้ดังนี้

รูป 5.37



เพื่อให้การอ่านโปรแกรมที่เขียนไว้ได้ง่ายเข้า ในที่นี้ขออธิบายลำดับขั้นตอนของโปรแกรมพร้อมทั้งเปรียบเทียบกับผังงานที่กล่าวมาแล้ว โดยโปรแกรมที่เขียนขึ้นในแบบฉบับมาตรฐานของภาษาแอสเซมบลี้

รูป 5.38

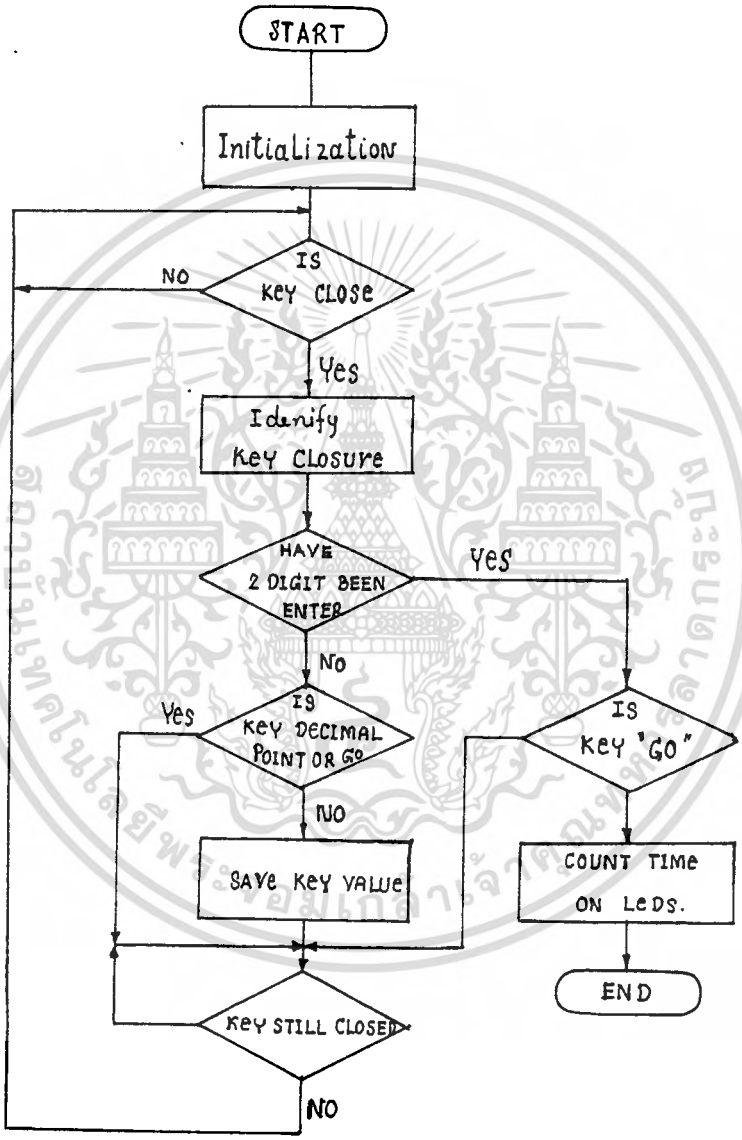


สำหรับโปรแกรมที่เขียนขึ้นไม่จำเป็นต้องเริ่มที่ 0 จึงต้องมีคำสั่งกระโดดไปที่ตำแหน่งที่เริ่มต้นของโปรแกรม โดยปกติการกำหนดค่าเริ่มต้นจะประกอบด้วยลำดับขั้นใหญ่ ๆ 3 ขั้น

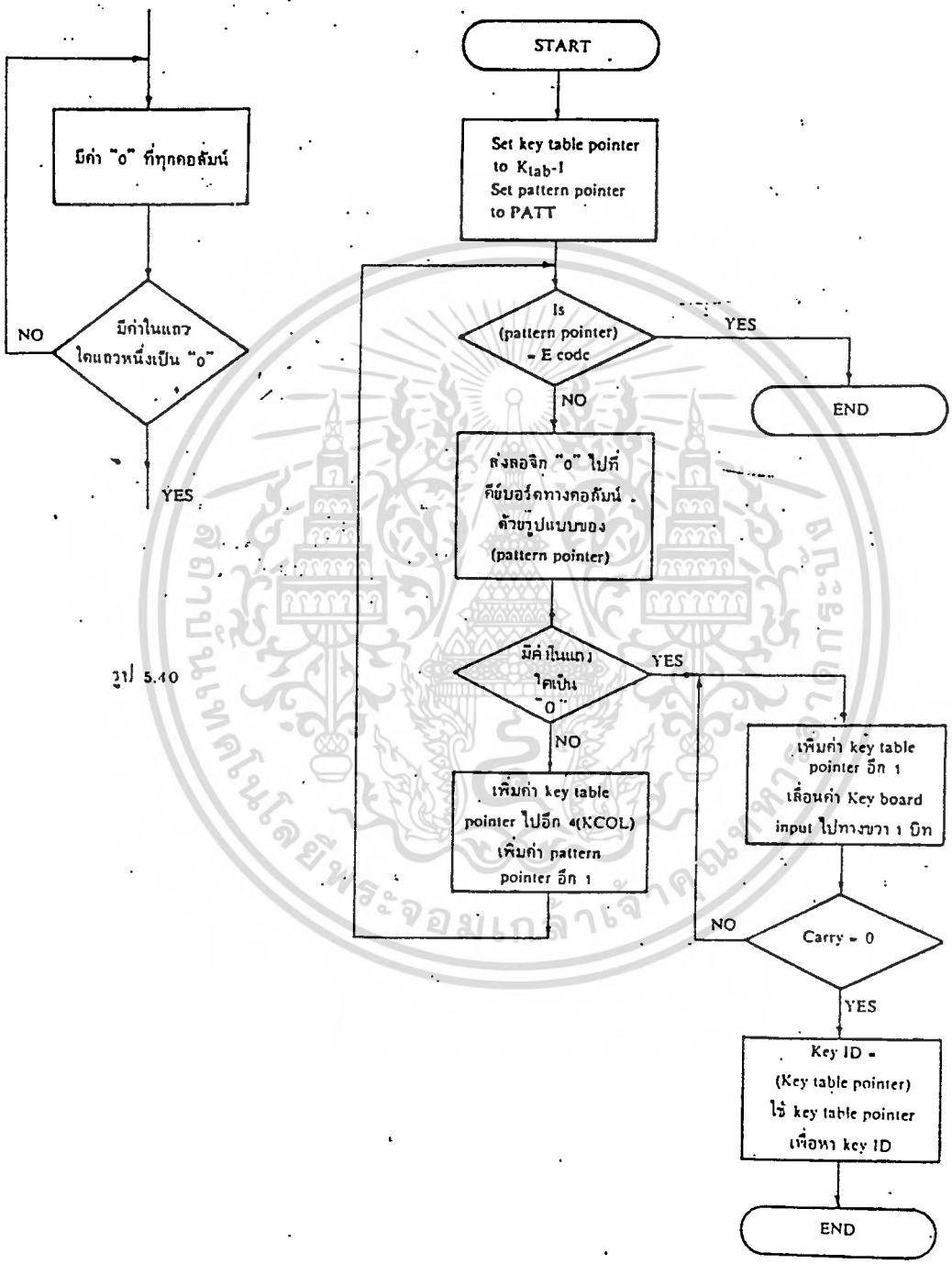
ก. กำหนดแอดเดรสให้กับสแตคว่าจะให้อยู่ที่แอดเดรสใด โดยการโหลดค่าข้อมูลแอดเดรสมาไว้ที่สแตคพอยท์เตอร์

ข. กำหนดค่าเริ่มต้นที่จำเป็นเช่น ค่าที่ใช้ในการตรวจสอบที่กดคือค่า number of digit keys pressed (NKEY) ให้เป็น 0

ค. กำหนดค่าของ KEYAD ซึ่งค่านี้จะเป็นตัวกำหนดแอดเดรสใน RAM และที่ตำแหน่งแอดเดรส KEYAD นี้จะเก็บค่าตัวเลขที่จะได้รับการคีย์ถัดไป และเมื่อโปรแกรมรับค่าที่คีย์มันจะเพิ่มค่า KEYAD ขึ้นไปอีกหนึ่งทุกครั้ง



รูป 5.39



รูป 5.40

รูป 5.41

ระบบหน่วยความจำ (Memory System)

ตามปกติไมโครโปรเซสเซอร์เป็นหัวใจของระบบไมโครคอมพิวเตอร์ทั่ว ๆ ไป และหน่วยความจำเปรียบเสมือนสมองของระบบ ซึ่งมีความสำคัญไม่ยิ่งหย่อนไปกว่าตัวไมโครโปรเซสเซอร์เอง ที่จริงแล้วหน่วยความจำยังเป็นส่วนที่มีราคาแพงมากในระบบอีกด้วย ดังนั้นจึงควรได้รับความสนใจอย่างจริงจัง หน้าที่สำคัญของหน่วยความจำในระบบไมโครคอมพิวเตอร์ก็คือ

1. เป็นหน่วยเก็บโปรแกรมคำสั่งที่จะสั่งให้ CPU ทำงานตามความต้องการ

2. เป็นหน่วยเก็บข้อมูลชั่วคราว ซึ่งหมายถึงข้อมูลสามารถเก็บเข้าไปในหน่วยความจำ หรือเอาออกจากหน่วยความจำได้ตามต้องการ ในขณะใช้งานหรือไม่ใช้ก็ได้

จะเห็นจากหน้าที่ทั้งสองของหน่วยความจำว่า ในบางครั้งสิ่งที่ถูกเก็บเอาไว้อาจเป็นในลักษณะถาวร นั่นคือไม่ต้องเปลี่ยนแปลงอีกต่อไป หลังจากเก็บไว้แล้ว เช่นเป็นโปรแกรมสั่งงานของ CPU ในงานประจำ แต่ในบางครั้งอาจมีการเก็บข้อมูลในลักษณะชั่วคราว เช่นเป็นตัวเลขจากการคำนวณพร้อมจะนำไปใช้งานต่อไป จากความจำเป็นนี้ทำให้ต้องมีหน่วยความจำมากกว่า 1 ประเภท เช่น อาจเป็นหน่วยความจำถาวรและหน่วยความจำชั่วคราวที่สามารถเก็บและเรียกออกมาใช้งานได้ทุกเมื่อ สำหรับในภาคนี้จะแบ่งเป็น 2 ตระกูล คือ ROM กับ RAM

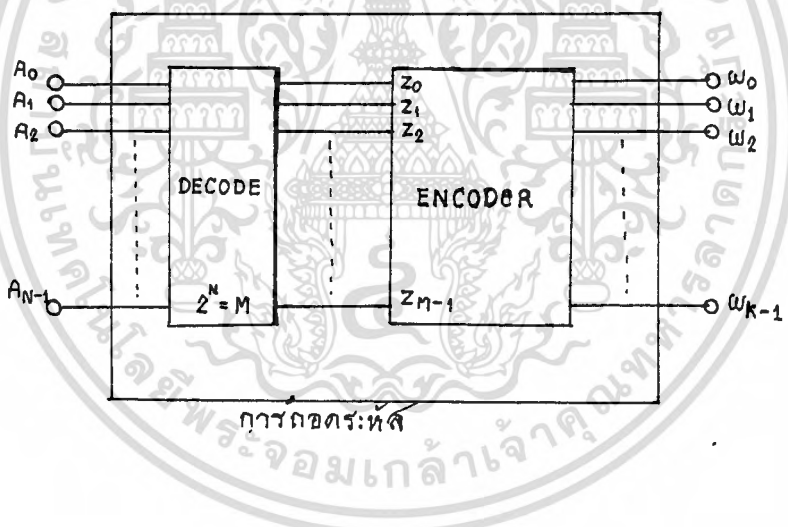
คุณสมบัติที่น่าสนใจ 2 อย่างของหน่วยความจำ ได้แก่ DESTRUCTIVENES คือการที่เราอ่านข้อมูลออกมาแล้วหน่วยความจำยังจำข้อมูลนั้นได้หรือไม่ และ VOLATILITY คือการที่เมื่อแหล่งจ่ายไฟยังจำข้อมูลนั้นได้หรือไม่ และหน่วยความจำในสมัยแรก ๆ ซึ่งเป็นพวก Magnetic Core Memory มักเป็น Destructive Memory แต่ Volatility ขึ้นอยู่กับชนิดและตระกูลของหน่วยความจำนั้น เช่น ROM มักเป็นพวก Nonvolatile ส่วน RAM มักเป็นพวก Volatile จึงต้องได้รับการเอาใจใส่เป็นพิเศษ โดยเฉพาะในแง่ของแหล่งจ่ายไฟเลี้ยงวงจร

ตระกูล ROM

ROM (Read - Only Memory) เป็นหน่วยความจำที่ใช้เก็บข้อมูลแบบถาวร หรือกึ่งถาวร นั่นคือข้อมูลต้องถูกเขียนไว้ในหน่วยความจำตั้งแต่ต้น หลังจากนั้นก็เป็นการอ่านข้อมูลออกมาเท่านั้น การเขียนข้อมูลทำได้ 2 วิธี คือ การเขียนข้อมูลจากโรงงานผู้ผลิต ตามความต้องการของลูกค้า เช่นเป็นชนิด Mask Programmed ส่วนอีกแบบหนึ่ง คือ ผู้ใช้สามารถโปรแกรมได้เองเรียกว่าเป็นชนิด PROM (Programmable Read-Only Memory) และก็มีบางประเภทที่ผู้ใช้สามารถเปลี่ยนข้อมูลจากหน่วยความจำมาก ประเภทนี้เป็นชนิด Reprogrammable ROM หรือบางครั้งอาจเรียกว่า Read - Mostly Memory (RMM)

ข้อดีอย่างหนึ่งของหน่วยความจำตระกูลนี้ คือ ข้อมูลจะไม่สูญหายถึงแม้ไฟฟ้าจะดับก็ตาม เราเรียกลักษณะอย่างนี้ว่า Nonvoltile ซึ่งต่างไปจากหน่วยความจำตระกูล RAM ซึ่งเป็นชนิด Volatile

ขอให้ดูรูปที่ 7.1 ซึ่งแสดงให้เห็นโครงสร้างภายในของ Rom ทั่วไป ตามรูป A0 ถึง AN-1 เป็นแอดเดรสของข้อมูลที่ต้องการอ่านซึ่งหลังจากผ่านวงจรถอดรหัส จะมีเพียงสายเดียวจากจำนวนทั้งหมด $2N = M$ สายที่ถูกกระตุ้นและหลังจากผ่านวงจรถอดรหัสแล้ว ข้อมูลจะมาปรากฏที่ขั้วออก WO - WK -1 จะเห็นว่าสัญลักษณ์เข้าที่มี N บิท และทำหน้าที่เสมือนเป็นแอดเดรสจะถูกแปลงเป็นสัญญาณออก K บิท ซึ่งก็คือข้อมูลที่เราต้องการโดยไม่สนใจว่า N จะมากกว่า หรือเท่ากับ หรือน้อยกว่า K ในที่นี้จะบอกได้ว่าหน่วยความจำตระกูล ROM นี้ มีขนาดเป็น M x K บิท ที่ M คือจำนวนเวิร์ดทั้งหมดในหน่วยความจำ และ K คือจำนวนบิทของแต่ละเวิร์ด



รูป 7.1 โครงสร้างภายในของ ROM

<u>ชนิดของ ROM</u>

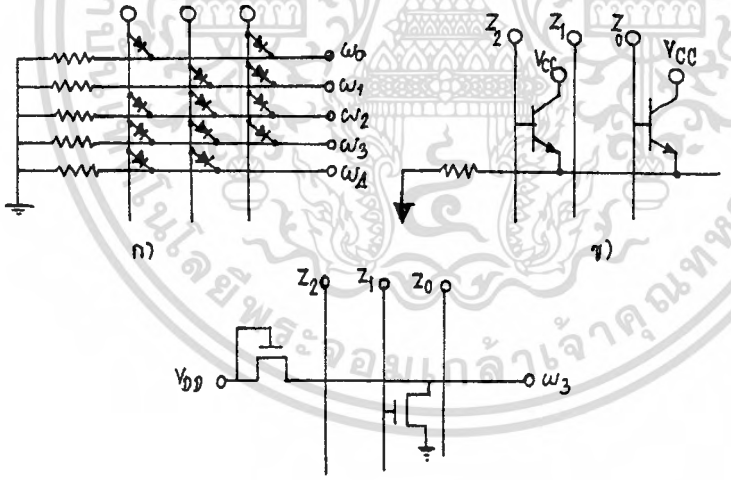
- Mask-Programmed ROM

Mask-Programmed ROM เป็นหน่วยความจำที่โปรแกรมเอาไว้อย่างถาวร ในระหว่างขบวนการผลิตโดยการทำ Mask บางอันเฉพาะสำหรับงานนั้น ๆ ROM ประเภทนี้สามารถทำได้จากสิ่งประดิษฐ์จำพวก ไดโอด ทรานซิสเตอร์ หรือ MOSEET อีกทั้งมีการสูญเสียกำลังค่อนข้างสูง ROM ที่ทำจาก BJT จึงมีจำนวนบิทน้อยกว่า ROM ที่ทำจาก MOSEET บนชิ้นสารกึ่งตัวนำที่มีพื้นที่เท่ากัน

ของหน่วยความจำ จะเห็นจากรูปที่ 7.2 ซึ่งแสดงเฉพาะภาควงจรเข้ารหัสของ ROM ว่าสายแอดเดรสและสายสัญญาณออกทั้งหมดจะประกอบกันเป็นตารางกริด ที่จุดตัด จะมีสิ่งประดิษฐ์อาจเป็นไดโอด หรือ BJT หรือ MOSEET ก็ได้ต่อเชื่อมโยงระหว่างสายแอดเดรสกับสายสัญญาณออกหรือไม่นั้น ขึ้นอยู่กับว่าที่บิทของแอดเดรสนั้น ๆ ต้องการให้เป็น 1 หรือ 0

- Programmabel ROM (PROM)

ที่จริงแล้ว PROM นั้นมีลักษณะวงจรภายในคล้ายคลึงกับพวก Mask-Programmed ROM และตามปกติมักจะทำเป็นตัว IC ที่มีขาอย่างเดียวกับ Mask-Programmed ที่เป็นคู่ของมันด้วย ทางผู้ผลิตได้ต่อหรือไม่ต่อเชื่อมโยงสายแอดเดรสกับสายสัญญาณออก ด้วยสิ่งประดิษฐ์สารกึ่งตัวนำที่จุดตัวในตารางกรีดตามที่เราต้องการให้เป็น 1 หรือ 0 ไว้เรียบร้อย ส่วนใน PROM ที่จุดตัดจากจุดในตารางกรีด จะมีสิ่งประดิษฐ์เพื่อให้ฟิวส์หลอมขาด ตัดการต่อเชื่อมโยงที่บิทนั้น ๆ หลังจากโปรแกรมเสร็จก็จะเหลือเฉพาะบิทที่ต้องการให้มีการต่อเชื่อมโยงด้วยสิ่งประดิษฐ์ลักษณะอย่างนี้ทำให้มีอีกชื่อหนึ่งว่า Field Programmable ROM ได้อย่างดีและโปรแกรมได้ง่ายด้วย

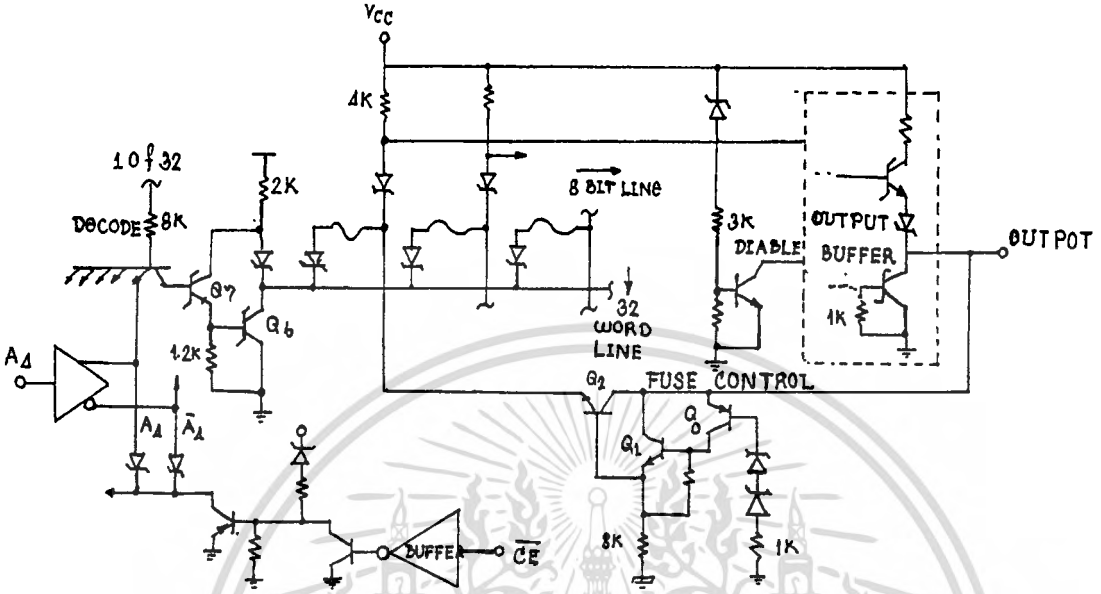


รูปที่ 7.2 วงจรภายในของภาควงจรเข้ารหัสของ ROM

ก. ไดโอด

ข. ทรานซิสเตอร์

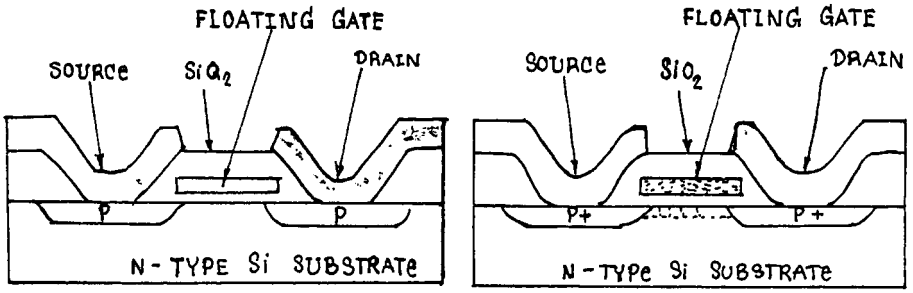
ค. MOSEET



รูป 7.3 วงจรภายในบางส่วนของ PROM

ลักษณะโครงสร้างภายในและแนวความคิดของ ROM ชนิดนี้แตกต่างจากของ PROM มากตามปกติ Reprogrammable ROM ทำจาก MOSEET โดยที่สภาวะ 0 หรือ 1 แทนการนำกระแสหรือไม่นำกระแสของ MOSEET จะเห็นว่าขาเกทถูกแยกจากส่วนที่นำกระแสอย่างเด็ดขาด ดูรูปที่ 7.4 ดังนั้นแรงดันที่ขาเกทจึงขึ้นอยู่กับประจุที่ขาเกท ลักษณะโครงสร้างของขาเกทกับส่วนนำกระแส เปรียบเสมือนตัวเก็บประจุที่มีชั้นออกไซด์เป็นตัวกลางไดอิเลคตริค นี่เองเป็นแนวความคิดในการพัฒนา Reprogrammable ROM ขึ้นมา โดยวิธีการดักประจุไว้ที่ขาเกท และประจุนั้นสามารถค้างอยู่ที่ขาเกทได้เป็นเวลานาน

แนวความคิดดังกล่าวยังแยกออกเป็น 2 วิธี ทั้งนี้ขึ้นอยู่กับผลของการดักประจุในขาเกท วิธีแรกคือการดักประจุไว้ที่ขาเกทซึ่งถูกแยกจากส่วนนำกระแสของ MOSEET ด้วยชั้นของซิลิกอนไดออกไซด์ ทำให้ MOSEET พาหะประจุข้อมูลที่ถูกโปรแกรมไปแล้ว ทำได้วิธีเดียวคือการลบ โดยฉายด้วยรังสีอุลตราไวโอเล็ตเพื่อให้พาหะประจุที่ถูกดักอยู่มีพลังงานเพียงพอ



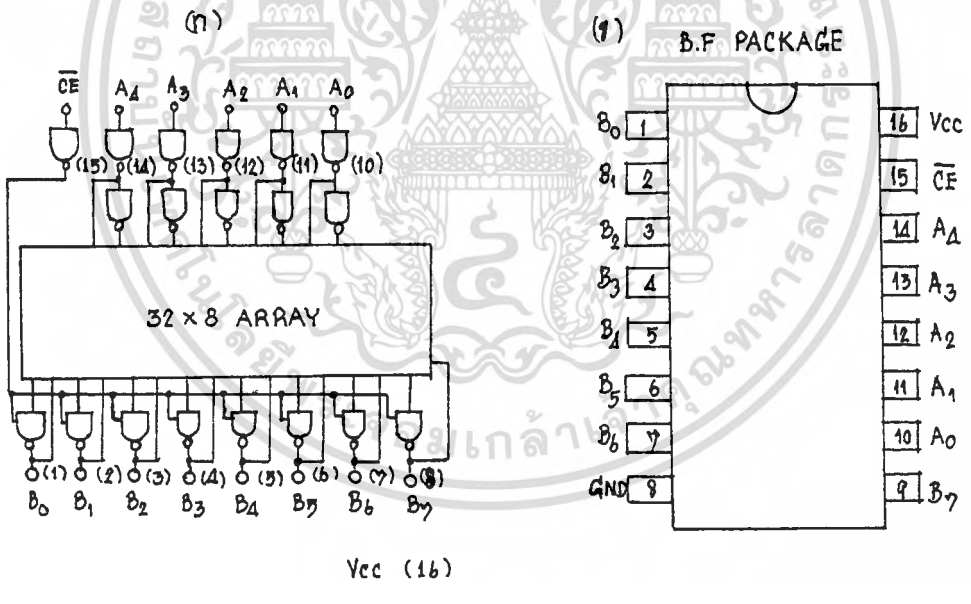
รูป 7.4 โครงสร้างของ MOSFET

ตัวอย่างและการโปรแกรม PROM และ EPROM

ในที่นี้จะยกตัวอย่างเพียงชนิด PROM และ EPROM เพราะเป็นหน่วยความจำที่เราสามารถใช้งานได้กว้างกว่า ROM ชนิดอื่น อีกทั้งเป็นพื้นฐานในการเข้าใจวิธีการโปรแกรม ROM ทั้ง 2 ชนิด ในหัวข้อถัดไป

ประเภทที่จะพูดถึงคือ PROM ขอยกตัวอย่าง IC เบอร์ 8s 23/123 ซึ่งเป็น Bipolar PROM ขนาด 32x8 บิท (ที่จริงแล้วเป็นชนิด Schottky-Diode Clamped Transistor)

การทำงานอย่างคร่าว ๆ เมื่อเลือกแอดเดรส (A0-A4) แล้วแอดเดรสจะถูกส่งรหัสเป็น 1 ใน 32 สาย ซึ่งจะกระตุ้นเซลหน่วยความจำทั้ง 8 บิท ผ่าน Q7 และ Q8 ในขณะเดียวกัน เราต้องต่อส่งสัญญาณมาด้วยว่าเรากำลังเลือกใช้ IC ตัวนี้ โดยบังคับขา CE เป็น 0 หลังจากผ่านบัฟเฟอร์และ Q4 มันจะบังคับให้เซลทั้ง 8 บิทของหน่วยความจำแอดเดรสนี้ทำงาน แรงดันที่ขั้วออกแต่ละบิท (B0-B7) ในรูปที่ 7.5 ที่ต้องการโปรแกรม (หลังจากโปรแกรมแล้ว บิทนั้นจะเป็น 0) จะต้องถูกยกระดับขึ้นให้สูงพอ ทำให้ Q0, Q1, Q2 นำกระแสได้ ในขณะที่ Q6, Q7 ก็นำกระแสด้วยเช่นกัน จึงมีกระแสไหลผ่านฟิวส์ได้มากพอทำให้ฟิวส์หลอมขาด ส่วนบิทใดที่ไม่ต้องการโปรแกรม ก็ไม่ต้องให้แรงดันที่ขั้วออกนั้น ซึ่งไม่มีกระแสไหลผ่านฟิวส์ บิทนั้นยังคงเป็น 0 และขั้วออกของแต่ละบิทยังผ่านบัฟเฟอร์ที่สามารถเอนเอเบิลได้ตามความต้องการ โดยบังคับด้วยสาย CE



รูป 7.5 ก. รูปตรรกของ 82 s 23/123

ข. รูปแสดงชื่อสัญญาณที่ขาต่าง ๆ

รูปที่ 7.6 แสดงวงจร Manual Programmer เครื่องโปรแกรมโดยใช้มือของ 82 s 23/123 รวมทั้งแสดงรูปแสดงลำดับต่าง ๆ ที่เกี่ยวข้องกับการโปรแกรม PROM วงจรนี้สามารถดัดแปลงเป็นเครื่องโปรแกรมอัตโนมัติได้ไม่ยากนัก (แต่ยังไม่มีความจำเป็นในรูปขณะนี้) R1 เป็นวงจรควบคุมแรงดันที่ 5 โวลต์ (Vcc ในรูป) สำหรับตรวจสอบสภาวะ (0 หรือ 1) ของเซลหน่วยความจำที่ถูกต้องการโปรแกรม R2 เป็นวงจรควบคุมกระแสคงที่ เพื่อเป็นแหล่งจ่ายกระแสไฟฟ้าให้ฟิวส์หลอมละลายตามต้องการ

สัญญาณสตาร์ทดังรูปแสดงลำดับช่วงเวลาในการโปรแกรม หลังจากนี้ T1-T5 จะให้สัญญาณพัลซ์ตามรูปแสดงลำดับช่วงเวลาในการโปรแกรมดังนี้

1. T1 จะเป็น 0 เป็นเวลา 5 มิลลิวินาที ระหว่างนี้ Q1 ถูกบังคับให้หยุดนำกระแสส่วน Q2 ยังคงไม่นำกระแส ดังนั้นสาย CE ยังเป็น 1 อยู่ PROM จึงยังไม่ถูกกระตุ้นให้ทำงานวงจรขับ 1 ของ 7005451 จะบังคับให้ R2 จ่ายแรงดัน 10 โวลต์ให้ PROM

2. T2 ซึ่งเป็น 1 พร้อมกับ T1 โดยเป็น 1 เป็นเวลาเพียง 1 มิลลิวินาที เป็นเพียงสัญญาณชะลอเท่านั้น

3. T3 จะเป็น 1 ในทันทีที่ T2 เปลี่ยนเป็น 0 และ T3 เป็นเวลาเพียง 3 มิลลิวินาที เพื่อเปิดทางให้กระแสจากแหล่งควบคุมกระแส R3 พร้อมที่จะจ่ายกระแสให้แก่บิทที่ต้องการโปรแกรมผ่านทางวงจรขับ 2 ของ 75451

4. T4 ซึ่งเป็น 1 พร้อมกับ T3 โดยเป็น 1 เป็นเวลาเพียง 1 มิลลิวินาที เป็นเพียงสัญญาณชะลอเท่านั้น

5. T5 เป็นสัญญาณเอเบิล PROM โดยบังคับให้ Q2 นำกระแสเป็นเวลา 1.5 มิลลิวินาที ทำให้สาย CE เป็น 0 ในช่วงเวลานี้เซลหน่วยความจำในเฉพาะแอดเดรสที่ถูกกำหนดโดยสวิทช์

รูปที่ 7.7 แสดงรายละเอียดสเปคในการโปรแกรม PROM 82s 23/123 ซึ่งเป็นเวลาที่มาของวงจรเครื่องโปรแกรมโดยใช้มือ และรูปแสดงลำดับช่วงเวลาในการโปรแกรมในรูปที่ 7.6 ข้อที่ต้องระวังในการโปรแกรม PROM ตัวนี้ เพื่อให้ได้ผลที่น่าเชื่อถือ ก็ได้แก่สิ่งต่อไปนี้

1. แหล่งควบคุมแรงดันและกระแสคงที่ ต้องไม่เกินพิกัดที่กำหนดไว้ในสเปค

2. ความกว้างของพัลซ์โปรแกรมของขา CE (ช่วงเวลาที่ TS เป็น 1) ต้องไม่สั้นกว่า 1 มิลลิวินาที และต้องไม่เกิน 2 มิลลิวินาที

3. เวลาในการกำหนด (ช่วงเวลาที่ T1 เป็น 1) ต้องไม่นานเกิน 2.5 วินาที

4. เวลาการพักโปรแกรม (ช่วงเวลาที่ไม่จ่ายแรงดันหรือกระแสให้กับ PROM) ต้องไม่น้อยกว่า 2 เท่าของเวลาในการโปรแกรม เพื่อให้ได้ Duty Cycle ของการโปรแกรมน้อยกว่า 33 เปอร์เซนต์ ตามที่ผู้ผลิตกำหนดมา

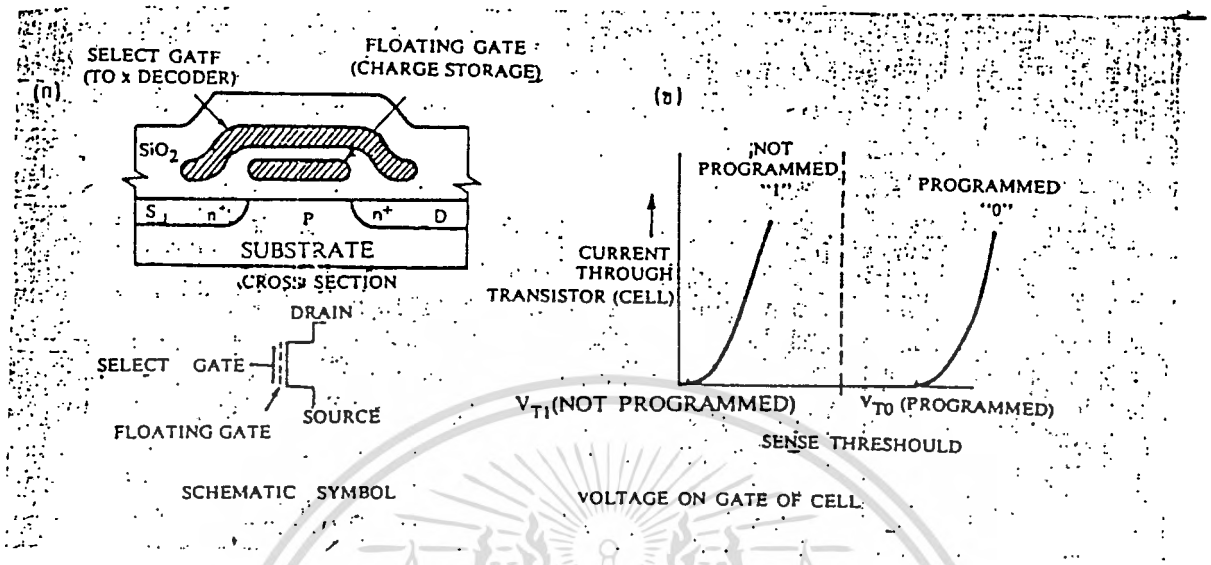
ที่จะยกตัวอย่างต่อไปนี้คือ EPROM ได้แก่เบอร์ 2708 ซึ่งเป็น EPROM ขนาด 1024X8 บิท ทำจาก MOSFET ชนิด Nchannel ที่มีขาเกทลอยฝังอยู่ในชั้นออกไซด์ ทำหน้าที่เก็บประจะโดยไม่ต่อกับวงจรส่วนอื่น และมีขาเกทเลือกต่อมาจากวงจรถอดรหัสเลือกแถว ทั้งนี้มันจะทำงานร่วมกับวงจรถอดรหัสเลือกคอลัม เพื่อเลือกว่าเซลหน่วยความจำ 8 บิทชุดใดถูกบังคับให้ทำงาน รูปที่ 7.8 แสดงเซลหน่วย 1 เซล เมื่อยังไม่ถูกโปรแกรม ดังนั้น เมื่อป้อนแรงดันที่ระดับ Sense Threshold ที่ขาเกทเลือก จะทำให้ตัวทรานซิสเตอร์สามารถนำกระแสได้ดีมาก ส่วนในกรณีที่ถูกโปรแกรมมีประจุในขาเกทลอย เซลจะมีสภาวะเป็น 0 และกราฟลักษณะสมบัติโอนย้ายจะเป็นดังเส้นทางขวามือ ดังนั้นเมื่อป้อนแรงดันระดับ Sense Thershold ที่ขาเกทเลือก ตัวทรานซิสเตอร์ไม่สามารถนำกระแสได้ การโปรแกรมนั้นจึงเป็นการเลื่อน กราฟลักษณะสมบัติโอนย้ายไปทางขวามือ หรือกล่าวอีกนัยหนึ่งเป็นการเปลี่ยนค่าของตัวพารามิเตอร์ที่สำคัญ อันได้แก่ Theshold Voltage VT ให้สูงขึ้น

รูปที่ 7.9 แสดงชื่อสัญญาณของขาต่าง ๆ ของ IC เบอร์นี้ รวมทั้งบล๊อคไดอะแกรมภายในด้วย A0-A9 เป็นแอดเดรส 10 บิท ของเซลหน่วยความจำที่ต้องการเขียนหรืออ่านโดย A0-A3 เลือกคอลัมน์ A4-A9 เลือกแถวของหน่วยความจำ O0-O8 เป็นขั้วเข้าหรือขั้วออกของข้อมูลที่ถูกเขียนหรืออ่านออกมา CS/WE คือขาที่ใช้เลือกว่าจะให้ IC ตัวนี้ทำงานหรือไม่ ถ้าทำงานจะทำงานในโหมดเขียนหรืออ่านออกมา ส่วนรูปที่ 7.10 แสดงการต่อขาเข้ากับแรงดันต่าง ๆ ในโหมดทั้งสามรวมทั้งแรงดันที่ปรากฏที่ขาต่าง ๆ

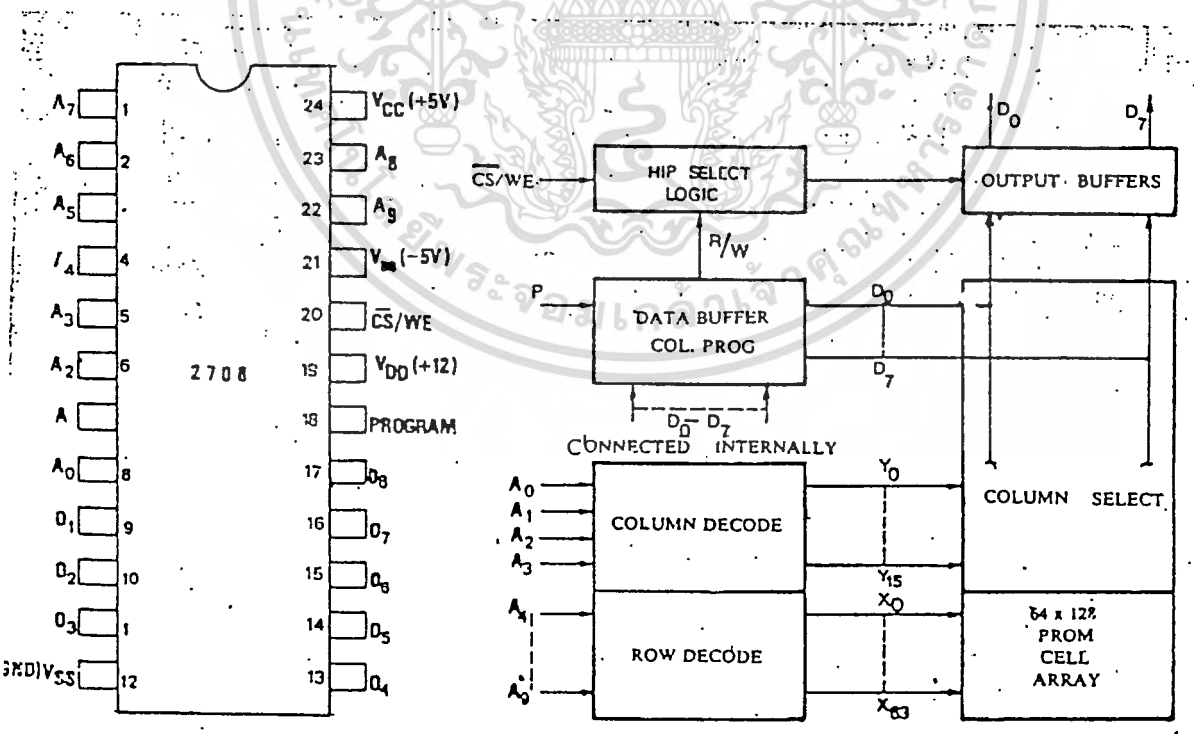
ในการใช้งาน EPROM ตัวนี้ สามารถเลือกใช้งานได้ 3 โหมด โหมดแรกคือโหมดไม่เลือก นั่นคือ O1-O8 เป็นอิมพีแดนซ์ค่าสูงโดยไม่คำนึงว่า A0-A9 เป็นอะไร ทำให้สามารถต่อร่วมกับหน่วยความจำตัวอื่น ๆ ได้ เพื่อเพิ่มขนาดของหน่วยความจำให้มากขึ้น รายละเอียดของการ OR Tie จะได้พูดกันอีกครั้งในภายหลัง EPROM จะได้ทำงานในโหมดนี้ได้โดยให้ขา CS/WE มีศักดาเป็น VIH

โหมดที่สองคือ โหมดอ่าน นั่นคือ O1-O8 จะให้ข้อมูลของแอดเดรส (A0-A9) ที่เราป้อนให้กับมันออกมา 2708 ทำงานในโหมดนี้โดยให้ขา CS/WE มีศักดาเป็น VIL รูปที่ 7.11 แสดงรูปคลื่นของสัญญาณต่าง ๆ กรณีที่ต้องการให้ 2708 ทำงานในโหมดอ่าน คือเวลาเข้าถึงหน่วยความจำได้แก่ช่วงเวลาหลังจากที่ป้อนแอดเดรสเข้าไป จนถึงหน่วยความจำออกมาและก่อนที่ข้อมูลจะเปลี่ยนแปลง TDF คือช่วงเวลาหลังจากให้ขา CW/WE เปลี่ยนเป็น VIH ก่อนที่ข้อมูลจะเปลี่ยนแปลง

โหมดที่สาม คือโหมดโปรแกรม ในโหมดนี้ O1-O8 คือขั้วเข้าของข้อมูลที่ต้องถูกกำหนดด้วย A0-A9



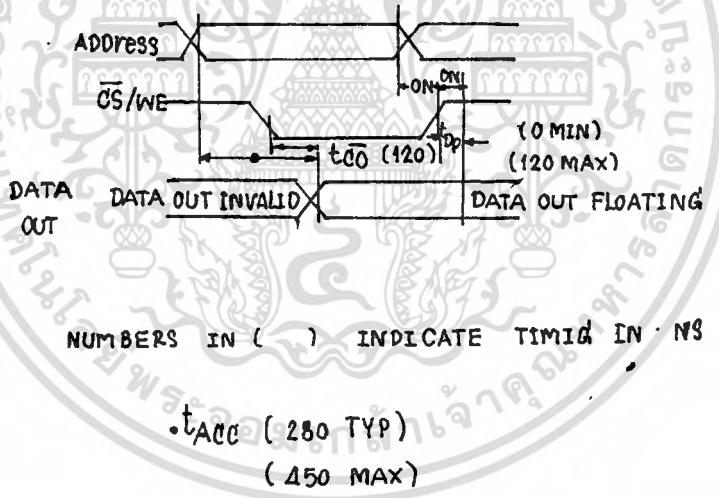
รูปที่ 7.8 (ก) เซลหน่วยจำพื้นฐานใน 2708
(ข) การเปลี่ยนค่า Threshold Voltage ของเซลหน่วยจำอัน
เกิดจากการโปรแกรม



รูปที่ 7.9 ชื่อสัญญาณที่ขาต่าง ๆ และบล็อกไดอะแกรมของ 2708

Function / Pin number / MODE	DATA I/O 9-11, 13-17	Address 1-7, 23, 22	V_{SS} (GND) 12	Program 18	V_{DD} Supply 19	CS/WE 20	V_{BB} supply 21	Vcc Supply 24
READ	D_{OUT}	A_{IN}	GND	GND	+12 V	V_{IL}	-5V	+5V
Deselect	High Impedance	Don't Care	GND	GND	+12V	V_{IH}	-5V	+5V
Program	D_{IN}	Ain	GND	Pulsed +26V	+12V	V_{IHW}	-5V	+5V

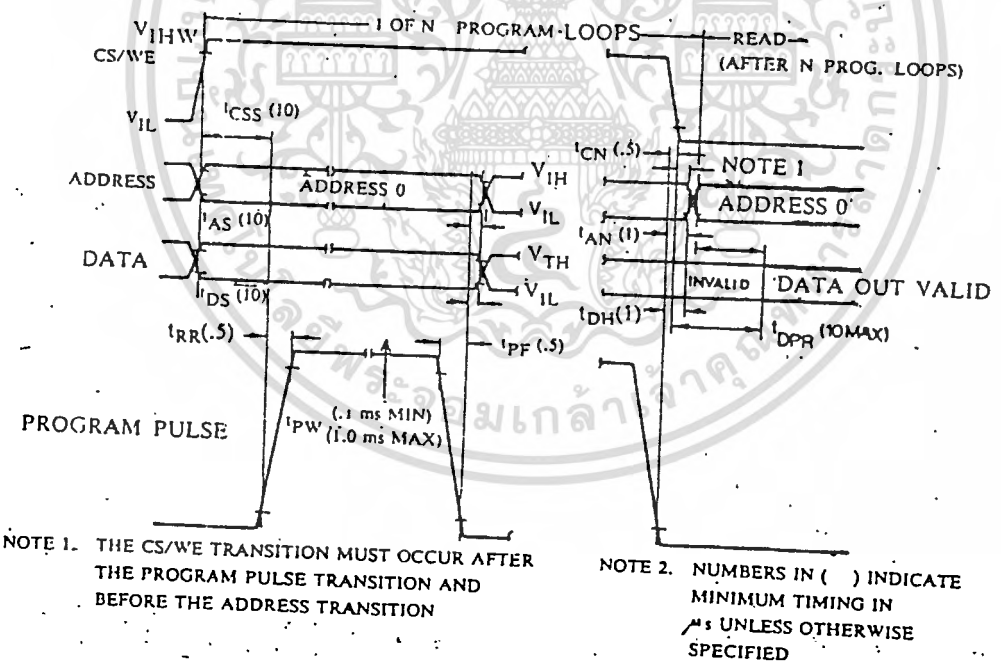
รูปที่ 7.10 ตารางต่อขาในโหมดทั้งสามของ 2708



รูปที่ 7.11 รูปคลื่นในจังหวะอ่านของ 2708

เมื่อต้องการโปรแกรม 2708 ให้ขา CS/WE มีศักดาเป็น VIHW (12+0.6 Volts) แล้วป้อนเดรสให้ 2708 และป้อนข้อมูลทั้ง 8 บิท (แบบขนานกัน) ตามที่ต้องการโปรแกรมเข้าที่ขั้วเข้าตามแอดเดรสนั้น ๆ ทั้งแอดเดรสและข้อมูลมีระดับแรงดันเช่นเดียวกับโหมดอ่าน และเท่ากับระดับแรงดันของ TLL ขอให้ดูระดับแรงดันต่าง ๆ ในรูปที่ 7.10 พอสิ้นเวลา TAS และ TDSS ดังรูปที่ 7.12 เราถึงจะป้อนพัลซ์โปรแกรมที่มีระดับแรงดันเป็น VIHP (26 +-1) และพัลซ์โปรแกรมมีค่าเท่ากับ

VIHP-VIL>=25 โวลท์ โปรแกรมแอดเดรสนี้เสร็จก็เปลี่ยนแอดเดรสแล้ว โปรแกรมต่อไปจนครบ 1024 แอดเดรสเรียกว่าครบหนึ่งรอบโปรแกรม เพื่อให้การโปรแกรมได้ผลที่น่าเชื่อถือ จะต้องโปรแกรมเช่นนี้เป็นจำนวนหลาย ๆ รอบโปรแกรม



รูปที่ 7.12 รูปคลื่นในจังหวะโปรแกรมของ 2708

Symbol	Parameter	Min.	Typ.	Max.	Units
t_{AS}	Address Setup Time	10			μS
t_{CSS}	$\overline{CS}$/WE Setup Time	10			μS
t_{DS}	Data Setup Time	10			μS
t_{AH}	Address Hold Time	1			μS
t_{CH}	$\overline{CS}$/WE Hold Time	.5			μS
t_{DH}	Data Hold Time	1			μS
t_{DF}	Chip Deselect to Output Float Delay	0		120	ns
t_{DPR}	Program To Read Delay			10	μS
t_{PW}	Program Pulse Width	1		1.0	S
t_{PR}	Program Pulse Rise Time	.5		2.0	ms
t_{PF}	Program Pulse Fall Time	.5		2.0	μS

รูปที่ 7.13 ตารางเวลาที่เกี่ยวข้องกับการโปรแกรม

ตามรูปที่ 7.12 และ 7.13 ช่วงเวลาในการโปรแกรมที่ได้ผลดีมาก ก็คือการกำหนดค่าเวลาต่าง ๆ ดังนี้

0 TCSS = TAS = TDS = 10 ไมโครวินาที

TPW = 10 มิลลิวินาที

TAS = TDS = 1.0 ไมโครวินาที

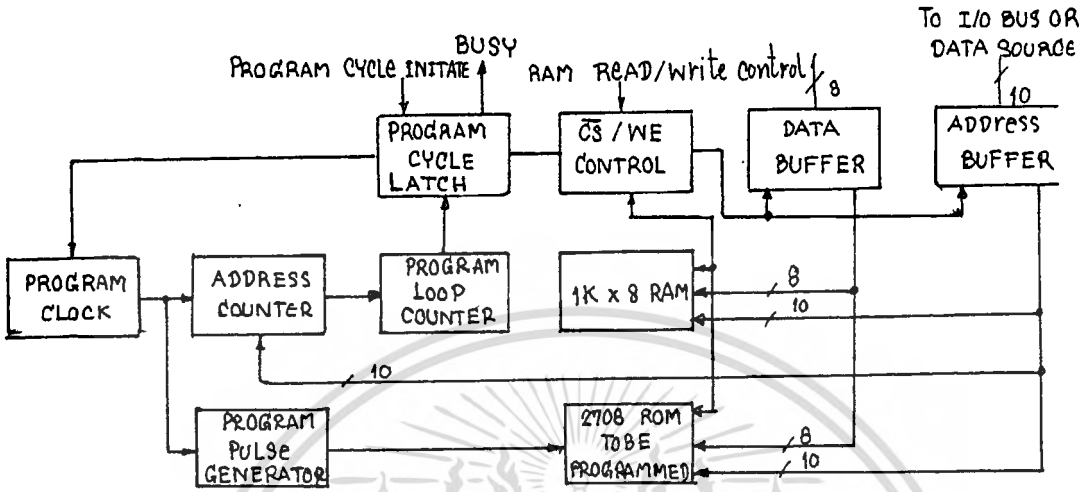
TPR = TPF = 0.5 ไมโครวินาที

ดังนั้นในการโปรแกรม 1 แอดเดรส จะใช้เวลาเป็น

TAS + TPR + TPF + TAH = 1.012 มิลลิวินาที และสำหรับ IC 1 ตัว (1024 แอดเดรส จำนวน 100 รอบ จะใช้เวลา 1:012 x 100 x 1024 = 103.6 วินาที

รูปที่ 7.14 แสดงบล็อคไดอะแกรมของชุดโปรแกรม 2708 ซึ่งมีการทำงานดังนี้ ข้อมูลทั้ง 8 บิท ผ่านบัฟเฟอร์เข้ามาเก็บใน RAM ตามแอดเดรสที่ผ่านบัฟเฟอร์เข้าไป ทั้งนี้ข้อมูลและแอดเดรสอาจมาจากระบบไมโครโปรเซสเซอร์ ที่ทำหน้าที่จ่ายข้อมูลออกมาก็ได้ หรืออาจมาจากกดแป้นข้อมูลเข้าก็ได้ เมื่อข้อมูลถูกเก็บไว้ใน RAM ครบทั้ง 1024 แอดเดรสแล้ว จะมีสัญญาณสั่งเริ่มต้นโปรแกรมผ่านเข้ายัง Program Cycle Latch ซึ่งจะส่งสัญญาณ Busy กลับไปยังโปรเซสเซอร์ และยุติการส่งสัญญาณทั้งข้อมูลและแอดเดรส Program Cycle Latch จะส่งสัญญาณให้วงจรกำเนิดสัญญาณนาฬิกาควบคุมโปรแกรมเริ่มทำงานและให้ RAM เริ่มทำงานด้วยทำให้ข้อมูลจาก RAM จะเพิ่มค่าขึ้นอีก 1 แล้วการเขียนข้อมูลจาก RAM

ลงใน EPROM ก็จะดำเนินต่อไปจนครบ 1024 แอดเดรสวงจร



รูป 7.14 บล๊อคไดอะแกรมของเครื่องโปรแกรม 2708

ชนิดของ RAM

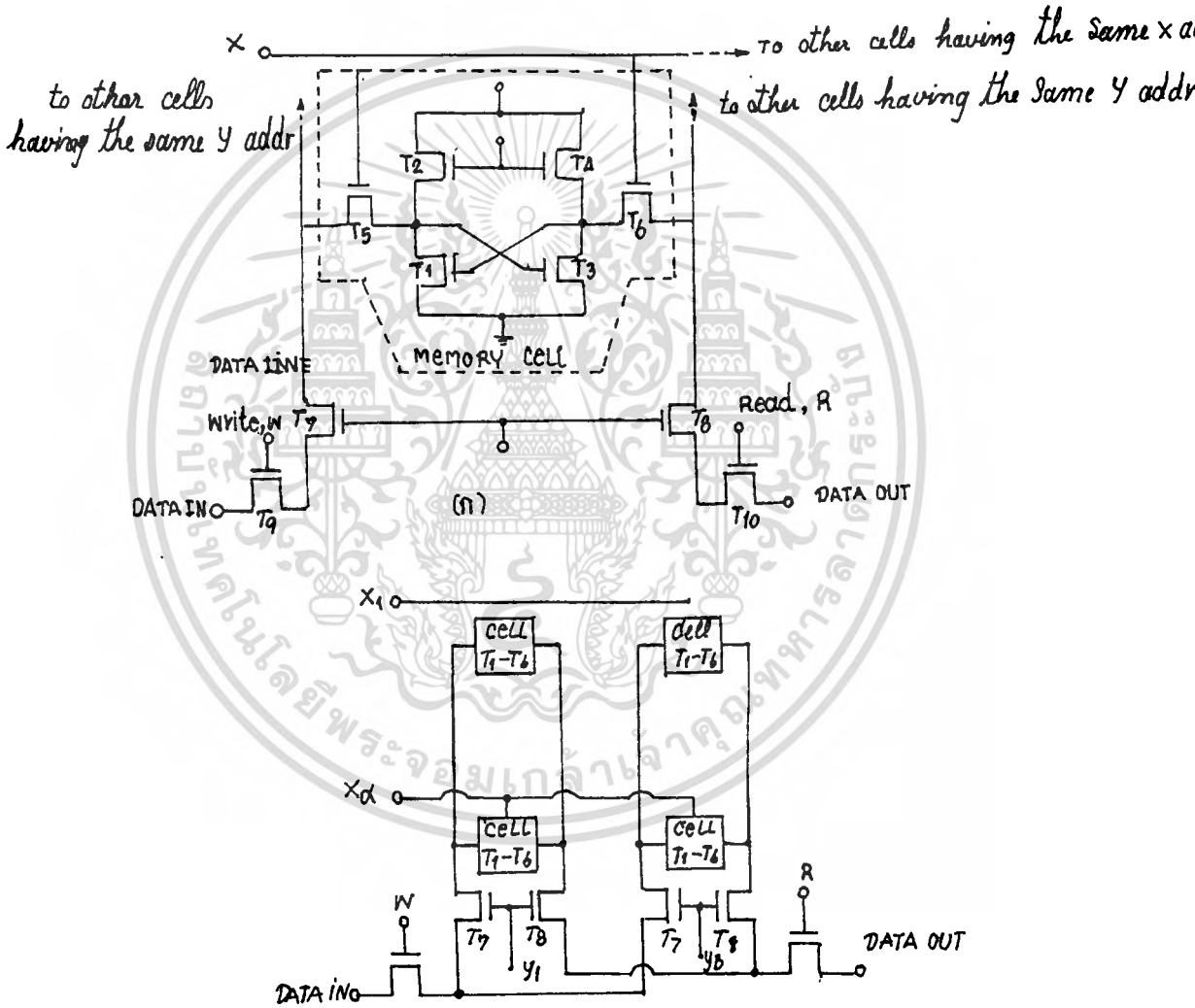
RAM ชนิดสตาติค

RAM ชนิดนี้ใช้ลักษณะวงจรฟลิปฟลอปเป็นวงจรพื้นฐานสำหรับเซลหน่วยความจำแต่ละเซล ตามปกติวงจรพื้นฐานก็มักไม่แตกต่างกันมากนัก และโดยมากเป็นดังรูปที่ 7.15 ก. T1-T6 คือ ทรานซิสเตอร์ที่ประกอบเป็นวงจรฟลิปในเซลพื้นฐาน ส่วน T7-T10 เป็นเกทที่เปิดให้ข้อมูลไหลผ่านไปมาได้ เป็นวงจรร่วมของเซลหน่วยความจำหลาย ๆ เซล ดูรูปที่ 7.15 ข. ดังนั้นเราจึงเรียกเซลหน่วยความจำแบบนี้ว่าเซลหน่วยความจำชนิดสตาติคแบบทรานซิสเตอร์ 6 ตัว

ลักษณะการทำงานเป็นดังนี้ T1,T2 และ T3,T4 ประกอบกันเป็นวงจรกลับเฟสที่ต่อโยงไขว้กันอยู่ทำหน้าที่เป็นฟลิปฟลอปโดย T5,T6 เป็นเกทเปิดหรือปิดตามแต่จะเลือกด้วยระดับแรงดันในสาย (X) (มาจากวงจรถอดรหัสเลือกแถว) T7,T8 เป็นเกทเปิดหรือปิด ตามแต่จะเลือกด้วยระดับแรงดันในสาย Y (มาจากวงจรถอดรหัสเลือกคอลัมน์) T9,T10 เลือกว่าจะให้เขียนข้อมูลเข้าไปหรืออ่านข้อมูลออกมา

เมื่อต้องการเลือกเซลใดก็ให้สาย X,Y ของเซลนั้นเป็น 1 ทำให้สาย DATA ต่อกับเซลได้ และยังต่อออกมาที่สาย DATA IN หรือสาย DATA OUT ก็ได้ ด้วยการให้ W หรือ R เป็น 1 ตามลำดับ ถ้าต้องการเขียนข้อมูลเข้าไปในเซล ก็ให้ข้อมูลนั้นปรากฏที่สาย DATA IN และ W=1 ตามปกติเรามักให้

R = W = 0 ดังนั้นข้อมูลจึงถูกส่งผ่าน T9,T7,T5 เข้าไปเก็บเป็นองค์ประกอบของแต่ละเซลฟ ส่วนการทำงานยังเหมือนที่กล่าวไปแล้วเพียงแต่มีสาย X มากขึ้นเป็น 2 สาย และสาย Y มากขึ้นเป็น 3 สาย โดยมาทั้งสาย X,Y นี้มาจากวงจรถอดรหัสเลือกแถวและคอลัมน์ที่อยู่ในตัว IC และตามปกติแอดเดรสที่จะเข้ามักจะอยู่ในรูปรหัสเลขฐาน 2 n บิท การบอกขนาดของ RAM ก็เช่นเดียวกับ ROM คือบอกเป็น MxK บิท เมื่อ M คือจำนวนเวิร์ด และ K คือจำนวนบิทของแต่ละเวิร์ด หรือจำนวนบิททั้งหมดที่สามารถเข้าถึงได้ด้วยแอดเดรสเดียวกัน



รูปที่ 7.15 ก. เซลพื้นฐานของ RAM ชนิดสถาติค

ข. การต่อเซลต่า ๆ เข้าด้วยกันภายในตัว IC

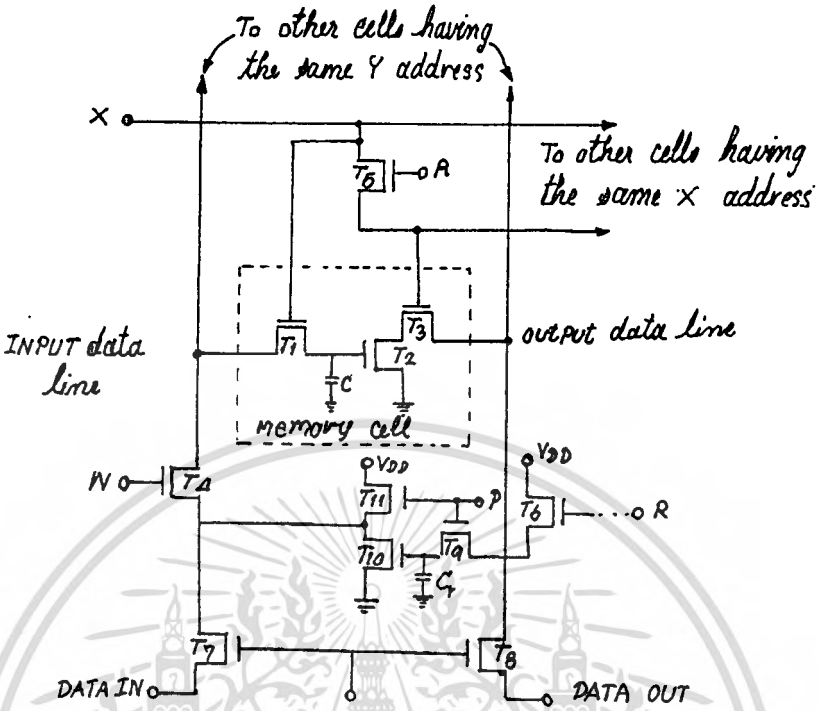
RAM ชนิดไดนามิค

RAM ชนิดนี้ใช้การเก็บประจุในตัว เก็บประจุเข้าที่ขาเกทของ MOSFET แทนการเก็บข้อมูลวงจรพื้นฐานของ RAM ชนิดไดนามิคมีได้หลายรูปแบบ และใช้จำนวนทรานซิสเตอร์แตกต่างกันออกไป RAM ชนิดไดนามิคมีได้มีข้อดีเหนือกว่า RAM ชนิดสตาติคมากนัก นอกจากการมีเวลาเข้าถึงหน่วยความจำน้อยกว่า RAM ชนิดสตาติคเล็กน้อย แต่ต้องเสียเวลาในการรีเฟรชเท่านั้น สิ่งที่น่าสนใจอย่างยิ่งก็คือการใช้จำนวนทรานซิสเตอร์ต่อ 1 เซลน้อยกว่าของ RAM ชนิดสตาติคค่อนข้างมาก ทำให้เนื้อที่ 1 ตารางหน่วยสามารถบรรจุ RAM ชนิดไดนามิคได้จำนวนเซลมากกว่าของ RAM ชนิดสตาติคค่อนข้างมาก ที่จริงแล้วแต่ละเซลของ RAM ชนิดไดนามิคอาจประกอบด้วยทรานซิสเตอร์ 1 ตัว และตัวเก็บประจุ 1 ตัวเท่านั้น (คือ T1 และ C ในรูปที่ 7.15)

เพื่อจะเข้าถึงเซลใดเซลหนึ่ง เราต้องให้สาย X และ Y ของเซลนั้นเป็น 1 และแยกวงจรรีเฟรชออกจากเซลหน่วยความจำให้ P=0 ถ้าเราต้องการเขียนข้อมูลจากเซลหน่วยความจำ ต้องให้ R=1, W = 0 T5,T6,T3 จะนำกระแสโดย T2 เป็นวงจรกลับเฟสของข้อมูลที่เก็บใน C ออกมาปรากฏในสาย DATA OUT

พอเวลาผ่านไปเล็กน้อย เราต้องรีเฟรชหน่วยความจำทั้ง โดยให้ Y=0,X=1,P=1 และ R=1 T7-T8 จะหยุดกระแสข้อมูลใน C ถูกกลับเฟสโดย T2 และผ่าน T9 มาประจุให้ Cr เราเรียกช่วงขณะนี้ว่าช่วงเวลาประจุล่วงหน้า (Precharge Period) หลังจากให้ประจุแก่ Cr เต็มที่แล้วก็ให้ R=0, W=1, T10 จะกลับเฟสของข้อมูลที่เก็บไว้ใน Cr ผ่าน T4, T1 กลับไปประจุให้ C ใหม่อีกครั้งหนึ่งจะเห็นว่าข้อมูลของ C ถูกกลับเฟส 2 ครั้ง ด้วยวงจรกลับเฟส T2,T3,T6 และ T10,T11 ดังนั้นเมื่อรีเฟรชข้อมูล C โดยผ่าน T4,T1 ในการรีเฟรชหน่วยความจำ เราจะรีเฟรชหน่วยความจำหมดทั้งแถวเวลาในการรีเฟรชหน่วยความจำหมดทั้งแถวจะเป็นพร้อมกันเวลาในการรีเฟรชแต่ละแถว

ไม่ว่าจะเป็นวิธีใดก็ตาม การรีเฟรชจะเกิดขึ้นได้ก็ต่อเมื่อหน่วยความจำนั้นต้องไม่อยู่ในระหว่างที่ใช้งานทั้งนี้การรีเฟรชอาจกระทำในลักษณะที่ซิงโครไนซ์กับสัญญาณนาฬิกาของระบบหรือไม่ซิงโครไนซ์ก็ได้



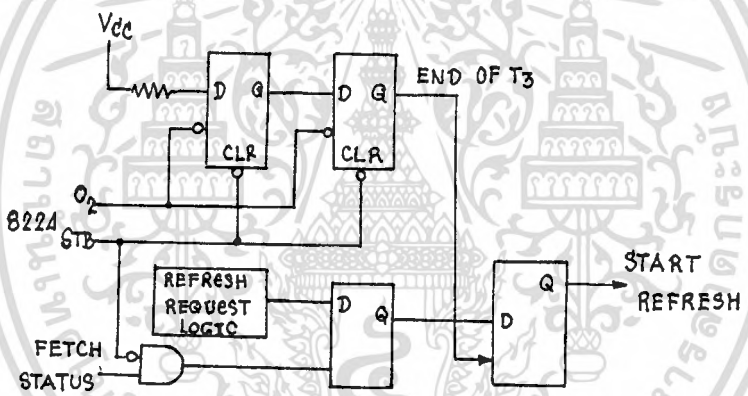
รูปที่ 7.16 เซลหน่วยความจำ RAM ชนิดไดนามิคแบบทรานซิสเตอร์ 8 ตัว

1. Asynchronous Access สัญญาณขอทำการรีเฟรชจะถูกส่งออกมาด้วยอัตราคงที่ เช่น ทุก 31 ไมโครวินาที (ในกรณีมี 64 แถว) โดยไม่ขึ้นกับชนิดหรือเบอร์ของไมโครโปรเซสเซอร์ ข้อเสียคือ ต้องใช้วงจรควบคุมการรีเฟรชที่ยุ่งยาก และแน่นอนผลเสียที่ตามมาอีกอย่างหนึ่งก็คือ การมีเวลาชลอซึ่งยาวนานขึ้น นอกจากนี้ สิ่งที่ต้องคำนึงถึงอีกสิ่งหนึ่งก็คือ ปัญหาความสำคัญก่อนหลัง ในการเข้าถึงหน่วยความจำ

2. Synchronous Access ซึ่งเรารู้จักกันในชื่ออื่นอีกอย่างเช่น Hidden Refresh หรือ Transparent Refresh จากชื่อหลังทั้งสองนี้ทำให้พอนึกภาพออกได้ว่า มันเป็นวิธีการรีเฟรชหน่วยความจำในขณะที่ MPU ไม่ได้เรียกใช้งาน ในการใช้งานโดยทั่วไปมักมีช่วงเวลาอยู่อย่างน้อยหนึ่งไซเคิล MPU ไม่ได้ใช้งานหน่วยความจำ และถ้าเราสามารถทราบช่วงเวลานี้ได้อย่างแน่ชัด ก็สามารถทำการรีเฟรชหน่วยความจำได้โดยที่ MPU ไม่ได้สูญเสียประสิทธิภาพอะไร แน่นอนเมื่อ MPU ไม่มีการสูญเสียเวลาไป MPU จึงย่อมไม่รู้ว่ามีการรีเฟรชหน่วยความจำขึ้นในจังหวะนี้ วิธีการนี้มีข้อดีในเรื่องความเร็ว แต่ข้อเสียก็มีเช่นกัน ได้แก่วงจรควบคุมการรีเฟรชของไมโครโปรเซสเซอร์ชนิดใด ก็เป็นชนิดนั้นเราไม่สามารถนำไปใช้กับไมโครโปรเซสเซอร์ชนิดอื่นได้ นอกจากนี้เรายังต้องระวังเกี่ยวกับเหตุการณ์ที่ผิดปกติ

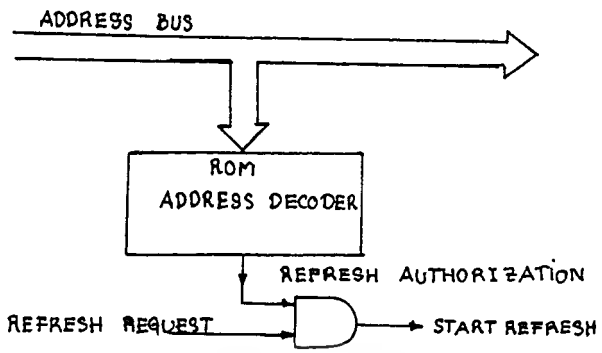
เช่น กรณีของการใช้ไมโครโปรเซสเซอร์ 8080 A ในการใช้คำสั่ง HALT RESET คำสั่ง WAIT นาน ๆ ในกรณีที่ใช้หน่วยความจำเร็วต่ำ หรือกรณีที่ทำงานแบบทีละจังหวะ และภาวะในระหว่างขอ DMA ทั้งนี้การระมัดระวังจะต้องรวมไปถึงการมีวงจรลอจิก ควบคุมให้มีการโอเวอร์ไรด์เหตุการณ์นี้ เพื่อให้การรีเฟรชหน่วยความจำทุก ๆ 2 มิลลิวินาทีตามเงื่อนไข

ในกรณีของไมโครโปรเซสเซอร์ 8080 A เราสามารถใช้จังหวะ T4 ของแมชชีนไซเคิล M1 ในการรีเฟรชหน่วยความจำได้ จึงไม่จำเป็นต้องใช้งานหน่วยความจำ รูปที่ 7.17 แสดงวงจรการรีเฟรชแบบซิงโครนัสของไมโครโปรเซสเซอร์ 8080 A ตามรูปจะมีวงจรนับสี่เป็นวงจรคอยตรวจสอบการสิ้นสุดของจังหวะ T3 เพื่อนำไปสร้างสัญญาณเริ่มต้นการรีเฟรช



รูปที่ 7.17 วงจรควบคุมการรีเฟรชแบบซิงโครนัสที่จังหวะ T4 ของแมชชีนไซเคิล M1 ของ 8080 A

เรายังสามารถรีเฟรชหน่วยความจำได้ขณะที่ใช้งาน ROM รูปที่ 7.18 แสดงวงจรควบคุมการรีเฟรชในขณะใช้งาน ROM โดยนำแอดเดรสมาถอดรหัส และตรวจสอบดูว่าเป็นแอดเดรสของ ROM หรือไม่ ถ้าเป็นก็สามารถนำไปสร้างสัญญาณเริ่มต้นการรีเฟรชได้ไมโครโปรเซสเซอร์แต่ละชนิด ตัวอย่างเช่น เราทราบว่าในขณะที่สัญญาณนาฬิกา 02 เปลี่ยนระดับจาก 0 เป็นระดับ 1 ตัว 8080 A ยังไม่มีความจำเป็นต้องใช้หน่วยความจำ เราก็สามารถใช้ช่วงเวลาการเปลี่ยนแปลงจากระดับ 0 เป็น 1 ของสัญญาณนาฬิกา 02 มากำหนดการรีเฟรชแบบไม่ซิงโครนัสได้ จึงตัดทอนความยุ่งยากของวงจรได้มาก



รูปที่ 7.18 วงจรควบคุมการรีเฟรชในขณะที่ใช้งาน ROM

RAM ที่จะยกเป็นตัวอย่างในหัวข้อนี้ เป็น RAM ชนิดสตาติค เนื่องจากไม่จำเป็นต้องมีวงจรรีเฟรช และวงจรกำหนดช่วงเวลาพิเศษที่ยุ่งยากนัก ใช้ได้ง่ายและมีระดับแรงดันของสัญญาณเข้าและออก เช่นเดียวกับของ TTL อีกทั้งเป็น IC มาตรฐานที่หาซื้อได้ไม่ยากนัก หรือถ้าหาซื้อไม่ได้ก็ยังพอใช้หลักการของ IC ตัวนี้ กับ IC ตัวอื่นที่มีขายในท้องตลาดได้ ในที่นี้จะไม่พูดถึง RAM ชนิดไดนามิคอีกต่อไปเพราะในภาคนี้เป็นการกล่าวนำทำความเข้าใจเกี่ยวกับการใช้งานของหน่วยความจำ เพื่อความสะดวกและง่ายแก่การอธิบายประกอบกันกับในระบบหน่วยความจำที่ไม่ใหญ่นัก โดยเฉพาะพวกซิงเกิลบอร์ดมักไม่นิยมใช้ RAM ชนิดไดนามิค ขอให้หาอ่านได้ในหนังสืออ้างอิงที่ให้ไว้ท้ายบทนี้

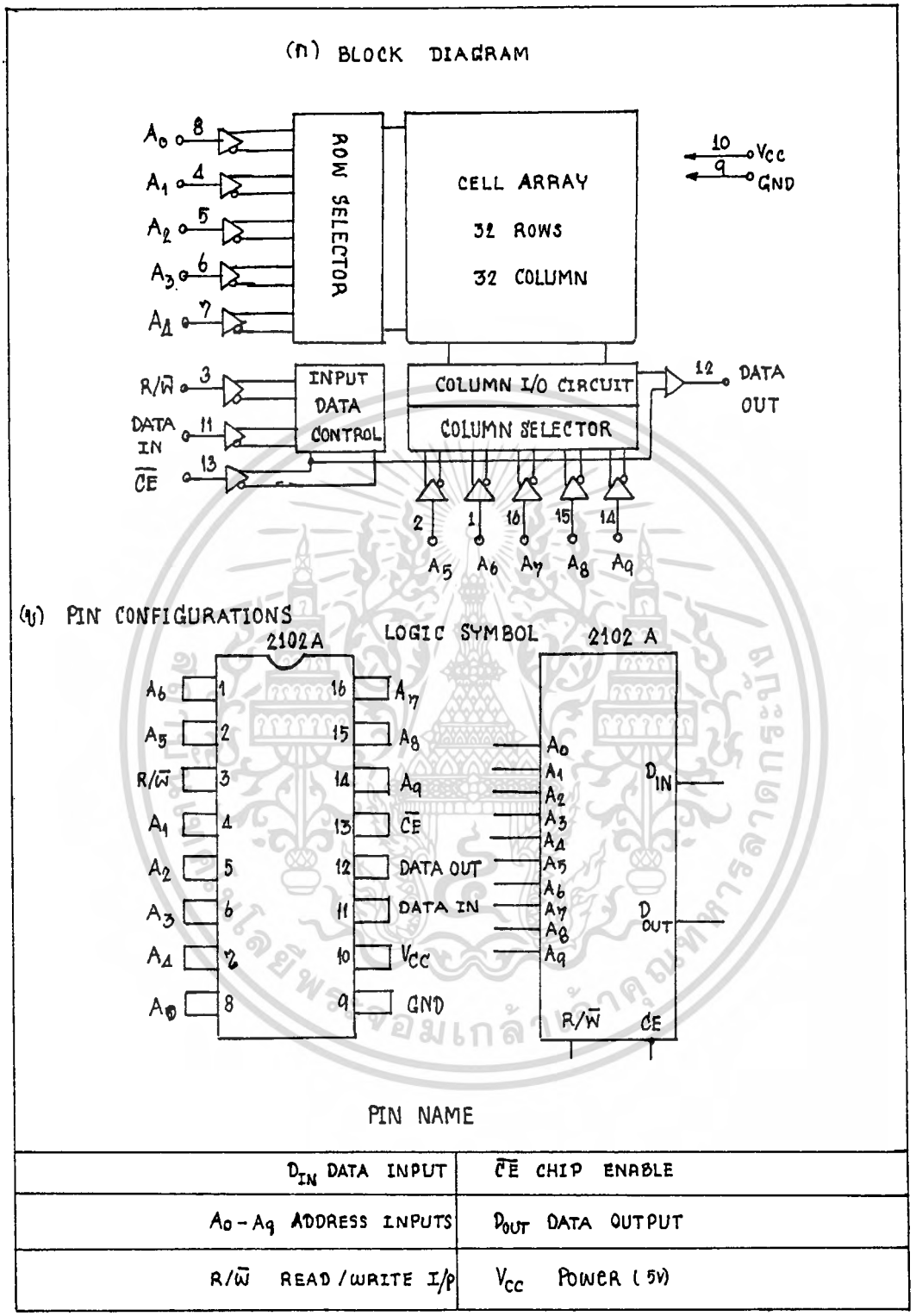
RAM ที่จะใช้ประกอบเป็นตัวอย่างในคำอธิบายต่อไปนี้ คือ 2102 ซึ่งเป็น RAM ชนิดสตาติค ขนาด 1Kx1 บิท ในลักษณะของเซลพื้นฐานของหน่วยความจำหน่วยนี้ก็คล้ายกับในรูปที่ 7.15ก. รูปที่ 7.19 แสดงบล็อคไดอะแกรมขาต่าง ๆ ของ IC และสัญญลักษณ์ทางตรรกของ 2102 นอกจากนี้ยังได้บอกชื่อของขาบางขาไว้ด้วย ขอให้ดูในบล็อคไดอะแกรมก่อน

รูปที่ 7.20 แสดงฐานของข้อมูลที่เขียนเข้าไป หรืออ่านออกมา และการควบคุมให้ RAM กลายสภาพเป็นอิมพีแดนซ์สูงข้อมูลเข้าที่ DATA IN ในขณะที่ขาเลือกคอลัมน์ ขาเขียน และขาชิพเอนเอเบิลเป็น 1 และเข้าบัฟเฟอร์โดยผ่านบัสข้อมูลเข้าภายใน ในขณะที่เลือกแถวเป็น 1 เซลหน่วยความจำจะเก็บข้อมูลเข้าภายใน ในขณะที่ขาเลือกแถวเป็น 1 ในการอ่านข้อมูลจากเซลหน่วยความจำ I/O "0" หรือสาย I/O "1" ในการอ่านข้อมูลจากเซลหน่วยความจำ มีวงจรขยายเซนซ์คอยตรวจดูว่าข้อมูลที่เก็บไว้นั้นเป็น 0 หรือ 1 แล้วส่งผ่านเกทที่ควบคุมด้วยสัญญาณชิพเอนเอเบิล สัญญาณอ่านและสัญญาณเลือกคอลัมน์ผ่านออกมาที่บัสข้อมูลออกภายใน และผ่านบัฟเฟอร์ออกมาที่สาย DATA OUT

บัฟเฟอร์สุดท้ายนี้ประกอบด้วย MOSFET 2 ตัว ซึ่งสามารถทำให้กระแสหรือหยุดนำกระแส โดยสัญญาณจากบัสข้อมูลออกภายในและสัญญาณชิพเอนเอเบิล ทำให้ MOSFET เปลี่ยนเป็นอิมพีแดนซ์ค่าสูง ทั้งหมดนี้ RAM แบบนี้ทำงานลักษณะ TRI STATE คือ โหมดอ่าน โหมดเขียน และโหมดอิมพีแดนซ์ค่าสูง

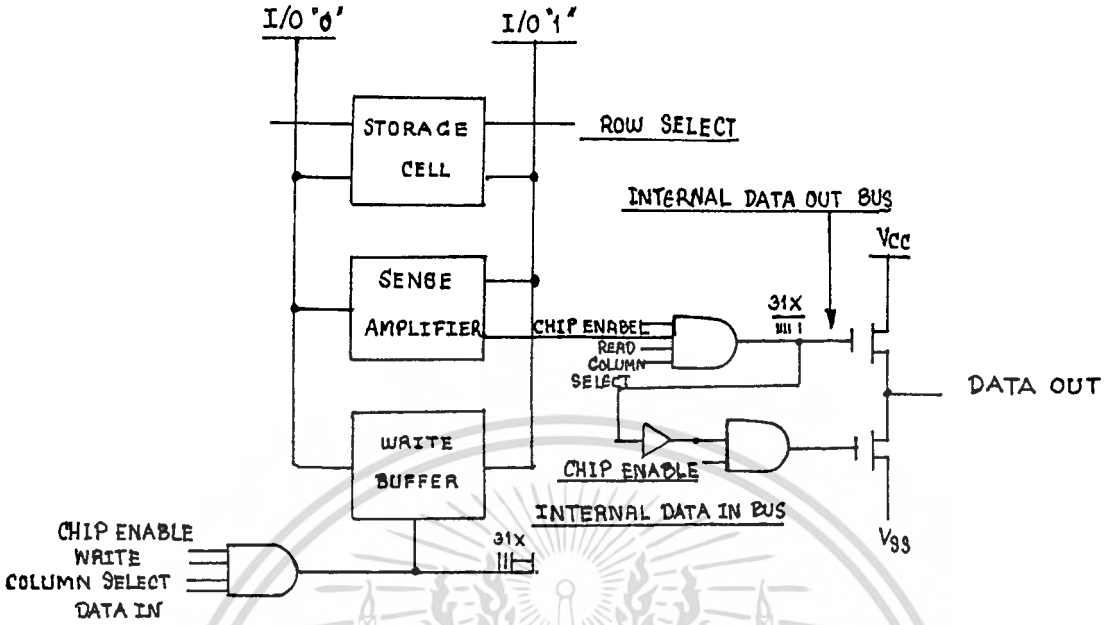
ในการอ่านหรือเขียนข้อมูลเข้าใน RAM นั้น สิ่งที่นับว่าสำคัญที่สุดก็คือ ช่วงเวลาสัมพันธ์ระหว่างสัญญาณต่าง ๆ ซึ่งจะต้องให้เหมาะสมกับ IC แต่ละเบอร์ สำหรับเบอร์ 2101 A นั้นมีรูปคลื่นสัญญาณและช่วงเวลาสัมพันธ์ระหว่างสัญญาณต่างๆ ในขณะที่ต้องการอ่านและที่ต้องการเขียนดังรูปที่ 7.21 โดยมีความหมายและขีดจำกัดของเวลาต่าง ๆ ดังปรากฏในรูปที่ 7.22

ตามรูปที่ปรากฏในรูปที่ 7.21ก. จังหวะอ่านในขณะที่ขา R/W อยู่ในสภาวะ 1 เราต้องให้สัญญาณแอดเดรสคงที่เป็นเวลา TRC ส่วนสัญญาณชิพเอนเอเบิล มีได้เป็น 2 ลักษณะ ลักษณะแรก เมื่อขั้วออกของ RAM ไม่ได้ต่อแบบ OR-Tie ขา CE อาจต่อกับ GND ได้โดยตรงทำให้ RAM ตัวนี้ทำงานตลอด ดังนั้นข้อมูลสามารถปรากฏที่ขั้วออกได้ภายในเวลา TA ลักษณะที่สอง เมื่อขั้วออกของ RAM ต่อแบบ OR-Tie อยู่ที่ขา CE สามารถเปลี่ยนสภาวะเป็น 0 เมื่อใดก็ได้ เพื่อให้ข้อมูลปรากฏที่ขั้วออก ถ้าหากมีการเปลี่ยนสภาวะของสัญญาณชิพเอนเอเบิลก่อนเลือกแอดเดรส หรือหลังจากเลือกแอดเดรสไปเป็นเวลาไม่เกิน TA-TCO ขั้วออกจะมีข้อมูลปรากฏภายในเวลา TCO นับจากเวลาที่มีการเปลี่ยนสภาวะของสัญญาณชิพเอนเอเบิล หากมีการเปลี่ยนแอดเดรสใหม่ ข้อมูลของแอดเดรสเดิมจะยังคงค้างอยู่เป็นเวลา TOH1 หรือในกรณีที่เปลี่ยนสภาวะของสัญญาณชิพเอนเอเบิลเป็น 1 ข้อมูลของแอดเดรสเดิมจะยังคงค้างอยู่เป็นเวลา TOH2 ดังนั้น ในการออกแบบระบบหน่วยความจำ เราต้องการอ่านข้อมูลออกมาจากหน่วยความจำ เราต้องคำนึงถึงช่วงเวลาที่รอ เพื่อให้ได้ข้อมูลออกมาหลังจากเลือกแอดเดรสและสัญญาณเอนเอเบิลแล้ว และช่วงเวลาที่ข้อมูลแอดเดรสยังคงค้างอยู่ด้วย



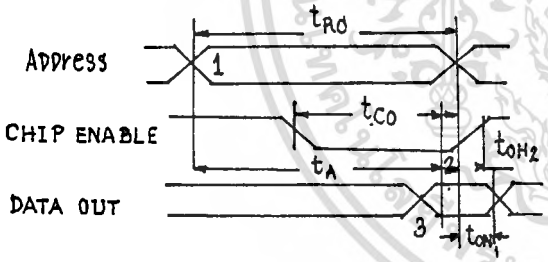
รูปที่ 7.19 ก. บล๊อคไดอะแกรมภายในของ 2102 A

ข. ชื่อสัญญาณที่ขาต่าง ๆ และสัญญลักษณ์ทางตรรกของ 2420 A



รูปที่ 7.20 ทางผ่านของข้อมูลใน RAM

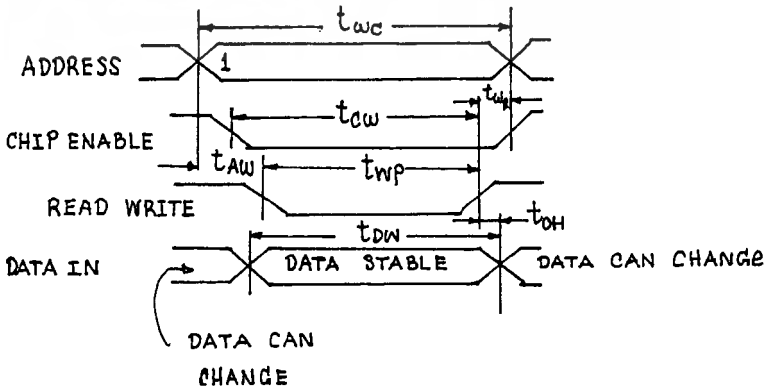
1 READ CYCLE



1 1.5 VOLTS
2 2.0 VOLTS
3 0.8 VOLTS

(ก) รูปคลื่นในขณะอ่าน 2102 A

2. WRITE CYCLE



(ข) รูปคลื่นในขณะเขียนของ 2102 A

1 1.5 Volts

READ CYCLE

Symbol	Parameter	Min	Typ[1]	Max	Unit
t_{RC}	Read Cycle	350			ns
t_A	Access Time			350	ns
t_{CO}	CHIP ENABLE to Output Time			180	ns
t_{OH1}	Previous Read Data Valid with Respect to Address	40			ns
t_{OH2}	Previous Read DATA Valid with Respest to CHIP ENABLE	0			ns

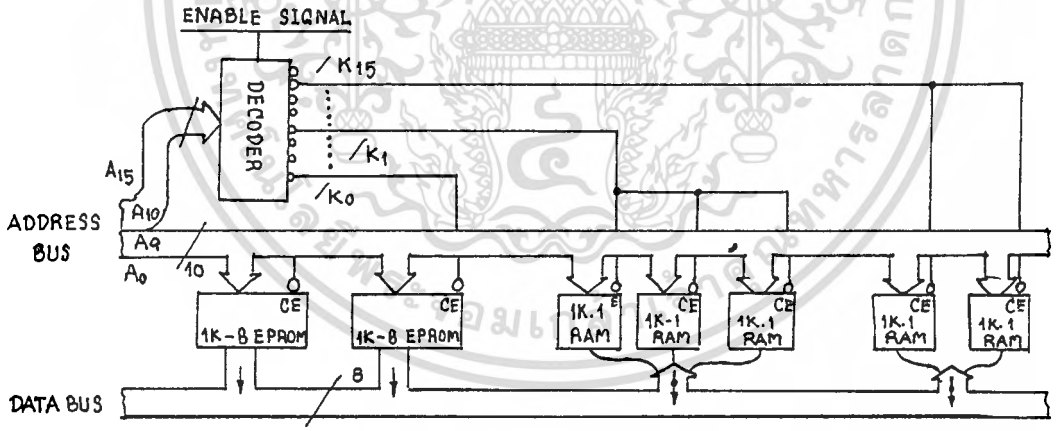
Symbol	Parameter	Min	Typ [1]	Max	Unit
t_{WC}	WRITE CYCLE	350			ns
t_{AW}	Address to Write Setup Time	20			ns
t_{WP}	WRITE PULSE Width	250			ns
t_{WR}	Write RECORVERY TIME	0			ns
t_{DW}	write DATA Setup TIME	250			ns
t_{DH}	DATA HOLD TIME	0			ns
t_{CW}	CHIP Enable to write Setup TIME	250			ns

NOTE : 1 Typical values are for T_A = 25°C and nominal Supply Voltage

รูปที่ 7.22 1. ข้อมูลเกี่ยวกับช่วงเวลาในจังหวะอ่านของ 2102 A

2. ข้อมูลเกี่ยวกับช่วงในจังหวะเขียนของ 2102 A

รูปที่ 7.23 แสดงการต่อระบบหน่วยความจำที่ใช้ EPROM ขนาด 1Kx8 บิท จำนวน 2 ตัวและ RAM ขนาด 1Kx1 บิท จำนวนหลาย ๆ ตัว ในบัสแอดเดรสมีทั้งหมด 16 สายจึงสามารถต่อกับหน่วยความจำได้ ทั้งนี้รวมทั้งของ EPROM และ RAM (ในที่นี้แต่ละเวิร์ดมี 8 บิท) ในบัสข้อมูลมีทั้งหมด 8 สาย ตามปกติ IC 1ตัว ไม่สามารถมีจำนวนเวิร์ดและจำนวนบิทได้มากตามต้องการ เช่นนี้จึงต้องใช้ IC หลายตัว และเนื่องจากข้อมูลจาก IC แต่ละตัวต้องมาปรากฏในบัสข้อมูล ซึ่งเป็นสายสัญญาณร่วม จึงนิยมใช้วิธีต่อขั้วออกเข้าด้วยกันในลักษณะ OR-TIE นั่นคือ O0 ของ IC ทุกตัวต่อถึงกันหมด O1 ก็เช่นกันจนถึง O7 ในการใช้งานเมื่อต้องการเข้าถึงเซลหน่วยความจำใน IC ตัวไหน เราเลือกให้ IC ตัวนั้นทำงาน โดยให้ขา CE ของ IC ตัวนั้นเป็นสภาวะ 0 ในขณะที่ตัวอื่นไม่ทำงาน (ขา CE เป็นสภาวะ 1) ข้อนี้ต้องระวังเป็นพิเศษ มิฉะนั้นข้อมูลที่ได้อาจผิดพลาด

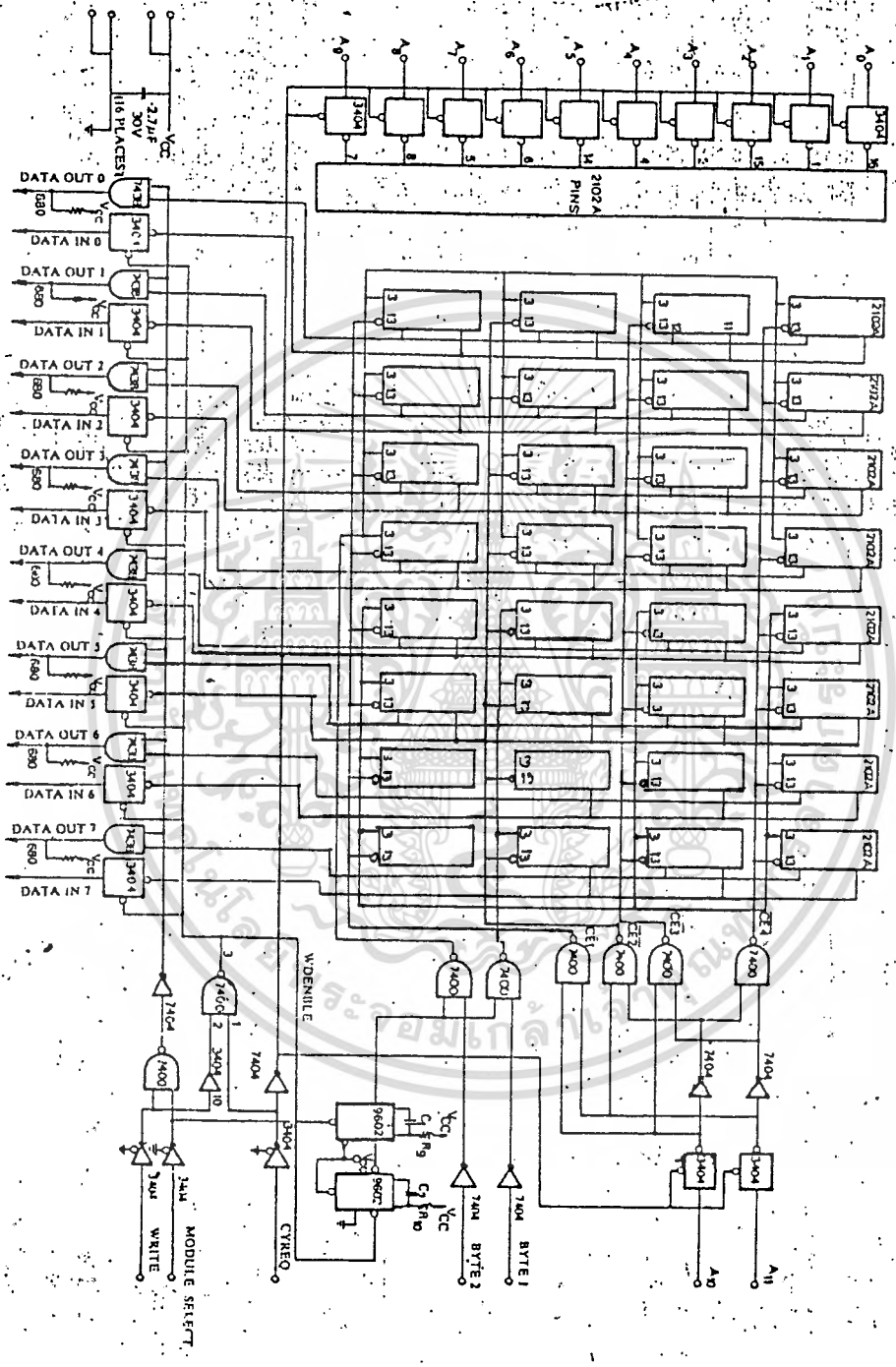


รูปที่ 7.23 ระบบหน่วยความจำขนาด 8 บิทที่ประกอบจาก EPROM (2706) และ RAM (2102 A)

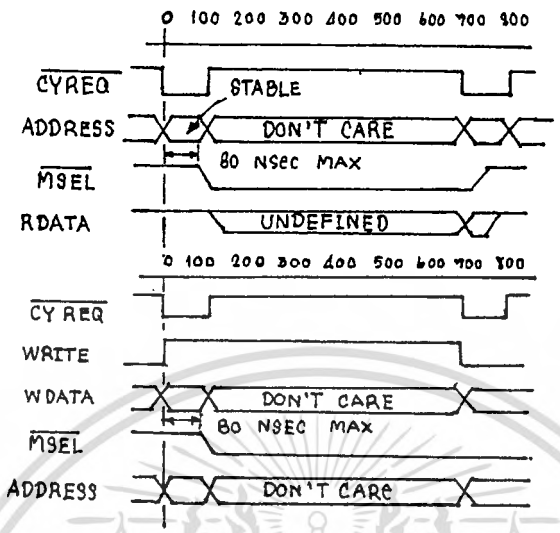
รูปที่ 7.24 เป็นระบบหน่วยความจำขนาด 4Kx8 โดยมีรูปคลื่นของสัญญาณต่าง ๆ ดังในรูปที่ 7.25 หน่วยความจำพื้นฐานที่ใช้ก็คือ 2102 A โดยมีแอดเดรส A0-A9 ผ่านแลทช์ความเร็วสูง (3404) มายังแอดเดรสทั้งหมดของ IC ทุกตัว เนื่องจากเป็นระบบหน่วยความจำขนาด 4K เวิร์ดจึงต้องมีวงจรถอดรหัสซึ่งประกอบด้วย 3404,7400 ทำหน้าที่แปลง A10-A11 เป็น 4 สาย วงจรถอดรหัสจาก 2 สายเป็น 4 สายมีขั้วเข้าและขั้วออกของข้อมูลแยกจากกัน โดยผ่าน 3404 และ 7438 ตามลำดับ วงจรดังในรูปที่ 7.24 นี้เรียกว่าเป็น 1 โมดุล อาจประกอบกันอยู่ในแผ่นวงจร 1 แผ่น วงจร 1 แผ่นในบางระบบอาจมีหน่วยความจำแบบนี้หลายแผ่น จึงต้องมีสายเลือกโมดุลเลือกว่าเป็นแผ่นวงจรใด มีสาย Byte 1 และ Byte 2 เลือกว่าจะทำอะไรกับข้อมูลใน 4 บิทต้นหรือ 4 บิทท้ายโดยมีการควบคุมอีกชั้นหนึ่ง จากสายสัญญาณเขียนซึ่งผาน 3404, 7400, 9602 (Retriggerable Monostable) และ 2 เกทสุดท้ายคือ 7400 ทั้งสองตัวก่อนที่จะเข้าไปยังขา R/W ของ 2102 A แต่ทั้งหมดนี้จะทำงานไม่ได้หากไม่มีสัญญาณ Cycle Request (สาย CYREQ)

ขอให้ดูรูปที่ 7.25 ที่เวลา T=0 หน่วยเวลาในรูปคิดนาโนวินาที ป้อนพัลซ์ CYREQ ขนาดกว้าง 100 นาโนวินาที เข้ามาขณะที่แอดเดรสอยู่คงที่ตามต้องการ และถูกเก็บในแทช (3404) แอดเดรสคงที่เป็นเวลา 100 นาโนวินาที สาย MSEL จะต้องเปลี่ยนคงที่แล้ว และในกรณีที่ต้องการเขียนข้อมูลเข้าไปในหน่วยความจำสายที่ป้อนสัญญาณเขียนจะต้องเป็น 1 เป็นเวลา 650 นาโนวินาทีหน่วยความจำข้อมูลจะปรากฏที่ขั้วออก ในเวลาไม่นานหลังจากที่สาย MSEL เปลี่ยนสภาวะเป็น 1 ในกรณีที่ใช้ผ่นวงจรหน่วยความจำเพียงแผ่นเดียว 1 โมดุล สายเลือกโมดุลอาจต่อกับ GND ได้เลย

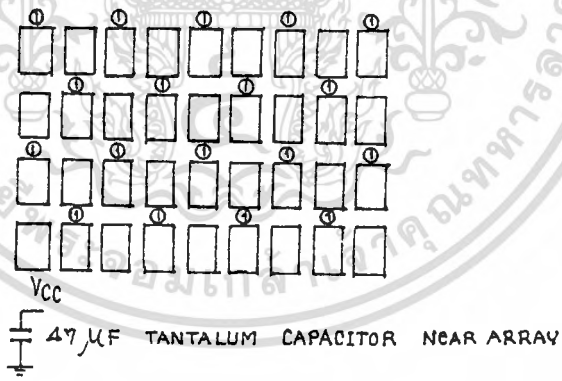
เพื่อป้องกันการรบกวนอันเนื่องมาจากการเชื่อมโยงผ่านทางสาย VCC และ GND จึงต้องมีการต่อตัวเก็บประจุขนาดเล็ก ลงในแผ่นวงจรหน่วยความจำด้วย เพื่อกรองสัญญาณรบกวนในลายจ่ายไฟตรง; เช่น การต่อตัวเก็บประจุขนาด .1 ลงในตำแหน่งที่หมายด้วยเลข (1) ในรูปที่ 26 เราเรียกว่าวิธีการนี้ว่า การคัปเปิลในระบบหน่วยความจำ



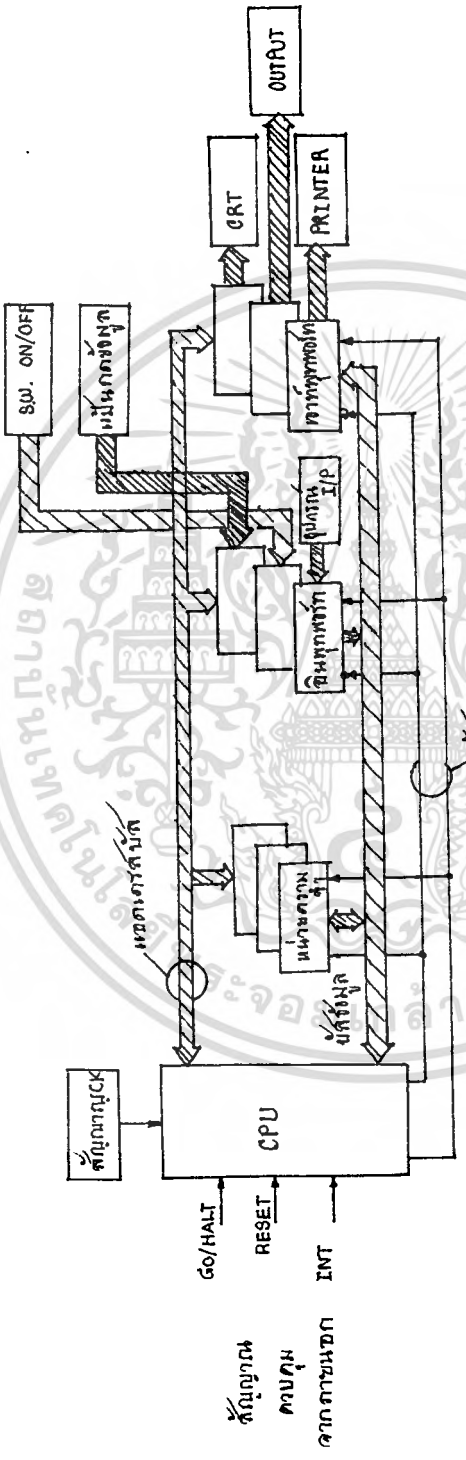
รูปที่ 7.24 ระบบหน่วยความจำขนาด 4K x 8 ทำจาก 2102 A



รูปที่ 7.25 รูปคลื่น และ ช่วงเวลาต่าง ๆ ที่เกี่ยวข้องในระบบหน่วยความจำขนาด 4K X 8



① 1 μF CERAMIC CAPACITOR EVERY OTHER DEVICE AS SHOW



รูปที่ 8.1 การต่ออุปกรณ์ภายนอกกับ CPU ของไมโครโปรเซสเซอร์

อินพุท เอาท์พุท อินเตอร์รัพท์ และ DAM

การทำงานต่าง ๆ ในระบบคอมพิวเตอร์ จะเป็นไปอย่างสมบูรณ์ได้จำเป็นต้องมีขั้นตอนเกี่ยวกับการโอนย้าย ระหว่าง CPU กับอุปกรณ์ภายนอกประกอบอยู่ด้วย ถ้าเราสามารถส่งข่าวสารหรือข้อมูลเข้าไปในระบบได้แล้วต่อไปเราจะจัดการกับข่าวสาร หรือข้อมูลนั้นอย่างไรก็สามารถทำได้ตามความต้องการ อาจส่งข่าวสารหรือข้อมูลระหว่างอุปกรณ์ต่าง ๆ กัน โดยจะกล่าวถึงโครงสร้างและวิธีการอินพุทและการเอาท์พุทขั้นพื้นฐาน เทคนิคการโพลลิ่ง ไปจนถึงการอินพุท เอาท์พุทด้วยวิธีการอินเตอร์รัพท์ และ DAM

เรื่องของอินพุท เอาท์พุท ถ้าจะให้เข้าใจอย่างถ่องแท้จำเป็นต้องรู้ว่าระบบคอมพิวเตอร์ทั่วไป มีโครงสร้างของระบบอย่างกว้าง ๆ ก่อน แล้วจึงมาดูที่รายละเอียดอีกครั้งหนึ่ง

ในรูป 8.1 แสดงถึงการต่ออุปกรณ์ต่าง ๆ เข้ากับ CPU การส่งถ่ายข้อมูลนั้น จะผ่านบัสที่ใช้ร่วมกันทั้ง CPU และหน่วยความจำ การส่งถ่ายข้อมูลจากอุปกรณ์ภายนอกเข้าไปบัสข้อมูล CPU จะมี I/O พอร์ท (I/O Port) หรือ I/O อินเตอร์เฟส (I/O Interface) เป็นตัวกลาง มีแอดเดรสบัสเป็นตัวเลือกพอร์ทและบัสควบคุมจะส่งสัญญาณมาควบคุมการส่งถ่ายข้อมูล การที่จะส่งข้อมูลเข้าหรือออกที่อุปกรณ์ตัวใด ทำโดยใช้คำสั่งอินพุท และเอาท์พุทข้อมูล เช่น IN หรือ OUT ตามปกติคำสั่งอินพุทหรือเอาท์พุทจะมีส่วนหนึ่งของคำสั่งสำหรับบอกหมายเลขอุปกรณ์ที่จะติดต่อกับ CPU หมายเลขนี้จะถูกถอดรหัสโดยส่วนควบคุมการเลือก เพื่อเลือกอุปกรณ์ที่จะติดต่อ การส่งหมายเลขอาจใช้แอดเดรสเป็นตัวเลือกอุปกรณ์โดยตรง ได้ตัวอย่างของคำสั่งจะเป็นดังนี้

IN 03 (ACC) (ข้อมูลจากอุปกรณ์ 03)

คำสั่งนี้เป็นการนำข้อมูลจากอุปกรณ์ผ่าน I/O พอร์ทหมายเลข 03 เข้าเก็บในแอคคิวมูเลเตอร์ของ CPU ข้อมูลที่ผ่านจากพอร์ทนี้จะมีผลใช้ทำอะไรต่อไปขึ้นอยู่กับคำสั่งต่าง ๆ ที่ตามมา

I/O พอร์ท

อุปกรณ์ภายนอกที่ต้องการส่งถ่ายข้อมูล CPU จะต้องอาศัยพอร์ทเป็นตัวกลาง โดยพอร์ทจะทำหน้าที่เป็นช่องทางส่งผ่านข้อมูลสำหรับอุปกรณ์แต่ละตัว ในระบบคอมพิวเตอร์หนึ่ง ๆ อาจมีหลาย ๆ พอร์ท เพื่อเป็นช่องทางส่งผ่านข้อมูลของอุปกรณ์หลาย ๆ ชนิด

I/O พอร์ทจะประกอบด้วยวงจรที่สามารถเลือกเมื่อได้รับซิกนิฟิคันท์คำสั่งเกี่ยวกับอินพุทและเอาท์พุท I/O พอร์ทมีทั้งสำหรับเป็นอินพุทพอร์ทและเอาท์พุทพอร์ท อินพุทพอร์ทเป็นช่องทางสำหรับนำข้อมูลเข้า CPU เอาท์พุทพอร์ทเป็นช่องทางสำหรับนำข้อมูลออกจาก CPU ส่งให้อุปกรณ์ภายนอก ปกติพอร์ท I/O พอร์ทมีแบบส่งถ่ายข้อมูล 2 แบบ คือ การส่งถ่ายข้อมูลแบบขนาน และการส่งถ่ายข้อมูลแบบต่อเนื่อง

ด้านหนึ่งของพอร์ทจะต่อกับบัสข้อมูลของ CPU ส่วนอีกด้านสำหรับต่อเข้ากับอุปกรณ์ภายนอก ถ้าเป็นอินพุทพอร์ทด้านสำหรับที่ต่อเข้ากับอุปกรณ์ CPU เป็นลอจิกเกทพวกไตรสเตทบัฟเฟอร์ คือ เมื่อพอร์ทไม่ถูกเลือกให้ทำงานจะมีลักษณะเหมือนปลดสายออกจากบัสข้อมูลของระบบ มีคุณสมบัติเป็นอิมพีแดนซ์สูง

I/O พอร์ทแบบขนาน

I/O พอร์ทสำหรับการส่งถ่ายข้อมูลแบบขนาน เป็นไอซีจำพวก LSI ภายในเป็นวงจรลอจิกประกอบกันขึ้นอยู่ในตัวเดียวกัน ส่วนประกอบที่สำคัญ ๆ ดังนี้คือ

1. บัสบัฟเฟอร์ (Bus Buffer) คือ ช่องทางส่งผ่านข้อมูลประกอบด้วยลอจิกเกทประเภทไตรสเตทบัฟเฟอร์ โดยทั่วไปจะมีสายอินพุทและเอาท์พุท 8 สาย ใช้ในการส่งผ่านข้อมูลแบบขนาน 8 บิท พอร์ทนี้อาจใช้สำหรับทั้งอินพุทและเอาท์พุท โดยด้านที่ต่อกับบัสข้อมูลของ CPU จะต้องมีคุณสมบัติเป็นบัสสองทิศทาง (Bi-directional Bus)

2. การกำหนดแอดเดรสของ I/O พอร์ทแต่ละตัวจะมีสายสำหรับกำหนดหมายเลขของพอร์ท ซึ่งใช้ในการเรียกพอร์ท ปกติจะมีสายกำหนด 1 สาย หรือมากกว่า อุปกรณ์ที่ต่อเข้ากับพอร์ทแต่ละพอร์ทเมื่อต้องการให้ทำงานก็ใช้หมายเลขที่กำหนดไว้ที่พอร์ทเป็นรหัสสำหรับเลือก

3. บัสควบคุมและไทม์มิงค์ของสัญญาณ I/O พอร์ทมีสายสำหรับควบคุมการทำงานอีกหลายสาย ซึ่งจะควบคุมการทำงาน เช่น การส่งข้อมูลผ่านเข้าออก และการเคลียร์ การเลทช์ข้อมูล นอกจากนี้ยังต้องคำนึงถึงไทม์มิงค์ของสัญญาณต่าง ๆ ในการทำงานด้วย คือช่วงสัญญาณควบคุมอะไรมาก่อนหลัง ข้อมูลที่ส่งรับทำในช่วงใดของสัญญาณ สิ่งต่าง ๆ เหล่านี้จำเป็นต้องคำนึงถึงในการออกแบบ

ตัวอย่างในรูปที่ 8.2 เป็น I/O พอร์ทแบบขนาน (Parallel I/O port) ขนาด 8 บิท อินพุท เอาท์พุท ประกอบขึ้นเป็น [illegible] ผ่านข้อมูลแยกเป็น 2 ส่วน คือ ด้านรับข้อมูลเข้า D10-D17 ด้านส่งข้อมูลออก D00-D07 ด้านรับข้อมูลเข้าประกอบด้วย 8 บิทแลทช์ latch เป็นพวก D ฟลิปฟลอป์ ด้านส่งข้อมูลประกอบด้วย 8 บิทเอาท์พุทบัฟเฟอร์ เป็นพวกไตรสเตทบัฟเฟอร์ นอกจากช่องทางส่งผ่านข้อมูลแล้ว ยังมีสายควบคุมซึ่งใช้ควบคุมการเคลื่อนย้ายข้อมูลต่าง ๆ อีก เช่น mode select (MD) strobe (STB) และ device select logic DS 1, DS 2 เบอร์ 3212 นี้สำหรับใช้เป็นอินพุทหรือเอาท์พุทพอร์ทก็จะต่อต่อในทางตรงข้าม

ในกรณีของเอาท์พุทโหมด ขา STB จะไม่มีผลต่อการส่งถ่ายข้อมูล การทำงานจะเป็นไปตามไทม์มิงในรูป 8.3 สัญญาณ DS1 และ DS2 จะสโตรป (Strobe) ข้อมูลใน 8 บิทแลทช์ การทำงานจะเป็นไปตามไทม์มิงค์ ในรูป 8.4

I/O พอร์ทแบบอนุกรม (Serial I/O Port)

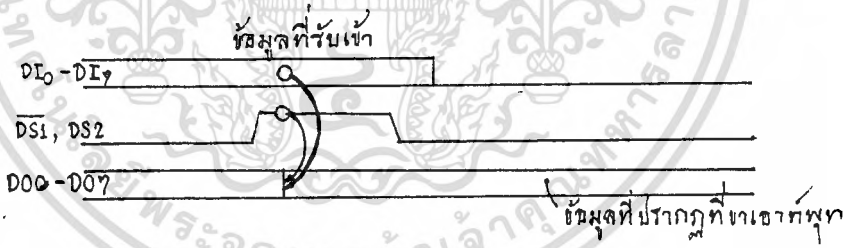
I/O พอร์ทที่ใช้ในการส่งถ่ายข้อมูลแบบต่อเนื่องก็เป็น IC จำพวก LSI เช่นเดียวกัน มีวงจรต่าง ๆ ประกอบกันไว้ใน IC ตัวเดียว สำหรับถ่ายข้อมูลกับอุปกรณ์ I/O

USART (UNIVERSAL SYNCHRONOUS ASYNCHRONOUR RECEIVER/TRANSMITTER) และ UART (UNIVERSAL ASYNCHRONOUR RECEIVER/TRANSMITTER) ไอซีที่ใช้ในการรับส่งข้อมูลแบบต่อเนื่อง มีส่วนประกอบที่สำคัญ ๆ ดังนี้คือ

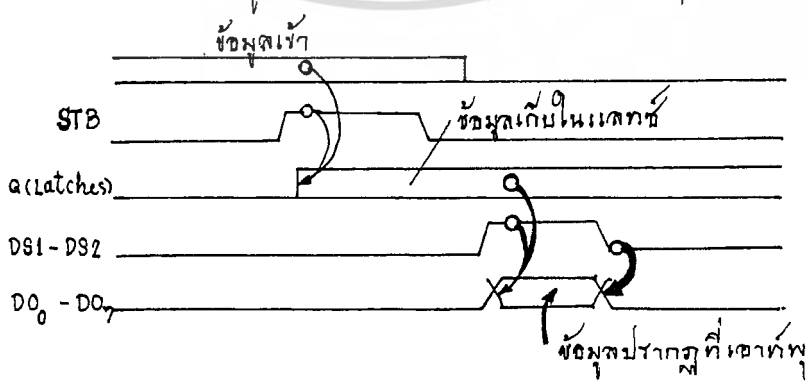
1. วงจรส่งข้อมูลออกแบบอนุกรมและรับข้อมูลเข้าแบบอนุกรม ประกอบด้วยบัฟเฟอร์ด้านรับและส่ง ลิฟท์รีจิสเตอร์ สำหรับรับและส่งแบบอนุกรม

2. การกำหนดแอดเดรสสำหรับและส่งจะมีข้อกำหนดการทำงานของพอร์ท โดยรับสัญญาณจากบัสแอดเดรสหรือตัวถอดรหัสแอดเดรส

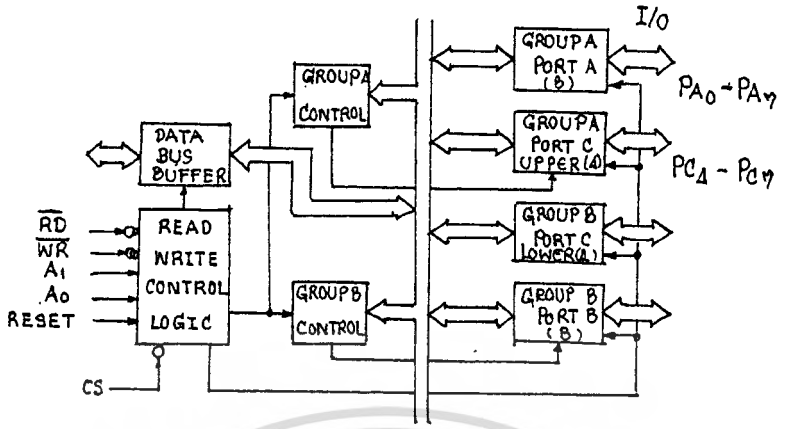
3. บัสควบคุมพอร์ทรับส่งข้อมูลแบบอนุกรม จะมีสายสำหรับควบคุมทั้งด้านรับและส่งมีการควบคุมรูปแบบของสัญญาณที่ส่งออก เช่น การกำหนดจำนวนสตอปบิท จำนวนบิทข้อมูล จำนวนบิทพาริตี้ นอกจากนี้ยังควบคุมการตรวจสอบพาริตี้ของสัญญาณที่รับเข้าว่ามีการผิดพลาดหรือไม่ด้วย



รูปที่ 8.3 ไทมิงค์เมื่อทำงานเอาท์พุทโหมด



รูปที่ 8.4 ไทมิงค์เมื่อทำงานในอินพุทโหมด



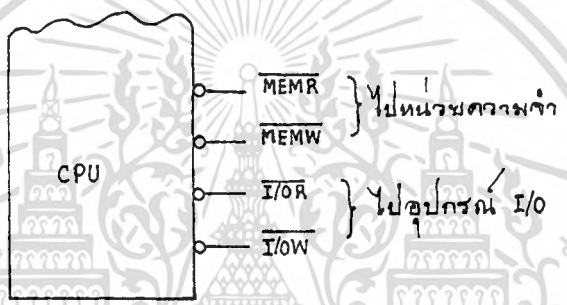
รูปที่ 8.5 พอร์ทเบอร์ 8255

ตารางที่ 8.1 การเลือกวิธีทำงานของ 8255

CS	A1	A0	RD	WR	OPERATION	
0	0	0	0	1	PORT A TO DATA BUS	MPU READ (A.B.C)
0	0	1	0	1	PORT B TO DATA BUS	
0	1	0	0	1	PORT C TO DATA BUS	
0	0	0	1	0	DATA BUS TO PORT A	MPU WRITE
0	0	1	1	0	DATA BUS TO PORT B	
0	1	0	1	0	DATA BUS TO PORT C	
0	1	1	1	0	DATA BUS TO CONTROL	
0	1	1	0	1	ILLEGAL	
1	-	-	-	-	DATA BUS TO 3-STATE (DISABLE)	

สัญญาณควบคุมการทำงานของพอร์ทและการกำหนดแอดเดรส

รับส่งข้อมูลโดยใช้คำสั่งในไมโครโปรเซสเซอร์ ปกติจะมีคำสั่งเฉพาะสำหรับการรับและส่งข้อมูล CPU ทำการเอ็กซีคิวท์คำสั่งอินพุท เอาท์พุท จะมีสัญญาณควบคุมออกจากระบบควบคุมโดยตรง สัญญาณนี้เรียกว่า สัญญาณ Isolated I/O นอกจากนี้ยังมีวิธีการใช้คำสั่ง เมื่อติดต่อกับอุปกรณ์ I/O อีกวิธี คือใช้คำสั่งเกี่ยวกับการอ่านเขียนหน่วยความจำ ซึ่ง CPU จะให้สัญญาณควบคุมการอ่านและเขียนข้อมูลที่หน่วยความจำ สัญญาณนี้สามารถนำมาดัดแปลงใช้เป็นสัญญาณควบคุมอุปกรณ์ I O ได้ เราเรียกสัญญาณนี้ว่า Memory Mapped I/O



รูปที่ 8.6 Isolated I/O

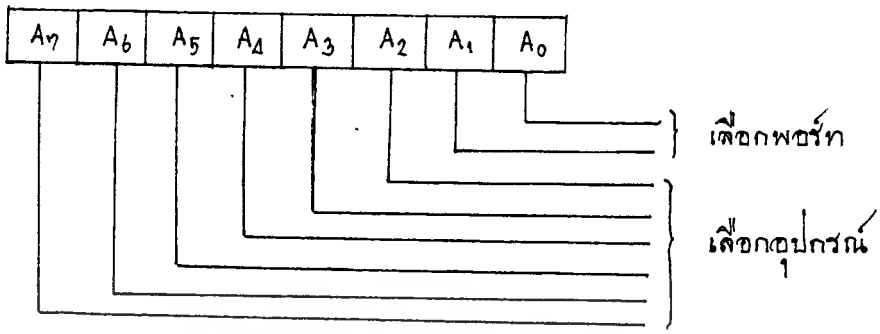
ตัวอย่างในรูปที่ 8.7 เป็นการใช้แอดเดรสบิทควบคุมการเลือกอุปกรณ์โดยตรง เมื่อใช้วิธีนี้ I/O พอร์ท เบอร์ 8255 ใช้ 4 บิทแอดเดรส (A1-A4) เป็นตัวเลือกรีจิสเตอร์ และใช้ 2 แอดเดรส A1,A0 เป็นตัวเลือกพอร์ท

Memory Mapped I/O

การส่งถ่ายข้อมูลระหว่างอุปกรณ์ I/O กับ CPU ด้วย Memory Mapped I/O มีข้อดี คือ เมื่อต้องการส่งถ่ายข้อมูลจำเป็นต้องผ่านรีจิสเตอร์ต่าง ๆ ได้โดยตรง ไม่จำเป็นต้องผ่านแอคคิวมูเลเตอร์ก่อนเหมือนวิธี Isolated I O ทำให้สามารถลดเวลาในการส่งถ่ายข้อมูลลงได้ การทำงานจะทำขณะที่เอ็กซีคิวท์คำสั่ง เช่นเดียวกับคำสั่ง IN หรือ OUT โดยใช้คำสั่งสูงสุด (A15) จะไม่ถูกใช้ เราสามารถนำมาเป็น I/O แฝงได้โดยกำหนดดังนี้

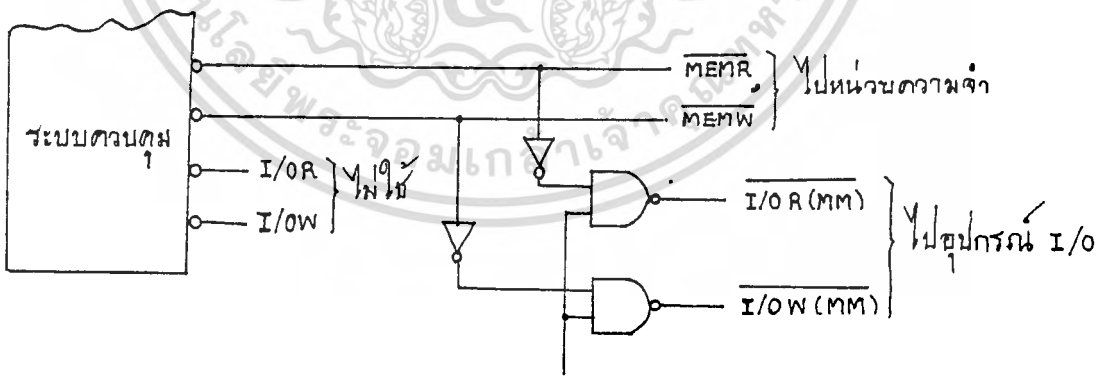
ถ้า A15 เป็น 0 สำหรับการทำงานของหน่วยความจำ

ถ้า A15 เป็น 1 สำหรับการทำงานของ I/O



รูปที่ 8.7 การถอดรหัสแอดเดรสสำหรับ 8255

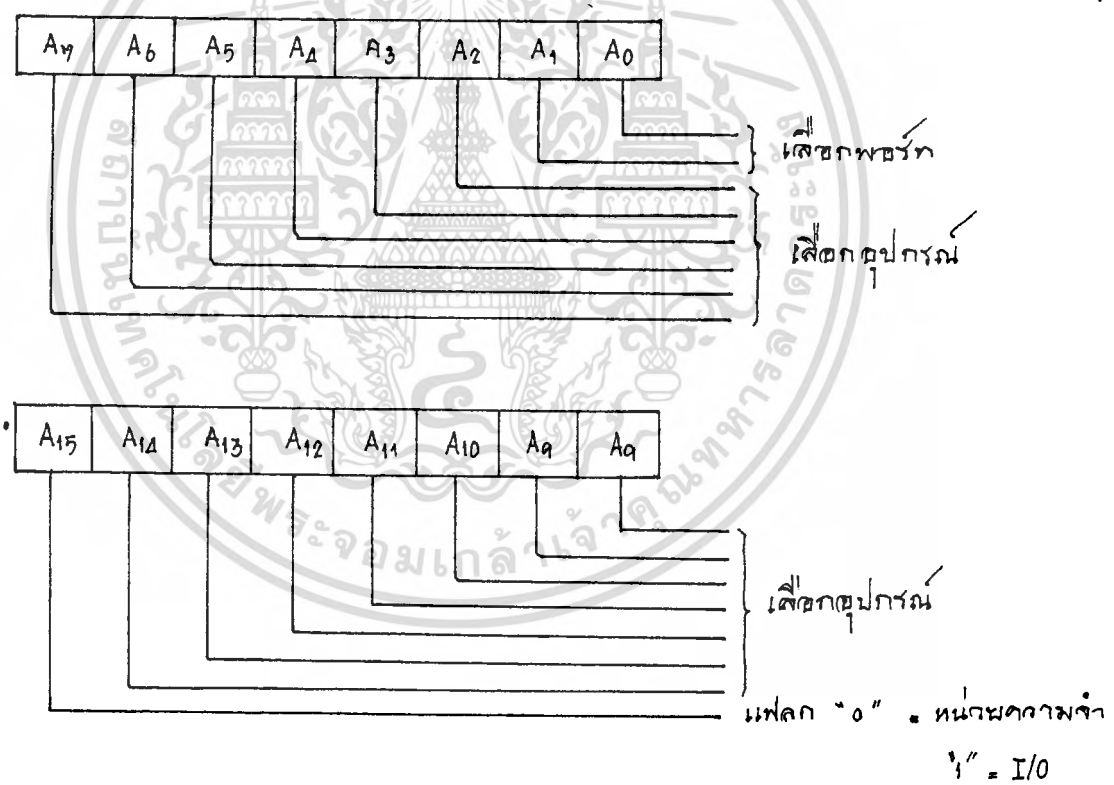
ในรูปที่ 8.8 สัญญาณควบคุมถูกสร้างขึ้นใหม่โดยใช้สัญญาลักษณ์ MEMR และ MEMW ร่วมกับแอดเดรสบิท 15 เป็นสัญญาณควบคุมอุปกรณ์ I/O ใหม่ ซึ่งจะใช้แทนสัญญาณ Isolated I/O



รูปที่ 8.8 Memory mapped I/O

ตัวอย่าง "หน่วยความจำ", Memory Mapped I/O

MOV	r, M	(อินพุทจากพอร์ทเข้ารีจิสเตอร์ต่าง ๆ)
MOV	M, r	(เอาท์พุทจากรีจิสเตอร์ไปยังพอร์ท)
MVI	M	(เอาท์พุทค่าไบท์ที่ตามมาไปยังพอร์ท)
LDA		(อินพุทจากพอร์ทเข้าแอคคิวมูเลเตอร์)
STA		(เอาท์พุทค่าในแอคคิวมูเลเตอร์ออกพอร์ท)
LHLD		(16 บิท)



รูปที่ 8.9 การถอดรหัสแอดเดรสสำหรับ 8255

การสร้างสัญญาณเลือกอุปกรณ์โดยใช้ตัวถอดรหัส

การสร้างสัญญาณเลือกอุปกรณ์โดยใช้ตัวถอดรหัส Decoder เพื่อถอดรหัสแอดเดรส ซึ่งเป็นรหัสไบนารี่ให้เป็นสัญญาณออกมาหลาย ๆ ตัว เพื่อเลือกอุปกรณ์ตามต้องการ ในที่นี้จะยกตัวอย่างโดยใช้ไอซีเบอร์ 74154 ซึ่งเป็นตัวถอดรหัส 4 สายเป็น 16 สาย 4 to 16 line decoder เป็นวงจรง่าย ๆ แสดงไว้ในรูป 8.10 ซึ่งใช้ 4 บิทต่ำสุด (A3-A0) ของแอดเดรสร่วมกับสัญญาณ IN หรือ OUT โดยที่สัญญาณ IN หรือ OUT จะเป็นสัญญาณสโตรบให้กับตัว 74154 ที่ขา G1 ส่วน G2 ต่อกับกราวน์หรือลอจิก "0" ไว้

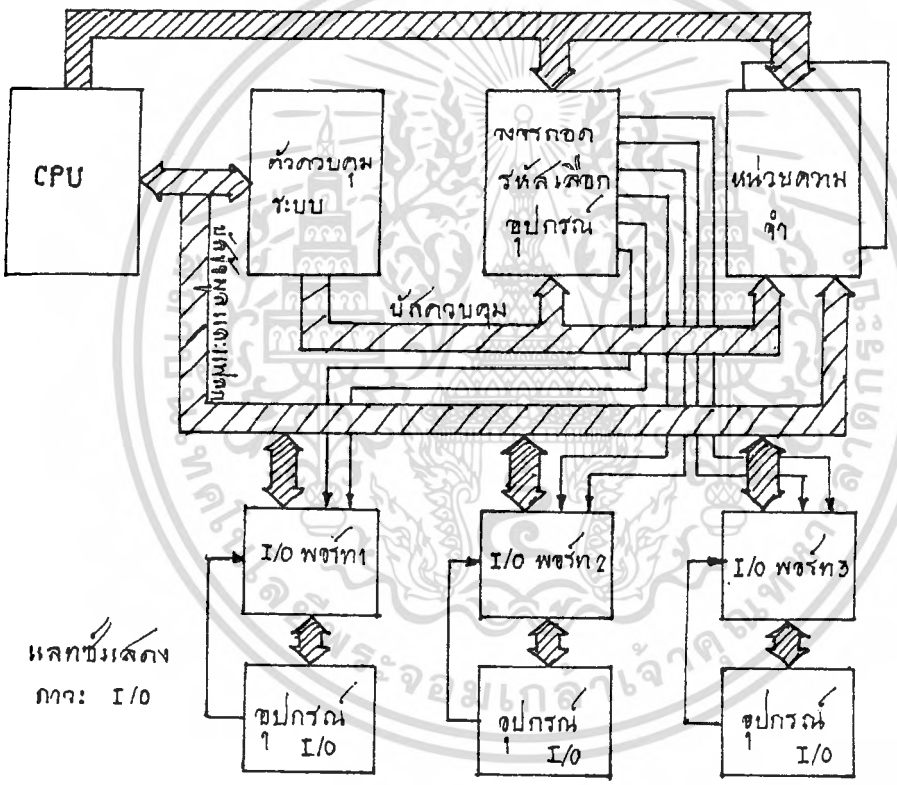
รูปที่ 8.10 การสร้างสัญญาณเพื่อเลือกอุปกรณ์ 16 ตัว

วิธีโพลลิ่งหรือโปรแกรม

ในแบบของการโพลลิ่งหรือโปรแกรม I/O การส่งถ่ายข้อมูลของอุปกรณ์ทำโดยใช้โปรแกรม ไมโครโปรเซสเซอร์จะส่งหรือรับข้อมูลต่างๆ ทุกครั้ง อยู่ภายใต้การควบคุมของโปรแกรมที่กำลังจะทำงานอยู่นั้น การส่งถ่ายข้อมูลจำเป็นต้องทำร่วมกับการตรวจสอบสเตัสซึ่งกันและกัน ระหว่าง CPU กับอุปกรณ์โดยใช้วิธี แฮนเช็คกิ้ง วิธีการขั้นมีพื้นฐาน ได้แก่ การตรวจเช็คความต้องการส่งข้อมูล ความพร้อมที่จะรับข้อมูล โดยการใช้แฟลกต่าง ๆ สำหรับการทำอะไร ตัวอย่างแฟลกที่แสดง device ready ถ้าเป็นอุปกรณ์เอาท์พุทก็จะบอกว่าบัฟเฟอร์ว่างพร้อมกับข้อมูล เป็นต้น

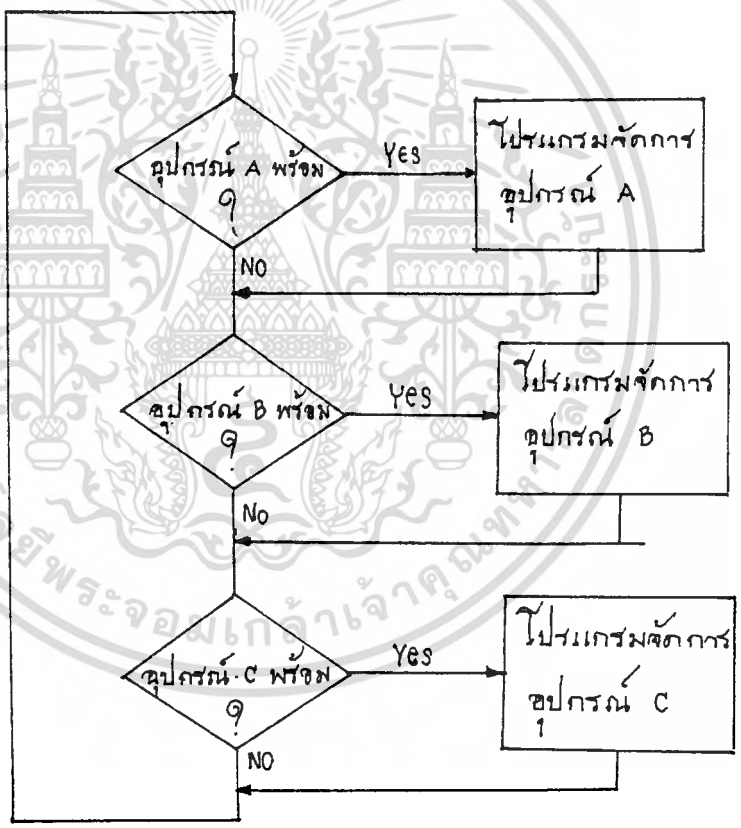
ในวิธีการของโพลลิ่ง เมื่อ CPU ต้องการติดต่อกับอุปกรณ์ภายนอก จะใช้การตรวจเช็คแฟลกทีนอร์ทตาง ๆ โดยเริ่มด้วยการอ่านภาวะของแฟลกทุกครั้งก่อน แล้วจึงเข้าสู่โปรแกรมที่จะจัดการกับแต่ละอุปกรณ์อีกทีหนึ่ง จะเห็นว่าที่อุปกรณ์แต่ละตัวจะมีสัญญาณชี้แสดงอยู่ เพื่อแสดงสภาวะว่าอุปกรณ์นั้นพร้อม (Reay)

หรือไม่ก็โดยการส่งสัญญาณสโตรปจากวงจรถอดรหัสเลือกอุปกรณ์ไปควบคุมการอ่านอีกทีหนึ่ง รูป 8.11 แสดงโครงสร้างระบบการโอนย้ายข้อมูลในลักษณะนี้



รูปที่ 8.11 ระบบวงจรที่ใช้วิธีโพลลิ่ง

ผังงานของโพลลิ่งแสดงไว้ในรูป 8.12 มีโปรแกรมจัดการกับอุปกรณ์ A, B, C โดยเฉพาะการทำงานจะเป็นลักษณะวนตรวจสอบภาวะของแฟลกที่อุปกรณ์ต่าง ๆ อยู่ตลอดเวลา เริ่มด้วยการตรวจสอบแฟลกของอุปกรณ์ A ถ้าพร้อมก็จะเข้าสู่โปรแกรมการจัดอุปกรณ์ B ซึ่งถ้าพร้อมก็จะเข้าไปทำโปรแกรมจัดการอุปกรณ์ B และตรวจสอบต่อมาจนถึง C แล้วจึงวนกลับไปที่อุปกรณ์ A โปรแกรมจัดการอุปกรณ์และความต้องการอุปกรณ์ของผู้ใช้ที่จะเขียนขึ้นโดยอาจเป็นการเรียกเพื่อนำข้อมูลเข้าคำนวณเก็บหรือแสดงผล



รูปที่ 8.12 แผนภูมิแสดงวิธีการโพลลิ่ง

อินเตอร์รัพท์ (Interrupt)

การที่ CPU จะติดต่อกับอุปกรณ์ภายนอกได้นั้นมีหลายวิธีด้วยกัน ในหลาย ๆ วิธีเหล่านี้การอินเตอร์รัพท์ก็เป็นวิธีหนึ่งที่ใช้กันอย่างกว้างขวาง

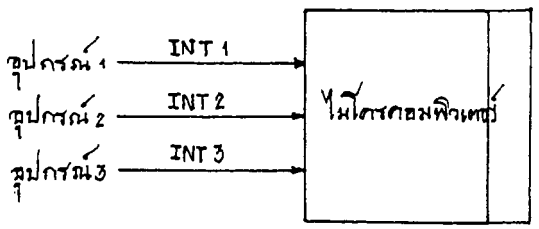
อินเตอร์รัพท์ หมายถึง การขัดจังหวะการทำงานของ CPU เพื่อให้ CPU หยุดการทำงานของโปรแกรมที่กำลังทำอยู่ เมื่อได้รับสัญญาณอินเตอร์รัพท์จากอุปกรณ์ภายนอก CPU จึงไปบริการให้กับงานที่ส่งสัญญาณอินเตอร์รัพท์เข้ามา เมื่อบริการงานที่อินเตอร์รัพท์เสร็จแล้วก็จะกลับไปทำงานที่โปรแกรมเดิมต่อไป การอินเตอร์รัพท์จะกระโดยไปก็ต่อเมื่อมีคำสั่ง JUMP แต่การอินเตอร์รัพท์จะกระโดดไปทำงานในอีกโปรแกรมหนึ่ง เมื่อ CPU ได้รับสัญญาณอินเตอร์รัพท์จากอุปกรณ์

ชนิดอินเตอร์รัพท์มีอยู่ 3 ชนิด คือ

ก. อินเตอร์รัพท์สายเดี่ยว (Single Interrupt) การอินเตอร์รัพท์แบบนี้ เมื่อ CPU ได้รับสัญญาณอินเตอร์รัพท์ CPU จะต้องกวาดดูว่าอุปกรณ์ตัวใดเป็นตัวส่งสัญญาณอินเตอร์รัพท์รีเควสท์เข้ามา การกวาดแบบนี้ CPU จะใช้วิธีการที่เรียกว่า โพลลิ่ง คือ การเควสท์ขอตัวใดแล้วก็จะจัดบริการให้กับอุปกรณ์ตัวนั้นได้อย่างถูกต้อง วิธีนี้นิยมกันมากในระบบไมโครโปรเซสเซอร์ เพราะสามารถต่ออุปกรณ์ได้มากโดยไม่มีขีดจำกัด

รูป 8.13 การอินเตอร์รัพท์สายเดียว

ข. อินเตอร์รัพท์หลายระดับ (Mutilevel Interrupt)

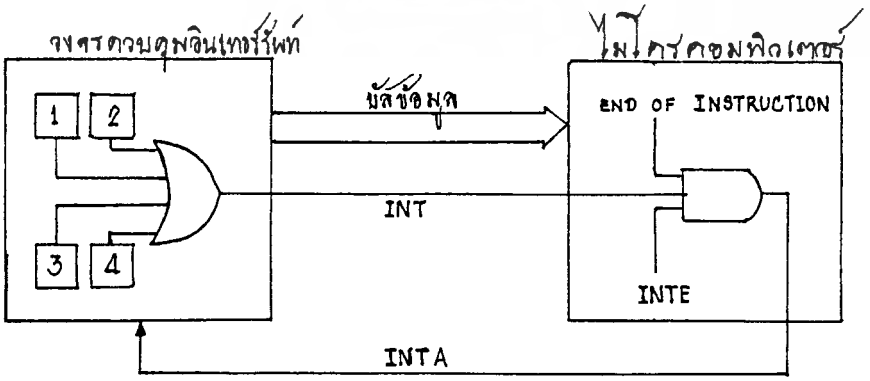


รูป 8.14 การอินเตอร์รัพท์หลายระดับ

การอินเตอร์รัพท์แบบนี้ CPU ไม่ต้องกวาด SCAN ดูแหล่งที่มาของอินเตอร์รัพท์เหมือนแบบแรก เพราะ CPU มีขาอินพุทสำหรับอินเตอร์รัพท์อยู่หลายขา CPU รู้ได้เลยว่าในขณะนั้นอุปกรณ์ตัวใดส่งสัญญาณอินเตอร์รัพท์รีเควสท์เข้ามา

ค. เวคอินเตอร์รัพท์ (Vectored Interrupt)

การอินเตอร์รัพท์แบบนี้สัญญาณอินเตอร์รัพท์รีเควสต์ (INT) จะส่งมาที่ไมโครคอมพิวเตอร์เพียงเส้นเดียวเช่นกันแบบอินเตอร์รัพท์สายเดียว แต่การแยกแหล่งที่มาของอินเตอร์รัพท์ไม่ต้องใช้การกวาดข้อมูลนอกแหล่งที่มาของอินเตอร์รัพท์จะถูกส่งจากวงจรควบคุมการอินเตอร์รัพท์เข้าสู่ CPU โดยผ่านทางบัสข้อมูล ข้อมูลนี้จะถูกใช้เป็นเวคเตอร์ สำหรับชี้แอดเดรสของโปรแกรมอินเตอร์รัพท์ที่ต้องการได้ ไมโครโปรเซสเซอร์ 8080 ใช้วิธีการอินเตอร์รัพท์ชนิดนี้ เมื่อถูกอินเตอร์รัพท์ 8080 จะอ่านข้อมูลในบัสข้อมูล [illegible] จะกำหนดได้ถึง 8 แห่ง เมื่อ CPU ได้รับคำสั่งนี้ก็จะกระโดดไปทำงานตามโปรแกรมบริการที่อุปกรณ์นั้น



รูปที่ 8.15 เวคเตอร์อินเตอร์รัพท์

บทที่ 4

POWER SUPPLY 5 V. FOR TTL

L M 723 VOLTAGE REGULATOR SHORT CIRCUIT PROTECT

คุณสมบัติของวงจร

1. ให้ out put ออกมา 5 v. และจ่ายกระแสได้สูงสุด 3 A สำหรับจ่ายให้วงจร TTL
2. สามารถ SHORT OUT PUT ได้โดยไม่ทำให้วงจรเสียหาย

การทำงานพร้อมคำนวน

L M 723 จะต่อแบบ Basic Low Voltage ซึ่ง Voltage Reference จาก Data sheet กำหนดมาให้ 7.15 v. จากข้อกำหนด $V_{out} = \left[v_{ref} \times \frac{R_2}{R_1+R_2} \right]$ ซึ่งในวงจร

$R_1 = V_R + R_3$

$R_{1MIN} = 0+750 = 750$, $R_{1MAX} = 2K+750 = 2750$

และ $R_2 = 3.6$ K

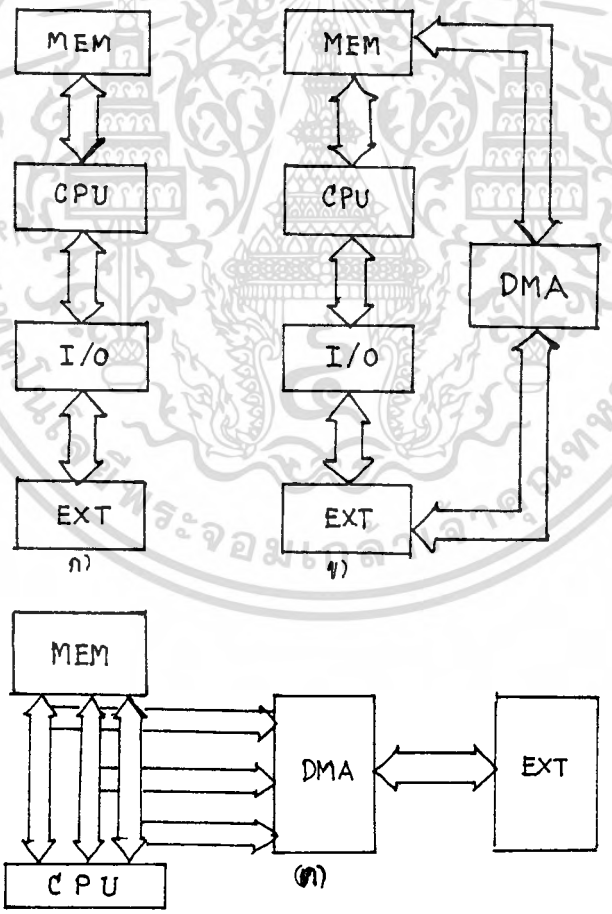
Vout เราสามารถปรับค่าได้สูงสุด $V_{out} = \left[7.15 \times \frac{3.6\ K}{4.35\ K} \right] = 5.9$ Volt

Vout เราสามารถปรับได้ต่ำสุด $V_{out} = \left[7.15 \times \frac{3.6\ K}{6.35\ K} \right] = 4.05$ Volt

DIRECT MEMORY ACCESS (DMA)

คำจำกัดความ Direct Memory Access คือ การส่งข้อมูลเข้าไปเก็บในหน่วยความจำ หรือการนำข้อมูลที่เก็บอยู่ในหน่วยความจำออกไปยังอุปกรณ์ภายนอกโดยไม่ผ่าน CPU

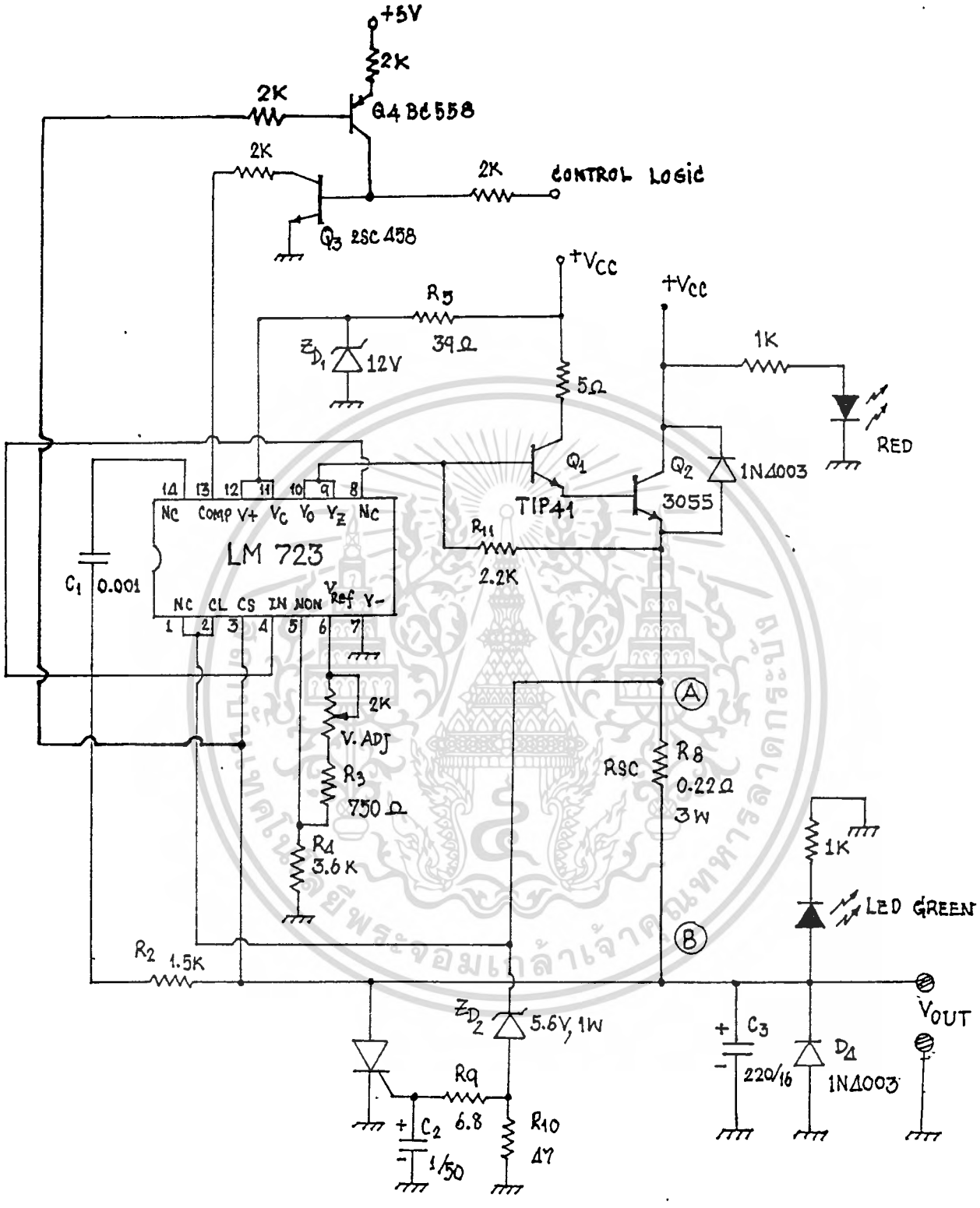
เหตุใดจึงต้องใช้ DMA ในขณะที่ไมโครคอมพิวเตอร์กำลังจัดการข้อมูลอยู่นั้น มักต้องอ่านข้อมูลจากอุปกรณ์ภายนอก วิธีหนึ่งที่ไมโครคอมพิวเตอร์จะสามารถกระทำดังกล่าวได้ก็โดยการอินเตอร์รัพท์เป็นการโอนย้ายข้อมูลผ่านตัว CPU ทำให้อัตราการโอนย้ายต่ำมาก ถ้าคอมพิวเตอร์จะต้องปรับพอยท์เตอร์ รักษาเคาน์เตอร์ และตรวจสเตตัสบิท สำหรับการโอนย้ายแต่ละครั้ง อัตราการโอนย้ายก็จะยิ่งต่ำลงไปอีก ในไมโครโปรเซสเซอร์ทั่วไป การโอนย้ายข้อมูลหนึ่งไบท์จะใช้เวลา 30 ถึง 50 ไมโครวินาที ดังนั้นอัตราการโอนย้ายข้อมูลสูงสุดถึง 16 กิโลไบท์ต่อวินาที ซึ่งเมื่อเทียบกับความเร็วในการทำงานของ CPU สามารถทำงานได้เป็นล้านไซเคิลต่อวินาที



รูป 3.16 ก. การติดต่อระหว่างอุปกรณ์ภายนอกกับหน่วยความจำ โดยผ่านทาง CPU

ข. การติดต่อระหว่างอุปกรณ์ภายนอกกับหน่วยความจำโดยตรงเชิงอุดมคติ โดยใช้ DMA

ค. การติดต่อโดยตรงผ่าน DMA ในสภาพจริงโดยที่ตัว DMA ใช้บัสร่วมกับตัว CPU



รูป [illegible] POWER SUPPLY 5 VOLT FOR TTL

(SHORT CIRCUIT PROTECT)

ถ้าเราต้องการ Vout = 5 V. จะได้ค่า R_1

$$5 = \left[7.15 \times \frac{3.6\ K}{R_1 + R_2} \right]$$

$$R_1 + R_2 = \frac{7.15 \times 3.6\ K}{5} = 5148$$

จากข้อกำหนดให้ $R_2 = 3.6\ K$

$$R_1 = 5.148\ K - 3.6\ K = 1.548\ K$$

จากสูตร $R_3 = \frac{R_1 \cdot R_2}{R_1 + R_3}$ ซึ่งค่านี้จะทำให้วงจรมีเสถียรภาพดีขึ้น

$$R_3 = \frac{1.548\ K \times 3.6\ K}{1.548\ K \times 3.6\ K} = \frac{7.967\ M}{5148} = 1548.17$$

ซึ่งค่า R_3 นี้จะตรงกับ R_2 ในวงจร

และจะเห็นว่าในวงจรมีการ Transistor เพิ่มขึ้นมาอีก 2 ตัว เพื่อทำการขยายกระแสให้สูงขึ้นโดยต่อลักษณะ Darlington และค่ากระแสตัวนี้จะกำหนดโดยค่าของ R S C ซึ่งในที่นี้ใช้ค่า R S C = 0.23 ILimit = $\frac{0.7\ v}{0.23}$ = 3.04 A ซึ่งค่านี้มีความหมายว่า

ถ้า Load ดึงกระแสเกิน 3 A จะทำให้ Vout จะ off ทันทีซึ่งควบคุมโดยขา Current Limit (cl) และขา Current sense (cs) เพราะกระแส 3A นี้จะทำให้ Voltage นี้จะทำให้ Transistor ภายใน IC LM 723 ทำงานและจะไป off out put Voltage อีกทีหนึ่ง

- ส่วนที่ PROTECT SHORT OUT PUT นั้นจะประกอบด้วย Q_3, Q_4 ในกรณีที่มีการ SHORT OUT PUT จะทำให้ Vout =0 v. คือลงกราวด์ จะทำให้ทรานซิสเตอร์ Q_3, Q_4 ทำงานและขา 13 ของ IC 723 เสมือนต่อลงกราวด์ก็จะทำให้ Vo ที่ขา 10 ของ IC 723 มีค่าเท่ากับ oV เช่นกันซึ่งจะไม่ทำให้วงจรภายในเสียหายเลย

- ในวงจรมีการต่อ L E D 2 สีไว้เพื่อบอกว่ามีการ SHORT CIRCUIT หรือเปล่าในกรณีที่วงจรทำงานปกติจะทำให้ L E D SHOW เป็นสีส้ม และถ้ามีการ SHORT CIRCUIT จะทำให้ L E D SHOW เป็นสีแดงทันที

บทที่ 5

โครงสร้างวงจรรักษาระดับแรงดัน

วงจรรักษาระดับแรงดันสามารถสร้างขึ้นจากอุปกรณ์อิเลคทรอนิคส์ที่มีคุณ Dynamic Resistance ต่ำ มีแรงดันจ่ายออก ที่เราต้องการ ซึ่งอุปกรณ์ที่ว่านี้ก็คือ Zener Diode หรือวงจรที่เกิดจากการนำเอา Zener Diode ไปต่อรวมอยู่ก็ได้

วงจรรักษาแรงดันสามารถจำแนกประเภทตามสภาพต่อโหลดได้ 2 แบบ คือ

1. แบบขนาน (Shunt Type)
2. แบบอนุกรม (Series Type)

Regulator Unit

Regulator Unit

Series Type

Shunt Type

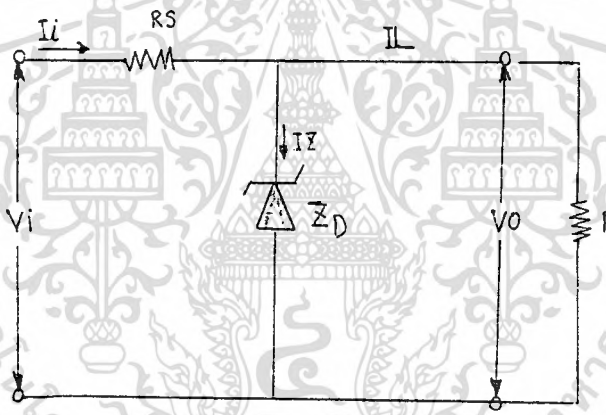
ข้อแตกต่างที่สำคัญของวงจรรักษาระดับแรงดันสองแบบนี้สามารถสรุปได้ดังนี้

1. ประสิทธิภาพของวงจรแบบอนุกรมจะพิจารณาที่แรงดัน เนื่องมาจากกระแส input กับกระแส output จะมีค่าใกล้เคียงกัน $= \frac{vo}{vi}$

2. ประสิทธิภาพของวงจรแบบขนานจะพิจารณาที่กระแส นี้เนื่องมาจากแรงดัน input กับแรง output จะมีค่าใกล้เคียงกัน $= \frac{Io}{Ii}$
3. วงจรแบบขนานมักใช้งานเมื่อ Vo และ RL คงที่ นอกจากนี้ยังนิยมใช้เฉพาะในกรณีที่แรงดันจ่ายออกมีค่าต่ำ กระแสจ่ายเข้ามีค่าสูง
4. วงจรแบบขนานจะมีราคาต่ำกว่าอนุกรม เนื่องมาจากเป็นวงจรแบบง่าย
5. วงจรแบบขนาน ไม่จำเป็นต้องมีการป้องกันกระแสไหลเกิน
6. วงจรอนุกรมนิยมใช้งานกับงานโดยทั่วไป

ในบทนี้จะได้นำเอาวงจรรักษระดับแรงดันต่างๆ มาวิเคราะห์พอเป็นพื้นฐานแก่การศึกษาในระดับสูงขึ้นและสามารถนำไปประยุกต์ใช้งานได้

วงจรรักษระดับแรงดันแบบซีเนอร์ไดโอด



เป็นวงจรรักษระดับ แรงดันพื้นฐานซึ่งประยุกต์ใช้คุณสมบัติของ ซีเนอร์ไดโอดจากรูปวงจร โดยใช้กฎแรงเคลื่อนของ เคอร์ชอฟฟ์

$$Vi = (Iz \_IL)\, Rd + Vo$$

โดยใช้กฎของโอห์ม

$$IL = \frac{VO}{RL}, \quad IZ = \frac{VO}{rf}$$

เมื่อ Rz = ความต้านทานของ ซีเนอร์ไดโอด

$$Vi = \left(\frac{vo}{rf} + \frac{vo}{RL}\right) RDL + VO$$

$$= VO \left(\frac{RD}{Rz} + \frac{RD}{R1} + 1\right)$$

$$\frac{VI}{VO} = \frac{KD}{Rz} + \frac{RD}{RL} + 1$$

$$\frac{VO}{VI} = \frac{1}{1 + \frac{RD}{Rz} + \frac{Rd}{R1}}$$

$$Sv = \frac{\frac{VO}{VI}}{\frac{VO}{VI}}$$

$$SV = \frac{1}{1 + \frac{DR}{Rz} + \frac{DR}{R1}}$$

ถ้า $RL >> RD >> RZ$

$$SV = \frac{Rz}{RD}$$

จากรูปวงจร $RO = RZ \;//\; RD$

วงจรรักษาระดับแรงดันแบบนี้มี Temperature Co-efficient ต่ำ นอกจากนี้ยังมีการสูญเสียกำลังใน ซีเนอร์ไดโอดเป็นจำนวนมาก จึงมีประสิทธิภาพค่อนข้างต่ำ

ในการคำนวนหาพิกัดของอุปกรณ์ในวงจรสามารถคำนวนโดยใช้สูตร

$$VZ = VO$$

$$PZ\ (MAX) = \left[\frac{VI - VO}{RD} - IL\ (MIN) \right] VZ$$

$$Rd\ (MAX) = \frac{VI\ (MIN) - VZ\ (MAX)}{IL\ (MIN) + IL\ (MAX)}$$

$$RD\ (MIN) = \frac{VI\ (MAX) - VZ\ (MIN)}{\dfrac{PZ\ (MAX) + IL\ (MIN)}{VZ\ (MAX)}}$$

<u>ตัวอย่าง</u> วงจรรักษาระดับแรงดันแบบ ซีเนอร์ไดโอด ดังรูปในบทเรียนมีค่า Rd = 200 ohm., Vz = 20 ohm. แรงดันจ่ายออกที่มีค่าปกติเท่ากับ 10 v. ที่กระแสปกติ 5 mA จงคำนวนหา

1. การเปลี่ยนแปลงของแรงดันจ่ายออกเมื่อแรงดันจ่ายเข้าเปลี่ยนแปลงไป 1.1 v. และกระแสโหลดคงที่ 5 m A

2. การเปลี่ยนแปลงของแรงดันที่จ่ายออกเมื่อแรงดันจ่ายเข้าคงที่ และกระแสโหลดเปลี่ยนแปลงจาก 5 m A ไปสู่ 1 m A

SO จากโจทย์เราจะได้ว่า $RL = \frac{VO}{IL} = 2{,}000$ ohm.

1. จากสมาการ $SV = \frac{1}{1 + \frac{RO}{RZ} + \frac{RO}{RL}} = \frac{1}{1 + \frac{200}{20} + \frac{200}{2{,}000}}$

$= \frac{1}{11.1}$

จาก $SV = \frac{\Delta VO}{\Delta VI}$

$\Delta VO = SV\ \Delta VI = \frac{1}{11.1}.1.1 = 0.099$ V. Ans

2. จาก $RO = RZ\ //\ RD = \frac{20}{}x\ \frac{200}{} = \frac{200}{11}$

$20 + 200$

$RO = \frac{VO}{IL} \quad \Delta \frac{VO}{\Delta IL}$

$\Delta VO = \Delta IL \cdot RO$

$$= (5-1) \times 10^{-3} \times 200 = 0.073 \text{ Ans}$$

11

ตัวอย่าง จงคำนวนหาค่า Rd สำหรับวงจรรักษาระดับแรงดันแบบ ซีเนอร์ไดโอด ดังรูป ในบทเรียนกำหนด Vi (max) = 15 v. VO (max) = 7.5 v. IL (max) = 100 mA. Vi (min) =13.5 v. Vz (min) =7.3 v. Iz (min) = 20 mA. IL (min) = 8.2 v. และ Vi (max) = 16.5 v.

SO/ จาก

$$RD\ (MAX) = \frac{VI\ (MIN) - VZ\ (MAX)}{IZ\ (MIN) + IL\ (MAX)}$$

$$= \frac{13.5 - 8.2}{20 \times 10 - 100 \times 10^{3}} = 44.17 \text{ Ohm.}$$

$$RD\ (MIN) = \frac{VI\ (MAX) - VZ\ (MIN)}{\frac{PZ}{VZ\ (MAX)} + IL\ (MIN)} = 40.8 \text{ Ohm.}$$

$$RD = \frac{RD\ (MAX) + RD\ (MIN)}{2}$$

$$= 42.48 \text{ Ohm.} \quad \text{Ans}$$

Single Bjt Series Regulator

เรียกอีกอย่างหนึ่งว่า Emitter Followrr Regulator เป็นวงจรที่ปรับปรุงข้อบกพร่องของแบบ ซีเนอร์ เพื่อให้วงจรใช้งานกับกระแสที่มากขึ้นและมี Sv ดีขึ้น วงจรแบบนี้จัดได้ว่าเป็นแบบ Series Regulator

ตัวอย่าง __ จากวงจร Emitter Follower Regulator ถ้า Vi = 18 +- 2 V, Vo (in) = 12 V, IL 50 ถึง 250 m A จงคำนวนหา Pd (max) ของไดโอด Sv, Ro ของวงจรและค่า Rd เมื่อทรานซิสเตอร์มีค่า hfe = 44 และ hie = 5.5 Ohm. ที่ IC = 250 m A, Vz = 12 Ohm. Vz (max) 12.6 V, Vz (min) = 11.4 V, Pz (max) = 500 mw, hfe (max) = 130, Vz (max) = 11.7 V, Iz (min) = 5.44 m A

SO/N PD (MAX) = (VI (MAX) - VO (MIN) IL (MAX)

$= (20 - 12) \times 250 \times 10^{-3} = 2$ WATT

$$IB\ (MAX) = \frac{IL\ (MAX)}{HFE + 1} = \frac{250 \times 10^{-3}}{44 + 1} = 5.56\ m\ A$$

$$RD\ (MAX) = \frac{VI\ (MIN) - VZ\ (MAX)}{IB\ (MAX) + IZ\ (MIN)}$$

$$= \frac{16 - 12.6}{(5.56 + 5.44) \times 10^{-3}}$$

$$IB\ (MIN) = \frac{IL\ (MIN)}{HFE\ (MAX)} = \frac{50 \times 10^{-3}}{130} = 0.385\ m\ A$$

$$RD\ (MIN) = \frac{VI - (MAX) \frac{VZ}{PZ} - (MIN) \frac{VZ}{VZ\ (MIN)}}{+\ IB\ (MIN)}$$

$$SV = \frac{VO}{VI} = \frac{VO}{Vi} = \frac{VZ}{Rd + RZ}$$

$$RO = \frac{HIE + RZ}{HIE + 1}$$

$$Pd = (VI\ mAX) - VO\ (MIN)\ \ IL\ (MAX)$$

$$IB\ (MAX) = \frac{IL\ (MAX)}{HFE + 1}$$

$$RD\ (MAX) = \frac{VI\ (MAX) - VZ\ (MAX)}{IB\ (MAX) + IZ\ (MAX)}$$

$$RD\ (MIN) = \frac{VI\ (MAX) - VZ\ (MIN)}{\frac{PZ}{VZ\ (MIN)} + IB\ (MIN)}$$

$$RD = \frac{RD\ (MAX) + RD\ (MIN)}{2}$$

$$= \frac{20 - 11.4}{0.5 + 0.385 \times 10^{-3}} = 194.4 \text{ Ohm.}$$

$$\frac{11.4}{RD\ (MAX) + RD\ (MIN)}$$

$$RD = \frac{}{} = 251.8 \text{ Ohm.}$$

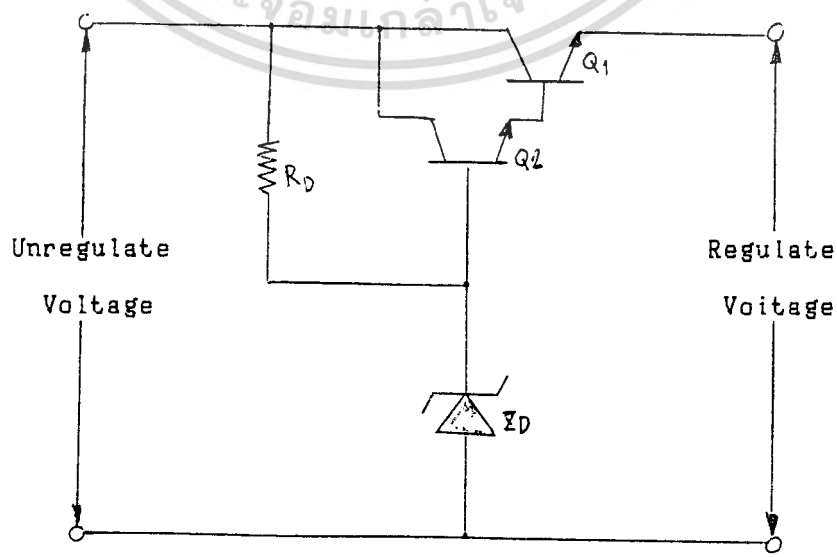
$$RO = RZ + \frac{HIE}{HFE + 1} = 12 + \frac{5.5}{44 + 1}$$

$$= 0.388 \text{ Ohm.}$$

$$SV = \frac{RZ}{FD + RZ}$$

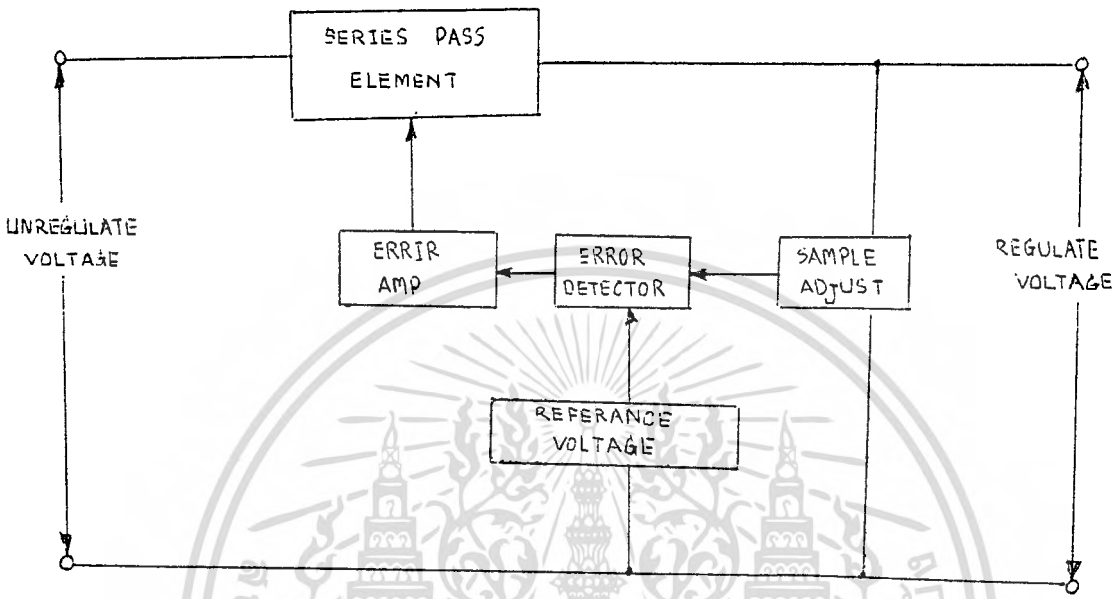
$$= \frac{12}{251.8 + 12} = 0.0445$$

ข้อเสียของวงจรแบบนี้คือ ในกรณีที่กระแสโหลดสูงเกินไปค่า Pd ของ Zener จะมีค่ามาก ซึ่งทำให้วงจรมีราคาแพงแต่อาจแก้ไขได้โดยใช้ทรานซิสเตอร์มีค่า hfe สูงหรือใช้ทรานซิสเตอร์ที่มีค่า hfe สูง หรือใช้ทรานซิสเตอร์ที่มี hfe ต่ำ 2 ตัว ต่อแบบ Dartington ดังรูป



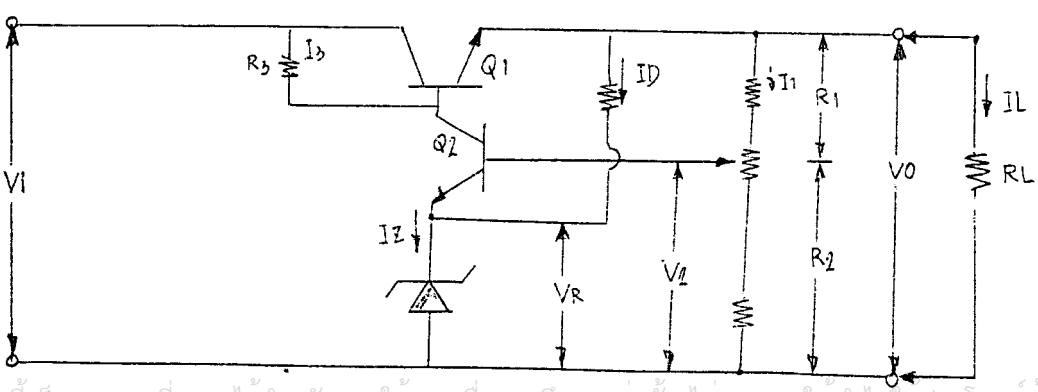
Feed Back Regulate

เป็นแบบที่นิยมใช้งานกันมากที่สุดแบบหนึ่ง เพราะเป็นวงจรที่มีเสถียรภาพทางแรงดัน
เพื่อความสะดวกในการอธิบายการทำงานของวงจร Feed Back Regulator จะใช้ Block Diagram ข้างล่างนี้ประกอบ



วงจร Sampling ซึ่งต่อขนานไว้กับ output จะทำการสุ่มตัวอย่างโดยการใช้วงจรโวลเตอดิ ไวเตอร์ นำเอาแรงดันที่จ่ายออก output บางส่วนมาตรวจสอบเปรียบเทียบกับแรงดันมาตรฐาน (Reference Voltage) ว่าแตกต่างกันเท่าใดในภาค Error Detector หากมีความแตกต่างกัน เพียงเล็กน้อยก็จะถูกขยายให้มีความแรงมากขึ้นในภาค Error Amp สัญญาณจากภาค Error Amp จะไปควบคุมความต้านทานของ Series pass Element ให้มีค่าความต้านทานที่เหมาะสม

จากรูปเป็นวงจร Transistor Feed Back Regulator อย่างง่าย



ในการออกแบบวงจรดังกล่าวสามารถทำได้โดยทำตามลำดับดังนี้

1. $IZ = ID2 + IE2$

2. $RD = \dfrac{VO - VR}{ID}$

3. $IB2 = \dfrac{IC2}{HFE2}$

4. $V2 = VB2 + VR$

5. $R1 = \dfrac{VO - V2}{I1}$

6. $R2 = \dfrac{V2}{I1}$

7. $IB1 = \dfrac{IL + IL + ID}{HFE\ 1} = \dfrac{IO}{HFE\ 1} = \dfrac{IC}{HFE\ 1}$

8. $I3 = IB\ 1 + IC\ 2$

9. $R3 = \dfrac{VI - (VBE\ 1 + VO)}{I3}$

นอกจากนี้เรายังสามารถคำนวนหาคุณสมบัติของวงจรได้โดยใช้สูตร

$$SV = \frac{1}{GM \quad R3}$$

$$GM = \frac{HFE\ 2 \ .\ R2}{R1 + R2} \ . \ \frac{1}{(R1//E2)+HIE2 + (1+HFE2)\ RZ}$$

$$RO = \frac{RO + (R3 + HIE1) \ / \ (C1 + HFE1)}{1 + GM\ (R3 + BO)}$$

$$VO = +- \ SV \ \Delta \ VI \ + \ RO \ \Delta \ IL$$

<u>ตัวอย่าง</u> จงออกแบบ Feed Back Regulator กำหนด

VO = 25 V.

IL =< 1 Amp.

vi = 50 +- 5 v. ที่ ro = 10 Ohm.

การคำนวนหาค่า VO เมื่อ IL เปลี่ยนจาก 0-1 Amp. และ VI เปลี่ยนแปลงไป

I 5 E

SO/^{2} เลือก VR - <u>VO</u> โดยใช้ zenner diode ซึ่งมี vz = 7.5 v. 2 ตัว เมื่อจ่ายเป็น VR = 15 ซึ่งจะทำให้เกิด RZ = 12 ที่ IZ = 20 MA

2. ตัวต่ออันคั่นกันเพื่อจ่าย เป็น VR = 15 V. ซึ่งจะทำให้เกิด rz = 12 Ohm. ที่ IZ = 20 mA

* **หมายเหตุ** ข้อมูล Iz และ Iz หาได้จากคู่มือ Zener Diode
ประมาณให้ IC2 - IE2 = 10 mA คือทนได้ถึง 30 mA ที่ VCE = 45 V.
คู่มือทรานซิสเตอร์ถ้า ICL = 10 mA
HFE = 220, HFC = 200, HIE = 800 Ohm.

เลือก ID = 10 mA

1. IZ = IC2 + IE2 = 20 mA

2. $RD = \frac{VO - VR}{ID} = 1\ K\ Ohm.$

3. $ib2 = \frac{IC2}{HFE2} = 4\ \mu A$

เราต้องการใช้ I1 >> IB2 เลือก I1 = 10 mA ที่ VBE = 0.7 V.

4. V2 = VBE2 + VR = 15.7 V.

5. $R1 = \frac{VO - VZ}{I1} = 930\ Ohm.$

6. $R2 \quad \frac{VZ}{IL} = 1,570$

เลือก ZN 1722 Silicon Power Transistor เป็น Q1 ที่ IC = 1 mA
hFE = 125, hfel = 1000, hie = 20 Ohm.

7. IB2 = IL + I1 + ID / HFE1 - 8 mA

8. I3 = IB1 + IC2 = 18 mA

9. R3 = $\frac{VI - (VBE1 + VO)}{I} = 1,350$

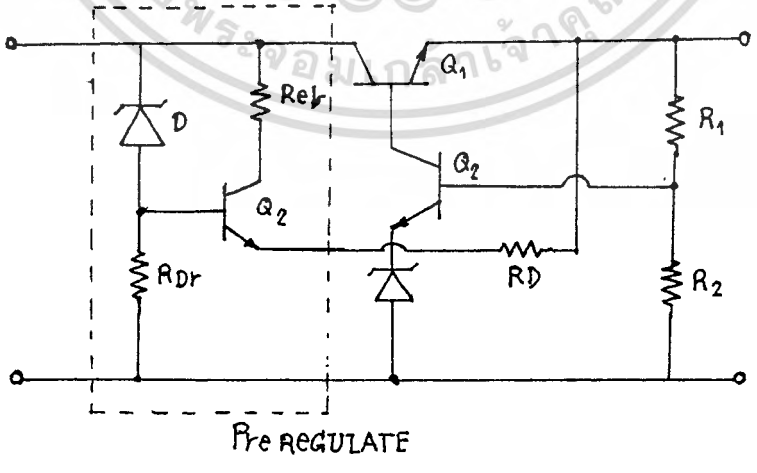
SV = 0.022

GM = $\frac{I}{SV}$ = 0.033, RO = 0.51 Ohm. VO = 0.73

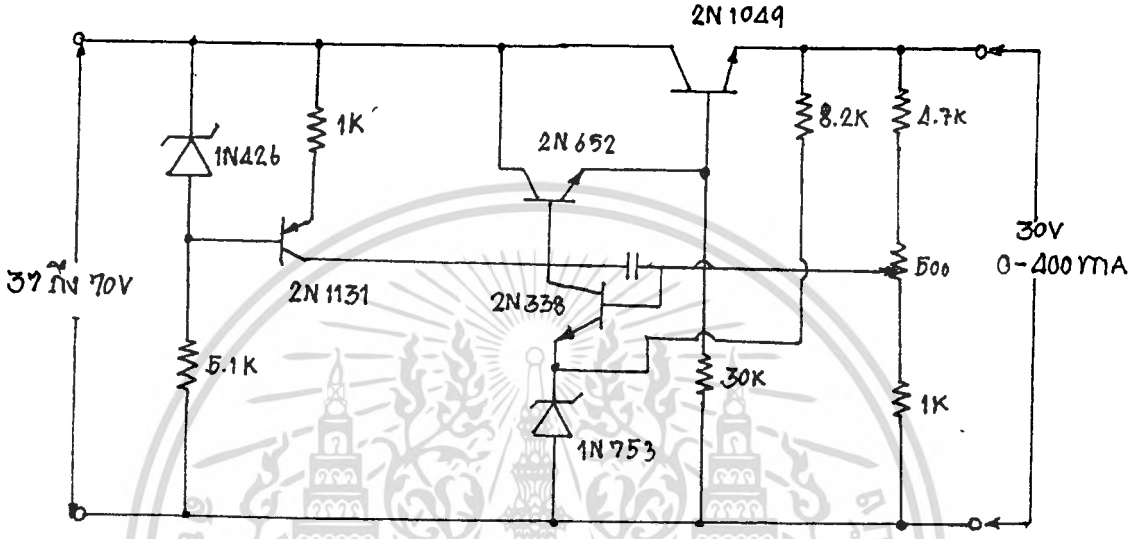
R3

High Performance Feed Buck Regulator

ในกรณีต้องการ ให้วงจร Feed Back Regulator ทำงานได้ดีและมีประสิทธิภาพสูงขึ้น อาจปรับปรุงโดยแทนที่ตัวต้านทาน R3 ด้วย Perl Regulator (ซึ่งเป็น Shunt Regulator ที่ใช้งานได้ดีกับกระแสต่ำ) ดังรูป

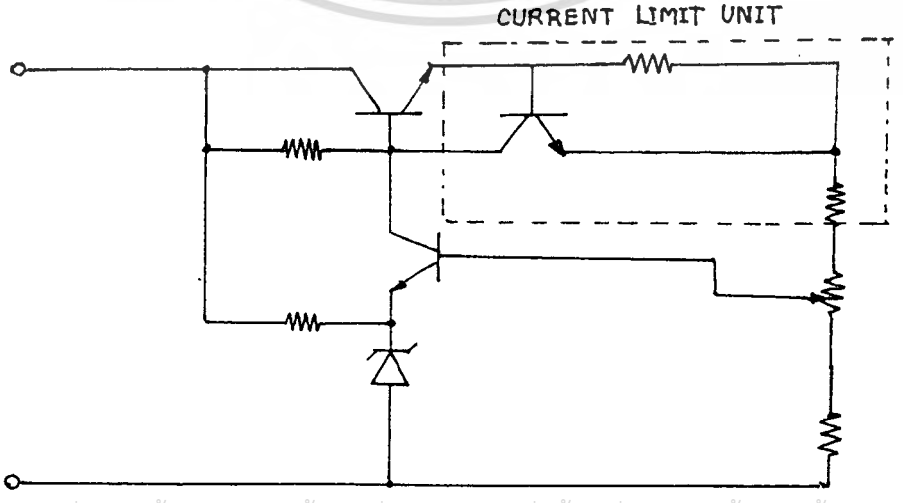


และหากเราต้องการให้วงจรมีประสิทธิภาพยิ่งขึ้นควรใช้ Q1 ซึ่งมี hFE สูงมากแต่จะหาทรานซิสเตอร์ซึ่งมี hFE สูง ๆ ทำได้ยาก อาจแทนที่ Q1 โดยใช้ทรานซิสเตอร์ต่อแบบดาร์ลิงตันดังที่ได้อธิบายมาแล้วในบทเรื่อง Serics Regjlator แล้วเพิ่มเติมอุปกรณ์บางตัวเพื่อความเหมาะสมในการใช้งานดังรูป



Current Limitting In Power Supply

ข้อเสียประการหนึ่งของ Series Regulator คือ Pass Element จะต่ออนุกรมกับ Load ในกรณีเช่นนี้หากเกิดการชอร์ท ของโหลดจะส่งผลทำให้ Pass Element ชำรุด เราอาจป้องกันการชำรุดของ Pass Element เนื่องจากการ์ชอร์ทโหลดนี้ได้ โดยใส่ภาคจำกัดกระแส ซึ่งเป็นวงจรง่าย ๆ ดังรูป



ค่า Rs ที่ใส่เข้าไปสามารถคำนวนได้จากสูตร

$$RS = \frac{VBE}{I\ (MAX)}$$

เช่นต้องการจำกัดกระแสสูงสุดที่ 1 Amp และ VBE = 0.7 V.

$$RS = \frac{0.7}{1} = 0.7\ Ohm.$$

Monolithic Regulator

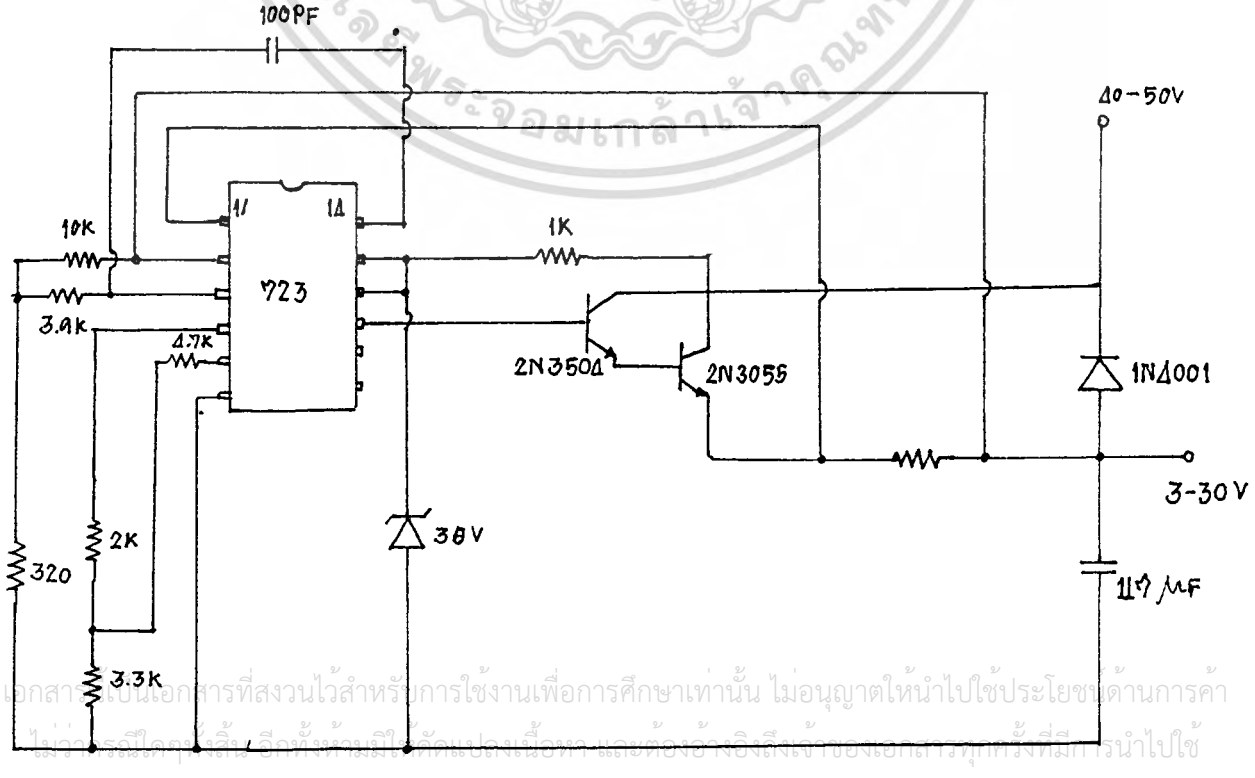
เนื่องจากการออกแบบวงจรรักษาระดับแรงดันด้วย Discreat Element ยุ่งยากและสิ้นเปลืองในปัจจุบันจึงมีผู้ออกแบบ Monolithic Regulator Ic สำหรับใช้งานเป็นภาครักษาระดับแรงไฟ ซึ่ง Ic ชนิดดังกล่าวสามารถแบ่งออกได้เป็น 2 กลุ่มใหญ่ ๆ

1. รักษาระดับแรงไฟค่าคงที่
2. รักษาระดับแรงไฟไปกับค่าแรงดันได้

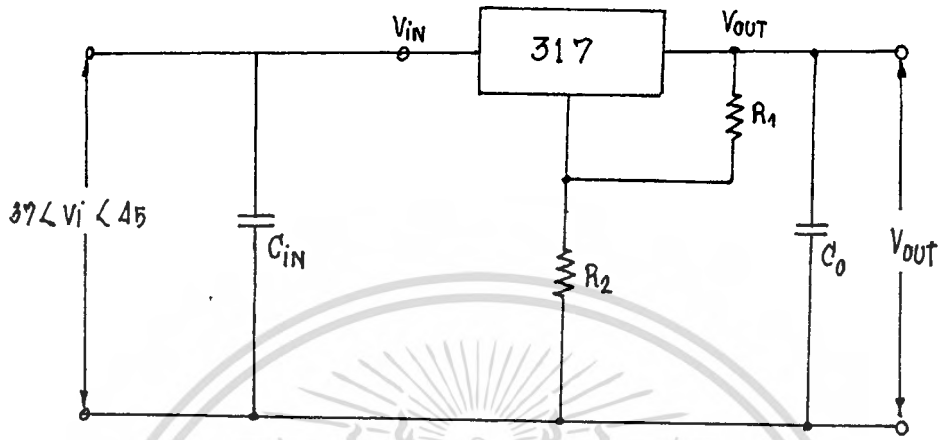
สองชนิดนี้ยังสามารถแยกย่อยออกได้เป็น 2 ประเภท คือ รักษาระดับแรงดันบวกและรักษาระดับแรงดันลบ

Adjustable Out Put Ic Regulator

ไอซีตระกูลนี้เป็นที่นิยมใช้ในปัจจุบัน มีด้วยกัน 2 เบอร์ คือ 723 และ 317 สำหรับเบอร์ 723 มีข้อเสียคือต้องอาศัยทรานซิสเตอร์มาต่อเป็น Pass element ดังรูป



สำหรับเบอร์ 317 ในปัจจุบันเป็นที่นิยมใช้กันมากที่สุดเนื่องจากมีความสะดวกในการใช้งาน และเป็นวงจรที่มีประสิทธิภาพสูงขอให้พิจารณาวงจรตามรูปซึ่งเป็นวงจรใช้งานมาตรฐาน



จากรูปวงจรมีส่วนที่จะต้องคำนวนออกแบบเพียงค่าเดียวคือ R2 ซึ่งสามารถคำนวนได้โดยใช้โดยสมการ

$$VOUT = 1.25 \left(1 + \frac{R2}{R1} \right) + IAdj \quad R2$$

ตามปกติ IADI จะมีค่าน้อยกว่า 100 nA ดังนั้น IADI R2 ซึ่งมีค่าน้อยมากและสามารถตัดทิ้งได้ในทางปฏิบัติ หรืออาจได้ว่า

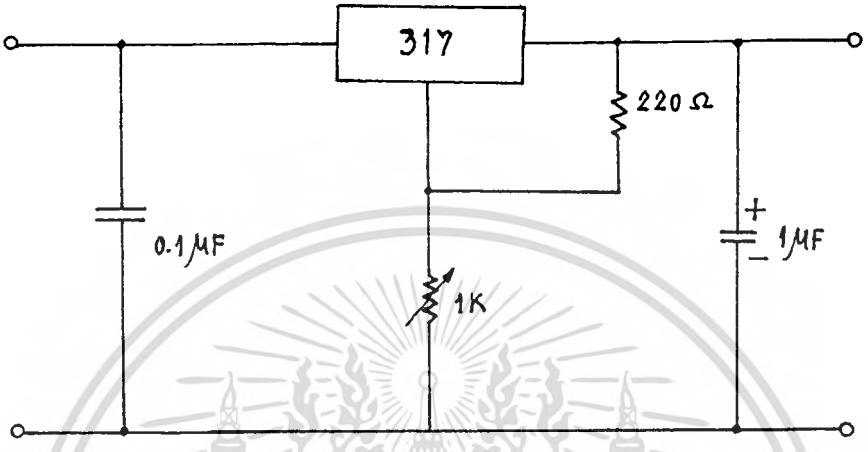
$$VOUT = 1.25 \left(1 + \frac{R2}{R1} \right)$$

จากรูปวงจรเห็นว่า R2 จะเป็นตัวกำหนด VOUT ซึ่งจะมีค่าสูงสุดได้ไม่เกิน 37 V. ตามคู่มือ IC และ VOUT จะมีค่าต่ำสุด ได้เท่ากับ 1.25 V. เมื่อ R2 = 0 Ohm.

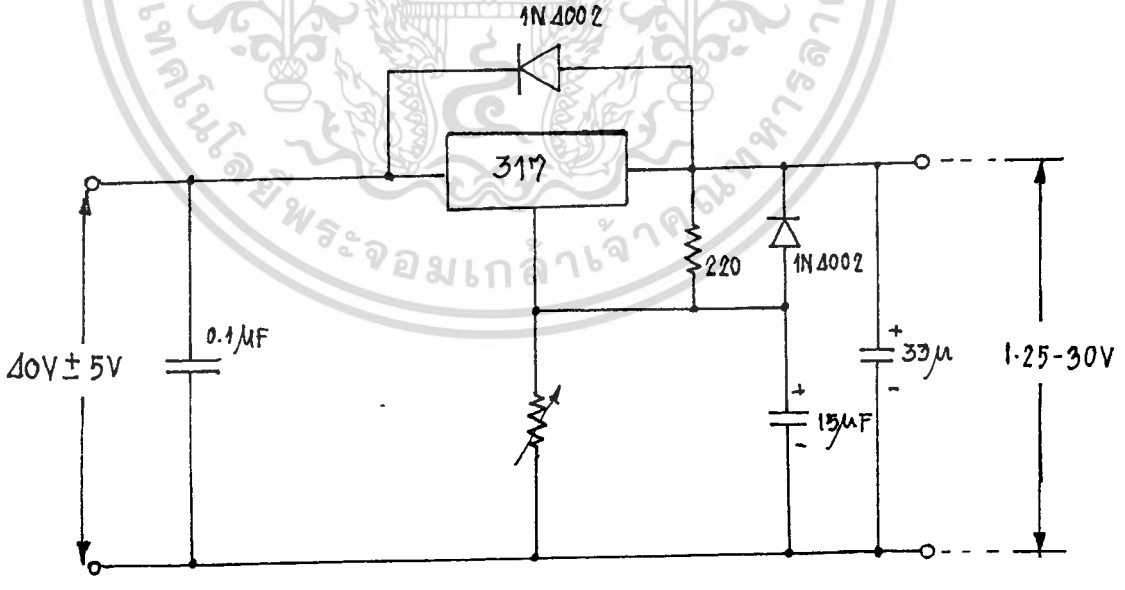
ในกรณีที่ออกแบบในวงจรดังกล่าวสามารถปรับแรงดันได้ 1.25 - 2.5 Volts เราต้องใช้ R2 เป็น Pot ที่มีค่าเท่ากับ

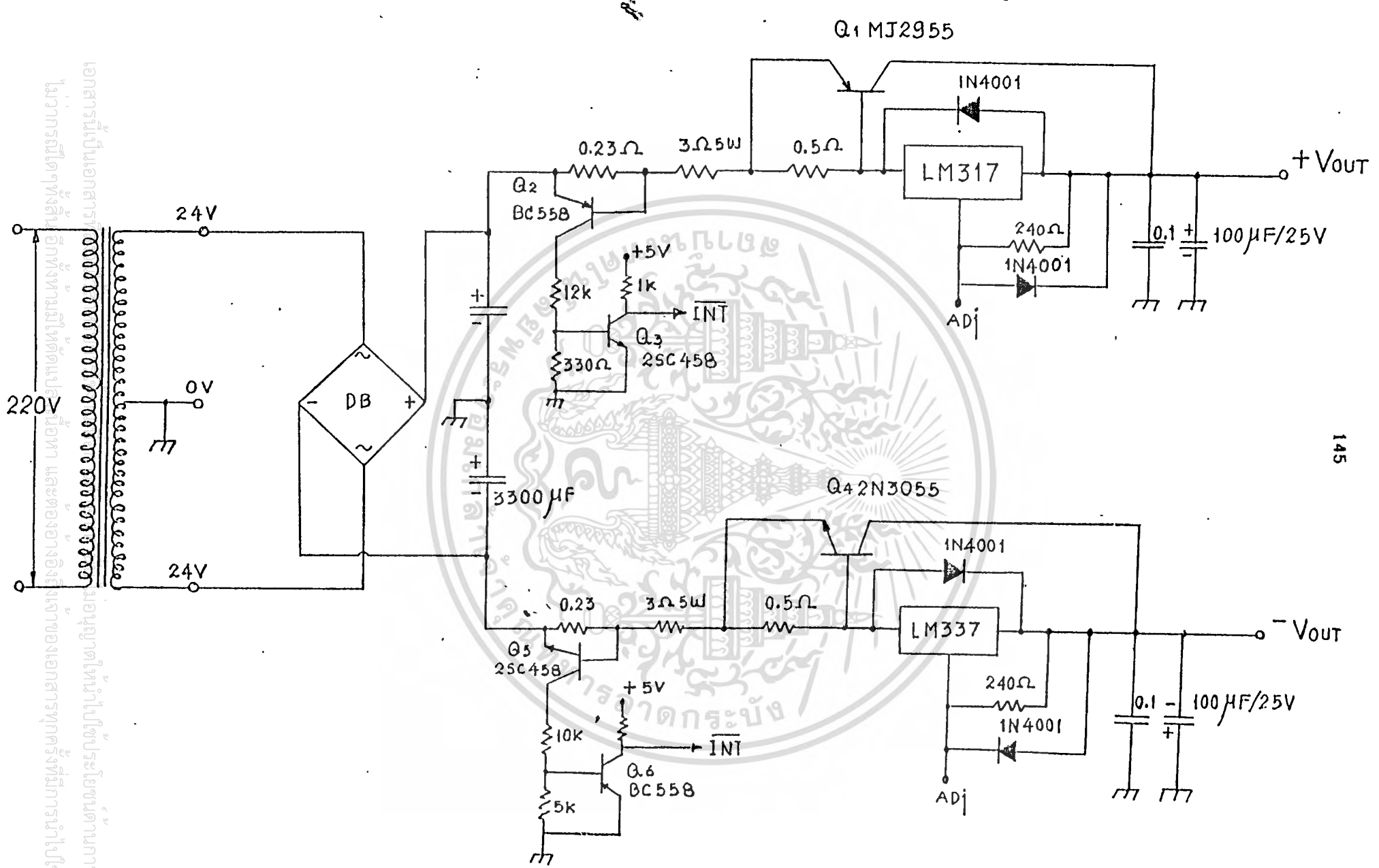
$$R1 = \frac{6250}{28.25} = 221.23 \text{ Ohm.}$$

เลือกใช้ R1 = 221 Ohm. +- 5 %



สำหรับวงจรในทางปฏิบัติอาจมีอุปกรณ์มากกว่าวงจรมาตราฐานเพื่อป้องกันสิ่งอื่น ๆ เช่น Transistant Bock E.K.F. ได้สมบูรณ์ จากรูปเป็นตัวอย่างวงจร




Q1 MJ2955
1N4001
0.23Ω
3Ω5W
0.5Ω
LM317
+VOUT
Q2
BC558
240Ω
1N4001
0.1
100μF/25V
ADj
+5V
12k
1k
INT
Q3
2SC458
330Ω
24V
220V
0V
DB
3300μF
Q4 2N3055
1N4001
0.23
3Ω5W
0.5Ω
LM337
-VOUT
Q5
2SC458
+5V
240Ω
1N4001
0.1
100μF/25V
10k
INT
Q6
BC558
5k
ADj

อธิบายการทำงาน

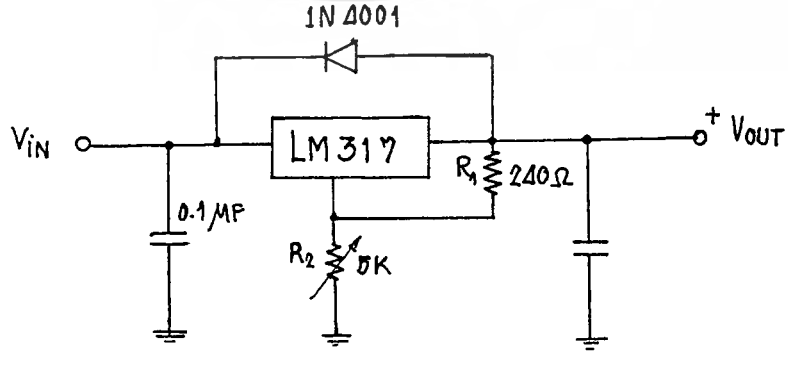
จะเห็นได้ว่า VCC เราจะได้ค่าประมาณ 34 V. เพราะ VAC = 24 V. และต่อ Bildo Bridge ให้เป็นลักษณะ +,-, GND VCC = √2 x VAC = 33.9 V. เช่นเดียวกัน - VEE ก็จะได้ประมาณ - 33.9 v.

มาดูวงจรส่วนไฟบวก LM 317 เสียก่อนโดยวงจรจะประกอบด้วยอุปกรณ์ต่าง ๆ หลายตัว ซึ่งสามารถแบ่งออกเป็นชุด ๆ ดังนี้

1. ชุด Protect กระแสเกินจะประกอบด้วยค่า R 0.23 , Q2,Q3,R 12K, R 330 R 1 K. ซึ่งการทำงานเป็นดังนี้คือ เมื่อโหลดดึงกระแส 3 A และจะทำให้กระแสนี้ไหลผ่าน R 0.23 ด้วยเพราะต่อแบบอนุกรมกันและจะทำให้มีแรงดันตกคร่อม R 0.23 นี้ = 0.23 x 3 = ..69 V. หรือ = 0.7 V. ซึ่งเป็นแบบแรงดัน Bias ให้กับ TRANSISTOR Q2 ทำงาน และจะส่งกระแสให้ Q3 ทำงานด้วยเช่นกันและจะทำให้ขา INT ตกลงระดับศูนย์ ซึ่งเป็นการ INT CPU ให้เป็นไปตาม SUBROUTINE ที่จะไป OOF OUT PUT อีกทีหนึ่ง

2. ชุดขยายกระแสซึ่งประกอบด้วย R 0.5 และ Q 1 ซึ่งต่อ SHUNT กับ LM 317 Q1 นี้จะทำงานก็ต่อเมื่อมีแรงดันตกคร่อม R 0.5 นี้มีค่า = 0.7 V. เพราะฉะนั้นเมื่อ LOAD ดึงกระแส IL = $\frac{0.7}{0.5}$ = 1.4 A จะทำให้ Q1 ทำงานช่วยเป็นทางผ่านของกระแสของ LM 317 อีกที่หนึ่งเพราะตาม SPECIFICATION บอกไว้ว่า LM 317 สามารถทนกระแสได้ 1.5 A

3. ชุด CONTROL OUTPUT VOLTAGE (LM 317) ซึ่งเป็น IC REGULATOR สามารถปรับค่า VOLTAGE OUT PUT ได้ตั้งแต่ 1.25-37 V. โดยปรับค่า VR ที่ขา ADJ ดังรูปข้างล่างนี้



$$V_{out} = 1.25 \left(1 + \frac{R_2}{R_1}\right)$$

และจากสูตรข้างบนนี้ก็ให้ได้ค่า VOUT ตั้งแต่ 1.25-37 V. แต่เราสามารถนำมาประยุกต์ใช้โดยให้ OUT PUT VOLTAGE ตั้งแต่ 0V-37 ได้โดยการป้อน VOLTAGE ที่ขา ADJ ตั้งแต่ -1.25 ถึง 35.75 V.

ซึ่งผลจากการทดลอง OUT (LM 317) เป็นดังตารางข้างล่างนี้

ป้อน VOLTAGE ที่ขา ADJ (V)	แรงดัน OUT PUT ที่วัดได้ (V)
- 1.25	0V
0	1.25
1	2.25
2	3.25
3	4.25
4	5.25
5	6.25
6	7.25
7	8.25
8	9.25
9	10.25
10	37

จากการทดลองจะเห็นว่า OUT PUT จะมีค่ามากกว่า VADJ อยู่ 1.25 V. ตัวอย่างเช่นถ้าเราต้องการ OUT PUT VOLTAGE เท่ากับ 5 V. ดังนั้นเราต้องป้อน VADJ มีค่าเท่ากับ 5-1.25 = 3.75 นั่นเอง

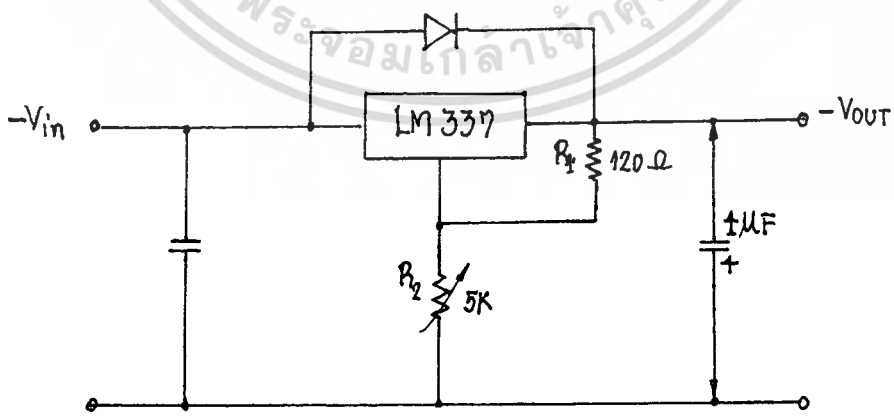
* เช่นเดียวกันส่วนควบคุมไหล (LM 337)

ประกอบด้วย 3 ส่วนใหญ่ คือ

1. ชุด PROTECH กระแสเกิน จะประกอบด้วย R 0.23 , Q5,Q6,R10K,R5K,R1K ซึ่งการทำงานเป็นดังนี้คือ เมื่อโหลดดึงกระแส 3 A และจะทำให้กระแสนี้ไหลผ่าน R 0.23 ด้วยเพราะต่ออนุกรมกันและจะทำให้มีแรงดันคร่อม R 0.23 นี้ เท่ากับ 3.23 x 3 = 0.69 V. หรือประมาณ 0.7 V. ซึ่งเป็นแรงดัน Bias ให้กับ TRANSISTOR Q5 ทำงานและจะส่งกระแสให้ Q5 ทำงานด้วยเช่นกันและจะทำให้ขา INT ตกระดับศูนย์ ซึ่งเป็นการ INT CPU ให้เป็นไปตาม SUBROTINE ที่จะไป OFF OUT PUT อีกทีหนึ่ง

2. ชุดขยายกระแสซึ่งประกอบด้วย R 0.5 และ Q4 ซึ่งต่อ SHUNT กับ LM 337 Q4 นี้จะทำงานก็ต่อเมื่อมีแรงดันตกคร่อม R 0.5 นี้มีค่าเท่ากับ = 0.7 V. เพราะฉะนั้นเมื่อ LOADดึงกระแส IL = $\frac{0.7}{0.5}$ = 1.4 A จะทำให้ Q4 ทำงานช่วยเป็นทางผ่านของกระแสของ LM 337 อีกทีหนึ่งเพราะตาม SPECIFICATION บอกไว้ว่า LM 337 สามารถทนกระแสได้ 1.5 A

3. ชุด CONTROL OUT PUT VOLTAGE (LM 337) ซึ่งเป็น IC REGULATOR สามารถปรับค่า VOLTAGE OUTPUT ได้ตั้งแต่ -1.25 ถึง -37 V. โดยปรับค่า VR ที่ขา ADJ ดังรูปข้างล่างนี้



$$-V_{OUT} = -1.25\left(1+\frac{R2}{R1}\right)$$

และจากสูตรข้างล่างนี้สามารถให้ค่า OUT PUT VOLTGAE ตั้งแต่ -1.25 ถึง -37 V. และเราสามารถนำมาประยุกต์ใช้ โดยให้ค่า OUT PUT VOLTAGE ตั้งแต่ 0 ถึง -37.75 ได้โดยการป้อน VOLTAGE ที่ขา ADJ ตั้งแต่ 1.25 ถึง -35.75 V.

ซึ่งจากผลการทดลอง VOUT (LM 337) เป็นดังตารางข้างล่างนี้

ป้อน VOLTAGE ที่ขา ADJ (V)	แรงดัน OUT PUT ที่วัดได้ (V)
1.25	0
0	-1.25
-1	-2.25
-2	-3.25
-3	-4.25
-4	-5.25
-5	-6.25
-6	-7.25
-7	-8.25
-8	-9.25
-35.75	-37

จากผลการทดลองจะเห็นว่า OUT PUT จะมีค่าน้อยกว่า VADJ อยู่ 1.25 ตัวอย่าง เช่น ถ้าเราต้องการ OUT PUT VOLTAGE เท่ากับ -5 V. ดังนั้นเราต้องป้อน VADJ มีค่าเท่ากับ -5- (-1.25) = -3.75 นั้นเอง

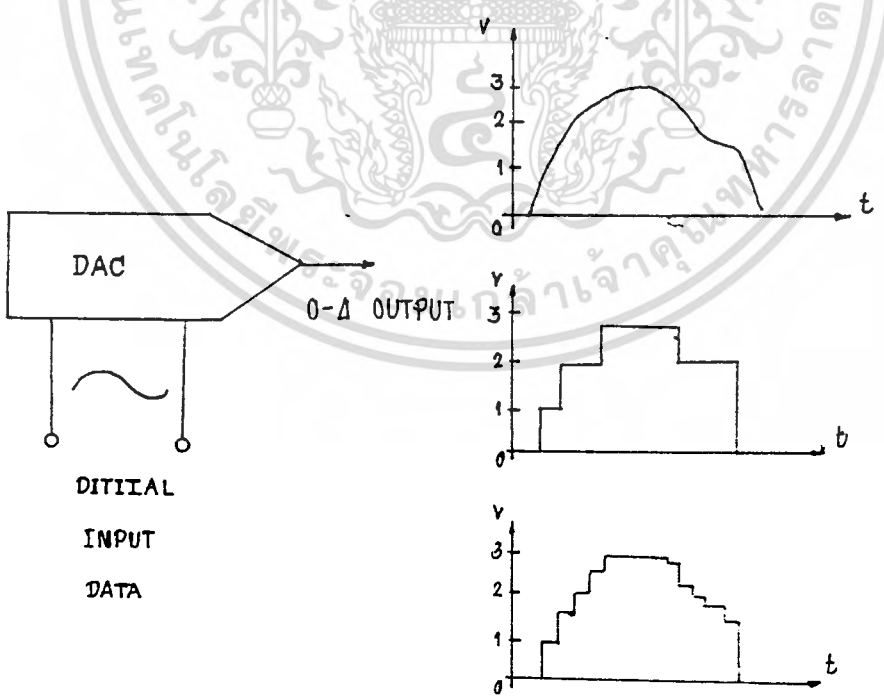
บทที่ 6

D/C คอนเวอร์ทเตอร์ (Digital-to-Analog converters)

D/A คอนเวอร์ทเตอร์ หรือเรียกย่อ ๆ ว่า ตัว DAC เป็นตัวแปลงรหัสเลขฐานสองรองจากคอมพิวเตอร์ หรือจากวงจรดิจิตอลใด ๆ ให้หลายเป็นระดับแรงดันอะนาล็อก ที่มีความสัมพันธ์กับระบบเลขฐานสองตัว DAC สามารถนำไปใช้ขับอุปกรณ์ที่เป็นอะนาล็อกได้ เช่น มิเตอร์ , มอเตอร์ อุปกรณ์ควบคุม หรือวงจรที่เกี่ยวกับสัญญาณเสียง เช่น เครื่องเล่นคอมแพ็คดิสก์ ตัว DAC ในเครื่องเล่นคอมแพ็คนั้นถูกใช้สำหรับการเปลี่ยนข้อมูลที่บันทึกเป็นสัญญาณดิจิตอลบนแผ่น CD ให้กลายเป็นสัญญาณเสียงที่มีคุณภาพสูง ออกมาให้เราได้ยิน

ต่อไปเราจะพิจารณาแนวความคิดที่สำคัญของ D/A เริ่มจากความละเอียดของ DAC เราจะนิยมไว้เป็น ระดับแรงดันในแต่ละขั้น ที่เอ้าท์พุทสามารถจะผลิตออกมาได้ ซึ่งมีความสัมพันธ์โดยตรงต่อจำนวนของบิททางด้านอินพุทที่อยู่ในรูปของรหัสไบนารี DAC ขนาด 4 บิท จะมีอินพุทบิทอยู่ 4 อินพุท ซึ่งจะมีความละเอียดเท่ากับ 4 จำนวนของระยะและความแตกต่างของระดับสัญญาณอะนาล็อกทางด้านเอ้าท์พุท ที่ DAC ขนาด 4 บิท สามารถผลิตได้ด้วยระดับแรงดัน 16 ขั้นด้วยกัน

ที่นี้มาดู DAC ขนาด 8 บิท ๆ สามารถให้สัญญาณอะนาล็อกทางด้านเอ้าท์พุทที่เป็นระดับแรงดันได้ 2^8 หรือ 256 ระดับ DAC ขนาด 12 บิท สามารถในระดับแรงดันทางเอ้าท์พุทได้ 2^{12} หรือ 4096 ระดับ อย่างที่เราได้เห็นแล้วว่า ADC มีขนาดอินพุทบิทมากเท่าไร ความละเอียดและความถูกต้องของระดับแรงดันอะนาล็อกทางเอ้าท์พุทที่ DAC สามารถผลิตได้จะมากขึ้นตามดังแสดงในรูปที่ 1



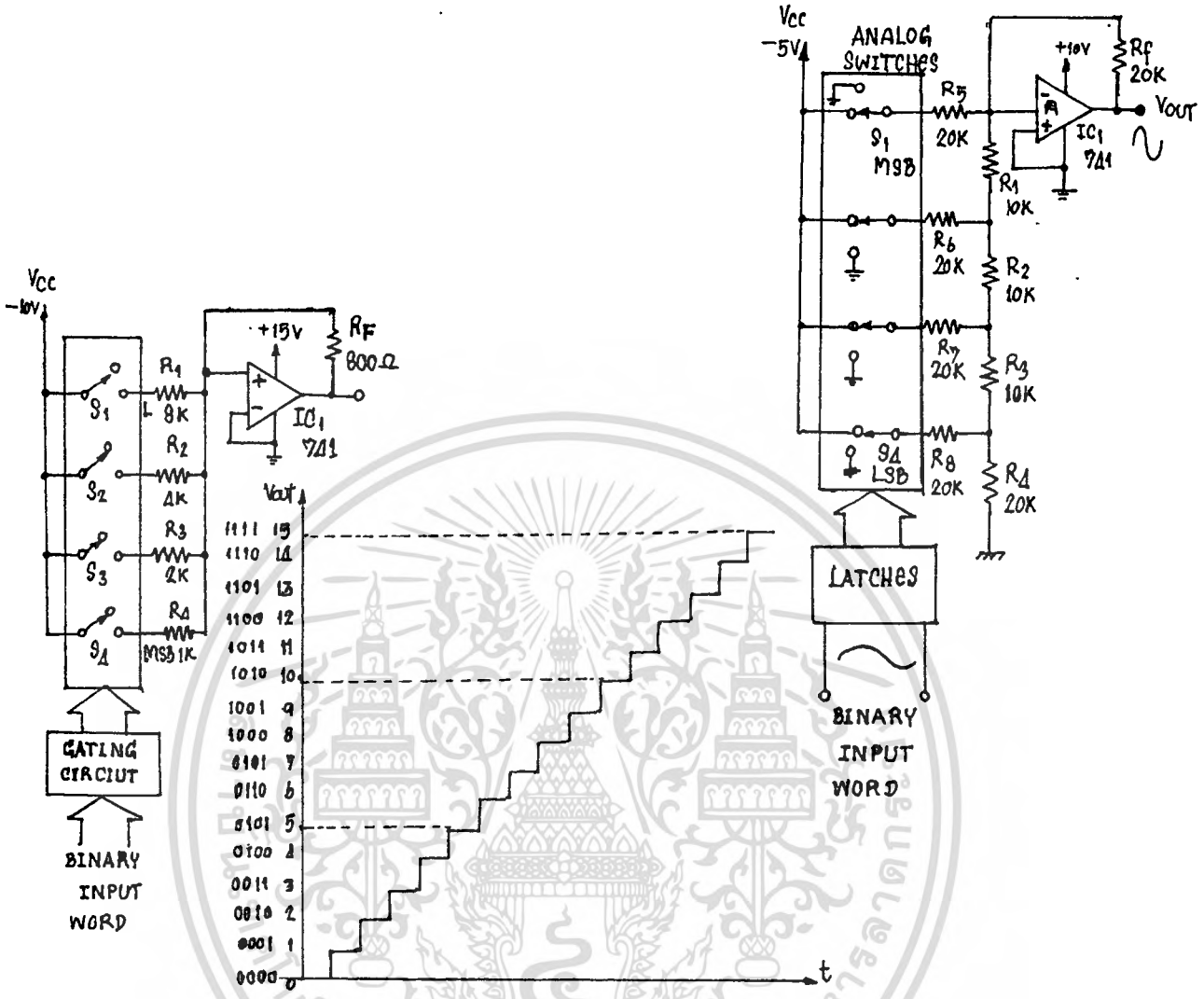
รูปที่ 1 แสดงความละเอียดของแรงดันทางเอ้าท์พุท ตัว DAC ยิ่งมีอินพุทมากเท่าไร ความละเอียดทางเอ้าท์พุทจะมากขึ้นตาม

ถัดมาจากความละเอียดของ DAC เราจะมาพิจารณาถึง เวลาเข้าสู่สภาวะของตัว (Setting time) เวลาเข้าสู่สภาวะคงตัว เป็นค่าของเวลาที่ระดับแรงดันเอ้าท์พุทเข้าสู่สภาวะคงที่เมื่อรหัสไบนารีทางอินพุทเปลี่ยนแปลงไป โดยปกติจะคิดที่สัญญาณทางเอ้าท์พุทคงที่ ในช่วง $-^{+}$ $^{1}/2$ ของ LSB (Least significant bit) ของค่าที่คาดว่าจะเป็นหลังจากไบนารี่ทางด้านอินพุทเปลี่ยนไป นั้นหมายความว่าในเงื่อนไขของการปฏิบัติงานจริง ๆ มีความสัมพันธ์กับค่าซึ่งเป็นอยู่ในขณะนั้นต่อ LSB ของมันเอง ถ้า DAC ขนาด 8 บิท มีแรงดันทางเอ้าท์พุทอยู่ในช่วง 0-10 โวลท์ ดังนั้น LSB มีค่าเท่ากับ $10/2^{8}$ หรือ 0.039 โวลท์ ครึ่งหนึ่งของค่า $10/2^{8}$ เป็น 0.0195 โวลท์ ดังนั้น ค่าเวลาที่ระดับแรงดันเอ้าท์พุทเข้าสู่สภาวะคงที่ควรจะเป็นค่าเวลาที่เอ้าท์พุทเพิ่มขึ้นถึง 0.0195 โวลท์ ของค่าระดับที่คาดหมายไว้ ตามปกติค่าเวลาเข้าสู่สภาวะคงตัวมีค่าน้อยกว่า 10 μs ค่าความแม่นยำเป็นแฟกเตอร์ที่สำคัญอีกตัวหนึ่งของ DAC ในเงื่อนไขปกติ ค่าความแม่นยำของ DAC คือ $-^{+}$ ทุก ๆ ตำแหน่งจาก $^{1}/2$ ถึง 2 ค่า LSB แรงดันเอ้าท์พุทสามารถเปลี่ยนแปลงไปในทาง + หรือ - ค่าของ 1 บิท ถ้า DAC มีแรงดันเอ้าท์พุทอยู่ในช่วง 0 ถึง 5 โวลท์ มีความละเอียดเท่ากับ 12 บิท LSB ควรจะเป็น $^{5}/2^{12}$ หรือ 0.00122 โวลท์ สำหรับทุก ๆ ค่าของรหัสไบนารีทางด้านเอ้าท์พุทแรงดันอาจจะสูงหรือต่ำกว่าค่าที่คาดหมายไว้ 0.00122 โวลท์ ถ้า DAC ตัวเดียวกันมีค่าความแม่นยำเท่ากับ $^{1}/2$ ค่าความถูกต้อง LSB ค่าเอ้าท์พุทจะสามารถผิดพลาดได้ $-^{+}$ 0.00122/2 หรือ $-^{+}$ 0.00061 โวลท์ ยิ่งค่าความแม่นยำน้อยเท่าไร ค่าความละเอียดก็จะมากขึ้นตามและจะมีค่าใกล้เคียงกับค่าเอ้าท์พุทที่คาดไว้

หลายปีที่ผ่านมาได้มีการค้นคิดพัฒนาวิธีการของการเปลี่ยนสัญญาณดิจิตอลไปเป็นอะนาล็อก 2 วิธีด้วยกัน : Binary weighted และ Binary ladder D/A

Binary -weighted resistor D/A

เทคนิคจัดน้ำหนักของรหัสไบนารี่เป็นวิธีที่ง่ายที่สุดและเก่าที่สุดของการแปลงดิจิตอลบิทให้กลายเป็นสัญญาณอะนาล็อก วงจรของ Binary-weighted resistor D/A แสดงไว้ในรูปที่ 2



รูปที่ 2 แสดงวงจร Biary weighted DAC และกราฟแสดงเอ้าท์พุทของ DAC ต่อสัญญาณดิจิตอล

รหัสไบนารี่จะถูกป้อนให้ขาเขตของอะนาล็อกสวิทช์ เมื่อรหัสไบนารี่เป็น 0000 ถูกป้อนให้กี่เกท อะนาล็อกสวิทช์ทั้งหมดจะเปิดวงจร ดังนั้น จึงไม่มีแรงดันเอ้าท์พุทจ่ายไปให้ออปแอมป์ เอ้าท์พุทจากออปแอมป์จึงเป็นศูนย์ เมื่อรหัสไบนารี่เป็น 0001 สวิทช์ S_1 จะปิดลง และแรงดัน 10 โวลท์จะจ่ายให้กับ R_1 เพราะว่าอินพุทของออปแอมป์จะมองได้ว่าเป็นกราวด์เสมือน (Virtual Ground) เป็นผลให้แรงดัน 10 โวลท์ ตกคร่อมตัวต้านทาน 8 k (R_1) ซึ่งเป็นเหตุให้เกิดกระแส 1.25 mA (10 V/8000) ไหลผ่านความต้านทานป้อนกลับ (R_f) ค่า 800 แรงดันตกคร่อม R_f ควรจะมีค่า 800 x 1.25 mA หรือเท่ากับ 1 โวลท์

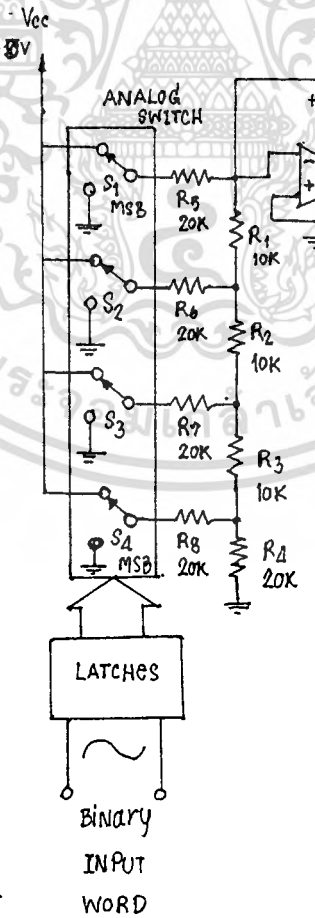
เมื่อรหัสไบนารี่เปลี่ยนไปเป็น 0010 S_1 จะเปิด และ 2.5 mA (10 V/4000) ไหลผ่าน R_2 แรงดันตกคร่อม R_f มีค่าเท่ากับ 800 x 2.5 mA หรือ 2 โวลท์ รหัสไบนารี่ 0100 จะให้แรงดันเอ้าท์พุทเท่ากับ 4 โวลท์ และรหัสไบนารี่เป็น 1000 แรงดันเอ้าท์พุทจะมีค่าเป็น 1000 และแรงดันเอ้าท์พุทจะมีค่าเป็น 8 โวลท์ จะสังเกตุได้ว่า รหัสทางอินพุทและค่าของ R_f มีผลต่อระดับแรงดันทางเอ้าท์พุท

สวิทช์แต่ละตัวสามารถปิดวงจรพร้อมกันได้เมื่อทำการร่วมกันเพื่อสร้างสัญญาณอะนาล็อกทางเอ้าท์พุทที่มีค่าจาก 0 ถึง 15 โวลท์ (0000 = 0 โวลท์ , 0111 = 7 โวลท์ และ 1111 = 15 โวลท์) ในการเพิ่มขึ้น 1 โวลท์ต่อ 1 ขั้น

ถึงแม้นว่า Binary - weighted resistor DAC มีลักษณะวงจรง่าย ๆ ตรงไปตรงมา แต่ไม่สะดวกในการนำไปใช้งาน ถ้าต้องการความละเอียดของ DAC มากกว่า 4 บิท เพราะว่าค่าของตัวต้านทานที่ใช้มากมายหลายค่าเกินไป ซึ่งต่างจาก Ladder network ที่ต้องการใช้ตัวต้านทานเพียง 2 ค่าเท่านั้น

Ladder network D/A

เทคนิคเลดเดอร์เน็ทเวิร์ด สามารถสร้างแรงดันตามน้ำหนักของรหัสไบนารี่โดยอาศัยความต้านทานเพียง 2 ค่าเท่านั้นที่จัดในลักษณะวงจรแบ่งแรงดัน หรือที่เรียกว่าไบนารี่ เลดเดอร์ (Binary ladder) ดังแสดงในรูปที่ 3



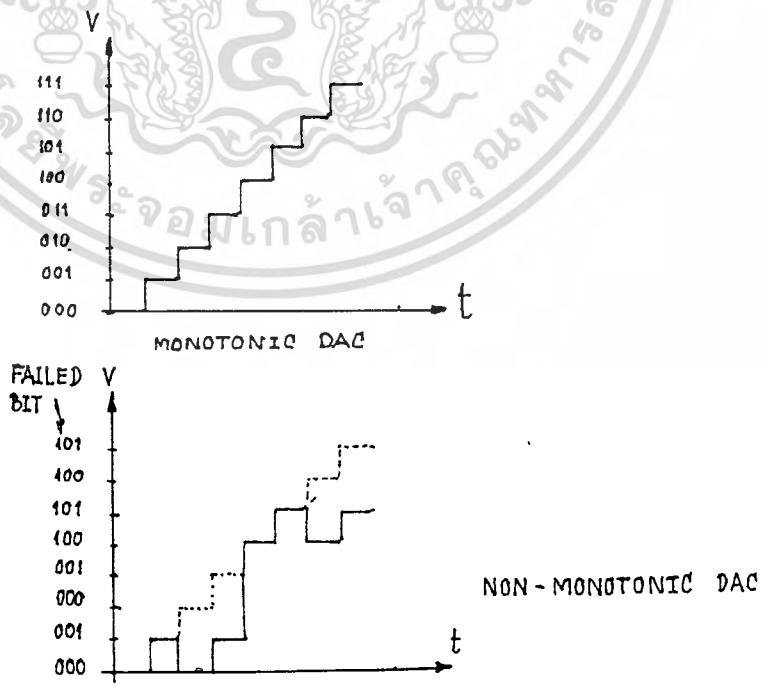
รูปที่ 3 แสดงวงจรของวงจร Binary Ladder DAC

ถึงแม้ว่าวงจร DAC แบบเลดเดอร์เน็ทเวิร์ดนั้นดูผ่าน ๆ แล้วค่อนข้างจะยาก แต่การทำงานของวงจรก็ยังคล้ายกับการทำงานของวงจร Binary weighted DAC (รูปที่ 2) เกทที่ต่ออยู่ในลักษณะอนุกรมถูกใช้สำหรับขับอะนาล็อกสวิทช์ทั้งหมดจะเปิดออก ดังนั้น แรงดันเอ้าท์พุทที่ได้จากออปแอมป์จึงมีค่าเป็นศูนย์ สวิทช์ S_1 จะปิดเมื่อเกทได้รับรหัสไบนารี่เป็น 1000 เป็นผลให้เกิดแรงดันเอ้าท์พุท 5 โวลท์ ออกมาจากออปแอมป์ถ้ารหัสไบนารี่เป็น 0010 สวิทช์ S_3 ก็จะปิด และทำให้เกิดแรงดัน 1.25 โวลท์ ที่เอ้าท์พุท และสุดท้ายถ้าอินพุทเป็น 0001 สวิทช์ S_4 จะปิดลง ออปแอมป์จะให้แรงดันเอ้าท์พุทออกมา 0.0625 โวลท์ จะสังเกตุได้ว่า แต่ละแรงดันเอ้าท์พุทอยู่ในรูปอันดับของไบนารี่ คือเอ้าท์พุทสามารถเปลี่ยนจาก 0 ถึง 10 โวลท์ เพิ่มขึ้นขั้นละ 0.625 โวลท์ (24 หรือ 16 ขั้น)

ข้อดีของเลดเดอร์เน็ทเวิร์ด DAC คือ สามารถออกแบบได้ง่าย เนื่องจากใช้ความต้านทานเพียง 2 ค่าเท่านั้น และในทุกวันนี้ บริษัทผู้ผลิต DAC เกือบทั้งหมดจะใช้เทคนิคแบบเลดเดอร์ เน็ทเวิร์ด ในการผลิต DAC

วงจรเลดเดอร์ มักจะมีความถูกต้องแม่นยำมากกว่าวงจร binary weighted เพราะว่า เราจะหาค่าความต้านทานที่ถูกต้อง 2 ค่า (เช่น 10 K หรือ 20 K) ได้ง่ายกว่าความต้านทานหลาย ๆ ค่า ที่ใช้ในวงจร Binary weighted DAC ไอซี DAC สำเร็จรูปที่นิยมใช้อยู่เบอร์ DAC-08

ที่นี้เราจะอธิบายหลักการเริ่มการทำงานของ DAC เราสามารถพิจารณารายละเอียดที่สำคัญสุดท้ายของ monotonicity (ดูความหมายในเทอมและนิยาม) ตามที่คุณได้รู้มา แรงดันเอ้าท์พุทที่เป็นสัญญาณอะนาล็อกของ DAC จะเพิ่มขึ้นเป็นลำดับคล้ายกับการเพิ่มของรหัสไบนารี่ทางอินพุทดังแสดงอยู่ในรูปที่ 4



รูปที่ 4 Monotonic DAC จะมากขึ้น ทุก ๆ ค่า ของสัญญาณอะนาล็อกที่ถูกต้องของค่ารหัสไบนารี่ทางอินพุท

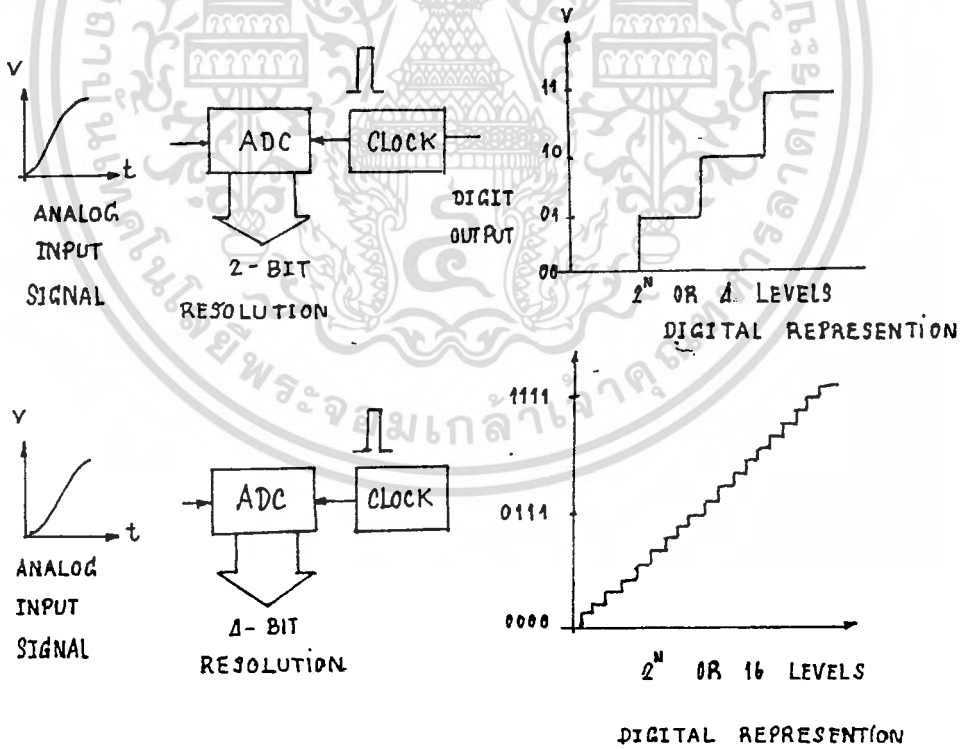
ในทางอุดมคติแล้ว การเพิ่มขึ้นของสัญญาณอินพุทที่รหัสไบนารี่เป็นเหตุให้เราสามารถทายและรู้ถึงแรงดันเอ้าท์พุทได้ ในอุปกรณ์บางตัว ส่วนของการสวิทช์ชิ่งและส่วนขยายสัญญาณไม่สามารถจ่ายกระแสไฟฟ้าภายในเงื่อนไขเหล่านั้นได้อย่างเพียงพอ จึงเป็นสาเหตุที่ทำให้ DAC เกิดการ " Skip "หรือการกระโดดข้ามขั้นนั้นเอง การเกิด Skip นี้จะมีปัญหาน้อยในบิทนัยต่ำ ๆ แต่จะมีมากขึ้นเมื่อน้ำหนักของบิทเพิ่มขึ้น

บทที่ 7

A/D คอนเวอร์ทเตอร์ (Analog-to-digital converters)

A/D คอนเวอร์ทเตอร์ หรือ ADC ใช้สำหรับการแปลงสัญญาณอินพุทที่เป็นอะนาล็อก ให้เป็นจำนวนจำกัดของดิจิตอลบิท ผลลัพท์ที่ได้จะอยู่ในรูปของ " word " ทางดิจิตอลซึ่งจะกลายเป็นรหัสเลขฐานสองที่แทนระดับ แต่ละระดับของสัญญาณอะนาล็อกในขณะที่ DAC กำลังทำงานแปลงสัญญาณอยู่

ความละเอียดของ ADC จะคล้ายกับความละเอียดของ DAC อย่างมาก ๆ กล่าวคือ จำนวนบิทางเอ้าท์พุทมีหลาย ๆ บิท ความละเอียดของ ADC ตัวนั้นก็จะมีมากขึ้นเช่น ADC ขนาด 12 บิท จะมีความละเอียดเท่ากับ 12 เป็นต้น ดังแสดงในรูปที่ 5



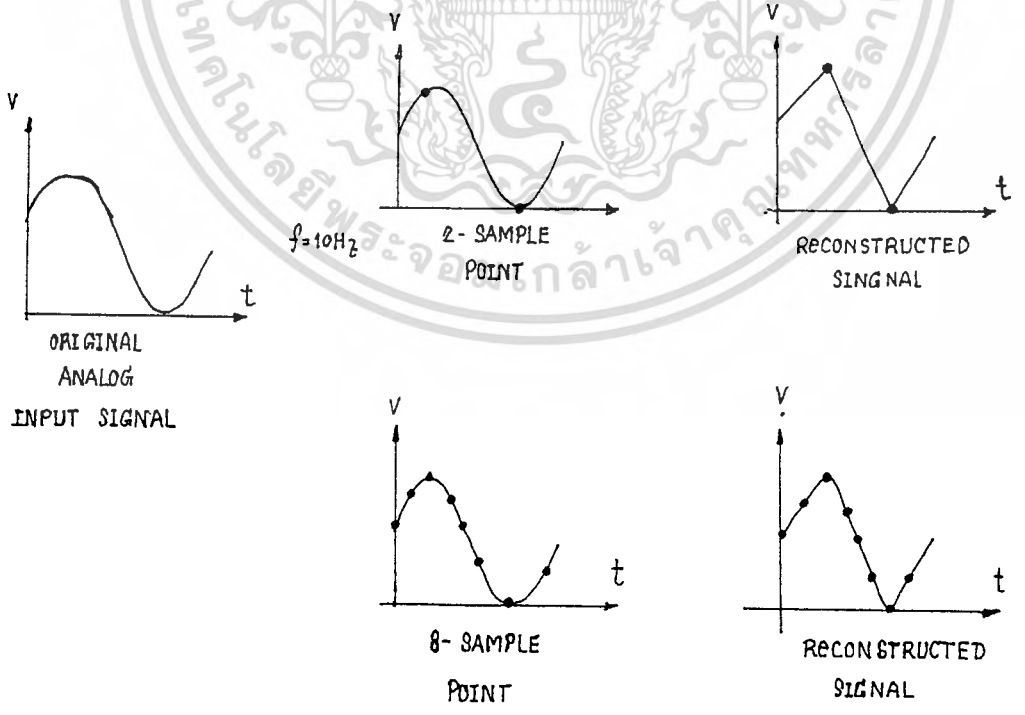
รูปที่ 5 ความละเอียดของ ADC จะแปรผันตรงกับจำนวนบิททางด้านเอ้าท์พุท

ค่าเวลาในการแปรผัน (CONVERSION TIME) เป็นเกณฑ์ที่สำคัญอีกตัวหนึ่งของ ADC ตามที่คุณได้เห็นมาว่า การแปลงสัญญาณอะนาล็อกให้กลายเป็นสัญญาณทางดิจิตอลไม่ได้เกิดขึ้นโดยทันทีทันใด แต่ต้องมีการผ่านขบวนการต่าง ๆ ด้วยเหตุที่ผลลัพท์ต้องเวลาค่าเวลาขณะหนึ่งที่จะทำการสุ่ม (Sample) สัญญาณอินพุทและสัญญาณดิจิตอลที่เป็นรหัสไบนารี่ออกมาที่เอ้าท์พุท ดังนั้น ค่าเวลาการแปรผันคือ ช่วงเวลาที่ต้องการกระทำขบวนการให้เสร็จสิ้น ซึ่งมีค่าอยู่ประมาณ μS สำหรับ DAC ความเร็วสูงและเป็น ms สำหรับ DAC แบบธรรมดาเนื่องจากการเปลี่ยน A/D นั้นต้องการกระบวนการซิงโครไนส์ ที่แน่นอนและแม่นยำ แหล่งกำเนิดสัญญาณนาฬิกาจึงจำเป็นต้องมีในวงจร

ทฤษฎีการสุ่มตัวอย่าง (Sampling theory)

เนื่องจาก ADC ต้องการค่าเวลาขณะหนึ่งใช้ในขบวนการเปลี่ยนแปลงสัญญาณอะนาล็อกเป็นสัญญาณดิจิตอลช่วงเวลาช่วงหนึ่งจะใช้สำหรับการสุ่มตัวอย่าง ของสัญญาณตัวอย่าง เช่น ADC สามารถเปลี่ยนสัญญาณเสร็จสมบูรณ์ได้ผ่านในเวลา 1 ms ดังนั้นมันจึงเปลี่ยนสัญญาณได้ 1000 ครั้งใน 1 วินาที (ในทางทฤษฎี) อัตราการเปลี่ยนแปลงสัญญาณสูงสุดมีค่าเท่ากับส่วนกับของเวลาการเปลี่ยน (conversion rate = ^{1}/conversion time)

ตัวคอนเวอร์ทเตอร์จะสุ่มตัวอย่างของสัญญาณด้วยอัตราต่ำสุดเป็น 2 เท่าของความถี่สูงสุดของสัญญาณอินพุทที่เข้ามา อัตราการสุ่มนี้เรียกว่า " Nyquist rate " พิจารณาสัญญาณอะนาล็อกที่เป็นรูปคลื่นซายน์ 10 Hz จ่ายในกับตัว ADC ตามรูปที่ 6



รูปที่ 6 การสุ่มหลาย ๆ ช่วงจากสัญญาณอินพุทอะนาล็อกจะมีลักษณะใกล้เคียงกับสัญญาณดังเดิมเมื่อมีการสร้างสัญญาณขึ้นมาใหม่ DAC

อัตราต่ำสุดของการสุ่มตัวอย่างเป็น 2f หรือ 20 Hz ซึ่งจะให้ข้อมูลดิจิตอลขนาด 2 บิท ออกมาในแต่ละไซเคิ้ล เมื่อข้อมูลดิจิตอลถูกนำมาสร้างเป็นสัญญาณอะนาล็อกขึ้นมาใหม่โดย DAC สัญญาณอะนาล็อกตัวใหม่มีลักษณะคล้ายกับสัญญาณดังเดิม (ตัวฟิลเตอร์บนตัว DAC จะทำให้รูปร่างของสัญญาณเอ้าท์พุทเรียบขึ้น) ค่าความถี่ 10 Hz เป็นความถี่สูงสุดที่เข้ามายังตัว ADC ค่าเวลาที่ใช้ในการเปลี่ยนสัญญาณสูงสุดเป็น 1 /20 Hz หรือ 500 ms เป็นต้น

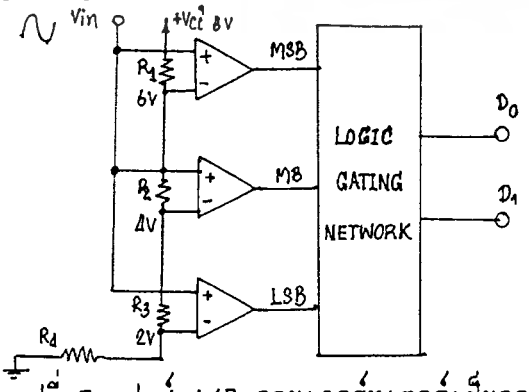
การที่เราจะปรับปรุงประสิทธิภาพของ ADC ในแง่ความเหมือนจริงของสัญญาณการแปลงให้อยู่ในรูปของดิจิตอลเราจะต้องเพิ่มอัตราการสุ่มขึ้นในค่าเวลาเท่าเดิม อัตราการสุ่ม 8 จุดต่อไซเคิ้ล ต้องการอัตราสุ่มของส่วนประกอบความถี่สูงสุดอินพุท 8 ครั้งเช่น ความถี่อินพุท 10 Hz จะต้องสุ่มตัวอย่างที่ 80 Hz ดังนั้นตัวคอนท์เวอร์เตอร์ควรจะมีค่าการเปลี่ยนเป็น 1/80 Hz หรือ 12.5 ms ถ้าตัว ADC ไม่สามารถสุ่มตัวอย่างได้เร็วพอต่อสัญญาณอินพุทที่เปลี่ยนแปลงไปข่าวสารที่บรรจุในสัญญาณอะนาล็อกทางอินพุทจะสูญหายไป

ค่าความสัมพันธ์ระหว่างความถี่ทางอินพุทค่าเวลาในการเปลี่ยนสัญญาณและอัตราการสุ่ม เป็นพารามิเตอร์ของ ADC ที่สำคัญตัวหนึ่งวิธีการหลาย ๆ วิธีได้ถูกพัฒนาหลาย ๆ ปีผ่านมา เพื่อที่จะทำการเปลี่ยนสัญญาณอะนาล็อกให้อยู่ในรูปของสัญญาณดิจิตอล หลาย ๆ วิธีที่ยังใช้อยู่ทุกวันนี้มี 6 วิธีด้วยกันคือ

1. Flash techniques เทคนิคแบบแฟลช์
2. Single slope techniques เทคนิคแบบสโลปเดียว
3. Double slope techniques เทคนิคแบบสโลปคู่
4. Single counter techniques เทคนิคแบบเคาเตอร์เดียว
5. Tracking counter techniques เทคนิคแบบเทร็คกิ้งเคาเตอร์
6. Successive approximation techniques เทคนิคการประเมินค่าหลาย ๆ ครั้ง

A/D แบบแฟลช์ (FLASH CONVERSTOR)

แฟลช์คอนเวอร์ทเตอร์เป็น ADC ที่เร็วที่สุดในบรรดา ADC ที่ใช้เทคนิคแบบอื่น ๆ ลักษณะของแฟลช์คอนเวอร์ทเตอร์ จะใช้ชุดของตัวเปรียบเทียบ (comparator) ที่ก่อในลักษณะขนานกันเพื่อทำการแปลงสัญญาณอะนาล็อกทางอินพุทให้เป็นรหัสดิจิตอลดังนั้น แฟลช์คอนเวอร์ทเตอร์เป็นแบบขนานนิจารณาในรูปที่ 7



V_{IN}	BINARY BIT O/P		COMPARATOR O/P		
	D_1	D_0	MSB	MB	LSB
0-2V	0	0	0	0	0
2-4V	0	1	0	0	1
4-6V	1	0	0	1	1
6-8V	1	1	1	1	1

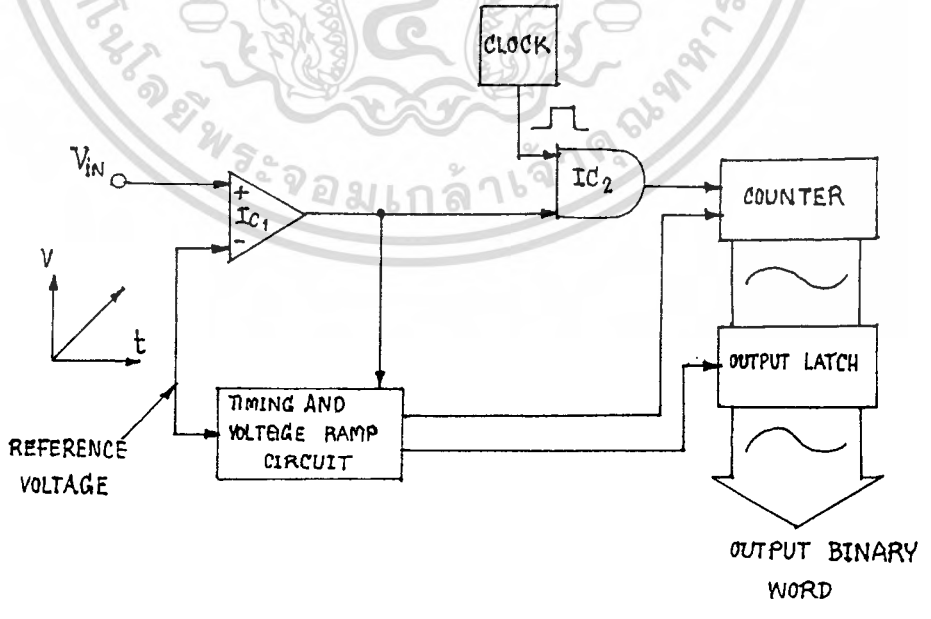
รูปที่ 7 แฟลช์ A/D คอนเวอร์ทเตอร์เป็นคอนเวอร์ทเตอร์ที่ความเร็วในการเปลี่ยนแปลงสัญญาณสูงและมีลักษณะวงจรง่ายความละเอียดต่ำ

จากรูปที่ 7 ตัวต้านทานที่ต่ออนุกรมกันจะอยู่ในรูปวงจรแบ่งแรงดันที่ตกคร่อมตัวเปรียบเทียบแต่ละตัว แรงดันอินพุทสูงสุดจะขึ้นอยู่กับค่า VCC สัญญาณเอ้าท์พุทจากตัวเปรียบเทียบแต่ละตัวจะเป็น 1 หรือ 0 ซึ่งเป็นระดับสัญญาณโลจิตของวงจรดิจิตอล เมื่อไม่มีแรงดันอินพุทเข้ามาเอ้าท์พุทของตัวเปรียบเทียบแต่ละตัวจะเป็นโลจิก " 0 " ต่อมา แรงดันอินพุทเพิ่มขึ้น เอ้าท์พุทของตัวเปรียบเทียบแต่ละตัวจะเป็นโลจิก " 1 " ไล่ตามลำดับขึ้นไป เมื่อแรงดันอินพุทมีมากกว่าแรงดันอ้างอิงแต่ละค่าที่ถูกเซทโดยวงจรแบ่งแรงดัน เน็ทเวอร์คของดิจิตอลเกท ถูกใช้ในการเรียงลำดับของสัญญาณจากตัวเปรียบเทียบ ให้อยู่ในรูปรหัสเลขฐานสองซึ่งเป็นการสร้างรหัสที่เอ้าท์พุทของคอนเวอร์ทเตอร์

ถ้าเราสังเกตุวงจรในรูปที่ 7 ให้ดีจะพบว่า วงจรที่ใช้ตัวเปรียบเทียบ 2^n-1 เป็นตัวแสดงความละเอียดของคอนเวอร์ทเตอร์จากตัวอย่างคอนเวอร์ทเตอร์ขนาด 2 บิท ของเรา ต้องการ 2^2-1 เท่ากับ 3 และคอนเวอร์ทเตอร์ขนาด 8 บิท ต้องใช้ตัวเปรียบเทียบถึง 2^8-1 ตัว หรือ 255 ตัว แฟลช์คอนเวอร์ทเตอร์มิได้มีเพียงข้อเสียของมันเพียงอย่างเดียว แต่มันยังมีของดีที ADC แบบอื่น ๆ ไม่มีหรือสู้ไม่ได้คือความเร็วช่วงเวลาในการเปลี่ยนจึงมีค่าเท่ากับเวลาหน่วงในตัวเปรียบเทียบแต่ละตัว และวงจรเกทในวงจรเท่านั้น ซึ่งในเวลาไม่เพียงไมโครเซค

ADC แบบสโลปเดียว (Single-Slope ADC)

การแปลงสัญญาณอะนาล็อกให้เป็นสัญญาณดิจิตอลที่มีประสิทธิสูงวิธีหนึ่งคือ วิธีแบบ A/D สโลปเดียว หรือเรียกว่า A/D แรมป์เดียว (Single-ramp A/D) ดังแสดงไว้ในรูปที่ 8



รูปที่ 8 ตัวผลิตสัญญาณแรมป์ ตัวเปรียบเทียบและตัวนับ เป็นส่วนประกอบที่สำคัญของ A/D คอนเวอร์ทเตอร์แบบสโลปเดียว

การทำงานของวงจร A/D สโลปเดียวเริ่มต้นการรีเซทวงจรนับและแรงดันแรมป์อยู่ที่ 0 เอ้าท์พุท ของตัวเปรียบเทียบที่จุดเริ่มต้นเป็น "0" ดังนั้นจึงไม่มีสัญญาณนาฬิกาจ่ายให้กับวงจรนับ เมื่อแรงดันอินพุท ถูกจ่ายให้กับตัวคอนเวอร์ทเตอร์ขาอินพุท นอนอินเวอติ้ง (+) จะมีค่าแรงดันเกินกว่าแรงดันขาอินพุทอินเวอติ้ง (-) ดังนั้นเอ้าท์พุทของตัวเปรียบเทียบจึงเป็นโลจิก " High " สัญญาณโลจิก " High " นี้ไปเอนาเบิ้ล ให้ and เกททำงานให้ยอมรับพัลส์ของสัญญาณนาฬิกาผ่านตัวมันเข้าไปยังวงจรนับเลขฐานสองให้ทำงาน ขณะเดียวกัน วงจรไทม์มิ่ง จะขับให้แรงแรมป์เพิ่มขึ้น ซึ่งทำให้แรงดันอินพุทที่ขาอินเวอติ้งของตัวเปรียบเทียบเพิ่มขึ้นอย่างรวดเร็ว เมื่อแรงดันแรมป์อ้างอิงเริ่มมากกว่าแรงดันอินพุทเอ้าท์พุทของตัวเปรียบเทียบจะตกลงเป็นโลจิก " low " อีกครั้ง วงจรนับจะเกิดการค้าง (latch) ค่าที่นับขณะหนึ่งต่อมาทำการรีเซตตัวนับสำหรับวัฏจักรการแปลงสัญญาณช่วงต่อไป

เมื่อแรงดันแรมป์อ้างอิงมีค่าเท่ากับแรงดันอินพุทที่จ่ายเข้ามาวงจรนับจะถูกกระตุ้น ให้นับเลขฐานสอง ขณะเดียวกันค่าที่นับได้จึงเป็นสัญญาณดิจิตอลของสัญญาณอะนาล็อกทางด้านอินพุทที่เข้ามาในขณะนั้น จะสังเกตุได้ว่าความเร็วของสัญญาณนาฬิกาและอัตราการเพิ่มขึ้นในลักษณะเป็นแรงดันแรมป์จะต้องมีความสัมพันธ์กันอย่างถูกต้อง เพื่อให้วงจรนับทำงานตามหน้าที่ได้อย่างถูกต้องแน่นอน ค่าเวลาที่ต้องการทำการเปลี่ยนขึ้นอยู่ระดับสัญญาณอะนาล็อกทางอินพุท เพราะว่าวงจรนับและแรงดันแรมป์อ้างอิงทั้งคู่เริ่มต้นจากศูนย์ที่ทุก ๆ วัฏจักรการแปรผัน มันจึงใช้เวลาค่อนข้างนานที่จะทำให้แรงดันอ้างอิงเท่ากับแรงดันอินพุท ในทางตรงข้าม ถ้าแรงดันอินพุทมีค่าน้อย ช่วงเวลาที่แรงดันแรมป์ที่อ้างอิงเพิ่มขึ้นจนเท่ากับแรงดันอินพุทจึงใช้เวลาน้อยกว่ากรณีแรงดันอินพุทมีค่ามาก ๆ

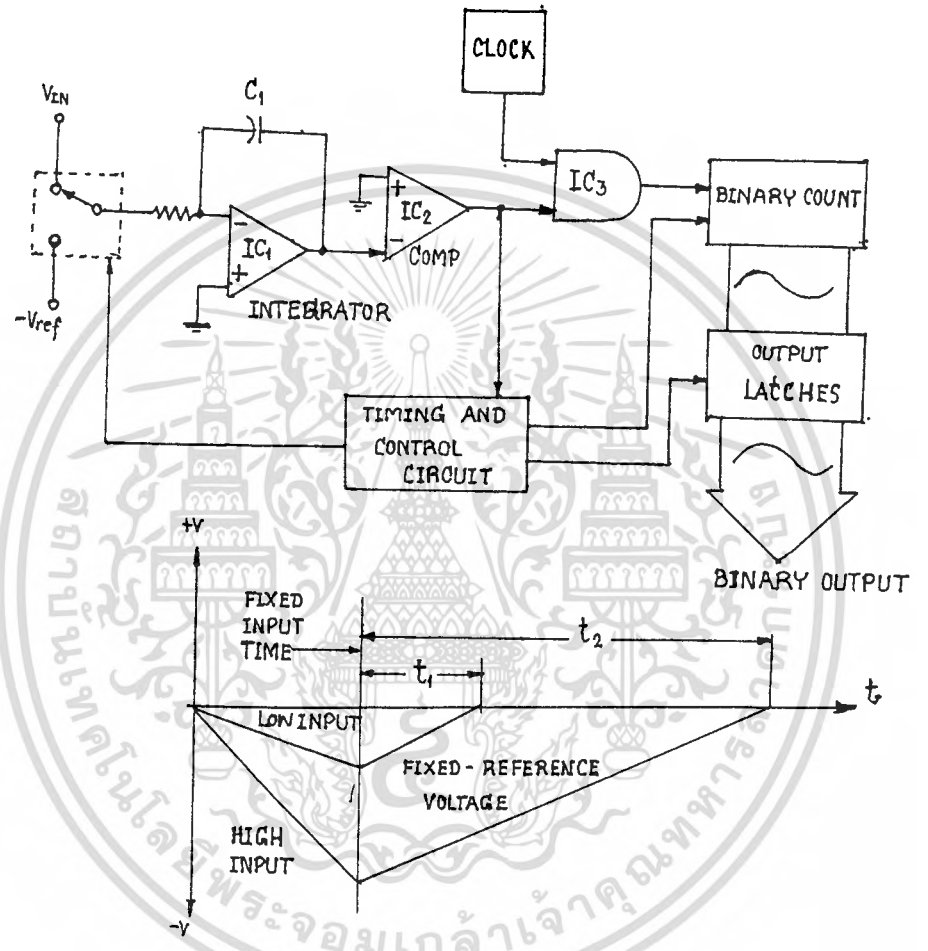
แรงดันแรมป์อ้างอิงสามารถเปลี่ยนแปลงมากขึ้นจนเท่ากับแรงดันอินพุทได้ เร็วกว่า 1 โวลท์ ต่อ 1 /1000 วินาที เช่นถ้าแรงดันอินพุทเป็น 2 โวลท์ ถูกจ่ายให้กับวงจรในรูปที่ 8 วงจรจะใช้เวลา 2 x 1 Volt/ms ซึ่งเท่ากับ 2 ms สำหรับแรงดันแรมป์ที่จะเพิ่มขึ้นจนมีแรงดันเท่ากับแรงดันอินพุท การนับเลขฐานสองจะกระทำหลังจาก 2 ms ไปแล้ว ความเร็วในช่วงนี้ขึ้นอยู่กับความเร็วสัญญาณนาฬิกามีค่าสูงจะทำให้จังหวะในการนับเร็วขึ้นด้วย

เนื่องจากการทำงานของสัญญาณนาฬิกาขึ้นอยู่กับแรงดันแรมป์ จึงเป็นลักษณะพิเศษของ A/D สโลปเดี่ยว ที่มีสัญญาณเอ้าท์พุทออกมาเป็นเลขฐานสองโดยตรง ไอซีเครื่องมือวัดบางตัวที่ใช้เทคนิคแบบสโลปเดี่ยวนี้จะแปลงรหัส BCD ไปขับภาคแสดงผล 7 เซ็คเมนท์ได้โดยตรง ซึ่งทำให้สะดวก และข้อได้เปรียบกว่าเทคนิค A/D แบบอื่นอย่างมาก

ข้อเสียของ A/D สโลปเดี่ยว คือ การทำงานที่ไม่ค่อยมีเสถียรภาพเมื่อใช้งาน A/D เป็นเวลานาน ๆ โดยปราศจากการประสานจังหวะ (Synchroninzation) ระหว่างวงจรผลิตสัญญาณนาฬิกาและวงจรสร้างสัญญาณแรมป์ ทุก ๆ การเลื่อนของความเร็วสัญญาณนาฬิกา หรือแรงดันแรมป์ เป็นเหตุทำให้เกิดการผิดพลาดที่รหัสทางเอ้าท์พุท จึงเป็นสาเหตุที่สำคัญที่ทำให้ A/D สโลปเดี่ยวไม่นำไปใช้งานที่ต้องการความถูกต้องสูง

A/D แบบสโลปคู่ (Double-Slope ADC)

เทคนิคการเปลี่ยนสัญญาณอะนาล็อกไปเป็นสัญญาณดิจิตอลแบบสโลปคู่ เป็นเทคนิคที่ให้ข้อดีทางเสถียรภาพของการแปลงสัญญาณ เมื่อสัญญาณอินพุทมีการเปลี่ยนแปลงอย่างรวดเร็ว วงจรผลิตสัญญาณแรมป์อ้างอิงได้ปรับปรุงขึ้นโดยตัดเอาผลกระทบของการเลื่อนไหล เมื่อใช้วงจรไปนาน ๆ สัญญาณอินพุทของตัวคอร์เวอร์ทเตอร์แบบสโลปคู่จะป้อนให้กับวงจรอินทิเกรตเตอร์ เมื่อสัญญาณอินพุทที่เป็นบวกถูกป้อนเข้ามายังตัว ADC



รูปที่ 9 ADC แบบสโลปคู่ เป็นคอนเวอร์ทเตอร์ที่ให้ประสิทธิภาพการทำงานสูงกว่า ADC แบบสโลปเดียว

ความชันของแรงดันแรมป์ทางด้านเอ้าท์พุทของวงจรอินทิเกทเตอร์มีทิศทางเป็นลบและจะมีค่าเป็นลบด้วยแรงดันลบที่ได้นี้ทำให้เอ้าท์พุทวงจรเปรียบเทียบเป็น " Hing " ด้วยเหตุนี้จึงเป็นการกระตุ้นให้เกิดสัญญาณนาฬิกาป้อนเข้าไปยังขั้วอินพุทของวงจรนับ ซึ่งเป็นการเริ่มต้นนับขึ้นไปเรื่อย ๆ วงจรอินทิเกทเตอร์จะให้สัญญาณแรมป์เพียงคาบเวลาที่คงที่ขณะหนึ่งเท่านั้น หลังจากช่วงเวลานี้แล้ว วงจรควบคุมทำการเคลียวงจรนับ เอ้าท์พุทของวงจรเปรียบเทียบจะกลายเป็น " low " ซึ่งเป็นสัญญาณนาฬิกาที่ป้อนวงจรนับหยุดลง วงจรควบคุมจะทำการตรวจสอบซึ่งเปลี่ยนและแลทช์ การนับที่เอ้าท์พุทไว้เราทำการเคลียวงจรอีกครั้ง

อัตราอินทิเกทขึ้นอยู่กับขนาดของแรงดันอินพุท เช่นเดียวค่าของ R_1 และ C_1 ดังนั้น แรงดันอินพุทที่ต่ำ ๆ จะลงของวงจรอินทิเกทเตอร์ให้น้อยกว่าแรงดันอินพุทที่มีค่าสูง ๆ ในช่วงคาบเวลาอินพุทที่แน่นอนของวัฏจักรการแปรผัน (Conversion cycle) ดังนั้น คอนเวอร์ทเตอร์แบบสโลปคู่มีประสิทธิภาพเหมาะสำหรับการประยุกต์ใช้งานที่มีความแม่นยำสูง

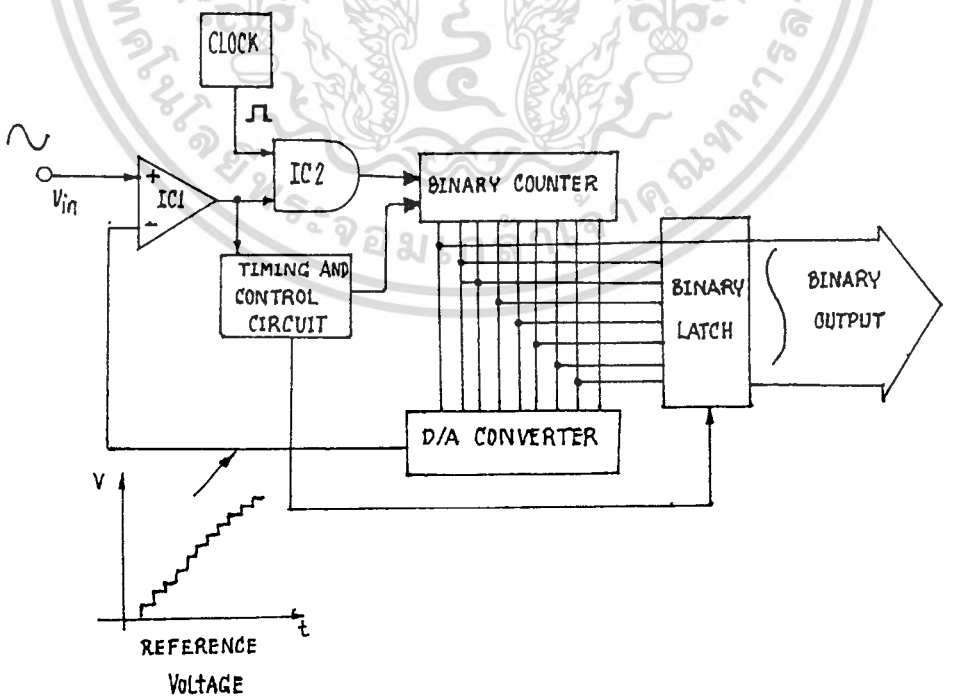
ข้อเสียของการแปรผันแบบสโลปคู่ คือ คาบเวลาที่ขยายออกไปที่ต้องการใช้ในการแปรผัน คอนเวอร์ทเตอร์แบบสโลปคู่ ต้องการคาบเวลาที่มากกว่า 100 ms ต่อการเปลี่ยนสัญญาณอินพุทที่มีแรงดันสูง ๆ ให้อยู่ในรูปสัญญาณดิจิตอล

DAC แบบมีการป้อนกลับ (D/A feedback converters)

D/A คอนเวอร์ทเตอร์ที่ใช้สัญญาณป้อนกลับมาเป็นสัญญาณอ้างอิงที่วงจรเปรียบเทียบมีสองชนิดคือ

1. วงจรนับแบบเดียว (Single counter)
2. วงจรนับแบบแทร็คกิ้ง (Tracking counter)

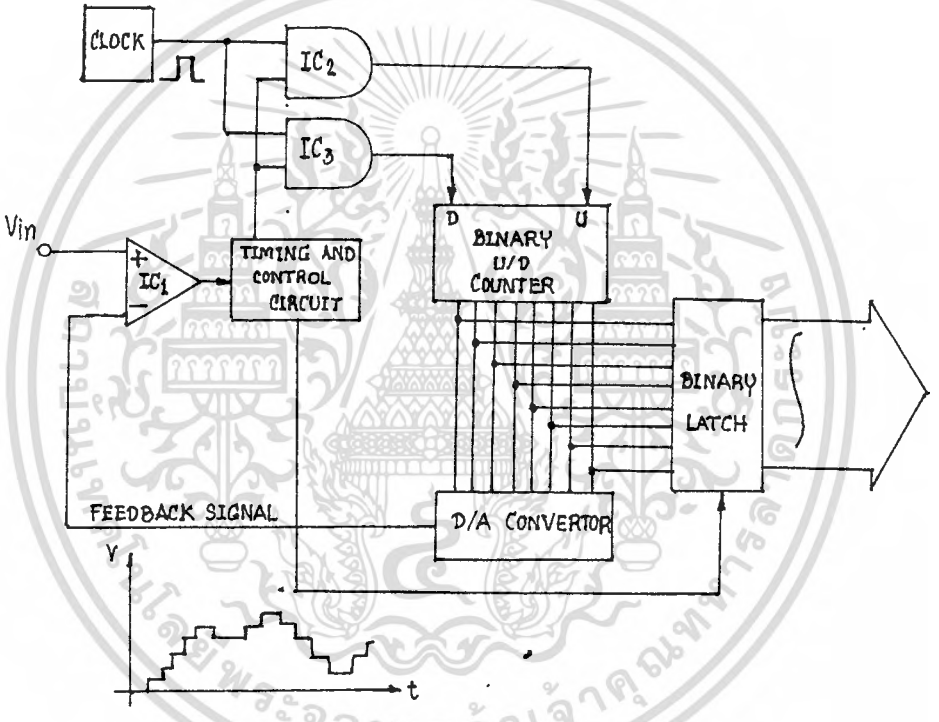
วงจรของนับ ADC แบบวงจรนับเดียวได้มีการพัฒนาจนมีลักษณะคล้ายคลึงกับ ADC แบบสโลปเดียวตลอดจนการทำงานของวงจรทั้งสองยังคล้ายกันอีกด้วย แต่ ADC แบบวงจรนับเดียวจะอ่านการนับสัญญาณนาฬิกาที่ได้จากวงจรนับเลขฐานสอง แล้วทำให้เป็นแรงดันป้อนกลับไปยังวงจรเปรียบเทียบแทนวงจรอินทิเกทเตอร์หรือแหล่งแรงดันแรมป์อื่น ๆ ดังรูปที่ 10



รูปที่ 10 D/A คอนเวอร์ทเตอร์ถูกใช้ในวงจร D/A คอนเวอร์ทเตอร์เพื่อที่จะเป็นตัวสร้างแรงดันแรมป์ไปควบคุมการทำงานของ A/D คอนเวอร์ทเตอร์

เมื่อสัญญาณอะนาล็อกทางอินพุทถูกจ่ายเป็นวงจรเปรียบเทียบ เอ้าท์พุทของมัน " High " ดังนั้น วงจรควบคุมจึงยอมให้สัญญาณนาฬิกาผ่านเข้าไปในวงจรนับ ๆ จะนับเลขฐานสองขึ้นไปเรื่อย ๆ วงจรควบคุมจะส่งสัญญาณไปยังวงจรแลทช์เลขฐานสอง (Binary latch) ให้ไปค้างค่าที่ได้จากวงจรนับที่เอ้าท์พุทไว้ หลังจากนั้นวงจรควบคุมจะทำการรีเซท วงจรนับสำหรับวัฏจักรการแปรผันต่อไป

เทคนิคของวงจรนับแบบแทร็คกิ้ง สามารถทำการแปรผันได้เร็วกว่าแบบวงจรนับเดียว ซึ่งเป็นการแปรผันที่รวดเร็วเพราะวงจรนับแทร็คกิ้งใช้วงจรนับเลขฐานสองแบบขึ้นลงได้แทนวงจรนับขึ้นอย่างเดียว เหมือนกับตัวอย่างที่ผ่านมา ดังรูปที่ 11



รูปที่ 11 วงจรนับเลขฐานสองแบบขึ้นลงทำให้ A/D คอนเวอร์ทเตอร์เปลี่ยนแปลงสัญญาณป้อนกลับตามสัญญาณอะนาล็อกทางอินพุทได้ซึ่งใช้ใน ADC แบบแทร็คกิ้ง

วงจรนับสามารถเพิ่มค่าขึ้นลงได้ขึ้นอยู่กับฐานะทางด้านเอ้าท์พุทของวงจรเปรียบเทียบ ซึ่งจะทำให้รหัสไบนารี่ที่ได้มีความเป็นจริงต่อสัญญาณอะนาล็อกมากขึ้น

การทำงานของ ADC แทร็คกิ้ง เริ่มต้นที่สัญญาณอะนาล็อกถูกป้อนมายังอินพุทของวงจรเปรียบเทียบ การนับบนวงจรนับเลขฐานสองขึ้นลง อาจจะเริ่มที่ค่าใด ๆ ก็ได้นั้นหมายถึง แรงดันป้อนกลับที่มาจากตัว DAC อาจจะมากกว่าหรือน้อยกว่าสัญญาณอะนาล็อกทางอินพุทก็ได้ ถ้าแรงดันป้อนกลับมีค่ามากกว่าสัญญาณอะนาล็อกทางอินพุท เอ้าท์พุทจะมีฐานะเป็น " low " และวงจรควบคุมจะส่งสัญญาณไปเปิดเกทให้พัลส์ของสัญญาณนาฬิกาผ่านไปยังวงจรนับ ดังนั้นจึงเป็นการลดค่าเลขฐานสองซึ่งเป็นเอ้าท์พุทของวงจรนับและเป็นการลงแรงดัน

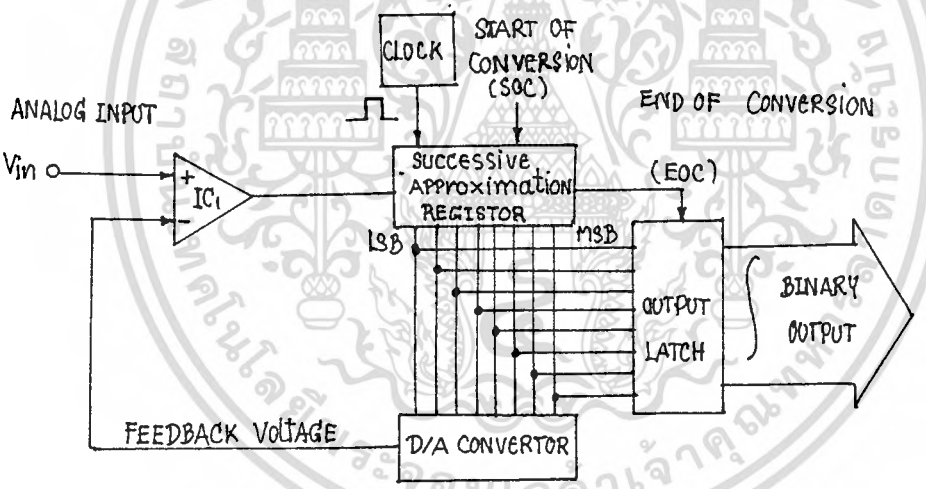
ป้อนกลับที่วงจรเปรียบเทียบ เมื่อแรงดันป้อนกลับตกลงต่ำกว่าแรงดันอินพุท เอ้าท์พุทของวงจรแบบเปรียบเทียบจะฐานะเป็น " High " ทันทีและวงจรควบคุมจะส่งสัญญาณไปยังวงจรแลทช์ทางด้านเอ้าท์พุทให้ค้างเอ้าท์พุทไว้ วงจรเกทจะส่งสัญญาณนาฬิกาไปเพิ่มเอ้าท์พุทของวงจรนับขึ้น เป็นเหตุให้วงจรนับค่าขึ้นอีกครั้ง

เทคนิคแบบวงจรนับแทร็คกิ้งนั้นมีความเร็วสูงกว่าเทคนิคแบบวงจรนับเดียว แต่มันยังมีข้อดีคือ เทคนิคแบบวงจรนับแทร็คกิ้งเหมาะสำหรับการแปรสัญญาณอินพุทที่มีการเปลี่ยนอย่างรวดเร็ว ให้อยู่ในรูปสัญญาณดิจิตอลได้ดี

ADC แบบประมาณค่าหลาย ๆ ครั้ง (Successive-approximation ADC)

เทคนิคการประมาณค่าหลาย ๆ ครั้ง เป็นเทคนิคที่มีความสามารถสูงและใช้งานได้ดีซึ่งสามารถแปลงสัญญาณอะนาล็อกให้เป็นสัญญาณดิจิตอลได้รวดเร็ว และมีประสิทธิภาพ เพราะไม่มีการออกชิลเลต

หัวใจของ SA คอนเวอร์ทเตอร์คือ อุปกรณ์ที่เรียกว่า "Successive-approximation registor" (SAR) ซึ่งเป็นอุปกรณ์ที่มีจุดประสงค์ต่างจากวงจรนับทั่ว ๆ ไปอย่างมากดังรูป 12



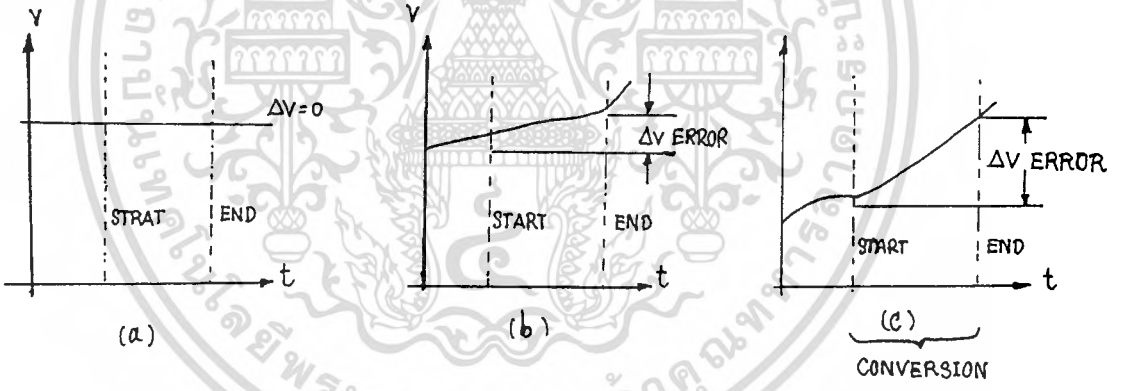
รูปที่ 12 SAR เป็นตัวเร่งความเร็วของกระบวนการแปลงสัญญาณอะนาล็อกเป็นดิจิตอล

การทำงานเริ่มต้นเมื่อสัญญาณอะนาล็อกถูกป้อนให้กับคอนเวอร์ทเตอร์ และพัลส์การแปลงเริ่มต้น (Start conversion pulse " SOC ") ถูกป้อนให้กับตัว SAR พัลส์สัญญาณนาฬิกาลูกแรกที่ป้อนให้กับตัว SAR จะ " on " เอ้าท์พุทของบิทนับสูงสุด ดังนั้นจะเป็นปรับเอ้าท์พุทของ DAC เป็น 50% ของแรงดันเอ้าท์พุทของอินเวอร์ทเตอร์ ตัว SAR จะมองไปยังเอ้าท์พุทของวงจรเปรียบเทียบว่า เอ้าท์พุทของ DAC มีค่ามากกว่าหรือน้อยกว่าสัญญาณอะนาล็อกทางอินพุท ถ้าแรงดันของ DAC มีค่ามากกว่า วงจรเปรียบเทียบจะยังคงอยู่ในสภาวะ " off " ดังนั้นตัว SAR จะ " off " บิทในนัยสูงสุดลงและให้ชื่อว่าสภาวะ " 0 " ถ้าแรงดันของ DAC มีค่าน้อยกว่าสัญญาณอะนาล็อกทางอินพุทของวงจรเปรียบเทียบจะยังคงทำงานอยู่ ดังนั้นตัว

SAR ยังปล่อยให้บิทนัยสูงสุด " on " อยู่ และเราเรียกว่าสภาวะหนึ่งนี้ว่า " 1 " ซึ่งสภาวะ " 1 " หรือ " 0 " นี้ จะทำภายในพัลส์ของสัญญาณนาฬิกาเพียงพัลส์เดียวบนสัญญาณนาฬิกาถัดไป ตัว SAR จะ " on " บิทนัยสูงสุดอันดับสองจะเปรียบเทียบไปเรื่อย ๆ ตัว SAR จะส่งสัญญาณสิ้นสุดการแปรผัน (End of conversion : EOC) ไปทำการค้างผลลัพท์ไว้ที่เป็นเลขฐานทางเอ้าท์พุทไว้

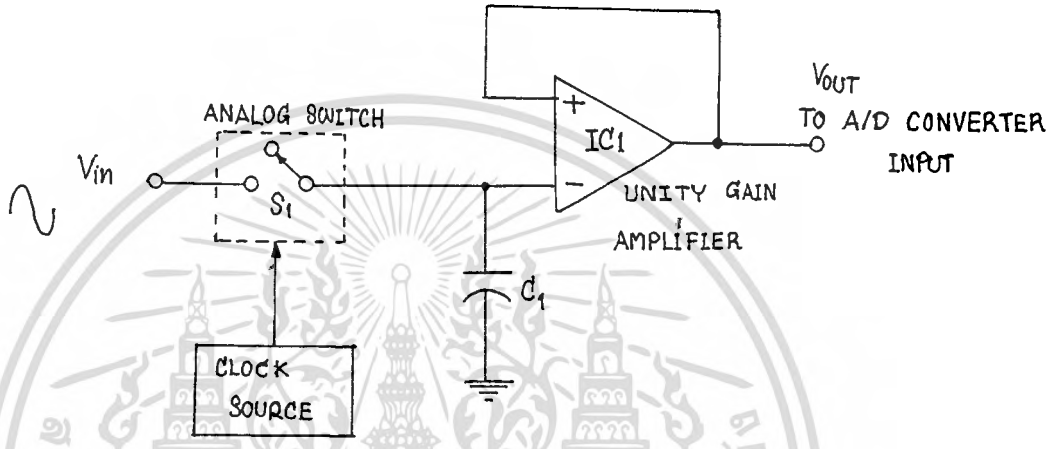
Quantizing Error

Quantizing error ซึ่งมีความสามารถเนื่องมาจากการเปลี่ยนแปลงอะนาล็อกทางด้านอินพุทอยู่ระหว่างที่ทำการแปลงสัญญาณอยู่การทำงานเริ่มต้นขึ้นที่ ADC ต้องการเวลาในช่วงหนึ่ง เพื่อที่จะสร้างดิจิตอลออกมาทางด้านเอ้าท์พุท ถ้าแรงดันอินพุทเกิดการเปลี่ยนแปลงขึ้นในระหว่างการแปลงสัญญาณ ไบนารี่เอ้าท์พุทสุดท้ายจะแทนระดับแรงดันที่ท้ายสุดของวัฏจักรแทนที่จะเป็นช่วงเริ่มต้นเมื่อไม่การเปลี่ยนแปลงแรงดันอินพุทขึ้น เช่น ในกรณีแรงดันไฟตรง ในกรณีนี้จะไม่เกิด Quantizing error ขึ้นดังแสดงในรูป 13 A สัญญาณที่มีการเปลี่ยนแปลงอย่างรวดเร็ว หรือ ที่เราเรียกว่า " slew rate " นั้นจะก่อให้เกิด Quantizing error มากยิ่งขึ้นดังแสดงในรูป 13 B และ 13 D ตามลำดับ



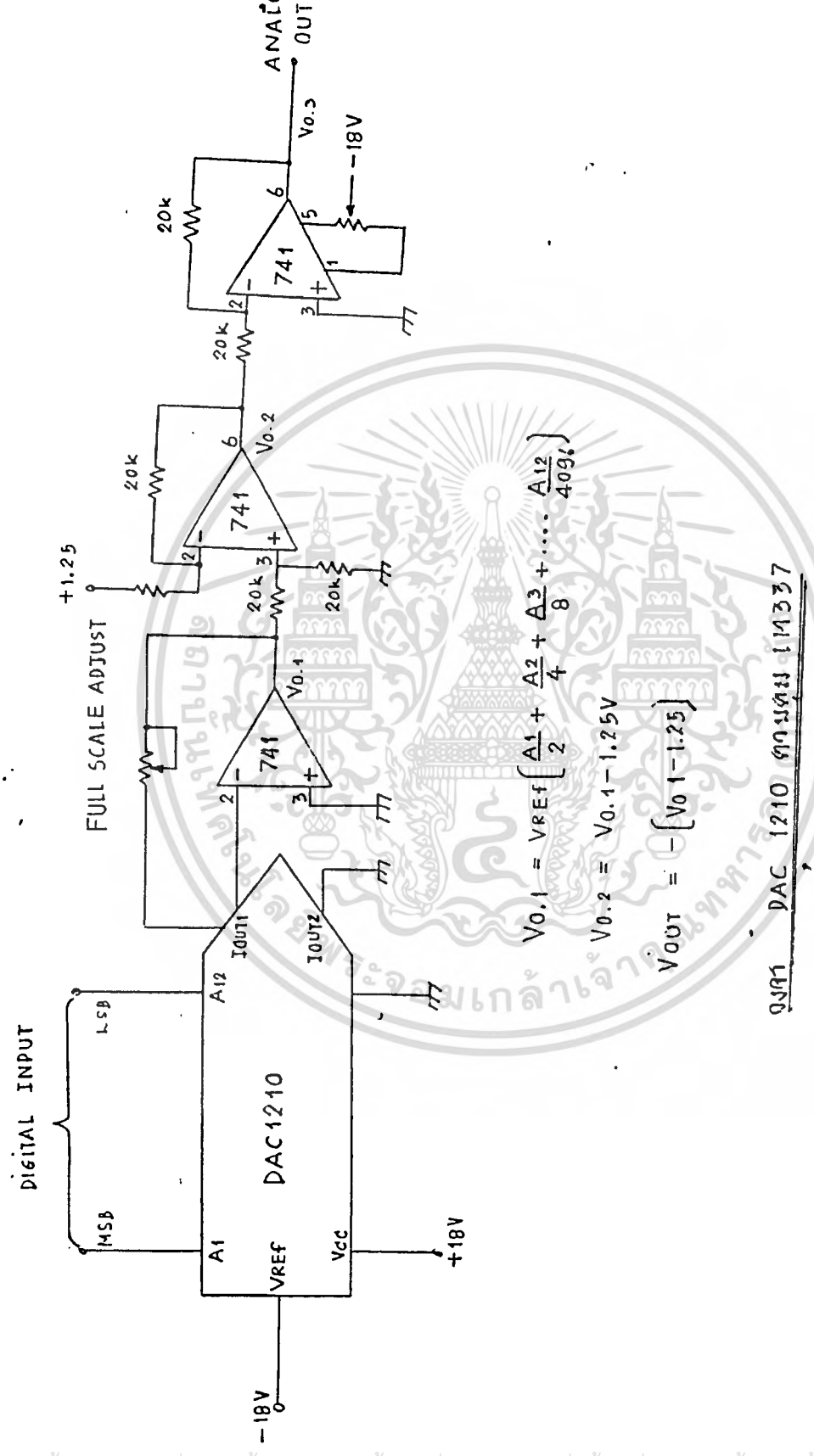
รูปที่ 13 Quantizing error เกิดขึ้นเนื่องจากสัญญาณอะนาล็อกทางด้านอินพุทเกิดการเปลี่ยนแปลงขึ้นในระหว่างวัฏจักรการแปลงสัญญาณในรูป A จะไม่เกิด quantizing error ขึ้นเนื่องจากแรงดันอินพุทไม่มีการเปลี่ยนแปลง B) เกิด quantizing error เล็กน้อย C) เมื่อสัญญาณอินพุทความถี่สูงขึ้นระดับ quantizing error จะมากขึ้นด้วย

วิธีหนึ่งสามารถกำจัด quantizing error ได้ก็คือใช้วงจร S/H (Sampling and Hold circui ก่อนวงจรเปรียบเทียบในรูป 14 แสดงวงจรของ ADC ที่ใช้ S/H

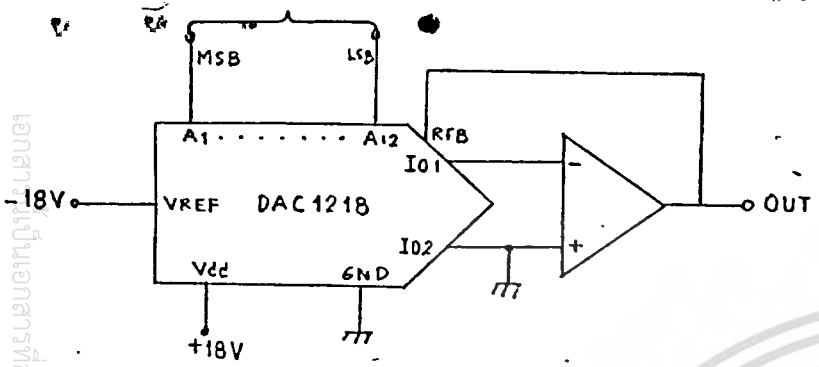


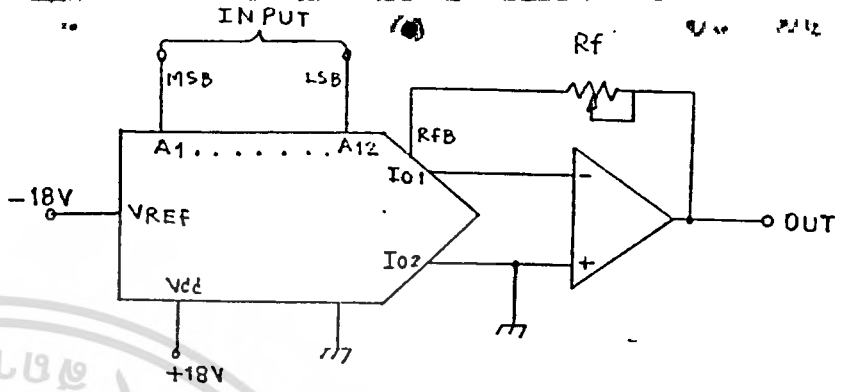
รูปที่ 14 วงจร S/H แบบง่าย ๆ

การทำงานในรูปที่ 14 สวิทช์อีเลคทรอนิคจะปิดวงจรเพื่อทำการสุ่มสัญญาณอะนาล็อกทางอินพุท ตัวเก็บประจุ C_1 มีหน้าที่เก็บประจุค่าของสัญญาณอินพุทที่เข้ามา ต่อมาสวิทช์อีเลคทรอนิคจะเปิดวงจรออกดังนั้นจึงเป็นกำจัดผลกระทบเกิดจาก quantizing error ทิ้งไป เพราะตัวเก็บประจุที่สุ่มไว้โดยไม่คำนึงถึงสัญญาณอินพุทจะเปลี่ยนแปลงอย่างไร เมื่อต้องการแปลงสัญญาณในช่วงเวลาถัดไป วงจรก็จะทำการสุ่มสัญญาณขึ้นใหม่อีกครั้ง



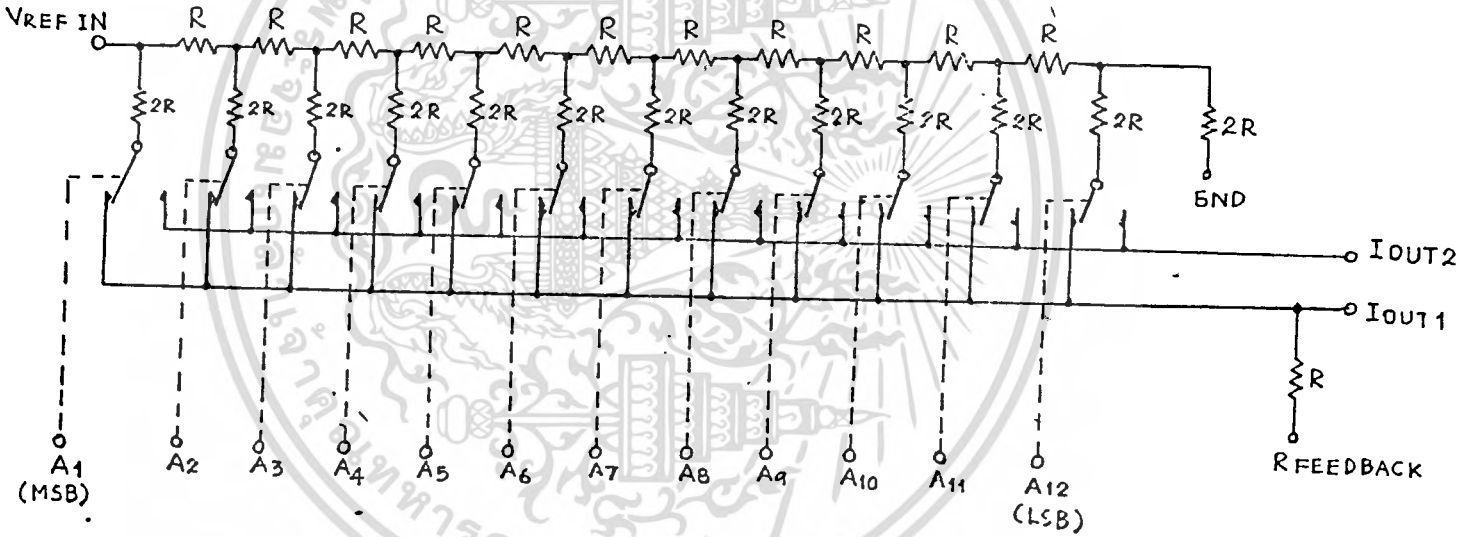
รูปที่ DAC 1210 ตามหน้า 11337





$$V_{OUT} = V_{REF}\left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \ldots \frac{A_{12}}{4096}\right)$$

$$V_{OUT} = I_{OUT1}(R+Rf) = \frac{V_{REF}}{R}(R+Rf)\left(\frac{A_1}{2} + \frac{A_2}{4} + \ldots \frac{A_{12}}{4096}\right)$$



$$I_{OUT1\,(MAX)} = \frac{V_{REF}}{R}\left(\frac{4095}{4096}\right) \quad ; R = 15K\Omega$$

The R-2R CURRENT SWITCHING LADDER NETWORK

จากสูตร $I_{OUT} = \frac{V_{REF}}{R}\left(\frac{A1}{2} + \frac{A2}{4} + \cdots \frac{A12}{4096}\right)$

$\therefore$ ถ้า 1 STEP จะได้ $I_{OUT} = \frac{18V}{15k}\left(\frac{1}{4096}\right) = 0.29297\mu A$

และถ้าต้องการ 1 STEP ให้ได้ $V_{OUT} = 10\,mV$

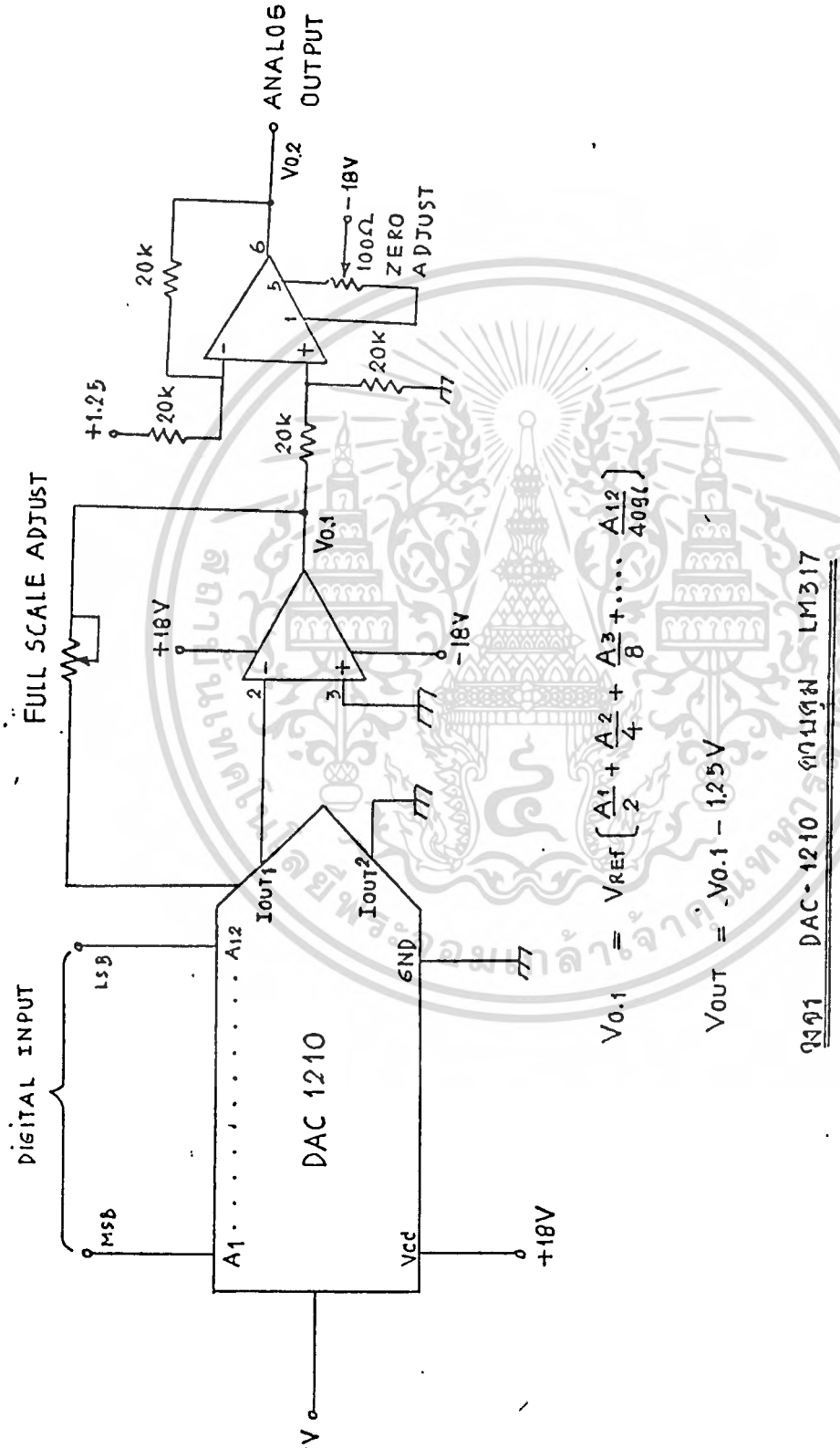
จากสูตร $V_{OUT} = I_{OUT}(R+Rf)$

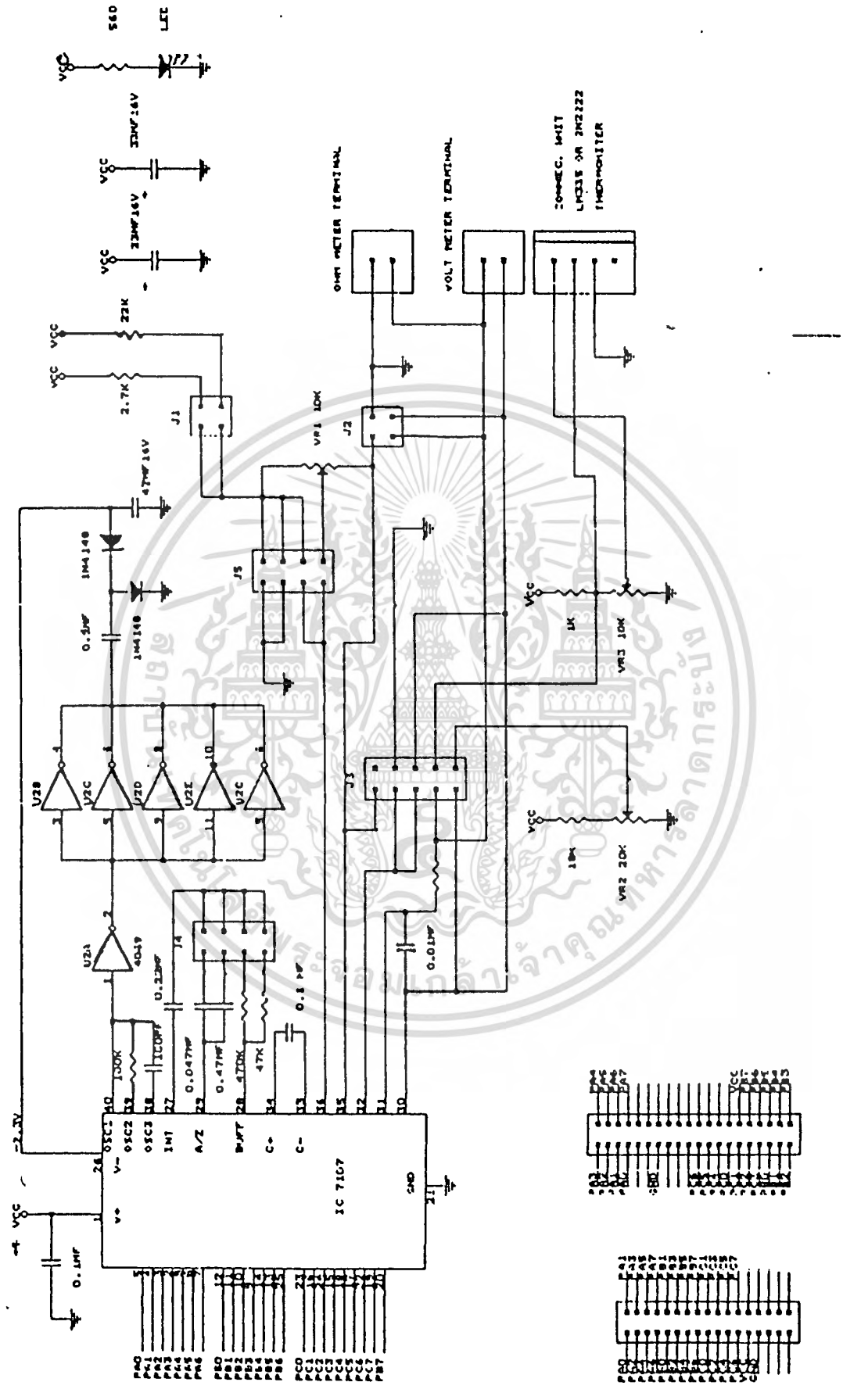
$\therefore Rf = \frac{V_{OUT}}{I_{OUT}} - R$

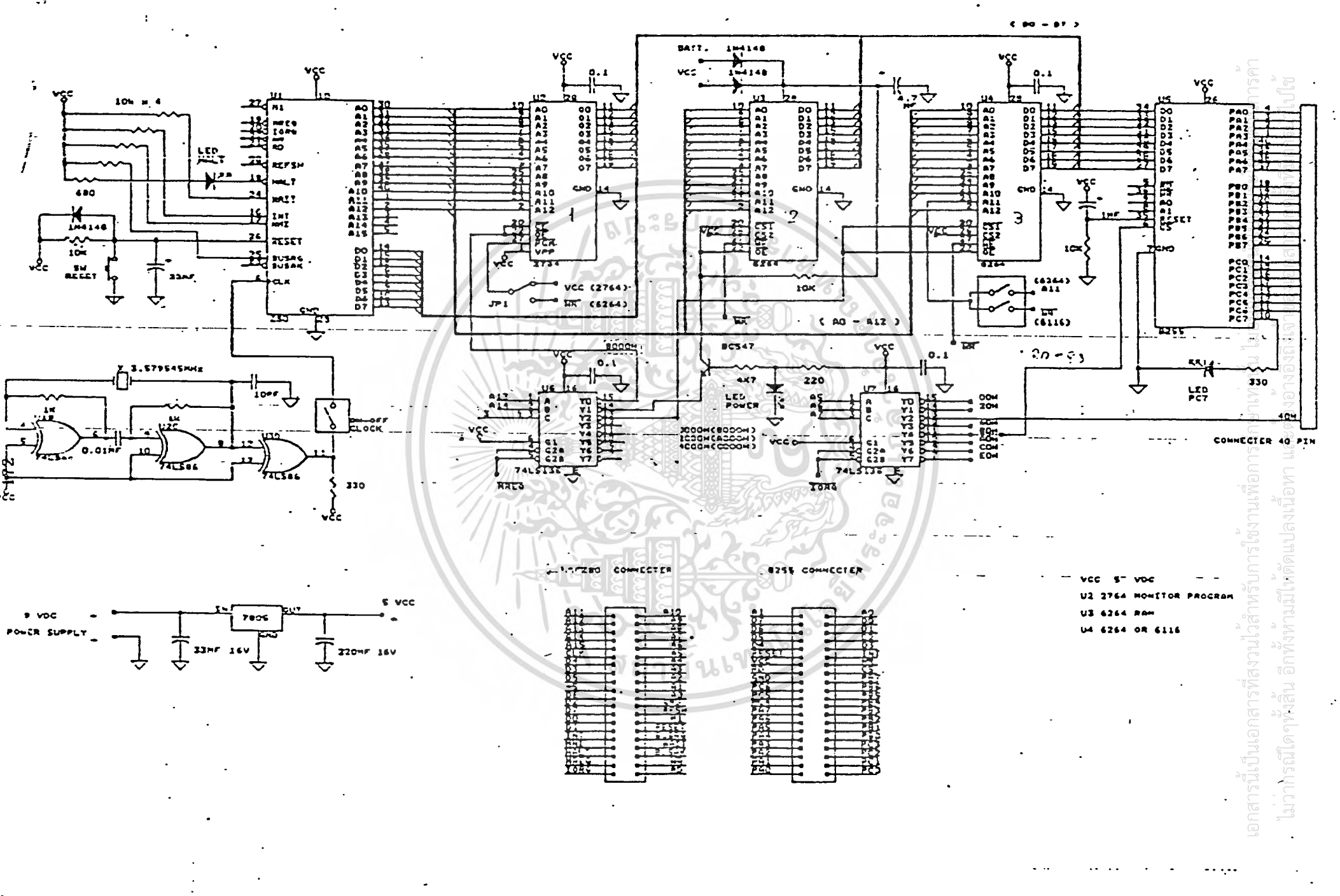
$= \frac{10\,mV}{0.29297\mu A} - 15k$

$= 19.133\,K\Omega$

$\therefore$ ใช้ VR ปรับค่า $= 20\,K\Omega$



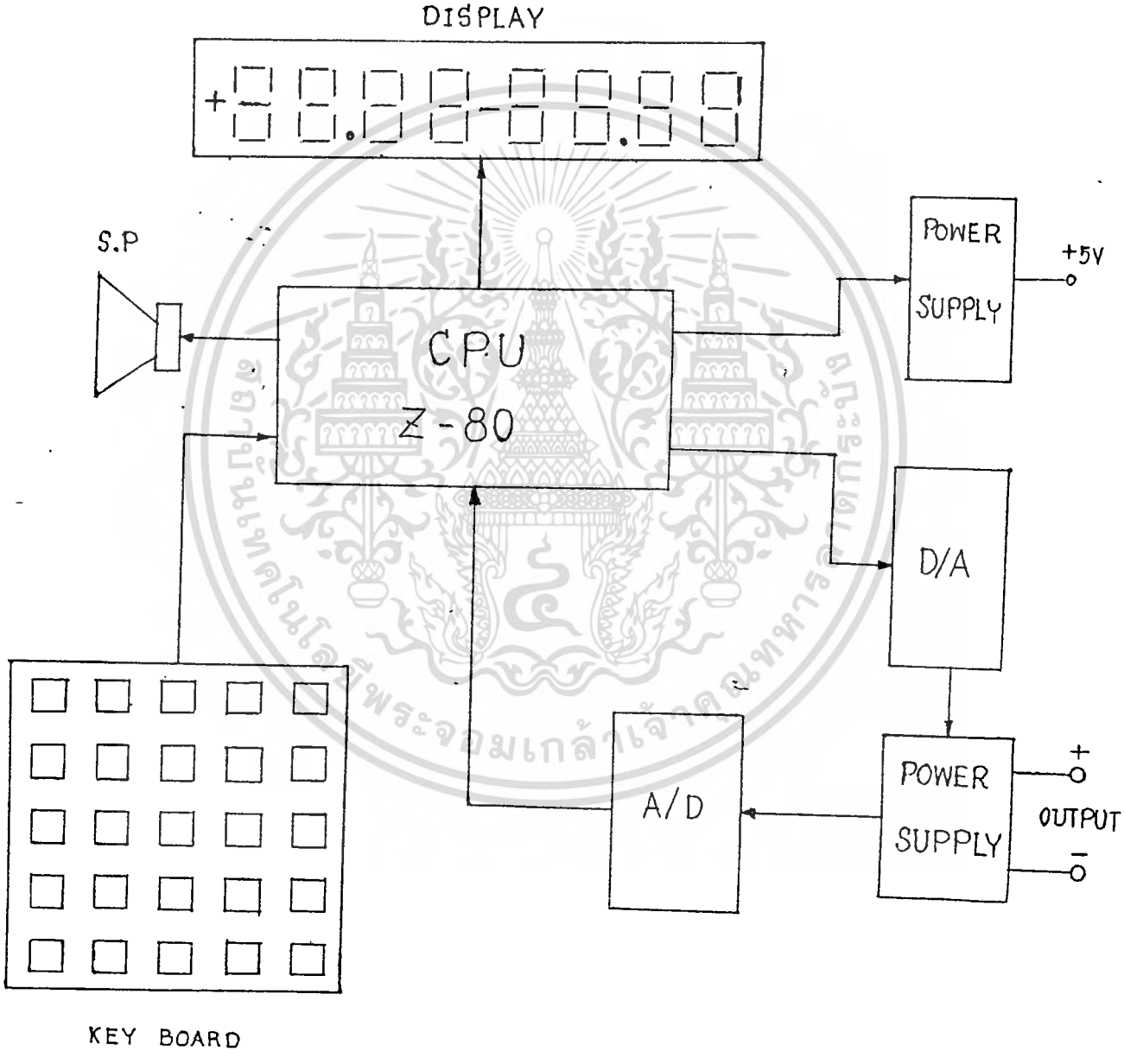



VCC
10K R.A
LED RESET
680
1N4148
10K
SW RESET
33MF
U1
RESET
CLK
GND
3.579545MHz
10PF
74LS86
0.01MF
74LS86
74LS86
CLOCK
330
U2
2764
JP1
VCC (2764)
PR (6264)
0.1
BATT.
1N4148
4.7
U3
6264
10K
(A0 - A12)
U4
6264
(6264) A11
(6116)
1MF
U5
8255
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
RESET
GND
(D0 - D7)
BC547
4K7
220
LED POWER
U6
U7
74LS138
8000H
A000H
C000H
E000H
A13
A14
330
LED PC7
CONNECTER 40 PIN
9 VDC
POWER SUPPLY
7805
5 VCC
33MF 16V
220MF 16V
Z80 CONNECTER
8255 CONNECTER
VCC 5 VDC
U2 2764 MONITOR PROGRAM
U3 6264 RAM
U4 6264 OR 6116

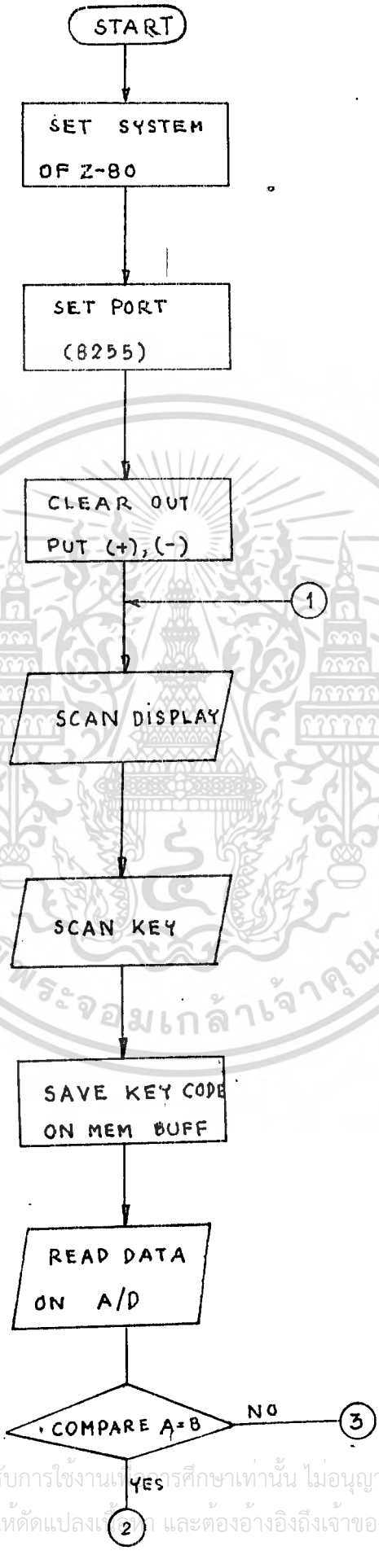
บทที่ 8

การทำงานของ Programmable Power Supply

BLOCK DIAGRAM OF POWER SUPPLY CONTROLLED BY CPU



รูป แสดงการทำงานโดยอาศัย SLOCK DIAGRAM



การใช้งานของเครื่อง Programmable Powre Supply

เครื่อง Programmable Power Supply เป็นเครื่องที่ทำหน้าที่จ่ายไฟตรง ที่มีความสามารถจ่ายไฟได้ ตั้งแต่ 0 - 18 โวลท์ จ่ายได้ทั้งไฟบวกและลบ ที่ปรับค่าได้ด้วย ซีพียู ซึ่ง ซีพียู ใช้ควบคุมระดับแรงดันไฟ พร้อมกัน ๆ กัน ทั้งไฟบวก และไฟลบ กระแสสูงสุดจ่ายได้ 3 แอมป์แปร์ ด้านเอาท์พุทที่วงจรป้องกันกระแสเกินแบบ โฟลด์แบค (Fold - Back Current Limiting) และมีการ short output จะมีการส่งสัญญาณ Interrupt ไปให้ cpu เพื่อหยุดการจ่ายไฟด้วย นอกจากนี้ยังมีเอาท์พุท 5 โวลท์ คงที่ และให้กระแส 3 แอมป์แปร์ สำหรับใช้กับงานที่ทดลองวงจร TTL ให้ด้วย.

ในการใช้งาน จะประกอบด้วยส่วนใหญ่ ๆ อยู่ 4 ส่วน คือ

1. วงจรควบคุม
2. ส่วนอินพุท
3. ส่วนเอาท์พุท
4. วงจรแหล่งจ่ายไฟตรงที่รักษาระดับแรงดันได้ ตั้งแต่ 0 ถึง 18 โวลท์ และ แหล่งไฟตรงคงที่ 5 โวลท์

1. วงจรควบคุม ทำหน้าที่รับเอาข้อมูลจาก Key Board จากเรากดปุ่มต่าง ๆ เพื่อทำแปลงระหัสจาก key board แล้วส่งไปยัง D/A CONVERTERS ซึ่ง D/A CONVERTERS จะทำการเปลี่ยนระหัสทางดิจิตอลเป็น อะนาล็อก เพื่อให้ แหล่งจ่ายไฟตรงที่รักษาระดับแรงดันได้จ่ายไฟ ตามที่เราสั่งงานตามต้องการ แล้วยังมีเช็คไฟที่ป้อนออกไปว่าตรงกันกับที่กำหนดใน KEY BOARD หรือไม่ โดยใช้วงจร A/D CONVERTERS และ A/D CONVERTERS จะส่งข้อมูลที่ได้ให้ CPU ตรวจสอบอีกทีหนึ่งถ้า อินพุท กับ เอาท์พุทตรงกันให้ แสดงออกบนหน้าจอ 7 - Segment (DISPLAY 7-Segment) ในกรณีระหว่าง input กับ output มีข้อมูลไม่ตรงกัน cpu ก็จะไม่มีการแสดงออกทางหน้าจอ

มีความสามารถเพิ่มโวลท์และลดโวลท์ ได้ที่ละ 10 mili - volts หรือ 0.01 โวลท์ โดยกดปุ่ม Inc , dec ตามลำดับ พร้อมทั้งเพิ่ม , ลด ตาม cpu สั่งงาน คือ ที่ละ 1 volts , 2 volts และ สามารถเพิ่มโวลท์ได้ทั้งบวก , ลบ พร้อมกัน หรือ เพิ่มเฉพาะ ลบ ด้านเดียวก็ได้ และบวกด้านเดียวก็ได้

ในอีกกรณีหนึ่งคือ ถ้ามีการ short ทางด้าน output และ กระแสเกินทางด้าน output cpu จะทำการ check แรงดันไฟเป็นศูนย์จริงหรือไม่ ถ้าเป็นศูนย์จริงจะทำการ off สัญญาณ หรือหยุดการทำงานของวงจร โดยจะมีสัญญาณ Interrupt ให้การ cpu และ display จะแสดงเป็นศูนย์

ทั้งหมดพร้อมมีเสียงเตือน

2. ส่วนอินพุท (INPUT SECTION)

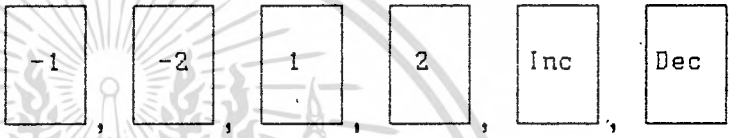
ส่วนนี้จะมีวงจรอยู่ 2 วงจร คือ

2.1 วงจร Key Board เป็น Key Board ขนาด 5x5 และทำการดัดแปลงจาก 5x5 เป็น 3x8 ซึ่งมีปุ่มต่าง ๆ ดังนี้

กลุ่มการทำงาน ตัวเลขได้ตั้งแต่ 0 - 9 และ จุด (.)

กลุ่มทำการสั่งงานจะมีปุ่ม Track , Set , Positive , Negative , Enter on/off

กลุ่มทำการเพิ่มและลด จะมีปุ่ม



2.2 D/A CONVERTERS ทำหน้าที่แปลงระหัสที่ cpu ส่งมาให้ที่มีสัญญาณเป็นดิจิตอลให้เป็นสัญญาณ Anatog เพื่อที่จะสั่งให้ แหล่งไฟจ่ายไฟ ออกทาง output ได้ตามต้องการ

3. ส่วนเอาท์พุท (OUTPUT SECTION)

ส่วนนี้จะประกอบด้วย 2 ส่วนคือ

3.1 วงจร A/D Converters เป็นวงจรที่ทำหน้าที่เปลี่ยนแปลงระบบจากอะนาล็อก เป็นดิจิตอล (analog to digital) ใช้ Ic # 7107 เป็น A/D ขนาด 3 1/2 gidit หรือ 0 - 1999 ระดับ คือ เทียบเป็น A TO D ได้ขนาดประมาณ 10 bit วงจรนี้จะส่งสัญญาณที่ได้ส่งไปยัง cpu เพื่อให้ cpu ทำการตัดสินใจในแสดงผล

3.2 ส่วน Display 7 - Segment

ส่วนนี้จะแสดงตัวเลข 7- Segment เป็นบวก , ลบ และแสดงตัวเลขได้ 8 หลัก และแสดงเป็น 5 volts อีก 2 หลัก

4. วงจรแหล่งจ่ายไฟตรงที่รักษาระดับแรงดันได้ตั้งแต่ 0 ถึง 18 โวลท์และ แหล่งจ่ายไฟตรงคงที่ 5 โวลท์

สามารถที่ปรับค่าได้ด้วยวงจร D/A. CONVERTERS ใช้ควบคุมระดับแรงดันไปพร้อม ๆ กันทั้งไฟบวก และไฟลบ กระแสสูงสุดจ่ายได้ 3 แอมป์แปร์ ด้านเอาท์พุทจะมีวงจรป้องกันกระแสเกิน และการป้อนกันการช๊อค ซึ่งจะมี ทรานซิสเตอร์ เป็นตัวป้องกันและส่งสัญญาณไปยังขา Interrupt ของ cpu เพื่อหยุดการทำงาน

แหล่งจ่ายไฟตรงคงที่ 5 โวลท์ ส่วนนี้จะเป็นแหล่งจ่ายไฟที่คงทีตายตัวเพื่อใช้ในการเลี้ยง Ic ตระกูล TTL และให้กระแสสูงสุด 3 แอมป์แปร์ (3 - Amp maximum) .

2050		CP 12	FE 1
2052		CALL Z, KEY ON / OFF	CC
2055		JUMP LUMP L1 ;	C 84020
2058	KEY SET :	LD HL, 300A	2 10 A 30
205B		LD A, (HL)	7 E
205C		AND 8 F	E 68 F
205E		LD (HL),A	77
205F		OUT (A2),A	D 3 A 2
2061		CALL SCAN	CD 0035
2064		CP 0C	FE0C
2066		CALL Z, POS	CC 7620
2069		CP 0D	FE0D
206B		CALL Z, NEG	CC
206E		CD 0E	FE0E
2070		CALL Z, TRACK	CC
2073		JUMP KEY SET	C 35820
2076		LD HL, 800 A	210 A 30
2079		LD A, (HL)	7 E
207A		AND 40	F6 40
207C		LD (HL), A	77
207D		OUT (A2), A	D 3 A 2
207F		CALL SCAN	CD 0035
2082		CP 13	FE 13
2084		CALL Z, INC POS	CC CF 26
2087		CP 11	FE 11
2089		CALL Z, DEC POS	CC F 27
208C		CP 0F	FE0F
208E		CALL Z, + 1V STEP	CC 7F 27
2091		CP 16	FE 16
2093		CALL Z, + 2V STEP	CC D0 27

2096	CP 14	FE 14
2098	CALL Z, - 1V STEP	CC AF 27
209B	CO 15	FE 15
209D	CALL Z, - 2V STEP	CC 00 28
20A0	NOP	00 00
20A2	JUMP CHECK O-A	C3 6021
20A5	CP 10	FE 10
20A7	CALL Z, ENT POS	CC
20AA	CP OB	FEOB
20AC	CALL Z, KEY SET	CC 5820
20AF	CP 12	FE 12
20B1	CALL Z, KEY ON / OFF	CC
20B4	JUMP POS :	C3 7620

20D0	NEG:	LD HL,300A	21 0A 30
20D3		LD A,(HL)	7E
20D4		AND 20 H	E6 20
20D6		LD (HL), A	77
20D7		OUT (A2), A	D3 A2
20D9		CALL SCAN	CD 00 35
20DC		CP 13	FE 13
20DE		CALL Z, INC NEG	CC
20E1		CP 11	FE 11
20E3		CALL Z, DEC NEG	CC
20E6		CP 0F	FE 0F
20E8		CALL Z, 1V STEP	CC
20EB		CP 16	FE 16
20ED		CALL Z, 2V STEP	CC
20F0		CP 14	FE 14
20F2		CALL Z, - 1V	CC
20F5		CP 15	FE 15
20F7		CALL Z, - 2V	CC
20FA		CP 0A	FE 0A
20FC		CALL C, KEY 0-9 NEG	DC
20FF		CP 10	FE 10
2101		CALL Z, ENT NEG	CC
2104		CP 0B	FE 0B
2106		CALL Z, KEY SET	CC 60 20
2109		CD 12	FE 12
210B		CALL Z, ON / OFF	CC
210E		JUMP NEG	C3 D0 20

TRACK

2120	TRACK :	LD HL, 300A	21 0A 30
2123		LD A, (HL)	7E
2124		AND 60 H	E6 60
2126		OUT (A2), A	D3 A2
2128		LD (HL), A	77
2129		CALL SCAN :	CD 00 35
212C		CP 13	FE 13
212E		CALL Z, INC TRACK	CC
2131		CP 11	FE 11
2133		CALL Z, DEC TRACK	CC
2136		CP 0F	FE 0F
2138		CALL Z, 1V	CC
213B		CP 16	FE 16
213D		CALL Z, 2V	CC
2140		CP 14	FE 14
2142		CALL Z, -1V	CC
2145		CP 15	FE 15
2147		CALL Z, -2V	CC
214A		CP 0A	FE 0A
214C		CALL C, K 0-9 TRACK	DC
214F		CP 10	FE 10
2151		CALL Z, ENT TRACK	CC
2154		CP 0B	FE 0B
2156		CALL Z, KEY SET	CC 60 20
2159		JUMP TRACK	C3 20 21

2160	CHECK 0-9	CP 00	FE 00
2162		CALL KEY 0-9 POS CD	CC
2165		CP 01	FE 01
2167		CALL KEY 0-9 POS	CC
216A		CP 02	FE 02
216C		CALL KEY 0-9 POS	CC
216F		CP 03	FE 03
2171		CALL KEY 0-9 POS	CC
2174		CP 04	FE 04
2176		CALL KEY 0-9 POS	CC
2179		CP 05	FE 05
217B		CALL KEY 0-9 POS	CC
217E		CP 06	FE 06
2180		CALL KEY 0-9 POS	CC
2183		CP 07	FE 07
2185		CALL KEY 0-9 POS	CC
2188		CP 08	FE 08
218A		CALL KEY 0-9 POS	CC
218D		CP 09	FE 09
218F		CALL KEY 0-9 POS	CC
2192		JUMP L2	C3 A5 20

SCAN

3500	SCAN :	LD HL, 3000	21 00 30
3503		LD DE, 3FE7	11 E7 3F
3506		LD B, 08	06 08
3508	MOVE :	LD A, (HL)	7E
3509		LD (DE), A	12
350A		INC HL	23
350B		INC DE	13
350C		DJNZ MOVE :	10 FA
350E		LD A, 04	3E 04
3510		LD B, 01	06 01
3512		RST 10	D7
3513		LD A, FF	3E FF
3515		LD (3008), A	32 08 30
3518		LD B, 02	06 02
351A		LD E, 06	1E 06
351C		LD HL, 3006	21 06 30
351F	SCAN 1 :	XOR A	AF
3520		OUT (01), A	3D 01
3522		LD A, E	7B
3523		OUT (02), A	D3 02
3525		LD A, (HL)	7E
3526		OUT (01), A	D3 01
3528		PUSH BC	C5
3529		LD A, 0F	3E 0F
352B		RSP 10	D7
352C		POP 3C	C1
352D		INC HL	23
352E		INC E	1C
352F		DJNZ SCAN 1 :	10 EE

```
3531                    CALL MAIN SCAN :        CD 00 26
3534                    LD A,(3FFB)             3A FB 3F
3537                    LD (300B), A            32 08 30
353A                    RET                     C9
```

MAIN SCAN

```
2600    MAIN SCAN : SCAN LD B, 8                06 08
2602                    LD C, B                 48
2603                    LD E, 0                 1E 00
2605                    LD HL, 3FE7             21 E7 3F
2608    SCAN 1 :        CALL SCANS              CD B2 26
260B                    LD (MEMX),HL            22 F7 3F
260E                    IN A,(DITIG)            DB 02
2610                    AND 70                  E6 70
2612                    CP 70                   FE 70
2614                    JR NZ, SCAN 3           20 15
2616                    DEC C                   0D
2617                    JR NZ, SCAN 2           20 0A
2619                    LD HL,SYSFAG            21 FD 3F
261C                    RES 0, (HL)             C3 86
261E                    LD HL, REPDLY           21 FC 3F
2621                    LD (HL), REP 1          36 60
2623    SCAN 2 :        LD HL, (MEMX)           2A F7 3F
2626                    INC E                   1C
2627                    DJNZ SCAN 1             10 DF
2629                    JR SCAN                 18 D5
```

KEY PRESS

262B	SCAN 3 :	OR E	B3
262C		PUSH BC	C5
262D		LD HL, KEY TAB + 22	21 4C 1F
2630		LD B, 16 H	06 16
2632	SCAN 31 :	CP (HL)	BE
2633		JR Z, SCAN 32	28 03
2635		DEC HL	2B
2636		DJNZ, SCAN 31	10 FA
2638	SCAN 32 :	LD D, B	50
2639		POP DC	C1
263A		LD HL, SYSFLAG	21 FD 3F
263D		BIT 0, (HL)	C3 46
263F		JR Z, SCAN 5	28 1C
2641		LD 07	3E 07
2643		CP E	BB
2644		JR Z, SCAN 4	28 06
2646		LD A, D	7A
2647		CALL SCAN, C	CD 9E 26
264A		JR C, SCAN 2	38 D7
264C	SCAN 4 :	LD A, (KEY IN)	3A FB 3F
264F		CALL SCAN C	CD 9E 26
2652		JR C, SCAN 2	38 CF

AUTO REPEAT

2654		LD HL, REPDLY	21 FC 3F
2657		DEC (HL)	35
2658		JR NZ, SCAN 2	20 C9
265A		LD (HL), REP2	36 16
265C		REP	C9

NEW PRESS

265D	SCAN 5 :	LD A, D	7A
265E		LD (KIN), A	32 FB 3F
2661		SET 0, (HL)	CB C6
2663		BIT 3, HL	CB 5E
2665		JRNZ, SCAN 7	20 14
2667		LD HL, 0400H	21 00 04
266A		LD C, A	4F
266B		LD A, (GAIN)	3A FF 3F
266E		LD B, A	47
266F		LD A, C	79
2670		LD C, HIGFRE	0E 10
2672		CP 10	FE 10
2674		JR NC, SCAN 6	30 02
2676		LD C, LOWFRE	0E 14
2678	SCAN 6 :	CALL SOUND	CD 6C 18

INTANT KEY CHECK

267B	SCAN 7:	LD A,(KEYIN)	3A FB 3F
267E		LD HL,SYSFAG	21 FD 3F
2681		BIT 4,(HL)	CB 66
2683		RET NZ	C0
2684		CP 10H	FE 10
2686		JP Z,MONF	CA 33 03
2689		CP 12H	FE 12
268B		JR Z,RUN1	CA D3 04
268E		CP 14	FE 14
2690		JR Z,MEM1	CA 92 03
2693		CP 15	FE 15
2695		JP Z,STEP1	CA 3D 05
2698		CP 16H	FE 16
269A		JP Z,REG	CA E5 03
269D		RET	C9

AUTO -REPAET-KEY CHECK SUB

269E	SCAN C:	CP 11H	FE 11
26A0		RET Z	C8
26A1		CP 13H	FE 13
26A3		RET Z	C8
26A4		CP 15H	FE 15
26A6		RET Z	C8
26A7		CP 0FH	FE 0F
26A9		RET Z	C8
26AA		CP 16H	FE 16
26AC		RET Z	C8
26AD		CP 14H	FE 14

```
26AF                    RET Z                   C8
26B0                    SCF                     37
26B1                    RET                     C9

                SCAN DISPLAY SUB

26B2    SCAN S:         LD A,E                  7B
26B3                    OUT (DIGIT),A           D3 02
26B5                    LD A, (HL)              7E
26B8                    XOR  A                  AF
26B9    SCAN S 1:       DEC A                   3D
26BA                    JR NZ,SCAN S1           20FD
26BC                    OUT (SEGM),A            D3 01
26BE                    INC HL                  23
26BF                    RET                     C9
                INC POS
26EF                    LD A",A                 7F
26D0                    LD A,(3101)             3A 01 31
26D3                    LD H,A                  67
26D4                    LD A,(3100)             3A 00 31
26D7                    LD L,A                  6F
26D8                    LD BC,0708              01 08 07
26DB                    XOR A                   AF
26DC                    SBC HL,BC               ED 42
26DE                    JUMP  NC L10:           D2 2D 27
26E1                    LD A,(3101)             3A 01 31
26E4                    LD H,A                  67
26E5                    LD A,(3100)             3A 00 31
26E8                    LD L,A                  6F
26E9                    INC HL                  23
26EA    L4 :            LD A,L                  7D
26EB                    LD (3100),A             32 00 31
```

26EE		LD A,H	7C
26EF		LD (3101),A	32 01 31
26F2		LD A,(3100)	3A 00 31
26F5		LD (3FE2),A	32 E2 3F
26F8		LD A,(3101)	3A 01 31
26FB		LD (3FF3),A	32 FB 3F
26FE		LD A,0A	3E 0A
2700		RST 10	D7
2701		LD A,(3FF5)	3A F5 3F
2704		LD (3FF0),A	32 F0 3F
2707		LD A,(3FF4)	3A F4 3F
270A		LD (3FEF),A	32 EF 3F
270D		LD A,07	3E 07
270F		RST 10H	D7
2710		LD A,(3FE7)	3A E7 3F
2713		LD (3000),A	32 00 30
2716		LD A,(3FE8)	3A E8 3F
2719		LD (3001),A	32 01 30
271C		LD A,(3FE9)	3A E9 3F
271F		LD (3002),A	32 02 30
2722		LD A,(3FEA)	3A EA 3F
2725		LD (3003),A	32 03 30
2728		LD HL, 3001	21 01 30
272B		SET 7,(HL)	CB FE
272D	L10:	LD A,A"	7F
272E		RET	C9

DEC POS

273F		EXX AF , AF"	08
2740		LD A,(3101)	3A 01 31
2743		LD H,A	67
2744		LD A, (3100)	3A 00 31
2747		LD L,A	6F
2748		LD A,H	7C
2749		OR L	B5
274A		JUMP Z , L11:	CA 59 27
274D		LD A,(3101)	3A 01 31
2750		LD H,A	67
2751		LD A,(3100)	3A 00 31
2754		LD L,A	6F
2755		DEC HL	2B
2756		JUMP L4:	C3 EA 26
2759	L11:	EXX AF, AF"	08
275A		RET	C9

```
2760    SAVE 1 :        LD A, (3101)            3A 01 31
2763                    LD H, A                 67
2764                    LD A, (3100)            3A 00 31
2767                    LD L, A                 6F
2768                    RET                     C9

2770    SAVE 2 :        LD A, H                 7C
2771                    LD (3101), A            32 01 31
2774                    LD A, L                 7D
2775                    LD (3100), A            32 00 31
2778                    RET                     C9

277F                    EXX AF, AF'             08
2780    + 1V POS :      CALL SAVE 1 :           CD 60 27
2783                    LD BC, 0709             01 09 07
2786                    XOR A                   AF
2787                    SBC HL, BC              ED 42
2789                    JUMP NC L12 :           CA A6 27
278C                    CALL SAVE 1 :           CD 60 27
278F                    LD B, 64                06 64
2791                    L0 : INC HL             23
2792                    DJNZ L0 :               10 FD
2794                    L3 : LD BC, 0709        01 09 07
2797                    XOR A                   AF
2798                    PUSH HL                 E5
2799                    SBC HL, BC              ED 42
279B                    JUMP NC L1              D2 A5 27
279E                    POP HL                  E1
279F                    CALL SAVE 2 :           CD 70 27
27A2                    JUMP L4 :               CB EA 26
```

```
27A5    L 1 :           POP HL                  E1
27A6    L 12 :          EXX AF, AF'             08
27A7                    RET                     C9


                    - 1V POS

27AF                    EXX AF, AF'             08
27B0    - 1V POS :      CALL SAVE 1 :           CD 60 27
27B3                    LB BC, 0709             01 09 07
27B6                    XOR A                   AF
27B7                    SBC HL, BC              ED 42
27B9                    JUMP NC L4 :            CA C6 27
27BC                    CALL SAVE 1 :           CD 60 27
27BF                    LDB, 64                 06 64
27C0    L5 :            DEC HL                  2B
27C1                    DJNZ L5 :               10 FD
27C3                    JUMP L3 :               C3 94 27
27C6    L4 :            EXX AF, AF'             08
27C7                    RET                     C9


                    + 2V POS

27D0    + 2V POS        EXX AF, AF'             08
27D1                    CALL SAVE 1 :           CD 60 27
27D4                    LD BC, 0709             01 09 07
27D7                    XOR A                   AF
27D8                    SBC, HL, BC             ED 42
27DA                    JUMP NC L13 :           CA F7 27
27DD                    CALL SAVE 1 :           CD 60 27
27E0                    LD B, C8                06 C8
27E2    L 14 :          INC HL                  23
27E3                    DJNZ L 14 :             10 FD
27E5    L 16 :          LD BC, 0709             01 09 07
```

```
27E8                    XOR A                   AF
27E9                    PUSH HL                 E5
27EA                    SBC HL, BC              ED 42
27EC                    JUMP NC L 15 :          D2 F6 27
27EF                    POP HL                  E1
27F0                    CALL SAVE 2 :           CD 70 27
27F3                    JUM L4 :                C3 EA 26
27F6    L 15 :          POP HL                  E1
27F7    L 13 :          EXX AF, AF'             08
27F8                    RET                     C9

                    - 2V POS

2800    - 2V POS :      EXX AF, AF'             08
2801                    CALL SACE 1 :           CD 60 27
2804                    LB BC, 0709             01 09 07
2807                    XOR A                   AF
2808                    SBC HL, BC              ED 42
280A                    JUM P NC L 19 :         CA 18 28
280D                    CALL SAVE 1 :           CD 60 27
2810                    LD B, C8                06 C8
2812    L 18 :          DEC HL                  2B
2813                    DJNZ L 18 :             10 FD
2815                    JUMP L 16 :             C3 E5 27
2818    L 19 :          EXX AF, AF'             08
2819                    RET                     C9

281C    L 28 :          JUMP L 30 :             C3 D0 28
2820    KEY 0-9 POS :   LD (3000), A            32 0D 30
2823                    CALL CHANGE :           CD 80 28
2826                    LD (3000),A             32 00 30
2829                    LD B, 03                06 03
```

282B		LD A, 00	3E 00
282D		LD HL, 3001	21 01 30
2830	L 2 :	LD (HL), A	77
2831		INC HL	23
2832		DJNZ L 20 :	10 FC
2834		CALL SCAN	CD 00 35
2837		CP 10	FE 10
2839		JUMP Z, KEY ENT1	CA 3E 28
283C		TR L 28 :	18 DE
283F	KEY ENT1	LD A, (3000)	3A 00 30
2841		LD (3001), A	32 01 30
2844		LD A, 00	3E 00
2846		LD (3000), A	32 00 30
2849		LD A, 3F	3E 3F
284B		LD (3002), A	32 02 30
284E		LD (3003), A	32 03 30
2851		CALL CHANG SEVEN	CD 90 28
2854		CALL DEC TO HEX	CD B0 28
2857		LD A, (3FF2)	3A F2 3F
285A		LD (3100), A	32 00 31
285D		LD A, (3FF3)	3A F3 3F
2860		LD (3101), A	32 01 31
2863		LD A, (3100)	3A 00 31
2866		OUT (A1), A	D3 A1
2868		LD A, (3101)	3A 01 31
286B		OR 60	FD 60
286D		OUT (A2), A	D3 A2
286F		OUT (A0), A	D3 A0
2871		LD HL, (3001)	21 01 30
2874		SET 7, (HL)	CB FE
2876		LD A, (300D)	3A 0D 30

```
2879                    JUND POS :          C3 76 20

2880    CHANGE :        AND OF              E6 OF
2882                    LD HL, 6000         21 00 60
2885                    LD L, A             6F
2886                    LD A, (HL)          7E
2887                    RET                 C9

2890    CHANGE SEVEN TO DEC :   LD DE 3FE7  11 E7 3F
2893                    LD HL, 3000         21 00 30
2896                    LD B, 04            06 04
2898                    LD A, (HL)          7E
2899    L 21 :          LD (DE), A          12
289A                    INC DE              13
289B                    INC HL              23
289C                    DJNZ L 21 :         10 FA
289E                    LD A, 06            3E 06
28A0                    RST 10              D7
28A1                    RET                 C9
28B0    DEC TO HEX :    LD A, (3FEF)        3A EF 3F
28B3                    LD (3FF4), A        32 F4 3F
28B6                    LD A, (3FF0)        3A F0 3F
28B9                    LD (3FF5), A        32 F5 3F
28BC                    LD A, 0B            3E 0B
28BE                    RST 10              D7
28BF                    RET                 C9

28D0    L 30 :          LD (3000), A        32 0D 30
28D3                    SUB 09              D6 09
28D5                    JUMP C L 31 :       DA 50 2A
28D8                    CALL SCAN           CD G0 35
```

```
2910    CH 2 :      LD A, (3000)          3A 00 30
2913                LD (3100), A          32 01 30
2916                LD A, 3F              3E 3F
2918                LD (3000), A          32 00 30
291B                LD (3002), A          32 02 30
291E                LD (3003), A          32 03 30
2921    L 18 :      CALL CHANG SEVEN      CD 90 28
2924                CALL DEC TO HEX       CD B0 28
2927                LD A, (3FF2)          3A F2 3F
292A                LD (3100), A          32 00 31
292D                LD A, (3FF3)          3A F3 3F
2930                LD (3100), A          32 01 31
2933                LD HL, 3101           21 01 31
2936                SET BIT 7             CB FE
2938    L 24 :      CALL SCAN             CD 00 35
                    RET                   C9

2A00    ENT 2 :     LD HL, 3001           21 01 30
2A03                RES 7, (HL)           CB BE
2A05                LD HL, 3000           21 00 30
2A08                LD DE 3FE7            11 E7 3F
2A0B                LD B, 04              06 04
2A0D    L 25 :      LD A, (HL)            7E
2A0E                LD (DE), A            12
2A0F                INC DE                13
2A10                INC HL                23
2A11                DJNZ L 25 :           10 FA
2A13                LD A, 06              3E 06
2A15                RST 10 H              D7
2A16                LD A, (3FF0)          3A F0 3F
2A19                LD (3FF5), A          32 F5 3F
```

```
2A1C                LD A, (3FEF)          3A EF 3F
2A1F                LD (3FF4), A          32 F4 3F
2A22                LD A, 0B              3E 0B
2A24                RST 10 H              D7
2A25                LD A, (3FF2)          3A F2 3F
2A28                LD (3400), A          32 00 31
2A2B                OUT (A1), A           D3 A1
2A2D                LD A, (BFF3)          3A F3 3F
2A30                LD (3100), A          32 01 31
2A33                OUT (A2), A           D3 A2
2A35                OUT A0, A             D3 A0
2A37                LD HL, 3001           21 01 30
2A3A                SET 7, (HL)           CB FE
2A3C                CALL SCAN             CD 00 35
2A3F                JUMP POS :            C3 76 20

                SET COLUMN 2
2A50    L 38 :      LD A, (300D)          3A 0D 30
2A53                CALL CHANGE           CD 80 28
2A56                LD (3001), A          32 01 30
2A58    L 34 :      CALL SCAN             CD 00 35
2A5B                SUB 09
2A5D                JUMP C L 35 :         DA 63 2A
2A60                JUMP L 34 :           C3 58 2A
2A63    L 35 :      LD A, (300D)          3A 0D 30
2A64                LD HL, 3001           21 01 30
2A67                SET 7, HL             CB FE
2A69    L 37 :      CALL SCAN             CD 00 35
2A6C                CP 0A                 FE 0A
2A6E                JUMP C L 36 :         DA 75 2A
2A71                JUMP  L 37            C3 6A 2A
```

```
2A74    L 36 :      CALL CHANGE          CD 80 28
2A77                LD (3002), A         32 02 30
2A7A    L 39 :      CALL SCAN            CD 00 35
2A7D                CP 0A                FE 0A
2A7F                JUMP C L 38 :        DA 85 2A
2A82                JUMP L 39 :          C3 7A 2A
2A85    L 38 :      CALL CHANGE          CD 80 28
2A88                LD (3003), A         32 03 30
2A8B    L 40 :      CALL SCAN            CD 00 35
2A8E                CP 10                FE 10
2A90                JUMP L 41 :          CA 96 2A
2A93                JUMP L 40 :          C3 8B 2A
2A96    L 41 :      LD HL, 3FE8          21 E8 3F
2A99                RES 7, (HL)          CB BE
2A9B                LD A, 06             3E 06
2A9D                RST 10               D7
2A9E                LD A, (3FF0)         3A F0 3F
2AA1                LD (3FF5), A         32 F5 3F
2AA4                LD A, (3FEF)         3A EF 3F
2AA7                LD (3FF4), A         32 F4 3F
2AAA                LD A, 0B             3E 0B
2AAC                RST 10 H             D7
2AAD                LD A, (3FF2)         3A F2 3F
2AB0                LD (3100), A         32 00 31
2AB3                LD A (3FF3)          3A F3 3F
2AB6                LD (3101), A         32 01 31
2AB9                LD H, A              67
2ABA                LD A, (3100)         3A 00 31
2ABD                LD L, A              6F
2ABE                LD BC, 0709          01 09 07
2AC1                XOR A                AF
```

2AC2		SBC BC, HL	ED 42
2AC4		JUMP Z, L 35 :	C2 CA 2A
2AC7		JUMP START	C3 00 20
2ACA	L 35 :	LD A, (3100)	3A 00 31
2ACD		OUT (A1), A	D3 A1
2ACF		LD A, (3100)	3A G1 31
2AD2		OUT (A2), A	D3 A2
2AD4		OUT (A0), A	D3 A0
2AD6		JUMP POS :	C3 76 20

สรุป

ชุดควบคุมแหล่งจ่ายไฟด้วยไมโครโปรเซสเซอร์ จัดได้ว่าเป็นเครื่องมือทางอิเลคทรอนิคส์ ที่ประยุกต์ขึ้นมาใหม่ชนิดหนึ่ง โดยทั่วไปแล้วในท้องตลาด เป็น แหล่งจ่ายไฟ (REGULATOR) แบบใช้มือปรับแรงดันเอาท์พุทตามต้องการ แต่ชุดควบคุมแหล่งจ่ายไฟด้วยไมโครโปรเซสเซอร์ นี้เราเพียงแต่กดคีย์บอร์ด ก็จะได้แรงดันตามที่ต้องการ ความต้องการของแรงดันเอาท์พุทสูงกว่าแหล่งจ่ายไฟ ทั่ว ๆ ไป

จากการที่กล่าวมาข้างต้นนั้น ชุดควบคุมแหล่งไฟ ด้วยไมโครโปรเซสเซอร์ยังมีน้อย และมีใช้กันยังไม่แพร่หลายมากนัก ดังนั้น การสร้างชุดควบคุมแหล่งจ่ายไฟ ด้วยไมโครโปรเซสเซอร์ ขึ้นมาจึงพบอุปสรรค์หลายประการ เช่น อุปกรณ์ที่นำมาสร้างการหาข้อมูลอ้างอิง หรือ บางทีมีอุปกรณ์แต่ไม่มีคู่มือ ในการใช้ , บางทีมีคู่มือในการใช้อุปกรณ์นั้นแต่หาซื้อตามท้องตลาดไม่มี

การทดลองมีความยุ่งยาก เนื่องจากอุปกรณ์ที่ใช้ในวงจรบางตัวไม่ได้มาตรฐาน ทั้งนี้ ก็มีสามารถเหตุมาจากเทคโนโลยีทางด้านอิเล็คทรอนิคส์ ของประเทศยังไม่เจริญพอ เมื่อเปรียบเทียบที่เป็นผู้นำทางเศรษฐกิจต่าง ๆ เป็นผลให้การทดลองไม่เป็นไปตามทฤษฎี จึงทำให้เกิดการล่าช้ามาก

ถึงจะมีอุปสรรค์ ต่าง ๆ ก็ตาม ชุดควบคุมแหล่งจ่ายไฟด้วยไมโครโปรเซสเซอร์ ก็สามารถสร้างขึ้นจนสำเร็จ และมีประสิทธิภาพ จัดได้ว่าอยู่ในขั้นที่ยอมรับได้

อุปสรรค์ในการทำงาน

การทดลองใช้เครื่องมือ อุปกรณ์ทางอิเล็คทรอนิคส์ ต่าง ๆ ไม่เป็นไปตามทฤษฎี เช่น วงจร ANALOG TO DIGITAL และ วงจร DIGITAL TO ANALOG ซึ่ง วงจรทั้งสองนี้เป็นวงจร ที่มีขนาด 12 bit โดยส่วนมากแล้วไม่ค่อยมีการใช้ที่แผ่หลายมากนัก ถ้ามีการใช้งานก็มีใช้เฉพาะงานเท่านั้น และ การใช้ ไมโครคอมพิวเตอร์เข้าไป ควบคุมสัญญาณต่าง ๆ จะต้องไม่มีความผิดพลาด โดยเฉพาะวงจรที่มีความแม่นยำสูง ถ้าเกิดการผิดพลาดขึ้นมาจะทำให้ ไมโครคอมพิวเตอร์ ทำงานไม่ถูกต้อง

กิตติกรรมประกาศ

โครงการนี้เป็นผลสำเร็จเรียบร้อยได้ โดยได้รับคำแนะนำปรึกษาและช่วยเหลือจากอาจารย์ที่ปรึกษา ตลอดจนสถานที่จัดทำโครงการของภาค วิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์

ขอขอบคุณ อาจารย์ ชวลิต เบญจางคประเสริฐ ไว้ ณ. ที่นี้ด้วย

คณะผู้จัดทำ

หนังสืออ้างอิง

ออปแอมป์

อิเล็คทรอนิคส์ คอมพิวเตอร์

เซมิคอนดักเตอร์ อิเล็คทรอนิคส์

ไมโครโปรเซสเซอร์

ภาคผนวก

National Semiconductor Corporation

DAC1218, DAC1219
12-Bit Binary Multiplying D/A Converter

General Description

The DAC1218 and the DAC1219 are 12-bit binary, 4-quadrant multiplying D to A converters. The linearity, differential non-linearity and monotonicity specifications for these converters are all guaranteed over temperature. In addition, these parameters are specified with standard zero and full-scale adjustment procedures as opposed to the impractical best fit straight line guarantee.

This level of precision is achieved though the use of an advanced silicon-chromium (SiCr) R-2R resistor ladder network. This type of thin-film resistor eliminates the parasitic diode problems associated with diffused resistors and allows the applied reference voltage to range from −25V to 25V, independent of the logic supply voltage.

CMOS current switches and drive circuitry are used to achieve low power consumption (20 mW typical) and minimize output leakage current errors (10 nA maximum). Unique digital input circuitry maintains TTL compatible input threshold voltages over the full operating supply voltage range.

The DAC1218 and DAC1219 are direct replacements for the AD7541 series, AD7521 series, and AD7531 series with a significant improvement in the linearity specification. In applications where direct interface of the D to A converter to a microprocessor bus is desirable, the DAC1208 and DAC1230 series eliminate the need for additional interface logic.

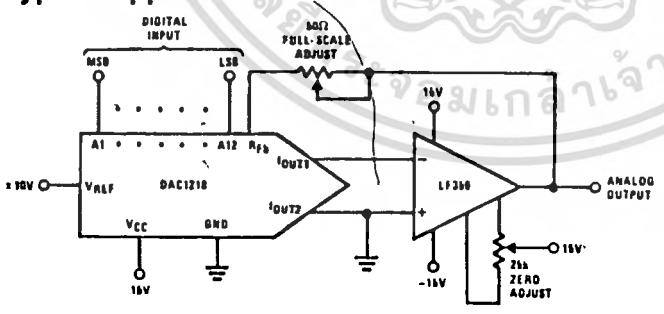
Features

- Linearity specified with zero and full-scale adjust only
- Logic inputs which meet TTL voltage level specs (1.4V logic threshold)
- Works with ±10V reference—full 4-quadrant multiplication
- All parts guaranteed 12-bit monotonic

Key Specifications

Current Settling Time	1 μs
Resolution	12 Bits
Linearity (Guaranteed over temperature)	12 Bits (DAC1218) 11 Bits (DAC1219)
Gain Tempco	1.5 ppm/°C
Low Power Dissipation	20 mW
Single Power Supply	5 V_{DC} to 15 V_{DC}

Typical Application

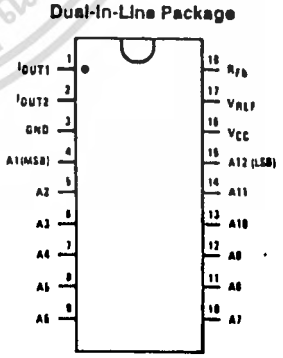


$$V_{OUT} = -V_{REF}\left(\frac{A1}{2} + \frac{A2}{4} + \frac{A3}{8} + \ldots \frac{A12}{4096}\right)$$

where: AN = 1 if digital input is high
AN = 0 if digital input is low

Connection Diagram

Dual-In-Line Package



Top View

Ordering Information

Temperature Range		0°C to +70°C	−40°C to +85°C	Package Outline
Non Linearity	0.012%	DAC1218LCJ-1	DAC1218LCJ	J18A Cerdip
	0.024%	DAC1219LCJ-1	DAC1219LCJ	J18A Cerdip

Absolute Maximum Ratings (Notes 1 and 2)

If Military/Aerospace specified devices are required, contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Supply Voltage (V_{CC})	17 V_{DC}
Voltage at Any Digital Input	V_{CC} to GND
Voltage at V_{REF} Input	±25V
Storage Temperature Range	−65°C to +150°C
Package Dissipation at T_A = 25°C (Note 3)	500 mW
DC Voltage Applied to I_{OUT1} or I_{OUT2} (Note 4)	−100 mV to V_{CC}
Lead Temp. (Soldering, 10 seconds)	300°C
ESD Susceptibility (Note 11)	800V

Operating Conditions

Temperature Range	$T_{MIN} \le T_A \le T_{MAX}$
DAC1218LCJ, DAC1219LCJ	$-40°C \le T_A \le +85°C$
DAC1218LCJ-1, DAC1219LCJ-1	$0°C \le T_A \le 70°C$
Range of V_{CC}	5 V_{DC} to 16 V_{DC}
Voltage at Any Digital Input	V_{CC} to GND

Electrical Characteristics

V_{REF} = 10.000 V_{DC}, V_{CC} = 11.4 V_{DC} to 15.75 V_{DC} unless otherwise noted. **Boldface limits apply from T_{MIN} to T_{MAX}; see Note 9);** all other limits $T_A = T_J$ = 25°C.

Parameter	Conditions	Notes	Typ (Note 10)	Tested Limit (Note 11)	Design Limit (Note 12)	Units
Resolution			12	**12**	**12**	Bits
Linearity Error (End Point Linearity)	Zero and Full-Scale Adjusted DAC1218 DAC1219	4, 5, 9		 **0.012** **0.024**	 **0.012** **0.024**	 % of FSR % of FSR
Differential Non-Linearity	Zero and Full-Scale Adjusted DAC1218 DAC1219	4, 5, 9		 **0.018** **0.024**	 **0.018** **0.024**	 % of FSR % of FSR
Monotonicity		4	12	12	**12**	Bits
Gain Error (Min)	Using Internal R_{Fb}, V_{REF} = ±10V, ±1V	5	−0.1	0.0		% of FSR
Gain Error (Max)		5	−0.1	−0.2		% of FSR
Gain Error Tempco		5	±1.3		**±6.0**	ppm of FS/°C
Power Supply Rejection	All Digital Inputs High	5	±3.0	±30		ppm of FSR/%
Reference Input Resistance	(Min)	9	15	**10**	**10**	kΩ
	(Max)	9	15	**20**	**20**	kΩ
Output Feedthrough Error	V_{REF} = 120 Vp-p, f = 100 kHz All Data Inputs Low	6	3.0			mVp-p
Output Capacitance	All Data Inputs High I_{OUT1} I_{OUT2} All Data Inputs Low I_{OUT1} I_{OUT2}				200 70 70 200	pF pF pF pF
Supply Current Drain		9		2.0	**2.5**	mA
Output Leakage Current I_{OUT1} I_{OUT2}	 All Data Inputs Low All Data Inputs High	7, 9		 **10** **10**	 **10** **10**	 nA nA
Digital Input Threshold	Low Threshold High Threshold	9		**0.8** **2.2**	**0.8** **2.2**	V_{DC} V_{DC}
Digital Input Currents	Digital Inputs <0.8V Digital Inputs >2.2V	9		**−200** **10**	**−200** **10**	μA_{DC} μA_{DC}
t_s Current Settling Time	R_L = 100Ω, Output Settled to 0.01%, All Digital Inputs Switched Simultaneously		1			μs

Electrical Characteristics Notes

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its specified operating conditions.

Note 2: All voltages are measured with respect to GND, unless otherwise specified.

Note 3: The 500 mW specification applies for all packages. The low intrinsic power dissipation of this part (and the fact that there is no way to significantly modify the power dissipation) removes concern for heat sinking.

Note 4: Both I_{OUT1} and I_{OUT2} must go to ground or the virtual ground of an operational amplifier. The linearity error is degraded by approximately $V_{OS} \div V_{REF}$. For example, if V_{REF} = 10V then a 1 mV offset, V_{OS}, on I_{OUT1} or I_{OUT2} will introduce an additional 0.01% linearity error.

Note 5: The unit FSR stands for full-scale range. Linearity Error and Power Supply Rejection specs are based on this unit to eliminate dependence on a particular V_{REF} value to indicate the true performance of the part. The Linearity Error specification of the DAC1218 is 0.012% of FSR. This guarantees that after performing a zero and full-scale adjustment, the plot of the 4096 analog voltage outputs will each be within 0.012% × V_{REF} of a straight line which passes through zero and full-scale. The unit ppm of FSR (parts per million of full-scale range) and ppm of FS (parts per million of full-scale) are used for convenience to define specs of very small percentage values, typical of higher accuracy converters. 1 ppm of FSR = $V_{HLF}/10^6$ is the conversion factor to provide an actual output voltage quantity. For example, the gain error tempco spec of ±8 ppm of FS/°C represents a worst-case full-scale gain error change with temperature from −40°C to +85°C of ±(8)($V_{REF}/10^6$)(125°C) or ±0.75 (10^{-3}) V_{REF} which is ±0.075% of V_{REF}.

Note 6: To achieve this low feedthrough in the D package, the user must ground the metal lid. If the lid is left floating the feedthrough is typically 6 mV.

Note 7: A 10 nA leakage current with R_{Fb} = 20k and V_{REF} = 10V corresponds to a zero error of ($10 \times 10^{-9} \times 20 \times 10^{3}$) × 100% 10V or 0.002% of FS.

Note 8: Human body model, 100 pF discharged through 1.5 kΩ resistor.

Note 9: Tested limit for ÷1 suffix parts applies only at 25°C.

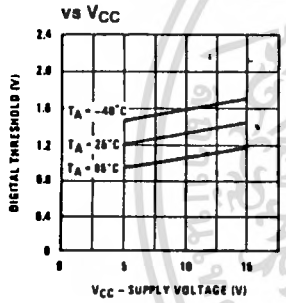
Note 10: Typicals are at 25°C and represent the most likely parametric norm.

Note 11: Tested limits are guaranteed to National's AOQL (Average Outgoing Quality Level).

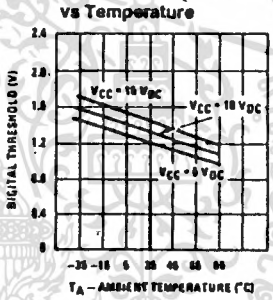
Note 12: Design limits are guaranteed but not 100% production tested. These limits are not used to calculate outgoing quality levels.

Typical Performance Characteristics

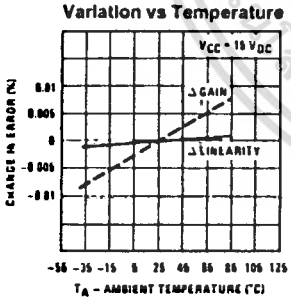
Digital Input Threshold vs V_{CC}



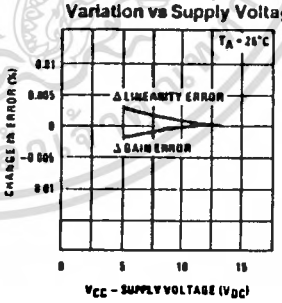
Digital Input Threshold vs Temperature



Gain and Linearity Error Variation vs Temperature



Gain and Linearity Error Variation vs Supply Voltage



TL/H/5891–2

Definition of Package Pinouts

(A1–A12): Digital Inputs. A12 is the least significant digital input (LSB) and A1 is the most significant digital input (MSB).

I_{OUT1}: DAC Current Output 1. I_{OUT1} is a maximum for a digital input of all 1s, and is zero for a digital input of all 0s.

I_{OUT2}: DAC Current Output 2. I_{OUT2} is a constant minus I_{OUT1}, or $I_{OUT1} + I_{OUT2}$ = constant (for a fixed reference voltage).

R_{Fb}: Feedback Resistor. The feedback resistor is provided on the IC chip for use as the shunt feedback resistor for the external op amp which is used to provide an output voltage for the DAC. This on-chip resistor should always be used (not an external resistor) since it matches the resistors in the on-chip R-2R ladder and tracks these resistors over temperature.

V_{REF}: Reference Voltage Input. This input connects to an external precision voltage source to the internal R-2R ladder. V_{REF} can be selected over the range of 10V to −10V. This is also the analog voltage input for a 4-quadrant multiplying DAC application.

V_{CC}: Digital Supply Voltage. This is the power supply pin for the part. V_{CC} can be from 5 V_{DC} to 15 V_{DC}. Operation is optimum for 15 V_{DC}.

GND: Ground. This is the ground for the circuit.

Definition of Terms

Resolution: Resolution is defined as the reciprocal of the number of discrete steps in the DAC output. It is directly related to the number of switches or bits within the DAC. For example, the DAC1218 has 2^{12} or 4096 steps and therefore has 12-bit resolution.

Linearity Error: Linearity error is the maximum deviation from a *straight line passing through the endpoints of the DAC transfer characteristic.* It is measured after adjusting for zero and full scale. Linearity error is a parameter intrinsic to the device and cannot be externally adjusted.

National's linearity test (a) and the best straight line test (b) used by other suppliers are illustrated below. The best straight line (b) requires a special zero and FS adjustment for each part, which is almost impossible for the user to determine. The end point test uses a standard zero FS adjustment procedure and is a much more stringent test for DAC linearity.

Power Supply Sensitivity: Power supply sensitivity is a measure of the effect of power supply changes on the DAC full-scale output.

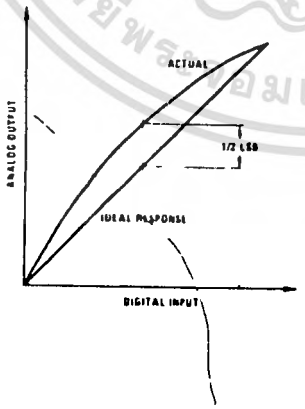
Settling Time: Full-scale current settling time requires zero to full-scale or full-scale to zero output change. Settling time is the time required from a code transition until the DAC output reaches within ± 1/2 LSB of the final output value.

Full-scale Error: Full-scale error is a measure of the output error between an ideal DAC and the actual device output. Ideally, for the DAC1218 full-scale is V_{REF} − 1 LSB. For V_{REF} = 10V and unipolar operation, $V_{FULL\text{-}SCALE}$ = 10.0000V − 2.44 mV = 9.9976V. Full-scale error is adjustable to zero.

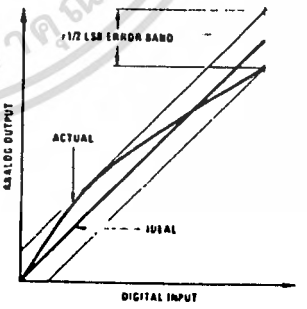
Differential Non-Linearity: The difference between any two consecutive codes in the transfer curve from the theoretical 1 LSB is differential non-linearity.

Monotonic: If the output of a DAC increases for increasing digital input code, then the DAC is monotonic. A 12-bit DAC which is monotonic to 12 bits simply means that input increasing digital input codes will produce an increasing analog output.

a) End point test after zero and FS adjust



b) Shifting FS adjust to pass best straight line test



TL/H/5691-3

Application Hints

The DAC1218 and DAC1219 are pin-for-pin compatible with the DAC1220 series but feature 12 and 11-bit linearity specifications. To preserve this degree of accuracy, care must be taken in the selection and adjustments of the output amplifier and reference voltage. Careful PC board layout is important, with emphasis made on compactness of components to prevent inadvertent noise pickup and utilization of single point grounding and supply distribution.

1.0 BASIC CIRCUIT DESCRIPTION

Figure 1 illustrates the R-2R current switching ladder network used in the DAC1218 and DAC1219. As a function of the logic state of each digital input, the binarily weighted current in each leg of the ladder is switched to either I_{OUT1} or I_{OUT2}. The voltage potential at I_{OUT1} and I_{OUT2} must be at zero volts to keep the current in each leg the same, independent of the switch state.

The switches operate with a small voltage drop across them and can therefore conduct currents of either polarity. This permits the reference to be positive or negative, thereby allowing 4-quadrant multiplication by the digital input word. The reference can be a stable DC source or a bipolar AC signal within the range of ±10V, for specified accuracy, with an absolute maximum range of ±25V. The reference can also exceed the applied V_{CC} of the DAC.

The maximum output current from either I_{OUT1} or I_{OUT2} is equal to

$$\frac{V_{REF(max)}}{R}\left(\frac{4095}{4096}\right);$$

where R is the reference input resistance (typically 15 kΩ). A high level on any digital input steers current to I_{OUT1} and a low level steers current to I_{OUT2}.

2.0 CREATING A UNIPOLAR OUTPUT VOLTAGE (A DIGITAL ATTENUATOR)

To generate an output voltage and keep the potential at the current output terminals at 0V, an op amp current to voltage converter is used. As shown in *Figure 2*, the current from I_{OUT1} flows through the feedback resistor, forcing a proportional voltage at the amplifier output. The voltage at I_{OUT1} is held at a virtual ground potential. The feedback resistor is provided on the chip and should always be used as it matches and tracks the R value of the R-2R ladder. The output voltage is the opposite polarity of the applied reference voltage.

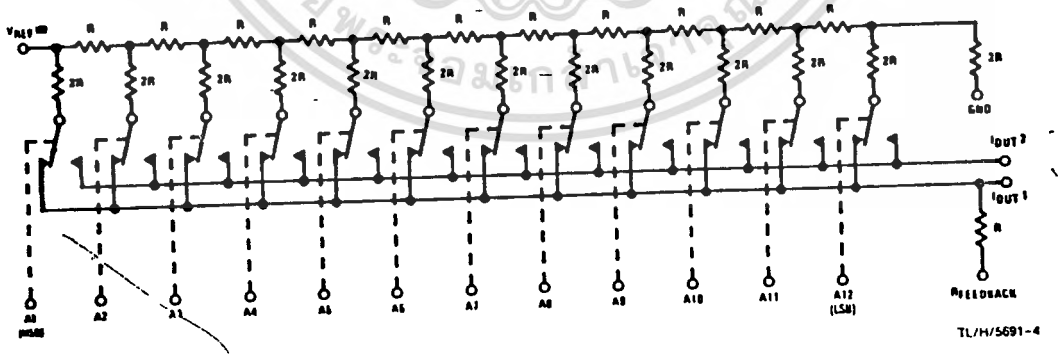
2.1 Amplifier Considerations

To maintain linearity of the output voltage with changing digital input codes the input offset voltage of the amplifier must be nulled. The resistance from I_{OUT1} to ground (R_{IOUT1}) varies non-linearly with the applied digital code from a minimum of R with all ones applied to the input to near ∞ with an all zeros code. Any offset voltage between the amplifier inputs appears at the output with a gain of

$$1 + \frac{R_F}{R_{IOUT1}}.$$

Since R_{IOUT1} varies with the input code, any offset will degrade output linearity. (See Note 4 of Electrical Characteristics.)

If the desired amplifier does not have offset balancing pins available (it could be part of a dual or quad package) the nulling circuit of *Figure 3* can be used. The voltage at the non-inverting input will be set to $-V_{OS}$ initially to force the inverting input to 0V. The common technique of summing current into the amplifier summing junction cannot be used as it directly introduces a zero code output current error.

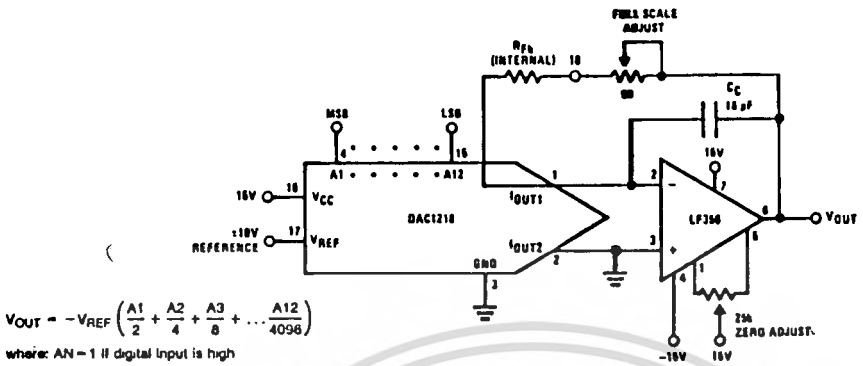


TL/H/5691-4

Note: Switches shown in digital high state.

FIGURE 1. The R-2R Current Switching Ladder Network

Application Hints (Continued)

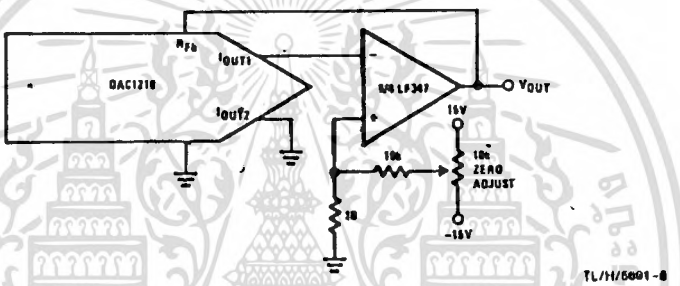


$$V_{OUT} = -V_{REF}\left(\frac{A1}{2} + \frac{A2}{4} + \frac{A3}{8} + \ldots \frac{A12}{4096}\right)$$

where: AN = 1 if digital input is high
AN = 0 if digital input is low

TL/H/5691-5

FIGURE 2. Unipolar Output Voltage



TL/H/5691-6

FIGURE 3. Zeroing an Amplifier Which Does Not Have Balancing Provisions

The selected amplifier should have as low an input bias current as possible since input bias current contributes to the current flowing through the feedback resistor. BI-FET™ op amps such as the LF356 or LF351 or bipolar op amps with super β input transistors like the LM11 or LM308A produce negligible errors.

2.2 Zero and Full-Scale Adjustments

The fundamental purpose is to make the output voltages as near 0 V_{DC} as possible. This is accomplished in the circuit of *Figure 2* by shorting out the amplifier feedback resistance, and adjusting the V_{OS} nulling potentiometer of the op amp until the output reads zero volts. This is done, of course, with an applied digital input of all zeros if I_{OUT1} is driving the op amp (all ones for I_{OUT2}). The feedback short is then removed and the converter is zero adjusted.

A unique characteristic of these DACs is that any full-scale or gain error is always negative. This means that for a full-scale input code the output voltage, if not inherently correct, will always be less than what it should be. This ensures that adding an appropriate resistance in series with the internal feedback resistor, R_{Fb}, will always correct for any gain error. The 50Ω potentiometer in *Figure 2* is all that is needed to adjust the worst case DAC gain error.

Conversion accuracy is only as good as the applied reference voltage, so providing a source that is stable over time and temperature is important.

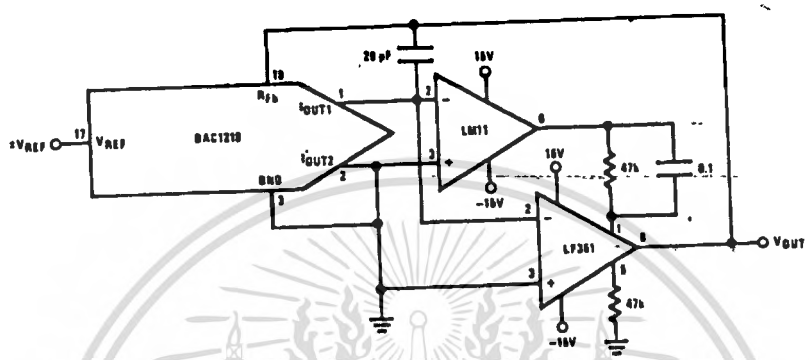
2.3 Output Settling Time

The output voltage settling time for this circuit in response to a change of the digital input code (a full-scale change is the worst case) is a combination of the DAC's output current settling characteristics and the settling characteristics of the output amplifier. The amplifier settling is further degraded by a feedback pole formed by the feedback resistance and the DAC output capacitance (which varies with the digital code). First order compensation for this pole is achieved by adding a feedback zero with capacitor C_C shown in *Figure 2*.

In many applications output response time and settling are just as important as accuracy. It can be difficult to find a single op amp that combines excellent DC characteristics (low V_{OS}, V_{OS} drift and bias current) with fast response and settling time. BI-FET op amps offer a reasonable compromise of high speed and good DC characteristics. The circuit of *Figure 4* illustrates a composite amplifier connection that combines the speed of a BI-FET LF351 with the excellent DC input characteristics of the LM11. If output settling time is not so critical, the LM11 can be used alone.

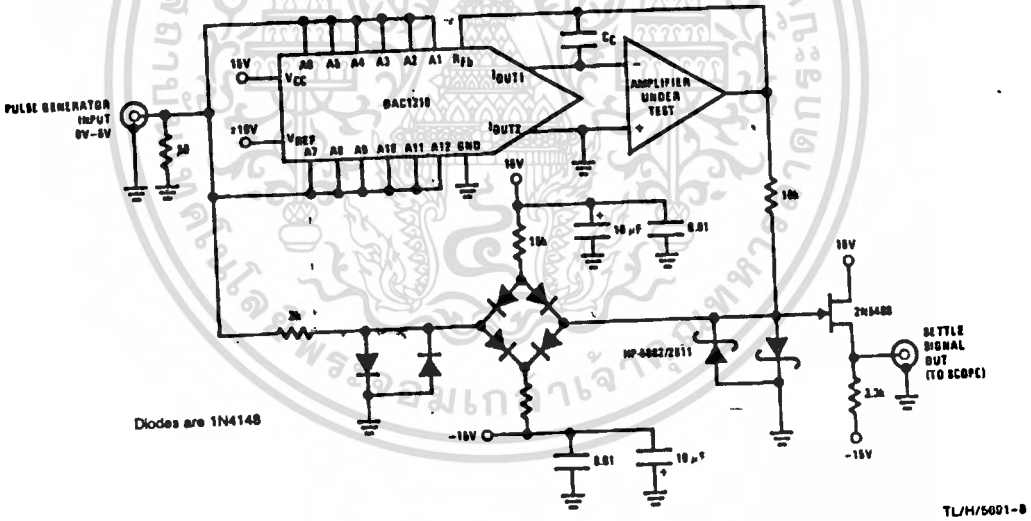
Figure 5 is a settling time test circuit for the complete voltage output DAC circuit. The circuit allows the settling time of the DAC amplifier to be measured to a resolution of 1 mV out of a zero to ±10V full-scale output change on an oscilloscope. *Figure 6* summarizes the measured settling time for several output amplifiers and feedback compensation capacitors.

Application Hint (Continued)



TL/H/5691-7

FIGURE 4. Composite Output Amplifier Connection



TL/H/5691-8

FIGURE 5. DAC Settling Time Test Circuit

Amplifier	C_C	Settling Time to 0.01%
LM11	20 pF	30 µs
LF351	15 pF	8 µs
LF351	30 pF	5 µs
Composite LM11-LF351	20 pF	8 µs
LF356	15 pF	6 µs

FIGURE 6. Some Measured Settling Times

Application Hints (Continued)

3.0 OBTAINING A BIPOLAR OUTPUT VOLTAGE FROM A FIXED REFERENCE

The addition of a second op amp to the circuit of *Figure 2* can generate a bipolar output voltage from a fixed reference voltage (*Figure 7*). This, in effect gives sign significance to the MSB of the digital input word to allow two quadrant multiplication of the reference voltage. The polarity of the reference voltage can also be reversed to realize full 4-quadrant multiplication.

The output responds in accordance to the following expression:

$$V_O = V_{REF}\left(\frac{D - 2048}{2048}\right), 0 \le D \le 4095$$

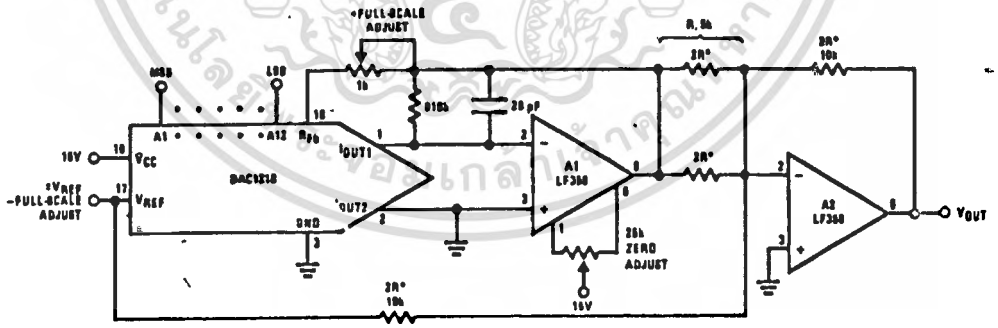
where D is the decimal equivalent of the true binary input word. This configuration inherently accepts a code (half-scale or D = 2048) to provide 0V out without requiring an external ½ LSB offset as needed by other bipolar multiplying DAC circuits.

Only the offset voltage of amplifier A1 need be nulled to preserve linearity. The gain setting resistors around A2 must match and track each other. A thin film, 4-resistor network available from Beckman Instruments, Inc. (part no. 694-3-R10K-D) is ideally suited for this application. Two of the four resistors can be paralleled to form R and the other two can be used separately as the resistors labeled 2R.

Operation is summarized in the table below:

Applied Digital Input MSB											LSB	Decimal Equivalent	V_{OUT} $+V_{REF}$	V_{OUT} $-V_{REF}$
1	1	1	1	1	1	1	1	1	1	1	1	4095	$V_{REF} - 1$ LSB	$-V_{REF} + 1$ LSB
1	1	0	0	0	0	0	0	0	0	0	0	3072	$V_{REF}/2$	$-\lvert V_{REF}\rvert/2$
1	0	0	0	0	0	0	0	0	0	0	0	2048	0	0
0	1	1	1	1	1	1	1	1	1	1	1	2047	−1 LSB	+1 LSB
0	1	0	0	0	0	0	0	0	0	0	0	1024	$-V_{REF}/2$	$+\lvert V_{REF}\rvert/2$
0	0	0	0	0	0	0	0	0	0	0	0	0	$-V_{REF}$	$+\lvert V_{REF}\rvert$

Where 1 LSB = $\frac{|V_{REF}|}{2048}$



*0.1% matching

TL/H/5691-9

FIGURE 7. Obtaining a Bipolar Output from a Fixed Reference