ปีการศึกษา พ.ศ.2531

เรื่อง CARD READER

จัดทำโดย

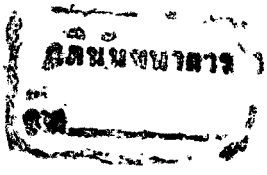
นาย จิตรพร หล่อประดิษฐ์

นาย วุฒิชัย ชีวัฒนารมย์

นาย วันชัย แซ่เล้า

อาจารย์ที่ปรึกษา

อาจารย์ อุทัย ศรีธีระวิโรจน์



รายงานนี้เป็นส่วนหนึ่งของปีการศึกษา พ.ศ.2531 ตามหลักสูตรปริญญาตรี

ภาควิชา เทคนิคอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง CARD READER

ผู้จัดทำ

1. นาย จิตรพร หล่อประดิษฐ์ เลขที่ 293403
2. นาย วุฒิชัย ชีวัฒนารมย์ เลขที่ 293409
3. นาย วันชัย แซ่เล้า เลขที่ 293421

_____ อาจารย์ที่ปรึกษา

( )

_____ อาจารย์ที่ปรึกษา

( )

_____ อาจารย์ที่ปรึกษา

( )

เครื่องอ่านบัตร

นายจิตรพร หล่อประดิษฐ์
นายวุฒิชัย ชีวัฒนารมย์
นายวันชัย แซ่เล้า

_____ อาจารย์ที่ปรึกษา

ปีการศึกษา 2531

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นเรื่องเกี่ยวกับหลักของการอ่านข้อมูลจากการ์ดเคลือบสารแม่เหล็ก ซึ่งมีประโยชน์ใช้สอยอย่างกว้างขวางมากในปัจจุบัน

ประโยชน์ใช้งานซึ่งเราเห็นกันในชีวิตประจำวัน ก็คือ การฝาก – ถอนเงินสดของธนาคาร อีกตัวอย่างหนึ่ง ได้แก่ การใช้บัตรเครดิตของห้างสรรพสินค้าต่าง ๆ บัตรประจำตัวและบัตรผ่านระบบรักษาความปลอดภัย เป็นต้น

Megnetic Card Reader

Mr.Chittaporn Lawpradist

Mr.Woottichai Cheewattanarom

Mr.Wanchai Saelao

Mr.U-Thai Sritheeravirojana

1988

Abstract

This thesis are concerning about fundamental of reading data from magnetic card. There will be useful and widely use inrecent future. An application which we have seen in daily life are cash control for banking. Another example is credit card for department stores. I.D. card and security passed card etc.

สารบัญ

บทนำ

ปัจจุบันนี้ในระบบคอมพิวเตอร์ได้พัฒนาไปไกลมาก จนเราไล่ตามไม่ทันเพราะการพัฒนาทางด้านคอมพิวเตอร์ไม่มีการหยุดอยู่กับที่คอยพัฒนาสิ่งต่าง ๆ ขึ้นมาอีกมากมาย เพราะฉะนั้นรายงานเล่มนี้ที่ได้ทำขึ้นมาเพื่อที่จะให้คุณผู้อ่านได้รู้เรื่องเกี่ยวกับคอมพิวเตอร์ ที่นำมันมาดัดแปลงเพื่อประโยชน์ต่อผู้อ่านบางคนและอาจจะเป็นแนวทางนำไปปรับปรุงให้ดีกว่าเดิมเพื่อประโยชน์ในอนาคตข้างหน้า และรายงานเล่มนี้จัดทำเรื่องที่เกี่ยวกับการอ่านบัตร (CARD READER) เพื่อที่จะนำไปใช้ประโยชน์ได้ต่าง ๆ อีกมากมาย เช่นการอ่านบัตรประจำตัว หรือใช้เป็นที่เปิดประตูอัตโนมัติ และยังสามารถนำไปใช้อย่างอื่นได้อีกมากมาย แล้วแต่จะนำไปดัดแปลงทำสิ่งที่ตนเองต้องการ

โครงสร้างภายในไมโครคอมพิวเตอร์

ส่วนประกอบภายในที่สำคัญของไมโครโปรเซสเซอร์มี ALU (Arithmetic Logic Unit) รีจีสเตอร์ข้อมูลและวงจรถอดรหัสคำสั่งหรือ IR และ ID (Instruction Register and Decoder) วงจรควบคุม (Control Circuit) และรีจีสเตอร์สำหรับใช้งาน ซึ่งแบ่งเป็นรีจีสเตอร์ข้อมูล (Data Register) และแอดเดรสรีจีสเตอร์ (Address Register) ส่วนประกอบที่สำคัญภายในเหล่านี้จะติดต่อกันด้วยบัสภายใน (Internal Bus) ซึ่งเป็นกลุ่มของเส้นสัญญาณมีจำนวนเท่ากับจำนวนบิทของไมโครโปรเซสเซอร์ บัสภายในนี้จะต่อกับบัสภายนอกโดยผ่านบัฟเฟอร์ เพื่อเป็นการติดต่อกับวงจรภายนอกอีกทีหนึ่ง

ALU

ALU เป็นหน่วยคำนวณสำคัญในไมโครโปรเซสเซอร์ หน่วยนี้จะทำการคำนวณทางเลขคณิต เช่น การบวก การลบ การเพิ่มหนึ่ง (Increment) การลดหนึ่ง (Decrement) การคำนวณทางลอจิก (Logic) เช่น AND OR NOT EXOR การเลื่อน (Shift) และการหมุน (Rotate) เป็นต้น

ข้อมูลที่จะคำนวณจะเข้าสู่ ALU ได้สองทาง คือ ทางหนึ่งจะรับข้อมูลจากรีจีสเตอร์พิเศษตัวหนึ่ง เรียกว่า แอคคิวมูเลเตอร์ (Accumulator) และอีกทางจะรับข้อมูลเข้าจากบัสข้อมูลภายใน ซึ่งเป็นบัสที่ใช้ร่วมกันภายใน เพื่อทำการติดต่อระหว่างรีจีสเตอร์ต่าง ๆ ข้อมูลจากรีจีสเตอร์จะส่งเข้า ALU โดยใช้บัสภายในนี้

แอคคิวมูเลเตอร์เป็นรีจีสเตอร์สำคัญทำงานร่วมกับ ALU เสมอ แอคคิวมูเลเตอร์จะเป็นทั้งอินพุทและเอาท์พุทของ ALU เมื่อไมโครโปรเซสเซอร์ทำงานในคำสั่งหนึ่ง เช่น ทำการบวกข้อมูลแอคคิวมูเลเตอร์กับข้อมูลในรีจีสเตอร์ตัวใดตัวหนึ่ง ผลของการบวกจะให้ผลลัพธ์จะมาเก็บไว้ในแอคคิวมูเลเตอร์นี้เสมอ การคำนวณทางตรรกเช่น การเลื่อนข้อมูลก็จะทำการคำนวณกับข้อมูลในแอคคิวมูเลเตอร์เช่นกัน ให้เปรียบเทียบแอคคิวมูเลเตอร์เหมือนกระดานลูกคิดที่ทั้งตัวบวกและผลลัพธ์จะปรากฏบนกระดานเองเสมอ และมือของผู้ดีดลูกคิดก็คือ ALU นี้เอง แอคคิวมูเลเตอร์ของไมโครโปรเซสเซอร์บางชนิดอาจมีมากกว่าหนึ่ง

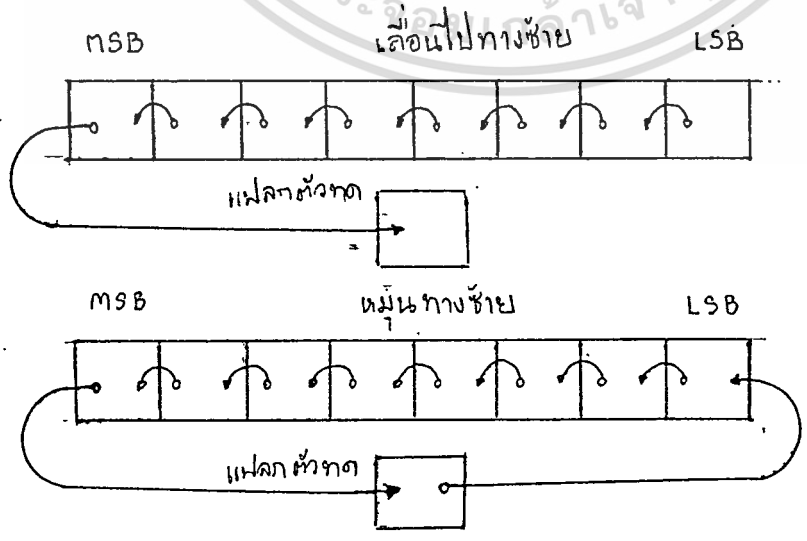
ที่ ALU ยังมีแฟลกรีจีสเตอร์ (Flag Register) ซึ่งใช้แสดงสภาพของข้อมูลและภาวะการคำนวณของ ALU อีกด้วย ALU จะติดต่อกับแฟลกรีจีสเตอร์ได้โดยตรง การทำงานของ ALU เช่นการป้อนข้อมูลเข้า การคำนวณ การส่งข้อมูลออก จะถูกควบคุมโดยวงจรควบคุม ซึ่งการควบคุมจะขึ้นอยู่ชนิดของคำสั่งอีกทีหนึ่ง

แฟลกรีจีสเตอร์ (Flag Register)

แฟลกรีจีสเตอร์เป็นรีจีสเตอร์ที่แสดงภาวะการทำงานของ ALU บางครั้งมีชื่อว่า Status Register ภายในรีจีสเตอร์นี้ แต่ละบิทจะแสดงภาวะการทำงานของ CPU ที่แตกต่างกัน แต่ละบิทเรียกว่า แฟลกฟลิปฟลอป (Flag Flip-Flop) แฟลกฟลิปฟลอปที่สำคัญมีแฟลกตัวทด (Carry Flag) แฟลกโอเวอร์โฟล (Overflow Flag) แฟลกเครื่องหมาย (Sign Flag) แฟลกศูนย์ (Zero Flag) แฟลกพาริตี้ (Parity Flag)

แฟลกตัวทด (Carry Flag) เป็นแฟลกที่ใช้ในการทดเลขที่เกิดจากการบวกในแอคคิวมูเลเตอร์ เปรียบเสมือนเป็นบิทที่เกินมาของแอคคิวมูเลเตอร์ เลื่อนและหมุนข้อมูลในแอคคิวมูเลเตอร์ แฟลกตัวทดก็เหมือนบิทที่เกินมาของแอคคิวมูเลเตอร์ เช่นเดียวกันการเลื่อนข้อมูลในแอคคิวมูเลเตอร์ ถ้าเลื่อนทางซ้ายจะเห็นว่าบิทที่ MSB (บิทแรกซ้ายสุด) จะถูกเลื่อนออกไปข้างนอกแฟลกตัวทดจะเป็นตัวรับข้อมูลนี้ การหมุนเป็นวงกลมก็ใช้แฟลกตัวทดเป็นตัวกลางระหว่างบิท MSB และ LSB ได้

แฟลกโอเวอร์โฟล (Overflow Flag) ใช้ในกรณีที่ ALU ทำการคำนวณทางเลขคณิตกับตัวเลขที่แสดงด้วย 2's Complement การแสดงเลขลบด้วย 2's Complement นี้ หลักสุดท้าย (MSB) จะแสดงเครื่องหมายบวกหรือลบของตัวเลข คือจะเป็น '1' เมื่อเป็นตัวเลขลบ และจะเป็น '0' เมื่อเป็นตัวเลขบวก เมื่อทำการบวกตัวเลขลบ 2 จำนวน อาจทำให้ MSB นี้เปลี่ยนจาก '1' เป็น '0' ได้ ซึ่งเป็นการผิด แฟลกโอเวอร์โฟลนี้จะใช้ในการตรวจจับความผิดพลาดนี้



<u>รีจีสเตอร์ข้อมูล</u> (Data Register)

รีจีสเตอร์ข้อมูลบางครั้งเรียกว่ารีจีสเตอร์สำหรับใช้งานทั่วไป (General Purpose Register) เป็นรีจีสเตอร์สำหรับเก็บข้อมูลชั่วคราวในขั้นตอนการคำนวณ รีจีสเตอร์นี้ต่อเข้ากับบัสภายใน สามารถส่งถ่ายข้อมูลเข้าออก ALU หรือ แอคคิวมูเลเตอร์ได้อย่างรวดเร็ว ข้อมูลจากหน่วยความจำซึ่งส่งผ่านเข้ามาทางบัสภายนอก มักจะถูกนำมาเก็บไว้ในงานแต่ละตัวอย่างอิสระแล้ว ยังสามารถใช้งานเป็นคู่รีจีสเตอร์ (Register) ได้อีกด้วย

<u>แอดเดรสรีจีสเตอร์</u> (Address Register)

แอดเดรสรีจีสเตอร์ เป็นรีจีสเตอร์ที่ใช้เก็บข้อมูลของแอดเดรส ในกรณีที่เป็นไมโครโปรเซสเซอร์แบบ 8 บิท จะมีแอดเดรสรีจีสเตอร์ซึ่งมีความยาว 16 บิท

การโหลด (Load) ข้อมูลเข้าแอดเดรสรีจีสเตอร์ สามารถทำได้โดยผ่านทางบัสข้อมูล

แอดเดรสรีจีสเตอร์มีหลายชนิดแบ่งตามการใช้งาน ได้แก่ โปรแกรมเคาว์เตอร์ (Program Counter) สแตคพอยเตอร์ (Stack pointer) อินเดกส์รีจิสเตอร์ (Index register) เป็นต้น

โปรแกรมเคาว์เตอร์ (PC) เป็นแอดเดรสรีจิสเตอร์ที่สำคัญจะมีในไมโครโปรเซสเซอร์ทุกชนิด มีไว้สำหรับชี้บอกแอดเดรสของคำสั่งต่อไปที่เก็บไว้ในหน่วยความจำ เมื่อไมโครโปรเซสเซอร์ทำงานของคำสั่งแรกเสร็จสิ้น จะอ่านคำสั่งต่อไป โดยการชี้บอกของ PC นี้ PC จะบวกหนึ่งให้กับตัวเองเสมอเมื่อไมโครโปรเซสเซอร์ทำการอ่านคำสั่ง เพื่อเป็นการชี้บอกตำแหน่งของคำสั่งต่อไป ด้วยการชี้บอกตำแหน่งกับการเพิ่มตำแหน่งให้ตัวเองนี้ ทำให้ไมโครโปรเซสเซอร์สามารถทำงานตามโปรแกรมตามขั้นตอนที่กำหนดไว้ได้

ข้อมูลภายใน (PC) สามารถนำไปเก็บในหน่วยจำได้ โดยส่งผ่านทางบัสข้อมูล นอกจากนั้น เรายังสามารถเปลี่ยนข้อมูลใน PC โดยการนำข้อมูลใหม่เข้าได้ ด้วยการที่เราสามารถเก็บข้อมูลและเปลี่ยนข้อมูลใน PC ได้นี้ ทำให้เราสามารถสั่งงานให้ไมโครโปรเซสเซอร์กระโดด (Jump) ไปทำงานในโปรแกรมย่อย (Subroutine) และกระโดดกลับมาทำงานในโปรแกรมเดิมได้

อินเดกส์รีจิสเตอร์ (IX) เป็นแอดเดรสรีจิสเตอร์ที่มีเฉพาะในไมโครโปรเซสเซอร์บางชนิดเท่านั้น มีไว้สำหรับการอ่านหรือเขียนข้อมูลจากหน่วยความจำเป็นชุด ๆ ได้ การโหลดข้อมูล การเพิ่มหนึ่งหรือลดหนึ่งที่อินเดกส์รีจิสเตอร์ สามารถทำได้โดยคำสั่ง อินเดกส์รีจิสเตอร์จะ

กระแสที่ใช้ในการไอแอสอินพุท

ในทางอุดมคติ ออป-แอมป์จะมีค่าอิมพีแดนซ์ทางอินพุทเป็นอนันต์ แต่ในความจริงแล้วไม่เป็นเช่นนั้น เนื่องจากทรานซิสเตอร์ที่ทำหน้าที่เป็นตัวรับอินพุทจำเป็นจะต้องมีกระแสไบแอสป้อนให้อยู่ค่าหนึ่ง จึงจะทำงานได้ ดังนั้นเมื่อกระแสนี้สามารถวัดค่าได้ จึงเป็นการชี้ให้เห็นว่าค่าของอินพุทอิมพีแดนซ์นั้นมีค่าสูงแต่ไม่ได้สูงจนกว่าเป็นอนันต์

ค่าของอิมพีแดนซ์นี้จะขึ้นอยู่กับชนิดของอุปกรณ์ สำหรับออป-แอมป์ที่มีอินพุทเป็นแบบไบโพลาร์ทรานซิสเตอร์จะต้องการกระแสไอแอสประมาณในช่วงนาโนแอมป์ ($X10^{-9}A$) ส่วนอินพุทที่เป็นแบบเจเฟ็ตยิ่งต้องการน้อยลงไปอีกและแบบมอสเฟ็ตต้องการน้อยที่สุดถึงประมาณเป็นพิโคแอมป์ ($X^{12}A$) เท่านั้น ซึ่งถือว่าใกล้เคียงกับคุณสมบัติในอุดมคติที่สุด โดยจะมีค่าอิมพีแดนซ์สูงถึง 10^{12} โอห์ม

กระแสไบแอสทางอินพุทนี้จะไม่ขึ้นอยู่กับค่าของแรงดันอินพุท จึงทำให้เป็นการง่ายที่จะวัดค่าของกระแสไบแอสนี้ ซึ่งสามารถวัดได้โดยใช้วงจรในรูปที่ 1 เพียงแต่ให้ค่าแรงดันที่อินพุทแล้วจึงอ่านค่ากระแสที่ไหลผ่านมิเตอร์ทั้งสองโดยตรง

กระแสออฟเซตทางอินพุท

สำหรับออป-แอมป์ในอุดมคติแล้วอินพุททั้งสองนั้นจะมีลักษณะที่เหมือนกันทุกอย่าง แต่ในความเป็นจริงแล้ว จะมีความแตกต่างกันระหว่างอินพุททั้งสองนี้ เนื่องจากทรานซิสเตอร์ภายในเป็นคนละตัวกันนั่นเอง

ความแตกต่างที่เกิดขึ้นทางอินพุทนี้อย่างหนึ่งที่สำคัญคือ ค่ากระแสออฟเซตทางอินพุท (I OS) ซึ่งผลก็คือผลต่างของกระแสไบแอสของอินพุททั้งสองนั่นเอง ดังนั้น ถ้ากำหนดให้ I B1 - I B2 เป็นค่ากระแสไบแอสของอินเวอร์ติ้งและนอน-อินเวอร์ติ้งอินพุทแล้วจะได้ว่า

$$I\ OS = I\ B1 - I\ B2$$

สำหรับในการใช้งานค่าของกระแสไบแอสทางอินพุทนี้จะใช้ค่าเฉลี่ยของกระแสทั้งสอง ดังนั้น

$$I\ B = (I\ B1 + I\ B2) / 2$$

แรงดันออฟเซตทางอินพุท

ผลที่เกิดจากความแตกต่างของทรานซิสเตอร์ที่อินพุทของออป-แอมป์อีกอย่างหนึ่งคือ แรงดันออฟเซต (V OS) ซึ่งจะเป็นผลของแรงดันที่ปรากฏที่เอาท์พุทในขณะที่อินพุททั้งสองต่อลงกราวด์

วงจรที่ใช้ในการวัดค่าของ V OS ได้แสดงไว้ในรูปที่ 2 ถ้าดูจากรูปวงจรแล้วแม้จะเห็นว่าเป็นการต่อขาอินพุทเพียงข้างเดียวลงกราวด์เท่านั้น แต่ขาอินพุทอีกข้างหนึ่งก็ถือว่าเป็นกราวด์เสมือน (Virtual Ground) ที่เป็นเช่นนี้เพราะว่าจากคุณสมบัติพื้นฐานของออป-แอมป์ในอุดมคติแล้ว จะมีค่าของแรงดันที่ขออินเวอร์ติ้งอินพุทเป็นกราวด์ด้วย

สำหรับในอุดมคติของออป-แอมป์แล้ว ถ้าไม่มีสัญญาณอินพุท ก็จะไม่มีแรงดันที่เอาท์พุทด้วย แต่ในความเป็นจริงแล้ว เนื่องจากความต่างกันของแรงดันที่เบส-อีมิตเตอร์ (หรือเกตซอร์สในกรณีที่เป็นเฟ็ต) ของทรานซิสเตอร์ ที่อินพุตทั้งสองจะทำให้เกิดแรงดันออฟเซตที่เป็นไฟตรงเกิดขึ้นที่เอาต์พุต ดังนั้นค่าแรงดันออฟเซตทางอินพุตนี้จะถูกหมายถึงค่าแรงดันที่จำเป็นต้องป้อนให้แก่อินพุตในกรณีที่จะทำให้แรงดันที่เอาต์พุตเป็นศูนย์ได้

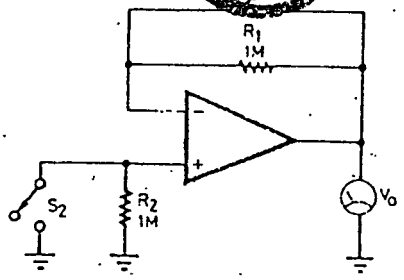
ขอให้กลับไปดูที่วงจรในรูปที่ 2 ลักษณะการต่อวงจรแบบนี้ จะเป็นการทำให้ออป-แอมป์ เกิดการชดเชยแรงดันที่อินพุตทั้งสองให้มีระดับเท่ากัน ด้วยแรงดันที่เอาต์พุตซึ่งมีค่าเท่ากับความแตกต่างของแรงดันที่อินพุททั้งสอง ดังนั้น ค่าของ V OS จึงสามารถอ่านค่าได้โดยตรงจากมิเตอร์ที่ต่ออยู่นั้น

สำหรับวงจรที่สามารถใช้ในการวัดค่าคุณสมบัติทั้งสามอย่างที่ได้กล่าวไปแล้วข้างต้น (กระแสไบแอส กระแสออฟเซตและแรงดันออฟเซตทางอินพุต) ได้แสดงไว้ในรูปที่ 3 ในการวัดค่า V OS คือ S1 และ S2 ค่าที่ต้องการสามารถอ่านได้โดยตรงจากมิเตอร์และในการวัดค่ากระแสไบแอสนั้น ก็สามารถทำได้ตามขั้นตอนดังนี้ ขั้นแรก เปิดวงจรสวิตช์ S1 และปิดวงจรสวิตช์ S2 เราจะสามารถอ่านค่าแรงดันได้ค่าหนึ่งจากมิเตอร์ซึ่งจะขอเรียกว่า V1 ต่อไป ก็ทำการปิดวงจรสวิตช์ S1 และเปิดวงจรสวิตช์ S2 ก็จะอ่านค่าแรงดันได้อีกค่าหนึ่งเรียกว่า V2 ดังนั้นค่า I B1 และ I B2 จะสามารถคำนวณได้จากสมการต่อไปนี้คือ

$$I_{B1} = V_{OS}$$

และ $$I_{B2} = V_2 - OS$$

ส่วนค่าของ IB หรือกระแสไบแอสจริง ๆ นั้นจะเป็นค่าเฉลี่ยของ I B1 และ I B2 นั่นเอง และ I OS จะเป็นค่าผลต่างของ I B1 และ I B2 ซึ่งสามารถคำนวณออกมาได้อย่างง่าย ๆ



รูปที่ 3- วงจรที่ใช้ในการวัดได้ทั้งค่าของแรงดันและกระแสออฟเซตโดยการเลือกการทำงานจากสวิตช์ S1 และ S2

ผลที่มีต่อออฟเซต

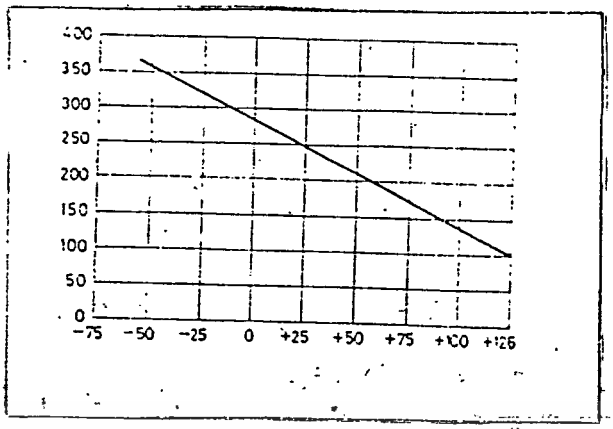
ค่าของ I OS และ V OS จะขึ้นอยู่กับกำลังขยายของทรานซิสเตอร์ภายในอินพุททั้งสอง โดยทั่วไปแล้วถ้ากำลังขยายของทรานซิสเตอร์ลดลง จะทำให้ผลของออฟเซตลดลงตามไปด้วย แต่อย่างไรก็ตาม การลดกำลังขยายของอินพุตลงนั้น ก็จะทำให้ต้องเพิ่มค่าของกระแสไบแอสทางอินพุตขึ้นจากเดิมและนั่นก็หมายความว่าค่าอิมพีแดนซ์ทางอินพุตก็จะต้องลดลงด้วย นอกจากนี้ค่ากำลังขยายแบบหลูปเปิด (open-loop gain) ของออป-แอมป์ จะมีผลโดยตรงกับค่าออฟเซตต่าง ๆ นี้ด้วย

ผลของอุณหภูมิ

ค่าของออฟเซตต่าง ๆ นั้นมีผลกับอุณหภูมิในขณะใช้งานมาก แต่ไม่เป็นเรื่องยากที่จะทำการชดเชยค่าของแรงดันและกระแสออฟเซตสำหรับออป-แอมป์ทั่ว ๆ ไป เพียงแต่ใช้ตัวความต้านทานปรับค่าได้ ต่อลงตามวงจรที่ระบุไว้ในคู่มือของออป-แอมป์ที่ใช้เท่านั้น

แต่ปัญหาที่เกิดขึ้นก็คือ เราจะทำอย่างไรเพื่อชดเชยค่าเหล่านั้น ที่เปลี่ยนไปตามอุณหภูมิในขณะใช้งาน แม้ว่าจะใช้ทรานซิสเตอร์ที่เหมือนกัน มาสร้างเป็นส่วนอินพุตของออป-แอมป์ก็ตามแต่ก็ยังมีผลที่เกิดจากอุณหภูมิแตกต่างกันเล็กน้อย ความแตกต่างนี้เกิดจากผลของอุณหภูมิที่แตกต่างกัน ในสารกึ่งตัวนำที่ใช้สร้างอุปกรณ์นั้น และจากผลของสารเจือที่ปะปนเข้าไปในขณะที่ทำการสร้างอุปกรณ์นั้นด้วย

ผลที่ตามมาก็คือ ทรานซิสเตอร์ตัวใดตัวหนึ่งจะมีค่าอัตราการเปลี่ยนแปลงของกระแสรั่วไหลหรือแรงดันที่เบสอิมิตเตอร์ ที่มีต่ออุณหภูมิสูงกว่าอีกตัวหนึ่ง ซึ่งจะทำให้มีผลต่อเอาต์พุตที่แตก



รูปที่ 4 กราฟแสดงความสัมพันธ์ของค่ากระแสไบแอสทางอินพุตกับอุณหภูมิในขณะใช้งาน

ต่างกัน ตัวอย่างเช่น ถ้าทำการชดเชยค่าออฟเซตให้เป็นศูนย์ที่อุณหภูมิห้อง แล้วทำการเพิ่มอุณหภูมิให้สูงขึ้นเป็น 125° ฟาเรนไฮต์ จะเห็นผลว่าจะเกิดออฟเซตขึ้นมาอีก ซึ่งเป็นผลอันเนื่องมาจากอุณหภูมิที่มีการเปลี่ยนแปลง

ออป-แอมป์ทั่วไป ได้ถูกออกแบบให้มีผลอันเนื่องมาจากการเปลี่ยนแปลงอุณหภูมิน้อยที่สุด ซึ่งก็แล้วแต่เบอร์ของออป-แอมป์ที่นำมาใช้งาน ผลที่เกิดขึ้นเนื่องจากการเปลี่ยนแปลงของอุณหภูมินี้สามารถทำการวัดได้โดยใช้วงจรในรูปที่ 3 เช่นกัน โดยในขณะที่สวิตช์ S และ S2 ถูกปิดวงจร แล้วทำการบันทึกค่าของ V OS ที่อุณหภูมิห้อง ต่อไปจึงเพิ่มอุณหภูมิขึ้น แล้วดูการเปลี่ยนแปลงของ V OS ที่เกิดขึ้นสำหรับการฟที่แสดงไว้ในรูปที่ 4 แสดงถึงความสัมพันธ์ของกระแสไบแอสอินพุตที่มีต่อการเปลี่ยนแปลงอุณหภูมิในขณะใช้งาน

การเปลี่ยนแปลงอันเนื่องมาจากอุณหภูมินี้ โดยปกติสามารถคำนวณได้จากค่าของสัมประสิทธิ์ทางอุณหภูมิซึ่งค่านี้ของแรงดันออฟเซต สามารถคำนวณได้จากสมการดังต่อไปนี้

สัมประสิทธิ์ทางอุณหภูมิ = SV os /ST

และค่าสัมประสิทธิ์ทางอุณหภูมิของกระแสออฟเซตก็สามารถคำนวณได้จากสมการต่อไปนี้เช่นกัน

สัมประสิทธิ์ทางอุณหภูมิ = SI os /ST

สัญลักษณ์ S หมายถึงค่าการเปลี่ยนแปลง ดังนั้น SV OS, SI OS และ ST ก็จะหมายถึงค่าการเปลี่ยนแปลงของ V OS , I OS และอุณหภูมิ

ผลของการเปลี่ยนแปลงอันเนื่องมาจากอุณหภูมินี้ มีหน่วยเป็นไมโครโวลต์ต่อองศา (uV/°C) และในหน่วยของไมโครแอมป์ นาโนแอมป์หรือพิโกแอมป์ต่อองศาเซลเซียส (uA/ °C, nA/°C, pA/°C) ซึ่งขึ้นอยู่กับชนิดของอุปกรณ์นั้น ๆ

ค่า CMRR

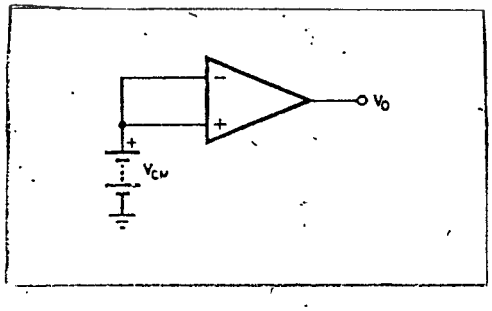
CMRR มาจากคำว่า Common Mode Rejection Ratio ซึ่งเป็นพารามิเตอร์ตัวหนึ่งที่มีความสำคัญมากกับออปแอมป์ ก่อนอื่นขอให้มาทำความเข้าใจเสียก่อนว่า CMRR นี่คืออะไร

สำหรับออป-แอมป์ในอุดมคติอินพุตทั้งสองคือทั้งอินเวอร์ติ้งและนอนอินเวอร์ติ้งอินพุตจะมีลักษณะที่เหมือนกันทุกประการ ดังนั้นโดยทฤษฎีแล้วเมื่อทำการป้อนสัญญาณที่เหมือนกันให้แก่อินพุตทั้งสอง ผลที่ได้คือเอาต์พุตจะเป็นศูนย์

แต่ความจริงแล้วผลที่แตกต่างกันของกำลังขยายที่ส่วนอินพุตทั้งสองนั้นทำให้เกิดแรงดันขึ้นที่เอาต์พุต ในขณะที่อินพุตมีลักษณะเป็นคอมมอน-โหมด (common-mode) นั่นคือมีระดับแรงดันที่อินพุตทั้งสองเท่ากัน ดังตัวอย่างที่แสดงไว้ในรูปที่ 5: การทำให้อินพุตอยู่ในลักษณะที่เป็นคอมมอน-โหมดนั้นโดยการต่อเชื่อมอินพุตทั้งสองเข้าด้วยกันแล้วป้อนด้วยค่าแรงดันค่าหนึ่งที่สูกว่ากราวด์ ค่าแรงดันเอาต์พุตของวงจรนี้คือค่าแรงดันผิดพลาดในคอมมอน-โหมด (common-mode error voltagee)

ถึงอย่างไรก็ตามค่าแรงดันผิดพลาดในคอมมอน-โหมดนี้ ก็เป็นค่าพารามิเตอร์ที่ไม่ค่อยมีประโยชน์มากนัก เนื่องจากค่านี้เปลี่ยนแปลงไปตามกำลังขยายของออป-แอมป์ และค่าแรงดันอินพุตในคอมมอน-โหมด ซึ่งจะเห็นได้ว่าเป็นการขึ้นอยู่กับวงจรแต่ละวงจร จึงเป็นการยากที่จะนำมาเป็นตัวเปรียบเทียบระหว่างออป-แอมป์แต่ละตัวได้

ค่าอัตราส่วนระหว่างค่าแรงดันเอาต์พุตและอินพุตในคอมมอน-โหมดนั้น จะเป็นค่าคงที่ที่ไม่ขึ้นอยู่กับโครงสร้างของวงจรแต่ละอัน ซึ่งเราเรียกพารามิเตอร์ตัวนี้ว่า Common-Mode Rejectjion Ratio หรือเรียกสั้น ๆ ว่า CMRR สูงเท่าใด จะทำให้ค่าแรงดันผิดพลาดในคอมมอน-โหมดลดลงเท่านั้น



รูปที่ 5 แสดงถึงการทำให้อินพุตมีลักษณะเป็นคอมมอน-โหมด ดังนั้น อัตราส่วนระหว่างค่าแรงดันเอาต์พุต และอินพุตในลักษณะนี้ถูกเรียกว่าค่า CMRR

ถึงแม้ว่าค่า CMRR นี้จะมีลักษณะเป็นค่าคงที่แบบไฟตรงก็ตาม แต่ก็ไม่ได้เป็นการยากเลยที่จะทำการวัดค่านี้โดยใช้สัญญาณที่ป้อนให้แก่อินพุตเป็นแบบไฟสลับดังวงจรที่แสดงไว้ในรูปที่ 6

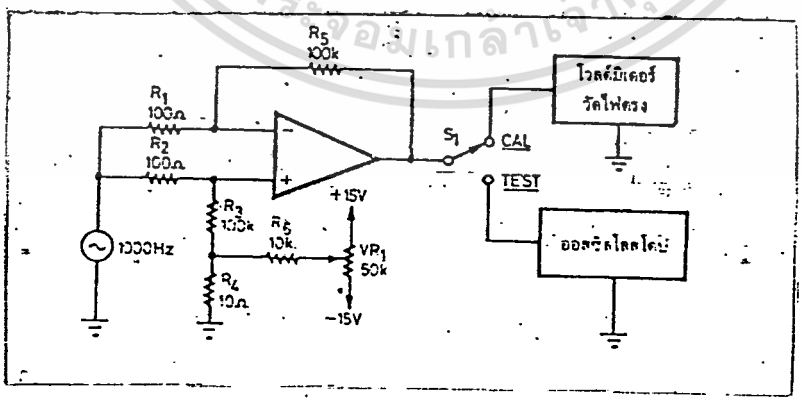
จากวงจรนี้จะเห็นได้ว่า สัญญาณอินพุตแบบคอมมอน-โหมดที่ให้นั้นเป็นสัญญาณไฟสลับความถี่ 1 กิโลเฮิรตซ์ที่ป้อนแก่อินพุตโดยผ่านความต้านทาน R1 และ R2 ดังนั้น ค่าแรงดัน คอมมอน-โหมดที่ออป-แอมป์ได้รับจริง ๆ นั้น คือ ค่าแรงดันที่คร่อมตัวความต้านทาน R3 และ R_4 ที่ต่ออนุกรมกันอยู่ ส่วน VR1 ที่เป็นตัวความต้านทานปรับค่าได้ (50 K) และ R5 นั้นใช้สำหรับเป็นตัวชดเชยค่าแรงดันออฟเซตทางอินพุต

ในการใช้งานของวงจรนี้ ขั้นแรกยังไม่ต้องป้อนสัญญาณไฟสลับที่อินพุตให้แก่วงจร ปรับค่าความต้านทาน 50 K นั้นจนกระทั่งโวลต์มิเตอร์ไฟตรงที่ต่ออยู่ที่เอาต์พุตนั้นอ่านค่าได้เป็นศูนย์ในขณะนี้สวิตช์ S1 อยู่ในตำแหน่ง CAL นั่นก็หมายความว่าเราได้ชดเชยค่าแรงดันออฟเซตทางอินพุทจนมีค่าเป็นศูนย์แล้ว ต่อไปก็ทำการป้อนสัญญาณไฟสลับที่อินพุทความถี่ 1 กิโลเฮิรต์ โดยมีค่าแอมปลิจูดเท่ากับ 1 โวลต์ ปรับสวิตช์ S1 ให้มาอยู่ในตำแหน่ง TEST วัดค่าแรงดันของสัญญาณไฟสลับที่เอาต์พุท (V0) โดยใดยใช้ออสซิลโลสโคป และคำนวณค่า CMRR ได้ดังสมการ

$$CMRR = 100/V_0$$

ค่า CMRR นี้จะมีค่ามาก ซึ่งอาจจะทำให้ยุ่งยากต่อการนำไปใช้งาน ดังนั้นโดยทั่วไปแล้วค่า CMRR นี้จะถูกกำหนดให้อยู่ในหน่วยของเดซิเบล สมการของ CMRR จึงเปลี่ยนรูปเป็นดังนี้

$$CMRR\ dB = 20 \log (100\ V_0)$$



รูปที่ 6 วงจรที่ใช้ในการวัดค่า CMRR โดยใช้สัญญาณอินพุทเป็นแบบไฟสลับความถี่ 1 กิโลเฮิร์ต ภายในวงจรมี VR1 ที่ใช้ในการปรับค่าชดเชยแรงดันออฟเซตทางอินพุท

นั่นก็หมายความว่า เราได้ชดเชยค่าแรงดันออฟเซ็ตทางอินพุทจนมีค่าเป็นศูนย์แล้ว ต่อไปก็ทำการป้อนสัญญาณไฟสลับที่อินพุทความถี่ 1 กิโลเฮิรตซ์ โดยมีค่าแอมปลิจูดเท่ากับ 1 โวลต์ ปรับสวิตช S1 ให้มาอยู่ในตำแหน่ง Test วัดค่าแรงดันของสัญญาณไฟสลับที่เอาท์พุท (V_o) โดยใช้ออสซิลโลสโคป และคำนวณค่า CMRR ได้ดังสมการ

$$CMRR = 100 / V_o$$

ค่า CMRR นี้จะมีค่ามาก ซึ่งอาจจะทำให้ยุ่งยากต่อการนำไปใช้งาน ดังนั้นโดยทั่วไปแล้วค่า CMRR นี้จะถูกกำหนดให้อยู่ในหน่วยของเดซิเบล สมการของ CMRR จึงเปลี่ยนรูปเป็นดังนี้

$$CMRR\ dB = 20 \log (100 V_o)$$

หลักการบันทึกข้อมูลและอ่านข้อมูลจากเทปแม่เหล็ก

2.1 สารแม่เหล็ก

สารแม่เหล็กแบ่งเป็น 3 ประเภทคือ

2.1.1 DIAMAGNETIC มีความซึมซาบสนามแม่เหล็กต่ำ และสภาพการเกิดขั้วแม่เหล็กตรงข้ามกับสนามที่ใส่เข้าไป

2.1.2 PARAMAGNETIC มีความซึมซาบสนามแม่เหล็กต่ำ แต่สภาพการเกิดขั้วแม่เหล็กเกิดด้านเดียวกันกับสนามที่ใส่เข้าไป

2.1.3 FERROMAGNETIC มีความซึมซาบสนามแม่เหล็กสูง และสภาพการเกิดขั้วแม่เหล็กเกิดด้านเดียวกับสนามที่ใส่เข้าไป

ตัวอย่างสารแม่เหล็กประเภทต่าง ๆ ได้แสดงไว้ในตารางที่ 2.1

Substance	Group type	Relative permeability μ_r
Bismuth	Diamagnetic	0.99983
Silver	Diamagnetic	0.99998
Lead	Diamagnetic	0.999983
Copper	Diamagnetic	0.999991
Water	Diamagnetic	0.999991
Vacuum	Nonmagnetic	1†
Air	Paramagnetic	1.0000004
Aluminum	Paramagnetic	1.00002
Palladium	Paramagnetic	1.0008
2-81 Permalloy powder (2 Mo, 81 Ni)‡	Ferromagnetic	130
Cobalt	Ferromagnetic	250
Nickel	Ferromagnetic	600
Ferroxcube 3 (Mn-Zn-ferrite powder)	Ferromagnetic	1,500
Mild steel (0.2 C)	Ferromagnetic	2,000
Iron (0.2 impurity)	Ferromagnetic	5,000
Silicon iron (4 Si)	Ferromagnetic	7,000
78 Permalloy (78.5 Ni)	Ferromagnetic	100,000
Mumetal (75Ni, 5Cu, 2Cr)	Ferromagnetic	100,000
Purified iron (0.05 impurity)	Ferromagnetic	200,000
Supermalloy (5 Mo, 79 Ni)§	Ferromagnetic	1,000,000

† By definition.
‡ Percentage composition. Remainder is iron and impurities.
§ Used in transformer applications with continuous tape-wound (gapless) cores.

ตารางที่ 2.1 แสดงตัวอย่างสารแม่เหล็กประเภทต่าง ๆ

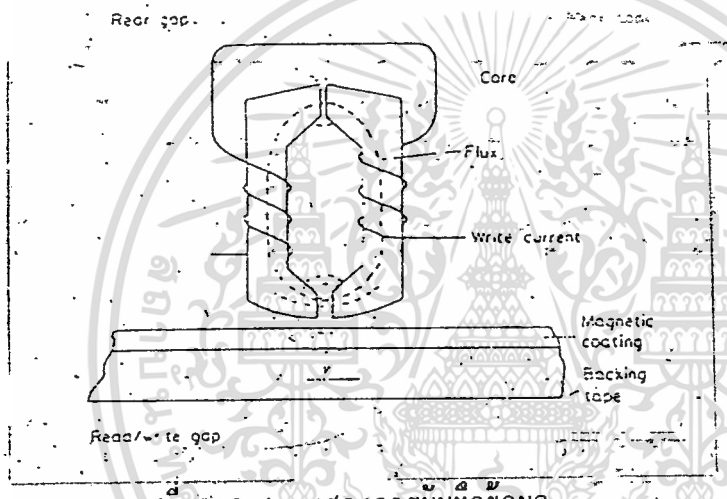
ตารางที่ 2.1 แสดงตัวอย่างสารแม่เหล็กประเภทต่าง ๆ

การบันทึกข้อมูล

การบันทึกข้อมูลจากหัวเทปลงบนบัตรเคลือบแถบสารแม่เหล็ก มีหลักการว่าระหว่างหัวเทปกับบัตรต้องมีการเคลื่อนที่ โดยแบ่งเป็น 2 กรณี คือ

1. หัวเทปอยู่กับที่แล้วบัตรเคลื่อนที่ผ่าน
2. หัวเทปเคลื่อนที่ผ่านบัตรที่อยู่กับที่

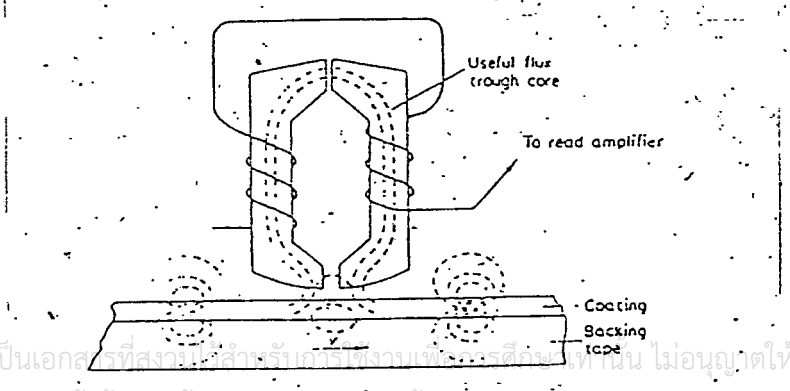
สัญญาณที่จะใช้ในการบันทึกจะเป็นกระแสไฟฟ้าที่ไหลผ่านขดลวดในหัวเทป ซึ่งจะทำให้เกิดสนามแม่เหล็ก และสนามแม่เหล็กนี้จะทำให้เกิดการเรียงตัวของเซสแม่เหล็กที่อยู่บนบัตร ทำให้เกิดสภาพแม่เหล็กอยู่บนบัตร ดังรูป 2.2.1



รูปที่ 2.2.1 แสดงการบันทึกข้อมูล

การอ่านข้อมูล

สำหรับในการอ่านข้อมูลจากบัตร ก็ใช้หลักการทำนองกลับกัน เมื่อขดลวดเคลื่อนที่ตัดกับสนามแม่เหล็ก ก็จะทำให้เกิดการเหนี่ยวนำสภาพแม่เหล็กในขดลวด ทำให้เกิดกระแสไฟฟ้าไหลในขดลวดซึ่งจากสภาพของกระแสไฟฟ้าจากขดลวดก็จะทำให้ทราบว่าสัญญาณที่บันทึกเข้ามาเป็นอย่างไรดังรูป 2.2.2



รูปที่ 2.2.2 แสดงถึงการอ่านข้อมูล

2.2 รูปแบบในการอ่านบันทึกข้อมูล

ในการบันทึกข้อมูลลงบนแถบแม่เหล็กนั้นปรกติ จะใช้ในช่วงอิ่มตัวของสภาพแม่เหล็ก เนื่องจากในการอ่านข้อมูลกลับอ่านได้ง่ายขึ้น, เพราะว่าศักดาไฟฟ้าสูง

2.3.1 RETURN-TO-ZERO (RZ)

ในแบบนี้ Logic 1 ใน Digital จะถูกแทนด้วยสภาพอิ่มตัวของแม่เหล็ก สำหรับ Logic 0 จะไม่มีการป้อนกระแสไฟฟ้าให้กับหัวบันทึก ความกว้างของสัญญาณบนบัตรขึ้นอยู่กับคุณสมบัติของหัวเทปและคุณสมบัติของเนื้อแม่เหล็ก ถ้าสัญญาณนี้สั้นไป ศักดาไฟฟ้าที่อ่านกลับมาจะมีค่าต่ำลง

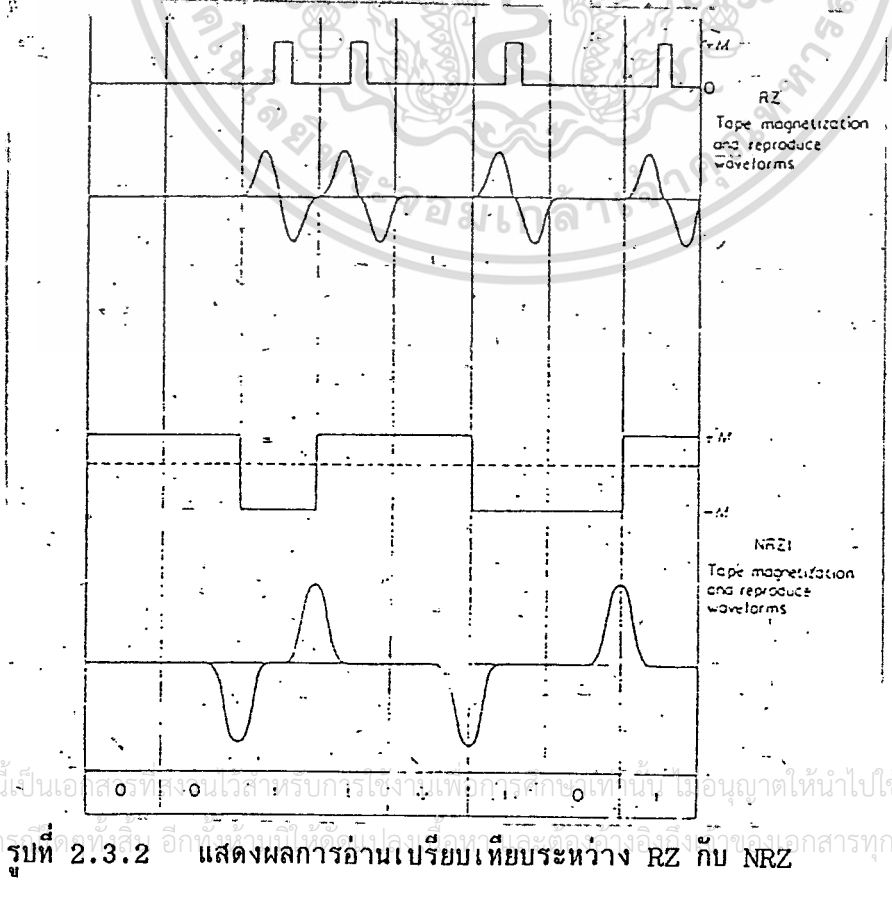
ในระบบนี้ Logic 0 กับสภาพไม่มีข้อมูลเหมือนกัน ดังนั้นความแตกต่างของสภาพ Logic จึงมีแค่ 50% และในระบบนี้ยังต้องมีการส่ง Clock ออกไปด้วย มิฉะนั้นจะอ่านข้อมูลไม่ได้

จากข้อจำกัดในระบบนี้จึงได้มีการพัฒนาเป็นแบบถัดไป

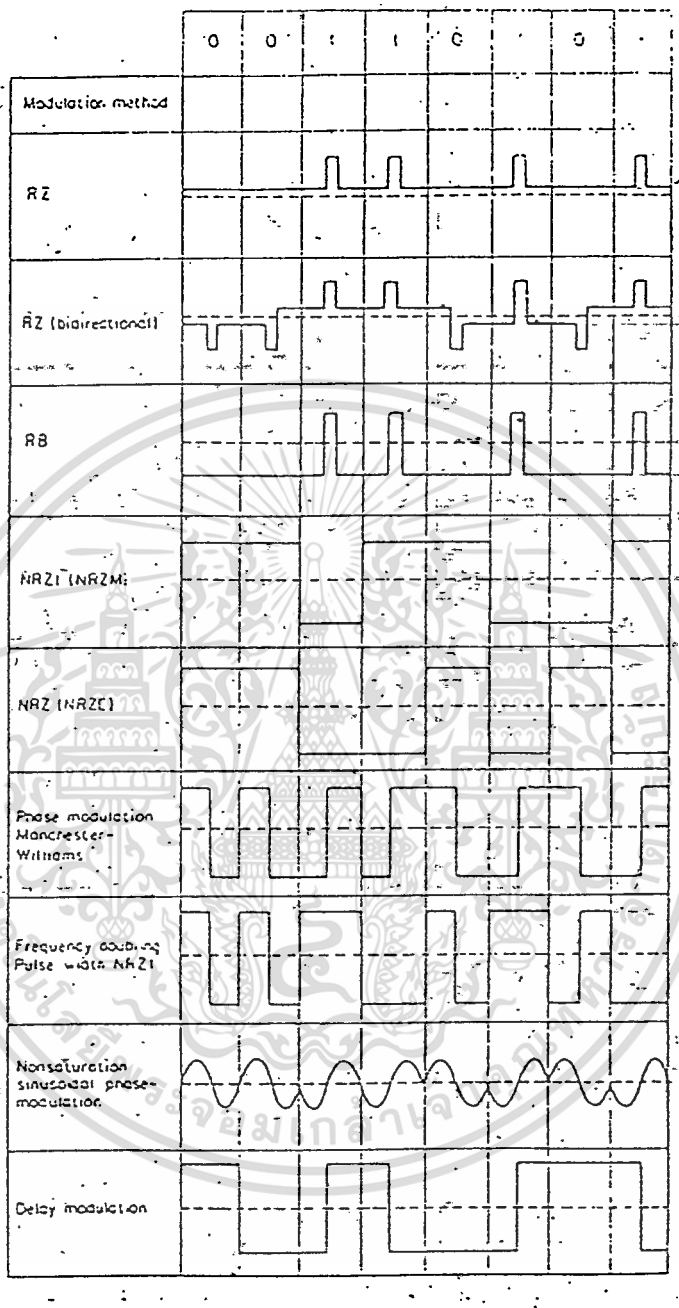
2.3.2 RETURN-TO-ZERO (BIDIRECTIONAL)

ในแบบนี้ แถบสารแม่เหล็กจะถูกทำให้อิ่มตัวในสภาพขั้วหนึ่งเมื่อข้อมูลเป็น Logic 1 และจะถูกทำให้อิ่มตัวในสภาพขั้วตรงกันข้ามใน Logic 0 ระบบนี้จะกลายเป็นแบบ Self-Clock และการอ่านข้อมูลทำได้ง่าย

ระบบ RZ นี้ Noise-Immunity มีค่าต่ำและจากสัญญาณที่ได้จากการอ่านข้อมูลของแบบนี้เทียบกับแบบ NRZ ดังในรูป 2.3.2 จะเห็นได้ว่า แบบ NRZ จะมีความแน่นของข้อมูลสูงกว่า



รูปที่ 2.3.2 แสดงผลการอ่านเปรียบเทียบระหว่าง RZ กับ NRZ



รูปที่ 2.3.1 แสดงรูปแบบในการบันทึกข้อมูล

RETURN-TO-BIAS (RB)

ในแบบนี้ ปรกติหัวเทปที่ใช้ในการบันทึกข้อมูลจะถูกทำให้อิ่มตัวในทิศทางหนึ่ง เมื่อสัญญาณข้อมูลเป็น Logic 1 จะทำให้เกิดสภาพอิ่มตัวในทิศตรงกันข้าม ในช่วงเวลาสั้น ๆ ในระบบนี้ค่าศักดาไฟฟ้าที่อ่านได้จะมีค่าสูงเนื่องจากใช้ในช่วงอิ่มตัวด้านหนึ่งถึงอิ่มตัวอีกด้านหนึ่ง

NON-RETURN-TO-ZERO

มี 2 แบบคือ

- NRZ or NRZC (NRZ-CHANGE) ในระบบนี้ หัวเทปบันทึกข้อมูลจะถูกทำให้อิ่มตัวไม่ด้านใดก็ด้านหนึ่ง โดยเมื่อข้อมูลมีการเปลี่ยน Logic (ไม่ว่าจาก Logic 1 เป็น Logic 0 หรือจาก Logic 0 เป็น Logic 1) จะมีการเปลี่ยนสภาพการอิ่มตัวไปด้านตรงข้าม ระบบนี้ต้องมี Clock ช่วย ระบบนี้ได้ข้อมูลมากกว่าแบบ RZ แต่ถ้าเกิดการผิดพลาดของข้อมูลที่ตำแหน่งใด ข้อมูลทั้งหมดก็จะผิดตามไปด้วย

- NRZ or NRZM (NRZ-MARK) เป็นแบบที่ใช้กันมาก หัวเทปบันทึกข้อมูลจะมีการเปลี่ยนสภาพการอิ่มตัวไปในทางสภาพอิ่มตัวตรงกันข้าม ทุกครั้งที่เจอ Logic 1 ถ้าข้อมูลตัวใดเกิดการผิดพลาดข้อมูลที่เหลือจะไม่เกิดปัญหา

PHASE MODULATION (MANCHESTER-WILLIAMS)

ความแตกต่างของ Logic อยู่ที่ขอบของสัญญาณที่เวลา T/S (T = คาบเวลาของข้อมูล) ถ้าเป็น Logic 1 ที่เวลา T/2 จะเป็นขอบขาขึ้น ถ้าเป็น Logic 0 ที่เวลา T/2 จะเป็นขอบขาลงระบบนี้เป็นแบบ Self-Clock

FREQUENCY DOUBLING PLUSE WIDTH (NRZ1)

โดยที่ความแตกต่างของ Logic คู่ที่แตกต่างของความถี่ ซึ่งจะเป็น 2 เท่าของกัน ปัญหาคือ ศักดาไฟฟ้าที่ได้จากการอ่านข้อมูลของทั้ง Logic 1 และ Logic 0 จะไม่เท่ากัน

NONSATURATION SINUSOIDAL PHASE-MODULATION

ในแบบนี้ถ้ามีการเปลี่ยน Logic จะเกิดการเปลี่ยนเฟส ไปเป็นตรงกันข้าม เวลาอ่านเจอการกลับเฟสก็รู้ว่ามีการเปลี่ยน Logic

DELAY MODULATION

ถ้าเป็น Logic 0 จะมีการเปลี่ยนแปลงสภาพความอิ่มตัวของสนามแม่เหล็ก ที่รอยต่อของข้อมูล แต่ถ้าเป็น Logic 1 จะมีการเปลี่ยนแปลงสภาพความอิ่มตัวของสนามแม่เหล็ก ที่ตรงกลางของข้อมูล

จากรูปแบบในการบันทึกข้อมูลที่กล่าวมาแล้วนั้น ไม่เหมาะสมในการใช้เป็นรูปแบบ เนื่องจากในการอ่านข้อมูลเราใช้มือรูดบัตรผ่านหัวอ่าน ซึ่งทำให้ความถี่ที่อ่านได้มีค่าไม่คงที่ ดังนั้น รูปแบบที่ใช้แยกความแตกต่างของข้อมูลโดยใช้ความถี่ จึงไม่เหมาะสม และรูปแบบที่มีการจ่ายกระแสให้กับหัวบันทึกในขณะที่ไม่มีการอัดข้อมูล ก็ไม่เหมาะสม ดังนั้นจึงใช้รูปแบบที่สร้างขึ้นเอง

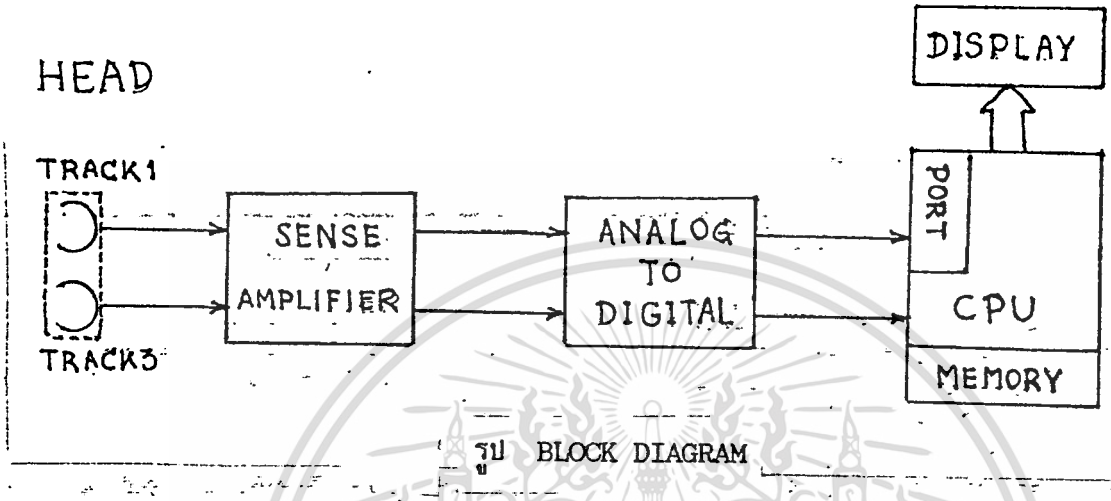
สำหรับในการสร้างเครื่องบันทึกข้อมูลและอ่านข้อมูล ชุดนี้ใช้รูปแบบในการบันทึกข้อมูล ซึ่งรูปแบบที่ใช้แบบนี้ มีข้อดีที่

- ขนาดของศักดาไฟฟ้า ที่อ่านได้จากหัวอ่านมีค่าเท่ากัน ถึงแม้ข้อมูลจะต่างกัน ทำให้วงจรภาครับไม่ยุ่งยาก

- ในภาวะปรกติที่ไม่มีการอัดข้อมูล จะไม่มีการจ่ายกระแสให้หัวบันทึกข้อมูล ทำให้หัวบันทึกข้อมูลไม่เสื่อมเร็ว

- จากรูปแบบที่สร้างขึ้น สัญญาณที่ใช้สามารถสร้างได้ง่าย วงจรไม่ยุ่งยาก ทั้งในส่วนของเครื่องบันทึกข้อมูล และอ่านข้อมูล

2.3 หลักการของเครื่องอ่านแถบแม่เหล็ก



รูป BLOCK DIAGRAM

2.3.1. HEAD

การอ่านแถบแม่เหล็ก มีลักษณะการอ่าน 2 แบบ

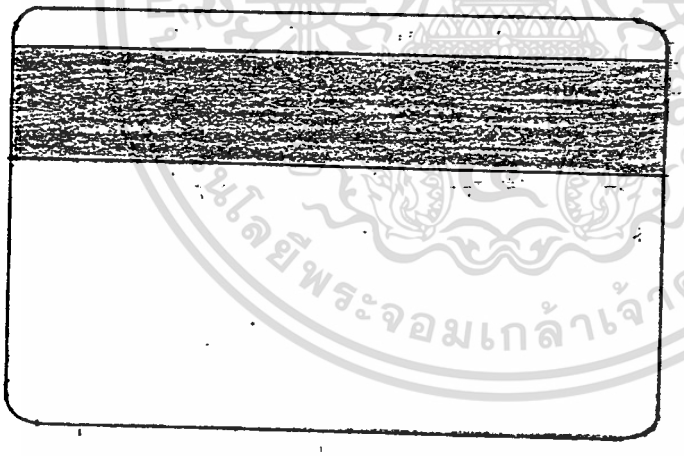
1.1 แบบหัวอ่านเคลื่อนที่ โดยให้แถบแม่เหล็กอยู่กับที่ โดยที่หัวอ่านจะเคลื่อนที่ไปบนแถบแม่เหล็ก

1.2 แบบหัวอ่านอยู่กับที่ โดยให้แถบแม่เหล็กเคลื่อนที่ผ่านหัวอ่าน ซึ่งกรณีนี้มีทั้งแบบใช้มือรูดบัตรผ่านหัวอ่านและอีกวิธีคือใช้มอเตอร์เป็นตัวเลื่อนบัตรให้ผ่านหัวอ่าน

หัวอ่าน (Header Reader) คือ หัวอ่านบัตรนี้จะมีลักษณะคล้ายกับหัวเทปวิทยุคลาสเซ็ทซึ่งมีลักษณะดังรูป และหัวเทปที่ใช้อ่านบัตร (Card) นี้จะใช้แบบหัวเทปสเตอริโอและเป็นแบบรีเวิส (Reverse) คือจะเป็นลักษณะที่มี 4 แท็ป (Tap) ใช้ในการอ่านข้อมูลในขณะที่มีบัตรรูดเข้ามาผ่านบริเวณหัวด่าน หัวอ่านนี้ก็จะทำให้หน้าที่อ่านข้อมูลที่เก็บไว้หรือบันทึกไว้อยู่ในสารแม่เหล็กบนแผ่น Card นั้น ๆ ว่าข้อมูลที่บันทึกลงในแผ่น Card นั้นมีข้อมูลอะไรที่บันทึกไว้ ก็จะถูก

หัวอ่าน อ่านออกมาและส่งต่อไปยังภาค Sense Amp อีกที หัวอ่านนี้ได้บอกแล้วว่าเป็นแบบสเตอริโอ (Stereo) เพราะฉนั้นจะแบ่งออกเป็น 2 ชั้น คือ Channal A และ Channal B โดยแต่ละ Channal จะทำหน้าที่อ่านข้อมูลจากบัตรออกมาเพียงแต่ว่าข้อมูลของแต่ละ Channal จะทำหน้าที่อ่านข้อมูลจากบัตรออกมาเพียงแต่ว่าข้อมูลของแต่ละ Channal จะมีข้อมูลอะไร เก็บไว้เท่านั้นเอง คือ เราให้ที่ Channal A ทำหน้าที่อ่านข้อมูล (Data) ที่ถูกบันทึกเอาไว้ที่เป็นข้อมูลที่เราต้องการนำไปเพื่อแสดง (Display) ว่าเก็บอะไรอยู่ใน Card นั้นว่ามีรหัสอะไร ส่วน Channal B จะทำหน้าที่อ่านสัญญาณนาฬิกา (Clock) เพื่อมา Reference กับข้อมูลใน Channal A เพื่อที่เราจะได้ทราบค่าของข้อมูลว่าข้อมูลใน Channal A เก็บข้อมูลอะไรไว้

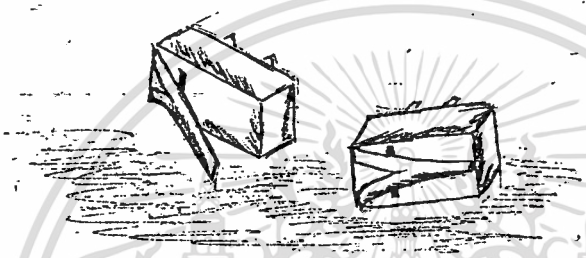
2.3.2: <u>บัตร</u> (Card)



คือบัตรที่เราจะอ่านข้อมูลใน Card นี้ว่ามีข้อมูลอะไรอยู่ที่ Write ไว้ในสารแม่เหล็ก โดย Card นี้จะแบ่งช่องออกเป็น 3 ช่อง (Track) โดยแต่ละช่องจะมีข้อมูลอยู่ทั้งสิ้น โดยแต่ละช่องจะ Write ข้อมูลอะไรไว้ก็แล้วแต่การ Format

2.3.3. Switch Detect

คือเป็นสวิทช์ (Switch) ที่จะทำหน้าที่ตรวจจับว่ามีบัตร (Card) สอดใส่เข้ามาหรือยัง ถ้ามีบัตรสอดใส่เข้ามา Switch นี้ก็จะทำงานและก็ส่ง Signal ไปยัง CPU เพื่อบอกว่ามีการสอดใส่บัตรเข้ามาแล้ว เพื่อให้ CPU เตรียมรับข้อมูลจากหัวเทปที่อ่านเข้ามา



จากรูป กล่าวได้คือ เมื่อมีการรูดบัตรเข้ามา ที่ SW Detect ก็จะทำหน้าที่คือส่ง Signal ให้เป็น 0 คือ หัวอ่านก็จะอ่านข้อมูลจาก Card โดยมี Channal A เป็น Data และ Channal B เป็น Reference Clock Data ที่บันทึกไว้ใน Card นี้ซึ่งจะขึ้นอยู่กับการ Write ข้อมูลอะไรไว้ โดยจะเทียบค่าไปทีละ Bit จนครบ 8 Bit และส่งต่อไปยังภาคหลัง

2.3.4. Sense Amplifier

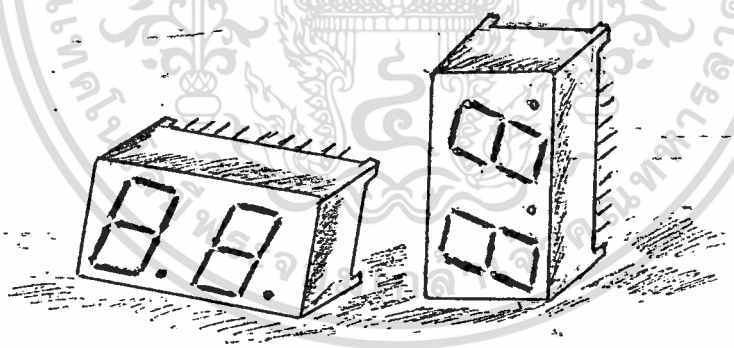
คือวงจรขยายสัญญาณที่ได้รับจากหัวอ่าน (Head Reader) เพื่อมาขยายสัญญาณให้แรงขึ้นอีกเพื่อที่จะได้ส่งต่อไปอีก เนื่องจากต้องมีวงจร Sense Amp นี้ก็เพราะว่าสัญญาณที่ได้จากหัวอ่านนี้ จะมีสัญญาณอ่อนมากไป ไม่เหมาะกับที่จะส่งสัญญาณนี้ไปยังภาค CPU โดยตรงได้เลย วงจร Sense Amp นี้ เราจะใช้ Op-Amp เบอร์ 741 เป็นตัวขยายสัญญาณโดยให้มี Gain การขยายมากพอสมควร และวงจร Sense Amp เมื่อขยายสัญญาณแรงพอสมควรแล้วก็จะส่งไปยัง Rectifier เพื่อทำหน้าที่ให้สัญญาณที่เข้ามานี้เรียบพอสมควรและทำให้ช่วงลบ By Pass ลง Ground เพื่อที่จะได้เหลือสัญญาณเฉพาะช่วงบวกเท่านั้นพอแล้วก็นำไปผ่าน Schmitt trigger เพื่อทำให้เป็นสัญญาณเป็น Pluse เพื่อส่งไปยังภาค CPU ต่อไปอีกทีหนึ่ง

2.3.5. Analog to Digital

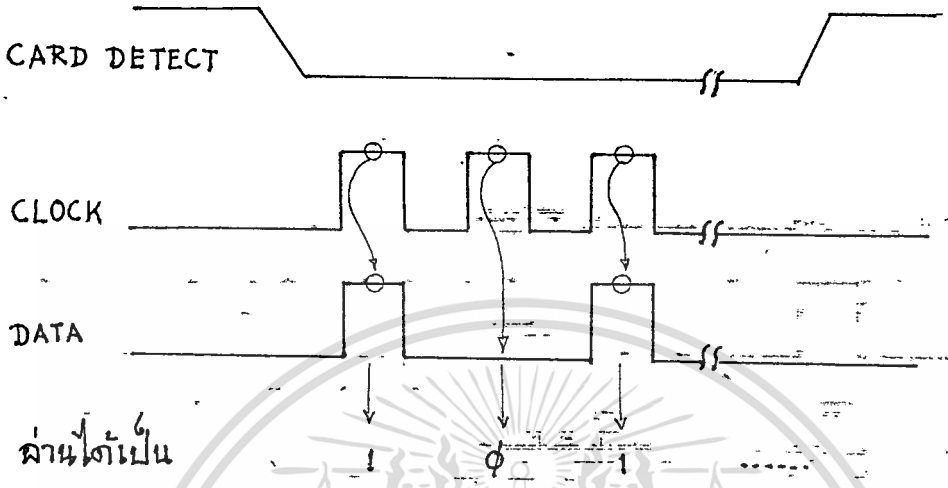
ถึงแม้ว่าเราจะอัดสัญญาณเป็นดิจิตอล (Digital) เข้าไปในแถบแม่เหล็ก สัญญาณที่เราอ่านได้ออกมาก็ยังไม่เป็นดิจิตอล (Digital) จริง ๆ ดังนั้นเมื่อสัญญาณที่ได้จากหัวอ่าน (Head) ผ่าน Sense Amplifier มา สัญญาณที่ได้ออกมาจริง ๆ เมื่อใช้สโคปจับดูปรากฏว่าเป็นสัญญาณที่มีลักษณะอนาล็อก (Analog) และสัญญาณที่ได้ออกมามีระดับสูงมาก ดังนั้นเราจึงนำมาลดระดับให้เหลือ 0-5 โวลท์ เพื่อให้ CPU สามารถอ่านค่าทางลอจิก (Logic) นั้นได้

Display

คือเราจะใช้ LED Seven Segment ทำหน้าที่เป็นตัวแสดงผลลัพธ์ (Output) ให้เห็นว่า Card ที่รูดเข้ามานั้นมีรหัส (Code) อะไร Write อยู่ใน Card นั้น เพื่อตรวจดูว่าตรงกับรหัสที่ป้อนไว้หรือไม่



Program



จาก Timing Diagram ดูเทียบกับ Flow Chart ซึ่งโดยหลักการ เริ่มแรกเราจะ Check ว่ามีการรูด Card เข้ามาหรือไม่ โดยมี Micro Switch ไว้คอย Delete ช่วง Low เมื่อ CD เป็น Low แล้ว เราถึงจะมา Check Clock ว่ามี Clock มาแล้วหรือยัง ถ้ามี Clock มา จะ Delay เวลาไปสักเล็กน้อย แล้วจึงอ่าน Data ว่าเป็น 1 หรือ 0 จากนั้นก็นำไป Shift เข้าไปเก็บไว้ใน Register Buffer (R7) จนกว่าจะครบ 8 Bit แล้วจึงนำไปเก็บไว้ใน RAM (20H-3FH) โดยผ่าน R6 ซึ่งเป็น Pointer-Buffer ของ Program นี้ จะ Check Clock แล้วอ่าน Data เก็บ Bit Data จนครบ 8 Bit เช่นนี้เรื่อย ๆ จนกว่า Port ที่รับ Signal ของ Card Detect ว่า Card ได้รูดผ่านพ้นไปแล้ว ก็จะหยุดการอ่าน และจะนำเอาค่า ID ของ Data ที่นำไปเก็บไว้ใน RAM ของ CPU ออกมา Display

การจัด Memory Register

การจัด Register : R0 - General Purpose

R1 - Column Position of Digit

R2 - Bit Count

R3 - Count Delay

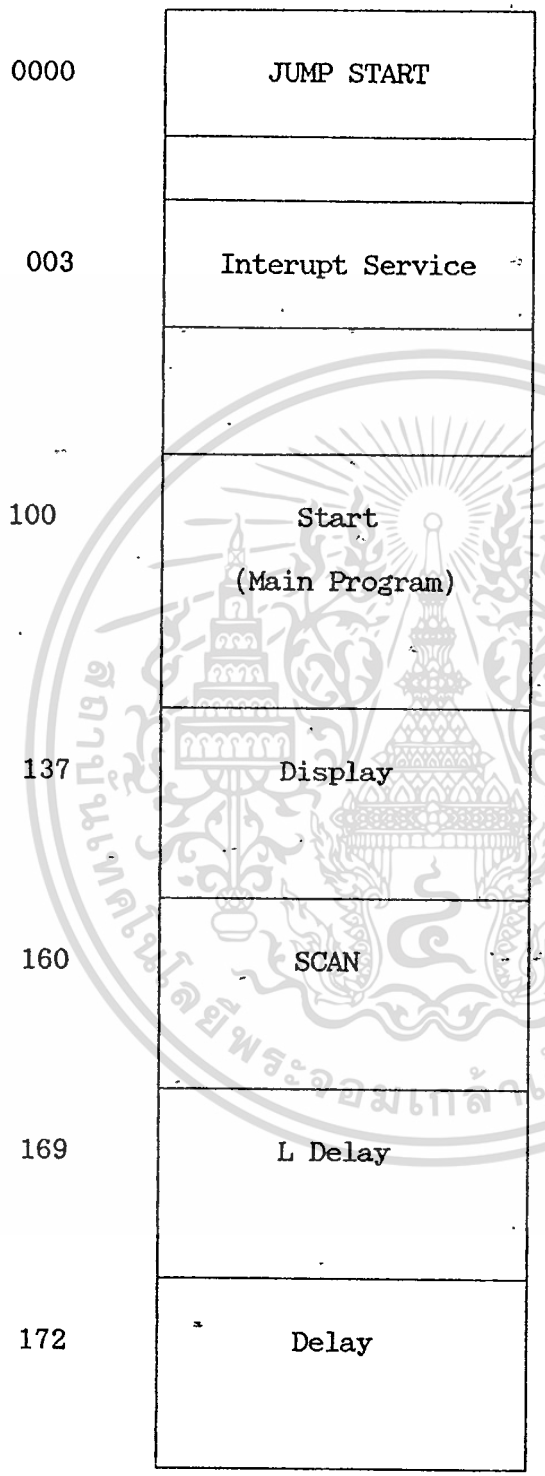
R4 - Count Delay

R5 - Main Pointer Adr.

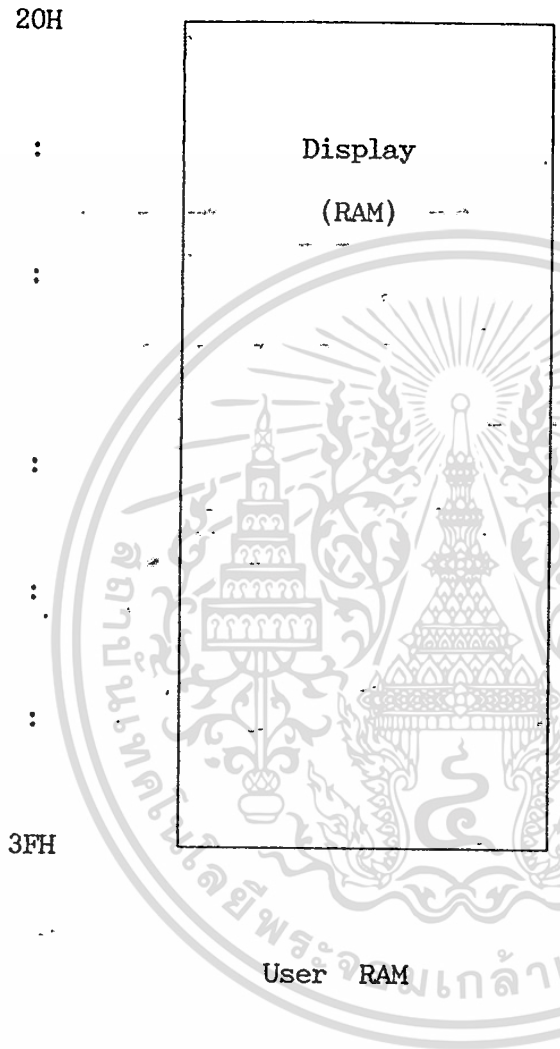
R6 - Pointer Buffer Adr.

R7 - Byte Buffer

การจัด Mememory



Operation System



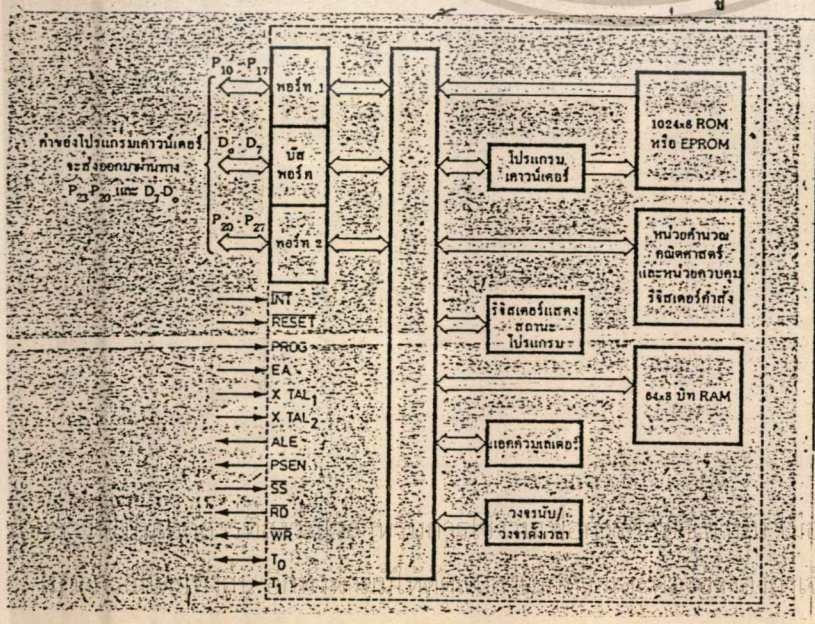
2.4 8040 : โครงสร้างที่ซับซ้อนแต่ใช้งานง่าย

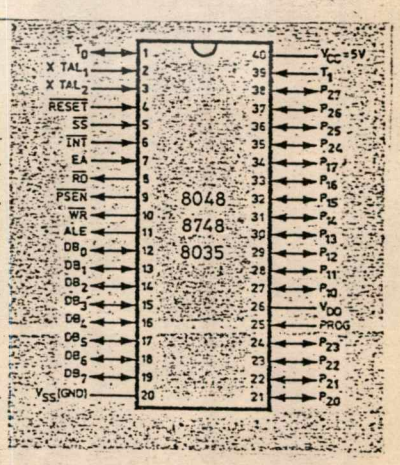
โครงสร้างของ 8048 ก็เหมือนกับโครงสร้างของไมโครคอมพิวเตอร์ทั่วไป กล่าวคือ ประกอบด้วยบัสที่ทำหน้าที่เชื่อมต่อกับอุปกรณ์อื่น ๆ โดยขาของไอซีที่ออกมาส่วนหนึ่งจะเป็นขาอินพุท-เอาท์พุท

ไมโครคอมพิวเตอร์ที่เห็นนี้มีส่วนเอาท์พุท-อินพุท 3 พอร์ท โดยพอร์ทหนึ่งทำหน้าที่เสมือนเป็นข้อมูลบัส และพอร์ทอินพุทเอาท์พุทนี้ยังมี 12 สายที่ส่งสัญญาณมัลติเพล็กซ์แทรกมาเป็นสัญญาณแอดเดรส คือ บิท P23 - P20 และ D7 - D0

เบอร์	หน่วยความจำในชิป		ช่วงเวลาไซเคิล	พอร์ทอินพุท เอาท์พุท	อินเตอร์รัพท์	วงจรตั้งเวลา	จำนวนขา	สามารถขยายได้	A/D
	EPROM / ROM	RAM							
8048	ROM1024	64	2.5μS	3x8 bit	1	มี	40	ได้	ไม่มี
8035	0	64	2.5μS	3x8 bit	1	มี	40	ได้	ไม่มี
8035-8	0	64	5.0μS	3x8	1	มี	40	ได้	ไม่มี
8748	1024EPROM	64	2.5μS	3x8	1	มี	40	ได้	ไม่มี
8748-8	1024EPROM	64	5.0μS	3x8	1	มี	40	ได้	ไม่มี
8049	2048ROM	64	1.4μS	3x8	1	มี	40	ได้	ไม่มี
8041	1024ROM	64	2-5	3x8	0	มี	40	ไม่ได้	ไม่มี
8741	1024EPROM	64	2-5	3x8	0	มี	40	ไม่ได้	ไม่มี
8021	1024ROM	64	10μ	2x8 1x4	0	มี	28	ไม่ได้	ไม่มี
8022	2048ROM	64	10μS	3x8 bit	1	มี	40	ไม่ได้	มี

สืบเนื่องจากไอซีในตระกูล 8048 มีมากมายหลายเบอร์ แต่ละเบอร์มีโครงสร้างสถาปัตยกรรมในการรับรู้คำสั่งทางซอฟท์แวร์เหมือนกัน แต่จะแตกต่างกันทางฮาร์ดแวร์ ตารางที่ 1 นี้เป็นตารางแสดงรายละเอียดของไอซีไมโครคอมพิวเตอร์ในตระกูลนี้





ขอบเขตทางสถาปัตยกรรมของไอซีตระกูล 8048

8048 เป็นไมโครคอมพิวเตอร์ที่มีความสามารถมากตัวหนึ่ง ตัว 8048 มี ROM ในตัว 1Kx8 บิท ส่วน 8748 มี EPROM และ 8035 ไม่มี ROM ในตัว ทั้งสามตัวนี้มี RAM ที่ทำหน้าที่เป็นส่วนเก็บข้อมูลได้ 64 ไบท์ในแผ่นชิปของมัน

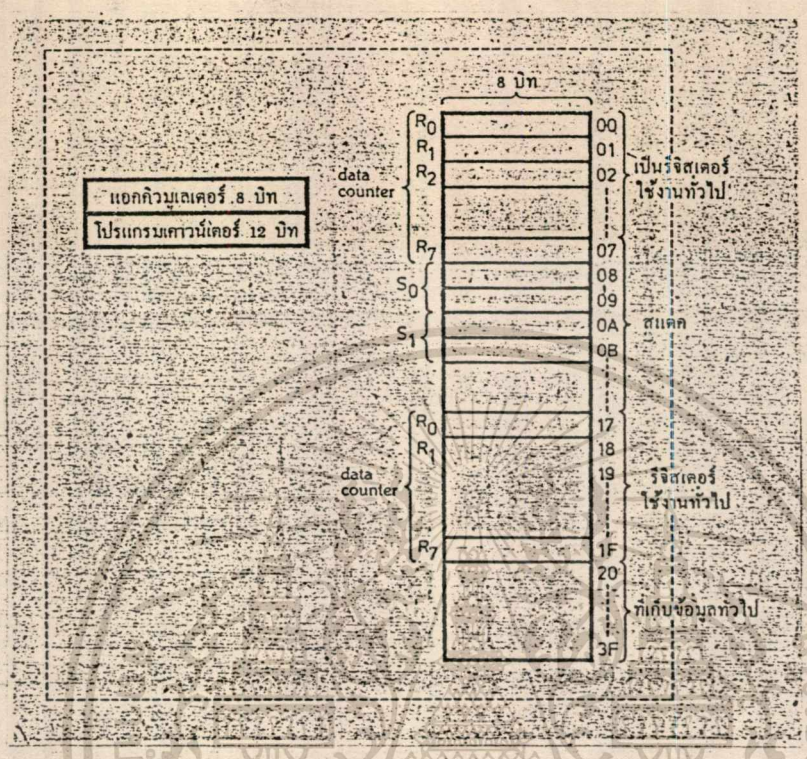
8048 มีโปรแกรมเคาว์เตอร์ 12 บิท ซึ่งจะทำให้มันสามารถแอดเซสข้อมูลโปรแกรมได้สูงถึง 4K แต่เนื่องจาก 8048 และ 8748 สามารถทำงานร่วมกับ ROM ภายในของมันแล้ว 1K ดังนั้น มันจึงสามารถอ้างถึงหน่วยความจำภายนอกได้อีก 3072 ไบท์ ส่วน 8035 ต้องอ้างถึงหน่วยความจำทั้งหมดจาก ROM ภายนอก

8048, 8035 และ 8748 มีโครงสร้างสำหรับอินพุท-เอาท์พุท 3 พอร์ท หนึ่งในสามพอร์ทนี้ใช้เป็นบัสพอร์ท ส่วนบัสพอร์ทจะทำหน้าที่มัลติเพล็กซ์สัญญาณแอดเดรสและข้อมูลเข้าด้วยกัน ส่วนอินพุทเอาท์พุทพอร์ทที่เหลือมีลักษณะสำคัญคือ ลักษณะเอาท์พุทของมันจะมีคุณสมบัติการแลทช์อยู่ ส่วนอินพุทนั้นเป็นแบบไม่มีแลทช์ กล่าวคือ ทั้งพอร์ท 1 และพอร์ท 2 จะทำหน้าที่เหมือนอินพุท-เอาท์พุทเหมือนกัน โดยเมื่อมีการส่งเอาท์พุทมายังพอร์ทนี้ มันจะแลทช์ไว้จนกว่าจะมีการส่งข้อมูลมาใหม่อีกครั้งมันจึงจะทับข้อมูลเดิมของมันออกไป แต่สำหรับกรณีอินพุทจะแตกต่างจากหลักการทั่วไปคือ การอ่านข้อมูลอินพุทของซีพียูจะได้ตามสถานะของการใช้ลอจิกจากภายนอกมาดึงขาอินพุทเอาท์พุทให้เป็น "0" ทั้งนี้เพราะ ถ้าซีพียูส่งข้อมูลใดเป็นเอาท์พุทมา การอ่านกลับจะได้เช่นนั้น แต่ถ้าเรามีการดึงบางขาที่เป็น "1" ให้เป็น "0" ผลการอ่านจะได้เป็น "0" แต่ถ้าซีพียูส่ง "0" ออกมาแล้ววงจรภายนอกพยายามดึงเป็น "1" ซีพียูก็ยังคงอ่านได้ "0" อยู่นั่นเอง

สถาปัตยกรรมของ 8048, 8748 และ 8035

กลุ่มของไอซี 8048 เป็นไมโครคอมพิวเตอร์ชนิด 8 บิท โดยตัวซีพียูมีแอคคิวมูเลเตอร์ขนาด 8 บิท มีโปรแกรมเคาว์เตอร์ขนาด 12 บิท และยังมีส่วนของหน่วยความจำภายในจำนวน 64 ไบท์ ส่วนของหน่วยความจำภายในนี้สามารถได้รับการเขียนหรืออ่านได้ แต่ในขณะเดียวกันก็ทำงานในรูปของการเป็นรีจิสเตอร์ทั่วไปด้วย ภายในโครงสร้างของซีพียูจึงมีลักษณะเป็นดังรูปที่ 4

แอคคิวมูเลเตอร์ เป็นรีจิสเตอร์หลักที่จะมีส่วนในการเข้าไปกระทำในส่วนของ ALU ในทางคำสั่งเกี่ยวกับทางคณิตศาสตร์ และลอจิก การทำงานของแอคคิวมูเลเตอร์จะถือเป็นโอเปอร์แรนด์ตัวหนึ่งร่วมกับรีจิสเตอร์อื่นหรือหน่วยความจำ



ในหน่วยความจำภายใน 64 ไบท์ จะมี 8 ไบท์แรกทำหน้าที่เหมือนเป็นรีจีสเตอร์ใช้งานทั่วไป และยังมีส่วนแอดเดรส 17-20 อีกชุดหนึ่งที่ทำหน้าที่เป็นรีจีสเตอร์ทั่วไปด้วย ทุกครั้งที่มีการเรียกชุดรีจีสเตอร์หนึ่งจะไม่ได้รับการเลือก

รีจีสเตอร์ทั่วไปนี้สองตัวแรกใช้เป็นตัวสำหรับแอดเดรสข้อมูลในหน่วยความจำ ซึ่งโดยทั่วไปเราใช้ R0 และ R1 เป็นตัวชี้แอดเดรส

นอกจากนี้ยังมีส่วนของหน่วยความจำแบบสแตค รายละเอียดของการใช้งานสแตคจะได้กล่าวถึงต่อไป

การแอดเดรสของ 8048 : นอกจากใช้โปรแกรมเคาน์เตอร์แล้ว ยังใช้รีจีสเตอร์อื่นด้วย

8048 ให้โครงสร้างการจัดหน่วยความจำที่แปลกต่างกับไมโครคอมพิวเตอร์ทั่วไป กล่าวคือ หน่วยความจำที่ใช้กับ 8048 จะแยกออกเป็น 2 ส่วนคือ หน่วยความจำสำหรับเก็บข้อมูล ส่วนของหน่วยความจำเก็บโปรแกรมจะมีได้มากที่สุดถึง 4096 ไบท์ ส่วนหน่วยความจำเก็บข้อมูลก็จะมีได้มากที่สุดถึง 320 ไบท์

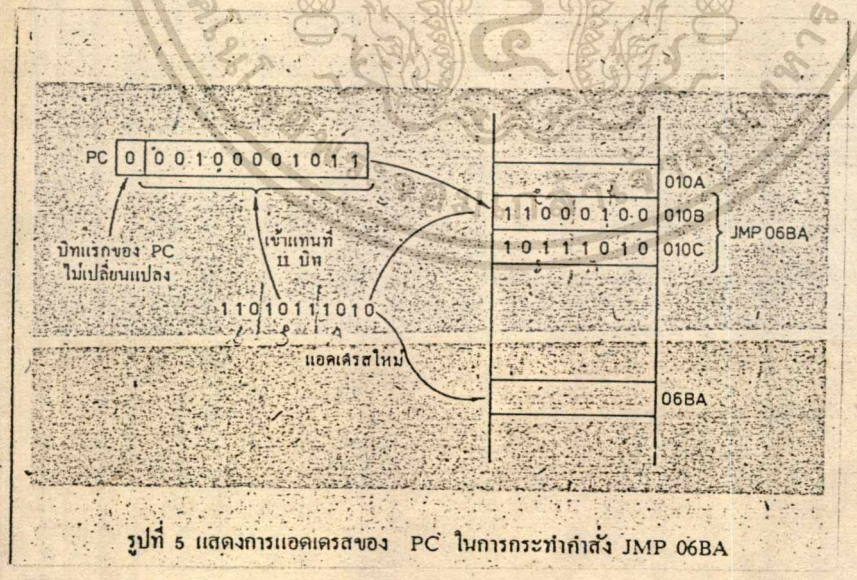
8048 และ 8748 มีส่วนหน่วยความจำในตัวแล้ว 1024 ไบท์ หน่วยความจำนี้เป็นหน่วยความจำเก็บโปรแกรมด้วย ROM ภายนอก ซึ่งจะเก็บได้สูงสุด 4K เช่นกัน

ตระกูล 8048 มีหน่วยความจำเก็บภายใน 64 ไบท์ และสามารถขยายหน่วยความจำภายนอกเพื่อเก็บข้อมูลได้อีก 256 ไบท์ โดยหน่วยความจำเก็บข้อมูลภายนอกนี้จะใช้ร่วมกับส่วนของหน่วยความจำสำหรับพอร์ทอินพุทหรือเอาท์พุท กล่าวคือ การเรียกพอร์ทอินพุทหรือเอาท์พุท ซีพียูสามารถแอดเดรสได้ 256 พอร์ทเช่นกัน

ในที่นี้จะกล่าวถึงวิธีการแอดเดรสในแต่ละส่วนแยกจากกัน

การแอดเดรสในส่วนหน่วยความสำหรับโปรแกรม

สืบเนื่องจากส่วนของ PC (โปรแกรมเคาว์เตอร์) เป็นรีจีสเตอร์ขนาด 12 บิท ดังนั้น PC จึงแอดเดรสหน่วยความจำได้โดยตรง 4096 ตำแหน่ง แต่อย่างไรก็ตาม การจัดโครงสร้างของหน่วยความจำเก็บโปรแกรม เราแบ่งหน่วยความจำออกเป็นสองแบล็งค์ แต่ละแบล็งค์มีขนาด 2K ดังนั้นโดยปกติการแอดเดรสผ่านทาง PC จะกระทำได้เพียงข้อมูล 11 บิท หรือภายใน 2K เท่านั้น การเรียกหรือกระทำคำสั่งเกี่ยวกับ JMP, CALL หรือ RET จึงทำได้ภายในหน่วยความจำ 2K นี้เท่านั้น ดังตัวอย่างของการกระทำคำสั่ง JMP ดังรูปที่ 5

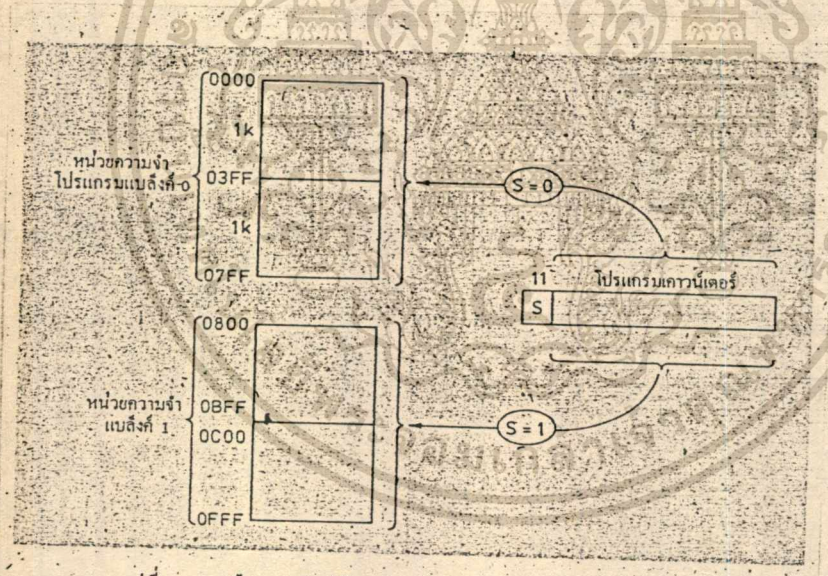


รูปที่ 5 แสดงการแอดเดรสของ PC ในการกระทำคำสั่ง JMP 06BA

แรกเริ่มเดิมที PC มีค่า 010B ซึ่ง PC จะส่งแอดเดรสไปเฟทช์คำสั่งในตำแหน่ง 010B มา คำสั่งที่เฟทช์มานี้เป็นคำสั่ง JMP 06BA ซึ่งซีพียูจะนำข้อมูล 11010111010 โหลดไปเก็บใน PC โดยบิทนัยสำคัญสูงสุดของ PC ไม่เปลี่ยนแปลง กล่าวคือ คำสั่ง JMP นี้จะไม่สามารถทำให้ CPU ข้ามแบล็งค์ได้เลย

การข้ามแบล็งค์ของหน่วยความจำโปรแกรมนี้ได้ต่อเมื่อเราให้ซีพียูกระทำคำสั่ง JMP, CALL หรือ RET ร่วมกับคำสั่ง SEL MB หรือเลือกแบล็งค์หน่วยความจำนั่นเอง

การเลือกแบล็งค์หน่วยความจำนี้ จะมีผลต่อบิท 11 ของโปรแกรมเคาวน์เตอร์นั่นเอง พิจารณาการทำงานดังรูปที่ 6



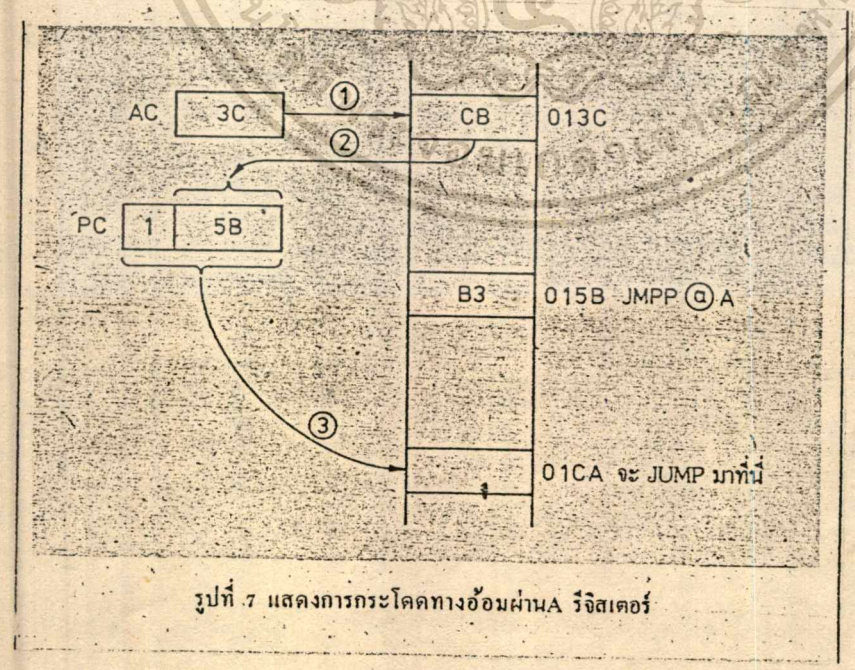
รูปที่ 6 การเลือกแบล็งค์มีผลต่อการเซทบิท 11 ของ PC นั่นเอง

การแบ่งโครงสร้างของหน่วยความจำโปรแกรมออกมาเป็นแบล็งค์ โดยแต่ละแบล็งค์ยังสามารถแบ่งแยกกลุ่มของหน่วยความจำให้เล็กลงโดยแบ่งเป็นเพจ (Page) แต่ละเพจจะมีหน่วยความจำขนาด 256 ไบท์ ดังนั้น ใน 1 แบล็งค์จะแบ่งออกเป็น 8 เพจ หรือหน่วยความจำโปรแกรมทั้งหมดแบ่งออกเป็น 16 เพจ การแอดเดรสภายในเพจจะใช้ข้อมูลเพียง 8 บิทเท่านั้น จึงทำให้กรรมวิธีในการแอดเดรสมีประสิทธิภาพดียิ่งขึ้น เพราะสามารถทำการแอดเดรสแบบทางอ้อมได้ โดยใช้ข้อมูลภายในแอคคิวมูเลเตอร์ขนาด 8 บิท นั่นเอง ลองมาดูวิธีการกระโดดทางอ้อมดูบ้าง

พิจารณาตัวอย่างการกระโดดแบบทางอ้อมในสองกรณีนี้ดู

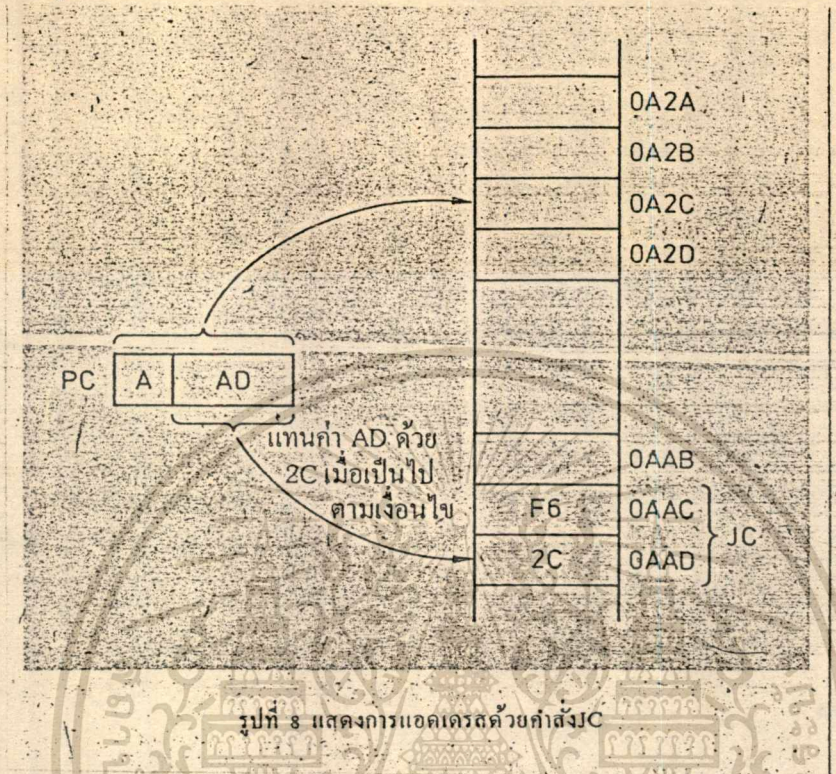
กรณีที่ 1 เป็นการ JMPP@A หรือเป็นการ JUMP แบบทางอ้อมผ่านข้อมูลในแอคคิวมูเลเตอร์ดู

กรณีนี้เดิมข้อมูลใน PC มี 15B ซึ่งจะทำการเฟทช์คำสั่งที่ตำแหน่ง 015B มา ซึ่งในที่นี้เก็บคำสั่ง JMPP@A ซึ่งมีรหัส 3B คำสั่งนี้สั่งให้กระทำการกระโดด โดยทางอ้อมผ่านรีจีสเตอร์ A วิธีการคือ ข้อมูลในแอคคิวมูเลเตอร์จะเป็นแอดเดรสชี้ไปยังหน่วยความจำ เพื่อโหลดข้อมูลในหน่วยความจำกลับมาใส่ใน PC การใส่ใน PC จะใส่ในเฉพาะ 8 บิทล่างเท่านั้น ผลคือการกระโดดมาตำแหน่ง 01CB ซึ่งยังคงอยู่ในเพจเดียวกัน



รูปที่ 7 แสดงการกระโดดทางอ้อมผ่านA รีจิสเตอร์

กรณีที่ 2 เป็นการกระโดดไปด้วยเงื่อนไข เช่น JC 2C ลักษณะการทำงานเขียนเป็นไดอะแกรมได้ดังรูปที่ 8



รูปที่ 8 แสดงการแอดเดรสด้วยคำสั่ง JC

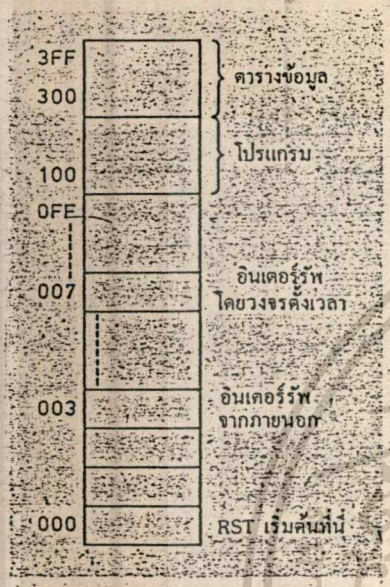
ข้อสังเกต ในส่วนของหน่วยความจำสำหรับโปรแกรมจะเป็นส่วนที่เราสามารถอ่านข้อมูลมาทำการตรวจสอบหรือทำอะไรได้ แต่เราไม่สามารถเขียนโปรแกรมลงในส่วนนี้ได้เลย เราจะไม่มีคำสั่งใดที่ทำให้เกิดการเขียนลงในหน่วยความจำนี้ได้

หน่วยความจำโปรแกรม : หน่วยความจำที่อ่านได้อย่างเดียว

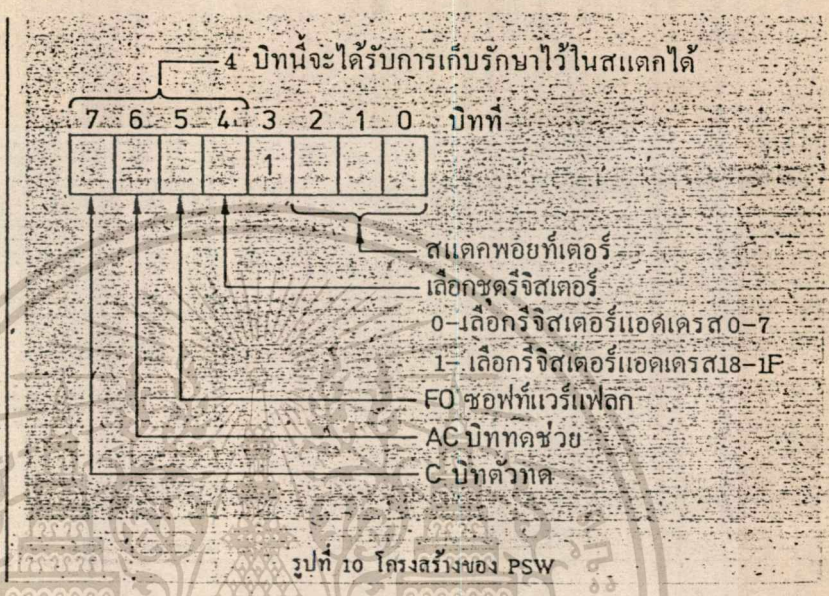
โครงสร้างหน่วยความจำมราเก็ลโปรแกรมของ 8048 มีโครงสร้างการจัดตำแหน่งดังรูปที่ 9 โดยแอดเดรส 000 เป็นแอดเดรสเริ่มต้น และ RST ส่วนการอินเตอร์รัพท์จะเริ่มที่แอดเดรส 003 แต่ถ้าเป็นการอินเตอร์รัพท์โดย CTC จะเริ่มที่ที่แอดเดรส 007 และจากที่กล่าวในหัวข้อก่อนว่า การแอดเดรสแบบง่าย ๆ โดยชุดคำสั่งจะอ้างถึงภายในเพจจะสะดวก ดังนั้นโปรแกรมเราไว้ที่ 100 เป็นจุดเริ่มต้นของเพจนั่นเอง

เวิร์ดแสดงสถานะของโปรแกรม (PSW - Program Status Work)

ภายในซีพียูของไมโครคอมพิวเตอร์ตระกูล 8048 นี้ มีรีจีสเตอร์ขนาด 8 บิท ตัวหนึ่งที่ทำหน้าที่เก็บสถานะของโปรแกรมไว้ โดยเราเรียกว่า PSW รีจีสเตอร์ PSW เป็นรีจีสเตอร์ที่เก็บสแตคพอยเตอร์และสถานะโปรแกรมที่สำคัญดังรูปที่ 10



รูปที่ 9 แสดงการจัดโครงสร้างหน่วย

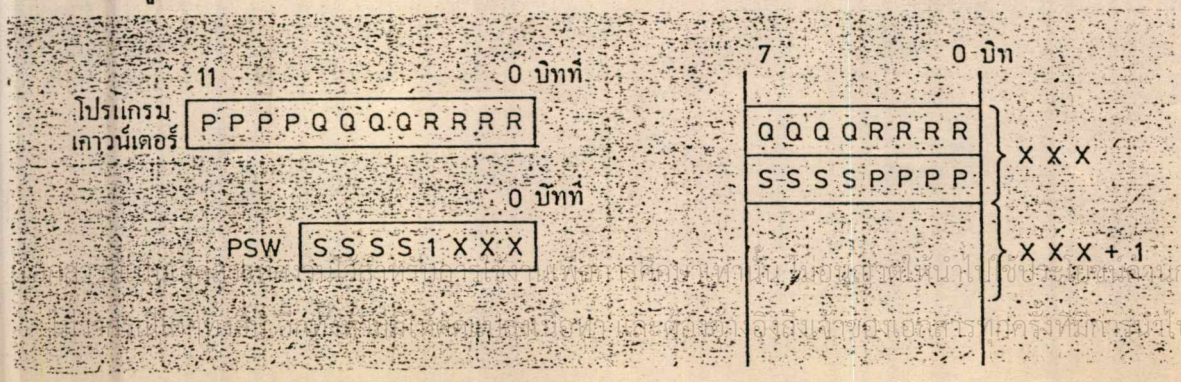


รูปที่ 10 โครงสร้างของ PSW

บิท 0 ถึงบิท 2 เป็นสแตคพอยเตอร์ ซึ่งถ้าพิจารณาในที่นี้จะเห็นว่าจะชี้ได้ 8 ตำแหน่ง สแตคพอยเตอร์ของ 8048 ก็เหมือนกับ 8080 แต่ชี้ได้เพียง 8 ตำแหน่งเท่านั้น โดยตำแหน่งของสสแตคจะเริ่มจากแอดเดรส 08 โดยคู่ 08 กับ 09 คือ S0 ดังนั้น 3 บิทนี้จึงใช้ได้ 3 ตัวคือ จาก S0 ถึง S8

แฟลก C และ AC เหมือนกันใน 8080 แฟลก ส่วนแฟลก F0 แฟลกนี้สามารถได้รับการเซ็ทหรือรีเซ็ทตามสถานะของคำสั่งได้ ซึ่งสามารถทดสอบแฟลก F0 นี้ได้ด้วยคำสั่งกระโดดตามเงื่อนไข

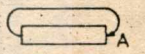
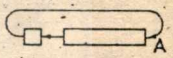
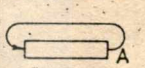
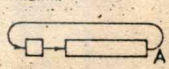
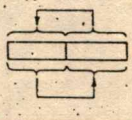
เมื่อมีคำสั่ง CALL หรือคำสั่งที่ต้องใช้เกี่ยวกับสแตคจะเก็บได้ถึง 16 บิท โดยมีโครงสร้างการเก็บดังรูปที่ 11



ชนิดคำสั่ง	นีโมนิค	ไอเปอร์แรนด์	จำนวนไบท์	สถานะแฟลก C	สถานะแฟลก AC	ความหมาย
อินพุท เอาท์พุท	ANL	PORT, #DATA	2			(PORT)←(PORT)∨DATA ทำการ AND ข้อมูลแบบอิมมีเดียทกับข้อมูลพอร์ท P1,P2 และบัส
	ANLD	EP,A	1			(EP)←(A)∧(EP) AND ข้อมูลพอร์ทส่วนขยาย P4,P5,P6 หรือ P7 กับแอคคิวมูเลเตอร์บิท 0-3
	IN	A,PN	1			(A)←(PN) อินพุทข้อมูลจากพอร์ท P_1 หรือ P_2 มายังแอคคิวมูเลเตอร์
	IN	A,DBB				(A)←(BUS) อินพุทจาก BUS เข้ามายังแอคคิวมูเลเตอร์จากส่วนข้อมูลในบัฟเฟอร์ส่วนบัส
	INS	A,BUS				(A)←(BUS) อินพุทจาก BUS เข้ามายังแอคคิวมูเลเตอร์ด้วยสโตรป
	MOVD	A,EP	1			(A03)←(EP) อินพุทข้อมูลจากพอร์ทที่ขยาย P4,P5,P6 หรือ P7 มายังแอคคิวมูเลเตอร์บิท 0-3
	MOVD	EP,A	1			(EP)←(A03) เอาท์พุทแอคคิวมูเลเตอร์บิท 0-3 ออกไปยังพอร์ทส่วนขยาย P_4,P_5,P_6 หรือ P_7
	ORL	PORT, #DATA	2			(PORT)←(PORT)∨DATA เป็นการ OR ข้อมูลอิมมีเดียทกับข้อมูลพอร์ท P_2,P_1 และบัส
	ORLD	EP,A				(EP)←(A03)∨(EP) OR แอคคิวมูเลเตอร์บิท 0-3 กับพอร์ทส่วนขยาย P_4,P_5,P_6 หรือ P_7
	OUT	DBB,A				(BUS)←(A) เอาท์พุทจากแอคคิวมูเลเตอร์ไปบัฟเฟอร์บัสข้อมูล
	OUTL	PORT,A				(PORT)←(A) เอาท์พุทแอคคิวมูเลเตอร์ไปยังพอร์ท P_1,P_2
คำสั่งอ้างถึงหน่วยความจำ	MOV	A,@R	1			(A)←[(R)] โหลดข้อมูลในหน่วยความจำส่วนข้อมูลที่แอดเดรสด้วย R_0 หรือ R_1 มายังแอคคิวมูเลเตอร์
	MOV	@R,A				[(R)]←(A) เก็บข้อมูลจากแอคคิวมูเลเตอร์ไปยังหน่วยความจำข้อมูลที่อ้างถึงด้วยแอดเดรสที่มาจาก R_0 หรือ R_1
	MOVP	A,@A				(A)←((PCH)(A)) โหลดข้อมูลจากหน่วยความจำโปรแกรมที่แอดเดรสด้วยข้อมูลในแอคคิวมูเลเตอร์กับบิทบนของ R
	MOVP3	A,@A				(A)←((3)(A)) โหลดข้อมูลจากหน่วยความจำที่โปรแกรมที่แอดเดรสด้วย 0011 xxxxxxxx เมื่อ xxxxxxxx เป็นข้อมูลในแอคคิวมูเลเตอร์

ชนิดคำสั่ง	นีโมนิค	ไอเปอร์แรนด์	จำนวนไบท์	สถานะแฟลก C	สถานะแฟลก AC	ความหมาย
	MOVX	A,@R	1			(A)←((R)) โหลดข้อมูลจากภายนอกจากหน่วยความจำข้อมูลภายนอกที่แอดเดรสด้วย R_0,R_1 เข้ามาเก็บยังแอคคิวมูเลเตอร์
	MOVX	@R,A	1			((R))←(A) เก็บข้อมูลจากแอคคิวมูเลเตอร์ไปยังหน่วยความจำข้อมูลภายนอกที่แอดเดรสด้วย R_0,R_1
	XCH	A,@R	1			(A)←((R)) แลกเปลี่ยนข้อมูลระหว่างแอคคิวมูเลเตอร์กับข้อมูลในหน่วยความจำข้อมูลภายใน CPU ที่แอดเดรสด้วย R_0,R_1
	XCHD	A,@R	1			(A03)←((R)03) แลกเปลี่ยนข้อมูลในแอคคิวมูเลเตอร์บิท 0-3 กับบิท 0-3 ของหน่วยความจำข้อมูลภายในที่แอดเดรส โดย R_0 หรือ R_1
กลุ่มคำสั่งที่อ้างหน่วยความจำโดยใช้ข้อมูลในหน่วยความจำเป็นไอเปอร์แรนด์	ADD		1	X	X	(A)←(A)+[(K)] บวกข้อมูลในซีพียูที่แอดเดรสด้วย R_0 หรือ R_1 กับแอคคิวมูเลเตอร์
	ADDC		1	X	X	(A)←(A)+[(R)]+(C) เหมือน ADD แต่บวกบิทหด้วย
	NAL		1			(A)←(A)∧((R)) AND กับหน่วยความจำที่แอดเดรสที่หน่วยความจำข้อมูลด้วย R_0,R_1
	ORL		1			(A)←(A)∨((R)) เหมือน ANL แต่เป็น OR
	INC		1			((R))←((R))+1 เพิ่มค่าหน่วยความจำข้อมูลภายในอีกหนึ่งด้วยแอดเดรสด้วยรีจิสเตอร์ R_0,R_1
	XRL		1			(A)←(A)∀((R)) เหมือน ANL แต่เป็น EX-OR
อิมมีเดียท	MOV		2			(REG)←DATA โหลดข้อมูลเข้าไปยังรีจิสเตอร์ R_0.....R_1
	MOV		2			((R))←DATA โหลดข้อมูลไปยังหน่วยความจำภายข้อมูลในซีพียูที่แอดเดรสด้วย R_0 หรือ R_1
กระโดด	JMP	ADDR	2			(PC10)←ADDR กระโดดไปยัง ADDR ภายในบล็อกเดียวกัน
	JMPP	@A	1			(PC)←(PCH)(A),(PCL)←(PCH)(A) โหลดข้อมูลจากหน่วยความจำมายังโปรแกรมเคาน์เตอร์ 8 บิทล่างโดยแอดเดรสโดยแอคคิวมูเลเตอร์กับ 4 บิทของโปรแกรมเคาน์เตอร์
	SEL	MBO	1			ทำให้บิทบนสุดของ PC = 0
	SEL	MBI	1			ทำให้บิทบนสุดของ PC = 1

ชนิดคำสั่ง	นีโมนิค	โอเปอร์แรนด์	จำนวนไบท์	สถานะแฟลก C	สถานะแฟลก AC	ความหมาย
โปรแกรมย่อย	CALL	ADDR	2			STACK←STATUS-(PC) (SP)←(SP) + 1 (PC)←ADDR เรียกโปรแกรมย่อย
	RET		1			(PC)←STACK, (SP)←(SP) -1 กลับจากโปรแกรมย่อย
	RETR		1	X	X	(PC)← STATUS STACK, (SP)< (SP) - 1
กระทำแบบอิมมีเดียท	ADD	A,#DATA	2	X	X	(A)←(A) + DATA
	ADDC	A,#DATA	2	X	X	(A)←(A) + DATA + (C)
	ANL	A,#DATA	2			(A)←(A)∧DATA
	ORL	A, #DATA	2			(A)←(A)∧DATA
	ORL	A# DATA	2			(A)←(A)▼DATA
คำสั่งกระโดดตามเงื่อนไข	OJNZ	RN, ADDR8	2			(RN)←(RN) -1 ถ้า ถ้า (RN) = 0 (PCL)←ADDR8 ทำการลดค่า R_0 !!!R_1 ลง 1 ทดสอบว่าเป็นศูนย์หรือไม่ถ้าผลลัพธ์ไม่ใช่เป็น "0" ก็ไม่มีการกระโดด
	JB	ADDR 8	2			(PCL)←ADDR 8 กระโดดไปที่ ADDR 8 ถ้าบิทที่ b เป็น "1"
	JC	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อ C = 1
	JFO	ADDR 8	2			(PCL)←ADDR 8 ทดสอบเมื่อบิทแฟลก F_0 = 1
	JFI	ADDR 8	2			(PCL)←ADDR 8 ทดสอบเมื่อแฟลก F_1 = 1
	JNC	ADDR 8	2			(PCL)←ADDR 8 ทดสอบเมื่อบิททด = 0
	JNI	ADDR 8	2			(PCL)←ADDR 8 ทดสอบเมื่ออินเตอร์รัพท์เป็น "0"
	JNIBF	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อ IBF แฟลกเป็น "0"
	INTO	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่ออินพุท T_0 เป็น "0"
	JN TI	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อ T_1 เป็น "0"
	JNZ	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อข้อมูลใน A เป็นศูนย์
	JOBF	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อ OBF เป็น "1"
	JTF	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อแฟลก timer เป็น "1"
	JTO	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อ T_0 = "1"
	JTI	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อ T_1 = "1"
	SZ	ADDR 8	2			(PCL)←ADDR 8 กระโดดเมื่อข้อมูลใน A เป็นศูนย์

ชนิดคำสั่ง	นีโมนิค	โอเปอร์แรนด์	จำนวนไบท์	สถานะแฟลก C	สถานะแฟลก AC	ความหมาย
กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์	MOV	A, RN	1			(A)←(RN) (RN) คือรีจิสเตอร์ใด ๆ
	MOV	RN, A	1			(RN)←(A) เคลื่อนย้ายข้อมูลจากแอคคิวมูเลเตอร์มายังรีจิสเตอร์
	XCH	A, RN	1			(A)←(RN) แลกเปลี่ยนข้อมูล
การกระทำระหว่างข้อมูล	ADD	A, RN	1	X	X	(A)←(A) + (RN) เป็นการบวกระหว่างแอคคิวมูเลเตอร์กับรีจิสเตอร์ผลลัพธ์เก็บในแอคคิวมูเลเตอร์
	ADDC	A, RN	1	X	X	(A)←(A) + (RN) + (C) เป็นการบวกเมื่อคิดบิททดด้วย
	ANL	A, RN	1			(A)←(A)∧(RN) เป็นการ AND ทางลอจิก
	ORL	A, RN	1			(A)←(A) V (RN) เป็นการ OR ทางลอจิก
	XRL	A, RN	1			(A)←(A) EXOR (RN) เป็นการ exor ทางลอจิก
ทำงานภายใน	CLR	A	1			(A)←0 เคลียร์แอคคอาบูเลเตอร์
	CLL	A	1			(A)←(A) คอมพลีเมนท์แอคคิวมูเลเตอร์
	DAA		1			
	DEC	REG	1			(REG)←(REG) -1 ลดค่ารีจิสเตอร์ 1
	INC	REG				(REG)←(REG) + 1 เพิ่มค่าในรีจิสเตอร์อีก 1
	RL	A				หมุนแอคคิวมูเลเตอร์ทางซ้าย 
	RLC	A	1	X		หมุนแอคคอวมูเลเตอร์ผ่านบิททดทางซ้าย 
	RR	A	1			หมุนแอคคิวมูเลเตอร์ทางขวา 
	RRC	A	1	X		หมุนแอคคิวมูเลเตอร์ทางขวาผ่านบิททด 
	SEL	RBO	1			เลือกรีจิสเตอร์แบงค์ "0"
	SEL	RB1	1			เลือกรีจิสเตอร์แบงค์ 1
	SWAP	A	1			ย้ายข้อมูลใน A สี่บิทบนกับสี่บิทล่างสลับที่กัน 
	DIS	TCNTI	1			ดิสเอเบิ้ลการอินเตอร์รัพท์ของวงจรตั้งเวลา
	EN	TCNTI	1			อีนาเบิ้ลวงจรตั้งเวลา
	DIS	I	1			ดิสเอเบิ้ลอินเตอร์รัพท์จากภายนอก

รหัสคำสั่ง	นีโมนิค	โอเปอร์แรนด์	จำนวนไบท์	สถานะแฟลก		ความหมาย
				C	AC	
	EN	I	1			อินาเบิ้ลอินเตอร์รัพท์
	ENTO	CLK	1			อินาเบิ้ลการตั้งเวลาเอาท์พุท
	MOV	A, T	1			A ← T
						อ่านจากรีจิสเตอร์วงจรตั้งเวลา มายังแอคคิวมูเลเตอร์
	MOV	T, A	1			T ← A
						โหลดค่าจากแอคคิวมูเลเตอร์ไปยังวงจรตั้งเวลา
	STOP	A, PSW	1			หยุดการวงจรตั้งเวลา
	STRT	CNT	1			เริ่มให้วงจรนับทำงาน
	STRT	T	1			เริ่มวงจรตั้งเวลาใหม่
	CLR	S	1	0		เคลียร์ PSW
	CPL	S	1	X		คอมพลีเมนท์
	MOV	A, PSW	1	X		เคลื่อนย้ายข้อมูล PSW มายังแอคคิวมูเลเตอร์
	MOV	PSW, A	1	X		เคลื่อนย้ายข้อมูลจากแอคคิวมูเลเตอร์มายัง PSW
	NOP		1			รอเวลาไปหนึ่งไซเคิลคำสั่ง

บทที่ 3

หลักการออกแบบและการสร้าง

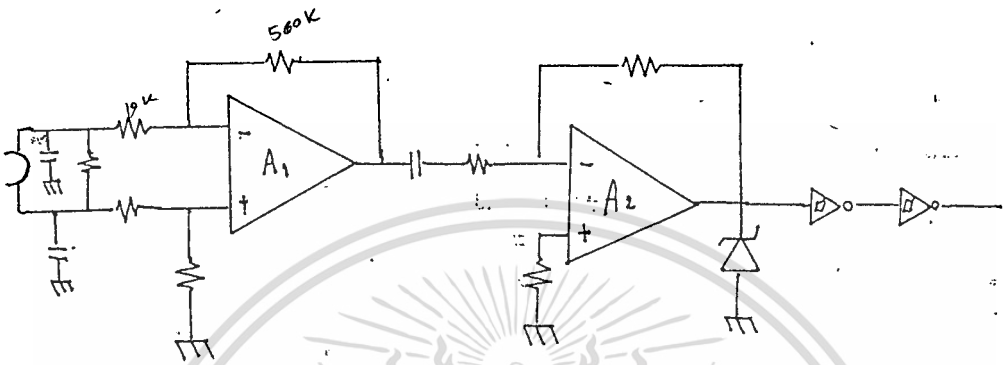
3.1 การสร้างแมคแคนนิค (Machanice) ส่วนของหัวอ่าน

ในส่วนของหัวอ่านเราต้องสามารถปรับให้เอียงซ้าย ขวา ขึ้นบน และลงล่างได้ เพื่อจะได้เป็นส่วนที่ใช้ในการปรับหัวอ่านให้สามารถอ่านสัญญาณได้แรงที่สุดและตรง Track ที่สุด ซึ่งมันเป็นส่วนที่ยากมากในการออกแบบ ในกรณีรูดบัตร เราจำเป็นที่จะต้องให้หัวเทปแนบบัตร เพื่อที่จะทำให้หัวเทปสามารถอ่านข้อมูลจากแถบแม่เหล็กออกมาได้ ถ้าเราให้หัวเทปกดลงไปในแถบแม่เหล็กมาก ก็จะทำให้แถบแม่เหล็กกดลงไปมากทำให้แถบแม่เหล็กสึกในที่สุด เมื่อรูดได้ไม่กี่ครั้งก็จะทำให้แถบแม่เหล็กไม่สามารถใช้ได้อีกต่อไป และถ้าเรากดเบามากก็จะทำให้สัญญาณที่ได้ออกมาอ่อนหรือในบางครั้งอ่านได้บ้าง ไม่ได้บ้าง ซึ่งในขั้นแรกเราต้องทำส่วนนี้ให้ได้ก่อน

ส่วนร่องสำหรับให้บัตรผ่านเข้าไปต้องมีขนาดแคบพอดีที่บัตรสามารถเคลื่อนที่ผ่านโดยที่จะไม่ทำให้บัตรเสียหาย คือถ้าแคบเกินไปจะทำให้บัตรเคลื่อนผ่านได้ยาก และพลาสติกที่ใช้ทำร่องจะไปครูดกับบัตร ซึ่งจะทำให้บัตรเสียหายได้ ถ้าหากเรากว้างเกินไปก็จะทำให้บัตรเคลื่อนที่ผ่านหัวเทปแบบไม่สม่ำเสมอ ก็จะทำให้ข้อมูลที่อ่านได้นั้นผิดพลาด ซึ่งในการทำส่วนนี้เราจะต้องออกแบบให้ดี ซึ่งจะเห็นว่า เราไม่มีสูตรในการคำนวณ ดังนั้นต้องอาศัยการทดลอง เมื่อได้ส่วนสร้างส่วนนี้เสร็จสิ้น เราก็จะต้องทำการทดลองนำบัตรมารูดดู และใช้ออสซิลโลสโคป (Oscilloscope) จับดูสัญญาณที่ผ่านหัวเทปมา ซึ่งจากการทดลองจะได้ว่าสัญญาณดีแล้วหรือไม่ ถ้าดีแสดงว่าร่องรูดบัตรแคบพอใช้ได้แล้ว แต่ถ้ายังไม่ดีก็ต้องปรับแต่งกันใหม่อีก การปรับแต่งนั้นคือต้องปรับแต่งหัวเทปพร้อมกับร่องรูดบัตรที่จะได้ให้ ได้สัญญาณดีที่สุด ผลจากการรบกวนน้อยลง และสัญญาณที่ได้ออกมาจากหัวเทปจะต้องม่ C ขนาด 1 MF ต่อกราวด์ทั้ง 2 Track ที่ใช้เพื่อเป็นการลด Noise ให้น้อยลงไปอีก เพื่อส่งไปยังภาค Sense Amplifier ต่อไป

3.1.2. Sense Amplifier

ส่วนของ Sense Amplifier ส่วนนี้จะเป็นส่วนที่ใช้ขยายสัญญาณต่อจากหัวเทปมา ดังรูป

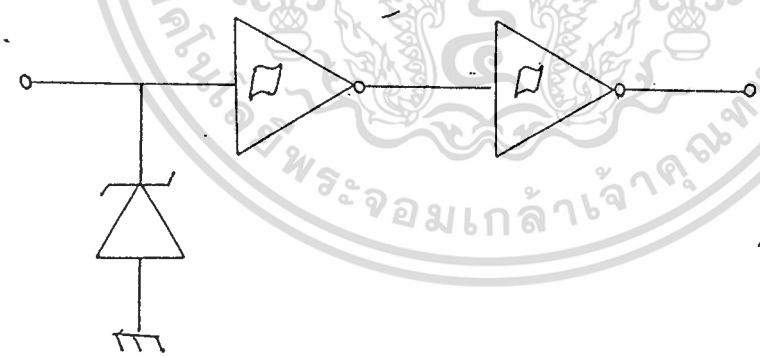


รูป SENSE AMPLIFIER

ดูจากรูป เราจะเห็นได้ว่า ส่วนแรกของวงจรคือจะเป็น Op-Amp ที่ต่อแบบ Differential Amp โดยมี Gain การขยาย = 560K / 10K จะได้เท่ากับ 56 เท่า เพื่อทำให้ S/N มีค่าสูง ต่อจากนั้นก็นำมาต่อเข้ากับ Op-Amp ตัวที่ 2 ที่ทำหน้าที่เป็น Rectifier แล้วส่งต่อไปยัง Schmittrigger เพื่อทำให้เป็นสัญญาณ Pluse ทั้ง 2 ขา คือ ขา VCC และขา VSS ในการต่อ C นี้ เราจะต้องต่อให้ใกล้กับตัย ออป-แอมป์ ให้มากที่สุดจึงเกิด Noise ได้มาก หลังจากการต่อนั้นเสร็จสิ้น ยังเห็นว่ามี Noise อีก ซึ่งในขณะนั้นเราก็ไม่ทราบสาเหตุว่ามาจากไหน จนในที่สุดก็ทำการเปลี่ยน Op-Amp เบอร์ต่าง ๆ ก็ไม่สามารถแก้ Noise นี้ได้ จนกระทั่งในที่สุด ก็ทดลองเปลี่ยนไอซี (Integrate Circiut) ที่มีอัตราการขยายดี ๆ และมี Noise ในตัวมันเองน้อย ถ้าพูดง่าย ๆ ก็คือ การทำงานจะต้องใช้ไอซีเกรดดี ไม่นั้นแล้วจะไม่สามารถแก้ปัญหา Noise ตัวนี้ลงได้ ซึ่งเป็นวิธีลด Noise จุดสุดท้าย จากนั้นการจะได้สัญญาณ C ขนาด 100 MF ผ่านไปยังวงจรแอมปิไฟเออร์ เพื่อทำการขยายสัญญาณให้แรงขึ้นไปอีก

3.1.3. วงจร ANALOG tO DIGITAL

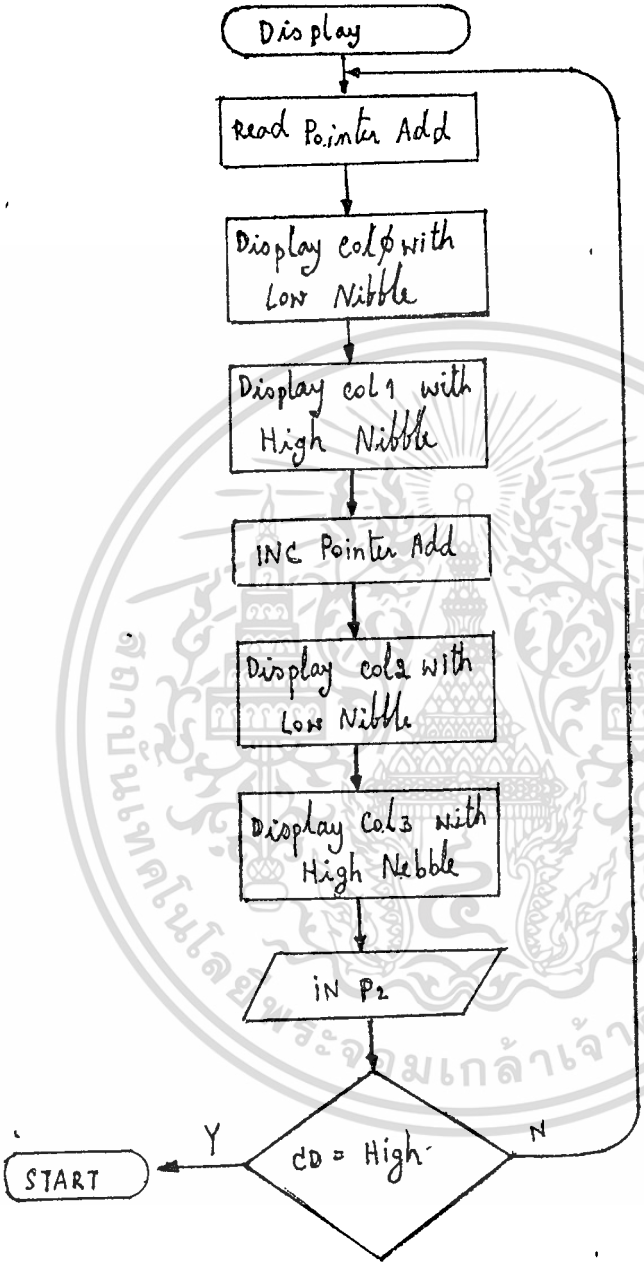
เนื่องจากสัญญาณที่ได้มาจากภาค Sense Amplifier เป็นสัญญาณแบบอนาล็อกและมีค่าแรงดันสูงมาก เนื่องจากเราใช้วงจร Amplifier มี Gain สูงมาก ถ้าเรานำสัญญาณนี้ไปเข้ายัง CPU ก็จะทำให้เกิดการผิดพลาดเกิดขึ้นมา ดังนั้น เราจึงต้องทำการทำให้เป็นสัญญาณดิจิตอล เนื่องจากสัญญาณที่ได้ มาจาก Sense Amplifier ก็เป็นสัญญาณที่เกือบเป็นดิจิตอลแล้ว ดังนั้นเราจึงต้องใช้ ชมิท ทริกเกอร์ (Schmitt trigger) เบอร์ 74LS14 เมื่อทำการจัด Wave Form ให้สวยงาม แต่เนื่องจากว่าสัญญาณที่ผ่านวงจร ชมิท ทริกเกอร์(Schmitt trigger) มีค่าแรงดันที่ออกมามีค่ามากและมีความสูงต่ำ (Vp-p) ไม่เท่ากัน ถ้าเรานำไปเข้า CPU จะทำ CPU ไม่สามารถรับสัญญาณนี้ได้ ดังนั้น จึงทำการปรับระดับของสัญญาณให้มีขนาด 0-5 Volt เมื่อทำจะนำไปเข้า CPU ต่อไป ดังนั้นจึงต่อซีเนอร์ไดโอดเข้าไป เพื่อให้ได้ Wave Form ออกมามีลักษณะสวยงามดังรูป

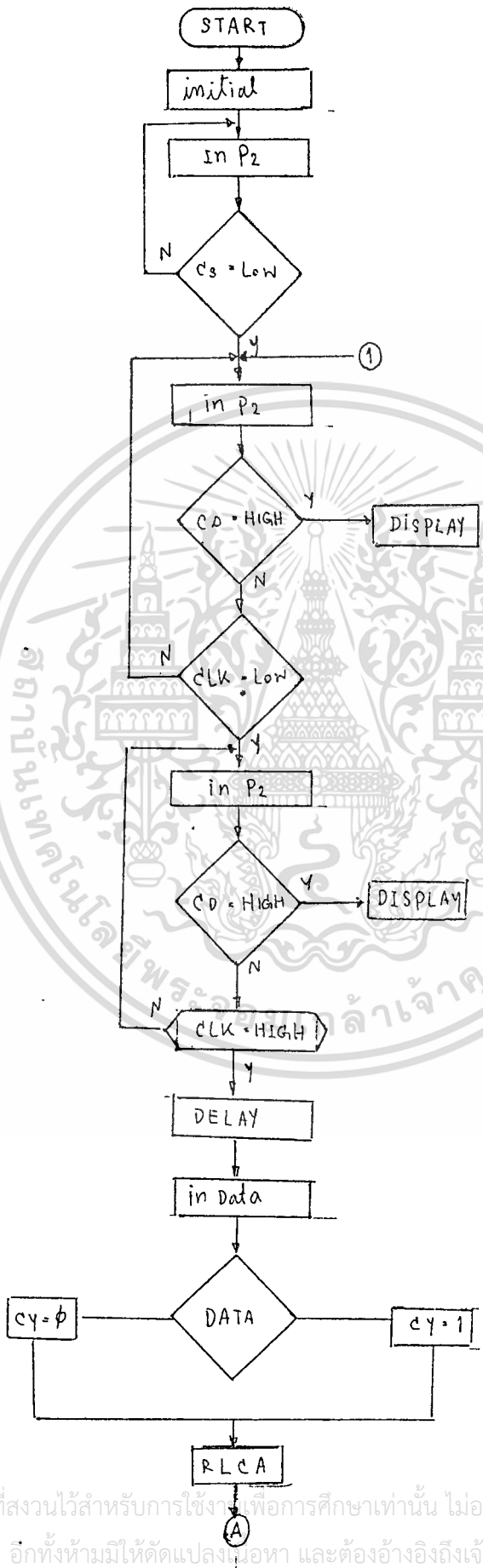


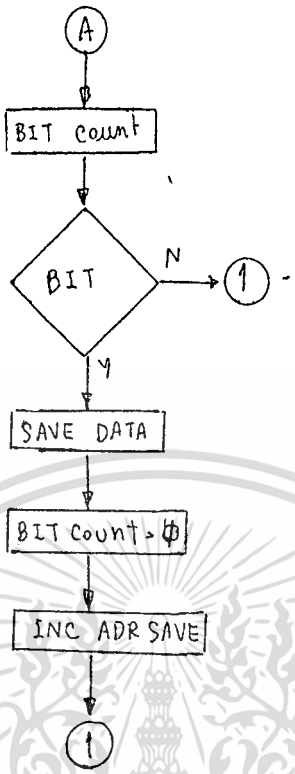
3.1.4. CPU

ซึ่งนับว่าเป็นหัวใจอันสำคัญ ซึ่งในการทำงานทั้งระบบ (System) ส่วนใหญ่จะขึ้นอยู่กับ CPU ตัวนี้ ซึ่งจะเป็นวิธีที่ประหยัดค่าใช้จ่ายทางด้านฮาร์ดแวร์มากซึ่งเราจะใช้โปรแกรมเป็นตัวควบคุม ซึ่งในการทดลองเราจำเป็นที่จะต้องสร้างส่วนนี้ขึ้นมาก่อน ซึ่งการสร้างส่วนนี้ในครั้งแรกเราก็ต้องสร้างวงจรที่ใช้ในการทดลอง เพื่อให้ได้การทำงานที่ออกมาจริง ๆ ซึ่งวงจรส่วนนี้ไม่สามารถนำไปใช้งานได้จริง เพราะดาต้าจริง ๆ นั้นมามาจากแมมโมรี่ภายนอก (External Memory) ซึ่งเราซึ่งเขียนเข้าไปโปรแกรมซึ่งอัดเข้าไปแมมโมรี่ภายนอก (External Memory) นั้นก็คือแรมแพค (RAM PACK) ที่เราสร้างขึ้นมา ซึ่งจากการทดลองก็ได้ลองต่อวงจรดังรูป

ซึ่งในครั้งแรกวงจรนี้จะสร้างใน โฟโต้บอร์ด (Photo Board) ซึ่งในการใช้โฟโต้บอร์ดนี้สิ่งสำคัญคือ จะต้องต่อขาทุกขาให้แน่นที่สุด และสายต้องพยายามสั้นและแนบติดกับบอร์ดให้มากที่สุดไม่นั้นแล้วจะทำให้ผลการทดลองผิดพลาดได้ เมื่อนำอุปกรณ์ลงบอร์ดเสร็จ เราจำเป็นต้องเขียนโปรแกรมเข้าไปเพื่อที่จะทำการทดสอบว่า วงจรนี้สามารถทำงานได้หรือไม่ การแสดงผล (Display) เราให้หลอดสแกน (Scan) ที่ละหลอด โดยใช้โปรแกรม ซึ่งในการแสดงผลที่สำคัญในส่วนแสดงผล (Seven Segment) เราจะต้องนำ R ไป Drop ถ้าไม่นั้นจะทำให้หลอดขาด ซึ่งจากการทดลองหลอด Seven Segment ก็ขาดไปหลายหลอด ส่วนโปรแกรมที่ใช้ในการทดลองสามารถเขียนได้หลายแบบ แต่ที่ใช้ทดลองได้ให้โปรแกรมดังโฟลชาร์ทนี้







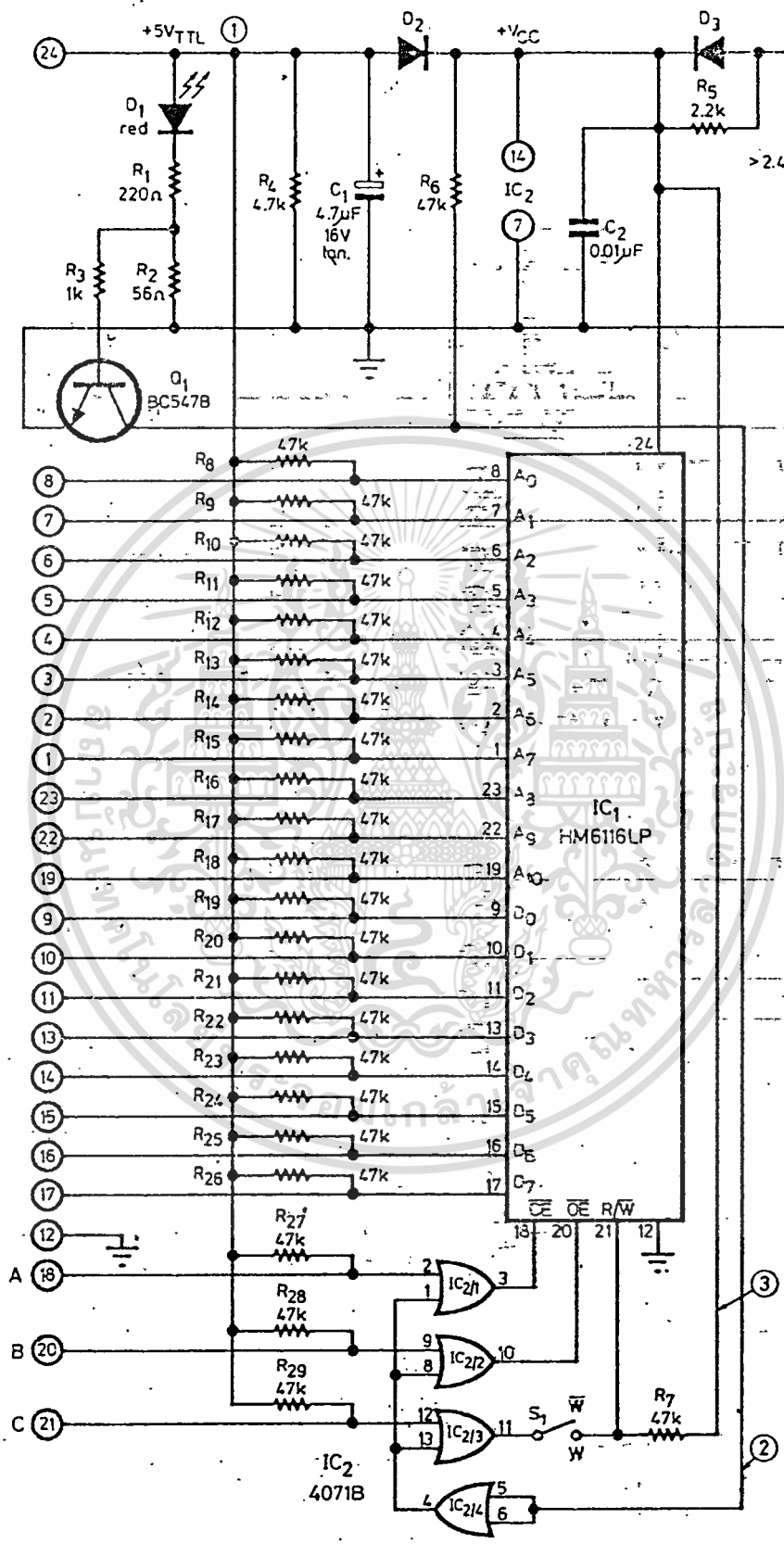
3.2 การทำงานของวงจร

CPU & DISPLAY

จากรูป เป็นวงจรที่ใช้งานจริง ซึ่งเราจะเห็นว่าใช้ พอร์ท 1 เป็นพอร์ทแสดงผล (Output Port) ซึ่งใช้ 4 บิทบนเป็นตัวแสดงผล โดยค่าที่อ่านได้จากบัตร โดยสัญญาณที่ส่งออกมานี้เป็นเลข BCD และใช้ไอซีเบอร์ 7448 เป็นตัวเปลี่ยน BCD โค้ดไปเป็น Seven Segment แล้วผ่านไอซีเบอร์ 244 เพื่อเป็นตัวรับสัญญาณให้แรงขึ้น แล้วจ่ายให้แก่หลอด Seven Segment ส่วน 4 บิทล่าง เป็นสัญญาณที่กำหนดขึ้นจากโปรแกรมเมื่อส่งออกมา เพื่อควบคุมหลอดว่าจะให้หลอดไหนติด ซึ่งอยู่ที่คำสั่ง เนื่องการแสดงผลของพอร์ท Seven Segment เราใช้ SCAN คือให้ติดทีละหลอดแต่เร็วมากจึงทำให้เราเห็นว่า หลอดทั้ง 4 ติดตลอด

ส่วนพอร์ท 2 เราใช้เป็นอินพุทพอร์ท (Input Port) โดยเราจะใช้เพียง 3 เส้น คือ ขา P25-P27 หรือขา 36-37 ของ CPU นั่นเอง โดยให้ขา P27 เป็นขาสัญญาณเพื่อเช็คว่ามีบัตรเข้ามาหรือยัง และขา P26 เป็นขาสัญญาณที่แทนสัญญาณ Clock Reference ซึ่งได้จาก Track 1 ส่วนขา P25 เป็นขาแทนสัญญาณดาต้าจากบัตรที่อ่านได้จาก แถบแม่เหล็ก ซึ่งการทำงานต่าง ๆ จะถูกควบคุมโปรแกรมที่เขียนขึ้นว่าจะให้ทำอะไร ซึ่งการใช้งานจริง ๆ เราจะต้องเขียนโปรแกรมให้ทำงานตามที่เราต้องการ เช่น ควบคุมสวิทช์ แต่ในที่นี้จะเขียนโปรแกรมให้เห็นเพียงตัวเลขเท่านั้น ซึ่งการใช้งานจริง ๆ จะนำสัญญาณที่ได้จากการอ่านมาเปรียบเทียบกับข้อมูลแล้ว ให้มันแสดงผลตามต้องการ ซึ่ง โฟลชาร์ท (Flow Chart) นี้จะเป็นตัวแสดงผลของบัตรเท่านั้น

จากรูปจะเห็นว่าวงจรที่เราใช้งานนั้นเราต้องสร้างชุดหลังให้เสร็จสิ้นเสียก่อน โดยจากรูปจะนำขา EA (External Access) ต่อเป็น High ดังนั้น CPU ก็จะใช้โปรแกรมจากภายนอก คือ จากแรมแพค (RAM PACK) ซึ่งภายใน RAM PACK โดยจาก D B0 - D B7 สามารถใช้เป็นทั้งบัสข้อมูล (Data Bus) และแอดเดรสบัส (Address Bus) ดังนั้น จึงต้องต่อไอซีเบอร์ 74L3373 ซึ่งไอซีเบอร์นี้จะเป็นตัวแลทช์ (Latch) ข้อมูลเอาไว้โดยในตัว CPU เมื่อต้องการส่ง Address ออกมาจากตัว CPU มันจะทำการส่งสัญญาณ ALE ออกมาเพื่อมาทำการอีนาเบิ้ลแอดเดรสแลทช์ (Enable Address Latch) มายังขา G ของไอซีเบอร์ 373 ไอซีเบอร์ LS373 จะแลทช์แอดเดรสไว้ โดยเป็นตัวกำหนดแอดเดรสให้แก่แรมแพค (RAM PACK) จากนั้น ข้อมูลในแรมแพคก็จะส่งออกมา CPU ก็จะส่งสัญญาณ PSEN ออกมาเพื่อเป็นควบคุมการอ่านข้อมูลจากดาต้าบัส (Data Bus) จากนั้นข้อมูลก็จะเข้าไปใน CPU ส่วนการไรท์ (Write) CPU ก็จะส่งสัญญาณ WR ออกมา จากนั้น CPU จะส่งข้อมูล โดยข้อมูลจะนำไปเก็บใน RAM PACK แต่แอดเดรสของแรมแพคมีค่า 2K ดังนั้น สัญญาณ A8 - A10 ก็จะถูกส่งมาทาง P20 - P23 แต่การอัดข้อมูลเข้าไปในแรมแพค (RAM PACK) เราจะอัดโดยเครื่องไมโครคอมพิวเตอร์ เช่น เครื่องไมโครคอมพิวเตอร์ Z-80 หรืออาจใช้เครื่องไมโครคอมพิวเตอร์แอปเปิล II (CPU เบอร์ 6502) ฯลฯ เป็นตัวอัดข้อมูลเข้าไปในแรมแพคก่อน จากนั้นจึงนำแรมแพคมาต่อลงในวงจรดังรูป เมื่อทำการทดลองว่าโปรแกรมทำงานได้จริงหรือไม่ ซึ่งจะช่วยให้เราไม่ต้องลบข้อมูลซึ่งอัดลงไปใน EPROM ในขณะที่เราต้องการแก้ไขโปรแกรม ซึ่งเสียเวลามาก ดังนั้นถ้าเราใช้แรมแพคจะสามารถแก้ไขโปรแกรมได้โดยไม่ยากเลย หลังจากที่เราได้โปรแกรมที่เราต้องการซึ่งอยู่ในแรมแพคนั้นแล้ว จากนั้นเราค่อยนำไปอัดลงใน EPROM ของไมโครคอมพิวเตอร์ เบอร์ 8748 ซึ่งมีอยู่ภายในตัวของมันซึ่งมีขนาด 1024 ไบท์ เมื่อเราอัดข้อมูลเข้าไปใน CPU เบอร์ 8748 เสร็จแล้ว เราก็จะถอดแรมแพค จากนั้นจะต้องนำขา EA ขา 7 ของ CPU ต่อลงกราวด์ เพื่อเป็นการบอกให้ CPU รู้ว่าจะต้องใช้ข้อมูล ซึ่งอยู่ภายในตัวมันเอง จากนั้นเมื่อเราถอดแรมแพคออกก็จะทำให้เราสามารถทำให้เครื่องคาร์ดดีดเดอร์เล็กลง ซึ่งเป็นความต้องการของผู้สร้าง ดังนั้นจะเป็นว่าผู้สร้างจะเน้นด้านประหยัด และต้องการทำให้เล็ก



รูปที่ 2 วงจร ROM จาก RAM

หลักการทำงาน

การใช้ RAM เบอร์ 6116LP ซึ่งเป็นของบริษัท ฮิตาชิ ที่หาซื้อได้ในเมืองไทยเรา ลักษณะโครงสร้างของ RAM ตัวนี้ ให้ข้อดีที่ใช้กระแสค่อนข้างต่ำเมื่ออยู่ในสภาวะ เก็บข้อมูล สามารถเก็บไว้ใน RAM ทั่ว ๆ ไป ดังนั้นการเขียนโปรแกรมข้อมูลเข้าไปเก็บไว้ใน RAM นี้ จึงทำเหมือนกับการเขียนโปรแกรมลงใน RAM จากบอร์ดไมโครทั่ว ๆ ไป และเมื่อโปรแกรมข้อมูลเข้าไปเก็บไว้ใน RAM หมดแล้ว และเราจะใช้ถ่านไฟฉายก้อนเล็กใส่ไว้ประมาณ 3 ก้อน หรือ 4.5 โวลท์ ก็สามารถเก็บรักษาโปรแกรมไว้ให้ได้ เหมือนกับเป็น ROM และใช้ได้เป็นเวลาหลาย ๆ วันกว่าถ่านไฟฉายจะหมด แต่เมื่ออยู่บนบอร์ดไมโครคอมพิวเตอร์ RAM โมดูลนี้ก็จะใช้ไฟเพียง 5 โวลท์ ที่จ่ายให้โดยตรงจากแหล่งจ่ายไฟ คงใช้ไฟจากถ่านไฟฉาย ก็ต่อเมื่อ เราดับไฟ 5 โวลท์นั้นแล้ว หรือเมื่อได้ถอด RAM ออกจากโมดูลนี้ไปใช้งานด้านอื่น ๆลๆ

เมื่อเราต่อเข้ากับไฟ + 5 โวลท์ แหล่งจ่ายไฟ ขา 24 ของโมดูล LED D1 ก็จะสว่าง บอกให้รู้ว่าวงจรกำลังใช้ไฟเลี้ยงจากแหล่งจ่ายไฟภายนอก R1 และ R2 จะเป็นตัวแบ่งแรงดันทำให้ทรานซิสเตอร์ Q1 นำกระแส เมื่อมีแหล่งฉายไฟ Q1 นี้ทำให้อินพุท ของ IC 2/4 มีค่าลอจิก '0' เป็นการสร้างลอจิก เพื่อให้สัญญาณ จากของไอซีผ่านเข้าไปยังขา CE OE และ R/W ได้ การต่อเพิ่มสวิทช์ W-W ก็เพื่อป้องกัน การเขียนข้อมูลใน RAM ในโอกาสที่ใช้โมดูลนี้แบบ ROM ข้อมูลที่อยู่ภายในจึงไม่มีการผิดพลาดจากการเขียนลงไปได้ แต่ถ้าต้องการโปรแกรมหรือเขียนข้อมูลลงใน RAM ถ้าทำได้ด้วยการเขียนข้อมูลลงใน RAM ตามขบวนการของคอมพิวเตอร์นั่นเอง

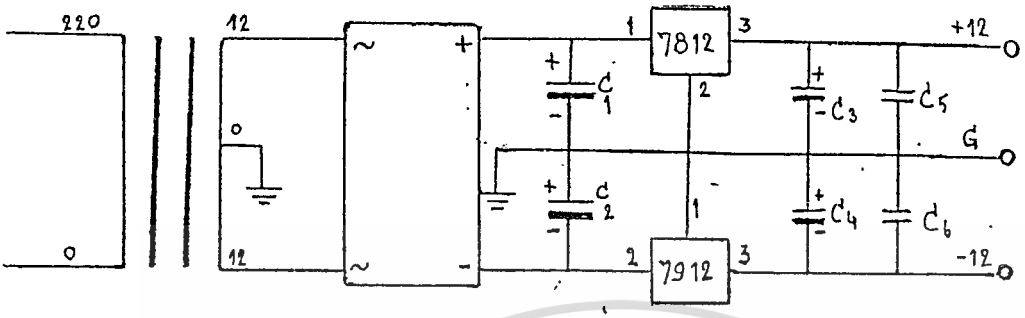
```
0002    23  77      LOOP    MOV  A,#77   ;กำหนดค่าที่จะแสดงผล
0004    39                  OUTL P1,A    ;ออกไปแสดงผล
0005    23  00              MOV  A,00    ;กำหนดค่าให้หลอดที่ 1 ติด
0007    3A                  OUT P2,A ;ออก Port ไปควบคุมหลอดที่ 1
0008    F4  F0              CALL DELAY   ;หน่วงเวลา
000A    23  40              MOV  A,40 ;กำหนดค่าให้หลอดที่ 2 ติด
000C    3A                  OUT  P2,A ;ออกพอร์ทไปควบคุมหลอดที่
000D    F4  F0              CALL DELAY     ; หน่วยเวลา
000F    23  80              MOV  A,80
0011    3A                  OUT  P2,A
0012    F4  F0              CALL DELAY
0014    23  C0              MOV  A,C0
0016    3A                  OUT  P2,A
0017    F4  F0              CALL DELAY
0019    04  02              JMP  LOOP
```

โปรแกรมหน่วงเวลา เพื่อให้เห็นผลการทำงาน

```
07F0    BB  F0      DELAY   : MOV  R3 # F0
07F2    BC  00      K2      : MOV  R4 # 0
07F4    EC  F4      B1      : DJNZ R4,B1
                              DJNZ R3,K2
                              RET
```

ซึ่งในวงจรนี้เป็นวงจรที่ใช้ในการทดลองครั้งแรก และต่อมาได้ปรับปรุงจนเหลือวงจรในการใช้งานจริงในรูปดังกล่าวมาแล้วในบทก่อน

POWER SUPPLY



POWER SUPPLY สำหรับ SENSE AMPLIFIER

เนื่องจากคุณสมบัติวงจรห้วงจร Sense Amp นี้จะต้องใช้แรงดันที่คงที่ไม่เปลี่ยนแปลงตาม Load จากการที่ได้อธิบายไปแล้วข้างต้นที่พูดถึงการใช้ Op-Amp เพราะฉนั้น เราจึงจะใช้ IC เรกูเลทเพื่อให้แรงดัน O/P คงที่

หลักการทำงานของวงจรเรกูเลท

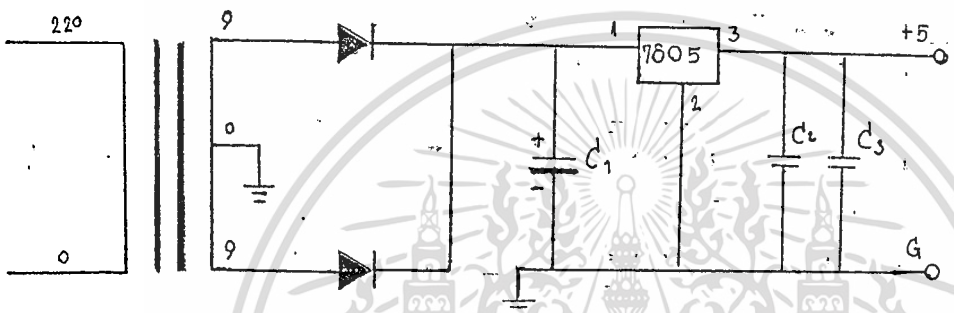
ด้วยคุณสมบัติของไอซีเรกูเลทเตอร์ 3 ขา เบอร์ LM7812 กับ LM7912 ซึ่งถูกออกแบบมาให้ค่าแรงดันเอาท์พุท (Output) จ่ายกระแสได้สูง นำมาประกอบกับอุปกรณ์เพียงไม่กี่ตัวก็สามารถนำมาใช้เป็นเครื่องจ่ายไฟแบบแรงดันคู่ได้ดี

จากรูปจะเห็นได้ว่า IC เบอร์ LM7812 นี้จะทำหน้าเป็นไฟบวก คือให้แรงดันที่เอาท์พุทออกมาเป็นบวก ส่วนIC เบอร์ LM7912 จะทำหน้าที่เป็นไฟลบ คือให้แรงดันที่เอาท์พุทออกมาเป็นลบ เนื่องจาก IC ทั้ง 2 เบอร์นี้จะให้แรงดันที่เอาท์พุทคงที่ไม่มีการเหลี่ยนแปลงตามโหลด Load แล้ว ยังมีการกำจัดความร้อนบางอยู่ภายในตัว IC เอง IC เบอร์LM7812 กับ 7912 จะแตกต่างเรื่องขากันอยู่ที่ขา 1 กับขา 3 คือ

7812 จะมีขา 1 - INPUT
2 - GROUND
3 - OUTPUT

ส่วน 7912 จะมีขา 1 - GROUND
2 - INPUT
3 - OUTPUT

จากวงจรเราจะเห็นได้ว่านอกจาก IC ทั้ง 2 เบอร์นี้แล้ว ยังประกอบไปด้วย Capacitor ค่าต่าง ๆ คือ C1 และ C2 จะทำหน้าที่กรองกระแสให้เรียบและแรงดันให้เรียบ ส่วน C3, C4, C5 และ C6 จะทำหน้าที่เป็นตัวป้องกันการเกิดสัญญาณรบกวนที่เข้ามา และเนื่องจากไดโอดด้วย เพื่อไม่ให้ผ่านไปยังภาคอื่น



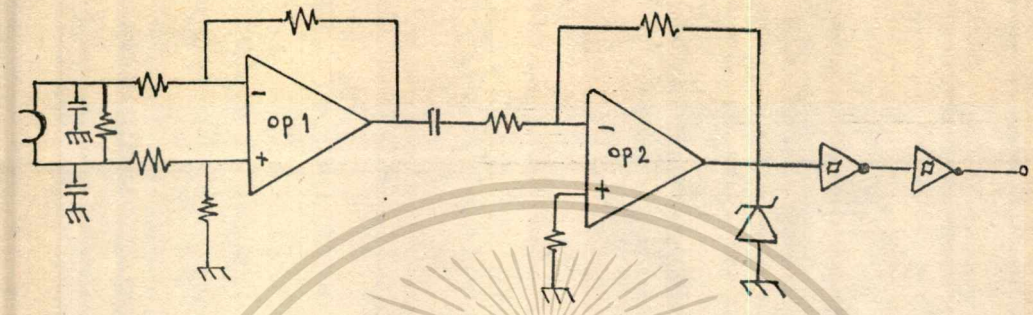
POWER SUPPLY สำหรับ CPU

เนื่องจากไมโครโปรเซสเซอร์จะต้องใช้แรงไฟที่คงที่ ถ้าไม่คงที่ก็จะทำให้เกิดปัญหาขึ้นในภายหลัง เพราะฉนั้นเราจึงใช้วงจร IC เรกูเลเตอร์ที่ใช้เบอร์ 7805 เพื่อให้แรงดันไฟที่เอาท์พุทมีค่าเป็นบวก 5 volt ที่จะนำไปเลี้ยงในวงจรไมโคร จากวงจรจะเห็นได้ว่ามีวงจรคล้ายคลึงกับวงจรแรกแต่แตกต่างกันตรงที่วงจรแรกมีไฟลบด้วย และ IC เบอร์นี้ก็จะมีขาคล้ายกับเบอร์ 7812 เพียงแต่ระดับแรงดันต่างกันตรงที่วงจรแรกมีแรงดันไฟเป็น +12 V ส่วนวงจรนี้จะมีระดับแรงดันไฟเพียง +5 V เท่านั้น

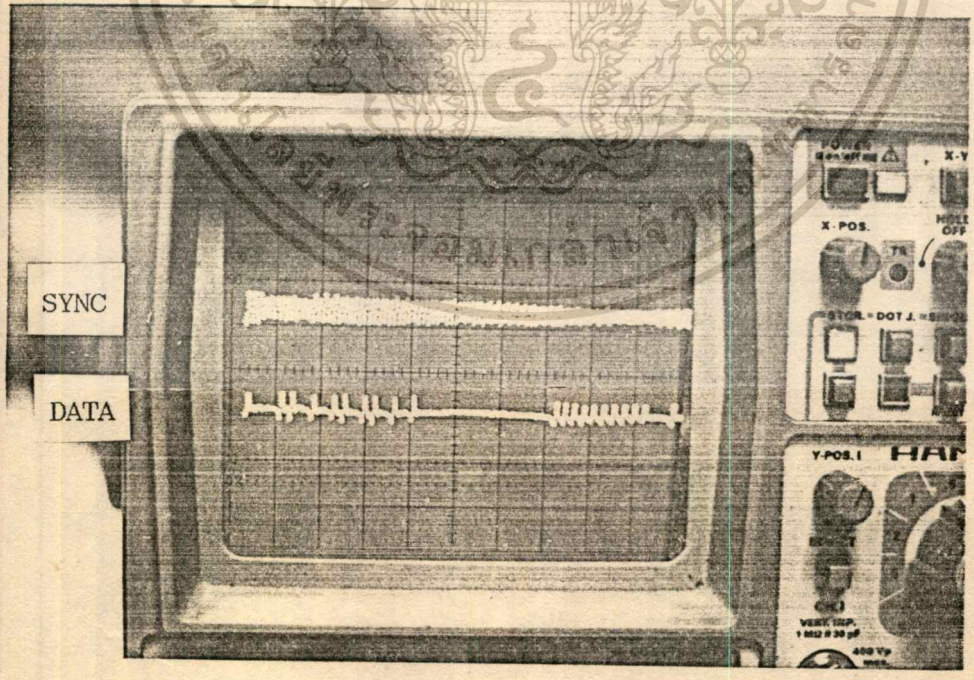
การสร้าง CPU & DISPLAY

การทดลองและผลการทดลอง

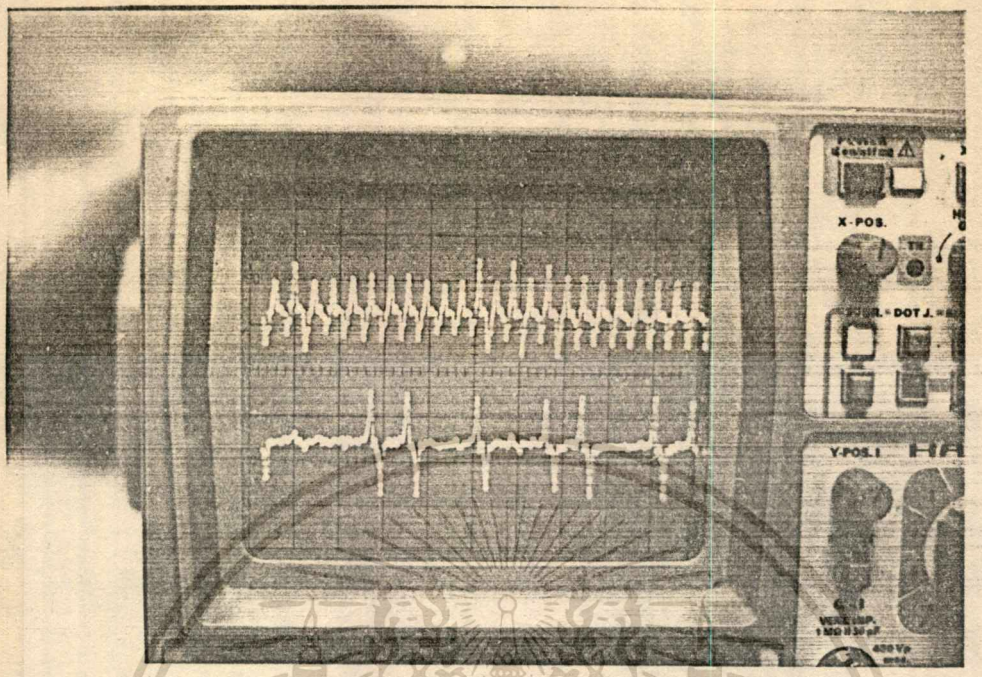
4. แสดงภาครับสัญญาณ



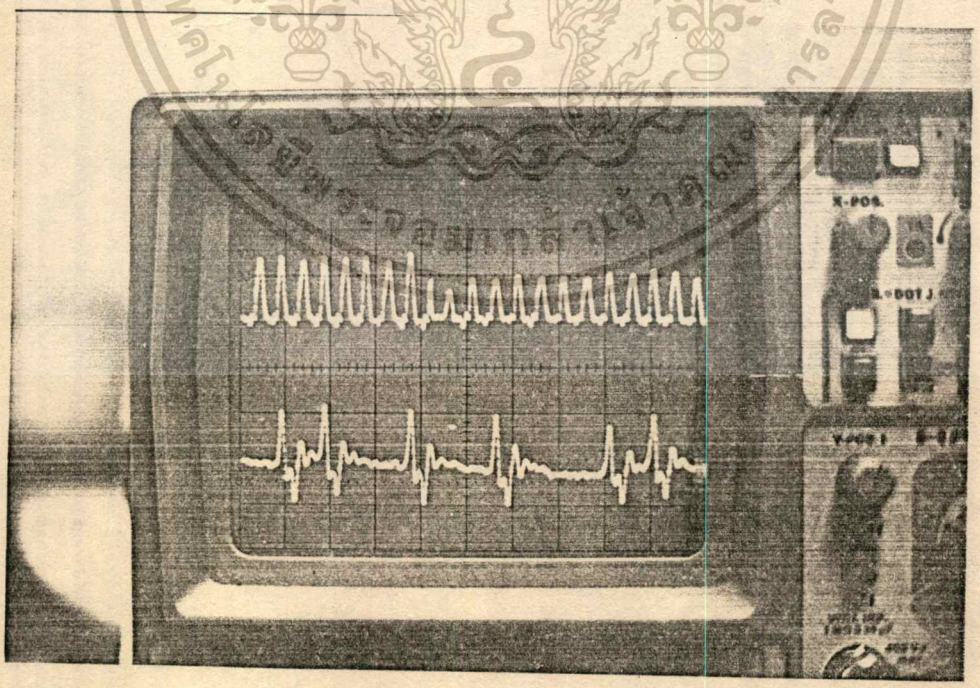
4.1 การแสดงสัญญาณ SYNC กับ DATA ที่จุดต่าง ๆ



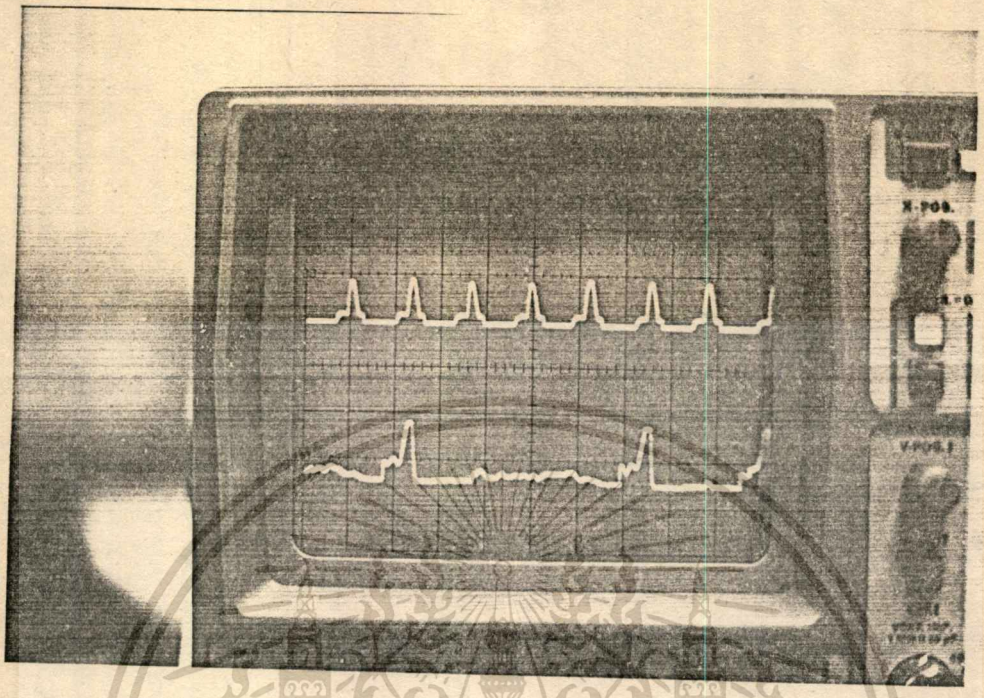
รูปที่วัดได้จาก SCOPE ที่จุดหัวเทป



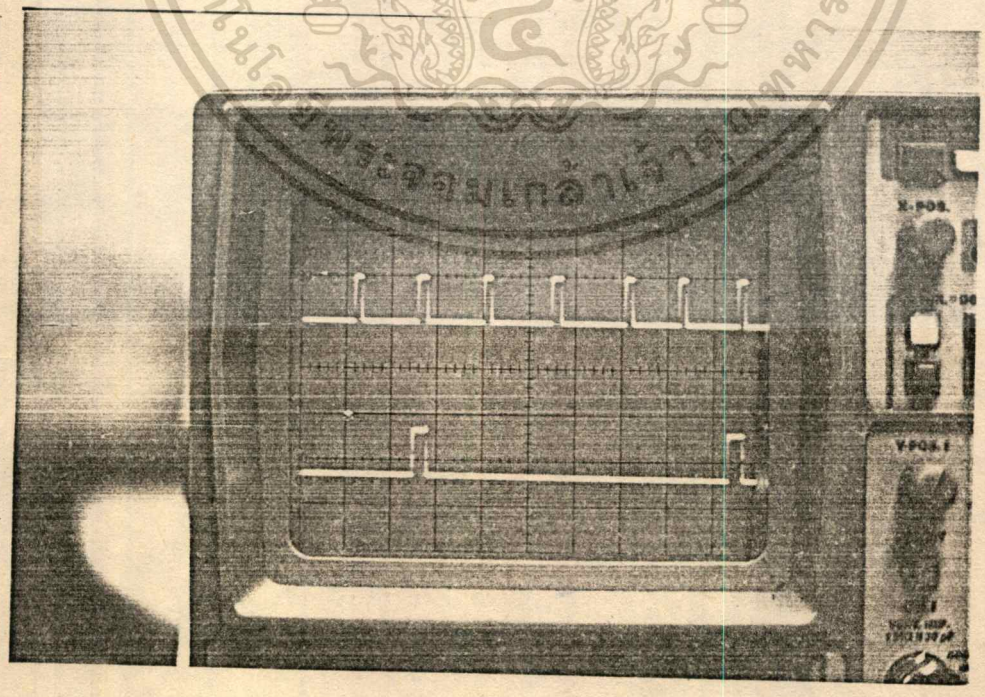
แสดงสัญญาณที่ผ่าน Op-Amp A1



แสดงสัญญาณที่ผ่าน C ขา Op-Amp A2

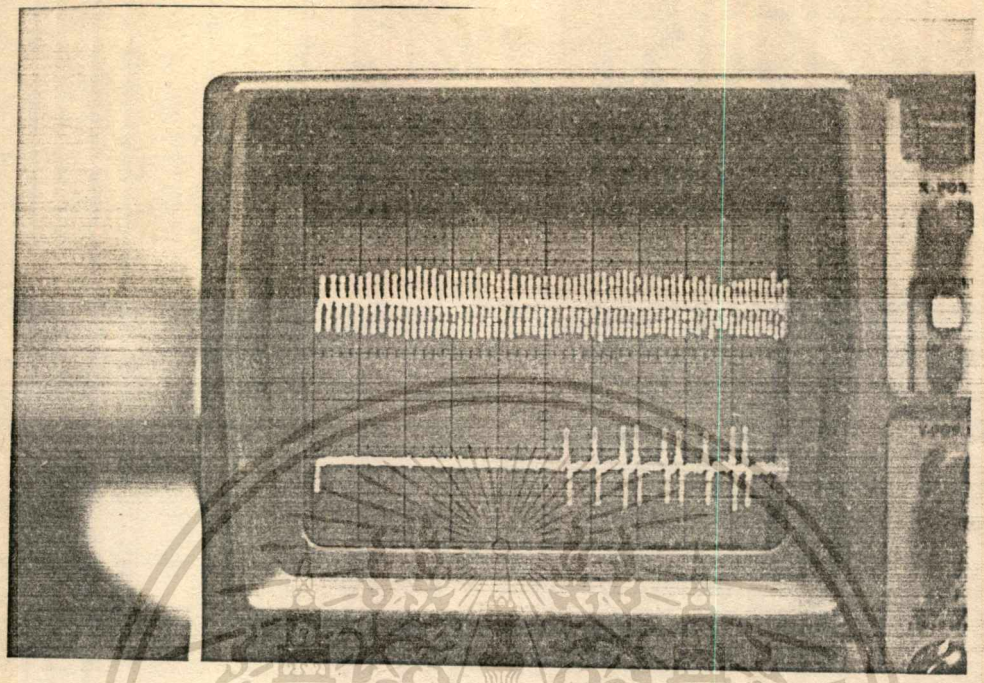


แสดงสัญญาณที่ผ่าน Op-Amp A2

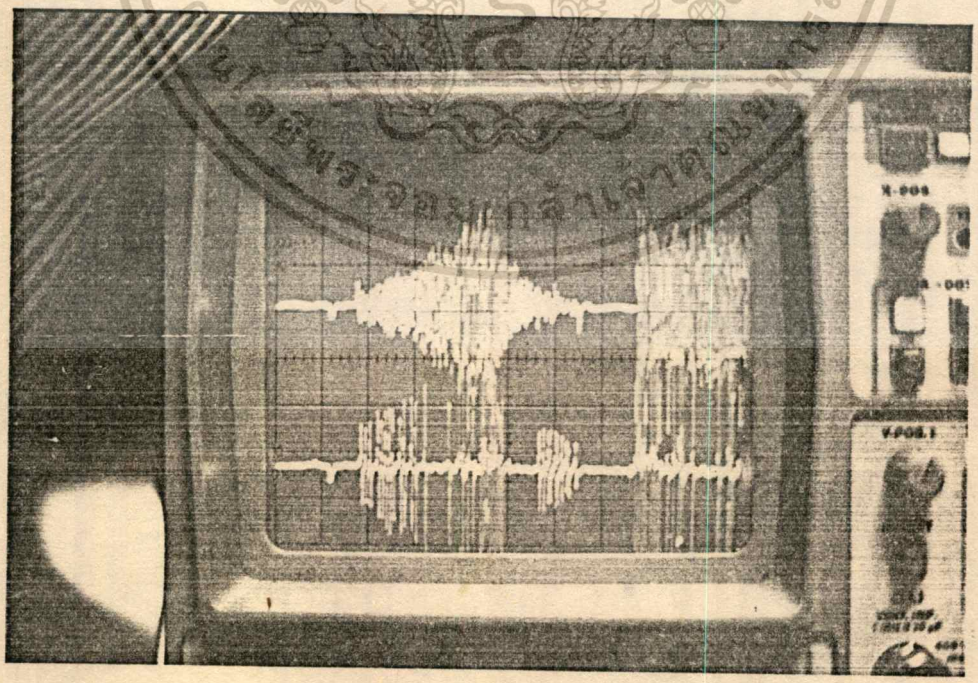


แสดงสัญญาณที่ผ่านวงจร Shermittrigger

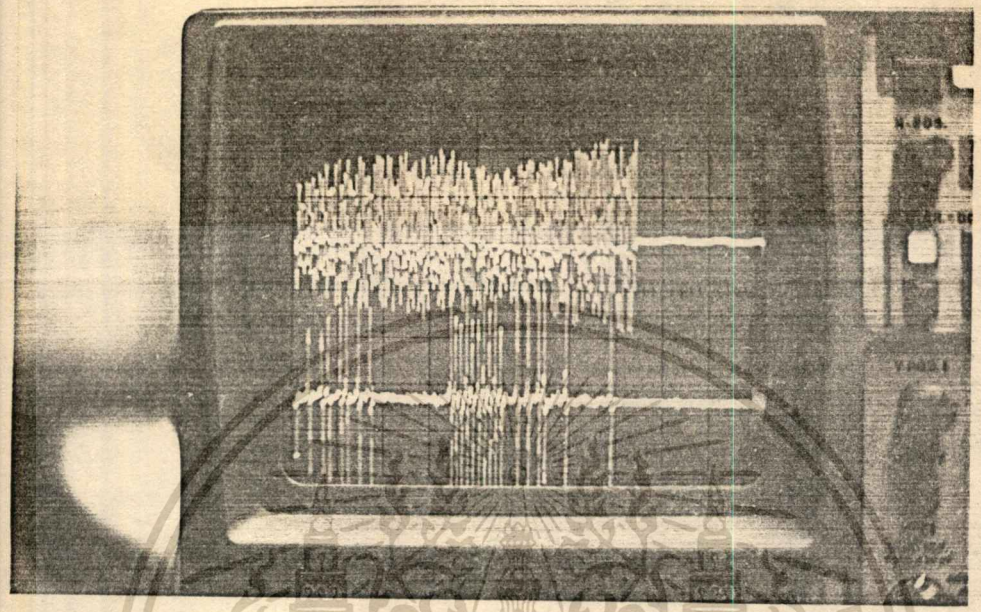
4.2 ผลแสดงการของความเร็วในการรูดบัตร



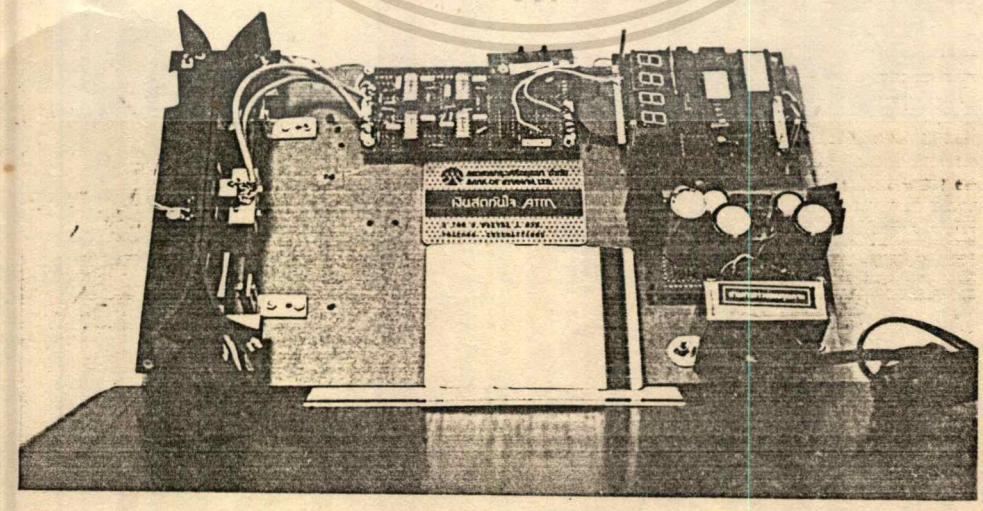
แสดงผลการรูดบัตรช้า

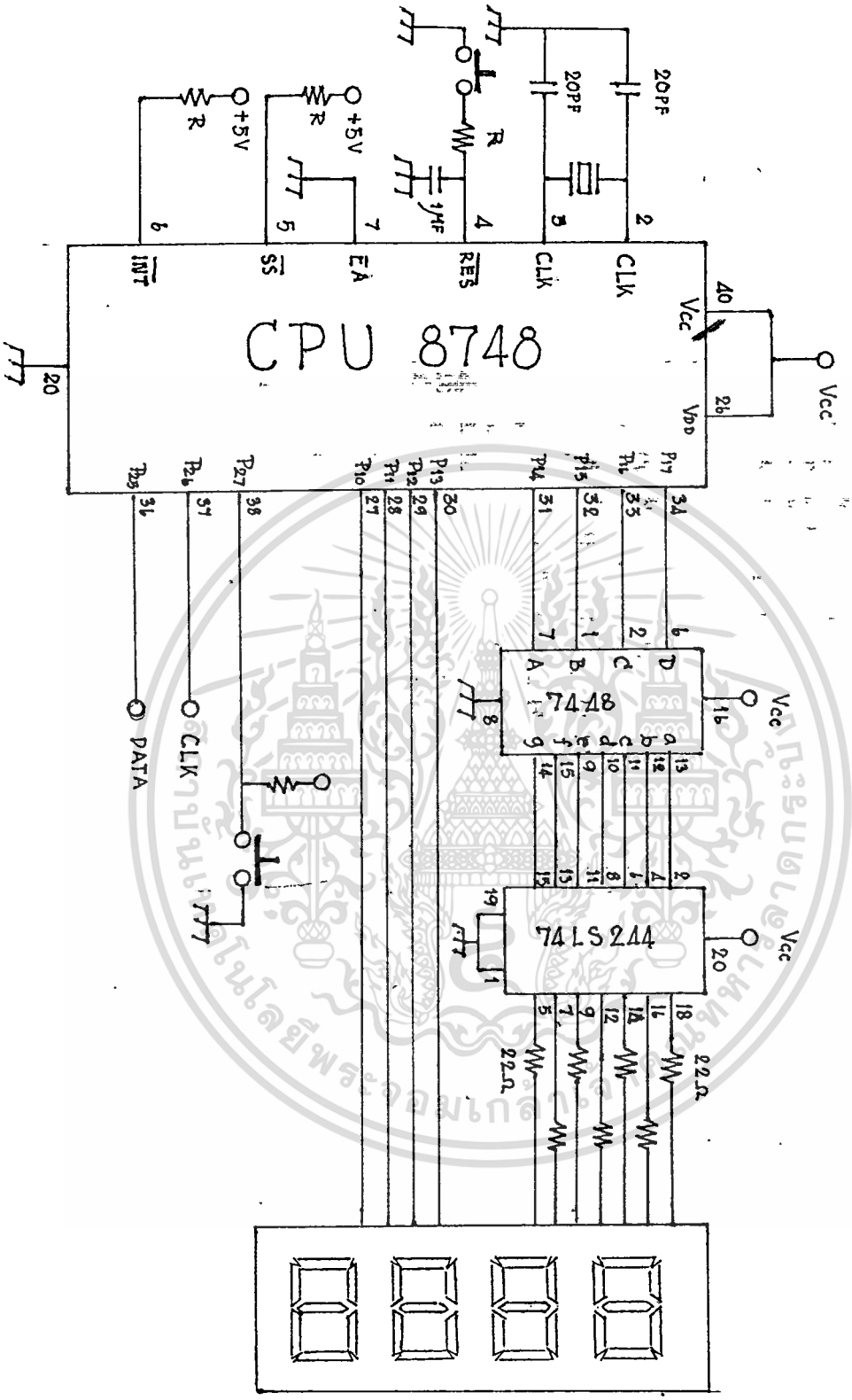


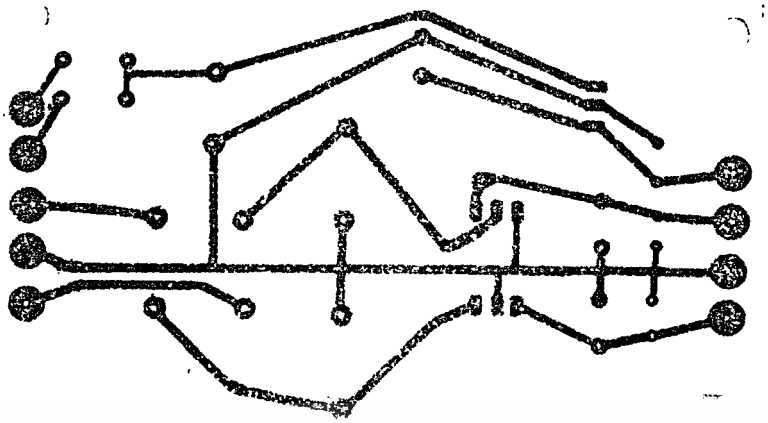
แสดงผลการรูดบัตรเร็ว

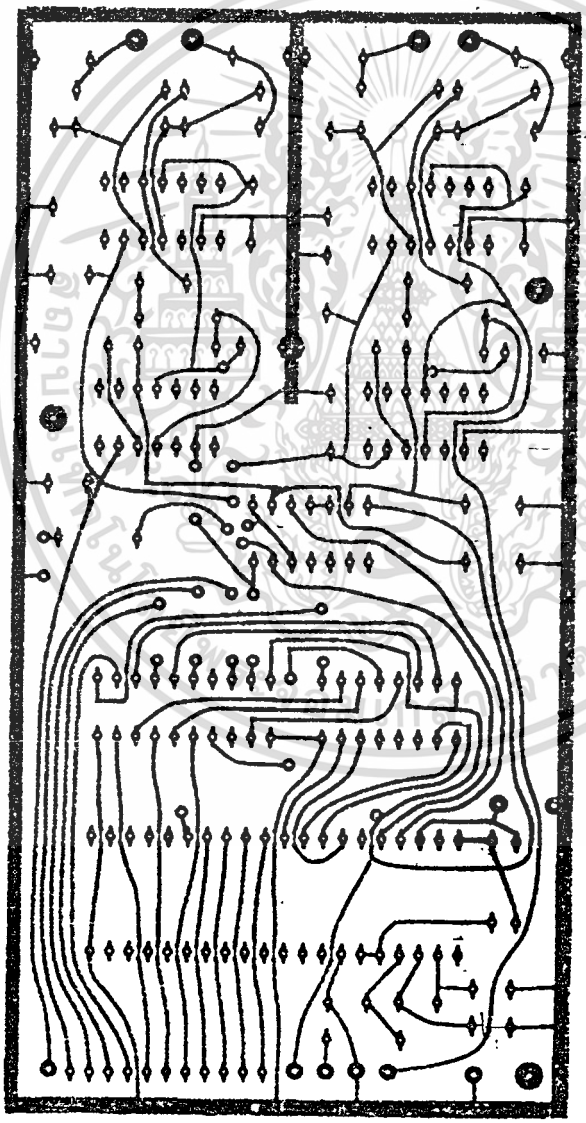


แสดงผลการรูดบัตรปรกติ









```
000     24 00                   :JMP  START

003     34 69       INTRP       :CALL LDELAY
005     1D                       INC  R5
006     93                       RETR

100     23 B7       START       :MOV A, #B7 ;]
                                 OUTL P1,A
103     BA 08                    MOV  R2, #8  ;Bit Count
105     BE 20                    MOV  R6, #20 ;Point Buffer
107     BD 20                    MOV  R5, #20 ;Main Pointer
109     BF 00                    MOV  R7, #0  ;Byte Buffer
10B     05                       EN   I
10C     00
10D     00
10E     00
10F     0A          1            :IN  A, P2      ;CHK CD
110     F2 0F                    JB   7, 1

112     0A          2            :IN  A,P2
113     F2 37                    JB   7,DISP     ;card out
115     D2 12                    JB   6,  2      ;CHK CLK

117     04          3            :IN  A,P2
118     F2 37                    JB   7,DISP     ;card out
11A     D2 1E                    JB   6, 4       ;
11C     24 17                    SMP  3          ;Untit Lood
```

```
11E     00  00        4       :CALL DELAY
120     0A                    IN  A,P2     ;Read Data
121     97                    CLR C
122     B2  26                JB  5, 5
124     24  27                JMP 6

126     A7            5       :CPL C

127     FF            6       :MOV A,R7    ;Shift bit
128     F7                    RLC A        ;7 - 0
129     AF                    MOV R7,A
12A     EA  12                DJNZ  R2, 2  ;Bit count
12C     FE                    MOV A,R6
12D     A8                    MOV R0,A     ;R0 - R6
12E     FF                    MOV A,R7
12F     A0                    MOV @R0,A    ;@R6 - R7
130     BF  00                MOV  R7,#0
132     1E                    INC  R6
133     BA  08                MOV  R2,#8   ; 8 Bit count
135     24  12                JMP  2
137     FD            DISP    :MOV  A,R5
138     A8                    MOV  R0,A
139     B9  0E                MOV  R1,#0E  ;Col 0
13B     34  60                CALL SCAN    ;LOW Nibble
13D     F0                    MOV  A,@R0   ;High Nibble
13E     53  F0                ANL  A,#F0
140     43  0D                ORL  A,#0D   ;Col 1
142     39                    OUTL P1,A
```

```
143     34  72                  CALL  DELAY
145     18                      INC   R0
146     B9  0B                  MOV   R1,#0B    ;Col 2
148     34  60                  CALL SCAN       ;LOW Nibble

14A     F0                      MOV   A,@R0     ;High Nibble
14B     53  F0                  ANL   A,#F0
14D     43  07                  ORL   A,#07     ;Col 3
14F     39                      OUTL P1,A
150     34  72                  CALL DELAY
152     0A                      IN    A,P2
153     F2  37                  JB  7,DISP
155     24  00                  JMP START
160     F0          SCAN        :MOV  A,@R0
161     47                      SWAP A
162     53  F0                  ANL   A,#F0
164     49                      ORL   A,R1
165     39                      OUTL P1,A
166     34  72                  CALL DELAY
168     83                      RET
169     BB  F0      DELAY       :MOV  R3,#F0
16B     BC  00      L2          :MOV  R4,#0
16D     EC  6D      L1          :DJNZ R4,L1
16F     EB  6B                   DJNZ R3,L2
171     83                       RET
172     BB  05      DELAY       :MOV  R3,#10
174     BC  A0      LL2         :MOV  R4,#10
176     EC  76      LL1         :DJNZ R4,LL1
178     EB  74                   DJNZ R3,LL2
17A     83                       RET
```

สรุป

จากการทดลองของเครื่อง Card Reader นี้ เราจะสังเกตุเห็นได้ว่า เราจะเน้นทางด้านประสิทธิภาพของเครื่องในระบบให้ได้สูงสุด ซึ่งจากการทดลองนี้พบว่าประสิทธิภาพของเครื่องจะเกิดจากตัวผู้ใช้เครื่องเอง คือ การรูดบัตรในแต่ละครั้ง เมื่อสอดบัตรเข้าไปในเครื่องแล้ว ควรรูดบัตรพอเหมาะกับการอ่านของหัวเทป คือไม่รูดบัตรเร็วจนเกินไป หรือช้าเกินไป อาจทำให้เครื่องอ่านผิดพลาดได้ และเวลารูดบัตรก็ไม่ควรที่จะบิดบัตรไปทางด้านใดด้านหนึ่ง ควรจะรูดในลักษณะตรง ๆ

การที่เครื่องอ่านผิดพลาดนี้อาจเกิดจากเครื่องอัดก็ได้ถึงแม้ว่าข้อมูลที่อัดลงไปใน Card จะสูงพอก็ตาม และอาจเกิดจากเครื่องอ่านบัตรด้วย คือ เครื่องอ่านอาจอ่านผิดพลาดบ้าง เพราะฉนั้น จะต้องมีการปรับปรุงทางด้าน Mechanism ของตัวอ่านข้อมูลบ้าง เช่น ให้ปรับปรุงทางด้านการอ่านข้อมูลให้สามารถ Detect สัญญาณได้ถึงแม้นจะรูดบัตรให้ช้าลงหรือเร็วขึ้นก็จะมีผลน้อย นอกจากนี้ยังต้องปรับปรุงทางระบบหัวเทปที่ใช้ในการอ่านข้อมูล ให้มีความเที่ยงตรงในการอ่านจากสารแม่เหล็กที่เคลือบไว้ในบัตรและอาจเกิดจากอุปกรณ์บางชิ้นที่นำมาประกอบวงจรมีคุณภาพไม่ดีเท่าที่ควร

กิตติกรรมประกาศ

ปริญญานิพนธ์ ฉบับนี้ที่ได้สำเร็จลุล่วงไปได้ เพราะด้วยความกรุณาของอาจารย์ที่ปรึกษา ที่ให้คำแนะนำและปรึกษาเกี่ยวกับปัญหาต่าง ๆ มาตลอดเวลาที่ทำการวิจัยทางด้านผลงานชิ้นนี้ ทางผู้จัดทำก็ได้ขอขอบคุณ อาจารย์อุทัย ศรีธีระวิโรจน์ ไว้เป็นอย่างสูงไว้ ณ ที่นี้ด้วย

ผู้จัดทำ

บทความและหนังสืออ้างอิง

1. พลผดุง ผดุงกุล 'ทฤษฎี OP-AMP' วารสารเซมิคอนดัคเตอร์ อิเลคทรอนิคส์
ฉบับที่ 5 ปี 2531 หน้า 200-204

2. ยืน ภู่วรรณ 'RAM PACK' วารสารเซมิคอนดัคเตอร์ อิเลคทรอนิคส์
ฉบับที่ 50 ปี 2528 หน้า 71-74

3. PHILIPS APADOR 'CARD READER' FIELD SUPPORT MANUAL BADGDE
P.P. 1-17 - 1-25

4. TEXAS INSTRUMENTS ' TTL COOK BOOK' THE TTL DAATA BOOK
INC. 1976 P.P. 5-55 - 7-31

5. INTELLEC '8748' MICROCOMPUTER USER'S MANUAL
INC. 1977 P.P. 2-2 - 6-5

PRELIMINARY

8048/8748/8035

SINGLE COMPONENT 8-BIT MICROCOMPUTER

*8048 Mask Programmable ROM
*8748 User Programmable/Erasable EPROM
*8035 External ROM or EPROM

- 8-Bit CPU, ROM, RAM, I/O in Single Package
- Interchangeable ROM and EPROM Versions
- Single 5V Supply
- 2.5 μsec and 5.0 μsec Cycle Versions All Instructions 1 or 2 Cycles.
- Over 90 Instructions: 70% Single Byte
- 1K x 8 ROM/EPROM
- 64 x 8 RAM
- 27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Compatible with MCS-80™ Peripherals
- Single Level Interrupt

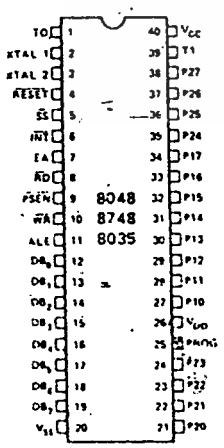
The Intel® 8048/8748/8035 is a totally self-sufficient 8-bit parallel computer fabricated on a single silicon chip using Intel's N-channel silicon gate MOS process.

The 8048 contains a 1K x 8 program memory, a 64 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on board oscillator and clock circuits. For systems that require extra capability, the 8048 can be expanded using standard memories and MCS-80™ (8080A) peripherals. The 8035 is the equivalent of an 8048 without program memory.

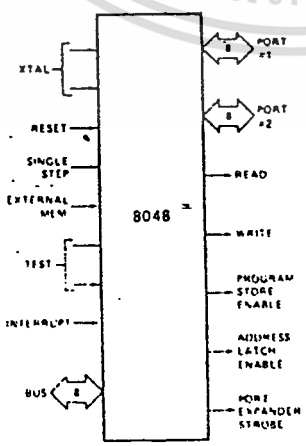
To reduce development problems to a minimum and provide maximum flexibility, three interchangeable pin-compatible versions of this single component microcomputer exist: the 8748 with user-programmable and erasable EPROM program memory for prototype and preproduction systems, the 8048 with factory-programmed mask ROM program memory for low-cost high volume production, and the 8035 without program memory for use with external program memories.

This microprocessor is designed to be an efficient controller as well as an arithmetic processor. The 8048 has extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over two bytes in length.

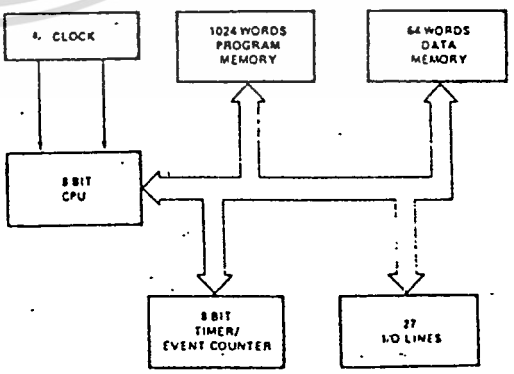
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycles
Accumulator	ADD A, R	Add register to A	1	1
	ADD A, @R	Add data memory to A	1	1
	ADD A, #data	Add immediate to A	2	2
	ADDC A, R	Add register with carry	1	1
	ADDC A, @R	Add data memory with carry	1	1
	ADDC A, #data	Add immediate with carry	2	2
	ANL A, R	And register to A	1	1
	ANL A, @R	And data memory to A	1	1
	ANL A, #data	And immediate to A	2	2
	ORL A, R	Or register to A	1	1
	ORL A, @R	Or data memory to A	1	1
	ORL A, #data	Or immediate to A	2	2
	XRL A, R	Exclusive Or register to A	1	1
	XRL A, @R	Exclusive or data memory to A	1	1
	XRL A, #data	Exclusive or immediate to A	2	2
	INC A	Increment A	1	1
	DEC A	Decrement A	1	1
	CLR A	Clear A	1	1
	CPL A	Complement A	1	1
	DA A	Decimal Adjust A	1	1
	SWAP A	Swap nibbles of A	1	1
	RL A	Rotate A left	1	1
	RLC A	Rotate A left through carry	1	1
	RR A	Rotate A right	1	1
	RRC A	Rotate A right through carry	1	1
Input/Output	IN A, P	Input port to A	1	2
	OUTL P, A	Output A to port	1	2
	ANL P, #data	And immediate to port	2	2
	ORL P, #data	Or immediate to port	2	2
	INS A, BUS	Input BUS to A	1	2
	OUTL BUS, A	Output A to BUS	1	2
	ANL BUS, #data	And immediate to BUS	2	2
	ORL BUS, #data	Or immediate to BUS	2	2
	MOVD A, P	Input Expander port to A	1	2
	MOVD P, A	Output A to Expander port	1	2
	ANLD P, A	And A to Expander port	1	2
	ORLD P, A	Or A to Expander port	1	2
Registers	INC R	Increment register	1	1
	INC @R	Increment data memory	1	1
	DEC R	Decrement register	1	1
Branch	JMP addr	Jump unconditional	2	2
	JMPP @A	Jump indirect	1	2
	DJNZ R, addr	Decrement register and skip	2	2
	JC addr	Jump on Carry = 1	2	2
	JNC addr	Jump on Carry = 0	2	2
	J Z addr	Jump on A Zero	2	2
	JNZ addr	Jump on A not Zero	2	2
	JT0 addr	Jump on T0 = 1	2	2
	JNT0 addr	Jump on T0 = 0	2	2
	JT1 addr	Jump on T1 = 1	2	2
	JNT1 addr	Jump on T1 = 0	2	2
	JF0 addr	Jump on F0 = 1	2	2
	JF1 addr	Jump on F1 = 1	2	2
	JTF addr	Jump on timer flag	2	2
	JNI addr	Jump on $\overline{INT}$ = 0	2	2
	JBb addr	Jump on Accumulator Bit	2	2

	Mnemonic	Description	Bytes	Cycles
Subroutine	CALL	Jump to subroutine	2	2
	RET	Return	1	2
	RETR	Return and restore status	1	2
Flags	CLR C	Clear Carry	1	1
	CPL C	Complement Carry	1	1
	CLR F0	Clear Flag 0	1	1
	CPL F0	Complement Flag 0	1	1
	CLR F1	Clear Flag 1	1	1
	CPL F1	Complement Flag 1	1	1
Data Moves	MOV A, R	Move register to A	1	1
	MOV A, @R	Move data memory to A	1	1
	MOV A, #data	Move immediate to A	2	2
	MOV R, A	Move A to register	1	1
	MOV @R, A	Move A to data memory	1	1
	MOV R, #data	Move immediate to register	2	2
	MOV @R, #data	Move immediate to data memory	2	2
	MOV A, PSW	Move PSW to A	1	1
	MOV PSW, A	Move A to PSW	1	1
	XCH A, R	Exchange A and register	1	1
	XCH A, @R	Exchange A and data memory	1	1
	XCHD A, @R	Exchange nibble of A and register	1	1
	MOVX A, @R	Move external data memory to A	1	2
	MOVX @R, A	Move A to external data memory	1	2
	MOVP A, @A	Move to A from current page	1	2
	MOVP3 A, @A	Move to A from Page 3	1	2
Timer/Counter	MOV A, T	Read Timer/Counter	1	1
	MOV T, A	Load Timer/Counter	1	1
	STRT T	Start Timer	1	1
	STRT CNT	Start Counter	1	1
	STOP TCNT	Stop Timer/Counter	1	1
	EN TCNTI	Enable Timer/Counter Interrupt	1	1
	DIS TCNTI	Disable Timer/Counter Interrupt	1	1
Control	EN I	Enable external interrupt	1	1
	DIS I	Disable external interrupt	1	1
	SEL RB0	Select register bank 0	1	1
	SEL RB1	Select register bank 1	1	1
	SEL MB0	Select memory bank 0	1	1
	SEL MB1	Select memory bank 1	1	1
	ENT0 CLK	Enable Clock output on T0	1	1
	NOP	No Operation	1	1

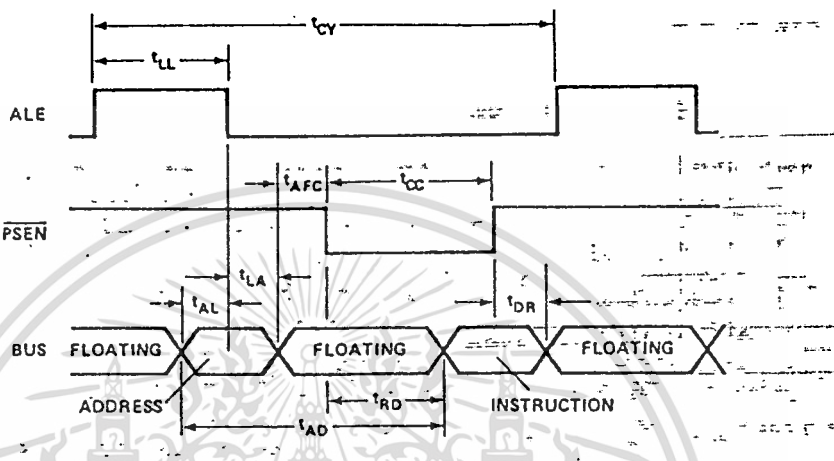
PIN DESCRIPTION

Designation	Pin #	Function
V_{SS}	20	Circuit GND potential
V_{DD}	26	Programming power supply; +25V during program, +5V during operation for both ROM and PROM. Low power standby pin in 8048 ROM version.
V_{CC}	40	Main power supply: +5V during operation and programming.
PROG	25	Program pulse (+25V) input pin during 8748 programming.
		Output strobe for 8243 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-P27 Port 2	21-24 35-38	8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243
D0-D7 BUS	12-19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$, $\overline{WR}$ strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$, and $\overline{WR}$.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction. T0 is also used during programming.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)

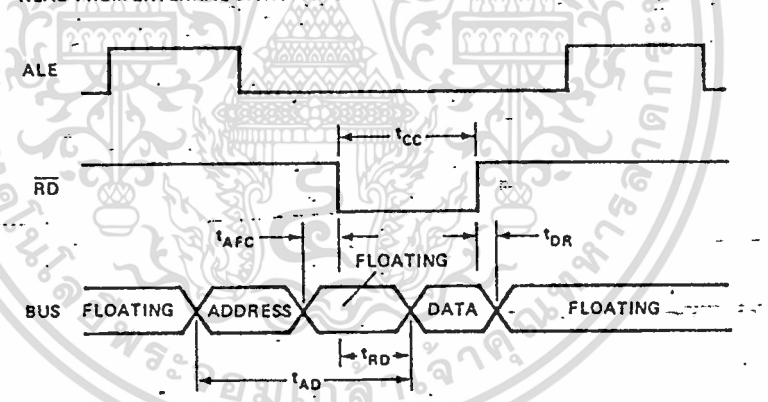
Designation	Pin #	Function
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. Used as a Read Strobe to External Data Memory. (Active low)
$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during PROM programming verification, and power-down. (Active low)
$\overline{WR}$	10	Output strobe during a BUS write. (Active low)(Non TTL V_{IH}) Used as write strobe to External Data Memory.
ALE	11	Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.
$\overline{PSEN}$	9	Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)
$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low)
EA	7	External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Not TTL Compatible)
XTAL2	2	Other side of crystal input.

WAVEFORMS

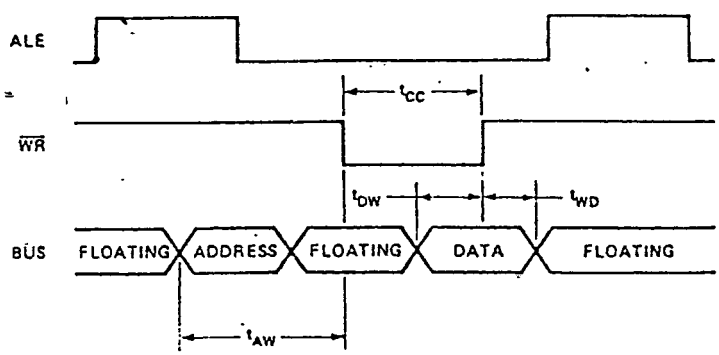
INSTRUCTION FETCH FROM EXTERNAL PROGRAM MEMORY



READ FROM EXTERNAL DATA MEMORY



WRITE TO EXTERNAL DATA MEMORY



PRELIMINARY

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... -65°C to +150°C
Voltage On Any Pin With Respect to Ground ..... -0.5V to +7V
Power Dissipation ..... 1.5 Watt

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

D.C. AND OPERATING CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ±10%*, V_{SS} = 0V

Symbol	Parameter	Limits Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage (All Except XTAL1, XTAL2)	-.5		.8	V	
V_{IH}	Input High Voltage (All Except XTAL1,XTAL2,RESET)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage (RESET,XTAL1)	3.0		V_{CC}	V	
V_{OL}	Output Low Voltage (BUS, RD, WR, PSEN, ALE)			.45	V	I_{OL} = 2.0mA
V_{OL1}	Output Low Voltage (All Other Outputs Except PROG)			.45	V	I_{OL} = 1.6mA
V_{OH}	Output High Voltage (BUS, RD, WR, PSEN, ALE)	2.4			V	I_{OH} = 100µA
V_{OH1}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = 50µA
I_{IL}	Input Leakage Current (T1, EA, INT)			±10	µA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{OL}	Output Leakage Current (Bus, T0) (High Impedance State)			-10	µA	$V_{CC} \ge V_{IN} \ge V_{SS}$ +.45
I_{DD}	Power Down Supply Current		10	25	mA	T_A = 25°C
$I_{DD} + I_{CC}$	Total Supply Current		65	135	mA	T_A = 25°C

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ±10%*, V_{SS} = 0V

Symbol	Parameter	8048/8748/8035 Min.	8048/8748/8035 Max.	8748-8 -8035-8 Min.	8748-8 -8035-8 Max.	Unit	Conditions
t_{LL}	ALE Pulse Width	400		800		ns	
t_{AL}	Address Setup to ALE	150		150		ns	
t_{LA}	Address Hold from ALE	80		80		ns	
t_{CC}	Control Pulse Width (PSEN, RD, WR)	900		1800		ns	
t_{DW}	Data Set-Up Before WR	500		1000		ns	
t_{WD}	Data Hold After WR	120		120		ns	C_L = 20pF
t_{CY}	Cycle Time	2.5	15.0	5.0	15.0	µs	6 MHz XTAL (3 MHz XTAL for -8)
t_{DR}	Data Hold	0	200	0	200	ns	
t_{RD}	PSEN, RD to Data In		500		1000	ns	
t_{AW}	Address Setup to WR	230		260		ns	
t_{AD}	Address Setup to Data In		950		1900	ns	
t_{AFC}	Address Float to RD, PSEN	0		0		ns	

A.C. TEST CONDITIONS

Control Outputs: C_L = 80 pF, 2.2K to V_{SS}, 4.3K to V_{CC}
BUS Outputs: C_L = 150 pF, 2.2K to V_{SS}, 4.3K to V_{CC} t_{CY} = 2.5µs

*Standard 8748 and 8035 ±5%, ±10% available.

INSTRUCTION SET SUMMARY

	Mnemonic	Description	Bytes	Cycle
Accumulator	ADD A, R	Add register to A	1	1
	ADD A, @R	Add data memory to A	1	1
	ADD A, #data	Add immediate to A	2	2
	ADDC A, R	Add register with carry	1	1
	ADDC A, @R	Add data memory with carry	1	1
	ADDC A, #data	Add immediate with carry	2	2
	ANL A, R	And register to A	1	1
	ANL A, @R	And data memory to A	1	1
	ANL A, #data	And immediate to A	2	2
	ORL A, R	Or register to A	1	1
	ORL A, @R	Or data memory to A	1	1
	ORL A, #data	Or immediate to A	2	2
	XRL A, R	Exclusive Or register to A	1	1
	XRL A, @R	Exclusive or data memory to A	1	1
	XRL A, #data	Exclusive or immediate to A	2	2
	INC A	Increment A	1	1
	DEC A	Decrement A	1	1
	CLR A	Clear A	1	1
	CPL A	Complement A	1	1
	DA A	Decimal Adjust A	1	1
	SWAP A	Swap nibbles of A	1	1
	RL A	Rotate A left	1	1
	RLC A	Rotate A left through carry	1	1
	RR A	Rotate A right	1	1
	RRC A	Rotate A right through carry	1	1
Input/Output	IN A, P	Input port to A	1	2
	OUTL P, A	Output A to port	1	2
	ANL P, #data	And immediate to port	2	2
	ORL P, #data	Or immediate to port	2	2
	INS A, BUS	Input BUS to A	1	2
	OUTL BUS, A	Output A to BUS	1	2
	ANL BUS, #data	And immediate to BUS	2	2
	ORL BUS, #data	Or immediate to BUS	2	2
	MOVD A, P	Input Expander port to A	1	2
	MOVD P, A	Output A to Expander port	1	2
	ANLD P, A	And A to Expander port	1	2
	ORLD P, A	Or A to Expander port	1	2
Registers	INC R	Increment register	1	1
	INC @R	Increment data memory	1	1
	DEC R	Decrement register	1	1
Branch	JMP addr	Jump unconditional	2	2
	JMPP @A	Jump indirect	1	2
	DJNZ R, addr	Decrement register and skip	2	2
	JC addr	Jump on Carry = 1	2	2
	JNC addr	Jump on Carry = 0	2	2
	JZ addr	Jump on A Zero	2	2
	JNZ addr	Jump on A not Zero	2	2
	JT0 addr	Jump on T0 = 1	2	2
	JNT0 addr	Jump on T0 = 0	2	2
	JT1 addr	Jump on T1 = 1	2	2
	JNT1 addr	Jump on T1 = 0	2	2
	JF0 addr	Jump on F0 = 1	2	2
	JF1 addr	Jump on F1 = 1	2	2
	JTF addr	Jump on timer flag	2	2
	JNI addr	Jump on $\overline{INT}$ = 0	2	2
	JBb addr	Jump on Accumulator Bit	2	2

	Mnemonic	Description	Bytes	Cycles
Subroutine	CALL	Jump to subroutine	2	2
	RET	Return	1	2
	RETR	Return and restore status	1	2
Flags	CLR C	Clear Carry	1	1
	CPL C	Complement Carry	1	1
	CLR F0	Clear Flag 0	1	1
	CPL F0	Complement Flag 0	1	1
	CLR F1	Clear Flag 1	1	1
	CPL F1	Complement Flag 1	1	1
Data Moves	MOV A, R	Move register to A	1	1
	MOV A, @R	Move data memory to A	1	1
	MOV A, #data	Move immediate to A	2	2
	MOV R, A	Move A to register	1	1
	MOV @R, A	Move A to data memory	1	1
	MOV R, #data	Move immediate to register	2	2
	MOV @R, #data	Move immediate to data memory	2	2
	MOV A, PSW	Move PSW to A	1	1
	MOV PSW, A	Move A to PSW	1	1
	XCH A, R	Exchange A and register	1	1
	XCHA, @R	Exchange A and data memory	1	1
	XCHD A, @R	Exchange nibble of A and register	1	1
	MOVX A, @R	Move external data memory to A	1	2
	MOVX @R, A	Move A to external data memory	1	2
	MOVP A, @A	Move to A from current page	1	2
	MOVP3 A, @A	Move to A from Page 3	1	2
Timer/Counter	MOV A, T	Read Timer/Counter	1	1
	MOV T, A	Load Timer/Counter	1	1
	STRT T	Start Timer	1	1
	STRT CNT	Start Counter	1	1
	STOP TCNT	Stop Timer/Counter	1	1
	EN TCNTI	Enable Timer/Counter Interrupt	1	1
	DIS TCNTI	Disable Timer/Counter Interrupt	1	1
Control	EN I	Enable external interrupt	1	1
	DIS I	Disable external interrupt	1	1
	SEL RB0	Select register bank 0	1	1
	SEL RB1	Select register bank 1	1	1
	SEL MB0	Select memory bank 0	1	1
	SEL MB1	Select memory bank 1	1	1
	ENT0 CLK	Enable Clock output on T0	1	1
	NOP	No Operation	1	1

MCS-48™ INSTRUCTION SET

SYMBOLS AND ABBREVIATIONS USED

A	Accumulator
AC	Auxillary Carry
addr	12-Bit Program Memory Address
Bb	Bit Designator (b=0-7)
BS	Bank Switch
BUS	BUS Port
C	Carry
CLK	Clock
CNT	Event Counter
D	Mnemonic for 4-Bit Digit (Nibble)
data	8-Bit Number or Expression
DBF	Memory Bank Flip-Flop
F0, F1	Flag 0, Flag 1
I	Interrupt
P	Mnemonic for "in-page" Operation
PC	Program Counter
Pp	Port Designator (p=1, 2 or 4-7)
PSW	Program Status Word
Rr	Register Designator (r=0, 1 or 0-7)
SP	Stack Pointer
T	Timer
TF	Timer Flag
T0, T1	Test 0, Test 1
X	Mnemonic for External RAM
#	Immediate Data Prefix
@	Indirect Address Prefix
$	Current Value of Program Counter
(X)	Contents of X
((X))	Contents of Location Addressed by X
←	Is Replaced by

INSTRUCTION SET

ADD A,R_r Add Register Contents to Accumulator

0 1 1 0	1 r r r

The contents of register 'r' are added to the accumulator. Carry is affected.

(A) ← (A) + (Rr) r=0-7

```
Example: ADDREG: ADD A,R6   ;ADD REG 6 CONTENTS
                            ;TO ACC
```

ADD A,@R_r Add Data Memory Contents to Accumulator

0 1 1 0	0 0 0 r

The contents of the resident data memory location addressed by register 'r' bits 0-5 are added to the accumulator. Carry is affected.

(A) ← (A) + ((Rr)) r=0-1

```
Example: ADDM: MOV R0, #0AFH  ;MOVE 'AF' HEX TO REG 0
               ADD A, @R0     ;ADD VALUE OF LOCATION
                              ;47 TO ACC
```

ADD A,#data Add Immediate Data to Accumulator

0 0 0 0	0 0 1 1	$d_7\ d_6\ d_5\ d_4$	$d_3\ d_2\ d_1\ d_0$

This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected

(A) ← (A) + data

```
Example: ADDID: ADD A,#ADDER: ;ADD VALUE OF SYMBOL
                              ;'ADDER' TO ACC
```

ADDC A,R_r Add Carry and Register Contents to Accumulator

0 1 1 1	1 r r r

The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.

(A) ← (A)+(Rr)+(C) r=0-7

```
Example: ADDRGC: ADDC A,R4  ;ADD CARRY AND REG 4
                            ;CONTENTS TO ACC
```

ADDC A,@R_r Add Carry and Data Memory Contents to Accumulator

0111	000r

The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'r' bits 0-5 are added to the accumulator. Carry is affected.

(A) ← (A)+((Rr))+(C) r=0-1

```
Example: ADDMC: MOV R1,#40   ;MOVE '40' DEC TO REG 1
                ADDC A,@R1   ;ADD CARRY AND LOCATION 40
                             ;CONTENTS TO ACC
```

ADDC A,#data Add Carry and Immediate Data to Accumulator

0001	0011	$d_7 d_6 d_5 d_4$	$d_3 d_2 d_1 d_0$

This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

(A) ← (A)+data+(C)

```
Example:  ADDC A,#225   ;ADD CARRY AND '225' DEC
                        ;TO ACC
```

ANL A,R_r Logical AND Accumulator With Register Mask

0101	1rrr

Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

(A) ← (A) AND (Rr) r=0-7

```
Example: ANDREG: ANL A,R3   ;'AND' ACC CONTENTS WITH MASK
                            ;IN REG 3
```

ANL A,@R_r Logical AND Accumulator With Memory Mask

0101	000r

Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'r', bits 0-5.

(A) ← (A) AND ((Rr)) r=0-1

```
Example: ANDDM: MOV R0,#0FFH ;MOVE 'FF' HEX TO REG 0
                ANL A, @R0   ;'AND' ACC CONTENTS WITH
                             ;MASK IN LOCATION 63
```

INSTRUCTION SET

ANL A,#data Logical AND Accumulator With Immediate Mask

0 1 0 1	0 0 1 1	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.

(A) ← (A) AND data

```
Examples: ANDID: ANL A,#OAFH   ;'AND' ACC CONTENTS
                               ;WITH MASK 10101111
                 ANL A,#3+X/Y  ;'AND' ACC CONTENTS
                               ;WITH VALUE OF EXP
                               ;'3+X/Y'
```

ANL BUS,#data Logical AND BUS With Immediate Mask

1 0 0 1	1 0 0 0	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. Data on the BUS port is logically ANDed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS, A' instruction.

(BUS) ← (BUS) AND data

```
Example: ANDBUS: ANL BUS, #MASK ;'AND' BUS CONTENTS
                                ;WITH MASK EQUAL VALUE
                                ;OF SYMBOL 'MASK'
```

ANL Pp,#data Logical AND Port 1-2 With Immediate Mask

1 0 0 1	1 0 p p	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.

(Pp) ← (Pp) AND data p=1-2

```
Example: ANDP2: ANL P2,#0F0H  ;'AND' PORT 2 CONTENTS
                              ;WITH MASK 'F0' HEX
                              ;(CLEAR P20-23)
```

ANLD Pp,A Logical AND Port 4-7 With Accumulator Mask

1 0 0 1	1 1 p p

This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.

(Pp) ← (Pp) AND (A0-3) p=4-7

CLR A Clear Accumulator

0 0 1 0	0 1 1 1

The contents of the accumulator are cleared to zero.

A ← 0

CLR C Clear Carry Bit

1 0 0 1	0 1 1 1

During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA instructions. This instruction resets the carry bit to zero.

C ← 0

CLR F1 Clear Flag 1

1 0 1 0	0 1 0 1

Flag 1 is cleared to zero.

(F1) ← 0

CLR F0 Clear Flag 0

1 0 0 0	0 1 0 1

Flag 0 is cleared to zero.

(F0) ← 0

CPL A Complement Accumulator

0 0 1 1	0 1 1 1

The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

(A) ← NOT (A)

Example: Assume accumulator contains 01101010.

```
CPLA: CPL A          ;ACC CONTENTS ARE COMPLE-
                     ;MENTED TO 10010101
```

CPL C Complement Carry Bit

1 0 1 0	0 1 1 1

The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

(C) ← NOT (C)

Example: Set C to one; current setting is unknown.

```
CTO1: CLR C          ;C IS CLEARED TO ZERO
      CPL C          ;C IS SET TO ONE
```

CPL F0 Complement Flag 0

1 0 0 1	0 1 0 1

The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.

F0← NOT (F0)

CPL F1 Complement Flag 1

1 0 1 1	0 1 0 1

The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.

(F1)← NOT (F1).

DA A Decimal Adjust Accumulator

0 1 0 1	0 1 1 1

The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one; otherwise, it is cleared to zero.

Example: Assume accumulator contains 10011011. 9B H = 155 D

```
DA A                    ;ACC ADJUSTED TO 00000001
                        ;WITH C SET

C  AC  7  4  3  0
0  0   1  0  0  1   1  0  1  1
                    0  1  1  0      ADD SIX TO BITS 0-3
0  0   1  0  1  0   0  0  0  1
       0  1  1  0                   ADD SIX TO BITS 4-7
1  0   0  0  0  0   0  0  0  1      OVERFLOW TO C
```

DEC A Decrement Accumulator

0 0 0 0	0 1 1 1

The contents of the accumulator are decremented by one.

(A)← (A)-1

Example: Decrement contents of external data memory location 63.

```
MOV R0,#3FH      ;MOVE '3F' HEX TO REG 0
MOVX A,@R0       ;MOVE CONTENTS OF LOCATION 63
                 ;TO ACC
DEC A            ;DECREMENT ACC
MOVX @R0,A       ;MOVE CONTENTS OF ACC TO
                 ;LOCATION 63 IN EXPANDED
                 ;MEMORY
```

DEC R_r Decrement Register

1 1 0 0	1 r r r

The contents of working register 'r' are decremented by one.

$(Rr) \leftarrow (Rr)-1$ r=0-7

Example: DECR1: DEC R1 ;DECREMENT CONTENTS OF REG 1

DIS I Disable External Interrupt

0 0 0 1	0 1 0 1

External interrupts are disabled. A low signal on the interrupt input pin has no effect.

DIS TCNTI Disable Timer/Counter Interrupt

0 0 1 1	0 1 0 1

Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

DJNZ R_r, address Decrement Register and Test

1 1 1 0	1 r r r	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Register 'r' is decremented and tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

$(Rr) \leftarrow (Rr)-1$ r=0-7
If Rr not 0
$(PC_{0-7}) \leftarrow$ addr

Note: A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

Example: Increment values in data memory locations 50-54.

```
        MOV R0,#50      ;MOVE '50' DEC TO ADDRESS
                        ;REG 0
        MOV R3,#5       ;MOVE '5' DEC TO COUNTER
                        ;REG 3
INCRT:  INC @R0         ;INCREMENT CONTENTS OF
                        ;LOCATION ADDRESSED BY
                        ;REG 0
        INC R0          ;INCREMENT ADDRESS IN REG 0
        DJNZ R3, INCRT  ;DECREMENT REG 3 — JUMP TO
                        ;'INCRT' IF REG 3 NONZERO
        NEXT —          ;'NEXT' ROUTINE EXECUTED
                        ;IF R3 IS ZERO
```

EN I Enable External Interrupt

0 0 0 0	0 1 0 1

External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.

EN TCNTI Enable Timer/Counter Interrupt

0 0 1 0	0 1 0 1

Timer/counter interrupts are enabled. An overflow of this register initiates the interrupt sequence.

ENT0 CLK Enable Clock Output

0 1 1 1	0 1 0 1

The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.

Example:

```
EMTST0: ENT0 CLK        ;ENABLE TO AS CLOCK OUTPUT
```

IN A,Pp Input Port or Data to Accumulator

0 0 0 0	1 0 p p

This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.

(A) ← (Pp) p=1-2

```
Example:  INP12: IN A,P1          ;INPUT PORT 1 CONTENTS
                                  ;TO ACC
                 MOV R6,A         ;MOVE ACC CONTENTS TO
                                  ;REG 6
                 IN A,P2          ;INPUT PORT 2 CONTENTS
                                  ;TO ACC
                 MOV R7,A         ;MOVE ACC CONTENTS TO REG 7
```

INC A Increment Accumulator

0 0 0 1	0 1 1 1

The contents of the accumulator are incremented by one.

(A) ← (A)+1

```
Example:  Increment contents of location 100 in external
          data memory.
          INCA: MOV R0,#100      ;MOVE '100' DEC TO ADDRESS
                                 ;REG 0
                MOVX A,@R0       ;MOVE CONTENTS OF LOCATION
                                 ;100 TO ACC
                INC A            ;INCREMENT A
                MOVX @R0,A       ;MOVE ACC CONTENTS TO
                                 ;LOCATION 100
```

INC R_r Increment Register

0 0 0 1	1 r r r

The contents of working register 'r' are incremented by one.

(Rr) ← (Rr)+1 r=0-7

```
Example:  INCR0: INC R0          ;INCREMENT ADDRESS REG 0
```

INC @R_r Increment Data Memory Location

0 0 0 1	0 0 0 r

The contents of the resident data memory location addressed by register 'r' bits 0-5 are incremented by one.

((Rr)) ← ((Rr))+1 r=0-1

```
Example:  INCDM: MOV R1,#OFFH    ;MOVE ONES TO REG 1
                 INC @R1         ;INCREMENT LOCATION 63
```

INS A,BUS Strobed Input of BUS Data to Accumulator

0 0 0 0	1 0 0 0

This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the RD pulse is dropped. (Refer to section on programming memory expansion for details).

(A) ← (BUS)

```
Example: INPBUS: INS A,BUS    ;INPUT BUS CONTENTS
                              ;TO ACC
```

JBb address Jump If Accumulator Bit is Set

$b_2\ b_1\ b_0\ 1$	0 0 1 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

$(PC_{0-7}) \leftarrow$ addr	If Bb=1
(PC) = (PC)+2	If Bb=0

```
Example: JB4IS1  JB4 NEXT    ;JUMP TO 'NEXT' ROUTINE
                             ;IF ACC BIT 4=1
```

JC address Jump If Carry Is Set

1 1 1 1	0 1 1 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

$(PC_{0-7}) \leftarrow$ addr	If C=1
(PC) = (PC)+2	If C=0

```
Example: JC1: JC OVFLOW      ;JUMP TO 'OVFLOW' ROUTINE
                             ;IF C=1
```

JF0 address Jump If Flag 0 Is Set

1 0 1 1	0 1 1 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

$(PC_{0-7}) \leftarrow$ addr	If F0=1
(PC) = (PC)+2	If F0=0

```
Example: JF0IS1: JF0 TOTAL   ;JUMP TO 'TOTAL' ROUTINE
                             ;IF F0=1
```

JF1 address Jump If Flag 1 Is Set

0 1 1 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.

$(PC_{0-7}) \leftarrow$ addr If F1=1
(PC) = (PC)+2 IF F1=0

```
Example: JF1IS1: JF1 FILBUF     ;JUMP TO 'FILBUF'
                                ;ROUTINE IF F1=1
```

JMP address Direct Jump Within 2K Block

a_{10} a_9 a_8 0	0 1 0 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.

$(PC_{8-10}) \leftarrow$ addr 8-10
$(PC_{0-7}) \leftarrow$ addr 0-7
$(PC_{11}) \leftarrow$ DBF

```
Example: JMP SUBTOT     ;JUMP TO SUBROUTINE 'SUBTOT'
         JMP $-6        ;JUMP TO INSTRUCTION SIX LOCATIONS
                        ;BEFORE CURRENT LOCATION
         JMP 2FH        ;JUMP TO ADDRESS '2F' HEX
```

JMPP @A Indirect Jump Within Page

1 0 1 1	0 0 1 1

This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0-7).

$(PC_{0-7}) \leftarrow ((A))$

```
Example: Assume accumulator contains OFH.
         JMPPAG: JMPP @A    ;JUMP TO ADDRESS STORED IN
                            ;LOCATION 15 IN CURRENT PAGE
```

JNC address Jump If Carry Is Not Set

1 1 1 0	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

	$(PC_{0-7}) \leftarrow$ addr	If C=0
	(PC) = (PC)+2	IF C=1
Example:	JC0: JNC NOVFLO	;JUMP TO 'NOVFLO' ROUTINE ;If C=0

JNI address Jump If Interrupt Input is Low

1 0 0 0	0 1 1 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (=0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

	$(PC_{0-7}) \leftarrow$ addr	If I=0
	(PC) = (PC)+2	If I=1
Example:	LOC 3: JNI EXTINT	;JUMP TO 'EXTINT' ROUTINE ;If I=0

JNT0 address Jump If Test 0 Is Low

0 0 1 0	0 1 1 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low

	$(PC_{0-7}) \leftarrow$ addr	If T0=0
	(PC) = (PC)+2	If T0=1
Example:	JTOLOW: JNT0 60	;JUMP TO LOCATION 60 DEC ;IF T0=0

JNT1 address Jump If Test 1 Is Low

0 1 0 0	0 1 1 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.

$(PC_{0-7}) \leftarrow$ addr	If T1=0
(PC) = (PC)+2	If T1=1

JNZ address Jump If Accumulator Is Not Zero

1 0 0 1	0 1 1 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

This is a 2-cycle instruction. Control pases to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

	$(PC_{0-7}) \leftarrow$ addr	If A≠0
	(PC) = (PC)+2	If A=0
Example:	JACCN0: JNZ 0ABH	;JUMP TO LOCATION 'AB' HEX ;IF ACC VALUE IS NONZERO

INSTRUCTION SET

JTF address Jump If Timer Flag Is Set

0 0 0 1	0 1 1 0	a_7 a_8 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

$(PC_{0-7}) \leftarrow$ addr	If TF=1
(PC) = (PC)+2	If TF=0

```
Example:  JTF1: JTF TIMER      ;JUMP TO 'TIMER' ROUTINE
                               ;IF TF=1
```

JT0 address Jump If Test 0 Is High

0 0 1 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (=1).

$(PC_{0-7}) \leftarrow$ addr	If T0=1
(PC) = (PC)+2	If T0=0

```
Example:  JT0HI: JT0 53        ;JUMP TO LOCATION 53 DEC
                               ;IF T0=1
```

JT1 address Jump If Test 1 Is High

0 1 0 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (=1).

$(PC_{0-7}) \leftarrow$ addr	If T1=1
(PC) = (PC)+2	If T1=0

```
Example:  JT1HI: JT1 COUNT     ;JUMP TO 'COUNT' ROUTINE
                               ;IF T1=1
```

JZ address Jump If Accumulator Is Zero

1 1 0 0	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

$(PC_{0-7}) \leftarrow$ addr	If A=0
(PC) = (PC)+2	If A≠0

```
Example:  JACCO: JZ OA3H       ;JUMP TO LOCATION 'A3' HEX
                               ;IF ACC VALUE IS ZERO
```

INSTRUCTION SET

MOV A,#data Move Immediate Data to Accumulator

0 0 1 0	0 0 1 1	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

(A) ← data

```
Example: MOV A,#0A3H        ;MOVE 'A3' HEX TO ACC
```

MOV A,PSW Move PSW Contents to Accumulator

1 1 0 0	0 1 1 1

The contents of the program status word are moved to the accumulator.

(A) ← (PSW)

Example: Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.

```
BSCHK: MOV A,PSW     ;MOVE PSW CONTENTS TO ACC
       JB4 RB1SET    ;JUMP TO 'RB1SET' IF ACC
                     ;BIT 4=1
```

MOV A,R_r Move Register Contents to Accumulator

1 1 1 1	1 r r r

8-bits of data are moved from working register 'r' into the accumulator.

(A) ← (Rr) r=0-7

```
Example: MAR: MOV A,R3       ;MOVE CONTENTS OF REG 3
                             ;TO ACC
```

MOV A,@R_r Move Data Memory Contents to Accumulator

1 1 1 1	0 0 0 r

The contents of the resident data memory location addressed by bits 0-5 of register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

(A) ← ((Rr)) r=0-1

Example: Assume R1 contains 01110110.

```
MADM: MOV A,@R1      ;MOVE CONTENTS OF DATA MEM
                     ;LOCATION 54 TO ACC
```

MOV A,T Move Timer/Counter Contents to Accumulator

0 1 0 0	0 0 1 0

The contents of the timer/event-counter register are moved to the accumulator.

(A) ← (T)

Example: Jump to "EXIT" routine when timer reaches 64, that is, when bit 6 set — assuming initialization 64.

```
TIMCHK: MOV A,T     ;MOVE TIMER CONTENTS TO
                    ;ACC
        JB6 EXIT    ;JUMP TO 'EXIT' IF ACC BIT
                    ;6=1
```

MOV PSW,A Move Accumulator Contents to PSW

1 1 0 1	0 1 1 1

The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.

(PSW) ← (A)

Example: Move up stack pointer by two memory locations, that is, increment the pointer by one.

```
INCPTR: MOV A,PSW   ;MOVE PSW CONTENTS TO ACC
        INC A       ;INCREMENT ACC BY ONE
        MOV PSW,A   ;MOVE ACC CONTENTS TO PSW
```

MOV R_r,A Move Accumulator Contents to Register

1 0 1 0	1 r r r

The contents of the accumulator are moved to register 'r'.

(Rr) ← (A) r=0-7

Example:

```
MRA: MOV R0,A       ;MOVE CONTENTS OF ACC TO
                    ;REG 0
```

MOV R_r,#data Move Immediate Data to Register

1 0 1 1	1 r_2 r_1 r_0	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

(Rr) ← data r=0-7

```
Examples: MIR4: MOV R4,#HEXTEN ;THE VALUE OF THE SYMBOL
                               ;'HEXTEN' IS MOVED INTO
                               ;REG 4
          MIR 5: MOV R5,#PI*(R*R);THE VALUE OF THE
                               ;EXPRESSION 'PI*(R*R)
                               ;IS MOVED INTO REG 5
          MIR 6: MOV R6, #0ADH ;'AD' HEX IS MOVED INTO
                               ;REG 6
```

MOV @R_r,A Move Accumulator Contents to Data Memory

1 0 1 0	0 0 0 r

The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5 of register 'r'. Register 'r' contents are unaffected.

((Rr)) ← (A) r=0-1

Example: Assume R0 contains 11000111

```
MDMA: MOV @R0,A    ;MOVE CONTENTS OF ACC TO
                   ;LOCATION 7 (REG 7)
```

MOV @R_r,#data Move Immediate Data to Data Memory

1 0 1 1	0 0 0 r	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'r', bits 0-5.

((Rr)) ← data r=0-1

Examples: Move the hexadecimal value AC3F to locations 62-63.

```
MIDM: MOV R0,#62       ;MOVE '62' DEC TO ADDR REG 0
      MOV @R0,#0ACH    ;MOVE 'AC' HEX TO LOCATION 62
      INC R0           ;INCREMENT REG 0 TO '63'
      MOV @R0,#3FH     ;MOVE '3F' HEX TO LOCATION 63
```

MOV T,A Move Accumulator Contents to Timer/Counter

0 1 1 0	0 0 1 0

The contents of the accumulator are moved to the timer/event-counter register.

(T) ← (A)

Example: Initialize and start event counter.

```
INITEC: CLR A      ;CLEAR ACC TO ZEROS
        MOV T,A    ;MOVE ZEROS TO EVENT COUNTER
        STRT CNT   ;START COUNTER
```

MOVD A,Pp Move Port 4-7 Data to Accumulator

0 0 0 0	1 1 p p

This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

(0-3) ← (Pp) p=4-7
(4-7) ← 0

Note: Bits 0-1 of the opcode are used to represent ports 4-7. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits 1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

```
Example:  INPPT5: MOVD A,P5    ;MOVE PORT 5 DATA TO ACC
                               ;BITS 0-3, ZERO ACC BITS 4-7
```

MOVD Pp,A Move Accumulator Data to Port 4-7

0 0 1 1	1 1 p p

Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

(Pp) ← (A_{0-3}) p=4-7

```
Example:  Move data in accumulator to ports 4 and 5.
          OUTP45: MOVD P4,A    ;MOVE ACC BITS 0-3 TO PORT 4
                  SWAP A       ;EXCHANGE ACC BITS 0-3 AND 4-7
                  MOVD P5,A    ;MOVE ACC BITS 0-3 TO PORT 5
```

MOVP A,@A Move Current Page Data to Accumulator

1 0 1 0	0 0 1 1

The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored following this operation

(PC_{0-7}) ← (A)
(A) ← ((PC))

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the following page.

```
Example: MOV128: MOV A,#128     ;MOVE '128' DEC TO ACC
                 MOVP A,@A      ;CONTENTS OF 129th LOCATION
                                ;IN CURRENT PAGE ARE MOVED TO
                                ;ACC
```

MOVP3 A,@A Move Page 3 Data to Accumulator

1 1 1 0	0 0 1 1

This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

$(PC_{0-7}) \leftarrow (A)$
$(PC_{8-10}) \leftarrow 011$
$(A) \leftarrow ((PC))$

Example: Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

```
TABSCH: MOV A,#0B8H  ;MOVE 'B8' HEX TO ACC (10111000)
        ANL A,#7FH   ;LOGICAL AND ACC TO MASK BIT
                     ;7 (00111000)
        MOVP3 A,@A   ;MOVE CONTENTS OF LOCATION
                     ;'38' HEX IN PAGE 3 TO ACC
                     ;(ASCII '8')
```

Access contents of location in page 3 labelled TAB1. Assume current program location is not in page 3.

```
TABSCH: MOV A,#LOW TAB1 ;ISOLATE BITS 0-7 OF LABEL
                        ;ADDRESS VALUE
        MOVP3 A,@A      ;MOVE CONTENTS OF PAGE 3
                        ;LOCATION LABELED 'TAB1'
                        ;TO ACC
```

MOVX A,@R_r Move External-Data-Memory Contents to Accumulator

1 0 0 0	0 0 0 r

This is a 2-cycle instruction. The contents of the external data memory location addressed by register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

$(A) \leftarrow ((Rr))$ r=0-1

Example: Assume R1 contains 01110110.

```
MAXDM: MOVX A,@R1   ;MOVE CONTENTS OF LOCATION
                    ;118 TO ACC
```

MOVX @R_r,A Move Accumulator Contents to External Data Memory

1001	000r

This is a 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'r'. Register 'r' contents are unaffected.

((Rr)) ← A

Example: Assume R0 contains 11000111

```
MXDMA: MOVX @R0,A      ;MOVE CONTENTS OF ACC TO
                       ;LOCATION 199 IN EXPANDED
                       ;DATA MEMORY
```

NOP The NOP Instruction

0000	0000

No operation is performed. Execution continues with the following instruction.

ORL A,R_r Logical OR Accumulator With Register Mask

0100	1rrr

Data in the accumulator is logically ORed with the mask contained in working register 'r'.

(A) ← (A) OR (Rr) r=0-7

```
Example: ORREG: ORL A,R4      ;'OR' ACC CONTENTS WITH
                              ;MASK IN REG 4
```

ORL A,@R_r Logical OR Accumulator With Memory Mask

0100	000r

Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register 'r', bits 0-5.

(A) ← (A) OR ((Rr)) r=0-1

```
Example: ORDM: MOV R0,#3FH     ;MOVE '3F' HEX TO REG 0
               ORL A,@R0       ;'OR' ACC CONTENTS WITH MASK
                               ;IN LOCATION 63
```

ORL A,#data Logical OR Accumulator With Immediate Mask

0100	0011	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

(A) ← (A) OR data

```
Example: ORID: ORL A,#'X'      ;'OR' ACC CONTENTS WITH MASK
                               ;01011000 (ASCII VALUE OF 'X'
```

ORL BUS,#data Logical OR BUS With Immediate Mask

1 0 0 0	1 0 0 0	$d_7\ d_6\ d_5\ d_4$	$d_3\ d_2\ d_1\ d_0$

This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS,A' instruction.

(BUS) ← (BUS) OR data

```
Example: ORBUS: ORL BUS,#HEXMSK ;'OR' BUS CONTENTS WITH
                                ;MASK EQUAL VALUE OF SYMBOL
                                ;'HEXMSK'
```

ORL Pp, #data Logical OR Port 1 or 2 With Immediate Mask

1 0 0 0	1 0 p p	$d_7\ d_6\ d_5\ d_4$	$d_3\ d_2\ d_1\ d_0$

This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.

(Pp) ← (Pp) OR data p=1-2

```
Example: ORP1: ORL P1, #0FFH  ;'OR' PORT 1 CONTENTS WITH
                              ;MASK 'FF' HEX ( SET PORT 1
                              ;TO ALL ONES)
```

ORLD Pp,A Logical OR Port 4-7 With Accumulator Mask

1 0 0 0	1 1 p p

Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

(Pp) ← (Pp) OR (A_{0-3}) p=4-7

```
Example: ORP7: ORLD P7,A  ;'OR' PORT 7 CONTENTS
                          ;WITH ACC BITS 0-3
```

OUTL BUS,A Output Accumulator Data to BUS

0 0 0 0	0 0 1 0

Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.

(BUS) ← (A)

```
Example: OUTLBP: OUTL BUS,A  ;OUTPUT ACC CONTENTS TO BUS
```

OUTL Pp,A Output Accumulator Data to Port 1 or 2

0 0 1 1	1 0 p p

Data residing in the accumulator is transferred (written) to port 'p' and latched.

(Pp) ← (A) p=1-2

```
Example: OUTLP: MOV A,R7     ;MOVE REG 7 CONTENTS TO ACC
                OUTL P2,A    ;OUTPUT ACC CONTENTS TO PORT 2
                MOV A,R6     ;MOVE REG 6 CONTENTS TO ACC
                OUTL P1,A    ;OUTPUT ACC CONTENTS TO PORT 1
```

RET Return Without PSW Restore

1 0 0 0	0 0 1 1

This is a 2-cycle instruction. The stack pointer (PSW bits 0-2) is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored.

(SP) ← (SP)-1
(PC) ← ((SP))

RETR Return With PSW Restore

1 0 0 1	0 0 1 1

This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine.

(SP) ← (SP)-1
(PC) ← ((SP))
(PSW 4-7) ← ((SP))

RL A Rotate Left Without Carry

1 1 1 0	0 1 1 1

The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.

(AN+1) ← (An)
(A0) ← (A7) n=0-6

```
Example: Assume accumulator contains 10110001.
         RLNC: RL A          ;NEW ACC CONTENTS ARE 01100011.
```

SEL MB0 Select Memory Bank 0

1 1 1 0	0 1 0 1

PC bit 11 is set to zero on next branch instruction. All references to program memory addresses fall within the range 0-2047.

(DBF) ← 0

Example: Assume program counter contains 834 Hex and the carry bit is set.

```
SEL MBO          ;SELECT MEMORY BANK 0
JC $+20          ;IF C=1, JUMP TO LOCATION
                 ;48 HEX
```

SEL MB1 Select Memory Bank 1

1 1 1 1	0 1 0 1

PC bit 11 is set to one on next branch instruction. All references to program memory addresses fall within the range 2048-4095.

(DBF) ← 1

SEL RB0 Select Register Bank 0

1 1 0 0	0 1 0 1

PSW bit 4 is set to zero. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

(BS) ← 0

SEL RB1 Select Register Bank 1

1 1 0 1	0 1 0 1

PSW bit 4 is set to one. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

(BS) ← 1

Example: Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

```
LOC3: JNI INIT    ;JUMP TO ROUTINE 'INIT' IF
                  ;INTERRUPT INPUT IS ZERO
```

STRT CNT Start Event Counter

0 1 0 0	0 1 0 1

The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

Example: Initialize and start event counter. Assume overflow is desired with first T1 input.

```
STARTC: EN TCNTI      ;ENABLE COUNTER INTERRUPT
        MOV A,#0FFH   ;MOVE 'FF' HEX (ONES) TO
                      ;ACC
        MOV T,A       ;MOVE ONES TO COUNTER
        STRT CNT      ;ENABLE TIAS COUNTER
                      ;INPUT AND START
```

STRT T Start Timer

0 1 0 1	0 1 0 1

Timer accumulation is initiated in the timer-register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

Example: Initialize and start timer.

```
STARTT: CLR A         ;CLEAR ACC-TO ZEROS
        MOV T,A       ;MOVE ZEROS TO TIMER
        EN TCNTI      ;ENABLE TIMER INTERRUPT
        STRT T        ;START TIMER
```

SWAP A Swap Nibbles Within Accumulator

0 1 0 0	0 1 1 1

Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

$(A_{4-7}) \rightleftarrows (A_{0-3})$

Example: Pack bits 0-3 of locations 50-51 into location 50.

```
PCKDIG: MOV R0, #50   ;MOVE '50' DEC TO REG 0
        MOV R1, #51   ;MOVE '51' DEC TO REG 1
        XCHD A,@R0    ;EXCHANGE BITS 0-3 OF ACC
                      ;AND LOCATION 50
        SWAP A        ;SWAP BITS 0-3 AND 4-7 OF ACC
        XCHD A,@R1    ;EXCHANGE BITS 0-3 OF ACC AND
                      ;LOCATION 51
        MOV @R0,A     ;MOVE CONTENTS OF ACC TO
                      ;LOCATION 50
```

XCH A,R_r Exchange Accumulator-Register Contents

0 0 1 0	1 r r r

The contents of the accumulator and the contents of working register 'r' are exchanged.

(A) ⇄ (Rr) r=0-7

Example: Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7      ;EXCHANGE CONTENTS OF REG 7
                      ;AND ACC
        MOV A, PSW    ;MOVE PSW CONTENTS TO ACC
        XCH A,R7      ;EXCHANGE CONTENTS OF REG 7
                      ;AND ACC AGAIN
```

XCH A,@R_r Exchange Accumulator and Data Memory Contents

0 0 1 0	0 0 0 r

The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5 of register 'r' are exchanged. Register 'r' contents are unaffected.

(A) ⇄ ((Rr)) r=0-1

Example: Decrement contents of location 52.

```
DEC52: MOV R0,#52     ;MOVE '52' DEC TO ADDRESS
                      ;REG 0
       XCH A,@R0      ;EXCHANGE CONTENTS OF ACC
                      ;AND LOCATION 52
       DEC A          ;DECREMENT ACC CONTENTS
       XCH A,@R0      ;EXCHANGE CONTENTS OF ACC
                      ;AND LOCATION 52 AGAIN
```

XCHD A,@R_r Exchange Accumulator and Data Memory 4-Bit Data

0 0 1 1	0 0 0 r

This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5 of register 'r'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'r' are unaffected.

(A_{0-3}) ⇄ ((Rr0-3)) r=0-1

Example: Assume program counter contents have been stacked in locations 22-23.

```
XCHNIB: MOV R0,#23      ;MOVE '23' DEC TO REG 0
        CLR A           ;CLEAR ACC TO ZEROS
        XCHD A,@R0      ;EXCHANGE BITS 0-3 OF ACC
                        ;AND LOCATION 23 (BITS 8-11
                        ;OF PC ARE ZEROED, ADDRESS
                        ;REFERS TO PAGE 0)
```

XRL A,R_r Logical XOR Accumulator With Register Mask

1 1 0 1	1 r r r

Data in the accumulator in EXCLUSIVE ORed with the mask contained in working register 'r'.

(A) ← (A) XOR (Rr) r=0-7

```
Example: XORREG: XRL A,R5      ;XOR' ACC CONTENTS WITH
                               ;MASK IN REG 5
```

XRL A,@R_r Logical XOR Accumulator With Memory Mask

1 1 0 1	0 0 0 r

Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'r', bits 0-5.

(A) ← (A) XOR ((Rr)) r=0-1

```
Example: XORDM: MOV R1, #20H   ;MOVE '20' HEX TO REG 1
                XRL A,@R1      ;'XOR' ACC CONTENTS WITH MASK
                               ;IN LOCATION 32
```

XRL A,#data Logical XOR Accumulator With Immediate Mask

1 1 0 1	0 0 1 1	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified-mask.

(A) ← (A) XOR data

```
Example: XORID: XOR A,#HEXTEN;XOR CONTENTS OF ACC WITH
                             ;MASK EQUAL VALUE OF SYMBOL
                             ;'HEXTEN'
```

A 1010
B 1011
C 1100
D 1101