กุญแจ การ์ดอินฟาเรด

INFARRED GARD KEY

โดย

นายยงกฤต ครองสวัสดิ์กุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2534

ปีการศึกษา 2534

คู่มือจัดการคีย์ระบบดิจิตอลอินฟาเรด

Infarred card key

โดย

นายยงกฤต ครองสวัสดิ์กุล

อาจารย์ที่ปรึกษา

ผศ. ดร. ภนก เจนจิระพงศ์เวช

ปริญญานิพนธ์ปีการศึกษา 2534

สาขาวิชาโปรแกรมมาตร

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

เรื่อง กุญแจ การ์ดอินฟาเรด (INFARRED CARD KEY)

ผู้จัดทำ

นายพงศกร ครองสวัสดิ์กุล

อาจารย์ที่ปรึกษา

(ผศ. ดร. กมล เจนจิระพงศ์เวช)

ปัจจุบันเทคโนโลยีทางด้าน ระบบฝังตัวได้เริ่มแพร่หลายในเมืองไทยในหลาย ๆ รูปแบบ

โครงงานนี้ก็เป็นโครงงาน ของระบบฝังตัวอีก รูปแบบหนึ่ง ซึ่งเกี่ยวกับ อินฟาเรดและการใช้ไมโครคอนโทรลเลอร์ ทำงานร่วมกันโดยแบ่งลักษณะงานออกเป็น 2 ส่วน คือ Hardware และ Software

สำหรับการทดลอง ศึกษาของผู้จัดทำนั้นปรากฏว่า การทำงานของระบบ ทั้งหมด เช่น การอ่านระหัส การควบคุมฝังตัว เป็นไปอย่างแม่นยำและเที่ยงตรงปราศจากข้อผิดพลาดแม้แต่เพียงเล็กน้อย

สุดท้ายนี้ ขอขอบพระคุณ อาจารย์ ผศ. ดร. ณนท เจษฎิระพงศ์เวช ที่ได้กรุณาให้คำแนะนำในเรื่องต่าง ๆ เป็นอย่างดี และผู้จัดทำหวังเป็นอย่างยิ่งว่า ปริญญานิพนธ์ฉบับนี้ คงเป็นประโยชน์แก่ผู้อ่านบ้าง

ผู้จัดทำ

กุญแจการ์ดอินฟาเรด

ยงยุทธ กรองสวัสดิ์กุล

ผศ.กร. ดนย เจนจิระพงศ์เวช

ปีการศึกษา 2534

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ เสนอการออกแบบระบบกุญแจการ์ดอินฟาเรดโดยการใช้ลูกกุญแจเป็นแผ่นฟิลม์ภายใน แผ่นฟิลม์ บรรจุระหัสตามต้องการ การ์ดสอดเข้าไปในรูกุญแจ ภายในรูกุญแจก็จะอ่านระหัส ถ้าตรงกับหมายเลขที่ต้องการ ก็จะมีสัญญาณ ไปบังคับ รีเลย์ให้เปิดประตูได้

Yongkit Prongsawadsul

Asistant Professor Kanok Jenchirapongwet

1991

ABSTRACT

The thesis describes the hardware desigh of key card. By using the film and setting the code. When it correct code the door will open. This system compose of Micro Controller single shift No. MCS-48.

สารบัญ

บทที่ 1

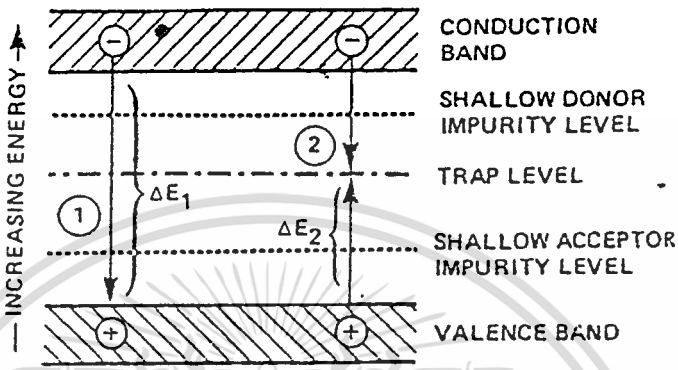
INFRARED LED

INFRARED LED มีคุณสมบัติ หลายประการ คือ

1. ใช้กระแส และ Voltage มีค่าน้อยมากในการผลิตคลื่น
2. เราสามารถควบคุมการทำงาน LED ให้มีค่าแน่นอนได้ โดยพิจารณาจากกราฟของ factor ที่มาเกี่ยวข้อง
3. มี Speed Response สูง

ลักษณะ Infrared LED ที่เหมือนกับ LED โดยทั่วไป คือ จะเป็นสารกึ่งตัวนำที่ถูก dope โดยสารที่มี Valence electron ต่างกัน โดยสารที่มี electron มากกว่าปกติ จะเป็นสาร N-type และสารที่มี electron น้อยกว่าปกติ จะเป็น P-type เมื่อนำสารทั้ง สองชนิดมาต่อกัน จะทำให้มีการ share electron กันขึ้น ทำให้สารทั้งสองมีความต่างศักดากันขึ้นเรียกว่า Potential Barrier (Eb) ซึ่งมีค่าน้อยกว่าขนาดของ Eneygy gap ดังนั้น electron จากด้าน N-type จะเคลื่อนที่มายังด้าน P-type จะต้องใช้พลังงานค่าหนึ่ง

เมื่อเราป้อน Supply Voltage ให้กับ LED จะมี electron เคลื่อนที่จากด้าน N-type มารวมกับ hole ด้าน P-type ซึ่งการรวมตัวกันนี้ มีอยู่ 2 ลักษณะ คือ Radiative process ซึ่งเมื่อรวมตัวกันแล้ว จะมีพลังงานบางส่วนสูญหายไปจาก electron- pair และกระจายออกมาเป็น photon ถ้าเป็นแบบ Non-Radiative Process พลังงานจะแสดงออกมาในรูป Phonon or heat



รูปที่ ๒ Recombination Processes in the P-N Junction

Radiative Process มี 2 แบบ คือ

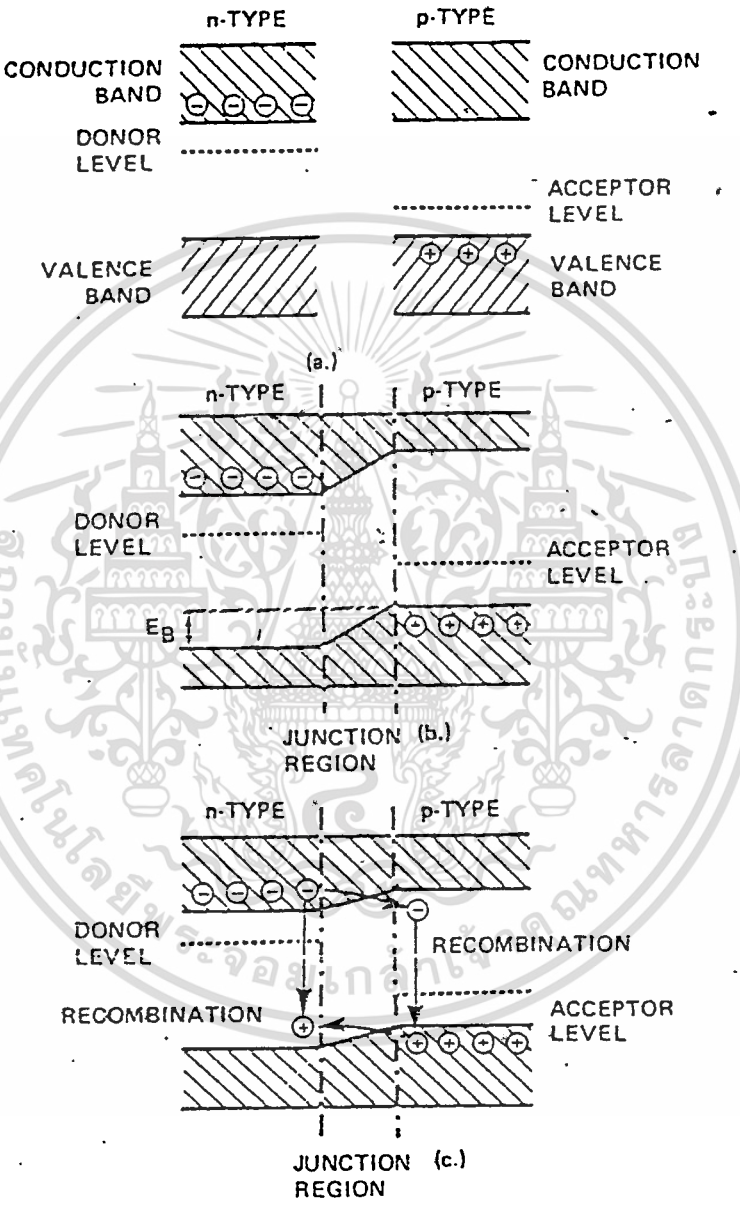
1. เกิดจาก electron and hole ส่วนที่อยู่ใกล้ ๆ ลับ junction ซึ่งจะให้ photon energy ประมาณค่าเท่ากับค่า vand Gap energy ซึ่งมีค่าค่อนข้างสูง

2. เป็นการที่ electron and hole จะเคลื่อนมารวมตัวกันที่บริเวณ trap level photon ที่ได้จะมีพลังงานเท่ากับ energy gap ลบด้วยค่า Binding energy และจาก Process ทั้งสองจะได้ photon ออกมา ซึ่งจะมีค่า wave length ตามสมการข้างล่าง

$$\lambda = \frac{1240}{\triangle E}$$

$\triangle E$ คือ ค่าความต่างของพลังงานหน่วยเป็น electron Volt

จากที่กล่าวมาแล้วจะเห็นได้ว่า ถ้าใช้สารต่างชนิดกันค่า $\triangle E$ จะมีค่าแตกต่างกันออกไปซึ่งเราจะสามารถดูได้จากตารางในรูปที่ 3

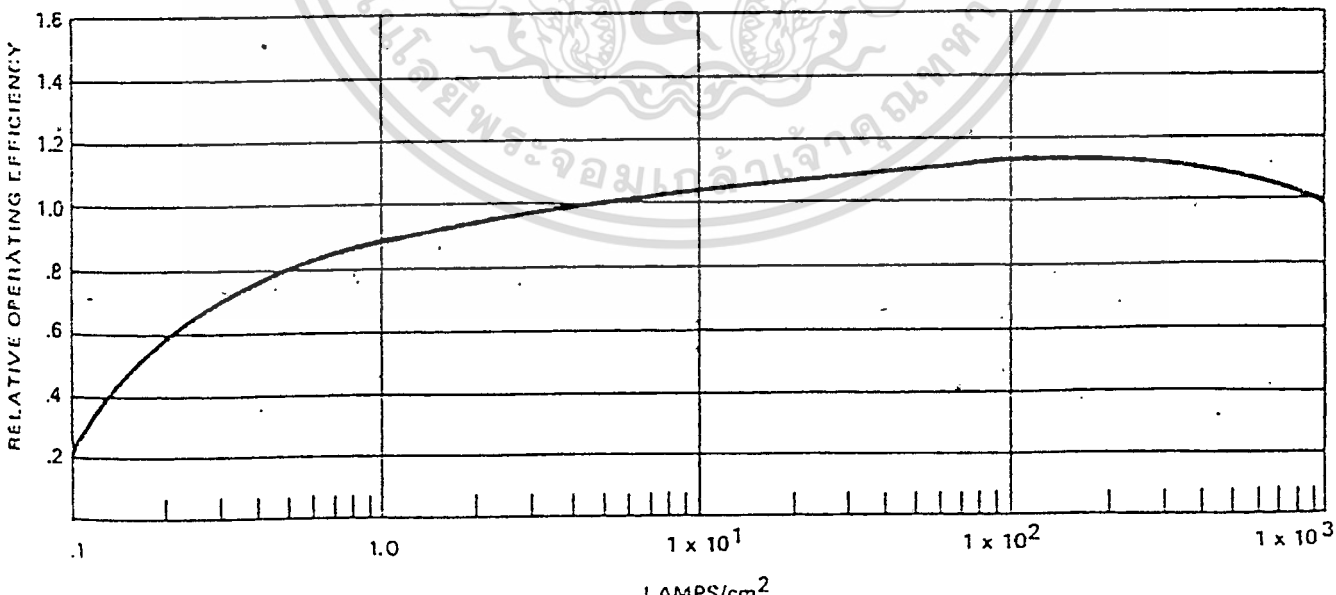


รูปที่ ๑ Schematic Representation of the P-N Junction

MATERIAL	BAND GAP ENERGY	EMISSION λ nm	TRANSITION TYPE
	eV		
Ge	0.66	1880	INDIRECT
Si	1.09	1140	INDIRECT
GaAs	1.43	910	DIRECT
GaP	2.24	560	INDIRECT
$GaAs_{60}P_{40}$	1.91	650	DIRECT
Al Sb	1.60	775	INDIRECT
In Sb	0.18	6900	DIRECT
Si C	2.2-3.0	563-413	INDIRECT

รูปที่ ๓ Some of the Materials Available for LED Devices

เป็นการแสดงถึง LED ชนิดต่าง ๆ ซึ่งใช้ PHOTON ที่มี WAVELENGTH ค่าต่าง ๆ กันไป และจะเห็นได้ว่า LED ที่สร้างจาก gallium Arsenide (GA AS) จะให้คลื่นในช่วง Infrared ซึ่งมีค่าประมาณ 900 nm.



รูปที่ ๔ Normalized Operating Efficiency vs. Current Density for an LED

จากรูป ที่ 4 เป็นรูปกราฟแสดงความสัมพันธ์ระหว่างประสิทธิภาพ ในการทำงาน ของ LED กับค่า current density และจากกราฟ จะเห็นได้ว่า led แม้ว่าจะมี current density เปลี่ยนแปลงไปเป็นจำนวนมาก แต่ประสิทธิภาพในการทำงานก็ยังใกล้เคียงกัน

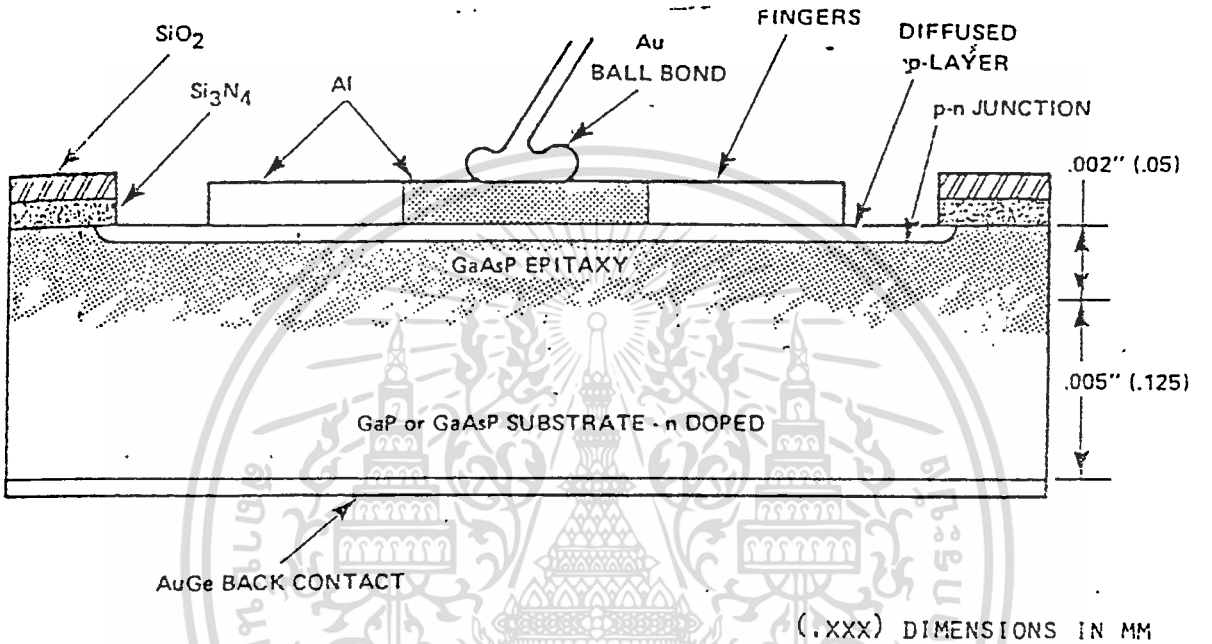


Figure A. CROSS SECTION OF AN LED

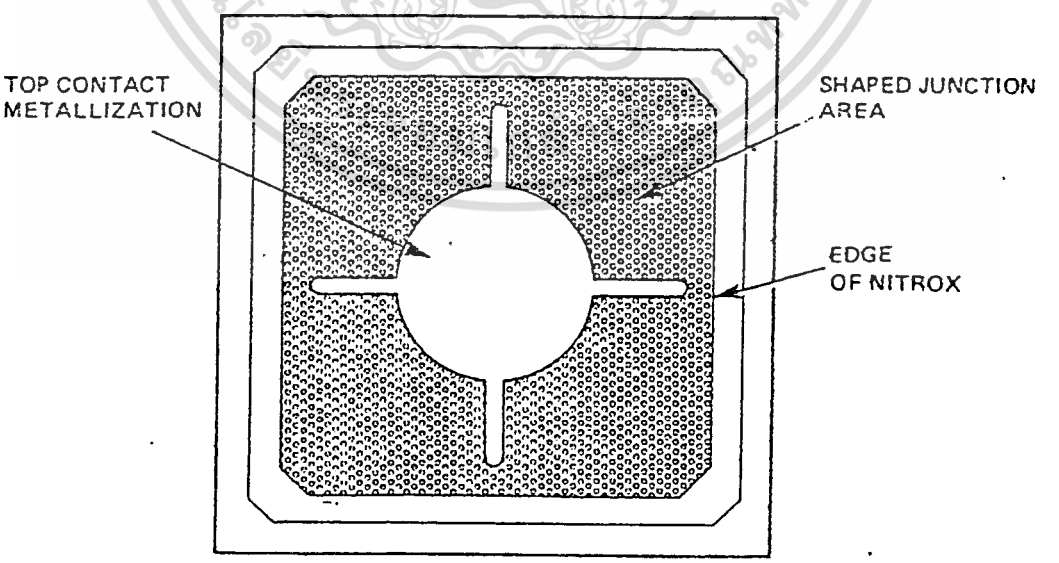
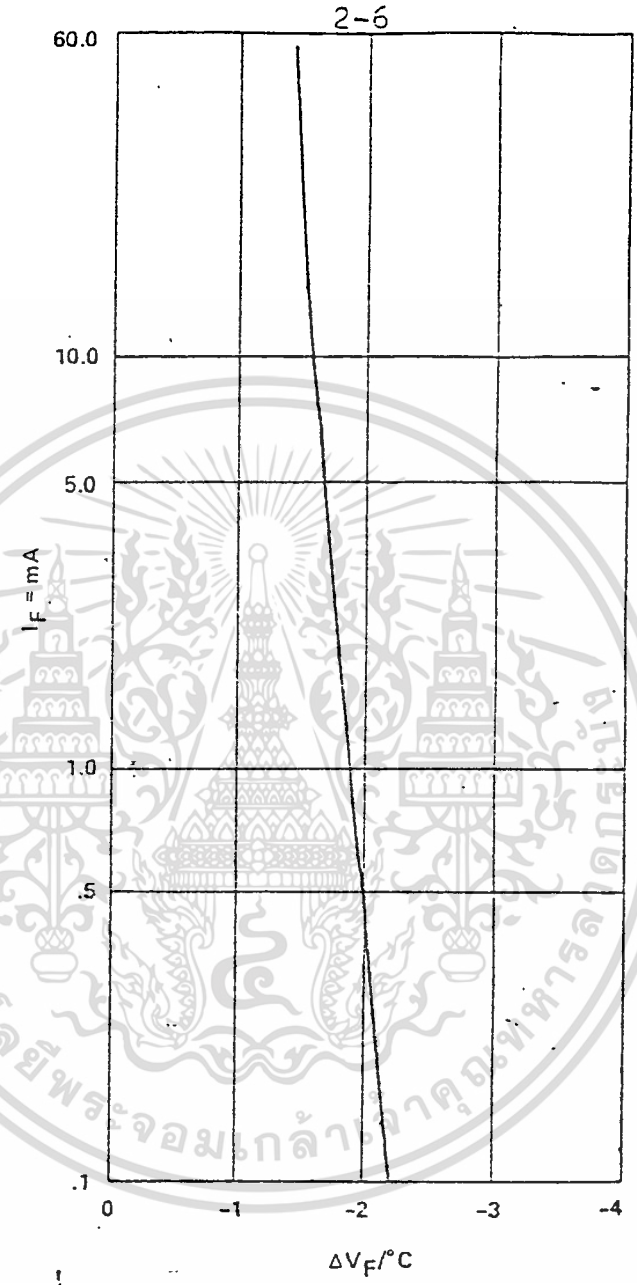


Figure B. PLAN VIEW OF AN LED

จากรูปที่ 5 เป็นรูปที่แสดงถึงโครงสร้างภายในของ LED



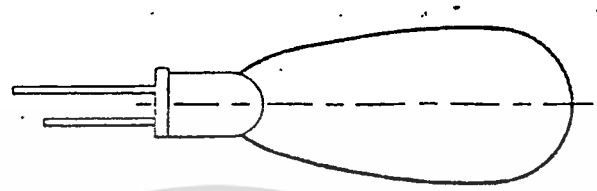
รูปที่ 5 Temperature Coefficient of Forward Voltage as a Function of Forward Current

รูปที่ 6 เป็นการแสดงถึง forward voltage/current and temperature

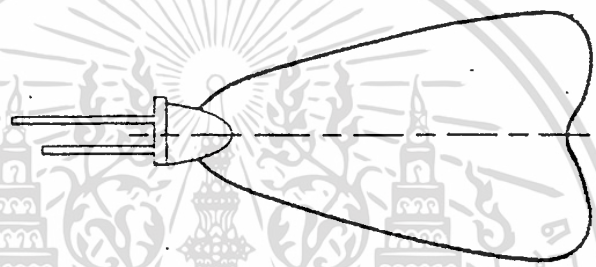
จากรูปข้างบนสามารถแสดง ความสัมพันธ์ด้วยสมการ

$$I_F = I_o \exp\left(\frac{q V_F}{n k T}\right)$$

จากที่เราได้ทราบมาแล้วว่า ค่า wavelength นั้นได้จากสูตร $\lambda = \frac{1240}{\triangle E}$ เพราะฉะนั้น เมื่อ LED มีอุณหภูมิเปลี่ยนแปลงไป จะทำให้ $\triangle$ E เปลี่ยนแปลงด้วย ทำให้ค่า wavelength เปลี่ยนไป



T-1 3/4 LAMP WITH SPHERICAL DOME LENS



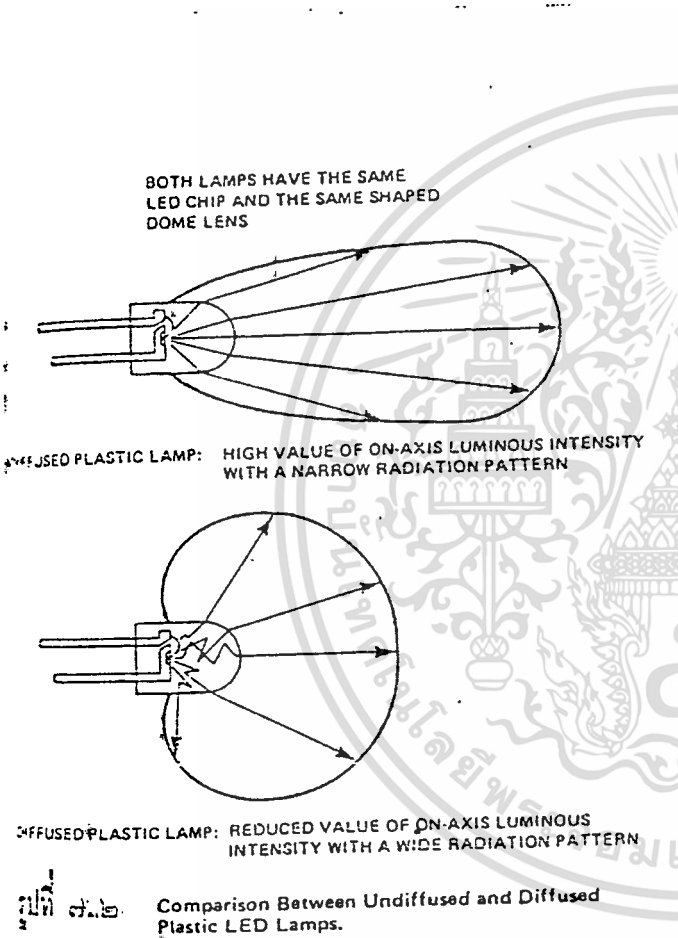
T-1 3/4 LOW PROFILE LAMP
WITH AN ASPHERIC DOME LENS

รูปที่ ๗.๑ Radiation Patterns for Undiffused Lamps with Spherical and Aspheric Dome Lenses.

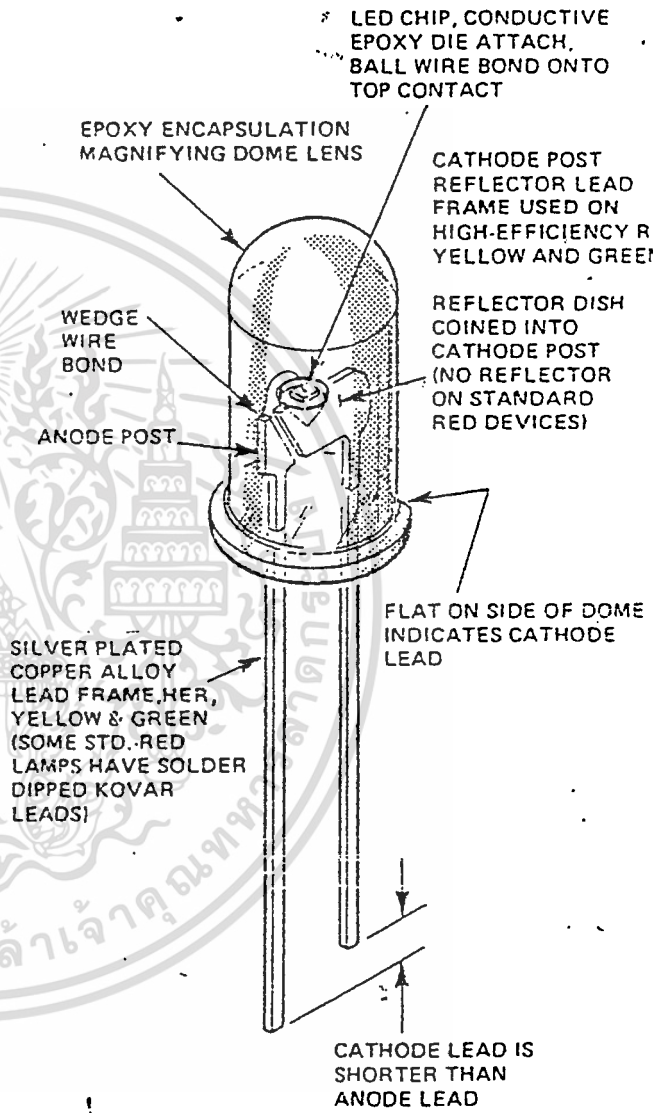
จากรูปที่ 7.1 แสดงถึงรูปร่างของ dome lens ซึ่งแตกต่างกัน จะให้ Radiation Pattern ต่าง ๆ กัน แต่ใน Plastic LED แล้ว SPHERICAL DOME LENS จะให้ pattern เหมาะสมกับการส่งแบบ point to point ซึ่งจุดทั้งสองจุดจะถูกตรึงกับที่ค่อนข้างแน่นอน และจะส่งได้ไกลกว่าแบบอื่น ส่วนแบบ ASPHERIC DOME LENS เหมาะสำหรับครอบคลุมพื้นที่ แต่ส่งได้ใกล้กว่า เช่น Remote control

นอกจากนี้ แม้ว่า LED จะเป็นชนิดเดียวกัน และมี Dome LENS เหมือนกัน แต่ถ้าพลาสติคที่ใช้ทำต่างชนิดกันก็จะให้ผลต่อการส่งคลื่นออกไป ทำให้ pattern ไม่เหมือนกัน เช่น ดังรูปที่ 7.2 แสดงถึง LED สองตัวที่มีขนาดเท่ากัน และเล็นชนิดเดียวกัน แต่ใช้

diffused plastic อันหนึ่งอีกอันใช้ undiffused plastic จะให้ pattern ออกมาต่างกัน

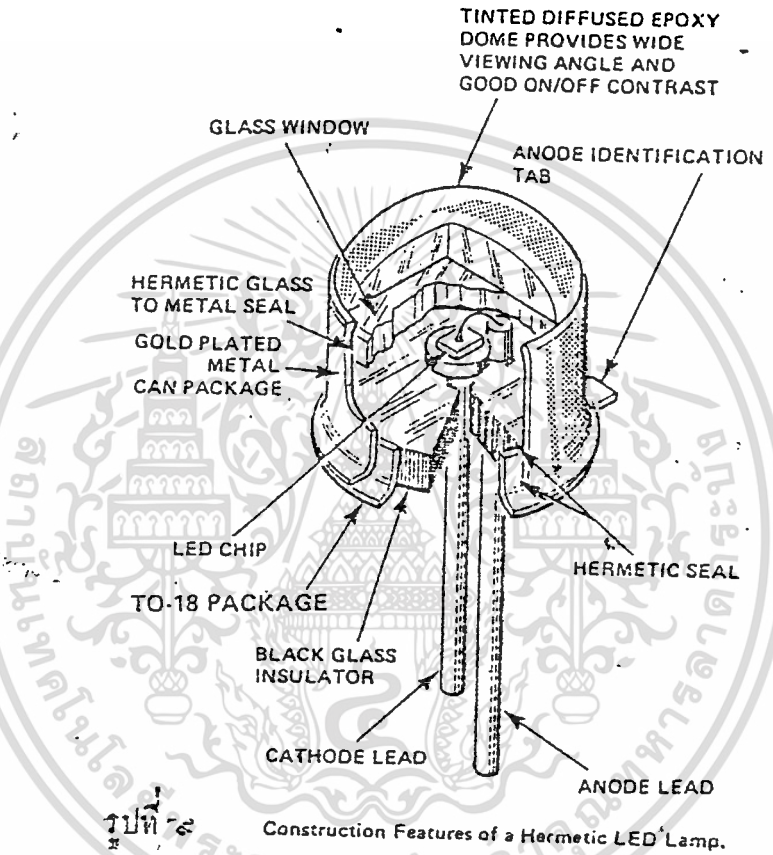


รูปที่ ๗.๒ Comparison Between Undiffused and Diffused Plastic LED Lamps.



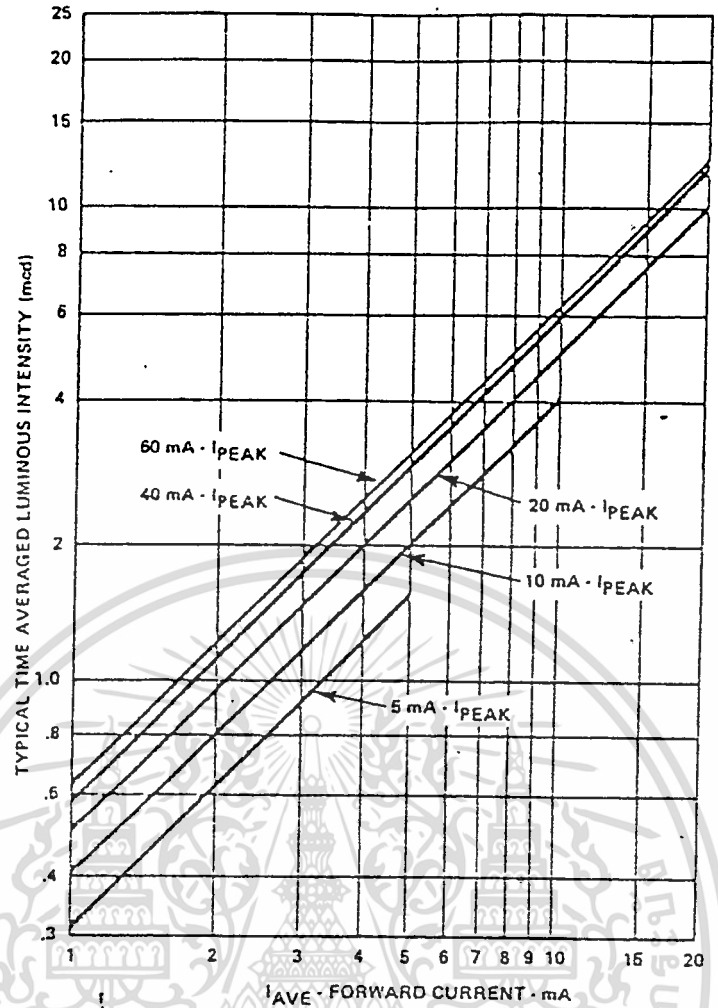
รูปที่ ๘ Construction Features of a T-1 3/4 Plastic LED Lamp.

รูปที่ 8 เป็นรูปที่แสดงถึงส่วนประกอบต่าง ๆ ของ plastic LED

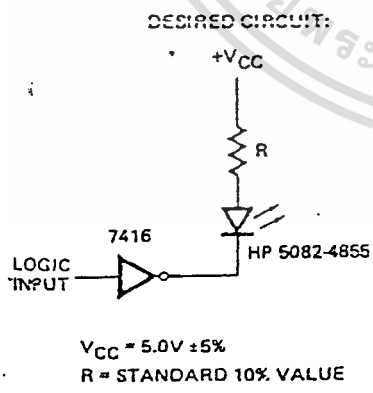


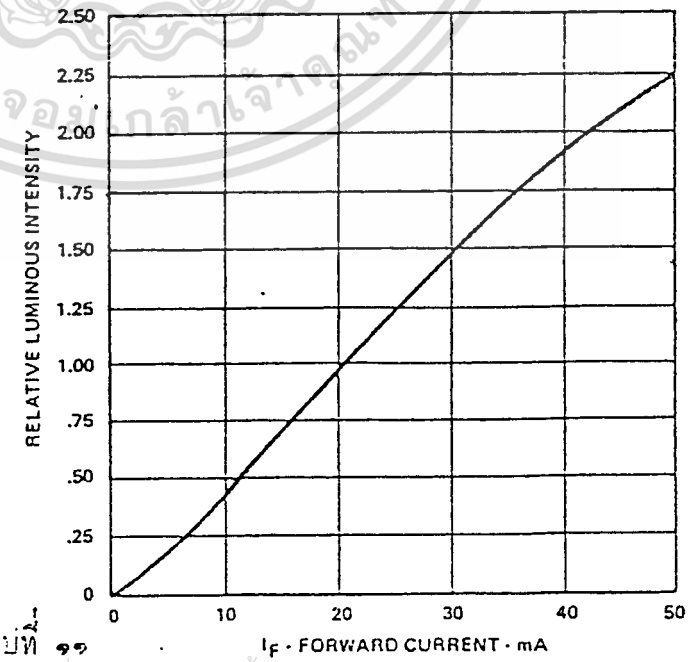
รูปที่ ๘ Construction Features of a Hermetic LED Lamp.

รูปที่ 9 เป็นรูปของ Hermetic LED ซึ่งเป็น LED ที่มีคุณภาพดีกว่าแบบแรก



รูปที่ ๑๐ Typical Time Averaged Luminous Intensity vs. Average Current for a High Efficiency Red LED.

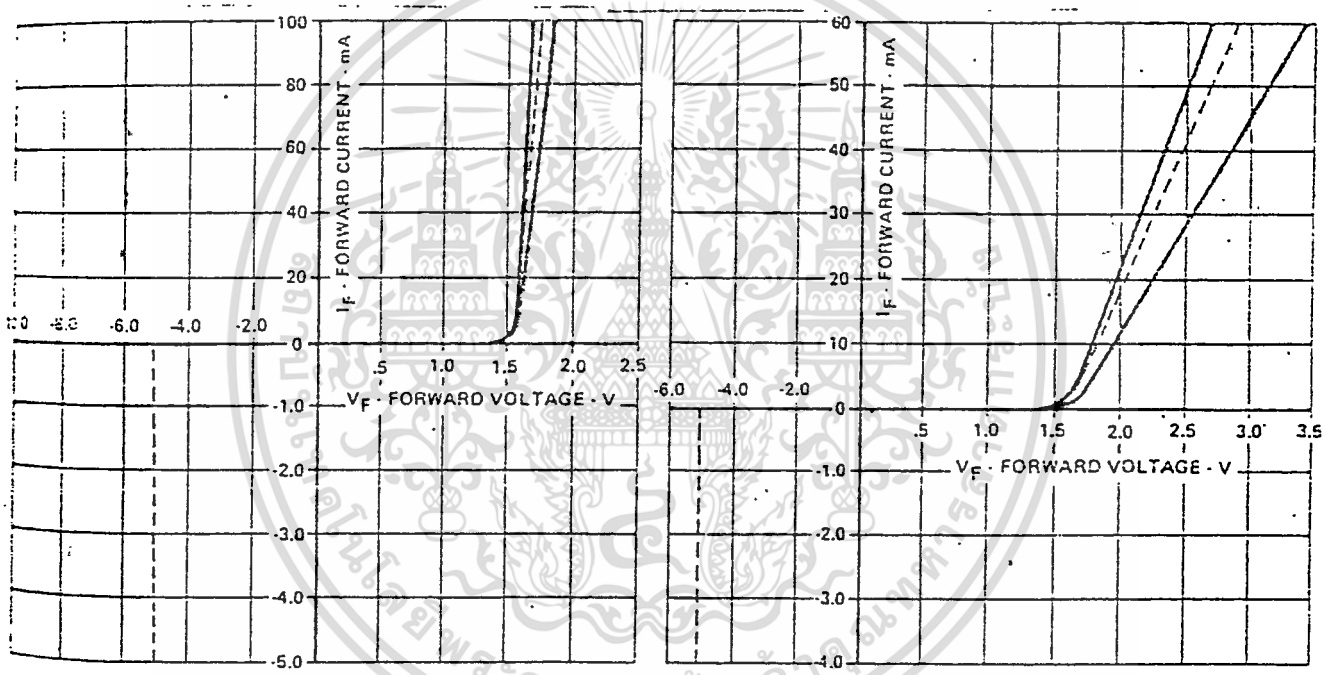




รูปที่ ๑๑

จาก 2 รูปดังกล่าว เป็นรูปที่แสดงถึงความสัมพันธ์ระหว่างกระแสกับความเข้มของ photon ที่เปล่งออกไป โดยรูปที่ 10 นั้น เป็นค่าความสัมพันธ์ โดยที่ลักษณะกระแสไม่สม่ำเสมอมีลักษณะเป็นคลื่น ดังนั้นค่ากระแส จึงแสดงออกมาเป็นค่า Iaverage and Ipeak

รูปที่ 11 นั้นเป็นลักษณะของกระแสธรรมดาหรือกระแส DC และจากทั้ง 2 ภาพ ทำให้เราทราบว่า ความเข้มของ photon นั้น ขึ้นอยู่กับกระแส



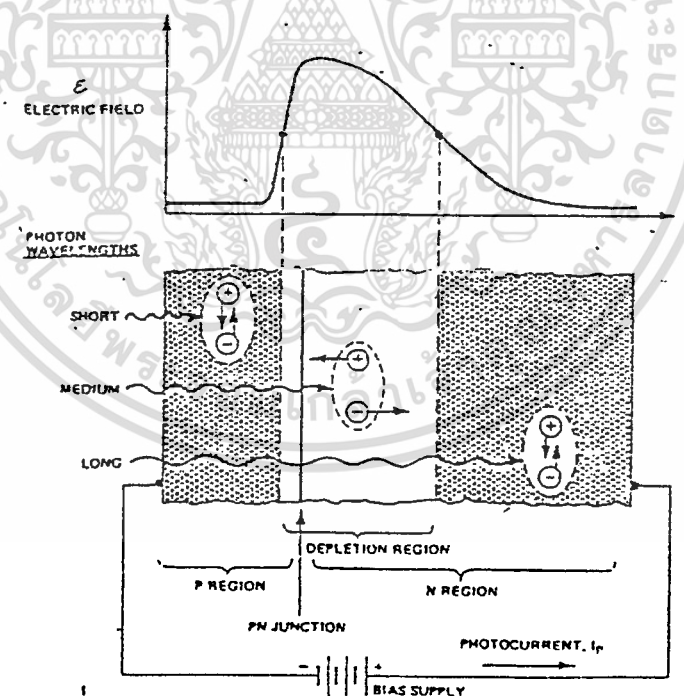
รูปที่ ๑๒ Typical Electrical Characteristics of LED Lamps.

จากกราฟในรูปที่ 12 จะเห็นได้ว่าโดยปกติแล้ว Voltage ที่ตกคร่อม LED จะมีค่าคงที่ แน่นอนมีแต่เพียงกระแสเท่านั้นที่มีการเปลี่ยนแปลงและจากรูปด้านขวาแสดงถึงผลของ High frequency ซึ่งจะส่งผลให้ Voltage มีค่าเปลี่ยนแปลงตามกระแสไปด้วย

Photodiode and Phototransistor (detector)

เมื่อมีการส่งสัญญาณที่เป็น Photon ออกมาแล้ว ทางด้านรับจะต้องรับสัญญาณนั้นแล้วเปลี่ยนกลับมาเป็นสัญญาณทางไฟฟ้าอีกทีหนึ่ง โดยจะต้องตัดสัญญาณ Infrared ที่เป็นสัญญาณพาออกไปโดยจะต้องให้ได้สัญญาณไฟฟ้าเหมือนกับตอนส่งมาด้วย การ detect สัญญาณนี้เราจะใช้ Phototransistor or Photodiode

เมื่อมี photon มากระทบ สารกึ่งตัวนำจะทำให้ photon นั้นถ่ายเทพลังงานให้กับ electron-hole pair ซึ่งจะทำให้ electron and hole แยกตัวออกจากกัน และถ้ามีการ bias ไว้แบบ Reverse bias จะทำให้ electron เคลื่อนที่ไปยัง N-Region และ hole เคลื่อนที่ไปยัง P-Region ดังรูปที่ 1



รูปที่ 1 P-N Photodiode Junction; Diagram of Internal Field Effect on Detection.

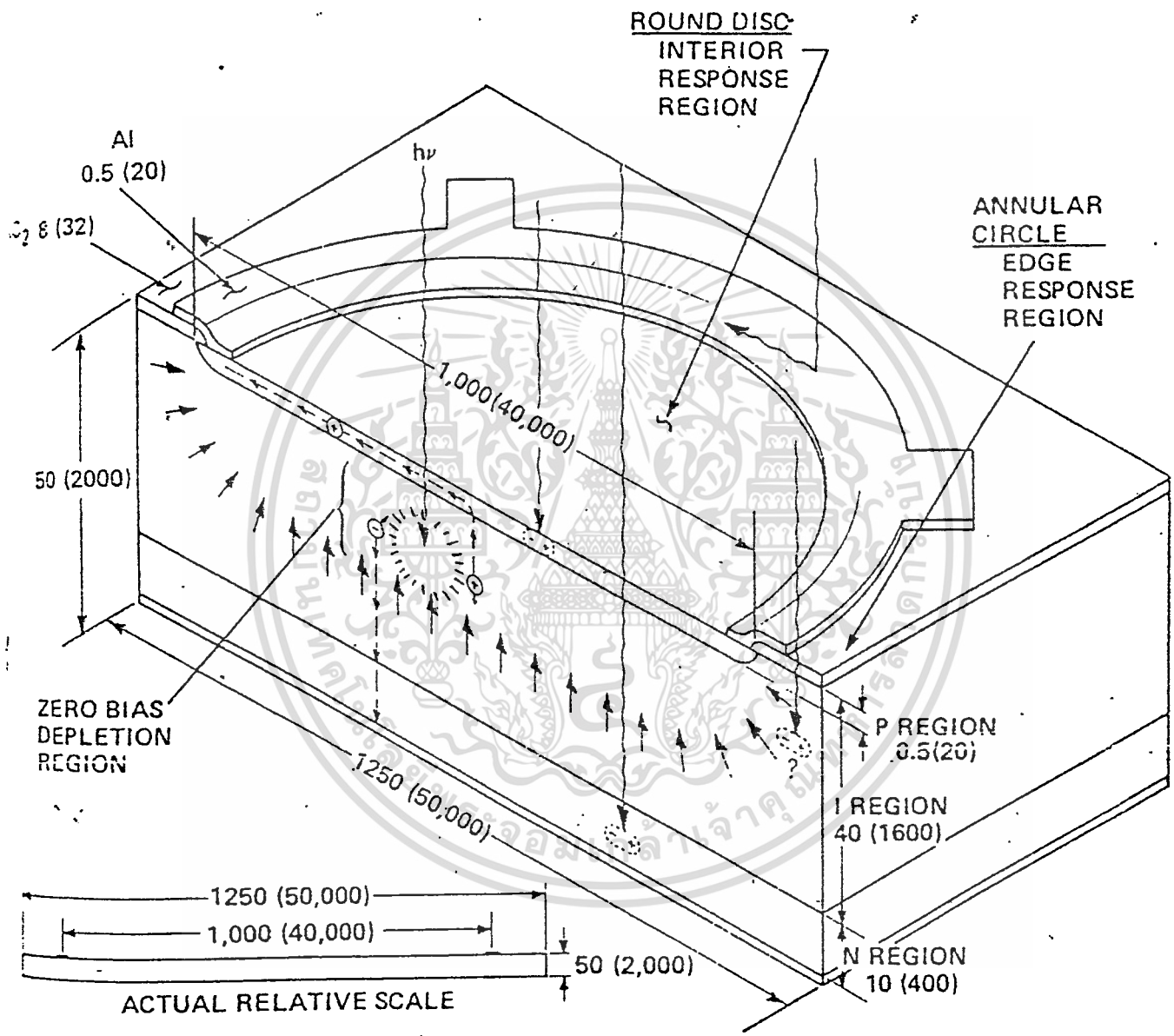
การที่ photon ทำให้ electron-hole pair แยกตัวจากกันนั้น จะเป็นการง่ายขึ้น ถ้า pair นั้นอยู่ภายในสนามไฟฟ้าที่มีค่าสูง จากรูปจะเห็นได้ว่า สนามไฟฟ้าใน PIN Photodiode นี้ มีค่าไม่สม่ำเสมอเท่ากันตลอด โดยจะมีค่าสนามไฟฟ้าในช่อง P and N Region ส่วนในช่วง Depletion Region จะมีค่าสนามไฟฟ้าสูงกว่ามาก เพราะฉะนั้นเราจะต้องให้ photon ส่วนใหญ่ ถ่ายเทพลังงานให้กับช่วง Depletion Region นี้ แต่เนื่องจาก photon ที่มีความถี่สูงจะมีอำนาจการทะลุทะลวงต่ำ และถ้าความถี่ต่ำเกินไปก็จะทะลุผ่านช่วง Depletion ไป ทั้งในช่วงความถี่สูงและต่ำนี้ จะสามารถถ่ายเทพลังงานให้กับ electron-hole pair ในสาร P and ได้ แต่ Hole and electron ก็จะไม่เคลื่อนที่ไปไหน จะรวมตัวกันในบริเวณเดิม ซึ่งไม่ทำให้เกิดกระแสขึ้น

ดังนั้นจะเห็นได้ว่า Photodiode or Phototransistor จะสามารถตอบสนองต่อคลื่นในช่วงหนึ่งเท่านั้น

ช่วง Depletion นี้จะมีอยู่ตลอดเวลา แม้ว่าจะไม่มีการ bias แต่ขนาดของมันจะเปลี่ยนแปลงได้ โดยขึ้นอยู่กับ Voltage ที่ vias และความต้านทาน ก็จะเปลี่ยนแปลงด้วยเช่นกัน โดยถ้าช่อง Depletion กว้างจะมีความต้านทานสูง

ใน transistor นั้น ช่วง Depletion ที่เป็นส่วนรับ photon นั้น เป็นช่วงระหว่างสารของ collector กับสารของ base

รูปที่ 2 เป็นรูปโครงสร้างของ PIN Photodiode ซึ่งเป็น diode ที่มีความต้านทานสูง และมีช่วง depletion กว้างประมาณครึ่งหนึ่งของ I layer และถ้ามี Reverse bias ประมาณ 5 Volt จะทำให้ช่วง Depletion กว้างถึงเกือบทั้งหมดของ I layer (Intrinsic region)



รูปที่ ๒ P-I-N Photodiode; Isometric Cutaway Distorted to Clarify Main Features.

Photodiode Characteristic

Quantum efficiency แสดงคุณสมบัติ การตอบสนองของ Photodiode ซึ่ง photon ตัวหนึ่ง ๆ จะทำให้ electron เคลื่อนที่เกิด Photo current จากค่า flux Responsitioity R_{ϕ} .ซึ่งแสดงค่าของ photo energy ในค่าของอัตราส่วนของ photocurrent กับ Sport flux

โดยสมการ $$R_{\phi} = Nq \frac{X}{1240} = \frac{Ip}{\phi e}$$

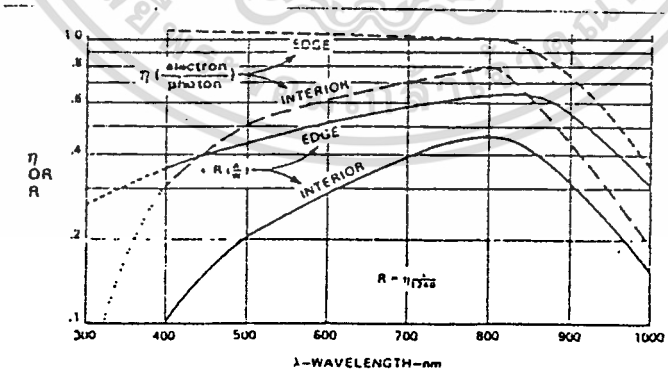
R_{ϕ} = flux Responsitivity

N_q = quantum effciency

= wavelength

Ip = photo current

ϕe = ratio Flux



--- η – QUANTUM EFF. – $\frac{\text{ELECTRON}}{\text{PHOTON}}$ —— R – RESPONSIVITY – $\frac{\text{AMP}}{\text{WATT}}$

รูปที่ ๓ Spectral Response of Interior and Edge Regions of HP P-I-N Photodiode.

จากรูปที่ 3 quantum effieiency ได้แสดงไว้ด้วยเส้นไข่ปลา ส่วน responsivity ได้แสดงด้วยเส้นเต็ม และจากรูป ยังแสดงให้เห็นถึงความแตกต่างของ reponsivity of edge region และ interior Region โดย edge region จะแสดงผลเมื่อมี reverse bias ส่วน Interior region นั้นจะทำงานเป็นอิสระไม่ขึ้นกับ bias และทำงานที่ความยาวคลื่นสั้น ๆ หรือความถี่สูง ถ้าเป็นคลื่นความถี่ต่ำ interior response จะเพิ่มขึ้นเมื่อมี reverse bias

Speed of Response

ความเร็วในการตอบสนองขึ้นอยู่กับพื้นที่ส่วนใดได้รับ photon จากการ radiate และการ bias มีค่ามากน้อยเท่าใด นอกจากนี้ยังต้องคำนึงถึง Ioad Resistance ว่ามีค่าเท่าไรอีกด้วย โดยปกติ ถ้าเป็น Phototransistor or Photodiode ที่มีคุณภาพมาตรฐาน โดยทั่วไป เมื่อได้รับ photon แล้วจะใช้เวลาไม่กี่ pi osecond ในการเริ่มให้กระแสไหล แต่บางที response อาจต้องช้าลง เนื่องจากต้อง charge ประจุให้กับค่า C ภายในก่อน เช่น junction Capacitance or package Capacitance ส่วน rise/fall time constant ขึ้นอยู่กับค่า load resistance ถ้า load resistance มีค่าน้อยความต้านทานภายในตัว diode จะเป็นตัวกำหนด ค่าความต้านทานภายในจะมีค่าประมาณ 50

Low noise

ใน Phototransistor and Photodiode จะเห็นได้ว่า จะมีความต้านทานค่อนข้างสูง และในการกำหนด หรือจำกัด noise นั้นเราจะทำได้ตั้งแต่ที่เป็น dc จนกระทั่งความถี่ 1 OKHZ สมการข้างล่างแสดงถึง noise current

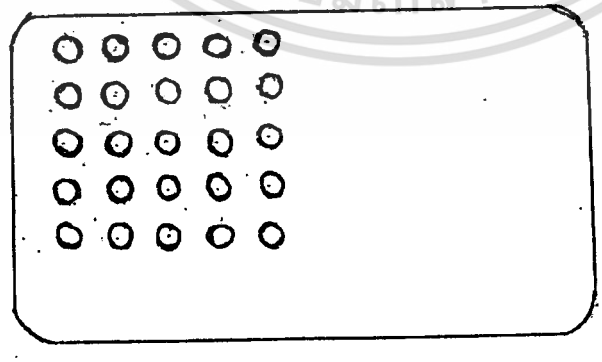
$$\frac{I_{N,SHOT}}{\sqrt{B}} = 2\sqrt{qIdc} = 17.9\sqrt{Idc(nA)}\ \frac{(fa)}{\sqrt{Hg}}$$

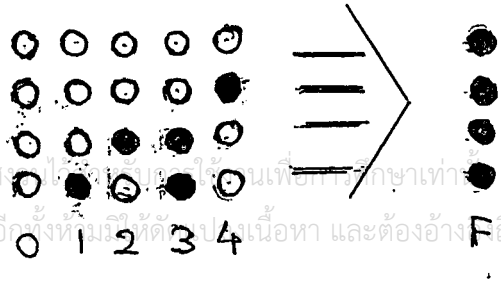
บทที่ 2

2 หลักการทำงานของกุญแจการ์ดอินฟาเรด

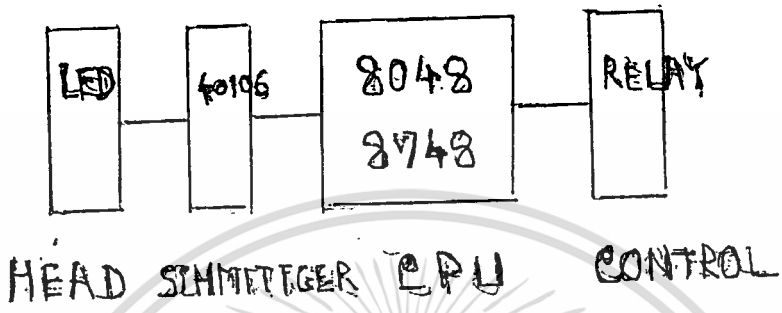
2.1 บัตร

การบรรจุข้อมูล บัตรนี้จะเจาะเป็นรูมาตรฐานถ้าต้องการทำระหัส ก็จะใช้สติกเกอร์สีดำติดไปบนรูเพื่อไม่ให้แสงผ่านได้ตามต้องการ เช่น 0 ก็จะไม่ปิดเลย แสงผ่านได้หมด แต่ถ้าเป็น F จะต้องปิดหมด 4 รู เป็นวิธีการของเลขฐาน 16 ตามรูป





2.2 รูปแบบการอ่านข้อมูล

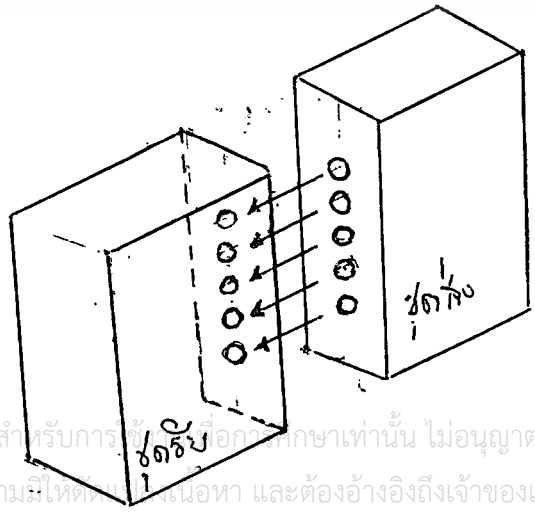


Block Diagram

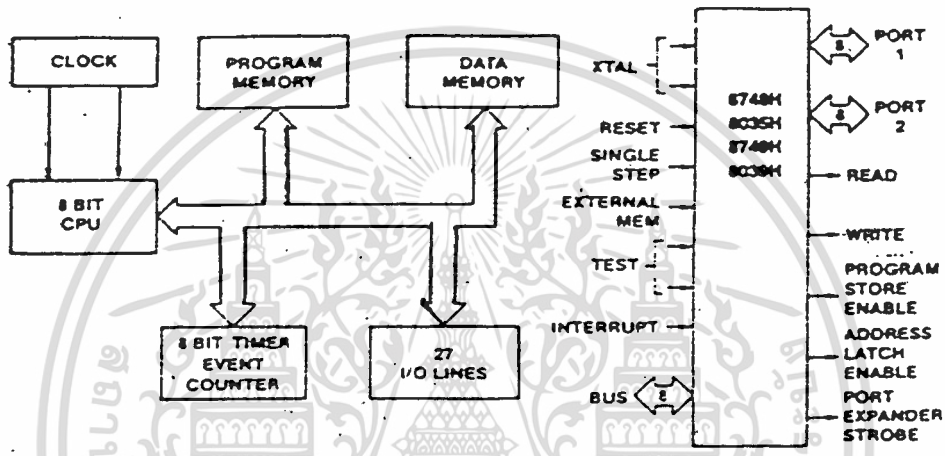
หัวอ่าน หัวอ่านบัตรประกอบด้วย

อินฟาเรด LED ซึ่งมีทั้งตัวส่งและตัวรับ ตั้งให้หัวตรงกัน เพื่อให้การรับและการส่งลำแสง เต็มที่ ในการอ่านข้อมูลในขณะที่มีการสอดบัตรเข้าไปในหัวอ่านนี้ รูที่มีการปิดไว้แสงจะไปไม่ได้ รูที่ไม่มีการปิดแสงจะไปได้ ซึ่งหัวอ่านนี้จะทำหน้าที่อ่านข้อมูลทีละตัวอักษร จนครบหมดเป็น สัญญาณเลขระหัสฐาน 16 สัญญาณ ที่อ่านได้ออกมานี้จะไม่สวยงามจึงต้องไปผ่านวงจร Schmittiger เพื่อทำให้สัญญาณ สวยงามเสียก่อน จากนั้นจึงเข้าไปยัง

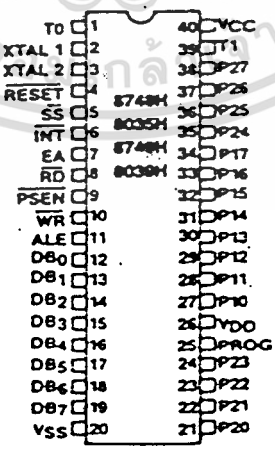
ลักษณะหัวอ่านดังรูป



ไมโครคอนโทรลเลอร์

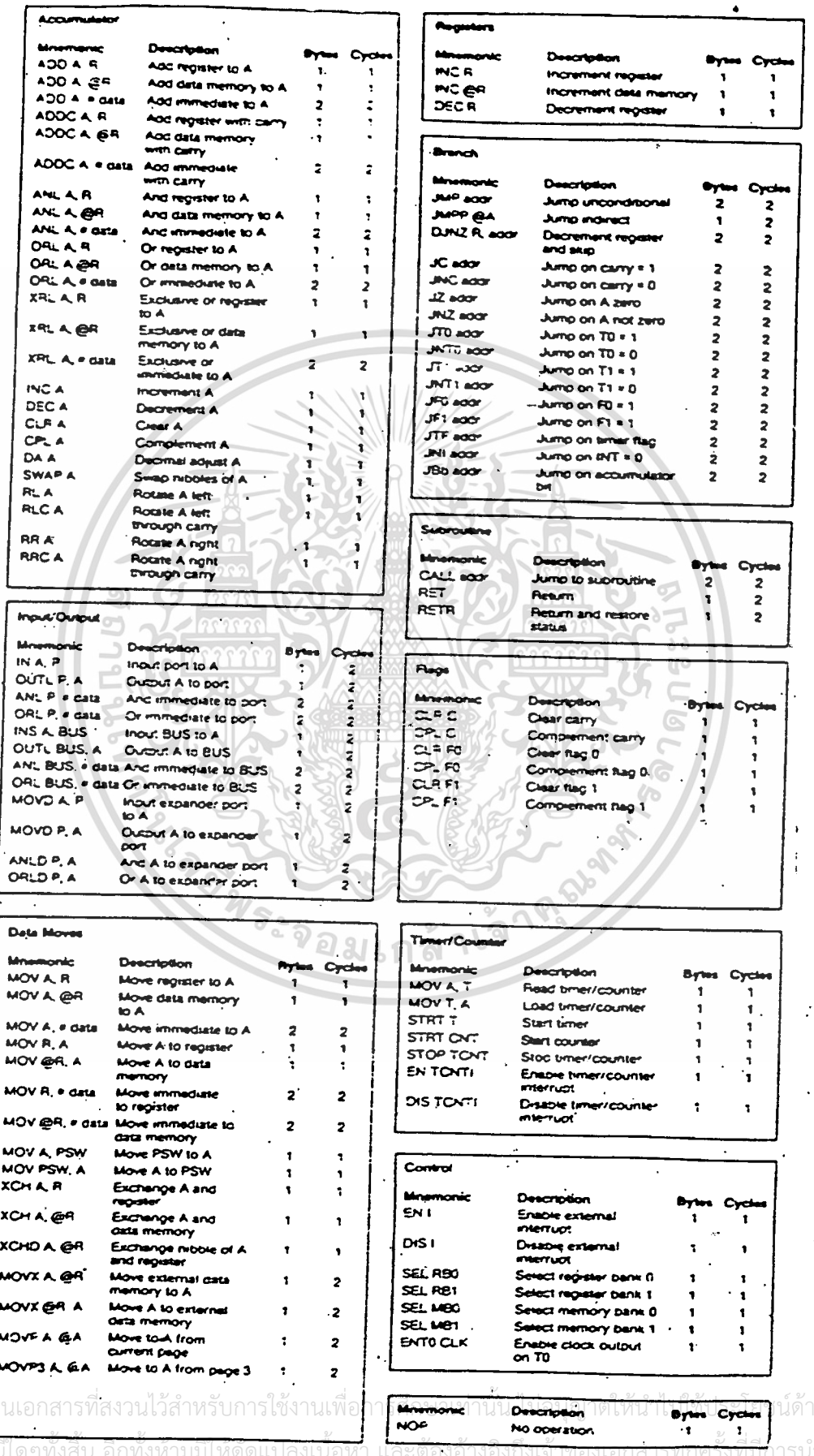


รูปที่ 1 บล็อกไดอะแกรมโครงสร้างของ MCS-48



รูปที่ 2 ลักษณะการจัดตำแหน่งขา

รูปที่ 3 ชุดคำสั่งของ MCS-48



Accumulator

Mnemonic	Description	Bytes	Cycles
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

Registers

Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch

Mnemonic	Description	Bytes	Cycles
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 INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine

Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Input/Output

Mnemonic	Description	Bytes	Cycles
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

Flags

Mnemonic	Description	Bytes	Cycles
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

Mnemonic	Description	Bytes	Cycles
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

Mnemonic	Description	Bytes	Cycles
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

Mnemonic	Description	Bytes	Cycles
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

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1

2.3 หลักการของ CPU

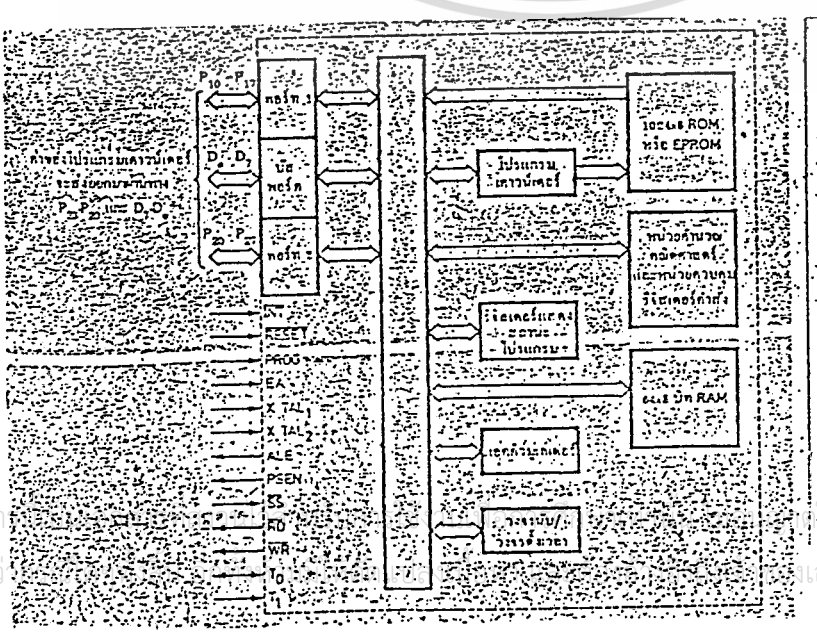
2.4 8048 : โครงสร้างที่ซับซ้อนแต่ใช้งานง่าย

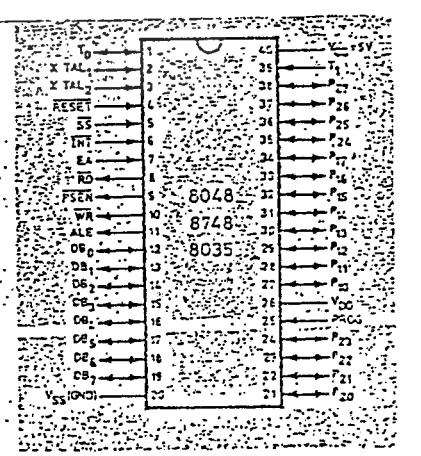
โครงสร้างของ 8048 ก็เหมือนกับโครงสร้างของไมโครคอมพิวเตอร์ทั่วไป กล่าวคือ ประกอบด้วยบัสที่ทำหน้าที่เชื่อมต่อกับอุปกรณ์อื่น ๆ โดยขาของไอซีที่ออกมาส่วนหนึ่งจะเป็นขาอินพุท-เอาท์พุท

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

เบอร์	หน่วยความจำในชิป		ช่วงเวลาไซเคิล	พอร์ทอินพุท เอาท์พุท	อินเตอร์รัพท์	วงจรสัญเสา	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 ROMในตัว ทั้งสามตัวนี้มี RAM ที่ทำหน้าที่เป็นส่วนเก็บข้อมูลได้ 64 ไบท์ในแผ่นชิปของมัน

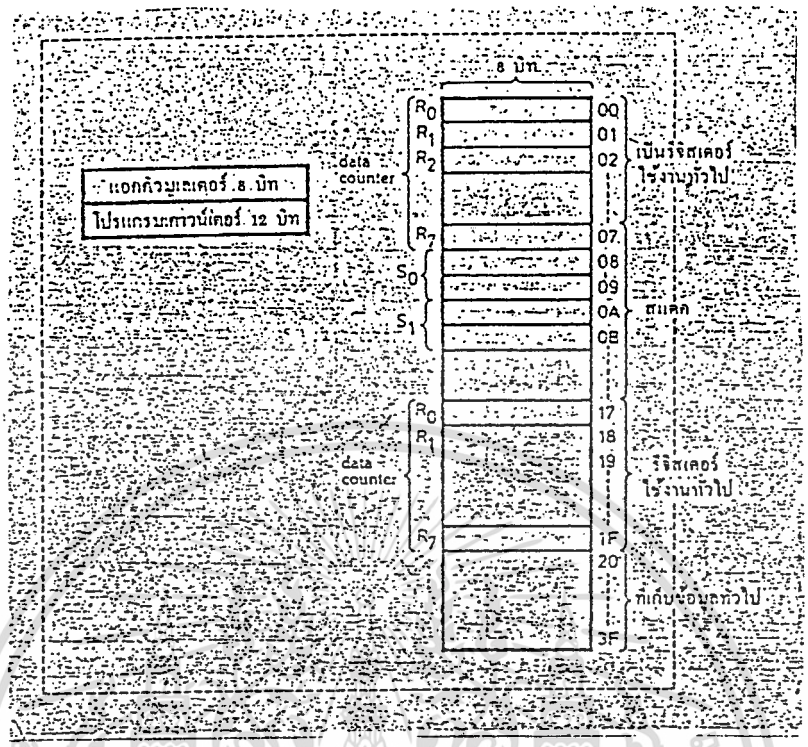
8048 มีโปรแกรมเคาวน์เตอร์ 12 บิท ซึ่งจะทำให้มันสามารถแอดเซสข้อมูลโปรแกรมได้สูงถึง 4Kแต่เนื่องจาก 8048 และ 8748 สามารถทำงานร่วมกับ ROM ภายในของมันแล้ว 1K ดังนั้น มันจึงสามารถอ้างถึงหน่วยความจำภายนอกได้อีก 3072 ไบท์ ส่วน 8035 ต้องอ้างถึงหน่วยความจำทั้งหมดจาก ROMภายนอก

8048, 8035 และ 8748 มีโครงสร้างสำหรับอินพุท-เอาท์พุท 3 พอร์ท หนึ่งในสามพอร์ทนี้ใช้เป็นบัสพอร์ท ส่วนบัสพอร์ทจะทำหน้าที่มัลติเพล็กซ์สัญญาณแอดเดรสและข้อมูลเข้าด้วยกัน ส่วนอินพุทเอาท์พุทพอร์ทที่เหลือมีลักษณะสำคัญคือ ลักษณะเอาท์พุทของมันจะมีคุณสมบัติการแลทช์อยู่ ส่วนอินพุทนั้นเป็นแบบไม่มีแลทช์ กล่าวคือ ทั้งพอร์ท 1 และพอร์ท 2 จะทำหน้าที่เหมือนอินพุท-เอาท์พุทเหมือนกัน โดยเมื่อมีการส่งเอาท์พุทมายังพอร์ทนี้ มันจะแลทช์ไว้จนกว่าจะมีการส่งข้อมูลมาใหม่อีกครั้งมันจึงจะทับข้อมูลเดิมของมันออกไป แต่สำหรับกรณีอินพุทจะแตกต่างจากหลักการทั่วไป คือการอ่านข้อมูลอินพุทของซีพียูจะได้ตามสถานะของการใช้ลอจิกจากภายนอกมาดึงขาอินพุทเอาท์พุทให้เป็น "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 มีโปรแกรมเคาวน์เตอร์ขนาด 12 บิท และยังมีส่วนของหน่วยความจำภายในจำนวน 64 ไบท์ ส่วนของหน่วยความจำภายในนี้สามารถได้รับการเขียนหรืออ่านได้ แต่ในขณะเดียวกันก็ทำงานในรูปของการเป็นรีจิสเตอร์ทั่วไปด้วย ภายในโครงสร้างของซีพียูจึงมีลักษณะเป็นดังรูปที่ 4

แอคคิวมูเลเตอร์ เป็นรีจีสเตอร์หลักที่จะมีส่วนในการเข้าไปกระทำในส่วนของ ALU ในทางคำสั่งเกี่ยวกับทางคณิตศาสตร์ และลอจิก การทำงานของแอคคิวมูเลเตอร์จะถือเป็นโอเปอร์แรนด์ตัวหนึ่งร่วมกับรีจีสเตอร์อื่นหรือหน่วยความจำ



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

รีจีสเตอร์ทั่วไปนี้สองตัวแรกใช้เป็นตัวสำหรับแอดเดรสข้อมูลในหน่วยความจำ ซึ่งโดยทั่วไปเราใช้ RO และ R1 เป็นตัวชี้แอดเดรส

นอกจากนี้ยังมีส่วนของหน่วยความจำแบบสแตค รายละเอียดของการใช้งานสแตคจะได้กล่าวถึงต่อไป

การแอดเดรสของ 8048 : นอกจากใช้โปรแกรมเคาวน์เตอร์แล้ว ยังใช้รีจีสเตอร์อื่นด้วย

8048 ให้โครงสร้างการจัดหน่วยความจำที่แปลกต่างกับไมโครคอมพิวเตอร์ทั่วไป กล่าวคือหน่วยความจำที่ใช้กับ 8048 จะแยกออกเป็น 2 ส่วนคือ หน่วยความจำสำหรับเก็บข้อมูล ส่วนของหน่วยความจำเก็บโปรแกรมจะมีได้มากที่สุดถึง 4096 ไบท์ ส่วนหน่วยความจำเก็บข้อมูลก็จะมีได้มากที่สุดถึง 320 ไบท์

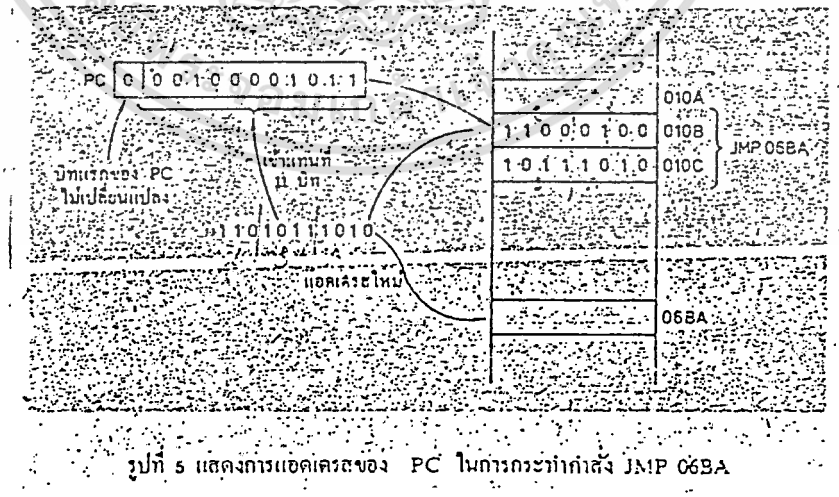
8048 และ 8748 มีส่วนหน่วยความจำในตัวแล้ว 1024 ไบท์ หน่วยความจำนี้เป็นหน่วยความจำเก็บโปรแกรมด้วย ROM ภายนอก ซึ่งจะเก็บได้สูงสุด 4K เช่นกัน

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

ในที่นี้จะกล่าวถึงวิธีการแอดเดรสในแต่ละส่วนแยกจากกัน

การแอดเดรสในส่วนหน่วยความสำหรับโปรแกรม

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

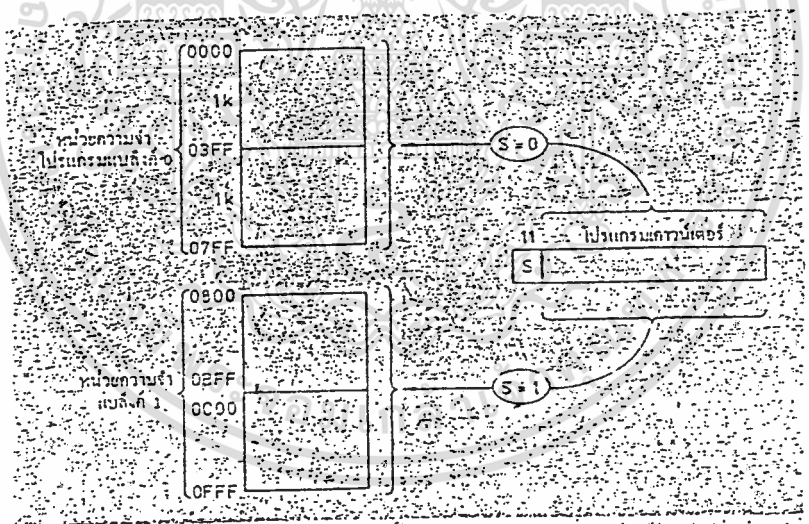


รูปที่ 5 แสดงการแอดเดรสของ PC ในการกระทำคำสั่ง JMP 06BA

แรกเริ่มเดิมที PC มีค่า 010B ซึ่ง PC จะส่งแอดเดรสไปเฟทช์คำสั่งในตำแหน่ง 010B มาคำสั่งที่เฟทช์มานี้เป็นคำสั่ง JMP 06BA ซึ่งซีพียูจะนำข้อมูล 110100111010 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

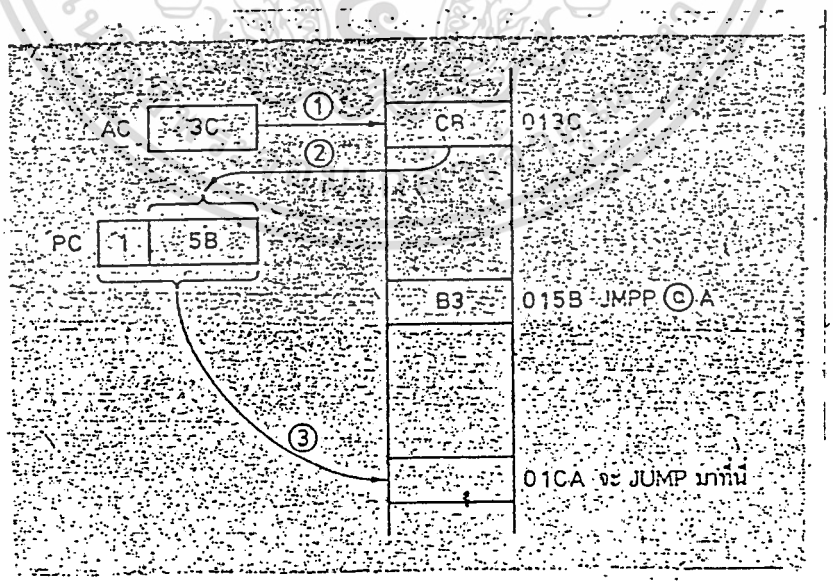
การแบ่งโครงสร้างของหน่วยความจำโปรแกรมออกมาเป็นแบล็งค์ โดยแต่ละแบล็งค์ยัง สามารถแบ่งแยกกลุ่มของหน่วยความจำให้เล็กลงโดยแบ่งเป็นเพจ (Page) แต่ละเพจจะมี หน่วยความจำขนาด 256 ไบท์ ดังนั้น ใน 1 แบล็งค์จะแบ่งออกเป็น 8 เพจ หรือหน่วยความจำ โปรแกรมทั้งหมดแบ่งออกเป็น 16 เพจ การแอดเดรสภายในเพจจะใช้ข้อมูลเพียง 8 บิทเท่านั้น จึงทำให้กรรมวิธีในการแอดเดรสมีประสิทธิภาพดียิ่งขึ้น เพราะสามารถทำการแอดเดรสแบบทางอ้อม ได้ โดยใช้ข้อมูลภายในแอคคิวมูเลเตอร์ขนาด 8 บิท นั่นเอง ลองมาดูวิธีการกระโดดทางอ้อมดูบ้าง

พิจารณาตัวอย่างการกระโดดแบบทางอ้อมในสองกรณีนี้ดู

กรณีที่ 1 เป็นการ JMPP@A หรือเป็นการ JUMP แบบทางอ้อมผ่านข้อมูลในแอคคิวมู เลเตอร์ดู

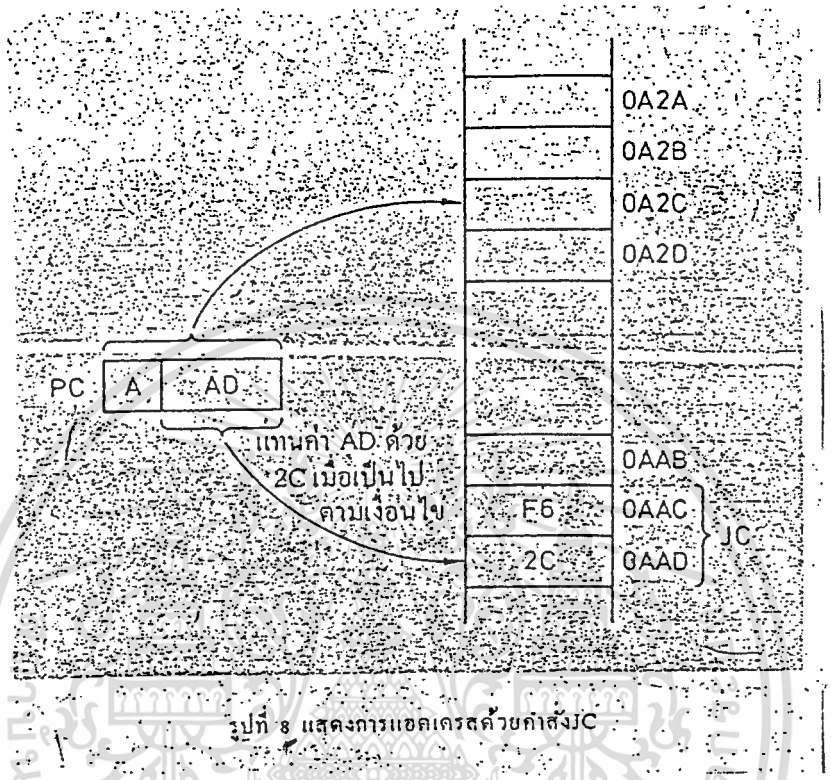
กรณีนี้เดิมข้อมูลใน PC มี 15Bซึ่งจะทำการเฟทช์คำสั่งที่ตำแหน่ง 015B มา ซึ่งในที่นี้ เก็บคำสั่ง JMPP@A ซึ่งมีรหัส 3B คำสั่งนี้สั่งให้กระทำการกระโดด โดยทางอ้อมผ่านรีจีสเตอร์

วิธีการคือ ข้อมูลในแอคคิวมูเลเตอร์จะเป็นแอดเดรสชี้ไปยังหน่วยความจำ เพื่อโหลดข้อมูลใน หน่วยความจำกลับมาใส่ใน PC การใส่ใน PC จะใส่ในเฉพาะ 8 บิทล่างเท่านั้น ผลคือการกระโดด มาตำแหน่ง 01CB ซึ่งยังคงอยู่ในเพจเดียวกัน



รูปที่ .5 แสดงการกระโดดทางอ้อมผ่านA รีจิสเตอร์

กรณีที่ 2 เป็นการกระโดดไปด้วยเงื่อนไข เช่น JC 2C ลักษณะการทำงานเขียนเป็นไดอะแกรมได้ดังรูปที่ 8



รูปที่ 8 แสดงการแอดเดรสด้วยคำสั่ง JC

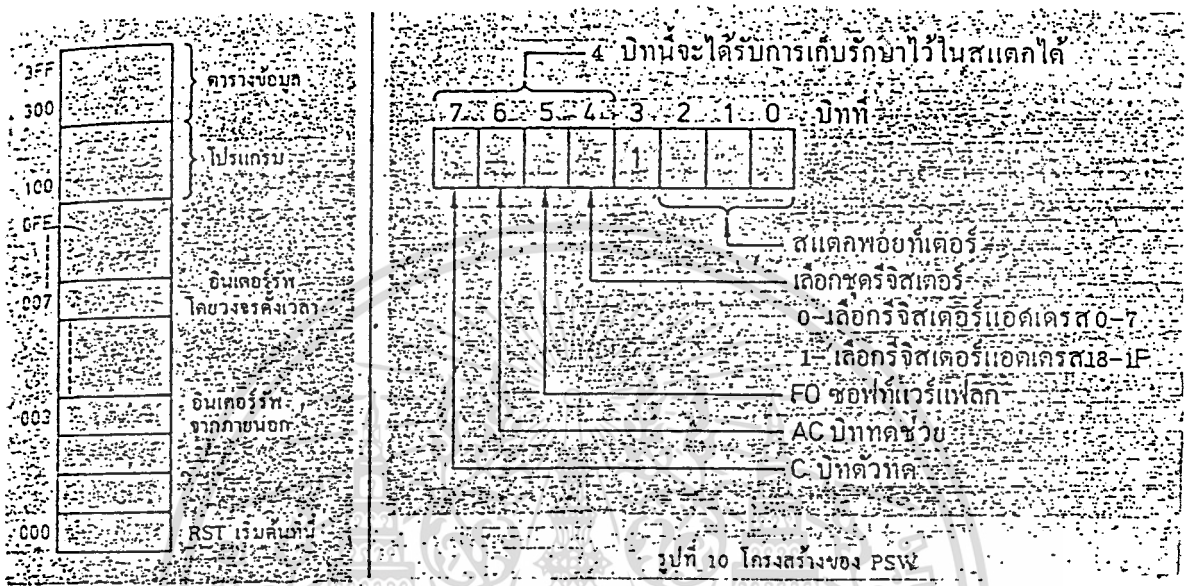
ข้อสังเกต ในส่วนของหน่วยความจำสำหรับโปรแกรมจะเป็นส่วนที่เราสามารถอ่านข้อมูลมาทำการตรวจสอบหรือทำอะไรได้ แต่เราไม่สามารถเขียนโปรแกรมลงในส่วนนี้ได้เลย เราจะไม่มีคำสั่งใดที่ทำให้เกิดการเขียนลงในหน่วยความจำนี้ได้

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



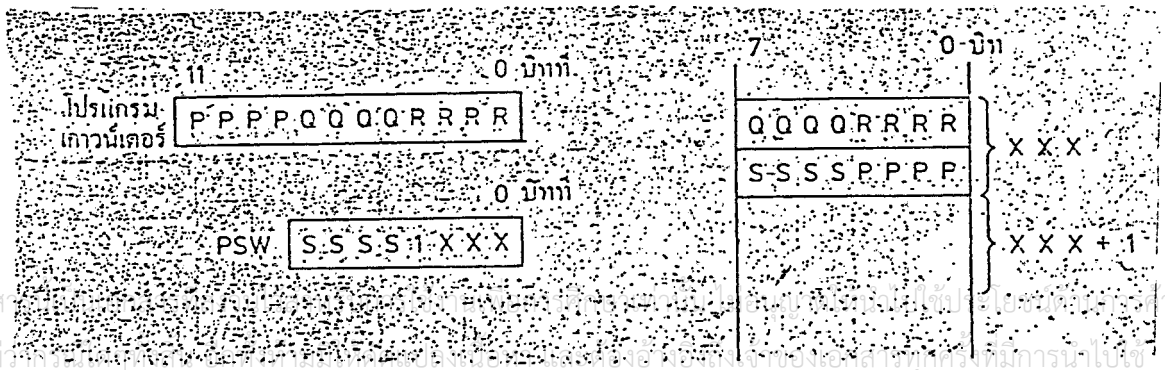
รูปที่ 10 โครงสร้างของ PSW

รูปที่ 9 แสดงการจัดโครงสร้างหน่วย

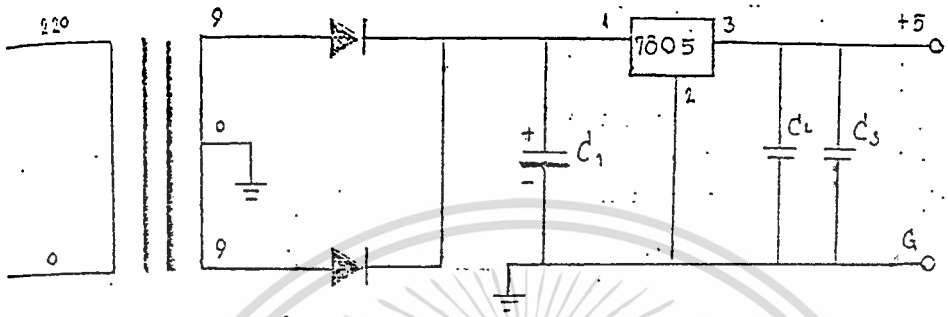
บิท 0 ถึงบิท 2 เป็นสแตคพอยเตอร์ ซึ่งถ้าพิจารณาในที่นี้จะเห็นว่าจะชี้ได้ 8 ตำแหน่ง สแตคพอยเตอร์ของ 8048 ก็เหมือนกับ 8080 แต่ชี้ได้เพียง 8 ตำแหน่งเท่านั้น โดยตำแหน่งของสแตคจะเริ่มจากแอดเดรส 08 โดยคู่ 08 กับ 09 คือ S0 ดังนั้น 3 บิทนี้จึงใช้ได้ 3 ตัว คือ จาก S0 ถึง S8

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

เมื่อมีคำสั่ง CALL หรือคำสั่งที่ต้องใช้เกี่ยวกับสแตคจะเก็บได้ถึง 16 บิท โดยมีโครงสร้างการเก็บดังรูปที่ 11



การสร้างแหล่งจ่ายไฟ



POWER SUPPLY สำหรับ CPU

เนื่องจากไมโครโปรเซสเซอร์จะต้องใช้แรงไฟที่คงที่ ถ้าไม่คงที่ก็จะทำให้เกิดปัญหาขึ้นในภายหลัง เพราะฉะนั้นเราจึงใช้วงจร IC เรกูเลเตอร์ที่ใช้เบอร์ 7805 เพื่อให้แรงดันไฟที่เอาท์พุทมีค่าเป็นบวก 5 volt ที่จะนำไปเลี้ยงในวงจรไมโคร จากวงจรจะเห็นได้ว่ามีวงจรคล้ายคลึงกับวงจรแรกแต่แตกต่างกันตรงที่วงจรแรกมีไฟลบด้วยและ IC เบอร์นี้ก็จะมีขาคล้ายกับเบอร์ 7812 เพียงแต่ระดับแรงดันต่างกันตรงที่วงจรแรกมีแรงดันไฟเป็น +12 V ส่วนวงจรนี้จะมีระดับแรงดันไฟเพียง +5 V เท่านั้น

3 การทดลอง

3.1 ผลการทดลอง

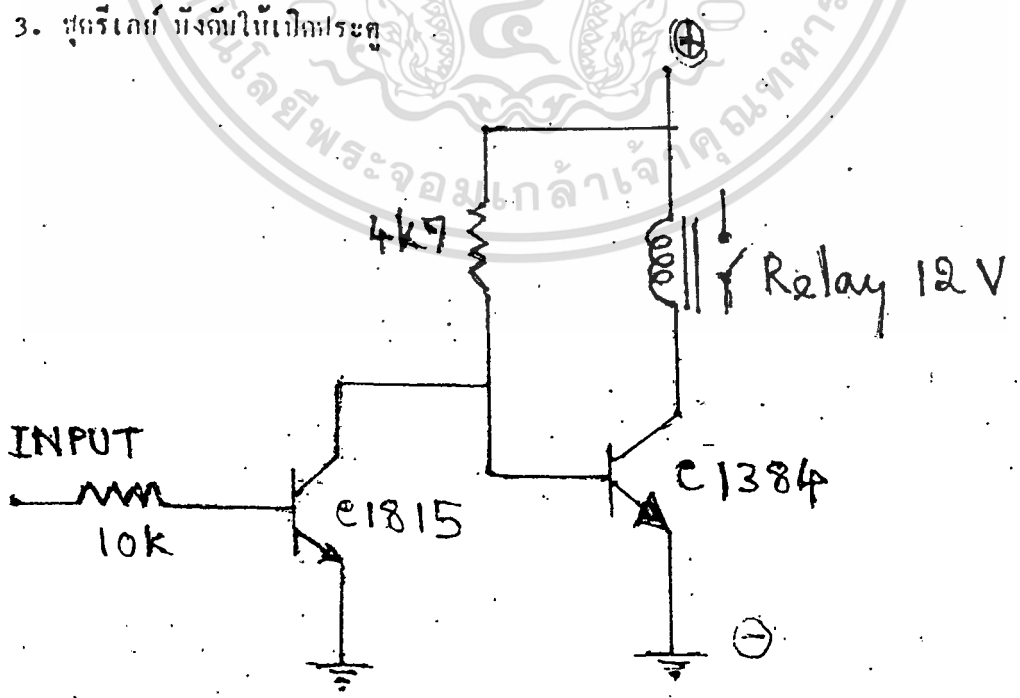
ในการทดลองครั้งแรกนั้นปัญหาที่เกิดขึ้นเนื่องจากหัวอ่านจะอ่านระหัสผิดบ่อย ๆ เนื่องจาก LED ตัวส่งและตัวรับ จะไม่ตรงกันดังนั้นในการออกแบบสร้างหัวอ่านระหัสนี้จะต้องให้ตัวรับและตัวส่งตรงกัน อีกประการหนึ่งตัว Inter rumpt จะต้องอยู่ในตำแหน่งที่ตรงกัน และจำต้องออกแบบทางแมคคานิคให้ระหัสตรงแล้วจึงทำการ Inter rumpt การอ่านจึงจะถูกต้อง

สำหรับปัญหาอื่นนั้นไม่มีเมื่อแก้ไขส่วนนี้แล้วผลการทดลองจะแม่นยำมากสามารถนำไปใช้งานได้ทันที

3.2 วิธีนำไปใช้งาน สำหรับการทดลองครั้งนี้ได้นำไปใช้เปิดประตูโดยการต่อที่ Out Put ซึ่งปกติเป็น High เมื่อสอดบัตรเข้าไปถ้าระหัสถูกต้องจะเป็น LED ดังวงจรข้างล่างนี้โดยการนำ Transister N - PN มาต่อเข้ากับชุด Relay เปิด-ปิด ประตู

ดังนั้นเมื่อสอดบัตรเข้าไปประตูก็สามารถเปิดได้

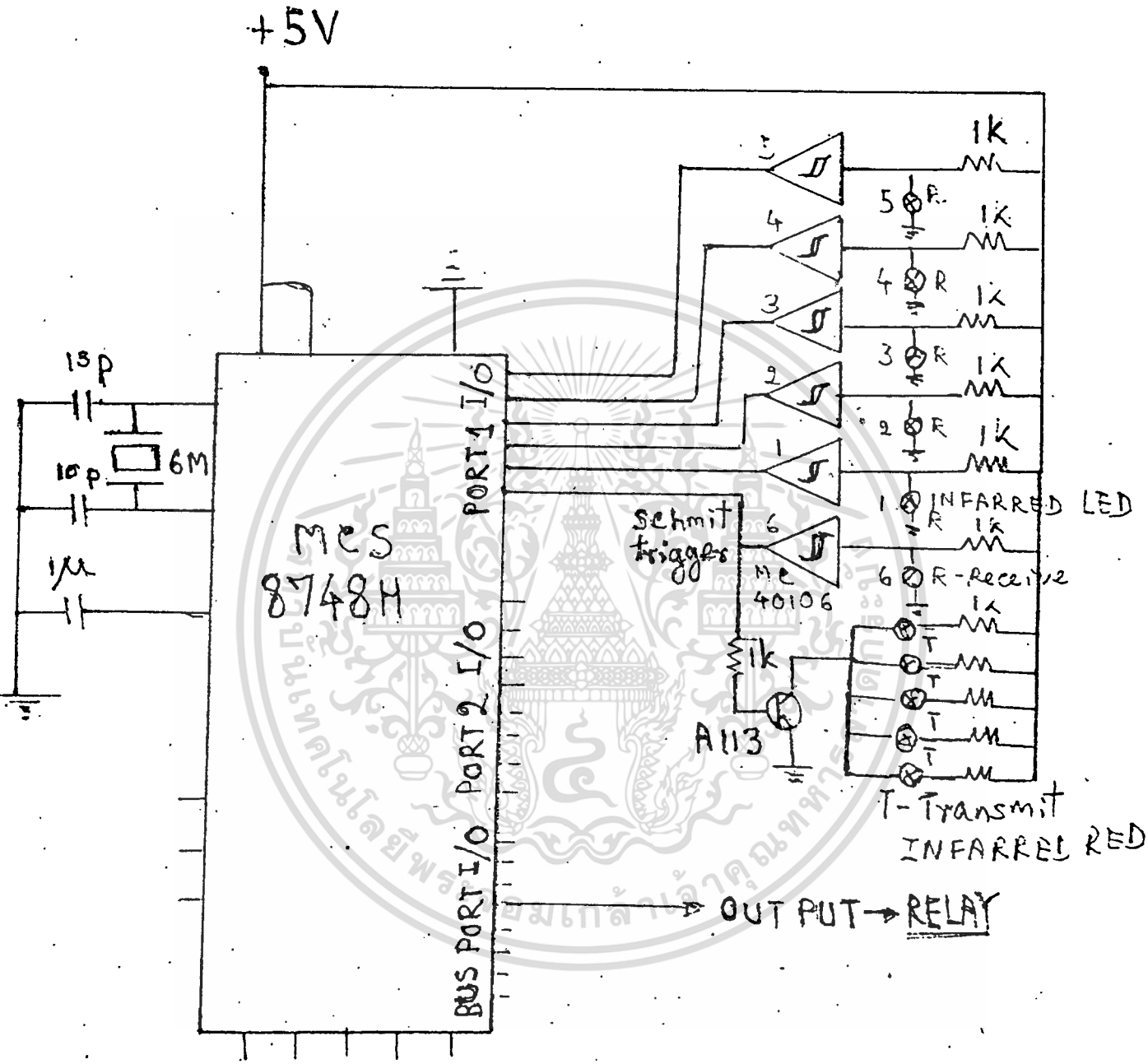
3. ชุดรีเลย์ บังคับให้เปิดประตู



บทสรุปและวิจารณ์

การดำเนินการสร้างกุญแจ การ์ด อินฟาเรดนี้ ได้ดำเนินการทดลอง ซึ่งในครั้งแรกนั้น ก็มีอุปสรรคเกิดขึ้นมากมายอาทิ การอ่านระหัสไม่ค่อยถูกต้องนัก จึงทำให้ระบบการบังคับ เปิดประตูทำงานได้ไม่ถูกต้อง แต่อย่างไรก็ตาม ก็สามารถแก้ปัญหาเหล่านี้ได้จนสามารถทำงานได้ถูกต้อง

ดังนั้นโครงงานนี้จึงนับได้ว่าเป็นโครงงานที่สมบูรณ์โครงงานหนึ่ง สามารถนำไปใช้งานได้อย่างดียิ่ง



วงจร

หนังสืออ้างอิง

1. "Introductory Experiments in Digital Electronics and 8048A Microcomputer Programming and Interfacing"

2. JOHN P.HAYES
"Digital System Design and Microprocessors"

3 เซมิคอนดัคเตอร์ อิเลคทรอนิคส์ วารสาร ฉบับที่ 110 กย-ตค 34

PROGRAM

0000	04	22	JMP	022H
0002	00		NOP	
0003	93		RETR	
0004	00		NOP	
0005	00		NOP	
0006	00		NOP	
0007	93		RETR	
0008	28		XCH	A,R0
0009	43	29	ORL	A,#29H
000B	43	4F	ORL	A,#4FH
000D	50		ANL	A,@R0
000E	59		ANL	A,R1
000F	52	49	JB2	49H
0011	47		SWAP	A
0012	48		ORL	A,R0
0013	54	20	CALL	220H
0015	31		XCHD	A,@R1
0016	39		OUTL	P1,A
0017	38		DB	38H
0018	38		DB	38H
0019	20		XCH	A,@R0
001A	42		MOV	A,T
001B	2E		XCH	A,R6
001C	57		DA'	A

0076	09	IN	A,P1
0077	B2 76	JB6	76H
0079	BA 0A	MOV	R2,#0AH
007B	54 2B	CALL	22BH
007D	EA 7B	DJNZ	R2,7BH
007F	99 7F	ANL	P1,#7FH
0081	88 48	ORL	BUS,#48H
0083	04 2B	JMP	02BH
0085	98 F7	ANL	BUS,#0F7H
0087	04 76	JMP	076H
0089	09	IN	A,P1
008A	37	CPL	A
008B	B2 89	JB5	89H
008D	B8 20	MOV	R0,#20H
008F	BA 07	MOV	R2,#07H
0091	09	IN	A,P1
0092	B2 91	JB5	91H
0094	B9 14	MOV	R1,#14H
0096	E9 96	DJNZ	R1,96H
0098	09	IN	A,P1
0099	B5 D8	JB4	98H
009D	53 0F	ANL	A,#0FH
009F	A0	MOV	@R0,A
00A0	18	INC	R0
00A1	EA AB	DJNZ	R2,0ABH
00A3	09	IN	A,P1
00A4	D2 B2	JB6	0B2H

001D	2E	XCH	A,R6
001E	4B	ORL	A,R3
001F	41	ORL	A,@R1
0020	4F	ORL	A,R7
0021	20	XCH	A,@R0
0022	23 FF	MOV	A,#0FFH
0024	02	OUTL	BUS,A
0025	23 7F	MOV	A,#7FH
0027	39	OUTL	P1,A
0028	23 FF	MOV	A,#0FFH
002A	3A	OUTL	P2,A
002B	14 89	CALL	089H
002D	E6 2B	JNC	2BH
002F	0A	IN	A,P2
0030	37	CPL	A
0031	53 3F	ANL	A,#3FH
0033	C6 41	JZ	41H
0035	BA 06	MOV	R2,#06H
0037	B9 03	MOV	R1,#03H
0039	67	RRC	A
003A	F6 43	JC	43H
003C	19	INC	R1
003D	19	INC	R1
003E	19	INC	R1
003F	EA 39	DJNZ	R2,39H
0041	B9 00	MOV	R1,#00H

0043	14 DA	CALL	ODAH
0045	C6 74	JZ	74H
0047	B9 15	MOV	R1,#15H
0049	0A	IN	A,P2
004A	D2 4E	JB6	4EH
004C	B9 18	MOV	R1,#18H
004E	14 DA	CALL	ODAH
0050	C6 74	JZ	74H
0052	B9 1B	MOV	R1,#1BH
0054	0A	IN	A,P2
0055	F2 59	JB7	59H
0057	B9 1E	MOV	R1,#1EH
0059	14 DA	CALL	ODAH
0058	C6 74	JZ	74H
005D	B9 21	MOV	R1,#21H
005F	14 DA	CALL	ODAH
0061	C6 74	JZ	74H
0063	B9 24	MOV	R1,#24H
0065	14 DA	CALL	ODAH
0067	C6 74	JZ	74H
0069	B9 27	MOV	R1,#27H
006B	14 DA	CALL	ODAH
006D	C6 85	JZ	85H
006F	09	IN	A,P1
0070	D2 6F	JB6	6FH
0072	04 7F	JMP	07FH
0074	98 BF	ANL	BUS,#0BFH

00A6	09		IN	A,P1
00A7	B2	D8	JB5	0D3H
00A9	04	A3	JMP	0A3H
00AB	09		IN	A,P1
00AC	B2	D8	JB5	0D8H
00AE	92	98	JB4	98H
00B0	04	AB	JMP	0ABH
00B2	89	80	ORL	P1,#80H
00B4	B8	21	MOV	R0,#21H
00B6	F0		MOV	A,@R0
00B7	47		SWAP	A
00B8	18		INC	R0
00B9	40		ORL	A,@R0
00BA	C8		DEC	R0
00BB	C8		DEC	R0
00BC	A0		MOV	@R0,A
00BD	18		INC	R0
00BE	18		INC	R0
00BF	18		INC	R0
00C0	F0		MOV	A,@R0
00C1	47		SWAP	A
00C2	18		INC	R0
00C3	40		ORL	A,@R0
00C4	C8		DEC	R0
00C5	C8		DEC	R0
00C6	C8		DEC	R0

00C7	A0	MOV	@R0,A
00C8	18	INC	R0
00C9	18	INC	R0
00CA	18	INC	R0
00CB	18	INC	R0
00CC	F0	MOV	A,@R0
00CD	47	SWAP	A
00CE	18	INC	R0
00CF	40	ORL	A,@R0
00D0	C8	DEC	R0
00D1	C8	DEC	R0
00D2	C8	DEC	R0
00D3	C8	DEC	R0
00D4	A0	MOV	@R0,A
00D5	97	CLR	C
00D6	A7	CPL	C
00D7	83	RET	
00D8	97	CLR	C
00D9	83	RET	
00DA	F9	MOV	A,R1
00DB	E3	MOVP3	A,@A
00DC	B8 20	MOV	R0,#20H
00DE	D0	XRL	A,@R0
00DF	96 ED	JNZ	0DEH
00E1	19	INC	R1
00E2	18	INC	R0

00E3	F9		MOV	A,R1
00E4	E3		MOVP3	A,@A
00E5	D0		XRL	A,@R0
00E6	96	ED	JNZ	0EDH
00E8	19		INC	R1
00E9	18		INC	R0
00EA	F9		MOV	A,R1
00EB	E3		MOVP3	A,@A
00EC	D0		XRL	A,@R0
00ED	83		RET	
00EE	00		NOP	
00EF	00		NOP	
00F0	00		NOP	
00F1	00		NOP	
00F2	00		NOP	
00F3	00		NOP	
00F4	00		NOP	
00F5	00		NOP	
00F6	00		NOP	
00F7	00		NOP	
00F8	00		NOP	
00F9	00		NOP	
00FA	00		NOP	
00FB	00		NOP	
00FC	00		NOP	
00FD	00		NOP	

01EA	00	NOP	
01EB	00	NOP	
01EC	00	NOP	
01ED	00	NOP	
01EE	00	NOP	
0200	B8 20	MOV	R0,#20H
020	BA 03	MOV	R2,#03H
0204	F0	MOV	A,@R0
0205	47	SWAP	A
0206	53 0F	ANL	A,#0FH
0208	54 13	CALL	213H
020A	F0	MOV	A,@R0
020B	53 0F	ANL	A,#0FH
020D	54 13	CALL	213H
020F	18	INC	R0
0210	EA 04	DJNZ	R2,04H
0212	83	RET	
0213	A9	MOV	R1,A
0214	96 18	JNZ	18H
0216	B9 10	MOV	R1,#10H
0218	98 BF	ANL	BUS,#0BFH
021A	54 2B	CALL	22BH
021C	88 40	ORL	BUS,#40H
021E	54 2B	CALL	22BH
0220	54 2B	CALL	22BH

01E0	00	NOP
01E1	00	NOP
01E2	00	NOP
01E3	00	NOP
01E4	00	NOP
01E5	00	NOP
01E6	00	NOP
01E7	00	NOP
01E8	00	NOP
01E9	00	NOP
01EA	00	NOP
01EB	00	NOP
01EC	00	NOP
01ED	00	NOP
01EE	00	NOP
01EF	00	NOP
01F0	00	NOP
01F1	00	NOP
01F2	00	NOP
01F3	00	NOP
01F4	00	NOP
01F5	00	NOP
01F6	00	NOP
01F7	00	NOP
01F8	00	NOP
01F9	00	NOP

00FE	00	NOP
00FF	00	NOP
0100	00	NOP
0101	00	NOP
0102	00	NOP
0103	00	NOP
0104	00	NOP
0105	00	NOP
01CD	00	NOP
01CE	00	NOP
01CF	00	NOP
01D0	00	NOP
01D1	00	NOP
01D2	00	NOP
01D3	00	NOP
01D4	00	NOP
01D5	00	NOP
01D6	00	NOP
01D7	00	NOP
01D8	00	NOP
01D9	00	NOP
01DA	00	NOP
01DB	00	NOP
01DC	00	NOP
01DD	00	NOP
01DE	00	NOP
01DF	00	NOP

```
0222    E9  18      DJNZ    R1,18H
0224    B9  0F      MOV     R1,#0FH
0226    54  2B      CALL    22BH
0228    E9  26      DJNZ    R1,26H
022A    83          RET
022B    BC  4E      MOV     R4,#4EH
022D    BB  00      MOV     R3,#00H
022F    EB  2F      DJNZ    R3,2FH
0231    EC  2F      DJNZ    R4,2FH
0233    83          RET
0234    00          NOP
0235    00          NOP
0236    00          NOP
0237    00          NOP
0238    00          NOP
0239    00          NOP
023A    00          NOP
023B    00          NOP
023C    00          NOP
023D    00          NOP
023E    00          NOP
023F    00          NOP
0240    00          NOP
```

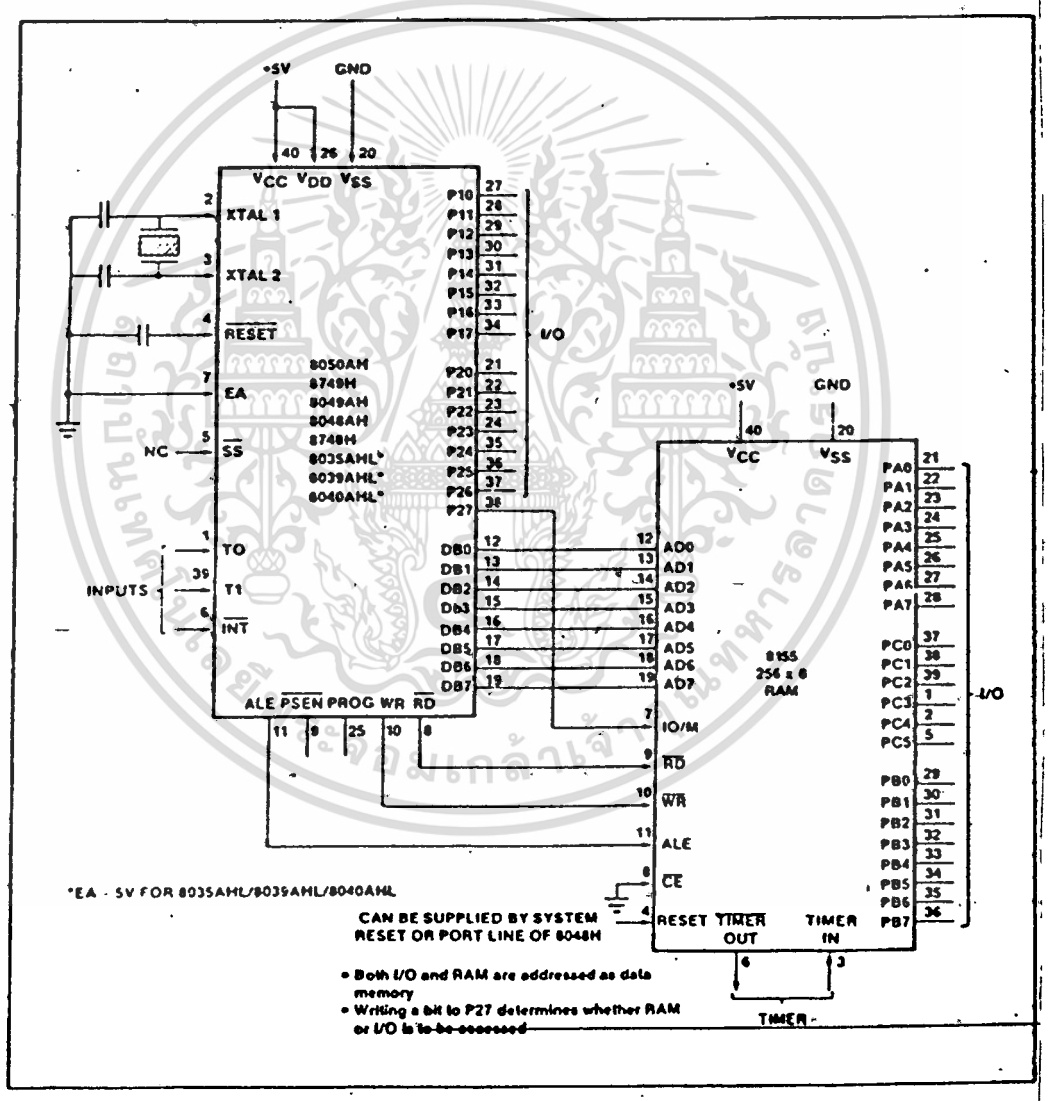
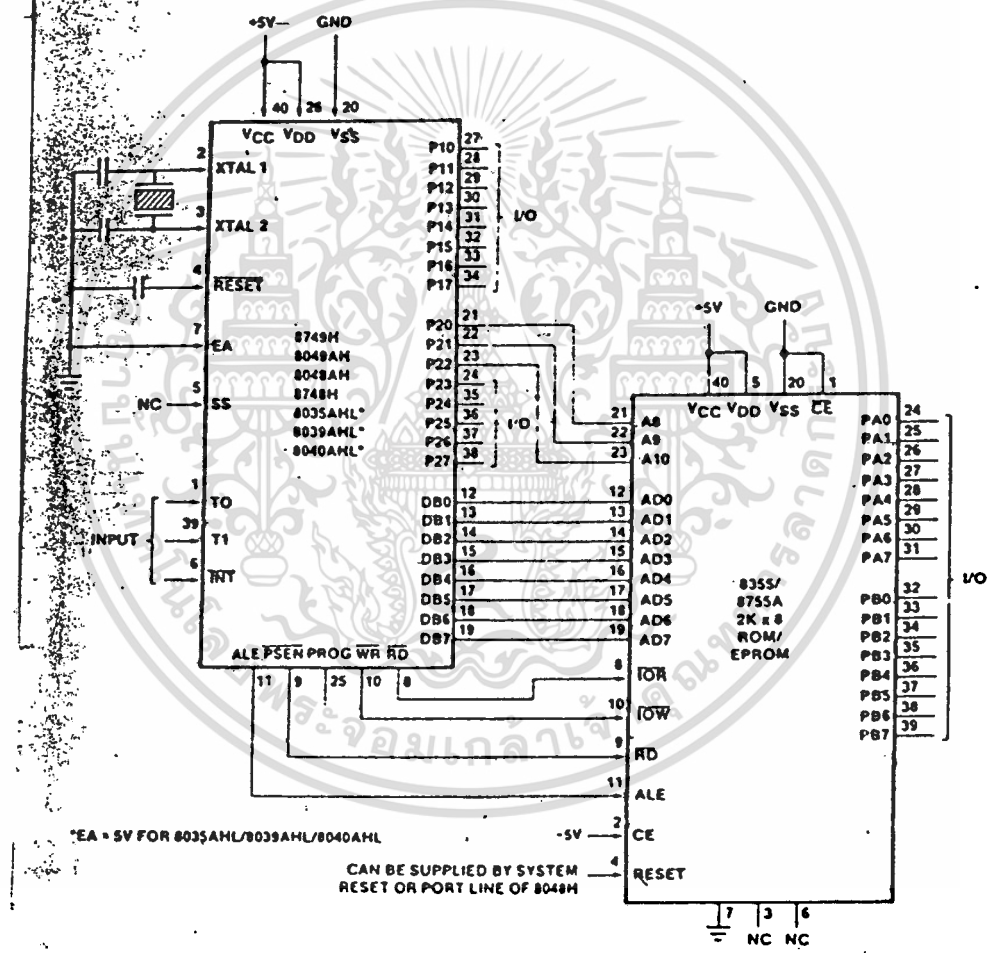


Figure 5-9. Adding a Data Memory and I/O Expander



- External I/O ports are addressed as data memory PA = 00 PB = 01
- If the 8048H's internal Program Memory is used, this configuration will result in the upper 1K of external memory being addressed before the lower 1K. Inverting A10 will correct this if necessary. This inversion is not necessary if the 8049H is used.

Figure 5-8. Adding a Program Memory and I/O Expander

5.2 I/O EXPANSION TECHNIQUES

The following are several examples of how the basic I/O capability of the MCS-48 microcomputers can be easily expanded externally using either the 8243 I/O expander device or standard logic circuits. These techniques can be used whenever the combination memory I/O expanders illustrated on the preceding pages are not required.

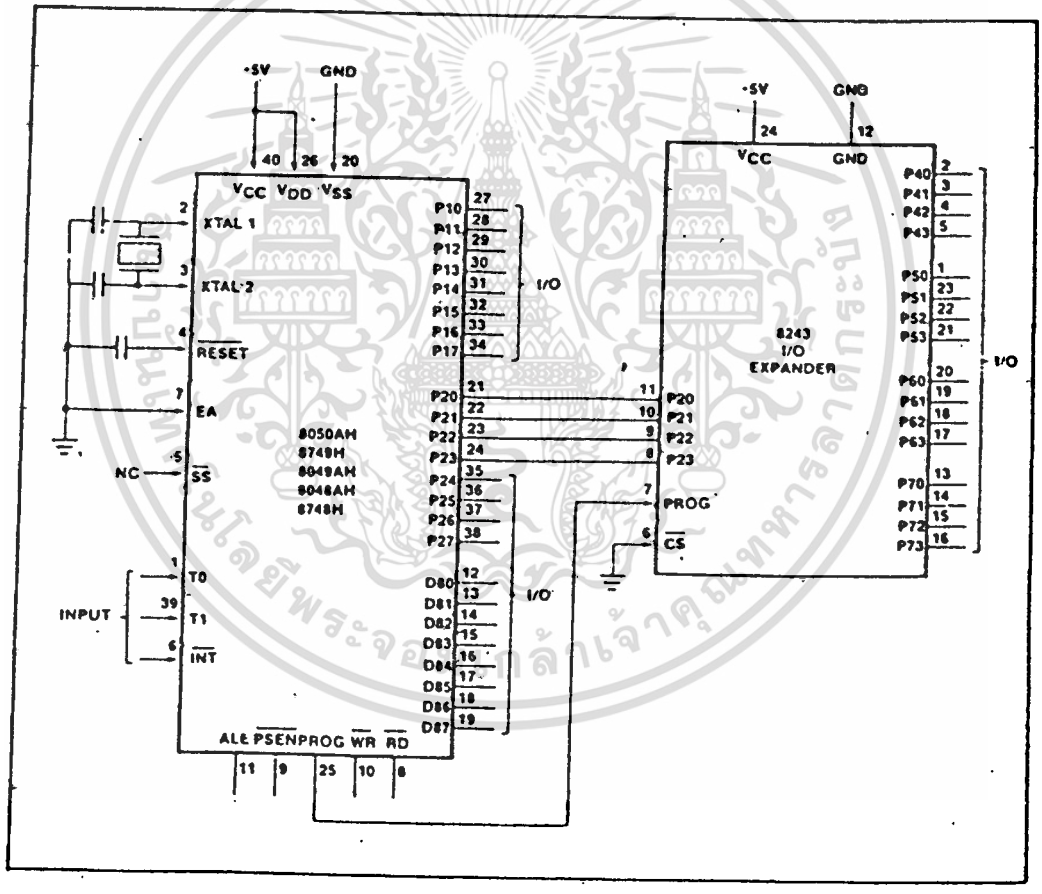


Figure 5-11. Adding an I/O Expander

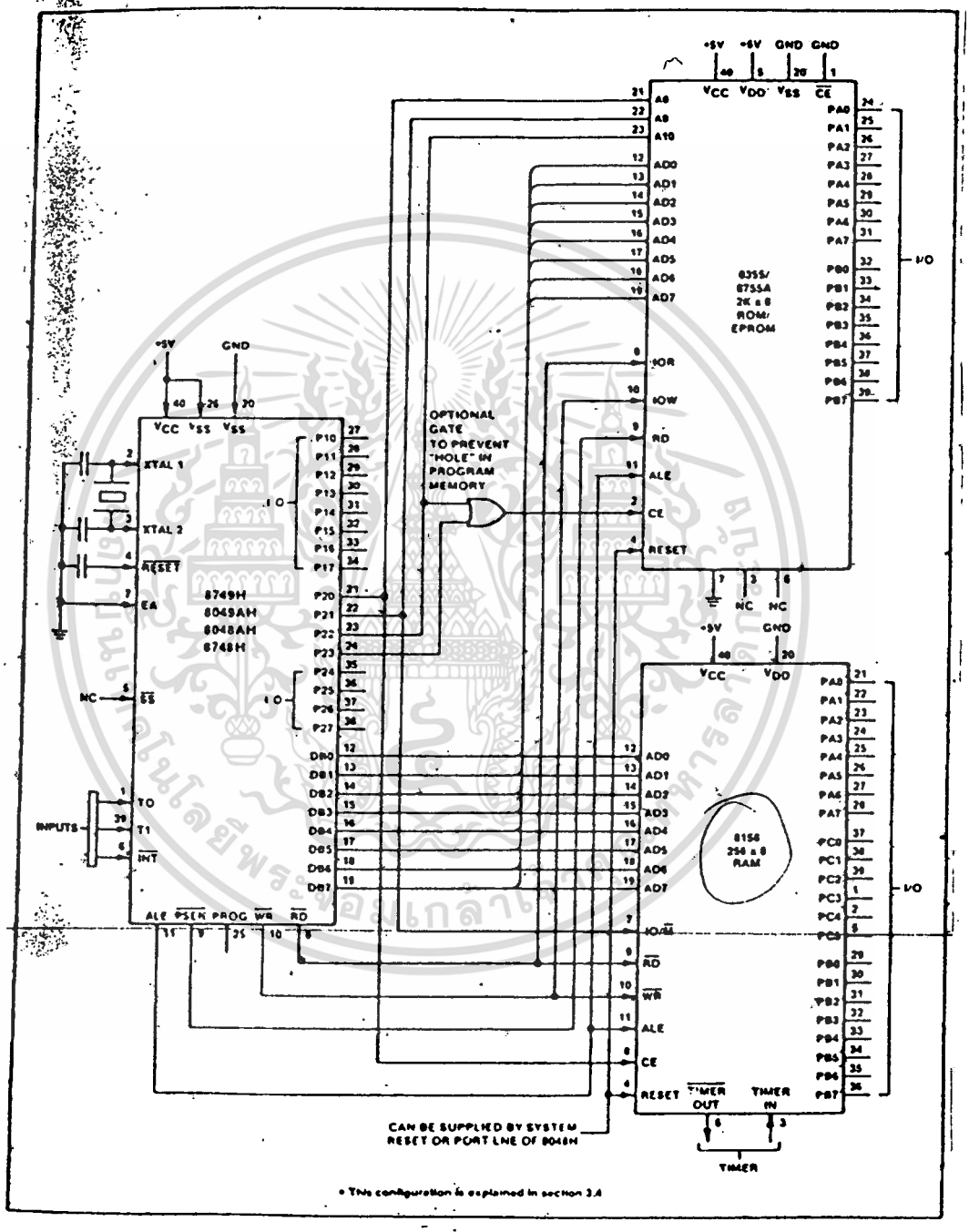


Figure 5-10. The Three-Chip System

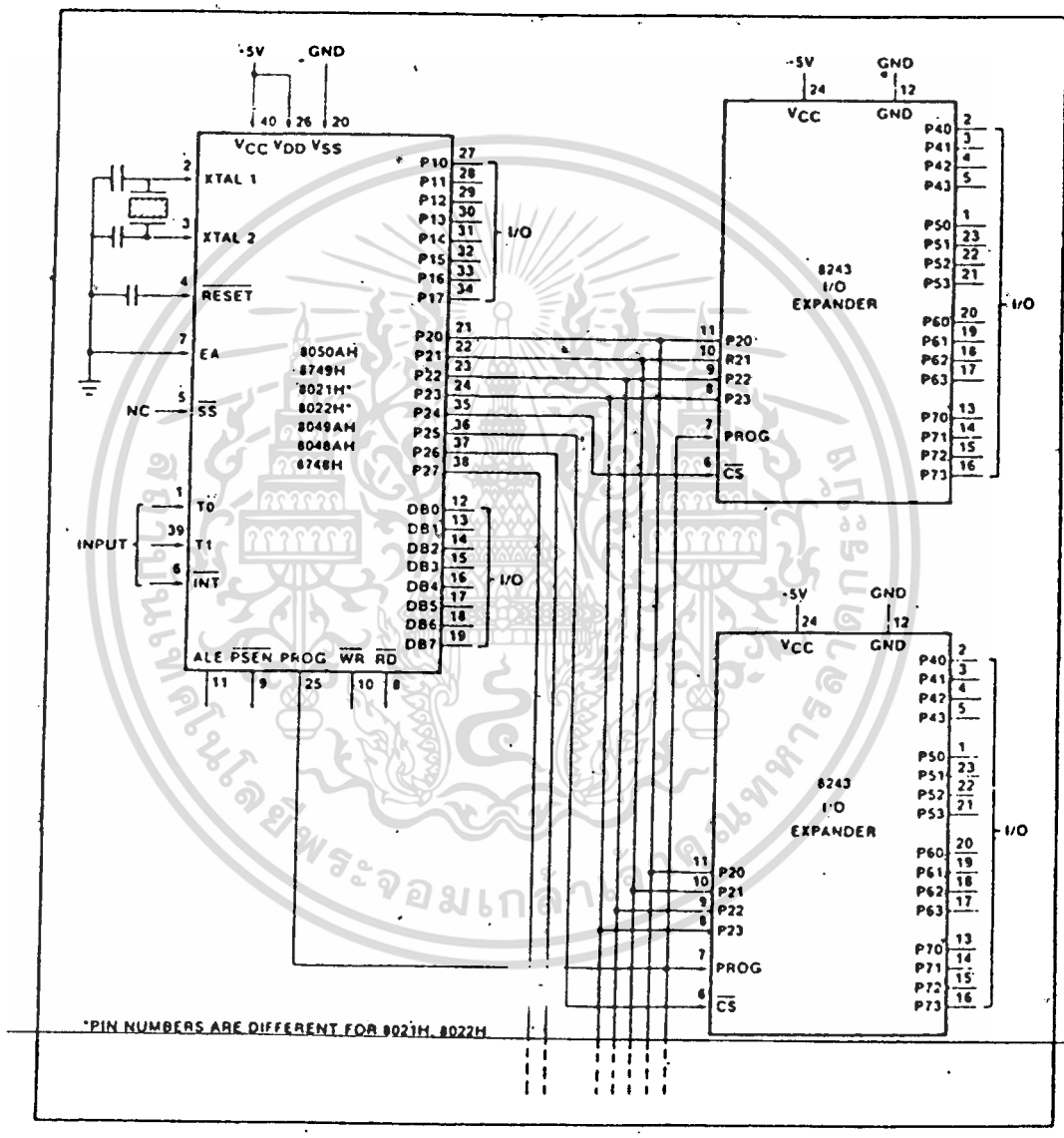


Figure 5-13. Adding Multiple I/O Expanders

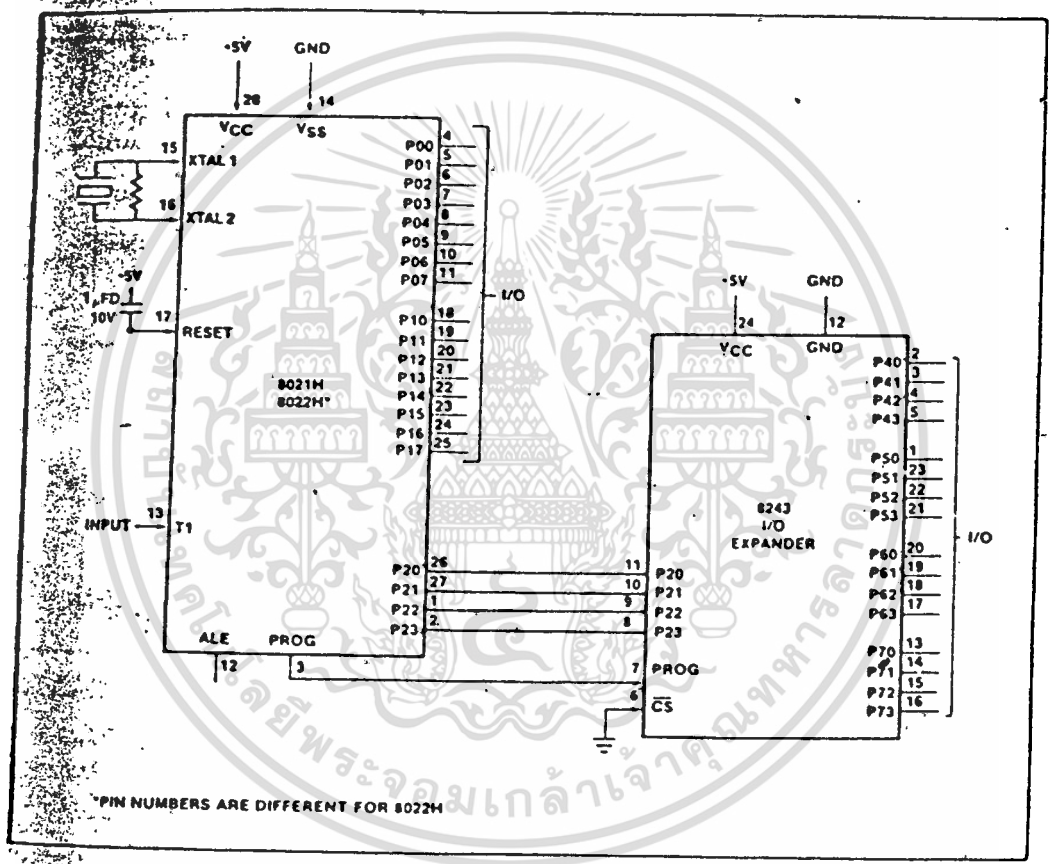


Figure 5-12. Adding an I/O Expander to the 8021H, 8022H

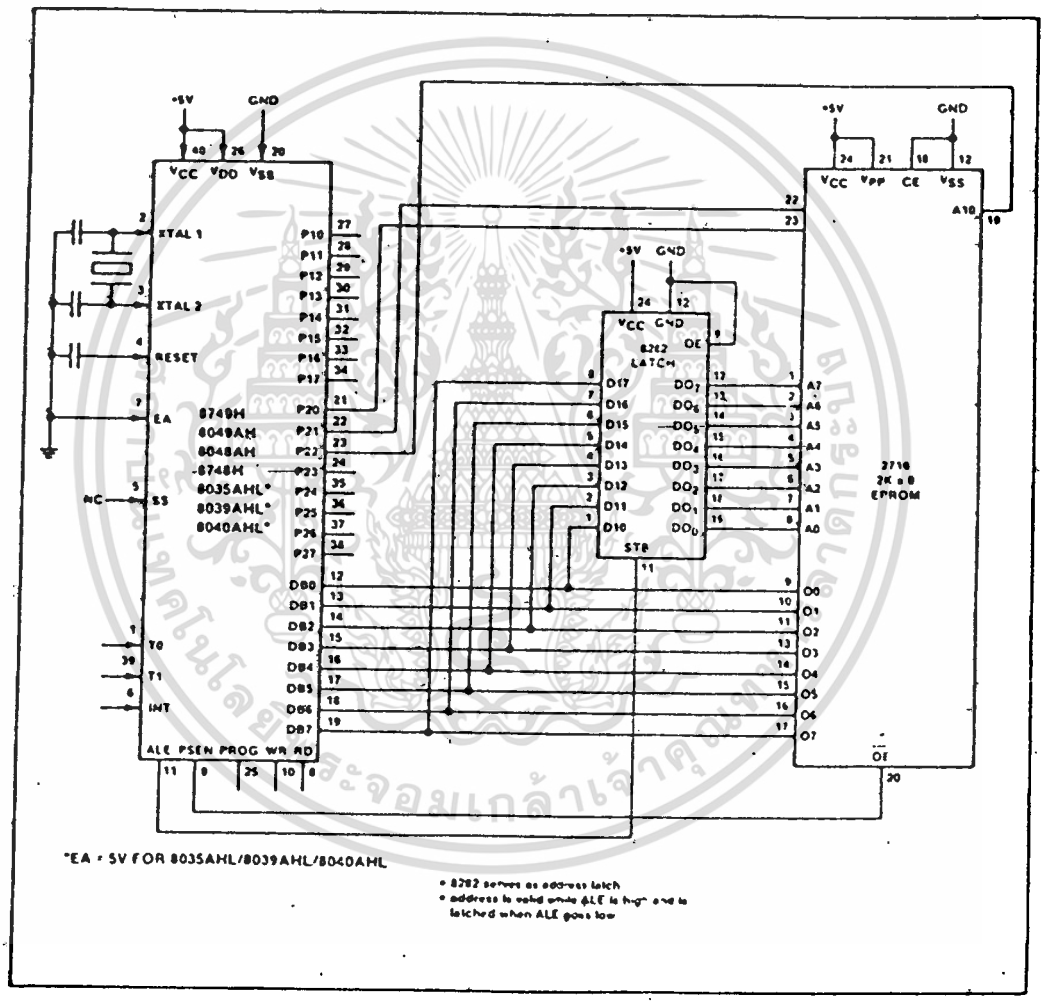


Figure 5-7. External Program Memory

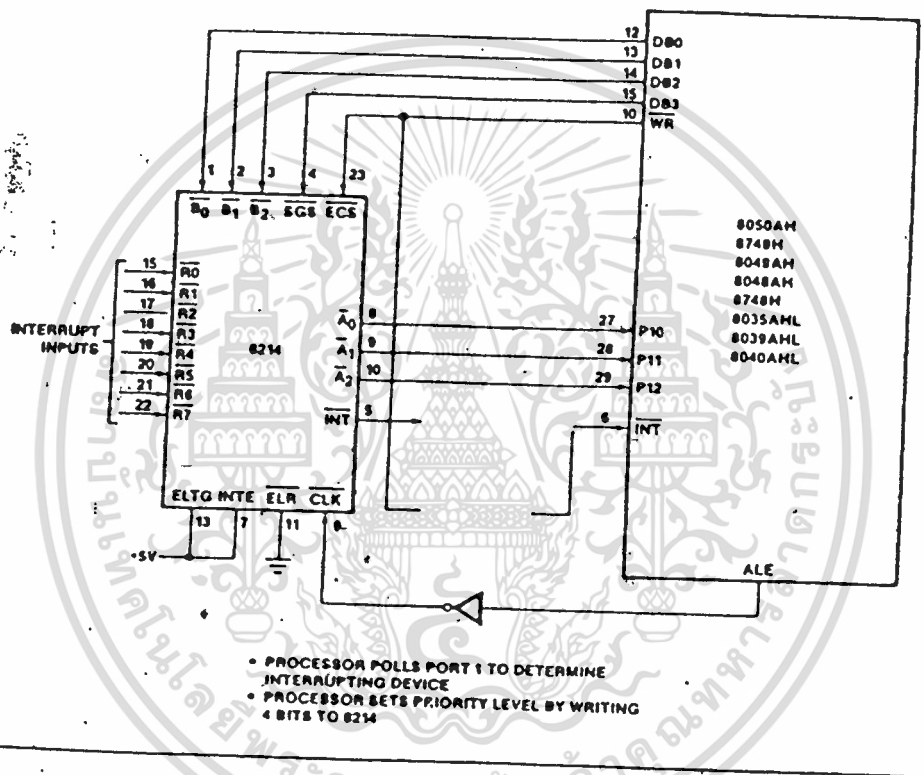


Figure 5-6. Multiple Interrupts with Priority Levels

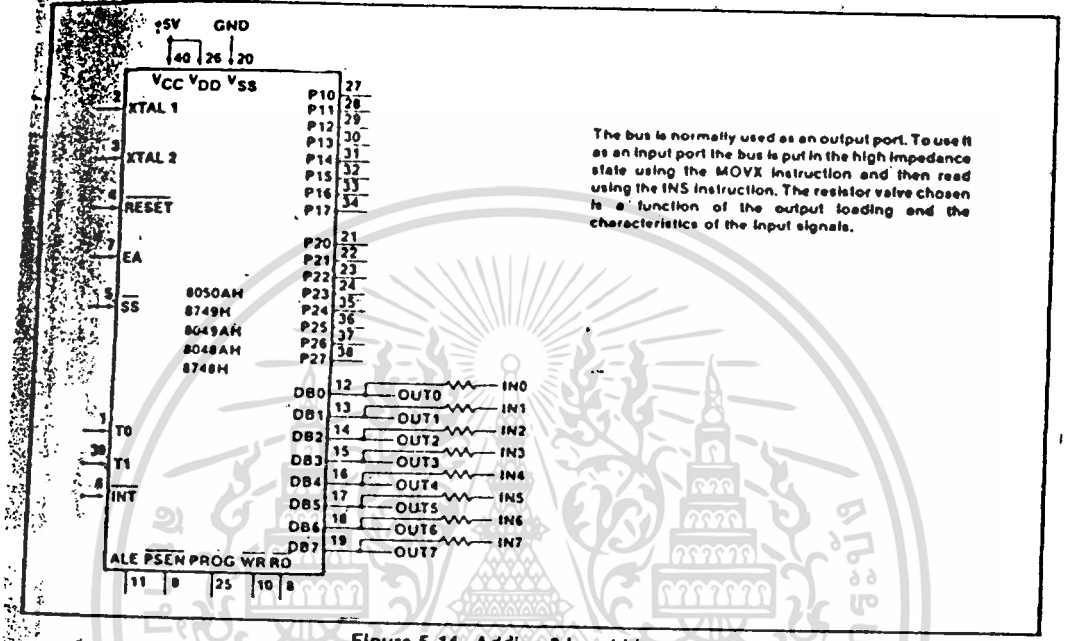


Figure 5-14. Adding 8 Input Lines

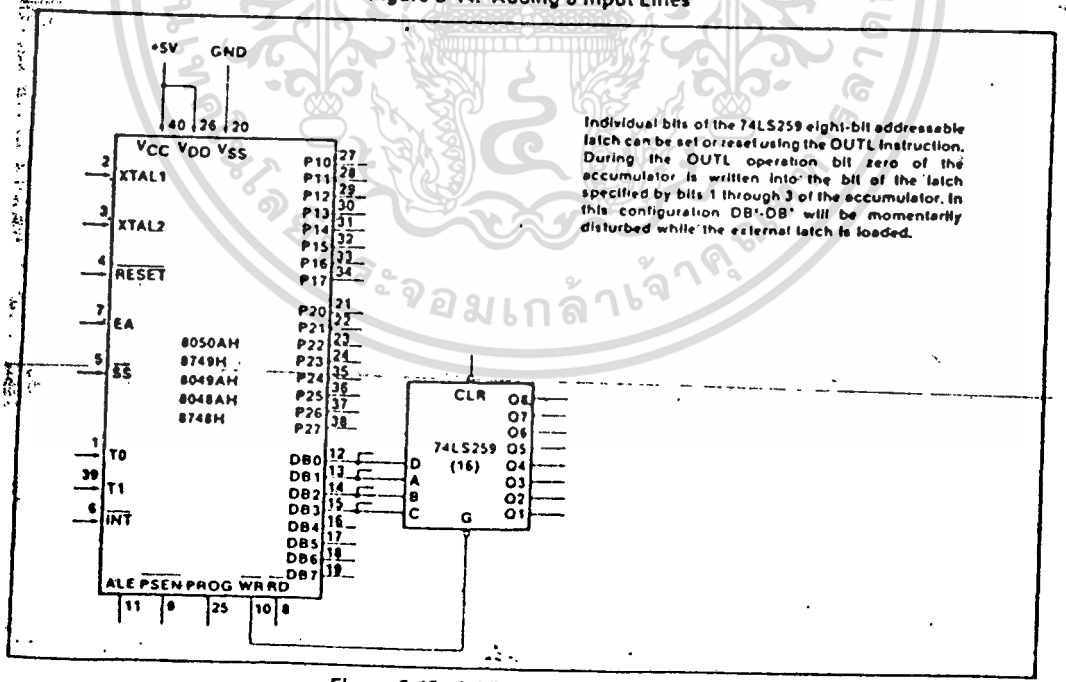


Figure 5-15. Adding 8 Output Lines

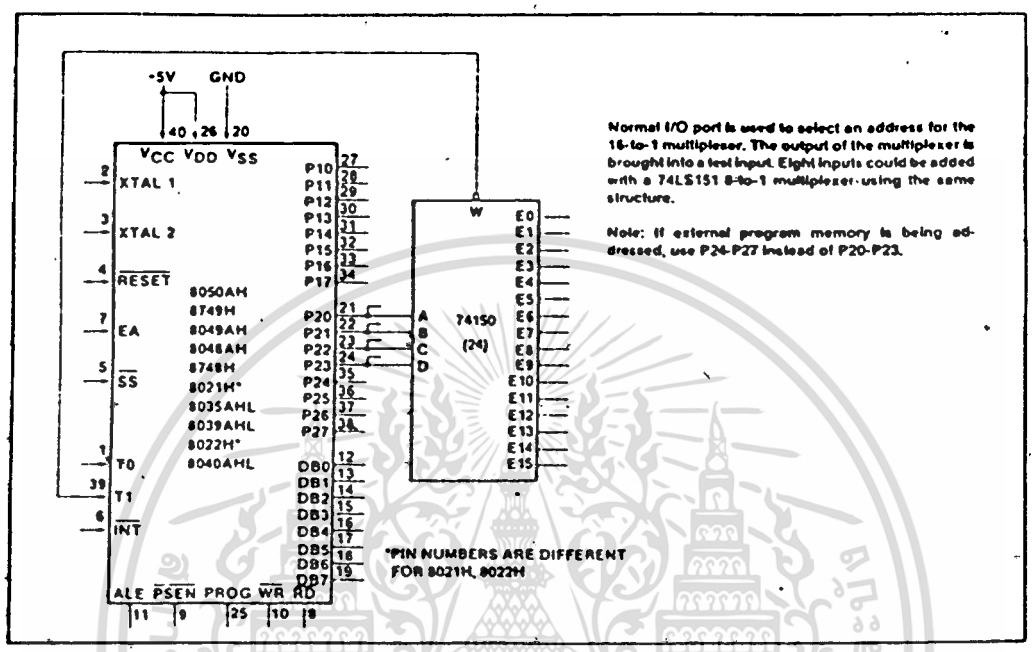


Figure 5-16. Adding 16 Input Lines

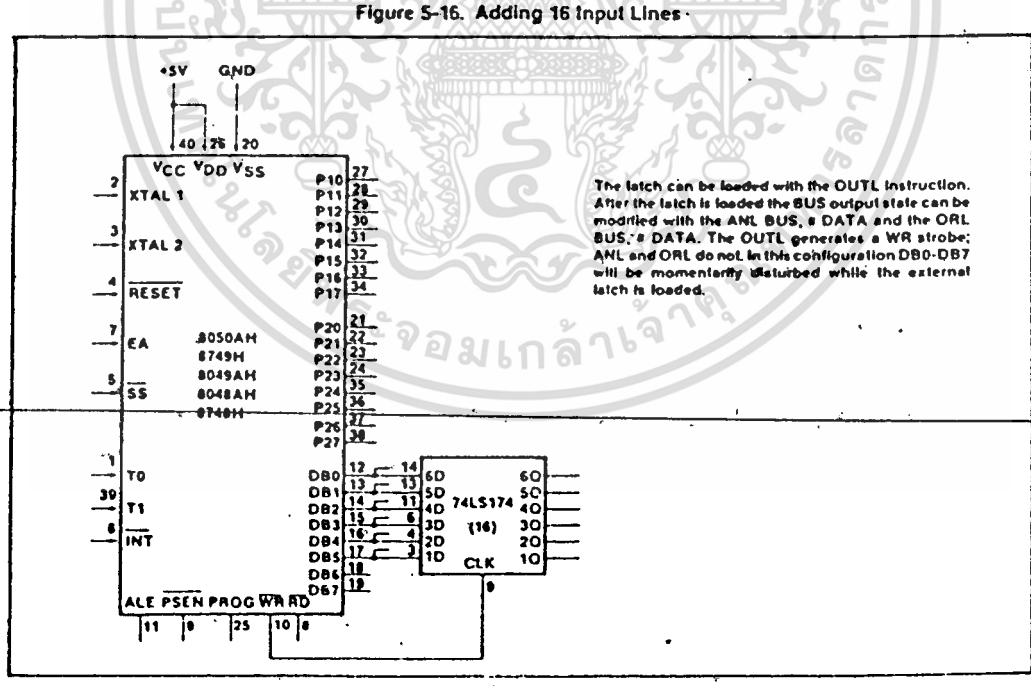


Figure 5-17. Adding 6 Output Lines

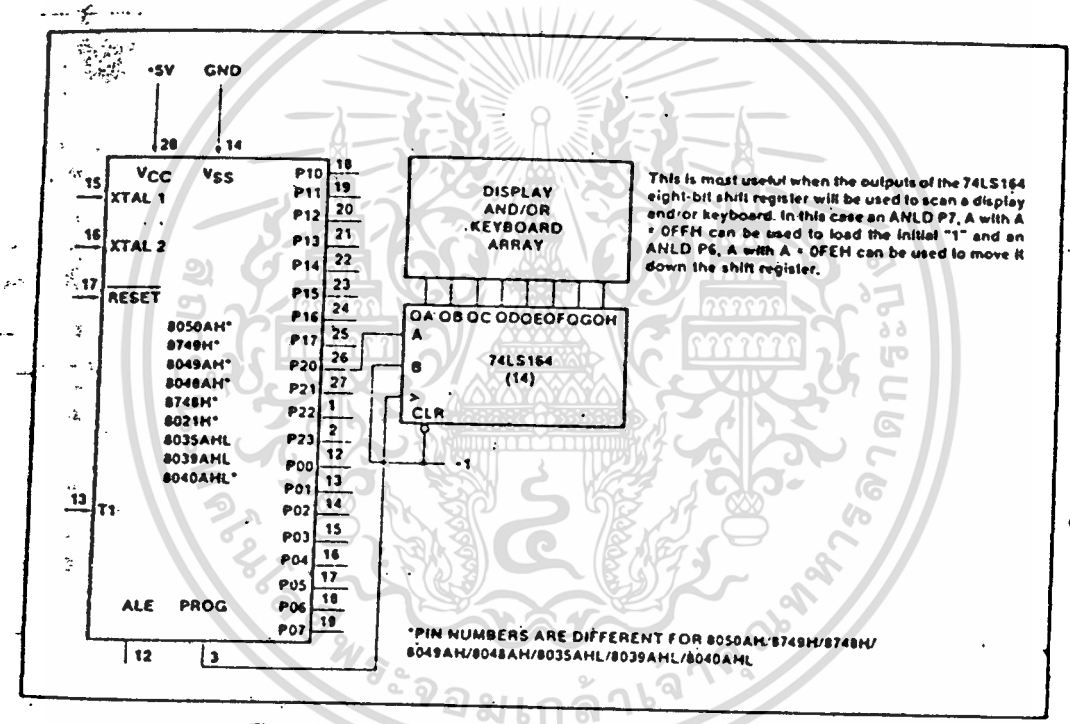


Figure 5-18. Adding Output for Keyboard/Display Scanning

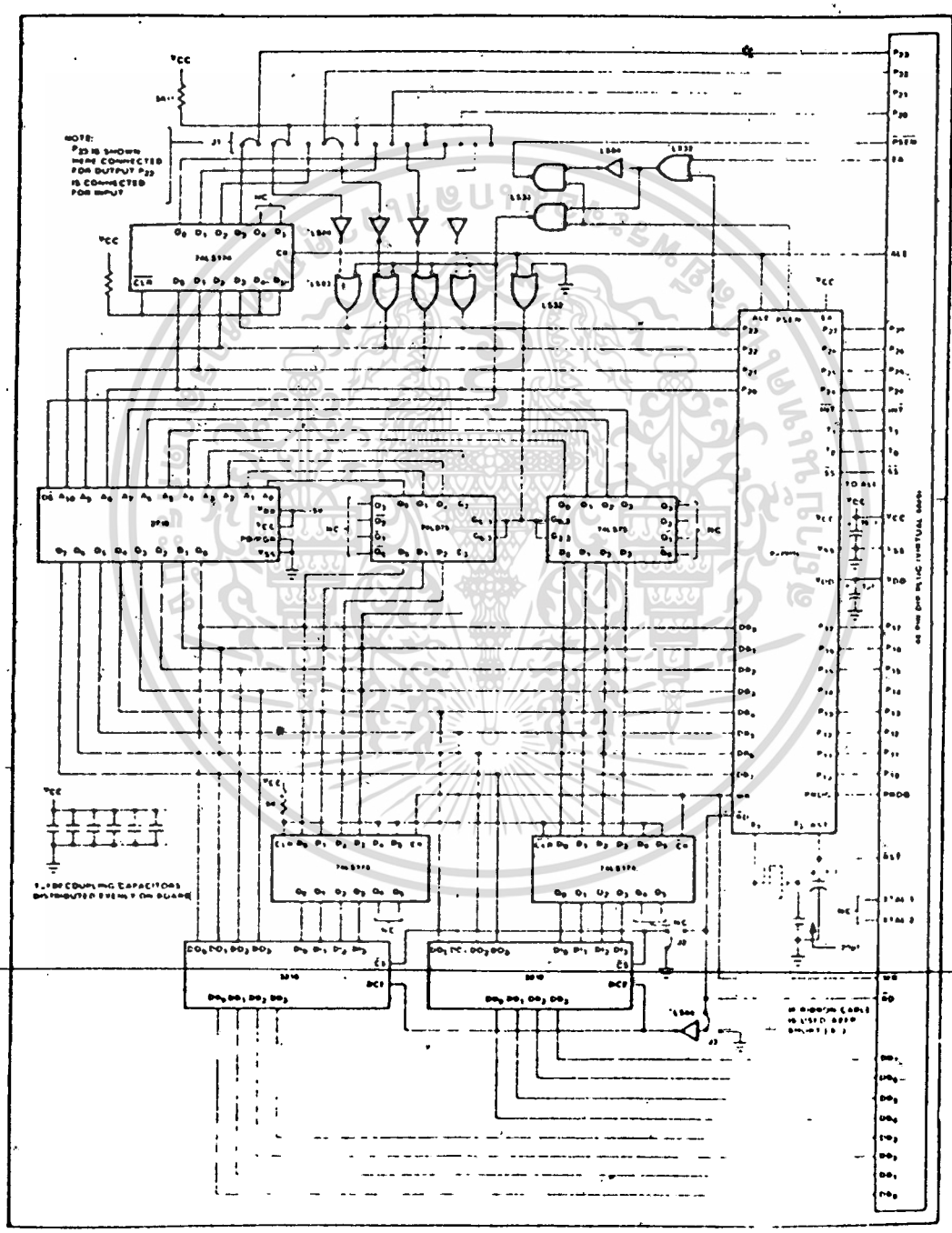


Figure 5-19. 8049AH Emulator Circuit

5.3 8049AH EMULATOR CIRCUIT DESCRIPTION—6 MHZ

The following is an explanation of a circuit which emulates the operation of an Intel 8049AH using a standard EPROM for program storage.

With the 8049AH, software may be developed by running external program memory, but doing so requires the use of the bus and P_{23}-P_{20} to access this memory.

The circuit shown may be used to restore the normal functioning of these twelve I/O pins. The circuit consists of an 8039AHL CPU, 2716 EPROM, two 8216 bidirectional bus drivers, and eight other 7400 Series Low-Power Schottky TTL packages. The whole assembly can be built on a 2¼" x 4" board.

A cable coming off the board can be terminated by a forty-pin plug which may be inserted directly into the CPU socket intended for the 8049AH in a system undergoing design or testing. Alternatively, a pattern of forty pins extending below the board can be used to plug the board directly into the system undergoing testing, "piggy-back" fashion. The emulator board may be configured in various ways so that the 40 pin plug is the logical equivalent of an 8049AH in every legal operating mode. (In the following explanation of the operation of the circuit, an asterisk appearing before a signal or pin number as in *PSEN refers to that pin on the "virtual 8049AH" represented by the forty-pin plug).

Since the CPU is identical with the 8049AH in all respects other than its lack of program memory, most of the pins of the 8039AHL are simply connected directly to the corresponding pins of the forty-pin plug. These include all of Port 1, the high order bits of Port 2, the test pins, etc. Signals which are emulated with additional logic include the rest of Port 2, DB_7-DB_0, *PSEN, etc. $\overline{RD}$, $\overline{WR}$, ALE, and $\overline{PSEN}$ are obtained from the 8039AHL, but are also used by the emulation circuitry.

The EA input of the 8039AHL is hard-wired high so all instruction fetches are made from the 2716. Two 74LS75 four-bit latches gated by the buffered ALE signal are used to hold the lower eight bits of address from the time-multiplexed data bus. Since the Bus is being used for fetching instructions, data latched to the Bus will be lost on the next instruction fetch. Two 74LS174 latches are used to retain the output data when a bus write is executed. These latches are triggered by the trailing edge of the WR pulse, so their outputs are glitch free. Since logical operations to the bus do not generate a WR strobe, the "OUTL BUS, #data" "ANL BUS,#" and "ORL BUS,#" instructions may not be used, though they do function properly with the other ports.

The two 8216 bidirectional bus drivers normally buffer the latched bus contents to the DB pins of the virtual 8049AH. When an "INS A, BUS" instruction is executed, they buffer the input signals on to the emulator data bus. Thus, the circuit is designed to use the Bus for both latched output and strobed input. If DB-DB_0 of the 8049AH are to be used solely for input data, J2 and J3 may be changed from what is shown in the Figure, so that DB-DB_0 act as high impedance inputs and the 8216s are enabled only when the read operation is performed. If the bus is to be used only for latched output, the 8216s can be omitted entirely.

Bidirectional data transfers which require the transfer of address information as well as data, such as to and from external data memory, require removal of the 8216s and replacement with 18 pin jumper blocks on which the DB_x pins are connected with the respective DO_x pins.

The lower four bits of Port 2 are also used in fetching instructions from the 2716, in addition to their use in input or output pins in the user's system. In configuring the emulator for a particular application, the user must dedicate each of these as either an input or output pin and connect jumper set J1 accordingly. Any mix of input and output pins is allowed. At the beginning of each instruction fetch, the last data written to P2 will be present on P_{23}-P_{20} at the rising edge of ALE and will be latched by a 74LS174. The latched data may be connected through the jumpers to those pins which will be used as outputs on the 8049AH. Emulator pins used as inputs should be pulled above 2.0V for a logic "one". If this is not the case, reed switches to Ground are to be used. 50K pullup resistors should be added to the circuit on each input. They were omitted from the diagram to minimize input loading.

Pins which will be used as inputs may be connected to the input of an OR gate formed of inverters and open-collector NAND gates. The input signals will be relayed directly to the 8039AHL and will be read by an "IN A,P" instruction. But when $\overline{PSEN}$ is low, the NAND outputs are forced off, allowing the 8039AHL pins to be used for high-order program addressing. Open-collector outputs are needed to prevent line contention when $\overline{PSEN}$ is not low.

If 8243s will be used in the final system, the low order pins of Port 2 must be connected directly to the plug. This may be done by replacing the Port 2 latch with four jumpers connecting the inputs to the outputs. The NANDs should be removed or disabled by grounding the common NAND inputs.

The cluster of three OR gates is used to enable the on-board 2716 and generate the *PSEN signal, each of which is a function of $\overline{PSEN}$, *EA, and the high order bit of the

program counter. Thus *PSEN is generated, forcing an off-board read, only when a jump has been made to Memory Bank 1 or when *EA is brought high. If this feature is to be used to address off-board memory, DB_7-DB_0 may not be used for normal I/O. The 8216s and 74LS174 must be replaced with jumper blocks and the open collector NAND gates disabled, as explained above. The same changes are required to operate the board in single step mode.

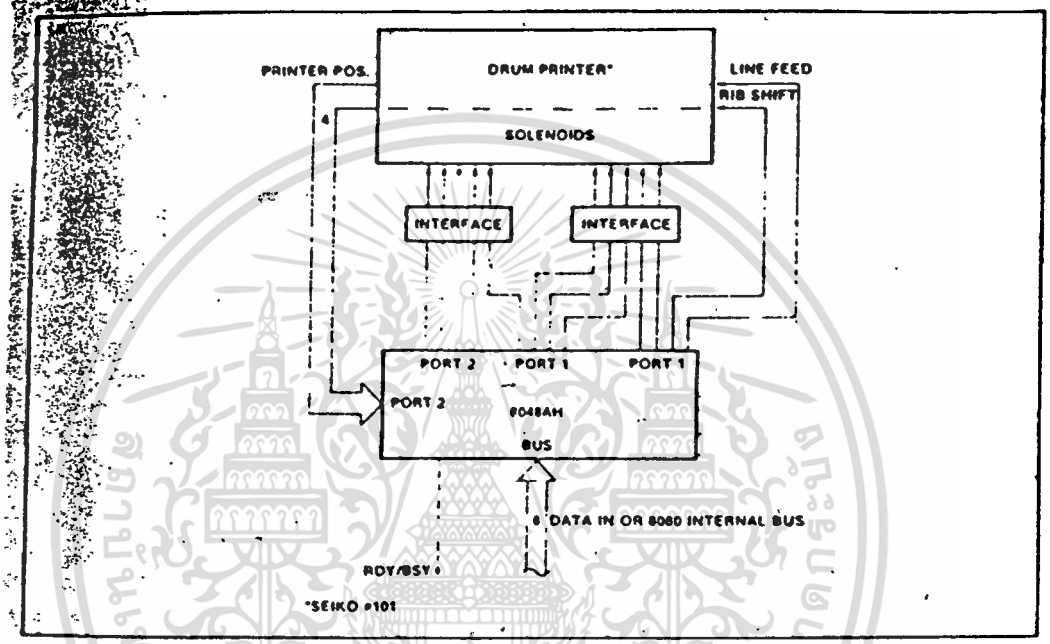


Figure 5-20. 8048AH Interface to Drum Printer

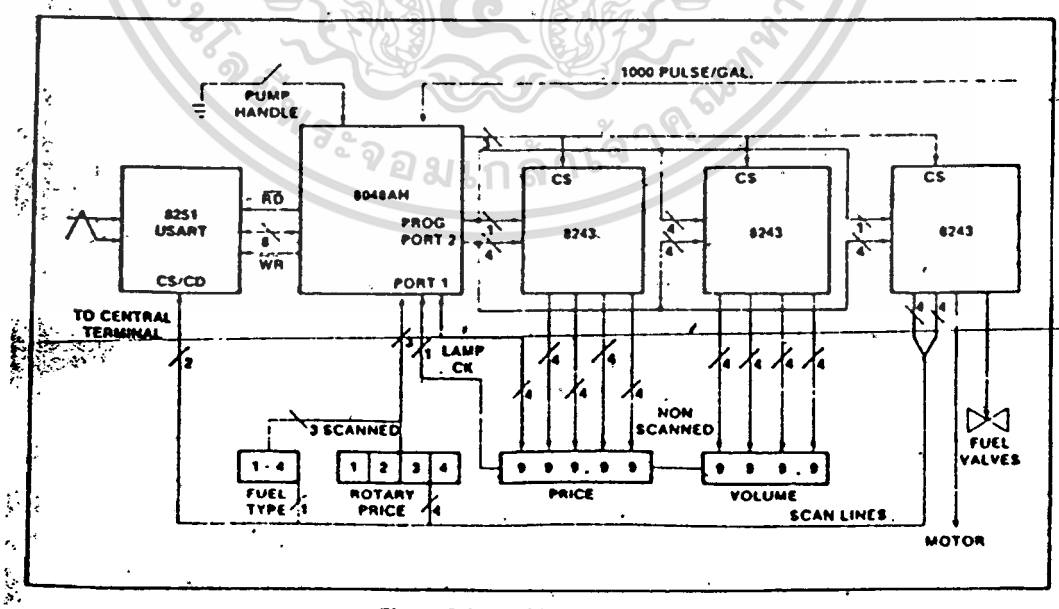


Figure 5-21. MCS-48 Gas Pump

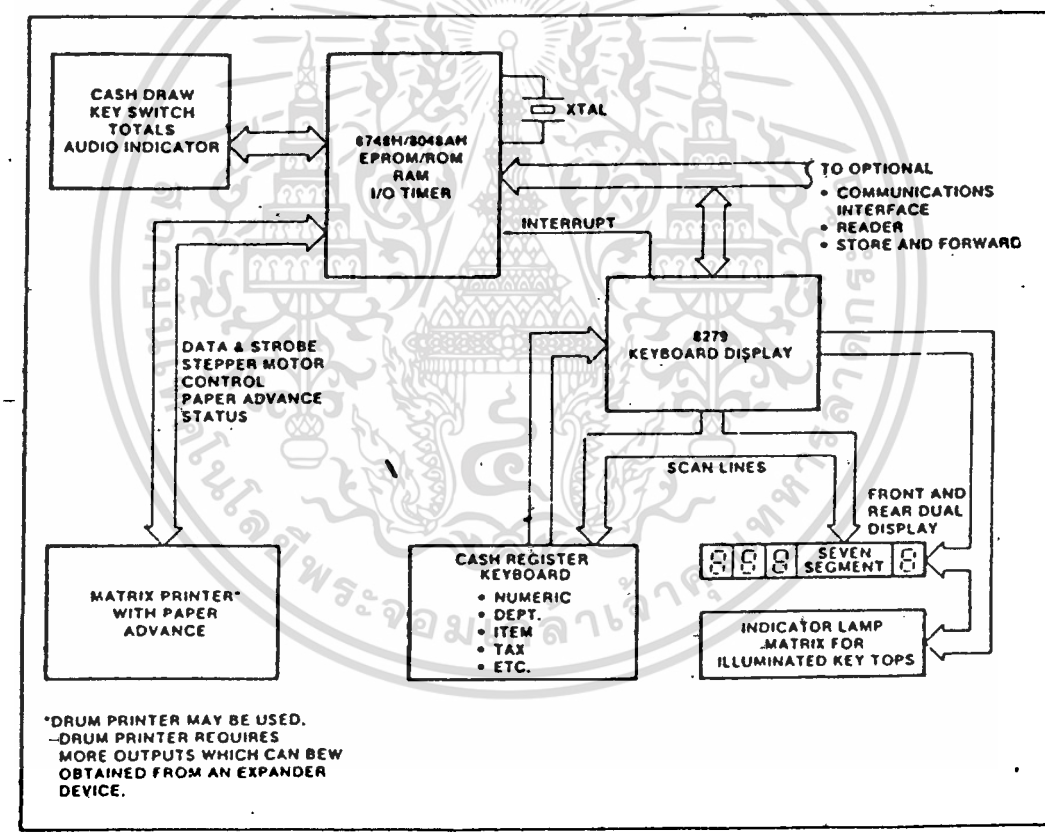


Figure 5-22. Low Cost Point of Sale Terminal

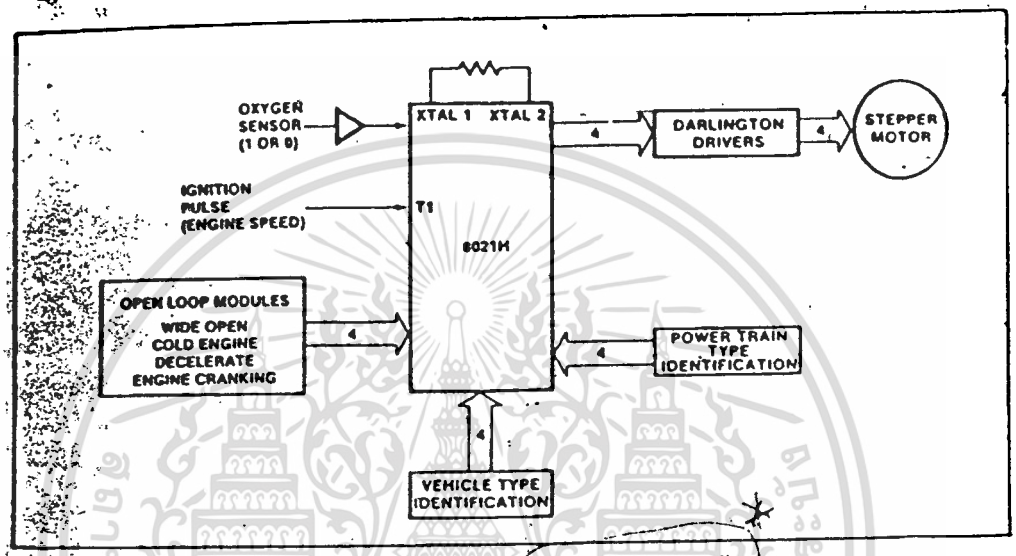


Figure 5-23. Simple Feedback Carburetor Controller

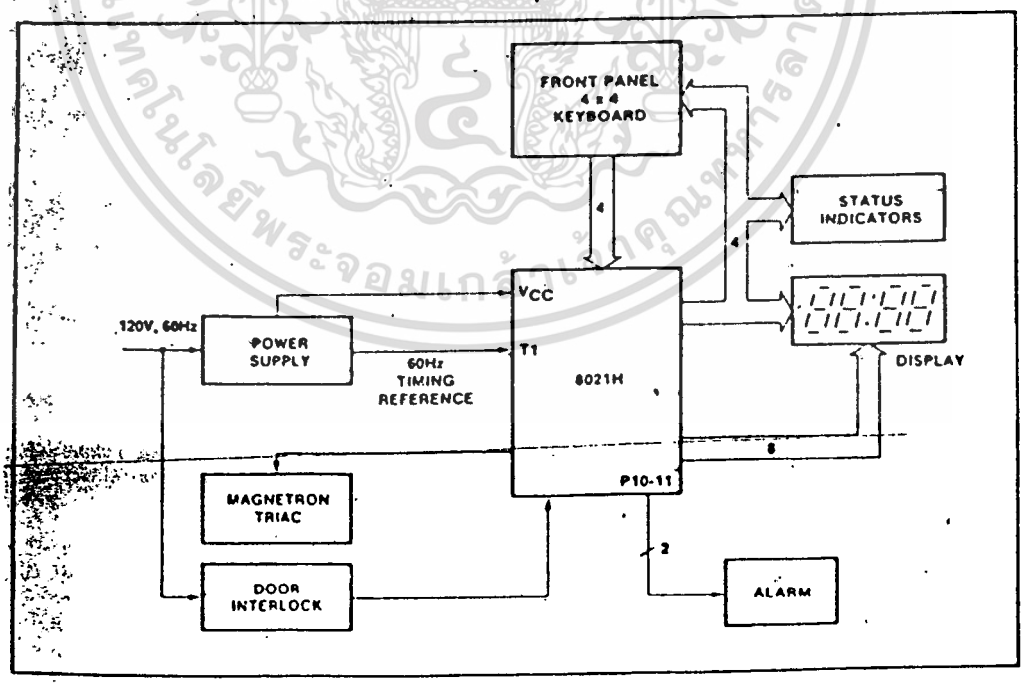
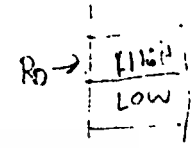


Figure 5-24. Microwave Oven Controller

5.4 SOFTWARE EXAMPLES

The following routines are written as subroutines. R0 and R1 are used as data pointers, R2 is used as an extension of the accumulator and R3 is used as a loop counter.




```
                RX0 = R0        R0 data pointer
                AEX = R2        loop counter
DOUBLE ADD
DADD:   DEC  RX0        ;GET LOW BYTE AND ADD TO A
        ADD  A,@RX0
        INC  RX0        ;GET HI BYTE AND ADD TO AEX
        XCH  A,AEX
        ADDC A,@RX0
        XCH  A,AEX
        RET             ;RETURN
DOUBLE SUBTRACT
DMIN:   DEC  RX0        ;GET LOW BYTE AND SUB FROM A
        CPL  A
        ADD  A,@RX0
        CPL  A
        INC  RX0        ;GET HI BYTE AND SUB FROM AEX
        XCH  A,AEX
        CPL  A
        ADDC A,@RX0
        CPL  A
        XCH  A,AEX
        RET             ;RETURN
DOUBLE LOAD
DLD:    DEC  RX0        ;GET LOW BYTE AND PLACE IN A
        MOV  A,@RX0
        INC  RX0        ;GET HI BYTE AND PLACE IN AEX
        XCH  A,AEX
        MOV  A,@RX0
        XCH  A,AEX
        RET             ;RETURN
DOUBLE STORE
DST:    DEC  RX0        ;MOVE A INTO LOW BYTE
        MOV  @RX0,A
        INC  RX0        ;MOV AEX INTO HIGH BYTE
        XCH  A,AEX
        MOV  @RX0,A
        XCH  A,AEX
        RET             ;RETURN
```

```
DOUBLE EXCHANGE
DEX:    DEC   RX0        ;EXCHANGE A AND LOW BYTE
        XCH   A,@RX0
        INC   RX0        ;EXCHANGE AEX AND HIGH BYTE
        XCH   A,AEX
        XCH   A,@RX0
        XCH   A,AEX
        RET              ;RETURN

DOUBLE LEFT LOGICAL SHIFT
LLSH:   RLC   A          ;SHIFT A
        XCH   A,AEX      ;SHIFT AEX
        RLC   A
        XCH   A,AEX
        RET              ;RETURN

DOUBLE RIGHT LOGICAL SHIFT
RLSH:   XCH   A,AEX      ;SHIFT AEX
        RRC   A
        XCH   A,AEX
        RRC   A          ;SHIFT A
        RET              ;RETURN

DOUBLE RIGHT ARITHMETIC SHIFT
RASH:   CLR   C          ;SET CARRY
        CPL   C
        XCH   A,AEX      ;IF AEX[7]< >1 THEN
        JB7   $+3
        CLR   C          ;CLEAR CARRY
        RRC   A          ;SHIFT C INTO AEX
        XCH   A,AEX
        RRC   A          ;SHIFT A
        RET              ;RETURN
```

5.4.1 Single Precision Binary Multiply

This routine assumes a one-byte multiplier and a one-byte multiplicand. The product, therefore, is two-bytes long.

The algorithm follows these steps:

1) The registers are arranged as follows:
   ACC — 0
   R1 — Multiplier
   R2 — Multiplicand
   R3 — Loop Counter (=8)

   The Accumulator and register R1 are treated as a register pair when they are shifted right (see Step 2) . . .

2) The Accumulator and R1 are shifted right one place, thus the LSB of the multiplier goes into the carry.

3) The multiplicand is added to the accumulator if the carry bit is a 'one'. No action if the carry is a 'zero'.

4) Decrement the loop counter and loop (return to Step 2) until it reaches zero.

5) Shift the result right one last time just before exiting the routine.

*The result will be found in the Accumulator (MS Byte) and R1 (LS Byte).

BINARY MULTIPLY

```
BMPY:   MOV   R3,#08H   ;SET COUNTER TO 8
        CLR   A         ;CLEAR A
        CLR   C         ;CLEAR CARRY BIT
BMPI:   RRC   A         ;DOUBLE SHIFT RIGHT ACC & R1
        XCH   A,R1      ;INTO CARRY
        RRC   A
        XCH   A,R1
        JNC   BMP3      ;IF CARRY = 1 ADD, OTHERWISE DON'T
        ADD   A,R2      ;ADD MULTIPLICAND TO ACCUMULATOR
BMP3:   DJNZ  R3,BMPI   ;DECREMENT COUNTER AND LOOP IF 0
        RRC   A         ;DO A FINAL RIGHT SHIFT AT THE
        XCH   A,R1      ;END OF THE ROUTINE
        RRC   A
        XCH   A,R1
```

5.4.2 Interrupt Handling

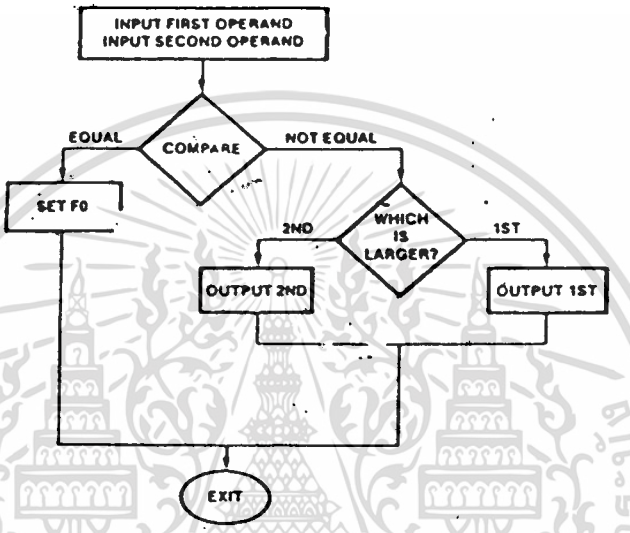
This interrupt routine assumes single level interrupt. The purpose is to store the status of the machine at the time the interrupt occurs by storing contents of all registers, accumulator, and the status word. At the end of the interrupt the state of the machine is restored and interrupts are enabled again.

```
INTRPT: SEL   RB1       ;SAVE WORKING REGISTERS
        MOV   @R0,A     ;R0 IN ALTERNATE REGISTER
                        ;BANK CONTAINS SACC
                        ;POINTER FOR SAVING
                        ;ACCUMULATOR
        |     |
        |     |         INTERRUPT SERVICE
        |     |         ROUTINE
        MOV   R0,SACC   ;RESTORE SACC
        MOV   A,@R0     ;RESTORE ACCUMULATOR
        RETR            ;RESTORE WORKING REGISTERS
                        ;RESTORE PSW AND
                        ;RE-ENABLE INTERRUPTS
```

5.4.3 Two-Byte Processing System

A suggested model of a processing routine takes two single byte inputs from different ports, compares them, and performs the following, depending on the result of the comparison:

(If Equal) Sets Flag and Exits
(If Not Equal) and Outputs the Larger to a Third Port




```
PROCESS:  CLR   F0        ;CLEAR F0 BIT (INITIALIZE)
          IN    A,P1      ;READ FIRST INPUT, STORE IN R0
          MOV   R0,A
          IN    A,P2      ;READ SECOND INPUT, STORE IN R1
          MOV   R1,A
          CPL   A         ;SUBCONTRACT SECOND FROM FIRST
          INC   A         ;(2's COMPLEMENT AND ADD)
          ADD   A,R0
          JZ    EQUL      ;BRANCH IF THEY ARE EQUAL
          JNC   SECOND    ;IF NEGATIVE, SECOND WAS LARGER
          MOV   A,R0      ;ELSE, OUTPUT FIRST
          OUTL  BUS,A
          JMP   DONE      ;EXIT
SECOND:   MOV   A,R1      ;OUTPUT SECOND
          OUTL  BUS,A
          JMP   DONE      ;EXIT
EQUL;     CPL   F0        ;SET F0
          JMP   DONE      ;EXIT
```

8 × 8 MULTIPLY-ASSEMBLED BY MCS®-48 MACRO ASSEMBLER*

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

 LOC  OBJ          LINE          SOURCE STATEMENT

                    1  $INCLUDE(:F1:MPY8)
                    2* 1  $INCLUDE(:F1:MPY8.POL)
                    2* 2  ;*****************************************************************
                    2* 3  ;*                                                               *
                    2* 4  ;*       MPY8X8                                                  *
                    2* 5  ;*                                                               *
                    2* 6  ;*****************************************************************
                    2* 7  ;*                                                               *
                    2* 8  ;*       THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIP'      *
                    2* 9  ;*       AT ENTRY:                                               *
                    2* 10 ;*         A = LOWER EIGHT BITS OF DESTINATION OPERAND            *
                    2* 11 ;*         XA= DON'T CARE                                        *
                    2* 12 ;*         R1= POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMORY *
                    2* 13 ;*                                                               *
                    2* 14 ;*       AT EXIT:                                                *
                    2* 15 ;*         A = LOWER EIGHT BITS OF RESULT                        *
                    2* 16 ;*         XA= UPPER EIGHT BITS OF RESULT                        *
                    2* 17 ;*         C = SET IF OVERFLOW ELSE CLEARED                      *
                    2* 18 ;*                                                               *
                    2* 19 ;*****************************************************************
                    2* 20 ;
                    2* 21 ;
                    2* 22 MPY8X8:
                    2* 23 MULTIPLICAND[15-8]=0
                    2* 24 COUNT:=8
                    2* 25 REPEAT
                    2* 26     IF MULTIPLICAND[8]=0 THEN DEC8
                    2* 27     MULTIPLICAND:=MULTIPLICAND/2
                    2* 28     ELSE
                    2* 29         MULTIPLICAND[15-8]:=MULTIPLICAND[15-8]+MULTIPLIER
                    2* 30         MULTIPLICAND:=MULTIPLICAND/2
                    2* 31     ENDIF
                    2* 32     COUNT:=COUNT-1
                    2* 33 UNTIL COUNT=0
                    2* 34 END MPY8X8
                    1* 35 ;
                    1* 36 ;
                    1* 37 OBJECT
```

8 × 8 MULTIPLY-ASSEMBLED BY MCS®-48 MACRO ASSEMBLER*

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

LOC  OBJ          LINE        SOURCE STATEMENT

                  1= 149 ;1 MPY8X8:
                  1= 150 MPY8X8:
                  1= 151 ;1 MULTIPLICAND[15-8]:=0
0835 B408         1= 152          MOV     RA,#08
                  1= 153 ;1 COUNT:=8
0837 B808         1= 154          MOV     COUNT,#08
                  1= 155 ;1 REPEAT
                  1= 156 MPY8LP:
                  1= 157 ;2     IF MULTIPLICAND[0]=0 THEN BEGIN
0839 1243         1= 158          JB0     MPY8A   ;.
                  1= 159 ;3       MULTIPLICAND:=MULTIPLICAND/2
083B 2A           1= 160          XCH     A,RA
083C 97           1= 161          CLR     C
083D 67           1= 162          RRC     A
083E 2A           1= 163          XCH     A,RA
083F 67           1= 164          RRC     A
0840 E839         1= 165          DJNZ    COUNT,MPY8LP
0842 83           1= 166          RET
                  1= 167 ;2     ELSE
                  1= 168 MPY8A:
                  1= 169 ;3       MULTIPLICAND[15-8] :=MULTIPLICAND[15-8]+MULTIPLIER
0843 2A           1= 170          XCH     A,RA
0844 61           1= 171          ADD     A,PR1
0845 67           1= 172          RRC     A
0846 2A           1= 173          XCH     A,RA
0847 67           1= 174          RRC     A
0848 E839         1= 175          DJNZ    COUNT,MPY8LP
084A 83           1= 176          RET
                  1= 177 ;3       MULTIPLICAND:=MULTIPLICAND/2
                  1= 178 ;2     ENDIF
                  1= 179 ;2     COUNT:=COUNT-1
                  1= 180 ;1   UNTIL COUNT=0
                  1= 181 ;1 END MPY8X8
                  1= 182 $EJECT
```

16× 8 DIVIDE—(ASSEMBLED BY MCS®-48 MACRO ASSEMBLER)*

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

LOC  OBJ          LINE        SOURCE STATEMENT

                    * 183 $INCLUDE(:F1:DIV16)
                   1* 184 ;*****************************************************************
                   1* 185 ;*                                                               *
                   1* 186 ;*      DIV16                                                    *
                   1* 187 ;*                                                               *
                   1* 188 ;*****************************************************************
                   1* 189 ;*
                   1* 190 ;*      THIS UTILITY PROVIDES AN 16 BY 8 UNSIGNED DIVIDE
                   1* 191 ;*      AT ENTRY:
                   1* 192 ;*        A = LOWER EIGHT BITS OF DESTINATION OPERAND .
                   1* 193 ;*        XA= UPPER EIGHT BITS OF DIVIDEND
                   1* 194 ;*        R1= POINTER TO DIVISOR IN INTERNAL MEMORY
                   1* 195 ;*
                   1* 196 ;*      AT EXIT:
                   1* 197 ;*        A = LOWER EIGHT BITS OF RESULT
                   1* 198 ;*        XA= REMAINDER
                   1* 199 ;*        C = SET IF OVERFLOW ELSE CLEARED
                   1* 200 ;*
                   1* 201 ;*****************************************************************
                   1* 202 ;
                   1* 203 ;
                   1* 204 ;1 DIV16:
0040 20            1* 205 DIV16:  XCH     A,XA      ; ROUTINE WORKS MOSTLY WITH BITS 15-8
                   1* 206 ;1 COUNT:=8
004C 0008          1* 207         MOV     COUNT,#8
                   1* 208 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
004E 37            1* 209         CPL     A
004F 61            1* 210         ADD     A,@R1
0050 37            1* 211         CPL     A
                   1* 212 ;1 IF BORROW=0 THEN /* IT FITS*/
0051 F656          1* 213         JC      DIV1A
                   1* 214 ;2    SET OVERFLOW FLAG
0053 A7            1* 215         CPL     C
0054 8456          1* 216         JMP     DIV1B
                   1* 217 ;1 ELSE
                   1* 218 DIV1A:
                   1* 219 ;2    RESTORE DIVIDEND
0056 61            1* 220         ADD     A,@R1
                   1* 221 ;2    REPEAT
                   1* 222 DIV1LP:
                   1* 223 ;4            DIVIDEND:=DIVIDEND*2
                   1* 224 ;4                QUOTIENT:=QUOTIENT*2
0057 97            1* 225         CLR     C
0058 20            1* 226         XCH     A,XA
0059 F7            1* 227         RLC     A
005A 20            1* 228         XCH     A,XA
005B F7            1* 229         RLC     A
005C E663          1* 230         JNC     DIV1C
005E 37            1* 231         CPL     A
005F 61            1* 232         ADD     A,@R1
0060 37            1* 233         CPL     A
0061 8468          1* 234         JMP     DIV1C
                   1* 235 ;4            DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
0063 37            1* 236 DIV1C:  CPL     A
0064 61            1* 237         ADD     A,@R1
```

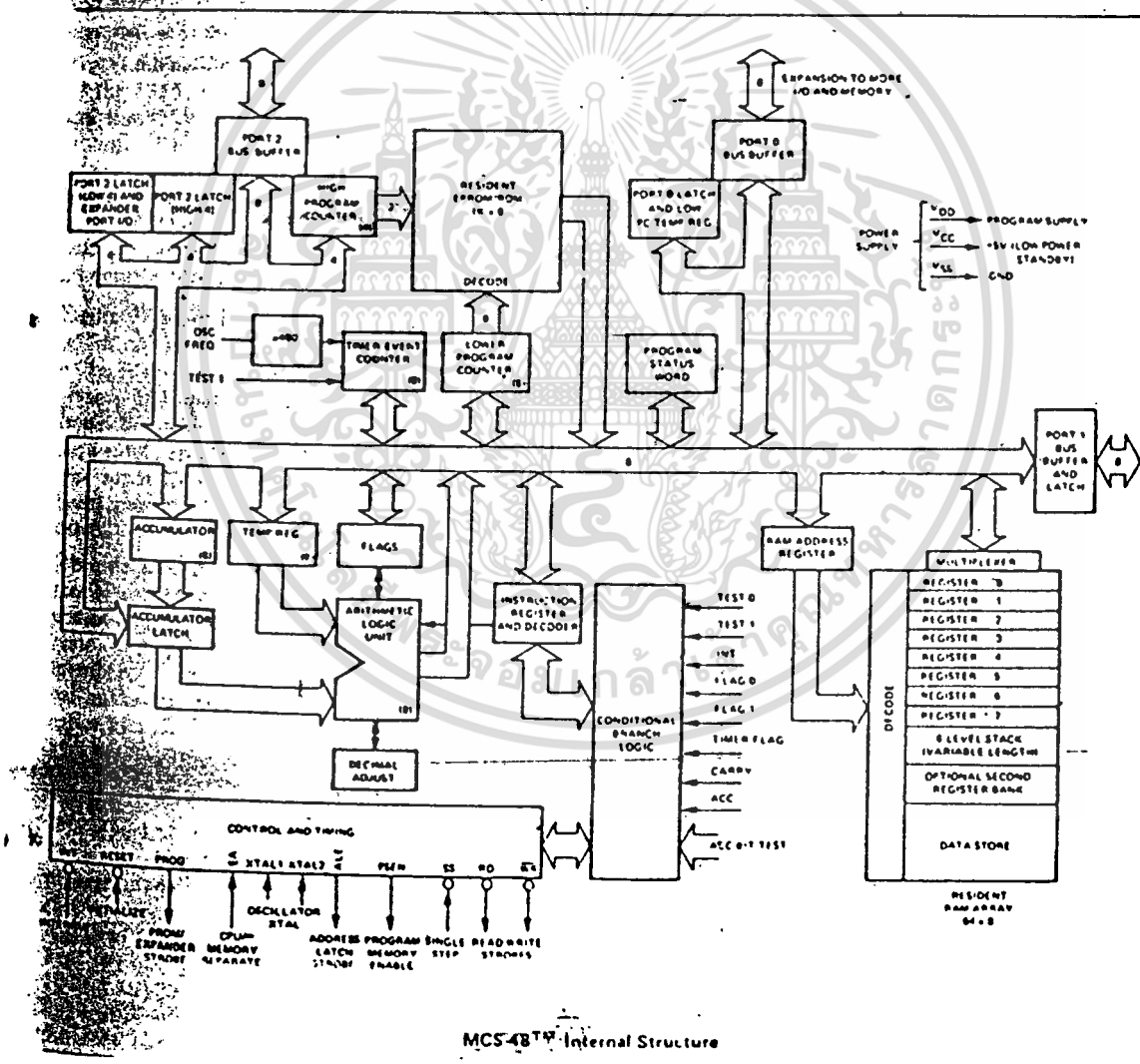
INTRODUCTION

The INTEL® MCS-48™ family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer, and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "subsystems" which are common to many microprocessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



MCS-48™ Internal Structure

INSTRUCTION SET

	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 Movers	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
	ENTO CLK	Enable Clock output on T0	1	1
	NOP	No Operation	1	1

Figure 2. 8048/8748/8035 Instruction Set

operate from a single 5-volt power supply. The 8748 requires an additional 25-volt supply only while the on board EPROM is being programmed. When installed in a system only the 5-volt supply is required. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including many designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit programmable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 family can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any combination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog i; Both of these units were

[ ] Number of Available Timers
( ) Number of Available I/O Lines

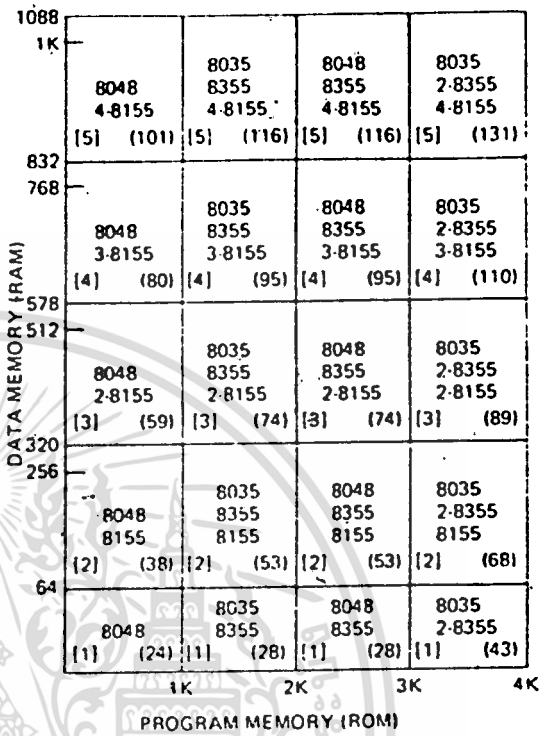


Figure 3. The Expanded MCS-48™ System

specifically designed to interface with microprocessors.

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog converters, two eight bit latches which are loaded from the data bus, and address decoding logic to determine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 with an output impedance of 1Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse ($\overline{WR}$) whenever the proper address is presented to the MP-10. The lower two addresses (A0 and A1) are used internally by the device. Addresses A2 & A3 are compared with the address determination inputs B2 and B3. If their signals are found to be equal, and if addresses A4-A13 are all high, then the device is selected and one of the latches will be loaded. Address bit A1 selects between output 1 and output 2. If address bit A0 is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80H must

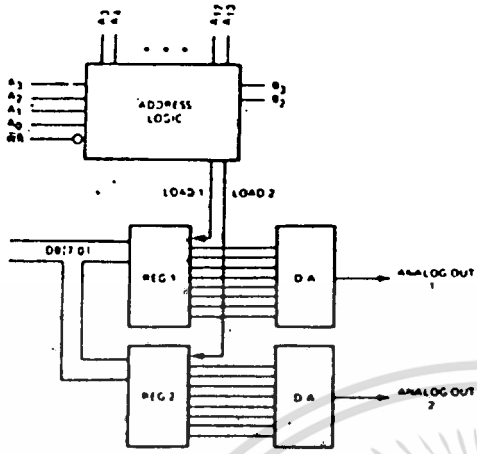


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ±5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

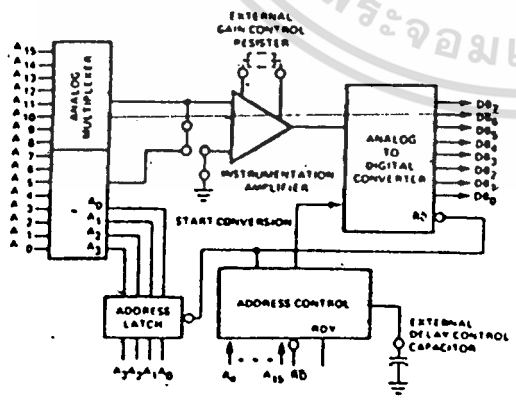


Figure 5. MP-20 Analog Subsystem

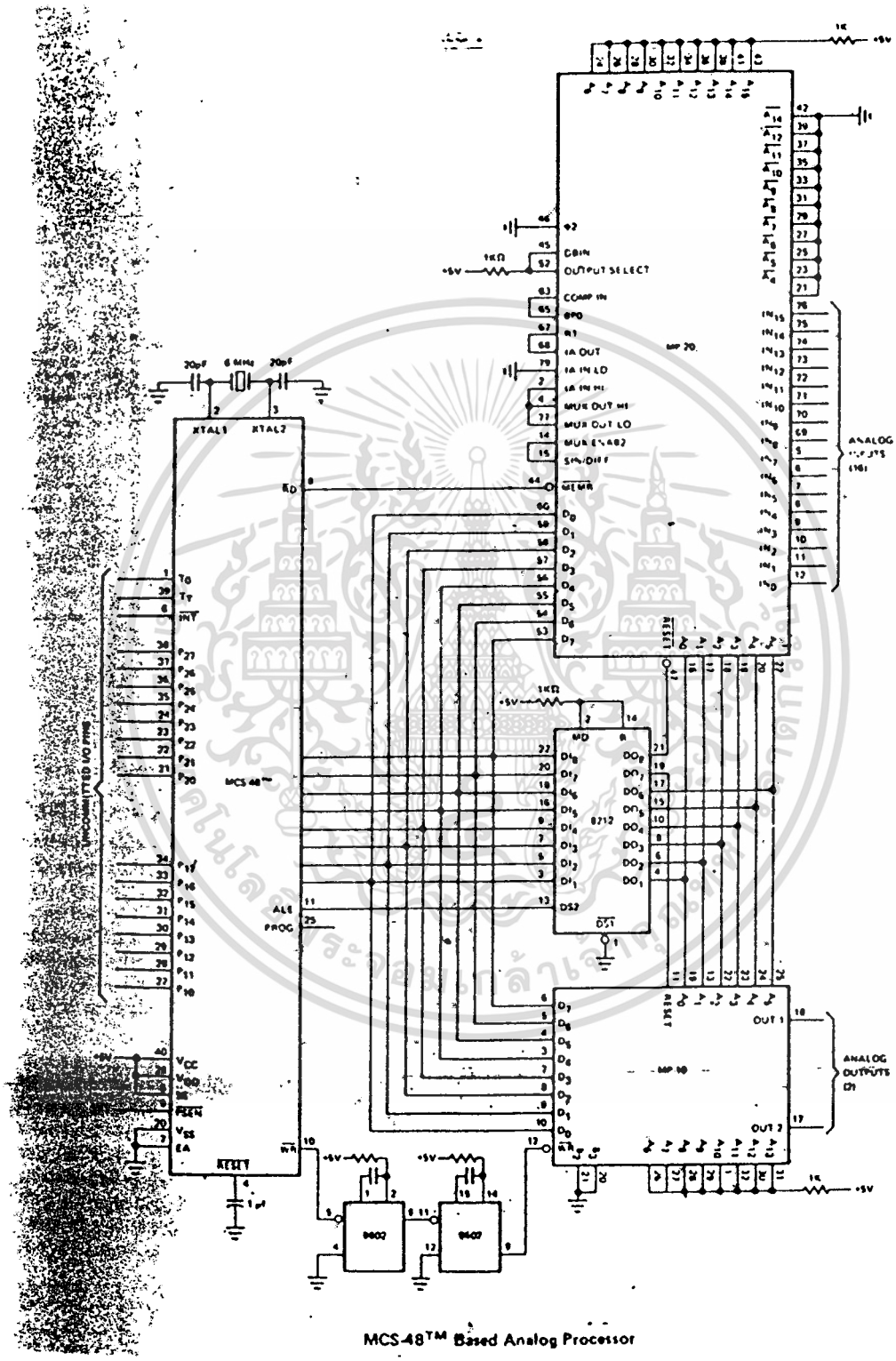
With no R_{ext} ($R_{ext} = \infty$) the gain is two and the input is 0-5 or ±5 volts full scale. Adding an external resistor results in higher gain so that low level (±50mV) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of ±0.4% of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A_{15} through A_4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled $\overline{A_4}$ through $\overline{A_{14}}$. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when $\overline{MEMR}$ pulse occurs, the lower four addresses (A_3-A_0) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and $\overline{MEMR}$ also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80™ system. When a $\overline{MEMR}$ occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 μsec. For R = 510Ω the gain is:

$$G = 2 + \frac{50k\Omega}{.51k\Omega} \cong 100$$

and the conversion time becomes 100μsec. These settling times are specified in the MP-20 data sheet and range from 35 to 475 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the MP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for



MCS-48™ Based Analog Processor

the system designer. The four components provide:

a. An eight bit microprocessor
b. 64 bytes of RAM
c. 1024 bytes of UV erasable PROM
d. A timer/event counter
e. 16 digital I/O pins
f. 2 testable input pins
g. An interrupt capability
h. 16 eight bit analog inputs
i. 2 eight bit analog outputs

The MCS-48 communicates with the D/A and A/D converters in a memory mapped mode (i.e., it treats the devices as if they were external RAM). By setting an address in either R_0 or R_1 and then executing a MOVX the software can transfer data between the accumulator and the analog I/O. When the MCS-48 executes the MOVX instruction it first sends the eight bit address out on the bus and strobes it into the 8212 latch with the ALE (Address Latch Enable) signal. After the address is latched, the MCS-48 uses the same bus to transfer data to or from the accumulator. If data is being sent out (MOVX @Rj, A) the $\overline{WR}$ strobe is used; if the data is being moved into the accumulator (MOVX A, @Rj) the $\overline{RD}$ strobe is used. The one shots on the $\overline{WR}$ line are used to delay the write strobe of the MCS-48 to meet the data set up specifications of the MP-10.

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must be written to following a reset to initialize its internal logic) all channels involve some form of data transfer.

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μsec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

The MP-20 is shown in Figure 6 strapped to input 16 single ended ±5 volts signals. Programs which were used to test this configuration are shown in Figure 7. The first of these programs uses the D/A converter to generate sawtooth waveforms by outputting an incrementing value to the D/A converters. The second program scans the analog inputs and stores their digital values in a table located in RAM.

Table 1. Analog Interface Addresses

INPUT OR OUTPUT	
0XXX XXXX	Reset A/D
X1XX XXXX	Reset D/A
INPUT	
0011 nnnn	Read A/D Channel nnnn
OUTPUT	
1011 0001	Initialize D/A
1011 0000	Write Channel 1
1011 0010	Write Channel 2

```
LOC  OBJ      SEQ     SOURCE STATEMENT

                1
                2 ;-----------------------------
                3 ; TEST PROGRAM FOR ANALOG OUTPUT
                4 ;   THIS PROGRAM OUTPUTS A SAW-
                5 ;   TOOTH WAVEFORM BY OUTPUTING
                6 ;   AN INCREMENTING PATTERN.
                7 ;-----------------------------
                8
                9 ; -------
               10 ; EQUATES
               11 ; -------
               12
0083           13 INITCH:  EQU     083H    ; D/A INITIALIZATION CHANNEL
0080           14 INITDT:  EQU     080H    ; D/A INITIALIZATION DATA
0000           15 DATCH    EQU     000H    ; D/A DATA CHANNEL
               16
               17 ; -----  ------
               18 ; START OF TEST
               19 ; -------------
0100           20          ORG     100H
                                            ; INITIALIZE D/A
0100 2380      21 START:   MOV     A,#INITDT
0102 0883      22          MOV     R0,#INITCH
0104 90        23          MOVX    @R0,A
                                            ; TEST LOOP-OUTPUT SAWTOOTH
0105 8A00      24 LOOP:    MOV     R0,#DATCH
0107 17        25          INC     A
0108 90        26          MOVX    @R0,A
0109 2405      27          JMP     LOOP
                                            ; END OF PROGRAM
               28
               29          END
```

Figure 7a. D/A Exercise Program

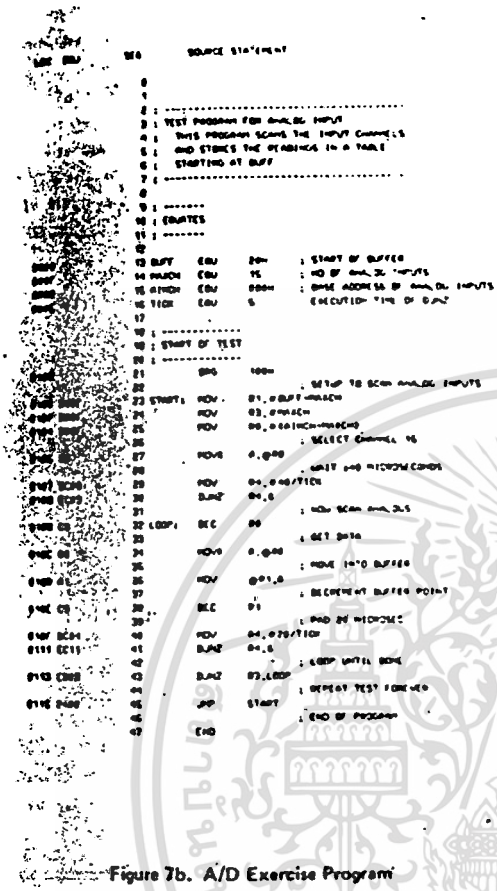


Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of 256 times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\text{Sin } x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots \frac{(-1)^k \times 2k+1}{(2K+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

The above calculation, although conceptually simple, would be time consuming and would severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. the MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last 256 bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOVP3 A, @ A

This instruction uses the initial contents of the accumulator to index into page 3 of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than 256 entries is required, then the table can be located in any page of program memory and the instruction:

MOVP A, @ A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to contain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilized by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

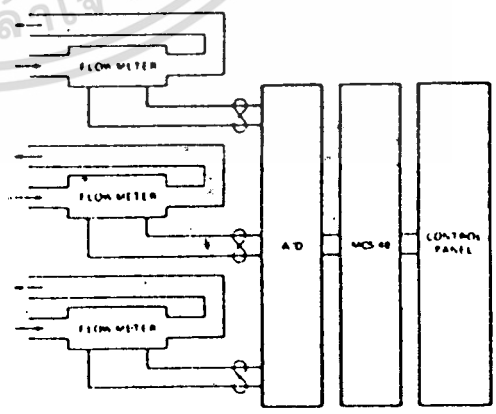


Figure 8. Flow Monitoring System

process consider the hypothetical system shown in Figure 8. The purpose of this system is to measure the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (0H-100H in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from 00H to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

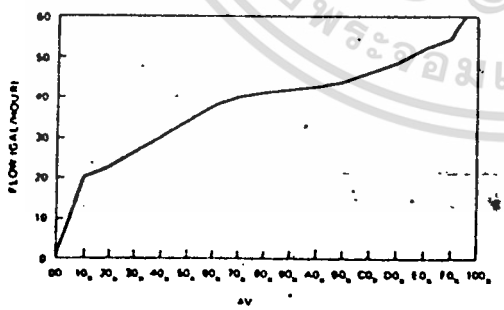


Figure 9. Flow Calibration Curve

TRANSDUCER VOLTAGE (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
MEASURED FLOW (GAL/HOUR)	0	10	27	29	30	34	38	40	41	42	43	45	48	49	53	56	63

Figure 10. Tabulated Flow Data

The eight bits of independent variable (voltage) can be looked on as two four bit fields. The most significant four bits (7-4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16}\,[F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16}\,(34-30) = 32$$

A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE—BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

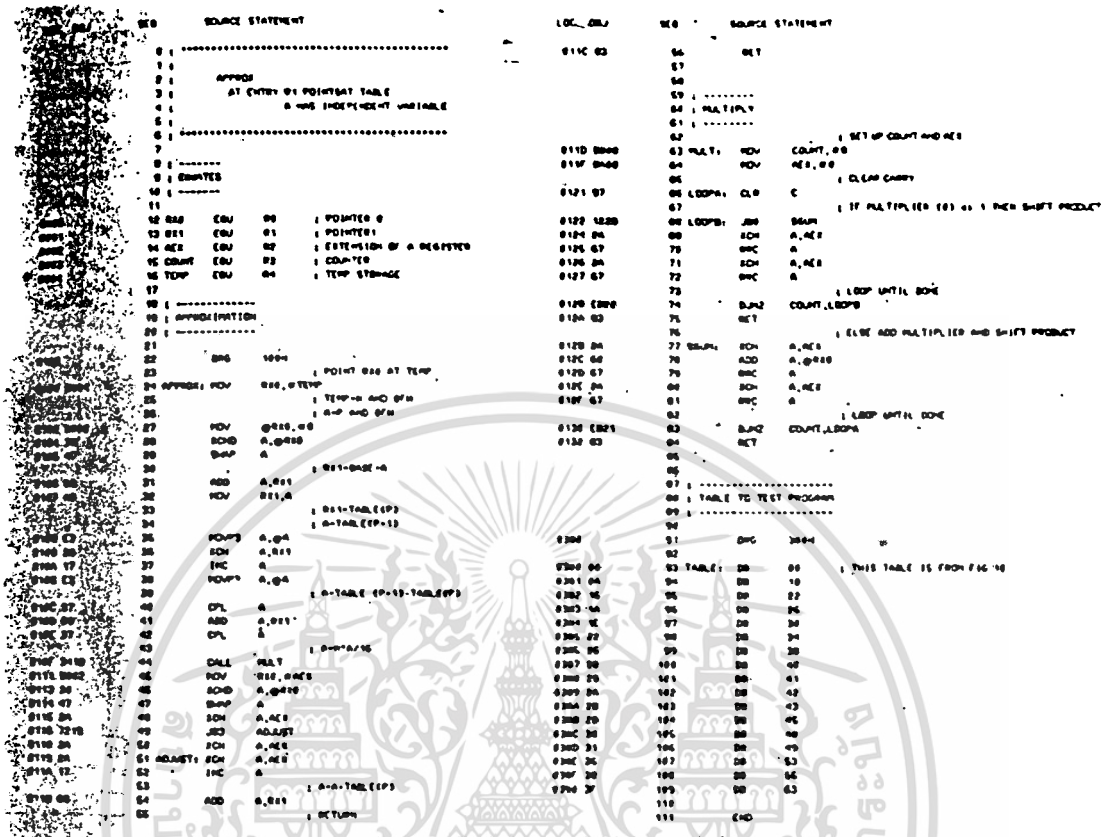


Figure 11. Table Lookup With Interpolation

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial communication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48™ can be brought to bear on the problem of serial communication.

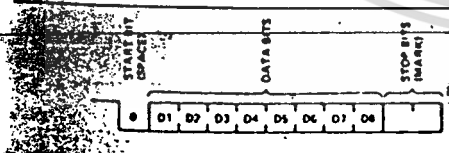


Figure 12. Serial ASCII Code

Probably the most common form of serial communication is that used by the obiquitous Teletype-serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the eighth data bit usually consists of even parity on the remaining seven data bits; for the purposes of this discussion the eighth bit will be considered only as data. A minor variation of this format deletes one of the STOP bits. An algorithm which might be used to sample serial data under software control using a microprocessor is shown in Figure 13. The basic intent of this algorithm is to minimize the effects of distortion and transmission rate variations on the reliability of the communication by sampling each data bit as close to its center as possible. Upon entry to this routine the software first samples the incoming data in a tight loop until it is sensed as a MARK (logical one). As soon as a MARK is detected, a second loop is entered during which the software waits until the received data goes to a SPACE (logical zero). The purpose of this construction is to detect as accurately as possible the leading edge of the START bit. This instant of time will be used as a reference point for sampling all of the following bits in the character. After sensing the leading edge of the START bit a wait of one half the expected bit time is implemented. The period of the incoming signal is called P for convenience. At the end of this wait the serial line is tested—if it is MARK then the START bit was

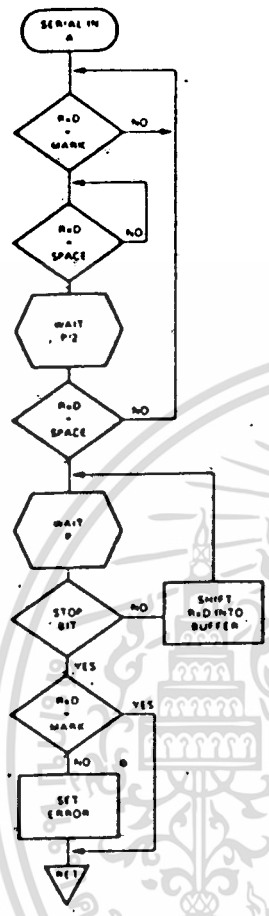


Figure 13. Sample Serial Input Routine

invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype the processor will spend only a 100μsecs or so processing data and the rest of the 100 millisecs waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in microprocessor systems.

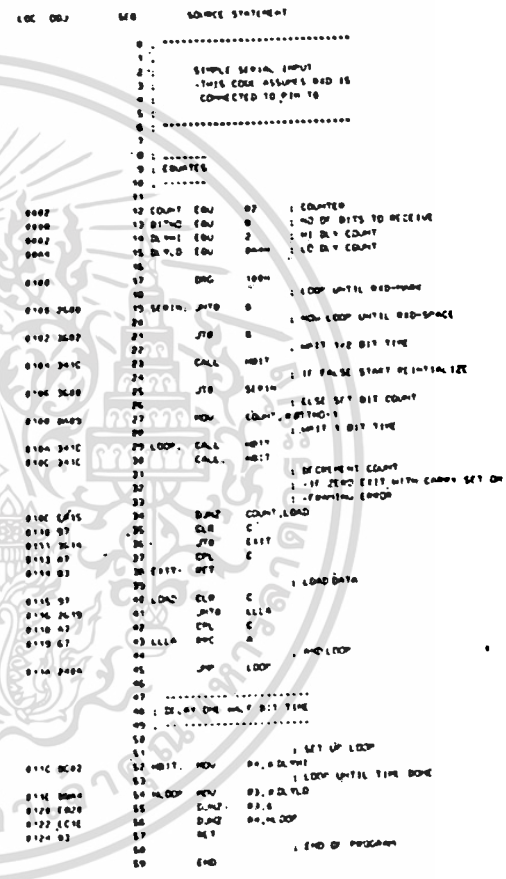


Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initilization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal se ncing by the USART) is provided by con-

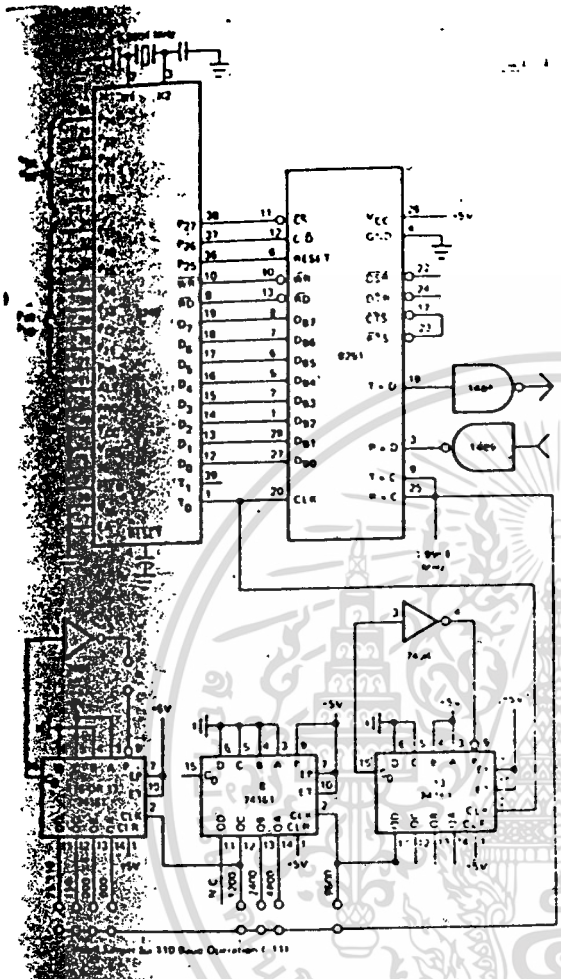


Figure 15. MCS-48™ to 8251 Interface

esting the CLK signal of the USART to the T0 pin of the MCS-48. The T0 pin of the MCS-48 can either be used as a directly testable input pin or it can become, under program control, an output pin which oscillates at one third of the crystal frequency. (Note that once this pin is designated by the software to be an output it will remain so until the system is reset.) In Figure 15 the crystal frequency is 5.9904 MHz so the clock provided to the 8251 is 1.9968 MHz, which conforms to its specifications.

The initialization signal to the USART (RESET) is provided programmatically by manipulation of bit 6 of Port 2. It was necessary to place the reset of the 8251 under program control for two reasons. The first reason is that the MCS-48 does not supply a RESET signal to other devices. The reason for this is that it was felt to be more useful to provide another pin of I/O function instead of a RESET OUT signal from the MCS-48. Although this situation could have been circumvented by the use of an externally generated reset which drove both the MCS-48 and the 8251, the second reason for program control of the reset to the USART still stands. The USART requires the presence of the CLK signal during reset in order to properly initialize itself. The ENT0 CLK instruction which the MCS-48 must execute before the 8251 will receive the CLK can obviously not be executed until after the system reset has ended. Reset of the USART can be accomplished by the following code segment:

```
        ENT0    CLK             ;TURN ON CLOCK
        ORL     P2, #00100000B  ;START RESET
        MOV     R2, #DELAY      ;DELAY USART
LOOP:   DJNZ    R2, LOOP        ;RESET TIME
        ANL     P2, #11011111B  ;END RESET
```

This code first enables the clock, then asserts the reset signal of a time period determined by the constant DELAY. The delay invoked is (10 + 5*DELAY) microseconds for DELAY >0. The USART requires a reset of approximately 6 CLK periods so DELAY is chosen to be 1 which ensures adequate reset timing. Note that for delays this short, NOP instructions could also be used to time the pulse.

The data clocks required by the USART are provided by the modem if the USART is operated in the synchronous mode. In the more common asynchronous mode, however, these clocks must be provided by circuitry associated with the 8251.

The 5.9904 MHz crystal was chosen because the resulting 1.9968 MHz clock to the USART can be evenly divided to provide transmit and receive clocks to the USART. Assuming the USART is in the x16 mode (i.e. it requires data clocks 16 times the baud rate) the 1.9968 MHz signal can be divided by 13 to generate the proper clock rate for 9600 baud operation. This 9600 baud clock can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud required by Teletypes.

The MCS-48 communicates with the 8251 in a memory mapped mode (i.e. as if the 8251 were external RAM). The instructions available to do this are MOVX @Rj, A which stores the contents of the accumulator at the external RAM location addressed by Rj (j=0 or 1), and its complement, the MOVX A,@ Rj instruction which moves data from the external RAM into the accumulator. Since the MCS-48 multiplexes addresses and data on the same eight bit bus an external latch would be required in order to address the USART with

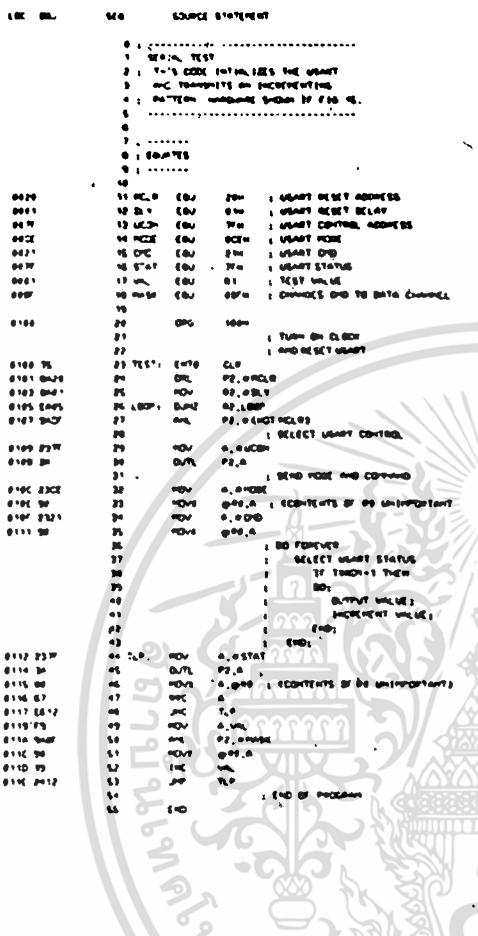


Figure 16. 8251 Test Program

R0 or R1. In order to minimize the circuitry in Figure 15 an approach utilizing some of the I/O pins of the MCS-48 to address the 8251 was chosen instead. By connecting the chip select ($\overline{CS}$) input of the 8251 to bit 7 of port 2 (P27) and similarly connect the C/$\overline{D}$ address line of the 8251 to bit 6 of port 2 (P26) it is possible to address the 8251 without using R0 or R1. The instruction sequence to access the 8251 is to first reset P27 and set P26 to the appropriate state, use a MOVX instruction to perform the appropriate operation, and then finally set P27 to deselect the 8251. As a concrete example of this addressing, Figure 16 shows the code necessary to initialize the 8251 and output an incrementing test pattern on a status driven basis.

If more than one 8251 were to be added to the MCS-48, or if other types of peripheral circuitry would be required (e.g. an 8253 timer to generate the data clocks) it would probably become desirable to add the circuitry necessary to use R0 or R1 to address the peripheral devices. The circuitry which has to be added to Figure 15 in order to make use of R0 or R1 to address the USART is shown in Figure 17. Note that only the changes to Figure 15 are shown. The additional component required is the 8212 eight bit latch. This latch is loaded, whenever a valid address is on the bus by the Address Latch Enable (ALE) signal provided by the MCS-48. During an external read or write cycle this address is used to address the 8251 in a linear select mode. In the circuit shown, the 8251 will be selected by any address with bit 1 a logical zero (XXXXXX0X) and the selection of control or data transfer (C/$\overline{D}$) will be based on bit zero of the address obtained from R0 or R1. Figure 18 shows the program of Figure 16 modified to utilize the addressing inherent in the MOVX instructions.

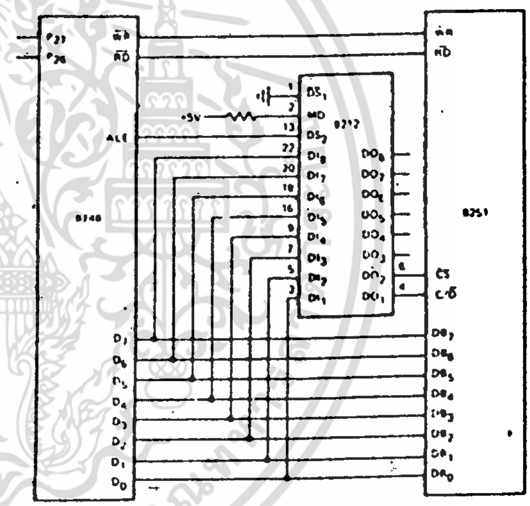


Figure 17. Modified MCS-48 to 8251 Interface

RECEIVING SERIAL CODE—A MORE SOPHISTICATED ALGORITHM

Although the USART does an admirable job of performing the serial I/O function for the MCS-48™, there are some situations where it can not be used. These situations may be caused by economic factors, such as an extremely cost sensitive design, or because the code which must be utilized cannot be accommodated by the USART. An example of of such a code will be discussed later. Recall that the principal objection to the approach to serial input shown in Figure 13 was that it consumes much of the processor's power by merely spinning in loops in order to wait preset time delays.

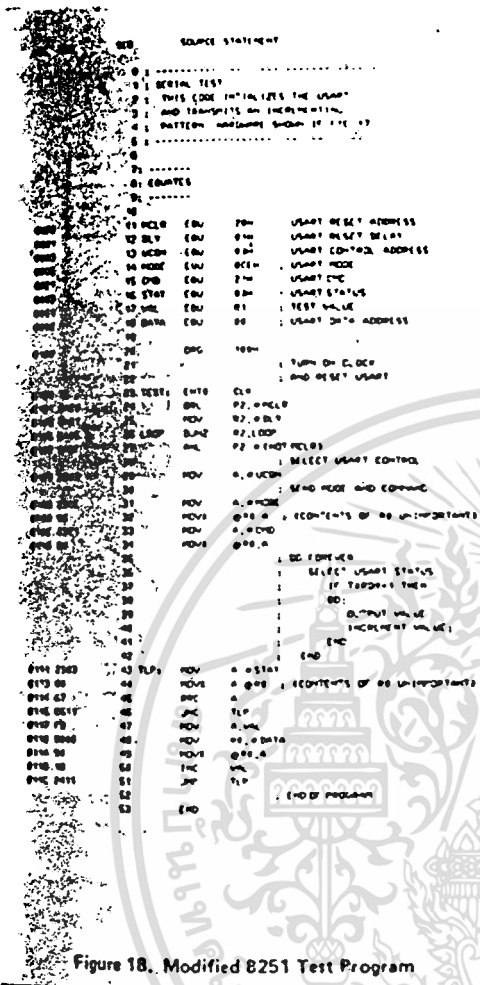


Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion in shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the

interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

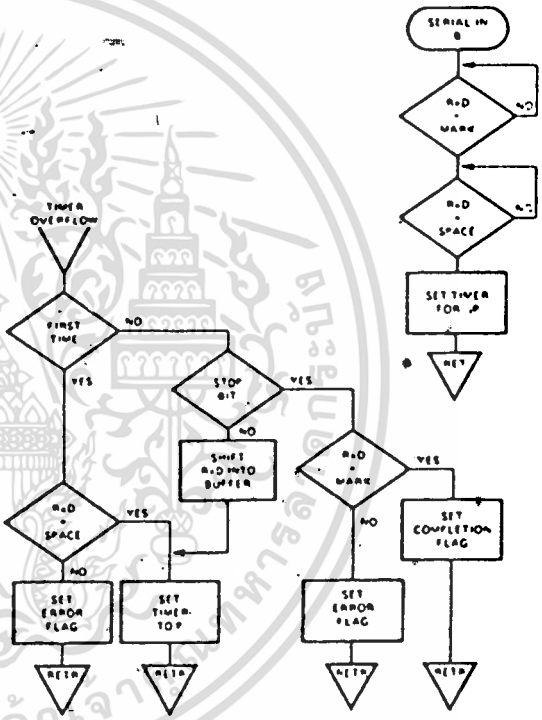


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$Emax = \frac{0.5 * BIT\ TIME}{CHARACTER\ TIME} - \frac{0.5P}{11P} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$Emax = \frac{0.5P}{32P} = 1.6\%$$

Since this calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsecs with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (RxD). A hardware construct which allows this is shown in Figure 20.

The received data (RxD) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either RxD or $\overline{RxD}$. (If P27 = 1 then T1 = RxD; if P27 = 0 then T1 = $\overline{RxD}$.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and RxD is given by the Boolean expression:

$$\overline{T1} = P27 \cdot \overline{RxD} + \overline{P27} \cdot RxD$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow $\overline{RxD}$. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the RxD goes from the MARKING to SPACING state (i.e. the START

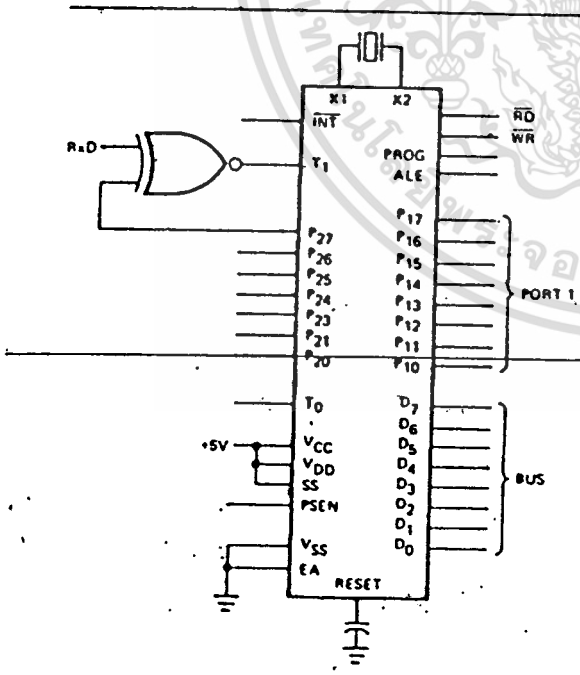


Figure 20. Detecting RxD Edges

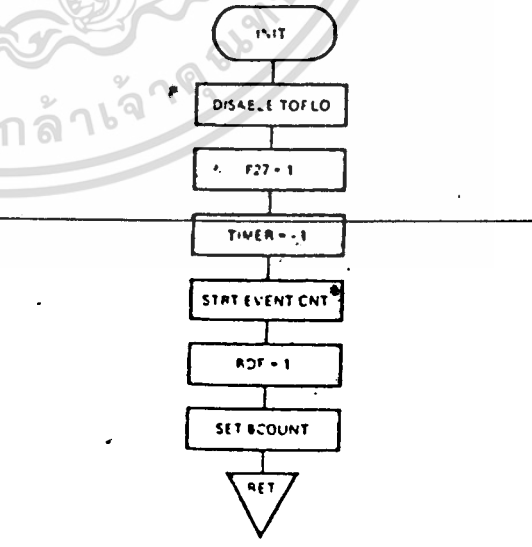


Figure 21a. Interrupt Driven Serial Receive I art

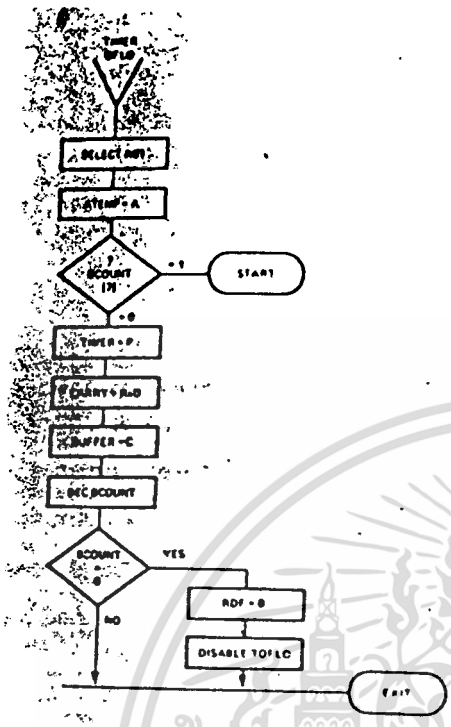


Figure 21b. Interrupt Driven Serial Receive Flowchart

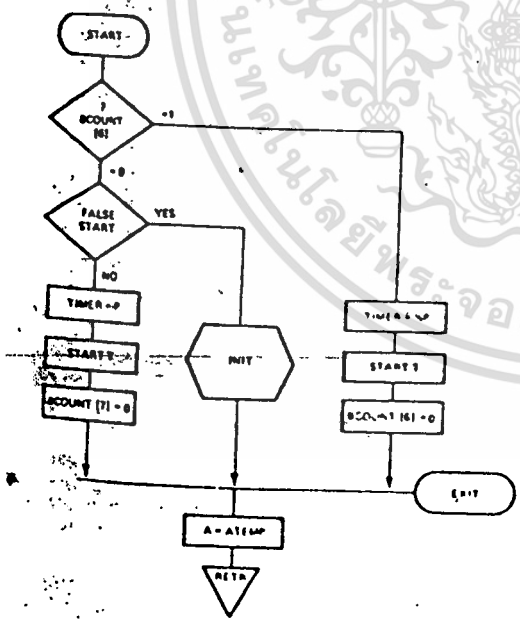
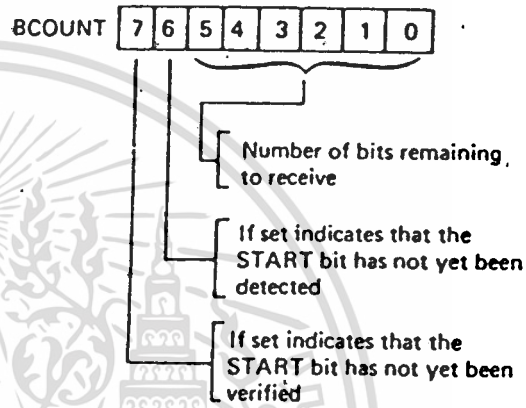


Figure 21c. Interrupt Driven Serial Receive Flowchart

bit occurs). By setting the timer to 0FFH and enabling it in the event count mode, the INIT routine sets up the MCS-48 to generate a timer overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:



In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 11001011B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the $\overline{INT}$ pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from 0FFH to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

events occur simultaneously the external interrupt will take precedence. The CALL automatically saves the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RB1) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (R0 and R1). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implemented by locking out new interrupts between the time an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of RxD after INIT is called), control is passed (by the hardware generated CALL) to the point labled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a RETR is executed. Note that since the state of the flip flop which selects RB1 is saved as part of the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time (1/2 P), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

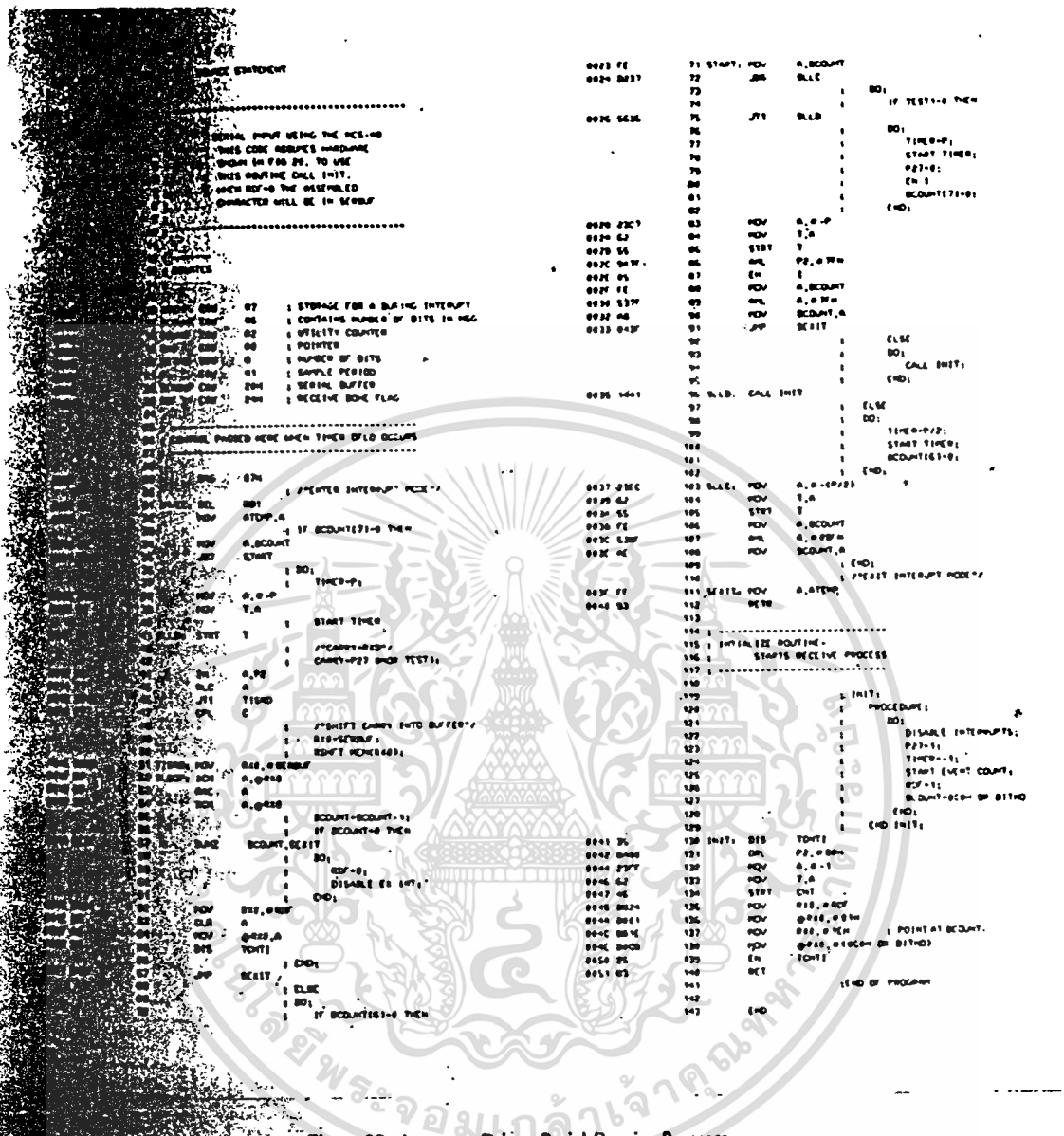


Figure 22. Interrupt Driven Serial Receive Program

code, such as those used in synchronous data communications (e.g. BISYNC or SDLC). BCH codes, named for their originators Bose, Chaudhuri, and Hocquenghem, are characterized by having a length of 2^m-1. The number of redundant check bits can be any integer t where t is a positive integer (clearly mt < m). The 31,26 code fits this format with m=5 and t=1. The length of each message is $n=2^5-1=31$ bits, of which 5 are redundant bits, leaving 26 bits available for data transmission. With an appropriate polynominal BCH codes can detect all errors consisting of 2t error bits and all burst errors of mt or fewer bits. The 31,26 BCH code will therefore detect any erroneous messages with 1 or 2 errors or bursts of errors of less than 5 bits. The 31,26 format (shown in Figure 23) requires the reception of a start bit followed by 31 information bits, clearly beyond the capability of the USART but perhaps within reach of a program controlled approach using the MCS-48 itself.

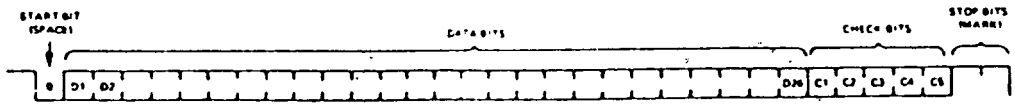


Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; at this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme does seem to work, the question of what additional information is needed remains.

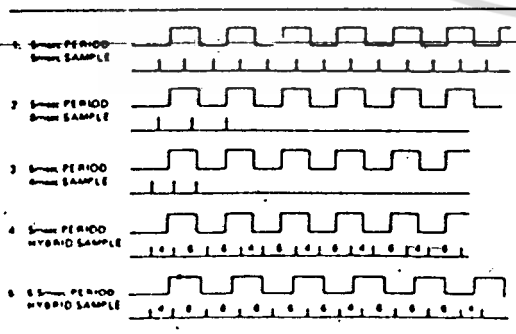


Figure 24. Various Sampling Alternatives

The MSC-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithum did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

If no edge occurs on the RxD line during a sample, then change sampling period from short to long or vice versa.

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive – NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the INT input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

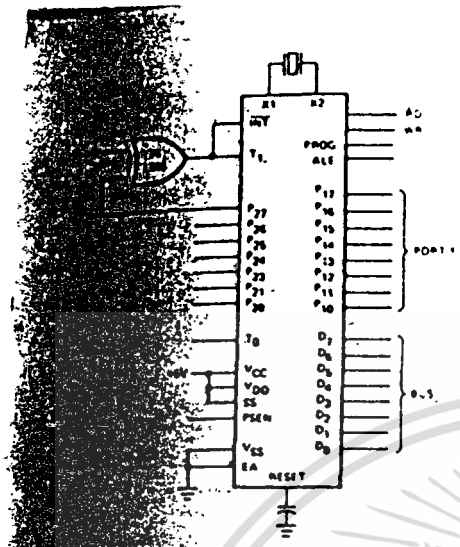
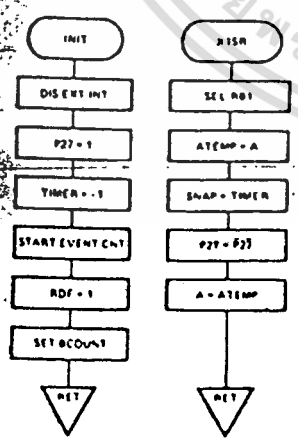


Figure 25. Modified Edge Detection

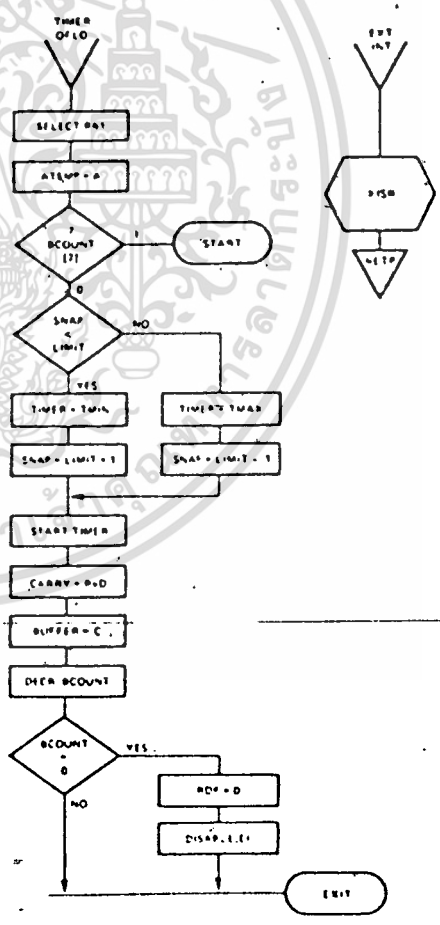
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a) which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RB1). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (for another edge).



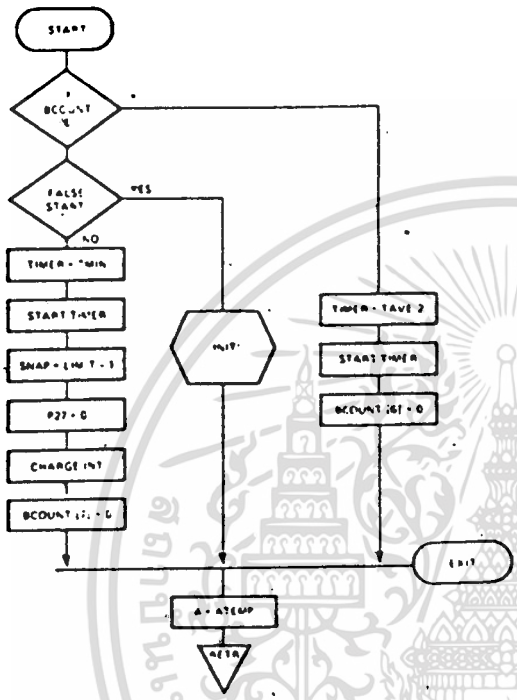
Hybrid Sampling Flowchart

Because of this edge detection it is important to condition RxD with hardware filters to ensure that the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for

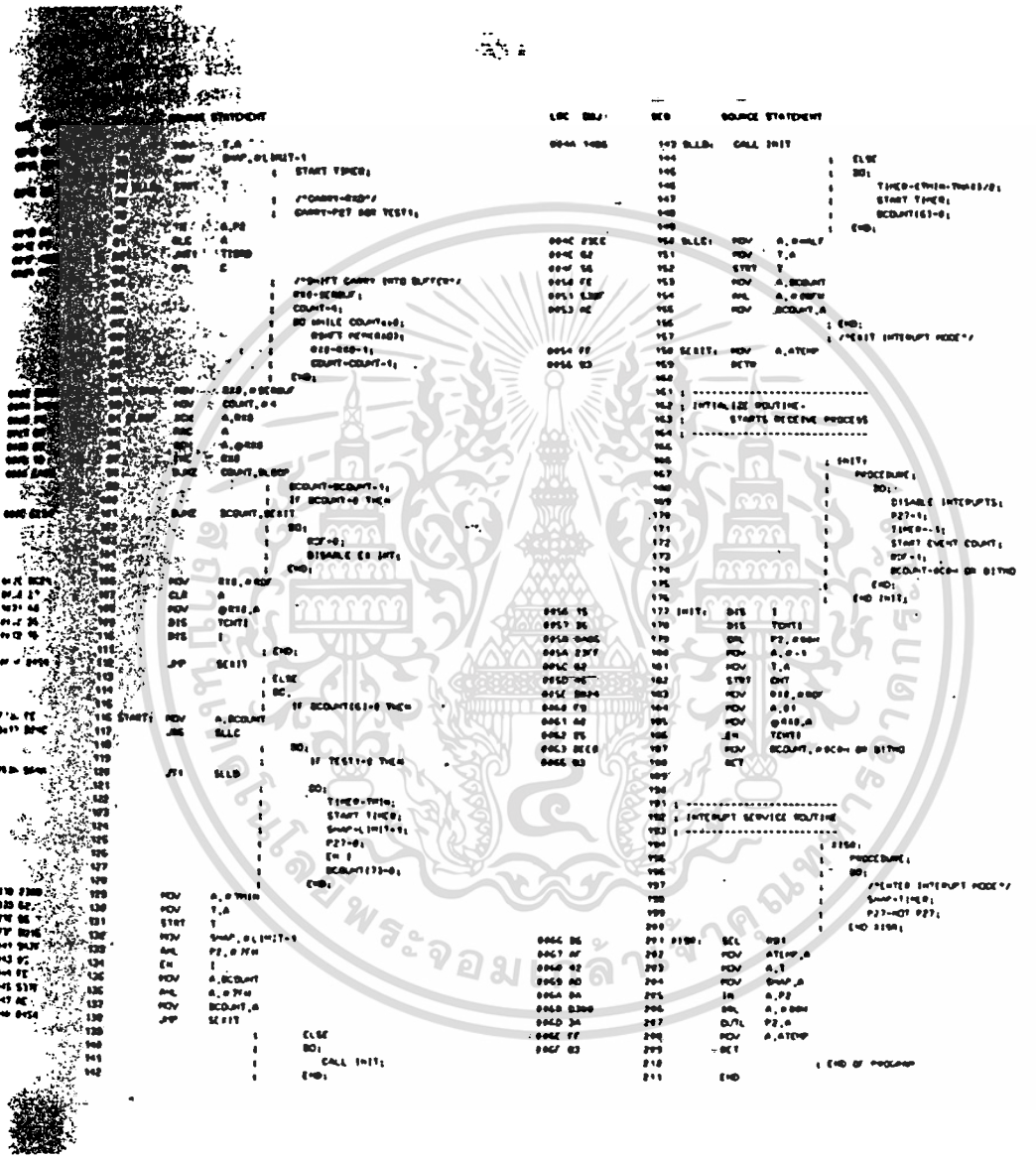


Hybrid Sampling Flowchart



Hybrid Sampling Flowchart

TMAX. By presetting SNAP in the manner shown in the flowcharts the second rule of the algorithm, (*i) no edge appears on the RxD line during a sample, then change the sampling periods short to long or vice versa*) is automatically met. If an edge occurs then XISR will modify SNAP, if XISR is not invoked between two samples then the choice of timer periods will alternate. The only other significant change to the algorithm is that the INIT routine must now lock out all interrupts, not just the timer overflow interrupt, while it is operating. A program which uses this algorithm to receive a 32-bit message is shown in Figure 27.



TRANSMITTING SERIAL CODE

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

```
LOC  OBJ        SEQ     SOURCE STATEMENT

                  1 ;-------------------------------
                  2 ; SERIAL TRANSMIT ON THE 8051H
                  3 ;   TO USE PUT A CHAR IN BUFF AND
                  4 ;   SET Cmdrdy TO 0FFH, WHEN THE
                  5 ;   TRANSMITTED IS READY FOR ANOTHER
                  6 ;   CHAR IT WILL CLEAR Cmdrdy TO
                  7 ;   TRANSMISSION IS DOUBLE BUFFERED
                  8 ;-------------------------------
                  9
                 10 ; --------
                 11 ; EQUATES
                 12 ; --------
                 13
0007             14 ATEMP    EQU     07      ; STORAGE FOR A DURING INT.
0006             15 PTDS     EQU     06      ; PARALLEL TO SERIAL CONVERTER
0005             16 BUFF     EQU     05      ; CONVERTED BUFFER
0004             17 Cmdrdy   EQU     04      ; CHARACTER AVAILABLE FLAG
0003             18 COUNT    EQU     03      ; BIT COUNTER
0BFF             19 CBIT     EQU     0BFH    ; MASK TO CLEAR TXD IN P3
0040             20 SBIT     EQU     040H    ; MASK TO SET TXD IN P3
FFD7             21 P        EQU     -41     ; PERIOD OF TXD
                 22
                 23 ;-------------------------------
                 24 ; CONTROL PASSED HERE ON TIMER OVERFLOW
                 25 ;-------------------------------
0007             26          DPC     07H
                 27                          ; ENTER INTERRUPT MODE
0007 05          28 TOFLO    SEL     RB1
0008 AF          29          MOV     ATEMP,A
                 30                          ; SET TIMER FOR P
0009 23D7        31          MOV     A,-P
000B A2          32          MOV     T,A
000C 55          33          STRT    T
                 34                          ; SET BIT TIMER COUNT
000D 1410        35          CALL    BIT
                 36                          ; SET TXD TO Cmdrdy

LOC  OBJ        SEQ     SOURCE STATEMENT

000F 08          37          IN      A,P2
0010 D3D0        38          XRL     A,#0D0H
0012 3A          39          OUTL    P2,A
0013 FC19        40          JC      BITOK
0015 B4E7        41          ANL     P2,#0E7H
0017 0418        42          JMP     EXIT
0019 0A10        43 BITOK:   ORL     P2,#SBIT
001B FF          44 EXIT:    MOV     A,ATEMP
001C 93          45          RETR
                 46
                 47 ;-------------------------------
                 48 ; BIT ROUTINE
                 49 ; PICKS THE NEXT BIT TO TRANSMIT
                 50 ;-------------------------------
                 51
001D F8          52 BIT:     MOV     A,COUNT
001E C629        53          JZ      IDLE
0020 FE          54          MOV     A,PTDS
0021 67          55          RRC     A
0022 A3D0        56          ORL     A,#0D0H
0024 AE          57          MOV     PTDS,A
0025 C8          58          DEC     COUNT
0026 83          59          RET
                 60
0027 97          61 IDLE:    CLR     C
0028 FC          62          MOV     A,Cmdrdy
0029 B62B        63          JNZ     GOTONE
002B A7          64          CPL     C
002C 83          65          RET
                 66
002D F8          67 GOTONE:  MOV     A,BUFF
002E AE          68          MOV     PTDS,A
002F B804        69          MOV     COUNT,#4
0031 BC00        70          MOV     Cmdrdy,#0
0033 83          71          RET
                 72
                 73          END                 ; END OF PROGRAM
```